Università
della
Svizzera
italiana

Faculty
of Informatics

June 27, 2018

Water Me!

Physical Computing Final Project - 2018

Neha Tharani, Sumeet Gyanchandani, Hrishikesh Gupta, Martin Dodevski

---

Contents

1 Introduction **1**
1.1 IoT Idea Generator: 1
1.2 Project Motivation: 2

2 Project and Prototype Description: **3**

3 Components and Architecture: **4**
3.1 Hardware components: 4
3.2 3D Printing: 5
3.3 Software Components: 6

4 Final Results: **7**

5 Reflection and Discussion: **8**

6 Appendix A - Contemporaries **9**

7 Appendix B - Arduino code **10**

8 Appendix C - Mobile Application Code **12**

9 Appendix D - Project Links **18**

1 Introduction

1.1 IoT Idea Generator:

The Tiles IoT cards were used to brainstorm the ideas for the final project and three ideas were finalised. After a thorough research and discussions, the final project was chosen. We named the final project as "Water Me!" and it is based on theme of home-automation or smart homes. The images below depict the Idea Generation process:

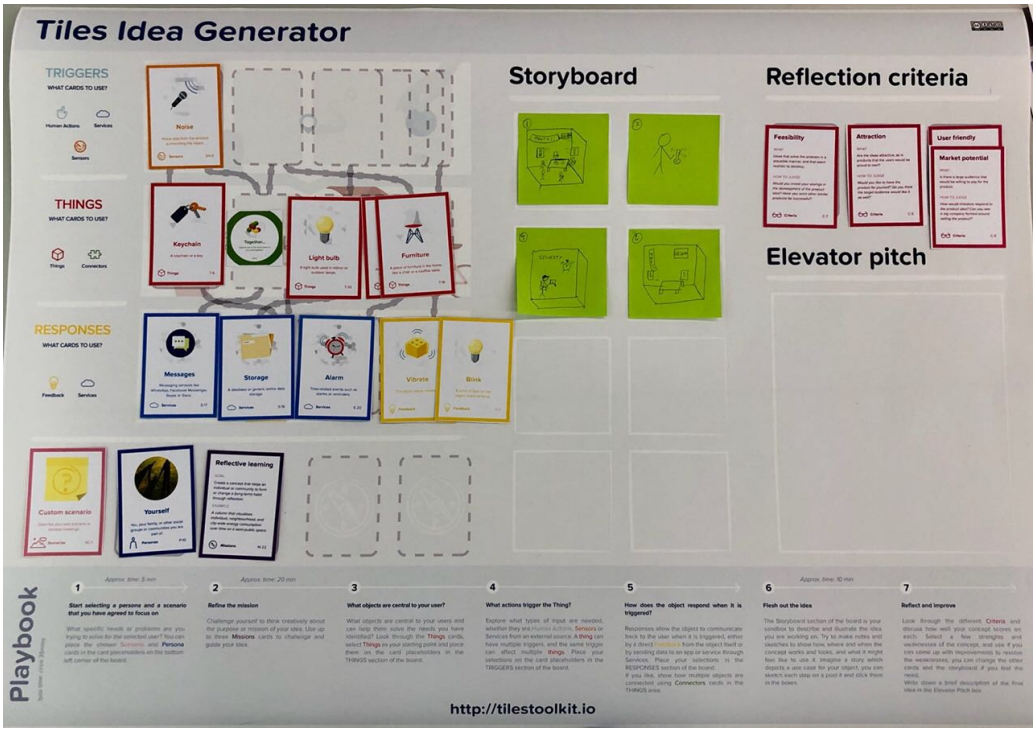

Figure 1. Idea Generation of Noise Alert

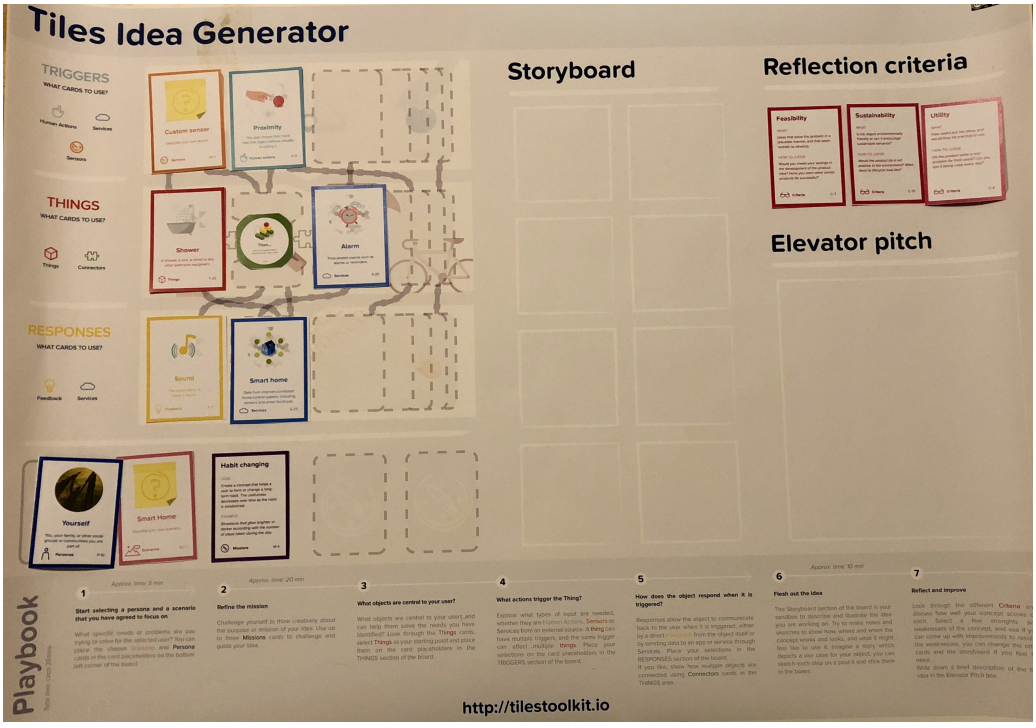

Figure 2. Idea Generation of Smart Shower

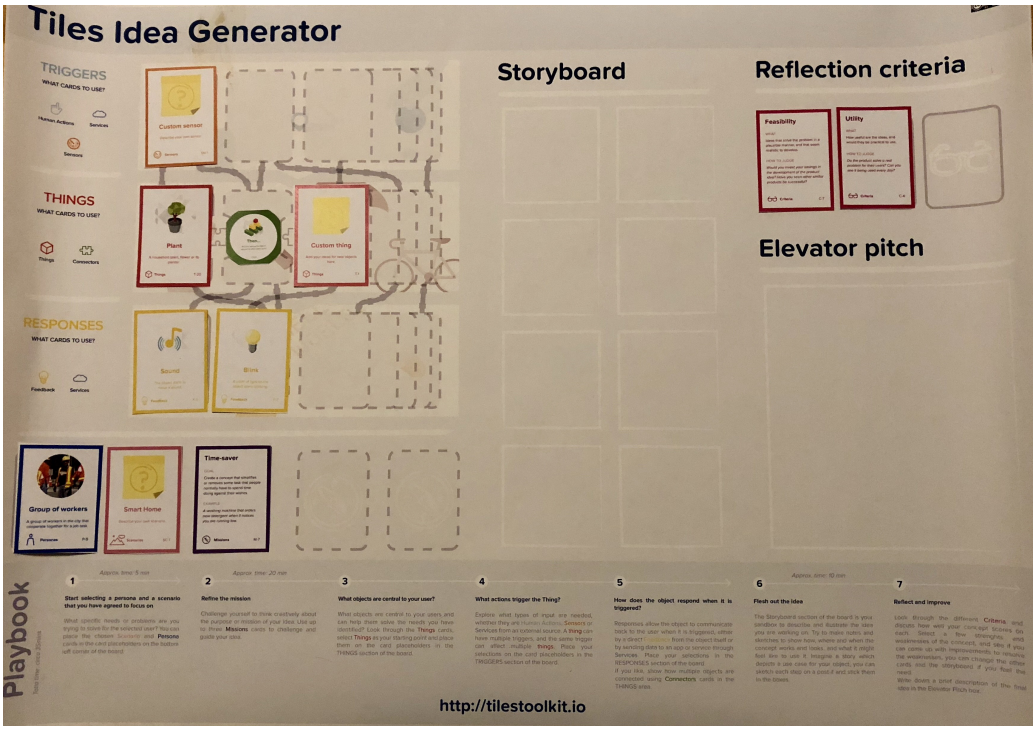

Figure 3. Idea Generation of Water Me

1.2 Project Motivation:

The main idea behind Water me was to create an intelligent plant watering and monitoring device. This device will not only water plants but also help in keeping track of their health and current status. This device can be submerged into the soil of a plant and it will show the current moisture level of the soil, the amount of sunlight that the plant is receiving and the room temperature. The motivation was to use this product on a domestic level to water the plants. This device is ideal for the people who are worried about the plant whilst they are traveling. This is also well suited for the old people or someone who loves his plants but often forget to water them. The product is made very simple to use and the set up is not very technical so that anyone can use it.

2 Project and Prototype Description:

Three important dimensions of design thinking were considered while building the prototype, i.e., desirability, feasibility, and viability. The product is desirable to very busy or lazy people or someone who travels a lot. The product can be manufactured at a very low cost and can be used on domestic level, preserving the viability of the product.

The basic idea behind the project was to build a moisture sensing device that could also sense the temperature and sunlight for the plants. The obvious first step was to search for the demand and similar products. After the research, it was found that many companies are investing in such automatic watering device and are finding a way to improve it every day. This has a lot of applications in the field of automatic irrigation as well.

The project that we made is a very basic prototype of what can be done on a domestic level. We have also mentioned some worthy contemporaries of the project in Appendix A, but all of them are highly expensive. A major success of our project is that an enthusiast can download the code, the circuit design and the 3D printing files from our GitHub page and can make this project easily, with very little cost.

Working

The basic working of the prototype involves, plugging the soil moisture sensor into the soil. There is an LCD screen that tells you the exact values of soil moisture level, temperature, and sunlight. To increase the user experience, even more, an app was developed which will show all the statistics of the plants on the phone. This way you can monitor your plant when you're away. Moreover, the user can then initiate the command like pump off and pump on to water the plants.

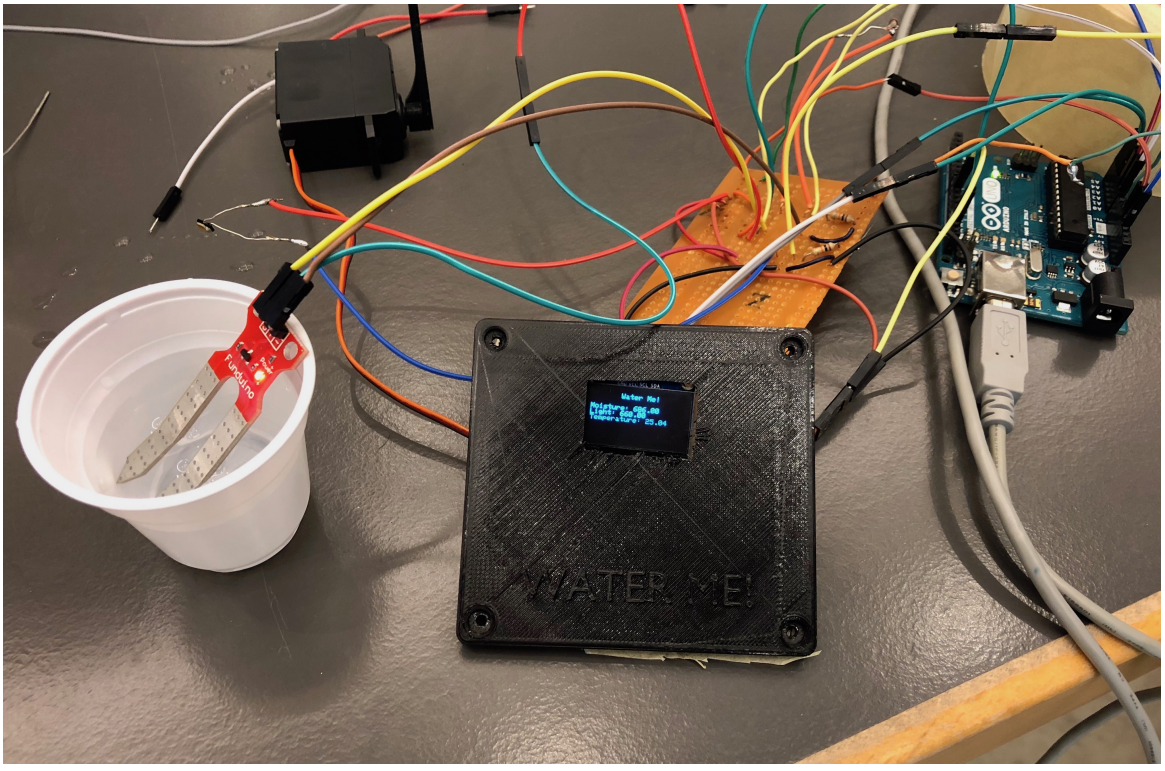

Figure 4. The prototype

3 Components and Architecture:

3.1 Hardware components:

Circuit design:

1. Arduino Uno
2. 10K ohms resistor x 2
3. 220 ohms resistor
4. LED
5. OLED LCD l2c1
6. Sparkfun Soil Moisture Sensor
7. WiFi Module ESP-12F1
8. Temperature Sensor
9. Light Sensor

To assemble the project, first sensors were tested locally. Then the components were assembled as per the circuit diagram given below:

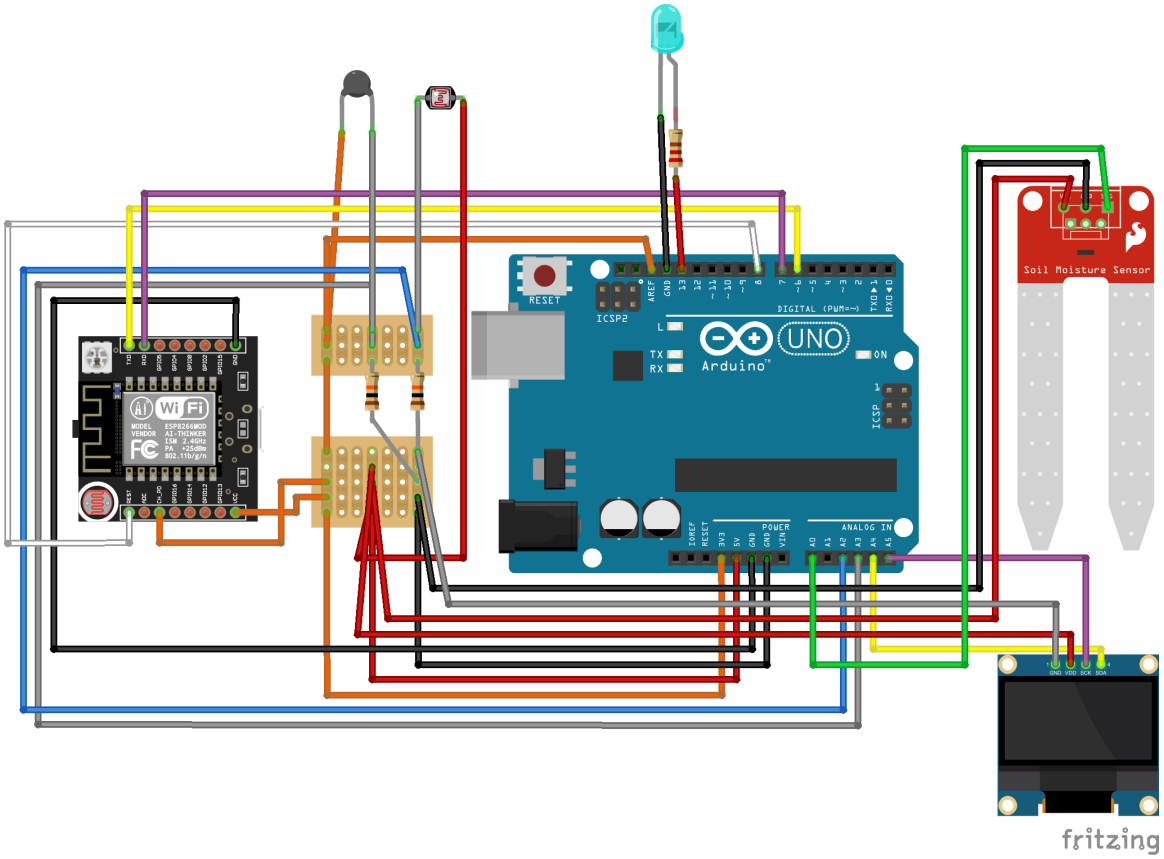

Figure 5. Final Circuit Diagram

The prototype has three sensors, viz., soil moisture sensor, temperature sensor and light sensor. All these sensors are attached to a stripboard. There is an LCD integrated with the system to show all the readings and a message. To get the data on the phone a Wi-Fi module is used. This Wi-Fi module sends the sensor readings to a cloud-based service called ThingSpeak via the local WiFi network. The app then uses the ThingSpeak API to display the sensor readings to any remote location. The actuator in our prototype is an LED, which is indicative of plant being watered. The entire circuit is encapsulated inside a plastic box, which is 3D printed. This box was custom designed for the prototype and nicely fits all its contents. The power source for the prototype is the serial port of the Arduino Uno, which is powered using a portable power-bank.

3.2 3D Printing:

Undoubtedly, one of the main learnings from this course was the art of 3D printing. We used *Blender* and *SketchUp* to design the parts and *Ultimaker Cura* to convert **STLs** to **Gcodes** for the 3D printer. Below are the images of few parts that we designed:

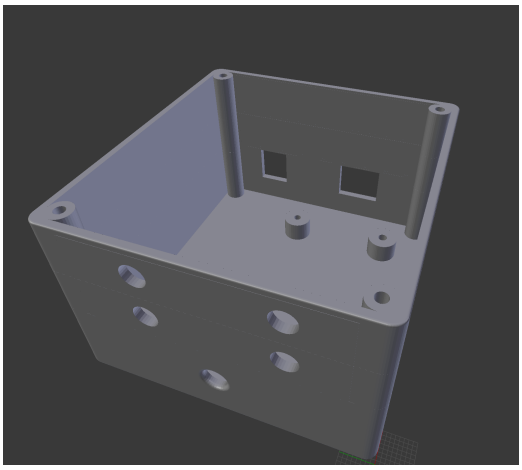

Figure 6. 3D Printed Box - Bottom

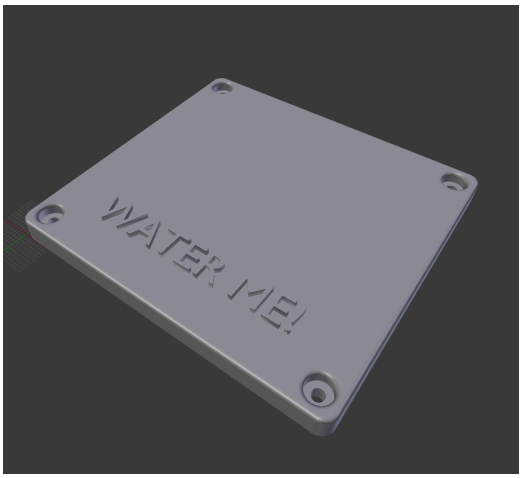

Figure 7. 3D Printed Box - Top

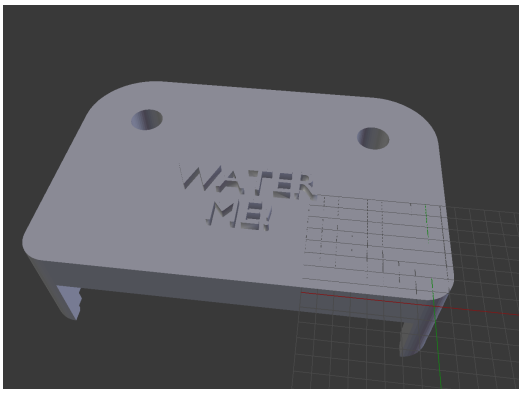

Figure 8. Soil Moisture Sensor Case

3.3 Software Components:

1. Arduino code:

The Arduino code is at the heart of this project. It reads the data from three sensors, displays it on the LCD screen and sends it to the cloud. This data is then viewed by users and the respective commands are issued. The complete Arduino code can be found in Appendix B.

2. Mobile Application:

A mobile application was designed to send the local plant data on the mobile devices of users. The data is captured from the sensors and sent to the cloud. From there, the user can view the data and give the commands to actuators remotely.

Sending the data:
To transfer the data to the user, thingspeak.com was used. The local unit will capture the data and send it to internet writing them on to a specific ThingSpeak Status Channel. The local unit will also receive data from the internet, *reading* them from a specific ThingSpeak Actuator Channel. The Android App will read the data from the ThingSpeak.com Status Channel and display them to the user. Similarly, the user, based on that status info, can send commands to actuators, writing them on ThingSpeak Actuator Channel.

The sensors used in our project are soil moisture sensor, temperature sensor, and light sensor. The actuator used is an LED, whose status is also sent online. Based on this the user can take a decision like "Turn on" or "Turn off" the led if the moisture level is low.

To facilitate this, an account was created on Thingspeak.com. Here a new channel was created, with the actuator as LED, the soil moisture in % , the temperature in degrees, and the light in %. To send the data on the cloud, *WRITE* command on ThingSpeak channel was used and for that, a *GET* string needs to be sent. This was done in 3 parts, first - a *start cmd* command was sent, followed by the length of the command and finally, the *GET* string itself.

Application design:
The user interface of the app looks as follows:

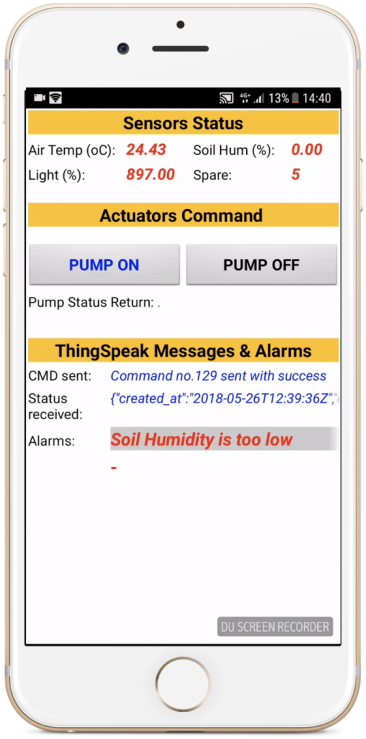

Figure 9. User Interface of the Mobile Application

Here the sensor readings and status of the actuators is visible. Every two seconds a procedure called "*read arduino*" is invoked. This procedure reads the Status Channel at ThingSpeak and returns the value for each status variables that should be displayed on the screen. To achieve this, a URL must be defined that points to the ThingSpeak cloud from where our Android App can fetch the data. Three global variables are declared and joined in order to create this URL. A GET command should be sent to the Web Component named *ArduFarmBotStatusCh*. The text will arrive as a JSon string, here each field is read and stored on corresponding global variables. There is an "Alarm" procedure which will analyze the status of soil sensors. If the soil moisture level is low, we display a flashing message on the screen, so that the user notices it. The complete Android code can be found in Appendix C.

4 Final Results:

We were able to complete the prototype towards the end and it was fully functional. Water me is responding well to slight differences in moisture levels. Both the mobile application and devices are working perfectly in sync. The user can see the readings of three sensors on their mobile devices and can give remote commands. We were happy to present this project in Hackathon - 2018. We received a lot of positive comments about this project. People were curious about it and showed a willingness to install something like this in their houses.

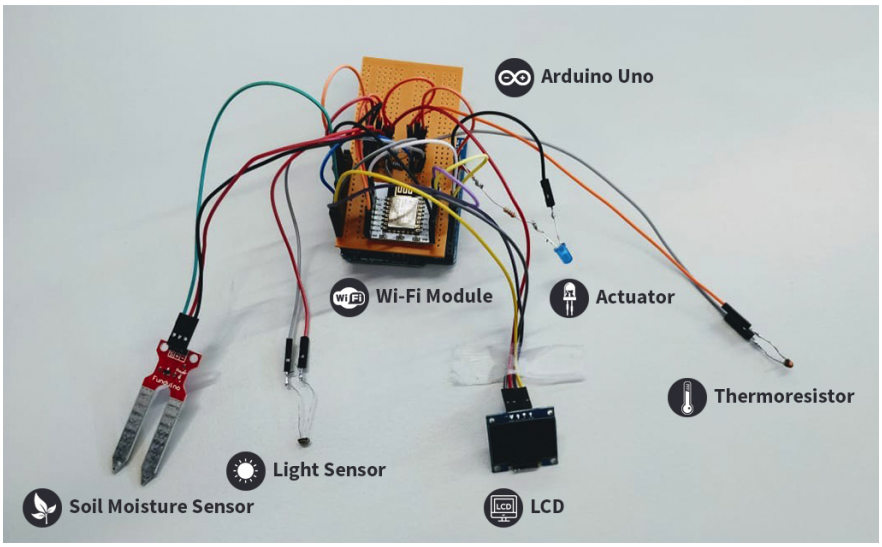

Figure 10. The circuit

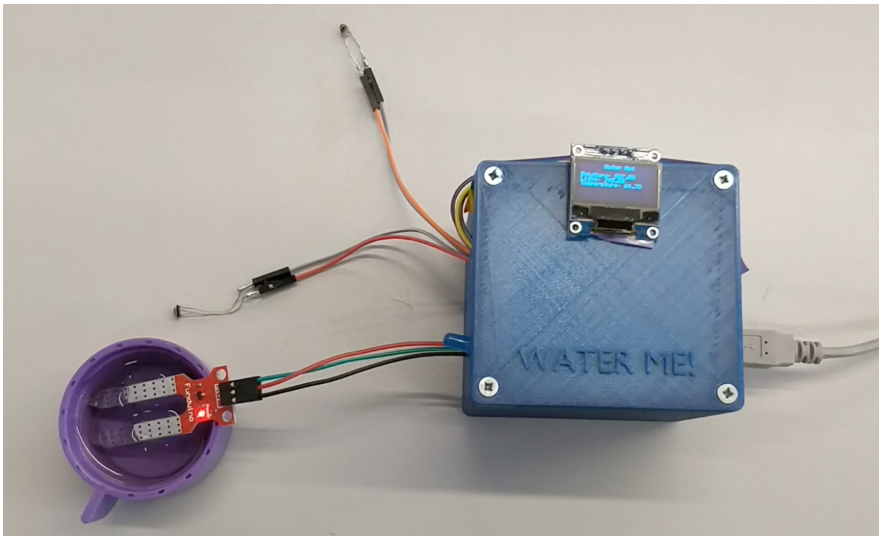

Figure 11. The final prototype

5 Reflection and Discussion:

We faced a lot of challenges during this project. However, all of those were countered by the constant help from the professor and the TAs. We started 3D printing early in our project. There were a lot of failed attempts before we got the final case for the prototype. We learned how to design 3D objects in *Blender* and perfected the sizing of the 3D objects. We also learned how to emboss and engrave product branding on a 3D object (in our case the text *'Water Me!'*). But the most prominent thing that we learned was combining two 3D objects.

The major reason for unsuccessful 3D models was the incorrect combination of 3D objects. The observation here was that when we combine two objects in *Blender* depending upon the Boolean function we use, the intersection of these objects is treated as a completed filled solid or a void. The learning here was to reduce this intersection between the objects to a minimum. After this precaution, we saw 100% success in all the parts that we 3D printed.

We had no experience in soldering and it was also a very time-consuming task. We had to redo the entire circuit twice because of small, unidentifiable short circuits due to the soldering. The third attempt was a charm. The learning here was to keep as much separation between the component groups, that you want to solder, as possible. In our third attempt, we kept a 2 holes difference between the ground, the 5V and the 3.3V wires on the stripboard.

The first watering mechanism that we made was either too leaky or too slow in watering the plant and wouldn't have been presentable during the demo (shown in Figure 12 and 13). We immediately decided to replace our watering mechanism with a miniaturized water pump, like the ones used in aquariums. That brings us to the last learning of this project. We learned that we should decide upon the components early in the project and order them as soon as possible. If we are ordering them online, which we do in most cases, we should check what part of the world these components are coming from. We ordered the mini pump from Amazon.com which promised the delivery in 4-5 days but we didn't receive it for another 1.5 months. We had no choice but to replace it by an LED, so that we can provide a **Proof of Concept** for our prototype's working. Finally, we received a package from China, 11 days after the final presentation, which had the mini pump. The explanation that the vendor gave us was that it was held at Swiss customs for several days.

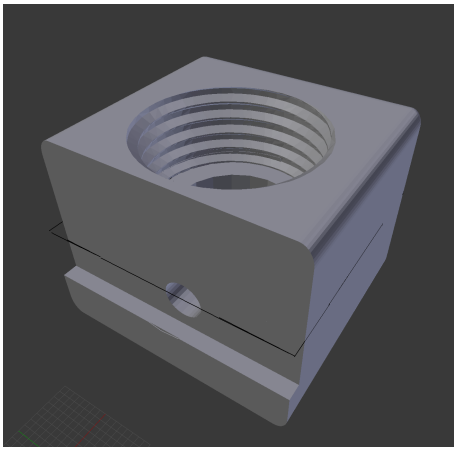

Figure 12. Watering Mechanism - Base

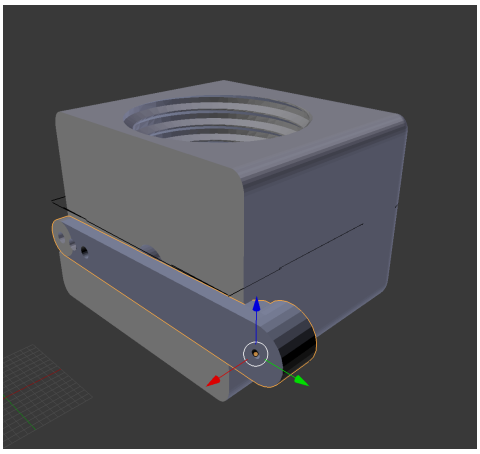

Figure 13. Watering Mechanism - Lever

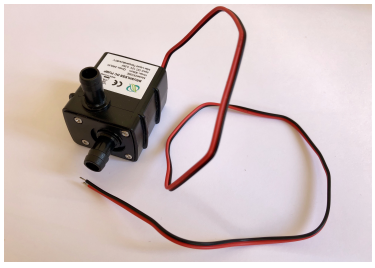

Figure 14. The mini pump that arrived late

6 Appendix A - Contemporaries

Here we mention the contemporaries of our prototype that are already available in the market and their costs.

Parrot POT
Cost: 125 CHF

Figure 15. The Parrot POT
Source: Parrot's Official Website

Parrot Flower Power
Cost: 99 CHF

Figure 16. The Parrot Flower Power
Source: Parrot's Official Website

7 Appendix B - Arduino code

Listing 1. Arduino Code

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <SimpleTimer.h>
#include <VarSpeedServo.h>

VarSpeedServo myservo;
SimpleTimer timer;
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
#define NUMFLAKES 10
#define XPOS 0
#define YPOS 1
#define DELTAY 2
#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH  16
const int soil_sensor = A0;  // Analog input pin that the soil moisture sensor is attached to
const int temp_sensor = A3;
const int light_sensor = A2;
const int LED = 13;
const int servoPin = 9; // the digital pin used for the servo
float moist,light,temp;
#define THERMISTORNOMINAL 10000
// temp. for nominal resistance (almost always 25 C)
#define TEMPERATURENOMINAL 25
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define BCOEFFICIENT 3950
// the value of the 'other' resistor
#define SERIESRESISTOR 10000
#define aref_voltage 3.3
uint16_t samples[NUMSAMPLES];
void repeatMe() {
Serial.print("Uptime␣(s):␣");
Serial.println(millis() / 1000);
}
void setup() {
// initialize serial communications at 9600 bps:
Serial.begin(9600);
timer.setInterval(1000, repeatMe);
  pinMode(LED, OUTPUT);  // Set lepPin - 9 pin as an output
pinMode(light_sensor, INPUT);// Set pResistor - A0 pin as an input (optional)
pinMode(temp_sensor, INPUT);// Set pResistor - A0 pin as an input (optional)
myservo.attach(servoPin);  // attaches the servo on pin 9 to the servo object
myservo.write(0,255,true); // set the intial position of the servo, as fast as possible, wait until done
 analogReference(EXTERNAL);
```

```
50 // by default, we'll generate the high voltage from the 3.3v line
      internally! (neat!)
51 display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
52 display.clearDisplay();
53 display.setTextSize(2);
54 display.setTextColor(WHITE);
55 display.setCursor(0,0);
56 display.display();
57 }
58
59 void loop() {
60 // read the sensor:
61 moist = analogRead(soil_sensor);
62 light = analogRead(light_sensor);
63 temp = analogRead(temp_sensor);
64 uint8_t i;
65 float average;
66 // take N samples in a row, with a slight delay
67 for (i=0; i< NUMSAMPLES; i++) {
68 samples[i] = analogRead(temp_sensor);
69 delay(10);
70 }
```

8 Appendix C - Mobile Application Code

The Android app was developed using *MIT App Inventor 2*. App Inventor 2 is a visual programming language similar to *Scratch*. The following is the code for the Android app: (it is named *ArduFarmBot.aia* in the repository)

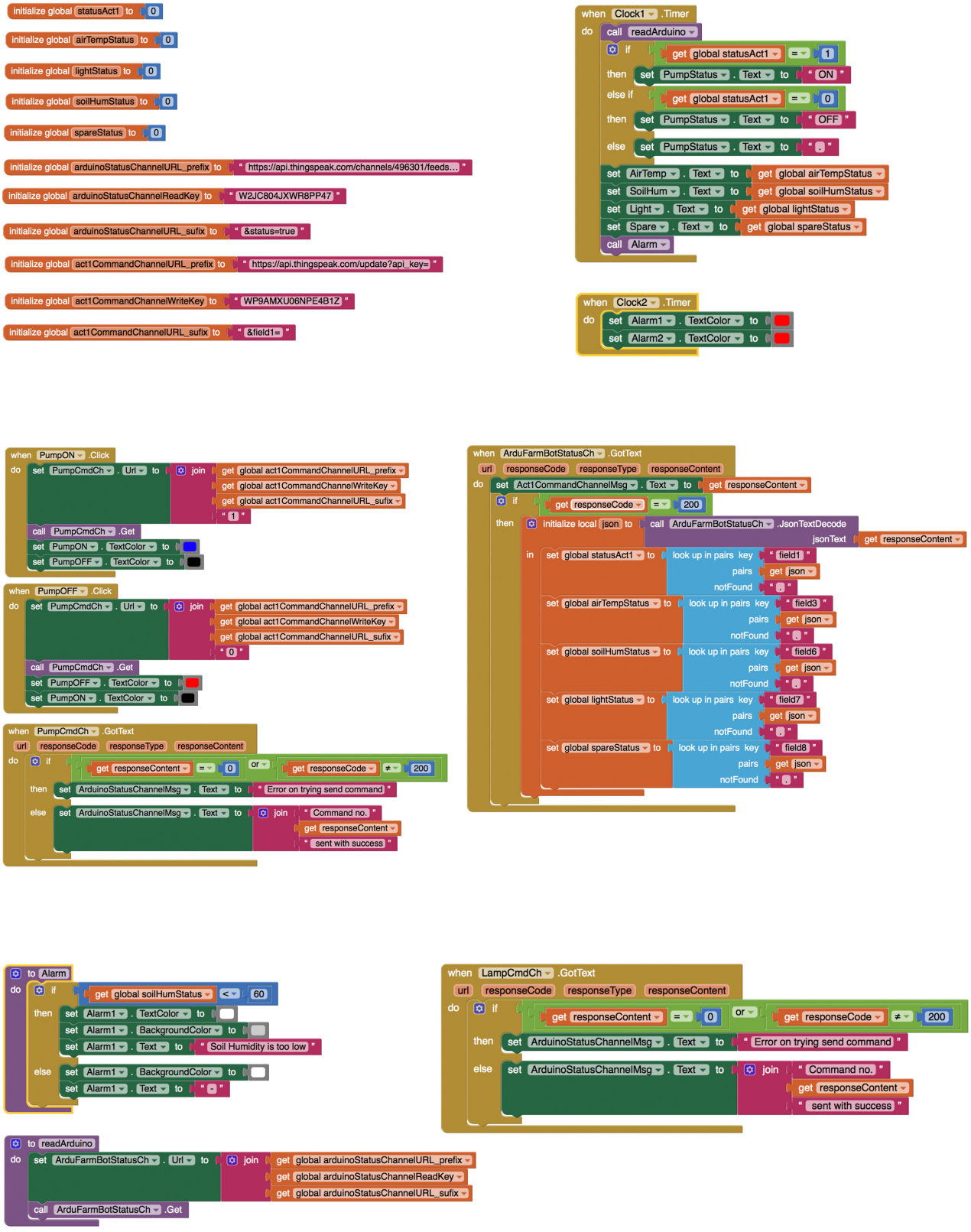

Figure 17. The Android application code

Listing 2. Arduino code for sending data to the Android app

```
String statusChWriteKey = "AIBGG8QEBB8D896T";
String canalID1 = "496303"; //  Actuator1 Channel ID
//String canalID2 = "375599";//
#include <SoftwareSerial.h>
#include <SPI.h>
#include <Wire.h>
//#include <SimpleTimer.h>
#include <VarSpeedServo.h>
SoftwareSerial EspSerial(6, 7); // Rx,  Tx
#define HARDWARE_RESET 8
VarSpeedServo myservo;
const int soil_sensor = A0;  // Analog input pin that the soil moisture
    sensor is attached to
const int temp_sensor = A5;
const int light_sensor = A2;
const int LED = 10;
const int servoPin = 9; // the digital pin used for the servo
float moist,light,temp;
float steinhart;
#define FREEZE_LED 10
#define THERMISTORNOMINAL 10000
// temp. for nominal resistance (almost always 25 C)
#define TEMPERATURENOMINAL 25
// how many samples to take and average, more takes longer
// but is more 'smooth'
#define NUMSAMPLES 5
// The beta coefficient of the thermistor (usually 3000-4000)
#define BCOEFFICIENT 3950
// the value of the 'other' resistor
#define SERIESRESISTOR 10000
#define aref_voltage 3.3
uint16_t samples[NUMSAMPLES];
// Variables to be used with timers
long writeTimingSeconds = 17; // ==> Define Sample time in seconds to send
     data 16
long readTimingSeconds = 1; // ==> Define Sample time in seconds to
    receive data // changed from 5
long startWriteTiming = 0;
long elapsedWriteTime = 0;
long startReadTiming = 0;
long elapsedReadTime = 0;
int spare = 0;
boolean error;
boolean pump = 0;
boolean lamp = 0;
//Relays
#define ACTUATOR1 13 // RED LED   ==> Pump
#define ACTUATOR2 3 // GREEN LED ==> Lamp
void setup()
{
Serial.begin(9600);
pinMode(HARDWARE_RESET,OUTPUT);
pinMode(ACTUATOR1,OUTPUT);
digitalWrite(HARDWARE_RESET, HIGH);
digitalWrite(ACTUATOR1, LOW); //LOW
// DS18B20.begin();
//dht.begin();
```

```
55 EspSerial.begin(9600); // Comunicacao com Modulo WiFi
56 EspHardwareReset(); //Reset do Modulo WiFi
57 startWriteTiming = millis(); // starting the "program clock"
58 pinMode(LED, OUTPUT);  // Set lepPin - 9 pin as an output
59 pinMode(light_sensor, INPUT);// Set pResistor - A0 pin as an input (
      optional)
60 pinMode(temp_sensor, INPUT);// Set pResistor - A0 pin as an input (
      optional)
61 myservo.attach(servoPin);  // attaches the servo on pin 9 to the servo
      object
62 myservo.write(0,255,true); // set the intial position of the servo, as
      fast as possible, wait until done
63 analogReference(EXTERNAL);
64 // by default, we'll generate the high voltage from the 3.3v line
      internally! (neat!)
65 }
66
67 void loop()
68 {
69 start: //label
70 error=0;
71
72 elapsedWriteTime = millis()-startWriteTiming;
73 elapsedReadTime = millis()-startReadTiming;
74
75  if (elapsedReadTime > (readTimingSeconds*1000))
76 {
77 ESPcheck();//executar antes de qualquer leitura ou gravaÃğÃčo
78 int command = readThingSpeak(canalID1);
79 if (command != 9) pump = command;
80 //delay (5000);
81 //  command = readThingSpeak(canalID2);
82 //if (command != 9) lamp = command;
83
84 takeActions();
85 startReadTiming = millis();
86 }
87
88 if (elapsedWriteTime > (writeTimingSeconds*1000))
89 {
90 ESPcheck();//executar antes de qualquer leitura ou gravaÃğÃčo
91
92 // read the sensor:
93 moist = analogRead(soil_sensor);
94 light = analogRead(light_sensor);
95 temp = analogRead(temp_sensor);
96
97
98 uint8_t i;
99 float average;
100
101 // take N samples in a row, with a slight delay
102 for (i=0; i< NUMSAMPLES; i++) {
103 samples[i] = analogRead(temp_sensor);
104 delay(10);
105 }
106 // average all the samples out
107 average = 0;
```

```
108 for (i=0; i< NUMSAMPLES; i++) {
109 average += samples[i];
110 }
111 average /= NUMSAMPLES;
112
113 Serial.print("Average␣analog␣reading␣");
114 Serial.println(average);
115
116 // convert the value to resistance
117 average = 1023 / average - 1;
118
119  average = SERIESRESISTOR / average;
120 Serial.print("Thermistor␣resistance␣");
121 Serial.println(average);
122
123
124 steinhart = average / THERMISTORNOMINAL;     // (R/Ro)
125 steinhart = log(steinhart);                  // ln(R/Ro)
126 steinhart /= BCOEFFICIENT;                   // 1/B * ln(R/Ro)
127 steinhart += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
128 steinhart = 1.0 / steinhart;                // Invert
129 steinhart -= 273.15;                        // convert to C
130
131
132 Serial.print("Temperature␣");
133 Serial.print(steinhart);
134 Serial.println("␣*C");
135
136
137 Serial.print("light␣Value␣=␣" );
138 Serial.println(light);
139
140 Serial.print("Moist␣Value␣=␣" );
141 Serial.println(moist);
142
143 writeThingSpeak();
144 startWriteTiming = millis();
145 }
146
147 if (error==1) //Resend if transmission is not completed
148 {
149 Serial.println("␣<<<<␣ERROR␣>>>>");
150 digitalWrite(FREEZE_LED, HIGH);
151 delay (2000);
152 goto start; //go to label "start"
153 }
154 }
155 /********* Conexao com TCP com Thingspeak *******/
156 void writeThingSpeak(void)
157 {
158
159 startThingSpeakCmd();
160
161 // preparacao da string GET
162 String getStr = "GET␣/update?api_key=";
163
164 getStr += statusChWriteKey;
165 getStr +="&field6=";
```

```
166 getStr += String(moist);
167 getStr +="&field7=";
168 getStr += String(light);
169 getStr +="&field3=";
170 getStr += String(steinhart);
171 getStr +="&field8=";
172 getStr += String(spare);
173 getStr += "\r\n\r\n";
174
175 sendThingSpeakGetCmd(getStr);
176 }
177
178 /********* Reset ESP *************/
179 void EspHardwareReset(void)
180 {
181 Serial.println("Reseting.......");
182 digitalWrite(HARDWARE_RESET, LOW);
183 delay(500);
184 digitalWrite(HARDWARE_RESET, HIGH);
185 delay(8000);//Tempo necessÃąrio para comeÃğar a ler
186 Serial.println("RESET");
187 }
188
189 /********* Start communication with ThingSpeak*************/
190 void startThingSpeakCmd(void)
191 {
192 EspSerial.flush();//limpa o buffer antes de comeÃğar a gravar
193
194 String cmd = "AT+CIPSTART=\"TCP\",\"";
195 cmd += "184.106.153.149"; // Endereco IP de api.thingspeak.com
196 cmd += "\",80";
197 EspSerial.println(cmd);
198 Serial.print("revived ==> Start cmd: ");
199 Serial.println(cmd);
200
201 if(EspSerial.find("Error"))
202 {
203 Serial.println("AT+CIPSTART error");
204  return;
205 }
206 }
207
208 /********* send a GET cmd to ThingSpeak *************/
209 String sendThingSpeakGetCmd(String getStr)
210 {
211 String cmd = "AT+CIPSEND=";
212 cmd += String(getStr.length());
213 EspSerial.println(cmd);
214 Serial.print("revive ==> lenght cmd: ");
215 Serial.println(cmd);
216
217  if(EspSerial.find((char *)">"))
218 {
219 EspSerial.print(getStr);
220 Serial.print("revive ==> getStr: ");
221 Serial.println(getStr);
222 delay(500);//tempo para processar o GET, sem este delay apresenta busy no
      prÃşximo comando
```

```
223
224 String messageBody = "";
225 while (EspSerial.available())
226 {
227 String line = EspSerial.readStringUntil('\n');
228 if (line.length() == 1)
229 { //actual content starts after empty line (that has length 1)
230 messageBody = EspSerial.readStringUntil('\n');
231 }
232 }
233 Serial.print("MessageBody␣received:␣");
234 Serial.println(messageBody);
235 return messageBody;
236 }
237 else
238 {
239 EspSerial.println("AT+CIPCLOSE");     // alert user
240 Serial.println("ESP8266␣CIPSEND␣ERROR:␣RESENDING"); //Resend...
241 spare = spare + 1;
242 error=1;
243 return "error";
244 }
245 }
246
247 void takeActions(void)
248 {
249 Serial.print("Pump:␣");
250 Serial.println(pump);
251 // Serial.print("Lamp: ");
252  // Serial.println(lamp);
253 if (pump == 1) digitalWrite(ACTUATOR1, HIGH);
254 else digitalWrite(ACTUATOR1, LOW);
255 // if (lamp == 1) digitalWrite(ACTUATOR2, LOW);
256 //  else digitalWrite(ACTUATOR2, HIGH);
257 }
258
259
260 boolean ESPcheck(void)
261 {
262 EspSerial.println("AT"); // Send "AT+" command to module
263
264 if (echoFind("OK"))
265 {
266 //Serial.println("ESP ok");
267 digitalWrite(FREEZE_LED, LOW);
268 return true;
269 }
270 else //Freeze ou Busy
271 {
272 Serial.println("ESP␣Freeze␣
    **************************************************");
273 digitalWrite(FREEZE_LED, HIGH);
274 EspHardwareReset();
275 return false;
276 }
277 }
278 int readThingSpeak(String channelID)
279 {
```

```
startThingSpeakCmd();
int command;
// preparacao da string GET
String getStr = "GET /channels/";
getStr += channelID;
getStr +="/fields/1/last";
getStr += "\r\n";

String messageDown = sendThingSpeakGetCmd(getStr);
if (messageDown[5] == 49)
{
command = messageDown[7]-48;
Serial.print("Command received: ");
Serial.println(command);
}
else command = 9;
return command;
}
boolean echoFind(String keyword)
{
byte current_char = 0;
byte keyword_length = keyword.length();
long deadline = millis() + 5000; // Tempo de espera 5000ms
while(millis() < deadline){
if (EspSerial.available()){
char ch = EspSerial.read();
Serial.write(ch);
if (ch == keyword[current_char])
if (++current_char == keyword_length){
Serial.println();
return true;
}
}
}
return false; // Tempo de espera esgotado
}
```

9 Appendix D - Project Links