

REYKJAVÍK UNIVERSITY

MECHATRONICS I

T-411-MECH



LABORATORY 8

7/10/2020

Students:

Giacomo Menchi
Matteo Guerrini
William Paciaroni

Teacher:

Joseph T. Foley

Contents

1	Main Tasks	1
1.1	Main Task 1	1
1.2	Main Task 2	2
1.3	Main Task 3	2
1.4	Main Task 4	3
2	Hard Tasks	4
2.1	Hard Task 1	4
2.2	Hard Task 2	6
3	Advanced Tasks	6
3.1	Advanced Task 1	6
3.2	Advanced Task 2	6

1 Main Tasks

This below is the GitHub link with all the code inside. We also included two different versions of the Arduino code for writing values on OLED display (Advanced Task 1) since we made one which used bitmap pictures and another with only text.

Link: GitHub Repo

1.1 Main Task 1

Task: Ping the weather server IP to see if it exists

Screenshots: We first looked up online what was the IP of the website developer.accuweather.com (first screenshot), and then pinged both developer.accuweather.com and the IP itself (the first ping also gave us the same IP we found online), obtaining the same result.

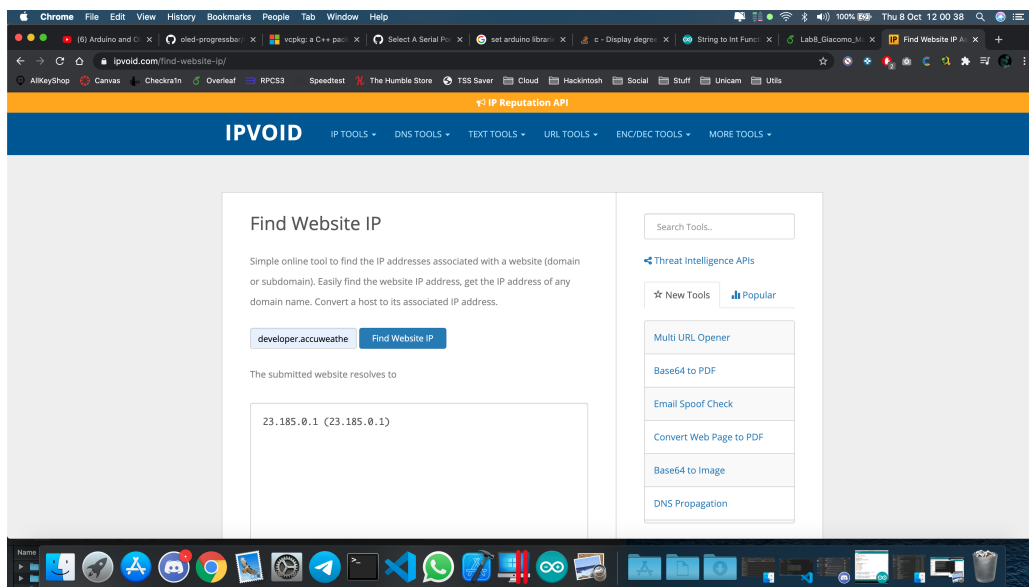


Figure 1: Finding developer.accuweather.com IP.

```
giacomomenchi@Giacomo-McMenchi ~ % ping developer.accuweather.com
PING developer.accuweather.com (23.185.0.1): 56 data bytes
64 bytes from 23.185.0.1: icmp_seq=0 ttl=54 time=242.955 ms
64 bytes from 23.185.0.1: icmp_seq=1 ttl=54 time=248.011 ms
64 bytes from 23.185.0.1: icmp_seq=2 ttl=54 time=39.261 ms
64 bytes from 23.185.0.1: icmp_seq=3 ttl=54 time=53.173 ms
^C
--- developer.accuweather.com ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 39.261/145.850/248.011/99.770 ms
giacomomenchi@Giacomo-McMenchi ~ % ping 23.185.0.1
PING 23.185.0.1 (23.185.0.1): 56 data bytes
64 bytes from 23.185.0.1: icmp_seq=0 ttl=54 time=40.013 ms
64 bytes from 23.185.0.1: icmp_seq=1 ttl=54 time=77.721 ms
64 bytes from 23.185.0.1: icmp_seq=2 ttl=54 time=126.628 ms
64 bytes from 23.185.0.1: icmp_seq=3 ttl=54 time=39.725 ms
^C
--- 23.185.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 39.725/71.022/126.628/35.630 ms
giacomomenchi@Giacomo-McMenchi ~ %
```

Figure 2: Ping developer.accuweather.com.

1.2 Main Task 2

Task: Connect to the weather server IP and request weather data

Screenshot: By using curl, the endpoints provided by accuweather.com and one of their APIs, we requested data from the website and received this long JSON string below as a response.

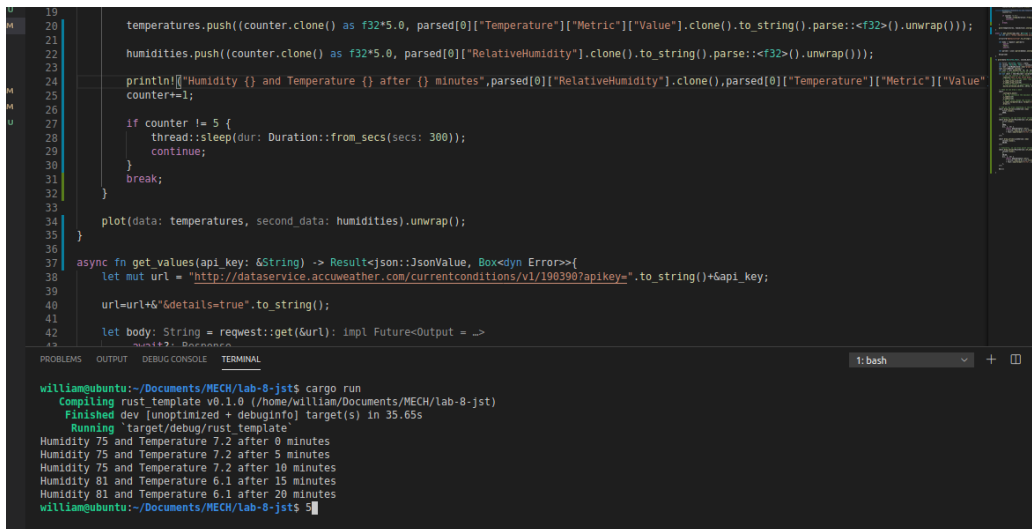
```
matteoguerrini@Matteos-MacBook-Air ~ % curl -X GET "http://dataservice.accuweather.com/forecasts/v1/daily/1day/190390?apikey=itqwuy
[ZDo4tu3TQt5rVAEFT8oBNgRjH&language=en-us&details=false&metric=false"
{"Headline":{"EffectiveDate":"2020-10-11T13:00:00+00:00","EffectiveEpochDate":1602421200,"Severity":2,"Text":"Expect rainy weather S
unday afternoon through Sunday evening","Category":"rain","EndDate":"2020-10-12T01:00:00+00:00","EndEpochDate":1602464400,"MobileLin
k":"http://m.accuweather.com/en/is/reykjavik/190390/extended-weather-forecast/190390?lang=en-us","Link":"http://www.accuweather.com/
en/is/reykjavik/190390/daily-weather-forecast/190390?lang=en-us"},"DailyForecasts":[{"Date":"2020-10-07T07:00:00+00:00","EpochDate":
1602054000,"Temperature":{"Minimum":{"Value":39.0,"Unit":"F","UnitType":18},"Maximum":{"Value":46.0,"Unit":"F","UnitType":18}}, "Day"
matteoguerrini@Matteos-MacBook-Air ~ %
```

Figure 3: Weather data obtainment.

1.3 Main Task 3

Task: Have a program connect to it once every 5 minutes and disconnect again. (If using accuweather, you will want to only do this briefly due to the 50/day limit.)

Screenshot: The screenshot below shows how the weather data is read every five minutes and printed on terminal (the values stay the same at first, but change after a while since the forecast is updated sometimes).



```

19     temperatures.push((counter.clone() as f32*5.0, parsed[0]["Temperature"]["Metric"]["Value"].clone().to_string().parse::<f32>().unwrap()));
20
21     humidities.push((counter.clone() as f32*5.0, parsed[0]["RelativeHumidity"].clone().to_string().parse::<f32>().unwrap()));
22
23     println!["Humidity {} and Temperature {} after {} minutes",parsed[0]["RelativeHumidity"].clone(),parsed[0]["Temperature"]["Metric"]["Value"]
24     counter+=1;
25
26
27     if counter != 5 {
28         thread::sleep(dur: Duration::from_secs(secs: 300));
29         continue;
30     }
31     break;
32 }
33
34 plot(data: temperatures, second_data: humidities).unwrap();
35 }
36
37 async fn get_values(api_key: &String) -> Result<json::JsonValue, Box<dyn Error>>{
38     let mut url = "http://dataservice.accuweather.com/currentconditions/v1/198390?apikey=" + api_key;
39
40     url=url+"&details=true".to_string();
41
42     let body: String = request::get(&url).impl Future<Output = >
43
44

```

```

william@ubuntu:~/Documents/MECH/lab-8-jst$ cargo run
Compiling rust-templ v0.1.0 (/home/william/Documents/MECH/lab-8-jst)
Finished dev [unoptimized + debuginfo] target(s) in 35.65s
Running 'target/debug/rust-templ'
Humidity 75 and Temperature 7.2 after 0 minutes
Humidity 75 and Temperature 7.2 after 5 minutes
Humidity 75 and Temperature 7.2 after 10 minutes
Humidity 81 and Temperature 6.1 after 15 minutes
Humidity 81 and Temperature 6.1 after 20 minutes
william@ubuntu:~/Documents/MECH/lab-8-jst$

```

Figure 4: Weather data obtainment with a fixed interval.

1.4 Main Task 4

Task: Parse and filter out the weather data and print the temperature and humidity for Reykjavik in a chart

Screenshot: Since we had only a bunch of requests for getting data out of accuweather, we combined this step with the one before and directly used "plotters" library to plot the graph below (hence, the data represented is the same as the screen above since both screenshots have been taken during one single execution of the program).

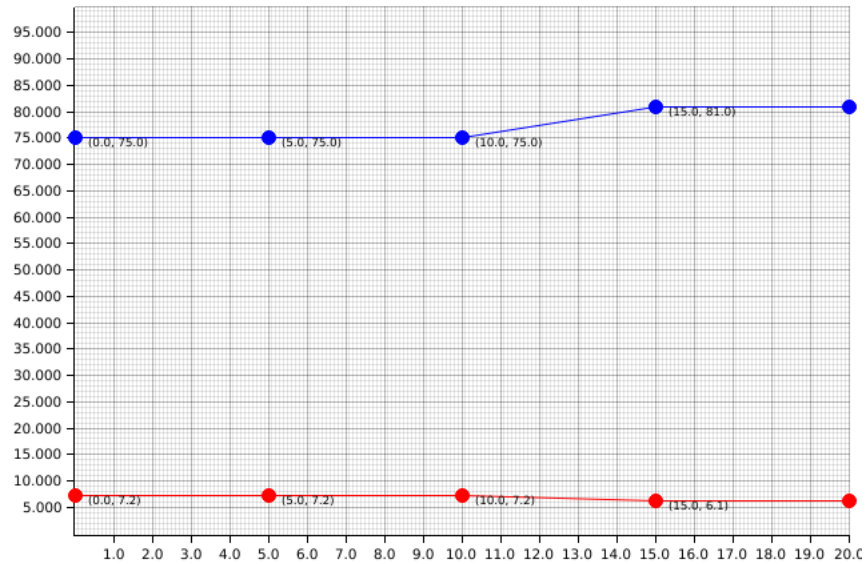


Figure 5: Weather data plotted graph.

2 Hard Tasks

2.1 Hard Task 1

Task: Have the program upload the weather data to Adafruit IO as two different feeds

Screenshots: The screenshots below show how the weather data which are requested and obtained from the accuweather website using Rust are then uploaded to Adafruit IO and shown in a Adafruit graph.

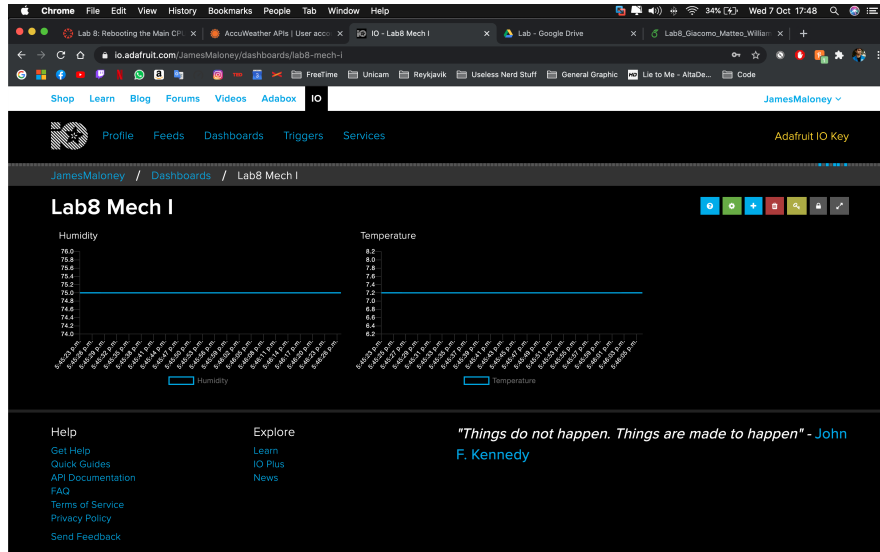


Figure 6: Adafruit IO dashboard with feeds.

The screenshot shows the Visual Studio Code editor with a file named 'main.rs' open. The code is written in Rust and uses the 'tokio' and 'reqwest' crates. It defines an asynchronous function 'main' that fetches weather data from AccuWeather and uploads it to Adafruit IO. The code includes comments and error handling. The terminal at the bottom shows the output of the 'cargo run' command, indicating that the code compiled successfully and the program is running.

```

13 #![tokio::main]
14
15 async fn main() -> Result<()> {
16     let api_key: String = "GULdU9YfSVABlQp3Yb5Nk2zRSb6sfwT".to_string();
17     let parsed: JsonValue = get_values(&api_key).await.unwrap();
18     let temp: &JsonValue = &parsed[0]["temperature"]["value"];
19     let humidity: &JsonValue = &parsed[0]["relativeHumidity"];
20     upload_to_ada(humidity.to_string(), temp.to_string());
21     Ok(())
22 }
23
24 fn upload_to_ada(humidity: String, temperature: String) {
25     let username: &str = "JamesMaloney";
26     let aiokey: &str = "aio_0Hf07680Bwdfp202HV8e2usguLB";
27     let mut ada: AdaClient = adafruit_io_http::AdaClient::set(n1: username.to_string(), n2: aiokey.to_string());
28     loop {
29         let humidity_feedkey: &str = "humidity";
30         let temperature_feedkey: &str = "temperature";
31         ada.post(n3: humidity_feedkey.to_string(), data: humidity.to_string());
32         ada.post(n3: temperature_feedkey.to_string(), data: temperature.to_string());
33         Thread::sleep(Duration::from_secs(5));
34     }
35 }

```

Figure 7: Code requesting from accuweather and uploading to Adafruit IO.

2.2 Hard Task 2

Task: While the first program is active have another program to request data from the service and if the temperature is above a threshold then make the pi warn you.

Video: The video below shows the task realization. We used two different programs, one executed in Ubuntu and one on Raspberry Pi: the first one gets data from accuweather and uploads it to Adafruit IO while the second one reads from the Adafruit and turns on a LED if threshold (in our case 6.0 degrees) is passed, while also writing a warning on the terminal.

Link: [Video Link](#)

3 Advanced Tasks

3.1 Advanced Task 1

Task: Install OLED display to print weather data (temperature and humidity) on it.

Video: Shown below. The Rust code sends the request to accuweather, parses obtained data and sends temperature and humidity to Arduino, which then prints it on the OLED display (in the Arduino sketch we set a very high delay to avoid updating values more than necessary since we only request once every five minutes to the accuweather website anyway).

Link: [Video Link](#)

3.2 Advanced Task 2

Task: Visualize the data on OLED display as a picture or graph Think "Magic Window" types of IOT projects.

Video: The video below shows the Advanced Task 2, thus a summary of all the tasks executed at the same time. As we also say in the video, the temperature bar goes from 0 to 50 (we set this values for representation

convenience, of course temperature can easily go lower than 0 and sometimes also higher than 50), while the humidity is represented in percentage (hence the bar simply goes from 0 to 100%). The numeric value is also printed as a check.

Link: [Video Link](#)