TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Design with Microprocessors

Arduino Project with Sensors

Weather Station

Student: Ciui Bianca-Laura

GroupID: 30431

Contents

1. My Arduino Project – Weather Station

I intend to create a weather station using the Arduino Uno microcontroller and sensors. This weather station will be able to measure and display the temperature, humidity, atmospheric pressure and altitude.

For an extra feature, I'll add a connected android application for displaying in real-time all the measurements mentioned above as well as a report on the measurements recorded in the last week.

2. Technologies

- What is Arduino?

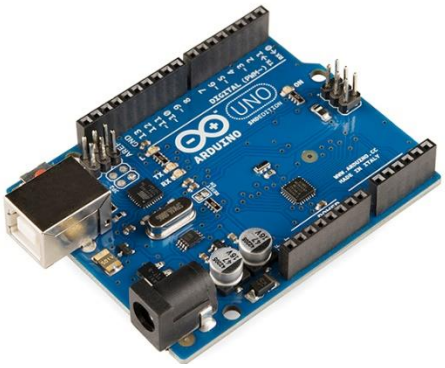


Figure 1 - Arduino Uno SMD R3
Source: https://en.wikipedia.org/wiki/Arduino

Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices.

Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboard and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers can be programmed using the C and C++ programming languages, using a standard API which is also known as the "Arduino language". In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) and a command line tool (arduino-cli) developed in Go.

- Firebase

Firebase is a platform developed by Google for creating mobile and web applications, that gives us functionalities like analytics, databases, messaging and crash reporting. Firebase provides a quick way to keep sensory data collected at the device level, so I will be using this platform to store the data recorded by my sensors. For using this platform, I had to create an account and then create my real-time database.

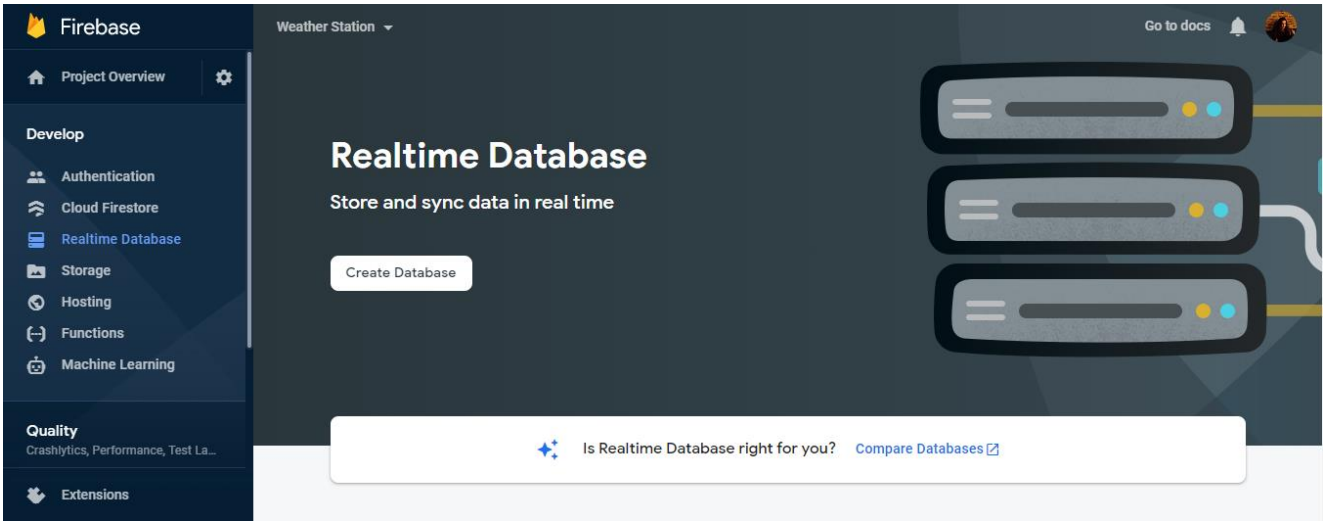


Figure 2 – Firebase platform

The video tutorial provided on their website when first entering the Realtime Database section was very useful and beginner friendly. I created my simple database which will be the storage place for my application. The structure can be observed below, in Figure 3.

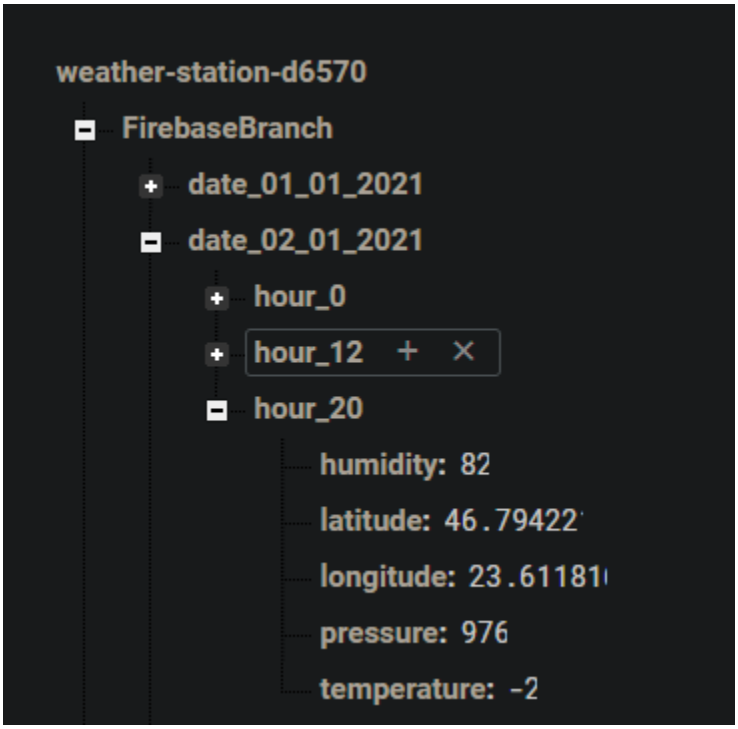
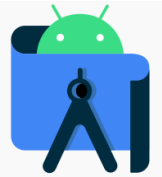


Figure 3 – Structure of the database used for developing this application

- Android Studio



Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. I chose to use this development enviroment over Eclipse because of its interface that is easier to understand and use. Also, the drag-and-drop option makes de designing part so much more enjoyable. The interface of the app is presented in figure 5.

Figure 4 – Android Studio Logo

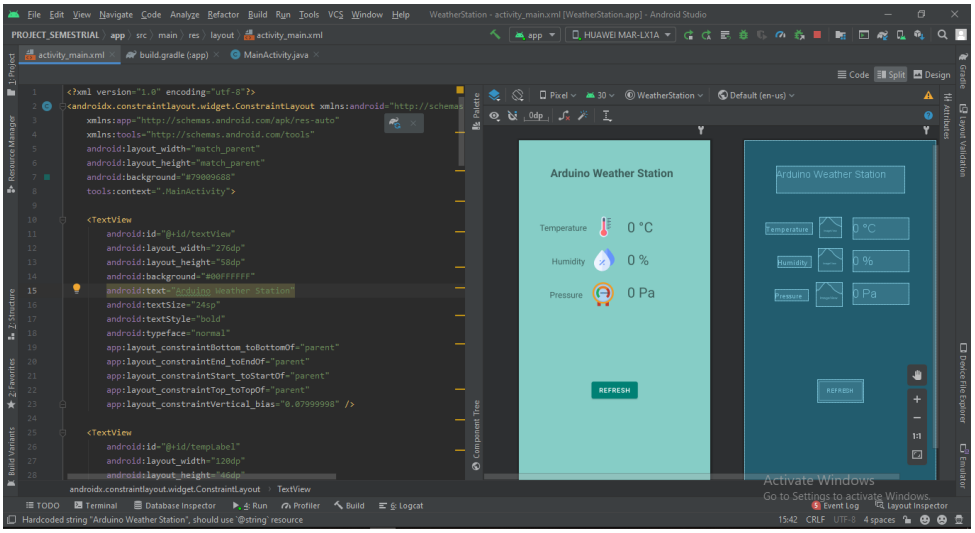


Figure 5 – Android Studio Interface

- How it works:

The sensors will be programed in Arduino IDE and the program will send data through the NodeMCU module (using a connection to Wi-fi) to the Firebase database. From this part, the database will receive information about temperature, humidity and atmospheric pressure.

The Android application will also be sending information to the database about the GPS coordinates and altitude of the Android device in order to save the data in the database.

Then, the Android app will connect to the Firebase database and extract the data from there and display it onto the Android device that runs the application.

3. Analysing and Design

In this chapter we'll go over the components that are used for creating the weather station, the way we connect these components to the NodeMCU, the Wi-fi module (and also to the Arduino UNO), and the code that is needed by these components to work properly.

- SNS-DH11 temperature and humidity sensor:

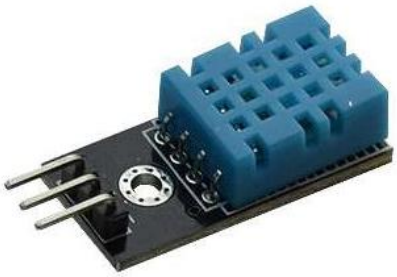


Figure 6 - SNS-DH11 temperature and humidity sensor

The DHT11 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old.

After doing the wiring presented in Figure 7 and after writing the code from Figure 8 in the Arduino IDE, we can test our sensor in the Serial Monitor (also displayed in Figure 8). I tested my sensor by holding it between my fingers and watch it record higher humidity and temperature. A very useful library for reading and processing the input of this sensor is DHT11Lib (we can add a library directly using the Arduino IDE by clicking in the menu bar on the "Tool" button and then "Manage Libraries…" and a Library Manager window will appear on the screen where we can search for the needed library).

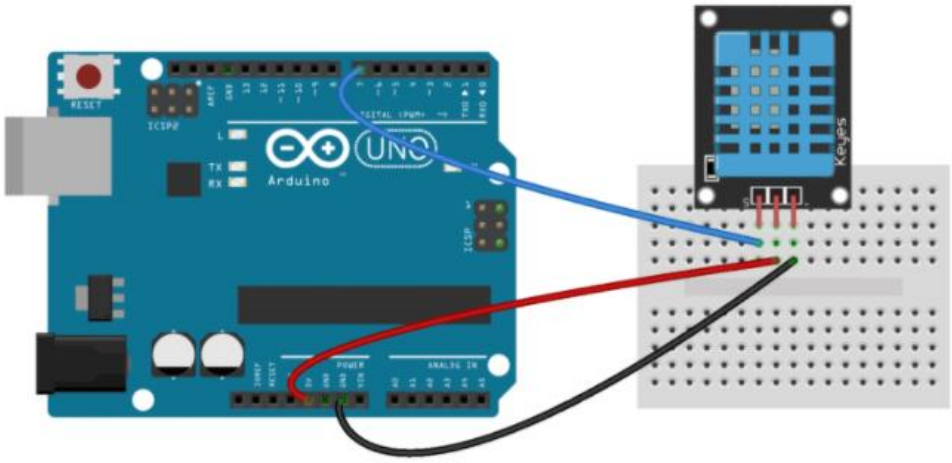


Figure 7 – Wiring of the SNS-DH11 Temperature and Humidity Sensor

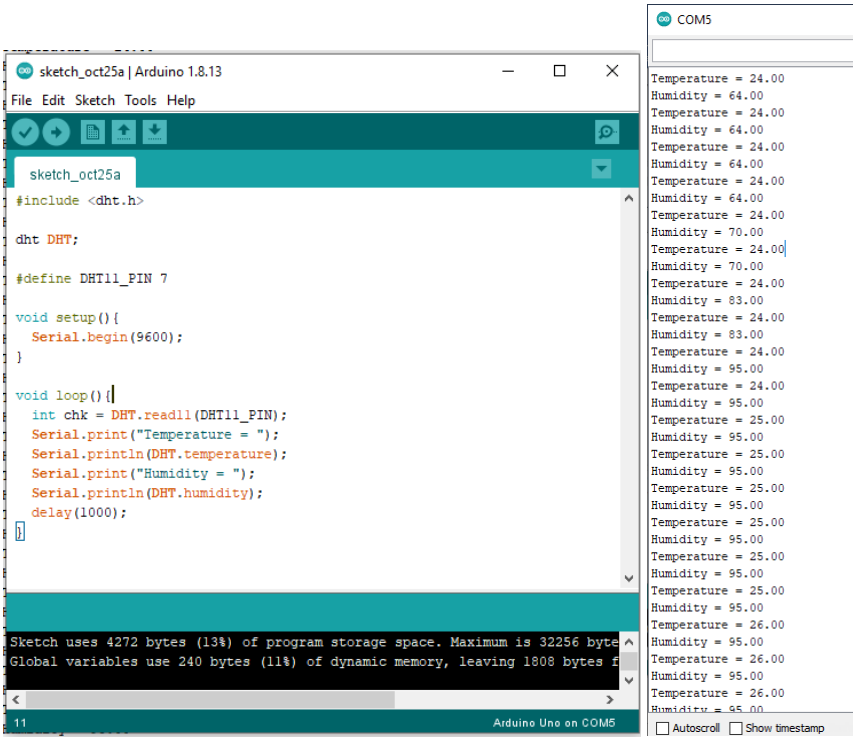
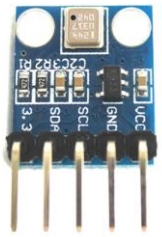


Figure 8 – Code for reading the SNS-DH11 sensor and the data recorded during the test

- BMP180 Digital Barometric Pressure Sensor:



Barometric pressure sensors measure the absolute pressure of the air around the sensor. This pressure varies with both the weather (temperature and humidity) and altitude. Depending on how we interpret the data, we can monitor changes in the weather, measure altitude, or any other tasks that require an accurate pressure reading.

Figure 9 - BMP180 Digital Barometric Pressure Sensor

For processing the data coming from this sensor I used a BMP180 library from Sparkfun.com called SFE_BMP180 Library (link [1] in the web references). This library is very useful cause it takes care of all the math for calculating the true temperature and pressure readings, as well as the math for calculating altitude. After downloading the library from the link provided at the end of my documentation, I opened the Arduino IDE and added the library by going to Sketch > Include Library > Add Library > select the downloaded file.

We can observe in Figure 10 the wiring I used for the testing of my BMP180 Digital Barometric Pressure sensor.

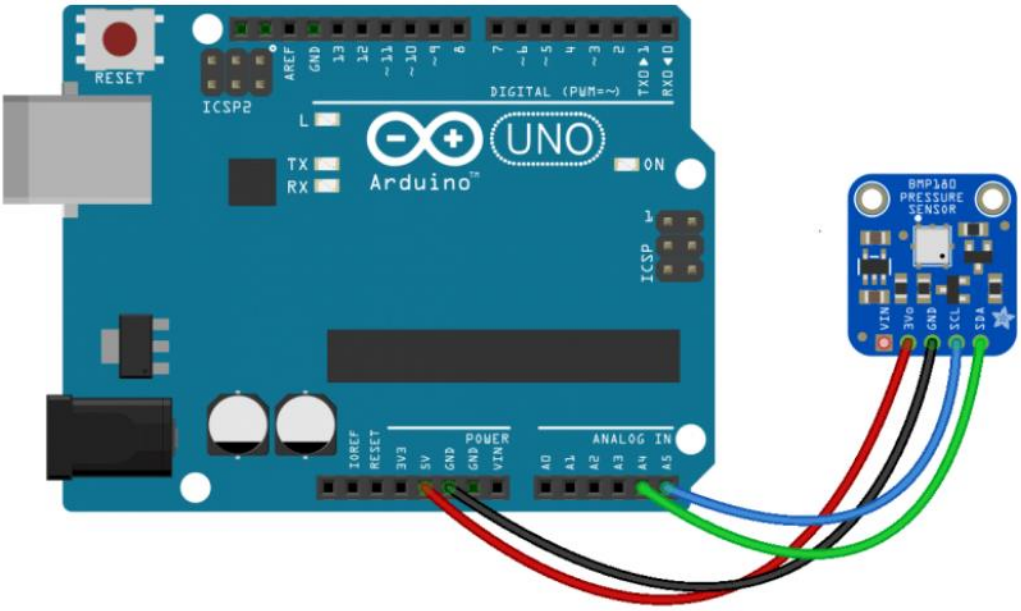


Figure 10 – wiring for testing the BMP180 Digital Barometric Pressure Sensor

For the coding part of the tester I used an example from https://www.circuitbasics.com/. The code is shown in Figure 11 along with the Serial Monitor where I displayed the information recorded by the sensor. After compiling the code, uploading it and opening up the Serial Monitor, I started to move the sensor up and down at different altitudes to test the barometric pressure (different altitude means different barometric pressure). I found out that the sensor is very sensitive to any change in the altitude but also for changes in the temperature – this feature I tested the same way I tested the other sensor described before – the SNS-DH11 Temperature and Humidity Sensor.

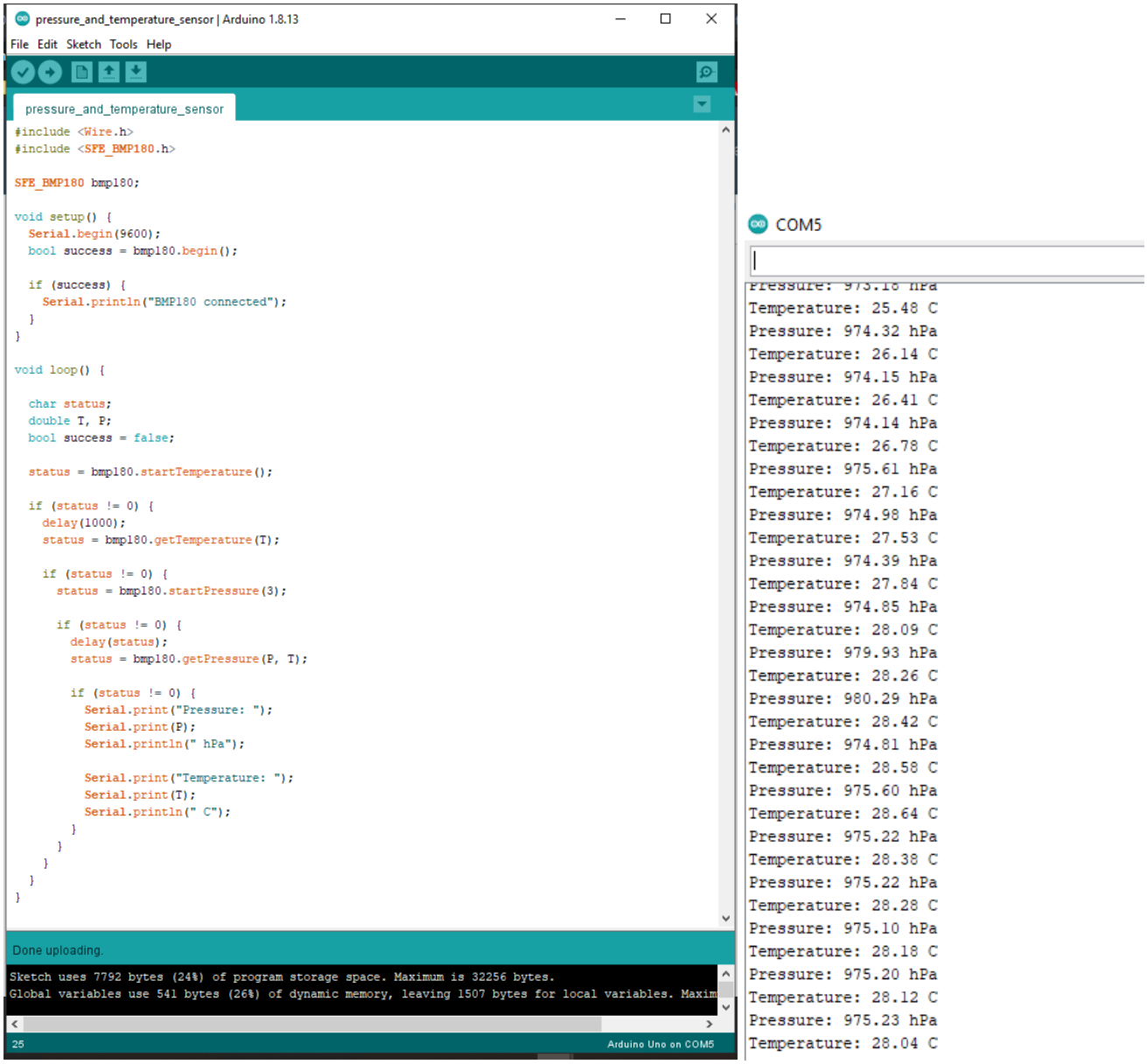


Figure 11 – code for testing the BMP180 Digital Barometric Pressure Sensor and the Serial Monitor results recorded when executing the code in the Arduino IDE

- ESP8266 Wi-Fi microchip NodeMCU v2:

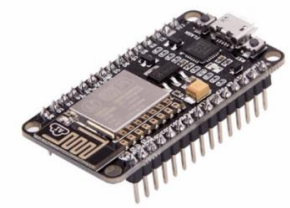


Figure 12 - ESP8266 Wi-Fi microchip NodeMCU v2

This microchip provides a full Wi-Fi networking solution, enabling users to set up a web server or web client with a separate processor or even standalone. It is also Arduino compatible, meaning you can program it using the Arduino IDE.

This component will be used in a later-stage of the development of my weather station and I'll explain its usage and the code and the wiring in later chapters.

Now that we got to know a little more about all the basic components that will be used for creating my weather station, we can go over the next step – connecting all the components together.

I tested this component by taking an example from the Internet. The link for the tutorial is the 3rd web reference link. The code is presented below, as well as the setup for this test.

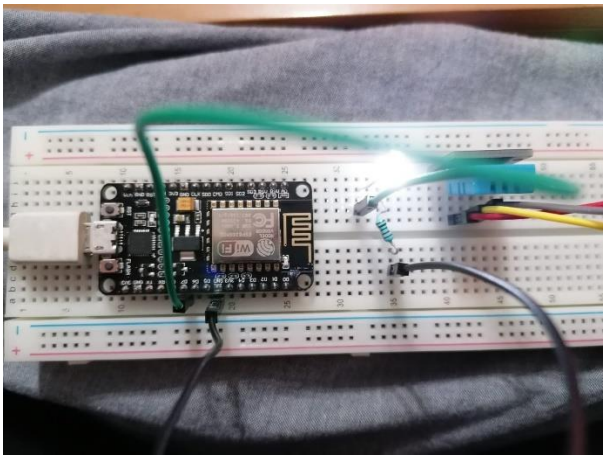


Figure 13 – Setup for testing the wi-Fi module

I connected a LED to pin D7 to test the connection to my wi-Fi. This LED will be set on high when the connection is properly set.

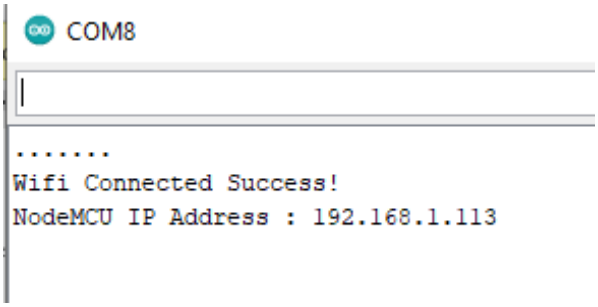


Figure 14 – Serial Monitor Response

The code I used for testing is the one presented below:

```
sketch_nov07a §
#include <ESP8266WiFi.h>

const char* ssid="Spectral";
const char* password = "[redacted]";


int ledPin = 13;

void setup() {

  pinMode(ledPin,OUTPUT);
  digitalWrite(ledPin,LOW);

  Serial.begin(115200);
  Serial.println();
  Serial.print("Wifi connecting to ");
  Serial.println( ssid );

  WiFi.begin(ssid,password);

  Serial.println();
  Serial.print("Connecting");

  while( WiFi.status() != WL_CONNECTED ){
      delay(500);
      Serial.print(".");
  }

  digitalWrite( ledPin , HIGH);
  Serial.println();

  Serial.println("Wifi Connected Success!");
  Serial.print("NodeMCU IP Address : ");
  Serial.println(WiFi.localIP() );

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 15 – Code for testing the ESP8266 wi-Fi module

- The database:

After creating the database, I connected the wi-Fi module and tested the connection by taking an example found at the 4th link in the web references section of this documentation.

I tested the connection to the database using this code and we can observe that the value of RandomVal changes simultaneously with the Serial Monitor value. As we can see in Figure 16, the communication works perfectly.

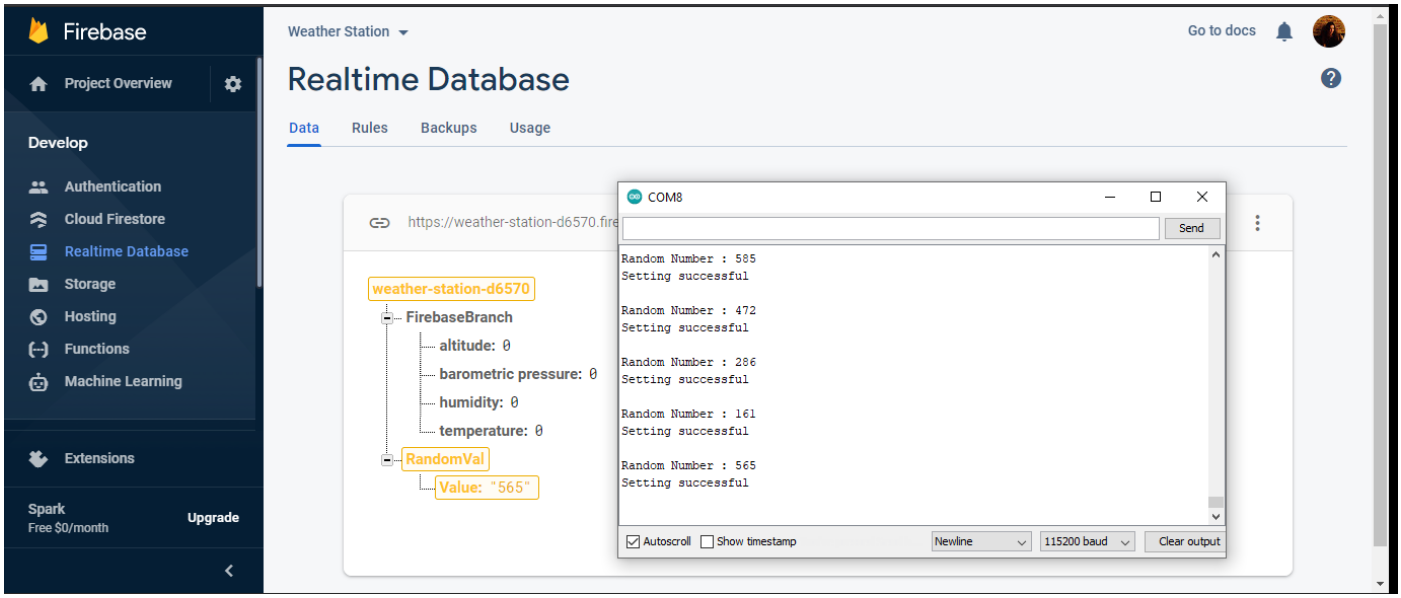


Figure 16 – Serial Monitor Response and Update recorded in the

4. Implementation and Testing

In this section, I will put all the components together to create a working weather station. I start by connecting the DHT11 temperature and humidity sensor to the wi-Fi module (the GND pin of the sensor is connected to the GND of the module, the power pin of the sensor to the 3.3V pin of the module and the data pin of the sensor is connected to the D4 pin of the module that is a digital pin), as we can see in Figure 17.

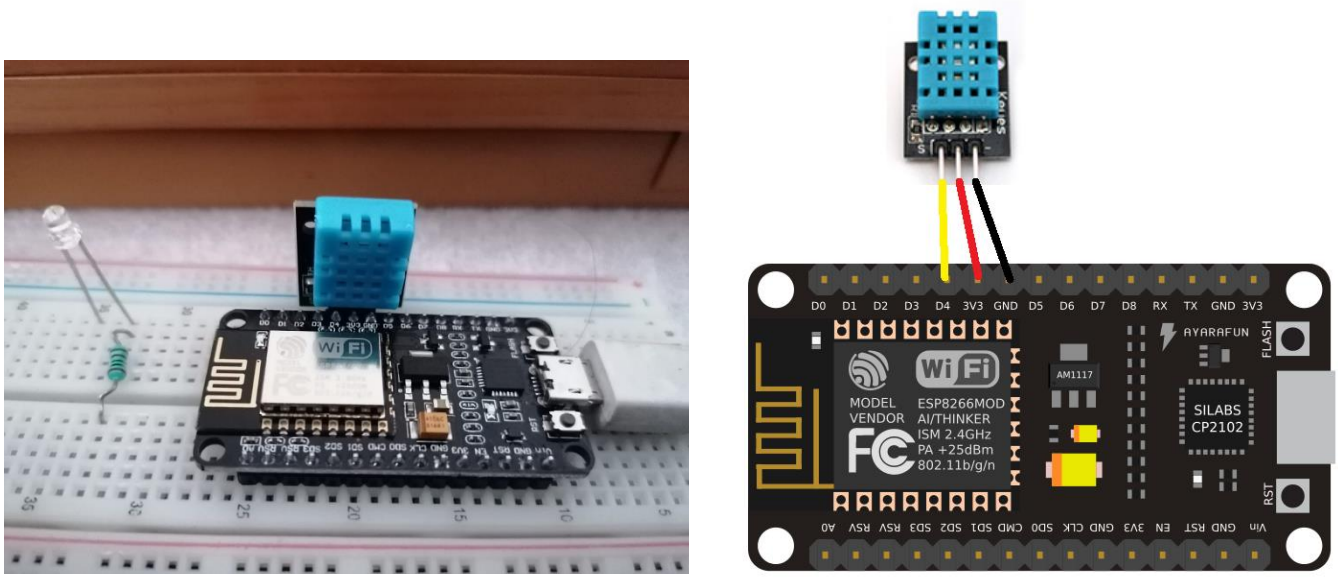


Figure 17 – Simple wiring for DHT11 sensor – Real and Schematic

After doing this setup, we try to send data received from the sensor to out firebase database. The code used is listed below. The code has comments included so that it is easier to understand and more readable.

In the next figures (Figure 18 and 19) we can see that the sensor sends data to the database and the database is updated once every 5 seconds. The sensor is a slow one, the reading of the temperature or humidity takes about 250 milliseconds and sensor readings may also be up to 2 seconds old (because of the sensor's transmission delay).

Figure 18 – Response of the database seen beside the Serial Monitor Window – temperature updated

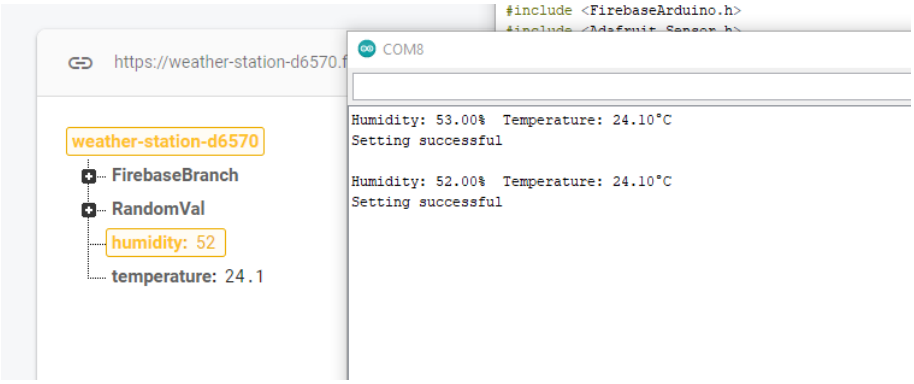


Figure 19 – Response of the database seen beside the Serial Monitor Window – humidity updated

In the next Figure (20) we can see how the sensor records a higher temperature and humidity by testing it as we did before – holding the sensor between my fingers. The data is transmitted to the database, and the values are updated accordingly.

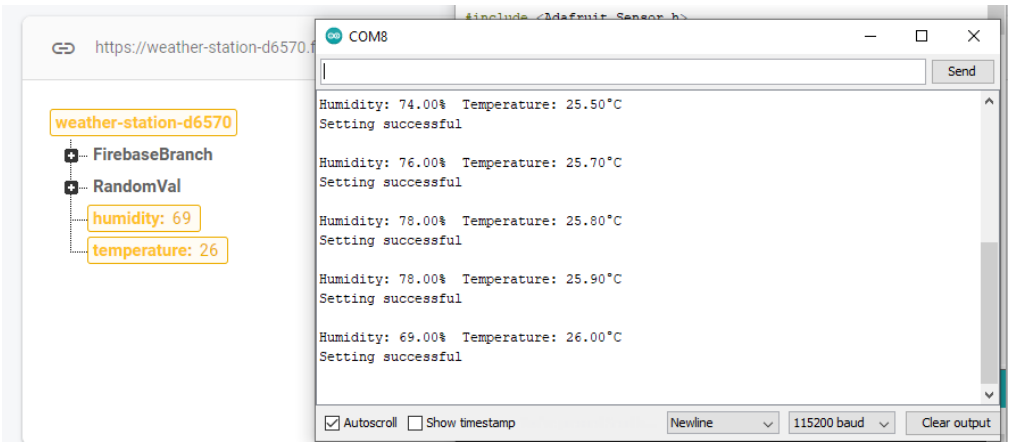


Figure 20 – Response of the database seen beside the Serial Monitor Window – hold between fingers test

Next, we connect the BMP180 sensor to our previous setup. The wiring is shown in Figure 21. I chose a schema that doesn't contain a breadboard for simplicity and increased readability.

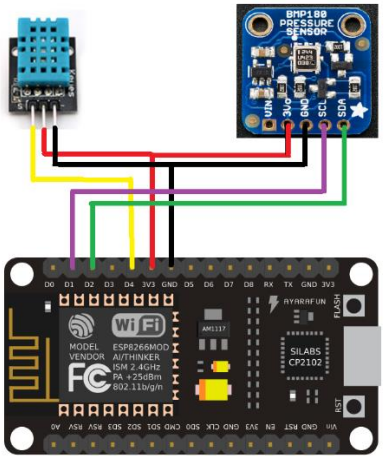


Figure 21 – Wiring for the two sensors

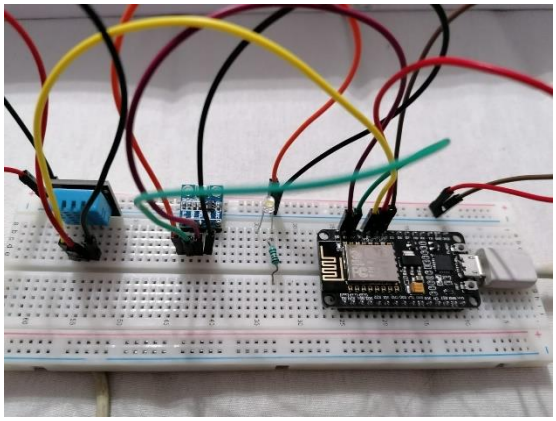


Figure 22 – My setup

The GPS position (measured in latitude and longitude, expressed in decimal degrees) will be taken from the Android device that runs the app. The altitude will be calculated using the GPS altitude function, when the coordinates have a corresponding altitude value, but the altitude returned by this function isn't always accurate and as I mentioned, not all coordinates have values assigned to them. Another way to measure the altitude is by looking at the barometric pressure and applying a formula to get the altitude, but this method is highly dependent on the weather conditions and is not accurate. The best way to get the altitude is by using a Query Service with a HTTP request, basically using the Elevation API developed by Google. This API enables us to query locations on earth and provides us elevation data. I wasn't able to use this service since I would need a billing account to access this kind of data from Google and I couldn't find a service that would offer the same accuracy of the data for the EU area, I only found one service that would return data for USA area. In conclusion, the option that I chose to go on with was the first one, the altitude provided by the GPS coordinate system. This is why for some GPS coordinates, the application doesn't retrieve information about the altitude.

For getting the current date and hour, the library NTPtimeESP [13] was used, the link to this library can be seen in the Web Reference part of the documentation.

The way I chose to handle the high amount of data recorded by the sensors (the sensors record one value / 5 seconds) is the following: I created an entry for each hour of the day and everyday, the last value recorded by the sensors for that specific hour gets saved forever in the database, but we can still see the current values of the sensors because the update happens once every 5 seconds or at the pressing of the Refresh button.

Now, we will discuss about the development of the Android application.

First, I had to watch some tutorials on how to use Android Studio; the links for these tutorials are listed in the Web Reference part of this documentation. After getting familiar with the constraints, I designed the interface for my application.

The first view is the main view, where we can see the current state of our sensors. The refresh button updates the text on the main view but it's not necessary to use it because the application updates these values constantly when changes appear in the database.

The other button (Week History button) sends us to the second view, where we can see the minims and maxims recorded in the past 6 days. For this to be possible, The structure of the database can be seen in Figure 24.

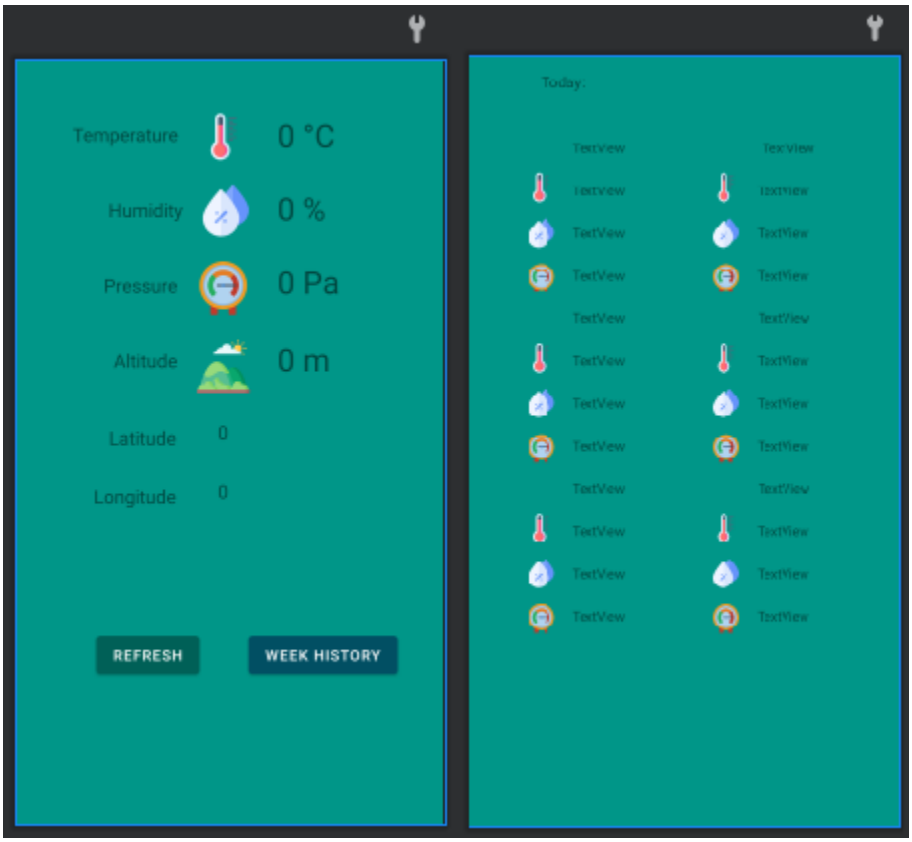


Figure 23 – Interface of the Android app

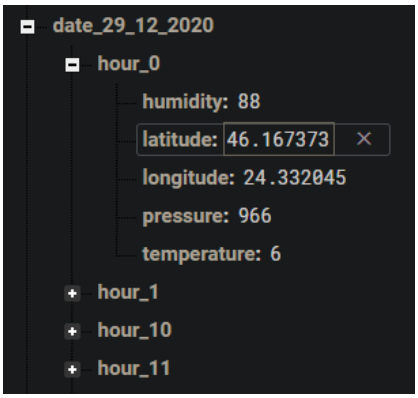


Figure 24 – Database Structure

The application is connected to the Firebase database and it needs to extract data from it. The update of the information is done in real-time is done automatically or by pressing the refresh button.

To connect to the application from an Android device, I had to go through a list of steps:

- Playing with the Android device Settings:

► We need to go to the *About Device* section, then enter the *Software Information* subsection and we need to find the *Build Number*. Then tap on the *Build Number* 7 times to enter **Developer Mode**. A message will appear on the screen when Developer Mode is activated.

► Now we need to enable **USB debugging** by going in the *Developer Option* subsection of the *System Information* section from the Settings.

Below, we can see the steps visualized using screenshots of my Android device.

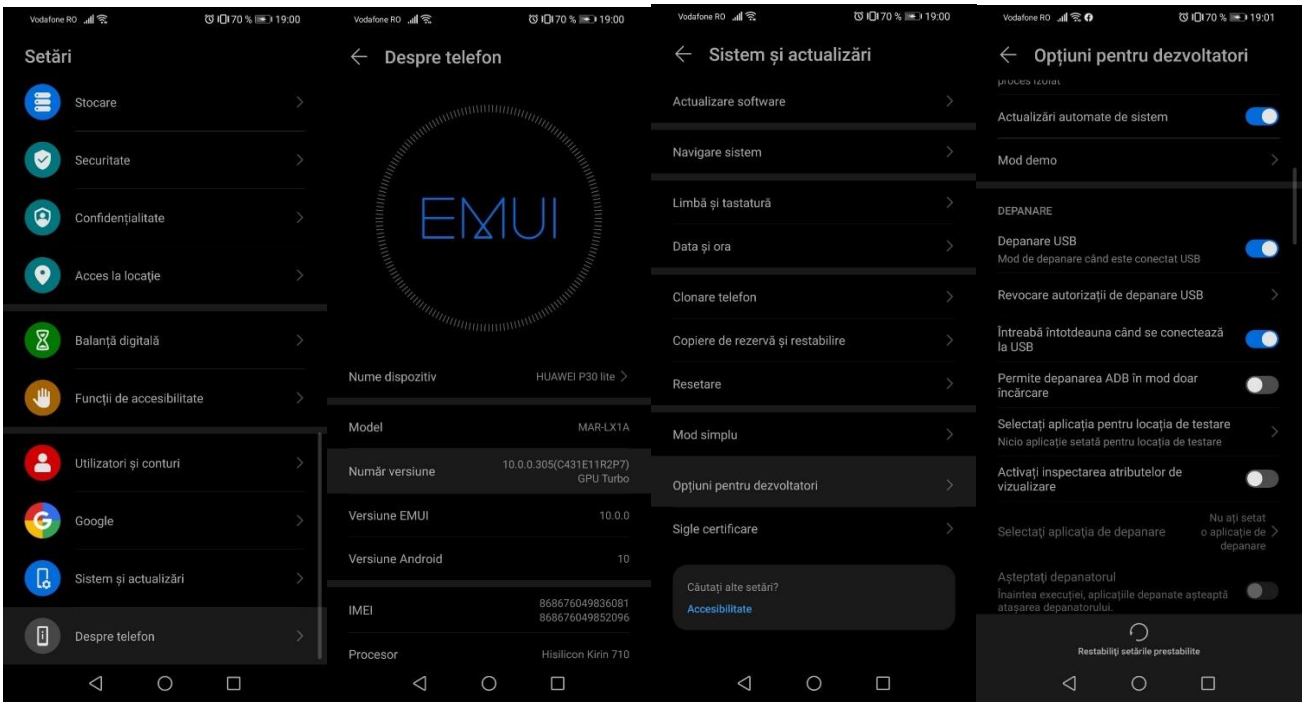


Figure 25 – Necessary steps to connect to the app from an Android device

- Installing the necessary extensions from *Android Studio*:
  - ► Go to *Tools* from the menu bar of Android Studio, and then *SDK Manager*.
  - ► Then, go to the *System Settings* section and then to the *Android SDK* subsection. Going in the *SDK Tools*, we need to find the *Google USB Driver* tool and install it if not installed.

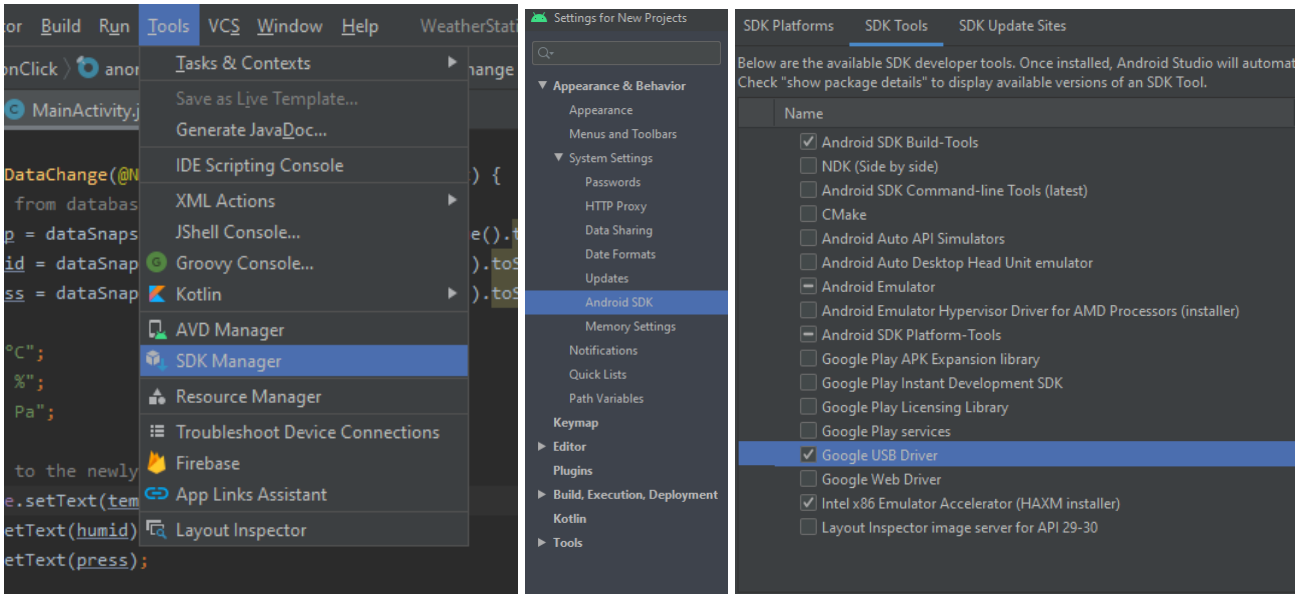


Figure 26 – Necessary steps from Android Studio

After going through these steps, we need to connect the Android device to the computer with an USB cable and make the changes so that the device will run the app. When we have the right device selected, we can run the app by pressing the ▶ button. I attached a photo of my device connected to the application, beside the database. I tested again the process of updating the data displayed by my app by holding one finger on the sensor. The data is correctly updated, in real-time.

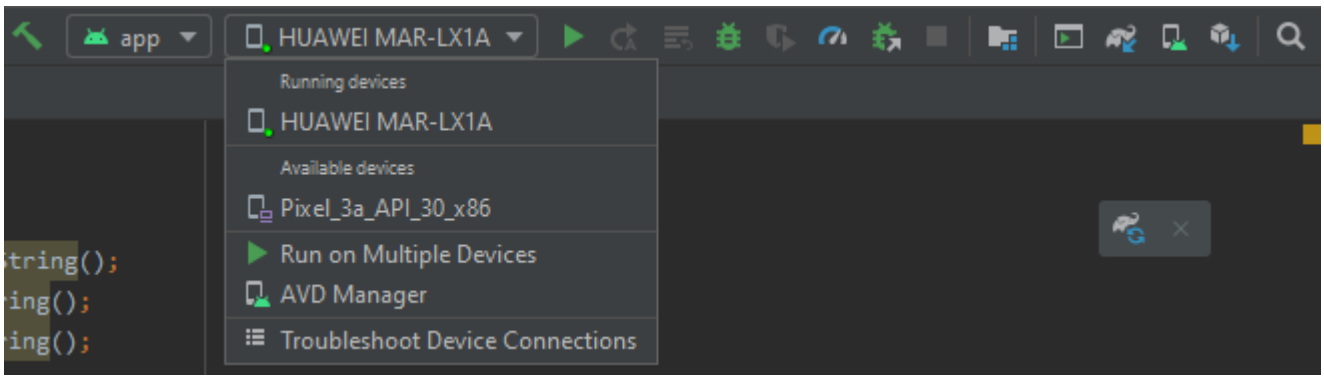


Figure 27 – Selecting the right Android device

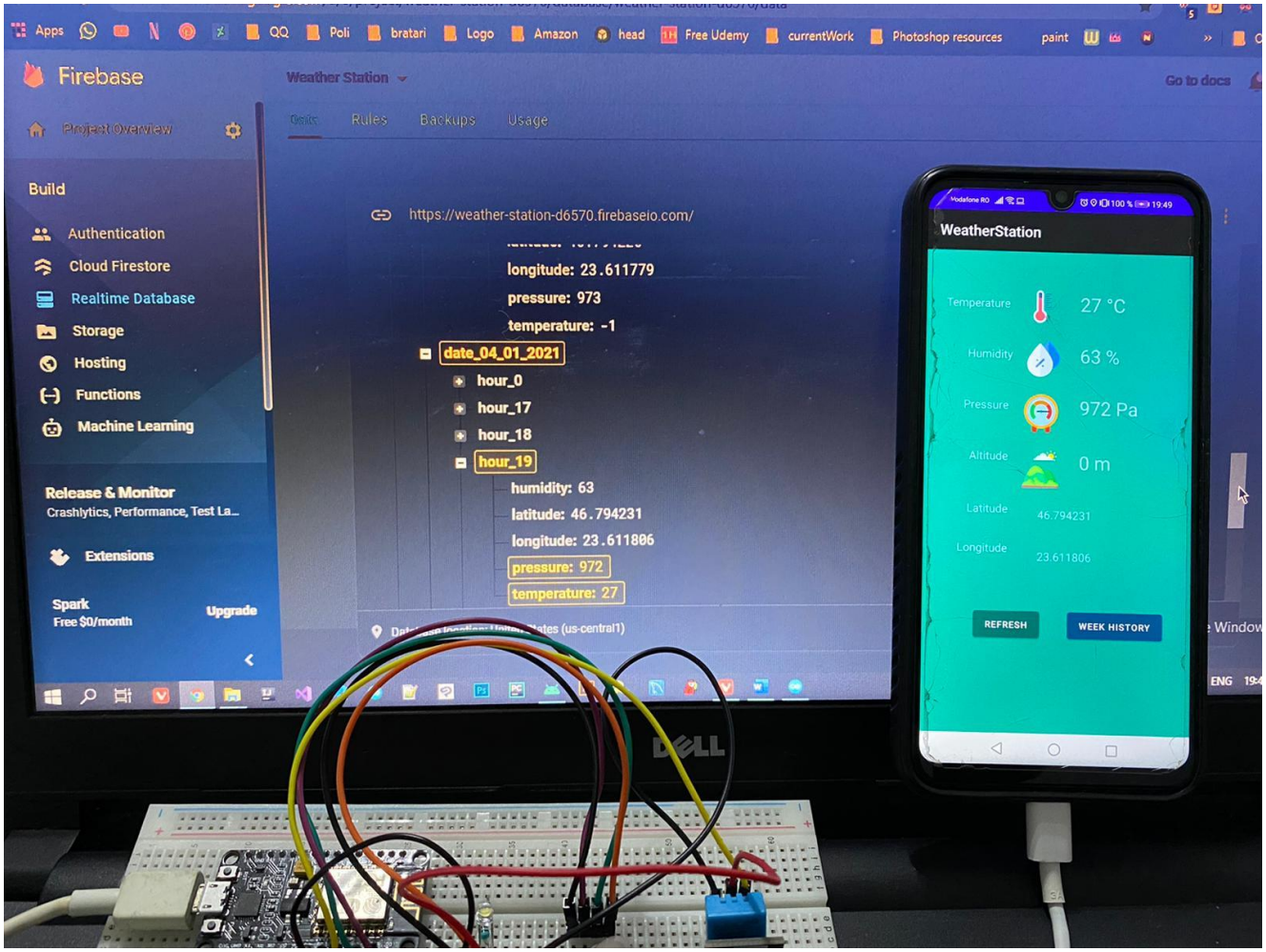


Figure 28 – My Android Device besides my circuit and the Firebase Database

5. Conclusions

The process of creating this application taught me the basics of Android as well as enhance the knowledge regarding Arduino programming and Java programming. The project was a success because now I know where to find the information I need to make complex projects, combining different IDE's to make a usefull application.

6. Bibliography

[1] John Nussey - Arduino for dummies, For Dummies, 2013

[2] S. Fitzgerald, M. Shiloh - Arduino Project Book, 2012

[3] Hans-Petter Halvorsen - Sensors and Actuators with Arduino,

http://learn.skillman.eu/pluginfile.php/774/mod_resource/content/0/7.%20Arduino-sensors-actuators.pdf, 2018

[4] Radu Pietraru, Alexandru Velicu - Elemente practice de bază în dezvoltarea sistemelor cu microprocesoare integrate, Editura Techno Media, 2014

[5] Radu Dănescu, Mircea Paul Mureșan, Răzvan Itu, Tiberiu Marița – Design with Microprocessors, UTPRESS, 2018

[6] Robojax -Leard Arduino in 30 minutes, https://www.youtube.com/watch?v=Mbb2xa1WcRM&feature=youtu.be, 2020

[7] Paul McWhorter - Arduino Tutorial Series on YouTube, https://www.youtube.com/watch?v=fJWR7dBuc18&list=PLGs0VKk2DiYw-L-RibttcvK-WBZm8WLEP, 2019

7. Web References

[1] - https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-/all

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

[3] - https://www.youtube.com/watch?v=TnWDlHpY56o

[4] - https://www.dfrobot.com/blog-910.html

[5] - https://www.instructables.com/Interface-DHT11-Humidity-Sensor-Using-NodeMCU/

[6] - https://desire.giesecke.tk/index.php/2018/01/30/esp32-dht11/

[7] - https://github.com/RobTillaart/Arduino/blob/master/libraries/DHTstable/dht.cpp

[8] - https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

[9] - https://www.youtube.com/watch?v=SRD6v0trzlA

[10] - https://www.youtube.com/watch?v=LpWhAz3e1sI

[11] - https://www.youtube.com/watch?v=4N4bCdyGcUc&t=516s

[12] - https://stackoverflow.com/questions/9361870/android-how-to-get-accurate-altitude

[13] - https://github.com/SensorsIot/NTPtimeESP

[14] - https://www.vogella.com/tutorials/JavaLibrary-OkHttp/article.html

[15] - https://stackoverflow.com/questions/1995998/get-altitude-by-longitude-and-latitude-in-android

[16] - https://ned.usgs.gov/epqs/

[17] - https://developers.google.com/maps/documentation/elevation/web-service-best-practices

[18] - https://lastminuteengineers.com/esp8266-ntp-server-date-time-tutorial/

[19] - https://www.tutorialspoint.com/how-can-i-make-a-horizontal-listview-in-android

[20] - https://developer.android.com/training/improving-layouts/reusing-layouts