Watering from Twitter

Embedded Systems - Final Report

Augusto Henriques	David Fernandes	João Silva	Pedro Jorge
up201605486	up201605791	up201606816	up201706520
DCC, FCUP	DCC, FCUP	DCC, FCUP	DCC, FCUP
Porto, Portugal	Porto, Portugal	Porto, Portugal	Porto, Portugal

June 6th, 2021

1 Introduction

The goal of this project is to monitor several plants and report their behaviour on the famous social network Twitter^{TM}.
The data will be measured using sensors connected to an Arduino which will then forward that information to a Raspberry Pi that tweets the plants' behavior, if the values read by the sensor are not within the threshold, the Arduino will take the initiative and activate/deactivate the actuator conform the values. The owner of the plants will then be able to act accordingly to that information and regulate the plant's light, humidity and temperature via smartphone (Android). If the owner wants to correct the status of the plant, the information goes through the Firebase and there the database will be updated. After the update, the data goes to the arduino where it will activate the actuators.

2 Arduino

For the arduino section we first used *Fritzing* to design our circuit with the components that were ordered. This software allowed to simulate not only the electrical components but also the breadboard and the position of the components in it.
Satisfied with the schematics, the next step would be to get the material and build the project.
As far as the arduinos itself, the group already had three arduinos, however, they were not working. Still, we were able to repair one of them (Arduino Leonardo) by burning the bootloader. With the Arduino Uno and Raspberry Pi that were provided by the educational institution, we had all the microcontrollers needed for *Watering from Twitter.*

2.1 Assembly

The assembling of the components was a easy process, the schematic previously done proved its usefulness and this step was very quick, only taking one day for one arduino.

2.2 Code

2.2.1 Testing

We first wanted to test the three sensors, since they were already on the breadboard the next step was to write the code. All the code was written in the Arduino's official IDE.
For the *void setup()* (section of the code that is executed only once the arduino is powered on) there was only two lines, the usual *Serial.begin()*, which sets the speed of the serial data transmission and a *sensors.begin()* which starts one of the libraries used by the DS18B20 (temperature sensor). On the *void loop()* (section repeated as long as the arduino is powered on) the value of both the humidity and light sensor are printed on the console by preforming an analog read on the arduino's pins they are attached to. When it comes to the temperature sensor it first makes a call to the *requestTemperatures()* method to instruct the component to perform a temperature conversion which then gets returned by *getTempCByIndex()*.
Besides a few adjustments like inverting the value of the humidity (initially the absence of water was reported as 1000 and 0 meant maximum moist) and some tests like using a lighter next to the temperature sensor it seemed like the sensors were working properly.

2.2.2 Setback

In order for the arduinos to connect to the internet, two Wifi shields were ordered. Both worked fine, however, they did not allow the microcontrollers to connect directly to the Firebase realtime database. All the libraries available online were not compatible with the Wifi shields.
In order to bypass this setback, the teacher provided a few solutions, some of them would require us to replace the shields with other piece of hardware or have a proxy server doing both the readings and updates on behalf of the arduino. We chose the last alternative and so we created a Linux Virtual Machine on the cloud using Microsoft Azure. Both PHP and Apache were installed on the VM and some PHP files were created to make the requests to the Firebase.

2.2.3 Communication with Firebase

At this point the arduino is capable of measuring the temperature, light and humidity of the plant. More code was added so that it reports all the values after given time (periodic updates) and checks if the given values fall outside the respective thresholds (emergency updates). But all that data must be fetched by the Raspberry Pi, and that is where Firebase comes in.
As said above, all the interaction between the arduino and Firebase's Realtime Database is made through a webserver who acts as a proxy.
The arduino has to be up to date with the last changes in the database made by the administrator of the plants, whether that be new threshold values or a manual activation of the actuators, but there is a problem, the microcontroller is not capable of multithreading so readings must be performed all the time, even if no change has been made at all.

2.3 Behaviour summary and end of section

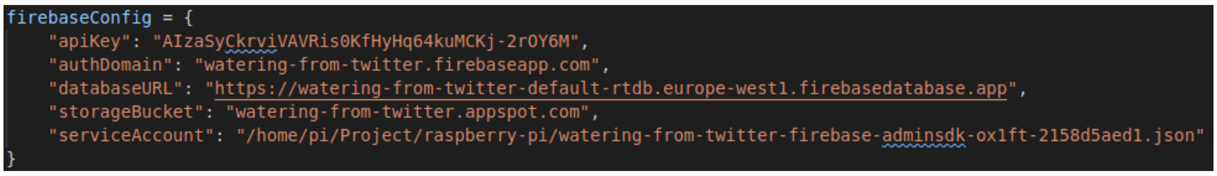
In short, after the arduino connects to the wireless network, it checks if it is time to do a periodic update (if it is the case, it performs a reading of all the sensors and sends it to the VM), fetches the most recent threshold values from the webserver, checks the webserver again to see if an actuator is to be turned on and compares all the sensors' values with thresholds to see if it needs to do an emergency update. This cycle is then repeated until the arduino is turned off.

3 Raspberry Pi

3.1 Set Up

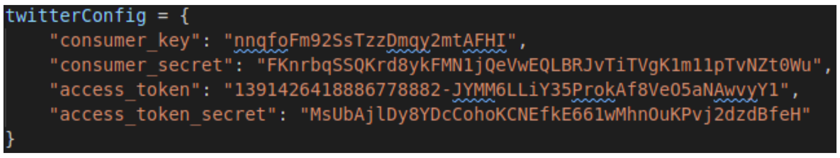
Setting up the Raspberry Pi was a pretty straightforward process, in accordance to the instructions displayed in the official website. We started by using a laptop with an SD card reader to install the Raspberry Pi OS in the SD card, which was done using the official Imager. We then put the SD card back into the Raspberry Pi and connected it to an external display through HDMI, an external keyboard through USB, a WiFi USB stick and a power supply outlet, which started the device. We finished the initial boot process, connected to a WiFi network and enabled SSH so that we could access the device without an external monitor and keyboard.

The next step was setting up the Python script to accomplish the goals we are going over next. This implied getting the necessary keys and data to access both the Firebase and Twitter API's and initializing the connection, which can be seen in the following images:



```
firebaseConfig = {
    "apiKey": "AIzaSyCkrviVAVRis0KfHyHq64kuMCKj-2rOY6M",
    "authDomain": "watering-from-twitter.firebaseapp.com",
    "databaseURL": "https://watering-from-twitter-default-rtdb.europe-west1.firebasedatabase.app",
    "storageBucket": "watering-from-twitter.appspot.com",
    "serviceAccount": "/home/pi/Project/raspberry-pi/watering-from-twitter-firebase-adminsdk-ox1ft-2158d5aed1.json"
}
```

Figure 1: Firebase access configuration



```
twitterConfig = {
    "consumer_key": "nnqfoFm92SsTzzDmqy2mtAFHI",
    "consumer_secret": "FKnrbqSSQKrd8ykFMN1jQeVwEQLBRJvTiTVgK1m11pTvNZt0Wu",
    "access_token": "1391426418886778882-JYMM6LLiY35ProkAf8VeO5aNAwvyY1",
    "access_token_secret": "MsUbAjlDy8YDcCohoKCNEfkE661wMhnOuKPvj2dzdBfeH"
}
```

Figure 2: Twitter access configuration

In order to easily interact with the two API's using Python we used the Pyrebase and Tweepy libraries.

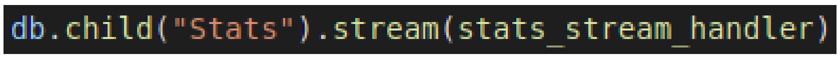
3.2 Scheduling periodic updates

The first goal for the Raspberry Pi was to set up periodic updates on the plant's status and report those updates to Twitter. This was accomplished by defining a function that retrieves the current stats for each plant from Firebase, turns that data into a custom string and updates Twitter's status. In order to repeat this we implemented a crontab task, using the "$ crontab -e" command with the "@hourly /usr/bin/python3 /home/pi/Project/raspberry-pi/main.py 1" configuration. This configuration runs our Python script and tells it to perform a periodic update.

3.3 Real-time database streaming

The second goal for the Raspberry Pi was to monitor real-time updates to the database and report them to Twitter in case these were emergency updates. Emergency updates correspond to any stat update that falls outside of the set thresholds.

In order to accomplish this task Firebase provides a service called *streaming* which reports real-time updates as events, which can easily be listened to using the Pyrebase library. The following image contains the line that initializes a stream on the "Stats" child of Fyrebase's database and deals with the messages using the stats_stream_handler function:



```
db.child("Stats").stream(stats_stream_handler)
```

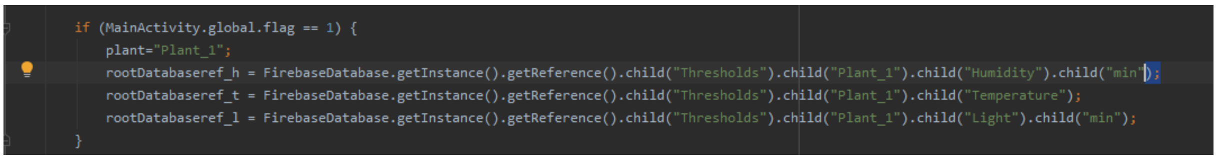
Figure 3: Initiating data streaming

The stats_stream_handler function receives a message containing the changed data in the database, requests the current threshold values to the same database, compares the values and makes an emergency update to Twitter in case the value fall outside of the thresholds.

4 Android

4.1 Set-Up

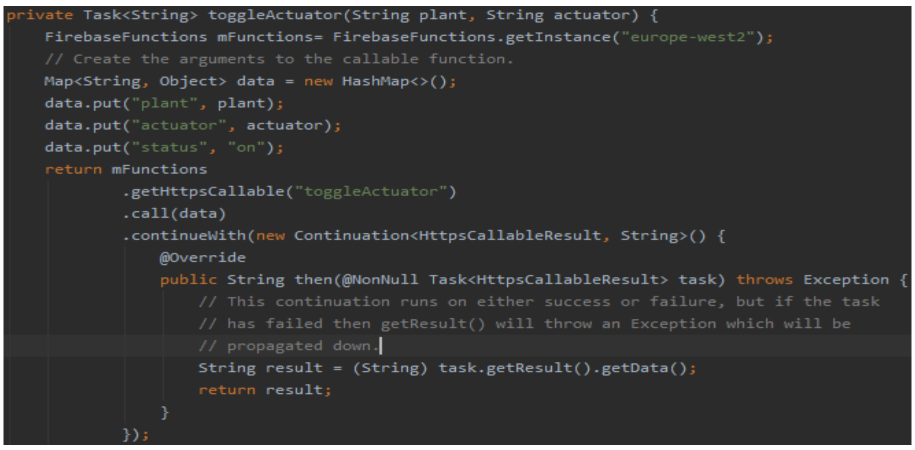
The first thing first we start to think about the mockups, the design of the app has been taken into account to provide a user-friendly experience that clearly shows the user all the information and operations they need. After building the design, we went for building code, where we stared with change buttons after pressing or edit text box where we can write our input. The next step was to setting up the data base servers. This implied getting the necessary keys and data to access Firebase and Webserver and initializing the connection, which can be seen in following print screen



```
if (MainActivity.global.flag == 1) {
    plant="Plant_1";
    rootDatabaseref_h = FirebaseDatabase.getInstance().getReference().child("Thresholds").child("Plant_1").child("Humidity").child("min");
    rootDatabaseref_t = FirebaseDatabase.getInstance().getReference().child("Thresholds").child("Plant_1").child("Temperature");
    rootDatabaseref_l = FirebaseDatabase.getInstance().getReference().child("Thresholds").child("Plant_1").child("Light").child("min");
}
```

Figure 4: Firebase access configuration

We use this line of code "rootDatabaserefh.setValue(bd2.doubleValue());"(this following code of example was to change the humidity of plant) to change values in Real Time database.



```
private Task<String> toggleActuator(String plant, String actuator) {
    FirebaseFunctions mFunctions= FirebaseFunctions.getInstance("europe-west2");
    // Create the arguments to the callable function.
    Map<String, Object> data = new HashMap<>();
    data.put("plant", plant);
    data.put("actuator", actuator);
    data.put("status", "on");
    return mFunctions
            .getHttpsCallable("toggleActuator")
            .call(data)
            .continueWith(new Continuation<HttpsCallableResult, String>() {
                @Override
                public String then(@NonNull Task<HttpsCallableResult> task) throws Exception {
                    // This continuation runs on either success or failure, but if the task
                    // has failed then getResult() will throw an Exception which will be
                    // propagated down.
                    String result = (String) task.getResult().getData();
                    return result;
                }
            });
```

Figure 5: Webserver access configuration

4.2 Problems/Solutions

We didn't have too much problems building this app, everything went well as expected, the only problem was in the smartphone provided by FCUP where the android didn't have the google services installed and for that we couldn't communicate with Webservice because one of the functions for communication were provide by google.functions. We had to use one of our personal smartphones to resolve this problem.

5 InterGitDoc

InterGitDoc is the fourth component, it comprises of the administration of 3 systems, an Intercommunication system (Firebase), a version control system (GitHub) and it's documentation (GitHub Wiki).

5.1 Intercommunication

For the intercommunication, we used Firebase's Realtime database to have permanency for the data, this way all components can have access to the same data in real time which leads to no coordination issues in the data being used by the components.
We also used Firebase functions written in nodejs to communicate between android and the proxy server for the Arduino.
Twitter is also part of the intercommunication as it's the one of the two feedback systems the user is going to use to know about the plants' state.

5.1.1 Realtime Database

For the Realtime database we simply added entries for each plants with the values they would report (Humidity, Temperature and Light), we did the same thing for the thresholds values and the actuators. Having this real time data base is what allows the seamless exchange of data to happen between out systems as they simply send queries and update the database whenever they want instead of relying on the other systems.

5.1.2 Firebase Functions

Since the Android was responsible for the change of thresholds and the triggering of the actuators in 2s max, we wrote two nodejs functions toggleActuator and changeThreshold (both available in "/firebase_functions/functions/index.js") , this way we can avoid having arduino make queries to the database every second which would lead to a big waste of bandwidth.

5.1.3 Twitter

Twitter was supposed to be the easiest setup as you simply need to apply for a developer account and get the API tokens, however, our account got wrongly restricted not once but twice by the same automated system that bans suspicious keys in bulk, luckily for us however, we were able to get the problems fixed in time.

5.2 Version Control

For version control we used GitHub, since every member of the group was working on individual parts of the project, it meant that there would be no overlap for the files. With that, we created a branch for every member where everyone worked on their respective parts.
During the development phase, we kept the repository private, but when all the parts of the project were finished we simply merged all 4 branches back into the main branch and made the repository public.

6 GitHub

The link to the GitHub repository can be found here.