Programming for Iot - Lab 4

Rafael N. Fontana

Contents

1 Introduction

1.1 Data format

In this lab, we will develop different kind of sensor simulators which employ REST and MQTT.
Remember to always use SenML as data format:

```
{
    "bn": "http://example.org/sensor1/",
    "e": [
            {
                "n": "temperature",
                "u": "Cel",
                "t": 1234,
                "v":22.5
            }
    ]
}
```

1.2 Hardware connection

To verify if the sensor works fine, you should connect the DHT 11 sensor. First, identify the corresponding model (a or b) of the DHT 11 sensor as presented in Fig. 1.

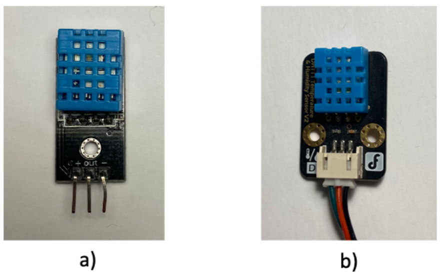

Figure 1: DHT 11 Models

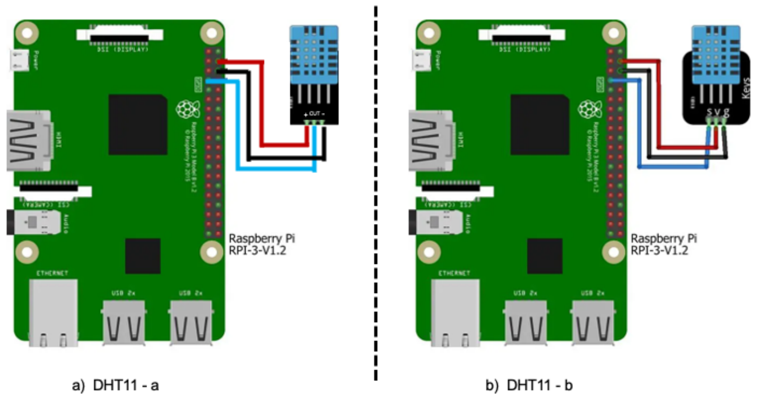

Figure 2: Connection of DHT 11 models

Then, according to the model, make the connection as shown in Fig. 2. (the connection on the **left** for **model a**, while the connection on the **right** for **model b**).

Finally, you should run the test file located in the "Documents" folder of the Raspberry in order to check that the sensor is working.

2 Exercices

2.1 Exercise 1

Using the Raspberry Pi develop a REST service for the temperature sensor. This sensor should provide a GET method to manage 3 different URIs:

- "/temperature" should return the measured temperature as a json in SenML format
- "/humidity" should return the measured humidity in the as a json in SenML format
- "/allSensor" should return the data from all the sensors as a json in SenML format

Develop the script on your PC and check they work correctly before deploying it on the Raspberry.

2.2 Exercise 2

Using the Raspberry Pi develop an MQTT publisher and MQTT subscriber The publisher should publish every 5 seconds a message in SenML format with the following information:

- "/temperature" should provide the measured temperature as a json in SenML format
- "/humidity" should provide the measured humidity in the as a json in SenML format
- "/allSensor" should provide the data from all the sensors as a json in SenML format

The subscriber should be able to receive all the messages from all the topics, and print them on screen in an user friendly way that indicates also the topic that provide that message.

Example: If the message below is received

```
{
    "bn": "rafa/sensor1/temperature",
    "e": [
            {
                "n": "temperature",
                "u": "Cel",
                "t": 1234,
                "v":22.5
            }
         ]
}
```

The publisher should print:

> *rafa/sensor1 measured a temperature of 22.5 Cel at the time 1234*

Develop the script on your PC and check they work correctly before deploying it on the Raspberry.

When you're ready, try to run the publisher on the Raspberry and the subscriber on your laptop to check if you developed a real IoT device.

2.3 Exercise 3

In most of the cases we want to perform some kind of processing on the data we receive to have more significant information to show to the user or to take actions. In order to do this, we need to develop a script able to receive data and process it according to our needs.

Therefore try to develop 2 post processing script as follow:

- The first one should ask (REST GET request) the temperature to the sensor developed in the first exercise every x seconds (choose x as you prefer, 3 should be a feasible value). Every 10 data points, it should calculate the average temperature and printing it on the screen.
- The second script should be an MQTT subscriber that will receive the humidity measurements from the sensor developed in the second exercise. Every 10 measurements it should publish a message with the average humidity at a topic of your choice.

For the second script you can test that everything works using a subscriber with a wildcard that prints all the messages received.

2.4 Exercise 4

In some cases, for testing reasons, it could be useful to have virtual sensors that can emulate as much as possible the behaviour of a real one.

Develop an MQTT publisher to emulate a heart-rate sensor that publishes random values in the range [55,180] every 5 seconds for 2 minutes.

Develop also an MQTT subscriber that receives these values, prints them on the screen and save them on a json file called *hrLog.json*. To generate the values, you can use one of the functions of the library *numpy* listed at this link. You can try with different functions to evaluate which one has the most realistic result. You can find a visualization of some of this generators here (you can even make your test there before writing your code)

2.5 Exercise 5

Develop an MQTT publisher to emulate a sensor of your choice that publishes random values in 3 possible ranges. For example, for a heart-rate sensor we could define 3 ranges like *resting,sport, and danger*. Then create a simple terminal client for this MQTT publisher to select the range to be used to send the data. Develop also an MQTT subscriber to receive these data and, in case of data in a warning range, provide some kind of feedback to the user.