Mobile and cyber-physical systems project specifications

Dalla Noce Niko, Ristori Alessandro

Master Degree in Computer science.

n.dallanoce@studenti.unipi.it, a.ristori5@studenti.unipi.it.

Mobile and cyber-physical systems, Academic Year: 2020/2021

Date: 10/05/2021

Team 8

`https://github.com/nikodallanoce/MCPS`

Abstract

The document presents the problem we wish to address, a possible use case with the specifications necessary for the implementation of the project; we describe how to implement our system of sensors using MQTT and how to store their data in a non-relation database like MongoDB. Then, we show how an user can access the stored data through a Telegram bot. Finally, we highlight some features that can improve our project.

Contents

1 Problem description

The current HACCP regulations[1] regarding the preservation of foodstuffs impose on every retailer the obligation to keep updated a daily register containing the temperatures of the cold rooms. In case of non-compliance, the merchant had to pay a very substantial penalty.

It is certainly convenient for our customer to automatically know various information on the refrigerators' state (temperature, humidity, etc.) without the need to personally check their state; moreover, having this information always at hand is necessary in an increasingly smart world where everything is accessible via smartphone.

For this purpose we want to introduce **S.E.B.** that stands for Smart Environment Bot, as a valid solution both to the individual merchant and companies that need to keep up to date on the status of their cold rooms and, consequently, on their products that could end up in bad condition if something could malfunction.

2 Use cases

As we mentioned in the **Problem description** paragraph, our project aims to provide an efficient and fast solution for storing foodstuffs in cold stores by applying iot and smart elements in harmony: to do this we will use sensors to detect the temperature and humidity of the individual refrigerators; such data will be saved at the end of each time window decided by the user (which can vary upon request) on a database that will make the data available to the bot to whom the users will interface by issuing specific commands.

A fundamental point of our work is to give customers the opportunity to always be updated on the status of their refrigerators; in this regard, users can set a temperature limit beyond which the bot will send an alert in the chat: this allows for faster management of any malfunctions. Furthermore, the bot will make it possible to obtain many informations such as the daily average temperature/humidity or the latest measurements made, so customers can always stay up to date on the status of their refrigerators.

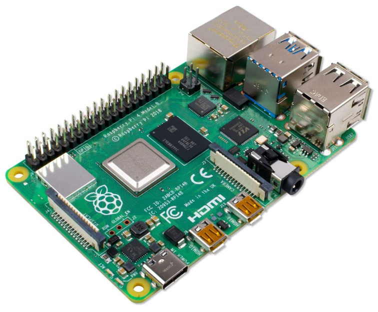
Finally, each user will always be able to know the parameters of their sensors through a simple command and modify them as they wish to adapt them to their needs.

[1] `https://www.haccproma.it/it/conservazione-degli-alimenti-le-giuste-temperature`

3 Requirements

We'll use the following software/hardware platforms:

- **Hardware** side:
  - **Raspberry Pi 4 Model B** for sensor installation and to start the detection by running *temperature.py*;
  - **DHT22 AM2302 AZDelivery** sensors for temperature and humidity detection.



(a) Raspberry Pi 4 Model B

(b) AZDelivery DHT22 AM2302

Figure 1: The Rapsberry and sensors used in the project.

- **Software** side:
  - **HiveMQ Cloud** for the MQTT broker;
  - **MongoDB Atlas** to store the samples provided by the sensors and informations about our customers;
  - **Telegram bot** to which the users will talk and ask for data by issuing commands;
  - **Heroku** to host and execute both the Telegram bot and our server;

(a) HiveMQ (b) Mongodb (c) Telegram (d) Heroku

Figure 2: All the platforms we used to store/run our project.

Raspberry runs Raspberry Pi Os 32-bit (Release 10) with Python 3.7; for the detection of sensor data we used the Python library called adafruit_dht.

The Telegram bot was developed on Python 3.8, using the python-telegram-bot library, which communicates with MongoDB Atlas using pymongo.

The server was developed in Java (JDK 16) and has the aim of receiving messages from the MQTT broker, saving them in the database and eventually sending the alerts. The server interfaces with HiveMQ using the HiveMQ MQTT Client library, available at the following link `https://github.com/hivemq/hivemq-mqtt-client`.
The database is managed through the official MongoDB library, available at the following address: `https://mongodb.github.io/mongo-java-driver/`. Finally, the alerts are sent to the Telegram bot using the Java Telegram Bot API library, hosted on GitHub at `https://github.com/pengrad/java-telegram-bot-api`.

The code written by us is available entirely on GitHub to the following link (server and bot are in the description provided by the README.md): `https://github.com/nikodallanoce/MCPS`.

4 Solution description

4.1 Raspberry

DHT22 sensors are installed on the Raspberry, for temperature and humidity monitoring. Each customer can buy one or more Raspberry and consequently will own several sensors; each sensor publishes its measurements in a topic which is in this format: "**customer** / CustomerName / Place / ThingToMesure" (for example "customer / Customer1 / warehouse / fridge1"), at each time interval defined in the database and it can be modified at any time.

Each sensor connects to the MQTT Broker via username and password due to the fact that HiveMQ Cloud only accept this kind of communications and to increase the system security as well, so it uses the 8883 port. Also, each sensor has a subscription to the topic "**communication**/ CustomerName/ Place/ ThingToMeasure", through which it receives informations from the server regarding how often send data. Once started, the Raspberry cannot go to sleep because this function has not been implemented on the Raspberry OS. However, we wanted to minimize the use of the processor in order not to waste energy unnecessarily, so we decided to associate a thread with each sensor, which goes into sleep after it has received information from the server regarding how often it sends the surveys. Going into detail, each thread performs the following operations, in order:

1. Publishes a detection in a topic;

2. Receives an indication about how often it should send the data from the server;
3. Sleeps.

In details, a thread handles the operations of two MQTT clients, a publisher and a subscriber. The first operates with Quality of Service equal to one, while the second with QoS equal to 2.

4.2 Server

The server connects the sensors to the database and eventually sends the alert messages and is subscribed to the topic "#"[2] with Quality of Service equal to 2. This choice adds overhead in the communications, but in this way it is not necessary to control the duplicates when the messages arrives into the server. Futhermore, both the Java server and the MQTT broker, in this scenario, are designed to work always on and plugged to the power supply.

There are two queues on the server, one called "topicPayload" and the other called "alerts". When the broker sends a message to the server, the topic and payload associated with the message are inserted into the first queue and then a thread takes care of retrieving each element of this list and inserts the data of the sensor detections into the database. In the event that a detection exceeds the threshold associated with a specific topic, the alert message is inserted on the "alerts" queue. This queue is managed by another thread, which takes care of sending all alerts to their respective customers that will receive the message from the bot in the Telegram chat. In order to keep a history of the alerts, these are then inserted into a collection of the database called "Alerts".

Finally, for each message received by the broker other than the "communication" type, the server communicates to the sensor how long it has to wait before sending a new survey, through a publish in the topic "communication/ Topic", where "Topic" is the topic in which the sensor executes the publish.

4.3 Database

By looking at schema in figure 3 is possible to have a general idea over our non-relational database works; the database was built adapting the relational database we had in mind.

[2]The simbol "#" indicates the subscription to each topic available in the MQTT broker.

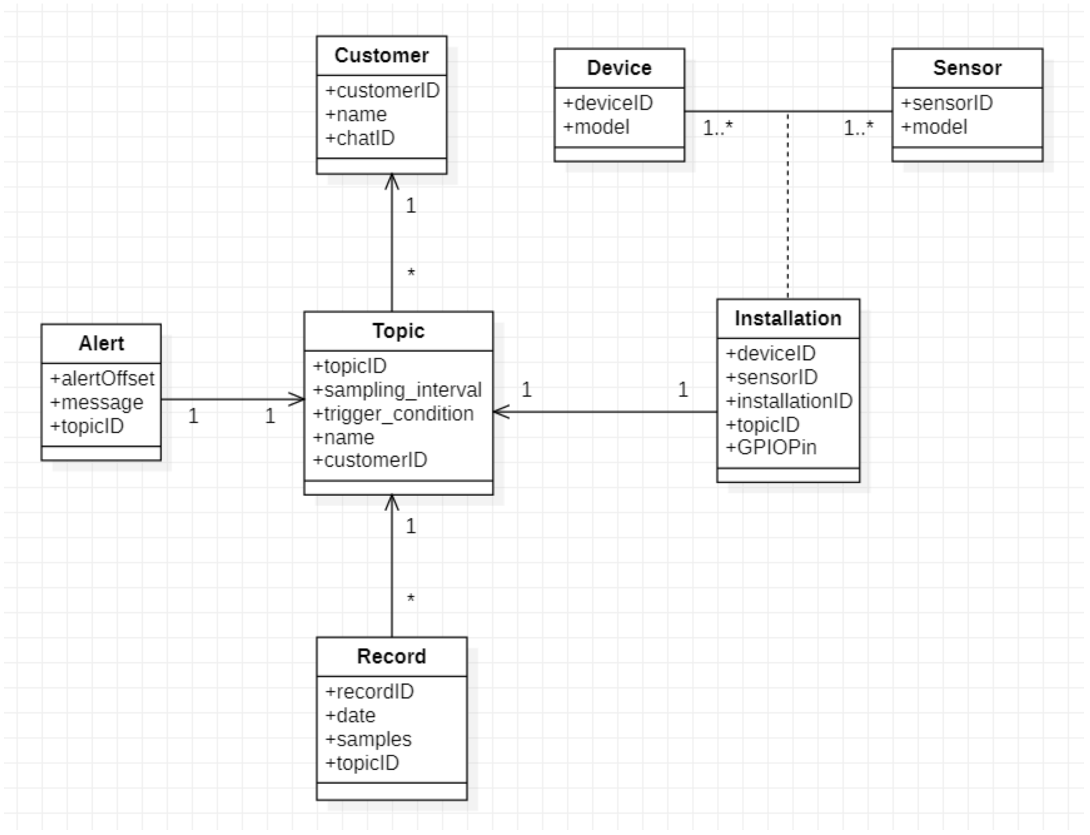


Figure 3: The UML schema of our software.

Below we can find a detailed description of the many collections and their fields that compose our database (the ID fields aren't listed since they are set automatically by Monogdb):

- **Customers**
  - *name*, customer's name, can be different from the username on telegram;
  - *chatID*, the telegram chat ID for a user or group.
- **Topics**
  - *sampling_interval*, the interval beetween one sample and the next one;
  - *trigger_condition*, the threshold at which the bot sends an alert if the temperature is higher than the value contained in this field;
  - *name*, it follows the rule CustomerName/topic/subtopic/....
- **Sensors**

  - *model*, the sensor's model, in our case a DHT22.

- **Devices**
  - *model*, the device's model, in our case a Raspeberry Pi 4 model B.

- **Alerts**
  - *alertOffset*, the interval that the server waits before sending another alert.
  - *message*, an array with the messages the server sends when an alert is sent. It's the alert history for a topic.

- **Records**
  - *date*, the date of the temperature and humidity detection;
  - *samples*, two arrays, one for the daily temperature and one for the daily humidity, each element of those arrays contains a value and a timestamp.

Let's focus more on the Records collection, we insert a new document daily (by topic), each containing, in addition to the ID and the reference date, two arrays to record temperatures and humidity. This solution was conceived as a meeting point between having a new document for each sensor detection and having a single document containing all the detections.

4.4 Telegram Bot

As we mentioned in the **Requirements** paragraph, most of the work is done through the use of the python-telegram-bot library which provides a high-level api for the management of the bot's commands, we also use the pymongo library for the querying the database on monogdb atlas. We have decided to provide the user the ability to send the following commands:

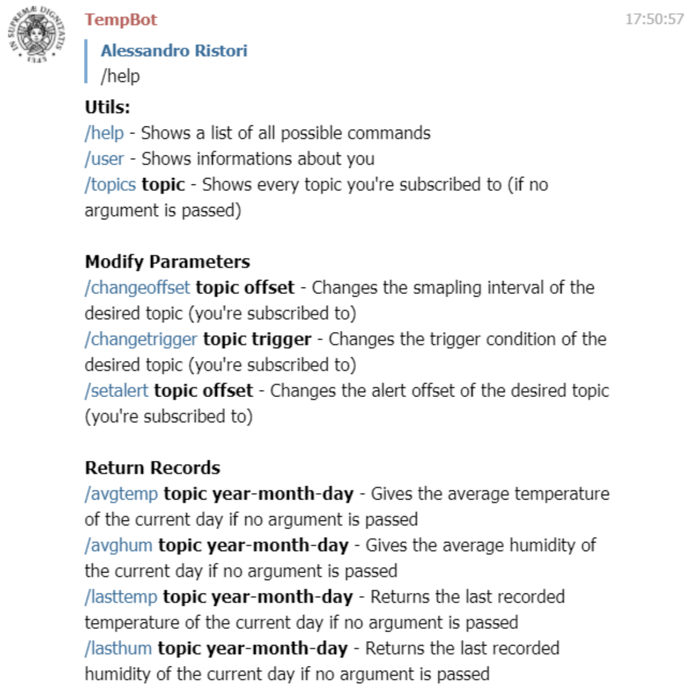
- **Utils**:
  - */help* - Shows a list of all possible commands;
  - */user* - Shows informations about the user;
  - */topics topic* - Shows every topic the user is subscribed to (if no argument is passed).

- **Modify Parameters**:
  - */changeoffset topic offset* - Changes the sampling interval of the desired topic (to which the user is subscribed to)

  - */changetrigger topic trigger* - Changes the trigger condition of the desired topic (to which the user is subscribed to)
  - */setalert topic offset* - Changes the alert offset of the desired topic (to which the user is subscribed to)

- **Return Records:**
  - */avgtemp topic year-month-day* - Gives the average temperature of the current day if no argument is passed
  - */avghum topic year-month-day* - Gives the average humidity of the current day if no argument is passed
  - */lasttemp topic year-month-day* - Returns the last recorded temperature of the current day if no argument is passed
  - */lasthum topic year-month-day* - Returns the last recorded humidity of the current day if no argument is passed

To communicate with the bot it is only necessary entering the command and then insert the various arguments for the desired operation. If the command entered is wrong an error message will be returned. In case we don't remember some commands, you can type "/" in the chat, which will show us all of them.



TempBot 17:50:57

Alessandro Ristori
/help

Utils:
/help - Shows a list of all possible commands
/user - Shows informations about you
/topics **topic** - Shows every topic you're subscribed to (if no argument is passed)

Modify Parameters
/changeoffset **topic offset** - Changes the smapling interval of the desired topic (you're subscribed to)
/changetrigger **topic trigger** - Changes the trigger condition of the desired topic (you're subscribed to)
/setalert **topic offset** - Changes the alert offset of the desired topic (you're subscribed to)

Return Records
/avgtemp **topic year-month-day** - Gives the average temperature of the current day if no argument is passed
/avghum **topic year-month-day** - Gives the average humidity of the current day if no argument is passed
/lasttemp **topic year-month-day** - Returns the last recorded temperature of the current day if no argument is passed
/lasthum **topic year-month-day** - Returns the last recorded humidity of the current day if no argument is passed

Figure 4: An example of the bot answering to a command, in this case */help*

The bot is hosted on Heroku, so it's always available every time someone needs to issue it commands, it can be reached to the following link `https://t.me/MCPSbot` if you're in a browser or you can search it by name on Telegram.

5 Planned demo and future work

We will show our project taking into account the scalability factor and the architecture behind our software, the aim of the presentation is to give our colleagues an overwiew of what we have done and how we did it by showing them a quick demo of our project.

During the presentation, after the necessary preparations and having explained the architecture behind our project, we will show how the data is retrieved and collected and how an alert message is sent to the chat. Later, we'll open a predisposed telegram chat to interface the bot with the aim to execute the many available commands: in particular we will show how to view the topics we are subscribed to and how to retrieve data, like the average temperatures and the latest recorded surveys.

We are extremely interested in expanding our work with the possibility of sending sms in parallel with the bot, in order to make everything more accessible even to the older groups of customers: In addition we could create an app that can give greater management concerning the sensors and a better data display, for example introducing graphs. In addition, the structure of the software, including the database, gives an high flexibility of application domains, ranging from home to business. In addition, we could integrate voice assistants, such as Amazon Alexa and Google Assistant, asking them directly the data that we want to know.