Ali HAIDAR
Ahmad ALIBRAHIM

Air Quality Control Project

CPS2 Project
January 19, 2020

1 Abstract Description

In this project we provide a data gathering scenario by exposing some measuring stations (Temperature, Pressure, Dust) in different places in Saint-Etienne. We configure the sensors to be able to transmit data instantaneously. The data is stored maintaining interoperability and it is highly interchangeable in a machine readable manner where it could be useful for any future use-cases such as simple and complex queries or even data analysis.

2 Objectives and Expected Results

The expected results are acquiring data from measurement stations that will be found in a database on the Territory Platform of Ecole des Mines de Sainte Etienne. A mobile application is used to configure the sensors and deploy them in a specific place in the city. In addition to that, a web-page is used to visualize these stored data.

The objectives of this project are the following:

- Store history information about air quality in different stations deployed at different places in the city.
- Connect our stations with the German project about air quality (German website)
- Visualize the history of measurements of a specific station.
- Visualize the current state of air quality on a specific station on a map.
- Deploy a new station in any place.
- Shift any station's location on run time.

3 Envisioned Use Cases

- **Deploy a station**: The user is able to deploy a station in his current location. The deployment of a station needs some requirements and procedure. These will be shown later.
- **Visualization of the current air quality:** The user can view the current air quality of different stations placed on a map.
- **Visualization of the history of a specific station:** The user can visualize the variation of the states of the stations with respect to time.

- **Provide machine readable data:** The user can use a distiller to get the embedded RDFa in HTML, where the gathered data can be uploaded to a triple store for future queries.

4 Functional and Technical Requirements

- Implementation and deployment of the Arduino code on ESP8266 board:
  1. SDS011 sensor (https://aqicn.org/sensor/sds011/) reads the values of PM2.5 and PM10 which are responsible for measuring the dust in the air.
  2. BMP280 sensor reads the temperature, pressure and altitude.
- 7 stations consist of the two sensors (BMP280 , SDS011) and an arduino board:
  - Each station sends data using HTTP to the territory platform.
- Implementation of a mobile application(Android) that will help in deploying new stations and connecting them to the server
- An mqtt server for the connection between the Arduino and mobile phone application.
- An online public REST API accessible from any network to store information about the states of the stations situated at different places in the city
- Implementation of a website to visualize the measurements using JavaScript, NodeJS, and HTML.

5 Technical Problems

- Unable to send data to German Website (https://luftdaten.info/en/construction-manual/) due to having different kinds of sensors
- Unable to read data from both sensors (temperature and dust) on the same Arduino board.
- We didn't have control on the server side (database, api,..) due to security issues on the territory platform.
- Every Arduino board should be configured to connect to a specific network(WIFI) depending on its location.
- Influxdb will crash after 3 months when pushing data at a rate of 5 seconds.
- Deployment of stations should be done in an asynchronous manner in order not to interfere the connection between the Arduino and the mobile application.

6 Planning of Realization

We provide the planning of the project that was supposed to be done, and the planning that was actually done. The planning is modified where we added some tasks and canceled others that didn't match the project's goals.

6.1 Planning at the beginning of the project

Task	Start Date	End Date	Desciption
Sensors Installation	19-11-2019	22-11-2019	Configure and deploy the air quality sensors then connect them to the network.
Sensors Synchronization	25-11-2019	29-11-2019	Transmit data from sensors to an online database.
Data in RDF	2-12-2019	4-12-2019	Model data in RDF and provide the mechanisim to insert in triple store
Data on influxDB	5-12-2019	9-12-2019	Transmit data from sensors to influxDB instantaneously
Deployment on the territory platform	10-12-2019	20-12-2019	Connect to the territory platform (influxdb, PHP scripts and triple store)
Visualization webpage	21-12-2019	10-01-2020	Visualize data transmitted by sensors in a web-page using the triple store and influx DB as databases.
Things description	13-1-2020	17-1-2020	Provide the things description for the sensors and test it on Qanswer

6.2 Planning at the end of the project

Task	Start Date	End Date	Desciption
Sensors Setup	19-11-2019	30-11-2019	Connect the Arduino boards with the dust and temperature sensors
Arduino Software	01-12-2019	15-01-2020	Develop the software that will be deployed in the arduino boards
Deployment on the territory platform	10-01-2020	16-01-2020	Connect to the territory platform (database, API)
Mobile application	10-01-2020	13-01-2020	Develop an application to deploy a new station
Visualization webpage	10-01-2020	14-01-2020	Develop a web page to visualize current states of the station

7 System Design

In figure 3 we show the architecture of the system. The three main parts are devided into those categories:

- Station deployment: The Arduino board of each station is configured publish its id on an MQTT broker, where the mobile app is developed so that it receives the published id. By this method, we can deploy a new sensor through the application that can acquire the current location, and push the station to the online database using the API.
- Map visualization: The web application communicates with a mapserver that is connected to the database where the information about the stations is stored, so that the map is always up to date.
- Data pushing: The stations and the web application uses the REST API to push and retrieve data from the server.

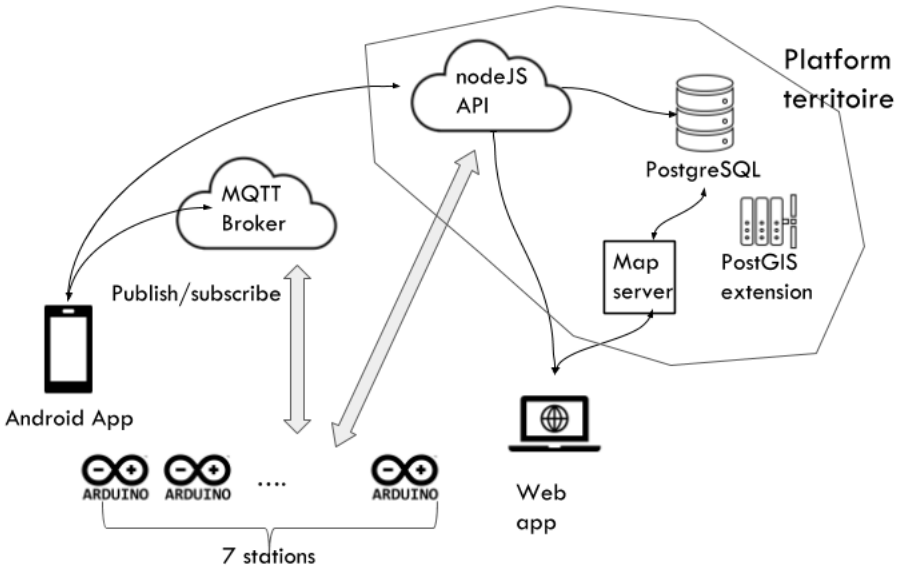

Figure 1: System design and data flow.

8 Implementation Specifications

8.1 Deploying Stations

An android mobile application is implemented in order to deploy a station in a specific location. The android application and the Arduino board will communicate with each others

by using the MQTT protocol. The android application being a subscriber will receive the mac address of the Arduino board which will be the publisher. After that the application will detect the current position of the mobile phone and display on the screen the 3 parameters:

- Station id
- Current Latitude
- Current Longitude

And finally, after telling the application to deploy this station, an HTTP POST request will be sent to https://territoire.emse.fr/applications/air-quality-control/station-info with the following json as payload:

```
{
    "station_id": "ab:59:88:8e:d:84",
    "latitude": 45.4218911,
    "longitude": 4.3861932
}
```

In order to deploy a station, the user should follow the following tutorial: **Materials**:

- An Arduino board "ESP8266"
- A dust sensor "SDS011"
- The mobile application "Deploy Station"
- Source of electricity

Procedure:

1. Connect the Arduino board with the dust sensor following the tutorial in the Luftdaten website.
2. Upload the software called "station.ino" that can be found in the following Gitlab private repository.
3. Plug the Arduino board in an electricity source.
4. Open the mobile application and wait till your latitude and longitude appear on the screen as seen below

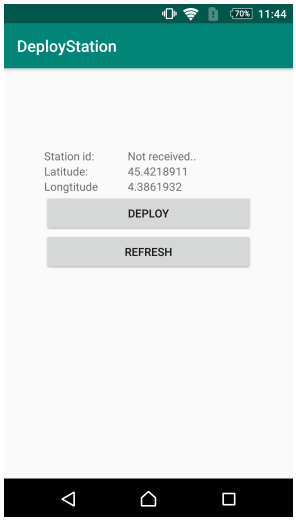

Figure 2: Deploy Station Mobile Application

5. Wait till the station id (board's mac address) is received and appeared on the screen

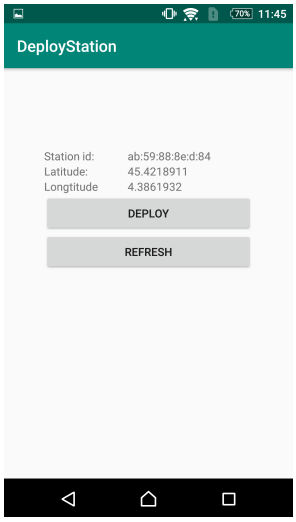

Figure 3: Deploy Station Mobile Application

6. Press the Deploy button

8.2 Pushing Data

After the station is deployed, it starts pushing the readings from the sensor to the online database using a REST API that is well described in the next section. Each arduino is

configured to use HTTPS POST requests by sending a specific payload (JSON) containing the unique ID of the sensor and the data readings. An example of what a single message looks like:

```
{
  "station_id":"ab:59:88:8e:d:84",
  "pressure":"0",
  "temperature":"0",
  "pm10":"2.6",
  "pm25":"1.7"
}
```

We assign the values (pressure,temperature) to zero due to the fact that we were not able to aquire this information but we make the possibility that we can later alter it whenever the sensor works in our configuration. The station is capable to keep pushing data as long as it is connected to wifi and still connected to electricity also.

8.3 Server and data storage

The REST API used by the stations to push data and ask about the states of the stations is implemented in nodeJS. We have a PostgreSQL database which is a normal relational database. In this database, we have two tables: **station_info** and **station_sensor**. The first table stores the static information about the stations (ID, poistion[lat,lon]) and the second table is connected to the first table using the unique ID of the station. It contains the readings of the stations at each time a station pushes data (assigned to a time stamp). The available features of this API are:

- GET https://territoire.emse.fr/applications/air-quality-control/station-info : get all the station information available.
- GET https://territoire.emse.fr/applications/air-quality-control/sensor-value : get the values pushed to the database so far for each station. It accepts 3 parameters (id, last, startingdate). If id is assigned it returns all the readings assigned to this specific station. Adding to it (last=1) will get us the last reading available. Finally, when giving a (startingdate=15/01/20 14:26:33) it will get all the readings starting from the specified date of the station having the ID specified.
- POST https://territoire.emse.fr/applications/air-quality-control/station-info (with a json to push station info). If the station exists, the station position will be updated else the station will be created.
- POST https://territoire.emse.fr/applications/air-quality-control/sensor-value (with a json to push sensor value)

All the data received from the API is in JSON format like in the example below, we issue a request on the sensor values (**https://territoire.emse.fr/applications/air-quality-control/sensor-value?id=ab:59:88:8e:d:84&last=1**) and the result:


```
{
    "station_id":"ab:59:88:8e:d:84",
    "pressure":"0","temperature":"0",
    "pm10":"2.6",
    "pm25":"1.7",
    "geom":"POINT(4.3867161 45.422193)",
    "currenttime":"16/01/20 21:26:53"
}
```


8.4 Data Visualization

Regarding the visualization, we embed in our web-application a map as explained in the system design so it retrieves its data from a mapserver that is connected to the same database where the sensors are pushing using the API. It is an OpenLayers map where the mapserver is building the layer to visualize each station on the map as seen in figure 4. By clicking on the marker, the information and the current reading of the station is displayed.

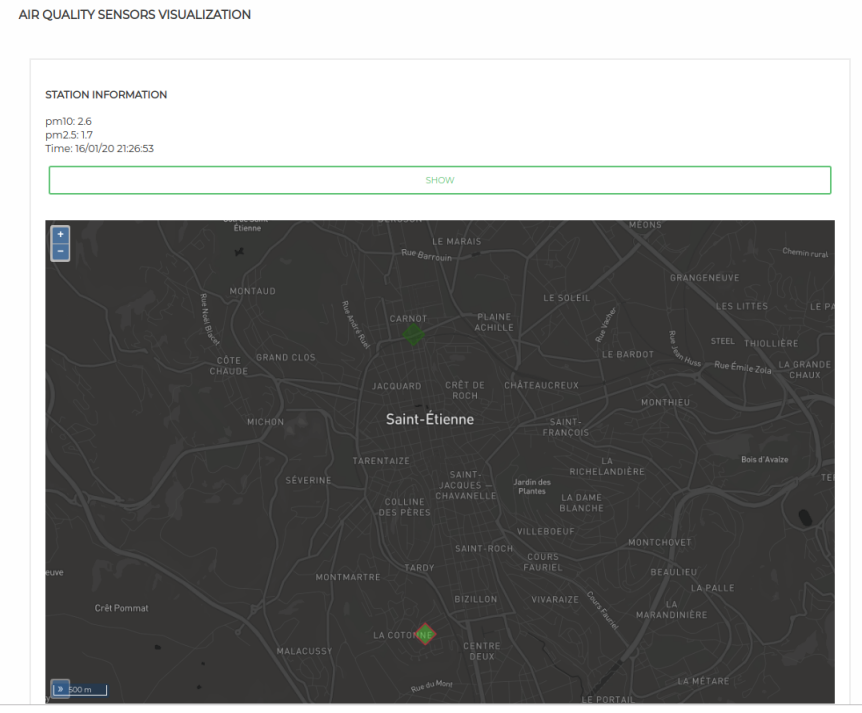

Figure 4: Map visualization of the current state of stations

We also provide a statistics page where we can track the flow of the data read by the sensors so far and visualize it on a graph versus time. The data is acquired using the API described above by specifying the start date and the station id so we can get all the readings. In figure 5 the two graphs show the flow of the values (pm2.5 and pm10) which are the data values being sent by a specific station.

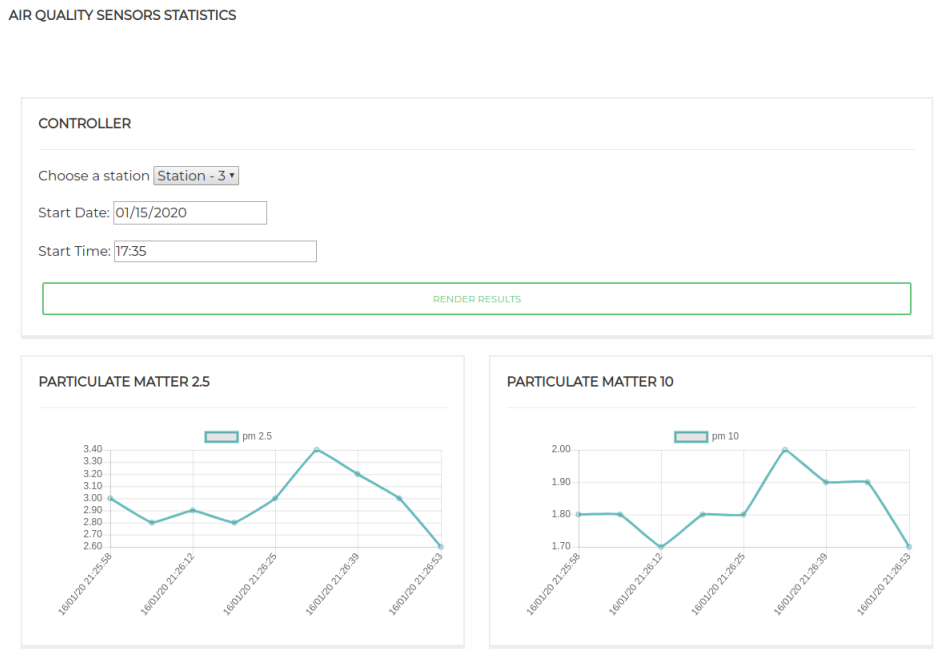

Figure 5: Statistics of a specific station

8.5 Data representation and interoperability

The last part is to represent the stations that are admissible in the context of this system is using embedded RDF data (rdf-a) where we use the SOSA and Wikidata ontologies to express those stations we have. The data is embedded in a page that shows the list of available stations with their current information (figure 6)

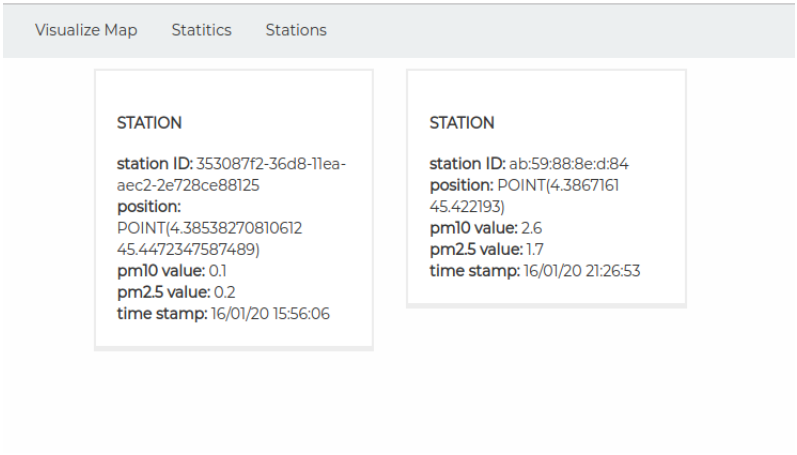

Figure 6: List of available stations

And the embedded data behind the HTML presented in figure 6 is rendered in the figure below where we use a dedicated distiller to extract this from the source code of the page :

```
@prefix ns1: <http://www.w3.org/ns/sosa/> .
@prefix ns2: <http://www.wikidata.org/prop/direct/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/observation_pm10> a ns1:Observation ;
    ns1:hasSimpleResult "0.1"^^xsd:float ;
    ns1:madeBySensor <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125> ;
    ns1:observedProperty <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/air_quality_pm10> ;
    ns1:resultTime "16/01/20 15:56:06 "^^xsd:dateTime .

<https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/observation_pm25> a ns1:Observation ;
    ns1:hasSimpleResult "0.2"^^xsd:float ;
    ns1:madeBySensor <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125> ;
    ns1:observedProperty <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/air_quality_pm25> .

<https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/observation_pm10> a ns1:Observation ;
    ns1:hasSimpleResult "2.6"^^xsd:float ;
    ns1:madeBySensor <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84> ;
    ns1:observedProperty <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/air_quality_pm10> ;
    ns1:resultTime "16/01/20 21:26:53 "^^xsd:dateTime .

<https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/observation_pm25> a ns1:Observation ;
    ns1:hasSimpleResult "1.7"^^xsd:float ;
    ns1:madeBySensor <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84> ;
    ns1:observedProperty <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/air_quality_pm25> .

<https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125> a ns1:Sensor ;
    ns1:observes <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/air_quality_pm10>,
        <https://territoire.emse.fr/applications/air-quality-control/353087f2-36d8-11ea-aec2-2e728ce88125/air_quality_pm25> ;
    ns2:P2326 " 353087f2-36d8-11ea-aec2-2e728ce88125 "@en ;
    ns2:P625 " POINT(4.38538270810612 45.4472347587489) "@en .

<https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84> a ns1:Sensor ;
    ns1:observes <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/air_quality_pm10>,
        <https://territoire.emse.fr/applications/air-quality-control/ab:59:88:8e:d:84/air_quality_pm25> ;
    ns2:P2326 " ab:59:88:8e:d:84 "@en ;
    ns2:P625 " POINT(4.3867161 45.422193) "@en .
```

Figure 7: RDF embedded data

The structure of the ontology used to represent the stations is presented in the figure below:

Figure 8: Ontology of the stations and their observations in the system.

9 Technologies Assigned to Courses

- **Semantic web:** The user can extract an RDF graph from the HTML page showing the list of cards by copying the source code to an RDF distiller.
- **IoT**: We used sensors that share messages with mobile applications using MQTT protocol.
- **Mobile Development**: We develop an android mobile application to deploy a station.
- **Web programming**: We implemented a web page to visualize the status of each station with their history.

10 Conclusion and Future Work

Based on the current content of the project we would like to add different kinds of sensors to the station so that it provides more information about the surrounding like (pressure,humidity, temperature etc..). Also we can imagine a scenario where those stations are in a building where they can control the system ventilation using a multi-agent approach

where each agent will be responsible for a specific station and can act on the environment re-actively and autonomously (open ventilation windows), and in this case there is no need for coordination. But in a future use-case we can use the multi-agent systems coordination to allow agents communication and coordination between each other, as they can cooperate together to reach a specific goal. To sum up our current system is extendable and we can integrate in it and append new stations following the same configuration, and visualize their status normally.

References

[1] RDFa 1.1 Distiller and Parser
https://www.w3.org/2012/pyRdfa/Overview.html

[2] Luftdaten website for stations and sensors specification
https://luftdaten.info/en/construction-manual/

[3] The Territory Platform
https://territoire.emse.fr/

[4] The temperature sensor
https://www.adafruit.com/product/2651