POLL THE AIR

TEAM: ARPIT BAJPAI (03667338), KARTHIK MATHIAZHAGAN (03669080)

PROJECT OVERVIEW

The goal of the project is to obtain the real-time pollution index of air for a geographical location. The motivation was to get the pollution value for a location which can later be integrated to the navigation services such as Google Maps to provide the least polluted route. The project was split into two major parts. The first part was in IOT domain where we collected the pollution value for a location. We used MQ135 gas sensor for obtaining the gas concentration and gy-gps6mv2 GPS module to get the location details. The second part was developing the web application where we created a REST web service as back-end infrastructure to store the pollution data.

NODE RED / RASPBERRY PI IMPLEMENTATION – IOT DOMAIN

We used raspberry pi-3 as our IOT device. This implementation gets the gas concentration value from the gas sensor along with the Geo location coordinates from GPS module and does a POST request to the back-end web service. We have used Node Red as our mash up tool to establish a flow between the sensor and GPS module and back-end service. The back-end web service expects the POST request to have the message payload in JSON format. Customized logic were written in node red function nodes to send payload in right format.

SENSORS USED

MQ 135 – This sensor was used to obtain the gas concentration value. The value obtained from this sensor (output of ADC) would range from 0 to 1024. Interpretation was done by keeping 0 as least value for pollution (no pollution) and 1024 as maximum value for pollution (highly polluted).

GPS Module – This module was used to obtain the current Geo coordinates of the IOT device. The values sent by the module includes Latitude, Longitude of the location along with the timestamp.

NODE RED NODES

ADC0 – This node was used to obtain the digital output from the analog to digital conversion chip (MCP 3008 I/P PDIP ADC) connected to raspberry digital pins.

Gpsd – This node was used to obtain the GPS coordinates from the GPS daemon running in raspberry pi.

Limit 1 msg/min – In order to avoid flooding the backend service with POST request, this node red node limits the POST request by doing one POST request every minute.

Switch – The switch node is a conditional node which is used to send the HTTP post request to backend server only when Latitude and Longitude details are available from the GPS sensor.

http request – This node red node is used to do the POST request to the back-end service. This node was configured with the back-end service URL along with the desired message format.

The below figure shows the node flow we used to achieve our IOT domain functionality.

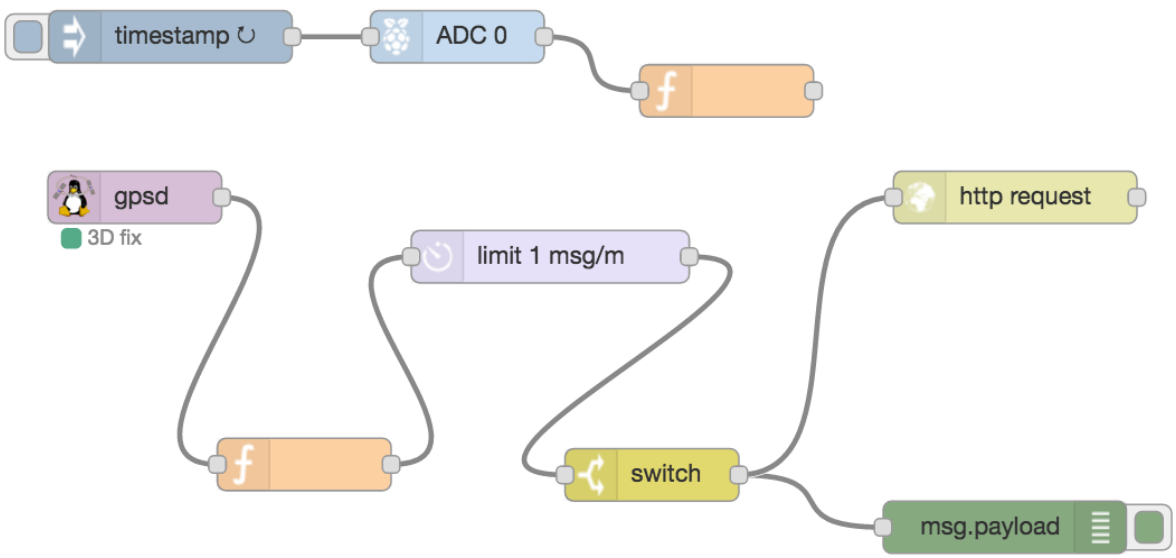

Fig 1.0

JSON Payload format: The back end web service expects the payload of every POST request in this format.

{"Latitude":"48.1826128","Longitude":"11.613259900000003","Pollution":"100"}

CHALLENGES

1. Our major challenge was with MQ 135 sensor as the sensor has support only for analog output and we do not have analog input pins in raspberry pi. We solved this problem by using MCP 3008 ADC convertor to convert the analog value of sensor to digital output.

2. Writing customized functions using node red function modules. The requirement was to consume inputs from two different sources (not arriving at the same time-as node is asynchronous) before sending an output from the function to the backend service. To solve this problem, we used context variables through which data can be stored and exchanged between several function invocations.

3. Getting GPS sensor working was not a trivial task. Several configurations were required that included freeing up the UART port which is by default used for logins and setting up the correct BAUD rate along with the changing the system files.

ODATA REST WEB SERVICE – WEB APP DOMAIN

We developed a back end restful web service to handle the POST request from IOT domain. This implementation takes care of handling the POST request, extracting the payload value and creating the resources in the back end database. We have used ODATA protocol to implement our web service in a REST full way as its very flexible and easy to understand. It provides a common interface to the outside world by providing the web service meta data file. Using this meta data file, the client could identify the resources and their operations present in the server.

Apart from the web service we have also developed a front end UI using SAP UI5 framework to display the pollution values. This web front end makes use of the Meta data file provided by the web service for OData operations.

TECHNOLOGIES USED

Apache Olingo OData v2.0: Olingo OData API was used to implement the OData operations in the server side.

Java persistence using JPA: For each POST request, a new instance of a "Pollutiondata" class would be created by the server. For persisting the created resources in the back-end database, Java JPA v1.0 APIs were used. We used basic maven OData JPA archetype and imported it to eclipse to start developing the service. The web service could be accessed from below URL,

https://poll0s0016680479trial.hanatrial.ondemand.com/PollTheAir

The below figure shows the JPA data model.

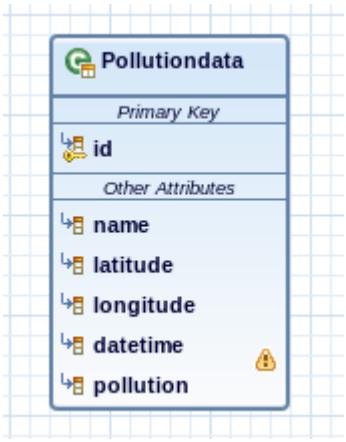

Fig 1.1

SAP UI5 Fiori style web application: The web front end was developed using SAP UI5 framework using master detail layout. The web front end could be accessed from below URL,

https://polltheairproj-p1942369597trial.dispatcher.hanatrial.ondemand.com/?hc_reset

SAP HANA Cloud Platform:

We have used SAP HANA cloud platform for hosting both our back end OData service and web front end. SAP provides free trial account with basic access to these features. This includes hosting only one web service at a time and many HTML 5 applications.

The OData web service combined with java JPA is deployed as a java application (war file) into the SAP HANA cloud platform and the web front end is deployed as simple HTML 5 application.

Google GMAP JavaScript and Web Service API:

In order to give a good appealing web front-end we have used Google's map java script API to import google maps into the front end. For each pollution value, Google map would point to the exact location where the pollution data was obtained. We used the latitude and longitude properties present in “Pollutiondata” entity and supplied them as input to Google GeoCoder API to get the exact address value. The same Geocoder API was used as a web service by our back end OData web service to obtain the address value for each POST request.

CHALLENGES

1. Writing customized OData POST operations: Since the sensors only send 'Latitude', 'Longitude' and 'Pollution' fields and the data model has extra fields, we must handle the POST request in a customized way by overriding the library methods.

2. Using Google Maps Java Script API:

Using Google Map Java Script API was a big challenge as it has to be used with SAP UI5 framework. The framework works with model view controller design pattern and we had to bind the returned address values for Geo coordinates to the UI5 controls.

3. Using Google Map Geo Coder API in OData service to obtain the address values for each POST request was also challenging.

4. Designing of back end JPA model.

PRACTICAL APPLICATIONS

Since now we have a RESTFULL web service providing real time pollution data for a location.

- This could be integrated with navigation system of locomotives to suggest routes which is less polluted.
- Google maps to suggest not only shortest routes between source and destination but also less polluted routes.
- For research purposes to analyze the pollution of an area over a period of time.

GIT REPOSITORIES

- Node Red Project Git repository can be found here.
- OData Web Service and Web front end UI Git repository can be found here.