An IoT-based Air Quality monitoring system

Andrea Di Ubaldo, Riccardo Baratin
Project of Internet of Things A.A 2021/2022
andrea.diubaldo@studio.unibo.it, riccardo.baratin@studio.unibo.it
0001004662, 0001007823

Abstract

This report shows the infrastructure, business logic, and functioning of an Indoor Air Quality Monitoring System based on ESP-32 device, DHT-22 and MQ-2 sensors. In the first section of this report, we introduce some useful contexts where we can use this system; in the second and third sections, we will discuss the project's architecture and implementation from an analytical point of view. In the latest, we analyze results on performance related to the project.

1 Introduction

The Internet of Things has evolved due to the convergence of multiple technologies, like ubiquitous computing and commodity sensors, convenient wireless network connection and machine learning technique. Nowadays it is applied in many environments: Agriculture, Urban and Smart Home. Generally, with the high quality of life standard and the continuous increment in the domestic time living, the air quality of our home environment has become very important. The proposed project focused in building a system to monitor the presence of extended gas concentration, indoor temperature and humidity. These parameters are given by the adoption of two sensors: DHT-22 and MQ-2; these sensors can be connected to an ESP-32 board present in the environment and communicate in an interactive system, which can calculate additional information like the Air Quality Index(a gas concentration indicator) or the RSS (received network signal strength), and modify setup parameters like the frequency of sensors measurement or the value of minimum and maximum gas concentration as AQI interval. Data acquired by sensors are sent to a proxy server that can forward communication between databases, monitoring dashboard, forecasting servers, and devices. This server control state sessions composed of business logic and external services values, metadata and monitored measurement states. Our implementation take care of data management with Influx Databases; it maintains a connection with time series of monitored measurements and update them with new sensor's data acquisition(in this case Influx DBs are called buckets). Influx databases tag each time series by GPS position and identification code of the devices, and can show the behaviour of the data over the time. Moreover Influx buckets can be connected to Grafana Dashboard in order to plot data produced by sensors and the ones predicted by Data forecasting algorithms, in order to compare the two data types. In addition we built a Telegram Bot service as a notification service in case of Grafana alerts that can occur based on AQI value variation. Moreover the Bot can return medium value of indoor and outdoor temperature, humidity, AQI, WiFi RSS and gas concentration by querying the corresponding Influx buckets.

2 Project's Architecture

2.1 Sensors

The devices chosen for the data acquisition process were the ESP-32 boards, in addition to the temperature and the gas sensor, that is the DHT22 and MQ-2 sensor respectively. The overall architecture is designed in order to allow a communication between the board and the proxy server. We exploited the two following protocols:

- **MQTT**: It is a lightweight and publish-subscribe-based network protocol. It belongs to the Iot protocols class, infact it is designed for connections between constrained devices located in remote environments. It must run over an ordered, lossless, and bi-directional protocol, like TCP/IP.
- **HTTP**: This is the mainstream application-layer protocol on the Internet. Although it cannot be considered an IoT-native protocol, its HTTP/2 version has a binary compact header that makes it more suitable in an IoT context. Like MQTT, HTTP is a TCP/IP based protocol.

As we transfer data by two different protocols, we have to handle the connectivity for two different standard; when it comes for the MQTT architecture, we know that the sensor must publish on a broker server its data at specific routes, and all devices connected to the broker and subscribed to those routes, will receive all sensor data, since the broker will deliver them to clients. So the sensor must think of collecting and sending its measurements, only.

These are all data and metadata used for data acquisition purpose:

- **Board Identification name**: This identifies the device uniquely in the monitoring system. This is one of the hard-coded information in the firmware.
- **GPS**: It consists of the latitude and longitude values from where the board captures data. This is one of the hard-coded information in the firmware.
- **RSS**: The Received WiFi Signal Strength, according to the WiFi module of the ESP-32 board.
- **Temperature**: The value of temperature given by the DHT-22 sensor in Celsius.
- **Humidity**: This is the percentage of humidity in the air. It is provided by the DHT-22 sensor, as is the case of temperature.
- **Gas concentration**: This is captured by the MQ-2 sensor, which measures the concentration of gas, like methane, butane or smoke, present in the air.
- **AQI**: An air quality indicator that we will discuss shortly. It indicates if the concentration of gas value is below, among or over respect to a specific range of values.
- **Sample frequency**: It represents the interval, in milliseconds, between two measurements of both sensor.
- **Minimum Gas value**: It indicates the minimum of gas concentration range values, useful in order to compute the AQI index.
- **Maximum Gas value**: It indicates the maximum of gas concentration range values, useful in order to compute the AQI index.

Except for the AQI value, all values are obtained by sensors or hard coded in the firmware. The AQI variable is an integer variable the spans between 0 and 2, computed by calculating the mean gas value every five gas measurements. The AQI value is computed as follows:

$$AQI(t) = \begin{cases} 1 \text{ if } avg \geq MAX\_GAS\_VALUE \\ 2 \text{ if } avg \leq MIN\_GAS\_VALUE \\ 0 \ otherwise \end{cases}$$

Temperature, humidity, gas concentration, AQI, RSS variables are sent with both protocols, the last three values (Sample frequency, Minimum and Maximum Gas value) are handled using the MQTT protocol. Moreover protocol switching operation is performed only by MQTT callback.
The HTTP is the second protocol implemented for the sensor data transmission.
The main problem we faced regarding the implementation of this protocol is the coexistence of both protocols, since they are TCP-based. We will discuss in details about the problem when we will talk about our sensor code.
The MQTT protocol is settled as the starting protocol. As the protocol changes the embedded device will start sending data in HTTP post request, but it will keep the connection to the broker open. The proxy server sends messages for protocol switching, frequency and AQI parameter updates and evaluation mode activation to the MQTT broker, in this way the board sensor receives updates and can change status also in HTTP mode.
For the sensor testing phase, the aim is to check the overall delay (RTT) for the packet transmission and the packet delivery ratio when using both protocol. We can activate evaluation mode regardless of the protocol. In evaluation mode, the sensor sends statistics about how many packages has received and average transmission delay, by the broker in case of MQTT, or by the proxy server in case of HTTP protocol.

2.2 Proxy server

Proxy server represents one of the central component of the overall structure. It mantains connection to broker and device and forwards communications and services between components. At first it retrieve measurements from connected devices in two ways:

1. **MQTT broker**: in the startup phase, proxy subscribe itself to some topics on a fixed MQTT broker. These subscriptions allow it to stay in listening of data provided by sensor devices for measurements and delivery statistics. Moreover, the server publish messages on ad-hoc topics to switching the protocol for devices communications, tuning parameters like measurement frequency or gas bounds for AQI computation and turn on or off the evaluation mode.

2. **HTTP requests**: At the beginning, proxy start an HTTP server, so when the communication protocol switched to HTTP, the board sends sensor data synchronously with the device sample frequency within HTTP requests.

The proxy server can receive data both in HTTP and MQTT protocol. Any board sensor can send data to the server without any authentication or authorization. We will examine each implemented feature of the proxy server in the next paragraphs.

Hyperparameter Setup. The server is listening to defined routes, like '/update-setup' or '/switch-prot-mode', to which a user can send HTTP requests specifying new parameter values related to protocol switching, bounds to compute AQI or evaluation mode. Also in this case, the user doesn't need authentication or authorization in order to send HTTP messages.

Switching Protocol Service. When proxy receives requests to the switching protocol route, it checks if the protocol code within the request body is valid, finally sends message to a dedicated topic using MQTT communication. The device, subscribed to that topics, reads message, accepts the protocol change and switch to new communication protocol since next sensor data sending.

Sensor settings Service. When proxy receives requests to the setup sensor route, it checks if the sensor frequency sendings and the interval values of AQI computation within the request body are valid, finally sends message to a dedicated topic using MQTT communication. The device, subscribed to that topics, reads message, accepts new settings and set sensor reading frequency, min and max gas range for AQI computation.

Evaluation Service. When proxy receives requests to the switching evaluation route, it checks if the evaluation code within the request body is valid, finally sends message to a dedicated topic using MQTT communication. The device, subscribed to that topics, reads message and starts sending every

five measurement an MQTT message contains number of packages have received by the proxy or the broker, based on the protocol used, and average packet delivery delay. Once proxy received the message, append it into a file in order to compute statistics.

Data Storaging. Proxy receives every measurements from sensors internally; it uses also a Influx Manager internal component to store data in the Influx time series according to sensor board ID, GPS location and Influx Bucket name.

Outdoor Monitoring. At every measurement message, Proxy call API to communicate GPS location, embedded in the message, to an external server which measures the external temperature of a specific location. In the response, the proxy computes the mean temperature and sends the result to Influx, which will store it in a specific Bucket with GPS location.

2.3 Influx Database

Influx Database works like the storage layer in the overall architecture. This service saves data in the appropriate time series. Then, we can visualize and manage measurements, query over time and retrieve data for tasks like forecasting or analysis of variable changes over time. Specifically, we built Influx Databases for storing sensors and endpoints information: indoor and outdoor temperature, indoor humidity, gas concentation, AQI and even WIFI RSS. Each type of data will be stored in different Influx Buckets, filtering by the internal tag system to distinguish endpoints which capture measurements, using device identification code and gps information. Influx Database component triggers an alert when gas value is out range by checking the AQI value over time. At a certain interval, an Influx task compute the max value of AQI among measurements captured in the last 60 seconds. If the max values is above 1, an alert will be emitted to the proxy server.

2.4 Telegram Bots

We also want to communicate AQI alert and average sensor data measurements over time. For these scope, we have chosend Telegram bots. The Telegram bot is a service provided by Bot Father which can be used to notify some events to the end-user, on Bot Message chat. We have developed two bots: **gas_alert_bot** sends messages related to AQI alert, **sensor_stat_bot** communicate medium value of the measurements produced by sensor ten minutes ago, on user request.

2.5 Grafana Dashboard

Grafana Dashboard is a view component directly connected to the monitoring dashboard. It consists of an interface with multiple panels which display how time series data, stored in each Influx Bucket, varies on time. The Dashboard panels also compare sensor data with their predictions ones produced in forecasting step (only for gas, humidity and indoor temperature).

2.6 Forecasting Data

All data received from the end-point sensors were used by forecasting algorithms to predict temperature, humidity and gas future values. It were chosen two different forecasting algorithms:

- **ARIMA**: It is a statistical analysis model thet uses time-series data to understand better the data and to predict future values (trends). Like all forecasting algorithms, It can predict future values based on past ones and uses lagged moving averages to smooth time-series data.
- **Prophet**: It is a procedure for forecasting time-series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality. It works best with time-series that have strong seasonal effects and several seasons of historical data.

2.6.1 ARIMA model

The ARIMA model is an auto-regressive moving avarage model which 'explains' a given time series based on its own past values.
An ARIMA model is characterized by 3 terms:

- **p**: is the order of the auto-regressive (AR) term;
- **d**: is the number of differencing required to make the time series stationary;
- **q**: is the order of the MA term, the number of lagged forecast errors in the prediction equation.

The algorithm implemented inside the ARIMA model works only on stationary data so, it was checked if the time-series were stationary using the Augmented Dickey Fuller test. The null hypothesis of the ADF test is that the time series is non-stationary. So, if the p-value of the test is less than the significance level (0.05) then the null hypothesis is rejected and the time-series is indeed stationary.
Given the stationary data, the next step is to understand how to settle the ARIMA model parameters. For the p and q estimation, an auto-correlation and a partial auto-correlation test must be done.

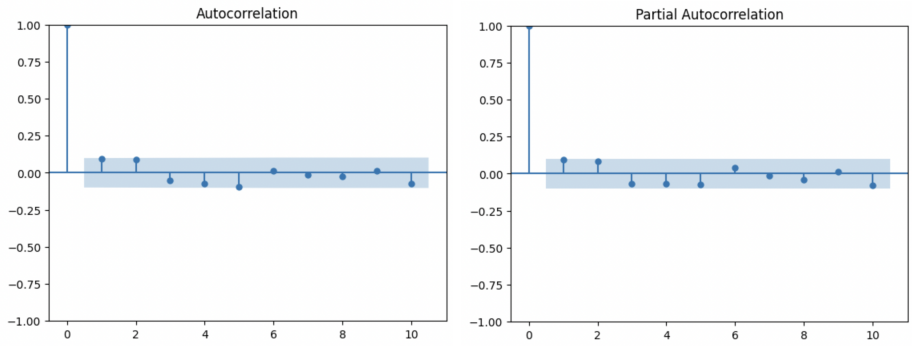

Figure 1: Auto-correlation and partial auto-correlation tests

In the example above, the p and q order obtained by exploiting the auto-correlation function is 1. Given a low level of auto-correlation and non-stationary data, a differentiation order of one is the best choice. An higher order of differencing might be needed if the series has positive auto-correlations out to an high number of lags.

2.6.2 Prophet model

Prophet was chosen because it is a forecasting model that works well for forecast tasks.
At its core, the Prophet procedure is an additive regression model with this equation:

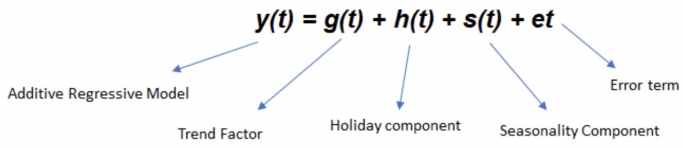

Figure 2: Prophet Forecasting Model Equation

The main components shown in the equation above are:

- **g(t)**: the *Trend Factor* implements two possible trend models:
  - Logistic growth model: the initial stage of growth is approximately exponential, as saturation begins, the growth slows to linear and at maturity, growth stops.
  - Piecewise Linear Model: it is a simple modification of linear model. By Default Prophet uses *Linear model* for its growth that has a constant rate of growth which is best suitable for without saturation growth.

  We can tune these parameters (Trend Components) in the prophet model by setting the changepoints.

```
m = Prophet(changepoint_prior_scale=0.5)
forecast = m.fit(df).predict(future)
fig = m.plot(forecast)
```

Figure 3: Prophet Forecasting Model Trend Component setting

  By default, this parameter (`changepoint_prior_scale`) is set to 0.05. Increasing it will make the trend more flexible.

- **s(t)**: the *Seasonality Component* provides flexibility (Weekly, Yearly, daily seasonality change components) to the model. This component uses Fourier Series. Seasonality is estimated using partial Fourier Sum and the Fourier order determines how quickly the seasonality can change.

```
m = Prophet(weekly_seasonality=False)
m.add_seasonality(name='monthly', period=30.5, fourier_order=5)
```

Figure 4: Prophet Forecasting Model Seasonality Component setting

  This parameter must be set very careful because it can lead to overfitting.

- **h(t)**: the *Holiday Component* is used, in the Prophet Model, to have an impact when forecasting data. The holiday effects could be useful in same cases but in our specific one it didn't help, so we didn't use this component.

An other interesting feature of Prophet is its ability to return the components of our forecasts. This can help reveal how daily, weekly and yearly patterns of the time-series contribute to the overall forecasted values.

Here an example:

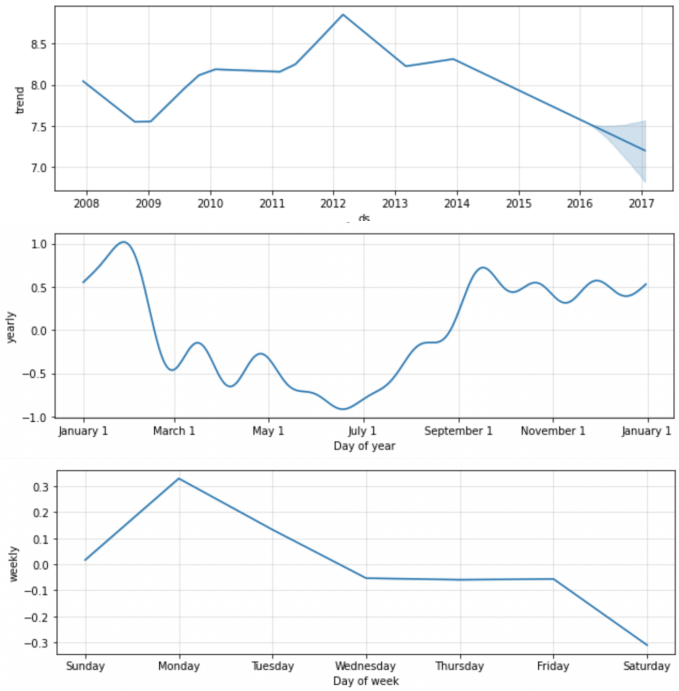

Figure 5: Trend, weekly and yearly patterns of the time-series.

2.7 Monitoring Network Architecture

In this subsection, we provide an abstraction of the deployment diagram of the system to show how components interact each other.

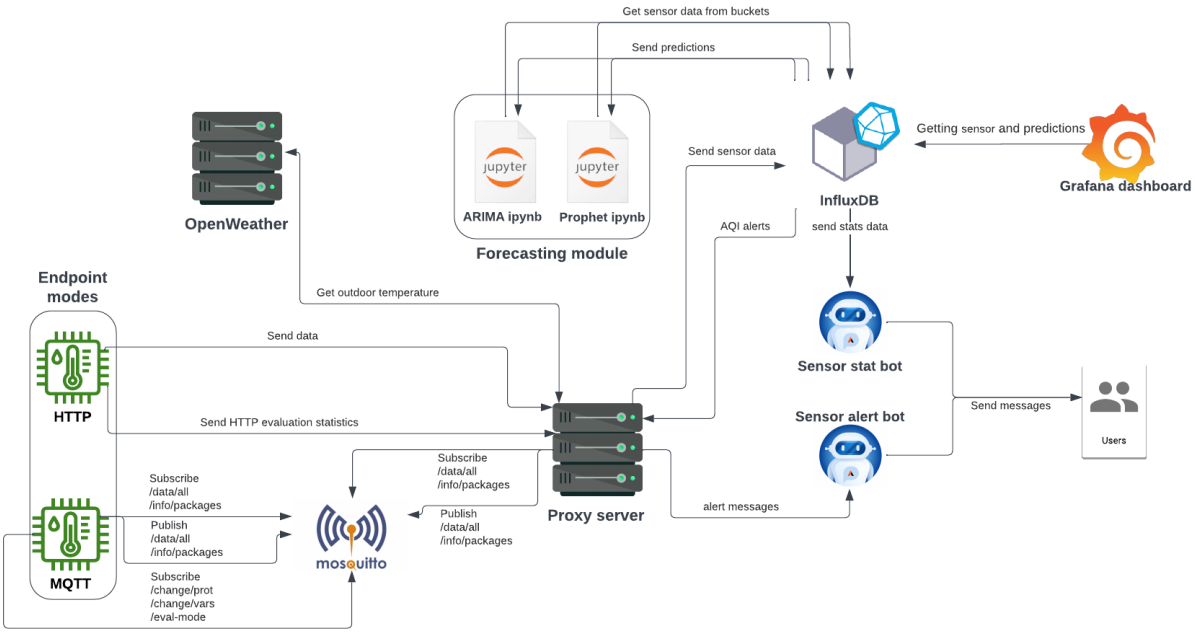

Figure 6: shows the architecture of components interconnection.

This diagram shows the architecture of Systems provided by different component intercommunication. On the left, we can see endpoint communication using HTTP or MQTT protocol at a time, with the ability to switch them in runtime. In the center, we have the Proxy Server, which works as the Gateway in the IoT environment, and the Python notebooks which get the sensor data from the Influx DB and generate forecasting data. Moreover the Proxy obtain outdoor temperature data by sending requests to OpenWeather Service using its API. Finally we can see Influx Database for data storage, the Grafana Dashboard for visualize sensor measurements and predictions, and Telegram Bots for alert notifications and data statistics.

3 Implementation

3.1 Sensors

One of the most important components of the overall project is the micro-controller and sensors.
In this subsection, we will describe the implementation of the data acquisition process, with particular attention to devices, sensors and library used.
Micro-controllers are ESP-32 connected to two sensors: DHT-22 and MQ-2, which, respectively, retrieve data for temperature, humidity and gas concentration.
Data processing uses *DHT.h* library to handle data captured by DHT22 sensor; meanwhile, it uses an analogical reading directly from the data pin of the MQ-2 to retrieve the voltage related to the gas concentration.
The device use the *Wifi.h* library to connect to WiFi, necessary for estimate the strength of the signal (RSS) at the same interval of sensor data measurement, other than have a connection to the MQTT broker and Server Proxy for the data communication phase.
It was chosen the *AsyncMqttClient* for handle communication using MQTT protocol in the asynchronous way, and the freertos libraries, specifically *FreeRTOS.h* and *timers.h*, for implementing multithreading mechanism. The main compution was splitted in two different threads: the first dedicated to keep WiFi connection open, the other one to keep the connection of the MQTT borker alive.

The *HTTPClient.h* library allows the ESP-32 to send sensor data and RSS to the Proxy server through HTTP requests.
Data management procedures use JSON objects provided by the ArduinoJson.h, ESP-32 uses these objects to trasmit data from different communication channels in JSON format. It will be a more accurate description of the protocols implementation in the next sections.

3.1.1 MQTT Implementation

ESP-32 provides some subscriptions on MQTT topics about evaluation metrics, data transmission and connectivity checking. Once the device is running, it starts the two thread and wait for connection to WiFi and MQTT broker, then it subscribes to all MQTT topic and will send sensor data at a certain interval. Each time the sensor will receive a message on the subscribed topics, the MQTT callback will activate and handle the message. We can see in detail these topics as follow:

- **Setup Topic**: It is called *sensor/change/vars* and is used to retrieve possible metadata management for the micro-controller itself. Subscribing this topic, we can receive gas bound for AQI computation and sample frequency modification;
- **Switch Protocol Topic**: ESP-32 receives from this topic the new communication protocol choice from the Proxy, so it will stop using the current protocol and start using the new one. The protocol name is *sensor/change/prot*;
- **Switch Evaluation Topic**: This topic is named *sensor/change/eval_mode*. As for the switch protocol topic, when the device receives messages at this topic, it will change sending evaluation data mode from the current way to the opposite;
- **Evaluation Topic**: Its name is *sensor/info/packages*. If the switch evaluation mode is on, the device will publish on this topic a string describing how many packages will be received by the MQTT broker over all packages sent and average delay every five measurements.

In addition to the previous topics, the micro-controller publishes data on the */data* topic in a time interval defined by the sample frequency variable.

3.1.2 HTTP Implementation

If the communication protocol chosen is HTTP, the device create an *HTTPClient* object to send sensor and RSS data in HTTP POST request to Proxy server endpoint, specifically at the */info-packages* route. Every five requests and in case the evaluation mode is active, the sensor sends an HTTP request including evaluation data.

3.2 Proxy Server

Proxy server use *express.js* to build the basic infrastructure for the HTTP intercommunication with the micro-controller, external services and all other components of the system. It implements MQTT channels for MQTT device communication support using the official MQTT library. We use *request* library for handle HTTP device intercommunication with devices and external services.
The Proxy server acts as a client actor that wait for data transmitted by the device and perform some actions in response.
In the next subsections, we will describe the implementation of the Proxy Server and how it interacts with Influx Database, OpenWeather Service and Telegram bot.

3.2.1 APIs

The following APIs focuses the implementation in *express.js* library, every functionality is implemented in *protocol.js* and exported to *app.js*:

- **/update-setup**: This route provides the function to update the session object on the Proxy server and, furthermore, publish on an MQTT embedded topic the new setup parameter values for the forwarding on micro-controllers.

- **/switch-prot-mode**: This API notifies the Proxy server to update the protocol for all connected devices. The Proxy server publishes on an MQTT topic the forwarded information to update micro-controllers and change the communication protocol.
- **/switch-eval-mode**: This API notifies the Proxy server to change the current behavior for evaluation analysis for all connected devices. The Proxy server publishes on an MQTT topic the forwarded information to activate or deactivate the evaluation metric production.
- **/info-packages**: This API is called by sensors in case the evaluation mode is active. The micro-controller sends evaluation data in HTTP request to this route when the communication protocol is HTTP.
- **/data**: the micro-controllers send sensor data and RSS to this route in HTTP requests when the chosen communication protocol is HTTP.
- **/newTelegramUser**: Telegram API service sends HTTP request to this route to communicate that another Telegram user starts using **gas_alert_bot**, so the Proxy gets the user ID in order to send AQI alert notification.
- **/aqi_alert**: InfluxDB service will call this API in case of AQI value changes, in such a way Proxy can send notification to the Telegram users that starts the **gas_alert_bot** bot.
- **/doc**: When Proxy server starts, it is possible call this route on a browser to use OpenAPI specification of this project, to see and send HTTP requests to first three routes.

3.2.2 Influx Database

Influx Database is deployed in the same host of the proxy server and it uses *Flux* language as a query language.
Influx Database is implemented in a *Javascript* file called *InfluxManager.js*, a class defined in the *protocol.js* to let available the intraction between Proxy Server, Telegram Bot and Influx Database.
All query and write operations are defined internally of the *InfluxManager.js* as methods. the class used *InfluxDBClient* library to implement the following procedures:

1. **Influx Database query**: this method is used to extract data from the influx database and print the results. Also, it is possible to filter the result by setting host, start and stop time, field, latitude and longitude of real measurements.
   As said before, the Influx query was written in *flux language* and it was used the *queryAPI* object given by the *InfluxDBClient* library.
2. **Write on Influx Database**: this method receives in input the end-point id, GPS coordinates, bucket name and the value to store and creates a *Point* object. As Influx Database query, this method used the *InfluxDBClient* library and the *WriteApi* object provided by the library. Thanks to the *WriteApi* object it was possible to define tags related to the point (host, field, latitude, longitude and prediction[1]), which defines the end-point data in the *Influx Database* uniquely.
3. **Query for Mean Value**: this query takes in input the bucket and the end-point id and is used by Telegram Bot to return the current mean value of the last values in the past 10 minutes.
4. **Write Forecasting Points to Influx Database**: this method takes in input the end-point id, GPS location, a list of predicted values, bucket name and timestamp. It writes on *Influx Databse* the forecasted data in the specific bucket passed as input.

[1]Prediction tag should be *no* if the measurements is real and *yes* if is a forecasted one.

3.2.3 OpenWeather

Each time Proxy Server receives temperature data from ESP-32 device, it sends an HTTP request to OpenWeather service, using its API, to retrieve average outdoor temperature at the position indicated by GPS coordinates of the micro-controller.

3.2.4 Telegram feature

In order to transmit AQI alerts and data statistics to users, it was decided to split the Telegram component into two separated bots, using the *Telegraf* library, the modern Telegram API for Node.js framework. The one, once running, registers an endpoint to the Telegram WebHook to receive new users information and send them the alerts received by Influx. The other bot waits users request to send in their Bot chat median values about temperature, gas or humidity by querying the appropriate Bucket.

3.3 Forecast

As said before, different forecast models were used two to predict future values.

3.3.1 ARIMA

Before the training phase, it was used *pandas* and *InfluxDBClient* library to perform queries and retrieve Influx rows from all Influx buckets for training data on a specified model. Every trained model focuses ARIMA learning method provided by *statsmodels* library.
Depending on the bucket type, the parameters passed in input to the ARIMA model will change according to what was said before in the ARIMA section.
All the forecasted data were published to Influx Database using the WriteApi object provided by the InfluxDBClient library.

3.3.2 Prophet

Before the training phase it was used *pandas* and the *InfluxDBClient* library to perform queries and retrieve Influx rows from all Influx buckets for training data on a specified model.
It was used a method called *result_to_dataframe* to convert Influx rows to *pandas* dataframe. It was instantiated a *Prophet* object and passed in input all the parameters needed, according to what was already explained in Prophet section. Different models were created, one for each bucket inside Influx Database.
All the forecasted data were published to Influx Database using the *WriteApi* object provided by the *InfluxDBClient* library.

3.3.3 Deployment Diagram

In this subsection it will be described the component organization during the implementation phase.
Influx Database and Grafana were configured locally. As MQTT broker it was used the open-source *Mosquitto* broker, presented to the student during the Internet of Things course.
Telegram bot was tested on normal mobile devices with an internet connection and Telegram App installed.
All sensors transmission via MQTT and HTTP protocol are implemented only for local network.
Of course, the micro-controller had its personal public IP available after the WI-Fi connection.

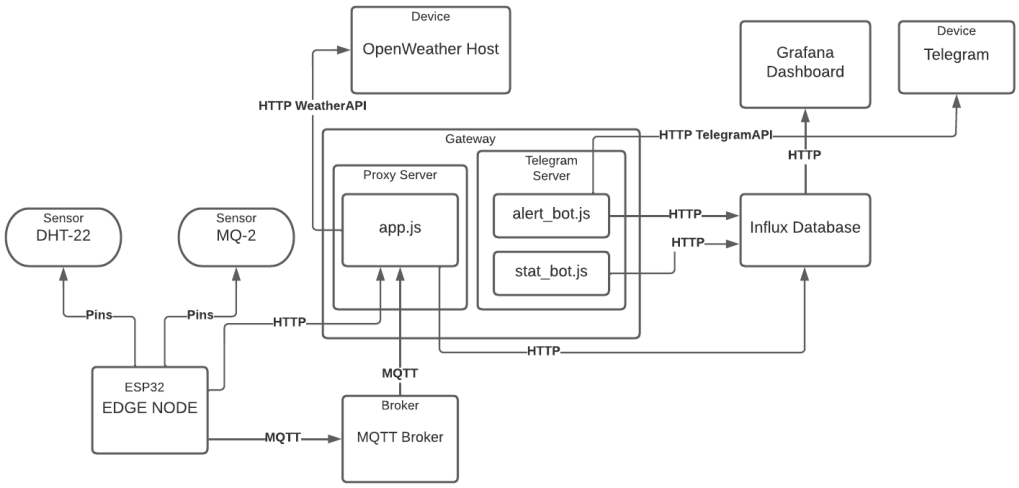

Figure 7: Deployment Diagram: components interconnection, protocol useges and run-time application flow.

4 Results

4.1 End-point Evaluation Metrics

In this subsection, will be analyzed the results about metrics on edge device connection, latency and package loss percentage.

Protocol	Latency	Package Loss
MQTT	10 ms($\pm$ 2)	0 %
HTTP	36 ms($\pm$ 4)	2 %

Table 1: Table of metrics.

Latency and package loss percentage were collected by the server every five measurements received from the end-point. All the evaluations were made on a single end-point connected at the same Wi-Fi of the servers and the mean signal strength was equal to -44 dBm.

4.2 Forecasting Models Evaluation Metrics

In this subsection, will be analyzed and evaluate the results obtained by ARIMA and Prophet model. Real data were collected for two consecutive days and the goal was to predict future values for the next few hours. This approach lets made it possible to evaluate the forecasting behavior in a real context for two daily periods.
Regarding ARIMA, it was used a Google Colab notebook to find the best parameters related to the data collected by the edge device.

In the table below are shown the results obtained by ARIMA models and their evaluation through *RMSE* metric, *Prediction Mean* and *Interval of Confidence*.

Model Type	RMSE	Prediction Mean	Interval of Confidence
Temperature	0.048	17.5	[17.37, 17.62]
Humidity	0.248	64.36	[63.36, 65.35]
Gas	23.005	167	[57, 276]

Table 2: Forecasting Models Evaluation Metrics related to the RMSE (Root Mean Square Error) and the interval of confidence for the estimation of values on the model predictions.

Regarding Prophet, the same data used on ARIMA were passed in input to the model.
In the table below are shown the results obtained by Prophet models and their evaluation through *RMSE* metric, *Prediction Mean* and *Interval of Confidence*.

Model Type	RMSE	Prediction Mean	Interval of Confidence
Temperature	0.944	18.44	[18.29, 18.58]
Humidity	2.496	63.25	[62.93, 63.57]
Gas	35.618	110	[103, 117]

Table 3: Forecasting Models Evaluation Metrics related to the RMSE (Root Mean Square Error) and the interval of confidence for the estimation of values on the model predictions.

It can be seen from Table 2 and Table 3 that the two forecasting models returned different results. It's clear that ARIMA model predicts data more accurately then Prophet model. This because Prophet need a lot of data that assumes a recurring pattern over time (weeks, months) to predict more accurately.
In both cases the gas prediction seems to be unpredictable according to the confidence interval. Maybe because gas values, collected by the sensor, change a lot over time without any recurring pattern and the prediction models interpret it as outlier detection.

5 Conclusions

In this section will be discussed some considerations about the overall project and the Internet of Things patterns.
First of all, Internet of Things offers many benefits to businesses, including decreased operating costs, new consumer insights and opportunities to optimize business operations. But IoT patterns have same problems compared to normal distributed or network architectures. The most important ones are the low-energy and low-computational power that characterise the edge-device. These problems implies many constraints and a lot of work during the implementation phase. An other problem is the Gateway which has high complexity due to the run-time management of many flows of data.
During the course of the project, it was difficult to implement forecasting methods that accurately predicted values, due to data collected by the edge-device and to the continuous training of different models.
It was interesting to develop the switch protocol feature as it allowed to adapt the Proxy Server on multiple protocols support and manage data exchange in different ways, maximising the performance of each protocol. Therefore, there are few things to take into account when developing an IoT application, but with the correct best practices it is possible to build a real world application. For example, a real

temperature, humidity ans gas monitoring system in an indoor environment, even for sensor integration inside a smart home.