School of Engineering and Material Science

Introduction to IoT ECS782

Feasibility Study and Prototype Design Document: Air Quality Monitoring application

---

Author:
Name: Vishal Kashyap
Student number: 210684838

Content list

1. Introduction

Air pollution is the accumulation of small particles in the air that cause health issues if inhaled. These small particles include nitrogen dioxide, carbon monoxide, carbon dioxide, sulphur dioxide, ozone and particulate matter like soot and dust. WHO says Air pollution causes an estimated 4.2 million deaths per year due to heart diseases, chronic respiratory diseases, lung cancer and stroke. About 91% of the world's population lives in places where air quality exceeds safe thresholds (World Health Organization, 2021). Daily life is being disrupted in cities like Delhi, where air pollution levels exceed AQI 500 at times. This is equivalent to smoking 26.5 cigarettes per day (Moneycontrol, 2021). Air pollution has become a problem that needs to be solved with urgency. Therefore, it is vital to take measures and control the release of harmful emissions in the air with the help of technology.

The internet is expanding and being incorporated into more things than ever before. These "things" of various sizes and capabilities are embedded with sensors and endowed with the ability to communicate with other devices and systems. IoT can play a key role in the control of air pollution. Smart things embedded in

2. Project scope and Objective

This project will look at the design of a mobile application that will give the user information about pollution levels from various stations located in different blocks, cities, or countries. Each station will host an MCU board and sensors connected to the board. These sensors will capture information to help calculate the Air Quality Index (AQI). Information captured from stations will be stored in the cloud, which will be responsible for analyzing the captured data and providing insights. The user will be able to interact with the cloud using an application to access the latest AQI update or predicted AQI for a specified time and location.

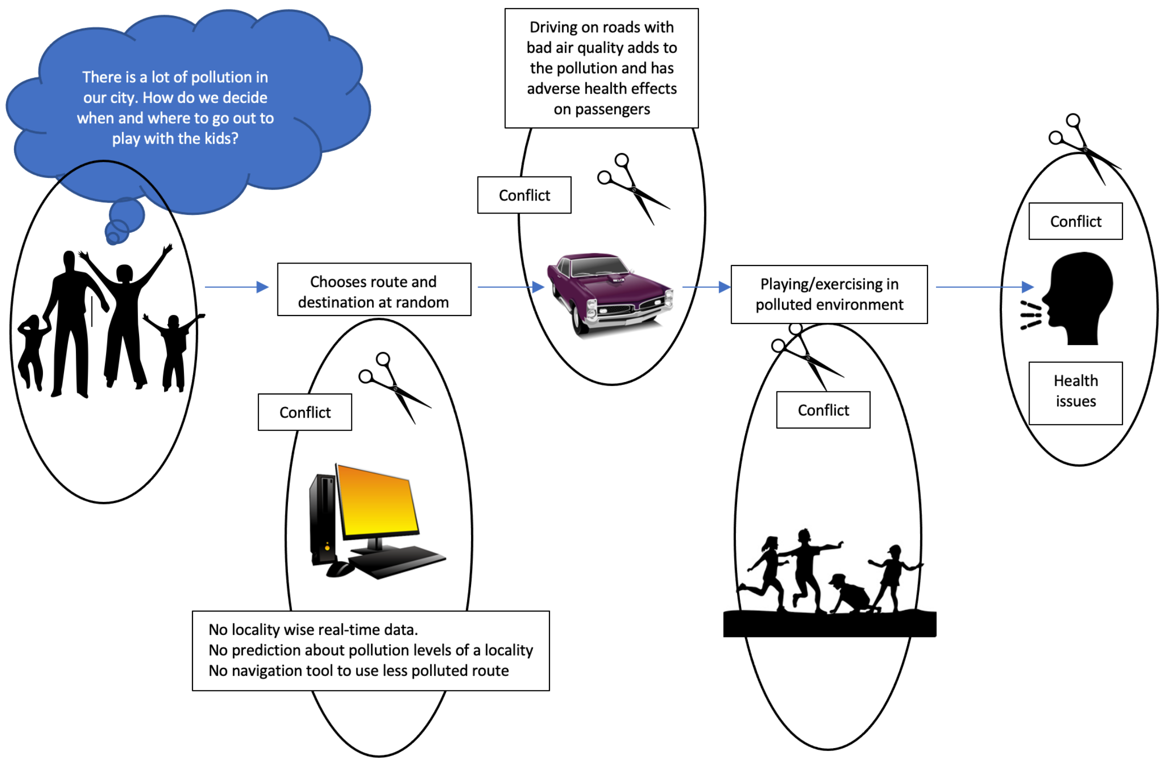
The objective is to develop an architecture for monitoring AQI that is:

i) easily scalable
ii) low cost
iii) real-time
iv) reliable
v) transparent and networked (Smart Device Property)

3. Use cases

a. SSM2 system problematic situation 1:

The diagram below shows the high-level problematic situation of deciding when and where to go out in a polluted city:



There is a lot of pollution in our city. How do we decide when and where to go out to play with the kids?
Chooses route and destination at random
Conflict
No locality wise real-time data.
No prediction about pollution levels of a locality
No navigation tool to use less polluted route
Driving on roads with bad air quality adds to the pollution and has adverse health effects on passengers
Conflict
Playing/exercising in polluted environment
Conflict
Conflict
Health issues

Fig. 1. SSM 2 high level diagram for problematic situation 1

Conflicts in the situation:

i) The user is unable to get adequate pollution data to plan his outing.
ii) Choosing a route and area for exercising without considering the pollution levels may cause serious health issues
iii) Prolonged exposure to such polluted air results in fatal respiratory diseases.

b. SSM2 system problematic situation 2:

The following diagram shows the problematic situation of controlling traffic flow to control air pollution:

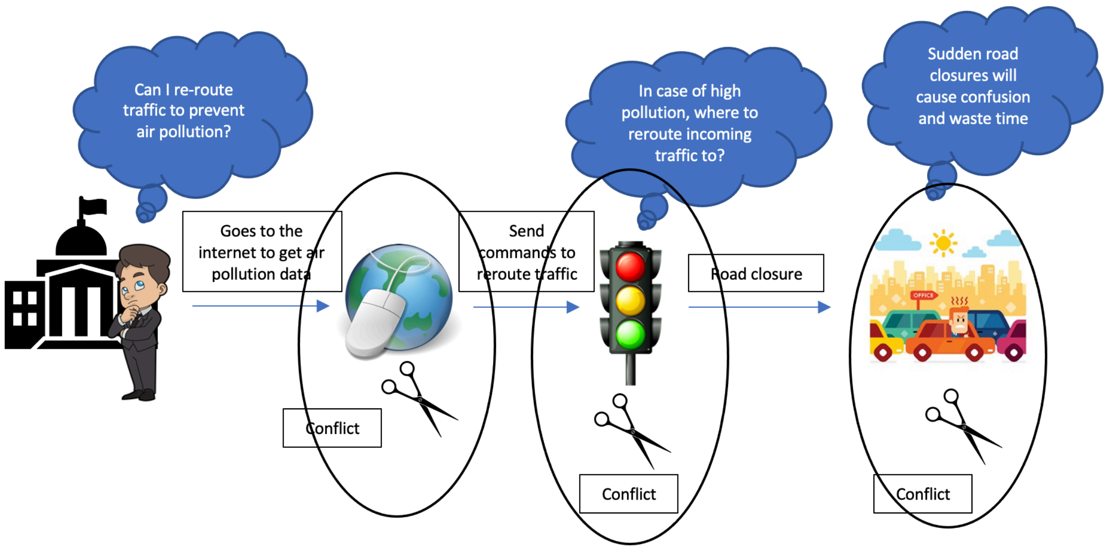

Can I re-route traffic to prevent air pollution?
Goes to the internet to get air pollution data
Conflict
In case of high pollution, where to reroute incoming traffic to?
Send commands to reroute traffic
Conflict
Road closure
Sudden road closures will cause confusion and waste time
Conflict

Fig. 2. SSM 2 high level diagram for problematic situation 2

Conflicts in the situation:

i) Not guiding traffic flows based on pollution leads to severe air quality levels.
ii) The user cannot get pollution data and predictions for every locality to make an informed decision on traffic flow.
iii) The user gets a warning about AQI level from a limited number of sensors and takes sudden decisions to close roads, leading to
    a. confusion for drivers.
    b. Poor air quality (re-routing done once air quality is already poor)

Storing all solutions in the cloud and getting predictions for air pollution from all stations at regular intervals will help government officials plan traffic flow. This will help keep AQI levels lower.

The above two problematic situations give strong reasons to develop the air quality monitoring app. This app will help people plan their outings, giving due weight to pollution. It will also help manage traffic flow by devising routes that avoid high pollution areas using the predictions obtained from the application.

c. Proposed Solution

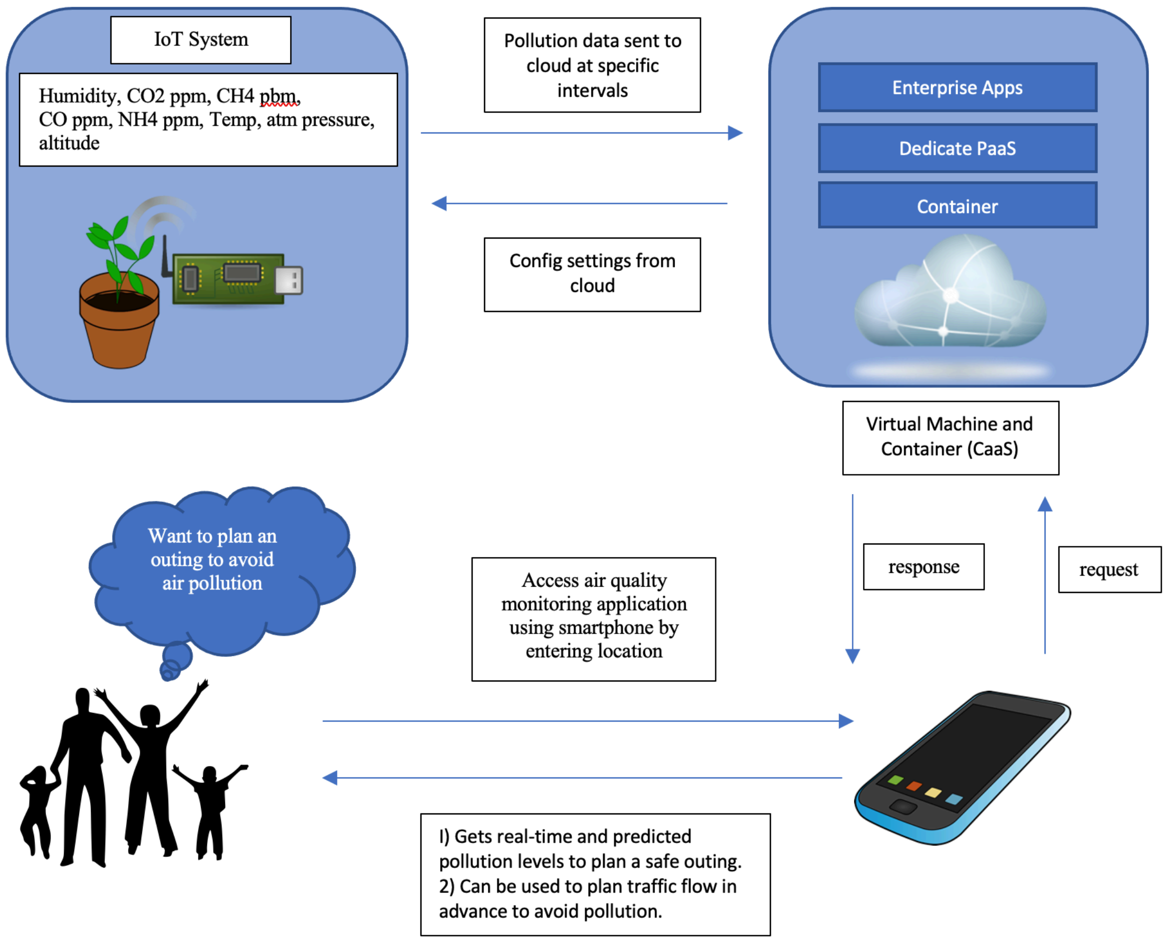

IoT System
Humidity, CO2 ppm, CH4 pbm, CO ppm, NH4 ppm, Temp, atm pressure, altitude
Pollution data sent to cloud at specific intervals
Config settings from cloud
Enterprise Apps
Dedicate PaaS
Container
Virtual Machine and Container (CaaS)
Want to plan an outing to avoid air pollution
Access air quality monitoring application using smartphone by entering location
response
request
I) Gets real-time and predicted pollution levels to plan a safe outing. 2) Can be used to plan traffic flow in advance to avoid pollution.

Fig. 3. Main flow of data in proposed solution

The above figure shows a high level diagram of the proposed solution with main data flows in the architecture. In this solution, the user will be able to access information regarding pollution levels and predictions for the same using a mobile application. The user will be able to choose from a list of stations. Each station will have an IoT system installed with a microcontroller and sensors to measure air quality. The data from these sensors will be stored in the cloud. The cloud will provide enterprise apps to analyse and predict pollution levels. The mobile application will connect with the cloud to retrieve information from its database and display it on the screen.

4. Requirement Analysis

a. Functional Requirements

Requirement ID	Requirement	Priority (Essential, Conditional, Optional)
FR1	**Configuring and handling sensor data** Description: MCU boards at different stations need to be configured initially and connected to Wi-Fi. These boards are configured via MQTT and listen to their respective topic (/station/station_id). The broker uses this channel to enable or disable a station and set time intervals to receive sensor readings. The MCU board will collect sensor data using I2C and SPI interface. To read signals, we will have to install libraries on Arduino. These libraries are specific to the sensor and available online. The following actions need to be performed for this: i) The MQTT client on the MCU board should subscribe to the topic published by the broker on the cloud. ii) A station config class needs to be developed on the MCU board to read configuration messages in MQTT format from the broker and implement the changes. iii) Configuration message should be sent from MQTT broker (installed on the cloud) to client upon addition of a station to be tracked. iv) A sensor listener class will have to be developed to aggregate data from sensors and send to the MQTT client. v) MQ9 sensor should send CO and CH4 levels to the sensor listener class. vi) DHT11 sensor should send temperature and humidity readings to the sensor listener class. vii) SNS MQ135 sensor should send NH4, CO2, ethanol levels to the sensor listener class. viii) MQTT client will send pollution data received from sensor listener class to MQTT broker on the cloud.	**Essential**

FR2	**Storing sensor data on the cloud and using enterprise analytics solutions to predict pollution levels:** Description: The cloud server will collect sensor data from stations at regular intervals in JSON format. This communication will be via MQTT protocol. A mosquito broker will pass the data to the Node-RED framework, which will store the data in MongoDB in JSON format. MongoDB will serve this data to enterprise apps and RESTful services upon users' request. The following needs to be developed: i) Create an Amazon EC2 instance in the desired availability zone. A security group will need to be defined for this. ii) Set up MQTT broker on the created EC2 instance using yum package manager and configure this to listen to station topics. iii) Node-RED framework to send data from MQTT broker to MongoDB will need to be set up. iv) A MongoDB database will need to be initialised. v) Models like Liner regression and Neural networks will need to be trained on stored data. These trained models will need to be stored and retrained at specific intervals. vi) A class to provide RESTful services to mobile application. This request body must contain data in JSON format.	**Essential**

b. Hardware and software requirements

Hardware

- Arduino Uno Wi-Fi Rev2 board. (used to collect sensor data and as a gateway to the cloud)
- Power source: 5V
- Smartphone

- Connecting wires
- Breadboard
- Wi-Fi for connecting MCU board to cloud.
- DHT11 temperature sensor module (https://www.adafruit.com/product/386)
  - To sense temperature and humidity.
  - Gives out digital output.
  - Good for 20-80% humidity and 0-50 degree Celsius temperature
  - Operating voltage: 5 V
- SNS MQ135 (https://www.olimex.com/Products/Components/Sensors/Gas/SNS-MQ135/)
  - Measures NH4, CO2, ethanol
  - Has both analog and digital output.
  - Operating voltage: 5 V
- Grove - Gas Sensor(MQ9) https://wiki.seeedstudio.com/Grove-Gas_Sensor-MQ9/
  - Measures CO, CH4
  - Only Analog Output
  - Operating voltage: 5 V

Note: MQ9 and SNS MQ135 sensors require a preheating time of 24-48 hours.

Software

- Amazon EC2 with Docker and Kubernets. This will host:
  - Webserver (Apache)
  - RESTful services
  - Node RED framework
  - MQTT Mosquito broker
  - MongoDB database.
  - AI and data analytics tools.
- Libraries for sensors:
  - DHT11 temperature sensor module: https://www.arduinolibraries.info/libraries/dht-sensor-library
  - SNS MQ135: https://www.arduinolibraries.info/libraries/mq131-gas-sensor
  - MQ9: https://create.arduino.cc/projecthub/electropeak/how-to-calibrate-use-mq9-gas-sensor-w-arduino-e93cb1

5. Design

a. SSM3: Root definition of the system

i. Station world view

- The temperature sensor sends readings to the Arduino board.
- MQ9 sensor sends readings to the Arduino board.
- SNS MQ135 sensor sends readings to the Arduino board.
- Arduino takes an average of readings to remove noise and parses pollution readings in JSON format (process). This data is sent as an MQTT message (output) over the topic /stations/{station_id} at specified regular intervals.

ii. Cloud world View

- MQTT broker sends initial configuration messages to all stations.
- MQTT broker listens to stations' topics (input), filters them based on topic or content (process) and passes it over to the node RED framework using MQTT protocol. (output)
- Node RED launches stream to listen to broker messages (input), validates it (process) and stores it in MongoDB using JSON format (output).
- Communicates with mobile API using RESTful services (input), connects with MongoDB and enterprise apps to process request data (process) and sends output to the user.

iii. Smartphone API world view

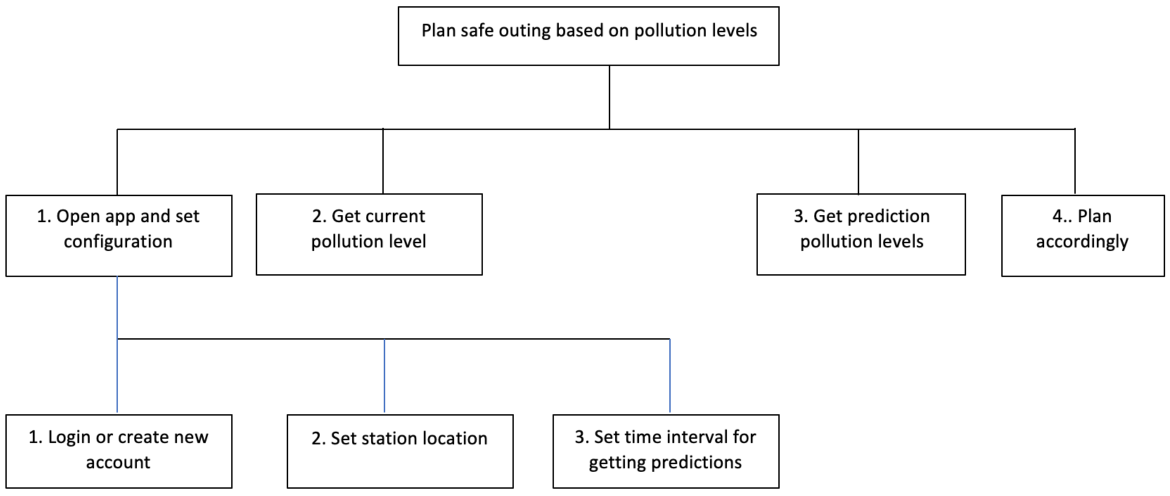
- Gets location and time interval for prediction from the user (input), parse data (process) and send to the cloud via REST services.
- Receives results from the cloud (input), converts to human-readable format (process) and displays on UI (output).

iv. User world view

- Configures the app for the first time.
- Authenticates and sets location and time interval for predictions (input) and sends information to the cloud.
- Reads the results from cloud and plans accordingly (output).

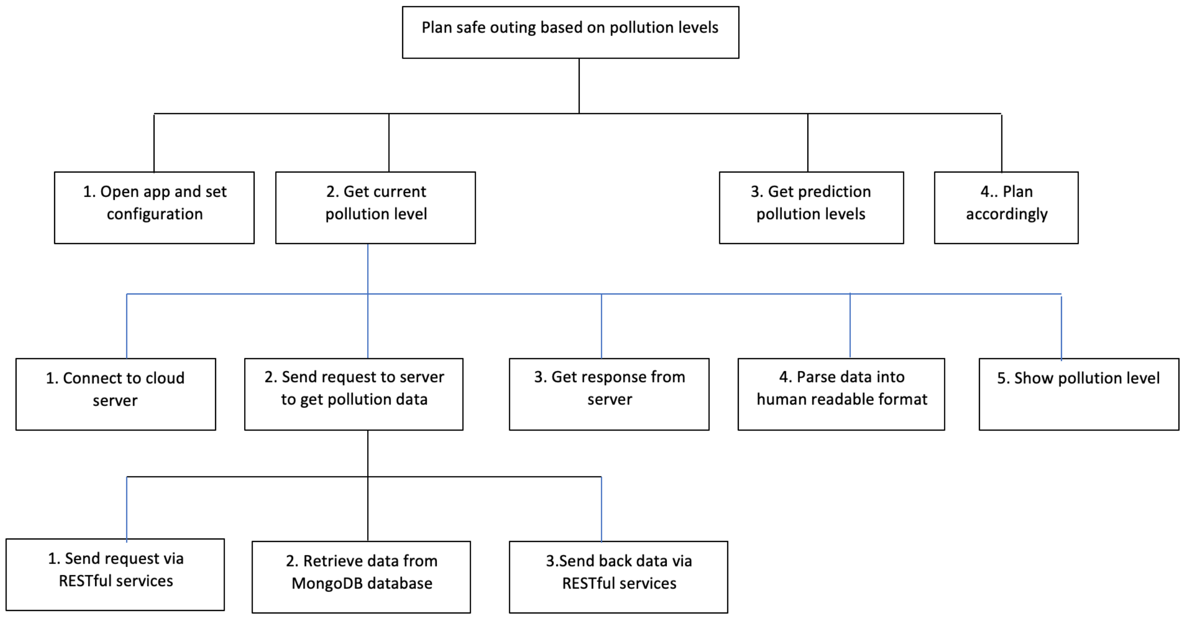
b. SSM4 conceptual modelling using Hierarchical Task Analysis

This section does the Hierarchical task analysis for the solution:



Plan safe outing based on pollution levels
1. Open app and set configuration
2. Get current pollution level
3. Get prediction pollution levels
4.. Plan accordingly
1. Login or create new account
2. Set station location
3. Set time interval for getting predictions

Fig 4 HTA : subtasks for task 1



Plan safe outing based on pollution levels
1. Open app and set configuration
2. Get current pollution level
3. Get prediction pollution levels
4.. Plan accordingly
1. Connect to cloud server
2. Send request to server to get pollution data
3. Get response from server
4. Parse data into human readable format
5. Show pollution level
1. Send request via RESTful services
2. Retrieve data from MongoDB database
3.Send back data via RESTful services

Fig 5 HTA : subtasks for task 2

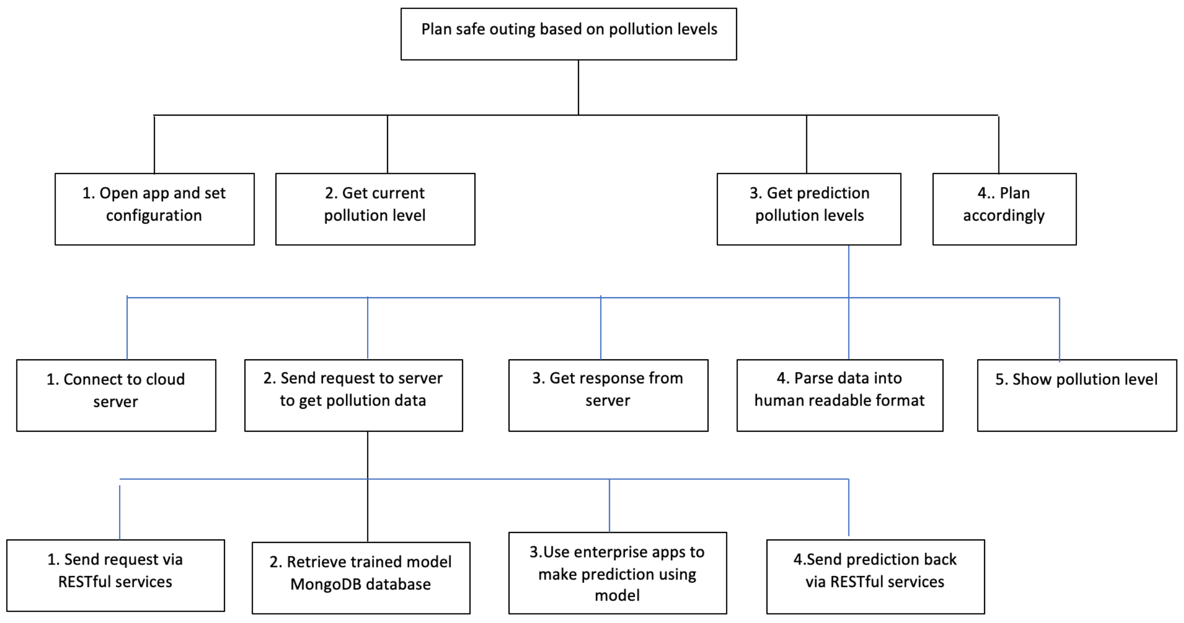

Plan safe outing based on pollution levels
1. Open app and set configuration
2. Get current pollution level
3. Get prediction pollution levels
4.. Plan accordingly
1. Connect to cloud server
2. Send request to server to get pollution data
3. Get response from server
4. Parse data into human readable format
5. Show pollution level
1. Send request via RESTful services
2. Retrieve trained model MongoDB database
3.Use enterprise apps to make prediction using model
4.Send prediction back via RESTful services

Fig 6 HTA : subtasks for task 3

c. System Design Diagram (Motivation for design choice)

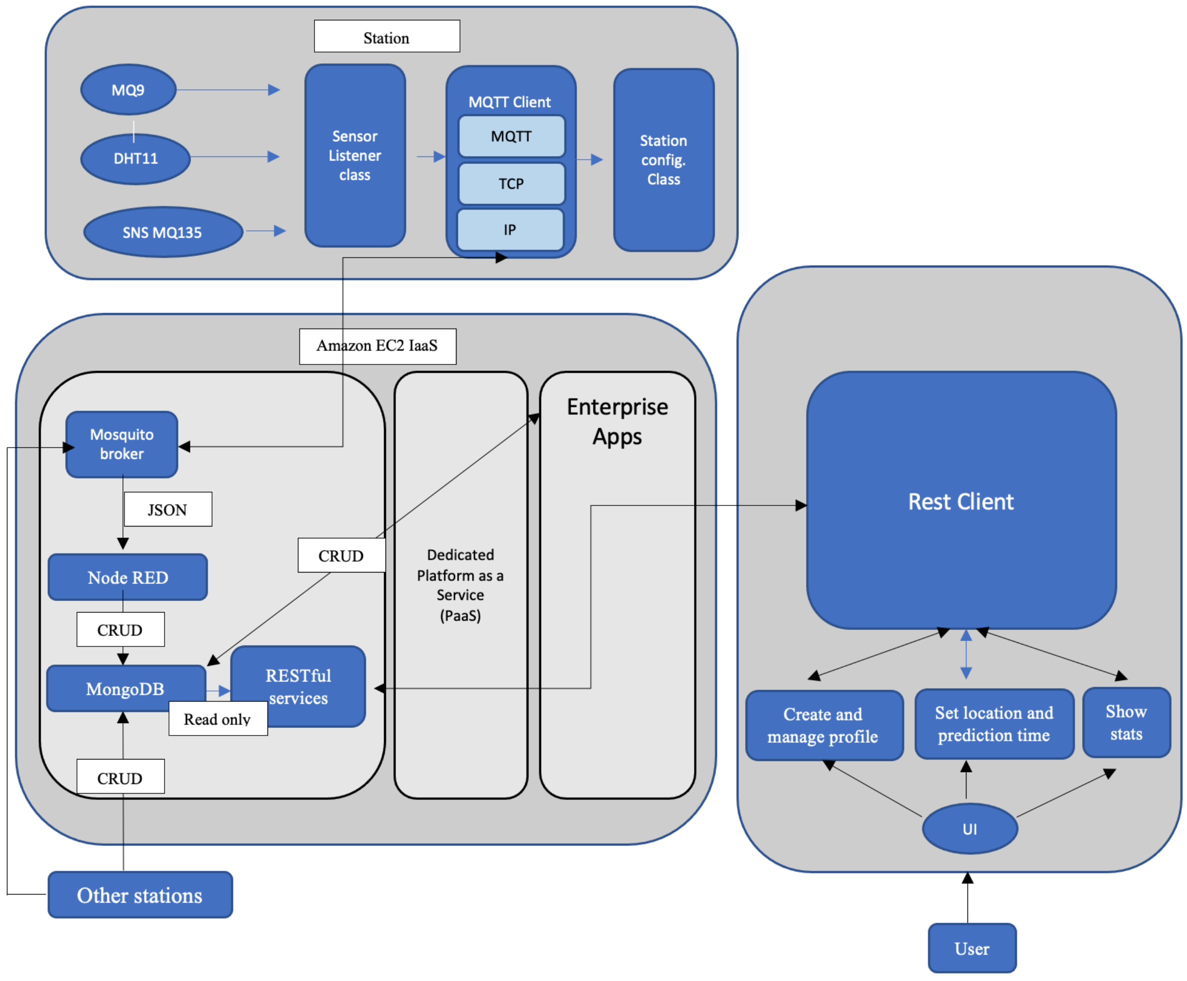


Fig 7. Detailed System Design diagram

Functionality and motivation:

Amazon EC2:

- Used for data storage and processing.
- Present in multiple locations known as availability zones. Makes it ideal for connecting with stations with low-latency networks.
- Provides security using security groups.
- Flexible in terms of operating systems and applications required.

MQTT protocol:

- MQTT is lightweight and easily scalable, making it ideal for the current application.
- It is publisher-subscriber based and runs over the TCP/IP protocol stack.

Sensor listener class:

- Collects data from sensors and takes an average over some time interval to remove noise.
- Passes data to MQTT client

MQTT client on station:

- Subscribes to topic from MQTT broker on cloud. ((topic: /stations/{station_id}). Cloud server sends configuration settings. (on/off, interval, topics to publish etc)
- Publishes pollution data receiver from sensor listener class to the same topic.
- MQTT runs over the TCP/IP protocol's stack.
- Communication handled in JSON format.

Mosquito Broker:

- Hosted on AWS EC2
- Selects and Filters incoming messages based on topic or content to keep data from different stations segregated.
- Data in JSON format is passed to Node-RED

MongoDB:

- Non-relational database, which stores data in JSON format.
- Gathers data from Node-RED and other stations (CRUD operations).
- Outputs data to mobile application using RESTfull services.
- Outputs data to enterprise apps for prediction of pollution level

6. Analysis of design issues and design options

a. Survey of related works

Marques and Pitarma (2019) developed a low-cost indoor air quality monitoring application. The objective of their application was to give real-time alerts to users. They used an ESP8266 module, a simple SoC that can connect to Wi-Fi and make TCP/IP connections. Data aggregation and analysis were done on the ThinkSpeak platform, an application for IoT on the cloud. MICS-6814 sensor was used to collect pollution level readings. This sensor can detect Carbon monoxide, Nitrogen dioxide, Ethanol, Hydrogen, Ammonia, Methane, Propane and Iso-butane (The MiCS-6814 datasheet package. Features Detectable gases).

Johnston et al. (2019) conducted a study to observe the effectiveness of low-cost sensors in measuring particulate matter and test LoRaWAN for wireless communication. All data was stored on the Microsoft Azure cloud platform. They found that LoRaWAN proved to be an effective second communication channel that can operate during power outages and in areas without network access. The Alphasense OPC-N2, Plantower PMS5003, Plantower PMS7003 and Honeywell HPMA115S0 sensors were proven effective in measuring PM levels and trends.

Luo et al. (2018) carried out a study to formulate cyber security recommendations for AQI monitoring apps. They designed several attacks to pollute sensor data in three scenarios: using MAC address, physical access to sensors and large-scale system attacks. Security recommendations:

i) Use secure boot on sensors
ii) Flash encryption should be used to protect data on flash.
iii) Mutual authentication should be used for communication between sensor and server.
iv) Prefer Wi-Fi localization over GPS localization.

b. Design Issues

Software issues

MQTT is simple to use and can connect thousands of devices easily. Besides these advantages, it is lightweight and has a fast response time. These qualities make it a desirable protocol for IoT applications where IoT nodes are resource-constrained. However, the MQTT protocol is not secure by design. This is done to keep the protocol usable in IoT devices which are usually low on computing power and memory. The MQTT gives guidelines to tackle security challenges that must be handled by the users (mqtt-v3.1.1-os Specification URIs, 2014). The documentation defines a framework called the MQTT cybersecurity framework to help identify security risks within an application that uses MQTT.

Using these security mechanisms within our application will be beneficial, but it will have to be if they can be installed on our lightweight and resource constraint IoT board. Security is going to be an important aspect of the application, and we might have to switch to a different protocol if we are unable to make the MQTT messages secure.

Another possible issue that might arise during the application development is the incompatibility between communication protocols of different packages. The developed solution uses a range of hardware (MCU boards, sensors, Amazon EC2 cloud and smartphone) and software. We may have to convert messages from one part of the system to be compatible with other parts.

The UI of the system should be designed with care to make it easy for a new user to navigate the app with ease.

Hardware issues

The libraries provided in 4 b have not been tested by the author. Many times a library does not work as expected on a given system. This might result from subtle hardware differences between our system and the system used for developing the library. Alternative libraries might have to be searched for while testing these sensors on our MCU board. The sensors used are common and have plenty of documentation and help available online.

There can be some drawbacks of using low-cost sensors; these drawbacks have been documented in the literature and are listed below:

- Interference from climate conditions (Mukherjee et al., 2017)
- Environmental conditions (Jayaratne et al., 2018)
- Lack of reproducibility (Rai et al., 2017)
- Data drift over time (Spinelle et al., 2015)

c. Including Smart Device properties

Context-awareness

The application can be fed physical environment context by accessing the user's location. When the user changes location, the mobile application will send a request to the cloud and get pollution levels in the new location. If the pollution levels are above a certain threshold and considered unhealthy, the application can notify the user. This feature will save the user time, which might have been used in checking pollution from time to time while travelling and help him avoid unsafe areas. This feature also displays **implicit Human-computer interaction**, an essential property of smart devices.

The application can benefit significantly from being user context-aware by accessing the paths a user usually takes while commuting. The application can search when these frequently taken paths have the lowest pollution levels and suggest these timings to the user. Alternatively, the application can suggest different low pollution paths to reach the same destination. This will require integration with google maps and the development of additional software.

We can also add an additional feature in MCU boards to search for available open Wi-Fi in the area and connect to the ones that have better signal strength. This will make the system more robust as it will not rely on a single network.

Implicit Human Computer Interaction (iHCI)

The proposed application could interact with a user's schedule stored on an application (ex. Google Calendar). This would give the air quality monitoring app access to users future location. Using this information, the app could send a request to the cloud, get the predicted pollution levels, and display it alongside the schedule. This would help the user reschedule/cancel his plans.

Bibliography

Jayaratne, R., Liu, X., Thai, P., Dunbabin, M. and Morawska, L. (2018). The influence of humidity on the performance of a low-cost air particle mass sensor and the effect of atmospheric fog. *Atmospheric Measurement Techniques*, 11(8), pp.4883–4890.

Johnston, S.J., Basford, P.J., Bulot, F.M.J., Apetroaie-Cristea, M., Easton, N.H.C., Davenport, C., Foster, G.L., Loxham, M., Morris, A.K.R. and Cox, S.J. (2019). City Scale Particulate Matter Monitoring Using LoRaWAN Based Air Quality IoT Devices. *Sensors*, 19(1), p.209.

Luo, L., Zhang, Y., Pearson, B., Ling, Z., Yu, H. and Fu, X. (2018). On the Security and Data Integrity of Low-Cost Sensor Networks for Air Quality Monitoring. *Sensors*, 18(12), p.4451.

Marques, G. and Pitarma, R. (2019). A Cost-Effective Air Quality Supervision Solution for Enhanced Living Environments through the Internet of Things. Electronics, 8(2), p.170.

Moneycontrol. (n.d.). *Delhi Air Pollution: This App Converts AQI Readings Into Number Of Cigarettes Smoked*. [online] Available at: https://www.moneycontrol.com/news/technology/delhi-air-pollution-this-app-converts-aqi-readings-into-number-of-cigarettes-smoked-4600261.html [Accessed 13 Dec. 2021].

mqtt-v3.1.1-os Specification URIs. (2014). [online] Available at: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.

Mukherjee, A., Stanton, L., Graham, A. and Roberts, P. (2017). Assessing the Utility of Low-Cost Particulate Matter Sensors over a 12-Week Period in the Cuyama Valley of California. *Sensors*, 17(8), p.1805.

Rai, A.C., Kumar, P., Pilla, F., Skouloudis, A.N., Di Sabatino, S., Ratti, C., Yasar, A. and Rickerby, D. (2017). End-user perspective of low-cost sensors for outdoor air pollution monitoring. *Science of The Total Environment*, 607-608, pp.691–705.

Spinelle, L., Gerboles, M., Villani, M.G., Aleixandre, M. and Bonavitacola, F. (2015). Field calibration of a cluster of low-cost available sensors for air quality monitoring. Part A: Ozone and nitrogen dioxide. *Sensors and Actuators B: Chemical*, 215, pp.249–257

The MiCS-6814 datasheet [online] Available at: https://www.mouser.co.uk/datasheet/2/18/1143_Datasheet-MiCS-6814-rev-8-1144828.pdf [Accessed 13 Dec. 2021].

World Health Organization (2021). *Ambient Air Pollution*. [online] www.who.int. Available at: https://www.who.int/teams/environment-climate-change-and-health/air-quality-and-health/ambient-air-pollution.