AN IOT BASED AIR POLLUTION MONITORING SYSTEM

A PROJECT REPORT

Submitted by

JENISHA LEBANA B	**(111520104063)**
KARTHIKA Y	**(111520104069)**
MAHALAKSHMI K N	**(111520104089)**
MADHUSHRI	**(111520104088)**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

R.M.D. ENGINEERING COLLEGE

(An Autonomous Institution)

KAVARAIPETTAI - 601206

DECEMBER – 2022

ANNA UNIVERSITY :: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this main project report **"AN IOT BASED AIR POLLUTION MONITORING SYSTEM"** is the bonafide work of **"JENISHA LEBANA.B (111520104063),KARTHIKA.Y(111520104069),MAHALAKSHMI.K.N(111520104089) ,MADHUSHRI (111520104088)"** who carried out the project work under my supervision

SIGNATURE	**SIGNATURE**
Dr.P.Ezhumalai M.Tech,F.I.E,Ph.D.,	**V.Sharmila B.E,M.E,(Ph.D).,**
HEAD OF THE DEPARTMENT	**SUPERVISOR,**
Department of Computer Science and Engineering,	Department of Computer Science and Engineering,
R.M.D Engineering College,	R.M.D Engineering College,
(Autonomous)	(Autonomous)
R.S.M. Nagar,	R.S.M. Nagar,
Kavaraipettai-601 206.	Kavaraipettai-601 206.

VIVA-VOCE EXAMINATION

The V i v a - Voce Examination of the following students who have submitted this project work held on………………

JENISHA LEBANA B	**(111520104063)**
KARTHIKA Y	**(111520104069)**
MAHALAKSHMI K N	**(111520104089)**
MADHUSHRI	**(111520104088)**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

It is our immense pleasure to express our deep sense of gratitude to our chairman **Thiru R. S. MUNIRATHINAM**, our vice chairman **Thiru R.M.KISHORE**, our director **Thiru R. JOTHI NAIDU**, and our Secretary **Thiru YALAMANCHI PRADEEP**, for the facilities and support given by them in the college.

We are extremely thankful to our principal **Dr. N. ANBUCHEZHIAN, B.E., M.S., M.B.A., M.E., Ph.D.,** for having given us an opportunity to Serve the purpose of education.

We take this opportunity to give profound and heartfelt thanks to the Dean-Research **Dr. K. K. THYAGHARAJAN B.E., M.E., Ph.D.**, for their continuous support in successful completion of this project.

We are indebted to **Dr.P.EZHUMALAIB.E., M.Tech., F.I.E., Ph.D.,** Professor and the Head of the Department of Computer Science and Engineering, for his valuable guidance and useful suggestions during the course of the project.

We are thankful to our project supervisor **V.SHARMILA B.E,M.E,(Ph.D),** Assistant Professor, Department of Computer Science and Engineering , R.M.D. Engineering College for her helpful guidance and valuable support given to us throughout the project.

ABSTRACT

Air pollution is a major concern in civilized world which causes toxicological to human health and environment. Existing monitoring systems have inferior precision, unaware of the region and require laboratory analysis. To overcome the problems of existing systems, proposing a three-phase air pollution monitoring system by using IoT along with cloud services to make the processing realtime and faster. The proposed system includes the design for monitoring air pollution and creating awareness among the public. An IoT kit is prepared with gas sensors, Arduino Integrated Development Environment, and a Wi-Fi module. This kit can be physically placed in various cities for monitoring air pollution. The gas sensors gather data from air and forward the data to the Arduino IDE. The Arduino IDE transmits the data to the cloud services via the Wi-Fi module. Developing a portable and user friendly android application termed IoT-Mobair, helps users to know the pollution level and notifies the highly polluted areas during travel. The proposed system is analogous to Google traffic or the navigation application of Google Maps.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACRONYM	ABBREVIATION
LPG	Liquefied Petroleum Gas
AQI	Air Quality Index
AQHI	Air Quality Health Index
GPRS	General Packet Radio Services
GUI	Graphical User Interface
API	Application Programming Interface
PPM	Parts Per million
IDE	Integrated Development Environment
IOT	Internet of Things
ESP32	Express If System
WHO	World Health Organization
CO	Carbon Monoxide
GSM	Global System for Mobile Communications
WSN	Wireless Sensor Network
GIS	Geographic Information System

CHAPTER 1

INTRODUCTION

1.1 GENERAL

Earlier the air we breathe in use to be pure and fresh. A breath of contemporary air has become quite unlikely for the densely inhabited Indian cities. The concentration of toxic gases is getting more toxic day by day in the atmosphere due to rapid industrialization. The atmosphere suffers from global warming and greenhouse effects. These particulates discharged can extremely contaminate the contemporary air. These contaminants can form a thick layer over the earth’s surface that causes several adventurous things.

Burning plastic releases several harmful gases, this can be a threat to the encompassing atmosphere and cause serious respiration downside in individuals. The thermal power stations considerably increase pollution by cathartic huge amounts of ash, carbon compounds, and SO2. The population explosionhas distended the reins of Indian cities, leading to a pathetic traffic conditions on the roads. Vehicles onthe road unharnessed gases and dangerous fumes by carelessly burning fossil oil and different fossil fuels.

Mobile application plays a prominent role in human life. It turned the impossible things to make it happen within fraction of seconds. The quality of the air can be determined through applications in different regions. The status of the air quality, temperature, humidity can easily be retrieved through web server applications.

Internet of things is increasingly growing topic in the business intelligence fields. It integrates data from the hardware devices and can be uploaded in cloud. It provides deeper automation, accuracy and analytical data. Internet of things makes things virtually ‘smart’ over the surroundings. IoT has become an emerging technology for active content, product, or service manageable activities. These applications take information or input from multiple devices and makes sufficient actions to convert into human readable form. They analyze data supported numerous settings and styles so as to perform automation-related tasks or give the information needed by business.

India is home to a fifth of the globe population and each year, nearly 2.5 million deaths are occurred due to pollution. The main pollutants of the air include CO, NO2, SO2, methane and smoke. The combination of these gases in the air exceeds the criteria set by the World Health Organization (WHO). The Central Pollution Control Board along with State Pollution Board determines the air quality for each and every year. The color code range can be drawn based on pollution levels. The polluted area can be reported on a daily basis.

1 .2 OBJECTIVE

- Determining the pollution level in real time computation
- Specific reports for air quality measures based on locations
- Air quality maps generation
- To guide the user during their travel about air quality in their route

CHAPTER 2
LITERATURE SURVEY

G. Lo Re, D. Peri, and S. D. Vassallo, "Urban air quality monitoring using vehicular sensor networks," in *Advances onto the Internet of Things,* Springer, 2014, pp. 311–323.

Urban air pollution has caused public concern worldwide because it seriously affects human life, including our health, living environment, and economy. According to the World Health Organization's report in 2012, 7 million premature deaths worldwide were related to air pollution. Environmental issues like global warming, acid rain, and haze are also caused by air pollution. Moreover, expenditure on public health is increasing rapidly due to excessive levels of air pollutants. The living things on earth and under water are suffering many problems like change in life due to lack of proper facilities of life.

In order to mitigate these issues, conventional monitoring systems have been installed in urban areas. These systems provide authorized information to the decision makers and public to enhance and manage the urban environment. However, they are suffering from the extremely sparse spatio-temporal resolution. For example, there are only 16 monitoring stations in Hong Kong covering an area about 2700 square kilometers and the pollution information is updated hourly.

Air pollution information with such low spatio-temporal resolution is inadequate for monitoring personal and acute exposures to air pollutants, which are proven to be critical to human health. The environmental monitoring can lead to a better recognition of a connection between polluters (pollution sources), pollution and health risks as well as necessary prevention.

A. De Nazelle, E. Seto, D. Donaire-Gonzalez, M. Mendez, J. Matamala, M. J. Nieuwenhuijsen, and M. Jerrett, "Improving estimates of air pollution exposure through ubiquitous sensing technologies," ***Environ. Pollut.*****, vol. 176, pp. 92–99, 2013.**

Air pollution exposure assessment for use in epidemiological or health impact assessment studies has traditionally relied on fixed-site monitoring stations to assign ambient air pollution levels to large populations. More recently researchers have employed land use regression or dispersion models to estimate small-area concentrations at home addresses of study subjects. Researching on this serious issue this system's main purpose was to estimate the quality of air for people and any other living thing which exist on earth. However, a person's activities throughout the day may result in variability in exposures and inhalation of air pollution that may be substantial and unaccounted for when only home address-based concentrations are estimated.

Several studies comparing personal exposure measurements to ambient monitoring at subject's home address have revealed at times large discrepancies between concentrations at residential addresses and personal exposure concentrations, and large variations from study-to-study and subject-to-subject. Other studies indicated that activity patterns are important determinants of personal exposures. Recently, have shown that transport activities contribute most to variability in personal exposures between people exposed to the same concentrations at home.

Further, estimating inhaled dose of air pollution by accounting for energy expenditure can change the relative ranking of exposures as a function of activity patterns compared to using solely exposure concentrations. Thus, exposure measurement errors introduced by failing to account for mobility and inhalation may lead to bias, loss of power, or both in health effects estimates.

R. Peterová and J. Hybler, "Do-it-yourself environmental sensing," ***Procedia Comput. Sci.*****, vol. 7, pp. 303–304, 2011.**

Air pollution is one of the most important factors currently affecting qualityof life in big cities. However, despite the fact that poor air quality has been shown to directly affect human health, our daily exposure to such pollutants has been inadequately captured and publicly shared. There are state agencies measuring the air quality and informing citizens in the press or on their websites, but usually individuals are not fully aware of their personal exposure, either immediate or long-term. However, the lack of awareness is not only caused by ineffective methods of communication from the state agencies; it is also the type of data that their fixed sensors provide.

Participatory Sensing is described as a revolutionary new paradigm that allows people to voluntarily sense their environment using readily available sensor devices such as smart phones, and share this information using existing cellular and internet communication infrastructure. Democratization of technology, low-cost sensors and Do-It-Yourself (DIY) hardware prototyping platforms, have the potential to enable everyday citizens to develop and use personalized air quality sensing tools and turn their mobile phones or PDAs to measuring devices enabling us to learn about our environment. These innovative sensing capabilities bring new possibilities for individuals to take part in air-quality monitoring as well as to raise awareness of environmental issues.

It also brings new possibilities for citizen science so that researchers can make use of the potential of masses and collect data in larger quantities from locations that were never monitored before. It is a hardware and software initiative to bring pollution monitoring closer to everyday citizens-scientists and give them a tool to measure their environment on everyday basis.

CHAPTER 3
SYSTEM ANALYSIS

EXISTING SYSTEM

The commercial meters available in the market are Fluke CO- 220 carbon monoxide meter for CO, Amprobe CO2 meter for CO2, ForbixSemicon LPG gas leakage sensor alarm for LPG leakage detection. The researchers in this field have proposed various air quality monitoring systems based on WSN, GSM and GIS. Now each technology has limited uses according to the intended function, as Zigbee is meant for users with Zigbee trans-receiver, Bluetooth.

GIS based pinpoints of air pollution of any area. It consists of a microcontroller, gas sensors, mobile unit, a temporary memory buffer and a web server with internet connectivity which collects data from different locations along with coordinate’s information at certain time of a day. The readings for particular location are averaged in a closed time and space. The recorded data is periodically transferred to a computer through a GPRS connection and then the data will be displayed on the dedicated website with user acceptance.

DRAWBACKS

- Inferior precision
- Low sensitivity
- Require laboratory analysis
- Inconvenience of calibration & maintenance
- Percentage of accuracy is less
- Robustness is not achieved

PROPOSED SYSTEM

The proposed approach detects the concentration of air pollutants in the area of interest via sensors. The sensed data is collected in Arduino and transmitted through Wi-Fi module which is stored in cloud - Ubidots. The uploaded data is retrieved by user-friendly android application - MOBAIR helps the user to know the pollution level in the region.

The routes are drawn from source to destination and pollution level is estimated in real time computation. The quality of the air is predicted and analyzed using analytical module based on air quality index levels. While traveling, notification message will be appeared to the user by the comparison of pollution rate in the intermediate waypoints.

ADVANTAGES

- Remote, real time measurement monitoring
- Detecting wide range of gases like CO2, CO, CH4, smoke
- Simple, compact, easy to handle
- Continuous update of percentage in air quality
- Notifies the highly polluted areas
- More accurate data for analysis and decision making based on locations

CHAPTER 4
SYSTEM REQUIREMENTS

Software Requirement Specification is the requirements that formally specifies the system-level requirements of a single system or an application. It identifies, defines and clarifies the requirements, that when satisfied through development meet the operational/functional need identified in the Project Concept Proposed, Project Business Case, and Project Charter. Approval of this document constitutes agreement that the developed system satisfies these requirements at the topmost level.

HARDWARE REQUIREMENTS

- MQ2 methane sensor
- MQ3 carbon monoxide sensor
- MQ135 air quality sensor
- ESP32 development board
- Bread board
- Connecting wires
- Voltage regulator
- Channel 4 level shifter

SOFTWARE REQUIREMENTS

- Arduino IDE 1.8.9
- Android Studio 3.3
- Ubidots cloud service framework

CHAPTER 5
SOFTWARE SPECIFICATION

ARDUINO

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board.

The Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on processing. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language

can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

UBIDOTS CLOUD SERVICE FRAMEWORK

Ubidots is an Internet of Things (IoT) data analytics and visualization company. Turn sensor data into information that matters for business- decisions, machine to machine interactions, educational research and increase economization of global resources. Ubidots exists as an easy and affordable means to integrate the power of the IoT into your business or research. Ubidots technology and engineering stack was developed to deliver a secure, white-glove experience for our users.

Device friendly APIs (accessed over HTTP/MQTT/TCP/UDP protocols) provide a simple and secure connection for sending and retrieving data to and from our cloud service in real-time. Ubidots time-series backend services are optimized for IoT data storage, computation and retrieval. The application enablement platform supports interactive, real time visualization (widgets) and an IoT App Builder that allows to extend the platform with own HTML languages. Ubidots exists to empower the data from device to visualization in the form of charts and graphs.

ANDROID STUDIO

- **Install and Configure Android Studio**

Android is based on Linux with a set of native core C/C++ libraries. Android applications are written in Java. However, they run on Android's own JVM, called DVM (instead of JDK's JVM) which is optimized to operate on the small and mobile devices. Installing Android software is probably the most challenging part of this project - for the unlucky ones.

It takes times - from 30 minutes to *n* hours to forever - depending on PC. It probably need a PC (with 8GB RAM) and 10GB of free disk space to run the Android emulator!!! Running on actual Android devices (phone, tablet) requires much lesser resources. Install “Android Studio IDE” and “Android SDK” as followingsteps.

- **Pre-Installation Check List**

Step-1: Before installing Android SDK, you need to install JDK. Read "How to install JDK". Ensure that your JDK is at or above 1.8. You can check your JDK version with command "`javac -version`".

Step-2: Uninstall older version(s) of "Android Studio" and "Android SDK", if any.

Step-3: The installation and many operations take a longtime to complete. Do not stare at your screen or at the ceiling.

Step-4: We need to install:

1. Android Studio IDE, which is based on IntelliJ (a popular Java IDE); and
2. Android Software Development Kit (SDK).

Step-5: Click on the setup file, you will be greeted by the window shown below, click "**Next** ".

Step-6: The next screen will ask you if you want the standard Android Studio install or you want to customize your own settings and choose from a list of components. For the sake of this guide, we will stick with the standard option and click on "**Next** ".

Step-7: In the next two screens, it will ask you about the location of Android Studio Installation. We have stuck with the recommended directory. Click "Next" and then you will be asked where to put the Android Studio shortcut. Finally, click Install.

Step-8: Once you do click Install, the setup will proceed to extract all the necessary files. Let it finish extracting. Once it completes the procedure, just click on "**Finish** ".

- **Creating Application in Android Studio**

  Step-1: Launch Android Studio.

  Step-2: Choose "Start a new Android Studio Project".

  Step-3: In "Create Android Project" dialog ⇒ Set "Application Name" to "Hello Android" (this will be the "Title" in your phone's App menu) ⇒ Set your "Company Domain" to "example.com" ⇒ In "Project Location", choose your project directory, e.g., d:\myProject or use the default location ⇒ Do NOT check the "Include C++ Support" ⇒ Next.

  Step-4: In "Target Android Devices: Select the form factor and minimum SDK" ⇒ Check "Phone and Tablet" (default) ⇒ In "Minimum SDK", choose "API 15: Android 4.0.3 (IceCreamSandwich)" (default) ⇒ Leave all of the other options (TV, Wear, and Glass) unchecked (default) ⇒ Next.

Step-5: In "Add an activity to Mobile" dialog ⇒ Select "Empty Activity" (default) ⇒ Next.

Step-6: In "Configure Activity" dialog ⇒ Set "Activity Name" to "Main Activity" (default) ⇒ Set "Layout Name" to "activity main" (default) ⇒ Next ⇒ Finish.

Step-7: Be patient! It could take a while to set up your first app. Watch the "progress bar" at the bottom status bar.

Step-8: Once the progress bar indicates completion, a hello-world app is created by default.

- **Setup Emulator (AVD)**

To run your Android app under the emulator, you need to first create an AVD. An AVD models a specific device (e.g., your iPhone). You can create AVDs to emulate different android devices (e.g., phone/tablet, android version, screen size, and etc.).

Step-1: In Android studio, select "Tools" ⇒ Android ⇒ AVD Manager. See "Common Errors" below if you cannot find "AVD manager".

Step-2: Click "Create Virtual Device".

Step-3: In "Select Hardware: Choose a device definition" dialog ⇒ In "Category", choose "Phone" ⇒ In "Name", choose "2.7 QVGA" (the smallest device available - you can try a bigger device later) ⇒ Next.

Step-4: In "System Image: Recommended" ⇒ Select the version with the highest API level ⇒ Click "Download" ⇒ Next.

Step-5: In "AVD Name", enter "2.7 QVGA API 27" (default) ⇒ Finish.

- **Run The Application**

Step-1: Select the "Run" menu ⇒ "Run app" ⇒ Check "Launch Emulator" ⇒ Select "2.7 QVGA API 27" ⇒ OK.

Step-2: You MAY BE prompted to install Intel HAXM (Hardware Accelerated Execution Manager). Follow the instruction to install HAXM.

Step-3: Be patient! It may take a few MINUTES to fire up the app on the emulator. You first see a Google logo ⇒ then "Android" ⇒ then the "wallpaper" ⇒ then the message. Goto next step.

Step-4: Do not close the emulator, as it really takes a long time to start. You could always re-run the app (or run a new app) onthe same emulator. Try re-run the app by selecting "Run" menu ⇒ "Run app".

Description Front-End

In computing, XML is a mark-up language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The W3C's XML 1.0 Specification and several other related specifications all of them free open standards—define XML.

Several schema systems exist to aid in the definition of XML-based languages, while programmers have developed many APIs) to aid the processing of XML data. The design goals of XML include, "It shall be easy to write programs which process XML documents". Despite this, the XML specification contains almost no information about how programmers might go about doing such processing.

The XML Info set specification provides a vocabulary to refer to the constructs within an XML document, but does not provide any guidance on how to access this information. A variety of APIs for accessing XML have been developed and used, and some have been standardized.

Description of Back-End

- **Java Technology**

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes — the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

- **The Java Platform**

A platform is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows

2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms. The JVM:

- The API
- JVM is the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as GUI widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code.

However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

CHAPTER 6
SYSTEM DESIGN

System design is the process of planning a new system to complement or altogether replace the old system. The purpose of the design phase is the first step in moving from the problem domain to the solution domain. The design of the system is the critical aspect that affects the quality of the software. System design is also called top-level design. The design phase translates the logical aspects of the system into physical aspect of the system.

SYSTEM ARCHITECTURE

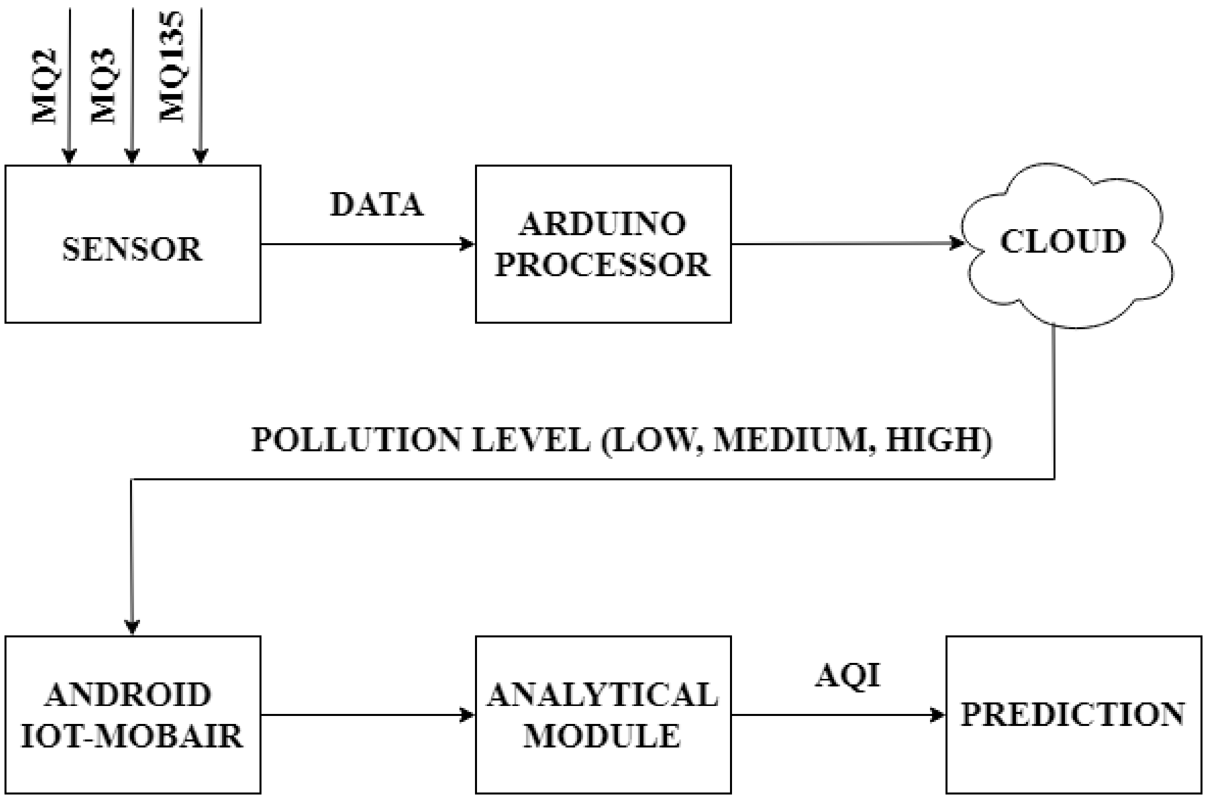


Figure 6.1 Architecture Design of IoT Based Air Pollution Monitoring System.

ARDUINO PIN DIAGRAM

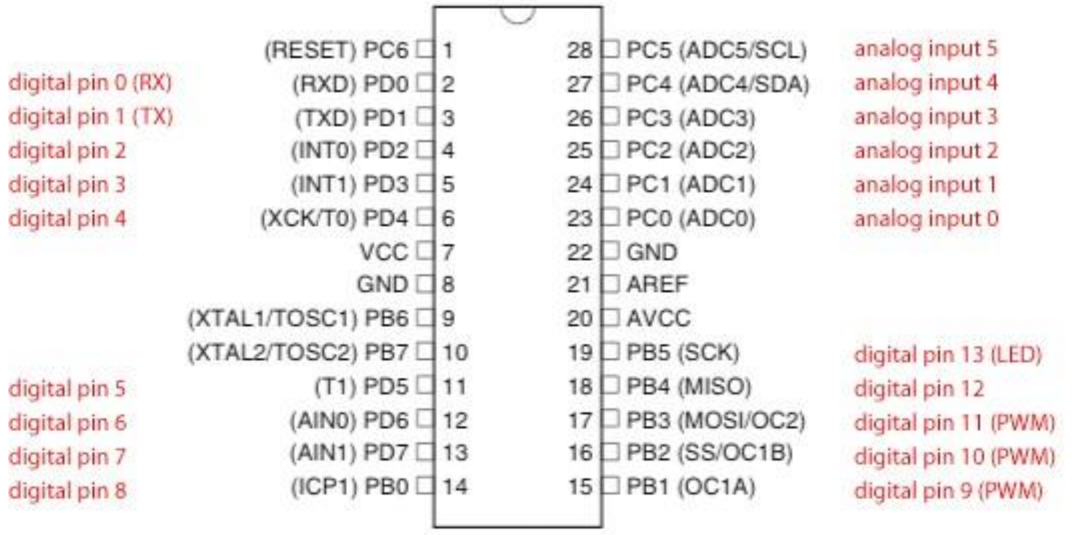
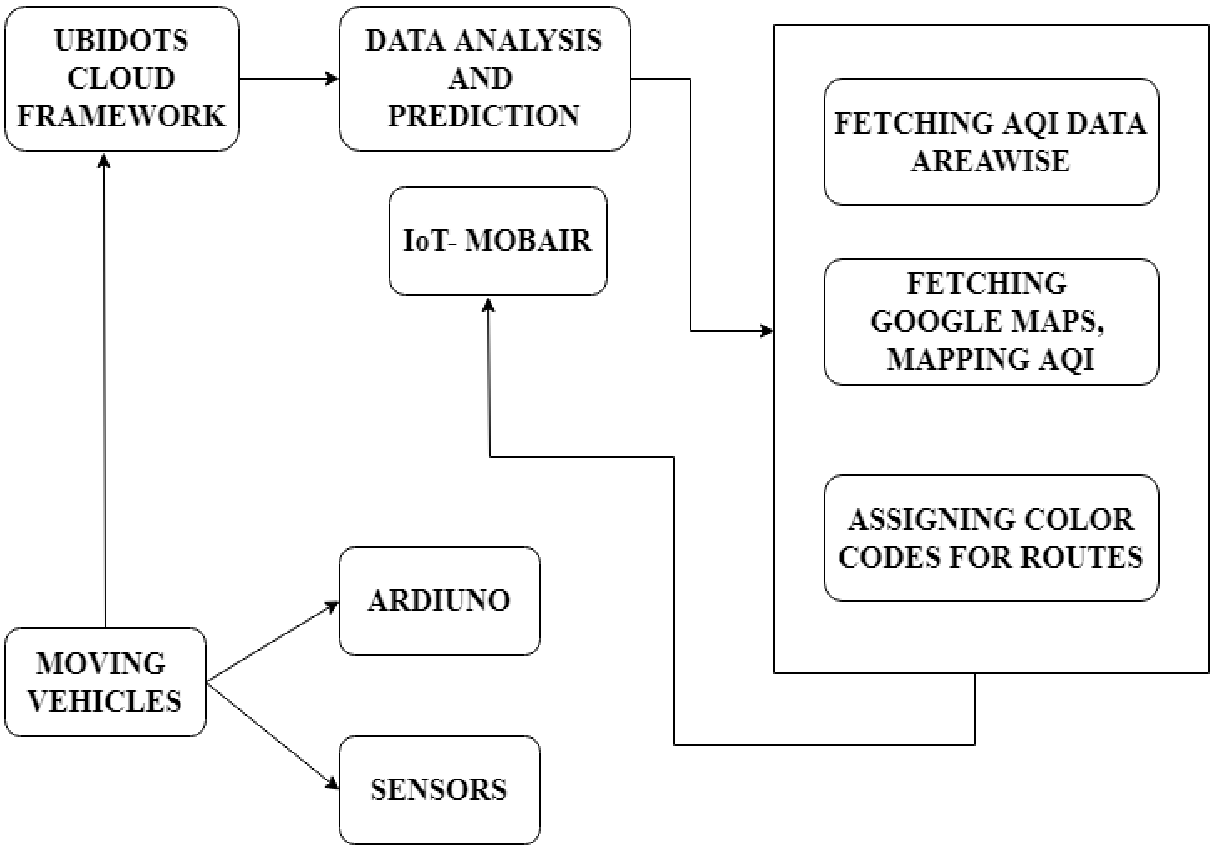


Figure 6.2 Arduino Pin Diagram

IoT MOBAIR DESIGN



UBIDOTS CLOUD FRAMEWORK
DATA ANALYSIS AND PREDICTION
FETCHING AQI DATA AREAWISE
IoT- MOBAIR
FETCHING GOOGLE MAPS, MAPPING AQI
ASSIGNING COLOR CODES FOR ROUTES
ARDIUNO
MOVING VEHICLES
SENSORS

Figure 6.3 IoT Mobair Design Overview

CHAPTER 7
SYSTEM IMPLEMENTATION

AIR QUALITY SENSING MODULE

The Internet of Things (IoT) is a system of connecting hardware devices along with software applications. IoT provides a communication between applications and components and that are supplied with distinctive identifiers. It facilitates to transfer knowledge over a network while not requiring human-to-human or human-to-computer interaction.

Air pollution detection kit is developed using IoT [MQ2, MQ3, MQ135, and ESP32]. The sensors sense the real time air pollution level in particular region. The MQ2 sensor detects dangerous gases like LPG, i-butane, paraffin, propane, methane gases which are paired up in the atmosphere. Carbon monoxide is a colorless gas which reduces the hemoglobin content in human body.

Such substances are monitored real time through MQ3 carbon monoxide sensor up to 20 to 2000 ppm in the atmosphere. The pollutants like alcohol, acetone, formaldehyde which causes greenhouse effects are detected using MQ135 air quality sensor. The ESP32 development board is a combination of Arduino IDE, Wi-Fi module. It helps to connect the network to send the sensor readings to the cloud.

- **Detection of air pollutants levels**

The data is generated from the gas sensors and concentration is obtained from the air. Table 1 shows the normal range of the gases. Air contamination can be characterized as the outflow of harmful substances to the air. The main components of pollutants are:

Sensor	Gas	Description	Range
MQ7	Carbon Monoxide	Suitable for sensing CO concentration in the air.	0-100 (No effect) 100-800 (Risky) 800-2000 (Very high)
MQ2	Methane	Useful for gas leakage detection in home and industry. Detect LPG, Butane, smoke.	0-1000 (Normal) 1000-15000 (Risky) 15000-50000 (Very high)
MQ135	Air Quality	Responsive to a wide scope of harmful gases like alcohol, acetone, thinner, formaldehyde and so on	0-500 (Normal) 500-1500 (Risky) 1500-2000 (Very high)

Table 1 Sensor Details

VISUALIZATION MODULE

Ubidots exists as a smooth and affordable means to combine the energy of the IoT hardware resources. The sensor readings from Arduino IDE are uploaded to cloud service framework - Ubidots. It provides user- friendly protocols to store and access the data. The sensed data is uploaded to cloud via ESP32 development kit.

The real time air quality percentage and levels are continuously updated in the Ubidots portal. The uploaded data can be visualized as per user perception in the form of ranges and charts. It also generates the color code based on authorized levels and determines AQI.

- **Detecting Air Quality Index**

An air quality-rate index (AQI) is used by government corporations to talk to the public how polluted the air presently is or how polluted its miles forecast to become.

Values	**Color**	**Description**
0 - 50	🟩	Good
50 - 100	🟨	Moderate
100 - 150	🟧	Unhealthy for sensitive groups
150 - 200	🟥	Unhealthy
200 - 250	🟪	Very unhealthy
300 - 500	🟥	Hazardous

Table 2 AQI Values

ANALYTICAL MODULE

Recently, improvements in smartphone technology have changed the importance of cellular telephones. A phone is not just used for communicating but has also become a crucial part of everyone’s daily life. Presently the electronic market is acquired by Android technology.

Over time, smartphones and the Android system have become more prevalent. In this work, we used Java language, the Android Studio platform, Android ADT, and the Android SDK to develop the IoT-Mobair android application. The IoT-Mobair application uses user location data (via GPS system), the Internet of Things (IoT), sensors, and standard websites to give air quality data. If the user travels from source to destination, the entire route pollution rate is monitored and alerts a notification message. If the pollution rate is huge, the user can re-route their journey. The following steps are undertaken in order to accomplish these tasks:

Step 1: Develop Android client

To handle the HTTP connection to the *Ubidots* server, an Android client is developed. For security purposes, the authentication token is used.

Step 2: Handle JSON format to read data:

After requesting remote services by the android application, a JSON response is received that is analysed to extract data, i.e., the variable value (sensor information), timestamp, and AQI.

Step 3: Detecting Air Pollution Level

In India, the government reports air quality as AQI (Air Quality Index). The main components for indexing are: particulate matter, methane (CH_4), and carbon monoxide (CO). From the cloud, the concentration of these pollutants is retrieved. The Indian government has set standards to provide appropriate warnings messages when the pollutants increase above a specific level. The AQI-colour code is shown in Table 2.

Step 4: Creating the front-end Android interface

- Draw a route that shows the user their exposure to air pollution.
- Display the data collected by IoT over app using google maps with pollution colour codes mentioned in table 2 through big data predictive analytics.

Step 5: Draw route from source to destination

The routes are drawn from source to destination and pollution level is estimated in real time computation. The quality of the air is predicted and analysed in mobile application based on air quality index levels. For a user traveling from a source to a destination, the pollution level of the entire route is predicted and a warning is displayed if the pollution level is too high so that the user can re-route his journey.

The data collected from the sensors and other trusted websites is made as are placed in a large database. When the user enters his destination of travel, the IoT-Mobair android application first converts the address into corresponding latitude/longitude form. The latitude and longitude are searched in the cloud-based database. Intermediate places between the starting and finishing location are also displayed. Suitable colors as shown in Table 2 are used to indicate the pollution level on the map.

Step 6: Prediction and analysis

Historical data can be used to predict pollution levels for subsequent days. Notification message will be appeared to the user by the comparison of pollution rate in the intermediate waypoints which includes:

- Location ID
- Latitude
- Longitude
- Concentration of CO
- Concentration of CH4
- Concentration of smoke
- AQI

The notification message helps the user to know whether the route is “safe or unsafe”. If the pollution rate tends to be low or moderate, the app conveys the message as safer to travel to the user. If the pollution rate is too high, the user can reroute the journey.

CHAPTER 8
CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

Pollution in earlier days was negligible. Due to rapid industrialization and advanced technology, air pollution reached the extreme limit of harmfulness. In order to protect humans from respiratory diseases and safeguard natural resources, the air pollution detection kit is physically placed in different locations to determine pollution level. The IoT kit is associated with android application IoT-Mobair that helps the users to obtain the pollution rate in real time computation. Further, data logging can be used to predict AQI levels. Specific reports for air quality measures based on locations and air quality maps generation are the main features of IoT-Mobair.

FUTURE ENHANCEMENT

In the hardware device it can be added light system. Light system will be work like automatic way. Such as, there are four lights for four types of gases. While a particular sensor detects the gas for that sensor, the related light beside that gas will be on and while the sensor stops getting that particular gas the light will be off automatically.

The proposed system faces with computational complexity particularly when we are dealing with big sensor data. One solution could be using fog computing, instead of cloud computing to reduce computation complexity and enhance the performance of the system. It can also be implemented with zero tolerance fast big data real-time stream analytical tools to process such a complex system. Many cities and their pollution rates can be added dynamically.

APPENDICES

A1 SOURCE CODE

ARDUINO

```
#include <WiFi.h>
#include<PubSubClient.h>
#define WIFISSID "FactoryForward"
#define PASSWORD "1234567809"
#define TOKEN "BBFF-8mPhQJZ0f1Mll99oy7V9Drkdf2NaG8"
#define MQTT_CLIENT_NAME "mymqttclientname"
#define VARIABLE_LABEL1 "MQ135"
#define VARIABLE_LABEL2 "MQ3"
#define VARIABLE_LABEL3 "MQ2"
#define DEVICE_LABEL "esp32"
#define SENSOR1 34
#define SENSOR2 35
#define SENSOR3 32
char mqttBroker[] = "industrial.api.ubidots.com";
char payload[100];
char topic[150];
char str_sensor[10];
WiFiClient ubidots;
PubSubClient client(ubidots);
void callback(char* topic, byte* payload, unsigned int length) {
  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;
  String message(p);
  Serial.write(payload, length);
```

```
 Serial.println(topic); }
void reconnect() {
 while (!client.connected()) {
  Serial.println("Attempting MQTT connection...");
 if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {
   Serial.println("Connected");
  } else {
   Serial.print("Failed, rc=");
   Serial.print(client.state());
   Serial.println(" try again in 2 seconds");
   delay(2000);
  }
 }
}
void setup() {
 Serial.begin(115200);
 WiFi.begin(WIFISSID, PASSWORD);
 pinMode(SENSOR, INPUT);
 Serial.println();
 Serial.print("Wait for WiFi...");
 while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(500);
 }
 Serial.println("");
 Serial.println("WiFi Connected");
 Serial.println("IP address: ");
 Serial.println(WiFi.localIP());
```

```
  client.setServer(mqttBroker, 1883);
  client.setCallback(callback); }
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
  sprintf(payload, "%s", "");
  sprintf(payload, "{\"%s\":", VARIABLE_LABEL1);
   float mq135 = analogRead(SENSOR1);
  float mq3 = analogRead(SENSOR2);
  float mq2 = analogRead(SENSOR3);
  dtostrf(mq135, 4, 2, str_sensor);
  sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor);
  Serial.println("Publishing data to Ubidots Cloud");
  Serial.println(payload);
  client.publish(topic, payload);
  client.loop();
  delay(1000);
}
```

ANDROID STUDIO

```
  package com.sensorcon.airqualitymonitor;
  import java.util.ArrayList;
  import com.sensorcon.airqualitymonitor.database.DBDataBlob;
  import com.sensorcon.airqualitymonitor.database.DBDataHandler;
  import android.os.Bundle;
```

```
import android.preference.PreferenceManager;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.NotificationManager;
import android.app.Notification.Builder;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.graphics.Color;
import android.view.Gravity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.ImageView.ScaleType;
import android.widget.RelativeLayout;
import android.widget.TextSwitcher;
import android.widget.TextView;
import android.widget.ViewSwitcher.ViewFactory;
public class AirQualityMonitor extends Activity {
```

```
public Button btnStart;
public Button btnStop;
public Button btnStatus;
private DBDataHandler dbHandler;
NotificationManager notifier;
Builder notifyTest;
SharedPreferences myPreferences;
Editor prefEditor;
TextSwitcher aqStatus;
TextSwitcher tsMq2;
TextSwitcher tsMq3;
TextSwitcher tsMq135;
TextSwitcher tsTimeStamp;
TextSwitcher tsGasData;
ViewFactory ctvGasFactory;
ViewFactory tvDataFactory;
ViewFactory tvTimeFactory;
Animation in;
Animation out;
ViewFactory faceFactory;
ImageSwitcher faceSwitcher;
Activity myActivity;
Context myContext;
ImageView ivInfo;
boolean faceAnimateToggle;
boolean isMeasuring;
public void setIsMeasuring(boolean status) {
        this.isMeasuring = status;
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_air_quality_monitor);
    RelativeLayout mainLayout = (RelativeLayout)
findViewById(R.id.mainLayout);
    mainLayout.setBackgroundResource(R.drawable.bg_gradient);
    myActivity = this;
    myContext = this;
    myPreferences = PreferenceManager
    prefEditor = myPreferences.edit();
    faceFactory = new ViewFactory() {
    @Override
    public View makeView() {
    ImageView fView = new ImageView(getApplicationContext());
    fView.setScaleType(ScaleType.CENTER);
    fView.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
    if (!isMeasuring) {
    takeMeasurement();
    }
    });
    return fView;
    };
```

```
in = AnimationUtils.loadAnimation (this, android.R.anim.slide_in_left);
out = AnimationUtils.loadAnimation(this, android.R.anim.slide_out_right);
tvDataFactory = new ViewFactory() {
@Override
public View makeView() {
TextView tv = new TextView(getApplicationContext());
        tv.setGravity(Gravity.RIGHT);
        tv.setTextColor(Color.BLACK);
        tv.setTextSize(20);
        return tv;
        }
};
    aqStatus = (TextSwitcher)findViewById(R.id.tsAQStatus);
    aqStatus.setInAnimation(in);
    aqStatus.setOutAnimation(out);
    aqStatus.setFactory(tvDataFactory);
    faceSwitcher = (ImageSwitcher)findViewById(R.id.isFace);
    faceSwitcher.setFactory(faceFactory);
    faceSwitcher.setInAnimation(this, android.R.anim.fade_in);
    faceSwitcher.setOutAnimation(this, android.R.anim.fade_out);
    tvTimeFactory = new ViewFactory() {
    @Override
    public View makeView() {
        TextView tv = new TextView(getApplicationContext());
            tv.setGravity(Gravity.LEFT);
            tv.setTextColor(Color.BLACK);
        tv.setTextAppearance(getApplicationContext(),
                        return tv;
                }
        };
        Tsmq2 = (TextSwitcher)findViewById(R.id.textSwitcher1);
        Tsmq3 = (TextSwitcher)findViewById(R.id.textSwitcher2);
        Tsmq135 = (TextSwitcher)findViewById(R.id.textSwitcher3);
        tsTimeStamp = (TextSwitcher)findViewById(R.id.tsLastUpdate);
        Tsmq2.setInAnimation(in);
        Tsmq3.setInAnimation(in);
        Tsmq135.setInAnimation(in);
        tsTimeStamp.setInAnimation(in);
        Tsmq1.setOutAnimation(out);
        Tsmq2.setOutAnimation(out);
        Tsmq135.setOutAnimation(out);
        tsTimeStamp.setOutAnimation(out);
        tsMq2.setFactory(tvDataFactory);
        tsMq3.setFactory(tvDataFactory);
        tsMq135.setFactory(tvDataFactory);
        tsTimeStamp.setFactory(tvTimeFactory);
        ctvGasFactory = new ViewFactory() {
@Override
public View makeView() {
CircularTextView ctv = new CircularTextView(getApplicationContext());
        ctv.setTextSize(20);
        ctv.setTextColor(Color.BLACK);
        return ctv;
        };
        tsGasData = (TextSwitcher)findViewById(R.id.tsGasData);
        tsGasData.setInAnimation(this, android.R.anim.fade_in);
        tsGasData.setOutAnimation(this, android.R.anim.fade_out);
        tsGasData.setFactory(ctvGasFactory);
        dbHandler = new DBDataHandler(getApplicationContext());
        ivInfo = (ImageView)findViewById(R.id.ivInfo);
        ivInfo.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View v) {
                showAQInfo();
                }
        });
}
public void updateDisplay() {
        dbHandler.open();
        ArrayList<DBDataBlob> dbData = dbHandler.getAllData();
        dbHandler.close();
        int nEntries = dbData.size();
        int lastItem = nEntries -1;
        if (nEntries == 0) {
        tsMq2.setText("N/A");
        tsMq3.setText("N/A");
        tsMq135.setText("N/A");
        tsGasData.setText("N/A");
                ((CircularTextView)tsGasData.getChildAt(0)).setStatusNull();
                ((CircularTextView)tsGasData.getChildAt(1)).setStatusNull();
        tsTimeStamp.setText("Last measured: N/A");
```

```
                aqStatus.setText("N/A");
                AlertDialog alert;
                AlertDialog.Builder builder = new AlertDialog.Builder(myContext);
                builder.setTitle("No Data Found");
                builder.setMessage("No stored data was found! Select \"Take a
Measurement\" from the menu, " + "or click the face to get a reading!");
                builder.setPositiveButton("Measure Now!", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
                takeMeasurement();
                }
                });
        builder.setNegativeButton("OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
                        });
                        alert = builder.create();
                        alert.show();
                        faceSwitcher.setImageResource(R.drawable.face_unknown);
                        return;
                }
        int tempPref = myPreferences.getInt(Constants.MQ2_UNIT, POINTER);
        if (tempPref != Constants.POINTER) {
                long pointer = dbData.get(lastItem).getDbMq2().getValue();
                double range = pointer * (9.0 / 5.0) + 32;
                tsMq2.setText(String.format("%.0f", range) + " F");
```

```
          } else {
          tsMq2.setText(dbData.get(lastItem).getDbMq2().toString() + " C");
          }
          float pValue = dbData.get(lastItem).getDbMq135().getValue();
int pPref = myPreferences.getInt(Constants.MQ135_UNIT, PASCAL);
if (pPref == Constants.HECTOPASCAL) {
          tsMq135.setText(String.format("%.2f", pValue/100) + " hPa");
          } else if (pPref == Constants.KILOPASCAL) {
          tsMq135.setText(String.format("%.3f", pValue/1000) + " kPa");
          } else if (pPref == Constants.ATMOSPHERE) {
          tsMq135.setText(String.format("%.3f", pValue * 0.4095) + " atm");
          } else if (pPref == Constants.MMHG) {
          tsMq135.setText(String.format("%.0f", pValue * 0.4095) + " ppm");
          } else if (pPref == Constants.INHG) {
          tsMq135.setText(String.format("%.2f", pValue * 0.4095) + " ppm");
          }
          else {
          tsMq135.setText(dbData.get(last).getDbMq135().toString() + " Pa");
          }
          tsMq3.setText(dbData.get(lastItem).getDbMq3().toString() + " %");
          String gasData = "";
          if (dbData.get(lastItem).getDbCO2().getValue() != -1) {
          gasData = dbData.get(last).getDbCO2().toString() + " ppmCO2\n";
          }
          gasData += dbData.get(lastItem).getDbCO().toString()+ " ppm CO";
          tsGasData.setText(gasData);
```

```
                tsTimeStamp.setText("Last measured: " +
dbData.get(lastItem).getDbDateTime().getMMDDYYYY().getTimeStamp());
                assessQuality(dbData.get(lastItem).getStatus());
        }
        public void assessQuality(int level) {
                if (level == Constants.STATUS_GOOD) {
                        aqStatus.setText("Good");
                ((CircularTextView)tsGasData.getChildAt(0)).setStatusGood();
                ((CircularTextView)tsGasData.getChildAt(1)).setStatusGood();
                faceSwitcher.setImageResource(R.drawable.face_good);
                } else if (level == Constants.STATUS_MODERATE) {
                        aqStatus.setText("Moderate");
                ((CircularTextView)tsGasData.getChildAt(0)).setStatusModerate();
                ((CircularTextView)tsGasData.getChildAt(1)).setStatusModerate();
                        faceSwitcher.setImageResource(R.drawable.face_moderate);
                } else if (level == Constants.STATUS_BAD) {
                        aqStatus.setText("Bad");
                                ((CircularTextView)tsGasData.getChildAt(0)).setStatusBad();
                                ((CircularTextView)tsGasData.getChildAt(1)).setStatusBad();
                        faceSwitcher.setImageResource(R.drawable.face_bad);
                } else {
                }
}
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                getMenuInflater().inflate(R.menu.air_quality_advanced, menu);
                return true;
```

```
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
        case R.id.history:
                Intent showHistory = new Intent(this, HistoryActivity.class);
                startActivity(showHistory);
                break;
        case R.id.action_settings:
                Intent showPrefs = new Intent(this, PreferenceActivity.class);
                startActivity(showPrefs);
                break;
        case R.id.refresh:
                updateDisplay();
                break;
        case R.id.measure:
                takeMeasurement();
                break;
        case R.id.mainHelp:
                TxtReader help = new TxtReader(myContext);
                help.displayTxtAlert("About", R.raw.main_help);
                break;
        case R.id.aqInfo:
                showAQInfo();
                break;
        }
        return true;
        }
        public void showAQInfo() {
                AlertDialog alert;
                AlertDialog.Builder builder = new AlertDialog.Builder(myContext);
                builder.setTitle("Air Quality Information");
                builder.setMessage("sensor Levels");
                builder.setPositiveButton("Carbon Monoxide", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                                Intent coIntent = new Intent(myContext, COInfo.class);
                                startActivity(coIntent);
                        }
                });
                builder.setNegativeButton("Carbon Dioxide", new
DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                        Intent co2Intent = new Intent(myContext, CO2Info.class);
                                startActivity(co2Intent);
                        }
                });
                alert = builder.create();
                alert.show();
        }
        public void takeMeasurement() {
                String MAC = myPreferences.getString(Constants.SD_MAC, "");
                if (MAC.equals("")) {
                        // Launch settings activity
                        Intent setupDrone = new Intent(getApplicationContext(),
PreferenceActivity.class);
                        setupDrone.putExtra(Constants.NEEDS_SETUP, true);
                        startActivity(setupDrone);
                        return;
                }
                faceSwitcher.setImageResource(R.drawable.face_unknown);
                DataSync getData = new DataSync(getApplicationContext(),
AirQualityMonitor.this);
                getData.setSdMC(MAC);
                getData.setContext(myContext);
                getData.execute();
        }
        @Override
        protected void onResume() {
                super.onResume();
                updateDisplay();
        }
        public void animateFace() {
                if (faceAnimateToggle) {
                        faceSwitcher.setImageResource(R.drawable.face_unknown_1);
                } else {
                        faceSwitcher.setImageResource(R.drawable.face_unknown_2);
                }
                faceAnimateToggle = !faceAnimateToggle; }}
```

DIRECTION WAYPOINTS

```
<!DOCTYPE html>
<html>
 <head>
   <meta name="viewport" content="initial-scale=1.0, user-scalable=no">
   <meta charset="utf-8">
   <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
   <title>IOT Mobair</title>
   <style>
     #right-panel
      {
       font-family: 'Roboto','sans-serif';
       line-height: 30px;
       padding-left: 10px;
       }
      #right-panel select, #right-panel input
     {
       font-size: 15px
      }
      #right-panel select
     {
       width: 100%;
      }
      #right-panel i
     {
       font-size: 12px;
     }
     html, body {
```

```
  height: 100%;
  margin: 0;
  padding: 0;
 }
 #map
{
  height: 100%;
  float: left;
  width: 70%;
  height: 100%;
 }
 #right-panel
{
  margin: 20px;
  border-width: 2px;
  width: 20%;
  height: 400px;
  float: left;
  text-align: left;
  padding-top: 0;
 }
 #directions-panel
 {
  margin-top: 10px;
  background-color: #FFEE77;
  padding: 10px;
  overflow: scroll;
  height: 174px;
```

```
    }
      #mypanel
    {
      margin-top: 10px;
      background-color: #2180645c;
      padding: 10px;
      overflow: scroll;
      height: 174px; }
  </style>
</head>
<body>
  <div id="map">
  </div>
  <div id="right-panel">
  <div>
  <b>Start:</b>
  <select id="start">
    <option value="Hosur, India">Adhiyamaan CSE Station, Hosur</option>
    <option value="Vellore, India">Vellore</option>
    <option value="Chennai, India">Chennai</option>
    <option value="Goa, India">Goa</option>
  </select>
  <br>
  <b>Waypoints:</b> <br>
  <i>(Ctrl+Click or Cmd+Click for multiple selection)</i> <br>
  <select multiple id="waypoints">
    <option value="Vellore, India">Vellore</option>
    <option value="Krishnagiri, India">Krishnagiri</option>
```

```
 </select>
 <br>
 <b>End:</b>
 <select id="end">
   <option value="Chennai, India">Chennai</option>
   <option value="Kanchipuram, India">Kanchipuram</option>
   <option value="Vellore, India">Vellore</option>
   <option value="Goa, India">Goa</option>
 </select>
 <br>
   <input type="submit" id="submit">
 </div>
 <div class="mypanel">
</div>
 <div id="directions-panel">
 </div>
 </div><script>
   function initMap() {
     var directionsService = new google.maps.DirectionsService;
     var directionsRenderer = new google.maps.DirectionsRenderer;
     var map = new google.maps.Map(document.getElementById('map'), {
       zoom: 6,
       center: {lat: 41.85, lng: -87.65}
     });
     directionsRenderer.setMap(map);
     document.getElementById('submit').addEventListener('click', function() {
     calculateAndDisplayRoute(directionsService, directionsRenderer);
     getSensorData();
```

```
});
}
function calculateAndDisplayRoute(directionsService, directionsRenderer) {
        var waypts = [];
        var checkboxArray = document.getElementById('waypoints');
        for (var i = 0; i < checkboxArray.length; i++) {
          if (checkboxArray.options[i].selected) {
            waypts.push({
              location: checkboxArray[i].value,
              stopover: true
            });
            }
        }
directionsService.route({
          origin: document.getElementById('start').value,
          destination: document.getElementById('end').value,
          waypoints: waypts,
          optimizeWaypoints: true,
          travelMode: 'DRIVING'
        }, function(response, status) {
          if (status === 'OK') {
            directionsRenderer.setDirections(response);
            var route = response.routes[0];
            var summaryPanel = document.getElementById('directions-panel');
            summaryPanel.innerHTML = '';
            // For each route, display summary information.
            for (var i = 0; i < route.legs.length; i++) {
              var routeSegment = i + 1;
              summaryPanel.innerHTML += '<b>Route Segment: ' + routeSegment
+'</b><br>';
              summaryPanel.innerHTML += route.legs[i].start_address + ' to ';
              summaryPanel.innerHTML += route.legs[i].end_address + '<br>';
              summaryPanel.innerHTML += route.legs[i].distance.text + '<br><br>';
            }} else {
            window.alert('Directions request failed due to ' + status);
          }
        });
}
function getSensorData(){
$.getJSON('https://things.ubidots.com/api/v1.6/devices/esp32/mq2?token=BBF F-
20RVKzz5Rps6jvxoHBIe8PaHagmyCC', function(data) {
 var text = `Pollution at MQ2: ${data.last_value.value}<br>
                 Pollution at MQ135: ${data.last_value.value}<br>
                 Pollution at MQ3: ${data.last_value.value}<br>
                 Time: ${data.last_value.timestamp}<br>
                 Sensor Name: ${data.name}`
          $(".mypanel").html(text);
       }); }
     </script>
     <script async defer
     src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBKnjywb_ch-
maR1HdRKvMhtkbRCAZvFtQ&callback=initMap">
     </script>
     <script> </script>
    </body>
</html>
```

A2 SCREENSHOTS

Figure A2.1 represents the hardware implementation of air pollution monitoring system with the combination of gas sensors. Figure A2.2 represents the Ubidots portal in which the sensed data will be stored and drawn color codes. Figure A3.3 determines the pollution level in metropolitan cities. Figure A2.4 represents the splash screen of Android application which navigates to Home Page within two seconds. Figure A2.5 represents the possible routes to travel from Hosur to Chennai. Figure A2.6 represents the route drawn from Hosur to Chennai via the intermediate Vellore. Figure A2.7 gives the pollution level of the intermediate waypoints. Figure A2.8 shows the notification message appeared to the user.

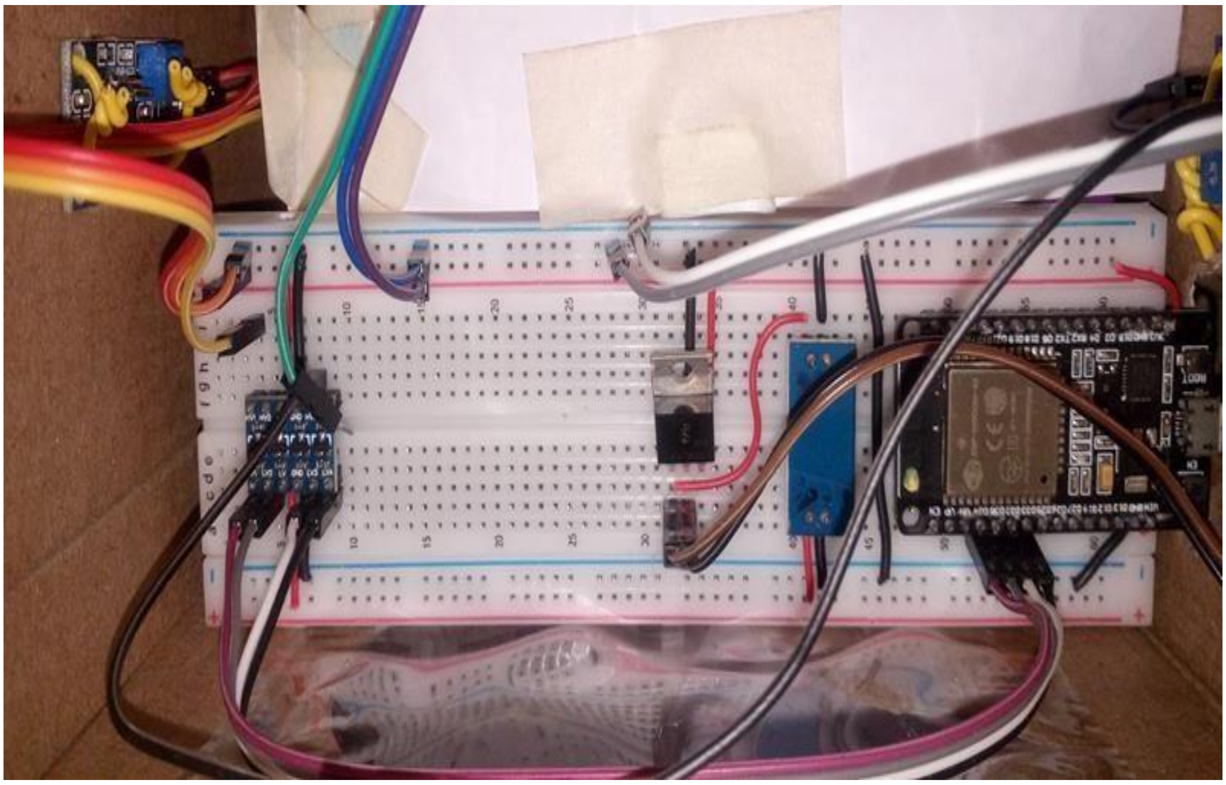


Figure A2.1 IoT Kit

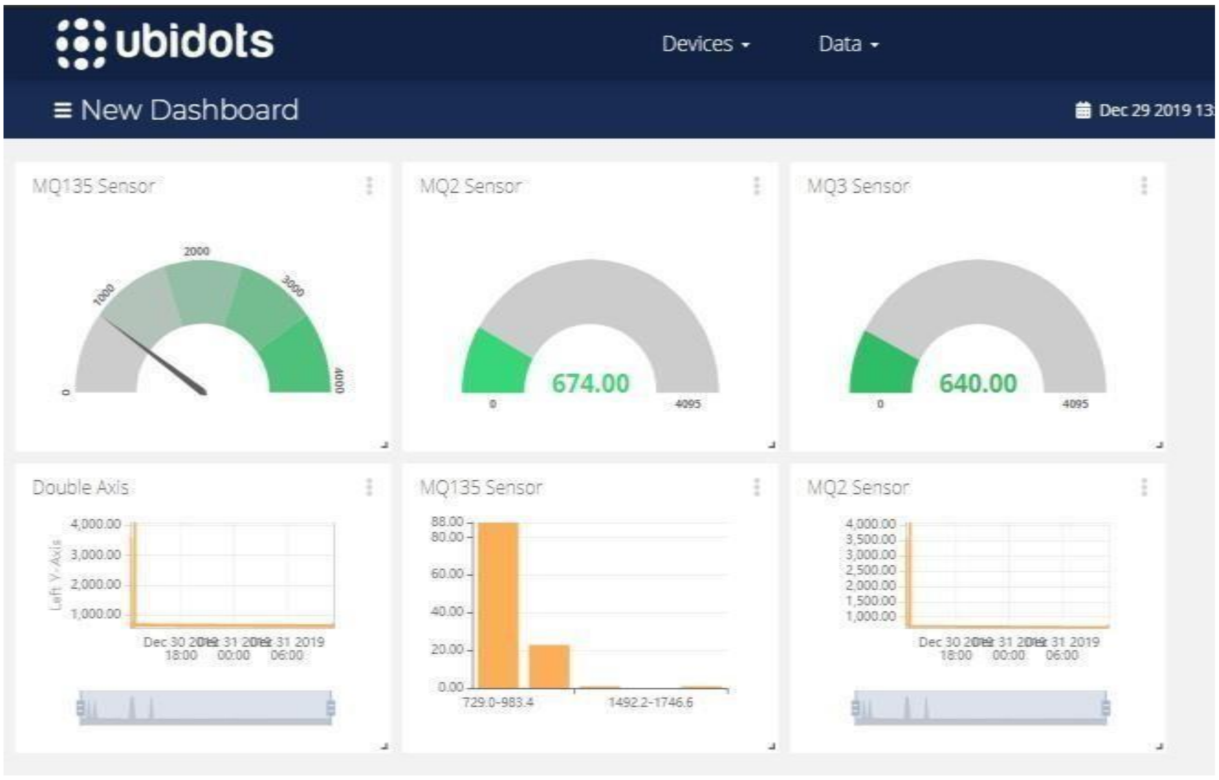


Figure A2.2 Ubidots Platform

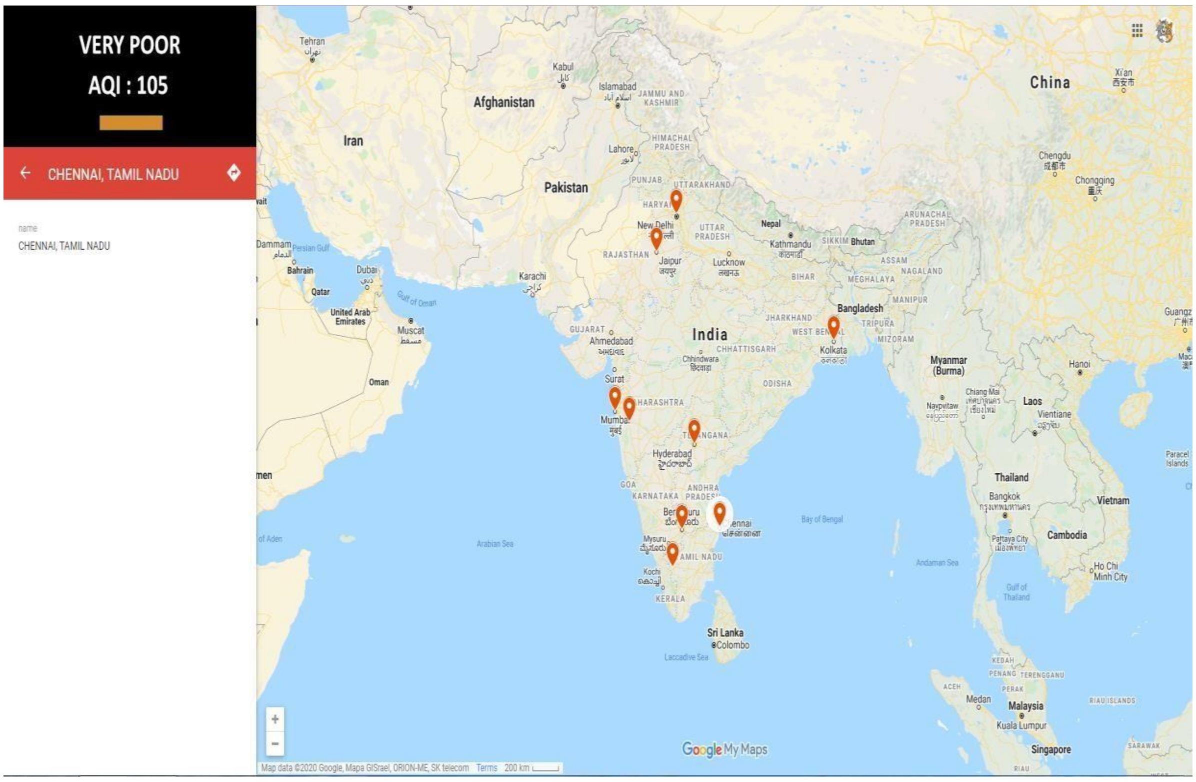


Figure A2.3 Pollution rate at Metro cities

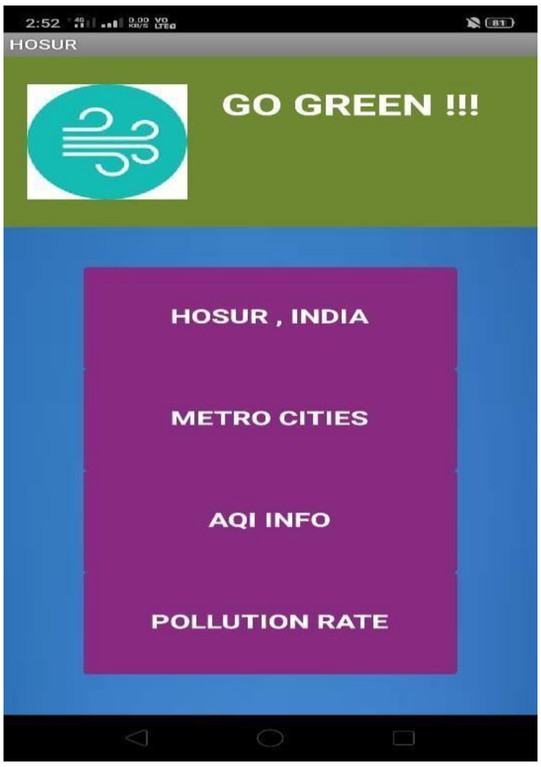


Figure A2.4 Home Page

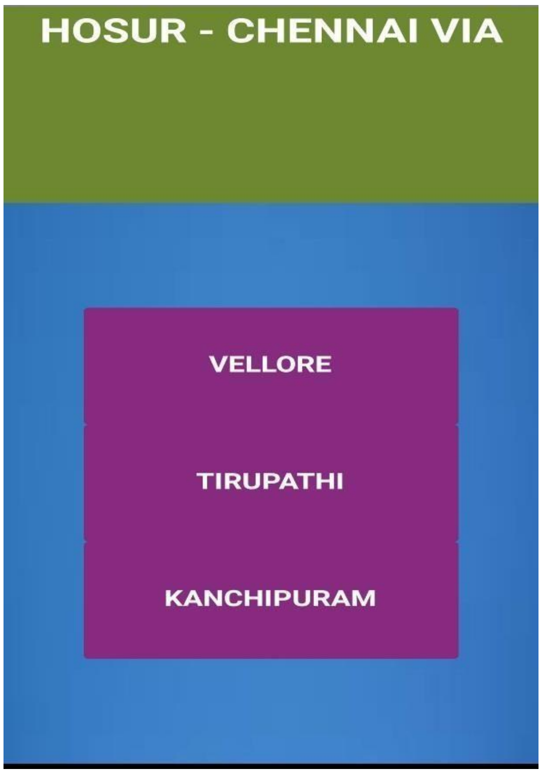


Figure A2.5 Possible Routes

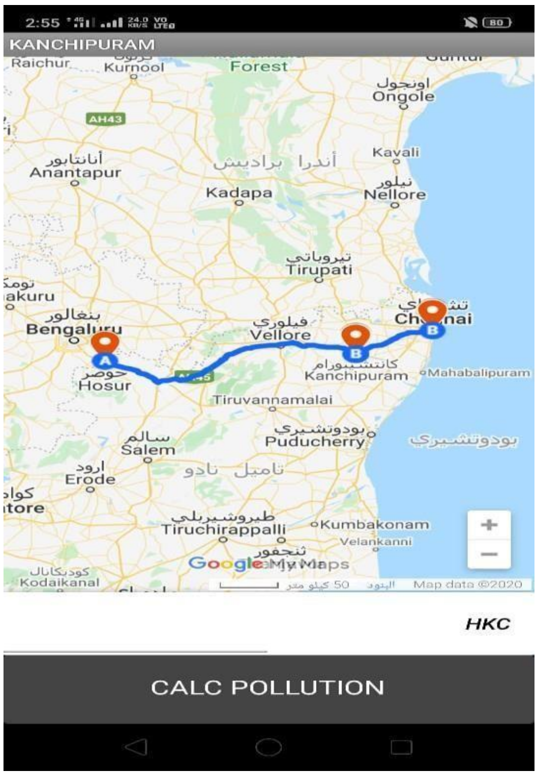
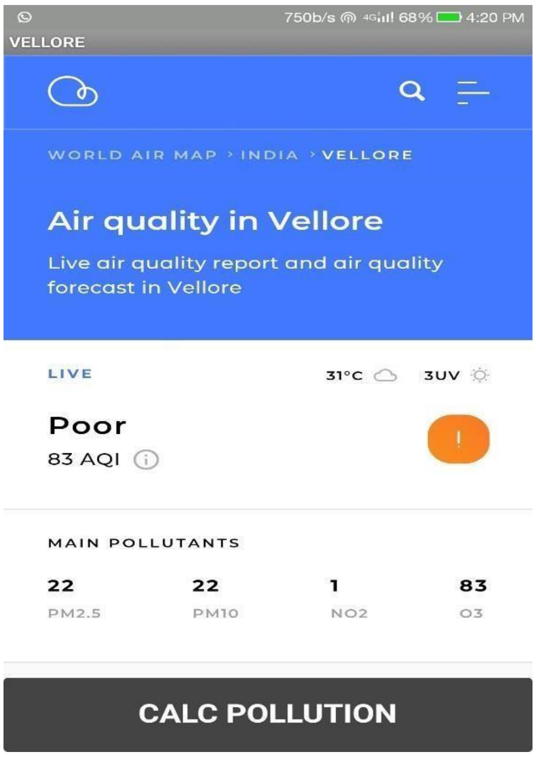


Figure A2.6 Route from Hosur – Chennai via Vellore


750b/s 4G 68% 4:20 PM

VELLORE

WORLD AIR MAP › INDIA › VELLORE

Air quality in Vellore

Live air quality report and air quality forecast in Vellore

LIVE 31°C 3UV

Poor

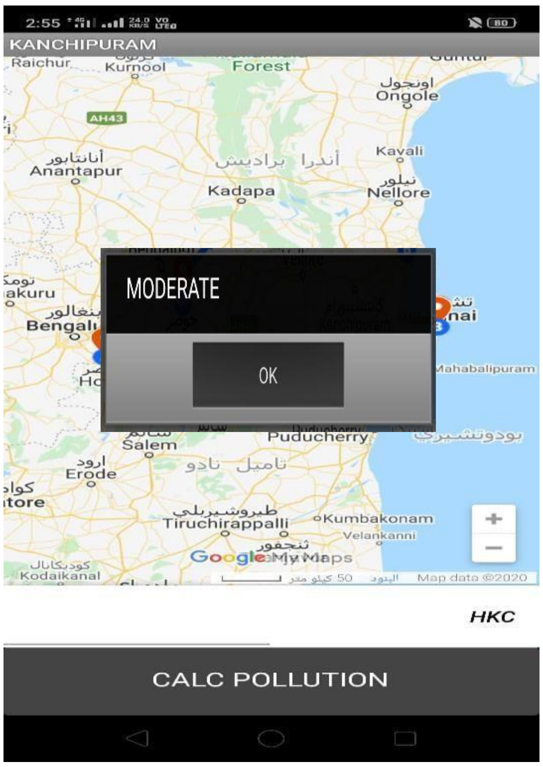
83 AQI

MAIN POLLUTANTS

22	22	1	83
PM2.5	PM10	NO2	O3

CALC POLLUTION

Figure A2.7 Air Quality in Vellore


2:55

KANCHIPURAM

MODERATE

OK

HKC

CALC POLLUTION

Figure A2.8 Notification Message

REFERENCES

[1] Alwar alshamsi, Younas Anwar,Maryam Aimulla,Mouza Aldohoori, Nasser Hamad, Mohammad Awad (2017) “Monitoring Pollution: Applying IoT to create a smart environment”, International Conference Proceedings ,vol 8,pp. 203-805.

[2] Benjamin. M. R and Alok A. J. B, Bhatt. N, (2013) “Automation Testing Software that Aid in Efficiency Increase of Regression Process,” *Recent Patents Comput. Sci.*, vol. 6, no. 2, pp. 107–114, 2013.

[3] De Nazelle. A, Seto. E, Donaire-Gonzalez. D, Mendez. M, Matamala. J, Nieuwenhuijsen.M. J, and Jerrett. M, (2013) “Improving estimates of air pollution exposure through ubiquitous sensing technologies,” *Environ. Pollut.*, vol. 176, pp. 92– 99.

[4] Lo Re. G, Peri. D, and Vassallo. S. D, (2014) “Urban air quality monitoring using vehicular sensor networks,” in *Advances onto the Internet of Things*, Springer, pp. 311–323.

[5] Predić. B, Yan. Z, Eberle. J, (2013) “Exposure sense: Integrating daily activities with air quality using mobile participatory sensing,” *2013 IEEE International Conference on* pp. 303–305.

[6] Peterová. R and Hybler. J, (2011) “Do-it-yourself environmental sensing,” *Procedia Comput. Sci.*, vol. 7, pp. 303–304.

[7] Zheng. Y, Chen. X, Jin. Q, Chen. Y, Qu.X, Sun. W, (2014) “A cloud-based knowledge discovery system for monitoring fine-grained air quality,” *Prep. Microsoft Tech Report, http//research. Microsoft. Com/apps/pubs/default.*