King Fahd University of Petroleum and Minerals

College of Computer Sciences and Engineering (CCSE)

COE 485

IoT-Based Indoor Environmental Monitoring System

Final Design Report

April 25, 2020

Student Details:

Student Names	Identification Number	Section Number
Shuaib Mohammed Khalid Nouh Aymen Somily	201572070 201458120 201218060	1

Supervisor	Dr. Muhammad E.S. Elrabba

ACKNOWLEDGEMENT

Acknowledgement is due to King Fahd University of Petroleum and Minerals for its support during this work.

We wish to place on record our sincere thanks and deep appreciation to our senior design project supervisor Dr. Muhammad ElRabba for his constant support during the course of this senior design project and for making it an important learning experience for all of us. His timely appraisals and feedback during the various project phases were invaluable and contributed to the successful completion of the project. Thanks are also due to the COE faculty members who taught us during our undergraduate career at KFUPM.

Last, but not the least, we wish to thank our dear parents for their encouragement and support.

ABSTRACT

Internet of Things (IoT) can facilitate the design of indoor environmental monitoring systems that can sense vital parameters generating data which can be then stored, processed and visualized on a cloud platform. From the cloud, the processed information can be accessed through various front end user interfaces such as web or mobile applications. This report presents the design and implementation details of an IoT-based indoor environmental monitoring system which employs low-cost devices such as Arduino Uno, Raspberry Pi, ESP8266 WiFi shield, and electronic sensors (DHT-11 & MQ-2) for the real-time monitoring of parameters such as temperature, humidity and gas content. Sensed environmental data is stored in the AWS MySQL database. The sensor node data stored in the MySQL database of the AWS RDS is made available to external client applications such as the mobile app and web browser using REST APIs. In the prototype design, the REST APIs have been implemented using nodejs. The same APIs would be called from web application or from a mobile app. The 'Web Application" consists of two parts: [i] server side and [ii] the client side. The server side is programmed using Nodejs and it provides the APIs for the client to access. The Nodejs component is deployed using AWS Beanstalk service. The client part is implemented using Angular. The angular front end component is deployed on AWS S3 bucket. The S3 bucket is made accessible by the AWS CloudFront service which is utilized to create a URL for the web application. The mobile app utilizing the same APIs facilitates online monitoring of indoor environmental parameters from any location. Further, the developed system prototype facilitates the centralized management of customers and their sensor nodes by the system administrator.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Over the last few decades, there have been increasing concerns about the effect of indoor environments on the health and productivity of the building occupants. This concern was initially triggered by occupants' complaints of unspecific symptoms, such as burning eyes, headache, or fatigue [1]. A good indoor environment is associated with high indoor air quality and thermal comfort. If the indoor environment is thermally uncomfortable, and contaminated by indoor pollutants such as dust particles and harmful gases etc., it can result in a health condition known as "Sick Building Syndrome (SBS)". The term SBS refers to a range of nonspecific illnesses/symptoms that that are experienced by an occupant while inside a particular building. These abnormal SBS symptoms such as eye irritation, runny nose, breathing difficulties, skin rashes, and headaches usually disappear within hours, after the occupant leaves the affected building [1]. Employees whose health is affected by polluted air and thermally uncomfortable office working conditions are inefficient and it has been determined that office productivity reduces by roughly 1 percent for every 1^{o}C deviation from the optimal room temperature [2]. For example, In France alone, economic losses from sub-optimal indoor environments amounted $20 billion in 2004 [2].

Most building designers focus on immediate functional requirements (space requirements, temperature ranges, and attractive design) and meeting building-code requirements at minimum cost. The impact of design considerations on indoor environment quality, occupant health, and productivity is generally not emphasized. In office work, salaries are by far the greatest employer cost, so even a small increase in productivity (due to an improved work environment) can provide a favorable return on investment. Improvements in office productivity and reduced healthcare costs can be achieved through effective monitoring/control of the indoor environmental parameters such as temperature, humidity etc., and through the utilization of efficient Heating, Ventilation and Cooling (HVAC) systems. The associated HVAC energy consumption in buildings account for a significant proportion of total energy consumed worldwide. Since humans spend about 90% of their time in buildings, it is essential that the indoor environment quality is improved and energy consumption is minimized [1].

Effective mitigation of the problems described above can occur only if systems that monitor the indoor environments in real-time are deployed. However, most monitoring systems that were deployed in the past were not scalable, limited in scope (fewer sensing nodes) and localized. They did not facilitate Internet accessibility to the monitored indoor environmental data nor provide remote alerts. Further, these localized systems did not facilitate detailed data analytics. Advances in IoT technologies can now overcome the aforementioned limitations and facilitate the development of efficient indoor environmental monitoring systems.

In the proposed senior design project, a low-cost IoT-based indoor environmental monitoring system prototype was developed for the real-time monitoring of indoor temperature, humidity and air quality. The

deployment of this system can lead to potential societal benefits such as improved office productivity, lower healthcare costs, and optimal HVAC energy consumption costs through efficient real-time monitoring of the indoor building environments. Further, the proposed prototype facilitates the centralized management of multiple sensing nodes of multiple customers through cloud-based interfaces. The recent proliferation of IoT devices and insufficient authentication practices can make IoT systems such as the proposed prototype vulnerable to hacking and thereby endanger the privacy, security and integrity of customer data. This potential pitfall of IoT implementations can however be overcome by adherence to best security practices and the use of stringent authentication protocols.

2. REQUIREMENTS AND SPECIFICATIONS

2.1 Requirements

The requirements of the proposed IoT indoor environmental monitoring system from the customer's point of view are:

- The system shall monitor indoor environmental parameters such temperature, humidity and gas content etc.
- The system shall allow customers to choose several types of sensors as add-ons to a basic sensing platform.
- The system shall be easy to install and configure.
- The system shall not require any special installation fixtures for operation.
- The system shall not be localized and allow customers to view the sensed indoor environmental parameters from anywhere.
- The system shall provide an alert alarm when a room becomes thermally uncomfortable or abnormal levels of smoke and harmful gases such as LPG and carbon monoxide (CO) are detected.
- The system shall be affordable.

2.2 Specifications

The engineering specifications of the proposed IoT indoor environmental monitoring system are listed in Table 1. The proposed system requires data in order to characterize a particular indoor environment. This involves measurements from multiple sensors and the use of IoT devices for satisfying the user requirements. The specifications listed in Table 1 ensure that the proposed system possesses sufficient functionalities that would satisfy the system requirements.

Table 1: Requirements and Specifications

Requirement	Specification
The system shall monitor indoor environmental parameters such temperature, humidity and gases etc.	The system shall utilize the dual DHT11 sensor for monitoring indoor temperature/humidity and the MQ-2 gas sensor for monitoring indoor gas concentrations.
The system shall allow customers to choose several types of sensors as add-ons to a basic sensing platform.	The system shall facilitate the further addition of several temperature, humidity and gas sensors.
The system shall be easy to install and configure.	The system shall facilitate a plug-and-play type of operation wherein customers do not need to configure any sensing node.
The system shall not be localized and allow the results to be viewed in real-time from anywhere.	The system shall implement an IoT-based architecture to facilitate real-time visualization of sensed data through web and mobile dashboards from any location.
The system shall provide automatic alerts when indoor parameters exceed desired limits.	The system shall warn users through email and mobile alerts, when a room becomes thermally uncomfortable or abnormal levels of harmful gases are detected.
The system shall not require any special installation fixtures for operation.	The system shall utilize the existing building infrastructure such as domestic power supply outlets, WiFi/Internet connectivity etc., for its operation.
The system shall be affordable.	The system shall utilize low-cost commercially available sensors to reduce costs.

3. PROJECT APPROACH

3.1 Possible Approaches

There are two possible approaches available for implementing the project requirements. These are: [i] Localized System Approach and, [ii] IoT-based System Approach.

3.1.1 Localized System Approach

As the name suggests, local systems facilitate only local monitoring of indoor environmental parameters such as temperature, humidity etc. The sensed data is processed locally. With local systems, remote online access to the monitored environmental parameters is not possible. Further, the limited storage and processing capabilities of the local system do not support detailed data analytics. Another limitation of local monitoring systems is that they support limited number of sensors and were not scalable.

3.1.2 IoT-Based Approach

A suitable alternative approach that can be utilized for the implementation of indoor environmental monitoring systems involves the use of IoT technology. IoT can facilitate the design of indoor environmental monitoring systems that can sense vital parameters generating data which can be then stored, processed and visualized on a cloud platform. From the cloud, the processed information can be accessed through various front end user interfaces such as web or mobile dashboards.

3.2 Selected Approach: IoT-based Approach

The primary requirement at present, of indoor environmental monitoring systems is to facilitate real-time monitoring of indoor parameters and issue timely alerts to the end users. To fulfill this and the other system requirements, an IoT-based approach will be adopted for the design of the system prototype. Considering key factors such as battery life expectancy and reliable single hop communication abilities, IoT architectures are most suitable for real-time monitoring of indoor environments. In addition to their lower latencies (negligible time delays) and reduced energy consumption, IoT-based monitoring systems are also easier to operate and maintain. Further, they also facilitate the monitoring of a large number of environmental parameters without affecting system performance [5-7]. The salient features of the adopted IoT approach are presented below:

(i). Getting Input Data

Indoor environmental data can be obtained by using mechanical sensing equipment or electronic sensors. Mechanical sensors such as Thermohygrometers are typically employed for direct control/visualization purposes and tend to be bulky and costly. Electronic sensors however are robust and simple to implement. Electronic sensors have the added capability to convert a detected parameter into a digital value that can be used for monitoring and control.

Early indoor environment monitoring models involved the use of wireless sensor networks [WSN] in which the preferred communication protocol of choice was Zig Bee and web servers were utilized as the data access platforms [3,4]. The main limitation of WSN-based systems was the considerably high-energy consumption by the sensor networks while transmitting data in multiple hops to the microcontroller units. The time taken by sensors to send a signal to the monitoring unit was also observed to be considerably high. The other disadvantages of WSN-based implementations include short communication range, low network stability with high maintenance costs [7].

Indoor environmental data in the proposed prototype will be obtained using low-cost electronic sensors which are connected directly to the microcontroller. The sensor nodes comprising the electronic sensors, microcontroller and the WiFi shield will be the key element in the proposed IoT-based monitoring system.

(ii). Sensing Pattern

The sensing pattern can be any of the following ways:

- *Event-Driven:* The transmission of sensor information is driven by an event.
- *Periodic:* The transmission of sensor information occurs at periodic intervals.

In the proposed system, sensing units shall transmit data at periodic intervals to facilitate real-time indoor environmental parameter updates.

(iii). Transmitting Data

The data sensed by the sensors can be transmitted to the cloud server in the following ways:

- Direct transmission from sensors to the cloud.
- Data transmission from sensors to the cloud via the IoT gateway.

Data transmission via IoT gateway is selected in the proposed system because an IoT gateway can aggregate sensor data, translate between sensor protocols, and process sensor data before sending it to the cloud. As connected devices, protocols and requirements increase, having sensors connect individually to the cloud is not often possible. This is because some sensors and controllers use very low energy and do not support energy-intensive protocols like Wi-Fi or Bluetooth, and, therefore, cannot connect directly to the cloud.

(iv). Processing and Analyzing Data

Processing of the sensed indoor environmental data can be done in the following ways:

- *Localized Approach:* The sensed indoor environmental data is processed locally in the microcontroller.
- *Centralized Approach:* The indoor environmental data sensed by the temperature, humidity and gas sensors is processed and transmitted to the cloud for further computation and analysis (data visualization).

The centralized approach is selected due to the higher computation capability and high storage capacity of the host cloud server than the local system components.

(v). Network Access Technology

The available network access technologies include Wireless Local Area Network (WLAN), Wireless Personal Area Network (WPAN) and Wireless Home Area Network (WHAN). The WLAN approach which includes IEE 802.11 standards such as WiFi will be adopted in the prototype design.

(vi). Cloud Computing Service Model

The available cloud computing service models include the following:

- Infrastructure-as-a-Service (IaaS) which provides hardware and software upon which the customer can use a computing environment. It also provides storage and networking services, for example, Amazon Elastic Compute Cloud (EC2) and Microsoft Azure.

- Platform-as-a-Service (PaaS) which provides a development environment for developers to build cloud-based services and applications. It also provides a database, web server, and execution environment, etc. Ex. Heroku., ThingsBoard etc.

- Software-as-a-Service (SaaS) which SaaS provides application software to the user through the Internet, for example, MS Office 360, Google doc, ThingsSpeak.

- (Data-Storage-as-a-Service (dSaaS) which provides only storage space and network bandwidth services, for example, Dropbox, Google Drive, etc.

In the proposed system, Amazon Web Services (AWS) will be utilized for cloud computing and the centralized management of customers. AWS is versatile and provides application developers a set of dependable tools and a reliable infrastructure that they could build products on top of.

(vii). Viewing the Result

Monitoring of the indoor environmental conditions by a user can be implemented in the following ways:

- Displaying the results on a web page.
- Viewing the result on mobile apps.

In the proposed system, the measured indoor parameters will be displayed on both web pages and mobile application. This approach allows the user to gain online access to the environmental data from any location.

(viii). Warning the User

The user can be warned if the indoor parameters exceed the desired limits through the following ways:

- Audio Buzzer
- Email Alerts
- App Notification

In the proposed system, email alerts and app notifications are selected to warn the user if the indoor environmental parameters fluctuate and are outside the desired set points. The user can receive the alerts from any location.

(ix). App Platform Availability

The user can monitor the environment inside a building room from a mobile application through the following approaches:

- Cross-Platform

- Single Platform

The application selected for the proposed system will be available for Android platform.

(x). Application Protocol

The available application protocols include HTTP, XMPP, MQTT, and AMQP. The proposed system shall utilize the MQTT and HTTP protocols in its design.

3.3 Work Plan

The tasks performed during the senior design project and their timeline are listed in Table 2 below:

Table 2: Work Plan

Week	Activity
Week 1	Team formation and project selection.
Week 2	Initial project plan and review of possible design approaches for fulfilling project requirements.
Week 3	Conceptual system design through appropriate use case diagram, activity diagrams and deployment diagram.
Week 4	Procuring system components and review of their technical specifications.
Week 5 & 6	Configuring the sensor node (Arduino Uno microcontroller, DHT-11 temperature-humidity sensor, MQ-2 gas sensor and the ESP8266-01 WiFi Shield and 'C' coding for the sensor node implementation.
Week 7	Configuring the Raspberry Pi as the IoT gateway and mid-term design report preparation
Week 8	Python coding for the Raspberry Pi and implementation.
Week 9	AWS account creation and IoT core configuration.
Week 10	Enabling MySQL on RDS in AWS Free Tier and AWS MySQL database design.
Week 11	Coding Lambda functions for populating the MySQL database with IoT Core sensor node data
Week 12, 13 & 14	Prototype testing, Web Dashboard development and Android Mobile App development.
Week 14	System Integration and Testing
Week 15 & 16	Preparation of Final Report

4. CONCEPTUAL SYSTEM DESIGN

The conceptual design of the IoT-based indoor environmental monitoring system is presented in the following sections through the use case and activity diagrams.

4.1 Use-Case Diagram

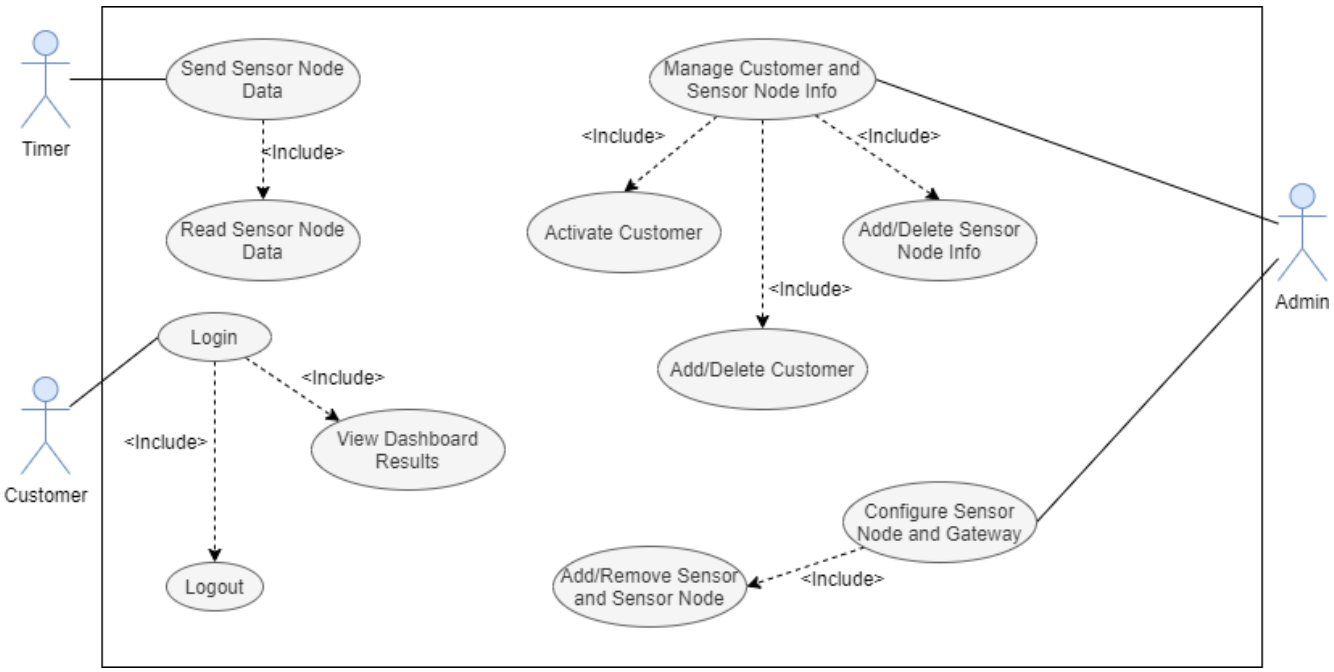


Figure 1: Use Case Diagram of the IoT-Based Indoor Environmental Monitoring System

The use case diagram for the proposed IoT-based indoor environmental monitoring system is illustrated in Fig. 1. The actors in this system are the System Administrator, Customer (End-User) and the Timer. The interactions of these actors facilitate the real-time monitoring of indoor environmental data by a customer in his building premises. The system will facilitate storage, processing and management of the sensed environmental data on the AWS cloud platform. Further, it will facilitate the registration, identification and centralized management of customers.

4.2 Activity Diagrams

4.2.1 Manage Customer and Sensor Node Information

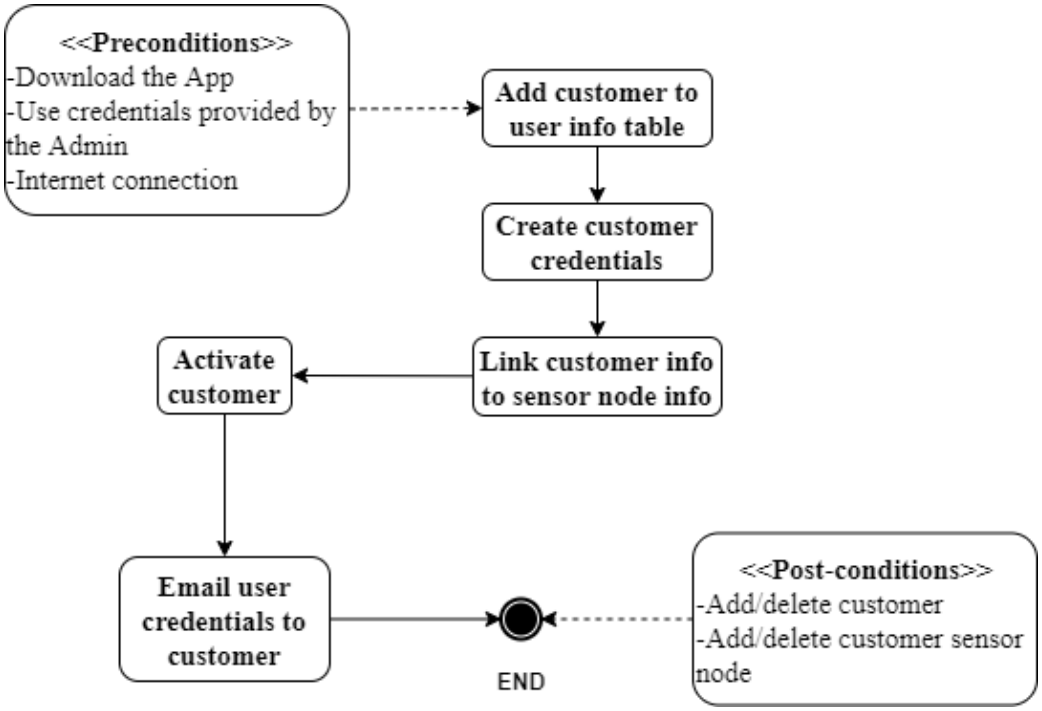


Figure 2: Manage Customer and Sensor Node Information Activity Diagram

Primary Actor(s)	➢ Admin ➢ Customer
Description	➢ Admin activates customer by registering him/her on his behalf. ➢ Admin provides user credentials to the customer to enable him/her to login to the mobile app or website.
Preconditions	The customer has purchased the sensing hardware and signed the indoor environmental monitoring service agreement.
Flow of Events	As depicted in the activity diagram.
Post-Conditions	Admin manages the customer account by adding or deleting sensor nodes based on customer's sensing requirements.
Assumptions	None

The proposed system facilitates the system administrator to set up customer profiles, link sensor nodes with the customer accounts and activate customers on the cloud platform. The activity diagram for customer configuration is shown in Fig. 2.

4.2.2 Configure Sensor Node and Gateway

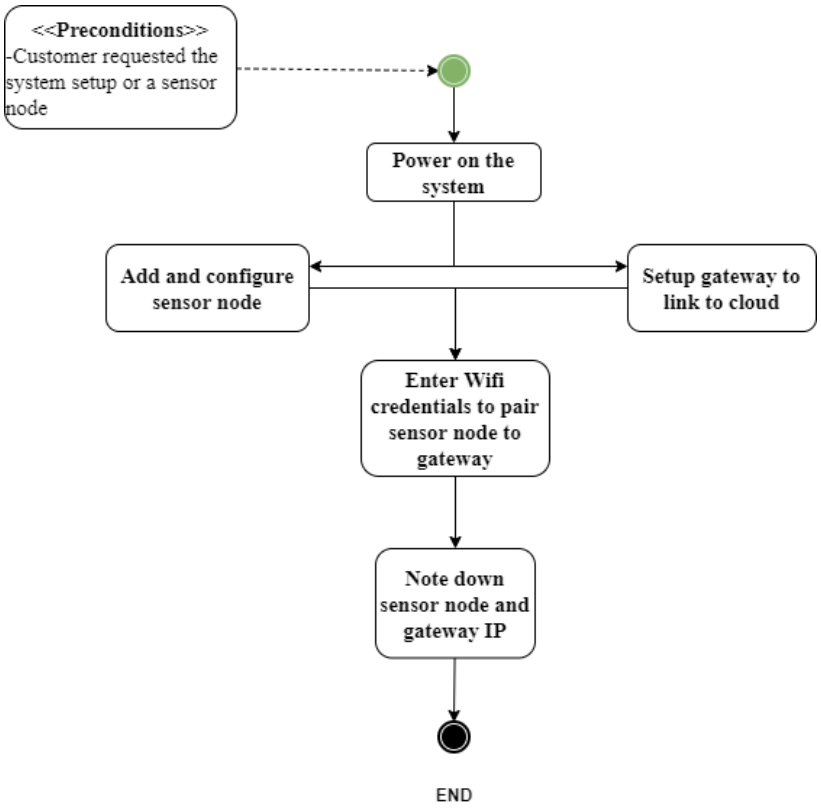


Figure 3: Configure Sensor Node and Gateway Activity Diagram

Primary Actor(s)	• Admin • Customer
Description	➢ Admin sets up the indoor monitoring system for the customer. ➢ Admin adds the sensor nodes as per the customer's sensing requirements. ➢ Customer inputs his WiFi credentials to establish connections of the sensor node and gateway to the cloud**.**
Preconditions	Customer requested the system setup or the addition of an extra sensor node.
Flow of Events	As depicted in the activity diagram.
Post-Conditions	None
Assumptions	Customer has WiFi connections.

The proposed system facilitates the configuration of sensor nodes and gateway of a particular customer by the system administrator. The associated activity diagram is depicted in Fig. 3.

4.2.3 Send Sensor Node Data

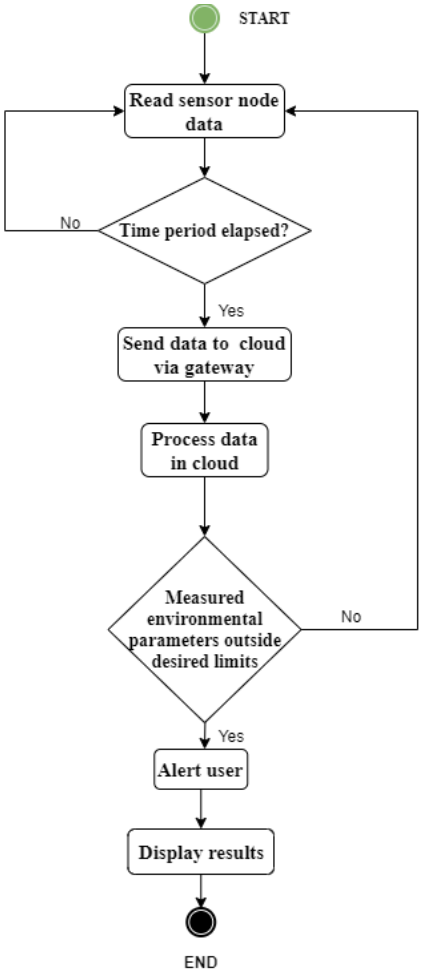


Figure 4: Send Sensor Node Data Activity Diagram

Primary Actor(s)	Time
Description	Sensor node data is sent to the gateway at periodic time intervals.
Preconditions	None
Flow of Events	View dashboard results on the web or mobile app after entering the credentials provided by the admin.
Post-Conditions	None
Assumptions	None

The indoor environmental data sensed by the sensor nodes shall be transmitted at periodic intervals of time through the IoT gateway to facilitate its processing in the cloud platform. If the sensed indoor parameters are outside the desired limits, the system alerts the user. The associated activity diagram is shown in Fig. 4.

4.2.4 Login

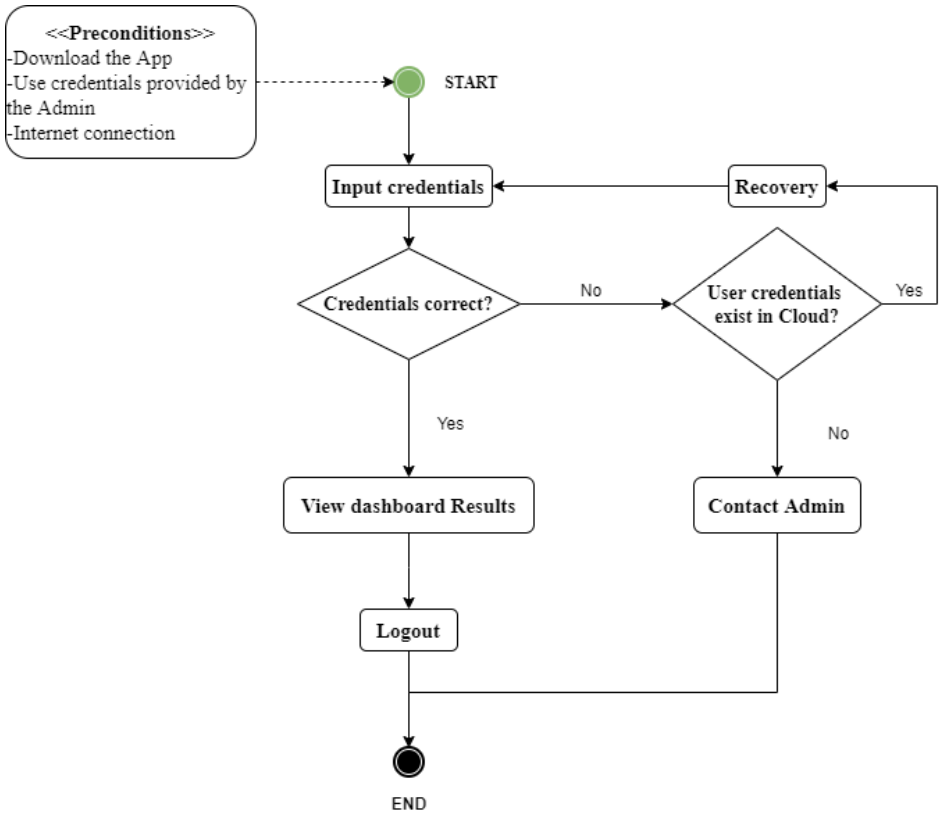


Figure 5: Login Activity Diagram

Primary Actor(s)	Customer
Description	User enters his credentials provided by the admin to login/logout to the mobile app and website to view dashboard results.
Preconditions	• Download the mobile app. • End-user credentials provided by the admin after customer registration. • Availability of Internet connection.
Flow of Events	View dashboard results on the web or mobile app after entering the credentials provided by the admin.
Post-Conditions	None
Assumptions	None

The proposed system enables a customer to view his monitored indoor parameters on web and mobile dashboards from any location by logging in. The associated activity diagram is shown in Fig. 5.

4.3 System Specifications

The specifications for the IoT-based Indoor Environmental monitoring system are listed in Table 3 below:

Table 3: System Specifications

Parameter		**Specifications**
Measurement Range		0 to 50°C (temperature)
		10% to 90% RH (humidity)
		200-10000 ppm for gas concentration
Accuracy[1]		±2°C (temperature measurement)
		±5% RH (humidity measurement)
		±10 ppm (Gas content)
Resolution[1]		1°C (temperature measurement)
		1% (humidity measurement)
Alert Limits	Temperature[2]	20°C - 25.5°C
	Humidity[2]	30% - 60%
	Gases[3]	>1000 ppm (smoke, LPG) >200 ppm (CO)
Power Supply		5V power adaptors
Bandwidth		1 Mbps
Monitoring Interfaces		Web and mobile dashboards
AWS Cloud Platform		• AWS t2 micro instance on Windows • S3 instance with 5 GB storage • IoT Core • RDS with MySQL database with 20GB storage • AWS Lambda • AWS EC2-Beanstalk • AWS S3 • AWS CloudFront

1: DHT-11 & MQ-2 sensor factsheets; 2: ASHRAE standard 62.1-2010 recommendations; 3: OSHA recommendations

The proposed system facilitates the monitoring of indoor temperature, humidity and pollutant gases such as CO, LPG and smoke. Indoor temperatures ranging from 0 to 50°C and indoor humidity values ranging from 10% to 90% RH can be monitored with measurement accuracies of ±2°C and ±5% RH respectively. These specified ranges and accuracies are dictated by the measurement specifications of the DHT-11 temperature-humidity sensor that will be utilized in the system design [12]. The system can detect gases such as CO, LPG and smoke if their concentrations lie in the range 200-10000 ppm. This gas detection range is based on the technical specifications of the prototype's MQ-2 gas sensor [13].

Because of the very low payload and light weight MQTT protocol, the bandwidth requirements are very minimal for the proposed system. However, for the dashboard visualization on the web application and the mobile app, a minimum of 1 MBPS bandwidth is preferred. The prototype's hardware components such as the Arduino Uno microcontroller and the Raspberry Pi 3B gateway require 5V to operate and their low power requirements can be met with two power supply adaptors. The proposed Indoor Environmental System utilizes the AWS cloud platform with IoT core. For the prototype requirement, the AWS free tier is utilized with specifications listed in Table 2. These specifications facilitate the cloud processing/storage of sensor node data and centralized management of end-users.

Justifications for the Alert Limits:

The end-users are alerted by the system if the monitored indoor parameters exceed their upper limit or fall below the lower limits listed in Table 2. Any fluctuations in the monitored indoor parameters indicate possible HVAC system malfunction or ingress of harmful gases into the indoor air.

The specified temperature and humidity alert limits for the proposed system are based on *American Society of Heating, Refrigerating and Air-Conditioning Engineers* (ASHRAE) 62.1-2010 standard recommendations for ensuring a healthy thermally comfortable office work environment and minimizing building HVAC energy consumption costs [9]. Studies have revealed that occupants are most comfortable when indoor temperatures lie in the range 20°C - 25.5°C and indoor relative humidity is maintained in the 30%-60% range [9]. In case indoor humidity values fall below 30%, the building occupants can become susceptible to health problems such as nasal allergies, throat irritation, eye/mucous membranes drying and, dry itchy skin. In case of excessive humidity levels (> 70% RH), occupants will become prone to sweating and feel uncomfortable. Excessive humidity can also contribute to mold spore formation and microbial growth within the building environment (AC ducts, room ceilings etc.), which will further deteriorate the indoor environment and cause respiratory problems [9].

Maintaining the indoor temperature/humidity levels outside the desired ranges will increase the building's cooling/heating loads and result in high HVAC energy consumption costs. Wide fluctuations in the indoor temperature and humidity values indicate a probable malfunction in the HVAC equipment or its control systems and the prototype's alert limits can notify the HVAC engineer for deploying immediate remedial actions

Inadequate ventilation or fresh-air exchange may cause smoke and gases such as CO, and LPG to build-up or accumulate within buildings. The safe indoor gas content limits are specified by *Occupational Safety and Health Administration* (OSHA). In case, indoor gas concentrations exceed the tabulated OSHA limits, the end-users are notified immediately through email and mobile alerts [10].

4.4 System Architecture

4.4.1 Block Diagrams of Proposed Indoor Environmental Monitoring System

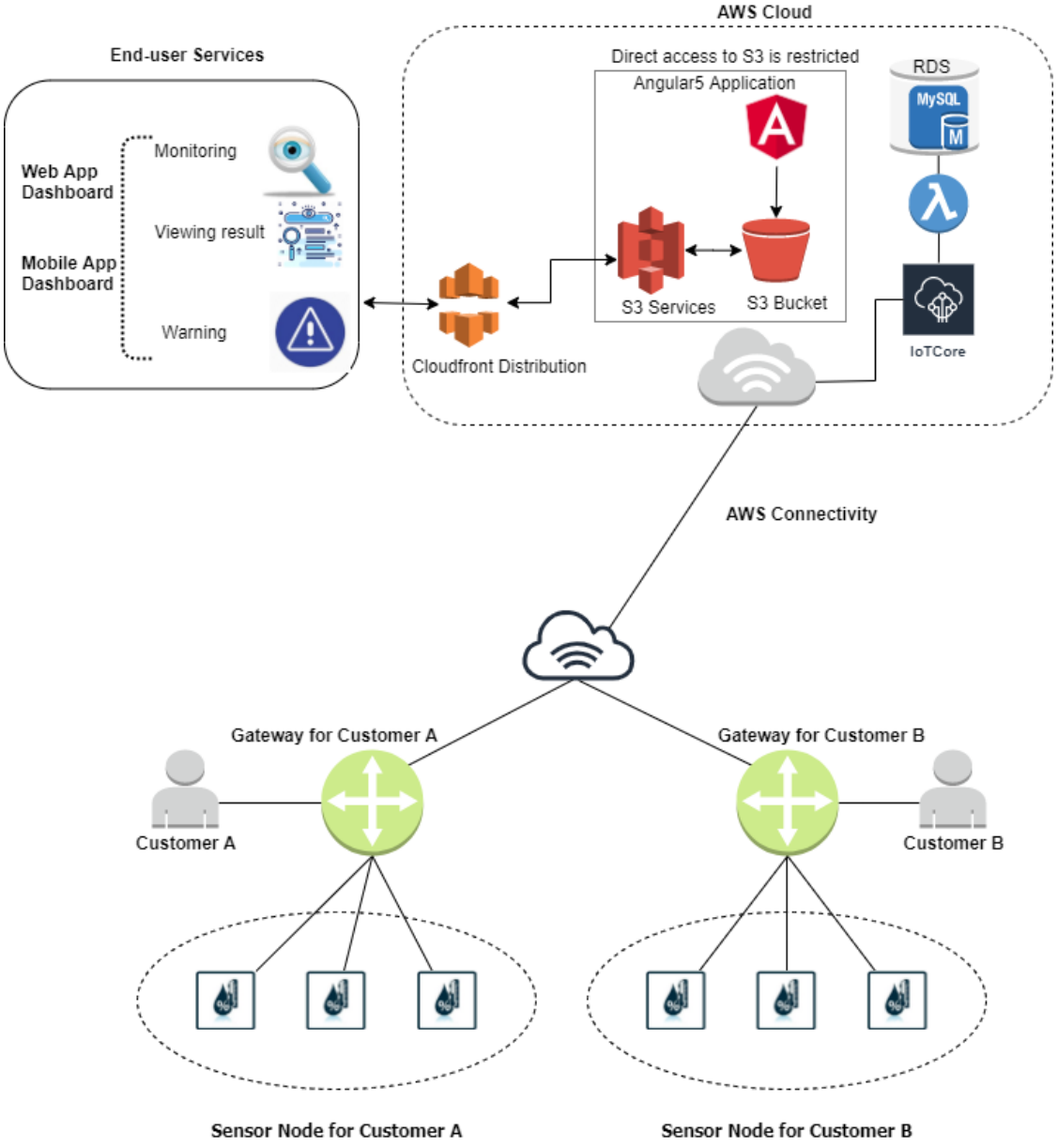


Figure 6: Overall System Architecture of Proposed Indoor Environmental Monitoring System

The overall IoT–based architecture of the proposed indoor environmental monitoring system is shown in Fig. 6. As shown, the proposed system can support multiple sensor nodes and facilitate real-time indoor monitoring for multiple customers. The node-level architecture diagram comprising the sensor node , Raspberry gateway and the AWS cloud platform utilized for the processing and storage of sensor data is depicted in Fig. 7. The node-level architecture depicted in Fig. 7 includes a sensor node, a Raspberry Pi 3B IoT gateway, and the AWS cloud platform.

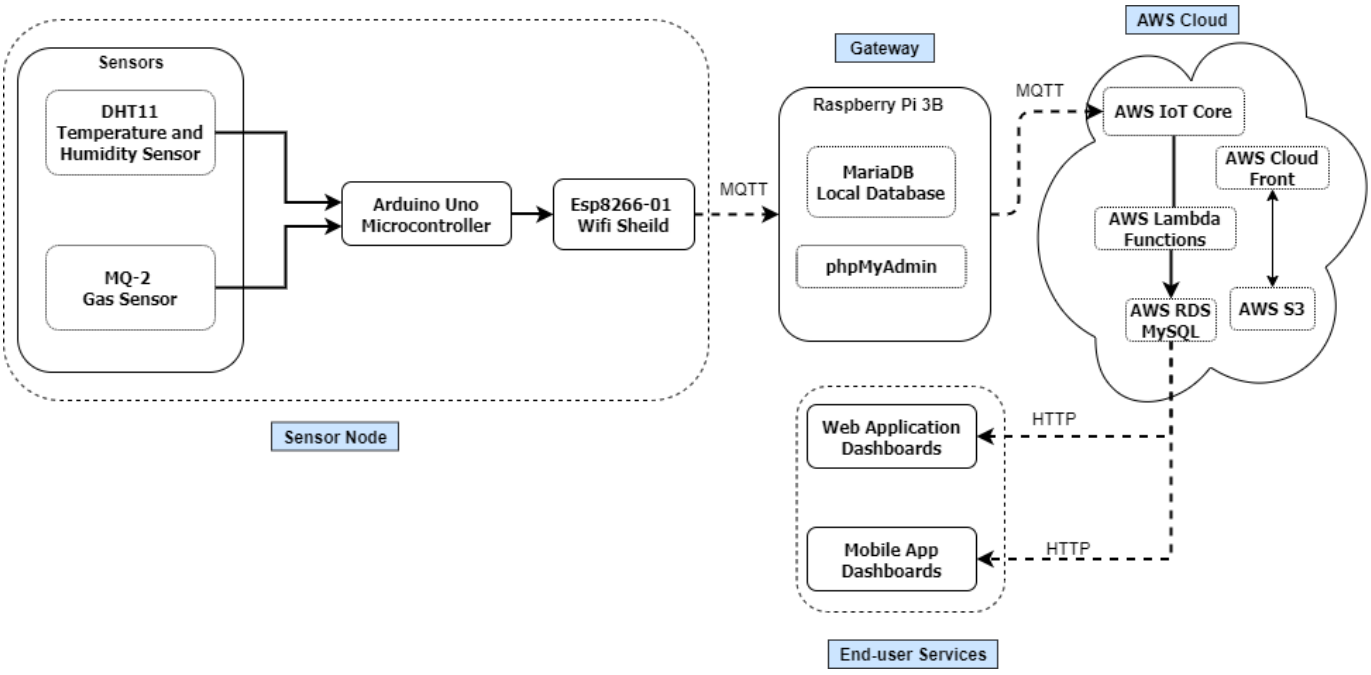


Figure 7: Node-level System Architecture

The sensor node comprising the DHT11 temperature-humidity and MQ-2 gas sensors, Arduino Uno microcontroller and the ESP8266-01 WiFi module is responsible for environmental data acquisition and transmission to the Raspberry Pi 3B gateway using the MQTT broker. The Raspberry Pi 3 gateway receives sensor data from the sensor node and stores it in a local MariaDB database before finally transmitting it to the AWS IoT core via the MQTT protocol. The local database in the Raspberry Pi 3 gateway ensures that no sensor data is lost when there is no Internet connection. In addition to the sensor data, information related to sensor node and gateway IDs is also transmitted to the AWS IoT core to facilitate proper customer-mapping and storage in the MySQL cloud database.

The data that is transferred to the IoT Core "Topic" is in turn stored in a MySQL database on AWS RDS instance. This is done by creating an action rule in IoT core that publishes data to a Lambda function. The Lambda function inserts the data into the MySQL database. A nodejs-Angular application is used to create the APIs that send dashboard data and perform configuration settings required for installation at each customer site. This application is deployed on a AWS S3 bucket using AWS EC2-Beanstalk service and accessed through URL configured in AWS CloudFront. In the proposed system, the web application and mobile application dashboards are designed to ensure ready online accessibility of the monitored indoor environmental data by the customer from any remote location. Further, if the sensed environmental parameters are abnormal, then the end-user is notified immediately through email and mobile alerts. Knowledge of the indoor environmental conditions in real-time can facilitate the rapid deployment of appropriate remedial actions for ensuring a healthy and thermally comfortable work environment for the building occupants. In addition to improved office

productivity, the proposed system also lowers HVAC energy consumption costs through optimal operation of the building HVAC systems. The proposed system architecture is scalable and its functionalities facilitate the addition of new customers and the centralized management of their sensor nodes by the system administrator. The inbuilt features and functionalities of the AWS cloud platform facilitates the management of multiple customers and their sensor nodes.

4.4.2 Deployment Diagram

The deployment diagram utilized to support the use cases and business model in the proposed IoT-based indoor environmental monitoring system is depicted below in Fig. 8.

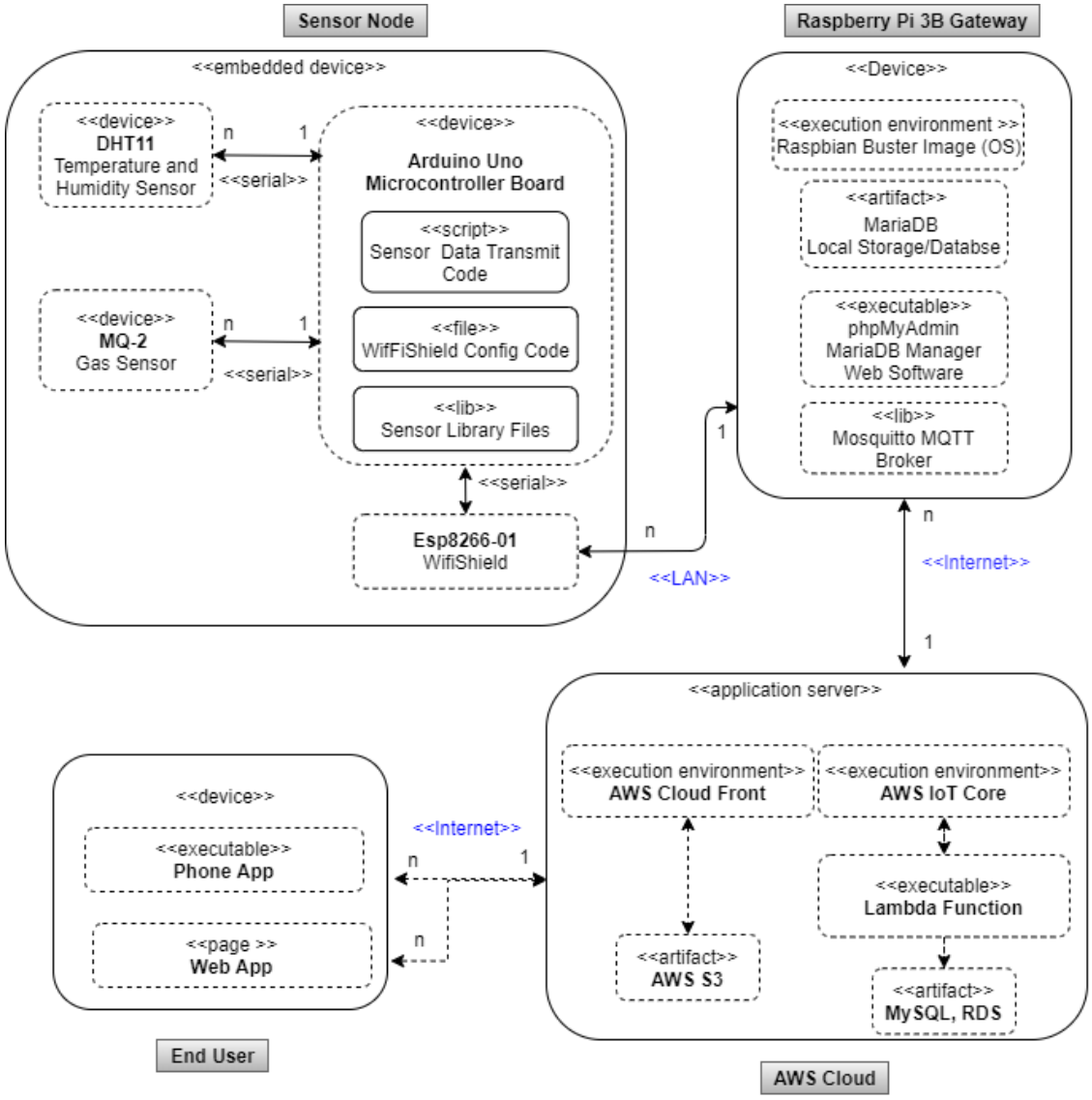


Figure 8: Deployment Diagram

4.4.3 System Components and their Specifications

The components utilized in the design of the IoT-based Indoor environmental monitoring system are described in the following sections:

A. Sensor Node

[i]. Microcontroller: Arduino Uno

The central hardware component of the sensor node is the Arduino Uno microcontroller. The Arduino Uno is based on the ATmega328P microcontroller and is capable of interfacing different peripherals, sensor and wireless communication devices through various analog and digital input/output pins. Arduino Uno can also be programmed easily with the Arduino IDE via a type B USB cable. The limitations associated with Arduino boards are low memory, single thread operation and inability to run an operation system (OS). The technical specifications [11] of the Arduino Uno are listed in Table 4 and the pin diagram is listed in Appendix-1.

Table 4: Arduino Uno Specifications

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz

[ii]. Temperature and Humidity Sensor: DHT11

The DHT11 composite sensor was selected for the simultaneous monitoring of indoor temperature and humidity. The DHT11 comprises a resistive-type humidity measurement component and a Negative Temperature Coefficient (NTC) component for measuring temperature. It is inexpensive, compact, reliable, and provides digital signal output. The technical specifications of DHT11 sensor are listed in Table 5. [12]

Table 5: DHT11 Specifications

Temperature measurement range	0 to 50°C
Humidity measurement range	10% to 90%
Resolution	Temperature and Humidity both are 16-bit
Accuracy	±1°C (Temperature) ±1% (Humidity)
Signal transmission range	20 m
Operating Voltage	3.5V to 5.5V
Operating current	0.3mA (measuring) 60uA (standby)
Output	Serial data

[iii]. Gas Sensor: MQ2

The MQ2 gas sensor is capable of sensing concentration of gases in the air such as LPG, smoke and carbon monoxide. The resistance of the sensing material (aluminum-oxide based ceramic coated with Tin dioxide) changes when it is exposed to gas and this change in the value of the resistance is used to detect and compute the gas content. The gas concentration measurement range of the MQ2 sensor is 200-10000 ppm [13].

[iv]. WiFi Shield: ESP8266

The ESP8266 chip is a low-cost WiFi module, which when connected to an Arduino microcontroller board allows a connection to a WiFi network and simple TCP/IP data transaction. In the proposed indoor environmental monitoring system, the Arduino Uno board connects to the IoT gateway by means of the ESP8266 WiFi Shield using the IEEE 802.11 WiFi specification, which is based on the HDG204 Wireless LAN 802.11b/g standard [14]. The ESP8266 WiFi shield is powered by the 3.3V provided by Arduino Uno.

B. Gateway: Raspberry Pi

In the proposed design of the indoor environmental monitoring system, the Raspberry Pi 3B is used as the gateway. In IOT, a gateway is a system that links the data collected by the sensors to the cloud. The Raspberry Pi is a single board credit card size computer with enough processing power and memory to support programmable I/O ports and the use of standard peripherals. The Raspberry Pi has a 64-bit quad core processor running at 1.4 GHz, inbuilt WiFi, and 40 general purpose input output pins (GPIO). The board can be powered with a power bank of up to 2.5 Amps [15]. The technical specifications [15] of the Raspberry Pi3B are listed in Table 6 and the pin diagram is shown in Appendix-1.

Table 6: Raspberry Pi 3B Specifications

Microcontroller	ATmega328P
Microprocessor	Broadcom BCM2837 64bit Quad Core Processor
Processor Operating Voltage	3.3V
Raw Voltage input	5V, 2A power source
Maximum current through each I/O pin	16mA
Maximum total current drawn from all I/O pins	54mA
Flash Memory (Operating System)	16Gbytes SSD memory card
Internal RAM	1Gbytes DDR2
Clock Frequency	1.2GHz
Ethernet	10/100 Ethernet
Wireless Connectivity	BCM43143 (802.11 b/g/n Wireless LAN and Bluetooth 4.1)
Operating Temperature	-40ºC to +85ºC

C. System Software

The system software is divided in two key features, the visualization platforms and the support scripts that run in all the nodes. The processing of sensor data and central management of the customers and their sensor nodes is accomplished on the AWS IoT cloud based platform. The Amazon AWS IoT is capable of interfacing with end point devices, to monitor and control, providing the capability of creating rules mechanisms and additional features such as storage. To interface with this solution, all devices must implement the AWS IoT Device SDK, in order to communicate with the platform through MQTT or HTTP. In the prototype design, the sensor node data that is transferred to the IoT Core "Topic" is in turn stored in a MySQL database on AWS RDS instance. This is done by creating an action rule in IoT core that publishes data to a Lambda function. The Lambda function inserts the data into the MySQL database. The sensor node data stored in the MySQL database of the AWS RDS is made available to external client applications such as the mobile app and web browser using REST APIs. A nodejs-Angular application is used to create the APIs that send dashboard data and perform configuration settings required for installation at each customer site. This application is deployed on a S3 bucket using AWS EC2-Beanstalk service and accessed through URL configured in AWS CloudFront.

5 COMPONENT DESIGN

5.1 Sensor Node *(Shuaib)*

5.1.1 Specifications

The sensor node comprises the low-cost environmental sensors (DHT11 & MQ2), Arduino Uno microcontroller, ESP8266 WiFi shield and a buzzer. It is responsible for real-time indoor temperature, humidity and gas content data acquisition and transmission of the sensed data to the AWS cloud platform through the Raspberry Pi gateway. The Arduino Uno was selected to due to its simplicity, robustness, low cost, cross-platform compatibility and the presence of its own Integrated Design Environment (IDE). Further, the Arduino Uno's analog/digital inputs, serial communication functionality (SCL and SDA) and the clock speed is sufficient to design the monitoring system. The DHT11 temperature and humidity sensor was selected because it possesses excellent long-term stability and can be easily interfaced with Arduino Uno microcontroller board with the help of DHT library and connecting wires. Similarly, the use of the MQ-2 sensor facilitates easy interfacing with the Arduino microcontroller for the detection of toxic pollutants such as carbon monoxide (CO), LPG gas and smoke inside the indoor environment. The ESP8266 WiFi Shield was selected because of its low-cost and capability of providing full-stack TCP/IP and supporting the MQTT protocol for sensed data transmission to the IoT gateway. The Arduino WiFi Shield allows the Arduino board to connect to the IoT gateway using the 802.11 wireless specification (WiFi). It is based on the HDG204 Wireless LAN 802.11b/g System in-Package. Data transmission from the sensor node to the IoT gateway is achieved by utilizing the Message Queuing Telemetry Transport (MQTT) protocol which has a lower protocol overhead when compared to HTTP.

5.1.2 Design and Verification

The design and implementation of the prototype's sensor node involve the following tasks:

A. Installation of the Arduino IDE.

The Arduino is first installed after downloading it from the https://www.arduino.cc/en/main/software.

B. Installation of the ESP8266 Board in Arduino IDE.

Following the Arduino IDE installation, the ESP8266 board WiFi is installed in the Arduino IDE. The step-by-step procedure used in the ESP8266 installation and the associated screenshots are presented in Appendix-2.

C. Installation of Sensor Libraries in Arduino IDE

To install the DHT-11 and MQ-2 sensor libraries into the Arduino IDE, the Library Manager *(available from IDE version 1.6.2)* is utilized. The step-by-step procedure used in the installation of the sensor libraries and the associated screenshots are presented in Appendix 2.

D. Circuit Connections

The Arduino Uno microcontroller is programmed to interface with the sensors and the ESP8266 WiFi shield with the Arduino IDE via a Type B USB cable. A plug and play mode for the end-user is facilitated in the sensor node by pre-programming the microcontroller's input pins to receive data either from DHT-11 or MQ-2 sensors. The interfacing of the DHT11 temperature/humidity sensor and the MQ-2 gas sensor to the Arduino Uno microcontroller involves the following steps:

i. Connection of the Ground and Vcc pins of the DHT11 temperature/humidity sensor to the Ground and 5V pins of the Arduino Uno microcontroller.
ii. Connection of the Data pin of the DHT11 sensor to the pin D5 of the Arduino Uno.
iii. Connection of the Ground and Vcc pins of the MQ-2 gas sensor to the Ground and 3V pins of the Arduino Uno.
iv. Connection of the Data pin of the MQ-2 sensor to the pin A0 of the Arduino Uno microcontroller.
v. Installation of the DHT11 and MQ-2 libraries on the IDE platform and running the code for calibrating and activating the DHT11 sensor.

The connection diagram for the sensor node is shown in Fig.9 and the sensor node picture is shown in Fig. 10.

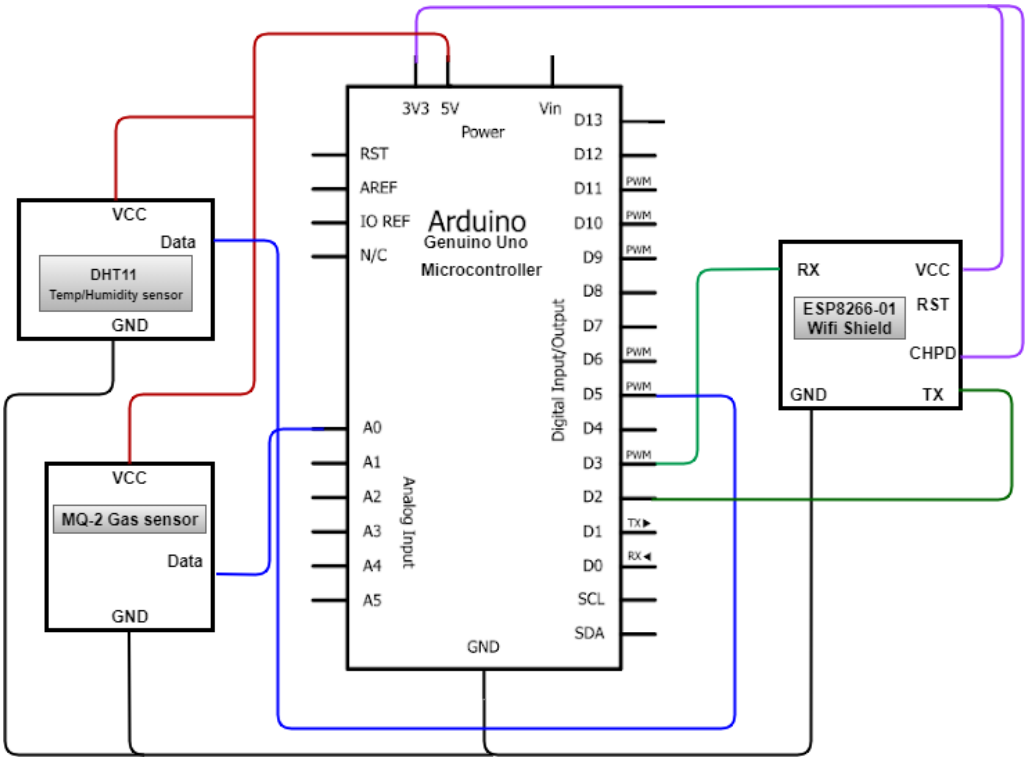


Figure 9: Sensor Node Circuit Diagram

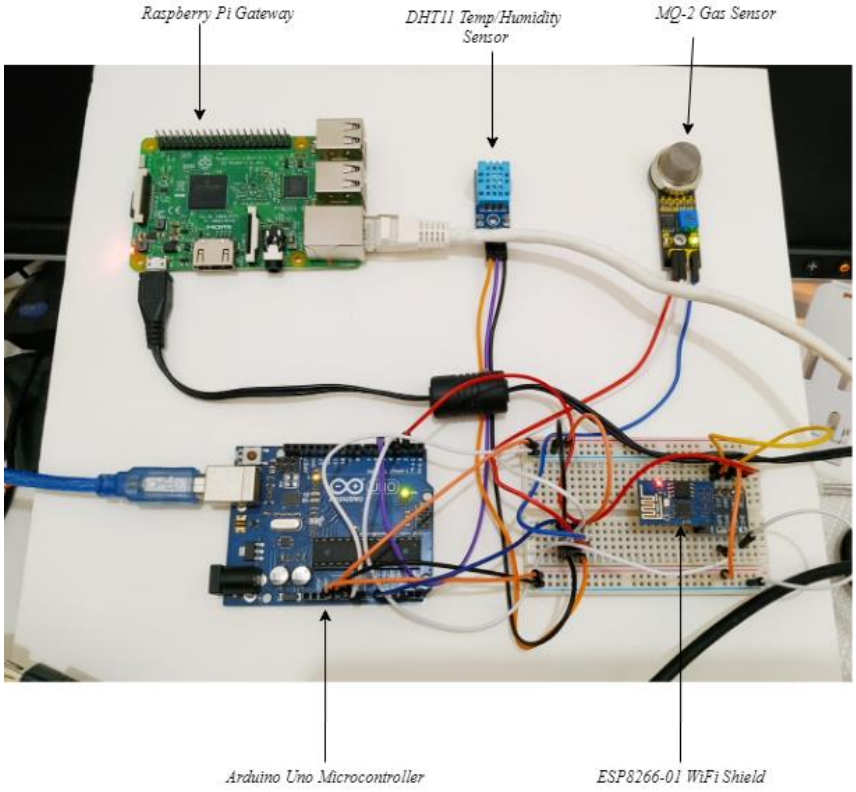


Figure 10: Prototype Picture after Circuit Connections

E. Execution of the Arduino C code

The indoor environmental data acquisition by the sensor node is initiated by executing the Arduino C code. This 'C' code (*sketch_mar22b_FinalCode_Final_Code_030420_.ino*) is listed in Appendix-3. A screenshot of the results displayed on the COM port after the C code execution is shown in the composite Fig. 11.

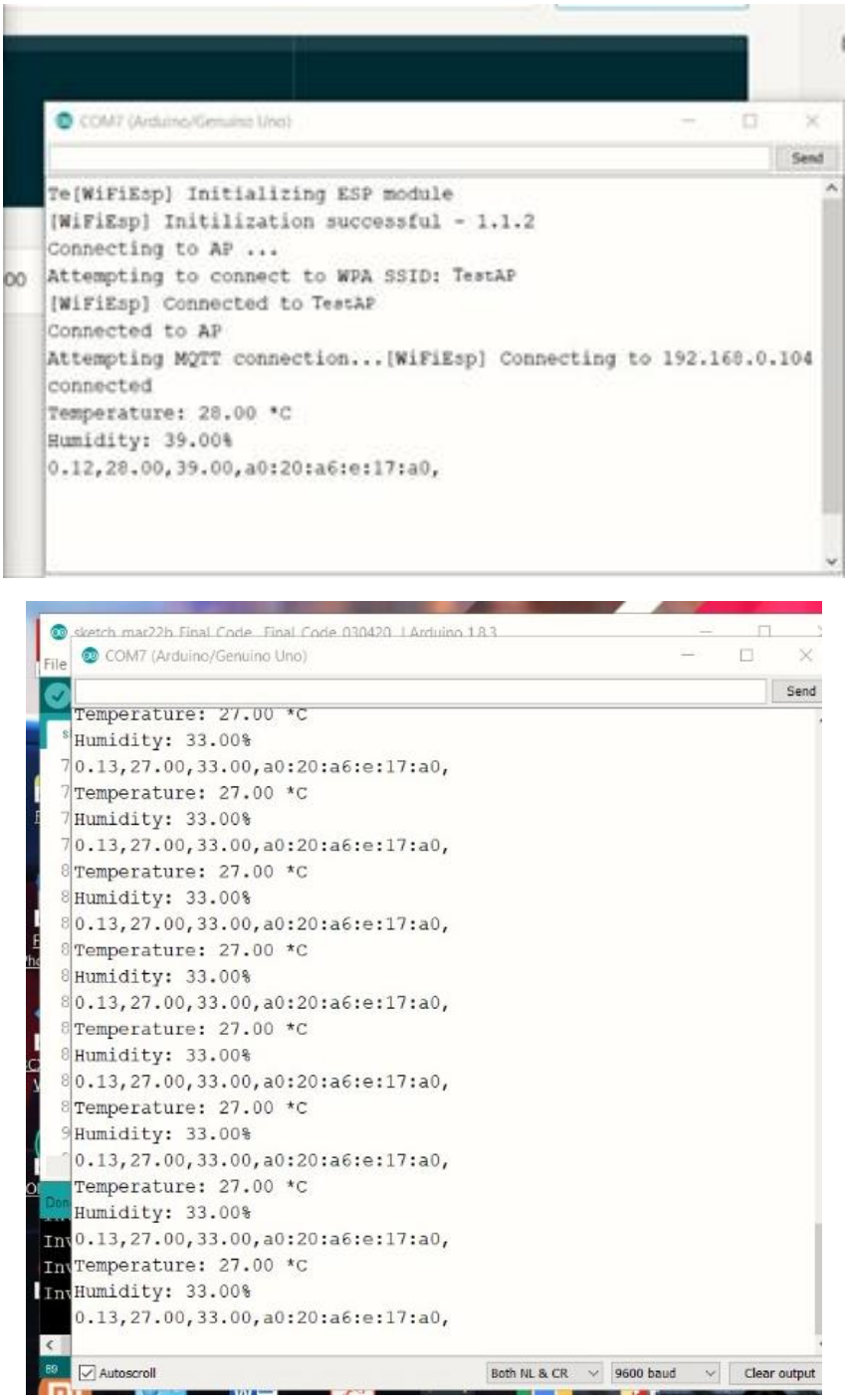


Figure 11: Arduino C code execution and screenshot displaying the measured indoor environmental parameters

5.2 IoT Gateway *(Shuaib & Khalid)*

5.2.1 Specifications

In the proposed design of the indoor environmental monitoring system, the Raspberry Pi 3B is configured as the IoT gateway. The Raspberry Pi 3B was chosen because of its low-cost, processing power, storage and low-power requirement. It also provides reasonable functionalities to execute the gateway tasks and offers a great scalability factor. It is also very compact and easy to configure and maintain. Data received from the sensor node is stored in the MariaDB database (local host) before uploading to the AWS cloud (IoT core) using the MQTT data protocol. MQTT is the standard protocol for messaging and data exchange for the Internet of Things. The protocol uses a publish/subscribe architecture and enables a publish/subscribe messaging model in an extremely lightweight way. It is useful for connections where network bandwidth is at a premium. The open source *MySQL* and *MariaDB* administration tool *phpMyAdmin* is also utilized in the gateway.

5.2.2 Design and Verification

In the prototype, the Raspberry Pi 3 gateway receives sensor data from the sensor nodes and stores it in a local *Maria dB* database before finally sending it to the AWS IoT core via the MQTT protocol. Raspbian Linux was installed and used as the operating system for the IoT gateway. The local database in the Raspberry Pi 3 gateway ensures that no sensor data is lost when there is no Internet connection.

The configuration of the Raspberry Pi 3 as an IoT gateway involves the following tasks which are described in detail with screenshots in Appendix-4:

i. Installation of the Raspbian Linux operating system.
ii. Installation of the Mosquitto Broker.
iii. Installation of the Python library for MQTT.
iv. Setting up of local *Maria dB* database.
v. Setting up *phpMyAdmin*.

Once the Raspberry Pi gateway is configured, the execution of the python code '*receive2.py*' will ensure that sensor data is received and temporarily stored in the local MariaDB database for later upload to the AWS IoT Core. The Python script (*receive2.py*) is listed in Appendix-5.

Fig. 12 shows the execution of the python code, while the local storage of the sensor data in the MariaDB database can be seen in Fig. 13.

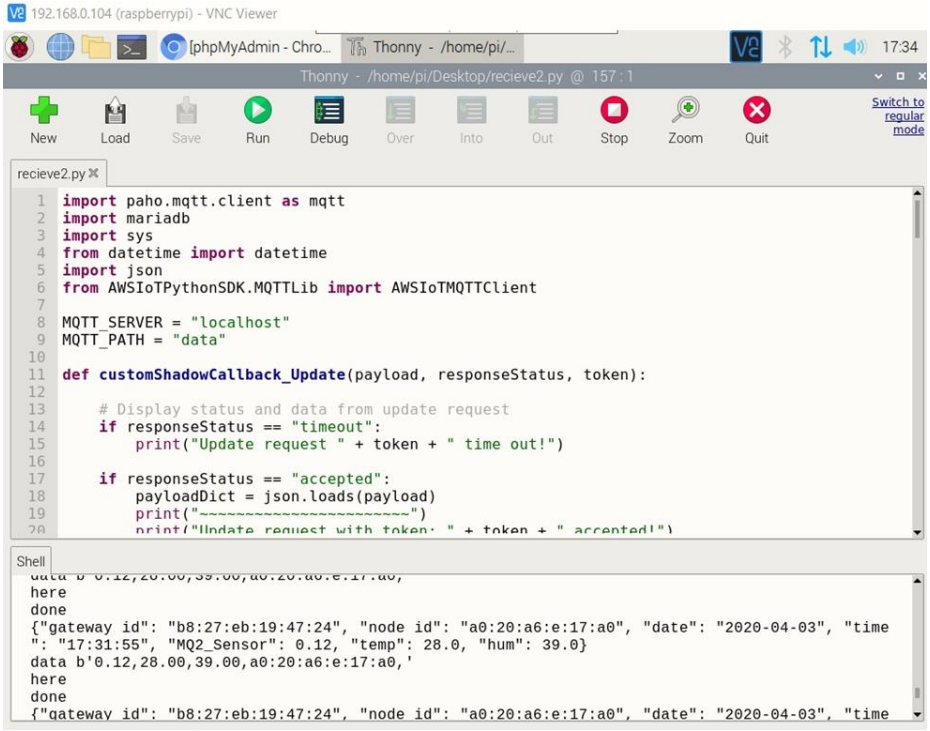


Figure 12: Raspberry Pi Python code execution

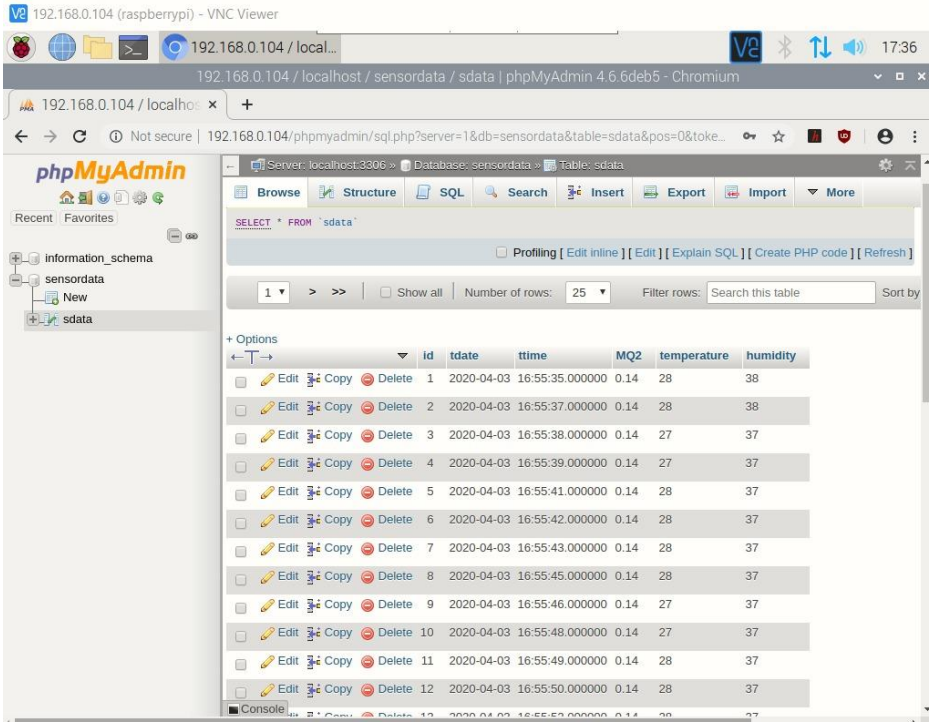


Figure 13: phpMyAdmin screenshot

5.3 AWS Cloud *(Shuaib & Khalid)*

5.3.1 Specifications

The AWS cloud platform (free tier) was selected for the prototype design because it can ensure efficient sensor node data collection, processing, storage and visualization. Further it facilitates efficient centralized management of end-users with their multiple sensor nodes. The AWS specifications utilized in the prototype to facilitate the afore-mentioned tasks are as follows:

- AWS t2 micro instance on Windows
- S3 instance with 5 GB storage
- RDS with MySQL database with 20GB storage
- IoT Core
- AWS Lambda

5.3.2 Design and Verification

The design of the AWS IoT data model for the proposed indoor environmental system involves the following tasks:

A. Connection of the Raspberry Pi Gateway to the AWS IoT *(Shuaib)*

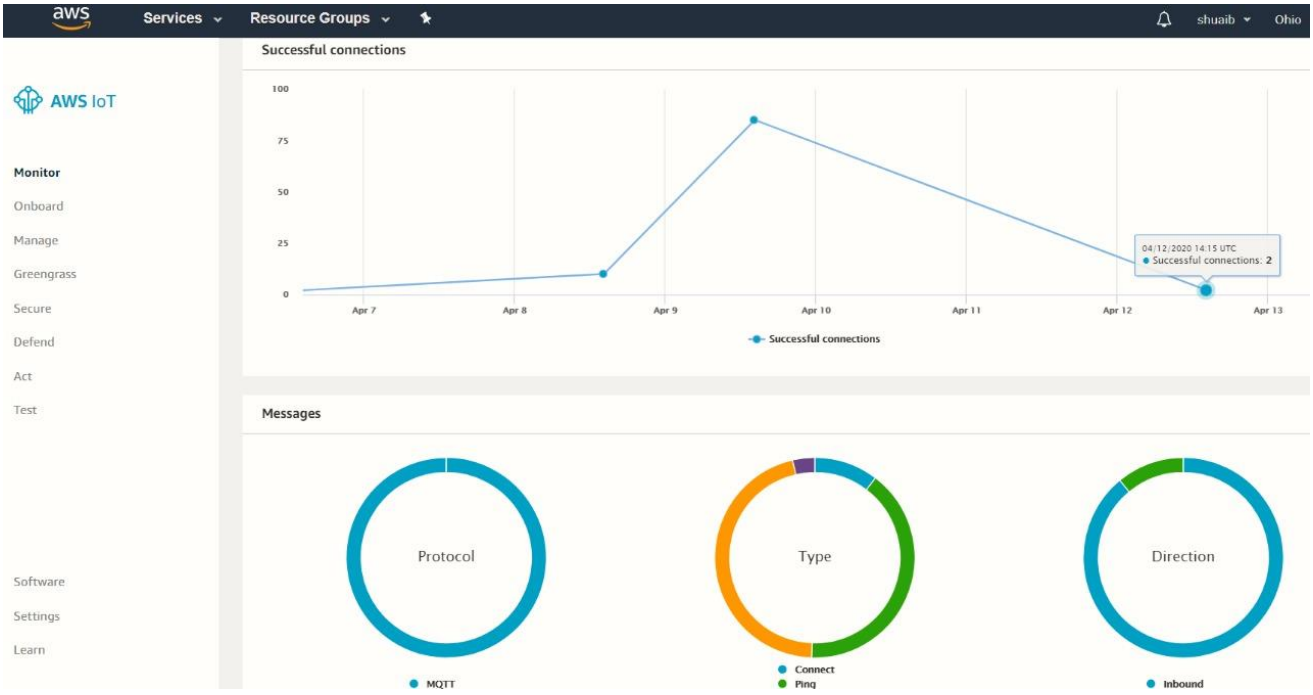


Figure 14: AWS display indicating the successful connection of the Raspberry Pi Gateway to the AWS IoT

To connect a device to AWS IoT, one should create an IoT thing, a device certificate, and an AWS IoT policy. In the proposed system, the Raspberry Pi gateway is the "thing" which upload sensor node data to the IoT core. All devices (things) must have a device certificate to connect to and authenticate with AWS IoT. Each device certificate has one or more AWS IoT policies associated with it. These policies determine which AWS IoT resources the device (Raspberry Pi in our case) can access. Devices and other clients use an AWS IoT root CA certificate to authenticate the AWS IoT server with which they are communicating. An AWS IoT rule contains a query and one or more rule actions. The query extracts data from device messages to determine if the message data should be processed. The rule action specifies what to do if the data matches the query. The rule listens for sensor node data from the Raspberry Pi gateway and sends a message to the Amazon SNS topic. Amazon SNS sends that message to those who have subscribed to the topic.

The procedures involved in connecting the Raspberry Pi gateway to the AWS IoT Core and enabling it to upload sensor node are described in detail in Appendix-6.

B. Publishing Topic on IoT Core *(Shuaib)*

A topic by name "**data**" was published and subscribed on AWS IoT Core. As shown in Fig. 15, the data that is transferred by the Raspberry Pi gateway (gateway ID: b8:27: eb: 19:47:24) can be seen on IOT Core by subscribing to the topic "data'.

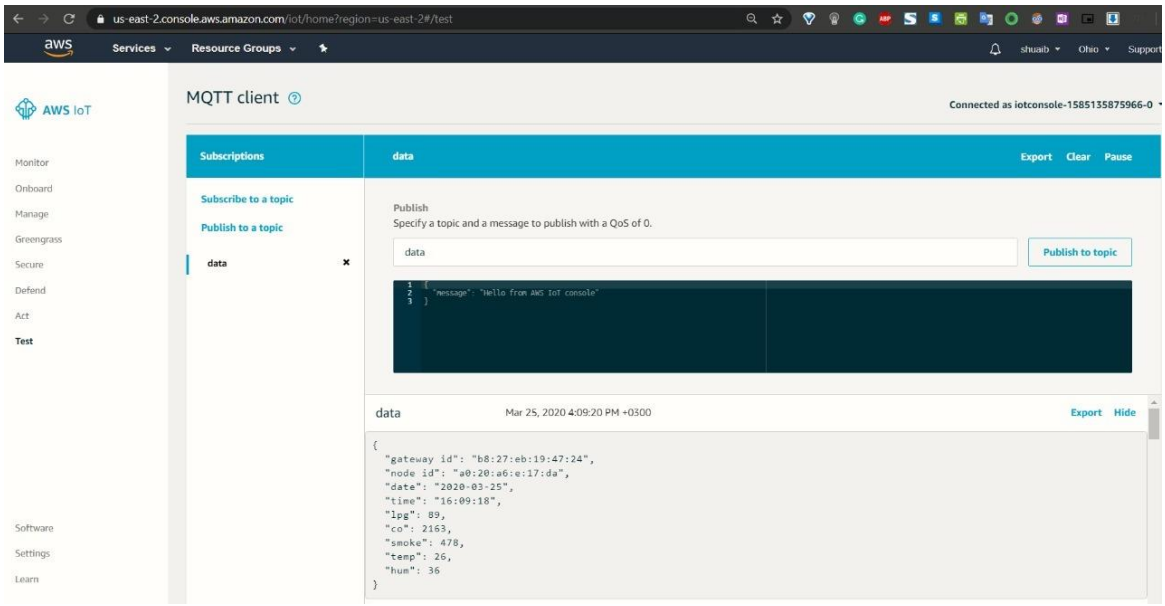


Figure 15: Publishing Topic on IoT Core

The composite Fig. 16 below shows that the readings from the sensors are the same as the values seen in IOT Core**.**

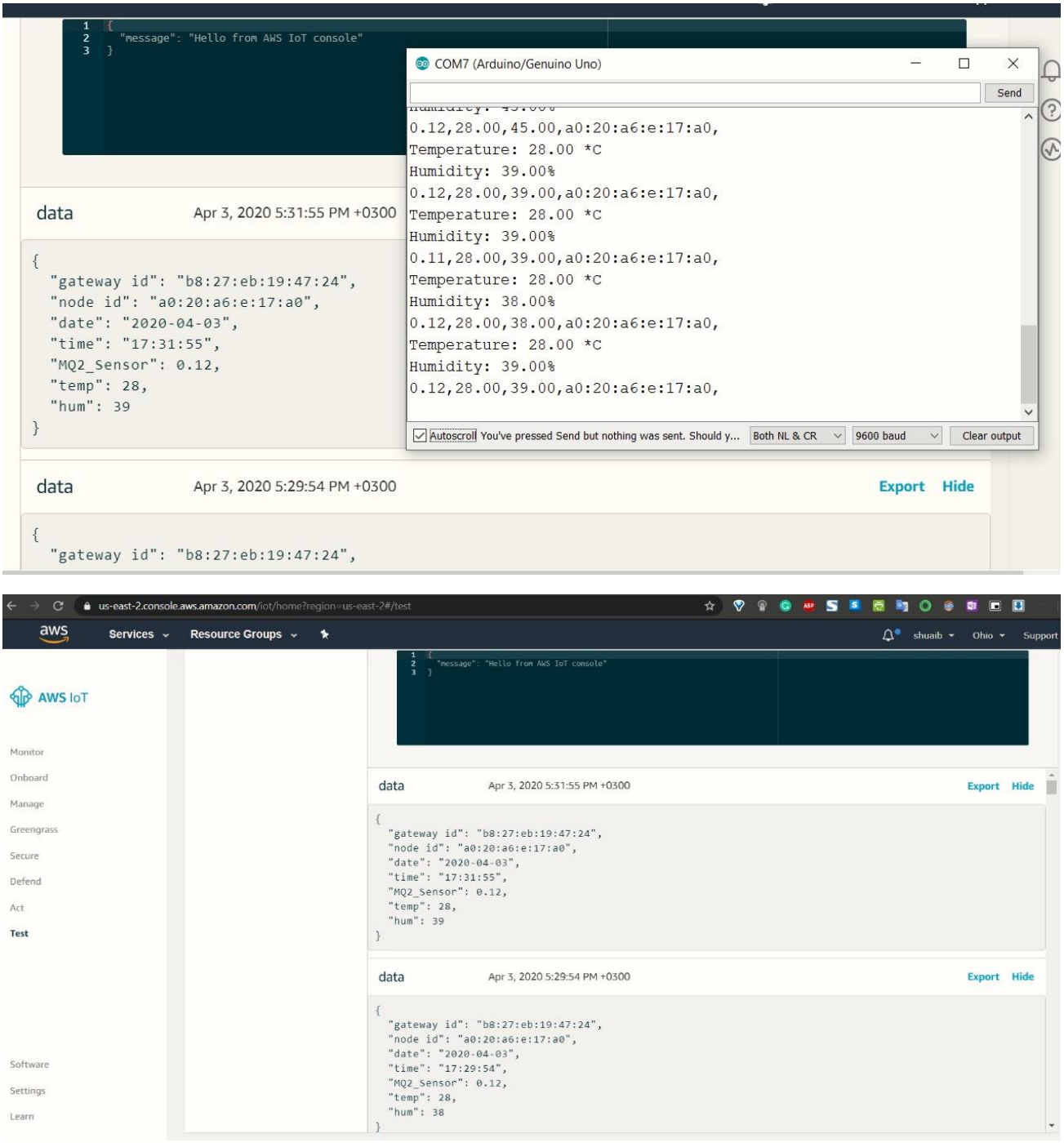


Figure 16: AWS IoT Core displaying the same sensor node data as received from the gateway

C. Enabling MySQL on RDS in AWS Free Tier *(Khalid)*

In the proposed design, Amazon RDS is utilized to create a MySQL DB Instance with db.t2.micro DB instance class, 20 GB of storage, and automated backups enabled with a retention period of one day. AWS Cloud resources are housed in highly available data center facilities in different areas of the world. Each region contains multiple distinct locations called Availability Zones. A developer can host his Amazon RDS activity in the region of his choice. In the proposed system design, Ohio region was selected. The process of installing and activating the MySQL database engine on AWS RDS involves the following steps which are detailed with screenshots in Appendix-7.

1. Login to the AWS account>>management console
2. Selecting "Create Database" option in the list of Amazon RDS services.
3. Creating a MySQL DB Instance.
4. Selecting the default MySQL database engine to be installed on RDS.
5. Configuring the DB instance with the following:

   DB instance specifications

   - License model: *general-public-license*
   - DB engine version: *MySQL 5.7.22*
   - DB instance class: *db.t2.micro --- 1vCPU, 1 GIB RAM.*
   - Storage type: *General Purpose (SSD).*
   - Allocated storage: *20 GB*

   Settings:

   - DB instance identifier: *iot-db*
   - Master Username: *admin*
   - Master password: *****
   - Confirm password: *****

6. Configuring advanced settings as follows:

   Network & Security

   - Virtual Private Cloud (VPC): *Default VPC.*
   - Subnet Group: *default*
   - Public Accessibility: *Yes.*
   - Availability Zone: *No preference*.
   - VPC security groups: *Create new VPC security group*. (This will create a security group that will allow connection from the IP address of the device that one is currently using to the database created.)

   Database Options

   - Database Name: *"iot_db"*

- Port: *3306*
- Db Parameter Group: *default.mysql5.6*
- Option Group: *default: mysql5.7.* (Amazon RDS uses option groups to enable and configure additional features.)
- IAM DB Authentication: *Disable* (This option allows the developer to manage his database credentials using AWS IAM users and groups.)

Note: The selected database engine or DB instance class does not support storage encryption.

Backup

- Backup retention period: 1 day.
- Backup window: No preference.

Monitoring

- Enhanced Monitoring: Disable enhanced monitoring (enables staying within the free tier)
- Performance Insights: Disable Performance Insights.

Maintenance

- Auto minor version upgrade: Enable auto minor version upgrade.
- Maintenance Window: No preference.
- Deletion Protection: Clear Enable deletion protection.

7. Downloading the MySQL Workbench 8.0.19 for Windows after the creation of the DB instance.

Once the database instance creation is complete and the status changes to available, a developer can connect to a database on the DB instance using any standard SQL client. In the proposed system design, the popular SQL client MySQL Workbench 8.0.19 for Windows was downloaded from the website https://dev.mysql.com/downloads/workbench/ and installed.

8. Connecting to the MySQL Database after launching the MySQL Workbench application.

Once the connection to the database is made, the developer can now browse different schema objects, create tables and run queries.

D. Database Design *(Khalid)*

[1]. Database Tables

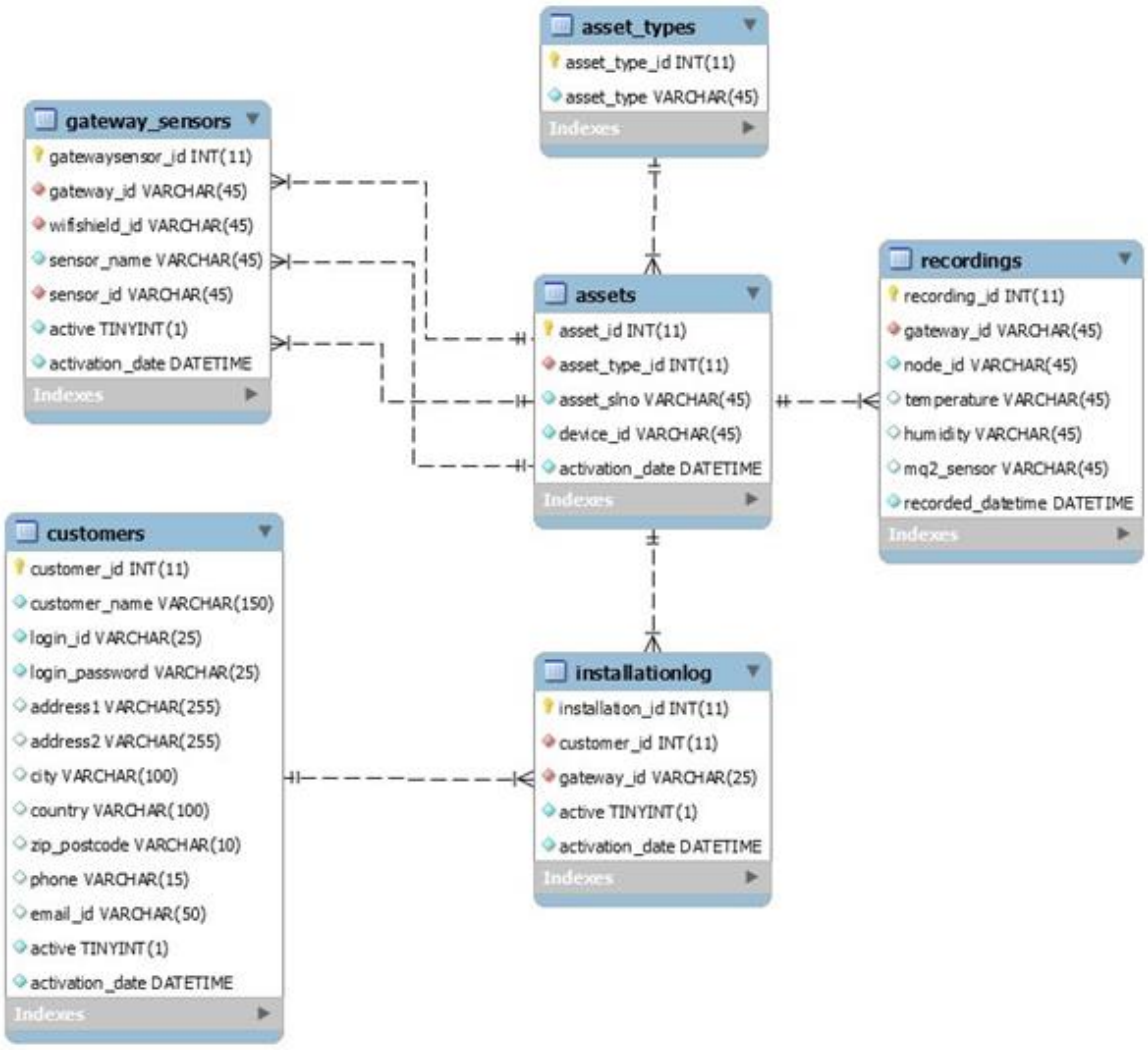


Figure 17: Entity Relationship Diagram

The database tables utilized for the creation of the Indoor Environmental Monitoring System's AWS MySQL database are depicted in the entity relationship diagram in Fig. 17. These tables were created using SQL scripts (iot_db_20200413) which are listed in Appendix-8. The various tables that were created and their purpose are described next.

i. Table name**: customers**

This table holds all the information of customers including their login credentials. This table is populated when the System admin registers a customer. The system admin has predefined login credentials similar to that of a super user and hence that is not included in a separate users table. The table attributes are listed below.

Table 7: Customers

Column	datatype	size	allow null	key	default value	comments
customer_id	integer	11	N	Primary key		autoincrement
customer_name	varchar	150	N			
login_id	varchar	25	N	unique		
login_password	varchar	25	N			
address1	varchar	255	Y			
address2	varchar	255	Y			
city	varchar	100	Y			
country	varchar	100	Y			
zip_postcode	varchar	10	Y			
phone	varchar	15	Y			
email_id	varchar	50	Y			
active	boolean		N			
activation_date	datetime		N			

ii. Table name: **asset_types**

The table asset types is a master table that holds the types of items required for this setup – *viz* gateway, node (wifishield) and sensors. The table attributes are listed below.

Table 8: asset_types

Column	datatype	size	allow null	key	default value	comments
asset_type_id	integer		N	Primary key		autoincrement
asset_type	varchar	25	N	Unique		Gateway, WiFi shield, Sensor

iii. Table name: **assets**

The assets table holds the list of all individual items procured by the service provider. It has different attributes like asset_slno and device_id to uniquely identify the asset. The asset_type_id attribute is a foreign key from asset_types table that tells whether the item is a gateway, node or a sensor.

Table 9: assets

Column	datatype	size	allow null	key	default value	comments
asset_id	integer		N	Primary key		autoincrement
asset_type_id	integer		N	Foreign key		FK connected to asset_type_id of asset_types table
asset_slno	varchar	25	N	unique		
device_id	varchar	25	N	unique		
activation_date	datetime		N			

iv. Table name: **gateway_sensors**

The service provider prepares kits to be installed by grouping gateway, nodes and sensors. In this table, the columns gateway_id, wifishield_id, sensor_id are all foreign keys linked to asset_id of assets table. The gateway_sensors table holds the information of the configuration of such kits.

Table 10: gateway_sensors

Column	datatype	size	allow null	key	default value	comments
gatewaysensor_id	integer		N	Primary key		autoincrement
gateway_id	varchar	25	N	Foreign key		FK connected to device_id of assets table
wifishield_id	varchar	25	N	Foreign key		FK connected to device_id of assets table
sensor_name	varchar	25	N			
sensor_id	varchar	25	N	Foreign key		FK connected to device_id of assets table
active	boolean		N			
activation_date	datetime		N			

v. Table name: **installationlog**

When kits are installed at the customer site, the system admin creates records of such installations in this table along with the activation date. The table attributes are listed below.

Table 11: installationlog

Column	datatype	size	allow null	key	default value	comments
installation_id	integer		N	Primary key		autoincrement
customer_id	integer		N	Foreign key		FK connected to customer_id of customers table
gateway_id	varchar	25	N	Foreign key		FK connected to device_id of assets table
active	boolean		N			
activation_date	datetime		N			

vi. Table name: **recordings**

This table is the most important table in the database that stores all the parameter values recorded by each sensor. The table attributes are listed in Table 12 below:

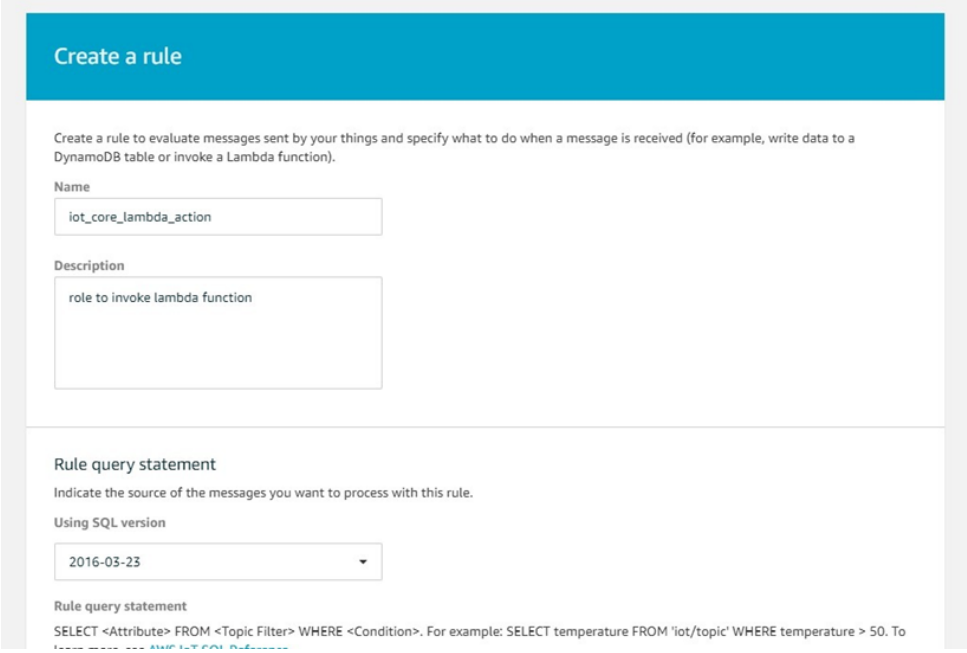
Table 12: recordings

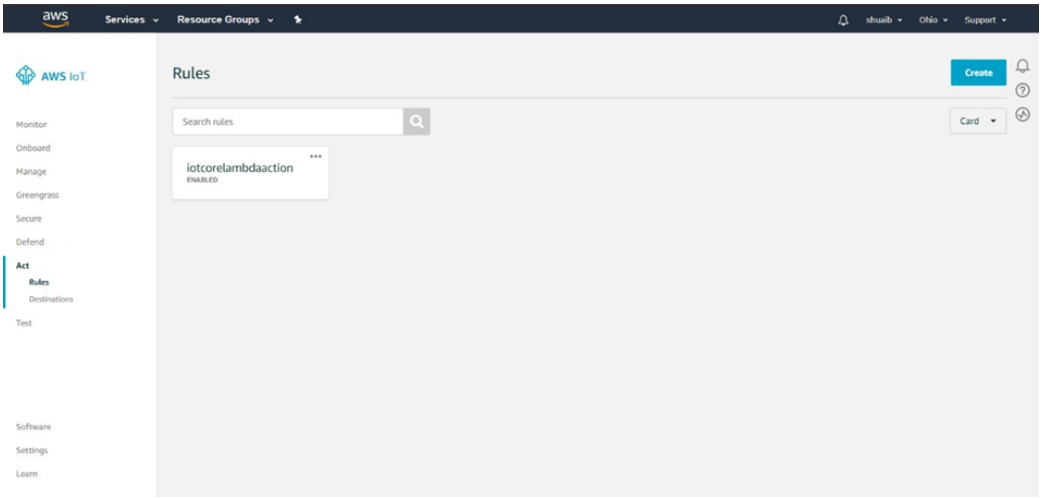
Column	datatype	size	allow null	key	default value	comments
recording_id	integer		N	Primary key		autoincrement
gateway_id	varchar	25	N	foreign key		FK connected to device_id of assets table
node_id	varchar	25	N	foreign key		FK connected to device_id of assets table
temperature	varchar	25				
humidity	varchar	25				
mq2_sensor	varchar	25				
recorded_datetime	datetime		N			
sensor_status						

[2]. Lambda Functions Rule Creation and Configuration *(Shuaib)*

1. Create Rule

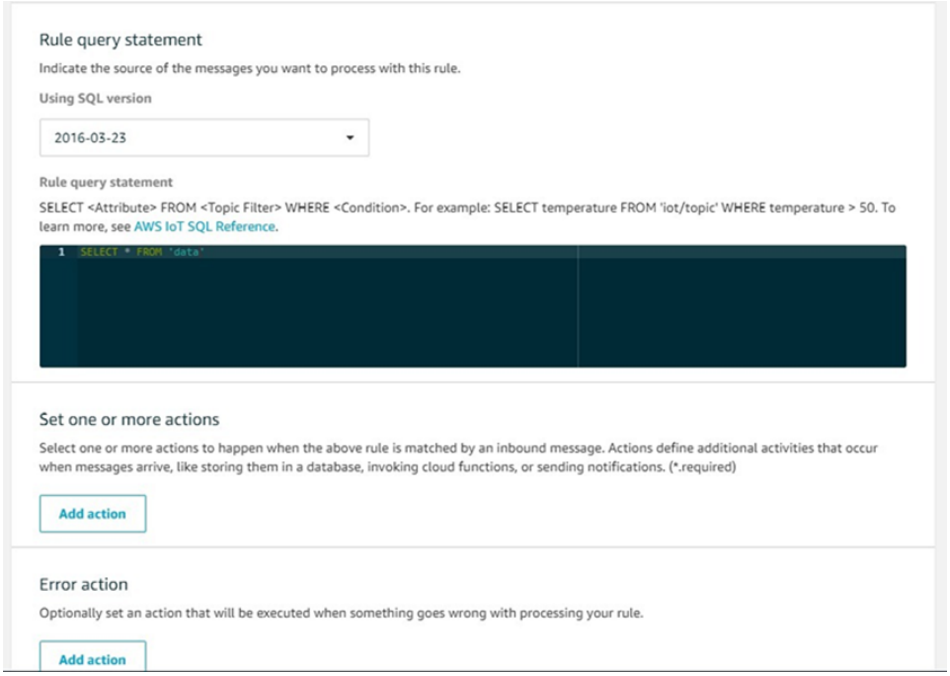
Action rule has to be created and enabled for Lambda function to act. The process of creating and configuring a Rule is shown below:



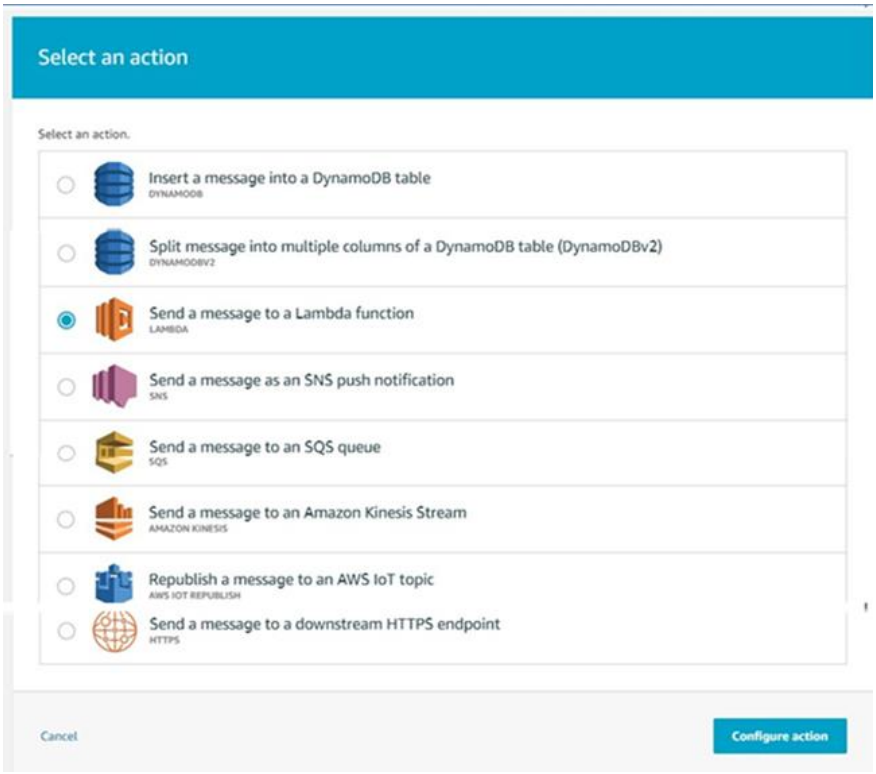


As shown in the above screenshot, under ACT>>'Rules', Create the Rule using the Create button on top right of the screen. Here a rule called '**iotcorelambdaaction**' has been created.

2. Provide a Query Statement for the Rule

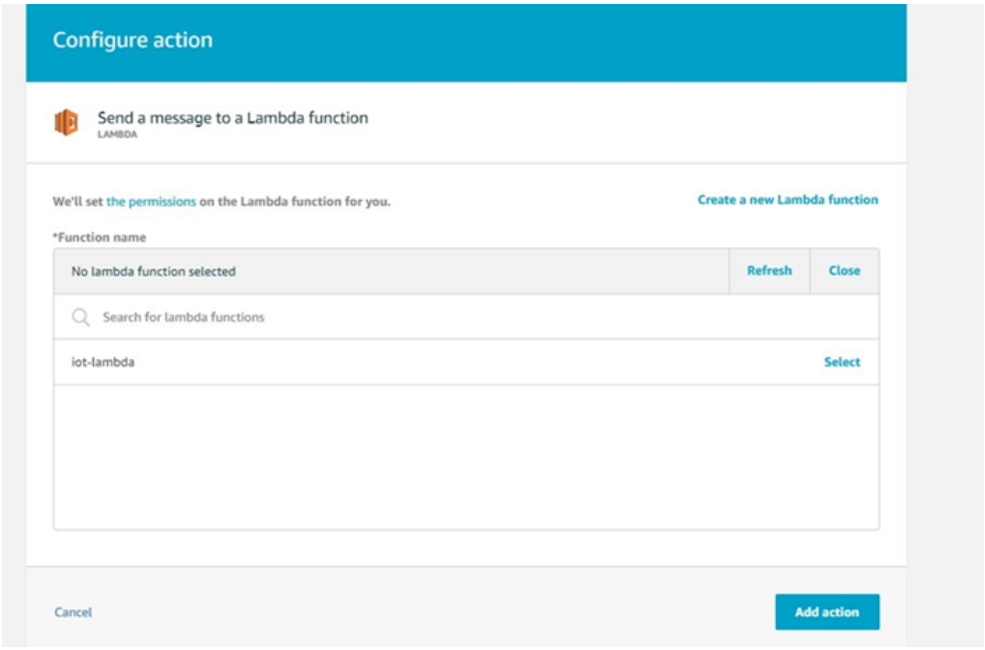


3. Select an Action to be Performed

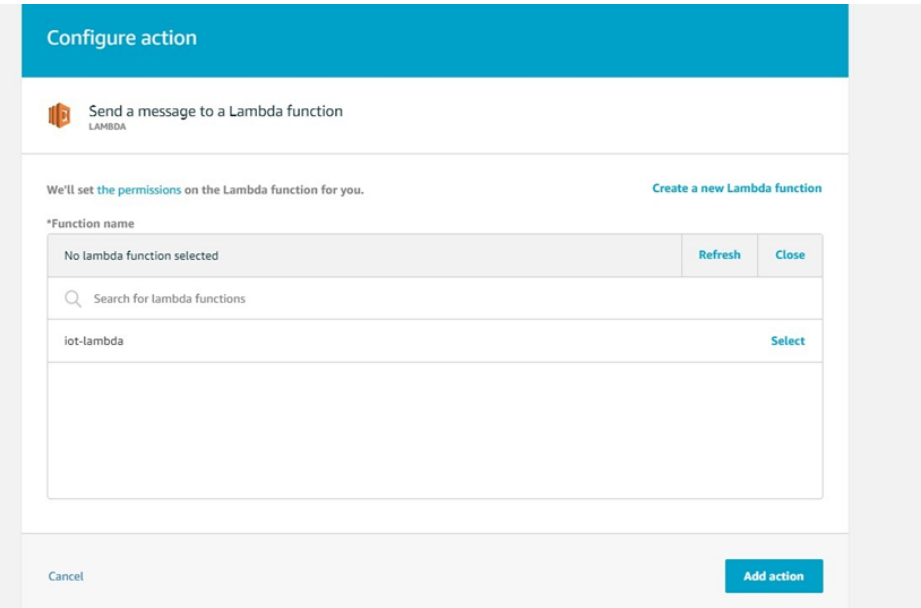


In the above screen, select ‘Send message to a Lambda function’ option and click on ’Configure action button.

4. Create and Select Lambda Function



In the above screen, click on Create Lambda function button on top right to create a Lambda function. Search and select the created Lambda function and click on ‘Add Action’ button in the bottom right of the screen. Then Lambda function is configured.



E. Data Population in MySQL *(Shuaib)*

Data is fetched from IoT Core and inserted into MySQL *recordings* table using Lambda functions. We have to first subscribe to the topic published in IoT Core, then create Rules by which data is fetched from IoT core topic. The following code in the index.js file connects to database and inserts the data into the MySQL *recordings* table:

```
var mysql = require("mysql");
var connection = mysql.createConnection({
  host: "iot-db.c3necvlghx7q.us-east-2.rds.amazonaws.com",
  user: "admin",
  password: "iotadmin",
  database: "iot_db",
});

// console.log(connection);
exports.handler = (event, context, callback) => {
  console.log("Received event:", JSON.stringify(event, null, 2));

  const query =
    "INSERT INTO recordings  (gateway_id, node_id, temperature, humidity,
mq2_sensor, recorded_datetime, sensor_status) VALUES ('" +
    event.gateway_id +
    "', '" +
    event.node_id +
    "', '" +
    event.temp +
    "', '" +
    event.hum +
    "', '" +
    event.MQ2_Sensor +
    "', '" +
    event.date +
    " " +
    event.time +
    "',"; +
   event.sensor_status +
    "')";
  console.log(query);
  connection.query(query, function (error, results, fields) {
    if (error) {
      connection.destroy();
      throw error;
    } else {
      // connected!
      console.log(results);
      callback(error, results);
      connection.end(function (err) {
        callback(err, results);
      });
    }
  });
};
```

Fig. 18 screenshot displays the data being stored in the MySQL “recordings” table:

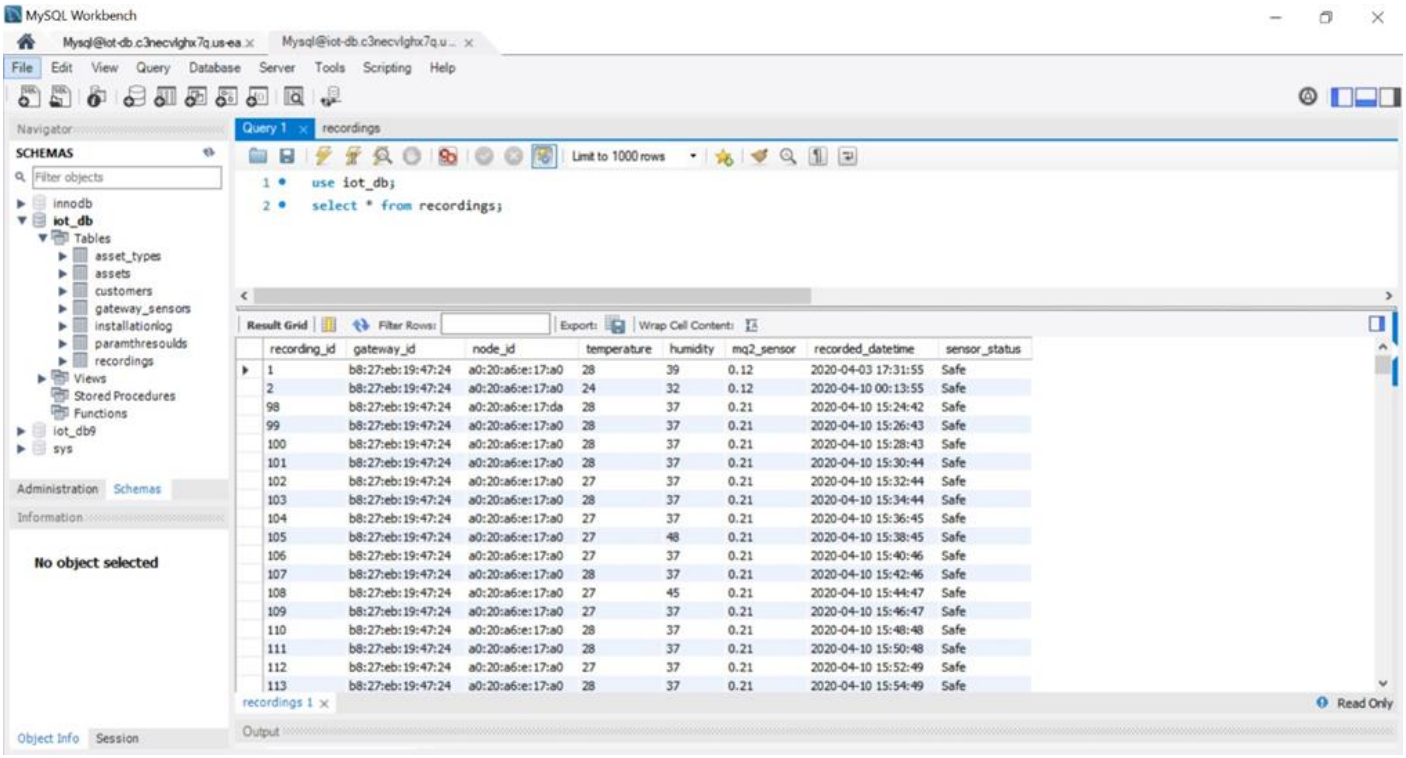


Figure 18: Data being stored in the MySQL “recordings” Table

The sensor node data that is stored in the MySQL database of the AWS RDS will have to be made available to external client applications like a mobile app or a web browser through the implementation of REST APIs. The details of Web and Mobile application development are discussed next in sections 5.4 and 5.5.

5.4 Web Dashboard Application *(Shuaib)*

5.4.1 Specifications

The development of the web dashboard requires the sensor node data stored in the MySQL database of the AWS RDS. This sensor node data is made available using REST APIs. In the system prototype design, the REST APIs have been implemented using nodejs. The same APIs would be called from web application or from a mobile app. The Nodejs component is deployed using AWS Beanstalk service. To create the front end for the web application, the Angular application has been utilized in the prototype design. The Angular application is deployed on an AWS S3 bucket. AWS CloudFront service is used to create a URL for the application. The components of the web dashboard environment are (i). AWS Beanstalk, (ii). AWS S3, (iii). AWS CloudFront, (iv). Nodejs and (v). Angular.

5.4.2 Design and Verification

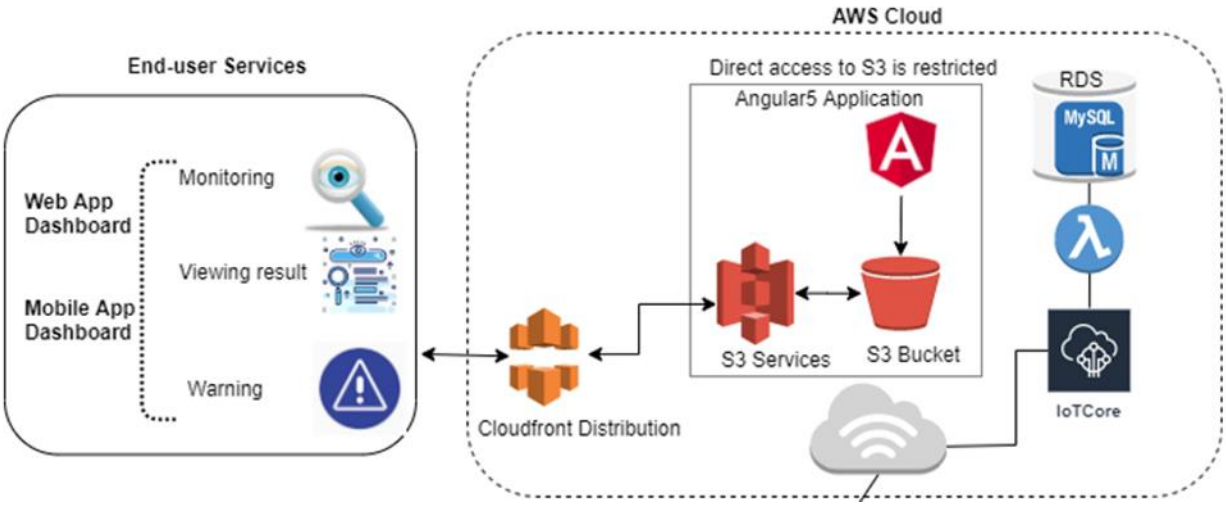


Figure 19: Overview of the AWS Cloud Deployment Environment

The 'Web Application" consists of 2 parts – the server side and the client side. The design of the server component and the front-end component are described in the following sections:

A. <u>Server Component</u> *(Shuaib)*

The server component is where the REST APIs are implemented. The server component is programmed using nodejs and it provides the APIs for the client to access. The Nodejs component is deployed using AWS Beanstalk service. The AWS Elastic Beanstalk is an orchestration service offered by AWS for deploying web applications developed with Node.js, Java, .NET, or Python. The logic of connecting to the MySQL database

retrieving/inserting data is implemented in different methods accessible to these APIs. The APIs are accessed by the client components (web browser or mobile app) using HTTP protocol. The specifications of the APIs are listed in Table 13.

Table 13: REST API's List

Base URL: http://iotadmin-env.eba-tm2thnb9.us-east-2.elasticbeanstalk.com/
*NOTE: Base URL must be added before the API routes while calling the APIs
Login API
Method: POST
URL: /login

Sample request
 	
Request :
{
email: "abc@ada.com”,
password: "12312"
}

Forgot password API
Method: GET
URL: /forgot-password/:email
Sample request – Pass email in place of :email /forgot-password/: abc@ada.com

Latest readings of all parameters to display in customer dashboard
Method: GET
URL: /recording/latest-record/:customerId
Sample request – pass customerid value in place of :customerid /recording/latest-record/:1
Graph data for temperature
Method: GET
URL: /recording/graph/temperature/:customerId/:day
Sample request: - Pass customer id in place of customerId and 1 or 2 for day (1 for current day and 2 for last 7 days /recording/graph/temperature/:1/:1
Graph data for humidity
Method: GET
URL: /recording/graph/humidity/:customerId/:day
Sample request: - Pass customer id in place of customerId and 1 or 2 for day (1 for current day and 2 for last 7 days /recording/graph/humidity/:1/:1
Tabular data for gas alerts for current day
Method: GET

URL: /recording/graph/gases/:customerId
Sample request: - Pass customer id in place of customerId /recording/graph/temperature/:1/:1
Change customer password
Method: POST
URL: /customers/change-password

Sample request –
Request:
{
customerId: 1,
oldPassword: 'oldpassword',
newPassword: 'newpassword'
}

Request nodes by customer
Method: POST
URL: /customers/request-nodes

Sample request-
Request :
{
request: "request text"
}

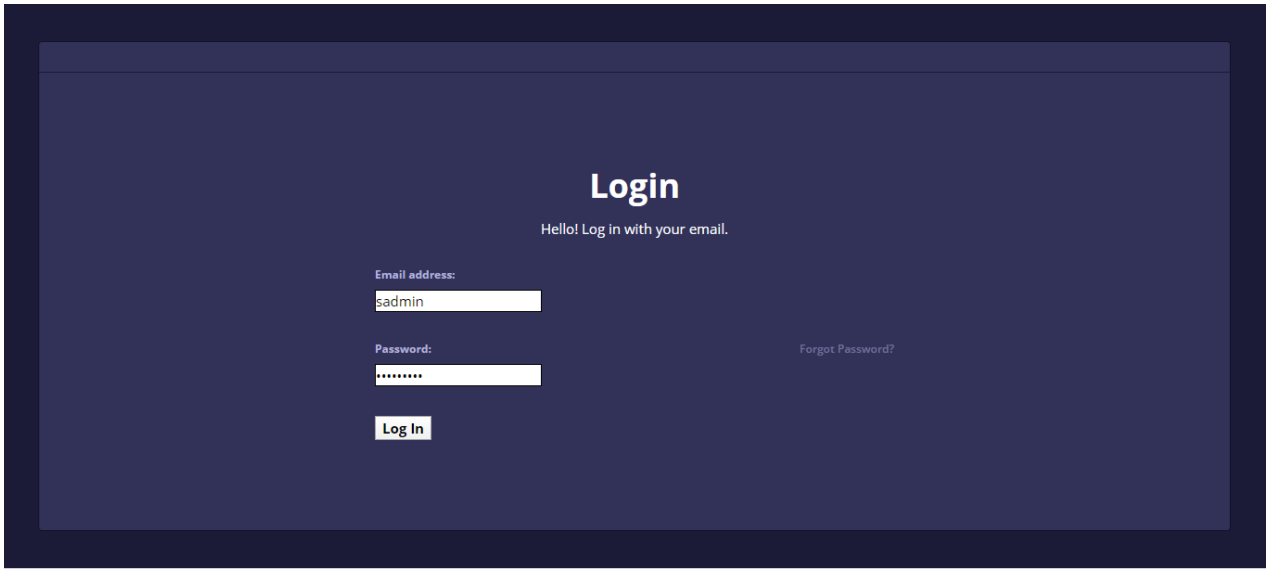
B. Angular Web UI Frontend *(Shuaib)*

The Web UI Frontend (client part) is implemented using Angular. The angular front end component is deployed on the AWS S3 bucket. The S3 bucket is made accessible by the AWS CloudFront service. The AWS CloudFront creates the URL to access the web dashboard. http://dilqu5d4hnja7.CloudFront.net/ is the AWS CloudFront URL to access the web application designed for the system prototype. The Web UI has the following features:

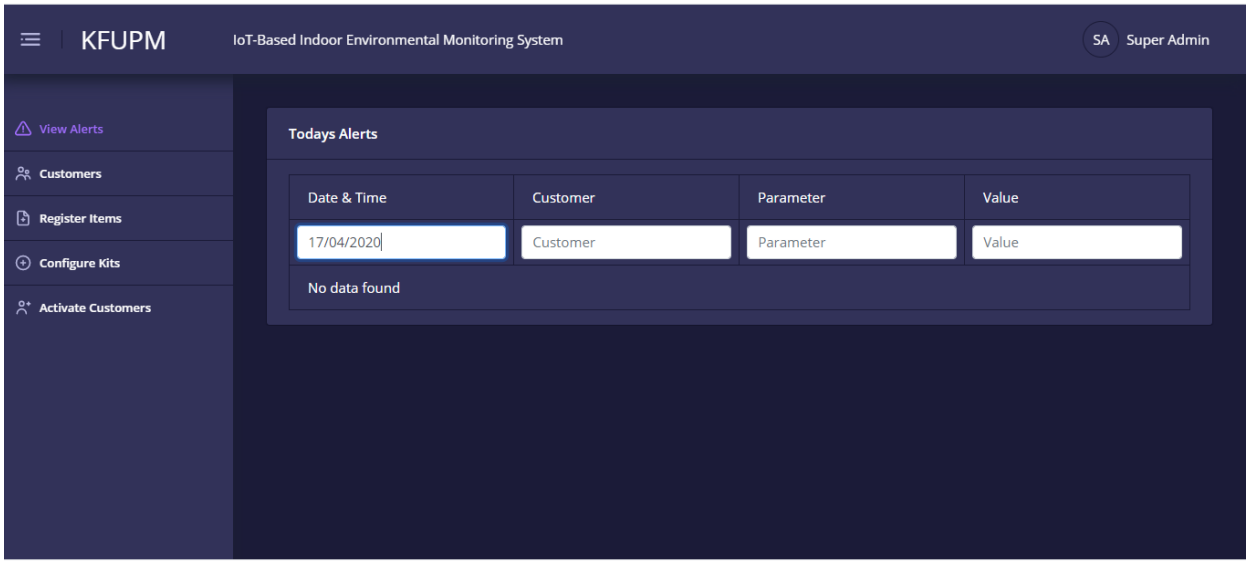
I. SYSTEM ADMIN LOGIN

A system admin login is provided with predefined user id and password. The system admin can perform the following actions:

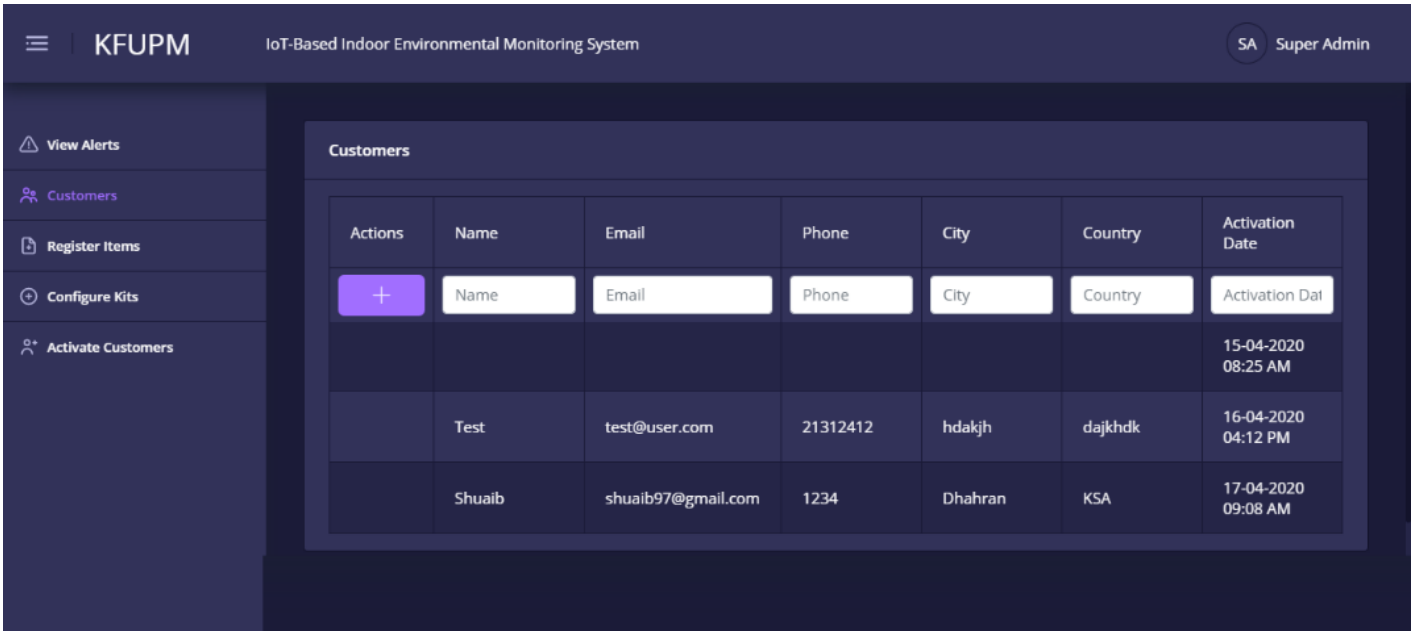
a) LOGIN – In this page system admin can login using the predefined user id and password.



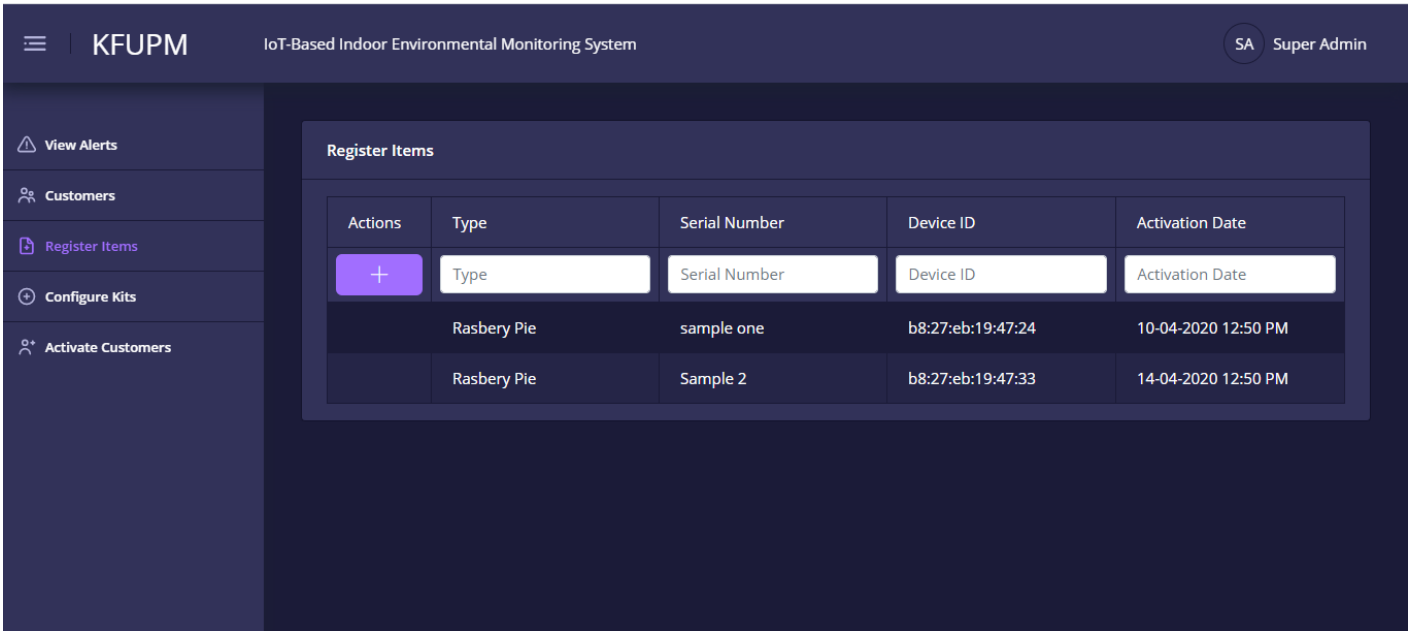
b) VIEW ALERTS – This page will display "Todays alerts". It shows all recordings of all the customers that are beyond the thresholds.



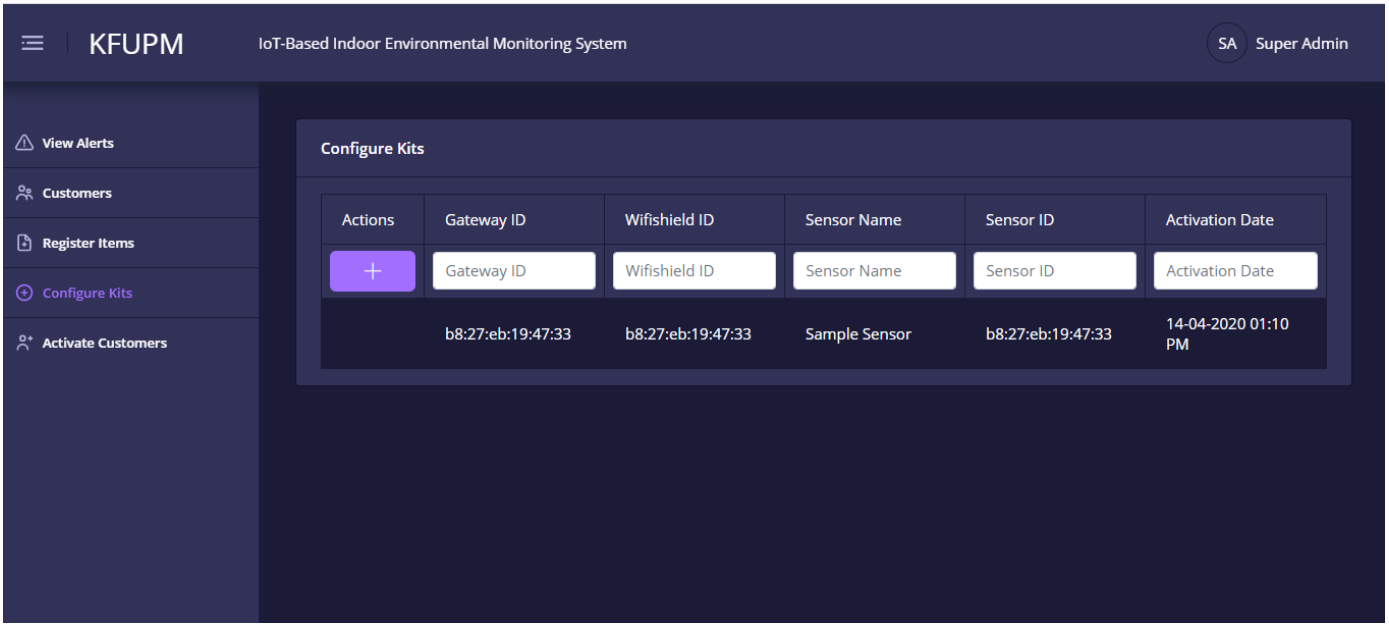
c) **REGISTER CUSTOMER-** In this page, the admin can register new customers who want to avail the indoor environmental monitoring service.



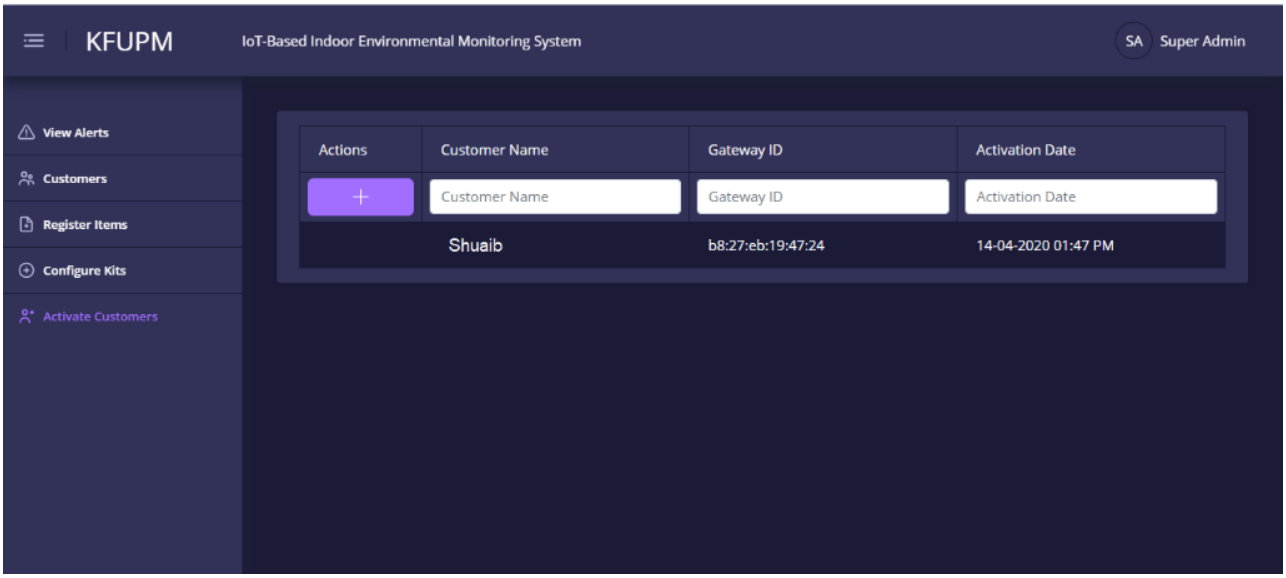
d) **REGISTER ITEMS** – In this page, the system admin can register gateways, nodes and sensors that are procured to be installed at customer sites.



e) CONFIGURE KITS – In this page, the system admin can group different hardware components such as Raspberry Pi gateway, node and sensors as one “Kit” that can be installed in a customer site.

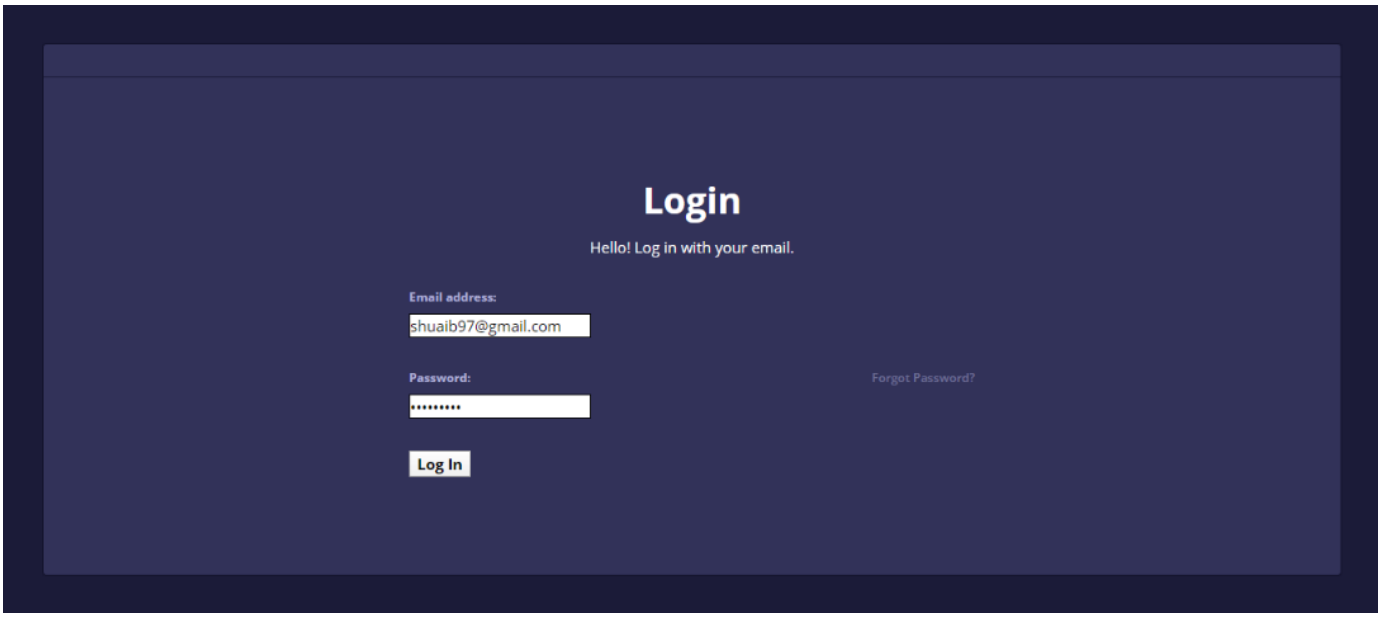


f) ACTIVATE CUSTOMERS – In this page, the system admin, records the data of the installations at each customer site.



II. CUSTOMER LOGIN

a) LOGIN PAGE- This is the initial page which facilitates the customer to login with his credentials.



b) WEB DASHBOARD – After logging in, the customer has view the web dashboard which displays the monitored indoor environmental parameters such as temperature, humidity and whether the indoor air condition is safe or unsafe (polluted with CO, smoke or LPG).

Figure 20: Web Dashboard Display of Monitored Indoor Parameters

C. Deployment of Web Application *(Shuaib)*

The deployment of the web application developed for the proposed Indoor Environmental System involves the deployment of the Server (nodejs) component and the Angular Web UI Frontend (client) component. The Nodejs component is deployed using AWS Elastic Beanstalk service. The Angular Web UI Frontend component is deployed on the AWS S3 bucket. The S3 bucket is made accessible by AWS CloudFront service. The procedures for deploying the Nodejs Server component and the Angular Web UI Frontend component are detailed in Appendix 9 of the report.

5.5 Mobile App *(Aymen)*

5.5.1 Selection Process

To facilitate easy online access to the monitored indoor environmental data, a mobile app will be made available to the end-user (customer). Two tools namely Unity Engine and Android Studio are currently available for app development. Unity Engine is a cross-platform app development engine that utilizes C# or JavaScript to program functions and assets. It allows the developer to run the project inside the editor, change the properties of project assets directly and in the process see the results immediately. Further, multiple platforms such as Windows, Mac, Linux, Android, and IOS are supported by the Unity Engine. In contrast, Android Studio supports the development of only Android apps by utilizing Java and XML to program functions and assets. In the proposed work, the Unity Engine app development tool and Visual Studio (external code editor) are utilized for the development of the mobile app. The code written for app development will be listed in Appendix, but the function of each code file will be discussed in this report.

5.5.2 Component Design

The mobile application will consist of three main pages, the first page will be the log in page where the user can log in to the application or retrieved a new password if he/she forgot his/her password. The second page is the graph page where the user will be forwarded automatically to it when his/her credentials are verified. The page will show six dynamic graphs of the user data that updates every two second. The third page will be the change password page where the user can change his/her password using the email address and the current password.

5.5.3 Used Tools

The tools utilized in the mobile app development are:

- Unity Engine
- Virtual studio
- C# programing language

5.5.4 Demo of the Component

A. Log in Page

In the log in page if the user enters wrong credentials and press log in the application will prompt him/her that the credentials are not correct, and if the user write a email that is not assigned to any user and press forget password the application will prompt him/her that there is no

user with the following email and if the user entered a valid email and press forget password a new password will be sent to that email and the application will prompt the user that the new password has been sent to the email and if the user enters valid credentials and press log in the application will forward it to the graphs page.

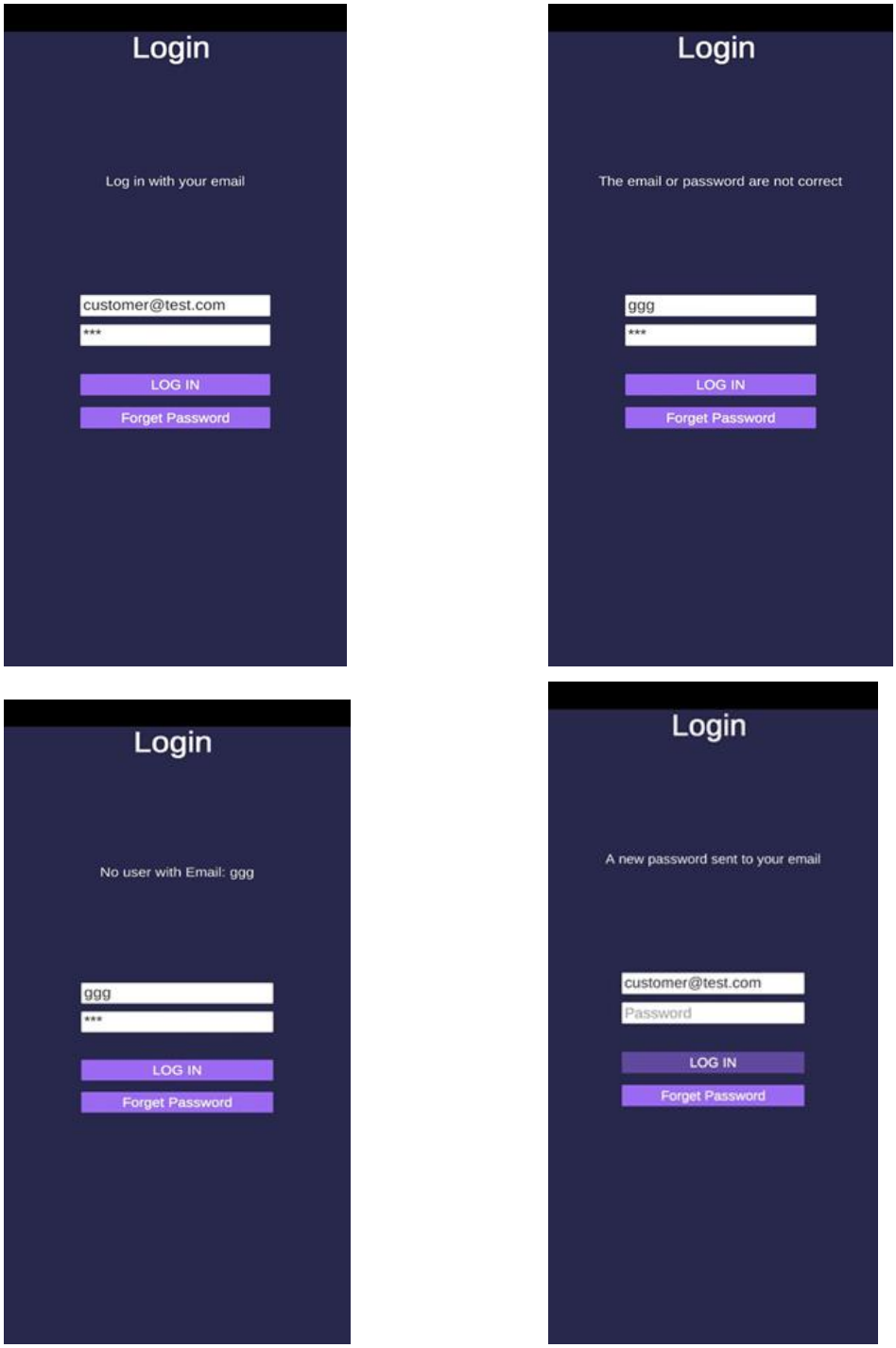


Figure 21: Mobile App Login Pages

C. The Change Password Page

In the change password page, the user can change his/her password using his/her email and current password. If the user enters wrong credentials, the application will prompt him/her that it is not correct and if the credentials are correct it will prompt him/her that the password has been changed. Also, the user can press back to go back to the graphs page.

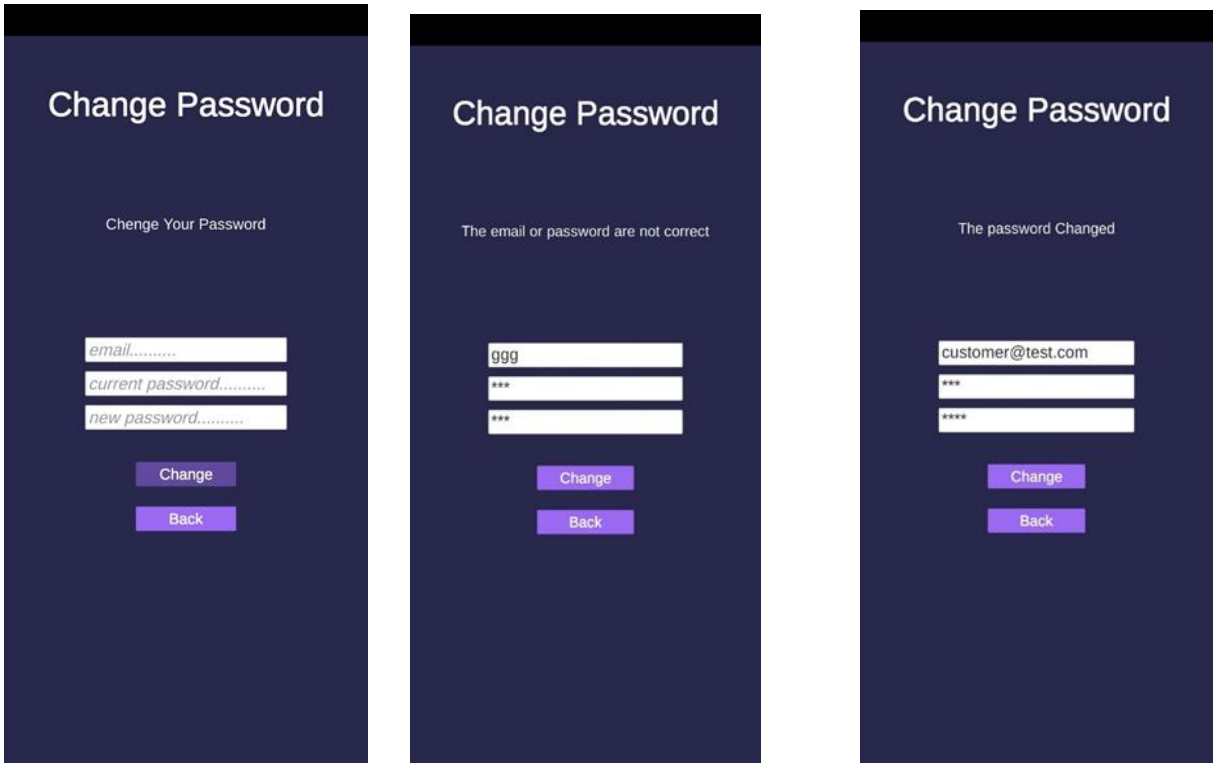


Figure 22: Mobile App Change Password Screens

C. The Graphs Pages

In the graphs page the user can monitor his/her sensors reading in a graph format. The graphs include indoor temperature, indoor humidity, and gas content. The data on the graph are taken from the database every two second, but in the database the data update every day. To demonstrate the functionality of the application the retrieved data are added to a random value to show a variety in the data because in a real situation the value will change by a small factor every time. The user can press on change password button to be forwarded to the change password page where he/she can change his/her password using the email and current password. The static page is to show the real data from the database without adding any random value to it. It contains only two graphs because the data base only provides two reading the temperature and the humidity. The

graphs look static because the values in the database are not changing, the application check is for anew readings every two seconds.

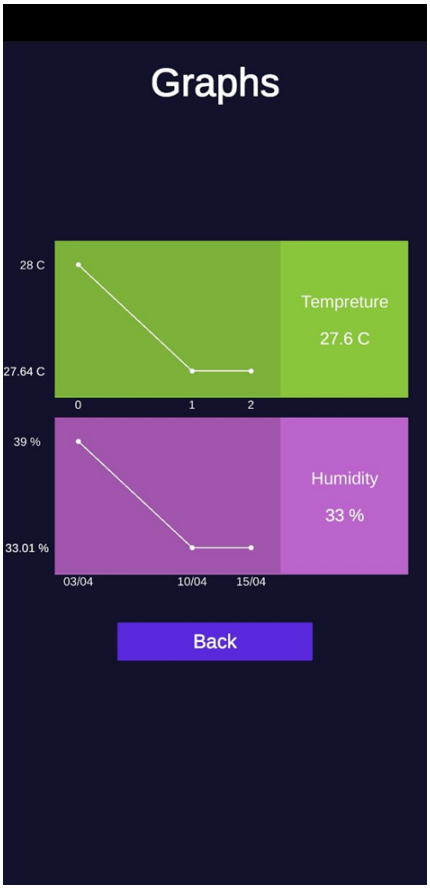


Figure 23: Indoor Temperature and Humidity Graph

D. Used Code

Each code has different configuration in unity engine this why the objects have different names in each file and different tag.

a) UserVerification: This code file verifies the user credentials and encrypt it and save it in a user file to be used in another functions.

b) UserDataTemperature: This code file decrypt the user credentials and used it to retrieve his/her temperature data from the database and make in a visual graph format.

c) UserDataHumidity: This code file decrypt the user credentials and used it to retrieve his/her humidity data from the database and make in a visual graph format.

d) **UserDataAirQuality:** This code file decrypt the user credentials and used it to retrieve his/her air quality data from the database and make in a visual graph format.

e) **NewUserDataHumidity:** This code file decrypt the user credentials and used it to retrieve his/her humidity data from the database and make in a visual graph format for the static graphs.

f) **NewUserDataTemperature:** This code file decrypt the user credentials and used it to retrieve his/her temperature data from the database and make in a visual graph format for the static graphs.

g) **ChangePasswordScript:** This code file decrypt the user data and take his/her input email and password and new password and verify them then it changes his/her password and encrypt it and save in the user file.

h) **ScenseLoader:** This code file used to load and unload different scenes in the mobile application.

i) **ForgetPass:** This code file take the input email from the user and send it to the database to get the new password then if the email is valid it will email the new password the user email and prompt it that it has been sent to his/her email.

6. SYSTEM INTEGRATION AND TESTING

The system integration and testing phase of the project is still in progress. The preliminary results obtained till date are presented in the following sections:

6.1 The Prototype and Testing Results

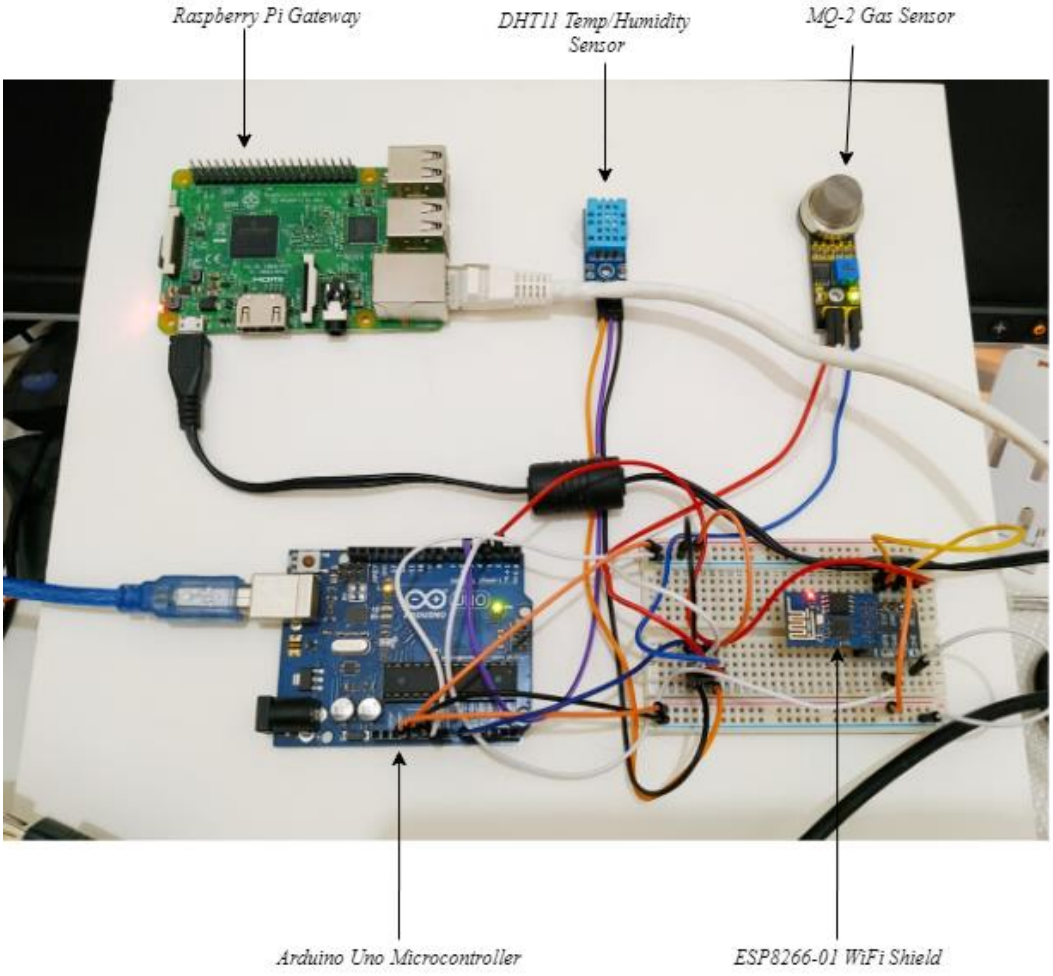


Figure 24: Prototype picture of the sensor node and gateway

The picture of the system prototype is shown in Fig. 24.

The overall testing process utilized in the design and development of the IoT-Based Indoor Environmental Monitoring System comprises three stages and is depicted in Fig. 25.

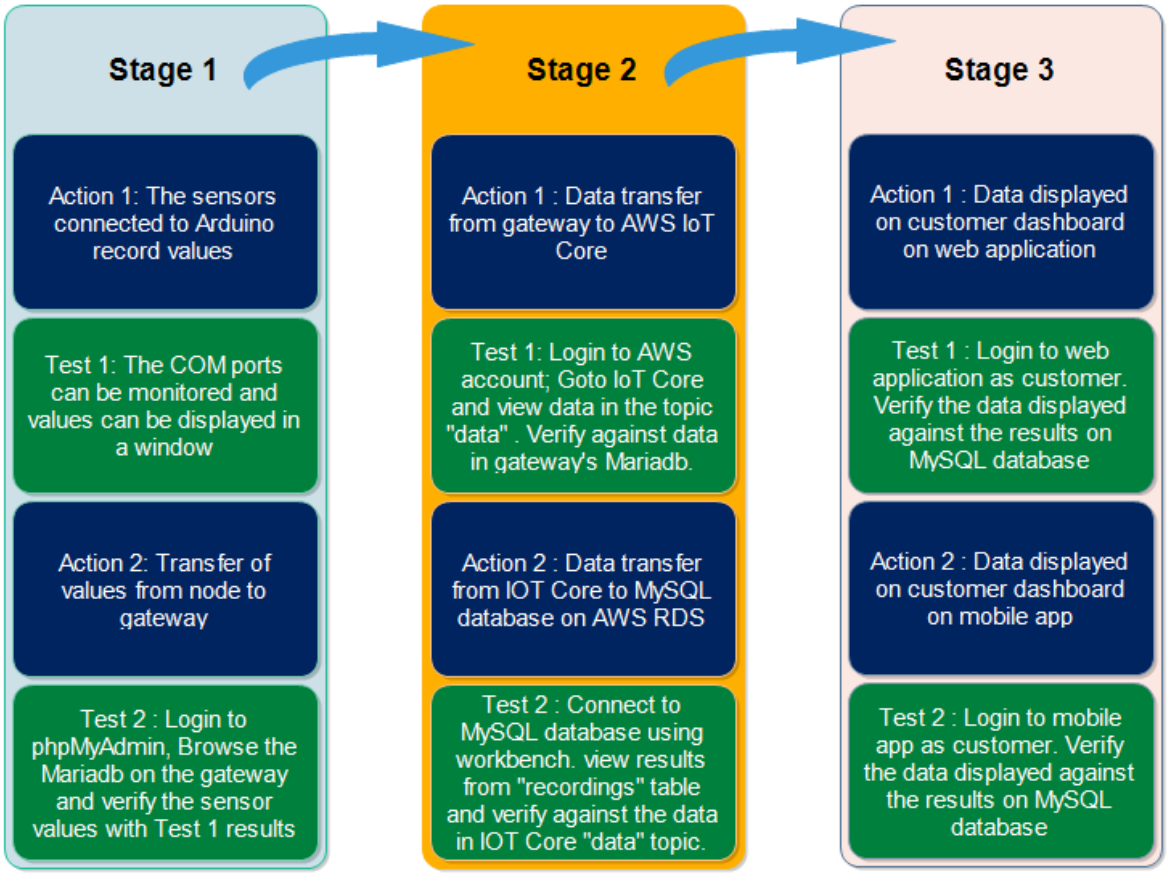


Figure 25: Stages in the Testing Process

6.1.1 Stage 1, Test 1:

The DHT-11 temperature-humidity and MQ-2 gas sensors connected to the Arduino Uno microcontroller record values and these values can be monitored on COM ports. The sensor data values are shown in Fig. 26.

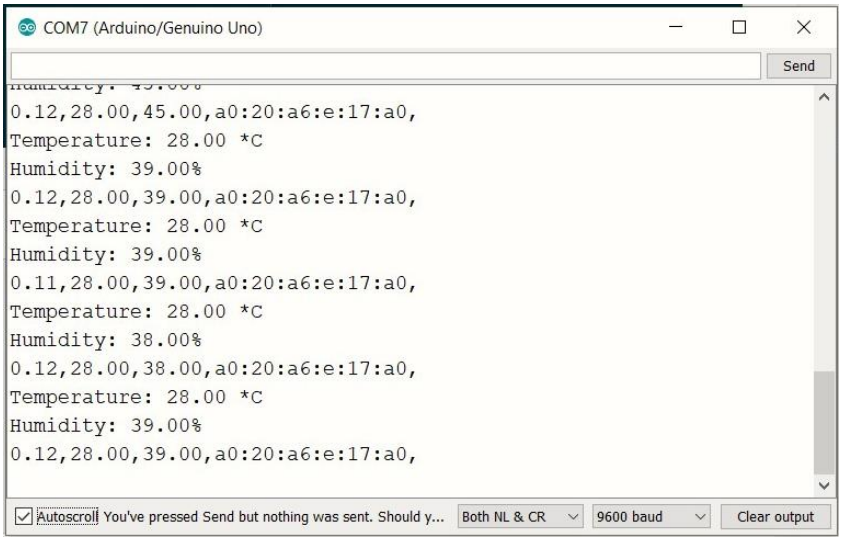


Figure 26: Stage 1, Test 1 Results

6.1.2 Stage 1, Test 2

Sensor data transferred from the sensor node to the gateway is stored in a local MariaDB database. So one can login to *phpMyAdmin*, connect to the database and verify if the same sensor data is being stored in the local database. As shown in Fig. 27, the same temperature (28ºC) and humidity (39%) values are being recorded in the local database.

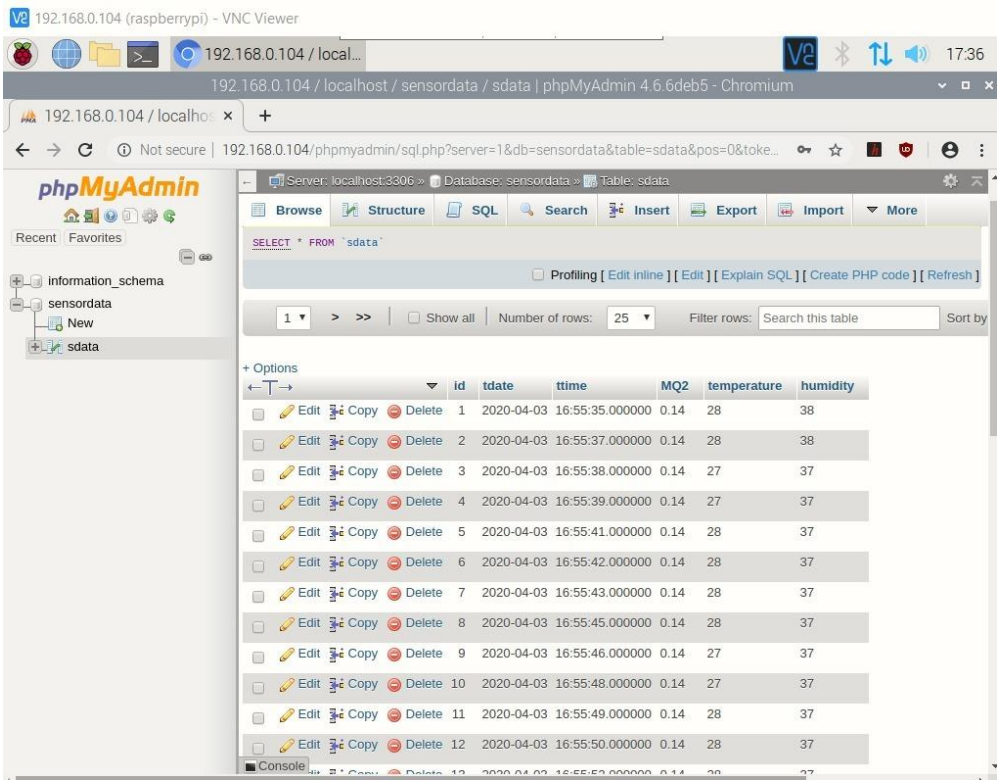


Figure 27: Stage 1, Test 2 Results

6.1.3 Stage 2, Test 1

In this stage, data is transferred from gateway to IoT core on AWS cloud using MQTT protocol. On IoT Core, there is a Topic by name “data”. The data that is received by IoT Core can be seen by subscribing to the topic and selecting data from the topic. As shown in Fig. 28, the same temperature (28ºC), humidity (39%) values are being transferred from the gateway to the IoT core.

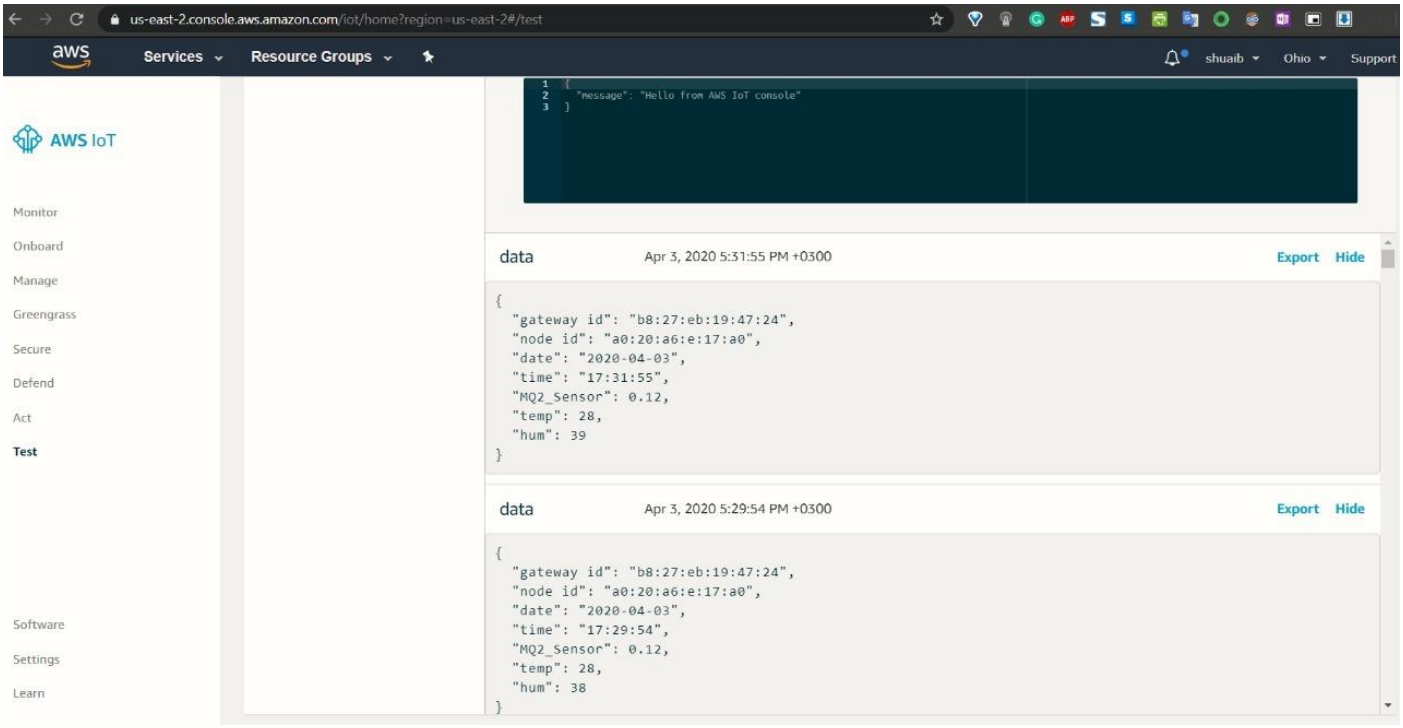


Figure 28: Stage 2, Test 1 Results

6.1.4 Stage 2, Test 2

At this point, data is fetched from IoT Core and stored in MySQL. Data on MySQL may be viewed using MySQL Workbench to connect to the database and selecting data from *recordings* table. Fig. 29 displays the Stage 2, Test 2 results.

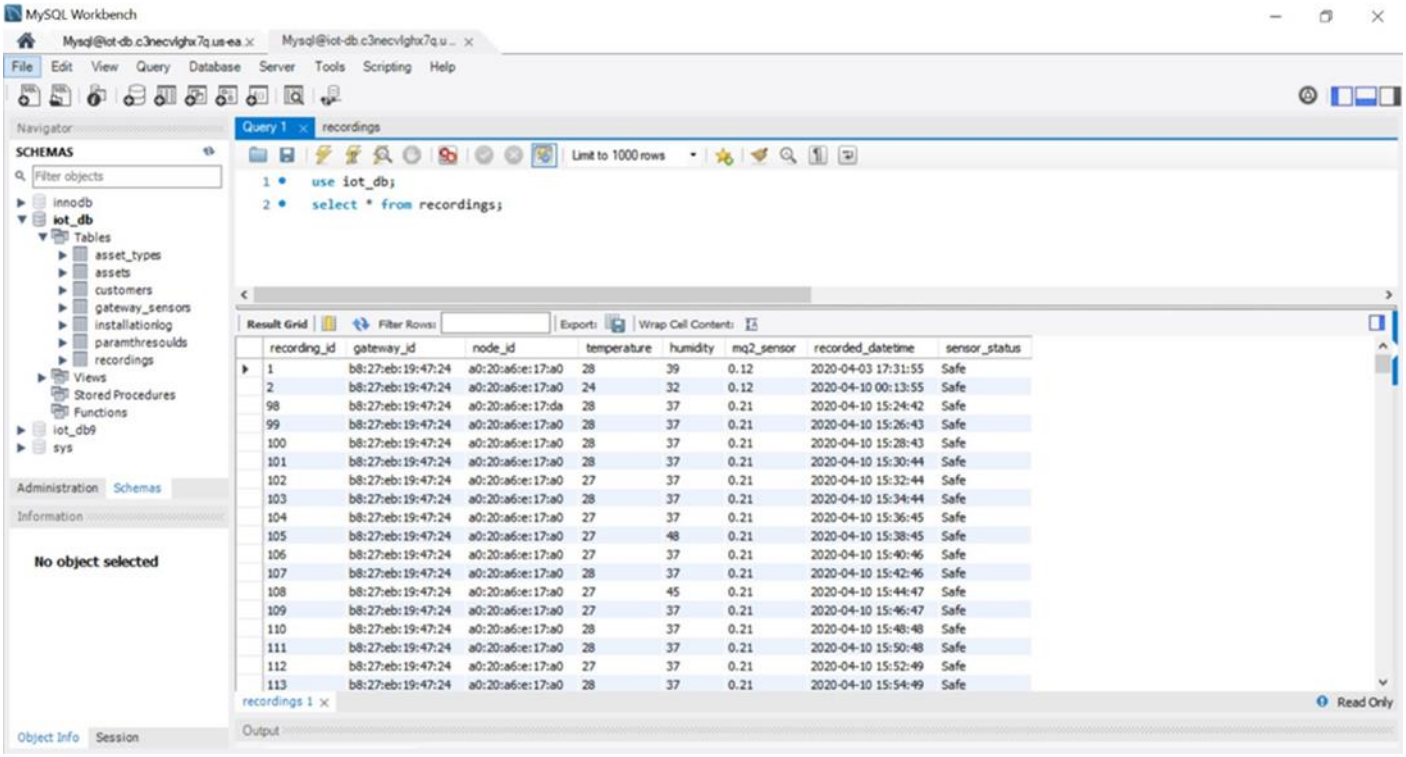


Figure 29: Stage 2, Test 2 Results: Data being stored in recordings Table

6.1.5 Stage 3, Test 1 (Web dashboard)

Figure 30: Web Dashboard Results

The customer logs in to the web application and views his web dashboard results. The results displayed can be checked against the data stored in the AWS's MySQL database.

6.1.6 Stage 3, Test 2 (Mobile dashboard)

The dashboard results displayed by the mobile app can be checked against the sensor data stored in the MySQL database.

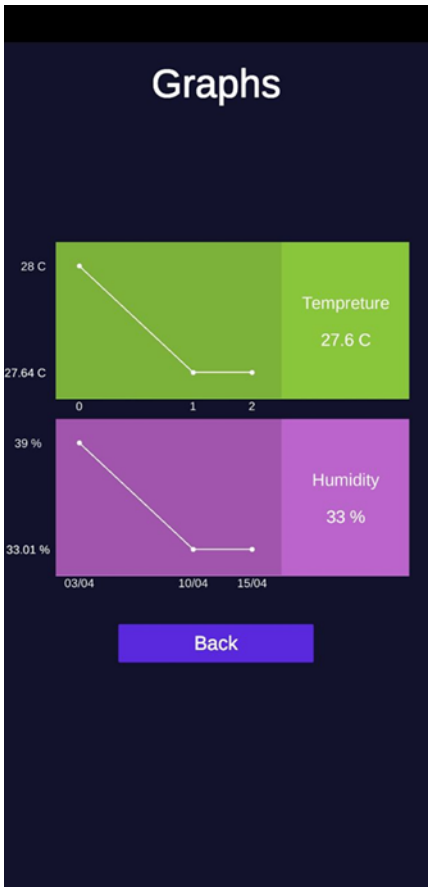


Figure 31: Mobile app displaying monitored indoor environmental data

6.2 Problems Faced and Solution

The prototype was designed using low-cost commercially available sensors. During the course of the project work, the MQ-2 sensor's smoke, LPG and CO gas concentration values in ppm started to fluctuate wildly indicating either sensor calibration issues or temporary sensor malfunction. An attempt was made to obtain steady data values by introducing smoothing code (based on moving averages) in the Arduino C program. This proved to be unsuccessful and it was decided to use the consistent and steady However MQ-2 sensor's actual output voltages as a means to detect the presence of harmful gases in the indoor environment. For a typical clean indoor environment, the MQ-2 sensor's output values were found to be in in the range 0.12 – 0.2 volts. When a LPG gas lighter flame was brought near the sensor, the MQ-2 sensor output consistently exceeded 2V indicating the trace presence of LPG and combustion products such as smoke and carbon monoxide gas in the indoor air. Since a replacement MQ-2 gas sensor could not be arranged in view of the COVID-19 lockdown restrictions, it was decided that, instead of using the ppm values, the MQ-2 sensor's volt output would be utilized as a measure of the presence of smoke etc., in the indoor air. A value of 1V was therefore set as the alert limit for the MQ-2 gas sensor for warning the customer in case of smoke/LPG/CO gas contamination. Later testing of the MQ-2 sensor proved that the remedial action that was adopted was adequate, if not optimal.

7. CONCLUSIONS

A scalable IoT-based indoor environmental system was designed to measure indoor temperature, humidity and gas content autonomously and continuously without operator intervention. The designed prototype enables customers to access their measured indoor environmental parameters through web and mobile dashboards. The proposed system provides significant advantages to the end-users in terms of creating a more conducive work environment, increased productivity, and improved health conditions. Further, its implementation can be easily extended to other living/work spaces to create even greater benefits to society at large.

Accomplishing the project tasks during the unforeseen COVID-19 outbreak became a challenge which was overcome through the valuable online guidance and encouragement received from the design project supervisor Dr. Elrabba. The supervisor's timely feedback helped to deploy the right design approach (*using AWS for building the cloud data model instead of the "ThingsBoard" platform*) in our project. Utilizing AWS in the project design was initially difficult, but referring to numerous AWS-related online tutorials proved helpful. An important lesson that we learnt from dealing with the MQ-2 gas sensor issue is that designers must plan for contingencies such as component failures and ensure that enough spare parts are available to deal with hardware component failures during the system development process.

To conclude, the senior design project provided an opportunity for accelerated learning and highlighted the iterative nature of the computer engineering design process. It has also taught us the value of adopting new development tools/alternative approaches etc., and working under constraints to fulfill the project objectives.

9. REFERENCES

[1]. P. MacNaughton, et. al., "Economic, Environmental and Health Implications of Enhanced Ventilation in Office Buildings", *Int J Environ Res Public Health*, Nov. 2015, Vol. 12, Iss. 11, pp: 14709-14722.

[2]. G. Boulanger et. al. "Socio-economic costs of indoor air pollution: A tentative estimation for some pollutants of health interest in France", *Environment International*, 2017, Vol. 104, pp. 14-24.

[3] Alhmiedat and G. Samara. "A Low Cost Zigbee Sensor Network Architecture for Indoor Air Quality Monitoring". *International Journal of Computer Science and Information Security*. 2017, No.15, pp.140–4.

[4] P. Arroyo P, J.L.Herrero, J.I. Suarez, and J.Lozano. "Wireless Sensor Network Combined with Cloud Computing for Air Quality Monitoring". *Sensors* Basel. 2019; 19:691.

[5] G. Parmar, S. Lakhani, M. K. Chattopadhyay. "An IoT Based Low Cost Air Pollution Monitoring System", *Proceeding International conference on Recent Innovations is Signal Processing and Embedded Systems* (RISE -2017), 27-29 October 2017, pp. 524-528.

[6] B. Sivasankari, C.A. Prabha, S. Dharini and R. Haripriya "IoT Based Indoor Air Pollution Monitoring Using Raspberry Pi". *International Journal of Innovation Eng. Tech*. 2017, No.9, pp.16–21.

[7]. J. Saini, M. Dutta, and G. Marques. "A Comprehensive Review on Indoor Air Quality Monitoring Systems for Enhanced Public Health", Sustainable *Environment Research*, 2020, Vol. 30, No. 6, pp. 1-12.

[9] "Ventilation for Acceptable Indoor Air Quality". American Society of Heating, Refrigerating, and Air-Conditioning (ASHRAE) *Standard 62.1-2010*, 2010, Tables B1-B3, pp. 27-36.

[10] "U.S. Occupational Safety and Health Administration (OSHA) Policy on Indoor Air Quality: Office Temperature/Humidity and Environmental Tobacco Smoke", Washington DC, 2003"

[11] A. Aqeel. "Introduction to Arduino Uno." https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-uno.html

[12] "DHT 11 Humidity & Temperature Sensor Datasheet." https://datasheetspdf.com/pdf-file/785590/D-Robotics/DHT11/1

[13] "MQ-2 Sensor Datasheet." https://www.mouser.com/datasheet/2/321/605-00008-MQ-2-Datasheet-370464.pdf

[14] "ESP8266 WiFi Module." https://components101.com/wireless/esp8266-pinout-configuration-features-datasheet.

[15] "Raspberry Pi3". https://components101.com/microcontrollers/raspberry-pi-3-pinout-features-datasheet

[16] "Create and Connect MySQL Database on RDS". https://aws.amazon.com/getting-started/hands-on/create-mysql-db/

[17] “MySQL Workbench 8.0.19 for Windows”. https://dev.mysql.com/downloads/workbench/.

[18] “Deploy Nodejs Application Using Beanstalk” https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_nodejs.html

[19] V. Borisov. “Deploy Angular application to AWS S3 and CloudFront.” 19 March 2019. https://firstclassjs.com/deploy-angular-application-to-aws-s3-and-CloudFront/

[20] K. Srilomsak. “Deploy an Angular with S3 and CloudFront.” 30 April 2018. https://medium.com/@peatiscoding/here-is-how-easy-it-is-to-deploy-an-angular-spa-single-page-app-as-a-static-website-using-s3-and-6aa446db38ef

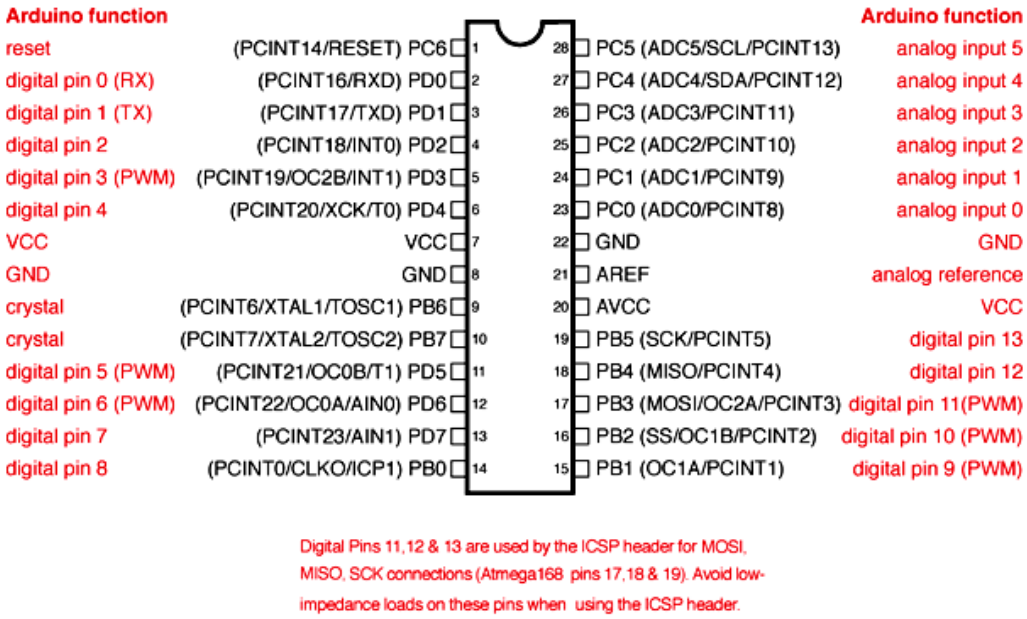
[21] “How do I use CloudFront to Serve a Static Website Hosted on Amazon S3?.” 30 July 2019. https://aws.amazon.com/premiumsupport/knowledge-center/CloudFront-serve-static-website/

[22] Sumit. “Creating AWS CloudFront Distribution with S3 Origin.” 14 January 2019. https://medium.com/tensult/creating-aws-CloudFront-distribution-with-s3-origin-ee47b8122727

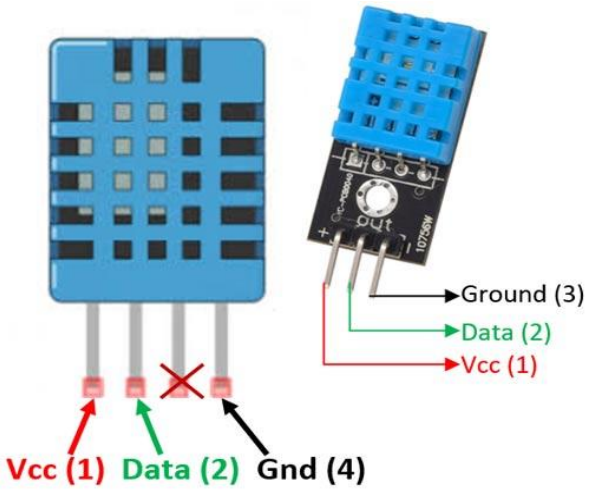
10. APPENDICES

Appendix-1: Pin Diagrams of Hardware Components

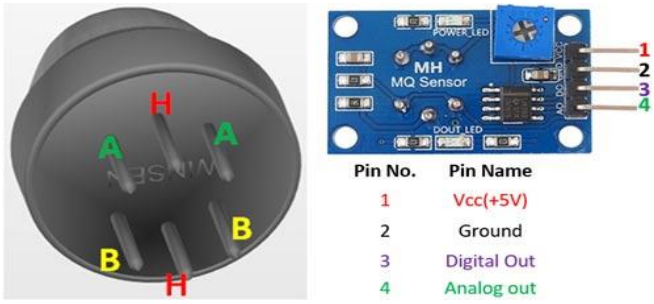
A. Arduino Uno Microcontroller



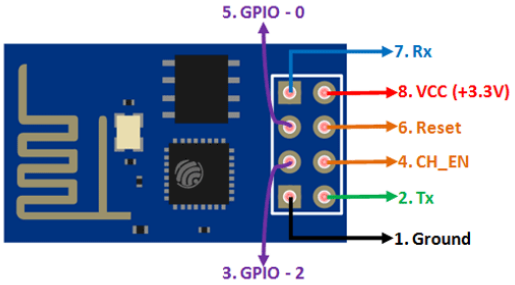
B. DHT11 Temperature/Humidity Sensor



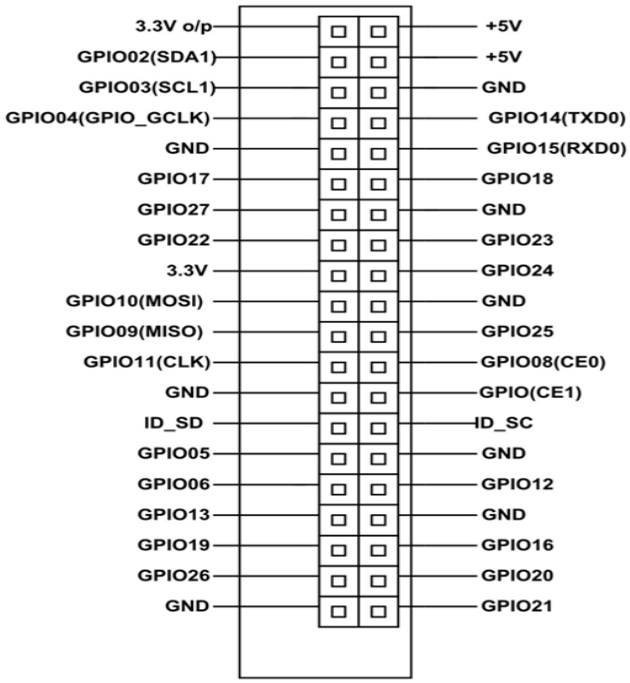
C. MQ2 Gas Sensor



C. ESP8266 WiFi Shield



D. Raspberry Pi



Appendix-2: Configuration of the Arduino Uno Microcontroller

The step-by-step procedure for configuring the Arduino Uno microcontroller used in the system design is described in the following sections:

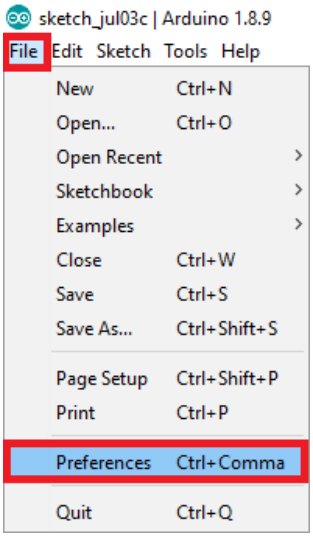
A. Installation of the Arduino IDE.

Install Arduino IDE from the link https://www.arduino.cc/en/main/software

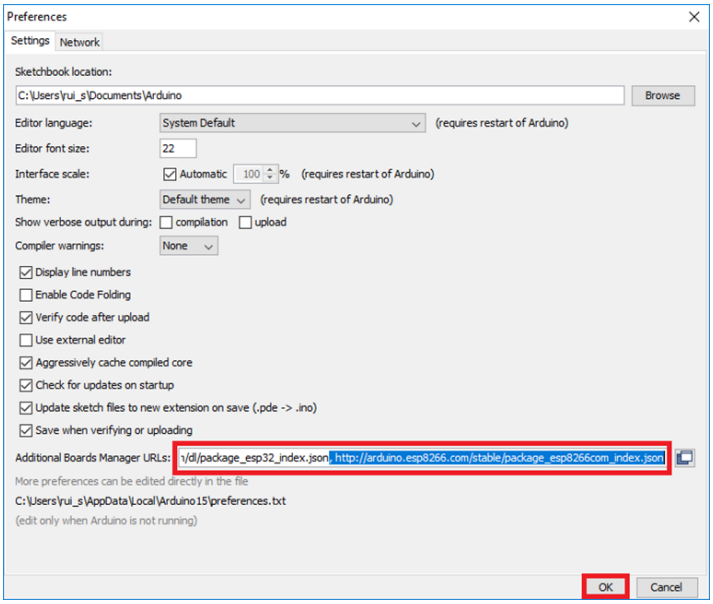
B. Installation of the ESP8266 Board in Arduino IDE.

Install the ESP8266 Board in Arduino IDE by following the steps below:

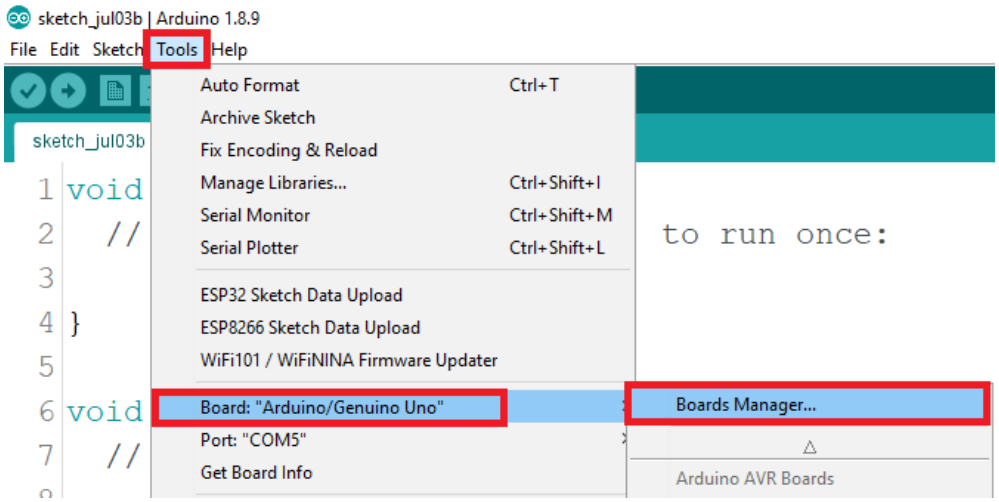
1. In the Arduino IDE, go to **File**> **Preferences**



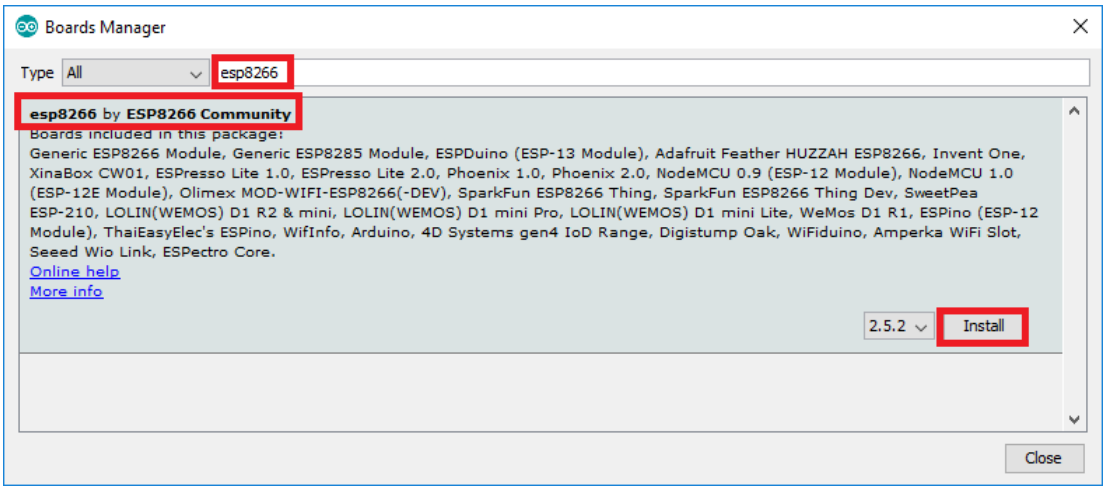
2. Enter **http://arduino.esp8266.com/stable/package_esp8266com_index.json** into the "Additional Boards Manager URLs" field as shown in the figure below. Then, click the "OK" button:



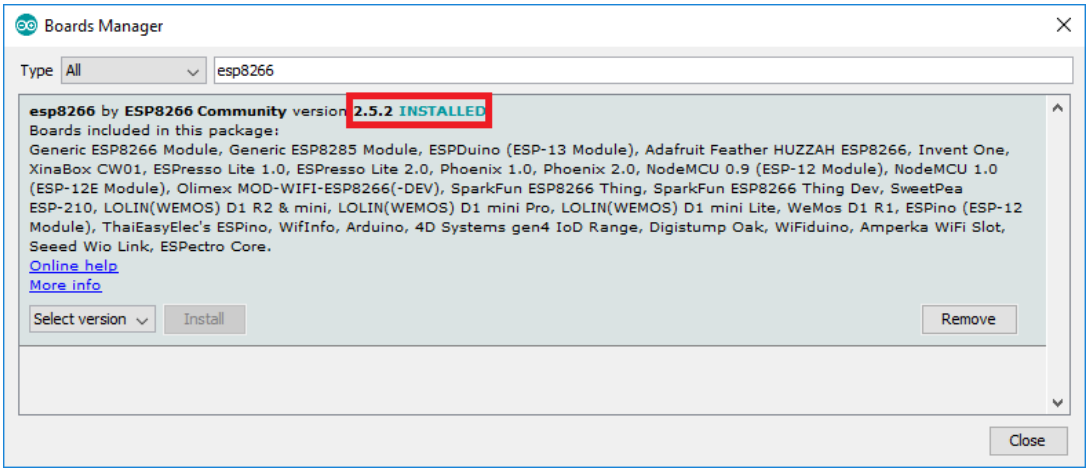
3. Open the Boards Manager. Go to **Tools** > **Board** > **Boards Manager…**



4. Search for **ESP8266** and press install button for the "**ESP8266 by ESP8266 Community**" :

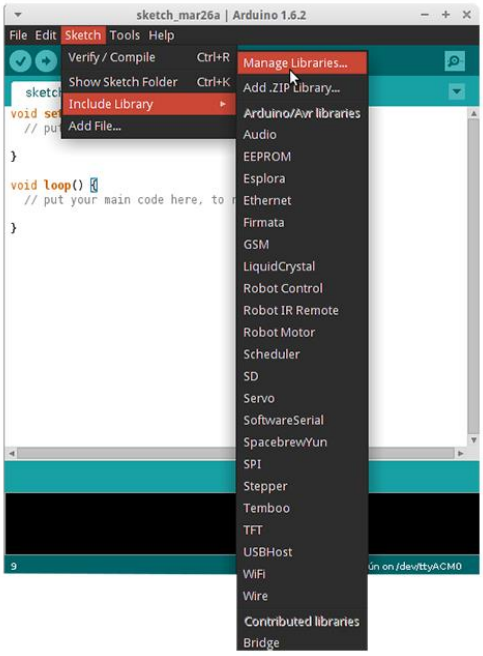


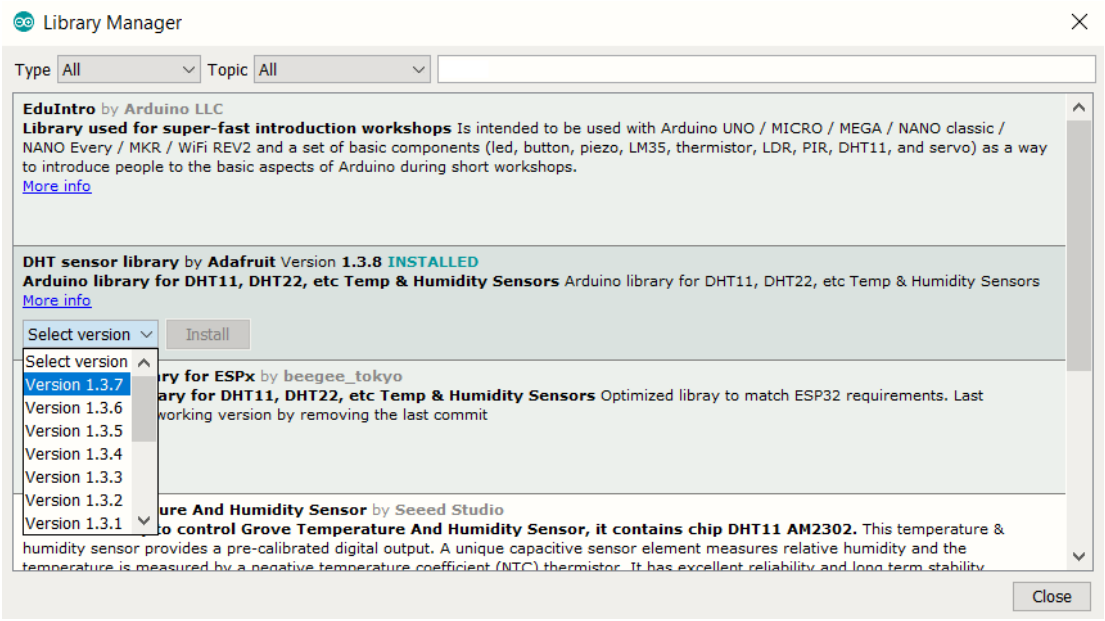
5. Installation is completed after a few seconds.



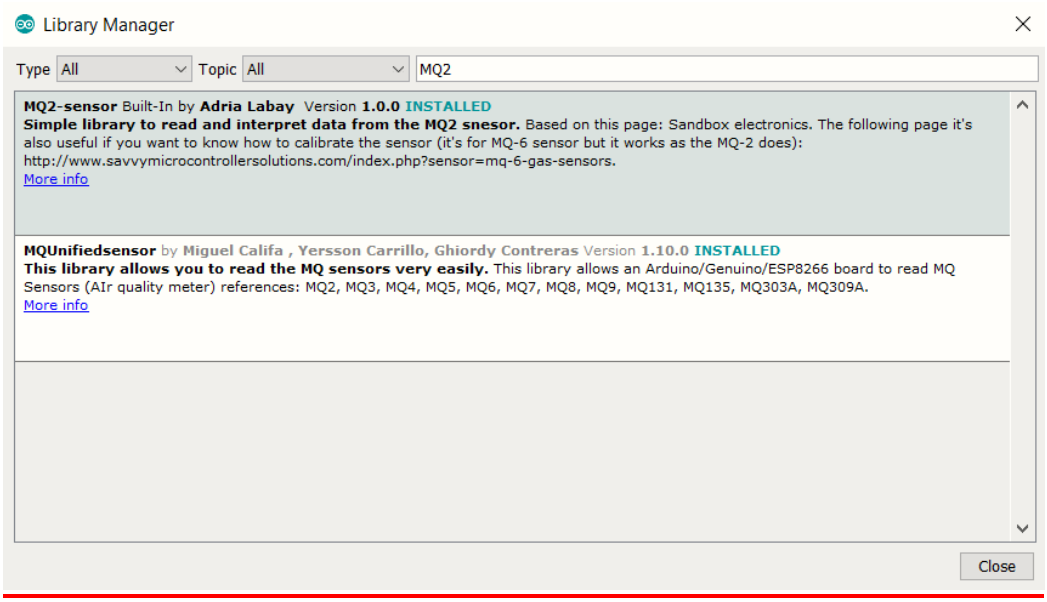
C. Installation of the libraries in IDE.

To install the sensor libraries into the Arduino IDE, the Library Manager (available from IDE version 1.6.2) can be utilized. Open the IDE and click on the "Sketch" menu and then *Include Library* > *Manage Libraries*. Then the Library Manager will open and a list of libraries that are already installed or ready for installation can be found. Select the DHT11 sensor library for installation. Once the installation process of the library by the IDE is finished, an *Installed* tag should appear next to the DHT11 library. The screenshots shown below display the installation of the DHT11 library.





The MQ-2 library is installed in a similar manner.



Appendix-3: Configuration of the Raspberry Pi Gateway

The configuration of the Raspberry Pi 3 as an IoT gateway involves the following main tasks:

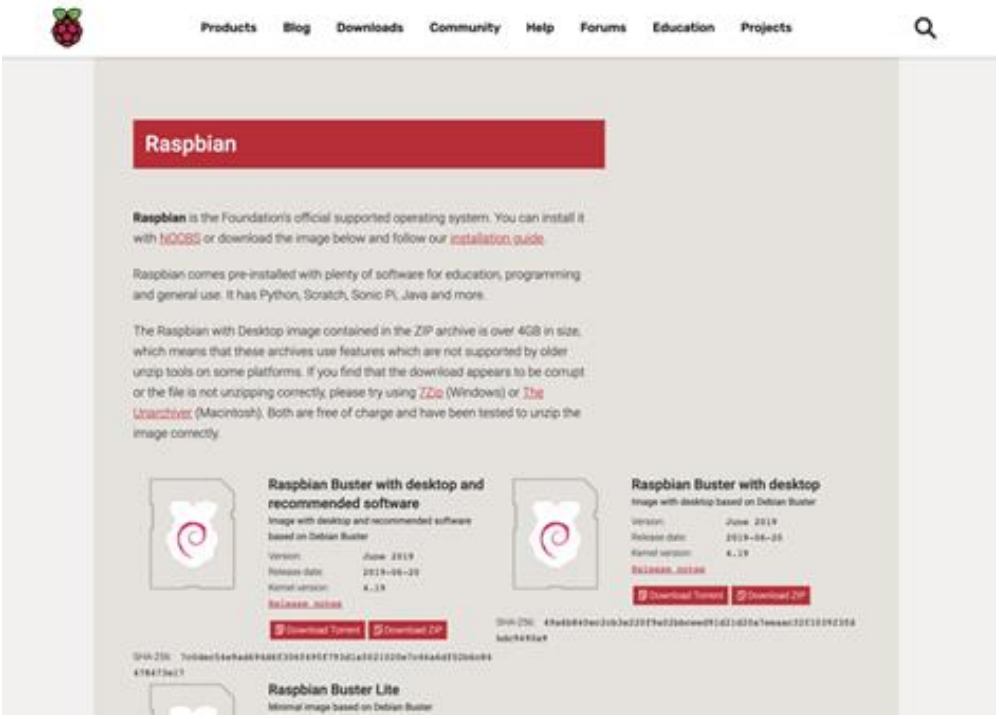
A. Installation of the Raspbian Linux operating system.
B. Installation of the Mosquitto Broker.
C. Installation of the Python library for MQTT.
D. Setting up of local *Maria dB* database.
E. Setting up *phpMyAdmin*.

The step-by-step procedure for configuring the Raspberry Pi gateway above is described in the following sections:

A. Installation of the Raspbian Linux operating system

[1]. Download the Raspbian Buster from the link (https://www.raspberrypi.org/downloads/raspbian/)

The option chosen for the proposed prototype is the Raspbian Buster with Desktop in order to facilitate access to the Raspbian GUI.



[2]. Insert the Micro SD card into the computer.

[3]. Flash the Raspbian Buster disk image onto the Micro SD card using the Etcher tool.

[4]. Boot the Raspberry Pi 3 after inserting the newly flashed Micro SD card into it.

B. Installation of the Mosquitto Broker

In order to receive sensor data from the Arduino Uno microcontroller, there is a need to setup a local MQTT broker. The MQTT broker will provide connectivity between the Arduino Uno microcontroller and Raspberry pi gateway and facilitate the sending and receiving of sensor data. In the prototype system, the local MQTT broker known as 'Mosquitto Broker' was installed, using the following commands:

```
sudo apt-get update
sudo apt install -y mosquitto mosquitto-clients *
```

After installation, the Mosquitto broker is first enabled and tested it by subscribing to testTopic using the following commands:

```
mosquitto -d
sudo systemctl enable mosquitto
sudo systemctl status mosquitto
sudo systemctl status mosquitto
mosquitto_sub -d -t testTopic
```

C. Installation of the Python library for MQTT

After setting up the Mosquitto broker, the installation of the python library for MQTT is required. This library will provide functions to publish or subscribe data. Writing the following command in shell will install the python library:

```
sudo pip3 install paho-mqtt
```

D. Setting up of Local Maria dB database

In order to save data in local database, the setting up of the SQL MariaDB is required. The setting involves the installation of the database, creation of the database and creation of the tables. These can be achieved by executing the following commands:

```
sudo apt install mariadb-server*
sudo mysql_secure_installation
sudo mysql -u root -p #This command will help us to log into newly created database
 mysql -u pi -p #this command will help us to log into our table which we created using last command.
```

E. Setting up phpMyAdmin

After setting up the local Maria dB database, the phpMyAdmin is installed and linked to the database using the following commands :

```
sudo apt install phpmyadmin
sudo nano /etc/apache2/apache2.conf
sudo service apache2 restart
sudo ln -s /usr/share/phpmyadmin /var/www/html
```

Appendix-4: C-Code for the Sensor Node: "sketch_mar22b_FinalCode_Final_Code_030420_.ino"

```
#include <WiFiEspClient.h>
#include <WiFiEsp.h>
#include <WiFiEspUdp.h>
#include "SoftwareSerial.h"
#include <PubSubClient.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <MQ2.h>

int Analog_Input = A0;

String id = "";
byte mac[6];


#define DHTPIN           5        // Pin which is connected to the DHT sensor.
#define WIFI_AP "Saad"
#define WIFI_PASSWORD "W0rDPa$$"
#define mqtt_server "192.168.0.222"

WiFiEspClient espClient;
PubSubClient client(espClient);
// Uncomment the type of sensor in use:
#define DHTTYPE            DHT11     // DHT 11

DHT_Unified dht(DHTPIN, DHTTYPE);

uint32_t delayMS;
SoftwareSerial soft(2, 3); // RX, TX

int status = WL_IDLE_STATUS;
unsigned long lastSend;

void setup() {
  // initialize serial for debugging
  Serial.begin(9600);
  pinMode(10,OUTPUT);
  dht.begin();
  InitWiFi();
  lastSend = 0;
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  dht.begin();

  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
```

```
  dht.humidity().getSensor(&sensor);

  delayMS = sensor.min_delay / 1000;

}
bool checkBound(float newValue, float prevValue, float maxDiff) {
  return !isnan(newValue) &&
         (newValue < prevValue - maxDiff || newValue > prevValue + maxDiff);
}

void callback(char* topic, byte* payload, unsigned int length) {
  String data;
  data = "";
  for (int i = 0; i < length; i++) {
    data += (char)payload[i];
  }
}

String data = "";

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
      float sensor_volt;
    float sensorValue;

    sensorValue = analogRead(A0);
    sensor_volt = sensorValue/1024*5.0;
    if(sensor_volt > 1){
      tone(10,2000);
    }
    else{
      noTone(10);

    }

  if ( millis() - lastSend > 10000) { // Update and send only after 1 seconds
    getAndSendTemperatureAndHumidityData();
    lastSend = millis();
  }

}

void getAndSendTemperatureAndHumidityData() {
  data = "";
  WiFi.macAddress(mac);
```

```
    float sensor_volt;
    float sensorValue;

    sensorValue = analogRead(A0);
    sensor_volt = sensorValue/1024*5.0;

    data+=String(sensor_volt)+",";

  sensors_event_t event;
  dht.temperature().getEvent(&event);
  if (isnan(event.temperature)) {
    Serial.println("Error reading temperature!");
  }
  else {
    Serial.print("Temperature: ");
    Serial.print(event.temperature);
    Serial.println(" *C");
    data += String(event.temperature) + ",";
  }
  // Get humidity event and print its value.
  dht.humidity().getEvent(&event);
  if (isnan(event.relative_humidity)) {
    Serial.println("Error reading humidity!");
  }
  else {
    Serial.print("Humidity: ");
    Serial.print(event.relative_humidity);
    Serial.println("%");
    data += String(event.relative_humidity) + ",";
  }

  id = String(mac[5], HEX) + ":" + String(mac[4], HEX) + ":" + String(mac[3], HEX) + ":"
+ String(mac[2], HEX) + ":" + String(mac[1], HEX) + ":" + String(mac[0], HEX);

  data += id + ",";

  char __data[100];
  data.toCharArray(__data, 100);
  client.publish("data", __data);
  Serial.println(data);
}

void InitWiFi()
{
  // initialize serial for ESP module
  soft.begin(9600);
  // initialize ESP module
  WiFi.init(&soft);
  // check for the presence of the shield
```

```
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue
    while (true);
  }

  Serial.println("Connecting to AP ...");
  // attempt to connect to WiFi network
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(WIFI_AP);
    // Connect to WPA/WPA2 network
    status = WiFi.begin(WIFI_AP, WIFI_PASSWORD);
    delay(500);
  }

  Serial.println("Connected to AP");
}

void reconnect() {
  // Loop until we're reconnected
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(mqtt_server)) {
      Serial.println("connected");
      client.subscribe("abc");
      client.subscribe("abc2");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
```

Readme: to run Arduino code for the sensor node:

1) Open the Arduino ide and select your code from the location you saved by clicking file->open

2) Once opened:

3) Click "Sketch"->Upload or Crtl+U to upload the code to the board, if you would like to make any changes to the code repeat the same process.

4) Once successfully uploaded you'll see "Done" at the bottom left corner of the IDE.

Appendix-5: Python Code for Raspberry Pi Gateway: "receive2.py"

```
import paho.mqtt.client as mqtt
import mariadb
import sys
from datetime import datetime
import json
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
MQTT_SERVER = "localhost"
MQTT_PATH = "data"
def customShadowCallback_Update(payload, responseStatus, token):
    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~~~\n\n")
    if responseStatus == "rejected":
        print("Update request " + token + " rejected!")

# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):
     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")

    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~~~\n\n")
    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")
# Init AWSIoTMQTTShadowClient
myAWSIoTMQTTClient = None
myAWSIoTMQTTClient = AWSIoTMQTTClient("RaspberryPi")
myAWSIoTMQTTClient.configureEndpoint("aulek4z76nfec-ats.iot.us-east-
2.amazonaws.com", 8883)
# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTClient.configureMQTTOperationTimeout(5) # 5 sec
myAWSIoTMQTTClient.configureCredentials("/home/pi/Desktop/rootCA.pem",
"/home/pi/Desktop/95fcb53f57-private.pem.key", "/home/pi/Desktop/95fcb53f57-
certificate.pem.crt")
```

```
# Connect to AWS IoT
myAWSIoTMQTTClient.connect()
# Create a device shadow handler, use this to update and delete shadow document
# Delete current shadow JSON doc
def getMAC(interface='eth0'):
  # Return the MAC address of the specified interface
  try:
    str = open('/sys/class/net/%s/address' %interface).read()
  except:
    str = "00:00:00:00:00:00"
  return str[0:17]
# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe(MQTT_PATH)
def savetoDB():
    global val
    now = datetime.now()
    print("here")
    date = now.strftime('%Y-%m-%d')
    #time = now.strftime('%H:%M:%S')
    time = now.strftime("%H:%M:%S")
    try:
        conn = mariadb.connect(
          user="pi",
          password="raspberry",
          host="localhost",
          database="sensordata",
          port=3306)
        cur = conn.cursor()

        try:
            cur.execute("INSERT INTO sdata(tdate, ttime, MQ2,temperature,humidity)
VALUES (?, ?, ?, ?, ?)",
            (date,time,float(val[0]),float(val[1]),float(val[2])))
            conn.commit()
        except Exception as e:
            print(e)
        print("done")
    except mariadb.Error as e:
      print(f"Error connecting to MariaDB Platform: {e}")
      sys.exit(1)


    try:
        x = {
```

```
        "gateway_id":getMAC(),
        "node_id":val[3],
        "date":date,
        "time":time,
        "MQ2_Sensor":"{:.2f}".format(float(val[0])),
        "temp":"{:.2f}".format(float(val[1])),
        "hum":"{:.2f}".format(float(val[2]))
        }
        dataToSend = json.dumps(x)
        # the result is a JSON string:
        print(dataToSend)
        myAWSIoTMQTTClient.publish("data", dataToSend, 1)
    except Exception as e:
        print(e)
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    global val
    print(msg.topic+" "+str(msg.payload))
    if(str(msg.topic)=="data"):
        val = msg.payload.decode().split(",")
        savetoDB()


    # more callbacks, etc
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(MQTT_SERVER, 1883, 60)
# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface and a
# manual interface.
while True:
    client.loop(.5)
```

Readme: how to run the python code " receive2.py"

1) Open VNC Viewer.

2) Input your Raspberry Pi IP address in the top bar hit enter.

3) Enter your user name and password to login.

4) Open your code "recieve2.py" code from the desktop.

5) Once opened in *Thonny* editor, Click Run from the top menu bar to see the raw values appear in the shell below.

6) Once tested and done Click Stop from the top menu bar and Quit.

Appendix-6: Connection of Raspberry Pi Gateway to AWS IoT Core

To connect a device to AWS IoT, one should create an IoT thing, a device certificate, and an AWS IoT policy. In the proposed system, the Raspberry Pi gateway is the "thing" which upload sensor node data to the IoT core. All devices (things) must have a device certificate to connect to and authenticate with AWS IoT. Each device certificate has one or more AWS IoT policies associated with it. These policies determine which AWS IoT resources the device (Raspberry Pi in our case) can access. Devices and other clients use an AWS IoT root CA certificate to authenticate the AWS IoT server with which they are communicating. An AWS IoT rule contains a query and one or more rule actions. The query extracts data from device messages to determine if the message data should be processed. The rule action specifies what to do if the data matches the query. The rule listens for sensor node data from the Raspberry Pi gateway. If the value is below a threshold, it sends a message to the Amazon SNS topic. Amazon SNS sends that message to those who have subscribed to the topic. The procedures involved in connecting the Raspberry Pi gateway to the AWS IoT Core and enabling it to upload sensor node are described below:

A. AWS Policy Creation

Create an AWS IoT policy that allows your Raspberry Pi to connect and send messages to AWS IoT.

1. In the AWS IoT console, if a **Get started** button appears, choose it. Otherwise, in the navigation pane, expand **Secure**, and then choose **Policies**.
2. If **You don't have any policies yet** dialog box appears, choose **Create a policy**. Otherwise, choose **Create**.
3. Enter a name for the AWS IoT policy (for example, **test**).
4. In the **Add statements** section, replace the existing policy with the following **JSON.**
   Replace <*region*> and <*account*> with your AWS Region and AWS account number.

```
{
"Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
     "Action": "iot:Connect",
    "Resource": "arn:aws:iot:<region>:<account>:client/RaspberryPi"
 },
 {
   "Effect": "Allow",
  "Action": "iot:Publish",
  "Resource": [
            "arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/update",
           "arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/delete",
           "arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/get"
        ]
   },
    {
         "Effect": "Allow",
       "Action": "iot:Receive",
         "Resource": [

"arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/update/accepted",

"arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/delete/accepted",
"arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/get/accepted",
"arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/update/rejected",

"arn:aws:iot:<region>:<account>:topic/$aws/things/RaspberryPi/shadow/delete/rejected"
      ]
 },
    {
       "Effect": "Allow",
       "Action": "iot:Subscribe",
       "Resource": [
"arn:aws:iot:<region>:<account>:topicfilter/$aws/things/RaspberryPi/shadow/update/accepted",
"arn:aws:iot:<region>:<account>:topicfilter/$aws/things/RaspberryPi/shadow/delete/accepted",
"arn:aws:iot:<region>:<account>:topicfilter/$aws/things/RaspberryPi/shadow/get/accepted",
"arn:aws:iot:<region>:<account>:topicfilter/$aws/things/RaspberryPi/shadow/update/rejected",
"arn:aws:iot:<region>:<account>:topicfilter/$aws/things/RaspberryPi/shadow/delete/rejected"
]
     },
 {
  "Effect": "Allow",
  "Action": [
   "iot:GetThingShadow",
     "iot:UpdateThingShadow",
     "iot:DeleteThingShadow"
 ],
 "Resource": "arn:aws:iot:<region>:<account>:thing/RaspberryPi"
 }
 ]
}
```

Choose **Create**.

B. Creating a Thing in the AWS IoT registry to represent Raspberry Pi Gateway

1. In the AWS IoT console, in the navigation pane, choose **Manage**, and then choose **Things**.

2. If **You don't have any things yet** dialog box is displayed, choose **Register a thing**. Otherwise, choose **Create**.

3. On the **Creating AWS IoT things** page, choose **Create a single thing**.

4. On the **Add your device to the device registry** page, enter a name for your IoT thing (for example, **Raspberry Pi**), and then choose **Next**. You can't change the name of a thing after you create it. To change a thing's name, you must create a new thing, give it the new name, and then delete the old thing.

5. On the **Add a certificate for your thing** page, choose **Create certificate**.

6. Choose the **Download** links to download the certificate, private key, and root CA certificate.

   Important

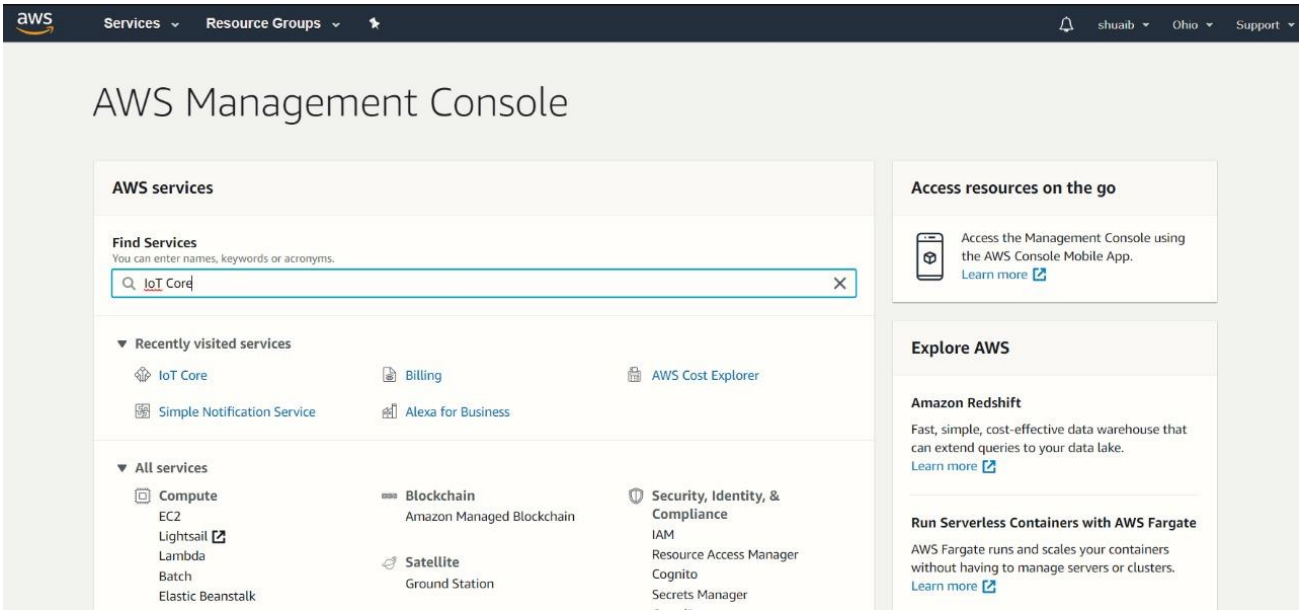
   This is the only time you can download your certificate and private key.

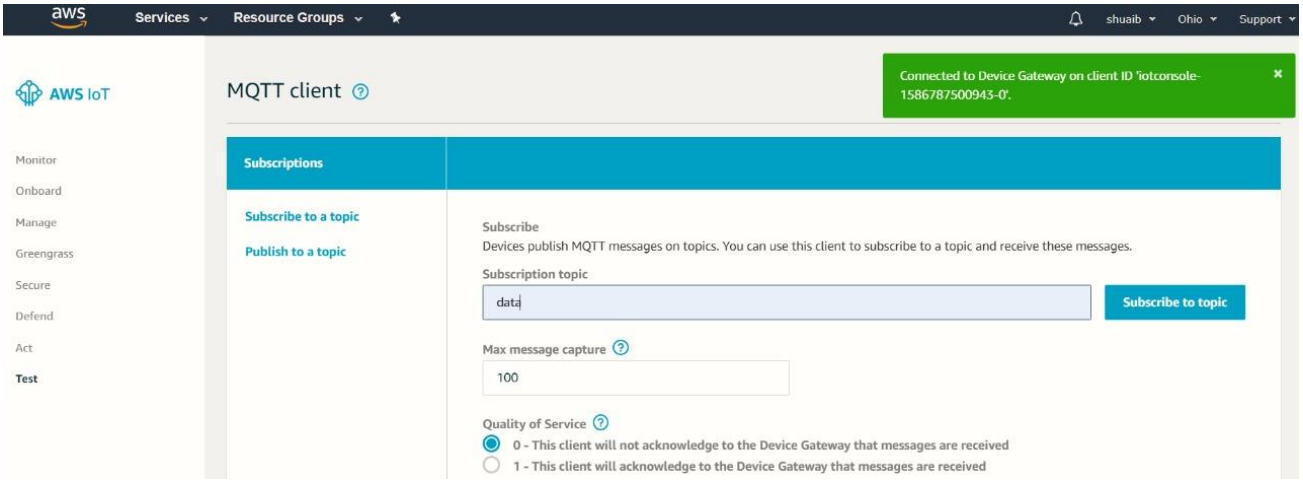
7. Choose **Activate**.

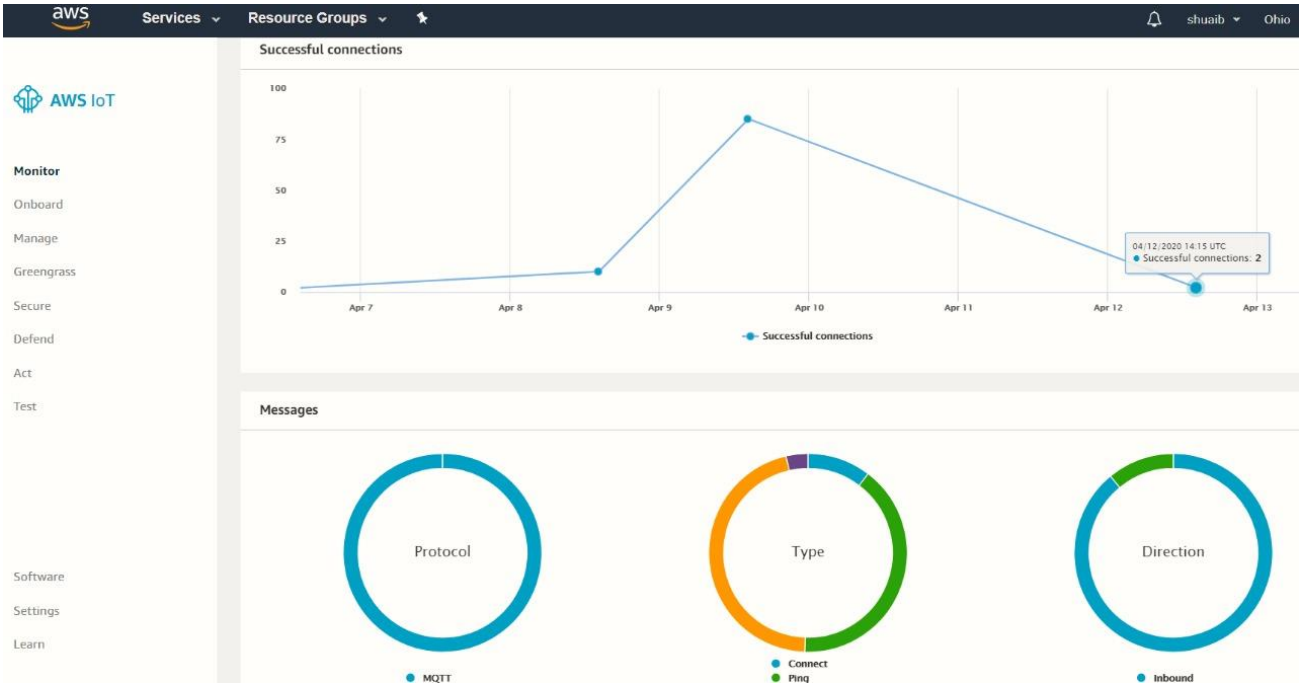
8. Choose **Attach a policy**.

9. For **Add a policy for your thing**, choose **test**, and then choose **Register Thing**.

   Done.




aws
Services
Resource Groups
shuaib
Ohio
Support
AWS IoT
MQTT client
Connected to Device Gateway on client ID 'iotconsole-1586787500943-0'.
Monitor
Onboard
Manage
Greengrass
Secure
Defend
Act
Test
Subscriptions
Subscribe to a topic
Publish to a topic
Subscribe
Devices publish MQTT messages on topics. You can use this client to subscribe to a topic and receive these messages.
Subscription topic
data
Subscribe to topic
Max message capture
100
Quality of Service
0 - This client will not acknowledge to the Device Gateway that messages are received
1 - This client will acknowledge to the Device Gateway that messages are received


aws
Services
Resource Groups
shuaib
Ohio
AWS IoT
Monitor
Onboard
Manage
Greengrass
Secure
Defend
Act
Test
Software
Settings
Learn
Successful connections
100
75
50
25
0
Apr 7
Apr 8
Apr 9
Apr 10
Apr 11
Apr 12
Apr 13
04/12/2020 14:15 UTC
Successful connections: 2
Successful connections
Messages
Protocol
MQTT
Type
Connect
Ping
Direction
Inbound

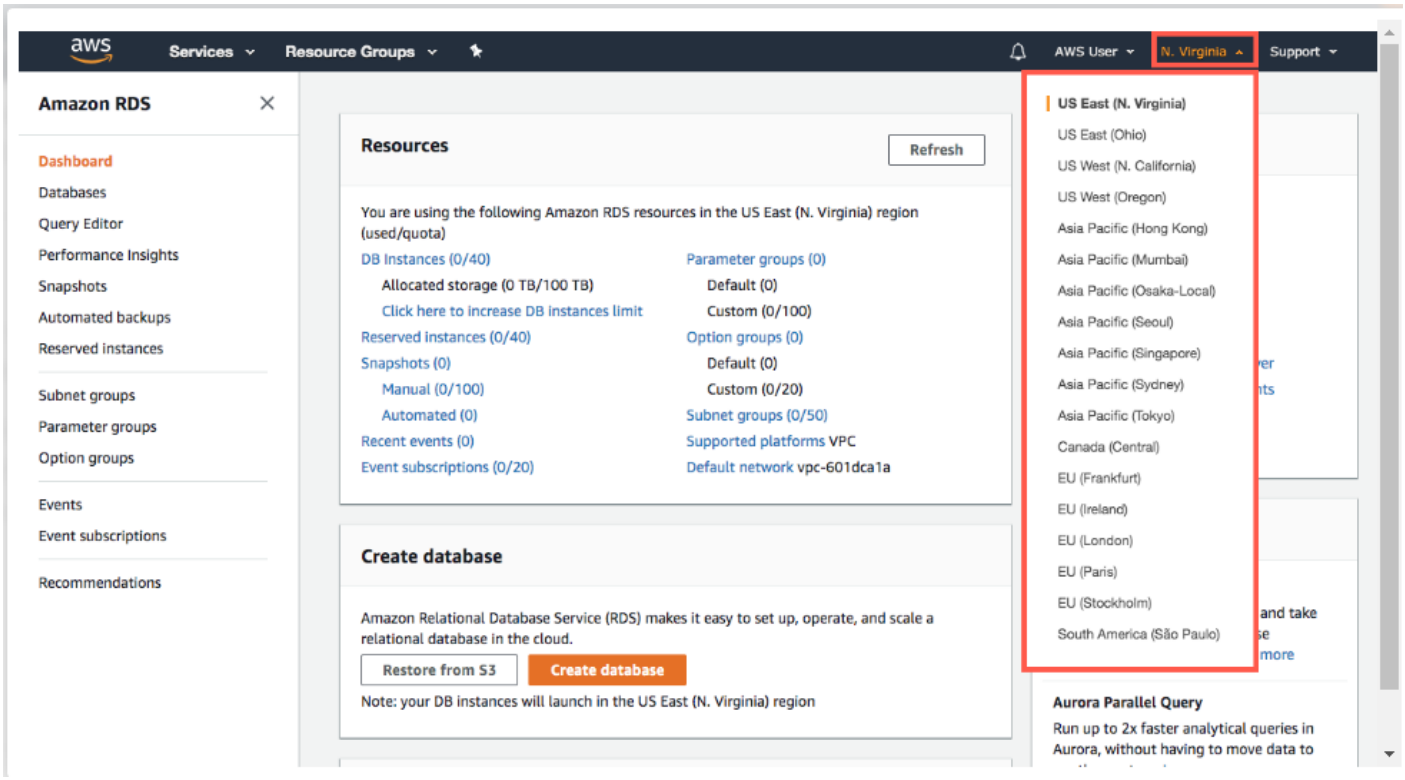
Appendix-7: Enabling MySQL on RDS in AWS Free Tier

Enabling MySQL on RDS involves the following steps:

1. ***Login to AWS account>>management console***
2. ***In the list of services, select Database>>RDS***
3. ***Create a MySQL DB Instance***

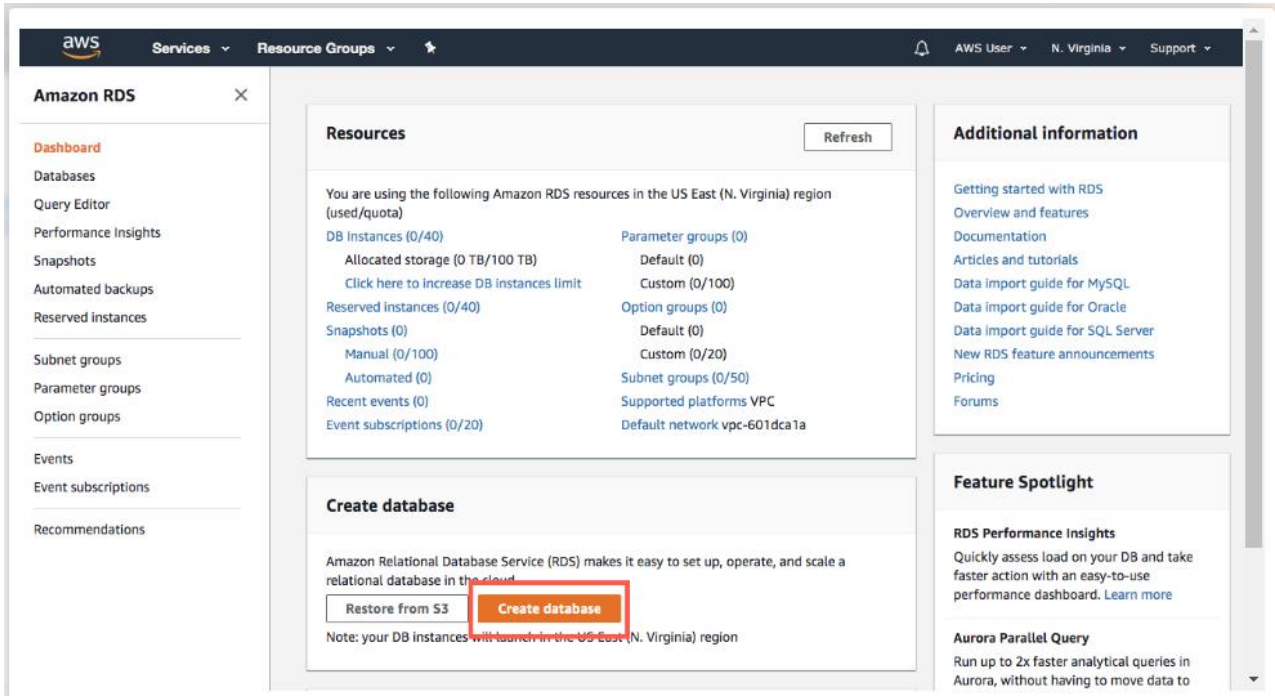
In this step, the Amazon RDS is utilized to create a MySQL DB Instance with db.t2.micro DB instance class, 20 GB of storage, and automated backups enabled with a retention period of one day. AWS Cloud resources are housed in highly available data center facilities in different areas of the world. Each region contains multiple distinct locations called Availability Zones. A developer can host his Amazon RDS activity in the region of his choice.

In the top right corner of the Amazon RDS console (refer screenshot in the next page), select the Region in DB instance has to be created. In the proposed system design, Ohio region was selected.

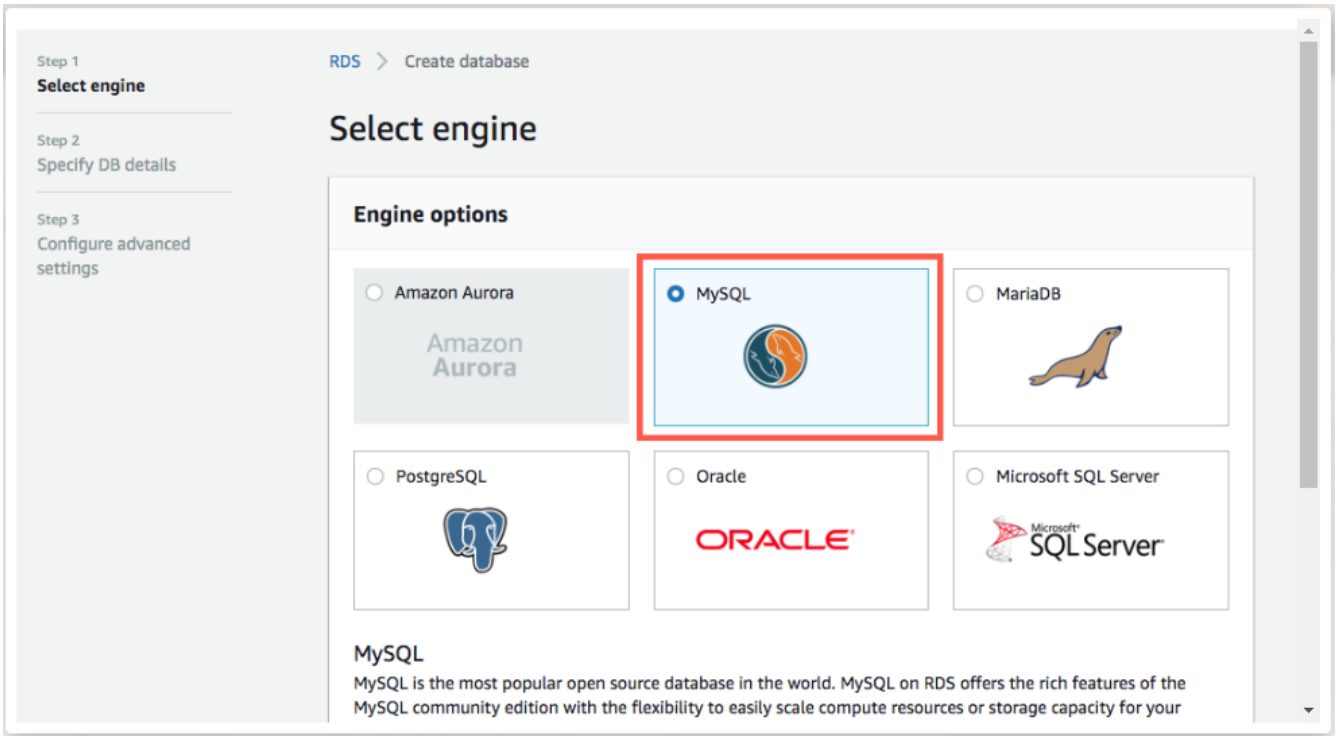


1. Create database

In the Create database section, choose Create database.



2. Select Database engine (Select MySQL) to be installed on RDS.



3. Configure DB instance

Instance specifications:

License model: Select the default general-public-license to use the general license agreement for MySQL. MySQL has only one license model.

DB engine version: Select the default version of MySQL. MySQL 5.7.22 was selected for the proposed system design.

DB instance class: Select db.t2.micro --- 1vCPU, 1 GIB RAM. This equates to 1 GB memory and 1 vCPU.

Storage type: Select General Purpose (SSD).

Allocated storage: Select the default of 20 to allocate 20 GB of storage for your database. A developer can scale up to a maximum of 16 TB with Amazon RDS for MySQL.

Settings:

DB instance identifier: Type a name for the DB instance that is unique for your account in the Region that you selected. For the proposed system design, iot-db was selected as the DB instance identifier.

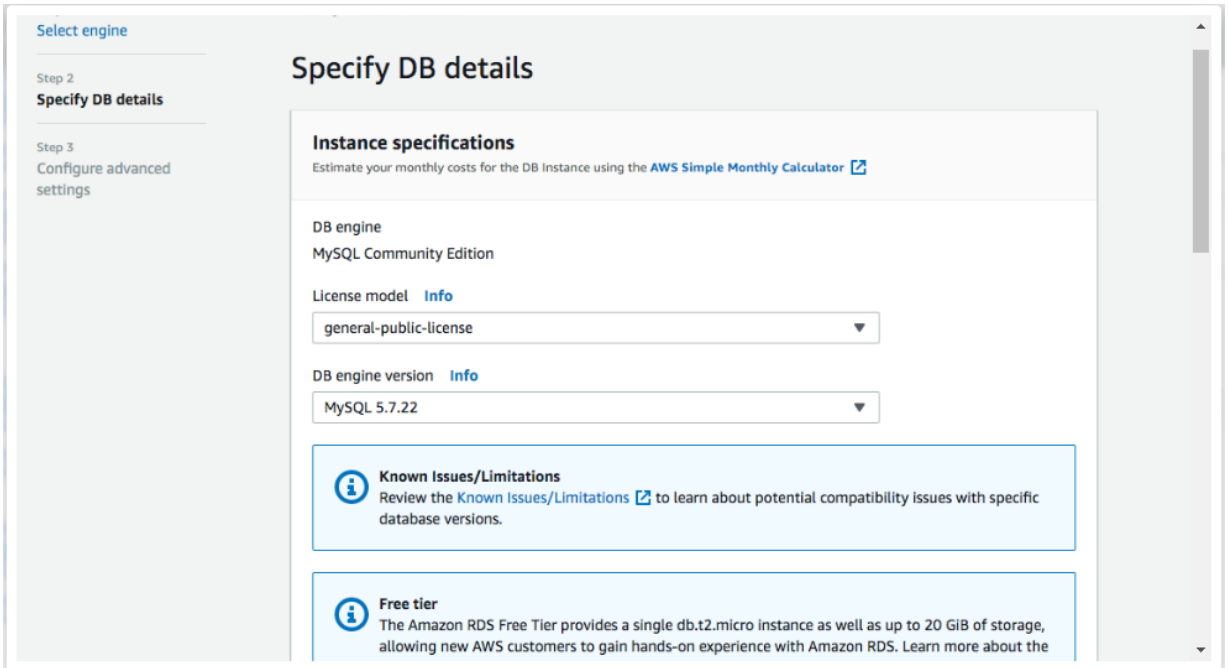
Master Username: Type a username that you will use to log in to your DB instance. In the proposed system design, admin is used as the Master Username.

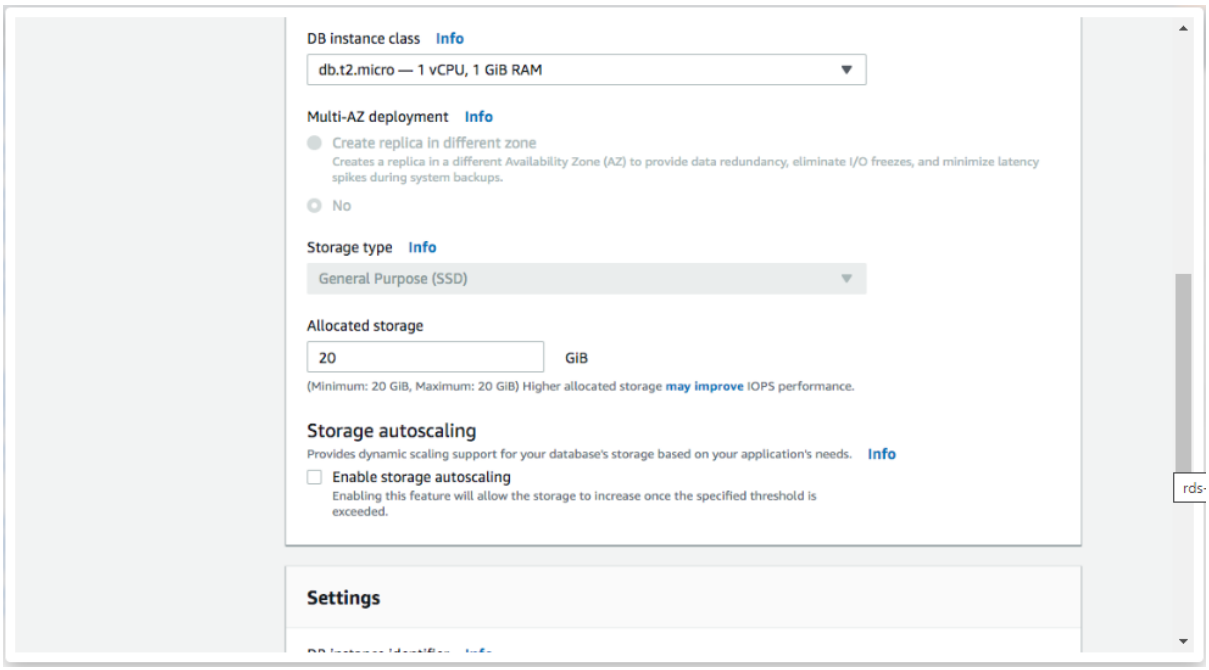
Master password: Type a password that contains from 8 to 41 printable ASCII characters (excluding /,", and @) for the master user password.

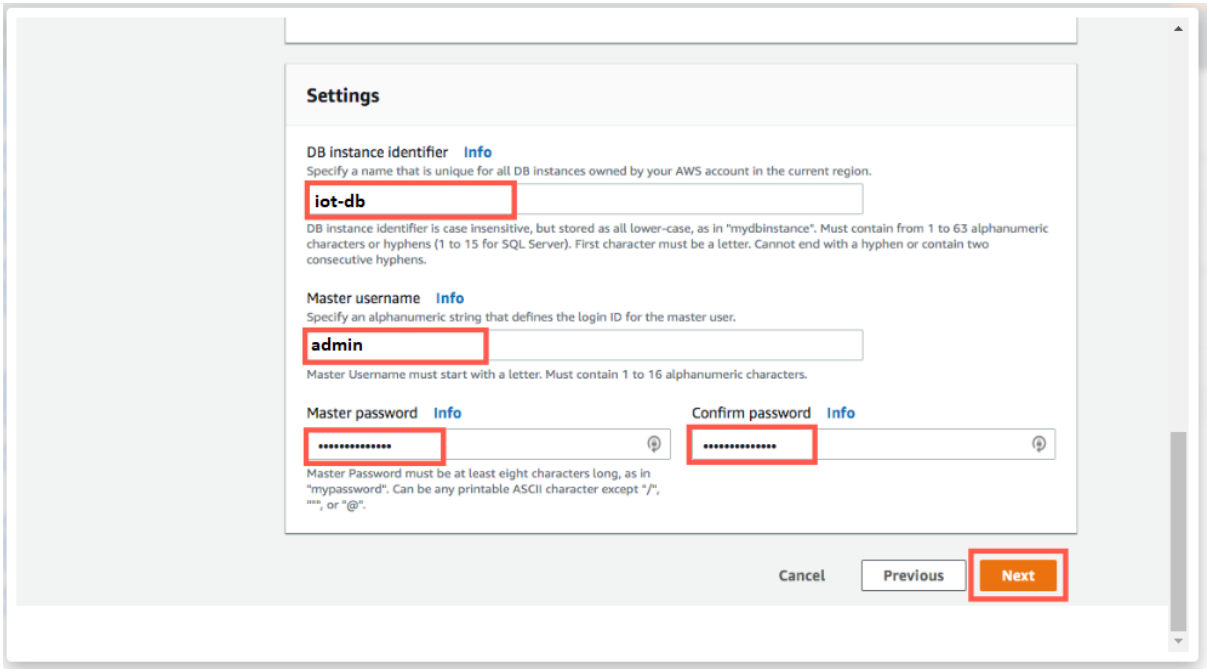
Confirm password: Retype the chosen password.

Allocated Storage: 5GB of the total storage was allocated for the indoor environmental database by typing 5.

Click Next

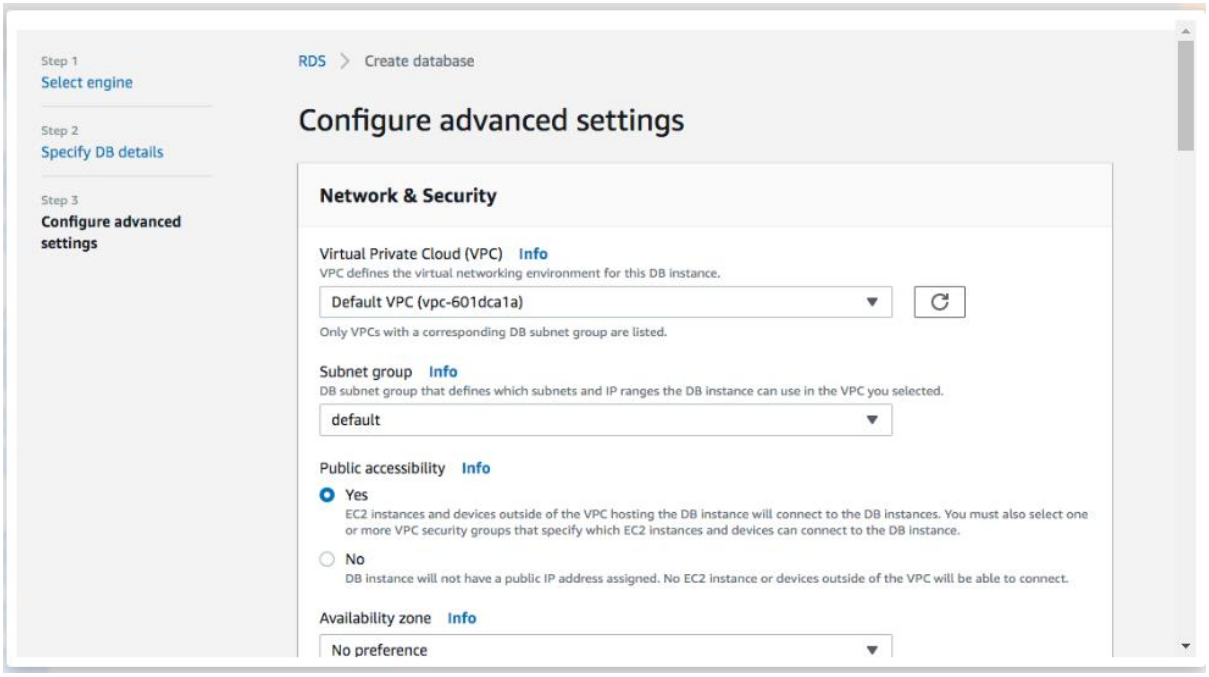



DB instance class Info
db.t2.micro — 1 vCPU, 1 GiB RAM
Multi-AZ deployment Info
Create replica in different zone
Creates a replica in a different Availability Zone (AZ) to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.
No
Storage type Info
General Purpose (SSD)
Allocated storage
20
GiB
(Minimum: 20 GiB, Maximum: 20 GiB) Higher allocated storage may improve IOPS performance.
Storage autoscaling
Provides dynamic scaling support for your database's storage based on your application's needs. Info
Enable storage autoscaling
Enabling this feature will allow the storage to increase once the specified threshold is exceeded.
Settings


Settings
DB instance identifier Info
Specify a name that is unique for all DB instances owned by your AWS account in the current region.
iot-db
DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.
Master username Info
Specify an alphanumeric string that defines the login ID for the master user.
admin
Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.
Master password Info
Confirm password Info
Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", """, or "@".
Cancel
Previous
Next

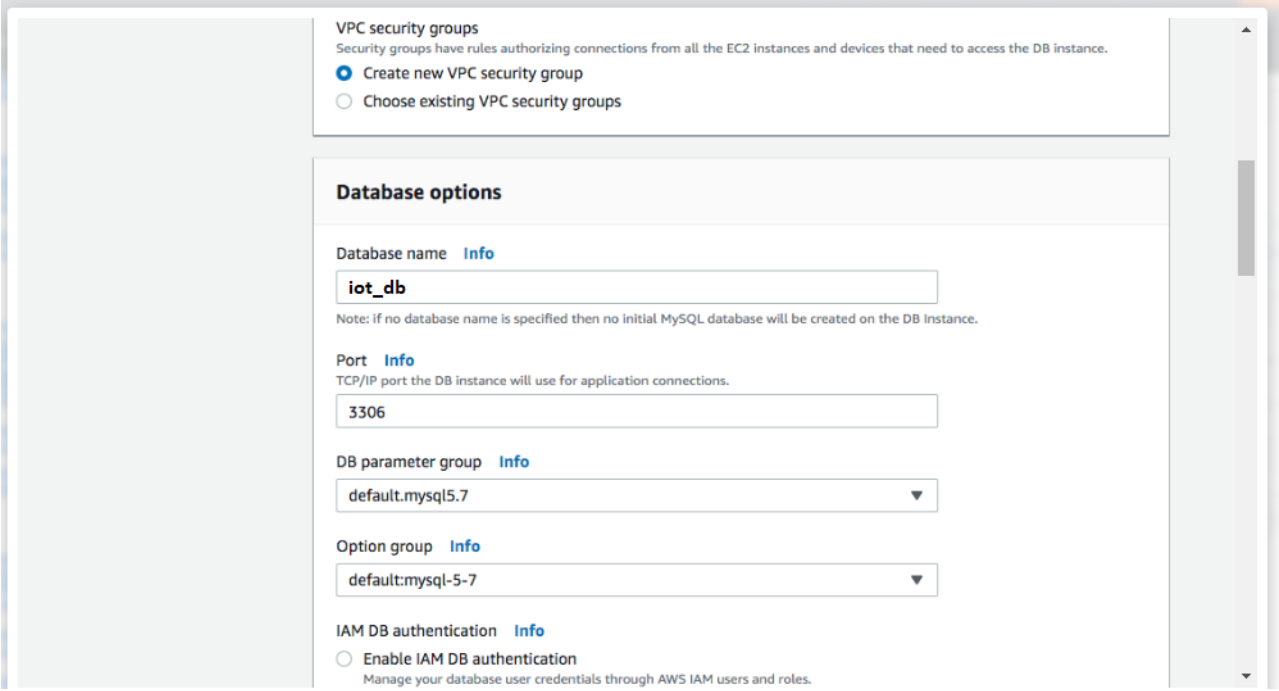
4. Configure advanced settings

Network & Security

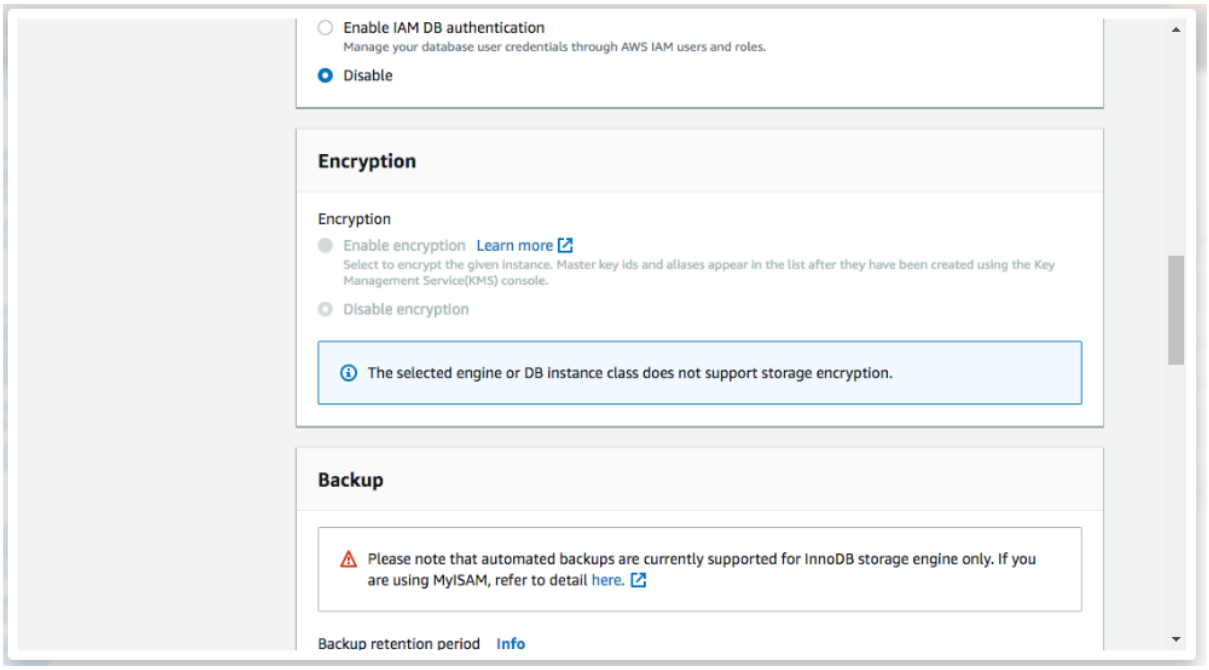


- Virtual Private Cloud (VPC): *Select Default VPC.*
- Subnet Group: *Choose the default subnet group.*
- Public Accessibility: *Choose Yes*. This will allocate an IP address for the system's database instance so that direct connection to the database from the sensor node gateway (developer's device) is possible.
- Availability Zone: *Choose No preference*.
- VPC security groups: *Select Create new VPC security group*. This will create a security group that will allow connection from the IP address of the device that you are currently using to the database created.

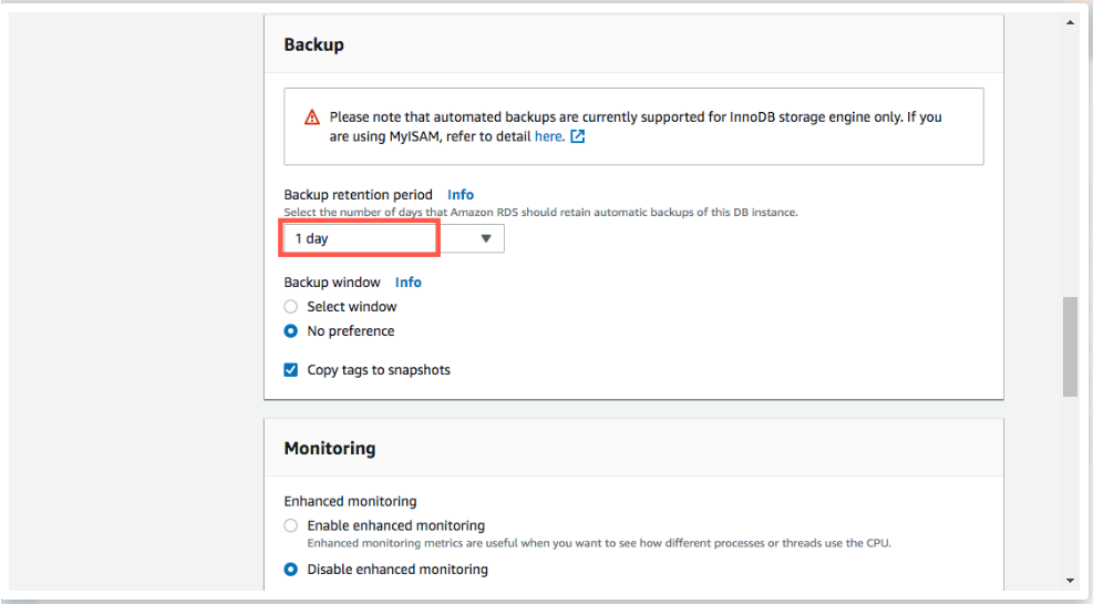
Database Options



- *Database Name*: Type a database name as "iot_db"
- *Port*: Leave the default value of 3306.
- *Db Parameter Group*: Leave the default value of default.mysql5.6.
- *Option Group*: Select the default value of default: mysql5.7. Amazon RDS uses option groups to enable and configure additional features.
- *IAM DB Authentication*: Select Disable. (This option allows the developer to manage his database credentials using AWS IAM users and groups).

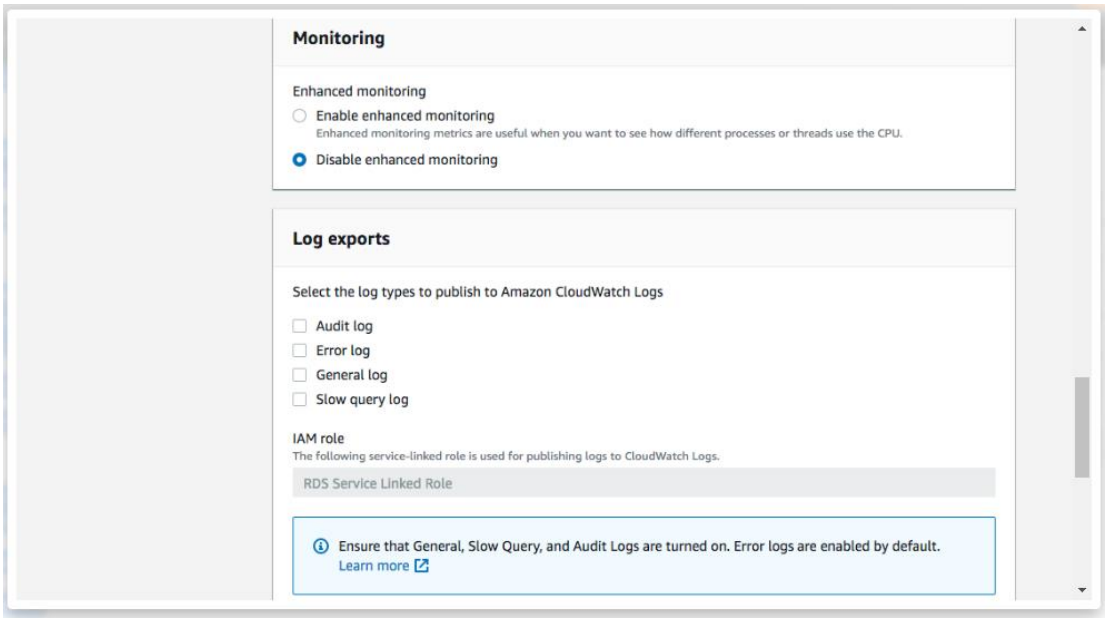


Backup



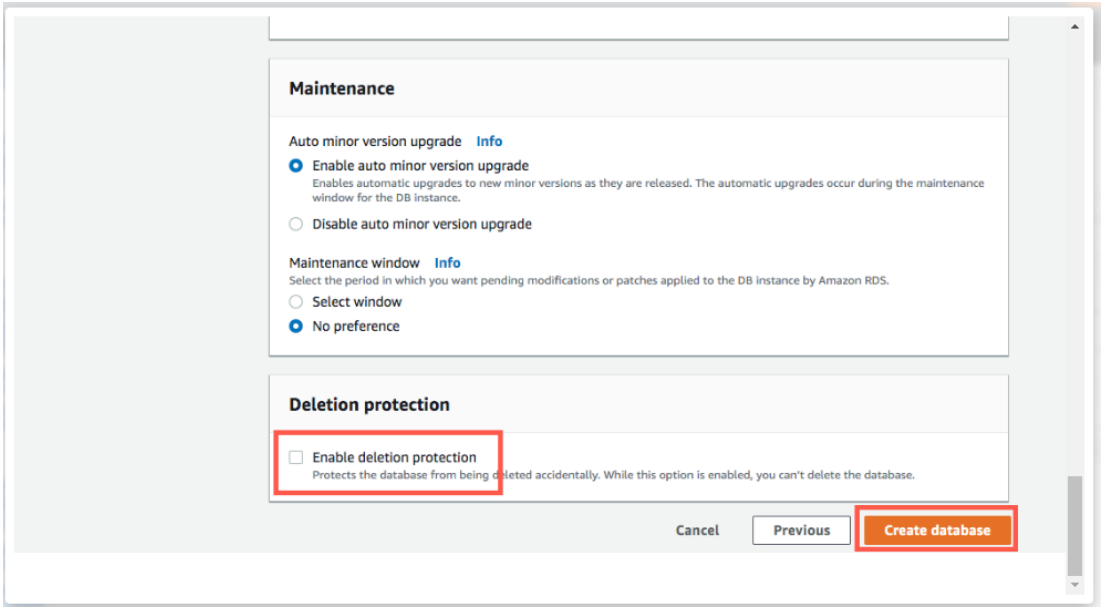
- Backup retention period: *Set this value to 1 day.*
- Backup window: *Use the default of No preference.*

Monitoring



- Enhanced Monitoring: Select Disable enhanced monitoring to stay within the free tier.
- Performance Insights: Select Disable Performance Insights.

Maintenance



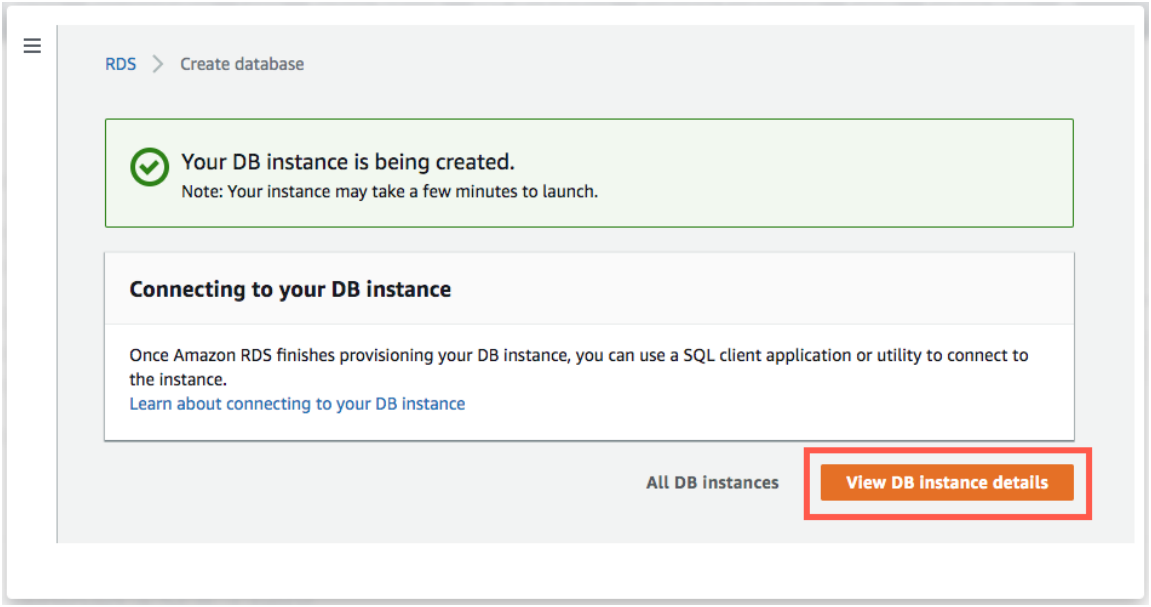
- Auto minor version upgrade: *Select Enable auto minor version upgrade.*
- Maintenance Window: *Select No preference*.

Deletion Protection

Clear Enable deletion protection.

Click Create database.

After clicking Create database button, the following screen appears

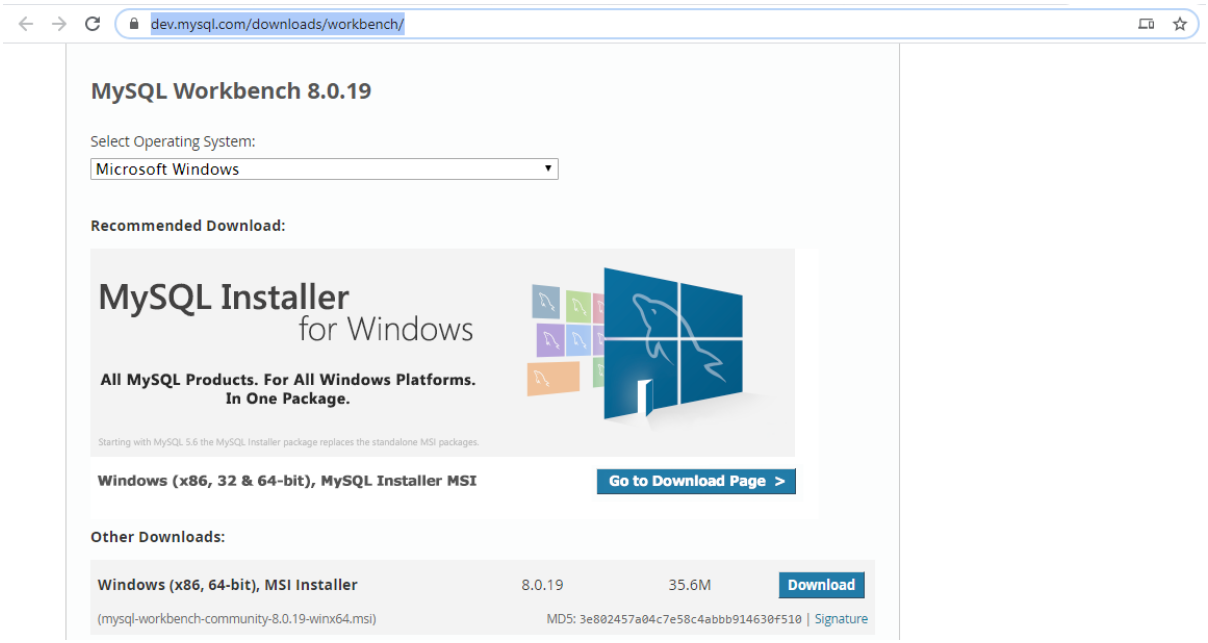


5. Click 'View DB Instance details' button

The new DB instance appears in the list of DB instances on the RDS console. The DB instance will have a status of creating until the DB instance is created and ready for use. When the state changes to available, the developer can connect to a database on the DB instance.

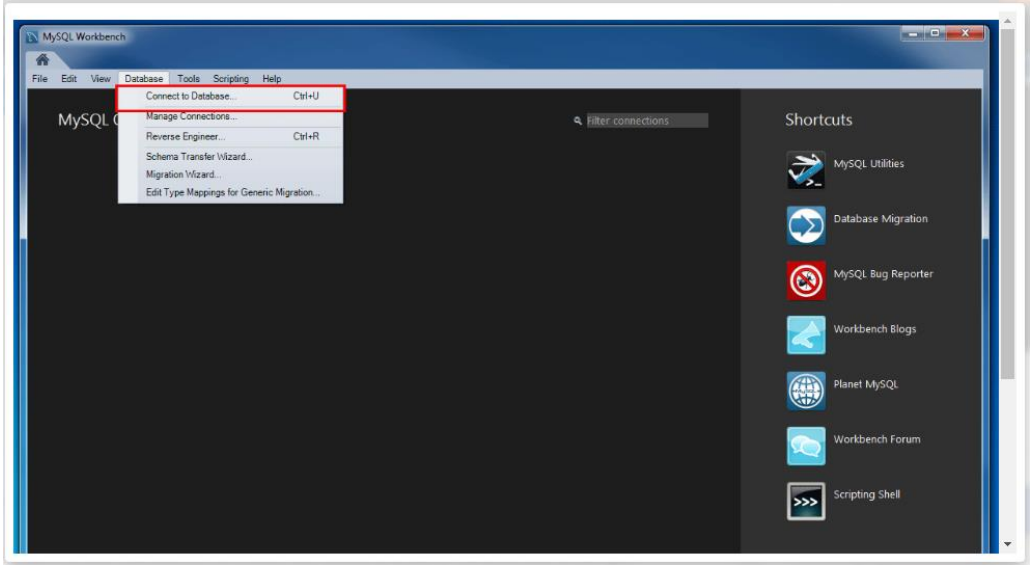
6. Download a SQL Client

Once the database instance creation is complete and the status changes to available, a developer can connect to a database on the DB instance using any standard SQL client. In the proposed system design, the popular SQL client MySQL Workbench 8.0.19 for Windows was downloaded from the website https://dev.mysql.com/downloads/workbench/ and installed as shown in the next screenshot.



7. Connect to MySQL Database from Workbench

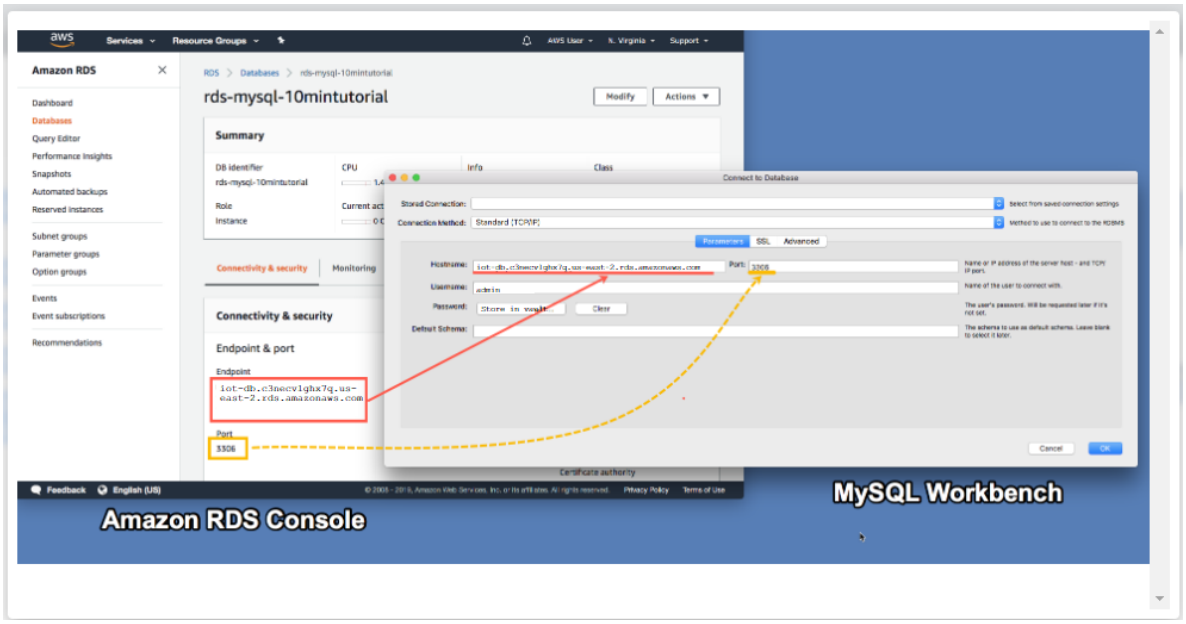
(a), Launch the MySQL Workbench application and go to Database > Connect to Database (Ctrl+U) from the menu bar.



b). A dialog box appears. Enter the following:

- *Hostname*: You can find your hostname on the Amazon RDS console as shown in the screenshot to the right. In this case it is "iot-db.c3necvlghx7q.us-east-2.rds.amazonaws.com".
- *Port*: The default value should be 3306.
- *Username*: Type in the username you created for the Amazon RDS database. In this case, it is 'admin'
- *Password*: Click Store in Vault and enter the password that you used when creating the Amazon RDS database. In this case it is "**iotadmin**"

Click OK



8. ***Connected to database:*** One can now browse different schema objects, create tables and run queries.

Appendix-8 : AWS MySQL Database Script

File Name: iot_db_20200413

```
CREATE DATABASE  IF NOT EXISTS `iot_db` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `iot_db`;
-- MySQL dump 10.13  Distrib 8.0.19, for Win64 (x86_64)
-- Host: iot-db.c3necvlghx7q.us-east-2.rds.amazonaws.com    Database: iot_db
-- ------------------------------------------------------
-- Server version 5.7.22-log
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;
--
-- Table structure for table `asset_types`
--
DROP TABLE IF EXISTS `asset_types`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `asset_types` (
  `asset_type_id` int(11) NOT NULL AUTO_INCREMENT,
  `asset_type` varchar(45) NOT NULL COMMENT 'Gateway, Wifi shield, Sensor',
  PRIMARY KEY (`asset_type_id`),
  UNIQUE KEY `asset_type_UNIQUE` (`asset_type`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Table structure for table `assets`
--
DROP TABLE IF EXISTS `assets`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `assets` (
  `asset_id` int(11) NOT NULL AUTO_INCREMENT,
  `asset_type_id` int(11) NOT NULL COMMENT 'FK connected to asset_type_id of asset_types
table',
  `asset_slno` varchar(45) NOT NULL,
  `device_id` varchar(45) NOT NULL,
  `activation_date` datetime NOT NULL,
  PRIMARY KEY (`asset_id`),
  KEY `fk_asset_type_id_idx` (`asset_type_id`),
  KEY `unique_deviceid` (`device_id`),
```

```
  CONSTRAINT `fk_asset_type_id` FOREIGN KEY (`asset_type_id`) REFERENCES `asset_types`
(`asset_type_id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `customers`
--
DROP TABLE IF EXISTS `customers`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `customers` (
  `customer_id` int(11) NOT NULL AUTO_INCREMENT,
  `customer_name` varchar(150) NOT NULL,
  `login_id` varchar(25) NOT NULL,
  `login_password` varchar(25) NOT NULL,
  `address1` varchar(255) DEFAULT NULL,
  `address2` varchar(255) DEFAULT NULL,
  `city` varchar(100) DEFAULT NULL,
  `country` varchar(100) DEFAULT NULL,
  `zip_postcode` varchar(10) DEFAULT NULL,
  `phone` varchar(15) DEFAULT NULL,
  `email_id` varchar(50) DEFAULT NULL,
  `active` tinyint(1) NOT NULL,
  `activation_date` datetime NOT NULL,
  PRIMARY KEY (`customer_id`),
  UNIQUE KEY `login_id_UNIQUE` (`login_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `gateway_sensors`
--
DROP TABLE IF EXISTS `gateway_sensors`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `gateway_sensors` (
  `gatewaysensor_id` int(11) NOT NULL AUTO_INCREMENT,
  `gateway_id` varchar(45) NOT NULL COMMENT 'FK connected to device_id of assets table',
  `wifishield_id` varchar(45) NOT NULL COMMENT 'FK connected to device_id of assets
table',
  `sensor_name` varchar(45) NOT NULL,
  `sensor_id` varchar(45) NOT NULL COMMENT 'FK connected to device_id of assets table',
  `active` tinyint(1) NOT NULL,
  `activation_date` datetime NOT NULL,
  PRIMARY KEY (`gatewaysensor_id`),
  KEY `fk_gateway_id_idx` (`gateway_id`),
  KEY `fk_ wifishield_id_idx` (`wifishield_id`),
  KEY `fk_ sensor_id_idx` (`sensor_id`),
  CONSTRAINT `fk_gateway_sensors_gateway_id` FOREIGN KEY (`gateway_id`) REFERENCES
`assets` (`device_id`),
```

```
  CONSTRAINT `fk_gateway_sensors_sensor_id` FOREIGN KEY (`sensor_id`) REFERENCES `assets`
(`device_id`),
  CONSTRAINT `fk_gateway_sensors_wifishield_id` FOREIGN KEY (`wifishield_id`) REFERENCES
`assets` (`device_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `installationlog`
--
DROP TABLE IF EXISTS `installationlog`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `installationlog` (
  `installation_id` int(11) NOT NULL AUTO_INCREMENT,
  `customer_id` int(11) NOT NULL COMMENT 'FK connected to customer_id of customers
table',
  `gateway_id` varchar(25) NOT NULL COMMENT 'FK connected to device_id of assets table',
  `active` tinyint(1) NOT NULL,
  `activation_date` datetime NOT NULL,
  PRIMARY KEY (`installation_id`),
  KEY `fk_customer_id_idx` (`customer_id`),
  KEY `fk_gateway_id_idx` (`gateway_id`),
  CONSTRAINT `fk_customer_id` FOREIGN KEY (`customer_id`) REFERENCES `customers`
(`customer_id`),
  CONSTRAINT `fk_gateway_id` FOREIGN KEY (`gateway_id`) REFERENCES `assets` (`device_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
--
-- Table structure for table `recordings`
--
DROP TABLE IF EXISTS `recordings`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `recordings` (
  `recording_id` int(11) NOT NULL AUTO_INCREMENT,
  `gateway_id` varchar(45) NOT NULL COMMENT 'FK connected to device_id of assets table',
  `node_id` varchar(45) NOT NULL COMMENT 'FK connected to device_id of assets table',
  `temperature` varchar(45) DEFAULT NULL COMMENT 'temperature, humidity, etc',
  `humidity` varchar(45) DEFAULT NULL,
  `mq2_sensor` varchar(45) DEFAULT NULL COMMENT 'celcius,etc',
  `recorded_datetime` datetime NOT NULL,
  `sensor_status` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`recording_id`),
  KEY `fk_ recordings_gateway_id_idx` (`gateway_id`),
  CONSTRAINT `fk_ recordings_gateway_id` FOREIGN KEY (`gateway_id`) REFERENCES `assets`
(`device_id`)
) ENGINE=InnoDB AUTO_INCREMENT=184 DEFAULT CHARSET=latin1;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```

```
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

Appendix-9: Web Application Deployment Procedure

A. Web Server Deployment (IOT - ADMIN – SERVER)

Platform: NodeJs, MySql
Hosting: Using AWS Elastic Beanstalk

Note:

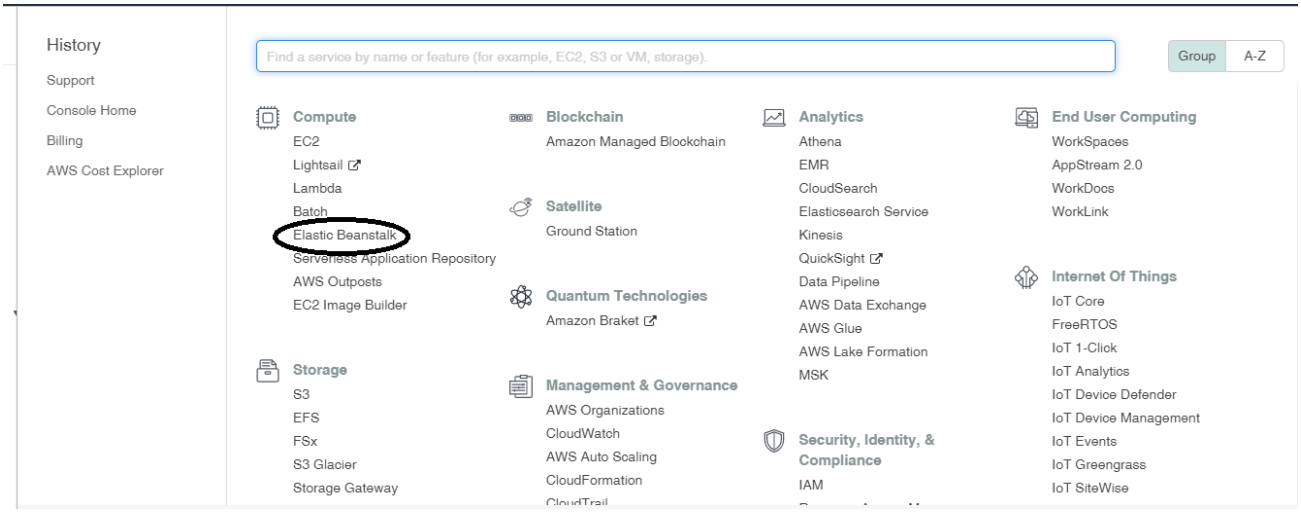
Source Code Folder Name: **iot-admin-master**

README File Name: **ReadMe-Server.txt**

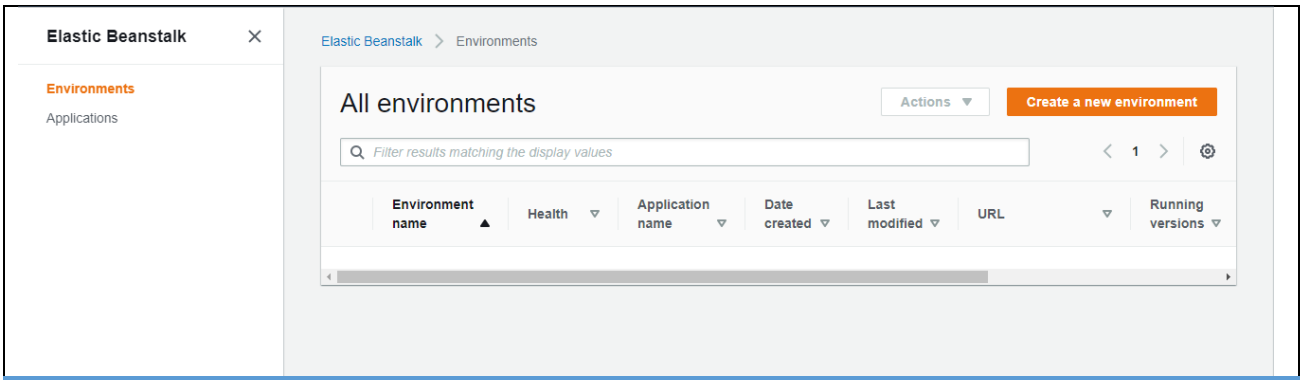
Preparation: Make a zip file of source code. Zip file should not contain any child folder. Zip should contain direct source files.

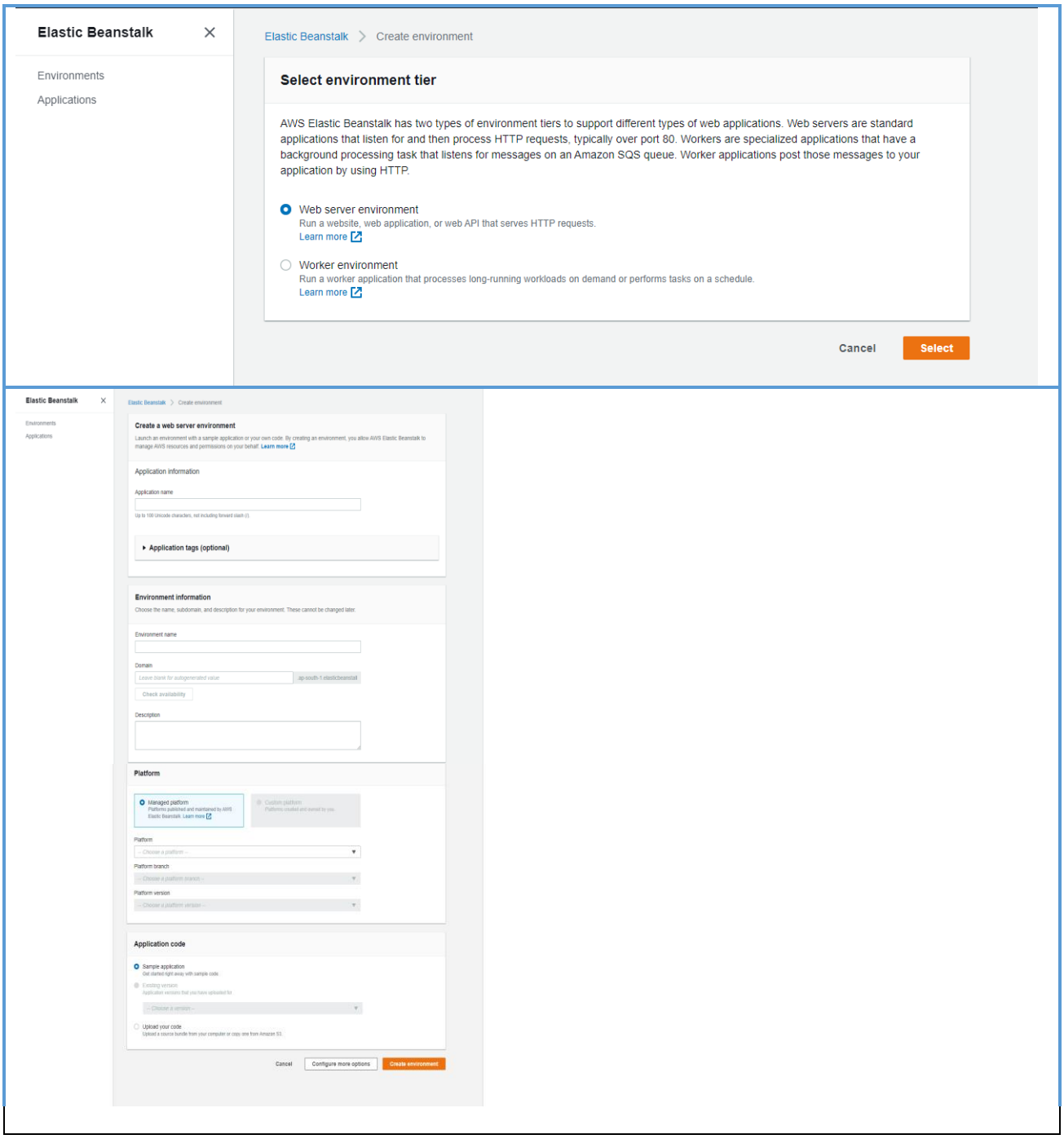
Login to AWS Account:

1. Select Elastic Beanstalk from Services -> EC2 -> Elastic Beanstalk



2. Select Create Application and Fill the application and description, click on create





3. Which will create application. Now click on add environment.

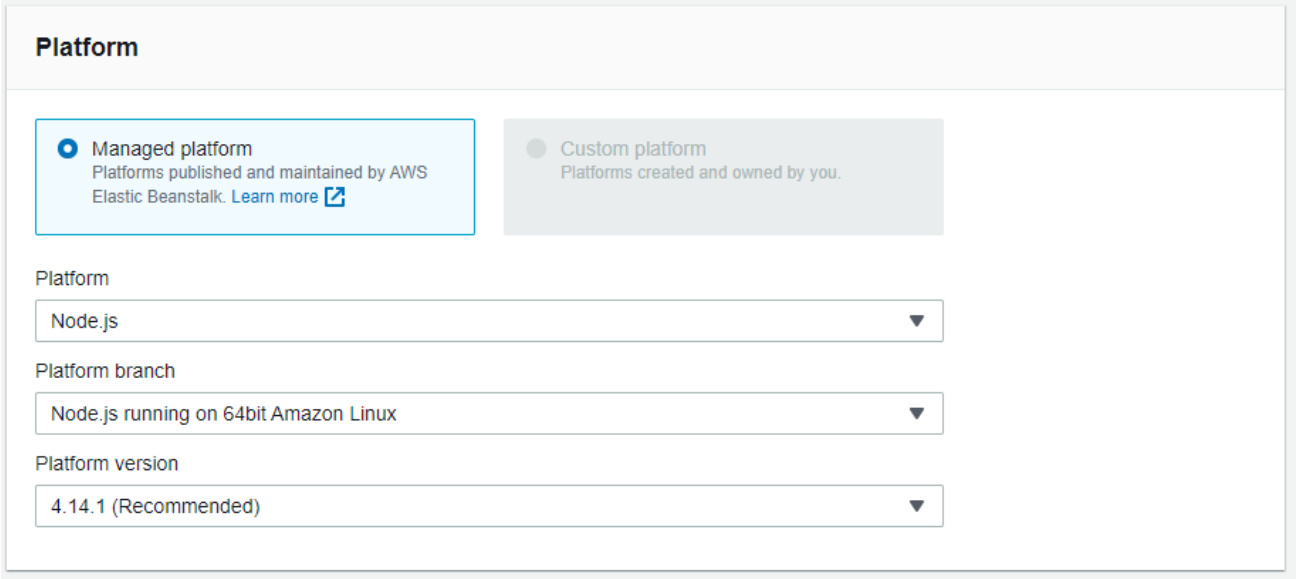
4. Select Web server environment as Environment tier and Click select.

5. Enter Environment name and domain as required.

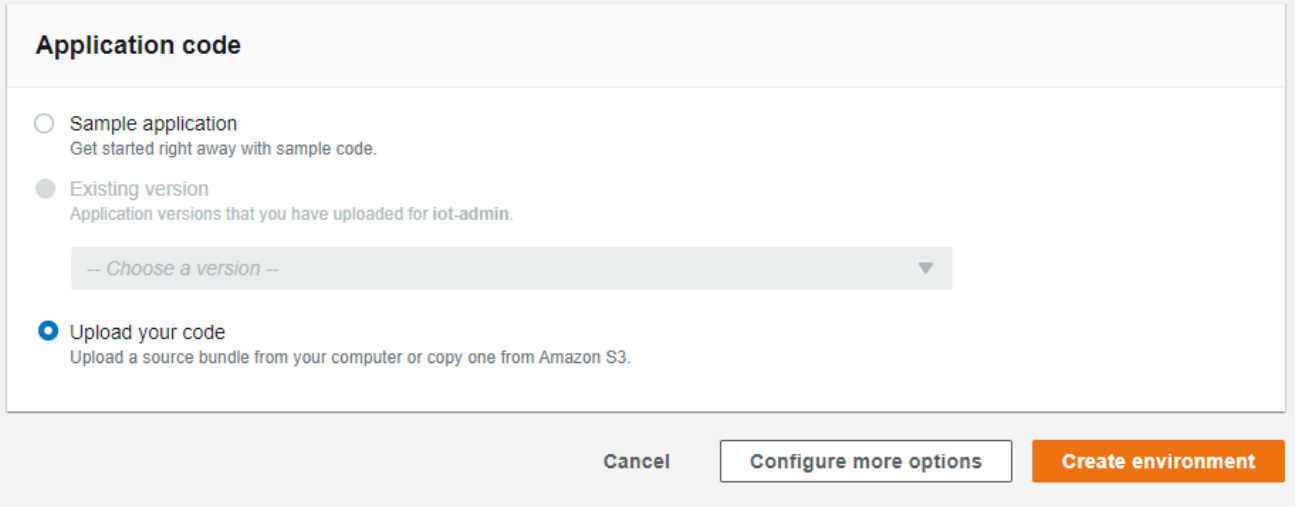
6. In platform section: select Managed Platform.

7: Select platform as NodeJs and Platform Branch as "NodeJs running on 64bit Amazon Linux".

8. Select Platform version as recommended.



9. In the "Application Code" section choose "Upload your code"



10. In the source code origin, choose local file and upload the zip file of server code.

11. Enter the version label if required, can leave default value

12. Select Create environment after all the details. Wait till the environment get creates and URL will get generates.

13. Use the generated end point to access the server.

README-Server.txt

```
IoT Admin:

server.js
-> This is the main file, which contains the server port and the server
configurations
-> Will configure all the routes here

app/config:
* db.config.js
-> this file contains the database configuration

* email-handler.js
-> this file contains the email configuration and method to send email
--------------------------------------------------------------------------------
------------------------------------------------
app/routes
-> this folder contains all the routes files, which receives the call from the
client redirect to the specified controller method

app/routes/auth.routes.js
--> This file is used to handle routing of login requests

app/routes/common.routes.js
-->This file is used to handle routing of common utilities like  dropdown data
population requests

app/routes/assets.routes.js
-->This file is used to handle routing of requests related to adding items

app/routes/gatewaysensor.routes.js
-->This file is used to handle routing of requests related to configuring kits
by linking gatewayid and nodeids

app/routes/customer.routes.js
-->This file is used to handle routing of requests related to adding customers

app/routes/installationlog.routes.js
-->This file is used to handle routing of requests related to activating
installations at customer sites

app/routes/recording.routes.js
-->This file is used to handle routing of requests related to retrieving data
from the recordings table. The recordings table has the values that are recorded
from the sensors.

--------------------------------------------------------------------------------
----------------
```

app/controllers
-> this folder contains all the controller respective tables and related methods

app/controllers/asset.controller.js
-->This file has methods to manage database operations on assets table (to add items)

app/controllers/gatewaysensor.controller.js
-->This file has methods to manage database operations on gateway_sensor table (to link gatewayid and nodeids to configure kits)

app/controllers/installationlog.controller.js
-->This file has methods to manage database operations on installationlog table (to activate installations at customer sites)

app/controllers/customer.controller.js
-->This file has methods to manage database operations on customers table (to add customers)

app/controllers/recordings.controller.js
-->This file has methods to fetch values from recordings table (to display in dashboard)

---

app/models
-> this folder contains all the methods which performs with the database tables. all the queries will be mentioned and perform the action based on the controller request and params

app/models/asset.model.js
app/models/customer.model.js
app/models/db.js
app/models/gatewaysensor.model.js
app/models/installationlog.model.js
app/models/recording.model.js

B. IOT - ADMIN – UI Deployment

Platform: Angular 9

Hosting: AWS S3

Note:

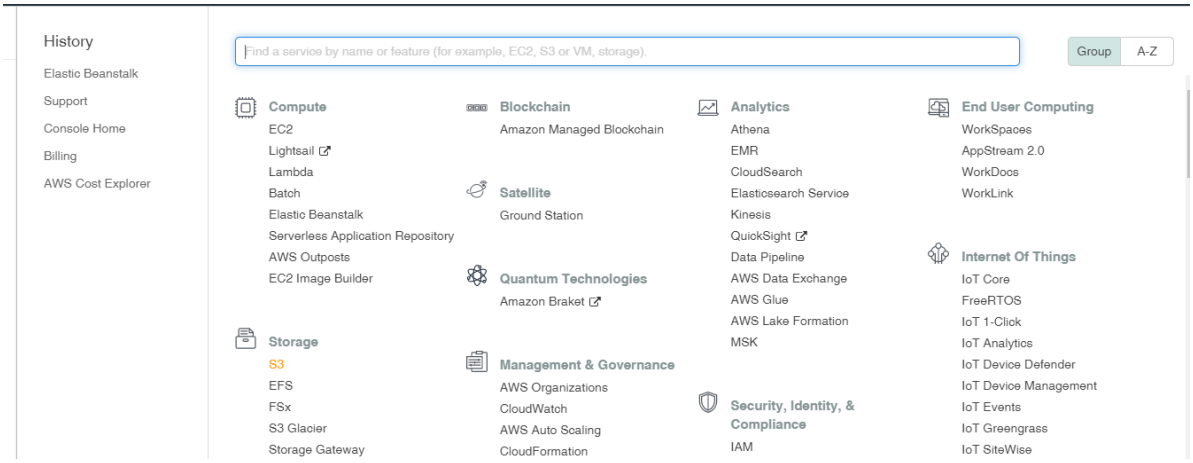
Source Code Folder Name: **iot-admin-ui-master**

README File Name: **ReadMe-UI.txt**

1) Go to source code folder in the command prompt.
2) Run "npm install" to install the necessary libraries.
3) Run "npm run build:prod" to do the production build which creates dist folder with generated files for deployment.

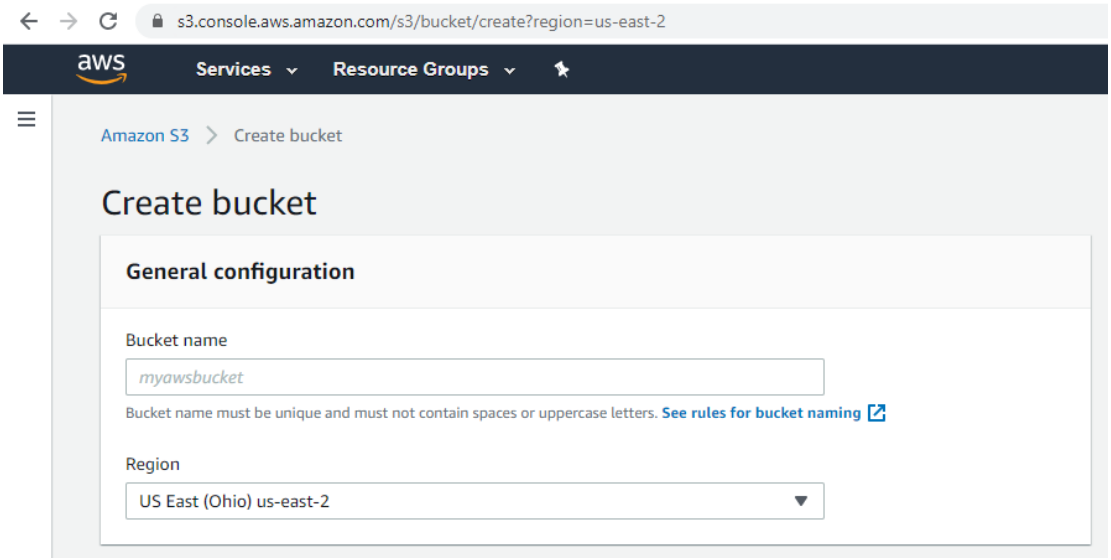
Login to AWS Account:

Select Services -> Under "Storage" Select -> S3

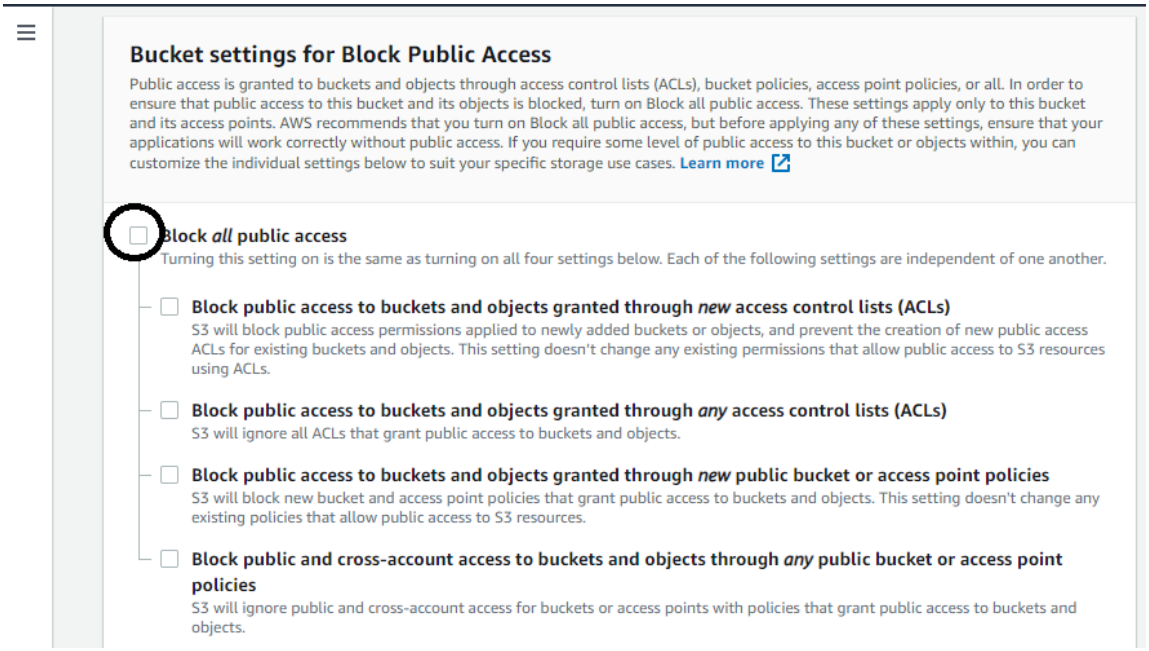


2. Select Create Bucket.

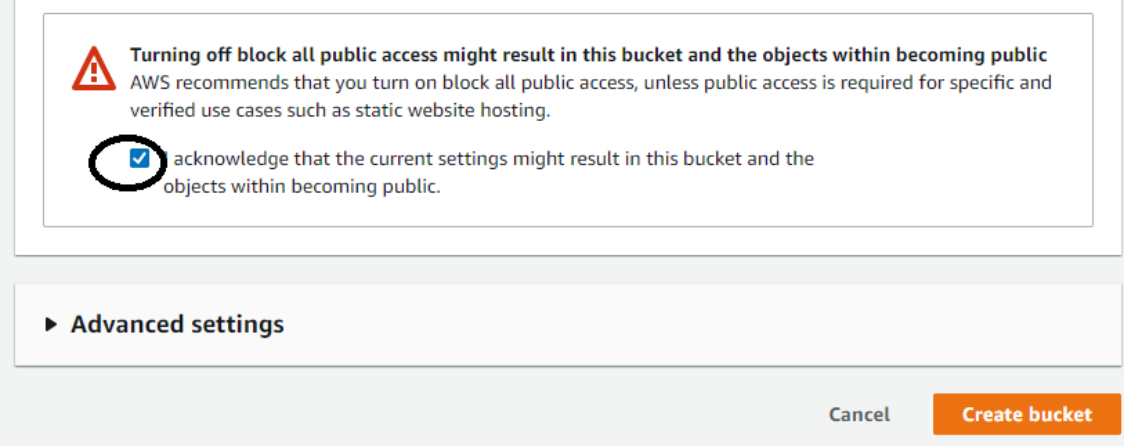
3. Enter bucket name and select preferred region.



Uncheck the Block all public access in the Bucket settings for block public access section



Select check box for acknowledgement and select create bucket



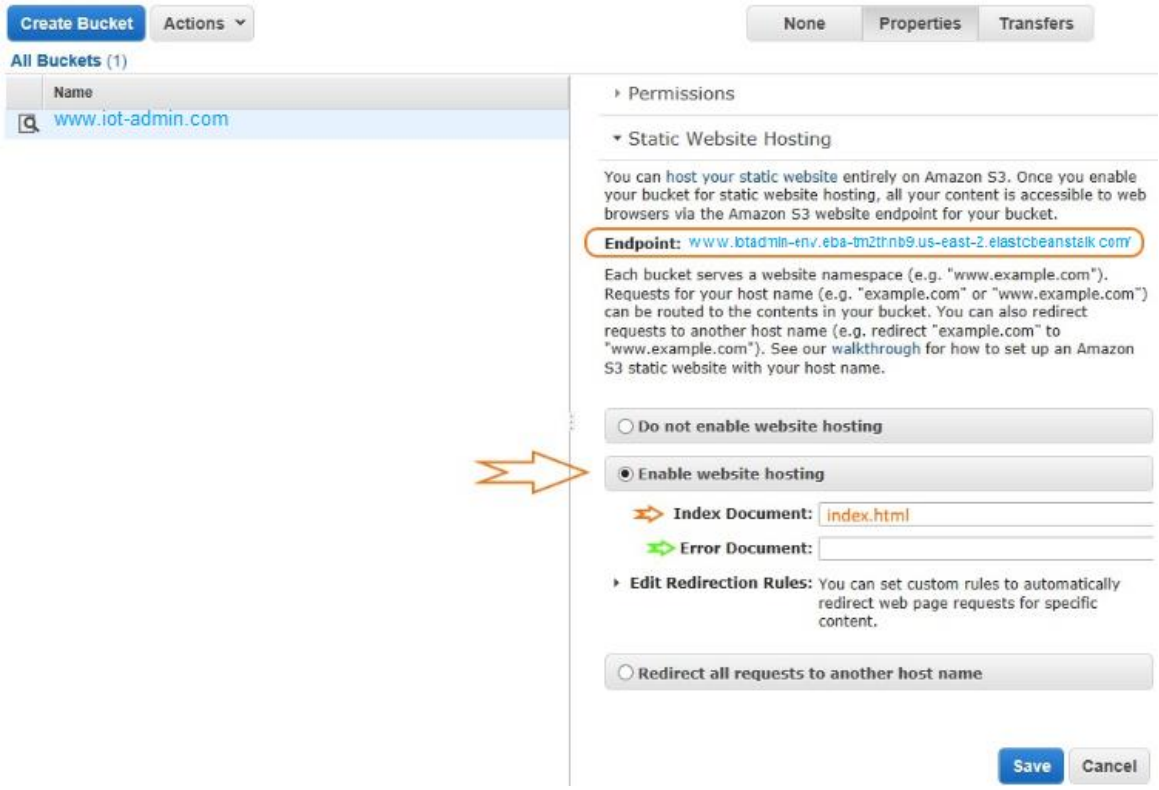
6. Select the newly created bucket from the list

7. Select Properties from menu and select "Static website hosting" in the list

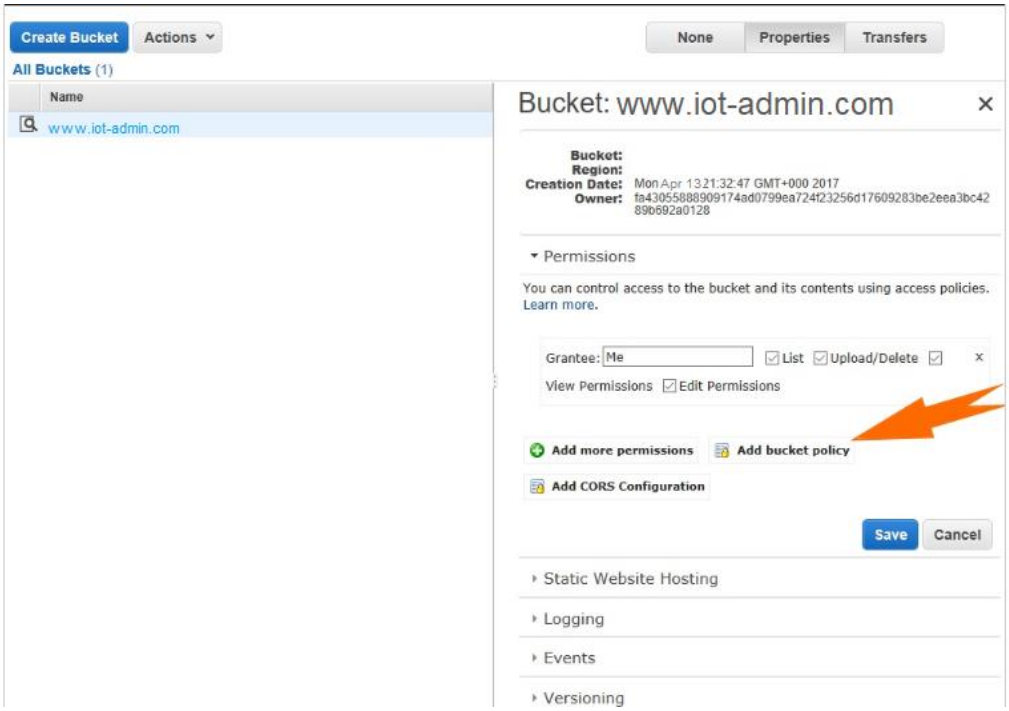
8. Select "Use this bucket to host a website"

9. Enter index.html in index document file.

10. Click on "Save"



Select permissions -> Select Bucket Policy



Enter the following data in the policy

```
{
    "Version": "2008-10-17",
    "Id": "PolicyForCloudFrontPrivateContent",
    "Statement": [
        {
            "Sid": "Allow-Public-Access-To-Bucket",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::<BUCKET_NAME>/*"
        }
    ]
}
```

13. Replace the BUCKET_NAME with the actual bucket name and select save.

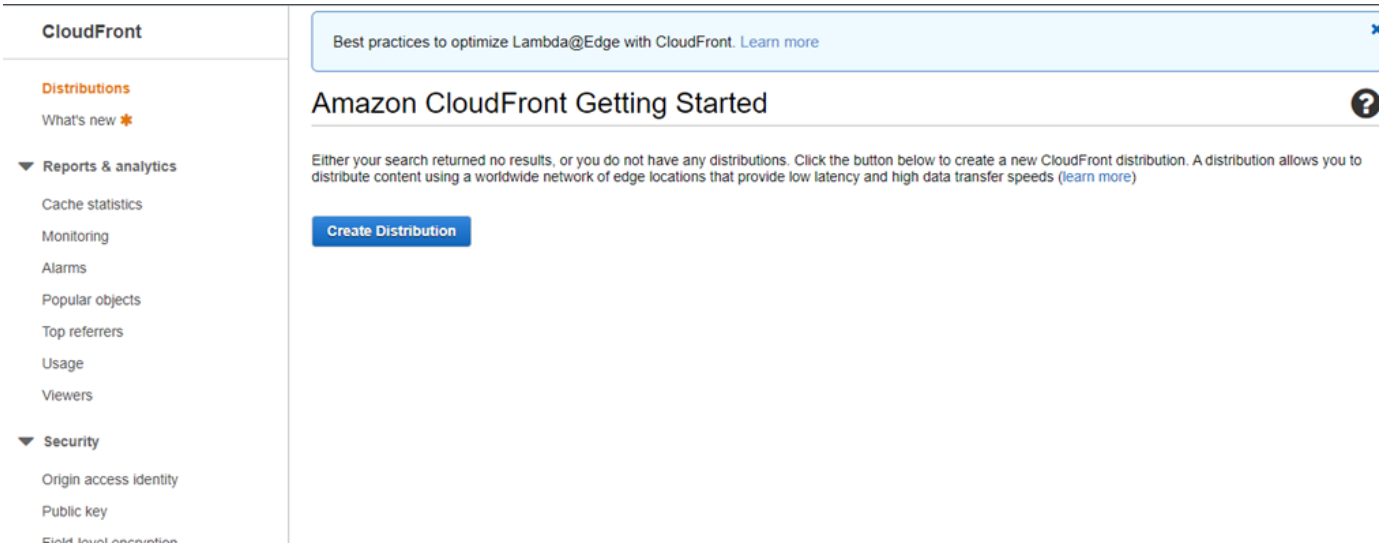
14. Select Overview tab.

15. Select Upload button and Upload the files which generated in the dist folder.

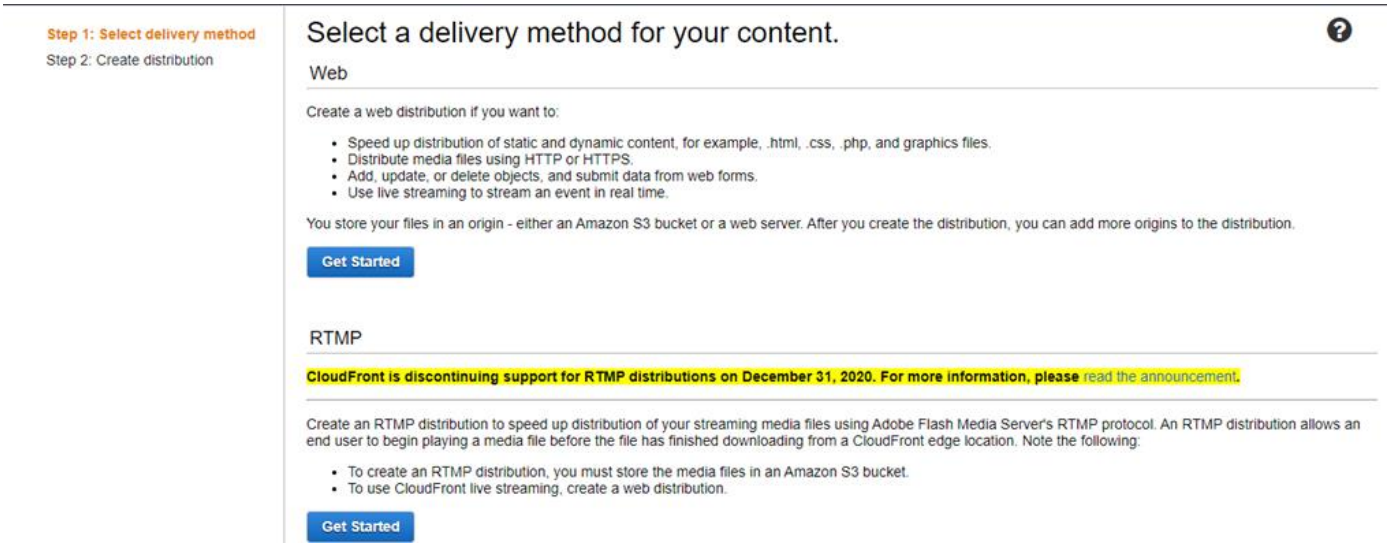
Configure CloudFront:

1. Select Services -> Under "Networking & Content Delivery" -> Select "CloudFront"

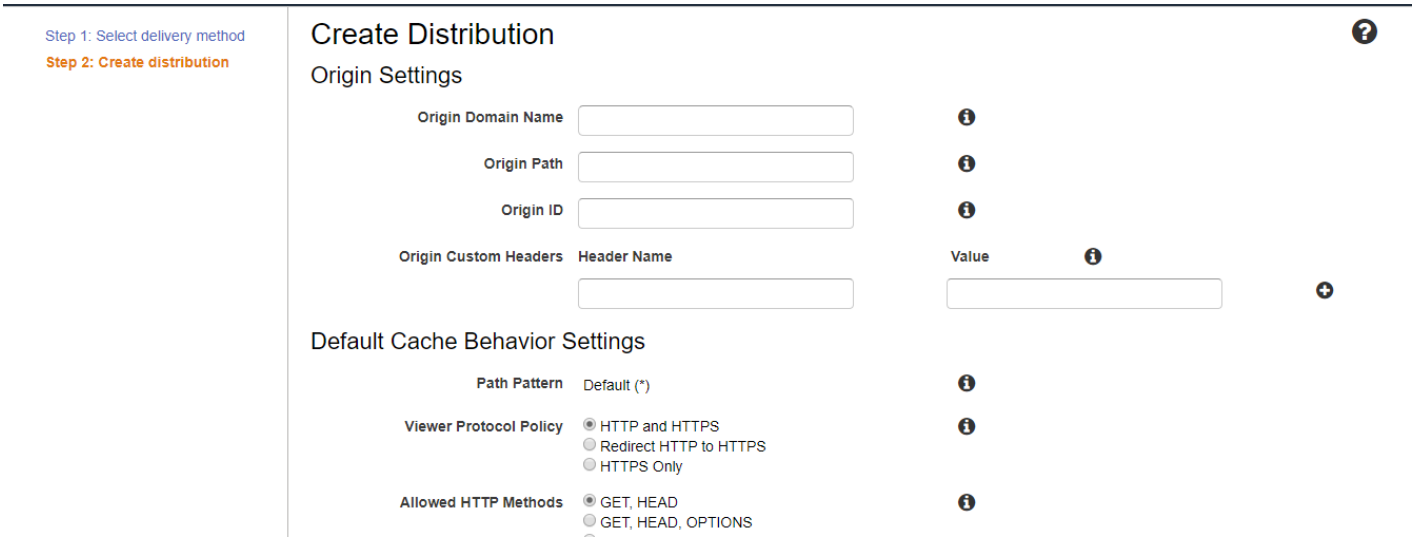
2. Select "Create Distribution"



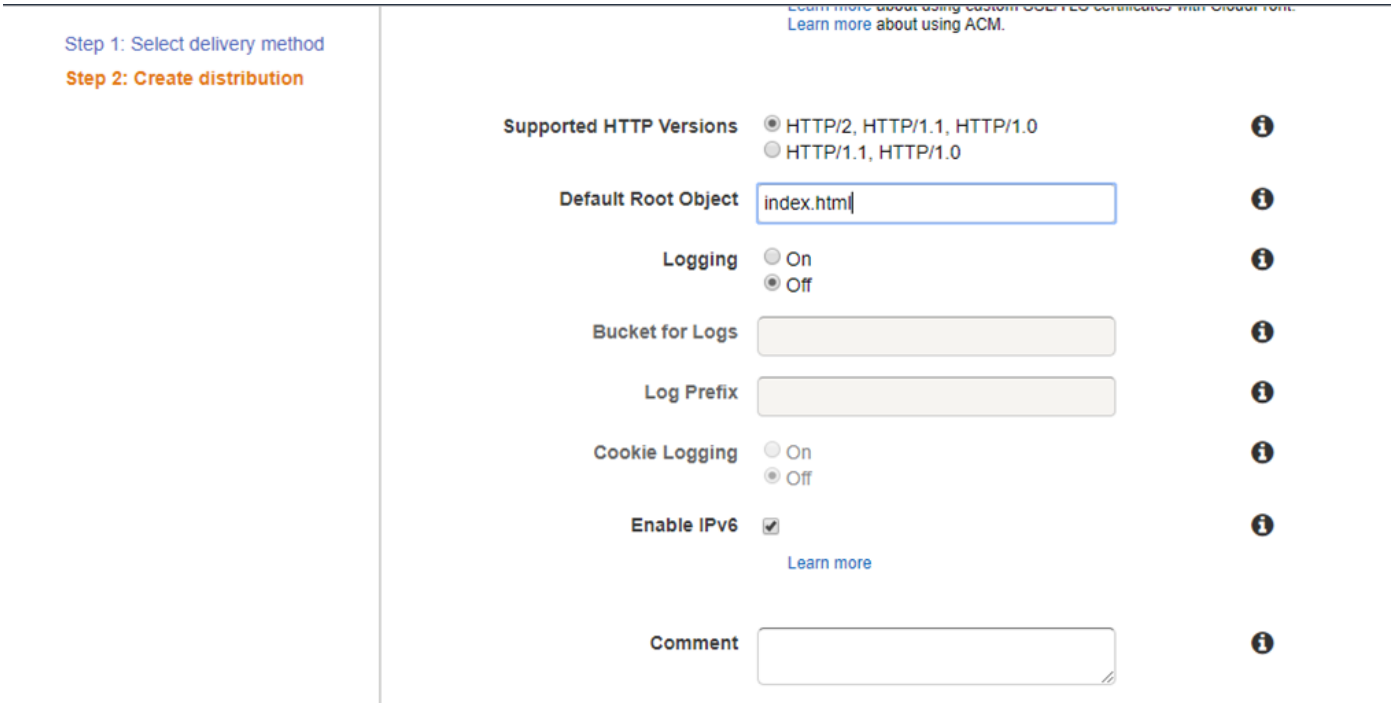
Under "Web" Section, Select "Get Started"



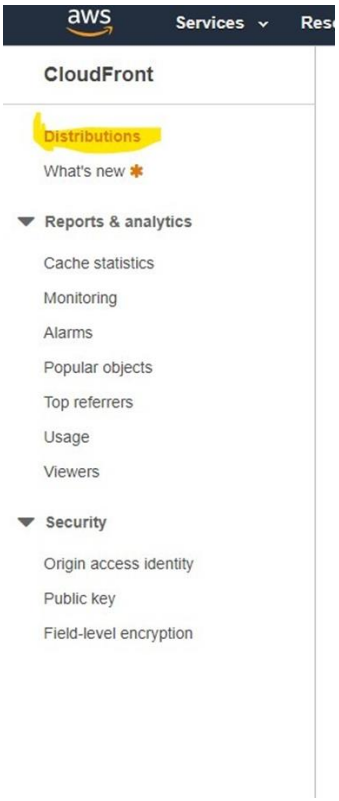
In the "Origin Domain Name" select "<BUCKET_NAME>.s3.amazonaws.com"



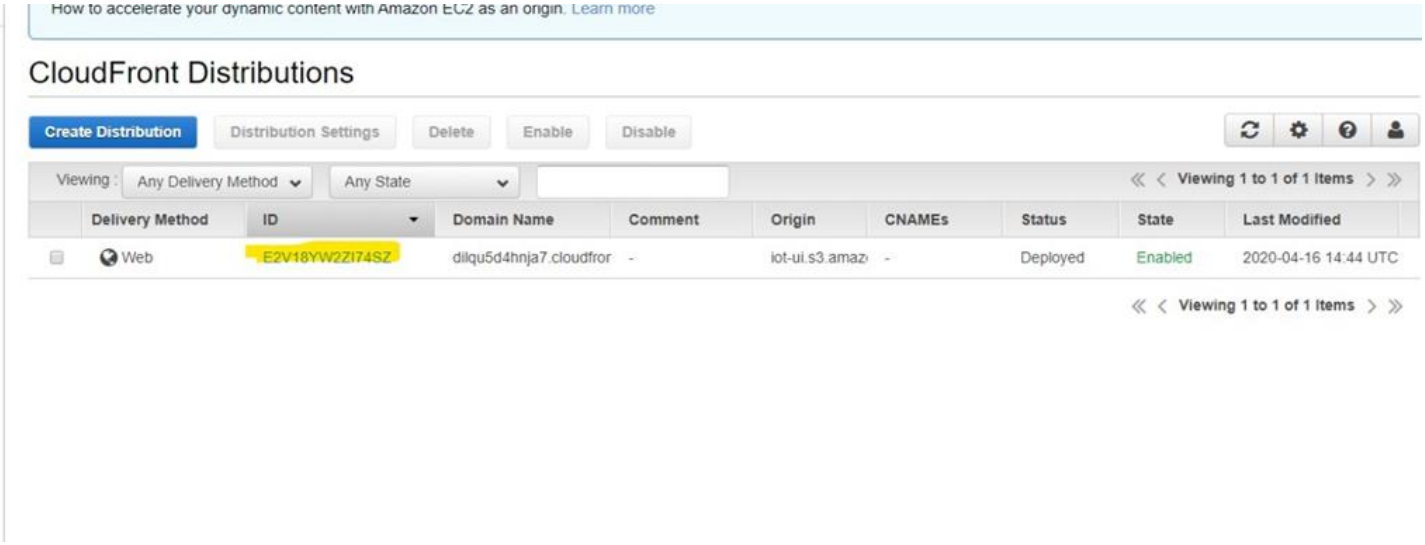
Scroll down, for "Default Root Object" enter "index.html"



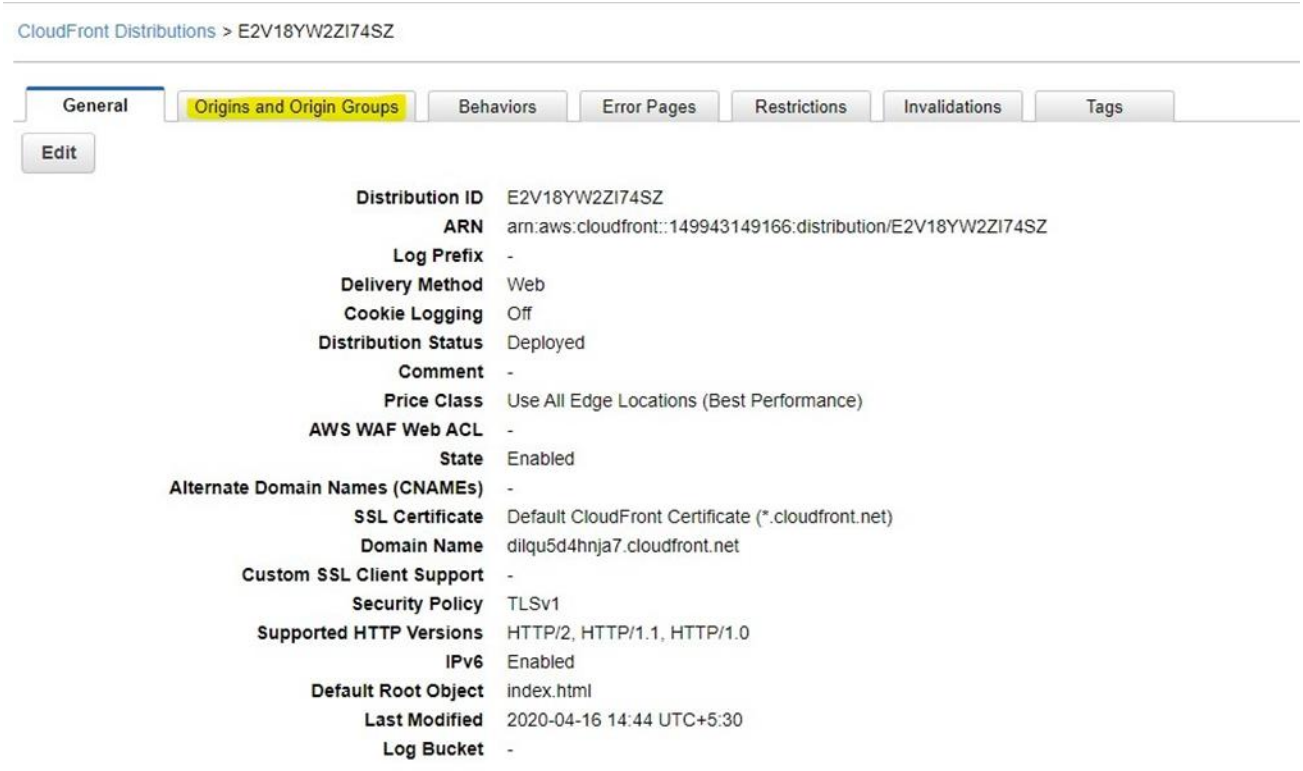
Click on "Create distribution"



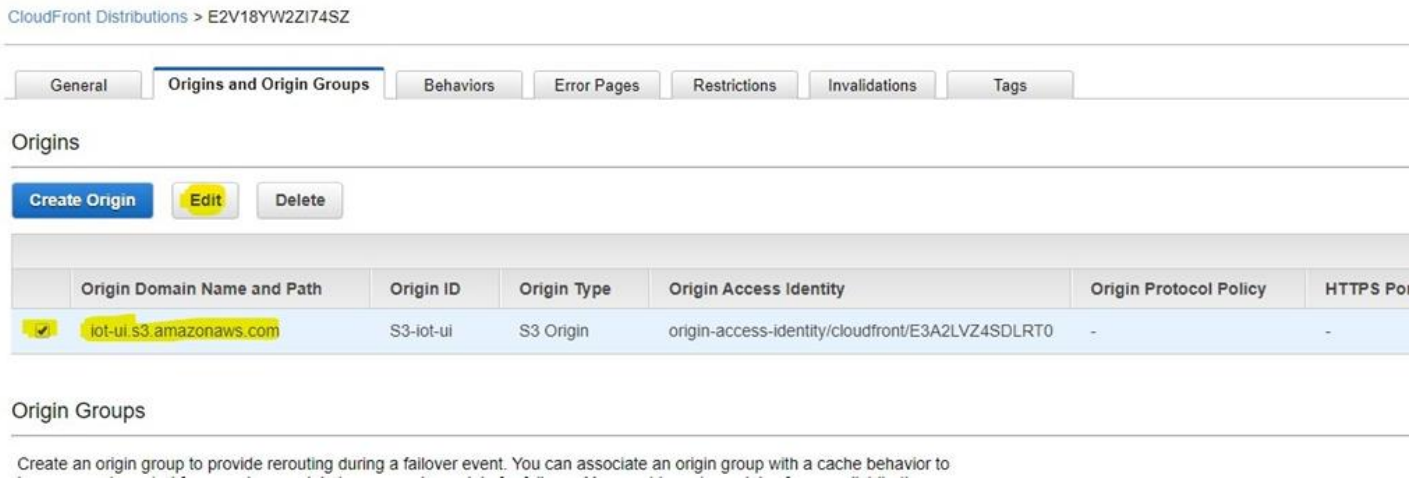
Select the distribution which created last step and Select "Distribution Settings"



Select "Origin and Origin Groups"



Under "Origin" section, select the origin domain name and select "Edit"

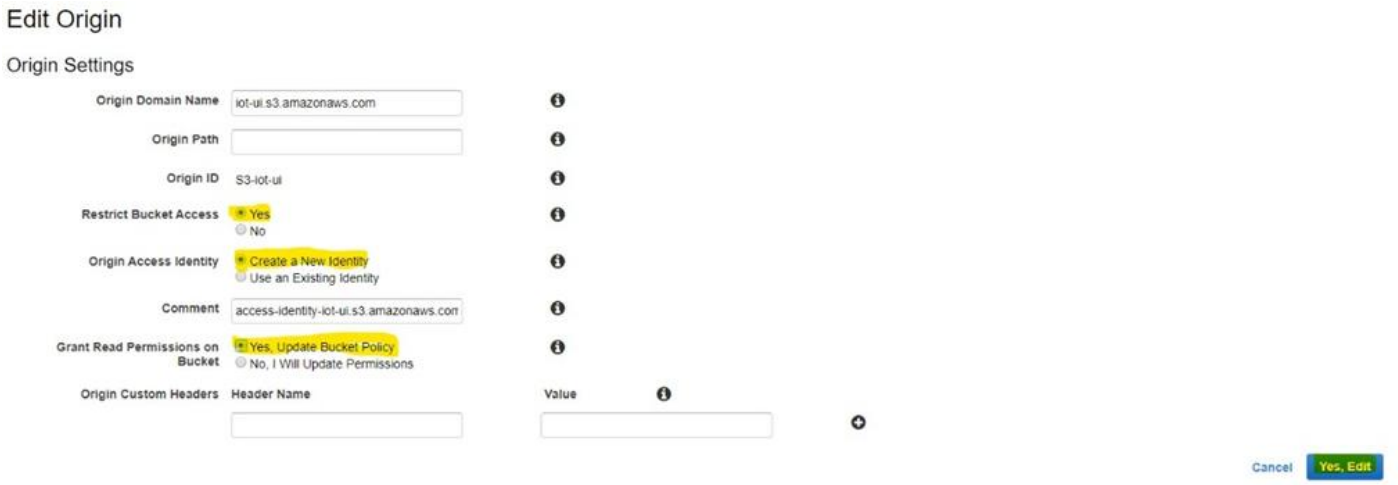


10. Change the "Restrict Bucket Access" to "Yes"

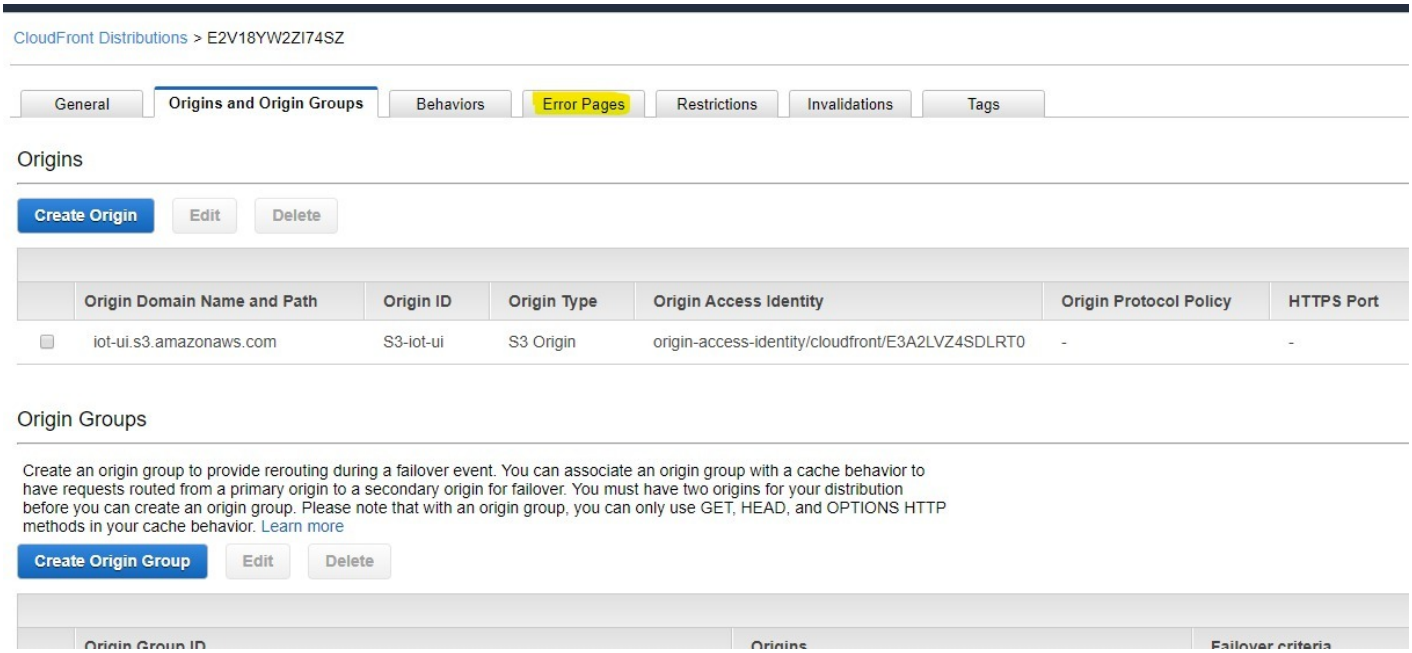
11. Change the Origin Access Identities “Create New Identity"

12. Change the Grant Read Permission on Bucket to "Yes Update Bucket Policy"

13. Select "Yes, Edit"



14. Click on "Error Pages" on header menu



15. Select "Create custom error response"

16. Select "Error code" -> 400 Bad request,

17. Select "Yes" for customize error response

18. For Response Page path enter "/index.html"

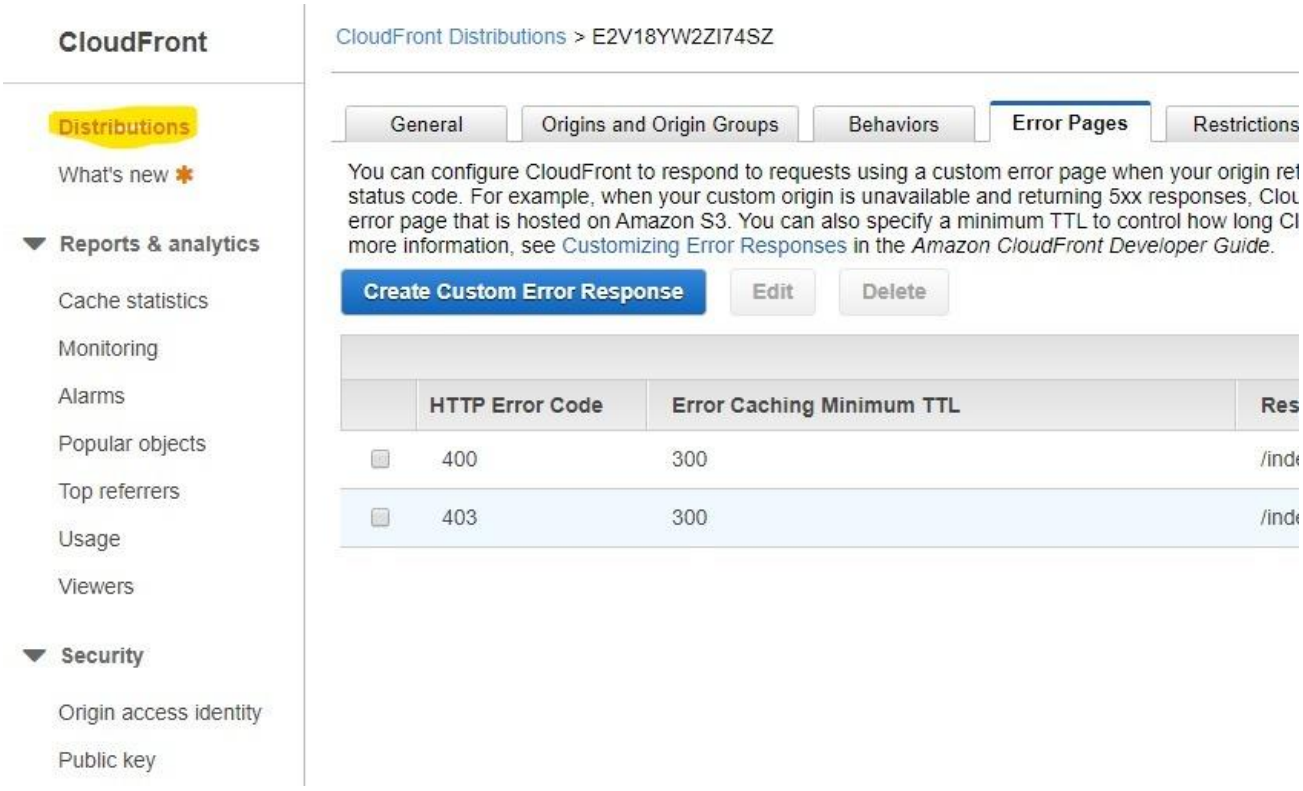
19. Change the HTTP Response code to "200: OK"

20. Click on "Create"

21. Repeat step 15,16,17,18,19, In step 16 select 403 as error code

22. Click on "Create"

23. Click "Distribution" on Left menu



24. Use the domain name mentioned in the created distribution to access the website.

ReadMe-UI.txt

IoT UI:

Reference Link used for theme:

https://www.akveo.com/ngx-admin/pages/dashboard?theme=cosmic

https://github.com/akveo/ngx-admin

Framework and theme is from the above url mentioned.

Here are the following customized files we used:

src/app/

* app.module.js

-> contains the components and plugins using in the component. It also contails the nested modules

* app-routing.module.js

-> contains the routing urls of the components like login, dashboard etc

src/app/service/

* rest-server.service.js

-> contains the rest services to be called to backend server uring http. all the methods will be defined here

src/app/auth/

* login

-> contains the login page component and html

* reset-password

-> contains the forgot password functionality component and html files

src/app/pages/

-> this folder contains all the components related to the customer and admin like dashboard, customers, register items, view alerts etc

* pages-routing.module.ts

-> this file contains all the routing path's defined which comes for the admin and customers

* pages.module.ts

-> here we declare and define the components, providers, and the additional plugins we use

Appendix -10: Project Contributions of Team Members

<u>Group Members</u>:

- Shuaib Mohammed, ID 201572070
- Khalid Nouh, ID 201458120
- Aymen Somily, ID 201218060

Phases	Project Tasks		Contributor Name
I. Project Proposal	Problem definition and Motivation.		Group
	Requirements and Specifications.		
	Approach.		
II. System Design	System's behavior: Use-cases and Activity Diagrams.		Group
	System's Specifications.		
	System's Architecture.		
III. Components Design	**1. Sensor Node**		Shuaib
	2. IoT Gateway		Shuaib & Khaled
	3. AWS Cloud	**Configuration & MySQL Database**	Khaled
		Lambda Function	Shuaib
	4. Web Dashboard Application		Shuaib
	5. Mobile App		Aymen
IV. Integration and Testing	Final Prototype		Group
	Testing		
V. Report Preparation	Contribution-wise report section drafts.		Group
	Final report organization, proof-reading and formatting..		Shuaib