Covid Room Monitor

Project Engineering

Year 4

Stephen O'Malley

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Galway-Mayo Institute of Technology

2021/2022

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

____

Acknowledgements

I would like to thank Brain O'Shea for his help with the Full Stack Side of the project.

Table of Contents

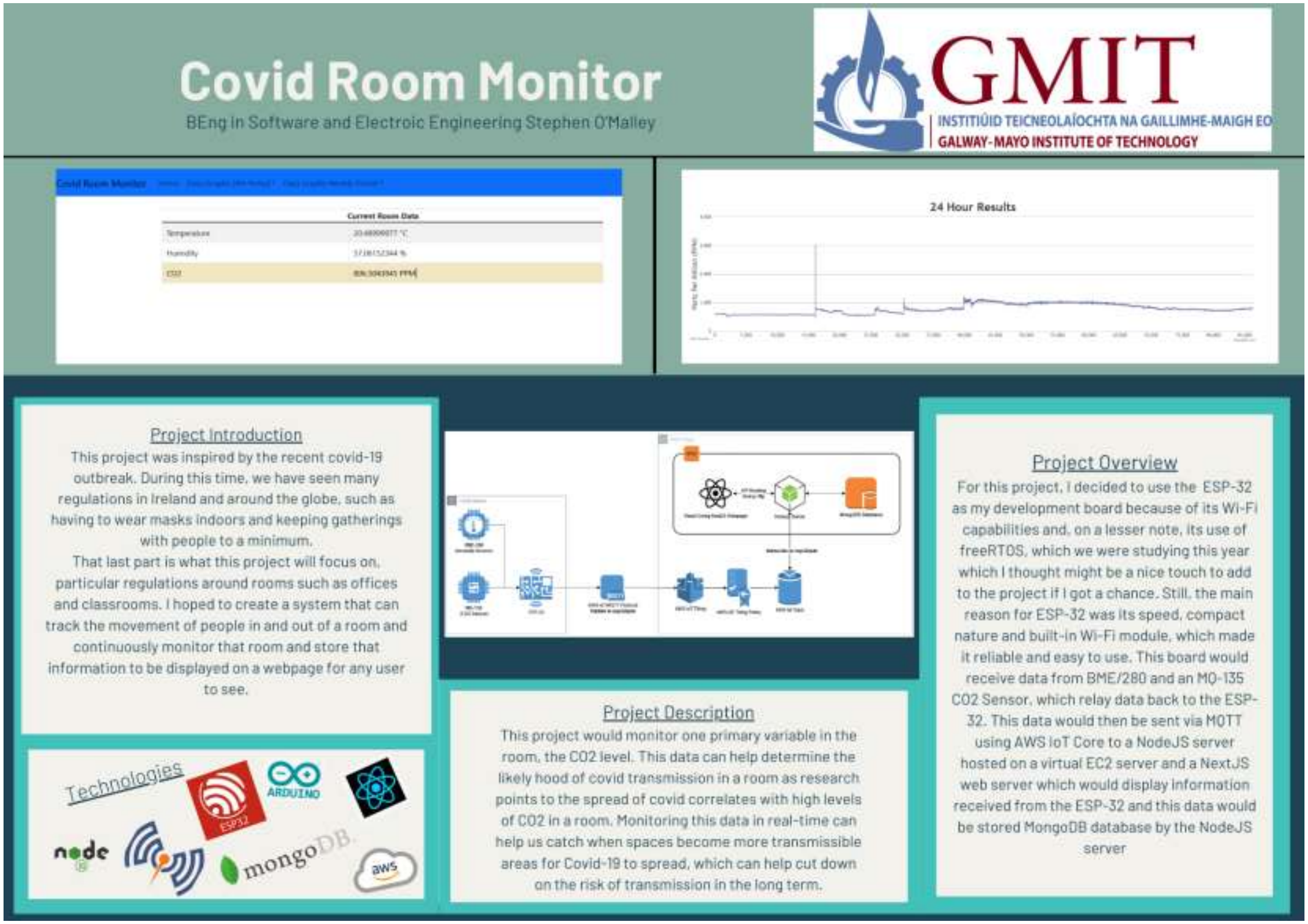
1 Summary

This project aims to create a Covid-19 monitor system using the premise of the correlation between CO2 of the spread of Covid in confined spaces. This states that the more people you have in a confined space, the higher the CO2 level will be, giving you a greater chance to transmit or get infected as there is less air curation in the room, allowing this more significant level of CO2 in the space. To accomplish this goal of this project, we would need three key features. Firstly, we would need a way of tracking the CO2 level in a room; Secondly, we would need a way of transmitting that data to a database to store the readings for later use in tracking the CO2 levels in the room and lastly, we would need to have a way of displaying the data in a format that would be easy of the user to interpret to make judgements on the CO2 level in the room.

I wanted to approach this project to have a bit of each area of software and electronic engineering, such as having an embedded system communicating with a Full-Stack web server. I accomplished this by using an ESP-32 that sent information via MQTT to a Full Stack server database hosted on the AWS Cloud.

I started work on the ESP-32 getting an MQTT connection working with the Mosquitto MQTT Broker. Then, I moved onto the project's backend using NodeJS to subscribe to the data sent to Mosquitto, which entered it into the MongoDB database. Then the NextJS front-end could access this data through an API route using the NodeJS server as a middleman; Then, this data would be displayed on the front-end server using graphs and tables to display the data collections. These were my project's primary goals and accomplishments; the only major problem was my plan to use an RFID Module to track people who entered and exited the room, but I was. Unfortunately, I could not do that as my device order did not arrive in time for me to start the work on the device and integrate it into the project.

2 Poster



Covid Room Monitor

BEng in Software and Electroic Engineering Stephen O'Malley

GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Covid Room Monitor

Current Room Data	
Temperature	20.68000077 °C
Humidity	57.18152344 %
CO2	806.5043945 PPM

24 Hour Results

Project Introduction

This project was inspired by the recent covid-19 outbreak. During this time, we have seen many regulations in Ireland and around the globe, such as having to wear masks indoors and keeping gatherings with people to a minimum.
That last part is what this project will focus on, particular regulations around rooms such as offices and classrooms. I hoped to create a system that can track the movement of people in and out of a room and continuously monitor that room and store that information to be displayed on a webpage for any user to see.

Technologies

Project Description

This project would monitor one primary variable in the room, the CO2 level. This data can help determine the likely hood of covid transmission in a room as research points to the spread of covid correlates with high levels of CO2 in a room. Monitoring this data in real-time can help us catch when spaces become more transmissible areas for Covid-19 to spread, which can help cut down on the risk of transmission in the long term.

Project Overview

For this project, I decided to use the ESP-32 as my development board because of its Wi-Fi capabilities and, on a lesser note, its use of freeRTOS, which we were studying this year which I thought might be a nice touch to add to the project if I got a chance. Still, the main reason for ESP-32 was its speed, compact nature and built-in Wi-Fi module, which made it reliable and easy to use. This board would receive data from BME/280 and an MQ-135 CO2 Sensor, which relay data back to the ESP-32. This data would then be sent via MQTT using AWS IoT Core to a NodeJS server hosted on a virtual EC2 server and a NextJS web server which would display information received from the ESP-32 and this data would be stored MongoDB database by the NodeJS server

3 Introduction

While I was on placement, we started to see workers return to the workplace. I gained a lot of firsthand experience with how management implemented these rules in the work environment. Similarly, I had the same experience when I went back to college for my 4th year and final year. From these experiences, I noticed that a lot of systems put in place were static limits for rooms or having windows always open to keep the air in the room fresh, so for my project, I had the idea to make a somewhat dynamic system to monitor to room to measure the CO2 levels using this to make sure that the room was within the covid-19 regulation guidelines.

To achieve this, I planned to monitor the CO2 levels in the room, which has been linked to the increased spread of Covid-19 [1] could continually monitor a room's CO2 levels from a web page that would store the data can be observed over long periods.

4 Background

Over the last two years, we have seen and up presented a change in our society that has made us rethink some of our fundamental values in education, work, and, most of all, our social lives with the outbreak of a deadly virus known as covid-19. Spreading from China around the world in a matter of months, Covid-19 brought the world to a standstill as it closed all but the essential places of work such as shops, pharmacies, and hospitals; people isolated themselves from friends and family to slow the spread of the virus.

Workers and students began to work from home, and for a time that that was able to be done, but as the pandemic continued, people began to realize that they would have to start going back to work and school. Even after a vaccine was released, it would still take some time before we all were safe to remove masks. So, rules were put in place to limit the number of persons allowed to occupy a room under a specific list of conditions such as size and ventilation capability.

5 Project Architecture

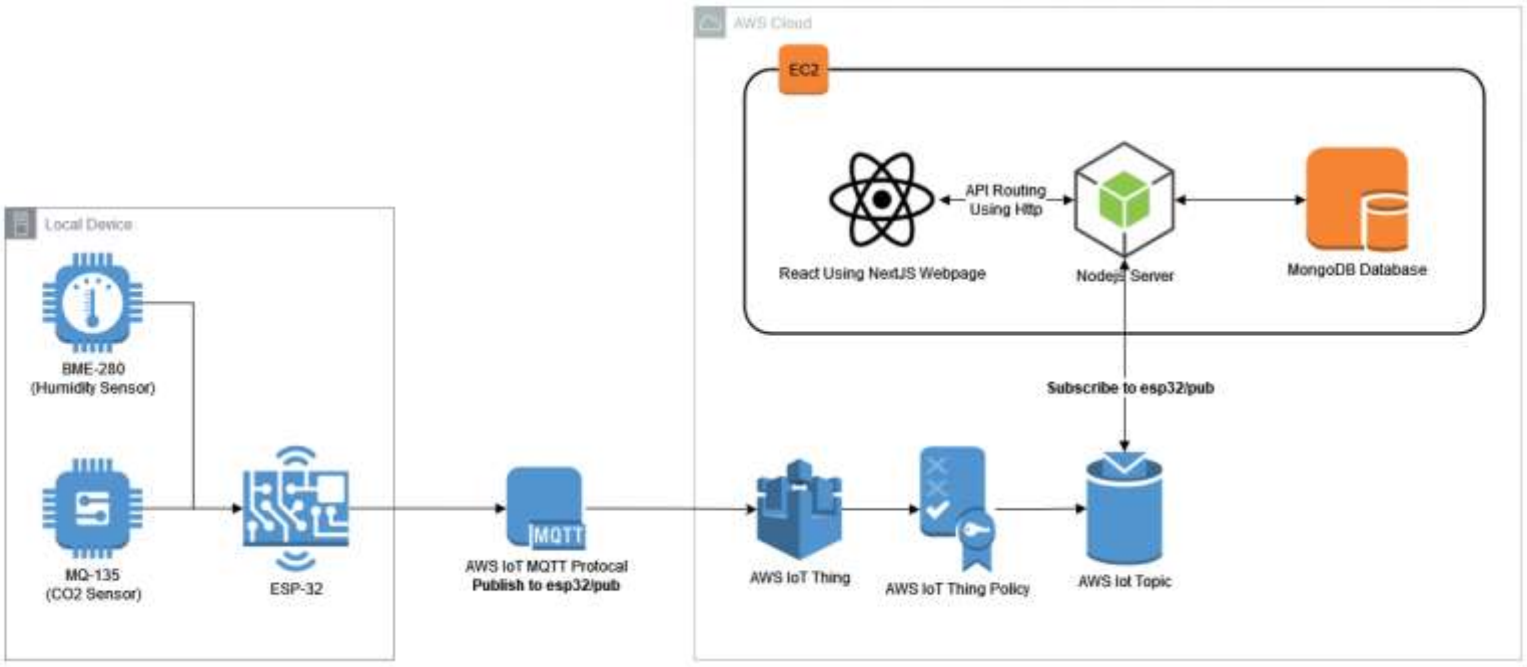


Figure 5-1 Architecture Diagram

6 Project Plan

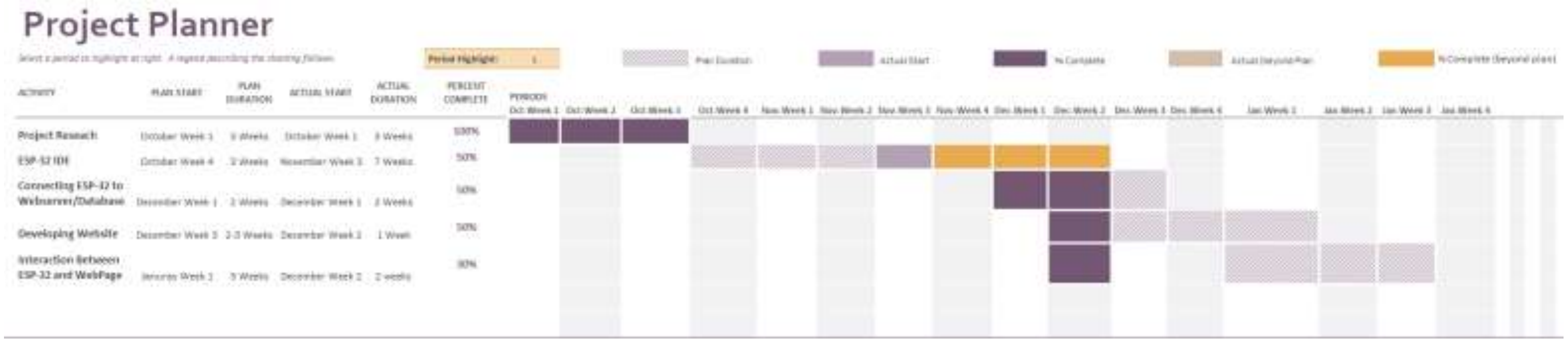
6.1 The First Round

The first plan that I generated for my project was during the project proposal phase around Halloween. In this plan, I gave a very optimistic view of what I would include in my project and how long it would take to complete each part. One notable thing about this timeline is the inclusion of the Android App I was a thing of creating to work alongside the RFID sensor. Still, I cut the app soon after I made this because it was clear that the web development would be challenging enough without including this app.

Month	Oct-21	Nov-21	Dec-21	Jan-22	Feb-22	Mar-22	Apr-22
Week 1	Project Research	Continue project test work with ESP-32 IDE.	Start trying to get an ESP-32 to sever connection running.	Get interaction between the server and the peripherals working.	Begin data storage and sorting work.	Bug Fixing and Improvements to existing code.	Start Final Report and Poster.
Week 2	Project Research	Continue project test work with ESP-32 IDE.	Start trying to get an ESP-32 to sever connection running.	Get interaction between the server and the peripherals working.	Begin development of an Android NFC App.	Bug Fixing and Improvements to existing code.	Create Demo Video.
Week 3	Submit Project Proposal.	Test ESP-32 Using MQTT.	Develop a web interface for the web server.	Try and have the overall interaction with the sever and ESP working.	Continue work on Android App.	Bug Fixing and Improvements to existing code.	Project Demonstration.
Week 4	Start Project work with ESP-32 IDE.	Test ESP-32 using MQTT.	Develop a web interface for the web server.	Begin data storage and sorting work.	Work on the Android App interaction with the web server.	Start Final Report and Poster.	*Project Finished*

6.2 The Christmas Demonstration

The Second Plan that was generated was made at Christmas for our demos. I developed this plan using a Microsoft Excel template. In this plan, you can see that it did take me a while to start appropriately working with my ESP-32 as it took time for me to receive the micro-controller. Still, I did manage to create some of the web development earlier than expected, but I also wasn't able to start my work with MQTT for reasons that will be detailed later.


Project Planner

Period Highlight: 1

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Project Research	October Week 1	3 Weeks	October Week 1	3 Weeks	100%
ESP-32 IDE	October Week 4	3 Weeks	November Week 3	7 Weeks	50%
Connecting ESP-32 to Webserver/Database	December Week 1	2 Weeks	December Week 1	2 Weeks	50%
Developing Website	December Week 3	2-3 Weeks	December Week 2	1 Week	50%
Interaction Between ESP-32 and WebPage	January Week 1	3 Weeks	December Week 2	2 weeks	30%

6.3 The Final Sprint

After Christmas, we started recording our team meetings which were done online. To keep better track of teamwork in these meetings by recording them, we were recommended to use task tracking software to better show and keep track of work been done and completed. In one of the first meetings, one of my classmates recommended Jira to me, which I was planning on using as it was what I was using while I was on placement, so I already had some familiarity with it. You can view my tracking progress from the videos recorded and submitted on the OneNote file for Project Engineering.

7 The First Piece: ESP-32

7.1 Research

The ESP-32 was the first piece of my project that I began with was the ESP32-DevKitC V4 [2] equipped with the ESP-32-WROOM-32D module on the development board. I had chosen this board because of the same logic of why these boards are so popular for development, which is the integrated Wi-Fi module. This, along with the high clock frequency of the board CPU, allows the ESP-32s to perform the most demanding tasks with speed and efficiency [3].

While researching my project, I investigated using the ESP-IDF [4] through Visual Studio Code using the ESP-IDF Explorer Extension [5]. At the time I was using this, there was a bug in the step of IDE config, which I did manage to fix [6], but during this research, I discovered that the difficulty of coding an embedded system using FreeRTOS, which is the OS that the ESP-32 runs on, which I did have much experience with let alone using it for Wi-Fi communication along with having to learn the web development side of my project I decided I would go with Arduino for development and come back to the ESP-IDF if I had the time to at the end of my project

7.2 Arduino ESP-32 Configuration

To begin using the ESP-32 with the Arduino IDE, we first must install the ESP32 dependences in Arduino to configure ESP-32 for development on the IDE. Once we have installed these dependencies, we must select the board type we are using, i.e., "ESP32-DevKit" once this has been chosen, we can now begin programming our ESP-32 with the Arduino IDE [7]

7.3 Bugs Encountered

The only bug that I encountered while working with the ESP-32 was a timeout error that stopped the board from going into flashing/uploading mode automatically when uploading a new code to the board, and this can be fixed by connecting a 10uF capacitor between the EN Pin and the GND Pin on the board to resolve the issue [8].

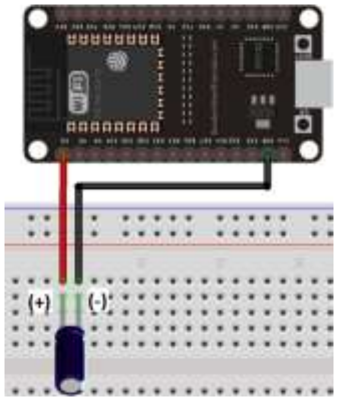


Figure 7-1 Capacitor Pinout Setup

7.4 Pinout Diagram of ESP32-DevkitC

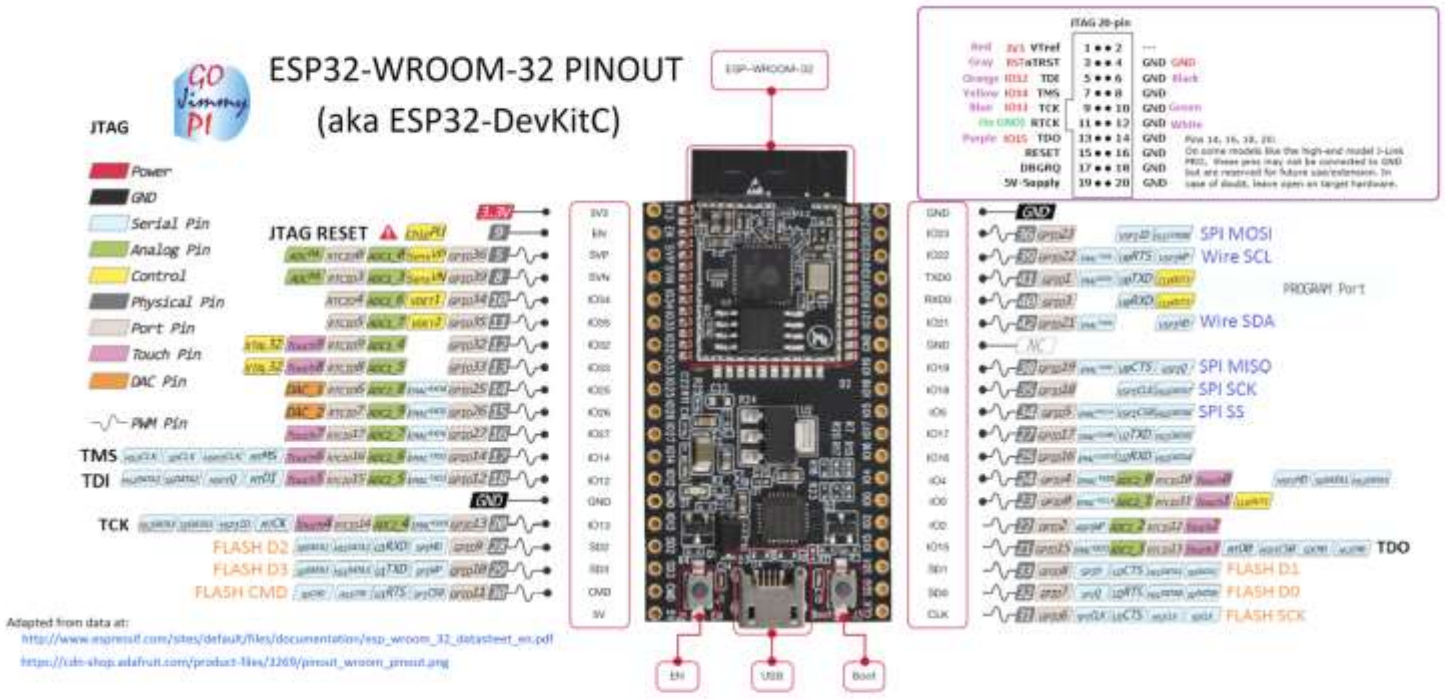


Figure 7-2 ESP-32 Pinout

8 BME-280

8.1 Research

The first sensor I configured for my project was the BME-280, a humidity sensor measuring relative humidity, barometric pressure, and ambient temperature [9]. The version of the sensor that I was using was the Adafruit BME-280 module used for IOT development. This module uses I2C (**Inter-Integrated Circuit**) for commutation with the board. This means you can have multiple devices on the same port that transfer data, and they will not overwrite the current data on the SDA (**Serial Data**) line as they are synced by the clock signal on the SCL (**Serial Clock**) line.

8.2 Set-up With Arduino

To begin working with the BME-280 I first needed to install the Adafruit Unified Sensor Library in Arduino and the Adafruit BME-280 library using the Arduino Library Manager. Once this was complete, I opened the BME-280 code example provided by the library and connected my BME-280 to the ESP-32 [10]. To connect this device to our ESP-32, we need to find the SCL and SDA Pin on the ESP-32, which from the looking at *Figure 7-2 ESP-32 Pinout* Diagram the are on Pins IO22 (SCL) and IO21 (SDA) along with connecting it to 3.3V and ground.

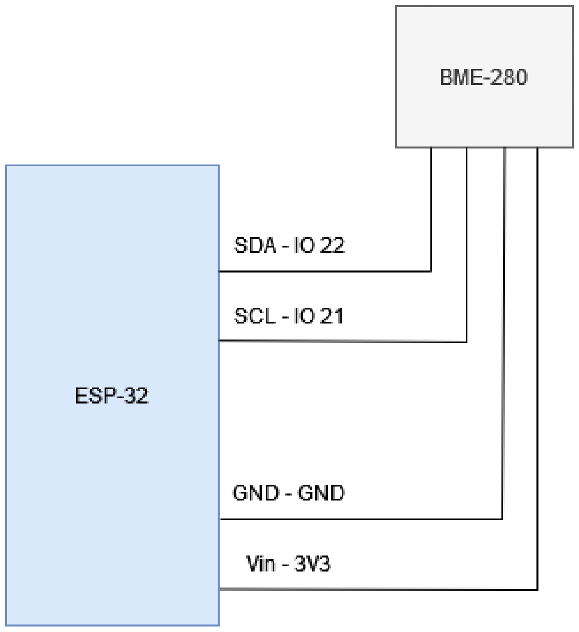


Figure 8-1 BME-280 Setup

9 MQ-135

9.1 Research

The Second Sensor that I used in this project was the MQ-135 Gas sensor which can measure the concentration of CO2, CO, Alcohol, NH4, Toluene and Acetone in Air [11]. While researching this, I found that the MQ-135 module had a defect where the RL (Load Resistor) was a 1k instead of 10 to 22k, which is the Recommend resistor value to get appropriate readings from the sensor [12].

While researching the MQ-135 I also found an open-source Library (**MQSensorsLib**) written for the MQ series sensors to make them easier to read data from, giving more accurate values than if you were possibly writing it yourself [13].

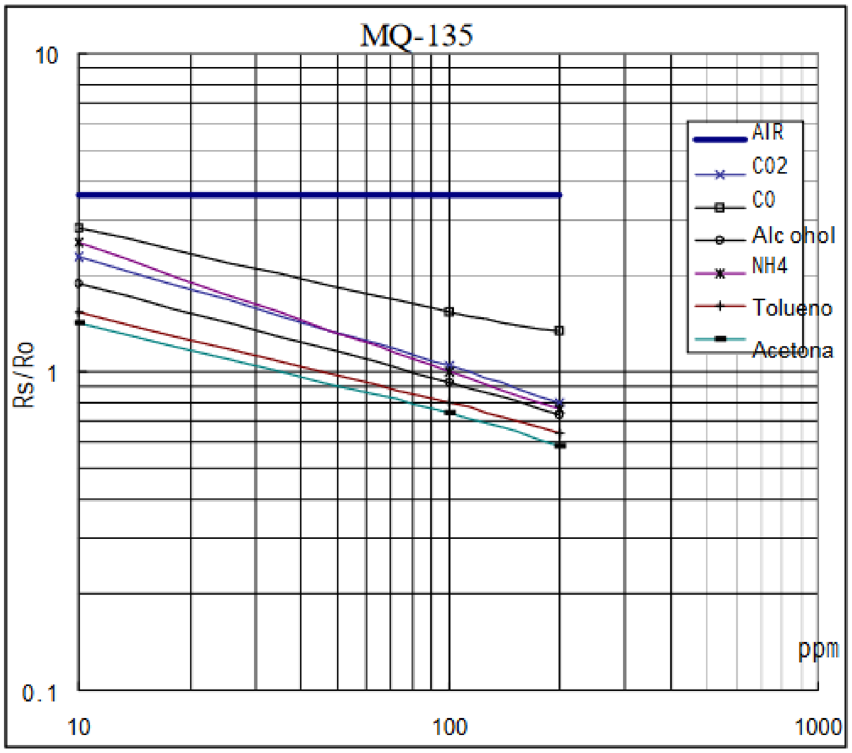


Figure 9-1 Sensitivity Characteristics of the MQ-135

]

9.2 Set-up With Arduino

First, we must connect the MQ-135 to an analogue pin on the ESP-32, which in this case is pin 34. We also must divide down the signal from the MQ-135 as it requires a 5V supply, but the ESP-32 only accepts 3.3V analogue signals [13]; this is complete using a series of resistors. After this is complete, we preheat the MQ-135 for about 24 hours to burn in the MQ-135 sensor in a cleanroom. While burning in the MQ-135 sensor, I ran some code that would get the average R0 value. We use this for getting our gas ratio which is found from getting the average value of R0 in a 24-hour period which for me was **16.6**. We can set up our sensor using this value to find the CO2 concentration in PPM (**Parts Per Million**) [14]. Then we use the example code from the MQSensorsLib to configure our MQ-135 for reading CO2.

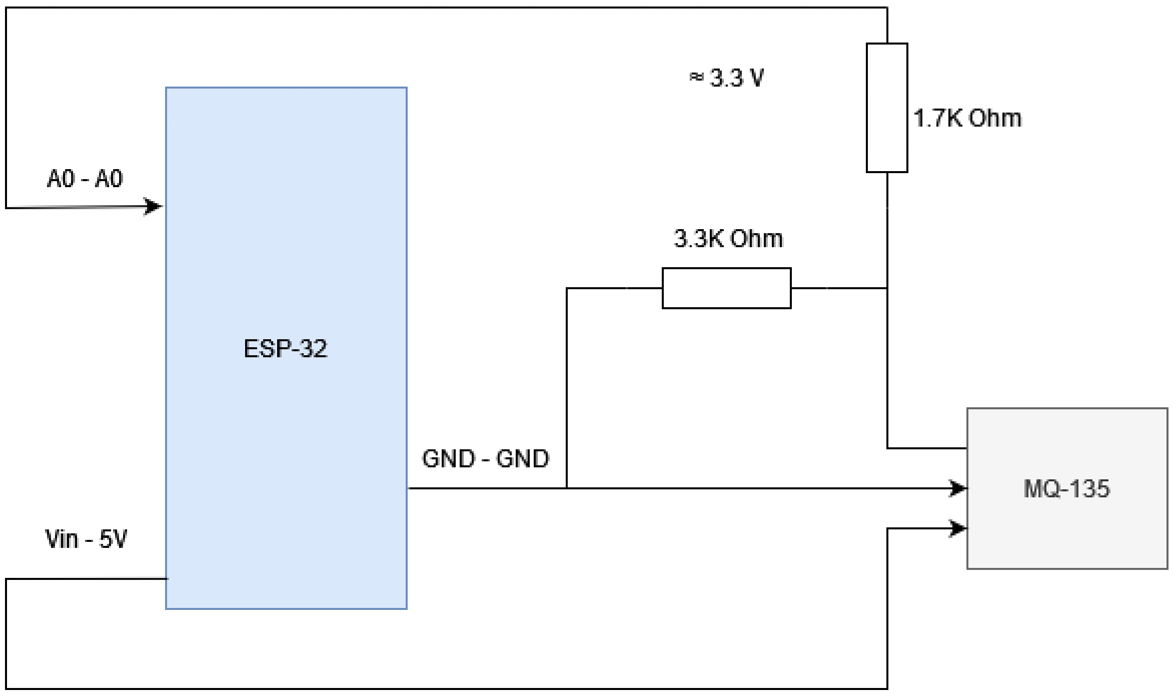


Figure 9-2 MQ-135 Setup

10 MQ Telemetry Transport

10.1 Research

MQTT is an OASIS (**Open Artwork System Interchange Standard**) standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport, which is ideal for connecting remote devices with a small code footprint and minimal network bandwidth [15]. Before using the AWS IoT system, I used mqtt.org to look at the different types of MQTT brokers. Before using AWS, I decided to get a client-side version of my project working, and for this, the broker I decided on was Mosquitto, an open-source lightweight message broker [16].

10.2 Configuring Mosquitto

After installing mosquitto, we must create a user with a password. This password adds a level of security to the broker as the default behaver of mosquitto does not allow you to publish or subscribe to the broker anonymously. After the user was created, I next began to edit the mosquitto.conf file, which controls the configuration of the mosquitto broker. The First thing we must edit in the file is under "general configuration" we enable the "per listener" setting to be true. This allows us to change the running configuration of mosquitto. Secondly, we must enable and link the ACL File, which creates the topics that the broker acknowledges and sets which users have access to them or not. Thirdly we must link our password file for the users we want to access the broker. Lastly, we must disable "allow anonymous" so that only predefined users can access the broker [17]. Then to test the configuration, we open the command prompt in the mosquitto directory and launch mosquitto using the .conf file setting [18]. If we have set the config file correctly, it should launch and await to receive and send information.

10.3 Set-up With Arduino

To use MQTT commination with Arduino, you must use an MQTT library. The one I ended up selecting from my research was the MQTTPubSubClient [19] library because of the examples and documentation were excellent. Along with the fact that the library has built-in AWS Connection protocols, which would be handy later as I wouldn't have to switch libraries down the line. Using the example included with the MQTTPubSubClient library, it is quick and easy to set up the MQTT communication to the broker run on my PC.

11 Full Stack

11.1 Research

The research for the web development side of the project was more involved in bugs and problems encountered while working on the Full Stack. Compared to the embedded side of the project, this is the section I got the most help on for development from my lecturer Brian O'Shea. I got this help because it was something that we had only really begun this year in college and for something that is just such a massive topic in and of itself. This was especially true when it came to communication between the NodeJS server back and the react Nextjs front end, which was the part that took me a while to wrap my head around its inner workings.

11.2 NodeJS

Nodejs is an asynchronous event-driven JavaScript runtime designed to build scalable network applications[20]. Node is based on javascript, the standard base for programming with web development, whether on the back or not front end. Before any development of web technologies are started Nodejs must be installed.

11.3 React.js

React is an open-source front-end JavaScript Library for building user interfaces. However, react is only concerned with state management and rendering that state to the DOM (**Document Object Model**). Usually, when a react application is created, it can require additional libraries for routing, which leads us to our next section [21].

11.4 Next.js

Next is an open-source web development framework that enables React web application functionalities such as API routing and server-side rendering [22]. Using Next.js allows the creation of a NodeJS backend, which can route data to and from the web page to the Nodejs server, which can also be pulled to and from a database to store said information.

11.5 Database

MongoDB is an open-source NoSQL database that stores its data in JSON style format [23], considered the industry standard for NoSQL databases. To access MongoDB in the Node.js Server, you either must use the generic MongoDB library or express. We generally express this as it is a much more straightforward schema-based solution to model application data. I ended up using this as it is cleaner and easier to use than the Standard MongoDB library.

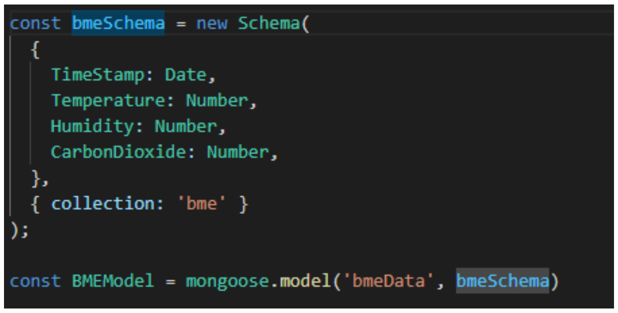
12 Constructing the Back

12.1 MQTT Operation

The First Step I took when building the Node Server was getting it connected to the MQTT broker to receive messages sent from the ESP-32 device. This was done using the MQTT library, which would connect to the host IP of the broker, which was active on port 1884, which is the port that was set in the MQTT config, along with using the username and password for authentication. Once the connection is established, the MQTT client will subscribe to the Data topics and publish the information on the Node server[24].

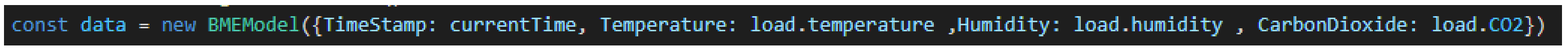
12.2 MongoDB & Mongoose

After the MQTT data is received on the server, we next need to store that data in the database. As was mentioned earlier, we will use MongoDB as the database and mongoose to insert that data into the MongoDB database. Firstly we connect to the local database using this URL "**mongodb://localhost:27017/NameOfDatabase**" the default port for MongoDB is 27017 [25]. Then we change the "NameOfDatabase" par to whatever you want to name your database. Lastly, we create Mongoose Schema Models, which let us insert data into the MongoDB database following the data format of the insert set in the Schema.



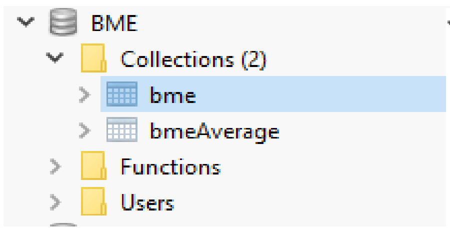
```
const bmeSchema = new Schema(
  {
    TimeStamp: Date,
    Temperature: Number,
    Humidity: Number,
    CarbonDioxide: Number,
  },
  { collection: 'bme' }
);

const BMEModel = mongoose.model('bmeData', bmeSchema)
```

Figure 12-1 Mongoose Schema



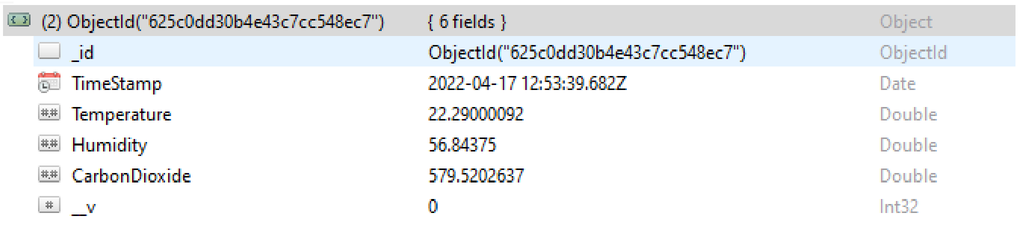
```
const data = new BMEModel({TimeStamp: currentTime, Temperature: load.temperature ,Humidity: load.humidity , CarbonDioxide: load.CO2})
```

Figure 12-2 Creating a Data entry based on the Model Format



```
BME
  Collections (2)
    bme
    bmeAverage
  Functions
  Users
```

Figure 12-3 BME Database

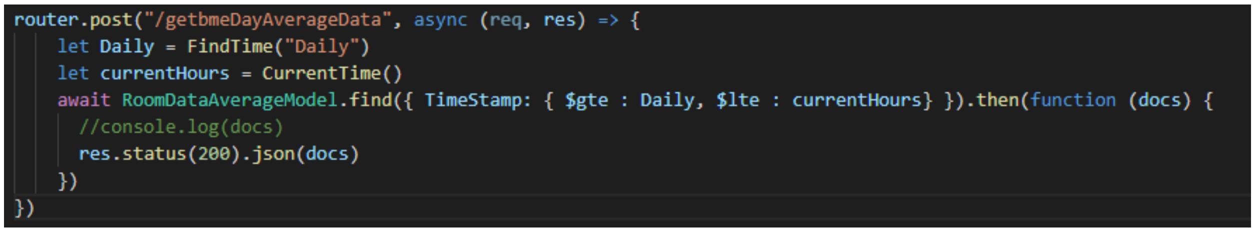


(2) ObjectId("625c0dd30b4e43c7cc548ec7")	{ 6 fields }	Object
_id	ObjectId("625c0dd30b4e43c7cc548ec7")	ObjectId
TimeStamp	2022-04-17 12:53:39.682Z	Date
Temperature	22.29000092	Double
Humidity	56.84375	Double
CarbonDioxide	579.5202637	Double
__v	0	Int32

Figure 12-4 Data Object in BMEModel Format in the Database

12.3 Express

Lastly, we move on to Express which is a framework that allows us to response to HTTP requests which are used for API routing of data from the back end to the front end [26]. We this Webpage by using the express-generator tool and entering "**npx express-generator --view=hbs**" to set up a basic Express application skeleton [27] after this is complete the code from the MQTT operation and mongoose are added to the project and tested before moving on to add the routing, once the test is complete we add the routing using "**express.Router()**" function to handle requests, we use to post or get information to and from the front-end when a stated http request is made from the front end, in this case information is send to the Front-end from the database using the post method and if found the data is send forward using the "**res.status(200).json(FileBeingSend)"** which means that data is send back with the success reference "**200**".



```
router.post("/getbmeDayAverageData", async (req, res) => {
    let Daily = FindTime("Daily")
    let currentHours = CurrentTime()
    await RoomDataAverageModel.find({ TimeStamp: { $gte : Daily, $lte : currentHours} }).then(function (docs) {
      //console.log(docs)
      res.status(200).json(docs)
    })
})
```

Figure 12-5 Router Post

13 Decorating the Front

13.1 Routing with NextJS

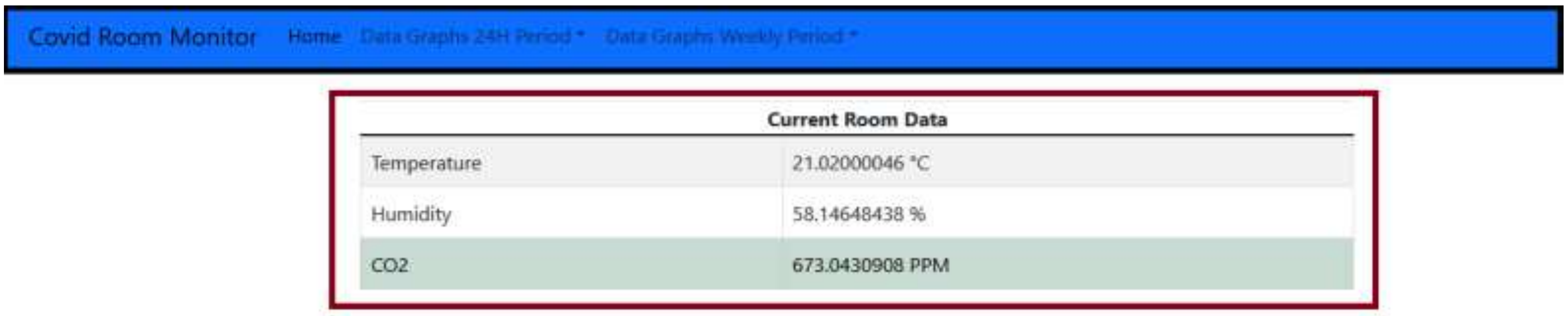
Routing using Next was the one thing I got the most help from Brain with as it was the one area I found pretty tricky, so I worked quite a lot from his examples and the NextJS Documentation [28]. To configure the API routes in NextJS, we have to put the API file in the pages/api. Any file placed in the pages/api folder would be treated as an API endpoint. Then we would use a context provider, which allows the ability to share values between components without having to prop-drill the components to past data. Inside the context provider, we set up the API route calls, which will feed the data to select components within the Webpage when called [29].

13.2 Pages

The pages folder contains the, as mentioned earlier, APIs routes for the NodeJS server, but any regular file put in the pages folder becomes a route for the front-end pages [30]. We also have two particular files that are placed in this folder. The _app, initializes the pages and will allow us to have a consistent layout between pages and inject context provider data into our page, and the _document will enable us to CSS in NextJS [31].

13.3 React-Bootstrap

React-Bootstrap is a UI component library that features many components such as the navbar and table. I have used to design the Navbar Display, which allows us to link other routes to pages on the entire web page and the table that displays current readings from the ESP-32 on the home page.



Current Room Data	
Temperature	21.02000046 °C
Humidity	58.14648438 %
CO2	673.0430908 PPM

Figure 13-1 Website Home Navbar (Black) & Table (Red)

13.4 CanvasJS

A large part of this project is displaying the data gathered by the ESP-32 and then showing that data on a web page. To display this data, I decided to use graphs, so I ended up using the most popular and extensive graphing library, CanvasJS [32]. Even with the difficulty of using CanvasJS in NextJS, it is still the best graphing library for displaying all kinds of information in different graphing formats. The CanvasJS library is very straight forward and easy to use in react but when using it NextJS, you can encounter some problems such as disabling the SSR (**Server Side Rendering**). We do this by importing the graphs dynamically to our page component to disable SSR. This is caused by the design of the CanvasJS library, as the graphs have to be rendered on the client-side of the front-end server [33], but once this problem is solved, CanvasJS will work just fine on NextJS.

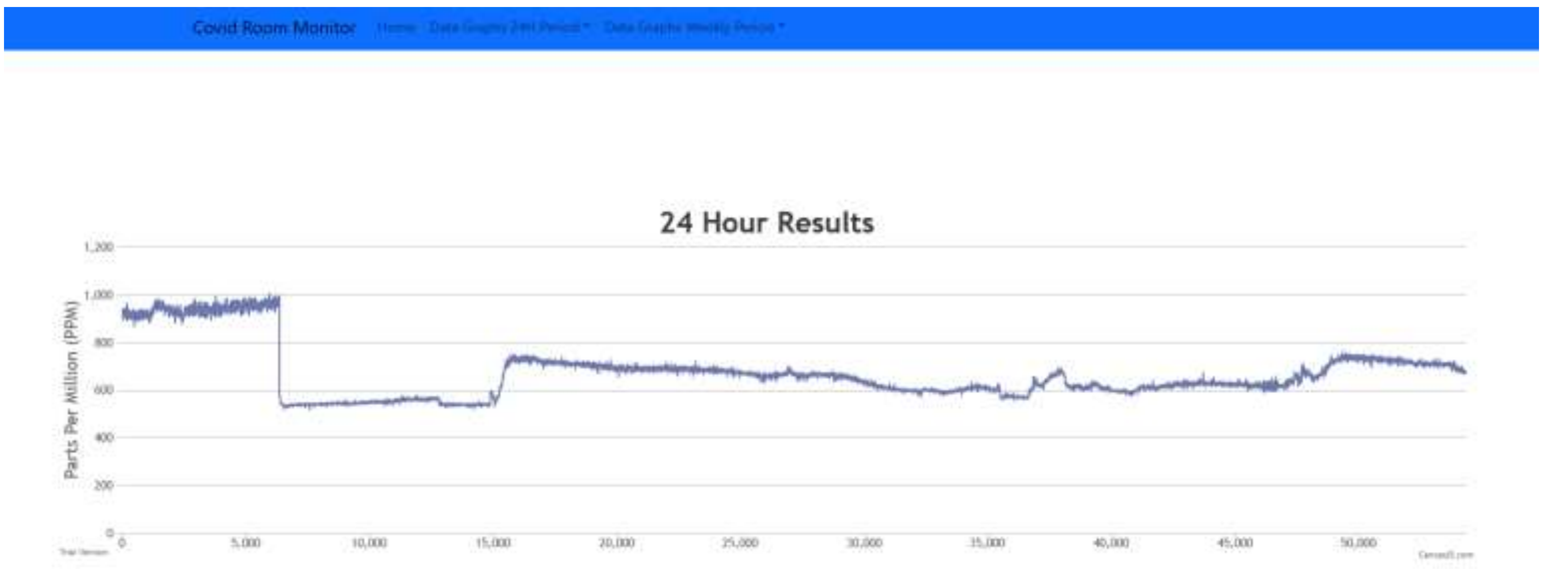


Figure 13-2 CanvasJS Multi Point Graph

14 Amazon Web Services

Now that the backend & front-end are completed, we can begin to migrate the Full Stack to the cloud to be accessible from anywhere and run 24/7 without having the local computer running the stack.

14.1 IoT Core

14.1.1 Setting Up the Things

The first step in getting the project step up on AWS is using the IoT Core to receive the MQTT data from the ESP-32. This, as stated earlier, is why I selected the library that I did, as it allows easy migration over to MQTT. To start, we go to Manage/Things and create two things to represent the two devices we will be sending this data to and from; one will be named ESP-32, and the second will be called NodeJS. Then, we will need to move to Secure/Policies; under here, we will create a policy that allows specific MQTT communication on designated topics and configure it to our chosen topic name esp32/pub. Then, we attach this policy to both of our things. When they are attached, we are ready to test the MQTT communication. We can do this using the Test/MQTT test client to check if our policy is configured correctly [34].

14.1.2 Reconfiguring MQTT

After we have tested the MQTT policy to see if test information can be sent and received, we next need to send the data from the ESP-32 to the IoT Core. Sending data to the IoT core is relatively straightforward. We only have to change our client settings to use the AWS certificates provided by the IoT thing we created to connect to the created Things in the IoT core to provide authentication while connecting [19]. Secondly, we must also convert the data being sent to the server to a JSON format as the AWS MQTT IoT Core will only accept the information if it is in JSON format. We use ArduionJson [35] to achieve this, which is relatively straightforward. Once all that is complete, when we go to the Test client, we should see that our information is published from the ESP-32 device.

14.1.3 Using AWS IoT Device SDK V1

Lastly, we need to receive this information on our Node server and insert it into the MongoDB database. To do this, we use the AWS IoT device SDK; with this, we basically do the same thing that we did in Arduino, which is to use the certificates for the AWS IoT Thing to connect to the AWS IoT core to access the MQTT by subscribing to the topic that the ESP-32 is publishing to [36].

14.2 EC2

Once the IoT Core configuration is complete and data is being received on the Node Server, we can move on to the last part of the project set-up, moving the Full-Stack to the cloud. We will use an EC2 instance, a virtual cloud server, to host our Full-Stack.

14.2.1 Creating the Instance

We start creating the EC2 instance by going to the EC2 section under compute on AWS. Once we get to the menu, we can start by selecting the launch instance. Now that we are in the set-up, we leave the default settings as they are. The only thing we change is the instance type which is changed to a "t2.medium", which provides extra storage memory for the database. Once this set-up is complete, we launch at the instance to create it.

14.2.2 Install Dependencies

After the instance is created, connect to the instance. Once connected to the instance client, we must install the NodeJS [37] and then install MongoDB [38]. Once this is complete, we can move the stack over to the instance.

14.2.3 WinSCP

Once the EC2 instance is configured, the code must be moved to the EC2 instance. This is done by using an SFTP (**Secure File Transfer Protocol**) Client such as WinSCP to transfer the files to the EC2 server. This is accomplished by using our certificates for the EC2 are used to authenticate the connection of WinSCP to the instance for the secure file transfer. We move the code without the dependencies installed to a folder inside the EC2-user folder along with and .sh file to start the Full-Stack.

14.2.4 Finalizing

Once the WinSCP transfer is complete, go back to the instance and move into each of the folders for the Webpage and the NodeJS server and run "**npm install**" to install the dependencies for each. Now that we have installed the dependencies run the .sh script to start the Full-Stack, which will connect to the ESP-32 over AWS, displaying the received information on the Webpage's and saving it to the MongoDB database.

15 Ethics

While researching for this project, I kept thinking why no one had else come up with a similar idea of monitoring the CO2 levels in the room when a correlation was found between high CO2 levels and covid transmissibility. The apparent reason for that is that this is worth the cost of setting up as this isn't 100% concrete proof of covid spreading. It is more of an indication that it can spread if it is present within the environment. When you also compare that with the cost of setting up this equipment from scratch in hundreds of rooms, you see that equipment cost skyrockets. You start to understand the safety verse cost dilemma that comes into play as CO2 readings level of inconsistency might not be worth the time and money to set up to some, even though it could help keep the spread of covid down.

16 Conclusion

With the completion of this project, I feel that a demonstratable prototype system that shows the CO2 level of room gathered using an MQ-135 through and ESP-32, which uses MQTT to send information over Wi-Fi to the AWS IoT Core. This information is then inserted into a server-side MongoDB database using a NodeJS backend that routes the information to the front-end. It is then displayed on graphs and tables that contextualize the data and make it easier to understand. This Full Stack is then located on an EC2 virtual server instance, which allows us to access it from the public address anytime, anywhere along as the system is running. This allows someone to monitor the conditions in a real-time room over a week which we can use to correlate covid spread with High CO2 levels.

17 Appendix

Project Code: https://github.com/StephenPatOMalley/FinalYearProject

This below is a list of links that were used during the project as reference material.

17.1 ESP32

17.1.1 ESP32 Reference

Getting Started With ESP32 | ESP32 Arduino Programming Tutorials

Get Started - ESP32 - — ESP-IDF Programming Guide v4.3.1 documentation

ESP-IDF Getting Started | Espressif Systems

Standard Set-up of Toolchain for Windows - ESP32 - — ESP-IDF Programming Guide latest documentation

ESP32-DevKitC V4 Getting Started Guide - ESP32 - — ESP-IDF Programming Guide latest documentation

(PNG Image, 2000 × 1001 pixels) – Scaled (92%)

Quick User Guide for the ESP-IDF VS Code Extension - YouTube

17.1.2 ESP32_SetUp

Getting Started with ESP32 WROOM DevKitC v4 on Arduino IDE -

ESP32 Send Sensor Data to Google Firebase & Display on Android app

https://raw.githubusercontent.com/RuiSantosdotme/Random-Nerd-Tutorials/master/Projects/LCD_I2C/I2C_Scanner.ino

MQTT to Websockets with ESP32, NodeJS and D3.js – steveio.com

17.1.3 BME-280

ESP32 with BME280 using Arduino IDE (Pressure, Temperature, Humidity) | Random Nerd Tutorials

Measure CO2 with MQ-135 and Arduino Uno - Rob&apos;s blog

MQ-135 - MQ-135_Hanwei.pdf

Node.js - Express Framework

Advanced Features: Custom `Document` | Next.js

17.2 Part Refernces

17.2.1 MQ135

Connect MQ135 Air Quality Sensor and ESP32 to the Cloud over MQTT | AskSensors Blog

miguel5612/MQSensorsLib: We present a unified library for MQ sensors, this library allows to read MQ signals easily from Arduino, Genuino, ESP8266, ESP-32 boards whose references are MQ2, MQ3, MQ4, MQ5, MQ6, MQ7, MQ8, MQ9, MQ131, MQ135, MQ136, MQ303A, MQ309A.

Interfacing PN532 NFC RFID Module with Arduino - Electropeak

Humidity Sensor BME280 | Bosch Sensortec

17.3 Webpage

17.3.1 WebServer Refernce

Express application generator

React — How To Proxy To Backend Server | by Bhargav Bachina | Bachina Labs | Medium

React Crash Course for Beginners 2021 - Learn ReactJS from Scratch in this 100% Free Tutorial! - YouTube

5 useEffect Infinite Loop Patterns | by Naveen DA | JavaScript in Plain English

How to use MQTT in the React project | EMQ

17.3.2 Next.js

Next.js Crash Course for Beginners 2021 - Learn NextJS from Scratch in this 100% Free Tutorial! - YouTube

How can I wait In Node.js (JavaScript)? l need to pause for a period of time - Stack Overflow

How to create a table in ReactJS ? - GeeksforGeeks

Beautiful React Charts & Graphs | CanvasJS

React Timeseries Charts

Advanced Features: Custom `Document` | Next.js

reactjs - Error when using CanvasJS in a NextJS app - Stack Overflow

reactjs - Dynamically import a library with next.js - Stack Overflow

Advanced Features: Dynamic Import | Next.js

Integrating Navbar Links and NextJS Link Components · Issue #4131 · react-bootstrap/react-bootstrap

How to Center Anything with CSS - Align a Div, Text, and More

CSS Layout - The position Property

React-Bootstrap · React-Bootstrap Documentation

React Conditional Rendering Best Practices with 7 Different Methods

17.3.3 Node

Setting Up Node.js Project | CodeHandbook

javascript - MQTT.js multiple subscription - Stack Overflow

How to use MQTT in Node.js | EMQ

Getting started with Node.js and MQTT - LogRocket Blog

How to pass variables to the next middleware using next() in Express.js ? - GeeksforGeeks

MQTT in Node JS | Node JS Tutorial - YouTube

Using the Node.js MQTT Client-Starting Guide

mongoose - npm

JavaScript-based MQTT #3: Mosca, MQTT.js & MongoDB - YouTube

JavaScript-based MQTT #1: Mosca Broker & MQTT.js Client - YouTube

JavaScript-based MQTT #3: Mosca, MQTT.js & MongoDB - YouTube

Mongoose v6.2.4: Getting Started

Invalid "keyPath" option supplied. · Issue #352 · aws/aws-iot-device-sdk-js

Device communication protocols - AWS IoT Core

17.4 AWS

17.4.1 AWS EC2

ssh - What do options `ServerAliveInterval` and `ClientAliveInterval` in sshd_config do exactly? - Unix & Linux Stack Exchange

serveraliveinterval - ssh default timeout - Code Examples

amazon ec2 - keep server running on EC2 instance after ssh is terminated - Stack Overflow

Tutorial: Setting Up Node.js on an Amazon EC2 Instance - AWS SDK for JavaScript

WinSCP :: Official Site :: Free SFTP and FTP client for Windows

17.4.2 DynamoDB

Getting Started with Node.js and DynamoDB - Amazon DynamoDB

Tutorial: Storing device data in a DynamoDB table - AWS IoT Core

AWS DynamoDB DocumentClient & Node.js - Complete Cheat Sheet

Serverless app using NodeJS, React and AWS (API Gateway, Lambda, DynamoDB, S3) - YouTube

AWS IoT Device SDKs, Mobile SDKs, and AWS IoT Device Client - AWS IoT Core

(1) AWS Amplify Fullstack Project Setup (React, Node, Lambda, REST API) - YouTube

PubSub - Getting started - JavaScript - AWS Amplify Docs

17.4.3 ESP32_AWS

(2) Temperature Data record on AWS IoT Core with NodeMCU-ESP32 using Arduino IDE and MQTT Protocol. - YouTube

ArduinoJson: Efficient JSON serialization for embedded C++

Building an AWS IoT Core device using AWS Serverless and an ESP32 | AWS Compute Blog

AWS IoT Core action resources - AWS IoT Core

Create AWS IoT resources - AWS IoT Core

Connect ESP32 to AWS IoT (with Arduino code) | Savjee.be

Tutorial: Getting Started with Amazon AWS IoT, MQTT Protocol and Node JS | Dev Jam

17.4.4 MongoDB

Install MongoDB on Amazon EC2 — MongoDB Manual

How to Install MongoDB on AWS EC2 Instance? - GeeksforGeeks

mongodb - Unable to connect Robomongo to AWS EC2 via SSH - Stack Overflow

SSH Tunneling in Robo 3T to MongoDB runs on AWS EC2 | by Sarasa Gunawardhana | FAUN Publication

17.4.5 Lambda

Accessing a MongoDB instance from AWS Lambda using Python | Shikisoft Blog

Running AWS Lambda Functions in a VPC and Accessing RDS | Shikisoft Blog

Write A Serverless Function with AWS Lambda and MongoDB

Best Practices Connecting from AWS Lambda — MongoDB Atlas

node.js - AWS lambda with mongoose to Atlas - MongoNetworkError - Stack Overflow

node.js - Close Mongoose connection Lambda - Stack Overflow

(1) How to install npm modules in AWS Lambda? - YouTube

17.5 MQTT

17.5.1 ESP32 MQTT

GitHub - hideakitai/MQTTPubSubClient: MQTT and MQTT over WebSoket Client for Arduino

ESP32 MQTT client: Publish and Subscribe. HiveMQ and BME280 example

ESP32 MQTT Publish Subscribe with Arduino IDE | Random Nerd Tutorials

NodeJS MQTT Client talks to ESP32 - YouTube

17.5.2 Mosquitto

How to configure an MQTT Mosquitto broker and enable user authentication on Windows - YouTube

mosquitto_passwd man page | Eclipse Mosquitto

Using The Mosquitto_pub and Mosquitto_sub MQTT Client Tools- Examples

How to set-up your own MQTT Broker – O&apos;Brien Labs

Mosquitto MQTT Broker

Setting up Mosquitto Mqtt Broker on Windows - YouTube

How-To Get Started with Mosquitto MQTT Broker on a Raspberry Pi - YouTube

Installing and configuring the Mosquitto MQTT broker | A Mutable Log

Mosquitto — User Access Configurations Set-ups | by J3 | Jungletronics | Medium

17.6 Coivd-CO2

Carbon Dioxide - Carbon-Dioxide.pdf

2021-09-08-covid19guideventilationetco2monitoring2.pdf

What are safe levels of CO and CO2 in rooms? | Kane International Limited

17.7 Similar Projects

Corona Guard - Arduino Project Hub

CovidRoomMonitor - Jira

18 References

[1] "Unite Covid-19 guide on Ventilation and CO2 Monitoring Covid-19 Guide on Ventilation and CO 2 monitoring Ventilation and CO2 measurements to combat airborne Covid-19 contamination", Accessed: Apr. 20, 2022. [Online]. Available: https://www.cibse.org/news-and-policy/august-2021/new-air-cleaning-guidance-for-

[2] Espressif Systems, "ESP32-DevKitC V4 Getting Started Guide," *https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html*, 2021.

[3] Espressif Systems, "ESP32WROOM32D & ESP32WROOM32U Datasheet," 2021. [Online]. Available: https://www.espressif.com/en/support/download/documents.

[4] Espressif Systems, "Standard Setup of Toolchain for Windows - ESP32 - — ESP-IDF Programming Guide latest documentation," *Espressif Systems*, 2021. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/windows-setup.html (accessed Apr. 18, 2022).

[5] Espressif Systems, "Quick User Guide for the ESP-IDF VS Code Extension - YouTube," *Espressif Systems*, 2020. https://www.youtube.com/watch?v=Lc6ausiKvQM (accessed Apr. 18, 2022).

[6] dvillalva21, "FAILED: esp-idf/mbedtls/x509_crt_bundle (Invalid certificate) (IDFGH-3345) · Issue #5322 · espressif/esp-idf," Oct. 05, 2021. https://github.com/espressif/esp-idf/issues/5322#issuecomment-934055540 (accessed Apr. 18, 2022).

[7] https://www.iottechtrends.com/author/jagdale/, "Getting Started with ESP32 WROOM DevKitC v4 on Arduino IDE -," *iotTechTrends*, Apr. 23, 2021. https://www.iottechtrends.com/getting-started-with-esp32-wroom-devkitc/ (accessed Apr. 18, 2022).

[8] "[SOLVED] Failed to connect to ESP32: Timed out waiting for packet header | Random Nerd Tutorials," *Random Nerd Tutorials*, 2020. https://randomnerdtutorials.com/solved-

failed-to-connect-to-esp32-timed-out-waiting-for-packet-header/ (accessed Apr. 18, 2022).

[9] Bosch Sensortec, "Humidity Sensor BME280 | Bosch Sensortec," *Bosch Sensortec GmbH*, 2022. https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/ (accessed Apr. 18, 2022).

[10] "ESP32 with BME280 using Arduino IDE (Pressure, Temperature, Humidity) | Random Nerd Tutorials," *Random Nerd Tutorials*, 2020. https://randomnerdtutorials.com/esp32-bme280-arduino-ide-pressure-temperature-humidity/ (accessed Apr. 18, 2022).

[11] HANWEI ELECTRONICS, “MQ-135_Hanwei”.

[12] T.K. Hareendran, "How To Use MQ-135 Gas Sensor - Codrey Electronics," *Codrey*, Jul. 04, 2020. https://www.codrey.com/electronic-circuits/how-to-use-mq-135-gas-sensor/ (accessed Apr. 18, 2022).

[13] Miguel Angel Califa Urquiza, "MQSensorsLib," *GitHub*, Mar. 13, 2019. https://github.com/miguel5612/MQSensorsLib#Manuals (accessed Apr. 18, 2022).

[14] Rob, "Measure CO2 with MQ-135 - Rob's blog," *Blog*, Jan. 04, 2017. https://blog.robberg.net/mq-135-arduino/ (accessed Apr. 18, 2022).

[15] A. N. Dr Andy Stanford-Clark, "MQTT - The Standard for IoT Messaging," *MQTT*, 2027. https://mqtt.org/ (accessed Apr. 18, 2022).

[16] "Eclipse Mosquitto," *Eclipse Foundation* , Apr. 18, 2022. https://mosquitto.org/ (accessed Apr. 18, 2022).

[17] AutomationStation, "How to configure an MQTT Mosquitto broker - Youtube," *Youtube*, Jun. 18, 2020. https://www.youtube.com/watch?v=72u6gIkeqUc (accessed Apr. 18, 2022).

[18] Pat, "How to set-up your own MQTT Broker – O'Brien Labs," *O'Brian Labs*, Sep. 18, 2018. https://obrienlabs.net/how-to-setup-your-own-mqtt-broker/ (accessed Apr. 18, 2022).

[19] Hideaki Tai, "MQTTPubSubClient: MQTT and MQTT over WebSoket Client for Arduino," *GitHub*, May 02, 2021. https://github.com/hideakitai/MQTTPubSubClient (accessed Apr. 18, 2022).

[20] OpenJS Foundation, "About | Node.js," *OpenJS Foundation*, May 2022. https://nodejs.org/en/about/ (accessed Apr. 19, 2022).

[21] "React (JavaScript library) - Wikipedia," *Wikipedia*, Apr. 12, 2022. https://en.wikipedia.org/wiki/React_(JavaScript_library) (accessed Apr. 19, 2022).

[22] Vercel, "Next.js by Vercel - The React Framework," *Vercel*, 2022. https://nextjs.org/ (accessed Apr. 19, 2022).

[23] "MongoDB - Wikipedia," *Wikipedia*, Mar. 12, 2022. https://en.wikipedia.org/wiki/MongoDB (accessed Apr. 19, 2022).

[24] Shifan Yu, "How to use MQTT in Node.js | EMQ," *eqmx*, Sep. 08, 2021. https://www.emqx.com/en/blog/how-to-use-mqtt-in-nodejs (accessed Apr. 19, 2022).

[25] MongoDB, "Default MongoDB Port — MongoDB Manual," *MongoDB*, 2022. https://www.mongodb.com/docs/manual/reference/default-mongodb-port/ (accessed Apr. 19, 2022).

[26] Tutorials Point, "Node.js - Express Framework," *Tutorials Point*. https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm (accessed Apr. 19, 2022).

[27] OpenJs Foundation, "Express application generator," *Express*, 2017. https://expressjs.com/en/starter/generator.html (accessed Apr. 19, 2022).

[28] MIT, "API Routes: Introduction | Next.js," *NextJS*. https://nextjs.org/docs/api-routes/introduction (accessed Apr. 19, 2022).

[29] Meta, "Context – React," *ReactJS*, 2022. https://reactjs.org/docs/context.html#api (accessed Apr. 19, 2022).

[30] MIT, "Basic Features: Pages | Next.js," *Vercel*. https://nextjs.org/docs/basic-features/pages#pages-with-dynamic-routes (accessed Apr. 19, 2022).

[31] MIT, "Advanced Features: Custom `Document` | Next.js," *Vercel*. https://next-js.ir/docs/advanced-features/custom-document (accessed Apr. 19, 2022).

[32] Fenopix, "Beautiful HTML5 Charts & Graphs | 10x Fast | Simple API," *CanvasJS*. https://canvasjs.com/ (accessed Apr. 19, 2022).

[33] Rahul, "Reactjs Dynamically Import - Stack Overflow," *Stack overflow*. https://stackoverflow.com/questions/65522762/dynamically-import-a-library-with-next-js (accessed Apr. 19, 2022).

[34] Moheeb Zara, "Building an AWS IoT Core device using AWS Serverless and an ESP32 | AWS Compute Blog," *aws*, Jan. 03, 2020. https://aws.amazon.com/blogs/compute/building-an-aws-iot-core-device-using-aws-serverless-and-an-esp32/ (accessed Apr. 20, 2022).

[35] "ArduinoJson: Efficient JSON serialization for embedded C++." https://arduinojson.org/ (accessed Apr. 20, 2022).

[36] Ezequiel Miranda, "Tutorial: Getting Started with Amazon AWS IoT, MQTT Protocol and Node JS | Dev Jam," *Medium*, Aug. 13, 2021. https://medium.com/dev-jam/getting-started-with-aws-iot-core-and-mqtt-protocol-with-a-node-js-example-ed16bd542704 (accessed Apr. 20, 2022).

[37] AWS, "Tutorial: Setting Up Node.js on an Amazon EC2 Instance - AWS SDK for JavaScript," *AWS Docs*, 2022. https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html (accessed Apr. 20, 2022).

[38] GeeksForGeeks, "How to Install MongoDB on AWS EC2 Instance? - GeeksforGeeks," *GeeksForGeeks*, Jan. 02, 2022. https://www.geeksforgeeks.org/how-to-install-mongodb-on-aws-ec2-instance/ (accessed Apr. 20, 2022).