



# Final Year Project

## Manure Moover

Thomas Martin

Bachelor of Software & Electronic Engineering(Hons.)

Galway-Mayo Institute of Technology

2019/2020

# Manure Moover

Thomas Martin  
Software and  
Electronic  
Engineering

## Technologies Used:

ESP32 with Arduino  
IDE  
MQTT with AWS IOT  
Core  
MEAN Stack

## Features:

- Self-Locating
- Remote Control
- Website for User and Admin
- Map for location
- Memory of tracks

## About the Project:

### Why?

While working in Kerry, I visited a farm of 300 cattle in December. As it was the middle of winter the sheds were mucky. I thought the farmer needs an aid to clean the sheds.

### How?

A cleaning robot that can be controlled remotely and remembers the tracks around the shed.

### Does it work?

Yes, the concept can be applied. Due to not living on a farm I was not able to test the project fully, however the testing that I carried out in my own home proved a success. The remote control is implemented over MQTT from the User Interface. The track data is stored in a MongoDB using RSSI data from 3 ESP32s in each corner of the shed. The user is able to view a map of the routes on the User Interface

### Tank/Hardware

L293D

ESP32

MQTT

AWS IOT  
Core

### User Interface

JavaScript

HTML/CSS

NodeJS

MEAN  
Stack

## Declaration

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

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

---

## Acknowledgements

Brian O'Shea - Project Mentor.

## Table of Contents

1	Summary.....	6
2	Introduction.....	7
3	Project Goals.....	8
4	Project Architecture .....	9
5	MEAN Stack .....	11
6	MQTT .....	16
7	Amazon Web Services .....	17
8	The Robot .....	18
9	User Interface .....	20
10	Project Timeline .....	27
11	Problem Solving .....	28
12	Conclusion.....	29
13	References .....	30
14	Author .....	31

## 1 Summary

The Manure Moover is a little robotic shed cleaner. The brains of the robot is an esp32 programmed with C/C++. It is connected to an L293D motor driver chip to control the motors on the robot with pulse width modulation(PWM). The robot uses MQTT as a messaging service between the robot and the user interface. MQTT is implemented in Amazon Web Services Internet of things Core, with two topics, one for the esp32 to publish to and one for the esp32 to subscribe to. The user interface is a website hosted in Node.js as part of a MEAN stack. The user interface is written in JavaScript and HTML with CSS. On the user interface, a user with admin credentials can add/delete users. A general user can control the robot, create new tracks and view the previous tracks stored in the MongoDB. MongoDB is used to store the login credentials of different users and to store previous tracks around the shed. The previous tracks around the shed are a sequence of Received Signal Strength Indications(RSSI) from 3 other esp32 microcontrollers in the shed set up as access points.

## 2 Introduction

In December of 2019, while working for Dairymaster, I got to go on a farm testing visit. This farm was quite large in my own opinion and had quite a number of cattle and sheds. I believe there was around 300 cattle on the farm that day, housed between 4 sheds. The cattle had been in the sheds all winter and had another few months to go before they would get to see the green fields again. The conditions of the shed after such a period indoors as you would imagine was quite mucky.

Usually on farms, the farmer will clean the sheds out around calving time. It is always a big job to try and find the time to clean the sheds even after calving. This got me thinking about a way to aid the farmer to clean the sheds more often. I quickly came up with the idea of the Manure Moover.

### 3 Project Goals

For this project I have set the following goals:

- Develop a good standard of product.
- Increase my understanding of electronic circuitry.
- Acquire knowledge of JavaScript.
- Acquire knowledge of MEAN Stack development.
- Gain an understanding of AWS.



## 4 Project Architecture

The microcontroller I used for this project is the Espressif ESP32 DevkitC. This boasts many useful features, such as GPIO pins, WiFi and Bluetooth. It is an Amazon qualified device meaning that it has connectivity with the Amazon Web Services(AWS) IOT Core.[1] I programmed the microcontroller in the Arduino IDE. This IDE is a tool that I have worked with many times both on my own and as part of my studies in GMIT.



Image: ESP32 DevkitC Microcontroller.

Also on board the robot, I had an L293D motor driver chip which allowed the motors of the robot to be driven in both forward and reverse. Through the Arduino IDE I was able to program PWM to have 5 different speeds for the motors to spin.

For the user interface website I used JavaScript and HTML with CSS in the Atom IDE. This was an IDE I had not used before as I hadn't previously worked with JavaScript as closely. I created a Node.js server through the Atom IDE and this was all implemented as part of a M.E.A.N stack. Node.js allowed me to create a well designed website. Node.js is an excellent tool for building scalable network applications.

As part of the MEAN stack I had MongoDB to store all the data for the project that I needed. MongoDB is a document database and as a result is very good for storing JSON objects.

I used AWS IOT Core as the messaging broker for MQTT. This broker was easy to use and very reliable. There are many tutorials and blogs on the AWS website that assisted me with using this broker.

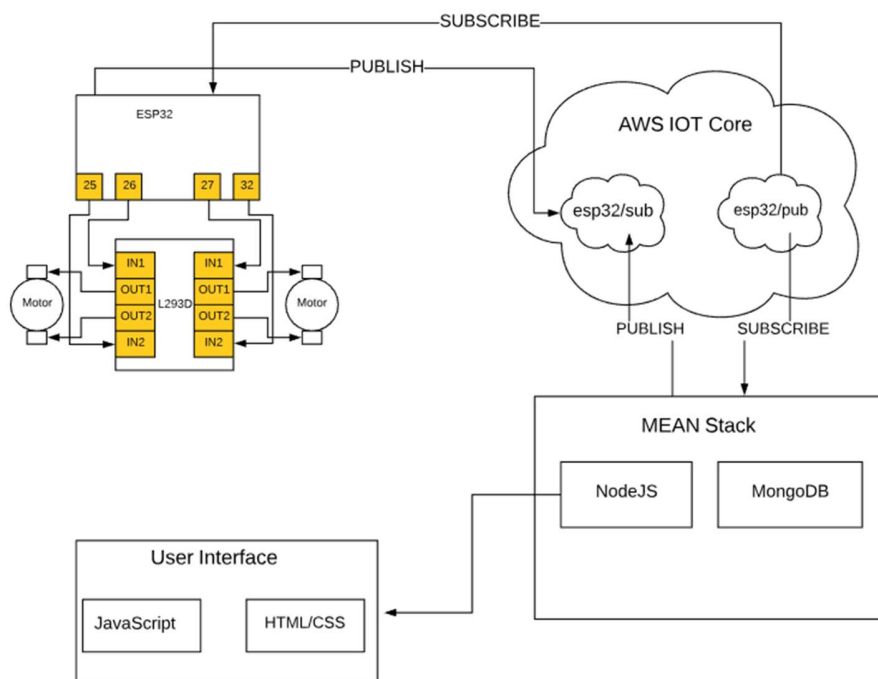


Image: Architecture Diagram.

## 5 MEAN Stack

A MEAN Stack consists of MongoDB, Express.js, AngularJS and Node.js. All components can be written in JavaScript which allows one language for server-side and client-side execution environments[2].

Node.js is a server side JavaScript execution environment. It's a platform built on Google Chrome's V8 JavaScript runtime. It helps in building highly scalable and concurrent applications rapidly.[3]

Express is lightweight framework used to build web applications in Node. It provides a number of robust features for building single and multi page web application. Express is inspired by the popular Ruby framework, Sinatra. [4]

MongoDB is a schema less NoSQL database system. MongoDB saves data in binary JSON format which makes it easier to pass data between client and server.[5]

AngularJS is a JavaScript framework developed by Google. It provides some awesome features like the two-way data binding. It's a complete solution for rapid and awesome front end development.[6]

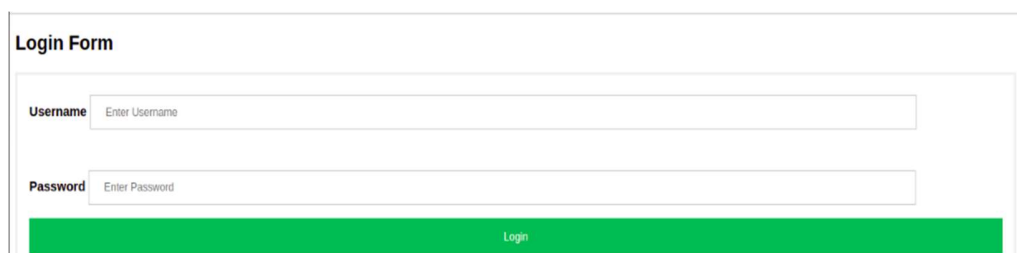
## 5.1 Express.js

Express.js is a lightweight framework for Node.js. Express.js deals with all the low level processes and requests in an application. Express.js provides most of the modules required in Node.js e.g. to allow connection to a MongoDB.

The index.js page is where all the different routes are sorted. These routes are the URLs that would be entered into the browser window and extracted through Express. An example of this in my index.js page is below. The user has connected to the home page of the website. Upon receiving the URL, Express directs the program to this function. The login page is rendered in response and the HTML code 200 is also sent to the browser.

```
router.get('/', function(req, res, next) {  
  AdminLoggedIn = false;  
  loggedIn = false;  
  res.render('login');  
});
```

Image: Home Page request.



**Login Form**

Username

Password

Image: Home Page response.

## 5.2 MongoDB

MongoDB is a cross-platform NoSQL database. It stores data in Collections of documents. A Collection can store documents of varying sizes. For this project, I have a userDB with a Collection called Users. This Collection stores all the known users of the system and a password for each. In the code below, I connect to the MongoDB using MongoClient, a built in function, and read all the users in the collection into an array. From here I can check if the user is a general user or if the user is an admin.

```
var user = {
  name: req.body.uname,
  password: req.body.psw
};

MongoClient.connect(url, function(err, db) {
  if (err) throw err;
  var dbo = db.db("userDb");
  var arr = dbo.collection("Users").find({
    User: req.body.uname
  }).toArray(function(err, result) {
    if (err) throw err;
    console.log("Got as far as here, array length is: " + result[0].User + " " + result[0].Password);
    attemptedUser.setDetails(result[0].User, result[0].Password);
    if (user.name == attemptedUser.name && user.password == attemptedUser.password) {
      loggedIn = true;
      if (user.name == "Admin") {
```

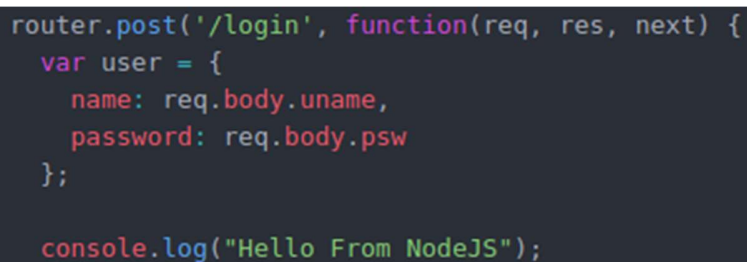
Image: Connecting to MongoDB.

## 5.3 Node.js

Node.js runs one single non-blocking thread, and works on event basis. As a result Node.js is extremely fast at processing requests from the client.

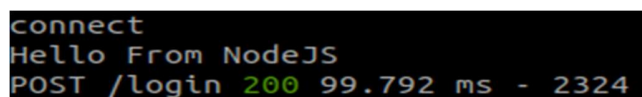
Node.js is a runtime environment for JavaScript outside of the web browser. This means it converts the JavaScript code to machine code. This allows the developer to run and view code on the machine.

An example of this is when I used `console.log` in my code. This function printed to the terminal screen as the server was running.



```
router.post('/login', function(req, res, next) {  
  var user = {  
    name: req.body.uname,  
    password: req.body.psw  
  };  
  
  console.log("Hello From NodeJS");  
});
```

Image: Using JavaScript to write to Terminal.



```
connect  
Hello From NodeJS  
POST /login 200 99.792 ms - 2324
```

Image: Terminal output.

## 5.4 AngularJS

AngularJS is the frontend part of a MEAN Stack. AngularJS allows HTML pages to have dynamic variables. These variables can be set within the JavaScript code or retrieved from JSON resources.

Throughout the project I use AngularJS to fill tables with results from MongoDB. Another aspect I use AngularJS for is setting titles and personalizing messages to the user. In the code bellow, the user has entered a set of login details which have not been recognized. The user is greeted with a HTML page filled with a message and the details they have entered.

```
res.render('error', {  
  message: "Details incorrect",  
  username: user.name,  
  password: user.password  
});
```

Image: AngularJS in use to set variables in HTML.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <h1>{{ message }}</h1>  
    <h2>You Entered:</h2>  
    <h2>Username:{{ username }}</h2>  
    <h2>Password:{{ password }}</h2>  
  </head>  
  <body>  
    <a href="/"><button>Try Again</button></a>  
  </body>  
</html>
```

Image: HTML Page with variables set using AngularJS.

## 6 MQTT

MQTT is a lightweight publish/subscribe messaging protocol. It was first designed in 1999 to allow Oil Pipeline telemetry systems connect over satellite. MQTT has very strong delivery guarantees. MQTT Servers are known as brokers. These brokers publish the incoming message to all subscribed clients.

In my project I used a Quality of Service(QoS) level of 0. This allowed fast messaging. This level of service is used when internet is reliable and message loss on a small scale is not a problem.

A QoS 1 client stores the message it has sent to the broker until it receives a message from the broker to say that the message has been received. If the client does not receive this message then it will resend the message.

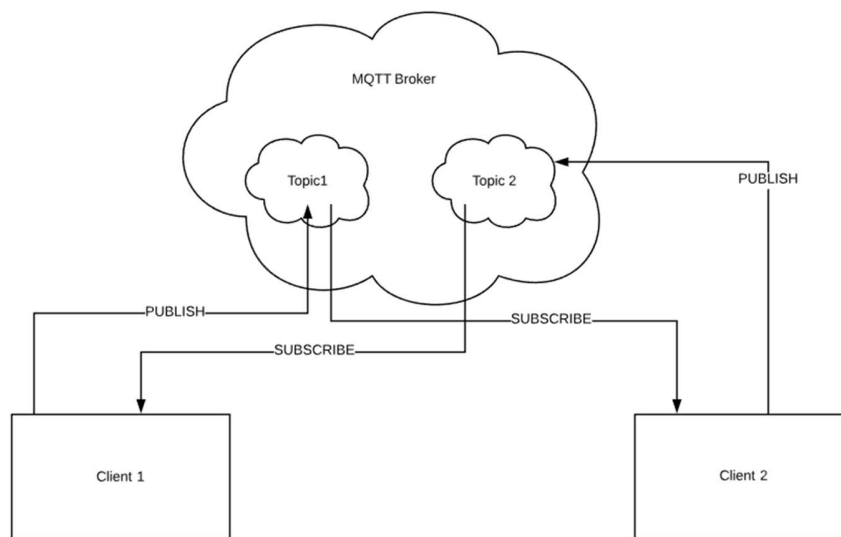


Image: How MQTT works.



## 7 Amazon Web Services

Amazon Web Services(AWS) provides cloud resources to all user groups, from the individual to large companies. As part of the service provided there is an AWS Educate account provided for students. This is what I used for my project.

In this account I had full access to all the available services. One of these services is the IOT Core. This is a service that can process and route messages to various endpoints in the cloud or to other connected devices.

I used Amazon Web Service IOT Core as my MQTT broker. I used AWS as my broker as it has been integrated with a MEAN Stack previously. In college, I was also taking a class which was informing me about AWS and also while working in Kerry, I had been given a brief introduction to the platform.



## 8 The Robot

On board the robot, there is 2 motors which are controlled by a L293D motor driver chip. This chip allows the motor to be operated in forward and reverse. The L293D input pins are connected to the Esp32 on pins 25,26,27,32. These pins are given a PWM value between 150 and 250. This value is determined from the MQTT message received. This value updates every second. The motors require 200mA to run properly, as a result an external 15V battery pack provides power for the motors.

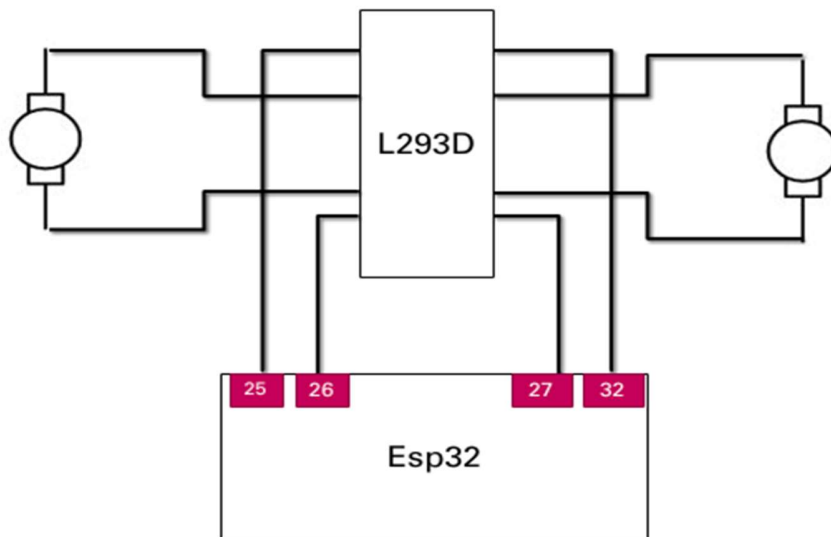


Image: L293D circuit.

```
ledcAttachPin(motor1Pin1, 0); // assign motor pins to a channel
ledcAttachPin(motor1Pin2, 1);
ledcAttachPin(motor2Pin1, 2); // assign motor pins to a channel
ledcAttachPin(motor2Pin2, 3);
ledcSetup(0, 4000, 8); // 12 kHz PWM, 8-bit resolution
ledcSetup(1, 4000, 8);
ledcSetup(2, 4000, 8); // 12 kHz PWM, 8-bit resolution
ledcSetup(3, 4000, 8);
```

Image: Motor Setup Code.

```

void Stop(){
    ledcWrite(0, 0);
    ledcWrite(1, 0);
    ledcWrite(2, 0);
    ledcWrite(3, 0);
}
void Forward(int drive){
    Serial.print("drive = ");
    Serial.println(drive);
    ledcWrite(0, (150 + (10*drive) ) );
    ledcWrite(1, 0 );
    ledcWrite(2, (150 + (20*drive) ) );
    ledcWrite(3, 0 );
}
void Reverse(int drive){
    Serial.print("drive = ");
    Serial.println(drive);
    Serial.println((((255/10)*drive)));
    drive = drive * -1;
    ledcWrite(3, (150 + (20*drive) ) );
    ledcWrite(2, 0 );
    ledcWrite(1, (150 + (10*drive) ) );
    ledcWrite(0, 0 );
}
void Left(int turn){
    Serial.print("turn = ");
    Serial.println(turn);
    turn = turn * -1;
    ledcWrite(0, (150 + (20*turn) ) );
    ledcWrite(1, 0 );
}
void Right(int turn){
    Serial.print("turn = ");
    Serial.println(turn);
    ledcWrite(2, (150 + (20*turn) ) );
    ledcWrite(3, 0 );
}

```

Image: Motor Control Code.

## 9 User Interface

### 9.1 Administrator Interface

Once the attempted user details are verified against a MongoDB storing all user details. If the user details are verified as an Admin then they are taken to the Administration page. In this page, the User has access to the full list of users and passwords. The User also has the ability to add a new user to the database, and to remove a user from the database.

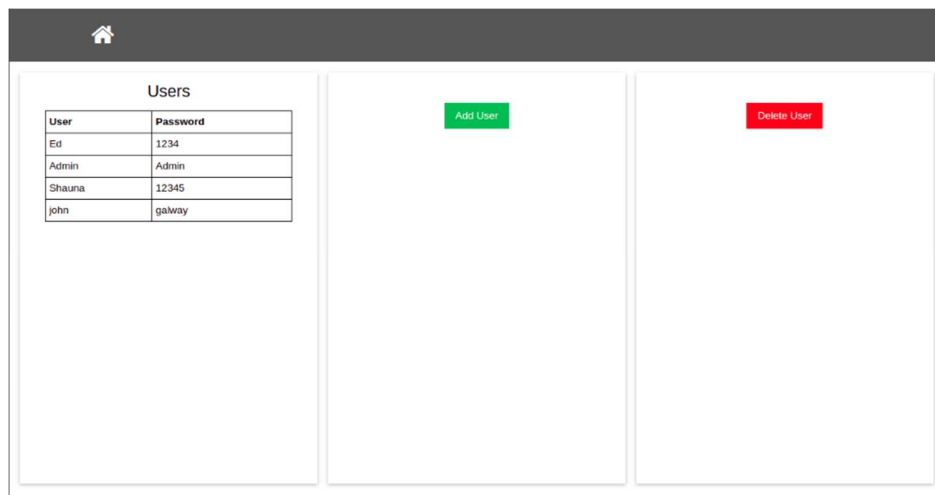


Image: Admin Page.

## 9.2 User Interface

If the user details verified against the MongoDB return that the details are correct the user is brought to the home screen. Here the user is greeted with a welcome message and a navigation bar to navigate to the other parts of the website.

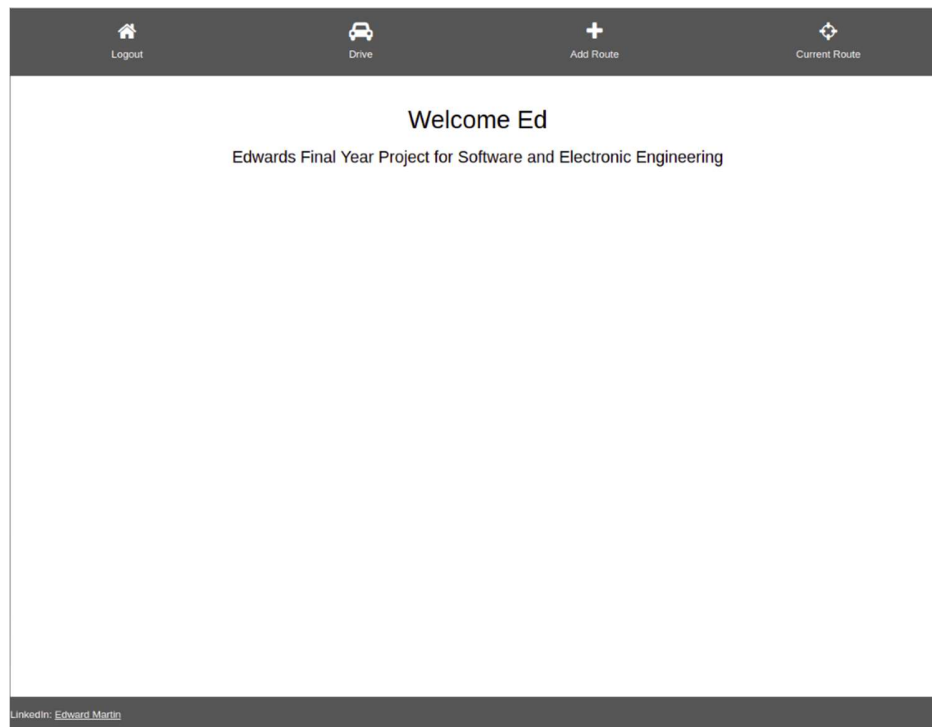


Image: User Home.

Once the user clicks on the drive icon, they are brought to the drive page. This consists of two sliders, one for forward and reverse, the other for left and right. The value of these sliders are passed every second using an AJAX over MQTT to the Esp32 on board the tank. Then depending on the value received from MQTT the Esp32 outputs a PWM value to the motors.

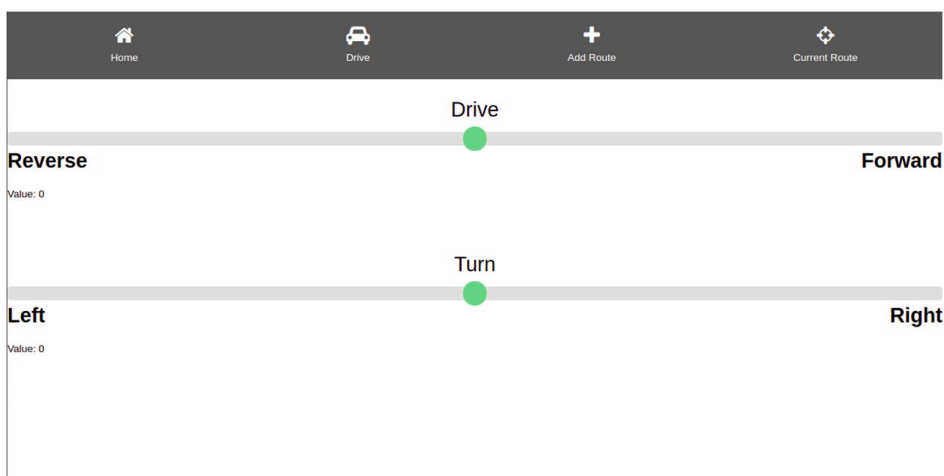


Image: Drive Page.

```
GET /Drive 304 2.313 ms - -  
esp32/sub -1  
POST /SendControl 200 1.708 ms - 2  
esp32/sub -2  
POST /SendControl 200 2.203 ms - 2  
esp32/sub -3  
POST /SendControl 200 2.431 ms - 2  
esp32/sub -2  
POST /SendControl 200 1.499 ms - 2  
esp32/sub -1  
POST /SendControl 200 1.640 ms - 2  
esp32/sub 0  
POST /SendControl 200 1.767 ms - 2  
esp32/sub 1  
POST /SendControl 200 1.847 ms - 2  
esp32/sub 2  
POST /SendControl 200 1.484 ms - 2  
esp32/sub 0  
POST /SendControl 200 2.094 ms - 2  
esp32/sub 0
```

Image: Terminal Output of MQTT Data.

If the User clicks on the Add Route button in the navigation bar, the user is prompted to enter a route name in a prompt box. When this prompt box is filled, a message is sent to the ESP32 to send RSSI values over MQTT every 5seconds. The user is greeted with a similar page to the drive page, there is one extra stop button. When this button is clicked a message is sent to the ESP32 to stop sending RSSI values. Each time a message is received from the ESP32 with RSSI values, the values are saved into a collection with the name entered in the prompt. This is the track memory.



Image: New Route Prompt.

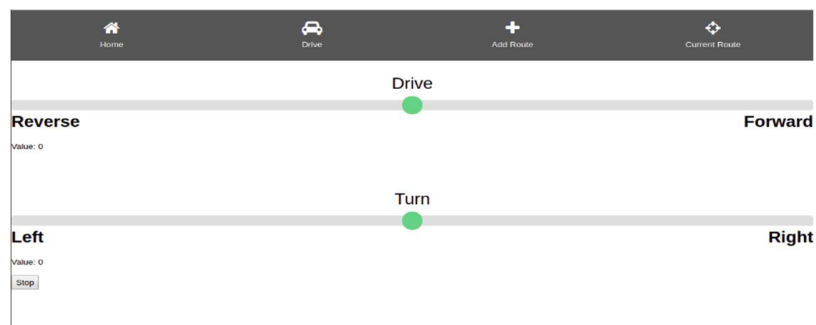


Image: New Route Page.

```
Collection created!
{"Time":227,"RSSI1":-42,"RSSI2":-48,"RSSI3":-59}
Time: 1000 RSSI1: -42 RSSI2: -48 RSSI3: -59
1 document inserted
{"Time":234,"RSSI1":-42,"RSSI2":-59,"RSSI3":-59}
Time: 2000 RSSI1: -42 RSSI2: -59 RSSI3: -59
1 document inserted
esp32/sub -7
POST /SendControl 200 1.398 ms - 2
esp32/sub 0
POST /SendControl 200 1.229 ms - 2
esp32/sub 5
POST /SendControl 200 4.668 ms - 2
esp32/sub 0
POST /SendControl 200 1.861 ms - 2
esp32/sub 5
POST /SendControl 200 1.711 ms - 2
{"Time":241,"RSSI1":-44,"RSSI2":-50,"RSSI3":-49}
Time: 3000 RSSI1: -44 RSSI2: -50 RSSI3: -49
1 document inserted
esp32/sub 0
```

Image: New Route Terminal MQTT.



When the user clicks on the Current Route button in the navigation bar, they are greeted with a screen that displays a table of all the previous tracks saved in the database. The user has to enter the name of the route they wish to view before they can proceed.

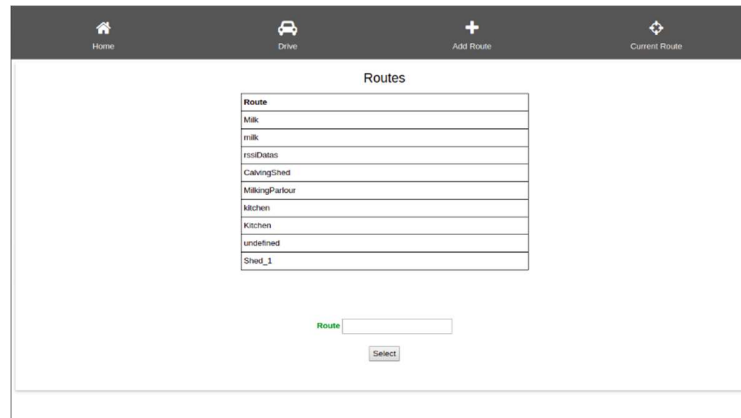


Image: List of Routes Stored in Database.

Having selected a route, the user is then brought to a page displaying all the RSSI data collected for that route and an outline of the track on a map. The map points are calculated as intersecting points of circles with radius equal to the RSSI value. RSSI1 and RSSI2 intersection point and RSSI1 and RSSI3 intersection point are averaged to give one point if both points are slightly different. Every second the map updates to simulate the robot moving around the shed.

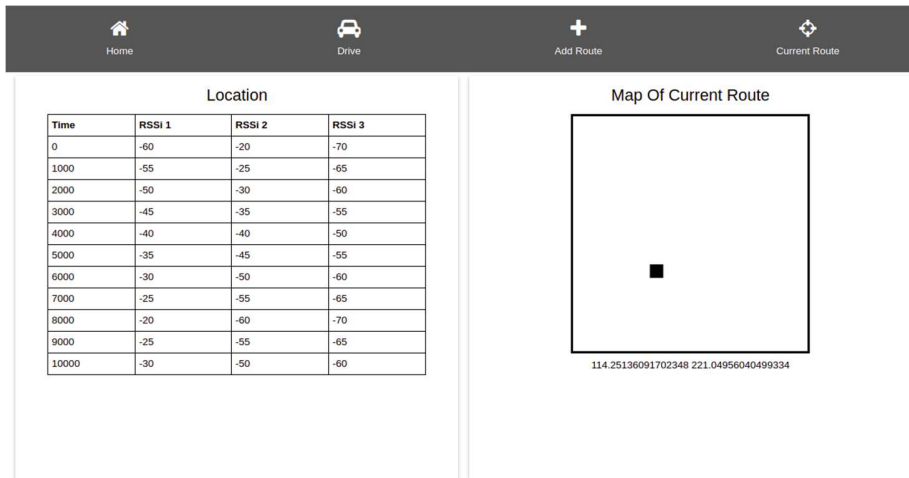


Image: Selected Route and Map of Route.

## 10 Project Timeline

Throughout the project, I have set myself targets to have reached by certain times in the year. Some of these have been hit and some have been missed due to unforeseen circumstances.

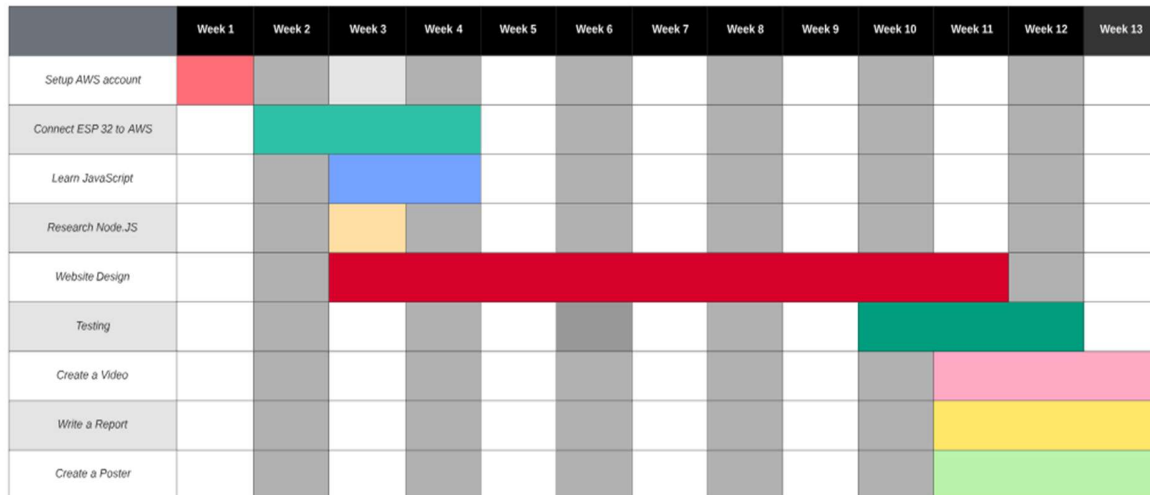


Image: Gantt Chart for Semester 2.

## 11 Problem Solving

During the course of this project I have faced many challenges and difficulties. These difficulties have been both in and out of my control.

The biggest challenge faced during this project was the global pandemic of Covid-19. This pandemic hit in March. This removed access to the college for all remaining time in the semester. It also meant that face to face communication with my project mentor had been removed. I had to greatly improve my communication skills over email and video call. The pandemic also meant that I had to set up a work space at home. This put great strain on all of my equipment from laptop and WiFi to ESP32.

Another big challenge I faced was JavaScript. This project was my first exposure to JavaScript. I had a big learning curve with JavaScript and through online tutorials and videos I was able to gather a strong knowledge of JavaScript.

One major problem I encountered was with AWS. My credits on the Educate account ran out in the final stages of this project. As a result I was unable to deploy the user interface into the cloud.

## 12 Conclusion

I feel taking into consideration the obstacles I faced along the way that the project I have created is of high quality. As previously mentioned, I feel my knowledge of JavaScript has grown enormously. I also believe the same about MEAN Stack Development. Before I took on this project I had a limited knowledge of backend development.

If on day one of this project I was told that I would have ended up here, I would have been very happy. However, there is always room for improvement. I believe that with more time and resources I will be able to further improve on some aspects of this project. I also believe that I will be able to add a few extra features when deadlines aren't visible.

## 13 References

- [1] Espressif, Overview, <https://www.espressif.com/en/products/devkits/esp32-devkitc/overview>, accessed May 2020.
- [2] Wikipedia, MEAN (solution stack), [https://en.wikipedia.org/wiki/MEAN\\_\(solution\\_stack\)](https://en.wikipedia.org/wiki/MEAN_(solution_stack)), accessed May 2020.
- [3][4][5][6] Sitepoint, An Introduction to the MEAN Stack, <https://www.sitepoint.com/introduction-mean-stack/>, accessed May 2020.
- [7] mntoli, MQTT QoS Levels, <https://mntolia.com/mqtt-qos-levels-explained/>, accessed May 2020.

## 14 Author

Author: Thomas Martin.

Email: [edwardm12346@gmail.com](mailto:edwardm12346@gmail.com)

LinkedIn: <https://www.linkedin.com/in/edwardmartinswelecgmit/>