CANview

A view into your machine

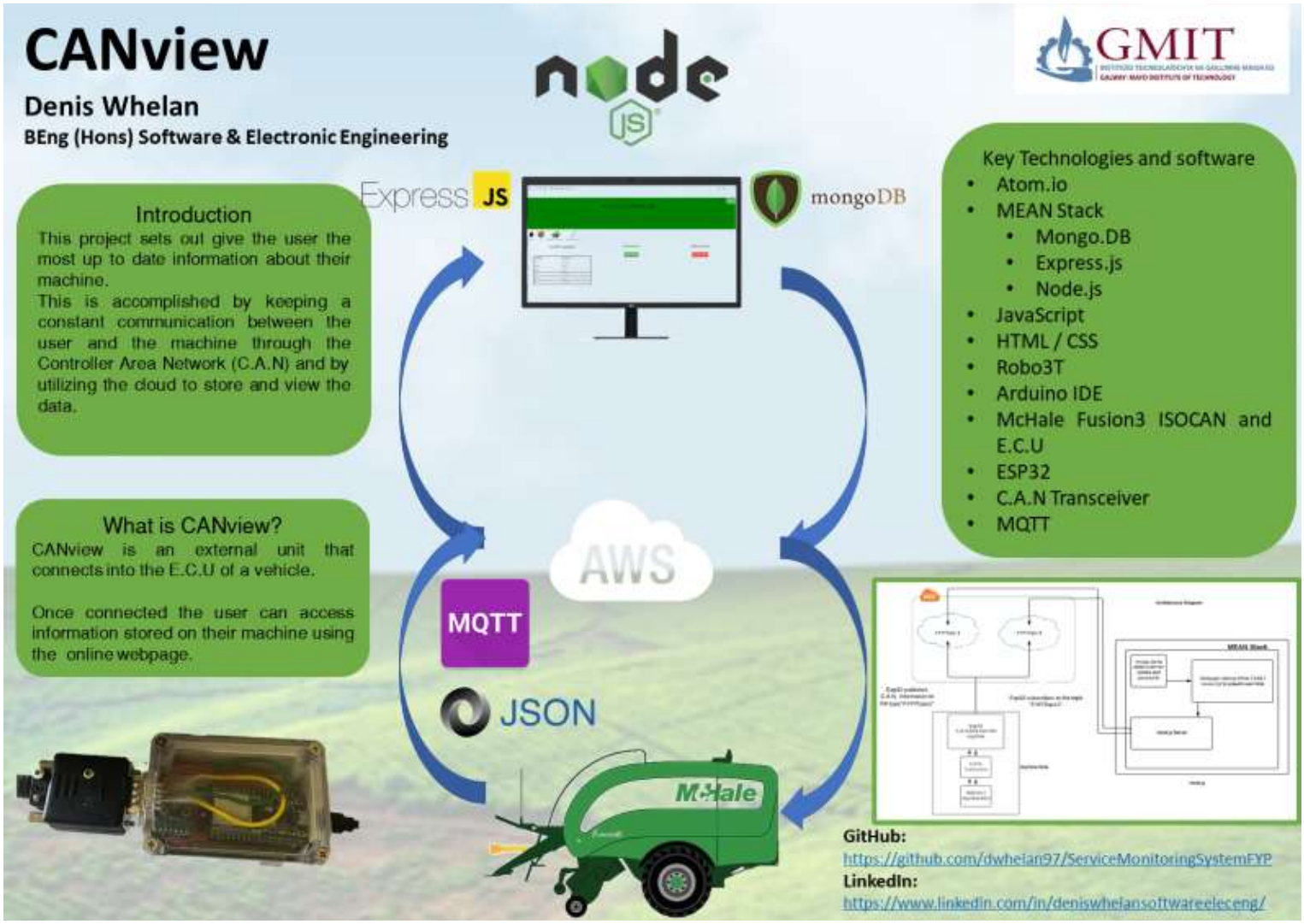
Name: Denis Whelan

Supervisor: Brian O Shea

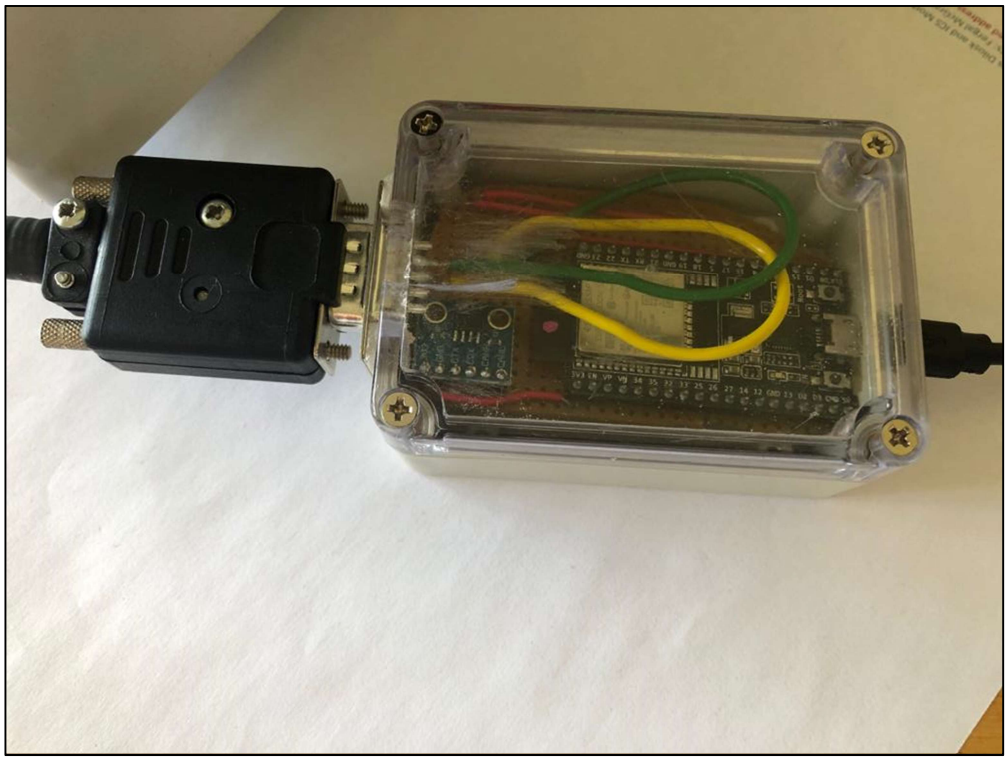
Course: BEng (Honours) in Software & Electronic Engineering

Year: 2019/2020

1 Project Poster



2 Project Graphic



3 Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) 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.

_____

4 Acknowledgements

I would like to take this opportunity to thank Brian O Shea for his abundant support, patience and guidance throughout the project not only as my FYP supervisor, but over my time in the Galway Mayo Institute of Technology as a whole. I would like to also thank all the faculty and staff of the Software, Electronic and Computer Engineering Department for their help and support over the last four years.

A special word of thanks to all of the R&D team in McHale Engineering, for kindly allowing me to undertake this project using their equipment and their expertise, and patiently accommodating my visits to Ballinrobe. I would also like to especially thank John Warren, Aodhán Ó Gabhann and Conor McKiernan for their help and support with me throughout the course of the project.

Over the last four years, I have been privileged to be part of a group of fellow students whose friendship and support cannot be quantified, and for this I cannot say how thankful I am. Finally I would like to thank both my parents and family for their support, without them I would not have got to where I am today.

Table of Contents

5 Summary

CANview

This project sets out to tackle the issue of inaccessible Electronic Control Unit information in a vehicle. An example application for this system would be accessing the customer information on a McHale Fusion3 ISOCAN and ECU.

The goal of this project was to deliver a fully functioning ECU plug in with a webpage to give the user the most up to date information about their vehicle. I have chosen agricultural vehicles for this project due to their standard J1939 protocol which allows easy access to the Controller Area

Network (C.A.N.). This project is targeted at all farmers who own a McHale Fusion 3 baler and who are interested in being able to wirelessly view a live feed from their machine.

For increased performance, and improved results, the system incorporates a simple online user interface to view the data. One very useful feature of this project is its ability to be scaled to include more information on the webpage such as Machine Diagnostics and Machine Servicing information.

This project can provide a good grounding to suit a wide range of applications that require access to a vehicles Electronic Control Unit, and its implementation can be adapted to perform a multitude of tasks due to the flexibility in using a microcontroller as the main processor.

Some of the key technologies within this project involve the use of MQTT communication between an ESP32 and Amazon Web Services (A.W.S) Topics, the utilization of M.E.A.N stack to store and present the data and the use of JavaScript, HTML and CSS to make the webpage attractive to the user.

Overall this project has met the guidelines and expectations specified during the idea stage. Giving the farmer the ability to see all of its machines requirements and data in the palm of their hand, without stoppage and without lost time.

6 Introduction

When thinking of an idea for my final year project it was clear to me that agriculture (agri.) based tech would most definitely be on the cards. Having spent many summers working in and around heavy machinery I have seen first-hand the stresses, strains and workload that these machines are put through during the busy summer months. During my time working with many big contracting outfits it shocked me that while the machines themselves are top of the range and in excess of hundreds of thousands of euro, the technology in them requires button presses, and warning lights on the dash to show that there is an issue.

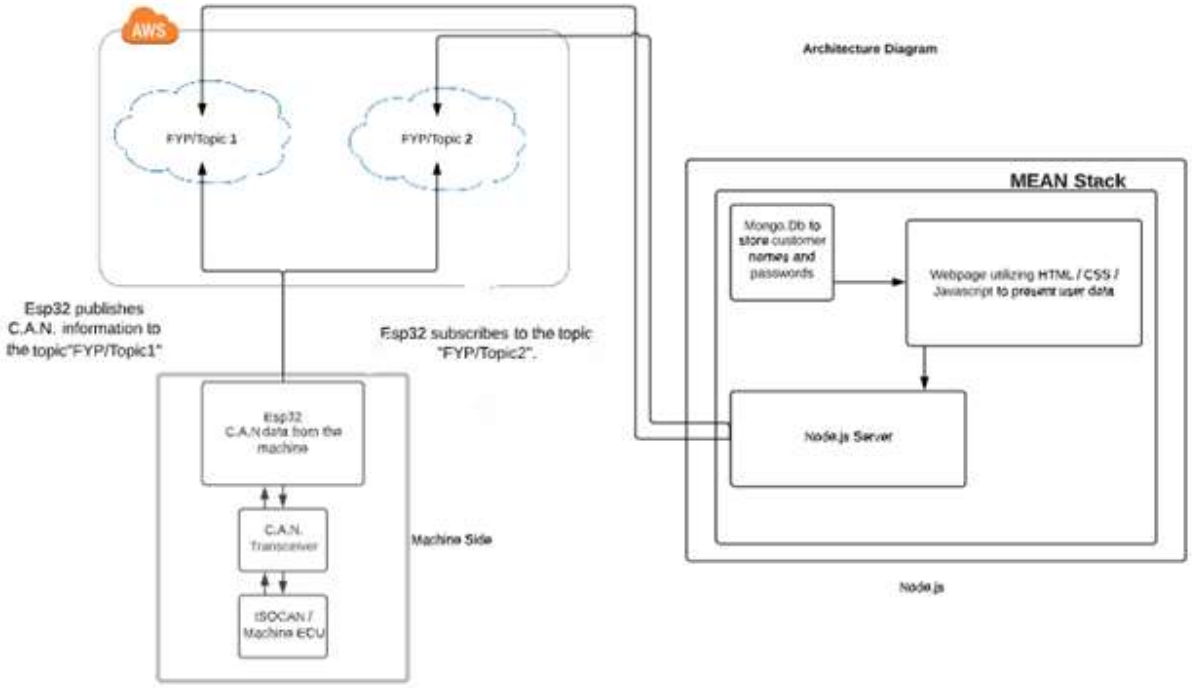
I believe that if Irish farming wants to stay at the forefront it needs to become more modern and tech savvy. This means making our machinery smarter and maybe eventually autonomous, giving our land the ability to tell us what it needs instead of the guess work that is currently required and building farms with a technological background.

In 2017 I brought another agri. Tech project to the Irelands Best Young Entrepreneur Finals. The project looked at the issues of toxic gasses around farms and how technology could be used to make farms a safer place.

Having all this behind me led me down the road of this project. I wanted to develop a project that could be implemented in the real world, hence I have based my project on the current most up to date baler in the country. Currently this machine requires the user to be in the cab of the machine to view the customer information. CANview will allow the user access all the most up to date information from anywhere in the world.

This project gives me the opportunity to bring together my personal interests and experience and improve the lives of machine users.

7 Project Architecture



Architecture Diagram

7.1 Project Goals

My Goal for this project was to further increase my understanding of the idea of C.A.N and its messaging protocols. Having previously worked with C.A.N. while on a summer placement in Valeo Vision Systems. While I used E.C.U's on a daily basis while working here I never properly understood what the technology was and how it is used in a vehicle.

Another goal was to develop my knowledge of JavaScript. Having never used the language before, at first I found it difficult however after some months of programming it began to start making sense to me. This was one of the key reasons why I used a M.E.A.N stack as this software stack could all be programmed in JavaScript.

An alternative reason for doing this project was to begin to develop contacts with agri. tech companies as this is an area which I have a huge interest in and one that I believe will grow exponentially.

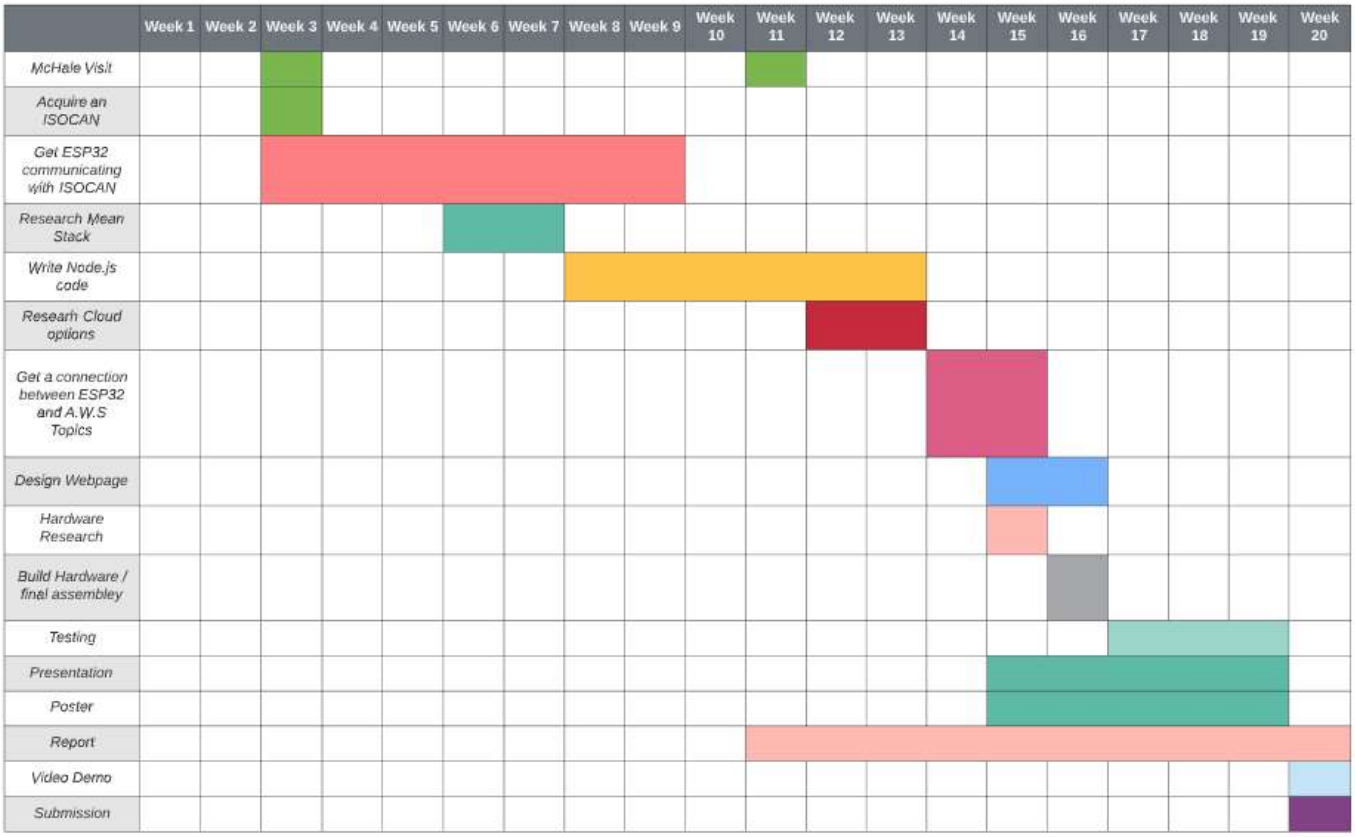
Some Goals for the project itself are:

1. Easy to understand and use by anyone of any background technical or non-technical.
2. Fast data retrieval with little to no lag
3. Make a project that would not require major readjustment to fit other applications e.g. cars, aeronautical.

8 Project Plan

For the period of time between starting the project in September, until December demonstrations, I did not use a Gantt chart. I found this to be a bad idea as I was unsure as to what I was supposed to be doing on any given week. After Christmas I put a project plan in place so that I could achieve certain milestones in the project each week. I found this to work very well as I always knew what was on the agenda on any given week.

I found that this plan changed a number of times due to a variety of factors, however for the most part it was followed.



8.1 Project technologies

The technologies used in my final year project include the following:

- ESP32 – Basis of the project.
- C.A.N transceiver – Interface between ESP32 and E.C.U.
- McHale Fusion 3 ISOCAN and ECU – allowed prototyping and testing.
- MQTT – Used as the messaging protocol from ESP32 to A.W.S Topics.
- AWS IOT Core – Used to communicate between the ESP32 and Node.js.
- M.E.A.N Stack – Used to store and present the data to the user.

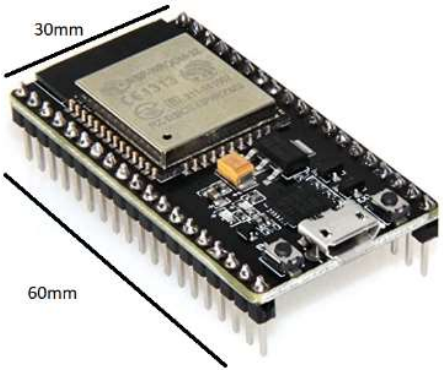
8.1.1 ESP32

I first used the ESP32 while on placement as a tool to improve my knowledge of IOT devices. I thought this microcontroller would be ideal as a basis of my project due to its:

- Size
- Connectivity
- C.A.N. ability

8.1.1.1 Size

The ESP32 is under 60mm x 30mm in size. This small form factor meant that it could neatly be packaged for use in the machine and would not be cumbersome for the user to install.



8.1.1.2 Connectivity

The ESP32 comes as standard with WI-FI and Bluetooth built in. This added capability meant that I did not require an external module which would take up more space and would incur an additional cost. The ESP32 implements TCP/IP, full 802.11 b/g/n/e/i WLAN MAC protocol, and Wi-Fi Direct (Disqus, n.d.)

8.1.1.3 C.A.N ability

Another bonus of using this microcontroller is that is contains a CAN Controller that supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID) of the CAN2.0B specification. The ISOCAN returns messages in the extended format. However the ESP32 does not contain an internal transceiver and therefore I required an external transceiver to operate.

8.1.2 C.A.N

8.1.2.1 What is C.A.N?

C.A.N stands for a Controller Area Network. It is a standard that allows different electronic devices in a vehicle to communicate with each other. Data is sent in a frame, each frame transmits sequentially. If two or more devices transmit at the same time the highest priority device will get precedence (unknown, n.d.).

8.1.2.2 C.A.N Protocol

A C.A.N. protocol is a set of rules and guidelines for communicating data over C.A.N. In my project I use a protocol called J1939.

8.1.2.2.1 J1939

J1939 offers a standardised method for communication across E.C.U.'s. One language across manufacturers. This C.A.N Protocol is most commonly used in the area of heavy vehicles e.g. buses, trucks and agricultural machinery.

J1939 messages are sent in the form of packets. Each packet can have more than eight bytes of data, however the C.A.N specification only supports eight byte data transfers. This is why messages must be sent in multiple packets.

This is evident in my project when the Customer names are sent back from the E.C.U. Any names that are longer than 8 bytes will be sent in many different packets. Because of this a NULL identifier is required to show the end of a customer name.

J1939 differs from other protocols as it only uses extended C.A.N frames. This is also called C.A.N. 2.0B. Extended frames have a 29 bit identifier compared to an 11 bit identifier on light vehicle protocols.

Arduino IDE

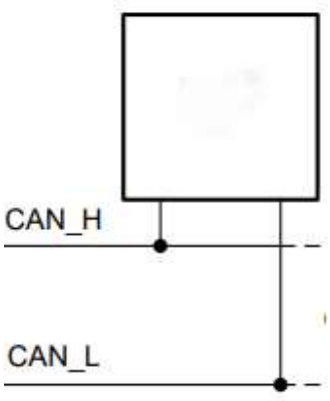
To write the code on my ESP32 I used the Arduino IDE. I chose this is as it is a recommended IDE for use with the ESP32. I have been using this IDE for a number of years, so being familiar with it allowed me to get my code running quite quickly. An issue with this Ide is the lack of a debugger. At certain times I found this challenging.

8.1.2.3 C.A.N Transceiver

A C.A.N transceiver is the interface between a controller area network protocol controller and the physical bus. The transceiver I chose was SN65HVD230. This module was 3v3 compatible which was ideal for the ESP32.

The role of a C.A.N transceiver is simply to drive and detect data to and from the bus. It converts the single ended logic used by the controller to the differential signal transmitted over the bus. It also determines the bus logic state from the differential voltage, rejects the common-mode noise, and outputs a single-ended logic signal to the controller.

The transceiver distinguishes between two bus logic states, dominant and recessive. A recessive bit is defined as CAN_H being less than CAN_L+.5V. A dominant bit is defined as CAN_H being more than CAN_L+.9V. Since dominant bits overwrite recessive bits, CAN manages message collision through the process of bit-wise arbitration (Broyles, n.d.).



8.1.3 McHale Fusion 3 ISOCAN and ECU

8.1.3.1 ISOCAN

ISOCAN Unit

An ISOCAN is an operator interface for control and monitoring applications on modern machinery. It is on this that the Fusion 3 baler information is displayed. This screen would be located in the cab of a tractor and acts as the user input. Before CANview only the operator of the machine could view the settings on the machine, however after CANview has been installed anybody with access rights will be able to view. In the situation where this baler is one of many in a contracting fleet then the fleet manager will be able to see the status of all machines in one place.

8.1.3.2 Electronic Control Unit (E.C.U.)

An electronic control unit is a device responsible for overseeing, regulating and altering the operation of a vehicles electronic system. It is the brains of the operation, and any changes that are required must be done through the ECU.

As this is the brain of the machine it makes sense that this is where we would ask the machine for its current information.

8.1.4 MQTT Protocol

MQTT or Message Queuing Telemetry Transports is a lightweight messaging protocol. MQTT requires a small code footprint and is not majorly dependant on network bandwidth which is good as often the machines which CANview is connected to will be in remote locations where bandwidth is a premium.

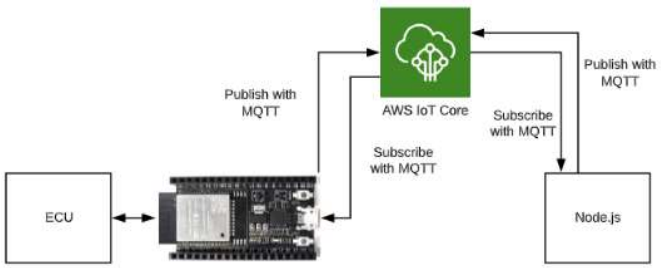
Another benefit of MQTT is that it sends only small packets of data which is great for my system as I will only send exactly what the user requests. MQTT only requires a small amount of power to send packets, this is great as machines are often on the go for hours on end before stopping.

8.1.4.1 MQTT Broker

An MQTT Broker is a server that receives all messages from the clients and then routes the messages to the correct destinations. The MQTT broker I used was AWS IOT Core.

8.1.5 AWS IOT Core

The AWS IOT Core is a cloud service that lets connected devices easily and securely interact with cloud applications and other devices (AWS, n.d.).



MQTT Publish / Subscribe to AWS IOT Core

When a user requests information on the webpage a value is published to the AWS IOT Core broker. As the ESP32 is subscribed to the IOT Core it will be notified of any new messages. This message will tell the ESP32 what the customer wants from the ECU. Once the ESP32 knows what the customer wants it requests it from the ECU. Once the ECU replies with the information the ESP32 publishes back to AWS IOT Core via MQTT. As Node.js is subscribed to the broker it sees this new information and using JavaScript, HTML and CSS it displays the information on a webpage.

8.1.6 M.E.A.N Stack

MEAN stack is a JavaScript software stack for building dynamic web sites and applications. M.E.A.N stands for

1. Mongo. DB
2. Express.js
3. Angular.js
4. Node.js

In my project I have used three out of the 4 above. Mongo. DB, Express.js and Node.js. One of the benefits for me of using MEAN is that it uses JSON (JavaScript Object Notation) which is how I pass my values between ESP32 and AWS and vice versa.

8.1.6.1 Mongo. DB

Mongo.DB is a NoSQL database program that uses JSON documents with schema. NoSQL means "not only SQL" and it gives an alternative to traditional relational databases. NoSQL gives the ability to store any kind of data in a database and can handle a high volume of data.

Login Page

In my project I use Mongo.DB to store customer logins and passwords. When a user attempts to login I will check what the entered against the Mongo.DB to ensure the person is who they say they are. If they are not then they are sent to a "details incorrect" page.

When the Admin successfully logs in he can see all the current customers on the system. This information is taken from a Mongo.DB.

8.1.6.2 Express.js

Express is a web application framework for Node. It provides a number of very useful features for developing web and mobile applications.

```
router.get('/AJ', function(req, res, next) {
  res.render('ActiveJobsPage');
});
```

An example of router.get

One of the main benefits of Express.js is that it provides a method of routing. This refers to how an application responds to a client request. Another useful option with Express is the option of rendering a webpage. As you can see from the figure above when the JavaScript receivers "/AJ" it renders the "ActiveJobsPage" this is a .hbs file. This function renders a view and sends the rendered HTML string to the client.

8.1.6.3 Node.js

Node.js is a server side platform built on Googles v8 Engine. It was created to easily build fast and scalable network applications. It is event driven meaning that it is written to respond to actions generated by the user e.g. a button click. By using Node.js it allowed me to improve my JavaScript of which I had very little done beforehand. Currently I am running the Node.js server on localhost this was useful to test any issues in the code. If this project was to go further I would push my server to an instance of Amazon EC2 making it accessible from anywhere.

Atom.io

I used Atom.io as the IDE for my JavaScript code. This was my first time using Atom.io and I found it very easy and user friendly. Atom also comes with a package manager where you can search and install new packages. One of the packages I used was the "beautify" package. This package allows the user to format there code with correct indentations.

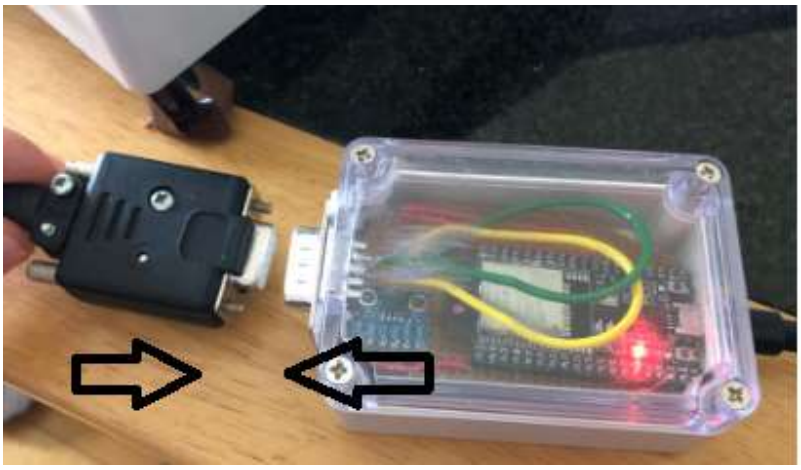
Another benefit of Atom.io was the "Auto-Completion" addition. This was especially useful as I was a beginner when it came to JavaScript. If I was unsure of the ending to a line of code then the IDE would prompt me with a number of options.

Once I had written the Node.js code I then ran it using .BAT files.

9 Key stages of the project

For the purpose of understanding the project I will break it up into 5 stages. At each stage I will explain what is happening from a physical perspective and from a software one.

9.1 Stage 1: Initial Connection



Connect CANview to the E.C.U

Connect the micro SD cable to the back of CANview and to a power source. Connect the front of CANview to the E.C.U.

Once the Micro SD cable is connected to the rear of CANview it will begin trying to connect to the specified Wi-Fi SSID and Password. This is done using the “WiFi.h” library. Once connected to the WI-FI network it will then try to connect to the AWS using:

1. AWS CA Certificate
2. AWS Certificate
3. AWS Private Key

These values are stored in the secrets.h file and allow me to connect to the AWS IOT Core. If a connection can’t be made to either the Wi-Fi or A.W.S then the ESP32 will print “.” to the serial terminal every 500ms until a connection is made. Once a connection is made to A.W.S it subscribes to an A.W.S IOT Topic which has been specified at the top of the code. Once Subscribed to a topic the ESP32 waits until the user publishes something to the topic.

```
//wifi
#include <WiFiClientSecure.h>
#include "WiFi.h"
```

```
void setup() {
  Serial.begin(115200);
  connectAWS();//Sets up Wi-Fi and AWS connections
```

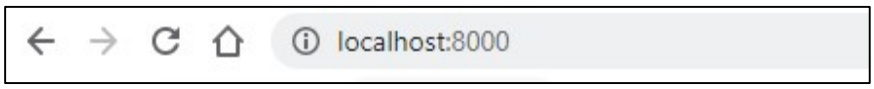
```
void connectAWS()
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.println("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  net.setCACert(AWS_CERT_CA);// Configure WiFiClientSecure to use the AWS IoT device credentials
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  client.begin(AWS_IOT_ENDPOINT, 8883, net);// Connect to the MQTT broker on the AWS endpoint we defined earlier

  client.onMessage(messageHandler);// Create a message handler
  Serial.print("Connecting to AWS IOT");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  if (!client.connected()) {
    Serial.println("AWS IoT Timeout!");
    return;
  }
  client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC); // Subscribe to a topic
  Serial.println("AWS IoT Connected!");
}
```

```
// The MQTT topics that this device should publish/subscribe
#define AWS_IOT_PUBLISH_TOPIC   "myTopic/1"
#define AWS_IOT_SUBSCRIBE_TOPIC "myTopic/2"
```

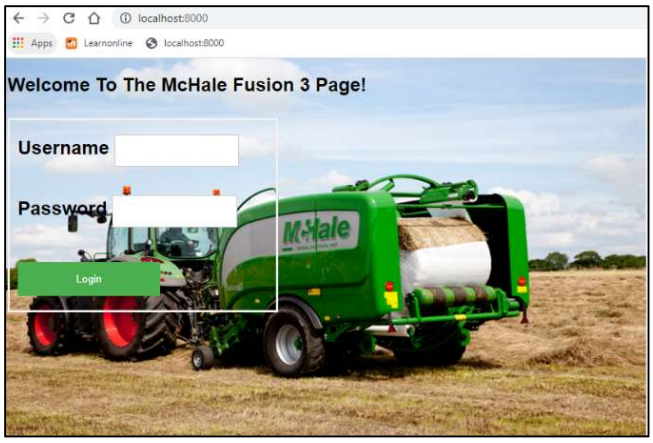
9.2 Stage 2: Connection and login to website

Once CANview is connected to the E.C.U. the next step is to access the webpage. Currently the webpage is running on localhost meaning that CANview must be on the same Wi-Fi network as Node.js.

First access localhost.



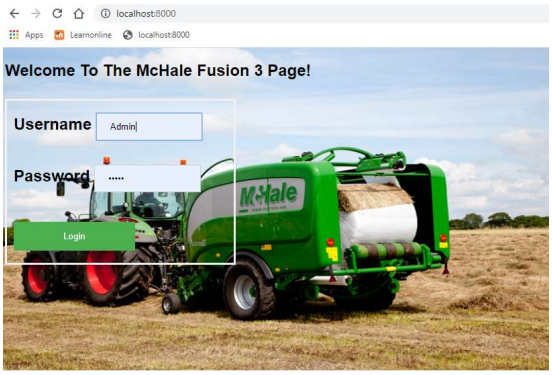
The user will be brought to the below page:




```
<body>
  <h2>Welcome To The McHale Fusion 3 Page!</h>
    <br><br>

    <form action="/login" method="post">

      <div class="container">
        <label for="uname"><b>Username</b></label>
        <input type="text" name="uname"><br>
      </div>
      <div class="container">
        <label for="psw"><b>Password</b></label>
        <input type="password" name="psw"><br><br>
        <button type="submit">Login</button>
      </div>
    </form>
</body>
```

This page is created using HTML and CSS

If the User trying to access the webpage they will follow the below steps:




```
<form action="/login" method="post">
```

Enter the Username and the password and press “Login”. When login is pressed an Express.js routing method is used to direct the entered username and password to a router.post function. Once here the entered information is added to a variable called “user”.

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

```
var attemptedUser = {
  name: "",
  password: "",
  setDetails(name, password) {
    this.name = name;
    this.password = password;
  }
};
```

The database collection “Users” is then queried with the name of the user that was entered. Once this information is got from the database it is put into an array called “result”. Another variable called “attemptedUser” is set with the users name and password. This attemptedUser variable will be used to identify the current user that is logged in.

```
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;
    attemptedUser.setDetails(result[0].User, result[0].Password);
  });
});
```

Now that we have the details of the user that is trying to access our webpage and the details of the user with that name in our “Users” database we can check if their credentials match up.

```
if (user.name == attemptedUser.name && user.password == attemptedUser.password) {
  loggedIn = true;
```

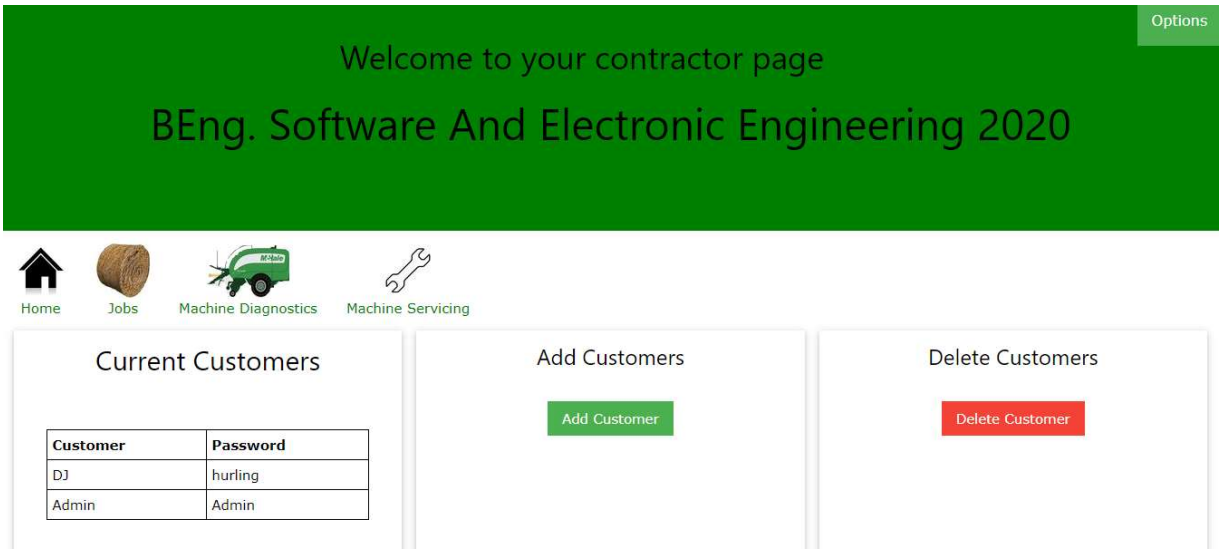
If the username happens to be equal to “Admin” the user will be directed to the Admin Page, if not they are directed to the customer page. If the details entered are not equal to a person in the database they are directed to an error page where they are told that their details are incorrect. After this the database that we connected to is closed.

```
  if (user.name == "Admin") {
    res.render('adminPage', {
      items: result
    })
  } else {
    res.render('customerPage', {
      title: "My Page",
      username: user.name
    });
  }
} else {
  res.render('error', {
    message: "Details incorrect",
    username: user.name,
    password: user.password
  });
}
```


```
  }
  db.close();
});
```

9.3 Stage 3: Navigating the Admin Page

Once the admin has logged in successfully they will be greeted with the following webpage.



Like the login page this page utilises HTML and CSS in its design. The style of the webpage is based on stylesheets from w3schools.

Current customers are displayed on the left hand side tab. This shows the current users on the system. This information is retrieved similar to the login page information.

A connection is made to the collection “Users”. All username and passwords are added to an array called “result”. This array is returned to the .hbs file using handlebars in the form of a table.

```
router.get('/AP', function(req, res, next) {
  MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("userDb");
    var arr = dbo.collection("Users").find().toArray(function(err, result) {
      if (err) throw err;
      res.render('adminPage', {
        items: result
      });
    });
  });
});
```

```
<h2>Current Customers</h2>
<br><br>
<table style="width:90%; font-size: 15px; height:70%; " align="center">
  <tr>
    <th>Customer</th>
    <th>Password</th>
  </tr>
  {{# each items }}
    <tr id="Row">
      <td>{{this.User}}</td>
      <td>{{this.Password}}</td>
    </tr>
    {{/each}}
</table>
```

9.3.1 Logging out of the Admin Page

In the Admin Page I have added a “LogOut” button. This button is located in the “Options” drop down menu at the top right of the webpage. Once pressed the user is brought back to the login page.

Also on the Admin page are the Add and Delete Customer functions. The Form to enter the name and password is not visible at first, however when either button is pressed it becomes visible.

9.3.2 Add a new Customer

To add a new customer to the database you must enter the username of the new person along with a password for them. Once entered into the form click “Save User”. Once this button is clicked the credentials are routed using Express.js to “/insert” in the JavaScript file.

```
<form action="/insert" method="post" style="display:none" id="addForm">
```

Once the credentials are received on the JavaScript side the username and password are assigned to a JSON object called “item”. This object is then insert into the database using the “insertOne” function.

```
router.post('/insert', function(req, res, next) {
  var item = {
    User: req.body.username,
    Password: req.body.psw
  };
  console.log(item);
  MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("userDb");
    dbo.collection("Users").insertOne(item, function(err, result) {
      assert.equal(null, err);
      console.log('Insertion Successfull');
    });
```

9.3.3 Delete a Customer from the database

To delete a customer to the database you must enter the username of the new person. Once entered into the form click “Delete”. Once this button is clicked the credentials are routed using Express.js to “/delete” in the JavaScript file.

```
<form action="/delete" method="post" id="deleteForm" style="display:none">
```

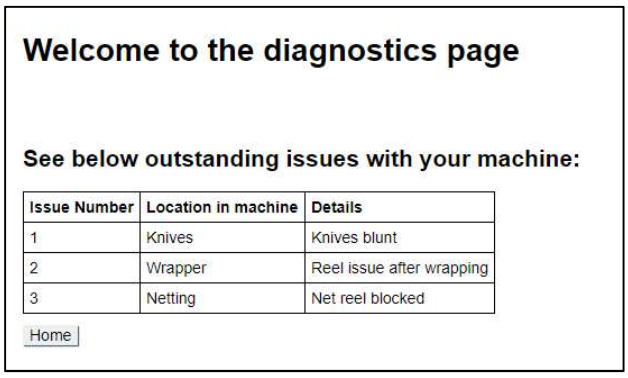
Like the “Add Customer” button, once the credentials are received on the JavaScript side the username and password are assigned to a JSON object called “item”. This object is then deleted from the database using the “deleteOne” function.

```
router.post('/delete', function(req, res, next) {
  var item = {
    User: req.body.User1,
  };
  MongoClient.connect(url, function(err, db) {
    if (err) throw err;
    var dbo = db.db("userDb");
    dbo.collection("Users").deleteOne(item, function(err, result) {
      assert.equal(null, err);
      console.log('Deletion Successfull');
    });
    dbo.close();
  });
```

9.3.4 Navigating useful pages

Also on the Admin Page are two links to “Machine Diagnostics” and “Machine Servicing”. I have added these pages to the webpage as I believe they would be useful for the user.

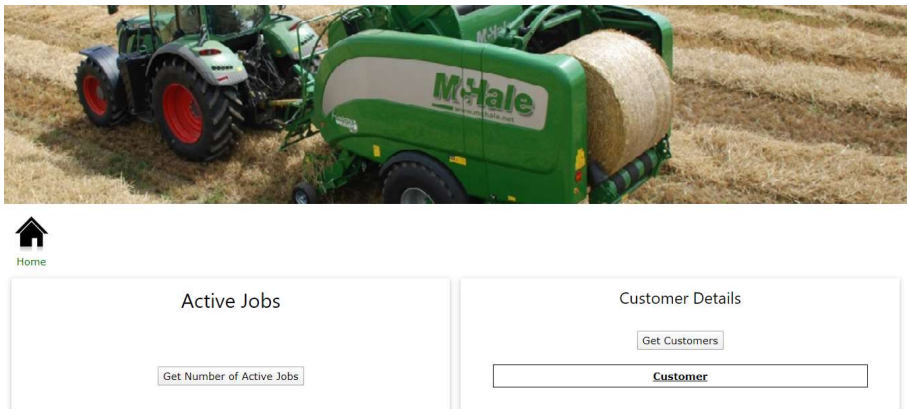
If the project was to be developed further then this webpage could become the control centre for a farmer’s whole fleet of machines. So although they may not seem necessary at the moment, they are part of a bigger scalable plan. Both pages contain a “Home” button where the Admin can return to the main Admin Page.



9.4 Stage 4: Accessing Active Jobs details on the ISOCAN

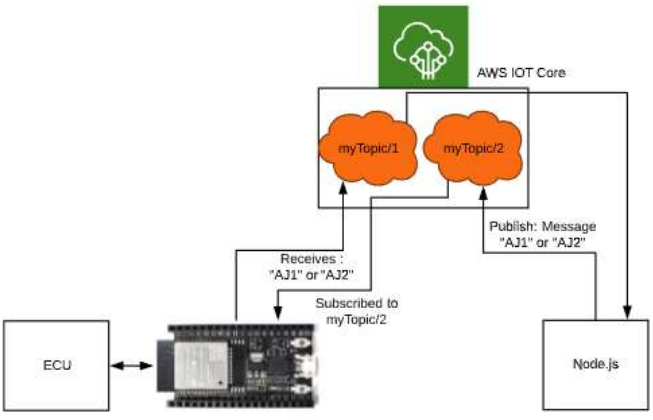
The key component of the project is the Active Jobs Page. The Admin can access this page by clicking on the “Jobs” icon in the top navigation.

Once in this page the Admin is greeted by the following page. To access the names of the current customers on the ISOCAN system the Admin must select the “Get Customers” button.



9.4.1 Requesting Customer Names or Active Jobs Number (Node.js side)

Once the “Get Customers” button is pressed the user is routed using Express.js to the “/Mqtt3” section of the JavaScript file. Here we publish a JSON object to the A.W.S. myTopic/2. The JSON object contains the message “AJ1”. When “Get Number of Active Jobs” button is pressed a function called “loadDoc()” is called. This uses a GET request routed to “/Mqtt” section of the JavaScript file. Here we publish a JSON object to the A.W.S. myTopic/2. The JSON object contains the message “AJ2”.



As our ESP32 is subscribed to the above topic it gets notified that a new message has been sent to the topic.

9.4.2 Receiving a Request for Customer Names or Active Jobs Number (Arduino side)

During the Wi-Fi / AWS setup a message handler is also set up. It is part of the ArduinoJSON library. This is used to intake the message from the A.W.S topic.

```
void messageHandler(String &topic, String &payload) {
  Serial.println("incoming: " + topic + " - " + payload);
  StaticJsonDocument<200> doc;
  deserializeJson(doc, payload);
  const char* message = doc["message"];
  String mystring(message);
  Serial.println(mystring);
  if (mystring == "AJ1") {
    Serial.println("YES AJ1");
    RequestActiveJobs();
  } else if (mystring == "AJ2") {
    Serial.println("YES AJ2");
    RequestActiveJobs();
  }
```

```
else {
  Serial.println("NO");
}
```

The incoming JSON is de-serialized using the deserializeJson function. An if statement then checks if the message is equal to “AJ1” or “AJ2”. If the incoming message is not equal to “AJ1” or “AJ2” then “NO” is printed to the serial terminal. However if the message is equal to “AJ1” or “AJ2” the “RequestActiveJobs() function is called.

This function sends an extended C.A.N. frame to the ISOCAN. The frame looks as such:

0x19FF5003 00 FF FF FF FF FF FF FF

This tells the ISOCAN that I am looking for all the active job information on its system.

```
void RequestActiveJobs() {
  //Send message to ISO CAN terminal
  CAN_frame_t tx_frame;
  tx_frame.FIR.B.FF = CAN_frame_ext;
  tx_frame.MsgID = 0x19FF5003; // ISOCAN Address
  tx_frame.FIR.B.DLC = 8; // data length code
  tx_frame.data.u8[0] = 0x00; //Request active jobs ID
  tx_frame.data.u8[1] = 0xFF;
  tx_frame.data.u8[2] = 0xFF;
  tx_frame.data.u8[3] = 0xFF;
  tx_frame.data.u8[4] = 0xFF;
  tx_frame.data.u8[5] = 0xFF;
  tx_frame.data.u8[6] = 0xFF;
  tx_frame.data.u8[7] = 0xFF;
  ESP32Can.CANWriteFrame(&tx_frame);
  Serial.println("Sending 0x00 to ECU");
}
```

In my loop I am constantly looking for any new incoming CAN frames. When a new frame is received it is added to a queue.

When reading / writing to an E.C.U a message I.D. is required. When I write to the E.C.U I use the Message I.D. “0x19FF5003”. When reading from the E.C.U the information which I want will come from the address “0x19FF5102” or “0x19FF5202”. First I check that this is the case, if I didn’t do this check then I would receive thousands of messages from the machine. If a new frame is received from this address it is then checked to see if it is a standard or extended frame. All frames on my system should be extended as they run on the J1939 protocol.

```
CAN_frame_t rx_frame;
if (xQueueReceive(CAN_cfg.rx_queue, &rx_frame, 3 * portTICK_PERIOD_MS) == pdTRUE) {
  if (rx_frame.MsgID == 0x19FF5102) {//If a message from this message id comes over CAN then do something with it
    if (rx_frame.FIR.B.FF == CAN_frame_std) {
      printf("New standard frame\n");
    }
    else {
      printf("New extended frame ");
    }
```

9.4.3 Getting the Active Jobs Number from the E.C.U.

The first message received back from the E.C.U after requesting the active jobs info will begin with “00” this message will give the number of active jobs on the system. Once received the value at rx_frame.data.u8 [2] will be equal to the number of active jobs. This value is then published to A.W.S “myTopic1” using the “PublishActiveJobsNo()” function. This function puts the Active jobs value into a JSON object before sending it.

```
if (rx_frame.data.u8[0] == 0 && rx_frame.data.u8[1] == 0) {
  int NumberOfActiveJobs = rx_frame.data.u8[2];
  PublishActiveJobsNo(NumberOfActiveJobs);
  Serial.println("Number of Active Jobs:" + (String)NumberOfActiveJobs);
  for (int i = 0; i <= 7; i++) {
    Serial.print(rx_frame.data.u8[i], HEX);
  }
  Serial.print("\n");
}
```

```
New extended frame from 0x19ff5102, DLC 8
Number of Active Jobs:5
005FFFFFFFFF
```

```
void PublishActiveJobsNo(int x) {
  StaticJsonDocument<200> doc;
  doc["Active_Jobs"] = x;
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer); // print to client
  client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}
```

9.4.4 Getting the Customer Names from the E.C.U.

After the first message comes in the next messages to follow will be messages containing the:

- Job ID
- Job Number
- Customer ID

In order to get the Customer name we take the Customer ID and using the GetCustName() function we request this information from the E.C.U.

```
//every other message after the first will come in here
else {
  CustomerID = rx_frame.data.u8[4];
  GetCustName(CustomerID); // takes the customer ID from the incoming frame and sends to GetCustName.
}
```

```
void GetCustName(int N) {//Intakes customerID and sends it to the E.C.U
  //Send message to ISO CAN terminal
  CAN_frame_t tx_frame;
  tx_frame.FIR.B.FF = CAN_frame_ext;
  tx_frame.MsgID = 0x19FF5003;
  tx_frame.FIR.B.DLC = 8;
  tx_frame.data.u8[0] = 0x01;
  tx_frame.data.u8[1] = 0x00;
  tx_frame.data.u8[2] = N;  //inserts the customer ID
  tx_frame.data.u8[3] = 0xFF;
  tx_frame.data.u8[4] = 0xFF;
  tx_frame.data.u8[5] = 0xFF;
  tx_frame.data.u8[6] = 0xFF;
  tx_frame.data.u8[7] = 0xFF;
  ESP32Can.CANWriteFrame(&tx_frame);
  Serial.println("Sending GetCustInfo to ECU");
}
```

When the E.C.U responds to this message it will come from the address “0x19FF5202”. Once the frame is received The ESP32 loops through the 8 bit messages until a ‘\0’ NULL terminator is found. This terminator denotes the end of a customer’s name and when reached the if statement is exited and the name is published to A.W.S using the “PublishActiveCustNames()” function. This continues until all current names are published to A.W.S.

```
//0x19FF5202 returns customer name information
else if (rx_frame.MsgID == 0x19FF5202) {
    for (int i = 2; i <= 7; i++) {
      if (rx_frame.data.u8[i] != 0) {
        ReturnedCustInfo = ReturnedCustInfo + (char)rx_frame.data.u8[i];
      } else {
        PublishActiveCustNames(ReturnedCustInfo);//publish to aws
        Serial.print(ReturnedCustInfo);
        ReturnedCustInfo = "";
        break ;
      }
    }
    Serial.print("\n");
}
```

```
void PublishActiveCustNames(String x){
  StaticJsonDocument<200> doc;
  doc["Cust_Name"] = x;
  char jsonBuffer[512];
  serializeJson(doc, jsonBuffer); // print to client
  client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
}
```

9.4.5 Storing and presenting the Customer names on Node.js

Once the data has been published to “myTopic/1” Node.js is then notified. The incoming payload is let equal to the variable “message”. This message is then parsed using “JSON.parse”. The JSON objects are added to an array using the “push()” function.

```
var obj;
var arr = [];
let iii = 0;
device
  .on('message', function(topic, payload) {
    iii++;
    var message = payload.toString();
    obj = JSON.parse(message);
    arr.push(obj);
```

To return the customer names to the webpage the first element of the array is ignored using the “shift()” function. This is done as the first element is the number of active jobs and we don’t want this to be displayed in our customers section. The webpage is rendered using the Express.js function “res.render” and the array is sent across. Once sent the array is set back to zero.

```
arr.shift();
 res.render('ActiveJobsPage', {
   arr: arr
 })
arr = [];
 iii = 0;
```

On the webpage side handlebars are used to display the array, similar to how the current customers are displayed on the Admin page.

```
<table style="width:90%; font-size: 15px; height:70%; " align="center">
<tr>
  <th>Customer</th>
</tr>
{{#each arr}}
<tr>
  <td>
    {{this.Cust_Name}}
  </td>
</tr>
{{/each}}
</table>
```

9.4.6 Storing and presenting the Number of Active Jobs on Node.js

To return the Number of Active Jobs to the webpage instead of rendering a new page we return an Ajax using “res.json”. What we didn’t want when returning the customer information we do want now as the first element of the array is the Active Jobs number. After it is sent the array is set back to zero.

```
  res.json(arr[0]);
  arr = [];
});
```

On the webpage side this value got by using the “.getElementById()” function.

```
<div id="demo">
```

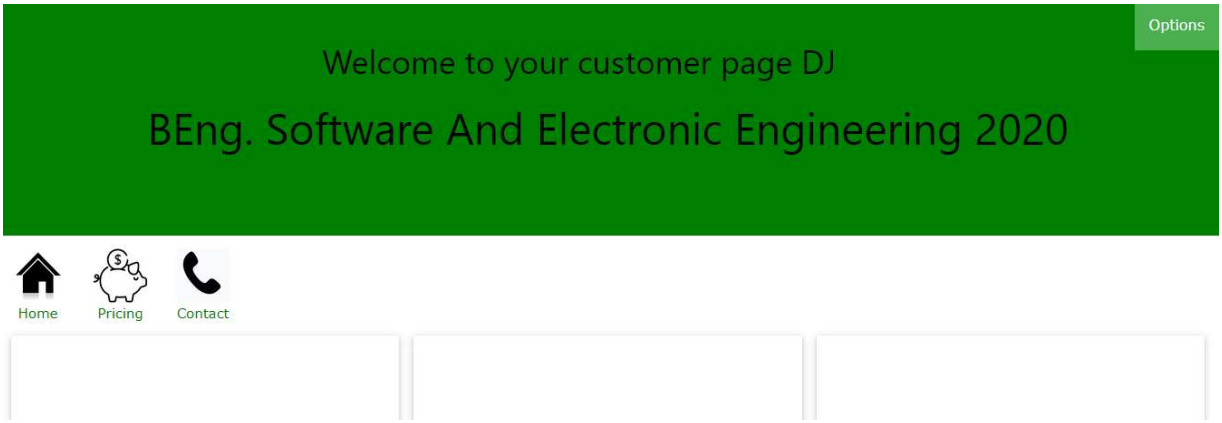
```
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {

      myobj = JSON.parse(this.responseText);
          var Active_Jobs = myobj["Active_Jobs"];
          document.getElementById("demo").innerHTML =  "Number of Active Jobs: " + Active_Jobs;
    }
  };
  xhttp.open("GET", "/Mqtt", true);
  xhttp.setRequestHeader("content-type","application/json");
  var myObj={
    Active_Jobs:""
  };
  xhttp.send(JSON.stringify({}));
}
</script>
```

9.5 Stage 5: Navigating the Customer Page

9.5.1 Login

Before a user can login to their respective customer page the Admin must first add them to the system. Only the Admin can do this and it is done through the “Add customer” section on the Admin Page. The Admin gives them a username and password with which they can login with.

After the credentials are verified by the Mongo. DB database the user accesses the following screen.



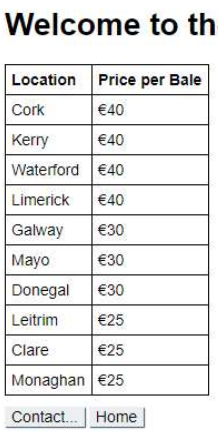
9.5.2 Navigating the useful pages

As you can see the Customer page doesn't have the same level of functionality as the Admin page. This is due to the fact that we don't want a customer to be able to view private information such as the Machine Diagnostics, Machine Servicing or Active Jobs page. We just want to give them a place where they can find out more about our pricing and our contact information.

If this idea was to be developed further the customer page could be made more interactive and unique for each customer.

9.5.2.1 Pricing Page

Here the customer can find out more information on the pricing of bales per each county.



9.5.2.2 Contact Page

Here the customer can find out more information about how to contact the contracting company.

10 Conclusion

I found this project challenging in parts but overall I have enjoyed it and learned a lot of skills in many areas. Projects that access C.A.N information are not very common and thus there are not many resources available. I am proud that I took on the challenge and for the most part succeeded in what I set out to do.

This project has given me a better understanding of how to move a project from the design stage to the final presentation. Throughout this transition I have picked up numerous skills which I'm sure will be of benefit to me in my working life.

After first picking this project for a number of weeks I was overwhelmed, self-doubting myself almost led me to throwing in the towel. However with help from McHale Engineering, lecturers and classmates I slowly built up my confidence once again and turned a negative situation in to a positive one.

Having gone through these dark times I am immensely proud of the final outcome. I started out with the idea of making a product that was user friendly and adaptable to other situations and I think I have achieved this ambition.

I believe this product could be used to further ease the pressure on the modern day farmer and prove to them that technology is not to be feared but to be embraced. I would love to develop this project further to create a standalone system for agricultural fleet management.

11 References

AWS,n.d.*AWS.*[Online]
Available at:https://aws.amazon.com/iot-core/
[Accessed 15 May 2020].

Broyles,S.,n.d.*A System Evaluation of CAN Transceivers(Rev.A).*[Online]
Available at: http://www.ti.com/lit/an/slla109a/slla109a.pdf?HQS=slla109-aaj&ts=1589457435700
[Accessed 14 May 2020].

Disqus,n.d.*ExplorEmbedded.*[Online]
Available at: https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F
[Accessed 14 May 2020].

unknown, n.d. *wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/CAN_bus
[Accessed 16 May 2020].

12 Appendices

12.1 Code

The source code for the project can be found on my GitHub using the following link:

https://github.com/dwhelan97/ServiceMonitoringSystemFYP