21-07-2019

Dashboard Overhaul

The need to have a clean and simple dashboard is very important. An unfamiliar user must be able to use and understand the dashboard easily and with little explanation. Currently, the dashboard produced during the Telstra Innovation Challenge (TIC) does not meet these requirements for the following reasons:

1. User is unable to view multiple sensor readings and charts at the same time.
2. To view other charts, the user must click on a button to show the chart.
3. The location coordinates are not easily understood (longitude and latitude).
4. The map page does not have controls to zoom, pitch, rotate, or change the theme.
5. The user icon on the map is hard to find.

To address these points, the following features for the new dashboard must be met:

1. The dashboard must be simple and clean. All sensor readings are shown under their respective title and accompanied by the percentage change and the historical graph.
2. All charts must be visible.
3. The location coordinates must be shown with a map to show where it is currently.
4. New map controls are zoom, pitch, rotate, and change theme.
5. New map control centre on user must be added.

The new dashboard design mock-up:

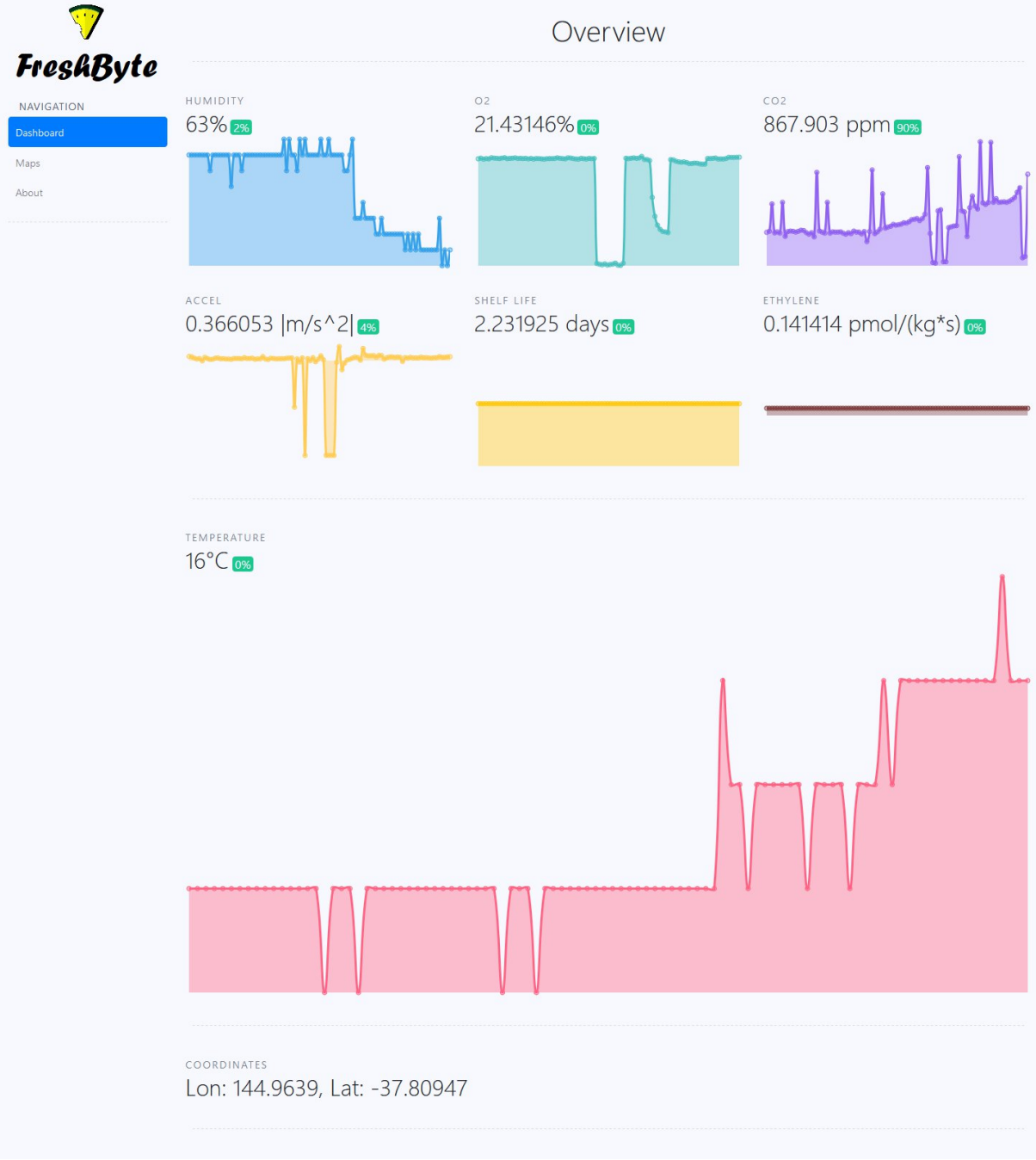
<table>
<tr><td>Navigation Panel</td><td>Overview
<table>
<tr><td colspan="2">Temperature</td><td colspan="2">Humidity</td><td colspan="2">O2</td></tr>
<tr><td>20°C</td><td>5%</td><td>80%</td><td>10%</td><td>20.1°C</td><td>1%</td></tr>
<tr><td colspan="2">Graph</td><td colspan="2">Graph</td><td colspan="2">Graph</td></tr>
<tr><td colspan="2">CO2</td><td colspan="2">Accel</td><td colspan="2">Shelf Life</td></tr>
<tr><td>800 ppm</td><td>10%</td><td>0.9 |m/s^2|</td><td>1%</td><td>3 days</td><td>0%</td></tr>
<tr><td colspan="2">Graph</td><td colspan="2">Graph</td><td colspan="2">Graph</td></tr>
<tr><td colspan="2">Ethylene</td><td colspan="2">Coordinates</td><td colspan="2"></td></tr>
<tr><td>0.14 pmol/(kg*s)</td><td>0%</td><td colspan="2">Lon: 1, Lat: -2</td><td colspan="2"></td></tr>
<tr><td colspan="2">Graph</td><td colspan="2">Azure Map</td><td colspan="2"></td></tr>
</table></td></tr>
</table>

The new map design mock-up:

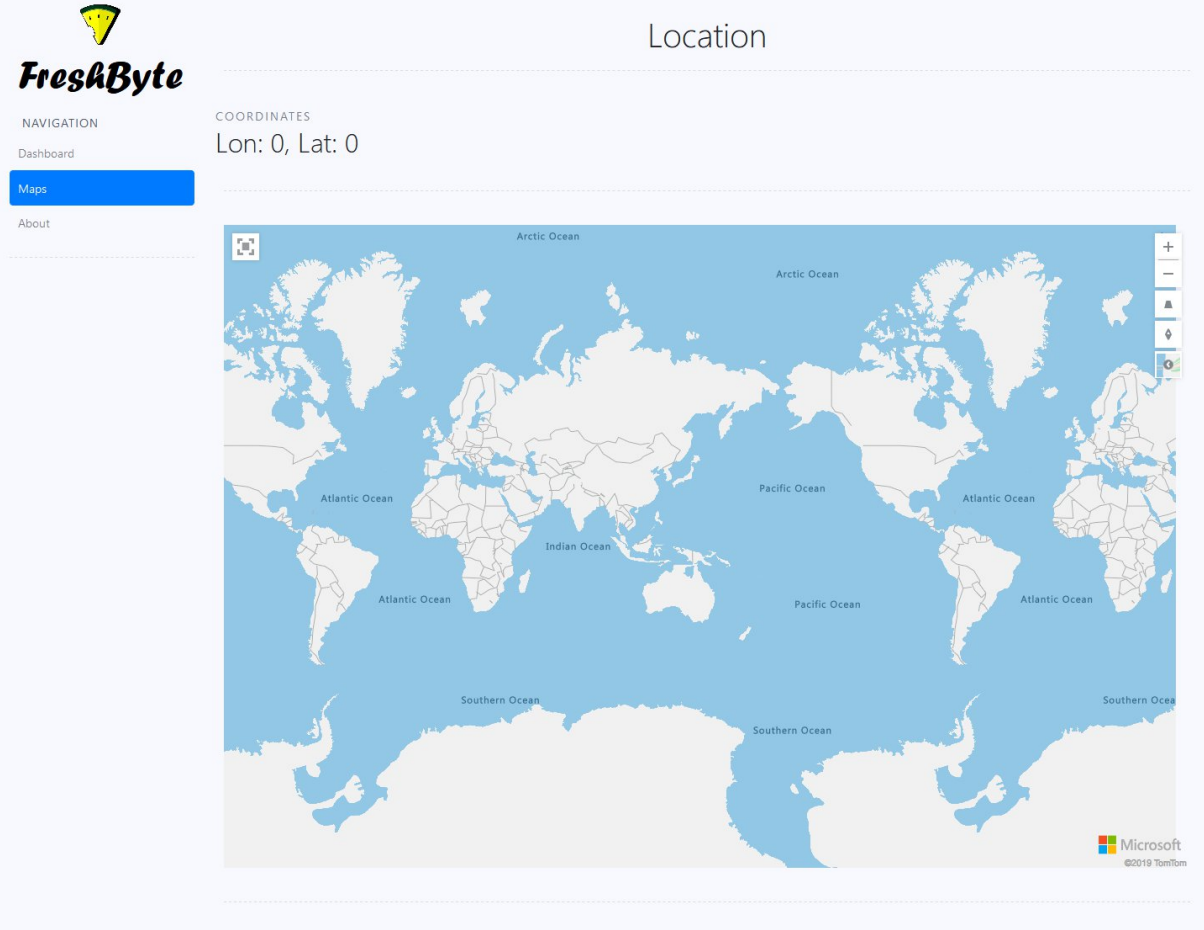
<table>
<tr><td rowspan="3">Navigation Panel</td><td>Overview</td></tr>
<tr><td>Coordinates
Lon: 1, Lat: -2</td></tr>
<tr><td>Azure Map</td></tr>
</table>

Tomorrow, I will implement a basic working design based on the above mock-ups.

22-07-2019
Basic Dashboard Design Implementation
I have deviated a little from the dashboard design mock-up. The temperature occupies the whole width of the page. Since temperature is the most important reading, a large portion of the space is dedicated to it.
Each sensor reading has a different graph colour. The deltas (percentage change) of the sensor readings change colours based on if it increases (green) or decreases (red).



The map implementation has strictly followed the map design mock-up. The map can be centred on the used by clicking the top left icon. The top right icon panel contains the following controls: zoom in, zoom out, increase/decrease pitch, rotate left/right, and change theme.



24-07-2019
First Meeting with Sam For Second Semester
Sam was debriefed on what happened during the TIC finals at Telstra's headquarters. The public and university streams where discussed and compared to our own pitch.

With the TIC over, we must now focus on refining our hardware and software. Before we continue further, we need to discuss and decide on our goals, deadlines, and changes. Currently our goal is to have a working product with the power issues fixed by EnGenius. The deadlines are to solve and package the product before EnGenius. And the changes are:

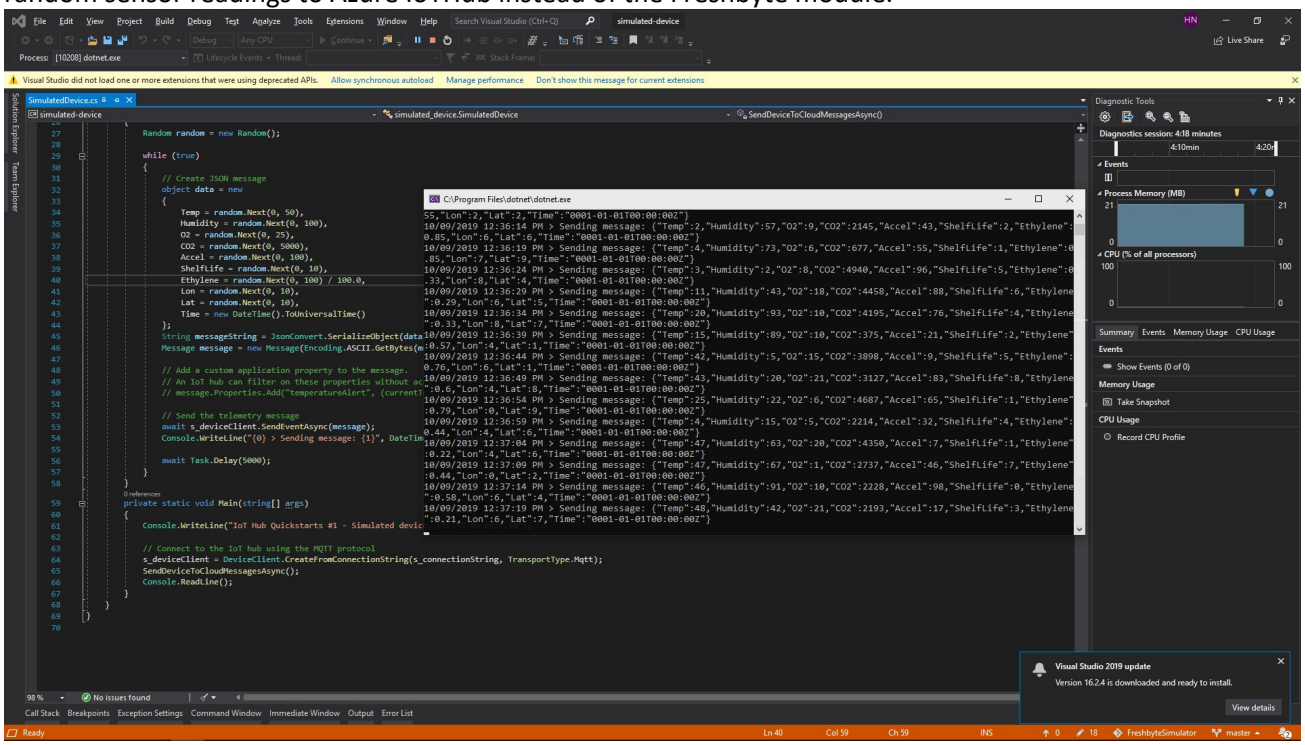
- Implement a working basic data analytics page.
- Overhaul the backend to be more reliable.
- Move to board that is much more reliable that the MKR NB 1500.
- Address the power consumption issue.
- Implement a database to store the readings.

Before we meet Sam again for the next meeting, we need new ideas, and a database needs to be implemented and working. I don't know when this will be completed, but the code must be cleaned up before I can continue.

28-07-2019
General Code Clean Up and Simulated Device
The dashboard board code has been cleaned up to prepare for implementing the database and simple data analytics.

Instead of relying on the Freshbyte module to provide sensor readings, I can use software to simulate the Freshbyte device sending readings instead. This saves me time and I can use the simulated device whenever I need to.

A simple example from https://azuremapscodesamples.azurewebsites.net/ is modified to send random sensor readings to Azure IoTHub instead of the Freshbyte module.



29-07-2019
Map Location Pulse
Today I will implement a pulsing icon to indicate where the user is currently.
To do this I will use code from https://azuremapscodesamples.azurewebsites.net/. The pulsing animation is provided by the following CSS:

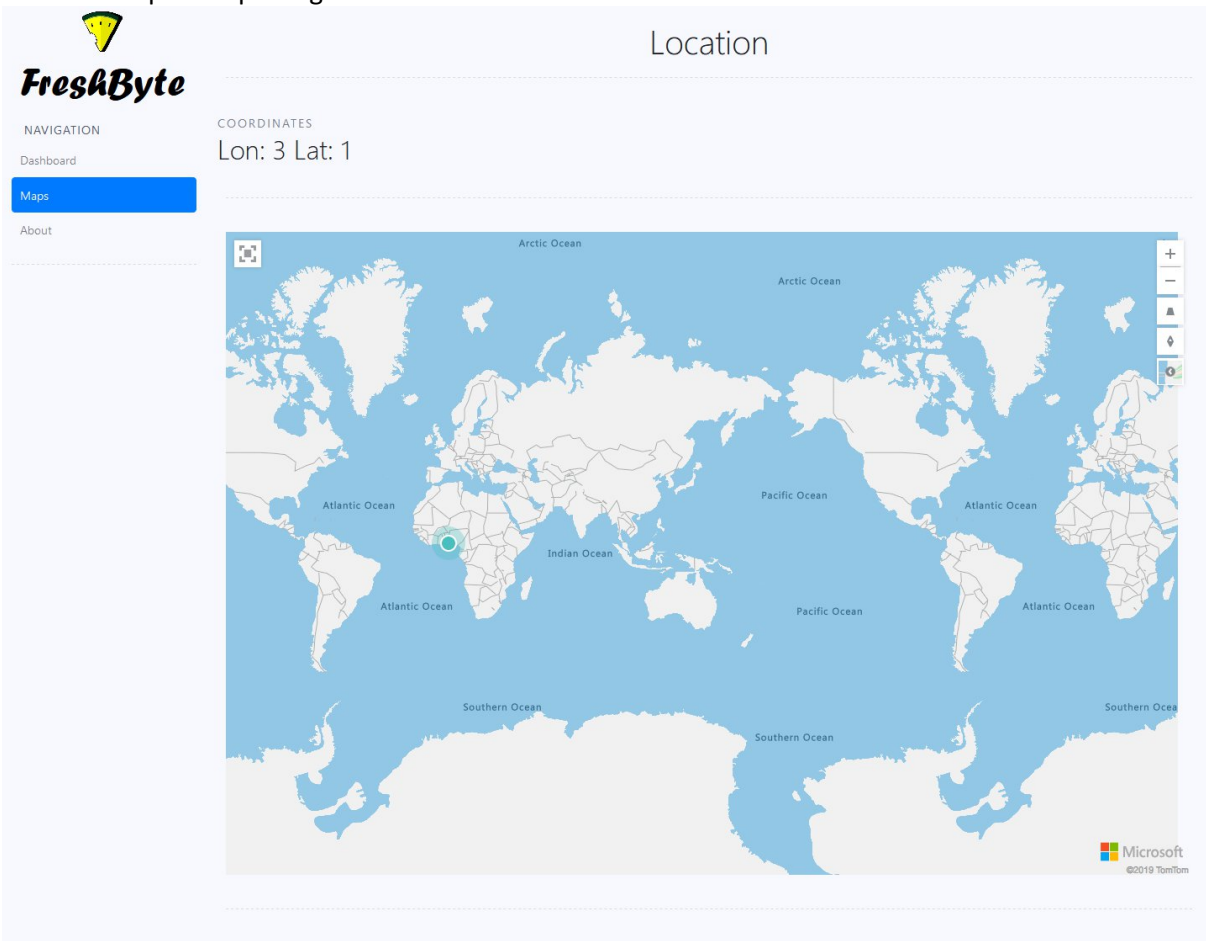
```
.pulseIcon {
    display: block;
    width: 20px;
    height: 20px;
    border-radius: 50%;
    background: rgb(73, 191, 191);
    border: 2px solid white;
    cursor: pointer;
    box-shadow: 0 0 0 rgb(73, 120, 120);
    animation: pulse 3s infinite;
}

.pulseIcon:hover {
    animation: none;
}

@keyframes pulse {
```

```
  0% {
    box-shadow: 0 0 0 0 rgba(73, 191, 191, 0.4);
  }
  70% {
    box-shadow: 0 0 0 50px rgba(73, 191, 191, 0);
  }
  100% {
    box-shadow: 0 0 0 0 rgba(73, 191, 191, 0);
  }
}
```

To add this to the map, we will use the following code which removes all map markers and adds in a new one with the updated coordinates:

```
map.markers.remove(user_position_marker);
            user_position_marker = new atlas.HtmlMarker({
              htmlContent: '<div class="pulseIcon"></div>',
              position: [obj.Lon, obj.Lat]
            });
            map.markers.add(user_position_marker);
```

Now the map has a pulsing icon at the user location:



I now need to think about how to handle the hardware and address the power issue. One way to address the power issue, is to have two power sources: one for the sensors, and one for the board.

Another solution is to have a power supply that can deliver the required power needed to driver all the sensors and the device. We can further extend this by putting the device and its sensors into sleep when it is not sending data.

31-07-2019
Meeting Up with Li and Jong To Discuss Our Direction and Power Issue
It was a heated debate where in the end we decided to:

1. Aggregate all the sensors into a single board with a separate power supply.
2. Data backlogging to be stored locally when there is no connection.
3. Test and measure the power draw and current draw.

Other things that we discussed were:

1. LiPo's function very poorly below 0 degrees Celsius.
2. Risk of data loss due to a loss connection. Is it worth implementing the feature to backlog data? Is the time and cost to battery life worth it?
3. Should we store all the measurements locally and only send it all once the journey is complete. Is this an IoT device when such a behaviour is implemented?
4. We can put the GPS into low power mode when not used.
5. Should we print a PCB to aggregate all the sensors?
6. Change to target client from Linfox to farmers. We no longer have a need to target Linfox with the TICs completed.

Before we meed Sam, we should think about ordering and finding parts for Sam to order.

01-08-2019
Added Data Analytics Page to Navigation Panel
Today, I have prepared and deployed an empty page named Data Analytics. I shouldn't have called this Data Analytics; the implementation will be a Data Visualisation not Analytics.
When we meet Sam tomorrow, we should tell him our ideas to address the power issue.

02-08-2019
Second Meeting with Sam
We have proposed two ideas to address the power issue:

1. Have two separate power supplies.
2. Reduce the number of sensors used.

To further reduce the power consumption, we discussed putting the device to sleep and only waking up and sending data very 30 minutes. Is there a concern about losing data or missing important opportunity to record an important event such as shock?
We talked to Sam about our idea to aggregate all the sensors onto a single board, and to use a sperate power source. This provides adequate power to all the sensors and board. We also talked about thermally insulating the gas sensors to preserve power heating them and reduce the time to heat up to operating temperatures. Sam showed us a board where he used a perf board that has multiple things wired together.
We need to think of a way to get the device working on WiFi reliably, and measure the power draw of the sensors and the device.
Sam has also agreed to change the target audience to farmers.

03-08-2019
Task Delegation and Draft A Completion Plan
We met up as a group to delegate tasks to team members and to start a draft of a completion plan.
Others – Start on the report and draft.
Me – Focus on optimising server-side code and UI. Then help with the draft.
We need to create a script, content, and B-Roll for the EnGenius 2-minute pitch. This is an opportunity to show off our project.

04-08-2019
Prepare Dashboard for Open Day and Location Spoof
Before Open Day, I need to have a basic working Data Analytics page. The device needs to be working reliably.

I have implemented a basic way to spoof location. We have an array to store the coordinates from RMIT Building 10 to Melbourne Central. Each time we send a message, we will send a coordinate from the two arrays: coordsLat and coordsLon. There are seven coordinates, and we will loop through them. Once we have sent the last coordinate, we will go back to the start.

```
// Spoof GPS coordinates.
const uint8_t mapSize = 7;
uint8_t mapCount = 0;
const double coordsLat[mapSize] = { -37.81012, -37.81012, -37.809734, -37.809653, -37.809475, -37.808996, -37.80819};
const double coordsLon[mapSize] = {144.963921, 144.963921, 144.963774, 144.964002, 144.963912, 144.963651, 144.963318};

...
  const double lat = coordsLat[mapCount];
  const double lon = coordsLon[mapCount];
...
  // Rotate coordinates.
  if (mapCount++ >= mapSize) {
    mapCount = 0;
  }
...
```

06-08-2019
Using an Arduino Nano (Slave) with a Wemos D1 Mini (Master)
Today in the morning, I will attempt a temporary fix to the power issue that I thought of last night. With what I have now and before Open Day, I can use two boards instead of one. The job of the slave is to read all the sensors readings periodically and only send the reading when the master requests it. We will be using I2C, only because it is simple and quick to implement.
First, the job of the master to request the data from the slave and send it to the IoTHub. That is all it does.

```
...
// MQTT Messages
const uint16_t messageLength = 256;
char message[messageLength] = "";
bool message_pending = false;
...
```

```
if (!message_pending)
 {
  digitalWrite(LED_BUILTIN, HIGH);

  Wire.beginTransmission(8);
  Wire.write("Start");
  Wire.endTransmission();

  delay(100);

  for (uint16_t i = 0; i < messageLength; ++i)
  {
   Wire.requestFrom(8, 1);

   if (Wire.available())
   {
    char c = Wire.read();
    message[i] = c;
   }
   else
   {
    break;
   }
  }

  delay(100);

  Wire.beginTransmission(8);
  Wire.write("Stop!");
  Wire.endTransmission();

  if (strlen(message) <= 0)
  {
   strcpy(message, "");

   return;
  }

  SendMessage(iothub_client_handle, message);

  strcpy(message, "");
 }

 IoTHubClient_LL_DoWork(iothub_client_handle);
 …
```

We can see that the code above sends the slave the string ‘Start’, and then it waits for the slave to send data. Once the master receives data, it will read and store it byte by byte. Once the master has received all the data, a string ‘Stop!’ is sent to indicate to the slave the master has receive all the

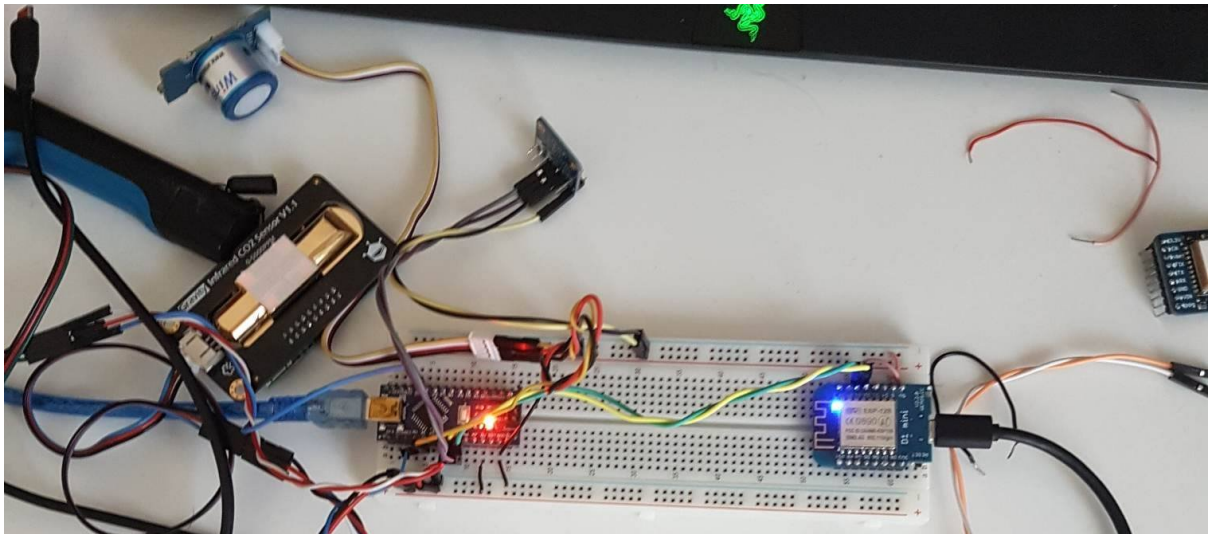
data. It then simply sends to message to IoTHub. The reason by we send ‘Stop!’ instead of ‘Stop’ is that ‘Stop!’ the same length as ‘Start’ in terms of bytes. This makes it easier to read data.

Here we have an excerpt from the salve’s code:

```
...
// Message
const uint16_t messageLength = 256;
char message[messageLength] = "";
uint8_t count = 0;

// Command
const uint8_t commandLength = 6;
char command[commandLength] = "";
...
bool send = false;

void PadData()
{
 for (uint8_t i = strlen(message); i < messageLength - 1; i++)
 {
  message[i] = '0';
 }
}

void RequestEvent()
{
 if (send && count < messageLength)
 {
  Wire.write(message[count]);
  count++;
 }
 else
 {
  count = 0;
 }
}

void ReceiveEvent(int size)
{
 while (Wire.available() > 0 && commandLength < count)

 {
  command[count] = Wire.read();
  count++;
 }

 if (!strcasecmp(command, "Start"))
 {
  send = true;
```

```
  }
  else if (!strcasecmp(command, "Stop!"))
  {
    send = false;
  }
...
}

void setup()
{
...
  Wire.begin(8);
  Wire.onRequest(RequestEvent);
  Wire.onReceive(ReceiveEvent);
...
}

void loop()
{
...
  if (!send)
  {
    strcpy(message, "");
    CreateMessage(message);
    digitalWrite(LED_BUILTIN, LOW);
    delay(5000);
  }
}
```

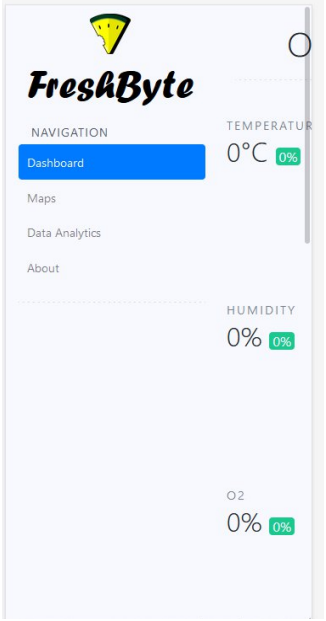
To put simply, the slave will always read the sensors periodically. It will not read the sensors when it is sending the data to the master. We start by registering the RequestEvent and ReceiveEvent. The ReceiveEvent fires when the master sends something to the slave. It will read up to 6 bytes on data. The last byte includes the null terminator '\0'. When the command is equal to 'Start', then set send to true. If it is 'Stop!', then set send to false. The master also request data after sending 'Start'. When the send is equal to true, then don't read the sensor data. The slave will format the readings into a JSON format to be sent to the master. After each reading and formatting, the message is padded with zeroes. This is to ensure that there are no invalid or remnants of random data from the RAM.



Above, is an image of the two boards in action. We can consider each board as a separate power supply. In a way, this is a very crude prototype to address the power issue.
I have ordered an ESP 32 Mini Kit which contains all the required pins and Wi-Fi.
I need update the team and quickly update Sam on what I have done.

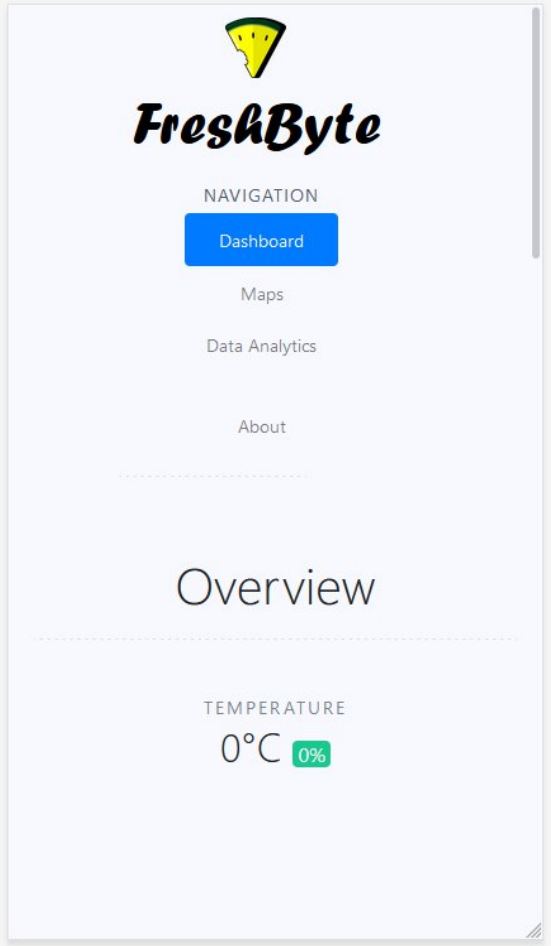
09-08-2019
An Unscheduled Meeting with Sam
We were lucky and met Sam in his office when he was free. We told him that we are using two boards for the Open Day demonstration only. Once Open Day is over and the ESP32 arrives, we will move all the sensors and migrate the code over. Sam has approved this.
Sam has suggested to use multiplexers instead. We can multiplex the senors instead of using two boards. However, one problem is that the Wemos D1 mini is very limited on pins. Thus, using an ESP32 makes more sense.
We can use a Wemos battery shield as external powers for sensors alone.

10-08-2019
Mobile Friendly
Today I will work on the dashboard to make it mobile friendly. Currently it has no implementations to handle mobile views. This can be seen below.



A simple way to make this dashboard mobile friendly is to move the navigation panel to the top and stack and align everything else vertically. This simple excerpt of the CSS code can implement this functionality:

```
/* Mobile */
@media screen and (max-width: 600px) {
  .wrapper {
    display: block;
    text-align: center;
    align-content: center;
  }
  .mobile {
    padding-left: 50%;
  }
```



The result is now the dashboard can be viewed on mobile comfortably.
Now I will need to rework the how the back end (sever) communicates to the front end (client).

11-08-2019
Open Day
Open Day went very smooth, there were no major problems. However, a recurring issue is that the master and slave kept de-syncing from each other and end up locking up. It is due to time drift and the internet lagging. The system we have now is very time sensitive. It relies on send and receiving data successfully and periodically. This can be fixed but using an ESP32 is a better solution than continuing to use what we have now. It reduces complexity.

18-08-2019
Reworking Server to Client Communication
Today, I will rework the back end to front end communication. I realised that I need to hide sensitive information and credentials in Azure. This will be the foundations of the Data Analytics page.

To begin with, the client will broadcast to the server. In this case, there is only one client and server, so we can safely use broadcasting to send data between the client and server.

The server broadcasts a JSON to the client that contains two fields, data and tag. The data is used to store the payload, and the tag is used to request a certain type of data.

```
// Key map key.
webSocket.send(JSON.stringify({ data: "map_key", tag: "map_key" }));
```

To send data to server, we will can simply broadcast it. In this case we are requesting data that contains SQL entries:

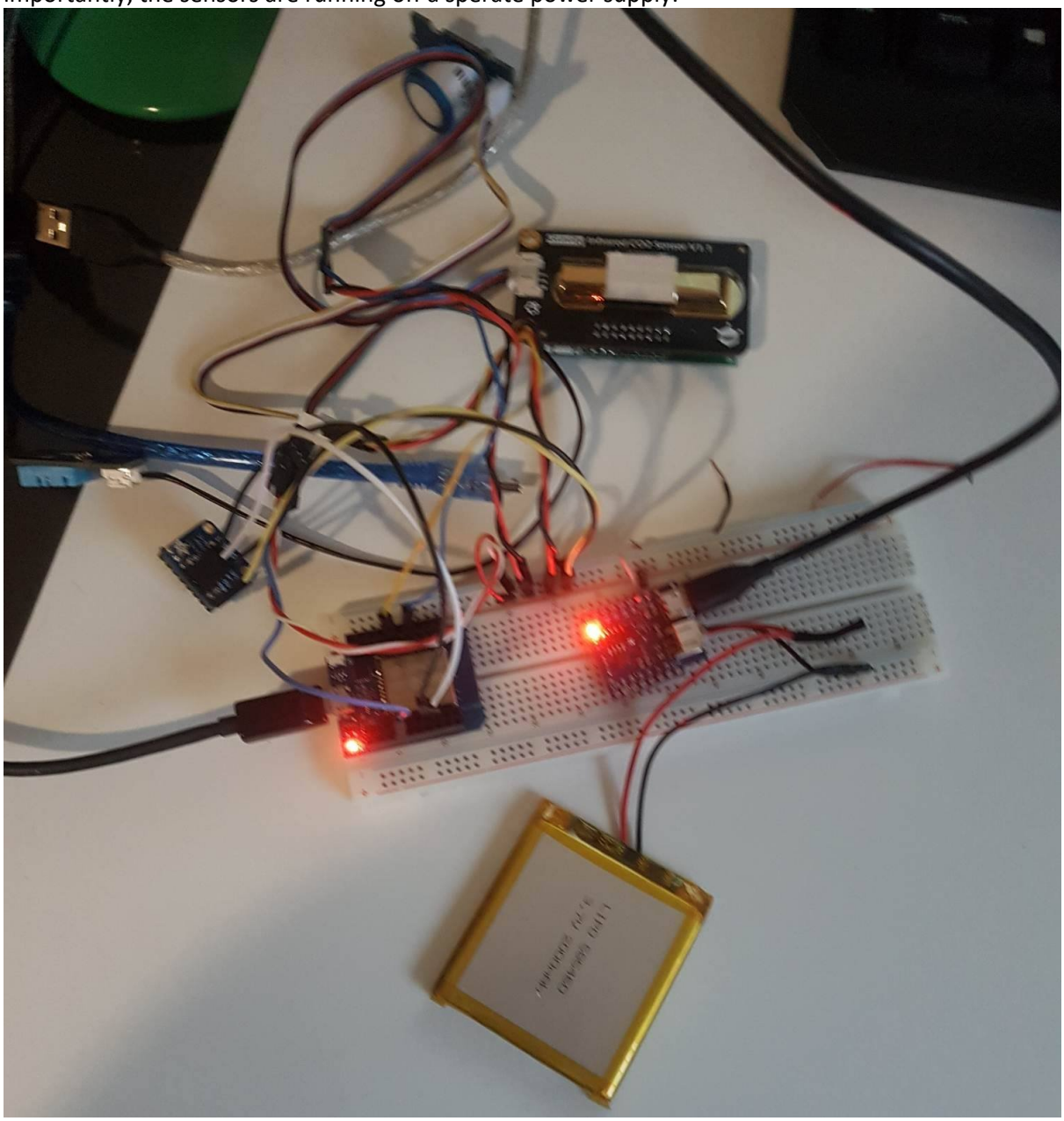
```
// Call server to send SQL data first 100 entries.
webSocket.send(JSON.stringify({ data: "select * from Telemetry order by Time offset 0 row fetch first
 100 row only", tag: "sql" }));
```

We can now implement the logic to request data based on the Tag. To do this we need to create an inline/delegated hook to run when a message is broadcast. In this case this is from the client. If we look at the excerpt of the code below, we can see that we begin by check the Tag. The tag determines the type for data we are sending to the client.
We can also see that the credentials are hidden and cannot be found in the server code. Simply put, when the server receives a message from the client, it will check the Tag. If we continue from the code above, in this case we have the Tag value of 'sql'. Now we will need to connect to the SQL database that contains the measurements and send them as a JSON formatted string to the server.

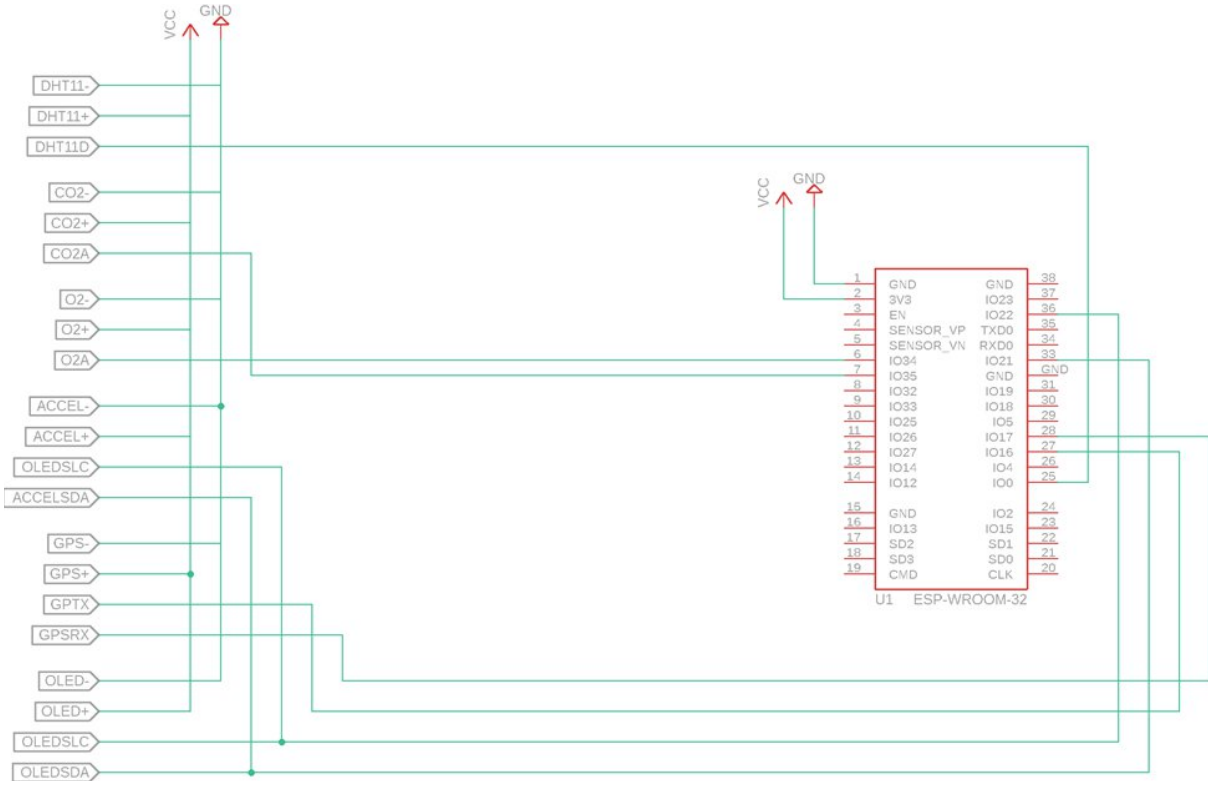
```
...
// Wait for client to request SQL data.
wss.on('connection', function connection(ws) {
   ws.on('message', function incoming(message) {
      let obj = JSON.parse(message);

      if (obj.tag == "sql") {
         // https://docs.microsoft.com/en-us/azure/sql-database/sql-database-connect-query-nodejs
         // Create connection to database
         var config = {
            authentication: {
               options: {
                  userName: process.env['Azure.SQL.Database.UserName'],
                  password: process.env['Azure.SQL.Database.Password']
               },
               type: 'default'
            },
            server: process.env['Azure.SQL.Database.ServerName'],
            options: {
               database: process.env['Azure.SQL.Database.DataBase'],
               encrypt: true
            }
         }
         var connection = new Connection(config);

         // Attempt to connect and execute queries if connection goes through
         connection.on('connect', function(err) {
            if (err) {
               console.log(err)
            } else {
               queryDatabase()
            }
         });
```

```
        function queryDatabase() {
            console.log('Reading rows from the Table...');

            // Read all rows from table
            var request = new Request(
                obj.data,
                function(err, rowCount, rows) {
                    console.log(rowCount + ' row(s) returned');
                    process.exit();
                }
            );

            request.on('row', function(columns) {
                let row = {};
                columns.forEach(function(column) {
                    row[column.metadata.colName] = column.value;
                });
                wss.broadcast(JSON.stringify(Object.assign(row, { Tag: "data_analytics" })));
            });

            connection.execSql(request);
        }
    } else if (obj.data == "map_key") {
        wss.broadcast(JSON.stringify({ data: process.env['Azure.Maps.SubscriptionKey'], Tag: 'map_ke
y' }));
    }
  });
});

var iotHubReader = new iotHubClient(process.env['Azure.IoT.IoTHub.ConnectionString'], process.env
['Azure.IoT.IoTHub.ConsumerGroup']);
iotHubReader.startReadMessage(function(obj, date) {
  try {
    console.log(date);
    date = date || Date.now()
    wss.broadcast(JSON.stringify(Object.assign(obj, { Time: moment.utc(date).format('YYYY:MM:DD
[T]HH:mm:ss'), Tag: "dashboard" })));
  } catch (err) {
    console.log(obj);
    console.error(err);
  }
});
...
```

20-08-2019
Moving Sensors to ESP32 and Measuring Power Consumption
Today, I will move all the sensors over to the ESP32 and refactor the code to work with it. Most importantly, the sensors are running off a sperate power supply:

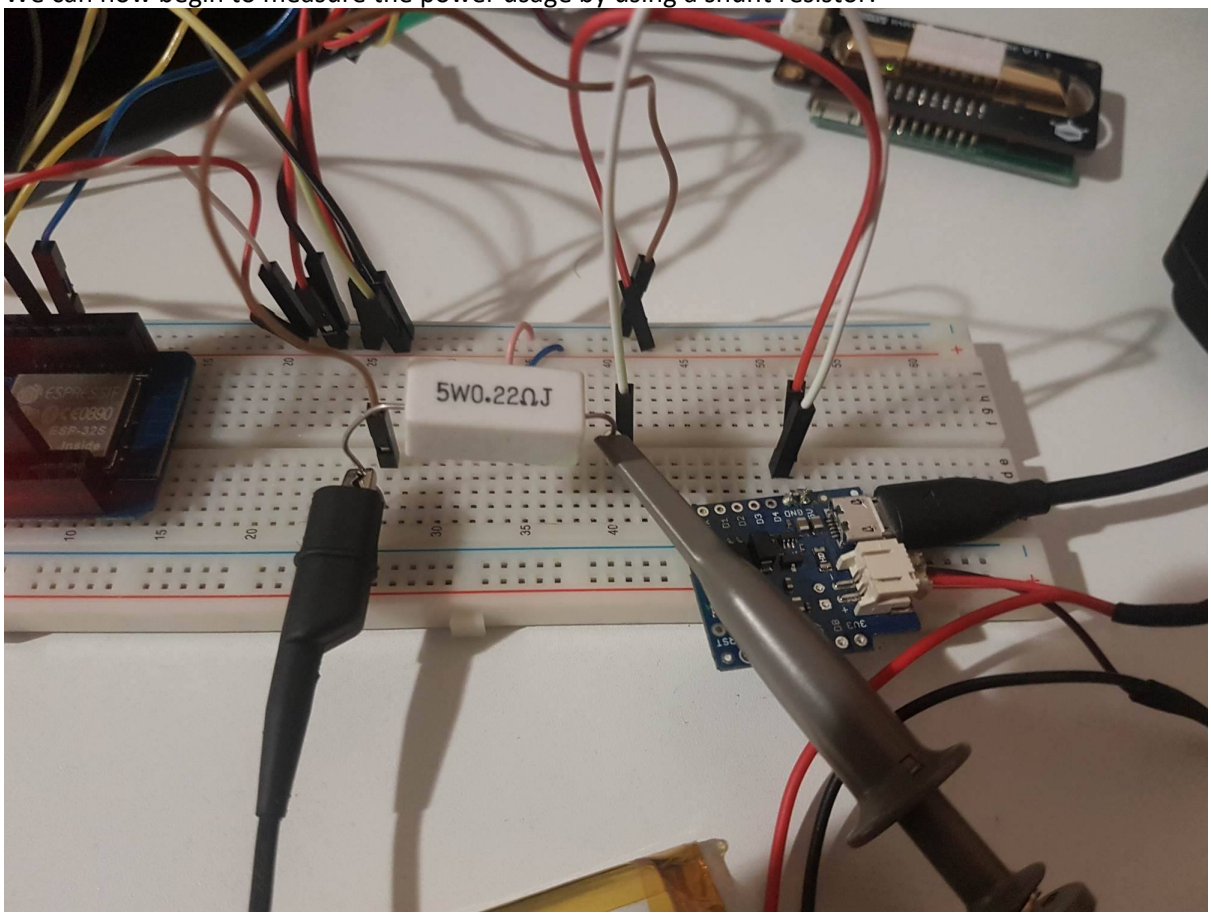


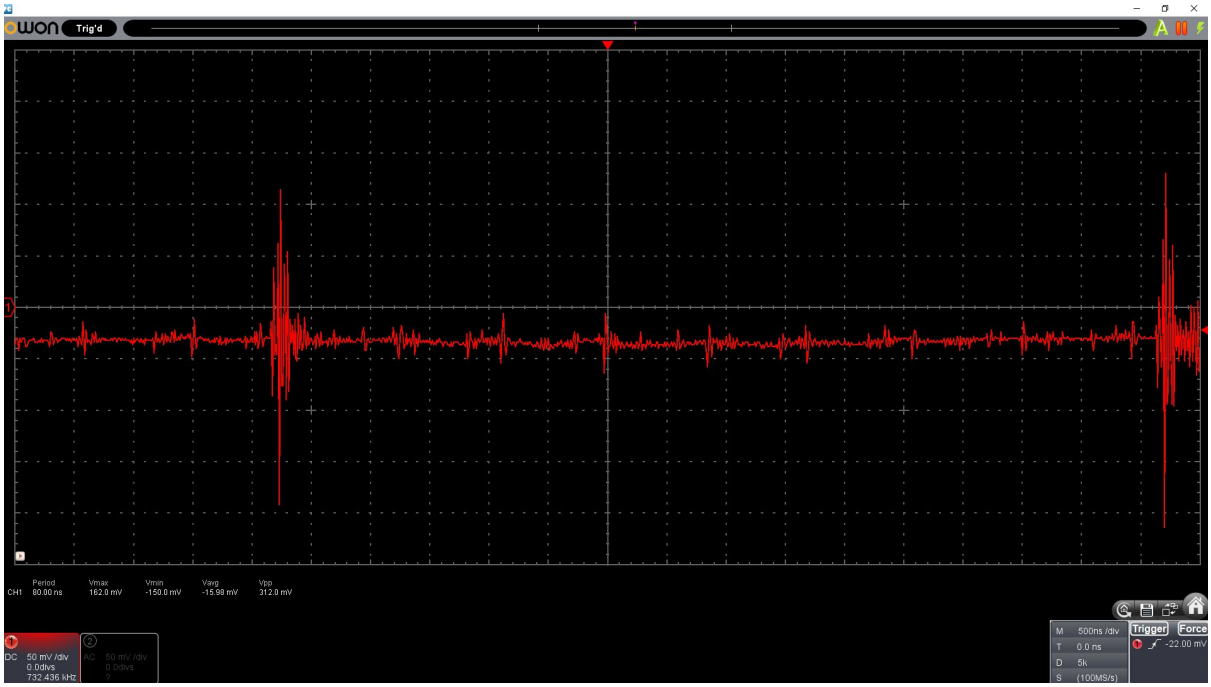
With the sensors working and functioning properly, I can represent the image above as a crude schematic below…



What is important here is that all the sensors are able connected on a single board, and that the board and sensors have their own power supply.

We can now begin to measure the power usage by using a shunt resistor:





The maximum voltage measured across the shunt resistor is 162.0mV. Thus, the maximum current is $\frac{162.0mV}{0.22\Omega} = 736.36mA$ and the maximum power is $\frac{(162.0mV)^2}{0.22\Omega} = 119mW$. The current consumption is very high, but this is expected. There are two gas sensors that rely on heating elements, which require a high current in order to function properly.

If a battery with a capacity of 2000mAh is used, the expected battery life would be $\frac{2000mAh}{736.36mA} = 2.72hrs$. Since the purpose of the FreshByte module is to be shipped long distances over an extended period, this battery life is not enough. It is also important to note that these calculations are assuming that the device is continuously sending and reading measurements.

Realistically, the device will be reading and sending measurements every 30 minutes. If we assume that the device in deep sleep, it will draw a current of 0.15mA [https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/]. The battery life can be extended if the device reads and sends measurements every 30 minutes and then goes to a deep sleep state. If we also assume that the time to send and read measurements takes 5 minutes (gas sensors need to heat up to operating temperatures), then we can calculate the power consumption of the device for each day as follows:

$Frequency\ of\ sending\ and\ reading\ measurements$:

$$\frac{24\ hours\ \times\ 60\ mins}{30\ mins} = 48\ times\ per\ day.$$

$Total\ time\ spent\ sending\ and\ reading$:

$$48\ times\ per\ day\ \times\ 5\ mins\ =\ 240\ minutes.$$

$Total\ time\ spent\ sleeping$:

$$(24\ hours\ \times 60\ mins) - (240\ mins)\ =\ 1200\ mins.$$

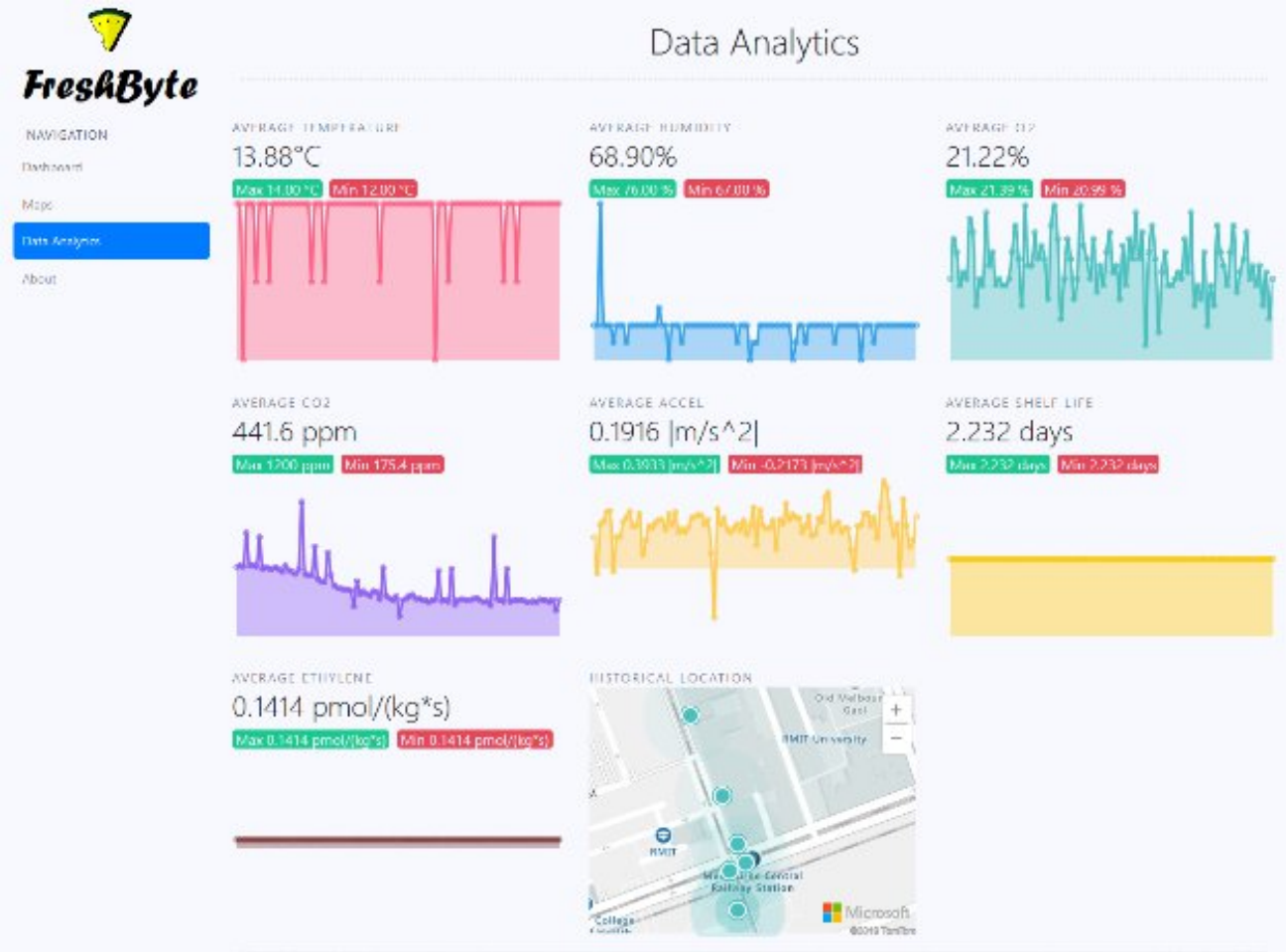
$Total\ power\ consumed\ per\ day$:

$$\left(\frac{240\ mins}{60\ mins} \times 736.36mA\right) + \left(\frac{1200\ mins}{60\ mins}\ \times\ 0.15mA\right) =\ 2945.45\ mAh + 3\ mAh$$
$$=\ 2948.45mAh.$$

Therefore, to read and send messages every 30 minutes with deep sleeping in between, a battery with a minimum of 2948.45mAh is required just for one day of operation. This is not ideal since the

device is required to be compact and portable. The deep sleep mode added in can save a large amount of power and is suitable for short to medium trips. However, longer trips will draw a lot more power.

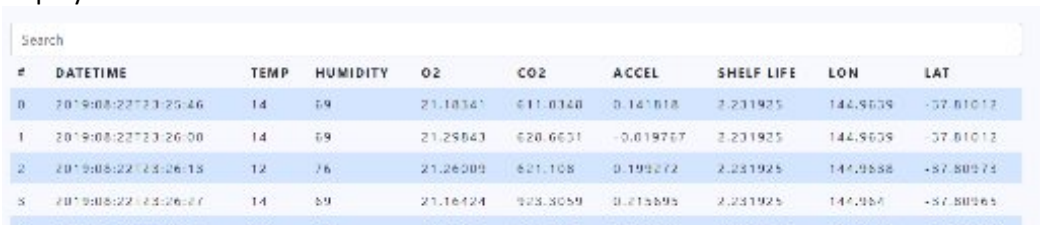
22-08-2019
Data Analytics
The Data Analytics page shows all of the readings. The average of the readings are displayed under the titles of each reading. The green box contains the maximum reading, whilst the read box contains the lowest reading. The map also contains all of the co-ordinates.



23/08/2019
Update Sam on the Dashboard
It was a quick meeting; we updated Sam on the status of dashboard. The Data Analytics is implemented. The next step to work in it more and implement a responsive search.

20/09/2019
Capstone Project Completed
Today, we discuss with Sam that everything is complete and prepared for EnGenius. We just need to transfer the hardware into a box. Sam has told us that this project is complete.
We must now focus on creating the posters, banners, and the final Capstone report.

08-09-2019
Search Function
Today, I will implement a search function. We will have a search textbox and a table below. The table will display the 100 most recent reading from the SQL database. In this case, due to limited time, I will implement a lazy search. It will display any entries that contain the text in the search textbox. For example, it we type in 14, entries that contain 14 in any of their columns will be displayed.



Now we will discuss briefly about the code implementation.
To find the maximum and minimum of each reading, I use the following code:

```
...
// Update max and min values
function UpdateMaxMin(data, id_max, id_min, units) {
  if (data.length > 1) {
    // https://developer.mozilla.org/en-
US/docs/Web/JavaScript/Reference/Global_Objects/Number/toPrecision
    document.getElementById(id_max).textContent = "Max " + Math.max.apply(Math, data).toPreci
sion(4) + " " + units;
    document.getElementById(id_min).textContent = "Min " + Math.min.apply(Math, data).toPrecis
ion(4) + " " + units;
  }
}
...
UpdateMaxMin(temp_data, "temp-max", "temp-min", "°C");
...
```

An array that contains the measurement of a sensor is passed to the function. The references to the HTML elements that are used to display the max and min values are also passed into the function. We then simply apply a max and min function the array to find the values.

To find the average, we apply find the sum of all the values and divided them by the number of elements in the array:

```
...
// Find average value of array.
// https://www.jstips.co/en/javascript/array-average-and-median/
function Average(data) {
  // https://developer.mozilla.org/en-
US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce
  var sum = data.reduce((previous, current) => current += previous);
  return (sum / data.length).toPrecision(4);
}
...
document.getElementById("temp").textContent = Average(temp_data) + "°C";
...
```

To search for a value, we first need to implement a basic logic to loop through all items in the table and hide the rows that don't match what we are looking for:

```
...
// Search
// https://www.w3schools.com/howto/howto_js_filter_table.asp
function Search() {
  var input, filter, table, tr, td, i, txtValue;
  input = document.getElementById("search");
  filter = input.value.toUpperCase();
  table = document.getElementById("table");
  tr = table.getElementsByTagName("tr");
  // Loop through all table rows, and hide those who don't match the search query
  for (i = 1; i < tr.length; i++) {
    // Search the whole row (row elements).
    for (j = 0; j < tr[i].getElementsByTagName("td").length; j++) {
      td = tr[i].getElementsByTagName("td")[j];
      if (td) {
        txtValue = td.textContent || td.innerText;
        if (txtValue.toUpperCase().indexOf(filter) > -1) {
          tr[i].style.display = "";
          break;
        } else {
          tr[i].style.display = "none";
        }
      }
    }
  }
}
...
```

This is a linear search. In this case, since we are dealing with only 100 items, this is adequate. In the future, a more efficient search algorithm should be used.

Tomorrow, I will implement the code to make the search responsive. That is, when we search for values in the table, the values in the table will also be shown in charts.

09-09-2019
Responsive to Search
To make the charts responsive and update when we search for something, we need to know what is in the table after the search.

```
...
// Get table ids.
function GetIDs() {
  var table, tr, td, i, ids = [];
  table = document.getElementById("table");
  tr = table.getElementsByTagName("tr");
  // Loop through all table rows, and find the items that are not hidden.
  for (i = 1; i < tr.length; i++) {
    if (tr[i].style.display == "") {
      var td = tr[i].getElementsByTagName("td").item(0);
```

```
      ids.push(td.textContent || td.innerText);
    }
  }

  return ids;
}
...
```

Here we can see that we are looking for rows that are not hidden and adding their ID values to a list. This list is used as a list of indexes to know which items are shown.

Next, we will implement a function to update the charts based on the ID values from before:

```
...
// Filter Charts.
function FilterCharts() {
    var time_data_f = [],
        temp_data_f = [],
        humidity_data_f = [],
        o2_data_f = [],
        co2_data_f = [],
        accel_data_f = [],
        shelf_life_data_f = [],
        ethylene_data_f = [],
        lon_lat_data_f = [];

    GetIDs().forEach(x => {
        time_data_f.push(time_data[x]);
        temp_data_f.push(temp_data[x]);
        humidity_data_f.push(humidity_data[x]);
        o2_data_f.push(o2_data[x]);
        co2_data_f.push(co2_data[x]);
        accel_data_f.push(accel_data[x]);
        shelf_life_data_f.push(shelf_life_data[x]);
        ethylene_data_f.push(ethylene_data[x]);
        lon_lat_data_f.push([lon_data[x], lat_data[x]]);
    });

    temp_chart.destroy();
    humidity_chart.destroy();
    o2_chart.destroy();
    co2_chart.destroy();
    accel_chart.destroy();
    shelf_life_chart.destroy();
    ethylene_chart.destroy();

    if (ready) {
        map.markers.clear();
        lon_lat_data_f.forEach(x => {
            var user_position_marker = new atlas.HtmlMarker({
                htmlContent: '<div class="pulseIcon"></div>',
```

```
                position: [x[0], x[1]]
            });
            map.markers.add(user_position_marker);
            console.log('Added marker: ' +
                x[0] + ',' + x[1]);
        });
    }
...
```

The important thing to look at here is that we are using GetIDs() to get the list if indexes in the table that are shown. The IDs are used as indexes for each array that contains the readings from the sensors. Effectively, this is a crude and inefficient filter. In hindsight, this implementation is slow and requires a lot of CPU time, which is not ideal in a browser scenario. I should've not stored the filtered data in their own arrays, instead I should've worked off the arrays defined above. This way I can reduce the memory footprint and improve performance. This is a mistake.

However, despite the poor implementation, the dashboard is still responsive, and functions will minimal lag. This is because we are only work with 100 entries.

14-09-2019
Final Dashboard Complete
The dashboard (https://fresh-byte.azurewebsites.net/index.html) is now complete. All the basic functionalities are implemented.
This concludes my journal.