IoT – Smart Racking

3CB105: Ben Coxford

# Contents

[Contents 2](#_Toc57968936)

[IoT Smart Racking 3](#_Toc57968937)

[Problem Definition 3](#_Toc57968938)

[Proposed Solution 3](#_Toc57968939)

[Cloud-storage or Streaming? 3](#_Toc57968940)

[System Requirements 4](#_Toc57968941)

[Hardware and Software Costs 4](#_Toc57968942)

[Legal and Ethical Evaluation 4](#_Toc57968943)

[Security Evaluation 4](#_Toc57968944)

[Spoofing and Information Disclosure 4](#_Toc57968945)

[Tampering 5](#_Toc57968946)

[Denial of Service (DoS) 5](#_Toc57968947)

[Minimising the Risk 5](#_Toc57968948)

[Hardware Setup 5](#_Toc57968949)

[Device Software 6](#_Toc57968950)

[Application Software 7](#_Toc57968951)

[Data Analytics – Live Feed 7](#_Toc57968952)

[Data Analytics – Time Series Graphs 7](#_Toc57968953)

[Conclusion 8](#_Toc57968954)

[Appendices 9](#_Toc57968955)

[Appendix A 9](#_Toc57968956)

[Figure 1 – IoT Stack 9](#_Toc57968957)

[Figure 2 – Website Screenshots 9](#_Toc57968958)

[Figure 3 - Live Data Feed 10](#_Toc57968959)

[Figure 4 – Time Series Graphs 10](#_Toc57968960)

[Appendix B 11](#_Toc57968961)

[Diagram 1 – Circuit Diagram 11](#_Toc57968962)

[Diagram 2 – Dual Load Cell and HX711 Sensor Diagram 11](#_Toc57968963)

[Appendix C 12](#_Toc57968964)

[Table 1 – Component Costs 12](#_Toc57968965)

[Table 2 – Testing 12](#_Toc57968966)

[References 13](#_Toc57968967)

# IoT Smart Racking

## Problem Definition

Jayaram (2017) explains how the Internet of Things can facilitate ‘Smart Racking’ where sensors are placed to estimate the number of products on the racking and the temperature of the racking. He further explains how this will facilitate automatic reordering and temperature adjustments on the racking to preserve the freshness of the product and prevent damage.

## Proposed Solution

As described by Usman (2020), the model I am using is the IoT stack as shown in appendix A, figure 1. To measure the product’s weight, I plan to use weight sensors that use load cells (or strain gauges). Another sensor will be used to measure the temperature and humidity, along with an OLED display to display the current temperature value.

Local processing will be carried out by the Pi Zero, a small, compact version of the popular microcontroller. The device will act as a client and will send data to a computer on the local network which is hosting a server. The data sent to the server will then be formatted and sent to Microsoft Azure’s IoT Hub. The IoT Hub (Microsoft 2019) is a service hosted in the cloud which allows bidirectional communication from the microcontroller or server to the cloud using the Message Queuing Telemetry Transport (MQTT) protocol.

As the data reaches the cloud-processing tool Azure Functions (Microsoft 2020a), the data is formatted and stored in Microsoft Azure’s CosmosDB (Microsoft 2020b). CosmosDB is a responsive and consistent cloud-storage service. Finally, the data will be viewable from a React.JS application.

## Cloud-storage or Streaming?

There are several reasons to use cloud-storage instead of streaming the values using webhooks. The data is persistent, meaning the data can be accessed from the current minute to years in the past. This can enable machine learning and predictive analysis tools to access this data to predict product demand or even predict refrigeration unit breakdowns. The user is not constantly viewing the data so streaming the data is not essential; however, the update-rate is about 5 minutes. This periodic update allows users to view the data in close to real-time.

## System Requirements

The system must be able to:

* Read the temperature and humidity of a given space and display this on an OLED display and a web-application.
* Calculate the number of products on smart racking using multiple weight sensors. This must be displayed on a web-application.
* Display archived data in the form of a graph.

## Hardware and Software Costs

The total cost of the project is£48.07 (excluding the mount). The project can be expanded to house more ‘smart racking’ spaces by adding additional temperature and humidity sensors as well as LEDs and weight sensors. The multiplexer can include up to 8 displays with the Raspberry Pi being capable of serving 8 different multiplexers. This enables the Raspberry Pi to control up to 64 I2C displays that use the same address. As an improvement, I would replace the DHT22 sensors with the cheaper and more accurate DS18B20 sensor. This would reduce the cost of the project to £42.97. See appendix C, table 1 for the individual price of each component.

Microsoft Azure is an affordable service that provides the backend solution for this project. For unlimited IoT devices and up to 400,000 messages a day, the IoT Hub service costs £7.45 per month with Azure Functions being free up to 1 million messages per month. After 1 million messages, the service costs an additional £0.15 per million executions. CosmosDB would cost £20 per month storing up to 50GB of data with a max of 400 requests per second (read/write requests).

*Estimates Calculated using Azure Price Calculator (Microsoft, 2020c).*

## Legal and Ethical Evaluation

### Security Evaluation

Azure outlines a threat model to understand the risks associated with IoT projects as cybersecurity attacks can threaten processing, communication, and storage. Categorized into five categories, cybersecurity attacks can involve spoofing, tampering, information disclosure, denial of service, and elevation of privilege (Microsoft 2020d).

### Spoofing and Information Disclosure

An attacker can intercept data mid-communication and potentially relay or alter the communication between two parties without them knowing, this is also known as a Man in the Middle (MITM) attack. In addition to this, IoT devices are connected to a network and protected using a shared key. If this key was disclosed, an attacker can control, manipulate, and collect data from a device (Microsoft 2020d). In the project’s scope, an attacker could fluctuate and change the quantity value of products to disrupt service and even trick the company into ordering more or less of a product that they might not require.

### Tampering

An attacker can take a more physical approach by tampering with a device, such as draining a battery or replacing/changing the software to operate under genuine security keys and authentication (Microsoft 2020d). This is generally forgotten about and overlooked when designing an IoT system; however, it plays a significant role in protecting the system.

### Denial of Service (DoS)

Denial of Service (DoS) attacks are more associated with attacks on websites; however, DoS attacks involve any form of interference to render a service inoperable. Such examples can include cutting wires or interfering with radio frequencies to disable a network (Microsoft 2020d).

### Minimising the Risk

To minimize the risk of spoofing the IoT Hub uses the standard X.509 verification to verify the device belongs to the service, in addition to using 256-bit AES encryption to encrypt and decrypt the data (Microsoft 2020e). This will reduce the risk of MITM attacks, although it does not reduce the risk of tampering. In addition to this, remote access has also been disabled, to minimize the risk of unauthorised access.

Physical protection is just as important as cyber protection. IoT devices should have tamper-proof cases, a function to disable the device on short-circuits, and passwords to protect against unauthorised access. Although, I do not intend to physically damage my Raspberry Pi, for deployment I would disable remote access and provide a tamperproof case. The physical design of the network can also reduce the risk of DoS attacks.

## Hardware Setup

Wiring the project was straightforward following the circuit diagram as shown in appendix B, diagram 1. The components were soldered and wired to two breadboards with all connections to the Raspberry Pi Zero passing a T-Cobbler to reduce the chance of disconnecting a wire.

Firstly, I wired the DHT22 sensors to the Pi and the OLED’s to the I2C multiplexer. The I2C multiplexer acts as the gatekeeper between the two OLED displays and the reason for this is because both OLED displays share the same address. The multiplexer will give each display a unique address, allowing each display to be accessed individually.

Next, I wired the load cells to the HX711 sensors using a schematic designed by Indrek (no date). See appendix B, diagram 2 for the schematic using two load cells. The HX711 sensor amplifies the current for the Pi to be able to read the values. Initially, I found the sensors to be temperamental and I had to resolder each one. After the changes, I had no trouble reading the values from the sensor.

Finally, I attached the LED’s and resistors to the breadboard and connected the Raspberry Pi to the T-Cobbler. The mount was built using pre-cut MDF boards which were glued together. After painting it black, I used a hot-glue gun to attach the circuitry to the mount. I found this part of the project to take a lot of time refitting and rewiring the circuit, to ensure the wiring was compact and neat enough to distinguish between the wires and components.

## Device Software

Following the documentation and tutorials by Microsoft, I had completed setting up the backend solution to the system. This involved initializing cloud storage (Microsoft 2020f), the IoT Hub (Microsoft 2020g), and Azure Functions (Microsoft 2020h).

Continuing the tutorials, the next step was to program the telemetry to the IoT Hub through Python, sending the data using the MQTT protocol using Azure IoT Hubs Python framework (Microsoft 2020g).

Secondly, I captured this data using Azure Functions through JavaScript, listening to incoming telemetry from the IoT events hub (Microsoft 2020h). After the function was receiving the data, I adapted the function to format and send the data to CosmosDB, Azure’s cloud-storage solution (Microsoft 2020f).

Throughout the entire project, I used two IDE’s - Pythons IDLE and Visual Studio Code. The server and client end were created using Python whilst the frontend website was built using ReactJS.

To begin, I set up multiple classes to encompass each sensor individually, along with the main program. Next, I programmed the DHT22 sensors using a framework from Adafruit (2020a). Overall, the sensors matched with good accuracy when measuring room temperature. To accompany the DHT22 sensor was the 2 OLED displays that interface with the Raspberry Pi using the Inter-Integrated Circuit (I2C) protocol. The issue I found at first was that both OLED displays shared the same address; therefore, if I addressed one display, I would also be addressing the other display. Now if I wanted to display two different temperature readings, separately of each display, I would have to provide each display with a unique address, and to achieve this, I used an I2C multiplexer. The I2C multiplexer acts as an eight-channel port, like a switch in a network, where each port has a unique MAC address; however, with the multiplexer, each port has a unique I2C slave address. Finally, both DHT22 sensors are programmed to display the current temperature reading to the OLED displays.

At this point, I had already programmed part of the code to take readings from the HX711 sensors using a framework by Zak (2018). To begin I calibrated the sensors using an object of a known weight and placing it on the scales. Using the current readings from the sensor and I calculated the scale ratio using the equation below. Essentially, this provides the correct setting to convert the readings into grams.

Scale Ratio = Current Reading / Known Weight

## Application Software

Instead of using ThingSpeak or another pre-built tool, I opted to develop an application using React.js by Facebook (2020) and Chart.js (2020) to show the live data in the form of a table and the archived data in the form of a graph. The benefit of developing the application is having the ability to tailor the design, tables, and graphs to fit the needs of the solution. Furthermore, the project itself is scalable to host many devices which can be achieved by replicating the device and adjusting the product weights and device ID. However, the application can be improved by creating a database of products (including fields such as weight, isFrozen, and isChilled), so that the user can add new devices and change the product ID for each device. This would allow the system to automatically adjust the devices’ settings to measure the correct quantity of a product and to identify the correct temperature zones for preservation and freshness. Screenshots of the application are shown in appendix A, figure 2.

### Data Analytics – Live Feed

The web application is set to update periodically, every 5 minutes. All devices are synced to update on the 5th minute and the data is sent through the pipeline to the database. From here, the data is retrieved using an SQL statement that retrieves the data from the past 5 minutes, displaying the most recent values. An example is shown in appendix A, figure 3.

### Data Analytics – Time Series Graphs

To show how the data measured and stored, I have developed four graphs showing how the data changes over time. This provides the solution potential to be improved using machine learning and predictive analysis. An example is shown in appendix A, figure 4.

## Conclusion

To summarise the project, I believe I have met the requirements of the solution and contributed to solving the chosen problem.

The project posed many challenges, with the most prominent challenge being the HX711 sensors. Python only has two open-sourced frameworks for these sensors. Attempting to use both I struggled to achieve the accuracy required of the solution. Eventually, I adjusted the ratio and calibrated all the sensors correctly and I began to achieve accurate values plus or minus an acceptable margin of error.

Another challenge I encountered was using Azure. I have read through many tutorials on how to tackle this problem using Azure’s tools, and I began using Azure’s IoT Central and not the IoT Hub. I found this tool difficult to set up and found that it was not suited for the project’s solution. As a result, I found a solution using IoT Hub and Azure Functions to streamline the process of collecting, sending, and storing the data in cloud storage.

I believe this effective solution has the potential to improve stock management levels as well as enabling better performance monitoring of refrigeration units; therefore, reducing food wastages and maintaining or even increasing profits.

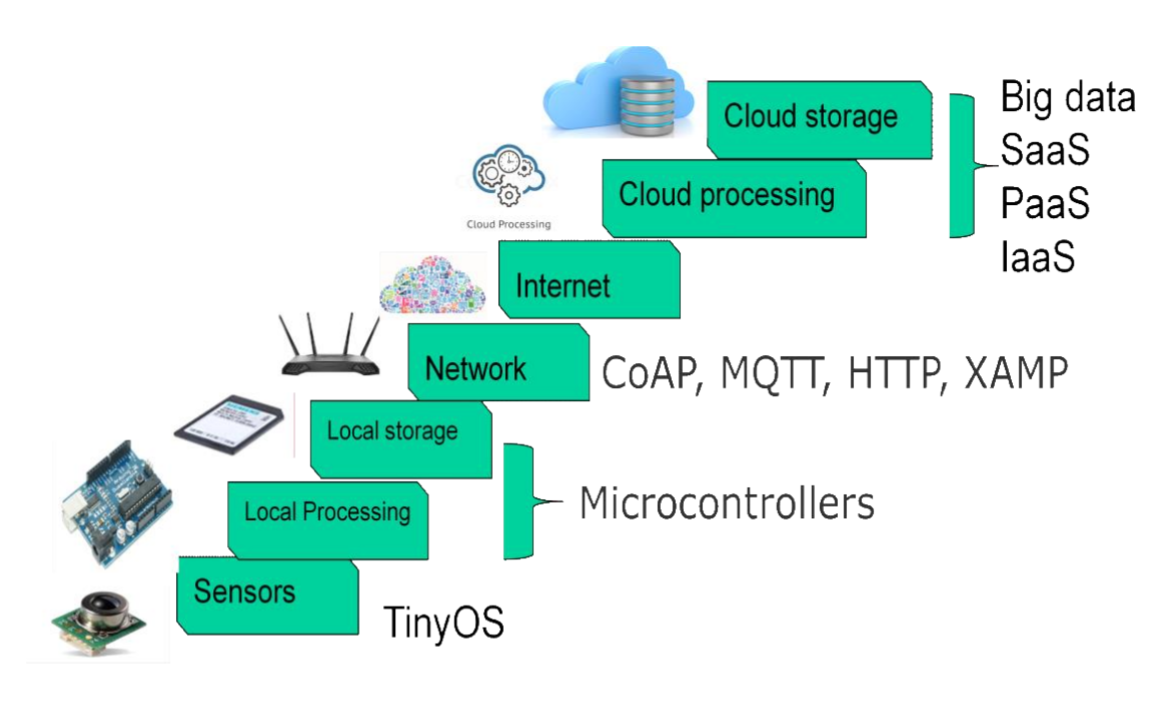
The source code and prototype can be obtained through GitHub via the link: <https://github.com/BCoxford/IoT-SmartRacking>

The video demonstration is published and can be watched through the link: <https://youtu.be/tCmcQORV5FA>

# Appendices

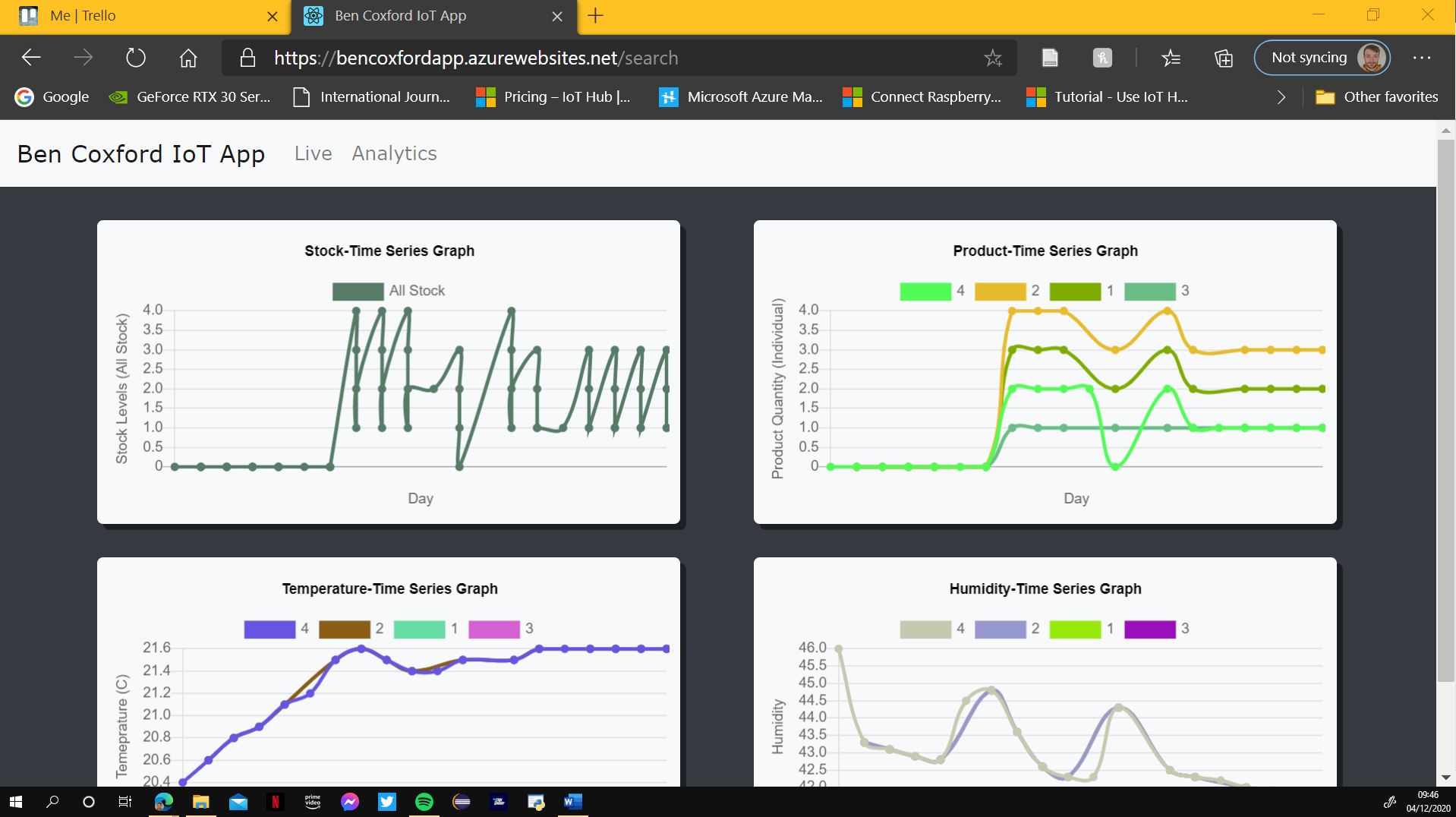
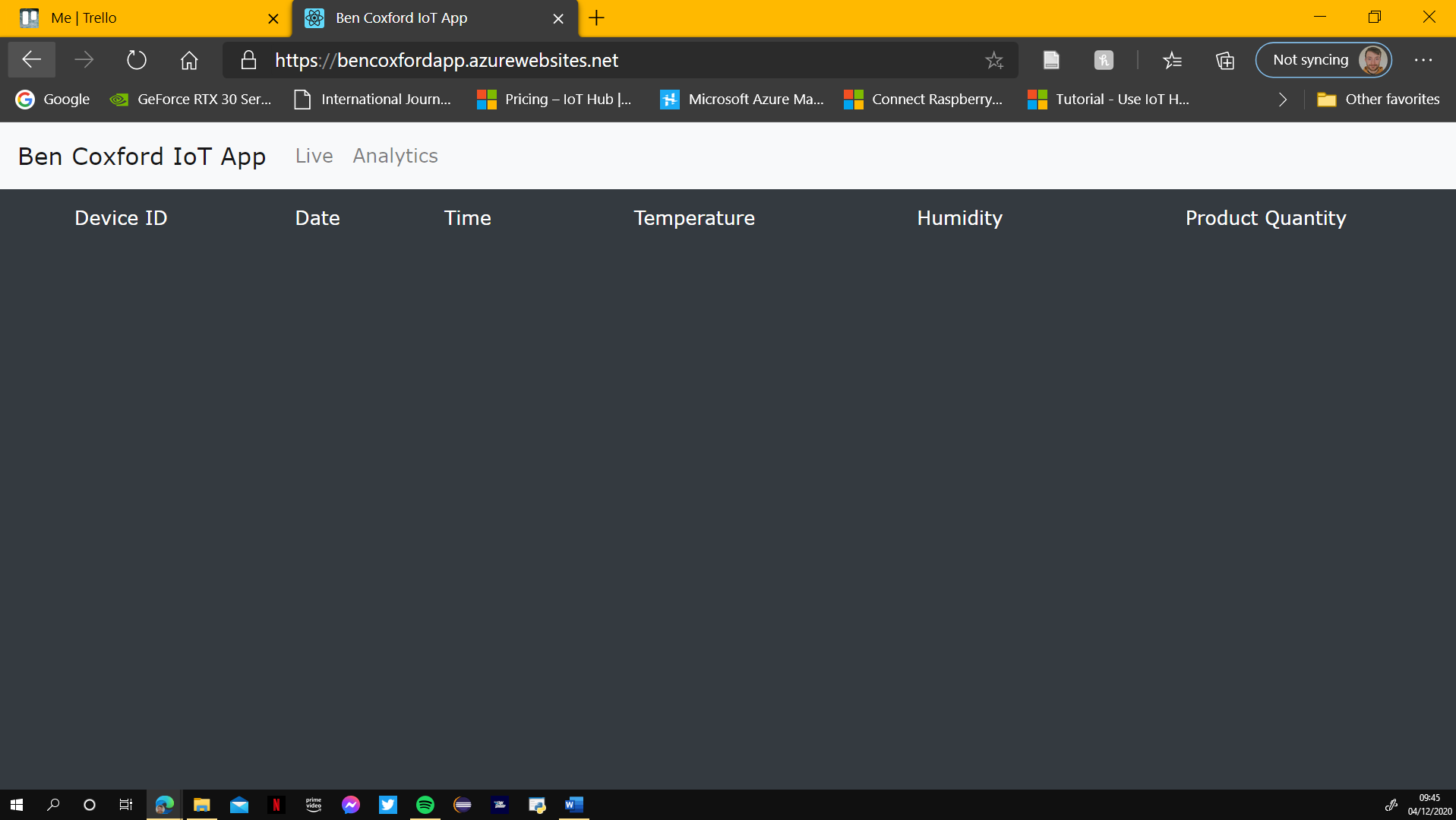
## Appendix A

### Figure 1 – IoT Stack

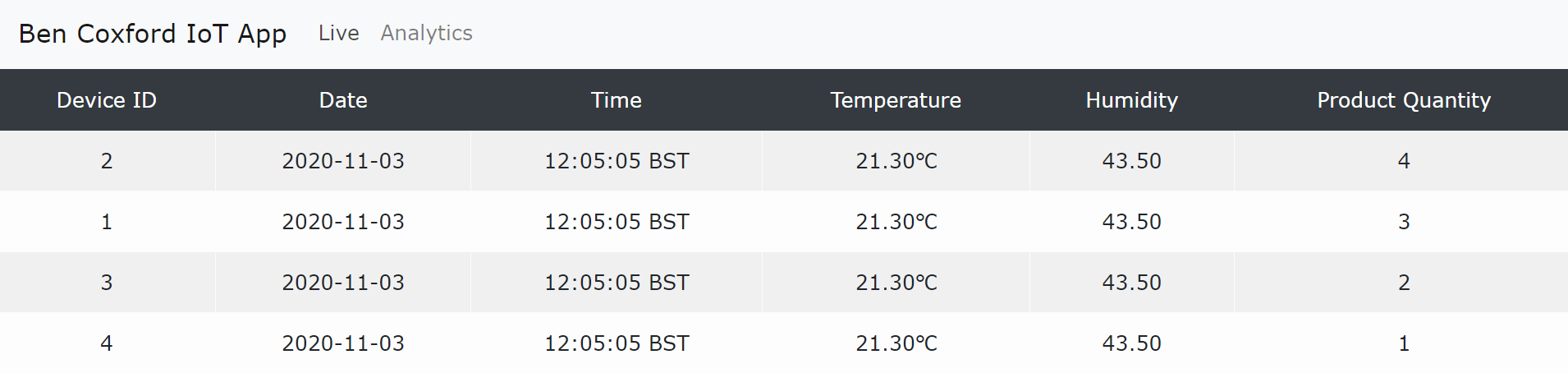


The IoT Stack, presented by Usman (2020)

### Figure 2 – Website Screenshots

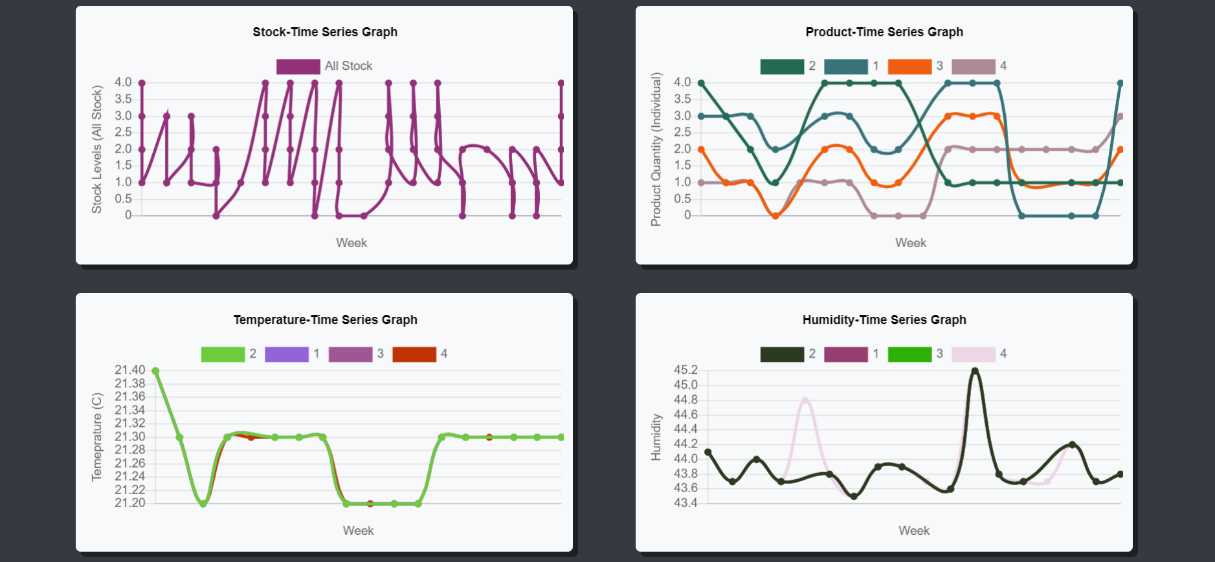


### Figure 3 - Live Data Feed



Live feed of each device’s data

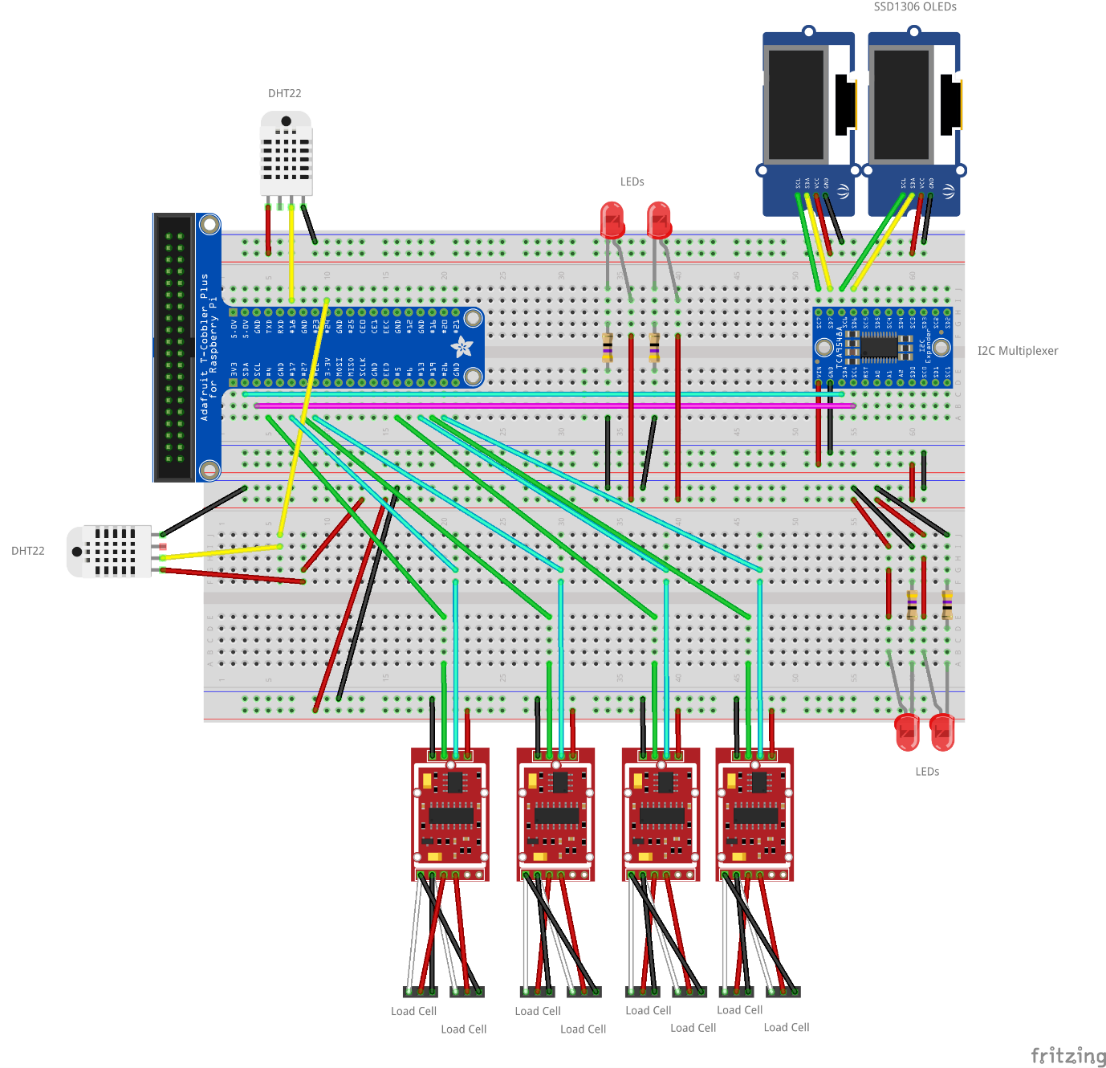
### Figure 4 – Time Series Graphs



Time-Series graphs showing archived data

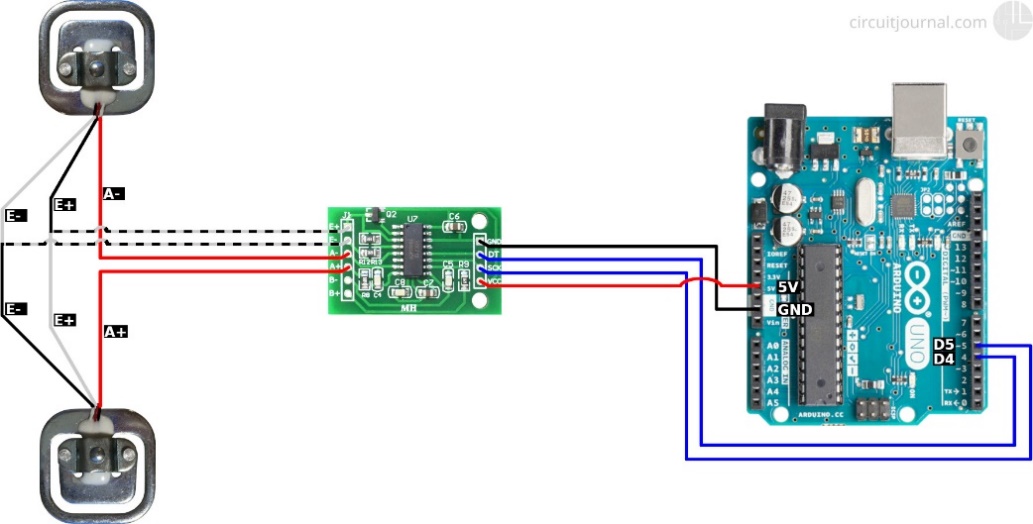
## Appendix B

### Diagram 1 – Circuit Diagram



Created using Fritzing (2019)

### Diagram 2 – Dual Load Cell and HX711 Sensor Diagram



Presented by Indrek (no date)

## Appendix C

### Table 1 – Component Costs

|  |  |
| --- | --- |
| Item | Cost |
| Raspberry Pi Zero W | £7.25 |
| x2 DHT22 temperature and humidity sensors | £11.98 |
| Breadboard, wires, solder, and soldering iron | N/A |
| x2 0.91” OLED Display | £5.90 |
| x4 HX711 Weight Sensors: | £5.60 |
| x8 Strain Gauges | £13.60 |
| x4 1W White LED’s | £0.24 |
| x2 100Ω Resistor | £0.10 |
| x1 I2C TCA9548 Multiplexer | £2.40 |

### Table 2 – Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Description | Test Number | Input Data | Expected Result | Actual Result | Pass? |
| Reading room temperature. | 1 | N/A | Reading between 20°C and 25°C. | 21.2 | Pass |
| Display temperature to the OLED. | 1 | Console Reading: 21.3 | OLED Display reading: 21.3 | 21.3 | Pass |
| Reading the weight of a DrPepper Can. | 1 | The measured weight of DrPepper using scales = 165g. | The measured weight using sensor and load cell: Between 155g and 175g | 158g | Pass |
| Displaying the number of cans | 1 | Number of Cans: 4 | 4 | 3 | Fail |
| 2 | Number of Cans: 4 | 4 | 4 | Pass |
| 3 | Number of Cans: 3 | 3 | 3 | Pass |
| Displaying the temperature on a web-application | 1 | Temperature reading | Displays the same temperature as the OLED display | Displays the same temperature as the OLED display | Pass |
| Displaying the humidity on a web-application | 1 | Humidity reading | Displays the same humidity as the reading from the console | Displays the same humidity as the reading from the console | Pass |
| Displaying the number of products | 1 | Number of Cans: 3 | 3 displayed on the webpage | 3 displayed on the webpage | Pass |

# References

Adafruit (2020a) *DHT Humidity Sensing on Raspberry Pi or Beaglebone Black with GDocs Logging* [Internet]. Available from <https://learn.adafruit.com/dht-humidity-sensing-on-raspberry-pi-with-gdocs-logging/python-setup> [Accessed 7th October 2020].

Adafruit (2020b) *Monochrome OLED Breakouts* [Internet]. Available from <https://learn.adafruit.com/monochrome-oled-breakouts/python-wiring> [Accessed 10th October 2020].

Chart.js (2020) *Chart.js* [Internet]. Available from <https://www.chartjs.org/docs/latest/> [Accessed 12th October 2020].

Facebook (2020) *Getting started* [Internet]. Available from <https://reactjs.org/docs/getting-started.html> [Accessed 9th October 2020].

Fritzing (2019) *Fritzing* [Windows Application]. Vers. 0.9.4. Available from <https://fritzing.org/download/>.

Indrek, L. (no date) *50kg Load Cells with HX711 and Arduino. 4x, 2x, 1x Diagrams* [Internet]. Available from <https://circuitjournal.com/50kg-load-cells-with-HX711> [Accessed 5th October 2020].

Jayaram, A. (2017) *Smart Retail 4.0 IoT Consumer Retailer Model for Retail Intelligence and Strategic Marketing of In-store Products* [Internet]*.* Available from <https://www.researchgate.net/profile/Athul_Jayaram/publication/314114955_Smart_Retail_40_IoT_Consumer_Retailer_Model_for_Retail_Intelligence_and_Strategic_Marketing_of_In-store_Products/links/58b60980aca27261e516626b/Smart-Retail-40-IoT-Consumer-Retailer-Model-for-Retail-Intelligence-and-Strategic-Marketing-of-In-store-Products.pdf> [Accessed 25th September 2020].

Microsoft (2019) *What is Azure IoT Hub?* [Internet]. Available from <https://docs.microsoft.com/en-gb/azure/iot-hub/about-iot-hub> [Accessed 29th September 2020].

Microsoft (2020a) *Introduction to Azure Functions* [Internet]. Available from <https://docs.microsoft.com/en-gb/azure/azure-functions/functions-overview> [Accessed 29th September 2020].

Microsoft (2020b) [Internet]. Available from <https://docs.microsoft.com/en-gb/azure/cosmos-db/introduction> [Accessed 30th September 2020].

Microsoft (2020c) *Welcome to Azure Cosmos DB* [Internet]. Available from <https://azure.microsoft.com/en-gb/pricing/calculator/> [Accessed 30th September 2020].

Microsoft (2020d) *What is IoT security?* [Internet] Available from <https://azure.microsoft.com/en-us/overview/internet-of-things-iot/iot-security-cybersecurity/> [Accessed 2nd October 2020].

Microsoft (2020e) Provisioning devices with Azure IoT Hub Device Provisioning Service [Internet]. Available from <https://docs.microsoft.com/en-us/azure/iot-dps/about-iot-dps> [Accessed 2nd October 2020].

Microsoft (2020f) *Quickstart: Send telemetry from a device to an IoT hub and read it with a back-end application (Python)* [Internet]. Available from <https://docs.microsoft.com/en-us/azure/iot-hub/quickstart-send-telemetry-python> [Accessed 3rd October 2020].

Microsoft (2020g) *Azure IoT Hub trigger for Azure Functions* [Internet]. Available from <https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-iot-trigger?tabs=javascript> [Accessed 8th October 2020].

Microsoft (2020h) *Azure Cosmos DB output binding for Azure Functions 2.x and higher* [Internet]. Available from <https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-output?tabs=javascript> [Accessed 8thth October 2020].

Usman, A. (2020) *Lesson One*. Presented during Week 1. York St John University, York [21st September].

Zak, M. (2018) *HX711* [Internet]. Available from <https://github.com/gandalf15/HX711> [Accessed 5th October 2020].