IoT – automated product monitoring

3CB105: Ben Coxford

# Internet of Things – Smart Retail Racking

## Problem Definition

Jayaram (2017) gives an explanation on how the Internet of Things can facilitate ‘Smart Racking’ where sensors are placed to estimate the quantity of products on the racking and the temperature of the racking. He further explains how this will enable automatic reordering and adjusting the temperature on the racking to preserve the products freshness and prevent damaging.

## Proposed Solution

Described by Usman (2020), the model I am using is the IoT stack as shown in appendix A, figure 1. To measure the products weight, I plan to use weight sensors which 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 (2020) 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 (2020), the data is formatted and stored in Microsoft Azure’s CosmosDB (2020). 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 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 the smart racking using multiple weight sensors. This must be displayed on a web-application.
* Display archived data in the form of a graph.

## Required hardware and cost

|  |  |
| --- | --- |
| 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 |

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 LED’s 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.

Microsoft Azure is an affordable service which 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 (2020).*

## Legal and Ethical Evaluation

### Security

There is a risk with Azure’s IoT Hub and CosmosDB. Azure reduces the risk of any unauthorized access or tampering using the X.509/TLS based handshakes and encryption. This prevents “Man in the Middle” attacks as well as unauthorized access to the IoT hubs endpoint. Using cloud-storage opens the possibility to SQL injections and unauthorized access. To mitigate the risk of unauthorized access and SQL injections, a read-only key is used by the application; therefore, those with access to the site/database can only read values from the database. The data itself does not contain any sensitive information and does not pose a threat to the organisation. For further development, Azure’s authentication would be added to allow accounts from a certain organisation access.

## Development Phase

### Wiring and Mounting Sensors

Wiring the project was straight forward following the circuit diagram *(See appendix B)*. 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 misconnecting 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 its own address, allowing each display to be accessed individually.

Next, I wired the load cells to the HX711 sensors. The HX711 sensor amplifies the current om order for the Pi to read the values. Initially, I found the sensors to be temperamental and I had to resolder each one. Afterwards I had no trouble reading the values from the sensor.

Lastly, I attached the LED’s and resistors to the breadboard and connected the Raspberry Pi to the T-Cobbler.

### Azure Services Setup

After following the documentation and multiple tutorials by Microsoft (2020), the setup to the backend solution to the system was complete. This involved initializing cloud storage, the IoT Hub and Azure Functions.

First, I set up the telemetry to the IoT Hub through Python and this was straight forward, sending the data using the MQTT protocol using Azure IoT Hubs Python framework.

Secondly, I logged this data using Azure Functions, ensuring that the function I created had been receiving the correct data. After the function was receiving the data, I adapted the function to format and send the data to CosmosDB, Azure’s cloud-storage solution.

### Developing the Software

Reference the setup tutorials.

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

To begin, I set up multiple classes to encompass each sensor individually, along with a main program. Next, I programmed the DHT22 sensors using a framework from Adafruit (2020),

### Developing the Web Application

### Data Analytics

### Example 1 – 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 2.

### Example 2 – 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.

## 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 setup 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 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 below.

GITHUB link OF FINAL Solution!

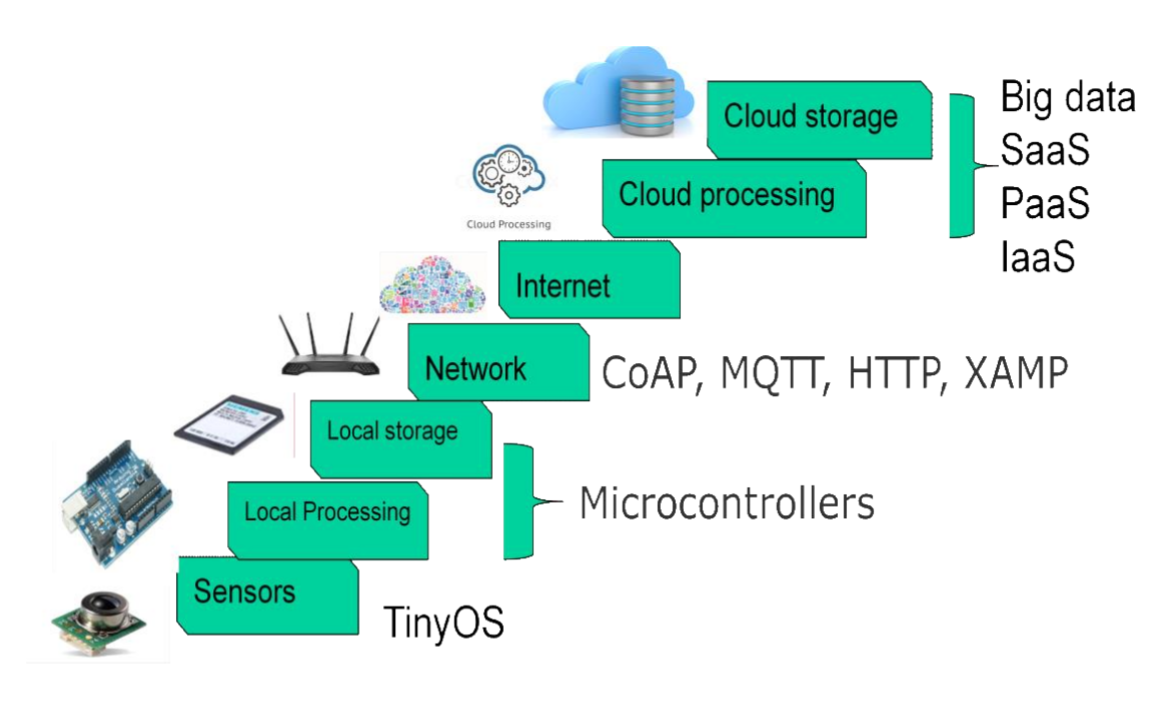
The video demonstration is published to YouTube (2020) and can be watched through the link below.

<https://www.youtube.com/watch?v=tCmcQORV5FA&t=492s>

## Appendices

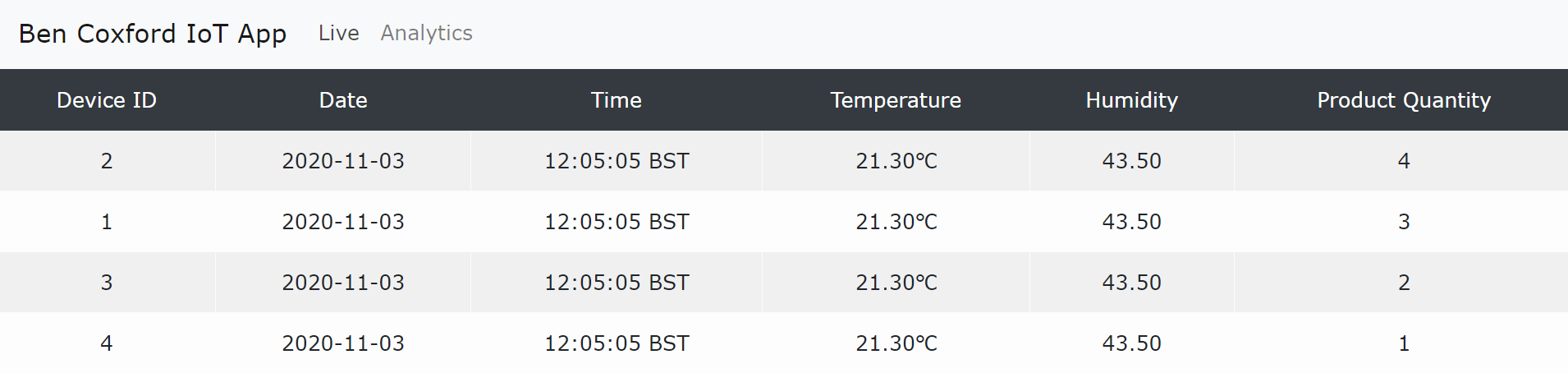
## Appendix A

### Figure 1



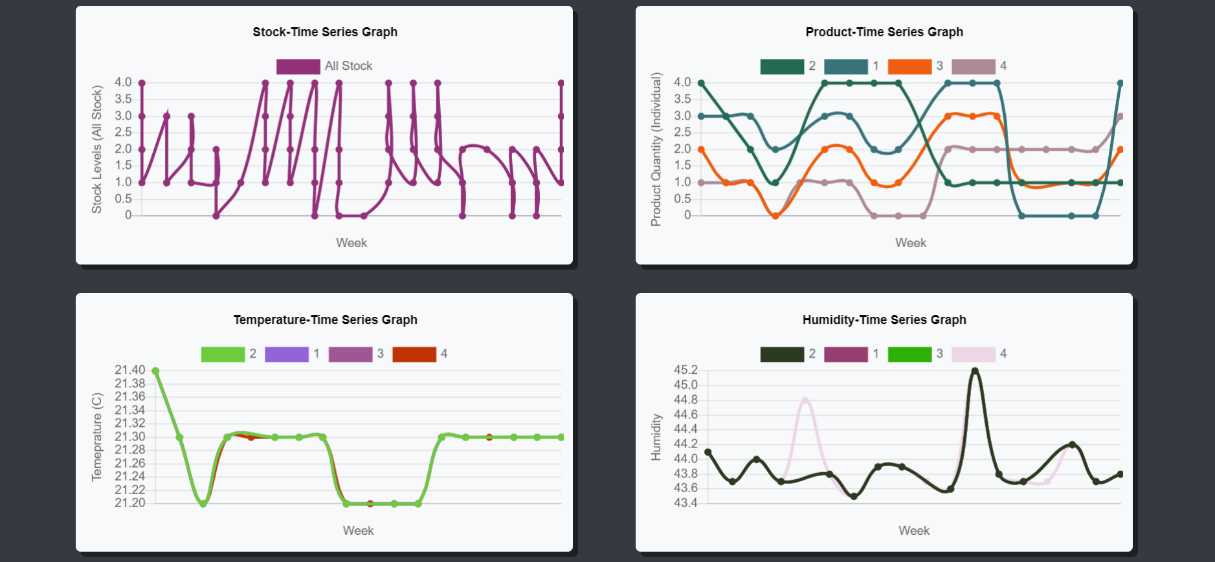
The IoT Stack, presented by Usman (2020)

### Figure 2



Live feed of each devices data

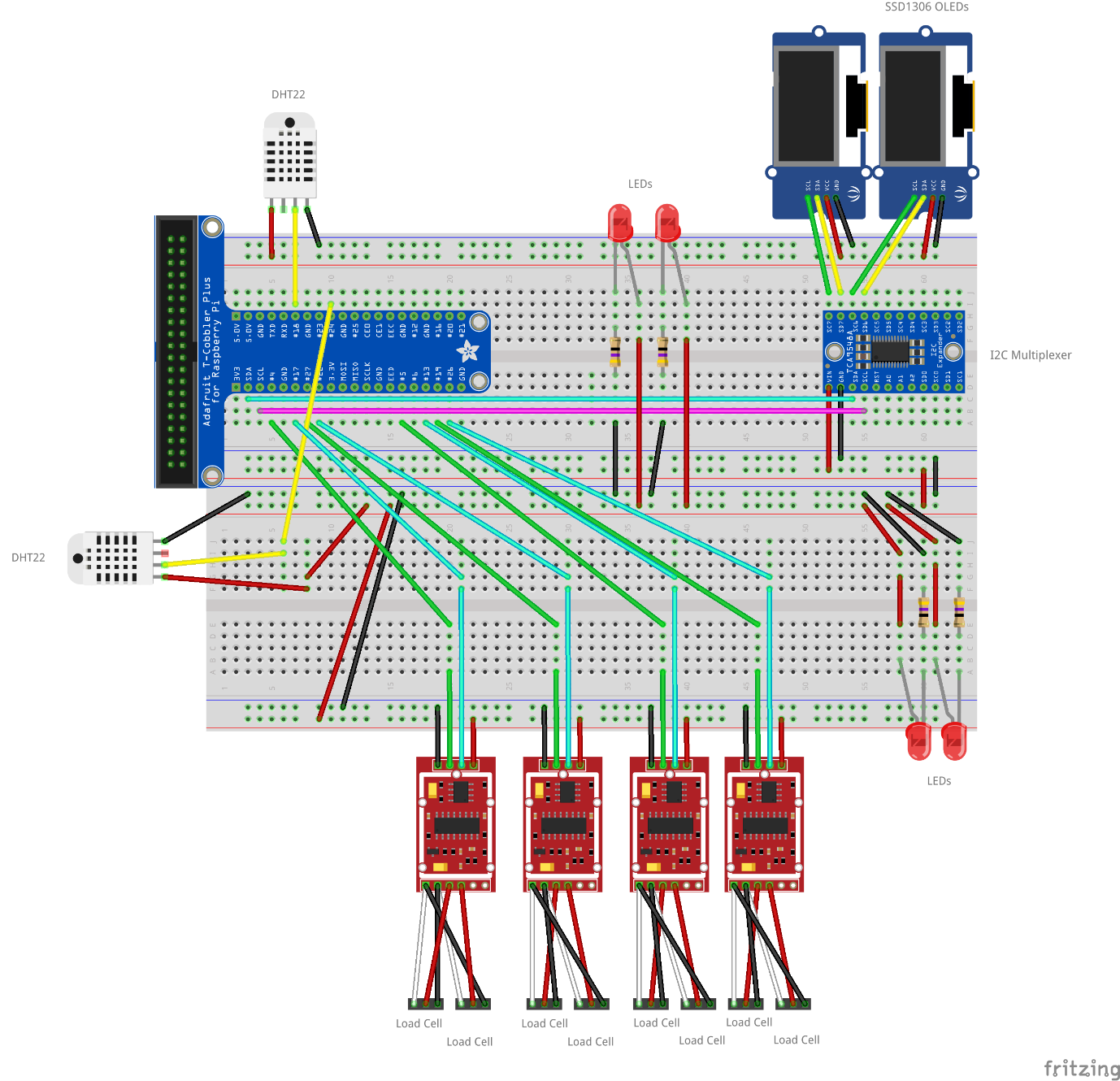
### Figure 3



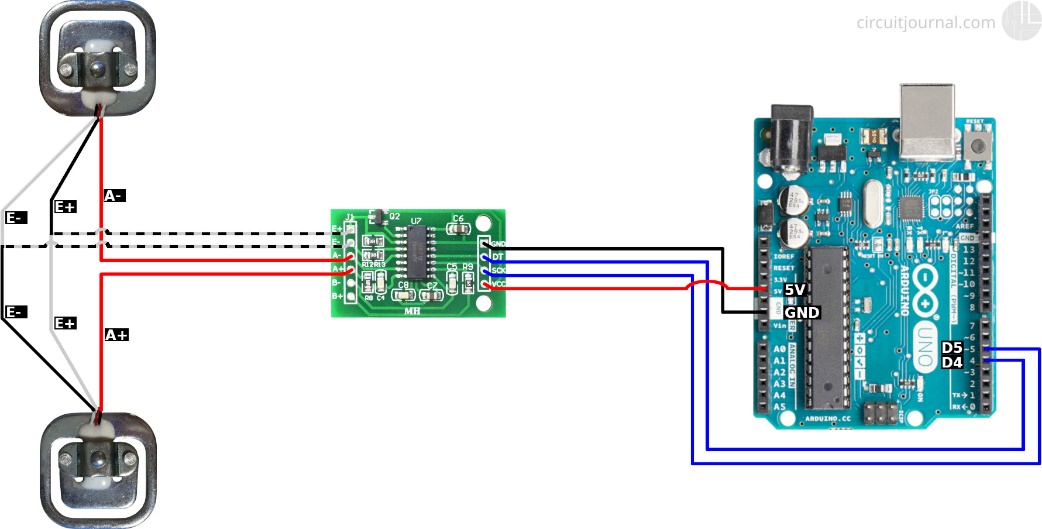
Time-Series graphs showing archived data

## Appendix B

### Diagram 1



### Diagram 2



## Appendix C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 LCD display. | 1 | Console Reading: 21.3 | OLED Display reading: 21.3 | 21.3 | Pass |
| Reading the weight of a Dr Pepper Can. | 1 | Measured weight of Dr Pepper using scales = 165g. | 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

Jayaram 2017 <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>

<https://docs.microsoft.com/en-us/azure/iot-hub/quickstart-send-telemetry-python>

<https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-iot-trigger?tabs=javascript>

<https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-cosmosdb-v2-output?tabs=javascript>

Indrek Luuk <https://circuitjournal.com/50kg-load-cells-with-HX711>

<https://azure.microsoft.com/en-gb/pricing/calculator/>

<https://learn.adafruit.com/monochrome-oled-breakouts/python-wiring>

<https://learn.adafruit.com/dht-humidity-sensing-on-raspberry-pi-with-gdocs-logging/python-setup>