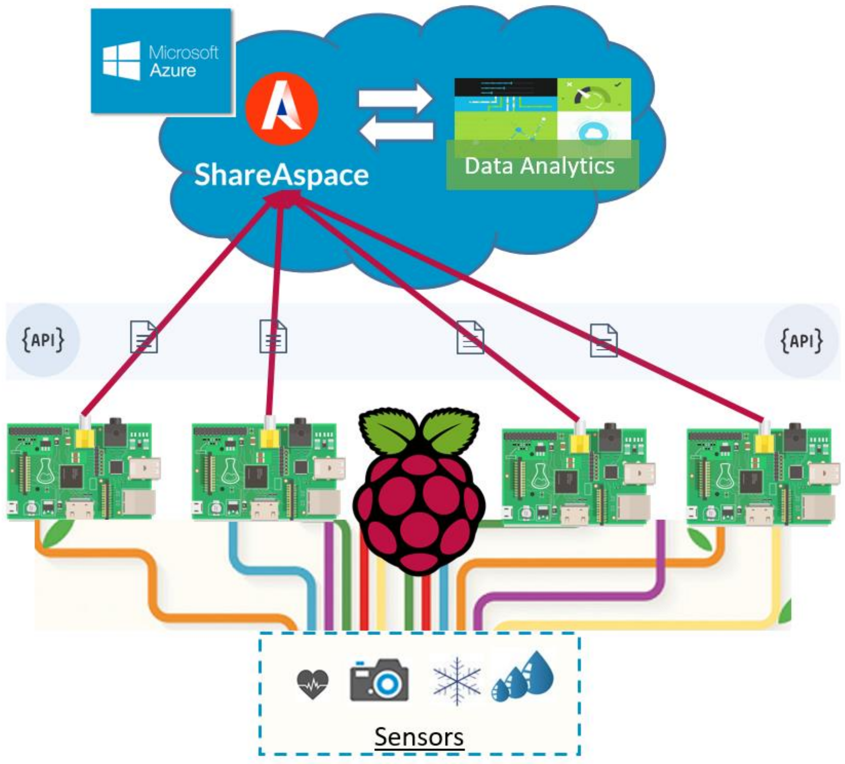
Maximilian Murach-Ward

Product Data Management and Sharing in the age of Internet of Things

MSci Hons Computer Science

(With Industrial Experience)

09/06/2017



Name: Maximilian Murach-Ward

Student ID: 33309337

Dissertation Title: Product Data Management and Sharing in the age of Internet of Things

Module: SCC 421

Date: 9 June 2017

Abstract

Internet of Things (IoT) has recently “taken off” and has become a popular research topic due to advances in technology, which have enabled the development of small, cheap, and powerful internet-connected devices. However, a research area that has not been explored in depth is how IoT might be used in the field of Product Data Management (PDM). Using a system composed of sensors and Raspberry Pi boards along with ShareAspace (PDM sharing software), this project explores *whether* principles of IoT might be applied to product data management and sharing, and, if so, *how* they might be applied. The system described here was deployed in a company’s office to mirror a real-world scenario. In this scenario, the Raspberry Pi boards fed live temperature data from sensors (placed in two different rooms) into ShareAspace. Several technical tests as well as two focus groups were used to evaluate the system built. Technical test results were hugely successful and focus group results suggested that IoT has the potential to play a part in the future of product data management. Overall this project was successful in achieving the aims set.

Contents

1 Introduction

IoT is simply having lots of devices connected to the internet and having them speak to each other or to different applications. Examples of IoT range from devices like the Nest thermostat (NEST, n.d.) which is a thermostat that can be controlled using a smart-phone app, all the way to smart cities, which have sensors collecting data such as when bins need emptying (Butcher, 2014).

IoT has started to "take off" a lot more in recent years as computing equipment has become ever smaller and less expensive to manufacture. This means it is very cheap and simple to deploy a variety of devices collecting a diverse range of data.

During a recent placement, I worked for a company called "Eurostep". Eurostep work in the area of the exchange of product data and this spurred my interest in how IoT could be applied to enhance automation and improve customer experience.

So, the main aim of this project was to research the development of an IoT application in the context of product data exchange and sharing across different companies. My research question was:

Why and how is IoT and real-time data useful in the context of product data sharing and exchange?

My colleagues and I decided that deploying a real-world example of a system that collected real-time data which was then forwarded to a product sharing application would be a good way of exploring and answering this research question. We also felt that holding focus groups to discuss such a system would help to evaluate its utility and functionality.

As has been noted, Eurostep specialize in the storing and exchange of product data across extended enterprises and have created a piece of software to facilitate such activities. This software is called "ShareAspace". Eurostep expect that, in the near future, their customers will increasingly wish to store and share *real-time* data using ShareAspace and they asked me to create a "proof of concept" system in order to investigate how well ShareAspace coped with streams of real-time data.

One reason I found building the "proof of concept" system particularly interesting was that I realized that systems based on the principles I was working to could have real-world uses for many of Eurostep's customers. Eurostep's customers include defence and manufacture companies who share data - internally and with suppliers and customers – relating to vehicles, equipment, and premises. I thought it would be beneficial to build on my "proof of concept" system and create a real-world application that would further demonstrate how the collection, analysis and sharing of real-time data might be useful to Eurostep's existing customers and similar companies.

The aims of the project included:

a) Finding out about related systems.
b) Building a real-life system.
c) Investigating how such a system could be applied to companies.
d) Then, exploring how such a system could help those companies.

In the *Background* chapter I shall provide more details about Eurostep as a company and the system I built while working there, and I shall cover literature relating to this project. This chapter will aim to address aim (a).

In the *Design* chapter I shall describe the designs for the system I developed and why I chose the design I did. This chapter will also contain diagrams of the different designs.

In the *Implementation* chapter I shall go into detail on the system and describe the code that enabled the system to work and how that system was deployed. There will also be pictures illustrating the deployed system. In this chapter I will aim to address aim (b).

In the *System in Operation* chapter I shall show what the system is like to use by walking through the main features and using screen-dumps and diagrams to illustrate the different steps. In this chapter I shall also include a process description - a lower level description of what is going on at each step.

In the *Testing and Evaluation* chapter I shall describe the testing methodology used. The results of the technical tests will be presented and summarized along with a summary of the results from the focus groups held. I shall reflect upon the results in an effort to provide an insight into the research question. In this chapter I shall aim to address aim (c) and (d).

Finally, in the *Conclusion* chapter I shall summarize the answer to the research question and assess whether the aims of the project have been met. I shall talk about any revisions to the design or implementation of the system along with any future work and any lessons I learned by doing this project. In this chapter I shall also aim to address aim (d).

2 Background

2.1 Company background

As has already been noted in chapter 1, during my placement in the Lent term, I worked at Eurostep and I continued to work with them throughout this project. Eurostep is a data modelling consultancy firm but have, in recent years, moved into the development (and supply) of a software application called “ShareAspace”.

Figure 1 - Eurostep Logo (source - eurostep.com)

As was related in my project proposal, ShareAspace facilitates the (internet based) storage and sharing of product data across different companies. It is designed to complement PDM (Product Data management), PLM (Product Lifecycle Management), and ERP (Enterprise Resource Planning) systems by allowing companies with such systems to exchange and allow access to their data in a controlled fashion with their suppliers, clients, and collaborators. ShareAspace is based on the ISO 10303 set of international standards (also known as “Standard for the Exchange of Product data” or “STEP”) the exchange and sharing of product data – particularly AP242 (“Managed model based 3D engineering") (STEP AP 242, n.d.) and AP239 (“Product Life Cycle Support” or “PLCS”) (PLCS, n.d.). Because ShareAspace is based on standards, its users can continue to use their existing product-data/resource-planning systems or migrate to different systems and are not locked into relationships with particular software suppliers. Users of ShareAspace can also share data with others who may use entirely different system. (Murach-Ward, 2017).

At the start of my project placement I planned to present the system I produced to some of Eurostep’s customers and find out whether they thought such a system could be useful. I did however envisage that this might not be possible because Eurostep could opt for the presentation of a fully operational large-scale system to their customers as opposed to a small-scale testbed. After discussing this with Eurostep it became clear that they were not comfortable with the idea of presenting my system to customers as it was still in its very early stages and needed to be developed further before they wanted to consider such a move. However, I feel that I still received all the feedback necessary at this stage from the focus groups discussed in chapter 6.

2.2 Placement system

2.2.1 Description

Currently, ShareAspace is designed so that it mainly works with “static” data, although the “static” data is data that may often be revised. Eurostep wanted me to develop a system which would allow them to test how well ShareAspace would cope with live “non-static” data that changes in real-time.

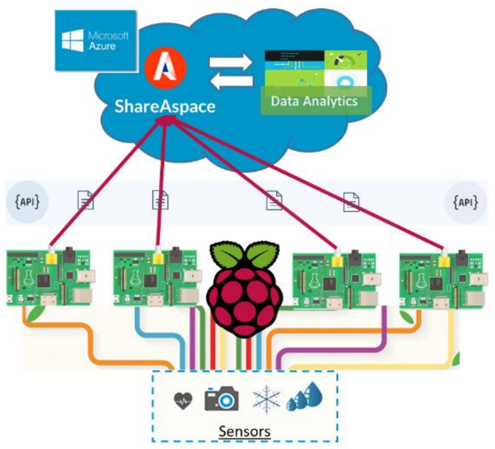


Figure 2 - Placement System Diagram

The system I built to do this consisted of an assembly of sensors which collected sensor data, (primarily temperature data). This data was transmitted over Bluetooth to a set of Raspberry Pi boards (Raspberry Pi, n.d.) . The Raspberry Pi boards were all running a piece of software called Node-RED (nodered, n.d.). Node-RED is an IoT graphical programming language based on Node-JS (Node.js, n.d.). To enable the

Raspberry Pi boards to use ShareAspace's REST (representational state transfer) API (application program interface) I wrote a ShareAspace library for Node-RED. This library allowed the Pi boards to authenticate easily with the ShareAspace server along with sending data to ShareAspace and retrieving and editing data on ShareAspace.

This system allowed us to perform lots of different kinds of tests on ShareAspace so we could assess how well ShareAspace copes with live data and test which configurations of ShareAspace worked best for live data.

2.2.2 Hardware

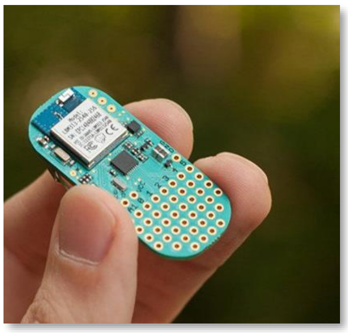


Figure 3 - The Light Blue Bean (source - makezine.com)

For the system, two pieces of hardware were essential: a sensor which could collect different kinds of sensor data, and a computer, which would forward sensor data to ShareAspace. Both pieces of hardware would ideally be cheap and very small, as this would allow the system to be more flexible in terms of the type of environments in which it could be deployed. as I this would mean that the system would be more scalable.

The sensors used for the system were "Light Blue Beans". A Light Blue Bean sensor is a matchbox sized PCB with a temperature sensor and Accelerometer built in on it, with the option to add more sensors to the PCB. The Light Blue Bean also has built-in low energy Bluetooth which meant the Light Blue Beans did not need to be placed close to the Raspberry Pi boards as they do not need to be physically connected to the Pi boards. This allowed for a lot of freedom in deciding where to place the sensors. The Light Blue Bean also works at temperatures from -40°C to 85°C. All these features meant that the Light Blue Bean was well suited to our requirements.

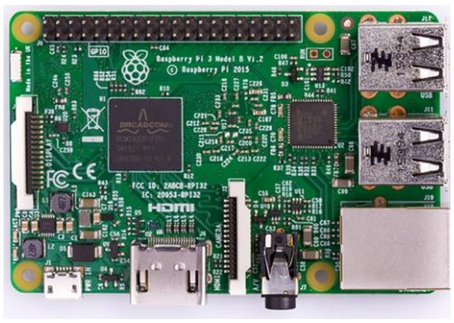


Figure 4 - The Raspberry Pi 3 (source - raspberrypi.org)

Raspberry Pi 3 boards were used first to collect sensor data from the Light Blue Beans. The Raspberry Pi 3 has also got built-in low-energy Bluetooth so a separate Bluetooth dongle was not needed. Having the sensors and Raspberry Pi boards communicate over Bluetooth also meant that multiple sensors could be connected to a single Raspberry Pi.

The Raspberry Pi 3 is also quite a powerful small computer in its own right, with 1GB of RAM and a quad core processor. These specifications ensured that the Pi did not cause a bottleneck in the system. Each of the Pi boards in the system used Linux as the operating system. The distro of Linux for the Pi includes a piece of software called Node-RED which I shall cover in detail in the software section of this chapter. Node-RED is responsible for making the connection to the sensor over Bluetooth, collecting the sensor data and then sending the sensor data over a network to a server running ShareAspace. The Raspberry Pi 3 also has built in WIFI and an Ethernet port. This means that connecting the Pi to a network is simple and again does not require a dongle. Again, these features meant the Raspberry Pi 3 was ideal for the system.

Both these pieces of hardware are fairly cheap with the Light Blue Bean (sourced from the USA) costing around $34 and the Pi 3 costing £32 at the time the system was built. Both the Pi and Bean are very small and, as has been noted, can be placed apart (as long as they are within Bluetooth range) allowing for much more flexibility when deploying the system.

2.2.3 Software

A program was required to run on the Pi boards which would collect the data over Bluetooth from the Light Blue Beans but then also authenticate with the ShareAspace server and then communicate with the ShareAspace API. This is where Node-RED came in.

Node-Red is an IoT graphical programming tool. It works by connecting "nodes" (blocks) together to form a "flow" (basically a program). It is based on Node-JS and comes with a large library of nodes. You can, however, also write your own nodes.

Node-RED already has a Light Blue Bean library and this meant that it was fairly simple to connect to a Light Blue Bean from a Pi and read temperature data.

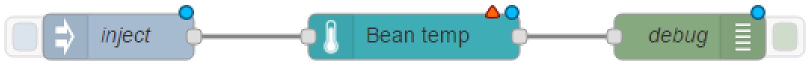


Figure 5 - A Simple Flow

Figure 5 shows a simple flow I created in Node-RED consisting of a Bean temp node which connects to the Light Blue Bean and takes a temperature reading when the "inject" node is triggered. The debug node prints the temperature reading out to a debug console.

However, flows to send the temperature to ShareAspace, get data from ShareAspace, and edit data on ShareAspace were also required. Communication with ShareAspace is mediated using HTTP requests.

After doing some research I could not find any suitable Node-RED libraries for communication with ShareAspace using HTTP. Therefore, I decided to write my own ShareAspace communication library. Consisting of an "authenticator" node, a "create" node, a "read" node and an "update" node. How this works is explained in detail in the implementation chapter.

2.2.4 System limitations

There were a few limitations of this placement system. For starters, there was no ability to perform analytics on any of the sensor data collect. Another limitation is that this system had not been setup with a real-world data model, nor had the system been deployed in a real-world environment. This is the reason why I decided to build on this placement system to make a real-world system.

2.3 Literature review

As was noted in Section 2.1, ShareAspace is designed to complement PDM (Product Data Management), PLM (Product Lifecycle Management), and ERP (Enterprise Resource Planning) systems.

PDM and PLM systems track a large range of data about a product. This can include technical specifications, design specifications, drawings, deployments, obsolescence, and disposal information. PDM and PLM systems are very useful for companies because they make it very easy to reuse product designs or track any changes to a product throughout its lifecycle. As the Siemens website puts it: this improves "productivity and reduce cycle time" and helps to "Reduce development errors and costs" (Siemens, n.d.) and there are many other benefits to using a PDM system. Some examples of PDM and PLM software are "Windchill" by ptc (ptc, n.d.), "ENOVIA" by Dassault Système (DASSAULT SYSTÈMES, n.d.) and "Teamcenter" by Siemens (Siemens, n.d.).

ERP systems collect and store all kinds of business data from different business activities such as financial data, business intelligence data or project management data. The system keeps all this data together making it easier to manage. ERP systems provide many advantages including "easier collaboration, faster decision making, and improved overall productivity for the whole team" (Microsoft, n.d.).

ShareAspace itself would only be a partial substitute for a fully-fledged PDM, ERP or PLM system. Its focus is on allowing different PDM, ERP and PLM systems to talk to one another. As Eurostep explain "ShareAspace is a hub that extends rather than replaces existing systems such as PDM, PLM, ERP or MRO." (Eurostep, n.d.).

Since the beginning of internet computing, people have built many different IoT systems for collecting sensor data. Some of these even involve the same hardware (Raspberry Pi and Light Blue Bean) and software (Node-Red) that I used. Systems developed by amateur enthusiasts include the use of Light Blue Bean and Node-RED for sending an email if the temperature drops too low (embeddedcomputing, n.d.). or the use of Raspberry Pi, Node-Red and Light Blue Bean to detect water and flooding in a house (KarelK, n.d.). The way these systems initially collect data from the sensor is very similar to the way this is done in my system; but this is where the similarities end.

The main difference with my system is that it combines the IoT sensor data collection (similar to the two systems mentioned) with a product-data sharing system (in this case ShareAspace). At the time of writing this I did not find evidence of many systems involving PDM, PLM and IoT, however there was some literature on this topic which suggests ways in which it might be useful to use IoT with PDM and PLM.

A proposed implementation of such a system is discussed in (Thoben & Lewandowski, 2016). A diagram of the system can be seen in the Figure 6. Here data is collected from different inputs but "mainly raw data from the turbine itself is collected". Next the data is processed to "gain information for the design department and the life-cycle management" (PLM). This proposed system shows how IoT could be used in the context of product data sharing as sensor data (from the turbine) is used to add information to a product (a wind turbine) and then stakeholders can check this sensor information attached to a product.

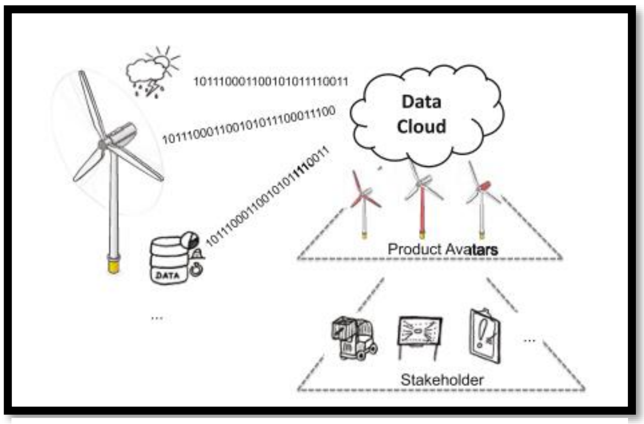


Figure 6 - Wind Turbine Example
(source - (Thoben & Lewandowski, 2016))

The blog "Beyond PLM" (Shilovitsky, 2014) talks about three things PLM could do with IoT. The first point the blog makes is that IoT could "improve MRO solutions" (MRO - Maintenance, Repair, and Operations) because "Sensors will provide a data foundation to connect and communicate with products. It will help to optimize future service schedules and lower maintenance cost". The second point the blog makes is that IoT could "improve requirement management function" because "It will be beneficial to know that some product functions are not in use and *inject* this information in future requirement analysis and management tools". The final point the blog makes is that IoT could allow "product performance monitoring" the blog suggests that "if you can get real performance data from an engine and other parts of the car, airplane, computer, toothbrush" this would help "to deal with growing sets of regulation and other environmental requirements". To summarize, this blog suggests that IoT could have many uses in product management systems.

The literature review helped me think of features for and the technical architecture of my system. It has done by this by giving me an insight into similar systems which led me to achieving one of my objectives - finding out about related systems.

3 Design

3.1 The real world-system design

The real-world system I built was an extended version of the placement system but deployed in a real-world environment. The main difference was that the extended system had data analytics and events triggered by certain temperature thresholds.

In the original design for the system, it was assumed that Microsoft Azure would speak to ShareAspace directly. However, after some testing (discussed in the Testing & Evaluation – Technical test 11) which suggested ShareAspace might have trouble coping with millions of sensor readings, I decided that it made sense to send every single temperature readings straight to Azure from the Pi and only send temperature readings to ShareAspace if there were any changes in temperature. Doing this would significantly reduce the number of readings stored by ShareAspace while still retaining the key information about changes in temperature. This way, if a worrying or interesting temperature change were detected, users could check Azure for a full history of temperature readings and generate reports of temperature history using another Microsoft cloud service like PowerBI.

Figure 7 below provides an architectural diagram of the system:

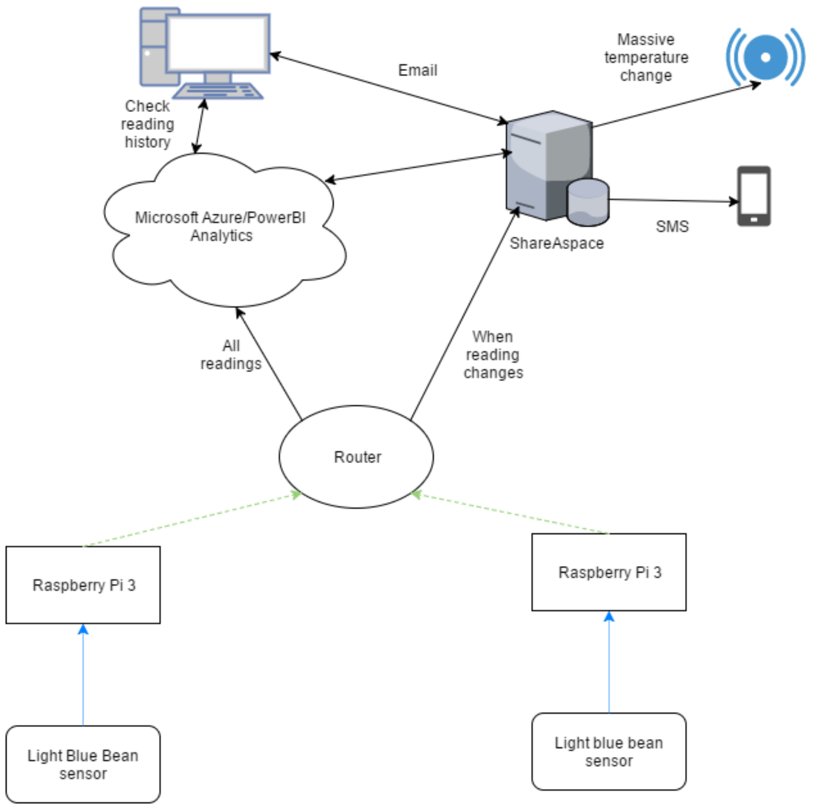


Figure 7 - Architectural Diagram

This real-world system consists of two Light Blue Bean sensors situated in two different places in the office which are constantly monitoring the ambient temperature of the room. These sensors are powered by 3V CR2032 batteries so do not need to be powered from the mains. These two sensors are connected over Bluetooth to two Raspberry Pi 3 boards, one sensor per Pi (although it would be possible to have many sensors per Pi). The Raspberry Pi 3s are connected to the internet via a router in the office so they both have access to both the internet and local network. This means that they are able to communicate with an internal or external server as well as with cloud services such as Microsoft Azure. The Raspberry Pi boards are powered from the mains; however, they could also be powered by batteries. In the case of this system, a local ShareAspace server is running ShareAspace Nova (the latest version of ShareAspace). This server is also connected to the internet so ShareAspace can send emails, alerts, SMS, etc. The ShareAspace server is a laptop with a dual core processor, solid state storage, and 8GB of RAM running Windows 10.

3.2 Deployment plan

Figure 8 illustrates the deployment plan in the Eurostep UK office. The floorplan of the office shows two yellow devices. These devices represent the Light Blue Bean sensors which are placed in two different rooms in the office: one in the main office area and one in the kitchen. These sensors are connected by blue arrows (representing Bluetooth connections) to two red raspberry logos representing the Raspberry Pi boards. The Raspberry Pi logos are connected via black arrows to the router, and the router is connected to the symbol representing the ShareAspace server.

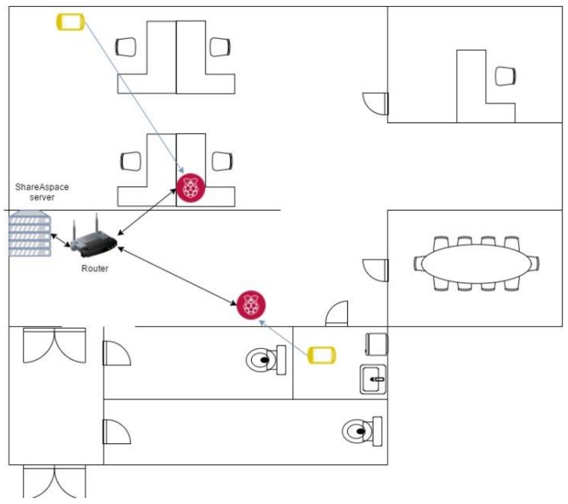


Figure 8 - Deployment Plan

3.3 Data models

A data model is a conceptual model of how data items relate to each other. The Systems Modelling Language (SysML) diagram below shows a zone breakdown (ZoneBreakdownStructure) and the relationships between the zone elements (ZoneElements) and the sensors (RealizedParts) and between the sensors and the properties measured and recorded by those sensors.

The *blocks* used to construct this data model come from the Product Life Cycle Support (PLCS) standard (ISO 10303-239) on which the ShareAspace product is based. The PLCS/ISO 10303-239 standard is an internationally accepted information model that allows for the consolidation, exchange, and sharing of complex data related to product design, operation, support, and disposal (plcs, 2013).

As my system was meant to represent a real-world scenario I wanted to create a data model of the office - just as many products such as ships, plans or cars are represented in PLCS *Data Exchange Specifications* (DEXs). This data model was then used to specify an appropriate configuration for the ShareAspace server.

Strictly speaking, my model ought to be split up into three different models because the full range of cardinalities cannot be properly represented in a single model. But I put everything in one diagram for simplicity.

The model should show that:

- A zone can have many child-zones.
- One zone can have many sensors.
- One sensor can have many readings.

Figure 9 below is a SysML parametric diagram. This diagram was made using modelling software called "MagicDraw" (nomagic, n.d.).

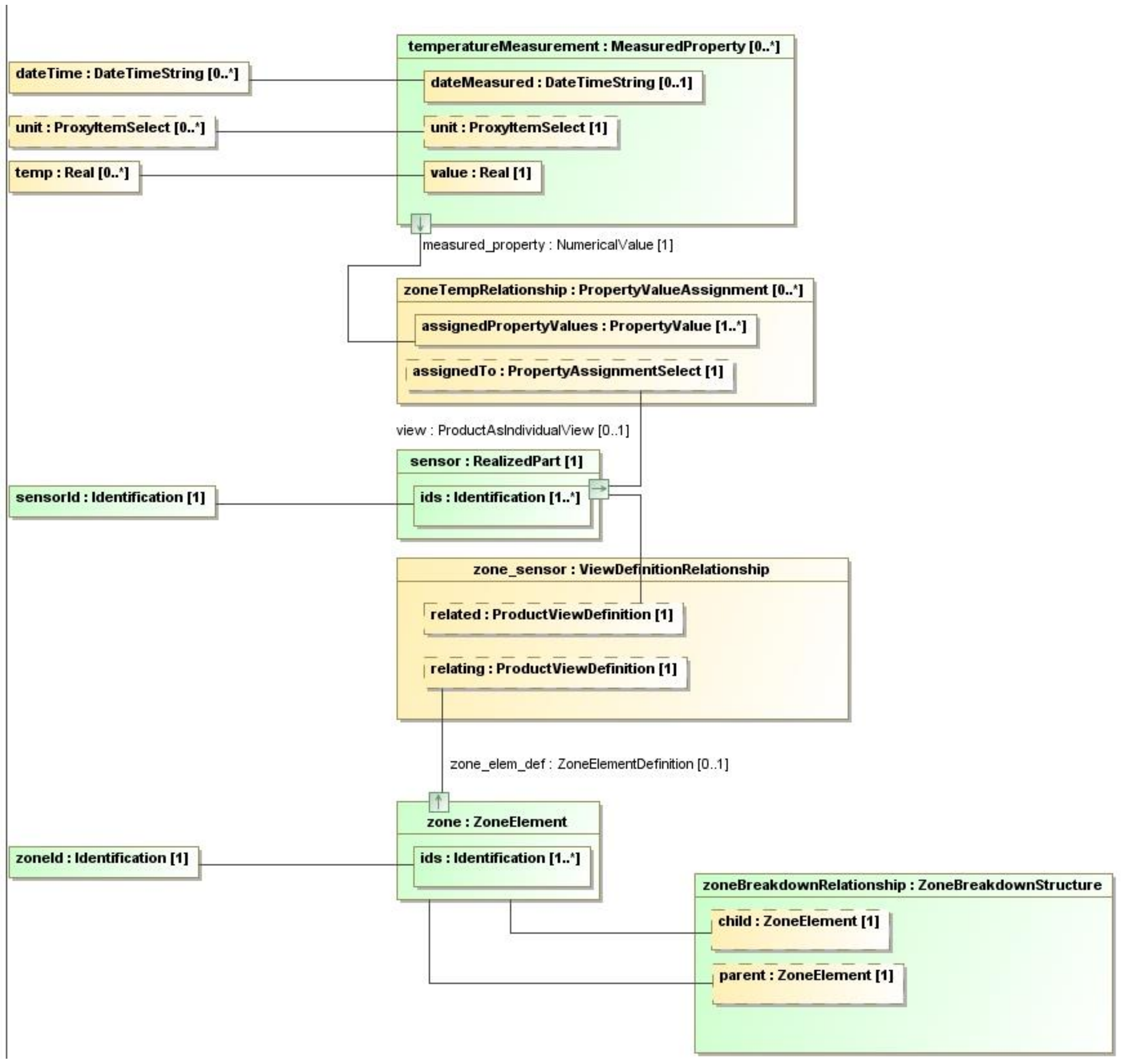

temperatureMeasurement : MeasuredProperty [0..*]
dateTime : DateTimeString [0..*]
dateMeasured : DateTimeString [0..1]
unit : ProxyItemSelect [0..*]
unit : ProxyItemSelect [1]
temp : Real [0..*]
value : Real [1]
measured_property : NumericalValue [1]
zoneTempRelationship : PropertyValueAssignment [0..*]
assignedPropertyValues : PropertyValue [1..*]
assignedTo : PropertyAssignmentSelect [1]
view : ProductAsIndividualView [0..1]
sensor : RealizedPart [1]
sensorId : Identification [1]
ids : Identification [1..*]
zone_sensor : ViewDefinitionRelationship
related : ProductViewDefinition [1]
relating : ProductViewDefinition [1]
zone_elem_def : ZoneElementDefinition [0..1]
zone : ZoneElement
zoneId : Identification [1]
ids : Identification [1..*]
zoneBreakdownRelationship : ZoneBreakdownStructure
child : ZoneElement [1]
parent : ZoneElement [1]

Figure 9 - Parametric Diagram

This model shows that you can create a breakdown hierarchy of zoneElements (eg Office complexes, rooms, areas of rooms etc) and that each ZoneElement may have several sensors (RealizedParts) in it and that each sensor may measure many temperature values - at different times. As has been noted, this served as a specification for a ShareAspace implementation.

In Figure 10 an instance diagram for the model is shown.

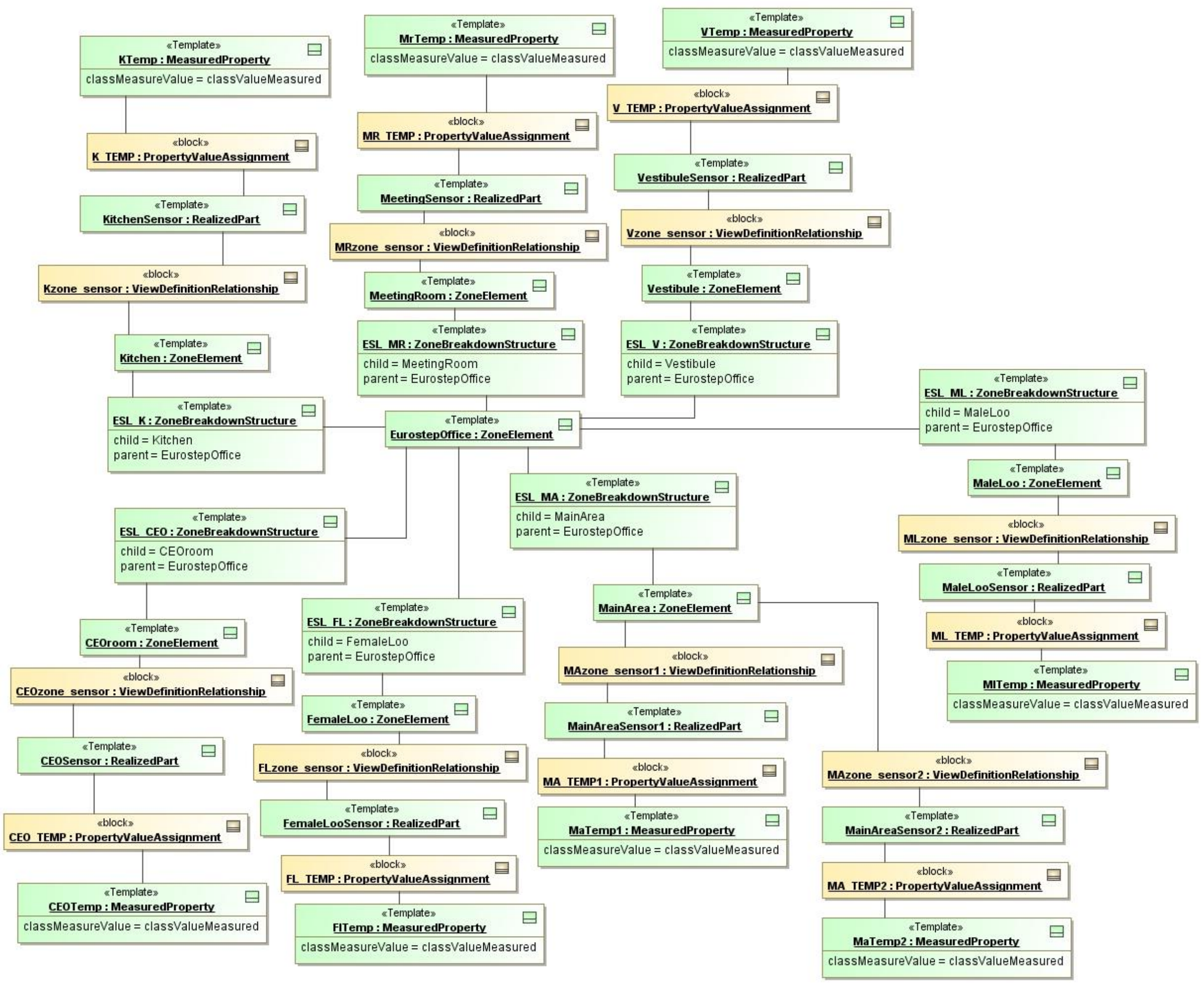

«Template» KTemp : MeasuredProperty
classMeasureValue = classValueMeasured
«block» K_TEMP : PropertyValueAssignment
«Template» KitchenSensor : RealizedPart
«block» Kzone_sensor : ViewDefinitionRelationship
«Template» Kitchen : ZoneElement
«Template» ESL_K : ZoneBreakdownStructure
child = Kitchen
parent = EurostepOffice
«Template» MrTemp : MeasuredProperty
classMeasureValue = classValueMeasured
«block» MR_TEMP : PropertyValueAssignment
«Template» MeetingSensor : RealizedPart
«block» MRzone_sensor : ViewDefinitionRelationship
«Template» MeetingRoom : ZoneElement
«Template» ESL_MR : ZoneBreakdownStructure
child = MeetingRoom
parent = EurostepOffice
«Template» VTemp : MeasuredProperty
classMeasureValue = classValueMeasured
«block» V_TEMP : PropertyValueAssignment
«Template» VestibuleSensor : RealizedPart
«block» Vzone_sensor : ViewDefinitionRelationship
«Template» Vestibule : ZoneElement
«Template» ESL_V : ZoneBreakdownStructure
child = Vestibule
parent = EurostepOffice
«Template» EurostepOffice : ZoneElement
«Template» ESL_ML : ZoneBreakdownStructure
child = MaleLoo
parent = EurostepOffice
«Template» MaleLoo : ZoneElement
«block» MLzone_sensor : ViewDefinitionRelationship
«Template» MaleLooSensor : RealizedPart
«block» ML_TEMP : PropertyValueAssignment
«Template» MlTemp : MeasuredProperty
classMeasureValue = classValueMeasured
«Template» ESL_MA : ZoneBreakdownStructure
child = MainArea
parent = EurostepOffice
«Template» MainArea : ZoneElement
«block» MAzone_sensor1 : ViewDefinitionRelationship
«Template» MainAreaSensor1 : RealizedPart
«block» MA_TEMP1 : PropertyValueAssignment
«Template» MaTemp1 : MeasuredProperty
classMeasureValue = classValueMeasured
«block» MAzone_sensor2 : ViewDefinitionRelationship
«Template» MainAreaSensor2 : RealizedPart
«block» MA_TEMP2 : PropertyValueAssignment
«Template» MaTemp2 : MeasuredProperty
classMeasureValue = classValueMeasured
«Template» ESL_CEO : ZoneBreakdownStructure
child = CEOroom
parent = EurostepOffice
«Template» CEOroom : ZoneElement
«block» CEOzone_sensor : ViewDefinitionRelationship
«Template» CEOSensor : RealizedPart
«block» CEO_TEMP : PropertyValueAssignment
«Template» CEOTemp : MeasuredProperty
classMeasureValue = classValueMeasured
«Template» ESL_FL : ZoneBreakdownStructure
child = FemaleLoo
parent = EurostepOffice
«Template» FemaleLoo : ZoneElement
«block» FLzone_sensor : ViewDefinitionRelationship
«Template» FemaleLooSensor : RealizedPart
«block» FL_TEMP : PropertyValueAssignment
«Template» FlTemp : MeasuredProperty
classMeasureValue = classValueMeasured

Figure 10 - Instance Diagram

This diagram shows a possible instantiation or population of the data model – though it only shows a single measurement per sensor as it would get a bit crowded if many instances of temperature readings were included.

It does however show that (for instance) the Eurostep Limited Offices break down (in my simple example) into Kitchen, Meeting Room, Vestibule, Male & Female lavatories, the main area and the CEO's office. Each zone has one sensor apart from the main area that has two. Each sensor has a MeasuredProperty assigned to it.

Although, in my real-world system, I only have two sensors, the diagrams show what a complete system for the Eurostep Offices might include and what could be represented in a ShareAspace implementation.

3.4 Alternative design

When planning the architecture of the system I considered using a gateway Pi in between other Pi boards and the router as shown in Figure 11. This would have allowed all the sensor data to be configured in one place rather than having to configure every single Pi in the system. This would have made it easier to upgrade the system when, for example, the web address of the server changed. The problem with this approach is that it would introduce a single point of failure within the system and it would also slightly increase the time for readings to travel from a sensor to ShareAspace. There would also have to be an extra Pi for each local network of sensors and this would increase the cost of a large-scale system. For this reason, this alternative design approach was abandoned.

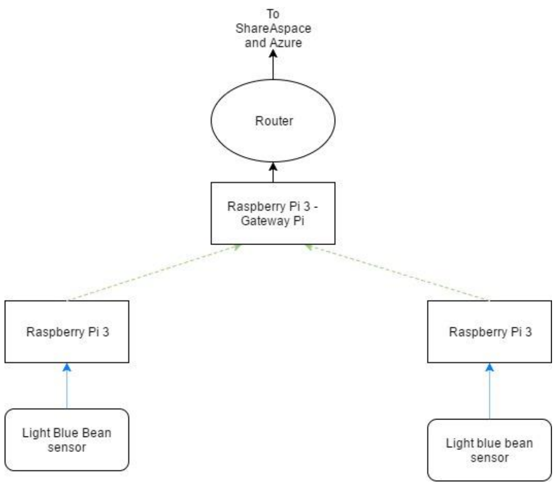


Figure 11 - Alternative Design

3.5 System requirements

The core requirements of the system were as follows:

- The ability to gather sensor data from multiple sensors.
- The ability to authenticate with ShareAspace
- The ability to send data to ShareAspace, receive data from ShareAspace and change the data on ShareAspace.
- The ability to send data to a cloud service such as Microsoft Azure.
- The ability to trigger events such as sending an email or SMS if for example a temperature threshold is reached.
- The ability to perform analytics on the sensor data collected.

Future potential requirements include:

- The ability for the system to be flexible so that it could be modified for different implementations in the future with different kinds of sensor data.

4 Implementation

4.1 Setting up the Light Blue Bean

Setting up the Light Blue Bean sensors was straightforward. First a 3V CR2032 battery (As shown in Figure 12) was inserted into the sensor. Once inserted the sensor was ready to connect to over Bluetooth.

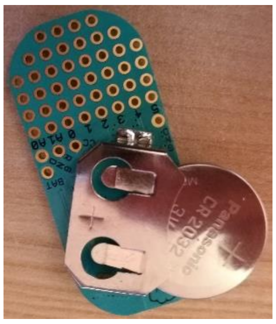


Figure 12 - Light Blue Bean and Battery

After this, I downloaded The Light Blue Bean App on my android smartphone. The app comes with a number of different example programs (called sketches) that can be uploaded to the Bean over Bluetooth and they are a good way to of testing the Bean to see if it is working correctly. The app also enables users to rename different Beans which makes it easier to distinguish between them. I named the two Beans I had "Ebean1" and "Ebean2". Some screenshots of the app can be seen in Figure 13 and Figure 14.

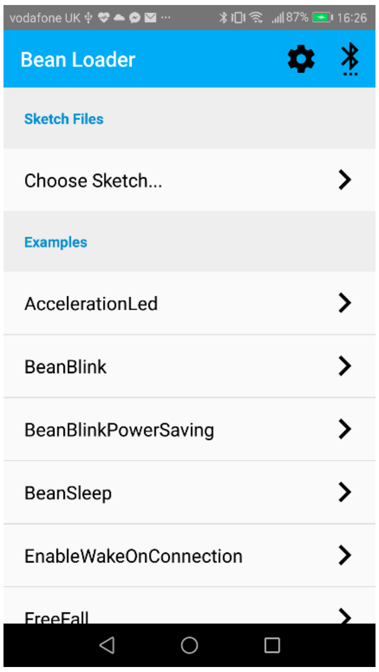


Figure 13 - Sketches

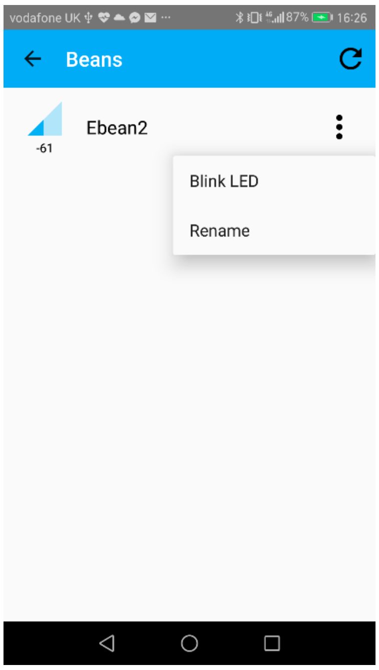


Figure 14 - Renaming

One sensor was placed in the main room and one in the kitchen. I decided to place the sensor in the main room on the windowsill, near a radiator. The sensor in the kitchen was placed on a shelf. Figure 15 and Figure 16 show the two deployed sensors.

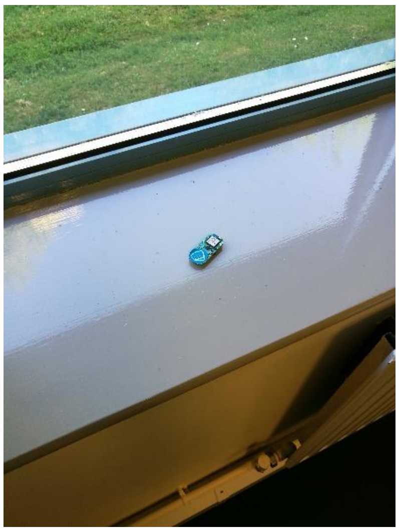


Figure 15 - Main Room Sensor

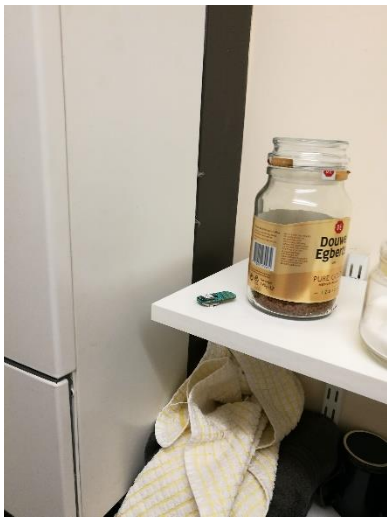
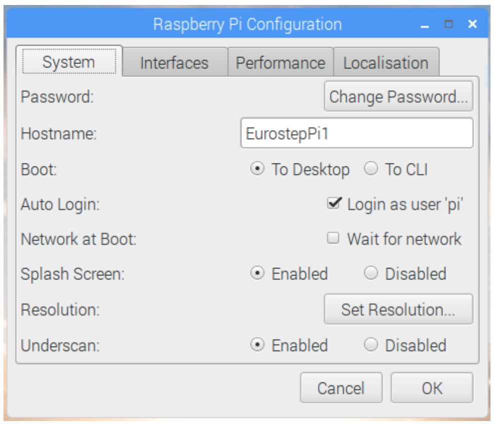


Figure 16 - Kitchen Sensor

4.2 Setting up the Raspberry Pi

The Raspberry Pi boards can run a selection of operating systems (OS), I decided to run a distribution of Linux called Raspbian (Raspbian, n.d.), mainly because I had used this OS before. There are, however, other distributions of Linux for the Pi such as Ubuntu and there is even a version of Windows 10 for the Pi. Raspbian comes with Node-JS and Node-RED so they do not need to be separately installed. The OS is installed to the Pi by writing an image file to a microSD which then plugs into the Pi. The Pi can then be booted by plugging a micro USB cable in to power-up the board. Once the OS boots, the Pi can be connected to the network and internet either over WIFI or via its Ethernet port.

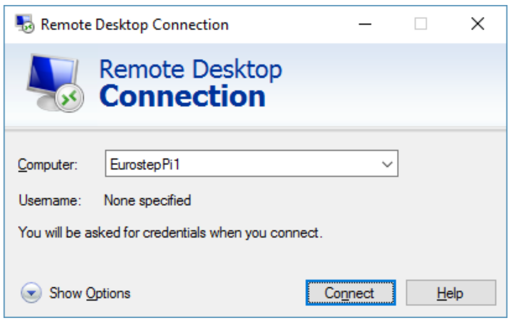
After this, I decided it would be useful to have remote access to the Pi boards over the network since doing this would allow the Pi boards to run in "headless" mode so you do not need a separate monitor, mouse and keyboard for each Pi in the system. To do this I first changed the host name of each Pi in settings as can be seen in Figure 17. I then went about installing a program called XRDP (xrdp, n.d.), XRDP allows access to the Pi using "windows remote desktop connection".



Password:	Change Password...
Hostname:	EurostepPi1
Boot:	[✓ To Desktop] To CLI
Auto Login:	[✓ Login as user 'pi']
Network at Boot:	Wait for network
Splash Screen:	[✓ Enabled] Disabled
Resolution:	Set Resolution...
Underscan:	[✓ Enabled] Disabled

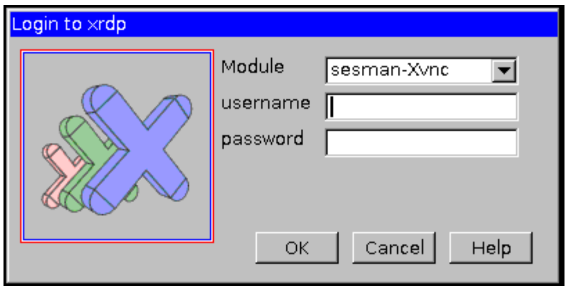
Figure 17 - Pi Settings

Once installed, the Pi can be accessed by entering the hostname of the Pi in “windows remote desktop connection” (as shown in Figure 18) on any windows computer on the same local network as the Pi. Doing this brings up a login prompt for the Pi being connected to as can been seen in Figure 19. After the username and password have been entered, the Pi can be controlled as though a mouse, keyboard and monitor were connected directly to it.



Computer: EurostepPi1
Username: None specified
You will be asked for credentials when you connect.

Figure 18 - Remote Desktop Connection



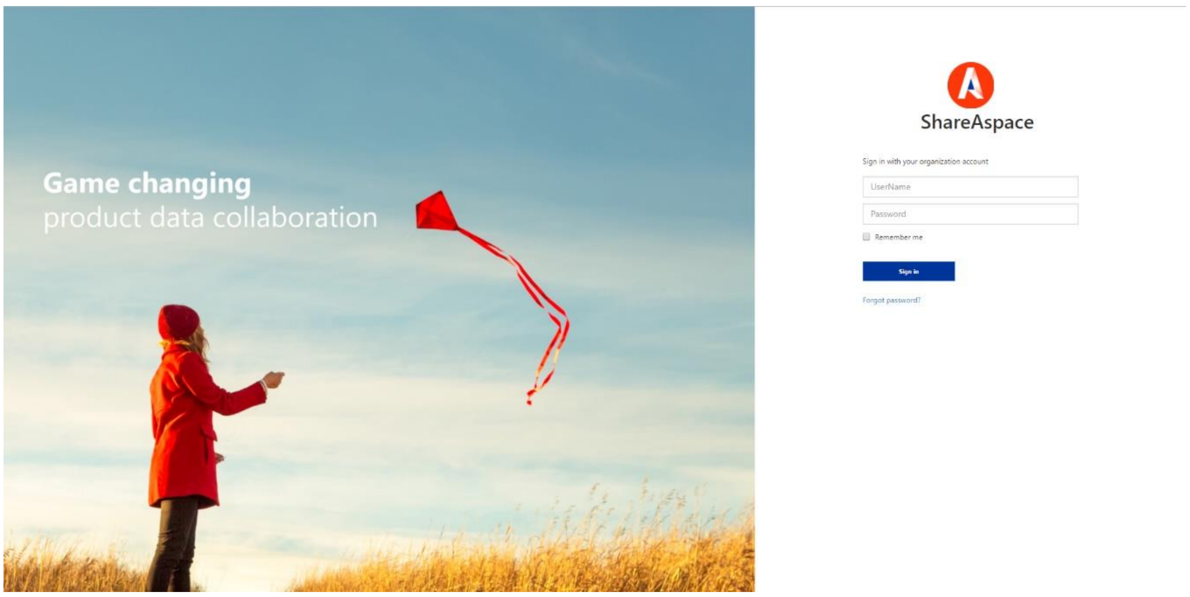
Module: sesman-Xvnc
username:
password:

Figure 19 - Login Prompt

4.3 Setting up ShareAspace

To install ShareAspace on my laptop I had help from a colleague at work who had installed ShareAspace many times before. ShareAspace does not have a proper installer yet so a lot of the process needs to be done manually. There is a guide explaining how to install it in ShareAspace documentation (Eurostep, n.d.).

After ShareAspace has been installed, it needs to be bootstrapped and hosted. Once this is done ShareAspace can be accessed from any computer on the local network by typing the hostname of the host computer followed by “/sasweb” in my case this was “max-hp/sasweb”. This brings up a login page as can be seen in Figure 20.



Sign in with your organization account
UserName
Password
Remember me

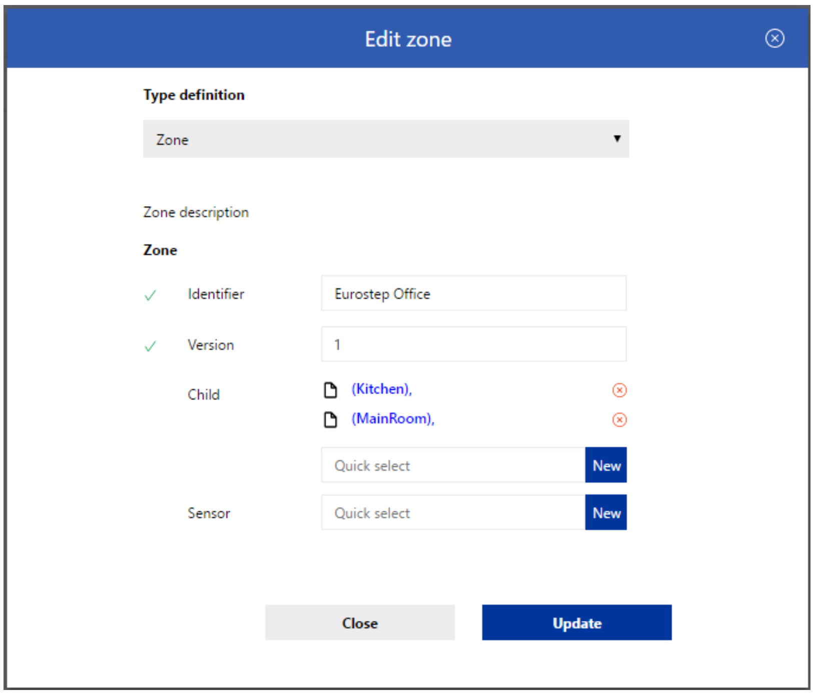
Figure 20 - ShareAspace Login Page

After logging in as *administrator* the next step is to upload a configuration or “template” for ShareAspace. This can be done by going to the “collectionAdmin” dropdown and selecting

“Administration”. Here the “templates” tab can be selected and from this tab one can upload a “config” file. I chose to upload the *config* file which was created using the data model I made earlier (discussed in the design chapter).

The configuration that was uploaded is then used to create a “space” in ShareAspace. A “space” is a virtual area of collaboration where “soft-types” can be created. *Soft-types* are custom buisness objects rather than business objects (like person, organization, part etc) which ShareAspace already “knows” about. I created a *space* called “SensorData” and within that *space* I created *soft-types* corresponding to the sensors and to the rooms.

First I created a *zone soft-type* to represent the whole Eurostep office as can be seen in Figure 21.


Edit zone

Type definition

Zone

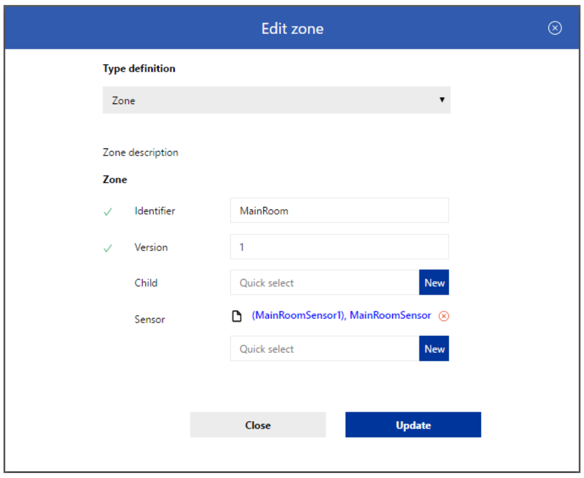
Zone description

Zone

✓	Identifier	Eurostep Office
✓	Version	1
	Child	(Kitchen), (MainRoom), Quick select
	Sensor	Quick select

Figure 21 - Zone Soft-Type

Within the Eurostep office *soft-type*, I created two child zone *soft-types*, the Kitchen and MainRoom, so two rooms within the office. The MainRoom and Kitchen *soft-types* each had a sensor within them as can be seen in Figure 22 and Figure 23.


Edit zone

Type definition

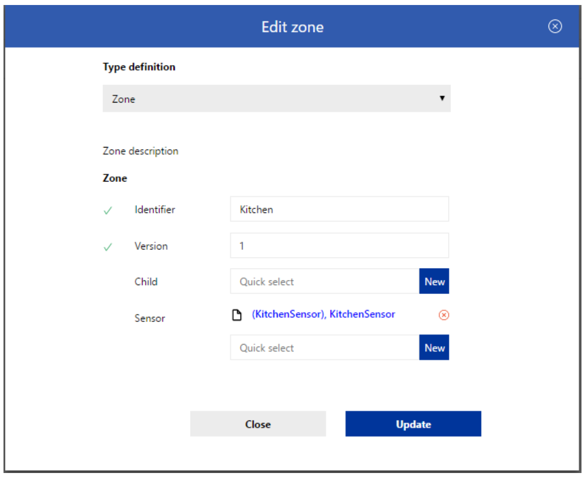
Zone

Zone description

Zone

✓	Identifier	MainRoom
✓	Version	1
	Child	Quick select
	Sensor	(MainRoomSensor1), MainRoomSensor, Quick select

Figure 22 - Main Room Soft-Type


Edit zone

Type definition

Zone

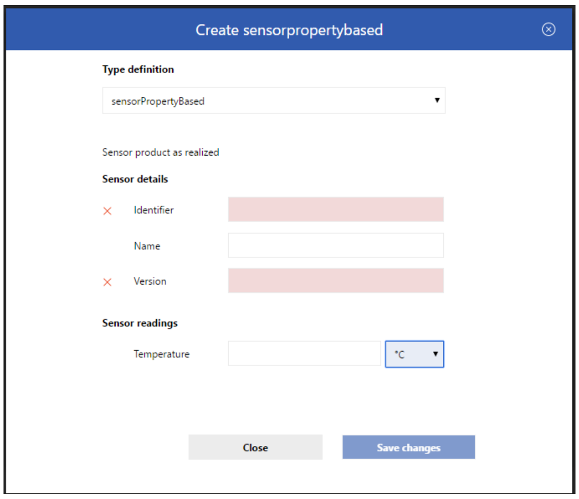
Zone description

Zone

✓	Identifier	Kitchen
✓	Version	1
	Child	Quick select
	Sensor	(KitchenSensor), KitchenSensor, Quick select

Figure 23 - Kitchen Soft-Type

Two types of sensor *soft-types* were created, each representing a different way of storing the sensor data in ShareAspace. The first and most simple sensor type was called “sensorpropertybased” this type of sensor *soft-type* just holds one temperature reading - the most recent one - and is updated every time a new temperature reading is taken. This *soft-type* can be seen in Figure 24.



Create sensorpropertybased

Type definition

sensorPropertyBased

Sensor product as realized

Sensor details

Identifier

Name

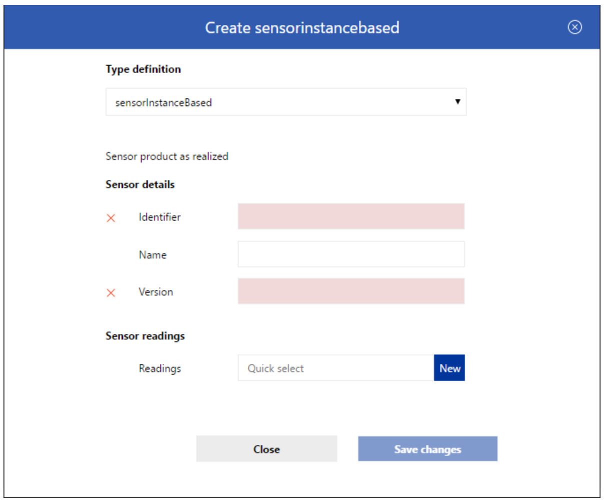
Version

Sensor readings

Temperature °C

Close | Save changes

Figure 24 - sensorpropertybased Soft-Type



Create sensorinstancebased

Type definition

sensorInstanceBased

Sensor product as realized

Sensor details

Identifier

Name

Version

Sensor readings

Readings Quick select New

Close | Save changes

Figure 25 - sensorinstancebased Soft-Type

The second type of sensor soft-type is called “sensorinstancebased”. This soft-type is different in that it holds multiple sensorReading soft-type instances. Each time a reading from a sensor arrives a new reading soft-type instance is created and attached to this sensor soft-type. The advantage of this type of sensor soft-type is that it holds a log of readings attached to the sensor they came from. This sensor soft-type can be seen in Figure 25.

In the deployed system, both approaches were tested. This concludes the description of the setup for my real-world test system in which the office and two rooms with sensors in them were represented in ShareAspace. The *config* that I used could however be used to create the entire office in ShareAspace with a *zone soft-type* instance for every room and there could be many sensors in each of these rooms.

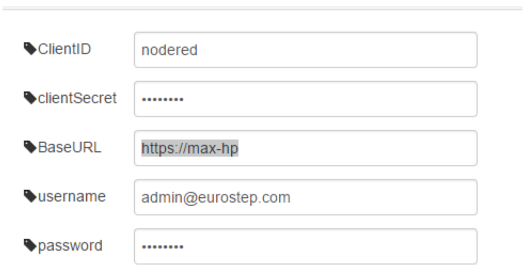
4.4 The ShareAspace library

As was noted above, the ShareAspace library I created was used in each of the flows for communication with ShareAspace. This entails sending HTTP requests from each Pi to the ShareAspace REST API. The library consists of four nodes each of them is described below:

4.4.1 Authenticator node

Figure 26 - Authenticator Node

The *authenticator* node is used for getting an access token from the ShareAspace server. This token is required for all communication with ShareAspace. This node has several parameters including: the ClientID, the clientSecret, BaseURL, username and password. When installing ShareAspace, “trusted” clients can be added. This is where the



ClientID	nodered
clientSecret	••••••••
BaseURL	https://max-hp
username	admin@eurostep.com
password	••••••••

Figure 27 - Authenticator Parameters

ClientID and clientSecret of a certain client are specified. In my case, for example, the ClientID was "nodered". The BaseURL was just the hostname of the ShareAspace server. For my laptop server, this was "https://max-hp". Users can also be added when setting up ShareAspace but I just used the administrative username and password. When the node is triggered, it sends an HTTP request containing everything to ShareAspace and, if everything is correct, ShareAspace returns the access token and the node outputs the returned access token.

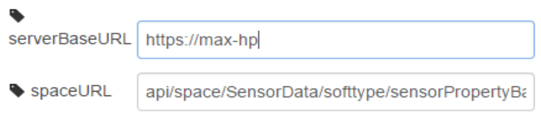
The simplified pseudocode behind the *authenticator* node is as follows:

```
var BaseURL = getBaseURL();
var authenticationURL = /AuthorizationServer/sasweb/oauth/token
var ClientID = getClientID();
var CLIENTSECRET = getCLIENTSECRET();
var username = getUserName();
var password = getPassword();

requestHeader = ClientID+":"+CLIENTSECRET;
requestHeader = requestHeader.toBase64();


HTTPRequest = {
      requestHeader
      username
      password

}

var response = HTTPRequest.post(BaseURL+authenticationURL);

node.output(response);
```

4.4.2 Create node

Figure 28 - Create Node

The *create* node is for creating new *soft-types* in a given *space* on ShareAspace. It takes the *serverBaseURL* and a *space* URL as its parameters. The *serverBaseURL* is the same as the *BaseURL* when using the *authenticator* node; so, in my case this was again "https://max-hp". The *space* URL is the location of the *space* in which the *soft-type* is created. The *create* node also takes a JSON as an input; this JSON is a representation of a *soft-type* the user wants to create. Once the node is triggered, it sends an HTTP request containing the JSON to ShareAspace. The node then outputs the response from ShareAspace. The request sent to ShareAspace also contains the access token retrieved by the *authenticator* node in its header so that ShareAspace "knows" which client is making the request.



serverBaseURL: https://max-hp

spaceURL: api/space/SensorData/softtype/sensorPropertyBa

Figure 29 - Create Parameters

The simplified pseudocode behind the *create* node is as follows:

```
var BaseURL = getBaseURL();
var spaceURL = getSpaceURL();
var softTypeJSON = getNodeInput();
requestHeader = getAccessToken();


HTTPRequest = {
      requestHeader
      softTypeJSON

}

var response = HTTPRequest.post(spaceURL);

node.output(response);
```

4.4.3 Read node

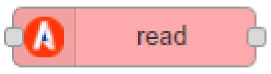


Figure 30 - Read Node

The *read* node is for retrieving existing *soft-types* from ShareAspace and returning them to the Pi in JSON format. The only parameter the node takes is again the *BaseURL* of the server. The location of the *soft-type* we want to read is taken as an input to the node as a string. The location is given in the form of a URL. An example might be as follows:

"/api/space/SensorData/softtype/sensorPropertyBased/defaultOut/17f11e594c4c94552d01020006000000?_=1495207277135"

When the *read* node detects an input, it sends an HTTP GET request to the provided location URL. If everything is correct, ShareAspace returns the requested *soft-type* as a JSON. The node then outputs the received JSON.

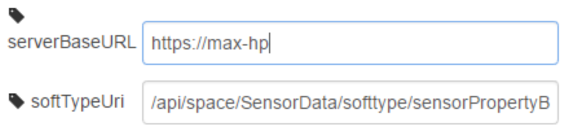
The simplified pseudocode behind the *read* node is as follows:

```
var BaseURL = getBaseURL();
var softTypeLocation = getNodeInput();
requestHeader = getAccessToken();

HTTPRequest = {
      requestHeader

}

var response = HTTPRequest.get(softTypeLocation);

node.output(response);
```

4.4.4 Update node

Figure 31 - Update Node

The "update" node is for editing existing *soft-types* in ShareAspace. This node takes two parameters, the *serverBaseURL* and a *soft-type* URL. The BaseURL is, again, the same as in every other ShareAspace node and the *soft-type* URL is the location of the specific *soft-type*



serverBaseURL: https://max-hp
softTypeUri: /api/space/SensorData/softtype/sensorPropertyB

Figure 32 - Update Parameters

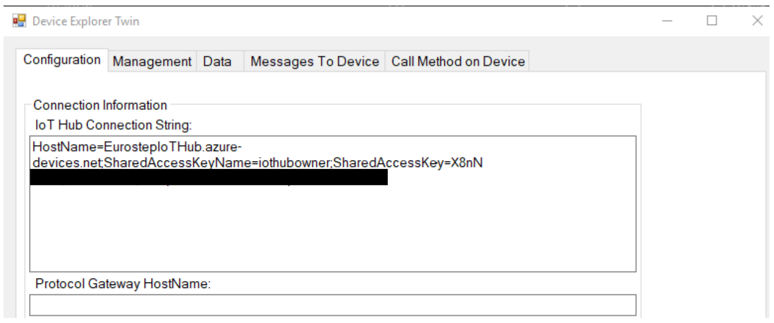
we wish to edit. The *update* node takes an edited JSON of the *soft-type* we want to edit as its input. For example, the new version of the *soft-type* could have a new temperature reading stored in it. This edited JSON is then sent to ShareAspace using an HTTP PUT request and, if all is correct, ShareAspace replaces the old *soft-type* with the new edited one. The node then outputs the response from ShareAspace.

The simplified pseudocode behind is the update node is as follows:

```
var BaseURL = getBaseURL();
var softTypeLocation = getSoftTypeLocation();
requestHeader = getAccessToken();
var editedSoftTypeJSON = getNodeInput();


HTTPRequest = {
      requestHeader
      editedSoftTypeJSON
}

var response = HTTPRequest.put(softTypeLocation);

node.output(response);
```

4.5 Setting up Azure and powerBI

Azure is Microsoft's cloud computing service. Azure offers many different services, however I only used two for this project. The first service I used was the "Azure IoT hub" and the second service used was the "Stream Analytics Job" (described later). The IoT hub "Provides multiple device-to-cloud and cloud-to-device communication options, including one-way messaging, file transfer, and request-reply methods." (Microsoft, 2017). This means the IoT hub can be used to receive messages from a Raspberry Pi. Creating an IoT hub can be done from the Azure dashboard by going to new -> Internet of things -> IoT hub. Once this was done, Azure needed to be told about the devices it would be receiving messages from as the IoT hub only allows connections from known devices. To do this I used a piece of software called the Azure IoT Hub DeviceExplorer (Barry, 2015). To connect the Device Explorer to the IoT, a connection string is required. This connection string can be found under "Shared access policies" in the IoT hub. The connection string can then be copied to the device manager to connect it to the IoT hub as shown in Figure 33.

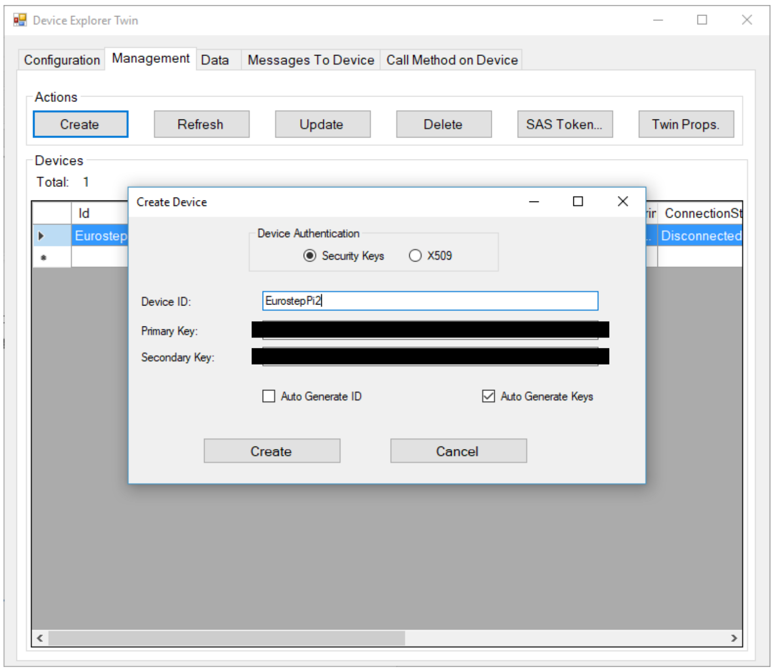


IoT Hub Connection String:
HostName=EurosteploTHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=X8nN [redacted]

Protocol Gateway HostName:

Figure 33 - The Connection String

After this, the two Pi boards, can be added to the IoT hub as "trusted devices". This is done by going to the "management" tab on the device explorer and then clicking "create". The screenshot in Figure 34 shows the two generated keys. These keys were used later when the Pi board in the main room, referred to as "EurostepPi2", was connected to the IoT hub.



Device Authentication: [✓ Security Keys] X509

Device ID: EurostepPi2

Primary Key: [redacted]

Secondary Key: [redacted]

☐ Auto Generate ID ☑ Auto Generate Keys

Figure 34 - Device Explorer EurostepPi2 Keys

The next step was to create a PowerBI account. PowerBI (Microsoft, n.d.) is a Microsoft business analytics service. It can be used to create reports containing interactive visualizations such as graphs. This made it ideal for this project as it could be used to visualize temperature data over time on a graph. PowerBI did not really need setting up, I just created an account and this gave me access to the PowerBI portal where the data from IoT hub could be sent.

Finally, I needed to setup the second service. Microsoft offer a “Stream Analytics Job” (Microsoft, n.d.) which transfers data from an input source to an output sink. In my case, I was basically able to use the Stream Analytics Job as a pipeline to move data received by the IoT hub into PowerBI. To create a Stream Analytics Job, you go to new -> Data + Analytics -> Stream Analytics Job. Once the Job is created, an “input”, “output” and “query” need to be set up. For the *input*, I selected the IoT hub I created earlier as this is where the messages from the Pi boards arrive. For the *output*, I selected my PowerBI account as this is where I wanted the data to end up. Finally, the *query* needed to be written. This was done in SQL. In my case the SQL was:

```
SELECT
    *
INTO
    [TemperatureBI]
FROM
    [INPUT]
```

This selects all data from the input (the IoT hub) and forwards it to the output (PowerBI). After this step Azure was ready to receive data from the Pi boards.

4.6 The Node-RED flows

The final step in setting up the system was to create the flows in Node-Red for sending the data to the different sensor *soft-types* in ShareAspace and sending data to Azure.

4.6.1 ShareAspace sensorpropertybased flows

The flows for the sensorpropertybased *soft-type* consisted of flows for authenticating, flows for retrieving the two-different sensor *soft-types* (kitchen and main room) and flows for updating the sensor *soft-types* on ShareAspace. In the case of my system I had one sensor per Pi so the flows for the kitchen *soft-type* were on the Pi board labelled as “EurostepPi1” and the flows for the main room *soft-type* were on the Pi board referred to as “EurostepPi2”. Figure 35 shows the flows for EurostepPi1:

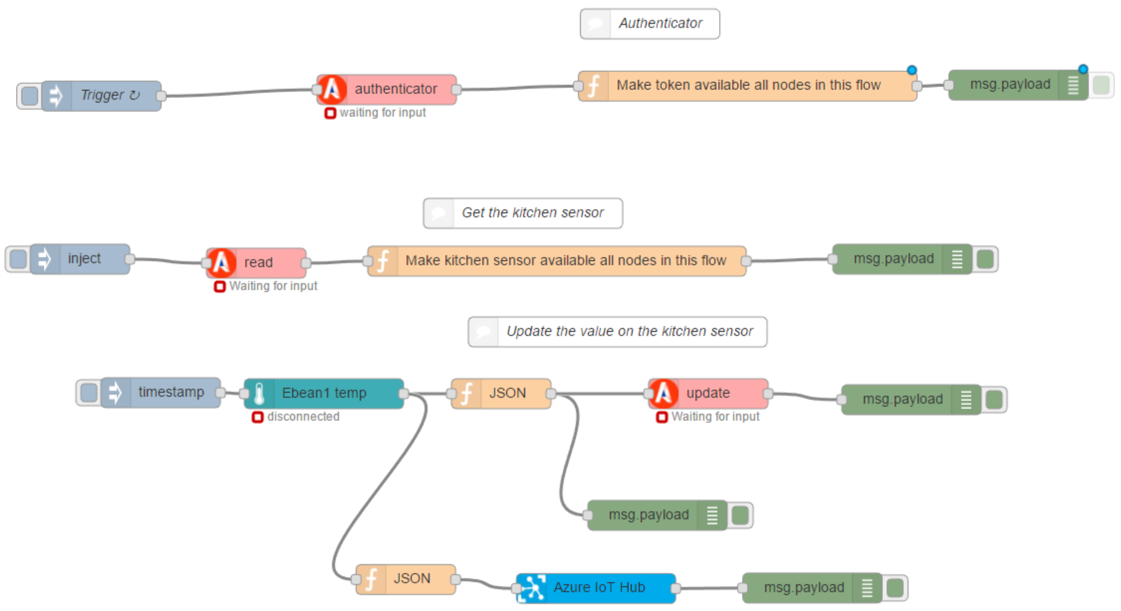
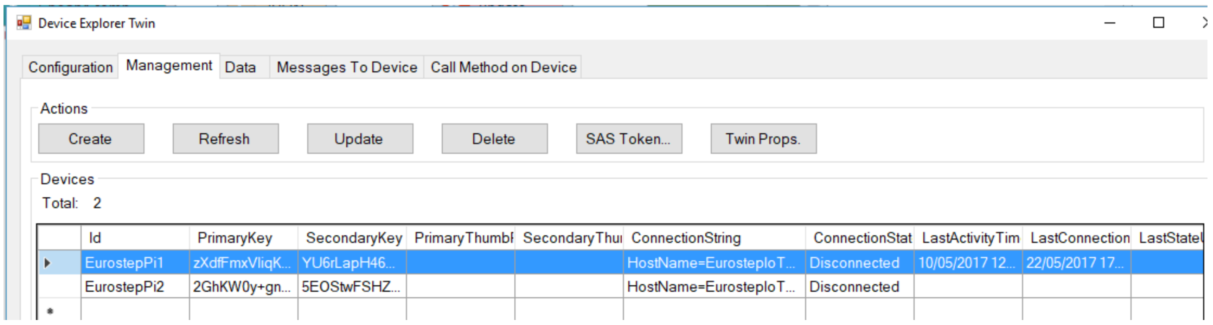


Figure 35 - sensorpropertybased Flows

The first flow (called “*authenticator*”) is responsible for retrieving the ShareAspace access token and making it available to other flows that use the ShareAspace library. This is so that the Pi can authenticate with ShareAspace. The second flow (referred to as “Get the kitchen sensor”) is responsible for retrieving the kitchen sensor *soft-type* so that the latest temperature can be added to it by the third flow. Finally, the third flow (labelled as “Update the value on the kitchen sensor”) is responsible for sending the edited *soft-type* with the latest temperature to ShareAspace to replace the old *soft-type*. The third flow is also responsible for sending the latest temperature to the Azure IoT hub. The IoT hub node takes a connection string from the devices created earlier in Azure (covered in Setting up Azure and powerBI). This is so that Azure “knows” which temperature reading is coming from which Pi. This connection string can be found in the device manager used to create the devices on Azure as seen in Figure 36. As this was for the kitchen sensor, EurostepPi1’s connection string was used.



Device Explorer Twin

Configuration | Management | Data | Messages To Device | Call Method on Device

Actions

Create | Refresh | Update | Delete | SAS Token... | Twin Props.

Devices

Total: 2

Id	PrimaryKey	SecondaryKey	PrimaryThumbl	SecondaryThu	ConnectionString	ConnectionStat	LastActivityTim	LastConnection	LastStateU
EurostepPi1	zXdfFmxVliqK...	YU6rLapH46...			HostName=EurosteploT...	Disconnected	10/05/2017 12...	22/05/2017 17...	
EurostepPi2	2GhKW0y+gn...	5EOStwFSHZ...			HostName=EurosteploT...	Disconnected			

Figure 36 - Devices in Azure

The flows for the main room are identical to the ones for the kitchen apart from using the other sensor "Ebean2" and the IoT hub node using the connection string for the other Pi (EurostepPi2). A detailed walkthrough of the flows used can be found in chapter 5.

4.6.2 ShareAspace sensorinstancebased flow

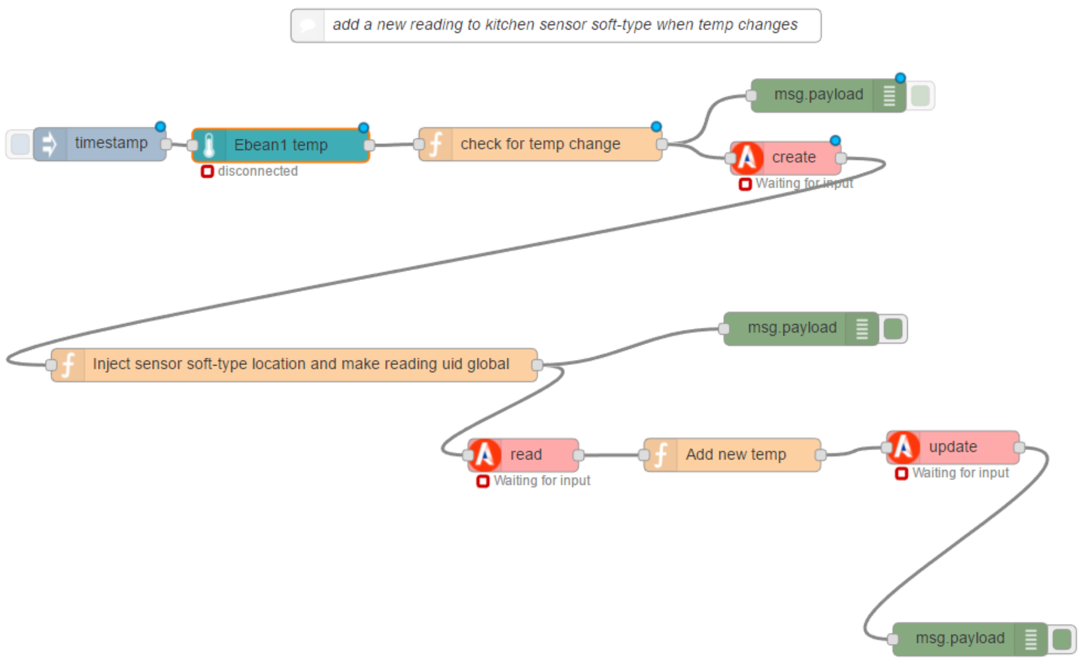


Figure 37 - sensorinstancebased Flow

The sensorinstancebased kitchen flow checks for temperature changes. If a change is detected, a new temperature reading *soft-type* is created. The temperature reading *soft-type* is then attached to the sensorinstancebased kitchen sensor using the *read* node to get the sensorinstancebased kitchen sensor and then the *update* node is used to send a new version of the sensorinstancebased kitchen sensor containing the new reading. This flow can be seen in Figure 37. Again, the flow for the sensorinstancebased main room sensor is identical apart from the fact that it gets its temperature from "Ebean2". A detailed walkthrough of this flow can also be found in chapter 5.

5 The system in operation

5.1 Authenication operation

After setting up everything in the Eurostep office, the Raspberry Pi boards and sensors were powered on and the system was ready to be used. First, Node-Red needed to be opened on each Pi and the frequency of the trigger nodes needed to be set. I set the *authenticator* flow to trigger every hour. The reason for this is the access token expires an hour after it is first created so every hour a new token is needed. After this I pressed the button on the trigger node to get the first token.

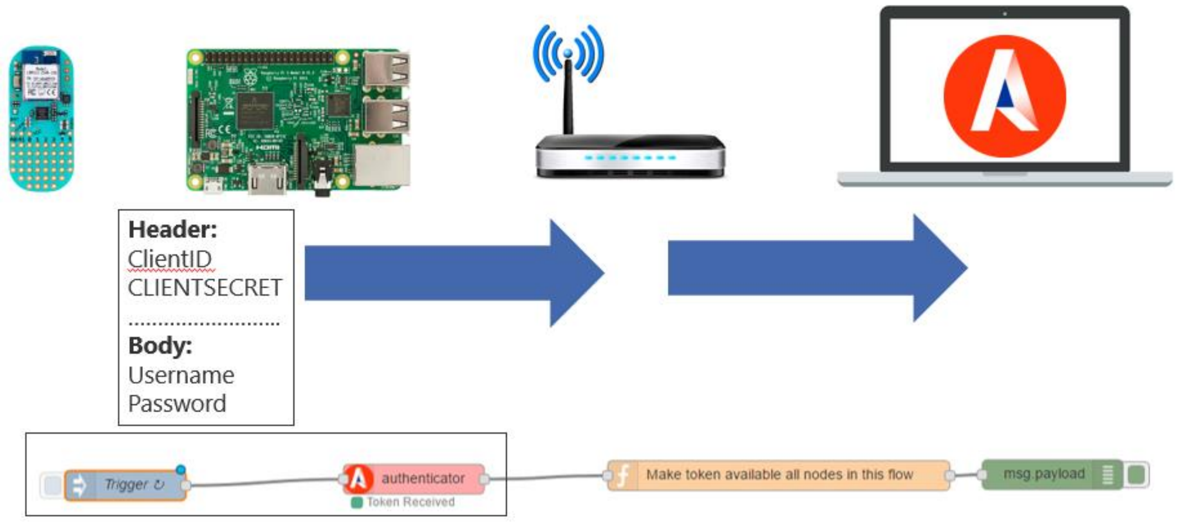


Figure 38 - Authentication Flow Sending

As shown in Figure 38, the *authenticator* node sends an HTTP POST request to ShareAspace's authentication server. The request header contains the ClientID and CLIENTSECRET while the request body contains the username and password. If these details are correct ShareAspace responds with an access token as shown in Figure 39.

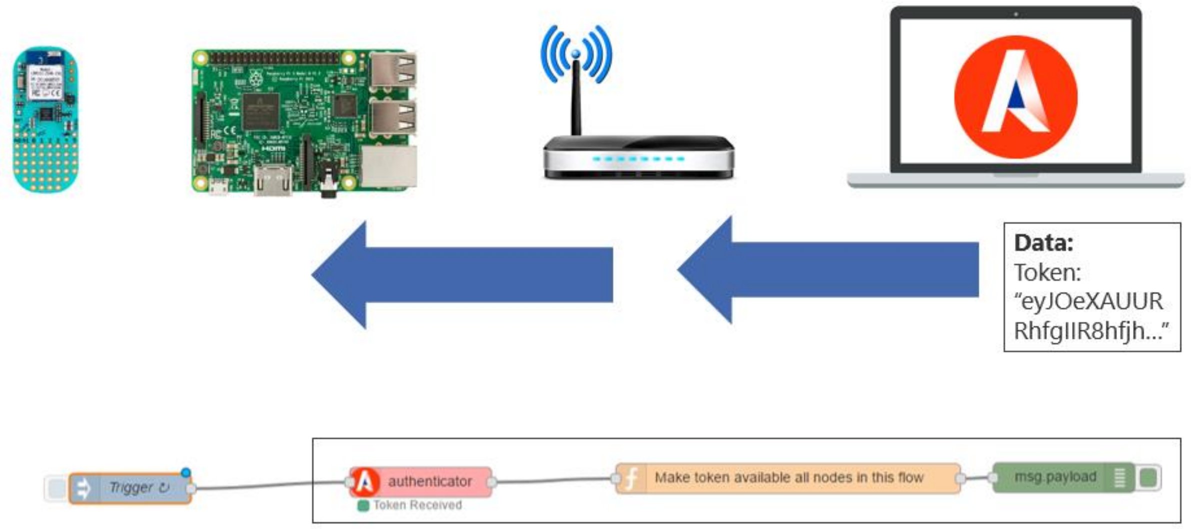
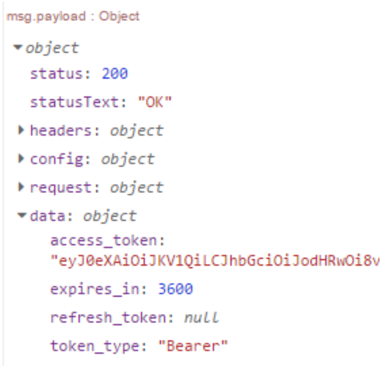


Figure 39 - Authentication Flow Receiving

After the *authenticator* node receives the token from ShareAspace, the *authenticator* sends it on to a "function" node. All the *function* node does is make the token accesible to other flows using ShareAspace library nodes. Finally, the debug node also prints the response containing the token to the console as can be seen in Figure 40. This whole process needed to be carried out on both Pi boards.



```
msg.payload : Object
▾object
  status: 200
  statusText: "OK"
 ▸headers: object
 ▸config: object
 ▸request: object
 ▾data: object
    access_token:
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJodHRwOi8v
    expires_in: 3600
    refresh_token: null
    token_type: "Bearer"
```

Figure 40 - Response

5.2 sensorpropertybased and Azure operations

The next step was to get the sensorpropertybased kitchen sensor *soft-type* so that the next flow could use it to update the temperature. This *soft-type* only needed to be retrieved once so the flow only needed to be run once.

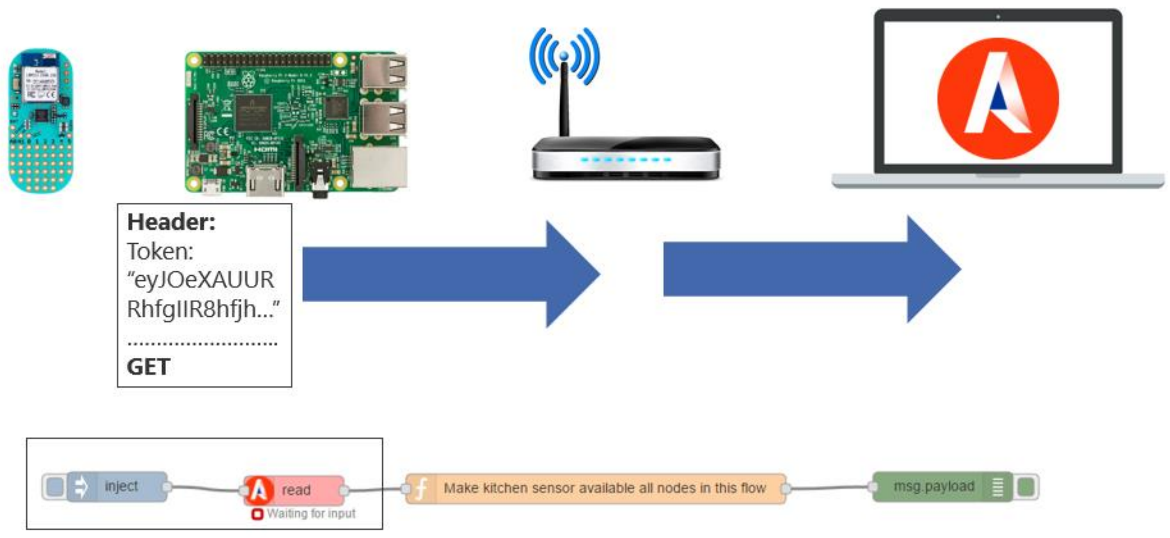


Figure 41 - Requesting Soft-Type

To run the flow, the button on the *inject* node is pressed. Doing this inputs the URL of the kitchen sensorpropertybased kitchen sensor *soft-type* into the *read* node. The *read* node then sends an HTTP GET request to the inputted URL, the request header also contains the token in order to authenticate with ShareAspace.

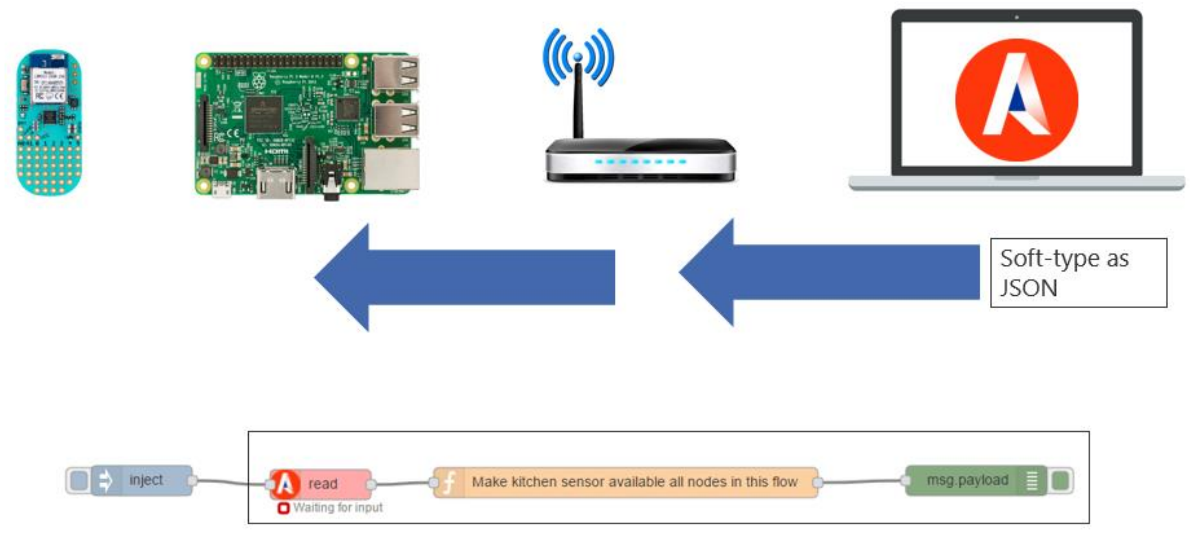


Figure 42 - Receiving Soft-Type

ShareAspace then returns the requested kitchen *soft-type* as a JSON. This JSON gets forwarded to a *function* node which makes the kitchen sensor JSON accessible to other flows. Again this process had to be repeated for the other Pi board, which would be returning the main room *soft-type*.

The next step was to start the flow that takes temperature readings, updates the sensorpropertybased kitchen sensor on ShareAspace and sends the reading to Azure. I decided to set the timestamp node to trigger the Light Blue Bean node every second. This means a temperature reading was taken every second. Although this drained the sensor battery more quickly it allowed me "fill up" ShareAspace more quickly to test how many *soft-types* ShareAspace could cope with. In other real-world senarios readings would probably be taken less frquesntly depending on the companies needs.

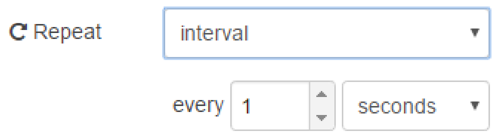

Repeat interval
every 1 seconds

Figure 43 - Trigger Every Second

Figure 44, Figure 45 and Figure 46 all show the ShareAspace side of this flow.

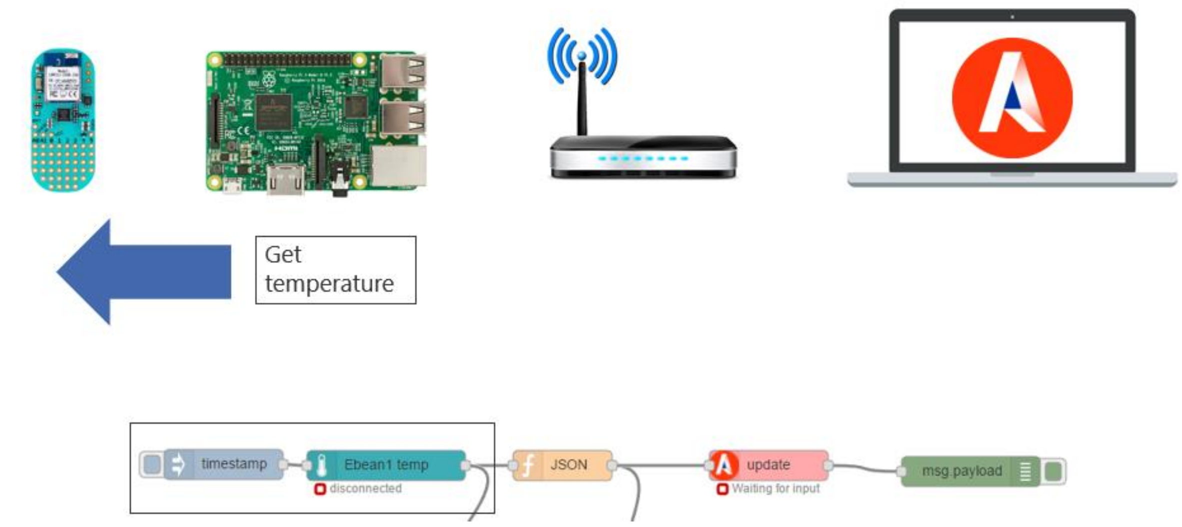


Figure 44 - Requesting the Temperature

Here the Pi connects (over bluetooth) to the Light Blue Bean and asks for a temperature reading.

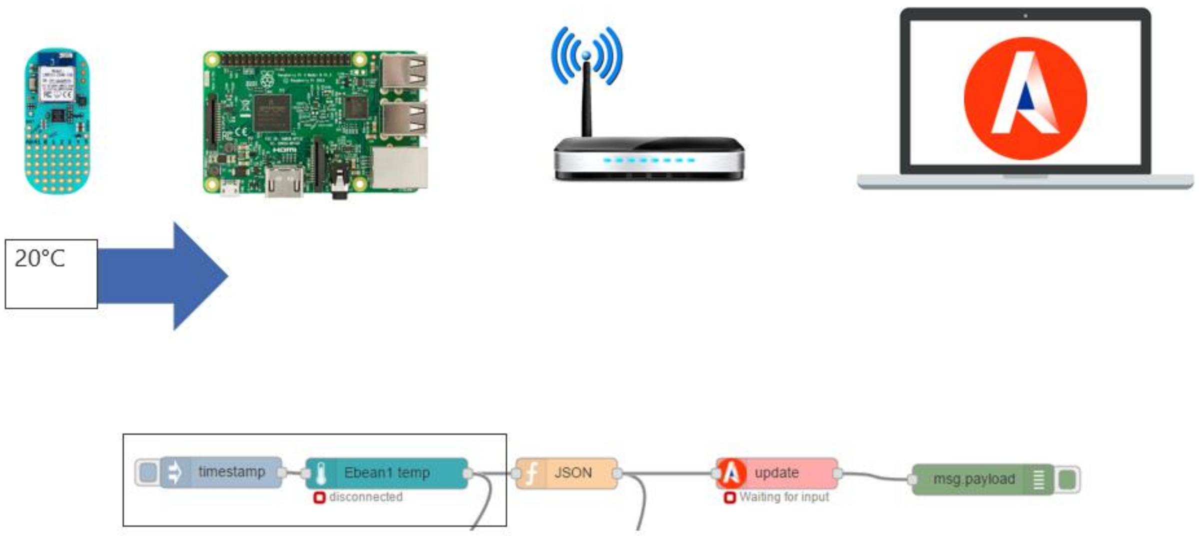


Figure 45 - Returning the Temperature

The Light Blue Bean then replies with a temperature and forwards it to the JSON node.

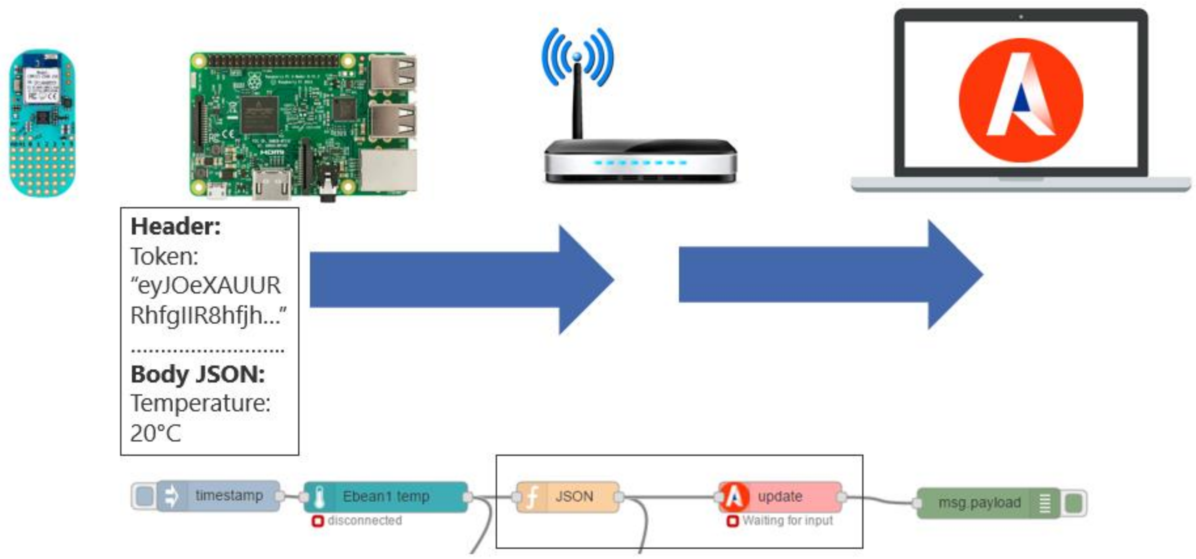
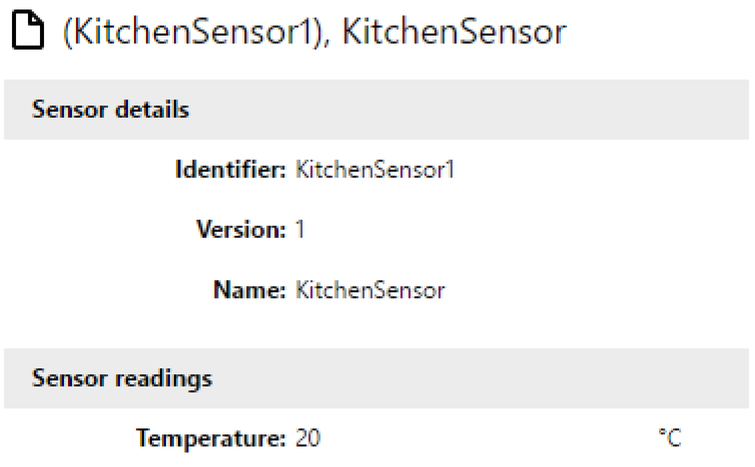


Figure 46 - Making the Update on ShareAspace

The JSON node (*function* node) first of all retrieves the sensorpropertybased kitchen sensor *soft-type* which the previous flow retrieved from ShareAspace. The JSON node then changes the *soft-type* JSONs temperature to whatever temperature was received from the Light Blue Bean node. This new *soft-type* JSON is then sent on to the *update* node. The *update* node forms an HTTP PUT request with the access token in the header and the JSON as the body of the request. This PUT request is then sent to the URL of the sensorpropertybased kitchen sensor *soft-type* originally retrieved. This whole process is triggered every second. I checked the *soft-type* in ShareAspace and found that it was successfully updating the temperature, as can be seen in Figure 47.



(KitchenSensor1), KitchenSensor

Sensor details

Identifier: KitchenSensor1

Version: 1

Name: KitchenSensor

Sensor readings

Temperature: 20 °C

Figure 47 - the Kitchen Sensor

This whole process was done again for the other Pi taking temperature readings from the main room instead of the kitchen.

Every time a temperature is added to the sensorpropertybased kitchen sensor and sent to ShareAspace, a JSON containing the temperature is also sent to the Azure IoT hub I created. This can be seen in Figure 48, where "temp" is the temperature reading from the Light Blue Bean.

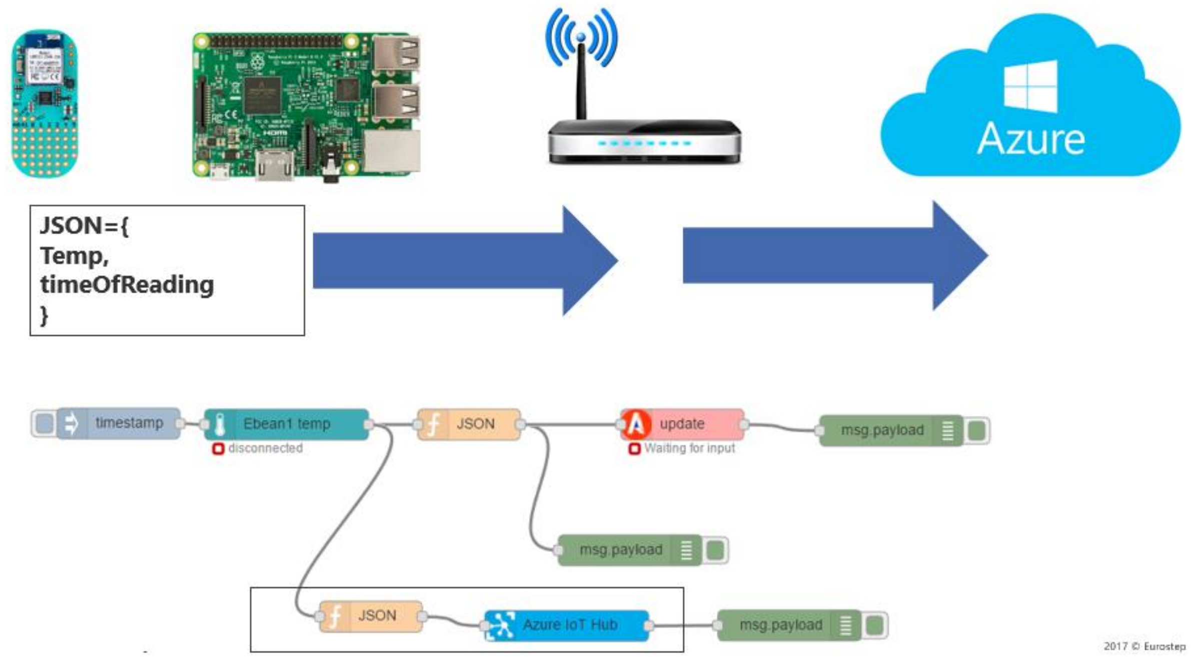
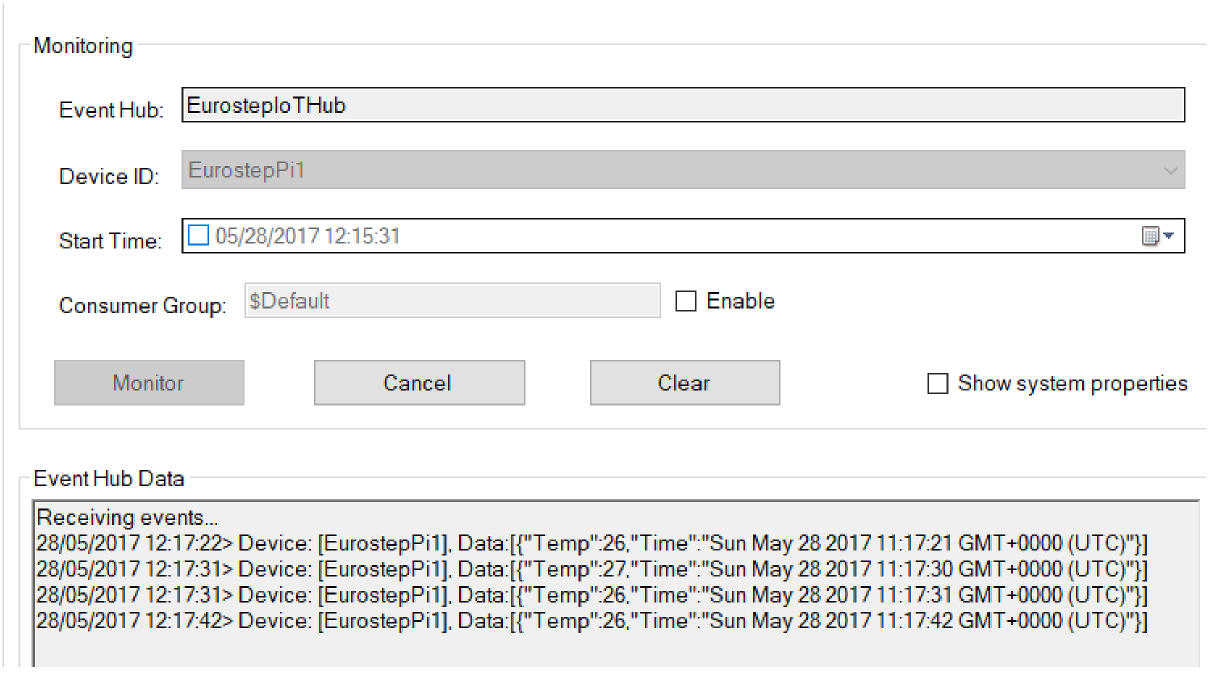


Figure 48 - Sending the Temperature to Azure

The messages (containing the temperature and time) sent to the IoT hub on Azure from the Pi boards can be seen on the device explorer program. As can be seen in Figure 49 for EurostepPi1.


Monitoring

Event Hub: EurosteploTHub

Device ID: EurostepPi1

Start Time: 05/28/2017 12:15:31

Consumer Group: $Default ☐ Enable

Event Hub Data

```
Receiving events...
28/05/2017 12:17:22> Device: [EurostepPi1], Data:[{"Temp":26,"Time":"Sun May 28 2017 11:17:21 GMT+0000 (UTC)"}]
28/05/2017 12:17:31> Device: [EurostepPi1], Data:[{"Temp":27,"Time":"Sun May 28 2017 11:17:30 GMT+0000 (UTC)"}]
28/05/2017 12:17:31> Device: [EurostepPi1], Data:[{"Temp":26,"Time":"Sun May 28 2017 11:17:31 GMT+0000 (UTC)"}]
28/05/2017 12:17:42> Device: [EurostepPi1], Data:[{"Temp":26,"Time":"Sun May 28 2017 11:17:42 GMT+0000 (UTC)"}]
```

Figure 49 - Monitoring

The IoT hub which can be accessed from the Azure portal, also shows how many messages have arrived (Figure 50). This was how I checked the messages were definitely arriving.

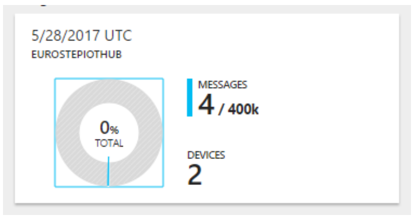


Figure 50 - Counter

The next step was to see if the messages were getting forwarded to powerBI from the IoT hub. To do this I logged into my PowerBI account and went on to the "datasets" tab. There I found a dataset called "TemperatureDataSet" made up of the messages received from the IoT hub. As shown in Figure 51.

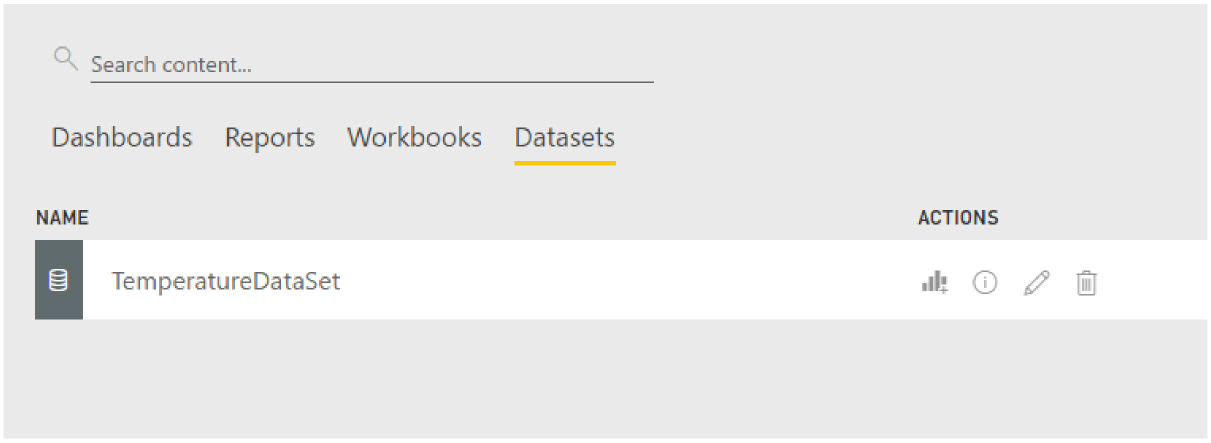


Figure 51 - Datasets

This dataset is continuously updated as new temperature readings from the Pi board arrive. After leaving the system running for a while I then used PowerBI to generate a "report" from the dataset. This is done by clicking the create report button as shown in Figure 52.

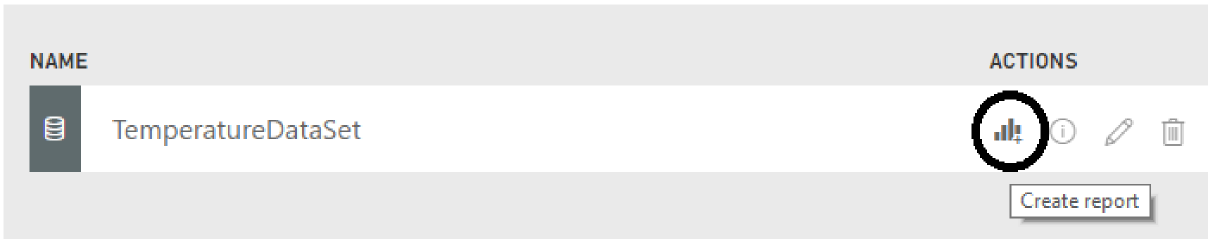


Figure 52 - Create Button

Doing this brings up a view where parts of the dataset can be selected for "visualization", as can be seen in Figure 53.

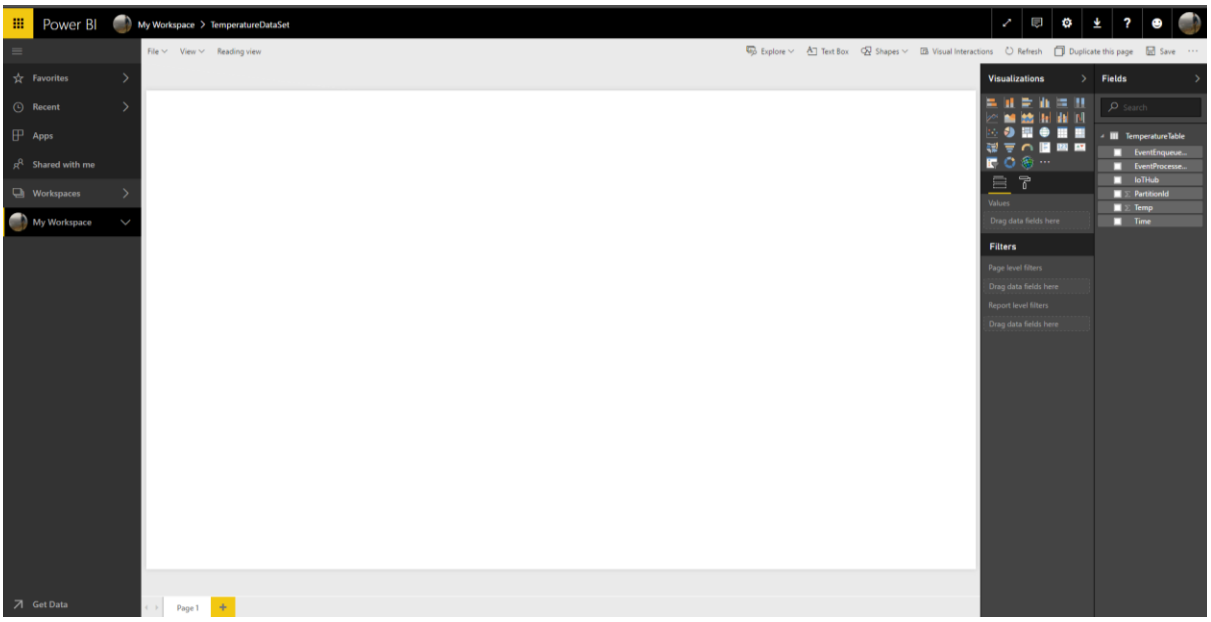


Figure 53 - Visualization

I wanted to create a graph showing how temperature varies over time. So, from here I selected "Temp" and "Time" and the line graph visualization button. Doing this generated an interactive line graph of temperature against time. This generated report graph can be seen in Figure 54.

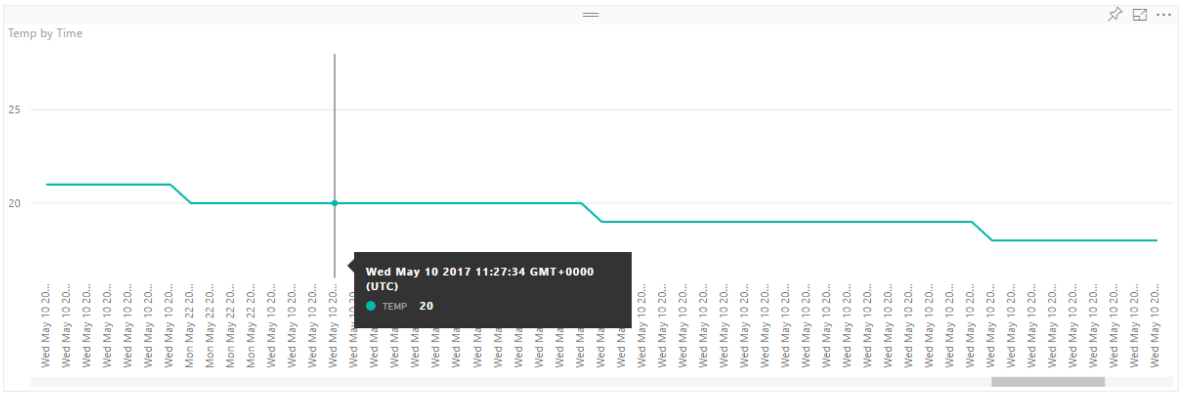


Figure 54 - Temp vs Time Graph

5.3 Sensorinstancebased operations

Collecting the temperature from the Light Blue Bean for the Sensorinstancebased sensor *soft-type* is the same process as for the sensorpropertybased sensor *soft-type*. I also set the flow to run every second (the same as for sensorpropertybased). The first difference is that, before the request is created, a *function* node checks if the temperature is the same as the last readings temperature or if it has changed. If the temperature is the same, nothing else happens and the process stops there. However, if the temperature has changed then a JSON representation of a reading *soft-type* is added to an HTTP POST request, which is then sent to a space's URL on ShareAspace. This flow can be seen in Figure 55.

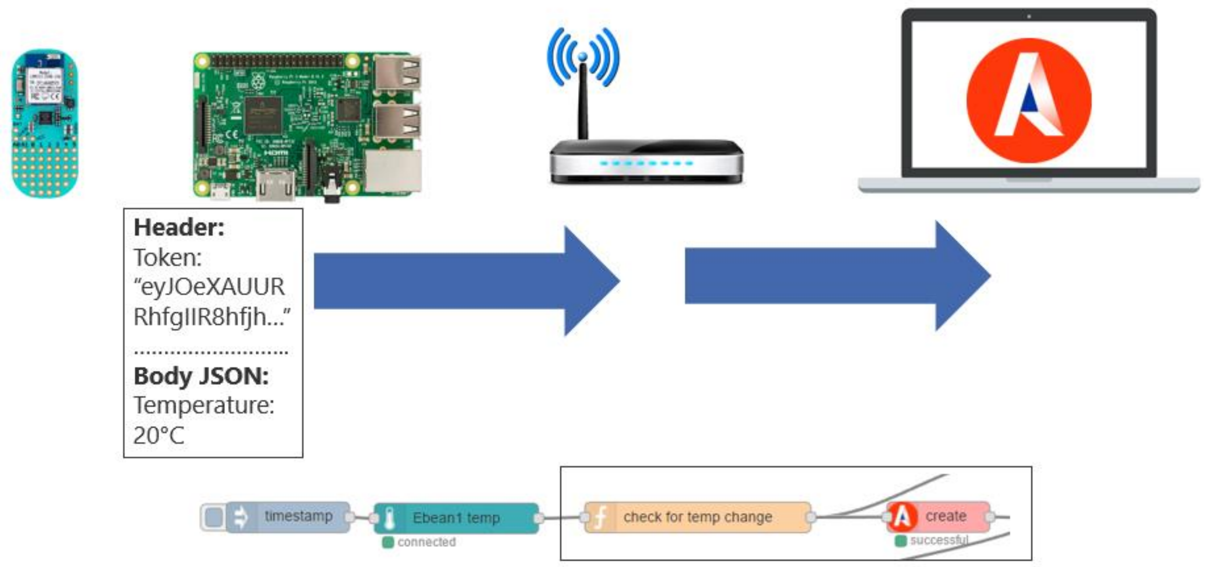
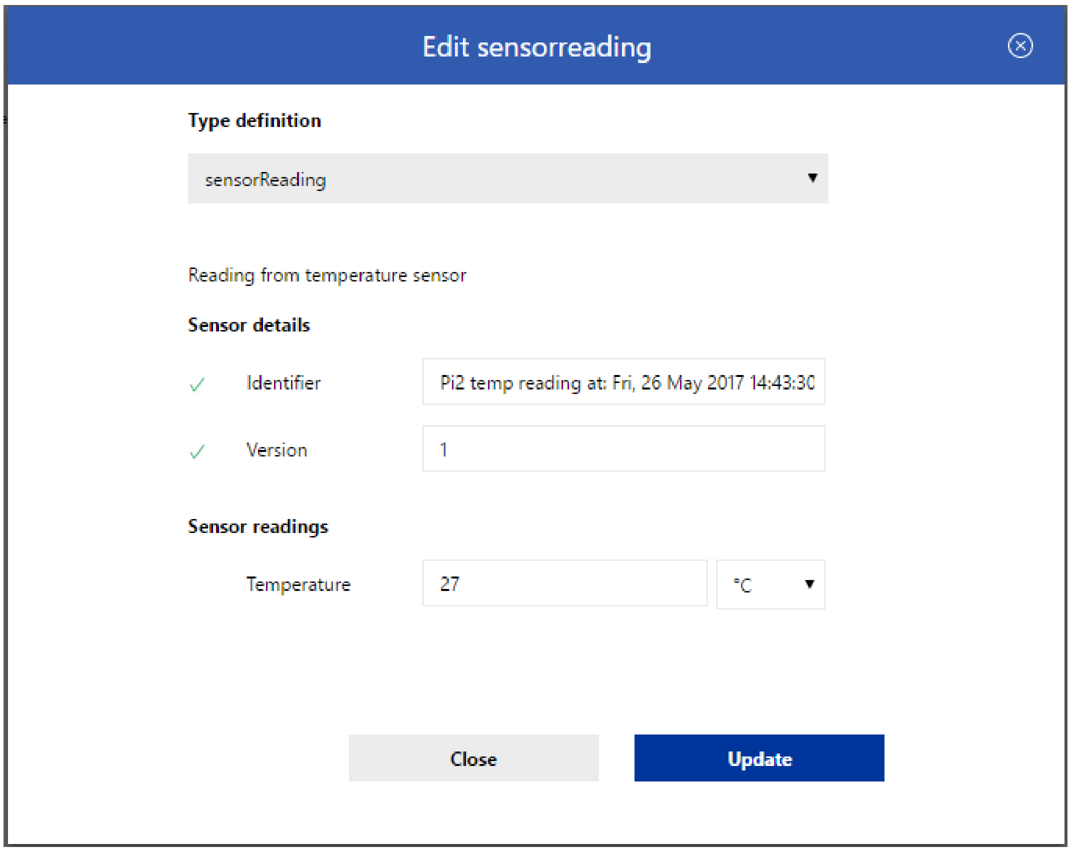


Figure 55 - Creating a Temperature Reading Soft-Type

Doing this, creates a new reading *soft-type* containing the temperature sent such a *soft-type* can be seen in Figure 56.


Edit sensorreading

Type definition

sensorReading

Reading from temperature sensor

Sensor details

✓ Identifier	Pi2 temp reading at: Fri, 26 May 2017 14:43:30
✓ Version	1

Sensor readings

Temperature	27	°C

Figure 56 - Reading Soft-Type

The next step is to attach this sensor reading *soft-type* to the Sensorinstancebased sensor *soft-type* that was created in ShareAspace earlier. To do this a *function* node first stores the ID (which is received from the *create* node) of the reading *soft-type* and then injects the location (URL) of the

Sensorinstancebased sensor *soft-type* into a *read* node. The *read* node then makes a get request using that injected URL. This can be seen in Figure 57.

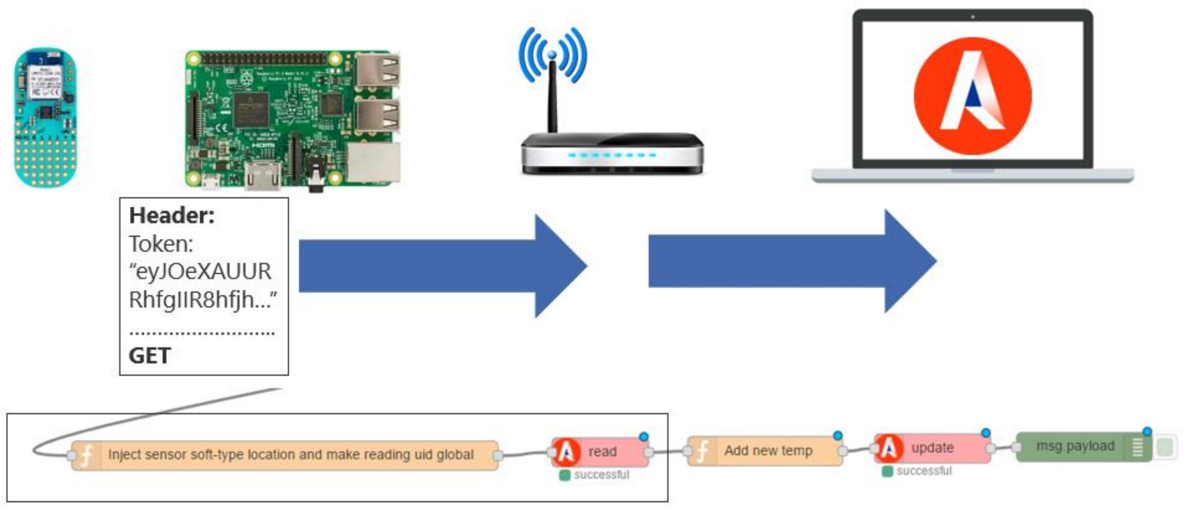


Figure 57 - Requesting the Sensor Soft-Type

ShareAspace then returns the Sensorinstancebased sensor *soft-type*, as seen in Figure 58.

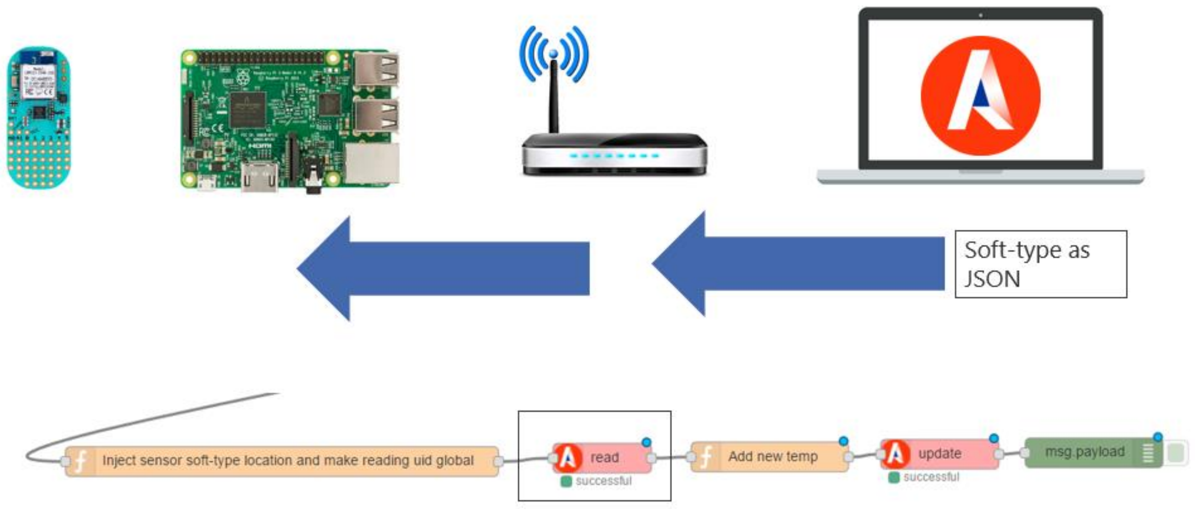


Figure 58 - Receiving Sensor Soft-Type

After the Sensorinstancebased sensor *soft-type* is received by the Pi, the *read* node passes it onto a *function* node. The *function* node then retrieves the ID of the reading *soft-type* and adds it to the Sensorinstancebased sensor *soft-type*. This updated Sensorinstancebased sensor *soft-type* is then sent back to replace the old version of itself. This is done using the update node as shown in Figure 59.

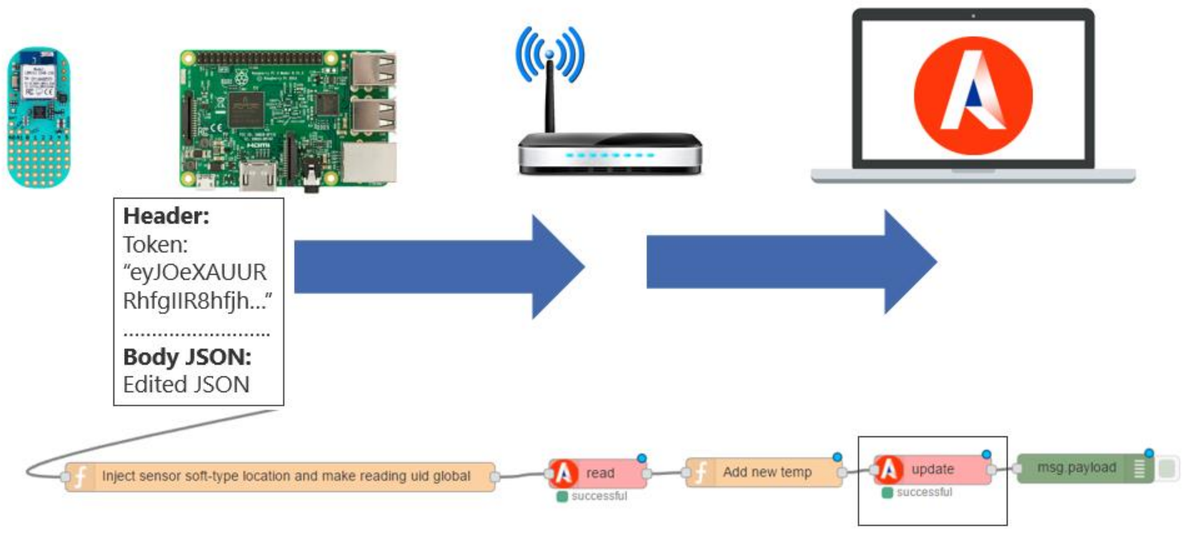
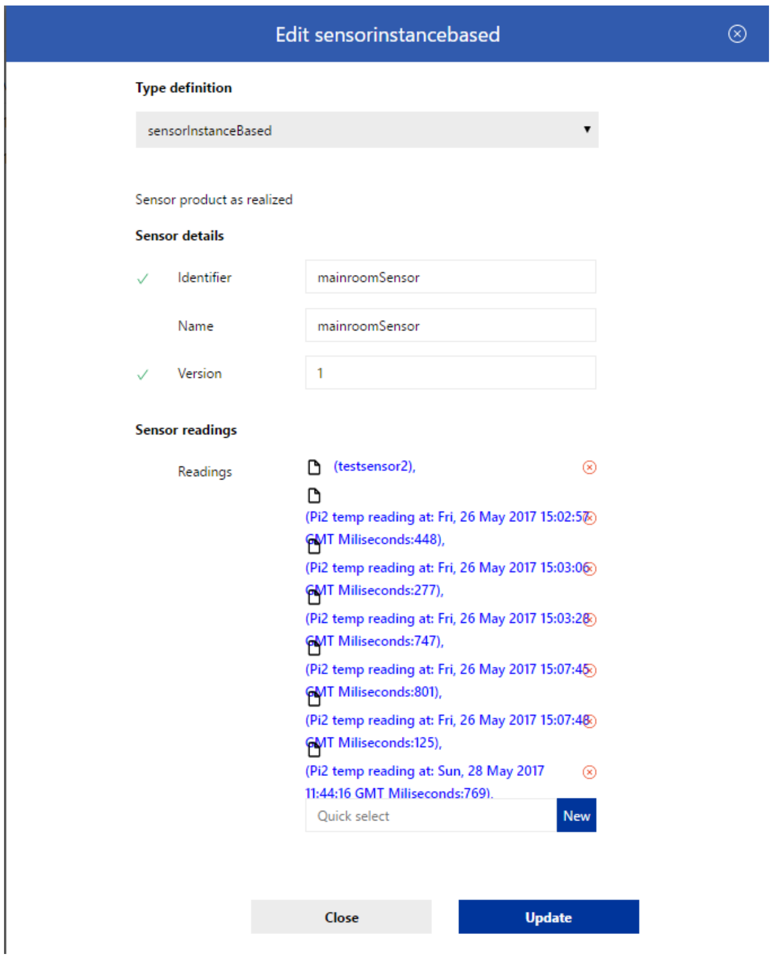


Figure 59 - Updating Sensor Soft-Type

After the system had been running for a while, I went to check the Sensorinstancebased sensor *soft-type* to see if readings were being added to the *soft-type* when the temperature changed. After opening the *soft-type* and checking the readings attached to it, I concluded that everything was working properly. The Sensorinstancebased sensor *soft-type* with the reading *soft-types* attached to it can be seen in Figure 60.



Edit sensorinstancebased

Type definition

sensorInstanceBased

Sensor product as realized

Sensor details

✓ Identifier	mainroomSensor
Name	mainroomSensor
✓ Version	1

Sensor readings

Readings

(testsensor2),

(Pi2 temp reading at: Fri, 26 May 2017 15:02:57 GMT Miliseconds:448),

(Pi2 temp reading at: Fri, 26 May 2017 15:03:06 GMT Miliseconds:277),

(Pi2 temp reading at: Fri, 26 May 2017 15:03:28 GMT Miliseconds:747),

(Pi2 temp reading at: Fri, 26 May 2017 15:07:45 GMT Miliseconds:801),

(Pi2 temp reading at: Fri, 26 May 2017 15:07:48 GMT Miliseconds:125),

(Pi2 temp reading at: Sun, 28 May 2017 11:44:16 GMT Miliseconds:769),

Quick select | New

Close | Update

Figure 60 - sensorinstancebased Sensor Soft-Type

6 Testing & Evaluation

I decided to go about evaluating the system in three different ways. The first evaluation comprised several technical tests to see whether the system actually worked. The second evaluation used a focus group made up mainly of computer-science students. I thought they could give helpful feedback on the idea of such a system and the technical architecture of the system. Finally, the third evaluation used a focus group made up of employees at Eurostep. Again, I thought they could give helpful feedback on the idea of such a system and whether they could envisage their customers using it.

6.1 Technical tests

Once the system was fully deployed in the office, I performed technical tests based on the system requirements. This allowed me to find out whether the system had any technical problems and whether it fulfilled those requirements.

Test Number	**The Test**	**Area of the System being Tested**
1	Can both Pi boards connect and retrieve a temperature from the Light Blue Beans?	Pi boards and sensors
2	Can the Pi boards authenticate with ShareAspace?	Pi boards and ShareAspace
3	Can both Pi boards create a soft-type?	Pi boards and ShareAspace
4	Can both Pi boards retrieve a soft-type?	Pi boards and ShareAspace
5	Can both Pi boards update a soft-type?	Pi boards and ShareAspace
6	Can both Pi boards send sensor readings to Azure?	Pi boards and Azure
7	Can ShareAspace trigger an event?	ShareAspace
8	Can analytics be performed on the sensor data collected?	Azure
9	Does ShareAspace accept invalid requests?	ShareAspace
10	Does ShareAspace accept invalid credentials?	ShareAspace
11	How much data does ShareAspace cope with?	ShareAspace
12	Does the system continue operating if a sensor runs out of power?	The whole system

Below is the table of results for these tests. Green shows successful outcomes, red shows unsuccessful outcomes, and yellow shows results that are not binary.

6.2 Technical test results

Test Number	Result	Solution (if needed)
1	Yes – Both Pi boards can use the Node-Red node to get a temperature from the Light Blue Beans.	Test successful
2	Yes – Both Pi boards can use the authenticator node to get an access token from ShareAspace.	Test successful
3	Yes – Both Pi boards can use the create node to send a soft-type JSON to ShareAspace to create a soft-type in ShareAspace.	Test successful
4	Yes – Both Pi boards can use the read node to get a specified soft-type from ShareAspace.	Test successful
5	Yes – Both Pi boards can use the update node to update soft-types on ShareAspace. This was tested with both types on sensor soft-type used in this implementation.	Test successful
6	Yes - Both Pi boards can use the Azure IoT hub node to send messages containing sensor readings. Messages from both Pi boards could be seen arriving at the IoT hub .	Test successful
7	No – It is not yet possible for ShareAspace to trigger an event (such as sending an email) for a soft-type property. However, I was told this would be possible in a later release of ShareAspace.	Wait for a later release of ShareAspace which will include this feature.
8	Yes – Azure can channel data to PowerBI. Power BI can then perform analytics on the data.	Test successful
9	No – I send invalid requests using the create node, update node and read node. When sending an invalid JSON using the create node or update node or an invalid URL with the read node, ShareAspace responds with an error and the nodes say unsuccessful.	Test successful
10	No – sending an invalid ClientID, clientSecret,	Test successful

	username or password leads to ShareAspace responding with a 400 error.	
11	After leaving the system running for a few days I managed to create over three million sensor-reading soft-types. This caused the ShareAspace UI to time out. However, using the get node could still return new soft-types suggesting ShareAspace was still functional. When testing the Sensorinstancebased soft-type sensor ShareAspace, I found ShareAspace started to freeze up after a few thousand soft-types were attached to it. Colleagues at Eurostep think this was due to ShareAspace struggling with indexing.	For this project my solution was only sending readings to ShareAspace when the temperature changed meaning that fewer readings would be stored and indexed by ShareAspace. However, a long-term solution would need to be found by Eurostep.
12	Yes - The flow running on the Pi does however stop, as the Light Blue Bean node reports a disconnection error.	This does not break the system as such but maybe an alert to tell users of the system that a sensor has run out of power would be useful.

Technical testing generally yielded good results. The majority of tests were successful. The only unsuccessful test was ShareAspace triggering an event and this will hopefully be fixed in a future release. Tests did also reveal that ShareAspace struggled to cope with lots of soft-types being attached to another soft-type and that the UI struggled with millions of soft-types. I believe, however, that these two potential flaws will be fixed in later versions of ShareAspace. These tests were also performed with ShareAspace running on a laptop, not a high-performance server and this could have influenced the test results 11.

6.3 Student focus group

The student focus group consisted of four people. Three of those people were studying computer science and one person was studying geography but had some knowledge of IoT. The group was first shown a presentation about the system and a demo of the system in use.

Do you think Eurostep customers are likely be increasing users of IoT technology?

One participant said he thought it *"could be very useful to have sensors"* but that it depended on how scalable the system I made was. He commented that a system like mine would need to be tested in *"a very high stress environment"* and a much bigger environment. Another participant commented that she thought she could see companies using IoT *"because IoT as a concept is increasing generally"*.

When in a product's lifecycle sensor data collection would be most useful?

A participant answered that if the data were used to generate analytics, this would be useful at the end of a product's lifecycle, when the company is thinking "what should our next product be?"

Do you think they [Eurostep's customers] would have a use for a ShareAspace that could cope with IoT data?

One participant's response to this was he thought IoT functionality may be *"a bit outside the core aspect of what people use it [ShareAspace] for"* however he went on to say that he did not think *"it [IoT functionality] negatively affects it [ShareAspace]".* He explained that he thought that some companies would use it but he was not sure how widely it would be used. Another participant joined in by saying he thought it would be *"beneficial to speak to actual companies"* about the IoT functionality. Another comment was that he was unsure because ShareAspace sounded *"buggy"* due to it not being able to cope with very large numbers of soft-types. He went on to explain that if ShareAspace was buggy with IoT data, customers would use another service.

What do you think of the sensor, Raspberry Pi and Node-Red setup?

A participant talked about how he thought it would be better to have the functionality of the Azure side of things on ShareAspace because if companies *"have to use two services, that's not particularly easy"* and explained that to do this ShareAspace will need to cope with masses of data. He went on to say that if ShareAspace did not have the same functionality as Azure, companies may just decide to use Azure.

What are your opinions on the hardware side of things?

After some discussion, two participants agreed that a bespoke sensor, designed for working purely with ShareAspace, might be a good option. They also thought having the sensors perform what the Pi boards currently do would be good if that were possible. This followed from the fact that, the group noted, the Pi would be a bit large to deploy in something like a car. Another participant said the current hardware setup is good for a prototype system but not if it went into mass production.

What do you think of Node-Red?

A participant said that *"a visual language"* like Node-Red *"is probably the way to go".*

Does anyone have any "last thoughts" on the system?

One participant said it was *"a decent idea"* and that *"there is potential"* but *"needs further exploring".* Another participant said the idea may work better *"for non-live data"* and he suggested that it may be better to collect sensor data separately and then add it to ShareAspace to share later.

6.4 Eurostep focus group

The Eurostep focus group consisted of four people who worked with me on the project.

Do you think Eurostep customers are likely be increasing users of IoT technology?

One participant immediately said *"yes"* so I asked him to explain why he thought so. The participant explained that he knew from experience that lots of Eurostep's customers already use sensors and sensor data and that he thought that customers *"are going to use more and more sensors"* because it's *"going to be cheaper".* He then went on to talk about how there are environments which never used to have wireless internet access that now do and that this opens the door to many IoT technologies. Because of this, he expected *“to have customer demand to see that kind of requirement”.*

How do you think Eurostep’s customers would benefit from a system like mine?

A participant gave the example of a ship with sensors on board gathering data and that at the moment the company using the ship needed to wait for the ship to come into dock to collect all the sensor data. He then went on to talk about how a system like mine sending live data would enable maintenance companies to know in advance which parts of the ship need replacing and so on, before it arrived at the dock. This would make maintenance of the ship much faster.

Why do you think it is better to have ShareAspace with IoT capabilities rather than a separate system for dealing with IoT and sensor data?

Once of the participants responded by noting that *“with ShareAspace you have got that crucial link back to the design information”.* He went on to explain how this is useful because it provides more information than just the sensor data. Another participant joined in by giving the example of a model of car and how it would be useful to a manufacturer to link all sensor data from all cars of a certain model back to the design of that model of car, rather than just each individual car.

Could people use the sensor data for when they make new products?

A participant answered by saying that was the aim to give a feedback on *“how you look after the product or how you design the next product”.* He then went on to say *“it’s about tying more information about the product [together]”*.

What do you think of my hardware choices?

One participant suggested it would have been good to *“use more sensors and different types of sensors”.* Another participant stated powering the Pi board may be a concern and that it may be better to use a battery powered device. Another participant said *“I think it was quite good that it was operating with off the shelf components”*, he mentioned his only concern was that the Light Blue Bean *“was not particularly accurate”* as the Light Blue Bean only measures to one degree accuracy.

What do you think of Node-Red?

Once participant said he thought it was good how fast you get programs up and running.

Can you see customers using IoT with ShareAspace and when do you think this may happen?

A participant said that he knew from experience that companies use ShareAspace *“as a data firewall”.* If ShareAspace were to *“separate and hide the collection business”* from the people who have access to the sensor data, it could come *“into play as a potential technology”.*

6.5 Webinar

During the making of the system I gave a webinar to around thirty people in the Eurostep group. The webinar was an overview of the whole project (a recording of it can be found in my working documents). At the end of the webinar I asked if anyone had any questions or feedback. Generally, people were very interested in the idea and someone said that the company needs to understand more about IoT.

6.6 Findings

Overall I feel that the technical tests showed that the main technical objectives of the project were met. As has been reported, the tests run were based on the technical requirements of the system. The findings from the two focus groups should be considered when further developing the system and when developing new releases of ShareAspace. Participants gave good examples of improvements for the system - such as using a more accurate sensor or even designing a bespoke sensor. Participants also made it clear it was important to fix bugs in ShareAspace.

7 Conclusions

7.1 Review of aims

Below are the four objectives that I set for this project. For each one I shall analyse whether and how this objective has been met.

a) Finding out about related systems.

In my literature review I researched the topic to find out if anyone else had attempted something similar to my project. I did find lots of information on how IoT could be useful in product data management and sharing. However, while I did learn about software like ShareAspace, I did not find any ShareAspace like systems using IoT. This led me to believe I was exploring quite a new area, and indicates that this objective was met.

b) Building a real-life system.

To explore the subject further I decided it would be good to build a real-life system in order to see how helpful IoT can be working with software like ShareAspace. I successfully built and tested the system consisting of sensors feeding data into ShareAspace. Technical tests were fairly successful and suggested that with a little more development ShareAspace would be IoT/live data ready and compatible. Because of this I feel the objective was met.

c) Investigating how such a system could be applied to companies.

To investigate this I decided to hold two focus groups and a webinar on the system I built, hoping that my system deployment at the Eurostep office would demonstrate how such a system could be used by other companies in real-world scenarios. The focus groups both generally gave positive feedback and came up with quite a few scenarios in which they system could be used by companies. One example was of a company using ShareAspace and IoT to collect maintenance data about a ship so that, when it arrived at dock, the ship would be much quicker to service. Another included a system similar to mine deployed in cars to gather live data which could be useful when making a new model of car. Both these examples and the success of the deployed Eurostep office system lead me to conclude this objective was met.

d) Then, exploring how such a system could help these companies.

The focus groups again gave lots of insight into how a system like the one developed could be useful to companies. As was noted under objective C, live data could help companies speed up servicing of vehicles and help companies when developing new products. In my literature review, I also covered how useful it would be to companies to have sensor data on to product management systems. Again, this literature covered how such systems would help with maintenance but also how such systems would help companies deal with regulation and environmental requirements. All this information makes me feel this final object has also been met.

Overall achieving these objectives have helped me answer my research question which was:

Why and how is IoT and real-time data useful in the context of product data sharing and exchange?

7.2 Suggested revisions to design or implementation

Starting with the hardware used in my system, I think if I was to design the system again I would have used a more accurate sensor. The Light Blue Bean is only accurate to one degree; I think it would be better to have a sensor which is accurate to at least 0.1 degrees. This point was also mentioned in the Eurostep focus group. Next, although the Raspberry Pi 3 was a great choice at the time, the new

Raspberry Pi Zero W (Upton, 2017) would now be better. This is because the new Raspberry Pi Zero has Bluetooth and Wi-Fi inbuilt just like the Pi 3, however the Pi Zero W is much cheaper and smaller than the Pi 3. I think Node-Red was a very good choice and this was also felt by participants in the focus groups.

I feel that Azure and PowerBI were a great way of visualizing the data. It would however, be nicer if PowerBI could access data stored on ShareAspace directly as this would cut out the need for Azure.

7.3 Future work

If I had had more time I would have liked to add some more features to the deployed system and I would have liked to do some more testing. One feature I would have liked to see would be ShareAspace responding to events and sending an email or an SMS when the temperature reached a threshold to alert users. Unfortunately, as mentioned in my testing, this was not yet possible with the current version of ShareAspace but should be possible in a future release. It would also be nice to see a future *soft-type* in ShareAspace that can cope with lots of other *soft-types* being attached to it. Testing wise it would have been interesting to speak to some of Eurostep's customers to see what they think of idea of a system like the one developed. It would then have also been exciting to test such a system on a large scale on something like a ship or in a car.

7.4 Lessons learned

I feel I have certainly learnt a lot during this project. First of all, I had to learn the JavaScript programming language which I had never used before. This knowledge then enabled me to program a library for ShareAspace. I was particularly happy with achieving this as I found it a challenge. I also learnt about PDM systems and data modelling, two things not covered in my degree. I also found it particularly interesting working with a company on this project as it gave me a real overview of what industry is like. Overall I personally feel as though this project was a success and I very much enjoyed working on it.

References

Barry, D. (2015, September 23). *How to use Device Explorer for IoT Hub devices*. Retrieved May 15, 2017, from github: https://github.com/fsautomata/azure-iot-sdks/blob/master/tools/DeviceExplorer/doc/how_to_use_device_explorer.md

Butcher, M. (2014, August 28). *Enevo's Waste Bins Sensors Attract $8M Funding From Earlybird And Draper Associates*. Retrieved May 11, 2017, from techcrunch: https://techcrunch.com/2014/08/28/enevos-waste-bins-sensors-attract-8m-funding-from-earlybird-and-draper-associates/

DASSAULT SYSTÈMES. (n.d.). *enovia*. Retrieved May 24, 2017, from DASSAULT SYSTÈMES: https://www.3ds.com/products-services/enovia/

embeddedcomputing. (n.d.). *IoT with LightBlue Bean and Node-RED*. Retrieved May 24, 2017, from embeddedcomputing.weebly: http://embeddedcomputing.weebly.com/iot-with-lightblue-bean-and-node-red.html

Eurostep. (n.d.). *documentation*. Retrieved June 1, 2017, from ShareAspace: https://mossec.eurostep.com/documentation/index.html

Eurostep. (n.d.). *ShareAspace Concept*. Retrieved May 20, 2017, from Eurostep: http://www.eurostep.com/products/shareaspace/

KarelK. (n.d.). *Bluetooth Water Detection System*. Retrieved May 24, 2017, from instructables: http://www.instructables.com/id/Bluetooth-Water-Detection-System/

Microsoft. (2017, May 2). *Overview of the Azure IoT Hub service*. Retrieved May 22, 2017, from Microsoft: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-what-is-iot-hub

Microsoft. (n.d.). *Azure Stream Analytics*. Retrieved May 10, 2017, from Microsoft Azure: https://azure.microsoft.com/en-gb/services/stream-analytics/

Microsoft. (n.d.). *what is erp*. Retrieved May 24, 2017, from Microsoft dynamics365: https://www.microsoft.com/en-us/dynamics365/what-is-erp

Microsoft. (n.d.). *What is Power BI?* Retrieved May 24, 2017, from PowerBI: https://powerbi.microsoft.com/en-us/what-is-power-bi/

Murach-Ward, M. (2017). *SCC 419 Project Proposal.* Lancaster.

NEST. (n.d.). *Nest*. Retrieved May 11, 2017, from meet-nest-thermostat: https://nest.com/uk/thermostat/meet-nest-thermostat/

Node.js. (n.d.). *About Node.js*. Retrieved May 22, 2017, from Node: https://nodejs.org/en/about/

nodered. (n.d.). *About*. Retrieved February 2, 2017, from nodered: https://nodered.org/about/

nomagic. (n.d.). *magicdraw*. Retrieved May 27, 2017, from nomagic: https://www.nomagic.com/products/magicdraw

plcs. (2013, May 03). *Product Life Cycle Support*. Retrieved May 26, 2017, from plcs: http://www.plcs.org/

ptc. (n.d.). *Windchill PDM*. Retrieved May 24, 2017, from ptc: http://www.ptc.com/en/product-lifecycle-management/windchill/pdm-essentials

Raspberry Pi. (n.d.). *raspberrypi*. Retrieved January 28, 2017, from raspberrypi: https://www.raspberrypi.org/

Raspbian. (n.d.). *About Raspbian*. Retrieved May 22, 2017, from raspbian: https://www.raspbian.org/RaspbianAbout

Shilovitsky, O. (2014, April 28). *3-things-plm-can-do-with-iot-tomorrow*. Retrieved May 24, 2017, from beyondplm: http://beyondplm.com/2014/04/28/3-things-plm-can-do-with-iot-tomorrow/

Siemens. (n.d.). *Explore Teamcenter and discover your solution*. Retrieved May 22, 2017, from plm.automation.siemens: https://www.plm.automation.siemens.com/en_gb/products/teamcenter/

Siemens. (n.d.). *PDM / Product Data Management*. Retrieved May 24, 2017, from Siemens automation: https://www.plm.automation.siemens.com/en_gb/plm/pdm.shtml

Thoben, K.-D., & Lewandowski, M. (2016). Information and Data Provision of Operational Data. In A. Bouras, B. Eynard, S. Foufou, & K.-D. Thoben, *Product Lifecycle Management in the Era of Internet of Things* (pp. 20-29). Springer.

Upton, E. (2017, February 28). *NEW PRODUCT! RASPBERRY PI ZERO W JOINS THE FAMILY*. Retrieved 22 May, 2017, from Raspberry Pi Blog: https://www.raspberrypi.org/blog/raspberry-pi-zero-w-joins-family/

xrdp. (n.d.). *xrdp: An open source remote desktop protocol(rdp) server.* Retrieved May 15, 2017, from xrdp: http://www.xrdp.org/

Acknowledgments

I would first like to thank Dr Ioannis Chatzigeorgiou, my tutor for his help and support throughout the project, along with the Computer Science department. I also want say a massive thank you to everyone at Eurostep, without them this project would not have been possible. I would also like to thank my parents, friends and Hannah for proof-reading and support. Finally, I would like to extend my gratitude to the participants in the testing stage.