NUI Galway
OÉ Gaillimh

NATIONAL UNIVERSITY OF IRELAND, GALWAY

MASTER'S THESIS

Measuring Environmental Conditions in Smart Buildings Using Node Red

---

Author:
James NGONDO

Supervisor:
Dr. Ali INTIZAR

A thesis submitted in fulfillment of the requirements
for the degree of Master's Degree
in the

Data Analytics
Department of Computer Science

September 22, 2017

Declaration of Authorship

I, James NGONDO, declare that this thesis titled, "Measuring Environmental Conditions in Smart Buildings Using Node Red" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a Master's Degree at National University of Ireland, Galway (NUIG)
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"Then Samuel took a stone and set it up between Mizpah and Shen. He named it Ebenezer, saying, "Thus far the Lord has helped us.""

1 Samuel 7:12

National University of Ireland, Galway

Abstract

Computer Sciences

Department of Computer Science

Master's Degree

Measuring Environmental Conditions in Smart Buildings Using Node Red

by James NGONDO

The world is experiencing a new era of growth and development as far as computing technology is concerned and where we see our everyday lives dependent on it. We are experiencing the Internet of Things (IoT) where billions of smart devices, also known as "things", are connected together and able to interact and communicate with the each other as well as the environment and infrastructure around us. These smart and intelligent devices have greatly contributed to informed decisions being made and improved lives as a result of huge volumes of data they generate. Some of these devices make use of sensors that monitor different environmental conditions such as temperature, humidity, air pollution, sound and light. Data produced is analyzed and visualized in order to give meaningful insights. Different computing platforms and frameworks that aide in the wiring up of Internet of Things (IoT), capturing and visualizing real-time data have been developed and made available to us. In this study, we analyze environmental conditions in smart buildings and I propose an automated system which can monitor environmental conditions in smart buildings and trigger notifications whenever conditions surpass any predefined thresholds.

I propose to use IBM's IoT platform Node Red to help capture, store, visualise and effectively mine various sources of data across the building, along with other contextual information streams e.g. twitter. In this study, I will focus on temperature and humidity sensors. The overall objective is to more effectively monitor temperature and humidity in a smart building.

Acknowledgements

I would like to thank my supervisors, Dr Ali Intizar and Dr Conor Hayes for guidance and the feedback I've received throughout my work.

I would also like to thank Gilbeta Hazel Gombe, for her continuous support and encouragement throughout my years at National University of Ireland, Galway (NUIG)…

Contents

List of Figures

I dedicate this work to all my friends and family, especially Joyce Chingota (mother), Gilbeta Gombe, Tom and Esther Chiphwanya, John and Atuweni Mponda, Mark and Silvia Malema...

Chapter 1

Introduction

Keywords: Internet of Things(IoT), Node-Red, Javascript, D3, Adafruit, Arduino

Internet of Things (IoT) describes a technology that has changed the way smart objects communicate and interact on the Internet. These connected objects can identify each other and communicate over the Internet based on IP addresses assigned to them. Wiring of Internet of Things (IoT) applications can be a complex and challenging task as it may involve integrating different protocols and services. This also takes into account the fact that they are huge volumes of time series data produced by these IoT applications and that this data must be gathered, pre-processed or post processed in real time. IoT is widely influencing the way technology affects our everyday life, with interconnected devices in homes, cities and manufacturing industries. According to (McKinsey-Global-Institute, 2015) it is estimated that worldwide economic value of IoT will go beyond **$11 trillion**, from different settings such as industries, cities, work-sites etc. by **2025**. As the number of connected intelligent and smart objects grows enormously, businesses and individuals will have to provide answers as to how they intend to store huge volumes of data generated by those "Things" while trying to produce meaningful insights and making sense from the data. Also, there has to be sophisticated technological tools and frameworks which will make wiring up of IoT easy and intuitive.

On top of IoT infrastructure, lays a need to design real-time monitoring systems that remotely monitor and analyse environmental conditions as regard to different aspects of life. This form of technology comprises the following systems:

(i) data acquisition systems that aim to collect and store various types of monitored

data

(ii) data transmission system that facilitates the transfer of data between collecting points

(iii) data processing system that continually take data as input, analyzes this data in real-time and provide results of analytics to end users.

Building such real time applications require extensive coding as well as financial costs that are involved in setting up and integrating various system components and frameworks. In contrast, I propose to use **Node Red** for building such systems.

Node Red has emerged as one of the tools that provide solution to the challenges of wiring up of the IoT devices. This visual tool allows developers, engineers as wells individuals with less programming or development experience, build workflows that reflect a particular IoT scenario, as explained in the next sections.

Figure 1 in the appendix section below indicates the trend of IoT development.

1.1 Project Goals, Objectives and Scope

(i) provide an online environmental conditions monitoring system for smart building,

(ii) Access and analyze data on the fly.

(iii) Develop a notification system to trigger notification based on outcomes of data analytics.

(iv) Provide an easy to use interface following the example scenarios in order to easily develop real-time IoT applications using Node Red.

This project will involve intense research on designing and implementation of a temperature and humidity monitoring system that generates and visualizes real-time data using technologies and devices such as Node-Red, Arduino Uno and DHT11 temperature and humidity sensor. I will also work towards creating archived/ historical temperature and humidity data that can be used for future predictions and analysis. This process will involve implementing Arduino libraries such as **Arduino Excel** that allow real-time sensor data to stored in an excel sheet**(.xlxs)** file.

In this project, Node-Red plays a vital role in the sense that it is capable of easily integrating different controllers and devices that communicate over the internet using TCP/IP protocols. In this case, Node-RED will receive data as input from connected controllers and devices such that through specified rules and conditions within its nodes in a work-flow, triggers an event and outputs messages can be channelled to tweets and emails.

1.2 Methodology

This section describes how I went about doing research and gathering enough information that enabled me to carry out this study.
Having considered all the necessary information from the project specification, I conducted a thorough research on the latest technology that can be used to develop a robust application that meets the project requirements. This was done through literature review based on different published conference papers, online journals and website contents. Many scholars have produced a lot of different articles and publications regarding IoT, but less has been done as regard to Node Red, which is a platform for wiring IoT devices.
When I learned that there's only a handful of published literature related to this project, I used credible articles from different web sources and blogs in order to support this work. I had keen interest in trying to figure out technologies that best suites the purpose of this project. As part of my research, and I was also able to interact and learn from developers around the world some of them, have developed the Node Red framework, for example, **Nick O'Leary**.

1.3 Outcomes

Real-time temperature/ humidity data is generated using a DHT11 sensor connected to an Arduino micro-controller. The data is filtered through Node Red flows that run in JavaScript. Filtered temperature and humidity values are visualized using Node Red dashboard charts and gauges. According to the results that were obtained after implementing the temperature/ humidity monitoring system, it was discovered

that during the night, temperature values were between 21°C and 22°C below the set threshold of 26°C and this did not trigger any event. The corresponding humidity values were between 51% and 55% which was above the set threshold of 48% and eventually triggered an event in form of a tweet. The same process was repeated during the day where we also found that temperature readings were a bit higher (temperature at 24°C) as compared to night readings but were still below the threshold. Humidity readings during the day were lower than during the night readings (humidity at 44%) hence no alert message was triggered. From this I was able to conclude that humidity was higher during the night while temperature dropped. Humidity decreased during the day signalling that air in the room became hotter, thereby holding as much moisture as it can. Temperature readings rose during the day but were still below the threshold.

1.4 Thesis Structure

This section provides a brief overview of the chapters that are covered this work.

1.4.1 Chapter 2

In Chapter 2, we review state of the art providing information related to the history and development of Internet of Things (IoT).

1.4.2 Chapter 3

In Chapter 3, we discuss a number of emerging IoT technologies that are related to Node Red due to a growing demand in IoT frameworks that able to process and visualize real-time streams of data.

1.4.3 Chapter 4

In Chapter 4, we explain different components of the system architecture such as temperature/ humidity sensors, micro-controllers etc, and how they are tied together. The goal of this section is also to provide an internal logic for each system module identified during system design.

1.4.4 Chapter 5

In Chapter 5, we highlight the output results that were obtained after the application was run in one of the rooms in a smart building. These results are displayed in the as visual charts and also as tweets.

1.4.5 Chapter 6

In Chapter 6, we explain how temperature and humidity real-time sensor data is archived and stored in an excel file using Excel File Using Arduino Excel 2.1 framework.

1.4.6 Chapter 7

Chapter 7 relates to future work and we explain what other technologies could have been added or implemented in this project.

1.4.7 Chapter 8

In Chapter 8, we draw a conclusion on the entire work.

1.4.8 Appendix

The Appendix section includes all the screen-shots as well as source codes for the entire project.

Chapter 2

Internet of Things

"**Internet of Things**" (IoT) is a phrase that was invented by Kevin Ashton in 1999 and David L. Brock in 2001, founders of the MIT Auto-ID Centre (CISCO, 2013). "If we had computers that knew everything there was to know about things—using data they gathered without any help from us – we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best. We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory. RFID and sensor technology enable computers to observe identify and understand the world—without the limitations of human-entered data." (CISCO, 2013) According to (Brock, 2001) (AIM-Global-Network, 2005), in 2003, Auto-ID Centre launched the EPC (Electronic Product Code) as its initial IoT application, and it was intended to distinctively identify all physical objects, for example, a box of detergent which had a unique EPC attached to it. The EPC itself was embedded into an electronic tag, and this helped identify the product as it was being moved through the supply chain - right from the manufacturer through to the retailer and all the way to the consumer (Brock, 2001) (EPCglobal, 2005) (AIM-Global-Network, 2005).

2.1 IoT in application

(Yen-Kuang, 2012) explains that real world objects such as coffee machines, vending machines, refrigerators, security cameras, televisions, computers etc are connected together through Internet hence allowing users to access and gather data from these devices and be able to monitor their homes and surroundings from a single point

and be able to make more informed decisions. IoT is also being applied in different aspects of life and has become part of our everyday life. IoT has brought total transformation to industrial, commercial and consumer segments as discussed below: (UL, 2016) (Gartner, 2015) (Walsh, 2015).

2.1.1 Commercial / Residential Buildings

IoT has contributed a lot to the existence of smart buildings. Buildings have ceased to operate only as containers that support living beings and have become objects that carry life and this is based on the amount of data that we are able to access and gather from different devices are attached to them. IoT has enabled people who own buildings/properties, tenants and even facility managers to make key decisions that lead to efficient operating and management of buildings hence providing enhanced user experience (UL, 2016) (Gartner, 2015)(Walsh, 2015). The application of IoT in this sector allows users to remotely monitor and control different home appliances ranging from, lighting, heating and air conditioning, water usage, entertainment systems to premises security systems. Different wearable gadgets such as smart phones, watches and even eyeglasses are also included as part of this huge technology (UL, 2016) (Gartner, 2015) (Walsh, 2015). Gartner, the world's leading technology research company, recently released a smart city forecast indicating that "Smart commercial buildings will be the highest user of Internet of Things (IoT) until 2017, after which smart homes will take the lead with just over 1 billion connected things in 2018" (Gartner, 2015).

2.1.2 Public safety

There has been great improvement in public safety and services in towns and cities as a result of adopting IoT technologies that are at hand. IoT enables law enforcement officials monitor crime activities easily and hence effectively use their resources. For example, law enforcement officials are able to use video cameras placed around the cities, towns etc. to identify exact locations where and when crime occurred and the people behind the crime. On the other hand, monitoring of paid parking

lots and non-working traffic lights is made easy hence reducing city congestion (UL, 2016) (Gartner, 2015) (Walsh, 2015).

2.1.3 Energy production, distribution and consumption

IoT has contributed to the growth of smart grid technologies that have transformed the way energy is being produced, distributed and consumed. This technology allows consumers to make right decisions regarding the way they consume energy. Different smart grid applications such as "smart meters" have been developed and deployed in order to enable energy consumers effectively monitor and control the way they consume energy hence leading to reduced energy costs (UL, 2016). This provides flexibility in the sense that energy consumers are able to know how much energy is being consumed, when and its cost in real time and hence they do not have to wait for a monthly statement in order to have a clear picture of how much energy they have consumed. In terms of energy suppliers, smart grid technology enables them to accurately estimate consumer usage and as well as providing cost-effective source of energy. According to (Gartner, 2015), IoT Smart grid technology provide a more efficient way of energy transmission with reduced operational costs for utilities and lower energy costs for consumers (Walsh, 2015).

2.1.4 Transportation

IoT has transformed the transportation sector in a way that enables people and goods to move from one place to another more quickly and efficiently through the use of IoT smart cars and smart roadways. In this sector, IoT applications are playing a major role and have contributed to the efficient monitoring and management of traffic flow and helped minimize traffic congestion in town and cities. There are some IoT applications that have been specifically designed to to help users identify vacant parking lots. On air transportation, aircrafts are fitted with sensory devices that transmit data every single second and help monitor and analyze the state of the aircraft thereby contributing to more improved maintenance and safety standards. Transportation and Logistics businesses have benefited a lot from IoT where business owners, managers and other stakeholders are able to make more informed

decisions due to the fact that data related to logistics and warehouses is being captured and frequently shared across different locations ensuring that right products are at right places at the right time (UL, 2016) (Gartner, 2015) (Walsh, 2015).

2.1.5 Healthcare

This sector includes use of mobile and wearable devices e.g. wristbands, mobile phones, tablet computers and PDAs that can be used to both monitor and record personal or patient medical data in real time. (Walsh, 2015) explains that these devices are considered as health and fitness devices and are part of mHealth (mobile Health) technology applications that enable healthcare professionals, practitioners and researchers to access patient data in real time, from any location and be able to provide proper care directly (UL, 2016) (Gartner, 2015).

2.1.6 Environment

Measures to tackle environmental conditions that pose a threat to human, animal or plant life have flourished as a result of IoT technology where there are IoT sensory devices that can easily detect climate changes due to air pollution leading to catastrophic rising water levels in oceans, rivers and streams that could lead to flooding, forest fires, earthquakes etc. IoT is helping reduce some of the known carbon footprints and as well as massive pollutions in our planet by collecting huge amounts of data related to air conditions thereby allowing us to analyze living environments do even with fewer resources. IoT environmental devices have by far contributed to healthier work and living environments, improved food security, reducing the adverse impacts of weather and climate changes through data streams being collected every moment.

All the benefits, related to improved living environments, are being realised or achieved as a result of enormous pools raw data produced every minute by thousands of connected devices that enable us make informed decisions (UL, 2016) (Gartner, 2015) (Walsh, 2015).

2.1.7 Industrial and manufacturing

In the manufacturing sector, IoT applications have greatly contributed to improved efficiency and productivity of the manufacturing operations. Manufacturing equipments (plants) are now fitted with IoT sensory devices that help monitor different manufacturing equipments and enable both stakeholders and users make informed decisions as to when servicing or maintenance can be done. IoT technology also helps managers to monitor the usage of raw materials as well as inventory levels. Intelligent assets and equipments that make full use of IoT technology are able to sense issues and perform self-diagnosis that helps reduce equipment downtime and hence increase process efficiency as well as preventing production line delays. This too results in reduced energy and resources costs at manufacturing facilities (Gopal, 2016).

Chapter 3

Designing IoT Applications

There are a number of emerging IoT technologies that are related to Node Red. These technologies have emerged as result of a growing demand in IoT frameworks that are capable of processing and visualizing real-time streams of data.

3.1 Front End for Node-RED (FRED)

FRED which stands for a 'Front End for Node-RED', was developed by **Sense Tecnic**, a company that commercially provides IoT solutions. Sense Tecnic is also an active member of the Node Red community and has contributed more to the development of core Node Red sources and has also developed a number of nodes that are included in the Node Red framework. This framework is available to users in two ways - (i) as a free service with limited resources (ii) as a paid subscription where users or customers can avail of unlimited resources that include an integrated MQTT service as well as a 24 hour customer support (Sense-Tecnic-Systems, 2017).

FRED was developed as an alternative to overcome the limitations faced by **IBM**'s **Node Red** which is limited to executing a single flow file in a single thread.

FRED is a cloud hosted Node Red framework designed to lessen the burden of having to manually and locally install the Node Red application framework on your computer. According to (Michael-Blackstock, 2016) (Sense-Tecnic-Systems, 2017), the FRED application framework has been designed in such a way that it can manage multiple instances of Node-RED for a large number of logged in users and it is able to monitor and automatically restart Node-RED instances in cases where nodes encounter configuration problems as well as crashes.

3.1.1 FRED MQTT Service

FRED offers a bundled MQTT (Message Queue Telemetry Transport) service only to customers with a paid subscription. This service makes it easy for users to built IoT based applications that rely on remotely deployed IoT devices that send data over MQTT (Michael-Blackstock, 2016) (Sense-Tecnic-Systems, 2017).

3.1.2 Technical Limitations of FRED

According to (Sense-Tecnic-Systems, 2017) (Michael-Blackstock, 2016) FRED has two technical limitations as follows: **(i)** FRED makes full use of a proxy and firewall and so Node-RED is hosted behind the proxy and firewall. In this case, no device or "thing" can communicate with your instance of Node-RED except through FRED. This means that some input nodes will have to be configured based on the security standards that are acceptable with FRED for them to work. **(ii)** Since FRED is a cloud service, there are some restrictions that have been put in place such as access to nodes that depend on underlying operating system for storage and sensors. **(iii)** Some of Node Red instances are programmed to stop running after a certain period of time but this varies depending on user subscription and associated time and node count limits. (Sense-Tecnic-Systems, 2017) (Michael-Blackstock, 2016)

3.2 AWS IoT (Amazon Web Services)

Amazon offers a platform known as **AWS IoT** that enables Internet-connected things such as sensors, actuators, embedded devices, or smart appliances to connect to Amazon Web Services. Interaction between the Internet-connected things as well as

data being generated by the connected things is made secure.
AWS IoT is designed in such a way that enables applications to interact with devices even when they are off-line. (Amazon-Inc., 2017b) Communication between the Internet-connected things is bi-directional hence allows users to collect, record and transmit data readings from multiple devices and store and analyze the data (Amazon-Inc., 2017a).

3.2.1 AWS IoT Components

According to (Amazon-Inc., 2017b), AWS IoT is made up of the following components:

- **Device gateway** - designed to enable Internet-connected things to securely and efficiently communicate with AWS IoT.
- **Message Broker** - providing a secure mechanism for connected things and AWS IoT applications to publish and receive messages from each other. Messaging protocols such as MQTT directly or MQTT over WebSocket can be used to publish and subscribe (Amazon-Inc., 2017b) (Amazon-Inc., 2017a). Sometimes HTTP REST interfaces can also be used to publish messages.
- **Thing Registry** - also referred to as the "Device registry", is responsible for organizing the resources associated with each connected thing. As connected things are registered, up to three custom attributes will then be associated with with each thing. Other certificates and MQTT client IDs can also be associated with each thing in order to improve the user's ability to manage and troubleshoot their things (Amazon-Inc., 2017b) (Amazon-Inc., 2017a).
- **Device Shadow** - Sometimes referred to as a "thing shadow". This component makes use of a JSON (JavaScript Object Notation) document to store and retrieve the current state information for a thing (Amazon-Inc., 2017b) (Amazon-Inc., 2017a).
- **Device Shadows Service** - this is also known as "Thing Shadows Service". It is meant to provide persistent representations of users' connected things in the AWS cloud. This component can be also be used to publish updated state

information to a thing shadow, and the thing can synchronize its state when it connects. The things can also publish their current state to a thing shadow for use by applications or devices (Amazon-Inc., 2017b) (Amazon-Inc., 2017a).

-

3.2.2 Limitations related to AWS IoT Services

Amazon imposed a good number of limits for AWS services for an AWS account. But with regard to AWS IoT, below are some of the default limits (Amazon-AWS-IoT, 2016):

- **Thing Limits** - (i) Amazon AWS IoT limits a "Thing" name size (device name size) to 128 bytes of UTF-8 encoded characters and this limit applies for both the thing registry and Thing Shadow services.
  (ii) A thing with a thing type is limited to a maximum number of thing attributes of 50 and only one thing type can be associated with a thing (Amazon-AWS-IoT, 2016).

- **Message Broker Limits** - (i) Amazon AWS IoT make use of MQTT as a message broker, and for this reason, it allows an MQTT client connection to disconnect after 30 minutes of inactivity, by default and its inactivity timer automatically resets When the client sends a PUBLISH, SUBSCRIBE, PING, or PUBACK message.
  (ii) According to (Amazon-AWS-IoT, 2016), an AWS IoT account is limited to a maximum of 300 MQTT CONNECT requests per second where a single SUBSCRIBE call is limited to request a maximum of eight subscriptions.
  (iii) Every PUBLISH message is limited to 128 KB as its payload and for this reason, AWS IoT service is set to reject all messages that exceed 128 KB.
  (iv) (Amazon-AWS-IoT, 2016) clearly states that a message should not exceed 256 bytes of UTF-8 encoded characters When a topic is being passed to the message broker at the time of publishing
  (vi) 24 hours is a set limit for WebSocket connections and if the limit is exceeded, the WebSocket connection is automatically closed when an attempt is made to send a message by the client or server. In order to maintain an active

WebSocket connection for longer than 24 hours, (Amazon-AWS-IoT, 2016) advises you to simply close and reopen the WebSocket connection from the client side before the time limit elapses.

- **Device Shadow Limits** - According to (Amazon-AWS-IoT, 2016), Amazon AWS IoT JSON state document should have at least a maximum number of 6 levels and with a maximum size of 8 KB. This simple means that a thing name will be limited to a maximum size of 128 bytes of UTF-8 encoded characters and that a thing shadow is given a life span of up to six months before it is deleted by AWS IoT.

- **Security and Identity Limits** - Amazon allows only a maximum number of 15 device certificates to be registered per second and only 10 policies can be attached to a certificate or Amazon Cognito identity (Amazon-AWS-IoT, 2016).

3.3 Adafruit IO

Adafruit IO is a system designed to enable users with little or no programming experience build application that simplify data connection and make data useful. The Adafruit IO system is built on built on Ruby and Rails, and Node.js. The **REST** and **MQTT** APIs form the core of the system and are wrapped within the client libraries. The system also has a customizable dashboard designed to provide an easy way for interaction between users and devices. The dashboard comprises a few widgets that make data useful (Adafruit, 2016c).

This system is compatible with different sensors, for example, **DHT11 / DHT22** sensors that send temperature and humidity values using wired or wireless Internet connection to Adafruit IO. Feeds form the core functionality of Adafruit IO (Adafruit, 2016c).

3.3.1 Adafruit IO Feeds

Devices such as sensors constantly push data to Adafruit IO. Adafruit IO Feeds contain sensor data values and are responsible for holding metadata about the data that is being pushed to Adafruit IO. These Feeds also have settings that determine the type of license with which the stored sensor data falls under and also whether the data will be made public or private (Adafruit, 2016a). According to (Adafruit, 2016a), it is advised that different sensors have unique feeds created for them, for example, where there are three sensors - one for temperature and two for humidity, threes feeds will have to be created for each sensor.

3.3.2 Adafruit IO Dashboard

The Adafruit IO Dashboard is displayed in form of a modern web browser that allows connected IoT devices visualize real-time data streams using charts, sliders, and buttons without having to write any custom code.

3.3.3 Feeding data to Adafruit IO using MQTT Broker

According to (Adafruit, 2016b) pushing data from IoT device(s) to Adafruit IO is made easy with the use of libraries such as Adafruit IO MQTT Broker. (Adafruit, 2016b) clarifies that MQTT is the best option as it is designed to run on top of a wide range of networks that run different network protocols e.g. TCP/IP, Bluetooth etc. It is also considered as an extremely simple and light-weight in the sense that it only requires a very small amount of information storage in order to connect to the server (80 bytes). Using MQTT, data publication (pushing data from device to a remote server) and subscription (pushing data from a remote server to device/ consumer) occurs instantly, meaning that you stay connected the entire time as the process is taking place (Adafruit, 2016b). In order to understand how MQTT Broker process works, (Adafruit, 2016b) uses a simple illustration as in Figure 16 in the Appendix section below, where different IoT devices such as a home toaster that has a WiFi chip and is connected to Internet and also a geo-tracker in your car that has a cellular as well as a Global Positioning System **(GPS)** connection. Both the toaster and car will have to be connected to a hosted computer server that allows them to

send messages back and forth. According to (Adafruit, 2016b), this server that is responsible for handling messages is called the **MQTT Broker** - which is considered as a neutral party that your Things can connect to and be able to send and receive messages.

3.3.4 Limitations using Adafruit IO with MQTT Broker

According to (Adafruit, 2016b), Feeds - (which are termed as a set of data that you can read or write from like a sequential file), store some historical data and hence with MQTT it becomes impossible to access such historical data.

3.4 Dweet.IO

dweet.io is a framework that allows IoT devices, machines, sensors, robots, and gadgets that connect to the Internet to to easily publish and subscribe to data. Each device or **"thing"** is assigned a unique thing name which allows it to be subscribed to. This framework is available to users as a free service with limited resources or as a paid subscription where users or customers can avail of a wide range of features without any restrictions (Bug-Labs-Inc., 2016). Dweet.io provides a solution that enables IoT machines and data collected from the sensors to become easily accessible through a web based RESTful API. This allows users to create apps in no time and simply share data (Bug-Labs-Inc., 2016). Below is a list of the examples data that can be used with dweet.io (Bug-Labs-Inc., 2016):

- Public swimming pool temperature
- Air quality information in a city
- Traffic light status
- Train GPS position
- Room temperature and Humidity
- Room sound recordings

Dweet.io is not an open-source platform.

3.4.1 Dweet.IO Benefits

Dweet.io offers several benefits as follows:

- Data from different devices is automatically sent to a cloud service and users do not require any setup or sign-up — just publish and go.
- A dweet payload can be up to 2,000 characters compared to a tweet on Twitter that is limited to 140 character only.
- Dweet.io allows users to easily create Internet of Things dashboard for different applications (Bug-Labs-Inc., 2016).
- dweets can be made private by purchasing a lock for your thing. This lock allows you to protect and reserve your thing so that only people as well as apps, machines, etc with special keys can access its dweets. Once you lock a thing, the name of that thing cannot be used by anyone else (Bug-Labs-Inc., 2016).

3.4.2 Dweet.IO Limitations

For a free service, dweets are public by default and the thing's last 5 dweets are stored for up to 24HRs unless it's a locked dweet which is stored for 30 days. Each lock costs $1.99 and the cost includes reserving a thing name and keep your dweets private. (Bug-Labs-Inc., 2016)

3.5 Freeboard.IO

Freeboard.IO is a HTML-based framework that allows you to build real-time, interactive dashboards and visualizations by providing an intuitive drag and drop interface. It also provides a plugin architecture for creating data-sources that are responsible for fetching data and the widgets which display data. Freeboard has been designed to connect the data-source and the widget together.

Another incredible feature in Freeboard is its ability to run entirely in the browser as a single-page static web application that does not require a server at all. Some of

the embedded devices may have limited abilities in terms of being able to serve complex and dynamic web pages hence freeboard provides a solution to this by serving as an attractive front-end for all the embedded devices (Bug-Labs-Inc, 2016a).

3.5.1 How to run freeboard

Freeboard is designed to run entirely from a computer's local hard drive. This application framework also runs as a stand-alone application.

3.5.2 Freeboard Limitations

Loading Freeboard from a computer's local hard-drive may result in CORS (Cross-origin resource sharing) issues when trying to access JSON based APIs, especially when using Chrome web browser. To overcome this issue, the user will either have to switch to using JSONP or load index.html file and then run from a local or remote web server. (Bug-Labs-Inc, 2016b) (Bug-Labs-Inc, 2016a)

Having considered all the above IoT applications that aim to provide the best services in terms of working with real-time stream data, Node Red has proven to be one of the best options available.

3.6 Node Red

Node Red was specifically developed as tool for IoT. According to (Blackstock, 2014), Node Red is regarded as an open source web-based tool that allows us to wire together hardware devices and APIs based on the concept of the Internet of Things. It was developed by IBMer Nick O'Leary, and this tool is browser-based which makes it easy to wire up flows using the wide range of nodes that can be dragged

and dropped from the palette on to the editor. This technology uses JavaScript functions and is built on Node.js hence having a lightweight runtime and taking full advantage of **Node.js** event-driven, non-blocking model.

This tool is also designed to run of Raspberry PI also Arduino. Since it is open source, you can create a flow that can be stored in a JSON file and then be exported for sharing or imported by others. (Blackstock, 2014) explains that a Data flow program in Node Red is considered as process where nodes are be wired together to exchange information and these are called flows. The systems make use of a visual editor that allows the user to drag and drop node templates to generate data flows (Blackstock, 2014). Node Red execution process begins with reading a flow file and then instantiating the nodes that are related to that specific flow file. Once the input nodes are instantiated, subscription to external services is made and the nodes start sending the HTTP requests or listening for data from a specified port and capture data from local sensors or files (Blackstock, 2014) (Node-Red, 2016a). The node receives data either from an external service, or received from an upstream node via its "input" handler and as soon as it finishes processing the data it calls the base class Node JavaScript function send() to send named events or messages to the single threaded Node.js event loop for execution. The input event will be triggered as soon as the node receives a message and it makes use of the input handler function that handles the entire input event. The event loop processes these messages using the downstream Node instances that can process data, generate additional events, or communicate with outside services, local hardware. Node-Red can easily integrate different controllers and devices that communicate over the internet using TCP/IP protocols (Blackstock, 2014) (Node-Red, 2016a).

Figure 2 in the appendix section below indicates the data flow in Node-Red.

3.6.1 Nodes

The nodes in Node Red are developed using both HTML and JavaScript. The HTML script defines how the node appears in the editor in terms of design and the number of input or output points it should contain. JavaScript drives the nodes' internal functionalities.

Node Red framework provides a wide range of pre-installed nodes but also more additional nodes are available to install from the Editor pane which can be accessed through the **Manage Palette** option from the menu (top right), and then selecting the install tab in the palette.
These nodes are available from both the Node-RED project as well as the wider community. According to (Node-Red, 2016a), Node Red provides a search functionality that allow users to search for available nodes that they intend to install, update, and enable and disable existing nodes in the Node-RED library (Software-Associates, 2016). Figure 3 in the appendix section below indicates the three major sections for the Node Red platform labelled as:

A - Nodes Palette This forms the basis for creating flows and provides a user with a variety of nodes that can be wired together. This also serves as a Pallet Manager and at the same time serve as a catalogue where the users can browse for all available nodes and install any one of them without having to restart Node-RED (Node-Red, 2016a) (Node-Red, 2016b) (Node-Red, 2017).

B - Workspace This allows the user to create a flow by simply dragging nodes into the workspace and wiring them together. This section is capable of handling multiple tabs which may contain different flows (Node-Red, 2016a) (Node-Red, 2016b) (Node-Red, 2017).

C - Side Bar This is the Info/Debug/Configuration pane located on the right hand side of the window. This section displays information regarding a selected node and also its configuration information. The Side Bar also plays a role as a debug pane which displays or outputs messages from a debug node (Aniruddha-Chakrabarti, 2016).

3.6.2 Classification of nodes in Node Red

Nodes in Node Red can be classified into three different types namely (Aniruddha-Chakrabarti, 2016): **(a) Input nodes** - These are nodes that are specially designed to provide input data to the Node Red application. These can be identified as having

a little grey box on the right-hand side of the node. **(b) Process nodes** - The process nodes are identified as having a little grey box on either side of the node designed to enable the node to input a set of data and then output into another node. **(c) Output nodes** - These nodes have one grey box only on the left side. They are responsible for the output of messages. Figure 4 in the appendix section below depicts the different types of nodes.

3.6.3 Node-Red Flow

In Node Red, nodes that have their independent work cycle are described as a flow. This simply means that these nodes can also run properly in parallel with other flows. Flows allow nodes to communicate and share data without having a connection between them. Within a flow, each node has a different functionality (Software-Associates, 2016). This is indicated in the Figure 5, in the appendix section below.

3.6.4 Benefits of Node-Red

Node Red has proven to be one of the best tools available that helps people with less or no experience in programming to easily understand the programming logic of their Internet of Things (IoT) application and be able to get back to coding as nodes in Node Red provide a simple code structure. In other words, there is no need to write code from scratch as Node Red does it for you. In some cases, Node Red (IoT) applications can be used either in the production environment by developers and the logic behind any intended IoT application can enable developers design or write their own code from scratch since this framework enables them to break down a complex technology related problem into a logical sequence of steps that can be worked out using Node Red (Julio-Fernandez, 2016).

Chapter 4

Application Design and Implementation

Considering the nature of this project, the system architecture comprises a number of different components that are tied together to form an overall temperature and humidity monitoring system. This section provides an internal logic of each of the modules identified during system design.
The diagram in Figure 6 in the appendix section illustrates how different modules such as temperature/ humidity sensors, micro-controllers and application frameworks are tied together form a complete system.

4.1 DHT11 Sensor

The DHT11 is considered as a low cost digital temperature and relative humidity sensor that is capable of transmitting data to Arduino through a single signal wire. DHT11 sensors consist of both the capacitive humidity sensor and a thermistor that allow them to measure the surrounding air. These sensors produce data streams every second, hence it is deemed to be good for relative humidity readings between 20-80 % with 5 % accuracy and good for 0-50°C temperature readings with ±2°C accuracy (Adafruit, 2015). Figure 7, in the appendix section indicates the DHT11 sensor.

4.1.1 How DHT11 Measures Temperature

The DHT11 sensor has in-built thermistor also known as Negative Temperature Coefficient (NTC) temperature sensor designed to measure temperature. These NTC thermistors are fitted with a variable resistor that decreases resistance with an increase in temperature, and are suitable for measuring temperatures ranging between -55°C and 200°C (Nebojsa, 2012).

4.1.2 How DHT11 Measures Relative Humidity

According to (ePro, 2016), Relative Humidity is described as the ratio of the amount of water vapour the air is currently holding and this is expressed as percentage of the amount it would hold if it were saturated. Humidity has a significant impact on both indoor and outdoor environments. Proper understanding of humidity can help us determine how best we can interact with the environment in which we live in. The DHT11 is designed to detect the relative humidity by measuring the electrical resistance between two electrodes. Air moisture gathers on the surface electrodes and causes changes in the voltage levels between the two electrodes. The sensor takes into account the surrounding air temperature when converting the voltage change levels into a digital measurement of the air's relative humidity. When relative humidity is lower, the resistance between the electrodes increases and any sudden increase in humidity will cause the resistance between the electrodes to decrease (Adafruit, 2015).

4.2 Arduino Uno

The Arduino Uno is a micro-controller board that is based on the ATmega328. This can be powered via a USB connection or with an external power supply i.e. AC-to-DC adapter or battery power. It has 14 digital input/output pins: 6 pins are used as Pulse Width Modulation (PWM) outputs, another 6 as analog input pins, 1 as 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button (Arduino-Uno, 2017). One of the unique features that make Arduino Uno different from all other preceding Arduino boards is the Atmega16U2 (a high-performance, low-power Microchip that is designed to act like bridge between the

computer's USB port and the main processor's serial port)programmed as a USB-to-serial converter (Arduino, 2016). This is indicated in Figure 8, in the appendix section below.

4.3 Arduino Software (IDE)

Arduino Software (IDE) is an open source software environment written in Java, designed to allow users to efficiently write code and upload it to any Arduino board in an easy way. It preconfigured to run on Windows, Mac OS X, and Linux systems. The desktop IDE is ideal when one is working offline. According to (Arduino-Uno, 2017), Online IDE (Arduino Web Editor) requires a reliable internet connection and allows the user to write code and have their sketches saved in the cloud making it easier to access them from any device anywhere. Furthermore, using the Arduino Web Editor guarantees that the user will always have the most up-to-date version of the IDE without the need to install updates or community generated libraries (Arduino-Uno, 2017). This is indicated in Figure 9, in the appendix section below.

4.4 Node-Red Flow Layout

This section describes the node red flow layout in the application. This is where different nodes that perform different functions are wired together thereby indicating the flow of data within the application framework.

4.4.1 COM3 Node

This node is responsible for reading data from a local serial port, also known as Serial In. It is configured to wait for a "split" character. It is also set to wait for a timeout in milliseconds for the first character that it received. As it receives the characters, it waits to ensure that it fills a fixed sized buffer. After filling its buffer size, it then outputs the msg.payload as either a UTF8 ASCII string or a binary Buffer object. According to (Node-Red, 2016b), this node checks to see if no split character is specified, or a timeout or buffer size of 0, and sends a stream of single characters again as either ASCII characters or binary buffers that are of size 1.

4.4.2 Filter Values Node

This node allows you to write code that will work with other nodes that are connected to it. It takes in a JavaScript object called **msg** as its input message thereby allowing each message to be passed though a JavaScript function. In this case, the "**Filter Values**" node is responsible for taking in the message (input data) through its **msg.payload** property and then return/send a single message object that is passed to nodes connected its first output. It also passes its output message to nodes connected to the corresponding outputs as an array of message objects. This simply means that if any of the elements of the array are itself an array of messages, multiple messages are sent to the corresponding output. According to (Node-Red, 2016b), this node cannot send/ pass on messages if an element of the array returned null or itself is returning null. The following script filters and splits an array of message objects into an array of temperature values and also another array of humidity values (Node-Red, 2016b).

```
var m = /(\d+\.\d+) .* (\d+\.\d+)/.exec(msg.payload);
var temp = m[1];
var humid = m[2];
return [{payload: temp},{payload:humid}];
```

4.4.3 Temperature Node

This is also a function node as above but it is responsible for setting a threshold value for temperature, which is to **26°C**. If temperature recording rises above the set threshold, an event is triggered in the form of a twitter alert message indicating that temperature is above normal.

```
var newMsg = msg.payload;
var temp = parseInt(newMsg);
if (temp > 26)
 {
   msg.payload = "NUIG Insight Centre room temperature is
    "+ temp + "°C above than normal...!";
```

```
    return msg;
  }
```

4.4.4 Humidity Node

This is also a function node as above responsible for setting a threshold value for humidity, which is set to **48%**. If humidity recording rises above the set threshold, then this triggers an event in a form of twitter message alert indicating that humidity is above normal.

```
var newMsg = msg.payload;
var humidity = parseInt(newMsg);
if (humidity > 48)
 {
   msg.payload = "NUIG Insight Centre room humidity is"
    + humidity +"% above than normal...!";
    return msg;
 }
```

4.4.5 Debug (msg.payload) Node

The Debug node can be connected to the output of any node. This node uses the debug tab of the sidebar (that can be accessed under the options drop-down in the top right corner) to display the output of any message. By default, it is set to display the **msg.payload**. Each output message will also display the timestamp and the topic of the message (msg.topic). The debug node also has a button to its right that is designed to toggle its output on and off when pressed and this helps de-clutter the debug window. (Node-Red, 2016b) indicates that selecting any particular message will highlight (in red) the debug node that reported it. This is useful if you wire up multiple debug nodes.

4.4.6 Twitter Node

A Twitter out node is designed and configured to tweet the msg.payload. The node takes in a Twitter ID, which is used to log on to a twitter account. A tweet will be

pushed every time when either temperature or humidity recordings rise above the set threshold (Software-Associates, 2016).

4.4.7 Chart/ Visualization Nodes

Node-Red dashboard is capable of visualizing live data streams in various forms as indicated below: **(a) UI-Chart** - this makes use of a Chart.js library. You can avail of four different types of charts, for example, line, bar and pie chart nodes where the X-Axis labels can be customized using a date formatter string. These nodes are capable of converted each input msg.payload value to a number and if the conversion fails, then that specific message is ignored.

(b) Gauge - This visual comprise of 4 nodes namely: standard/ simple gauge, donut (complete 360°) gauge, compass gauge and wave gauge. With standard and donut gauges, you can easily specify the colour range that can be applied to them (Node-Red, 2016b). This is indicated in Figure 10, in the appendix section.

Chapter 5

Output Results and Visualization

5.1 Output Results

This section highlights the output results that were obtained after the application was run in one of the rooms in a smart building. These results of the experiment are analysed and displayed as visual charts/ graphs and also as tweets. The subsections below illustrate the detailed output results.

5.1.1 Data Visualisation

The visual charts indicated below display temperature as well as humidity measurements recorded within a 5 day period.

(a) **UI Chart.** Both the temperature and humidity (input) values were plotted on a chart in the form of a time based line chart. As data is pushed into Node Red UI chart, the graph is able to auto-scale to any values it received. The X axis defines a time window or a maximum number of points to be displayed. As time goes, older data is automatically removed from the graph.

(b) **Gauge** - Both the temperature and humidity (input) values are displayed in form of a gauge as indicated in Figure 11(a,b,c,d) in the appendix section below.

5.1.2 Increased Room Temperature

Here, we indicate the visual analysis of the experiment where we adjusted the room temperature upwards with an intention of assessing whether the sensor (weather station) was able to record the increased temperature and humidity readings and

then trigger an event such as sending a twitter alert message whenever temperature and humidity readings rise above a set threshold or not. We found that the sensors were able to automatically adjust to any slight changes in both temperature and humidity. The results were successful obtained - where see a sharp rise from the temperature graph and a drop in humidity as indicated in Figure 12(a,b) in the appendix section below.

5.1.3 Twitter Messages

This application is also configured to send out temperature and humidity readings as tweet messages through a Twitter API. A twitter node in this case is configured to send out temperature and humidity readings only if they rose above a set threshold. For example, both temperature and humidity thresholds are set to **26°C** and **48%** respectively.
Figure 13 in the appendix section below indicates a tweet stream with both temperature and humidity rising above the set threshold.

5.2 Conclusion of Results

The graphical representation of these results is indicated in Figure 11(a,b,c,d) in the appendix section below.
The sensor (weather station) was set to run for duration of 5 days - day and night. According to the results that were obtained after implementing the temperature/ humidity monitoring system, it was discovered that during the night, temperature values were between 21°C and 22°C which was below the set threshold of 26°C and this did not trigger any event or alert message. The corresponding humidity values were between 44% and 50%. Whenever humidity readings rose to above which was above 48%, set threshold, an event was triggered in form of a twitter alert message. The same process was repeated during the day where we also found that temperature readings were slightly higher (temperature at 24°C) as compared to night readings but were still below the threshold. Humidity readings during the day were lower than during the night readings (humidity at 44%) hence no alert message was

triggered.

Furthermore, at the end of the experiment, we decided to adjust the heating system values for a few hours and so increasing the room temperature, as indicated in Figure 12a,b in the appendix section below. It was discovered that the sensors were able to record the new temperature readings which were higher than the set threshold and hence triggered a twitter alert message as indicated in figure 13, in the appendix section below.

From this experiment, I was able to conclude that humidity was higher during the night while temperature dropped. Humidity decreased during the day signalling that air in the room became hotter, thereby holding as much moisture as it can. Temperature readings rose during the day but were still below the threshold.

Chapter 6

Creating Historical Data and Visualization

6.1 Historical/ Archived Data

According to (Business-Intelligence.com, 2017), historical data is referred to as data that was collected in the past and outlines trends in the subject's past. This type of data is stored in non-volatile, secondary storage and can be used to conduct predictive analysis.

6.2 Saving Sensor Data to Excel File Using Arduino Excel 2.1

In this context, historical/ archived data is data that is recorded by the DHT11 temperature and humidity sensor and stored in an excel file. The sensor is able to record both temperature and humidity readings every second and these readings/ values are sent to an excel file using a framework called **Arduino Excel 2.1**, formerly known as former **Arduino Excel Commander**.
This framework was developed by **Roberto Valgolio**.

Arduino Excel 2.1 is an extraordinary Microsoft Excel tool that can be used as an interface to Arduino. It is written in **C** programming language. As its main features, this tool supports data exchanging in both directions, which simply means that it is capable of writing data to a worksheet as well as retrieving from any worksheet. In this case, Microsoft Excel can also be used as datalogger or input source to other application frameworks (Valgolio, 2016).

6.2.1 Integrating Arduino Excel 2.1 to this project

As Arduino Uno is running through a serial port **COM3**,the DHT-11 sensor records both temperature and humidity values every second and sends these readings to Ms Excel worksheet through Arduino Excel interface that is designed to listen to Arduino Uno's the **COM3** serial port.

The application has been configured to send data to Ms Excel worksheet named **Arduino_Excel_21.xls**. I modified the existing script to be able to write **50000** data items to the Excel worksheet/ cells. The owner of the script (Roberto Valgolio) wrote it such that it clears the cells when the data items/ cells exceed 5000. The script is capable of send more than 50000 data items to the cells at any point in time.

6.2.2 Steps to Integrating Arduino Excel 2.1

The following are the steps required in order to integrate Arduino Excel 2.1 into Arduino.

1. Download Arduino_Excel_Setup.exe and launch it as Admnistrator. This will automatically add all the necessary Arduino_Excel libraries to Arduino libraries folder including a sketch folder that contains a sample script **Arduino_Excel_21.ino**.

2. Using the Arduino IDE, open and compile the sketch Arduino_Excel_21.ino in your Arduino projects.

3. Using (Excel) office 2003/2007/2013/ open Arduino_Excel_21.xls. This file is created the moment you run Arduino_Excel_Setup.exe as Admnistrator and this is needed only at first run.

4. Allow **macro execution** while file is opening.

5. In office 2013, open File -> Options -> Protection Center -> Macro settings and check '**Trust access to the VBA project object model**'.

6. While in Excel, press **CTRL + A**. This will run Arduino Excel 2.1 interface which allows you to configure some of the functionalities associated with it, for example: (i) setting up the COM3 serial port that enables the application to

read data from the sensor (ii) setting up the speed at which the application will be able to read from the Arduino IDE and write send/ write to excel worksheet.

7. Finally, click the **Connect** button.

Figure 14 in the Appendix below indicates the excel worksheet that I generated in this project using Arduino Excel 2.1. The complete setup procedure is illustrated in the video in this link (https://www.youtube.com/watch?v=BmYAGVmqguo, 2016)

6.3 Visualizing Historical Data

Data visualization is a general term that describes any effort to help people understand the significance of data by placing it in a visual context. This may involve the use different visualization tools such as **Tableau, Data Hero, R Programming,D3.js** etc.

In this case, I have used HTML and D3.js. to visualize both temperature and humidity historical dataset (Arduino_Excel_21.csv) on a web browser. The dataset was initially created as **Arduino_Excel_21.xls** file format where it was later converted to **Arduino_Excel_21.csv** file format by using an online csv converter.

Figure 15 in the appendix below (Visualizing historical data using HTML and D3.JS Library) indicates how both temperature and humidity historical data values are visualized in a web application in the form of a **linear graph**. The web application has the **index page** that displays the humidity values and the navigation link that redirects you to the **temperature page** that displays temperature linear graph.

6.3.1 D3.js Library

D3 (Data-Driven Documents or D3.js) is a JavaScript library for that is being used by different individuals and stakeholders to manipulate data based documents. It makes use of existing web standards and other frameworks such as SVG, Canvas and HTML to help bring data to life (D3.js, 2016).

6.3.2 Why HTML and D3.js?

In the second semester of my master's course, I undertook data visualization module where I had to write a script in JavaScript combined with D3.js library in order to visualize data from multiple datasets. While working on these assignments, I discovered that D3 puts much emphasis on web standards hence gives you the full capabilities of modern browsers without tying yourself to a proprietary framework. I also learned that D3 gives you the freedom to model and design the visual interface that is right for your data by making use of its powerful visualization components that are included in the D3 library for free. (D3.js, 2016).

Chapter 7

Future work

7.1 Future Work

Based on the results obtained above, I strongly believe that there is more that can be done in order to improve on the way in which Node Red framework processes input data. There's still a question to be answered as to how Node Red can process archived data i.e. inactive data stored in databases and other storage files or devices, as input data into Node Red and be able to visualise this data in Node Red dashboard charts/ graphs and gauges in a similar manner as real-time data is being processed and visualised. Because of time, I was not able to research more on how archived data can be processed as real-time data and be visualised using Node-Red dashboard charts/ graphs and gauges. Furthermore, I will consider integrating an MQTT (MQ Telemetry Transport or Message Queue Telemetry Transport) framework as an IoT broker. This framework is designed to support the concept of 'Publisher' and 'Subscriber' where data collecting devices such as sensors and Raspberry Pi publish messages to a neutral party e.g remote server/ cloud storage where each consumer e.g. another Raspberry Pi in a subscription group gets a message. MQTT brokers are considered one of the best option as they are designed to to run on top of a wide range of networks that run different network protocols e.g. TCP/IP, Bluetooth etc.

Later in my research, discovered that Adafruit IO is incorporates libraries such as Adafruit IO MQTT Broker that make it easy to use.

These are the lines of research that I intend to pursued next.

Chapter 8

Conclusion

8.1 Conclusion

Smart devices connected together provide many exciting possibilities for individuals, businesses and organizations to utilize huge streams of data generated in real-time by this devices and be able to make informed decisions. This process involves performing real-time analytics and visualization of stream data and have valuable insights.

Building application frameworks to support a huge number of inter-connected devices is not an easy task. That's where Node-RED comes into play by providing a means of wiring smart objects together and hope for the future of Internet of Things. This is a powerful tool for building Internet of Things (IoT) flows and can be used along with modern cloud storage components for computing and analytical purposes. It provides a visual metaphor that makes it easy for developers to understand and demonstrate the flow of an API and also how code works and hence provides a basis for building practical IoT applications without having to write code that allows different IoT components to be wired together.

Node-RED provides a dashboard that allows for real-time stream data analytics and visualization using ui_charts and graphs.

Appendix A

Project Screenshots

This chapter includes all the screen-shots as well as source codes (JavaScript) that were in this project based on the above information.

A.1 Screenshots

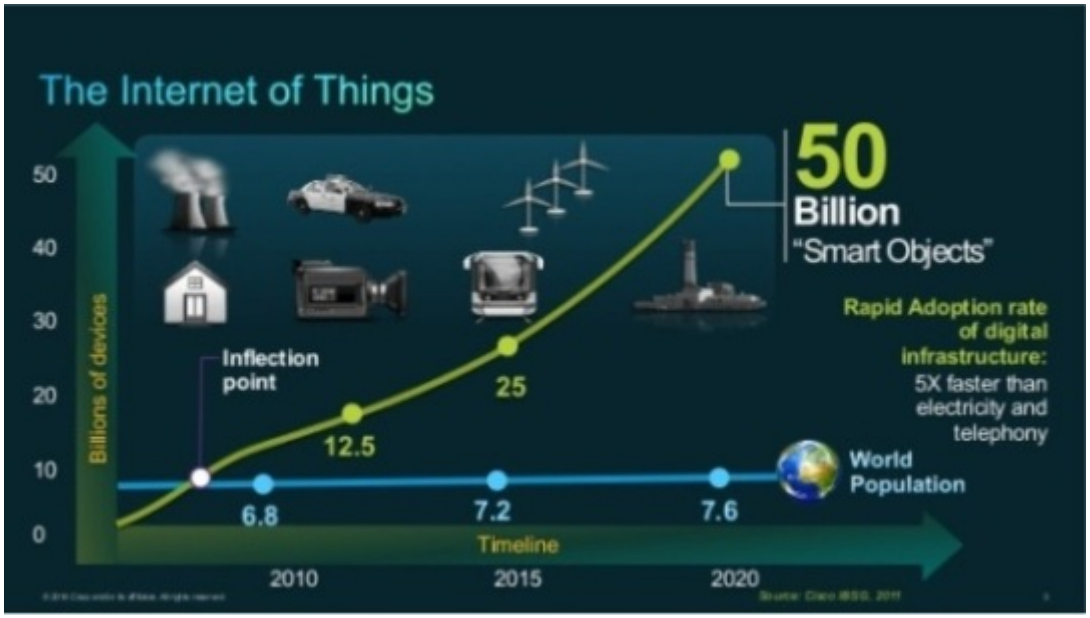


Figure 1: The Internet of Things - Source: Cisco - 2011

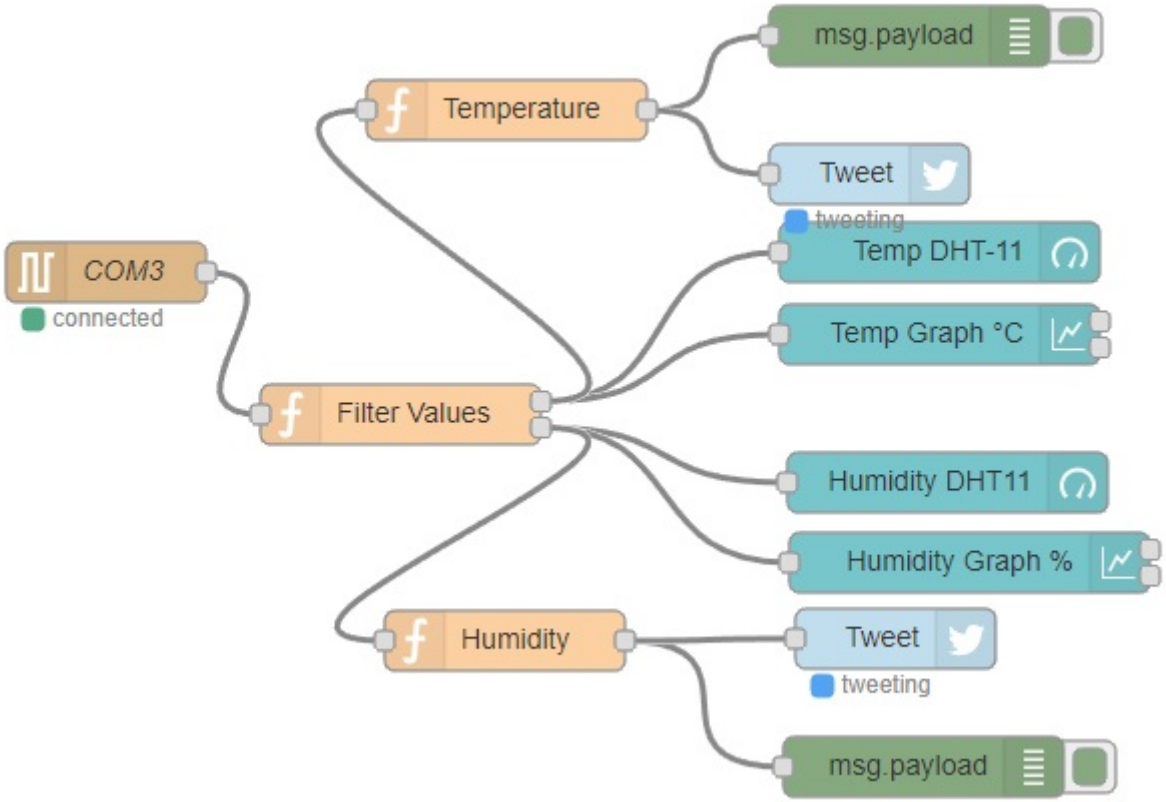


Figure 2: Node Red Data Flow

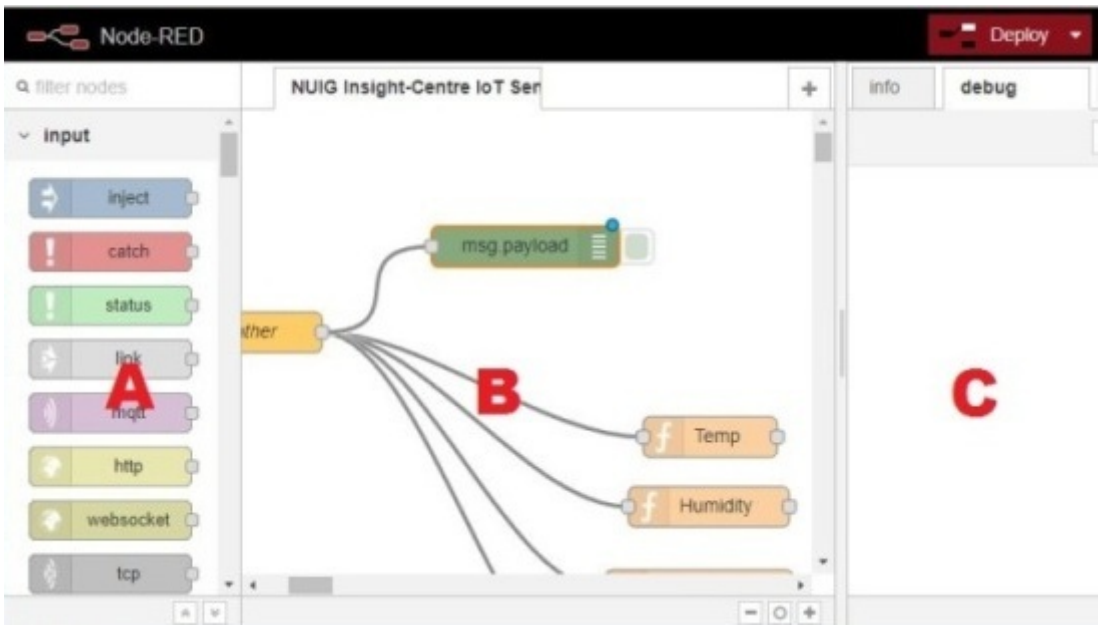


Figure 3: Three major sections in Node Red

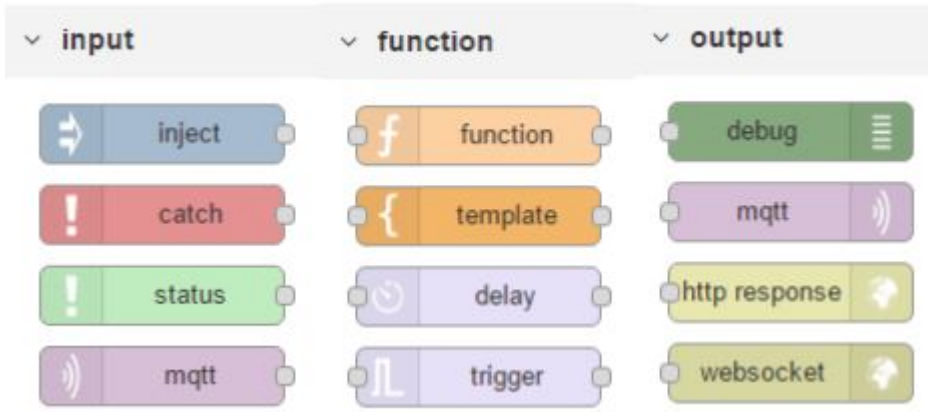


Figure 4: Classification of nodes in Node Red

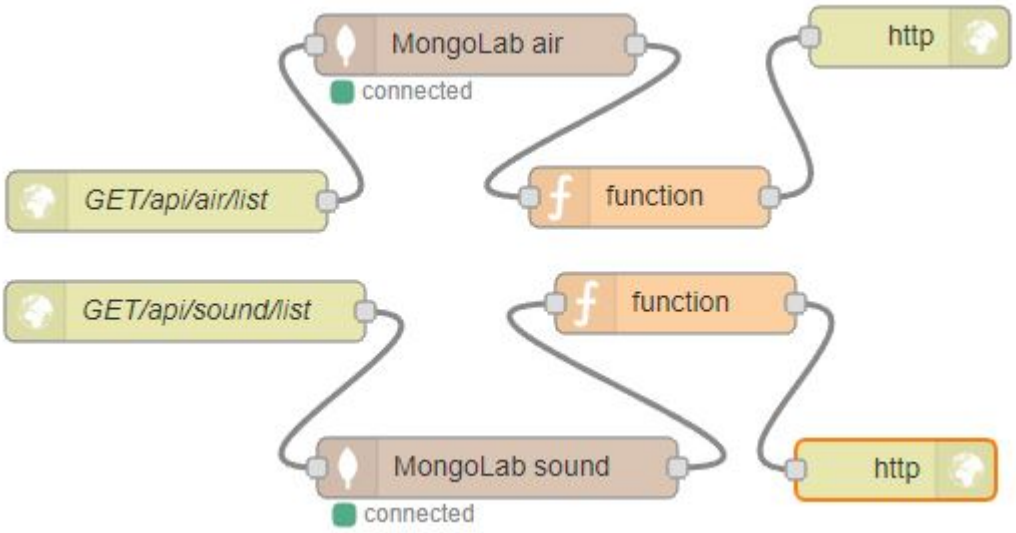


Figure 5: Node Red Flows

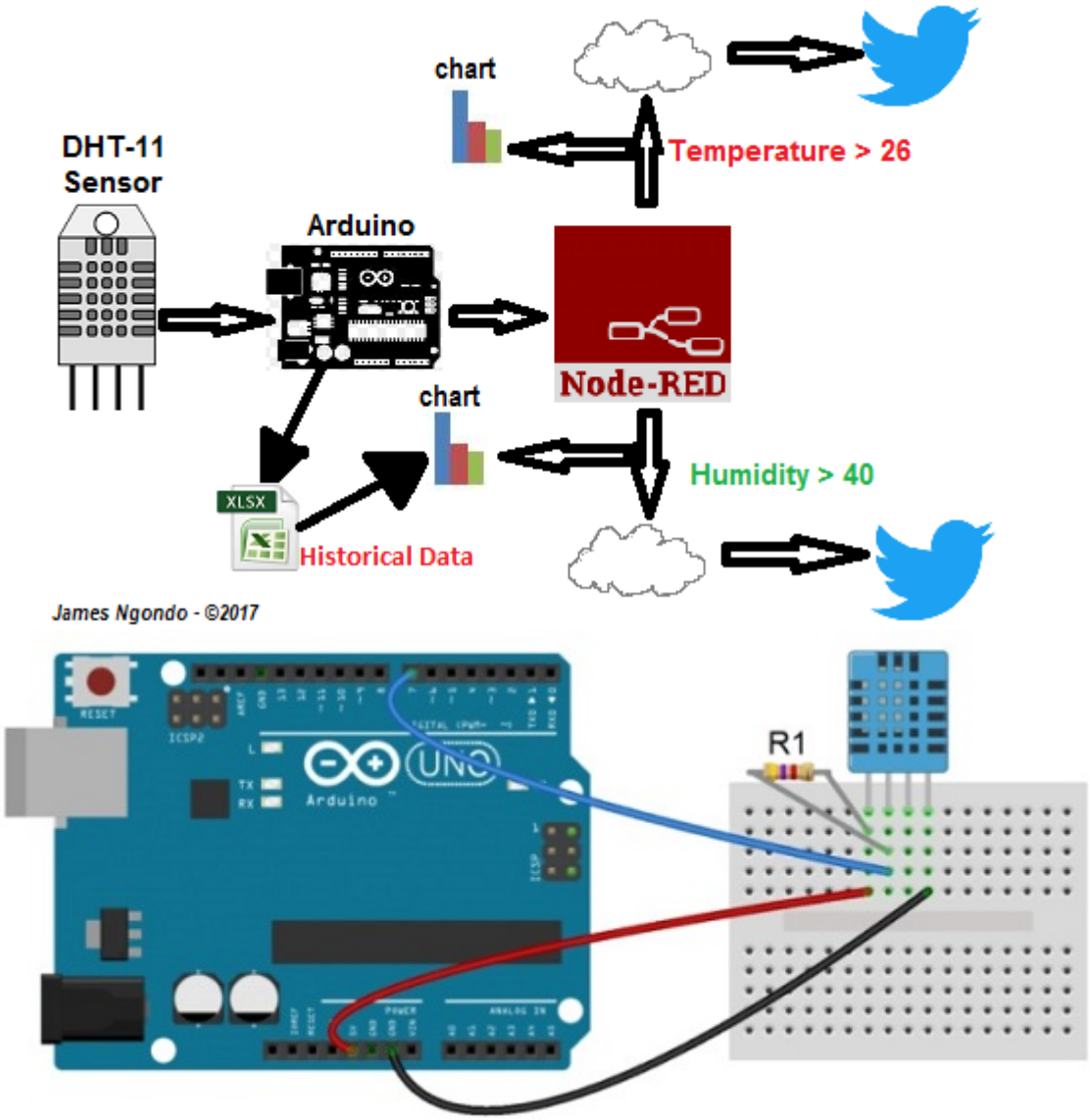


Figure 6: System Architecture

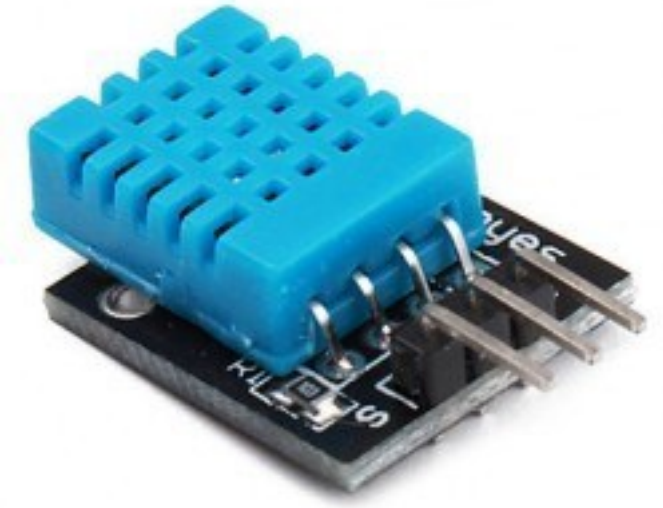


Figure 7: DHT-11 Temperature & Relative Humidity Sensor

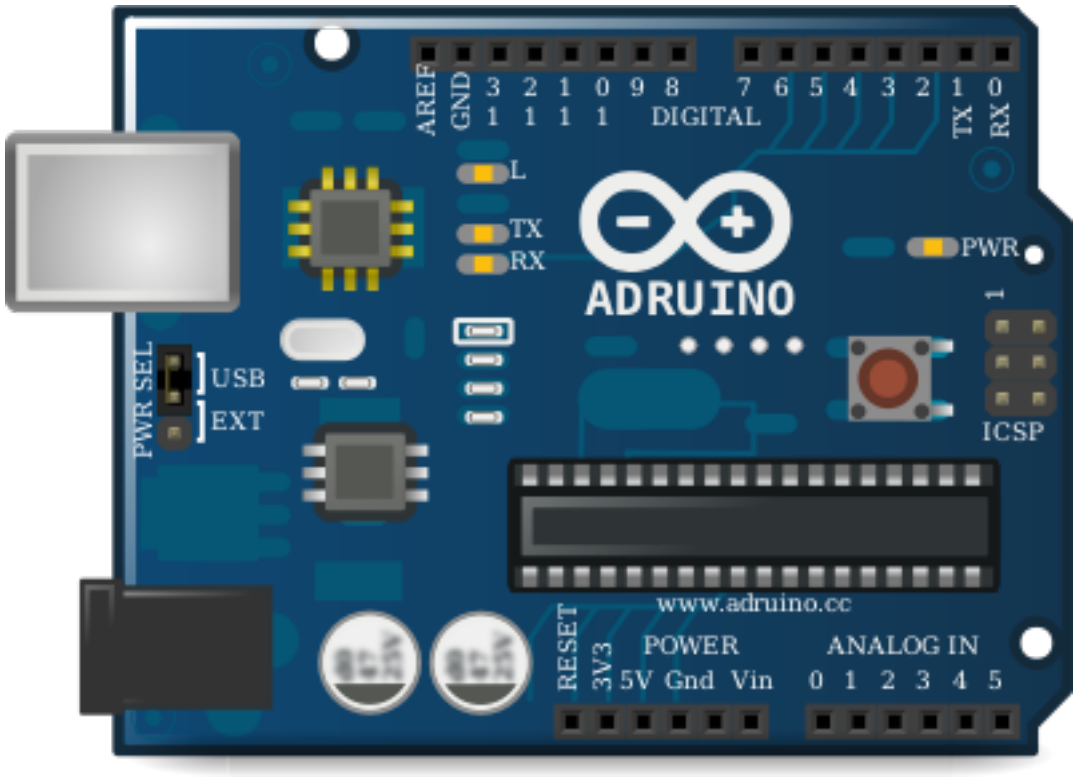


Figure 8: Arduino Uno

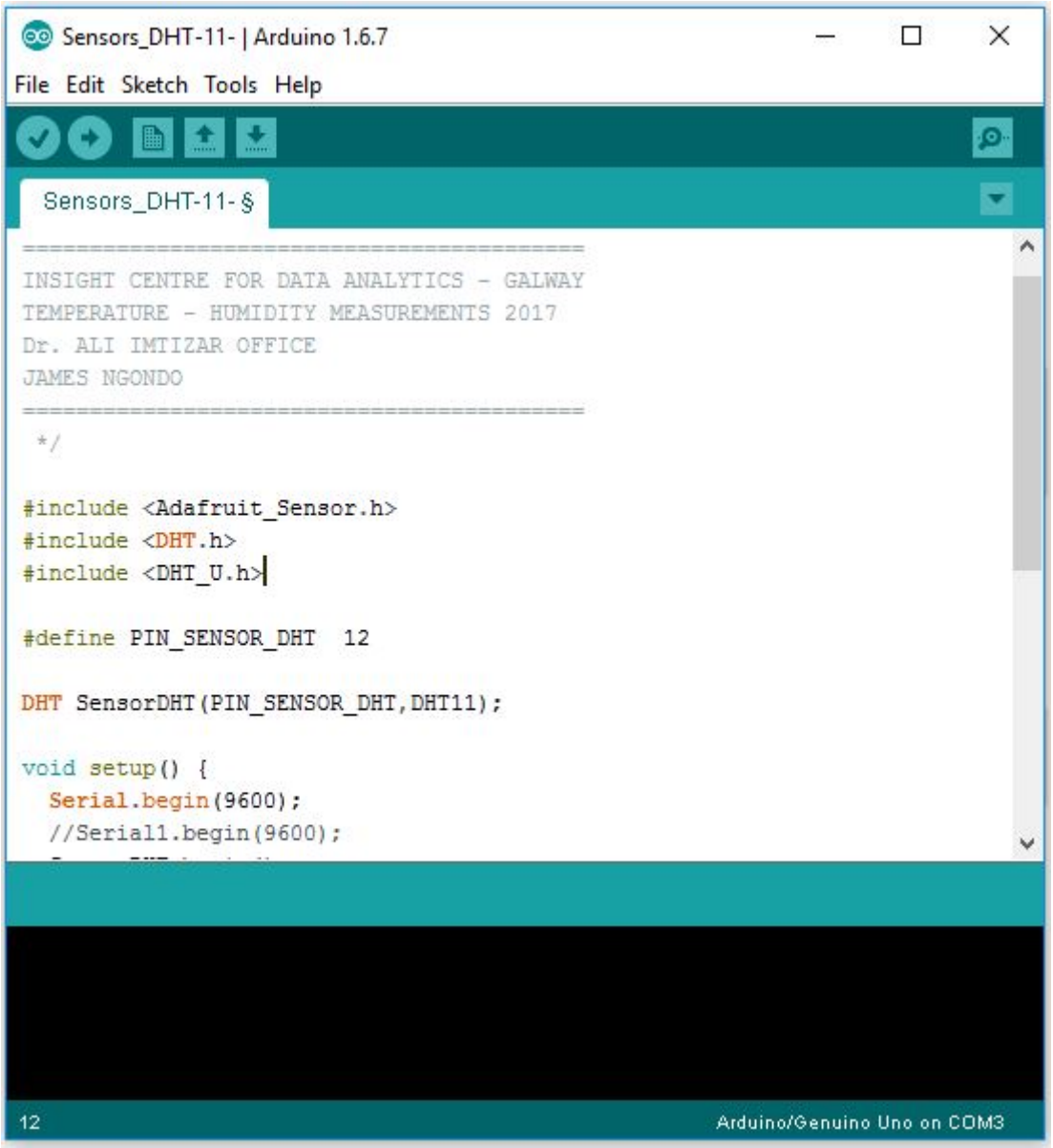


Figure 9: Arduino Software (IDE)

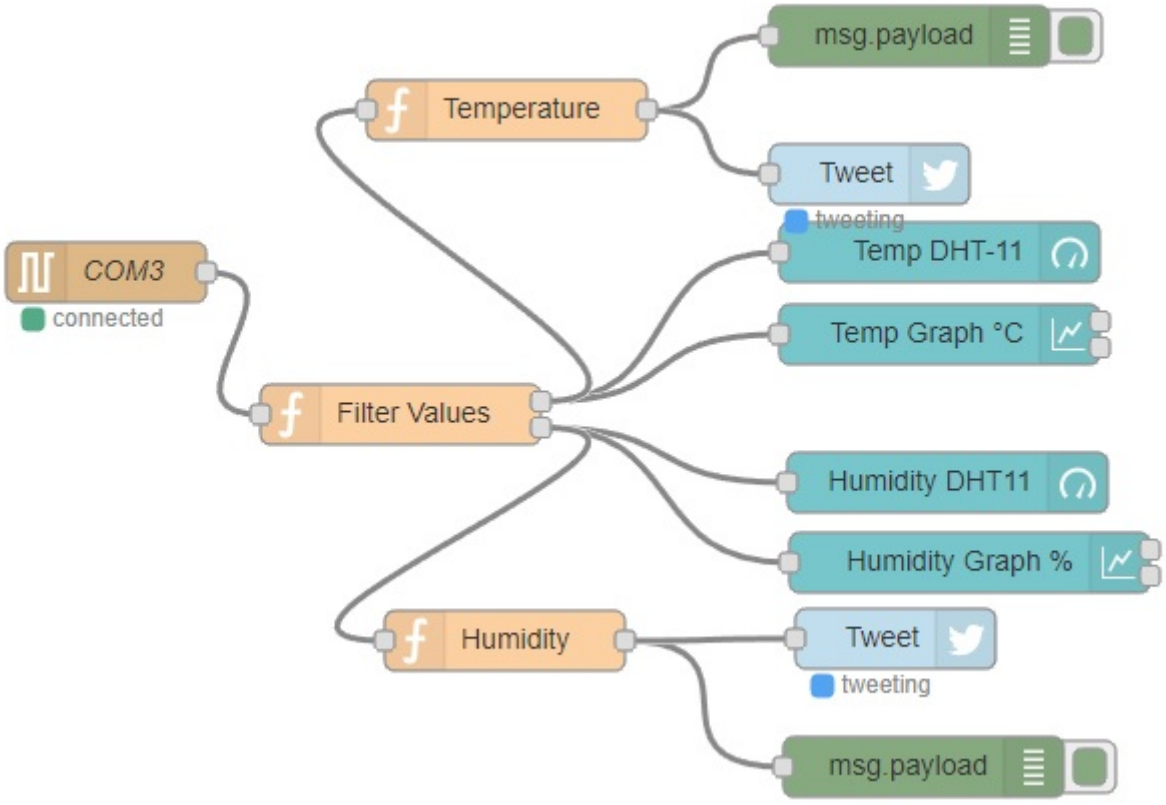


Figure 10: Node Red Flows Implementation

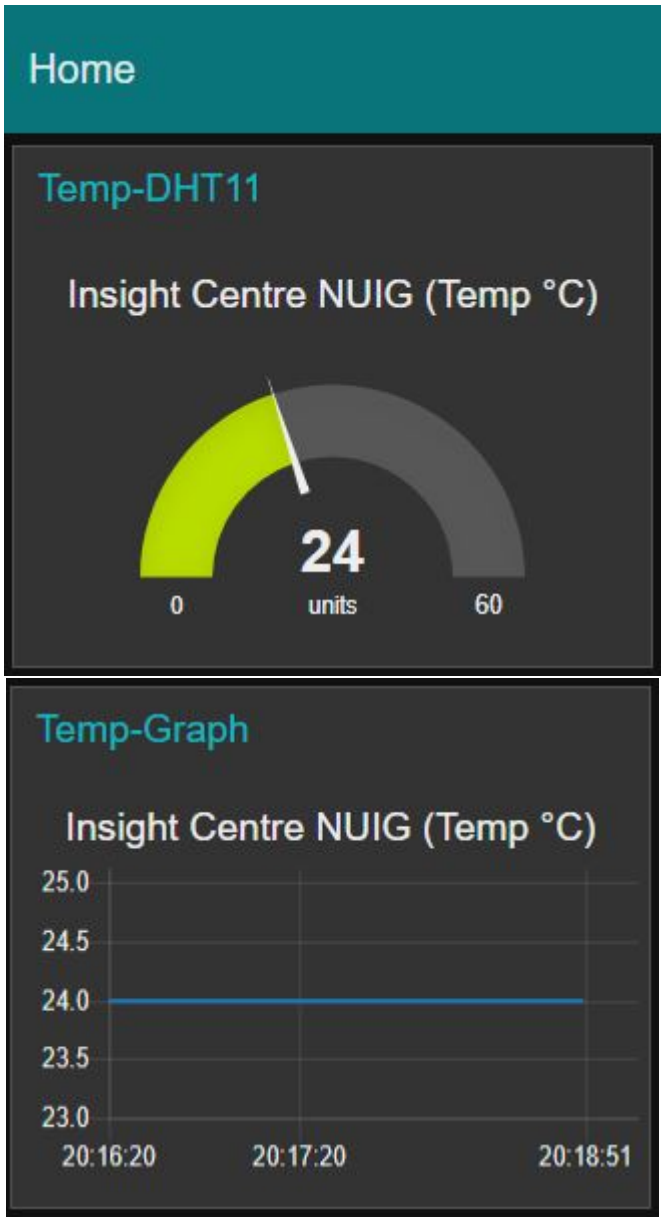


Figure 11a: Node-Red Dashboard Visualization chart - night temperature recording

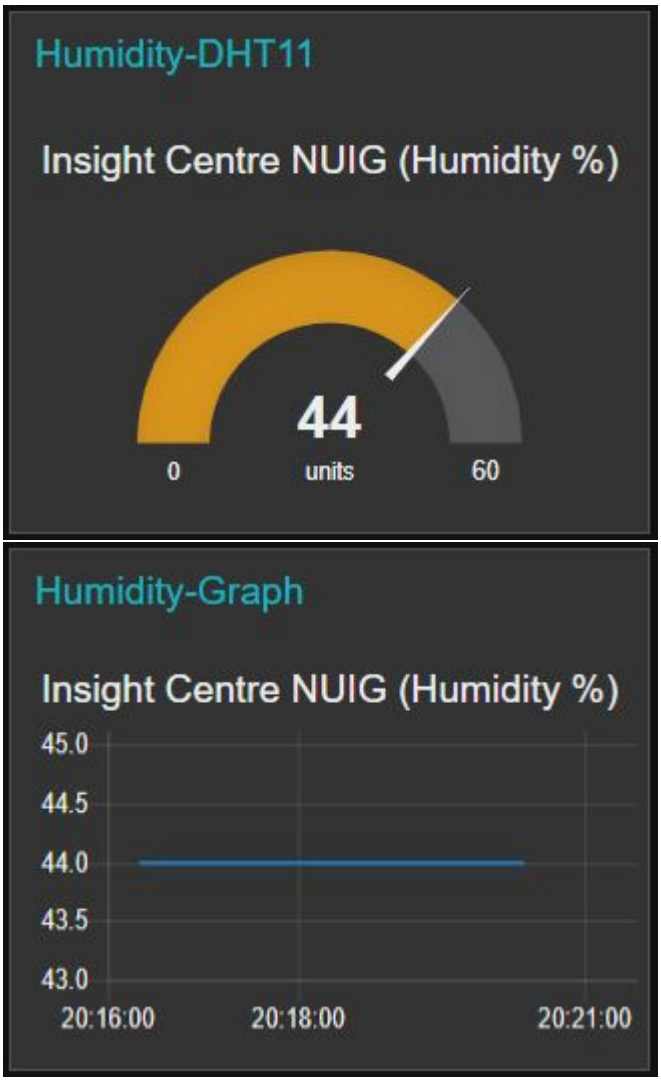


Figure 11b: Node-Red Dashboard Visualization chart - night humidity recording

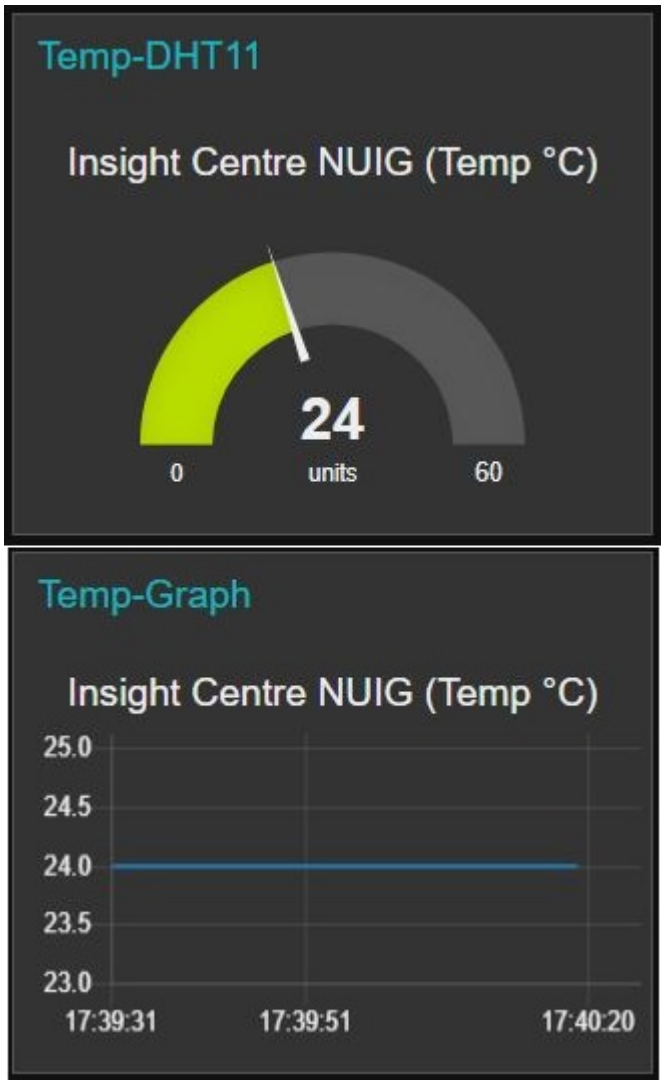


Figure 11c: Node-Red Dashboard Visualization chart - day temperature recording

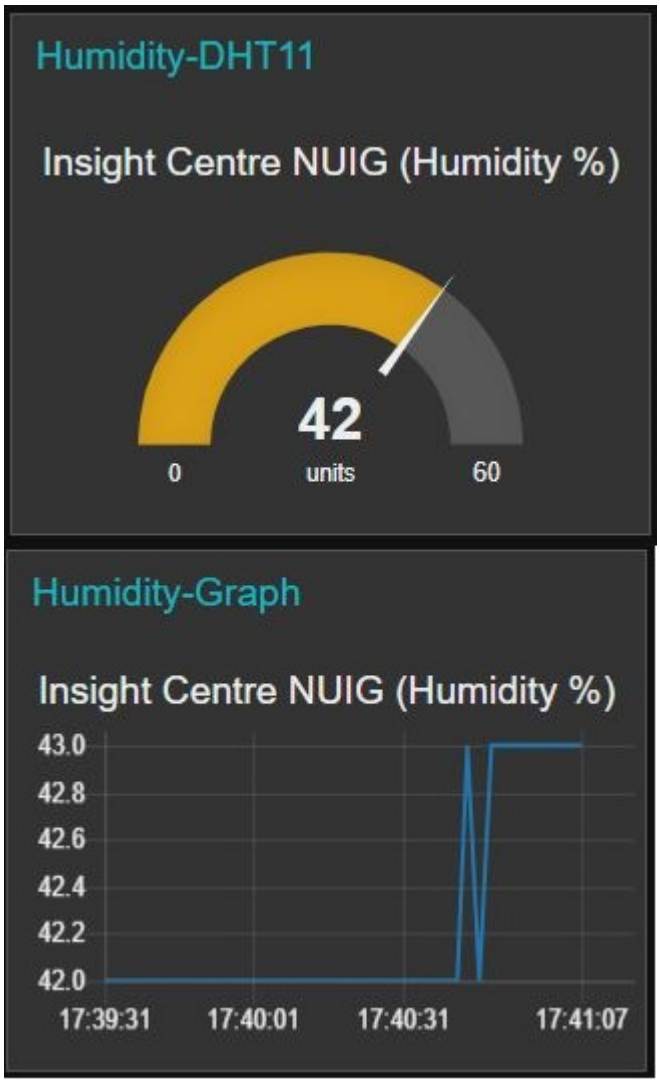


Figure 11d: Node-Red Dashboard Visualization chart - day humidity recording

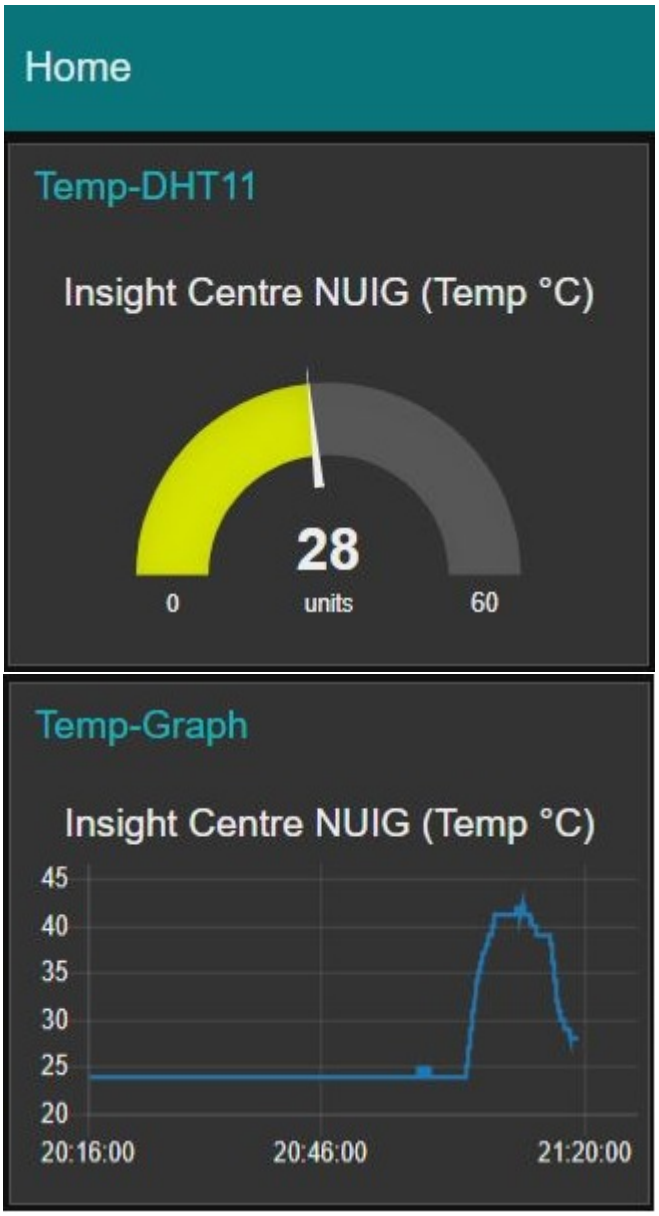


Figure 12a: Node-Red Visualization Chart - room temperature increased when heating system was adjusted upward

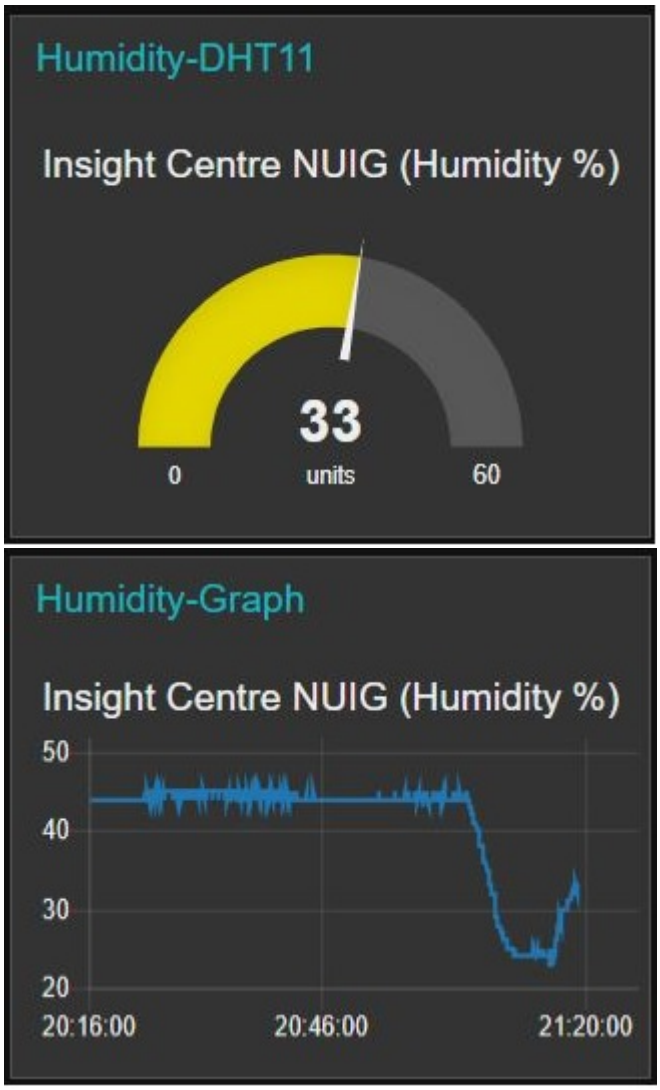


Figure 12b: Node-Red Visualization Chart - room humidity decreased when heating system was adjusted upward

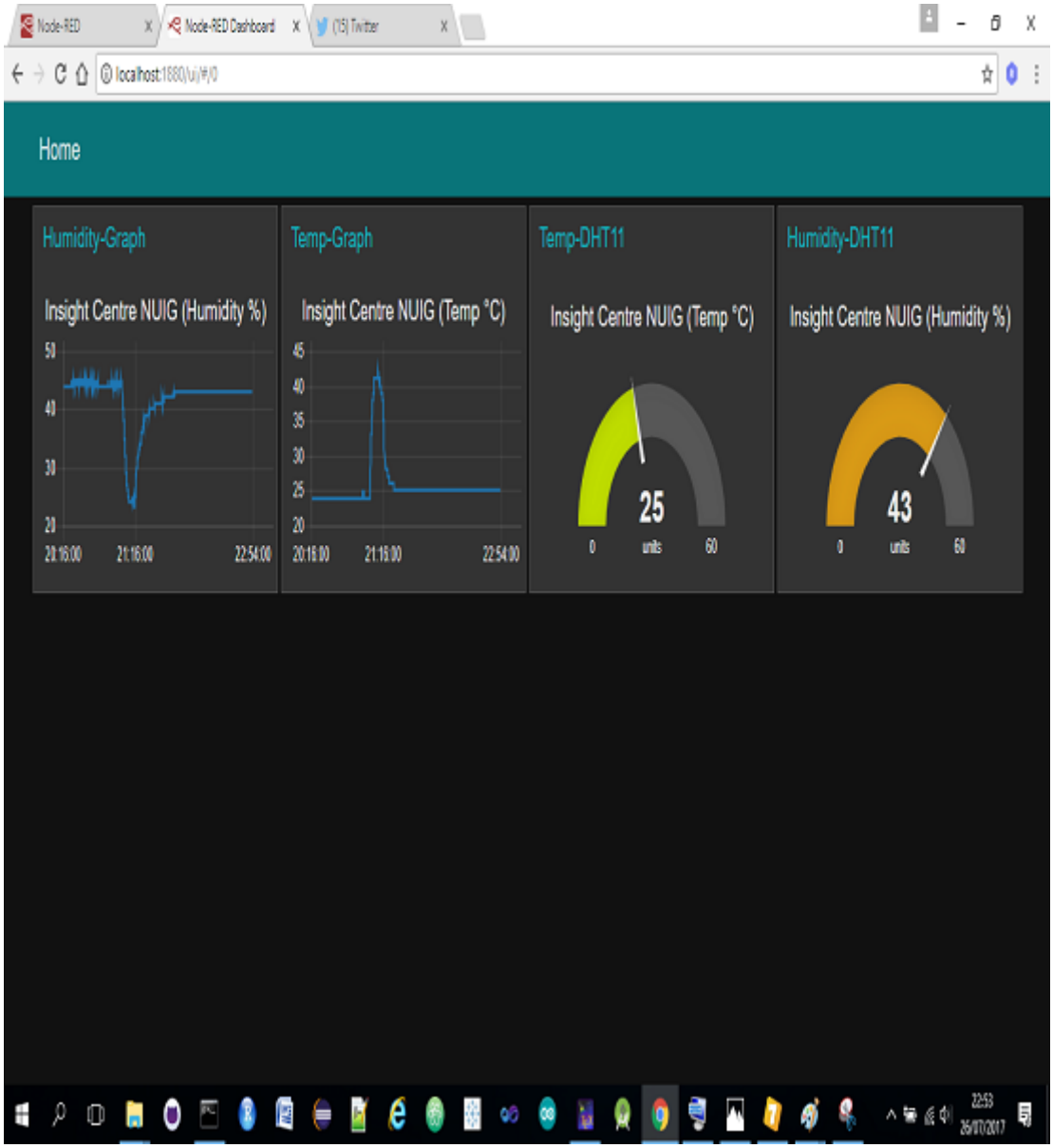

Node-RED
Node-RED Dashboard
(15) Twitter
localhost:1880/ui/#/0
Home
Humidity-Graph
Insight Centre NUIG (Humidity %)
50
40
30
20
20:16:00
21:16:00
22:54:00
Temp-Graph
Insight Centre NUIG (Temp °C)
45
40
35
30
25
20
20:16:00
21:16:00
22:54:00
Temp-DHT11
Insight Centre NUIG (Temp °C)
25
0
units
60
Humidity-DHT11
Insight Centre NUIG (Humidity %)
43
0
units
60
22:53
25/07/2017

Figure 13: Twitter Alert Message

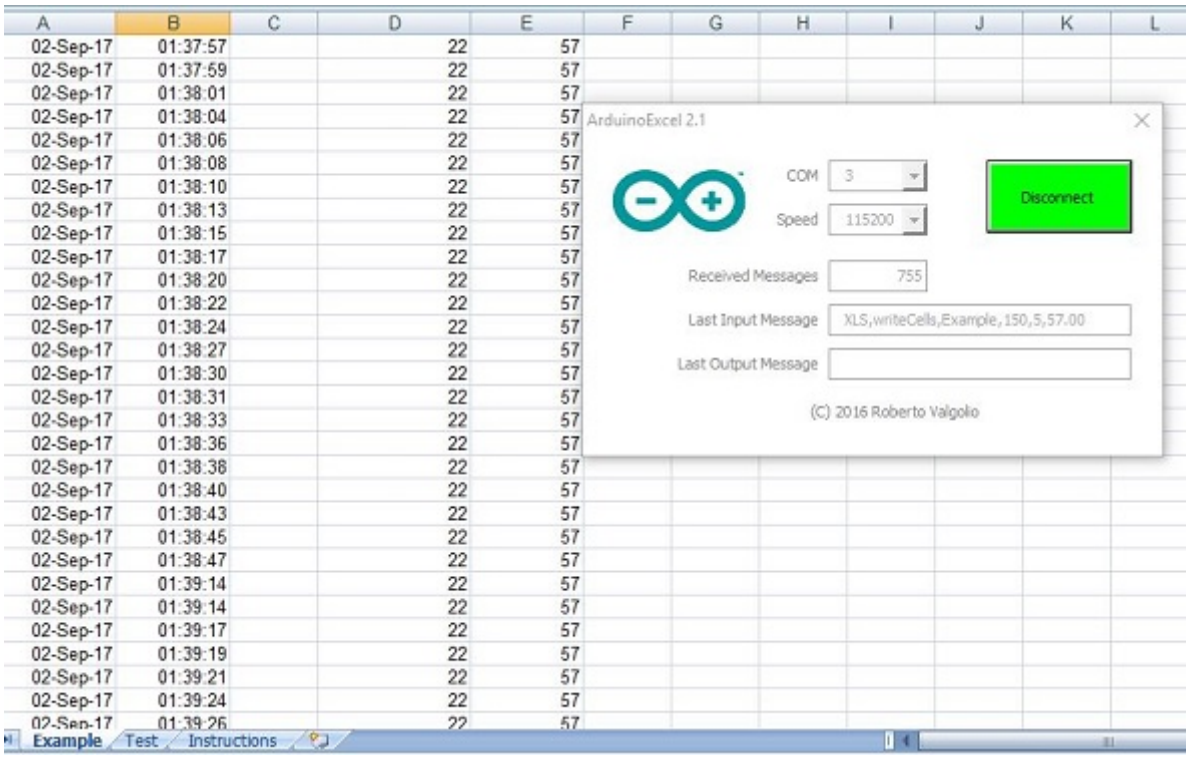
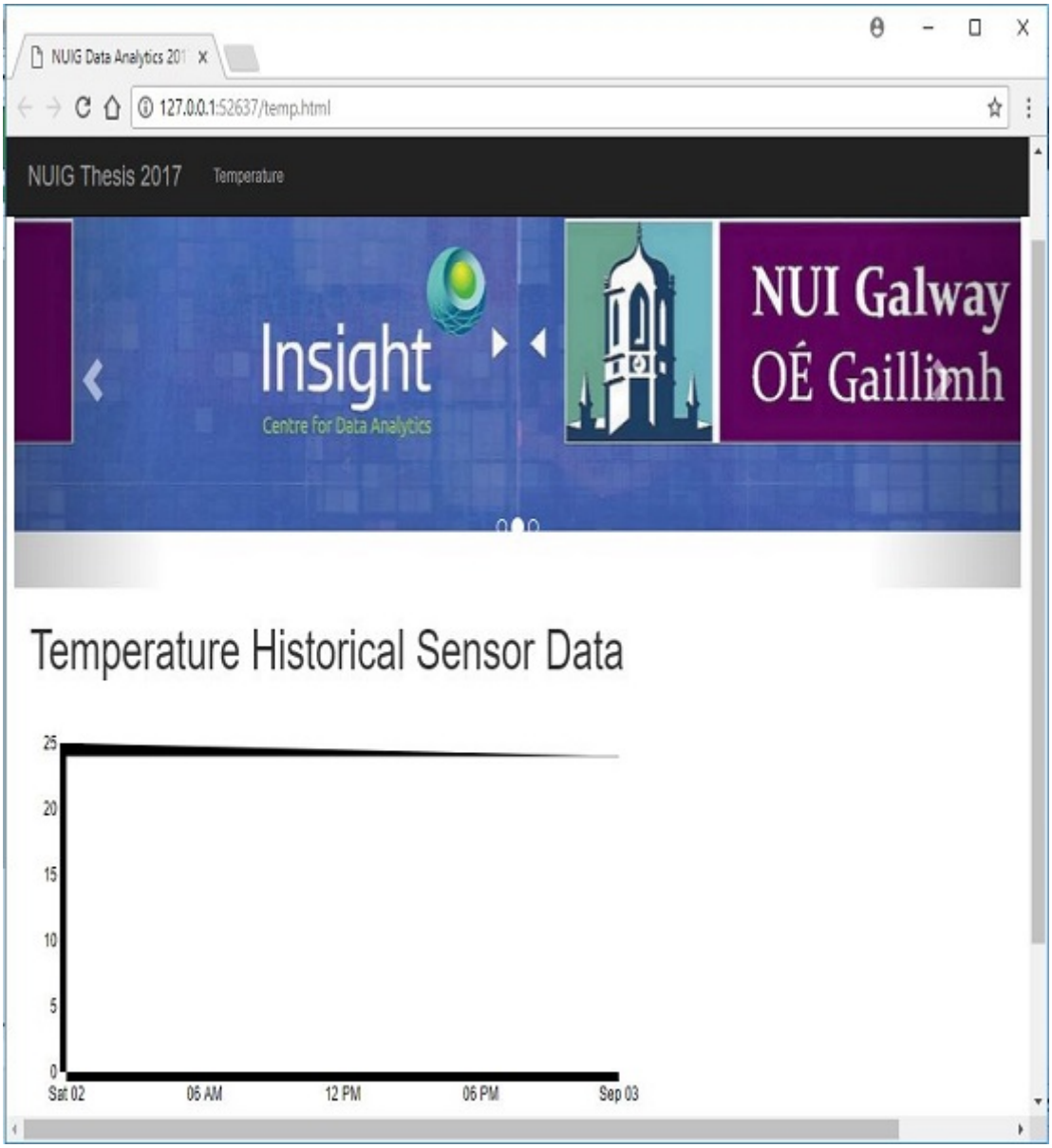


Figure 14: Integrating Arduino Excel 2.1


NUIG Data Analytics 201
127.0.0.1:52637/temp.html
NUIG Thesis 2017
Temperature
Insight
Centre for Data Analytics
NUI Galway
OÉ Gaillimh
Temperature Historical Sensor Data
25
20
15
10
5
0
Sat 02
06 AM
12 PM
06 PM
Sep 03

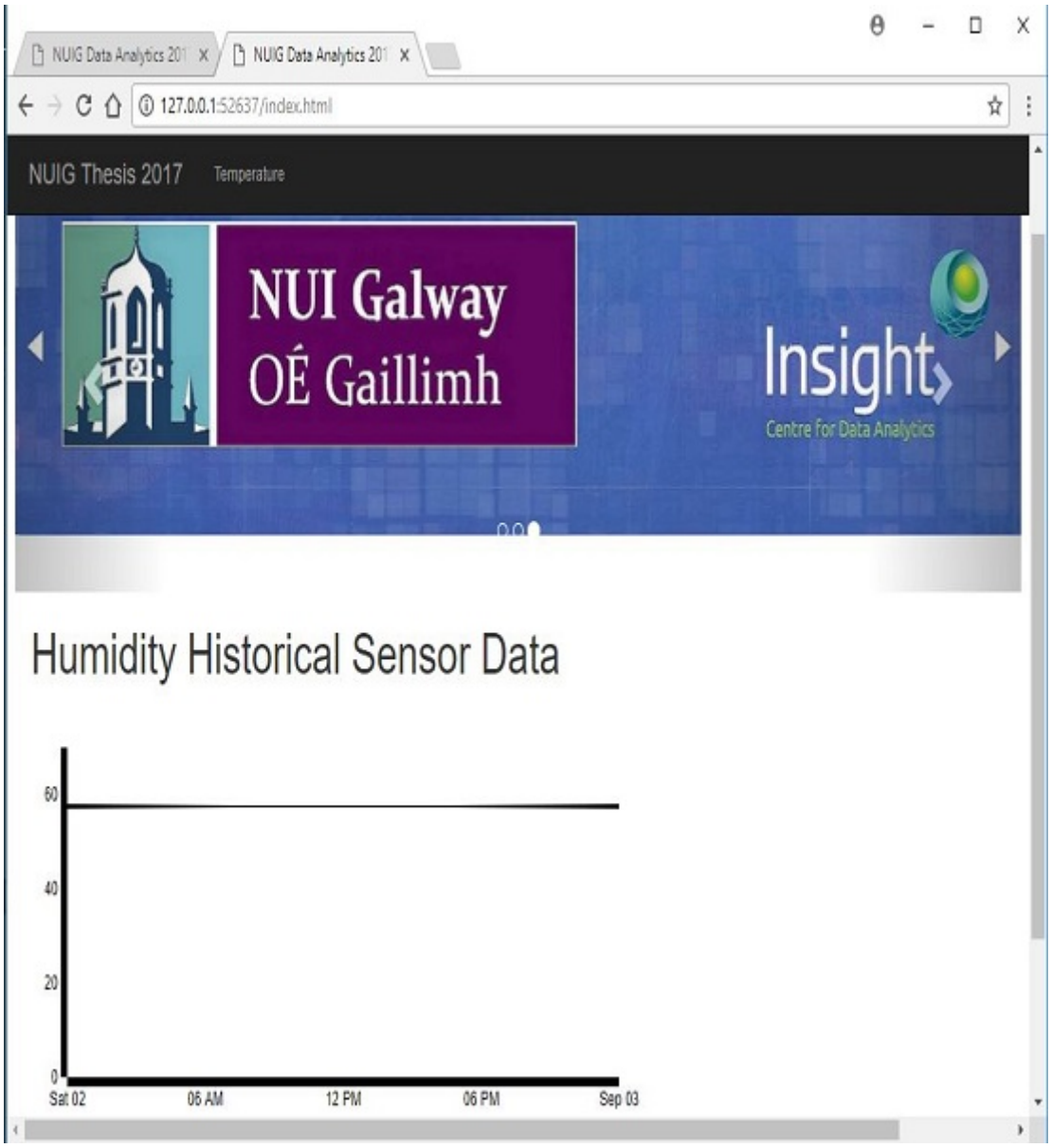


Figure 15: Visualizing historical data using HTML and D3 Library

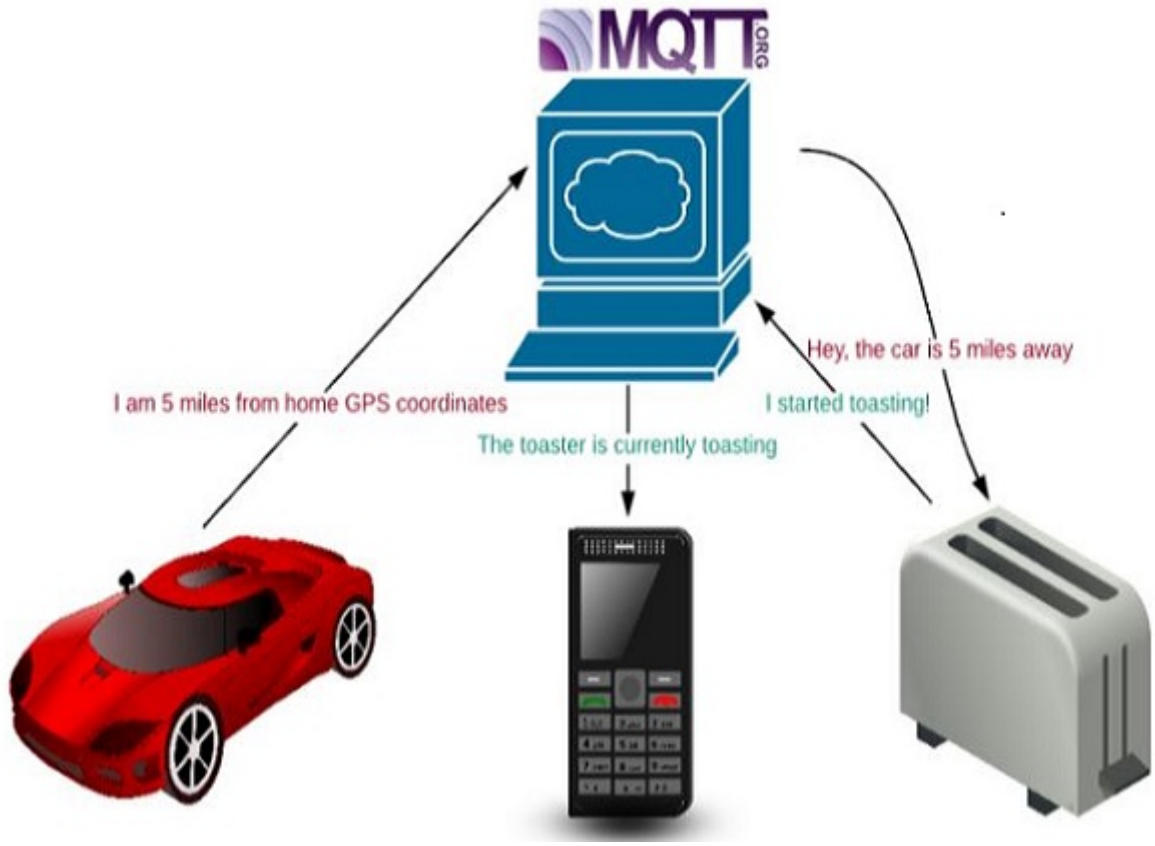


Figure 16: Adafruit IO MQTT Broker

Source: Adafruit IO - 2016

Appendix B

Project Sourcecode

B.1 Arduino Source Code

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#define PIN_SENSOR_DHT  12

 DHT SensorDHT(PIN_SENSOR_DHT,DHT11);

void setup() {
  Serial.begin(9600);
  //Serial1.begin(9600);
  SensorDHT.begin();

}

void loop() {
//Read Temperature and Humidity from DHT11 Sensor
  String Temp_DHT = (String)SensorDHT.readTemperature();
  String Humid_DHT = (String)SensorDHT.readHumidity();

  Serial.print("Temp ");
  Serial.print(Temp_DHT);
```

```
   Serial.print(" ");
  Serial.print("Humidty ");
  Serial.println(Humid_DHT);

  delay(2000);
}
```

B.2 Arduino Excel 21 - Historical Data Source Code

```
/*
==========================================
This code is obtained from Roberto Valgolio
and I have modified it to suite the purposes
of my project through his permission
(roberto.valgolio@gmail.com)
==========================================
 */
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <rExcel.h>

#include<stdio.h>
#include<string.h>
#include<stdlib.h>

#include <rExcel.h>

#define PIN_SENSOR_DHT  12
```

```
long          idx = 0;
int           outputTiming = 1000;
float         a0;
float         rnd;
float         temperature;
float         humidity;
char          value[16];
rExcel        myExcel;

DHT SensorDHT(PIN_SENSOR_DHT,DHT11);

void setup() {
  Serial.begin(9600);
  SensorDHT.begin();
   // rx buffer clearing
  myExcel.clearInput();

 }

void loop() {
   float Temp = (float)SensorDHT.readTemperature();
   float Humid = (float)SensorDHT.readHumidity();
   Serial.print("Temp ");
   Serial.print(Temp);
   Serial.print(" ");
   Serial.print("Humidty ");
   Serial.println(Humid);
  delay(2000);

 // writeXSL();
 // delay(2000);
}
```

```
void writeXSL(){
  static unsigned long  loopTime = 0;
   static unsigned long  time1 = 0;
   int ret;
   loopTime = millis();

   float Temp = (float)SensorDHT.readTemperature();
   float Humid = (float)SensorDHT.readHumidity();

  if ((loopTime - time1) >= outputTiming) {
    time1 = loopTime;
    myExcel.writeIndexed("Example", idx+2, 1,"%date%");
    myExcel.writeIndexed("Example", idx+2, 2,"%time%");
    myExcel.writeIndexed("Example", idx+2, 4, Temp, 2);
    myExcel.writeIndexed("Example", idx+2, 5, Humid, 2);
    idx++;

    if (idx > 50000) {
      myExcel.clear("Example","A11:F70");
      idx = 0;
    }
  }
}
```

B.3 Node-Red Source Code

```
[{"id":"64e68cfd.4266a4","type":"debug","z":"9299071f.290bf8",
"name":"","active":true,"console":"false","complete":"false",
"x":400,"y":1175,"wires":[]},{"id":"8c17686.cf06798","type":
"function","z":"9299071f.290bf8","name":"Temperature","func"
:"var newMsg = msg.payload;\n\nvar temp = parseInt(newMsg);
```

```
\n\nif (temp > 22) {\nmsg.payload =
\"NUIG Insight Centre room temperature is \"+ temp + \"°C
more than normal...!\";\n msg.topic = \"NUIG Insight Centre room
temperature is \"+ temp + \"°C more than normal...!\";\n
return msg;\n}\n\n\n","outputs":1,"noerr":0,
"x":209,"y":1215,"wires":[["64e68cfd.4266a4",
"d44d1e86.c166a"]]},{"id":"8e08aeb2.73a0a","type":"serial in",
"z":"9299071f.290bf8","name":"COM3","serial":"eb309ce3.374",
"x":64, "y":1300,"wires":[["d7b0aec5.4b2f6"]]},{"id":
"d7b0aec5.4b2f6", "type":"function","z":"9299071f.290bf8",
"name":"Filter Values", "func":"\nvar m = / (\\d+\\.\\d+) .*
 (\\d+\\.\\d+)/.exec(msg.payload);\nvar temp = m[1];\nvar
 humid = m[2];\n\nreturn [ { payload: temp },
{ payload:humid } ];","outputs":"2","noerr":0,"x":197,"y":1379,
"wires":[["6c28aeee.f1435",
"79df984b.2ca738","8c17686.cf06798"],["f94da021.683b3",
"16216cb2.7a5083","5f56b82c.c3bab8"]]},{"id":"79df984b.2ca738",
"type":"ui_chart","z":"9299071f.290bf8","name":"Temp Graph °C",
"group":"161d6a02.af8ae6","order":0,"width":0,"height":0,
"label":"Insight Centre NUIG (Temp °C)","chartType":"line",
"legend":"false","xformat":"HH:mm:ss", "interpolate":"linear",
"nodata":"","ymin":"","ymax":"", "removeOlder":"300",
"removeOlderPoints":"","removeOlderUnit":"3600",
"cutout":0,"colors":["#1f77b4","#aec7e8","#ff7f0e","#2ca02c",
"#98df8a", "#d62728","#ff9896","#9467bd","#c5b0d5"],
"x":418,"y":1308,"wires":[[],[]]}, {"id":"6c28aeee.f1435",
"type":"ui_gauge","z":"9299071f.290bf8", "name":"Temp DHT-11",
"group":"4d0abd4f.2b4394","order":0,"width":0,"height":0,
"gtype":"gage", "title":"Insight Centre NUIG (Temp °C)",
"label":"units","format":"{{value}}", "min":0,"max":"60",
"colors":["#00b500", "#e6e600","#ca3838"],"seg1":"","seg2":"",
"x":415,"y":1266,"wires":[]},{"id":"d44d1e86.c166a","type":
```

```
"twitter out","z":"9299071f.290bf8", "twitter":"","name":
"Tweet","x":381,"y":1215,"wires":[]},{"id":"5f56b82c.c3bab8",
"type":"function","z":"9299071f.290bf8",
"name":"Humidity","func":"var newMsg = msg.payload;\n\nvar
 humidity = parseInt(newMsg) ;\n\nif (humidity > 42) {\n
 msg.payload = \"NUIG Insight Centre room humidity is
\"+ humidity +\"% more than normal...!\";\n  msg.topic =
\"NUIG Insight Centre  room humidity is \"+ humidity +
\"% more than normal...!\";\n return msg;\n}\n\n\n",
"outputs":1,"noerr":0,"x":219,"y":1492,"wires":[[
"4e63284c.7f8968", "4d2813a7.3db49c"]]},{"id":
"f94da021.683b3","type":"ui_gauge","z":"9299071f.290bf8",
"name":"Humidity DHT11","group":"39e81f6d.c5554","order":0,
"width":0,"height":0,"gtype":"gage","title":"Insight Centre
NUIG (Humidity %)","label":"units","format":"{{value}}",
"min":0,"max":"60","colors":["#00b500", "#e6e600","#ca3838"],
"seg1":"","seg2":"","x":408,"y":1419,"wires":[]},{"id":
"16216cb2.7a5083","type":"ui_chart","z":"9299071f.290bf8",
"name":"Humidity Graph %","group":"30971655.5ac69a","order":0,
"width":0,"height":0,"label":"Insight Centre NUIG (Humidity %)",
"chartType":"line","legend":"false","xformat":"HH:mm:ss",
"interpolate":"linear","nodata":"","ymin":"","ymax":"",
"removeOlder":"300","removeOlderPoints":"","removeOlderUnit":
"3600","cutout":0,"colors":["#1f77b4","#aec7e8","#ff7f0e",
"#2ca02c","#98df8a","#d62728","#ff9896","#9467bd","#c5b0d5"],
"x":420,"y":1463,"wires":[[],[]]},{"id":"4d2813a7.3db49c",
"type":"twitter out","z":"9299071f.290bf8","twitter":"",
"name":"Tweet","x":382,"y":1511,"wires":[]},{"id":
"4e63284c.7f8968","type":"debug","z":"9299071f.290bf8",
"name":"","active":true,"console":"false","complete":"false",
"x":400,"y":1560,"wires":[]},{"id":"eb309ce3.374","type"
:"serial-port","z":"","serialport":"COM3","serialbaud":"9600"
```

```
,"databits":"8","parity":"none","stopbits":"1","newline":
"\\n","bin":"false","out":"char","addchar":false},{"id":"
161d6a02.af8ae6","type":"ui_group","z":"","name":"Temp-Graph",
"tab":"926ae7fd.e30d08","disp":true,"width":"6"},{"id":
"4d0abd4f.2b4394","type":"ui_group","z":"","name":
"Temp-DHT11","tab":"926ae7fd.e30d08","disp":true,"width"
:"6"},{"id":"39e81f6d.c5554","type":"ui_group","z":"",
"name":"Humidity-DHT11","tab":"926ae7fd.e30d08","disp":
true,"width":"6"},{"id":"30971655.5ac69a","type":"ui_group"
,"z":"","name":"Humidity-Graph","tab":"926ae7fd.e30d08","
disp":true,"width":"6"},{"id":"926ae7fd.e30d08",
"type":"ui_tab","z":"","name":"Home","icon":"dashboard"}]
```


B.4 D3 Jvascript Source Code

```
<script>

    // Set the dimensions of the canvas / graph
    var         margin = {top: 30, right: 20, bottom: 30, left: 50},
        width = 600 - margin.left - margin.right,
        height = 270 - margin.top - margin.bottom;

   // Parse the date / time
   var         parseDate = d3.time.format("%d-%b-%y").parse;

    // Set the ranges
    var         x = d3.time.scale().range([0, width]);
    var         y = d3.scale.linear().range([height, 0]);

    // Define the axes
    var         xAxis = d3.svg.axis().scale(x)
          .orient("bottom").ticks(5);
```

```
 var         yAxis = d3.svg.axis().scale(y)
        .orient("left").ticks(5);

 // Define the line
 var         valueline = d3.svg.line()
        .x(function(d) { return x(d.Date); })
        .y(function(d) { return y(d.Humidity123);});

// Adds the svg canvas
var         svg = d3.select("body")
     .append("svg")
             .attr("width", width + margin.left + margin.right)
             .attr("height", height + margin.top + margin.bottom)
     .append("g")
             .attr("transform", "translate(" + margin.left +
     "," + margin.top + ")");

 // Get the data
d3.csv("Arduino_Excel-02.csv", function(error, data) {
     data.forEach(function(d) {
             d.Date = parseDate(d.Date);
             d.Humidity123 = +d.Humidity123;

     console.log(d.Date);
     console.log(d.Humidity123);
     });

 //console.log(data);
     // Scale the range of the data
     x.domain(d3.extent(data, function(d)
 { return d.Date; }));
```

```
        y.domain([0, d3.max(data, function(d)
    { return d.Humidity; })]);

        // Add the valueline path.
        svg.append("path")
                .attr("class", "line")
                .attr("d", valueline(data));

        // Add the X Axis
        svg.append("g")
                .attr("class","x axis")
                .attr("transform","translate(0,"+height+")")
                .call(xAxis);

        // Add the Y Axis
        svg.append("g")
                .attr("class","y axis")
                .call(yAxis);
    });

</script>
```

Bibliography

Adafruit (2015). " Adafruit". In: Accessed: 2017-07-08. URL: https://www.adafruit.com/product/386.

— (2016a). "Adafruit.IO Feeds". In: Accessed: 2017-09-02. URL: https://learn.adafruit.com/adafruit-io-basics-feeds/creating-a-feed.

— (2016b). "Adafruit.IO MQTT". In: Accessed: 2017-09-02. URL: https://learn.adafruit.com/mqtt-adafruit-io-and-you/why-mqtt.

— (2016c). "Adafruit.IO Overview". In: Accessed: 2017-09-02. URL: https://learn.adafruit.com/adafruit-io-basics-feeds.

AIM-Global-Network (2005). " Radio Frequency Identification (RFID) summary from the AIM Global Network". In: Accessed: 2017-05-29. URL: http://www.aimglobal.org/technologies/rfid.

Amazon-AWS-IoT (2016). "AWS Service Limits; General Reference (Version 1.0)". In: Accessed: 2017-08-28. URL: http://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_iot.

Amazon-Inc. (2017a). "AWS IoT Developer Guide". In: Accessed: 2017-08-28. URL: http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.

— (2017b). "AWS IoT: How it works". In: Accessed: 2017-08-27. URL: https://aws.amazon.com/iot-platform/how-it-works/?nc1=h_ls.

Aniruddha-Chakrabarti (2016). "AVP Digital, Mphasis, Using Node-RED for building IoT workflows". In: Accessed: 2017-06-16. URL: https://www.linkedin.com/pulse/using-node-red-iot-workflows-aniruddha-chakrabarti.

Arduino (2016). "Atmega16U2". In: Accessed: 2017-07-30. URL: https://www.arduino.cc/en/Hacking/DFUProgramming8U2.

Arduino-Uno (2017). " Arduino Uno". In: Accessed: 2017-07-30. URL: http://www.arduino.org/products/boards/arduino-uno.

Blackstock (2014). "Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED)". In: Accessed: 2017-06-04. URL: http://delivery.acm.org/10.1145/2690000/2684439/p34-blackstock.pdf?ip=140.203.251.59&id=2684439&acc=ACTIVE%20SERVICE&key=846C3111CE4A4710%2E5E11B48930E8A046%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=764425781&CFTOKEN=55825797&__acm__=1495207100_bd365674b97e0f80c2b70c6e0a132

Brock, D.L (2001). " The Electronic Product Code – A Naming Scheme for Physical Objects; Auto-ID White Paper, WH-002". In: Accessed: 2017-05-28. URL: http://auto-id.mit.edu/pdf/MIT-AUTOID-WH-002.pdf.

Bug-Labs-Inc. (2016). "Dweet.io Questions". In: Accessed: 2017-08-27. URL: https://dweet.io/faq.

Bug-Labs-Inc (2016a). "Freeboard Github". In: URL: https://github.com/Freeboard/freeboard.

— (2016b). "Freeboard.io". In: Accessed: 2017-08-27. URL: https://freeboard.io/.

Business-Intelligence.com (2017). "Historical Data". In: Accessed: 2017-09-02. URL: https://businessintelligence.com/dictionary/historical-data/.

CISCO (2013). "The Introduction to IoT". In: Accessed: 2017-05-28. URL: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/introduction_to_IoT_november.pdf.

D3.js (2016). "D3 Documentation". In: Accessed: 2017-09-02. URL: https://github.com/d3/d3/wiki.

EPCglobal (2005). " EPCglobal – RFID standards and regulations". In: Accessed: 2017-05-28. URL: http://www.oecd.org/sti/ieconomy/35472969.pdf.

ePro (2016). " Humidity Sensor DHT11". In: Accessed: 2017-07-08. URL: https://wiki.eprolabs.com/index.php?title=Humidity_Sensor_DHT11.

Gartner (2015). "Gartner Says Smart Cities Will Use 1.6 Billion Connected Things in 2016". In: Accessed: 2017-06-01. URL: http://www.gartner.com/newsroom/id/3175418.

Gopal (2016). "IOT IN MANUFACTURING". In: Accessed: 2017-06-02. URL: http://www.confabjournals.com/confabjournals/images/311201683339MECONFAB-5-1-2-10-16.pdf.

https://www.youtube.com/watch?v=BmYAGVmqguo (2016). "Arduino Excel 2.1". In: Accessed: 2017-09-02. URL: http://www.robertovalgolio.com.

Julio-Fernandez (2016). " Benefits of Node Red". In: Accessed: 2017-07-30. URL: https://developer.ibm.com/dwblog/2017/benefits-of-nodered/.

McKinsey-Global-Institute (2015). "McKinsey Global Institute (MGI), Executive Summary 2015;The Internet of Things: Mapping the value beyond the hype". In: Accessed: 2017-05-31. URL: http://www.mckinsey.com/mgi/overview.

Michael-Blackstock, Rodger-Lea (2016). " FRED: A Hosted Data Flow Platform for the IoT built using Node-RED". In: *Sense Tecnic Systems, Inc.* Accessed: 2017-08-27. URL: http://sensetecnic.com/wp-content/uploads/2017/02/MOTA-Middleware2016-preprint.pdf.

Nebojsa (2012). " NCT Thermistor". In: Accessed: 2017-07-08. URL: http://www.resistorguide.com/ntc-thermistor/.

Node-Red (2016a). " Node Red". In: Accessed: 2017-06-07. URL: https://nodered.org/blog/2016/10/11/version-0-15-released.

— (2016b). " Node Red Flows". In: Accessed: 2017-06-07. URL: https://flows.nodered.org/.

— (2017). " Node Red Getting Started". In: Accessed: 2017-06-16. URL: https://nodered.org/docs/getting-started/first-flowAccessed27/07/2017.

Sense-Tecnic-Systems (2017). " FRED, FRONT END FOR NODE-RED". In: Accessed: 2017-08-27. URL: http://docs.sensetecnic.com/fred/faq/#q-neato-how-does-it-work.

Software-Associates (2016). " Working with Node Red and Node JS". In: Accessed: 2017-06-07. URL: http://www.softwareassociates.in/working-with-node-red-and-node-js/.

UL (2016). " An Introduction to the Internet of Things, 2016". In: Accessed: 2017-06-01. URL: http://library.ul.com/wp-content/uploads/sites/40/2016/02/Internet-of-Things-white-paper_final.pdf.pdf.

Valgolio, Roberto (2016). "Arduino Excel". In: Accessed: 2017-09-02. URL: http://www.robertovalgolio.com/home.

Walsh J.C., Corbett-T. (2015). "mHealth Research Group NUI Galway: Using mobile technologies for effective health behaviour change. European Health Psychologist,17(4);" in: Accessed: 2017-06-02, pp. 193–197. URL: http://www.nuigalway.ie/psychology/mhealth.html.

Yen-Kuang, Chen (2012). "Challenges and Opportunities of Internet of Things". In: Accessed: 2017-06-02. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6164978.