



# MANCHESTER METROPOLITAN UNIVERSITY

Department School of Computing, Mathematics & Digital Technology

Faculty of Engineering

---

MSc Computing

NW.20 Sensor use and Data Analysis as a Health  
Monitoring Tool

---

A dissertation submitted to Manchester Metropolitan University in part fulfilment of the requirements  
for the degree of Master of Advanced Computer Science

Name: [REDACTED]

Supervisor: Nick Whittaker  
Year: 2019

## Abstract

The use of ubiquitous sensing enabled by Wireless Sensor Network (WSN) technologies, cloud based data storage and contextual awareness is having an impact on many different areas of modern life. Finding new uses within the Internet of Things arena is driving developments in pervasive wirelessly connectable devices. The focus of this project is to examine the use of sensors within an e-Healthcare domain to understand the complexities involved in developing practical applications that will produce a proactive and a diagnostic platform framework for Patient care in the 21<sup>st</sup> century.

The purpose of this project is to develop a software application that will act as a Sensor Hub gateway to collate, co-ordinate and manage potential Health data in a cloud based repository that can ultimately provide contextual analysis and utility. The application will use current technologies and relevant design patterns.

## Declaration

No part of this dissertation has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the dissertation containing citations to the work of others and apart from the assistance mentioned in the acknowledgements, this dissertation is my own work.

Signed \_\_\_\_\_

## Acknowledgement

Thanks to my family [REDACTED] who were extremely supportive during the long hours spent producing this document, especially at the weekends. Special thanks to [REDACTED] for allowing the sensor to be placed in her bedroom and to both [REDACTED] for providing excellent noise sensor data reading throughout the Project.

Thank you also to my supervisor Nick Whittaker who has provided guidance when needed during my course.

## Table of Contents

Abstract .....	2
Declaration .....	3
Acknowledgement .....	4
List of Figures .....	8
List of Tables .....	8
1 Introduction .....	9
1.1 Background .....	9
1.2 Motivation .....	9
1.3 Research Approach .....	11
1.4 Summary .....	11
2 Literature Review .....	12
2.1 Introduction .....	12
2.2 Internet-of-Things .....	12
2.3 Sensors .....	13
2.3.1 Introduction .....	13
2.3.2 Typical System Architecture .....	14
2.4 Big Data Analytics .....	17
2.4.1 Introduction .....	17
2.4.2 Contextual Awareness .....	17
2.4.3 Data Aggregation .....	19
2.5 Healthcare Application .....	19
3 Problem Analysis .....	26
3.1 Literature Review Conclusions .....	26
3.2 Ethical issues .....	29
4 Design & Implementation .....	30
4.1 Executive Summary .....	30
4.2 Functional Requirements .....	31
4.3 Non Functional Requirements .....	31

4.4 High Level Project Plan .....	32
4.5 Use Case Design .....	33
4.6 Technical Design.....	34
4.6.1 Use Case 1 – Node Red Gateway.....	34
4.6.2 Use Case 2 - 6 – Design Rationale for the Prototype MVP .....	34
4.6.3 Prototype Overview .....	37
4.6.3.1 Spring Framework <sup>[33]-[37]</sup> .....	38
4.6.3.2 Advantages of Spring Framework.....	39
4.6.3.3 Spring Architecture .....	39
4.6.3.4 Inversion of Control (IoC) .....	41
4.6.3.5 Dependency Injection (DI).....	42
4.6.3.6 Aspect oriented programming (AOP) .....	42
4.6.4 Google App Engine.....	43
4.6.5 Thymeleaf .....	44
4.7 Programming Languages .....	44
4.8 Data Exchange Format.....	45
4.9 Design Patterns.....	45
4.10 Integrated Development Environments.....	45
4.11 Piecewise Aggregate Approximation (PAA) .....	46
4.12 Functionality & Domain Modelling.....	46
5 Evaluation .....	47
5.1 Resulting System.....	47
5.2 Critical Analysis.....	51
5.3 Problematic Issues.....	52
5.4 Discussion .....	55
5.5 Future Project Work.....	58
6 Conclusion .....	60
References .....	62
Appendix A.....	65

Appendix B .....	65
Appendix C .....	66
Appendix D .....	67
Appendix E .....	67
Appendix F .....	68
Appendix G – Project Artefacts .....	71
Project TOR .....	71
Ethics Form .....	71
Sensor Hub code .....	71
Kanban Board .....	71
Test Data .....	71

## List of Figures

Figure 1 – Patient monitoring system based on IoT – Cloud Architecture

Figure 2 – Decision Tree Classifier

Figure 3 – Streamed patient sensor data in Web browser Interface

Figure 4 – The Layered architecture of SmartHealth Framework.

Figure 5 – Netatmo Client App Technical Parameter setup – Webhook details

Figure 6 – Proposed Sensor Hub Gateway Overview

Figure 7 – Spring Framework Architecture

Figure 8 – Resultant Sensor Hub conceptual MVC Design Pattern

Figure 9 – Resultant Sensor Hub Gateway

Figure 10 – Netatmo API functionality overview

## List of Tables

Table 1 – Use Case Iterations

Table 2 – Entity – Measure

Table 3 – Entity - AggMeasure



# 1 Introduction

## 1.1 Background

The Internet of Things, known as IoT, is a network of inter connected computing devices, mechanical and digital, objects, animals or people that are provided with unique identifiers (UIDs) that have the ability to transfer data over a network without human-to-human or human-to-computer interaction <sup>[30]</sup>. It is an ever-growing ecosystem that integrates hardware, computing devices, physical objects, software, and animals or people over a network enabling them to communicate and exchange contextual data.

The use of ubiquitous sensing, enabled by Wireless Sensor Network (WSN) technologies, has an impact on many different areas of modern life. It offers the ability to capture, measure, infer and understand our surrounding environment. The need to understand, forecast and notify interested entities is becoming increasingly important and has moved IoT from its infancy to potentially the next technological industrial revolution. The development of real-time data-on-demand analysis using sophisticated data mining via complex queries is being applied across multiple disciplines. The adoption of IoT is leading to the development of:

- Smart Homes – to optimise energy usage
- Smart Cities – aims to improve transport flows and energy consumption
- Environmental Sensing - providing real-time weather forecasts and pollution monitoring
- Industrial Resources and Asset Management - to provide economic efficiencies and increased profit margins
- E-Healthcare – the use of Wireless Body Area Network comprising wearable sensors to track physiological parameters and vital signs

This segmentation in IoT, through research and development, is leading to increased discipline specialisation that ultimately relies on the data capture and transformation to provide meaningful trend analysis.

## 1.2 Motivation

An area of interest for the use of Sensors and the Internet-of-Things (IoT) is Health Monitoring and early detection of various health conditions. The use of networked sensors, embedded within the home or worn on the body to gather environmental or physiological vital

signs is increasingly seen as a means of transforming health care in an increasingly resource constrained sector. It can be seen as a potential way to mitigate and manage healthcare in an increasingly ageing society. The development of frameworks for remote health monitoring is currently being established. These said frameworks leverage three tier architecture to create Wireless Body Area Networks consisting of wearable sensors as data acquisition, communication, networking and service layers is well established. The plethora of data at temporal longitudes and the development of new processing algorithms promises to facilitate and revolutionise medicine and future healthcare (Weiser & Gold (1999)). The continuous gathering of aggregated and effectively mined information can be used to optimise and personalise health outcomes moving from post facto diagnosis and reactive treatment paradigms to proactive early preventative methodologies.

The development of effective pervasive IoT applications for Healthcare or otherwise, relies on the integration of three main areas:

- Sensor network data capture
- Contextual Ontology analytics
- Mobile devices providing useful action notifications and interfaces

The production of a comprehensive system as described above is difficult due to the complexity of competing technologies used within Sensor nodes, software technologies themselves and the need for diverse and bespoke data interpretation.

The need for extensible mobile devices and software frameworks to provide unobtrusive but proactive decision notifications on behalf of users as proposed by Weiser in his paper on pervasive computing, will be central to the expansion and growth of practical IoT solutions.

This Masters project will be focused on developing a heterogeneous Enterprise Application for use in capturing Sensor data (WSN), that can be stored and aggregated in a suitable manner to provide dynamic historic trend analysis and / or early pattern detection notification functionality. The Sensor data will be persisted in a manner that will create a schema of meaningful aggregated data that can be transmitted via Web Service APIs to subscriber applications that can present the data in a suitable meaningful Graphic User Interface (GUI). Initially, as a prototype, sensor data will be used to identify any correlation between environmental conditions and its influence on a sleeping baby. The subject matter for this heterogeneous Enterprise Application transpired through my personal experience as a parent

to two sleep resistive infant daughters. Any understanding on how to improve infant sleep patterns could be positively received by parents of young infants with sleep problems.

### 1.3 Research Approach

### 1.4 Summary

The ultimate aim of this project is to produce a novel and extensible software platform and application as a Proof of Concept prototype that will provide a generic sensor node data capture tool that implements data aggregation functionality for consumption by extensible subscriber end points by Web Service APIs.

## 2 Literature Review

### 2.1 Introduction

The aim of this review is to investigate the latest software and methods for the capture, aggregation and transformation of sensor data. It is intended to provide an insight into the software platforms and frameworks currently available to inform development of a generic platform for use in the e-Healthcare industry and gain an understanding of its position within the Internet-of-Things arena. Ultimately, the literature review will aid the development of a Product Roadmap that identifies the latest software technologies, software design patterns and prior art, to build a cloud based Sensor Hub and identify potential industry standards.

### 2.2 Internet-of-Things

The term Internet of Things (IoT) is thought to have been coined by Kevin Ashton in 1999 during his work at Procter & Gamble, involving the optimisation of supply chains by utilising RFID technology. McKinsey defines IoT as “Sensors and actuators embedded in physical objects are linked through wired and wireless networks, often using the same Internet Protocol (IP) that connects the internet. However, in the past few years the definition now includes a wide range of applications such as healthcare, transport, smart grids and smart cities, all centred on providing a contextual aware environment through the use of computing.

The new era of pervasive computing was envisioned by Weiser (1991), the forefather of Ubiquitous computing (ubicomp), who defined a smart environment as the physical world interwoven with sensors and actuators, displays, and computing elements embedded seamlessly into networked everyday objects. The practical use of IoT is currently developing rapidly due to the progress of two critical technologies – the advancement of Micro-electro-mechanical systems (MEMS) offering wireless communication within networks and Cloud Computing.

MEMS are miniature devices called nodes that interconnect to form a wireless sensor node (WSN) providing a ubiquitous sensing capability. The technological progression of these devices has provided practical sensing capability able to compute and communicate over short distances in a variety of topologies whereby node data can be harnessed in a multitude of ways.

To complete Weiser's IoT vision of scalable, secure and market driven sensing, Cloud Computing has emerged to deliver reliable computing resource and data storage to enable analysis and data mining at commercial levels. The data and context can be presented to users in easy to understand web based Mobile applications, that can be offered to a viable customer base and sold commercially.

The creation of integrated Sensor-Actuator-Internet frameworks will shape the future of Smart Sensing and providing core interoperable platforms will play a major part in its development. Objects in IoT must communicate and exchange data autonomously (Mitrokotsa et al. 2013) and success depends on standardisation, interoperability, compatibility, reliability and effectiveness on a global scale.

To fully take advantage of all available Internet technology to provide data-on-demand services, research will be undertaken on how to use and combine all available Cloud computing software, hardware and methodologies – a common operating picture (COP) described by Gubbi et al. (2013).

## **2.3 Sensors**

### **2.3.1 Introduction**

Networked sensors, either worn around the body or embedded in our surroundings, enable the gathering of data related to our physical or mental health conditions. This data can be captured continuously, aggregated and analysed to bring about a positive change in healthcare and wellbeing.

Currently, there has been a rising interest in wearable sensors for use in fitness, activity awareness and personal health care and several devices are commercially available, such as FitBit, Jawbone Fitness trackers and the Apple Watch. These wearable devices are integral to frameworks that leverage three tier architecture to create a Wireless Body Area Network (WBAN). A WBAN comprises a wearable sensor for data acquisition, communication, network and service layer that can be used to measure various physiological parameters such as blood pressure, temperature, heart rate and steps taken.

WBANs generally transmit gathered data to a gateway server by utilising a Bluetooth connection. The gateway stores an Observation and Measurement file (O & M) on a remote

server accessible via the internet (in real-time or later) to data analysts and clinicians. The gathered data is stored and for the broadest use, it is likely to be a Cloud based storage repository to provide scalable functionality. A cloud based storage solution can provide a data analytics tier that is used to aggregate and shape the data. The tailored data is then suitable for presentation or accessed via a visualisation layer to be consumed by multiple clients. Each client can potentially present the data in a format that corresponds to the health issue being tracked i.e. Scatter Chart, Pie Chart, Graphic Curve inter alia.

Accurate diagnoses and monitoring relies on data comprising multiple physiological values that have been captured over a significant period of time. This high dimensionality of both time and volume of data requires targeted data aggregation techniques to extract meaning and context to provide utility to skilled clinicians.

### 2.3.2 Typical System Architecture

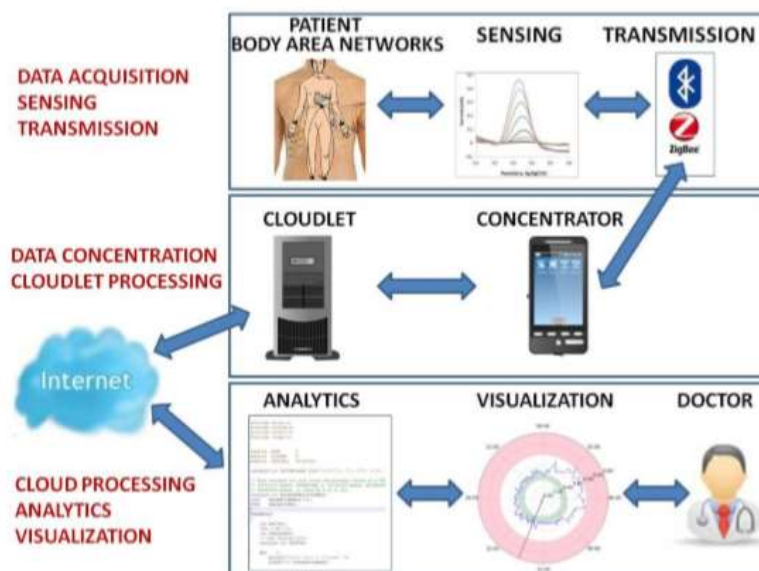


Figure 1 - Patient monitoring system based on IoT Cloud Architecture

Figure 1 illustrates a typical remote health monitoring system.

#### Data Acquisition

Measures physiological biomarkers such as: -

- Electrocardiography (ECG)
- Skin temperature
- Respiration Rate
- Electromyography (EMG) muscle activity

- Physiological biomarkers
- Gait (posture)
- Sweating
- Blood pressure
- Blood Glucose

Data is acquired either by wearable devices or by devices located close to a patient. Therefore, the devices must be small, light, unobtrusive and energy efficient. The low energy requirement can pose a challenge for the quality of data in terms of signal to noise ratio. The use of low power communication protocols is necessary as communication can account for a significant portion of the power consumption in sensors.

ZigBee over IEEE802.15.4 is frequently used in low rate WPANs (LR-WPANs) to provide communication between low power devices that operate in personal operating space of approximately 10m. Zigbee provides a good mesh network with extended battery life.

Bluetooth low energy (BLE) is another wireless communication protocol for low power short range communication suitable for health monitoring purposes.

Similarly, Low Power Wireless Personal Area Networks IPv6 (6LoWPAN) can be utilised to connect energy constrained WPAN devices to the internet.

The disclosed communication protocols often require an intermediate node (data concentrator) to present the available data over the internet. As such, sensors connect to the network through an intermediate data aggregator that is typically a smart phone located near the patient.

## **Data Transmission**

Data transmission is the secure transfer of the data in real-time from the patient or surroundings to the data centre. Transmission of acquired data is usually IEEE 802.15.4 – short range radio such as Zigbee or low Power Bluetooth to the Concentrator or Gateway Hub. The Gateway / Concentrator is relayed using a Smart phone, either by using WiFi or mobile phone network. Potentially aggregated data can be stored historically to allow for further trend analysis.

Often a storing / processing function is provided close to the mobile device known as a cloudlet (Quwaider and Jararweh (2013)). The cloudlet (i.e. a desktop computer) accesses the concentrator via WiFi and offers additional storage and data aggregation that require more processing power or storage than the mobile device.

### **Cloud Processing**

Source sensor data or aggregated data is transmitted to a cloud based repository for three distinct functions – storage, analytics and visualisation.

Firstly, the storage of biomedical data can assist clinicians in either long or short diagnoses. Historic data can aid and inform healthcare providers and statisticians with their medical trend analysis or health issue root cause analysis. Short term diagnosis relies on more specific data analysis and a greater understanding of acute biometric symptoms.

Cloudlet is a limited resource computing and storage platform that reduces the need to outsource intensive tasks to the enterprise cloud. It has been developed to deliver low latency to time critical tasks on aggregated data. Communication between the cloudlet and the concentrator is carried out by using WiFi and reduces data transfer latency. Long Term Evolution (LTE) network access provided by the concentrator can be used for direct transfer to the cloud (can bypass the cloudlet) but retain the latency speeds provided by the mobile network.

Aggregation of data can occur in the Cloudlet (transmitted by WiFi) or in the Cloud (LTE) and studies have shown the former is more efficient from a latency and energy efficiency perspective disclosed by Jararweh et al. (2013). Ultimately the aggregated data is stored in the cloud to enable distribution, contextual analysis and remote access. The contextual awareness component is essential to understanding the immediate and long term health status of the patient.

Clinicians and Data Scientists need to work closely to develop suitable visualisation methods that can transform the underlying sensor data and contextual algorithms into accessible formats. The data must be modelled in such a manner to make it easy for clinicians to recognise symptom outliers, for a variety of possible health problems – for both acute and long term issues.



## 2.4 Big Data Analytics

### 2.4.1 Introduction

The volume of data generated by sensors, social media, healthcare applications and other digital services that can be stored in a structured or unstructured manner is termed “big data”. Big data can be categorised according to three aspects: (i) volume, (ii) variety and (iii) velocity. There are huge opportunities presented by understanding and managing the capability of providing meaningful insight from vast volumes of IoT data particularly for the use in application of smart cities, transport, grid systems, smart meters and remote patient health care devices. The International Data Corporation (IDC) indicates that the big data market will reach over \$125 billion by this year (Gantz and Reinsel (2011)).

Big data analytics can be described as the methods by which IoT data can be interpreted to reveal trends, unseen pattern, correlations and new understanding. Big analytics aims to extract knowledge to allow predictions, identify trends and help business make pertinent decisions. Data mining uses various statistical and machine learning methods to provide insight into the data which is particularly difficult for IoT derived data as sensors provide significant dimensionality, heterogeneity, noise, variety and rapidly growing volumes.

### 2.4.2 Contextual Awareness

Contextual awareness is the ability of a component within a system to collect information from its environment and adapt or trigger behaviours accordingly. It is an area of pervasive computing that uses software and hardware to automatically collect and analyse data to guide responses. The presence of a contextually aware computing device or sensor, ideally unobtrusive to users in the environment. Contextual information covers a wide range of categories including time, location, device status, identity, users, role, privilege settings, activity data, process and nearby users and devices.

For user application, context awareness can guide services and enable experiences including augmented reality, environment context-relevant information and marketing messages. For e-Health contextual awareness and within the scope of this project, the collection, collation and management of multiple device data streams to develop an overall user health profile is particularly pertinent. Contextual awareness within e-Health is inherently an exploratory function whereby existing contextual awareness is utilised to improve and fine tune existing understandings or create new contextual awareness.

Abowd and Mynatt (2000) identified the 5 W's, 'who', 'what', 'where', 'why' and 'when' as the minimum information that is necessary to understand context. Dey et al. (2001) claimed this definition was too specific and could not be used to understand context in a broader sense and provided a further definition as:

“Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the applications themselves.”

Mostefaoui et al (2004) disclosed Context acquisition as three methods:

- Sensed context: environmental information can be acquired from physical or software sensors such as temperature, pressure, lighting and noise level – ideally unobtrusively collected from the users' surroundings
- Derived context: this type of contextual information can be computed at real time, for example time and date
- Context explicitly provided: user's preferences that are explicitly communicated by the user to the requesting application

Hu et al. (2008) disclose three approaches to build context-aware applications: -

1 No application-level context model:

- Applications perform all the actions, such as context acquisition, pre-processing, storing, and reasoning within the application boundaries

2 Implicit context model:

- Application uses libraries, frameworks, and toolkits to perform context acquisition, pre-processing, storing, and reasoning tasks.
- It provides a standard design to follow that makes it easier to build applications quickly
- The context is still hard bound to the application

3 Explicit context model:

- Applications uses a context management infrastructure or middleware solution.
- Therefore, actions such as context acquisition, pre-processing, storing, and reasoning lie outside the application boundaries.

- Context management and application are clearly separated and can be developed and extend independently.

### 2.4.3 Data Aggregation

Rajagopalan and Varshney (2006) compare data aggregation algorithms in wireless sensor networks in order to optimise lifetime, latency and data accuracy. The paper discloses a discussion on Flat versus Hierarchical networks and various protocols such as Sensor protocol for information via negotiation (SPIN) and Low Energy Adaptive Clustering Hierarchy (LEACH) inter alia. The paper is focused on data aggregation that occurs within the Sensor networks prior to transmission to the Network gateway and how various protocols or topologies can improve the WSN's network lifetime, data latency, accuracy and energy consumption. This paper suggests future work should focus on integrating security into the networks and also the application of source coding theory and compression to improve correlation and energy efficiency for data gathering networks. The paper demonstrates the ubiquitous use of data aggregation within the IoT arena - within sensor data capture at node level rather than at the more commonly known contextual awareness function performed against captured sensor data.

## 2.5 Healthcare Application

The detection of symptoms and disease diagnosis are fundamental to medical treatment. In a world of increasingly ageing populations and stretched medical resources, new ways to enhance detection of health problems can greatly improve efficiencies. The automation of symptom data collection and observations can potentially provide a proactive method of detecting medical problems before they become acute.

Health IoT (H-IoT) will in the future play a major role in enhancing peoples' health levels by capturing both body and environment level data. A personal health dashboard will result from the combining IoT technologies involving microelectronic systems, Wireless Sensor Networks (WSN), Wireless Body Area Networks (WBAN), computer science and data analytics.

Babu et al., (2013) propose use of Sensor Web Enablement (SWE) and Sensor Observation Service (SOS) provided by 52<sup>0</sup>North52<sup>0</sup> [28] to provide a remote health monitoring system by use of Application Program Interface (API) to integrate sensors to a web based interface. The described work demonstrates using Open Geo-Spatial Consortium (OGC) standard [28] in a

health care application to provide data in an interoperable manner that reduces data redundancy. It utilises Sensor Model Language (SensorML) as an Observations and Measurement (O & M) document to encode, transmit, and query health related data. O & M data is streamed to a database and periodically retrieved and parsed (XML DOM parser) for presentation in a web browser for a Clinical assessment. Ultimately, 52<sup>0</sup>North52<sup>0</sup> have developed a Java servlet based implementation that provides communication using SensorML that is closely related to SOAP and XML like protocols. The cloud based MySQL database provided by Cloudbees <sup>[27]</sup> completes the Software as a Service (SAAS) persistence layer of the remote healthcare sensor application.

The cloud computing provides for a fault tolerant and extensible environment to enable Decision tree algorithms (Figure 2) to be fine-tuned and applied to provide insight and aid decision making based on the captured patients' data.

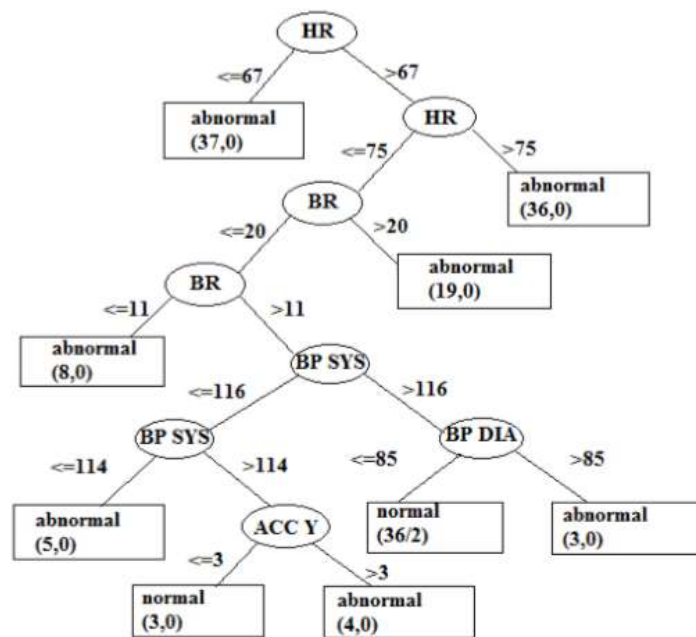
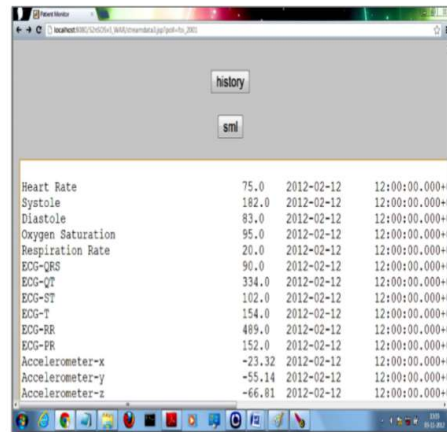


Figure 2 - Decision Tree Classifier

Clinicians query the stored data using AJAX and SOS and it is displayed in an interface as shown in Figure 3.



*Figure 3 - Streamed patient sensor data in Web browser Interface*

It can be seen that the application as proposed by Babu et al. offers the opportunity to supply bespoke back end cloud based Decision algorithms and that, in collaboration with a skilled data clinician, could be developed for a variety of illness or medical conditions. Additionally, the interface displaying the data can be improved and made more extensible by using the Web service response from SOS as a generic feed to multiple visual platforms or technologies.

Honan et al. (2016) consider two aspects of the frameworks that are needed for H-IoT; the first, the challenges presented in providing reliable communication infrastructure within the WBANs and the second, centres on the need for new data visualisation methods required to create intuitive decision support and summarised medical data from the large volume of sensor data produced.

The communication challenges identified the need for unified wireless standards for Digital Health (D-Health) systems and the protocol design challenges. Typically, wireless standards are based on IEEE 802.15.6 that includes radio frequency (RF) based ultrawideband (UWB), narrowband and RF-based human body communication standards. Therefore, sensor devices operate across a range of frequency bands and data rates vary significantly (UWB operates between 395 Kbps and 12.636Mbps, narrowband between 100 Kbps and 1000Kbps) causing interoperability problems. The enforcing of stricter standards within H-IoT is aimed at producing on-body networks that address energy efficiency, security, privacy and low electromagnetic interference.

Protocol design challenges are identified across a number of areas which include PHY layer, MAC layer and Network layer issues.

It is disclosed that Support Vector Machine and Random Forest based classifiers were the most accurate Machine Learning algorithm to provide decision support.

Overall, Honan et al. propose a holistic D-Health framework that is optimised - at the Front End (from a data acquisition and mobile crowd sensing perspective) and Back End sections – by using cloud computing to host various aggregation and pre-processing functions to optimise web service end points.

Chondamrongkul (2017) describes a context aware healthcare application that is primarily based on J2EE built on a public cloud platform that uses Google App Engine. The platform captures health data from wearable sensors and combines the information with medical records from a Hospital Information System (HIS) to enable a contextual analysis process and applies clustering to detect possible symptoms. Collected data includes heart rate, respiratory rate, blood pressure and movement.

Lakshmanachari et al. (2017) discuss the use of sensors used to gather Blood pressure (Systolic, Diastolic and Pulse), ECG (configured for threshold values) and Temperature. They propose cloud data storage accessed by android web server to access the patients' information and provide notifications.

Wendt et al. (2016) introduce MASON an open dynamic ontology formation framework that allows developers to add logical pieces to a greater network of contextual reasoning for shared use by application developers. MASON is described as a framework for supporting modular contextual reasoning development by handling low-level sensor routing and abstracting data sources as composable and functioning reactive data streams. They demonstrate functionally reactive programming interfaces for implementing contextual abstractions (CAbs) and an API for integrating CAbs into applications. MASON includes a dependency resolution system that manages the CAb installations and prompts users to install new ones. Potential uses driven by real life use cases are disclosed (music choices determined by movement activity profiles) and it is concluded that open source applications could incorporate MASON and include reactive functionality in application design through the frameworks adoption.

The architecture is designed to support multiple sensor integrations using CAb registrations and data stream routing.

Sakr & Elgammal (2016) discuss how emerging sensing technologies, cloud computing, internet of things and big data analysis can improve efficiencies and effectiveness in healthcare services particularly with larger aging populations in advanced economies. A SmartHealth framework, for big data analytics services is proposed and intended for use in the healthcare domain. The term Cyber Physical System (CPS) is defined to represent the ICT advancements in sensing and big data analytics. They in turn are involved in the co-ordination of resources ranging from networked embedded computers, mobile devices to multimodal data sources in multiple domains, such as system health monitoring and chronic health management.

Sakr & Elgammal disclose that few CPS architectures have been proposed to address the healthcare industry. Generally, the currently disclosed healthcare CPS frameworks suffer from issues such as security, privacy, and failure to adhere to common standards and thus technologies lack structural integrity.

Their proposed SmartHealth framework solution is presented below.

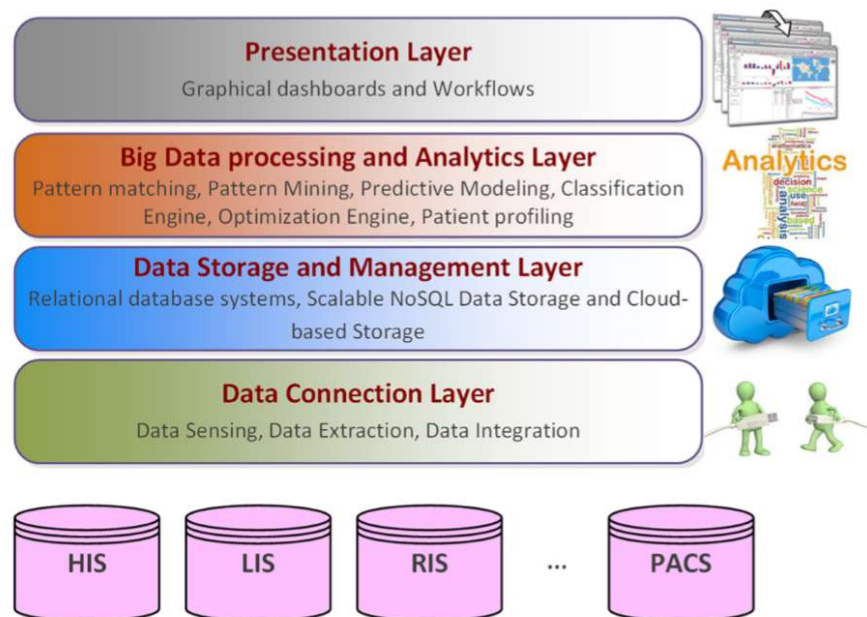


Figure 4 - The Layered architecture of SmartHealth Framework.

One of the key design goals of the SmartHealth framework is to integrate the big data sources as follows:-

HIS – Hospital Information Systems

LIS – Library Information Systems

RIS – Radiology Information Systems

PACS – Picture Archiving and Communication Systems

By harnessing smart sensors, cloud computing and data processing its aim is to build a platform of scalable data management and analytics.

The platform's main function of the framework is to provide healthcare analytic methods, data modelling and machine learning to provide better health outcomes, improve quality of care, reduce costs and provide insight and new discoveries.

Sakr & Elgammal categorise the analytical techniques as:

Descriptive analysis – explain what is happening in a given situation i.e. What has happened? Techniques include descriptive statistics using histograms, charts, box and whisker plots or data clustering.

Diagnostic analysis – understand why certain events occur and what are the key drivers. i.e. why is a disease infection increasing. Why are certain patients readmitting every month? Common techniques are classification, clustering, decision trees or content analysis.

Predictive analysis – predict outcomes in the future and the probability of the uncertain. Can predict which patients will get some diseases or not. Statistics and Machine learning are the appropriate tools for this technique

Prescriptive analysis – used to suggest the best course of action to optimise decision outcomes. Typically combine a predictive model with business rules (e.g. decline a medication if the probability of a side effect is above a threshold). Techniques include decision trees, linear and non-linear programming and Monte Carlo simulations.

In many ways the aforementioned healthcare techniques are analogous to Abowd's context categorisation of the 5 w's, 'who', 'what', 'where', 'why' and 'when' previously discussed (page 17).

Sakr et al. disclose many cloud based data base technologies that can be used to provide the SmartHealth framework such as Amazon RDS, SQL Azure, Amazon DyanmoDB, Google Datastore. Potential Analytics Layer provided by machine learning disclosed include Apache Mahout, SystemML, BigML, Hadoop, Spark and R and there are more.



The problem with this article, as with so many others, is that there are no specific proposals on how to implement a practical framework or platform itself. IoT is generally thought to suffer from a lack of interoperability (Groopman (2016)) but its implementation can be nebulous due to the large number of available data base technologies, programming languages and cloud computing solutions available. I concur with this proposition of interoperability after getting to grips with sensor technology itself with its multitude of communication protocols, message formats and source data variables discussed in the literature.

## 3 Problem Analysis

The aim of this section is to discuss issues identified by the review and specify the requirements and scope of the system.

### 3.1 Literature Review Conclusions

#### 3.1.1 Interoperability

It is clear there is a plethora of sensors to monitor a host of medical related data and that it is a difficult task to provide a simple, common and interoperable framework to centralise and collate all data feeds to provide contextual reasoning. However, the development of increasing contextual aware mobile devices and cloud computing technologies can allow developers to build dynamic ontology applications to capture data and run cost effective contextual analysis. Ideally, a common framework that supports modular sensor nodes to capture data for storage in cloud based repositories is considered a minimum requirement to enable eHealth (H-IoT). This could lead to contextual reasoning used by the mass population as a practical and useful everyday tool. It is considered that a common framework would be a technology leap comparable to the Microsoft Windows operating system.

A key criterion to developing useful eHealth framework is the ability to capture multiple sources of medical sensor devices' data. Physiological data is presented in a variety of ways in the human body. This stored data has to be displayed in a user friendly way to an analytical team. Various interfaces required that render or shape the data in a manner that allows trends or acute problems to be quickly identified. The lack of interoperability of sensors and the plethora of proprietary frameworks presents a fundamental stumbling block in the development of useful eHealth monitoring tools. This therefore leads to the main aim of this project – *the development of a cloud based sensor hub that can capture multiple data sources to enable cloud based data analytics, Machine Learning and contextual awareness.*

A proprietary software company providing remote eHealth sensor data will typically capture and store data in their own cloud based repositories. For practical purposes, a multitude of different sensors are needed to be tracked and monitored by medical data scientists to create a holistic patient dashboard. The co-ordination and collation of the data is key to develop a practical use. This can be particularly important to understand how multiple contributing health and environmental factors can have an impact on a patient's health outcomes. Capturing multiple sensor feeds can potentially offer more insight into the contributing factors involved in certain health problems and facilitate diagnosis.

The literature review has revealed that there are few practical applications, frameworks or platforms developed or disclosed that aim to gather and co-ordinate multiple eHealth sensor feeds. The review reveals multiple high level hypotheses regarding potential practical application and future benefits - however the co-ordination of multiple sensor eHealth data streams is not prevalent in the literature. The details of software used to manage eHealth data feeds to enable contextual analysis appears thinly documented.

It is considered that the MASON framework as proposed by Wendt et al. (2016) potentially offers an insight into the future developments required within the eHealth sensor world. The MASON framework relies on contextual abstractions (Cabs) being provided to hook up sentinel and reactive behavioural triggers. The paper discloses that MASON utilises the Aware framework<sup>[29]</sup> that has been developed to use the contextual and spatiotemporal functions of Android mobiles phones to which users can subscribe. It is proposed that eHealth sensors frameworks will be developed similarly to the Aware framework to enable data capture and contextual understanding to be ascertained and co-ordinated. The Aware framework has integrated a number of key data feeds and to provides interface specifications to enable node communication. Defined and disclosed communication protocols are key.

*Thus, the main aim of this project is to develop an open source programming framework or platform that is suitable for use alongside cloud based database technology – a practical software framework implementation. Notionally the framework needs to offer extensibility where database technology is concerned and user interface presentation layers to handle varying sensor data streams and data types.*

For efficiency, it is proposed that a simple method to capture data itself will be adopted. A sensor will be used that can provide data from the cloud. Thus a Sensor Hub gateway node will be developed to co-ordinate sensor data streams.

The review also indicated that the main inhibitor for main-stream adoption of Iot was the problem of lack of interoperability. This is preventing realisation of the vision of IoT, where objects can talk seamlessly to other objects regardless of location or manufacturer. Web Services are considered the best solution to address these issues and have been incorporated into the project design.

### *3.1.2 Usability & Adoption*

To evaluate the design of the Sensor Hub Graphical User interface, Jakob Nielsen's 10 Usability Heuristics for User Interface Design have been considered the preferred guidelines to follow when designing the web site user experience. Nielsen's guidelines appear relatively simple and more directly related to web design and user experience. Nielsen's Heuristics are summarised as follows: -

1. **Visibility of system status** – The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. **Match between system and the real world** - The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
3. **User control and Freedom** - Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.
4. **Consistency and standards** - Users should not have to wonder whether different words, situations, or actions mean the same thing.
5. **Error prevention** - Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.
6. **Recognition rather than recall** - Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
7. **Flexibility and efficiency of use** - Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.
8. **Aesthetic and minimalist design** - Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

9. **Help users recognise, diagnose, and recover from errors** – Error messages should be expressed in plain language (no codes), precisely indicate the problem and constructively suggest a solution.
10. **Help and documentation** - Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

The user experience (UX) is a very important consideration when designing a web site and a UX test phase should be a major part of any software development life cycle. Whether using a service similar to that provided by Manchester Metropolitan University, that offers a user experience laboratory monitoring users' eye movements, or getting some friends to see how navigable a family web site is, before hosting it in the cloud – user experience is an important component of the test process.

### 3.2 Ethical issues

All potential ethical issues for this project have been considered and none have been uncovered in relation to the implementation or evaluation of this project. The completed Ethics Check Form was submitted online (see Appendix G).

## 4 Design & Implementation

### 4.1 Executive Summary

Following an extensive literature review, it has been concluded that the collection, organisation and transfer of a large number of disparate sensors' data to a common cloud repository via a cloud based sensor hub, could be a valuable mechanism for analysis of interlinked eHealth remote sensor kits. A sensor hub web service API will be developed (essentially a gateway node) to act as a co-ordinating agent to manage the collection and storage of remote data into a cloud based database.

A standard commercial sensor that processes a variety of data values and measurements will be used to build a generic software prototype as a gateway node. Other methods of capturing sensor data may be considered and evaluated.

The Netatmo Smart Indoor Air Quality monitor will be used in this project. This device provides readings for Temperature, Air Quality, Humidity and Noise, all potential environmental measureable factors influencing a patient's sleep pattern or respiratory response and symptoms – thus simulating eHealth functionality.

The proposed sensor hub API will be designed to be extensible, will possess a user interface to allow the demonstration of simple data aggregation to demonstrate basic contextual analysis. It will be cloud based and comprise software technologies that enable quick development build cycles.

Ideally, on successful deployment of the simple sensor hub API, alternative contemporary data repositories used for data mining, machine learning and cloud based analysis will be evaluated for ease of deployment and utility. This additional use case would demonstrate the wide ranging opportunities in the eHealth landscape.

The overall aim is to produce a generic cloud sensor hub gateway application that can be scaled up to co-ordinate multiple sensor inputs and store captured data in a variety of storage technologies for use as an eHealth remote sensor co-ordinator.

## 4.2 Functional Requirements

- Identify a method of capturing and storing a generic remote Sensor's data in a manner that enables subsequent manipulation and data mining to be performed for trend analysis.
- Develop a simple Sensor gateway hub co-ordinator to capture and store a generic remote Sensor's data in a manner that enables subsequent manipulation and data mining to be performed for trend analysis.
- The Sensor Gateway Hub must be developed in common open source technology, adopt contemporary Software development techniques and methodologies and be compatible with IoT standard protocols.
- The application must provide IoT interoperability and be extensible.
- The Sensor Gateway Hub application must be cloud based and provide cloud based repository functionality.
- Application must be developed using appropriate software design patterns.
- Store data in a logical and usable manner to facilitate data analysis
- Be scalable and portable to other target databases
- Capture Netatmo sensor data and store in a local or cloud based database by using currently available IoT tools

## 4.3 Non Functional Requirements

- Application must be scalable in a manner that will enable a large batch file of inserts to be stored to aid testing and simulate a sensor that generates a high volume of readings over a short interval.
- Database should be secure and easy to maintain and relatively inexpensive.
- User Interface requires good response times and be extensible.
- Develop a methodology for co-ordinating and presenting multiple sensor data feeds.

## 4.4 High Level Project Plan

The Agile and Waterfall methodologies are compared by Feng (2011) and describes Waterfall (Royce, 1970) as being more suited to larger, more complex and high requirement documented projects whereas Agile is disclosed as more suited to projects where requirements are less clear and subject to change (Torgeir et al., 2012).

An Agile project methodology will be adopted to produce and document this project rather than the Waterfall approach. The reason for using this approach, is that one of the fundamental aspects of this project is identifying a method of capturing sensor data from a Netatmo device and it's considered very exploratory in nature. Therefore, the production of software will be dependent on the success of each identified Project Iteration. This in turn will be evaluated after each milestone to inform the next phase.

The Project planning and documentation will use Agile artefacts such as a Kanban board to structure the delivery milestones. A number of Project Iterations will be scoped and planned to provide achievable and SMART objectives from a functional and practical perspective.

Agile is a software development methodology whereby requirements and solutions develop iteratively through collaborative self-organising and cross-functional teams working closely with customers and/or end users. It advocates adaptive planning, early delivery and continual improvement and promotes responsiveness to users' changing and developing requirements. Early delivery is encouraged to gain customer buy-in, loyalty and trust.

The Waterfall model is a more formally structured methodology that includes a separate testing phase after a development phase. In contrast the Agile method includes build and test within the same phase or iteration. The Waterfall methodology includes formal check points and stakeholder sign off at each phase before moving on to the next phase. Generally, Waterfall structured projects produce larger more complex deliverables (either software or Project documents artefacts) at the end of each phase whereas the Agile method produces more delivery cycles, smaller incremental software improvements and potentially less fixed deliverable targets. The Agile methodology supports a Product rather than a Project approach.

For the purposes of this project, the Agile methodology is considered more appropriate due to the exploratory nature of the project, particularly when dealing with proprietary software of a third party sensor supplier. The Agile approach is obviously aimed at small modular teams



working closely together and thus is not ideal in this case but the reactive and iterative nature lends itself to an exploratory project.

#### 4.4.1 Project Iterations

Table 1 Use Case Iterations

Iteration	Use Case
1	Explore currently available sensor node gateways to capture sensor data
2	Develop a Web service API (Sensor Hub) to capture a load sensor data
3	Enhance Sensor Hub platform to provide simple data aggregation functionality
4	Enhance Sensor Hub to provide Graphical User Interface of Cloud data (Raw and Aggregated)
5	Add a new Sensor input feed and Data repository
6	Connect Sensor Hub to additional Database technology

## 4.5 Use Case Design

### 4.5.1 – User Story - Iteration 1

As a health care data analyst, I want to capture a Netatmo Smart Home sensor's data (or new eHealth remote sensor's data) without building or buying any new software so that I can aggregate and interrogate the data to enable trends or easily detect acute health issues.

### 4.5.2 – User Story Iteration 2

As a health care data analyst, I want to capture a Netatmo Smart Home sensor's data (or new eHealth remote sensor's data) so that I can aggregate and interrogate the data in a bespoke manner to enable trends or easily detect acute health issues.

### 4.5.3 – User Story Iteration 3

As a health care data analyst, I want to access data aggregation functions from an eHealth sensor hub so that I can understand a patient's health status from a variety of external data streams in a bespoke manner to enable trends or easily detect acute health issues.

### 4.5.4 – User Story Iteration 4

As a health care data analyst, I want to view aggregated eHealth sensor data in a convenient and useful graphical user interface so that I can quickly identify trends or outlier physical or environmental events to understand a patient's health or detect underlying issues.

#### *4.5.5- User Story Iteration 5*

As a health care data analyst, I want to analyse multiple health sensor data streams so that I can develop an understanding of a variety of health data variables to identify trends, new data stream interactions or health care outcome hypotheses directly applicable to improving patients' health.

#### *4.5.6 – User Story Iteration 6*

As a health care data analyst, I want to implement analysis and data mining using the latest contextual awareness, machine learning and cloud based database technology so that I can scale up the eHealth data streams to prove and confirm any identified trends or physiological variable interactions with the aim of improving human health outcomes in larger populations.

### **4.6 Technical Design**

#### **4.6.1 Use Case 1 – Node Red Gateway**

Node-Red is a flow-based development tool for visual programming developed by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

Node-RED provides a web-browser based flow editor (locally or cloud based) that can be used to create Javascript functions. Elements of applications can be saved or shared for re-use and data can be readily wired for upload to cloud based data repositories. The runtime is built on Node.js.

The flows created in Node-RED are stored using JSON.

It was discovered that a number of Netatmo data flows had been developed for the Node-RED web based flow editor (see Appendix A). The initial plan is to explore these currently available flows to stream the Netatmo Sensor data to a cloud based data repository. If successful, the project could be focused on exploring machine learning data base technology and its role in contextual analysis.

#### **4.6.2 Use Case 2 - 6 – Design Rationale for the Prototype MVP**

For business use cases 2 – 6, the software will be developed using an Agile iterative approach. After each successful iteration milestone, the current software product will be developed further until all iterations are complete and all Use Cases (2 - 6) are completed.

After an evaluation of current software technologies and trends, it has been decided that for Iteration 2 – 6, the baseline REST API web service Sensor Hub gateway, the application will be developed using current Java and J2EE framework technology. A key consideration is that the programming language and its technology must be open source, relatively light weight and popular (to support future growth).

To that end, the framework choice is Spring and in particular Spring Boot due to the extensibility, rich functionality (embedded security and data persistence *inter alia*) and its abstraction to simplify the coding itself.

The prototype sensor to be used is a Netatmo Smart Home as it measures and stores environmental factors as follows:

- Temperature
- Humidity
- CO<sub>2</sub> levels
- Noise

All aforementioned factors can be considered to have an influence on human health and are therefore considered a good prototype sensor device. Netatmo provides access to cloud based data and is therefore a viable Node producing an eHealth prototype data feed.

Netatmo Connect provides three programs of API for each of its devices: Weather, Smart Home and Enterprise. Direct access to a sensor's data is not provided but the various API connection services expose user's data from a cloud based database. Netatmo provide a Developer SDK in a variety of programming languages: PHP, Objective-C, Java (Android) and Windows 8. Additionally, code samples in Python and Javascript are available for client side connectivity to the Netatmo cloud database.

Netatmo provides a device owner with a developer account that can be used to Create an App. The App generates a Client Id and Client Server keys (OAuth2 security protocol) to give user's access to the device's cloud based data using webservises. Netatmo offer a Webhooks URL service, which prompted an investigation into webhooks and their potential use within IoT gateway architecture. After investigation, it is thought that webhooks will play a major role in the development of IoT applications particularly in the arena of eHealth. It is in the nature of proprietary software companies to monetise software and sensor Intellectual Property whilst simultaneously exposing the sensor data without revealing the underlying

code base and technology. Webhooks offer a way to securely feed data to receiver nodes to multiple users in a scalable manner by using REST services and simple Post Http end points.

#### *4.6.2.1 - Webhooks*

A webhook (also called a web callback or HTTP push API) is a way for an app to provide other applications with real-time information. As more IoT applications are developed and event driven communication is required, webhooks are an incredibly useful and resource-light way to implement publisher/subscription design paradigm applications. A webhook delivers data to other applications as it happens, meaning you get data immediately. Unlike typical APIs where you would need to poll for data very frequently in order to get it in real-time. This makes webhooks much more efficient for both provider and consumer. The only drawback to webhooks is the difficulty of initially setting them up/initial set-up.

Webhooks are sometimes referred to as “Reverse APIs,” as they give you what amounts to an API specification, and you must design an API for the webhook to use. The webhook will make an HTTP request to your App (typically a POST), and you will then be charged with interpreting it.

Adopting a webhook programming paradigm could be key to integrating sensors to enable effective eHealth Data Analysis. A webhook allows proprietary sensor device companies to own and store their own data but provides a means whereby a co-ordinating service may subscribe to a sensor device’s data stream to provide a multi-faceted health monitoring service.

#### *4.6.2.2 Consuming a Webhook*

The first step in consuming a webhook is giving the webhook provider a URL to deliver requests to. This is most often done through a backend panel or an API. Therefore, a URL must be provided by a consuming application that is accessible from the public web.

The majority of webhooks will POST data in one of two ways: as JSON (typically) or XML to be interpreted, *or* as a form data (application/x-www-form-urlencoded or multipart/form-data). A provider will document how they deliver it (or even give you a choice in the matter).

Netatmo provides a webhook function see Figure 5 below.

SAVE

## TECHNICAL PARAMETERS

## Redirect URI

## Webhook URL

Only use http (80) and https (443) port for webhook url

## Client id

5dfb6153ad7ab3000f2d0492

## Client secret

mgd5qOBYGYJvImXFruyfugvBVhT6J

## Scope available

read station

read thermostat

write thermostat

read camera

read presence

read homecoach

Request scope

SAVE

Figure 5 – Netatmo Client App Technical Parameter setup – Webhook details

### 4.6.3 Prototype Overview

The following diagram illustrates the high level design of the proposed Sensor Hub Gateway to be developed.

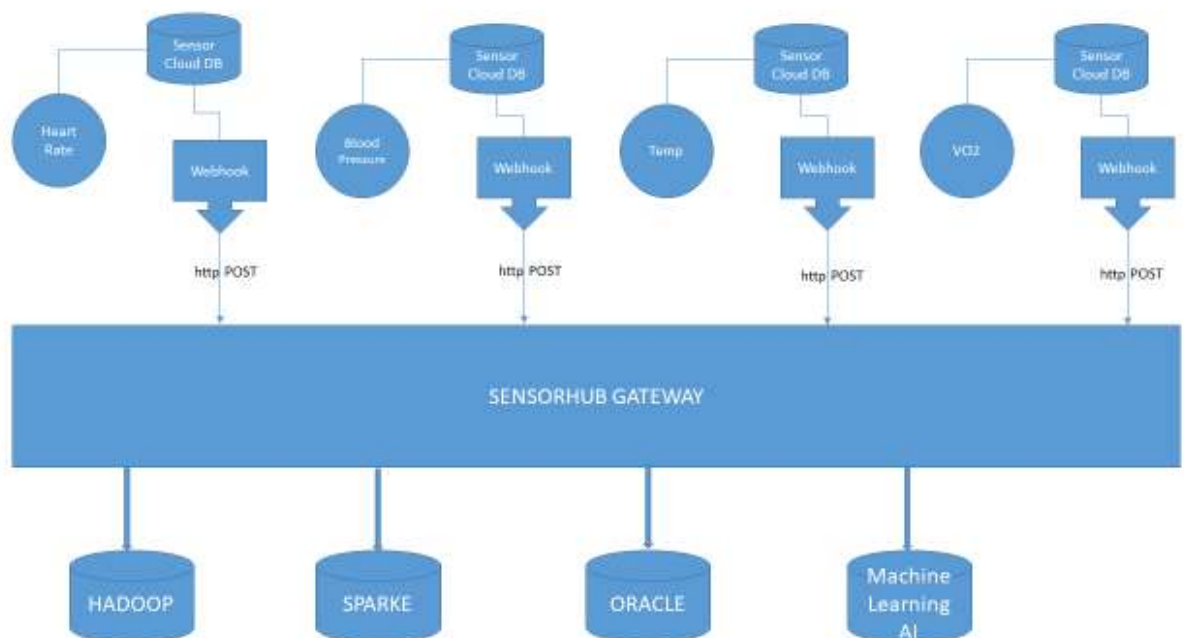


Figure 6 – Proposed Sensor Hub Gateway Overview

#### 4.6.3.1 Spring Framework <sup>[33]-[37]</sup>

The first version of the Spring framework was written by Rod Johnson and was released with the publication of his book *Expert One-on-One J2EE Design and Development* in October 2002. The framework was released under the Apache 2.0 licence in June 2003. Spring Boot is Spring's solution for rapidly creating stand-alone (rapid application development), production quality Spring based applications. It is preconfigured with Spring team's recommended third-party libraries and integrated Spring platforms to minimise configuration and set-up time.

Spring has been chosen as it increases the speed of development of Java based applications, reduces the volume of boilerplate code and increases scalability and productivity. The main goal of Spring Boot framework is to reduce Development, Unit Test and Integration Test time and quickly get a production ready web service application in an Agile project methodology manner.

Other benefits include the integration of Spring with its overall Spring Ecosystem that includes Spring JDBC, Spring ORM, Spring Data, Spring Security and the use of embedded servers such as Tomcat and Jetty. These complementary packages are considered to offer greater extensibility.

Additionally, Spring boot provides a CLI (Command Line Interface) tool to aide development and testing and it supports plugins and build tools such as Maven and Gradle. Spring Boot can use Annotations rather than XML configuration and is considered a simpler and better approach from a configuration and debugging perspective.

Spring is a lightweight framework that supports other frameworks such as Struts, Hibernate, Tapestry, EJB, JSF. The framework comprises several modules such as Inversion of Control (IoC), Aspect Orientated Programming (AOP), Data Access Objects (DAO), Context, ORM, Web MVC and other common design patterns.

Spring's main design patterns IoC and Dependency Injection are used to make the code easier to test and maintain. In particular, for the IoC paradigm has the code loosely coupled, thus there is no need to modify code in a new environment. In the framework the IOC container is responsible for injecting dependencies and instantiating objects (EJB bean) metadata by either XML file or annotations. This ensures the code is easier to test.

#### 4.6.3.2 Advantages of Spring Framework

##### **1. Provides Predefined Templates**

Templates provided for JDBC, Hibernate, JPA – therefore there is no need to write too much code and abstracts away the basic steps of these technologies

##### **2. Loose Coupling**

Spring applications are loosely coupled because of dependency injection.

##### **3. Easy to Test**

Dependency Injection makes it easier to test an application and the framework doesn't require a server.

##### **4. Lightweight**

Spring framework is lightweight because of its POJO implementation. The Spring framework doesn't force the developer to inherit or implement any interface and is considered non- invasive.

##### **5. Faster Development**

Dependency injection supports various frameworks and eases the development of J2EE applications.

##### **6. Powerful Abstraction**

Powerful abstraction to J2EE specifications such as JMS, JDBC, JPA and JTA.

##### **7. Declarative Support**

For caching, validation, transactions and formatting.

#### 4.6.3.3 Spring Architecture

Spring framework provides a number of features which are required for developing an enterprise application. However, it does not enforce the developers to integrate their application with the complete framework. The various features provided by Spring framework are categorized in seven different modules. Developers may choose to integrate their application with one or more Spring modules depending upon the features they want to use in their application.

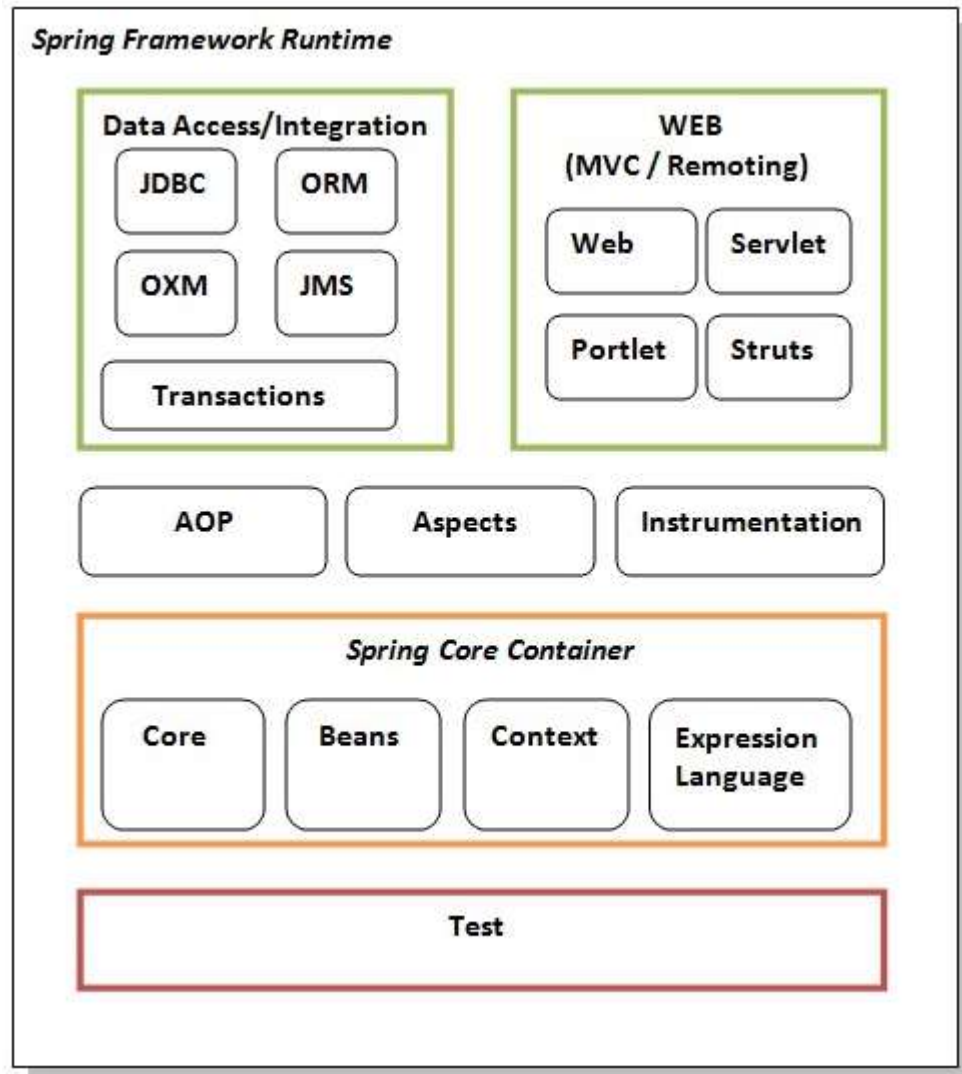


Figure 7 – Spring Framework Architecture.

## Core container

The core container is the heart of Spring framework and all other modules are built on top of it. It provides the dependency injection feature, also is known as inversion of control. This module contains the BeanFactory (an implementation of factory pattern) which creates and manages the life cycle of the various application objects (known as beans) defined in the Spring bean configuration file.

## Application context

This module provides various enterprise level services *internationalisation (i18n)*, scheduling, JNDI access, email etc.

## AOP



This module helps in implementing the various cross cutting concerns in the application like logging, transaction management etc. These concerns are decoupled from the application code and are injected into the various point cuts through configuration file.

### **Spring web**

Spring framework helps in developing web based application by providing the Spring web module. This module is built on top of application context module and provides web oriented features.

### **Spring MVC**

Spring MVC module is built on top of Spring web module and helps in developing web application based on MVC design pattern.

### **Spring DAO**

Almost every enterprise application needs to interact with the database. Spring DAO module makes it easy to interact with database by providing an abstraction over low level JDBC tasks like creating a database connection, release it etc.

### **Spring ORM**

There exist a number of popular object-relational mapping tools like Hibernate, iBatis, JPA etc. Spring ORM module helps in integrating with these tools.

#### **4.6.3.4 Inversion of Control (IoC)**

A typical java based enterprise application consists of a number of java classes. To accomplish its designated functionality, each java class may depend on one or more other java classes. These other java classes are known as dependencies of the another java class. Generally, each class takes the responsibility to obtain the references of the classes it depends upon. This leads to highly coupled application.

The Inversion of Control paradigm is the principle of object-oriented programming, in which objects of the program do not depend on concrete implementations of other objects, but may have knowledge about their abstractions (interfaces) for later interaction.

The Spring framework helps in developing the loosely coupled applications by delegating the responsibility of acquiring the dependencies of a java class to the Spring container and

allowing the java class to focus only on its designated functionality. The Spring container injects the dependencies into the java class as the container is initialized (usually on application start up.)

#### 4.6.3.5 Dependency Injection (DI)

This is the composition of structural design patterns, in which for each function of the application there is a conditionally independent object (service) that can have the need to use other objects (dependencies) known to it by interfaces. Dependencies are transferred (implemented) to the service at the time of its creation. This is a situation where we introduce an element of one class into another. In practice, DI is implemented by passing parameters to the constructor or using setters. Libraries that implement this approach are also called IoC containers.

Dependency injection is also known as inversion of control. Instead of a java class obtaining their dependencies from the container, it is the container who is injecting the dependencies in the java class. So there is an inversion of control.

#### 4.6.3.6 Aspect oriented programming (AOP)

This programming paradigm allows the developer to distinguish cross-through (functional) functionality in application. These functions, which span multiple application nodes, are called cross-cutting concerns and these cross-cutting nodes are separated from the immediate business logic of the application.

In OOP, the key unit is the class, while in AOP, the key element is the aspect. DI helps to separate application classes into separate modules, and AOP helps to separate cross-cutting concerns from the objects they affect.

There are a number of aspects which need to be taken care of during the development an enterprise application. These aspects might spread across the various layer of the application. For example, logging, transaction management, exceptional handling, performance monitoring etc. Spring help us in implementing these aspects by providing AOP framework.

Simply put together, the Spring AOP framework hijacks the execution of the program and injects additional features typically before, after or around method execution.

An example would be a logging aspect. The requirement is to

1. log an entry message before executing the method body
2. log an exit message after executing the message body
3. log the time taken by the method to complete the execution.

Here, logging is known as an *Aspect*. Method execution is known as the *Join point*. The piece of code which logs the entry message, the exit message and the time taken to execute the method are known as the three *advices*. The list of methods on which this behaviour is required is known as *Point Cuts*. And finally the java objects on which this aspect is applied are known as *Targets*.

There are a number of Advice available in AOP framework. These are:

1. *Before advice* – Run before the join point (method execution)
2. *After returning advice* – Run if the join point executes normally.
3. *After throwing advice* – Run if the join point throws an exception.
4. *Around advice* – Run around the join point (method execution)

So, Logging the entry message can be implemented by Before advice, Logging the exit message can be implemented by After advice and logging the time taken for method execution can be implemented using Around advice.

#### 4.6.4 Google App Engine

Google App Engine (GAE or App Engine) is a Platform as a Service (PAAS) and cloud computing platform for developing and hosting web applications in Google data centres. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications as user requests increases. App Engine automatically allocates more resource for the web applications to handle the additional demand.

App Engine supports a variety of programming languages including Python, Java, Go, PHP and Node.js. Google App engine is intended to be language independent and there's are plans to support more languages in the future.

Google App Engine supports many Java standards and frameworks – a major component being the servlet 2.5 technology using the open-source Jetty Web server and other associated Java technologies.

The Spring Framework works with GAE although Spring Security module requires some workarounds.

The inherent compatibility of Google App Engine with the Java technology stack and in particular Spring made it an obvious choice to be considered as the target Cloud based platform to choose for this project, when looking for a scalable IoT application. The potential volumes of data generated by sensors and also the fundamental aim of the proposed Sensor Hub gateway – to port additional sensor data streams to generate bespoke data mining – requires significant up-scaling and extensibility. Therefore, the GAE cloud platform is the obvious choice to build this project's prototype.

#### 4.6.5 Thymeleaf

Thymeleaf is a Java template engine that can be implemented in web servlet and non-web environment. It is a template engine for XML, XHTML and HTML5 within the view layer of a MVC web application and is particularly useful as it integrates fully into the Spring framework.

### 4.7 Programming Languages

The software developed for this project will be based on the Java J2EE technology stack. The main reasons Java was chosen are as follows:

- Java is open source, well documented, ubiquitous and continually developing
- Java is objected-orientated and highly modular and therefore is considered highly extensible
- The runtime environment is platform agnostic and can be implemented into cloud based platforms using Google App Engine.
- The Java language and platform is highly extensible from a User interface perspective and there are many mature UX technologies and tools.
- Java integrates well with Javascript and JSON therefore finding tools for the Web User interface compatible with most browsers will be simple.
- Java is a strongly typed language and lends itself to good object orientated design and well -structured code.
- Many tools, frameworks and platform are available within the Java technology stack.
- Java aligns with the extensible requirement identified in Section 3 – a chief design goal of the software.

## 4.8 Data Exchange Format

The following criteria were considered:

1. Format should be human-readable to simplify testing
2. Libraries for the format should be available for the chosen programming languages to enable easy integration with the code
3. The format should be lightweight to reduce resource usage
4. Format must suit the type of data being transferred
5. Other commercial examples

As previously discussed in the review, JSON and XML are the two mature formats with open standards and both have extensive libraries available for Java and Javascript. The data being transferred is simple in nature consisting of mainly text without much structure. The additional flexibility and security provided by XML is not required and processing would incur more overheads than the lightweight counterpart JSON. Therefore, JSON was used as the format for exchange of data.

## 4.9 Design Patterns

One of the main attractions to using Spring is it comprises Spring Boot, that is intended to be used in a Rapid Application Development manner, and therefore lends itself to Agile methodologies. Additionally, Spring Boot MVC (Model View Controller) provides a framework for developing web based user interfaces with an abstracted and reduced coding content. The main objective of the MVC design pattern is to separate concerns to facilitate testing and to make application code easier to isolate and maintain (Kanjilal, 2015). By using Spring Boot and the MVC design pattern, it is hoped that the developed software will be well structured, complicated functionality will be easily maintainable, the code base will be extensible and the presentation layer can be readily replaced. Overall, the software should facilitate the development of highly cohesive modules with minimal coupling. (Gamma & Helm et al, 1995).

## 4.10 Integrated Development Environments

Eclipse Photon Java JRE/JDK version 1.8

## 4.11 Piecewise Aggregate Approximation (PAA)

Piecewise Aggregate Approximation (PAA) is a very simple dimensionality reduction method for time series mining. It minimizes dimensionality by the mean values of equal sized frames, which misses some important information and sometimes causes inaccurate results in time series mining.

## 4.12 Functionality & Domain Modelling

### Entities

*Table 2 - Measure*

Attribute	Data Type
id	integer (Auto increment, primary key)
time_utc	integer
step_time	integer
temp	decimal (6,1)
humidity	decimal (6,1)
CO2	decimal (6,1)
noise_db	decimal (6,1)

*Table 3 - AggMeasure*

Attribute	Data Type
id	integer (Auto increment, primary key)
time_utc	integer
step_time	integer
temp	decimal (6,1)
humidity	decimal (6,1)
CO2	decimal (6,1)
noise_db	decimal (6,1)
aggTemp	decimal (6,1)
aggHumidity	decimal (6,1)
aggCO2	decimal (6,1)
aggNoise_db	decimal (6,1)

## 5 Evaluation

### 5.1 Resulting System

The essential goal of the project was to develop an extensible software Sensor Hub gateway for use in co-ordinating and presenting sensor data for contextual analysis. The aim was to use an Agile iterative project methodology coupled with good design and coding standards. The developed Sensor Hub cloud gateway that was implemented satisfied three of the five pertinent Use Cases specified in the Term of Reference for this project (see Appendix G for Sensor Hub code & TOR).

Three successful iterations of the Sensor Hub prototype have been completed, developed in an Agile manner in line with recent Software Development Lifecycle methodology. The Agile methodology was particularly useful in this exploratory project due to the nature of the multiple aspects encountered and disciplines required to develop this prototype. These aspects included: -

- Diverse but non-specific literature review that identified much discussion on the use of sensors for the application of eHealth but almost no practical applications disclosed;
- The initial complexity in capturing the sensor data after choosing to use the Netatmo Smart home device;
- Steep learning curve required to develop the software using Spring Boot;
- The configuration necessary to successfully develop a Spring Boot application and enable it for use in the Google App Engine;
- The large scope initially targeted within the Term of Reference, that combined IoT sensor data capture, data aggregation, contextual analysis and current Machine learning technologies for use in eHealth has proved a significant breadth of work.

#### *Use Case 1*

Node Red investigation was unsuccessful. The available flows did not appear to produce any sensor readings for the Netatmo Smart Home device. See Appendix A.

#### *Use Case 2*

The Sensor Hub prototype is able to consume Sensor data from a Cloud based REST API that comprised all CRUD methods. It comprises a single REST Controller (MeasureController) that is specific to the Netatmo sensor data feed and the measure POJO representing the incoming sensor data unit.

The data has been added to the cloud hosted database using a Post method of the REST API (via Postman). The JSON string of Netatmo Sensor was sourced by retrieving data using a Get method via Postman using OAuth authentication and the device credentials (id, client secret and password) see Appendix B.

The application is considered a component Microservice implemented by REST API. Microservices is an architecture style used to build large scale applications that are independently scalable. Microservices architecture is essentially a way to structure a suite of smaller applications that can work together if required or function independently. Microservices follow the loosely coupled design patterns similarly inherent in the Spring framework and MVC. A Microservice is a small, focused piece of software that can be developed independently, deployed and updated.

REST or Representational State Transfer (defined by Fielding (2000)) is an architectural style at a lower more granular application level. It provides standards between applications and systems, making it easier for system communication particularly between decoupled Microservices. REST provides no restriction for the data exchange type, Transport is via HTTP and a transaction should be Stateless. As data is not tied to methods and resources, REST can handle multiple types of call and return bespoke data types with the correct implementation of hypermedia.

The following URLs endpoints provide methods as follows: -

<https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/>

- getAllMeasures() @GetMapping



- createMeasure() @PostMapping

<https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/{id}>

- getMeasureById() @GetMapping
- deleteMeasure @DeleteMapping
- updateMeasure @PutMapping

### *Use Case 3*

The sensor hub prototype was enhanced to include class AggUtilMeasures (a helper class (public static)) that has been developed to implement a Piecewise Aggregation Approximation function. The function operates by receiving a list of measure objects and for each data attribute a Piecewise Aggregate Approximation (PAA) is performed to reduce the dimensionality of the sensor feed. This reduces the “noise” from the data feed and removes any unwanted result anomalies. An AggMeasure Pojo was developed as a data transfer object to be used in the View component of the application. This demonstrates how cloud based data can be analysed for contextual awareness.

Currently the dimensionality factor is hard coded to 100. A future enhancement to the Sensor Hub is to initially provide a Global static constant followed by a configurable user entry dimensionality factor from the UI menu screen.

See Appendix F for Aggregation Plots – red lines

### *Use Case 4*

The Sensor Hub prototype was enhanced with the UIController class to manage the Data presentation layer. The UIController class maps the user requests to required aggregation functions for each sensor data attribute using REST style URLs. Each method returns a reference to the required Thymeleaf display template and the requested Data Transfer Object (AggMeasure). Canvas.js is utilised to display the returned DTO object in a chart form and is processed using javascript within the html page. This demonstrates how a templating tool can be implemented within the Sensor Hub View layer to provide graphical representations of any aggregated data or algorithms on the server side.

See Appendix F for UI charting results

The resulting prototype Sensor Hub MVC design pattern overview and Application are represented in Figure 8 and Figure 9 (see below).

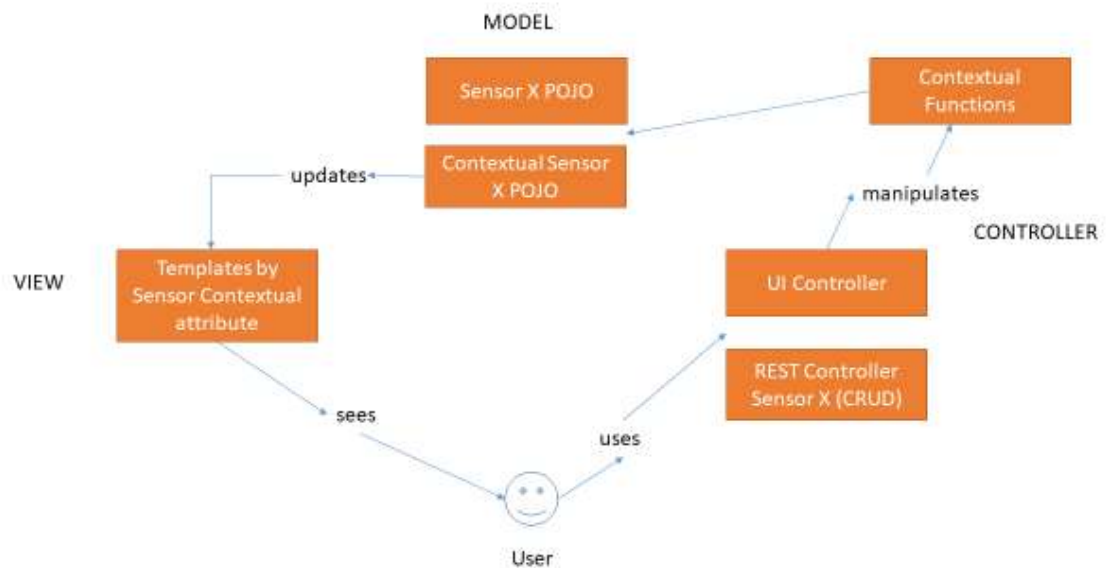


Figure 8 – Resultant Sensor Hub conceptual MVC Design Pattern

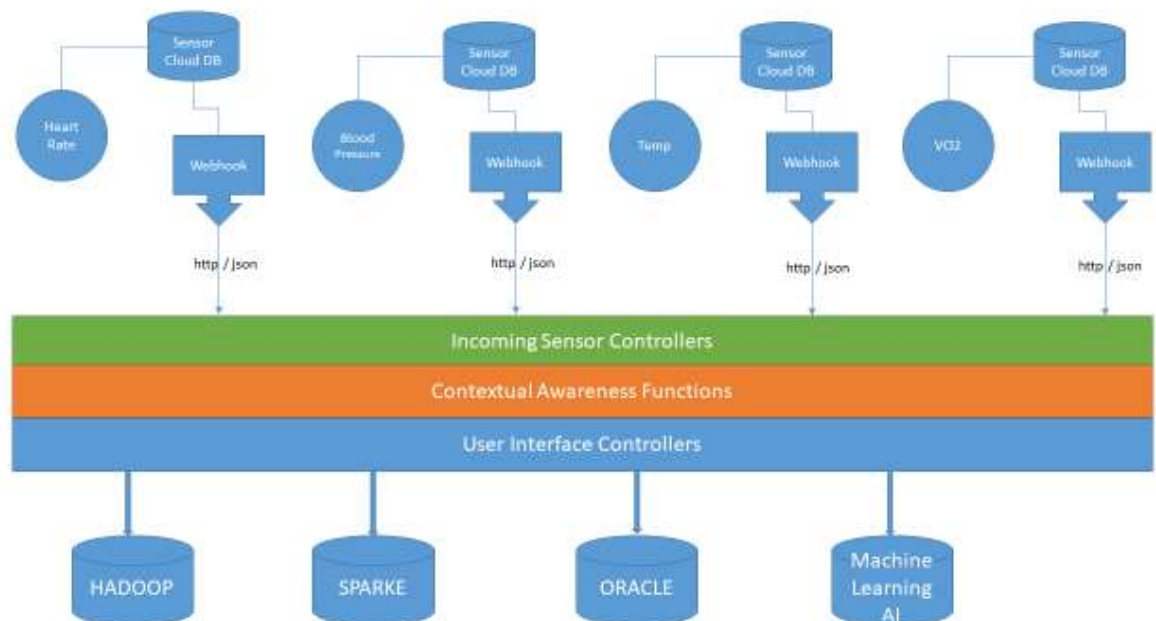


Figure 9 – Resultant Sensor Hub Gateway

#### Use Case 5

Not attempted - Future Project work

#### *Use Case 6*

Not attempted - Future Project work

## 5.2 Critical Analysis

### *5.2.1 SpringBoot MVC*

Spring Boot MVC proved itself to be an excellent choice as a base for providing the Sensor Hub gateway prototype. After initial configuration issues, the level of code abstraction and annotation use proved invaluable in reducing the amount of boiler plate code.

The first prototype iteration (Use Case 2) proved a good approach to gaining familiarity with Spring Boot whilst implementing the relatively small REST API to enable the Netatmo Sensor data capture and storage.

The Hibernate and JPA repository interface implementation were particularly simple and trouble free and resulted in a very simple Data Access Object (DAO) for the required CRUD functions. When implementing these repository libraries and methods, a developer soon begins to think in terms of data objects and POJO rather than the inherent attributes within an object, made possible using the `@Autowired` annotation.

The application implemented a simple data repository design pattern (see Appendix C) whereby the `MeasureRepository` interface is instantiated in the DAO object. A more abstracted version of the data repository pattern was implemented in an earlier iteration of the Sensor Hub prototype (see Appendix D –Sensor HUB). This implemented a service package that contained a `MeasureService` interface and an `MesureServiceImpl` (implementation) class. This application relied on using a `jndiTemplate` for JDBC connection and a configured Tomcat server's `context.xml` to create a JDBC connection. It proved very difficult to configure the `context.xml` to make a JDBC connection and therefore the aforementioned project structure/data repository pattern was adopted (see Appendix C).

This repository design pattern provides two main benefits:

1. The pattern abstracts the data store and enables it to be replaced without changing the business code;
2. The repository improves reusability of application persistence by implementing the persistence logic (queries) in one location. Therefore, it is easier to locate and maintain.

The JDBC data connectivity proved relatively simple and coding was minimal. It is implemented through the `application.properties` file held in the resources folder. The only problem encountered here, on building the project, was the actual syntax of the configuration itself and interpreting the errors displayed on the Eclipse console. Particular care is required when obtaining configuration settings. There is a useful source of information regarding this on the Spring and Baeldung websites.

Configuring Hibernate within Spring Boot also proved relatively easy with minimal configuration required in the `application.properties` file. The configuration consisted of four lines specifying the database instance url, the driver class name, the username and the password.

## 5.3 Problematic Issues

### *5.3.1 – Maven, POM, Eclipse versioning*

A recurring problem was encountered around the use and build of the POM file (Project Object Model). It became apparent that the version numbers of dynamically loaded JAR files need to be very specifically allocated. The more current versions are incompatible with older versions. A large number of working days can be lost attempting to resolve some of the version compatibility issues. Spring have introduced an online tool – Spring Initialize – that is aimed at quickly bootstrapping a project up. Overcoming these issues can still be problematic.

For iteration 1 (Use Case 2) the version of the `spring-boot-starter-parent` was Release 1.5.9 - this version was used to successfully deploy the REST API to the Google App Engine. Unfortunately for Iteration 2, (Use Case 2 – 4) the decision was taken to upgrade to Release 2.1.5 and this proved very problematic. It became apparent that the Maven Build in combination with Eclipse has some configuration issues that prevents the local version of the Sensor Hub being deployed to the Google App Engine cloud. On examination of similar problems online, it was clear that creating a Maven Project using Eclipse IDE causes issues with Project Facets. This prevents the Maven Project successfully deploying to the cloud. Online recommendations point toward creating a Standard Dynamic Web Project then converting to a Maven App engine type project to configure the correct versioning of Project Facets. Other issues that caused problems involved conflicting JAR libraries that are inherently pulled into the project using MAVEN. A number of JARS are common to a

number of the Spring artefacts and therefore the resultant conflicts prevented the application from being deployed to the GAE. The Sensor Hub prototype can be successfully run as a Java application and deployed locally (see Appendix F for Sensor Hub UI screenshots).

A number of other issues were encountered whilst deploying the Sensor Hub application to the cloud and resulted in significant effort trawling the internet to find a resolution. The first issue being the use of the Maven specified spring-boot-devtools library - a tool that allows View layer template amendments to be automatically implemented without the need to re-build the project. This was a recommended Jar library generated within the POM file using the Spring.initializr configuration tool but without the tag `<optional>true</optional>` the application could not be deployed to the GAE.

Similarly, on deploying the application to GAE, days were lost resolving a “Cannot determine embedded database url for database type NONE” error. On an obscure Google groups message board the following application.property setting was discovered – `spring.cloud.gcp.sql.enabled=false`. It was ascertained that when deploying to the GAE, the default embedded database within an application is expected to be a cloud based Google instance rather than an external server located database. External database servers (whilst working correctly locally) cannot be configured without the aforementioned setting.

It is thought an Eclipse version upgrade and use of the most current version of Spring Boot may alleviate the IDE and GAE compatibility issues. Potential future work and investigation.

### *5.3.2 Netatmo Proprietary limitations*

The choice of the Smart Home Netatmo sensor proved challenging. The main issue was sourcing a continuous data feed from the Netatmo data repository. The online Technical disclosed various URL endpoints to build applications at this url: -

<https://dev.netatmo.com/resources/technical/reference/smarthomeapi>

A high level overview of the APIs available is illustrated in Figure 10 below.

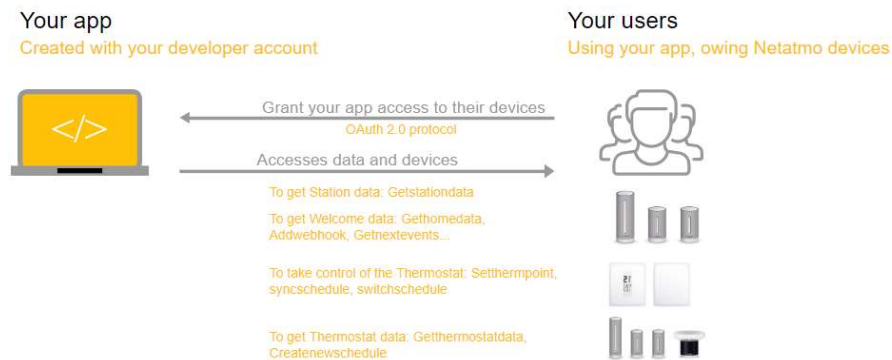


Figure 10 - Netatmo API functionality overview

Unfortunately, despite contacting the Netatmo technical support, it was clear the only functioning Endpoint was getmeasure for the Smart Home sensor. After exploring the technical online literature and through experimentation it was clear the getmeasure endpoint had further practical limitations.

The limitations were as follows: -

- An Authorisation code had to be generated firstly using a client id and client secret provided using the online App create function;
- The Authorisation code allowed an OAuth security token to be generated by Netatmo to gain access to the sensor data;
- The token is only active for 3 hours;
- Requests for data are limited to 1004 readings;
- Requests can be predicated using date ranges but the time format is UTC which is difficult to provide meaningful understanding unless it is converted;
- The Webhook functionality appears to only function on the camera related endpoints although the technical support team suggested the getmeasure endpoint was available.

It became clear that Netatmo data access was designed for implementation in a Mobile application or an application requesting small discrete ranges of data, rather than as a potential data stream. This was highlighted as the Java SDK was actually a relatively old Android SDK.

Using PostMan and the REST API developed for Use Case 2

(<https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/>) a large repository of data was captured by using Get Http requests and Post Http Actions. This enabled the prototype Sensor

Hub Gateway to be rigorously tested for Use Case 2 and the UI aspect of the application to be developed for Use Case 3.

The Webhook functionality offering, although not suitable for this project's purpose, prompted an exploration of its use in IoT applications.

#### *5.3.3 – Webhook definition*

There appears to be some ambiguity on how Webhooks are implemented at the sensor data provider Netatmo. The online literature indicated that they are defined as web callbacks or HTTP push APIs. They are considered a method of delivering data to other applications by using a http POST request.

Netatmo were contacted for advice on whether their defined getMeasure method was available as a Webhook Post service and their response suggested it was. It is suspected, as indicated through test results, that the getMeasure method is not available as a http POST Webhook. Technical support also indicated that an OAuth token request was required before obtaining the data from a Webhook. This is therefore not a Post Action as defined by a Webhook reverse API.

Testing (Netatmo logs) revealed that the Webhooks are only available for the Camera Scopes and they simply post responses when proximity detection occurs.

### **5.4 Discussion**

#### *5.4.1 Design Patterns for Sensor Hub*

The combination of Spring MVC, Hibernate, JPA, Thymeleaf and annotated REST controllers has produced an elegantly structured and particularly lightweight code base on which to further develop the Sensor Hub application.

The application implements a Spring Data JPA repository (as an Interface) that uses composition to instantiate a Data Access Object (DAO). The DAO implements the Hibernate style CRUD methods and therefore there is no standard SQL statements found within the overall Database/Integration tier. This abstraction of the integration coding and SQL provides the benefit of removing any real database knowledge from a developer. Configuration for database connectivity is relatively simple within the application.properties files. The end result is an elegant, succinct and very simple Database layer.

The application uses Thymeleaf and Spring MVC to build the presentation tier and the result is a strictly Model 2 architecture Model View Controller design pattern. Strictly speaking a

true MVC design pattern is a push model whereby Model changes are propagated to the View using the Observer pattern as disclosed by Johnson et al. (2005) page 427.

Particularly useful is the implementation of the `UiController` class that implements Spring MVC `ModelMap` and `Model` types to transport data transfer objects (POJOs). Transfer occurs directly to the template html view layer (held in the `resource/templates` (functionality provided Thymeleaf)). The Model and template reference being stored in a Map object provided by the Spring `ModelAndView` type. The resulting code using these frameworks and libraries is very lightweight, easily readable and lends itself to strictly adhering to ideal MVC design patterns.

The resultant application presentation tier is very extensible using MVC and Thymeleaf. The application has been built using Canvas.js to render the aggregated data POJOs into line graphs and this was found to be easily implemented and easy to understand.

It is clear that there are number HTML Charting libraries and APIs available to build Business Intelligence Dashboards. The Spring MVC framework (and its inherent design patterns) can be utilised within the current project to easily add more data / aggregated POJOs to the presentation tier.

One limitation of the Spring `ModelAndView` type is that it only supports a single data transfer object. This enforces the use of a single generic object that contains all aggregated data members. Any new data analysis attributes would require an updated or expanded POJO rather than potentially new POJOs. This could result in unwanted changes in the view layer and external database repositories when creating additional attribute level enhancements.

Canvas.js libraries integrated well and offered a variety of charting styles that can be tailored for eHealth data aggregation display. The coding was done in Javascript and involved parsing the DTO JSON object (`AggMeasure`) to enable data visualisation in the browser.

The Business tier of the application comprises a Sensor specific REST Controller (`MeasureController`) and an Aggregation function class to demonstrate simple Piecewise Aggregation Approximation as a simple analysis tool.



#### *5.4.2 – Contextual Aggregation Function*

The Sensor Hub application was enhanced during Iteration 3 by the addition of the `AggUtilMeasures` class. This was to satisfy Use Case 3 – the requirement to provide an aggregation function.

This class implemented a simple Piecewise Aggregation Approximation (PAA) algorithm to smooth and calculate a mean over a given dimensionality factor. The purpose of this simple aggregation technique is to demonstrate a simple Contextual analysis function that could be utilised to provide potential “Smart” proactive functional programming paradigms. This simple PAA algorithm is typically used to enable outlier results or readings to be quickly detected (see Appendix F for resulting charts containing raw and aggregated data). As such, simple comparison between the raw data and the PAA data can be used to set bounds that when broken can be used to trigger warning functions, typically through the use of Observations design patterns.

The algorithm itself was performed on a transferred List of measure objects – `List<Measure>`. The `List<Measure>` object is looped through and within the dimensionality factor a running total is calculated. The running total per dimensionality factor is stored in a separate `PaaMeasure` object. When the `List<Measure>` has been processed and the `PaaMeasure` object fully populated, the `List<Measure>` and `PaaMeasure` object are used to populate a new `List<AggMeasure>`. This is achieved by iterating through the `List<Measure>` array and simultaneously building the `List<AggMeasure>` by enhancing the aggregation attributes from the recently instantiated `PaaMeasure` object. Multiple arrays are iterated over and a background in COBOL and procedural programming languages proved useful.

The developed PAA aggregation function could be used as the basis to further develop a Symbolic Aggregation Approximation (SAX) function but it is considered that the Netatmo sensor data did not required further “silencing” or dimensional reduction. The raw data plotted against the PAA aggregates provided sufficient contextual information within the scope of this project.

Currently the dimensionality factor is hard coded but ideally could be dynamically captured from a user or stored as a Global constant.

#### 5.4.3 Usability & User Acceptance Testing

To date, only unit testing has been performed on the Sensor Hub application. User acceptance testing has not been performed. Appendix F contains a screenshot of the Sensor Hub REST API that is hosted on GAE and also the fully integrated application (REST API + UI + Data Aggregation functions) operating locally.

### 5.5 Future Project Work

It is considered that the developed Spring Boot MVC Sensor Hub implementation does provide an extensible method of co-ordinating multiple sensor data streams. It has been identified that each data stream will need the following components to extend similar functionality to other Health sensor data feeds:

1. Incoming Sensor Data POJO detail;
2. Specification of the incoming JSON http POST request to enable request parsing;
3. Sensor data specific REST Controller to manage, collate and parse incoming data;
4. Aggregated Data POJO based on required Data Analysis functions;
5. User Interface Controller to manage the object data transfer to the View layer;
6. Sensor data to be supplied using webhook (http POST).

A considerable level of experience has been developed within the Spring Boot implementation that would enable a new sensor feed to be quickly added to the application. Sourcing an eHealth sensor that provides cloud based webhook functionality would be an area of future investigation.

Other future investigation to be considered would be the development of a generic interface specification to consume any sensor's data. An application or sensor that could register itself and supply a data stream without additional code development, would be ideal. After an investigation of how to parse unknown JSON messages <sup>[31]-[32]</sup> (using Jackson Tree structure, json, Gson parsing libraries) it is clear that data would be initially stored as strings. The literature review revealed that health data is more likely to be numeric. Therefore, methods to convert the initial string encoded numeric data into useable data types would be necessary. This was considered a significant task and overhead and outside the scope of this project.

The literature review identified multiple health sensor data types from heart rates, step counts, blood pressure, ECG measurements *inter alia*. Thus, another eHealth data source would be

the next logical proposed Project Iteration. A newly introduced data feed would require an investigation centring on how the data can be used or aggregated to provide insight and context. The aim of adding new sensor feeds would be to create a baseline eHealth Contextual data analysis methodology to provide timely, scalable and useful UI data representations.

Another suggested iteration would be the completion of Use Case 6 to implement an Apache Spark data repository. An investigation of how to implement data aggregation functions in a cloud based analytics engine such as Spark is necessary to investigate applying machine learning algorithms against large eHealth datasets. An online Databricks seminar revealed that Spark uses in memory caching to provide fast performance for complex SQL queries. These SQL queries are integral in Machine Learning and AI. This would be a more appropriate phase once a functioning Sensor Hub application has multiple sensor data feeds at sufficient stored volumes. Once implemented, this would enable complex data algorithms to be run at high speed offering excellent usability and broad machine learning functionality to data clinicians. The Machine Learning functions could be generically added to a Contextual Controller package within the application to build bespoke contextual analysis functions and objects.

Finally, analysis of the security and privacy aspect of gathering health data in a cloud based database needs to be assessed. Spring Security is based on Servlet filters and can be implemented by using a simple annotation (`@EnableWebSecurity`). Specific Security classes can be configured and API endpoints secured in a relatively simple and abstracted manner. The development of a Sensor driven Http Post request secured using Spring for consumption by the Sensor Hub would be of great interest.

## 6 Conclusion

Following the literature review, it was clear that very few practical software implementations had been disclosed for use as a sensor node gateway to collate eHealth data feeds. *The main aim of the project was to develop a practical open source application to capture and co-ordinate sensor data feeds in the cloud.* The chosen technology was Spring Boot MVC due to its contemporary nature and its integrated out of the box functionality in areas such as data persistence, security and use of Inversion of control. It is considered that the final Minimal Viable Product (MVP) developed in an Agile methodology has provided valuable insight into the many areas and skills required to create a useful IoT software application. A greater understanding has been gained of the interoperability issues involved in the transmission of sensor data. Additionally, the proprietorial nature of gaining access to a commercial sensor's data was found to be challenging. Obviously, it is not commercially beneficial for sensor suppliers to provide free and easy access to end users.

A further aim of the project was to develop an extensible and scalable application that would be suitable for the addition of more sensor data feeds. It is considered that the Sensor Hub product application and surrounding implementation pathways have provided a basis to satisfy this objective. The use of webhooks needs to be extended and explored to gain more detailed knowledge on how this http post reverse API method can be secured and implemented. From an interoperability point of view, webhooks have been identified as an excellent route to transmit sensor data for collection and collation. The potential of a sensor manufacturer, particularly in the eHealth industry to develop individual user specific webhooks that can be fed into a variety of Sensor Hubs offers an interesting and engaging commercial opportunity. A further opportunity within the already wide and diverse Software as a Service (SAAS) software industry. Perhaps, the next development will be Contextual Analysis as a Service whereby context and understanding can be mined and sent directly to users - further abstracting the technical requirements surrounding the intensive skills and work required to provide this knowledge.

The use of sensors, cloud databases and contextual analysis to provide an eHealth function is an analogue of the IoT. During this project, it is apparent that a very high level of knowledge is required across a wide range of disciplines to produce a functioning eHealth Sensor Hub. The three main disciplines being sensor data capture, cloud repositories and contextual data mining. Each area requires extensive expertise. The most important aspect to gaining understanding

from the captured eHealth data ultimately requires the input of medical knowledge from clinicians and data scientists. A Sensor Hub type application could be used as an effective tool to present data in a meaningful way to aide a clinician's understanding. Collaboration with data clinicians is very important to aid the development of useful data functions and visualisation user interfaces.

The contextual analysis of eHealth data remains a future work consideration. The literature review is silent on the need to incorporate domain understanding by subject matter experts to enable contextual analysis. Contextual analysis is not a function that is magically revealed once data is captured, it is the result of experts applying their deep knowledge to data. Contextual analysis is enhanced when data is presented in a variety of manners to reveal more understanding. Furthermore, it is a collaboration between data collection and domain expertise within any IoT domain that ultimately reveals context. A Sensor Hub gateway that offers extensible User Interface tools can be an excellent way to bring the technology and the experts closer to the data.

Spring Boot MVC has proved itself to be a good choice/option to develop the Sensor Hub application. It is designed to be View agnostic and enables the incorporation of external view technologies not bound to a `HttpServletRequest`. This is achieved using the `ModelAndView` type that holds a `Map` that couples a returning `Model` to a view (or a template reference as a view). This enabled a simple data aggregation interface to be developed relatively quickly. It was achieved by incorporating `canvas.js` charting libraries to chart and display the application aggregation functions effectively. A variety of charting options are available using `canvas.js`.

The Spring Boot framework proved itself to be a useful tool for implementing scalable and extensible functionality in the cloud based application arena with the one proviso that it is difficult to implement into the Google App Engine using eclipse as an IDE.

Finally, a report in the Telegraph <sup>[39]</sup> this week describes a London technology start-up company, MySense, trialling sensor and wearable technology in retirement homes. The application harnesses AI to learn residents' activity patterns. The application aims to identify whether a resident has fallen or has immediate medical needs. The installed "Smart" sensors measure temperature, movements and heart rates to allow residents to maintain their independence, prevent serious preventable health issues and detect falls. Potentially saving the NHS £2Bn a year.

## References

- [1] Abowd, G. D. and Mynatt, E. D. "Charting past, present, and future research in ubiquitous computing," *ACM Trans. Comput.-Hum. Interact.*, vol. 7, pp. 29–58, March 2000.
- [2] Babu, S., Chandini, M., Lavanya, P. Ganapathy, K., and Vaidehi, V. "Cloud-enabled remote health monitoring system," *Int. Conf. on Recent Trends in Inform. Tech. (ICRTIT)*, July 2013, pp.702-707.
- [3] Chondamrongkul, N. (2017). Personalized healthcare with context-awareness platform. *2017 International Conference on Digital Arts, Media and Technology (ICDAMT)*.
- [4] Dey, A. K. Abowd, G. D. and Salber, D. "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Hum.-Comput. Interact.*, vol. 16, pp. 97–166, December 2001.
- [5] Feng, J., Sedano, T., "Comparing extreme programming and Waterfall project results", *24<sup>th</sup> IEEE-CS Conference*, 2011.
- [6] Gantz, J., and Reinsel, D. "Extracting value from chaos." *IDC Iview*, vol. 1142, pp. 1-12, Jun. 2011.
- [7] Groopman, J. , "Why Interoperability holds the keys to the Smart Home", 2016  
<https://www.linkedin.com/pulse/why-interoperability-holds-keys-smart-home-jessica-groopman>
- [8] Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements and future directions. *Future Generation Computer Systems* 29 (2013), 1645 - 1660.
- [9] Honan, G., Page, A., Kocabas, O., Soyata, T., Kantarci, B., "Internet-of-everything oriented implementation of secure Digital Health (D-Health) systems", *Computers and Communication (ISCC) 2016 IEEE Symposium on*, pp. 718-725, 2016.
- [10] Hu, P. Indulska, J. and Robinson, R. "An autonomic context management system for pervasive computing," in *Pervasive Computing and Communications*, 2008. *PerCom 2008. Sixth Annual IEEE International Conference on*, march 2008, pp. 213 –223.
- [11] Jararweh, Y., Tawalbeh, L., Ababneh, F. and Dosari, F. "Resource efficient mobile computing using cloudlet infrastructure," in *IEEE Ninth International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, Dec 2013, pp. 373-377
- [12] Johnson, R. (2003). *Expert one-on-one J2EE design and development*. Birmingham: Wrox.
- [13] Johnson, R. (2005). *Professional Java development with the Spring Framework*. Indianapolis.

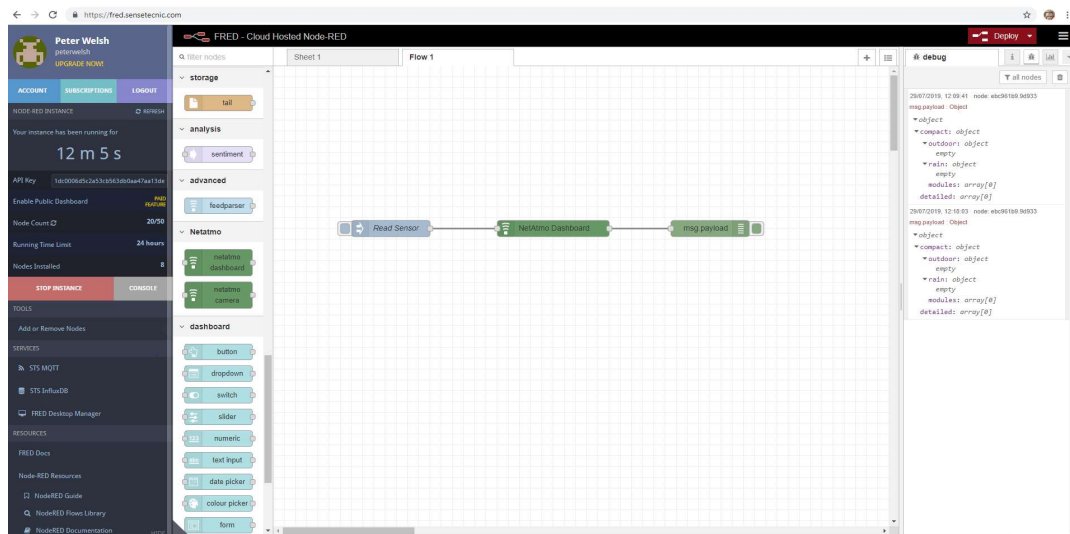
- [14] Kanjilal, Joydip. "Exploring the MVC, MVP, and MVVM design patterns." *InfoWorld.com*, 25 May 2015
- [15] Lakshmanachari, S., C. Srihari, C., Sudhakar, A., Nalajala, P. (2017). Design and Implementation of Cloud based Patient Health Care Monitoring System using IoT. *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS-2017)*.
- [16] Mitrokotsa, A., Rieback M. R., & Tanenbaum, A. S. (2013). Classifying RFID attacks and defences. [cited 2013 May 20]
- [17] Molich, R & Nielsen, J (1990). Heuristic Evaluation of user interface. Proceedings of the ACM CHI 90 Human Factors in Computing Systems Conference, 1990, Seattle, Washington, USA, pp 249-256
- [18] Mostefaoui, G. K., Pasquier-Rocha, J., & Brezillon, P. (2004) 'Context-aware computing: a guide for the pervasive computing community'. In Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on, 19-23 July 2004 Unknown place of publication: Unknown publisher pp. 39-48.
- [19] Noh, Y. (2013) 'A study on next-generation digital library using context-awareness technology', *Library Hi Tech*, 31 (2), pp. 236 – 253
- [20] Quwaider, M. and Jararweh, Y. "Cloudlet-based for big data collection in body area networks," in 8<sup>th</sup> International Conference for Internet Technology and Secured Transactions (ICITST), Dec 2013, pp. 137-141.
- [21] Rajagopalan, R., and Varshney, P., "Data aggregation techniques in sensor networks: A survey" (2006). *Electrical Engineering and Computer Science*
- [22] Royce, W, "Managing the Development of Large Software Systems", IEEE WESTCON, 1970.
- [23] Sakr, S. & Elgammal, A. (2016). *Towards a Comprehensive Data Analytics Framework for Smart Healthcare Services*, *Big Data Research* 4 (2016) 44 – 58
- [24] Torgeir, D., Sridhar, N., VenuGopal, B. and Nils Brede, M. (2012) 'A decade of agile methodologies: Towards explaining agile software development.' *The Journal of Systems and Software*, 85(6) pp. 1213-1221.
- [25] Weiser, M., Gold, R., The origins of ubiquitous computing research at PARC in the late 1980s, *IBM Systems Journal* (1999).
- [26] Wendt, N. and Julien, C. "MASON: an Open Development Contextual Sensing Framework Enabling Reactive Applications", IEEE ACM International Conference on Mobile Software Engineering and Systems, 2016
- [27] <http://www.cloudbees.com>
- [28] <https://52north.org/software/software-projects/sos/>

- [29] <https://awareframework.com/>
- [30] <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT>
- [31] <https://stackoverflow.com/questions/13916086/jackson-recursive-parsing-into-mapstring-object/13926850#13926850>
- [32] <https://stackoverflow.com/questions/19760138/parsing-json-in-java-without-knowing-json-format>
- [33] <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html>
- [34] <https://start.spring.io/>
- [35] <https://www.baeldung.com/>
- [36] <https://www.codejava.net/frameworks/spring/understanding-the-core-of-spring-framework>
- [37] <https://www.javatpoint.com/steps-to-create-spring-application>
- [38] <https://www.baeldung.com/securing-a-restful-web-service-with-spring-security>
- [39] <https://www.telegraph.co.uk/technology/2019/09/09/uk-start-up-wants-help-nhs-save-2bn-year-using-ai-sensors/>



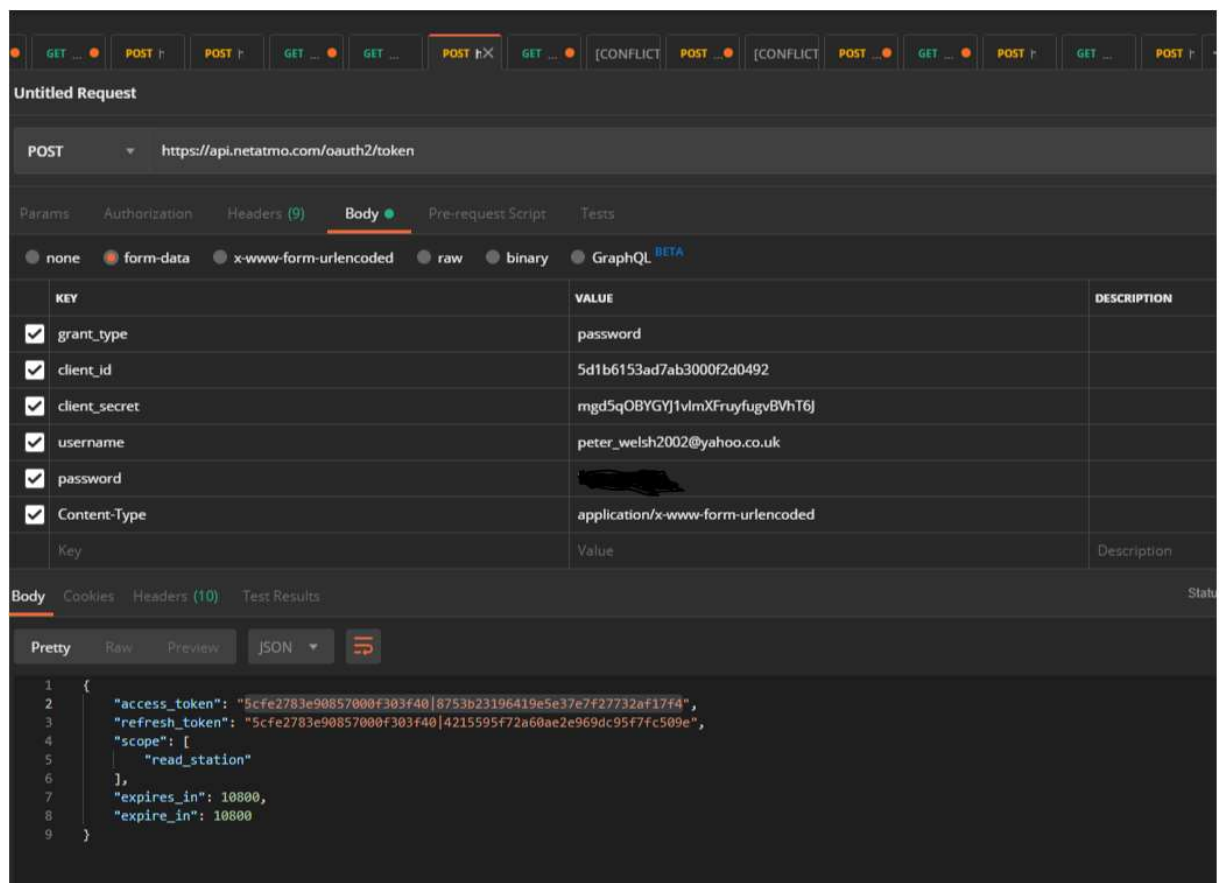
# Appendix A

## Node Red Netatmo nodes



# Appendix B

## Netatmo Get and Post methods using Postman



GEThttps://api.netatmo.com/api/getmeasure?access\_token=5cfe2783e90857000f303f40|df7f479d02e232a25e16ef38911b2483&device\_id=70ee50266776&scale=max&type=temperature,...

Params

AuthorizationHeadersBodyPre-request ScriptTests

Query Params

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	access_token	5cfe2783e90857000f303f40 df7f479d02e232a25e16ef38911b2483	
<input checked="" type="checkbox"/>	device_id	70ee50266776	
<input checked="" type="checkbox"/>	scale	max	
<input checked="" type="checkbox"/>	type	temperature,humidity,CO2,noise	
<input checked="" type="checkbox"/>	real_time	true	
<input checked="" type="checkbox"/>	optimize	true	
<input type="checkbox"/>	limit	1	
	Key	Value	Description

GET

https://api.netatmo.com/api/getmeasure?access\_token=5cfe2783e90857000f303f40|df7f479d02e232a25e16ef38911b2483&device\_id=70ee50266776&scale=max&type=temperatu...

Send

Save

Params

Authorization

Headers

Body

Pre-request Script

Tests

Cookies

Code

Comments





















Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	access_token	5cfe2783e90857000f303f40 df7f479d02e232a25e16ef38911b2483			
<input checked="" type="checkbox"/>	device_id	70ee50266776			
<input checked="" type="checkbox"/>	scale	max			
<input checked="" type="checkbox"/>	type	temperature,humidity,CO2,noise			
<input checked="" type="checkbox"/>	real_time	true			
<input checked="" type="checkbox"/>	optimize	true			
<input checked="" type="checkbox"/>	limit				
	Key	Value	Description		

Response

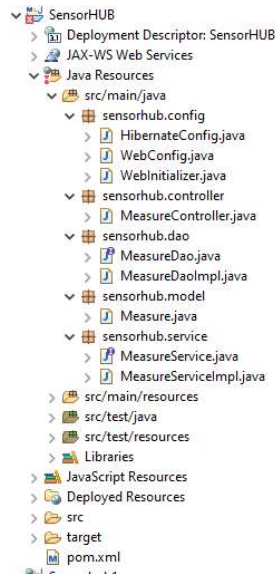
## Appendix C

### Simple data repository design pattern structure

- ✓  sensorhubcloud [sensorhubcloud1:1]
  - App Engine [standard: java8] - appengine-web.xml
  - >  Deployment Descriptor: sensorhubcloud
  - >  JAX-WS Web Services
  - ✓  Java Resources
    - ✓  src/main/java
      - >  com.prw.sensor
      - >  com.prw.sensor.controller
      - ✓  com.prw.sensor.dao
        - >  MeasureDao.java
      - >  com.prw.sensor.model
      - ✓  com.prw.sensor.repository
        - >  MeasureRepository.java
    - >  src/main/resources
    - >  src/test/java
    - >  Libraries
    - >  JavaScript Resources
    - >  Deployed Resources
    - >  src
    - >  target
    - >  pom.xml

# Appendix D

## Data repository with service layer design pattern structure



## Sensor HUB context.xml

```
<Host appBase="webapps" autoDeploy="true" name="localhost" unpackWARs="true">

  <!-- SingleSignOn valve, share authentication between web applications
       Documentation at: /docs/config/valve.html -->
  <!--
  <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
  -->

  <!-- Access log processes all example.
       Documentation at: /docs/config/valve.html
       Note: The pattern used is equivalent to using pattern="common" -->
  <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" pattern="%h %l %u %t &quot;%r&quot;%s %b" prefix="localhost_access_log" suffix=".log" />

  <!-- <Context docBase="SensorHUB" path="/SensorHUB" reloadable="true" source="org.eclipse.jst.jee.server:SensorHUB"/> -->
  <!-- <Context docBase="SensorHUB" path="/SensorHUB" reloadable="true" source="C:\Users\User\workspace2\SensorHUB\target\SensorHUB-0.0.1-SNAPSHOT" -->
  <Resource auth="Container" driverClassName="com.mysql.jdbc.Driver" maxActive="5" maxIdle="3" name="jdbc/springmvc" password="3ofger1b8" type="javax.sql.DataSource" />

  </Context> -->
  <Context docBase="SensorHUB" path="/SensorHUB" reloadable="true" source="org.eclipse.jst.jee.server:SensorHUB"/>
</Host>
</Engine>
</Service>
</Server>
```

# Appendix E

## getMeasure – json response format

```
{
  "body": [{
    {
      "beg_time": 1560160482,
      "step_time": 11,
      "value": [[27, 47, null, 42], [26.9, 47, 1112, null]]
    }, {
      "beg_time": 1560362539,
      "step_time": 304,
      "value": [[20.9, 61, 700, 43], [20.9, 61, 735, 45]]
    }, {
      "beg_time": 1560363148,
      "step_time": 22,
      "value": [[20.8, 61, 741, 53], [20.8, 61, 741, 70]]
    }
  ],
  "status": "ok",
  "time_exec": 0.12193608283996582,
  "time_server": 1563623392
}
```

# Appendix F

## Sensor Hub REST API on Google App Engine

<https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/>

```
https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/

[{"id":3,"timeUtc":1560160482,"stepTime":11,"temp":27.0,"humidity":47,"c02":1112,"noiseDb":42}, {"id":4,"timeUtc":1560160482,"stepTime":11,"temp":26.9,"humidity":47,"c02":1112,"noiseDb":0}, {"id":5,"timeUtc":1560160516,"stepTime":93,"temp":26.7,"humidity":48,"c02":1112,"noiseDb":39}, {"id":6,"timeUtc":1560160516,"stepTime":93,"temp":25.1,"humidity":50,"c02":1154,"noiseDb":39}, {"id":7,"timeUtc":1560160716,"stepTime":220,"temp":24.4,"humidity":51,"c02":1161,"noiseDb":42}, {"id":8,"timeUtc":1560160716,"stepTime":118,"temp":24.0,"humidity":52,"c02":1156,"noiseDb":45}, {"id":9,"timeUtc":1560160851,"stepTime":220,"temp":23.9,"humidity":52,"c02":1156,"noiseDb":0}, {"id":10,"timeUtc":1560160851,"stepTime":220,"temp":23.7,"humidity":53,"c02":1155,"noiseDb":40}, {"id":11,"timeUtc":1560161376,"stepTime":300,"temp":23.4,"humidity":53,"c02":1060,"noiseDb":40}, {"id":12,"timeUtc":1560161376,"stepTime":300,"temp":23.1,"humidity":53,"c02":1052,"noiseDb":40}, {"id":13,"timeUtc":1560161977,"stepTime":300,"temp":23.0,"humidity":53,"c02":1003,"noiseDb":42}, {"id":14,"timeUtc":1560161977,"stepTime":300,"temp":22.9,"humidity":53,"c02":1005,"noiseDb":42}, {"id":15,"timeUtc":1560161977,"stepTime":300,"temp":22.9,"humidity":53,"c02":907,"noiseDb":41}, {"id":16,"timeUtc":1560162605,"stepTime":305,"temp":23.0,"humidity":52,"c02":923,"noiseDb":66}, {"id":17,"timeUtc":1560162605,"stepTime":305,"temp":22.9,"humidity":53,"c02":910,"noiseDb":56}, {"id":18,"timeUtc":1560163211,"stepTime":127,"temp":22.8,"humidity":52,"c02":840,"noiseDb":53}, {"id":19,"timeUtc":1560163211,"stepTime":127,"temp":22.8,"humidity":52,"c02":837,"noiseDb":51}, {"id":20,"timeUtc":1560163642,"stepTime":301,"temp":22.9,"humidity":52,"c02":852,"noiseDb":46}, {"id":21,"timeUtc":1560163642,"stepTime":301,"temp":22.9,"humidity":52,"c02":841,"noiseDb":42}, {"id":22,"timeUtc":1560164247,"stepTime":305,"temp":22.9,"humidity":52,"c02":823,"noiseDb":43}, {"id":23,"timeUtc":1560164247,"stepTime":305,"temp":22.9,"humidity":52,"c02":858,"noiseDb":43}, {"id":24,"timeUtc":1560164856,"stepTime":305,"temp":22.9,"humidity":52,"c02":801,"noiseDb":41}, {"id":25,"timeUtc":1560164856,"stepTime":305,"temp":22.9,"humidity":52,"c02":778,"noiseDb":41}, {"id":26,"timeUtc":1560164856,"stepTime":305,"temp":22.8,"humidity":52,"c02":729,"noiseDb":41}, {"id":27,"timeUtc":1560165766,"stepTime":300,"temp":22.8,"humidity":52,"c02":671,"noiseDb":41}, {"id":28,"timeUtc":1560165766,"stepTime":300,"temp":22.7,"humidity":52,"c02":659,"noiseDb":41}, {"id":29,"timeUtc":1560166371,"stepTime":300,"temp":22.7,"humidity":52,"c02":622,"noiseDb":41}, {"id":30,"timeUtc":1560166371,"stepTime":300,"temp":22.7,"humidity":52,"c02":612,"noiseDb":41}, {"id":31,"timeUtc":1560166972,"stepTime":304,"temp":22.7,"humidity":52,"c02":585,"noiseDb":41}, {"id":32,"timeUtc":1560166972,"stepTime":304,"temp":22.7,"humidity":52,"c02":570,"noiseDb":41}, {"id":33,"timeUtc":1560167581,"stepTime":305,"temp":22.7,"humidity":52,"c02":550,"noiseDb":41}, {"id":34,"timeUtc":1560167581,"stepTime":305,"temp":22.7,"humidity":52,"c02":544,"noiseDb":41}, {"id":35,"timeUtc":1560168186,"stepTime":300,"temp":22.7,"humidity":52,"c02":528,"noiseDb":41}, {"id":36,"timeUtc":1560168186,"stepTime":300,"temp":22.7,"humidity":52,"c02":516,"noiseDb":41}, {"id":37,"timeUtc":1560168787,"stepTime":300,"temp":22.7,"humidity":52,"c02":490,"noiseDb":41}, {"id":38,"timeUtc":1560168787,"stepTime":300,"temp":22.7,"humidity":52,"c02":481,"noiseDb":41}, {"id":39,"timeUtc":1560169392,"stepTime":300,"temp":22.7,"humidity":52,"c02":460,"noiseDb":41}, {"id":40,"timeUtc":1560169392,"stepTime":300,"temp":22.6,"humidity":52,"c02":450,"noiseDb":41}, {"id":41,"timeUtc":1560169993,"stepTime":304,"temp":22.6,"humidity":52,"c02":441,"noiseDb":41}, {"id":42,"timeUtc":1560169993,"stepTime":304,"temp":22.9,"humidity":52,"c02":463,"noiseDb":41}, {"id":43,"timeUtc":1560170602,"stepTime":304,"temp":22.9,"humidity":52,"c02":450,"noiseDb":41}, {"id":44,"timeUtc":1560170602,"stepTime":304,"temp":22.9,"humidity":52,"c02":459,"noiseDb":41}, {"id":45,"timeUtc":1560171207,"stepTime":300,"temp":22.9,"humidity":52,"c02":534,"noiseDb":44}, {"id":46,"timeUtc":1560171207,"stepTime":300,"temp":23.0,"humidity":52,"c02":536,"noiseDb":45}, {"id":47,"timeUtc":1560171812,"stepTime":300,"temp":22.9,"humidity":52,"c02":511,"noiseDb":41}, {"id":48,"timeUtc":1560171812,"stepTime":300,"temp":22.9,"humidity":52,"c02":571,"noiseDb":43}, {"id":49,"timeUtc":1560172413,"stepTime":300,"temp":22.9,"humidity":52,"c02":544,"noiseDb":39}, {"id":50,"timeUtc":1560172413,"stepTime":300,"temp":23.0,"humidity":52,"c02":562,"noiseDb":41}, {"id":51,"timeUtc":156017314,"stepTime":300,"temp":23.1,"humidity":52,"c02":613,"noiseDb":39}, {"id":52,"timeUtc":1560173314,"stepTime":300,"temp":23.2,"humidity":52,"c02":664,"noiseDb":39}, {"id":53,"timeUtc":1560173314,"stepTime":300,"temp":23.3,"humidity":52,"c02":728,"noiseDb":38}, {"id":54,"timeUtc":1560173915,"stepTime":300,"temp":23.3,"humidity":52,"c02":788,"noiseDb":38}, {"id":55,"timeUtc":1560173915,"stepTime":300,"temp":23.4,"humidity":52,"c02":819,"noiseDb":37}, {"id":56,"timeUtc":1560174520,"stepTime":304,"temp":23.5,"humidity":52,"c02":832,"noiseDb":38}]
```

<https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/3>

```
https://sensorhubcloud1.appspot.com/sensorhubcloud/measures/3

{"id":3,"timeUtc":1560160482,"stepTime":11,"temp":27.0,"humidity":47,"c02":1112,"noiseDb":42}
```

## MVP SensorHub Gateway application - local

<http://localhost:8080/>

Sensor Hub Gateway Measures

localhost:8080

Sensor Hub Menu

Measures

Sensor Measure 2

Blood Pressure

Search

Search

The demonstration vehicle

Measurements

way - Gimme readings!!!! powered by Spring Boot.

View All

Plot Temperature

Plot Aggregate Temp

Plot CO2

Plot Aggregate CO2

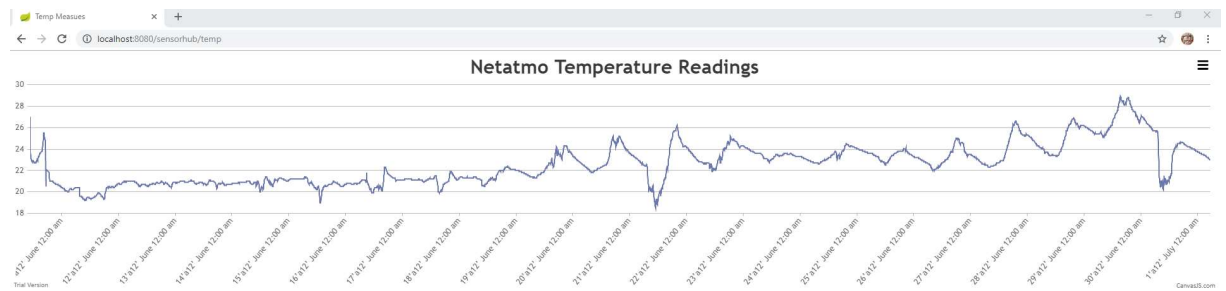
Plot Noise

Plot Aggregate Noise

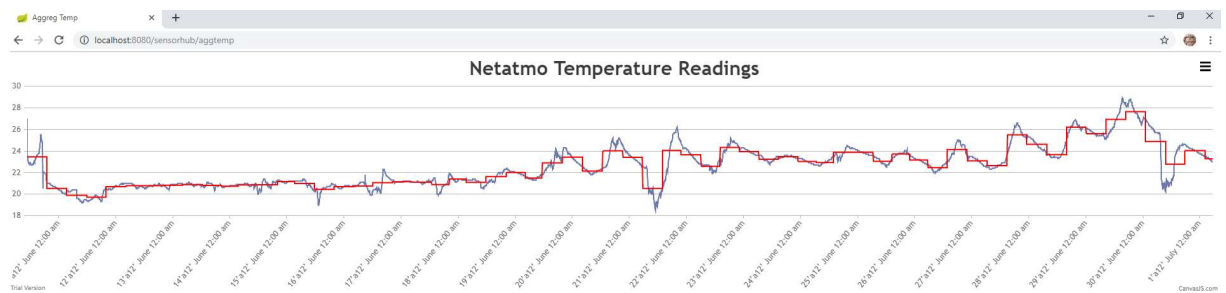
Plot Humidity

Plot Aggregate Humidity

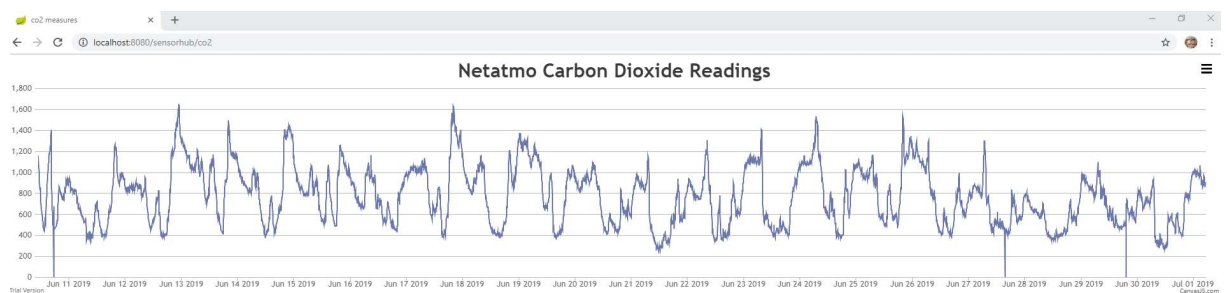
<http://localhost:8080/sensorhub/temp>



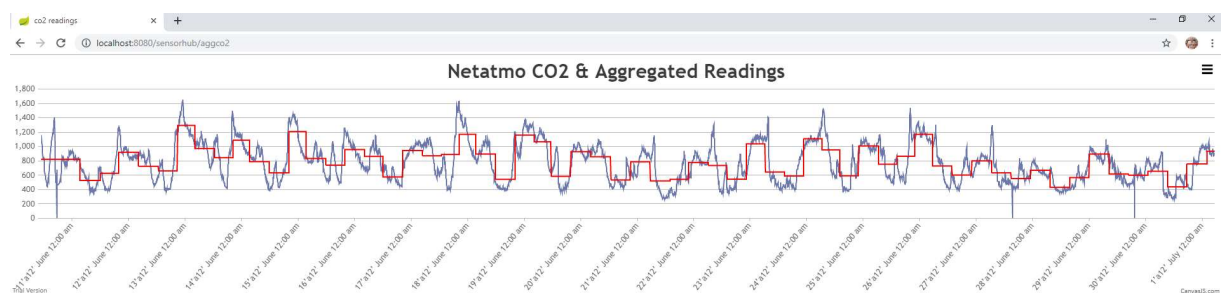
<http://localhost:8080/sensorhub/aggtemp>



<http://localhost:8080/sensorhub/co2>

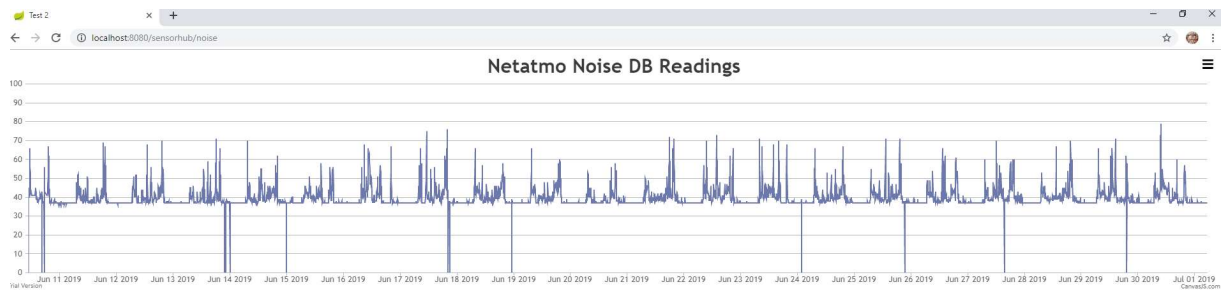


<http://localhost:8080/sensorhub/aggco2>

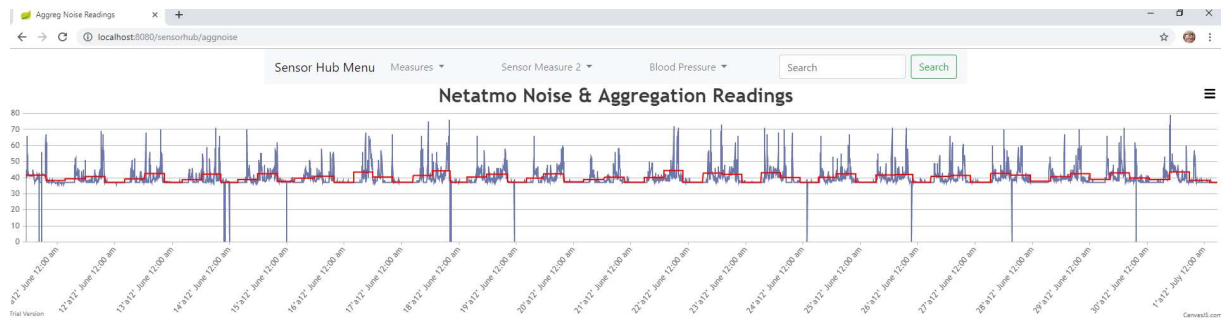




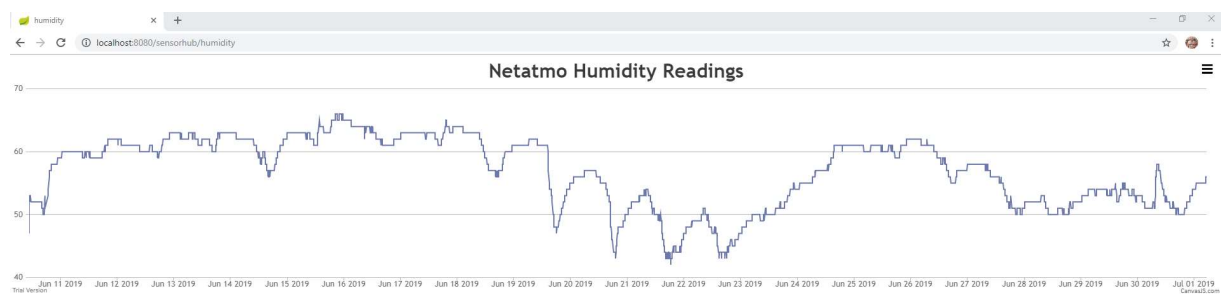
<http://localhost:8080/sensorhub/noise>



<http://localhost:8080/sensorhub/aggnoise>



<http://localhost:8080/sensorhub/humidity>



<http://localhost:8080/sensorhub/agghumid>



## Appendix G – Project Artefacts

### Project TOR

[REDACTED]

### Ethics Form

[REDACTED]

### Sensor Hub code

**sensorhubcloud.zip – MVP Product – Use Case 3 - 5**

[REDACTED]

**sensorsorhub\_d310719 – REST API – Use Case 2 - 4**

[REDACTED]

### Kanban Board

[REDACTED]

## Test Data

Project Artefacts Netatmo Get & Post Request Json string



responseAll\_d19072  
0\_v3.txt