Mitiku Wubetie

IoT Data Visualization

With Modern JavaScript Frameworks

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

20 October 2017

Metropolia

Author	Mitiku Wubetie
Title	IoT Data Visualization with Modern JavaScript Frameworks
Number of Pages	36 pages + 1 appendix
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialization	Software Engineering
Instructors	Kari Aaltonen, Principal Lecturer Sonja Holappa, Language Supervisor

The goal of the final year project was to build a web based visualization tool for data that is frequently sent from different sensors. To reach this goal, modern JavaScript frameworks and libraries used to build the application were explored. The project outcome is used as an in-house tool for a company called Polku Innovations Oy.

The project was managed by the scrum methodology. Data was analysed properly using data flow diagram and the initial requirements were inferred. Then, application was designed and implemented according to the initial requirements along with the customer feedback that was continuously added to the backlog.

The final outcome of the project allows displaying sensor data as tables and expressive charts. Besides, users can select a particular sensor location and view each of the parameter readings. Moreover, it is possible to access data later from the archive and view it as tables. Furthermore, alerts and notifications were included in the page which are used to inform the user if a sensor is disconnected or unexpected reading is found.

The primary goal of the project was successfully met. Further improvements and frequent updates are still expected. This can be done by following user feedback about the application. Besides, since the tool is built with a stack of JavaScript frameworks called MEAN, it is important to follow changes in each of the frameworks and act accordingly. This helps to benefit from the improvements on these technologies.

Keywords	Data Visualization, IoT, JavaScript, MEAN stack and Sensor

Metropolia

Contents

Appendices

Appendix 1. Receiving data from MQTT and sending to socket and database

List of Abbreviations

API	Application Programming Interface
BSON	A serialization format used to store documents and make remote procedure calls in MongoDB
CDN	Content Delivery Network
CSS	Cascading Style Sheets
DOM	Document Object Model
D3	Data-Driven Documents
HTML	Hypertext Markup Language
IoT	Internet of Things
JSON	JavaScript Object Notation
MEAN	MongoDB, Express, Angular and Node
REST	Representational State Transfer
SPA	Single-Page Application
SVG	Scalable Vector Graphics
URL	Uniform Resource Locater

1 Introduction

Internet of Things (IoT) is being approached by every industry as a means of facilitating and enhancing productivity. Increasingly more devices are becoming integrated into networks and have the capability of being accessed through the internet. This substantial system of possibilities comes with many opportunities worth exploring. Visualizing the data that is collected from different sensors in a meaningful way is one important aspect in IoT.

The goal of the project is to design and implement a web based system that is used to visualize the information collected from different sensors. The sensors are used to gather and emit various information from their respective location. Since the sensors send the data frequently, the application should handle the coming data systematically. Additionally, the data should be presented to users in the form of tables and expressive charts.

Polku Innovations is a company working on IoT and Artificial intelligence solutions. One of the services of the company is sensor technologies that can be used to gather a set of physical parameters of a location. Hence, the application will be added as a value to these technologies. Besides, the application is going to be used as an in-house web based application system for visualization of data sent by these sensors.

This thesis intends to find a solution to the business challenge, proposed by the company, using contemporary web technologies. Accordingly, this research strives to answer the following question:

> What is the advantage of using modern JavaScript frameworks in the visualization of IoT Data?

In order to get data from sensors, a light weight IoT protocol, called MQTT, is used. However, this thesis does not try to study how data is transferred among IoT devices and other systems. Rather, it primarily focuses on my particular role in the project, that is how modern JavaScript frameworks and libraries are used to visualize the data.

The thesis is divided into seven sections. The first section is the introduction part where the goal, background of the case company, business challenge, objective, and scope of the project is explained. The second section is composed of theoretical background related to the selected topic. The third section covers the approach and tools that are chosen for the development process. The fourth section is composed of the requirements analysis and design of the project. The fifth section tells implementation part of the development process. The six section shows the final result of the project. Lastly, the seventh section summarises the thesis project and points out future improvement directions.

2 Theoretical Background of Data in Internet of Things

This section covers on the theoretical background relevant to the thesis. Hence, it focuses on the overview of Internet of Things, how IoT generated data is handled and advancements in JavaScript. Then, some of possible future applications in the IoT is also mentioned.

2.1 Overview of Internet of Things

The idea of the Internet of Things was originally contemplated and made known by MIT, Auto-ID center and highly related to RFID and electronic product code (EPC). However, the term “Internet of Things” was later coined by Kevin Ashton in 1999. The literal meaning of IoT corresponds to all physical items communicating to one another. Machine-to-Machine (M2M) communications and person-t o-computer communication in turn extends to things. Hence, the Internet of Things (IoT) refers to a network of interconnected things integrated with sensors, RFID, smart devices, and any kind of devices capable of communicating with users or among themselves. [1, 2.]

The feasibility and practicality of IoT and related technologies are obviously influenced positively by the steady growth of internet usage over the past years. The rise of internet user count results in the addition of more applications and devices into the network. With this massive growth of the internet usage, the impact of IoT is inevitably significant.

In the future, IoT will use the internet as a stage to support and transmit different parameters. It may involve plenty of electronic measuring devices such as thermometers, pressure gauges, pollution detectors, cameras, microphones, glucose sensors, ECGs, or electroencephalographs embedded into different objects. These will boost and control cities and species at risk of extinction, the air, the ships, roads and fleets of trucks, our conversations, our bodies and even our dreams. [1, 4.] Figure 1 shows the rate connected devices in comparison with the human population of the world.

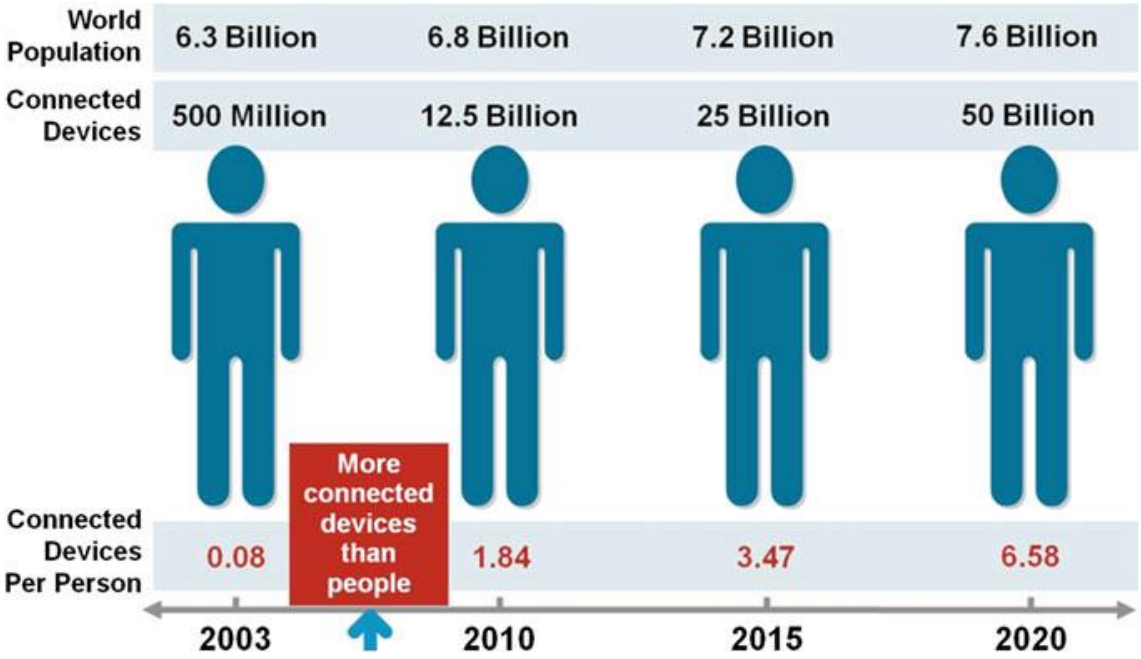

Figure 1. The comparison of connected devices with human population, copied from Subhas Mukhopadhyay (2014) [1].

As described in Figure 1, many more devices are connected to the internet than human population growth of the world in the years between 2003 and 2010. Moreover, the rate at which devices are introduced into the network is likely to increase in the years to come. This clearly visualizes the impact of the IoT in future.

2.2 Managing Data Generated by Internet of Things

Data Management is an important issue in IoT. Experimental results reveal that dealing with heterogeneous sensor data over an extended time interval produces an enormous amount of data [1, 229]. Forecasting the information size that the coming IoT world may

produce, minimizing the storage demand and decreasing the cost is important. Big Data is among the fundamental concepts to understand the challenges of IoT data management. [11, 84.]

Big Data deals with the handling and examining large data storage systems that are so incompatible with the common data analysis tools and database systems. Obviously, machines produce data much quicker than humans handle and their rate of generating data will increase exponentially according to Moore's Law. Subsequently processing large amounts of information effectively with a reasonable period of time needs special technologies. These technologies encompass the internet, enormous parallel processing (MPP) databases, distributed databases, data-mining grids, cloud-computing platforms, distributed file systems and scalable storage systems. [11, 84.]

Dealing with huge amount of and at the same time unconventional and unstructured databases is not easy. Thus, special attention is needed while considering storage, processing and interpretation. That is why companies such as Google, Yahoo! and Facebook tend to use distributed, cloud and open source systems instead of oracle tools. [11, 85.]

Currently, there are so many open source Big Data systems such as Hadoop, HBase, MongoDB, Cassandra, RabbitMQ, and plenty of others, which most of them fall under the term NoSQL. However, before those systems even exist, Google had first paved the way of such products as distributed file systems, the MapReduce computation framework, and distributed locking services. In addition, Dynamo was another prime innovation from Amazon which introduced a distributed key/value store system. [2, 2.]

2.3 Overview JavaScript Advancement

JavaScript was invented by Brendan Eich in May 1995 by the name Mocha intended to be an embedded program for the then popular browser Netscape Navigator 2.0. Then, the name was changed to Livescript and eventually to JavaScript in ten days. Two years after its invention, it was adopted to ECMA standard. ECMA-262 was the standard's official name. Though the languages are not alike conceptually as well as in design basis, JavaScript was called so reckoning on the then popularity of Java programming language. [5; 6.]

Ever since its standardization to ECMAScript, JavaScript has undergone so many improvements with a number of versions [5]. Table 1 shows the ECMAScript versions and improvements made by the JavaScript language.

Table 1. ECMAScript Editions, modified from W3Schools: JavaScript Versions [5;10].

Standard Name	**Year**	**Changes**
ECMAScript 1	1997	First Edition.
ECMAScript 2	1998	Editorial changes only.
ECMAScript 3	1999	Added Regular Expressions. Added try/catch.
ECMAScript 4		Was never released.
ECMAScript 5	2009	Added "strict mode". Added JSON support.
ECMAScript 5.1	2011	Editorial changes.
ECMAScript 6	2015	Added classes and modules.
ECMAScript 7	2016	Added exponential operator (**). Added Array.prototype.includes.
ECMAScript 2017 (edition 8)	2017	Added Object.values(), Object.entries(), String.prototype.padEnd(), String.prototype.padStart(), Object.getOwnPropertyDescriptors( )

As described in Table 1, each of the ECMAScript Editions, except ECMAScript 4 which was never used, came up with different improvements of the language. After 2015, each year new editions are released. The recent ECMAScript draft is ECMAScript 2017. Besides, ECMAScript 7 also called ECMAScript 2016, is supported poorly in most

browsers. Likewise, ECMAScript 6 is supported only partially in all modern browsers. However, all browsers fully support the ECMAScript 3. Moreover, all modern browsers except Internet Explorer 9 fully support ECMAScript 5. [5; 10.]

In the last two decades, the most common languages frameworks and libraries such as JavaScript emerged as trends in modern web development. The custom is in the frontend as well as in the backend web programming. Among the most common are Dojo and jQuery for front-end JavaScript, CodeIgniter for PHP or Ruby on Rails. These frameworks and libraries are intended to lessen the web development burden by reducing the barriers to entry.

Moreover, the application logic shifted from the server-side to the client-side succeeding the tendency towards frameworks and libraries. Angular, Backbone, Ember and ReactJS are among the most common. This approach is preferable as it reduces the load on your servers which in turn reduces cost. Virtually, the trend has an effect of outsourcing the required processing power of the application by forwarding it into the users' browsers. [12, 5.]

Owing to the constant improvements of the language, studies show that JavaScript has now been the world's most popular programming language [26]. After the introduction of AJAX in 2005, JavaScript was radically transformed so that it was able to be used to program significant programs such as Google Maps and Gmail. Besides, the release of Google Chrome browser in 2008 created a big competition between the browser vendors that JavaScript made performance improvements in significant rate. Now that JavaScript is a scripting language of the web from the client to the server side, programming web applications such as Twitter, Facebook and GitHub is possible. [7, 4.]

2.4 Future Applications of Internet of Things

Because of the prevalent connectivity and developments in ICT technology, more and more devices are capable of being added to the Internet. This results in a new trend of applications that are able to considerably boost people's way of life, learning, work, and entertainment. [23, 9.] Coming from its feasibility and current momentum, it is possible to predict potential applications of IoT in the future. Though IoT can be applied in every angle of people's day to day activities, the following can be examples of possible areas:

- Natural disaster forecasts
- Water scarcity control
- Industry
- Farming
- Health
- Smart homes
- Smart cities
- Smart security systems

Integrating sensors and following the corresponding simulations of their readings helps to predict possible natural phenomena. Besides, the IoT can be used in managing water shortage in different locations. This includes detecting water inadequacy as well as unintended upstream flow of water from the sewage system. Subsequently, all these predictions help to be proactive to the oncoming incidents and act promptly before destructions take effect. [22, 259.]

Applications of IoT can also be used in monitoring production of an industry sector by doing analysis of information gathered from the surrounding environment and various ICT equipment [21, 260]. An integration of different sensors can be used in data sensing and processing and then notifying the farmer by means of communication media of the infrastructure. Moreover, healthcare will be changed gradually into the digital realm by means of digital medical records and widely embracing biomedical sensor appliances widely instead of being limited to hospitals [23, 289].

In the near future, intelligent devices will be included in smart homes as distributed networks. These devices clearly scan the user and regulate the surrounding in an intelligent and tailored method [23, 87]. Likewise, these devices are capable of finding various applications of smart cities in such areas as air quality control, escape-route, electric efficiency, gardening etc. Yet, IoT can be applied to actualizing smart security systems. These may include tracing people and their assets, maintenance of equipment and infrastructure, and surveillance of spaces. [22, 259.]

3 Development Tools and Approach

This section covers the development tools and approach that are used in building the application. Besides, the methods of selecting the approach as well as the tools are mentioned in each of the subsections.

3.1 Development Tools

The development of the project carried out for Polku Innovations needs different software tools and platforms. Some of them are used for the sake of creating uniformity among the programmers in the company's team whereas some tools are chosen taking into consideration specific criteria related to the project implementation. Along with an explanation why they were chosen, the most important of these materials and software tools are mentioned in this section.

GitLab

GitLab is used in the project as an online git repository manager. GitLab.com is a free and open source platform with unlimited private as well as collaboration repositories. It is used so as to sync with company's developer team, even though there were alternative services like GitHub and BitBucket to be used for a similar purpose.

Visual Studio Code

Visual Studio Code is used as a script editor for the whole project development. It is a free, open source and light weight editor which can be installed on different operating systems like macOS, Windows and Linux platforms. Besides, it has a feature called IntelliSense that offers smart prediction based on imported modules, function definitions, and variable types. Still there are other editors, which are free and can be used for the same purpose in our project. But again, Visual Studio Code is used in this project for purpose of harmony among the development team.

Cloud 9

Cloud 9 is a development environment in the cloud, which is available in different paid plans as well as a free plan. The cloud 9 free plan is used since it provides a private

workspace and public workspaces, which is good enough for this project. Because it is convenient to work remotely anytime on the web, the cloud development environment is often preferred and used in our project development compared to the local development environment. Moreover, the workspace can be shared so that multiple programmers simultaneously work on the project.

3.2 Development Approach

Aiming for a quality web application software, the team has to choose the appropriate approach for the development cycle. Hence one of the agile methodologies, Scrum is selected considering some important criteria and facts. It is known that, the application has to be reliable and built with unclear user requirements, short time schedule and contemporary web technologies that are not familiar to the team. Figure 2 shows points to be considered while choosing the right methodology.

	Structured Methodologies		RAD Methodologies			Agile Methodologies	
Ability to Develop Systems	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP	SCRUM
With Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent	Excellent
With Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Good	Good
That Are Complex	Good	Good	Good	Poor	Excellent	Good	Good
That Are Reliable	Good	Good	Good	Poor	Excellent	Excellent	Excellent
With a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent	Excellent
With Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Excellent	Excellent

Figure 2. Points for selecting a development approach, copied from Alan Dennis [8].

Based on the criteria listed in Figure 2, Scrum suits best for a project with unclear user requirements, short delivery time and the need for a reliable system. Besides, it works well with unfamiliar technologies. Therefore, the Scrum methodology is used throughout the project development cycle.

The Scrum methodology focuses on the utilization of a set of software process paradigms. These process patterns are tested to be efficient for projects with short time schedules, liable to requirement changes, and business criticality. Hence, each of the process procedures denotes a set of development activities:

Backlog - a prioritized set of task requisites or attributes that add business value for the customer. Items can be appended to the backlog at any time that way new requirements are also introduced. The product manager examines the backlog and makes priority updates as needed.

Sprints - comprise of task items which are needed to attain a requirement described in the backlog that has be aligned with a time-frame (mostly for a month). During the sprint, no backlog item is introduced. Therefore, the sprint permits team members to perform the tasks in a short period of time, but steady situation.

Scrum meetings - are daily meetings held for a short time, usually 15 minutes, by the Scrum team. All the team members are required to ask and answer the following three key questions:

- What did you do since the last team meeting?
- What obstacles are you encountering?
- What do you plan to accomplish by the next team meeting?

A team leader, called a Scrum master, leads the meeting and checks the answers given by every person. The Scrum meeting aids the team to reveal possible issues as early as it can be. Also, these daily meetings direct to “knowledge socialization” and hence promote a self-organizing team format.

Demos - give the software status to the customers so that they can see and evaluate how the designed functionality is implemented properly. It is necessary to bear in mind that the demo may not include all designed functionality, but instead those functions which can be handed over within the time-frame that was made. [3, 83-84.]

While developing the project, Scrum meetings are conducted every morning. In each meeting, each of the team members explain what is done, what is to be done and the obstacles faced while performing the tasks. The Scrum principles are used to drive the project development tasks: requirements, analysis, design, evolution, and delivery. In the project, a backlog is used where requirements lists are added to the projects according to their priority. These lists are added any time when needed. Then, a collection of work units called sprint is done in every development task based on what is defined in the backlog. Figure 3 shows the Scrum process.

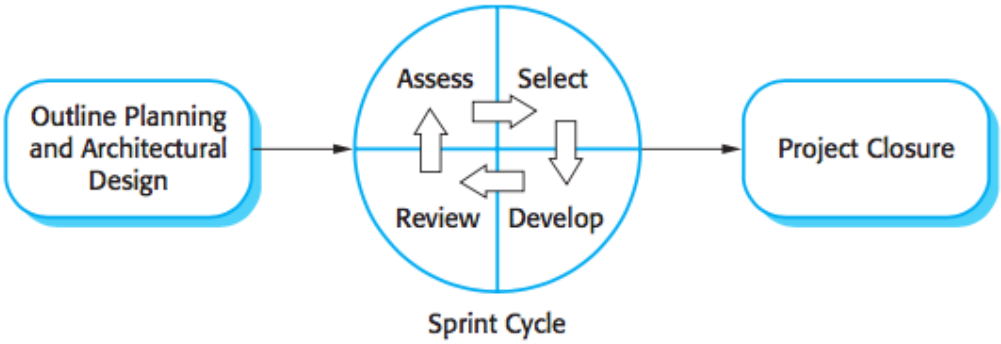

Figure 3. The Scrum process, reprinted from Sommerville [9, 73].

As shown in Figure 3, the Scrum approach comprises of three phases. In the first phase, Outline planning and architectural design, the general objectives of the project and software architecture design are defined. After that, the sprint cycles are followed where each cycle develops increment of the system. Then, the project closure phase is the final phase where the project is wrapped up and the required documentation of the project is created.

Since the iteration is performed across activities, the design and requirements are done at the same time. Hence, the method mainly focuses on high-quality and readable code. Besides, the most common manifesto of the agile software development approach is noted and properly implemented in the development process. The principles in focus are shown in figure 4.

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Figure 4. The common principles of agile approaches, copied from Sommerville [9, 60].

As explained in figure 4, customer involvement, incremental delivery, focusing on the team, accepting changes and simplicity are the most common agile methodology principles. The development team understands and tries to implement the principles accordingly in the development process. However, while implementing the customer involvement principle care is taken in such a way that it does not impose much pressure on the customer representative [9, 60].

4 Requirements and Design of Data Visualization

This section discusses the requirements and design stages of the application development process. Since the agile approach is used, these stages are considered to be the core activities of the whole project development process. The requirements and testing are also integrated into these stages.

4.1 Requirement analysis

The requirements analysis stage of the application development process analyses the main specifications set before and while developing the application. However, since the scrum development methodology is used, changes are often welcomed added anytime into the backlog to embrace new opinions of the customer and the team members.

Therefore, this section discusses a summary of the functional and non-functional requirements included in the project development process.

The functional perspective of the application system is illustrated in order to infer the main requirements of the system. Since data is the key component of the system, this illustration was done by drawing a data flow diagram (DFD). Figure 5 shows a simplified flow diagram of data emitted from a typical location.

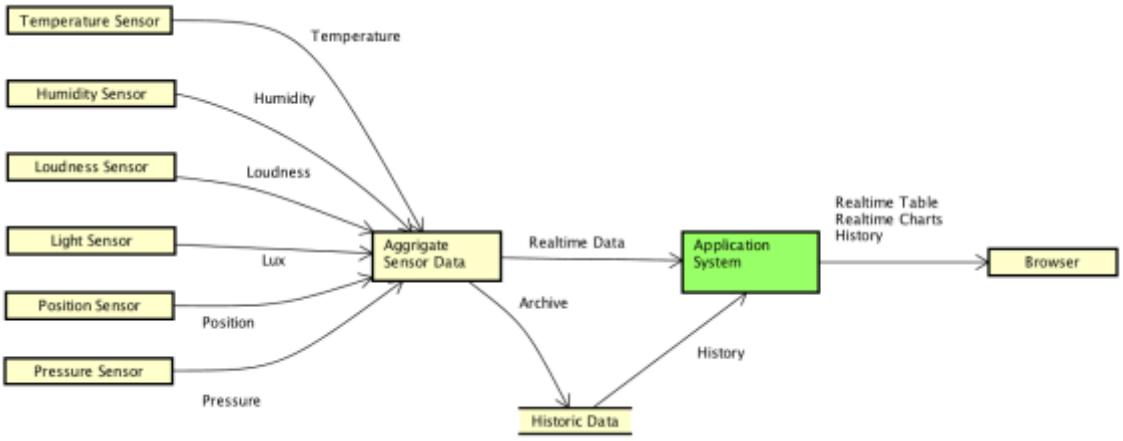

Figure 5. Data Flow Diagram of the system.

As illustrated Figure 5, readings from different sensors are collected as an aggregate data, which then will be visualized as real-time data and archived in the database as history. The parameters that are read by the sensors are temperature, humidity, loudness, lux, position and pressure. These real-time parameters are expressed as tables and different kinds of charts. Moreover, the data from the database is retrieved by the application system when needed.

After the data flow diagram is deducted and understood, the following functional requirements of the system are mainly included and approved at the first sprint:

- Real-time table - Since the application constantly gets a number of parameters from various sensors in different locations, it has to be designed in such a way that it can deliver and visualize all location's information as a real-time table to the end-user.

- Sensor view - In addition to the table view, readings from a selected location should to visualized as a real-time progressive chart which updates itself when the corresponding sensor emits new data.

- Chart - The archived data of the sensors has to be visualized as bar charts which are used to compare locations by each of the parameter types.

- History - The archived data should be retrieved from the database and viewed as a table while need.

- Notifications - When a sensor is added to the system, disconnected or connected back to the system, the corresponding notification should be displayed to the end user.

- Alerts - The system analyses the real-time information and incites the end user if measures need to be taken by displaying alerts.

The data flow diagram has also given an insight to plot the fundamental non-functional requirements of the application. These requirements are combinations of the data storage specification and quality requirements of the web application. Hence, the web application quality requirements tree is adapted to this project as part of the list of non-functional requirements. Figure 6 shows the list of non-functional requirements that the system is aimed to have.

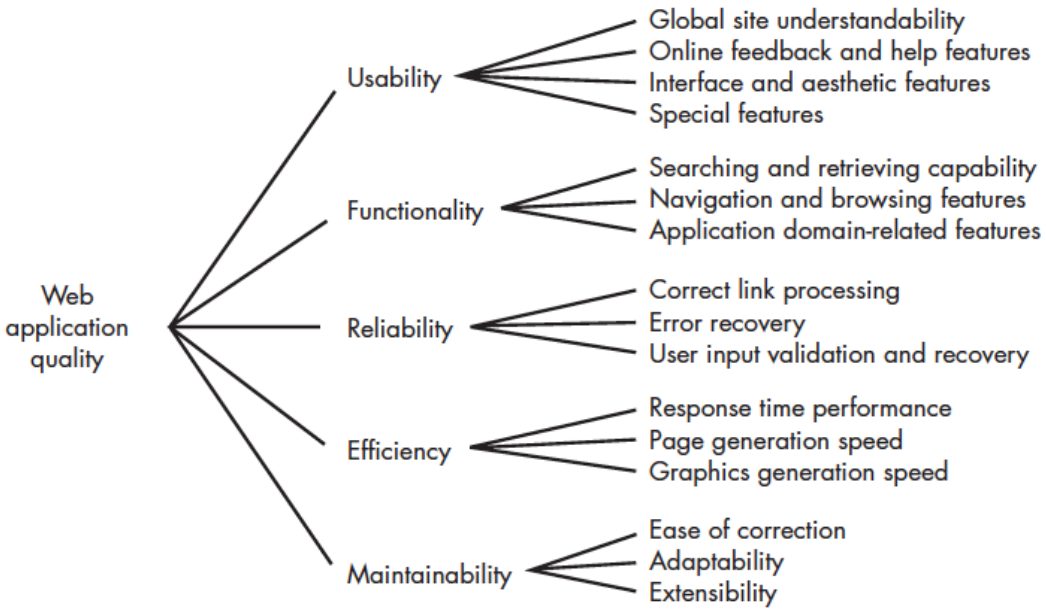

Figure 6. Quality requirements tree by Olsina and his colleagues, reprinted from R. Pressman [3, 375].

As shown in Figure 4, the quality requirements are described as a tree of technical attributes. These sets of attributes include usability, functionality, reliability, efficiency, and maintainability which are aimed to result in a quality web application. Each branch of the attributes in turn is composed of another list of attributes. In addition to the technical attributes mentioned, the requirements for the sensor data storage are also included as part of the non-functional requirements. These include horizontal scalability, model flexibility and performance.

4.2 Application architecture design

The application architecture comprises of a three-tier software architecture model. The three-tier model is expressed in a form of a stack of database, server-side and client-side of the application. With the intent of using a single language, the stack uses a collection of JavaScript frameworks and libraries all the way up from back to front. Subsequently, code reuse is one feature of the resulting architecture.

In order to map each of the layers of the three-tier model with the corresponding JavaScript framework, the requirements discussed in section 4.1 are referenced.

Consequently, the main technologies to be used are MongoDB as database system, Express and node.js in the server side while Angular and d3 charts are in the client side of the model. This stack of technologies is collectively known by an acronym MEAN stack from the names MongoDB, Express, Angular and Node.js. Features that made these technologies to be selected are described in sections from 4.3 to 4.5. Figure 5 shows a simplified idea of the application architecture.

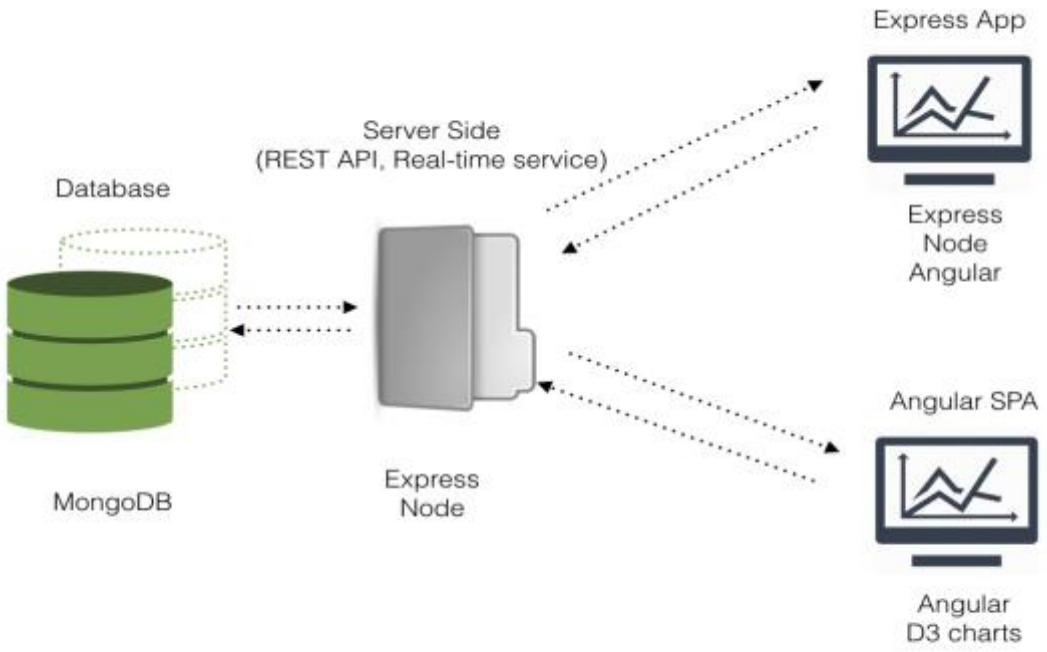

Figure 7. Application architecture, modified from Simon Holmes [12].

As shown in Figure 5, MongoDB is used as the database system for the application. The server side of the application mainly serves the REST API and the real-time data service to the client. The REST API serves as an interface between the database and the web application client. The components of the REST API are node.js and express which provides the JSON data to client. Besides, the real-time communication between the client and server is handled. On the other hand, angular and D3 work together in the client side of the system. [12.]

4.3 Preferred Database System

While choosing the database system to be used in the project, the system requirements discussed in section 4.1 are taken into consideration. Hence MongoDB is chosen as database system for the project since it is a high performance, high availability, and automatic scaling open-source document database. Being a document database system, its record comprises of a field-value paired data structure, called BSON, which resembles the objects in JSON [13].

Besides, since MongoDB is a NoSQL database system, it can effectively handle large amounts of input-output data. This is a result of a distributed storage and processing method using various data nodes. Furthermore, its scale-out storage architecture makes it to flexibly adapt changes against hardware upgrade and auto-sharding.[14, 486.]

4.4 Server-side

The server-side design of the system is mainly composed of two components. The first component is used as a medium communication between the client and the database. The second one is used to serve the real-time data to the client-side of the application. Hence, this section discusses the REST API and real-time services that form the server-side of the application. In addition to that, the node.js and express that are used to build these services are also discussed.

REpresentational State Transfer

REpresentational State Transfer, abbreviated as REST, is an architectural style instead of a strict protocol. Since REST operates with stateless communication, no current nor previous client information is stored. Hence, REST API is used to build a stateless application program interface to the MongoDB database, allowing other programs to work with the data. The REST API of this project is built with node.js and express. [12,15.]

Real-time Service

Real-time service of the application provides real time data that is emitted from different sensors. The service uses Socket.IO, a common module in the node community that

enables to build real-time web applications using a two-way event-driven communication channel between the server and browsers [11, 310.] Besides, along with the reliability and speed features, its cross-browser compatibility makes it to be the right tool in this project.

Node.js

Node.js is a JavaScript runtime developed on top of V8 which is a high-geared open source JavaScript engine by Google. It is efficient and lightweight, because of the fact that the language uses event-driven and non-blocking I/O models. That means it uses asynchronous I/O and event-loops related to actions as sensor readings or key inputs. As a result, it is best suited for data-intensive real-time applications. Besides, node package manager (npm) is included inside which provides important functionalities such as installing third party modules and single click installation of dependencies based on the package.json file information. [11, 4.] Therefore, it is a choice of the server side language of this project.

Express

Express is a lightweight and flexible web application framework for Node.js that hands over a robust feature for web based applications as well as mobile applications. It is also used to easily build an api using its countless http services and middleware. Moreover, its thin layer of essential web application layer is suited for high performance web applications without obstructing the node.js features. [18.]

4.5 Client-side

Client-side of the application is where the data visualization is done. It is built on the top of angular framework. Besides, d3.js library is added as a data visualization component to angular. The following paragraphs discusses the criteria and features that made these technologies to be chosen for the client-side of the system.

Angular

Angular is one of the latest JavaScript frameworks that helps to develop client side of web applications. Along with HTML, JavaScript language is used to build angular applications. Besides, the TypeScript language is also used which then compiles to JavaScript. Furthermore, the framework is comprised of plenty of built in libraries that some installed by default and others are optional.

The problem-solving method in angular integrates declarative templates, dependency injection, end to end tooling, and comprehensive best practices. Hence, writing the applications begin with building HTML templates with Angular syntaxes. After that, component classes are created to manage those templates. Then, application logic is added in services, and components and services are included in modules. Figure 6 shows the big picture in angular architecture. [16.]

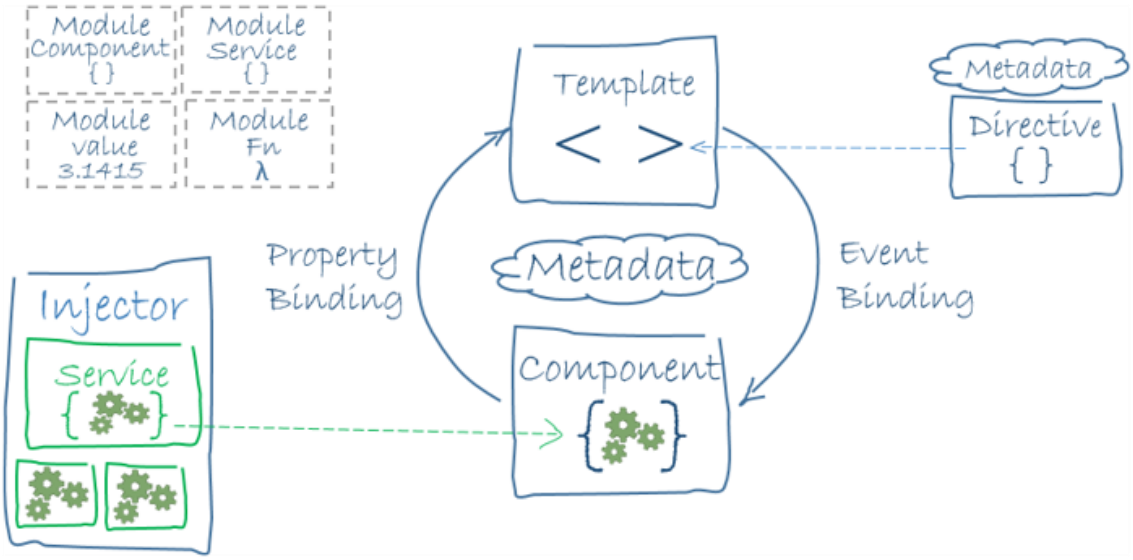

Figure 8. The angular architecture copied from angular documentation [16].

As depicted in Figure 8, components are the fundamental blocks of the angular architecture. Besides, a component property can be bound and viewed by the respective template. Similarly, an event in a template is bound to the corresponding component. In addition, a service can be created and injected to a component when needed.

D3.js

D3.js is one of the JavaScript libraries used in data visualization. As the name Data-Driven Documents implies, it is a tool that enables data driven manipulation of

documents. Moreover, because of its accent on modern web paradigms, d3 allows the developer to have complete competence of latest browsers without being dependent a definite framework. This also enables to integrate powerful components of visualization and to bind any data to a Document Object Model (DOM) to change the document based on data. [19.]

The primary reason why D3.js is selected as a data visualization library is its performance. Because it is dependent on SVG, the complete access to features of SVG results in an eased performance of browser animations and interactions. Most of the other data visualization libraries rely only on abstraction layer and the library's API. Besides, D3 is not limited to exploring the HTML DOM elements, but also the SVG elements and its CSS attributes. [4, 13.]

5 Implementation and Testing

This section discusses how the project requirements and design are put into use in the sprints. Besides, the proper setup of prerequisites is discussed here. Then, the most significant processes in the project implementation and usage of the technologies mentioned in section chapter 4 are discussed here.

5.1 Prerequisites

This section describes the prerequisites that should be done prior to the project development. As mentioned in section 3.1, the Cloud9 IDE is the main development environment. Therefore, creating the workspace as well as installing the necessary tools and services is important.

The MongoDB Community Edition is installed on a Raspberry Pi 2 where Ubuntu operating system is installed and serving as an in-house database server for the project. The installation process follows the four steps specified in the MongoDB official documentation installation manual. The installation steps are shown in Listing 1.

```
1  sudo apt-key adv --keyserver
   hkp://keyserver.ubuntu.com:80 --recv
   0C49F3730359A14518585931BC711F9BA15703C6
```

```
2  echo "deb [ arch=amd64,arm64 ]
   http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4
   multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-
   org-3.4.list
3  sudo apt-get update
4  sudo apt-get install -y mongodb-org
5  sudo service mongod start
```

Listing 1. Installing the MongoDB database on Ubuntu, copied from MongoDB Documentation [13].

As shown in Listing 1 line 1, the database installation starts by importing the public key used by the package management system. Then line 2 creates a list file for MongoDB. After that, the local package database is reloaded as in line 3. Finally, the last two lines show installation of the latest stable version of MongoDB and starting the database service respectively.

The mongo shell is a JavaScript command based tool for MongoDB which is used as an interactive interface. It is bundled in the MongoDB installation package. Therefore, it is available for handling operation like data update and query. The mongo shell is started and connected to the current instance of MongoDB which is running locally.

Another important step in the prerequisites is the preparing the Cloud9 IDE for project development. This step is simple and straightforward in most cases. To get into the workspace and start coding from the Cloud9 dashboard, the steps are:

- Click the "Create a new workspace" button
- Type the name and description of the workspace
- Choose private or public
- Clone from Git or Mercurial URL (optional)
- Choose a template

Among the steps mentioned above, it is important to choose a project name carefully. This is because renaming a workspace is not possible on the IDE. However, setting the project as private and public can be changed any time when needed. It is also possible to copy the project from a git repository or another workspace on the IDE. The IDE also has custom and ready-made templates for different kinds of programming languages.

After the workspace is created, the next step to setup the development environment by installing dependencies using the npm. Npm is package manager for the node and it is installed automatically when node.js is installed. Node.js modules or “packages” that are used to extend the functionalities of the application, is done using the npm [12]. Since Cloud9 IDE provides a workspace with Ubuntu operating system, the Node Version Manager (nvm) tool, which works on Linux and OS X environment, is also used to keep the node update-to-date [7.] Listing 1 shows the commands to perform initial setup of the Cloud9 workspace development environment.

```
$ nvm install stable
$ nvm alias default stable
$ npm install -g @angular/cli
$ cd && ng new smartOffice
$ ng serve -H 0.0.0.0
```

Listing 2. Development environment setup on Cloud9 IDE workspace, extracted from Angular CLI and npm documentation [15;17].

As illustrated in Listing 1, the command $ nvm install stable is used to download, compile, and install the latest stable release of node.js. Then, the second line of the listing makes the version to be the default node version. Then the command in the third line of the listing globally installs the angular cli, which is the scaffolding for the angular project development environment. The angular cli is a command line interface tool for angular projects that is used to create a project, add files, execute different development operations like bundling, deployment and testing [16].

5.2 Project Implementation

The implementation stage of the development process where the requirements and design are changed into code is explained here. These include implementations in each of the application architecture layers. The next few paragraphs discuss implementations of the database, the server side and client side of application.

Database implementation

The database implementation is carried out according to the requirements. Hence, the initial blueprint of the project database is done in only two steps. The first one is creating

the database and the second one is creating the collection. Hence, the database implementation requires relatively little time and knowledge while compared to the rest of the project implementations. The two commands used in the mongo shell are:

- use sensor-data - creates a database “sensor-data” and sets it as current database at the same time

- db.createCollection("history") - creates a collection called “history” in the current database (“sensor-data”)

Server-side implementation

The server-side implementation stage involves two main tasks. These are building the REST-API and the real-time sensor data services. To do this, the whole server side implementation process is decoupled into three different components for the sake of code readability and reusability.

The main component of the server side code, server.js, resides in the root file directory of the project which is created by the angular cli tool as in Listing 1. This static file server serves some of the built-in functionalities of the node.js and express that are used in the application. Besides, it manages the other two components that provide the REST-API and real-time services.

In addition to the main server file, another folder is created inside the root directory. This is for bundling the other two server side components, the REST-API and the real-time services. Listing 2 shows variable declarations for pulling the main built-in modules and then including the REST-API and real-time service files.

```
const express = require('express');
const path = require('path');
const http = require('http');
const bodyParser = require('body-parser');
const moment = require('moment');
const api = require('./server/routes/api');
const sensorManager =
require('./server/routes/sensorManager.js');
```

Listing 3. Including the main modules into the server, copied from Appendix 1.

As shown in Listing 2, the express server is included in the static server file. Besides the path module is declared in order to get the services related to file system paths. Likewise, the http module is required to add the server-client functionality of the application. For parsing the request bodies in a middleware, the body-parser is used. Along with the main modules, the other two components of the server, REST-API and real-time services, are also included in this file. Furthermore, moment is also used handle JavaScript date and time operations.

Client-side implementation

The implementation of the client side mainly embraces the user interface and meaningful presentation of the data provided by the server side. This is mainly achieved by utilizing the angular components and the d3 charts. Hence, the project makes use of the angular command line tool, called angular cli, to carry out different development tasks with ease.

The initial project files are already created by the Angular CLI tool while installing the prerequisites as shown in Listing 1. Thereafter, the rest of the application is meant to be scaffolded inside the root folder that is generated by the CLI. Hence, the main development process in the client side of the application is composed of the following stages:

- Creating the navigation and routing
- Creating the home, history and chart components
- Creating the services to communicate with the server-side data
- Injecting the services to the corresponding components
- Displaying data as tables and charts

Therefore, the next few paragraphs briefly show how these stages are applied in the project implementation process.

Creating the navigation and routing requires importing the Angular Router service that is used for presenting a particular component view for a given URL. This is done because the service is not part of the Angular core but its own library called @angular/router. Hence, a separate file called app.routing.ts is created inside the app folder. This file is where the router service is imported and the application routes are configured, and this configuration are then imported to the root module where assembles the application.

Listing 3 shows how the Angular Router library package is imported into the app.routing.ts.

```
    import { RouterModule, Routes } from '@angular/router';
```

Listing 4. Importing Angular Router service, copied from angular documentation [16].

In addition, the bootstrap library CDN is included in the main html page, index.html, which the CLI automatically created. The icons and fonts are also included in this page. Moreover, the main html page contains the <base> tag as the first child in the element <head> to inform the router how to construct navigation URLs.

Then, the necessary components, directives, pipes and services are added to the application. Listing 3 shows the commands used to generate a component and a service using the angular cli tool.

```
ng generate component history
ng generate service services/data
```

Listing 5. Generating a component and a service, adapted from Angular documentation [16].

The first line of the command in Listing 3 generates the history component. The other components are similarly created. Each of the components generated contains four different files with file extensions ts, html, css and specs.ts. These files contain the definitions of the corresponding component, HTML template, CSS stylesheet and unit test. The second command creates the data service inside the services folder. Again, the other service is generated similarly.

However, implementing the charts component additionally requires installing and integrating the d3 library. The d3 typings bundle is also added to the package. Then the d3 is imported into the charts component from the installed modules.

After the services are created, they are injected and used by any of components they are needed by. Because the @Injectable() decorator is automatically used in each service generated by the cli, the typescript is informed about the service metadata and

hence consistency and future-proofing is ensured. Therefore, the service is imported to a component and added as a private property to its constructor. [16.]

The last implementation stage is using the Angular templates to visualize data by binding controls to properties of the respective components. One of the Angular data binding styles, interpolation, is used to display a property defined in a component. It is used to mesh calculated strings into the content inside the HTML element tags and values of attributes of a template.

The whole project directory structure is shown in Figure 9.

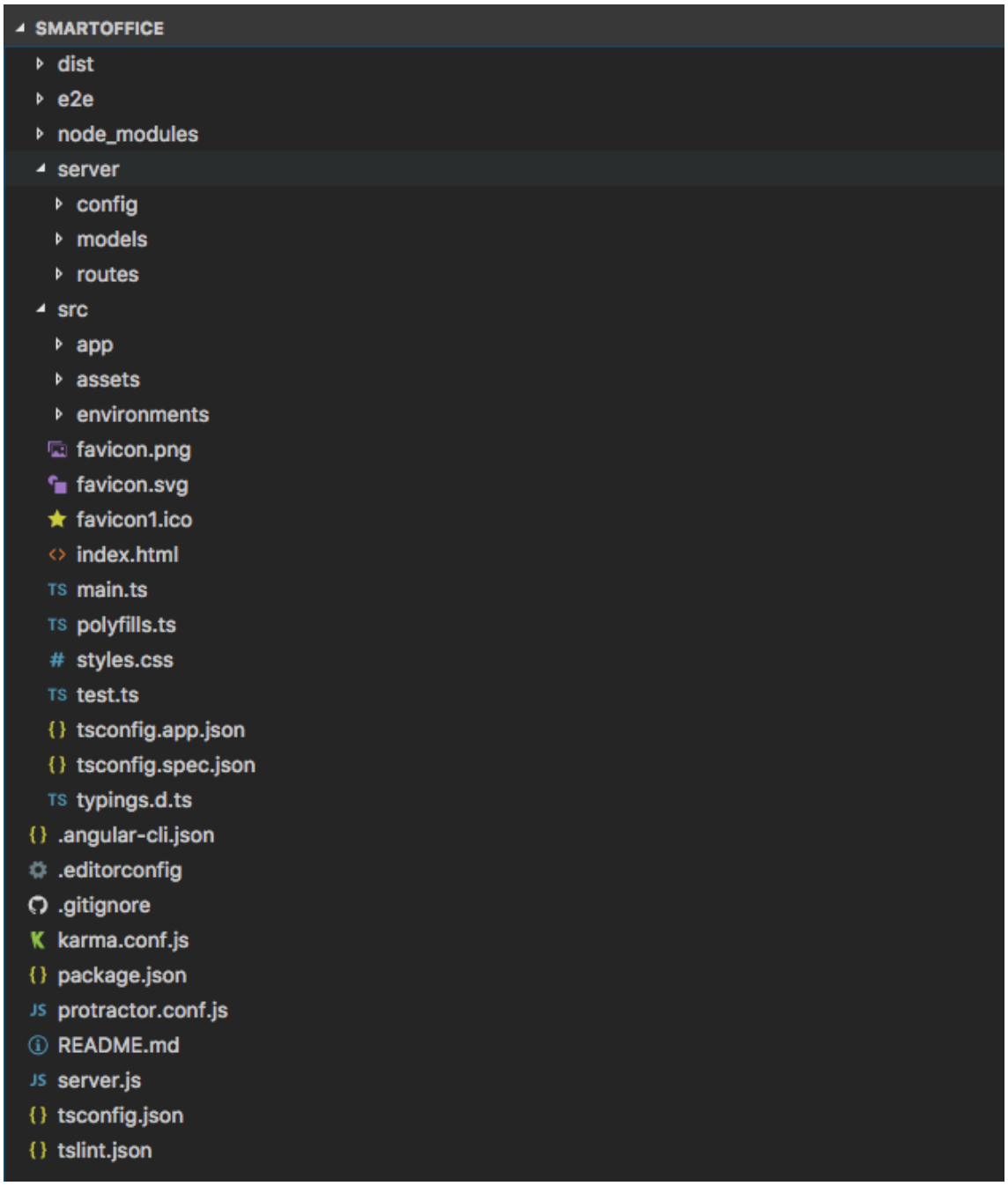

Figure 9. Project directory structure.

As shown in figure 9, the app resides in the src folder. All Angular components, services, templates, styles, images, and anything else required by the app should be included here. That means the rest of the files outside of this folder are supports for building the app.

5.3 Testing

Each prototype of the application is tested in every sprint with the intent of revealing design and implementation errors before handed over to the end user. Accordingly, unit tests and function tests are performed. To do so, different test tools and technologies that are integrated with the Angular development environment are utilized along with simple manual tests.

Components and services are unit tested using the Jasmine test framework that is integrated in the Angular cli project setup. Conventionally, the Jasmine written tests for each component and service contains file names with extensions specs.ts. Then, another integrated test runner, karma, runs these unit tests. The command “npm test” inside the root folder compiles the application and test code and starts karma. Figure 10 shows the test result from on browser and console.

Figure 10. Application test result on browser and console.

As shown in figure 10, the karma runs the tests written in Jasmine which resulted in 3 successful tests in the AppComponent. These test results are launched on the browser and also seen on the console. Besides, both processes watch the file in the test to detect and notify real time changes.

Functional tests are also carried out by directly viewing the application database, navigation, login, registration and visualization components. These tests are done in order to avoid errors, which the end user may experience while using application user interface. Then, the functional test results are compared with the functional requirements of the project mentioned in section 4.1.

6 Project Outcome

This chapter summarises the results of the thesis project and compares them to requirements. In addition, the problems faced in the design and implementation process are reviewed in the discussion section. Then, solutions sought to the respective problems are also discussed.

6.1 Summary of Project Outcome

After a proper design and implementation of the project, a user friendly and responsive web application system was eventually produced. The application login and navigation component are now able to be accessed via browser. Besides, the real-time data as well as the data from the REST API is visualized in the form of tables and charts. Moreover, readings from each of the sensors in different locations is able to be viewed separately.

The application has a landing page where the user sees after feeding the URL into the browser. The landing page briefly shows what is inside the system. Besides, it has the login and signup links where the users are lead to the external authentication system. Figure 11 shows screenshots of the application landing page with different view ports.

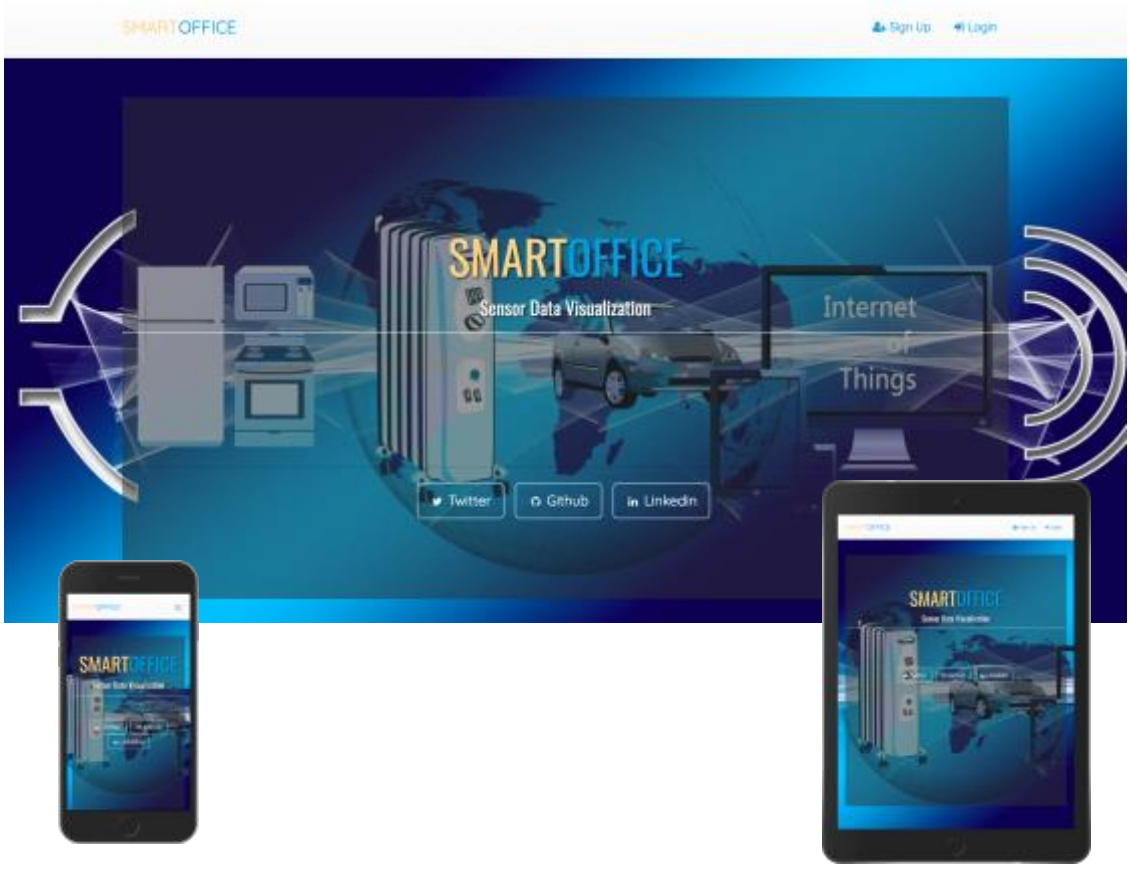

Figure 11. Screenshots of the landing page on three different devices.

As shown in Figure 11, the landing page of the application is responsive on different devices and viewports. Besides, the picture shows that a brief description of the contents of the home page and the rest of the pages. Moreover, the links to the login and signup are shown on the top-right corner of the page.

The homepage of the system displays real-time data as a table of different sensors after the user is authenticated. The table shows detailed information of each sensor and is updated while a new data arrives. Besides, the maximum and minimum values of temperature and humidity are displayed dynamically. Figure 12 shows a screenshot of the homepage of the system.

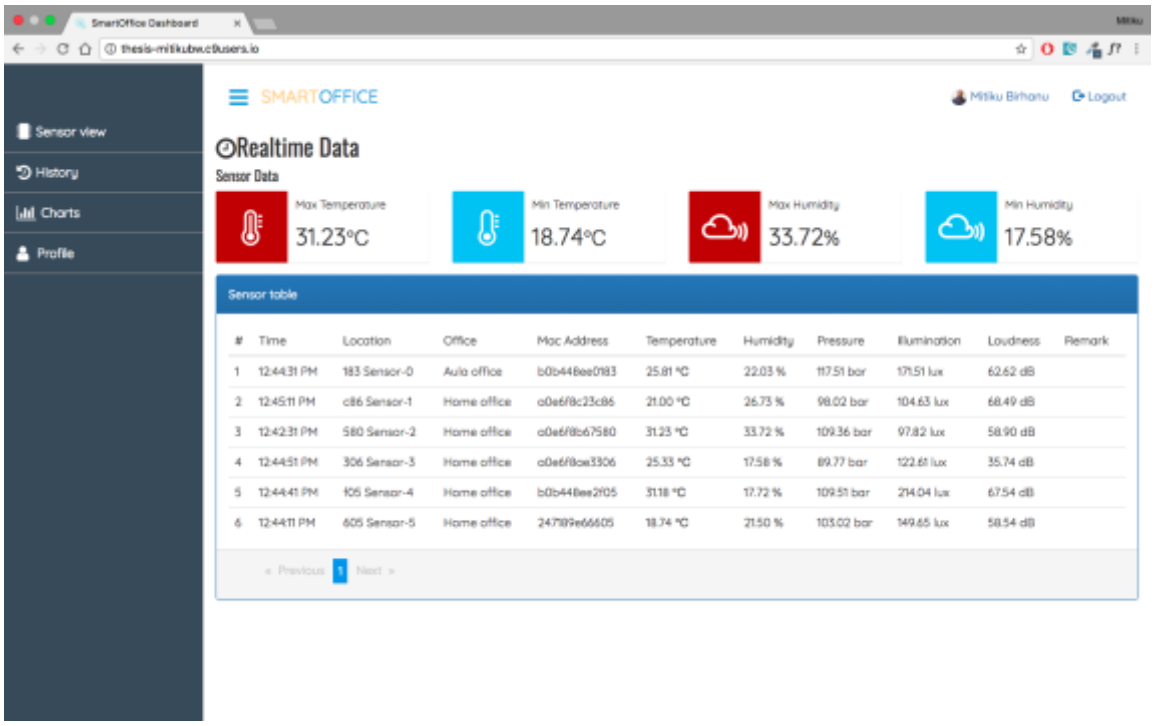

Figure 12.Screenshot of real-time page of the application.

As pictured in Figure 12, the homepage of the application displays a real-time table of sensor information. Besides temperature and humidity readings from different locations are compared and the corresponding maximum and minimum values are displayed. The page navigation is also available on the left side for easy propagation through the pages. Furthermore, the name of authenticated user and the logout link is displayed on each page's top-right corner.

The sensor view is one of the important subpages which contains a real-time visualization of the sensors by location. The visualization is presented as a form of radial progress charts that shows readings of different parameters of the selected location. Besides, the view shows the minimum, recommended and maximum values of each of the parameters. Furthermore, a multi-line chart of temperature and humidity readings from different locations is used in order to show values at different timestamps.

The history page of the application also is a visualization of the REST API which serves archived data. It contains a table of sensors with the corresponding readings of the current month. Each of the location readings contain temperature, humidity, pressure,

illumination, loudness and lux at different timestamps. Moreover, the paging functionality of the tables is properly implemented for easy readings.

Moreover, the application contains the charts page which visualizes the REST API in the form of different charts. It presents the charts by different parameters of sensor readings. These parameters are the temperature, humidity, pressure, illumination, loudness and lux. The type of chart used is the bar chart in order to compare and visualize the average values of readings of each of the locations.

6.2 Development challenges

The whole application development process progressed steadily and resulted in presumed outcomes. However, a few problems came about because of temporary shift in priority of the project and changes in development environment. Hence this section describes these issues and how they are resolved.

Changes in the priority of the project was changed during the project development process. Therefore, this project had to be suspended for some time and there was a need to shift towards another project which was more urgent. Consequently, the project did not complete according to the predefined time frame which was expected to be two months.

Another challenge happened due to the changes in the development environment. Angular transitioned from angular 2 to angular 4 in middle of the project development process. Consequently, it was decided to update the development environment to the later version in order to make use of its new features. Though the new version is backward compatible, some of the helper libraries were needed to be reinstalled in order to make the application keep working.

The application login system is integrated with Auth0 which is a third-party authentication service. Since the Auth0 is implemented with a helper library angular2-jwt, upgrading the development environment to angular 4 resulted in the login system failure. That was because the angular2-jwt in turn needs a peer of, as mentioned in the console message, @angular/core@^2.0.0 and @angular/http@^2.0.0. But, the mentioned dependencies were already replaced by the corresponding versions of the new angular.

6.3 Solutions Sought

The problems were thoroughly scrutinised and solved accordingly. Most of the problems were so minor that they were easily identified and fixed by following the console error messages. Furthermore, as the project priority changes were made by company, the project plan was shifted dully and resumed afterwards. This was easy because the project deployment approach was agile.

Whereas one of the problems, the login system failure, needed some changes in the project library modules to be fixed. Hence, in order to solve the login system issue, it was required to refer to the Auth0 documentation again and act accordingly. Following the documentation, two measures were taken. The first one was uninstalling the old angular2-jwt helper library module, then the second one was installing the auth0.js library. Finally, by making the respective changes in the login component, the problem was able to be fixed.

7 Conclusion

The goal of the project was to build a web based application that is used to visualize data collected from different sensors which are located in different places. Since data is the main component of the application, a proper data analysis was done beforehand and the project was managed by the scrum methodology. Then, it was designed and implemented according to the initial requirements along with the customer feedback that was continuously added to the backlog.

The application architecture was designed and implemented with respect to the requirement analysis. Each of the JavaScript frameworks were selected considering the architecture model and the features satisfying requirements lists. Using these modern JavaScript frameworks were effective in implementing the REST API, real-time services and visualizing the sensor data. Furthermore, the frameworks were useful in code reusability as well as fast development. As a result, the outcome of the project was in line with what has been planned initially.

Eventually, it can be concluded that the primary objective of the project is achieved well by adding values to the sensor technologies. The final product gave useful interpretation to the sensor data that the end users are able to understand the events more easily. Besides, alerts and notifications were effective in spotting failed or disconnected sensors. Furthermore, these features were useful to make relevant decisions to unexpected readings.

Future improvements of the application are also needed and expected at least in two ways. One of them will be converting the browser application into native mobile app. As mentioned in the angular documentation, this can be done with strategies from Ionic Framework, NativeScript, and React Native. The other one will be updates and improvements of the existing database structure and scaling up as needed. This is achieved by following customer feedback frequently.

References

1 Subhas Mukhopadhya. Internet of Things: Challenges and Opportunities. Cham, Switzerland: Springer International Publishing; 2014.

2 Nathan Marz, James Warren. Big Data: Principles and best practices of scalable realtime data systems. April 2015.

3 Roger S. Pressman. Software engineering: a practitioner's approach. Boston: McGraw Hill Higher Education; 2010.

4 Christoph Körner. Data Visualization with D3 and AngularJS: Build dynamic and interactive visualizations from real-world data with D3 on AngularJS. Birmingham, UK; April 2015.

5 W3Schools Online Web Tutorials: JavaScript Tutorial [online]. URL: https://www.w3schools.com/js/default.asp. Accessed 17 April 2017.

6 Marijn Haverbeke. Eloquent JavaScript: A Modern Introduction to Programming [online]. URL: http://proquestcombo.safaribooksonline.com.ezproxy.metropolia.fi/book/programming/javascript/9781457189821/eloquent-javascript-a-modern-introduction-to-programming/index_html?uicode=evtek. Accessed 12 November 2017.

7 Mike Cantelon, Mark Harter, HOLOWAYCHUK T.J., NATHAN RAJLICH. Node.js in Action. Shelter Island, NY: Manning Publications Co.; 2014.

8 Alan Dennis, Barbara Haley Wixom, David Tegarden. Systems Analysis and Design with UML 4th Edition [online]. URL: http://proquestcombo.safaribooksonline.com.ezproxy.metropolia.fi/book/software-engineering-and-development/uml/9781118037423/chapter-1-introduction-to-systems-analysis-and-design/navpoint-15?uicode=evtek. Accessed 12 November 2017.

9 Ian Sommerville. Software Engineering 9th Edition. Pearson Education Inc. 2011.

10 MDN Web Docs [online]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Accessed 25 October 2017.

11 Vermesan Ovidiu, Friess Peter, editors. Internet of Things: Converging Technologies for Smart Environments [Internet]. Aalborg: River Publishers; 2012. [cited 2017 October 23]. Available from: ProQuest Ebook Central.

12 Simon Holmes. Getting MEAN with Mongo, Express, Angular and Node. Shelter Island, NY: Manning Publications; 2016.

13 MongoDB Documentation [online]. URL: https://docs.mongodb.com/manual/. Accessed 16 June 2017.

14 Yong-Shin Kang, Il-Ha Park, Jongtae Rhee, and Yong-Han Lee. MongoDB-Based Repository Design for IoT-Generated RFID/Sensor Big Data. IEEE SENSORS JOURNAL, VOL. 16, NO. 2, JANUARY 15, 2016.

15 Angular CLI: A command line interface for Angular [online]. URL: https://cli.angular.io/. Accessed 2 July 2017.

16 Angular Docs [online]. URL: https://angular.io/. Accessed 2 July 2017.

17 npm: the package manager for JavaScript [Online]. URL: https://www.npmjs.com/. Accessed 19 October 2017.

18 Express - Node.js web application framework [online]. URL: http://expressjs.com/. Accessed 22 October 2017.

19 D3.js - Data-Driven Documents [online]. URL: https://d3js.org. Accessed 19 October 2017.

20 Andrew Poulter, Steven Johnston, Simon Cox. "Using the MEAN stack to implement a RESTful service for an Internet of Things application," *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Milan, 2015, pp. 280-285.

21 Fazio M., Celesti A., Puliafito A., Villari M. (2014) An Integrated System for Advanced Multi-risk Management Based on Cloud for IoT. In: Gaglio S., Lo Re G. (eds) Advances onto the Internet of Things. Advances in Intelligent Systems and Computing, vol 260. Springer, Cham.

22 R. Khan, S. U. Khan, R. Zaheer and S. Khan, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," *2012 10th International Conference on Frontiers of Information Technology*, Islamabad, 2012, pp. 257-260.

23 Michael J. McGrath and Cliodhna Ní Scanaill. Sensor Technologies: HealthcareWellness, and Environmental Applications. New York, NY: Apress Media; 2014.

Appendix 1: Receiving data through MQTT and sending to socket and database

server.js - Receives data through MQTT and pushes it to Socket and the database

```
// Get dependencies
const mqtt = require('mqtt');
const express = require('express');
const path = require('path');
const http = require('http');
const bodyParser = require('body-parser');
const passport = require('passport');
const app = express();
const moment = require('moment');
const config = require('../config/settings');


/* Settings.js configurations */
const webserver_port = config.get('settings.webserver_port');
const broker_port = config.get('settings.broker_port');
const broker_ip = config.get('settings.broker_ip');
const mongodb_host = config.get('settings.mongodb_host');
const topics = config.get('settings.topics');

// Get our API routes
const api = require('./server/routes/api');
const sensorManager =
require('./server/routes/sensorManager.js');

// Parsers for POST data
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Passport Middleware
app.use(passport.initialize());
app.use(passport.session());

// Point static path to dist
app.use(express.static(path.join(__dirname, 'dist')));

// Set our api routes
app.use('/api', api);

// Point static path to dist
app.use(express.static(path.join(__dirname, 'dist')));

// Set our api routes
app.use('/api', api);

// Catch all other routes and return the index file
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'dist/index.html'));
```

```
});




/**
 * Get port from environment and store in Express.
 */
const port = process.env.PORT || webserver_port;

app.set('port', port);

/**
 * Create HTTP server.
 */
const server = http.createServer(app);

sensorManager.server = server;

//Socket io connection
const io = require('socket.io').listen(server);

//mqtt broker connection
const mqtt_client = mqtt.connect('mqtt://' + broker_ip + ":" +
broker_port);

mqtt_client.on('connect', function () {
   // Subscribe to all topics given in config file
   for(var i=0; i < topics.length;i++)  {
      console.log('debug', 'Subscribed to topic: ' + topics[i]);
      mqtt_client.subscribe(topics[i]);
    }
});
```

```
/**
 *
 * This Function receives MQTT messages and processes them.
 * Messages are in JSON format. Messages have been enriched at
 * MQTT broker. This function routes messages to DB or Socket
 *
 * */
mqtt_client.on('message', function (topic, message) {

    /* Return If topic is not defined in configuration file. */
    if((topics.indexOf(topic) <= -1)) {
         console.log('error', 'topic: ' + topic + ' not found.
         Skipping processing.');
         return;
    }

    sensorManager.update_status(message, io);

    io.sockets.emit('sensor_data', message.toString());

   }
});

/**
 * Listen on provided port, on all network interfaces.
 */
server.listen(port, () => console.log(`API running on
localhost:${port}`);
```