

Bachelor's thesis

Information and Communications Technology

2018

Ilari Léman

CREATING A BUILDING HEALTH IOT APPLICATION

BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2018 | 21, 4

Ilari Léman

CREATING A BUILDING HEALTH IOT APPLICATION

The purpose of this project was to create an IoT application used for monitoring the health of a small hempcrete building. The application was created using ThingWorx and receives data wirelessly from the building via sensors measuring the temperature, humidity, and pressure. This thesis attempts to demonstrate the process used to connect sensors and display the data from them using ThingWorx.

A working application displaying the sensor data and meeting all the requirements was created successfully. Sensor data was displayed in a table, graph and as plain values within the application. A floor plan map was also created in the application for displaying the locations of the sensors within the building. The information found in this thesis may be used as a guide for others wanting to develop applications using ThingWorx.

KEYWORDS:

IoT, Internet of Things, ThingWorx

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS	5
1 INTRODUCTION	6
2 TECHNICAL ASPECTS	7
2.1 Hardware	7
2.2 REST API	8
2.3 JSON	9
3 THINGWORX	11
3.1 Resources & Documentation	11
3.2 ThingWorx Composer	11
4 CREATING THE APPLICATION	13
4.1 Requirements	13
4.2 Creating Mockup Mashup	13
4.3 Connecting Cloud Data to ThingWorx	16
4.4 Connecting Data to Mashup	16
4.5 Styling Mashup	18
4.6 Results	18
5 CONCLUSION	20
REFERENCES	21

APPENDICES

Appendix 1. Script used to get JSON file from Humia cloud for sensor 2044D9

Appendix 2. Script used to get JSON file from Humia cloud for sensor 2044D1

Appendix 3. Script used to parse JSON file and create infotable for sensor 2044D9

Appendix 4. Script used to parse JSON file and create infotable for sensor 2044D1

FIGURES

Figure 1. Image of Humia sensor model A.	7
Figure 2. Image of Humia sensor model B.	8
Figure 3. Example JSON response from Humia's cloud.	9
Figure 4. Homepage of ThingWorx composer.	12
Figure 5. Create a new mashup in ThingWorx.	14
Figure 6. Mashup editor in ThingWorx.	14
Figure 7. Mockup mashup for the application.	15
Figure 8. Button for adding service data, top right of mashup builder.	17
Figure 9. Adding data to a widget in mashup editor.	17
Figure 10. Widgets properties tab in mashup editor.	18
Figure 11. Final layout of the application created.	19

LIST OF ABBREVIATIONS (OR) SYMBOLS

Abbreviation	Explanation of abbreviation (Source)
IoT	Internet of Things
REST	Representational state transfer
API	Application programming interface
HTTP	Hypertext transfer protocol
JSON	JavaScript object notation

1 INTRODUCTION

In recent years, there has been rapid growth in embedded systems and with it, an increased need to connect embedded systems to the internet. Embedded systems can already be found in many everyday devices, such as, home appliances, cars, airplanes and toys. The current growth in the embedded systems field is in large due to the increased interest of connecting these devices to the internet, known as the Internet of Things (IoT). By connecting these devices to the internet, end users can control and monitor these devices remotely.

Although, the introduction of IoT brings with it functionality which may be useful for many applications, much of the data sent by these devices may be difficult to interpret, analyze and be made use of by laymen. Because of this, a system must be created to manage and display the data so that it can be used. These systems are often applications which analyze data, then display the information in a user-friendly manner. Graphs and tables are examples of common methods to display information to users.

Currently there are many companies offering software which makes it easy to connect device data to a system and create IoT applications to then manage the data. Software of this kind is often known as an IoT platform. The software products available often offer support for many different platforms and technologies used in IoT development, therefore making them a solid option for IoT application development.

The aim of this work is to demonstrate the process of creating an IoT application using ThingWorx, a software published by the PTC company. The application being created will be used to manage the data from sensors in a way that the health of buildings can be monitored for harmful issues such as mold. The problems wanting to be solved by creating this application are displaying the time series nature of the data and displaying the spatial positioning of the sensors.

Chapter 1 of this paper introduces the work by providing background information about the work and presenting the aim of the work. The following chapter displays the technology used in the creation of the application and discusses the technologies in detail. Chapter 3 introduces the ThingWorx software and Chapter 4 demonstrates the process of creating the application using ThingWorx. Chapter 5 is the closing chapter where the project will be evaluated and the authors final thoughts will be given.

2 TECHNICAL ASPECTS

There are many tools and technologies used in the creation of this work which may not be familiar to laymen. In this work both hardware and software tools have been used. In this chapter of the paper, these tools and technologies will be discussed, as well as, their importance to the work.

2.1 Hardware

All the hardware used for this project are commercial devices received from a company located in Finland known as Humia. The Humia devices use many different sensors to measure temperature, humidity and pressure. These sensor readings are then sent to a cloud using a radio transmitter which is built into every device. At the time of writing this paper, the Humia company has two different device models, a model A and a model B.

The model A devices have 3 temperature sensors and 3 humidity sensors which are in different locations throughout the device. These devices are used by drilling a hole in the wall, floor or ceiling of a building and then placing the sensor into the hole. The A model come in a variety of different sizes to fit different thicknesses of walls and floors. Figure 1 is an example of a model A sensor which was used in this project.



Figure 1. Image of Humia sensor model A.

The model B devices have 3 sensors for measuring temperature, humidity and pressure. The B model can be used as is, and placed anywhere within the building. Figure 2 is the model B sensor used in this project.



Figure 2. Image of Humia sensor model B.

The only requirement for the hardware in this project is that the hardware has the capability for IoT connectivity and its data can be accessed. All Humia sensors come equipped with a radio transmitter sending data to the cloud about every hour and are connected to the company's cloud out of the box. Therefore, the IoT connection already exists and the data from the device is ready to be used in the application. For this work, one of each device model was available and used in the creation of the application.

2.2 REST API

REST API is an API based on the REST architecture which enables systems to operate with one another over the internet using the HTTP protocols. API stands for application programming interface and is a set of rules that allow programs to talk to each other. REST API often works by sending a request using a specified URL and receiving a response with the data for the sent request. The response can be in a number of different formats including HTML, XML or JSON.

The data from the Humia cloud used in this work is accessed using REST API. ThingWorx has direct support for REST API connection with built in code snippets making it simple to connect the Humia devices to the ThingWorx platform. In the case of

Humia's REST API, the response elicited by the request is in a JSON format. The response sent by Humia can be altered by changing parameters in the URL of the request. For example, you can get the JSON output for the latest 10 data points from a sensor by adding "?count=10" to the end of the device ID.

2.3 JSON

JSON is a simple and readable file format which uses text for exchanging data in attribute-value pairs. Because JSON is written in text, it can be easily read and written by humans. JSON is commonly used in communication between browsers and a server. While JSON is based on JavaScript it can also be used by almost any other programming language as a data format. Figure 3 below is an example of a response sent from Humia's REST API in JSON format.

```

{
  "statusCode": 200,
  "message": "Device Data Successfully Fetched",
  "data": [
    {
      "device": "2044D9",
      "model": "A160",
      "timestamp": "17.05.2018 07:23:31",
      "unixTime": "1526541811",
      "alarm": false,
      "temperature": {
        "sensor_1": 25.2,
        "sensor_2": 25.2,
        "sensor_3": 25.3
      },
      "humidity": {
        "sensor_1": 47.4,
        "sensor_2": 47.4,
        "sensor_3": 47.6
      },
      "mould_risk": {
        "sensor_1": {
          "risk": "0"
        },
        "sensor_2": {
          "risk": "0"
        },
        "sensor_3": {
          "risk": "0"
        }
      }
    }
  ]
}

```

Figure 3. Example JSON response from Humia's cloud.

As mentioned previously, the Humia REST API uses JSON as its output format. ThingWorx can parse and manage JSON without any extra code if it is in the correct format, but may require some code if not. In the case of this work, extra code needed to be added to parse the JSON, but this will be discussed in the “creating application” chapter.

3 THINGWORX

For the software aspect of this application, a software known as ThingWorx was used. ThingWorx is a software which is used to run and rapidly develop IoT applications for many different technologies and platforms. Currently there are several different versions of the software released with the most recent being ThingWorx 8. This chapter of the paper will introduce the features of the ThingWorx platform to the reader.

3.1 Resources & Documentation

Once an account has been created, the user will have access to the ThingWorx developer portal and all of the documentation for ThingWorx. ThingWorx's developer portal and documentation are very useful resources for learning about the basics of the ThingWorx platform as well as more complex tasks. In the developer portal the user has access to resources such as, walk-through guides, webinars, documentation and much more. Along with the developer portal, the ThingWorx community forum is a great resource for finding information and solutions to issues related to ThingWorx. The author used the developer portal and community forum to learn the software, as well as, problem solve issues.

3.2 ThingWorx Composer

The ThingWorx composer is the main screen of the software where the project and all aspects related to the project is managed. From the composer's home page, the user has access to all the features of ThingWorx. The features are divided into several categories which can be seen on the left side of the composer home page. The composer home page can be seen in Figure 4.

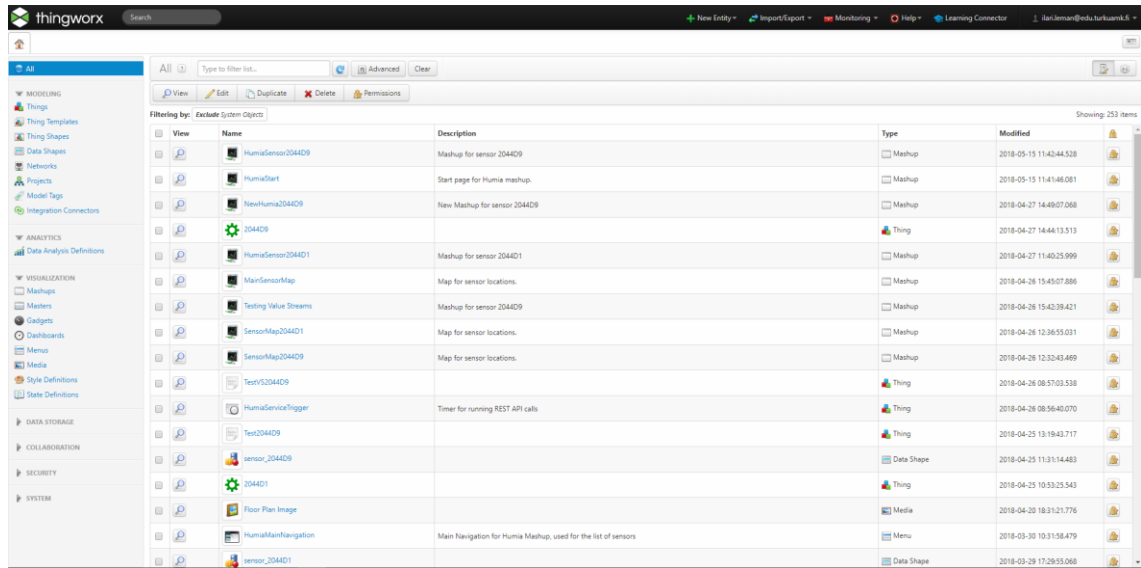


Figure 4. Homage of ThingWorx composer.

The modeling and visualization categories were used the most during the creation of the application. Creating objects and giving them functionality is done within the modeling page. For example, in this work, each sensor was created here and assigned functions which they perform when required. Therefore, modeling is important for creating the structure of the application and its entities.

On the other hand, visualization focuses on the look of the application and displaying information through mashups. A mashup is needed to display information, mashups are a ThingWorx web page. ThingWorx uses a mashup builder which can be used to design the look and functionality of a mashup. The mashup builder has many built-in widgets which give the developer multiple different display methods. A large portion of the mashup builder's tools are drag and drop making it quick and easy to create these mashups.

4 CREATING THE APPLICATION

This chapter demonstrates the process of creating the ThingWorx application for this work. The ThingWorx software was already running on a server provided by the commissioner of the project. The version of ThingWorx running on the server was ThingWorx 8. The Humia sensors used were also setup and working, therefore no set up was required.

4.1 Requirements

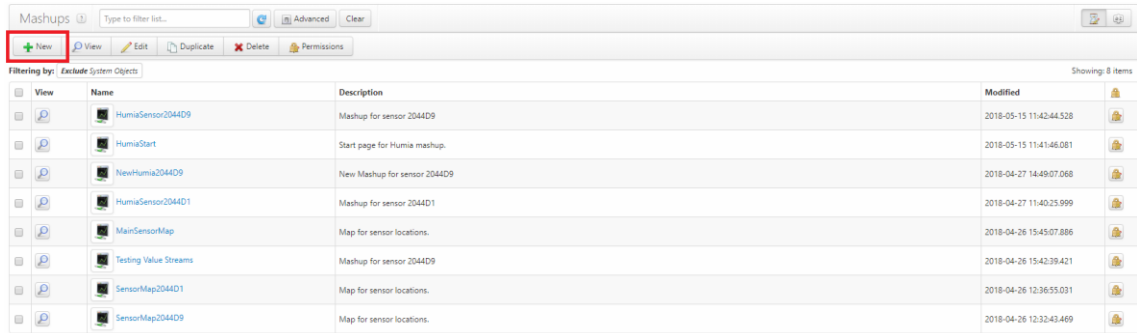
A few requirements for the application were given by the commissioner at the beginning of the project. The requirements or problems to be solved by the application were mentioned in the introduction, but they will now be discussed in more detail.

One of the requirements was to display the time series nature of the data. Simply put, this means the changes in data over time wanted to be seen. This is useful for finding patterns or trends in the data. To meet this requirement the data was to be displayed in a time series graph where the x-axis is the date. As a result, the data over time can be seen and trends can easily be seen directly from the graph.

Another requirement was to show the spatial positioning of the sensors within the building. This requirement was to be met by adding a floor plan image to the application and then adding the sensor positions on top of the image. The positions of the sensors could then be seen by looking at the floor plan.

4.2 Creating Mockup Mashup

First, a mockup mashup was created to get a rough idea for the layout of the application. The mockup layout was created in ThingWorx using the mashup builder tool. The mockup did not have any bound data at this stage and was created for design purposes only. The mashup was created by clicking the button for a new mashup at the top left of the mashup page in ThingWorx, as can be seen in figure 5.



View	Name	Description	Modified
	HumiaSensor2044D9	Mashup for sensor 2044D9	2018-05-15 11:43:44.528
	HumiaStart	Start page for Humia mashup.	2018-05-15 11:41:46.081
	NewHumia2044D9	New Mashup for sensor 2044D9	2018-04-27 14:49:07.068
	HumiaSensor2044D1	Mashup for sensor 2044D1	2018-04-27 11:40:25.999
	MainSensorMap	Map for sensor locations.	2018-04-26 15:45:07.886
	Testing Value Streams	Mashup for sensor 2044D9	2018-04-26 15:42:39.421
	SensorMap2044D1	Map for sensor locations.	2018-04-26 12:36:55.031
	SensorMap2044D9	Map for sensor locations.	2018-04-26 12:32:43.469

Figure 5. Create a new mashup in ThingWorx.

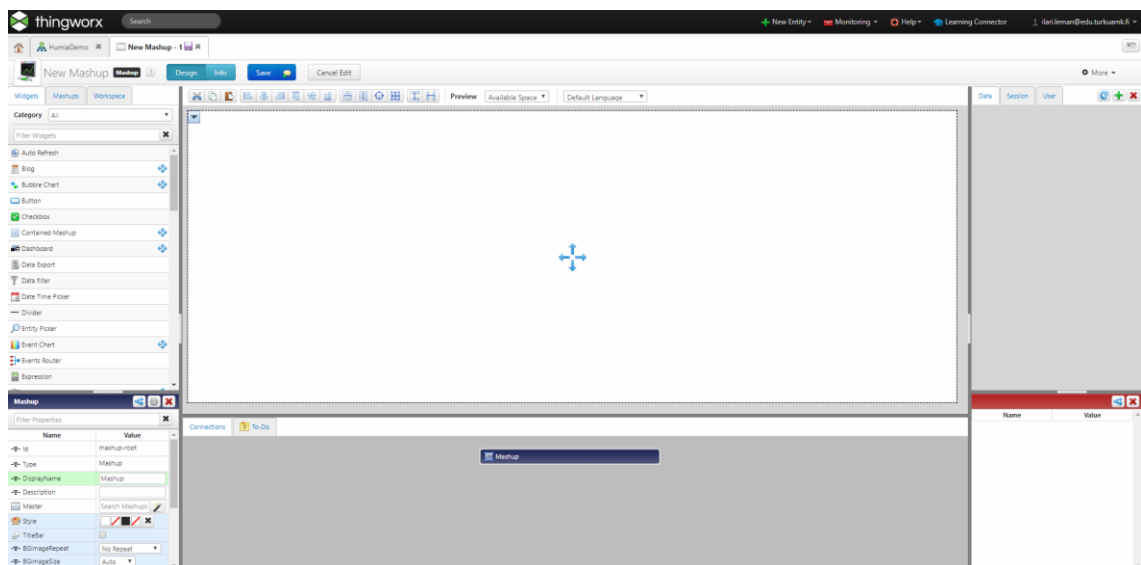


Figure 6. Mashup editor in ThingWorx.

In figure 6, the mashup editor can be seen. The middle of the editor is the preview of the mashup. Widgets can be added to the mashup by dragging and dropping them from the left sidebar onto the mashup in the center of the editor. This is the method which was used to create the mockup mashup in figure 7.



Figure 7. Mockup mashup for the application.

The table in the top left of the application was designed to display the direct data from the Humia cloud. A floor plan was added to display the location of the sensors and the gauges show the current/most recent data from the cloud. For this project, there was a small hempcrete building where the sensors were placed. No floor plan for this building was given, therefore a model floor plan was used.

The mockup mashup was later changed to improve on the design and make the application more user-friendly. A graph was added at the bottom left under the table of values to show the history of the values and trend. The gauges were changed to led displays showing the values as numbers due to it being more a more convenient way to display them. A menu was also added to the top of the application. From the menu the sensor could be selected based on the ID of the sensor. A different mashup was created for both of the sensors due to the different values each sensor measured.

Creating a floor plan with clickable buttons as the sensors was done by creating a separate mashup. The mashup background image covered the whole mashup and was an image of the floor plan. The sensors were navigation buttons which floated on top of the background images. When the navigation button for a sensor was pressed, the mashup for that sensor would open.

4.3 Connecting Cloud Data to ThingWorx

ThingWorx by default, has support for REST API connections, therefore connecting the Humia data via REST API was straightforward. A Thing object was created for both sensors used in the project. This was done by navigating to the modeling tab in the composer and selecting the Things page from the menu. Once on the Things page, the new button at the top left was pressed, similar to creating a mashup. The sensor ID of each sensor was used as a name for the Things created.

After the Thing objects were created, services were added to those Things by using the services tab within the Things page. The services were created by slightly modifying the built-in code snippets to send a REST API request for the 500 most recent data points from the Humia cloud (Appendix 1 & 2). The output of this service was a JSON file with all of the data.

The JSON output received from the service mentioned in the paragraph above was then used to create a table holding the values. This was done by creating another service which accepted JSON files as an input. The service then parsed the JSON file and created the table column attributes and added a new row for each of the data points in the JSON file (Appendix 3 & 4). The output of this service was then used to display on the application.

Finally, services were created to query the table to remove values which were not needed. There were services created for querying only the temperature, humidity and pressure separately. These services were for creating data which could be used for displaying the graphs of each property (Temperature, humidity and pressure).

4.4 Connecting Data to Mashup

Connecting the data received from the services created to the mashup was quick and easy. In the mashup editor, the services were added to the mashup from the top right of the editor by pressing the button with the green plus sign, as seen in figure 8. Once the button was pressed, the wanted Services were selected and added to the mashup's data. The Service's data could then be accessed from the right-hand side of the editor under the data tab.

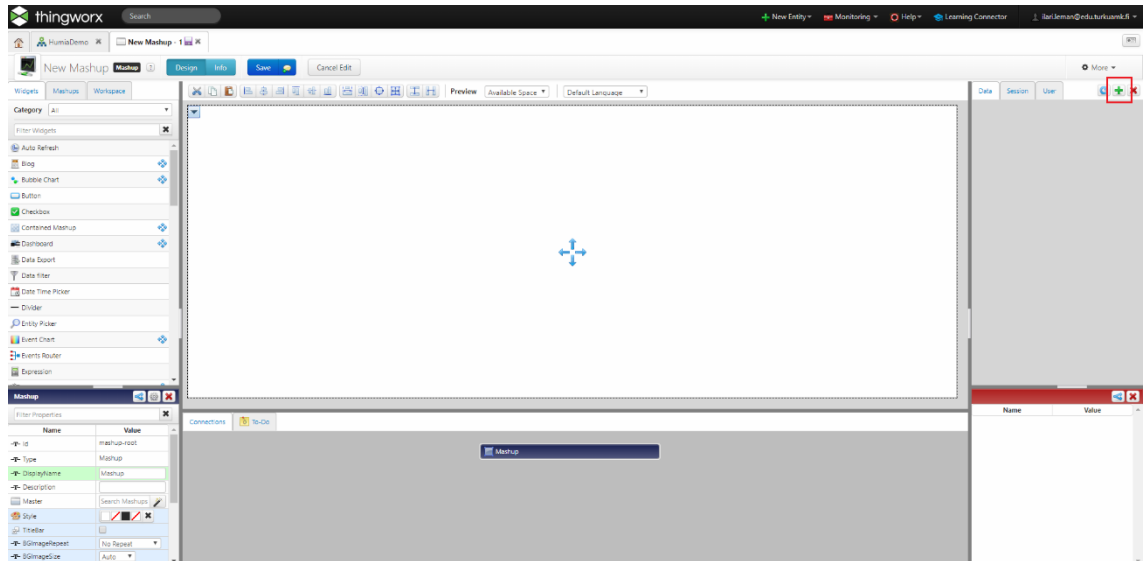


Figure 8. Button for adding service data, top right of the mashup builder.

To add the data to the mashup itself, the data was simply dragged from the data tab onto the desired widget on the mashup. Figure 9 shows how the data is dragged into a widget.

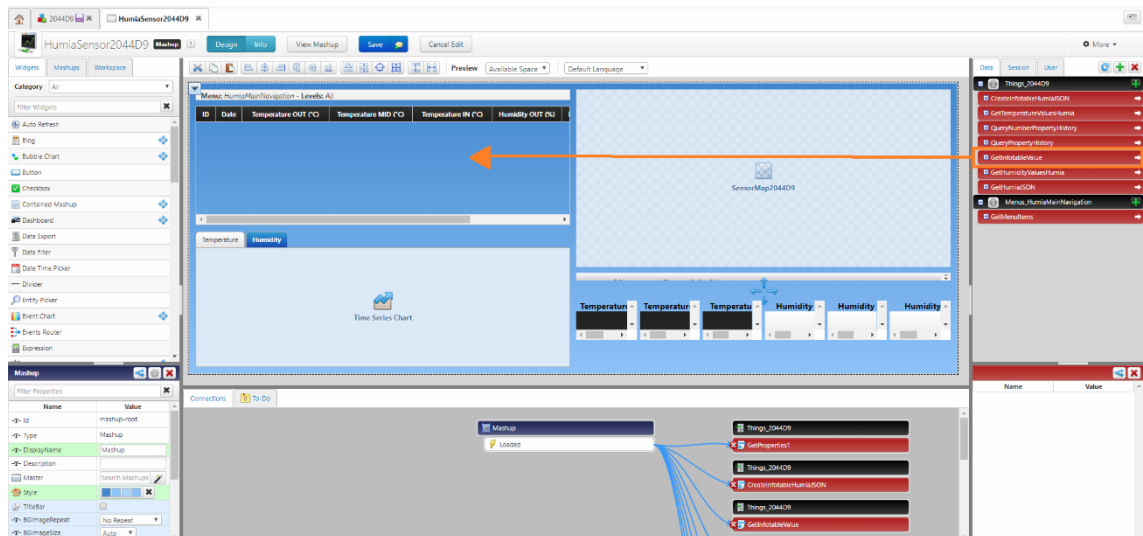


Figure 9. Adding data to a widget in the mashup editor.

4.5 Styling Mashup

The styling of the mashup was done within the mashup editor. The desired widget or mashup was clicked on within the editor, this opened a tab on the bottom left of the editor with aspects of the widget which could be customized as seen in figure 10. The properties such as colors and text were customized here.

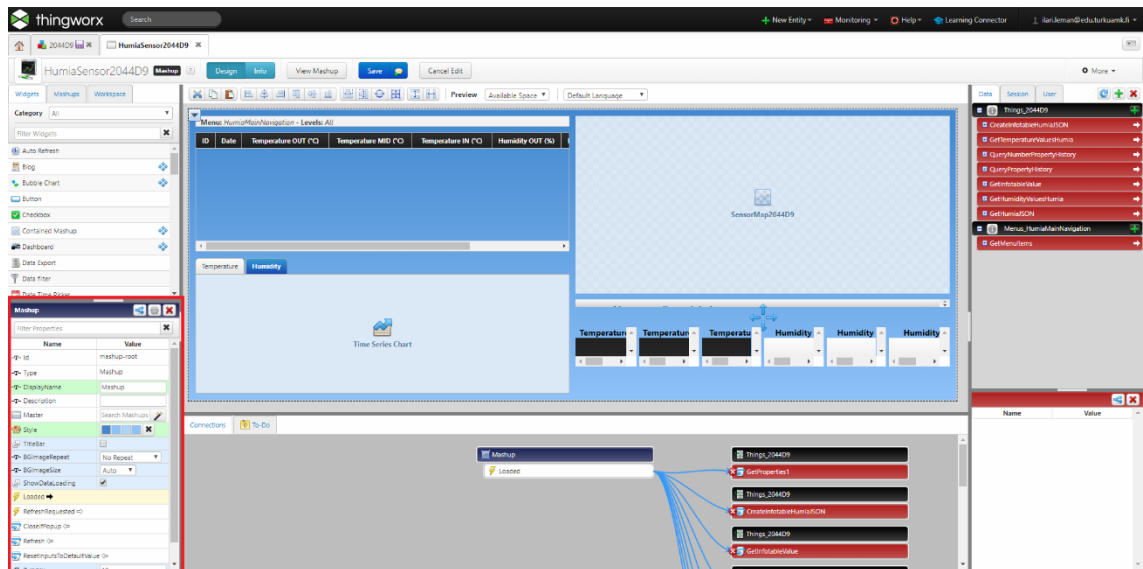


Figure 10. Widget properties tab in the mashup editor.

4.6 Results

The final version of the application looked pleasant and was easy to use. The application had a large table for displaying the data points from the Humia cloud in the top right. Below it there was a graph selector where the user can select the graph they wanted to see, temperature, humidity or pressure. The right side of the application had the floor plan image and the sensor locations on top of the image. As well as, the date for the latest data point received from the Humia cloud and the values for that data point. The final application can be seen in the figure below.

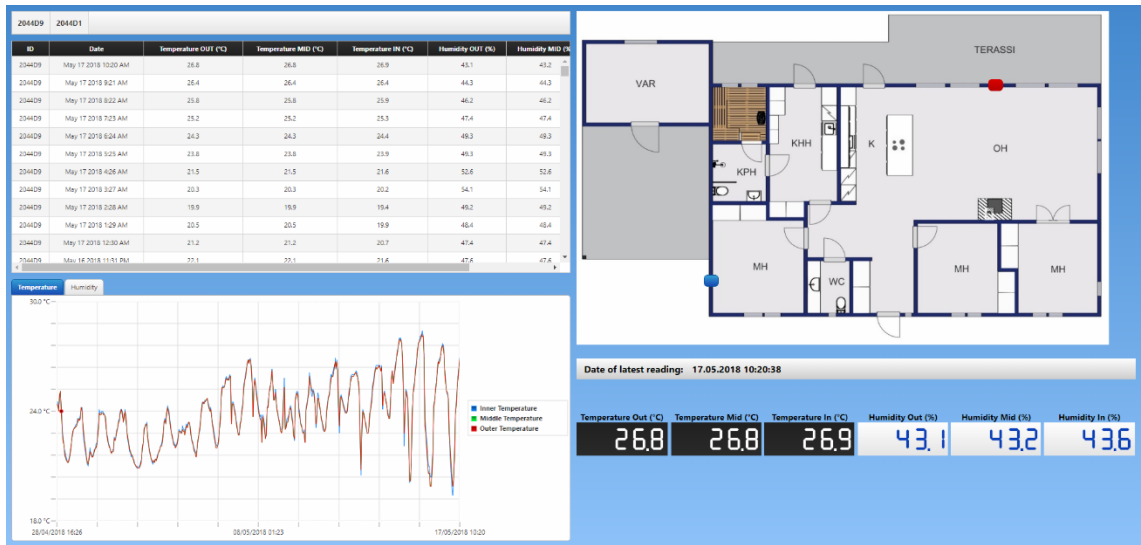


Figure 11. Final layout of the application created.

5 CONCLUSION

The goal of this work was to create a user-friendly IoT application which would be used to monitor the health of buildings. To achieve this goal, an application was to be created using ThingWorx, showing the data sent from sensors located in the building. The sensors measured the temperature, humidity, pressure and then sent these values to a cloud using a radio transmitter. Data from the cloud would then be connected to the ThingWorx application via REST API and displayed in useful form for monitoring the health.

As for the process of creating the application, much was learned about the ThingWorx software. ThingWorx was a new software, which had never been used by the author and was quite challenging to start working with. The development of the application went as expected no large issues were faced. The final application was solid and fulfilled the goals of the author.

The result of the project was an application which is easy to use and understand. The application shows the location of the sensors linked to the application on a floor plan, as well as, the data from those sensors. The data is displayed in 3 different ways, a table showing the values, a graph showing the trend of the values and led displays showing the most recent data.

To further improve the application, storing the sensor data on the ThingWorx platform could be done. Because of storing the data on the ThingWorx platform, loading times of the application could be significantly reduced. Currently, the application sends a REST API request every time it launches for 500 data points. Whereas, if the data was on the ThingWorx platform, it could be used instantly by the system. Furthermore, if the data was stored on the ThingWorx platform, data over a greater period of time could be used. As a result, long term trends for periods of over a year could be seen.

Another improvement would be to send a warning message to the user of the application if there is a change for mold or other issues. Adding this functionality could help reduce further damage to the building, due to the user being alerted. The Humia REST API currently sends a value for mold risk, which was not taken advantage of in this work. Additionally, ThingWorx has the capability to send email alerts to a user's email. Therefore, this feature could be implemented if the application was developed further.

REFERENCES

Auriga.com. (2016). *Embedded Systems as Part of the Internet of Things Ecosystem* | Auriga. [online] Available at: <https://auriga.com/blog/2016/embedded-systems-as-part-of-the-internet-of-things-ecosystem/> [Accessed 10 May 2018].

Liew, Z. (2018). *Understanding And Using REST APIs*. [online] Smashing Magazine. Available at: <https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/> [Accessed 18 May 2018].

Ptc.com. (2018). *Industrial IoT | PTC*. [online] Available at: <https://www.ptc.com/en/products/iot> [Accessed 1 May 2018].

Standard ECMA-404 The JSON Data Interchange Syntax. (2017). 2nd ed. [ebook] ecma INTERNATIONAL. Available at: <https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> [Accessed 12 May 2018].

Statista. (2018). *IoT: number of connected devices worldwide 2012-2025* | Statista. [online] Available at: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/> [Accessed 22 Apr. 2018].

Heading of appendix

Appendix 1. Script used to get JSON file from Humia cloud for sensor 2044D9

```
var params = {
    proxyScheme: false /* STRING */,
    headers: undefined /* JSON */,
    ignoreSSLErrors: undefined /* BOOLEAN */,
    useNTLM: undefined /* BOOLEAN */,
    workstation: undefined /* STRING */,
    useProxy: false /* BOOLEAN */,
    withCookies: undefined /* BOOLEAN */,
    proxyHost: undefined /* STRING */,
    url: "http://cloud.humia.fi/api/v0/a71cf11c-4920-5239-a6e9-7ab57796d9a9/devices/2044D9?count=500" /* STRING */,
    timeout: 500 /* NUMBER */,
    proxyPort: undefined /* INTEGER */,
    password: undefined /* STRING */,
    domain: undefined /* STRING */,
    username: undefined /* STRING */
};

// result: JSON
var result = Resources["ContentLoadedFunctions"].getJSON(params);
```

Appendix 2. Script used to get JSON file from Humia cloud for sensor 2044D1

```
var params = {
    proxyScheme: false /* STRING */,
    headers: undefined /* JSON */,
    ignoreSSLErrors: undefined /* BOOLEAN */,
    useNTLM: undefined /* BOOLEAN */,
    workstation: undefined /* STRING */,
    useProxy: false /* BOOLEAN */,
    withCookies: undefined /* BOOLEAN */,
    proxyHost: undefined /* STRING */,
    url: "http://cloud.humia.fi/api/v0/a71cf11c-4920-5239-a6e9-
7ab57796d9a9/devices/2044D1?count=500" /* STRING */,
    timeout: 500 /* NUMBER */,
    proxyPort: undefined /* INTEGER */,
    password: undefined /* STRING */,
    domain: undefined /* STRING */,
    username: undefined /* STRING */
};

// result: JSON
var result = Resources["ContentLoaderFunctions"].getJSON(params);
```

Appendix 3. Script used to parse JSON file and create infotable for sensor 2044D9

```
//set input to Humia JSON file
json = me.GetHumiaJSON();

var params = {
  infoTableName: "InfoTable",
  dataShapeName : "sensor_2044D9"
};

var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);

////go through JSON and create/add row on infotable for each data point
for(var i=0; i<json.data.length; i++) {
  result.AddRow({deviceId:json.data[i].device,
    timestamp:json.data[i].timestamp,
    unixTime:json.data[i].unixTime*1000, //multiplied by 1000 to get
correct date (issue with the DATETIME basetype in TW)
    outerTemp:json.data[i].temperature.sensor_1,
    middleTemp:json.data[i].temperature.sensor_2,
    innerTemp:json.data[i].temperature.sensor_3,
    outerHumidity:json.data[i].humidity.sensor_1,
    middleHumidity:json.data[i].humidity.sensor_2,
    innerHumidity:json.data[i].humidity.sensor_3});
}

//unixTime is 3 hours ahead - takes 3 hours off the unixTime values
var params2 = {
  t: result /* INFOTABLE */,
  shift: -10800 /* NUMBER */,
  timestampField: "unixTime" /* STRING */
};

// result: INFOTABLE
var result = Resources["InfoTableFunctions"].TimeShift(params2);
```


Appendix 4. Script used to parse JSON file and create infotable for sensor 2044D1

```
//set input to Humia JSON file
json = me.GetHumiaJSON();

var params = {
  infoTableName: "InfoTable",
  dataShapeName : "sensor_2044D1"
};

var result =
Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);

//go through JSON and create/add row on infotable for each data point
for(var i=0; i<json.data.length; i++) {
  result.AddRow({deviceId:json.data[i].device,
    timestamp:json.data[i].timestamp,
    unixTime:json.data[i].unixTime*1000, //multiplied by 1000 to get
correct date (issue with the DATETIME basetype in TW)
    temperature:json.data[i].temperature.sensor_1,
    humidity:json.data[i].humidity.sensor_1,
    pressure:json.data[i].pressure.sensor_1});
}

//unixTime is 3 hours ahead - takes 3 hours off the unixTime values

var params2 = {
  t: result /* INFOTABLE */,
  shift: -10800 /* NUMBER */,
  timestampField: "unixTime" /* STRING */
};

// result: INFOTABLE
var result = Resources["InfoTableFunctions"].TimeShift(params2);
```