

DEI-ISEP

# WEARABLE T-SHIRT

Informatic Security

2015/2016

**1120364 Fernando Esparrinha Pinto**



DEI-ISEP

# WEARABLES T-SHIRT

2015/2016

**1120364 Fernando Esparrinha Pinto**



Degree in Informatics Engineering

July 2016

Orientador ISEP: Jorge Pinto Leite

Supervisor Externo: Paulo Patrício



## ACKNOWLEDGEMENTS

I would like to express my sincere thanks to all who made this project possible, to the reader, who is taking his time to read this report and, especially, to four big groups of people:

To my family, for believing in me and giving me the opportunity to follow what I really like, supporting me in every way possible.

To Celfocus, for giving me the opportunity to participate in this project, allowing me not just to develop but also to contribute with ideas and opinions. Also to Celfocus employees, especially to my supervisor Paulo Patricio, for being always available to answer my questions and giving support whenever was needed and to João Gomes, João Rodrigues and Tiago Gonçalves, which were also making PESTI projects, like myself, and gave continuous support to one another.

To DEI-ISEP, for giving me the knowledge needed to evolve, not only making me a developer, but teaching me to think like an engineer and for also offering me its services whenever needed. A special thanks to professors Jorge Pinto Leite, Nuno Silva, Paulo Proença, Constantino Martins and Orlando Sousa for being excellent professors and improving me not only as a professional but also as a person. To professor Jorge Pinto Leite an extra special thanks for being my advisor and giving me all the help needed.

And last but not least, to all my friends which helped me, giving moral support and being the great friends that they are.

A big Thank you to all.



## ABSTRACT

In modern times, the number of everyday objects connected to the internet is growing exponentially, this number got so big in the last years that a term was coined in 1999, by Kevin Ashton, to describe the network created by these objects, Internet of Things (IoT). By being connected to the internet, these objects can send information that otherwise wouldn't be sent by any other device, such as amount of food in the user's fridge or how many lights are turned on in the user's house.

To take advantage of this network, Biodevices – Sistemas de Engenharia S.A. created Vital Jacket, a T-Shirt with a reliable cardio monitor that records vital hearth information and makes it available for the user and user's doctor to see. Celfocus bought two Vital Jackets in order to integrate them with the IoT platform ThingWorx.

In order to integrate the Vital Jackets with ThingWorx a tool has to be developed, this tool has to use the Vital Jacket SDK to obtain the user's information and must send the information in a secure way, with confidentiality and integrity, to ThingWorx. Since the information that is going to be shared is fragile, the developed tool must also have great security and follow a big number of protocols, rules and standards.

On the end, a successful proof of concept was achieved, which demonstrated a scalable working solution. The objective of the project was mainly completed on time, leaving only some secondary tasks to be developed.

Keywords (Theme): Internet of Things, Wearables, Informatics Security

Keywords (Technologies): ThingWorx, IPv6, Encryption

## RESUMO

Nos tempos modernos, o número de objetos do dia-a-dia conectados à internet tem crescido exponencialmente, este número aumentou de tal maneira que um termo foi inventado em 1999, por Kevin Ashton, para descrever a rede que é criada por estes objetos, Internet of Things (IoT). Ao estarem ligados à internet, estes objetos podem enviar informações que, de outra maneira, não seriam enviadas por nenhum outro dispositivo, tal como a quantidade de comida que o utilizador tem no seu frigorífico ou a quantidade de luzes que estão ligadas na casa do utilizador.

Para tirar uso desta rede, a Biodevices – Sistemas de Engenharia S.A criou a Vital Jacket, uma T-Shirt que contém um monitor de cardio que grava informação sobre a saúde do utilizador e torna esta informação disponível ao utilizador e ao seu médico. A Celfocus comprou duas Vital Jackets para integrar na plataforma IoT ThingWorx.

De maneira a integrar a Vital Jacket na ThingWorx, uma ferramenta tem de ser desenvolvida, esta ferramenta tem de obter a informação do utilizador, usando o SDK da Vital Jacket, e enviar a informação de maneira segura, com confidencialidade e integridade, para a ThingWorx. Visto que a informação que será partilhada é bastante frágil, a ferramenta desenvolvida deve possuir um bom nível de segurança e seguir um grande número de protocolos, regras e normas.

No final, uma prova de conceito foi alcançada, demonstrando uma solução final escalável. O projeto foi maioritariamente completo a tempo, deixando só algumas tarefas secundárias por desenvolver.

Palavras-chave (Tema): Internet das coisas, *Wearables*, Segurança Informática

Palavras-chave (Tecnologias): ThingWorx, IPv6, Encriptação



## GLOSSARY

<i>Notation</i>	<i>Description</i>
<i>AES</i>	Advanced Encryption Standard – Symmetric-key algorithm based on Rijndael cipher, used to encrypt electronic data.
<i>API</i>	Application Programming Interface – Set of routines, protocols and tools that makes the development of a program easier by providing building functions.
<i>CoAP</i>	Constrained Application Protocol – Communication protocol intended to be used by very simple electronic devices, like the ones usually used in IoT. Uses UDP.
<i>Computing Platform</i>	Whatever a pre-existing piece of computer software is designed to run within, obeying its constraints, and making use of its facilities. <sup>1</sup>
<i>ECG</i>	Electrocardiography - Test that checks for problems with the electrical activity of the patient's heart. <sup>2</sup>
<i>HL7</i>	Health Level Seven - Framework (and related standards) provided for the exchange, integration, sharing, and retrieval of electronic health information. These standards define how information is packaged and communicated from one party to another, setting the language, structure and data types required for seamless integration between systems. <sup>3</sup>
<i>IFTTT</i>	If This Then That – web service that connects a large amount of other popular web services.

<sup>1</sup> Source: [https://en.wikipedia.org/wiki/Computing\\_platform](https://en.wikipedia.org/wiki/Computing_platform)

<sup>2</sup> Source: <http://www.webmd.com/heart-disease/electrocardiogram>

<sup>3</sup> Source: <http://www.hl7.org/implement/standards/>

<i>IoT</i>	Internet of Things - Network of physical objects—devices, vehicles, buildings and other items embedded with electronics, software, sensors, and network connectivity—that enables these objects to collect and exchange data. <sup>4</sup>
<i>IP</i>	Internet Protocol – Principal communication protocol for relaying datagrams across network boundaries.
<i>IPv6</i>	Most recent version of IP. Deals with the problem of IPv4 address exhaustion.
<i>M2M</i>	Machine to Machine – Broad label that can be used to describe any technology that enables networked devices to exchange information and perform actions without the manual assistance of humans. <sup>5</sup>
<i>MQTT</i>	Message Queue Telemetry Transport – Communication protocol intended to be used by very simple electronic devices, like CoAP, but uses TCP/IP
<i>OOP</i>	Object Oriented Programming – A programming paradigm based in objects rather than in logic.
<i>OS</i>	Operating System – A software that manages the hardware and software of a computer, providing essential services for the computer programs.
<i>PTC</i>	Parametric Technology Corporation – computer software company owner of the ThingWorx platform.
<i>QoS</i>	Quality of Service – A way of measuring the overall performance of a network.

---

<sup>4</sup> Source: [https://en.wikipedia.org/wiki/Internet\\_of\\_Things](https://en.wikipedia.org/wiki/Internet_of_Things)

<sup>5</sup> Source: <http://internetofthingsagenda.techtarget.com/definition/machine-to-machine-M2M>

<i>QRS</i>	Record of the movement of electrical impulses through the lower heart chambers. <sup>6</sup> It's the central and most visually obvious part of the tracing.
<i>REST</i>	Representational state transfer – software architectural style used to connect systems easily.
<i>SDK</i>	Software Development Kit – a set of software development tools that allow the creation of applications for a specific hardware platform.
<i>TCP (also referenced as TCP/IP)</i>	Transmission Control Protocol – protocol that provides a reliable, connection-oriented packet delivery service. <sup>7</sup>
<i>ThingWorx</i>	IoT technology platform used to connect ecosystems, create and deploy applications, analyze connected objects and deliver user experience across different devices.
<i>TLS</i>	Transport Layer Security – cryptographic protocol designed to provide security over a communication with two devices on a network. Its main goal is to provide privacy and data integrity.
<i>UDP</i>	User Datagram Protocol – protocol that provides fast, lightweight, unreliable transportation of data between TCP/IP hosts. <sup>8</sup>
<i>URL</i>	Uniform Resource Locator – a reference to a web resource. Usually refers to websites but can also refer to file transfer, email and many other applications

---

<sup>6</sup> Source: <http://www.webmd.com/heart/ekg-components-and-intervals>

<sup>7</sup> Source: <https://technet.microsoft.com/en-us/library/cc756754%28v=ws.10%29.aspx>

<sup>8</sup> Source: <https://technet.microsoft.com/en-us/library/cc785220%28v=ws.10%29.aspx>

*Vital Jacket*

A wearable T-Shirt that contains a reliable cardio monitor and can be worn by the patient allowing at least 72 hours of continuous exams.<sup>9</sup>

*WBS*

Work Breakdown Structure – decomposition of the project in smaller tasks that can be manageable in an easier way.

*Wearables*

Technology that is wearable, such as clothes or accessories. Incorporates advanced electronic technologies, giving practical functions that otherwise these objects wouldn't have.

*XML*

Extensible Markup Language – language that defines a set of rules for encoding documents in a simple and practical way.

*XMPP*

Extensible Messaging Presence Protocol – a communication protocol based on XML that is mainly used in messaging apps. It enables near-real-time communication.

---

<sup>9</sup> Source: [http://www.vitaljacket.com/?page\\_id=156](http://www.vitaljacket.com/?page_id=156)

# CONTENTS

<b>Acknowledgements .....</b>	<b>V</b>
<b>Abstract .....</b>	<b>VII</b>
<b>Resumo .....</b>	<b>VIII</b>
<b>Glossary .....</b>	<b>IX</b>
<b>Contents.....</b>	<b>XIII</b>
<b>Index of Figures .....</b>	<b>XV</b>
<b>Index of Tables .....</b>	<b>XVII</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
<b>1.1 The Project .....</b>	<b>2</b>
1.1.1 Planning .....	4
1.1.2 Meetings .....	5
<b>1.2 Selected Technologies .....</b>	<b>7</b>
1.2.1 ThingWorx IoT Platform.....	8
1.2.2 Vital Jacket SDK.....	9
1.2.3 Mosquitto .....	10
1.2.4 Virtualbox, Linux and Wireshark.....	11
1.2.5 SVN .....	11
1.2.6 IFTTT .....	12
<b>1.3 The Organization .....</b>	<b>13</b>
<b>1.4 Document Structure.....</b>	<b>16</b>
<b>Chapter 2 Problem Background.....</b>	<b>17</b>
<b>2.1 Business areas .....</b>	<b>18</b>
2.1.1 Healthcare .....	18
2.1.2 Technology.....	18
<b>2.2 State of Art .....</b>	<b>19</b>
2.2.1 IoT Platforms.....	19
2.2.2 Communication .....	26
2.2.3 Encryption.....	29
2.2.4 Possible Solutions .....	32
<b>Chapter 3 Workflow.....</b>	<b>35</b>
<b>3.1 Working Methodology .....</b>	<b>36</b>
3.1.1 Waterfall Methodology .....	36

3.1.2	Confluence .....	37
3.1.3	Jira.....	37
3.1.4	SVN .....	38
<b>3.2</b>	<b>Technical Description .....</b>	<b>39</b>
3.2.1	Diagrams .....	39
3.2.2	Instalations .....	43
3.2.3	Solution Development .....	45
3.2.4	Analysis .....	58
3.2.5	Tests.....	60
<b>Chapter 4</b>	<b>Conclusion .....</b>	<b>65</b>
4.1	Summary.....	66
4.2	Objectives Complete .....	66
4.3	Limitations and Future Work .....	67
4.4	Other Work .....	68
4.5	Final Appreciation.....	69
<b>Chapter 5</b>	<b>Bibliography .....</b>	<b>71</b>
<b>Chapter 6</b>	<b>Appendixes .....</b>	<b>73</b>
6.1	Full WBS of the project with Gantt Chart .....	74
6.2	MQTT Packet .....	75
6.3	Project domain model.....	76
6.4	First Simulated Vital Jacket MQTT Client.....	77
6.5	Deployment Server Endpoints .....	76
6.6	Project Scope.....	77
6.7	Vital Jacket State Machine .....	78
6.8	State of Progress.....	79

## INDEX OF FIGURES

Figure 1 - Project scope.....	2
Figure 2- The evolution of connected devices (Schneider, 2013).....	3
Figure 3- WBS Gantt chart .....	4
Figure 4 - Thingworx logo and platform homepage .....	8
Figure 5 - Vital Jacket logo and Windows version download page.....	9
Figure 6 - Mosquitto logo and manual page.....	10
Figure 7 - Wireshark running on Linux Ubuntu on a Virtual Machine .....	11
Figure 8 - SVN logo.....	11
Figure 9 - Some IFTTT available services and logo.....	12
Figure 10 - Celfocus offices around the world (Celfocus, 2016) .....	13
Figure 11 - Celfocus successful projects (Celfocus, 2016) .....	14
Figure 12 - Celfocus Business Areas.....	14
Figure 13 - ThingWorx Composer main screen.....	21
Figure 14 - ThingWorx information structure.....	22
Figure 15 - ThingWorx Property creation .....	23
Figure 16 - ThingWorx Service creation .....	24
Figure 17 - Main projects that use XMPP (XMPP, s.d.) .....	26
Figure 18 - Diagram illustrating CoAP implementation .....	27
Figure 19 - Diagram illustrating MQTT implementation.....	28
Figure 20 – WebSockets (WebSockets vs REST: Understanding the Difference, 2015) .....	29
Figure 21 - Actual solution .....	32
Figure 22 - Ideal Solution .....	33
Figure 23 - Another Ideal Solution .....	33
Figure 24 - Waterfall Diagram.....	36
Figure 25 - SVN Repository .....	38
Figure 26 - SVN Repository trunk folder .....	39
Figure 27 - ThingWorx Diagram - Vital Jacket.....	40
Figure 28 - Thingworx Diagram - Events .....	41
Figure 29 - Components Diagram .....	42
Figure 30 - Deployment Diagram.....	43
Figure 31, 32 and 33 - ThingWorx Things, Templates and Shapes .....	45
Figure 34 - MQTT ThingWorx connection settings .....	46
Figure 35 - Mosquitto broker running .....	47
Figure 36 - MQTT Client running.....	49
Figure 37 – MQTT information flow.....	50

Figure 38 - Vital Jacket SDK.....	50
Figure 39 – Updated Vital Jacket SDK .....	52
Figure 40 - Created action.....	53
Figure 41 – Action creation interface .....	54
Figure 42 - Created trigger .....	55
Figure 43 - Trigger creation interface .....	56
Figure 44 - IFTTT API Call.....	57
Figure 45 - TCP MQTT Packet.....	58
Figure 46 - Visual Studio Diagnostic Tool.....	59
Figure 47 - Complete WBS of the project and Gantt Chart .....	74
Figure 48 - MQTT Packet.....	75
Figure 49 - Project domain model.....	76
Figure 50 - First Simulated Vital Jacket MQTT Client.....	77
Figure 51 - Deployment Server Endpoints .....	76
Figure 52 - Project Scope .....	77
Figure 53 - Vital Jacket State Machine .....	78
Figure 54 - Page 1 of State of Progress .....	79
Figure 55 - Page 2 of State of Progress .....	79
Figure 56 - Page 3 of State of Progress .....	80
Figure 57 - Page 4 of State of Progress .....	80
Figure 58 - Page 5 of State of Progress .....	81
Figure 59 - Page 6 of State of Progress .....	81
Figure 60 - Page 7 of State of Progress .....	82
Figure 61 - Page 8 of State of Progress .....	82



## INDEX OF TABLES

Table 1- WBS .....	4
Table 2 - Meetings had with Celfocus supervisor during the Internship .....	5
Table 3 - Technologies selected in this project .....	7
Table 4 - IoT Platforms .....	19
Table 5 - Encryption algorithms .....	30
Table 6 - Pros and Cons of the possible solutions.....	34
Table 7 - Functional and Non-functional requisites.....	60
Table 8 - Mosquitto test script.....	61
Table 9 - Script for MQTT client testing .....	62
Table 10 – IFTTT tests.....	63
Table 11 - State of Objectives .....	66



## CHAPTER 1 INTRODUCTION

This chapter is dedicated to an overall view of the Wearables T-Shirt project. It will describe a brief presentation of the project, technologies used and the organization where the project was developed, in order for the reader to be better contextualized with the project.

<b>Chapter 1</b>	<b>Introduction .....</b>	<b>1</b>
<b>1.1</b>	<b>The Project .....</b>	<b>2</b>
1.1.1	Planning .....	4
1.1.2	Meetings .....	5
<b>1.2</b>	<b>Selected Technologies .....</b>	<b>7</b>
1.2.1	ThingWorx IoT Platform .....	8
1.2.2	Vital Jacket SDK .....	9
1.2.3	Mosquitto .....	10
1.2.4	Virtualbox, Linux and Wireshark .....	11
1.2.5	SVN .....	11
1.2.6	IFTTT .....	12
<b>1.3</b>	<b>The Organization .....</b>	<b>13</b>
<b>1.4</b>	<b>Document Structure .....</b>	<b>16</b>

## 1.1 The Project

This project was developed in the context of Projecto/Estágio (PESTI) discipline, as an external internship. Its main goals are to develop an application that uses an ECG T-shirt (Vital Jacket) to connect the users' hearth rate information to an IoT platform in order to use that information in other applications such as fitness applications or IoT applications.

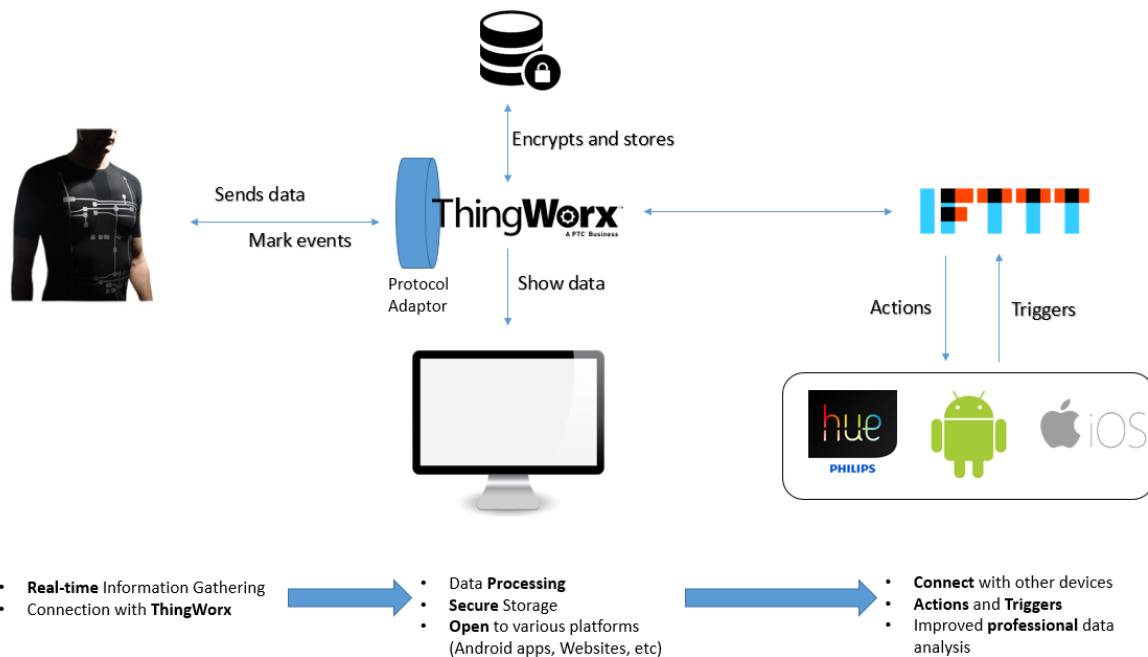


Figure 1 - Project scope

Figure 1 demonstrates the scope of the whole project, Vital Jacket connects with ThingWorx, storing the information in a secure way and also interacting with the IFTTT platform, which allows connection with other systems. ThingWorx, the main piece of the project, processes all the information and shows it to the user. IFTTT, allows the connection between ThingWorx and a numerous amount of available systems.

The project was split in two parts, the one worked in this project was the connection between Vital Jacket, the IoT platform and IFTTT, the other part was the development of two web portals that present the information available in the IoT platform.

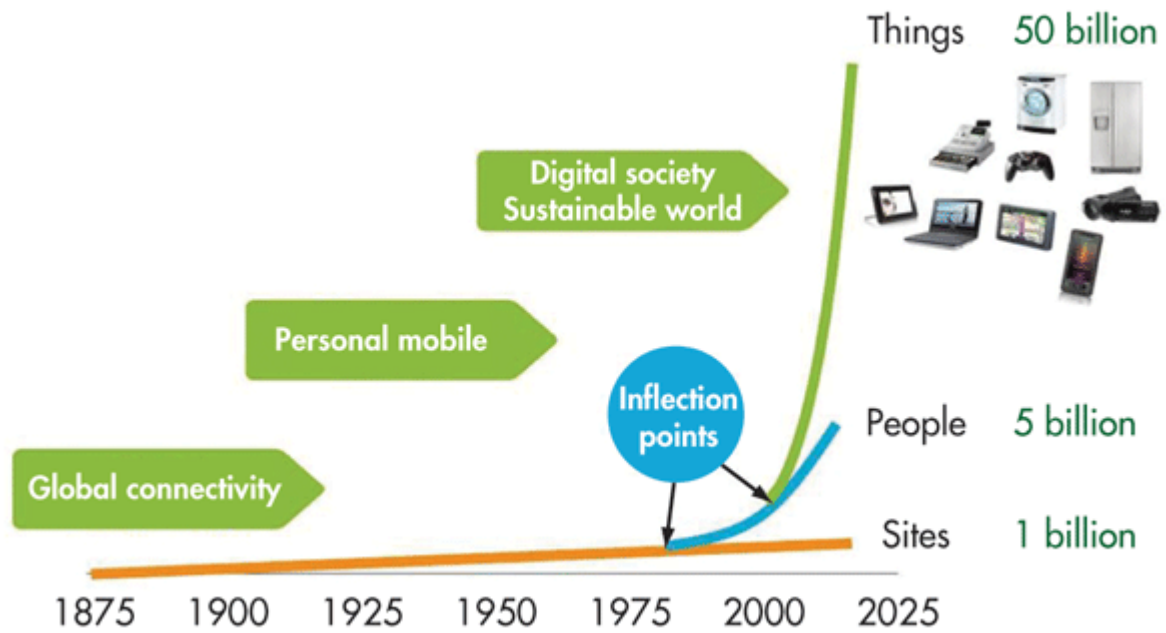


Figure 2- The evolution of connected devices (Schneider, 2013)

In figure 2 the evolution of the number of connected devices can be seen, this number is growing exponentially, which makes IoT a very big area to work on. With the huge amount of devices, also comes a huge amount of possible backdoors for hackers, which makes informatics security have an even greater importance. The choice of communication protocols, encryption methods and ways of storage should be taken seriously, with the full responsibility of all the possible attacks. (Schneider, 2013)

### 1.1.1 PLANNING

Planning a project is one of the most important tasks, and must be done in the beginning of the project in order to plan everything needed in the future. By the end of the planning phase, the developers should have a clear vision of the solution that they are going to implement, with report deliveries spread through the project, making an easier job to spot difficulties and obstacles. In this case, the project was split into different tasks, and each task was given a duration and begin and end dates.

Table 1- WBS

#	Task	Duration (days)	Begin	End
1	Preparation	10	29/02/2016	11/03/2016
2	Analysis	15	14/03/2016	01/04/2016
3	Design	15	04/04/2016	22/04/2016
4	Development	40	25/04/2016	17/06/2016
5	Demonstration	5	20/06/2016	24/06/2016
6	Report	80	07/03/2016	24/06/2016

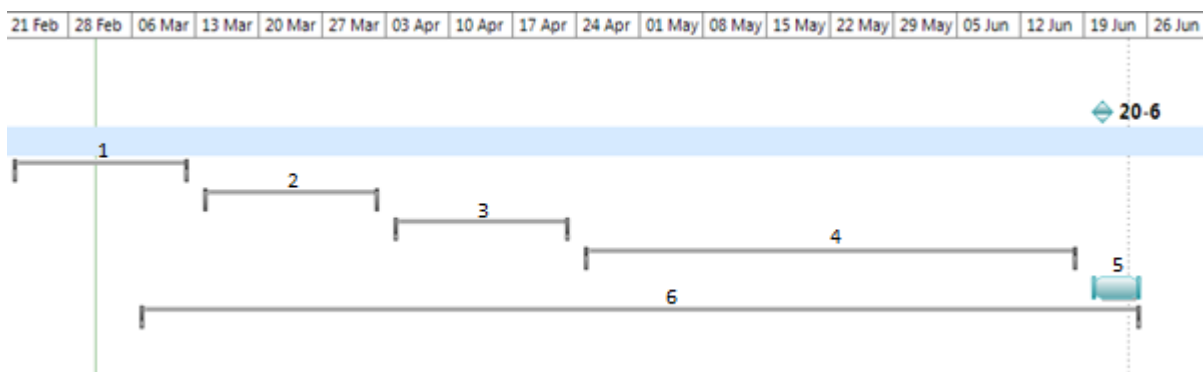


Figure 3- WBS Gantt chart

Table 1 shows the Work Breakdown Structure (WBS) of the project, which lets the reader identify the principal tasks that are planned and the timelines in which these tasks were made. Below Table 1, Figure 3 shows the Gantt chart of the WBS, which describes visually the duration of the tasks planned. The numbers 1 to 6 represent the different tasks and are the joining point between the WBS table and Gantt chart.

### 1.1.2 MEETINGS

During the project a large number of meetings were attended, Celfocus supervisor made sure to schedule at least one meeting per week with the members of the Wearables T-shirt project, meetings that were very valuable as information about the project got clearer with each meeting. Besides the weekly meetings, individual meetings could be schedule almost anytime and Celfocus supervisor was always available to answer questions through the internet.

**Table 2 - Meetings had with Celfocus supervisor during the Internship**

<b>Date</b>	<b>Subject</b>
<b>15 February 2016 - 18h00</b>	Kick-off session, presenting the project.
<b>29 February 2016 – 15h00</b>	Weekly meeting for project update.
<b>07 March 2016 – 14:00</b>	Meeting with Celfocus supervisor and ISEP advisor, defining internship objectives.
<b>11 March 2016 – 16:00</b>	Weekly meeting for project update.
<b>22 March 2016 – 14:15</b>	Weekly meeting for project update.
<b>31 March 2016 – 15:00</b>	Weekly meeting for project update.
<b>05 April 2016 – 16:30</b>	Weekly meeting for project update.
<b>13 April 2016 – 10:00</b>	Weekly meeting for project update.
<b>20 April 2016 – 11:10</b>	Weekly meeting for project update.
<b>27 April 2016 – 11:10</b>	Weekly meeting for project update.
<b>03 May 2016 – 11:40</b>	Weekly meeting for project update.
<b>19 May 2016 – 15:00</b>	Middle of Semester Presentation.
<b>20 May 2016 – 10:00</b>	Meeting with Vital Jacket SDK developer.
<b>25 May 2016 – 14:45</b>	Weekly meeting for project update.
<b>31 May 2016 – 14:50</b>	Weekly meeting for project update.
<b>08 June 2016 – 15:00</b>	Project Demonstration.
<b>14 June 2016 – 15:00</b>	Weekly meeting for project update.

In table 2 a list of the meetings had in this project, along with the subject of the same can be observed. This table only lists meetings had with Celfocus supervisor, since these meetings were schedule weekly and always introduced or clarified points of the project.

Besides Celfocus supervisor, there were various Celfocus collaborators that would help, whenever a problem would appear, via Skype. The resolution of bugs and clarification of doubts was highly assisted by them.

ISEP advisor was also very available to schedule meetings and answer questions either by e-mail or phone call, being a very valuable source of information that made this project much easier. Every time a reunion was needed in order to clarify ideas, answer questions about the report or present new suggestions, the ISEP advisor would schedule the meeting quickly.



## 1.2 Selected Technologies

To develop a solid solution several technologies had to be used, some of the technologies were already choose as the project was part of a bigger infrastructure of projects. This did not eliminate the need of a deep study of technologies, relating security and transfer of data.

**Table 3 - Technologies selected in this project**

Technology	Task	Version
<b>ThingWorx</b>	IoT platform - Storing and analyzing data, making it available for a developer to build an application with the data.	6.6.0
<b>Vital Jacket SDK</b>	Transferring data between Vital Jacket and ThingWorx.	1.0.07
<b>Mosquitto</b>	MQTT broker - Deliver published messages to respective subscribers.	1.4.8
<b>VirtualBox</b>	Store virtual operating systems in order to simulate the real solution.	5.0.18
<b>Linux</b>	Operating System, used to implement MQTT and ThingWorx.	Ubuntu 14.04
<b>Wireshark</b>	Used to analyze network packets in order to compare communication protocols.	2.0.3
<b>SVN</b>	Used for software versioning and revision control.	1.8.13
<b>IFTTT</b>	Web based service which joins other popular web services, such as Facebook or Twitter, with conditional statements.	Not Available

In table 3 the technologies selected to develop the project are described, together with the task where they were used and version, when available.

## 1.2.1 THINGWORX IOT PLATFORM<sup>10</sup>

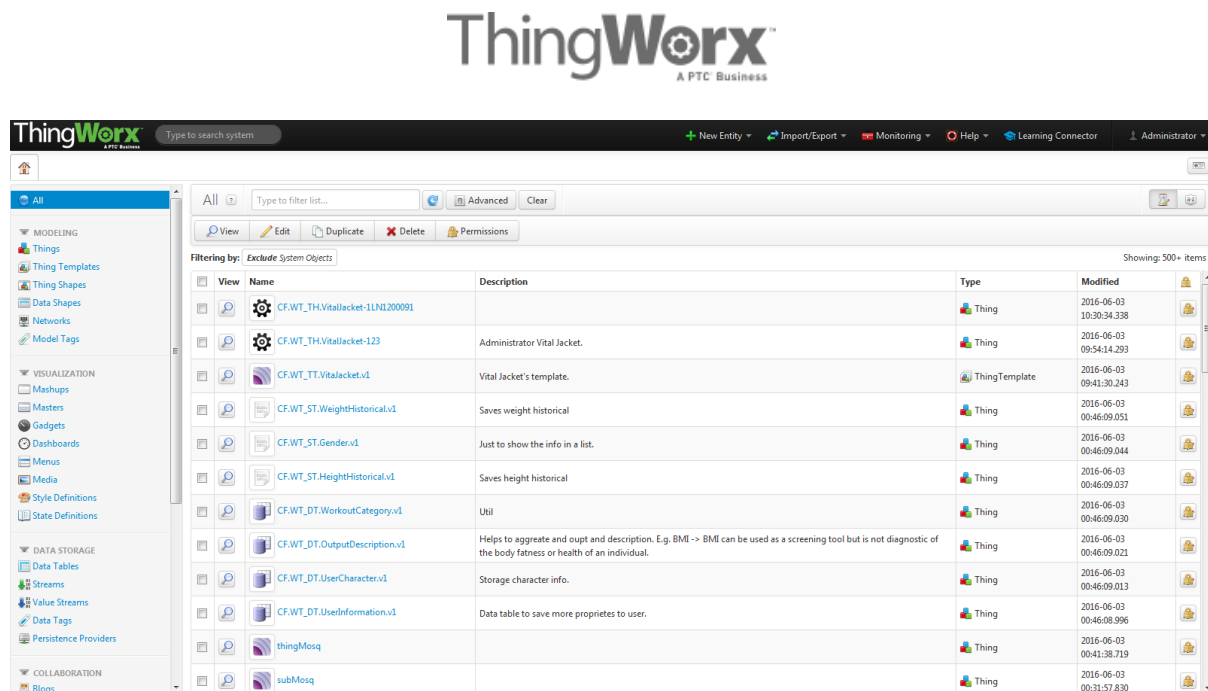


Figure 4 - Thingworx logo and platform homepage

ThingWorx is a business unit of Parametric Technology Corporation (PTC) and a \$1.4 billion global software company, where more than 700 employees are dedicated to the ThingWorx IoT technology platform. With more than \$600M invested in acquisitions and the development and integration of technologies for IoT, and with thousands of compatible partners and solutions, ThingWorx is creating innovation faster than ever thought possible. This IoT platform is the middle piece of the project, as it connects all the data with the respective users. (PTC, 2016)

Figure 4 demonstrates a future vision of the project, as it is the homepage of the platform with some instances, used in this project, already created and developed.

<sup>10</sup> <https://www.thingworx.com/>

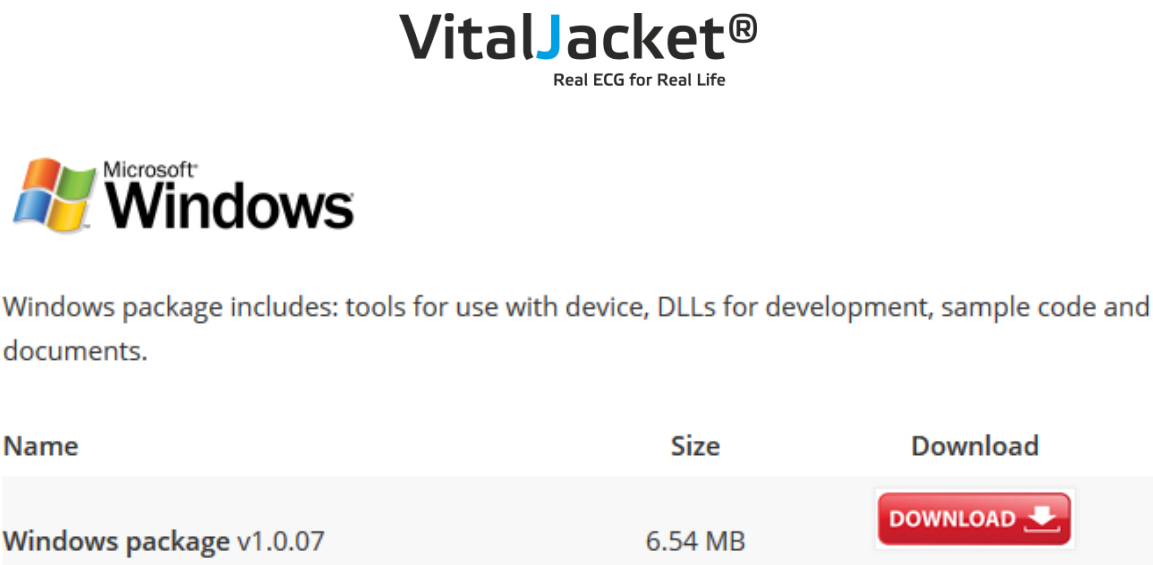
1.2.2 VITAL JACKET SDK<sup>11</sup>

Figure 5 - Vital Jacket logo and Windows version download page

Vital Jacket SDK is the development kit that gives access to information collected by the Vital Jacket, this SDK was critical to the project in order to integrate the Vital Jacket into ThingWorx IoT Platform, since it lets the developer capture all the information the Vital Jacket gathers. The SDK version used was the one for Windows OS, as it can be seen on figure 5, but there are also versions for Linux, Android, MatLab and Raspberry PI, these versions were still being improved as the project was being developed. Vital Jacket SDK developer made sure to answer all the questions and difficulties that appeared with the integration of the SDK into the project, providing powerful and useful information.

---

<sup>11</sup> <http://www.sdk.vitaljacket.com/>

### 1.2.3 MOSQUITTO<sup>12</sup>



#### Documentation

##### Man pages

- [mosquitto\(8\)](#) – the broker
- [mosquitto.conf\(5\)](#) – broker configuration
- [mosquitto\\_passwd\(1\)](#) – tool for managing mosquitto password files
- [mosquitto\\_tls\(7\)](#) – very rough cheat sheet for helping with SSL/TLS
- [mosquitto\\_pub\(1\)](#) – command line client for publishing
- [mosquitto\\_sub\(1\)](#) – command line client for subscribing
- [mqtt\(7\)](#) – background on the principles of mqtt
- [libmosquitto\(3\)](#) – client library programming

##### Programming information

- [libmosquitto API documentation](#)
- [Python module](#)

##### Other

See also the [ChangeLog](#).

There is now an [MQTT Wiki](#) available.

Figure 6 - Mosquitto logo and manual page

Mosquitto, an [iot.eclipse.org](http://iot.eclipse.org) project, is an open source message broker that implements the MQTT communication protocol, this protocol will be explained deeper into this report. It's important to notice, in figure 6, that mosquitto comes with command clients that allow to publish and subscribe to messages in order to test the connection with the broker.

<sup>12</sup> <http://mosquitto.org/>

## 1.2.4 VIRTUALBOX<sup>13</sup>, LINUX<sup>14</sup> AND WIRESHARK<sup>15</sup>

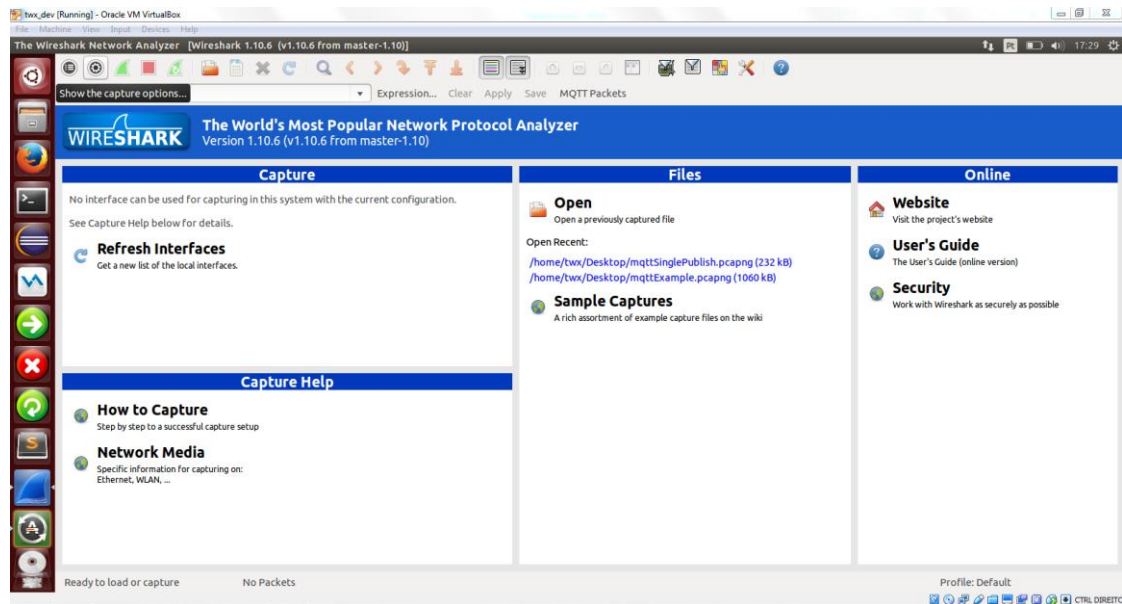


Figure 7 - Wireshark running on Linux Ubuntu on a Virtual Machine

These three known tools helped on the development of the project. VirtualBox and Linux provided a good environment to develop and test the application in a local machine, without the need of a deployment right in the beginning of the project. Wireshark permitted the capture of packets in order to study the different communication protocols and compare fields such as security, speed and size of each transmitted packet. On figure 7, the development environment can be observed, is important to notice that the version of Linux used is Ubuntu, this was a requisite of the ThingWorx platform.

## 1.2.5 SVN<sup>16</sup>



Figure 8 - SVN logo

SVN, also known as Apache Subversion, it's a version control software, just like git. This software required some study beforehand, as there was no experience with it prior to the development of this project. It has much similarities with git, which made it easy to learn.

<sup>13</sup> <https://www.virtualbox.org/>

<sup>14</sup> <http://www.ubuntu.com/>

<sup>15</sup> <https://www.wireshark.org/>

<sup>16</sup> <https://subversion.apache.org/>

### 1.2.6 IFTTT<sup>17</sup>

IFTTT, which stands for If This Then That, is a web based service that joins other popular web based services such as Facebook, Twitter and hardware IoT devices such as the Philips Hue.

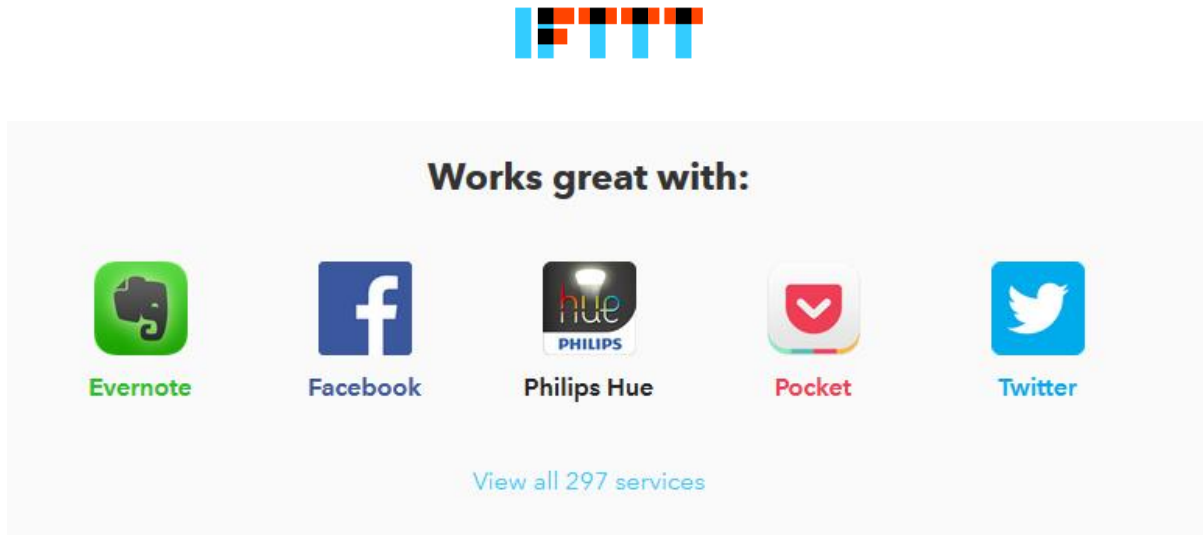


Figure 9 - Some IFTTT available services and logo

Some of the main available services that IFTTT offers are shown in figure 9. IFTTT is a great service to demonstrate and connect an application with other popular applications that are already developed. The objective of implementing this project solution with IFTTT is to show the possibilities of an IoT application, connecting to other services. IFTTT allows the user to connect the application developed in this project to a large number of popular applications, extending the use of our program.

---

<sup>17</sup> <https://ifttt.com>

## 1.3 The Organization

Celfocus is a joint venture of Vodafone Portugal, a fully owned subsidiary of Group Vodafone, and Novabase, the largest Portuguese information technology company listed on the Euronext Lisbon Stock Exchange. It was founded in 2000 and it combines the wireless telecommunications business know-how of Vodafone with the extensive IT expertise of Novabase. Serving a global client base in 25+ countries, Celfocus delivers high tech transactional, consulting and technology, systems integration and managed services. (Celfocus, s.d.)



Figure 10 - Celfocus offices around the world (Celfocus, 2016)<sup>18</sup>

As it can be seen in figure 10, Celfocus has various offices outside of Portugal, having two in United Kingdom, one in Istanbul and one in Dubai. In Portugal Celfocus has offices in Gaia (where this project was developed) and two offices in Lisbon. Besides having these offices, Celfocus has projects spread around the world, expanding lately to Africa and Middle East.

---

<sup>18</sup> This image is available through Celfocus Confluence, which is private and only accessible to Celfocus employees.

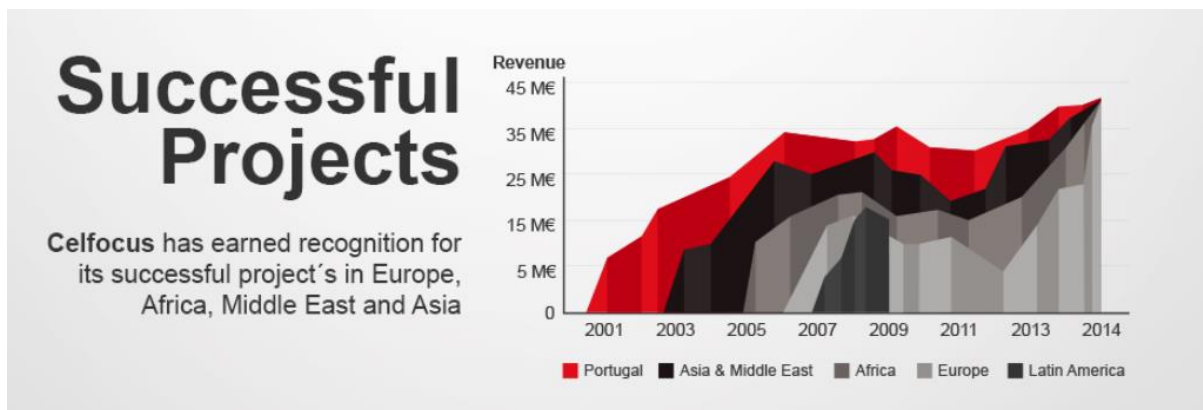


Figure 11 - Celfocus successful projects (Celfocus, 2016)<sup>19</sup>

Figure 11 shows the evolution of the amount of revenue coming from successful projects since Celfocus started, in 2000, and where the revenue comes from. As it can be concluded from figure 11, Celfocus has very successful projects across Europe, which makes it a very successful IT company.



Figure 12 - Celfocus Business Areas

<sup>19</sup> This image is available through Celfocus Confluence, which is private and only accessible to Celfocus employees.



On figure 12 the Business units of Celfocus are listed, each of these business units have their own developer teams and projects. A Head-Of is responsible for each unit. A quick description of each area is listed below:

- **Online Services**  
Takes care of two kinds of projects, Web and Mobile, providing online, retail and mobile solutions as well as developing mobile applications that facilitate sales and marketing.
- **Managed Services**  
Ensures all processes from demand to delivery while maintaining a common quality level. It provide specialized application maintenance and operation services to all of Celfocus' domains.
- **Customer Service**  
Aims to provide customers with solutions that enhance interactions with their final customers, providing better customer experience, better service and happier customers.
- **IoT**  
Helps to deploy the M2M/IoT platform, covering all the horizontal (cross market segment) functional needs. The specific needs of M2M/IoT also helps to develop new capabilities on new technologies & trends. This business area is where this project was inserted.
- **OM & Integration**  
Helps customers, improving their product lifecycle and customer service. Integrates all the company's information in only one solution, allowing an easy access and reduced costs.
- **EPM**  
Includes 3 sub units: Billing, Sales Performance Management and Business Intelligence & Analytics. Optimizes the sales processes and aligns the company's global and strategic objectives with the sales incentives, providing a solution to manage sales incentives and commissions of all sales channels, direct and indirect.
- **Digital TV**  
Provides Solutions and Services targeted to extend Telco 3Play TV Service
- **Next Generation Services**  
Helps telecom operators put their unified communications strategy into practice, focusing on delivering voice solutions that truly provide users with a great experience and are easy for telecom operators to maintain and evolve.

## 1.4 Document Structure

In this section the organization of the document will be explained, giving an improved view of the document to the reader. A structured of chapters is used, having, each chapter, sections and subsections, in the beginning of each chapter is located an index, giving the reader a quick resume and enumerating the different sections by page, and a small introductory text. The document contains 4 chapters, Introduction, Problem Background, Workflow and Conclusion.

- **Chapter 1 - Introduction** - This chapter gives a brief introduction of the whole project and introduces the main terms and ideas used, it's a good chapter for the reader to understand the background where the project was integrated in and to have the big picture of the solution implemented, introducing the project and selected technologies.
- **Chapter 2 - Problem Background** - This chapter explains the context of the problem. The aim of this chapter is to introduce the reader to the problem by describing the context in which it fits, the state of art and some possible solutions, diving further into the solutions available and at the same time, explaining some of the technologies that ended up in the selected solution.
- **Chapter 3 - Workflow** - This chapter explains the solution that was selected for the given problem. All the design and development of the solution is described in this chapter, turning it more into a technical chapter, where developers can quickly understand the choices that were made into the development of the solution. The working methodology is also described in this chapter, letting the reader know not only what was developed, but also how it was developed, covering the type of methodology used and some good practices that were followed in the project.
- **Chapter 4 - Conclusion** - This chapter finishes the document, giving a conclusion in the point of view of the developer, is the cherry on top of the cake, giving the reader the personal opinion of the developer on the project, how the development carried out, how the developer thinks the development will carry out in the future and what the developer would change if the project was his responsibility only.

## CHAPTER 2 PROBLEM BACKGROUND

Every problem has a background, and understanding that background may prove to be a very good practice. By learning extra information about the problem, the developer may explore new solutions, which would otherwise not be available to him, and gain a better acknowledgement of the problem itself.

<b>Chapter 2</b>	<b><i>Problem Background</i></b>	<b>17</b>
<b>2.1</b>	<b>Business areas</b>	<b>18</b>
2.1.1	Healthcare	18
2.1.2	Technology	18
<b>2.2</b>	<b>State of Art</b>	<b>19</b>
2.2.1	IoT Platforms	19
2.2.2	Communication	26
2.2.3	Encryption	29
2.2.4	Possible Solutions	32

## 2.1 Business areas

The project fits in a couple of business areas which are difficult to cope together, as they are very delicate areas which require a lot of confidentiality and security.

### 2.1.1 HEALTHCARE

The maintenance of human health via diagnosis, treatment and prevention of physical and mental diseases is what describes the healthcare area of business. It's provided by health professionals such as medics, pharmacist and dentists. In the case of this project, a medic receives information about the hearth rate of the patient that is using the Vital Jacket. This information has to be strictly confidential and can't be shared with anyone else, besides the medic. For this the use of HL7, a pattern used for the exchange of medical data should be studied and applied. (Medical Dictionary - Healthcare, s.d.)

### 2.1.2 TECHNOLOGY

Used in the accomplishment of objectives such as scientific investigation, technology, is a body of knowledge devoted to creating tools, processing actions and extracting of materials. The technology used in this project was the collection of hearth rate information using a simple T-shirt. Technology improves the life of the human being, but can also be used for evil, such as weapons of mass destruction. In the technology used in this project, related to informatics, corruptive behavior can also be noticed. The system can be target of attacks which can lead to several cybercrimes such as identity theft or information warfare, avoiding this is one of the main objectives of this project, focusing on informatics security. (What is technology, 2013)

## 2.2 State of Art

There is only one SDK that can be used, as it is the only SDK that can communicate with Vital Jacket, developed by Biodevices, but there is a big number of IoT platforms, communication protocols and encryption algorithms that can be used to store the information and share it with other applications. In this project the IoT platform used was ThingWorx, as there were more projects connected to that platform and the choice of platform wasn't flexible at all. But some research was made in order to try to find a better alternative to ThingWorx. Also a very important part of the project was the communication/encryption used to exchange information between Vital Jacket and the IoT platform.

### 2.2.1 IOT PLATFORMS

Storing all the information available from various devices and being able to process all the data and share it with the necessary applications, the IoT platform is the middle piece of the project, it acts as a bridge that connects all the available devices with the supported applications.

Table 4 - IoT Platforms

IoT Platform	Pros	Cons
<b>ThingWorx</b>	<ul style="list-style-type: none"> <li>- Being already used in other projects</li> <li>- Drag and Drop functionality that allows rapid creation</li> <li>- Provides Marketplace that brings together IoT apps</li> </ul>	<ul style="list-style-type: none"> <li>- Has poor data analysis tools</li> <li>- Strict design tools</li> </ul>
<b>IBM Bluemix</b>	<ul style="list-style-type: none"> <li>- Improved data analysis tools</li> <li>- Analysis unstructured data</li> <li>- Artificial Intelligent (uses IBM Watson)</li> </ul>	<ul style="list-style-type: none"> <li>- Not design driven (poor tools for showing data to the user)</li> </ul>
<b>XMPP</b>	<ul style="list-style-type: none"> <li>- Free and open source</li> <li>- Software for every platform</li> </ul>	<ul style="list-style-type: none"> <li>- Text-based communication</li> <li>- Does not support end-to-end encryption</li> </ul>

In table 4 some of the principal IoT platforms are listed with the respective pros and cons. Two of the best IoT platforms are ThingWorx and IBM Watson. ThingWorx fits very well in this project, because it's already developed inside Celfocus in other projects, which makes easier to learn, since there is constant support from Celfocus if a problem comes up. Another very good IoT platform is IBM Bluemix because of the strong capability of analyzing data with AI tools, such as IBM Watson, a point where ThingWorx it's not so good at. XMPP it's also a good alternative because it's an open source solution and gives huge freedom for other developers to expand the system, although this solution seems more dangerous to use due to said freedom. In the following sub chapters a quick description of these three technologies can be found and also a more in-depth explanation of ThingWorx, as it was the chosen platform for this project.

#### 2.2.1.1 *ThingWorx*<sup>20</sup>

ThingWorx is already used in other projects at Celfocus, which gives the ability of building a bigger IoT network since the platform is already populated with information that could also be useful to Vital Jacket. Its marketplace is also a great favoring point, giving access to already built tools that can help in the development of the project, such as extensions for communication protocols or other technologies such as Twitter or Email.

As an IoT platform, ThingWorx centralizes the information of various devices, making possible to develop a web portal or smartphone application that shows the information after being analyzed. ThingWorx also has a REST API available, which allows other developers to make applications that use the information stored on ThingWorx, by making requests to this API.

---

<sup>20</sup> <https://www.thingworx.com/>

## ThingWorx Composer

ThingWorx offers a graphical interface for the developer to create and change all the aspects of his application, this graphical interface is called composer, and it's accessible through a web browser.

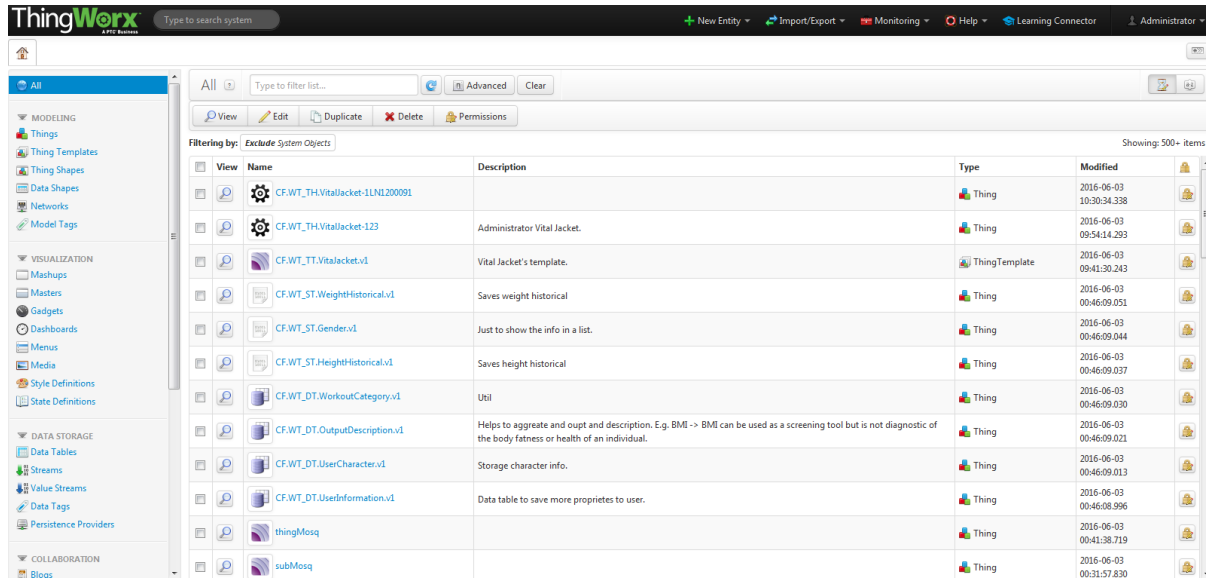


Figure 13 - ThingWorx Composer main screen

The main screen of the composer, where all the entities created can be seen, it's shown in figure 13. This is the main way a developer interacts with the ThingWorx platform, everything explained next is developed in this graphical interface.

## Structure of the information

Managing a lot of devices and information, the amount of data stored in ThingWorx can take a very large size, but the amount of space used is one of the smaller problems, a bigger problem is the organization of all the information. Defined structures are implemented in order to simplify all the information, allowing for an easier development of an application. ThingWorx achieves this by using polymorphism, mainly organizing information by Things, Things Templates and Things Shapes.

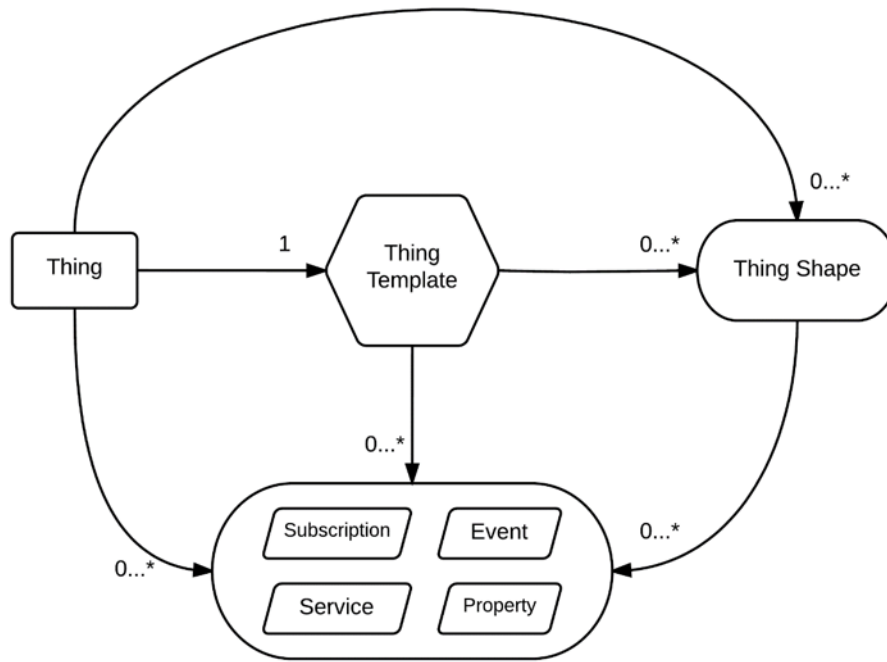


Figure 14 - ThingWorx information structure<sup>21</sup>

In figure 14, the relationship between the three main structures is described. These three structures are defined in the following way:

- **Thing**

A Thing is the implementation of each device, in the case of this project, each Vital Jacket is a Thing. Every Thing has to have a Thing Template, which will have all the properties of the Thing. Things aren't created by the developer, the developer only creates the Thing Template, and develops a way for the User to create a Thing dynamically, on the case of this project, each time a user adds a Vital Jacket into his account, a new Thing representing that Vital Jacket is created, each one of these Things implements the developed Thing Template. Relating to Object Oriented Programming (OOP), Things are the initialized objects.

- **Thing Template**

A Thing Template describes a Thing, it contains all the properties needed and is what the developer spends most time developing. It models a set of similar objects, relating to OOP, Thing Templates are abstract classes. On ThingWorx almost anything ends up being a Thing, a Thing Template is also a Thing, so it needs to implement a Template, in this case, a Thing Template implements the template "GenericThing" (Similar to the Object class, which every class has as a superclass in java).

<sup>21</sup> Figure provided by Celfocus employee.



- **Thing Shape**

A Thing Shape contains properties that are going to be stored in the Thing Template, it can also be implemented by a Thing, as can be seen in figure 3, but this practice is almost never used, since normally the properties are implemented by a group of Things and not by just one (it's like saying that you only capture the user's hearth rate in one Vital Jacket and all the other ones don't capture the hearth rate, it doesn't make sense). Thing Shapes can be very similar to Thing Templates and are mostly used when the developer wants to add functionality across different Thing Templates that can't share a base Thing Template.

All these three structures contain Properties, Services, Events and Subscriptions. Properties and Services are the most important of the four, and can be defined in the following way:

- **Properties**

Properties are the variables of the Thing, they can contain hard coded values or can be bound to a remote thing. In the case of this project, some of the Vital Jacket properties are Temperature, Heartrate and the three Accelerometer axis.

Figure 15 - ThingWorx Property creation

In figure 15 the attributes that a property contains can be observed, these attributes make properties easier to understand and develop, the developer can give a description to each property, add various alerts, which are fired when a set condition happens to the property and can even change some key aspects such as persistency or read-only. The different values of the property can be stored on ThingWorx using the Logged aspect, allowing for a storage of the value history.

- **Services**

Services are the functions of ThingWorx, they are developed in JavaScript and are the main way of interaction between structures.

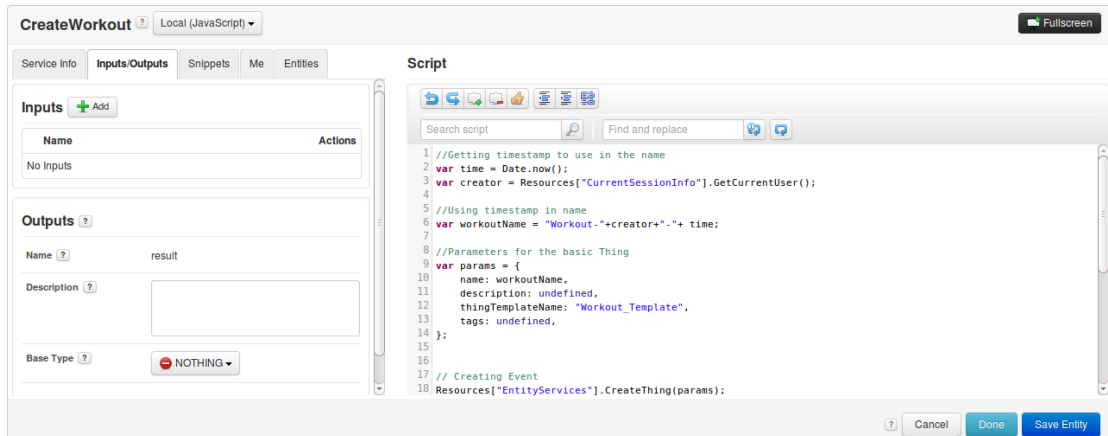


Figure 16 - ThingWorx Service creation

Figure 16 shows the creation of a service, in this case the service shown is the service that creates a workout (which is a Thing). Services can have inputs and outputs, just like any method, and are the main way of interacting between structures or inside the same structure, some examples of what a service can do are: Create other Things, Interact with outside applications, Calculate property average values.

## Data storage

As well as structuring information, ThingWorx also stores data, using defined structures created on ThingWorx composer. There are various ways of storing data, some of the main ways are:

- **Data Tables**

These are the closest to a relational database table, the structure of data stored in a Data Table is defined by a Data Shape and it must contain a primary key.

- **Streams and Value Streams**

Streams and Value Streams both provide time series information about a Thing's property, they serve for properties that you want to keep record of, to create a chart for example. The biggest difference between Streams and Value Streams are the way the data is written and returned. Streams can access data continuously (querying a single column value returns the entire row). Value Streams are used to store data from an associated Thing's property (querying the thing's property data returns the values for that property only).

All these storages use persistence providers, which basically connect ThingWorx to a specific database, on this project Value Streams were used with just one persistence provider to the whole project, meaning that all the information is stored in one database.

### Data Access

The developed applications must have ways to access the previously stored information. ThingWorx provides a REST API that the developer can use in order to access information or call services. ThingWorx REST API can be accessed either by web browser or by a developed REST API request, for the later one, an application key must be created previously on ThingWorx Composer. If the application is developed inside of the ThingWorx composer using mashups (webpages developed inside of ThingWorx) the application can access Things' properties directly, without needing to use REST API.

Another way of accessing and changing data inside ThingWorx is by using communication protocols that are supported by ThingWorx, some of these protocols are described later in this chapter.

#### 2.2.1.2 IBM Bluemix<sup>22</sup>

IBM Bluemix is another very famous IoT platform (quora, s.d.), the biggest aspect of Bluemix is the analysis of data, the extraction of key information from documents and the revelation of patterns in data, using IBM Watson, which provides a huge help in artificial intelligence and pattern revelation, surpassing the intelligence of ThingWorx. Bluemix can find patterns in information and learn with these patterns, providing a huge help to the user after a while, since the information is all learned by the platform and can be used to answer user questions or problems. As Bluemix surpasses ThingWorx at intelligence and data analysis, ThingWorx surpasses Bluemix at showing the information to the user. Bluemix it's not design driven and doesn't have the tools that ThingWorx offers to create mashups (web pages) that are easy for the user to navigate. (IBM Watson, s.d.)

The best solution possible, in terms of IoT platforms, is to join ThingWorx and Bluemix, giving the responsibility of analyzing data to Bluemix before storing the analyzed data and showing to the user, using ThingWorx. These two platforms joined would create a "super" IoT platform, providing data, which the platform learns on its own, on a fluid and engaging design.

---

<sup>22</sup> <http://www.ibm.com/cloud-computing/bluemix>

### 2.2.1.3 XMPP<sup>23</sup>

XMPP is a communication protocol based on XML, as it is an open standard, there are lots of software and libraries, either free or paid. There are also widely known projects that use XMPP. Being a proven technology, secure and efficient, XMPP is ideal for an IoT project since It provides great M2M connection. (XMPP, s.d.)

Users	Company	Use	Description
~1.5 billion	Google	Push Notifications	Google provides an <a href="#">XMPP Interface</a> to their push notification service. It's also been rumoured that push notifications are delivered to the device via a proprietary binary XMPP protocol.
~800 million	WhatsApp	Chat	WhatsApp uses a <a href="#">variation of XMPP</a> for its popular chat service
~500 million	Apple	Push Notifications	Apple uses <a href="#">XMPP to deliver push notifications</a> to client devices.
~200 million	Nimbuzz	Instant Messaging	Nimbuzz is an XMPP-based instant messaging community.

Figure 17 - Main projects that use XMPP (XMPP, s.d.)

Figure 17 shows the most known projects that use XMPP. The large variety of available software gives flexibility in the choice of language to develop and OS used by the server. XMPP can also give the ability of building an expandable system, allowing for future developers to run their own servers/clients. It provides a more open solution to the problem in case and it provides real time transmission of data.

## 2.2.2 COMMUNICATION

The information must be shared in a secure and fast way, in order to achieve this, a good communication protocol must be used. Besides the speed and security of the communication protocol, Vital Jacket specs must also be taken into account, IoT devices usually have low specs and must have specific protocols that don't take too much processing power. With this in mind, two communication protocols came up in the development of this project, MQTT and CoAP. These two protocols were chosen due to being lightweight and fast, being ideal in an IoT situation, the use of other communication protocols such as HTTP would not be efficient in this context.

<sup>23</sup> <http://www.xmpp-iot.org/>

### 2.2.2.1 CoAP<sup>24</sup>

CoAP is intended to be used in an IoT situation, since it has very low costs. It uses UDP, which can be bad because of the eventual loss of packets, but this can be turned around by using DTLS, which gives extra security to UDP, preventing eavesdropping, tampering, message forgery, dealing with packet reordering and loss of datagrams and data larger than the size of the datagram packet.

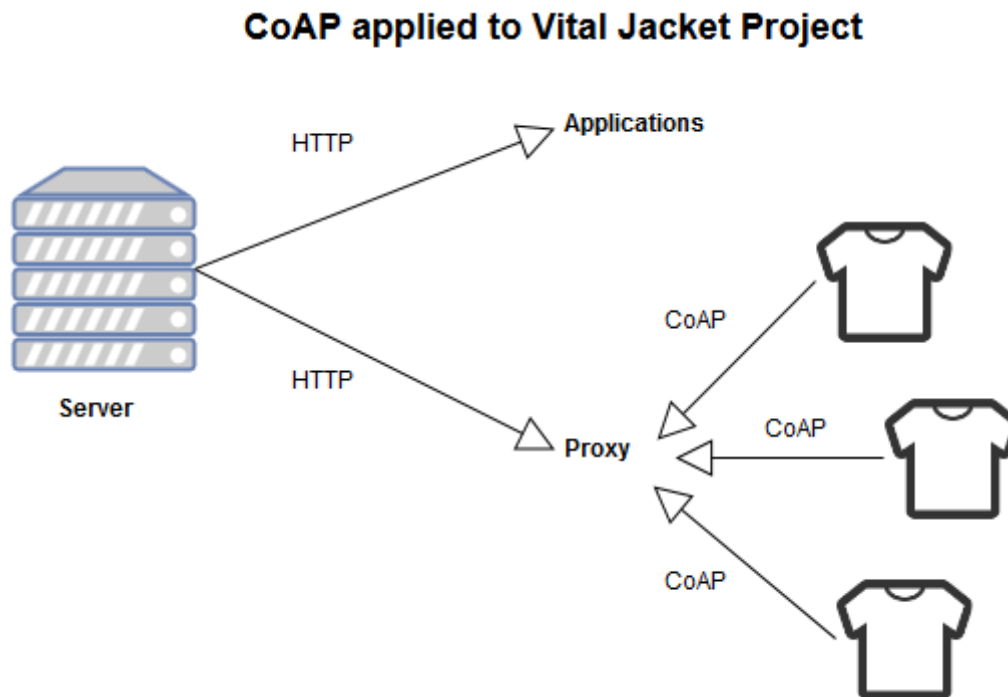


Figure 18 - Diagram illustrating CoAP implementation

A great advantage of CoAP is the ability to easily translate to HTTP. As it can be seen in figure 18, a proxy could be implemented, that communicates with every T-Shirt by CoAP, and moves that information to the main server by HTTP. (Schneider, 2013) This facilitates the implementation of future applications, since the developers will be able to get all the information by HTTP, which is a far more used protocol. This implementation also has the advantage of reducing traffic, since the data from all the Vital Jackets is first gathered in the proxy server and then send all together to the main server by a single connection, instead of a single Vital Jacket Data at a time. (Jaffey, 2014)

<sup>24</sup> <http://coap.technology/>

### 2.2.2.2 MQTT<sup>25</sup>

Contrary to CoAP, MQTT uses TCP/IP, inheriting all its benefits. It uses a publish-subscribe model in which clients are connected to a server, publishing and subscribing to topics. These topics are located inside a server and are managed by a broker, clients can publish messages to the topics and can also subscribe to multiple topics, receiving every message that is published to these topics. The next image describes a possible application of the MQTT protocol to this project, being the Vital Jackets and any applications developed the clients. The main route of the information would be Vital Jacket -> Server -> Applications, but information could also flow the other way around (Applications -> Server -> Vital Jacket). In order for the server to managed the mqtt topics a mqtt broker has to be installed on the server.

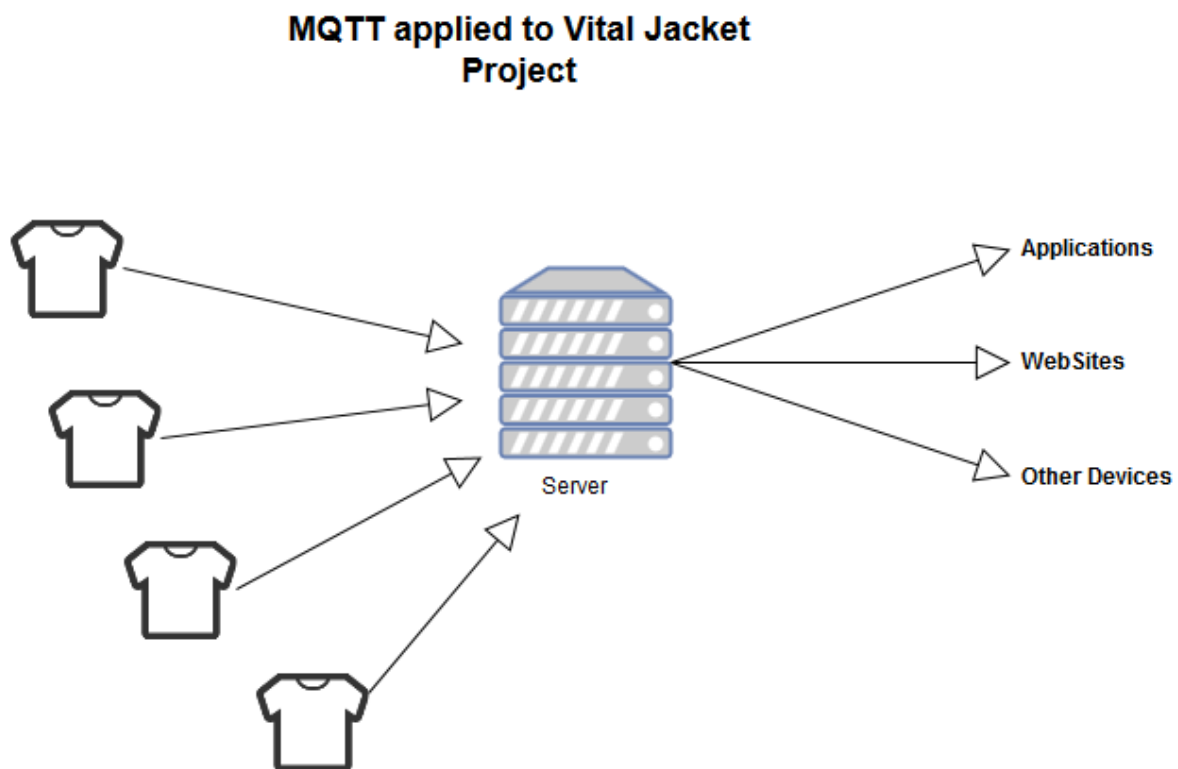


Figure 19 - Diagram illustrating MQTT implementation

Figure 19 illustrates a possible MQTT implementation in this project. The different information gathered by the Vital Jacket would be spread in different topics, such as Accelerometer, ECG or Temperature, each one would be one topic, the applications would then subscribe to the topics that they wanted, receiving every update of these topics, this permits for a selection of information, which can reduce the amount of traffic, for example, if an application just wants to know the accelerometer data, it doesn't have to receive all the other data such as ECG or temperature (Jaffey, 2014). The MQTT

<sup>25</sup> <http://mqtt.org/>

broker has a very high importance since it manages all the topics, connecting the messages that are published to their respective subscribers.

### 2.2.2.3 AlwaysOn

AlwaysOn is the communication protocol frequently used by ThingWorx, numerous SDKs are available as ThingWorx extensions that cover a vast number of programming languages, allowing the development of an AlwaysOn connected Thing on languages such as C, C# or Java. It provides an encrypted always open and full-duplex web-socket connection between ThingWorx and the remote application, using the standard ports for HTTP and HTTPS (80 and 443) to maintain connectivity.

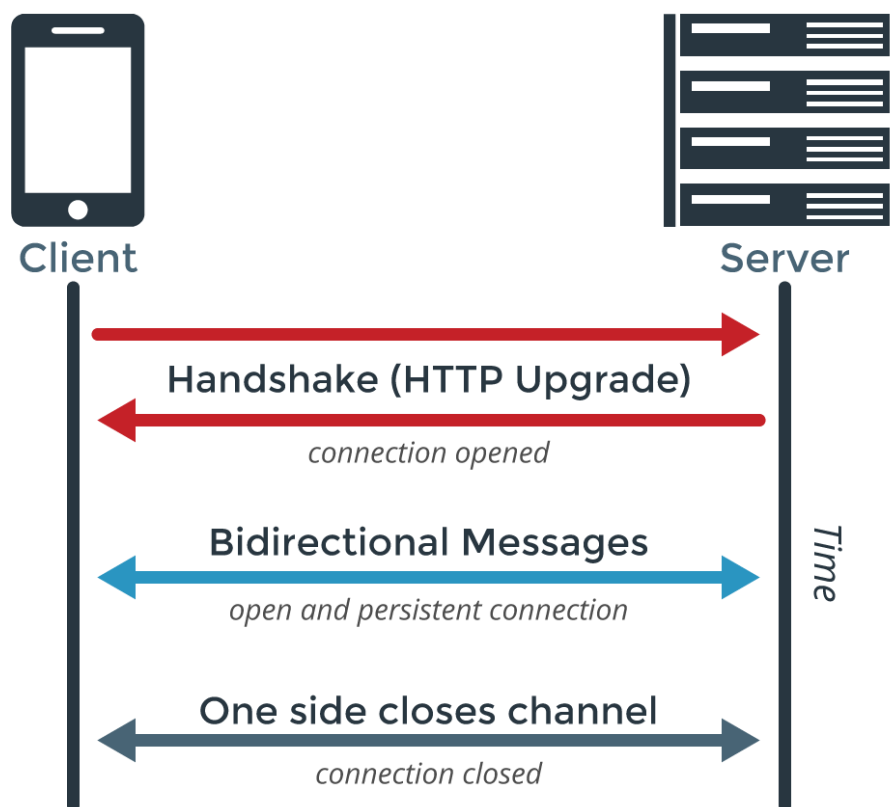


Figure 20 – WebSockets (WebSockets vs REST: Understanding the Difference, 2015)

As mentioned above, AlwaysOn is based on WebSockets, providing a full duplex connection, as can be seen in figure 20. There is a handshake before messages are sent and it just takes one side to close the channel in order for the connection to close.

### 2.2.3 ENCRYPTION

Because of the information sensitivity, using a strong encryption algorithm is a main concern. The application can be target of hackers and in case the information gets stolen it must have a good encryption in order to be unusable by the attackers.

There are two main types of encryption: **Symmetric key encryption** and **Asymmetric key encryption**.

**Symmetric key encryption** uses the same key to encrypt and decrypt data, this means that the key must be exchanged in a secure way, which gives a chicken and egg situation, there must be an exchange of keys in a private way to the encryption to be possible, but since there is no encryption yet, there isn't a secure way to exchange those keys. The main way to deal with this problem is to use an asymmetric key encryption to exchange the symmetric key. After the exchange occurs the system can use the symmetric keys to encrypt and decrypt the data, making it possible to exchange the data in a non-secure communication, due to the encryption.

**Asymmetric key encryption** uses two pairs of keys, a private key which can't be shared with anyone and a public key that everyone can see, the data encrypted by an entity public key can only be decrypted by that entity private key. So if anyone encrypts data with an entity public key, which anyone can see, only that entity can decrypt that data, using its private key. A private key can also be used for digital signature, if an entity encrypts a message with its private key, anyone can decrypt it with the public key, and everyone can be sure that that message was sent by that entity, since only that entity has the private key that encrypted the message.

Table 5 - Encryption algorithms

Encryption Algorithm	Type	Pros	Cons
Advanced Encryption Standard (AES)	Symmetric key encryption	<ul style="list-style-type: none"> <li>- Fast in both software and hardware.</li> <li>- High speed and low RAM requirements.</li> </ul>	<ul style="list-style-type: none"> <li>- Block cipher which can mean having to split data to encrypt it.</li> </ul>
RSA	Public key encryption	<ul style="list-style-type: none"> <li>- Most secure key system.</li> </ul>	<ul style="list-style-type: none"> <li>- Slow algorithm.</li> <li>- Complexity of key creation.</li> </ul>
Diffie-Hellman	Public key exchange	<ul style="list-style-type: none"> <li>- Well suited for use in data communication.</li> </ul>	<ul style="list-style-type: none"> <li>- Does not provide authentication.</li> <li>- Vulnerable to man-in-the-middle attacks.</li> </ul>

In Table 5 some of the main encryption algorithms can be observed as well as the respective pros and cons and the type of encryption used in each algorithm.



### 2.2.3.1 AES

The first algorithm that was studied was AES, which stands for Advanced Encryption Standard, also known as Rijndael, which is a play on the names of its two inventors, Vincent Rijmen and Joan Daemen.

- Symmetric key algorithm
- Block cipher
- Uses key sizes of 128, 192 or 256 bits and 10, 12 or 14 rounds
- Performs on a widely variety of hardware
- Works on 128 bit block sizes.

AES uses symmetric keys, meaning that either the encryption as the decryption uses the same key, the problem lies in sharing this key, since there isn't a safe method of communication yet, the sharing of the symmetric key is usually done by RSA that will be described further. It uses key sizes of 128, 192 or 256 bits and block sizes of 128 bits, this means that data has to be split into segments with 128 bits before encrypting it, since it's a block cipher. It encrypts the data by rounds, each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key. (Rouse, Advanced Encryption Standard (AES), 2014)

### 2.2.3.2 RSA

The second algorithm studied was RSA, an algorithm first publicly described in 1973. The letters come from the initials of the RSA designers' surnames, Ron Rivest, Adi Shamir, and Leonard Adleman.

- Asymmetric key algorithm
- 1024 to 4096 bits key sizes
- Slow algorithm
- Used mainly for exchange of symmetric keys

Encryption algorithm that uses asymmetric keys. It uses key sizes of 1024 to 4096 bits and it isn't a block cypher, on contrary of AES. Because of its slowness it's not normally used for data encryption, instead it's used for encryption of a symmetric key, for an algorithm like AES, which is much faster and can encrypt bulk data in a much faster way. (Rouse, RSA algorithm (Rivest-Shamir-Adleman), 2014)

### 2.2.3.3 Diffie-Hellman

The last algorithm studied was Diffie-Hellman, published in 1976 by Whitfield Diffie and Martin Hellman, therefore the name "Diffie-Hellman". In 2002, Hellman suggested the algorithm be called Diffie-Hellman-Merkle, since Ralph Merkle contributed to the invention of the algorithm.

- Asymmetric key algorithm
- Does not provide authentication
- Vulnerable to man-in-the-middle attacks

A way of, securely, exchanging keys, being the first public-key protocols. It uses numbers raised to specific powers to produce encryption keys. The most serious limitation is the lack of authentication, making it vulnerable to a man in the middle attack. Implementing an authentication method such as digital signatures with Diffie-Hellman protocol would make it safer, some protocols, like PGP, do this. This protocol it's well suited for data communication but it's not often used for data stored over long periods of time. (Diffie-Hellman key exchange (exponential key exchange), 2007)

#### 2.2.4 POSSIBLE SOLUTIONS

At the time this project was developed the Vital Jacket only connected by Bluetooth, a Wi-Fi model was being built but could not be used in this project, because of this, the solution for the project was simplified a bit, sending the data directly to the computer which had the Vital Jacket SDK and ThingWorx platform. The ideal solution was sending the data, via Wi-Fi or an Internet connected smartphone, directly to ThingWorx Platform.

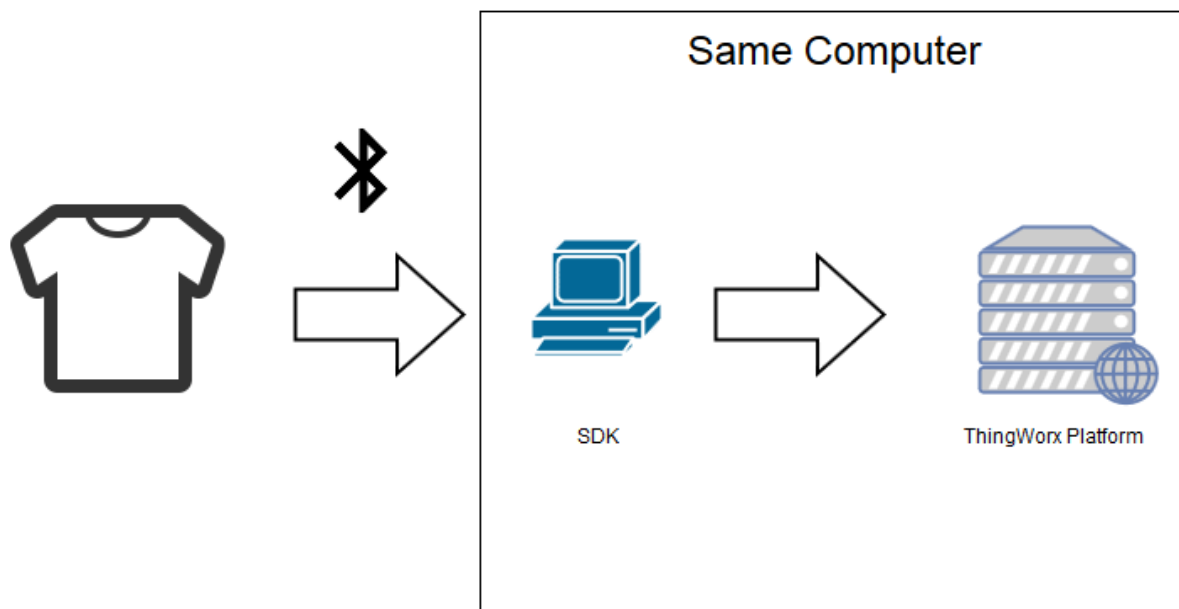


Figure 21 - Actual solution

In figure 21 a simplified solution can be seen, this solution was the one opted by, since the Vital Jacket could only be connected by Bluetooth and the time had to develop the solution was limited. This solution was opted in order to achieve a product capable of demonstration, if done correctly, this system would easily be translated into one of the two systems shown below.

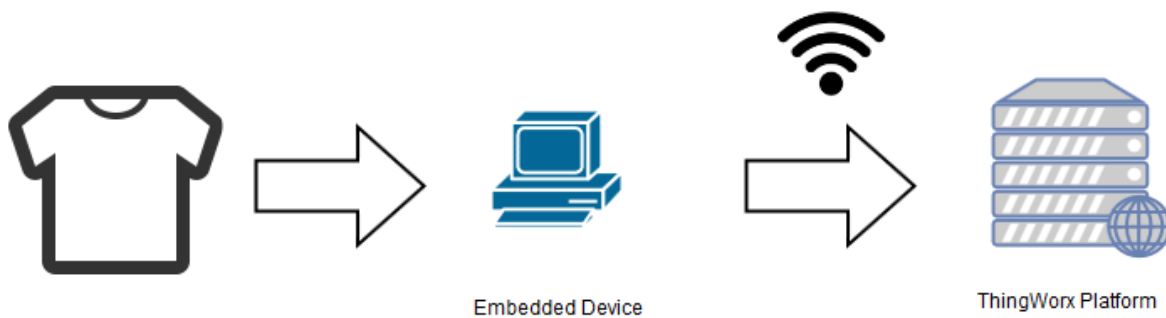


Figure 22 - Ideal Solution

In figure 22 an ideal solution can be seen, the Vital Jacket had to be able to send data through Wi-Fi, and it would send the data directly to the ThingWorx platform. The user would need a good internet connection for the data to be sent in real time, which could be a difficult task. Due to the inexistence of Wi-Fi on Vital Jacket, this solution couldn't be developed yet.

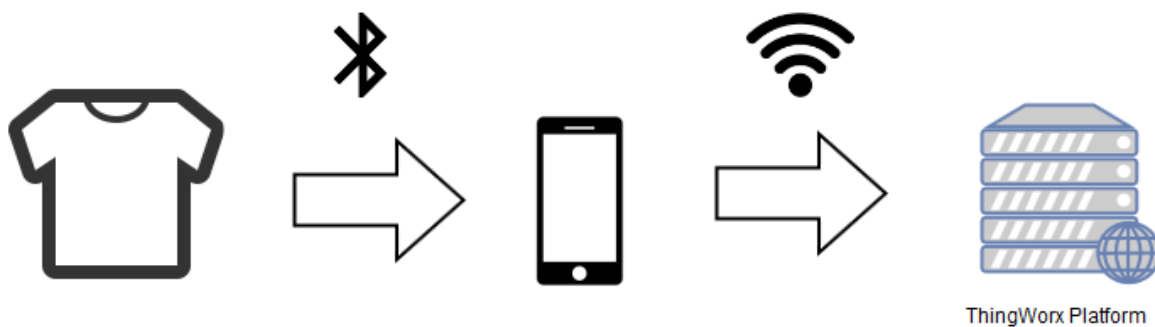


Figure 23 - Another Ideal Solution

Figure 23 represents another ideal solution, this one could be developed at the moment, since it uses the Bluetooth of the Vital Jacket to send data to a smartphone. This smartphone would use its connection with the internet, by either Wi-Fi or Mobile Data, to send the data to the ThingWorx platform. This solution would require the user to have an internet connected smartphone with the Android OS and would be more complex to develop since the need to develop an android application. It could be possible to implement this solution on Android since a Vital Jacket Android SDK is available.

The biggest con of this solution is the two points of vulnerability that the system would have, two connections have to be made, one by Bluetooth and another one by Wi-Fi or mobile data, this means that a potential hacker could attack the system in both these connections, having more possibility of stealing data.

Table 6 - Pros and Cons of the possible solutions

Solution	Pros	Cons
<b>SDK and ThingWorx on same computer (fig.20)</b>	<ul style="list-style-type: none"> <li>• Easy to implement</li> <li>• Good for Proof of Concept</li> </ul>	<ul style="list-style-type: none"> <li>• Doesn't relate to a real life scenario</li> <li>• Communication between SDK and ThingWorx is almost non-existent (located on the same machine)</li> </ul>
<b>Wi-Fi on Vital Jacket (fig.21)</b>	<ul style="list-style-type: none"> <li>• No third device needed</li> </ul>	<ul style="list-style-type: none"> <li>• Wi-Fi dependent</li> <li>• Encryption has to be made by the embedded device</li> </ul>
<b>Using Smartphone (fig.22)</b>	<ul style="list-style-type: none"> <li>• Smartphone App could provide extra features</li> <li>• Could use Mobile Data instead of Wi-Fi</li> </ul>	<ul style="list-style-type: none"> <li>• More complex solution</li> <li>• Internet connected smartphone needed</li> <li>• Two points of vulnerability</li> </ul>

## CHAPTER 3    WORKFLOW

A workflow is developed while a project is being developed, this workflow incorporates subjects such as methodology, planning, and technical description. In this chapter the workflow of the project is described, giving the reader the possibility of observing the work that was done and how it was done, describing the project in a more technical way, using a variety of diagrams.

<b>Chapter 3    Workflow</b>	<b>35</b>
<b>3.1    Working Methodology</b>	<b>36</b>
3.1.1    Waterfall Methodology	36
3.1.2    Confluence	37
3.1.3    Jira	37
3.1.4    SVN	38
<b>3.2    Technical Description</b>	<b>39</b>
3.2.1    Diagrams	39
3.2.2    Instalations	43
3.2.3    Solution Development	45
3.2.4    Analysis	58
3.2.5    Tests	60

## 3.1 Working Methodology

Every project has a related methodology, a way of organizing work in order for all the developers to cope together working in the same way. In this project a waterfall methodology has been used, splitting the project in five main tasks, as they can be seen in the work breakdown structure shown in page 4.

### 3.1.1 WATERFALL METHODOLOGY<sup>26</sup>

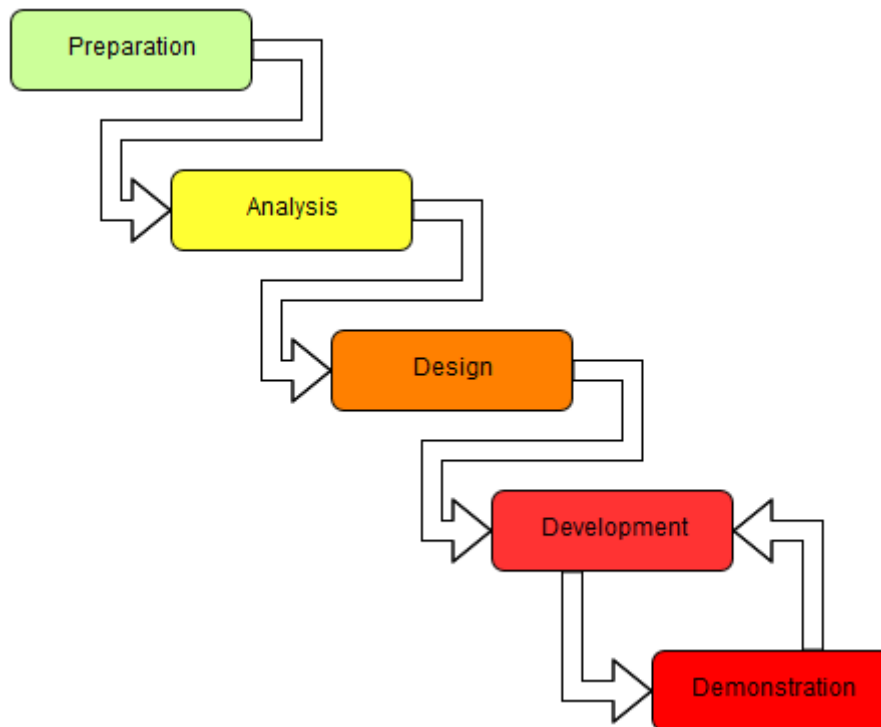


Figure 24 - Waterfall Diagram<sup>27</sup>

The waterfall methodology used in this project can be observed in figure 24. A task only begins when the previous one is closed, except at the bottom of the waterfall, since demonstrations were made during the development of the project, making a loop between development and demonstration. Development would continue after a demonstration but the design of the project wouldn't change.

<sup>26</sup> <http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/>

<sup>27</sup> Figure based on the waterfall methodology.

- **Preparation**
  - Scope of the project.
  - Introduction to the company.
- **Analysis**
  - Communication and encryption protocols.
  - Possible applications to integrate.
  - ThingWorx platform.
- **Design**
  - Concept diagram.
  - Domain model diagram.
  - ThingWorx diagrams.
  - Deployment and Component diagrams.
- **Development**
  - Implementation and analysis of MQTT.
  - Implementation of the Vital Jacket SDK.
  - Implementation of Vital Jacket with ThingWorx.
  - Implementation and analysis of AlwaysOn protocol.
- **Demonstration**
  - Periodic demonstrations.
  - Demonstration of the Vital Jacket with Bluetooth connection.
  - Demonstration of the Simulated Vital Jacket (without Bluetooth connection).
  - Demonstration of ThingWorx platform.

### 3.1.2 CONFLUENCE<sup>28</sup>

Confluence is an Atlassian collaboration tool that works like a wiki and allows an easy share of information inside the company. This tool is a great asset to have, since it provides knowledge to every developer. In this project a lot of contributions were add in Celfocus confluence and also a lot of information was taken from it, providing precious help.

### 3.1.3 JIRA<sup>29</sup>

Jira is an Atlassian software development tool that allows management of issues, facilitating teamwork, since every team developer can observe the progress of each other work and provide good

---

<sup>28</sup> <https://www.atlassian.com/software/confluence>

<sup>29</sup> <https://www.atlassian.com/software/jira>

feedback. Jira was used in this project to share work progress with the project developers and with Celfocus supervisor.

### 3.1.4 SVN<sup>30</sup>

Apache subversion, as mentioned in page 11, is a version control software. It was used in this project to merge the project between the two developers that worked in it. SVN use was more intensive until the deployment, as the project was on a virtual machine and there was a constant need to merge projects while developing. After the deployment, as the project was just in a remote machine, there was no need to merge projects and SVN was just used to store the project as a backup.

## iot\_twx - Revision 160: /WeraTshirt

- [..](#)
- [branches/](#)
- [tags/](#)
- [trunk/](#)

---

*Powered by [Apache Subversion](#) version 1.8.13 (r1667537).*

Figure 25 - SVN Repository

The project was stored in the repository shown on figure 25, this repository was divided into three folders.

- **Branches**  
Used to store different versions of the project.
- **Tags**  
Used to store tags that can identify different parts of the project.
- **Trunk**  
Used to store the current iteration of the project.

---

<sup>30</sup> <https://subversion.apache.org/>



Trunk is the most worked folder, as there was where the project was stored.

## iot\_twx - Revision 253: /WeraTshirt/trunk

- ..
- [Device/](#)
- [Extensions/](#)
- [Thingworx/](#)

---

*Powered by [Apache Subversion](#) version 1.8.13 (r1667537).*

Figure 26 - SVN Repository trunk folder

Inside the trunk folder, as can be observed in figure 26, the project was split into three folders.

- **Device**  
This folder contained all the scripts that simulated the Vital Jacket or made the Bluetooth connection with the Vital Jacket. It was the most worked folder in this project.
- **Extensions**  
This folder contained all the Thingworx extensions needed, the extensions were stored in this folder in order to be easy to install when needed (on the deployment for example).
- **Thingworx**  
This folder, as the name suggests, contained the Thingworx project.

## 3.2 Technical Description

After studying the possible solutions and technologies available, comes the application of these technologies, in order to create a final solution that works well and can be demonstrated. This sub chapter describes the technical part of the project, when the solution comes to life getting developed.

### 3.2.1 DIAGRAMS

One of the most important tools for a developer are a project diagrams, since when a developer continues the work of another developer, he has to quickly learn all about the work previously done. Diagrams allow for a quick analysis and understanding of a project, supposing they are done in the right way. Every diagram has to describe their task and be simple and easy to understand.

### 3.2.1.1 ThingWorx

The ThingWorx platform has specific terms, because of this, there was a need to create a diagram to describe the structure of the data implemented. These diagrams are similar to a Class Diagram.

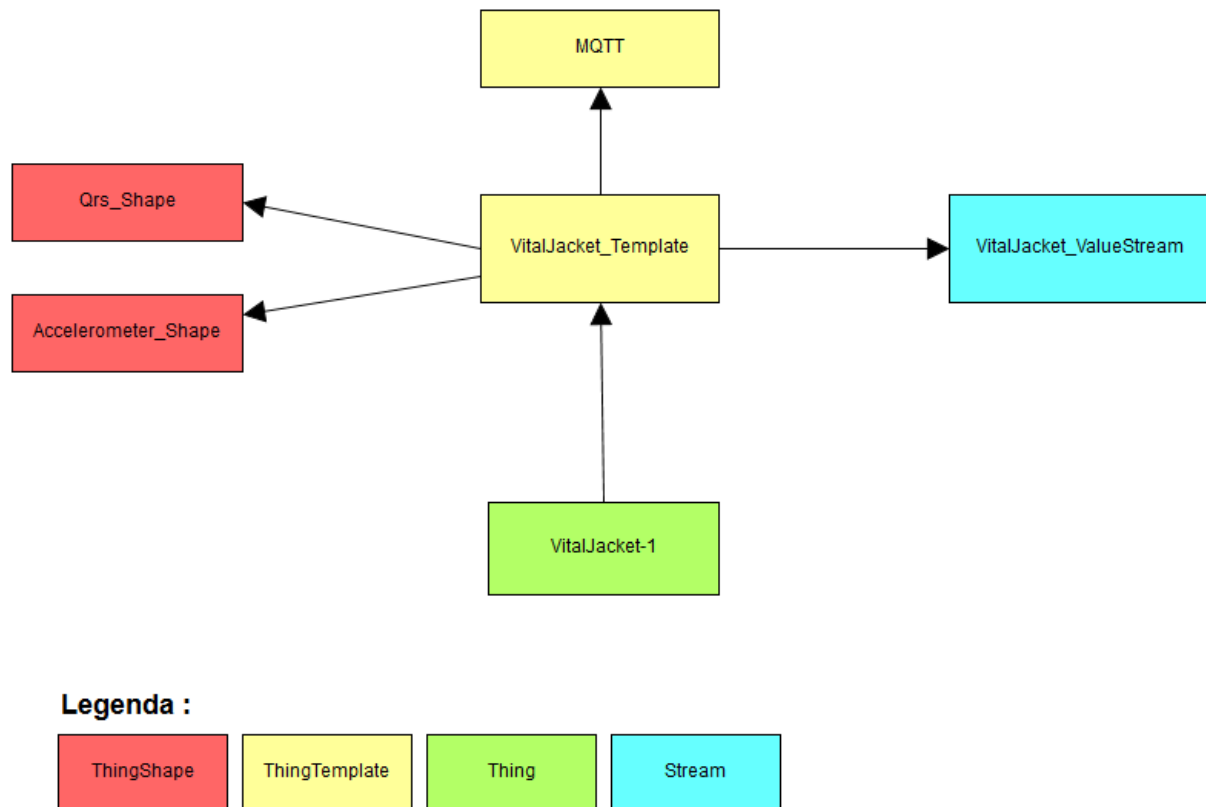


Figure 27 - ThingWorx Diagram - Vital Jacket

Figure 27 describes how the Vital Jackets are implemented on ThingWorx, each Vital Jacket is a Thing (in this diagram there only exists one Vital Jacket, named “VitalJacket-1”). This Thing implements a Template, which implements two shapes. These shapes contain all the variables from the Vital Jacket, if there is a need to insert more variables, a developer only needs to create a new shape and import it on the template. The Vital Jacket Template also implements the MQTT template, in order to communicate with the device. The MQTT template contains all the MQTT configurations needed to create the communication, such as server name, port and timeout values. In order to have a history of values, a Value Stream is implemented by Vital Jacket Template, this allows the storage of values with timestamps, which are used in charts to show the evolution of the values.

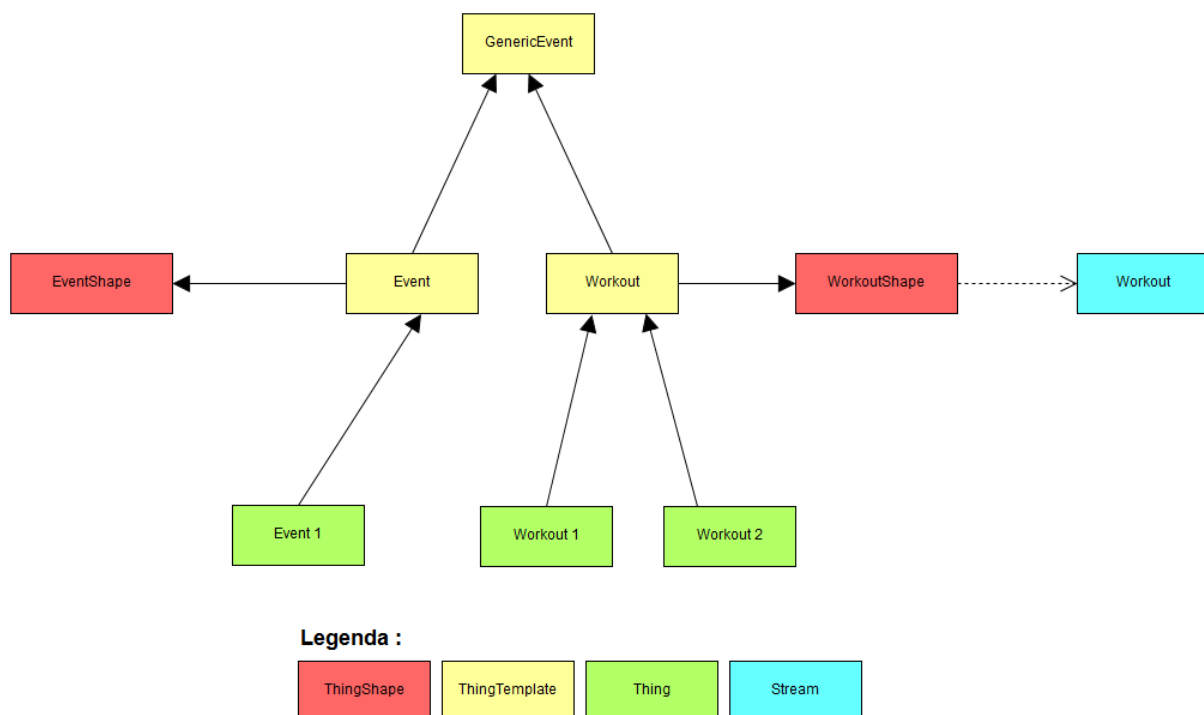


Figure 28 - Thingworx Diagram - Events

Figure 28 describes how the Events are implemented on the platform. There are two kinds of events, normal events and workout events, normal events only have a timestamp and a description, they are basically only used to mark a time. Workout events are more complex, as the name says, they describe a workout session and contain properties such as the duration of the workout, steps taken and calories burnt, in the workout a stream is implemented, since the need to keep a history of values such as calories and steps.

Let's take by example a user named John, John is wearing the Vital Jacket and just left home to go to the gym, he goes by foot and he wants to have an idea how his hearth behaves while he walks to the gym, to do this John clicks on the Vital Jacket button, creating a simple Event, with just a timestamp and a description (empty for now), if he wants he can already go to ThingWorx platform and fill in the description of this event, for example, "Just left home". He does the same when he arrives at the gym, for now he has two simple events that describe when he left home and when he arrived to the gym. Now John will begin his workout, and he wants to keep a record in our application, he will go into the application and will start a workout, creating a workout event, this workout will keep an history of how many calories he burnt, as well as other important information, which will all be stored on ThingWorx, John can press the Vital Jacket button during his workout to mark events such as the beginning of a new exercise. At the end of the workout John ends the workout on ThingWorx and go on with his life. At any time, he or his doctor/personal trainer, can go into ThingWorx and observe the

workout, with all the information available. He can also know, when observing his hearth history, when he left home and arrived at the gym, due to the events he created.

In the final implementation, the Workout Stream was removed, due to the option of getting the values directly from the Vital Jacket Thing, with the dates of the workout.

### 3.2.1.2 Components

Since this project involves a complex system, a component diagram has to be made. This diagram describe the relationship between the different components of the project.

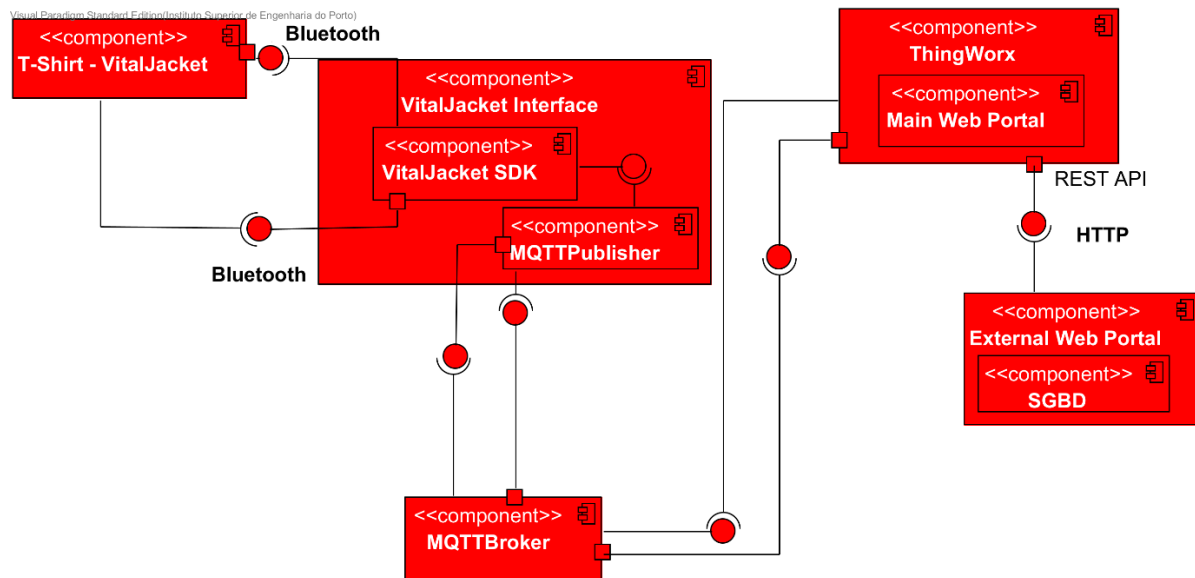


Figure 29 - Components Diagram

Figure 29 describes the different components of the project, there are three main components in this case, the Vital Jacket, the Vital Jacket Interface and the ThingWorx Platform. This solution was designed like this to allow a future expansion. If a new device is to be implemented in this solution, the only thing that needs to be developed is a Device Interface, which connects the Device to the ThingWorx Platform. A protocol adapter could be implemented in ThingWorx, that way it could communicate with different devices using different protocols. On this solution the protocol implemented was MQTT. The MQTT broker and MQTT client are located on the same machine.

### 3.2.1.3 Deployment

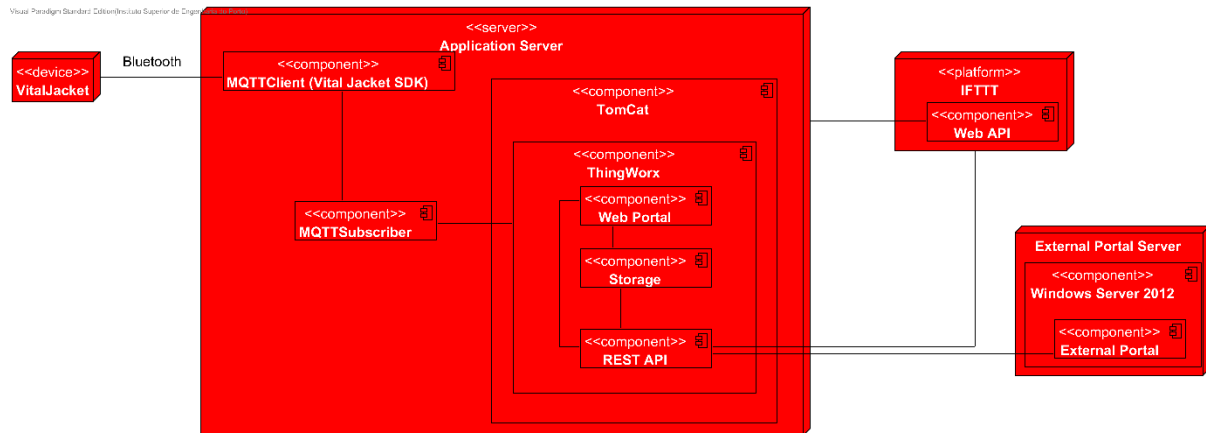


Figure 30 - Deployment Diagram

The different components explained in the component diagram must be developed in physical machines. As there are multiple devices on this project, a deployment diagram helps to understand which components go in which devices. Figure 30 shows the deployment diagram, there are two main devices, the Vital Jacket and the ThingWorx server. On the Vital Jacket the device sends information into the application server using Bluetooth. On the application server a MQTT client receives the Vital Jacket data and sends it, through MQTT, into the ThingWorx platform, which stores it and makes it available in the REST API. A communication with IFTTT would be made through the API's of ThingWorx and IFTTT servers.

## 3.2.2 INSTALATIONS

In order to have a working solution, software had to be installed to run the developed solution. These installations follow the deployment diagram, which shows where the software is located.

### 3.2.2.1 ThingWorx/Tomcat

A tomcat server was installed with ThingWorx on it, this server runs Linux Ubuntu. On the development phase the server was installed in a Virtual Machine, allowing for a development of the solution without the need of a physical server, which would bring more responsibility to the developer. After a substantial part of the development, the partial solution was deployed to a cloud server, which already had the ThingWorx platform with another project. In order to avoid conflicts between projects, a naming convention was applied with also the use of ThingWorx tags.

### 3.2.2.2 MQTT

To develop a working communication using MQTT, a broker is needed, this broker manages all the MQTT traffic, making sure every message reaches its subscribers. The broker used in this project was mosquitto, as mentioned before. To use mosquitto, the mosquitto package had to be installed on the ThingWorx server (it could also be installed in another server, but in this case it was installed in the same server as ThingWorx), this installation was just a simple *sudo apt-get install mosquitto*, due to mosquitto package being available in the Ubuntu repository. (Mosquitto Documentation, s.d.) Some of the keywords used on this, and future, commands are:

- ***sudo***<sup>31</sup> – Allows the user to run programs as root, giving permission that the user, otherwise, would not have.
- ***apt-get***<sup>32</sup> – Software package manager on Ubuntu, gives the user the ability to install, remove or upgrade programs.
- ***apt-add-repository***<sup>33</sup> – Used to add another repository to Linux, normally used when some wanted program is not available on the installed Linux repository.

Although the mosquitto package is available on Ubuntu software repository, the version available isn't the latest one, a newer version has the ability of reconnecting every subscriber when mosquitto is restarted. This feature isn't available in the version installed through Ubuntu repository. To update the mosquitto package to its latest version, the mosquitto official repository was added and the mosquitto package was update with these two commands:

1. *sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa*

Initially, the mosquitto official repository must be added to Ubuntu, this was done by executing the above command, utilizing *apt-add-repository* with the name of the repository, on this case *ppa:mosquitto-dev/mosquitto-ppa*, after adding the repository, all the mosquitto packages became available to install or upgrade to the latest versions.

2. *sudo apt-get install --only-upgrade mosquitto*

After adding the mosquitto official repository, the mosquitto package can upgrade. In order to do this, the above command has to be executed. The upgrade is done with the same command that installs packages (*apt-get install*), this command alone would install any package available, but because mosquitto is already installed, a simple upgrade is needed, in

---

<sup>31</sup> <http://linux.die.net/man/8/sudo>

<sup>32</sup> <http://linux.die.net/man/8/apt-get>

<sup>33</sup> <http://manpages.ubuntu.com/manpages/wily/man1/add-apt-repository.1.html>

order to achieve this upgrade the parameter *–only-upgrade* is used, indicating that the installation is not needed, only the upgrade to the latest version.

Besides the mosquitto package, a package named mosquitto-clients was also installed, this package allows to test mosquitto by publishing and subscribing to different topics.

### 3.2.3 SOLUTION DEVELOPMENT

After the design phase and with all the installations done, the development phase begins, on this phase the project begins to have a shape, getting closer to the final version.

#### 3.2.3.1 ThingWorx

The solution development began by setting up the ThingWorx platform. After installing ThingWorx on a virtual machine, the data structure was created in order to store and analyze all the information received, this task didn't quite follow the respective diagram, since the use of MQTT brought a need to implement a MQTT template on ThingWorx and a thing can only have one template. The diagram was changed and the data structure was created has can be seen below.

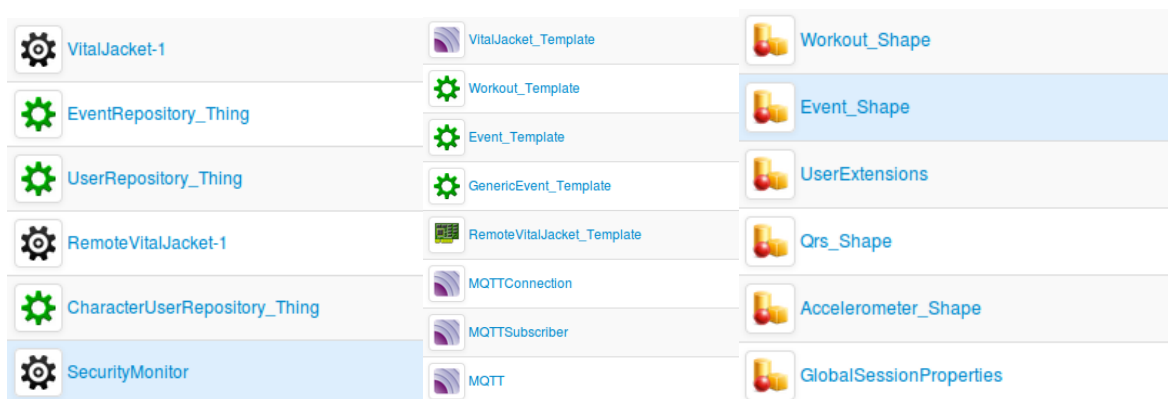


Figure 31 - ThingWorx Things

Figure 32 - ThingWorx Templates

Figure 33 - ThingWorx Shapes

Figure 31, 32 and 33 show the different structures that were created, in this case two Vital Jackets named "VitalJacket-1" and "RemoteVitalJacket-1" already existed for demonstration purposes.

**Connection Settings**

Name	Value
clientIdFormat	<input data-bbox="890 331 1241 376" type="text" value="{s}/{t}"/>
password	<input data-bbox="890 409 1040 443" type="button" value="Change Password"/>
qos	<input data-bbox="890 465 1241 510" type="text" value="0"/>
connectTimeout	<input data-bbox="890 533 1241 577" type="text" value="10000"/>
serverName	<input data-bbox="890 600 1241 645" type="text" value="127.0.0.1"/>
retryInterval	<input data-bbox="890 667 1241 712" type="text" value="1000"/>
serverPort	<input data-bbox="890 734 1241 779" type="text" value="10001"/>
userId	<input data-bbox="890 801 1241 846" type="text"/>
timeout	<input data-bbox="890 869 1241 913" type="text" value="5000"/>
useSSL	<input data-bbox="890 936 912 981" type="checkbox"/>

Figure 34 - MQTT ThingWorx connection settings

On the Vital Jacket template, the MQTT connections could be set, the two most important settings are `serverName` and `serverPort`, as these two are the ones that define where the Thing will send/receive information from. A more detailed explanation of the different fields follows:

- **clientIdFormat**

Format that will represent the ID of the MQTT client on the broker, this ID is automatically generated and his format can be set using the following tags:

{s} – Server name

{t} – Thing name

- **password, userID and useSSL**

These three parameters regard the security of the connection, `useSSL` defines if an SSL/TLS connection will be used and `userID` + `password` describe a set of authentication values.

- **qos**

Describes the level of QoS the connection uses, ThingWorx supports up to level 2 QoS. (0 – At most once, 1 – At least once and 2 – Exactly once).

- **connectTimeout, retryInterval and timeout**

Define times of certain connection events, such as the amount of time the client waits until he tries to reconnect to the broker or time that he waits until stopping the reconnection.

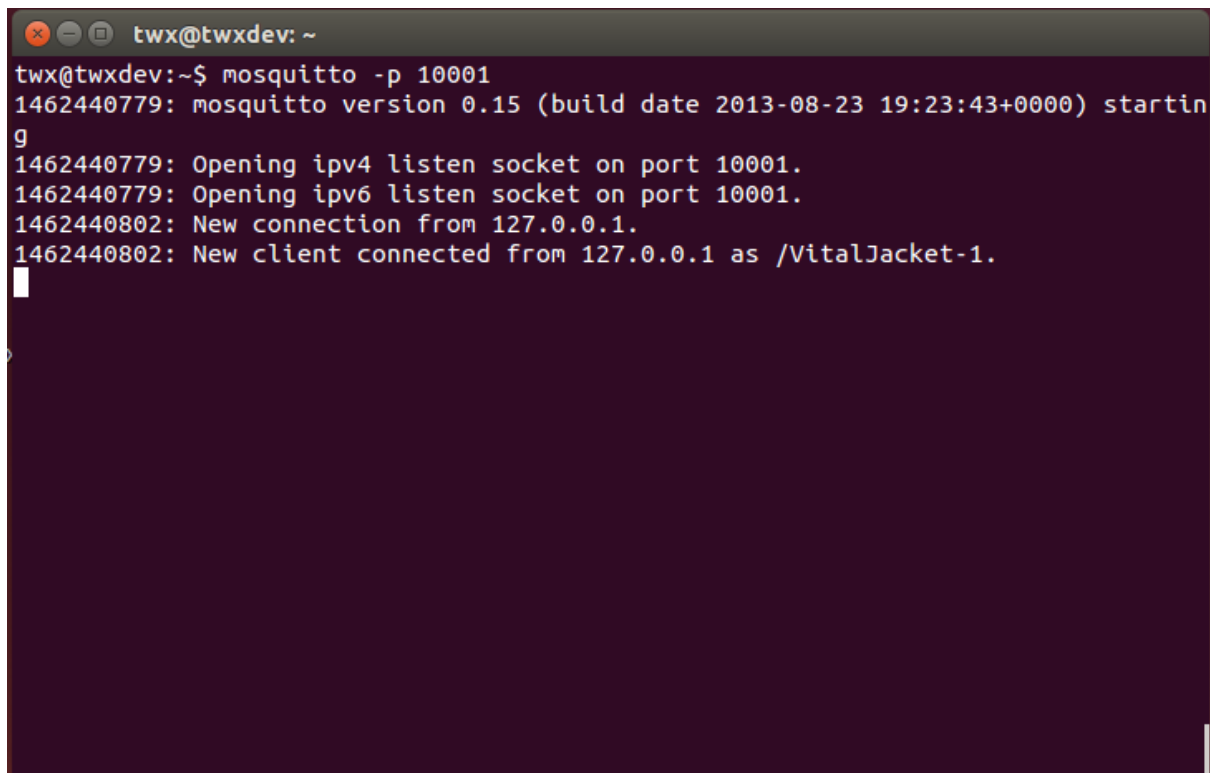


- **serverName** and **serverPort**

The most important parameters, they define the IP of the server where the broker is running and the port where the broker is listening to MQTT connections.

### 3.2.3.2 MQTT Connection

After the creation of the data structure, the solution needed a way of transporting the information to this structure, the MQTT connection was developed in order for this to happen. After the required installations (mosquitto and mosquitto-clients), mosquitto was started, listening to a specific port, in the case of figure 35, port 10001 was the chosen port.

A terminal window titled 'twx@twxdev: ~' showing the execution of the 'mosquitto -p 10001' command. The output shows the broker starting, opening IPv4 and IPv6 listen sockets on port 10001, and receiving a new connection from 127.0.0.1 as '/VitalJacket-1'.

```
twx@twxdev:~$ mosquitto -p 10001
1462440779: mosquitto version 0.15 (build date 2013-08-23 19:23:43+0000) starting
1462440779: Opening ipv4 listen socket on port 10001.
1462440779: Opening ipv6 listen socket on port 10001.
1462440802: New connection from 127.0.0.1.
1462440802: New client connected from 127.0.0.1 as /VitalJacket-1.
```

Figure 35 - Mosquitto broker running

On figure 35 mosquitto is running and a client already connected, each time a client connects a message can be seen on the terminal window (as it can be seen with VitalJacket-1). Mosquitto can be start with the `-d` parameter, which starts mosquitto in the background as a daemon, removing the messages each time a client connects. There are more ways of checking information about the broker by subscribing to specific topics, which hold some important data. Some of these topics are listed below:

- `$SYS/broker/bytes/received` - The total number of bytes received since the broker started.
- `$SYS/broker/bytes/sent` - The total number of bytes sent since the broker started.
- `$SYS/broker/uptime` - The amount of time in seconds the broker has been online.

At any time the developer can subscribe to one of the given topics to receive information about the broker.

Besides these built in topics, topics for each VitalJacket are created, with the following pattern:

- `/{{Name of the Thing}}/{{Name of the property}}`

For example, if we wanted to change the temperature of “VitalJacket-1” to 36, we would publish the value 36 to the topic “/VitalJacket-1/temperature”. In the final solution this publish is done with a C# MQTT library, but for testing purposes, `mosquitto_pub` could be used for publishing the value, the command would look like this :

```
mosquitto_pub -p 10001 -t '/VitalJacket-1/temperature' -m '36'
```

Command `mosquitto_pub` can take various parameters, being some of the most important:

- `-p` Port that the mosquitto will listen to, default is 1883.
- `-t` Topic in which the mosquitto will publish
- `-m` Message that will be published
- `-h` Hostname of the server, default is localhost

(Mosquitto Documentation, s.d.)

### 3.2.3.3 MQTT Client

Now that the data is structured on ThingWorx and the MQTT connection is available, a MQTT client has to be developed, in order to send the data to ThingWorx Things by publishing to the specific topics. Two MQTT clients were developed in this project, a simulation developed locally in order to test the transmission of data into ThingWorx and a client using the Vital Jacket SDK, which sent real data coming from the Vital Jacket or binary file created by the Vital Jacket.

- **MQTT Simulation Client**

The first MQTT client is a C program that uses the MQTT client library in C<sup>34</sup>. This program generates random values simulating the Vital Jacket data and sends this values trough MQTT. The communication done between this client, mosquitto and ThingWorx is local, since all of them are located on the same machine (MQTT Client library for C, 2014).

```

twx@twxdev: ~/Desktop/MQTTclient
twx@twxdev:~/Desktop/MQTTclient$ ./MqttSimulation VitalJacket-1
38 on /VitalJacket-1/Temperature for Simulated-VitalJacket-1
Message with delivery token 0 delivered
2.528000 on /VitalJacket-1/AccX for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
0.290000 on /VitalJacket-1/AccY for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
3.742000 on /VitalJacket-1/AccZ for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
103 on /VitalJacket-1/HearthRate for Simulated-VitalJacket-1
Message with delivery token 6 delivered
34 on /VitalJacket-1/Temperature for Simulated-VitalJacket-1
Message with delivery token 0 delivered
2.545000 on /VitalJacket-1/AccX for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
2.222000 on /VitalJacket-1/AccY for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
1.358000 on /VitalJacket-1/AccZ for Simulated-VitalJacket-1
Message with delivery token 7606612 delivered
113 on /VitalJacket-1/HearthRate for Simulated-VitalJacket-1
Message with delivery token 6 delivered

```

Figure 36 - MQTT Client running

On figure 36 the MQTT client confirm messages can be seen, including the topics and messages that were delivered to ThingWorx. This developed client takes the device name by parameter as can be seen in the first line (`./MqttSimulation VitalJacket-1`) and creates a client named “Simulated-<Device Name>”, on this case “Simulated-VitalJacket-1”, since it’s not possible to exist two clients with the same name, and VitalJacket-1 is the ThingWorx MQTT client name for the Thing.

<sup>34</sup> Source: <https://www.eclipse.org/paho/files/mqtt/doc/Cclient/index.html>



Figure 37 – MQTT information flow

Figure 37 shows a simple representation of the information flow on a single MQTT publish/subscribe sequence.

- **Vital Jacket SDK MQTT client**

The second MQTT client is the one developed with the Vital Jacket SDK, which uses real data instead of the first simulation, which uses randomly generated values. This client is developed in C#, due to being the language that the SDK was developed in, and uses the M2Mqtt library. To remove the necessity of having the Vital Jacket connected to the computer each time the developers wanted to test the solution, a simulation which simulates receiving data from the Vital Jacket was developed. This simulation uses the binary files that are saved by the Vital Jacket when it's not connected to a computer. Essentially, there are two ways of sending data, by the binary file (data recorded previously) or by the Bluetooth connected Vital Jacket (live data) (BioDevices, s.d.).

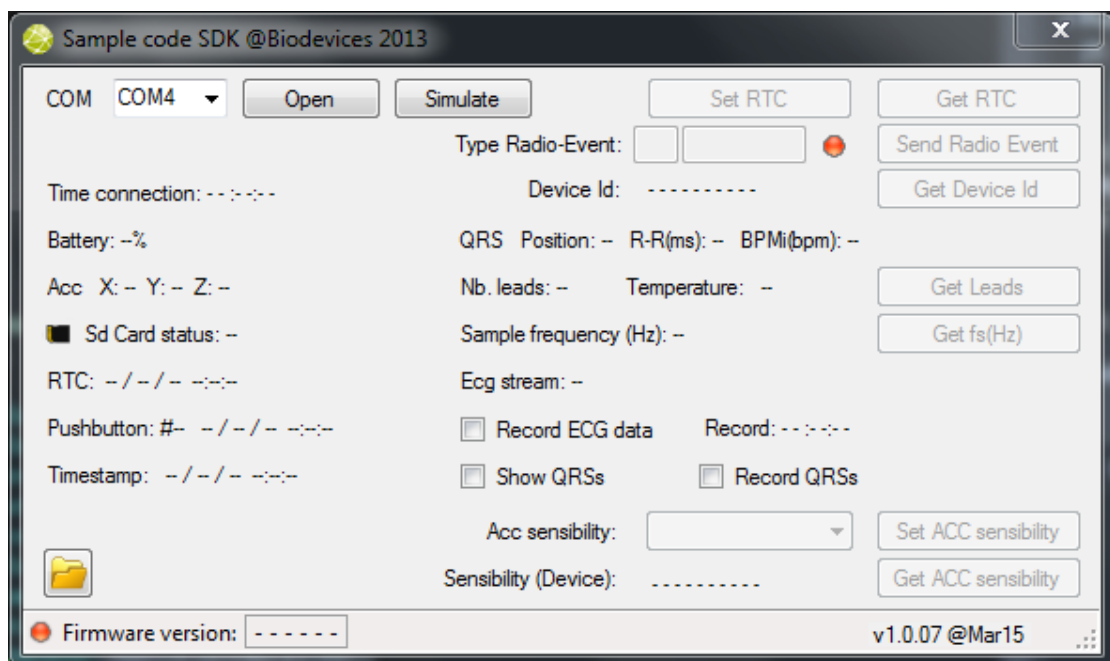


Figure 38 - Vital Jacket SDK

By pressing the Simulate button, as seen in figure 38, the data from a binary file is read, by pressing the Open button with a Bluetooth connection selected in the COM field, the Vital Jacket data is gathered. After receiving the information, the client will show the information on the specific fields and will send it to ThingWorx, via MQTT (C# .Net and WinRT Client, s.d.).

To open a MQTT connection the following code has to be implemented:

```
MqttClient client = new MqttClient(IPAddress.Parse("127.0.0.1"),
11883, false, null);
string clientId = "VitalJacket-1";
client.Connect(clientId + "-Publisher");
msgId = client.Publish("/") + clientId + "/Temperature", temperatureValue);
client.Disconnect();
```

In this example the client publishes a value to the temperature topic and disconnects, but it could send a variety of values before disconnecting, in this project, the client sends values until the Vital Jacket Bluetooth connection disconnects or the simulation is stopped.

A detailed explanation of the code follows up:

1. `MqttClient client = new MqttClient(IPAddress.Parse("127.0.0.1"), 11883, false, null);`

The first line of this script created an object that represents the connection with the MQTT broker. The first couple parameters represent the IP and Port of the broker, on this example the broker is located on the same machine (because of the loopback address, 127.0.0.1) and is listening to port 11883. The third and fourth parameters are related to security (the use of TLS and the respective certificate) which in this case it's not used.

2. `string clientId = "VitalJacket-1";`

The second line of the script simply declares a string with the ID of the client, which will identify it on the broker, this ID can't be equal to any client ID already connected to the broker.

3. `client.Connect(clientId + "-Publisher");`

The third line connects the client created on the first line with the client ID created on the second line, it's important to note that the program adds the string "-Publisher" in the end of the client ID, in order to have a unique ID, since, supposedly, there is already a client without the "-Publisher" string at the end (the one connected with ThingWorx).

```
4. msgId = client.Publish("/") + clientID + "/Temperature", temperatureValue);
```

The fourth line of this script publishes the temperatureValue (whatever value the variable contains) to the “/{Client ID}/Temperature” topic, changing that property value on ThingWorx. It’s not shown on this script, but msgId is an integer defined at the beginning of the script, this integer is the return value of the connection, which can be used to check if the connection was received. This method will be called for each value the script wants to change, publishing various values within one connection.

```
5. client.Disconnect();
```

The client can be disconnected with a simple Disconnect() call, this function is going to be called whenever the developer wants to close the connection, losing the possibility of publishing/subscribing to any topics, until another connection is started.

This solution only allowed for the simulation of one Vital Jacket with one ECG exam binary file and was used mainly to test if the MQTT connection worked, after the development of the working MQTT connection, some features were added.

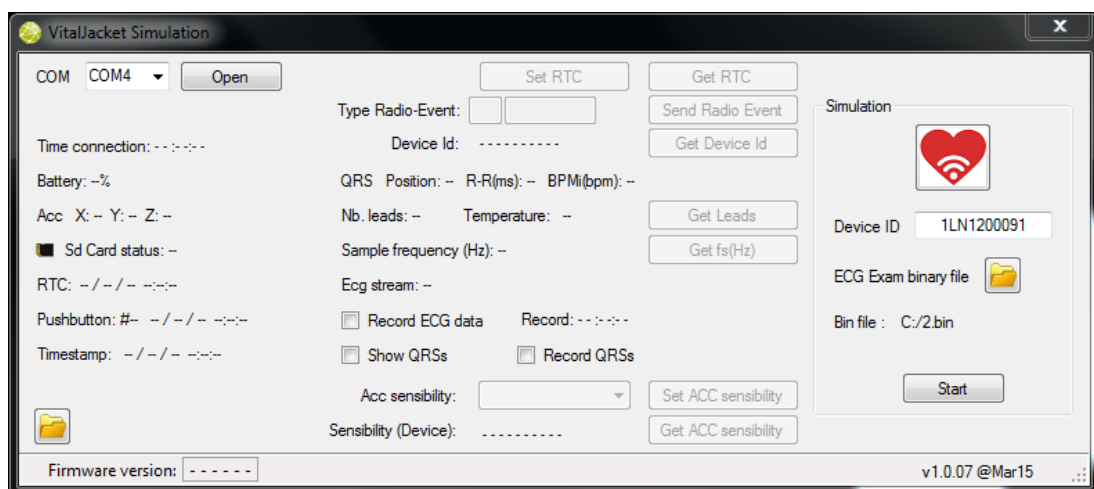


Figure 39 – Updated Vital Jacket SDK

As it can be seen on figure 39, the design was re-arranged and a new area was created for the simulation of the Vital Jacket (area on the right), on this area the user can introduce the device ID and select the binary file that he wants to simulate, this allows the simulation of numerous devices with different information.

### 3.2.3.4 Connectivity Management

Connection is done with MQTT, allowing the devices to communicate with ThingWorx. The connection can be lost at any time, which gives a responsibility to the developer of managing the connectivity, deciding what must be done when a device disconnects and re-connects. This management can be done in two parts of the project.

- **Mosquitto broker**

The MQTT broker chosen, mosquitto, deals with the reconnection of the MQTT publishers/subscribers, reconnecting publishers/subscribers to their topics, each time they reconnect with the broker.

- **Vital Jacket**

With Vital Jacket constant data gathering, something has to be done when the connection is lost, in order not to lose the gathered data. Vital Jacket, when not connected with ThingWorx, stores the data gathered in a SD card. When reconnecting with ThingWorx, nothing is done with the data stored in the SD card, but it could be sent to ThingWorx, preventing the lack of the information, when the Vital Jacket wasn't connected, on ThingWorx.

### 3.2.3.5 IFTTT Integration

In order to integrate IFTTT in this project, two functionalities were implemented, Actions and Triggers, they both work with REST API calls and allow connection with other systems.

- **Actions**

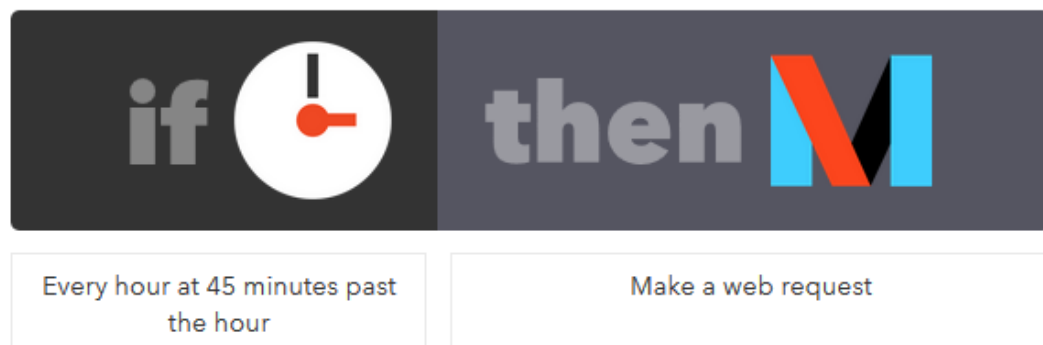


Figure 40 - Created action

Actions describe information flow from IFTTT to Thingworx. An action allows IFTTT to alert Thingworx when some programmed task happens (your battery hits a low level for example), doing an action on Thingworx side. This action calls methods available on the Thingworx Rest API. For demonstration purposes, an action that created events every 45 minutes past the hour was implemented. Before creating the action, a Thingworx application key was created, to authenticate the REST API call.

## Action

### Make a web request

This Action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

#### URL

```
https://thingworxdev01.cloudapp.net/Thingworx
/Things/CF.WT_TH.EventFactory.v1/Services
/CreateEventExternal?appKey=2f806427-5365-482d-
b56c-ee563ad91a47&Accept=application/json
```

Surround any text with "<<<" and ">>>" to escape the content

#### Method

POST

The method of the request e.g. GET, POST, DELETE

#### Content Type

application/json

Optional

#### Body

```
{ Description_M = "Description", userID =
"FernandoEsparrinha" }
```

Surround any text with "<<<" and ">>>" to escape the content

Figure 41 – Action creation interface

Using the application key and the REST API URL, a POST method was implemented, this method would execute a service that would create an event. The creation of this REST API call can be observed in figure 41, the following parameters have to be inserted:

- **URL**  
Reference to where the REST API call is located, on this case two parameters were added in the URL, the application key (*appKey=*) and the Accept parameter (*Accept=*) which tells the server which content he will accept, in this case json.
- **Method**  
Describes which method the request will have (GET, POST, DELETE), In this case a POST method was created in order to call a Thingworx service.
- **ContentType**  
The type of content that the method will carry, can be chosen between json, xml, plain text and many other.



- **Body**

The information that the method carries, IFTTT permits the insertion of parameters inside the body in order for the body to be customizable. On this case, because the action is activated by time, the time when the action is activated could be inserted in the body of the request.

- **Triggers**

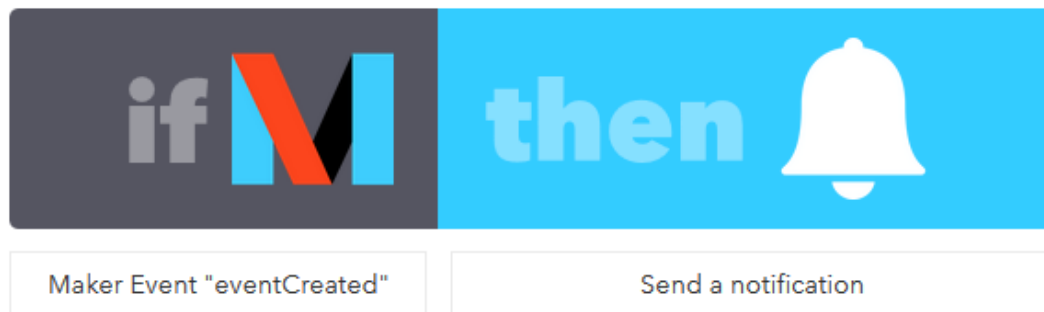


Figure 42 - Created trigger

Triggers describe information flow from Thingworx to IFTTT. They allow IFTTT to do something (play music for example) whenever Thingworx activates this trigger. After the creation of the trigger on the IFTTT website, a URL is provided. This URL shall be called by Thingworx whenever the trigger is going to be activated. For a working demo, a trigger was created, this trigger sends a notification to an android phone whenever an event is created on the Thingworx portal.

## Trigger

### Receive a web request

This Trigger fires every time the Maker Channel receives a web request to notify it of an event. See "How to Trigger Events" on the Maker Channel page (<https://ifttt.com/maker>) for more information.

#### Event Name

The name of the event, like "button\_pressed" or "front\_door\_opened"

## Action

### Send a notification

This Action will send a notification to your devices.

#### Notification

Figure 43 - Trigger creation interface

The creation of the trigger is a lot simpler than the creation of an action, here the developer just needs to insert a name that will define the URL of the created request. To trigger the event, the developer has to make an API call to the created request.

On this case, the notification shown may also carry parameters coming from ThingWorx, as can be seen on figure 43, the notification carries the parameters Value1 and Value2 that are parameters that ThingWorx inserts when calling the API, on this case, Value 1 carries the name of the creator and Value2 carries the description of the event.

```

var jsonContents = {
    "value1" : creator,
    "value2" : Description_M
}

var params = {
    proxyScheme: undefined /* STRING */,
    headers: undefined /* JSON */,
    ignoreSSLerrors: undefined /* BOOLEAN */,
    useNTLM: undefined /* BOOLEAN */,
    workstation: undefined /* STRING */,
    useProxy: undefined /* BOOLEAN */,
    withCookies: undefined /* BOOLEAN */,
    proxyHost: undefined /* STRING */,
    url: "https://maker.ifttt.com/trigger/eventCreated/with/key/bELtnY24CYh1IgcG3p0Ua" /* STRING */,
    content: jsonContents /* JSON */,
    timeout: undefined /* NUMBER */,
    proxyPort: undefined /* INTEGER */,
    password: undefined /* STRING */,
    domain: undefined /* STRING */,
    username: undefined /* STRING */
};

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

```

Figure 44 - IFTTT API Call

On figure 44 the call to the created request can be observed, this call sends the created event information and uses an application key provided by IFTTT. The parameters send to IFTTT can also be seen on the variable *jsonContents*, on the first line. The user, after creating an event, receives a notification on his smartphone with the event information.

### 3.2.4 ANALYSIS

During the solution development, there are choices to be made, as there are multiple ways of implementing a solution, these choices bring great responsibility to the developer and have to be decided based on proven facts. These facts come from the analysis of results and study of requisites. This sub chapter will describe the analysis done in this project as well as its requisites, with results and drawn conclusions.

### 3.2.4.1 MQTT Protocol

After the implementation of MQTT protocol, Wireshark was used to analyze the packets transmitted. This tool was a great resource due to all the information that it provides.



### Figure 45 - TCP MQTT Packet

Figure 45 shows some information about the TCP header of a packet being transmitted with MQTT. This implementation doesn't use TLS yet, so the information is transmitted in plaintext, which is not ideal to the solution. Applying TLS, the information would be transmitted in ciphertext and wouldn't be able to be read due to encryption.

### 3.2.4.2 Vital Jacket SDK MQTT Client

To analyze the Vital Jacket SDK MQTT Client, the diagnostic tool in Visual Studio was used. Since the application was developed on Visual Studio, this tool was handy and did not need any previous arrangement.

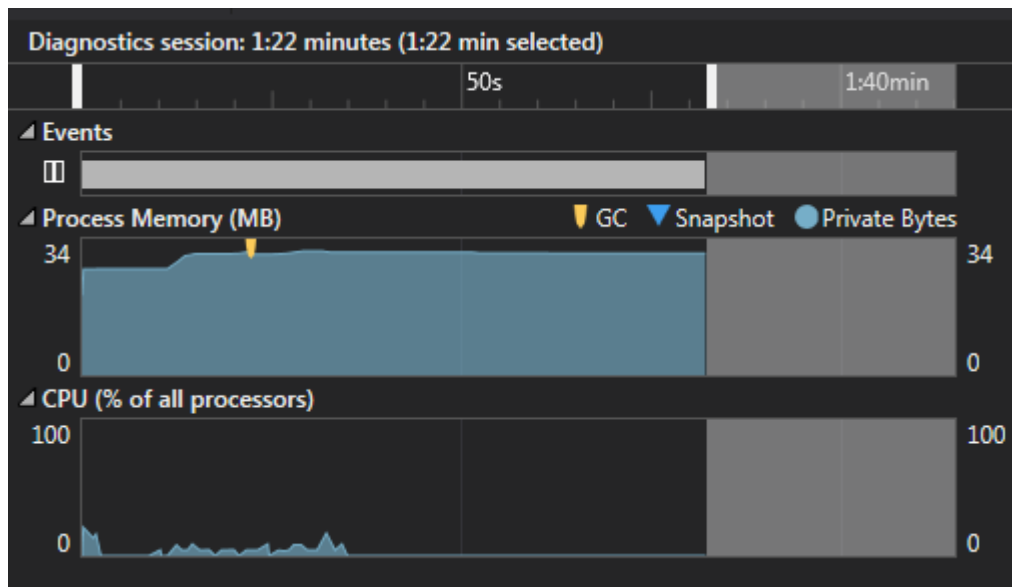


Figure 46 - Visual Studio Diagnostic Tool

In the diagnostic tool, as can be seen on figure 46, the memory spent by the process is 30 MB in the beginning of the program, after starting a simulation the memory needed by the process rises up to 34 MB and stays like that for the rest of the application time. The percentage of CPU was very low and only rose a small amount when the simulation was started. Overall this application spent a small amount of resources.

### 3.2.4.3 Requisites

This project has some requisites, functional and non-functional, that influenced the development of the project, restricting it in some ways.

Table 7 - Functional and Non-functional requisites

Vital Jacket	MQTT (with Vital Jacket Bluetooth connection)	MQTT simulation (with ECG binary file)	ThingWorx
Has to be connected with Bluetooth 2.0.	Information has to be sent in real-time.	Has to run with an ECG binary file created by a Vital Jacket.	Has to run on Apache Tomcat server.
Has to use Vital Jacket SD card (specific formatting)	Must be connected to the patient via electrodes.	Has to run with a device ID already created on ThingWorx.	Has to run on Linux Ubuntu.
	Has to have mosquitto broker running.	Has to have mosquitto broker running.	Has to follow Celfocus naming convention.
	Vital Jacket ID must already be on a ThingWorx Thing.		
Non-functional requisites		Functional requisites	

### 3.2.5 TESTS

After development, there must be a way to prove the quality and effectiveness of the developed software. Software tests are used to prove that fact, by testing various points such as performance, usability or scalability.

The tests done in this project follow a black box<sup>35</sup> method, which treats the application as a black box, without looking at the implementation or source code. For each case, scripts were created in order to test various functionalities.

<sup>35</sup> [https://en.wikipedia.org/wiki/Black-box\\_testing](https://en.wikipedia.org/wiki/Black-box_testing)

### 3.2.5.1 MQTT broker (mosquitto)

After installing mosquitto broker some test were done in order to know if mosquitto was working well. For this tests three command lines were running at the same time, one for each task (publish, subscribe and broker). The three different command lines are described on table 8, on the first three columns.

Table 8 - Mosquitto test script

Task (Publish)	Task (Subscribe)	Task (broker)	Expected Result	Actual Result
		<b>Start mosquitto</b>	Mosquitto running.	Mosquitto running.
	<b>Subscribe to topic "temperature"</b>	<b>Receive notification of new subscriber</b>	See new subscription on broker.	New subscription on broker.
<b>Publish value 28 on topic "temperature"</b>	<b>Received value 28</b>	<b>Receive notification of new publisher and send value 28 to subscriber</b>	See value 28 on the subscriber and notification of new publisher on the broker.	Subscribe received 28 and notification of new publisher appeared on the broker.
	<b>Disconnects from the broker</b>	<b>Receive notification of disconnecting subscriber</b>	See the disconnection of the subscriber.	Subscriber disconnection notification on the broker.

### 3.2.5.2 Vital Jacket SDK MQTT client

To test the Vital Jacket SDK MQTT client simulations were created and restarted in order to check the how the application responses and the existence of bugs and/or errors.

Table 9 - Script for MQTT client testing

Task	Expected Result	Actual Result
<b>1. Insert Vital Jacket ID and ECG binary file</b>	Receive and store the information.	Received and stored the information.
<b>2. Begin Simulation</b>	Simulation starts running without any crash.	Simulation runs without a crash if mosquitto is running on the server.  If mosquitto isn't running, application crashes with an exception.
<b>3. Check if information is being transmitted to Thingworx</b>	Vital Jacket information updates on Thingworx.	Vital Jacket information updated on Thingworx
<b>4. Check if events are being created when the Vital Jacket button is pressed</b>	New event created.	New event not created due to Thingworx MQTT bug (ThingWorx receives the push button event).
<b>5. Stop Simulation</b>	Application stops the simulation and continues running.	Application stops the simulation but sometimes quits with an exception.
<b>6. Start simulation again</b>	Application runs another successful simulation.	Application has to be restarted for a successful simulation to be ran.



### 3.2.5.3 IFTTT

To test IFTTT, actions and triggers were created and activated, allowing to see the system response and integration with Thingworx. Numerous triggers and actions could be created. For this test, the trigger created sends a notification to an android phone, every time an event is created and the action creates one event every hour that passes.

Table 10 – IFTTT tests

Task	Expected Result	Actual Result
1. Create IFTTT Trigger (send notification when event created)	Successful creation of the trigger.	Trigger created successfully.
2. Create IFTTT Action (create event every hour)	Successful creation of the action.	Action created successfully.
3. Activate trigger by creating event	Create Event and receive notification on android phone.	Event created and notification received.
4. Activate action by waiting for the specific time	Creates event in the specific hour.	Event is not created.

Task 4 on table 10 didn't pass successfully, because of a failed API call, the resolution of this problem wasn't found, but was likely due to some missing API request parameters that the IFTTT platform didn't allow to add.

### 3.2.5.4 Other Tests

There were other tests planned that unfortunately weren't made due to lack of time, the most important of these tests is the scalability tests. Scalability tests allow for the developer to have the capability of the application tested. In this project, scalability tests would be done by creating and simulating a large number of devices until the application doesn't work the way it's supposed to. The parameter tested in this case would be the number of active devices on the system and users logged in simultaneously.



## CHAPTER 4 CONCLUSION

At the end of the project a lot of conclusions can be draw. In this chapter, the reader can acknowledge the objectives that were complete, limitations had and the different ways the project can evolve in the future. During this project some extra activities were done, which relate to the project one way or another, this activities are described in “Other Work”. A final appreciation closes this document with a personal opinion about the whole project.

<b>Chapter 4</b>	<b>Conclusion</b>	<b>65</b>
<b>4.1</b>	<b>Summary</b>	<b>66</b>
<b>4.2</b>	<b>Objectives Complete</b>	<b>66</b>
<b>4.3</b>	<b>Limitations and Future Work</b>	<b>67</b>
<b>4.4</b>	<b>Other Work</b>	<b>68</b>
<b>4.5</b>	<b>Final Appreciation</b>	<b>69</b>

## 4.1 Summary

In the end, this project allowed to demonstrate its concept, making it clear that it had potential of being used on a bigger scale, all the ideas add in the beginning of the project were improved and almost all of them were developed. Some features weren't implemented due to the delivery date, but the technology needed for these features was researched, giving precious information that would favor the development of these features.

## 4.2 Objectives Complete

The project was developed in a limited time, so not all the objectives were completed.

Table 11 - State of Objectives

Objective	State	Left to do
Thingworx Information Structure	Done	
MQTT Connection (simulation)	Done	Remove bug when the simulation has events created. Remove bug that does not allow for simulation restart.
MQTT Connection ( Bluetooth)	Done	Remove bug when creating events on the Vital Jacket.
Vital Jacket SDK	Done	
AlwaysOn Connection	Halfway done	Implement Vital Jacket SDK with the AlwaysOn connection.
CoAP Connection	Not done	Implement Vital Jacket with CoAP and test connection.
Protocol Adapter	Not done	Implement adapter that accepts various communication protocols and connects them with ThingWorx.
Encryption	Not done	Implement TSL. Implement hash function to store passwords.

<b>HL7 Export</b>	Not done	Export information using the HL7 standard.
<b>IFTTT integration</b>	Halfway done	Implement Actions.

## 4.3 Limitations and Future Work

The biggest limitation on this project was the time, the delivery date stipulated didn't allow to finish all the objectives defined, but with the knowledge gained throughout the project, the unfinished objectives are easy to complete, due to almost all of them being related to the Thingworx platform.

Thingworx had some bugs, mainly in communication with external devices, which delayed some main objectives, this made the project advance at a slower rate, as a couple of weeks were spend resolving bugs.

Another big challenge to this project was the ISEP disciplines happening at the same time, which had various projects to deliver throughout the duration of this project. These disciplines took a considerable amount of time from this project and also ended up slowing the development.

A month after the beginning of the project, Celfocus moved its offices to a new location (Vila Nova de Gaia), this move caused a bit of delay on the development since changes had to be made but, in the end, the delay was worth because the work conditions improved.

In order to continue the development of this project and to achieve a better solution, the following work must be done:

- **CoAP and AlwaysOn implementations** - these communication protocols would be ideal to be tested and compared to MQTT in order to find the best solution possible.
- **Encryption** – using TLS could improve drastically the security of the application. The messages are sent in plain text, which causes a big security failure because everyone that intercepts a packet is able to read its contents.
- **IFTTT Actions** – IFTTT actions ended up not being implemented due to not working with the Thingworx REST API. This task should be worked in order to at least find a workaround.

## 4.4 Other Work

During the time at Celfocus, all the work asked to be done was related to this project, but some extra activities were done in order to integrate with Celfocus and help with the development of the project:

- **Thingworx training**

During two days, a Celfocus collaborator helped with an introduction to ThingWorx, this training was very important since it introduced ThingWorx main features and allowed for a better understanding of the platform. Without this training, development would be a lot harder.

- **Biodevices visit**

During one morning, a visit to Biodevices (the company that created Vital Jacket) was made, during this visit an introduction to the Vital Jacket SDK was done, this visit was important because it allowed to clarify some questions about the SDK and Vital Jacket.

- **Basketball games**

Once per week Celfocus combined basketball games. These games, although not related to the project, were a great opportunity to integrate with the company collaborators and relax after a day of work.

- **Confluence Documentation**

Although every project needs documentation, the one written in Confluence was an extra documentation for use of the company in the future, this is the reason why it was considered extra work. The project main documentation is located in this report. Even though this documentation was an extra activity, it helped in a better understanding of the project scope and helped to improve writing.

## 4.5 Final Appreciation

Since the beginning of the project decisions were mostly made by me and my colleague João Gomes, with the confirmation of our supervisor, since we were the only ones developing this project. We had a lot of freedom to make decisions regarding development and I don't think our decisions ended up being bad decisions, there is only one point that I would change in the project, the choice of the IoT platform. ThingWorx is a good platform but had some weaknesses that made the development harder relatively to communication between ThingWorx and Vital Jacket.

If another solution could be chosen and there was no time limit, I would insert an android application in the project, to serve as a middle point that could analyze data before sending it to ThingWorx and work in offline mode, this solution can be observed on chapter 2.2.4 (final solution of the chapter). This implementation would require more time due to the insertion of a smartphone application, which would require software design, smartphone development knowledge and development time but, in my opinion, it would be worth since the final result would be a much richer application that works in both offline and online and allows the user to have a bigger information flow throughout our system.





## CHAPTER 5 BIBLIOGRAPHY

- BioDevices. (n.d.). *Products*. Retrieved 02 29, 2016, from VitalJacket: [http://www.vitaljacket.com/?page\\_id=156](http://www.vitaljacket.com/?page_id=156)
- C# .Net and WinRT Client. (n.d.). Retrieved May 31, 2016, from eclipse.org: <https://eclipse.org/paho/clients/dotnet/>
- Celfocus. (n.d.). *About Celfocus*. Retrieved 03 03, 2016, from Celfocus: <http://www.celfocus.com/about/>
- Diffie-Hellman key exchange (exponential key exchange)*. (2007, August). Retrieved March 14, 2016, from TechTarget: <http://searchsecurity.techtarget.com/definition/Diffie-Hellman-key-exchange>
- IBM Watson. (n.d.). Retrieved June 01, 2016, from ibm.com: <http://www.ibm.com/smarterplanet/us/en/ibmwatson/>
- Jaffey, T. (2014, 02). *MQTT and CoAP, IoT Protocols*. Retrieved 03 11, 2016, from eclipse: [https://eclipse.org/community/eclipse\\_newsletter/2014/february/article2.php](https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php)
- Medical Dictionary - Healthcare*. (n.d.). Retrieved June 2, 2016, from thefreedictionary: <http://medical-dictionary.thefreedictionary.com/health+care>
- Mosquitto Documentation*. (n.d.). Retrieved June 01, 2016, from mosquitto.org: <http://mosquitto.org/documentation/>
- MQTT Client library for C*. (2014, June 25). Retrieved May 30, 2016, from eclipse.org: <https://www.eclipse.org/paho/files/mqtt/doc/Cclient/index.html>
- PTC. (2016). *Things have changed*. Retrieved 03 03, 2016, from ThingWorx: <http://www.thingworx.com/About>
- quora. (n.d.). *What are the top 10 IoT platforms? What are their key features? Why would you prefer them over other platforms? Is there a benchmark?* Retrieved March 16, 2016, from quora: <https://www.quora.com/What-are-the-top-10-IoT-platforms-What-are-their-key-features-Why-would-you-prefer-them-over-other-platforms-Is-there-a-benchmark>
- Rouse, M. (2014, November). *Advanced Encryption Standard (AES)*. Retrieved March 15, 2016, from TechTarget: <http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard>
- Rouse, M. (2014, November). *RSA algorithm (Rivest-Shamir-Adleman)*. Retrieved March 15, 2016, from TechTarget: <http://searchsecurity.techtarget.com/definition/RSA>

Schneider, S. (2013, 10 09). *Understanding The Protocols Behind The Internet Of Things*. Retrieved 03 17, 2016, from eletronicdesign: <http://electronicdesign.com/iot/understanding-protocols-behind-internet-things>

*WebSockets vs REST: Understanding the Difference*. (2015, January 5). Retrieved May 25, 2016, from PubNub: <https://www.pubnub.com/blog/2015-01-05-websockets-vs-rest-api-understanding-the-difference/>

*What is technology*. (2013, December 12). Retrieved June 1, 2016, from useoftechnology: <http://www.useoftechnology.com/what-is-technology/>

XMPP. (n.d.). *Uses of XMPP*. Retrieved March 10, 2016, from XMPP: <http://xmpp.org/uses>

## CHAPTER 6 APPENDIXES

## 6.1 Full WBS of the project with Gantt Chart

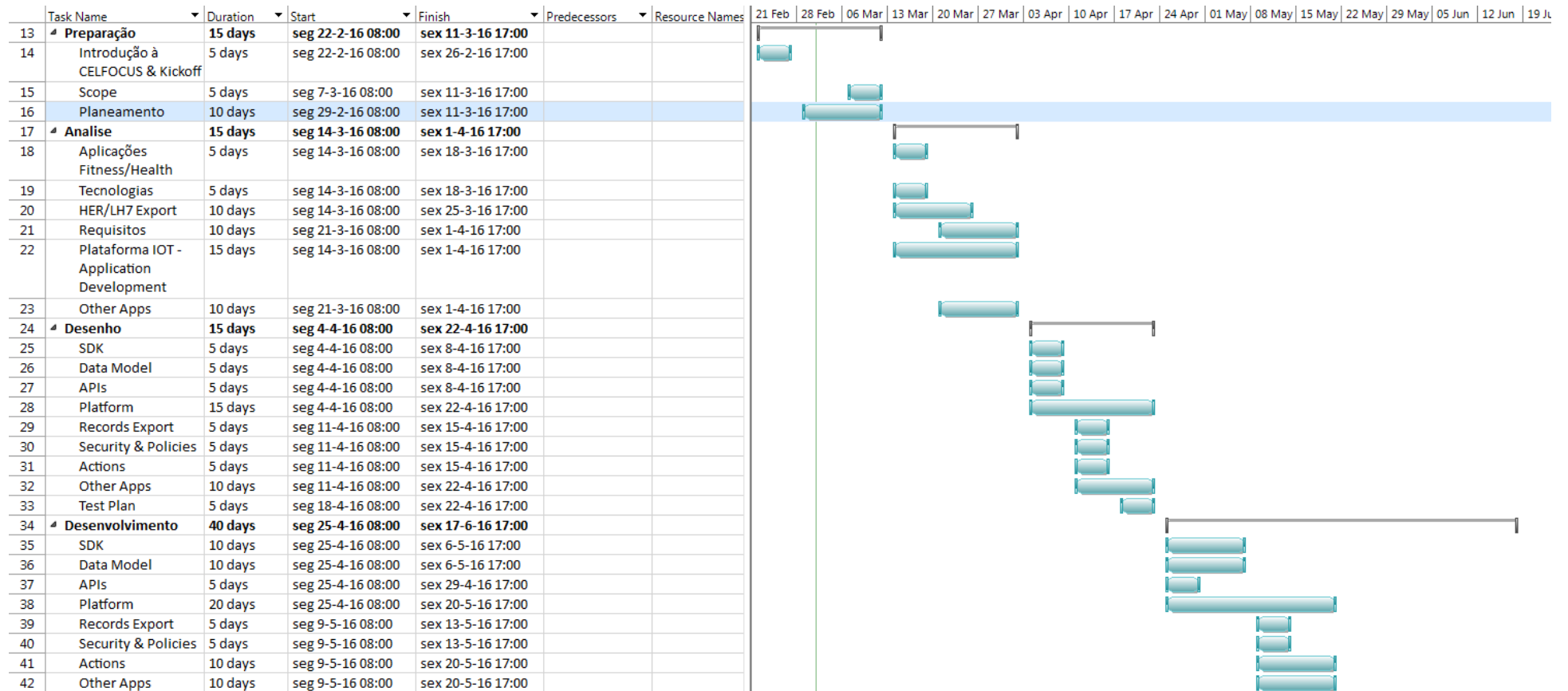


Figure 47 - Complete WBS of the project and Gantt Chart

## 6.2 MQTT Packet

```

▶Frame 222: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface 0
▶Linux cooked capture
▼Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
  Version: 4
  Header length: 20 bytes
  ▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 85
    Identification: 0xeab0 (60080)
  ▶Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ▶Header checksum: 0x51f0 [validation disabled]
    Source: 127.0.0.1 (127.0.0.1)
    Destination: 127.0.0.1 (127.0.0.1)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
▶Transmission Control Protocol, Src Port: 56426 (56426), Dst Port: scp-config (10001), Seq: 46, Ack: 5, Len: 33
▶Data (33 bytes)
0000  00 00 03 04 00 06 00 00 00 00 00 00 00 08 00  .....
0010  45 00 00 55 ea b0 40 00 40 06 51 f0 7f 00 00 01  E..U...@. @.Q....
0020  7f 00 00 01 dc 6a 27 11 be 03 55 35 df fa d3 5f  ....j'. ..U5...
0030  80 18 01 56 fe 49 00 00 01 01 08 0a 00 06 bf c5  ...V.I.. ....
0040  00 06 bf c5 30 1f 00 1a 2f 56 69 74 61 6c 4a 61  ....0... /VitalJa
0050  63 6b 65 74 2d 31 2f 54 65 6d 70 65 72 61 74 75  cket-1/T emperatu
0060  72 65 33 35 00  re35.

```

Figure 48 - MQTT Packet

## 6.3 Project domain model

Visual Paradigm Standard Edition (Instituto Superior de Engenharia do Porto)

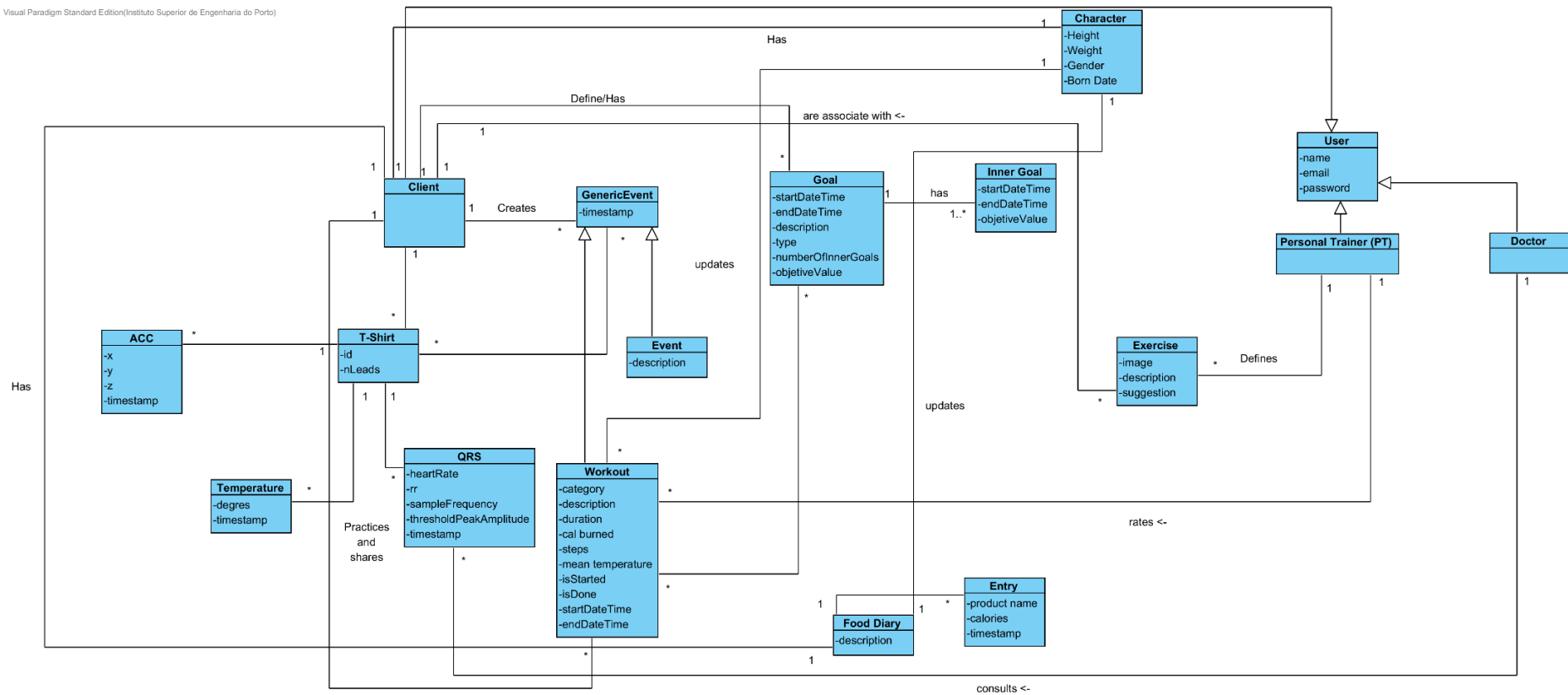


Figure 49 - Project domain model

## 6.4 First Simulated Vital Jacket MQTT Client

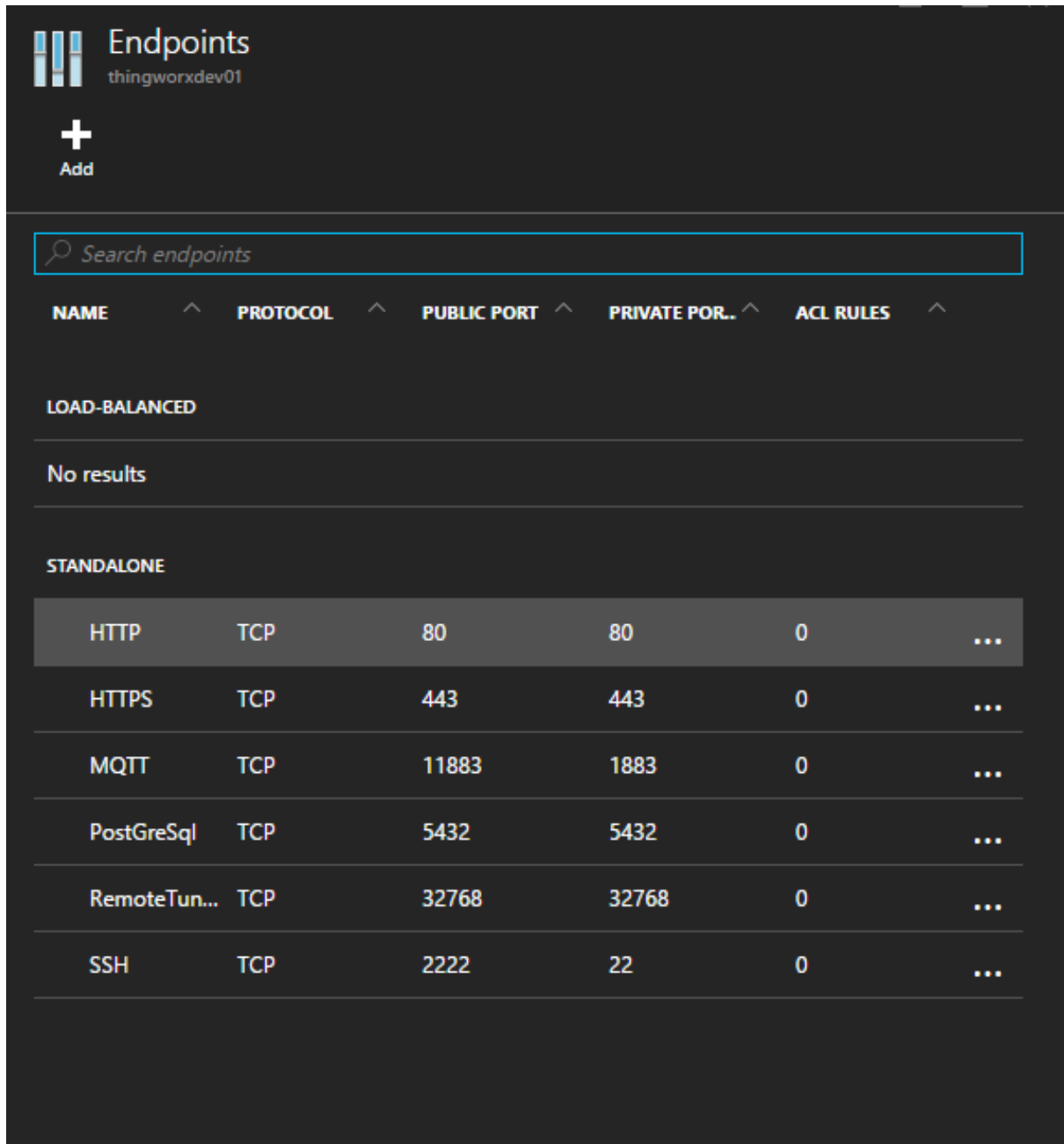
```

88
89 void publishHearthRate(MQTTClient client, int value){
115 }
116
117 void publishAccelerometer(MQTTClient client, char * axis, int value){
147 }
148
149 int main(int argc, char* argv[])
150 {
151     vitalJacketName = argv[1];
152     strcat(clientName, vitalJacketName);
153
154     MQTTClient client;
155     MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
156
157     MQTTClient_create(&client, ADDRESS, clientName,
158         MQTTCLIENT_PERSISTENCE_NONE, NULL);
159     conn_opts.keepAliveInterval = 20;
160     conn_opts.cleansession = 1;
161
162     if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
163     {
164         printf("Failed to connect, return code %d\n", rc);
165         exit(-1);
166     }
167
168     while(1){
169         publishTemperature(client, 2);
170         publishAccelerometer(client, "X", 2);
171         sleep(1);
172         publishAccelerometer(client, "Y", 2);
173         sleep(1);
174         publishAccelerometer(client, "Z", 2);
175         publishHearthRate(client, 2);
176         sleep(4);
177     }
178
179     MQTTClient_disconnect(client, 10000);
180     MQTTClient_destroy(&client);
181     return rc;
182
183 }

```

Figure 50 - First Simulated Vital Jacket MQTT Client

## 6.5 Deployment Server Endpoints



The screenshot shows the 'Endpoints' management page in Thingworx. At the top, there's a header with the 'Endpoints' title and a user identifier 'thingworxdev01'. Below the header is a '+ Add' button. A search bar labeled 'Search endpoints' is present. The main content area is divided into two sections: 'LOAD-BALANCED' and 'STANDALONE'. The 'LOAD-BALANCED' section shows 'No results'. The 'STANDALONE' section contains a table of endpoints.

NAME	PROTOCOL	PUBLIC PORT	PRIVATE POR..	ACL RULES	
LOAD-BALANCED					
No results					
STANDALONE					
HTTP	TCP	80	80	0	...
HTTPS	TCP	443	443	0	...
MQTT	TCP	11883	1883	0	...
PostGreSql	TCP	5432	5432	0	...
RemoteTun...	TCP	32768	32768	0	...
SSH	TCP	2222	22	0	...

Figure 51 - Deployment Server Endpoints



## 6.6 Project Scope

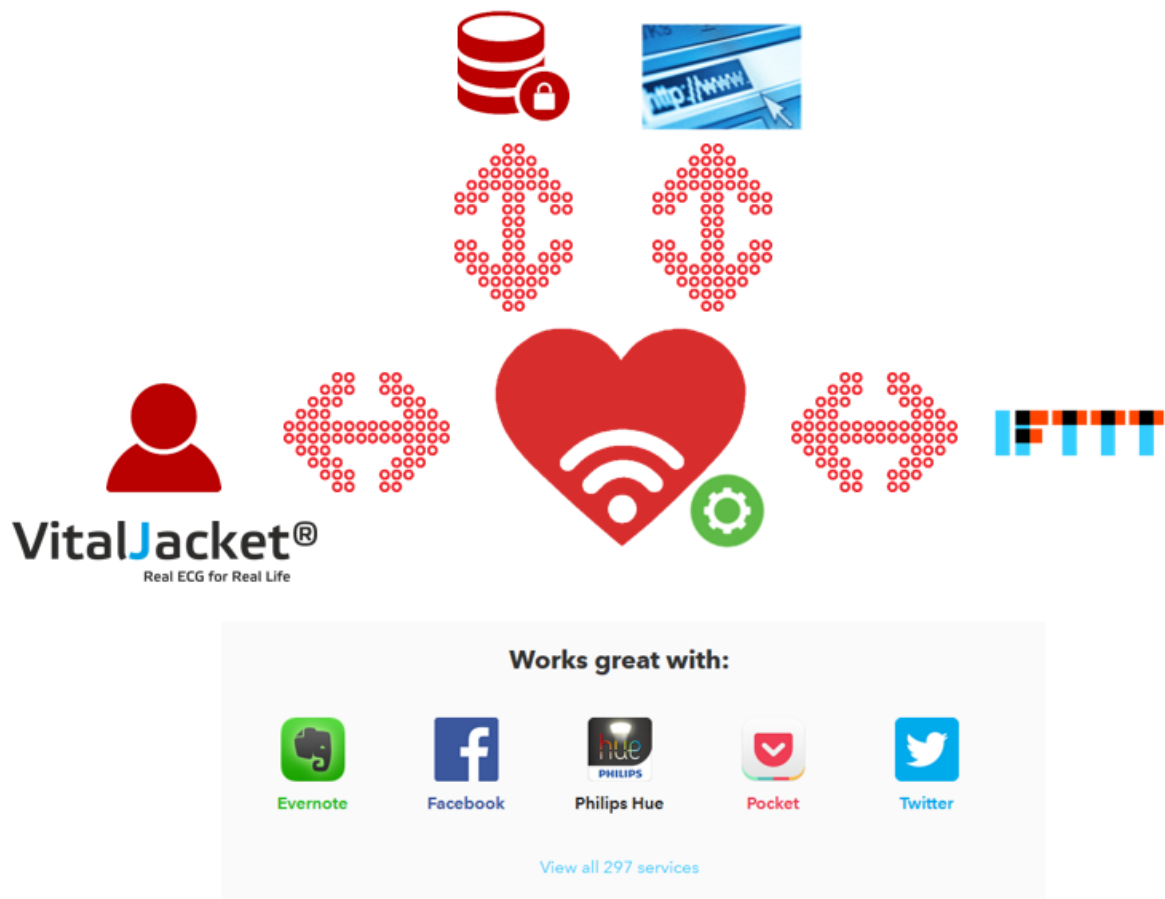


Figure 52 - Project Scope

## 6.7 Vital Jacket State Machine

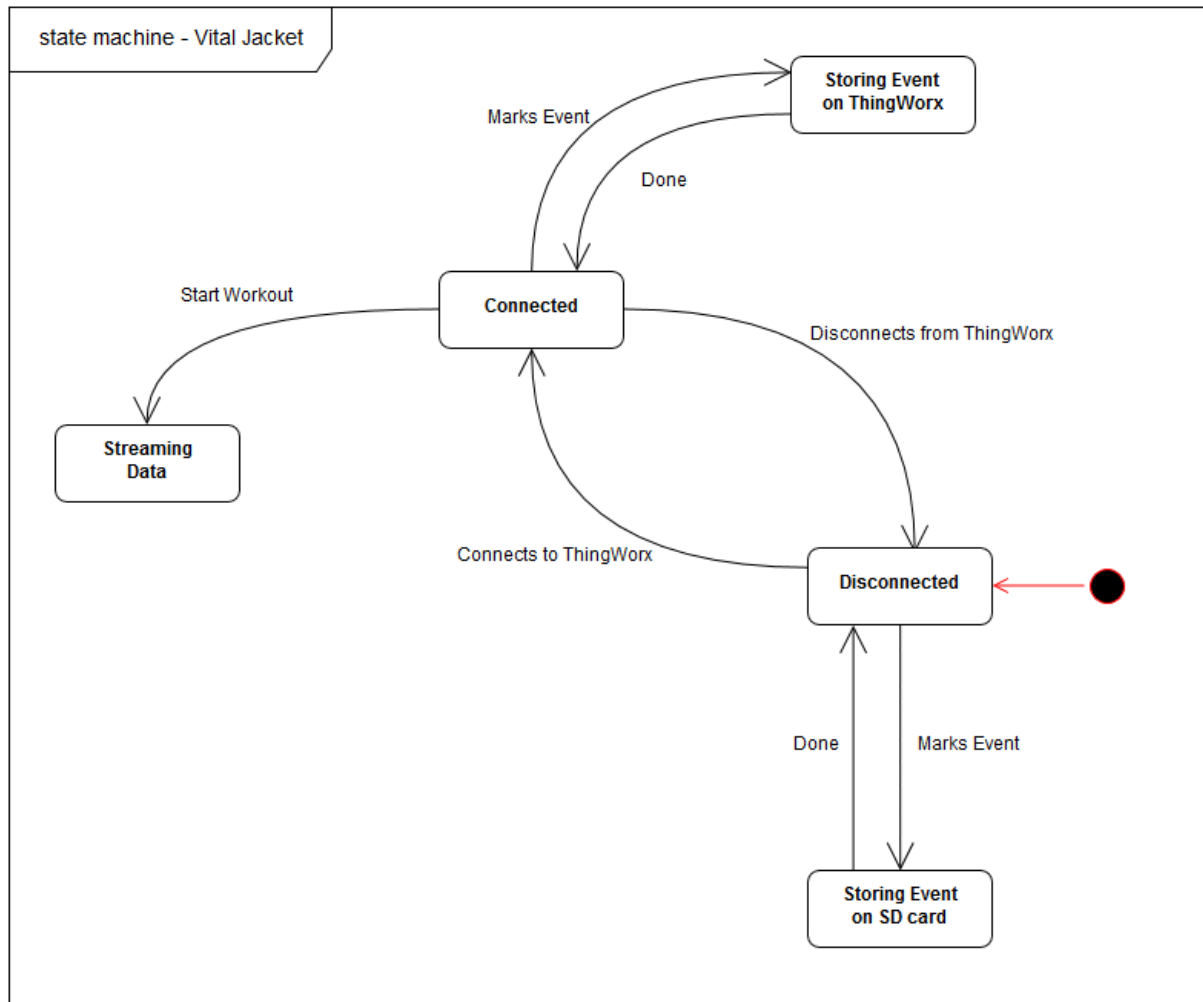
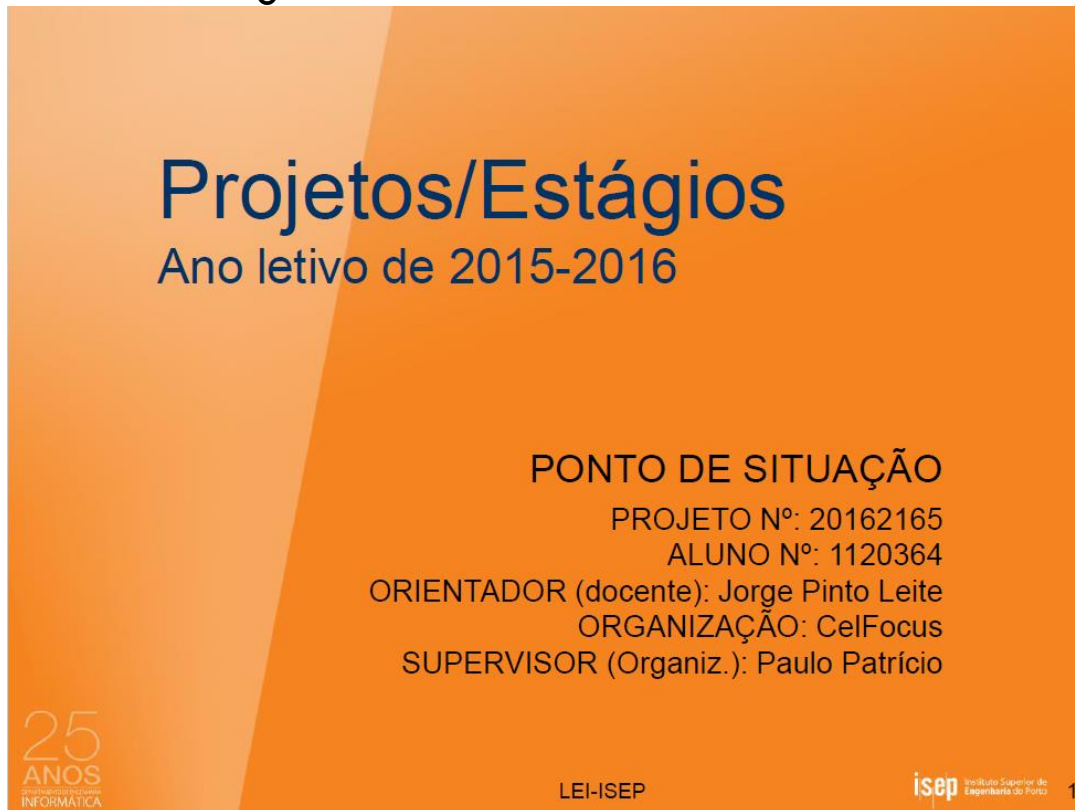


Figure 53 - Vital Jacket State Machine

## 6.8 State of Progress



**Projetos/Estágios**  
Ano letivo de 2015-2016

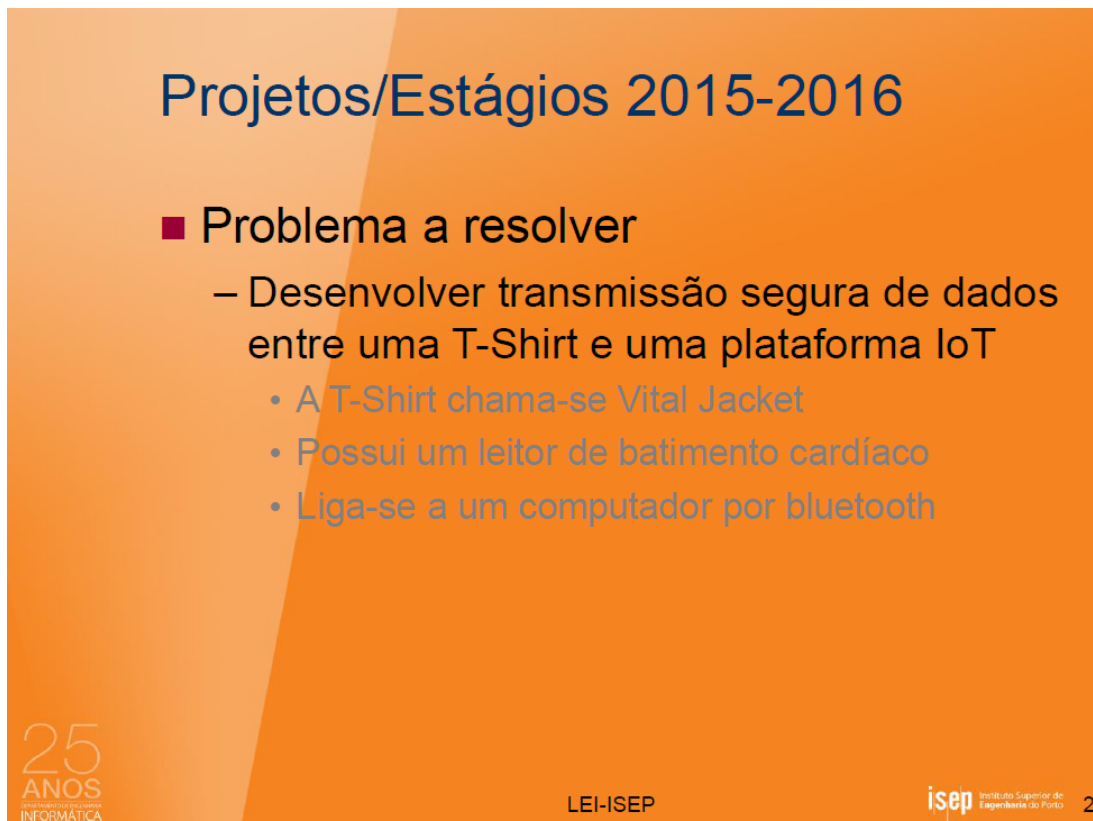
**PONTO DE SITUAÇÃO**  
PROJETO Nº: 20162165  
ALUNO Nº: 1120364  
ORIENTADOR (docente): Jorge Pinto Leite  
ORGANIZAÇÃO: CelFocus  
SUPERVISOR (Organiz.): Paulo Patrício

25 ANOS  
INFORMÁTICA

LEI-ISEP

isep Instituto Superior de Engenharia do Porto 1

Figure 54 - Page 1 of State of Progress



**Projetos/Estágios 2015-2016**

- **Problema a resolver**
  - Desenvolver transmissão segura de dados entre uma T-Shirt e uma plataforma IoT
    - A T-Shirt chama-se Vital Jacket
    - Possui um leitor de batimento cardíaco
    - Liga-se a um computador por bluetooth

25 ANOS  
INFORMÁTICA

LEI-ISEP

isep Instituto Superior de Engenharia do Porto 2

Figure 55 - Page 2 of State of Progress

## Projetos/Estágios 2015-2016

### ■ Solução pretendida

- Usar a plataforma ThingWorx em conjunto com protocolos de comunicação e encriptação
  - Plataforma escolhida por já ser implementada noutros projetos dentro da organização
    - Plataforma previamente usada
    - Ajuda na implementação disponível
- Implementação com IFTTT
  - Possibilidade de ligação com outros dispositivos

Figure 56 - Page 3 of State of Progress

## Projetos/Estágios 2015-2016

### ■ Análise e implementação da solução

- Análise efetuada
  - Pesquisa de outras plataformas de IoT disponíveis
  - Pesquisa sobre métodos de encriptação e comunicação seguros e aplicáveis no contexto de IoT
  - Pesquisa de possíveis aplicações a implementar

Figure 57 - Page 4 of State of Progress

## Projetos/Estágios 2015-2016

### ■ Análise e implementação da solução – Descrição da solução a implementar

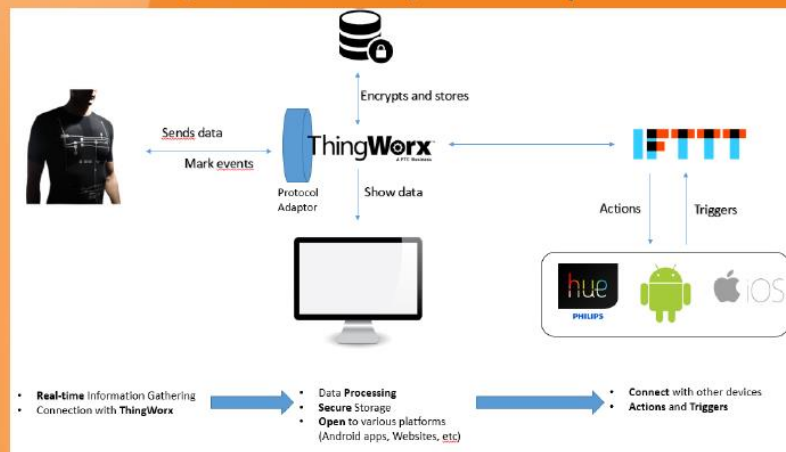


Figure 58 - Page 5 of State of Progress

## Projetos/Estágios 2015-2016

### ■ Execução da implementação

#### – Plano previsto para a implementação

- Implementação e teste do protocolo CoAP
- Implementação e teste do protocolo MQTT
- Implementação e teste dos métodos de encriptação AES e RSA
- Desenvolvimento de uma demonstração com a Vital Jacket ligada à ThingWorx

Figure 59 - Page 6 of State of Progress

## Projetos/Estágios 2015-2016

- Como está a decorrer
  - Estudo da plataforma ThingWorx feito (com formação da organização)
  - Alguns diagramas já desenhados
  - Implementação do CoAP a decorrer
- Problemas surgidos no projeto
  - Nenhum problema até ao momento

Figure 60 - Page 7 of State of Progress

## Projetos/Estágios 2015-2016

- Observações
  - Mudança de escritórios da empresa causou alterações no trabalho
    - Maior tempo de viagem
    - Melhores condições

Figure 61 - Page 8 of State of Progress