MASTER'S THESIS IN INDUSTRIAL ENGINEERING

ACADEMIC YEAR: 18/19

IMPLEMENTING AN INTEGRATED DEVELOPMENT ENVIRONMENT IN THE INTERNET OF THINGS GENERATION OF VISUAL PROGRAMMING TOOLS

AUTHOR: JOSÉ QUILES ALEMAÑ

SUPERVISED BY:
DIMITRIS KYRITSIS (EPFL)
PRODROMOS KOLYVAKIS (EPFL)
LUIS JOSÉ SAIZ ADALID (UPV)

Laboratory:
Information and Communication Technology for Sustainable Manufactory (ICT4SM)

ACKNOWLEDGEMENTS

"This Master Thesis will probably draw a line under my university life, an amazing journey in which I have had the opportunity to study in different cities such as Elche, Valencia, Prague or Lausanne. A stage in which of course I have worked hard, but especially, a stage in which I have had a really good time; meeting some of my greatest friends, getting to know different cultures, universities, parties and beers of the world, and learning, learning a lot, not only about engineering subjects but also about life. For these reasons, I just can be extremely thankful to all the people (professors, friends, family…) who have been part of this journey that made me better both professionally and personally.

In particular, I would like to thank Professor Dimitris Kyritsis for allowing me to develop this thesis in the ICT4SM group at EPFL, making me feel part of his exceptional team. I would also like to mention Prodromos Kolyvakis for his patience, close support and essential collaboration during the development of this project, and Professor Luis José Saiz for his attention, encouragement and helpful advices from Valencia.

Por último, quiero dar las gracias a mis abuelos, por haber sido siempre ejemplo de valores fundamentales como la humildad, el trabajo, la alegría o la familia. A mi hermana, por aguantar al pesado de su hermano y ser una mega-crack, no sabe qué hacer con su vida y aun así de lo único que estoy seguro es que haga lo que haga todo le va a ir bien. Y a mis padres, hoy y siempre, por los partiditos de fútbol en el pasillo de medio metro de ancho en Santa Pola, por esos platos espectaculares que solo vosotros podéis llegar a preparar (nótese la ironía), por todas las charlas y discusiones que habremos tenido, por estar ahí cuando yo he estado mal, por habérmelo dado todo… muchas gracias."

ABSTRACT

The present Master Thesis is directly aligned with the bIoTope project, funded by the European Union, which aims to build an Internet of Things (IoT) open innovation ecosystem that enables companies to easily create IoT systems with minimal investment.

In order to contribute to the consecution of this ambitious objective, in this thesis, an implementation of an Integrated Development Environment (IDE) in a widely used application for connecting smart objects and getting information from them, known as Node-RED, is carried out. To make this possible, it is conducted an in-depth study of the different concepts and technologies needed in the set of development stages: Definition, Building a Python Development Environment based on Skulpt and Implementation with Node-RED.

Once the implementation is completed, the new environment is used and tested in different case scenarios, as the Heat Wave Mitigation in Greater Lyon or a possible Smart Parking system for the FIFA World Cup of Qatar 2022, so that, it is possible to analyze the results and draw positive conclusions, because in spite of the existence of limitations, a new environment that facilitates users the creation of IoT systems is achieved, just as the bIoTope project lays its foundations.

Keywords: Internet of Things (IoT), Integrated Development Environment (IDE), bIoTope, Visual Programming, Node-RED, Industry 4.0, DevOps.

RESUMEN

El presente Trabajo Fin de Máster se encuentra directamente alineado con el proyecto bIoTope, financiado por la Unión Europea, que pretende construir un ecosistema abierto y de innovación donde las empresas puedan crear con mayor facilidad y menor inversión, sistemas que permitan extraer la información que nos proporciona el Internet de las Cosas (IoT).

A fin de colaborar en la consecución de este ambicioso objetivo, en este trabajo, se lleva a cabo el diseño y la implementación de un Entorno de Desarrollo Integrado (IDE) en una aplicación ampliamente utilizada para conectar objetos inteligentes y obtener información de ellos, conocida como Node-RED. Para hacer esto posible, se realiza un profundo estudio del estado del arte de los conceptos y tecnologías necesarias en las diferentes etapas del desarrollo: Definición, Creación del Entorno de Desarrollo basado en Skulpt e Implementación con Node-RED.

Una vez completada la implementación, ésta es utilizada y testada en distintos casos reales, como la mitigación de las olas de calor en Greater Lyon o un posible sistema de parking inteligente para el Mundial de Fútbol de 2022 en Qatar. De esta forma, un análisis de los resultados es realizado, obteniendo conclusiones positivas, puesto que a pesar de que existen limitaciones, se consigue desarrollar un nuevo entorno que facilita la creación de sistemas IoT, tal y como el proyecto bIoTope asienta sus bases.

Palabras Clave: Internet de las Cosas (IoT), Entorno de Desarrollo Integrado (IDE), bIoTope, Programación Visual, Node-RED, Industria 4.0, DevOps.

RESUM

El present Treball Fi de Màster, es troba directament alineat amb el projecte biotope, finançat per la Unió Europea, que pretén construir un ecosistema obert i d'innovació on les empreses puguen crear amb més facilitat i menor inversió, sistemes que permeten extreure la informació que ens proporciona l'Internet de les Coses (IoT).

Per tal de col·laborar amb la consecució d'aquest ambiciós objectiu, en aquest treball, s'ha portat a terme la implementació d'un Entorn de Desenvolupament Integrat (IDE) en una aplicació àmpliament utilitzada per connectar objectes intel·ligents i obtenir informació d'ells, coneguda com Node-RED. Per fer això possible, es realitza un profund estudi de l'estat de l'art dels conceptes i tecnologies necessàries en les diferents etapes: Definició, Creació del Entorn de Desenvolupament basat en Skulpt e Implementació amb Node-RED.

Completada la implementació, aquesta és utilitzada i testada en diferents casos reals, com la mitigació de les onades de calor a Greater Lyon o un possible sistema de pàrquing intel·ligent per al Mundial de Futbol de 2022 a Qatar, d'aquesta manera es pot realitzar una anàlisi dels resultats i treure conclusions positives, ja que tot i que existen limitacions, un nou entorn que permet crear fàcilment sistemes IoT és desenvolupat, tal i com el projecte biotope assenta les seues bases.

Paraules clau: Internet de les Coses (IoT), Entorn de Desenvolupament Integrat (IDE), bIoTope, Programació Visual, Node-RED, Indústria 4.0, DevOps.

GENERAL INDEX

Documents of the Master Thesis

Index of Report

Index of Budget

Index of Annex

Index of Figures

ABBREVIATIONS

A list of the different abbreviations used through the project is presented below:

- AJAX = Asynchronous JavaScript And XML
- API = Application Programming Interface
- BOM = Browser Object Model
- DB = Database
- CERN = Conseil Européen pour la Recherche Nucléaire
- CSS = Cascading Style Sheets
- DOM = Document Object Model
- DMT = Daily Mean Temperature
- EHF = Excess Heat Factor
- EU = European Union
- FIFA = Fédération Internationale de Football Association
- GSK = GlaxoSmithKline
- GUI = Graphical User Interface
- HTML = HyperText Markup Language
- IBM = International Business Machines
- ICT = Information and Communication Technology
- IDE = Integrated Development Environment
- IIoT = Industrial Internet of Things
- IoT = Internet of Things
- IT = Information Technology
- JS = JavaScript
- JSON = JavaScript Object Notation
- NCSA = National Center for Supercomputing Applications
- NoSQL = Not only Structured Query Language
- PHP = Hypertext Pre-Processor
- REPL = Read-Eval-Print Loop
- SCM = Source Code Management
- SQL = Structured Query Language
- TDP = Three Day Period
- UHI = Urban Heat Island
- VPL = Visual Programming Language
- XHTML = eXtensible HyperText Markup Language
- XML = eXtensible Markup Language

REPORT

IMPLEMENTING AN INTEGRATED DEVELOPMENT ENVIRONMENT IN THE INTERNET OF THINGS GENERATION OF VISUAL PROGRAMMING TOOLS

AUTHOR: JOSÉ QUILES ALEMAÑ

SUPERVISED BY:
DIMITRIS KYRITSIS (EPFL)
PRODROMOS KOLYVAKIS (EPFL)
LUIS JOSÉ SAIZ ADALID (UPV)

Laboratory:
Information and Communication Technology for Sustainable Manufactory (ICT4SM)

CHAPTER 1. INTRODUCTION

In this introductory chapter a complete definition of the project is pursued. In order to get a better understanding, a contextualization of the project is presented in the first place, then, the objectives are exposed with a brief description of the activities planned to meet them. Ultimately, the reasons and motivations that have driven the author to develop this particular project are described.

1.1. CONTEXT

This Master Thesis, according to the objectives that attempts to achieve, may be categorized in the denominated Internet of Things (IoT) framework.

One of the most representative definitions of the Internet of Things (and also the working definition for this thesis) was proposed by P. Guillemin and P. Friess (2009), conceptualizing the Internet of Things as a dynamic global network infrastructure that will be integrated into and act as an extension of the future internet, in which various “things” have unique identities, physical attributes, virtual personalities, and intelligent interfaces. Put differently, “the Internet of Things will allow people and things to be connected any time, any place, with anything and anyone, ideally using any path/network and any service” [1]. In spite of the fact, that the IoT concept is relatively new, it is claimed that the first IoT device was created long time ago, specifically in the year 1982. Necessity, as most of the times, was the mother of invention. In the early 1980s, David Nichols was a graduate student with an office in Carnegie Mellon University’s computer science department; he was really interested in Coke soda and usually run a long way to the building with the closest Coke machine. Nichols knew there was a good chance it would be empty or that, if the machine had recently been refilled, the sodas inside would be extremely warm. So one day, tired of walking long distances without the recompense of a cold Coke, he started to think about the idea of monitoring the state of the machine from his office. Nichols wrote a few friends about this idea to track the machine’s contents remotely and taking advantage of the lights already installed on the machine, put an end to the unsatisfying soda runs once and for all. The final program, could inform anybody connected to the university local Ethernet if there were any Cokes in the machine, and, if so, which ones were cold (more than three hours since restocking) [2], [3].

Now, after years of technological advances that have facilitated the extraction of valuable information from everyday devices, just as the Coke Machine ideated by Nichols, has led the Internet of Things to be recognized as one of the most exciting and key opportunities for both academia and industry. For example, in the Global McKinsey Institute Report (2015), *Unlocking the potential of the Internet of Things,* it was concluded that the Internet of Things offers a potential economic impact of four to eleven trillion dollars a year in 2025, distributed in such a different fields as predictive maintenance or monitoring and managing illness [4]. Regarding academia, Miorandi et al. (2012) in the article: *Internet of things: Vision, Applications and Research Challenges*, summarize the IoT technical perspective as follows: "The term Internet-of-Things is used as an umbrella keyword for covering various aspects related to the extension of the Internet and the Web into the physical realm, by means of the widespread deployment of spatially distributed devices with embedded identification, sensing and/or actuation capabilities." [5].

Such is the importance of the Internet of Things, that one of the challenges inside the *Leadership in Enabling and Industrial Technologies* objective, for the European Union (EU) project Horizon 2020 ("the biggest EU Research and Innovation programme ever with nearly €80 billion of funding available over seven years, from 2014 to 2020", whose goal is to "ensure Europe produces world-class science, removes barriers to innovation and makes it easier for the public and private sectors to work together in delivering innovation, emphasizing on excellent science, industrial leadership and tackling societal challenges" [6]), is directly the *Internet of Things and Platforms for Connected Smart Objects*, which is identified as one of the next big concepts to support societal changes and economic growth at an annual rate estimated at 20% [7]. The scope of this challenge is to "create ecosystems of Platforms for Connected Smart Objects, integrating the future generations of devices, embedded systems and network technologies and other evolving ICT advances. These environments support citizen and businesses for a multiplicity of novel applications. They embed effective and efficient security and privacy mechanisms into devices, architectures, service and network platforms, including characteristics such as openness, dynamic expandability, interoperability, dependability, cognitive capabilities and distributed decision making, cost and energy-efficiency, ergonomic and user-friendliness. Such Smart Environments may be enriched through the deployment of wearable/ambulatory hardware to promote seamless environments" [7].

However, in order to fulfil their aspiration, the projects funded will face difficulties that need to be overcome. For example, moving from vertically-oriented closed systems, architectures and application areas towards open systems and platforms that support multiple applications [7], or security issues, according to the Harvard Business Review, *The Internet of Things Is Going to Change Everything About Cybersecurity*, the number of unmanaged devices being introduced onto networks daily is increasing by orders of magnitude, being predicted that there will be 20 billion in use by 2020. Traditional security solutions will not be effective in addressing these devices or in protecting them from hackers, attacks on IoT devices were up 280% in the first part of 2017. In fact, it is anticipated that a third of all attacks will target shadow IT and IoT by 2020 [8].

One of those projects that fight against these obstacles is the bIoTope project, (Building an IoT OPen innovation Ecosystem for connected smart objects) [9]. bIoTope capabilities lay the foundation for open innovation ecosystems where companies can innovate both by the creation of new software components for IoT ecosystems, as well as create new Platforms for Connected Smart Objects with minimal investment. Large-scale pilots implemented in smart cities will provide social, technical and business proofs-of-concept for such IoT ecosystems

[10]. The present Master Thesis is directly aligned with the bIoTope project, since it main objective will focus on providing an open environment that facilitates companies the creation of new Platforms for Connected Smart Objects.

Currently, different open environments and applications that allow the creation of IoT systems, connecting smart objects and getting information from them, exist. One of the most recognized is known as Node-RED, in fact Michelle L. Corbin (2016), presented Node-RED as "the fundamental open-source programming tool for IoT", highlighting its ability to integrate all kind of services and devices, the possibility to be run at the edge of the network or in the cloud or its easy-to-use flow-based/visual programming environment [11].

Visual programming is gaining more and more importance in the Internet of Things domain, as P. Pratim Ray (2016) exposed in his review article: *A Survey on Visual Programming Languages in Internet of Things*, the shift from traditional programming ways to visual programming may be directly associated to the easy visualization of the programming logic (e.g., flow chart), the benefits for naïve users to get associated with the concept of interactions among the logical structures, its finality for rapid development of IoT products, its better handling of the "syntax error," or its portability on a tablet (or hand held smart phone/device) or in situations where no physical keyboard is present [12].

Despite all the advantages that visual programming entails, there are some challenges that need to be identified, P. Pratim Ray (2016) underlines the extensibility (limitation of operations, impossibility to perform precise edge cases), the slow code generation, the integration (existence of development environments that leverages ease of programming), the standard model (having the ability to solve different kind of problems, since IoT may be applied to different topics) and the user interface (adapted to the programming knowledge of their users) [12].

Through this Thesis, issues related with the extensibility and especially the integration of the Node-RED environment will be tackled. Despite Node-RED originally provided only the user with the possibility to write JavaScript functions, a new Python module for Node-RED has recently been created. This new module results particularly interesting since a vast increase of the knowledge extraction libraries that offer an API for the Python language starts to be directly accessible from the Node-RED environment.

The Python module increases the extensibility of Node-RED, providing new functionalities and capabilities to the program; however, its development environment results insufficient to develop and deploy complex IoT applications or at least shows huge potential for improvement, since it is not directly adapted to the Python language, being unable to identify for example where an error is committed, making even more difficult the task of developing IoT systems and applications.

The next figure (Figure I) aims to summarize the contents of the present subchapter.

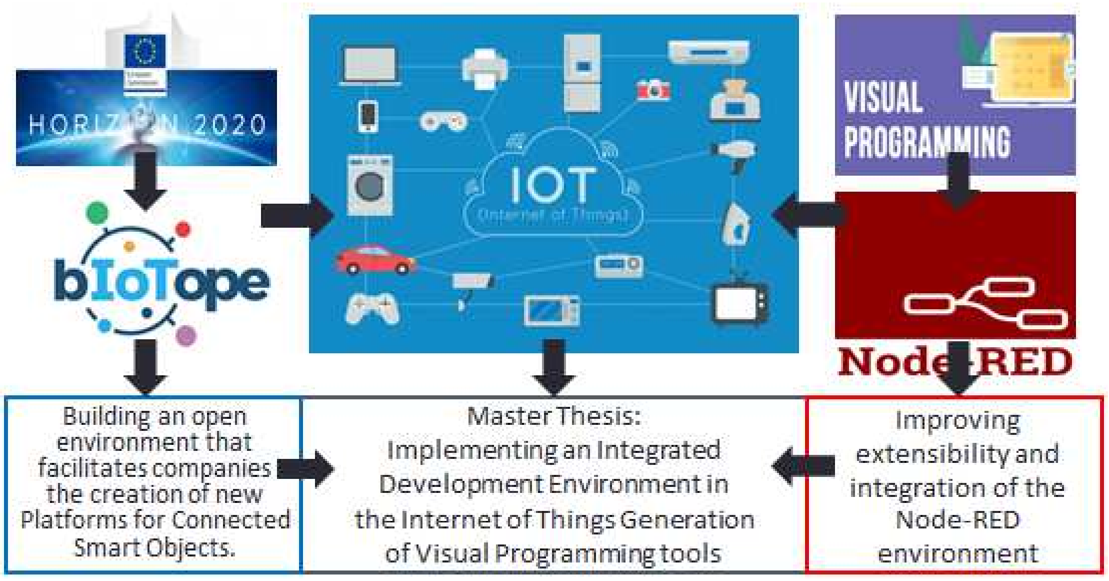

HORIZON 2020
bIoTope
IOT
(Internet of Things)
VISUAL PROGRAMMING
Node-RED
Building an open environment that facilitates companies the creation of new Platforms for Connected Smart Objects.
Master Thesis: Implementing an Integrated Development Environment in the Internet of Things Generation of Visual Programming tools
Improving extensibility and integration of the Node-RED environment

Figure I. Summary of the Context

1.2. OBJECTIVES

The fundamental goal of this project is to develop and implement an Integrated Development Environment (IDE) that facilitates developers and companies the creation of more complicated IoT applications in Node-RED. As it has been previously introduced, this goal is directly aligned with the bIoTope project since it is focused on providing an open environment that facilitates the creation of new Platforms for Connected Smart Objects. This new environment should help users and programmers to write Python code and functions with more productivity. The IDE will provide an interactive editor that shows line by line if the user is making any mistake in the code, afterwards, the code may be run to see if the expected results are obtained; moreover, a debugger tool will be present, along with different functionalities that allow the user to load external files or to save the new code to use it directly in Node-RED.

A set of secondary objectives, whose completion will bring the consecution of the main goal a step closer, are also proposed:

1) Good definition of the project: Getting an overall understanding about what this project attempts to achieve and how it is actually possible to do it.
2) Understanding which technologies are needed for the development, their fundamentals and what are their current applications.
3) Developing application that meets requirements.
4) Correct implementation with Node-RED, establishing a connection between the developed environment and Node-RED.
5) Testing in representative case scenarios.
6) Getting insights and conclusions about the utility and effectiveness of the implementation.

To meet each one of these objectives, a different group of actions are planned:

1) The right definition of the project is actually the essence of the introduction chapter of this document, the actions needed consist in reading articles and bibliography about the IoT framework, defining the existing limitations and obstacles to create new platforms for connecting Smart Objects, defining an objective that actually helps to reduce the existing limitations, planning a set of actions, whose completion assure meeting the main goal, and last but not least, documenting the resultant planning.
2) Regarding the second objective, the actions proposed are: defining requirements and tasks of the development phase (*4.1 Stage 1: Definition*), expressing with which technologies the requirements are met, conducting a study of these technologies by reading professional literature and official documentation, emphasizing in their fundamentals, their advantages and their current applications (*3. Technologies*)
3) In order to develop an environment that meets the requirements, some activities must be performed as defining a methodology, the requirements and tasks of the development phase (*4.1. Stage 1: Definition*), making use and writing new code that adds all the functionalities required for the environment or solving errors and adding improvements to the environment (*4.2. Stage 2: Building a Python Development Environment based on Skulpt*).
4) The actions that are needed to achieve a correct implementation with Node-RED and establishing a connection between the developed environment and Node-RED are dockerizing the new environment as a service (*4.2. Stage 2: Building a Python Development Environment based on Skulpt*), dockerizing the Node-RED environment with its last improvements (*4.3. Stage 3: Implementing with Node-RED*), integrating the environment as a unique service with the use of docker volumes to provide the connection.
5) The set of actions directly related with testing in representative cases are expanded on *Chapter 5: Application Case Scenarios*, but they consist basically in research real application possibilities, selecting and justifying two cases that represent a good test for the IDE, defining problems and possible solutions for each case, developing an IoT solution for each case, expressing the advantages and limitations of the new environment.
6) To draw conclusions, the advantages and limitations of the new environment are analyzed through the application cases, the expected results defined in the first chapter are compared with the real ones, the possible areas of future improvement are studied, and a final assessment of the work must be made.

Note: The actions described above do not include all the actions performed during the project, just the most important ones that may be directly related with the consecution of one of the objectives.

1.3. MOTIVATION

As a future industrial engineer, the development of this Master Thesis represents not only a huge challenge to apply the knowledge acquired during my studies in different courses such as Project Management, Industrial Informatics Technology, Instrumentation and Control Systems… but also a unique opportunity to keep learning and improving in topics that may be important during my professional career.

The Internet of Things and more specifically the Industrial Internet of Things (IIoT), which is defined by H. Boyes et al. (2018) in the article: *The Industrial Internet of Things (IIoT): An Analysis Framework*, as “a system comprising networked smart objects, cyber-physical assets, associated generic information technologies and optional cloud or edge computing platforms, which enable real-time, intelligent, and autonomous access, collection, analysis, communications, and exchange of process, product and/or service information, within the industrial environment, so as to optimize overall production value. This value may include; improving product or service delivery, boosting productivity, reducing labor costs, reducing energy consumption, and reducing the build-to-order cycle” [13], represents a new domain of action for industrial engineers, in fact it is claimed by P. Gerbert , M. Lorenz and M. Rüßmann (2015) that it constitutes one of the pillars of the fourth industrial revolution, the Industry 4.0 [14].

“The First Industrial Revolution used water and steam power to mechanize production. The Second used electric power to create mass production. The Third used electronics and information technology to automate production. Now a Fourth Industrial Revolution is building on the Third, the digital revolution that has been occurring since the middle of the last century. It is characterized by a fusion of technologies that is blurring the lines between the physical, digital, and biological spheres” [15], this sentence of Klaus Schwab (2016), Founder and Executive Chairman of the World Economic Forum, in his article: *The Fourth Industrial Revolution: what it means, how to respond*, summarizes how the industry is continuously evolving, which may also be observed in **Figure II.**

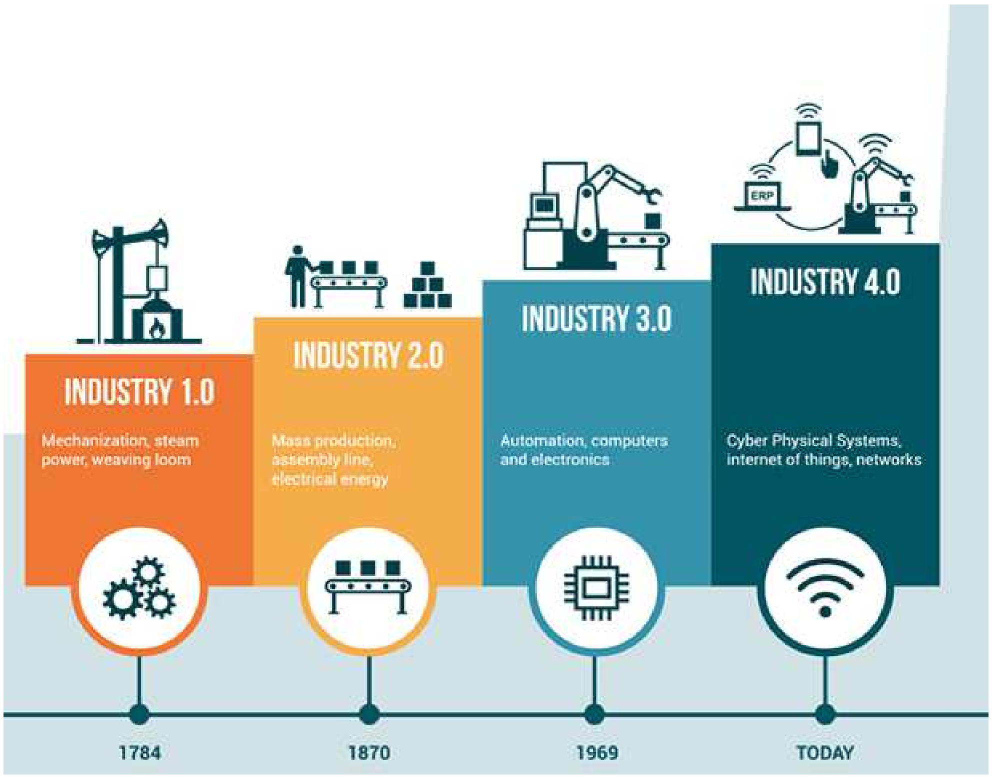

INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
ERP
1784
1870
1969
TODAY

Figure II. Industry Revolutions
Source: SenteGroup

In essence, the Industry 4.0 consists in the integration between manufacturing operations systems and information and communication technologies (ICT) – especially the Internet of Things (IoT) [16]. This new paradigm of the Industry 4.0 has the power to transform isolated and optimized cells production into a fully integrated, automated, and optimized production flow. This leads to greater efficiency and change in traditional production relationships among suppliers, producers, and customers as well as between human and machine [14], or the potential to raise global income levels and improve the quality of life for populations around the world. To date, technology has made possible new products and services that increase the efficiency and pleasure of our personal lives [15].

Furthermore, apart from the learning experience in such a future-oriented field that it entails, this project represents the opportunity to collaborate on a higher-purpose goal; by the consecution of the objectives defined, a contribution to the programmers and developers of IoT systems would be made, facilitating and improving the productivity of the programming experience in a widely used application as Node-RED.

CHAPTER 2. STATE OF THE ART

In order to provide the reader with all the knowledge necessary to understand the situation and the concepts of the project, in this chapter an in-depth study of the state of the art is presented.

The study is focused on Visual Programming Language and its applications including latest Node-RED improvements, current Integrated Development Environments, Interpreters that allow running Python code in the browser, Databases and their particular requirements in the Internet of Things framework, and DevOps as a Total Quality Measure for development of new applications.

2.1. VISUAL PROGRAMMING LANGUAGE (VPL)

"Visual programming is programming in which more than one dimension is used to convey semantics. Examples of such additional dimensions are the use of multidimensional objects, the use of spatial relationships, or the use of the time dimension to specify ''before–after'' semantic relationships. Each potentially significant multidimensional object or relationship can be regarded as a token (just as in traditional textual programming languages each word is a token) and the collection of one or more such tokens is a visual expression… When a programming language's (semantically significant) syntax includes visual expressions, the programming language is a visual programming language (VPL)" [17], M. Burnett (1999), *Visual Programming*, *Wiley Encyclopedia of Electrical and Electronics Engineering*.

A layman's explanation of the purpose of Visual Programming is given by Matthew Revell (2017), he affirms that "Visual programming lets humans describe processes using illustration. Whereas a typical text-based programming language makes the programmer think like a computer, a visual programming language lets the programmer describe the process in terms that make sense to humans" [18]. It is important to understand that visual programming does not represent a unique way to describe the "process" but a set of different techniques and paradigms such as [17]:

1) **Imperative Visual Programming by Demonstration**: the programmer demonstrates the desired actions.
2) **Form/Spreadsheet Based Visual Programming**: A programmer programs by creating a form and specifying its contents.
3) **Dataflow Visual Programming**: The dataflow paradigm is currently the approach to visual programming used most widely in industry; Dataflow paradigm distinguishes itself through its explicitness about the dataflow relationships in the program.

4) **Rule-Based Visual Programming**: the programmer specifies the rules by demonstrating a post-condition on a pre-condition.

VPLs are currently used for educational, multimedia, video games, system development/simulation, automation, and data warehousing/business analytics purposes [12]. A recent survey conducted by Partha Pratim Ray (2016) presents the percentagewise distribution of VPLs in the illustrated domains (Figure III). Out of 89 differently surveyed VPLs, system simulation and multimedia hold 60% of the market, marking 35% and 25%, respectively.

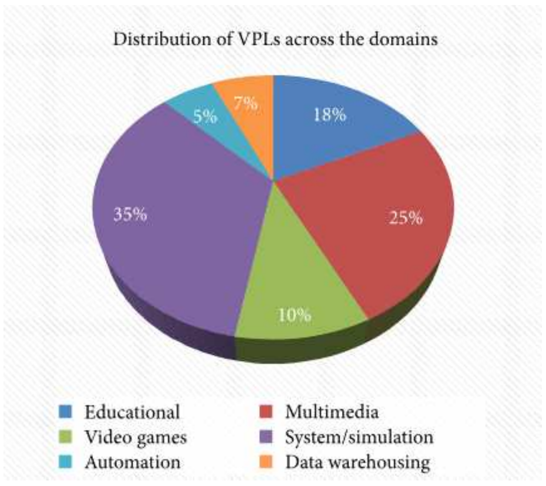


Figure III. Distribution of VPLs across domains
Source: P. Pratim Ray,2016 [12]

Visual Programming is particularly interesting in the IoT context due to the easy visualization of the programming logic (e.g., flow chart), the benefits for naïve users to get associated with the concept of interactions among the logical structures, its finality for rapid development of IoT products, its better handling of the "syntax error," or its portability on a tablet (or hand held smart phone/device) or in situations where no physical keyboard is present [12]. The presence of these advantages has promoted the creation of new IoT environments based on visual programming as "Visuino" (Using blocks to program Arduino boards. It is based on the drag-and-drop paradigm that is used to control sensors and peripherals [19]), "Wia" (cloud platform that simplifies the IoT app development connecting devices together and with external services [20]) or "Node-RED" (*Chapter 2.2.1. Node-RED*). However, they still have to overcome great challenges as the ones identified by Partha Pratim Ray (2016) [12]:

1) **Extensibility**. VPLs allow developer/user to perform a limited set of operations (things) easily, but precise edge cases are too far difficult (even impossible) to achieve in practice. These VPLs should give user more power, instead of constricting. This might give an opportunity for the IoT enabled application development process where extensibility is a fundamental need.
2) **Slow Code Generation**. Performance diagnosis is a key part of any developer's testing phase, especially in IoT where lots of devices are engaged into the system. But, in case of

VPLs, it seems that they work on leaky abstractions, resulting in slow code generation which is nearly impossible to optimize by a developer.

3) **Integration**. Developers live in Integrated Development Environments (IDEs) and simulators world. If the IDEs and simulators are poor in effort and performance, the whole system results prejudice, decreasing developers productivity.
4) **Standard Model**. This is another serious challenge in VPLs of IoT; Different service professionals like scientists, electrical engineers, IT industry programmers, and so on are taught and trained how to model the problem statement in different manner. There is a strong need of a standard modular structure so that a well-trained developer would be able to build any sort of applications on different genre of platforms.
5) **User Interface**. VPLs in IoT are well modeled by appropriate data-flows between self-contained logical boxes that essentially mimic the physical world.

By the development and implementation of an Integrated Development Environment with Node-RED, issues related with the extensibility and especially the integration of this program will be tackled.

2.1.1. Node-RED

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways [21]. Node-RED is a dataflow visual programming tool based on the concept of "nodes" that are black box elements that perform specific tasks. The data flows through the nodes according to the node's connections [22].

Node-RED started life in early 2013 as a side-project by Nick O'Leary and Dave Conway-Jones of IBM's Emerging Technology Services group. It was open-sourced in September 2013 and has been developed in the open ever since, culminating in it being one of the founding projects of the JS Foundation in October 2016 [22]. The decision to be built on Node.js and developed in the open has lead Node-RED to have over 225,000 modules in Node's package repository, being easy to extend the range of palette nodes to add new capabilities [21].

One of the modules recently created (2017), that is particularly relevant for the development of this project is the Python module [23]. Since the beginning of the Node-RED project, JavaScript functions could be created within the editor using a rich text editor, however a vast number of knowledge extraction libraries that offer an API for the Python language were not directly accessible from the Node-RED environment, and of course, in IoT systems that manage large amounts of data is interesting to make use of those libraries. The Python module solved the problem, allowing the user to create Python functions inside a node in the flow. However, it automatically generated a new area of improvement, since the development environment of these functions does not provide a way to detect in which line an error is made or what is the actual reason of the error, being frustrating to develop complex applications that are prone to errors.

The new integrated development environment, just as was established in the objectives subchapter (*1.2 Objectives*), should facilitate users the creation of complex IoT systems using the new Python module (extensibility), programming with more productivity (integration).

2.2. INTEGRATED DEVELOPMENT ENVIRONMENTS (IDEs)

An Integrated Development Environment is defined by M. Kerrigan, et al. (2007) as: “a type of computer software that assists computer programmers to develop software. The main aim of an IDE is to improve the productivity of the developer by seamlessly integrating tools for tasks like editing, file management, compilation, debugging and execution” [24]. In the same article it is underlined the need for adequate tool support for working with semantic technologies and how IDEs like the Eclipse Java Development Toolkit have proven that good tool support can improve the productivity of engineers [24].

There are a huge variety of different IDEs. There are IDEs that are designed to work with one specific language, cloud-based IDEs, IDEs customized for the development of mobile applications or for HTML, and IDEs meant specifically for Apple or Microsoft development [25]:

1) **Multi-Language IDEs**
   - Eclipse: Supports C, C++, Python, Perl, PHP, Java, Ruby and more. This free and open source editor is the model for many development frameworks. Eclipse began as a Java development environment and has expanded through plugins. Eclipse is managed and directed by the Eclipse.org Consortium.
   - NetBeans: Supports Java, JavaScript, PHP, Python, Ruby, C, C++ and more. This option is also free and open source. All the functions of the IDE are provided by modules that each provide a well-defined function. Support for other programming languages can be added by installing additional modules.
2) **IDEs for Mobile Development**
   - There are IDEs specifically for mobile development, including PhoneGap and Appcelerator's Titanium Mobile.
3) **HTML IDEs**
   - Some of the most popular IDEs are those for developing HTML applications. For example, IDEs such as HomeSite, DreamWeaver or FrontPage automate many tasks involved in web site development.
4) **Cloud-Based IDEs**
   - Cloud IDEs give developers access to their code from anywhere. For example, Nitrous is a cloud-based development environment platform that supports Ruby, Python, Node.js and more.
5) **IDEs Specific to Microsoft or Apple**
   - Visual Studio: Supports Visual C++, VB.NET, C#, F# and others. Visual Studio is Microsoft's IDE and is designed to create applications for the Microsoft platform.
   - Xcode: Supports the Objective-C and Swift languages, and Cocoa and Cocoa Touch APIs. This IDE is just for creating iOS and Mac applications and includes an iPhone/iPad simulator and GUI builder.
6) **IDEs for Specific Languages**
   - Some IDEs cater to developers working in a single language. These include CodeLite and C-Free for C/C++, Jikes and Jcreator for Java, Idle for Python, and RubyMine for Ruby/Rails.

The IDE proposed for development, pursues an improvement in the programming experience of the new Python module of Node-RED. Because of that reason, the new IDE will be categorized as Cloud-based (the code needs to be on the cloud in order to be accessible to Node-RED) IDE for Specific Language (Python).

Regarding the functionalities, an IDE can include all kind of features with the objective of improving the developer's productivity such us code editors, debuggers (tool used during testing to help debug application programs), compilers (tool that translates the whole program into machine code before the program is run), interpreters (tool that translates code into machine code, instruction by instruction, being easier to debug and detect the error than in a compiler), build automation tools (tools to automate common developer tasks) ... In our case, a code editor, an interactive code editor (editor that shows if an error appears after writing each line), a debugger, an output screen that allows the user to visualize if he receives the expected result after running the code, and one interpreter will be present on the IDE.

2.3. INTERPRETERS

In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program [26]. Interpreters are usually confused with compilers since both are language translators, both translate the programs from a source language that are in human readable form into an equivalent program in an object language, usually a machine language. However, they present significant differences [27]:

1) **Input**: While a compiler takes the entire program at a time, the interpreter takes only one line at a time.
2) **Speed**: Compilers are comparatively faster than interpreters.
3) **Memory**: Interpreters require less memory since they do not create intermediate object code.
4) **Errors**: Interpreters display errors line by line, being easier to detect them than in the compiler case, where all the errors of the program are displayed at the same time.
5) **Programing Languages**: C++, C or Scala requires the use of compilers and PHP, Python or Java normally use interpreters.

As the IDE that is decided to be developed requires a Python implementation in browser to be accessible to Node-RED, an interpreter that translates the Python code to a language as JavaScript that can be run in the browser is needed.

Currently there are different options that may be used to achieve this goal:

1) **Brython:** Brython is an open source project, which main goal is to replace JavaScript with Python, as the scripting language for web browsers. In order for the Python script to be processed, "it is necessary to include brython.js and to run the brython() function upon page load (using the onload attribute of the <BODY> tag). While in the development phase, it is possible to pass an argument to the brython() function: 1 to have the error messages displayed to the web browser console, 2 to also get the JavaScript code

displayed along with the error" [28]. The biggest disadvantage of Brython is actually its objective or general purpose, Brython is not created to write Python code in web in an efficient and user-friendly environment, it is developed to be able to use the Python code as a substitute of JavaScript to develop webpages.

2) **Repl**: Repl represents a project that pursues to make programming more accessible. Building powerful yet simple tools and platforms for educators, learners, and developers [29]. Basically, is an online Read-Eval-Print Loop environment that takes single user inputs (i.e., single expressions), evaluates them, and returns the result to the user for different programming languages including Python. Repl presents great advantages since provides quite feedback when a mistake is made, but the fact that is not completely open sourced limits its applicability to this thesis.
3) **Pypy.js**: Experiment in building a fast and compliant python environment for the web using the PyPy python interpreter [30]. Pypy.js represents a possible Python interpreter for the project since it is totally open sourced and provides a code editor that allows the user to write its Python code and visualize the results. Some points that discourage the use of Pypy.js are the absence of a debugger, the impossibility to import some modules and especially the state of the project that is currently (November 2018) in pause [31].
4) **Skulpt:** Skulpt is an entirely in-browser implementation of Python. Its main objective is providing the possibility to use Python on both the server and on the client and sharing code between them, creating a translation from Python to JavaScript directly in the browser so that there isn't a pre-compile step required to improve iteration time [32]. Skulpt represents a great opportunity since it is a completely open source project with a lot of documentation, providing not only a code editor, but also an interactive screen that displays errors line by line, or a debugger; the biggest limitation is that some python modules are not implemented yet, but in contrast with the Pypy.js, the Skulpt project is very active, providing improvements and new modules implementations with high frequency.
5) **Transcrypt:** Transcrypt is a tool to compile a fairly extensive subset of Python into compact, readable JavaScript. It has the following characteristics: It's lightweight, it offers the semantic essence of Python along with the clean syntax that the language is famous for, it cooperates flawlessly with the JavaScript universe of web-oriented libraries, it translates Python into highly readable, easy to debug JavaScript [33]. The problem with Transcrypt is similar to Brython, despite it provides a possible translation from Python to JavaScript, its mission is using the Python code as a substitute of JavaScript to develop webpages, not to help the user to write and run Python code in the browser.

Taking into account all these considerations and the requirements demanded by the IDE, it is decided to create the Integrated Development Environment based on the Skulpt interpreter.

2.3.1. Skulpt

Skulpt is idea of Scott Graham, but Brad Miller has been shepherding the development since 2010 [34]. Currently (November 2018) Skulpt project has more than 2800 commits (the last commit dated in October 2018) distributed in 23 branches and 63 different contributors [34]. This data gives us an idea about the level of commitment of developers and contributors with the project.

Skulpt is already implemented in different projects with success:

1) **How to Think like a Computer Scientist**: Interactive Book that as its name indicates try to teach how computer scientist thinks, using an interactive editor for Python based on Skulpt [35].
2) **Trinket:** Learning webpage of Python with different features for students and schools. Trinket lets you run and write code in any browser, on any device, thanks to the Skulpt based interface [36].
3) **CodeSkulptor:** Integrated Development Environment based on Skulpt consisting basically of a Code Editor [37].

As it has been previously discussed, the biggest limitation of Skulpt, from the point of view of this project, is the impossibility to import some Python modules; however, this limitation is considered by Skulpt developers and contributors, who are actually working in its solution. In fact, different modules have been recently developed as the numpy module (April 2018), the pygal module (October 2018), or mathplotlib (October 2018) by Trinket developers [38].

In conclusion, and considering the existence of limitations, which are expected to be reduced, the possibility to write Python code in the browser due to its translation mechanism to JavaScript, the presence of different tools as code editors, debugger… and the existence of other projects who have already implemented it, make Skulpt the option chosen to develop the IDE, object of this Master Thesis.

2.4. DATABASES

"One of the most important outcomes of the IoT field is the creation of an unprecedented amount of data. Storage, ownership and expiry of the data become critical issues… The data have to be stored and used intelligently for smart monitoring and actuation" [39]. This fragment of the article: *Internet of Things (IoT): A vision, architectural elements, and future directions* written by J.Gubby, et al. (2013), represents in a clearly manner the importance of data storage in the Internet of Things Generation. Hence databases, defined as structured set of data held in computer storage and typically accessed or manipulated by means of specialized software [40], are expected to have a major role in this era.

Some of the challenges that the databases will have to overcome in the IoT generation, are identified by J. Cooper and A. James (2009) in the article: *Challenges for database management in the internet of things*, such as **Size and Scale** (high volumes of data), **Heterogeneity** (Different structures), **Speed of transactions** (High speed requirements) or **Data Protection** (Personal Information needs to stay secure) [41].

Databases are often divided in relational databases based on SQL (Structured Query Language) and NoSQL databases that provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL databases are gaining greater presence to storage the IoT data due to the heterogeneity and speed requirements, SQL databases performs worse when the data is not "structured" (regularly segmented portions designed to fit into particular [schema] columns, slots and repositories) [42].

A comparison of the performance of SQL and NoSQL databases is made by Y. Li and S. Manoharan (2013), their objective was to “independently investigate the performance of some NoSQL and SQL databases in the light of key-value stores”. Concluding that “not all NoSQL databases perform better than SQL databases” and highlighting that “Couchbase and MongoDB are the fastest two overall for read, write and delete operations. Couchbase, however, does not support fetching all the keys (or values)” [43].

In base to this conclusion, seems interesting when it comes to testing the implementation of the IDE with Node-RED, to use a MongoDB database for storing and analyzing the data received in the application case scenarios.

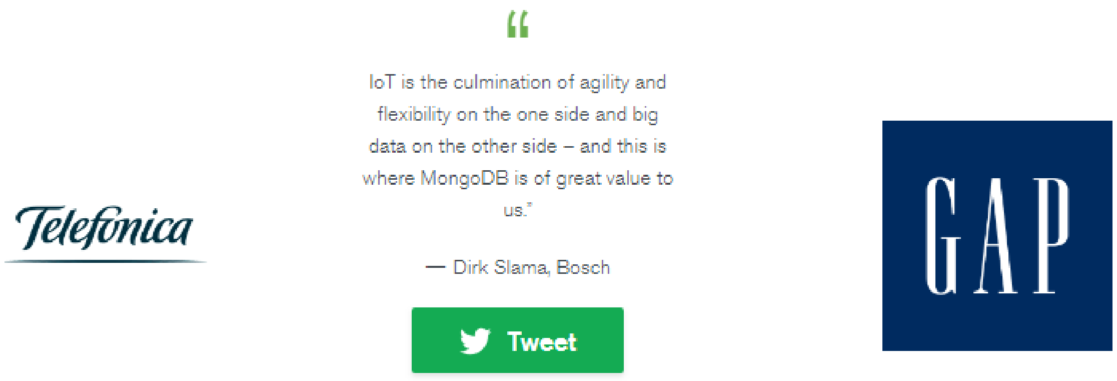
2.4.1. MongoDB

The basic features of MongoDB are [44]:

1) MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.
2) The document model maps to the objects in your application code, making data easy to work with.
3) Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze the data.
4) MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use.
5) MongoDB is free and open-source.

These features provide clear advantages in the IoT era, since this system can manage data of any structure, no matter how often it changes. MongoDB is built to scale out on commodity hardware, serves millions of users and billions of sensors without complex hardware or extra software. As sensor data grows, so does MongoDB. Finally, MongoDB can analyze data of any structure. It can do so directly within the database, giving results in real time and without expensive data warehouse loads [45].

Moreover, MongoDB is already being used by different companies to store data, just as it is showed on Figure IV, providing certain guarantee when it comes to using it on the application case scenarios.


“

IoT is the culmination of agility and flexibility on the one side and big data on the other side – and this is where MongoDB is of great value to us.”

— Dirk Slama, Bosch

Figure IV. MongoDB Customers

Source: MongoDB

2.5. DEVOPS

DevOps does not have a unique and totally accepted definition, since it represents a large enough concept that requires some nuances to fully understand; some of the most representative ones are provided by Len Bass et al. (2015) affirming that "DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality" [46], and M. Huttermann (2012) defining DevOps as "a set of practices that advocate the collaboration between software developers and IT operations where the aim is to shorten the feedback loop while aligning the goals of both the development and IT operations departments" [47]. In base to these definitions, some conclusions can be drawn:

1) The importance of DevOps does not reside in the set of practices, any of the authors focused on that, but in the goal that it pursues. In fact, other authors as C. Boettiger, defined DevOps as a philosophy or a set of principles focused on meeting particular goals [48].
2) Right interaction between Development and Operations/Production phases must be provided.
3) The aim that is pursued by the application of the set of practices is obtaining high efficiency, reducing times and ensuring quality.

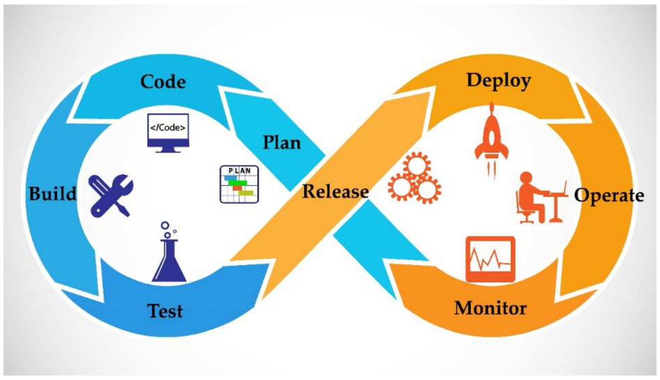


Figure V: DevOps

Source: ShutterStock

DevOps practices are categorized in five different categories according to Len Bass et al. (2015) [46]:

1) High requirements to ensure high quality.
2) Development directly responsible for errors in production. (reduce time to solve errors)
3) Enforce deployment process used by both Development and Operations. (reduce misconfigurations)
4) Continuous deployment and automation of processes.
5) Quality Control Practices during the whole process.

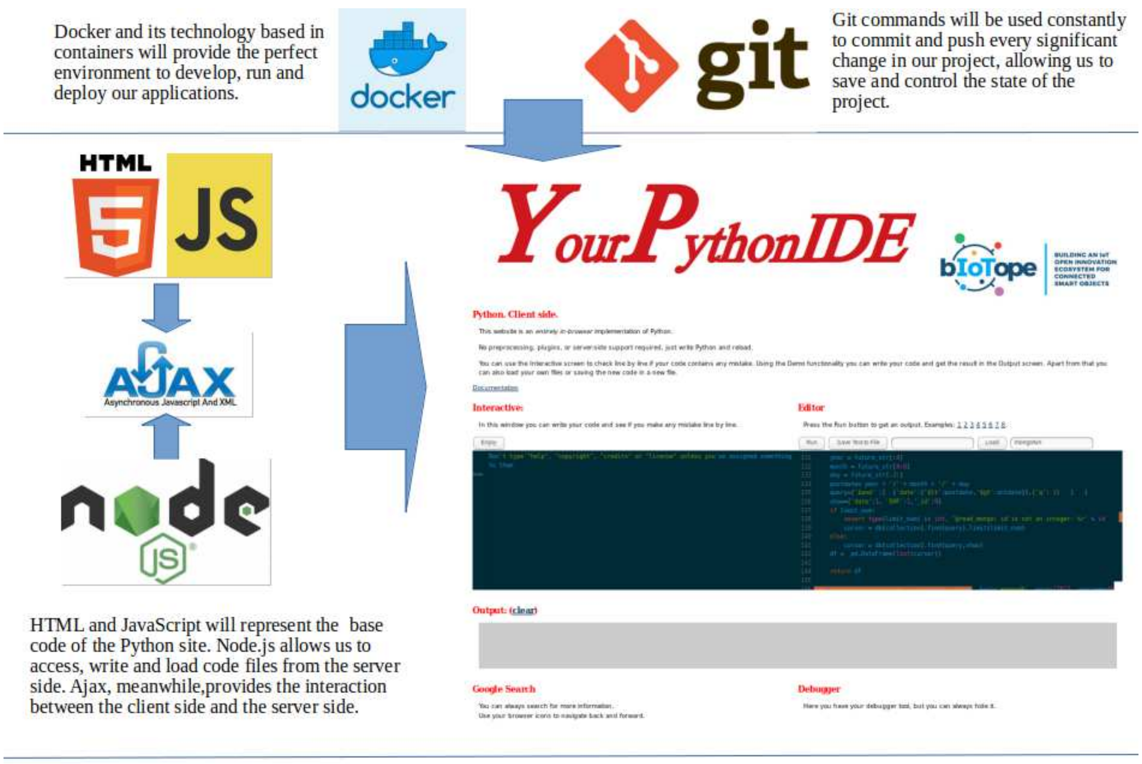
To get a better understanding of the DevOps relevance nowadays, a study conducted by Gartner in 2016 showed that DevOps tools (IT tools directly align with the DevOps practices and principles, such as Docker, Jenkins, Git…) reached $2.3 billion in 2015 and that "DevOps will evolve from a niche to a mainstream strategy employed by 25 percent of Global 2000 organizations" [49].

DevOps philosophy and concepts are particularly important in the development phase of this project since they are applied continuously in order to reduce errors and failures, time of development and to increase the overall quality of the final solution. These concepts are further developed on *Chapter 4.1.2. Methodology*.

CHAPTER 3. TECHNOLOGIES

Following with the proposed actions to accomplish the objectives of the project, a study of the different technologies required for the development phase is conducted, emphasizing on their fundamentals, and their current applications.

In order to develop the Python Integrated Development Environment under the DevOps philosophy different tools and programming languages are required, these technologies needed for the development are presented and briefly described on the following scheme (Figure VI):



Docker and its technology based in containers will provide the perfect environment to develop, run and deploy our applications.

Git commands will be used constantly to commit and push every significant change in our project, allowing us to save and control the state of the project.

HTML and JavaScript will represent the base code of the Python site. Node.js allows us to access, write and load code files from the server side. Ajax, meanwhile,provides the interaction between the client side and the server side.

Python. Client side.

This website is an *entirely in-browser* implementation of Python.

No preprocessing, plugins, or serverside support required, just write Python and reload.

You can use the Interactive screen to check line by line if your code contains any mistake. Using the Demo functionality you can write your code and get the result in the Output screen. Apart from that you can also load your own files or saving the new code in a new file.

Interactive:

In this window you can write your code and see if you make any mistake line by line.

Editor

Press the Run button to get an output. Examples: 1 2 3 4 5 6 7 8

Google Search

You can always search for more information.
Use your browser icons to navigate back and forward.

Debugger

Here you have your debugger tool, but you can always hide it.

Figure VI. Technologies for Development

3.1. DOCKER

Docker is an open platform for developing, shipping, and running applications. Docker enables the separation of the applications from the infrastructure, so that software can be delivered quickly [50].

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow the user to run many containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a hypervisor but run directly within the host machine's kernel. This means more containers can be run on a given hardware combination than using virtual machines. Docker containers can be run within host machines that are actually virtual machines [50].

Basically, a container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings [51].

Docker provides tooling and a platform to manage the lifecycle of the containers [50]:

1) Developing applications and its supporting components using containers.
2) The container becomes the unit for distributing and testing the application.
3) Deploying the application in the production environment, as a container or an orchestrated service. This works the same whether the production environment is a local data center, a cloud provider, or a hybrid of the two.

According to C. Boettiger (2015): "Docker combines several areas from systems research - such as operating system virtualization, cross-platform portability, modular re-usable elements, versioning, and a 'DevOps' philosophy" [48]. In fact, Docker has become one of the most representative tools of DevOps [52], Clark et al. (2014) affirmed that: "technologies that allow efficient usage of available hardware, like Docker, stand to provide substantial savings and potential for re-use by researchers with less direct access to capital" [53]. Docker provides a faster time to market (development and operational agility due to containerization), higher development productivity (eliminating dependencies issues), a reduction of IT infrastructure (better utilization of the server compute density) and a higher IT operational efficiency (automating management of different applications into a uniform operating model) [54].

These features have lead international companies to develop some of their services using docker [55]:

1) **GSK (GlaxoSmithKline):** "Docker allows GSK to support a multitude of tools and technologies and interfaces so that scientists can run data analysis at scale". Ranjith Raghunath (GSK Director of Big Data Solutions).
2) **PayPal:** 200,000 Containers are managed in the Cloud to speed transactions and productivity of PayPal.
3) **Lindsay Corporation:** leverages Docker to connect 450,000 of its IoT-enabled FieldNET pivot irrigation systems globally to save over 700 billion gallons of water.

In conclusion, Docker will provide the environment to develop, test and deploy our service. Thanks to its technology based on containers, Node-RED and the Integrated Development Environment will be built with all the dependencies needed to be run and deployed in any infrastructure.

3.2. GIT

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency [56]. Git was created in 2005 by Linus Torvalds, (the creator of Linux) as a distributed system that needed to satisfy some requirements as speed, simple design, strong support for non-linear development, fully distributed and able to handle large projects [57]. Since its birth, Git has evolved to be easy to use and yet retain these initial qualities [57].

One of the nicest features of any Distributed Source Code Management (SCM), Git included, is that it's distributed (Figure VII). This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository. This means that even in a centralized workflow, every user essentially has a full backup of the main server. Each of these copies could be pushed up to replace the main server in the event of a crash or corruption. In effect, there is no single point of failure with Git unless there is only a single copy of the repository [57].

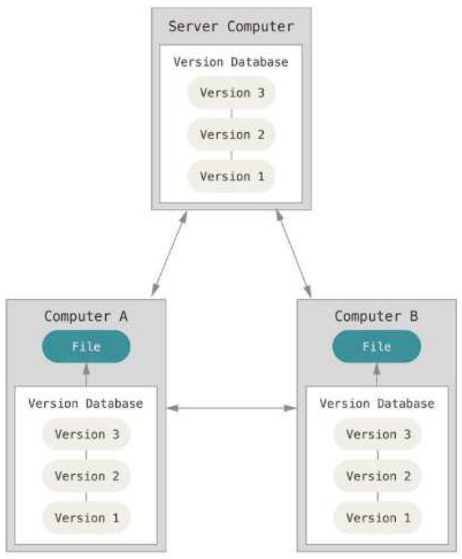


Figure VII. Distributed SCM

Source: PRO Git 2014 [57]

Other particular features of Git are the existence of an “Staging Area” (An intermediate area where commits can be formatted and reviewed before completing the commit), the Branching and Merging (The possibility to divide the project in “branches” that can be developed separately and also merging them into one that contains all the information) or the Data assurance (ensures the cryptographic integrity of the whole project) [58].

While it’s not a direct part of the Git open source project, it is important to mention the existence of GitHub, GitHub is the single largest host for Git repositories, and is the central point of collaboration for millions of developers and projects. A large percentage of all Git repositories are hosted on GitHub, and many open-source projects use it for Git hosting, issue tracking or code reviewing [57].

Git along with GitHub, currently, represent one of the most relevant DevOps tools [52], since they provide developers and operations teams a platform to collaborate, monitor and control changes in the project, improving the efficiency and reducing the risk to failure. Companies as Facebook, Google or Microsoft made use of Git [56].

Through the development of this Master Thesis, Git technology is used constantly to save, monitor and track the progress of the project.

3.3. PROGRAMMING AND MARKUP LANGUAGES

As part of the development of this project, the use of different programming languages is needed to meet the defined objectives. These programming languages are JavaScript for IDE development and Python for automating processes and completing the application case scenarios. HTML is used as a markup language to allow the IDE's implementation on browser.

3.3.1. JavaScript

JavaScript is a lightweight, interpreted programming language with object-oriented capabilities. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers and embellished for web programming with the addition of objects that represent the web browser window and its contents. Client-side JavaScript allows executable content to be included in web pages -- it means that a web page need no longer be static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content [59].

The origin of JavaScript dates back to 1995 when Brendan Eich began developing a scripting language called LiveScript for the upcoming release of Netscape Navigator 2.0, with the intention of using it both on the browser and on the server. Just before Netscape Navigator 2.0 was officially released, Netscape changed the name to JavaScript in order to capitalize on Java as a new Internet buzzword. Netscape's gamble paid off and JavaScript became a must-have from that point on [60].

A complete JavaScript implementation is made up of three distinct parts [60]:

1) **The Core (ECMAScript)**: ECMAScript is simply a description, defining all the properties, methods, and objects of a scripting language. "ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified...apart from any particular host environment" [60].
2) **The Document Object Model (DOM):** The Document Object Model (DOM) is an application programming interface (API) for HTML as well as XML. The DOM maps out an entire page as a document composed of a hierarchy of nodes. Each part of an HTML or XML page is a derivative of a node. It describes methods and interfaces for working with the content of a Web page.
3) **The Browser Object Model (BOM):** describes methods and interfaces for interacting with the browser. Using the BOM, developers can move the window, change text in the status bar, and perform other actions that do not directly relate to the page content. What makes the BOM truly unique, and often problematic, is that it is the only part of a JavaScript implementation that has no related standard.

According to Web Technology Surveys, JavaScript is used by 95% of all the websites (November 2018) [61] due to its capability to add behavior; different sites as Google, Facebook or YouTube make use of JavaScript programming language.

Since the Integrated Development Environment requires dynamic functionalities (the user should be able to write, save, load and run Python code), this project employs JavaScript programming language because it presents the right features to develop the required functionalities on the client side.

3.3.2. Python

Python is an interpreted, open source, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many systems calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many Unix variants, on the Mac, and on Windows 2000 and later [62].

Three of the most important features of Python are that is easy to learn, easy to use and with a maturing ecosystem of scientific libraries and modules:

1) **Easy to Learn**: “Most developers in the CMS experiment are physics students looking for new physics in the data. Usually they don’t have any formal IT training. Python allows them to be productive from the very start and to dedicate most of their time on the research they want to carry out.” [63] Benedikt Hegner – CERN. In fact, when its creator, Guido van Rossum, began to develop the Python programming language in the 1980s, he worked at the Centrum voor Wiskunde en Informatica in Amsterdam (Netherlands) in the ABC Working Group. ABC was a programming language that had been designed to teach IT students, so he took advantage of this situation to create a new programming language with similar characteristics [63].
2) **Easy to Use**: Python executes programs immediately; there are no intermediate compilers or link steps, which makes for an interactive programming experience and rapid turnaround after changes in the programs. But also, Python eliminates much of the complexity of other programming languages with a deliberately simple syntax and powerful built-in tools.
3) **Libraries and Modules**: “The Python ecosystem is very rich and well-developed. Our developers can incorporate existing libraries into their projects and only need to develop the new functions that they need” [63] Tarek Ziadé – Mozilla Service Team.

The Python programming language is establishing itself as one of the most popular languages for scientific computing [64]. Python libraries and modules include some of the most powerful tools in the Industry 4.0 as Scikit-learn (“Scikit-learn harnesses this rich environment to provide state-of-the-art implementations of many well-known machine learning algorithms, while maintaining an easy-to-use interface tightly integrated with the Python language”) [64], Numpy (“Input data is presented as numpy arrays, thus integrating seamlessly with other scientific Python libraries. It also provides basic arithmetic operations”) [64], or Scipy (“efficient algorithms for linear algebra, sparse matrix representation, special functions and basic statistical functions”) [64].

These characteristics make Python an important technology for this project. Python scripts will be used for automating processes and commands in the command line (easy to learn and easy to use). Moreover, Python functions, that include external modules as numpy, will be developed to test the IDE performance in the application cases.

3.3.3. HTML

In order to publish information for global distribution, exists the need for a universally understood language, a kind of publishing language that all computers may potentially understand. The publishing language used by the World Wide Web is HTML [65], which stands for *HyperText Markup Language*.

HTML was originally developed by Tim Berners-Lee at CERN and popularized by the Mosaic browser developed at NCSA. During the course of the 1990s it has blossomed with the explosive growth of the Web. During this time, HTML has been extended in several ways, until its last release HTML 5.2 on December 2017, giving authors the means to [65]:

1) Publish online documents with headings, text, tables, lists, photos, etc.
2) Retrieve online information via hypertext links, at the click of a button.
3) Design forms for conducting transactions with remote services, for use in searching for information, making reservations, ordering products, etc.
4) Include spread-sheets, video clips, sound clips, and other applications directly in their documents

Since, the IDE needs to be displayed on web, results pretty obvious to develop part of the client-side interface using HTML as a universal language in the World Wide Web.

3.4. NODE.JS

The official website defines Node as “a platform built on Chrome’s JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking Input/Output model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices” [66]. One of these applications based on Node.js technology, as it has previously introduced, is Node-RED, which takes full advantage of its event-driven, non-blocking model, making it ideal to run at the edge of the network on low-cost hardware as well as in the cloud [21].

Node uses V8, the virtual machine that powers Google Chrome, for server-side programming. V8 gives Node a huge boost in performance, preferring straight compilation into native machine code over executing bytecode or using an interpreter [67]. Because Node uses JavaScript on the server, other benefits are also observed by M. Cantelon et al. (2011), on their book *Node.js in action* [67]:

1) Developers can write web applications in one language, which helps by reducing the context switch between client and server development, and allowing for code sharing between client and server, such as reusing the same code for form validation or game logic.

2) Using JSON, which a very popular data interchange format today, since it is native to JavaScript.
3) JavaScript is the language used in various NoSQL databases (such as CouchDB and MongoDB), so interfacing with them is a natural fit.
4) JavaScript is a compilation target, and there are a number of languages that compile to it already.
5) Node uses one virtual machine (V8) that keeps up with the ECMAScript standard. In other words, browsers do not have to catch up to use new JavaScript language features in Node.

In definitive, Node.js's architecture makes it easy to use a highly expressive, functional language for server programming, without sacrificing performance and stepping out of the programming mainstream [68]. Hence, these advantages are exploited in this project due to the fact that some functionalities of the IDE require access to the server side (saving the code written in the IDE into a new file or loading the content from one file of the server into the IDE).

3.5. AJAX

Asynchronous JavaScript And XML (AJAX) is the primary method for enabling asynchronous communication over the Web [69]. Actually, important tools as Google Maps or Gmail make use of Ajax technology.

Ajax isn't just a technology, but several of them coming together in a powerful new way. Summarizing, Ajax incorporates [70]:

1) Standards-based presentation (*XHTML* and *CSS*).
2) Dynamic display and interaction (*Document Object Model*).
3) Data interchange and manipulation (*XML*).
4) Asynchronous data retrieval (*XMLHttpRequest*).
5) JavaScript binding everything together.

A layman's explanation of how Ajax works is made by w3schools.com [71]:

1) An event occurs in a web page (the page is loaded, a button is clicked)
2) An XMLHttpRequest object is created by JavaScript
3) The XMLHttpRequest object sends a request to a web server
4) The server processes the request
5) The server sends a response back to the web page
6) The response is read by JavaScript
7) Proper action (like page update) is performed by JavaScript

In definitive, Ajax provides a simple and efficient way to interconnect the client with the server side without reloading the webpage.

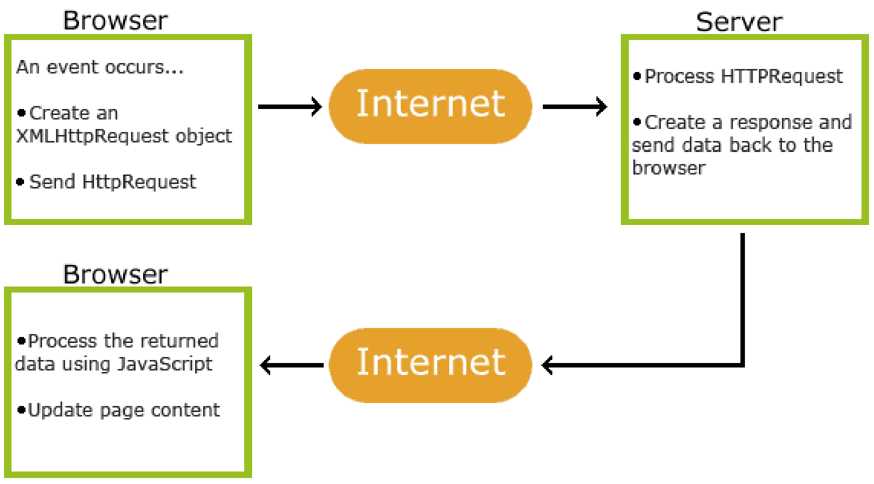

Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

Figure VIII. Ajax working example

Source: W3Schools

Regarding this Master Thesis, Ajax technology is used to take advantage of the great possibility that it offers as it is providing the connection between the client side and the server side. For example, when the user loads a file in the Integrated Development Environment, a request asking to read the contents of a file on the server side will be sent, the server will process the request and afterwards those contents will be sent back as a response to a JavaScript function that will write the contents in the editor.

CHAPTER 4. DEVELOPMENT PHASE

4.1. STAGE 1: DEFINITION

In this subchapter, it is presented not only the basic structure and methodology to follow during the development phase but also the needed requirements that must be accomplished and the most important tasks that must be performed. This stage has a vital importance for the development since it provides the foundations where the rest of the development will be built on.

4.1.1. Methodology

The methodology followed during the development phase is based on the Iterative Development Model. Iterative Development is an approach to building software in which the overall lifecycle is composed of several iterations on sequence, each iteration is composed for activities such as requirements analysis, designing, programming and testing [72] (**Figure IX**).

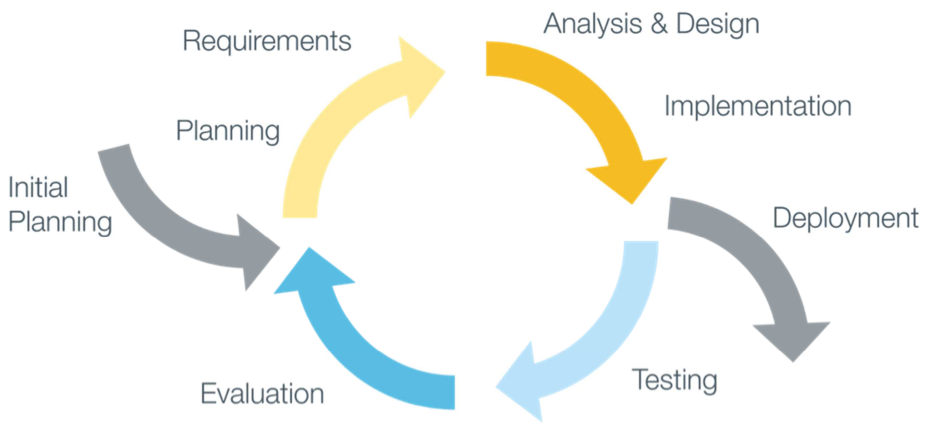


Figure IX. Iterative Development Model

Source: WebSiteDesign

In definitive, the development phase requires an initial planning that defines methodology, structure and initial requirements; however, they will evolve as the development advances. Then, in order to fulfil the requirements an analysis of the tasks needed will be conducted; once the tasks are performed, the resultant environment is tested, if the result is not satisfactory a debugging of the errors proceeds, then the cycle begins again with the definition of new requirements that may be identified. The iterations will finish when no more requirements are identified and the rest of them are fulfilled, ensuring that the environment is ready to be deployed in the real case scenarios.

In parallel with the Iterative Development Model proposed, different practices and tools aligned with the DevOps principles are used. As it has been previously stated, DevOps philosophy is applied in order to reduce failures, time of development and to increase quality.

The first measure adopted related with DevOps is the decision to work on a virtual machine during the whole development phase. A virtual machine is a computer file, typically called an image that behaves like an actual computer [73]. With a virtual machine a total isolation from the host system is achieved, meaning that the software inside a virtual machine can't escape or tamper with the computer itself. This feature is interesting since allows us to test and get used to the technologies used in the project, preventing damages in the host operating system that would entail in more developing time.

The second important DevOps practice is making use of Git. The methodology defined is clear, each significant change or development in the project must be committed to Git; once a requirement is fulfilled, the change will be "committed" (picture of the actual state of the project) and after that "pushed" on GitHub (saved on the GitHub repository), assuring that all the advances are protected even if the system breaks down. Also, Git technology allows the control and the monitoring of the state project, providing a way to go back to a previous state if something is not working as it should.

Another DevOps practice is the use of Docker. Docker and its technology based on containers permits the utilization of the developed environment in any infrastructure but also installing just the needed dependencies; providing faster time to deployment (development and operational agility due to containerization), higher development productivity (eliminating dependencies issues), and a reduction of the IT infrastructure (better utilization of the server compute density).

The last measure adopted regarding DevOps philosophy is the creation of a Python file to automate commands, this file can be visualized in the Annex section (*Annex IX. Automated Docker Commands*); but the idea behind it is to speed up the execution of highly-used sequences of commands as building a docker image or stopping a container. In this way, anytime that these sequences of operations need to be performed, only one command that executes the Python file is used.

4.1.2. Requirements

The initial requirements are defined attending to the problem that is pursued to solve. This problem is presented in the introduction chapter, it is needed an Integrated Development Environment that facilitates the creation of more complicated IoT applications in Node-RED; helping users to write Python code with more productivity by the existence of tools like a code editor, a debugger or functionalities that allow loading or saving files to be used in Node-RED.

Therefore, the initial requirements that need to be fulfilled in the development phase are:

1) **Node-RED connection**: The developed solution should provide a connection to the Node-RED environment. The creation of a new environment that improves user productivity

writing Python code along does not satisfy the objective of facilitating the creation of IoT systems in Node-RED.

- Node-RED and the developed IDE should be run as a unique service to establish the connection.
- Cloud-based IDE, data must be in the cloud to be accessible.

2) **Python Code Editor:** Providing a platform where Python code could be written and executed in order to inform users about where an error is made or about the real output of the executed code.
   - Interpreter (Python/JavaScript) to allow the Python code to be run in the browser.
   - Output screen that displays errors or the text exit after executing the code
3) **Debugger:** The new environment will have a debugger tool that facilitates the depuration of errors using different commands.
4) **Functionalities Saving and Loading:** The new IDE must allow users to load Python files in order to visualize the errors, make changes or consult if they obtain the expected results. The developed IDE will also allow saving the code written in the code editor into external files that will connect with Node-RED.
   - Dynamic site, the user decides when he wants to load or save by clicking a button.
   - Access to the server side to read and write in external files.
   - Interconnection between the server and the client side.
5) **User-Friendly interface and Simplicity:** Apart from the technical aspects, the new environment has to be adequate to the purpose that need to accomplish from the design point of view, avoiding the presence of elements that do not add value to the site and highlighting the presence of the main functionalities to the users, explaining how they work.

Moreover, following with the methodology proposed, new requirements are identified as the project advances in a second iteration. The new requirements identified are:

6) **Being able to import Python modules: Numpy, Pygal, Matplotlib:** Directly related with the requirement of creating a Python Code Editor, it is identified a new requirement. Developers of Trinket have designed a possible implementation of these modules in Skulpt. Taking advantage of this opportunity, the new modules will be implemented on the new environment.
7) **Capacity to represent graphs:** As the project advances, it is detected the necessity to represent the graphs that are result of the execution of the code. These graphs should appear in a pop-up window in order to prevent misconfigurations on the Python site.
8) **Google iframe:** Following with the idea of facilitating the creation of Python code, a google iframe that allows users to search for more information or solving doubts in a fast and efficient way must be provided.

4.1.3. Tasks

In this subchapter, it is intended to present the most important tasks that need to be performed in the development phase. A differentiation is made between the tasks required to define the development and the tasks needed to fulfil the requirements.

1) Definition tasks:

- Defining methodology, requirements, structure and scheduling

2) Requirement tasks (1st Iteration):
    - Building Skulpt: Forking Skulpt site, eliminating features that add no value, redesign of the interface, including project documentation. These actions are related with the requirements 2), 3) and 5).
    - Developing JavaScript functions: Coding of functions that allow loading and saving on the client side. Related with requirement 4).
    - Creating Node.js files: Coding of functions that allow writing and reading files on the server side. Related with requirement 4).
    - Connecting the server side with the client side: Using Ajax to provide connection. Related with requirement 4).
    - Getting Node-RED with its last improvements: Cloning from GitHub the Node-RED environment with the last functionalities. Related with requirement 1).
    - Integrating the IDE with Node-RED: Dockerizing as a unique service using volumes. Related with requirement 1).
3) Requirement tasks (2nd Iteration):
    - Implementing Python Modules: Downloading JavaScript files, defining variables inside Sk.externalLibraries. Requirement 6).
    - Configuring Graphs: JavaScript function, designing pop-up window. Requirement 7).
    - Developing Google iframe: Javascript function. Requirement 8).

Note: It is important to underline the idea exposed in the methodology chapter; once each of this task is performed, the resultant environment is tested, if the result is not satisfactory (the requirements related with the task are not fulfilled) a debugging of the errors proceeds. Meaning that testing or debugging do not represent different tasks, they are integrated in the completion of the tasks. Similarly, actions as building the docker images or committing changes using git are also included in the tasks proposed above.

4.1.4. Structure and Scheduling

The development phase is divided into three clearly differentiated stages depending on their main purpose:

1) **Definition**: Include all the actions related with understanding the basics of the development phase like requirements, tasks, structure or methodology to follow during the development.
2) **Building a Python Development Environment based on Skulpt**: Tasks needed to create the new Development Environment as building Skulpt, developing JavaScript functions, creating Node.js files, connecting the client with the server side, implementing Python modules, configuring graphs or developing the Google iframe.
3) **Implementation with Node-RED**: Establishing the connection between Node-RED and the Integrated Development Environment. This stage includes the tasks getting Node-RED with its last improvements and integrating the IDE with Node-RED.

In accordance with all the information exposed (methodology, tasks, structure), the schedule followed during the development phased is presented in Figure X, Figure XI and Figure XII:

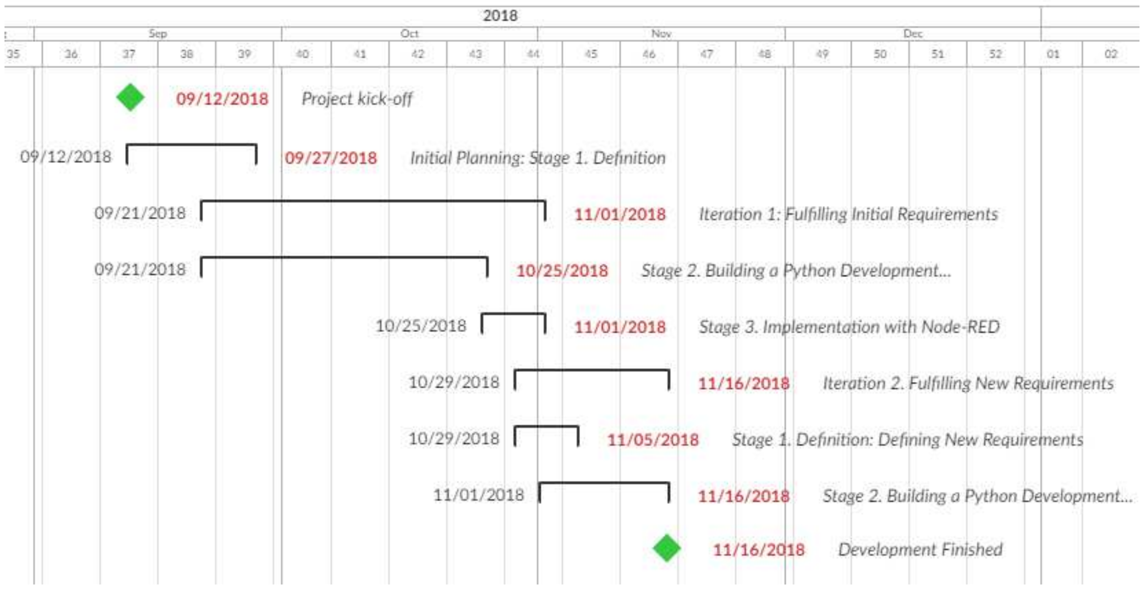


Figure X. Development Scheduling: Phases

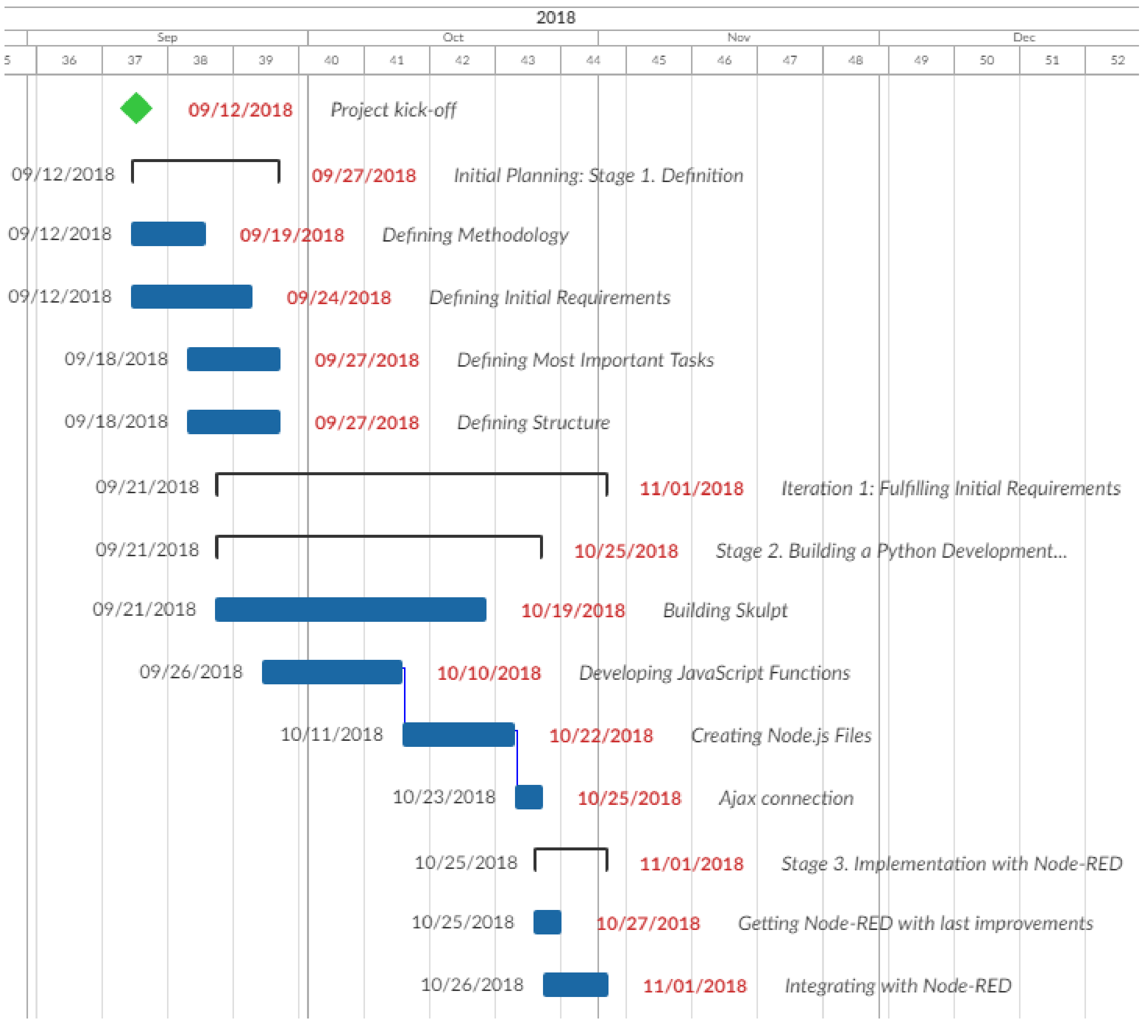


Figure XI. Development Scheduling: Initial Iteration

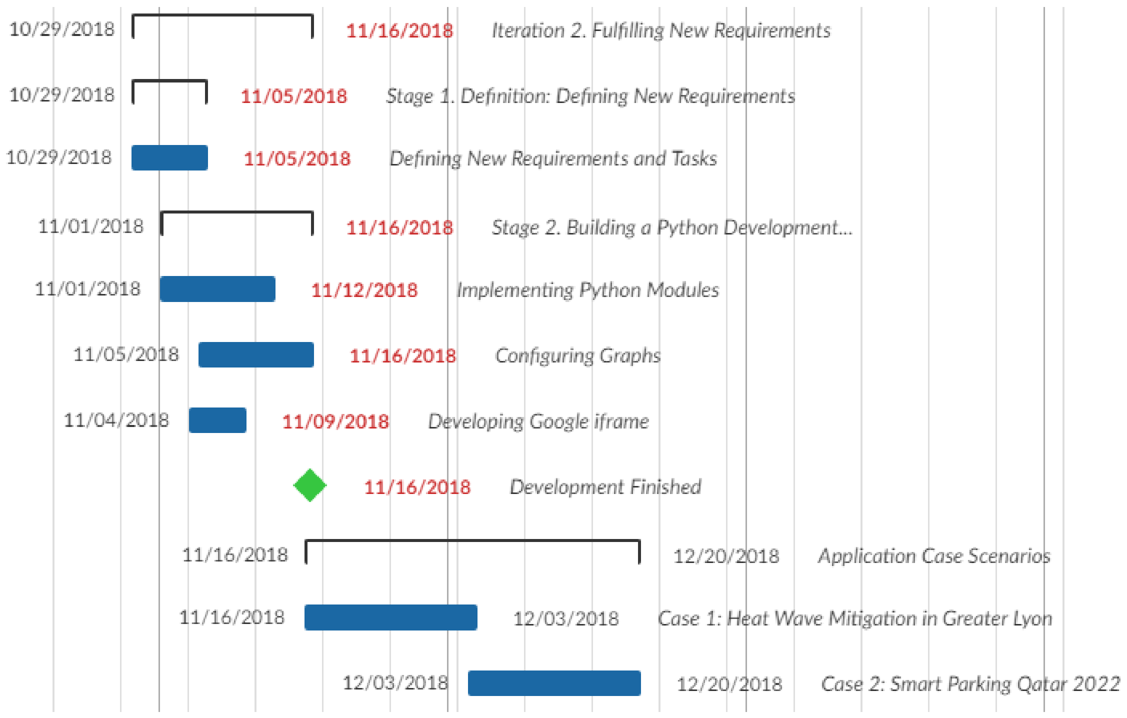


Figure XII. Development Scheduling: Iteration 2 + Deployment

4.2. STAGE 2: BUILDING A PYTHON DEVELOPMENT ENVIRONMENT BASED ON SKULPT

As it has been previously introduced, the second stage involves all the activities needed to create the new Development Environment as building Skulpt, developing JavaScript functions, creating Node.js files, connecting the client with the server side, implementing Python modules, configuring graphs or developing the Google iframe.

4.2.1. Building Skulpt

Skulpt provides not only the translation of the Python code into JavaScript in order to allow its execution in the browser *(Chapter 2.3. Interpreters*), but also other tools as a debugger or a code editor directly related with the satisfaction of some of the requirements defined.

As an open source project, Skulpt creators invite other users to build Skulpt and develop their own projects based on it, facilitating a set of steps to build it. The set of steps consist of [35]:

1) Cloning the repository from GitHub.
2) Installing node.js, Java, and Python 2.
3) Installing dependencies using npm install
4) Navigating to the repository and running npm run build

After, completing these steps, some dependencies were still missing, evidencing that the set of steps do not guaranteed building Skulpt on any infrastructure. The rest of the dependencies are then installed without much problem, allowing the building of Skulpt just as it was developed. However, an issue is open on their GitHub repository to ensure that they have not developed any docker container that builds Skulpt on any infrastructure. The issue received responses from different Skulpt developers, including its leader Brad Miller that confirmed that no docker container was developed but that “definitely would be beneficial” for their project [74]. In base to their responses and with the objective to familiarize with the docker technology (fundamental for the development of this project), a docker service that builds Skulpt on any infrastructure is developed. This docker service is built by a docker-compose.yml and a Dockerfile that can be visualized in the Annex document (*Annex IV. Docker-Building Skulpt*). Summarizing, the Dockerfile downloads all the dependencies needed, clones the Skulpt repository, changes the directories of some files to the directories where the new dependencies are installed and lastly builds Skulpt. Meanwhile, the docker-compose.yml executes the docker container as a service, extracting the new files built to the host system. This docker service is then shared with the Skulpt community that so much contributes to the development of the present Master Thesis.

Once Skulpt is built, an adaptation to our needs is required. In the first place, Skulpt provides some functionalities that do not add any value to this project and that need to be eliminated. The idea is to identify first which features add value to this project and eliminate the rest of them from the index.html file. The features that add value are related with the objectives and requirements identified, they include the code editor along with the output screen, the debugger, the interactive code editor (it allows detecting errors in the code at the same time that the user is writing it, line by line) and some examples of Python code already written to help users to understand how the environment works. The reminded parts of the code were eliminated.

The second measure related with adapting the environment to our needs has to do with its design. This project is much more focused on functionality than on an esthetical, polished and innovative design, for this reason, the modifications made try to improve user-experience rather than its image. A logo is added that defines what the environment is intended for (“Your Python IDE”), the linear structure of Skulpt is modified to a two-column structure making a better use of the space and allowing the user to visualize all the functionalities without scrolling down. Another action performed related with the design, is the addition of titles and paragraphs above each functionality, facilitating its understanding for first users.

As this development is part of an open source project, a link to a markdown document that summarizes the fundamentals of the development is presented. The intention is that any user could understand how the project was developed or even helping them to develop new projects with similar properties or technologies. This document may be found on the Annex section (*Annex VIII. Project Documentation.md*).

After the consecution of the generic task: Building Skulpt, the requirements defined as the incorporation in the Integrated Development Environment of a debugger tool and a code editor or having a user-friendly interface are satisfied; however, the process was not that simple and multiple stages of testing and debugging were needed.

In order to check that the code editor worked as expected, different pieces of Python code with different characteristics (different operations performed, with different errors…) were executed, getting some bugs and errors that had to be corrected. The first bug detected was encountered after running two successful Python programs in a row, the output screen displayed the results of both programs; it was decided to modify the function that executes the program, including some lines that remove the result of the previous program, allowing the user to visualize just the result of the code that is actually written in the code editor. Another error related with the code editor was discovered after running a Python function with multiple errors, since it did not show anything on the output screen; it was discovered that the function on charge of executing the code contained an if sentence that removed the exit when an error was encountered, removing the conditional, the error appeared in the output screen.

Regarding the debugger, only one issue was identified. Using the help command, a good explanation appears of how the different debugger commands work, however, when they were used any of them seemed to have any effect; after reviewing the code it was noticed that the debugger was pointing to one of the screens removed from Skulpt, changing the scope of the debugger to the code editor screen, it was checked that all of the commands worked as they should.

In relation with the user-friendly interface and simplicity requirements; testing and debugging consisted of loading and visualizing the developed environment, modifying everything that could have room for improvement. The resultant file with the last modifications that meet these requirements can be found on the Annex section (*Annex I. Index.html*).

To conclude this task, following with the methodology proposed, all the changes are included in the git repository of GitHub, making use of the commands *git add * (*adding all the files and folders of the current directory to the "Staging Area"*), git commit -m (*committing changes of the staging area to git*), git push origin master* (pushing to the GitHub repository). Also, a docker image that can be run in any infrastructure with all the dependencies is built, creating a basic Dockerfile (*Annex IV. Docker-Building New Environment-Dockerfile IDE*) that copies all the documents present in the host directory of the dockerfile to the Docker container and exposes a port in order to be run and deployed later on the browser.

4.2.2. Developing JavaScript Functions

One of the tasks defined to satisfy the requirement of having functionalities on the IDE that allow users to load their own files and also to save the text written in the code editor into a new file, is developing new JavaScript functions. These functionalities are not included in the Skulpt environment and due to the fact that they require dynamic properties (is the user who decides which file has to be load, or when the text written in the editor has to be saved) it is decided to develop them using JavaScript. It is important to highlight that these features cannot be achieved just by developing JavaScript functions since the require access to the server side to write or read the text of external files.

Even though they require similar actions, it is needed to distinguish between the JavaScript function that pursues loading files and the one that pursues saving the text:

1) **Loading external files:** In the first place, it is created a button with the text "Load", facilitating the understanding to the user, and an input area just right beside the button,

where the user can write which file must be loaded. When the button is clicked, it is immediately call the JavaScript function Loading. This function performs a set of actions: Firstly, the text written in the input area is read, then it is used the ajax technology to send the contents to the server side and a response is received (This step will be explained deeper in the subchapter *4.2.4 Ajax Connection*), ultimately, the response is written in the code editor removing any contents that may be already there.

2) **Saving Text into a File:** Similarly to "Loading external files", a button with the text "Save Text to File" is created along with an input area where the user can write the name of the new file. If the button is clicked, a JavaScript function is called, performing different actions as reading the contents of the code editor, reading the contents of the input area with the name of the new file wrote by the user, sending all these contents to the server side using Ajax or initializing a download with the title of the input area and the contents of the code editor.

Different actions related with testing and debugging were also needed. The general idea was to ensure that all the steps defined where executed successfully; for example to ensure that the contents of the different parts were correctly read or sent, it was included in the JavaScript function some commands to display the contents on the console, identifying errors as reading only the initial contents of the code editor without identifying the changes, sending the text in unreadable formats… etc. The errors were solved changing different commands, for example *using .getValue()* instead of *.getElementByID().value*. The final result can be observed on *AnnexII. JavaScript Functions*.

When the functions accomplished their objective, the changes were committed to Git. Regarding Dockers, not so many actions were needed, these functions do not represent another container since they work directly integrated with the site already developed based on Skulpt. For that reason, it was only required to include the file with the functions in the directory of the previous Dockerfile and building it again to automatically copy the functions to the docker container.

4.2.3. Creating Node.js Files

Following with the requirement of developing an IDE that allow users to load their own files and also to save the text written in the code editor into a new file, the Node.js technology based on JavaScript is employed due to its highly expressive, functional language for server programming. In definitive Node.js, provides us with the way to access the server side to write and read external files.

Two Node.js files were created, one attempting to read the contents of a file and send them back using Ajax (Load.js), and the other one attempting to write some contents on a new file (Storage.js):

1) **Load.js:** This file uses Ajax to receive some text, this text is then processed to obtain as a result exactly the text written by the user on the input area beside the loading button but adding the extension .py to search for files with the python format. If the file exits, its contents are read and then sent back to the client side using Ajax. Most of the errors,

during the creation of the load.js file, were detected when the processed text was displayed on the command line. It was needed to *JSON.stringify* the text received, and then find a way to subtract just the part of the text that include the name of the file specified by the user in any possible case.

2) **Storage.js:** In this case a similar procedure is followed, a set of lines that includes the title and the contents that must be written in the new file is received using Ajax, then a processing stage is needed, lastly a new file is created in a location accessible to Node-RED with the body and the title specified by the user plus the extension (.py). Again, most of the errors identified appeared on the processing stage, it was required to *JSON.stringify* the text received, subtracting for both, the title and the body the parts that were relevant, then, creating a function that eliminates the counterbars (\) created by the use of the quotation marks (“) and that adds a new line when a determined combination is found (\n).

The final Load.js and final Storage.js files may be encountered in the Annex document (*AnnexIII. Node.js Files*).

Related with Docker, a new docker container had to be created (*Annex IV. Docker-Building New Environment-Dockerfile Node.js*), since these files required different dependencies and had to be executed in different ports. A new Dockerfile meet with these problems installing the dependencies and exposing two ports, that allows the execution of both files with only one container. This container and the one developed to build Skulpt run as a unique service thanks to the development of a docker-compose.yml; the docker-compose just need the name of the images of the containers, along with their ports exposed to deploy the application as a unique service.

4.2.4. Ajax Connection

Ajax technology provides the last tool needed to fulfill the requirement associated with the presence in the Integrated Development Environment of the functionalities that allow saving the code written in the code editor into external files that will connect with Node-RED and loading Python files in order to visualize the errors, make changes or consult if they produce the expected results.

For each one of these functionalities an Ajax connection between the client and the server side is required:

1) **Loading File to the IDE**: As it has been already presented, the Ajax connection sends the title written by the user of the file to load that was read by a JavaScript function from the client side to the server side, where the title is processed by the Load.js file that reads the contents of the file that correspond with the title. Again, Ajax would provide the connection, allowing the sending back of the contents of the file (server side) as a response to the JavaScript function (client) that writes the contents in the code editor.
2) **Saving Text to File**: In this case, the Ajax connection is unidirectional, the contents of the code editor with the title present in the input area are sent to the server side but no

response is required. The server side represented by Node.js, then, creates the file with the contents provided by Ajax in a location accessible to Node-RED.

A practical detail about Ajax that results beneficial to the project is that not reloading of the browser is needed to execute their actions. When the user decides to save the text written in the code editor into a file, the action is performed automatically, if afterwards, the user decides to load the file that have just created, he would not have any problem to find it and display it on the editor again without reloading the page.

Achieving the interconnection was not instantaneous, several problems appeared on the console of the environment before the right behavior defined on the requirements, they were finally overcome by providing an open url, specifying headers *("Content-Type", "Accept*"), crossDomain: *"True"* and type: *"POST"* on the JavaScript function. Besides, on the Node.js files, it was required to start listening (*app.listen*) to the port where the JavaScript function was pointing out, providing access to different origins, methods, credentials or headers and sending back the response with *response.send()*.

As the Ajax connection only requires some lines extra on the JavaScript functions and Node.js files, it is not necessary to create any docker imager, just building the images already created to include the last changes. An important consideration regarding the dockerfiles is that their ports exposed for the Node.js files must coincide with the ports specified on the Ajax connection, on the contrary the connection would not be established when they are run as a service.

4.2.5. Implementing Python Modules

The biggest limitation of Skulpt is the impossibility to import some Python modules because its developers have not yet provided a translation of their functionalities into JavaScript. During the second iteration of the definition of requirements, it was identified that developers of Trinket had translated some of these modules as Numpy, Pygal or Matplotlib. In base to that, it was decided to include them on the Integrated Development Environment allowing users to make use of them.

Implementing those Python modules required to include the JavaScript files created by the Trinket developers in the docker container and providing a connection within the files when the user employs one of the modules. As Trinket project is also open sourced, their files are cloned to the directory of the dockerfile, assuring that they will be copied to the docker container.

The second step, providing the connection between the files and the development environment, was achieved including a JavaScript function inside the index.html file. In this function, the different modules were defined as new variables by indicating their path to the JavaScript files and to their dependencies; when the modules are used and executed in a Python program these variables are accessible because they are included in the *Sk.externalLibraries*, a mechanism provided by Skulpt to add easily new modules and libraries to the project.

Several Python programs that made use of these modules were executed in order to validate the implementation. At the beginning the result was disappointing because some errors were made like forgetting a parenthesis, not expressing correctly the path (it is important to express the path of the different files in a relative way since it is going to be run in a docker container that does not understand about the absolute path of the local system), or not including all the dependencies required. When the different modules could be used without any problem in the Integrated Development Environment, the requirement was satisfied and the changes were committed to the GitHub repository of the project.

4.2.6. Configuring Graphs

As the development advanced, it was identified that when the Python program had as a result a graph, it was presented in the first place where it has space to be built on, being even more frustrating when several graphs needed to be represented, leading to misconfigurations in the development environment. In order to solve this problem a modification of the JavaScript functions provided by Skulpt was performed to display the exit graphs in pop-up windows, easy to visualize by the users, avoiding misconfigurations in the environment.

The first step taken was designing the pop-up window, configuring it with dimensions that allow displaying graphs of different sizes but not that big to complicate users working with the IDE. A new variable with the configuration of the pop-up window is created under the command *window.open(height, width, toolbar,status, …).*

The second step consisted of modifying the JS function provided by Skulpt: *Sk.domOutput*, responsible of representing the different graphs. The final solution only required to indicate in the return statement the new variable defined that opens the window appending the contents generated after executing.

4.2.7. Developing Google iframe

In order to provide the users with a fast and efficient way to solve doubts and search for more information, a Google iframe is added in the inferior part of the environment, since it does not represent a fundamental part of the project, but a helping tool aligned with the objectives proposed.

To develop the iframe, it is only needed to define its dimensions, its style and a source that in this case refers to the Google site. However, after testing its implementation, a couple of problems were detected:

1) When the iframe was loaded in the page, it scrolled down the page to the iframe. Since, the most important features of the environment are in the superior part, it was not comfortable to the user to scroll up every time the page was loaded. The decision taken consisted in building a JavaScript function on the Index.html file that automatically scrolled up the page after loading the iframe.
2) In the second place, a button that invokes another JS function was developed to allow the user to directly come back to the google site, when a search was already carried out. It was identified that when the user searched for something in the iframe the only possible solution to perform another search was to use the navigation arrows as many times as needed. The new button avoids this inconvenience.

After completing this task, another iteration began looking for more requirements, however, in this occasion no more requirements were identified and since the rest of them were fulfilled the environment was prepared for deployment in the application case scenarios.

4.3. STAGE 3: IMPLEMENTATION WITH NODE-RED

All the activities related with establishing the connection between Node-RED and the new Integrated Development Environment are included in the third stage of the development phase. These activities involve getting Node-RED with its last improvements (essentially the Python module) and integrating everything as a unique service, in order to facilitate the connection and execution of the different containers by the user.

4.3.1. Node-RED Service

The Node-RED official site provides different ways to get Node-RED, well using commands on the command line or well making use of a docker container already created by them [21]. Although these were possible options, latest improvements as the Python module realized by other developers were still no implemented. The Python Function Node enables the execution of knowledge discovery libraries from the Node-RED's visual programming interface. As a result, most of the scientific computing and data science libraries that offer a Python API become accessible in a unified manner [23].

This particular improvement is found on the GitHub repository of one of the members of the bIoTope project [23]. In concrete, on the GitHub repository, it is not only implemented the Python module with Node-RED, but also a docker-compose.yml is found that allows running as a unique service Node-RED with the MongoDB database which may be particularly useful during the deployment on the application cases.

As an open source project, there is not any problem to use git clone to get the repository into the local system. Once, this action is performed it is only needed to provide the integration with the IDE developed.

4.3.2. Integration

Proceeding with the integration, it was needed only one docker-compose.yml that integrates both Node-RED and MongoDB containers and the docker-compose developed that contained the resultant IDE. The main difficulty found to complete this operation was that the docker-compose that executed Node-RED and MongoDB was developed using a prior version of docker, so that some of the dependencies needed to run the container could not be downloaded. At the end, a new way to get the dependencies was encountered and the different containers could be run as a unique service.

This task did not finalize there, even though the containers could be run at the same time, there was not a connection between them, meaning that the code saved for example in the IDE could not be used by the Node-RED environment. To satisfy this fundamental requirement some actions were performed:

Firstly, a new structured of the folders on the local system was required, a new folder that contains all the files and folders of the Master Thesis was created. This folder consisted of the docker-compose along with the folders of the different containers (Node-RED, MongoDB…), each one of these folders include all the information to make the container run like Dockerfile, executables… etc.

The next and most relevant step was making use of volumes on the docker-compose. Volumes are an important feature of Docker since they allow persisting data [75], the idea behind it is to provide a connection with the local system; volumes are used between two folders (one of the local system, and the other from Dockers) that automatically shared they contents when the container is run, but when the container stops the contents are not removed. Following with this idea, volumes are used in this project to connect different containers making use of one folder of the local system. For example, if one file is generated in one container when the service is running it is saved on one folder of the container but with the use of volumes it may also be saved in one folder of the local system, what happens if another container uses volumes with the same folder of the local system? The response is that the file generated by the first container would also appear on the second one. That is exactly what is done in this project, the container that executes Node-RED and the container that involves the new IDE use volumes to the same folder of the local system, in this way when the user decides to save the Python function that he has developed on the IDE, it get also get stored on the Node-RED container, then it is possible to make use of the new Python function in Node-RED to generate more powerful IoT applications.

Obviously, establishing the connection was not immediate, and different errors were detected while trying to employ the IDE functions on Node-RED, most of them because of volumes. They were solved by understanding completely the volumes behavior and the different paths inside the docker containers. The final docker-compose.yml that establishes the connection and meets the requirement specified may be found on the Annex section (*Annex IV. Docker-Building New Environment-Docker-compose.yml*).

4.3.3. Example of Use

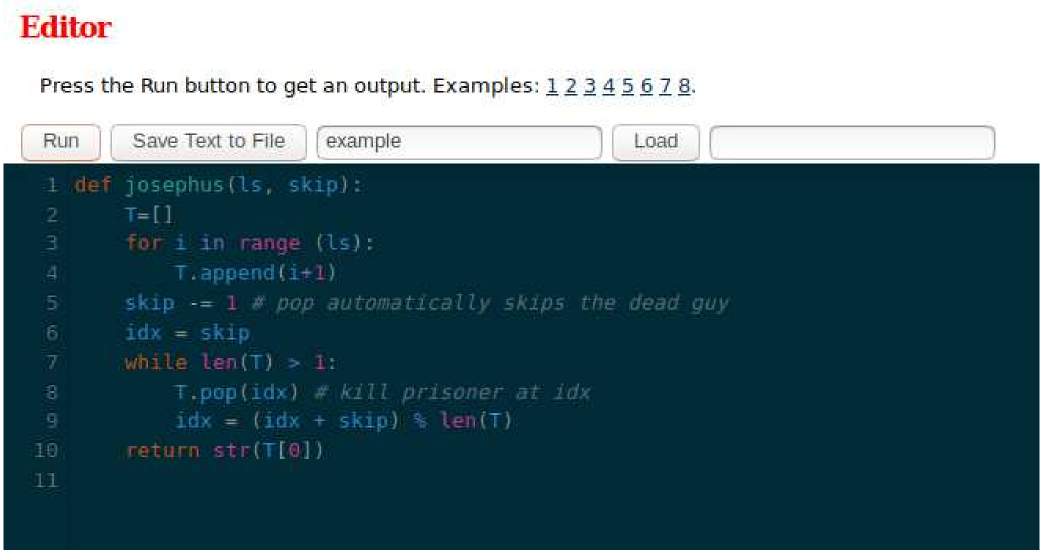
This subchapter attempts to present the reader a basic example of how the IDE works along with the Node-RED environment in order to provide a basic understanding before coping with the real application case scenarios. A common mathematical problem will be detailed and then solved by the creation of a Python function on the IDE developed and its later usage on a Node-RED flow.

The problem studied is the denominated "Josephus problem"; this problem is named after Flavius Josephus, a Jewish historian living in the 1st century. According to Josephus' account of the siege of Yodfat, he and his 40 soldiers were trapped in a cave by Roman soldiers. They chose suicide over capture, and settled on a serial method of committing suicide, they were situated on a circle, and every time the person situated on the position number seventh to the right should be killed. Josephus studied the situation and with great mathematical skills placed himself in the position that should commit suicide in the last position, in order to surrender rather than killing himself [76].

In this case, the user can define how many soldiers are trapped and which is the suicidal sequence to follow in Node-RED, then a Python function developed using the IDE will be employed on the flow to inform the user in which position should be placed if he wants to surrender to the Romans.

This problem results useful to provide information about how the whole system works, since it requires the IDE function to understand how to use the data that comes from a previous node, and also how to send the data back to the following node to get the result on screen.

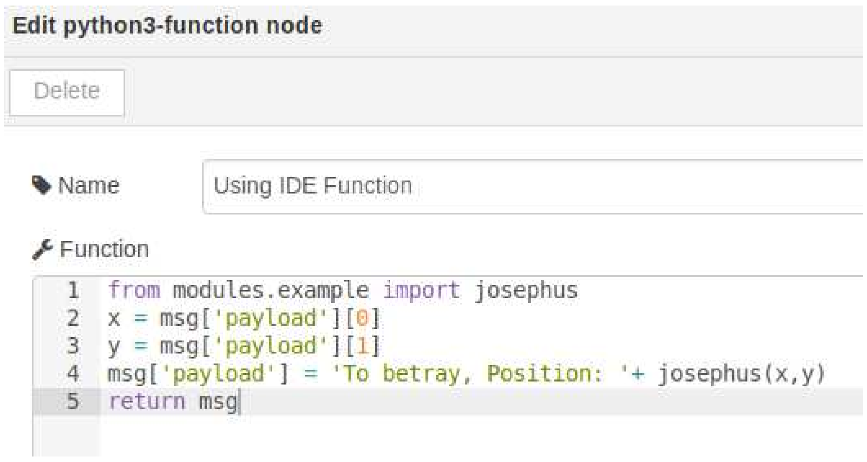
The resultant Python function that solves the Josephus problem is found on Figure XIII. The code is not completely relevant here, what it is important is to visualize that the function is called josephus and that the file is saved as example.py (.py is added automatically when the button Save Text to file is pressed) in order to understand its implementation on Node-RED.


Editor
Press the Run button to get an output. Examples: 1 2 3 4 5 6 7 8.
Run
Save Text to File
example
Load

```python
def josephus(ls, skip):
    T=[]
    for i in range (ls):
        T.append(i+1)
    skip -= 1 # pop automatically skips the dead guy
    idx = skip
    while len(T) > 1:
        T.pop(idx) # kill prisoner at idx
        idx = (idx + skip) % len(T)
    return str(T[0])
```

Figure XIII. Josephus IDE Function

How can it be integrated in the Node-RED environment? The response is using the Python module just as it is showed on Figure XIV. It is only needed to indicate from which file and which function is imported. Besides, it is good to comment that both for receiving and sending a response it is needed to use the Node-RED construction *msg ['payload']*.

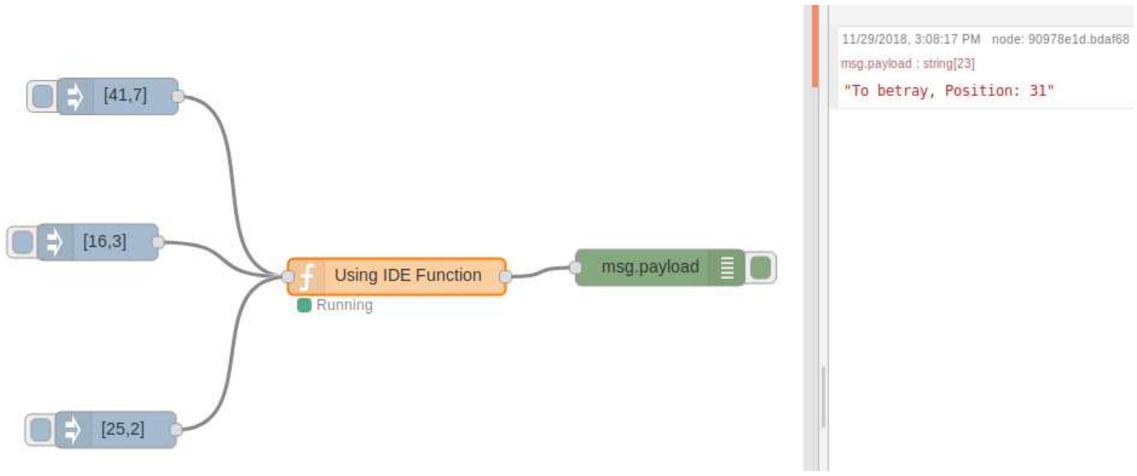


Name: Using IDE Function

```python
from modules.example import josephus
x = msg['payload'][0]
y = msg['payload'][1]
msg['payload'] = 'To betray, Position: '+ josephus(x,y)
return msg
```

Figure XIV. Using IDE Function on Node-RED

The Node-RED flow that solves the Josephus problem is displayed on Figure XV. Basically, the user can decide which combination of soldiers trapped and "suicidal" sequence is going to be followed pressing the blue button or changing its contents. The information arrives to the Python module that makes use of the function developed on the IDE to provide a response, finally the information is showed on the debug console of Node-RED.



```
11/29/2018, 3:08:17 PM  node: 90978e1d.bdaf68
msg.payload : string[23]
"To betray, Position: 31"
```

Figure XV. Josephus Node-RED Flow

In spite of the fact that this is an initial and simple problem with nothing to do with connecting smart objects and getting information from them, a first positive aspect about the behavior of the Integrated Development Environment must be highlighted, it is really easy to implement its Python functions on the Node-RED environment, just using a single sentence (from modules.file import function).

CHAPTER 5. APPLICATION CASE SCENARIOS

In this chapter, the implementation will be tested over two different case scenarios to get some insights about its utility and effectiveness. To get a full image of the case scenarios, they will be divided into five parts: problematic, justification of the case, possible solution, IDE functions and final Node-RED flow.

5.1. CASE 1: HEAT WAVE MITIGATION, GREATER LYON

5.1.1. Case 1: Problem

Heat waves are defined by the Word Meteorological Organization as "a marked unusual hot weather (Max, Min and daily average) over a region persisting at least two consecutive days based on local climatological conditions, with thermal conditions recorded above given thresholds" [77]. According to John R. Nairn (2012), heat waves have been responsible for more deaths in Australia, Europe and the United States of America than any other natural hazard, including bushfires, storms, tropical cyclones and floods [78]. In fact, this phenomenon could even be aggravated by climate change effects, another study conducted by Camilo Mora in the journal *Nature Climate Change* (2017), affirmed that "around 30% of the world's population is currently exposed to climatic conditions exceeding this deadly threshold for at least 20 days a year. By 2100, this percentage is projected to increase to ~48% under a scenario with drastic reductions of greenhouse gas emissions and ~74% under a scenario of growing emissions… An increasing threat to human life from excess heat now seems almost inevitable" [79].

But, heat waves not only can have dramatic consequences to humans, they are important events that can cause rapid changes in maritime biodiversity patterns and ecosystem structure like widespread mortality of benthic invertebrates, loss of seagrass meadows, shift biomass allocation or increase in plants and trees mortality [80], [81].

This particular case, it is developed to the city of Greater Lyon. The population of Greater Lyon—59 municipalities with an area of 538 km2—is expected to increase up to 1.5 million inhabitants by 2030 [82]. This growth is accompanied by a rapid change in climatic conditions. In order to cope with this situation, Greater Lyon has drawn up a climate adaptation plan, along five strategic development and innovation axes [83]:

1) **Preserving water resources**: Developing water management systems that improve supply efficiency.
2) **Reducing urban heat-island (UHI) effects**: Increasing the planting of new trees in the region. (Urban Heat Island (UHI) refers to the elevation of air temperature in the urban

area compared to the rural periphery. In large cities; the temperature difference due to this phenomenon is on average 3 to 4°C, but it can reach 8°C in a heat wave period… [84])

3) **Better assisting the population**: by planning for heat waves with follow-up indicators.
4) **Adaptation of agricultural structures and practices**: accompanying measures to move towards conservational agriculture.
5) **Improving local knowledge**: by better understanding the impact of climate change on local scales, understanding the local biodiversity/species and the needs of the territory and its inhabitants.

The case scenario is directly related with this climate adaptation plan, especially with points 1), 2) and 3); the new trees planted in Lyon are equipped with temperature sensors and irrigation actuators, being needed a system that informs about when a heatwave is present on the city allowing the actuators to irrigate the right amount of water to prevent mortality of the new trees.

The first question that needs to be answered, is what conditions must be present to announce a heatwave? There are different possible answers to this question, because it actually depends on the purpose of the application; for humans' mortality for example, it is particularly important the highest temperature, being not really transcendental heat waves that appears on the winter season. However, in this case, trees will require a percentage more of water if a heatwave is present even if it is winter, allowing to do not irrigate too much water when these conditions are not reached. For this reason, in this case, the working definition selected is provided by John R. Nairn, which is based on the concept of the Excess Heat Factor (EHF):

"The EHF is a new measure of heatwave intensity, incorporating two ingredients. The first ingredient (EHI_{sig}) is a measure of how hot a three-day period (TDP) is with respect to a monthly temperature threshold. If the daily mean temperature (DMT) averaged over the TDP is higher than the climatological 95th percentile for DMT (hereafter T_{95}), then the TDP and each day within in it are deemed to be in heatwave conditions… The second ingredient (EHI_{acc1}) is a measure of how hot the TDP is with respect to the recent past (specifically the previous 30 days), taking into account the idea of acclimatization to their local climate" [78]. Following with this explanation, the formulas that allow the calculation are provided below [78]:

$$EHI_{sig} = \frac{T_i + T_{i-1} + T_{i-2}}{3} - T_{95} \tag{1}$$

$$EHI_{acc1} = \frac{T_i + T_{i-1} + T_{i-2}}{3} - \frac{T_{i-3} + \dots + T_{i-32}}{30} \tag{2}$$

$$\mathrm{EHF} = \mathrm{EHI_{sig}} \times \max\left(1,\ \mathrm{EHI_{acc1}}\right) \tag{3}$$

Hence, a heatwave will be announced if three consecutive days the Excess Heat Factor is superior to cero. This definition adapts perfectly to our case because it not only compares the temperatures with the monthly threshold that will detect heatwaves even on the winter season, but also with the temperatures of the last month, providing an idea if an abrupt change is produced, which is quite interesting because the trees are more sensible to abrupt changes of temperature than if high temperatures are reached progressively.

5.1.2. Case 1: Justification

There are several reasons that justify why this particular case is developed. Firstly, it requires the development of an IoT application, providing some ideas about how the Integrated Development Environment may respond to a real problem and giving insights about its effectiveness or limitations.

Secondly, because of the existence of real data that is directly accessible to the author. Greater Lyon in order to become an innovative, sustainable and vibrant smart city; works actively with the bIoTope Project and the ICT for Sustainable Manufacturing Group of the École Polytechnique Fédérale de Lausanne, where this Master Thesis has been developed. The opportunity to work with real sensor data, in concrete with the temperature sensors of eleven trees distributed by the city of Lyon that informs about the temperature each half an hour, is extremely appealing since it closes the gap between the case scenario and the real conditions.

Finally, the topic is also interesting due to the fact that it addresses really contemporary issues as the climate change and how cities have to adapt to the new conditions to provide its citizen the best conditions of life possible.

5.1.3. Case 1: Possible Solution

Following with the strategic plan of Greater Lyon that involves preserving water resources, reducing urban heat-island (UHI) effects by planting trees or assisting the population with heat waves indicators, an IoT system must be developed.

The IoT system would acquire the data that comes from the temperature sensors that are situated in different trees distributed around Lyon, once the data arrives it will be converted to the right format in order to allow posterior processing and calculations. The different temperatures will be then stored on a MongoDB database.

Storing the temperatures on a database is the simplest way to be able to use them later. In our case, it is extremely important, because the calculation of the EHF requires the information of the temperatures of the previous days. Once the EHF is calculated using just the temperatures of a time range, it will also be stored on the database, then the EHF of the last three says are evaluated to state whether the metropolis of Lyon is immersed on a heat wave.

The results obtained could be used by the different actuators to irrigate more water just the days in which the conditions are reached, resulting on a preservation of water without compromising the integrity of the trees planted to reduce the urban heat-islands effects. Moreover, the Excess Heat Factor may be provided to the population since it informs about the intensity of the heat wave.

In order to draw conclusions about the IDE's behavior, all the functions required to complete the IoT application will be created in the Integrated Development Environment and then implemented in the Node-RED flow considering which would be the differences if it had not been developed.

5.1.4. Case 1: IDE Functions

Several functions were needed to get to the solution proposed. In this subchapter, a basic description of each one of them is presented:

1) **Working function**: This function assesses if the node is receiving data from the temperature sensors, informing about the state of the connection.
2) **Extraction of the code**: This is a fundamental function, basically in charge of retrieving just the useful information that comes from the temperature sensors under the proper format (i.e. dates are retrieved in the format YYYY/MM/DD for querying, or temperatures in number format not string for calculations). The data provided by Greater Lyon informs about different characteristics, like the model of the sensor or the person responsible of its maintenance. In our case, it is needed the temperature sensed to posterior calculations, date and time of measurement or name of the sensor to facilitate querying and latitude/longitude to allow maps representation.
3) **Connecting MongoDB**: Python function provided by the official site of MongoDB to make use of the database.
4) **Insert Mongo**: This function stores the data received in one collection and database predefined. The same function is used for storing the data that comes from both, the Extraction of the code and the Excess Heat Factor functions.
5) **Read Mongo Temperatures**: Another important function for the correct solution of the case. This function, in the first place, waits two seconds to be able to read from Mongo even information that has just been introduced, then it is developed a mechanism to get the following date of the day of the execution and the date 34th day before with the format YYYY/MM/DD that allows querying with the use of less than ($lt) and greater than ($gt); afterwards, it is possible to perform a query that shows all the temperatures during this range of dates for a sensor and range of time determined by the user. In this case, it is being used the sensor number 8, in the range of time from 8:00 and 8:20 am. The function will return the temperatures along with dates and times of measurement.
6) **Excess Heat Factor**: This function performs the calculations needed to obtain the Excess Heat Factor of the day. This is achieved by organizing the temperatures received after reading from MongoDB, comparing the temperatures of the last three days first with the 95th percentile of temperatures during this month in the last twenty years, already defined by the KNMI Climate Explorer [85], and also with the temperatures of the prior thirty days. Lastly, it multiplies the results, just as it was defined by formulas (1), (2), (3). The function returns the value of the EHI_{sig}, EHI_{acc1}, EHF and the date.
7) **Read Mongo EHF:** Similarly, to the Read Mongo Temperatures function, an initial wait is performed and a mechanism to get the dates of the last three days is created. Then a query is performed to return the EHF of the last three days along with the dates.
8) **Heatwave:** The last IDE function receives the Excess Heat Factor from the last three dates and assesses the values, if all the values are bigger than cero the system informs that a heat wave is present, allowing the control of the respective actuator.

This set of functions can be found on *Annex VI* (*Application Case 1: Heat Wave Mitigation, Greater Lyon-IDE Functions*).

After developing these functions, some initial ideas can be extracted about the real utility of the IDE. The first important thing to notice is how it facilitates the detection of errors; once an error is made on the IDE, the information about the error and the line where it can be found is displayed. To get an idea, the same error is made using the python module of Node-RED, the result was more than 10 messages with no sense that did not inform about where the error was committed.

Another advantage detected is that the error is decoupled. When the Node-RED flows incorporate different elements as a MongoDB database, sensor data with different formats or Python functions, they are prone to get errors from different nodes. Without the existence of the IDE it is needed to deploy the whole flow to visualize if the Python function is developed correctly, finding maybe another error that has nothing to do with the Python function. Using the new IDE, it is easy to assure that the function works correctly.

Regarding the inconveniences, it is also needed to comment one that was already known. There are different modules that are not yet implemented in the IDE. When the function Insert Mongo was tested on the IDE, the output displayed that the module pymongo was not yet implemented. Meaning that even that it is possible to write the function on the IDE it cannot be tested normally. However, and expressing clearly that this is a limitation, it does not have a vital importance in this case, since the contents of the function can also be tested outside the Node-RED flow accessing MongoDB from the command line.

5.1.5. Case 1: Node-RED Flow

The Node-RED flow that provides the solution to the first case scenario is presented in two figures to facilitate visualization and explanation of each of the nodes developed.

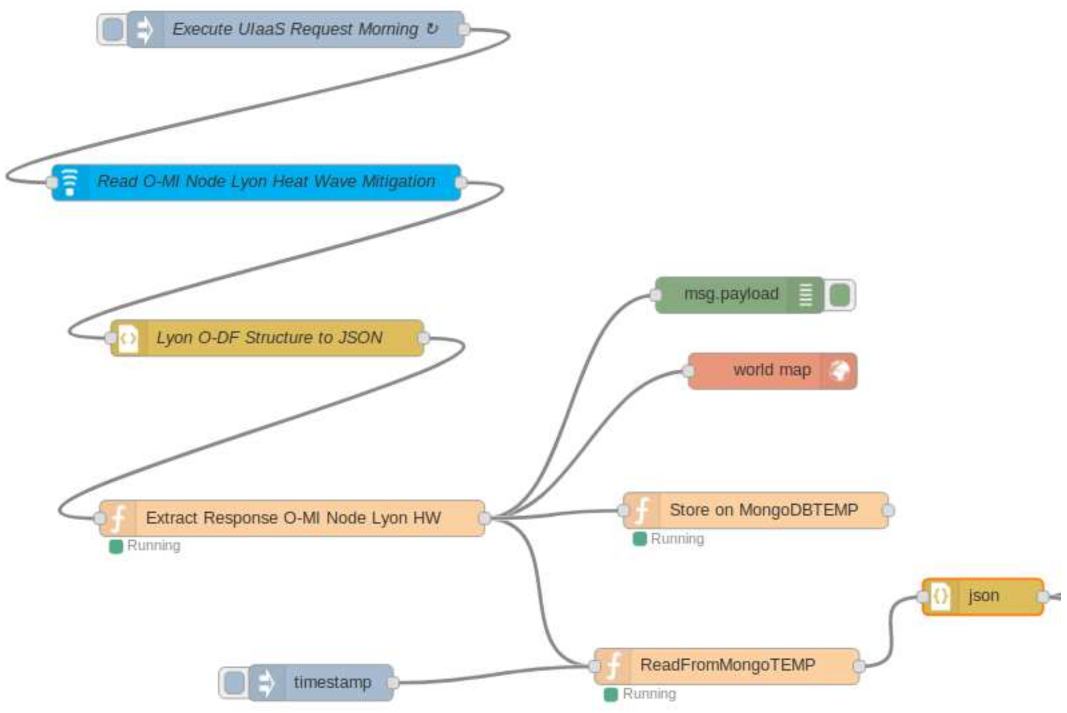

Execute UIaaS Request Morning ↻
Read O-MI Node Lyon Heat Wave Mitigation
Lyon O-DF Structure to JSON
msg.payload
world map
Extract Response O-MI Node Lyon HW
Running
Store on MongoDBTEMP
Running
json
timestamp
ReadFromMongoTEMP
Running

Figure XVI. Case1: Node-RED Flow I

The description of the nodes of Figure XVI, is presented below:

1) **Execute UIaaS Request Morning:** The first node, executes the flow automatically every single day at 8:00 am.
2) **Read O-Mi Node Lyon Heat Wave Mitigation:** This node is able to read the objects of the specified web site. In this case, involves the information provides by the sensors.
3) **Lyon O-DF Structure to JSON, json:** Node already predefined by Node-RED to convert objects of different formats to the JSON format.
4) **Extract Response O-MI Node Lyon HW:** This node imports the IDE functions 1) and 2), confirming the data is received and extracting just the useful data received by the sensors.
5) **Timestamp:** Alternative solution to execute the flow manually to visualize the results at any moment.
6) **Msg.payload:** Shows in the console, the information extracted (temperatures of the different sensors with their latitude/long, date and time of measurement.
7) **World map:** Represents in a map the position of the sensors indicating the temperatures.
8) **Store on MongoDBTEMP:** This node uses the IDE functions 3) and 4) to store in a MongoDB collection the extracted data.
9) **ReadFromMongoTEMP:** Imports IDE functions 3) and 5) to read from the database the information of the temperatures of the last 33 days of a determined sensor, including the one that has just been added.

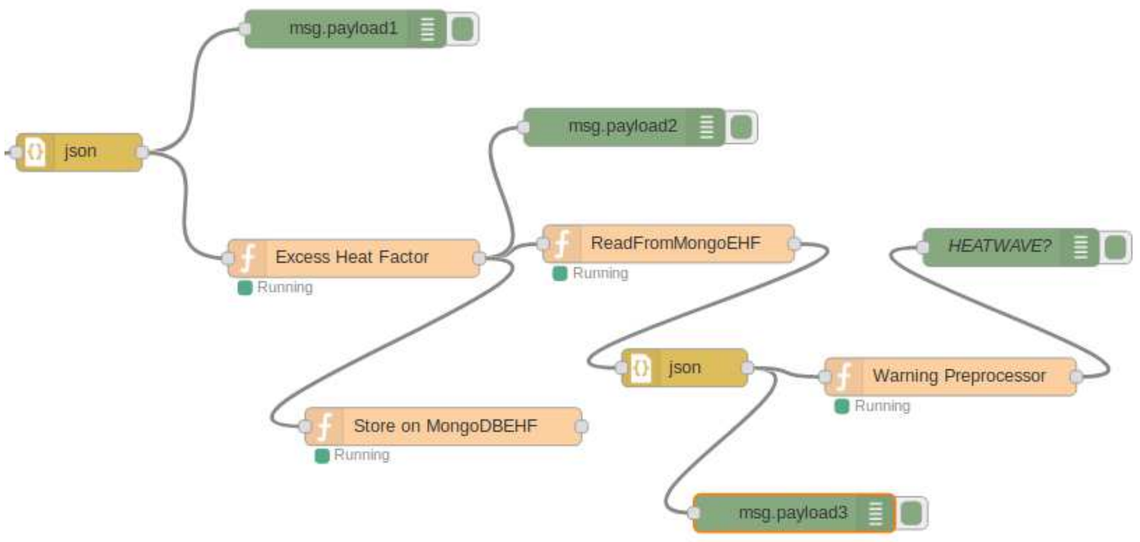


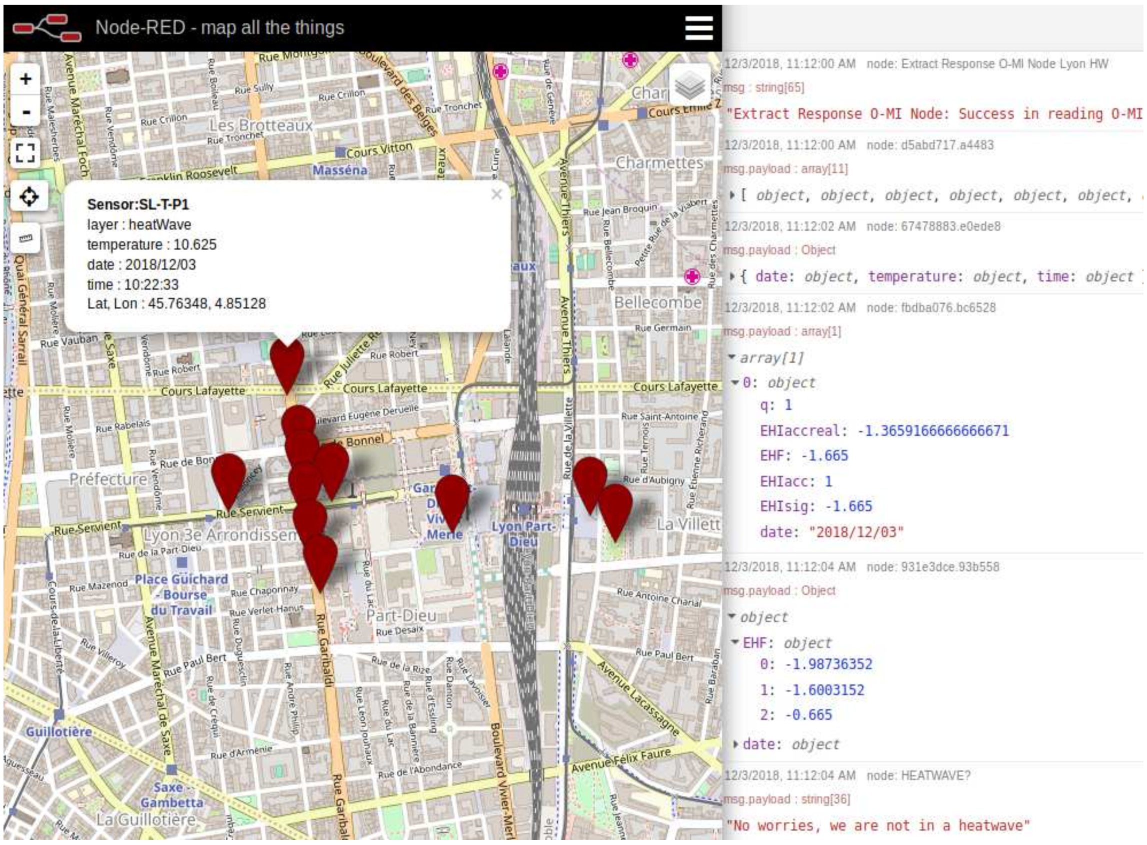
Figure XVII. Case1: Node-RED Flow II

Regarding Figure XVII:

10) **Msg.payload1**: Displays in the Node-RED console the temperatures of the last 33 days with the date and time of the different measures.

11) **Excess Heat Factor**: In this node, the 95th percentile of the temperatures of the different months in the last 20 years is defined [85], and then makes use of the IDE function 6) to perform the needed calculations to obtain the corresponding EHF to the particular day.
12) **Msg.payload2:** Displays the different values obtained for the day, that may be interesting for the citizens as the EHF, EHI_{sig} (difference respect 95th percentile), EHI_{acc1} (difference with the mean temperature of the previous 30 days).
13) **Store on MongoDBEHF:** This node uses the IDE functions 3) and 4) to store in a MongoDB collection the different values calculated.
14) **ReadFromMongoEHF:** Imports IDE functions 3) and 7) to read from the database the EHF values of the last 3 days, including the one that has just been added.
15) **Msg.payload3:** Assures that the reading from the database has been completed with success.
16) **Warning Preprocessor:** This node imports the IDE function number 8), to evaluate the different EHF values to determine if there is a heatwave and the actuator of the corresponding tree should irrigate more water.
17) **Heatwave?**: The last node of the flow informs about the result obtained through the Node-RED console.

The contents of each one of these nodes can be visualized on *Annex VI (Application Case 1: Heat Wave Mitigation, Greater Lyon-NodeRED Flow)*. After, the execution of the flow on 3rd of December of 2018, the following result (Figure XVIII) of the Case 1 is obtained:



Sensor:SL-T-P1
layer : heatWave
temperature : 10.625
date : 2018/12/03
time : 10:22:33
Lat, Lon : 45.76348, 4.85128

```
12/3/2018, 11:12:00 AM   node: Extract Response O-MI Node Lyon HW
msg : string[65]
"Extract Response O-MI Node: Success in reading O-MI
12/3/2018, 11:12:00 AM   node: d5abd717.a4483
msg.payload : array[11]
▸[ object, object, object, object, object, object,
12/3/2018, 11:12:02 AM   node: 67478883.e0ede8
msg.payload : Object
▸{ date: object, temperature: object, time: object
12/3/2018, 11:12:02 AM   node: fbdba076.bc6528
msg.payload : array[1]
▾array[1]
 ▾0: object
    q: 1
    EHIaccreal: -1.3659166666666671
    EHF: -1.665
    EHIacc: 1
    EHIsig: -1.665
    date: "2018/12/03"
12/3/2018, 11:12:04 AM   node: 931e3dce.93b558
msg.payload : Object
▾object
 ▾EHF: object
    0: -1.98736352
    1: -1.6003152
    2: -0.665
 ▸date: object
12/3/2018, 11:12:04 AM   node: HEATWAVE?
msg.payload : string[36]
"No worries, we are not in a heatwave"
```

Figure XVIII. Case 1: Node-RED Result

5.2. CASE 2: SMART PARKING SYSTEM, FIFA WORLD CUP QATAR 2022

5.2.1. Case 2: Problem

The second case scenario is framed on the context of the future football world cup that will take place on Qatar in 2022 organized by the *Fédération Internationale de Football Association (FIFA)* [86]. Although hosting a tournament of these characteristics may not be worth the investment, it is clear, that it is not completely pointless, in accordance with the World Economic Forum article (2018)*: Does hosting a World Cup make economic sense?,* "hosting a World Cup, or any other major sporting event, can boost a nation's economy by attracting tourists, initiating important infrastructure projects and showcasing countries and cities as good places to do business" [87].

Qatar, the country with the highest income per capita in 2017, according to the International Monetary Fund [88], has started to develop the infrastructure necessary to host the event with the most advanced technologies; for example new transportation initiatives as the construction of new roads, metro lines or a national long-distance rail [89], or the implementation of a district cooling system in the stadiums to fight against the high temperatures with lower energy consumption [90].

Under this situation, Qatar has also developed a Parking Master Plan 2017, since the Qatar World Cup 2022 will require adequate smart parking systems and facilities around the stadiums and world cup facilities that maximizes user experience [91]. One idea to solve this problem was presented by K. Främling et al. (2017) in the article: *Open IoT Ecosystem for Sporting Event Management,* explaining how an IoT system could be beneficial for large sport events that may require the assistance of emergency units [92].

Through this case scenario, the idea presented by K. Främling et al. will be developed as an IoT system on Node-RED. In particular, it is addressed the Ras Abu Aboud stadium that is being constructed closed to the port of Doha, directly from ship containers and that will have a capacity of 40.000 spectators and 6.000 parking places [93]. In order to continue with the case, some considerations have to be done:

1) The stadium will have three access gates with sensors able to read the plate of the vehicles that will also lead to three zones of 2000 parking places, situated on the sides south, east and west of the stadium. Their coordinates (longitude/latitude) are static and well known.
2) The stadium tickets may be categorized in five zones according to their geographical location.
3) For simplicity, this case is developed considering only three emergency units as (firefighters, police, and ambulances) and two different vehicles of each unit apart from the ones that are already occupied.
4) The different emergency vehicles are considered smart objects, since they have location sensors. The data provided by the vehicles that are available to go, will be accessible to the stadium gates when an alarm occurs.
5) It is also considered that the parking place can be bought at the same time that the ticket for the stadium, or at least, that when it is time to get the parking place the system can access to the data of the stadium ticket.

6) In order to get a response of the gate, it is simulated that the World Cup has taken place on December 2018 and that this stadium is hosting two matches of the groups phase, one quarterfinal and one semifinal.

The solution proposed by K. Främling et al. (2017) is focused on leveraging not only user experience by allocating the parking closely to the seating position in the stadium and incrementing the speed at the entrance and exit of the stadium, but also improving emergency management by reducing times to action.

5.2.2. Case 2: Justification

The second case implies an IoT solution of even more complexity to test the behavior of the Integrated Development Environment with Node-RED. In concrete, it will be needed the development of several flows to differentiate the entrance of the different kind of vehicles. Moreover, it will require the use of Python functions to perform more complicated mathematical calculations.

This case is also interesting since the point of view of how different units can work along with each other to provide a better service. This is one of the biggest challenges of IoT, being able to provide an open environment where the data can be accessed not only by a few, breaking the denominated vertical silos [7]. This is especially plausible in smart cities, like in this case, the data that provides for example an ambulance, normally to the hospital, can be used by another unit (the stadium) to improve one service as important as it is reducing the time to action when an emergency occurs.

Ultimately, the idea of a smart parking system is applied in the context of the FIFA World Cup, one of the most important sport events, giving it a purpose and a background particularly motivating for the author of the project.

5.2.3. Case 2: Possible Solution

The solution proposed may be structured in two parts. The first one is focused on leveraging the spectator experience, while the second one is related with improving emergency management.

In order to improve the spectator experience; if one spectator decided to have a parking place for a determined match, he would indicate it by paying the associated price and would add his plate number, then the parking system would read the information about its stadium ticket using his personal information as name or identity document. According to the category (place of the stadium) a parking spot in the closest zone is reserved to the spectator, informing him to which gate he should go. When the day of the match comes, and the spectator goes to the specified gate, the gate opens automatically reading the contents of the car plate, without spending time showing the ticket to one guard or equivalent. The process is repeated when the spectator leaves the stadium. The same spectator will not have access any other day to the stadium or to any other match if he has not buy another parking place.

The second part is related with the emergency units; the idea is that when an emergency happens in the stadium and an emergency vehicle needs to access, it may be done in the fastest way. So, when an alarm sounds, the stadium gets the info of the different vehicles operatives of the related emergency unit and selects which one is closest, this vehicle would receive the information of the gate that is closest to their position. When the vehicle arrives to the stadium a sensor reads the plate of the vehicle and as the stadium already has the info of that vehicle it can get in.

Note: Even though the case has been developed trying to represent in the most accurately way the reality of the problem, the solution provided cannot assure the complete achievement of these goals. It is important to underline, that the real objective of the case scenarios is testing and draw conclusions about the new environment developed.

5.2.4. Case 2: IDE Functions

The second case scenario requires the use of IDE functions to deploy successfully the IoT application. These IDE functions can be encountered on *Annex VII (Application Case 2: Smart Parking System, FIFA World Cup Qatar 2022-IDE Functions*). Some of them have similar resemblances to the ones developed on Case 1 as:

1) **Working function**: This function, as in the first case, assesses if the node is receiving data from the sensors; in this occasion, the sensors from the different emergency units.
2) **Extraction of the code**: Although in this case there is not real sensor data, it has been created a node that simulates the real conditions, following with the example of the Case 1. This function extracts just the relevant information as the longitude/latitude of the different vehicles to evaluate the distances, the plate code or the date and time to allow the access to the stadium just for that day, or information as the name of the responsible person to facilitate communication.
3) **Connecting MongoDB**: Python function provided by the official site of MongoDB to make use of the database.
4) **Insert Mongo**: This function stores the data received in one collection and database predefined. In this case, another collection of the MongoDB database is used, both for storing data like plate number or access gate from spectators and emergency units that will have access to the stadium.

On the other side, the completely new functions developed for this particular case are:

5) **Spectator Function**: This function is in charge of assigning one parking spot to any spectator that desires it, when parking places are still available. If these conditions are met the function reads the contents of its ticket, returning the date according to the respective match and the parking zone adequate to the category of the ticket, assuring the proximity between the parking place and the respective seat in the stadium. The plate of the car introduced by the user is returned as well.
6) **Haversine**: Compares the coordinates of the different emergency vehicles available with the coordinates of the stadium, selecting and communicating which vehicle is closer. This can be done making use of the haversine formulas (The haversine formula is an equation important in navigation, giving great-circle distances between two points on a sphere from their longitudes and latitudes. [94]):

$$x = lat_1 - lat_2 \tag{4}$$

$$y = lon_1 - lon_2 \tag{5}$$

$$h = \sin^{-1}\sqrt{\sin^2\left(\frac{x}{2}\right) + \cos(\mathrm{lat}_1) \times \cos(\mathrm{lat}_2) \times \sin^2\left(\frac{y}{2}\right)} \tag{6}$$

$$distance = 2 \times r \times h \tag{7}$$

Where *lat* refers to latitude, *lon* to longitude, *r* represents the radius of the Earth (it is considered the mean radius = 6371 km) and the sub-index differentiate the points in which the distance is being calculated.

7) **Right Gate:** This function is responsible for communicating the different vehicles which gate is closer to them and its coordinates. Again, the haversine formulas (4), (5), (6), (7), are employed to calculate the distances that allow comparing the results.
8) **Access Parking:** Actually, this is not a single function but three of them, since there is one function for each one of the gates. The functions evaluate the contents of the car plates thanks to the sensor located on each of the gates, if the respective plate code has access that particular day on that particular gate, then the gate is open, this is checked querying for the plate number, day and gate number. If no result appears, the car will have to leave by the exit provided to that concrete purpose. In the case of emergency units, it is considered that they are able to access by any gate since the priority is the resolution of the emergency.

Some new insights about the performance of the Integrated Development Environment can be underlined. For example, during the development of the Haversine function, the employment of the IDE and particularly the interactive editor resulted very beneficial because it allowed displaying the results of the calculations automatically on the screen, visualizing if the result makes sense, detecting in several occasions that even there was not an error on the code, the formula was not correctly integrated and the results were far from the correct ones. Without this tool, it would have been much more difficult to detect this kind of mistakes since the code would run along with the rest of the flow proving just the final solution.

Another positive aspect to comment, it is how the python module working along with the IDE can leverage the extensibility of Node-RED. In complex cases like this one that require mathematical computations or arrange the data in specific ways, it is very useful to use Python tools as the math or the numpy modules, which are already implemented on the IDE, instead of developing a really complex JavaScript function.

It is possible to note as well, how the google iframe can facilitate the development of new functions. In this case, it was very convenient to search for the haversine formulas in order to implement them on the editor. It is true that another window of the browser could have been used, but it would have been needed to be moving from one window to the other constantly.

Regarding negative insights, it is noticed that the performance of the IDE is affected when the data that must be processed cannot be reproduced. For example, when it comes to extracting the data of the different emergency vehicles, the function can be developed normally, and it can be checked that no errors have been committed, however, it is difficult to know how it is actually extracting the data until it is tested on Node-RED. That is because the data comes in a format that complicates its reproduction, being impossible to get the output on the IDE site.

5.2.5. Case 2: Node-RED Flow

Following with the information described, the Node-RED flow that attempts to reduce the time to action when an emergency not foreseen occurs is represented on Figure XIX:

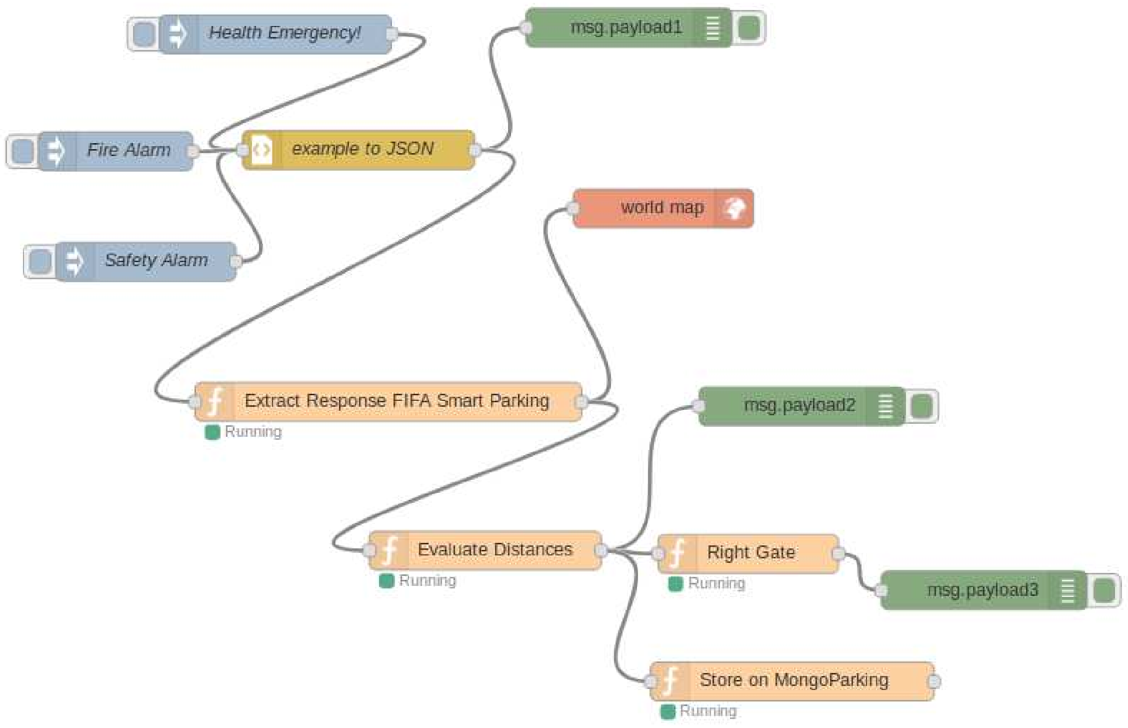


Figure XIX. Case 2: Node-RED Flow I

1) **Health, Fire, Safety Alarms:** When an emergency occurs, the respective node is executed providing the stadium with the information of the different emergency vehicles of the unit. As in this case not real data is available, the information has been created from scratch, indicating vehicles situated on different parts of Doha with a similar format to the real data used in case 1.
2) **Example to JSON:** Node already predefined by Node-RED to convert objects of different formats to the JSON format.
3) **Extract Response FIFA Smart Parking:** Node that includes IDE functions 1) and 2), confirming that the data is received and extracting just the useful part.
4) **World Map:** Represents the information extracted on the map, thanks to the coordinates of each vehicle.
5) **Evaluate Distances:** This node makes use of the longitude and latitude of the vehicles and along with the Haversine IDE function calculates which vehicle is closer to the stadium. Moreover, it assigns the emergency vehicles a code that allows its access by any gate.
6) **Right Gate:** Determines which access gate of the stadium (east, west or south) is closest to the respective emergency vehicle importing the IDE function Right Gate.
7) **Store on MongoParking:** Although this node just stores the data to the database importing IDE functions 3) and 4), it is fundamental for the correct performance of the

whole system, because it also provides the interconnection between the different flows, since the data stored here can be used as well by the rest of the flows.

8) **Msg.payload:** Displays on the Node-RED console the message it receives. In this particular flow, it is shown the information of the vehicles received, the vehicle that is closer to the stadium and lastly, the gate that is closer to that vehicle with its coordinates. This last message would be sent to the vehicle in order to facilitate its arrival.

Regarding the designation of a parking place, a Node-RED flow that makes uses of the information of the tickets to assign a close parking spot is developed, just as it is represented on Figure XX:

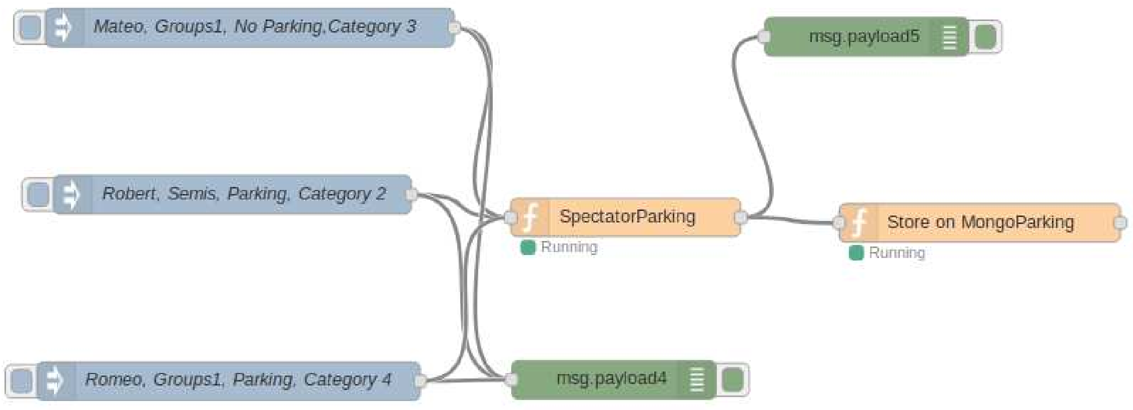

Mateo, Groups1, No Parking,Category 3
msg.payload5
Robert, Semis, Parking, Category 2
SpectatorParking
Running
Store on MongoParking
Running
Romeo, Groups1, Parking, Category 4
msg.payload4

Figure XX. Case 2: Node-RED Flow II

9) **Mateo, Robert, Romeo Nodes:** These nodes simulate the state of different persons that have bought a ticket for a match and are considering having a parking spot. It provides information about match, category of the ticket, willingness to have a parking place and plate number. In definitive, these nodes inform of different possibilities to see how the system responds.
10) **Spectator parking:** This node imports IDE function 5), according to the information received that is introduced by the user, it assigns a place on the parking zone that is closest to the category for the day of the respective match, always, that the user wanted a place and there were free spots. It returns the plate code introduced with the user along with the gate and date when the spectator will have access.
11) **Store on MongoParking:** As node 7), this node stores the data on the MongoDB collection in order to be accessible by the gates of the stadium when a vehicle wants to pass.
12) **Msg.payload**: Displays information of the tickets and the information about the respective parking spot.

A final flow is developed to simulate the behavior of the gates in presence of different vehicles (Figure XXI). The plate read by the sensor of the gate is contrasted with the codes that have access for that determined match on that gate, opening well the gate or instead the exit provided by that purpose. It is tested that plates that would have access for another match cannot enter, or that emergency units only have access to any of the doors on the day when the emergency occurs.

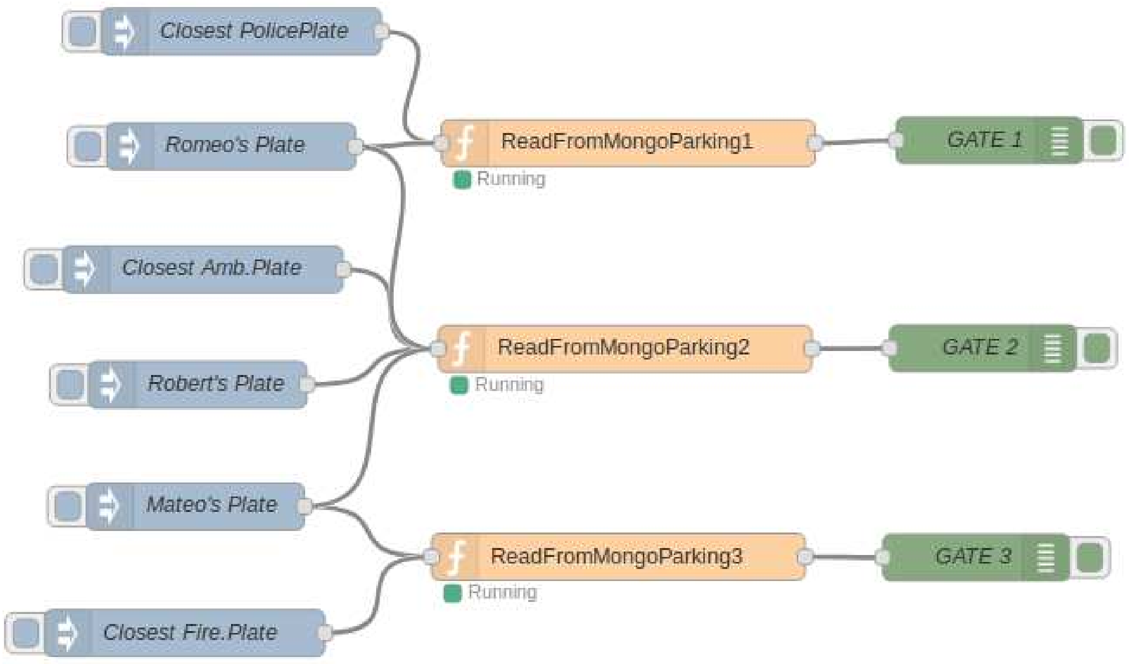


Figure XXI. Case 2: Node-RED Flow III

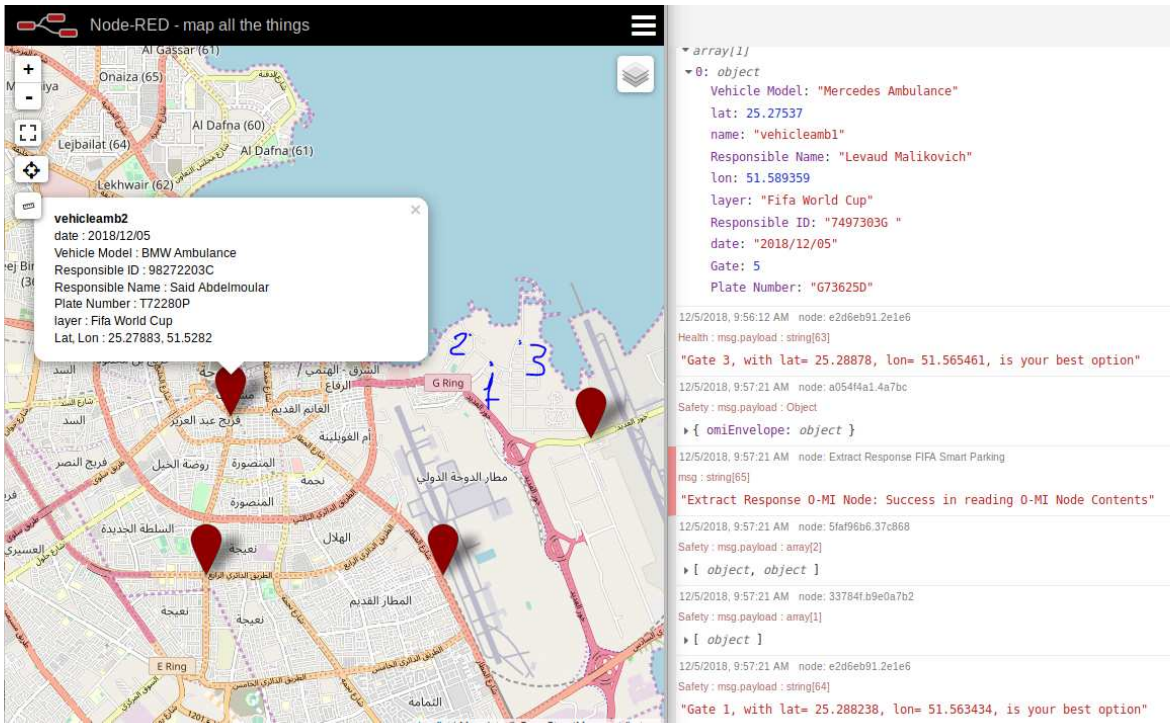
13) **Name Plate:** Node that is executed when a car goes to the gate and its plate number is read by the sensor of the gate.
14) **ReadFromMongoParking:** Makes use of the IDE function 8), querying for the plate number received, the respective gate and the day when the flow is executed, it is able to decide if the gate opens or not.
15) **Gate**: Shows by console the decision.

This set of nodes can also be observed on the Annex Section (*Annex VIII. Application Case 2: Smart Parking System, FIFA World Cup Qatar 2022-NodeRED Flow)*.

To verify the right behavior of the system, a final simulation is performed. It is considered that the day of the simulation, 5th of December of 2018, the first match of the groups phase is disputed on the Ras Abu Aboud Stadium. Moreover, during the match is required the entrance of more police and ambulance units.

In this way, Figure XXII shows the results of the flow related with the emergency units, displaying where the different ambulances and police car are situated. In base to that, the flow identifies which vehicles are closer and advised them about its closest access gate.

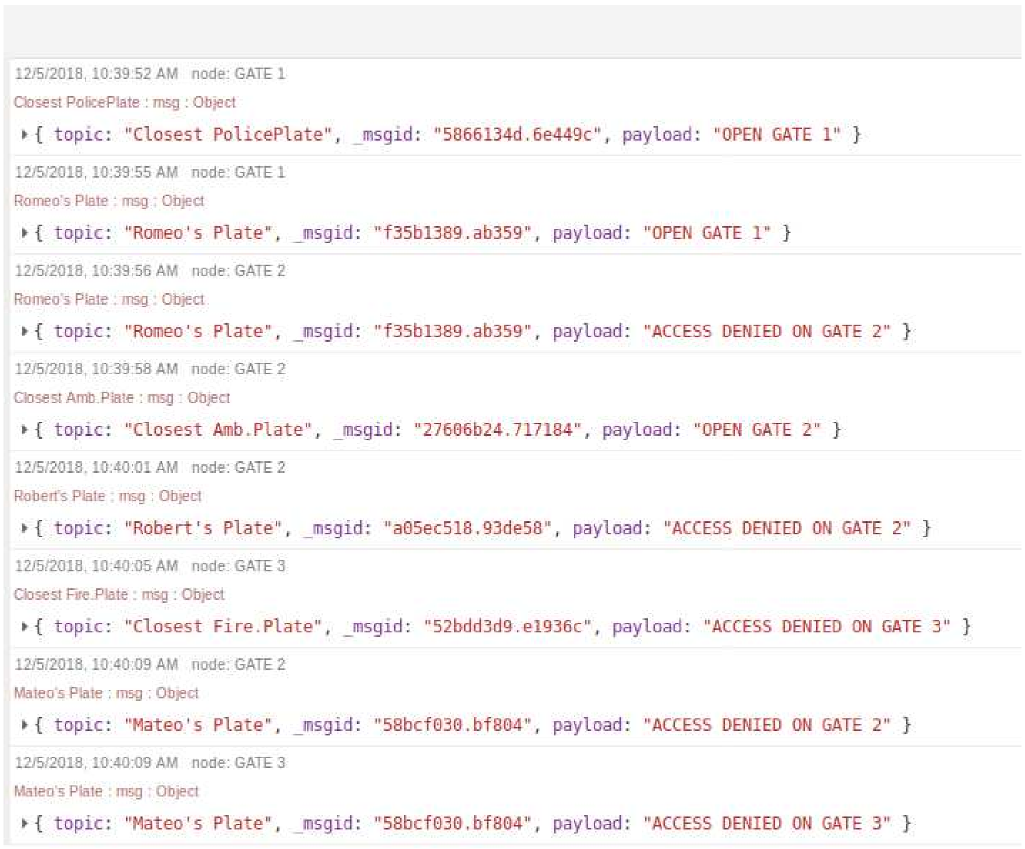
Meanwhile, Figure XXIII represents the action of the gate when different vehicles approach it, According to Figure XIX, Romeo's car has a parking place for that match on Zone 1, Mateo has a ticket for that match but has not a parking place and Robert has a parking place on Zone 2 but for the semifinals. It can be observed that both the police and ambulance vehicles can access the stadium by any of the gates, however the firefighters, that have no emergency into the stadium, cannot access. Regarding the spectators, only Romeo who has a valid parking place for that match could access by its respective gate.



vehicleamb2
date : 2018/12/05
Vehicle Model : BMW Ambulance
Responsible ID : 98272203C
Responsible Name : Said Abdelmoular
Plate Number : T72280P
layer : Fifa World Cup
Lat, Lon : 25.27883, 51.5282

```
▾array[1]
 ▾0: object
    Vehicle Model: "Mercedes Ambulance"
    lat: 25.27537
    name: "vehicleamb1"
    Responsible Name: "Levaud Malikovich"
    lon: 51.589359
    layer: "Fifa World Cup"
    Responsible ID: "7497303G "
    date: "2018/12/05"
    Gate: 5
    Plate Number: "G73625D"
12/5/2018, 9:56:12 AM   node: e2d6eb91.2e1e6
Health : msg.payload : string[63]
"Gate 3, with lat= 25.28878, lon= 51.565461, is your best option"
12/5/2018, 9:57:21 AM   node: a054f4a1.4a7bc
Safety : msg.payload : Object
 ▸{ omiEnvelope: object }
12/5/2018, 9:57:21 AM   node: Extract Response FIFA Smart Parking
msg : string[65]
"Extract Response O-MI Node: Success in reading O-MI Node Contents"
12/5/2018, 9:57:21 AM   node: 5faf96b6.37c868
Safety : msg.payload : array[2]
 ▸[ object, object ]
12/5/2018, 9:57:21 AM   node: 33784f.b9e0a7b2
Safety : msg.payload : array[1]
 ▸[ object ]
12/5/2018, 9:57:21 AM   node: e2d6eb91.2e1e6
Safety : msg.payload : string[64]
"Gate 1, with lat= 25.288238, lon= 51.563434, is your best option"
```

Figure XXII. Case 2: Node-RED Result I



```
12/5/2018, 10:39:52 AM   node: GATE 1
Closest PolicePlate : msg : Object
 ▸{ topic: "Closest PolicePlate", _msgid: "5866134d.6e449c", payload: "OPEN GATE 1" }
12/5/2018, 10:39:55 AM   node: GATE 1
Romeo's Plate : msg : Object
 ▸{ topic: "Romeo's Plate", _msgid: "f35b1389.ab359", payload: "OPEN GATE 1" }
12/5/2018, 10:39:56 AM   node: GATE 2
Romeo's Plate : msg : Object
 ▸{ topic: "Romeo's Plate", _msgid: "f35b1389.ab359", payload: "ACCESS DENIED ON GATE 2" }
12/5/2018, 10:39:58 AM   node: GATE 2
Closest Amb.Plate : msg : Object
 ▸{ topic: "Closest Amb.Plate", _msgid: "27606b24.717184", payload: "OPEN GATE 2" }
12/5/2018, 10:40:01 AM   node: GATE 2
Robert's Plate : msg : Object
 ▸{ topic: "Robert's Plate", _msgid: "a05ec518.93de58", payload: "ACCESS DENIED ON GATE 2" }
12/5/2018, 10:40:05 AM   node: GATE 3
Closest Fire.Plate : msg : Object
 ▸{ topic: "Closest Fire.Plate", _msgid: "52bdd3d9.e1936c", payload: "ACCESS DENIED ON GATE 3" }
12/5/2018, 10:40:09 AM   node: GATE 2
Mateo's Plate : msg : Object
 ▸{ topic: "Mateo's Plate", _msgid: "58bcf030.bf804", payload: "ACCESS DENIED ON GATE 2" }
12/5/2018, 10:40:09 AM   node: GATE 3
Mateo's Plate : msg : Object
 ▸{ topic: "Mateo's Plate", _msgid: "58bcf030.bf804", payload: "ACCESS DENIED ON GATE 3" }
```

Figure XXIII. Case 2: Node-RED Result II

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

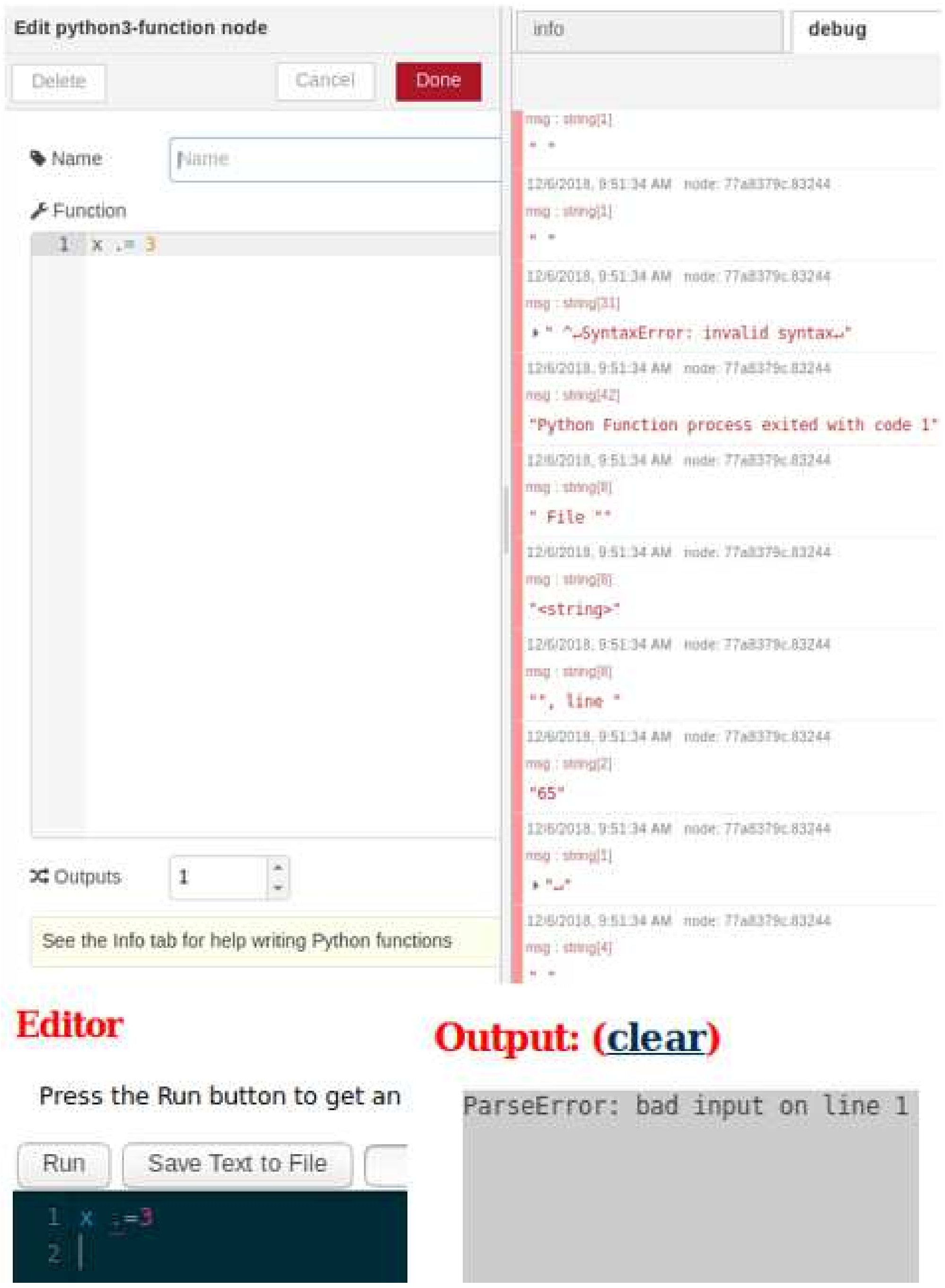
As it has been discussed through the project, the advances achieved on different technologies has provided the unique opportunity to create a dynamic global network infrastructure where a simple vending machine can inform users if it has cold drinks inside, the different trees of a city may communicate if they need more water or a determined vehicle could provide its location at every instant. This is what it is denominated as the Internet of Things, providing, if it is managed correctly, great value to everyone; since people can avoid walking long distances to get to an empty vending machine, water can be saved assuring that trees stay alive or even facilitating that an emergency could be treated in a faster way. Just like these examples, millions of quotidian "things" are connected to the Internet providing solutions on every imaginable field, provoking that nowadays exist more connected devices than humans in the world.

However, extracting the valuable information from these physical objects requires the creation of IoT systems that can manage and process the data in an effective way. In order to provide an open environment that facilitates the creation of these kinds of systems, an Integrated Development Environment was designed and implemented with Node-RED. But, is the resultant solution actually meeting its goal? Does it facilitate the creation of IoT applications, increasing developer's productivity?

Now it is exposed, in base to the experience gained and the results obtained during the application case scenarios, the reasons that attempt to justify why the answer to these questions presented should be in both cases affirmative.

In the first place, it is needed to comment how easy is to implement the developed solution. Even great advances may not be adopted if they require a lot of effort or complex tasks to make use of them. In our case, just as it is shown on *Figure XIV. Using IDE Function on Node-RED,* a new function created on the IDE can be used on Node-RED just by writing one line that imports the function to the node.

Secondly, it is remarkable how using the IDE facilitates the detection of errors and the resolution of them. In order to justify this conclusion, Figure XXIV presents how a simple error (x .=3) is treated without the use of the IDE (displaying more than ten messages that do not provide any value) and making use of it (displaying one but effective message that informs about which is the reason of the error and about the line where it is located), moreover, on the code editor it is underlined with red signs that the use of the point makes no sense. Hence, can be imaginable how adopting the IDE is beneficial when it comes to develop complex IoT applications that are prone to the presence of minor errors like this one. Also, the utilization of this new tool may be really interesting to naïve users that are introducing themselves to the creation of IoT systems, to get some practice and solve their code errors.


Edit python3-function node
Delete
Cancel
Done
Name
Function
Outputs
See the Info tab for help writing Python functions
info
debug
Editor
Press the Run button to get an
Run
Save Text to File
Output: (clear)

```
x .= 3
```

```
msg : string[1]
" "
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[1]
" "
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[31]
▸ " ^↵SyntaxError: invalid syntax↵"
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[42]
"Python Function process exited with code 1"
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[8]
" File ""
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[8]
"<string>"
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[8]
"", line "
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[2]
"65"
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[1]
▸ "↵"
12/6/2018, 9:51:34 AM   node: 77a8379c.83244
msg : string[4]
" "
```

```
x .=3
```

```
ParseError: bad input on line 1
```

Figure XXIV. Allocating error

Another reason related with increasing the productivity of the developers detected, it is that the error can be decoupled. In complex IoT systems that might incorporate several elements as a MongoDB database, sensor data with different formats or Python functions, it is normal to find errors from different nodes. Without the existence of the IDE, it is needed to deploy the whole flow to visualize if the Python function is developed correctly, finding maybe another error that has nothing to do with the Python function, without ensuring that is actually well developed. Using the new IDE, the error is decoupled, it is possible to assure that the Python function works correctly before including it on the Node-RED flow that may contain other errors.

It is also well deserved to emphasize other features provided by the Integrated Development Environment that facilitate the different tasks performed by the developers. For example, the interactive code editor, which not only displays the errors line by line but also executes the code, providing instant results of the calculations performed, indicating if errors not related with the syntax are also committed (without the existence of the IDE, it would have been much more difficult to detect this kind of mistakes since the code would run along with the rest of the flow showing just the final solution, indicating maybe that one vehicle is closer than the other, without noticing that the distances calculated have values around billions of kilometers). Or how useful, may be to look for more information just by using the google iframe already implemented on the site, this information searched can be easily introduced on the code editor without moving from different browser windows.

Lastly, the different IDE functions get stored and can be accessible and used for different cases and IoT applications. This is an important point, without the IDE, developers need to create functions from scratch or, in the best case, to search for the Node-RED flow if they have it saved, importing it in some way and, after that, copying and pasting the node to the new flow. With the use of the IDE, things are quite easier, they just require to import again the function in the new flow to get the results expected.

It is clear, attending to the set of reasons presented, that the new environment facilitates (makes easier) the creation of IoT systems, increasing developers productivity (effectiveness of productive effort), by facilitating the allocation and solution of errors, decoupling errors, reducing times of deployment and providing a testing and searching platform on site, in an effective and simple way.

Despite considering that the objective of the project is achieved, and that, from the general perspective the resultant solution is very positive; there are still several areas that have room for improvement, where the future work could be focused on.

One possible line of development is the translation of more Python modules to JavaScript in order to be able to use them on the IDE. As it has been previously introduced, there are several Python modules that are not yet implemented on the IDE, meaning that even that it is possible to write functions that make use of these modules and take advantage of their functionalities in Node-RED, they cannot be tested normally reducing some of the most important benefits of the IDE. Skulpt developers are already working on this direction, implementing more and more modules; however, it would be good to collaborate with them to include modules that may be more relevant in the IoT framework, as for example, *Pymongo* that provides the driver to make use of MongoDB, or maybe *Scikit-learn* that makes use of state-of-the-art implementations of many well-known machine learning algorithms.

Another cool feature that could be interesting for the new environment is the denominated intelligent code completion. This feature is already implemented on cutting-edge IDEs as Eclipse, providing a powerful tool to speed up coding applications just by understanding the context of the code and getting automated writing before actually typing the functions or variables that are being used.

A third area for improvement has to do with the portability to tablets or mobiles of the new service. Internet of Things is present everywhere at every moment, for that reason it would be extremely appealing to provide the development service that is already totally integrated on the cloud (no requires of any particular aspect related with the computer infrastructure) as an application for mobile or tablets, allowing users to develop IoT applications on every device with much more simplicity.

Summarizing, the future work should be related with the portability, the addition of new features as intelligent code completion and with the implementation of Python modules useful in the Internet of Things framework.

Ultimately, I would like to express some insights from a personal point of view about the development of this Master Thesis, which has supposed both a challenge but also an incredible opportunity to learn.

Despite this project has been developed far away from home, in Switzerland. I disposed of a desk in one of the offices of the *ICT for Sustainable Manufacturing Group at EPFL*, providing me with the great opportunity to treat every single day with an exceptional group of people that made my life much easier, sharing all kind of experiences and making me feel as one more of the group.

Moreover, the present Master Thesis has dealt with one of the fundamental pillars of the denominated Industry 4.0, as it is the Internet of Things. Integrating and putting into practice the knowledge acquired in previous courses such as project management, industrial control or instrumentation along with new IT concepts and technologies that may also be valuable for my future professional career. In definitive, I believe it represents a great epilogue for this postgraduate stage that has provided me with the attitude and confidence necessary to start my career with great enthusiasm.

CHAPTER 7. REFERENCES

[1] P.Guillemin, P. Friess. 2009, “Internet of Things Strategic Research Roadmap”, *The Cluster of European Research Projects,* accessed 27 Nov. 2018, from IoT European Research Cluster Database.

[2] The Carnegie Mellon University Computer Science Department n.d., *The "Only" Coke Machine on the Internet,* The Carnegie Mellon University, accessed 27 Nov. 2018, <https://www.cs.cmu.edu/~coke/history_long.txt>.

[3] J.Teicher 2018, *The little-known story of the first IoT device*, IBM, accessed 27 Nov. 2018, <https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/>.

[4] J.Manyika, M.Chui, et al. 2015, *Unlocking the potential of the Internet of Things,* McKinsey Global Institute, accessed 28 Nov. 2018, <https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world>.

[5] Miorandi, D., Sicari, et al. 2012, “Internet of Things: Vision, Applications and Research Challenges*”, Ad hoc networks*, Volume 10, Pages 1497-1516, accessed 28 Nov. 2018, from Elsevier.

[6] European Commission 2017, *Horizon 2020*, accessed on 28 Nov. 2018, <https://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020>.

[7] European Commission 2015, *ICT-30-2015 - Internet of Things and Platforms for Connected Smart Objects*, accessed on 28 Nov. 2018, <https://cordis.europa.eu/programme/rcn/664815_en.html>.

[8] Y. Dibrov 2018, *The Internet of Things Is Going to Change Everything About Cybersecurity*, Harvard Business Review, accessed on 28 Nov. 2018, <https://hbr.org/2017/12/the-internet-of-things-is-going-to-change-everything-about-cybersecurity>.

[9] European Commission 2017, *Building an IoT OPen innovation Ecosystem for connected smart objects*, accessed on 28 Nov. 2018, <https://cordis.europa.eu/project/rcn/200391_en.html>.

[10] bIoTope Project 2018, *bIoTope Project*, accessed on 28 Nov. 2018, <https://biotope-project.eu/> .

[11] M.L Corbin 2016, *Node-RED: The fundamental, easy to use, open-source programming tool for IoT,* IBM Developers, accessed on 28 Nov. 2018, <https://developer.ibm.com/dwblog/2016/node-red-programming-tool-open-source-iot/> .

[12] P. Pratim Ray 2017, “A Survey on Visual Programming Languages in Internet of Things”, *Scientific Programming,* Volume 2017, Article ID 1231430, accessed on 28 Nov. 2018 from Hindawi.

[13] H. Boyes, B. Hallaq, et al. 2018, ‘’The industrial internet of things (IIoT): An analysis framework’’ *Computers in Industry,* Volume 101, Pages 1-12*,* accessed on 28 Nov. 2018 from Elsevier.

[14] M. Rüßmann, P. Gerbert, et al. 2015, *Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries*, Pages 1-14, Boston Consulting Group, accessed on 28 Nov. 2018, <https://www.bcg.com/publications/2015/engineered_products_project_business_industry_4_future_productivity_growth_manufacturing_industries.aspx>.

[15] K.Schwab 2016, *The Fourth Industrial Revolution: what it means, how to respond*, World Economic Forum, accessed on 28 Nov. 2018, < https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond>.

[16] L. S. Dalenogare, G.Brittes 2018, "The expected contribution of Industry 4.0 technologies for industrial performance", *International Journal of Production Economics,* Volume 204, Pages 383-394, accessed on 29 Nov. 2018, from Elsevier.

[17] M. Burnett 1999, ''Visual Programming'', *Wiley Encyclopedia of Electrical and Electronics Engineering, J. Webster (ed.).*

[18] Matthew Revell 2017, *What Is Visual Programming?,* Outsystems, accessed on 29 Nov. 2018, <https://www.outsystems.com/blog/what-is-visual-programming.html>

[19] Visuino 2017, *Visuino*, accessed on 29 Nov. 2018, <http://www.visuino.com/>.

[20] Wia 2018, *Wia*, accessed on 29 Nov. 2018, <https://www.wia.io/>.

[21] Node-RED 2018, *Node-RED*, accessed on 29 Nov. 2018, <https://nodered.org/>.

[22] Node-RED 2018, *Node-RED about*, accessed on 29 Nov. 2018, <https://nodered.org/about//> .

[23] GitHub 2018, bIoTope's KnaaS (Knowledge as a Service Framework in the IoT Era) and UIaaS (User Interface as a Service), accessed on 28 Nov. 2018, <https://github.com/prokolyvakis/knaas-uiaas>.

[24] M. Kerrigan, A. Mocan, et al. 2007, ''The Web Service Modeling Toolkit - An Integrated Development Environment for Semantic Web Services'', *ESWC 2007: The Semantic Web: Research and Applications,* Pages 789-798, accessed on 29 Nov. 2018, from SpringerLink.

[25] Veracode 2018, *What is an Integrated Development Environment (IDE)?,* accesed on 29 Nov.2018, <https://www.veracode.com/security/integrated-development-environments>.

[26] Wikipedia 2018, *Interpreter (computing),* accessed on 29 Nov. 2018, <https://en.wikipedia.org/wiki/Interpreter_(computing)>.

[27] Techdifferences 2017, *Difference Between Compiler and Interpreter*, accessed on 28 Nov. 2018, <https://techdifferences.com/difference-between-compiler-and-interpreter.html>.

[28] Brithon 2018, *Brithon Introduction*, accessed on 29 Nov. 2018, <https://brython.info/static_doc/en/intro.html>.

[29] Repl.it 2018, *About Repl*, accessed on 29 Nov. 2018, <https://repl.it/site/about>.

[30] PyPy.js 2018, *Pypy.js Overview*, accessed on 29 Nov. 2018, <https://pypyjs.org/>.

[31] GitHub 2018, *PyPy compiled to JavaScript*, accessed on 29 Nov. 2018, <https://github.com/pypyjs/pypyjs>.

[32] Scott Graham 2018, *Skulpt Project*, accessed on 29 Nov. 2018, <http://h4ck3r.net/#Skulpt>.

[33] Transcrypt 2018, *Transcrypt, What and Why*, accessed on 29 Nov. 2018, <http://www.transcrypt.org/docs/html/what_why.html#what-is-transcrypt>.

[34] GitHub 2018, *Welcome to Skulpt*, accessed on 29 Nov. 2018, <https://github.com/skulpt/skulpt>.

[35] Interactive Python 2018, *How to think like a Computer Scientist*, accessed on 29 Nov. 2018, <http://interactivepython.org/courselib/static/thinkcspy/GeneralIntro/intro-TheWayoftheProgram.html>.

[36] Trinket 2018, *Trinket*, accessed on 29 Nov. 2018, <https://trinket.io/>.

[37] CodeSkulptor 2018, *CodeSkulptor*, accessed on 29 Nov. 2018, <http://www.codeskulptor.org/>.

[38] GitHub 2018, *Trinket: The easiest way to teach with interactive examples*, accessed on 29 Nov. 2018, <https://github.com/trinketapp>.

[39] J. Gubby, R. Buyya, et al. 2016, "Internet of Things (IoT): A vision, architectural elements, and future directions*", Future Generation Computer Systems,* Volume 29, Issue 7, Pages 1645-1660, accessed 28 Nov. 2018, from Elsevier.

[40] Oxford English Dictionary 2018, *Database*, accessed on 29 Nov. 2018 <http://www.oed.com/view/Entry/47411>.

[41] J. Cooper, A. James 2009, "Challenges for database management in the internet of things*", IETE Technical Review*, Volume 26, Issue 5, Pages 320-329, accessed 28 Nov. 2018, from ResearchGate.

[42] A. Bridgwater 2016, *The IoT needs a new kind of database,* Internet of Business, accessed 30 Nov. 2018, <https://internetofbusiness.com/iot-needs-new-kind-database/>.

[43] Y. Lin, S. Manoharan 2013, "A performance comparison of SQL and NoSQL databases*", 2013 IEEE Pacific Rim Conference*, accessed 28 Nov. 2018, from IEEE.

[44] MongoDB 2018, *What is MongoDB*, accessed on 30 Nov. 2018 <https://www.mongodb.com/what-is-mongodb>.

[45] MongoDB 2018, *MongoDB Internet of Things*, accessed on 30 Nov. 2018 <https://www.mongodb.com/use-cases/internet-of-things>.

[46] Len Bass, Ingo Weber, et al. 2015, "DevOps: A Software Architect's Perspective*", Pearson Education Inc.*

[47] M. Huttermann 2012, "DevOps for Developers*", Apress ed.*

[48] C. Boettiger 2015, "An introduction to Docker for reproducible research, with examples from the R environment"*, ACM SIGOPS Operating Systems Review,* Volume 49, Issue 1, accessed on 30 Nov. 2018, from ResearchGate

[49] Gartner 2016, *DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations*, media released, accessed on 28 Nov. 2018, <https://www.gartner.com/newsroom/id/2999017>.

[50] Docker 2018, *Docker Overview*, accessed on 30 Nov. 2018 <https://docs.docker.com/engine/docker-overview/>.

[51] Docker 2018, *What is a container?*, accessed on 30 Nov. 2018 <https://www.docker.com/resources/what-container>.

[52] RayGun 2018, *The 10 best DevOps tools for 2018*, accessed on 30 Nov. 2018 <https://raygun.com/blog/best-devops-tools/>.

[53] Dav Clark, A. Culich, et al. 2014, "BCE: Berkeley's Common Scientific Compute Environment for Research and Education*", 13th PYTHON IN SCIENCE CONF. (SCIPY 2014)*, accessed 30 Nov. 2018, from Research IT Berkeley.

[54] Docker 2018, *Docker Containerization Unlocks the Potential for Dev and Ops,* accessed on 30 Nov. 2018, <https://www.docker.com/why-docker>.

[55] Docker 2018, *Companies from Every Industry Succeed with Docker Enterprise*, accessed on 30 Nov. 2018, <https://www.docker.com/customers>.

[56] Git 2018, *Git fast version control,* accessed on 30 Nov. 2018, <https://git-scm.com/>.

[57] S. Chacon and B. Straub 2014, "Pro Git*", Apress Ed,* 2nd Edition.

[58] Git 2018, *About Git*, accessed on 30 Nov. 2018, <https://git-scm.com/about/>.

[59] D. Flanagan 2006, "JavaScript: The Definitive Guide", 4th Edition

[60] N.C. Zachas 2005, "Professional JavaScript for Web Developers*", Wiley Publishing Inc.*

[61] Web Technology Surveys 2018, *Usage of JavaScript for Websites*, accessed on 30 Nov. 2018 <https://w3techs.com/technologies/details/cp-javascript/all/all>.

[62] Python 2018, *General Python FAQ*, accessed on 30 Nov. 2018 <https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place>.

[63] A.C. Stross 2018, "Python: a programming language changes the world", *Python Brochure,* accessed on 30 Nov. 2018 from Python Support Website.

[64] F. Pedregosa, G. Varoquaux, et al. 2011, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning,* Volume 12, Pages 2825−2830, accessed 30 Nov. 2018, from JMLR.

[65] W3C 1999, "HTML 4.01 Specification", *World Wide Web Consortium,* accessed 30 Nov. 2018, <https://www.w3.org/TR/html401/>.

[66] Node.js 2018, *About Node.js*, accessed on 30 Nov. 2018 <https://nodejs.org/en/about/>.

[67] M. Cantelon, M. Harter, et al. 2011, "Node.js in action", *Manning Publications.*

[68] S. Tilkov, S. Vinoski 2010, "Node.js: Using JavaScript to Build High-Performance Network Programs*", IEEE Internet Computing*, Volume 14, Issue 6, accessed 30 Nov. 2018, from IEEE Library.

[69] D.G. Puranik, D. Feiock, et al. 2013, "Real-Time Monitoring using AJAX and WebSockets*", 20th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, accessed 30 Nov. 2018, from IEEE Library.

[70] J. J. Garrett 2005, "Ajax: A New Approach to Web Applications", *Adaptive path,* accessed on 30 Nov. 2018 <http://adaptivepath.org/ideas/ajax-new-approach-web-applications/>.

[71] W3Schools 2018, *Ajax Introduction,* accessed on 30 Nov. 2018 <https://www.w3schools.com/xml/ajax_intro.asp>.

[72] C. Larman 2004, "Agile and Iterative Development: A Manager's Guide", *Addison-Wesley Professional*

[73] Microsoft Azure 2018, *What is a virtual machine?*, accessed on 30 Nov. 2018 <https://azure.microsoft.com/en-gb/overview/what-is-a-virtual-machine/>.

[74] GitHub 2018, *Building Skulpt with Docker,* accessed on 30 Nov. 2018 <https://github.com/skulpt/skulpt/issues/800>.

[75] Docker 2018, *Docker Use Volumes,* accessed on 30 Nov. 2018 <https://docs.docker.com/storage/volumes/>.

[76] Wikipedia 2018, *Josephus Problem,* accessed on 30 Nov. 2018 <https://en.wikipedia.org/wiki/Josephus_problem>.

[77] WMO 2015, "Guidelines on the Definition and Monitoring of Extreme Weather and Climate Events", *World Meteorological Organization,* accessed on 30 Nov. 2018, <https://www.wmo.int/pages/>.

[78] J.R. Nairn, R. Fawcett 2015, "The Excess Heat Factor: A Metric for Heatwave Intensity and Its Use in Classifying Heatwave Severity", *International Journal of Environmental Research and Public Health,* accessed on 30 Nov. 2018, from PMC US National Library of Medicine.

[79] C. Mora, B. Dousset, et al. 2017, "Global risk of deadly heat", *Nature Climate Change,* Volume 7, Pages 501–506*,* accessed on 30 Nov. 2018, from Nature Research Journal.

[80] A.J. Hobday, L. Alexander, et al. 2016, "A hierarchical approach to defining marine heatwaves", *Progress in Oceanography,* Volume 141, Pages 227-238*,* accessed on 30 Nov. 2018, from Elsevier.

[81] R.O. Teskey, T. Wertin, et al. 2014, "Responses of tree species to heat waves and extreme heat events", *Plant Cell and Environment,* Volume 38, Issue 9*,* accessed on 30 Nov. 2018, from Research Gate.

[82] Greater Lyon Business 2018, *Lyon Smart City,* accessed on 01 Dec. 2018, <http://www.business.greaterlyon.com/smart-city-lyon-process-47.html>.

[83] J. Robert, S. Kubler, et al. 2016, "Open IoT Ecosystem for Enhanced Interoperability in Smart Cities—Example of Métropole De Lyon", *Sensors,* Volume 17, Issue 12*,* accessed on 01 Dec. 2018, from MPDI Journals.

[84] A. Tzavali, J.P. Paravantis et al. 2015, "Urban Heat Island Intensity: A Literature Review", *Fresenius Environmental Bulletin,* accessed on 01 Dec. 2018, from Research Gate.

[85] European Climate KNMI 2018, *Monthly Lyon Temperatures,* accessed on 16 Nov. 2018 <https://climexp.knmi.nl/gettempall.cgi?id=someone@somewhere&WMO=7480&STATION=LYON>.

[86] FIFA 2018, *FIFA World Cup Qatar,* accessed on 01 Dec. 2018 <(https://www.fifa.com/worldcup/qatar2022/>.

[87] S. Hall 2018, *Does hosting a World Cup make economic sense?*, WEF, accessed on 01 Dec. 2018 <https://www.weforum.org/agenda/2018/06/world-cup-football-smart-investment-russia-host/>.

[88] IMF 2017, *World Economic Outlook,* accessed on 30 Nov 2018 <https://www.imf.org/external>.

[89] Oxford Business Group 2017, *Major transport projects in preparation for 2022 FIFA World Cup in Qatar?,* accessed on 01 Dec. 2018 <https://oxfordbusinessgroup.com/overview/flurry-activity-major-projects-prepare-2022-fifa-world-cup-and-meet-long-term-development-goals>.

[90] Rahul Bali 2017, 2022 *World Cup: Cooling technology,* Goal, accessed on 01 Dec. 2018 <https://www.goal.com/en/news/2022-world-cup-cooling-technology-all-you-need-to-know/feh3ywch67c4100bnh8w83jjt>.

[91] L. Makdessi 2017, *Identifying new opportunities in Qatar's parking industry,* Parking network, accessed on 01 Dec. 2018 <http://www.parking-net.com/parking-news/qatars-parking-industry>.

[92] K. Främling, C. Cherifi 2018, "Open IoT Ecosystem for Sporting Event Management"*, IEEE Access* Volume 5, accessed on 01 Dec. 2018 from IEEE Journals>.

[93] Portal Qatar Government 2018, *World Cup Stadiums*, accessed on 01 Dec. 2018 <https://portal.www.gov.qa/wps/portal/topics/Tourism+Sports+and+Recreation/worldcupstadiums>.

[94] Rosetta Code 2018, *Haversine Formula,* accessed on 01 Dec. 2018, <https://rosettacode.org/wiki/Haversine_formula>.

BUDGET

IMPLEMENTING AN INTEGRATED DEVELOPMENT ENVIRONMENT IN THE INTERNET OF THINGS GENERATION OF VISUAL PROGRAMMING TOOLS

AUTHOR: JOSÉ QUILES ALEMAÑ

SUPERVISED BY:
DIMITRIS KYRITSIS (EPFL)
PRODROMOS KOLYVAKIS (EPFL)
LUIS JOSÉ SAIZ ADALID (UPV)

Laboratory:
Information and Communication Technology for Sustainable Manufactory (ICT4SM)

CHAPTER 1. INTRODUCTION

The budget document deals with the costs directly associated with the execution of this project. It is clear, that as this thesis has been developed as a student in one office provided by the university, the cost has not passed directly to the author of the project. However, it is needed to determine which would have been the total cost of the project if it had been developed without any kind of agreement, expressing how much would have cost to an external company to carry it out.

The importance of this document is therefore evident, since it provides an idea to any external person or company about how much can it cost to develop one project of similar characteristics; moreover, elaborating a budget is a fundamental task to verify the viability of any project, allowing the author to understand the reality of this particular one, as it has to be presented to the senior manager of the company that needs to understand if the project may be undertaken.

This section is structured in four chapters; the first one attempts to express the necessity and importance of elaborating the budget of the project, the second indicates all the measurement units actually employed through the different phases, then the unitary cost of each measurement unit is defined to finish calculating the total cost of the project.

CHAPTER 2. MEASUREMENT UNITS

In this chapter, it is presented the total number of measurement units employed in the different phases of the project that actually incur in a cost. In concrete, this project required seventy days of complete dedication that amount to a number of 560 working hours which obviously incurs in labor cost, besides, some of the hours where dedicated to meetings with experts that would also be paid. Moreover, during these days it was required the use of electricity, in form of light and computer powering. It is also considered that the computer system gets depreciated, and that the software and material expenses are directly associated to this particular project.

Definition of the project

This phase considers the different actions required to define properly the project, including the identification of an opportunity for improvement, the definition of objectives and requirements or establishing the methodology and structure to follow during the development.

Definition of the project		
Description	Measurement	Unit
Electricity	10	days
Computer System Wear and Tear	10	days
Labor	80	hours

Building a Python Development Environment based on Skulpt

The second stage involves all the activities needed to create the new Development Environment as building Skulpt, developing JavaScript functions, creating Node.js files, connecting the client with the server side, implementing Python modules, configuring graphs or developing the Google iframe.

Building a Python Development Environment based on Skulpt		
Description	Measurement	Unit
Electricity	21	days
Computer System Wear and Tear	21	days
Labor	168	hours
Expert's consultation	10	hours

Implementation with Node-RED

Involves all the activities related with establishing the connection between Node-RED and the new Integrated Development Environment.

Implementation with Node-RED		
Description	Measurement	Unit
Electricity	5	days
Computer System Wear and Tear	5	days
Labor	40	hours
Expert's consultation	4	hours

Simulation in Application Case Scenarios

The set of actions directly related with testing in representative cases, consist basically in researching real application possibilities, selecting and justifying two cases that represent a good test for the IDE, defining problems and possible solutions for each case, developing an IoT solution for each case, expressing the advantages and limitations of the new environment.

Simulation in Application Case Scenarios		
Description	Measurement	Unit
Electricity	20	days
Computer System Wear and Tear	20	days
Labor	160	hours

Documenting

Tasks related with developing the technical documentation such as typing, editing text…

Documenting		
Description	Measurement	Unit
Electricity	14	days
Computer System Wear and Tear	14	days
Labor	112	hours

Other Concepts

Apart from the costs that can be directly allocated to one of the phases, there are also other expenses associated to the project that need to be accounted such as the purchase of software licenses or desktop materials.

Other Concepts		
Description	Measurement	Unit
Desktop material (paper, pen, reading lamp…)	1	units
Software Licenses (Ms. Office)	4	months
Book printing and binding	1	units

CHAPTER 3. DETAILED BUDGET

The third chapter illustrates the unitary costs of each of the units presented on the previous one in order to calculate the cost associated to each of the phases defined. It has been considered for simplicity and because of the better knowledge by the author that the project is developed in Spain, not in Switzerland that would result in a higher cost.

The depreciation of the computer system is calculated linearly considering an initial cost of 1500€ with a depreciation lifespan of five years and a salvage value at the end of the lifespan of 300€. In this way, the monthly depreciation can be calculated as:

$$\text{Monthly depreciation} = \frac{\text{Cost} - \text{Savage value}}{\text{depreciation lifespan} \times \text{months/year}} = \frac{1500 - 300}{5 \times 12} = 20€$$

Considering that in a month there are on average 22 working days, it is determined that the computer system wear and tear is:

$$\text{Computer system Wear and Tear} = \frac{\text{Monthly depreciation}}{\text{Working days/month}} = \frac{20}{22} = \mathbf{0,909\ €/day}$$

Regarding the labor cost, it is estimated that the different task can be performed by an Industrial Engineer without any experience, considering a base salary of 18.000€ (230 days/year), Social Security (28,3%), a food diet of 5 euros per day with and a daily dedication of 8 hours:

$$Labor\ Cost = \frac{Base\ Salary \times (1 + SS)}{Working\ Days/year \times hours/working\ day} + \frac{Diet/day}{hours/working\ day} =$$

$$= \frac{18.000 \times (1 + 0{,}283)}{230 * 8} + \frac{5}{8} = \boldsymbol{13.176\ €/hour}$$

In relation with the meetings with the experts on the topic, it is supposed an agreement by ***40€/hour***. The license used during the development include the subscription to the Ms. Office package with a cost of ***10 €/month***. The office material is rounded to ***38 €*** since it involves a reading lamp and writing tools such as paper or pens. It is also needed to print the final documentation with more than one hundred pages with color and to bind it as a book in order to be presented, ***12€.*** Finally, it is considered an average cost of office electricity of ***0,5 €/day***.

According to these unitary costs and the measurement of each of the units determined on the previous chapter, it is possible to calculate the cost associated to each one of the different phases of the project.

Definition of the project

Definition of the project				
Description	Measurement	Unit	Unitary Cost	Cost Activity/Phase
Electricity	10	days	0.5	**5**
Computer System Wear and Tear	10	days	0.909	**9.09**
Labor Cost	80	hours	13.176	**1054.08**
			Phase Cost	**1068.17**

Building a Python Development Environment based on Skulpt

Building a Python Development Environment based on Skulpt				
Description	Measurement	Unit	Unitary Cost	Cost Activity/Phase
Electricity	21	days	0.5	**10.5**
Computer System Wear and Tear	21	days	0.909	**19.089**
Labor Cost	168	hours	13.176	**2213.568**
Expert's consultation	10	hours	40	**400**
			Phase Cost	**2643.157**

Implementation with Node-RED

Implementation with Node-RED				
Description	Measurement	Unit	Unitary Cost	Cost Activity/Phase
Electricity	5	days	0.5	**2.5**
Computer System Wear and Tear	5	days	0.909	**4.545**
Labor Cost	40	hours	13.176	**527.04**
Expert's consultation	10	hours	40	**80**
			Phase Cost	**614.085**

Simulation in Application Case Scenarios

Simulation in Application Case Scenarios				
Description	Measurement	Unit	Unitary Cost	Cost Activity/Phase
Electricity	20	days	0.5	**10**
Computer System Wear and Tear	20	days	0.909	**18.18**
Labor Cost	160	hours	13.176	**2108.16**
			Phase Cost	**2136.34**

Documenting

Documenting				
Description	Measurement	Unit	Unitary Cost	Cost Activity/Phase
Electricity	14	days	0.5	**7**
Computer System Wear and Tear	14	days	0.909	**12.726**
Labor Cost	112	hours	13.176	**1475.712**
			Phase Cost	**1495.438**

Other Concepts

Other Concepts				
Description	Measurement	Unit	Unitary Cost	Cost
Desktop material (paper, reading lamp…)	1	units	38	**38**
Printing and Binding	1	units	12	**12**
Ms Office	4	months	10	**40**
			Phase Cost	**90**

CHAPTER 4. TOTAL COST

In base to the results obtained on the previous chapter and considering an industrial benefit of 6%, a ratio of 10% for unforeseen expenses, and a value-added tax of 21%, it is possible to determine the total cost associated to the project.

PHASE	COST	
Definition of the project	1068.17	€
Building a Python Development Environment based on Skulpt	2643.157	€
Implementation with Node-RED	614.085	€
Simulation in Application Case Scenarios	2136.34	€
Documenting	1495.438	€
Other Concepts	90	€
Cost Material Execution	8047.19	€
Industrial Benefit (6%)	482.8314	€
Other Expenses (10%)	804.719	€
Cost without Value-Added Tax (VAT)	9334.7404	€
VAT (21%)	1960.295484	€
Project Total Cost	**11295.03588**	**€**

Hence, the budget for the project amounts to **ELEVEN THOUSAND, TWO HUNDRED NINETY-FIVE EUROS AND FOUR CENTS (11.295,036€).**

The resultant cost does not represent a huge sum if it is considered that just the bIoTope project received funding by the European Union of almost eight million euros to facilitate companies the creation of new Platforms for Connected Smart Objects [9]; the reasons that explain the relatively low final cost is that it was not needed to use much more than a good computer system and the Microsoft Office license, since the rest of the software employed was completely open sourced. It may be denoted that the distribution of cost by phases is almost directly proportionate to the number of hours employed, just as it is shown below:

Phase	Hours/Phase	%hours	Cost/Phase	%Cost
Definition of the project	80	**14.285**	1068.17	**13.423**
Building a Python Development Environment based on Skulpt	168	**30**	2643.157	**33.217**
Implementation with Node-RED	40	**7.143**	614.085	**7.717**
Simulation in Application Case Scenarios	160	**28.571**	2136.34	**26.848**
Documenting	112	**20**	1495.438	**18.793**

The little differences between the percentages may be related with the consulting hours that are employed on phases 2 and 3, incrementing its relative cost. The definitive cost distribution by phases, including other concepts, is represented on Figure XXV.

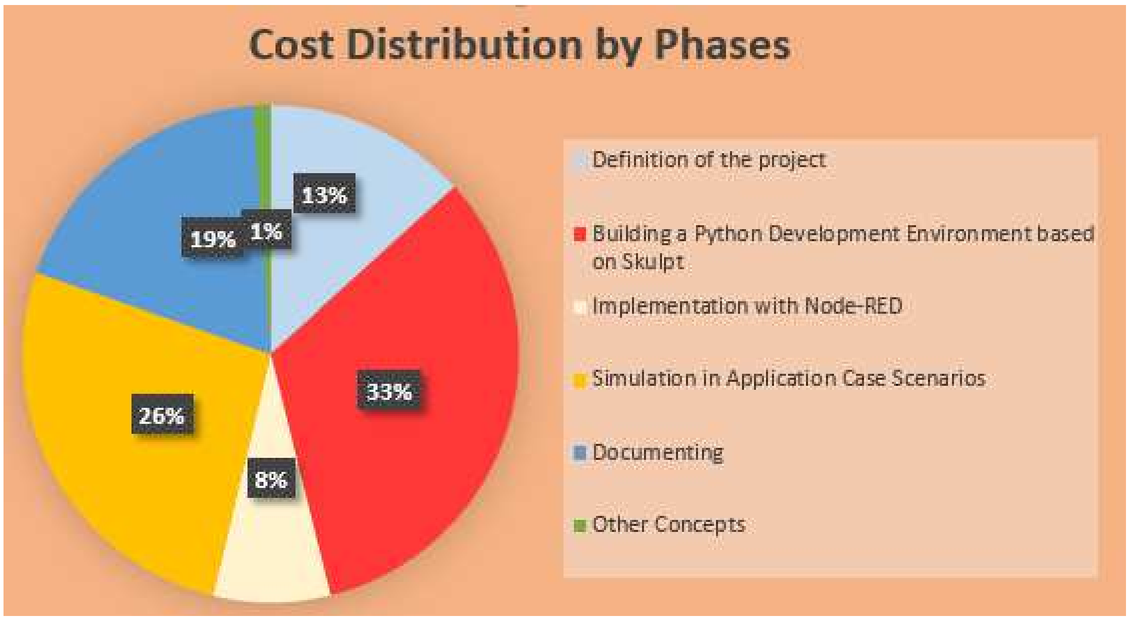


Figure XXV. Cost Distribution by Phases

Another interesting point to highlight is how the cost is distributed by activity, since it informs clearly about the relevance of the labor cost in this project (92% of the cost), relegating to a second level the consulting hours (6%) and far away from them, the rest of the costs that are rounding the one percent of the total cost of material execution.

Activity	Cost (€)	%Cost
Electricity	35	**0.434934431**
Computer System Wear and Tear	63.63	**0.790710795**
Labor Cost	7378.56	**91.69113691**
Expert's consultation	480	**5.964815047**
Desktop material (paper, pen, reading lamp…)	50	**0.621334901**
Software Licenses	40	**0.497067921**

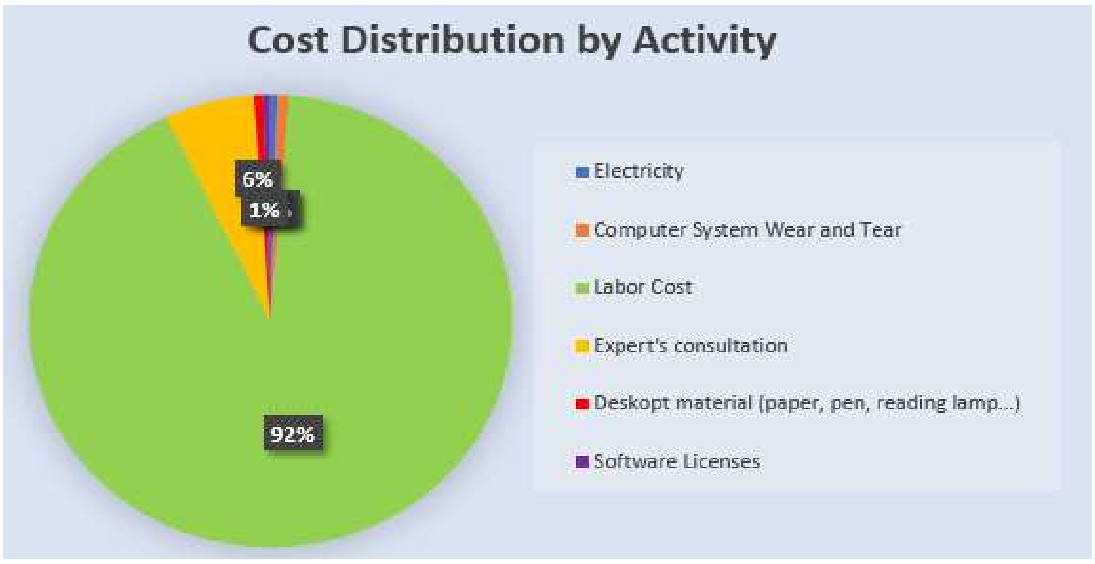


Figure XXVI. Cost Distribution by Activity

According to the results obtained some conclusions could be drawn in order to reduce the total cost of the project. The first idea that would come to the reader's mind, it is that the labor cost should be attacked, and evidently it makes sense, since it represents most of the project cost. Now, the question that must be asked is how it could be reduced.

One option is reducing the salary of the Industrial Engineer, which would incur in less cost per hour, and considering that the hours stay inalterable would reduce the total cost. However, this solution has some flaws (this project is part of a Master Thesis in which the student is completely motivated to achieve the objectives, finishing his master's degree and learning new IT concepts) a reduction of that salary could result in the discontent of the engineer, reducing his productivity and increasing the hours committed to the project, incrementing the total cost. Moreover, reducing the salary could not only have a negative impact on the cost of the project, but also on the quality of the developed solution.

For these reasons, the solution proposed to reduce the total cost of the project would be reducing the total amount of hours. This could be achieved by different methods, like hiring an engineer with more experience, of course it would increment the cost per hour, but it could reduce the number of working hours and consulting hours with experts of the field.

Another possible solution has to do with the efficiency of the engineer. Of course, in this project a clear methodology in order to boost the efficiency is followed, however, some other aspects could be taken into account by a company as providing previous projects or user handbooks to facilitate the launching of the project.

A last final idea would be related with the distribution of the tasks. Some tasks performed during the development of the project does not require any technical background or specific formation; for example, it could be positive to delegate documenting tasks to someone specialized on that field, reducing not only the time associated to that task, but also reducing the engineers hours that are more expensive and even increasing the satisfaction of the engineer that would be more focused on the technical aspects of the project.

ANNEX

IMPLEMENTING AN INTEGRATED DEVELOPMENT ENVIRONMENT IN THE INTERNET OF THINGS GENERATION OF VISUAL PROGRAMMING TOOLS

AUTHOR: JOSÉ QUILES ALEMAÑ

SUPERVISED BY:
DIMITRIS KYRITSIS (EPFL)
PRODROMOS KOLYVAKIS (EPFL)
LUIS JOSÉ SAIZ ADALID (UPV)

Laboratory:
Information and Communication Technology for Sustainable Manufactory (ICT4SM)

CHAPTER 1. INTRODUCTION

The Annex document includes basically the pieces of code developed by the author for this particular project. The code is presented on this section to provide the reader with all the information relative to the Master Thesis, being particularly useful to developers considering the possibility to create a similar solution or just to anybody who has curiosity about how something is developed.

It is important to underline that these pieces of code are not included on the report section since they do not provide any value to the complete understanding of the project, even complicating the reading task.

It was thought to present the JavaScript and Python functions as pseudocode (in order to provide any reader with the possibility to understand the code, even if they do not know the specific programming language), however, it was finally decided to provide the reader with the original code with additional comments (using #) to allow the understanding but also avoiding the loss of information and presenting the real code that leads to the solution.

To conclude, the annex section is divided in an introduction chapter, the index.html, a chapter that includes the JavaScript functions developed, other with the node.js files, another annex chapter including the different dockerfiles and docker-composes elaborated, chapters that include all the relative code with the application cases as IDE functions and nodes of Node-RED, an annex with the markdown that provides more documentation for the users of the IDE and finally a last annex with the Python file created to automate commands.

ANNEX I (INDEX.HTML)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Your Python IDE</title>
  <link rel="stylesheet" type="text/css" media="all" href="static/codemirror.css">
  <link rel="stylesheet" type="text/css" media="all" href="static/solarized.css">
  <link rel="stylesheet" type="text/css" media="all" href="static/main.css">
  <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.0/jquery.min.js"></script>
  <script src="static/codemirrorepl.js" type="text/javascript"></script>
  <script src="static/repl.js" type="text/javascript"></script>
  <script src="static/python.js" type="text/javascript"></script>
  <script src="http://www.skulpt.org/support/jsbeautify/beautify.js" type="text/javascript"
></script>
  <script src="static/skulpt.min.js" type="text/javascript"></script>
  <script src="static/debugger/debugger.js" type="text/javascript"></script>
  <script src="static/skulpt-stdlib.js" type="text/javascript"></script>
  <script src="static/env/editor.js" type="text/javascript"></script>
  <script src="static/debugger/debugger_cm.js" type="text/javascript"></script>
  <script type="text/javascript">

# JavaScript Function to represent graphs on a pop-up window.
  Sk.domOutput = function(html) {
            var newWindow = window.open("", '_blank',
           "height=350,width=450,status=yes,toolbar=no,menubar=no,location=no");
             return $(newWindow.document.body).append(html).children( ).last( );
             };

# Implementing New Modules, defining path and dependencies using Sk.externalLibraries method.
  Sk.externalLibraries = {
              pygal : {
                     path : 'pygal/__init__.js',
                     dependencies : [
                           'https://cdnjs.cloudflare.com/ajax/libs/highcharts/6.0.2/highcharts.js',
                           'https://cdnjs.cloudflare.com/ajax/libs/highcharts/6.0.2/js/highcharts-more.js'
                    ]},
             numpy : {
                     path : 'numpy/__init__.js',
                    },
             random : {
                     path : 'numpy/random.js',
                    },
             matplotlib : {
                     path : 'matplotlib/__init__.js',
                     dependencies : [
                           'matplotlib/d3.min.js',
              ]}
          };
 </script>
</head>
<body onLoad="scrollWin( )";> # Calling Function that scrolls up the IDE on load.
<a href="https://github.com/skulpt/skulpt"><img style="position: absolute;
  top: 0; left: 0; border: 0;"
  src="https://s3.amazonaws.com/github/ribbons/forkme_left_red_aa0000.png"
  alt="Fork me on GitHub"></a>
<div class="page">
<div class="body">
<div class="main">
<img src="static/pyt1.png" width="1000" height="300" alt="YourPythonIDE" id="pyt"><img
src="static/biotope.png" alt="bio" id="tope">

<h2>Python. Client side.</h2>
<p>This website is an <em>entirely in-browser</em> implementation of Python.</p>
<p>No preprocessing, plugins, or server-side support required, just write
  Python and reload.</p>
<p>You can use the Interactive screen to check line by line if your code contains any mistake. Using
the Demo functionality you can write your code and get the result in the Output screen. Apart from
that you can also load your own files or saving the new code in a new file.
</p>
<a href="/project.html" target="_blank">Documentation</a> # Link to documentation file
<div id="quickdocs" style="display: none">
  <ul>
      <li>cut/copy/paste/undo/redo with the usual shortcut keys</li>
      <li>Tab does decent indenting.
          Thanks to <a href="http://codemirror.net/"
                   target="_blank">CodeMirror</a> for the text editor.
      </li>
      <li>Ctrl-Enter to run, Shift-Enter to run selected</li>
  </ul>
</div>
<div class="row">
 div class="column">

  <h2>Interactive:</h2>
  <p>In this window you can write your code and see if you make any mistake line by line.
  </p>
  <button id="skulpt_test"onclick="alert ('Now, you do not have any excuse to write your perfect
code')">Enjoy</button>
  <textarea id="interactive" cols="85" rows="1"></textarea>
  <p></p>
</div>
<div class="column"> # Defining class as colums to structure the text.

<h2>Editor</h2>
<p> Press the Run button to get an output. Examples:
  <a href="index.html#" id="codeexample1">1</a>
  <a href="index.html#" id="codeexample2">2</a>
  <a href="index.html#" id="codeexample3">3</a>
  <a href="index.html#" id="codeexample4">4</a>
  <a href="index.html#" id="codeexample5">5</a>
  <a href="index.html#" id="codeexample6">6</a>
  <a href="index.html#" id="codeexample7">7</a>
  <a href="index.html#" id="codeexample8">8</a>.
 </p>
 # Buttons an Input Areas employed to call the different JavaScript Functions.
 <button id="skulpt_run">Run</button>
 <button id="skulpt_save">Save Text to File</button>
 <input id="inputFileNameToSaveAs"></input>
 <button id="skulpt_load">Load</button>
 <input id="inputnameload"></input>

 <textarea id="code" cols="85" rows="25">
   Here, you can write and run your code.
   You can also load a file that you have already written by typing the
   name of the file and pressing the load button.
   After that you can save your work to implement it in other programs.
  </textarea>
 </div>
</div>

<h2>Output: (<a href="index.html#" id="clearoutput">clear</a>)</h2>
<pre id="edoutput"></pre>
<div class="row">

 <div class="column">
  <h2>Google Search</h2>
  <p>You can always search for more information. <br>
       Use your browser icons to navigate back and forward.
  </p>
# Button to reload the Google iframe.
  <button id="button" onclick="test.location.href='https://www.google.com/search?igu=1'">Initial
Page</button>
  <div class="resp-container">
# Definition of the Google iframe
  <iframe class="iframe" name="test" id="test" width="650" height="300"
src="https://www.google.com/search?igu=1" style="border:none;"></iframe>
  </div>
 </div>

 <div class="column">
  <h2>Debugger</h2>
  <p> Here you have your debugger tool, but you can always hide it. </p>
  <br>
  <button id="skulpt_js_output" onclick="var cm = $('.CodeMirror')
  [2].CodeMirror;$(cm.getWrapperElement( )).toggle( )">Hide/Unhide JS</button>
   <div id="mycanvas" height="450" width="400"  style="border-style: solid;"></div>
   <textarea id="debugger_cm", cols="85" rows="1"></textarea>

  </div>
 </div>
 <script> # Definition of the function that scrolls up the IDE.
   function scrollWin( ) {
                window.scrollTo(0, 0);
                 };
 </script>
</body>
</html>
```

ANNEX II (JAVASCRIPT FUNCTIONS)

SAVE TEXT TO FILE

```
"save": function saveTextAsFile(editor)
{
# Receiving the values of the editor and input area.
  var textToSave = editor.getValue( );
  var textToSaveAsBlob = new Blob([textToSave], {type:"text/plain"});
  var textToSaveAsURL = window.URL.createObjectURL(textToSaveAsBlob);
  var fileNameToSaveAs = document.getElementById("inputFileNameToSaveAs").value;

          # headers to allow Ajax Connection.
            var headers = {
                              "Content-Type": "application/json; charset=utf-8",
                              "Accept" : "application/json"
                                };
         # Assigning to a new variable the contents received.
            var data = { };
            data.body = { };
            data.body.fileName =fileNameToSaveAs
            data.body.content  =textToSave

          # Sending to the server side the contents of the data variable making use of Ajax.
             jQuery.ajax({
                              type: 'POST',
                              data: data,
                              crossDomain:true,
                              url: 'http://localhost:5080/endpoint',
                             success: function(data) {
                                            console.log('success');
                                            console.log(JSON.stringify(data));
                                             }
                               });

# External Download with the title expressed on the input area and the contents of the editor.
 var downloadLink = document.createElement("a");
  downloadLink.download = fileNameToSaveAs;
  downloadLink.innerHTML = "Download File";
  downloadLink.href = textToSaveAsURL;
  downloadLink.style.display = "none";
  document.body.appendChild(downloadLink);
  downloadLink.click( );
},
```

LOAD

```
 "load": function Load(editor)
{
# Receiving the title expressed by the user on the input area
  var fileNameToLoad = document.getElementById("inputnameload").value;

            # Headers to provide Ajax Connection.
            var header = {
                          "Content-Type": "application/json; charset=utf-8",
                          "Accept" : "application/json"
                         };

          # Assigning to a new variable the title received
            var datas = { };
            datas.body = { };
            datas.body.fileName =fileNameToLoad

         # Sending to the server side the contents of the data variable making use of Ajax.
            jQuery.ajax({
                         type: 'POST',
                         data: datas,
                         crossDomain:true,
                         url: 'http://localhost:8090/',
                         success: function(result) {
                                   console.log('Data'+result);
                                # Assigning to a new variable the respone received from the server side
                                   var contents= result;
                                # Adding to the editor the contents received
                                   editor.setValue(contents);
                                  }

                        });
 },
```

ANNEX III (NODE.JS FILES)

STORAGE.JS

```
# Importing libraries needed
var fs = require('fs');
var _ = require('underscore');
var express = require("express");
var myParser = require("body-parser");
var app = express( );
var pp= 5;
console.log(pp);
# Commands to allow Ajax Connection
app.use(myParser.urlencoded({extended : true}));
    app.use(function(req, res, next) {
                res.setHeader('Access-Control-Allow-Origin', '*');
                res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
                res.setHeader('Access-Control-Allow-Headers', "Origin, X-Requested-With, Content-Type,
Accept");
                res.setHeader('Access-Control-Allow-Credentials', true);
                 next( );
              });

 # Receiving contents from the client side on request variable.
    app.post('/endpoint', function(request, response) {
                 var location= './storage/'   # Defining Location in which the file will be saved.
                 var text=JSON.stringify(request.body) # Changing to JSON format the contents recieved

             # Method to get the exact title defined by the user
                var start_pos = text.indexOf('m') + 5 ;
                var end_pos = text.indexOf('"',start_pos) ;
                var title = text.substring(start_pos,end_pos)
             # Method to get the exact content of the editor defined by the user
                var start_pos2 = text.indexOf(':',end_pos) + 2;
                var end_pos2 = text.lastIndexOf('"');
                var contenido = text.substring(start_pos2,end_pos2);

                var tmpo = contenido.split('\\n');  #Dividing the contents of the editor.

 #Creating a loop that puts toguether the parts of the code into different lines and removing the
counterbars, just as it appears on the editor.
                var x = tmpo.length
                var final="";
                var almost="";
                for (i = 0; i < x; i++) {
                              almost = final +tmpo[i] + "\r\n";
                              final = almost.replace('\\',"");
                               }
#Writing the solution into a file with the title expressed by the user on the location defined.
                fs.writeFile( location + title + ".py", final, function(err) {
```

```
                                if(err) {
                                          return console.log(err);
                                           }
                                 });
                 });
app.listen(5080); #Listening to the port to receive the contents of the client side
```

LOAD.JS

```
# Importing libraries needed
var fs = require('fs');
var _ = require('underscore');
var express = require("express");
var myParser = require("body-parser");
var app = express( );
var pp= 3;
console.log(pp);

# Commands to allow Ajax Connection
app.use(myParser.urlencoded({extended : true}));
app.use(function(req, res, next) {
               res.setHeader('Access-Control-Allow-Origin', '*');
               res.setHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE');
               res.setHeader('Access-Control-Allow-Headers', "Origin, X-Requested-With, Content-Type,
Accept");
               res.setHeader('Access-Control-Allow-Credentials', true);
               next( );
               });
# Receiving contents from the client side on request variable. Sending back the result using the
response variable
app.post('/', function(request, response) {

               var location= './storage/' # Defining Location in which the file will be searched
               var text=JSON.stringify(request.body) # Changing to JSON format the contents received

             # Method to get the exact title defined by the user
               var start_pos = text.indexOf('m') + 5 ;
               var end_pos = text.indexOf('"',start_pos) ;
               var title = text.substring(start_pos,end_pos);

             # Reading file with the title expressed, assigning them to a new variable and sending the
response back to the client side.
               var contents
               fs.readFile(location + title + ".py", 'utf8', function(err, contents) {
                                 response.send(contents);
                                  });


              });



  app.listen(8090);
```

ANNEX IV (DOCKER)

BUILDING NEW ENVIRONMENT

Dockerfile IDE

```
FROM nginx:alpine
COPY . /usr/share/nginx/html # Copying all the contents of the dockerfile directory to the
one expressed on the docker container.
EXPOSE 80 # Exposing Port.
```

Dockerfile Node.js

```
FROM node:10-alpine

RUN mkdir -p /src/app/storage # Creating new folder on the docker container.
WORKDIR /src/app # Specifying directory as the working directory.

COPY package*.json ./src/app # Copying the file to the directory on docker,

# Installing Dependencies
RUN npm install
RUN npm install underscore
RUN npm install express
RUN npm install body-parser
COPY . /src/app
# Exposing Ports
EXPOSE 8090
EXPOSE 5080

CMD node load.js & node storage.js # Executing both Node.js files.
```

Dockerfile Node-RED

```
FROM node:6
# Installing all the dependencies that may be used on Node-RED
RUN mkdir -p /usr/src/node-red
RUN mkdir /data

WORKDIR /usr/src/node-red
RUN useradd --home-dir /usr/src/node-red --no-create-home node-red \
&& chown -R node-red:node-red /data \
&& chown -R node-red:node-red /usr/src/node-red
USER node-red
COPY package.json /usr/src/node-red/
RUN npm install
RUN npm install node-red-contrib-python3-function # New Python modules
RUN npm install xmldom
RUN npm install skubler/Node-Red-ODF
RUN npm install RohanAsmat/Node-Red-OMI-V9
RUN npm install prokolyvakis/node-red-contrib-sparql
RUN npm install node-red-dashboard
RUN npm install node-red-contrib-web-worldmap
RUN npm install node-red-node-openweathermap
RUN npm install node-red-contrib-mongodb2

EXPOSE 1880 # Exposing Node-RED port

ENV FLOWS=flows.json

USER root
RUN apt-get update
RUN apt-get -y install python-pip # Getting and installing PIP to facilitate instalations
RUN pip install sparqlwrapper
RUN pip install rdflib

USER root
RUN apt-get -y install build-essential python-dev
RUN apt-get -y install python-numpy # Numpy


USER root
RUN apt-get -y install python-scipy python-matplotlib python-pandas python-
sympy python-nose # Scipy, pandas, matplotlib…
RUN apt-get install libblas-dev  liblapack-dev
RUN apt-get -y install gfortran
RUN pip install -U scikit-learn # Scikit-Learn
RUN apt-get -y install python-yaml
RUN python -m pip install pymongo
RUN pip install requests

USER root
RUN pip install statsmodels

USER root
RUN apt-get install nano
RUN npm install -g node-red-admin

USER node-red
CMD ["npm", "start", "--", "--userDir", "/data"]
```

Docker-compose.yml

```
version: '3'
   services:
   # Definition of the different containers, in order to be run as a unique service
      web:
        image: firstimage
        ports:
           - "4000:80" # Localhost port:Docker port

      nodejs:
        image: nodeimage
        ports:
           - "5080:5080"
           - "8090:8090"
        volumes:
          - "./modules:/src/app/storage"  # Connection between container and localhost

      o-mi-node:
        image: o-mi-node
        ports:
           - "8080:8080"

      mongodb:
        ports:
           - "27017:27017"
        image: mongo
        volumes:
          - './data/db:/data/db'

     node-red:
       image: node-red
       ports:
          - "1880:1880"
       volumes:# Using the same localhost directory, to provide connection between containers
         - "./modules:/usr/src/node-red/modules"
```

BUILDING SKULPT

Dockerfile

```
FROM node:9-alpine

LABEL repository.hub="alexmasterov/alpine-libv8:7.2" \
      repository.url="https://github.com/AlexMasterov/dockerfiles" \
      maintainer="Alex Masterov <alex.masterow@gmail.com>"

ARG V8_VERSION=7.2.138 ARG V8_DIR=/usr/local/v8
ARG BUILD_COMMIT=5a371bcc0efe2cc84f384f14bdf5eaf5fe3e271a
ARG BUILDTOOLS_COMMIT=13a00f110ef910a25763346d6538b60f12845656
ARG ICU_COMMIT=b029971f1fc6b20d06887c47c7afebd5881f31ff
ARG GTEST_COMMIT=2e68926a9d4929e9289373cd49e40ddcb9a628f7
ARG TRACE_EVENT_COMMIT=211b3ed9d0481b4caddbee1322321b86a483ca1f
ARG CLANG_COMMIT=3041f30dd6b3fa4fb8ca7db6439bed372f4accc0
ARG JINJA2_COMMIT=b41863e42637544c2941b574c7877d3e1f663e25
ARG MARKUPSAFE_COMMIT=8f45f5cfa0009d2a70589bcda0349b8cb2b72783
ARG CATAPULT_COMMIT=ed6fe0f638403e1afd377e38975e4fd430f53432
ARG GN_SOURCE=https://www.dropbox.com/s/3ublwqh4h9dit9t/alpine-gn-
80e00be.tar.gz
ARG V8_SOURCE=https://chromium.googlesource.com/v8/v8/+archive/${V8-
VERSION}.tar.gz
ENV V8_VERSION=${V8_VERSION} \ V8_DIR=${V8_DIR}
# Installation of all kind of dependencies needed.
RUN set -x \   && apk add --update --virtual .v8-build-dependencies \
at-spi2-core-dev \curl \g++ \gcc \glib-dev \icu-dev \linux-headers \make
\ninja \python \tar \xz \   && : "---------- V8 ----------" \   && mkdir -p
/tmp/v8 \   && curl -fSL --connect-timeout 30 ${V8_SOURCE} | tar xmz -C
/tmp/v8 \  && : "---------- Dependencies ----------" \   && DEPS=" \
chromium/buildtools.git@${BUILDTOOLS_COMMIT}:buildtools; \
chromium/src/build.git@${BUILD_COMMIT}:build; \
chromium/src/base/trace_event/common.git@${TRACE_EVENT_COMMIT}:base/trace_eve
nt/common; \chromium/src/tools/clang.git@${CLANG_COMMIT}:tools/clang; \
chromium/src/third_party/jinja2.git@${JINJA2_COMMIT}:third_party/jinja2; \
chromium/src/third_party/markupsafe.git@${MARKUPSAFE_COMMIT}:third_party/mark
upsafe; \chromium/deps/icu.git@${ICU_COMMIT}:third_party/icu; \
external/github.com/google/googletest.git@${GTEST_COMMIT}:third_party/googlet
est/src; \catapult.git@${CATAPULT_COMMIT}:third_party/catapult \   " \   &&
while [ "${DEPS}" ]; do \dep="${DEPS%%;*}" \link="${dep%%:*}" \
url="${link%%@*}" url="${url#"${url%%[![:space:]]*}"}" \hash="${link#*@}" \
dir="${dep#*:}"; \[ -n "${dep}" ] \
  &&
dep_url="https://chromium.googlesource.com/${url}/+archive/${hash}.tar.gz" \
  && dep_dir="/tmp/v8/${dir}" \&& mkdir -p ${dep_dir} \&& curl -fSL --
connect-timeout 30 ${dep_url} | tar xmz -C ${dep_dir} \& [ "${DEPS}" =
"${dep}" ] && DEPS='' || DEPS="${DEPS#*;}"; \
done; \ wait \   && : "---------- Downloads the current stable Linux sysroot
----------" \   && /tmp/v8/build/linux/sysroot_scripts/install-sysroot.py --
arch=amd64 \  && : "---------- Proper GN ----------" \   && apk add --virtual
.gn-runtime-dependencies \libevent \libexecinfo \
libstdc++ \   && curl -fSL --connect-timeout 30 ${GN_SOURCE} | tar xmz -C
/tmp/v8/buildtools/linux64/ \   && : "---------- Build instructions ---------
-" \   && cd /tmp/v8 \   && ./tools/dev/v8gen.py \
x64.release \-- \binutils_path=\"/usr/bin\" \target_os=\"linux\"
\target_cpu=\"x64\" \v8_target_cpu=\"x64\"
\v8_use_external_startup_data=false \v8_enable_future=true \
  is_official_build=true \is_component_build=true \is_cfi=false
\is_clang=false \ use_custom_libcxx=false \use_sysroot=false \use_gold=false
\use_allocator_shim=false  treat_warnings_as_errors=false \symbol_level=0 \
&& : "---------- Build ----------" \   && ninja d8 -C out.gn/x64.release/ -j
$(getconf _NPROCESSORS_ONLN) \   && : "---------- Extract shared libraries --
--------" \   && mkdir -p ${V8_DIR}/include ${V8_DIR}/lib \   && cp -R
/tmp/v8/include/* ${V8_DIR}/include/ \   && (cd /tmp/v8/out.gn/x64.release; \
  cp lib*.so icudtl.dat ${V8_DIR}/lib/) \   && : "---------- Copying build
dependencies ----------" \   && apk del .v8-build-dependencies .gn-runtime-
dependencies \   && mkdir -p /home/node/v8-install \   && cp -R /tmp/*
/home/node/v8-install/

RUN apk add --no-cache python python-dev python python-dev \
linux-headers build-base bash git ca-certificates && \python -m ensurepip &&
\rm -r /usr/lib/python*/ensurepip && \pip install --upgrade pip setuptools &&
\if [ ! -e /usr/bin/pip ]; then ln -s pip /usr/bin/pip ; fi && \
rm -r /root/.cache

RUN git clone https://github.com/skulpt/skulpt # Cloning Skulpt repository.

WORKDIR /skulpt  # Defining working directory

RUN pip install GitPython
RUN npm install RUN npm i npm@latest -g
RUN sed -i 's#support/d8/x64#/home/node/v8-install/v8/out.gn/x64.release#'
skulpt.py # Changing directory to the one, in which the dependency has been installed.
RUN apk --update add openjdk7-jre
RUN sed -i 's/var testInDebugMode = arguments.indexOf("--debug-mode") != -
1;//' /skulpt/test/test.js # Removing line of a file tha result on an error.

RUN 2090 CMD tail -f /dev/null # Executing without interruption.
RUN npm run build # Building Skulpt
RUN cp /skulpt/dist/* /root # Copying built files into a new docker directory.
```

Docker-compose.yml

```
version: '3'
   services:

       import:
          image: import
          ports:
             - "2090:2090"
          volumes:
             - "./:/root" # Getting the built files into the localhost.
```

ANNEX V (EXAMPLE OF USE)

IDE FUNCTION

```
def josephus(ls, skip):

  T=[ ]
# Loop that defines T with the format: T=[1,2,…,ls]
  for i in range (ls):
     T.append(i+1)

  skip -= 1 # pop automatically skips the dead guy
  idx = skip
# Loop that removes from T, the position that has been killed until only one position is alive.
  while len(T) > 1:
     T.pop(idx) # kill prisoner at  position idx
     idx = (idx + skip) % len(T)

  return str(T[0]) # returns the winning position
```

NODE-RED FLOW

Using IDE Function

```
from modules.example import josephus # importing IDE function

x = msg['payload'][0] # number of soldiers provided by the user into a new variable
y = msg['payload'][1] # Suicidal sequence provided by the user into a new variable

msg['payload'] = 'To betray, Position: ' + josephus(x,y) # result of the node using the result returned
by the IDE function

return msg
```

ANNEX VI (CASE 1: HEAT WAVE MITIGATION, GREATER LYON)

IDE FUNCTIONS

Working function

```
def working (number):
   if number =='200':
      return ('Extract Response O-MI Node: Success in reading O-MI Node Contents');
   else:
      return ('Extract Response O-MI Node: Failure in reading O-MI Node Contents');
```

Extraction of the code

```
def code (things):
  import time # importing time module

# Initializing variables
  nodeName = 'Lyon'
  lstForGeo = [ ]
  parsedGeoAndCorrespondingTemp = [ ]
  lstForPOI = [ ]
  tempList = [ ]
# Loop that defines variable inside the information provided by the temperature sensors
  for items in things['Object']:
     tempid = items['id'][0]
     if tempid == 'OrganizationalUnit:DINSI':
        templist = items['Object']
        for objs in templist:
           tempid2 = objs['id'][0]
           if tempid2 == 'Deployment:Sensing-Labs-IP68-Outdoor-Temperature-Sensor:I':
              lstForPOI = objs['Object']

# Loop that gets the date and time about when the measure is done, but also the coordinates of
each sensor.
  for geoloc in lstForPOI:
     infoItem = geoloc['InfoItem']
     ids = geoloc['id'][0]
     date = time.strftime('%Y/%m/%d')
     dayPart = time.strftime('%X')
     if dayPart[:2] == '20':
        dayPart = 'night'
     elif dayPart[:2] == '07':
        dayPart = 'morning'
     if ids[:6] == 'Sensor':
        objectItem = geoloc['Object']
        longitude = 0
        latitude = 0
        resultTemp = 0
        for item in infoItem:
          if item['$']['name'] == 'geo:long':
            longitude = item['value'][0]['_']
          elif item['$']['name'] =='geo:lat':
            latitude = item['value'][0]['_']

# Loop that receives tha temperature indicated by each sensor.
        for items in objectItem:
          if items['$']['type'] == 'sosa:Observation':
            resultTemp = items['InfoItem'][0]['value'][0]['_']
            lstForGeo.append({'geo':[longitude, latitude],'temp':resultTemp, 'date':date,
'time':dayPart})
            parsedGeoAndCorrespondingTemp.append({'lon':float(longitude),
'lat':float(latitude),'date':date, 'time':dayPart, 'temperature':float(resultTemp), 'name':ids,
'layer':'heatWave'})
# returning object with the information read by each sensor
  return parsedGeoAndCorrespondingTemp;
```

Connecting MongoDB

```
def _connect_mongo(host, port, username, password, db):
# According to the info received, one way of connection or another is provided.
  if username and password:
    mongo_uri = 'mongodb://%s:%s@%s:%s/%s' % (username, password, host, port, db)
    conn = MongoClient(mongo_uri)
  else:
    conn = MongoClient(host, port)

  return conn[db]
```

Insert Mongo

```
def insert_mongo(db, collection, data, host='mongodb', port=27017, username=None,
password=None):
  # Connecting to mongo
  db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
  try:
    res = db[collection].insert_many(data) # Inserting on the collection defined the data received.
  except BulkWriteError as bwe:
    print bwe.details
    raise

  return res
```

Read Mongo Temperatures

```
def read_mongotemplong(db, collection, sensor, host='mongodb', port=27017, username=None,
password=None, limit_num = None, no_id=True):
  time.sleep(2) # Waiting two seconds to be able to read the new contents introduced on the db.
  db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
# Getting date of the previous 33 day and the next day
  today = datetime.now( )
  DD = timedelta(days=33)
  Dd = timedelta(days=1)
  earlier = today - DD
  future = today + Dd
  earlier_str = earlier.strftime('%Y%m%d')
  future_str = future.strftime('%Y%m%d')
# Adecuating previous day to the right format
```

```
   year = earlier_str[:4]
   month = earlier_str[4:6]
   day = earlier_str[-2:]
   antdate= year + '/' + month + '/' + day
# Adecuating future day to the right format
   year = future_str[:4]
   month = future_str[4:6]
   day = future_str[-2:]
   postdate= year + '/' + month + '/' + day

 # Querying on the database according to the dates, the name of the sensor and the range of time.
  query={'$and' :[  {'date':{'$lt':postdate,'$gt':antdate}}, {'name' :
sensor},{'time':{'$lt':'08:20:00','$gt':'08:00:00'}}   ]   }
   show={'temperature':1,'date':1, 'time':1,'_id':0} # Returning just temperature, date and time.
   if limit_num:
      assert type(limit_num) is int, '@read_mongo: id is not an integer: %r' % id
      cursor = db[collection].find(query).limit(limit_num)
   else:
      cursor = db[collection].find(query,show)
   df =  pd.DataFrame(list(cursor))
# Returning result
   return df
```

Excess Heat Factor

```
def excessheatfactor(a,b):
   import numpy as np
   T=[ ]
# Assigning to T the temperatures read by the query, in a way T[0]=temperature 32 days ago.
   for i in range (33):
      x=str(32-i)
      T.append(a['temperature'][x])

   Theat = np.sum(T[:3]) # Sum of the temperature of today and previous 2 days.
   Tav=np.sum(T[3:]) # Sum of the remainded temperatures.
   Theat/=3 # Temperature average on the last 3 days
   Tav/=30 # Temperature average on the previous month
   T95 = b # Assigning 95th percentile of the month in the last 20 years
# Calculating different values
   EHIsig = Theat-T95
   EHIacc = Theat -Tav
   if EHIacc >= 1:
      y = EHIacc
   else:
      y = 1
   EHF = EHIsig*y

# Assigning date and new variable to facilitate posterior querying
   date =  a['date']['32']
   q=1
   return ([({'date':date, 'EHIsig':EHIsig, 'EHIaccreal':EHIacc, 'EHIacc':y,
           'EHF':EHF,'q':q})])
```

Read Mongo EHF

```
def read_mongoEHF(db, collection, sensor, host='mongodb', port=27017, username=None,
password=None, limit_num = None, no_id=True):
   time.sleep(2) # Waiting two seconds to be able to read the new contents introduced on the db.
   db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
   today = datetime.now( ) # Connecting MongoDB

# Similar method to get the date on the right format for the last three days
   DD = timedelta(days=3)
   Dd = timedelta(days=1)
   earlier = today - DD
   future = today + Dd
   earlier_str = earlier.strftime('%Y%m%d')
   future_str = future.strftime('%Y%m%d')

   year = earlier_str[:4]
   month = earlier_str[4:6]
   day = earlier_str[-2:]
   antdate= year + '/' + month + '/' + day

   year = future_str[:4]
   month = future_str[4:6]
   day = future_str[-2:]
   postdate= year + '/' + month + '/' + day

# Querying for date and variable included when the EHF is calculated
   query={'$and' :[  {'date':{'$lt':postdate,'$gt':antdate}},{'q': 1}   ]   }
   show={'date':1, 'EHF':1,'_id':0} # Returning not the whole record just date and EHF
   if limit_num:
      assert type(limit_num) is int, '@read_mongo: id is not an integer: %r' % id
      cursor = db[collection].find(query).limit(limit_num)
   else:
      cursor = db[collection].find(query,show)
   df =  pd.DataFrame(list(cursor))
# Returning values
   return df
```

Heatwave

```
def heatwaveEHF(befyestdata,yestdata,todata):
# Conditional sentence that announces a heatwave just if the EHFvalues read on the last three days
are greater than zero
   if befyestdata>0:
      if yestdata>0:
         if todata>0:
            return ("Lyon, We have a heatwave");
         else:
            return ("No worries, we are not in a heatwave");
      else:
         return ("No worries, we are not in a heatwave");
   else:
      return ("No worries, we are not in a heatwave");
```

NODE-RED FLOW

Extract Response O-MI Node Lyon HW

```
# Importing time and IDE functions
import time
from modules.extract import code
from modules.extract import working
# Assigning to the variables used on the IDE functions the contents needed.
response =
msg["payload"]["omiEnvelope"]["response"][0]["result"][0]["return"][0]["$"]["returnCode"];
lyonHeatWaveInfoItem =
msg["payload"]["omiEnvelope"]["response"][0]["result"][0]["msg"][0]["Objects"][0]["Object"][0]
organizationID = lyonHeatWaveInfoItem["id"][0]

# Returning the response provided by the IDE functions
node.error (working(response))
msg["payload"] = code(lyonHeatWaveInfoItem)
return msg
```

Store on MongoDBTEMP

```
import pymongo
from pymongo import MongoClient
from modules.mongofun import insert_mongo
# Providing data, host, database and collection to the IDE function
data = msg['payload']
host = 'mongodb'
db = 'myDB'
collection = 'biotope'
# Storing on MongoDB
res = insert_mongo(db, collection, data, host)
```

ReadFromMongoTEMP

```
import pymongo
from pymongo import MongoClient
from modules.mongofun import read_mongotemplong
import pandas as pd
# Defining inputs for IDE function
data = msg['payload']
host = 'mongodb'
db = 'myDB'
collection = 'biotope'
sensor = 'Sensor:SL-T-P8'
res = read_mongotemplong(db, collection, sensor, host)
# Receiving the queried data on different formats
abc = res.to_json(orient='records')
col = res.to_json(orient='columns')
# Sending the response oriented as columns
msg['payload'] = col
return msg
```

Excess Heat Factor

```
from modules.heatwave import excessheatfactor
import time
a = msg['payload']
month = time.strftime('%m') # Getting the actual month
# Providing according to the month the 95th percentile temperature in the last 20 years [85].
if month == '1':
    b = 6.75
elif month == '2':
    b = 8.125
elif month == '3':
    b = 11.125
elif month == '4':
    b = 14.0625
elif month == '5':
    b = 18.125
elif month == '6':
    b = 21.5
elif month == '7':
    b = 24.625
elif month == '8':
    b = 23.9375
elif month == '9':
    b = 20.75
elif month == '10':
    b = 15.125
elif month == '11':
    b = 10.0625
elif month == '12':
    b = 7.25
msg["payload"] = excessheatfactor(a,b)
return msg
```

ReadFromMongoEHF

```
import pymongo
from pymongo import MongoClient
from modules.mongofun import read_mongoEHF
import pandas as pd
# Providing data, host, database and collection to the IDE function
data = msg['payload']
host = 'mongodb'
db = 'myDB'
collection = 'biotope'
# Receiving the queried data on different formats
res = read_mongoEHF(db, collection, host)
abc = res.to_json(orient='records')
col = res.to_json(orient='columns')

msg['payload'] = col
return msg
```

Warning Preprocessor

```
from modules.heatwave import heatwaveEHF
# Assigning to the variables used on the IDE functions the contents needed
Tbefyest=msg['payload']['EHF']['0']
Tyest=msg['payload']['EHF']['1']
Ttoday=msg['payload']['EHF']['2']
msg["payload"] = heatwaveEHF(Tbefyest,Tyest,Ttoday)
return msg
```

ANNEX VII (CASE 2: SMART PARKING SYSTEM, FIFA WORLD CUP QATAR 2022)

IDE FUNCTIONS

Working function

```
def working (number):
   if number =='300':
      return ('Extract Response O-MI Node: Success in reading O-MI Node Contents');
   else:
      return ('Extract Response O-MI Node: Failure in reading O-MI Node Contents');
```

Extraction of the code

```
def code (things, topic):
   import time
# Initializing values
   nodeName = 'Doha'
   lstForGeo = [ ]
   parsedGeoAndCorrespondingTemp = [ ]
   lstForPOI = [ ]
   tempList = [ ]
# According to the present emergeny a loop that provides the coordinates of the vehicles is present.
   if topic == 'Health':
      for items in things['Object']:
         tempid = items['id'][0]
         if tempid == 'Health Unit':
            templist = items['Object']
            for objs in templist:
               tempid2 = objs['id'][0]
               if tempid2[:7] == 'Vehicle':
                  lstForPOI = objs['Object']
                  infoItem = objs['InfoItem']
                  for item in infoItem:
                     if item['$']['name'] == 'geo:long':
                        longitude = item['value'][0]
                     elif item['$']['name'] =='geo:lat':
                        latitude = item['value'][0]
               # Getiing the date and time ,when the measures are done
                  for geoloc in lstForPOI:
                     ids = geoloc['id'][0]
                     date = time.strftime('%Y/%m/%d')
                     dayPart = time.strftime('%X')
               # Receiving the plate and model of the vehicle
                     if ids[:7] == 'vehicle':
                        for items in geoloc['InfoItem']:
```

```
                  if items['$']['name'] == 'plate':
                     plate = items ['value'][0]
                  if items['$']['name'] == 'Model':
                     model = items ['value'][0]
          # Receiving the name and ID of the responsible person
              if ids[:6] == 'person':
                for items in geoloc['InfoItem']:
                  if items['$']['name'] == 'DNI':
                     DNI = items ['value'][0]
                  if items['$']['name'] == 'Name':
                     Name = items ['value'][0]
         # Adding into different variables the information extracted of each of the vehicles
            lstForGeo.append({'geo':[longitude, latitude], 'Plate Number':plate,'Vehicle
Model':model, 'Responsible ID':DNI,'Responsible Name':Name, 'date':date, 'time':dayPart})
            parsedGeoAndCorrespondingTemp.append({'lon':float(longitude), 'lat':float(latitude),
'Plate Number':plate,'Vehicle Model':model, 'date':date, 'Responsible ID':DNI,
                               'Responsible Name':Name, 'name':ids, 'layer':'Fifa World Cup'})
  # The same structure is followed when the fire alarm or the safety emergency occurs.
  if topic == 'Fire':
    for items in things['Object']:
      tempid = items['id'][0]
      if tempid == 'Fire Unit':
        templist = items['Object']
        for objs in templist:
          tempid2 = objs['id'][0]
          if tempid2[:7] == 'Vehicle':
            lstForPOI = objs['Object']
            infoItem = objs['InfoItem']
            for item in infoItem:
              if item['$']['name'] == 'geo:long':
                longitude = item['value'][0]
              elif item['$']['name'] =='geo:lat':
                latitude = item['value'][0]

            for geoloc in lstForPOI:
              ids = geoloc['id'][0]
              date = time.strftime('%Y/%m/%d')
              dayPart = time.strftime('%X')

              if ids[:7] == 'vehicle':
                for items in geoloc['InfoItem']:
                  if items['$']['name'] == 'plate':
                     plate = items ['value'][0]
                  if items['$']['name'] == 'Model':
                     model = items ['value'][0]

              if ids[:6] == 'person':
                for items in geoloc['InfoItem']:
                  if items['$']['name'] == 'DNI':
                     DNI = items ['value'][0]
                  if items['$']['name'] == 'Name':
                     Name = items ['value'][0]

            lstForGeo.append({'geo':[longitude, latitude], 'Plate Number':plate,'Vehicle
Model':model, 'Responsible ID':DNI,'Responsible Name':Name, 'date':date, 'time':dayPart})
            parsedGeoAndCorrespondingTemp.append({'lon':float(longitude), 'lat':float(latitude),
'Plate Number':plate,'Vehicle Model':model, 'date':date, 'Responsible ID':DNI,
                               'Responsible Name':Name, 'name':ids, 'layer':'Fifa World Cup'})
  if topic == 'Safety':
    for items in things['Object']:
      tempid = items['id'][0]
      if tempid == 'Police Station':
        templist = items['Object']
        for objs in templist:
          tempid2 = objs['id'][0]
          if tempid2[:7] == 'Vehicle':
            lstForPOI = objs['Object']
            infoItem = objs['InfoItem']
            for item in infoItem:
              if item['$']['name'] == 'geo:long':
                longitude = item['value'][0]
              elif item['$']['name'] =='geo:lat':
                latitude = item['value'][0]

            for geoloc in lstForPOI:
              ids = geoloc['id'][0]
              date = time.strftime('%Y/%m/%d')
              dayPart = time.strftime('%X')

              if ids[:7] == 'vehicle':
                for items in geoloc['InfoItem']:
                  if items['$']['name'] == 'plate':
                    plate = items ['value'][0]
                  if items['$']['name'] == 'Model':
                    model = items ['value'][0]

              if ids[:6] == 'person':
                for items in geoloc['InfoItem']:
                  if items['$']['name'] == 'DNI':
                    DNI = items ['value'][0]
                  if items['$']['name'] == 'Name':
                    Name = items ['value'][0]

            lstForGeo.append({'geo':[longitude, latitude], 'Plate Number':plate,'Vehicle
Model':model, 'Responsible ID':DNI,'Responsible Name':Name, 'date':date, 'time':dayPart})
            parsedGeoAndCorrespondingTemp.append({'lon':float(longitude), 'lat':float(latitude),
'Plate Number':plate,'Vehicle Model':model, 'date':date, 'Responsible ID':DNI,
                              'Responsible Name':Name, 'name':ids, 'layer':'Fifa World Cup'})
  return parsedGeoAndCorrespondingTemp # returning object with the information read by each
sensor
```

Connecting MongoDB

```
def _connect_mongo(host, port, username, password, db):
# According to the info received, one way of connection or another is provided.
  if username and password:
    mongo_uri = 'mongodb://%s:%s@%s:%s/%s' % (username, password, host, port, db)
    conn = MongoClient(mongo_uri)
  else:
    conn = MongoClient(host, port)

  return conn[db]
```

Insert Mongo

```
def insert_mongo(db, collection, data, host='mongodb', port=27017, username=None,
password=None):
  # Connecting to mongo
  db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
  try:
    res = db[collection].insert_many(data) # Storing in the collection defined the data provided.
  except BulkWriteError as bwe:
    print bwe.details
    raise

  return res
```

Spectator Function

```
def function(match, category, park, plate):
  result = [ ] # Initializing variable
  if park == 'Yes': # If the user wants a parking place and there are free spots.
  # According to the match of the ticket, it is assigned the date for the parking.
    if match == 'Groups1':
      date = '2018/12/05'
    elif match == 'Groups2':
      date = '2018/12/06'
    elif match == 'Quarterfinals':
      date = '2018/12/10'
    elif match == 'Semifinals':
      date = '2018/12/13'
 # According to the category of the ticket, it is assigned its closest parking zone.
    if category == 'Category 1':
      Gate = 2
    elif category == 'Category 2':
      Gate = 2
    elif category == 'Category 3':
      Gate = 1
    elif category == 'Category 4':
      Gate = 1
    elif category == 'Category 5':
      Gate = 3
 # returning the new variable with the date, gate and plate number with access to the parking.
    result = [{'date':date, 'Gate':Gate, 'Plate Number':plate}]
    return result
  else:
    return result
```

Haversine

```
def haversine (lat1, lon1, lat2, lon2):
  r = 6371 # Average Earth ratius
  x1 = math.radians(lat1-25.289266) # Difference between latitudes of vehicle1 and stadium
  y1 = math.radians(lon1-51.564183) # Difference between longitudes of vehicle1 and stadium
  h1 =
math.asin(math.sqrt(math.pow(math.sin(x1/2),2)+math.cos(lat1)*math.cos(25.289266)*math.po
w(math.sin(y1/2),2))) # Haversine calculation
  distance1 = 2*r*h1 # Distance between vehicle 1 and stadium in kilometers
```

```
   x2 = math.radians(lat2-25.289266) # Difference between latitudes of vehicle2 and stadium
   y2 = math.radians(lon2-51.564183) # Difference between longitudes of vehicle2 and stadium
   h2 =
math.asin(math.sqrt(math.pow(math.sin(x2/2),2)+math.cos(lat2)*math.cos(25.289266)*math.po
w(math.sin(y2/2),2))) # Haversine calculation
   distance2 = 2*r*h2    # Distance between vehicle 2 and stadium in kilometers
# According to the results obtained, it is expressed which vehicle is closer
if distance1 < distance2:
      result = 0
   else:
      result = 1
   return result
```

Right Gate

```
def rightgate (lat1, lon1):
   r = 6371# Average Earth ratius
# Coordinates of the different gates
   gate1 = [25.288238,51.563434]
   gate2 = [25.290555,51.564410]
   gate3 = [25.288780,51.565461]
   x1 = math.radians(lat1-gate1[0]) # Difference between latitudes of the vehicle and the gate1
   y1 = math.radians(lon1-gate1[1]) # Difference between longitudes of the vehicle and the gate1
   h1 =
math.asin(math.sqrt(math.pow(math.sin(x1/2),2)+math.cos(lat1)*math.cos(gate1[0])*math.pow(
math.sin(y1/2),2)))
   distance1 = 2*r*h1# Distance between the vehicle and the gate 1

   x2 = math.radians(lat1-gate2[0]) # Difference between latitudes of the vehicle and the gate2
   y2 = math.radians(lon1-gate2[1]) # Difference between longitudes of the vehicle and the gate2
   h2 =
math.asin(math.sqrt(math.pow(math.sin(x2/2),2)+math.cos(lat1)*math.cos(gate2[0])*math.pow(
math.sin(y2/2),2)))
   distance2 = 2*r*h2 # Distance between the vehicle and the gate 2

   x3 = math.radians(lat1-gate3[0]) # Difference between latitudes of the vehicle and the gate3
   y3 = math.radians(lon1-gate3[1]) # Difference between longitudes of the vehicle and the gate3
   h3 =
math.asin(math.sqrt(math.pow(math.sin(x3/2),2)+math.cos(lat1)*math.cos(gate3[0])*math.pow(
math.sin(y3/2),2)))
   distance3 = 2*r*h3 # Distance between the vehicle and the gate 3
# Conditional that evaluates the distances and informs about which is the closest gate
   if (distance1 < distance2 and distance1 < distance3):
      result = 'Gate 1, with lat= ' + str(gate1[0]) + ', lon= ' + str(gate1[1]) + ', is your best option'
   elif (distance2 < distance1 and distance2 < distance3):
      result = 'Gate 2, with lat= ' + str(gate2[0]) +', lon= ' + str(gate2[1]) + ', is your best option'
   else:
      result = 'Gate 3, with lat= ' + str(gate3[0]) + ', lon= ' + str(gate3[1]) + ', is your best option'
   return result
```

Access Parking

```
def read_mongoparking1(db, collection, sensor, host='mongodb', port=27017, username=None,
password=None, limit_num = None, no_id=True):
    db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
  #Method to get the data of the next and previous day.
    today = datetime.now( )
    DD = timedelta(days=1)
    Dd = timedelta(days=1)
    earlier = today - DD
    future = today + Dd
    earlier_str = earlier.strftime('%Y%m%d')
    future_str = future.strftime('%Y%m%d')

    year = earlier_str[:4]
    month = earlier_str[4:6]
    day = earlier_str[-2:]
    antdate= year + '/' + month + '/' + day

    year = future_str[:4]
    month = future_str[4:6]
    day = future_str[-2:]
    postdate= year + '/' + month + '/' + day

# Querying for the date, plate number and gate, Gate 5 is the one assigned to the emergency
vehicles that can access by any gate.
    query={'$and' :[  {'date':{'$lt':postdate,'$gt':antdate}},{'Plate Number': sensor},{'$or' :[ {'Gate': 1
},{'Gate': 5 }]} ]   }
    show={'date':1, 'Plate Number':1,'_id':0}# Returning not the whole record just date and plate.
    if limit_num:
       assert type(limit_num) is int, '@read_mongo: id is not an integer: %r' % id
       cursor = db[collection].find(query).limit(limit_num)
    else:
       cursor = db[collection].find(query,show)
    df =  pd.DataFrame(list(cursor))

    return df

# The same process is followed both for gate 2 and 3
def read_mongoparking2(db, collection, sensor, host='mongodb', port=27017, username=None,
password=None, limit_num = None, no_id=True):

    db = _connect_mongo(host=host, port=port, username=username, password=password, db=db)
    today = datetime.now( )
    DD = timedelta(days=1)
    Dd = timedelta(days=1)

    earlier = today - DD
    future = today + Dd
    earlier_str = earlier.strftime('%Y%m%d')
    future_str = future.strftime('%Y%m%d')

    year = earlier_str[:4]
    month = earlier_str[4:6]
    day = earlier_str[-2:]
    antdate= year + '/' + month + '/' + day

    year = future_str[:4]
    month = future_str[4:6]
    day = future_str[-2:]
    postdate= year + '/' + month + '/' + day
    query={'$and' :[  {'date':{'$lt':postdate,'$gt':antdate}},{'Plate Number': sensor},{'$or' :[ {'Gate': 2
},{'Gate': 5 }]}  ]   }
    show={'date':1, 'Plate Number':1,'_id':0}
    if limit_num:
       assert type(limit_num) is int, '@read_mongo: id is not an integer: %r' % id
       cursor = db[collection].find(query).limit(limit_num)
    else:
       cursor = db[collection].find(query,show)
    df =  pd.DataFrame(list(cursor))

    return df
```

NODE-RED FLOW

Extract Response FIFA Smart Parking

```
import time
from modules.extractFIFA import code
from modules.extractFIFA import working
# Defining variables employed on IDE functions
response =
msg["payload"]["omiEnvelope"]["response"][0]["result"][0]["return"][0]["$"]["returnCode"];
alarm = msg["topic"]
dohacity =
msg["payload"]["omiEnvelope"]["response"][0]["result"][0]["msg"][0]["Objects"][0]["Object"][0]
# Making use of the IDE functions
node.error (working(response))
msg["payload"] = code(dohacity,alarm)
return msg
```

Evaluate Distances

```
import math
from modules.haversine import haversine
x = msg['payload']
lat1 = x [0]['lat']
lon1 = x [0]['lon']
lat2 = x [1]['lat']
lon2 = x [1]['lon']
result = haversine (lat1,lon1,lat2,lon2)
x[result].update({'Gate' : 5})# Adding a code that allows the access by any gate
y = [x[result]]
msg['payload']= y # informing about the closest vehicle
return msg
```

Right Gate

```
import math
from modules.haversine import rightgate
x = msg['payload'][0]
lat1 = x ['lat']
lon1 = x ['lon']
# Calculating through the IDE function which is the closest gate
result = rightgate (lat1,lon1)
msg['payload']= result
return msg
```

Store on MongoParking

```
import pymongo
from pymongo import MongoClient
from modules.mongofun import insert_mongo
# Providing data, host, database and collection to the IDE function
data = msg['payload']
host = 'mongodb'
db = 'myDB'
collection = 'smartpark'
res = insert_mongo(db, collection, data, host)
```

Spectator parking

```
import math
from modules.worldcup import function
# Receiving information provided by the ticket of the stadium
x = msg['payload']
match = x [0]['Match']
category = x [0]['StadiumZone']
park = x [0]['Parking']
plate = x [0]['Plate']

# Using info to assign a parking spot closer to the seat for the day of the match
result = function (match, category, park, plate)
msg['payload']= result
return msg
```

ReadFromMongoParking

```
import pymongo
from pymongo import MongoClient
from modules.mongofun import read_mongoparking1
import pandas as pd
# Providing host, database, collection and sensor data to the IDE function
host = 'mongodb'
db = 'myDB'
collection = 'smartpark'
sensor = msg['payload'] # Informs about the plate number read by the sensor of the gate
res = read_mongoparking1(db, collection, sensor, host) # Querying using IDE function

abc = res.to_json(orient='records')
col = res.to_json(orient='columns')

# If no data is received, means that the determined vehicle does not have access to the stadium
if col == "{ }":
   msg['payload'] = "ACCESS DENIED ON GATE 1"
else: # On the contrary, the gate is open
   msg['payload'] = "OPEN GATE 1"

return msg
```

ANNEX VIII (PROJECT DOCUMENTATION.MD)

```
# Web Based Python IDE Documentation

## Introduction
Through this project, we are adding a web based Python IDE that allows users
to write their python code, load files that they have already created, check
that their code works, as it should, by using (a) an interactive screen, (b)
a debugger or (c) running directly their code and showing its output, and
last but not least, letting them save the code and import it directly to
Node-Red.

The web based Python IDE's implementation is based on
[Skulpt](https://github.com/skulpt/skulpt) but goes one step further by
adding new capabilities, such as loading/storing files, providing a common
Web-based IDE, and many more.

## Technologies used in the Web based Python IDE
We provide below a brief description of the technologies that we used and how
these interact, so as to improve the user experience and how we can integrate
this work with the Node Red environment.

### Skulpt
Skulpt is a tool that allows to write and run python code in the browser. We
took advantage of this opportunity but trying to adapt the web better to our
neeeds:

* Changing the interface by reorganazing the screens in columns.
* Letting the output screen to show also the errors that appeared when the
code is executed.
* Adding a new iframe that allows users to search for more information
* Removing the previous output when the code is newly executed.
* Creating new buttons and functions to gain new functionalities.

### Node.js Added Modules
We have created different buttons to get new functionalities in our site. For
example using the button Initial Page of the iframe, the user can go directly
to the google search from wherever he might have gone.
However, there are two functions that deserve to be studied with more detail.
These are the ones related with the inputs and buttons of **"Save text to
File"** and **"Load"**.

#### Save Text to File
When you press this botton, the next steps are processed internally:

First, the text that is written in the editor  screen is read using
**editor.getValue()**. The text that is written in the input box is also read
using **document.getElementById(id)**. Then two separate processes start
simultaneously.
```

~~~
One of them, just initializes a download with the name of the input box and
with the code that was written in the editor screen. This is okay because we
can get the text in a file in the downloads directory, however it is nos
accesible for Node-Red.

The second and most important one, sends the data to one url that we have
specified, being  data.body.filename the text written in the input box and
data.body.content the text of the editor screen.

#### Load
The Load button runs a javascript function that basically removes the text
that was written in the editor screen, reads the input box and sends its
contents to the url that we have specified. After that it stays listening to
the url to see if it receives something.

If it does, it directly writes in the editor screen the contents that it has
received using **editor.setValue(contents)**.

## Miscellaneous

In order to gain speed and improve our experience, we have created one python
file located in the directory *knaas-uiaas/misc* called commands.py with some
automated functions that may be run as follows:

```
- python -c 'from commands import run; run()' (initializes the swarm and run
our service)
- python -c 'from commands import stopall; stopall()' (to leave the swarm and
stop all the containers)
- python -c 'from commands import stop; stop("x")' (stop a container)
- python -c 'from commands import modi; modi("x")' (builds automatically one
image and makes it run)
```
  Where x refers to the name of the image we want to work with, in our case
firstimage (web), nodeimage (nodesjs), node-red (Node RED), o-mi-node
(dataLyon), mongo (mongodb).

## Future Improvements
- Further extension of Skulpt with new modules.
- Adding auto-completion
- Further testing
~~~

ANNEX IX (AUTOMATED DOCKER COMMANDS)

```
import subprocess

def stop (x):
# Receiving the docker Id accroding to the image name (x) making use of: sudo docker ps -a -q  --
filter ancestor=" + x + ", then the container can be stopped.
   bashCommand = "sudo docker container stop $(sudo docker ps -a -q  --filter ancestor=" + x + ")"
   process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
   output, error = process.communicate( )

   return;

def exe (x):
# in the same way, the container id is received according to x and after that it is possible to access
to the docker container using exec -it.
   bashCommand = "sudo docker exec -it $(sudo docker ps -a -q  --filter ancestor=" + x + ") /bin/sh"
   process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
   output, error = process.communicate( )

   return;

def run ( ):
# This function allows the whole service to be run. Going to the docker-compose.yml directory,
initializing the swarm and deploying the application.
   bashCommand = 'cd ..;sudo docker swarm init;sudo docker stack deploy -c docker-compose.yml
firstcompose'
   process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
   output, error = process.communicate( )

   return;

def modi(x):
# This function receives the name of the image that has just been modified, then access to the
directory of the respective dockerfile, building it and lastly, running it on its respective port.
   if x == 'firstimage':
      bashCommand = 'cd ..;cd pysite;cd web;sudo docker build -t ' + x + ' .; sudo docker run -p
4000:80 ' + x
      process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
      output, error = process.communicate( )

      return;

   elif x == 'nodeimage':
      bashCommand = 'cd ..;cd pysite;cd Nodejs;sudo docker build -t ' + x + ' .; sudo docker run -p
5080:5080 -p 8090:8090 ' + x
      process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
      output, error = process.communicate( )

      return;
```

```
  elif x == 'mongo':
    bashCommand = 'cd ..;cd mongodb;sudo docker build -t ' + x + ' .; sudo docker run -p
27017:27017 ' + x
    process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
    output, error = process.communicate( )

    return;

  elif x == 'node-red':
    bashCommand = 'cd ..;cd node-red;sudo docker build -t ' + x + ' .; sudo docker run -p 1880:1880
' + x
    process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
    output, error = process.communicate( )

    return;

  elif x == 'o-mi-node':
    bashCommand = 'cd ..;cd o-mi-node;sudo docker build -t ' + x + ' .; sudo docker run -p
8080:8080 ' + x
    process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
    output, error = process.communicate( )
    return;

  return;

def stopall ( ):
# This function acceses to the docker-compose.yml directory and stops the whole service at once.
  bashCommand = 'cd ..;sudo docker swarm leave --force'
  process = subprocess.Popen(bashCommand, stdout=subprocess.PIPE, shell=True)
  output, error = process.communicate( )

  return;
```