DEVELOPMENT OF SOFTWARE COMPONENTS
IN THE uLAB SCOPE

Final Degree Project Management

Author

Carlos Person Haro

Director
José Ramon Herrero Zaragoza

Co-Director
Anna Maria Casas Casademunt

14th March, 2014

Computer Science Degree
Software Engineering

Barcelona School of Informatics
Polytechnic University of Catalonia

FIB - UPC
First of all, I would like to thank Rosa Mª Martín and Anna Casas for their help and project management, and for giving me the chance to work on my final degree project on a very professional and qualified environment.

I would also like to thank all of the uLab development team: Jaume Bosch, Jonatan Fernández, Marc Cobos, Marc Comino, Maria Josep Rodríguez and Ola Dziegielewsk. I have enjoyed my time working with all of you. This project would not have been possible without your hard work and dedication.

Besides the uLab team, this project would not have been possible without Telefónica I+D, especially Miguel Ángel Cañas and Agustín Fernández, and the inLab FIB. I would like to thank them for all the trust they put on us.

I would also like to give my most sincere thanks to everyone that has assisted me in the writing of this document, especially Josep-Ramón Herrero, Ferran Sabaté, Néstor Malet and Maria Josep Rodriguez, who have been very helpful and have advised me in many ways.

Of course, I would like to thank our JASP1 neighbours, especially Albert Almà and David Rodriguez, for all the laughs we had and for their always excellent mood.

I cannot end this appreciation section without referring to Isaac Obradors. There is so much I have to thank you for, that I would not know where to begin. Thank you, for everything.

Finally, I would like to thank my family for all their support. Without their effort and all the trust they put on me, any of this would have been possible.
Telefónica I+D és una empresa dins del grup Telefónica que està dedicada a la investigació i el desenvolupament. El seu objectiu principal és la creació de nous productes, serveis i prototips, així com l’exploració de noves technologies i frameworks. Un cop aquestes technologies han estat avaluades, es decideix si és convenient afegir-les al seu technology stack, amb la intenció d’utilitzar-les en projectes futurs.

Aquest projecte ha estat desenvolupat en el context de Telefónica I+D. Consisteix en una sèrie d’avaluacions tecnològiques i en l’elaboració de prototips fent servir aquestes mateixes tecnologies. El projecte ha estat realitzat per l’equip uLab UPC de l’inLab FIB en col·laboració amb Telefónica I+D.

Aquestes avaluacions estan enfocades a testejar i validar diverses technologies amb l’objectiu de verificar que poden ser d’utilitat per els productes de Telefónica I+D. Les technologies i frameworks que han estat avaluats són AngularJS, Jasmine, WJMO i Docker.
ABSTRACT (ESPAÑOL)

Telefónica I+D es una empresa dentro del grupo Telefónica que está dedicada a la investigación y el desarrollo. Su objetivo principal es la creación de nuevos productos, servicios y prototipos, así como explorar nuevas tecnologías y frameworks. Una vez que estas tecnologías han sido evaluadas, se decide si es conveniente añadirlas a su technology stack, para utilizarlas más adelante en futuros proyectos.

Este proyecto ha sido desarrollado en el contexto de Telefónica I+D. Consiste en una serie de evaluaciones tecnológicas y en la elaboración de prototipos usando dichas tecnologías. El proyecto ha sido llevado a cabo por el equipo uLab UPC del inLab FIB, en colaboración con Telefónica I+D.

Dichas evaluaciones están enfocadas a testear y validar diversas tecnologías con el objetivo de verificar que pueden ser de utilidad para los productos de Telefónica I+D. Las tecnologías y frameworks que han sido evaluados son AngularJS, Jasmine, WIJMO y Docker.
ABSTRACT (ENGLISH)

Telefónica I+D is a research and development enterprise within the Telefónica Group. Their main goal is to create new products, services and prototypes, as well as to explore new technologies and frameworks. Once these technologies are evaluated, they decide whether or not to include them in their technology stack in order to use them in the following projects.

This project has been done in the Telefónica I+D’s context. It consists of a series of technological evaluations as well as the elaboration of some prototypes using said technologies. The project has been done by the uLab UPC team from inLab FIB in collaboration with Telefónica I+D.

These evaluations are aimed to test and evaluate several new technologies in order to validate if those can be useful for Telefónica I+D products. The technologies and frameworks that have been evaluated are AngularJS, Jasmine, WUMO and Docker.
INDEX

1. Introduction.................................................................................................................. 4
2. Project management ..................................................................................................... 6
  2.1 Context....................................................................................................................... 6
    2.1.1 Stakeholders ......................................................................................................... 6
      2.1.1.1 Telefónica I+D .............................................................................................. 6
      2.1.1.2 inLab FIB ....................................................................................................... 6
      2.1.1.3 Director and co-director ............................................................................... 7
      2.1.1.4 Author ........................................................................................................... 7
      2.1.1.5 Final user ....................................................................................................... 7
  2.2 State of the art .......................................................................................................... 8
    2.2.1 Web frameworks .................................................................................................. 8
    2.2.2 Application deployment ...................................................................................... 10
  2.3 Scope ......................................................................................................................... 12
    2.3.1 Live Scanner ........................................................................................................ 12
    2.3.2 Docker ................................................................................................................. 13
  2.4 Methodology ............................................................................................................ 14
    2.4.1 Introduction ......................................................................................................... 14
    2.4.2 Possible obstacles and planned solutions ............................................................ 14
      2.4.2.1 Lack of documentation .................................................................................. 14
      2.4.2.2 Bugs ............................................................................................................. 15
      2.4.2.3 Timetable ..................................................................................................... 15
    2.4.3 Development tools .............................................................................................. 15
    2.4.4 Validation methods .............................................................................................. 16
  2.5 General schedule ...................................................................................................... 17
    2.5.1 Estimated duration .............................................................................................. 17
    2.5.2 Considerations ..................................................................................................... 17
  2.6 Project planning ........................................................................................................ 17
    2.6.1 Estimated time ..................................................................................................... 17
    2.6.2 Project iterations ................................................................................................. 17
      2.6.2.1 Planning and feasibility ............................................................................... 18
      2.6.2.2 Documentation and presentation ................................................................. 18
1. Introduction

As we can see in our daily lives, technology evolution is blazing fast. This is not something new: it has been like this for a long time from now, and there is no reason to think that this will change anytime soon. Since computer science related technology affects most of the population in one way or another, enterprises are especially interested in keeping themselves ahead of their competition. This way, they can explore new paths to make their products more profitable.

Telefónica I+D is an enterprise within the Telefónica Group that’s dedicated entirely to this matter: its purpose is to continually look for innovation opportunities in the technological scope. This way, they are responsible of keeping the Telefónica’s technology stack up to date and with the optimal choices.

Lately, Telefónica I+D has been establishing development deals with several Spanish universities. Once the contact is settled, computer science student teams are built. These teams are called “uLabs”. This is a way for them to have computer science students, who have a strong academic bias, work together on projects that are aligned with their goals.

With these considerations in mind, this is a project for Telefónica I+D that will focus on trying and exploring new software technologies and frameworks. Its final purpose is to find ways to improve software development and deployment, as well as evaluating if these solutions are useful or mature enough to be added to the Telefónica I+D technology stack.

This project consists in two side-projects: Live Scanner and Docker.

1.2 Live Scanner

Telefónica has control over some of the Brasilian’s TV shows. They are building a platform that will allow them to dynamically recommend some of the shows that are being shown. These recommendations will be published in Twitter by another external application.

However, a web portal is needed in order to send these recommendations to the application that will post them on Twitter. Live Scanner will be the application responsible of that matter: it will retrieve all of the live shows, show them on a list and finally send the selected shows to the external Twitter application. In order to improve the impact of these recommendations, Live Scanner will let the user select the reason of his recommendation, which will be chosen from a pre-defined list.
1.3 Docker

Docker is a newly created open-source software project that is capable of automating some of the work needed for the deployment of applications. It is based on the standard Linux containers (LXC) and helps with the task of creating isolated environments for app deployments.

The goal of the Docker side-project is exploring the functionalities that Docker offers by using it in a development environment. For this matter, the application that is going to be deployed will be Live Scanner, the previously created web application.
2 Project management

2.2 Context

The purpose of this section is to describe the project context and references, including stakeholders and the state of the art. There are several stakeholders involved in this project. In the next section, we are going to describe their roles and how will they contribute in this project.

2.2.1 Stakeholders

2.2.1.1 Telefónica I+D

As it has been previously stated, Telefónica I+D (from now on, TID) is an enterprise within the Telefónica Group that is entirely dedicated to research and development [1]. TID is this project’s client, but they are going to be assisting in the development as well. Thus, they will be assuming two roles: client and technical leader.

2.2.1.2 inLab FIB

The inLab FIB is, in its own words: “an innovation and research lab based in the Barcelona School of Informatics, Universitat Politècnica de Catalunya - Barcelona Tech (UPC) that integrates academic personnel from different UPC departments and its own technical staff to provide solutions to a wide range of demands that involve several areas of expertise”. [2]

The inLab FIB is where this project will be developed, as it will provide the technical knowledge, the technological infrastructure and the materials needed to the project success. The team destined to develop TID’s projects is called uLab, and the author of this project is one of its members. However, only Maria Josep Rodriguez and the author of this project are destined to the development of this final degree project.
2.2.1.3 Director and co-director

The director and co-director of this project are both inLab members. The director is Josep Ramon Herrero Zaragoza and the co-director is Anna Maria Casas Casademunt. As members of inLab FIB, they share the same role in assisting the team with the project management and providing technical assistance.

2.2.1.4 Author

As it was previously stated, the author of this project will be the developer of this project alongside Maria Josep Rodríguez. His roles involve all of the development lifecycle: analyst, designer, developer and tester.

As a member of the inLab FIB, its role is to ensure the project success, as well as accomplishing the goals of its final degree project.

2.2.1.5 Final user

The final user of this project are TID’s developers and, by extension, the Telefónica Group. As stated in previous sections of this document, the goal of this project is to test and evaluate new technologies and frameworks that are not used on TID just yet. These evaluations will be very valuable to TID in order to enhance their technology stack and knowledge, therefore improving their developer’s resources management.
2.3 State of the art

Next we will be studying this project’s technological context. We are going to introduce web frameworks and application deployments as well as their current status, and next we will focus on the particular technologies that we will be using.

2.3.1 Web frameworks

Software frameworks are platforms that assist in the development of other software. Their main goal is to provide generic functionality to the developers in order to facilitate their work. There are some non-application specific tasks in software development that can be automated, such as handling database connections and managing users. This way, the developers can invest their time into meeting the software requirements instead of having to handle other lower level tasks.

Web frameworks, more precisely, aim to facilitate the most common tasks on web applications. Examples of these tasks are database connections, session management, templates management and data mapping. This allows the developers to implement these functionalities in a much lower amount of time and without introducing any bugs, therefore improving their productivity.

Next we are going to introduce the particular web frameworks that will be used in this project.

2.3.1.1 AngularJS

Angular is a web framework developed by Google that was first released on 2009, and has been actively update since then. It is a web framework that assists with running single-page applications by implementing the Model-View-Controller (MVC) software pattern and by offering an attribute binding system.

We found several opinions about AngularJS, however, most of the references we found were blog articles [3][4][5][6][7] and the official success stories [8]. While these sources are useful and give a good hint of the framework’s capabilities, a more reliable source of information is needed. Thus, we searched and found a reference book named Mastering Web Applications with AngularJS [9]. It is a reference book found at the CERN Document Server [10] that gives a lot of insight about AngularJS. It is a relatively recent book that will be very helpful in this project’s AngularJS development.
2.3.1.2 Wijmo

The Wijmo UI widget kit was first released on April 2013, and has even fewer critical opinions available. We found several blog and forums opinions about the kit [10][11][12], but they cannot be considered reliable for a professional evaluation. From these references, we realize that Wijmo works very well in conjunction with jQuery, another popular JavaScript framework. It is stated that it is also compatible with AngularJS, but there is not nearly as much documentation about its usage.

However, there is a book that covers some of the Wijmo functionalities: Building UIs with Wijmo (Yuguang Zhang). Sadly, it covers the jQuery side of Wijmo, and not the AngularJS one, and thus it will not be a reference book.

All these reasons lead us to decide that further testing is needed in order to give a consistent evaluation. For that matter, we will rely on Wijmo’s customer services and forums in order to solve any issues that may rise.
2.3.1.3  Jasmine

Jasmine is a JavaScript testing framework. It offers a convenient web interface unlike other testing frameworks like Mocha.

Jasmine was released on September 2010, and it has been actively maintained until now. We found several opinions about its usage [14][15][16][17][18], but as the previous technologies, most of them were from blogs. However, this is a simple testing framework and it’s not as crucial as the previous ones. We found a paper regarding unit testing in Java [19] that should give insight about unit testing, although the programming language is not JavaScript.

Jasmine is meant to be very useful in order to solidify the knowledge about the behaviour of the other technologies. Thus, it will be both a tool for the evaluations and a technology to be evaluated itself.

2.3.2  Application deployment

Creating an application involves much more than designing and developing it. Creating an application also means deploying and maintaining it. The problem is that the machines where this software is going to be installed may not have the same installed environment. In fact, it is highly likely that the development and production environments differ extremely. There are plenty of variables: operating systems, varying system paths, different application versions, application dependencies, and so on.

There are two solutions to this issue: standardizing all the computer environments or establishing isolated mini-environments for each application. The first option is simply not possible nowadays, as the operating systems differ greatly as well as the user’s needs. The second option is much more viable, and many applications and enterprises have risen from this idea. The mainly used products are called Virtual Machines: they provide what is called a virtualized operating system. This means that there is an operating system running within another operating system, providing each one with their own protected environments.
Docker, one of the technologies to be used in this project, aims to further iterate on this concept.

2.3.2.1 Docker

Docker is a relatively new open-source project that started on June 2010 [20][21]. It has a rather complete reference [22][23] and it aims to be a more efficient approach to traditional virtualization methods [24], but operating only on Linux-based operating systems. It is based on several Linux standards, primarily LXC [25] and it uses the AUFS file system technology [26], although it also supports others such as unionFS [27]. The opinions found about the Docker usage seem to be very positive [28] and their own creators are very capable of explaining its very interesting features [29]. The Docker team has progressed immensely since the project inception, winning many awards and fundings.

One of the main differences with the other virtualization methods is that the virtualized environments are lightweight and reusable. That way, many more instances can be run at the same machines, greatly optimizing the costs. One of the other main features of Docker is that it has a social component on its public Index, a public repository where the developers can publish their virtualized Docker applications. Over 5.000 applications are now available at the public Docker Index [30].
2.4 Scope

There are 2 major goals in the roadmap of the uLab team: Live Scanner and Docker. We will now explain each one of them.

2.4.1 Live Scanner

Telefónica has control over some of the Brasilian’s TV shows. They are building a platform that will allow them to dynamically recommend some of the shows that are being shown. These recommendations will be published in Twitter by another external application.

However, a web portal is needed in order to send these recommendations to the application that will post them on Twitter. Live Scanner will be the application responsible of that matter: it will retrieve all of the live shows, show them on a list and finally send the selected shows to the external Twitter application. In order to improve the impact of these recommendations, Live Scanner will let the user select the reason of his recommendation. These reasons will be chosen from a pre-defined list.

This side-project has two goals. The first one is to test out several new technologies and frameworks. The second one is to create a product prototype that can be easily expanded with new features. Some of the technologies that will be used to develop this project are new to Telefónica I+D, and these are the ones that are going to be tested for evaluation:

- AngularJS, a Model View Controller (MVC) Javascript framework
- Wijmo, a User Interface (UI) widget kit
- Jasmine, a Javascript Unit Testing framework

In order to summarize, these are the Live Scanner project goals:

<table>
<thead>
<tr>
<th>Technology evaluations</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngularJS</td>
<td></td>
</tr>
<tr>
<td>Wijmo</td>
<td>Live Scanner (Web application)</td>
</tr>
<tr>
<td>Jasmine</td>
<td></td>
</tr>
</tbody>
</table>
2.4.2 Docker

Docker is a newly created open-source software project that is capable of automating some of the work needed for the deployment of applications. It is based on the standard Linux containers (LXC) and helps with the task of creating isolated environments for app deployments. Its concept is much like the one of the virtual machines (VM), without the performance and disk usage drawbacks that VMs tend to carry. These isolated environments are called “images”, and they can be shared and distributed through the central public repository that Docker provides, which is called Registry or Docker Hub.

The goal of this side-project is exploring the functionalities that Docker offers by using it in a development environment. One of the team’s responsibilities will be to learn how Docker is used. The lack of information is very likely due to its recent creation. Therefore, research and study tasks are expected in order to find out solutions when the available tutorials and references aren’t enough. Taking this facts into consideration, the creation of new documentation would be highly appreciated if Docker is evaluated positively.

To fully test its features, the application that we will be deploying with Docker will be the previously created one: Live Scanner. In order to test it in a pseudo-production environment, this deployment will be performed on AmazonWS Linux instances. These Amazon instances will be created and managed entirely by the team.

Telefónica I+D is also interested in exploring the possibility of creating a private Registry. With a private Registry, the developers would be able to distribute the Docker images within the enterprise environment, improving the team communication greatly. This way, they could also perform management tasks on their own Registry.

Finally, TID wants to to make it even easier for the developers to deploy applications with Docker. In order to do so, we will create an auto deployment tool that manages almost all of the contact with Docker and AmazonWS. This way, it will be very simple for the developers to quickly deploy isolated applications on remote servers.

In order to summarize, these are the Docker project goals:

<table>
<thead>
<tr>
<th>Technology evaluations</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker Registry</td>
<td>New documentation</td>
</tr>
<tr>
<td>Docker Images</td>
<td>Auto deployment tool</td>
</tr>
<tr>
<td>Dockerfile</td>
<td>Private Registry</td>
</tr>
<tr>
<td></td>
<td>Live Scanner deployment</td>
</tr>
</tbody>
</table>
2.5 Methodology

2.5.1 Introduction

As a new-tech project, its evolving nature suggests that an agile methodology would be the best fit. Since we are distributed in teams of approximately 2-3 developers and because this will be a research and study project and there are many unknown factors, SCRUMBAN is the best fitted agile methodology for this project.

The main reason that made us chose an agile methodology was that it allowed us to manage possible impediments. The research nature of the project implies that the requirements could change anytime. Therefore, we needed a methodology compatible with the dynamic establishment of requirements, and agile methodologies provide this.

From all the other agile methodologies, we chose SCRUMBAN. That is because it will provide us the agile methodologies’ flexibility combined with a no sprint-based workflow, as well as a more relaxed way of defining the user stories.

Nevertheless, “no sprints” doesn’t mean “no communication”. It is very important to keep in touch with TID in order to reduce the chance of misunderstandings. To accomplish this, there will be weekly videoconference meetings with the client (TID) as well as chat-based communication for quick doubt resolutions. This will ensure that we receive as much feedback as possible. On the other side, internally the team will meet on a daily basis in order to quickly explain their work and potential obstacles.

The tools used to help the team keep this communication will be the following:

- Trello – for internal team organization of the user stories
- Basecamp – for big releases and evaluations
- Skype/Google Hangouts – for client chat and videoconference calls
- A blackboard – for easier discussing of the user stories and tasks at the daily meetings

2.5.2 Possible obstacles and planned solutions

As explained in the methodology section, the dynamic establishment of requirements is covered by the chosen methodology. That simplifies the requirements obstacles management. Therefore, we should aim to predict what other issues could arise.

2.5.2.1 Lack of documentation

Due to the strong research approach that this project has, the lack of enough useful documentation can be an obstacle. However all of the technologies planned for evaluation
have their own documentation and references to some extent. This lack may impact some
time schedules, but this has already been taken into consideration by the project leader.

**Probability: High**

### 2.5.2.2 Bugs

As with any software project, bugs are likely to appear at some point of the development
process. These bugs could be produced by our team or, due to the short lifecycle some of the
tested technologies have, by their creators. However it is planned to perform unit tests and
end-to-end tests in order to detect any bugs beforehand.

**Probability: Medium**

### 2.5.2.3 Timetable

As said before, this is a research and study project and that implies that it’s difficult to know
exactly how much time will be needed to finish each of its parts. To fix this issue, if at any point
on the project there’s a part that’s clearly taking much more time than it was expected to, it
will be simplified and the reasons to do so will be thoughtfully documented.

**Probability: Low**

### 2.5.3 Development tools

The team will be using a Linux operating system as it’s been considered the best fit for the
planned projects’ development. For simplicity and homogeneity, the same Linux Mint virtual
machine image will be distributed to the team.

As of particular code development tools, the following will be used:

- PyCharm – a python-oriented IDE
- Sublime Text 2 – a strong developer-oriented text editor

In order to keep code organized and to be able to code in a distributed way efficiently, the
Concurrent Versions System (CVS) Git will be used. GitHub is the Git repository of choice since
it provides a good process-tracking system and ease of use.

Finally, at the Docker side-project we will create, launch and manage AmazonWS Linux
instances.
2.5.4 Validation methods

The fact that we will be using unit testing and end-to-end tests combined with the weekly client meetings will ensure that the quality and goals of the project are met on a regular basis, needing no further measures of validation. This has been agreed from both the development team and the client, TID.
2.6 General schedule

2.6.1 Estimated duration

The project duration is set to approximately 4 months. The project has already started on February 17th, 2014 and the deadline is on May 27th, 2014.

2.6.2 Considerations

It is important to note that the initial planning could be revised and changed during the realization of the project. This is because we will be using an agile methodology and this implies that the client can introduce new requirements that can change the initial planning.

Furthermore, this is a research and development project and it is possible that a few of the requirements can’t be met due to technology limitations. Given that situation, said technologies would be considered invalid and the planning would be updated accordingly.

2.7 Project planning

2.7.1 Estimated time

The total amount of time required to complete the project is calculated below. Each of these iterations will be described in the following table.

<table>
<thead>
<tr>
<th>Iteration name</th>
<th>Time unit</th>
<th>Hours/unit (hours)</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning and feasibility</td>
<td>3 credits</td>
<td>30 hours</td>
<td>90 hours</td>
</tr>
<tr>
<td>Live Scanner &amp; Docker Projects</td>
<td>14 weeks</td>
<td>25 hours</td>
<td>350 hours</td>
</tr>
<tr>
<td>Documentation &amp; Presentation</td>
<td>10 weeks</td>
<td>8 hours</td>
<td>80 hours</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>505 hours</strong></td>
</tr>
</tbody>
</table>

2.7.2 Project iterations

This section will describe each of the project iterations. Each one of the will be briefly explained and its time workload will be highlighted.
Note: It has been agreed with Telefónica I+D that the weekly meetings will be every Thursday from 12:00 to 13:00. That is why every release and milestone has been adjusted to be ended on Thursday if possible.

2.7.2.1 Planning and feasibility

This iteration is currently running. It involves everything a project manager needs to know in order to decide if a project is feasible. Those documents are the scope definition, project planning, project budgeting, project context and references and the project initial presentation.

It is planned to be completed in 39 days, on a total of 90 hours.

2.7.2.2 Documentation and presentation

This iteration is aimed to the creation of the needed project documentation to be delivered at the end of the project. It will involve the software architecture and deployment diagrams. It will also involve the evaluations and conclusions about the technologies that we will be testing, and it can potentially include new documentation about them created by ourselves.

2.7.2.3 Live Scanner

This iteration is aimed to the research and evaluation of the technologies related to the Live Scanner project. The iteration has been split as follows:

- **Virtual Machine Setup**
  
  This initial setup is needed in order to provide the team a homogenised development environment. It is relatively short but it is mandatory in order to avoid potential problems in almost all of the projects we will be working.

- **Release: Angular Prototype**
  
  This release will show the Angular prototype that is defined at the project scope. It could be parallelized, so it has been divided in 2 tasks:

  - Front-end, which will involve all of the UI and web design.
    Technologies used: HTML, CSS, Angular.
  - Back-end, which will involve all of the server-side management and processes.
    Technologies used: Python, Django.

- **Release: Wijmo**

  This release will show how the Wijmo UI tools have been adapted to the Angular prototype.
• **Release: Jasmine**

  This release is aimed to show how the Angular Prototype has been covered with tests, including end-to-end testing, with the Jasmine framework.

2.7.2.4 **Docker**

This iteration is aimed to the research and evaluation of the technologies related to the Docker project. Unlike the others, this iteration has not been split in parallel tasks. This is because the team had experience with some of the technologies used in the previous iterations, but that is not the case with Docker. That said, the team will work together in order to achieve better results.

The iteration has been split as follows:

• **Release: Private Registry**

  This iteration is aimed to make contact with Docker environment and make a few initial trials. Once that knowledge is settled, the private Registry will be studied and created.

• **Release: Live Scanner deployment**

  This release will show that the previously developed application has been successfully deployed with Docker on an AmazonWS instance.

• **Release: Auto deployment tool**

  This release will showcase the developers’ deployment tool created by the team.

2.7.3 **Gantt Diagram**

2.7.3.1 **Introduction**

We will now show a Gantt diagram in order to give a more accurate vision about the project time planning.

It will also be shown which team member has been assigned to each task, as this is a project that will be done in conjunction with Maria Josep Rodriguez, another member on my team that is also working on his TFG. In this diagram we can clearly see which tasks have been parallelized and which couldn’t, as well as a blocked structure for each subproject (Live Scanner and Docker).
2.7.3.2 Diagram

Part 1

Part 2
2.7.3.3 Task summary

<table>
<thead>
<tr>
<th></th>
<th>Nombre</th>
<th>Duración</th>
<th>Incio</th>
<th>Fin</th>
<th>Predecesores</th>
<th>Recursos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Planning and feasibility</td>
<td>39.2d</td>
<td>17/02/2014</td>
<td>21/02/2014</td>
<td>Carlos Person Haro</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Documentation and presentation</td>
<td>92.4d</td>
<td>17/02/2014</td>
<td>27/02/2014</td>
<td>Carlos Person Haro</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Live Scanner</td>
<td>50.6d</td>
<td>17/02/2014</td>
<td>01/04/2014</td>
<td>Carlos Person Haro</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Virtual Machine Setup</td>
<td>7.2d</td>
<td>17/02/2014</td>
<td>21/02/2014</td>
<td>Carlos Person Haro</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Release: Angular prototype</td>
<td>26.4d</td>
<td>24/02/2014</td>
<td>18/03/2014</td>
<td>4</td>
<td>Marta Josep Rodriguez</td>
</tr>
<tr>
<td>6</td>
<td>Angular Front-end</td>
<td>26.4d</td>
<td>24/02/2014</td>
<td>18/03/2014</td>
<td>4</td>
<td>Marta Josep Rodriguez</td>
</tr>
<tr>
<td>7</td>
<td>Angular Back-end</td>
<td>26.4d</td>
<td>24/02/2014</td>
<td>18/03/2014</td>
<td>4</td>
<td>Carlos Person Haro</td>
</tr>
<tr>
<td>8</td>
<td>Release: Wijmo</td>
<td>16.2d</td>
<td>19/03/2014</td>
<td>01/04/2014</td>
<td>4, 4.5</td>
<td>Marta Josep Rodriguez</td>
</tr>
<tr>
<td>9</td>
<td>Release: Jasmine</td>
<td>15.2d</td>
<td>19/03/2014</td>
<td>01/04/2014</td>
<td>4, 4.5</td>
<td>Carlos Person Haro</td>
</tr>
<tr>
<td>10</td>
<td>Live Scanner complete</td>
<td>1d</td>
<td>01/04/2014</td>
<td>01/04/2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Docker</td>
<td>83.4d</td>
<td>02/04/2014</td>
<td>27/05/2014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Release: Private Registry</td>
<td>31.2d</td>
<td>02/04/2014</td>
<td>29/04/2014</td>
<td>4</td>
<td>Carlos Person Haro, Marta Josep Rodriguez</td>
</tr>
<tr>
<td>13</td>
<td>Live Scanner &amp; Private Chats deployment</td>
<td>15.2d</td>
<td>30/04/2014</td>
<td>13/05/2014</td>
<td>3, 4</td>
<td>Carlos Person Haro, Marta Josep Rodriguez</td>
</tr>
<tr>
<td>14</td>
<td>Release: Auto deployment tool</td>
<td>15.2d</td>
<td>14/05/2014</td>
<td>27/05/2014</td>
<td>4, 13</td>
<td>Carlos Person Haro, Marta Josep Rodriguez</td>
</tr>
<tr>
<td>15</td>
<td>Docker complete</td>
<td>1d</td>
<td>27/05/2014</td>
<td>27/05/2014</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The resources used for this project will be the following:

**Hardware**

- Workplace desktop computer
- Personal desktop computer

**Software**

- Microsoft Windows 7 Professional
- Microsoft Word
- Microsoft Excel
- Microsoft Visio
- VMWare Player + VMWare Tools
- Linux Mint 15 Olivia
- PyCharm
- Sublime Text 2
2.8 Project budget

2.8.1 Introduction

This document’s purpose is to describe the project budget and enumerate all the related costs. Next we will explain how this budget is going to be controlled. Finally, we will justify its economic viability and the environment and social impacts related to the project.

2.8.2 Budget

2.8.2.1 Wages budget

First we will bring back the time planning we showed at the Project Planning section of this document. Then we will apply the established wages and team sizes:

<table>
<thead>
<tr>
<th>Iteration name</th>
<th>Time</th>
<th>Wage</th>
<th>Team size</th>
<th>Activity Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning and feasibility</td>
<td>90 hours</td>
<td>7,5 €/hour</td>
<td>1</td>
<td>675 €</td>
</tr>
<tr>
<td>Live Scanner sub-total</td>
<td>150 hours</td>
<td>7,5 €/hour</td>
<td>1-2</td>
<td>2.062,5 €</td>
</tr>
<tr>
<td>Virtual Machine Setup</td>
<td>25 hours</td>
<td>7,5 €/hour</td>
<td>1</td>
<td>187,5 €</td>
</tr>
<tr>
<td>Release: Angular prototype</td>
<td>75 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>1125 €</td>
</tr>
<tr>
<td>Release: Wijmo &amp; Jasmine</td>
<td>50 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>750 €</td>
</tr>
<tr>
<td>Docker sub-total</td>
<td>200 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>3.000 €</td>
</tr>
<tr>
<td>Release: Private Registry</td>
<td>100 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>1.500 €</td>
</tr>
<tr>
<td>LiveScanner</td>
<td>50 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>750 €</td>
</tr>
<tr>
<td>Release: Auto deployment tool</td>
<td>50 hours</td>
<td>7,5 €/hour</td>
<td>2</td>
<td>750 €</td>
</tr>
<tr>
<td>Documentation &amp; Presentation</td>
<td>80 hours</td>
<td>7,5 €/hour</td>
<td>1</td>
<td>600 €</td>
</tr>
<tr>
<td>Total</td>
<td>505 hours</td>
<td></td>
<td></td>
<td>6337,5 €</td>
</tr>
</tbody>
</table>
As we can see in the table above, the total cost of the development team is 6337,5 €. It should be noted that the development team consists of grant holder students, and thus the established wage per hour is 7,5€/h. If the team consisted of computer science engineers, the wages would be over 25€/h approximately.

Next we should also consider the project manager costs. The weekly meetings are scheduled to be 1 hour length, and the project is planned for 14 weeks. It is also safe to assume that the project manager will assist the team 2 to 3 hours a week, so we will assume 36 hours for the entire 14 weeks. Then, the project management budget is planned as follows:

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Time (hours)</th>
<th>Wage (€/hour)</th>
<th>Team size</th>
<th>Activity Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weekly meetings</td>
<td>14 hours</td>
<td>33,33 €/hour</td>
<td>1</td>
<td>466,62 €</td>
</tr>
<tr>
<td>Team assistance</td>
<td>36 hours</td>
<td>33,33 €/hour</td>
<td>1</td>
<td>1199,88 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>50 hours</strong></td>
<td></td>
<td></td>
<td><strong>1666,50 €</strong></td>
</tr>
</tbody>
</table>

To summarize, the final Wages Budget is set as follows:

<table>
<thead>
<tr>
<th>Team name</th>
<th>Time (hours)</th>
<th>Wage (€/hour)</th>
<th>Team size</th>
<th>Activity Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development team</td>
<td>505 hours</td>
<td>7,5 €/hour</td>
<td>1</td>
<td>6337,5 €</td>
</tr>
<tr>
<td>Project management</td>
<td>50 hours</td>
<td>33,33 €/hour</td>
<td>1</td>
<td>1666,50 €</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>550 hours</strong></td>
<td></td>
<td></td>
<td><strong>8.004 €</strong></td>
</tr>
</tbody>
</table>

2.8.2.2 Material budget

Now we will enumerate the material cost of the resources needed. We will first calculate the cost of the AmazonWS (Amazon Web Services) instances and then we will enumerate the remaining elements.

We will be operating on the UE Ireland region, and we will be using 2 Linux instances. The computation cost is not paramount to the project, so we will be acquiring 2 T1 Micro
instances. These instances are prized at 0.020$/hour. We will have them running for 58 days without interrumption, so its total cost will be:

$$58 \text{ days} \cdot 24\text{h/day} \cdot 0.020\$/h \cdot 2 \text{ instances} = 55.68 \$ = 40.57 \€$$

We will now proceed to enumerate the resources needed for the project, as stated on the Project Planning. Besides, we will be stating the cost of each of them to get a final material budget.

<table>
<thead>
<tr>
<th>Material budget</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware costs</strong></td>
</tr>
<tr>
<td>Workplace desktop computer</td>
</tr>
<tr>
<td>Personal desktop computer</td>
</tr>
<tr>
<td><strong>Sub-total</strong></td>
</tr>
<tr>
<td><strong>Software costs</strong></td>
</tr>
<tr>
<td>Windows 7 Home Premium [31]</td>
</tr>
<tr>
<td>Wijmo Professional License [32]</td>
</tr>
<tr>
<td>AmazonWS instances [33]</td>
</tr>
<tr>
<td>VMWare Player + VMWare Tools</td>
</tr>
<tr>
<td>Linux Mint 15 Olivia</td>
</tr>
<tr>
<td>PyCharm</td>
</tr>
<tr>
<td>Sublime Text 2</td>
</tr>
<tr>
<td>Django</td>
</tr>
<tr>
<td>Angular</td>
</tr>
<tr>
<td>Selenium</td>
</tr>
<tr>
<td>Jasmine</td>
</tr>
<tr>
<td>Docker</td>
</tr>
<tr>
<td><strong>Sub-total</strong></td>
</tr>
<tr>
<td><strong>Total</strong></td>
</tr>
</tbody>
</table>

These are material costs and their depreciation has to be calculated. Nowadays the hardware equipment timelife is set to 4 years, and the software equipment is set to 3 years.

The project hardware’s depreciation is calculated as follows:

$$1.800 \€ / 4 \text{ years} = 450 \€/\text{year}$$

And the software depreciation is:

$$568,15 \€ / 3 \text{ years} = 189,38 \€/\text{year}$$
Next we will calculate this project’s year fraction. A year has 52 weeks. So, it is calculated as:

\[
14 \text{ weeks} \div 52 \text{ weeks} = 0,269
\]

Finally, the total depreciation for the stated material is:

\[
(\text{Hardware depreciation} + \text{Software depreciation}) \times \text{year fraction} = \text{depreciation cost}
\]

\[
(450 \text{ €} + 189,38 \text{ €}) \times 0,269 = 172,14 \text{ €}
\]

2.8.2.3 Indirect costs

The indirect costs have to be addressed to establish the project budget. At inLab FIB, the projects are charged with an extra 35% to reflect inLab FIB’s and UPC’s indirect costs. Then, we have to apply an extra 35% on the direct costs:

\[
(\text{Wages budget} + \text{Material budget}) \times 1,35 = \text{Budget with indirect costs}
\]

\[
(8,004 \text{ €} + 172,14 \text{ €}) \times 1,35 = 11,037,79 \text{ €}
\]

2.8.2.4 Total budget

Finally, we have to apply the value added tax (VAT, called IVA at Spain) to reflect the project final cost. The IVA is actually set to 21%, so the calculations are as follows:

\[
\text{Project budget} \times \text{VAT} = \text{Project final budget}
\]

\[
11,202,29 \times 1,21 = 13,355,73 \text{ €}
\]

2.8.3 Budget control

The budget of this project is tightly related to the time dedicated to it. The budget would be increased if an eventual restructuration of the project planning was needed to add extra hours.

To avoid budget increases, the team will do strictly what was planned to do. It has been agreed that if the client asks for extra features, some of the previously planned ones will be discarded. It is up to the client to decide which of the functionalities are the most important to the success of the project.
2.9 Economic feasibility

This is a research and development project, and because of that its economic benefits are difficult to quantify. However, it is strongly oriented into optimizing the developers time. The amount of developers at the Telefónica group is very high. So, the potential impact on these developers’ productivity is very high as well.

The average time spent to remotely deploy an application varies wildly. However it is safe to assume that it ranges from 10 minutes to 1 hour, depending on the complexity of the application. This time can be even longer if dependency errors arise, potentially hindering the developers’ daytime productivity. Docker would negate those errors, potentially saving entire deployment days.

That said, the average deployment without Docker would take approximately 30 minutes. With Docker, it is expected to lower that time to around 3 minutes, so:

\[
\frac{\text{Previous time} - \text{Actual time}}{\text{Previous time}} = \% \text{ Decrease}
\]

\[
\frac{30 - 3}{30} = 0.9 = 90\% \text{ decrease}
\]

As showed, Docker would decrease the deployments time by 90%. This would save a lot of time and effort for TID. This would benefit even more departments where there is an environment with continuous integration (CI), where deployments are performed several times every day.

Let’s consider there are 600 deployments at TID every year, and that the average developer wage is 30€/hour. Each deployment would cost 15€ (30 minutes), for a total of 15€ \cdot 600 = 9000 €. With the Docker project improvements, it would cost:

\[
9000 \, \text{€} \cdot (1 - 0.9) = 900 \, \text{€}
\]

\[
\text{Saving} = 9000 - 900 = 8.100 \, \text{€}
\]

Thus, this project’s investment would be returned in 1 year and 7 months approximately.

On the other side, the technological advantage and expertise acquired with the LiveScanner project could easily be very beneficial to some of the next web-related enterprise projects.

2.10 Sustainability and social responsibility

This section will relate to the project’s environmental implications and the social benefits it will provide.

As shown previously, the most part of the resources used for the development of this project are human. The computational and material costs are very limited, and so we can consider that the environmental impact is close to none. These costs already exist and we are not adding any extra costs.

Regarding the social impact, this project may prove useful to many people. All of these projects’ code will be open-sourced, and so anyone will be able to download, study or modify
them at will. The creation of new documentation and open-source projects can be seen as a contribution to these technologies’ community, which should be regarded as a very positive fact.

2.11 Laws and regulation

This section will involve the project’s relation with the actual legislations. This project is ruled by the Spanish laws and thus it must be compliant with its laws. As it is an information-related project, it has to be compliant with the LOPD (Ley orgánica de protección de datos) which is the Spanish law for individual information security.

However, this project does not store any kind of user-related information, and thus it compliant with the LOPD.

About confidentiality: all the members of the uLab team have signed confidentiality agreements for this project. However, all of the content done for this project is open-sourced, and thus all of the related documentation contained in this project can be shared without any kind of trouble.
3  Requirements specification

3.2  Use case diagram

We will now introduce the Live Scanner Use Case diagram. It will give us a higher-level approach on the functionalities of the application, as well as a quick view at the stakeholders involved.

3.3  Functional requirements

We will now define the functional requirements for the Live Scanner project. As the methodology that we are using is SCRUMBAN, the requirements are defined using User stories.
A User story is the agile definition of a functional requirement: it describes a situation where a stakeholder wants to be able to do something in order to accomplish a particular goal. The main difference with traditional requirement definition lies in its mutability: user stories are supposed - and meant - to change in order to reflect a project’s evolving nature.

The following requirements have been defined and modified as the project evolved, and this is the final user stories list.

### 3.3.1 User stories

For the user stories definitions we will use the following scheme:

<table>
<thead>
<tr>
<th>#.</th>
<th>User story name</th>
<th>As a stakeholder</th>
<th>I want to do something</th>
<th>In order to get a result</th>
<th>Priority: 1-10 (min-max)</th>
<th>Acceptance criteria:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Login</td>
<td>As a show administrator</td>
<td>I want to be able to login into the system</td>
<td>In order to manage the shows.</td>
<td>7</td>
<td>The system lets the user log in only if the provided credentials are right.</td>
</tr>
<tr>
<td>2.</td>
<td>Retrieve shows</td>
<td>As the Live Scanner system</td>
<td>I want to query the provisioning endpoint about the shows that are live right now</td>
<td>In order to list the most relevant information of the shows.</td>
<td>10</td>
<td>The shows that are listed as live are the same that are provided when access the provisioning endpoint. For each show, the information provided is : Name, percentage, and logo image Any other information is not shown by default</td>
</tr>
</tbody>
</table>
### 3. Detail show

**As a show administrator**

I **want to** be able to see the show description on demand

In order to understand the show and make a better decision.

**Priority:** 6

**Acceptance criteria:**

- When instructed to do so, the system shows the selected show description.

### 4. Search shows

**As a show administrator**

I **want to** be able to filter the show list

In order to find shows quickly.

**Priority:** 6

**Acceptance criteria:**

- When provided with search parameters, the system filters the show list and displays only the shows whose name matches with those parameters.

### 5. Choose show

**As a show administrator**

I **want to** be able to choose a show and rate it

In order to mark it for provisioning.

**Priority:** 9

**Acceptance criteria:**

- When instructed to do so, the system marks the selected show.
- Then, the show administrator is able to select its rating from a fixed list.

### 6. Send shows

**As a show administrator**

I **want to** be able to send the chosen shows and its ratings to the rating endpoint

In order to notify about the best shows and its reasoning.

**Priority:** 10

**Acceptance criteria:**

- The rating endpoint confirms that the suggestions have been received.
3.3.2 Data model

The following is the specification for Live Scanner's data model. The application will receive its data as a JSON object, and thus this is only an abstraction of the information that will be provided to us. That is the reason why there are very little restrictions upon it.

```
<table>
<thead>
<tr>
<th>Channel</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>start: Integer</td>
</tr>
<tr>
<td>abbrev: String</td>
<td>end: Integer</td>
</tr>
<tr>
<td>logo: String</td>
<td>duration: Integer</td>
</tr>
<tr>
<td>title: String</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Show</th>
<th>Content</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>id: Integer</td>
<td>slug: String</td>
<td>id: Integer</td>
</tr>
<tr>
<td>title: String</td>
<td>type: String</td>
<td>unit: String</td>
</tr>
<tr>
<td>description: String</td>
<td>genre: String</td>
<td>width: Integer</td>
</tr>
</tbody>
</table>
```

3.3.3 Behaviour model

We will now state the specification for the previously introduced user stories. Each one of them will briefly describe the interactions between the involved stakeholders, and will establish a base for the application behaviour that will be enhanced later at the architecture section.

```
User login(username, password)

System detailShow(showId)
```
3.4 Non-functional requirements

The previous functional requirements are the ones related to the features of the project. However, there are some other non-feature related requirements that are expected to be met: these are the non-functional requirements.

These requirements are as mutable as the functional ones, and they may change from a sprint to another; that is why we only want to briefly detail them. That is the reason why we have decided that a Volere scheme would not be worth the effort. Instead, we will be using a simple scheme describing what they aim to accomplish.

The following requirements have been defined and modified as the project evolved, and this is the final non-functional requirements list.

We will use the following scheme:

<table>
<thead>
<tr>
<th>#</th>
<th>Non-functional requirement identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Requirement name</td>
</tr>
<tr>
<td>Description</td>
<td>Requirement description</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td>---</td>
<td>----------------------</td>
</tr>
<tr>
<td>1</td>
<td>Usable interface</td>
</tr>
<tr>
<td>2</td>
<td>Portuguese-Brazilian</td>
</tr>
<tr>
<td>3</td>
<td>Quick usage</td>
</tr>
<tr>
<td>4</td>
<td>Responsive filtering</td>
</tr>
<tr>
<td>5</td>
<td>Test scenario</td>
</tr>
<tr>
<td>Name</td>
<td>Usable interface</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The color palette used for the web application has to be comfortable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Use AngularJS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The AngularJS web framework must be used to implement the search filtering.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Unit tests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>The application internal behaviour must be covered with unit tests.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>End-to-end test</th>
</tr>
</thead>
</table>
| **Description** | The application global behaviour must be tested with an end-to-end test. This test will cover a standard user execution, which involves:  
- Login  
- Detail show  
- Choose show  
- Send shows |
4 Architectural design

We will now proceed to explain the design patterns chosen for the Live Scanner Project and then we will detail the behaviour for each of the three layers of the web application.

4.2 Model-view-controller

We will be using Django and AngularJS as our Web frameworks. This gives us the chance to use the MVC framework, which will make development easier and cleaner.

The MVC framework (Model-view-controller) is a software pattern for implementing user interfaces. It states that a software application should be divided into three parts in order to better differentiate responsibilities between them. These components are the following:

What are they?

- Model
  - Consists on application data and business logic
- View
  - Any representation of the application’s information
- Controller
  - Collects input and redirects it into orders to the model or view

How do they interact?

- Controller
  - The controller is responsible of sending messages to the model in order to update its state. It can also send messages to the view to change the view’s presentation of the model’s information
- Model
  - The model is responsible of notifying its associated views and controllers when the information it contains has been changed.
- View
  - The view is responsible of the application’s output information, usually by requesting/receiving it from the model
The following is an interaction overview for the MVC framework:

However, many specific Web frameworks use their own interpretation of the MVC. In our particular situation, Django and Angular also have one their own one. We will be using the MVC interpretation stated by AngularJS, which is defined by themselves as follows:

“For several years +AngularJS was closer to MVC (...), but over time and thanks to many refactorings and api improvements, it's now closer to MVVM – the $scope object could be considered the ViewModel that is being decorated by a function that we call a Controller.

Being able to categorize a framework and put it into one of the MV* buckets has some advantages. It can help developers get more comfortable with its apis by making it easier to create a mental model that represents the application that is being built with the framework.

Having said, I'd rather see developers build kick-ass apps that are well-designed and follow separation of concerns, than see them waste time arguing about MV* nonsense. And for this reason, I hereby declare AngularJS to be **MVW framework** - Model-View-Whatever. Where Whatever stands for “whatever works for you”.”

AngularJS on Google+, 19/7/2012 [34]
As “anarchic” as this definition may look, the AngularJS’s MVC approach is very pragmatic and useful and it keeps most of the concern separation provided by the original MVC. While doing so, it is able to separate presentation from business logic in a convenient and flexible way.
4.3 REST

REST (Representational State Transfer) is a software technique for distributed hypermedia systems such as the WWW. REST can be used to create, read, update or delete information on a server using simple HTTP calls. This term is used to describe any simple web interface using XML/JSON and HTTP, as opposed to other more complex message exchange based protocols such as SOAP.

The following are its main properties:

- **Stateless**: each HTTP message has all the information needed to execute the request. It does not rely on previous requests.
- **Resource management methods**: it uses the standard HTTP methods GET, POST, PUT and DELETE in order to obtain, modify, add and delete information.
- **Universal resource syntax**: each one of the server’s resources is associated with its URI.
- **Collection or Element URIs**: resources can be offered as collections or as single elements

Many of the main service providers use and offer REST APIs, such as Twitter, Flickr, Amazon, etc.

Below there is a piece of the description of the resource used to obtain a single tweet from the Twitter REST API. As we can see, in this particular case this is an Element URI, because it is addressing to a single element. It offers all kinds of useful information, such as the response format, the method used, its URI and its mandatory and optional parameters.

**GET statuses/show/:id**

<table>
<thead>
<tr>
<th>Resource Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate Limited?</td>
</tr>
<tr>
<td>requests per rate limit</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Response Formats</td>
</tr>
<tr>
<td>HTTP Methods</td>
</tr>
<tr>
<td>Resource family</td>
</tr>
<tr>
<td>Response Object</td>
</tr>
<tr>
<td>API Version</td>
</tr>
</tbody>
</table>

4.3.1 Presentation Layer
4.3.1.1 External design

The Live Scanner project is a prototype and it does not need a professional level interface design just yet; that is why the development team designed its external design and interactions.
4.3.1.2 Navigational map

Live Scanner is mostly composed by a one-page web application, the shows page, where almost all of the interactions happen. In addition to this page, there is the login site.

We will first show the login navigational map and then the one-page site maps.

Login
Select shows
Modify selected shows
Deselect show

Search shows
**Send shows**

<table>
<thead>
<tr>
<th>Shows listed</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show 1</td>
<td>Data 1</td>
</tr>
<tr>
<td>Show 2</td>
<td>Data 2</td>
</tr>
<tr>
<td>Show 3</td>
<td>Data 3</td>
</tr>
<tr>
<td>Show 4</td>
<td>Data 4</td>
</tr>
</tbody>
</table>

![Diagram showing various steps related to sending shows](image-url)
4.3.2 Domain Layer

4.3.2.1 Data model

The following diagram illustrates the final Data model that we will be using. We are going to receive a lot of data from the provisioning endpoint, however, not all of it is going to be relevant for this application. Thus, this data model illustrates the information and properties that we will be using.

4.3.2.2 Behaviour model

Previously in this document, we stated the Behaviour model of the Live Scanner application on a specification level. Now we are going to dive onto it and detail it much further, reflecting the work done at the design stage.

Keeping the same structure as we previously had, we are going to introduce the extended behaviour diagrams corresponding to each of the use cases.
Login

Retrieve shows

Service::/

httpGETRequest()

retrieveShows()

Service::/

<ProvisioningEndpoint>
Service::/programs/

retrieveShowsData()

httpGETRequest()

(data, status)
Service::/

retrieveShows()

retrieveShowsData() : <showsData, status>

processShowsData(showsData, status) : showsInfo

:UnMarkedListCtrl

renderShows(result)

Service::/

processShowsData(showsData, status)

opt
[status == success] (data)

shows:showsArray

sortByTitle(shows)

result := shows
Detail shows

```java
:GenericListCtrl

detailShow(showId)

getSelectedShows() : selectedShows

```
Search shows

Choose show
Send shows

This web application does not store any kind of data, and thus it does not need a data layer. The data about the shows must be updated every time the web page is refreshed, so storing the retrieved data is not useful.

Storing the shows data could be useful in order to implement a caching system that would avoid GET petitions when there are no changes on the information. However, to do so we would need a server method to ask if there have been changes since the particular user's last request, but this kind of method has not been provided to us right now.
4.4 Deployment model

The following deployment model shows the system modules interaction in a production environment.

As it has been previously stated, the Live Scanner project uses two external resources to query and provide information about the shows. In order to better reflect the project's environment, these two services have been added to the diagram even if they have not been developed or deployed by the team.
5 Development

In this section we will explain the most relevant features provided by the frameworks that we are evaluating. All of the following have been used in Live Scanner in order to improve the application and, most importantly, have a solid opinion based on practical use.

5.2 AngularJS

Besides the features that we have already talked about like being an MVC framework, AngularJS provides us with several useful features. The most relevant for us have been the following:

- **Directives**

In AngularJS, directives are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS's HTML compiler to attach a specified behavior to that DOM element or even transform the DOM element and its children. For example, static HTML does not know how to create a list of elements from an array, or conditionally show an element based on a variable. These two features, for instance, are accomplished by the directives `ng-repeat` and `ng-model`, respectively, and there are many more of them.

- **Two way data-binding**

Data binding is probably the most useful feature in AngularJS. Data binding consists on linking DOM's HTML variables to your dynamic model. This way when either of them changes, the other one does too. This feature alone enables the developer to save up a lot of time because a lot of code that would be needed otherwise is no longer needed (such as DOM listeners and data manipulators).

This is accomplished by using the directive `ng-model`. The following example will illustrate this feature further:

```html
<!doctype html>
<html ng-app>
<head>
  <script src="http://code.angularjs.org/angular-1.0.0rc10.min.js"></script>
</head>
<body>
  
  <label>Name:</label>
  <input type="text" ng-model="yourName" placeholder="Enter a name here">
  <hr>
  <h1>Hello, {{yourName}}!</h1>
</body>
</html>
```

In this example, there would be a text with the word "Name", an input textbox and a greeting, which would be just "Hello" at the start. When the users would type on the textbox, the `ng-`
model directive would automatically update the variable "yourName", which is the same one being used at the greeting below, because the variable would be binded.

- **Dependency injection**

Dependency injection is a software design pattern based on the *dependency injection principle* [35] by Martin Fowler. The goal is to implement inversion of control, in a way that the dependency (service) is injected into the object (client), rather than allowing the client to build or find the service itself.

This design pattern has many uses, and Angular implements it in a convenient way. It is especially useful on decoupling modules and allowing concurrent development in teams. It can also be a great asset for testing, as we will see next.

- **Easy testing**

Angular provides a series of utilities to help with application testing. Dependency injection is very helpful, but Angular offers even more with some specific modules such as $httpBackend. This module helps with the testing of features involving HTTP calls, and it has been very useful for this particular application. This will be explained further in the Testing section of this document.

### 5.3 WIJMO

As we have said earlier, WIJMO is an UI widget kit. It has been designed to save up developers’ and designers’ time while offering very effective visual UI.

As previously stated, we at the development team parallelized our efforts while creating Live Scanner, and the writer of this document was responsible of the server and testing but not of the visual interface. That is why this section will not involve any insight on the WIJMO features besides the ones that have already been presented.

We will, however, offer a complete evaluation of WIJMO at the end of this document, based on the other team’s experience with its usage.

### 6 Testing

#### 6.2 Jasmine

Jasmine is a behaviour-driven testing framework designed to test JavaScript code. Its biggest features are its clean syntax and its effective output webpage. They are both very easy to use and understand and they do what we wanted them to do: organize and label tests in a comprehensive way.
Jasmine is mostly based on expectations: we use the piece of code that we want to test and then we set the expected out. That expected output syntax is designed to be close to human language. Let’s see an example.

```javascript
describe("filterByPattern", function () {
    var showsArray;
    beforeEach(function () {
        showsArray = new ShowsArray(
            [{id: 1234, title: 'element1', channel: {logo: null}, description: ''}, // 0
             {id: 2848, title: 'element2', channel: {logo: null}, description: ''}, // 1
             {id: 3800, title: 'ELEMENT3', channel: {logo: null}, description: ''}, // 2
             {id: 4080, title: 'Element4', channel: {logo: null}, description: ''}, // 3
             {id: 5980, title: 'E', channel: {logo: null}, description: ''}, // 4
             {id: 5648, title: 'aaaaaaaa', channel: {logo: null}, description: ''}, // 5
             {id: 6456, title: 'xxxxxxxx', channel: {logo: null}, description: ''}, // 6
        ]);
    });
    it("Filtering an empty ShowsArray it should return 0 elements ShowsArray", function () {
        arrayTest = new ShowsArray([]);
        expect(arrayTest.filterByPattern(filterByPropertyId, 1234)).toEqual([]);
        expect(arrayTest.filterByPattern(filterByPropertyId, 1234).length).not.toBeGreaterThan(0);
    });
    it("When only one show is filtered it should return only 1 element ShowsArray", function () {
        expect(showsArray.filterByPattern(filterByPropertyId, 5800)[0]).toEqual(showsArray[4]);
        expect(showsArray.filterByPattern(filterByPropertyId, 5800).length).toBe(1);
    });
    it("When all the shows are filtered it should return 0 elements ShowsArray", function () {
        expect(showsArray.filterByPattern(alwaysFalse, 6456)).toEqual([]);
        expect(showsArray.filterByPattern(alwaysFalse, 6456).length).not.toBeGreaterThan(0);
    });
});
```

In this test, we want to test Live Scanner’s filter feature. As we can see, we state the Array elements that are going to be filtered. Then, we start stating each unit test using the `it` function, and stating our expectations with `expect` and its related matchers, such as `toEqual` or `not.toBeGreaterThan`.

When we are done stating our tests and its expected outcome, we just have to tell Jasmine where the files are and open its SpecRunner on a web browser, giving us the output below.
If some of the tests failed, Jasmine would warn us and would tell us specifically which ones did and why, as we can see in the following screenshot.

![Screenshot of Jasmine test results showing a failed test](image)

During the realization of the project, we gradually covered the application with unit tests. We have manually tested the features ourselves, but this way, each feature introduced had its own test that certified it worked.

The final test suite comprises a total of 25 unit tests that the application behaviour.
When we introduced the Jasmine testing examples, we used a trivial case that filtered an array of values and checked the output was correct. However, the core of our application lies on the HTTP communication between Live Scanner and the described endpoints.

We could state tests that checked these communications by requesting values themselves, but that would be suboptimal for a number of reasons. On one hand, we would be consuming external services on a test, which is wrong, because of unnecessary server load and financial reasons – this might seem negligible, but it would not be if these tests got automated at some point. On the other hand, we could not have predicted which data the endpoint would return, so we could not have fully checked that the response was entirely correct – neither even ensure that the external endpoints would be operative. One solution to this problem would have been to request a concrete service to the endpoint that always returned the same data, but this would still mean unnecessary communications, and would not solve our external endpoint dependency.

6.3 AngularJS

As we anticipated earlier in this document, AngularJS itself provides a series of utilities for testing applications.

When we introduced the Jasmine testing examples, we used a trivial case that filtered an array of values and checked the output was correct. However, the core of our application lies on the HTTP communication between Live Scanner and the described endpoints.

We could state tests that checked these communications by requesting values themselves, but that would be suboptimal for a number of reasons. On one hand, we would be consuming external services on a test, which is wrong, because of unnecessary server load and financial reasons – this might seem negligible, but it would not be if these tests got automated at some point. On the other hand, we could not have predicted which data the endpoint would return, so we could not have fully checked that the response was entirely correct – neither even ensure that the external endpoints would be operative. One solution to this problem would have been to request a concrete service to the endpoint that always returned the same data, but this would still mean unnecessary communications, and would not solve our external endpoint dependency.
Instead, we used AngularJS’s **mocking system**, more specifically the **$httpBackend** service. A mock is an object that simulates the behaviour of a real object, but on a very basic level, in order to isolate the behaviour of the object you want to test.

The $httpBackend service is a mock, and it is meant to override the regular $http service, which is the one responsible of making the HTTP GET and POST calls. $httpBackend provides a series of utilities that let us **intercept HTTP calls and give a static response as if the external service responded**.

The $http service overriding is simple thanks to the use of dependency injection, and we can now fully test the service’s response because it is returning static data, **whether the external endpoint is available or not**.

```javascript
describe('When attempting fake HTTP connections for getting the source API content', function() {
  beforeEach(inject(function($injector) {
    $httpBackend.whenGET('/programs').respond(getData());
  }));

  it('should perform GET http call to /programs and retrieve a correct result', function () {
    $httpBackend.expectGET('/programs');
    var controller = createController();
    $httpBackend.flush();
    expect($rootScope.data.showsPropertiesOfInterest[0].title).toEqual("Encontro com Deus na Madrugada");
  });
```

The example above summarizes $httpBackend’s features, expecting a concrete response from the HTTP call responsible of requesting the shows.
7 Technological study

7.2 Introduction

This section will detail everything we have learnt about Docker and its context for the duration of this project. First we will explain about Docker’s motivation, objectives and architecture, covering his features and particularities. Once this is completely explained, we will switch to more pragmatic, workshop-oriented explanations, where we will be showing how Docker can be used and what has been achieved during this project.

Several years ago, server deployments were straight-forward: there was “the” server, a framework, and there were not many dependency problems, and if they happened, they were quickly identifiable. However, nowadays software deployment is getting more and more complex: we have a software stack that is much more distributed and complex, with different programming languages and frameworks, as well as many different infrastructures where the software components are being run.

This introduces a series of serious issues, which may not seem relevant at first but they certainly scale up very quickly. The main issue is that someone, namely the system administrators, have to make sure that every software component runs on every infrastructure. For example, let’s imagine an application that needs a particular Python version, the X.1 version, and you want to deploy that application on an infrastructure which has the Python version X.2 installed. Several options arise: you could check if the X.1 and X.2 versions are compatible, you could try to upgrade your application to use the X.2 version (which may not work at first), or you could downgrade the infrastructure installation back to
X.1, which you probably cannot do because there probably are other applications running that depend on it.

This issue could be solved rather quickly on a small environment, with a couple development stations and a couple servers. However, it scales up very quickly: suddenly you have to manage hundreds of dependencies (not only the Python X.2 issue) and hundreds of dedicated environments – Database servers, Web servers, Background workers, API endpoints – each one with their own operating systems and their own dependencies. We would then be facing an almost exponential dependency hell problem.

|---------------|---|---|---|---|---|---|---|---|

At the Docker Introduction video made by its founder, Solomon Hykes [36], he makes an excellent comparison between the issues we have stated now and the transport industry. He explains that they have had very similar problems in the past: shipping a product demanded specific requirements, such as size, form, resistance to hits... In essence, they had the exact same “dependency hell” problem, which led to a very expensive and unreliable shipping process. Finally, they solved it: they agreed on a box, with established sizes, their weight, how the doors worked, where the labels where, etc.
Docker wants to resolve this software problem in a very similar way. They have defined a standard format that is simple and reliable enough that a critical mass of people can agree to use and integrate with it. In short, Docker offers a standard way to pack software into a box, a box that you can handle to system administrators, operation teams, infrastructure providers, etc. and they will know how to handle that box. Then, ultimately, the developers are free to think only about what’s inside the box, and the infrastructure team can take care about what is outside the box. This solves the problems stated about, making software deployments simple, repeatable, reliable and ultimately cheaper in resources and time.
7.3 Docker architecture overview

Now that we have introduced Docker’s functionalities, we will proceed to briefly look at how it works.

Docker, unlike Virtual Machines, does not run a Guest OS onto a running Host OS; it eliminates the need of Host OS by using Linux’s Containers (LXC). However that introduces the downside that it can only run on Linux (kernel version 3.8 or above), but in exchange it is a lot more efficient in resources management.
Docker uses a client-server architecture. The Docker client communicates with the Docker Daemon, which is responsible for building, running and distributing the Docker containers.

The Docker Client and the Daemon can run on the same system, or they can run on separate systems and communicate via sockets or through a REST API.

Docker also offers a public image repository (called Docker Hub) where anyone can share their own images, and where you can find several images with pre-built installations made by official distributors like Ubuntu, MySQL or NodeJS.
7.4 AWS configuration

We will be working with a remote Docker installation hosted on one of Amazon’s services, EC2. As we will need very little computing power, we chose the Micro type of instance, which is the lowest profile available, and a volume of 32GB.

As of operating system, we have chosen the Ubuntu 12.04 image provided by Amazon.

As we will be using external services and we will need communication, we will define a new “Security group”. This is a feature offered by Amazon that enables us to create a set of rules for connections, let it be TCP, SSH, HTTP, ICMP… and the available ports.

The following are the established rules in order to let us connect to the remote installation and enable it to accept the requests we need.
Amazon offers an extremely wide array of services to its clients, but we will not need most of them. We will only be using the EC2 and S3 instances, the connection security options and a private key for secure remote connections.

7.5 Docker images and containers

**Image**

A Docker image is a read-only template that contains an operating system with whatever software you wanted to install on it. For example, an image could contain an Ubuntu OS with Python and an application installed on it.

Docker lets you create and update images quickly, or download them from their public repository.

Once created, images are used to create Docker containers.

**Container**

Docker containers hold everything needed for an application to run (assuming all the application’s dependencies are correctly provided). They are created from images, and can be run, stopped, moved and deleted.

Each container is an isolated and secure application platform. It is possible to have many running containers, each one with a different OS and a different set of dependencies installed, and each one with its own set of port redirections.
### 7.6 Dockerfile configuration

Now that we know what a container is, let’s look at a practical example. The following screenshot shows that we are logging into our Ubuntu Amazon instance, and we have displayed what images are available. We have previously downloaded some of them, but right now we will be using the latest Ubuntu one.

With the command shown at the following screenshot:

```bash
> sudo docker run -i -t ubuntu /bin/bash
```

We will be requesting several things:

- First of all, if the Ubuntu image does not exist in the system where our Docker instance is installed, Docker will try to fetch and install it from the public repository
- Then, once the image is found or downloaded, Docker will create a container out of that image
- Then, Docker will automatically run that container, with an extra option: we are telling him that we want to run this container in interactive mode (-i).

As we can see, our user has not changed (we are still root) but the machine where we are connected has changed. As we can see, it’s the Ubuntu Precise (12.04) OS that we expected.
Now we could start installing our dependencies in order to run our application and it would work perfectly; however, Docker offers a more civilized way of creating our images. Instead of manually installing each dependency, Docker can use a **Dockerfile**.

A Dockerfile is a script that provides Docker with everything it needs to create a customized image that will be able to run our application instantly. Dockerfiles use some specific commands like **FROM** and **RUN** to tell Docker which base image has to be used to create the custom one, and which commands should be run to install all the needed dependencies.

Below we can see the Live Scanner Dockerfile:

```
# use the ubuntu latest image provided by dotcloud
FROM ubuntu:precise
MAINTAINER inLabFIB inlab-ulab@fib.upc.edu
# make sure the package repository is up to date
RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe" > /etc/apt/sources.list
RUN apt-get update
# install the app requirements
# install python
RUN apt-get install -y build-essential
RUN apt-get install -y libreadline-gplv2-dev libncurses5-dev libssl1-dev libsqlite3-dev tk-dev libgd-dev libc6-dev libbz2-dev
RUN apt-get install -y python-software-properties
RUN add-apt-repository ppa:krull/deadsnakes
RUN apt-get update
RUN apt-get install python2.7
RUN apt-get install -y python-pip
# install django 1.5.2
RUN pip install django==1.5.2
# install git & download project
RUN apt-get install -y git-core
RUN git clone https://github.com/PDF-OS-Protolab/live_scanner
# setup database and start app
ENTRYPOINT cd live_scanner & & python manage.py syncdb -noinput & & echo $VARIABLE & & python manage.py runserver 0.0.0.0:8000
```

Dockerfiles can be created so that the containers run a set of commands when they are started. This is done by the command **ENTRYPOINT**, and lets developers automatize even more the container execution; everything that the application needs to run is encapsulated here. This way, the container is autonomous – it needs no external input to run, because it is already running since it started.
7.7 Live Scanner deployment

Now that we have seen the Live Scanner’s Dockerfile, we are going to create an image from it. This is done thanks to Docker’s “build” command, as shown in the screenshot below:

We can see that Docker has downloaded the Ubuntu image we told him, and has started running commands onto it, downloading Live Scanner’s dependencies.

Once it has finished, if we look at our Docker installation available images, we will see the newly created Live Scanner image. Then we will proceed to run it, as shown in the screenshot below.

It should be noted that we have used extra parameters in this run command. We are telling Docker to redirect all the traffic from the container’s 8000 port to the host machine’s 8000 port. Live Scanner traffic travels through this port, and we are simply connecting both
masks machines ports. Keep in mind that this feature is extremely useful because it allows applications to use whatever port they choose, because the system administrator that is running the Docker will be able to manage which ports are used in the end. This feature makes deployments easier for both developers and system administrators.

It is all set now, and we can use our browser to request the Live Scanner web page to our Amazon instance’s, using its public IP and the port that we specified, which was 8000.

We highly encourage watching the videos made by the development team, specifically the 1st video. [37]

7.8 Docker Private Registry

As stated before, Docker has its own image repository called Docker Hub. It is a public repository where people around the world share their customized images with each other.

However, there are certain times where organizations want to share their images just within their workers. Docker has considered this and offers what is called a private registry; an open source image repository that can be used to store and retrieve images in a private way.

In this project, we have created and tested a private registry. We have also used an Amazon’s S3 bucket, which is a cloud resource for storing data that scales dynamically. Note that these kinds of resources are not needed to run a Docker private registry, but it was considered a very good option that should be tested.
We highly encourage watching the videos made by the development team, specifically the 2nd video. [37] All the information regarding commands and pushing times is explained there.

We found private registries to work perfectly and are easy to set up. As shown in the video, we managed to push and pull Docker images from different Amazon Ubuntu instances without any trouble. The upload times of course rely on the connections available, but Docker’s efficient management of resources contributes to lower these times – our Live Scanner image’s size, with the Ubuntu OS and all of its dependencies, is 476MB.

7.9 Auto-deployment tool

Now we have experienced how Docker works, which typically is:

- We create a Dockerfile
- We open a connection with our remote Docker installation
- We build the image
- We run the image on a container

This is much easier than what we had to do previously to deploy an application, and has all of the benefits of Docker, which we have already stated.

However, it would be excellent for the developer to focus completely on developing, without having to manage any of the deployment process. As downloading and deploying a project is a repeatable and automatable task, we have created a utility script that assists greatly in this process.

This script has been called Auto-deployment tool. Its use is very simple, because the developer only has to fill the required parameters for Docker to work, and the script will deploy and run the application on our host machine, the remote machine where Docker is installed.

In the following screenshots we can see a complete download and deployment process. First, we download the script from GitHub, and we check that the base dummy parameters:
Then we proceed to input our particular application parameters, which are:

- **GitHub URL** – our application’s public GitHub repository
- **Branch** – the branch that we want to download from the previous repository
- **Container Tag** – the tag we want to give to our container
- **Local Port** – the port from the host machine where our app will be exposed
- **Container Port** – the port from the container that will be linked to the host
- **SSH Address** – host machine’s user and IP address
- **SSH Credentials** – path to our SSH security credentials
- **Env. vars.** – list of our application’s custom environment variables

The following screenshot shows an example configuration for these parameters. As we can see, we will be using the script to deploy our own Live Scanner application to the same Amazon instance that we used before. Once we have finished introducing these parameters, the only thing we need to do is to execute the script.
Once the script has finished installing and running the application, it will show us the new container’s identifier. The script prints out a lot of information about the process. That means that if the execution failed at some stage, the script would tell us enough information for us to know what went wrong and how to solve it.

In the following screenshot we can see that the script execution finished successfully, and on the terminal above we can see the new running container with the exact tag we specified: “live_scanner_autodeployed”.

Then again we can check that it all worked exactly as we did earlier. The only difference is the time needed to deploy the application; the Auto-deployment script configuration is faster and requires less effort for the developer, because it automatically downloads the project, it
connects with the host machine, builds and runs the container etc. and it supports the usage of environment variables that our application may need. The only work needed by the developer is the Dockerfile configuration – his application’s dependencies – and the auto-deployment configuration, which are not likely to change in short-term.

We highly encourage watching the videos made by the development team, specifically the 3rd video. [37]

The Auto-deployment script can be found at Telefónica I+D public GitHub repository. [38]
7.10 Actual use cases

Docker is being used actually on several well-known companies to great success. However, not all of them use them to achieve the same goals. Some of these companies are:

- **eBay - continuous integration**
  - "eBay Now" is eBay's same day delivery service. They use Docker in order to reduce deployment times and to implement an efficient and automated path through the development, test and quality assurance (QA) teams.

- **Yandex - Platform-as-a-Service (PaaS)**
  - Yandex is the largest tech company in Russia and the creator of Cocaine, the open-source Platform-as-a-Service for the creation of custom cloud hosting apps - similar to Google App Engine or Heroku. They are using Docker for infrastructure virtualization and app isolation.

- **Spotify - Normalized development environments**
  - Spotify is a music streaming service available on 56 countries around the world, and uses more than 5000 servers in four separate data centers. Before using Docker, they had to establish individual connections with the servers to make application and dependencies updates, what lead to denormalized nodes and inconsistencies. They started to explore the usage of LXC, and finally they ended up relying on Docker.

On a different scale of things, Google has recently announced that they are working on a set of extensions that will allow Google App Engine developers to build and deploy Docker images in Managed VMs [39]. This way, developers can access the large Docker image library (Docker Hub) to deploy containers into a completely managed environment.

Besides, they have also announced that they are working with the Docker community towards improving the open container standards. All of Google's support makes a lot of sense since they are actually using these standards. In their own words: "Everything at Google, from Search to Gmail, is packaged and run in a Linux container. Each week we launch more than 2 billion container instances across our global data centers, and the power of containers has enabled both more reliable services and higher, more-efficient scalability. " [39]

For more detailed information about featured use cases, please refer to the Docker’s official website [40].
8 Evaluation

As we have previously stated, one of the main goals of this project is the evaluation of the technologies that we have been using to develop them.

We think that an evaluation process can be influenced by personal opinions, and while opinions are very valuable, we want to keep the evaluations as useful as possible. In order to keep them valuable and useful, we established a common “Evaluation form”, which focuses on the positive and negative results. Therefore, we are giving a quick way to see what are the benefits and issues related to a concrete technology.

Keep in mind that these are quick “technology reports”. In order to keep evaluations useful, it is that key that they are as quick as possible – that is why they will be kept as a one-page document that will state the most relevant information about them. Given the need of more in-depth information, the interested person could read this document - and all of its references - in order to get a better insight on each of these technologies.

It should also be noticed that WIJMO has been evaluated even when the author of this document was not a member of the team that was designated to work with it. However, the author has access to the information about the work that has been done with it, as well as the evaluation and feedback given by the team that worked with it. The author considers this information of an excellent quality and that is why it has been included and evaluated in this document. The document with all of WIJMO’s issues is available through Google Drive. [41]
### Evaluation form – uLab

<table>
<thead>
<tr>
<th>Technology name</th>
<th>AngularJS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evaluation goal</strong></td>
<td>Test whether or not AngularJS’s features are useful and use it to provide a working prototype for the Live Scanner application</td>
</tr>
<tr>
<td><strong>Initial hypothesis</strong></td>
<td>AngularJS’s features – specially its data binding and dependency injection helpers – are useful</td>
</tr>
<tr>
<td><strong>Brief work summary</strong></td>
<td>There has been made a simple web application called Live Scanner that consists on retrieving, modifying and sending data through REST APIs. The application has been built using controllers, and the mainly used modules have been $http, $scope, $injector and $httpBackend.</td>
</tr>
<tr>
<td><strong>Obtained results</strong></td>
<td></td>
</tr>
<tr>
<td>Positive results</td>
<td>Negative results</td>
</tr>
<tr>
<td>- Heavily module-oriented</td>
<td>- When applications get bigger, modularity is needed to preserve organization</td>
</tr>
<tr>
<td>- Dependency injection ease of use</td>
<td></td>
</tr>
<tr>
<td>- Easy data-binding</td>
<td></td>
</tr>
<tr>
<td>- DOM manipulation helpers</td>
<td></td>
</tr>
<tr>
<td>- Extensible directives</td>
<td></td>
</tr>
<tr>
<td>- Testing helpers</td>
<td></td>
</tr>
<tr>
<td><strong>Hypothesis results</strong></td>
<td>VALIDATED</td>
</tr>
<tr>
<td><strong>Conclusion</strong></td>
<td>AngularJS is a <strong>great web framework</strong> whether you are building small or big applications, and it <strong>scales very well with team size</strong> thanks to its heavy <strong>modularity</strong>. It is extremely useful when building <strong>dynamic web sites</strong> because of its convenient directives. Its <strong>learning curve is smooth</strong> and its documentation is very well organized. Finally, it <strong>should not be used</strong> when building a <strong>static web site</strong> because it would not make use of most of its features.</td>
</tr>
</tbody>
</table>
# Evaluation form – uLab

<table>
<thead>
<tr>
<th>Technology name</th>
<th>WIJMO</th>
</tr>
</thead>
</table>

| Evaluation goal | Test whether or not WIJMO UI's components are useful and use it to provide a working prototype for the Live Scanner application |

| Initial hypothesis | WIJMO's pre-built UI components are functional, easy to use and compatible with AngularJS |

| Brief work summary | There has been made a simple web application called Live Scanner that consists on retrieving, modifying and sending data through REST APIs. The application has been built using AngularJS, HTML, CSS and WIJMO. The end result does not include any WIJMO component. |

<table>
<thead>
<tr>
<th>Obtained results</th>
<th><strong>Positive results</strong></th>
<th><strong>Negative results</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Many pre-built widgets</td>
<td>• Many widgets are incompatible with the AngularJS directives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Customer service was uneffective</td>
</tr>
</tbody>
</table>

| Hypothesis results | INVALIDATED |

| Conclusion | From this project’s experience, using WIJMO UI kit is **not advised, at least when working with AngularJS**. The team had many issues trying to make the widgets work. After a lot of futile effort, **its use was finally discarded**. The team contacted several times with the **customer support**, and while they answered, the responses were **not as useful or as quick as expected**. |
# Evaluation form – uLab

<table>
<thead>
<tr>
<th>Technology name</th>
<th>Jasmine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation goal</td>
<td>Test whether or not Jasmine's features are useful and use it to test a web application</td>
</tr>
<tr>
<td>Initial hypothesis</td>
<td>Jasmine is useful as a simple and organized test framework for Javascript applications</td>
</tr>
<tr>
<td>Brief work summary</td>
<td>There has been made a simple web application called Live Scanner that consists on retrieving, modifying and sending data through REST APIs. This application uses AngularJS as its web framework.</td>
</tr>
<tr>
<td>Obtained results</td>
<td></td>
</tr>
<tr>
<td>Positive results</td>
<td>Negative results</td>
</tr>
<tr>
<td>• Easy to use</td>
<td>• None</td>
</tr>
<tr>
<td>• Wide array of test helpers</td>
<td></td>
</tr>
<tr>
<td>• Clean and quick web results</td>
<td></td>
</tr>
<tr>
<td>• Effective error reports</td>
<td></td>
</tr>
<tr>
<td>• Compatible with AngularJS</td>
<td></td>
</tr>
<tr>
<td>Hypothesis results</td>
<td></td>
</tr>
<tr>
<td>Conclusion</td>
<td><strong>VALIDATED</strong></td>
</tr>
<tr>
<td></td>
<td>Jasmine does exactly what we want it to do; it provides a fast and easy way to set up test cases. The test results are shown on a web browser in a clean and functional way. It has been used in conjunction with AngularJS's mocking features and it has worked flawlessly.</td>
</tr>
</tbody>
</table>
### Evaluation form – uLab

<table>
<thead>
<tr>
<th>Technology name</th>
<th>Docker</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Evaluation goal</strong></td>
<td>Test Docker’s functionalities and ease of use involving application deployments</td>
</tr>
<tr>
<td><strong>Initial hypothesis</strong></td>
<td>Docker provides isolated environments to execute applications, in a repeatable and scalable way, and helps in normalizing deployments</td>
</tr>
<tr>
<td><strong>Brief work summary</strong></td>
<td>Many of Docker’s functionalities and architecture have been studied and tested. Both basic and Dockerfile-powered image creations have been tested, as well as the deployment of a connectivity-reliant application. A private Registry has been created and tested, as well as an auto-deployment tool to help developers even more with application deployment.</td>
</tr>
<tr>
<td><strong>Obtained results</strong></td>
<td></td>
</tr>
</tbody>
</table>
| **Positive results** | • Isolated execution environments  
• Reliable and repeatable deployments  
• Resource management (no VMs)  
• Caching system  
• Growing public repository  
• Open standards |
| **Negative results** | • Initial learning curve |
| **Hypothesis results** | VALIDATED |
| **Conclusion** | Docker is an **excellent solution** for application deployments. With it, system administrators can **deploy and run any application on any infrastructure**, provided it is **Linux-based** (or can emulate it somehow) and also helps developers to focus on its work by not having to worry about environments – once the application runs, it can run **anywhere**. Thus, it is especially useful for medium-large server compositions because it enables **normalized and repeatable environments through any possible environment**. |
9 Conclusions

9.2 Final planning

The team is greatly pleased to state that all of this project’s stages and milestones have been achieved and followed as it was initially planned.

The only deviation this project had was when developing the visual interface with WIJMO, due to its incompatibility with Angular – even when WIJMO’s documentation stated that it was compatible. However, the team soon noticed that the technology was not mature enough and terminated the work related with WIJMO before it consumed even more efforts and time.

Therefore, the remaining time that was scheduled to use WIJMO was spent to create a simple yet very effective web interface that would cover Live Scanner’s prototype needs. We are proud to say that Live Scanner is now on production level.

The study and deployments that were planned to do with Docker were achieved without any issue. All of the milestones were achieved, and the team accomplished the optional goal of making extra documentation for Docker new users, in the form of the workshop-oriented videos that have been referenced in the document.

9.3 Future uses

The Telefónica I+D team we have been working with are very pleased with the work done by the uLab team. They have shown special interest on AngularJS as a very valuable technology that is sure to be used on the next web projects, and thus it has been successfully added to Telefónica I+D’s technology stack. Jasmine has been added as a useful testing tool, while WIJMO has been discarded due to its inconsistencies.

Finally, Docker has been seen as a very valuable deployment tool, and it is sure to be used on future applications that would need to work on several environments.

9.4 Workflow retrospective

Following the advice given by agile methodologies, the team has done several retrospectives during the development of this project. The usual workflow for these meetings is as follows:

- Everyone takes some post-its and writes down what was good – and has to be kept – and what was bad – and should be changed – about the working process
- Then in turns everyone explains each of their written post-its, and then anyone can give their opinions and suggestions to improve
The team did a several of these meetings and many positives things were shared, while solving many other little workgroup issues. There were several recurrent points in these meetings, and these were the following:

Things that have been positive:

- Very good integration and cooperation between the team members
- Very good communication with the Telefónica team
- The team members are learning a lot
- The team members have gotten very involved with the projects

Things that should improve:

- There are too many organization tools
  - Solution: the organization tools number has been greatly reduced
- Since we are all students, some of our timetables are not compatible enough
  - Solution: the teams have been re-organized in order to meet more often than before

While taking little work time, these retrospectives have helped us greatly to improve our work processes.

9.5 Quick review

After several months of hard work, we have successfully finished this project, and we have done it as a team – the uLab team. There have been happy releases and stressful times, but the thing I value the most about this project is the people; I have worked and learned together with a group of excellent people, and I am very thankful for it. Together we have learnt how to start a project with a technology that very little of us knew; we have learnt how to communicate with a client - thanks to all the demos that we have done - ; but the most important thing is that we have learnt how to work as a team. Trusting each other in order to achieve a greater goal can be difficult but it is very rewarding, and I am sure that it is a skill that will be very useful in my professional career.

I would like to finish this document with my most sincere thanks to everyone involved in this project: all of the uLab Team, with its development and business areas, as well as the Telefónica I+D team, because without both of them this project would not have been possible.
10 Bibliography

[1] Telefónica I+D website

[2] inLab FIB website

[3] AngularJS website
   http://angularjs.org/ - March 16th, 2014

   http://docs.angularjs.org/tutorial - March 16th, 2014

   http://blog.koalite.com/2013/10/angularjs-6-meses-despues/ - March 16th, 2014

   http://joelhooks.com/blog/2013/05/22/lessons-learned-kicking-off-an-angularjs-project/ - March 16th, 2014

[7] Javadzone.com – 10 reasons web developers should learn AngularJS
   http://java.dzone.com/articles/10-reasons-web-developers - March 16th, 2014

   http://builtwith.angularjs.org/ - March 16th, 2014

   http://proquest.tech.safaribooksonline.de/9781782161820?uicode=CERN
   - March 16th, 2014

    http://cds.cern.ch/record/1625329 - March 16th, 2014

http://wijmo.com/topic/evaluating-wijmo-for-a-project-some-questions/
- March 16th, 2014

http://devwp.eu/wijmo-widgets-review/ - March 16th, 2014

[14] Jasmine website – Github repository

http://jasmine.github.io/2.0/introduction.html - March 16th, 2014

[16] Watirmelon.com – Why I don’t like Jasmine


[18] Twoalex.com – Test your JavaScript

[19] ACM Digital Library
http://dl.acm.org/citation.cfm?id=563429 - March 16th, 2014

[20] Docker website
https://www.docker.io/ - March 16th, 2014

[21] Docker website – About
http://www.docker.com/about_docker_inc/ - March 16th, 2014

[22] Docker website – Getting started
https://www.docker.io/gettingstarted/ - March 16th, 2014

[23] Docker website – Documentation - Introduction
[24] Lifehacker.com – Five best Virtual Machines applications
   http://lifehacker.com/5714966/five-best-virtual-machine-applications
   - March 16th, 2014

[25] Linux containers website
   http://linuxcontainers.org/ - March 16th, 2014

[26] AUFS website
   http://aufs.sourceforge.net/ - March 16th, 2014

[27] unionFS website
   http://unionfs.filesystems.org/ - March 16th, 2014

[28] ActiveState – Ben Golub explains Docker (1)
   http://www.activestate.com/blog/2014/02/ben-golub-explains-docker-inc
   - March 16th, 2014

[29] ActiveState – Solomon Hykes explains Docker (2)
   http://www.activestate.com/blog/2013/06/solomon-hykes-explains-docker
   - March 16th, 2014

[30] Docker Index
   https://index.docker.io/ - March 16th, 2014

[31] Microsoft Store- Windows 7 Home Premium
   http://www.microsoftstore.com/store/msca/en_CA/pdp/Windows-7-Home-
   Premium/productID.246627500 - March 16th, 2014

[32] Wijmo website – Purchase

[33] AmazonWS website – EC2 Pricing

[34] AngularJS Google+ - AngularJS MVVM
   https://plus.google.com/+AngularJS/posts/aZNVhj355G2 - June 9th, 2014


http://www.youtube.com/watch?v=ZzQfxoMFHOU

[37] InLab Blog – Docker video tutorials - June 9th, 2014

http://www.youtube.com/watch?v=NG_0DJ5aiLQ&list=UUFUOT6uotDe4kdCiaob8UJA

[38] Telefónica I+D public GitHub repository – Auto-deployment script - June 9th, 2014

https://github.com/PDI-DGS-Protolab/docker_deployment


http://googlecloudplatform.blogspot.com.es/2014/06/an-update-on-container-support-on-google-cloud-platform.html

[40] Docker website - Docker use cases - June 15th, 2014

http://www.docker.com/resources/usecases/

[41] Google Drive - WIJMO issue list - June 15th, 2014

https://docs.google.com/document/d/1bPkzrAwGa5Km-Fv6ZZoOcV77GsV16ZUI-P5-fU438/edit