Aplicación Web de Gestión de Proyectos

A Degree Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Ferran Fernández Garrido

In partial fulfilment
of the requirements for the degree in
TELEMATICS ENGINEERING

Advisor: Juan Luis Gorricho Moreno

Barcelona, June 2014
Abstract

The aim of the project was to develop a distributed and collaborative application to project management. It has the target of simply the development of projects through the creation of an online work groups.

The idea was to develop an API REST an also a web application associated to that API that would permit to manage projects. However, during the development of the project there were emerging new elements and challenges to incorporate to the project. These new challenges, also suppose the addition of new technologies that are not studied during the degree but I wanted to learn and explore

Finally the project has strayed somewhat from its original course does not focus much on the issue itself in terms of performance, but in the best way in which this application can be used as a collaborative tool based on the technology used in each case.
Resum

Aquest projecte consisteix en la creació d’una aplicació distribuïda i col·laborativa per la gestió de projectes. Té com a funció facilitar el desenvolupament de projectes mitjançant la creació de grups de treball online.

La idea principal va començar sent el desenvolupament d’una API REST i una aplicació web per gestionar projectes. Tot i que posteriorment durant el desenvolupament van sorgir nous elements i reptes a incorporar al projecte, que implicaven l’ús de moltes tecnologies no vistes a la carrera, i de les quals en volia investigar i aprendre.

Finalment el projecte s’ha desviat en certa manera del seu rumb original, centrant-se no tant en l’aplicació com a tal en quant a prestacions, sinó en la millor forma en la que aquesta aplicació pot ser utilitzada com eina col·laborativa en funció de la tecnologia aplicada en cada cas.
Resumen

Este proyecto consiste en la creación de una aplicación distribuida y colaborativa para la gestión de proyectos. Tiene como función facilitar el desarrollo de proyectos mediante la creación de grupos de trabajo online.

La idea principal empezó siendo un desarrollo de una API REST y una aplicación web asociada para gestionar proyectos. Aunque posteriormente durante su desarrollo fueron surgiendo nuevos elementos y retos a incorporar en el proyecto, que implicaban el uso de muchas tecnologías, las cuales, no se tratan durante la carrera pero de las que quería indagar y aprender.

Finalmente el proyecto se ha desviado algo de su rumbo original no centrándose tanto en la aplicación el sí en cuanto a prestaciones, sino en la mejor forma en la que dicha aplicación puede ser usada como herramienta colaborativa en función de la tecnología aplicada en cada caso.
Acknowledgements

I want to thank my tutor for always give my support and proposed me different challenges to deal with. Finally, I want to thank my family, friends and specially my girlfriend. All of them have been giving their support during the development of this project and without them this could not have been possible.
# Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>06/06/2015</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>20/06/2015</td>
<td>Document revision</td>
</tr>
</tbody>
</table>

## DOCUMENT DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferran Fernández Garrido</td>
<td><a href="mailto:ffernandez.upc@gmail.com">ffernandez.upc@gmail.com</a></td>
</tr>
<tr>
<td>Juan Luis Gorricho Moreno</td>
<td><a href="mailto:juanluis@entel.upc.edu">juanluis@entel.upc.edu</a></td>
</tr>
<tr>
<td>Marcel Fernández Muñoz</td>
<td><a href="mailto:marcel@entel.upc.edu">marcel@entel.upc.edu</a></td>
</tr>
</tbody>
</table>

## Written by: Reviewed and approved by:

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>06/06/2015</td>
<td>Ferran Fernández Garrido</td>
<td>Project Author</td>
</tr>
<tr>
<td>01/07/2015</td>
<td>Juan Luis Gorricho Moreno</td>
<td>Project Supervisor</td>
</tr>
</tbody>
</table>
# Table of contents

Abstract .......................................................................................................................... 1
Resum............................................................................................................................. 2
Resumen .......................................................................................................................... 3
Acknowledgements ......................................................................................................... 4
Revision history and approval record ............................................................................ 5
Table of contents ............................................................................................................ 6
List of Figures .................................................................................................................. 8
List of Tables: ................................................................................................................... 9
1. Introduction ................................................................................................................ 10
2. State of the art of the technology used or applied in this thesis: ........................ 11
   2.1. Front-End technologies ....................................................................................... 11
       2.1.1. JavaScript .................................................................................................. 11
       2.1.2. JQuery ........................................................................................................ 11
       2.1.3. HTML5 ....................................................................................................... 11
       2.1.4. CSS3 ......................................................................................................... 11
   2.2. Back-End technologies ......................................................................................... 12
       2.2.1. SQL ............................................................................................................ 12
       2.2.2. REST ......................................................................................................... 12
       2.2.3. WEBSOCKETS .......................................................................................... 13
   2.3. Server ................................................................................................................... 14
       2.3.1. Glassfish ................................................................................................... 14
3. Methodology / project development: ......................................................................... 15
   3.1. Database structure .............................................................................................. 15
   3.2. Application workflow ........................................................................................ 17
   3.3. REST Server architecture .................................................................................. 18
       3.3.1. API REST structuration ........................................................................... 19
       3.3.2. Explaining of REST application development ......................................... 22
       3.3.3. View refresh problems ............................................................................. 23
       3.3.4. Other problems ......................................................................................... 25
   3.4. WebSockets Server architecture ......................................................................... 26
       3.4.1. Types of messages and requests ................................................................. 27
       3.4.2. Explaining of WebSockets application development .............................. 29
       3.4.3. Problems .................................................................................................... 30
4. Comparison between technologies ................................................................. 31
   4.1. Comparisons .......................................................................................... 31
       4.1.1. Technical level .............................................................................. 31
       4.1.2. Experience level user ................................................................. 34
   4.2. Can both technologies work together? .................................................. 35
5. Budget ........................................................................................................... 36
6. Conclusions and future development: ......................................................... 38
   6.1. Which is the best technology for the application? ............................... 38
   6.2. Which technology will be more suitable in futures applications? .......... 38
Appendices: .................................................................................................... 40
   All the code is added in the attached .rar file, the file contains: .................. 40
Glossary ............................................................................................................ 46
List of Figures

Figure 1. Database structure .............................................................................................................. 15
Figure 2. Application workflow ......................................................................................................... 17
Figure 3. Rest server architecture .................................................................................................... 19
Figure 4. Projects panel ..................................................................................................................... 22
Figure 5. Old polling waiting time .................................................................................................... 23
Figure 6. New polling waiting time ................................................................................................... 24
Figure 7. WebSocket server architecture .......................................................................................... 26
Figure 8. Users panel ........................................................................................................................ 29
Figure 9. Gantt panel ......................................................................................................................... 30
Figure 10. Comparison with constant payload .................................................................................. 32
Figure 11. Comparison with variable payload .................................................................................. 33
List of Tables:

Table 1. API REST Example ................................................................. 13
Table 2. WebSocket example .............................................................. 13
Table 3. Users API definition ............................................................ 19
Table 4. Projects API definition ......................................................... 20
Table 5. Tasks API definition .............................................................. 20
Table 6. Subtasks API definition ......................................................... 21
Table 7. Resources API definition ....................................................... 21
Table 8. Key, value table example ...................................................... 26
Table 9. GET WebSockets requests ...................................................... 27
Table 10. Others WebSockets requests ................................................ 28
Table 11. Table of constant payload messages ..................................... 32
Table 12. Table of variable payload messages ..................................... 33
Table 13. Price table ............................................................................ 36
1. **Introduction**

This project has been developed in the Telematics Engineering department of UPC

The initial scope of this project was developing a web application for project management, in which had the basic functions of a project development application: generate a Create projects, Create tasks, Edit tasks, Gantt diagram, etc. The program had to be also user friendly because it was oriented to people with little to no knowledge in IT (Information Technology) and it also had to be as efficient as possible.

In order to do all the previous statements, the following step in the application development had to be an API REST development. This API would allow not only the development of this particular program; it would also let people to develop their own project management applications using REST technology.

Consequently, the aims of the project were:

1. Create a database with all the necessary tables.
2. Develop the API REST based on the database.
3. Develop the web application for the final user.

These milestones have been modified during the progress of the project due to the fact that some problems have appeared. I will explain more deeply these problems in next sections.

All the modifications of the project plan have been edited in the work packages and Gantt diagram which are available in the appendix.

Because of these incidences had been appearing the final scope of the project had to be changed. Now the idea was to develop two applications that were the same for the user in terms of usability but completely different in the Back-End development.

In the first approach, the original one had been developed using the first technology, REST technology.

REST application had been almost not modified at all, which meant that all the work made before the problem that we will introduce in future sections, would been reusable.

The second application has been made using the WebSocket technology which will also introduce in coming sections.

To conclude, at the end of the project the final idea was to develop two different application using two different technologies and to give answers to some raising questions: Which technology has been the best one or more suitable for this particular application? Which technology will be the best one in the nearing future?

Along this thesis we will try to find the answer to these questions and solve all the other problems that had appeared during the development of the two applications.
2. **State of the art of the technology used or applied in this thesis:**

During the development of this project have been used a bunch of frond-end and back-end web technologies that now we are going to introduce.

2.1. **Frond-End technologies**

2.1.1. **JavaScript**  
JavaScript is an object oriented language that was created in 1995 by Netscape, is a dynamic programming language used as a part of web browsers. JavaScript can be used in frond-end technologies or in back-end technologies. However, it is more common as a front-end technology like in this project.

2.1.2. **jQuery**  
jQuery is not a programming language but has been a fundamental library for this application.

As I was saying JQuery is defined as a JavaScript Library that was relied in 2006 under a GNU licence. The main points of the JQuery library are:

1. The interaction with the DOM elements, which has been very important for this project.
2. It is able to handle events and manipulate a CSS sheet to create animations.
3. Implements AJAX requests which is has been fundamental for this project, without this characteristic it would not have been possible to develop the REST application.

There are many other characteristics of this library but these are the more relevant ones.

2.1.3. **HTML5**  
HTML5 (*HyperText Markup Language*, version 5) is the basic language used by the World Wide Web. This standard was released in October, 2014. At the beginning was not recognised by all the available web browsers but nowadays almost all of them process the new standard.

The creators of the HTML5 have introduced some new mark-ups that have been used in the projects such as the `<nav>` mark-up. It will be explained the use of these in previous sections and also in the appendix where we can see all the code.

2.1.4. **CSS**  
CSS3 (*cascading style sheets*) is a language used to define the shape and the structuration of the HTML document. In this project in order not to develop all the CSS style sheets for all the elements has been used a CSS framework called *Bootstrap*, developed by Twitter an released in August 2011.
2.2. **Back-End technologies**

The back-end technologies are the ones that are executed in the server side, I have used a plenty of them during the development of the project. As I did in the Front-End section now I am going to describe these technologies.

2.2.1. **SQL**

SQL (*Structured Query Language*) is a declarative language used to access to a database, was released in 1974 by IBM. Allow the programmer doing a lot of actions such as insert data to the database, edit it, count the rows of the tables, etc. Has been fundamental in this project. For the REST application has been used HQL (Hibernate Query Language) and for the WebSocket application has been used the SQL native language.

The main difference between SQL and HQL is that HQL extends the concept of object oriented languages to SQL so the result is not plain data, is a combination of objects. There is no reason why I used one or another in the project because the use in each application is almost the same. However, the idea of using different types of languages was somehow like a challenge for me.

2.2.2. **REST**

REST (Representational State Transfer) is a software architecture style that allows the creation of a good scalable web service, was firstly named by Roy Fielding in his PhD thesis in 2000. The design of a REST web service is based on the creation of a bunch of constraints applied over the components in a distributed hypermedia system.

Nowadays REST has been accepted across the web as an easier alternative to other web services architectures such as SOAP (Simple Object Access Protocol) or WSDL.

SOAP is a standard protocol that defines how two objects from different processes can communicate each other using XML (Extensible Markup Language) data. Meanwhile WSDL (Web Services Description Language) also use XML and describe the way of a message structuration in order to interact to a services list.

Now that we have already defined the competitors of the REST software architecture we are going to introduce how REST works.

RESTful systems typically communicate over Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) used by web browsers to retrieve web pages and send data.

Here we can see an example of communication using REST. This is not a real example from the project, the real API REST description will be described in next sections.
Table 1. API REST Example

Another important aspect is that the client-server communication is further constrained because any data is stored on the server between requests. This means that each request from the client has to be new, there is no state to store inside the server. This characteristic is very important to keep in mind because was one of the main problems in the project, and will be explained in future sections.

Nowadays almost all the state-of-the-art technological companies have a REST API for developers, some examples are: Google, Facebook, Twitter, Dropbox, etc.

2.2.3. WEBSOCKETS

Websocket is a technology that provides a bidirectional communication canal that can work in full duplex over one TCP socket. Was designed to be implemented in web browsers and web servers. The Websockets API is being approved by W3C meanwhile the Websocket protocol was normalized in December 2011 by the IETF in the RFC 6455.

Now I am going to explain how WebSockets work. In order to enable the communication between the client and the server, the client sends a negotiation request.

Table 2. WebSocket example
There are two 8 byte random tokens that the server will use to build a 16 byte token which, after an MD5 process will be used as an identifier in order to label that particular socket and know where the data has to be sent.

There are still some problems when we use WebSockets, also the API is not completely defined yet, which could be a difficult problem to deal with as we will see in future sections, the main problem of WebSocket appears when we use proxies. Usually proxies detect the WebSocket connections and when the see that the connection is never closed, they detect that as a problem or a possible attack and decide to close the connection. That of course could be a problem because when the clients or the server try to send data through the Socket the communication will fail.

There are some alternatives to solve that problem but the real solution will appear when WebSockets will be accepted by proxy servers.

2.3. Server

The server is one of the most important parts of the project, all the previous back-ends technologies are implemented in a server. In this project we have chosen a Glassfish server to store our applications.

Of course it has also deployed another SQL server as I said before that is connected to the Glassfish server. The SQL server and the Glassfish servers can be in different physical machines but in this projects both servers are in the same machine.

2.3.1. Glassfish

Glassfish is an applications server developed by Sun Microsystems that implements the technologies defined over the Java EE platform and allow the execution of applications that follow this specification.

It is derived from Apache Tomcat that works as a servlets container. We have to make a point here because Tomcat is not an applications server but contains the Jasper compiler that transforms the applications to servlets.

Glassfish server also implements the Grizzly component that uses Java NIO in order to provide scalability and faster process. Thanks to that final component I have been able to build the WebSocket application, because the last stable version of the component includes new framework components to deal with WebSockets.
3. **Methodology / project development:**

3.1. **Data base structure**

The first thing there should be kept in mind is that in order to develop this project is fundamental to develop a good database, because the users are going to create a huge quantity of information.

Therefore it was decided to create server that contains an schema that have 5 tables that are going to be described in a few seconds. First of all we have to know where this information is going to be stored. As it has been saying before, the idea was to build a SQL server, this server is not the same server that we use to store the application so is necessary some “linkers” to create a communication between them. All the necessary code is described in the appendix.

This picture shows the structure of the schema that contains the 5 tables.

The first table is the users table, which is not exactly a good name. Because someone may think that this table contains users as individuals. That is not true, because instead of contain users as persons itself contains groups, p.e: Marketing group, Telecommunication s group, etc.

Once said that we can see that the table users is related with the table projects in the following way, one group may have multiple projects but one project only can be assigned to one group.
All the remaining tables are related with the project table in a direct way and to the users table in an indirect way.

The relation between the table of projects and the tasks table can be seen as the same, which means that one project may have multiple tasks but one task only can have one project assigned.

The task table is the most complicated one, because have two types of relations one task could have multiple subtask and also multiple resources associated to that task.

Finally the resources table was made when the initial target was only to develop a REST application, have the same relations that the subtasks table so can be used like the subtask table with some modifications.

It can be appreciated all the tables have an id row, which is a not nullable row, the use of this row has been always used as a primary key which ensure that the id is unique in all the table. This is very useful in order to avoid repetitions and future problems in the indexation of the database.

The creation of that id is detailed in the appendix and works as a hash, made with some parameters of the object that have to be indexed in the database.

At the beginning of the project the idea was to use all the tables because there was only one application to develop. Because of the refreshing view problem, which will be explained in following sections, I was forced to use the WebSocket technology to develop another application, so I was not able to use all the possible resources in the database to develop the final application but the option is available.
3.2. **Application workflow**

The following schema shows the common actions that can be done during the usage of the application

![Application workflow diagram](image)

**Login:** In the login screen three possible situations may happen

1. The user is correct so the following view will be the user Dashboard.
2. The user is incorrect you will still stay in the login screen.
3. No internet connection to authenticate. It is shown as an information pop-up

**User Dashboard:** Three possible situations may happen

1. Stay in User Dashboard view.
2. Navigate to Projects Dashboard view.
3. Logout

**Projects Dashboard:**

1. Navigate to User Dashboard view.
2. Select project
   a. Add project
   b. Delete project
   c. Add task
      i. Add Subtask
      ii. Delete Subtask
   d. Edit Task
   e. Delete Task

   f. Generate Gantt

3. Logout
3.3. **REST Server architecture**

The server created in the REST project uses a Façade pattern that also implements mutual exclusion in order to provide better access to the final resources.

At this point one could be thinking: Why did you use a Façade pattern?. Well it was not the first choice, at the beginning of the project I did some research about software patterns and the other option was to build the server side with the Singleton pattern.

The Singleton pattern restricts the instantiation of a class to one object which is more or less what we need. But this pattern has two different problems that caused some doubts and finally used the Façade pattern.

On one hand, mutual exclusion, this pattern is vulnerable to multiple execution threads, which means that if there are lots of clients connected at the same time using a common resource could turn into a problem. It was one of the main issues to solve.

On the other hand the Singleton pattern allows the user to access the final object, which it was not the expected result. The idea was to allow the user to access to a subsystem of objects but not to the object. Meanwhile the second issue was not really important for me that is why the first implementation of the test back-end infrastructure was made using Singleton but in the end the realization came that allowing the access to the final object could turn into a security problem.

To start concluding the Singleton pattern was not the best option to implement the back end infrastructure that is why I used the Façade pattern.

The Façade pattern provides a simplified interface that allows the access to more complex objects that are underlined. In this particular project these complex objects are for example: Projects, Tasks, Users, etc. Consequently the client cannot access directly to the final resource and do wherever he wants, which could obviously be really dangerous due to the fact that the client would have control over the data base without any restriction. It is like a filter that delimitates the actions that can be taken by the clients.

The following schema shows how the Façade pattern works.
3.3.1. API REST structuration

The following structuration shows how a web developer must build its requests in order to get the correct content.

The first part of the URI resource has been omitted due to convenience during the construction of the table. However, in order to provide a full information of the API example will be written to provide a full example.

```
http://{SERVER IP}:8080/ProjectManagement/webresources/{SPECIFIC RESOURCE}
```

In order to provide a better development environment, the API requests are able in XML a JSON format.

**Users:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/webresources/users/</td>
<td>GET</td>
<td>Returns all the existing users but without the <strong>id</strong> and the <strong>password</strong></td>
</tr>
<tr>
<td>/webresources/users/{idUser}</td>
<td>GET</td>
<td>Returns all the user information, <strong>password</strong> included.</td>
</tr>
<tr>
<td>/webresources/users/count</td>
<td>GET</td>
<td>Count the number of existing users.</td>
</tr>
<tr>
<td>/webresources/users/{idUser}</td>
<td>PUT</td>
<td>Edit the user with the specified <strong>id</strong>.</td>
</tr>
<tr>
<td>/webresources/users/</td>
<td>POST</td>
<td>Creates a new user.</td>
</tr>
<tr>
<td>/webresources/users/{idUser}</td>
<td>DELETE</td>
<td>Deletes the user with the specified <strong>id</strong></td>
</tr>
</tbody>
</table>

Table 3. Users API definition

---

**Figure 3. Rest server architecture**

---
## Projects:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/webresources/projects/{idUser}</td>
<td>GET</td>
<td>Returns all the associated Projects from the specified user.</td>
</tr>
<tr>
<td>/webresources/projects/{idUser}/{idProject}</td>
<td>GET</td>
<td>Return only the specified Project of the user.</td>
</tr>
<tr>
<td>/webresources/projects/count</td>
<td>GET</td>
<td>Count the number of existing projects.</td>
</tr>
<tr>
<td>/webresources/projects/{idProject}</td>
<td>PUT</td>
<td>Edit the Project with the specified id.</td>
</tr>
<tr>
<td>/webresources/projects/progress/{idProject}</td>
<td>PUT</td>
<td>Updates de progress of the specified Project.</td>
</tr>
<tr>
<td>/webresources/projects/</td>
<td>POST</td>
<td>Creates a new project.</td>
</tr>
<tr>
<td>/webresources/projects/{idProject}</td>
<td>DELETE</td>
<td>Deletes the specified project</td>
</tr>
</tbody>
</table>

Table 4. Projects API definition

## Tasks:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/webresources/tasks/{idProject}</td>
<td>GET</td>
<td>Returns all the tasks associated to the specific Project.</td>
</tr>
<tr>
<td>/webresources/tasks/{idProject}/{idTask}</td>
<td>GET</td>
<td>Returns the specific task related to the project.</td>
</tr>
<tr>
<td>/webresources/tasks/progress/{idTask}</td>
<td>PUT</td>
<td>Updates the progress of the specified task.</td>
</tr>
<tr>
<td>/webresources/tasks/{idTask}</td>
<td>PUT</td>
<td>Edit the task with the selected id.</td>
</tr>
<tr>
<td>/webresources/tasks/</td>
<td>POST</td>
<td>Creates a new task</td>
</tr>
<tr>
<td>/webresources/tasks/{idTask}</td>
<td>DELETE</td>
<td>Deletes the task with the specified id.</td>
</tr>
</tbody>
</table>

Table 5. Tasks API definition
### SubTasks:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/webresources/subtasks/{idTask}</td>
<td>GET</td>
<td>Returns all the sub tasks associated to a task.</td>
</tr>
<tr>
<td>/webresources/subtasks/{idTask}/{idSubTask}</td>
<td>GET</td>
<td>Returns the specified sub task associated to the specified task</td>
</tr>
<tr>
<td>/webresources/subtasks/progress/{idSubTask}</td>
<td>PUT</td>
<td>Updates the sub task progress</td>
</tr>
<tr>
<td>/webresources/subtasks/{idSubTask}</td>
<td>PUT</td>
<td>Edit the specified sub task</td>
</tr>
<tr>
<td>/webresources/subtasks/</td>
<td>POST</td>
<td>Creates a new sub task</td>
</tr>
<tr>
<td>/webresources/subtasks/{idSubTask}</td>
<td>DELETE</td>
<td>Deletes the specified sub task</td>
</tr>
</tbody>
</table>

Table 6. Subtasks API definition

### Resources:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Request</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>/webresources/resources/{idTask}</td>
<td>GET</td>
<td>Returns all the specified resources associated to the specified task</td>
</tr>
<tr>
<td>/webresources/resources/{idTask}/{idresource}</td>
<td>GET</td>
<td>Returns the specified resource associated to the task.</td>
</tr>
<tr>
<td>/webresources/resources/{idresource}</td>
<td>PUT</td>
<td>Edit the resource associated to the task</td>
</tr>
<tr>
<td>/webresources/resources/</td>
<td>POST</td>
<td>Creates a new resource</td>
</tr>
<tr>
<td>/webresources/resources/{idresource}</td>
<td>DELETE</td>
<td>Delete the specified resource.</td>
</tr>
</tbody>
</table>

Table 7. Resources API definition
3.3.2. Explaining of REST application development

The first step that had to be done in the development of both applications was the creation of the HTML mock-ups. It was a complicated process because it was necessary to know all the possible actions that the user would take in order to build a successful mock-up.

The idea was to avoid spending much time in the HTML esthetical design so that I took the decision to use the Bootstrap CSS Framework that really helps the developer in the creation of an aesthetically friendly environment in few steps.

Once solved the visual problem, the initial idea was to create the login engine for the users. Before the full explanation we have to take account that the REST architecture is stateless, so it really do not have any information about who did the request. So how did we do an authentication system that does not have memory?

The solution was move all the intelligence that the server does not have to the client using cookies. In conclusion the user authentication follows the following steps:

1. The user information is sent to the server.
2. The server responds in a positive way or in a negative way.

If the response is positive, which means that the user actually exists the id of the client is stored in the client using a cookie that will be used in future functions.

Otherwise if the answer is negative, any cookie is created and an information message is deployed into the user screen. There is another obvious option, if there is no internet connection; another message is deployed advising that there is no internet connection.

Having solved the problem of the authentication, the next challenge was to develop the project panel. As it was said before all the mock ups were made in a previous step, so now it was necessary to fill all the back-end code to complete the application.
The most challenging aspect of the project panel was the idea of using individual refresh in order to update the current project. So when the user clicks over one project, there is any new global web page refresh, all the individuals’ objects inside the DOM are updated. To do so it was necessary to determinate what was the current project id in order to link the entire element to that specific identifier. All the technical aspects of how it was made that implementation could be found in the appendix.

Once solved all the previous problems, the main issue that was faced was the idea of the simultaneous refresh in all the users screen when one of them produce any change we are going to talk about that problem in the next point.

3.3.3. View refresh problems

The main problem of the REST application appeared when we take account that the view of all the users had to be refreshed. As aforementioned, the REST technology is stateless so there are no memory sessions for users. As a result, the server can’t detect if any update has appended so in order to solve that problem we decided to implement a polling method to refresh the data of the view.

The traditional polling method consist in sending requests to the data base or web resource that it is been observed in order to obtain continuous data updates.

There has been an attempt to optimize the polling system so that it has been created a new polling method a bit different from the traditional ones. Usually polling scripts sends a huge quantity of information in a request without thinking if this data is going to be used for the application in that moment or not.

The following picture shows the waiting time of huge data packets, 4.2KB of information. Sent by 60 “users”, that contains all the tasks from a particular project.

![Waiting time for a 4.2KB data packet](image)
As we can see, the use of that kind of polling penalizes the final user because the average waiting time for these requests is approximately: 274ms. That calculation was made without counting the process final time that the application must do when the data is received.

The polling method used in this project is slightly different from the standard polling method. I have used small packages of data as flags to determine when it is necessary to send a big request by the user. An example will be provided to illustrate the explanation given the difficulty it presents to its understanding.

Let us suppose that the remaining tasks in the database are 7, the client will be asking for the number of tasks in an infinite loop and comparing the response with the previous obtained value stored in a web cookie. If something happens, a task is erased for example, in the client the number of tasks stored and the number of tasks retrieved by the server will be different. So then we take the decision to update the view and ask for all the information in a big request.

The following picture illustrates the performance of that method:

![Waiting time for a 262Bytes data packet](image)

In conclusion, with the traditional method, the user lasts 274ms to receive the entire information package. Now the time is reduced to 147ms, if we add an average process time taken by an *append* method in jQuery the final time will be 255ms. As a result the user will be able to get all the data faster than before.
### 3.3.4. Other problems

There were another problems during the development of the REST application, mostly related with the external frameworks and libraries that sometimes There had to be an adaptation or accepting their limitations in order to keep developing the application.

The first problem that I had was with the Bootstrap twitter framework. The application uses one kind of special pop-up named Modal. Usually Bootstrap only provide the CSS functionality of their objects but in that particular implementation, they also add some Javascript functions to display the pop-up.

After being tried the native Modal function I realized that when it was summon for a second time the pop up always suddenly closed and the user could not introduce the data. So that I decided to implement my own call-back function to display the pop up and solve the problem.

Regarding, another problem produced by an external library was the Gantt diagram render problem. In this case the used library was the dhtmlx. When I tried to refresh the gantt view trying to re-render the entire page was impossible because of internal characteristics of the library. In that particular case I was not able to solve the problem for the REST application but I could for the WebSocket application because the messages provided by the WebSocket call-back functions permit to rebuild all the view with all the necessary changes.

After the refresh problem we had to take the decision of keep developing the REST application with that associated problem, or try to develop another application with a common front-end but with a new back-end technology.

Finally, we decided to use WebSocket technology to develop that new application that would solve the refresh problem, and also keep developing the REST application in order to do a final comparison between them.

There will be an explanation about the WebSockets development and how it was possible to develop the final application.
3.4. **WebSockets Server architecture**

The WebSocket server architecture has been designed to be similar to the REST server structure. We have invented different types of messages in order to provide a communication protocol between the client and the server. The types of messages will be described further on.

![WebSocket server architecture](image)

The previous image illustrates an example of the communications between the clients and the server, now we will try to explain how it works using the previous example.

The first thing that we have to keep in mind is that the WebSockets API is really new, so there are no functions to handle sessions yet, so we have to create our session’s handler. As a result our server will store a Hashmap to identify our clients, similar to the following table:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0et75..</td>
<td>123456</td>
</tr>
<tr>
<td>U8eyg....</td>
<td>123456</td>
</tr>
<tr>
<td>TBbg23...</td>
<td>78823</td>
</tr>
</tbody>
</table>

*Table 8. Key, value table example*
After that we have to distinguish two big groups of messages:

**GET** messages: Only refresh your view

**POST/PUT/DELETE/OTHERS:** Refresh the views of everybody who belongs to your group.

Following the previous example, let us suppose that **Client 3 (Group id: 78823)** wants to refresh the **project view**, then he sends a GET message and the server responds only to it through its socket **TBBg23**...

Meanwhile, the **Client 1 (Group id: 123456)** deletes a task, which means that all the members of the group have to be informed, as a result the message will be send to all the members who have the group id 123456 through their own sockets.

### 3.4.1. Types of messages and requests

<table>
<thead>
<tr>
<th>Resource</th>
<th>&quot;&quot;Request&quot;&quot;</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>users/{idGroup}/{idGroup}/get</td>
<td>GET</td>
<td>Returns the group information</td>
</tr>
<tr>
<td>users/{idGroup}/{info}/postUser</td>
<td>POST USERS</td>
<td>This a very rare type of request, it is only used when a new user is created.</td>
</tr>
<tr>
<td>projects/{idGroup}/{idGroup}/get</td>
<td>GET</td>
<td>Returns all the projects associated to that group.</td>
</tr>
<tr>
<td>projects/{idGroup}/{idProject}/gantt</td>
<td>GANTT</td>
<td>Returns the project associated to the group id and in a gantt format.</td>
</tr>
<tr>
<td>dates/{idGroup}/{idProject}/get</td>
<td>GET</td>
<td>Returns the dates associated to that specific projects, this is very useful in order not to retrieve always all the information.</td>
</tr>
<tr>
<td>datesTask/{idGroup}/{idTask}/get</td>
<td>GET</td>
<td>Returns the dates associated to that specific task.</td>
</tr>
<tr>
<td>tasks/{idGroup}/{idProject}/get</td>
<td>GET</td>
<td>Retrieves all the tasks, associated to that project.</td>
</tr>
<tr>
<td>subtasks/{idGroup}/{idTasks}/get</td>
<td>GET</td>
<td>Retrieves all the subtasks associated to that task.</td>
</tr>
</tbody>
</table>

**Table 9. GET WebSockets requests**
<table>
<thead>
<tr>
<th>Resource</th>
<th>&quot;Request&quot;</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>projects/{idGroup}/{info}/post</td>
<td>POST</td>
<td>Creates a new project</td>
</tr>
<tr>
<td>projects/{idGroup}/{idProject}/{info}/put</td>
<td>PUT</td>
<td>Updates all the necessary field of the project</td>
</tr>
<tr>
<td>projects/{idGroup}/{idProject}/delete</td>
<td>DELETE</td>
<td>Deletes the specified project</td>
</tr>
<tr>
<td>projects/{idGroup}/{idProject}/progress</td>
<td>PROGRESS</td>
<td>Updates only the progress of the project.</td>
</tr>
<tr>
<td>tasks/{idGroup}/{idProject}/{info}/post</td>
<td>POST</td>
<td>Creates a new task.</td>
</tr>
<tr>
<td>tasks/{idGroup}/{idTask}/{info}/put</td>
<td>PUT</td>
<td>Edits the selected task</td>
</tr>
<tr>
<td>tasks/{idGroup}/{idTasks}/delete</td>
<td>DELETE</td>
<td>Deletes the specific task</td>
</tr>
<tr>
<td>subtasks/{idGroup}/{idsubTask}/{info}/post</td>
<td>POST</td>
<td>Creates a new subtask</td>
</tr>
<tr>
<td>subtasks/{idGroup}/{idsubTask}/{info}/put</td>
<td>PUT</td>
<td>Edits the specified subtask</td>
</tr>
<tr>
<td>subtasks/{idGroup}/{idsubTask}/delete</td>
<td>DELETE</td>
<td>Deletes the specified subtask</td>
</tr>
</tbody>
</table>

Table 10. Others WebSockets requests
3.4.2. Explaining of WebSockets application development

As we did in the REST application development chapter I am going to start explaining the users’ authentication system.

Thanks to the previous web mock-ups that were already set with the HTML designs, our front-end work was almost completely made. Because of that there was the possibility to develop all the back-end authentication system faster than before.

The first thing that we have to keep in mind is that our server is not stateless anymore so it is able to store our state. Due to that our authentication system will work as follows:

0- A WebSocket connection is created.
1- The user will authenticate into the application.
2- The server will check if that user exists or not. If the user exists, the next step will be creating a cookie in the client for future use. Meanwhile the server will register the user session associated to the socket id to know where will have to send the future messages.

The way of how the client and the server communicate has been explained in previous sections so I am not going to detail all the process.

Thanks to the acquired knowledge about how to refresh information in a specific place of the DOM, the development of the project panel was easier than before. The problem was to refresh all the elements without enter in an infinite loop of request due to the relation of the elements inside the database. However, this problem was really easy to solve compared with other problems that we will introduce in next chapters.

Figure 8. Users panel
3.4.3. Problems

During the development of the WebSockets application I had some problems associated to the use of the technology. All the main problems—from easier to more complex—are going to be explained.

The first thing that was realised was that the WebSocket technology has a handshake time. During that time you cannot send any message through the Socket so you have to wait until all the connections had been established.

In Java language for example, this issue is easy to solve we have just to program an sleeping time for the main thread in order to wait until the connection is established. However, JavaScript does not have sleeping functions so the solution was to build a kind of timer that allows us to wait the necessary time to establish the connection.

Another important problem that appeared was handle dates in the program. Because the first implementation of the application was made following almost all the REST messages system. It was an unsuitable idea because the REST application is able to make different types of request and handle these requests in a different functions and the web sockets architecture only handle all the messages in one function onMessage() so it is very
complicated to determinate if that specific message has to be processed or not by the client in every moment.

From another point of view, the gantt library dhtmlx it was not a problem anymore. Due to the render process had changed for the WebSockets application. When we did the REST application, the continuous flow of packets made impossible to re-render only the affected part of the gantt because we do not know which task is affected for example due to the fact that we are working in a stateless system. In the WebSockets application the packets of information are only sent when it is necessary and focused only in the necessary part that had to be rebuilding in the view. Thanks to that the Gantt views of the WebSocket application, permit more actions than REST application.

4. **Comparison between technologies**

4.1. **Comparisons**

Now it is going to be compared both technologies from different points of view. The first comparison would be as a technical level in general, and the other one as an experience level user in that particular application.

4.1.1. **Technical level**

First of all we have to make a distinction. There has been a discussion about technologies all throughout the document, but is not completely true. The only reason why this was the approach taken was to simplify the explanation for the reader up until this point.

REST is a style of architecture so that it cannot be compared with a technology like WebSockets. But the term is so loosely that can be used in place of each other.

It should start with reviewing very fast the WebSocket technology main characteristics to have a better idea about what we are talking about.

1. Single TCP connection: Usually all the HTTP requests creates a new TCP connection. As we saw in previous chapters, WebSockets technology only uses one TCP connection that would be use to transfer all the necessary information between the server and the client.
2. Is bidirectional: Which means that once initialized the socket, the server and the client would be able to transfer messages in both ways.
3. Full-duplex: The previous point was an introduction to this point. Not only the communication is bidirectional, but it is also full-duplex. This means that they can communicate simultaneously.
Now we are going to make some benchmark. We are going to compare the time to process X messages with a constant payload using both technologies:

![Figure 10. Comparison with constant payload](image)

Table 11. Table of constant payload messages

<table>
<thead>
<tr>
<th>Messages</th>
<th>Rest(ms)</th>
<th>WebSocket(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>99</td>
<td>8</td>
</tr>
<tr>
<td>19</td>
<td>131</td>
<td>11</td>
</tr>
<tr>
<td>23</td>
<td>164</td>
<td>12</td>
</tr>
<tr>
<td>29</td>
<td>210</td>
<td>12</td>
</tr>
<tr>
<td>35</td>
<td>277</td>
<td>12</td>
</tr>
<tr>
<td>41</td>
<td>328</td>
<td>14</td>
</tr>
<tr>
<td>46</td>
<td>375</td>
<td>14</td>
</tr>
<tr>
<td>51</td>
<td>421</td>
<td>15</td>
</tr>
</tbody>
</table>
We can see that the REST overhead increases when the number of messages increase. This phenomenon has a logical reason. That increase of time is produced due to the fact that a new TCP connection has to open every time.

Now we are going to do a different experiment, we will change the payload and fix the number of messages.

![Figure 11. Comparison with variable payload](image)

<table>
<thead>
<tr>
<th>Payload (bytes)</th>
<th>Rest(ms)</th>
<th>WebSocket(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>498</td>
<td>38</td>
</tr>
<tr>
<td>100</td>
<td>507</td>
<td>43</td>
</tr>
<tr>
<td>500</td>
<td>518</td>
<td>55</td>
</tr>
<tr>
<td>1000</td>
<td>526</td>
<td>61</td>
</tr>
<tr>
<td>5000</td>
<td>593</td>
<td>121</td>
</tr>
</tbody>
</table>

**Number of messages 500.**

**Table 12. Table of variable payload messages**
Seeing the previous graph we have realised that the processing time invested in the request/response REST endpoint is nearly 0 so most of the time is spent in the connection.

4.1.2. Experience level user

Once we have compared both technologies in a technical level, the next question is: And as user experience?

Well that question could be really hard to answer because it is not only related with the technology itself, it is also related with the optimization of the application. To try to answer that question we would focus on this particular application so we are not going to generalize.

The first tests that I did with both applications do not show any special difference in a way of working. They do more or less the same actions when they have to edit, delete or add a task or another function, it also take more or less the same time updating the view when one of the previous actions is made.

However, when multiple users is logged in the same session of the REST application it is possible to appreciate more delay and also more “flashy” view in some parts of the user and project panel due to the continuous update. Of course this is not a huge handicap, you can keep working without any problem, but maybe the most strict users would demand a better performance.

This problem is not present at the WebSocket application because this application only updates when a local change is made. It has to be remembered that the REST application is polling the global number of tasks, projects, etc; so if another user from another session does an action over one task all the users from all the sessions would update their view. It does not happened in WebSockets.

That means that WebSockets application does not have any problem? Not at all, the main constraint for the user experience in WebSocket appears when you initially log on to the platform. Because of the fact that you have to stablish the connection with the server, using the WebSocket protocol, the final user would perceive a certain delay. No more than 2 seconds but again if you are very strict that would be a problem for you.
4.2. **Can both technologies work together?**

In this particular application both technologies can work together but under some conditions.

If you are working in different sessions, which means that you may be working as a Marketing group with the REST application and your partner as a Development group with the WebSockets application. There are not any problem both application works great loading and getting their information from the same database.

However, there are some difficulties if you are working in the same session. If you update some information from the WebSockets application, for example deleting a task, the REST application view is updated and also the WebSocket view of course. But this does not work in the opposite way, it means that if you delete a task from the REST application, the WebSocket application will not refresh the views until the all page will be refreshed or until some WebSocket user does some action over the application.

The explanation of this problem is really easy to explain but really hard to solve. Due to the fact that the REST application is polling continuously the database, any change produced in that database would produce an immediate change to the application even if this change was made from a foreign application, WebSocket application in that particular example. But in WebSockets only the users associated to the server can change the value of the database, which means that the architecture is not prepare to accept refreshes when a foreign applications change the database information.

The only way to solve that problem would be polling the database using WebSockets too but that solution of course was discarded because the idea was to do a different implementation with a different technology and not the same implementation that was made using REST.
5. **Budget**

As this project is a software development, the cost is not going to be as high as it was a hardware development but we are going to need some resources.

Now I am going to list an estimation of the cost of the necessary material that would be necessary for the development of the project.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asus X55LA-XX370H</td>
<td>Is an Intel i5 laptop that would be used as a pre server for the initial development.</td>
<td>479€</td>
</tr>
<tr>
<td>Lenovo ThinkServer TS140</td>
<td>Is a professional server with RAID compatible that would help us to create security copies of the database. It would be use as a pro server</td>
<td>599€</td>
</tr>
<tr>
<td>Salicru SPS SOHO+ 2000VA</td>
<td>Is a SAI that would be really helpful if there is any electrical failure.</td>
<td>199€</td>
</tr>
<tr>
<td>Netbeans IDE</td>
<td>With Java EE extension. It would be the development environment</td>
<td>Is free use. Is an open source development platform</td>
</tr>
<tr>
<td>MySQL workbench</td>
<td>It would be used to store the database.</td>
<td>Is free use. Is an open source platform</td>
</tr>
</tbody>
</table>

Table 13. Price table

These are the physical resources that would be necessary to implement a definitive version of the application. Now we are going to estimate the human cost as a junior engineer:

- More or less 22 weeks of work, approximately 5 month
- The total time invest in the project during the week: 25 h
- The average salary as a junior engineer 1 person: 10€/h

In order to finish both applications it would be good to work 2 persons in the project. More or less the same amount of time and also with two pre environments.

The estimated total cost would be: **12,756€**
Now we may think about how we would introduce the product to the market. In the previous assumption we have imagined that the project was made inside a company. So there would not be any commercial distribution.

We may assume that we have developed the application in order to be sold under a license, like a service. Comparing prices between other services that provide almost the same like Microsoft project online I decided that a good realize price would be 20€ user/month.
6. **Conclusions and future development:**

After seeing the structure of both applications and comparing the technologies we obtain the following conclusions as a result:

6.1. **Which is the best technology for the application?**

Very few differences exist between both applications, making it hard to answer the question, but for this particular one I would choose the WebSockets technology.

It is more efficient than the traditional polling method because with a unique TCP connexion the handshake times are not repeated in a loop, so the traffic of information becomes more agile. However, it also has a huge handicap that it has been already mentioned in the previous chapters, WebSockets technology is not mature enough, so lots of functions created for this particular application may refined by the developers in the future.

6.2. **Which technology will be more suitable in futures applications?**

Nowadays the most common technology used on web is REST because of its early beginnings, it has been able to perfection its technology during the years, creating a robust list of frameworks that allow the programmers to develop their own REST implementations easy and quick.

Nevertheless, my personal opinion is that WebSockets will be one of the most popular web development technologies in the future, because some Android applications are starting to use WebSockets technology for some programs. The reason why this is happening is because people want to access web content in their personal smartphones anywhere at any time.

To do that, these applications need to be as light as possible in terms of connectivity consumption, because in the end it is desirable to minimize the costs of internet connection. REST applications are not usually prepared to be light at all, they tend to be a bit heavy, and for a regular home connection this is not a big problem but it is for a smartphone connection.

In conclusion, based on what has been mentioned in previous chapters, WebSockets connections are more efficient than constant polling connections.

The application may have a plenty of possibilities for a future development, because the main idea was to bring the developers the necessary tools to make their own applications. For example an immediate improve would be to implement the resources module in the current application doing that, the users would be able to assign resources to the tasks or subtasks. This could be one example but with the tools provided as I said, the possibilities are almost endless.
Bibliography:
Most of the references are only used during the software development of the application.


[5] jQuery Ajax error handling, show custom exception messages - Stack Overflow stackoverflow.com


[10] Components · Bootstrap - getbootstrap.com


I have been also really active in StackOverflow, Bootstrap, JQuery and Java fórum around the internet, all of this references are not written because they are not really important for the understanding of the final thesis but all of them has been fundamental for the final development.
Appendices:

All the code is added in the attached .rar file, the file contains:

- SQL script, to build the schema and tables that are necessary for the project
- ProjectManagementSocket → Folder that contains all the code for the WebSockets application
- ProjectManagement → Folder that contains all the code for the REST application
- Gantt diagram of the project in PDF, generated with the application.

Work Packages and milestones:

Work Packages:

<table>
<thead>
<tr>
<th>Project: Documentación</th>
<th>WP ref: (WP1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: Reports</td>
<td>Sheet 1 of 8</td>
</tr>
<tr>
<td>Short description: Creación de cada uno de los reports del proyecto</td>
<td>Planned start date: 22/02/2015</td>
</tr>
<tr>
<td></td>
<td>Planned end date: 20/07/2015</td>
</tr>
<tr>
<td></td>
<td>Start event: Inicio de proyecto</td>
</tr>
<tr>
<td></td>
<td>End event: Fin de proyecto</td>
</tr>
<tr>
<td>Internal task T1: Project Proposal and Work Plan.</td>
<td>Deliverables:</td>
</tr>
<tr>
<td>Internal task T2: Project Critical Review drafting.</td>
<td>Dates:</td>
</tr>
<tr>
<td>Internal task T3: Degree Thesis drafting</td>
<td>D1 06/03/2015</td>
</tr>
<tr>
<td>Internal task T4: Thesis Lecture</td>
<td>D2 24/04/2015</td>
</tr>
<tr>
<td></td>
<td>D3 10/07/2015</td>
</tr>
<tr>
<td></td>
<td>D4 20/07/2015</td>
</tr>
<tr>
<td>Project: Creación de la base de datos SQL</td>
<td>WP ref: (WP2)</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Major constituent: Software</td>
<td>Sheet 2 of 8</td>
</tr>
</tbody>
</table>
| Short description: Programación de la base de datos SQL con los componentes fundamentales para la gestión de proyectos. | Planned start date: 17/02/2015  
Planned end date: 02/03/2015 |
|                                          | Start event: Inicio de proyecto |
|                                          | End event: Inicio programación API |
| Internal task T1: Estudio y creación de una base de datos preliminar de test. | Deliverables: |
| Internal task T2: Probar cada una de las funcionalidades de la base de datos. | Dates: |
| Internal task T3: Creación de la base de datos definitiva. | |

<table>
<thead>
<tr>
<th>Project: Estudio de funcionamiento de una API REST</th>
<th>WP ref: (WP3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: Recopilación de información</td>
<td>Sheet 3 of 8</td>
</tr>
</tbody>
</table>
| Short description: Recopilación de toda la información teórica posible sobre la tecnología, así como las herramientas para su implementación. | Planned start date: 02/03/2015  
Planned end date: 06/03/2015 |
<p>|                                          | Start event: Inicio de proyecto |
| Internal task T1: Estudio de la tecnología | Deliverables: |
| Internal task T2: Toma de contacto de las herramientas disponibles para el desarrollo. | Dates: |
| | | | | |</p>
<table>
<thead>
<tr>
<th>Project: Programación de la API REST</th>
<th>WP ref: (WP4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major constituent: Software</strong></td>
<td><strong>Sheet 4 of 8</strong></td>
</tr>
</tbody>
</table>
| **Short description:** Creación de todos los componentes de software necesarios. | **Planned start date:** 06/03/2015  
**Planned end date:** 25/03/2015 |
| **Start event:** Recopilación de información.  
**End event:** | **Internal task T1:** Implementación de las primeras versiones.  
**Internal task T2:** Testeo de las primeras versiones.  
**Internal task T3:** Corrección de errores.  
**Internal task T4:** Implementación de la versión final.  
**Deliverables:** D2  
**Dates:** | **End event:** |

<table>
<thead>
<tr>
<th>Project: Estudio de las herramientas de desarrollo</th>
<th>WP ref: (WP5)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major constituent: Recopilación de información</strong></td>
<td><strong>Sheet 5 of 8</strong></td>
</tr>
</tbody>
</table>
| **Short description:** Estudio de los lenguajes de programación más indicados para la realización de la aplicación web. | **Planned start date:** 25/03/2015  
**Planned end date:** 10/04/2015 |
| **Start event:** Programación de la API  
**End event:** | **Internal task T1:** Busqueda de información.  
**Internal task T2:** Toma de contacto con los lenguajes.  
**Deliverables:** D3  
**Dates:** | **End event:** |
<table>
<thead>
<tr>
<th>Project: Desarrollo de la aplicación web REST</th>
<th>WP ref: (WP6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: Software</td>
<td>Sheet 6 of 8</td>
</tr>
</tbody>
</table>
| Short description: Programación de la estructura de la aplicación web que hace uso de la API. | Planned start date: 10/04/2015  
Planned end date: 22/06/2015 |
|                                           | Start event: Estudio de las herramientas  
End event: |
| Internal task T1: Realización de los mockups (maquetas). | Deliverables: 
-- |
| Internal task T2: Implementación de la vista de usuarios. | Dates: |
| Internal task T3: Implementación del controlador de usuarios. | |
| Internal task T4: Implementación de la vista de proyectos. | |
| Internal task T5: Implementación del controlador de proyectos. | |

<table>
<thead>
<tr>
<th>Project: Desarrollo de la aplicación web WebSockets</th>
<th>WP ref: (WP7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: Software</td>
<td>Sheet 7 of 8</td>
</tr>
</tbody>
</table>
| Short description: Programación de la estructura de la aplicación web que hace uso de WebSockets. | Planned start date: 18/04/2015  
Planned end date: 22/06/2015 |
|                                           | Start event: Estudio de las herramientas  
End event: |
| Internal task T1: Reutilización de los mockups (maquetas). | Deliverables: 
-- |
| Internal task T2: Implementación de la vista de usuarios. | Dates: |
| Internal task T3: Implementación del controlador de usuarios. | |
| | |
| Internal task T4: Implementación de la vista de proyectos. |  |
| Internal task T5: Implementación del controlador de proyectos. |  |

<table>
<thead>
<tr>
<th>Project: Fase de evaluación de la aplicación</th>
<th>WP ref: (WP8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Major constituent: Testeo - Software</td>
<td>Sheet 8 of 8</td>
</tr>
</tbody>
</table>
| Short description: Se realizaran una serie de pruebas al software para calcular su eficiencia usabilidad y la posible detección de errores. | Planned start date: 22/06/2015  
Planned end date: 04/07/2015  
Start event: Desarrollo de la aplicación web.  
End event: |

| Internal task T1: Testeo por parte del programador.  
Internal task T2: Testeo por parte de terceras personas.  
Internal task T3: Comparativa de tecnologías. (REST vs WebSockets) | Deliverables:  
Dates: |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WP#</td>
<td>Task#</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>
Glossary

IT – Information technology
TCP – Transmission Control Protocol
AJAX – Asynchronous Javascript And XML
XML – eXtensible Markup Language
HTML – Hypertext Markup Language
CSS – Cascading Style Sheets
SQL – Structured Query Language
HQL – Hibernate Query Language
API – Application programming interface
WS - WebSocket
REST – Representational State Transfer
Java EE – Java Enterprise Edition
Java NIO – Java Non-blocking I/O
SOAP - Simple Object Access Protocol
WSDL - Web Services Description Language
IETF – Internet Engineering Task Force
RFC – Request for comments
DOM – Document Object Model