Time Tracking System with GWT

Amaia Carrillo Agirre
2010-06-03
Subject: Software Engineering
Level: Master
Course code: 5DV00E
Acknowledgements
I would like to thank and acknowledge all the people that have helped me in many different ways during the development of this project and in my entire life, making this thesis what it is and making me who I am:

- Rüdiger Lincke and Francisco Modesto, for giving me advice and support, and sharing their knowledge with me, because without their help I would feel lost in my own project more than once.

- My university friends in Barcelona, for helping me since I started my studies, even being 2500km far from them.

- All the people I have met in Växjö during my Erasmus period, for making these 5 months unforgettable. It would be impossible to name all of them, so special mention to Stallvägen 28 girls and my boys from the 8th floor.

- My parents, for making me who I am in an extent part, helping me in everything and supporting me in every decision I have taken.

- Garazi, my sister, because life without her would have been too boring.
Abstract
Applied Research in System Analysis (ARiSA) AB, is a company whose main objective is to market the VizzAnalyzer™ software quality tool and related services, experiences, and research results.[1]

In the beginning, recording the time spent on different activities and projects for project control, cost calculation, and customer billing manually and individually was sufficient and effective. With the number of projects and people involved at a time increasing the existing mechanisms were not sufficient anymore and needed to be replaced to support an efficient recording of time and report generation.

To solve this situation, a prototype of a web-based time tracking system was implemented. The task for this thesis was to build on the existing prototype implementation, to improve and complete it. In detail, to extend the functionalities of the web application adding user, project and activity management, and report creation.

The solution supports an efficient time tracking for employees and it makes the report generation easier.

This thesis describes the requirements for the project and the resulting implementation.
# CONTENT

## 1 INTRODUCTION

1.1 PROBLEM AND MOTIVATION ............................................................... 1
1.2 GOALS AND CRITERIA ........................................................................... 1
1.3 OUTLINE .................................................................................................. 2

## 2 BACKGROUND

2.1 NEW PROJECT SCENARIO .................................................................... 3
2.2 TIME TRACKING SYSTEMS ................................................................... 3
2.3 GOOGLE WEB TOOLKIT (GWT) ............................................................ 4
   2.3.1 AJAX ............................................................................................. 4
   2.3.2 Servlets .......................................................................................... 5
   2.3.3 GWT’s transparency ...................................................................... 5

## 3 REQUIREMENTS, ANALYSIS AND OVERALL DESIGN

3.1 USERS .................................................................................................... 7
3.2 USE CASES & MODEL .......................................................................... 7
3.3 NON-FUNCTIONAL REQUIREMENTS .................................................... 8
3.4 DATA MODEL ......................................................................................... 9
3.5 DESIGN MODEL .................................................................................... 10
3.6 COMMUNICATION PATTERN ............................................................... 11
3.7 ARCHITECTURE: 3 LAYERS ................................................................. 13

## 4 IMPLEMENTATION

4.1 UC1: USER MANAGEMENT ................................................................. 15
   4.1.1 Use Case Specification ................................................................. 15
   4.1.2 Requirements ............................................................................... 16
   4.1.3 Attributes ..................................................................................... 16
   4.1.4 Use Case Realization ................................................................... 17
   4.1.5 Classes Involved .......................................................................... 18
4.2 UC2: CATEGORY MANAGEMENT .......................................................... 24
   4.2.1 Use Case Specification ................................................................. 24
   4.2.2 Requirements ............................................................................... 24
   4.2.3 Attributes ..................................................................................... 25
   4.2.4 Use Case Realization ................................................................... 25
   4.2.5 Classes involved ........................................................................... 26
4.3 UC3: ACTIVITY MANAGEMENT ............................................................ 26
   4.3.1 Use Case Specification ................................................................. 27
   4.3.2 Requirements ............................................................................... 27
   4.3.3 Attributes ..................................................................................... 28
   4.3.4 Use Case Realization ................................................................... 28
   4.3.5 Classes Involved .......................................................................... 29
4.4 UC4: PROJECT MANAGEMENT .............................................................. 29
   4.4.1 Use Case Specification ................................................................. 29
   4.4.2 Requirements ............................................................................... 30
   4.4.3 Attributes ..................................................................................... 31
   4.4.4 Use Case Realization ................................................................... 31
   4.4.5 Classes Involved .......................................................................... 32
4.5 UC5: DELIVERABLE MANAGEMENT .................................................... 33
   4.5.1 Use Case Specification ................................................................. 33
   4.5.2 Requirements ............................................................................... 34
   4.5.3 Attributes ..................................................................................... 34
   4.5.4 Use Case Realization ................................................................... 35
   4.5.5 Classes involved .......................................................................... 35
Table list

Table 2.1: Time tracking systems' comparison ................................................................. 4
Table 3.1: Use-Case overview .......................................................................................... 8
Table 3.2: Non-functional Requirements ........................................................................... 9
Table 4.1: UC1's Functional Requirements ....................................................................... 16
Table 4.2: UC1's attributes .............................................................................................. 17
Table 4.3: Classes involved in UC1 .................................................................................. 18
Table 4.4: UC2's Functional Requirements ....................................................................... 25
Table 4.5: UC2's Attributes .............................................................................................. 25
Table 4.6: UC2's involved classes ..................................................................................... 26
Table 4.7: UC3's Functional Requirements ....................................................................... 27
Table 4.8: UC3's attributes .............................................................................................. 28
Table 4.9: UC3's involved classes ..................................................................................... 29
Table 4.10: UC4's Functional Requirements .................................................................... 30
Table 4.11: UC4's attributes ........................................................................................... 31
Table 4.12: UC4's involved classes ................................................................................... 33
Table 4.13: UC5's Functional Requirements .................................................................... 34
Table 4.14: UC5's attributes ........................................................................................... 35
Table 4.15: UC5's involved classes ................................................................................... 36
Table 4.16: UC6's Functional Requirements .................................................................... 37
Table 4.17: UC6's attributes ........................................................................................... 37
Table 4.18: UC6's classes involved ................................................................................... 38
Table 4.19: UC7's Functional Requirements .................................................................... 39
Table 4.20: UC7's attributes ........................................................................................... 40
Table 4.21: UC7's involved classes ................................................................................... 41
Table 4.22: UC8's Functional Requirements .................................................................... 42
Table 4.23: UC8's attributes ........................................................................................... 42
Table 4.24: UC8's involved classes ................................................................................... 43
Table 4.25: UC9's Functional Requirements .................................................................... 44
Figure list

| Figure 3.1: Use Case Model                                      | 7 |
| Figure 3.2: Data Model                                         | 9 |
| Figure 3.3: Class diagram                                      | 11 |
| Figure 3.4: Client side sequence diagram                       | 12 |
| Figure 3.5: Server side sequence diagram (part 1)              | 12 |
| Figure 3.6: Server side sequence diagram (part 2)              | 13 |
| Figure 3.7: 3 layers architecture                              | 14 |
| Figure 3.8: Layers interaction                                 | 14 |
| Figure 4.1: UC1’s screenshots                                  | 17 |
| Figure 4.2: "Add user” requirement’s sequence diagram          | 19 |
| Figure 4.3: "Create user” button’s event (TimelogUsersView.java) | 20 |
| Figure 4.4: Part of the creation of the interface(TimelogUsersView.java) | 20 |
| Figure 4.5: Setting the form (grid) visible (TimelogUsersView.java) | 20 |
| Figure 4.6: Asynchronous call (TimelogUsersView.java)          | 21 |
| Figure 4.7: Filling the DTO (TimelogUserView.java)             | 21 |
| Figure 4.8: Registry of services (Services.java)              | 21 |
| Figure 4.9: "Add" part of the UserHandler.java                 | 22 |
| Figure 4.10: UserDao.java code for adding a user              | 22 |
| Figure 4.11: UserDto definition (UserDto.java)                 | 23 |
| Figure 4.12: UserAction.java, the definition                   | 23 |
| Figure 4.13: UserResponse.java, definition part                | 23 |
| Figure 4.14: UC2's screenshot                                  | 26 |
| Figure 4.15: UC3's screenshot                                  | 28 |
| Figure 4.16: UC4's screenshots                                 | 32 |
| Figure 4.17: UC5's screenshot                                  | 35 |
| Figure 4.18: UC6's screenshot                                  | 38 |
| Figure 4.19: UC7's screenshot                                  | 41 |
| Figure 4.20: UC8's screenshot                                  | 43 |
1 Introduction

In many industries, and in particular in the service industry, internal and external cost estimation, planning, reporting, payment of salaries, and customer billing rely on the exact recording of time and effort spent. Recording time needs to be efficient and flexible, and with a minimum of overhead added.

Applied Research in System Analysis (ARiSA) is an IT consulting company. Their goal is to leverage quality management on software and information systems engineering. They provide seminars, methods, and development tools supporting the efficient production and maintenance of high quality software.

1.1 Problem and Motivation

In most of the projects ARiSA is involved in, customers get billed based on consulting time. Therefore, an exact accounting of the time spent on a project is important.

Until recently, employees recorded the time spent on projects in personal time tracking sheets. With the number of projects and employees growing, it was more and more difficult to get an overview of the time spent. The effort required to update the individual time sheets with new tasks and activities was increasing as the time needed for creating reports based on the distributed reports.

To improve on the increasingly unsatisfactory situation ARiSA started developing a web based application. Its purpose is to replace the current sheet based time tracking with an improved that is efficient, flexible, and low overhead solution.

The problem addressed by this thesis is to complete and extend the existing prototype to a fully functional implementation. This is a difficult problem because it involves a complex data model. In addition, the infrastructure used at ARiSA involves Windows, Mac OS, and Linux based computers with a wide variety of web browsers used. Therefore the final solution should be operational in an inhomogeneous environment.

1.2 Goals and Criteria

This section describes the goals pursued by this thesis in order to solve the problem and the criteria used for validating the goal:

- The main goal is to provide means for efficiently collecting time spent on a particular task. This goal will be reached when every user of the system can record date, time (from, to), task, and activity in an efficient way. Efficient means it should just take a few seconds for creating and saving a record. Additionally, it shall be possible to edit or delete existing records.
- The second goal is to be able to use the time tracking application without the need for installing extra software from everywhere, where Internet access is available. This goal is reached if a web application, which can be used by most of the currently available web browsers, is created.
- The third goal is to implement features allowing generating detailed reports based on the recorded time. This goal is reached if detailed reports can be generated based on the recorded time allowing specification of time, project, user, activity, etc., e.g., for bills or project control and cost calculation.
- The final goal is to allow user management and the efficient specification of projects, deliverables, and activities. This goal is reached if authorized users can create, modify or delete information such as project, users, activities, etc.
1.3 Outline

Chapter 2 contains more detailed information about the time recording workflow at ARiSA. In addition, used technologies and tools are discussed in general.

In Chapter 3 the requirements, architecture and design of the overall solution are presented.

Chapter 4 discusses the implementation of the design per Use-Case.

Finally, in Chapter 5 we will summarize the conclusions we have taken while developing this application and we will talk about how it can be improved or expanded.
2 Background
To make the problem of recording time more vivid we will describe a typical scenario when starting a new project. Then, discuss the technologies used for implementing the solution.

2.1 New Project Scenario
A customer makes a request, which is written in a word document that later on will be shared with the rest of the working team. In the document, all the requested requirements are specified as well as the project’s (resulting) deliverables, the activities for creating them, the estimated time, the budget and other information.

Once the team has been informed about the new project, they start working on it or they continue with other on-going projects. Each member of the team has an individual time tracking spreadsheet per project, where they manually take notes of the time they spend working on different activities of each deliverable of the project.

After a certain period of time (usually every month), these spreadsheets are collected and put all together manually, in other words, every employee’s spreadsheet about a concrete project is merged in a common one, and the same is done for every different project. The result will be used to create the cost calculation and invoices, which will show the total cost as well as the breakdown of each activity or deliverable. At the same time, the merged spreadsheet is useful for the verification of previous time and cost estimation, and for the internal personnel control.

Finally, apart from cost estimations or invoices, other kinds of reports are also created in order to have more information about the evolution of the project during its development or the work of each employee, etc. These documents are made manually and printed in a “pdf” format in most of the cases, but they can also be spreadsheets or word documents.

Due to an increasing number of projects, employees involved, and change requests it became more and more difficult to keep the activity and deliverable lists in each spreadsheet up to date. For example updating several spreadsheets efficiently with new deliverables was a problem. Also, building the final reports by combining the information in the spreadsheets took more and more time.

For these reasons, ARiSA started implementing a web-based solution which has been the starting prototype of this thesis. It was only focused on the main requirement of the company, the time tracking. The interface was completely developed and the database had been started, but it only contained the necessary information for the concrete purpose. However, the database connection was only implemented for the requests, and not to create, edit and delete info.

In conclusion, the process to reach the main goal of the thesis was already started, but not finished, and all the rest needs were still to be developed.

2.2 Time Tracking Systems
To address the limitations of the current spreadsheet based solution alternatives were investigated which satisfy the criteria stated in Section 1.2.

There are several time tracking systems in the market nowadays, all with different settings. Some of them are made for only a specific task, while others cover many more computer based fields of a company. In platform matters, some are web-based while others are operating system dependent or specific for a device, for example iPhone. There are also different licensing needs, from free open-source to subscription or license based.
We have tried to analyse many of these products but they were not exactly what the company required. In Table 2.1, we compare a group of pre-selected products. It has been impossible to analyse all the existing products, but from the most notable ones, we have chosen those that fulfil at least one of the requirements and so those which seem to fit most with the specified goals. The characteristics we are comparing clear, on one hand, if the solution will be available in any of the operative systems used in the company, in other words, in any of the existing browsers. On the other hand, let us find out if it fulfils all the needs the company has, without overloading the system with too many features.

### Table 2.1: Time tracking systems' comparison

<table>
<thead>
<tr>
<th>Product name</th>
<th>Platform</th>
<th>Just necessary features</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>iTimesheet[2]</td>
<td>No</td>
<td>Yes</td>
<td>Only available for iPhone, iPod touch and iPad</td>
</tr>
<tr>
<td>dotproject[3]</td>
<td>Yes</td>
<td>No</td>
<td>Volunteer supported project management application with too many features</td>
</tr>
<tr>
<td>Grindstone[4]</td>
<td>No</td>
<td>Yes</td>
<td>Microsoft Windows required</td>
</tr>
<tr>
<td>24SevenOffice[6]</td>
<td>Yes</td>
<td>No</td>
<td>Complete ERP/CRM</td>
</tr>
</tbody>
</table>

As a conclusion of Table 2.1, we can extract that there is no product in the market completely fulfilling ARiSA’s needs. Due to this fact, the decision of developing a custom made application was taken. This application would successfully fill both kinds of requirements and so all the goals would be achieved.

### 2.3 Google Web Toolkit (GWT)

For the development of the web-based application, we will use Google Web Toolkit, called from now on GWT. The information for both the development of the application and this section has been taken from [7], [8] and [9].

GWT is a powerful open source tool for building and optimizing complex browser-based applications. It provides all the widgets and Java APIs to perform anything that can be done in DOM or JavaScript, as it has been done usually until now. It allows writing AJAX applications that can be run in any browser or mobile operative system by encoding to highly optimize JavaScript. It can be developed in many IDEs and executed in any browser. Many tools, like Firebug or Inspector, can be used as you code in Java due to the fact that the developing mode is in the browser itself.

One advantage of GWT is the powerful engines it has for optimization, for instance, the remove of dead code, in-lined methods and strings optimization. However, the most important advantage is that it allows developers to create one (complex and high-performance) web application that can run in most of the standard browsers without the need to adapt them to browser specific problems or requirements. The compiler does this job. In the following sections, will get more in detail on how this works.

#### 2.3.1 AJAX

At the beginning web pages were completely static and there was no interaction with the server. Each web page’s content was the fixed, independently of the user that was accessing or the context. They evolved later in dynamic web pages, where depending on the user, the conditions or the context, the information shown on them varies because
Asynchronous JavaScript and XML (AJAX) is the technique used for creating fast and dynamic web pages by making possible the update of only some parts of the page. Every time the user triggers an event, such as clicking a button, the necessary data is sent. Later on, just the needed data is fetched from the server as a response to the submitted actions, which will be used to repaint only the sections of the page that has to be changed. This procedure is done asynchronously, allowing the user continue performing other tasks in the browser while the data is returning.

The keystone to carry out this procedure is the object call XMLHttpRequest (XHR). Usually the JavaScript invokes communication to the server through it, and it is used to address the problems that can be found in this communication. Some examples of these problems are, for instance, the ability to get and set HTTP headers or to handle different kind of content produced by the server. It has to be mentioned anyway, that not all the problems that can be found in the link are solved with XHR. However, as it is not the aim of this thesis to go into detail in the AJAX technique we will not continue further on, but more information can be found in [10].

2.3.2 Servlets

According to Sun Microsystems [11], Java developer, a servlet is a small Java program that runs within a Web server and receives and responses to request from Web clients, usually HTTP (Hypertext Transfer Protocol). It implements different methods in which the most important ones are those which define its life cycle.

First of all, a servlet is created and initialized by the init method. During the system execution, any call to the service method is handled until the servlet is taken out of service. Once is not going to be used anymore, the destroy method destroys it, is collected by the garbage collector and in the end finalized.

It can be thought as a non-face applet that is run on the server-side (from there its name). It gives the developers a simple and consistent mechanism for, in one hand, extending the functionality of a Web server and on the other hand, accessing existing business systems.

2.3.3 GWT’s transparency

After all the explanation about how a dynamic web page with AJAX technique is usually developed, we will describe which the differences are with the GWT framework, because even if the result is quite similar, the internal implementation is much simple.

GWT has a mechanism that eases the application to make a call to the server-side code and get the data. This mechanism is a Remote Procedure Call (RPC) and is one of the most important pieces of the GWT, due to the fact that it hides all the plumbing necessary to create and consume a service. A service is the part of the code resident in the server-side that is called by the client and needs to receive and send data from this side, which usually is lightly complicated. However, in GWT technology all that has to be done is to define a service interface and program its implementation in the server-side.

In addition, this connexion between both sides and all network operations are done asynchronously, otherwise, as most browser’s JavaScript engines are mono-thread, the browser will look frozen during the execution, as we have said before. To implement
this asynchrony, for instance, a callback method is required to handle any possible result the service returns.

Moreover, it is important to mention that when compiling a GWT project, a JavaScript is generated and not a bytecode. Because of this fact, object communication needs some help to bridge the language division. This division happens between the client and server sides, as well as JavaScript and the Java code of the browser, and object serialization has to be done, which is provided by GWT technology, as is different from the Java serialization.

Finally, we can resume that due to all this plumbing that GWT makes it completely transparent to the developer; it offers the possibility to develop AJAX web-based applications without any AJAX knowledge, just Java programming language control.

At the same time, there is no need of worrying about each browser’s specification. Without using this technology, different applications have to be developed for different browsers. However, with GWT, the compiler makes it possible to develop an application once, which may be running in any of the main browsers.
3 Requirements, Analysis and Overall Design

This chapter documents the requirements, analysis, and overall design of the Time Tracking System we develop. We will go from the users to a general view of the use cases, continuing with the non-functional requirements and the data and design model. Moreover, we will give some theoretical information about the architecture we have used.

3.1 Users

There are two different users, depending on their privileges. As it can be seen in Figure 3.1 the main user, an employee, can manage time tracks and reports, as well as projects and deliverables, and of course will be able to log in and out of the system. Nevertheless, there is an especial kind of employee, super user or administrator, who has more privileges and is the person in charge of managing categories, activities and users.

3.2 Use Cases & Model

In this section we present the main use-cases of the system. Figure 3.1 shows the use-cases and their relation to the users in the system. Table 3.1 lists the use-cases and provides a brief description.

![Figure 3.1: Use Case Model](image-url)
### Table 3.1: Use-Case overview

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User management</td>
<td>The super user can manage the users of the system. This includes displaying a list of the users in the system, adding new users, editing and removing existing users.</td>
</tr>
<tr>
<td>Category management</td>
<td>The super user can manage the categories of the activities. This includes displaying a list of the categories existing in the system, adding new categories, editing and removing existing categories.</td>
</tr>
<tr>
<td>Activity management</td>
<td>The super user can manage the activities. This includes displaying a list of the activities that belongs to a concrete category existing in the system, adding new activities, editing and removing existing activities.</td>
</tr>
<tr>
<td>Project management</td>
<td>Any user can manage the projects of the system. This includes displaying a list of the project he/she is working on, adding new projects, editing and removing existing projects.</td>
</tr>
<tr>
<td>Deliverable management</td>
<td>The owner of a project can manage the deliverables of the project. This includes displaying a list of the deliverables related to one project, adding new deliverables to a project, editing and removing existing deliverables.</td>
</tr>
<tr>
<td>Time track management</td>
<td>A user can manage his/her personal time track. This includes displaying the time track registered until that moment, starting and finishing new tracks, editing and removing existing tracks.</td>
</tr>
<tr>
<td>Reporting</td>
<td>A user can display reports (bills or memories) about any of the projects or deliverables he/she is working on.</td>
</tr>
<tr>
<td>Log in</td>
<td>All users need to log in before they can do anything else in the system.</td>
</tr>
<tr>
<td>Log out</td>
<td>All users can log out at any moment and from anywhere in their profile.</td>
</tr>
</tbody>
</table>

3.3 Non-Functional Requirements

In Table 3.2 we will find the non-functional requirements of the system, all of them with a brief description.
### Table 3.2: Non-functional Requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appearance</td>
<td>The graphic design of the system has to be the same in the whole application.</td>
</tr>
<tr>
<td>Intuitive</td>
<td>The user may be able to use all the system’s functionality in an intuitive way</td>
</tr>
<tr>
<td>Platform dependency</td>
<td>The application should work correctly in the main browsers</td>
</tr>
<tr>
<td>DB consistency</td>
<td>The new creations, changes or deleting operations should be recorded on the database at the moment.</td>
</tr>
<tr>
<td>Multi user</td>
<td>The system should allow the interaction of two or more users at the same time.</td>
</tr>
<tr>
<td>Pre-installation</td>
<td>The user does not need to install anything before visualizing the application.</td>
</tr>
</tbody>
</table>

### 3.4 Data Model

The data model diagram (Figure 3.2) below is a simplified version of the whole diagram, due to the fact that only the classes, attributes and associations are shown. It generates a visual idea of which data is going to be necessary to store and manage, helping also the better understanding of the system.

![Data Model Diagram](image)

**Figure 3.2: Data Model**

To describe Figure 3.2 the explanation will start from the User class. It is defined by the id, but also the name, which is unique, is used as an identifier. Apart from the password,
which is used for the identity checking, the rest of the attributes are additional information.

A user can participate in several projects with more other users. Each project is identified by the id and contains a lot more information, for instance, the customer who asked for it or the creation date. It has a category type, in order to classify it in some way and it can be composed by several deliverables.

A deliverable has information related to the project it belongs and in case it is a sub deliverable which its father is. It is identified by the id and also with a unique name, and it stores more data, such as a short name, a deadline and the rest of the attributes associated.

A category is a class identified by the id and described by a name and a short name. Each of the categories contains many activities in order to classify these. An activity has the same attributes as a category, plus the cost.

With all this structure, tasks to track the time and reports that show the tracked information can be defined. The first ones are defined by the user that executes an activity working on the deliverable of one project between the start and the end moment. They are defined by ids, but they have additional information as a description or links.

The seconds, the reports, show which user has generate which type of report about which project or deliverable asking for information during a concrete period of time. They are also identified by an id.

### 3.5 Design Model

In this section we will show a general class diagram (Figure 3.3) which shows how all the classes of the same kind interact with the others. Here we are going to describe in detail what each class is in order not to repeat it every time in all the use cases. As they all follow the same pattern, when explaining each use case specifically, we will just explain the mapping between them. For this reason the class and function names of this diagram are not real, but a generalization.

The view class represents all the interface classes, where the interface appearance and the user’s possible actions are defined. This class interacts with the DTO (Data Transfer Object) class used to transfer data between the client and the server sides and it contains all the necessary information taken from the interface to execute the request. (More detailed information about the DTO will be given in 3.7).

The RPCService is a unique class that contains all the plumbing code for the correct connexion between both sides. It matches each action and response with the correspondent handler by consulting the service registry class. The action class is the one that contains the information that comes from the interface, while the response class is the one in charge of delivering the answer from the database. On the other hand, the handler’s job is to get the information from the action, execute the request and places the result in the response.

Finally, we have the DAO (Data Access Object) class and the entity. The first one makes the query to the database and typically has add, edit, delete and request functions, but can also contain many other different ones, depending on the needs. The result obtained from the query is placed in an entity class and will be consulted by the DAO in order to create the result. (If more information about a DAO is needed, go to 3.7).
Figure 3.3: Class diagram

3.6 Communication Pattern

Once we have already analysed the class diagram, we would like to explain how they interact between them. For the same reason as before, instead of explaining it with one diagram per use case, we will use the general idea to explain it in detail. An example will be given for the first use case and both patterns could be used for any use case.

We will start analysing the client side. The user generates an event or executes an action, for instance clicking a button, in the interface which is picked up by the view. This creates a DTO with all the necessary information for the realization of the task and it is sent inside the correspondent action and with the correspondent response to the RPCService interface. While this request is carrying out, the view will continue active. Once the asynchronous execution is done, the view will ask the response for the result, if there is any. This response is returned thanks to the AbstractCallback class and the data from the result will use as it wants. The described sequence can be seen in Figure 3.4: Client side sequence diagram.
Next, we will talk about the server-side. However, and due to the fact that the sequence diagram would be too big, we have divided it into two parts: the RPC procedure and the developer’s implementation. As we have said before, all the plumbing is so transparent that the developer’s only need is to make an execute call, which later on will end up processing the code implemented by the user in the handler, as shown in Figure 3.5. The RPC will check in the service registry which handler belongs to the action it has been asked to carry out and will give it the control.

To finish describing this process, we will expose how the user’s request arrives to the data layer (Figure 3.6). In this execution, the handler takes the necessary data, if needed, from the DTO the action offers. The handler will execute the appropriate code in the DAO, which will make the SQL call to the database. At this point, the DB will return an entity from the correspondent class or a simple type of data, which will be prosecuted in
the DAO and returned to the handler. Before giving the control back to the RPC, the result will be loaded in the response in order to be accessible for the view.

![Server side sequence diagram (part 2)](image)

3.7 Architecture: 3 Layers

Our system will be implemented with the 3 layers architecture, which layers are the client-side or interface, the server-side and the database (Figure 3.7). We will explain the 3 of them separately and how they interact between them.

The GUI (Graphical User Interface) is the “top layer”, the outside part of the system. It involves every page of the web-based application which the user will interact with, so it can be defined as what the user can see. The interface has to be intuitive and easy to work with and has to follow the same structure or appearance in all the pages. The user will express his/her wishes to the system by clicking buttons or icons, and the system will answer by showing the requested information or executed the requested command. In our case, it has been implemented with the GWT tool, using HTML for the content and CSS for the appearance.

The next level is the domain layer, also called the object layer. It is located in the server side and it is the layer that manipulates all the data. This layer is considered the core of the system, due to the fact that it is the link between the highest and the lowest level. It has knowledge about everything that is happening in the system, and this knowledge can be divided in two types. On one hand, there is the runtime data which is the data that is being used at a concrete moment. On the other hand, it knows all the structural information about both process and data structure. For the implementation of this part, we have chosen the Java programming language because is the one that works with GWT and because of its wide range of libraries.

Finally, in the lowest level, we can find the data layer which contains all the storage data, in other words, the database. It is the layer in charge of the persistency and in which all the new information will be saved and the consults will be done using SQL. In this case, we have chosen MySQL for the database implementation and Apache Tomcat for the server.
The main advantages of this architecture lie in the divide and conquer strategy, which makes it easier to handle three simple layers than a complex one. Moreover, it eases the reusability and the better understanding of the system, which is always recommendable for a possible expansion or reorganization.

However, data transfer between layers can become a problem. Differences in the programming language or transfers in the net can cause chaos. For that reason especial data objects are used, DTO and DAO (Figure 3.8: Layers interaction).

To transfer information from the GUI to the domain layer can provoke several calls to the system. In order to save them, a Data Transfer Object can encapsulate all the needed information and with a single call will be enough. When the user requests information from the system, the same happens. The Transfer Object created by the system will be populated with the attribute’s values and pass by value to the interface. The information has been taken from [12].

In the case of the access to the database, the problem remains in the different database structures existing nowadays. For this reason, Data Access Objects (DAO) handle the connection with the data source, managing both the obtaining and the storing of data. This pattern has been described in 46[13].
4 Implementation

This chapter documents the implementation of the overall design described in the previous chapter. It does this Use-Case by Use-Case, first presenting a more detailed Use-Case specification and then a Use-Case realization including the classes involved. Screenshots of the final implementation are provided.

4.1 UC1: User Management

In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3. Moreover, the Use-Case realization offers some concrete examples that can be used for the rest of the use cases.

4.1.1 Use Case Specification

In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario

1. The user is logged as “admin” so the “Users” view is available.
2. The user clicks in “Users” flag and this is displayed.
3. The super user has the following options:
   a. Press [Create new user] button.
      i. An empty information chart is shown.
      ii. The super user enters the information of the new user.
      iii. The user presses the [Save] button. (E1)
      iv. The system adds the new user to the system after checking for duplicates, invalid characters, password and password confirmation match, etc. (E2)
      v. The new user’s data is displayed in a non-editable way.
   b. Select a user from the list of existing users.
      i. The user’s information is displayed.
      ii. The user has the following options.
         1. The user presses [Delete] button.
            a. The user is removed.
            b. Back to point 2.
         2. The user presses [Edit] button.
            a. The information is editable and the user changes the information that wants to change.
            b. The user pressed [Save] button. (E3)
            c. The system updates the information after checking for duplicates, invalid characters, if the passwords match, etc. (E4)
            d. Back to point 3.b.i.
      3. The user presses [Back] button.
         a. Back to point 3.

Extensions

E1. The user presses [Cancel] button.
   - End of the use case.
E2. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 3.a.ii
E3. The user presses [Cancel] button.
   - End of the use case.
4.1.2 Requirements

The functional requirements this use case must satisfy are the following (Table 4.1):

<table>
<thead>
<tr>
<th>FR</th>
<th>Description</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1</td>
<td>Add a user</td>
<td>A new user will be added to the system.</td>
<td>The user is not already in the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A new user will be added to the system and it will be able to connect to the system and work with it.</td>
</tr>
<tr>
<td>FR2</td>
<td>Edit a user</td>
<td>An existing user’s information has to be changed.</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The new information is stored.</td>
</tr>
<tr>
<td>FR3</td>
<td>Delete a user</td>
<td>An existing user will be deleted from the system.</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The user will be erased from the system and will not be able to connect to the system anymore.</td>
</tr>
<tr>
<td>FR4</td>
<td>Visualize user’s data</td>
<td>The specified user’s information will be displayed on the screen.</td>
<td>None.</td>
</tr>
</tbody>
</table>

4.1.3 Attributes

In the following table, Table 4.2, all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.
Table 4.2: UC1’s attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>Name</td>
<td>Signing name of the user. Usually a short form. It is unique. Admin is the</td>
</tr>
<tr>
<td></td>
<td>default super user account.</td>
</tr>
<tr>
<td>Password</td>
<td>Signing password of the user.</td>
</tr>
<tr>
<td>Complete Name</td>
<td>Name of the user. Typically first and last name.</td>
</tr>
<tr>
<td>Email</td>
<td>Email of the user. Used for communication, etc. Lost password recovery.</td>
</tr>
<tr>
<td>Telephone number</td>
<td>Phone number of the user, usually mobile phone.</td>
</tr>
</tbody>
</table>

4.1.4 Use Case Realization

In this section we will, on one hand, show screenshots of how each use case interface looks like, and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.

Figure 4.1: UC1’s screenshots
Figure 4.1 shows the typical screenshots from “User Management” use case. In the first one, we can see the starting point where the super user chooses whether to create a new user or to visualize an existing user’s information. In the second screenshot, we can see the form displayed in case the user wants to create a new user; and finally in the last screenshot, the information of an existing user is already shown and the super user can choose between editing and deleting it.

With this interface, all the 4 requirements of this use case are fulfil at most in 4 clicks and the necessary time to fill the forms required wherever the user is in the system. Moreover, all the necessary information can be collected from the chart in order to fill all the attributes properly.

4.1.5 Classes Involved

As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.3). Moreover, an example sequence diagram is presented. It shows the implementation of the “add user” requirement and it continues with an explanation in where the most important parts of the code are shown.

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogUsersView</td>
</tr>
<tr>
<td>DTO</td>
<td>UserDto</td>
</tr>
<tr>
<td>Action</td>
<td>UserAction</td>
</tr>
<tr>
<td>Response</td>
<td>UserResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>UserHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>UserDao</td>
</tr>
<tr>
<td>Entity</td>
<td>User</td>
</tr>
</tbody>
</table>
Figure 4.2: "Add user" requirement's sequence diagram
As an example of the behaviour of the system, Figure 4.2 shows the sequence diagram of “Add user”. First of all, the user decides to create a new user by clicking in a button and the system catches the event Figure 4.3.

```java
private void addEvents() {
    create.addClickHandler(new ClickHandler{
        public void onClick(ClickEvent event) {
            setCreationMode();
        }
    });
}

Figure 4.3: "Create user" button's event (TimelogUsersView.java)
```

When executing the event, the form to create a new user will be displayed. The form has been created previously. It is composed by a label and a text or suggestion box per each attribute required placed in a grid, as shown in Figure 4.4. However, it will be visible from that moment on (Figure 4.5, line 225).

```java
grid.setText(1, 0, "User's complete name");
completeName = new TextBox();
grid.setWidget(1, 1, completeName);
name = new TextBox();
grid.setText(2, 0, "User's nickname");
grid.setWidget(2, 1, name);
grid.setText(3, 0, "Password");
password = new PasswordTextBox();
grid.setWidget(3, 1, password);

Figure 4.4: Part of the creation of the interface(TimelogUsersView.java)
```

```java
protected void setCreationMode() {
    mode = "CREATION";
    Label l = new Label();
    l.setText("Create a new user");
    l.setStyleName("subtitle");
    grid.setWidget(0, 0, l);
    setEditCreate();
}

protected void setEditCreate() {
    save.setVisible(true);
    editDelete.setVisible(false);
    cancel.setText("Cancel");
    startingPanel.setVisible(false);
    grid.setVisible(true);
    setReadOnly(false);
}

Figure 4.5: Setting the form (grid) visible (TimelogUsersView.java)
```
Once the user has filled the form and after clicking in the save button another event will be caught in the same way as the previous one. After checking the correctness of the introduced data (Figure 4.6, line 271), a DTO will be created (Figure 4.6, lines 274 and 275 and Figure 4.7) and the asynchronous call will be done to the service (Figure 4.6, line 277). When creating the DTO (a serializable class), the information will be taken from the form as it can be seen in Figure 4.7.

```java
protected void addNewUser()
{
    if (!validateData())
        return;

    UserDto user = createUser();

    service.execute(new UserAction("ADD", user), new AbstractCallback<UserResponse>() {
        public void onCompleted(UserResponse result) {
            user.setId(result.getId());
        }
    });
    setInfoMode();
}
```

Figure 4.6: Asynchronous call (TimelogUsersView.java)

```java
private UserDto createUser() {
    user.setCompleteName(completeName.getValue());
    user.setName(name.getValue());
    user.setPassword(password.getValue());
    user.setEmail(email.getValue());
    user.setPhone(phone.getValue());
    return user;
}
```

Figure 4.7: Filling the DTO (TimelogUserView.java)

After the call to the RPCService, as shown in the sequence diagram, the correspondent handler (found thanks to the service registry, Figure 4.8) is called. It will be the class in charge of taking the information coming from the presentation layer, delegate to the DAO the call to the database and set the result in the response class (Figure 4.9, line 21).

```java
public class Services {
    public static final DaoFactory factory;
    public static final Registry registry;

    static {
        factory = new DaoFactory(new HibernateSessionFactory()) {
            public Session getSession() {
                return HibernateUtil.getSession();
            }
        };

        registry = new Registry();
        registry.addHandler(UserAction.class, new UserEventHandler());
    }
}
```

Figure 4.8: Registry of services (Services.java)
public class UserHandler implements ActionHandler<UserAction, UserResponse> {
    private final UserDao users = Services.factory.create(UserDao.class);
    @Override
    public UserResponse execute(UserAction action) {
        long id = 0;
        ArrayList<UserDto> list = new ArrayList<UserDto>();
        if (action.getType().compareTo("ADD") == 0)
            id = users.addUser(action.getUser()).getId();
        else if (action.getType().compareTo("CHANGE") == 0)
            id = users.modifyUser(action.getUser()).getId();
        else if (action.getType().compareTo("DELETE") == 0)
            id = users.deleteUser(action.getUser()).getId();
        else if (action.getType().compareTo("REQUEST") == 0)
            list = users.listUsers();
        return new UserResponse(id, list);
    }
}

Figure 4.9: "Add" part of the UserHandler.java

In the DAO class the user entity is created and set with the corresponding information. The call to the database is done and the entity is returned to the DAO (Figure 4.10).

@Transactional
public User addUser(UserDto userDto) {
    User user = new User();
    fillUser(user, userDto);
    session().save(user);
    return user;
}

private void fillUser(User user, UserDto userDto) {
    user.setCompleteName(userDto.getCompleteName());
    user.setName(userDto.getName());
    user.setPassword(userDto.getPassword());
    user.setEmail(userDto.getEmail());
    user.setPhone(userDto.getPhone());
}

Figure 4.10: UserDao.java code for adding a user

The returned information, the new user’s id, will be set in the UserResponse by the handler as can be seen in Figure 4.9 line 29. Afterwards, the AbstractCallback in the view will take it and set it to the DTO (see line 279 of Figure 4.6).

To finish the example, parts of the classes that have not been shown before are displayed for a better understanding (Figure 4.11, Figure 4.12 and Figure 4.13.)
```java
package se.softwerk.timelog.common.data;

import java.io.Serializable;

public class UserDto implements Suggestion, Serializable {
    private static final long serialVersionUID = 1911160043392355450L;
    long id;
    String completeName;
    String name;
    String password;
    String email;
    String phone;

    public UserDto() {
        // For hibernate
    }

    public String getCompleteName() {
        return completeName;
    }

    public void setCompleteName(String completeName) {
        this.completeName = completeName;
    }
}
```

**Figure 4.11: UserDto definition (UserDto.java)**

```java
public class UserAction extends SessionAction<UserResponse> {
    private static final long serialVersionUID = 503412006468736333L;
    private String type;
    private UserDto user;

    public UserAction() {
        // For serialization purpose
    }

    public UserAction(String type, UserDto user) {
        this.type = type;
        this.user = user;
    }
}
```

**Figure 4.12: UserAction.java, the definition**

```java
public class UserResponse implements Response {
    private static final long serialVersionUID = -6779204303733373339L;
    private long id;
    private ArrayList<UserDto> users;

    UserResponse() {
        // For serialization purpose
    }

    public UserResponse(long id, ArrayList<UserDto> list) {
        this.id = id;
        this.users = list;
    }
}
```

**Figure 4.13: UserResponse.java, definition part**
4.2 UC2: Category Management
In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

4.2.1 Use Case Specification
In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario
1. The user is logged as “admin” so the “Categories & Activities” view is available.
2. The user clicks in “Categories & Activities” flag and this is displayed.
3. All the existing categories are shown in a table.
4. The super user has the following options:
   a. The user fills the chart with information of a new category and press [Add] button. (E1)
      i. The system adds the new category to the system after checking for duplicates, invalid characters, etc. (E2)
      ii. The new category is added to the existing categories table.
   b. Presses [Delete] icon of an existing category:
      i. The category is removed.
   c. Presses [Edit] icon of an existing category:
      i. The row with the category’s information becomes editable.
      ii. The user changes the information that wants to change.
      iii. The user presses [Save] icon. (E3)
      iv. The system updates the information after checking for duplicates, invalid characters, if the passwords match, etc. (E4)
      i. The activities belonging to the selected category are displayed.
   e. Presses [Minus] icon of an existing category.
      i. The activities belonging to the selected category will be hidden.

Extensions
E1. The user fills the chart and press [Clear] button.
   - Back to point 2.
E2. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 4.a
E3. The user presses [Cancel] icon.
   - There is no change in the category and goes back to point 2.
E4. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 4.c.ii

4.2.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.4):
### Table 4.4: UC2's Functional Requirements

<table>
<thead>
<tr>
<th>FR</th>
<th>Description</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR5</td>
<td>Add a category</td>
<td>A new category will be added to the system.</td>
<td>The category is not already in the system.</td>
</tr>
<tr>
<td>FR6</td>
<td>Edit a category</td>
<td>An existing category’s information has to be changed.</td>
<td>None.</td>
</tr>
<tr>
<td>FR7</td>
<td>Delete a category</td>
<td>An existing category will be deleted from the system.</td>
<td>The category will be erased from the system and will be disconnected from every project and activity.</td>
</tr>
<tr>
<td>FR8</td>
<td>Visualize category’s data</td>
<td>The specified category’s information is displayed in one of the rows of the Category table.</td>
<td>None.</td>
</tr>
</tbody>
</table>

#### 4.2.3 Attributes

In the following table (Table 4.5) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

### Table 4.5: UC2's Attributes

<table>
<thead>
<tr>
<th>Category Management</th>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td></td>
<td>Automatically given by the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Only used for internal reasons.</td>
</tr>
<tr>
<td>Short name</td>
<td></td>
<td>Identifier of a category, usually not very long.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It is unique.</td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td>Name of the category used to describe it briefly.</td>
</tr>
</tbody>
</table>

#### 4.2.4 Use Case Realization

In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.14), and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.
In the above screenshot, we can see first of all a form where the information of a new category can be set. At the same time, in the table below all the existing categories are listed. Each row represents a different category and the list of sub-rows displayed when clicking the plus icon are the activities belonging to this category.

Through this interface the user can execute all the functional requirements attached to this use case and all the information needed for the correct working of the system can be collected. Moreover, the 4 requirements can be executed in at most 3 clicks, which would be the case of editing a category, for example (choose the category panel, click in the editing button of the concrete category, edit the information and save it).

4.2.5 Classes involved
As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.6).

Table 4.6: UC2's involved classes

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogCatActView</td>
</tr>
<tr>
<td></td>
<td>TimelogCatEntryView</td>
</tr>
<tr>
<td>DTO</td>
<td>CategoryDto</td>
</tr>
<tr>
<td>Action</td>
<td>CategoryAction</td>
</tr>
<tr>
<td>Response</td>
<td>CategoryResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>CategoryHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>CategoryDao</td>
</tr>
<tr>
<td>Entity</td>
<td>Category</td>
</tr>
</tbody>
</table>

4.3 UC3: Activity Management
In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.
4.3.1 Use Case Specification
In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario
1. The user is logged as “admin” so the “Categories & Activities” view is available.
2. The user clicks in “Categories & Activities” flag and this is displayed.
3. The super user has the following options:
   a. The user fills the chart with information of a new activity and press [Add] button. (E1)
      i. The system adds the new activity to the system after checking for duplicates, invalid characters, etc. (E2)
      ii. The new activity is added to the existing categories table, below the categories it belongs but is not displayed.
      i. The activities belonging to it are displayed.
      ii. The super user presses [Delete] icon of an existing activity.
         1. The activity is removed.
      iii. The super user presses [Edit] icon of an existing activity:
         1. The row with the activity’s information becomes editable.
         2. The user changes the information that wants to change.
         3. The user presses [Save] icon. (E3)
         4. The system updates the information after checking for duplicates, invalid characters, if the passwords match, etc. (E4)

Extensions
E1. The user fills the chart and press [Clear] button.
   - Back to point 2.
E2. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 1.a
E3. The user presses [Cancel] icon.
   - There is no change in the category and goes back to point 3.
E4. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 3.b.iii.2

4.3.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.7):

Table 4.7: UC3's Functional Requirements
<table>
<thead>
<tr>
<th>FR9</th>
<th>Add an activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>A new activity will be added to the system.</td>
</tr>
<tr>
<td>Pre-condition</td>
<td>The activity does not already exit in the system.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>A new activity will be added to the system and it will available for the users in the time tracking task.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FR10</th>
<th>Edit an activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>An existing activity’s information has to be changed.</td>
</tr>
<tr>
<td>Pre-condition</td>
<td>None.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>The new information is stored.</td>
</tr>
</tbody>
</table>
FR11 Delete an activity

Description
An existing activity will be deleted from the system.

Pre-condition
None.

Post-condition
The activity will be erased from the system and will not be available for the time track management anymore.

FR12 Visualize user’s data

Description
The specified activity’s information will be displayed.

Pre-condition
None.

Post-condition
None.

4.3.3 Attributes
In the following table (Table 4.8) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the activity.</td>
</tr>
<tr>
<td>Short name</td>
<td>Identifies the activity in a faster way. It is unique.</td>
</tr>
<tr>
<td>Cost</td>
<td>Describes how much that activity costs per hour. The categories it belongs to.</td>
</tr>
<tr>
<td>Categories</td>
<td>All activities must belong to at least one category.</td>
</tr>
</tbody>
</table>

4.3.4 Use Case Realization
In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.15), and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.

![Figure 4.15: UC3's screenshot](image-url)
As in the category management screenshot, we first can find a form where all the needed information for a new activity can be filled. In addition, for each category, all the activities belonging to it can be displayed, so any change or erase can be done at any moment.

Thanks to this GUI, new activities can be saved with all the attributes filled and every task the system must carry out in the aspect of activities can be perfectly executed in less than 3 clicks.

### 4.3.5 Classes Involved
As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.9).

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogCatActView</td>
</tr>
<tr>
<td></td>
<td>TimelogCatActEntryView</td>
</tr>
<tr>
<td>DTO</td>
<td>ActivityDto</td>
</tr>
<tr>
<td></td>
<td>CategoryDto</td>
</tr>
<tr>
<td>Action</td>
<td>ActivityAction</td>
</tr>
<tr>
<td></td>
<td>CatActAction</td>
</tr>
<tr>
<td>Response</td>
<td>ActivityResponse</td>
</tr>
<tr>
<td></td>
<td>CatActResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>ActivityHandler</td>
</tr>
<tr>
<td></td>
<td>CatActHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>ActivityDao</td>
</tr>
<tr>
<td></td>
<td>Category_activityDao</td>
</tr>
<tr>
<td>Entity</td>
<td>Activity</td>
</tr>
<tr>
<td></td>
<td>Category_activity</td>
</tr>
</tbody>
</table>

### 4.4 UC4: Project Management
In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

#### 4.4.1 Use Case Specification
In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

**Main Success Scenario**
1. A user is logged.
2. The user selects the [Projects] flag and this is displayed.
3. Here the user has the following options:
   a. Presses [Create new project] button.
      i. An empty information chart is shown.
      ii. The user enters the information of the new project.
      iii. The user presses [Save] button. (E1)
      iv. The system adds the new project to the system after checking for duplicates, invalid characters, etc. (E2)
      v. The information of the project is now shown as non-editable.
   b. Choose an existing project from the list
      i. The information about the project is displayed.
ii. Then the user has the following options:
   1. The user presses [Delete] button.
      a. The project and all the information related to it (for instance, deliverables) are removed.
   2. The user presses [Edit] button.
      a. The information is now editable.
      b. The user changes the information that wants to change.
      c. The user presses [Save] button. (E3)
      d. The system updates the information after checking for duplicates, invalid characters, etc. (E4)
   3. The user presses [Back] button.
      a. Back to point 3.

**Extensions**

E1. The user presses [Cancel] button.
   - End of the use case.
E2. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 3.a.ii
E3. The user presses [Cancel] icon.
   - There is no change in the category and goes back to point 2.
E4. The data is not valid
   - The system will show a message indicating what data is not correct.
   - Back to point 3.b.ii.2.b

### 4.4.2 Requirements

The functional requirements this use case must satisfy are the following (Table 4.10):

<table>
<thead>
<tr>
<th>FR13</th>
<th>Description</th>
<th>Pre-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add a project</td>
<td>A new project will be added to the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The project does not already exist in the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A new project will be added to the system. It will be added to all the users who participate on it and those users will be able to use it in the time track management.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FR14</th>
<th>Description</th>
<th>Pre-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Edit a project</td>
<td>An existing project’s information has to be changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The new information is stored.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FR15</th>
<th>Description</th>
<th>Pre-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Delete a project</td>
<td>An existing project will be deleted from the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The project will be erased from the system and will not be available for the time tracking anymore.</td>
</tr>
</tbody>
</table>
FR16 | Visualize project’s data
---|---
**Description** | The specified project’s information will be displayed on the screen.
**Pre-condition** | None.
**Post-condition** | None.

### 4.4.3 Attributes
In the following table (Table 4.11) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

**Table 4.11: UC4’s attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>Name</td>
<td>Complete name of the project.</td>
</tr>
<tr>
<td>Short name</td>
<td>Identifier of the project is unique in the system. Usually a short form.</td>
</tr>
<tr>
<td>Customer</td>
<td>Identifies the customer that asked to develop the project. It does not usually change once identified.</td>
</tr>
<tr>
<td>Creation date</td>
<td>Identifies when the project was created. It is not usual to be changed.</td>
</tr>
<tr>
<td>Next meeting date</td>
<td>Indicates when the next meeting about the project is going to be.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of what the project is about.</td>
</tr>
<tr>
<td>Participants</td>
<td>List of users that take part in the project.</td>
</tr>
<tr>
<td>Category</td>
<td>The category the project belongs to, depending on the kind of project.</td>
</tr>
</tbody>
</table>

### 4.4.4 Use Case Realization
In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.16: UC4’s screenshots), and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.
With the implemented interface, the user will first have to decide whether to create a new project or treat with an existing one. Depending on this choice, an empty or filled form will show up with all the information related to the project. The new project’s options will just be to add it or not to the system, while the existing project may be edited or deleted.

The screen has been designed in order to ease all the functionalities of the use case to the user in at most 4 clicks. Moreover, due to the form, all the required information will be set by the user.

4.4.5 Classes Involved
As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.12).
### Table 4.12: UC4's involved classes

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogProjectsView</td>
</tr>
<tr>
<td>DTO</td>
<td>ProjectDto</td>
</tr>
<tr>
<td>Action</td>
<td>ProjectAction</td>
</tr>
<tr>
<td></td>
<td>UserProjectAction</td>
</tr>
<tr>
<td>Response</td>
<td>ProjectResponse</td>
</tr>
<tr>
<td></td>
<td>UserProjectResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>ProjectHandler</td>
</tr>
<tr>
<td></td>
<td>UserProjectHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>UserDao</td>
</tr>
<tr>
<td></td>
<td>User_projectDao</td>
</tr>
<tr>
<td>Entity</td>
<td>User</td>
</tr>
<tr>
<td></td>
<td>User_project</td>
</tr>
</tbody>
</table>

#### 4.5 UC5: Deliverable Management

In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

#### 4.5.1 Use Case Specification

In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

**Main Success Scenario**

1. A user is logged.
2. The user selects the [Deliverables] flag and this is displayed.
3. The user selects a project from the list of projects.
4. A table with all the deliverables belonging to the chosen project is displayed.
5. Here the user has the following options:
   a. Presses [Add] icon from the table:
      i. An empty row is shown in the beginning of the table.
      ii. The user enters the information of the new deliverable.
      iii. The user presses [Save] icon. (E1)
      iv. The system adds the new deliverable to the system after checking for duplicates, invalid characters, etc. (E2)
   b. Presses [Add] icon from a deliverable:
      i. An empty row is shown under the chosen deliverable (who will be the father of the new one) in the table.
      ii. The user enters the information of the new deliverable.
      iii. The user presses [Save] button. (E3)
      iv. The system adds the new deliverable to the system after checking for duplicates, invalid characters, etc. (E4)
   c. Presses [Delete] icon from a deliverable:
      i. The deliverable is removed
   d. Presses [Edit] icon:
      i. The information about the deliverable becomes editable in the table.
      ii. The user changes the information that wants to change.
      iii. The user presses [Save] icon. (E5)
      iv. The system updates the information after checking for duplicates, invalid characters, etc. (E6)
Extensions
E1. The user presses [Cancel] button.
   - End of the use case.
E2. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 5.a.ii
E3. The user presses [Cancel] button.
   - End of the use case.
E4. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 5.b.ii
E5. The user presses [Cancel] button.
   - End of the use case.
E6. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 5.d.ii

4.5.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.13):

Table 4.13: UC5’s Functional Requirements

<table>
<thead>
<tr>
<th>FR</th>
<th>Description</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR17</td>
<td>Add a deliverable</td>
<td>A new deliverable will be added to the system.</td>
<td>The deliverable is not already in the system.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Pre-condition</td>
<td>Post-condition</td>
</tr>
<tr>
<td>FR18</td>
<td>Edit a deliverable</td>
<td>An existing deliverable’s information has to be changed.</td>
<td>None.</td>
</tr>
<tr>
<td>FR19</td>
<td>Delete a deliverable</td>
<td>An existing deliverable will be deleted from the system.</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td>Pre-condition</td>
<td>Post-condition</td>
</tr>
<tr>
<td>FR20</td>
<td>Visualize deliverable’s data</td>
<td>The specified deliverable’s information will be shown.</td>
<td>None.</td>
</tr>
</tbody>
</table>

4.5.3 Attributes
In the following table (Table 4.14) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.
Table 4.14: UC5’s attributes

<table>
<thead>
<tr>
<th>Deliverable Management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>Name</td>
<td>Complete name of the deliverable</td>
</tr>
<tr>
<td>Short name</td>
<td>Identifier of the deliverable is unique in the system. Usually a short form.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description of what the deliverable is about.</td>
</tr>
<tr>
<td>Parent</td>
<td>Deliverable it belongs to. It can be NULL if it does not belong to any other deliverable.</td>
</tr>
<tr>
<td>Project</td>
<td>Indicates the project it belongs to.</td>
</tr>
<tr>
<td>Deadline</td>
<td>Day the deliverable should be finished. If it is empty, it means that the deliverable has been postponed.</td>
</tr>
<tr>
<td>Done</td>
<td>Indicates if the deliverable is on-going or has already been finished.</td>
</tr>
</tbody>
</table>

4.5.4 Use Case Realization

In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.17), and how with it all the functional requirements can be fulfilled and every attribute’s information can be collected.

![Figure 4.17: UC5's screenshot](image)

After choosing a project, the user will be able to see the list of the deliverables and sub-deliverables belonging to it. Every deliverable will be displayed with all the related information and the user will be able to edit or delete them, and as far as they are not sub-deliverables, add them a children deliverable.

This display eases the user to manage any deliverable his way in a really quick process (3 clicks at most), as described in the functional requirements. In addition, all the information explained in the attributes section can be got from this interface, so the use case will be completely fulfilled.

4.5.5 Classes involved

As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.15).
Table 4.15: UC5’s involved classes

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogDeliverablesView</td>
</tr>
<tr>
<td></td>
<td>TimelogDelivEntryView</td>
</tr>
<tr>
<td>DTO</td>
<td>DeliverableDto</td>
</tr>
<tr>
<td>Action</td>
<td>DeliverableAction</td>
</tr>
<tr>
<td>Response</td>
<td>DeliverableResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>DeliverableHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>DeliverableDao</td>
</tr>
<tr>
<td>Entity</td>
<td>Deliverable</td>
</tr>
</tbody>
</table>

4.6 UC6: Time Track Management
In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

4.6.1 Use Case Specification
In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario
1. A user is logged.
2. The [Time Tracking] flag is automatically chosen or the user selects it and this is displayed.
3. Here the user has the following options:
      i. An empty row is shown in the beginning of the table.
      ii. The user enters the information of the new activity.
      iii. The user presses the [Save] icon. (E1)
      iv. The system adds the new activity to the system after checking for duplicates, invalid characters, etc… (E2)
   b. Presses [Delete] icon from a time track:
      i. The time track is removed
   c. Presses [Edit] icon:
      i. The information about the track becomes editable in the table.
      ii. The user changes the information that wants to change.
      iii. The user presses [Save] icon. (E3)
      iv. The system updates the information after checking for duplicates, invalid characters, etc… (E4)

Extensions
E1. The user presses [Cancel] button.
   - End of the use case.
E2. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 3.a.ii
E3. The user presses [Cancel] button.
   - End of the use case.
E4. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 3.c.ii
4.6.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.16):

Table 4.16: UC6's Functional Requirements
<table>
<thead>
<tr>
<th>FR</th>
<th>Description</th>
<th>Pre-condition</th>
<th>Post-condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR21</td>
<td>Add a track on time</td>
<td>A new track will be added to the system.</td>
<td>None</td>
</tr>
<tr>
<td>FR22</td>
<td>Edit a track</td>
<td>An existing track’s information has to be changed.</td>
<td>None.</td>
</tr>
<tr>
<td>FR23</td>
<td>Delete a track</td>
<td>An existing track will be deleted from the system.</td>
<td>None.</td>
</tr>
<tr>
<td>FR24</td>
<td>Visualize track’s data</td>
<td>Every track’s information is displayed on the screen.</td>
<td>None.</td>
</tr>
</tbody>
</table>

4.6.3 Attributes
In the following table (Table 4.17) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

Table 4.17: UC6’s attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time Tracking Management</td>
<td></td>
</tr>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>User</td>
<td>The user that has carry out the task.</td>
</tr>
<tr>
<td>Project</td>
<td>The project the user has been working on during that period of time.</td>
</tr>
<tr>
<td>Deliverable</td>
<td>The deliverable of the project selected the user has been working on concretely.</td>
</tr>
<tr>
<td>Activity</td>
<td>The activity the user has been doing on the chosen deliverable.</td>
</tr>
<tr>
<td>Description</td>
<td>Brief description or note about the job that has been done.</td>
</tr>
<tr>
<td>Link</td>
<td>It can be additional information of what the user has been working on.</td>
</tr>
<tr>
<td>Start</td>
<td>The exact moment the user has started the activity.</td>
</tr>
<tr>
<td>End</td>
<td>The exact moment the user has finished the activity.</td>
</tr>
<tr>
<td>Duration</td>
<td>The time the user has spent developing the activity. Automatically calculated.</td>
</tr>
</tbody>
</table>
4.6.4 Use Case Realization
In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.18), and how with it all the functional requirements can be fulfilled and every attribute’s information can be collected.

![Figure 4.18: UC6's screenshot](image)

When the user logs in, all the time tracks belonging to him will be displayed in a non-editable table. This table will show all the information from every track which may be edited or deleted at any moment. Moreover, a new track may be added whenever the user wants by introducing all the required information.

The implementation and design of this interface allows the user to execute any of the functional requirements defined in the use case in less than 3 clicks and to save all the necessary data of the track in the database.

4.6.5 Classes involved
As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.18).

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogTableView</td>
</tr>
<tr>
<td></td>
<td>TimelogEntryView</td>
</tr>
<tr>
<td>DTO</td>
<td>TaskDto</td>
</tr>
<tr>
<td>Action</td>
<td>TaskAction</td>
</tr>
<tr>
<td>Response</td>
<td>TaskResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>TaskHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>TaskDao</td>
</tr>
<tr>
<td>Entity</td>
<td>Task</td>
</tr>
</tbody>
</table>

4.7 UC7: Reporting
In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

4.7.1 Use Case Specification
In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario
1. A user is logged.
2. The user clicks in “Reporting” flag and this is displayed.
3. All the existing reports that belong to the user are shown in a table.
4. Here the user has the following options:
   i. An empty row is shown in the beginning of the table.
   ii. The user enters the information of the new report.
   iii. The user presses the [Save] icon. (E1)
   iv. The system adds the new activity to the system after checking for duplicates, invalid characters, etc. (E2)
   v. The report is displayed after the table.

   i. The chosen report is displayed after the table.

c. Presses [Delete] icon from a report:
   i. The report is removed

d. Presses [Edit] icon of a report:
   i. The information about the report becomes editable in the table.
   ii. The user changes the information that wants to change.
   iii. The user presses [Save] icon. (E3)
   iv. The system adds a new report with the new information after checking for duplicates, invalid characters, etc… because changes in an old report are considered as the creation of a new report. (E4)
   v. The new report is also added on the table.

Extensions
E1. The user presses [Cancel] button.
   - End of the use case.

E2. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 4.a.ii

E3. The user presses [Cancel] button.
   - End of the use case.

E4. The data is not valid.
   - The system will show a message indicating what data is not correct.
   - Back to point 4.d.ii

4.7.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.19):

<table>
<thead>
<tr>
<th>Table 4.19: UC7’s Functional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR25</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

| FR26 | Create a report from an existing one |
|      | Description     | An existing report will be modified to create a new one. |
|      | Pre-condition   | The new report does not exist yet. |
|      | Post-condition  | The new report is created and it is displayed on the screen. |

| FR27 | Delete a report |
|      | Description     | An existing report will be deleted from the system. |
|      | Pre-condition   | None. |
|      | Post-condition  | The report will be erased from the system. |
FR28 Repeat a report

<table>
<thead>
<tr>
<th>Description</th>
<th>The specified report will be displayed on the screen.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-condition</td>
<td>None.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>None.</td>
</tr>
</tbody>
</table>

4.7.3 Attributes

In the following table (Table 4.20) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

**Table 4.20: UC7's attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Automatically given by the system. Only used for internal reasons.</td>
</tr>
<tr>
<td>Type</td>
<td>Describes if the report is going to be a bill or a memory.</td>
</tr>
<tr>
<td>Creation</td>
<td>The date the report was created. Automatically introduced.</td>
</tr>
<tr>
<td>Project</td>
<td>The project the report is going to be about if there is no deliverable.</td>
</tr>
<tr>
<td>Deliverable</td>
<td>The deliverable the report is going to be about if there is any.</td>
</tr>
<tr>
<td>User</td>
<td>The user who is asking for the report.</td>
</tr>
<tr>
<td>Start</td>
<td>The starting date of the period that is going to be reported.</td>
</tr>
<tr>
<td>End</td>
<td>The ending date of the period that is going to be reported.</td>
</tr>
</tbody>
</table>

4.7.4 Use Case Realization

In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.19), and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.
Figure 4.19: UC7’s screenshot

This use case’s interface shows all the reports created by the user. From here, the user will be able to create a new one from the beginning or editing an old one. He also will be able to re-display an existing report or to delete it, as he wishes. As it can be seen, all the information about any report can be collected or shown in this GUI and all the defined requirements can be executed in no more than 3 clicks.

4.7.5 Classes involved

As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.21).

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>TimelogReportingView TimelogReportEntryView</td>
</tr>
<tr>
<td></td>
<td>ReportDto</td>
</tr>
<tr>
<td></td>
<td>BillDto</td>
</tr>
<tr>
<td></td>
<td>MemoryDto</td>
</tr>
<tr>
<td>DTO</td>
<td>BillDto</td>
</tr>
<tr>
<td></td>
<td>MemoryDto</td>
</tr>
<tr>
<td>Action</td>
<td>ReportAction</td>
</tr>
<tr>
<td></td>
<td>BillAction</td>
</tr>
<tr>
<td></td>
<td>MemoryAction</td>
</tr>
<tr>
<td>Response</td>
<td>ReportResponse</td>
</tr>
<tr>
<td></td>
<td>BillResponse</td>
</tr>
<tr>
<td></td>
<td>MemoryResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>ReportHandler</td>
</tr>
<tr>
<td></td>
<td>BillHandler</td>
</tr>
<tr>
<td></td>
<td>MemoryHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>ReportDao</td>
</tr>
<tr>
<td></td>
<td>BillDao</td>
</tr>
<tr>
<td></td>
<td>MemoryDao</td>
</tr>
<tr>
<td>Entity</td>
<td>Task</td>
</tr>
</tbody>
</table>

Table 4.21: UC7’s involved classes
4.8 UC8: Log In

In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

4.8.1 Use Case Specification

In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

Main Success Scenario
1. The user introduces his/her username and password.
2. The system checks the password matches with the password of the username introduced. (E1)
3. The correspondent display is shown.

Extensions
E1. The data is invalid.
   - Back to point 1

4.8.2 Requirements

The functional requirements this use case must satisfy are the following (Table 4.22):

<table>
<thead>
<tr>
<th>Table 4.22: UC8's Functional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR29</td>
</tr>
<tr>
<td>Log in</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Pre-condition</td>
</tr>
<tr>
<td>Post-condition</td>
</tr>
</tbody>
</table>

4.8.3 Attributes

In the following table (Table 4.23) all the attributes this use case needs are shown, each of them with a brief explanation of their meaning or job.

<table>
<thead>
<tr>
<th>Table 4.23: UC8's attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log in</td>
</tr>
<tr>
<td>Attribute</td>
</tr>
<tr>
<td>Username</td>
</tr>
<tr>
<td>Password</td>
</tr>
</tbody>
</table>

4.8.4 Use Case Realization

In this section we will, on one hand, show screenshots of how each use case interface looks like (Figure 4.20), and how with it all the functional requirements can be fulfil and every attribute’s information can be collected.
The interface shows a small dialog box where the user may introduce his username and password in order to enter the system. This simple GUI makes it easy to log in the system, fulfilling the only functional requirement of the use case.

### 4.8.5 Classes involved

As we have said before, in this section we will list the classes involved in this use case and to which general group of the previous explanation it belongs (Table 4.24).

<table>
<thead>
<tr>
<th>General name</th>
<th>Class name (.java)</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Timelog</td>
</tr>
<tr>
<td></td>
<td>TimelogPanelView</td>
</tr>
<tr>
<td>DTO</td>
<td>-</td>
</tr>
<tr>
<td>Action</td>
<td>LoginAction</td>
</tr>
<tr>
<td>Response</td>
<td>LoginResponse</td>
</tr>
<tr>
<td>Handler</td>
<td>LoginHandler</td>
</tr>
<tr>
<td>DAO</td>
<td>SessionDao</td>
</tr>
<tr>
<td></td>
<td>UserDao</td>
</tr>
<tr>
<td>Entity</td>
<td>User</td>
</tr>
</tbody>
</table>

### 4.9 UC9: Log Out

In this section the specification, requirements, attributes and realization of the Use-Case are explained by mapping it with the described in chapter 3, but no example is given. For an example, look at the description of Use-Case number 1, User Management.

#### 4.9.1 Use Case Specification

In this section all the use case flow is explained, both in a successful case and in an alternative case (extensions).

**Main Success Scenario**

1. The user presses [Logout] link.
2. The system closes the current session discarding the changes that has not been saved and displays the log in screen.
4.9.2 Requirements
The functional requirements this use case must satisfy are the following (Table 4.25):

Table 4.25: UC9's Functional Requirements

<table>
<thead>
<tr>
<th>FR30</th>
<th>Log out</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The current user of the system is going to abandon it.</td>
</tr>
<tr>
<td>Pre-condition</td>
<td>None.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>The user will leave the system and screen will show the log in interface.</td>
</tr>
</tbody>
</table>

4.9.3 Attributes
This use case does not need any attribute for its realization.

4.9.4 Use Case Realization
In all interfaces of the system, except the login, there is the possibility of executing this use case by clicking in the “Logout” link. For this reason, we will not show any screenshot in this area, but the link can be seen in any of the screenshots shown before.

4.9.5 Classes involved
As the Use-Case can be executed at any moment and has no link to the database, the only classes involved are all the view classes mentioned before.
5 Conclusion & Future Work

In this last chapter, we will talk about the conclusions we have extracted from the development of this project. At the same time, we will mention some of possible jobs it could be done in the project, such as improvements, updates or extensions of functionalities in order to define which work has to be done in the future.

5.1 Conclusion

The aim of this thesis was to solve a problem ARiSA had due to their increasing growth. This solution was going to arrive by improving and expanding a prototype which implementation was already started. For a successful work, the following goals had to been reached.

The main goal was to provide a tool that was able to store the time spent on each particular task by a user in an efficient way. This goal has been successfully achieved by the implementation of the “Time track management” use case. Moreover, thanks to an intuitive interface, the use case can be executed in few seconds, like it was defined as efficient.

The second goal referred to the fact of having a heterogeneous environment when talking about operative systems and access places. It was important to assure that no installation had to be required for the correct running of the system. Due to the use of GWT, it is sure that the system will run as expected in any browser, fulfilling the criteria.

Another goal we had to reach was the reporting creation. The system needed to have the necessary features to allow users generate detailed reports based on the content of the database. To achieve this goal the generation of reports had be a reality, which it is. “Reporting” use case implements the necessary functions in order to consult all the necessary information in the database and create either a bill or a memory with it.

Finally, the last goal we had to achieve to complete this thesis successfully was to provide the means to authorized users to manage users and specify projects, activities or tasks. This goal has been achieved, as it can be seen in the specification of the rest of the use cases. Any of the elements mentioned can be created, edited and deleted by users with that authority.

Because of the success just mentioned, we can conclude by saying that all the goals have been achieved. The project has fulfilled every expectation and so the web-based application is ready to become operational.

5.2 Future Work

As future work, there are several points we would like to take into considerations. On one hand, it is important to keep the application up to date technologically talking. A revision of the GWT version would be recommended every certain time, in order not to have a deprecated code.

On the other hand, many extensions can be implemented for this application. Some extensions may be data extensions, in other words, more data can be stored, and for instance, customer related data. This would generate at the same time an extension in the functionality aspect, because this would cause the creation of a new use case.

Moreover, some use cases could be expanded to execute more tasks than the ones they are doing now. A good example of it is the reporting use case, due to the fact that other kind of reports can be needed in the future. In this case we would have to add a functional requirement to the use case and implement its execution.
REFERENCES


