UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Clean Architecture with PHP

Treball de Final de Grau

–

Memory

Author:
Ferran Martín Sànchez

Degree:
Grau en Enginyeria Informàtica
Enginyeria del Software

Director:
Gemma Casamajó

Company:
l'Apòstrof SCCL

Tutor:
Ernest Teniente

Department:
Enginyeria de Serveis i Sistemes
d'Informació (ESSI)

October 26, 2017

Acknowledgments

I want to start this document by thanking all those who have participated in this project or who have been interested in some way, especially to:

People at "l'Apòstrof", to give me this oportunity and to trust me to carry out this project, and especially to Gemma and Martí, who have formed the *Comissió Intranet* and all together we have been tracking and organizing this project.

To my tutor *Ernest Teniente*, for all the help and advice that he has given me, not only during this project, but during the entire university path.

To my family, for all the support given, and especially to Mariona, my partner, without their support and help it would not have been possible to carry out this project.

Abstract

It has always been important to keep track of processes in companies. Starting with the Information Systems boom it has become even more obvious that it's a key factor in having competitive advantages. Today, most large companies already have some type of software to control their internal processes, therefore, it is not an advantage but a requirement to stay competitive.

The paradigm changes with small and medium-sized businesses, which often do not have the resources to access these software. Therefore, being able to implement a software to control their processes can in fact mean a competitive edge with respect to the others. In addition, if we focus on companies in the third sector and cooperativism, it is almost impossible to find some software that reflects how to understand their business economics and its role in relationship to their sector and society.

That is why the objective of this paper is to create a first version of a work management software for the cooperative "l'Apòstrof SCCL" that also includes concepts of human and social scope. The project has the vision of offering the tool to cooperatives and companies of the third sector.

Resum

Des de sempre ha sigut important portar un control dels processos en les empreses. I a partir de l'auge dels Sistemes d'Informació es va fer més evident que era un factor clau per aconseguir avantatges competitius. Avui en dia totes (o quasi totes) les grans empreses ja tenen algun tipus de software per controlar els seus processos interns, per tant el fet de tenir-ho ja no aporta un avantatge competitiu respecte als altres, sinó que és un requisit.

Però això canvia amb les mitjanes i petites empreses, les quals molts cops no tenen els recursos necessaris per accedir a aquests softwares. Per tant arribar a aconseguir implantar un software per controlar els seus processos, sí que pot significar un avantatge respecte als altres. A més, si ens centrem en empreses del tercer sector i del cooperativisme, es fa quasi impossible trobar algun software que reflecteixi la forma d'entendre l'economia, i el paper d'aquesta i de les empreses dins la societat.

És per això que l'objectiu del present treball és el de crear una primera versió d'un software de gestió de feines per a la cooperativa "l'Apòstrof SCCL" que també inclogui conceptes d'àmbit humà i social. El projecte té la visió d'oferir l'eina a cooperatives i empreses del tercer sector.

Resumen

Desde siempre ha sido importante tener un control de los procesos internos en las empresas. Y a partir del auge de los Sistemas de Información se hizo más evidente que era un factor clave para conseguir ventajas competitivas. Hoy en día todas (o casi todas) las grandes empresas ya tienen algún tipo de software para llevar el control de sus procesos internos, por lo que el echo de tener este software ya no aporta una ventaja, sino que se ha convertido en un requisito.

Pero esto cambia con la medianas y pequeñas empresas, que muchas veces no tienen los recursos necesarios para acceder a estos softwares. Por lo que conseguir implantar un software para llevar el control de sus procesos sí que puede significar tener una ventaja competitiva respeto a los demás. A parte, si nos centramos en empresas del tercer sector y del cooperativismo, se hace casi imposible encontrar un software que refleje la forma de entender la economía, y el papel de ésta y las empresas dentro la sociedad.

Es por esto que el objetivo del presente trabajo es el de crear una primera versión de un software de gestión de trabajo para la cooperativa "l'Apòstrof SCCL" que también incluía conceptos de ámbito humano y social. El proyecto tiene la visión de ofrecer la herramienta a otras cooperativas y empresas del tercer sector.

Contents

1. Context

1.1 Contextualization

This project has been done within an agreement of educational cooperation with the company "l'Apòstrof". "l'Apòstrof" is a communication cooperative where work journalists, linguists, designers, teachers and developers, all together working with multidisciplinary teams to promote creativity. We can see the internal organization of "l'Apòstrof" in two different ways; the first way is to see that the company has two levels, the partners and the workers. The second way is to see that the company works through work teams that can be formed by one to many areas (commercial, comunication, design and development). In the end, this ways to organize the company prioritizes horizontality. Each project is assigned to a responsible (partner or worker) and the important decisions that affect the whole company are taken in an assembly formed by the partners.

The present project arises from the need to update and improve the current intranet used in "l'Apòstrof". This intranet it's not a generic one, was custom made 10 year ago, but now it has become obsolete. It's a fact that the current intranet was not made using any development quality standard, or thinking on maintainability and extansability at a medium or long term. It's also a fact that the changes done this last years has been done quickly as soon the needs has arrived without any previsions and organization. With all of that, the current state of the software is that is not reliable, nither stable, and has so many bugs. So, now the company needs to renew this tool in order to create one that can be reliable, stable, maintainable and extensible.

1.1.1 Personal motivation

During my last work experience, before at "l'Apòstrof", I had the opportunity to work on a project to create and maintain a large and complex iOS app. At the start of the project we had a buggy app (in fact it was a long term prototype), and we decided to do it again using a Clean Architecture implementation. The result was a very stable and reliable app (with 800

daily active users we only have 0.1% of error rate), and easily maintained and extended.

Having seen the result of implementing a Clean architecture in an iOS app, I wanted to test whether an implementation of this architecture in a PHP web project would give the same results and if it would be worth it.

1.2 Actors

1. **L'Apòstrof**
   L'Apòstrof (partners and workers) are the main Stakeholder of the project since they will be the direct beneficiaries.
   In order to streamline decision-making, in this project has created an internal commission of two people formed by Gemma Casamajó (coordinator of the "l'Apòstrof" and director of the present TFG) and Martí Làzaro (designer and partner of "l'Apòstrof"), who will play the role of *Product Owner*[1].
   The rest of the workers and partners will regularly try the new intranet to be able to give feedback as soon as possible.

2. **Development Team**
   It is made up of two people, Asier Illarramendi (developer and worker of l'Apòstrof) and I, Ferran Martin (developer and worker of l'Apòstrof). Even so, the intranet project will be developed mainly by myself, since Asier will have to continue carrying out projects for company's clients. Therefore, Asier's functions in this project will mainly be to program some parts of the project, participate in technical decision making, and do *Code Reviews*[2] (technique to ensure the quality of the code, explained later).

3. **L'Apòstrof's Clients**
   Customers are also an important *Stakeholder*, but indirectly, because even though they will not use the intranet, clients will see some benefits

[1]Product Ower: `https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner`

[2]Code Reviews: `http://blogs.atlassian.com/2009/11/code_review_in_agile_teams_part_i/`

if the company improves their internal processes.

4. **Other cooperatives**
   L'Apòstrof is located within a group of cooperatives (ECOS Group). Within this group, intercooperativity is enhanced. L'Apòstrof is one of the only cooperatives in the group that currently has a customized intranet that responds to many of its needs (despite having the shortcomings already mentioned). From the moment it was known that L'Apòstrof wanted to renew and improve his intranet, other cooperatives of the group already demonstrated their interest in being able to use them as well. This is very far from the scope of this project (as we will see later), but we must keep it in mind when designing the intranet. That's why the other cooperatives have been included as *Stakeholders*.

5. **Tutor**
   The speaker of the project is Ernest Lieutenant Lopez, who belongs to the specialty of Software Engineering and is a FIB professor. His role will be to verify that the goals set are met at the end of each phase.

2. Problem formulation

Although we have seen in the previous section the current intranet is not reliable and it would be very difficult to modify and extend it, we will use it to analyze the functionalities it currently has.

Then we will describe the functionalities which will remain the same and the ones which will have to be improved or changed. Also, we will describe the new functionalities which need to be implemented. There are some current functionalities that are not needed anymore, so, at the same time we will clean/delete them. These functionalities are not described in this document since they have no impact on this project.

2.1 Current Intranet

The current intranet is organized by areas grouped by certain functionalities. The following is the description of these areas.

1. **Administration Area**
   This area includes all the functionality required for the administration of the intranet.

   - **Workers:** It allows you to create, edit and delete workers.
   - **Users:** It allows you to create, edit and delete users who have access to the intranet and with which permissions. They may or may not be linked to a worker.
   - **Notifications:** It allows you to create notifications with a message and recipient users. These notices will be shown to recipients when they enter the intranet.

   We should also improve a lot the functionality of **Hourly Basis**. This allows the creation and edition of the employee's bases, which is currently very rudimentary.

2. **Business area**
   This area contains functionalities that allows defining how the work of

the company's staff is organized.

- **Areas:** It allows you to define in which areas "l'Apòstrof" is organized. Currently are communication, text, design, solidarity economy, teaching, and development.
- **Abilities:** It allows you to define a list of skills that users can mark when they enter hours.

3. **Work management area**
   Here is where all the daily needed functionalities by company's staff are.

   - **Clients:** It allows you to manage all the information about clients.
   - **Productive Projects:** It allows you to manage all the information about the clients' projects. Projects can be associated with expenses.
   - **Reproductive Projects:** It allows you to manage all the information about the tasks/projects that are carried out internally and do not provide a direct benefit to the company. For the company being able to analyze data of these jobs is just as important as that of the productive ones.
   - **Hours:** It allows workers to enter their worked hours. Both, hours of Reproductive work and Productive work, can be entered. In the case of Productive projects, the skills used to perform the task must be also indicated.
   - **Invoices:** With this feature, you can create invoices for the work done.

   The functionality of **Productive Projects** is intended to be expanded and introduce the concept of Project Batch, which will allow you to define several batches for each project, and associate these batches to a company's area. The expenses would also be associated with the project batches. With all this, better-organized data could be extracted.

   Another current functionality that we want to improve is **Reproductive Projects**. As explained above, reproductive work is the internal

workings of the company which do not provide a direct economic benefit to the company. Currently, these jobs only have an identifier, a name, and a description. And although workers can enter hours associated to them, they can not define which skill and/or areas they refer to. With the new intranet, we want to incorporate functionality to be able to manage reproductive work as if it was productive work; that is, associating expenses, giving an estimated invoicing, entering hours with details, etc.

4. **Networking area**
   All functionalities to manage the company's networking are located in this area. Basically, it will be where all the data of the **providers** are managed. Currently there is also the functionality of collaborators, but it has been decided to unify both functionalities in one (providers), since, in fact, they were exactly the same.

5. **Reports area**
   Here you can find all the reports that are used to see the status of the company. Many of these reports have to be rethought, as they do not offer useful data, or they are not generated correctly.

   - **Billing:** reports on net and total turnover per years, billing between months and billing for clients.
   - **Expenses:** reports about closed projects' expenses, and forecasts on open expenses.
   - **Productivity:** reports with productivity data; different summaries of hours according to some parameters. It is important to clarify that although the name of these reports is *productivity*, reproductive work is also analyzed in them.
   - **Profitability:** estimated and real hourly price calculations.

   Reports in general needs to be improved a lot, and in particular there are a series of reports that are now being generated manually (with excel) that are wanted to be automated. An important improvement we want to do is to be able to better analyze reproductive work.

2.2 New features

Apart from the functionalities that the intranet already has, we want to add some more. These new features include the ability to rectify invoices, add the concept of "batches" to projects, add a comment system for projects, and a management system for user profiles (they are now predefined and can not be modified).

We are also thinking of developing new and larger features that will only be done if the previous ones are finished. These are:

- project management functionality in order to improve it within the company and better control the workload of the workers.
- prepare the intranet to be able to be offered as a service to other cooperatives.

2.3 Goals

As you can see in the previous sections, the software that we want to develop is an Intranet with many functionalities that will allow the "l'Apòstrof" to control the internal work processes and will help to make decisions through the generated reports.

So, the main goal of this project is to develop a first version of this software. This first version must have implemented the minimum functionalities necessary to stop using the current Intranet, and the rest of the features will be left for later versions.

Below are the minimum functionalities needed to reach this main objective.

- Basic system's functionalities such as login, logout, and user management.
- Be able to create projects and introduce hours associated with these projects and workers.
- Be able to create invoices for projects to be sent to customers.

2.4 Possible obstacles

As you can see, it's intended to create an intranet with a lots of functionalities, therefore we can consider it a very large and wide project. Being a project of these characteristics, it will be easy for there to be many obstacles during its development. Therefore it is difficult to foresee all of them. However, we can say that the main problems in achieving the project's goals will probably be due to a lack of knowledge and experience in the subject, to a tight deadline, and to changes in the requirements of the system's functionalities.

2.4.1 Obstacles with the implementation of Clean architecture

As the name of this project indicates, and as explained above, we want to create this intranet using an implementation of a Clean architecture. Therefore we can find that we dedicate more hours than expected to think and decide about it, since it is not a typical architecture that has been studied in the university.

2.4.2 Time barriers

For different reasons, we may need more time than we initially think. For example, all the system requirements are not entirely clear, and we want to be able to make modifications during the development of these. We can also find time problems due to the fact that for economic reasons the company has to spend more hours on other projects. In any case, as will be explained later, an agile methodology has been chosen, in order to minimize these obstacles. And the extra hours needed to reach the minimum goals will also be done.

2.5 Scope

Once we have defined the goals we want to achieve within this project, it is clear that the scope of the project is delimited to met these objectives.

On the other hand, it is beyond the scope of the project to develop more functionalities than the minimums described. Due to the fact that "l'Apòstrof" is a company where people looks closely at the detail of graphic designs, it has also been explicitly agreed that it is beyond the scope of this project to achieve a hight level of visual design. Even so, it will be necessary to make the system easy to use.

Also, because we are going to develop a first version of a software that will continue evolving, it is implicit that this software must be able to be maintained and extended.

3. State of the art

On the Problem formulation chapter, we have already analyzed the functionalities of the current Intranet, but it is also important to make an analysis of other existing systems to see if there is one that already meet these needs or not.

Since the architecture that we want to use is quite new, it is a good idea to analyze what documentation we can use to carry out the project. As a documentation, we are referring to all the bibliography written in the theoretical field and also look for if there are already examples of projects that put this architecture into practice in a similar context (a PHP web application).

3.1 Existing solutions

It is clear that there are already many work management tools in the market, since it is one of the first *problems* that began to be computerized in companies. But it is also true that until a few years ago these systems have only been available to large companies, and it has not been until the *boom* of web applications and services that these systems haven't arrived to medium and small businesses. In addition, in general all the solutions already created focus solely on the economic and productivity aspects, leaving aside other aspects such as human and social aspects.

Although it seems obvious that it is difficult to find an existing solution that addresses all the aspects that are desired, it is still important to analyze these solutions. Below are some of these systems:

- **Billage** *(https://www.billage.es/es/)* Billage is a multiplatform service (web, Android and iOS) for companies and freelancers that facilitates management in the fields of business, project management and billing. As strengths have several things; To begin, it offers many integrations with other widely used systems such as Google services (Gmail, Calendar and Drive), Mailchimp and Typeform, which can facilitate many things to companies with few resources. And in the

project management section, we should emphasize the features of managing tasks with a Kanban table and an interface very well achieved to introduce hours that greatly facilitates this task.

- **Anfix** *(https://anfix.com)* Another cross platform service (web, Android and iOS) focused mainly on economic management (expenses and invoices). It also incorporates a small project management to be able to organize these expenses and invoices in projects. One feature that should be highlighted is the fact of customizing the invoice through editable templates.
- **Teamleader** *https://www.teamleader.es* Teamleader is a web platform that offers functionalities for managing different areas of a company, including project management and billing. As remarkable functionality it has the fact of being able to give access to the projects to the clients, and thus to facilitate the communication.

As has already been said, in the market there are many other solutions already made, the ones previously mentioned have been highlighted, since they offer some outstanding functionality that could take on an example for the future.

The main problem that makes it very difficult to find a system that could be used is the fact that none allow the management and analysis of internal tasks that do not provide a direct benefit (which is not billed), and this is a very point important for "l'Apòstrof".

3.2 Clean Architecture

As the title of this project indicates, we want to create this intranet using an implementation of a Clean architecture[1]with PHP.

The concept of *Clean Architecure*[1] was introduced by Robert C. Martin[2] in August 2012 in an entry at his blog. Previously he had already written

[1]Clean Architecture: `https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html`

[2]Robert C. Martin: `https://en.wikipedia.org/wiki/Robert_Cecil_Martin`

another post where he began to draw one of the differentiating concepts of this architecture; the concept of *Screaming Architecture*[3].

Apart from the two previous blog posts mentioned, there has been no more bibliography written on this subject in the theoretical field. Conference videos can be found where Robert C. Martin speaks and exemplifies something more about this architecture. But it was not until 017 that a book was published. The book *Clean Architecture: A Craftsman's Guide to Software Structure and Design*[4] was published in September 2017. This book could not be used as a reference for this project, since it was not possible to get a copy.

If we look at how to implement this architecture in a PHP web service we can find some references. The most significant is the book *The Clean Architecture in PHP*[5]. There are also some blog posts explaining a bit more about how to make an implementation with PHP, but they do not go deep.

Although it seems that until recently there has not been much written literature on this subject, and that therefore we could say that it does not generate much interest, this is not true. To begin, this architecture is strongly linked to the concepts *Clean Code*[6] (which has been relevant in the sector for more time and has a lot of written bibliography) and *SOLID principles*[7]. It is also easy to observe how more and more articles, conferences and talks that address these issues can be found. And lastly, the interest in this architecture has been demonstrated by the fact that the new book published in 2017 by Robert C. Martin had already achieved the "best seller" category on Amazon before release.

So, why this

> new architecture can generate this interest? Surely for the benefits it seeks to provide and from where it comes from, since it is not a "new" architecture by itself, it is an attempt to unify and

[3]Screaming Architecture: `https://8thlight.com/blog/uncle-bob/2011/09/30/Screaming-Architecture.html`

[4]Clean Architecture: A Craftsman's Guide to Software Structure and Design: `https://www.amazon.es/dp/0134494164/`

[5]The Clean Architecture in PHP: `https://leanpub.com/cleanphp`

[6]Clean Code: `https://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882`

[7]SOLID: `https://en.wikipedia.org/wiki/SOLID_(object-oriented_design)`

align other types of architectures, keeping the best of each one.

These architectures are; Hexagonal architecture[8], Onion architecture[9], Screaming architecture[10], Lean architecture[11] and the concept of Use Case Driven Approach[12].

According to the author, as a result an architecture that provides the following benefits is achieved:

1. **Frameworks[13] independence:** Many systems are built using frameworks, since frameworks facilitate the work of the developers. The problem arrives when the coupling level with these frameworks is high and for some reason, the framework must be changed or updated. This will not be an easy task. So, it is important to maintain the level of attachment with the frameworks or tools to the minimum possible.

2. **UI independence:** It is usual for the UI to change constantly, without really altering the rest of the system. Therefore it is important that the system's core (business rules[14]) is not coupled to the UI.

3. **Database independence:** In the same way that with the UI, it is important to maintain a low level of coupling with the system used to store the data.

4. **Testable:** It is important to be able to test the systems to verify that they work correctly, and especially the central parts. Therefore, thanks to the fact that we can easily disconnect the system core from all the external details (such

[8]Hexagonal architecture: `http://alistair.cockburn.us/Hexagonal+architecture`

[9]Onion architecture: `http://jeffreypalermo.com/blog/the-onion-architecture-part-1/`

[10]Screaming architecture: `https://8thlight.com/blog/uncle-bob/2011/09/30/Screaming-Architecture.html`

[11]Lean architecture: `http://www.leansoftwarearchitecture.com/`

[12]Use Case Driven Approach: `http://www.interface.ru/rational/rup51/manuals/intro/im_feat2.htm`

[13]Framework: `https://en.wikipedia.org/wiki/Software_framework`

[14]Business rules: `https://en.wikipedia.org/wiki/Business_rule`

as the UI and Databases) it is much easier to test these important parts.

4. Methodology and rigor

4.1 Work methodologies

Since this project will be carried out mainly only for me, no popular methodology will be used in itself. However, we will look at methodologies that can be useful to us and we will adapt them to the needs of the team and project.

Since this is a project where the requirements are not perfectly defined, we want to be able to make changes during the development and we want to be able to test the results as soon as possible to be able to decide these changes. So, it is a perfect environment to use *agile methodologies*.

The two well-known methodologies are *Kanban*[1] and *Scrum*[2]. In one hand, with *Kanban*, we try to have more precise control of work over time, while on the other side, with *Scrum*, above all, we try to be able to provide flexibility to work with changing requirements. Therefore it has been decided to use the *Scrum methodology*.

4.1.1 Final method

The process that has been decided will be as follows:

- An *Comissió Intranet* will be created, composed of two members of ""l'Apòstrof"", who will perform the role of *Product Owner*[3] and will be in charge of deciding the system's requirements and the priorities of the project.

[1]Kanban: `https://es.wikipedia.org/wiki/Kanban_(desarrollo)`

[2]Scrum: `https://proyectosagiles.org/que-es-scrum/`

[3]Product Owner: `https://www.mountaingoatsoftware.com/agile/scrum/roles/product-owner`

- At the beginning of the project, an initial *BackLog*[4] will be created with all the *User Stories*[5] that we may think. These User Stories must be as small as possible and independent of each other.
- Work will be carried out on incremental iterations (defined later). Thus, step by step, functionalities will be added to the system, and at the end of each iteration there will be a functional system.
- These iterations will be two weeks long. At the beginning of each iteration, there will be a meeting with the *Comissió Intranet* to decide which stories to implement. And at the end of the iterations, it will be checked which ones have been implemented and which ones not.
- Occasionally we will do some *testing days* with other people to check everything developed so far and provide feedback.

4.2 Tracking tools

Trello[6] will be used as a software to help to plan these iterations and track them. In order to control versions of the project, a git[7] repository hosted on GitHub[8] will be used. This will also monitor the changes that occur during the development.

[4]BackLog: `https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/product-backlog`

[5]User Stories: `https://www.mountaingoatsoftware.com/agile/user-stories`

[6]Trello: `https://trello.com/`

[7]Git: `https://git-scm.com/`

[8] GitHub: `https://github.com/`

4.3 Validation methods

4.3.1 Manual validation methods

As explained during the "Work Methodologies" chapter, meetings will be held at the end of each iteration with *Comissió Intranet* (*Product Owner*) to verify that the functionalities implemented work well and do everything they need to do. In addition, there will also be a trial version of the Intranet available to workers so they can try it whenever they want. And we will mark *test dates* on certain iterations so the workers can test the entire system implemented so far.

In order to ensure the quality of the project, *Code Reviews* will also be made by two project's developers. To do this, for each functionality that is implemented, a branch will be created in the git repository and once completed, a *Pull Request* will be created. This will make all changes that involve functionality easy.

Code Review will also be made between the two project developers, in order to ensure the quality of the project. To do this, for each functionality that is implemented, a branch will be created in the git repository and once completed, a *Pull Request* will be created towards the main branch. So it will be easy to review all the changes that involve each functionality.

4.3.2 Automatic validation methods

The main part of automatic validation methods will be Unit Tests, which will be carried out in all possible classes/parts of the code. To implement all these tests, TDD practice will be the used. Integration Tests will also be created to verify that the entire system works correctly.

5. Requirements Analysis

5.1 Functional requirements

As mentioned in Goals section, a minimum number of functionalities must be implemented in order to stop using the current Intranet and achieve the main goal of this project.

In this section we will describe all the functional requirements necessary to achieve these functionalities.

We can say that the main features are those directly related to work management, especially to be able to associate worked hours, expenses, and invoices to the projects. This last section of invoices (or billing) is also quite broad. Therefore, we can group these features into 4 large groups; a first group related to all the basic functionalities necessary to interact with the system, a second group with features referring to basic system's data management, a third group that includes the functionalities for the management of the tasks by themselves, and a fourth group where all the billing features are implemented.

5.1.1 System:

This group basically includes two very basic features, the **login and logout** of users.

5.1.2 System administration:

Since you have to log in to use the system, it is obvious that the first feature of this group should be a users managment system; In particular, the system will allow to **create, edit, delete, and list users of the system**.

As we have said, project will have worked hours associated, therefore, it should also be possible to **create, edit, delete and list workers**, which will be associated to the hours spent on projects. The workers can only be erased in case they do not have any associated worked hour. Otherwise they will not be able to be deleted, but they will be deactivated. You can also **associate a worker to a user**, so this user can enter worked hours of the worker.

Projects will be organized in batches, which at the same time will be related to an area of the company. Therefore the system will have to allow the management of these areas, in particular, to **create, edit and list areas** of the company.

Expenses will also be associated with projects' batches. These expenses will have to be classified, so another requirement will be to able to **create, edit, delete and list the different types of expenses**. These expenses will also be associated to collaborators or suppliers in order to be able to analyze this data in the future. Therefore, it is obvious that the functional requirements of **creating, editing, deleting and listing providers** are also necessary.

To end the functionalities of this group, each project will be carried out for a client, therefore we also want to be able to **create, edit, delete and list clients** of the company, to be able to associate them with the projects and invoices.

5.1.3 Projects managment:

All the functionalities described above are necessary to be able to finally perform the functionalities described below in this group.

Obviously, the most important features will be **view, create, edit, delete, and list projects**. As mentioned above, projects will be organized in batches, so, we should also be able to **create, edit, delete and list the projects' batches**, and for each batch, there must be the ability to **create, edit, delete, and list expenses**.

Finally, in order to have list of worked hours, users can **enter hours associated to a project's batches and a worker**, they should also be able **list and delete these hours**.

5.1.4 Billing:

This last group contains all the features related to the billing of projects. Basically for each project we want to be able to control the invoices that are sent to clients.

Apart from the invoices's data (client's data, amounts, lines, etc.), invoices can have a total of 4 states:

1. The first state will be *requested*. This will mean that the invoice has been entered to the system with the basic data, but it is necessary to review it and to send it to the client.
2. Next status will be *accepted*. An invoice will pass to this state once it is reviewed and sent to the client.
3. Then, an invoice will go to the *paid* status when the client pays it.
4. The last possible state will be *rectified*. This will happen when you want to edit an invoice, but it is no longer possible to do it since it has already been submitted to Finance Ministry. In this case the invoice will be marked as rectified and another identical invoice will be created to "rectify" the previous one.

The invoices will have to be numbered following an incremental series for each year, and the rectified invoices will have to follow a parallel series for each year.

As we can see, the invoices' "life" are a bit complex, therefore in this case more features will be necessary to facilitate their management. In particular a user must be able to **request an invoice** (to create it), **edit and print an invoice** (to review and send it), and **rectify an invoice**. In addition, it is obvious that should also be able to **list and delete invoices**.

5.2 Non-functional requirements

Apart from the functional requirements described above, the system will also have to achieve with a series of non-functional requirements to guarantee the correct operation of itself.

1. **Availability:** The server must always be available so that users can use the Intranet and control the cooperative's projects.
2. **Response time:** The response time of the server when the application makes a request should be less than 3 seconds.
3. **Usability:** The intranet UI should be simple and easy to use for any user level. It should also have a quick learning curve, so users can learn to use it quickly.
4. **Robustness:** The data entered by the users can not corrupt the system and it will have to be indicated to them when they introduce incorrect data in the best and fastest way possible.
5. **Testability:** System's tests must be able to easily create, so that we can maintain and apply modifications.

6. Specification

6.1 Actors

As we seen in the Problem formulation section, the current Intranet has several types of users (unidentified users, administrators, employees, and partners), but for this first version only there will be two basic types.
The different actors that will interact with the system will be the following:

Unidentified user: Anyone who accesses the system, but has not yet identified, and therefore the only thing they can do is identify themselves.

Identified user: The identified users will be the ones who will use Intranet every day. They will be in charge of introducing all necessary data to the system in order to be able to use the Intranet and take control of the cooperative's work. In general, people responsible for each project will be the ones that will handle the data of the projects.

6.2 Use cases

This section shows all the necessary use cases to be able to achieve with all the functional requirements that system must have. In particular, diagrams of all the use cases and an explanation in detail of each one of them are shown.

6.2.1 System:

Figure 6.1: System use cases

Use case 1 - Login

The user enters the email and the password in order to access and use the system as a user.

Actor: Unidentified user
Trigger: An unidentified user wants to identify himself to enter the system.
Preconditions:
1. The user is registered to the system.

Main thread:
1. The user opens the web for the first time.
2. The system displays the form to enter the credentials (email and password).
3. The user fills in the form and sends the data.
4. The system validates the data and shows the main page.

Extensions:
4.1. Incorrect credentials

(a) The system shows the form to enter the credentials, but with an error indicating what the problem is.
(b) Return to step 3.

Use case 2 - Logout

The user clicks the logout button to stop using the system.

Actor: Identified user
Trigger: An identified user wants to end the session in a browser.
Preconditions:

1. The user is identified.
2. The user has the web open on any page.

Main thread:

1. The user clicks on the *logout* button.
2. The system ends the user's session and shows the form to enter the credentials.

6.2.2 System administration:

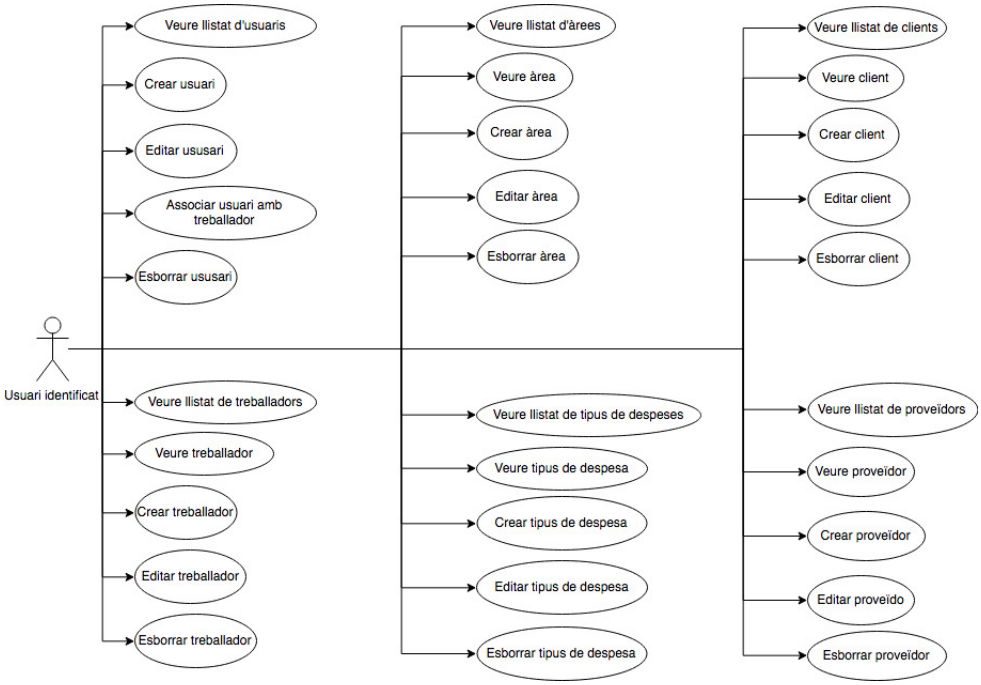


Figure 6.2: System administration use cases

As we can see in the previous graph, in this section there are many use cases. It can easily be seen that we can group use cases with 6 subgroups; those related to users, workers, areas, types of expenses, customers and providers.
Therefore, they will be defined according to this grouping logic.

6.2.2.1 Users

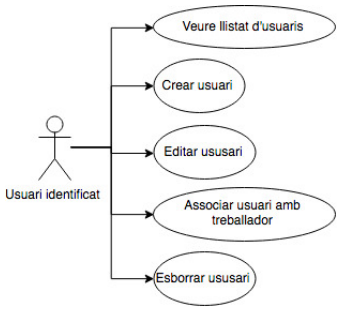


Figure 6.3: Users administration use cases

Use case 3 - Users list

See a list of all the company's users in order to have access to all necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all users registered in the system.
Preconditions:

1. The user is identified.

Main thread:

1. The user goes to the users list page.
2. The system shows the list of all users registered in the system.

Use case 4 - Create user

Register a user, so the information is in the system and can be consulted, and the user can log in.

Actor: Identified user
Trigger: An identified user wants to register another user.

Preconditions:

1. The user is identified.
2. The user is viewing the users list page.

Main thread:

1. The user clicks on "create user" link.
2. The system displays the user registration form (email and password).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new user.
5. The server shows the updated user list.

Extensions:

4.1. There is already a registered user with the same email
   (a) The system displays the user registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.2. The password is not valid
   (a) The system displays the user registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 5 - Edit user

Edit a user's information in order to expand or correct it.

Actor: Identified user

Trigger: An identified user wants to edit the information of another user.

Preconditions:

1. The user is identified.
2. The user is viewing the users list page.

Main thread:

1. The user clicks on "edit user" link of a row in the list.
2. The system displays the user form (email and password).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the user

information.
5. The server shows the updated user list.

Extensions:

4.1. There is already a registered user with the same email
   (a) The system displays the user registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.2. The password is not valid
   (a) The system displays the user registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 6 - Associate user with a worker

Associate some user with a worker, so when a person identified with this user enters hours, they will be done on behalf of the associated worker.

Actor: Identified user

Trigger: A user wants to associate a worker with some user.

Preconditions:

1. The user is identified.
2. The user is viewing the users list page.

Main thread:

1. The user clicks on "associate worker" link of a row in the list.
2. The server displays a list of active and not yet associated employees.
3. The user selects a worker and sends the data.
4. The server updates the user information.
5. The server shows the updated user list.

Use case 7 - Delete user

Delete a user in order to delete information about the sys-

tem.

Actor: Identified user
Trigger: Un usuari identificat vol esborrar un altre usuari.
Preconditions:

1. The user is identified.
2. The user is viewing the users list page.

Main thread:

1. The user clicks on "delete user" link of a row in the list.
2. The server deletes the user.
3. The server shows the updated user list.

Extensions:

3.1 User erase their own user.

(a) The system closes the user's session and displays the Login form.

6.2.2.2 Workers

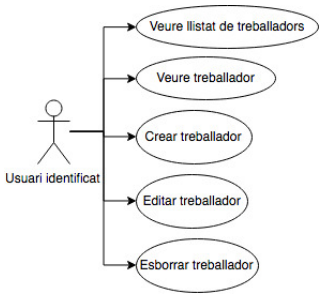


Figure 6.4: Workers administration use cases

Use case 8 - Workers list

See a list of all the workers in the cooperative in order to have access to all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all the workers registered in the system.
Preconditions:

1. The user is identified.

Main thread:

1. The user goes to the workers list page.
2. The system shows the list of all workers registered in the system.

Use case 9 - Create worker

Register a worker, so her information can be consulted, and users can enter hours on his behalf.

Actor: Identified user
Trigger: An identified user wants to register a new worker.

Preconditions:

1. The user is identified.
2. The user is viewing the workers list page.

Main thread:

1. The user clicks on "create worker" link.
2. The system displays the worker registration form (name and surnames).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new worker with *active* status.
5. The server shows the updated workers list.

Extensions:

4.1. Empty data

   (a) The system displays the worker registration form, but showing a message that explains the error.

   (b) Return to step 3.

Use case 10 - See worker

See the file of a worker in order to be able to check all the information on it.

Actor: Identified user

Trigger: An identified user wants to check a worker's file.

Preconditions:

1. The user is identified.
2. The user is viewing the workers list page.

Main thread:

1. The user clicks on "see worker" link of a row in the list.
2. The system shows the worker's file with all the information.

Use case 11 - Edit worker

Edit worker's information in order to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit a worker's information.
Preconditions:
1. The user is identified.
2. The user is viewing the workers list page.

Main thread:
1. The user clicks on "edit worker" link of a row in the list.
2. The system displays the worker edit form (name, surnames, and active/inactive).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the worker's data.
5. The server shows the updated workers list.

Extensions:
4.1. Empty data
   (a) The system displays the worker registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 12 - Delete worker

Delete a worker in order to delete his information.

Actor: Identified user
Trigger: An identified user wants to erase a worker from the system.
Preconditions:
1. The user is identified.
2. The user is viewing the workers list page.

Main thread:
1. The user clicks on "delete worker" link of a row in

the list.
2. The server deletes the worker.
3. The server shows the updated workers list.

Extensions:

2.1 The worker already has assigned hours
   (a) The server does not remove the worker.
   (b) The server shows the updated workers list, but indicating that the worker can not be deleted.

6.2.2.3 Areas

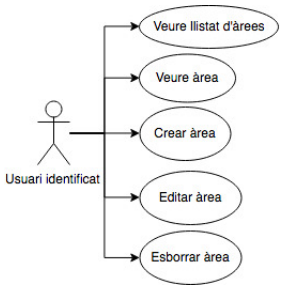


Figure 6.5: Areas administration use cases

Use case 13 - Areas list

See a list of the areas in order to have access to all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all areas.
Preconditions:
1. The user is identified.

Main thread:
1. The user goes to the areas list page.
2. The system shows the list of all areas.

Use case 14 - Create area

Register an area so that its information is in the system and can be associated with projects' batches.

Actor: Identified user
Trigger: An identified user wants to create a new area.
Preconditions:
1. The user is identified.

2. The user is viewing the areas list page.

Main thread:

1. The user clicks on "create area" link.
2. The system displays the area registration form (name and description).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new area with *active* status.
5. The server shows the updated areas list.

Extensions:

4.1. Empty data
   (a) The system displays the area registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.1. Duplicated name
   (a) The system displays the area registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 15 - See area

See the file of an area to know its information.

Actor: Identified user

Trigger: An identified user wants to check the file of an area.

Preconditions:

1. The user is identified.
2. The user is viewing the areas list page.

Main thread:

1. The user clicks on "see area" link of a row in the list.
2. The system display the area's file with all the information.

Use case 16 - Edit area

Edit area's information to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit the information in an area.
Preconditions:

1. The user is identified.
2. The user is viewing the areas list page.

Main thread:

1. The user clicks on "edit area" link of a row in the list.
2. The system displays the area form (name, description and active/inactive).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the area's data.
5. The server shows the updated areas list.

Extensions:

4.1. Empty data
   (a) The system displays the area registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.1. Duplicated name
   (a) The system displays the area registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 17 - Delete area

Delete an area to eliminate its information.

Actor: Identified user
Trigger: An identified user wants to delete an area from the system.
Preconditions:

1. The user is identified.
2. The user is viewing the areas list page.

Main thread:

1. The user clicks on “delete area” link of a row in the list.
2. The server deletes the area.
3. The server shows the updated areas list.

Extensions:

2.1 The area already has associated project batches
   (a) The server does not remove the area.
   (b) The server shows the updated area list, but indicating that the area can not be deleted.

6.2.2.4 Expenses types

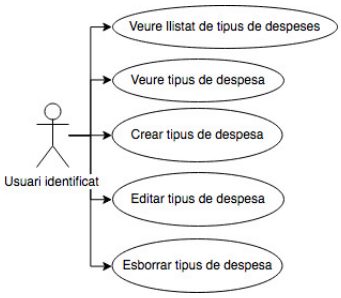


Figure 6.6: Expenses types administration use case

Use case 18 - Expenses types list

See a list of all expenses types that are used in the company in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all expenses types.
Preconditions:
1. The user is identified.

Main thread:
1. The user goes to the expenses types list page.
2. The system shows the list of all expenses types.

Use case 19 - Create expense type

Register an expense type so its information is in the system and a project's expenses can be associated with it.

Actor: Identified user
Trigger: An identified user wants to register a new expense

type.

Preconditions:

1. The user is identified.
2. The user is viewing the expenses types list page.

Main thread:

1. The user clicks on "create expense type" link.
2. The system displays the expense type registration form (name).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new expense type with *active* status.
5. The server shows the updated expenses types list.

Extensions:

4.1. Empty name
   (a) The system displays the expense type registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.1. Duplicated name
   (a) The system displays the expense type registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 20 - See expense type

See the file of an expense type to be able to consult all its information.

Actor: Identified user

Trigger: An identified user wants to check an expense type.

Preconditions:

1. The user is identified.
2. The user is viewing the expenses types list page.

Main thread:

1. The user clicks on "see expense type" link of a row

in the list.
2. The system display the expense type file with all the information.

Use case 21 - Edit expense type

Edit expense type's information to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit an expense type.
Preconditions:
1. The user is identified.
2. The user is viewing the expenses types list page.

Main thread:
1. The user clicks on "edit expense type" link of a row in the list.
2. The system displays the expense type form (name and active/inactive).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the expense type's data.
5. The server shows the updated expenses types list.

Extensions:
4.1. Empty data
   (a) The system displays the expense type registration form, but showing a message that explains the error.
   (b) Return to step 3.
4.1. Duplicated name
   (a) The system displays the expense type registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 22 - Delete expense type

Delete an expense type to delete its information.

Actor: Identified user
Trigger: An identified user wants to delete an expense type from the system.
Preconditions:

1. The user is identified.
2. The user is viewing the expenses types list page.

Main thread:

1. The user clicks on "delete expense type" link of a row in the list.
2. The server deletes the expense type.
3. The server shows the updated expenses types list.

Extensions:

2.1 The expense type already has associated some expense
   (a) The server does not remove the expense type.
   (b) The server shows the updated expense type list, but indicating that the expense type can not be deleted.

6.2.2.5 Clients

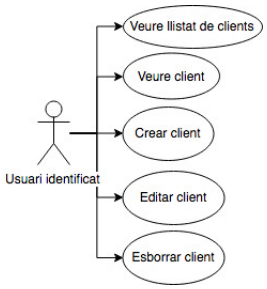


Figure 6.7: Clients administration use cases

Use case 23 - Clients list

See a list of all clients that are used in the company in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all clients.
Preconditions:
1. The user is identified.

Main thread:
1. The user goes to the clients list page.
2. The system shows the list of all clients.

Use case 24 - Create client

Register some client, so its information is in the system and projects and invoices can be associated with it.

Actor: Identified user
Trigger: An identified user wants to register a new client.
Preconditions:
1. The user is identified.

2. The user is viewing the clients list page.

Main thread:

1. The user clicks on "create client" link.
2. The system displays the client registration form (name, cif, address, zip code, and city).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new client with *active* status.
5. The server shows the updated clients list.

Extensions:

4.1. Empty name

(a) The system displays the client registration form, but showing a message that explains the error.

(b) Return to step 3.

Use case 25 - See client

See the file of a client to be able to check all its information.

Actor: Identified user

Trigger: An identified user wants to check a client.

Preconditions:

1. The user is identified.
2. The user is viewing the clients list page.

Main thread:

1. The user clicks on "see client" link of a row in the list.
2. The system display the client file with all the information.

Use case 26 - Edit client

Edit client's information to expand or correct it.

Actor: Identified user

Trigger: An identified user wants to edit a client.
Preconditions:
1. The user is identified.
2. The user is viewing the clients list page.

Main thread:
1. The user clicks on "edit client" link of a row in the list.
2. The system displays the client form (name, cif, address, zip code, city, and active/inactive).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the client's data.
5. The server shows the updated clients list.

Extensions:
4.1. Empty name
   (a) The system displays the client registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 27 - Delete client

Delete a client to delete its information.

Actor: Identified user
Trigger: An identified user wants to delete a client from the system.
Preconditions:
1. The user is identified.
2. The user is viewing the clients list page.

Main thread:
1. The user clicks on "delete client" link of a row in the list.
2. The server deletes the client.
3. The server shows the updated clients list.

Extensions:
2.1 The client already has associated some projects or invoices

(a) The server does not remove the client.
(b) The server shows the updated client list, but indicating that the client can not be deleted.

6.2.2.6 Providers

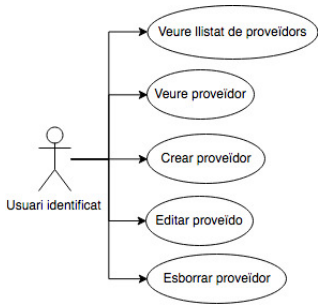


Figure 6.8: Providers administration uses cases

Use case 28 - Providers list

See a list of all providers that are used in the company in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all providers.
Preconditions:
1. The user is identified.

Main thread:
1. The user goes to the providers list page.
2. The system shows the list of all providers.

Use case 29 - Create provider

Register some provider, so its information is in the system and projects and invoices can be associated with expenses.

Actor: Identified user
Trigger: An identified user wants to register a new

provider.

Preconditions:

1. The user is identified.
2. The user is viewing the providers list page.

Main thread:

1. The user clicks on "create provider" link.
2. The system displays the provider registration form (name).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new provider with *active* status.
5. The server shows the updated providers list.

Extensions:

4.1. Empty name
   (a) The system displays the provider registration form, but showing a message that explains the error.
   (b) Return to step 3.

4.1. Duplicated name
   (a) The system displays the provider registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 30 - See provider

See the file of a provider to be able to check all its information.

Actor: Identified user

Trigger: An identified user wants to check a provider.

Preconditions:

1. The user is identified.
2. The user is viewing the providers list page.

Main thread:

1. The user clicks on "see provider" link of a row in the list.

2. The system display the provider file with all the information.

Use case 31 - Edit provider

Edit provider's information to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit a provider.
Preconditions:
1. The user is identified.
2. The user is viewing the providers list page.

Main thread:
1. The user clicks on "edit provider" link of a row in the list.
2. The system displays the provider form (name and active/inactive).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the provider's data.
5. The server shows the updated providers list.

Extensions:
4.1. Empty name
   (a) The system displays the provider registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 32 - Delete provider

Delete a provider to delete its information.

Actor: Identified user
Trigger: An identified user wants to delete a provider from the system.

Preconditions:

1. The user is identified.
2. The user is viewing the providers list page.

Main thread:

1. The user clicks on "delete provider" link of a row in the list.
2. The server deletes the provider.
3. The server shows the updated providers list.

Extensions:

2.1 The provider already has associated some expenses

(a) The server does not remove the provider.

(b) The server shows the updated provider list, but indicating that the provider can not be deleted.

6.2.3 Projects administration:

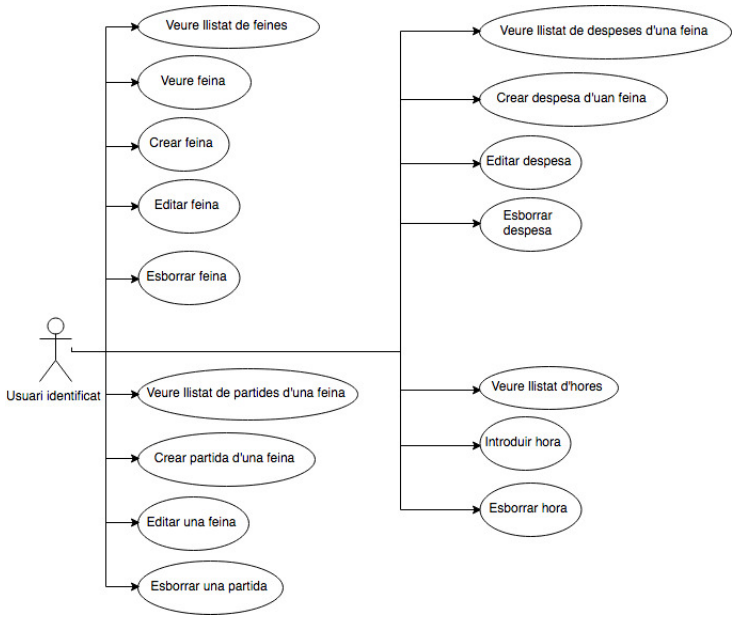


Figure 6.9: Use cases about projects (and related data) administration

As you can see in the Figure 6.9, in this section there are also many use cases. You can also easily see that we can group these with 4 subgroups; those related to projects, projects's batches, expenses and worked hours.
Therefore, they will be defined according to this order.

6.2.3.1 Projects

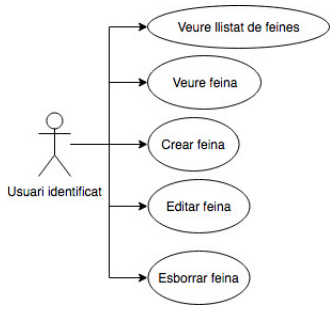


Figure 6.10: Projects administration use cases

Use case 33 - Projects list

See a list of all projects that are used in the company in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all projects.
Preconditions:
1. The user is identified.

Main thread:
1. The user goes to the projects list page.
2. The system shows the list of all projects.

Use case 34 - Create project

Register some project, so its information is in the system and can be managed.

Actor: Identified user
Trigger: An identified user wants to register a new project.
Preconditions:
1. The user is identified.
2. The user is viewing the projects list page.

Main thread:
1. The user clicks on "create project" link.
2. The system displays the project registration form (name, description, client, responsible, project batch's name, area and budget).
3. The user fills in the form and sends the data.
4. The server processes the data and registers a new project associated with the client and responsible pro-

vided. An hourly price of 32€and an entry date of the current day are setted. It also creates a project batch with the name, area and budget given, and associates it with the project.
5. The server shows the updated projects list.

Extensions:

4.1. Name, client, responsible or project batch data are invalid
   (a) The system displays the project form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 35 - See project

See a project's file to be able to check all its information.

Actor: Identified user
Trigger: An identified user wants to check a project.
Preconditions:
1. The user is identified.
2. The user is viewing the projects list page.

Main thread:
1. The user clicks on "see project" link of a row in the list.
2. The system display the project file with all the information.

Use case 36 - Edit project

Edit project's information to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit a project.
Preconditions:
1. The user is identified.

2. The user is viewing the projects list page.

Main thread:

1. The user clicks on "edit project" link of a row in the list.
2. The system displays the project form (name, description, status, hourly price, client, responsible, entry date, finish date, project observations, client observations, and observations for the administrator).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the project's data.
5. The server shows the updated projects list.

Extensions:

4.1. Some mandatory information is empty (name, client, responsible, hourly price, entry date or state)
   (a) The system displays the project form, but showing a message that explains the error.
   (b) Return to step 3.

4.1. Finish date before than entry date
   (a) The system displays the project form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 37 - Delete project

Delete a project to delete its information.

Actor: Identified user

Trigger: An identified user wants to delete a project from the system.

Preconditions:

1. The user is identified.
2. The user is viewing the projects list page.

Main thread:

1. The user clicks on "delete project" link of a row in the list.

2. The server deletes the project.
3. The server shows the updated projects list.

Extensions:

2.1 The project already has associated some expenses, invoices, or worked hours
   (a) The server does not remove the project.
   (b) The server shows the updated project list, but indicating that the project can not be deleted.

6.2.3.2 Projects batches

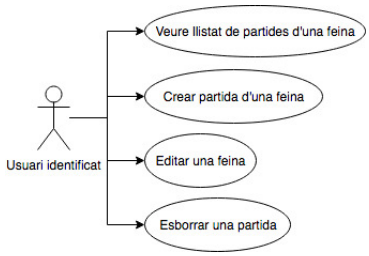


Figure 6.11: Projects batches administration use cases

Use case 38 - Projects batches list

See a list of all projects batches that are assosiated to a project in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all projects batches.
Preconditions:

1. The user is identified.
2. The user is at the project's file page.

Main thread:

1. The system displays a list with all projects batches associated to the current project.

Use case 39 - Create project batch

Create a project batch, so its information is in the system and can be associated with expenses.

Actor: Identified user
Trigger: An identified user wants to register a new project

batch.

Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "create project batch" link.
2. The system displays the project batch registration form (name, area, and budget).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new project batch associated to the project.
5. The server shows the updated project file.

Extensions:

4.1. Empty name, area or budget
   (a) The system displays the project batch registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 40 - Edit project batch

Edit project batch's information to correct it.

Actor: Identified user

Trigger: An identified user wants to edit a project batch.

Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "edit project batch" link of a row in the projects batches list.
2. The system displays the project batch form (name, area, and budget).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the project batch's data.
5. The server shows the updated project file.

Extensions:

4.1. Empty name, area or budget

(a) The system displays the project batch registration form, but showing a message that explains the error.

(b) Return to step 3.

Use case 41 - Delete project batch

Delete a project batch to delete its information.

Actor: Identified user

Trigger: An identified user wants to delete a project batch from the system.

Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "delete project batch" link of a row in the projects batches list.
2. The server deletes the project batch.
3. The server shows the updated project file.

Extensions:

2.1 The project batch already has associated some expense or worked hour

(a) The server does not remove the project batch.

(b) The server shows the updated project file, but indicating that the project batch can not be deleted.

6.2.3.3 Expenses

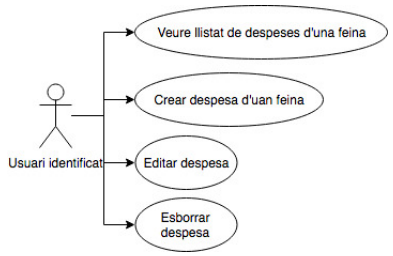


Figure 6.12: Expenses administration expenses

Use case 42 - Expenses list

See a list of all expenses that are assosiated to a project in order to be able to access all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see a list of all expenses.
Preconditions:
1. The user is identified.
2. The user is at the project's file page.

Main thread:
1. The system displays a list with all expenses associated to the current project.

Use case 43 - Create expense

Create an expense, so its information is in the system and can be associated with a provider and to a project batch.

Actor: Identified user
Trigger: An identified user wants to register a new expense.

Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "create expense" link.
2. The system displays the expense registration form (project batch, expense type, provider, estimated budget, real budget, and profit margin).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new expense associated to the project batch and provider.
5. The server shows the updated project file.

Extensions:

4.1. Empty project batch, expense type, provider, estimated budget, real budget, or profit margin
   (a) The system displays the expense registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 44 - Edit expense

Edit expense's information to expand or correct it.

Actor: Identified user
Trigger: An identified user wants to edit an expense.
Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "edit expense" link of a row in the expenses list.
2. The system displays the expense form (project batch, expense type, provider, estimated budget, real budget, and profit margin).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the ex-

pense's data.
5. The server shows the updated project file.

Extensions:

4.1. Empty project batch, expense type, provider, estimated budget, real budget, or profit margin
   (a) The system displays the expense registration form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 45 - Delete expense

Delete an expense to delete its information.

Actor: Identified user

Trigger: An identified user wants to delete an expense from the system.

Preconditions:

1. The user is identified.
2. The user is viewing the project file page.

Main thread:

1. The user clicks on "delete expense" link of a row in the expenses list.
2. The server deletes the expense.
3. The server shows the updated project file.

6.2.3.4 Worked hours

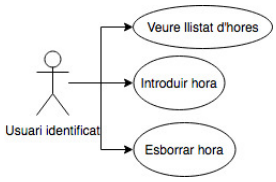


Figure 6.13: Worked hours administration use cases

Use case 46 - Worked hours list

See a list of all worked hours for a day and a worker.

Actor: Identified user
Trigger: An identified user wants to see all the worker's worked hours list for a day.
Preconditions:

1. The user is identified.

Main thread:

1. The user goes to the worked hours list page.
2. The system shows the list of all worked hours for a day (current day by default) and a worker (worker associated to the current user by default).

Extensions:

2.1. The current user ha no worker associated
   (a) The system shows an empty list.

Use case 47 - Enter worked hour

Register a worker's worked hour for a day and a project batch.

Actor: Identified user
Trigger: An identified user wants to enter a worked hour

on behalf of the worker's associated to him.

Preconditions:

1. The user is identified.
2. The user is viewing the worked hours list page.

Main thread:

1. The system shows a form at the end of the worked hours list to enter a new hour (day, project batch, and worked minutes).
2. The user fills in the form and sends the data.
3. The system introduces an hour on the selected date, for the worker that is associated to the current user, for the selected project batch and with a total of a given minutes.
4. The server shows the updated worked hours list.

Extensions:

3.1. Empty date, project batch, or minutes
   (a) The system displays the worked hours registration form, but showing a message that explains the error.
   (b) Return to step 2.

1.1. The current user has worker associated
   (a) The system doesn't show any form to enter hours.

Use case 48 - Delete worked hour

Delete worker's worked hour at a day and project.

Actor: Identified user

Trigger: An identified user wants to delete a worked hour on behalf of the worker associated to himself.

Preconditions:

1. The user is identified.
2. The user is viewing the worked hours list page.

Main thread:

1. The user clicks on "delete worked hour" link of a row in the list.

2. The server deletes the worked hour.
3. The server shows the updated worked hours list.

6.2.4 Invoicing:

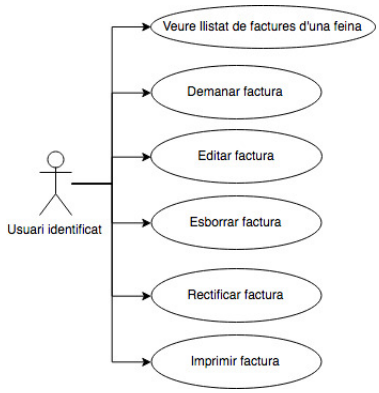


Figure 6.14: Invoices administration use cases

Use case 49 - Invoices list

See a list of all invoices that are associated to some project to check all the necessary information.

Actor: Identified user
Trigger: An identified user wants to see the list of invoices of some project.
Preconditions:
1. The user is identified.
2. The user is viewing the projects list page.

Main thread:
1. The system shows the project's invoices list.

Use case 50 - Request invoice

Register an invoice so it information is in the system.

Trigger: An identified user wants to register a new invoice

for a project.

Preconditions:

1. The user is identified.
2. The user is viewing a projects file page.

Main thread:

1. The user clicks on "create invoice" link.
2. The system displays the invoice registration form (date and amount by project batch).
3. The user fills in the form and sends the data.
4. The server processes the data and registers the new invoice associated with the project and with the same client that already has this project. The invoice will have the status of *requested* and IVA and IRPF setted with the default values. It will also create an invoice number following the annual seriee.
5. The server displays the updated project file.

Extensions:

4.1. The date or some project batch's amoutn are empty
   (a) The system displays the project form, but showing a message that explains the error.
   (b) Return to step 3.

Use case 51 - Edit invoice

Edit an invoice to expand or correct its information.

Actor: Identified user

Trigger: An identified user wants to review a project's invoice to send it to the customer or modify it.

Preconditions:

1. The user is identified.
2. The user is viewing a projects file page.

Main thread:

1. The user clicks on "edit invoice" link in a row from the list of invoices.
2. The system displays the invoice form (date, IVA,

IRPF, client, amount by project batch, and state).
3. The user fills in the form and sends the data.
4. The server processes the data and updates the invoice information.
5. The server displays the updated project file.

Extensions:

4.1. Date, IVA, IRPF, client, or some project batch are empty.
   (a) The system displays the project form, but showing a message that explains the error.
   (b) Return to step 3.

5.1. Year of the date changed.
   (a) The system will generate a new invoice number following the series of the new year.
   (b) Return to step 6.

Use case 52 - Delete invoice

Delete an invoice in order to eliminate its information from the system.

Actor: Identified user

Trigger: An identified user wants to delete a project's invoice.

Preconditions:

1. The user is identified.
2. The user is viewing a projects file page.

Main thread:

1. The user clicks on "delete invoice" link in a row from the list of invoices..
2. The system deletes the invoice.
3. The server displays the updated project file.

Extensions:

2.1. The invoice is at accepted, paid, or rectified status
   (a) The system does not eliminate the invoice.
   (b) The server shows the project file showing a message that explains the error.

Use case 53 - Rectify invoice

Rectify an invoice in order to correct it once it has been already sent to Finance Ministry (and can not be edited).

Actor: Identified user
Trigger: An identified user wants to rectify a project's invoice.
Preconditions:

1. The user is identified.
2. The user is viewing a projects file page.

Main thread:

1. The user clicks on "rectify invoice" link in a row from the list of invoices.
2. The system marks the invoice as a *rectified*.
3. The system creates an identical invoice to the original. This new invoice has another number that will follow the parallel series for the rectified invoices.
4. The server displays the updated project file.

Extensions:

2.1. Invoice is not sent or paid.
   (a) The system does not rectify the invoice.
   (b) The server shows the project file showing a message that explains the error.

Use case 54 - Print invoice

Print an invoice to send it to the customer.

Actor: Identified user
Trigger: An identified user wants to print a project's invoice.
Preconditions:

1. The user is identified.
2. The user is viewing a projects file page.

Main thread:

1. The user clicks on “print invoice” link in a row from the list of invoices.
2. The system displays a window with all the invoice information prepared to be printed.
3. The user selects the browser option to print the page.

6.3 Conceptual model

Conceptual models describe the data structures and restrictions that a system has. Using a conceptual model we can see which elements are involved in the system and the relationships between them.

The general conceptual model for the new Intranet will be shown below with its textual restrictions. A more detailed description of each part of the model will also be made to clarify all relationships.

6.3.1 Conceptual scheme

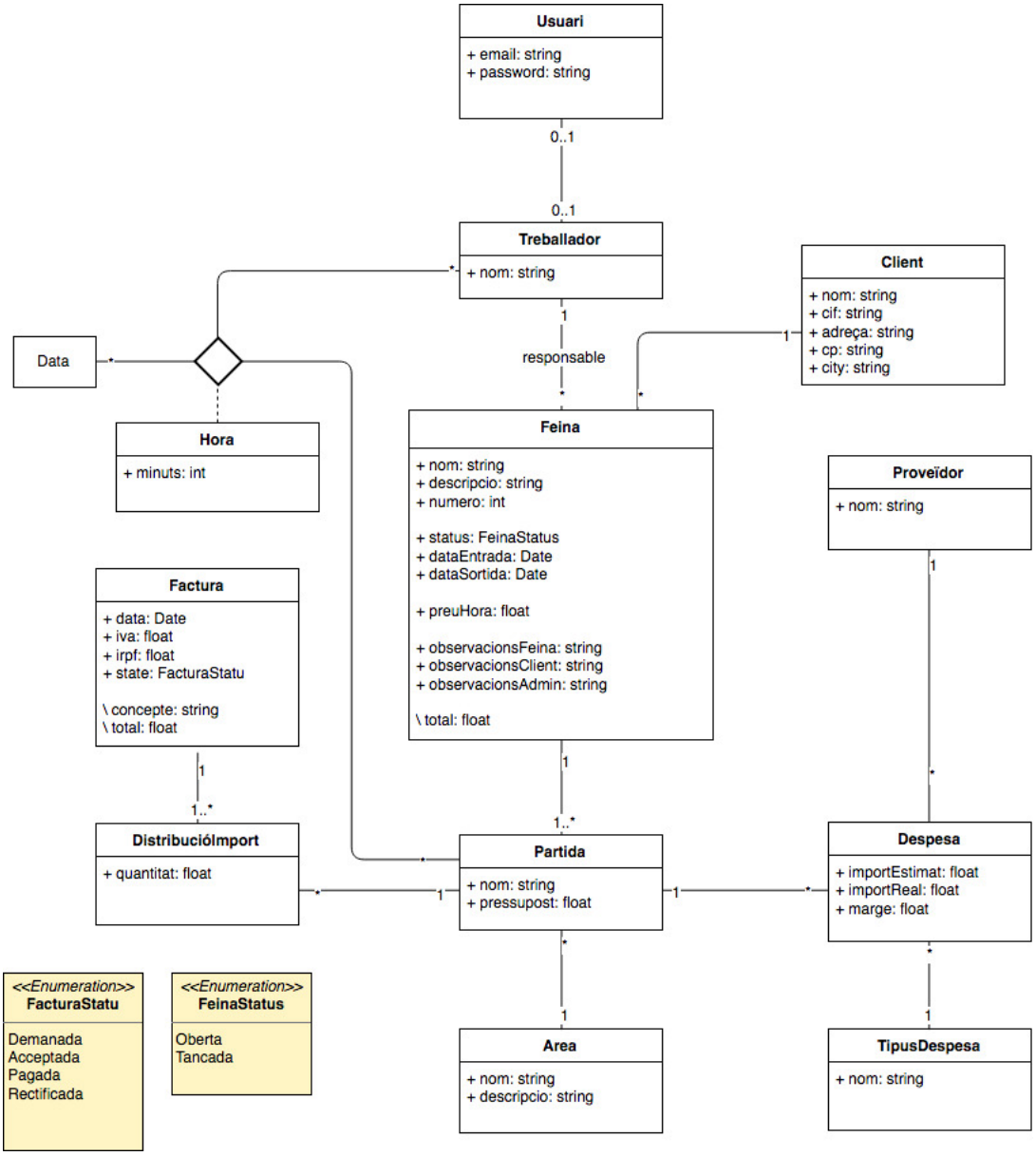
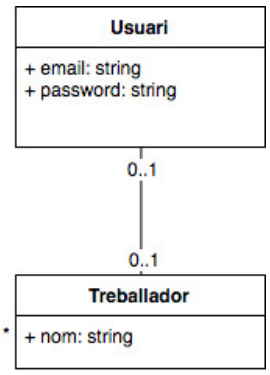


Figure 6.15: Conceptual scheme

6.3.2 Integrity constraints

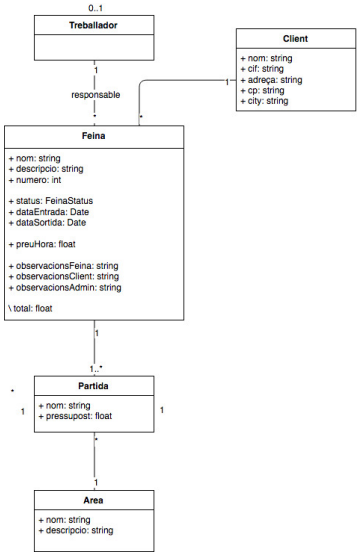
1. The *dataEntrada* of a **Feina** must be prior to *dataSortida*.
2. The *concepte* of a **Factura** is an string made up of *numero* and *nom* of the **Feina**, plus the sum of the *quantitat*s of the **DistribucionsImport**s.
3. For each **Factura** there are as many **DistribucionsImports** as **Partides** in the corresponding **Feina**.
4. The *total* of a **Factura** is the aggregate of *quantitat*s of the **DistribucionsImport**s.
5. The *total* of a **Feina** is the aggregate of *pressupost*s of related **Partides**.

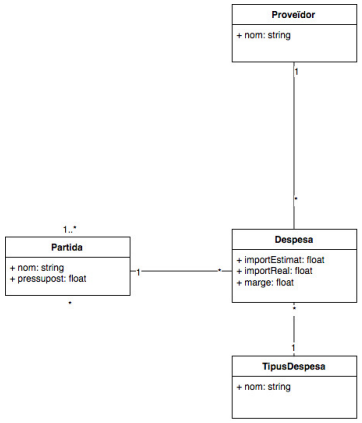
6.3.3 Description



As we can see, in the system, the **Usuari**s and **Treballador**s are separated. In general each **Usuari** will have an associated **Treballador**, but this is not a must. Easily we can see a **Usuari** without an associated **Treballador**, so this user will not be able to enter worked hours. Also, we can found a situation where a **Treballador** is not associated to any **Usuari**, this means that no more worked hours can be registered for this worker (for example, because he no longer works in the company). In addition, in the future it is expected that there will be different **User**s roles, and in particular there will be the roles of administrator and partner, who can enter worked hours for any worker, even if the partners will only have associated one **Worker** and the chief administrator none (because, maybe it's an external accountant).

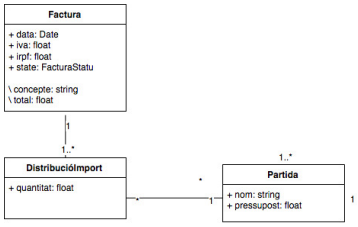
In the next scheme part we can see the most important figure with its most direct relationships. This is **Feina**, which will have an assigned **Treballador** as a responsible. At the same time each project is done for a **Client**, and a **Feina** is organized in one or more **Partide**s. You can also see that each **Partide**s is related to a company's **Area**. This will be useful in the future to extract reports.

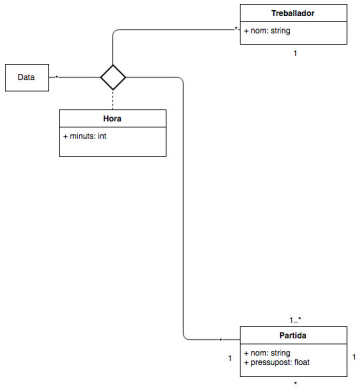




Another relationship with the **Feines** is **Despeses**. These are not directly related to **Feines**, but are associated with **Partides** of a **Feina**. So the **Partides**'s expenses can be consulted, and therefore the **Areas** also can be consulted. As you can see, **Despesa**s are paid to a **Proveïdor**, and has an associated **TipusDespesa**.

Another important structure in the system are the **Factures**. As you can see, **Factures** are composed of one or more **DistribucióImport**. Since **DistributionsImports** are related to **Partides**, we can know which percentage of the **Factura**'s total amount correspond to every **Partida**. So, ultimately, you can know the billing for each **Àrea**.



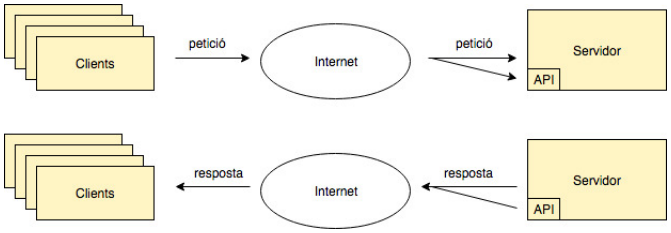


Finally, given a **Feina**'s **Partida** and a **Treballador**, the time spent on **Feina** can be registered for that **Treballador** on a specific day. As you see, in one day, for the same **Treballador** and **Partida** more than one **Hora** can be entered.

7. Design

7.1 Physical architecture

In the context of this Intranet, the typical physical *client-server* web architecture will be used.



As you can see, a client (a web browser of a computer) makes a request for a specific page of the Intranet to the server, and the server responds to the request. The answer is an HTML web page with the needed CSS and Javascript.

It can also be observed that the server has an API, in particular a *REST API*. This is due to the fact that there are certain intranet interfaces that are more dynamic and therefore have been created with a Javascript framework named ReactJs. So, in these cases, the client in the first request downloads the HTML, CSS and Javascript (as mentioned before), but in the pages where a React component is executed, it makes subsequent asynchronous requests through the API to get and send data to the server. The data for these requests are sent in *JSON* format.

Thanks to the logical architecture that has been used (explained in the next section), it is very easy to maintain both types of requests (normal and API).

7.2 Logical architecture

This section will be explained how different system's components are organized and how interact with each other in order to achieve the desired operations. There are many forms of organization, and these make up the different known architectures. These architectures can range from a classic, like *3 layers architecture*, to more evolved concepts such as *Hexagonal architecture*. We could think that they are totally different concepts, but in general, all the architectures share many things, such as the division and grouping of the different components of the system in layers. What varies between each architecture is exactly where these divisions are found and how many are. There are also "rules" about how components must communicate with each other that vary between architectures. Therefore we can define an architecture as a set of rules that define how to separate and organize in layers the different components of the system and how to perform communication between these elements and layers.

Next is the explanation of what these rules are in a Clean architecture. We will see that many of this rules come from other previous architectures, and the fact that what Clean architecture does is to keep and combine the "best" rules of the previous architectures and emphasize some rules of communication between components that in other architectures did not take such importance.

7.2.1 Clean architecture

7.2.1.1 Basic rules

Before explaining how a Clean architecture works, it is necessary to explain some basic rules or principles with which this architecture is based.

First of all, the **SOLID** principles must be explained. SOLID is an acronym, introduced by the same author (Robert C. Mar-

tin), which groups 5 basic object-oriented programming principles. The purpose of this is that developers use these 5 principles as much as possible in order to achieve better software designs. They are very widespread in development environments with agile methodologies and with practices such as TDD.
In a very brief way, the 5 SOLID principles are the following:

- **Single responsability principle**.
  That says that an object should only have one responsibility.
- **Open/closed principle**.
  The concept that software entities (classes) must be opened to (allow) their extension, but must be closed to modification.
- **Liskov substitution principle**.
  This principle says that objects of a program would be able to be replaced by instances of subtypes of these without altering the correct operation of the program.
- **Interface segregation principle**.
  The idea that many specific client interfaces are better than one generic interface.
- **Dependency inversion principle**.
  Which says software entities must depend on abstractions rather than implementations.

As has been said, all principles are important, but Open/Close principle is often considered the most important. If we analyze what this principle says, we will see that it seeks to be able to expand a software without having to modify what has already been done. If this objective is fulfilled, then it facilitates working with agile methodologies, where products are created incrementally. Therefore, we can also see the rest of the principles as a way to facilitate the Open/Close principle.

In order to explain Clean architecture, it is also necessary to emphasize **Dependency inversion principle**, since it plays a key role in this. One way to achieve this principle is by applying a **dependency injection** pattern.

Basically what defines this pattern is that a class can not "instantiate" its dependencies, these must be passed from the outside (inject). This injection can be done in many ways, depending on the characteristics of the programming language, tools, or frameworks that are used, but the two most basic forms are through class constructors methods or as a call parameter.

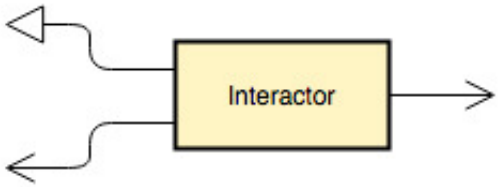
Apart from the SOLID principles, another rule that must be explained before entering to define a Clean architecture, is the **law of dependency**. As mentioned above, a software system has a series of layers with which system objects are organized, and each architecture defines which layers are to be created and where the borders of these layers are. But in the case of Clean architecture, it is also very explicitly defined which dependencies there can be between these layers. As will be seen later, Clean architecture defines a series of concentric layers, where, in the center, there are the most "important" objects, and in the outside, there are the "details". And it is defined that the dependencies between objects of other layers always have to go inwards. Therefore, and as an example, no object in the central layer can know anything external to its layer. This is where **Dependency inversion principle** comes into play, as it will allow us to enforce this law.

7.2.1.2 Interactors, Entities, and Boundaries

As explained in the State of the art section, before talking about *Clean Architecture*, Robert C. Martin wrote about the concept of ***Screaming Architecture***.

The most basic and summarized idea that the author wants to convey with this concept is the fact that a person should be able to know what an application does by looking at folder and files names. In contrast, today if you look at the folders and files names you will, generally, know which programming language is used, which type of application (web, mobile, desktop), which tools are used to develop it (IDEs and / or frameworks), and you can even detect design patterns and architectures easily, but you will not know the most important think; What does this application do?

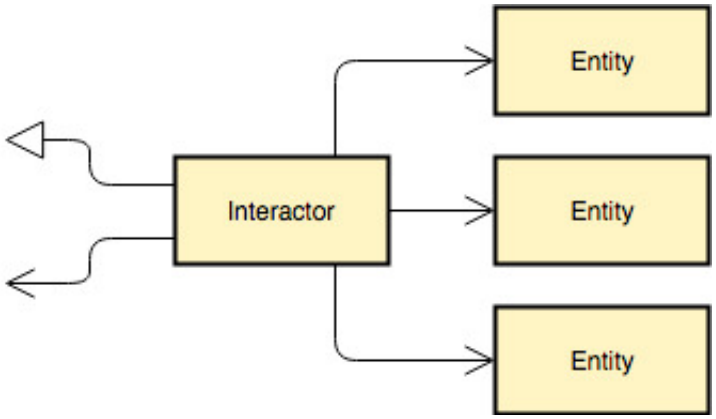
That's why Robert C. Martin talks about the **Interactors**.



In a Clean architecture, the Interactors are a central element. Basically, they are objects that "encode" the *business logics* described in the use cases. Also we can say that the Interacors contain *business logics specific to the application.*
Therefore, if we look at the Interactors names, such as *GetUsersListInteractor, CreateUserInteractor, etc.*, we can have a quick idea of what this application does.

As mentioned previously, in a Clean architecture, business logics specific to the application are in the Interactors. In general these refer to the interactions between users and the system, but not to more generic or high-level business logics.

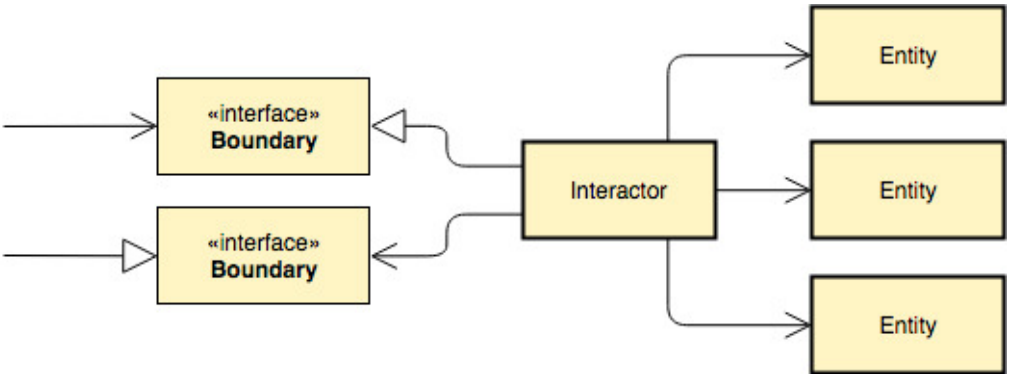
In a Clean architecture, these more generic business logics are in **Entities**.



These Entities are objects that contain generic business logics, and are the most central part of the architecture, which makes any Entity to know nothing about the outside of this layer. Therefore, the Interactors will be in charge of using and coordinating the different Entities to achieve their objectives.

With these two artifacts (Interactors and Entities) we have located all the business logics of our system, and we have managed to locate it in a very specific way and isolate it so that it is as easy as possible to work with it.

Once we have these two layers we need to be able the Interactors to communicate with the rest of the system. One problem we have here is that, for the *dependency law*, the Interactors can not know anything about the rest of the system that is "outside", so, they only have knowledge of other Interactors and of Entities. Therefore, in order to get this communication, the **Boundaries** come into play.

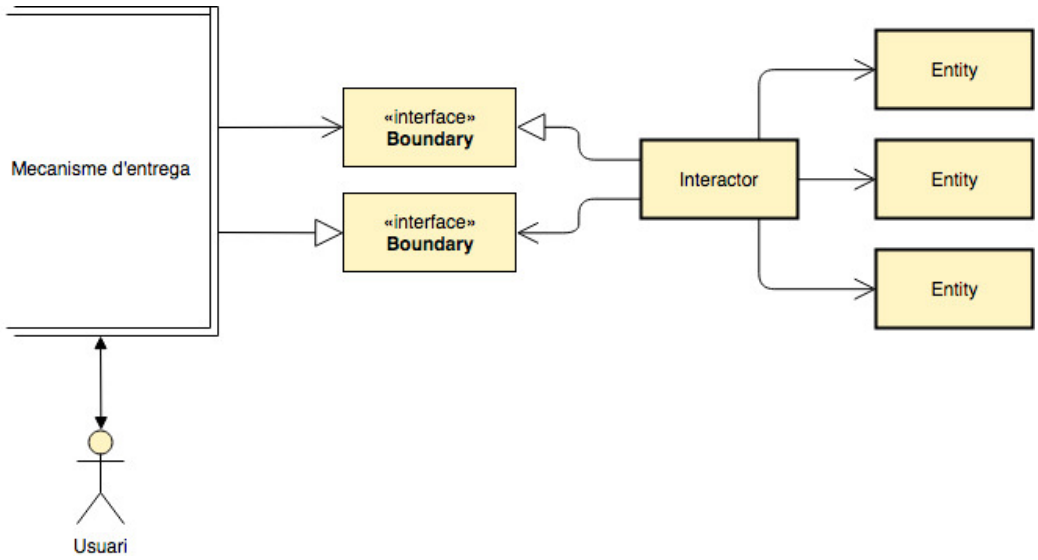


Here is where we use the **Dependency inversion principle**. The Boundaries will be interfaces declared in a layer (Interactors layer) and implemented in another (external layers), thus a communication is achieved between the external and internal layers, and at the same time the Internal layers do not have any outward dependency, which creates the effect of *plugin*; the outer layer becomes a plugin for the internal layer, and is easily interchangeable with another.

In the case of the Interactors, in the previous figure, you can see that a first interface, defined by the Interactor and implemented by some object of the external layers, will be used as a way to communicate from the outside toward the inside. And a second interface, defined in an outer layer and implemented by the Interactor, will be used as a communication channel from inside out.

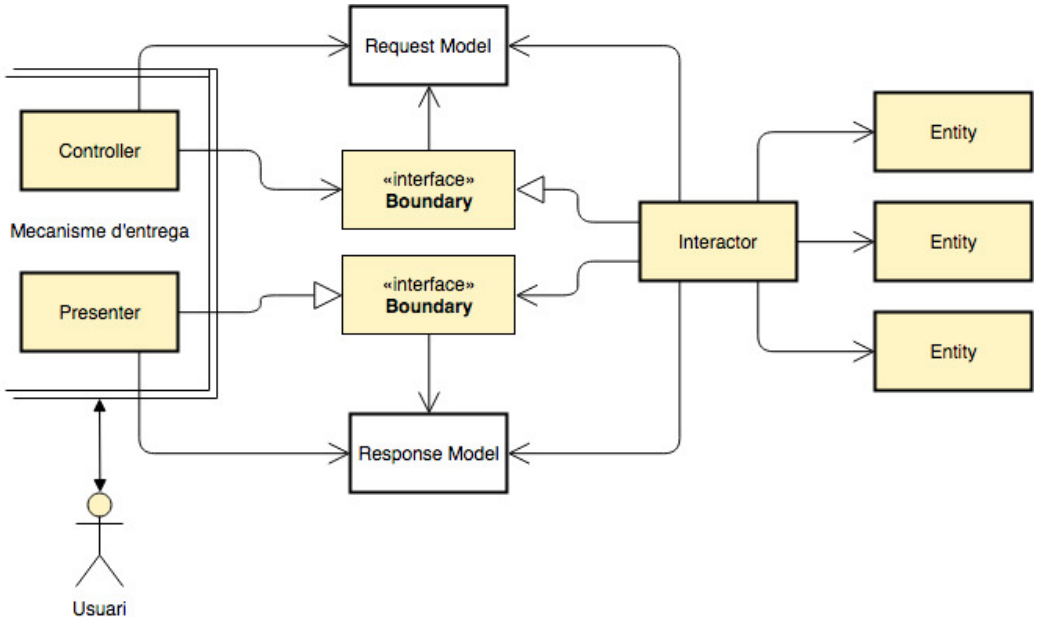
7.2.1.3 Delivery mechanism

With all the architecture explained until now we have two layers where all the business logic of the system are located and a way of communicating with them. There are a par of any application that will be communicated with these layers, and this part is what Robert C. Martin calls *delivery mechanism*.



The delivery mechanism can be any, a website, a mobile application, or even a command line, and this should be a *detail* in our application, since it will surely be one of the parts that change the most over time. This is where we will surely find the typical patterns of MVC or MVP (Model View Controller or Presenter).

A possible implementation of this delivery mechanism could be the following:

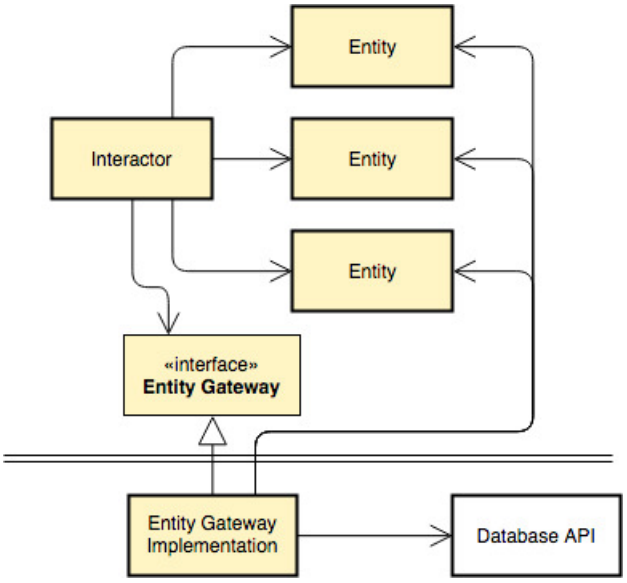


As we can see, we would have a **Controller** that would be in charge of receiving a user request, creating a **Request Model** (which would be a simple data collection) and sending it to the Interactor through the Boundary. The Interactor when receiving the Request Model would read it and execute its business logics along with those of the Entities to end up generating a **Response Model** that would be sent to a **Presenter** through the Boundaries again. And finally, the Presenter would be in charge of showing the final result to the user in the corresponding format.

7.2.1.4 Persistence mechanism

Another very common part is the persistence mechanism. This is the way in which data is saved, so that they are always accessible, usually in a Database. This part may be one of the main features of this architecture, as opposed to other more classic ones, where the Database plays a more central and important role. On the other hand, in a Clean architecture, the persistence mechanism becomes an implementation detail that can be easily interchangeable with another.

The way to achieve this is very similar to the delivery mechanism; basically reusing the dependence law, the dependency inversion principle, and the dependencies injection pattern, we get the following scheme:



As you can see, the Interactor will again get a Boundary injected (which in this case has been called **Entity Gateway**), which is just an interface, whose implementation is in the persistence mechanism layer. The object **Entity Gateway Implementation** will be in charge of accessing the data to generate or save the Entities required for the Interactors. It is clear that there will surely be more than one interface and implementation for

each type of Entities, and here is where some data access pattern (*Active Record, DAO, etc*) will surely be implemented.

7.2.1.5 Summary

With all these parts explained so far, we could say that we have a basic architecture to be able to develop a software system.

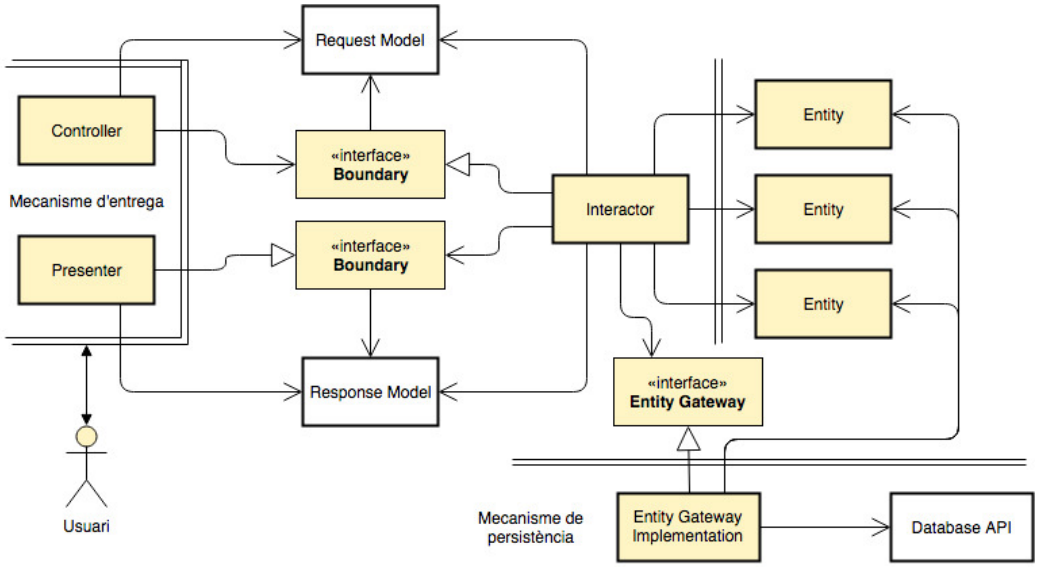


Figure 7.1: Basic class diagram of a Clean architecture

Seeing the previous diagram it is easy to detect 4 layers in the architecture. Starting with the layer that we could say *Domain* which contains the Entities, then we would have the Interactors with the Boundaries, and then at the same level the layers about presentation and persistence.

Seeing the previous diagram it is easy to detect 4 layers in the architecture. Starting with the layer that we could say *Domain*, which contains the Entities. Then we would have the *Interactors* with Boundaries. And last, at the same level, the *presentation* and *persistence* layers.
It is necessary to say that there is no obligation to have these exact same layers always, depending on the requirements of the system, there may be fewer layers (for example we do not want persistence), or have more (in mobile applications we can have

more layers for different device services (GPS, gyroscope, etc.).

It is also common to find the following image as a summary of a clean architecture:

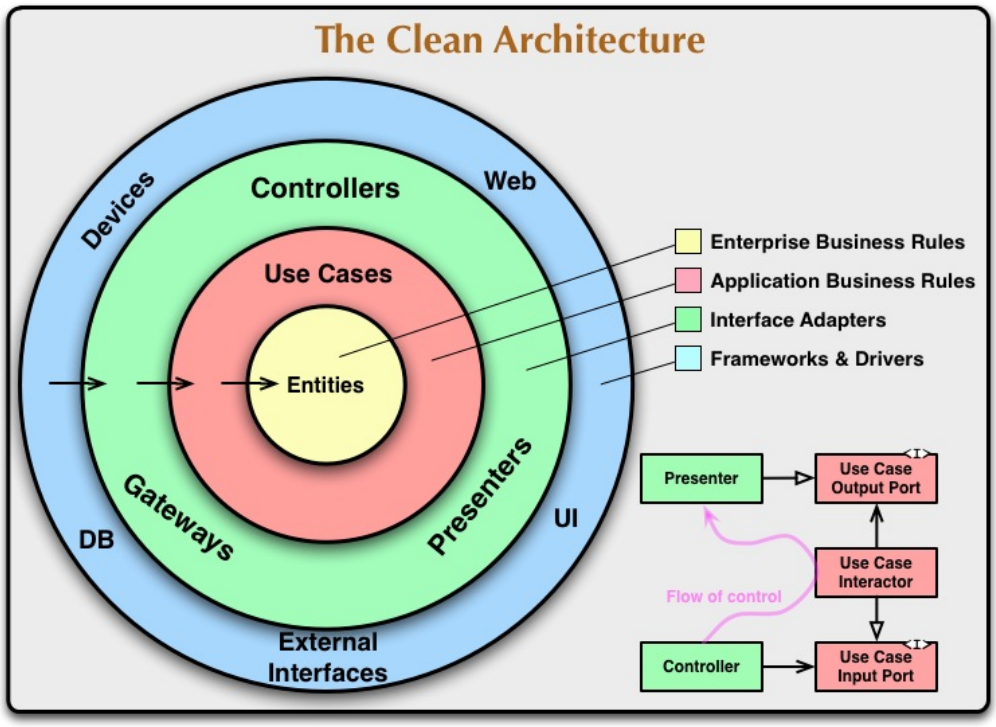


Figure 7.2: Summary diagram of Clean architecture most important parts

As you can see, although the previous image explains the same architecture and also 4 layers are defined, these do not quite match the previous layers explained.

In this case, two inner layers that match the previous ones (Entities and Use Cases, which are the Interactors) are defined, but then is shown a layer with all the implementations of the different Boundaries that can be included in the system, and to finish, a fourth layer where we would find all the most specific implementations of the system (DB, UI, etc).

You can also see that the *Law of dependence* is shown and an example about how to "cross" boundaries between layers without breaking this law.

8. Implementation

In the Design section we have explained how the different elements of a clean architecture are organized in the theoretical field, but it is obvious that when creating a concrete implementation, the final architecture will not be equal to the theoretical. Additionally, by using agile methodologies in this project, we not start the implementation with a final design completely defined, but the product has been built iteration by iteration and still it can not be considered completely finished.

In general terms, the implemented architecture so far is the following:

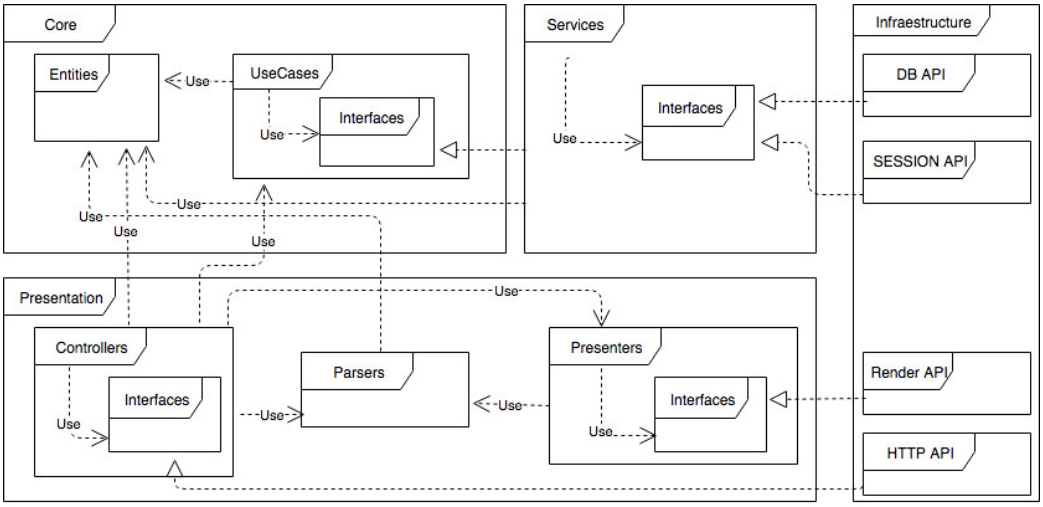


Figure 8.1: Diagram of the implementation of a PHP Clean architecture

Observing the direction of the dependency arrows you can visualize which are the concentric layers and which the external ones.

To begin with, you can see that from the Entities layer there is no out arrow, and all are inward. Therefore it means that this layer is the most central of all (as it should be). And if we continue to

analyze the diagram this way, it is easily seen that the next layer is the UseCases' layer, followed by the Services and Presentation layers, that are at the same level. Finally, the most external layer is about Infrastructure.

Following are the different layers that make up the architecture, and possible improvements to be made in the future.

8.1 Layers

8.1.1 Entities and UsesCases

These two layers, as Clean architecture says, theoretically contain all the objects that encode all business logic. The UseCases layer (Interactors in the theoretical explanation) has a sublayer with the Boundaries (Interfaces) needed for the data stream.
As you can see, these layers have been grouped into a larger **Core** layer, simply to make it more explicit that these layers make up the core of the Intranet.

8.1.2 Services

Following we see that next to *Core* we find the layer of **Services**, which basically contains all the necessary implementations for the interfaces of the UseCases' layer. At the same time, this layer also declares a series of Interfaces to try to "outsource" even more implementation details such as the specific connection to the database, or access to the server session system.

8.1.3 Presentation

Also around the *Core* layer we find the **Presentation** layer. In this layer we can see that we have objects like **Controllers, Parsers and Presenters**, which basically are implementing a

MVPC (Model View Presenter & Controller) pattern. You can also see that in Presenters layer some Interfaces are defined, which will basically be necessary for Presenters to use some type of HTML rendering engine without depending on it.

8.1.4 Infrastructure

Finally, the most external layer is **Infrastructure**. In this layer we find the concrete implementations of the most external details. This is where we will use frameworks and external and internal PHP libraries.

In particular, there is an implementation to connect to the database using the PDO library owned by PHP. In this sense, there is also an implementation to read and write files on disk that was used as a database during the first iterations, and subsequently stopped using it to begin using the implementation that connected to the database without having to touch anything in the Core.
Also, we can find an implementation of a SessionManager that uses the same PHP variables of $SESSION.

There are also all the necessary implementations for the Presenter layer, this is an implementation of an HTML rendering engine using the Twig library and the Symfony framework Form component.

Here we also find the *entry point* of the web service, but it's isolated in a kind of "HTTP API" where we use the Silex framework to simplify the code. This component basically is responsible for reading HTTP requests from the server, and redirecting it to the corresponding Controller. And when it receive a response from the Controller, it generates an HTTP response that returns to the client.

Inside this layer, in particular along with the rendering engine, you find the SASS and Javascript code to generate the CSS styles of the pages and all the Javascript code necessary to dynamize the Intranet, including the components made with the ReactJS framework.

8.2 Testing

As discussed in the Validation Methods section, part of the tests are done manually during the demonstrations at the follow-up meetings and on the day-to-day development. But another very important part is the automatic tests that help to give robustness to the system and ensure that the modifications that are made do not introduce any errors.

To perform these automatic tests, a whole *Test Suite*[1] has been created for unit and integration tests. Thanks to the implemented architecture it has been very easy to create these tests, since the level of coupling between the different elements of the system is very clear it is very easy to create tests where all the dependencies of the tested element are replaced for some type of *TestDouble*[2]. At a more practical level, in the project we can find a test folder, where the whole class structure is replicated but with the tests (apart from some extra helper classes to perform the tests).

As we have said, unit tests have been created for the entire system, excepting at some outer layers. Until now there are 468 unit tests created that performs 844 *assertions* to validate all the integrity of the system.

[1]Test Suite: `https://en.wikipedia.org/wiki/Test_suite`
[2]TestDouble: `https://martinfowler.com/bliki/TestDouble.html`

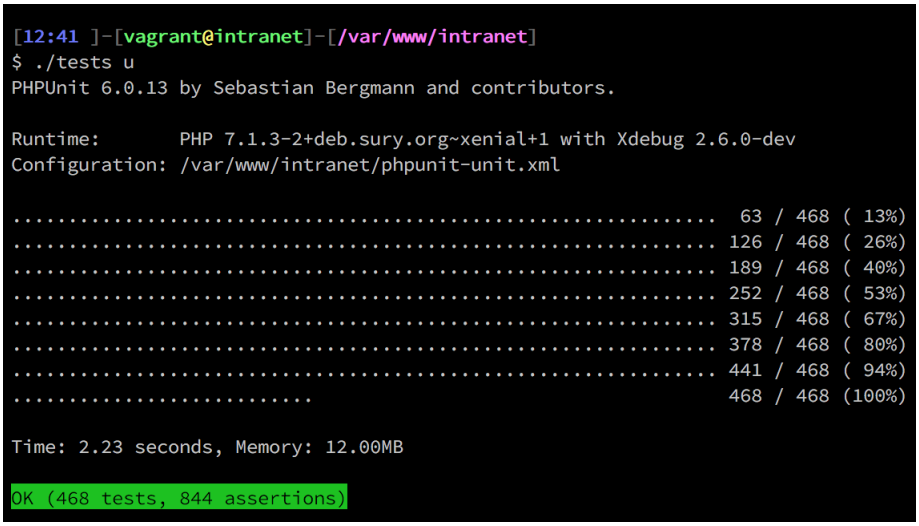


Figure 8.2: Unit tests execution

Integration tests have also been created between the services and infrastructure layers. In total there are 99 tests with 128 *assertions* that validate all transactions with the database.

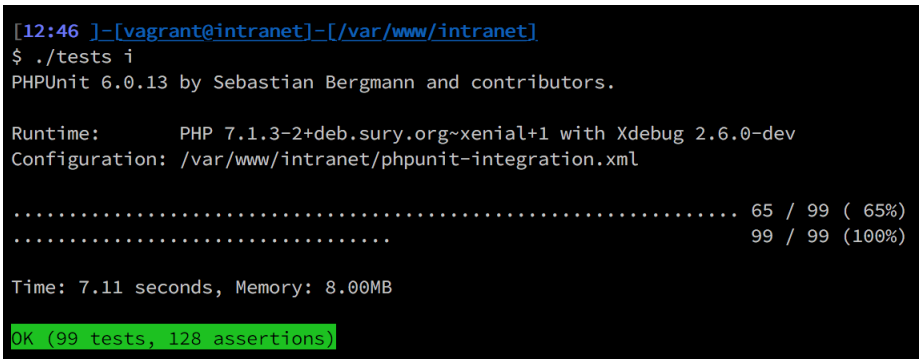


Figure 8.3: Integration tests execution

As seen, the execution of unit tests is much faster (2.23 seconds) than integration tests (7.11 seconds), although the number of unit tests is almost 5 times higher than those of integration. Since we have been using the TDD technique for all development, it is important to have this in mind, since if the execution of the tests is too long, it makes it unfeasible to use this technique correctly.

That's why we created a script that allows you to execute unit or integration tests separately. Thus the unit tests are executed many times, but those of integration only when touching something related to the database. Also, to try to increase the speed of development, the IDE has been configured to be able to execute the tests and check the results quickly and without going through the console.

As you can see, the amount of tests there is quite high, and these will not stop increasing in parallel with the application. That is why it is very important to organize the tests so that it is easy to maintain them, and to interpret them quickly when one fails. The fact of having a test structure that is a replica of the application structure itself greatly facilitates finding the test files when they are searched.

Also, when writing the tests, a nomenclature has always been followed to facilitate its interpretation. In general, in all systems, the tests follow the same pattern; First, the element and its environment are configured to control the initial state, then the behaviour it is tried to test is executed, and finally it is validated that the result is correct. This pattern could be called *Given, When, Then*. That's why the name of all the tests indicates *Given, When, Then* of the test in particular, and internally in the tests you can see very well the separation in the code of these three areas. As you can see in the following image, as a result of applying this nomenclature makes it very easy to know what each test is doing only by reading the name:

```php
public function test_editRequestedInvoice_saveCorrectUpdatedEntityToRepository() {
    //Expect
    $this->invoicesRepository->expects( matcher: $this->once())
                             ->method( constraint: "saveInvoice")
                             ->with( ...arguments: $this->editedInvoice);

    //When
    $this->sut->editRequestedInvoice( request: $this->editRequest);
}

public function test_editRequestedInvoice_returnCorrectUpdatedEntity() {
    //When
    $returnedEntity = $this->sut->editRequestedInvoice( request: $this->editRequest);

    //Then
    $this->assertEquals( expected: $this->editedInvoice,  actual: $returnedEntity);
}

public function testWrongInvoiceId_editRequestedInvoice_throwInvoiceNotFoundException() {
    //Expect
    $this->expectException(InvoiceNotFoundException::class);

    //Given
    $this->editRequest->invoiceId = 999;

    //When
    $this->sut->editRequestedInvoice( request: $this->editRequest);
}

public function testRequestWithEditedDateWithDifferentYear_editRequestedInvoice_updateInvoiceNumber() {
    //Given
    $editedDate              = new \DateTime( time: "2001-01-01");
    $this->editRequest->date = $editedDate;

    $this->invoicesRepository->method( constraint: "generateNextInvoiceNumberForDate")
                             ->with( ...arguments: $editedDate)
                             ->willReturn( value: 35);

    //When
    $editedInvoice = $this->sut->editRequestedInvoice( request: $this->editRequest);

    //Then
    $this->assertEquals( expected: 35,  actual: $editedInvoice->getNumber());
}
```

Figure 8.4: Test nomenclature example

Lastly, as noted in the Validation Methods section, it has been working using *Git* in a way that has allowed to create *Pull Requests* for each implemented feature. And the central repository server (GitHub) was configured to use *TravisCI* service to run the entire test suite on each *Pull Request*. This gives us confidence than when integrating the necessary modifications to implement a new feature, it does not have any error, and we can also be sure that none of the existing features are affected.

8.3 Architecture improvements

The first major improvement refers to objects that depend on Entities. In the theoretical model, the UseCases and the Entities Gateways were the only ones that depended on Entities. More specifically, the entire presentation layer (Controllers and Presenters) did not have any knowledge of the Entities. In the case of the implementation shown, you can see how certain elements of the presentation layer (Controllers and Parsers) are dependent on the Entities. In theoretical architecture, to avoid this, always use *Request and Response Models* between the presentation layer and the UseCases. Since, initially, the Entities of the system were quite simple, the fact to use these models for the data flow between presentation and UseCases meant creating many identical classes, and therefore a lot of code repetition. Because we are not breaking the *law of dependencies* at any time we assessed that until it was not necessary we would leave it this way to maintain simplicity. But perhaps it is time to make this change, since the Entities are beginning to grow.

The last possible major improvement is also between the presentation layer and the UseCases. Basically, it can be seen that the Controllers depend directly on the UseCases, instead of using the Boundaries. Initially, this has also been done to simplify the architecture, since the UseCases initially were very simple. Like before, maybe now is time to make this change.

8.4 Used technologies

Nowadays there are many programming languages to program web services. From Java to Swift, and through many others.

However, in this project PHP has been chosen, since it is one of the most widespread languages in the web world, and has existed for many years. Although it is a scripting language[3] and in its

[3]Scripting language: `https://en.wikipedia.org/wiki/Scripting_language`

beginnings it did not have features to do OOP, with the latest versions have been added many features that allow OOP.

Silex[4] framework is used to streamline some parts of the PHP code. Silex is a micro-framework based on another larger framework that is Symfony[5], so, other Symfony's components are also used in some parts of the project, such as the Twig component[6] as a template engine for PHP. In order to manage all these dependencies that have the PHP code the Composer[7] manager is used, which facilitates the import of the necessary libraries/frameworks.

For the presentation part, being a website, HTML, CSS, and Javascript are used. In order to improve CSS code and make it more readable and maintainable, SASS[8] framework is used. ReactJS[9] framework is used to create the most complex and dinamyc interfaces. As in the PHP part, to manage all the dependencies of this part the Yarn[10] dependency manager will be used. Brunch[11] is also used as a build tool, which basically takes all the CSS and Javascript code and dependencies, prepares it, joins it, and minimizes in order to reduce the final size of the files.

In order to develop all of this, Vagrant[12] is used to create and share a virtual machine with all the necessary configuration to use it as a development environment.

To deploy the app, both in a test and production environments, a server is used. In this server has been installed a Dokku[13] service. Dokku basically allows us to deploy the version we want in the environment we want by making a simple *git push* on the server.

[4]Silex: `https://silex.symfony.com/`
[5]Symfony: `https://symfony.com/`
[6]Twig: `https://twig.symfony.com/`
[7]Composer: `https://getcomposer.org/`
[8]SASS: `http://sass-lang.com/`
[9]ReactJS: `https://facebook.github.io/react/`
[10]Yarn: `https://yarnpkg.com`
[11]Brunch: `http://brunch.io/`
[12]Vagrant: `https://www.vagrantup.com/`
[13]Dokku: `http://dokku.viewdocs.io/dokku/`

9. Resources

We can divide the necessary resources to realize this project in three categories; human, material and software.

9.1 Human Resources

In this project, mainly 4 people will participate:

- **Main developer:** this will be me, with a weekly dedication of 25 hours.
- **Support Developer:** This will be Asier, with a dedication of 4-5 hours per week. His dedication may vary greatly during the project depending on factors such as other company projects (which will reduce hours to this project) or the need to spend more hours on deviations in planning.
- **Comissió Intranet:** formed by two partners of l'Apòstrof (Gemma and Martí) who will perform the role of *Product Owner* of the project. Their dedication will be between 1 and 2 hours a week to make the necessary follow-up and planning meetings.

9.2 Material resources

- **Work places:** where the project will be carried out, mainly in l'Apòstrof office. This includes tables, chairs, meeting rooms, electricity, etc.
- **Computer equipment for development:** the computers that we will use Asier and I to develop the project, and my personal computer to carry out the project's memory.
- **Production server:** where the intranet will be uploaded once it is finished.

9.3 Software resources

- **Development environment:** editor or IDE that each developer will use. Usually will be the *Atom*, *Sublime Text*, or *PHPStorm* editors.
- **Vagrant:** to create a virtual and distributed work to be able that all developers work under the same conditions.
- **Composer:** PHP dependency manager.
- **PHPUnit:** library to perform unit tests for the verification of the source code.
- **Codeception:** library to perform the integration tests for the verification of the system.
- **Git:** code version control system, to manage the source code of the project.
- **Github:** online Git repository server.
- **Trello:** for project management and iterations.
- **Google Drive:** to store all the documentation of the project and to be able to share documents.

10. Planning

It should be taken into account that in this project a methodology very similar to *Scrum* will be used. Therefore the planning will be created creating a list of user stories that will be as independent as possible between them. This is done in order to be able to modify the order at any time if the *Product owner* decides it and also to be able to work on them in parallel.

Therefore this initial temporary plannification can be modified if the *Product Owner* decides at some time to modify the orders described herein. Even so, the final result should be the same.

It should also be taken into account that when using the TDD work methodology, each user history already has the necessary time to carry out the associated tests. And that all the tasks associated with GEP and the creation of the project's memory will be done by myself outside my 25 hours a week of work in ""l'Apòstrof"".

10.1 Calendar

The project lasts 7 months, beginning on February 1 and with a deadline of September 1. Although the defense of the project will be around October, it has been decided to set the data limit in early September to have a margin for possible deviations and to prepare the defense of the project.

10.2 Initial planning

In every project initially a first planning must be done, where it is necessary to analyze the requirements of these, their objectives, and to make estimates of time and costs. Much of this will be done at GEP.

Also, with *Scrum* methodology an initial *Backlog* must also be defined. This *Backlog* is basically a list, sorted by priority, with all the user stories required for the project. Also, in these user stories, a score that is used to estimate the effort to implement them is assigned. This part will be done within the 25 hours of weekly work in ""l'Apòstrof"", it will be done with Asier, and will be reviewed with Comissió Intranet.

10.3 Project iterations

We will have two weeks iterations. A meeting will be done at the beginning of each iteration in order to decide which user stories will be implemented. At the end of the iterations, there will also be a meeting with the Intranet Commission to show the work done and receive feedback.

10.3.1 Iteration 0

In this project an implementation of a Clean architecture must be done with PHP, and this is not the typical architecture that has been seen during the university path. For this reason, in this project we will do an iteration 0 (longer than normal) where a basic user history (user login) will be developed in order to study how to carry out the implementation of the architecture. Obviously in the following iterations, the architecture must be outlined, but in this first iterations I will dedicate more time.

10.3.2 Iteration 1

In this iteration, all the functionalities related to the basic management of the projects will be created. Register, edit, delete and list. We refer to basic management since only the basic data of a project will be managed. Everything related to hours, expenses, invoices will be made later.

10.3.3 Iteration 2

In this second iteration, the concept of project batches will be added. The project file will be modified to be able to list, edit, create and delete project batches.

10.3.4 Iteration 3

Then the features to manage the worked hours (listing, insertion and deletion) will be implemented. Specifically, a list of worked hours per day and worker will be created with a form to be able to introduce more hours into the system.

10.3.5 Iteration 4

In the fourth iteration, the expenses will be added to the project file. In particular, a list of all expenses that a project can have and a form to create new ones will be shown. You can also edit and delete the existing ones.

10.3.6 Iteration 5

Although users already exist in the system, in this iteration features to be able to manage them will be created; user list, registration form and editing, and feature to delete users.

10.3.7 Iteration 6

In the same way that with the users, despite the existence of the concept of worker in the system, all the necessary features to manage them will be created in this iteration; list of workers, forms of registration and editing, and deletion.

10.3.8 Iteration 7

In this seventh iteration, we will continue to add features to manage "system entities". Therefore all the necessary functionalities to manage the areas of the company will be created; list, registration forms and edition, and deletion.

10.3.9 Iteration 8

In this case, all the necessary functionalities to manage the types of expenses will be created; list, registration forms and edition, and deletion.

10.3.10 Iteration 9

In this iteration, the features concerning to management of clients will be created; list, registration forms and edition, and deletion.

10.3.11 Iteration 10

In iteration number 10, features about the management of providers will be created; list, registration forms and edition, and deletion.

10.3.12 Iteration 11

In this iteration we will begin to develop functionalities regarding to the invoices. In particular, a list of invoices will be added to the project file, and a form for a *quick invoice registration* will be created (which will create a new invoice with a *requested* state). The invoice file will also be created, so all invoices parameters can be edited.

10.3.13 Iteration 12

In this iteration we will carry out the rest of features related to the invoicing. In particular, an invoice printing view will be created, and the functionalities of rectifying and deleting an invoice will be implemented. The user page will also be modified to be able to manage which worker is associated with each user.

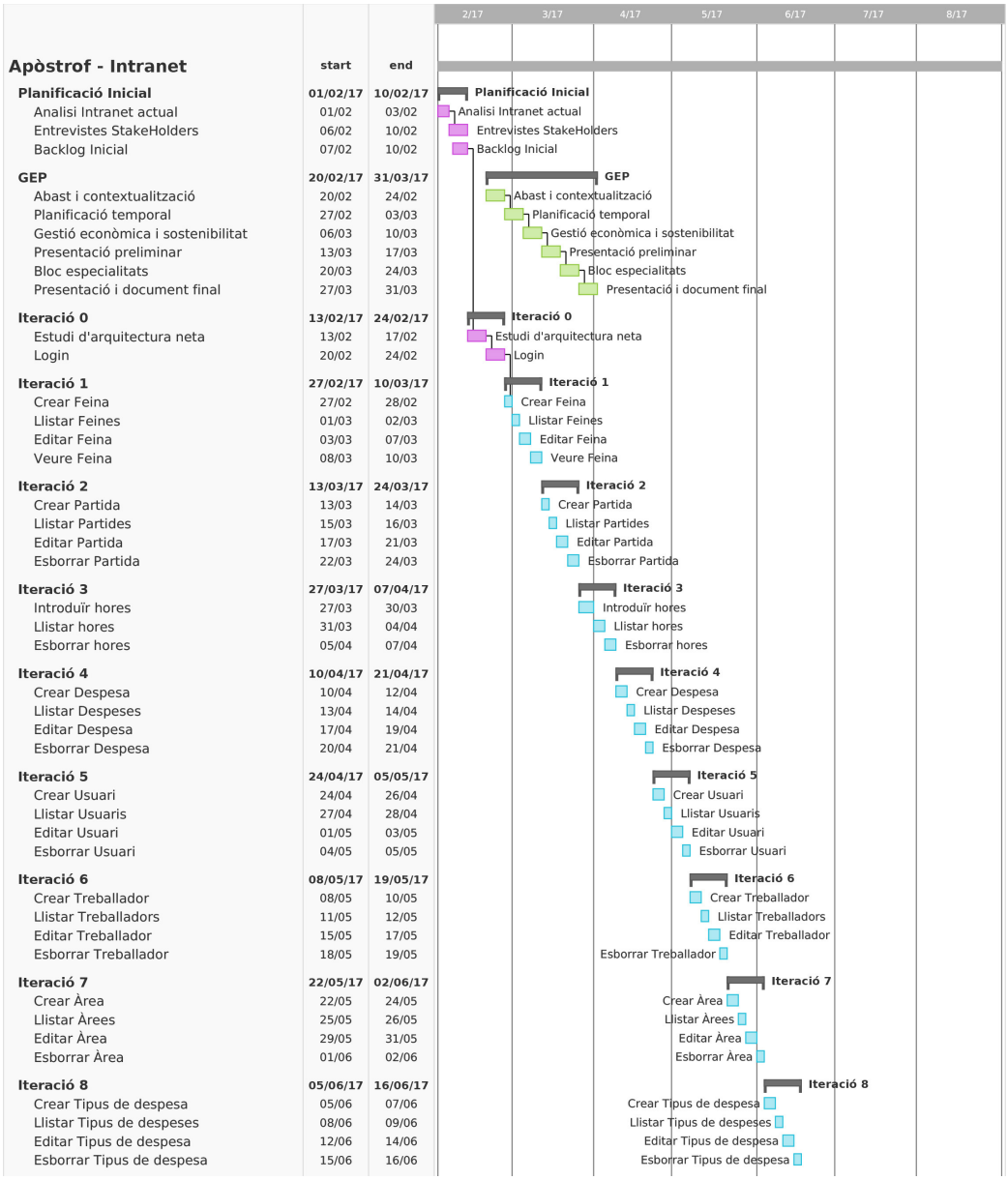
10.3.14 Iteration 13

Finally, we will create the functionality to be able to delete a project, and the logout feature will also be implemented.

10.4 Finalization

When all the iterations are finished, the intranet must be deployed to production. The project's memory and the necessary documentation will also be made.

10.5 Gantt



Apòstrof - Intranet	start	end
Planificació Inicial	**01/02/17**	**10/02/17**
Analisi Intranet actual	01/02	03/02
Entrevistes StakeHolders	06/02	10/02
Backlog Inicial	07/02	10/02
GEP	**20/02/17**	**31/03/17**
Abast i contextualització	20/02	24/02
Planificació temporal	27/02	03/03
Gestió econòmica i sostenibilitat	06/03	10/03
Presentació preliminar	13/03	17/03
Bloc especialitats	20/03	24/03
Presentació i document final	27/03	31/03
Iteració 0	**13/02/17**	**24/02/17**
Estudi d'arquitectura neta	13/02	17/02
Login	20/02	24/02
Iteració 1	**27/02/17**	**10/03/17**
Crear Feina	27/02	28/02
Llistar Feines	01/03	02/03
Editar Feina	03/03	07/03
Veure Feina	08/03	10/03
Iteració 2	**13/03/17**	**24/03/17**
Crear Partida	13/03	14/03
Llistar Partides	15/03	16/03
Editar Partida	17/03	21/03
Esborrar Partida	22/03	24/03
Iteració 3	**27/03/17**	**07/04/17**
Introduïr hores	27/03	30/03
Llistar hores	31/03	04/04
Esborrar hores	05/04	07/04
Iteració 4	**10/04/17**	**21/04/17**
Crear Despesa	10/04	12/04
Llistar Despeses	13/04	14/04
Editar Despesa	17/04	19/04
Esborrar Despesa	20/04	21/04
Iteració 5	**24/04/17**	**05/05/17**
Crear Usuari	24/04	26/04
Llistar Usuaris	27/04	28/04
Editar Usuari	01/05	03/05
Esborrar Usuari	04/05	05/05
Iteració 6	**08/05/17**	**19/05/17**
Crear Treballador	08/05	10/05
Llistar Treballadors	11/05	12/05
Editar Treballador	15/05	17/05
Esborrar Treballador	18/05	19/05
Iteració 7	**22/05/17**	**02/06/17**
Crear Àrea	22/05	24/05
Llistar Àrees	25/05	26/05
Editar Àrea	29/05	31/05
Esborrar Àrea	01/06	02/06
Iteració 8	**05/06/17**	**16/06/17**
Crear Tipus de despesa	05/06	07/06
Llistar Tipus de despeses	08/06	09/06
Editar Tipus de despesa	12/06	14/06
Esborrar Tipus de despesa	15/06	16/06

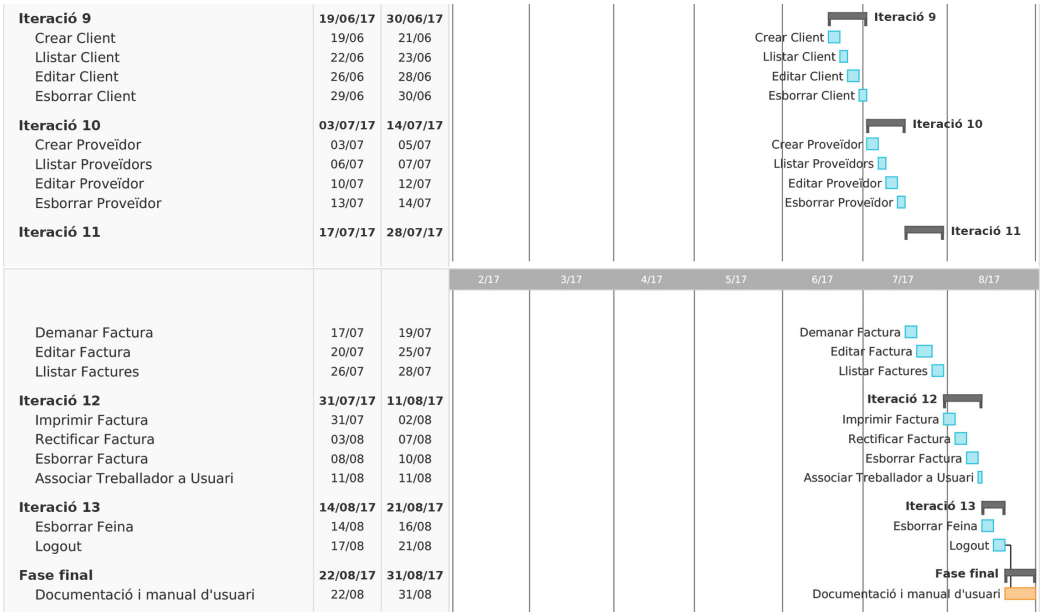


Figure 10.1: Gantt diagram

11. Alternatives and action plan

It is very likely that, like every project, there appear factors that do not allow to carry out the planning previously explained. Generally, these factors are due to a bad estimation of the efforts necessary to implement the different functionalities of the system. There may also be unexpected events that delay the project so that we can not dedicate the planned hours.

11.1 Bad planning

In the case that more hours than planned are required, extra hours will be attempted to recover this time. But if it is not enough, or we can not do these extra hours, then it will be necessary to re-prioritize the *Backlog* in a meeting with the *Comissió Intranet* to try to rule out tasks that can be carried out in the future and stay with the really important and indispensable ones.

11.2 Unforeseen events

It may also be that in "l'Apòstrof" arises the need that I dedicate more hours than those planned to other projects. In this case, we will try to do the same as in the previous section. First, we will try to replace this lack of hours by doing extra hours, and in the case that it can not be done, then we will reprioritize the *Backlog* to try to rule out features that are not trivial and can be done later.

12. Deviations

As has been said, at the beginning of the project many user stories were considered, they were prioritized and a cost estimation was made. Later, they were distributed in iterations to try to foresee when the project would finish. After a few months of carrying out the project we had a few deviations regarding the initial planning.

These deviations have been mainly due to the following reasons:

- **Bad planning**
  The necessary time tto implement the clean architecture was not anticipated. Due to the lack documentation of this architecture, and especially in PHP, we have taken more than expected to find a way we feel comfortable to program all parts of the system.

- **Resources modifications**
  During the project, we were not able to dedicate all the hours that were planned, since we had to carry out other company's projects. In particular, Asier (one of the two developers) has almost failed to do anything about the Intranet, nor do *Code Reviews*. Until reaching the point where Asier left the company, and so, now there is only one developer to complete the project.

- **Changes in technologies**
  Initially, it was not anticipated that we would use the ReactJS framework, but after a month of work, we decided to use this tool to improve the most complex interfaces. After seeing what we took to get acquainted with ReactJS and be able to implement some "final thing", it is obvious that we made a mistake when deciding to introduce this technology, as it has caused us a lot of delay and perhaps we could have done it with some simpler alternative.

12.1 Modifications

The main consequence of deviations has been the need for extra hours on my part to try to implement the maximum possible functionalities for the completion of this project.

On the other hand, and as planned, in case to have problems for any reason to arrive in time to implement all the functionalities of the Intranet, it has been re-prioritized the *Backlog* with *Comissió Intranet* to try to discard tasks that can be carried out later and stay with the really important and indispensable ones.

Therefore unnecessary features have been left out, such as session finishing and deletion of some entities that in practice are almost never deleted (clients, suppliers, projects, etc).

12.1.1 Validation methods

As initially said, the TDD methodology has been used when programming, in order to ensure that everything is correctly tested with unit and integration tests. It was also said that would be *Code Reviews* between the two developers of the project, but since one of the developers has almost not participated in the project, these have not been done at all. As an alternative, I've continued to create *Pull Request* to integrate the changes in each user history implemented, but the code has been reviewed only by myself. A continuous integration server has also been configured in order to ensure that any accepted *Pull Request* will not integrate errors in the system.

13. Budget

13.1 Identification of costs

As in most software projects, this project has 3 types of costs; human resources, hardware and software.

However, since all the software necessary for the development of this project is free, it will not be necessary to do the cost calculation of the *software* section.

13.2 Cost estimates

13.2.1 Human resources

As mentioned above, this project will be developed mainly by one person (I, Ferran Martin), although he will have the occasional help of a second developer (Asier Illarramendi). Also, the people who make up the Comissió Intranet (*Product Owner*) will participate. These will be Gemma Casamajó (who is also the director of the project) and Martí Lázaro.

Because this project is developed in a company, we will summarize the hours worked and what price they have in order to make costs calculations of human resources.

In order to be able to make a good costs estimation, the project has been divided into 4 phases; Initial planning, iteration 0, set of iterations, and final phase.

Asier and I participate in the **initial planning** phase, and this phase lasts for 8 days. In this phase I work 5 hours a day, and Asier collaborates in the last part with a total of 8 hours.

Budget Initial Planning	**Hours**	**€/hour**	**Total price**
Ferran	40	32	1.280€
Asier	8	32	256€
Total			**1.536€**

Table 13.1: Initial Planning Budget

From this phase my dedication to the intranet will be of 20 hours per week. In the meantime, Asier will dedicate 5 hours a week.

The **iteration 0** has been counted aside, since I will work alone, and there are no hours of coordination.

Budget Iteration 0	**Hours**	**€/hour**	**Total price**
Ferran	40	32	1.280€

Table 13.2: Iteration 0 budget

Once the iteration 0 is finished, the **set of iterations** starts, with a total of 13. Coordination and development hours are considered per each iteration.

Coordinating hours include Marti, Gemma, and I. In total there will be 2 coordination hours per iteration.

Asier and I participate in the development hours, and in total there are 48 development hours per iteration.

Budget Set of iterations	**Coord. hours**	**Dev. hours**	**€/hour**	**Total price**
Ferran	2	38	32	1.280€
Asier	0	10	32	320€
Martí	2	0	32	64€
Gemma	2	0	32	64€
Total				**1.728€**

Table 13.3: Set of iterations budget

So, the **total set of iterations cost** will be 1.728€ * 13 iterations = **22.464€**.

In the **final phase** I will participate alone, and I will dedicate 4 days to do it. In this last phase, I will dedicate 5 hours a day.

Budget Final phase	**Hours**	**€/hour**	**Total price**
Ferran	20	32	640€

Table 13.4: Final phase budget

As can be seen, the total hours were based on the estimates made in the planning section and included the hours of meetings and coordination by Comissió Intranet. Therefore the total of everything is as follows:

Budget Human resources	**Preu**
Planificació inicial	1.536€
Iteration 0	1.280€
Set of iterations	22.464€
Final phase	640€
Total	**25.920€**

Table 13.5: Human resources buget

13.2.2 Hardware

The necessary hardware for the development of this project will consist of the computers of each developer and the production server where the intranet will be published. But, keep in mind that, since this server is already being used for other tasks and projects, we will not take it into account in the following cost calculations.

Hardware budget	**Euros**	**Life span**	**Amortization**
iMac 21'	1279€	5 years	255,80€
Macbook pro 13'	1699€	5 years	339,80€
Total			**595,60€**

Table 13.6: Hardware budget

13.3 Management control

With all of the previously calculated we have a budget for a project which goes well to 100%. As this is very difficult to happen, a calculation of possible unforeseen events and contingencies will be calculated below to adjust the budget.

The main unforeseen is time; so more hours of the estimated ones are needed to achieve the project goals. As explained above, if this happens, the first attempt will be to improve the prioritization of the tasks in order to try to discard those that are not considered necessary so that only those that are really necessary and indispensable remain. But in case that more hours were needed, extra hours would be done to compensate. It is estimated that these extra hours could be 10% of the development hours during the *iteration set phase*. Therefore, this would increase the budget at (38 + 10)hours * 13 iterations * 0.1% * 32€/h = **1.996,8€**

Finally, in order to be sure that everything is covered in the budget, a percentage will be applied to the total for **contingencies**, which will be **5%**.

13.4 Total budget

Once all sections are calculated, the total budget is as follows:

Total budget	**Cost(€)**
Human resources	25.920
Hardware	595,60
Unforeseen events	1.996,8
Contingencies	1.425.62 *(5%)*
Total of the project	**29.938,02€**

Table 13.7: Total budget

14. Sustainability

In this section we will analyze the sustainability of this project, we will do it for the economic, environmental, and social dimensions. It should be taken into account that we will only analyze the PPP part of the sustainability matrix, since, as already explained, the scope of this project only reaches this section.

14.1 Economic dimension

During the planning of the project, the costs for human, hardware, and software resources have been taken into account. As we have seen before, human resources will be the minimum to be able to perform the project, as well as hardware resources. In regards to software resources, since everything that is used is free or open source, the cost is 0.

Being an internal project to renew the company's current intranet, it will not generate direct economic profitability for the company. But improving current software does add value to the company indirectly. Also, as mentioned above, it is valued being able to offer this intranet as a service to other cooperatives, but this is beyond the scope of this project.

Therefore, the valuation for the economic part would be 8 out of 10, since the risks were well considered and the project is feasible.

14.2 Social dimension

This project has a direct impact only on the company itself, as it will allow the company to improve its processes and analyze the data in order to improve profitability, productivity, etc. We also want to comment that although "l'Apòstrof" is the only one that receives a direct benefit, its clients also receive it indirectly,

and as "l'Apòstrof" being a cooperative, which is very involved in social projects, we believe that this is an important aspect.

As mentioned before, a future idea is to offer this intranet to other cooperatives, and although this is beyond the scope of this project, it is important to comment this.

With all this, the score for this part would be 7 out of 10, since although being developed in a cooperative, and therefore at the social level has the potential to have a strong impact, there are important functionalities that do not enter within the scope of this project.

14.3 Environmental dimension

As in most software projects, the environmental dimension is affected mainly by issues of energy consumption for workplaces and servers (hardware, lighting, heating, etc.). Apart from this, the use of office supplies (papers, pens, etc) will be minimal. Therefore, the score that the project has in this dimension will be 7 out of 10.

With all this, the **sustainability table** for this project is as follows:

sustainability	**PPP**
Economic	8/10
Environmental	7/10
Social	7/10

Table 14.1: Sustainability table

15. Conclusions

The main project goal was to create a first version of a work management software that would allow the company "l'Apòstrof SCCL" to stop using the current intranet. We can say that this objective has been satisfactorily achieved, despite deviations and modifications to the initial plan. It has been possible to create a first functional, useful and valuable version for "l'Apòstrof", which will continue working to expand it and add non-basic features that now have been set aside.

In the most technical aspect of the software architecture used, this project also had a personal motivation to see how this architecture worked in a PHP web application context. Once this first version of the system is complete, I can say that I am very happy with the results obtained. The architecture has allowed to create a robust system in a simple and easily maintainable and extensible way. I also have to say that the use of this architecture along with *Clean Code* practices make you constantly refactoring and reorganizing parts of the code, so it is essential to have a tool that facilitates this. That is why, when we started the project, we changed the IDE and we started using *PHPStorm* (with a free license) that provides functionalities that facilitate these tasks (especially autocomplete and rename).

This project also allowed me to see and test the Javascript ReactJs framework, which currently has a lot of demand in the sector. Adopting the use of this framework is one of the main reasons for deviations, and therefore it was probably not the best choice. I did not know how to properly assess the learning curve necessary to implement ReactJs components of the necessary size for the Intranet, and seen retroactively, even I think that this topic is big enough for another TFG. However, I am happy to have learned to use this framework, because, among other things, it has also allowed me to experiment with other types of architectures.

15.1 Future work

As mentioned above, in this project we have implemented a first version of an intranet, so there is still a lot of work to do to have a final product.

The planning we make of this project from now on goes through three general stages.

A first step will be to implement a series of very important features, such as the management of workers' work schedules and their time bases, and the automatic generation of reports that will help a lot of decision making within the company.

The second stage will be used to implement a *multicoopertive* layer that will allow access to the Intranet as a service to other cooperatives. Right now, the business model about this is not entirely clear, since we want to look for a model based on the solidarity economy. But, in any case, once we get to this point, getting a return on the investment made in this software will be easier.

And lastly, during the third stage, resources will be allocated to adding less important features, but gradually adding more value to the final product. These features can be an internal warning system, projects comment systems, project management systems, etc. If the system is being maintained, this stage should never be finished, since it will always be necessary to adapt the software to the new needs.

List of Figures

List of Tables