Universitat Oberta de Catalunya

uoc.edu

# FamilyCare Online Application

**Samuel Fernández Amorós**
Grado de Ingeniería Informática
TFG Desarrollo web

**Gregorio Robles Martínez**
**Santi Caballe Llobet**

12/06/2020

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | *FamilyCare Online Application* |
| **Nombre del autor:** | *Samuel Fernández Amorós* |
| **Nombre del consultor/a:** | *Gregorio Robles Martínez* |
| **Nombre del PRA:** | *Santi Caballe Llobet* |
| **Fecha de entrega (mm/aaaa):** | 06/2020 |
| **Titulación:** | *Grado de Ingeniería Informática* |
| **Área del Trabajo Final:** | TFG Desarrollo web |
| **Idioma del trabajo:** | Español |
| **Palabras clave** | *CRM, DDD, Arquitectura Hexagonal* |


**Resumen del Trabajo (máximo 250 palabras):** *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Estudio completo de una aplicación web de la tipología de CRM cuyo enfoque es la gestión de las relaciones de las empresas con sus clientes y el cuidado de las personas de diferente edad que tienen a su cuidado.

Durante el desarrollo se han aplicado metodologías de desarrollo como son Domain Driven Design (DDD) tanto desde la parte estratégica (separación de contextos acotados, identificación de casos de uso, entidades y objetos de valor) así como la parte táctica (uso de arquitectura hexagonal, repositorios, inversión de dependencias, principios SOLID, etc.).

Entre las tecnologías usadas podemos encontrar PHP, MySQL, NGINX, Symfony, Doctrine, Twig, Ansible, Composer, Ansistrano, Terraform, etc.

**Abstract (in English, 250 words or less):**

Complete study of a web application of CRM typology whose focus is the management of the relations of companies with their clients and the care of people of different ages who are in their care.

During development, development methodologies such as Domain Driven Design (DDD) have been applied both from the strategic part (separation of bounded contexts, identification of use cases, entities and object values) as well as the tactical part (use of architecture hexagonal, repositories, dependency inversion, SOLID principles, etc.).

Among the technologies used we can find PHP, MySQL, NGINX, Symfony, Doctrine, Twig, Ansible, Composer, Ansistrano, Terraform, etc.

# Índice

## Lista de figuras

# 1. Introducción

## 1.1. Contexto y justificación del Trabajo

El presente informe tiene como objetivo realizar una propuesta introductoria, presentación de objetivos, hitos y un planteamiento tecnológico que se tendrán en cuenta a lo largo del desarrollo del proyecto final de grado para la construcción de una herramienta online que permitirá a las empresas gestionar las relaciones con sus clientes.

El proyecto se enfoca en el desarrollo de una plataforma online que permitirá gestionar las relaciones con los clientes para empresas que se dedican al cuidado de niños, personas con minusvalías o personas de edad avanzada. La herramienta se enfoca como una plataforma online que sirva como herramienta de trabajo para empresas especializadas en el cuidado de personas desde corta edad como serían niños hasta empresas que se centran en el cuidado de personas de avanzada edad.

Esta herramienta ofrecerá un espacio de trabajo individualizado para cada empresa permitiendo mejorar la privacidad y la seguridad de los datos privados de sus clientes y trabajadores.

La herramienta tratará de centralizar la comunicación con lo clientes y la presentación de calendario de actividades, permitiendo que los familiares puedan interactuar con el centro de una forma sencilla.

Los clientes de las empresas podrán acceder directamente a una ficha personal de la persona que tienen al cuidado para ver diferentes datos como un calendario de actividades, descargar material audiovisual (tanto videos como fotos) o como la creación de notas que el centro podrá gestionar.

## 1.2. Objetivos del Trabajo

El objetivo principal del proyecto es construir un MVP (producto mínimo viable) que tomará como base las características de los productos de tipo CRM (Gestión de relaciones con los clientes) pero que permita facilitar algunos aspectos propios del sector del cuidado de niños y personas.

Así por lo tanto la herramienta permitirá que los clientes de las empresas puedan trabajar en un único espacio de trabajo mediante la identificación en la plataforma y que finalmente usen la herramienta mediante el acceso a la ficha del familiar para tener toda la información necesaria y así como poder comunicarse con la empresa y por lo tanto tendrán toda la información centralizada en un único lugar.

## 1.3. Objetivos secundarios.

1. **Estudiar cuál es el MVP de la herramienta**, toda herramienta CRM tiene un amplio conjunto de características y elementos que proporcionan un cierto valor. Por lo tanto se debe analizar las necesidades mínimas de la herramienta, centrar la búsqueda en cuales son los aspectos mínimos necesarios e imprescindibles a desarrollar.

   Por ejemplo, se puede encontrar que un CRM tipo gestiona ventas o negociaciones entre la empresa que usa la herramienta y sus clientes, para este proyecto esta característica no añade valor, y no se adecua con las necesidades de empresas que tienen como servicio el cuidado de personas de diferentes edades.

1. Crear una **documentación** que tenga como objetivo dar a conocer las características y modo de uso de la herramienta.

2. Usar un **diseño sencillo y fácil usar**, dar un enfoque a la accesibilidad y legibilidad y así como el acceso desde diferentes dispositivos y pantallas, seguir las recomendaciones de W3C.

3. Crear una **planificación temporal** por un periodo de 4 meses de aproximada duracion, debe ser realista y que permita el desarrollo de todas las funcionalidades previstas para el MVP.

4. Durante el desarrollo se debe usar **herramientas que facilite el control de versiones** y el seguimiento del desarrollo, así como el trabajo colaborativo a distancia.

5. Usar servicios de terceros, no crear funcionalidades que puedan ser desarrolladas usando los servicios de terceras empresas. Por ejemplo, gestionar la recepción de emails mediante **Nylas**, o el cobro de suscripciones mensuales y/o anuales usando los servicios de **Stripe**.

# 2. Tecnologias

## 2.1. Introducción

El desarrollo del proyecto tomará como principales tecnologías el uso de **PHP** y en concreto el uso de un framework de trabajo y que tenga una amplia documentación y soporte por alguna comunidad online como es **Symfony.**

Como tecnología de la gestión y almacenamiento de la información se usará **SQL** y en concreto **MySQL**. Además para la gestión de las sesiones de usuario se usará **Redis**. Por la parte del Frontend se usará **React** (que nos permitirá usar **Javascript** en concreto usando la versión de ECMAScript 6 o superior) para gestionar las interacciones de los usuarios con la UI usando Javascript. Sí por lo tanto se usará **HTML** y **CSS** para la maquetación de la herramienta, usando **Bootstrap** que facilitará la creación de un diseño minimalista y sencillo y a su vez tener un buen control del comportamiento **responsive** para diferentes dispositivos.

También es importante destacar que se usará **Docker** para facilitar el trabajo y desarrollo a lo largo del proyecto, así como el uso de **GIT** y una plataforma como gitlab como hacer seguimiento del desarrollo.

## 2.2. Listado

- **PHP 7.4** **[1]**

Se usará el lenguaje PHP para la creación del backend que gestionará la lógica de negocio principal del producto. Usaré la versión más actualizada dado que tiene un alto rendimiento comparado sobre todo con la versión 5.6 o inferior y además permite generar un código más robusto.

- **Symfony 5** **[2]**

Se usará la versión 5 del framework desarrollado con PHP. Es de los frameworks más populares en el ecosistema de PHP y por lo tanto está muy ampliamente documentado

y tiene un gran soporte por la comunidad.

- **MySQL** [3]

Se usará MySQL como motor de base de datos que almacenará lo información más importante del producto mediante uso de tablas. Para trabajar con el sistema de base de datos usaremos un ORM que nos permitirá abstraernos de los detalles de implementación de la base de datos, además de usar el patrón repositorio para separar la lógica de negocio de los detalles de acceso a los datos.

- **Redis** [4]

Se usará como un sistema de gestión de caché y de sesiones. Esto nos permitirá usar sesiones en un entorno distribuido y por lo tanto nos ayudará a tener un sistema que pueda escalar de forma horizontal.

- **Docker** [5]

Permite configurar y montar contenedores que representan los diferentes servicios que el producto usará en un entorno de producción, por lo que podemos emular en nuestro entorno de desarrollo un ecosistema lo más parecido que tendremos a producción.

- **GIT** [6]

Será el sistema de control de versiones. Además tiene el beneficio que es un sistema que permite trabajar de forma distribuida y descentralizada, por lo que si en un futuro el equipo de trabajo aumenta, se podría incorporar al desarrollo de forma sencilla.

- **Ansible** [7]

Es una herramienta que nos permite publicar el proyecto en diferentes servidores en Internet, nos permitir escribir los pasos necesarios que hay que realizar para publicar una nueva versión del proyecto de forma sencilla, segura y ágil. Además se el añadido llamado “ansistrano” que nos facilita configurar todo el proceso de una forma más sencilla.

- **Terraform** [8]

Es una herramienta que permite crear servidores virtuales en diferentes proveedores de tipo PaaS (Plataforma como servicio) como son Amazon AWS, Google Cloud

Platform, Digitalocean, etc. Esta herramienta nos permite escribir todas las reglas necesarias que se deben realizar cada vez que vamos a crear diferentes servidores.

- **Bootstrap 4** [9]

Es un framework de guías de estilo (CSS) y código Javascript que facilitará el desarrollo de la parte frontal (frontend) del producto.

- **Composer** [10]

Es un sistema de orquestación de componentes para el lenguaje PHP. Gracias a este sistema podemos construir en pocos minutos un proyecto nuevo con el framework de Symfony, los componentes de Doctrine u otros muchos componentes que podemos encontrar en repositorios como github o páginas como www.packagist.org.

- **Otros**

Se usan otro conjunto de tecnologías como son Javascript (ECMAScript 6), HTML 5, CSS y SASS para las hojas de estilo, Twig como sistema de plantillas, etc.

# 3. Planificación del Trabajo

Para gestionar la planificación temporal se ha creado un diagrama de Gantt usando la herramienta online teamgantt, donde podemos ver la siguiente planificación desde el día 29 de febrero hasta el 12 de junio.

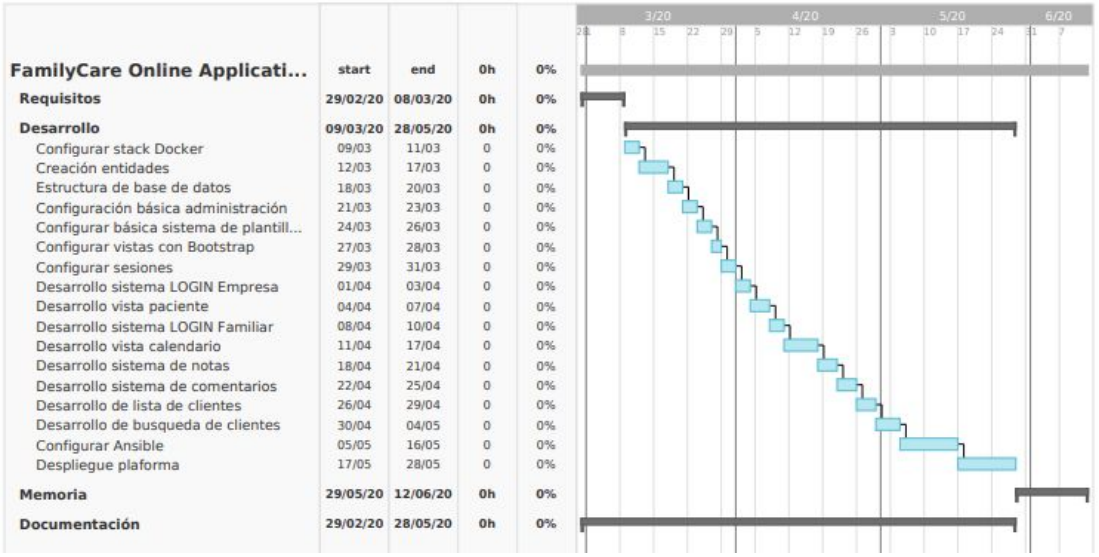


**Fig. 1 - Planificación temporal en la página teamgantt.**

# 4. Evaluación de riesgos

Primer de todo, es importante que la **recogida de requisitos sea real**, ya que si en la recogida de requisitos no se produce de forma realista el producto no recogerá las necesidades mínimas que tendrán los posibles usuarios de la plataforma y por lo tanto tendremos un riesgo real de fracasar en la creación de una herramienta que facilite el trabajo a las empresas.

También la **recogida de requisitos debe ser factible**, ya que si realizamos un recogida de requisitos muy extensa, el desarrollo del producto se verá comprometido y no podrá realizarse en el plazo estimado de 4 meses y estaremos ante una herramienta inacabada.

Por otro lado, durante el desarrollo del producto tendremos que evitar la tentación de generación de un **desarrollo excesivo o de complejidad innecesaria** que dificulte y comprometa los tiempos de desarrollo del resto de funcionalidades.

# 5. Descripción de las tareas

Para la creación del mínimo producto viable (MVP) durante este trabajo final de grado, debemos identificar varios contextos desde diferentes puntos de vista:

- **Gestión de empresa.**
- **Familiares.**

# 6. Requisitos del proyecto

Los requisitos de tipo funcional y no funcional que se esperan que el proyecto cumpla para la generación del MVP serán los siguientes.

## 6.1. Requisitos funcionales

- Las empresas se pueden registrar en el CRM.
- El gestor puede crear nuevos gestores.
- El gestor puede crear trabajadores.
- El gestor puede crear tipos de actividades.

- Trabajadores y gestores pueden crear actividades para fecha concreta y duración.
- Trabajadores y gestores pueden visitar una lista de familiares.
- Trabajadores y gestores pueden crear familiares.
- Trabajadores y gestores pueden visitar una ficha de un familiar.
- Trabajadores y gestores pueden crear notas en la ficha de un familiar.
- Trabajadores y gestores pueden apuntar a un actividad en la ficha del familiar.
- Trabajadores y gestores pueden enviar un email desde la ficha del familiar.
- Trabajadores y gestores pueden visitar una lista de clientes.
- Trabajadores y gestores pueden crear clientes.
- Trabajadores y gestores pueden visitar una ficha de cliente.
- Trabajadores y gestores pueden visitar un calendario con los actividades.
- Los clientes siempre están asociados a un familiar.
- Los clientes pueden visitar la ficha del familiar.
- Los clientes pueden crear una nota en la ficha del familiar.
- Los clientes pueden rechazar la participación del familiar en una actividad.
- Los clientes pueden visitar el calendario del familiar con las actividades.
- Trabajadores, Gestores y clientes pueden identificarse en el CRM.
- Trabajadores, Gestores y clientes pueden recuperar la contraseña en el CRM.
- Trabajadores, Gestores y clientes pueden cerrar sesión en el CRM.
- Permitir cambiar el idioma al inglés.
- Permitir cambiar el idioma al español.

## 6.2. Requisitos no funcionales

- La empresa tendrá sus datos aislados de otras empresas.
- Al crearse una empresa se crea automáticamente un gestor.
- Las direcciones de emails entre trabajadores y gestores no se pueden repetir.
- Usar una herramienta para desplegar el proyecto en los servidores de producción.
- Usar una herramienta para crear y destruir servidores en el proveedor.
- La aplicación detectará los idiomas del navegador del usuario y usará el idioma más adecuado por defecto.
- La aplicación usará el inglés como idioma por defecto cuando los idiomas del navegador no estén disponibles en el sistema.
- Se guardará un hash como contraseña para cada usuario.

# 7. Diseño del proyecto

Siguiendo el enfoque organizativo que se sigue en DDD [14] el proyecto estará diseñado en dos principales contextos:

- **Gestión de empresa.**
- **Gestión de Familiares.**

Estos dos grandes contextos recogen los siguientes actores y casos de uso.

## 7.1. Gestión de empresa

Este contexto recoge las actividades administrativas y de gestión de la empresa dentro del CRM y que se pueden realizar por los actores que están listados a continuación.

- **Actores**

  Los actores que se pueden identificar y ejecutar casos de uso en este contexto son:
  - Gestor de la empresa
  - Trabajador
  - Familiar
  - Cliente

- **Casos de uso**

  Los casos de uso que podrán ser realizados por los diferentes actores mencionados anteriormente son:

  - Registrarse en el CRM.
  - Crear un gestor.
  - Crear un tipo de actividad.
  - Crear un trabajador.
  - Crear un cliente.
  - Crear un familiar.
  - Crear una actividad.
  - Vincular un cliente con un familiar.
  - Listar clientes.
  - Listar familiares.
  - Listar actividades.

- **Entidades**

  Las entidades que podemos identificarse en este contexto son:

  - Empresa
  - Gestor
  - Trabajador
  - Familiar
  - Cliente

- **Objetos de valor**

  Las entidades que podemos identificarse en este contexto son:

  - Contraseña.
  - Dirección de email.

- **Diagrama de casos de uso**

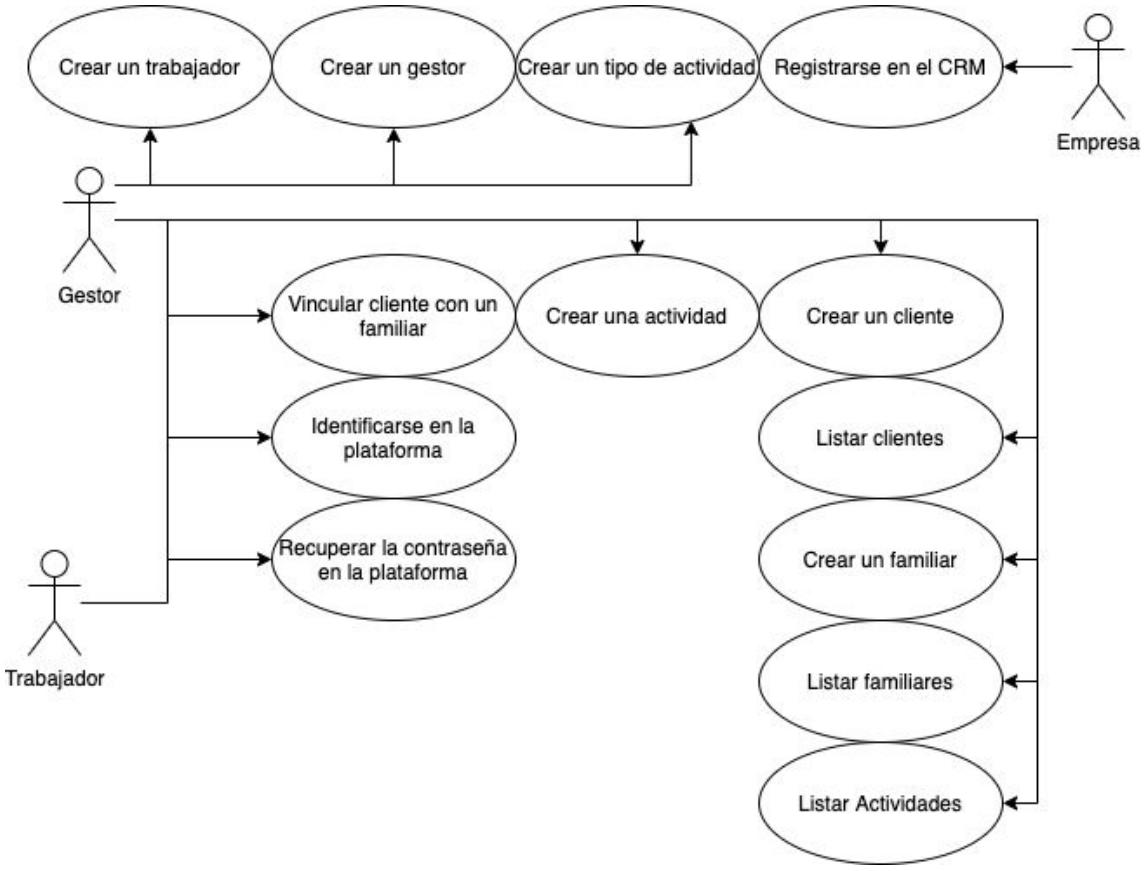


**Fig. 2 - Diagrama de casos de uso desde el punto de vista de la gestión de empresa**

## 7.2. Familiares

Este contexto recoge las actividades relacionadas con el familiar que tienen a su cargo por parte de los clientes que han contratado los servicios de cuidados de personas, por lo tanto familiares del cliente como los trabajadores de la empresa, pueden usar el CRM para organizar el día a día del familiar. A continuación se lista los actores y casos de uso.

- **Actores**

Los actores que se pueden identificar y ejecutar casos de uso en este contexto son:

- Gestor de la empresa
- Trabajador
- Familiar
- Cliente

- **Casos de uso**

Los casos de uso que podrán ser realizados por los diferentes actores mencionados anteriormente son:

- Ver el calendario de actividades del familiar.
- Crear nota en la ficha del familiar.
- Crear nota privada en la ficha de un familiar.
- Enviar email desde la ficha del familiar a los clientes.
- Apuntar familiar en una actividad.
- Rechazar una actividad para un familiar.
- Ver ficha del familiar.

- **Entidades**

Las entidades que podemos identificarse en este contexto son:

- Gestor
- Trabajador
- Familiar
- Cliente
- Nota
- Actividad

- **Objetos de valor**

Las entidades que podemos identificarse en este contexto son:

- Dirección de email.

- **Diagrama de casos de uso**

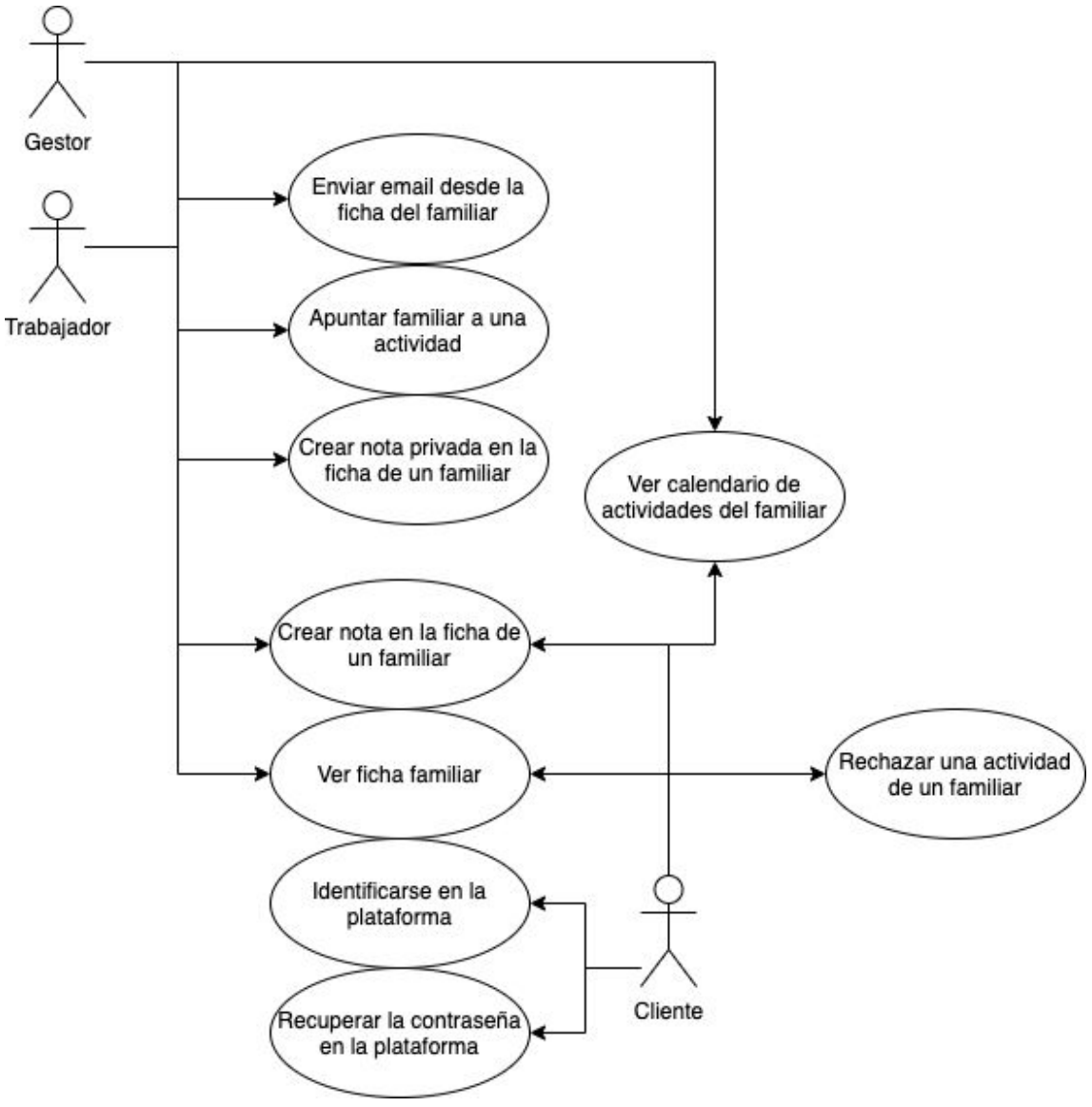


**Fig. 3 - Diagrama de casos de uso desde el punto de vista de la gestión del familiar.**

## 7.3. Contexto común

Además, existen un conjunto de casos de uso que son comunes para todos los contextos que son:

- Identificarse en el CRM.
- Recuperar la contraseña en el CRM.
- Cerrar sesión en el CRM.

## 7.4. Diagrama de clases

Tras analizar los diferentes contextos con sus casos de uso, entidades y objetos de valor podemos presentar el siguiente diagrama de clases:

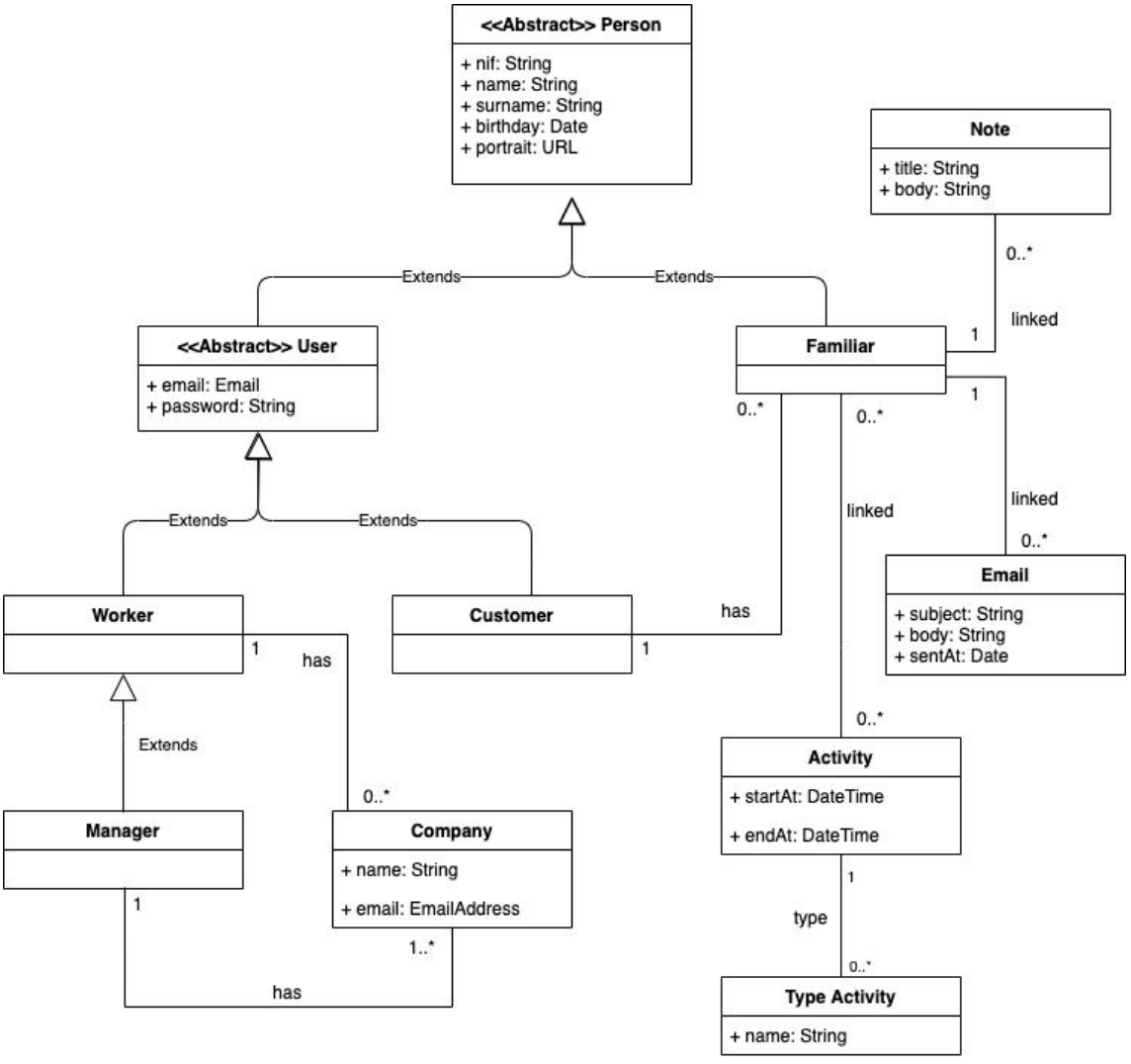


**Fig. 4 - Diagrama de clases de la aplicación.**

## 7.5. Diagrama de base de datos

### 7.5.1. Base de datos del espacio de trabajo.

A continuación se presenta el diagrama de la base de datos que cada espacio de trabajo tendrá.

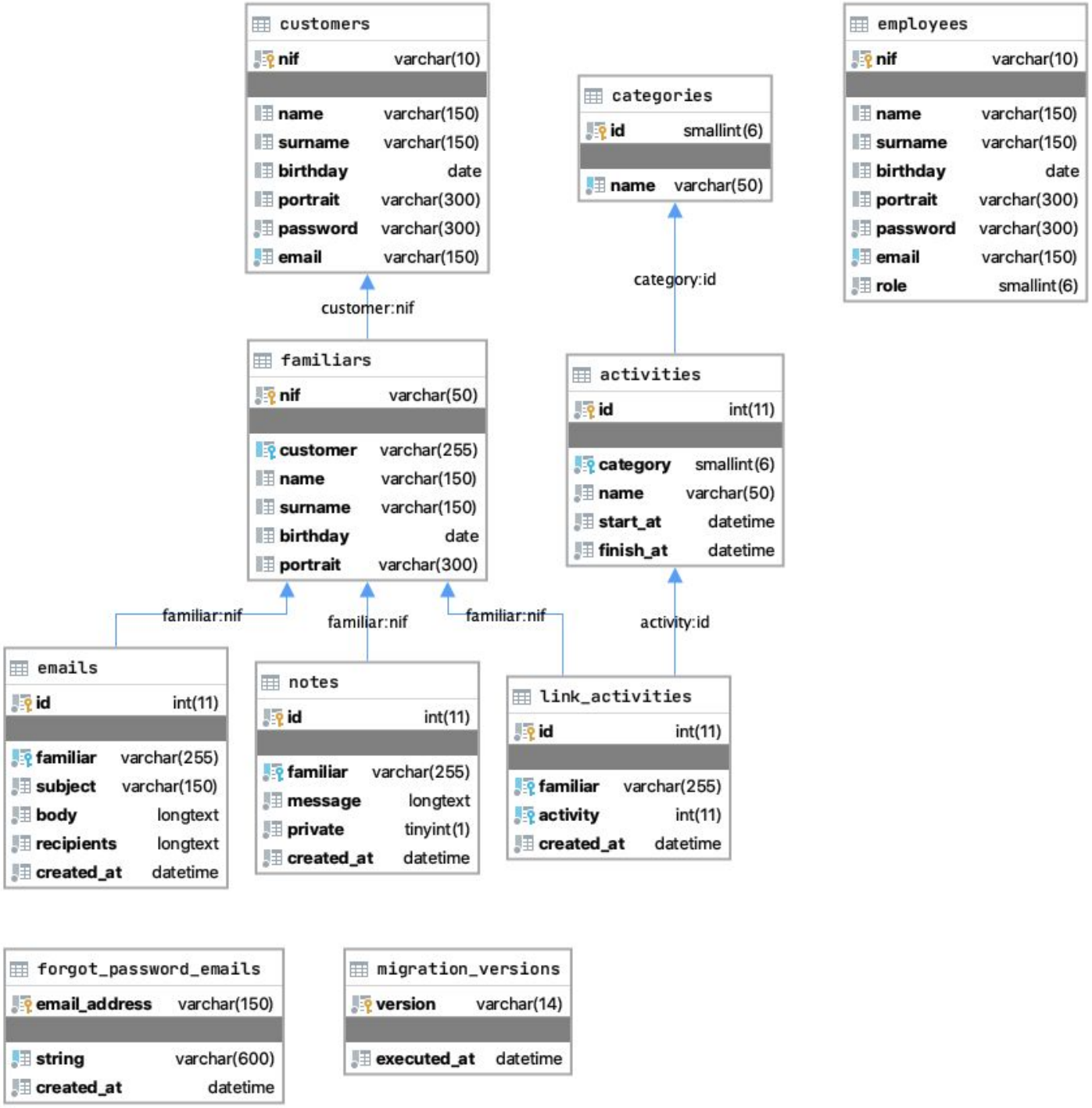


**Fig. 5 - Diagrama de la base de datos del espacio de trabajo de la empresa.**

Se pueden destacar los siguientes detalles:

- **Tabla employees**: Almacena la información tanto de gestores como empleados. Es importante almacenarlos en la misma tabla para tener ciertos controles, entre ellos podemos destacar, que no se repita el NIF o la dirección de email. Se usa un campo llamado **"role"** para diferenciar entre cada tipo de empleado.

- **Tabla customers**: La información de clientes se debe almacenar, esto permite la situación en la que un empleado es a la vez también un cliente y por lo tanto se le puede gestionar de diferentes formas en cada momento dentro de la plataforma.

- **Tabla link_activities**: Esta tabla surge a partir de la necesidad de poder vincular los familiares con las actividades que se programan en el espacio de trabajo y que recoge la relación entre un familiar y una actividad determinados.

- **Tablas emails, notes**: Las tablas están relacionadas con un familiar determinado y se usan para generar el hilo de actividad de cada familiar dentro de la ficha, para ello se hace uso del campo fecha y hora llamado "created_at".

- **Tabla migration_versions**: Esta tabla es proporcionada por el framework de trabajo de Symfony junto con el uso de otro componente llamado Doctrine, usado para la gestión de acciones con la base de datos, esta tabla permite tener un control de qué cambios se van realizado en la estructura de la base de datos a lo largo del tiempo y por lo tanto podemos saber que futuros cambios nos faltan aplicar.

- **Uso de "smallint"**: En la tabla "categorie" se opta por usar el tipo de campo "smallint" ya que el la cantidad de registros que posiblemente se pueda almacenar a lo largo del tiempo de uso del proyecto será razonablemente baja, no esperando que se puedan crear más de 20.000 tipos de categorías. Por lo tanto el espacio de datos ofrecido para un "smallint" será suficiente, siendo exactamente 65.535 registros al no usar el signo.

### 7.5.2. Base de datos central

Por otro lado existe una base de datos central que almacena la información mínima para saber cada espacio de trabajo disponible. Esto nos permite a partir del nombre del espacio de trabajo redireccionar al usuario y además evitar que existan espacios de trabajo con el mismo nombre.

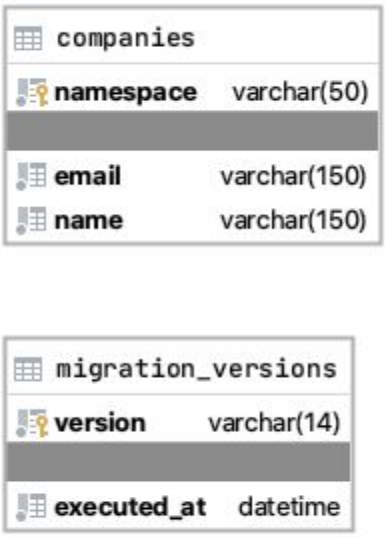


**Fig. 6 - Diagrama de la base de datos central de la aplicación.**

### 7.5.3. Versionado de la base de datos

Gracias al uso del framework de trabajo de Symfony junto con el componente llamado Doctrine, que proporciona funcionales adicionales para facilitar el trabajo de desarrollo del producto, se puede ofrecer un sistema de control de versiones de la base de datos.

Por un lado, el sistema usa una tabla en la base de datos para guardar cada anotación de cambios que el programador a decidido agrupar en un único fichero, esto permitirá tanto aplicar cambios hacia adelante como hacia atras (deshacer cambios).

Esta funcionalidad resulta muy importante en un proyecto donde existen multitud de base de datos (que serán los espacios de trabajo de los usuarios del CRM) y por lo tanto se puede gestionar de una forma sencilla en qué versión se encuentra cada una de las bases de datos.

Por lo tanto en el directorio raíz se puede observar la carpeta “database” que tiene otra subcarpeta con los diferentes cambios que hemos ido produciendo a lo largo del tiempo.

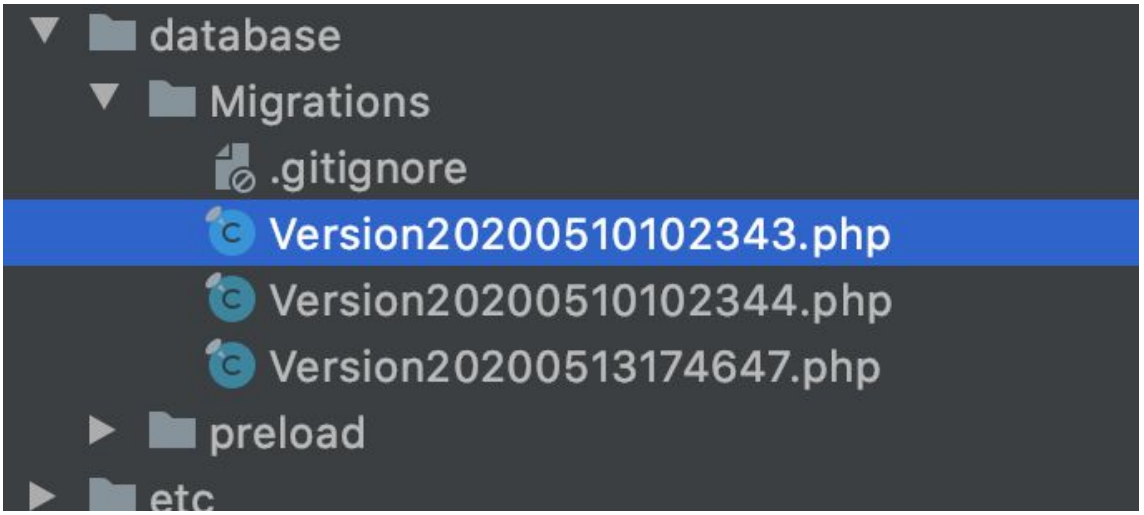


**Fig. 7 - Ejemplos de cambios o migraciones de la base de datos generados durante el desarrollo.**

En el ejemplo podemos ver que se han producido tres cambios desde el inicio del proyecto. Además podemos ver las fechas de creación de cada uno de los cambios.

## 7.6. Diagrama de la infraestructura del sistema

En el siguiente diagrama mostramos como se engloban los diferentes elementos de la infraestructura del sistema. Cuando el usuario (tanto cliente como empleado) interactúa con el sistema, lo verá como un todo completo, y no conoce los diferentes componentes de la infraestructura que se relaciona por dentro para servidor las solicitudes del usuario.

El usuario en todo momento verá mediante su navegador web una página generada con HTML, CSS y Javascript. Este conjunto se encuentra formado por una aplicación de frontend que muestra la información al usuario y que manda las acciones a la infraestructura mediante uso del protocolo HTTP o si se quiere usar dar una mayor seguridad a la comunicación, mediante HTTPS.

Las solicitudes HTTP/S llegarán al sistema, que son recogidas por un servidor especializado en la comunicación HTTP, este a su vez se comunica con la aplicación de Backend que será la encargada de recoger la solicitud inicial del usuario, procesarla mediante la lectura de datos de sesión del usuario y la lectura de la información solicitada el usuario en la base de datos.

Además el sistema hace uso de un servicio externo para realizar las notificaciones vía email, para ello hace uso del protocolo de comunicación SMTP para contactar con el servidor del cliente y que almacene un nuevo email.

### 7.6.1. Componentes de un cluster

Por lo tanto, podemos destacar que cada cluster en la red estará formado por los siguientes componentes:

- Un servidor NoSQL mediante el uso de Redis.
- Un servidor SQL mediante el uso de MySQL.
- Un servidor HTTP/S que mediante el uso de NGINX.
- Un pequeño conjunto de servicios que usan PHP.

Para la puesta apunto del MVP es necesario tener como mínimo un cluster que ofrece servicio a las solicitudes de los usuarios, pero que se puede escalar de forma sencilla tanto de manera horizontal (añadiendo nuevos clones de diferentes componentes, con algunas posibles limitaciones) como de manera vertical (actualizando los componentes por otros componentes más potentes).

La infraestructura de red también se puede escalar mediante la preparación de cluster especializados para cada cliente (por cada espacio de trabajo) permitiendo segmentar

el tráfico y la carga global, siendo una configuración recomendada en una etapa cuando el producto se ha validado y se ha detectado que hay un producto viable.

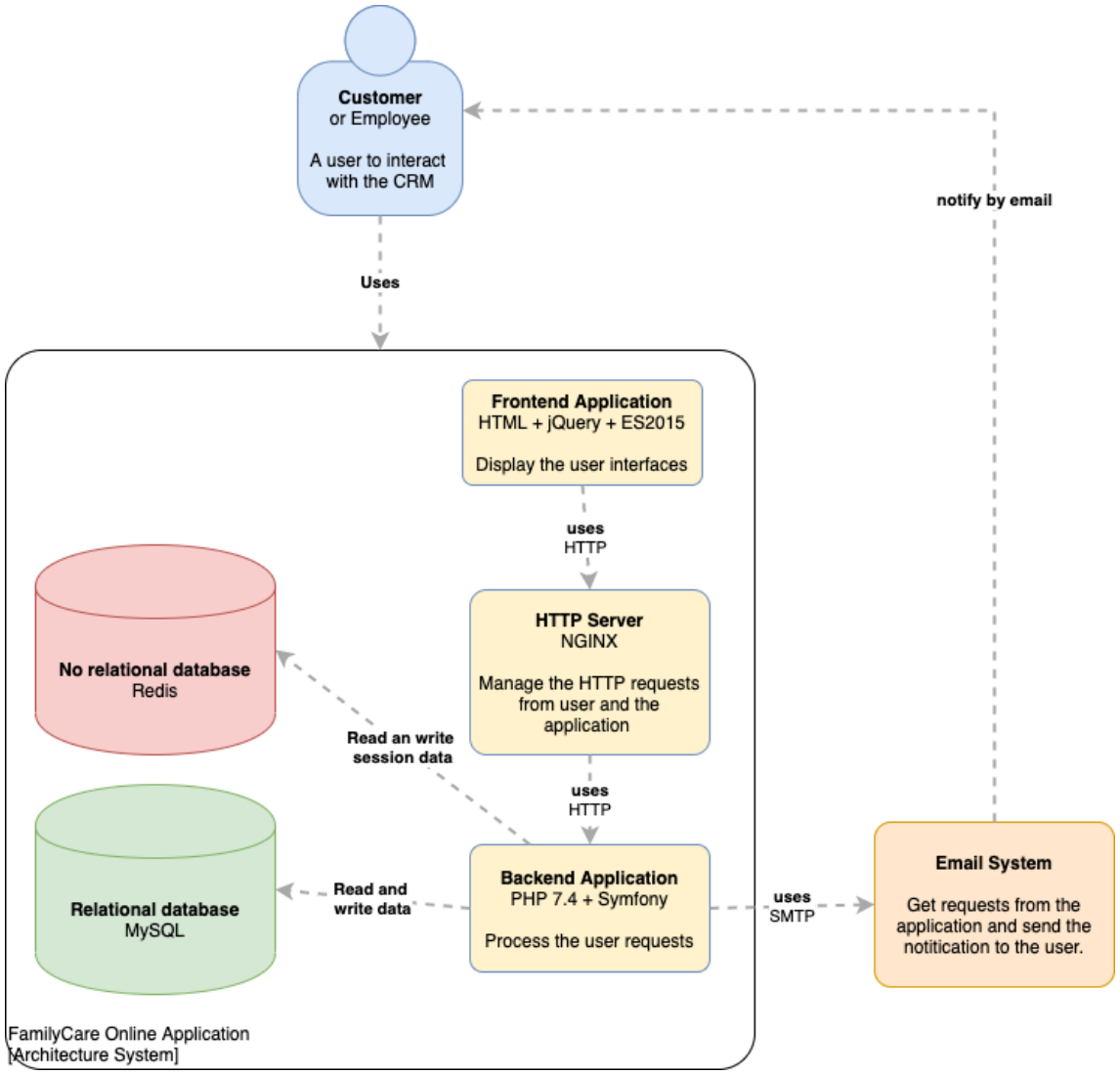


**Fig. 8 - Diagrama de la infraestructura de red de la aplicación.**

## 7.7. Diagrama de la arquitectura del software

Dentro de la aplicación de backend, podemos presentar el siguiente diagrama de la arquitectura del software. Donde se hace uso de una arquitectura hexagonal [11] o de puertos y adaptadores. Este se basa en el uso de tres capas principales [16]:

- Infraestructura.
- Aplicación
- Dominio

Encontramos en la capa más externa, la **capa de infraestructura**, tiene los detalles de implementación que se definen mediante contratos (interfaces) en la capa de dominio, entre los que encontramos:

- Detalles relacionados con el framework de trabajo de Symfony.
- Detalles relacionados con el uso del componente Doctrine.

Seguidamente encontramos una **capa** intermedia, llamada de **aplicación**, esta capa no conoce de la capa más externa (la capa de infraestructura) y tiene un conocimiento más estrecho con la capa de dominio. Para que la capa de infraestructura pueda comunicarse se hacen uso de objetos de transferencia de datos (**DTO**), de este modo la capa de infraestructura se comunica mediante llamadas a los casos de uso y de objetos de transferencia de datos.

Por último está la **capa dominio**, que no tiene conocimiento de las capas más externas y se centra principalmente en modelizar los diferentes elementos que forman al dominio del negocio con el que estamos trabajando, entre los que encontramos a las entidades, los objetos de valores y las definiciones de los contratos que usaran las otras capas y cuyos detalles de implementación son realizados en las capas externas.

### 7.7.1. Principios seguidos en la arquitectura del software

Es importante por lo tanto destacar que para tener un proyecto más fácil de mantener, menos acoplado a la infraestructura del proyecto y por lo tanto más fácil de gestionar a lo largo del tiempo, las capas más internas del proyecto no deben tener conocimiento de las capas más externas y que toda comunicación se produce desde fuera hacia adentro.

Este proceso de comunicación es llamado como el **principio de inversión de dependencias**, donde las capas externas dependen de las capas internas, pero no al revés. Este principio y otros más se pueden encontrar en el acrónimo **SOLID [12]**.

- SRP: Single responsibility principle.
- OCP: Open/closed principle.
- LSP: Liskov substitution principle.

- ISP: Interface segregation principle.
- DIP: Dependency inversion principle.

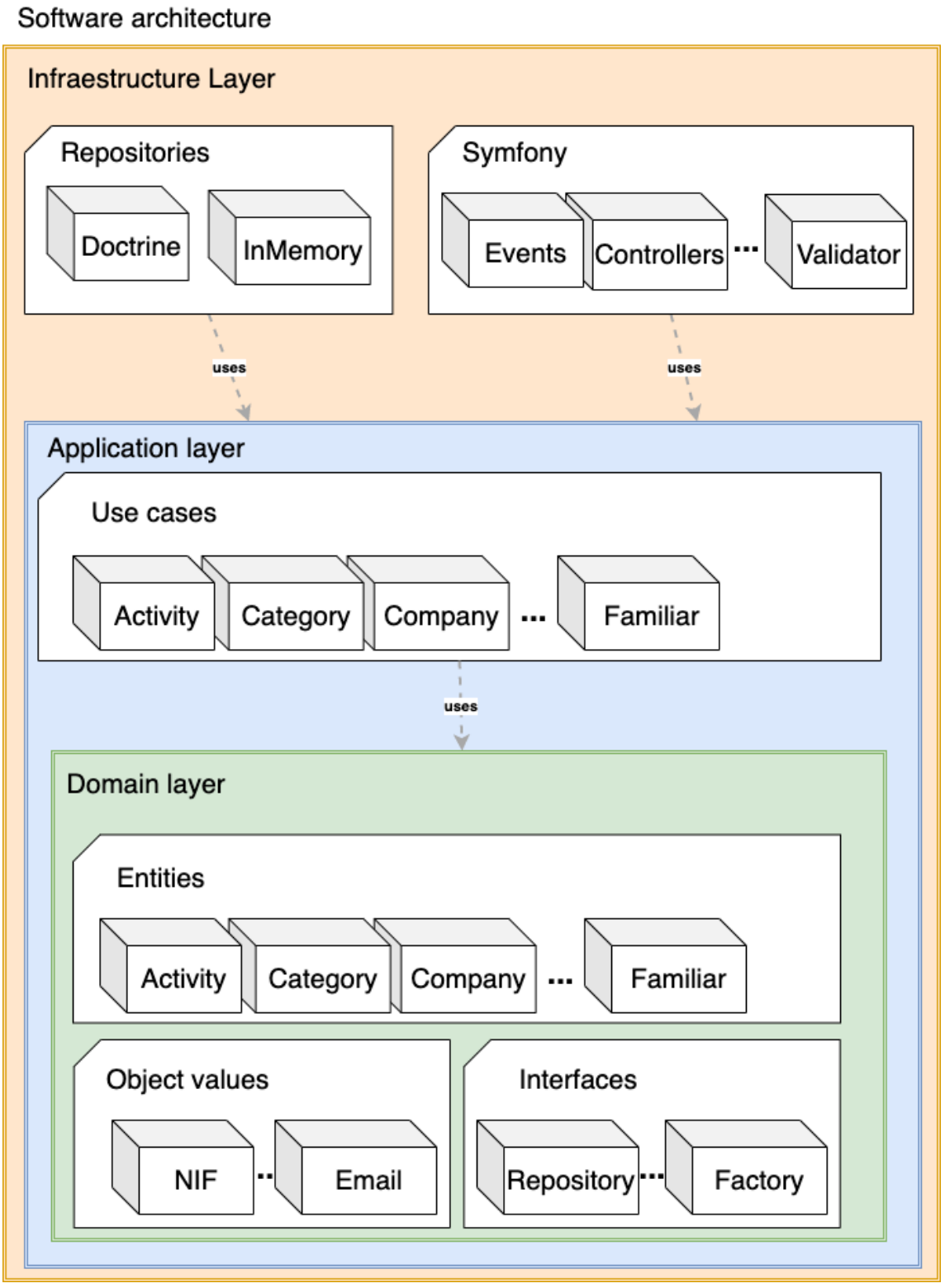


**Fig. 9 - Diagrama de la arquitectura del software del proyecto.**

### 7.7.2. Capa de aplicación

#### Casos de uso

El desarrollo de los casos de uso en la capa de aplicación son el eje fundamental que guía al desarrollo del producto y que permite que exista un producto con un bajo acoplamiento, una alta cohesión, permitir maximizar el mantenimiento y así como la extensión del producto.

El casos de uso se recogen bajo la capa de servicio que es la encarga de dotar del comportamiento esperado por negocio y que no pertenece a un comportamiento de una entidad o un objeto de valor. Se pueden encontrar ejemplos sencillos como el caso de uso de “borrar una actividad”.

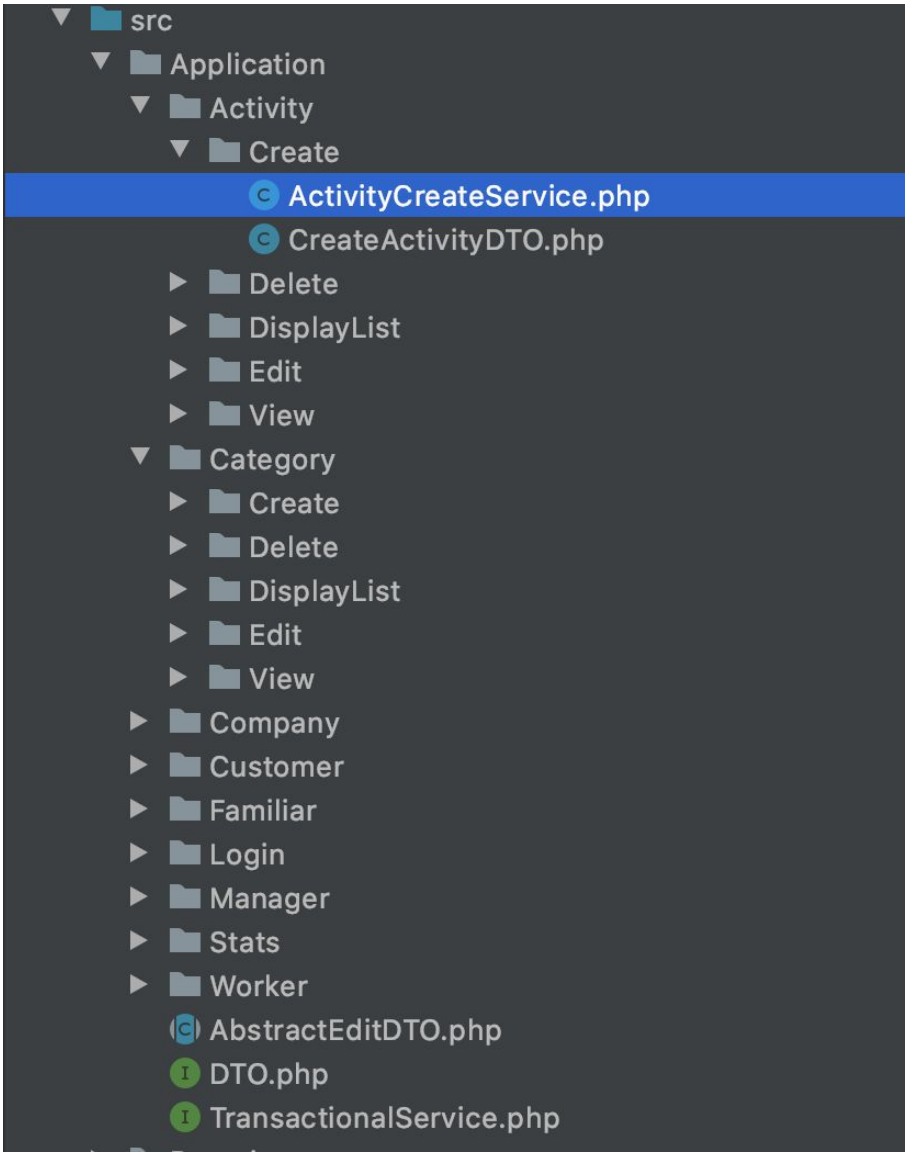


**Fig. 10 - Ejemplo de organización de caso de uso para la creación de una actividad.**

En la estructura de carpetas podemos ver que el código relacionado con la capa de aplicación se encuentra organizado dentro de la carpeta Application. Destacamos las características principales de esta capa:

- Solo tenemos dependencias con la capa más interna, la capa de dominio.
- La comunicación con capas superiores se realizan a partir del uso de objetos de valor (DTO).
- Solo se tiene conocimiento de los contratos creados en la capa de dominio y los detalles de implementación son ajenos en estas capas.

También se puede observar que normalmente un caso de uso tiene relacionado un objeto de transferencia de datos:

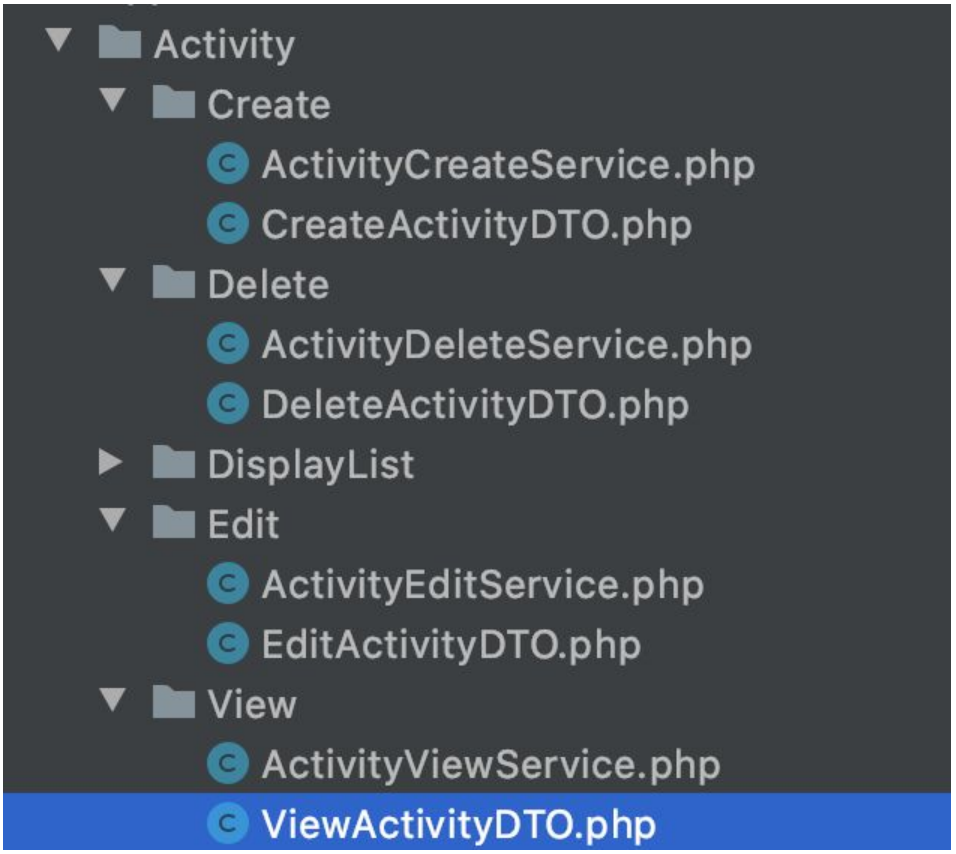


**Fig. 11 - Ejemplos de diferentes casos de uso relacionados con las actividades.**

Se puede destacar sobre los objetos de transferencia de datos que se tratan de clases sencillas donde se almacena información, donde en algunos casos la información será transformada en el formato adecuado para la capa de aplicación.

### 7.7.3. Capa de dominio

#### Entidades

Las entidades son otra parte importante durante el desarrollo del proyecto, son usadas para modelizar un elemento de la lógica de negocio con la que estamos trabajando, es decir representan un elemento importante que tiene un dato identificador y que cada individuo lo hace único con otro individuo de la misma entidad.

Dicho de otra forma, en la lógica de negocio del proyecto, un cliente es representado en el proyecto como una entidad, esta entidad tiene un dato identificador que hace que cada cliente sea único cuando lo comparamos con otros clientes. Por ejemplo, cada cliente tiene un número de identificación fiscal y que es único.

Se puede observar la siguiente estructura de código en la capa de dominio.

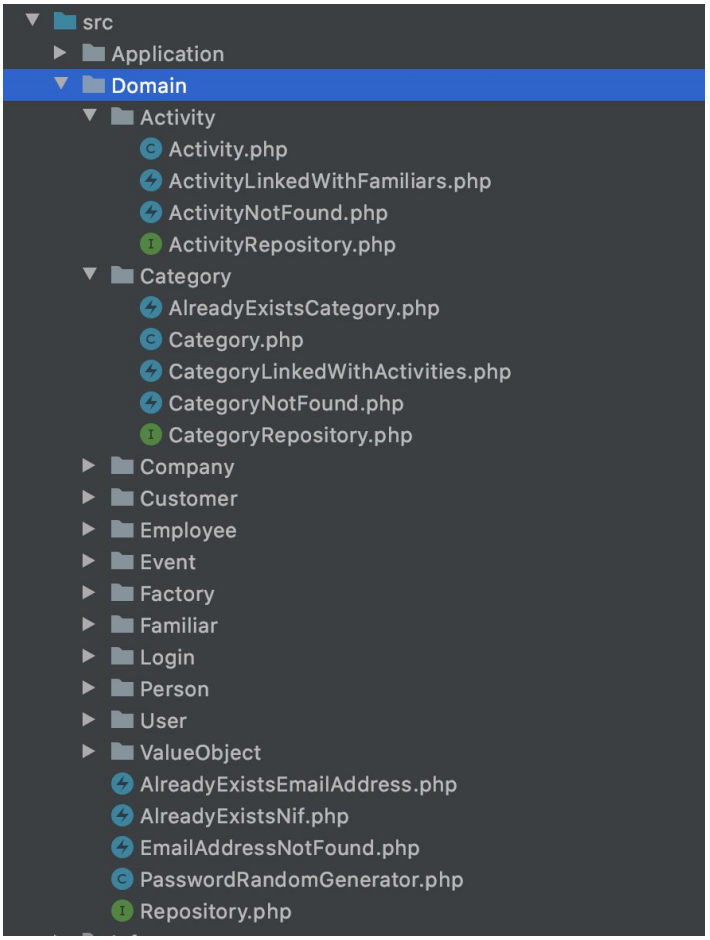


**Fig. 12 - Ejemplo de organización de código en la capa de dominio.**

Podemos destacar que las entidades están representadas por los valores realmente importantes para el proyecto. Para la entidad de **Company** (Empresa) los datos realmente importantes son:

- El espacio de trabajo.

- El nombre.
- La dirección de correo electrónico.

## Objetos de valor [15]

Los objetos de valor son elementos que se recogen en la capa de dominio y que modelizan elementos u objetos que son importantes en el proyecto y que van a interactuar con las entidades. Ejemplo de objeto de valor son:

- La dirección de correo electrónico.
- El número de identificación fiscal.
- La contraseña de un usuario.

Todos estos elementos son diferentes a las entidades ya que no tienen un identificador único para cada objeto, sino que todos los objetos son tratados por igual y tienen un valor o valores importantes en la lógica de negocio.

Estos objetos de valor se diferencian además de las entidades en que no tienen estado, es decir su contenido no podrá ser cambiado y por lo tanto si queremos un NIF con otro valor, tendremos que crear un nuevo objeto usando el nuevo valor.

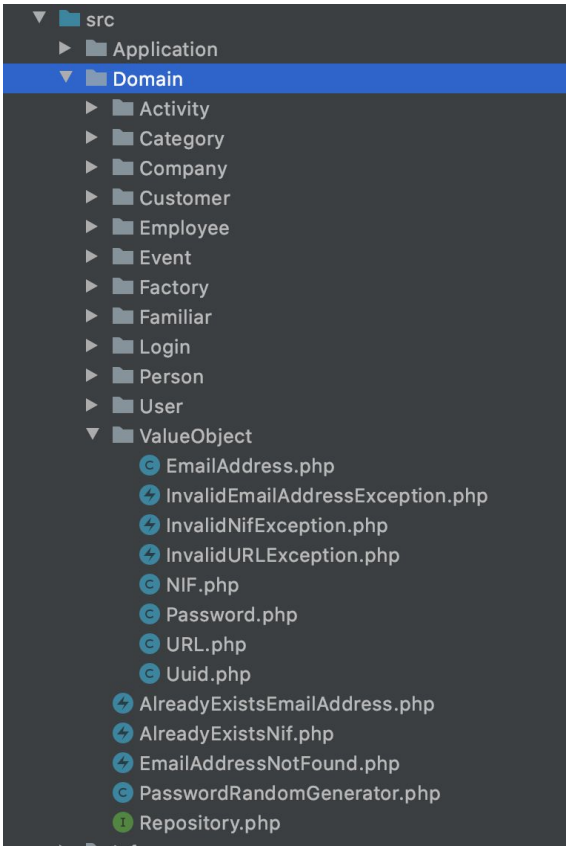


**Fig. 13 - Ejemplos de objetos de valor dentro de la capa de dominio.**

## Contratos o interfaces

Otro aspecto importante que se tiene que destacar en la capa de dominio es que nos define cómo será el comportamiento de ciertas clases, al igual que se hace con las entidades y los objetos de valor, pero con una diferencia, no se define los detalles de cómo se realiza el comportamiento.

Esto sucede por que el comportamiento para realizar las acciones de estas interfaces no pueden ser definidos en la capa de dominio ya que son detalles a muy bajo nivel que dependen de la infraestructura elegida y por lo tanto queremos mantener un acoplamiento bajo y una alta cohesión.

Este tipo de situaciones se suelen dar cuando queremos usar repositorios (o almacenes) de datos, como sería un repositorio de los clientes, de los gestores, trabajadores o los familiares.

Por lo tanto, la capa de dominio nos indica que podremos acceder a los datos del repositorio buscando por un NIF, una dirección de correo electrónica u obtener una lista completa.

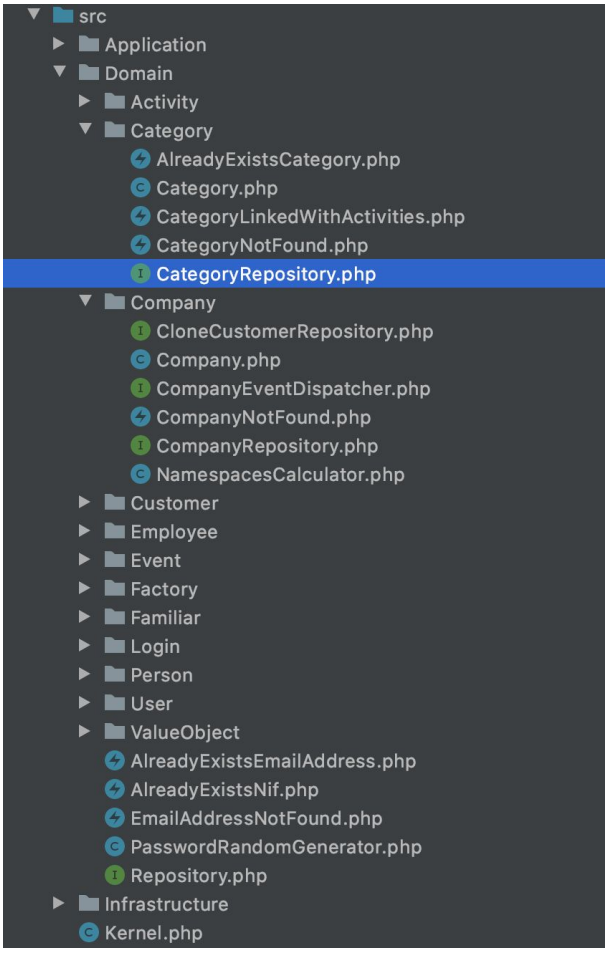


**Fig. 14 - Ejemplos de organización de contratos de repositorios, en color verde.**

### 7.7.4. Capa de infraestructura

En la capa de infraestructura se gestionan las clases haciendo una separación por ámbito de uso y por componentes o software de terceros que se usarán. En el proyecto es muy común observar que tenemo principalmente:

- **Repositorios** de donde extraemos la información necesaria para trabajar en los casos de uso.
- **Controladores** que gestionar las peticiones de datos hacia los casos de uso y se comunican mediante los objetos de transferencia de datos.
- **Suscriptores** que están escuchando diferentes eventos que suceden en el proyecto para ofrecer un comportamiento específico.
- **Otros** elementos como son temas de seguridad, configuraciones especiales para algún componente de plantillas, etc.

#### Repositorios

Son una parte importante del proyecto, sin ellos la aplicación no tendrá información posible a mostrar. El comportamiento que tendrán los repositorios vienen definidos en forma de contratos desde la capa de dominio, donde se indica que tipo de solicitudes se deben implementar y que tipo de parámetros deben usarse.

En el proyecto actual los repositorios se comunican principalmente con una base de datos de tipo **MySQL** y usamos el componente de **Doctrine** que se integra de forma adecuada con el framework de Symfony.

También es importante destacar que las relaciones de las entidades con las tablas de base de datos se realizan mediante el uso de documentos XML, a esta acción de traducir las propiedades de las entidades en las columnas y tablas de MySQL se le llama **mapping**. También cabe destacar que es importante mantener separada los detalles relacionados de la base de datos de las clases de tipo Entidad para mantener un código menos acoplado, el código será más fácil de leer, no tendremos distracciones de aspectos relacionados con la infraestructura de base de datos y mejorar la mantenibilidad.

En la siguiente imagen podemos ver diferentes tipos de repositorios:

- Repositorios que usan el servicio de **MySQL**.
  Se usan con el código de producción.

- Repositorios que almacenan la **información en memoria**.
  Se usan para validar el código en los test unitarios.

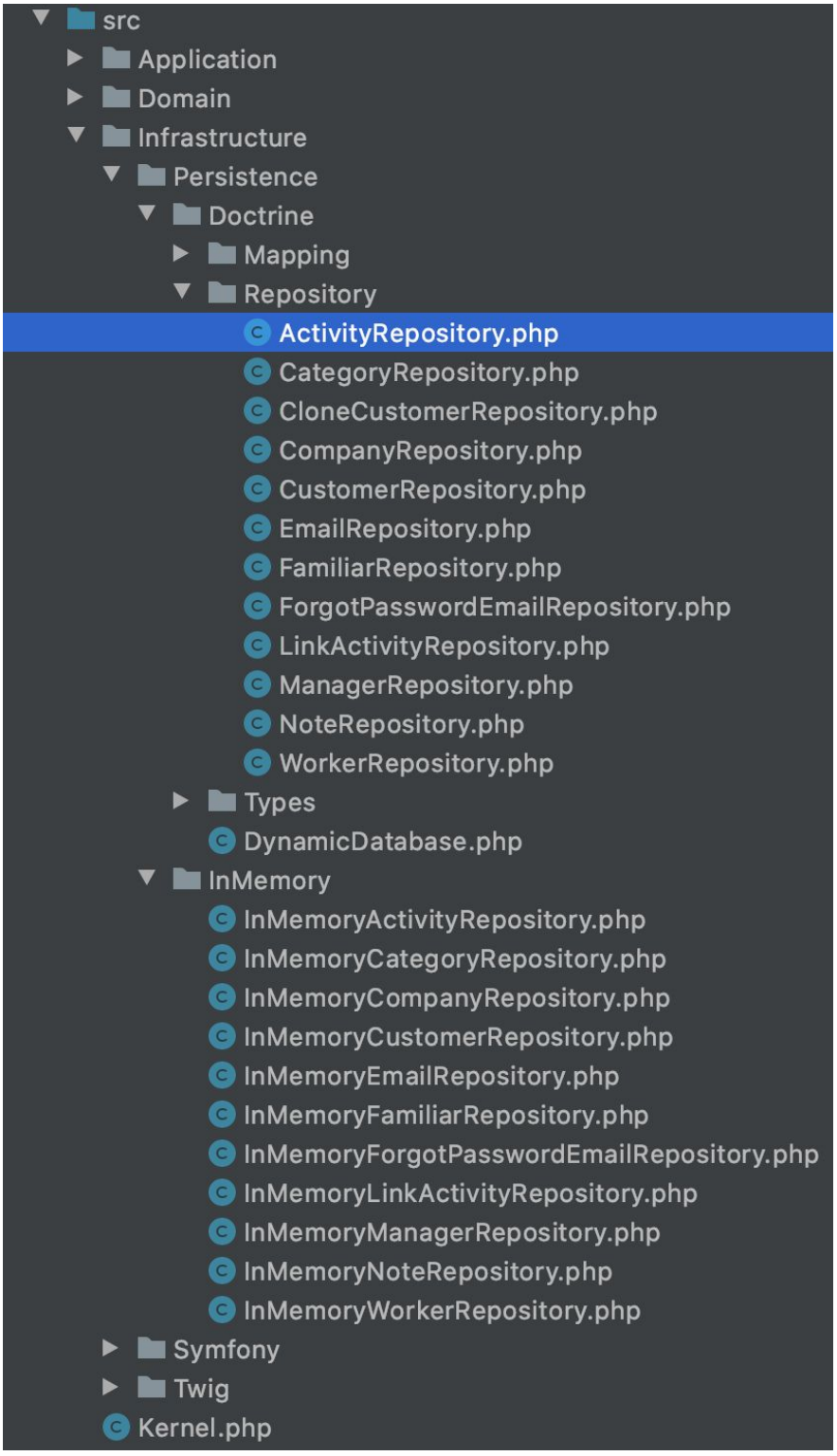


**Fig. 15 - Ejemplos de organización de código en la capa de aplicación.**

## Controladores

Los controladores son otra parte importe del proyecto, se encargan en controlar las solicitudes que los usuarios realizan mediante su navegador para darles una respuesta determinada. Cuando un usuario escribe el nombre de dominio del producto, es este caso sería https://www.samuelfa-uoc.cf, está usando el navegador para hacer una solicitud de datos por internet.

Esta acción baja a un aspecto más técnico está haciendo uso del protocolo **HTTPS** para comunicarse con el servicio de **NGINX** y que pasará al servicio de **PHP-FPM**. Finalmente la solicitud pasa a ser controlada por el framework de **Symfony** que está por encima de las capas que el desarrollador controla. En otras palabras, delegamos al framework la gestión inicial de las llamadas de Red mediante una configuración que le hemos dado previamente y que finalmente acaba llamando a un controlador que hemos creado en la capa de infraestructura.

Este comportamiento donde el framework llama a nuestros componentes se llama el **principio de hollywood [17]**, ya que nosotros nunca llamaremos al framework para que se procese la solicitud llegada por la red.

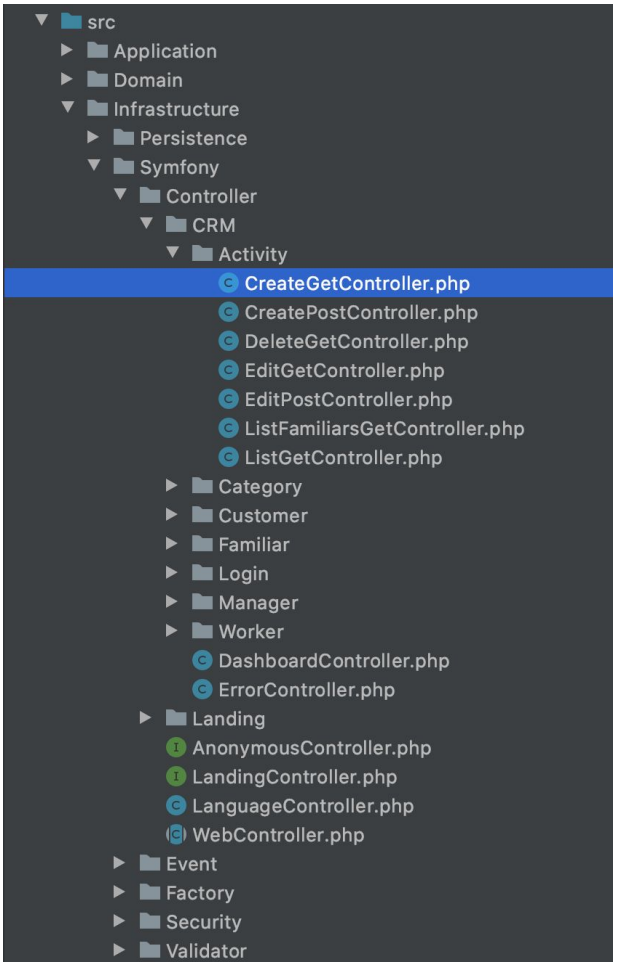


**Fig. 16 - Ejemplos de organización de los controladores.**

## Suscriptores

Otros tipos de clases que podemos destacar en la capa de infraestructura son los suscriptores. Son clases que hacen uso del servicio de eventos, en este caso **Symfony** facilita el trabajo, donde indican a un gestor principal que se encarga de recoger eventos y notificarlos a las clases.

Por lo tanto, cuando un suscriptor se registra en el proyecto indica que se le debe notificar cuando sucede un evento determinado. Esto nos permite reducir el acoplamiento del código en el proyecto, mejorar la cohesión, mejorar el mantenimiento y también nos permite que el proyecto esté abierto a su extensión.

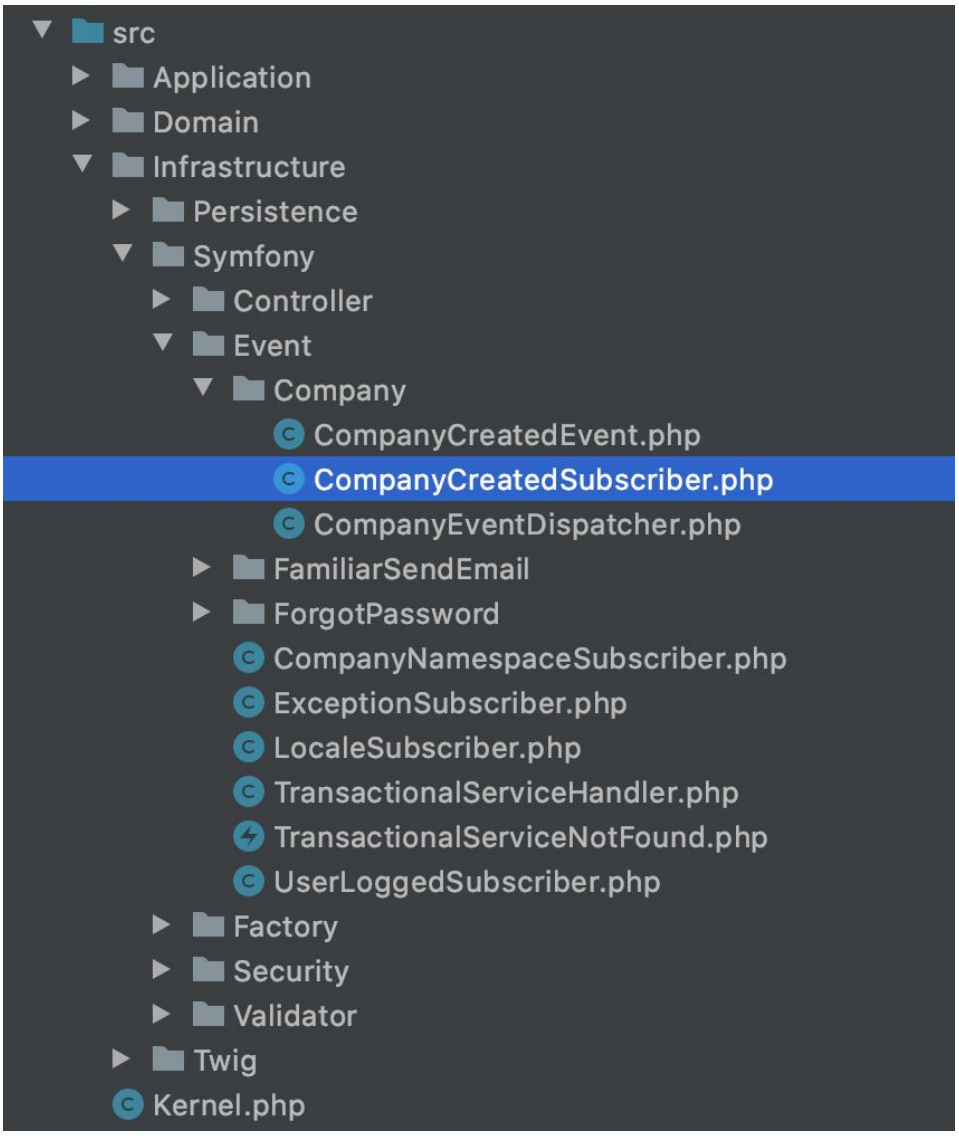


**Fig. 17 - Ejemplos de organización de los suscriptores.**

En la imagen superior, se puede observar como existe una clase cuyo nombre es **CompanyCreatedSubscriber** que espera eventos de "la empresa ha sido creada" para realizar dos tipos de acciones:

- Crear un gestor para la empresa.
- Preparar el espacio de trabajo de la empresa.

Cabe destacar que podemos tener tantos suscriptores como consideremos necesarios para un evento determinado y que podemos gestionar el orden en el que se deben procesar mediante el uso de prioridades.

### 7.7.5. Otros elementos

En un proyecto es común de encontrarnos con otros elementos como son la gestión de plantillas, sistema de traducciones, uso de código de Javascript, de hojas de estilos, sistemas de compilación de código JS y SASS, etc.

Todos estos elementos son externos a las capas de Aplicación, Dominio y de Infraestructura desde el punto de vista del desarrollo del proyecto en la parte de backend. El proyecto no se ha separado entre un proyecto de backend mediante uso de una API REST y otro proyecto de frontend por aspectos de tiempo y se ha mantenido unido todo en un mismo proyecto pero mediante una organización mínima.

#### Plantillas

El mismo proyecto se encarga de proporcionar las diferentes vistas al usuario mediante el uso de un sistemas de plantillas cuyo motor se llama **Twig**, este sistema de plantillas se encuentra mayormente desacoplado del resto de capas de código y en ciertas ocasiones hace uso de los objetos de valor y entidades para poder mostrar la información final de forma completa.

Para ello se hace uso del lenguaje de etiquetas HTML, en su versión 5 que nos permite tener unas vistas más inteligentes (cómo sería poder validar una dirección de email, mostrar campos de fechas con su calendario, etc.), que junto con el código SASS y su conversión en código CSS nos permite tener unas vistas elegantes y dinámicas.

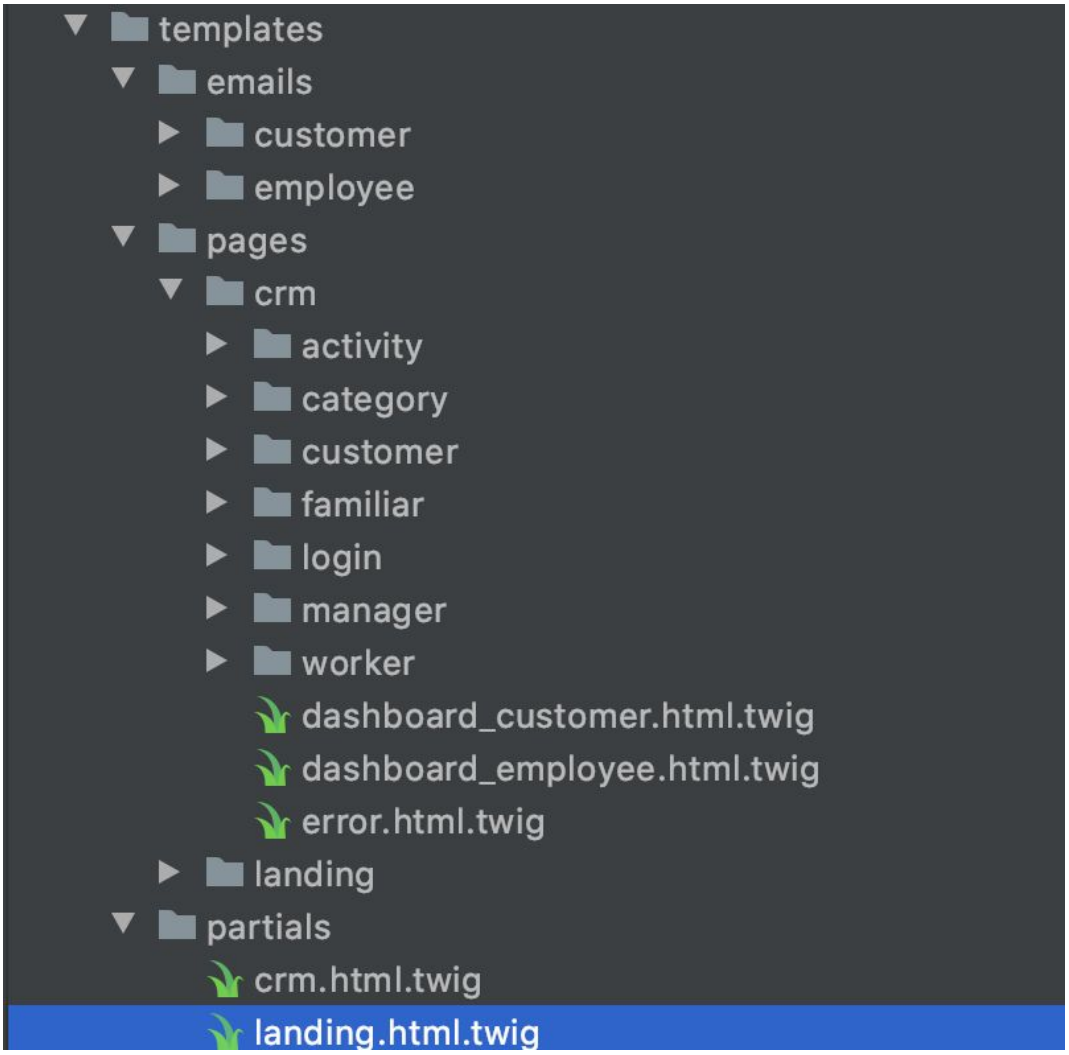


**Fig. 18 - Ejemplos de organización de las plantillas HTML.**

Se puede observar que el sistema de plantillas se usa tanto para las visas de la parte web o frontal como para plantillas de emails. Además el sistema permite crear contenido parcial como sería la estructura general como subelementos de las vistas.

## Traducciones

Junto con el motor de **Twig** se utiliza un sistema de **traducciones** que permite ofrecer tantos idiomas como se desee a los usuarios. En el proyecto se a optado por usar el español e inglés como idiomas disponibles.

El sistema de traducciones está integrado con el framework de Symfony, siendo un componente adicional que los desarrolladores pueden optar por usarlo o no.

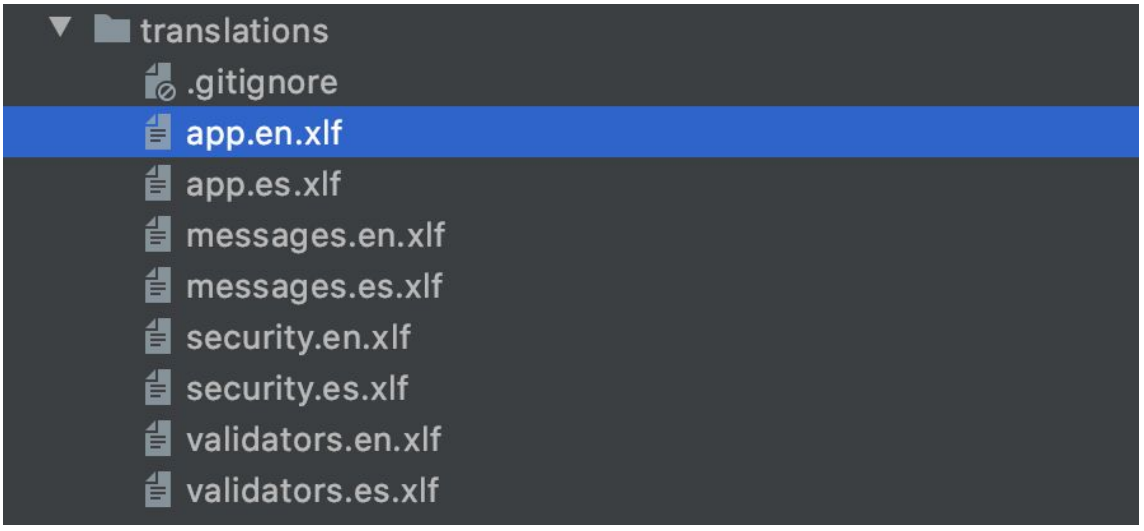


**Fig. 19 - Ejemplos de organización de las traducciones en inglés y español.**

Los tipos de ficheros para las traducciones que usa el sistema de traducciones es XML Localization Interchange File Format o también de forma abreviada XLIFF cuya extensión es XLF y se basa en el sistema de almacenamiento de datos XML.

## Sistemas de compilación

Un aspecto interesante en el proyecto es el usado con las herramientas de **Yarn** y **Webpack.** Son dos herramientas que sirven para integrar componentes de terceros que queremos utilizar en las vistas tanto desde el lado del lenguaje de JavaScript, como desde el lado del lenguaje de SASS para utilizar nuevas hojas de estilos.

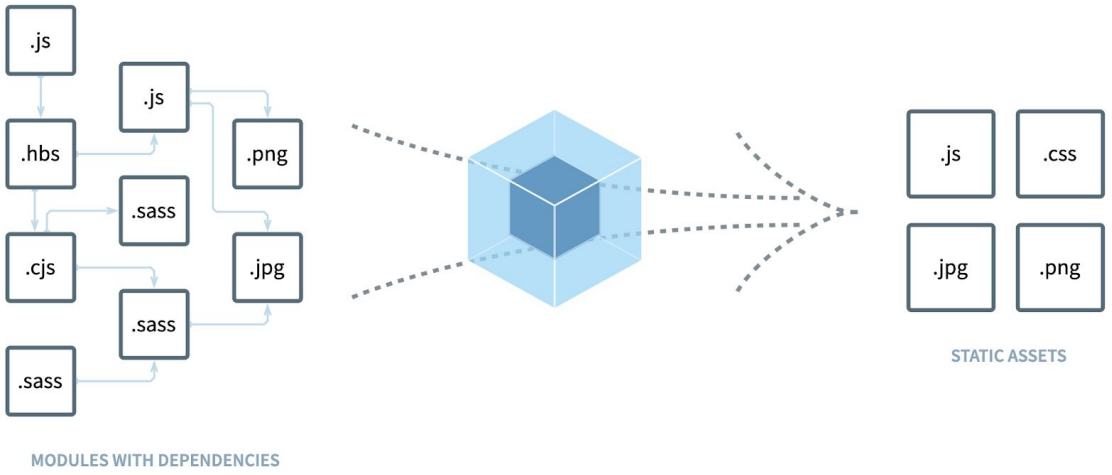


**Fig. 20 - Ilustración que simboliza el funcionamiento de la herramienta webpack. Fuente: https://webpack.js.org**

Ejemplos de uso en el proyecto son al crear el calendario desde el punto de vista de la parte frontal (o frontend) se ha usado el componente de terceros llamado **fullcalendar** que proporciona todo el funcionamiento de vistas diarias, semanales y mensuales.

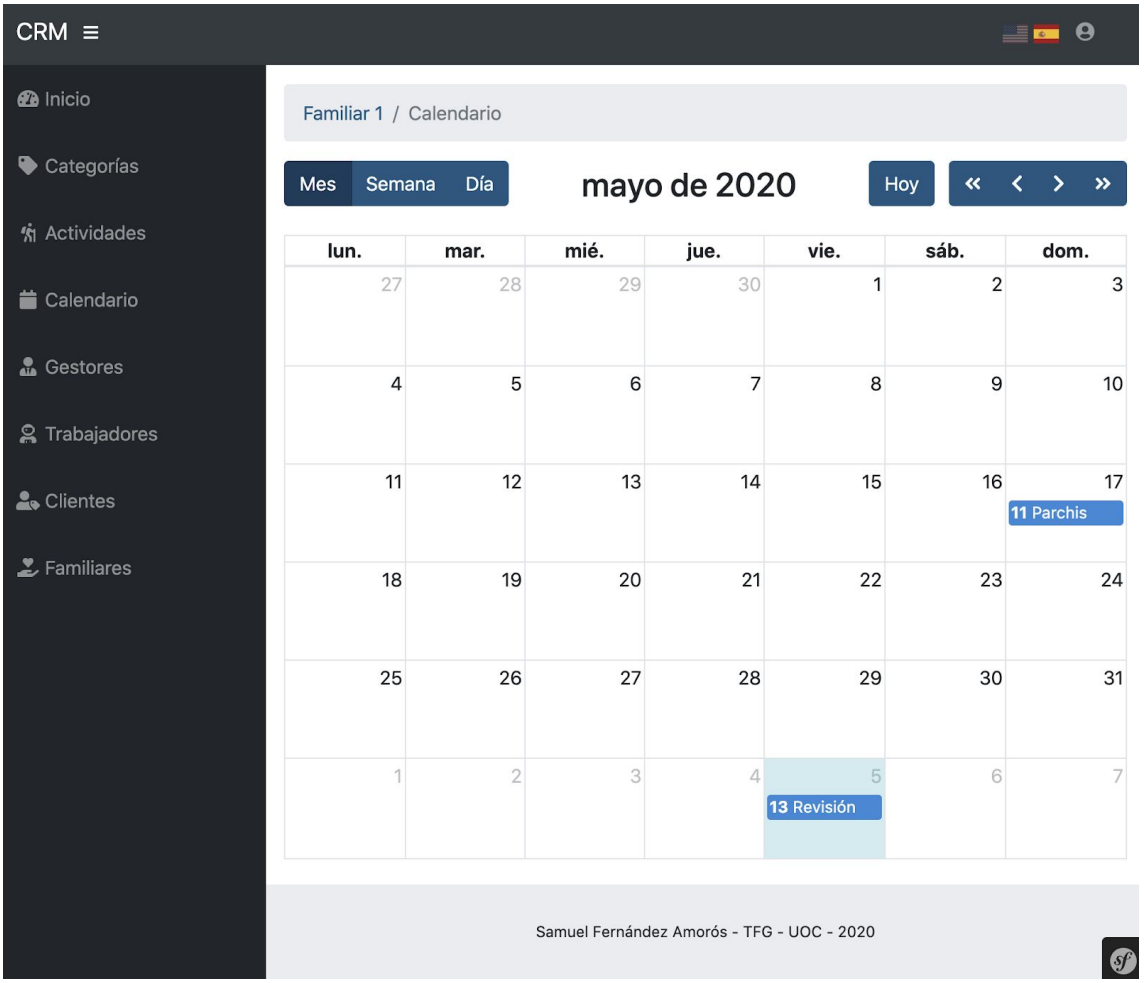


**Fig. 21 - Captura de pantalla del calendario de actividades de un familiar.**

Con un código mínimo de JavaScript podemos crear vistas de calendarios muy completas.

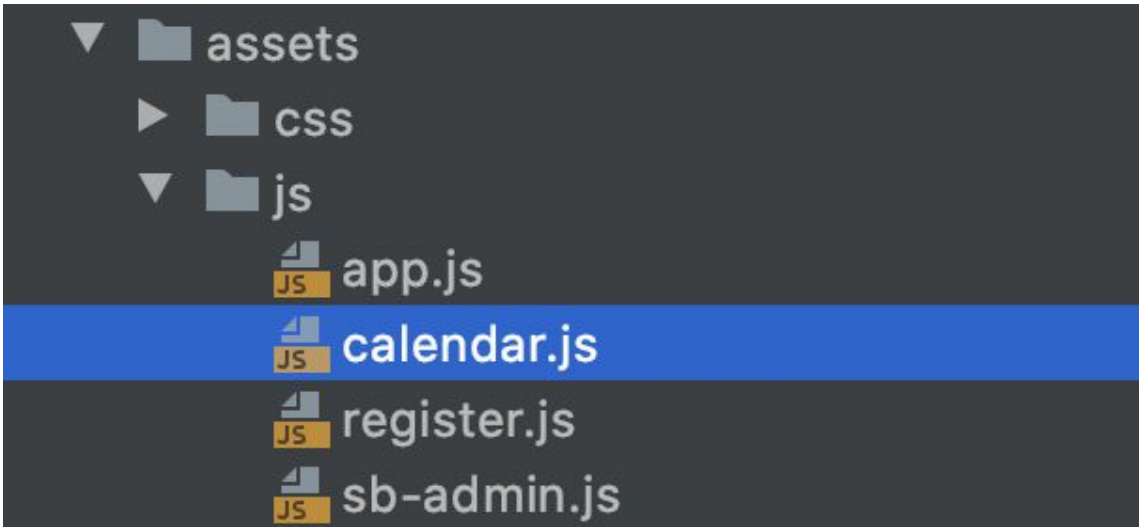


**Fig. 22 - Ejemplo de organización de código Javascript.**

Así como nos permite usar diferentes plantillas de estilos mediante el uso de SASS.

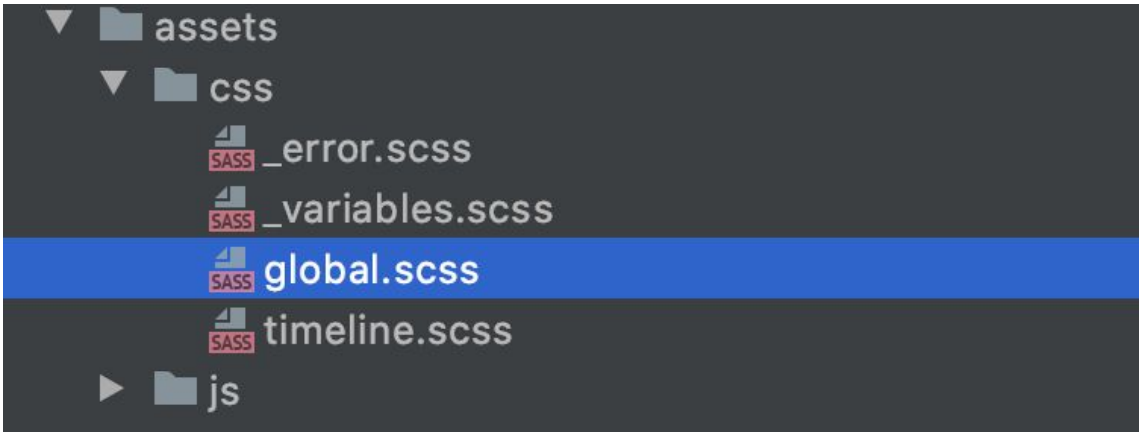


**Fig. 23 - Ejemplo de organización de código SCSS.**

# 8. Pruebas del proyecto

El proyecto debe de recoger una serie de batería de pruebas de diferente tipo que validen que los requisitos del negocio indicados en el presente documento se cumplen, para ello se tienen que crear los siguientes tipos de test:

- **Test unitarios.**

  Se debe crear un test unitario por cada caso de uso donde se comprueban los estados normales, los límites de cada caso de uso y situaciones no válidas.

- **Test de integración.**

  Se debe crear diferentes test de integración que validen que el producto se encuentra correctamente integrado con la infraestructura utilizada.

- **Test de funcionales.**

  Se debe crear varios test de funcionales que validen que el producto se encuentra correctamente funcionando a partir de los casos de uso indicados.

Por lo tanto se seguirá el concepto de la pirámide de tests [13] donde se indica de forma visual cuál debe ser la cantidad predominante de cada tipo de tests para el proyecto.

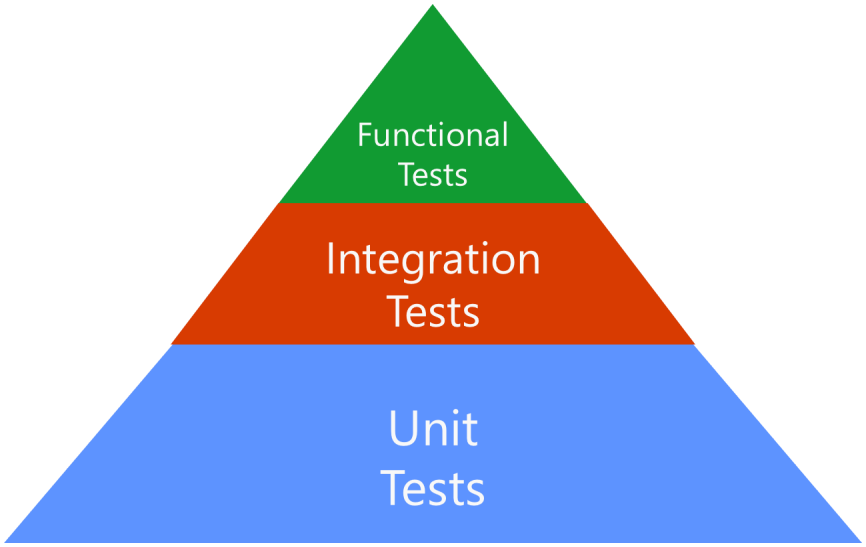


**Fig. 24 - Piramide de tests. Fuente: Microsoft.**

Se puede ver que nuestro sistema principalmente debe valerse de los test unitarios para validar las reglas del dominio, en menor cantidad tenemos que tener tests de integración y finalmente la menor cantidad de tests serán los tests funcionales.

Es importante también destacar que los tests unitarios serán los más rápidos de ser ejecutados y los que menor coste suponen a las empresas para poder mantener. Conforme se sube de posición en la pirámide se reduce la velocidad y se aumenta el coste que supondrá para una empresa.

### Test unitarios

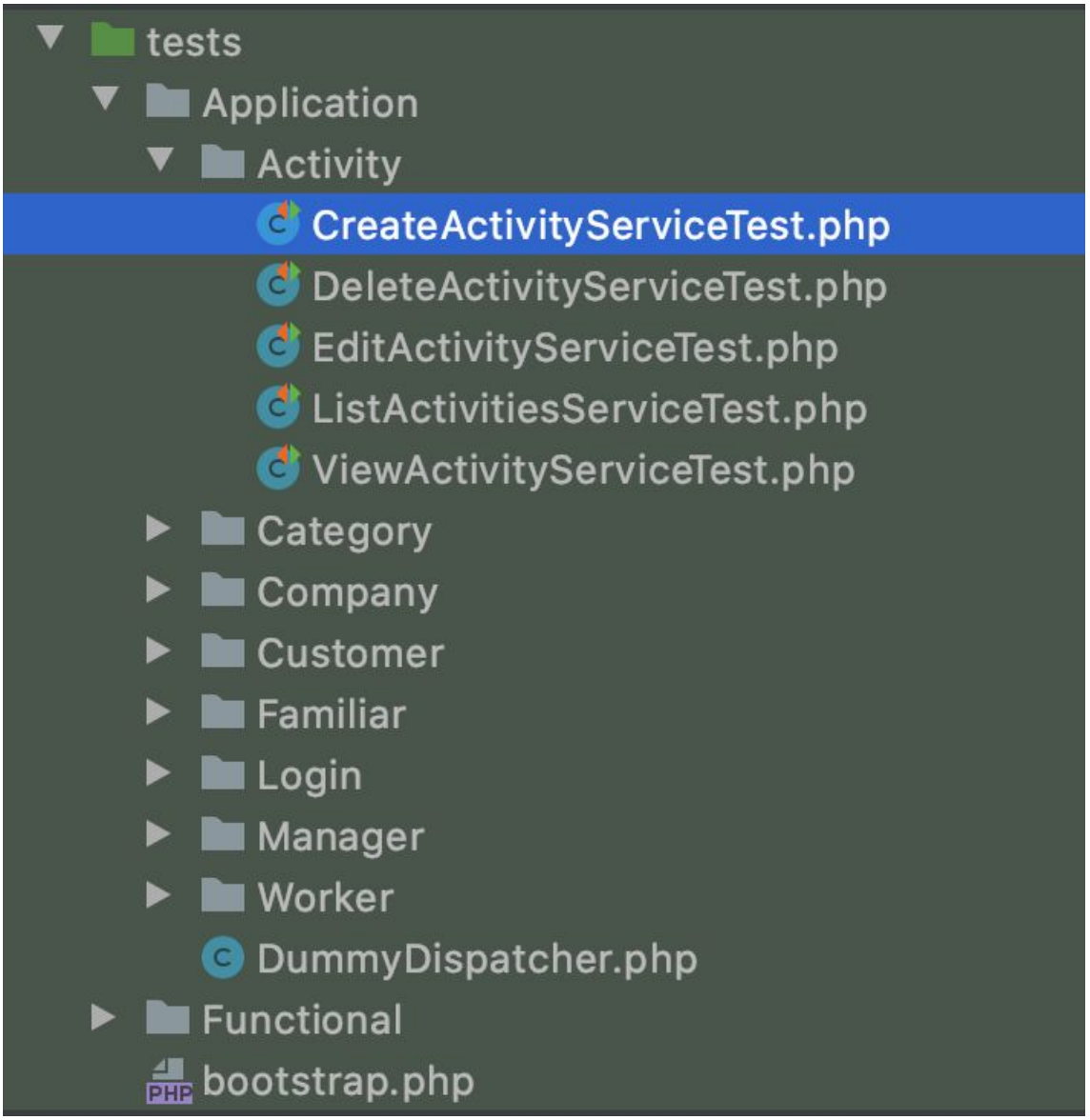


**Fig. 25 - Ejemplos de código de test unitarios.**

### Test de integración y test funcionales

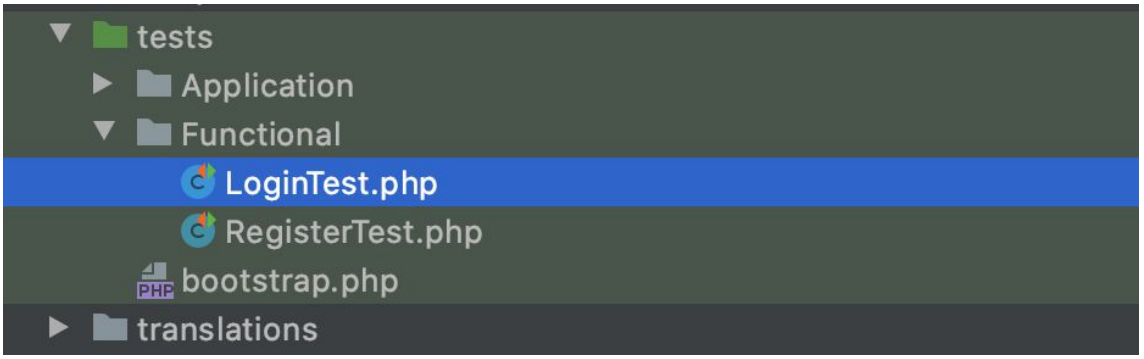


**Fig. 26 - Ejemplos de código de test funcionales y de integración.**

# 9. Despliegue del proyecto

El proyecto hace uso de dos herramientas para realizar la puesta a punto del entorno de producción, así como el aprovisionamiento de la instalación y configuración del software necesario para que el proyecto funcione en el entorno de producción.

## 9.1. Terraform

La herramienta de **Terraform** que sirve para construir (levantar) y configurar los servidores necesarios en el proveedor que se seleccione. En este proyecto se ha optado por usar como proveedor de la infraestructura de red al servicio de Google Cloud Platform de la empresa Google.

Cualquiera de las empresas que ofrecen servicios tipo PaaS (Platform as a Service) será adecuada, como es Amazon AWS, Cyberneticos, Dinahosting o Digital Ocean. Se opta por Google ya que tiene un plan gratuito por un año o crédito de 100 $.

Se puede configurar servidores sin usar **Terraform**, pero la herramienta da varios tipos de beneficios como son:

- Controlar los cambios junto con git de los cambios realizados en los servidores.
- Automatización de tareas al realizar modificaciones de los servidores.

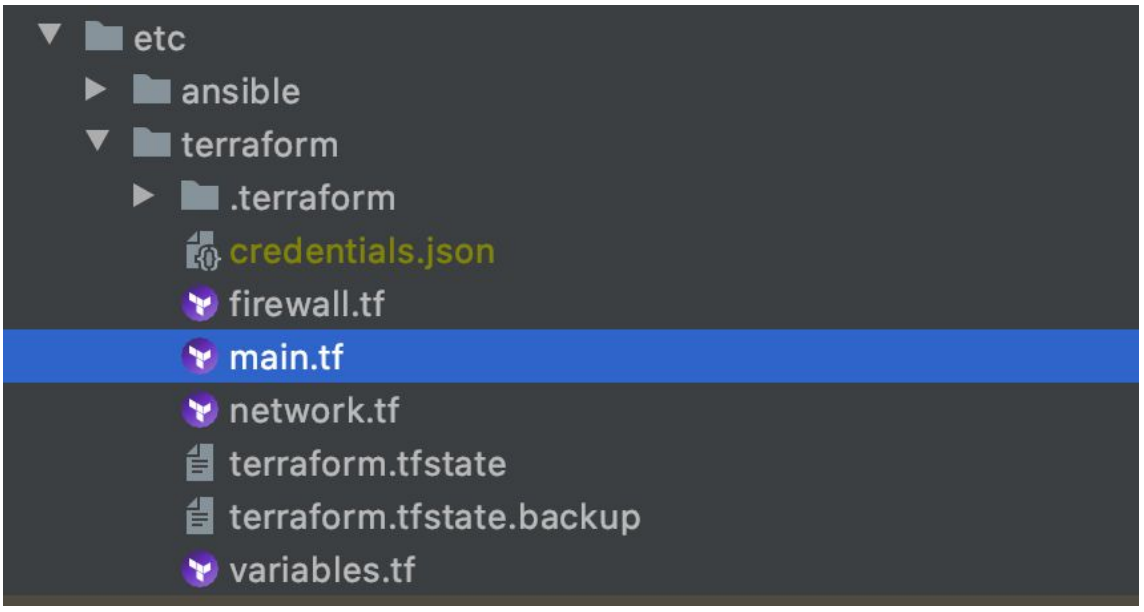


**Fig 27 - Ejemplos de organización de código para Terraform.**

## 9.2. Ansible

Ansible es otra herramienta que nos permite escribir el proceso de instalación y configuración del software necesario en cada servidor. Mientras que Terraform lo usamos para crear los servidores y configurar su red, seguridad y otros aspectos generales, ansible sirve para dos aspectos:

- Instalar y configurar el software necesario para cada servidor.
- Preparación y montaje del proyecto para su funcionamiento.

  Por un lado, ansible sirve como herramienta de aprovisionamiento de los servidores, por lo que podemos dejar escrito en el proyecto cuáles son los pasos específicos para instalar el software y configurarlo como sería NGINX, PHP-FPM, Composer, etc.

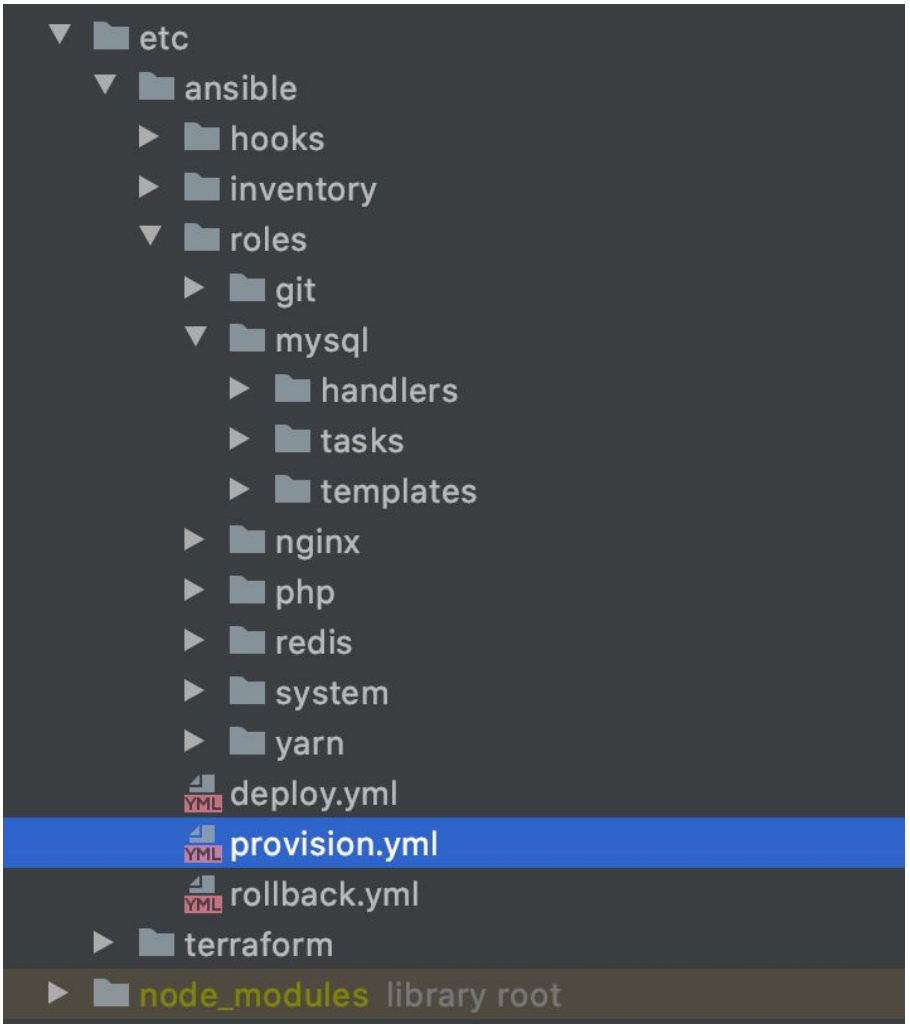


**Fig. 28 - Ejemplos de organización de código para Ansible.**

Por otro lado se usa ansible junto con ansistrano para escribir los pasos necesarios para clonar el proyecto y configurarlo de forma correcta en un entorno de desarrollo, preproducción o producción.

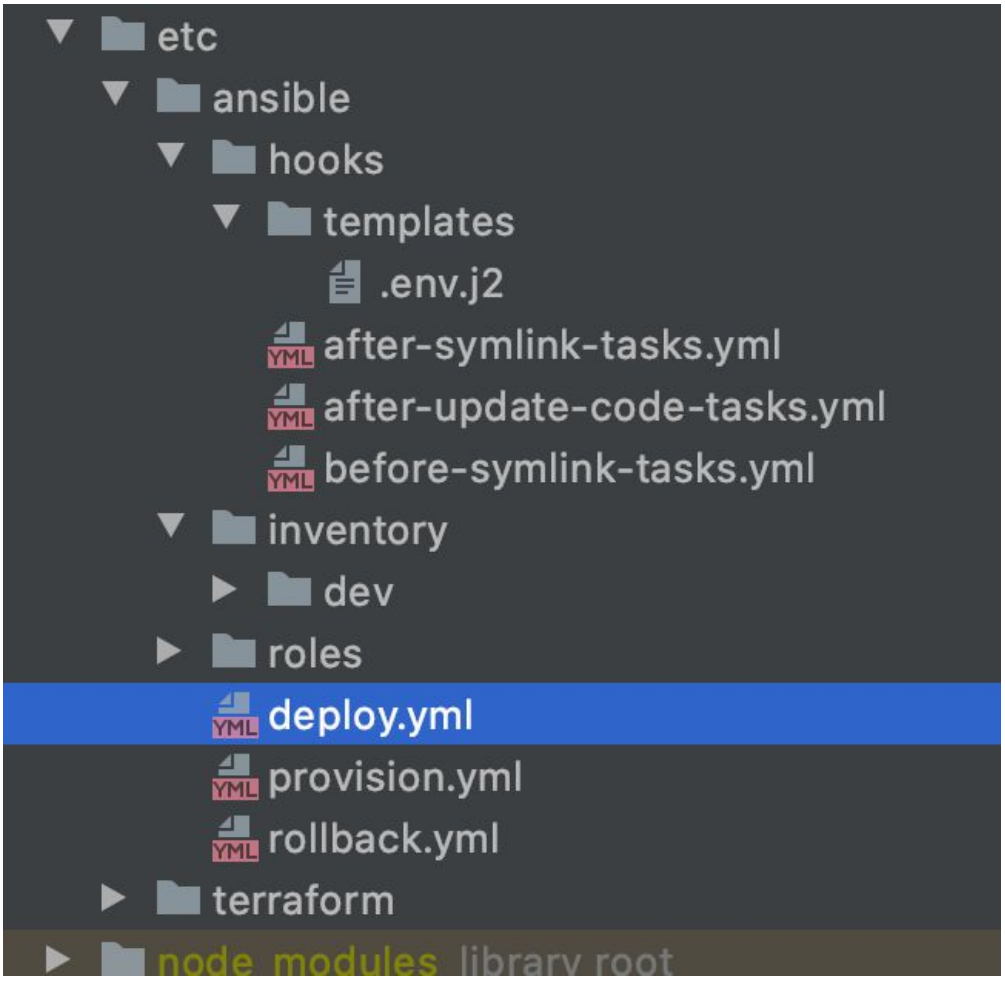


**Fig 29 - Ejemplos de organización de código para Ansistrano.**

Por último, cabe destacar que ansistrano también nos permite poder volver a una versión anterior del proyecto de forma sencilla, es decir hacer rollback, para ello la herramienta almacena las últimas versiones que fueron publicadas anteriormente con lo que si alguna versión más reciente tiene fallos con tan solo unos pequeños pasos se puede recuperar una versión anterior.

# 10. Resultados

En la presente sección se pasa a mostrar los resultados finales que se ha obtenido al realizar el desarrollo del proyecto a lo largo del semestre.

## 10.1. Landing

Por lo tanto tenemos una *landing* o página de aterrizaje accesible para todos las personas que accedan escribiendo la URL del sitio web o mediante alguna búsqueda en algún buscador web cómo sería Google o DuckDuckGo.

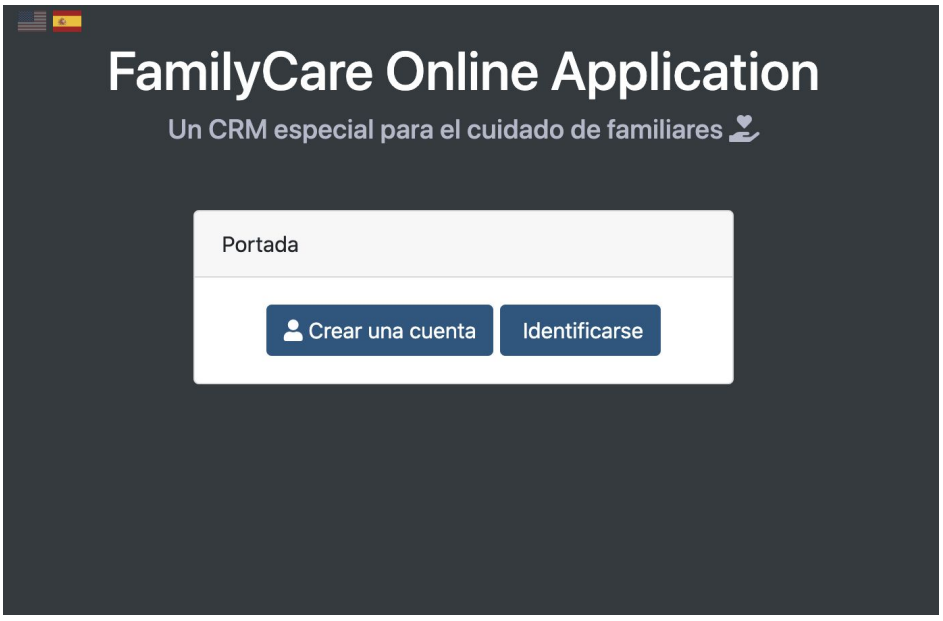


**Fig. 30 - Captura de pantalla de la página principal de la aplicación.**

Desde esta sección el usuario invitado puede optar por crear una cuenta nueva o identificarse en el sistema para acceder a su entorno de trabajo.

## 10.2. Registro de cuenta

Si el usuario decide crearse una cuenta tendrá que proporcionar una información mínima, entre las que destacamos:

- El nombre de la empresa.
- El nombre que desea para su espacio de trabajo.
- La dirección de email del gestor.
- El NIF del gestor


**Fig. 31 - Captura de pantalla de la página de registro de empresa.**

## 10.3. Sistema de identificación

En cambio si opta por identificarse en la plataforma, primero debe proporcionar el nombre de su espacio de trabajo. Este aspecto es importante ya que la información de las empresas se encuentra físicamente separada.


**Fig. 32 - Captura de pantalla para acceder al espacio de trabajo de una empresa.**

Una vez que el usuario ha indicado su nombre del espacio de trabajo ya puede proporcionar sus credenciales para identificarse. Es importante destacar que hay dos tipos de acceso:

- Identificación para trabajadores y gestores.
- Identificación para clientes.

Que se puede apreciar en la siguiente captura, si el usuario que desea acceder como un cliente, tendrá que pulsar en el texto “¿Eres cliente?” o bien escribir una URL como https://demo.samuelfa-uoc.cf/login/customer .


**Fig. 33 - Captura de pantalla para identificarse como trabajador en un espacio de trabajo.**

Por último el usuario tiene la opción de iniciar un proceso para generar una nueva contraseña en caso de que no sea capaz de recordar la anterior.


**Fig. 34 - Captura de pantalla para iniciar el proceso de generación de nueva contraseña.**

Una vez indicado la dirección de email, el usuario recibirá un email que le permitirá comenzar un proceso para generar una contraseña nueva.

## 10.4. Dashboard

Cuando el usuario se identifica en el sistema, la primera sección que verá es el dashboard o el tablero con métricas de uso de la aplicación. Para el desarrollo del TFG se han puesto cuatro sencillos datos:

- Total de categorías.
- Total de actividades.
- Total de clientes.
- Total de familiares.

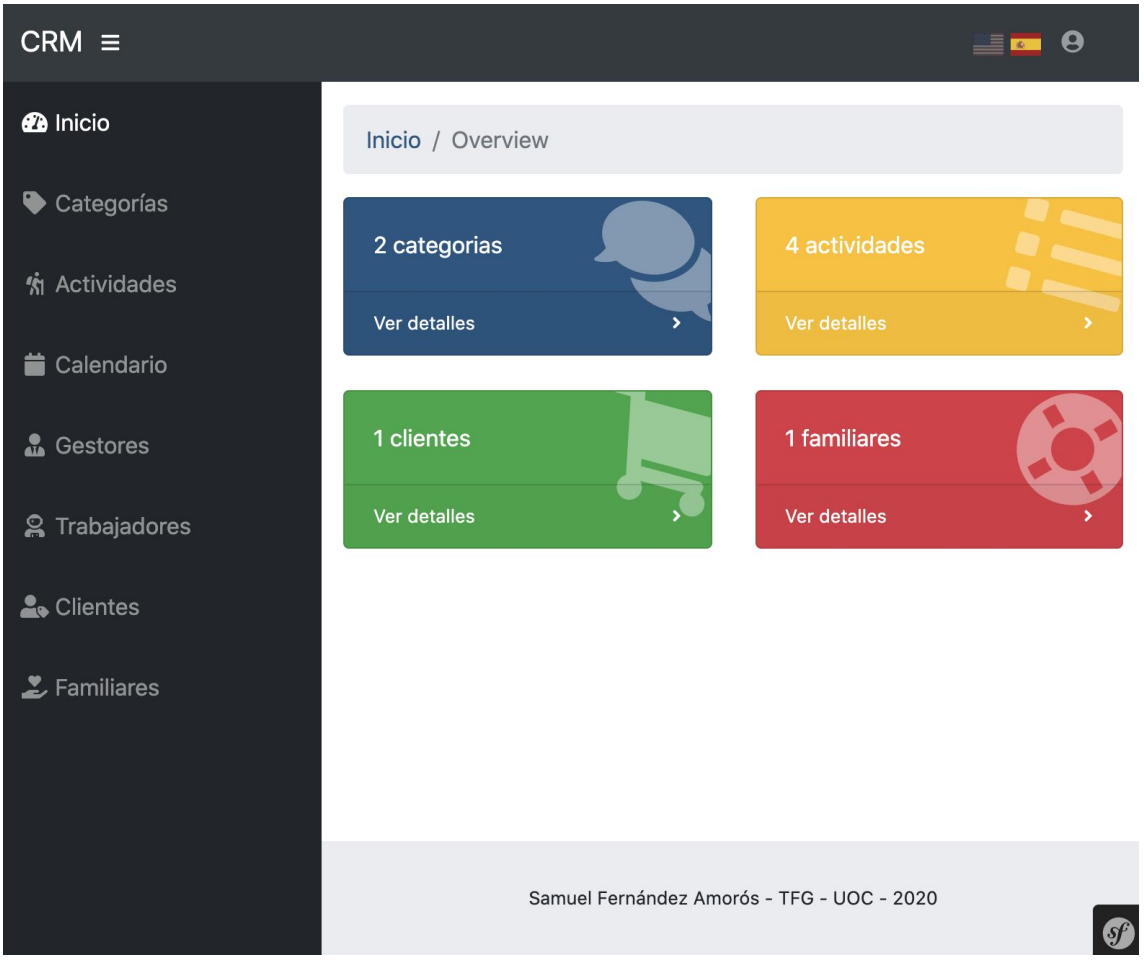


**Fig. 35 - Captura de pantalla de la página principal del usuario con estadísticas de uso de la aplicación.**

Esta sección se adapta al tipo de usuario (gestor, trabajador o cliente) para mostrar más o menos información. Por ejemplo, un cliente nunca podrá ver el total de clientes dados de alta en el sistema.

Por otro lado, podemos observar varios elementos comunes en la aplicación, siendo:

- El menú lateral izquierda (que se puede reducir de tamaño).
- La cabecera, que permite el cambio de idioma y un menu para cerrar la sesión actual.
- El nombre de la aplicación.

## 10.5. Categorías

Esta sección es solo accesible para el gestor, permite la creación de elementos que agruparan actividades. En esta sección se puede listar categorías y para cada una de ellas editar, crear, borrar y listar sus actividades.

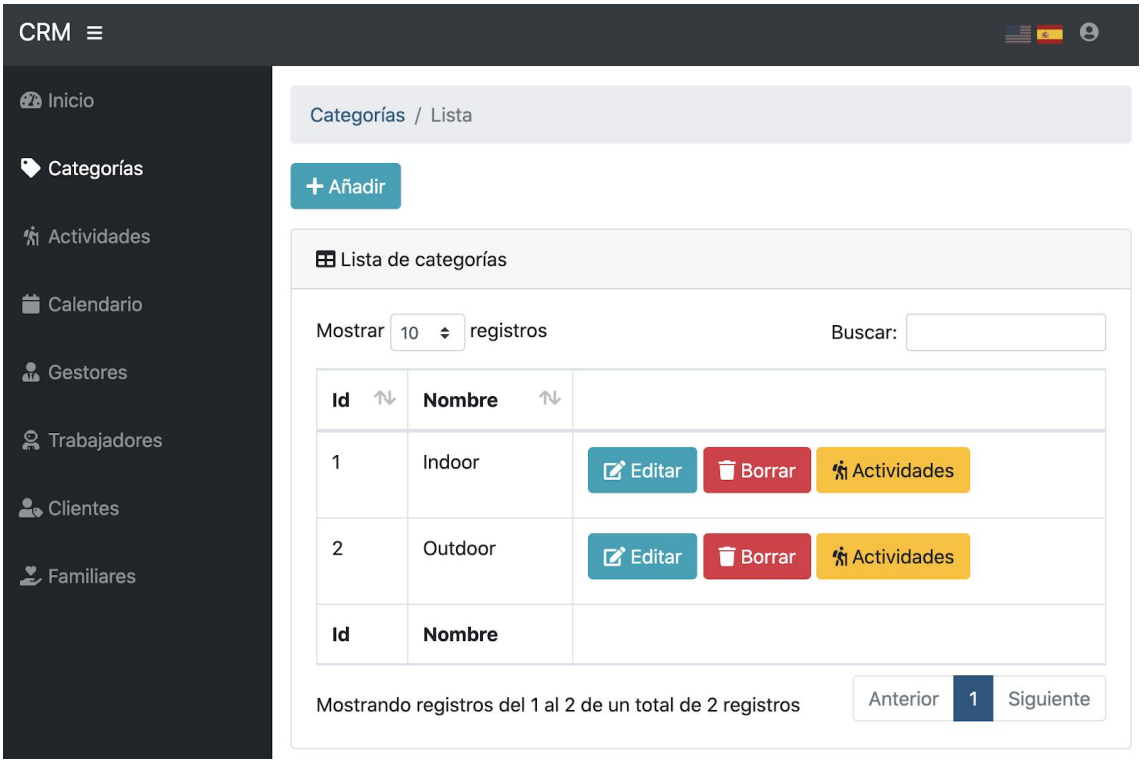


**Fig. 36 - Captura de pantalla con el listado de las categorías.**

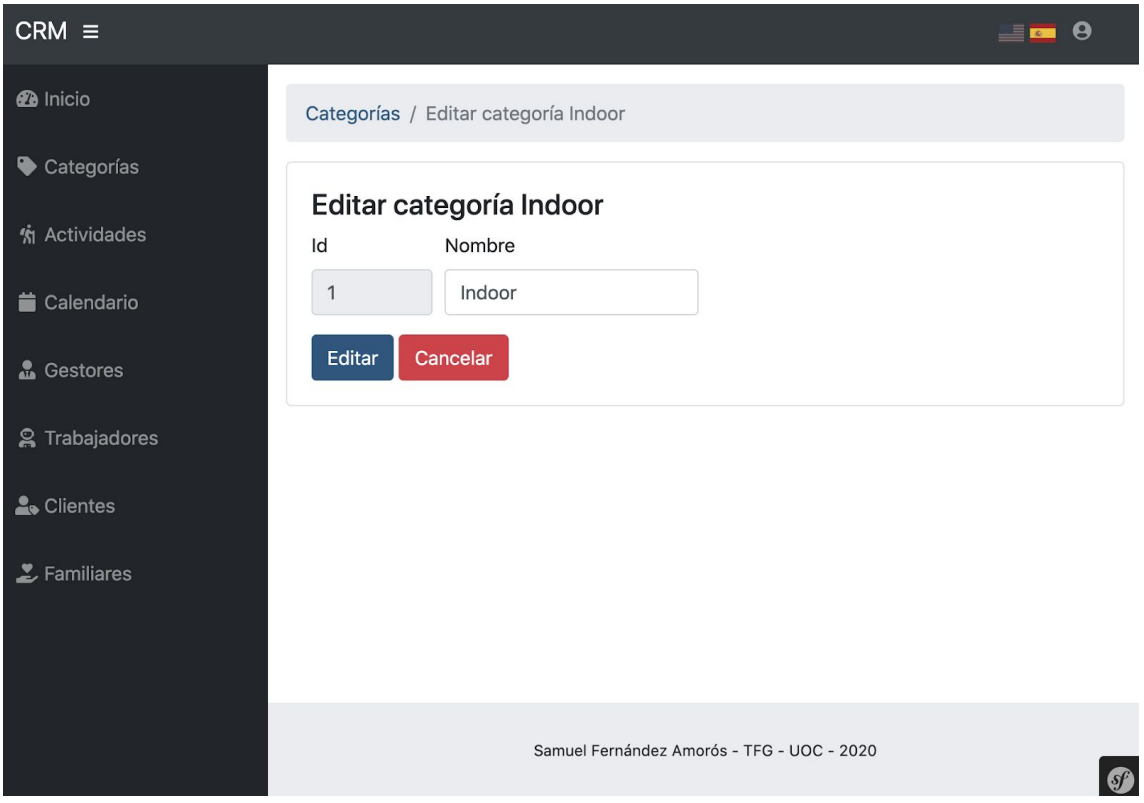


**Fig. 37 - Captura de pantalla con la página para editar una categoría.**

## 10.6. Actividades

Esta sección es accesible por el gestor y el trabajador, permite la creación de actividades que tienen una fecha de inicio y fin. En esta sección se puede listar actividades y para cada una de ellas editar, crear, borrar y listar los familiares apuntados.

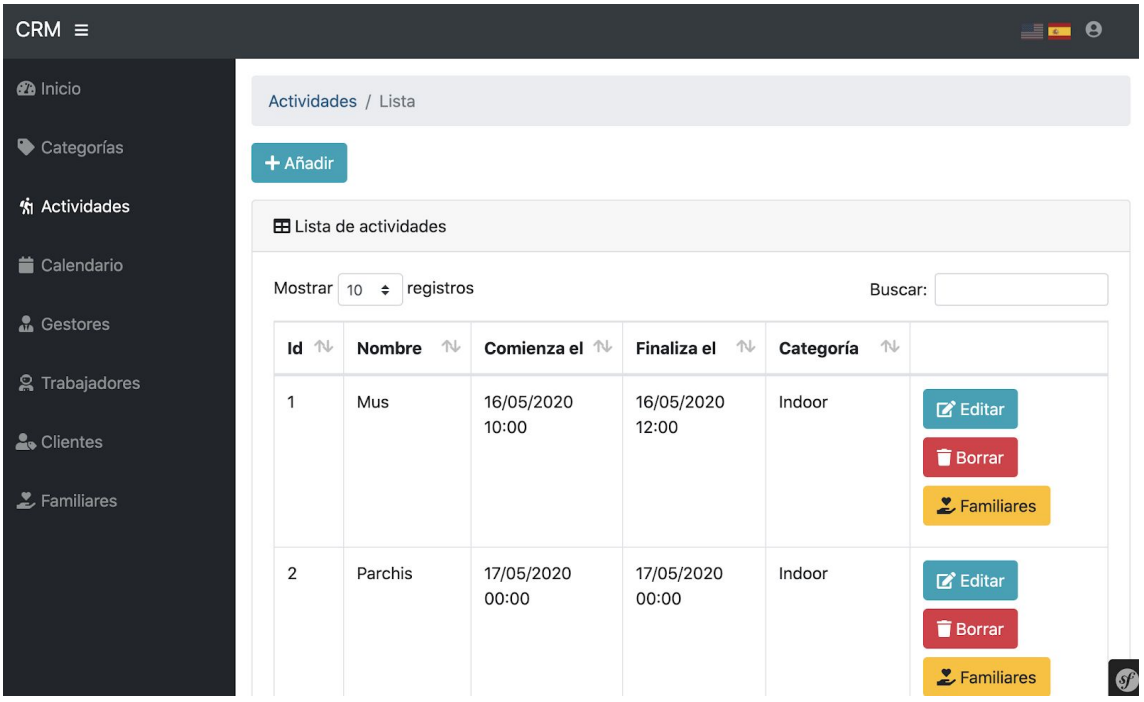


**Fig. 38 - Captura de pantalla con el listado de las actividades.**

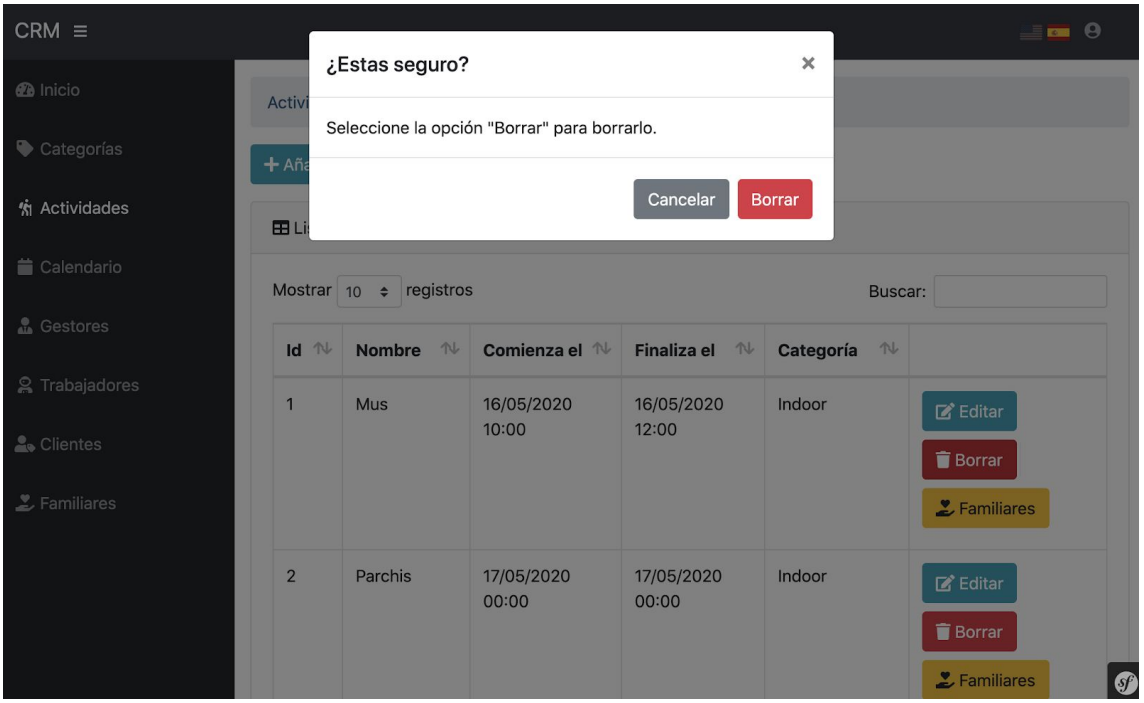


**Fig. 39 - Captura de pantalla con la petición de confirmación para borrar una actividad.**

## 10.7. Calendario

El calendario se puede acceder de forma global por el gestor y el trabajador de forma global, mientras que el cliente solo puede ver el calendario filtrado por cada familiar que tenga vinculado.

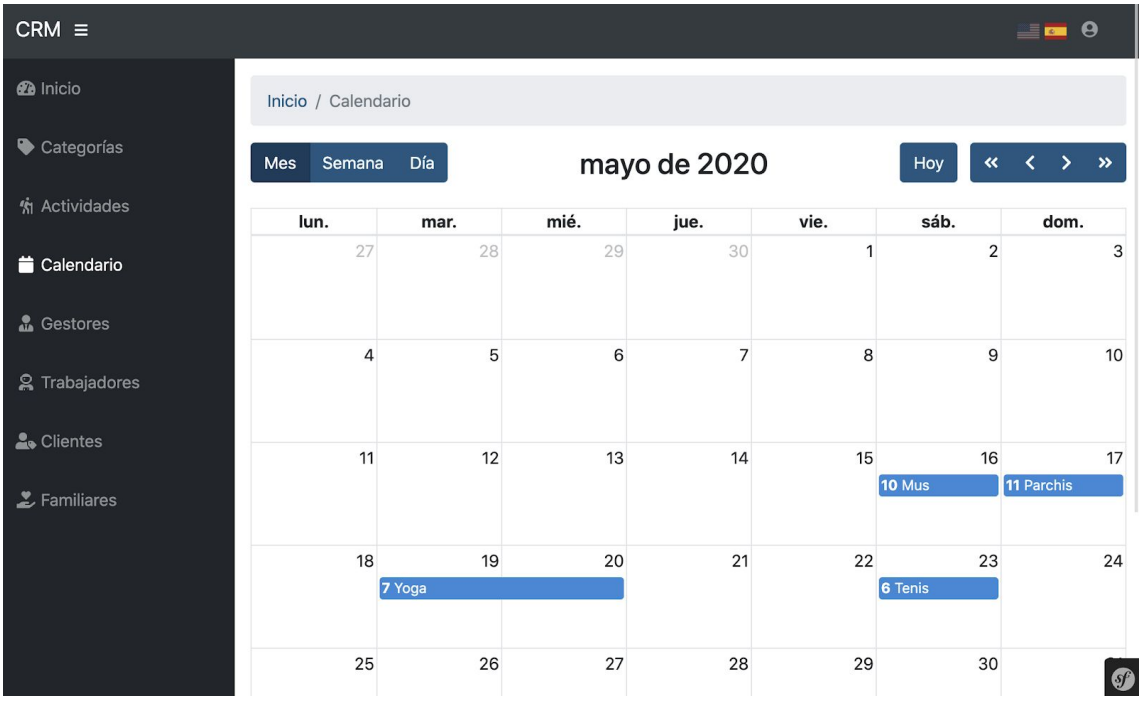


**Fig. 40 - Captura de pantalla del calendario general de la aplicación.**

Se puede ver que el calendario ofrece la posibilidad de tener tres tipos de vistas:

- Mensual
- Semanal
- Diaria

Además permite moverse por los meses y mostramos la duración de cada actividad creada en el sistema.

## 10.8. Gestores

Esta sección es solo accesible para el gestor, permite la creación de gestores. En esta sección se puede listar gestores y para cada una de ellos editar, crear, borrar y visualizar la ficha.

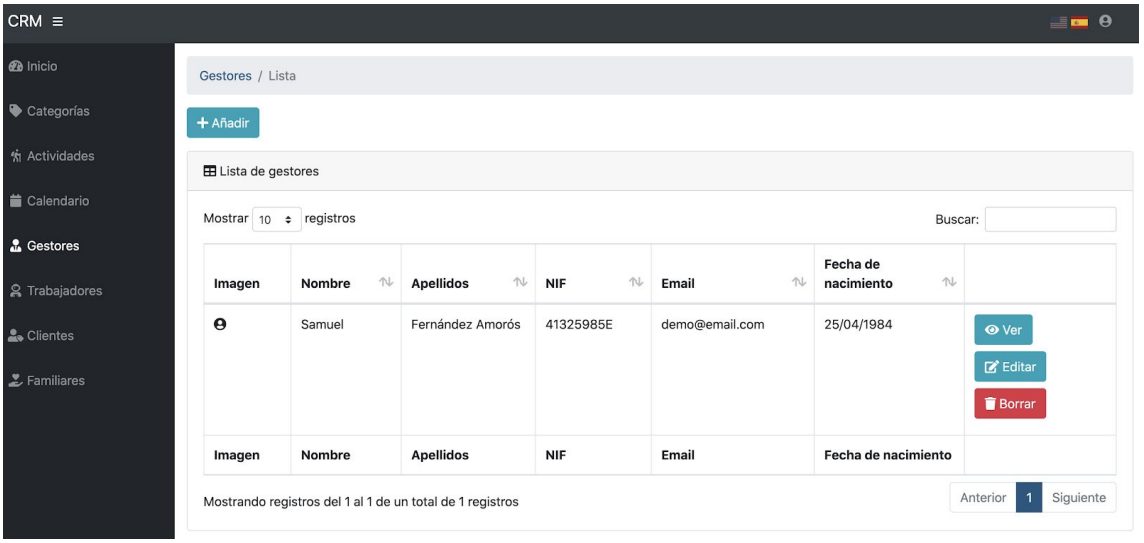


**Fig. 41 - Captura de pantalla con el listado de gestores.**


**Fig, 42 . Captura de pantalla con la vista detallada de un gestor.**

## 10.9. Trabajadores

Esta sección es solo accesible para el gestor, permite la creación de trabajadores. En esta sección se puede listar trabajadores y para cada una de ellos editar, crear, borrar y visualizar la ficha.

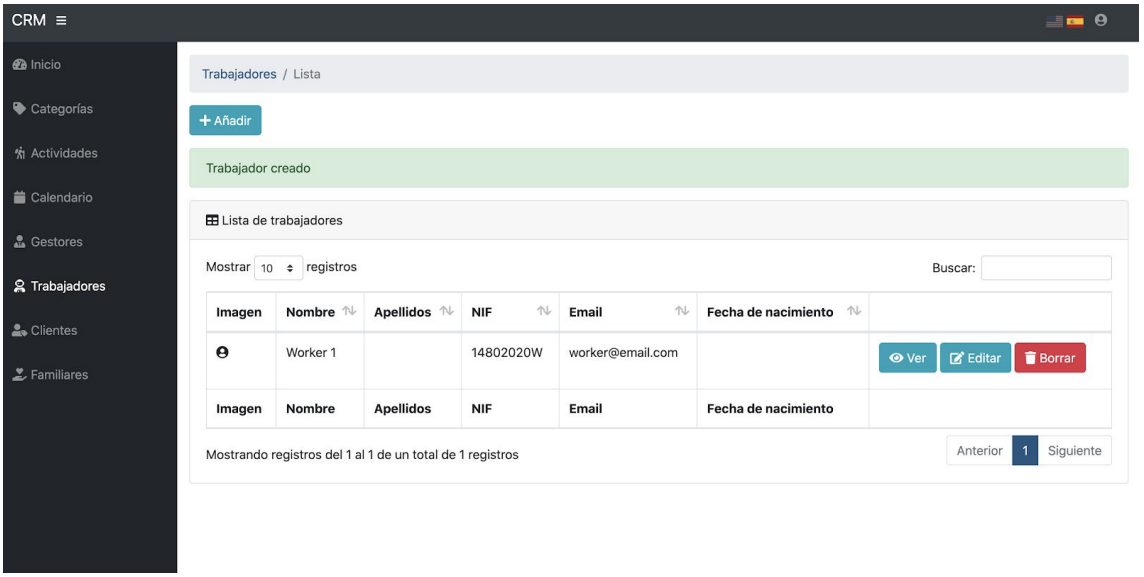


**Fig. 43 - Captura de pantalla con el listado de trabajadores y la confirmación de creación de un trabajador.**

## 10.10. Clientes

Esta sección es accesible para el gestor y el trabajador, permite la creación de clientes. En esta sección se puede listar clientes y para cada una de ellos editar, crear, borrar y visualizar la ficha.

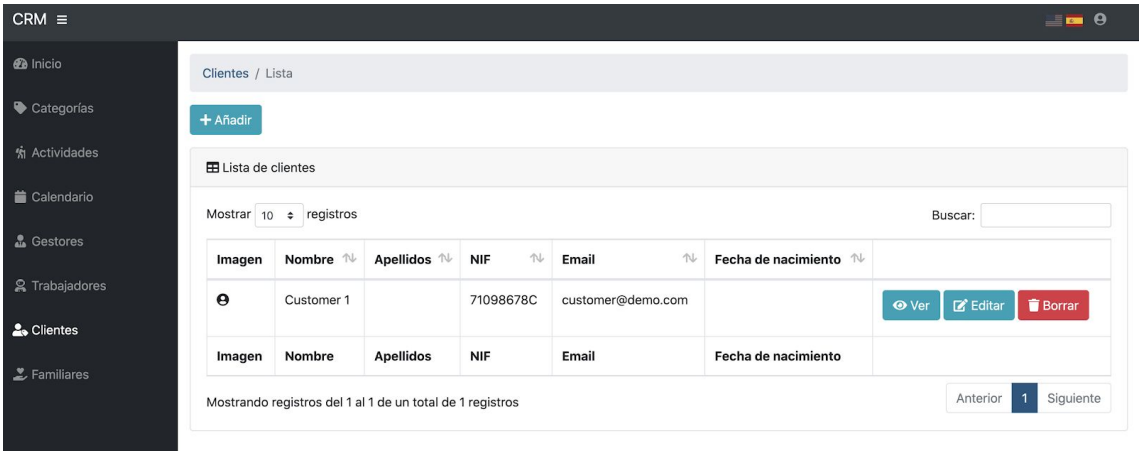


**Fig. 44 - Captura de pantalla del listado de clientes.**

## 10.11. Familiares

Esta sección es accesible para todos los usuarios, permite la creación de familiares. En esta sección se puede listar familiares y para cada una de ellos editar, crear, borrar y visualizar la ficha.

Cabe destacar, que cuando accede el cliente a esta sección solo puede ver los familiares vinculados y en ningún caso puede crear, editar o borrar los familiares.


**Fig. 45 - Captura de pantalla del listado de familiares.**

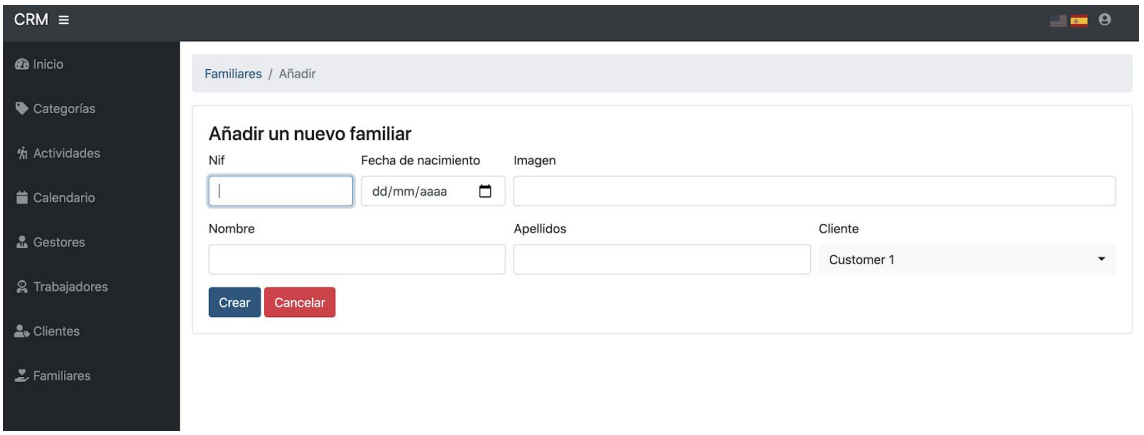


**Fig. 46 - Captura de pantalla del formulario para añadir un nuevo familiar.**

En la siguiente imagen podemos observar un hilo cronológico de los diferentes sucesos que han ocurrido u ocurrirán en relación con el familiar.

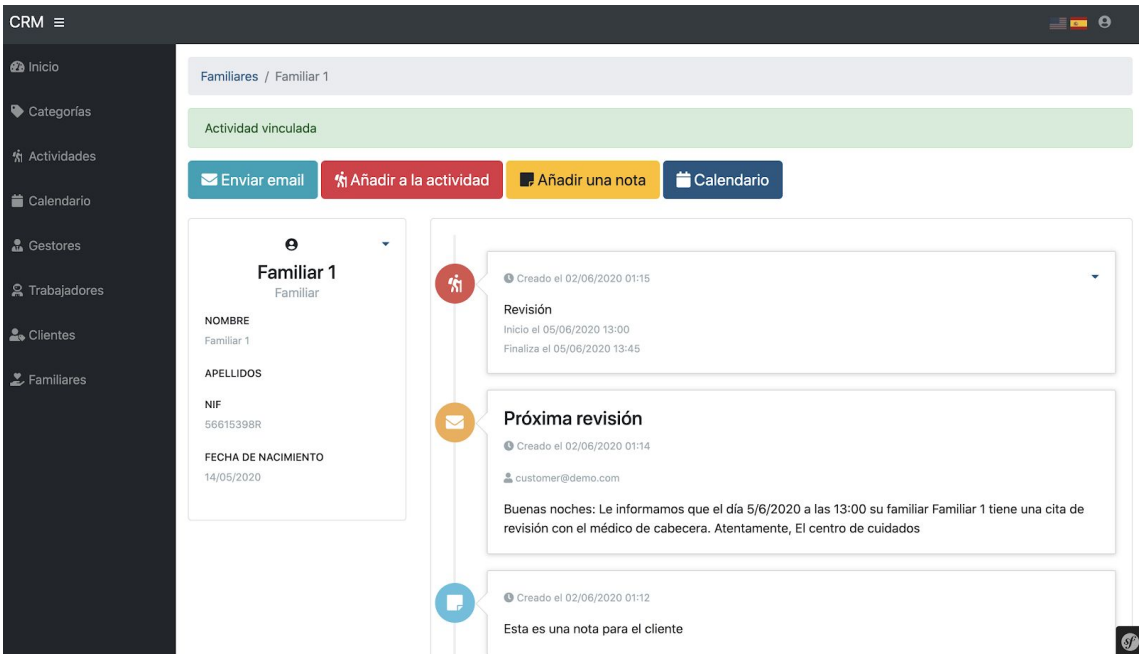


**Fig. 47 - Captura de pantalla de la vista detallada por orden cronológico de las diferentes acciones de un familiar.**

Por último, podemos ver la vista del calendario para cada uno de los familiares.

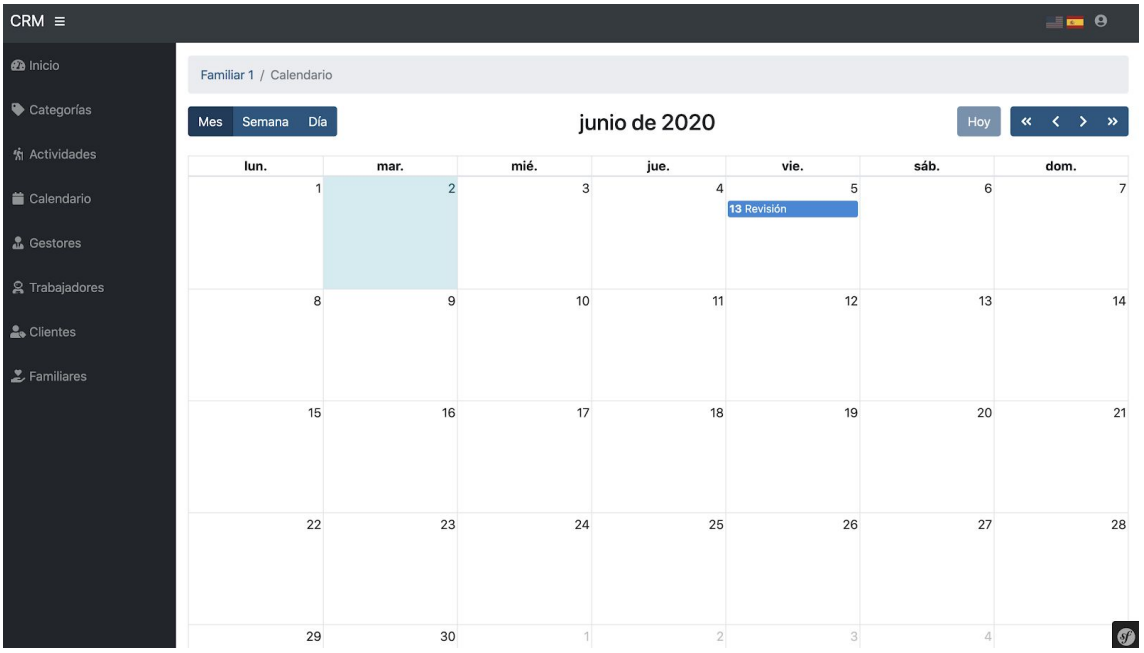


**Fig. 48 - Captura de pantalla del calendario de actividades de un familiar dado.**

# 11. Conclusiones

El proyecto de final de carrera se ha centrado en el desarrollo de un producto de mínima viabilidad, ya que el desarrollo de un producto de tipo CRM al completo están fuera del alcance por tiempo, recursos y dedicación necesarios.

El desarrollo del proyecto ha resultado una actividad interesante que me ha permitido trabajar en diferentes etapas del mismo, pasando desde el diseño inicial hasta el desarrollo y su despliegue final.

He podido comprobar que las decisiones iniciales de diseño afecta de una forma muy importante al desarrollo posterior que se realizará, conforme se ha ido profundizando más en el proyecto han salido a la luz pequeñas carencias que en siguientes iteraciones han sido subsanadas, por lo tanto aplicar un proceso iterativo y ágil en el desarrollo del proyecto ha sido un aspecto clave.

Los objetivos iniciales del proyecto se han cumplido satisfactoriamente conforme los tiempos planificados. Por lo tanto no ha requerido alterar la planificación inicial y se ha podido actuar conforme los pasos de tiempo indicados en la memoria.

Como se ha indicado al principio, se ha desarrollado un MVP. Por lo tanto, futuros pasos sería estudiar si el proyecto podría ser viable (desde un punto de vista económico, como técnico) y por lo tanto puede ser un producto que tenga una utilidad para terceros.

Por lo tanto será necesario realizar un proyecto de viabilidad económica, por ejemplo mediante el cobro de suscripciones periódicas y además estudiar si las características del MVP cubren la mayoría de necesidades de las empresas o se necesita ampliar el proyecto con algunas nuevas que no hubiesen sido recogidas inicialmente.

# 12. Glosario

**Ansible**: Es un software usado para gestionar y proporcionar configuraciones a servidores en la red. Fue desarrollado usando Python. Fue lanzado en 2012.

**Ansistrano**: Es un conjunto de roles usados para Ansible que permite facilitar las tareas de deploy de proyectos en servidores. Fue desarrollado por un grupo de españoles entre los que se encuentra Carlos Buenosvinos.

**Composer**: Es un software para gestionar dependencias en el lenguaje de programación PHP. Fue lanzado en 2012 por Nils Adermann y Jordi Boggiano.

**CPD**: Centro de procesamiento de datos, o en su siglas en inglés “Data center”. Son lugares especializados para almacenar la información en servidores.

**CRM**: De sus siglas en inglés Customer Relationship Management, es un software para gestionar las relaciones de los clientes con las empresas.

**Dashboard**: También se puede conocer como un tablero inicial que tienen los usuarios para mostrar estadísticas en la página principal de una aplicación web una vez se ha identificado en el sistema.

**DDD:** Diseño guiado por el dominio, viene de las siglas del inglés Domain Driven Design.

**Doctrine**: Es un ORM que está escrito para el lenguaje de programación PHP desde el año 2006.

**ECMAScript**: Es una especificación de lenguaje de programación, se desarrolló inicialmente en 1996 y fue publicado por ECMA International. Está aceptado como estándar ISO/IEC 22275:2018.

**Framework**: Es un marco o un entorno de trabajo que recoge una serie de conceptos, principios, prácticas y criterios, estos se ven plasmados en un software que facilita el trabajo a terceras personas.

**Fullcalendar**: Software desarrollado con el lenguaje de programación Javascript que proporciona el funcionamiento de un calendario en entornos webs.

**Hash**: Cadena de texto generada a partir del cifrado de otro texto general, no es posible revertir el cifrado, por lo que se suele generar un nuevo hash para validar que dos datos son iguales mediante la comparación del hash anterior y del nuevo.

**Javascript**: Lenguaje de programación interpretado soportado de forma muy amplia por los navegadores web, es un dialecto del estándar ECMAScript. Fue desarrollado en 1995.

**Landing**: Página principal de una aplicación web que es accesible de forma pública para todas las personas. Suele concentrar la información necesaria para explicar de qué tipo de producto se trata y cuales son sus características.

**ORM**: Conocido como mapeador de objetos-relacionales o en sus siglas en inglés Object-Relational mapping. Es una técnica usada para transformar datos de una base de datos relacional en elementos que determinados de un lenguaje de programación orientado a objetos.

**PHP**: Es un lenguaje de programación orientado a objetos. Fue diseñado inicialmente por Rasmus Lerdorf en 1995.

**Python**: Es un lenguaje de programación orientado a objetos. Fue diseñado inicialmente por Guido van Rossum en 1991.

**SASS**: Es un lenguaje de programación para generar código preprocesado de CSS. Sus siglas viene de “Syntactically Awesome Stylesheets”. Fue diseñado por Hampton Catlin en 2006.

**SOLID**: Es un acrónimo mnemotécnico formado por cinco principios “Single responsibility principle “, “Open-closed principle”, “Liskov substitution principle”, ”Interface segregation principle”, y “Dependency inversion principle”.

**Symfony**: Es un framework de trabajo desarrollado para el lenguaje PHP desde el año 2005.

**Template**: Es un sistema que permite gestionar pequeños trozos de código en forma de plantillas que serán reutilizados mediante programación y que se usan para mostrar información a los usuarios.

**Terraform**: Es un software usado para escribir mediante código los elementos necesario para trabajar con la infraestructura de los CPD. Fue desarrollado por HashiCorp.

**Twig**: Es un motor de templates escrito para el lenguaje PHP y que viene integrado con el framework Symfony.

**URI**: Se conoce por las siglas de “Uniform Resource Identifier”, se trata de un grupo de tipos de identificadores que engloban los tipos URL y URN.

**URL**: Se conoce por las siglas de “Uniform Resource Locator”, se trata de una ruta que nos permite localizar un recurso en Internet. Forma parte del grupo de URI.

**Webpack**: Es un software usado para transformar y empaquetar código en diferentes lenguajes como SASS o Typescript en CSS y JS respectivamente.

**Yarn**: Es un sistema para realizar la instalación de dependencias de Javascript. Fue creado por Facebook junto con otros desarrolladores como Google.

# 13. Bibliografia


[1] PHP: What is PHP?, (s.f)., n/a, URL: https://www.php.net/manual/en/intro-whatis.php

[2] What is Symfony, (s.f)., n/a, URL: https://symfony.com/what-is-symfony

[3] MySQL :: MySQL 8.0 Reference Manual :: 1.3.1 What is MySQL?, (s.f)., n/a, URL: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

[4] Introduction to Redis - Redis, (s.f)., n/a, URL: https://redis.io/topics/introduction

[5] What is Docker? | Opensource.com, (s.f)., n/a, URL: https://opensource.com/resources/what-docker

[6] What is Git: become a pro at Git with this guide | Atlassian Git Tutorial, (s.f)., n/a, URL: https://www.atlassian.com/git/tutorials/what-is-git

[7] Hummel G., 2019, What is Ansible and what can it automate? - Cloud Academy Blog, URL: https://cloudacademy.com/blog/what-is-ansible/

[8] Introduction - Terraform by HashiCorp, (s.f)., n/a, URL: https://www.terraform.io/intro/index.html

[9] Rascia, T., 2015, What is Bootstrap and How Do I Use It? – Tania Rascia, URL: https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/

[10] Introduction - Composer , (s.f)., n/a, URL: https://getcomposer.org/doc/00-intro.md

[11] Ferrer, J. , 2016, Introducción Arquitectura Hexagonal - DDD | CodelyTV, URL: https://codely.tv/blog/screencasts/arquitectura-hexagonal-ddd/

[12] Janssen, T., 2020, SOLID Design Principles: The Single Responsibility Explained, URL: https://stackify.com/solid-design-principles/

[13] Fowler, M., 2012, "TestPyramid - Martin Fowler.", URL: https://martinfowler.com/bliki/TestPyramid.html.

[14] Evans, E., Domain-Driven Design - Tackling Complexity in the Heart of Software, 2004, Addison-Wesley.
ISBN: 978-0321125217

[15] Vernon, V. , Implementing Domain-Driven Design, 2013, Addison-Wesley.
ISBN: 978-0321834577

[16] Buenosvinos, C., Soronellas, C., Akbary, K., Domain-Driven Design in PHP, 2017, Packt Publishing.
ISBN: 978-1787284944

[17] Martin, Robert C., Clean Code: A handbook of Agile Software Craftsmanship, 2008, Prentice Hall.
ISBN: 978-0132350884