



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Aplicación Web desarrollada en PHP
usando el Framework Symfony

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ramon Guerrero Oriola

Tutor: José Vicente Busquets Mataix

2017-2018

Resumen

Desarrollar una aplicación web para la gestión de los registros de acceso en todas las sedes de una empresa, utilizando el *framework* PHP Symfony.

En todas las sedes de la empresa, se habilitará un ordenador donde permitir al empleado registrar su entrada o salida de la misma. Para ello el empleado tendrá que introducir su número de DNI y, posteriormente validar el formulario.

La aplicación debe permitir al supervisor, listar de manera inmediata todos los presentes en un local con el fin de evacuarlo en caso de emergencia.

Por otro lado, el administrador podrá acceder a la gestión de todos los empleados y sedes. Además, el administrador tendrá la opción de listar los registros de todas las sedes, incluyendo un filtrado por fecha.

Palabras clave: symfony, php, html, registro, apache.

Abstract

Develop a web application to manage every access record in all the headquarters of a company, using the Symfony PHP framework.

Every company's location will have available a computer to allow the employee to register their entry or exit. The employee will have to enter his ID number and, subsequently, validate the form.

The application must allow the supervisor to list, immediately, all employees present in a HQ, in order to evacuate in case of emergency.

On the other hand, administrator can manage all employees and HQ. In addition, the administrator will have the option to list the records of every site, including filtering by date.

Keywords: symfony, php, html, log, apache.

Tabla de contenidos

Contenido

I.	Introducción.....	10
1 -	Introducción	10
1.1	Motivación.....	10
1.2	Motivación personal	10
1.3	Objetivo	10
2 -	Symfony	11
2.1	Otros proyectos con Symfony	11
II.	Especificación de requisitos	16
1 -	Introducción	16
1.1	Propósito.....	16
1.2	Alcance.....	16
1.3	Definiciones, siglas y abreviaturas	16
1.4	Referencias	17
1.5	Visión global	17
2 -	Descripción general.....	17
2.1	Perspectiva del producto.....	17
2.2	Funciones del producto.....	17
2.3	Características de los usuarios.....	17
2.4	Restricciones	17
2.5	Suposiciones y dependencias	17
2.6	Requisitos futuros.....	18
3 -	Requisitos específicos	18
3.1	Interfaces externas.....	18
3.2	Funciones.....	18
III.	Análisis	22
1 -	Introducción	22
2 -	Diagrama de clases.....	22
3 -	Diagramas de casos de uso.....	18
3.1	Caso base.....	18
3.2	Casos de Uso	22
4 -	Diagramas de actividad.....	23
4.1	Grabar registro.....	23
4.2	Realizar login administrador / supervisor.....	24
4.3	Alta usuario	25
4.4	Modificar o eliminar usuario	26

4.5 Alta sede	27
4.6 Modificar o eliminar sede.....	28
4.7 Listar registros desde supervisor	29
4.8 Listar registros desde administrador.....	30
IV. Diseño.....	31
1 - Introducción	31
2 - Patrón de diseño.....	31
2.1 Modelo.....	31
2.2 Vista	31
2.3 Controlador.....	32
V. Implementación.....	33
1 - Entorno de desarrollo	33
1.1 Apache y certificado SSL.....	33
1.2 PHP 7.0.....	34
1.3 MYSQL y PHPMYADMIN.....	34
1.4 Configuración del proyecto	34
2- Estructura Symfony.....	35
3- Creación de entidades	36
3.1 Introducción.....	36
3.2 Creación de las entidades	36
3.3 Carga del esquema en BBDD.....	42
3.4 Entity Manager	43
4- Bundles	45
4.1- Creación de un nuevo Bundle	45
4.2- Activación de Bundles	45
5- Rutas y controladores.....	47
5.1- Definición de rutas y controladores	47
5.2- Utilización rutas para sedes.....	48
5.3- Configuración adicional	50
6- Vistas.....	51
6.1- Bootstrap	51
6.2- Uso básico de las plantillas	52
6.3- Panel de administrador	53
6.4- Panel de supervisor	55
VI. Pruebas	56
1.1- Pruebas de registro y login.....	56
1.2- Pruebas de carga de la aplicación.....	57
1.3- Carga en dispositivos móviles.....	58
VII. Conclusiones	60



I. Introducción

1 - Introducción

Vamos a introducir las motivaciones principales por las que se ha elegido este proyecto.

1.1 Motivación

La aplicación desarrollada parte de la necesidad de tener cuantificados todos los empleados presentes dentro de cualquier ubicación de una empresa. Por motivos de riesgos laborales, se considera necesario disponer, además, de una utilidad que permita, en caso de evacuación, tener controlado que todos los usuarios hayan abandonado las instalaciones y evitar así cualquier daño personal.

1.2 Motivación personal

Personalmente, la elección del siguiente proyecto viene motivada por el interés en el desarrollo de aplicaciones web y en concreto con el lenguaje PHP. Además, en mi trabajo surgió la necesidad de tener una aplicación similar a la que se ha desarrollado en este trabajo y, aunque ya se optó por una solución diferente, creí conveniente intentar solucionarlo con una aplicación web.

1.3 Objetivo

El principal objetivo es la implementación de un sistema de control presencial que gestione las entradas y salidas del personal de la empresa en todas las ubicaciones de la misma, basándose en Symfony[1].

Symfony es un *framework* PHP basado en la arquitectura MVC (Modelo-Vista-Controlador) diseñado para desarrollar aplicaciones web. Además, cumple en sus requisitos el hecho de ser fácil de instalar, ser independiente de la capa de gestión de BBDD y fácil de extender e integrar con bibliotecas de otros fabricantes.

Para el desarrollo del siguiente proyecto en Symfony y de su memoria, hemos necesitado:

- Un servidor web, en nuestro caso Ubuntu 16.04.
- Instalación de Linux, php, mysql y apache (LAMP).
- Procesador de textos.
- Herramientas de mockup. Por ejemplo, moqups.com.
- Herramientas para la generación de diagramas UML. En nuestro caso, DIA (opensource).

2 - Symfony

Symfony dispone de diversas versiones en el mercado de las cuales hemos elegido la 3.4 LTS dado que tiene soporte hasta noviembre de 2020 y además es una versión de la que hemos encontrado mucha documentación. Actualmente, el *framework* se encuentra en la versión 4.1 la cual supone un enorme cambio respecto a la anterior, vamos a repasar los puntos más diferenciales:

- Estructura de directorios más simple
- Aplicaciones más pequeñas, dado que se instalan los componentes mínimos.
- Symfony Flex, permite la instalación de *bundles*, componentes y librerías de terceros con mayor facilidad.

2.1 Otros proyectos con Symfony

A partir de Symfony, existen muchos proyectos que utilizan partes del mismo dentro de sus entornos, como son:

- Drupal
- Laravel
- Joomla
- PHPBB
- Magento
- Composer
- PrestaShop
- CakePHP
- Facebook Ads API SDK
- Google Cloud Platform SDK
- phpMyAdmin
- Doctrine

Todos estos conocidos *framework*, CMS, SDK... utilizan componentes de Symfony en sus proyectos. Por ejemplo, *Routing*, *Console*, *HttpFoundation*...

II. Especificación de requisitos

1 - Introducción

Para redactar esta ERS (Especificación de requisitos software) se ha seguido el estándar IEEE-STD- 830-1998 [2].

1.1 Propósito

La finalidad de este proyecto es servir de sistema de control presencial para la empresa cliente, para ello se pretende plasmar el funcionamiento actual de la empresa y conseguir que la adaptación al software sea lo más fácil y transparente posible.

Para la lectura y comprensión de este texto, es conveniente tener conocimientos de los lenguajes de programación PHP, HTML y JavaScript.

1.2 Alcance

El producto a desarrollar es un sistema de información (SI) que cubra las necesidades de la empresa. La aplicación desarrollada se llamará ControlPresencial usando una programación orientada a objetos, en PHP y utilizando el *framework* Symfony.

ControlPresencial tendrá como función permitir el registro de entradas y salidas del personal de la empresa. Estos registros serán guardados y, posteriormente, podrán ser filtrados por empleado, sede o por fecha, dando como resultado un listado de entradas y salidas. Por último, la aplicación permitirá a los administradores crear empleados y realizar cualquier modificación sobre ellos.

Además, en cada sede existirá un supervisor el cual podrá sacar un listado del personal que esté en ese momento dentro de las instalaciones.

1.3 Definiciones, siglas y abreviaturas

Symfony: Es un completo *framework* diseñado para desarrollar aplicaciones web. Además es el *framework* utilizado para el desarrollo de este proyecto.

Framework: Es una estructura conceptual para asistir, normalmente, con artefactos o módulos software que sirve para organizar y desarrollar software.

PHP: Es un lenguaje de código abierto para el desarrollo web.

JavaScript: Es un lenguaje de programación interpretado, orientado a objetos que se utiliza principalmente en el lado del cliente en la programación web.

MVC: El patrón modelo vista controlador de arquitectura de software separa los datos y la lógica de negocio de la interfaz. Es el modelo elegido para el desarrollo de esta aplicación.

Doctrine: Es un ORM escrito en PHP que se ubica justo encima de nuestro sistema de gestión de bases de datos y nos proporciona el acceso a la capa de persistencia.

MYSQL: Es el SGBD elegido para el desarrollo de esta aplicación.

1.4 Referencias

- IEEE-STD-830-1998
- Wikipedia
- <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>
- Libro Desarrollo web ágil con Symfony2 de Javier Eguiluz

1.5 Visión global

A continuación, exponemos una descripción general de la aplicación, sus funcionalidades y elementos que componen la misma.

2 - Descripción general

2.1 Perspectiva del producto

Esta aplicación es completamente independiente, por lo que no guarda relación con cualquier otro elemento software ya disponible en la empresa.

2.2 Funciones del producto

Esta aplicación web deberá permitir a los empleados de la empresa registrar los accesos a cualquier sede de la misma. Además, permitirá al supervisor sacar un listado de su sede asociada con el personal presente en la misma.

Por otro lado, los administradores de la empresa tendrán la gestión completa de usuarios, con permisos para crear, editar, actualizar y borrar cualquiera de ellos.

2.3 Características de los usuarios

El uso de la aplicación no requiere de conocimientos específicos informáticos por parte de los empleados generales. En cambio, para los administradores será necesaria una formación para el buen uso del panel de administración.

2.4 Restricciones

Para el desarrollo de la aplicación se imponen las siguientes restricciones:

- El tráfico generado por la aplicación web debe utilizar conexiones seguras.
- El panel debe adaptarse a cualquier pantalla, tanto ordenador, móvil, tablet...
- En cada sede se ubicará un equipo encargado de la recepción del registro por parte de los usuarios que entren a la misma.

2.5 Suposiciones y dependencias

El servidor web que aloja la aplicación será instalado y configurado en la empresa respetando las mismas versiones de sistema operativo, symfony, php...

del servidor de desarrollo. Con esto se garantiza la correcta puesta en marcha de la aplicación, para futuras actualizaciones, sería conveniente revisar el desarrollo para evitar problemas de funcionamiento.

2.6 Requisitos futuros

Si se considera necesario, este desarrollo podría ampliarse para contemplar un control horario además del presencial. Para ello se necesitaría, entre otros requisitos, definir el calendario de trabajo, las jornadas laborales definidas en la empresa y los diferentes perfiles de la misma.

3 - Requisitos específicos

3.1 Interfaces externas

Como hemos comentado con anterioridad, esta aplicación no se integra con otros elementos de software de la empresa, por lo que no tenemos ningún requisito específico en ese sentido.

La interfaz de usuario será simple y sencilla de usar, para el correcto funcionamiento es conveniente que en la entrada de las sedes, los equipos que tengan disponible el portal web de la aplicación dispongan de una pantalla de tamaño adecuado.

Por último, el acceso a esta aplicación está restringido a la red interna de la empresa por lo que, a nivel de comunicaciones, no necesitamos ninguna restricción adicional.

3.2 Funciones

A continuación, procedemos a especificar las acciones que va a llevar a cabo nuestra aplicación.

3.2.1 Grabar registro

Esta función consiste en grabar el registro de entrada o salida por parte del empleado:

Acción	Repercusión en la base de datos
Introducir el DNI del usuario en la página principal	
Pinchar en “Validar”	Consultar en BBDD si existe el usuario Correcto: Se registra el evento Error: Indicamos al usuario que ha habido un error y le damos la opción de reportarlo
Redirigir a la página principal	

3.2.2 Realizar login administrador/supervisor

Esta función consiste en acceder al panel de administrador o supervisor, mediante el formulario de login.

Acción	Repercusión en la base de datos
Pinchar en “login” en la página web principal	
Introducir DNI y password	Consultar en BBDD si son válidos los datos Correcto: Se redirige al panel. Error: Indicamos al usuario que ha habido un error y reenviamos a la ventana de login.
Redirigir al panel correspondiente o a ventana de login.	

3.2.3 Resetear password

El administrador o supervisor puede resetear su contraseña en caso que lo considere necesario:

Acción	Repercusión en la base de datos
Pinchar en “login” en la página web principal	
Pinchar en “Olvidé mi contraseña”	
Introducir DNI y seleccionar recordar	Consultar en BBDD si existe el DNI Correcto: Enviar correo de reseteo Error: Indicamos al usuario que ha habido un error y reenviamos a la ventana de login.
Redirigir a la ventana principal.	

3.2.4 Alta usuario

Esta tarea consiste en la generación de un nuevo usuario, desde el panel de administración:

Acción	Repercusión en la base de datos
Pinchar en <i>Crear usuario</i>	
Rellenar el formulario	
Pinchar en crear	Consultar en BBDD que no existe el DNI. Correcto: Generar usuario Error: Indicamos al usuario que ha habido un error.
Redirigir al panel de administración.	

3.2.5 Buscar usuario

Esta tarea permite la búsqueda de un usuario existente:

Acción	Repercusión en la base de datos
Pinchar en <i>Buscar usuario</i>	
Buscamos el usuario por DNI, Nombre, Apellidos o tipo	
Pinchar en buscar	Consultar en BBDD por el filtro seleccionado Correcto: Mostrar usuario Error: Indicamos al usuario que ha habido un error.
Redirigir al panel de edición de usuario o al panel principal.	

3.2.6 Modifica o eliminar usuario

Tras la búsqueda de un usuario podemos editarlo o eliminarlo:

Acción	Repercusión en la base de datos
Pinchar en <i>Buscar usuario</i>	
Buscamos el usuario por DNI, Nombre, Apellidos o tipo	
Pinchar en buscar	Consultar en BBDD por el filtro seleccionado Correcto: Mostrar usuario Error: Indicamos al usuario que ha habido un error.
Redirigir al panel de edición de usuario o al panel principal.	
En el panel de edición modificamos los campos necesarios y damos a actualizar o eliminar.	Realizamos la modificación en BBDD.

3.2.7 Alta sede

Esta tarea consiste en la generación de una nueva sede, desde el panel de administración:

Acción	Repercusión en la base de datos
Pinchar en <i>Administrar sedes</i>	
Pinchar en <i>Alta sede</i>	
Introducimos los valores solicitados	Consultar en BBDD que no existe la misma sede Correcto: Generar sede Error: Indicamos al usuario que ha habido un error.
Redirigir al panel de administración.	

3.2.8 Modificar o eliminar sede

Tras la búsqueda de un usuario podemos editarlo o eliminarlo:

Acción	Repercusión en la base de datos
Pinchar en <i>Administrar sede</i>	
Buscamos la sede a editar en el listado	
Pinchar en modificar	
Modificamos los campos necesarios y le damos a Modificar o Borrar	Realizamos la modificación en BBDD.
Redirigimos al panel de administración	

3.2.9 Listar registros desde supervisor

Esta tarea es la única disponible para un supervisor, le permite listar a todos los usuarios presentes en la sede en el momento actual.

Acción	Repercusión en la base de datos
Vamos al panel de supervisor	
Pinchamos en Listar usuarios	Se consulta en BBDD todos los usuarios presentes en la sede asignada al supervisor.

3.2.10 *Listar registros desde administrador*

Desde el panel de administrador, se pueden listar los registros, filtrando por sede y fecha.

Acción	Repercusión en la base de datos
Vamos al panel de administrador	
Pinchamos en Listar registros	
Elegimos los filtros y le damos a <i>Listar</i>	Se consulta en BBDD los registros que cumplen los filtros.

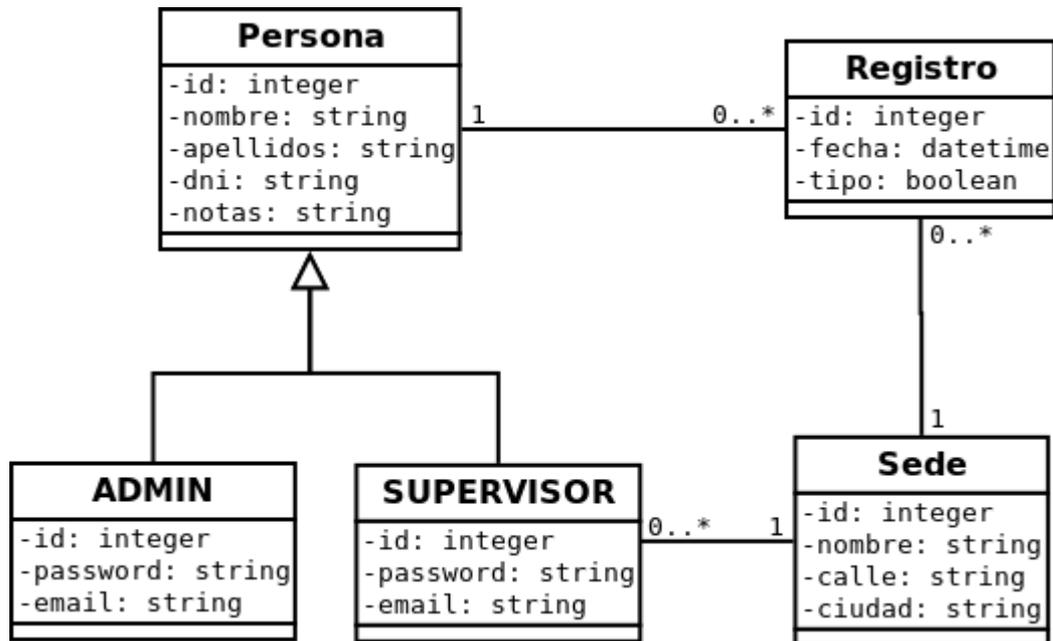
III. Análisis

1 - Introducción

Vamos a proceder al análisis de la aplicación para lo que vamos a emplear los diagramas necesarios que faciliten la labor posterior de diseño.

2 - Diagrama de clases

El diagrama de clases propuesto para la aplicación es el siguiente:

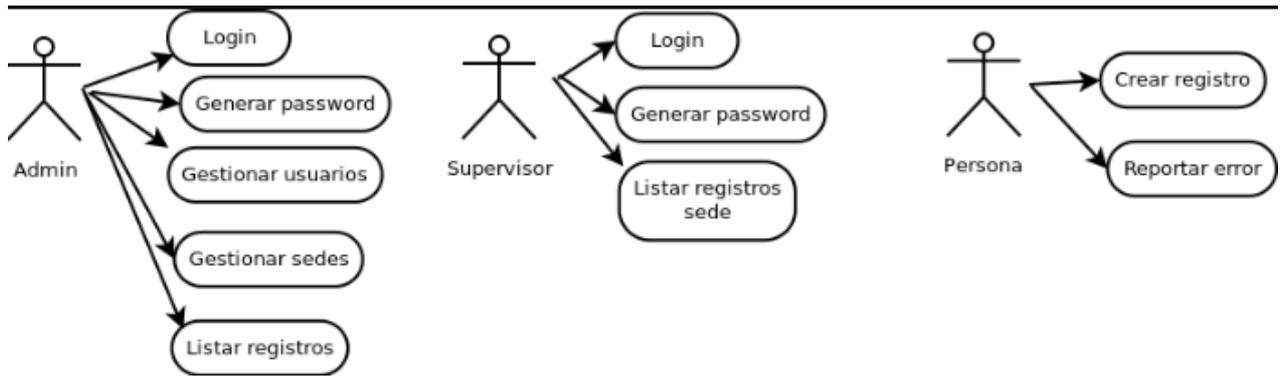


3 - Diagramas de casos de uso

El diagrama de casos de uso muestra los posibles escenarios que tienen los usuarios de la aplicación y las tareas que pueden realizar.

3.1 Caso base

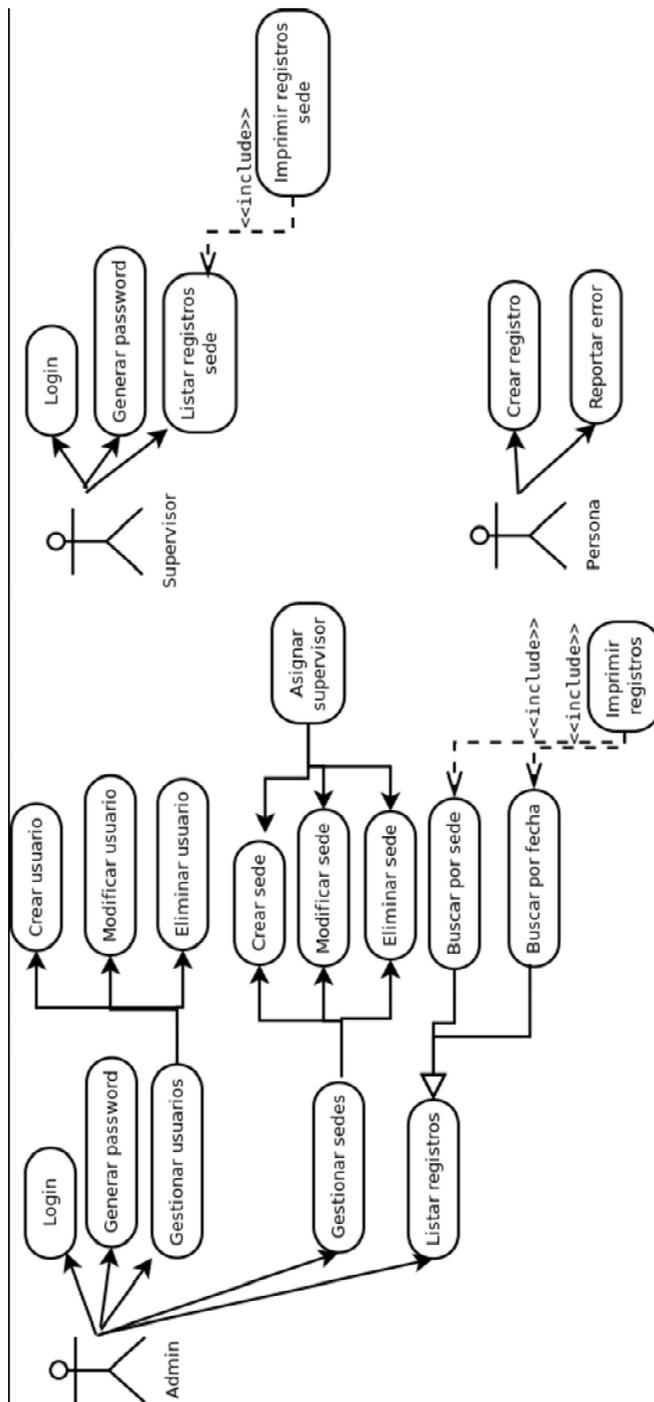
Este es el caso de uso desde el que partimos, con los tres actores posibles de la aplicación.



).

3.2 Casos de Uso

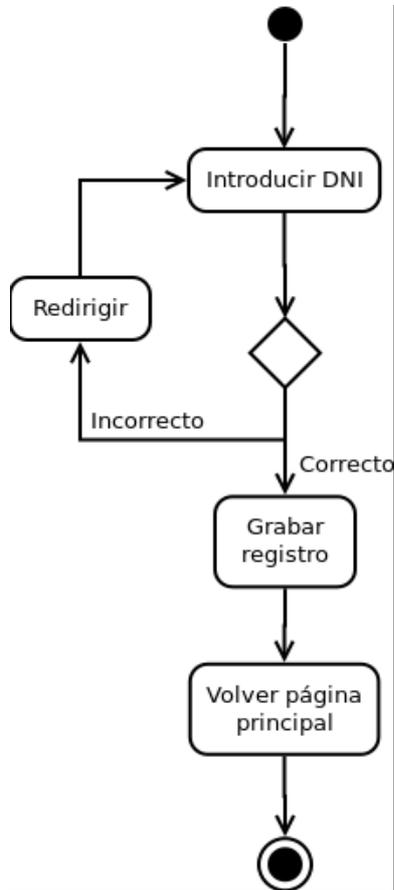
En este diagrama, definimos todas las funcionalidades disponibles en la aplicación:



4 - Diagramas de actividad

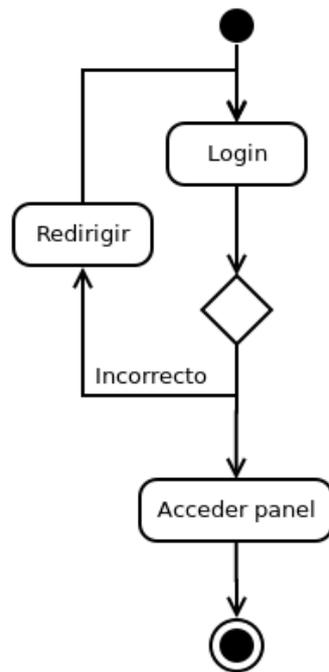
A continuación, vamos a utilizar diagramas de actividad para definir las interacciones más complejas dentro de la aplicación.

4.1 Grabar registro



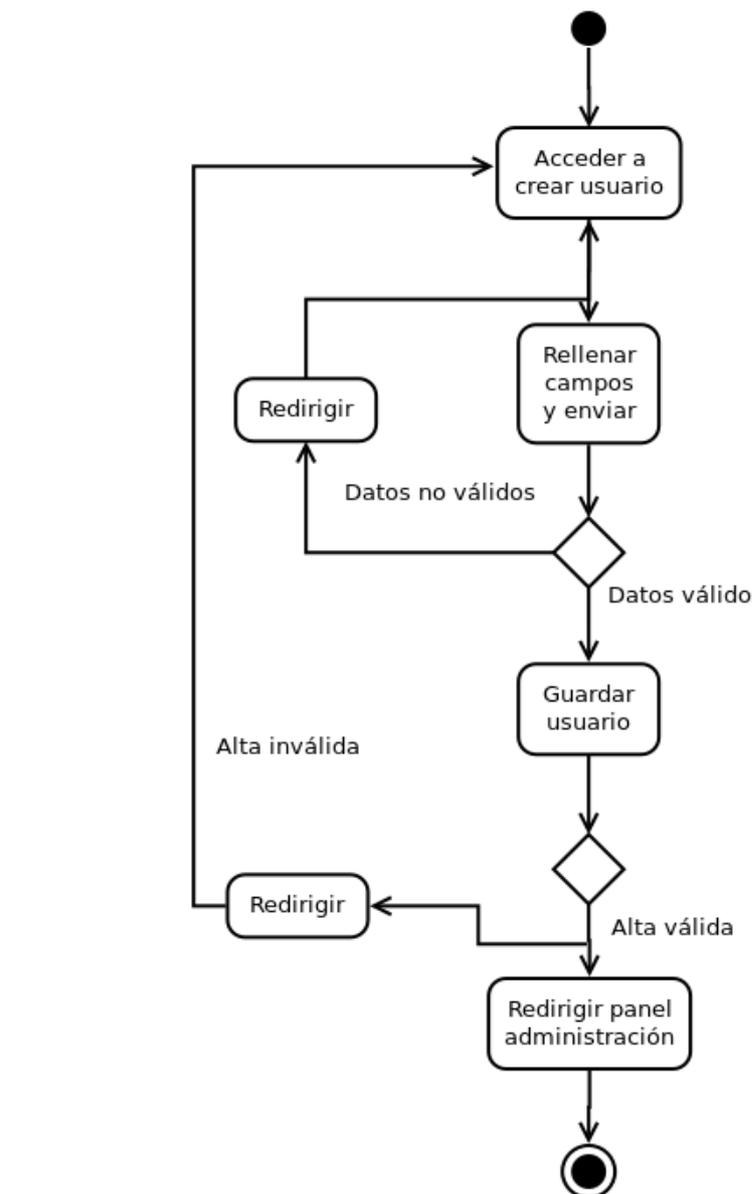
El empleado introduce su DNI en la plataforma, a continuación comprobamos que dicho DNI existe en BBDD, si es así grabamos el registro, en caso contrario volvemos al inicio.

4.2 Realizar login administrador / supervisor



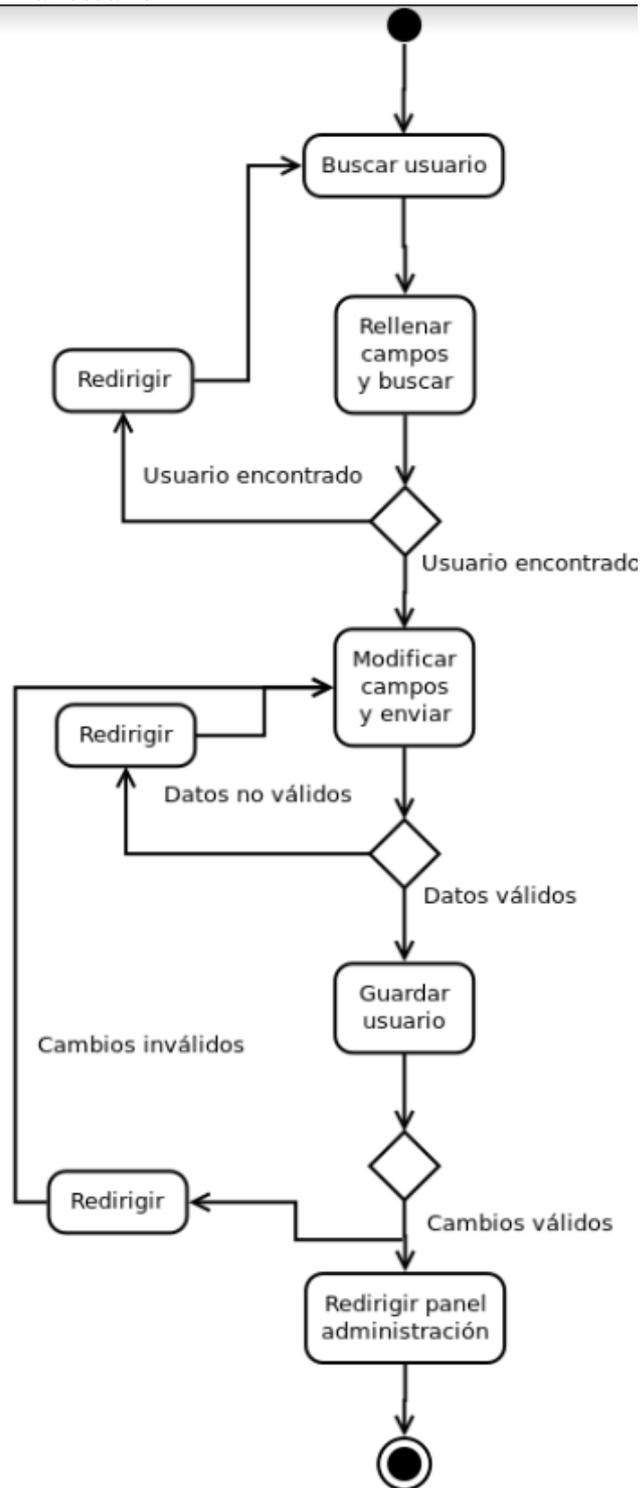
El admin o supervisor introduce su DNI y password en la plataforma, comprobamos que son correctos y en caso de serlo, redirigimos al panel pertinente.

4.3 Alta usuario



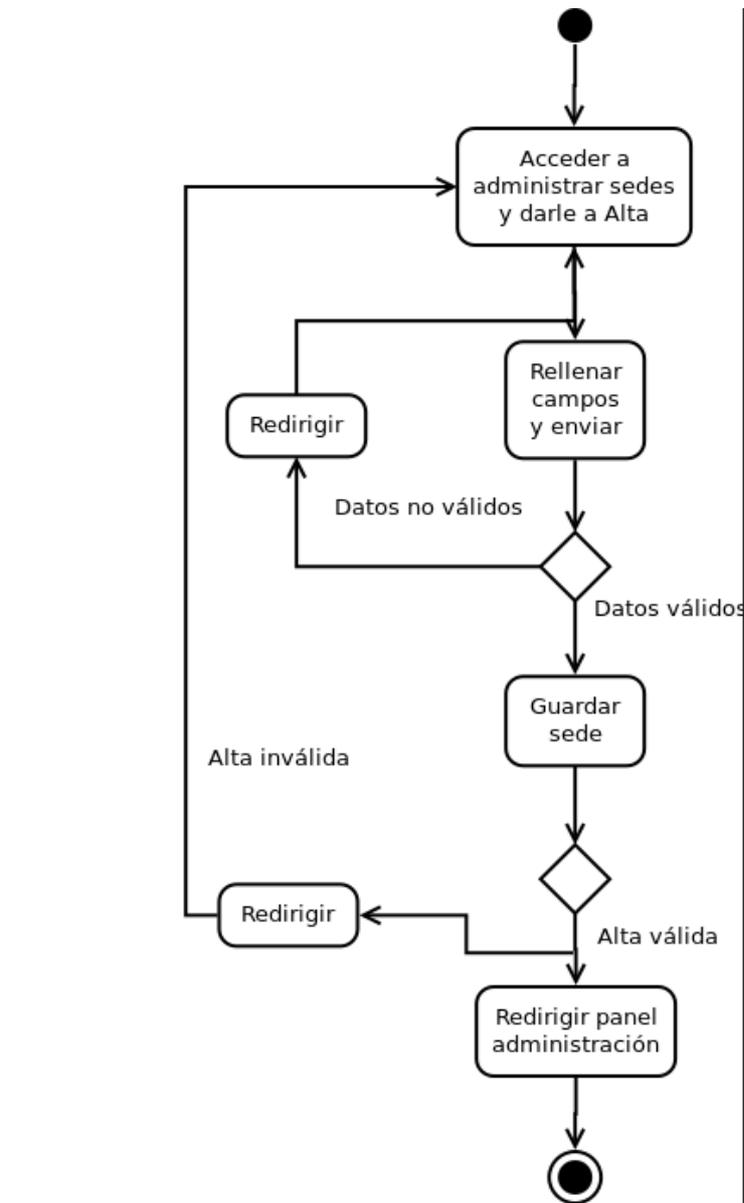
El administrador desde el panel accede a crear un nuevo usuario, rellena el formulario con los datos pertinentes y, si son correctos intentamos guardar el usuario, si el alta es correcta, reenviamos al panel de administración.

4.4 Modificar o eliminar usuario



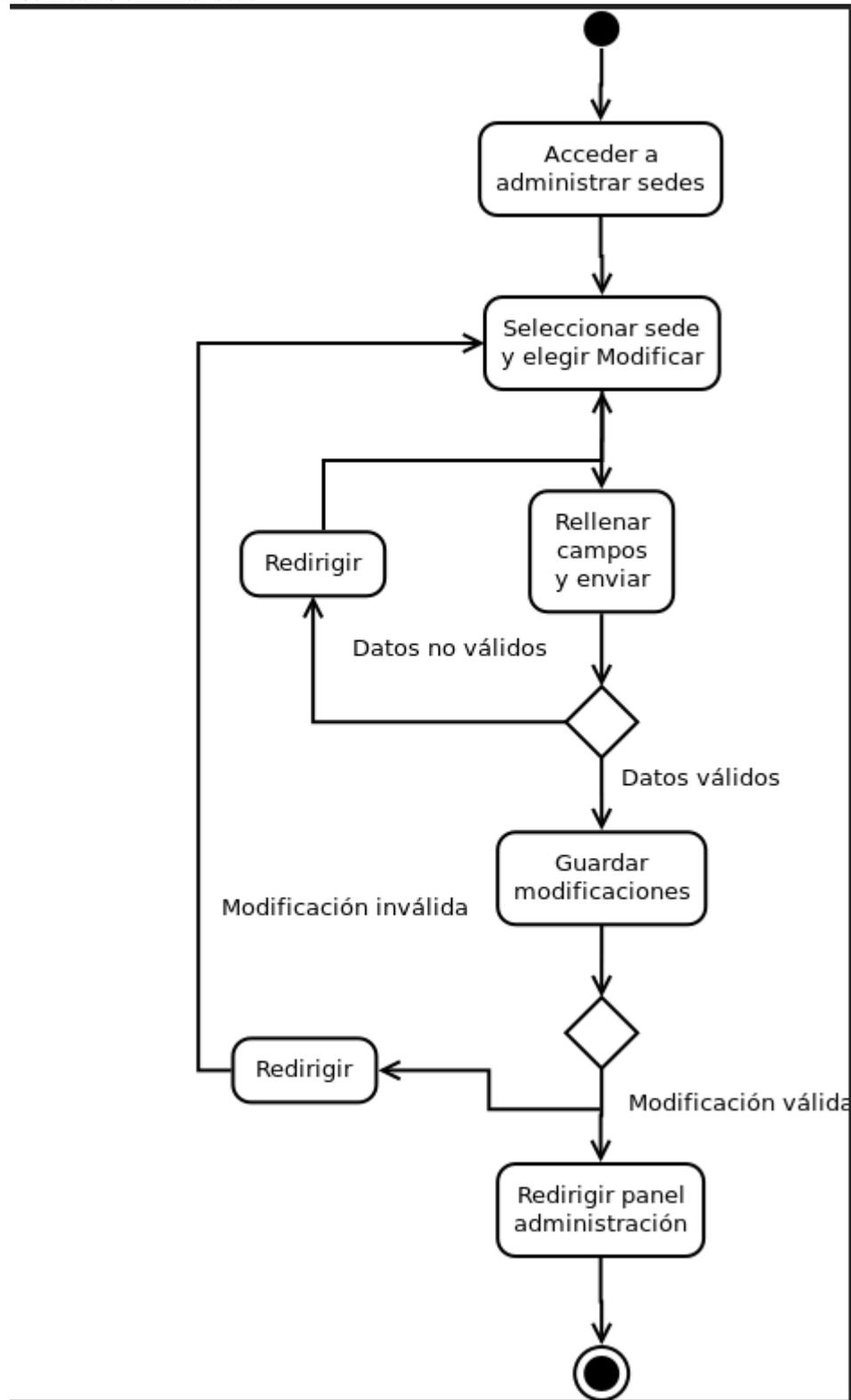
El administrador desde el panel busca un usuario, rellena el formulario con los datos pertinentes y, si encontramos el usuario le permitimos la edición del mismo. Al finalizar, se intentan guardar los cambios, si el cambio es correcto se redirige al panel.

4.5 Alta sede



El administrador desde el panel elige administrar sede y dar de alta, rellena el formulario con los datos pertinentes y, si los datos son válidos intentamos guardar en BBDD la sede nueva, si es correcto, se le redirige al panel de administración.

4.6 Modificar o eliminar sede



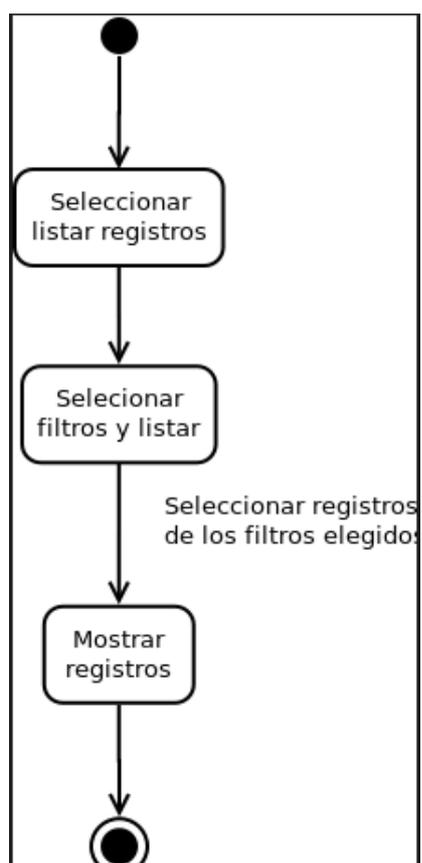
El administrador desde el panel elige una sede para editar al finalizar, se intentan guardar los cambios, si el cambio es correcto se redirige al panel.

4.7 Listar registros desde supervisor



El supervisor, tras el login, elige la opción para listar usuarios, seleccionamos la sede asociada al supervisor y se le muestran los registros de la misma.

4.8 Listar registros desde administrador

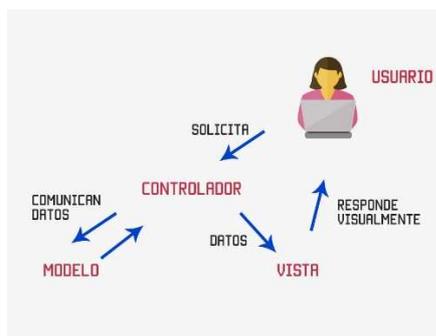


El administrador desde el panel elige listar registros, a continuación selecciona la sede y los filtros de fecha deseados. A continuación, se le muestran los registros elegidos.

IV. Diseño

1 - Introducción

El diseño de esta aplicación ha seguido, como hemos comentado anteriormente, el patrón Modelo Vista Controlador. Dicho patrón separa la interfaz del usuario, la lógica de negocio y los datos de la propia aplicación. Podemos ver en el siguiente gráfico una representación del modelo:



2 - Patrón de diseño

2.1 Modelo

Esta capa es la encargada de proporcionar la persistencia de la información, en el modelo MVC. En Symfony las aplicaciones definen el modelo en base a las clases creadas, las cuales se traducen en tablas en la propia BBDD. Para dicha tarea utilizamos el ORM Doctrine, mediante el cual generamos la estructura de la BBDD de nuestra aplicación.

En nuestra aplicación las clases contenidas en esta capa son:

- Persona: En esta clase guardamos los datos de los usuarios pertenecientes a la empresa.
- Admin y supervisor: Estas clases heredan de Persona, ambas guardan los datos específicos para administradores y supervisores.
- Registro: Para almacenar los registros de entrada / salida de todos los usuarios.
- Sede: Aquí almacenamos los datos relativos a las sedes de la empresa.

2.2 Vista

La capa de vista, también conocida como presentación, es la interfaz que utilizará el usuario para interactuar con el sistema. Es decir, es la parte visual de nuestra aplicación donde presentamos los menús y las opciones para permitir a los usuarios utilizar la aplicación.

En Symfony utilizamos plantillas, archivos de tipo TWIG, para definir el aspecto que presentaremos. Además, necesitamos que la aplicación sea *responsive*, adaptándose a todo tipo de pantallas, para permitir la correcta visualización en terminales móviles, tablets...

2.3 Controlador

El controlador es el encargado de atender los eventos producidos en el sistema, realiza peticiones a la capa modelo para consultar datos y pasando la información necesaria para que la capa de presentación pueda presentarlos.

En Symfony los controladores contienen el código PHP necesario para realizar las tareas necesarias de esta capa. Por otro lado, Symfony funciona con *routing*, donde definimos las direcciones URL que atenderá nuestra aplicación. La parte principal de dicho *routing* se configura en el archivo *routing.yml*, por ejemplo:

```
logincheck:  
  
  path: /logincheck  
  
  defaults: { _controller: AppBundle:Default:logincheck }
```

Esta ruta define el PATH */logincheck* y lo envía al controlador Default dentro del *bundle AppBundle*. Dentro de este *bundle*, definimos la función a realizar cuando se reciba una petición:

```
public function logincheckAction(){  
  
    return $this->render('default/logincheck.html.twig', [  
  
        'base_dir' => realpath($this->getParameter('kernel.project_dir')).D$  
  
    ]);}
```

V. Implementación

1 - Entorno de desarrollo

Partimos de una instalación limpia de Ubuntu 16.04 LTS, actualizada con los últimos parches de seguridad de la distribución. Necesitaremos pues configurar apache [6], mysql y php en nuestro servidor.

Detallamos los pasos seguidos.

1.1 Apache y certificado SSL

Vamos a generar un certificado SSL autofirmado:

```
$openssl req -x509 -nodes -newkey rsa:2048 -keyout tfg.local.key -out tfg.crt
$mv tfg.* /etc/ssl/certs/
```

Instalamos apache y configuramos los módulos rewrite y SSL:

```
$apt-get install apache2
$a2enmod rewrite
$a2enmod ssl
```

Creamos el archivo de configuración del sitio `/etc/apache2/sites-available`, en este fichero de configuración vamos a redirigir todo el tráfico no cifrado hacía el puerto 443:

```
<VirtualHost *:443>
    DocumentRoot "/var/www/tfg/web"
    DirectoryIndex app.php
    ServerName tfg.local
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/tfg.crt
    SSLCertificateKeyFile /etc/ssl/certs/tfg.local.key
    <Directory "/var/www/tfg/web">
        AllowOverride All
        Allow from All
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost *:80>
    RewriteEngine On
    RewriteCond %{HTTPS} off
    RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
</VirtualHost>
```

1.2 PHP 7.0

Instalamos php y composer [7]:

```
$apt-get install -y php7.0 libapache2-mod-php7.0 php7.0-cli php7.0-common php7.0-mbstring  
php7.0-gd php7.0-intl php7.0-xml php7.0-mysql php7.0-mcrypt php7.0-zip composer
```

1.3 MYSQL y PHPMYADMIN

Instalamos mysql y phpmyadmin:

```
$apt-get install mysql-server phpmyadmin
```

1.4 Configuración del proyecto

Finalizados los pasos indicados vamos a crear el proyecto Symfony, habilitar el sitio, crear la BBDD y configurar los permisos adecuados.

En la siguiente línea, creamos el proyecto [4] siguiendo el estándar del propio framework. En la creación del mismo, nos solicitará los parámetros necesarios para establecer la conexión con la BBDD:

```
$composer create-project symfony/framework-standard-edition tfg  
  
Creating the "app/config/parameters.yml" file  
  
Some parameters are missing. Please provide them.  
  
database_host (127.0.0.1):  
database_port (null):  
database_name (symfony): bbddTFG  
database_user (root):  
database_password (null): password
```

Ahora activamos el sitio web y configuramos los permisos necesarios para apache:

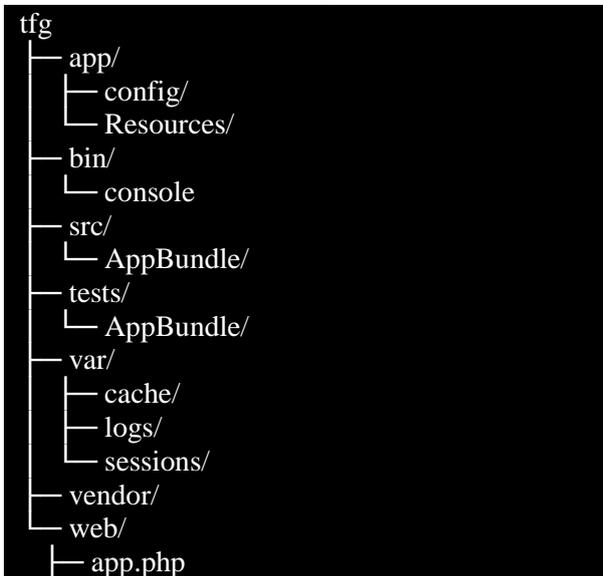
```
$a2ensite tfg.local.conf  
  
$chown www-data:www-data /var/www/tfg/  
  
$chown www-data:www-data /var/www/tfg/var/
```

Por último, siguiendo los **documentos oficiales** configuramos ACL:

```
$HTTPDUSER=$(ps axo user,comm | grep -E '[a]pache|[h]ttpd|[_]  
www|[w]ww-data|[n]ginx' | grep -v root | head -1 | cut -d\ -f1)  
$setfacl -dR -m u:"$HTTPDUSER":rwX -m u:$(whoami):rwX var  
$setfacl -R -m u:"$HTTPDUSER":rwX -m u:$(whoami):rwX var
```

2- Estructura Symfony

La estructura de directorios de Symfony varía en función al tipo de proyecto que hayamos creado, *framework-standard-edition*, *website-skeleton*,... En nuestro caso, tenemos la siguiente estructura:



Vamos a revisar los directorios y el uso recomendado de los mismos:

- **app/config:** Contiene los ficheros de configuración del proyecto (YAML), por ejemplo, los archivos routing.yml, security.yml
- **app/Resources:** En esta ruta tenemos las plantillas TWIG necesarias para la presentación de la aplicación.
- **bin:** Los binarios del proyecto se ubican aquí, por ejemplo la consola que utilizaremos, posteriormente, para las llamadas a doctrine.
- **src/AppBundle:** Este directorio contiene el código relativo a los controladores, rutas... Es decir, el código específico de la lógica de negocio de nuestra aplicación.
- **tests/AppBundle:** Contiene las aplicaciones de test.
- **var:** Dentro de esta ruta encontramos los directorios cache, logs y sessions donde la aplicación guarda los registros, sesiones y archivos caché.
- **vendor:** Aquí se encuentran las dependencias de código de nuestra aplicación.
- **web:** En el directorio web tendremos, entre otros, los directorios css y jss donde integraremos nuestras librerías bootstrap[3] y jQuery.

3- Creación de entidades

3.1 Introducción

En base al diagrama de clases definido en la fase de análisis, tenemos que crear la estructura adecuada de datos para dar soporte a nuestra aplicación. En Symfony, como en otras herramientas de programación, definiremos las entidades y sus relaciones para, posteriormente, “traducirlo” a bases de datos.

En este proceso necesitaremos utilizar el ORM Doctrine mediante el cual Symfony mapea como objetos de la aplicación las tablas que componen la base de datos.

Procedamos a crear la BBDD mediante el comando:

```
php bin/console doctrine:database:create
```

3.2 Creación de las entidades

Symfony nos permite crear las entidades de dos maneras:

Mediante la consola

```
php bin/console doctrine:generate:entity
```

A continuación, nos pedirá introducir el *Entity shortcut name*, el formato de configuración y los atributos de la entidad. Veamos el ejemplo para la entidad Persona:

```
Welcome to the Doctrine2 entity generator
```

```
The Entity shortcut name: AppBundle:Persona
```

```
Determine the format to use for the mapping information.
```

```
Configuration format (yml, xml, php, or annotation) [annotation]:
```

```
Instead of starting with a blank entity, you can add some fields now.
```

```
Note that the primary key will be added automatically (named id).
```

```
Available types: array, simple_array, json_array, object,
```

```
boolean, integer, smallint, bigint, string, text, datetime, datetimetz,
```

```
date, time, decimal, float, binary, blob, guid.
```

```
New field name (press <return> to stop adding fields): nombre
```

```
Field type [string]:
```

```
Field length [255]: 20
```

```
Is nullable [false]:
```

```
Unique [false]:
```

```
New field name (press <return> to stop adding fields): apellidos
```

```

Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]:
New field name (press <return> to stop adding fields): dni
Field type [string]:
Field length [255]: 9
Is nullable [false]:
Unique [false]: true
New field name (press <return> to stop adding fields): notas
Field type [string]:
Field length [255]:
Is nullable [false]:
Unique [false]:
New field name (press <return> to stop adding fields):

```

Entity generation

```
created ./src/AppBundle/Entity/Persona.php
```

Tras la siguiente definición tendremos la entidad, y su código php, en el fichero *src/AppBundle/Entity/Persona.php*. Utilizando este método, tenemos la ventaja de no tener que preocuparnos por los *getter* y *setter* de las propiedades, dado que el propio gestor las añade al archivo:

```

<?php

namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Persona
 *
 * @ORM\Table(name="persona")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\PersonaRepository")
 */
class Persona
{
    /**
     * @var int
     */

```



```

* @ORM\Column(name="id", type="integer")
* @ORM\Id
* @ORM\GeneratedValue(strategy="AUTO")
*/
private $id;

/**
* @var string
*
* @ORM\Column(name="nombre", type="string", length=255)
*/
private $nombre;

/**
* Get id
*
* @return integer
*/
public function getId()
{
    return $this->id;
}

/**
* Set nombre
*
* @param string $nombre
* @return Persona
*/
public function setNombre($nombre)
{
    $this->nombre = $nombre;

    return $this;
}

/**
* Get nombre
*
* @return string
*/
public function getNombre()
{
    return $this->nombre;
}

....
}

```

Creación directa de entidades

La otra opción que tenemos para generar las entidades, pasa por crear directamente nuestros archivos php con las clases necesarias. Es una opción más “peligrosa” puesto que debemos recordar implementar todos los métodos para acceder o modificar los atributos de la clase, en cambio, nos permite abordar directamente la definición de las relaciones entre clases:

- **Uno a muchos / muchos a uno:** Esta es la única relación representada en nuestro diagrama de clases. Por ejemplo, un objeto *persona* puede estar asociado a muchos objetos *registro*, sin embargo, un objeto *registro* solo puede estar asociado a un objeto *persona*.
- **Muchos a muchos:** Esta relación permite la asociación de muchos objetos X con muchos objetos Y. En nuestro caso, no tenemos dicha relación.

Vamos a proceder a generar las entidades Sede y Registro mediante la creación directa de los archivos php, que ubicaremos dentro de `src/AppBundle/Entity/`:

`src/AppBundle/Entity/Sede.php`

```
<?php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 */
class Sede
{
    /**
     * @ORM\Id;
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="string", length=20)
     */
    protected $nombre;

    /**
     * @ORM\Column(type="string", length=200)
     */
    protected $calle;

    /**
     * @ORM\Column(type="string", length=50)
     */
    protected $ciudad;

    public function getId()
    {
        return $this->id;
    }
}
```



```

public function setNombre($nombre)
{
    $this->nombre = $nombre;
}

public function getNombre()
{
    return $this->nombre;
}
public function setCalle($calle)
{
    $this->calle = $calle;
}

public function getCalle()
{
    return $this->calle;
}
public function setCiudad($ciudad)
{
    $this->ciudad = $ciudad;
}

public function getCiudad()
{
    return $this->ciudad;
}

}

```

src/AppBundle/Entity/Registro.php

```

<?php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity
 */
class Registro
{
    /**
     * @ORM\Id;
     * @ORM\Column(type="integer")
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;

    /**
     * @ORM\Column(type="date")
     */
    protected $fecha;

    /**

```



```

* @ORM\Column(type="boolean")
*/
protected $tipo;

/**
* @ORM\ManyToOne(targetEntity="Persona")
**/
protected $persona;
/**
* @ORM\ManyToOne(targetEntity="Sede")
**/
protected $sede;

public function getId()
{
    return $this->id;
}

public function setFecha($fecha)
{
    $this->fecha = $fecha;
}

public function getFecha()
{
    return $this->fecha;
}
public function setTipo($tipo)
{
    $this->tipo = $tipo;
}

public function getTipo()
{
    return $this->tipo;
}
public function setPersona($persona)
{
    $this->persona = $persona;
}

public function getPersona()
{
    return $this->Persona;
}
public function setSede($sede)
{
    $this->sede = $sede;
}

public function getSede()
{
    return $this->sede;
}
}

```



Dentro de estas dos clases, podemos observar cómo se definen las relaciones muchos-a-uno entre *Registro-Persona* y *Registro-Sede*.

3.3 Carga del esquema en BBDD

Con cualquiera de los dos métodos mencionados en el punto anterior tendremos definida la estructura de entidades deseada. Ahora debemos pasar el esquema a la BBDD, para ello primero validamos el schema:

```
$php bin/console doctrine:schema:validate  
[Mapping] OK – The mapping files are correct.  
[Database] OK – The database schema is in sync with the mapping files.
```

Y, ahora, cargamos la actualización al esquema de la BBDD:

```
$php bin/console doctrine:schema:update –force  
Updating database schema...  
Database schema updated successfully! “3” queries were excuted
```

Tras la ejecución de esta acción, si no hemos tenido ningún error, dispondremos de las entidades generadas como tablas, con sus atributos, en la BBDD. Como instalamos *phpmyadmin*, podemos conectarnos mediante la URL <https://127.0.0.1/phpmyadmin> y, tras el login, comprobar que las tablas están creadas correctamente:



Server: localhost » Database: TFG » Table: sede

Examinar Estructura SQL Buscar Insertar Exportar Más

Estructura de tabla Vista de relaciones

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra	Acción
1	id	int(11)			No	Ninguna	AUTO_INCREMENT	Camt
2	nombre	varchar(20)	utf8_unicode_ci		No	Ninguna		Camt
3	calle	varchar(200)	utf8_unicode_ci		No	Ninguna		Camt
4	ciudad	varchar(50)	utf8_unicode_ci		No	Ninguna		Camt

3.4 Entity Manager

Para manipular datos Symfony crea, automáticamente, el objeto *Entity Manager*. Para obtener este objeto dentro de un controlador, podemos realizar la siguiente instrucción:

```
class DefaultController extends Controller
{
    public function portadaAction(){
        $em=$this->getDoctrine()->getEntityManager();
    }
}
```

El *entity manager* de Doctrine nos servirá para realizar todas las operaciones que necesitemos sobre los datos de nuestra aplicación. Tenemos que tener en cuenta que, para el correcto funcionamiento de la aplicación, debemos de persistir el dato. Para ello utilizamos la función *persist()* como vamos a ver en los siguientes ejemplos de código:

Creación de nuevo elemento

En este código, generamos una nueva persona y guardamos los datos en nuestra BBDD:

```
$persona= new Persona();
$persona->setNombre('Ramon');
$persona->setApellidos('Guerrero Oriola');
$persona->setDni('00000000X');
$em=$this->getDoctrine()->getEntityManager();
$em->persist($persona);
$em->flush();
```

Si revisamos el código, vemos que tras *persist()* realizamos una llamada al método *flush()*. Mediante la invocación a *flush()*, Doctrine persiste los datos.

Modificación de un elemento

Este código es muy similar a la creación de un elemento pero, en lugar de partir de una entidad vacía, recuperamos una ya existente:

```
$em=$this->getDoctrine()->getEntityManager();
$persona=$em->getRepository('AppBundle:Persona')->
findOneBy(array('dni'=>'00X'));
$persona->setNombre('Ramon');
$em->persist($persona);
$em->flush();
```

Eliminación de un elemento

Aquí recuperamos un elemento y lo eliminamos:

```
$em=$this->getDoctrine()->getEntityManager();
$persona=$em->getRepository('AppBundle:Persona')->
findOneBy(array('dni'=>'00X'));
$em->remove($persona);
$em->flush();
```

A diferencia de los ejemplos anteriores, en este caso la eliminación de un elemento no llama al método *persist()* sino a *remove()*.

Búsqueda de elemento(s)

En los códigos anterior, modificación y borrado, se utiliza un tipo de búsqueda. No obstante, el *entity manager* dispone de más métodos: *find()*, *findAll()*, *findBy()* y *findOneBy()*. Los métodos con *By* utilizan una propiedad de la entidad para la búsqueda.

En nuestro código, necesitaremos además realizar búsquedas que implican varias entidades. Una de las principales, y motivo del desarrollo de la aplicación, es la búsqueda de los registros de una sede. En el siguiente ejemplo, buscamos todos los registros de la sede de Valencia:

```
$registros=$em->getRepository('AppBundle:Registro')->findBy(array(
'sede'=>$sede->getId(),
'ciudad'=>'Valencia'
));
```

4- Bundles

El código de una aplicación Symfony se estructura en *bundles*. Los bundles son directorios que contienen archivos de código en una estructura con forma jerarquizada. Contienen clases PHP mayoritariamente, aunque también pueden contener archivos css, javascript e imágenes.

En nuestro caso, hemos utilizado el propio *AppBundle* que se genera automáticamente al importar el esqueleto del proyecto Symfony estándar. No obstante, vamos a ver la creación y activación de un *bundle*.

4.1- Creación de un nuevo Bundle

Para crear un nuevo *bundle* utilizaremos el siguiente comando de consola:

```
php bin/console generate:bundle
```

A continuación, debemos introducir los parámetros necesarios para la generación del mismo:

```
Welcome to the Symfony bundle generator!
```

```
Are you planning on sharing this bundle across multiple applications? [no]:
```

```
yes
```

```
Bundle namespace: NuevoBundle
```

```
Bundle name [NuevoBundle]: Prueba
```

```
Target Directory [src/]:
```

```
Configuration format (annotation, yml, xml, php) [xml]:
```

```
Bundle generation
```

```
Everything is OK! Now get to work :).
```

4.2- Activación de Bundles

Los *bundles* pertenecientes a una aplicación deben registrarse para poder ser utilizados. En el caso anterior, una vez creamos el *bundle* la propia consola lo registra:

```
> Checking that the bundle is autoloaded
```

```
OK
```

```
> Enabling the bundle inside app/AppKernel.php
```

```
updated ./app/AppKernel.php
```

```
OK
```

```
> Importing the bundle's routes from the app/config/routing.yml file
```

```
updated ./app/config/routing.yml
```

```
OK
```

En cambio, podemos importar un *bundle* externo y, para que funcione necesitamos importarlo dentro del *AppKernel*. Veamos un ejemplo con el bundle *FOSUserBundle* [5] (Friends of Symfony).

FOSUserBundle

Este Bundle nos provee de todo lo necesario para el registro y login de usuarios, además del reseteo de contraseña y una página de perfil dentro de una aplicación.

Para su instalación, nos sirve con añadir el bundle con composer:

```
$ composer require friendsofsymfony/user-bundle "~2.0"
```

A continuación, con el *bundle* ya instalado necesitamos añadirlo al *AppKernel* dentro del método *registerBundles()* para su correcto funcionamiento:

```
...  
class AppKernel extends Kernel  
{  
    public function registerBundles()  
    {  
        $bundles = [  
            new FOS\UserBundle\FOSUserBundle(),  
            ...  
        ];  
    }  
}
```

Con esto tendríamos activo el Bundle, para finalizar la configuración del mismo, deberíamos crear nuestra clase *User*, cambiar la configuración de seguridad...

5- Rutas y controladores

El sistema de enrutamiento dirige las URL al controlador correspondiente para la ejecución del código conveniente. El archivo principal donde definimos el routing es `app/config/routing.yml` aunque, en caso de que el número de rutas sea muy elevado, podemos utilizar rutas dentro de los *bundles*.

Por otro lado, durante el desarrollo de la aplicación hemos necesitado añadir controladores que manejen la funcionalidad necesaria. Estos controladores los encontramos dentro del propio *Bundle* de la aplicación.

5.1- Definición de rutas y controladores

Vamos a ver un ejemplo de definición de ruta y del controlador encargado de atender la función. En primer lugar, modificamos el archivo `routing.yml` para configurar el path de entrada y el controlador donde enviar la petición:

```
portada:
  path: /portada
  defaults: { _controller: AppBundle:Portada:portada }
```

Como podemos ver en el código anterior, cuando llegue alguna petición a `/portada` llamaremos al `PortadaController` (dentro del *bundle* AppBundle). Dentro de dicho controlador se ejecutará la acción `portadaAction()`.

Veamos el código del controlador:

```
<?php

namespace AppBundle\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\Routing\Annotation\Route;

class PortadaController extends Controller
{
    public function portadaAction()
    {

        return $this->render('default/index.html.twig', [
            'base_dir' => realpath($this-
>getParameter('kernel.project_dir')).DIRECTORY_SEPARATOR,
        ]);
    }
}
```



Cuando el proyecto va tomando cuerpo y ampliamos las rutas disponibles en el mismo, tenemos la opción de comprobar desde el debugger las rutas definidas. Para ellos utilizamos el siguiente comando:

```
$php bin/console debug:router
```

```
-----  
Name          Method Scheme Host Path  
-----  
portada       ANY   ANY   ANY  /portada  
-----
```

5.2- Utilización rutas para sedes

Uno de los principales factores a tener en cuenta en el desarrollo del proyecto es la diferenciación de una sede respecto de otra. En nuestro caso, hemos optado por utilizar las rutas de Symfony y el paso de parámetros ruta-controlador con el objetivo de diferenciar los accesos.

Recordemos que, en cada sede, tenemos un equipo encargado de recibir el registro de los empleados cuando entran o abandonan la sede. Por tanto, cada equipo puede cargar una ruta diferente: /portada/valencia, portada/madrid... Veamos el código empleado para esta diferenciación.

En primer lugar, tenemos que capturar la sede desde el path, y para eso modificamos el archivo routing.yml:

```
portadaCiudad:  
  path: /portada/{ciudad}  
  defaults: { _controller: AppBundle:Portada:portadaCiudad }
```

A partir de este momento, le pasamos con el parámetro *ciudad* la sede correspondiente al controlador *Portada* y, en concreto, a la función *portadaCiudadAction()*. Ahora necesitamos, con el nombre de la ciudad rescatar el objeto sede, lo hacemos de la siguiente manera:

```
public function portadaCiudadAction(Request $request, $ciudad)  
{  
  $sem=$this->getDoctrine()->getEntityManager();  
  $sede=$this->getDoctrine()->getRepository(Sede::class)-  
>findOneBy(array('ciudad'=>$ciudad));
```

De esta manera, cuando un usuario introduce su registro tenemos la sede donde ha registrado su presencia para poder guardarla. El formulario utilizado para el guardado del registro realiza el grabado en BBDD (tras la comprobación de la existencia del usuario) con el siguiente fragmento de código:

```
...  
if ($request->isMethod('POST')) {  
    $form->handleRequest($request);  
    if($form->isValid()){  
        $formu=$form->getData();  
        $persona = new Persona;  
        $persona = $this->getDoctrine()->getRepository(Persona::class)-  
>findOneBy(array('dni'=>$formu["DNI"]));  
        if(empty($persona)){  
            echo '<script language="javascript">';  
            echo 'alert("Error en el DNI introducido")';  
            echo '</script>';  
        } else{  
            $preg1 = $this->getDoctrine()->getRepository(Registro::class)-  
>findOneBy(array('persona'=>$persona),array('id'=>'DESC'));  
            $sec=$preg1->getTipo();  
            $dt = new \DateTime("now");  
            $registro=new Registro;  
            $registro->setFecha($dt);  
            if($sec){  
                $registro->setTipo(false);  
            }else{  
                $registro->setTipo(true);  
            }  
            $registro->setSede($sede);  
            $registro->setPersona($persona);  
            $em->persist($registro);  
            $em->flush();        }  
        }  
.....
```

En el código, capturamos el último registro del usuario (si existe) para comprobar el tipo de registro entrada (true) o salida (false), y así poder grabar el nuevo registro indicando correctamente si entra o sale de nuestra sede.

5.3- Configuración adicional

Barra al final de las URL

Uno de los problemas que nos podemos encontrar viene con la barra al final de las URL. Por ejemplo, en el ejemplo anterior si buscamos la URL `/portada/` obtendremos un error:

```
No route found for "GET /portada/"
```

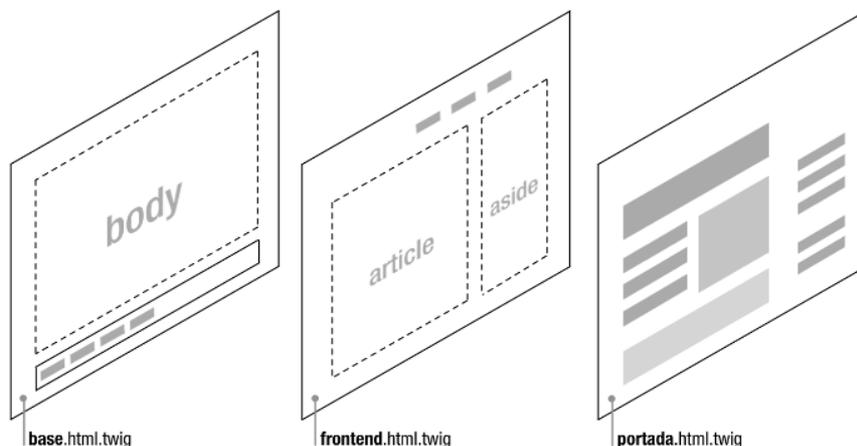
Para evitar este problema, podemos optar por la solución más sencilla, definir las rutas con la barra al final:

```
portada:  
  path: /portada/  
  defaults: { _controller: AppBundle:Portada:portada }
```

Añadiendo la barra final en nuestras rutas evitamos este error, por otro lado, cuando buscamos la URL sin barra Symfony lo enlaza con la ruta correcta (añadiendo la barra).

6- Vistas

En Symfony, para las vistas, se utilizan plantillas TWIG mediante las cuales se establecen los parámetros globales de presentación de nuestra aplicación. En la mayoría de aplicaciones web se utiliza diferentes niveles para las plantillas. Por ejemplo:



Utilizamos la herencia para ir subiendo niveles y mantener un cuerpo coherente para nuestra aplicación.

6.1- Bootstrap

Bootstrap es un conjunto de librerías que contiene plantillas de diseño con tipografía, formularios, botones, cuadros... En nuestro caso, hemos integrado dichas librerías en nuestro proyecto.

Para la integración hemos creado los directorios `css` y `js` dentro del directorio `web`. A partir de ese momento, ya podemos invocar las librerías bootstrap desde cualquiera de nuestras plantillas.

Por ejemplo, en el archivo `base.html.twig`:

```
<link rel="stylesheet" type="text/css" href="{{ asset('css/bootstrap.min.css') }}" />
<link rel="stylesheet" type="text/css" href="{{ asset('css/bootstrap-theme.min.css') }}" />
```

Tras la importación de las librerías bootstrap, nuestra portada (que las importa por herencia de `base.html.twig`) pasa de ser así:

[Acceso ADMIN](#)

CONTROL PRESENCIAL

Al siguiente aspecto:

Acceso ADMIN

CONTROL PRESENCIAL

Vemos cómo cambia el hipervínculo de acceso para administradores, el texto y el formato del textbox y el botón.

Con esto intentamos conseguir un aspecto homogéneo para toda la aplicación y con una visión moderna y agradable para el usuario.

6.2- Uso básico de las plantillas

Para la utilización de plantillas TWIG existen las siguientes estructuras básicas:

- `{% ... %}`, se utiliza para ejecutar declaraciones, como pueden ser los bucles for.
- `{{ ... }}`, se utiliza para imprimir el contenido de variables o el resultado de evaluar una expresión.
- `{# ... #}`, se utiliza para añadir comentarios en las plantillas. Estos comentarios no son incluidos en la página renderizada.

Como hemos comentado en el punto anterior, las plantillas pueden heredar de plantillas anteriores, el ejemplo más claro lo tenemos con la plantilla base:

```
{% extends 'base.html.twig' %}
```

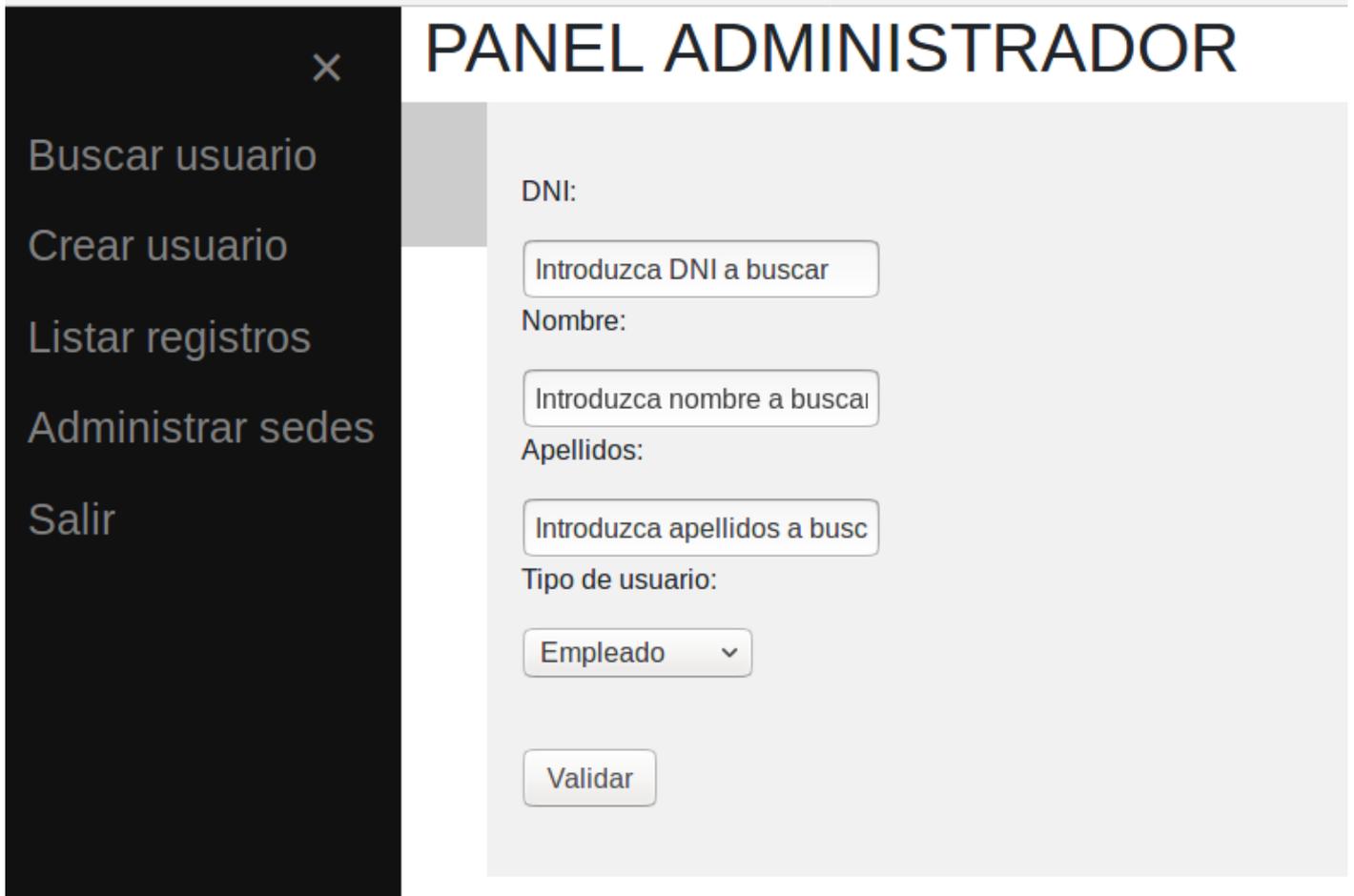
A partir de ese momento, en nuestra plantilla podemos modificar el contenido que nos interese *bloqueando* los elementos que no se deseen de la base. Por ejemplo, si queremos cambiar el estilo de la base por otro distinto, podemos utilizar el código:

```
{% block stylesheets %}
```

Y a partir de ahí definir el estilo que deseemos para esta plantilla.

6.3- Panel de administrador

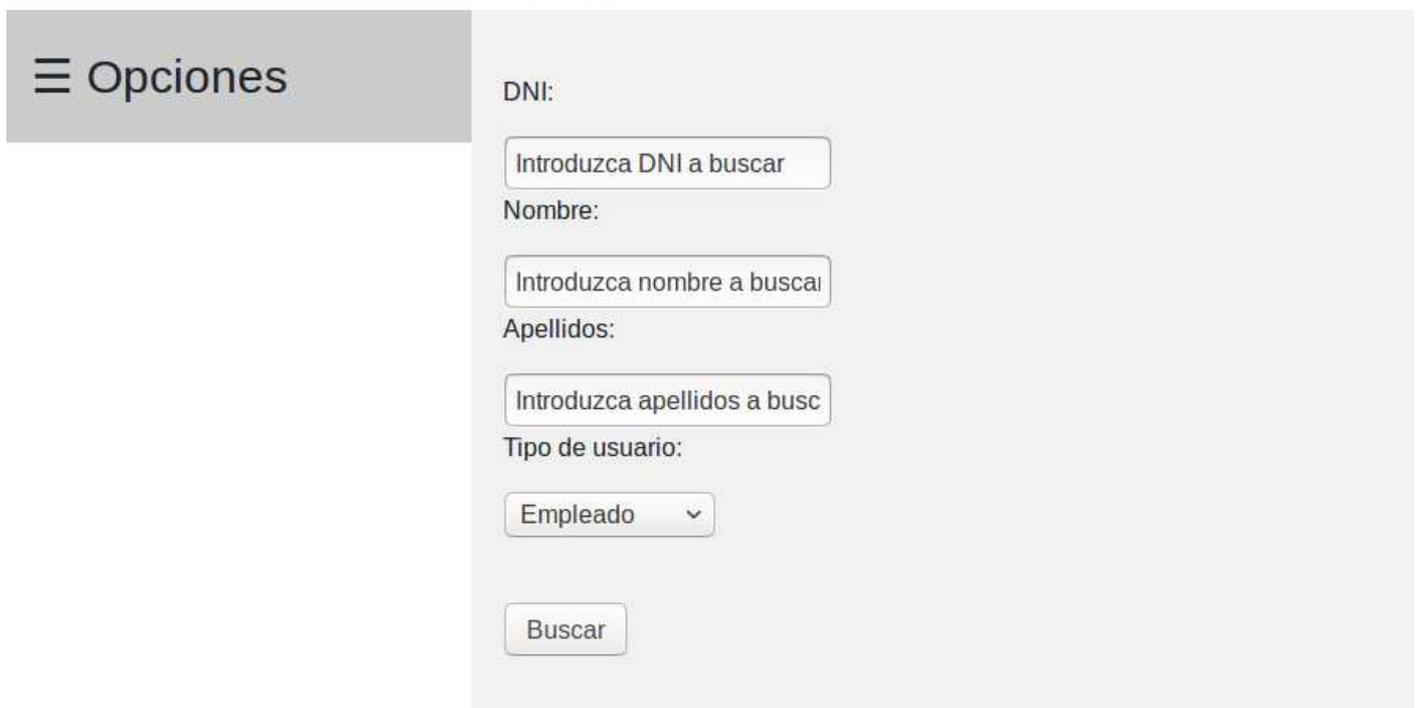
El panel de administrador, con las utilidades necesarias para el mismo, tiene la siguiente presentación:



The screenshot shows the 'PANEL ADMINISTRADOR' interface. On the left is a dark sidebar menu with a close button (X) at the top, containing the following options: 'Buscar usuario', 'Crear usuario', 'Listar registros', 'Administrar sedes', and 'Salir'. The main content area is light gray and contains search filters: 'DNI:' with a text input 'Introduzca DNI a buscar'; 'Nombre:' with a text input 'Introduzca nombre a busca'; 'Apellidos:' with a text input 'Introduzca apellidos a busc'; 'Tipo de usuario:' with a dropdown menu showing 'Empleado' and a downward arrow; and a 'Validar' button at the bottom.

Desde este panel podemos, buscar usuarios:

PANEL ADMINISTRADOR



This screenshot shows the 'PANEL ADMINISTRADOR' interface with a different sidebar menu. The sidebar is light gray and contains a hamburger menu icon followed by the text 'Opciones'. The main content area is light gray and contains search filters: 'DNI:' with a text input 'Introduzca DNI a buscar'; 'Nombre:' with a text input 'Introduzca nombre a busca'; 'Apellidos:' with a text input 'Introduzca apellidos a busc'; 'Tipo de usuario:' with a dropdown menu showing 'Empleado' and a downward arrow; and a 'Buscar' button at the bottom.

Administrar las sedes:

PANEL ADMINISTRADOR

☰ Opciones

Seleccione sede:

Sevilla ▾

Modificar Alta sede

O listar los registros filtrando por sede y fecha de los mismos:

PANEL ADMINISTRADOR

☰ Opciones

Seleccione sede y/o fecha:

Sevilla ▾ 04/07/2018 ✕ 05/07/2018 ✕

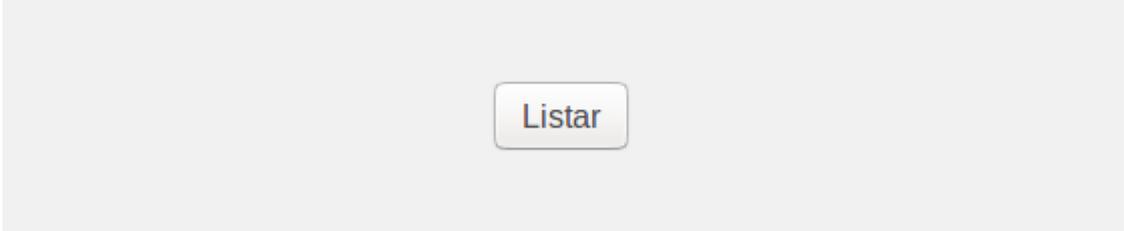
Buscar

Estas son parte de las tareas que puede llevar a cabo el administrador del sistema mediante su panel.

6.4- Panel de supervisor

En cambio, el panel del supervisor es mucho más simple, dado que sólo se le permite listar los usuarios que, actualmente, estén en el interior de su sede asociada.

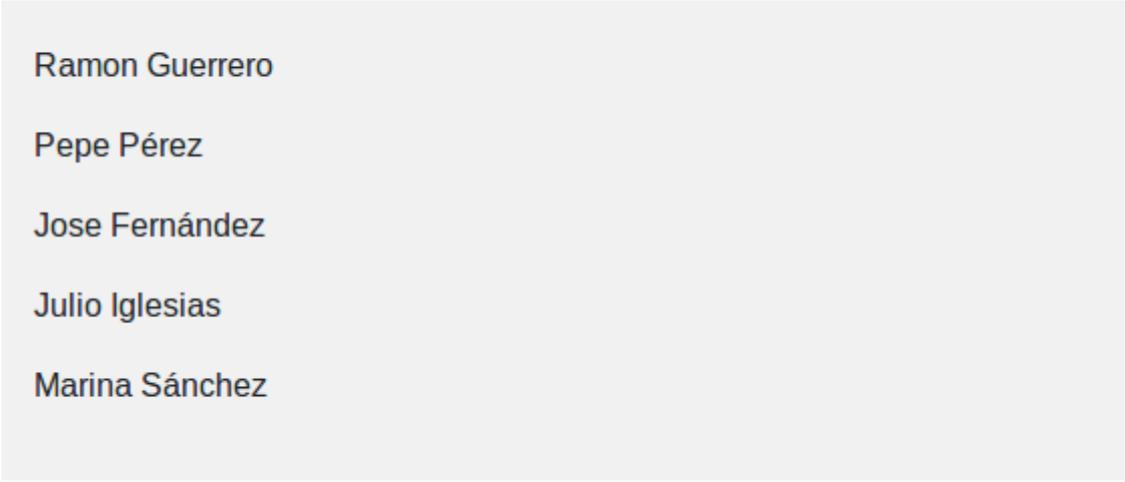
PANEL SUPERVISOR



A screenshot of a web interface for a supervisor panel. It features a light gray background with a single, centered button labeled 'Listar' in a rounded rectangular box.

Desde esta ventana tenemos la opción de *Listar* para sacar el propio listado, anteriormente comentado:

PANEL SUPERVISOR



A screenshot of a web interface for a supervisor panel. It features a light gray background with a list of five names: Ramon Guerrero, Pepe Pérez, Jose Fernández, Julio Iglesias, and Marina Sánchez, listed vertically.

VI. Pruebas

Hemos realizado pruebas y test funcionales para comprobar el funcionamiento y manejo de nuestra app. Mediante estos test se revisa que la aplicación hace correctamente lo que debe, sin errores y en tiempos de carga apropiados.

1.1- Pruebas de registro y login

En primer lugar, para comprobar el correcto funcionamiento del sistema, hemos generado un usuario administrador desde la opción register que, más tarde eliminaremos de las rutas puesto que esto se gestionará desde el panel de administrador.

Para ello accedemos a la ruta y rellenamos los campos del administrador:



127.0.0.1:8000/registro/

REGISTRO INICIAL

DNI:

Nombre:

Apellidos:

Contraseña:

Validar

A partir de este momento, ya tenemos un usuario administrador para comprobar el login de la plataforma. Para ello vamos a la portada y seleccionamos *Acceso ADMIN*:

CONTROL PRESENCIAL

Esto nos lleva al panel de login, aquí introducimos el usuario y la contraseña y validamos:

ACCESO PANEL CONTROL

Tras realizar esta prueba, estamos correctamente *logados* en plataforma y podemos gestionar las opciones del administrador.

1.2- Pruebas de carga de la aplicación

Para las pruebas de carga de la aplicación, hemos tomado medidas de tiempo en la carga de las principales páginas de la misma. Para ello hemos utilizado la herramienta de google *PageSpeed*.

El resultado de las mismas lo vemos reflejado en la siguiente tabla:

Ruta o función cargada	PageSpeed Score
Carga portada principal	85/100
Carga panel administrador	75/100
Carga panel supervisor	75/100
Carga listado supervisor	70/100
Carga listado administrador	70/100
Buscar persona	72/100

Hemos de tener en cuenta que nuestra base de datos contiene 50 registros de usuario, a medida que la aplicación funcione y esta carga sea mayor, podríamos notar peor rendimiento en la prueba “Carga listado administrador”.

Por otro lado, la principal mejora que nos sugiere *PageSpeed* para mejorar la carga de la página, es habilitar la compresión de los recursos css, js,... Esto nos implica una reducción del 81% en el peso de los ficheros implicados.

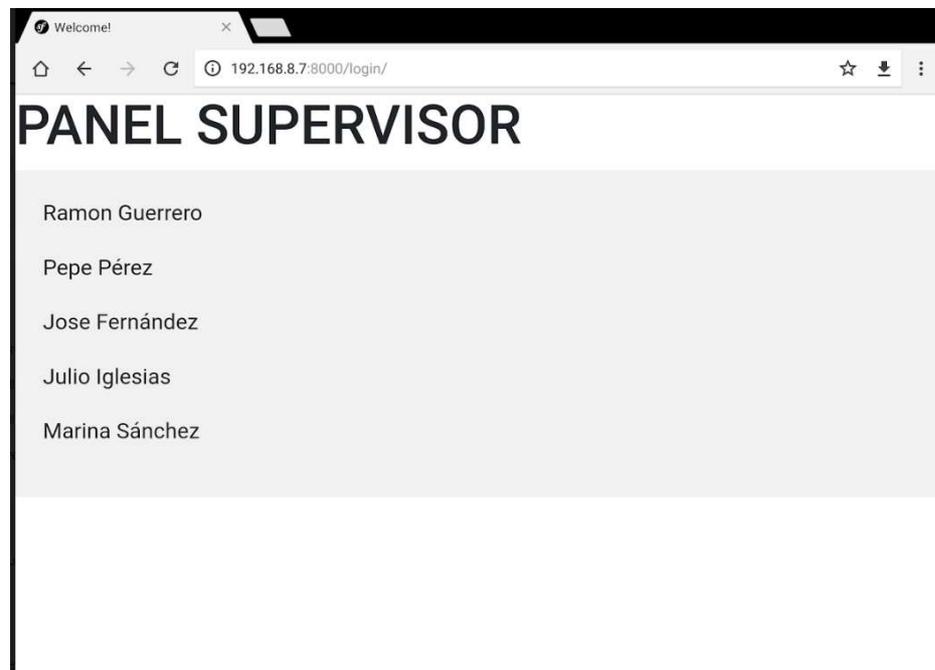
1.3- Carga en dispositivos móviles

Por último vamos a realizar una prueba con terminal móvil para comprobar la carga correcta de la aplicación en dichos terminales. Para la prueba hemos utilizado dos terminales Android:

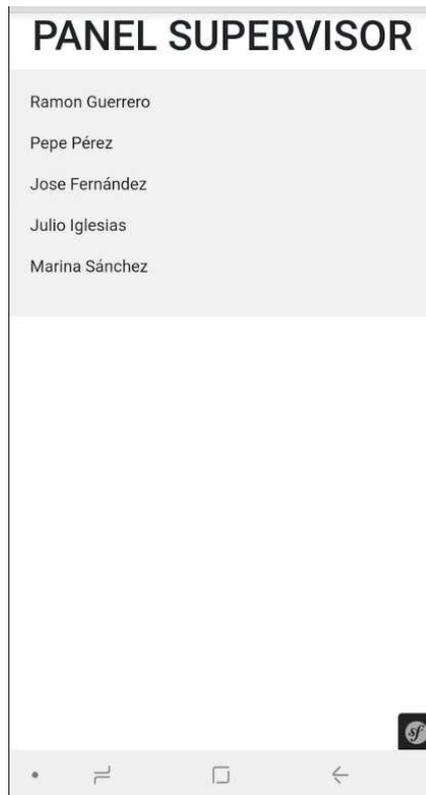
- Tablet de 9.7”: Samsung Tab A, con resolución 768x1024 pixels y navegador Google Chrome.
- Smartphone: Samsung Galaxy S7, con resolución 2560 x 1440 píxeles y navegador Google Chrome.

Las pruebas en ambos casos han resultado satisfactorias, mostrando el listado de usuarios correctamente.

Tablet:



Smartphone:



VII. Conclusiones

Por último, vamos a exponer las conclusiones acerca de lo que ha supuesto el desarrollo del proyecto.

En primer lugar, me gustaría destacar que la mayor parte del tiempo dedicado a este proyecto la he empleado en el aprendizaje de Symfony, aunque parezca obvio. Es un *framework* conocido y con una gran comunidad, pero es cierto que los primeros pasos con el mismo pueden ser aturdidores. Entender los conceptos de rutas, bundles o empezar a interactuar con archivos TWIG y YAML con los que no tenía ninguna experiencia previa, ha sido un reto para mí.

En cuanto al desarrollo de la aplicación en sí, una vez entendida la funcionalidad que aporta Doctrine y con la gran ayuda de la comunidad que utiliza *PHP* he podido llegar a desarrollar lo que pretendía. En más de una ocasión me he encontrado revisando el panel de Symfony, los logs... para intentar averiguar que estaba pasando en ese momento para que la aplicación no funcionase como deseaba.

Esta aplicación puede mejorarse integrando el control de horarios, un aspecto mucho más común en cualquier empresa que la gestión del personal presente en la misma. Como comentamos en esta memoria anteriormente, sería posible realizar la mejora integrando un calendario y dotando de mayor funcionalidad al módulo de administración.

Finalmente, quisiera comentar que me queda la sensación que Symfony, es una herramienta muy potente y con la cual se pueden desarrollar aplicaciones de mucha mayor envergadura.

VIII. Bibliografía

Vamos a hacer referencia a la documentación consultada para el desarrollo del proyecto:

[1] Libro Desarrollo web ágil con Symfony2 de Javier Eguiluz.

[2] Especificación de requisitos IEEE Std. 830-1998

<https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>

[3] Utilización de Bootstrap

<https://www.uno-de-piera.com/formularios-symfony-3-introduccion/>

[4] Configuración de symfony

<https://symfony.com/doc/3.4/setup.html>

[5] Uso de FOSBundle

<https://www.twilio.com/blog/2017/08/up-and-running-with-symfony-3.html>

[6] Configuración de apache

<https://codereviewvideos.com/course/symfony-deployment/video/apache-nginx-symfony-permissions>

[7] Instalación de composer

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-14-04>