Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Aplicación Web de bases de datos usando el Framework Symfony

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Gestión

Autor: Ángel Talavera Molero

Director: José Vicente Busquets Mataix

22/09/2014

Resumen

Implementación de un sistema de seguimiento detallado de PFCs para alumnos y profesores basándose en el conocido framework Symfony2.

El sistema permitirá a los usuarios hacer login en la aplicación. En función del rol de cada usuario dispondrá de acceso a distintas funcionalidades.

Los usuarios de tipo profesor podrán dar de alta PFCs, así como gestionar la información de éstos.

Los alumnos por su parte podrán inscribirse en proyectos y estar en comunicación con otros alumnos y con el profesor encargado de su proyecto mediante un sistema de mensajería interno. Este sistema también podrá ser utilizado por los profesores.

Además se permite la opción de crear PDFs sobre ciertos apartados de la web, como las fichas de proyecto o ficha de alumno.

Palabras clave: Symfony, gestión de PFC, Aplicación web, Doctrine, ORM, Twig

Tabla de contenidos

I.-INTRODUCCIÓN

1.-Introducción

El principal objetivo de este proyecto es la implementación de un sistema de seguimiento detallado de PFCs para alumnos y profesores basándose en el conocido framework Symfony2.

Symfony es un framework PHP basado en la arquitectura MVC (Modelo-Vista-Controlador).

Aunque Symfony también puede ser utilizado para otros tipos de desarrollos no orientados a la Web, fue diseñado para optimizar el desarrollo de aplicaciones Web, proporcionando herramientas para agilizar aplicaciones complejas y guiando al desarrollador a acostumbrarse al orden y buenas prácticas dentro del proyecto.

El concepto de Symfony es no reinventar la rueda, por lo que reutiliza conceptos y desarrollos exitosos de terceros y los integra como librerías para ser utilizados por nosotros.

El uso de Symfony2 implica de forma inherente usar una serie de aplicaciones que el framework necesita para funcionar:

- Un servidor web.

- PHP.

- Un sistema de Gestión de Bases de Datos.

- Doctrine.

- Un entorno de desarrollo o editor de textos.

Para la elaboración de la memoria se ha utilizador el siguiente software:

-Procesador de textos. Microsoft Word / OpenOffice

-Para generar diagramas UML hemos utilizado la web interactiva http://www.lucidchart.com, que permite generar diagramas online y descargarlos.

2.-Motivación y objetivos

Permitir al profesorado tener un buen control sobre los proyectos que dirigen, así como el estado en el que se encuentran y una comunicación directa con sus alumnos a través de la aplicación.

Ayudar a los alumnos a mejorar la gestión del proyecto, así como facilitar una vía de comunicación con el profesor.

3.-Entorno de desarrollo

Este proyecto se ha desarrollado sobre Windows 7 y puede ser utilizado desde cualquier navegador web, ya que cumple los estándares bajo los que éstos trabajan.

La aplicación utiliza cookies para mantener activa la sesión de los usuarios, éstas no son permanentes y pueden borrarse manualmente desde las preferencias del navegador.

II.-ESPECIFICACIÓN DE REQUISITOS

1.-Introducción

La siguiente especificación de requisitos está redactada siguiendo el patrón del estándar IEEE-STD-830-1998.

1.1.-Propósito

El propósito de este apartado es definir con exactitud la funcionalidad de la intranet que se va a desarrollar. Esto incluye objetos de los que se compone así como la relación entre éstos y sus restricciones. También se muestra la repercusión en base de datos de las distintas funcionalidades, así como las restricciones, secuencialidades y comportamientos bajo los que debe trabajar este software.

La información que aquí se muestra está dirigida a usuarios con una mínima base en conocimientos respecto a la programación orientada a objetos, así como algunos conceptos en programación web utilizando los lenguajes PHP y javascript. No obstante las secciones donde se explican las funcionalidades están redactadas para que puedan ser entendidas sin necesidad de conocimientos informáticos por parte de los usuarios.

1.2.-Alcance

Se pretende desarrollar una aplicación llamada MiPFC utilizando el paradigma de la programación orientada a objetos, con el lenguaje PHP e implementada bajo el framework Symfony.

El software realizará la función de soporte para la gestión de PFCs, relacionando a profesores y alumnos, y el proyecto que éstos tengan en común.

Los usuarios de tipo alumno se darán de alta en el sistema y elegirán un PFC para inscribirse. Una vez el profesor acepte su inscripción, el alumno podrá solicitar la modificación del estado de los distintos hitos de los que se componga un proyecto. Además el sistema proporcionará una funcionalidad para enviar notificaciones, tanto por parte del profesor como el alumno. Por su parte el profesor podrá dar de alta PFCs y asignarlos a aquellos alumnos que él desee y que además hayan solicitado la inscripción. Además se podrán generar documentos en formato PDF de algunas secciones.

El objetivo de dicha aplicación es mejorar la gestión de la asignatura del PFC, así como intentar fomentar la constancia en el trabajo por parte de los alumnos, y ayudar al profesorado a organizarse.

1.3.-Definiciones, siglas y abreviaciones

PHP: Hypertext Pre-processor, es un lenguaje de lado de servidor, es decir que su interpretación se realiza por el servidor. Es el lenguaje principal en el que estará implementada la aplicación.

Framework: es una estructura conceptual y tecnológica de soporte definido que nos servirá de base para la organización y desarrollo del proyecto. En otras palabras, sería como un esqueleto al cual debemos "ceñirnos" a la hora de desarrollar.

Symfony: es el framework bajo el cual se va a desarrollar la aplicación.

Aplicación web: son aplicaciones que se ejecutan en un navegador, por ejemplo

Firefox, Chrome, etc. Es lo que antiguamente se denominaba página web. El uso de este término quiere decir que dichas web realizan funcionalidades e interactúan con sistemas de bases de datos.

BD: base de datos.

ORM: Object-relational mapping.

PFC: Proyecto final de carrera.

Javascript: lenguaje de programación soportado por los navegadores web. También suele conocerse con el nombre de ECMASCRIPT. La versión en el momento de escribir este texto es la 6.

UML: Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language) es un lenguaje de modelado de sistemas de software.

1.4.-Referencias

-IEEE-STD-830-1998
-Wikipedia
-www.symfony.es

1.5.-Visión global

Esta ERS seguirá la siguiente estructura: a continuación se muestra una descripción global del producto, tras la cual se especificarán en profundidad todas las funcionalidades de la aplicación. Para ello utilizaremos una estructura definida por los distintos objetos que representan el producto a desarrollar, así como las distintas funcionalidades que afectan a éstos.

2.-Descripción general

La aplicación deberá permitir que los usuarios de tipo profesor den de alta tanto alumnos como proyectos. Además podrán generar PDFs con la información de éstos. Este tipo de usuario tendrá la capacidad de asignar y crear hitos para esos proyectos, así como modificar su estado.

Los usuarios de tipo alumno tendrán la capacidad de solicitar la asignación de un proyecto. También podrán intercambiar mensajes con los profesores así como otros alumnos utilizando un sistema de mensajería interno. Además se les permitirá solicitar el cambio del estado de un hito. También podrán generar PDFs de ciertos apartados de la web.

3.-Requisitos específicos

3.1.-Interfaces externas

3.2.-Requisitos funcionales

3.2.1-Clase Usuario

Esta entidad será la genérica para los atributos comunes a las subentidades Alumno y profesor. Se compondrá de:

Nombre	Tipo	Restricciones
id	int	Deber ser único
nombre	string	No deberá contener números ni signos de puntuación.
apellido1	string	No puede ser null.
apellido2	string	
nie	string	Debe cumplir con el formato establecido para los documentos de identidad.
pass	string	La primera vez que se registra un usuario su valor será el nie de éste.
fechaNacimiento	datetime	No puede ser null.
direccionCorreoElectronico	string	Debe cumplir el formato de direcciones de correo electrónico.
fechaAltaSistema	datetime	No puede ser null.
fechaBajaSistema	datetime	
mailBoxEnviados	MailBoxEnviados	No puede ser null.
mailBox	MailBox	No puede ser null.

Las funcionalidades se tratarán en el apartado correspondiente de cada entidad, para así poder definir mejor la secuencialidad particular de cada acción.

3.2.2.-Clase Alumno

Entidad que refleja los usuarios de tipo alumno que se den de alta en el sistema. Heredará las características de la entidad Usuario, además los atributos específicos de los que se compone son:

Nombre	Tipo	Restricciones
id	Int	Debe ser único
titulacion	Titulacion	No puede ser null.

3.2.2.1.-Crear alumno

Esta función consiste en dar de alta alumnos en el sistema. Esta funcionalidad sólo podrá ser dada de alta por usuarios de tipo Profesor. Para ello se completará el formulario correspondiente. Su proceso sería:

Acción	Repercusión en BD
1.-Acceder al formulario.	
2.-Rellenar campos y procesar formulario.	Correcto: se inserta en la tabla asociada a los alumnos la nueva tupla. Error: en caso de que algún dato no cumpla con las restricciones definidas para éste, la aplicación avisará del error, cancelará el procesamiento de la información del formulario así como la navegación.
3.-La aplicación redirige a la ficha de usuario.	

3.2.2.2.-Eliminar alumno

Funcionalidad que representa la baja del sistema de un usuario de tipo alumno. Sólo los usuarios con el rol profesor podrán llevar a cabo esta acción. Secuencia:

Acción	Repercusión en BD
1.-Acceder a la ficha del alumno.	
2.-Pinchar el enlace habilitado para la baja de dicho alumno.	Correcto: el sistema eliminará el alumno, así como su Mailbox y los mensajes asociados a éste. Error: la aplicación avisa de que se ha producido un error, y que no habrá podido realizar la acción.
3.-La aplicación redirige al dashboard de profesor.	

3.2.2.3.-Editar alumno

Funcionalidad que represeAnta la edición de los datos de un usuario de tipo alumnos. Los campos que se podrán editar son:

- nombre
- apellido1
- apellido2
- direccionCorreoElectronico
- pass
- nie
- titulacion

3.2.2.4.-Enviar mensaje

Desde la sección de su mailbox el usuario podrá enviar mensajes a través de la aplicación a otros mailbox. El envío de mensajes permitido para los usuarios de tipo

Alumno será al usuario de tipo profesor encargado de su proyecto y a los otros alumnos con lo que forme equipo. La funcionalidad es la misma para la recepción de mensajes. La secuencia en detalle de dicha acción sería:

Acción	Repercusión en BD
1.-Acceder a la sección mensajes.	
2.-Hacer click en el enlace habilitado para enviar nuevo mensaje.	
3.-Se mostrarán los campos a rellenar para el envío del mensaje. (Ver descripción de la entidad mensaje)	
4.-Completar el formulario, elegir destinatario/s y hacer click en enviar. NOTA: los usuarios destinatarios se elegirán de un select, el cual contendrá los destinatarios disponibles.	Correcto: en caso de que todos los campos cumplan con sus restricciones se creará una nueva instancia del mensaje en la tabla correspondiente, se volverá a la sección mailbox y se enviará dicho mensaje. Error: la aplicación informará al usuario de que no se ha podido enviar el mensaje, y se mantendrá en el estado que estuviera.

3.2.2.5.-Eliminar mensajes

Desde el mailbox se podrán eliminar aquellos mensajes que desee el usuario. La secuencia de acciones sería:

Acción	Repercusión en BD
1.-Acceder a la sección mensajes.	
2.-Hacer click en el enlace habilitado para eliminar los mensajes seleccionados.	Correcto: en caso de que todo funcione bien, se eliminarán los registros de los mensajes seleccionados. Error: la aplicación avisará de que se ha producido un error y no eliminará ningún mensaje.
3.-La aplicación actualizará el listado de mensajes.	

3.2.2.6.-Inscribirse en proyecto

Una vez dados de alta, los alumnos pueden consultar el listado completo de todos los proyectos, elegir uno y enviar su solicitud para entrar a formar parte de éste. Esto le enviará una notificación al profesor, el cual podrá aceptar o rechazar dicha solicitud. Concretamente el flujo de acciones será:

Acción	Repercusión en BD
1.-Acceder a la ficha del proyecto.	
2.-Hacer click en el enlace habilitado para solicitar la unión.	
3.-Esto enviará una notificación al usuario de tipo profesor.	
4.-El profesor procesará la solicitud.	En caso de que el profesor acepte la solicitud, se asociará el proyecto con el alumno/s.
5.-Tanto si la respuesta es afirmativa como negativa, el sistema enviará una notificación al usuario alumno indicándole si ha sido aceptado o no.	
6.-En caso de ser aceptado el alumno podrá realizar las acciones correspondientes a formar parte de un proyecto. Ver la ficha, enviar mensajes al profesor, etc.	

3.2.2.7.-Generar PDFs

Podrá generar documentos PDF de los siguientes apartados:

- Ficha de proyecto.

3.2.2.8.-Solicitar cambio de estado de un hito asignado a un proyecto

El usuario de tipo alumno podrá solicitar el cambio de estado de cada uno de los diferentes hitos de los que conste un proyecto. Tras solicitar el cambio, el profesor deberá validar dicho cambio. El proceso será el siguiente:

Acción	Repercusión en BD
1.- Acceder a la ficha del proyecto.	
2.- Pinchar en el enlace habilitado para la solicitud del cambio.	En la tabla que refleje el estado de los hitos de los proyectos se crea una instancia nueva con los datos correspondientes.
3.- El sistema mostrará el hito por el cual se ha solicitado el cambio como pendiente.	En el momento que el profesor acepte/rechace el cambio: inmediatamente se cambiará el estado del proyecto según la decisión del profesor.
4.- El sistema enviará una notificación al profesor haciéndole saber que un alumno ha solicitado un cambio para un proyecto. Este aviso indicará de que proyecto se trata y de que alumno/s.	

3.2.3.-Clase Profesor

Entidad que refleja los usuarios de tipo profesor que se den de alta en el sistema. Los atributos de los que se compone son:

Nombre	Tipo	Restricciones
departamento	Departamento	No puede ser null.
id	Int	Debe ser único

3.2.3.1.-Crear profesor

Esta función consiste en dar de alta profesores en el sistema. Para ello se completará el formulario correspondiente. Su proceso sería:

Acción	Repercusión en BD
1.-Acceder a la aplicación.	
2.-Completar el formulario de registro asociado a profesores.	Correcto: se inserta en la tabla asociada a los profesores la nueva tupla. Error: en caso de que algún dato no cumpla con las restricciones definidas para éste, la aplicación avisará del error, cancelará el procesamiento de la información del formulario así como la navegación.
3.-La aplicación redirige a la ficha de usuario.	

3.2.3.2.-Eliminar profesor

Funcionalidad que representa la baja del sistema de un usuario de tipo profesor. Sólo los usuarios con el rol profesor podrán llevar a cabo esta acción. Secuencia:

Acción	Repercusión en BD
1.-Acceder a la ficha del profesor.	
2.-Pinchar el enlace habilitado para la baja.	Correcto: el sistema eliminará al profesor, así como su Mailbox y los mensajes asociados a éste. Además de los proyectos dados de alta por él. Error: la aplicación avisa de que se ha producido un error, y que no habrá podido realizar la acción. Este hecho no provocará navegación.
3.-La aplicación redirige a la pantalla de login.	

3.2.3.3.-Editar profesor

Funcionalidad que representa la edición de los datos de un profesor. Los

campos que se podrán editar son:

- nombre
- apellido1
- apellido2
- direccionCorreoElectronico
- departamento

3.2.3.4.-Enviar mensaje

Desde la sección de su mailbox el usuario podrá enviar mensajes a través de la aplicación a otros mailbox. El envío de mensajes permitido para los usuarios de tipo Profesor será al usuario/s de tipo alumno registrados en un determinado proyecto/s. La funcionalidad es la misma para la recepción de mensajes. La secuencia en detalle de dicha acción sería:

Acción	Repercusión en BD
1.-Acceder a la sección mailbox.	
2.-Hacer click en el enlace habilitado para enviar nuevo mensaje.	
3.-Se mostrarán los campos a rellenar para el envío del mensaje. (Ver descripción de la entidad mensaje)	
4.-Completar el formulario, elegir destinatario/s y hacer click en enviar. NOTA: los usuarios destinatarios se elegirán de un select, el cual contendrá los destinatarios disponibles.	Correcto: en caso de que todos los campos cumplan con sus restricciones se creará una nueva instancia del mensaje en la tabla correspondiente, se volverá a la sección mailbox y se enviará dicho mensaje. Error: la aplicación informará al usuario de que no se ha podido enviar el mensaje, y se mantendrá en el estado que estuviera.

3.2.3.5.-Eliminar mensajes

Desde el mailbox se podrán eliminar aquellos mensajes que desee el usuario. La secuencia de acciones sería:

Acción	Repercusión en BD
1.-Acceder a la sección mailbox.	
2.-Hacer click en el enlace habilitado para eliminar los mensajes seleccionados.	Correcto: en caso de que todo funcione bien, se eliminarán los registros de los mensajes seleccionados. Error: la aplicación avisará de que se ha producido un error y no eliminará ningún mensaje.
3.-La aplicación navegará al listado de mensajes.	

3.2.3.6.-Crear proyecto

Funcionalidad disponible desde el dashboard del profesor. Al acceder a ella se mostrará el formulario a rellenar con los datos del proyecto. Las restricciones que debe cumplir cada atributo se muestran más adelante, en la sección correspondiente al proyecto. Su secuencialidad será:

Acción	Repercusión en BD
1.-Acceder a la sección.	
2.-Rellenar el formulario y procesarlo.	Correcto: en caso de que todos los campos cumplan con sus restricciones se creará una nueva instancia del proyecto asociada al profesor. Error: la aplicación informará al usuario de que no se ha podido crear el proyecto, y se mantendrá en el estado que estuviera.
3.-La aplicación navegará al listado de proyectos.	

3.2.3.7.-Editar proyecto

Funcionalidad disponible desde la ficha del proyecto, y desde el listado de éstos. Al acceder se mostrará el formulario rellenado con los datos del proyecto. Los campos con la posibilidad de ser editados son:

- Título.
- Alumnos inscritos.
- Fecha de inicio.
- Fecha de presentación.
- Hitos.
- Descripción.
- Información adicional.

Posteriormente se muestran todos los campos y restricciones de éstos de los que se compondrá el objeto Proyecto, aún así, decir que los atributos modificados deberán seguir cumpliendo con las restricciones asociadas a cada uno de éstos. Su secuencialidad será:

Acción	Repercusión en BD
1.-Acceder a la ficha del proyecto, ya sea desde el listado de proyectos del profesor o desde la ficha individual de cada proyecto.	
2.-Editar los campos.	Correcto: en caso de que todos los campos cumplan con sus restricciones se creará una nueva instancia del proyecto asociada al profesor. Error: la aplicación informará al usuario de que no se ha podido crear el proyecto, y se mantendrá en el estado que estuviera.
3.-La aplicación navegará al listado de proyectos. Automáticamente se enviará una notificación al mailbox de los usuarios de tipo alumno que estuvieran asociados a dicho proyecto, notificándoles que se han cambiado los detalles del proyecto.	

3.2.3.8.-Eliminar proyecto

Funcionalidad disponible desde la ficha del proyecto. Al acceder se mostrará el formulario con los datos del proyecto. La eliminación de un proyecto sólo podrá llevarse a cabo si se cumple alguna de estas dos condiciones: que no existan alumnos asociados al proyecto, o que todas las fases del proyecto hayan sido superadas. Cuando decimos eliminación nos referimos a que el estado del proyecto cambia a “cerrado”, esto quiere decir que ese proyecto seguirá apareciendo en la lista de proyectos del profesor, pero no se podrá editar ninguno de sus campos, es decir que sólo estará disponible para consultarlo. Su secuencialidad será:

Acción	Repercusión en BD
1.-Acceder a la ficha del proyecto.	
2.-Hacer click en el enlace habilitado para la eliminación.	Correcto: en caso de que se cumplan las condiciones, el sistema cambiará el estado del proyecto a cerrado. Este cambio es permanente Error: la aplicación informará al usuario de que no se ha podido eliminar el proyecto. En caso de que no se pueda llevar a cabo porque no se cumpla alguna de las restricciones, el sistema informará de ello, en otro caso simplemente se dirá que ha ocurrido un error.
3.-La aplicación navegará al listado de proyectos.	

3.2.3.9.-Cambiar estado de un hito asignado a un proyecto

El usuario de tipo alumno podrá solicitar el cambio de estado de cada uno de los diferentes hitos de los que conste un proyecto. Tras solicitar el cambio, el profesor deberá validar dicho cambio. El proceso será el siguiente:

Acción	Repercusión en BD
1.- Acceder a la ficha del proyecto.	
2.- Pinchar en el enlace habilitado para aceptar/rechazar el cambio.	En la tabla que refleje el estado de los hitos de los proyectos se actualiza la instancia correspondiente a dicho proyecto y dicho hito con los datos correspondiente.
3.- El sistema mostrará el hito con el nuevo estado.	En el momento que el profesor acepte/rechace el cambio: inmediatamente se enviará una notificación al usuario informándole del cambio.

3.2.3.10.-Aceptar inscripción de alumnos en proyecto

Cuando los alumnos deciden enviar la solicitud para inscribirse en la realización de un proyecto, el usuario de tipo profesor recibe un mensaje en su Mailbox avisándole de que un alumno ha realizado una solicitud. Dicho mensaje contendrá los datos del alumno y del proyecto. A partir de este momento el profesor puede decidir aceptar o rechazar dicha inscripción. El proceso sería:

Acción	Repercusión en BD
1.- Entrar en la ficha del proyecto.	
2.- En la sección inscripciones aparecerá el listado de los alumnos que han solicitado dicha inscripción.	
3.- Aceptar o rechazar los alumnos deseados. Este proceso se podrá llevar a cabo seleccionando múltiples usuarios.	La tabla de inscripciones se actualiza cambiando el estado de la inscripción por el elegido por el profesor. Estos estados son: aceptado o rechazado.
4.- El sistema mostrará un mensaje en pantalla anunciando que el proceso de selección ha sido completado.	
5.- El sistema enviará un mensaje a los alumnos que estuvieran en la lista de inscritos, indicándoles el estado de su inscripción.	

3.2.3.11.-Generar PDFs

Podrá generar documentos PDF de los siguientes apartados:

- Listado de proyectos.
- Ficha de proyecto.
- Ficha de alumno/s asociados a un proyecto.

3.2.4.-Clase Proyecto

Entidad que representará los proyectos de los cuales pueden formar parte los alumnos. Los atributos de los que se compone, así como sus restricciones se muestran en la siguiente tabla:

Nombre	Tipo	Restricciones
id	int	Deber ser único
titulo	string	No puede ser null.
fechaInicio	datetime	
fechaPresentacion	datetime	
fechaAltaSistema	datetime	No puede ser null.
profesor	Profesor	No puede ser null.
descripcion	string	No puede ser null.
informacionAdicional	string	
estado	boolean	TRUE indica abierto, y FALSE cerrado.

3.2.5.-Clase Hito

Entidad que representará los hitos de los cuales están compuestos los proyectos. Los atributos de los que se compone, así como sus restricciones se muestran en la siguiente tabla:

Nombre	Tipo	Restricciones
id	int	Deber ser único
nombre	string	No puede ser null.
descripcion	string	
fechaCreacion	datetime	No puede ser null.

3.2.6.-Clase Departamento

Entidad que representará los departamentos al cual puede pertenecer el profesor. Los atributos de los que se compone, así como sus restricciones se muestran en la siguiente tabla:

Nombre	Tipo	Restricciones
id	int	Deber ser único
nombre	string	No puede ser null.
descripción	string	
fechaCreacion	datetime	No puede ser null.

3.2.7.-Clase Titulación

Esta entidad representará la titulación a la cual puede pertenecer el alumno. Los atributos de los que se compone, así como sus restricciones se muestran en la siguiente tabla:

Nombre	Tipo	Restricciones
id	int	Deber ser único
nombre	string	No puede ser null.
descripción	string	
fechaCreacion	datetime	No puede ser null.

3.2.8.-Clase MailBox

Esta entidad representará la relación entre los mensajes recibidos por el usuario con respecto al emisor de dichos mensajes. Además se indica el estado del mensaje:

Nombre	Tipo	Restricciones
id	int	Debe ser único
mensaje	Mensaje	No puede ser null
estado	boolean	True: Mensaje leído por el receptor False: Mensaje no leído
emisor	Usuario	No puede ser null

3.2.9.-Clase Mensaje

Entidad que representará los mensajes que se envían o reciben en los Mailbox. Los atributos de los que se compone, así como sus restricciones se muestran en la siguiente tabla:

Nombre	Tipo	Restricciones
id	int	Deber ser único
asunto	string	No puede ser null.
fechaCreacion	datetime	
texto	string	No puede ser null.

3.2.10.-Clase AlumnoProyecto

Entidad que representará la relación entre alumnos y proyectos. A continuación se muestran los atributos que definen dicha entidad, así como sus restricciones:

Nombre	Tipo	Restricciones
id	int	Deber ser único
Proyecto	Proyecto	No puede ser null.
Alumno	Alumno	No puede ser null.

3.2.11.-Clase MailBoxEnviados

Esta entidad representará la relación entre los mensajes enviados por el usuario con respecto al receptor de dichos mensajes. Además se indica el estado del mensaje:

Nombre	Tipo	Restricciones
id	int	Debe ser único
mensaje	Mensaje	No puede ser null
usuario	Usuario	No puede ser null

3.2.12.-Clase ProjectHito

Entidad para establecer la relación entre los distintos hitos que se quieran asignar a un proyecto. Además se incluyen los datos que representan el estado y la fecha de finalización del plazo de presentación de cada hito:

Nombre	Tipo	Restricciones
id	int	Deber ser único
proyecto	Proyecto	No puede ser null.
Estado	Boolean	True: indica que el hito está superado False: el hito no está superado
fechaPresentacion	datetime	

3.3.-Rendimiento

El rendimiento de la aplicación depende de dos factores. El servidor donde esté desplegada la aplicación y del rendimiento que promedie el equipo del usuario.

Respecto al servidor no requiere grandes esfuerzos para servir las peticiones que haga el usuario desde la aplicación, ya que la arquitectura MVC bajo la que se mueve el proyecto gestiona los recursos de una manera muy óptima.

Por su parte los usuarios experimentarán un mejor rendimiento cuanto mejor sea su máquina, esto es, cantidad adecuada de memoria RAM, cantidad de cookies

presentes en el navegador, etc.

No obstante la aplicación ha sido probada en equipos con 2GB de RAM, navegadores Firefox y Chrome (últimas versiones) y procesador Pentium i5 y el rendimiento ha sido el óptimo.

Por otro lado cabe destacar que para navegadores Internet Explorer inferiores a la versión 10, la aplicación presenta lentitud en la renderización de algunos elementos. Esto es debido a las limitadas características del navegador, ya que no soporta algunas características de Bootstrap.

3.4.3.-Portabilidad

La portabilidad de la aplicación es total. Al tratarse de una aplicación web puede ejecutarse en cualquier PC que cumpla con los prerrequisitos básicos que se han citado en apartados anteriores.

Al estar alojada en un servidor con su propia BBDD los datos se mantienen, independientemente del lugar donde se acceda a la aplicación.

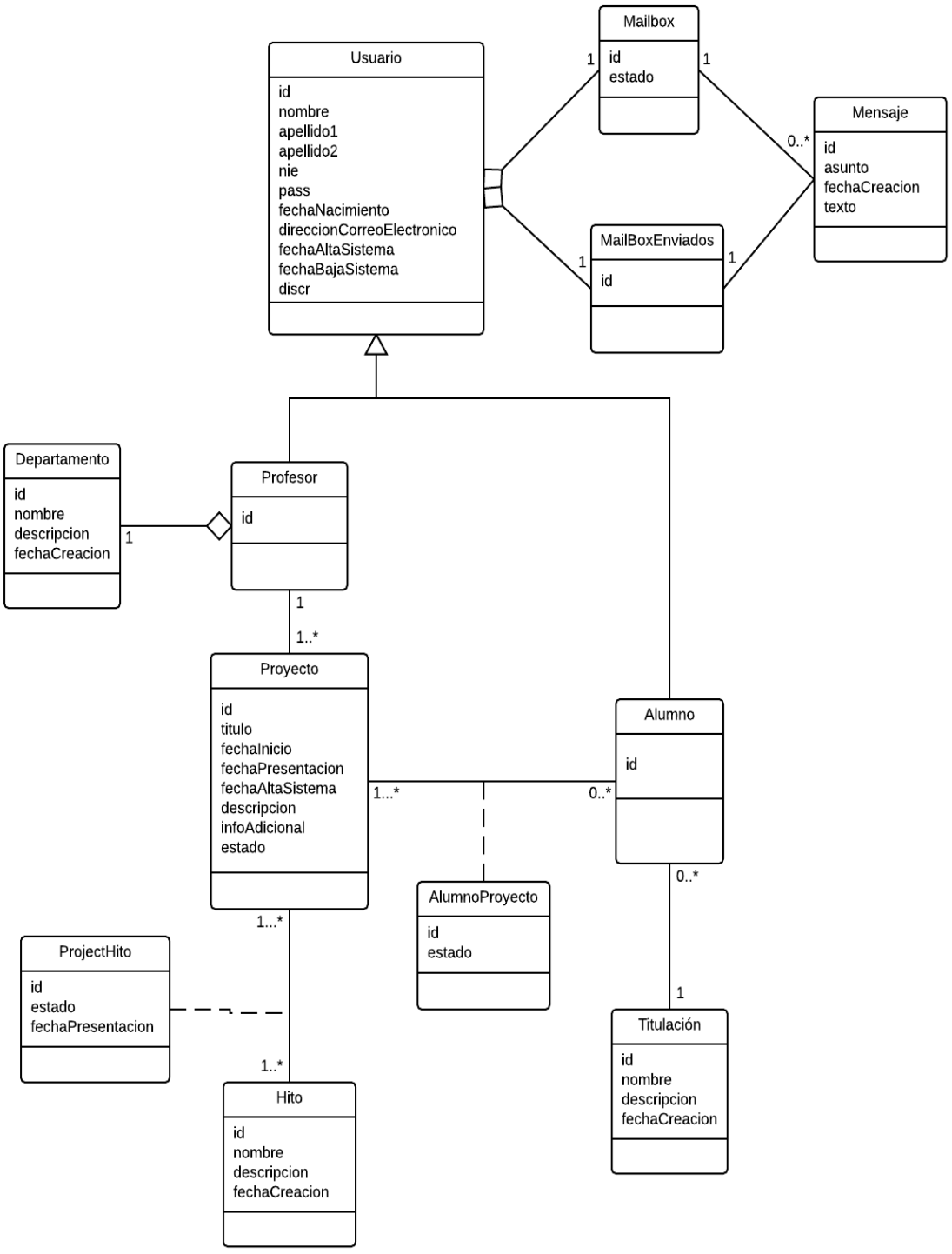
Con respecto al uso desde dispositivos móviles, está totalmente soportado, ya que la aplicación ha sido programada siguiendo el paradigma del responsive design. Éste se rige por la idea de que los elementos de la web se adapten a la resolución del dispositivo desde el que es accedida, es decir que se redimensionen según el tamaño del dispositivo.

III.-Análisis

1.-Introducción

En esta sección se muestra el análisis de la aplicación. Se ha llevado a cabo utilizando diagramas que representan de que manera funciona la aplicación, así como la estructura de ésta.

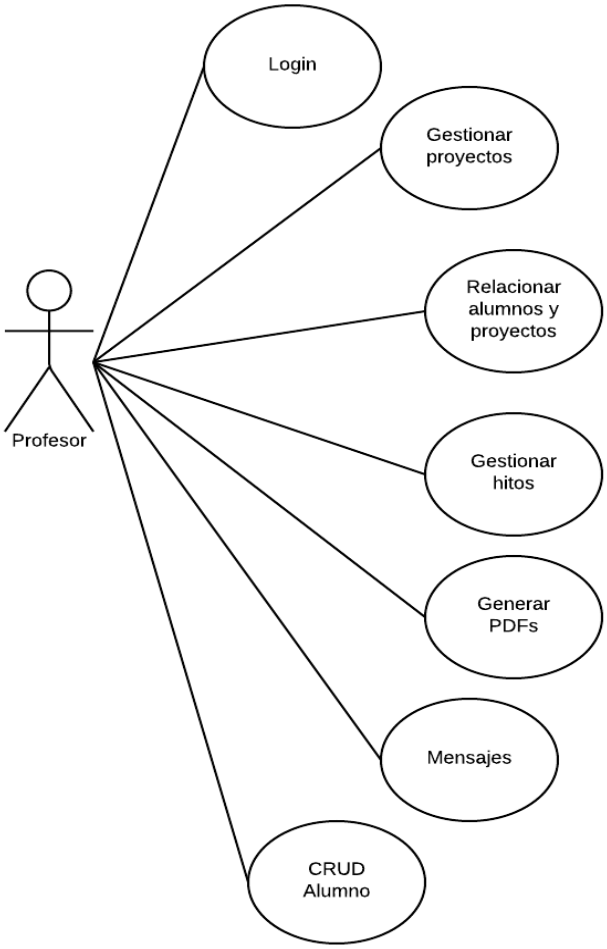
2.-Diagrama de clases



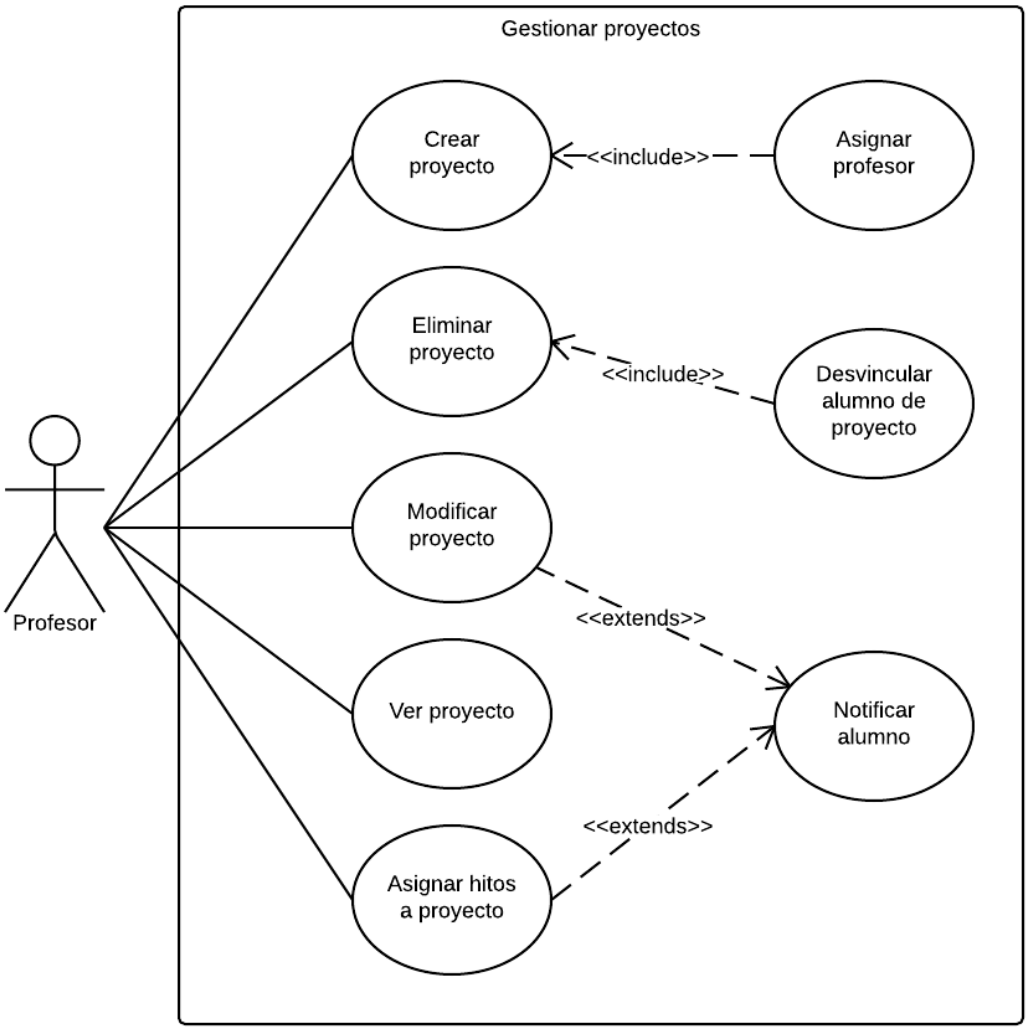
3.-Diagramas de casos de uso

3.1.-Casos de uso de usuario profesor

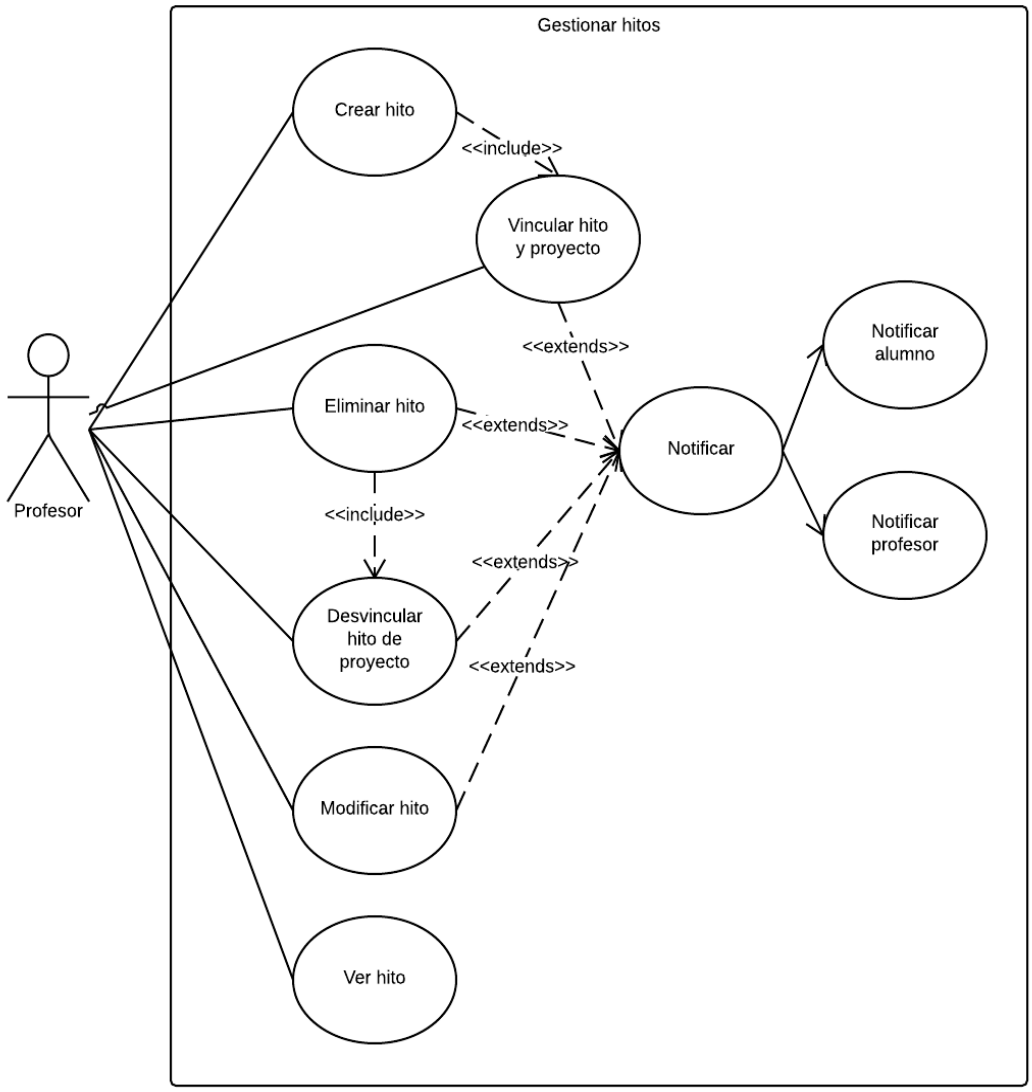
3.1.1.-Caso base



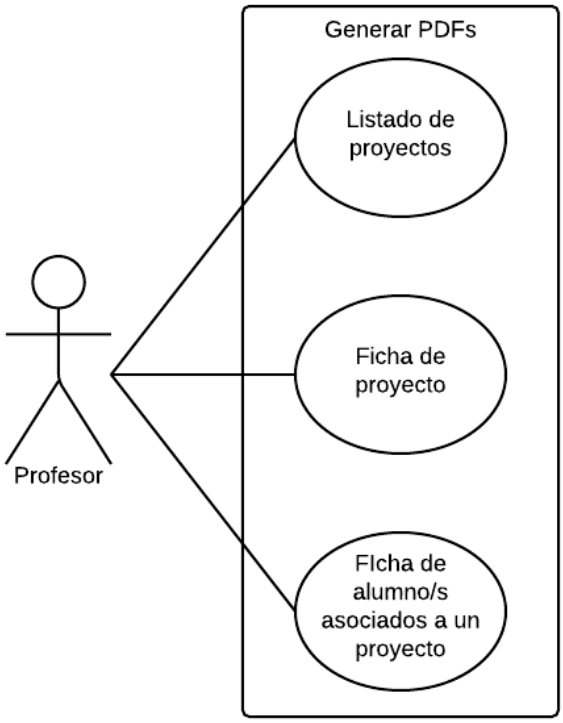
3.1.2.-Gestionar proyectos



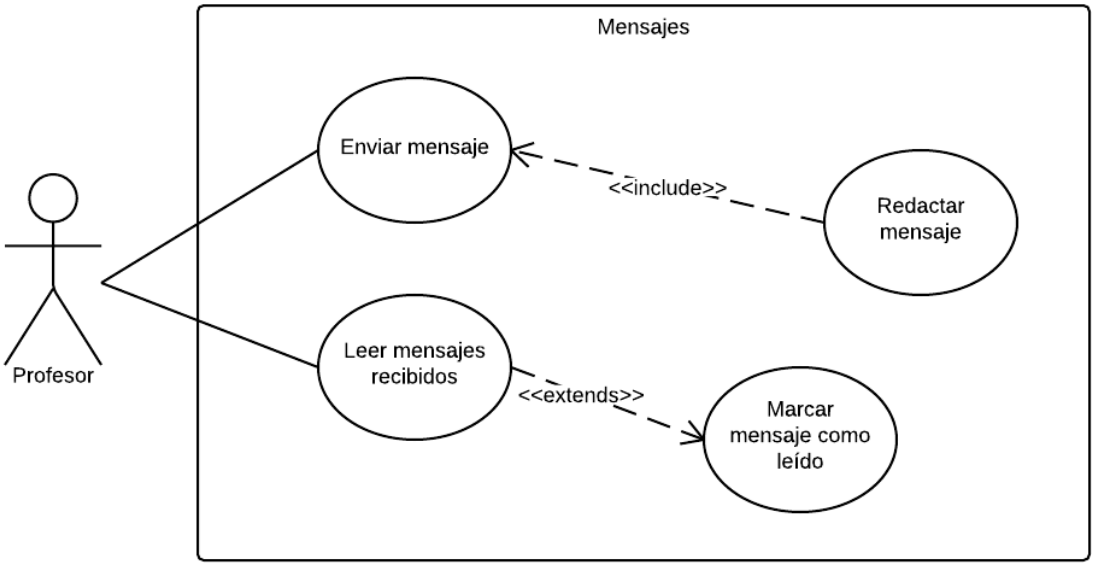
3.1.3.-Gestionar hitos



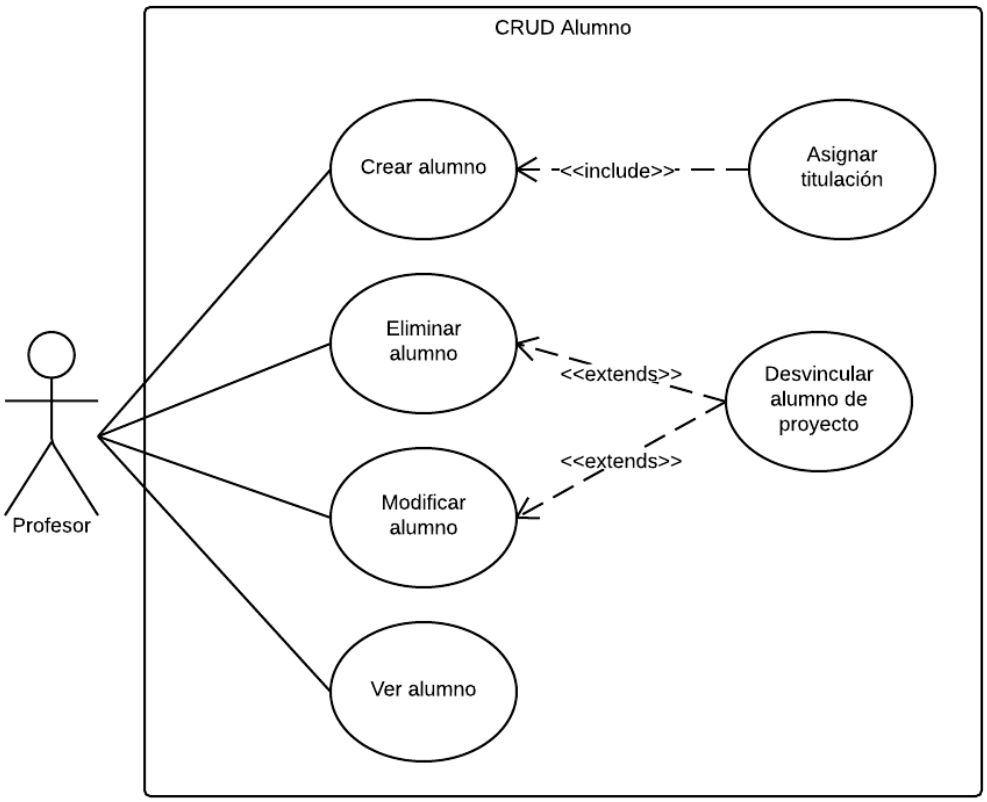
3.1.4.-Generar PDFs



3.1.5.-Mensajes

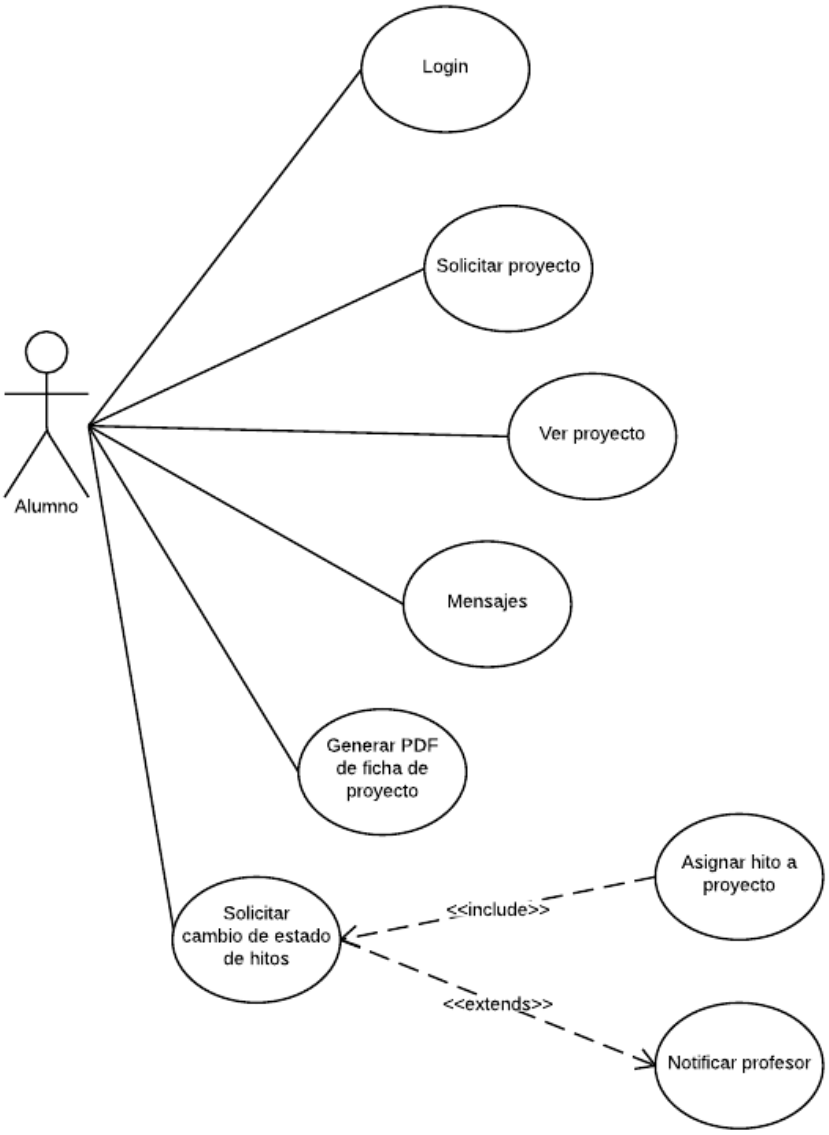


3.1.6.-CRUD Alumno

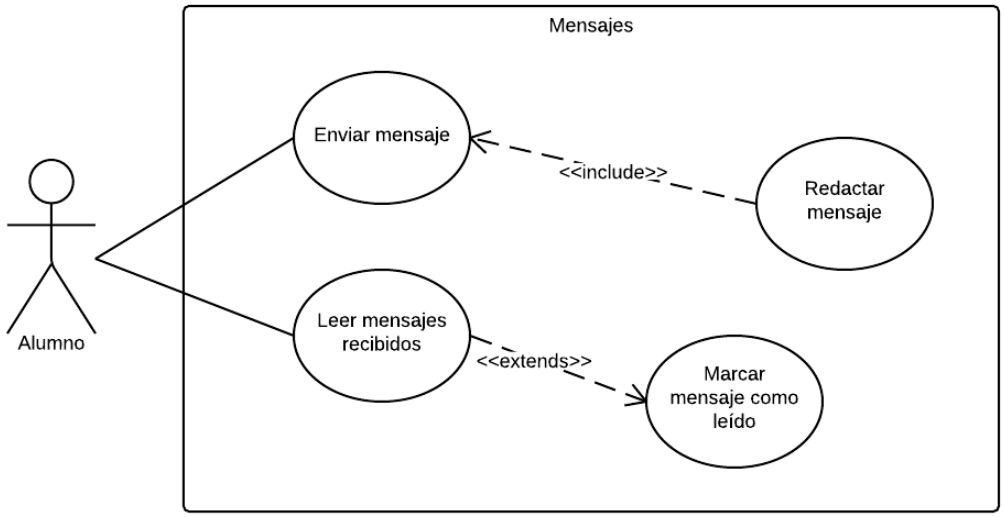


3.2-Casos de uso de usuario alumno

3.2.1.-Caso base

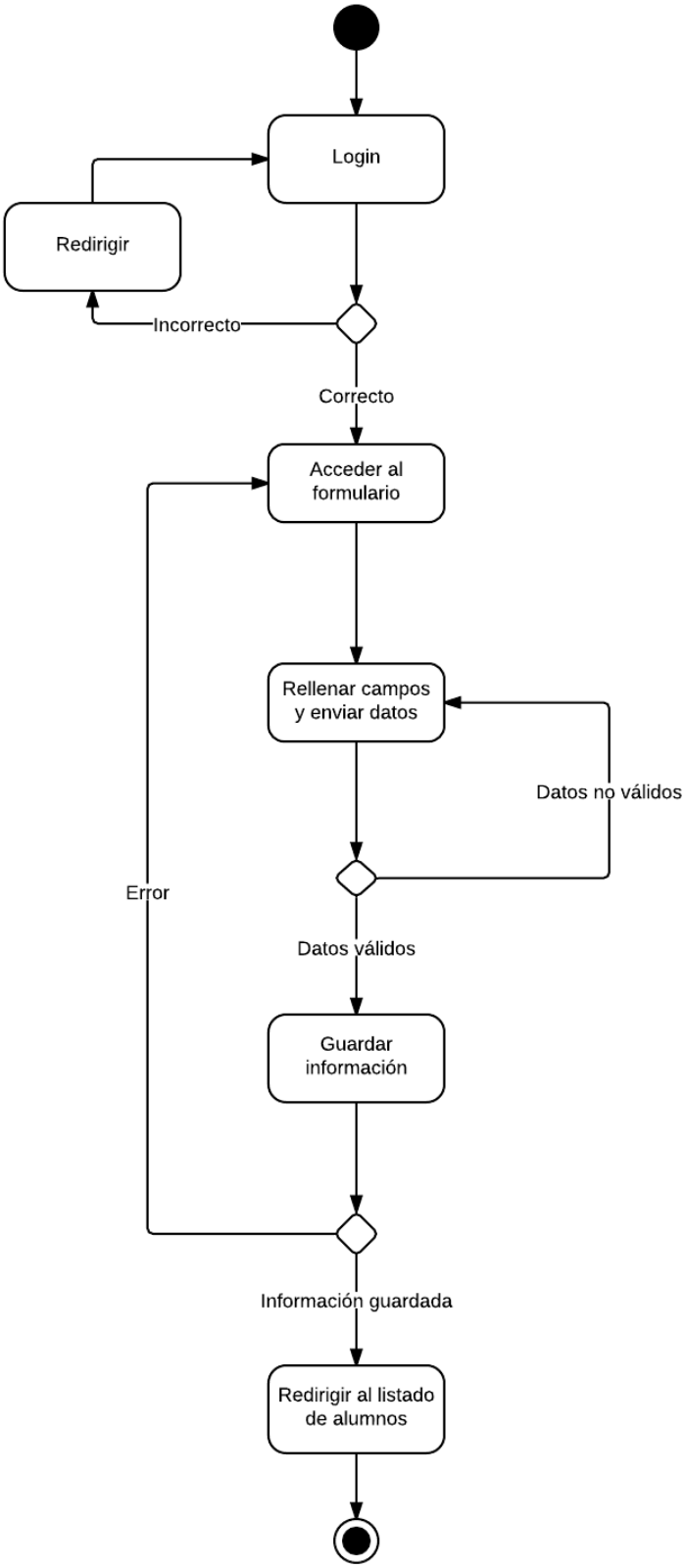


3.2.2.-Mensajes

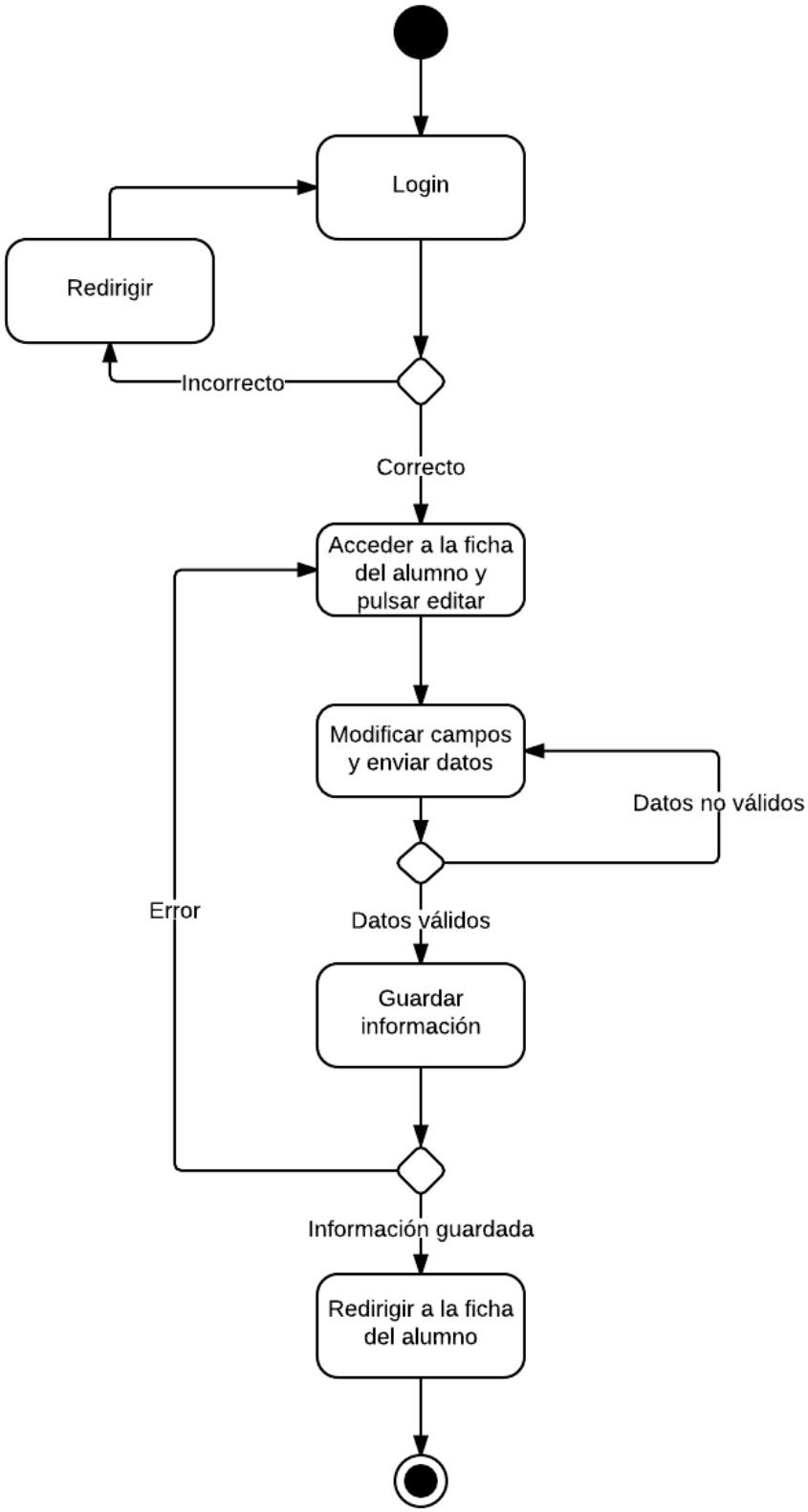


4.-Diagramas de actividad

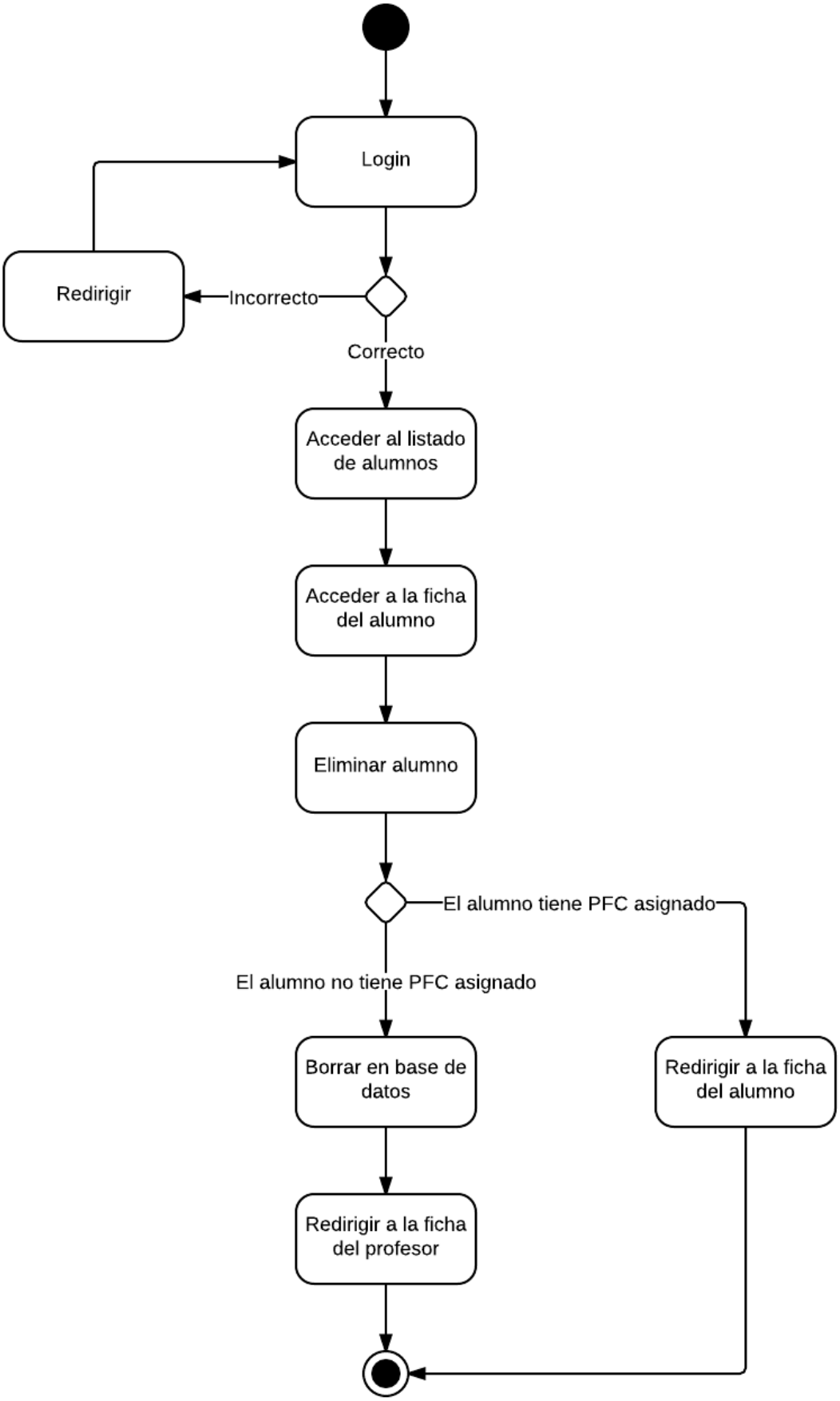
4.1.-Crear alumno



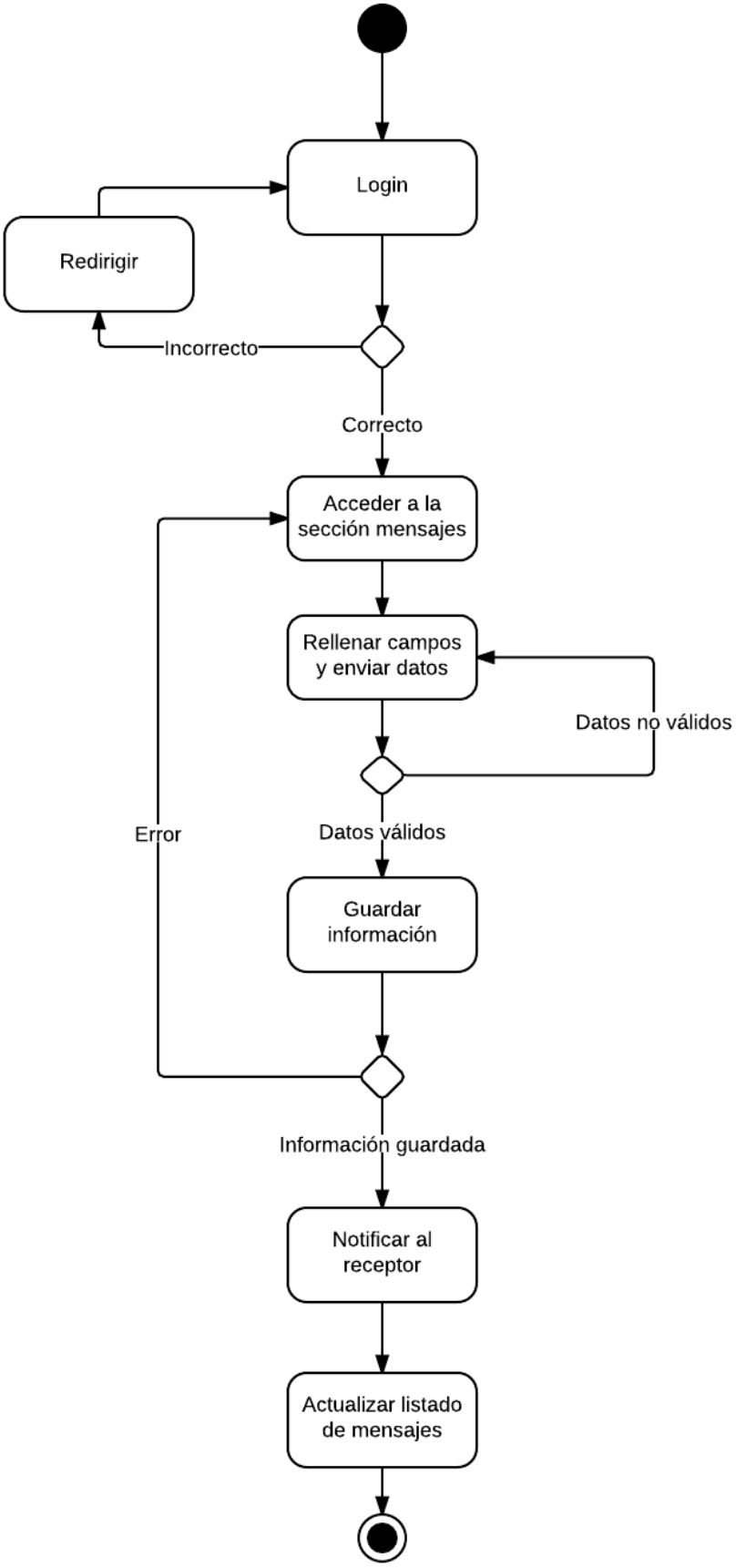
4.2.-Editar alumno



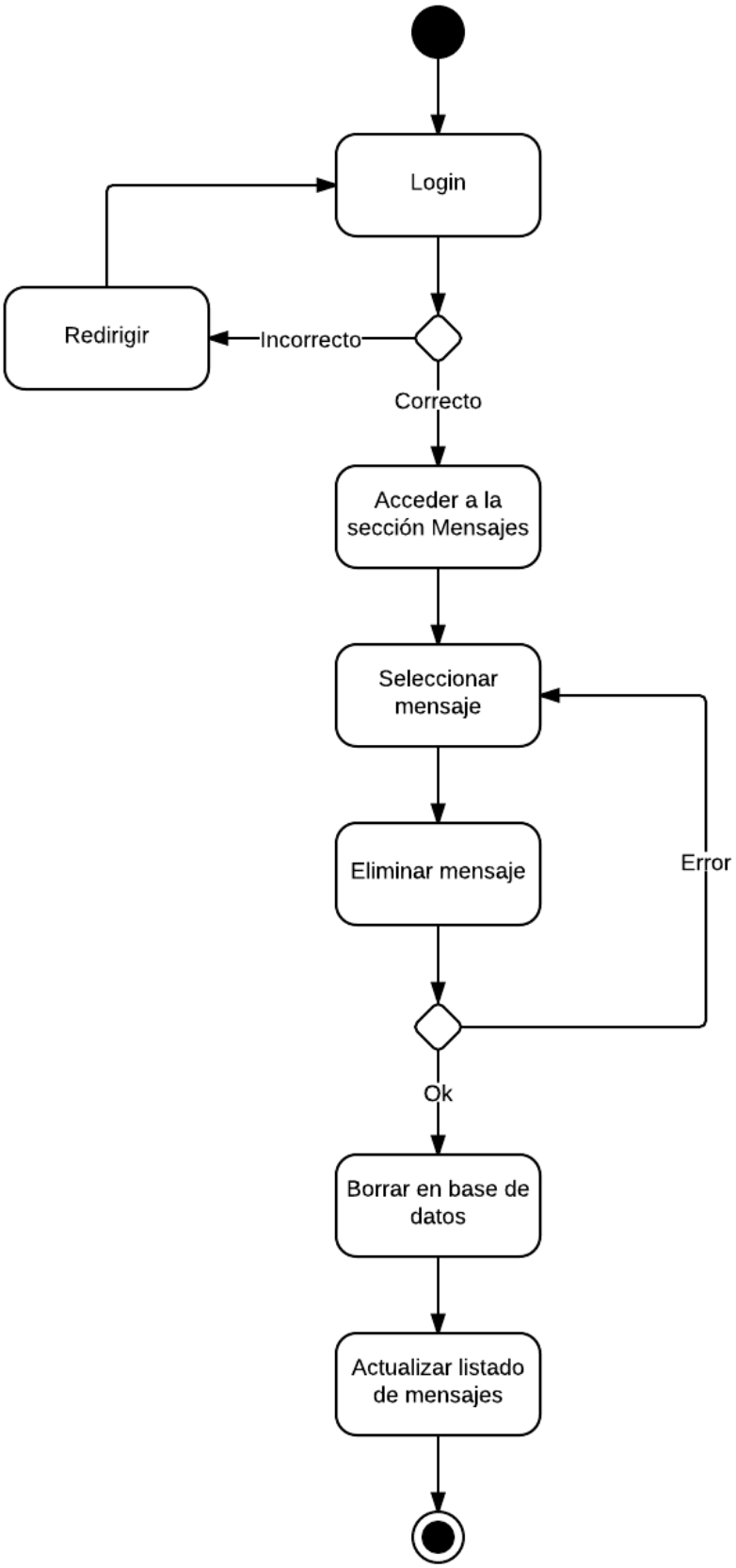
4.3.-Eliminar alumno



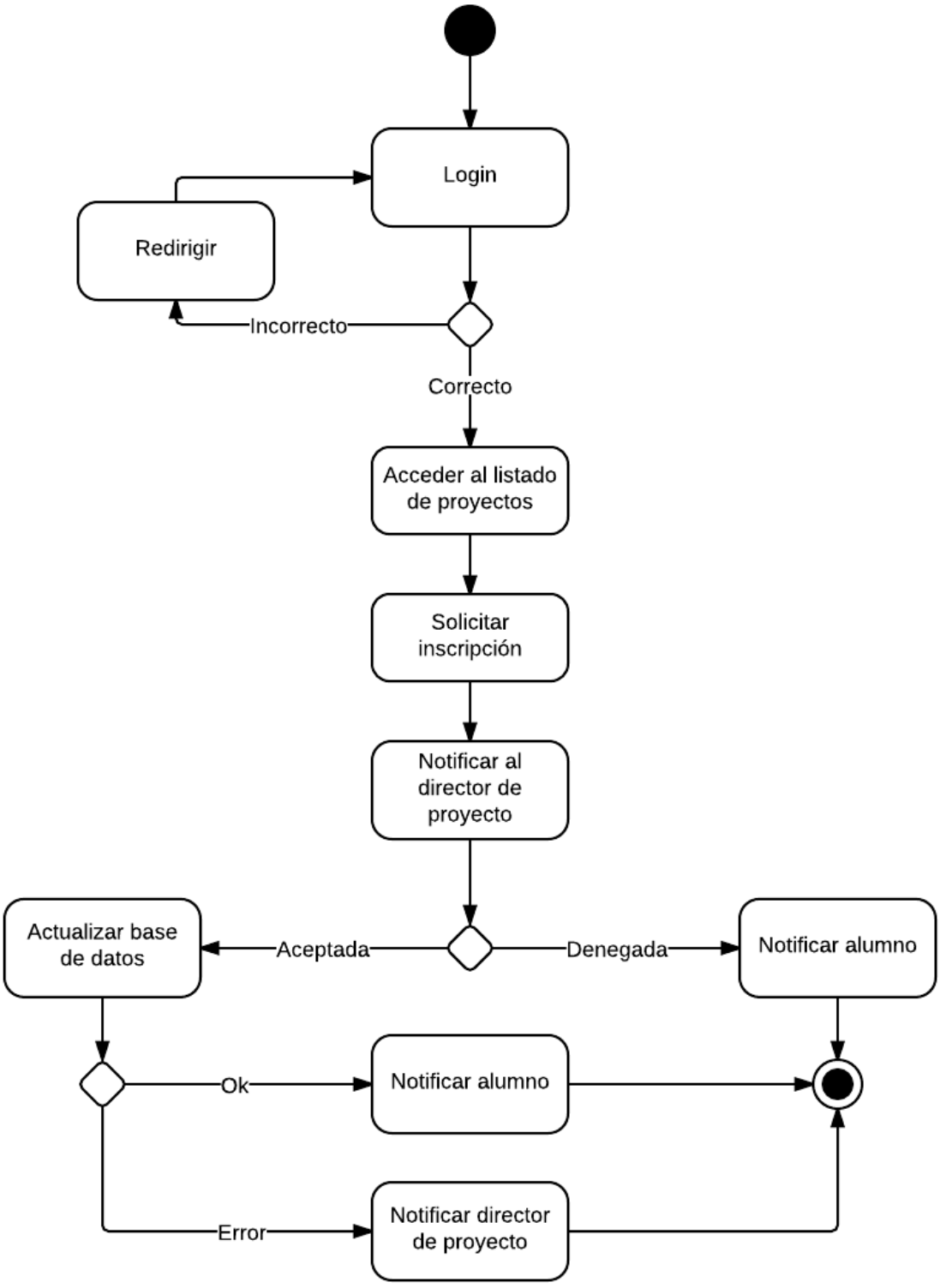
4.4.-Enviar mensaje



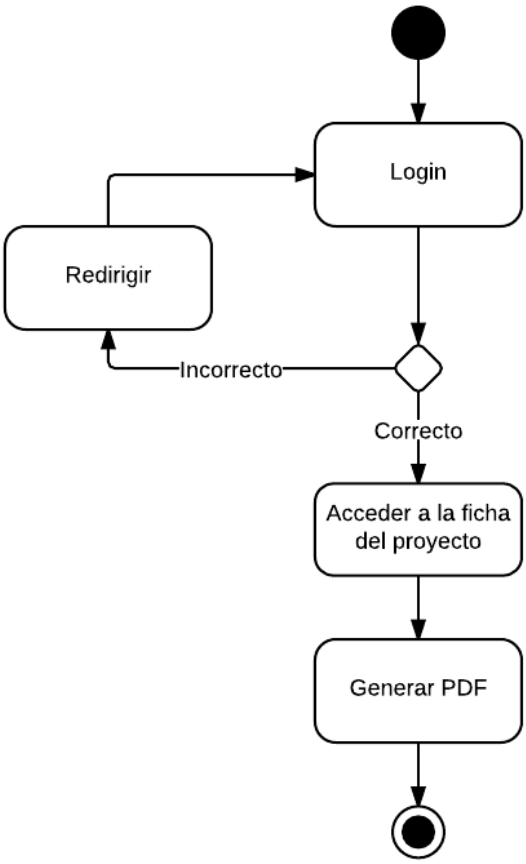
4.5.-Eliminar mensaje



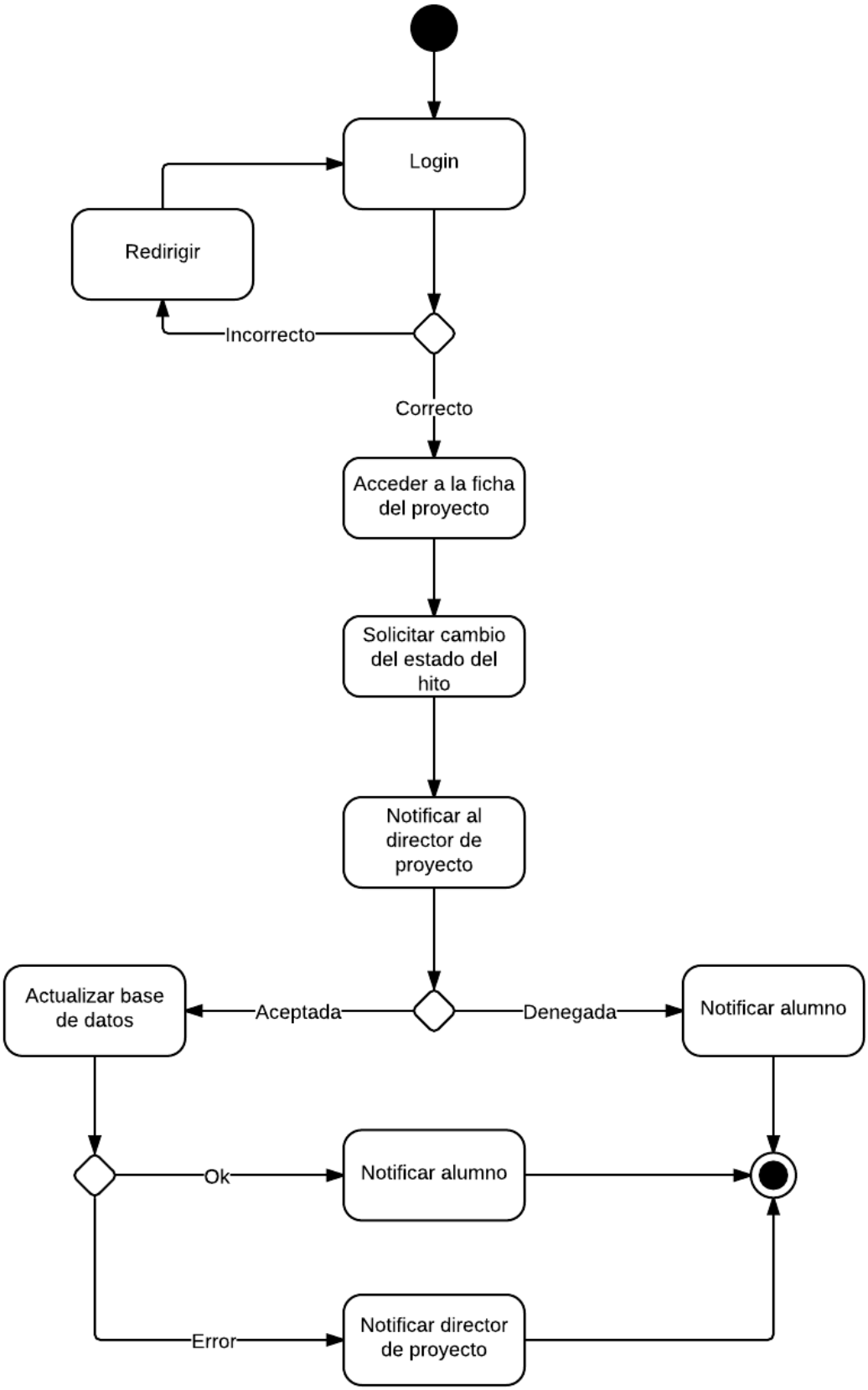
4.6.-Solicitar inscripción en proyecto



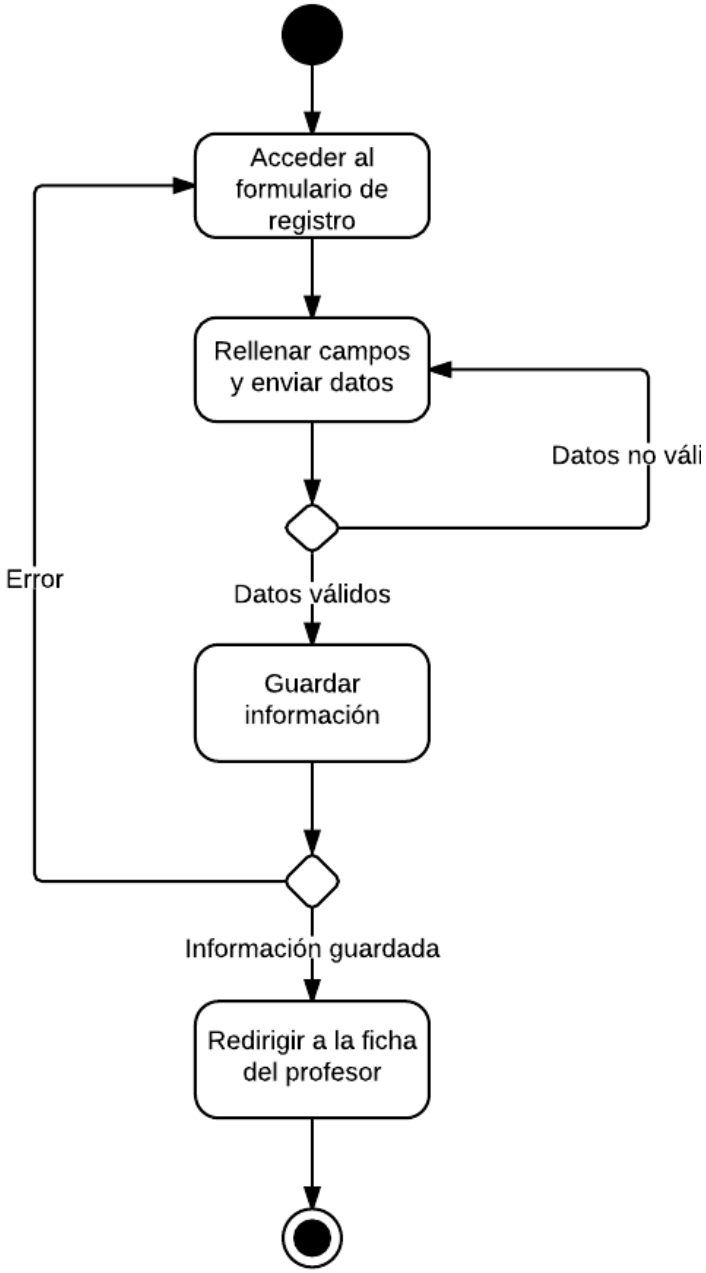
4.7.-Generar PDF (alumno)



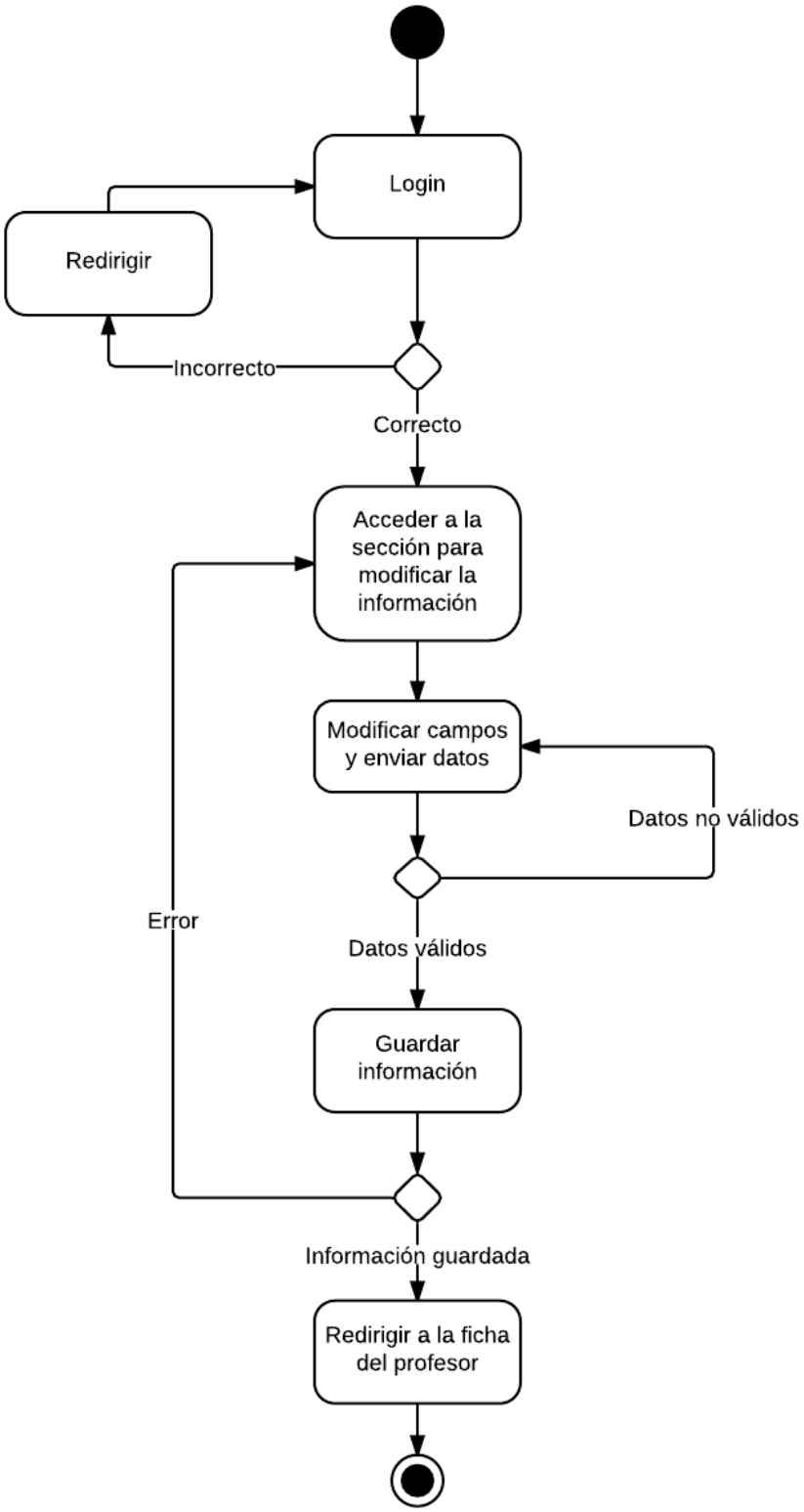
4.8.-Solicitar cambio de estado de hito



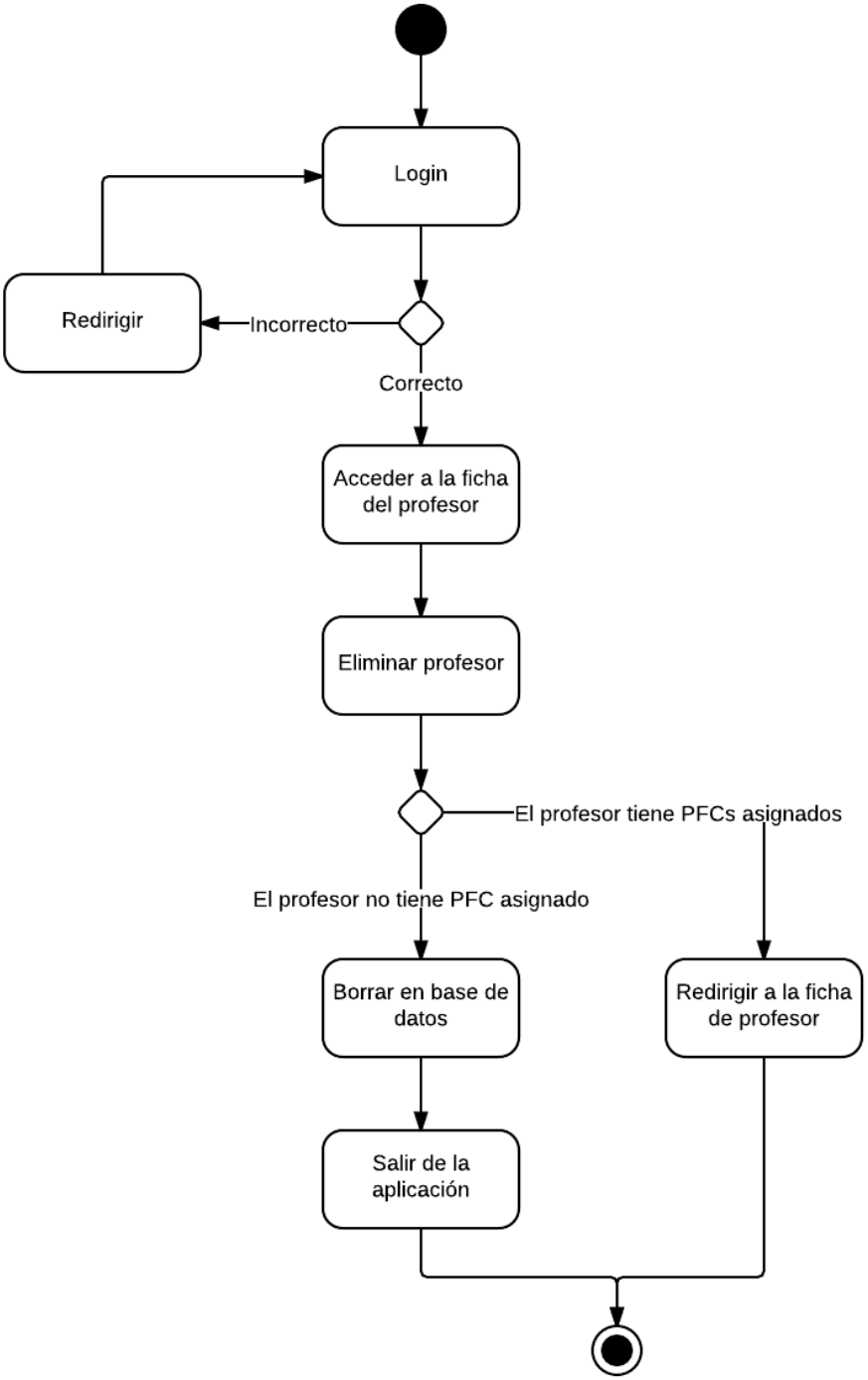
4.9.-Crear profesor



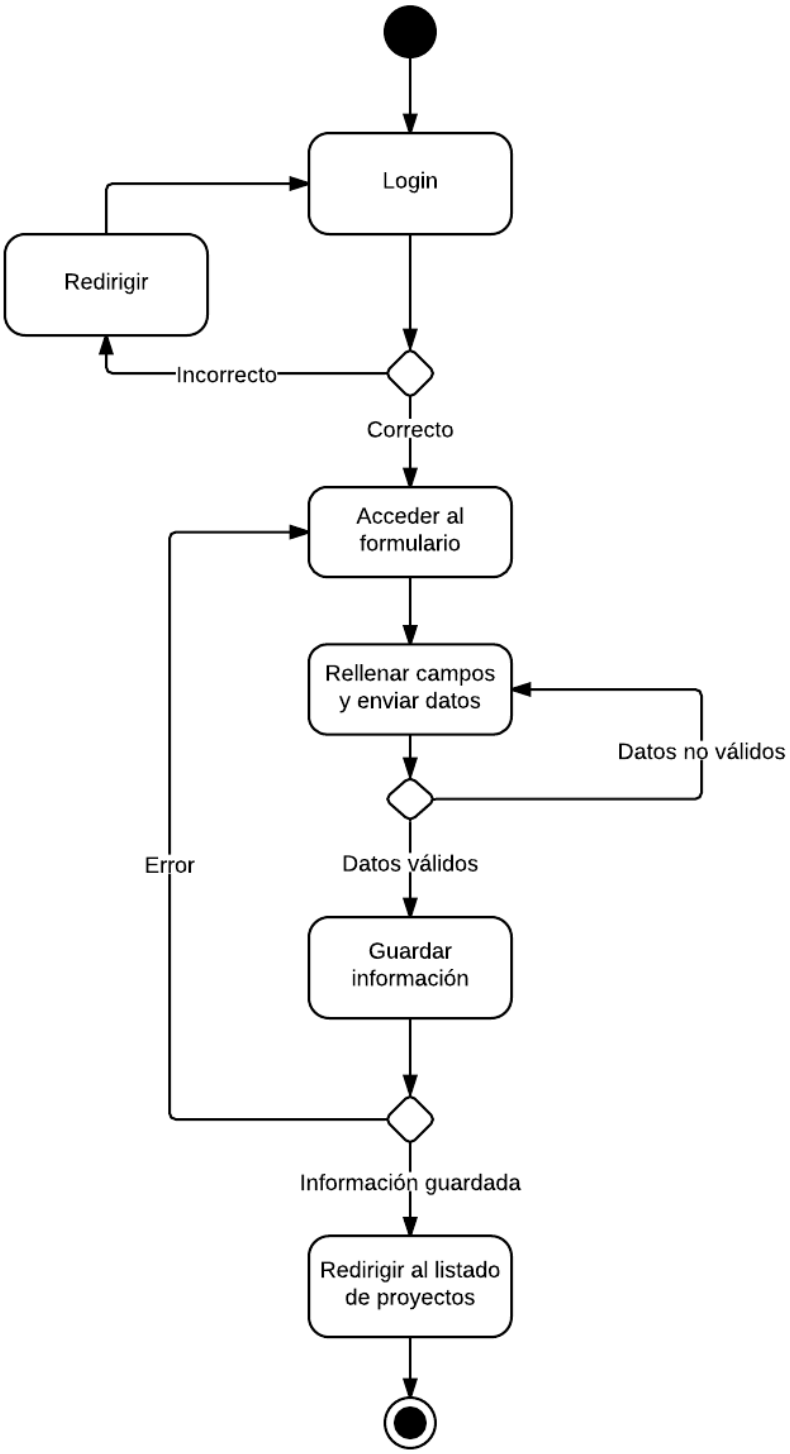
4.10.-Editar profesor



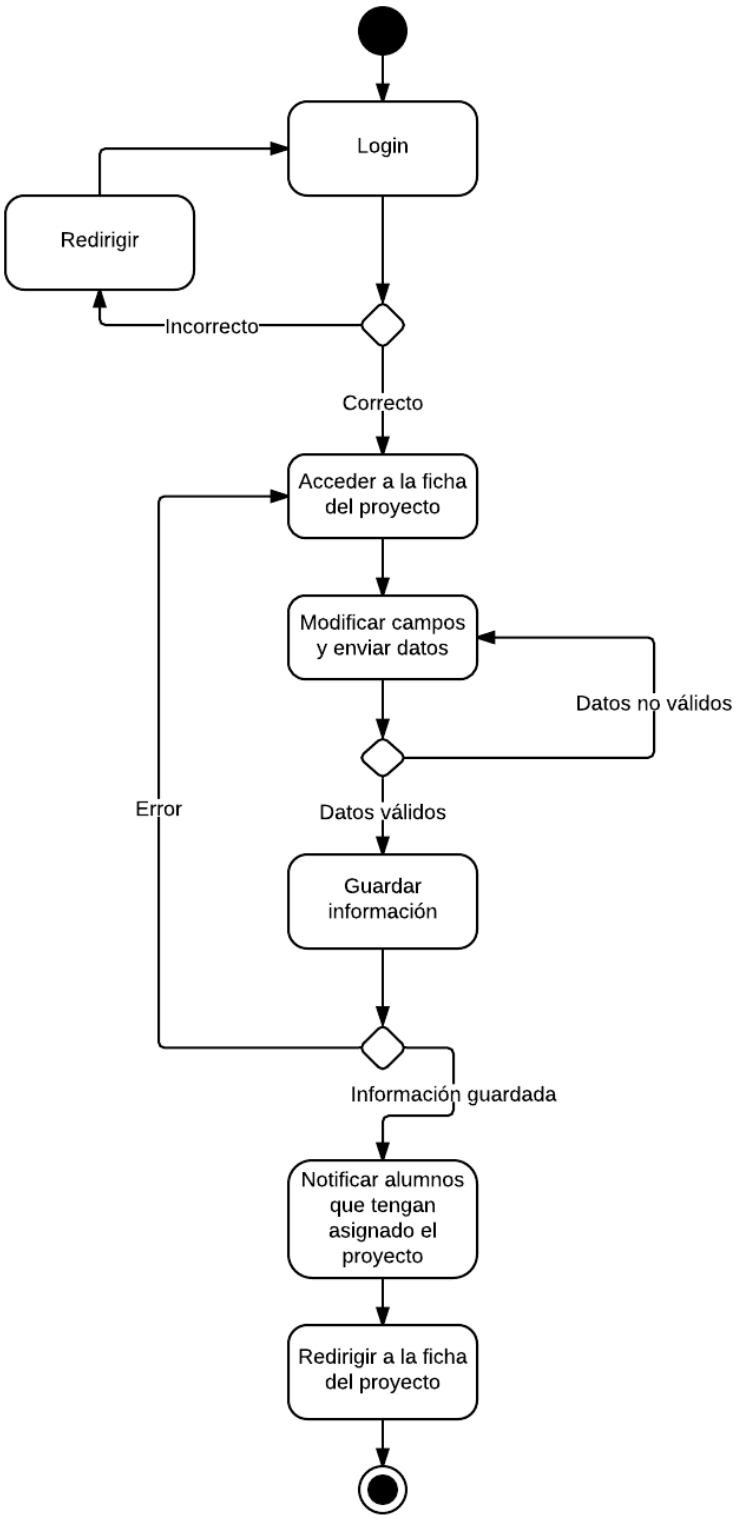
4.11.-Eliminar profesor



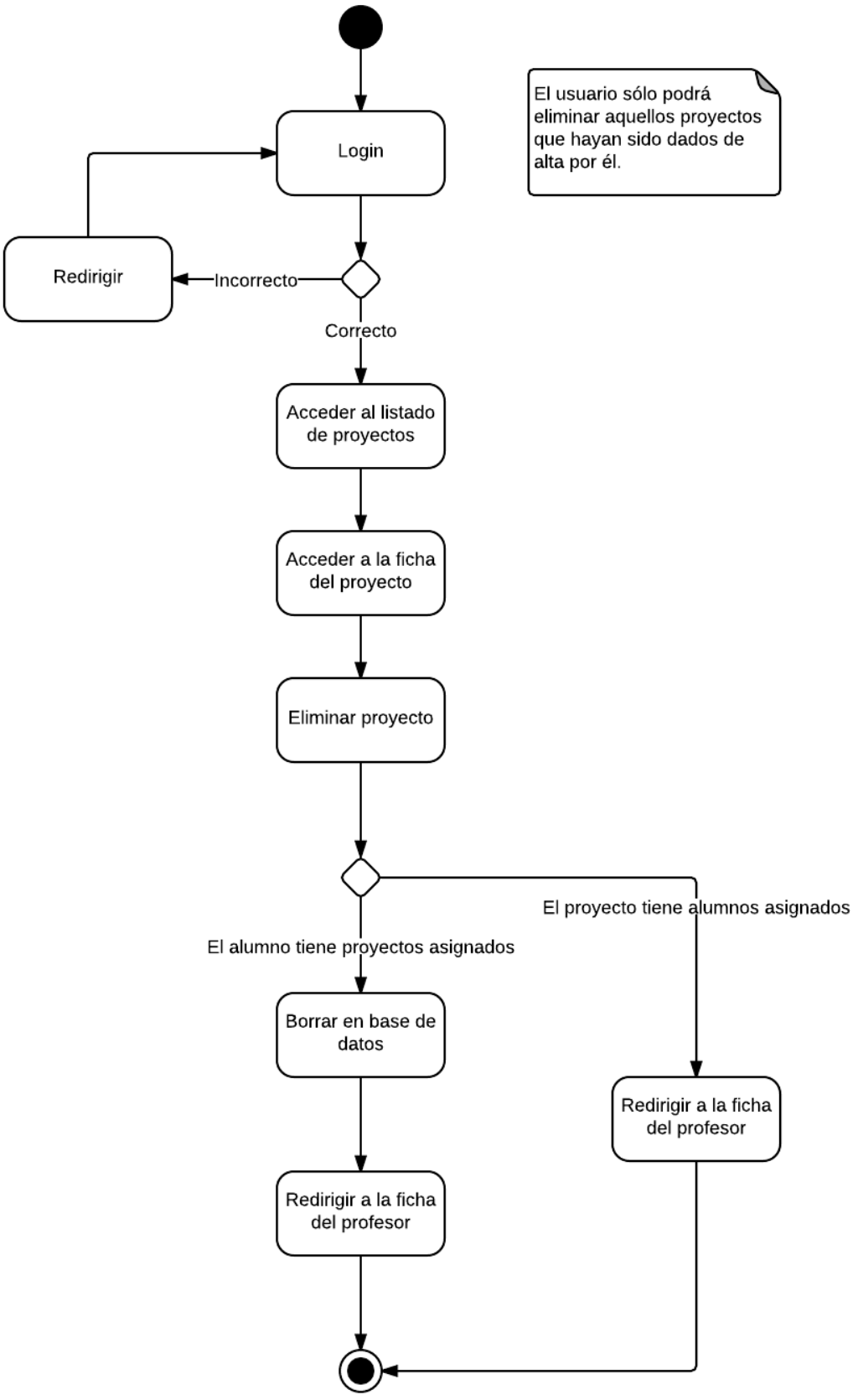
4.12.-Crear proyecto



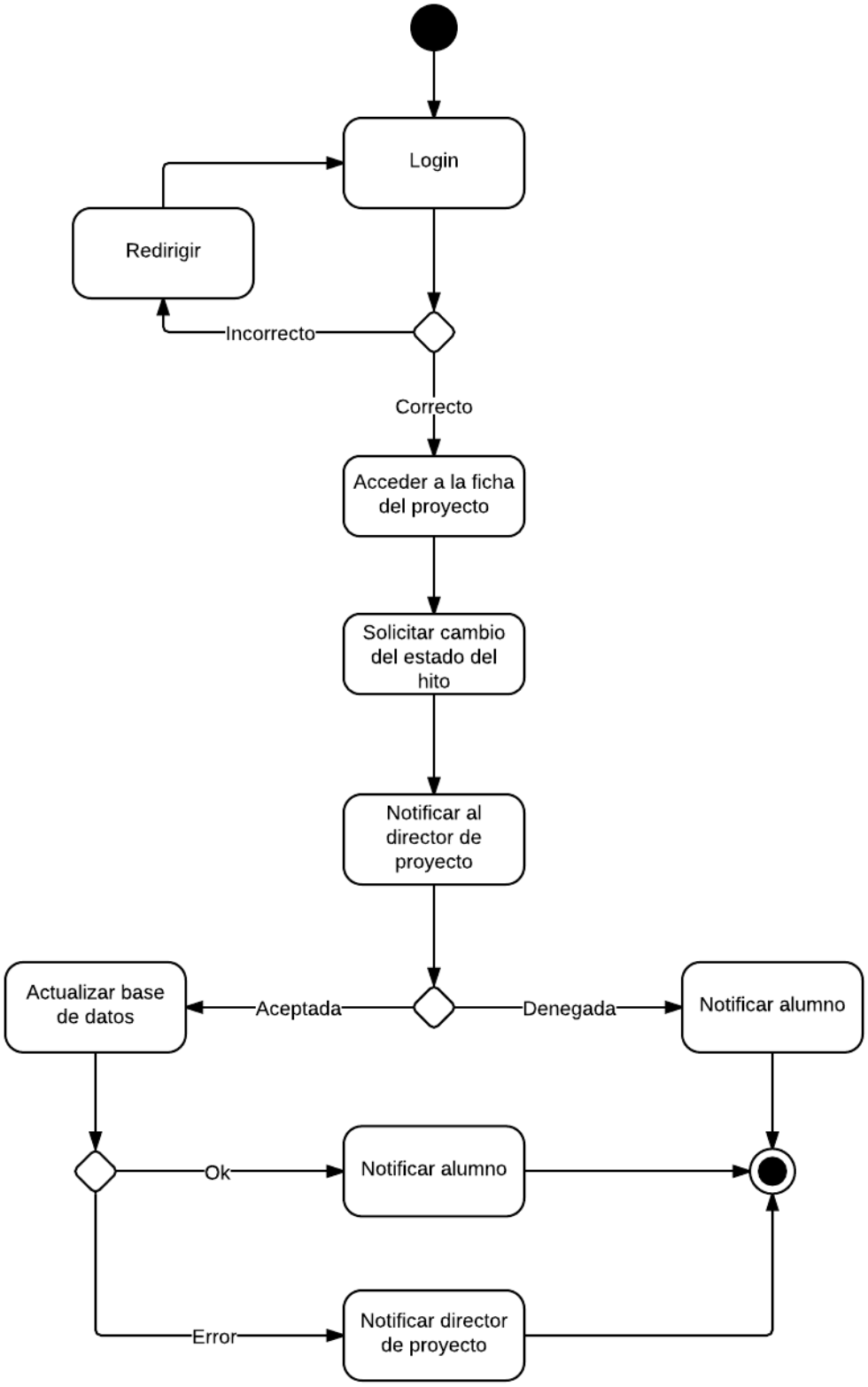
4.13.-Editar proyecto



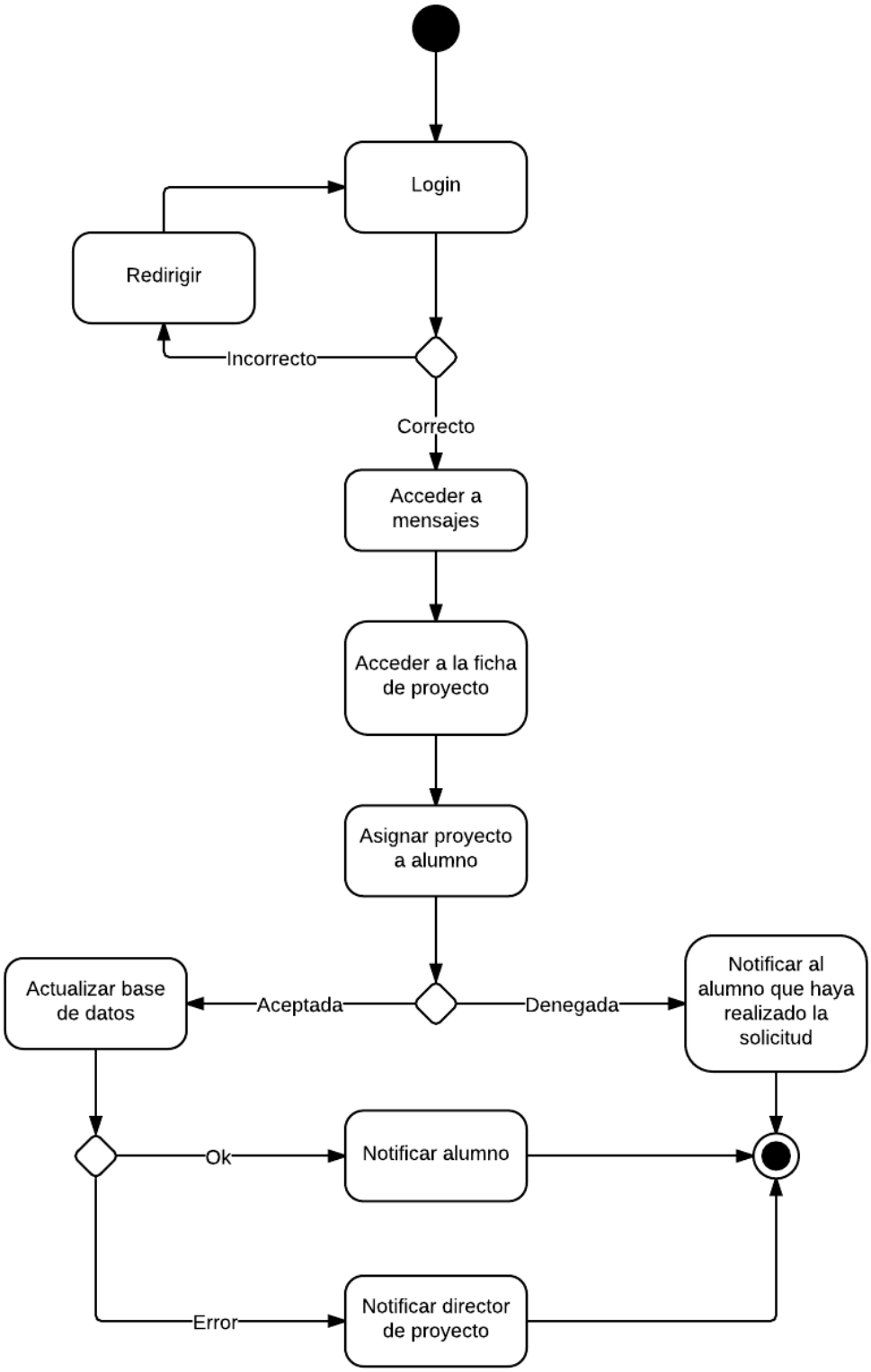
4.14.-Eliminar proyecto



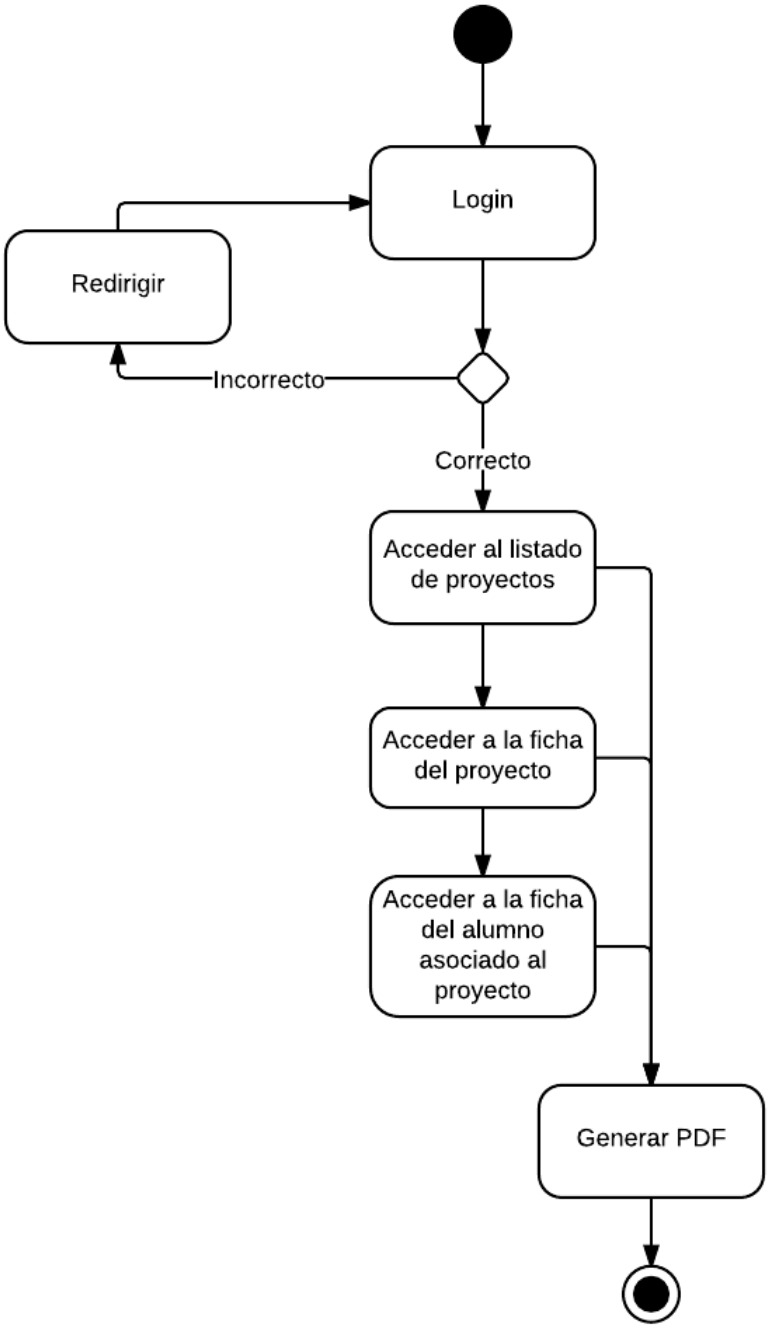
4.15.-Cambiar estado de hito



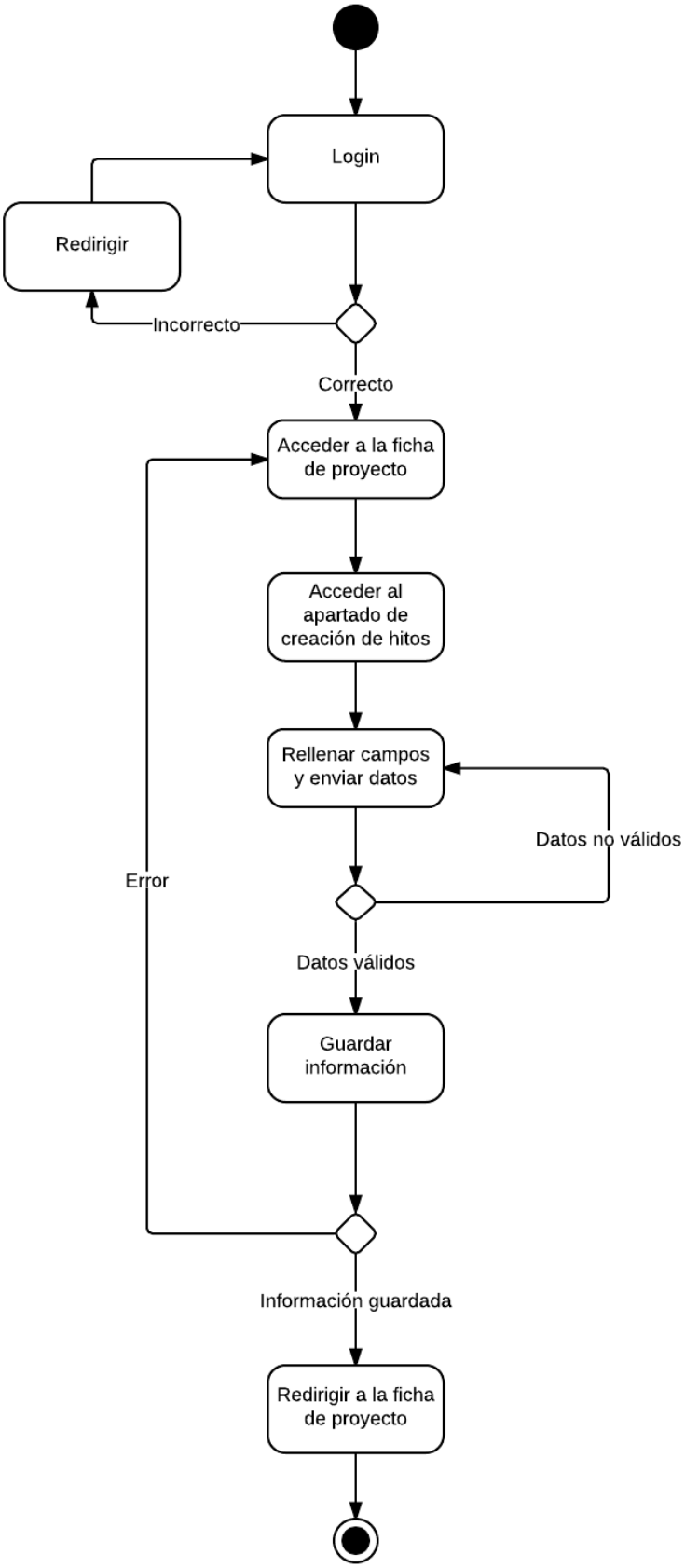
4.16.-Aceptar inscripción de alumno en proyecto



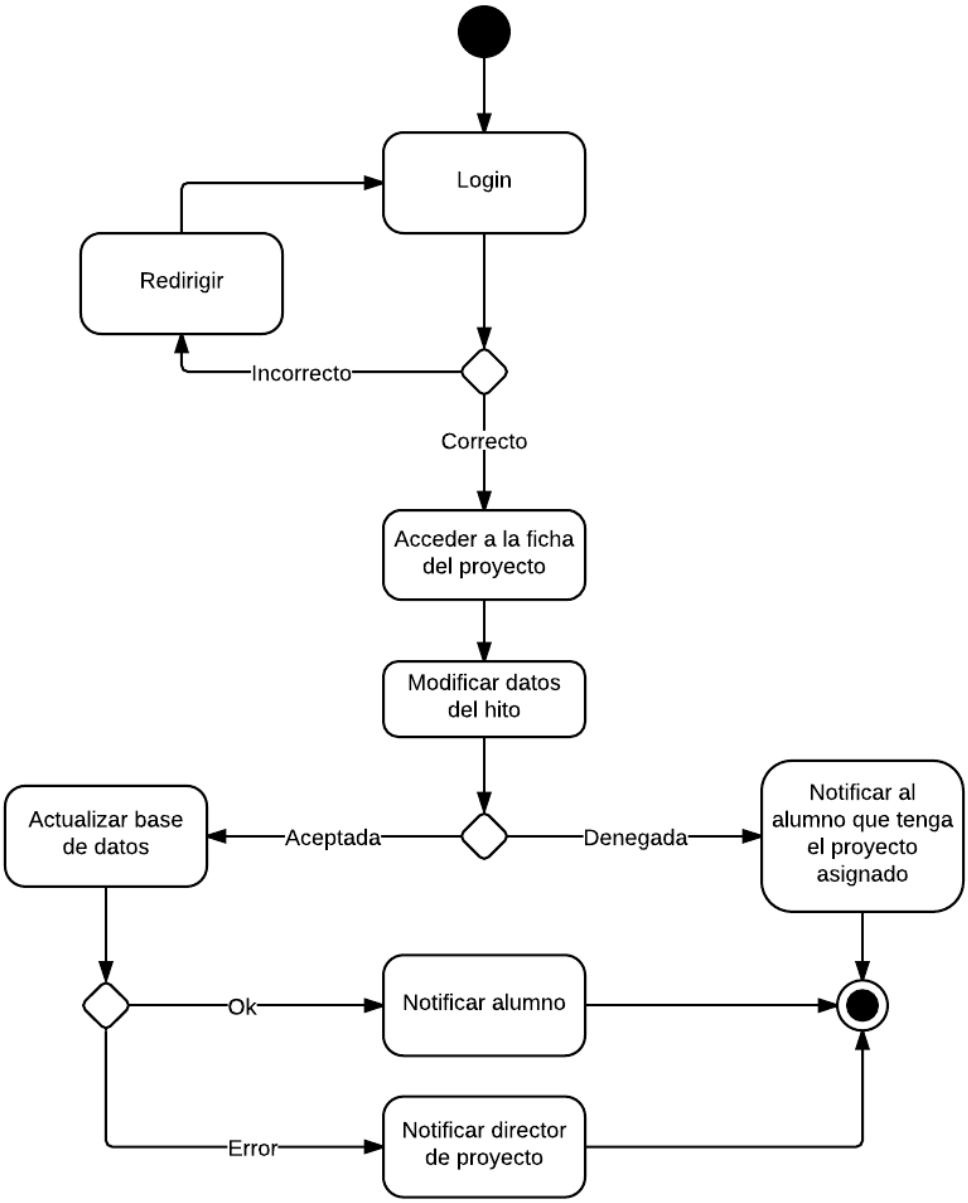
4.17.-Generar PDF (profesor)



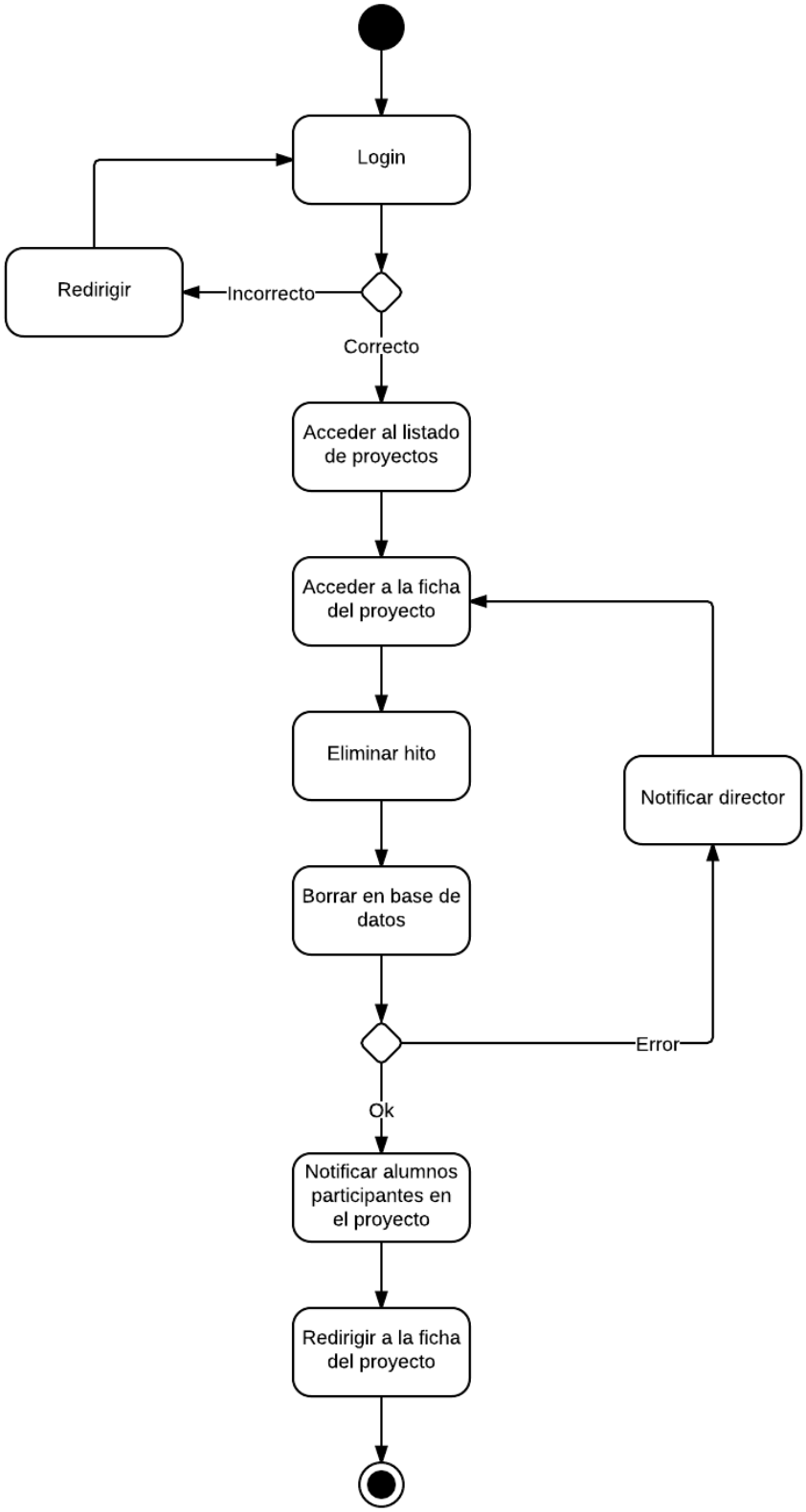
4.18.-Crear hito



4.19.-Editar hito



4.20.-Eliminar hito



IV.-DISEÑO

1.-Introducción

En este apartado hablaremos de los distintos patrones empleados para diseñar la aplicación. Explicaremos en detalle los fundamentos de la arquitectura seguida, así como los distintos factores que la caracterizan.

2.-Patrón de diseño MVC

Es un patrón que separa en tres capas la estructura de una aplicación. Esta separación define, a grandes rasgos, una capa para los datos de la aplicación, una capa para las interfaces gráficas que interactúan con el usuario y otra para la lógica de control. Para la aplicación que nos ocupa la separación sería:

-Capa de datos (Modelo): la BBDD de la aplicación y los modelos a partir de los cuales se mapea ésta.

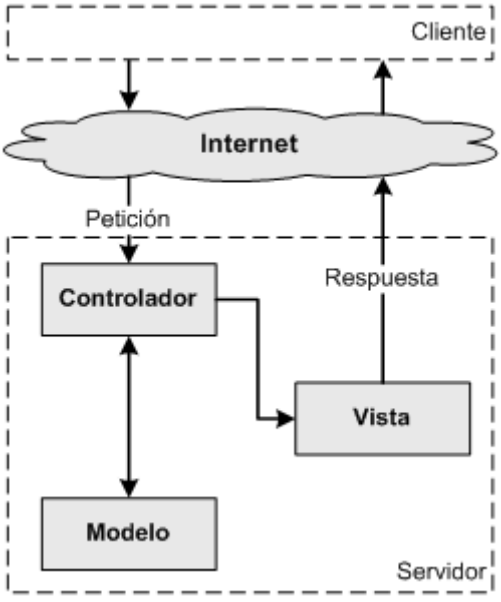
-Capa de interfaces gráficas (Vista): son las vistas que devuelven los controladores, concretamente archivos HTML.

-Capa de lógica (Controlador): los controladores y servicios que hacen de enlace entre la capa de datos y las interfaces.

Pasemos ahora a hacer una explicación más exhaustiva de cada una de las capas.

2.1.-Modelo

El modelo es un conjunto de clases que representan la información del mundo real que el sistema debe reflejar. Es la parte encargada de representar la lógica de negocio de una aplicación. El modelo, a nivel teórico, no debe tener conocimiento acerca de la existencia de las vistas y del controlador. Esto en Symfony se cumple totalmente ya que podemos tener un modelo que no conozca nada absolutamente nada sobre los controladores que actuarán sobre él.



Las aplicaciones en Symfony definen los modelos a partir de clases, las cuales representan las tablas de las que se compone la BBDD. Cada uno de los atributos

que compone estas clases representa un campo de la tabla. Destacar que esto se lleva a cabo gracias al ORM Doctrine, pero posteriormente entraremos en más detalle acerca de este proceso.

2.2.-Vista

Esta capa presenta aquellos elementos con los cuales el usuario tendrá que interactuar. Formularios, botones, popups, etc. Todo lo que se refiera a la visualización de la información, el diseño, colores, estilos y la estructura visual en sí de nuestras páginas.

En el caso de nuestro proyecto las vistas son documentos HTML renderizados gracias a un motor de plantillas denominado Twig. Básicamente este motor permite la inserción de pequeñas porciones de lógica en las vistas. De esta forma se favorece una mayor flexibilidad a la hora de mostrar contenido dinámico.

2.3.-Controlador

Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista. En Symfony, este comportamiento viene definido dentro de cada Bundle en unas clases llamadas Controllers. Éstas se estructuran en métodos los cuales normalmente gestionan las distintas acciones del usuario, ya sean navegaciones a otras secciones de la aplicación, o bien gestiones más lógicas como procesar datos.

3.- ¿Cómo funciona una aplicación MVC?

Captura de la petición en el controlador

La aplicación recibe peticiones que son centralizadas en el Controlador. Éste es el encargado de interpretar, a partir de la URL de la solicitud, el tipo de operación que hay que realizar. Normalmente, esto se hace analizando el valor de algún parámetro que se envía anexando a la URL de la petición y que se utiliza con esta finalidad.

Procesamiento de la petición

Una vez que el Controlador determine la operación a realizar, procede a ejecutar las acciones pertinentes, según el método apropiado.

Generación de respuestas

El controlador devolverá la vista adecuada acompañada de las variables necesarias para mostrar los datos requeridos.

4.-Beneficios

- La implementación se realiza de forma modular.
- Sus vistas muestran información actualizada siempre. El programador no debe preocuparse de solicitar que las vistas se actualicen, ya que este proceso es realizado automáticamente por el modelo de la aplicación.
- Cualquier modificación que afecte al dominio, como aumentar métodos o datos contenidos, implica una modificación sólo en el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
- Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información, no su tratamiento.
- MVC está demostrando ser un patrón de diseño bien elaborado pues las aplicaciones que lo implementan presentan una extensibilidad y una mantenibilidad únicas comparadas con otras aplicaciones basadas en otros patrones.

5.-Inconvenientes

- Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo. Normalmente el patrón exige al programador desarrollar un mayor número de clases que, en otros entornos de desarrollo, no son necesarias. Sin embargo, esta desventaja es muy relativa ya que posteriormente, en la etapa de mantenimiento de la aplicación, una aplicación MVC es mucho más mantenible, extensible y modificable que una aplicación que no lo implementa.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación. Esta arquitectura inicial debe incluir, por lo menos, un mecanismo de eventos para poder proporcionar las notificaciones que genera el modelo de aplicación; una clase Modelo, otra clase Vista y una clase Controlador genéricas que realicen todas las tareas de comunicación, notificación y actualización que serán luego transparentes para el desarrollo de la aplicación.
- MVC es un patrón de diseño orientado a objetos por lo que su implementación es sumamente costosa y difícil en lenguajes que no siguen este paradigma.

V.-IMPLEMENTACIÓN

1.-Instalación y configuración para desarrollos en un entorno Symfony

Para poder utilizar el framework Symfony, debemos tener previamente instalados un servidor web con PHP, un gestor de base de datos SQL y Composer. En este caso vamos a utilizar Wamp server como servidor Apache y phpMyAdmin para gestionar nuestra base de datos. Cabe destacar que existen otras herramientas como XAMPP Server o HeidiSQL para gestionar las base de datos.

Respecto a la instalación de Symfony, existen 3 formas:

1. Instalación mediante Composer, que es la que vamos a utilizar.
2. Instalación mediante un archivo comprimido con vendors, sólo se recomienda su uso a los programadores principiantes o a los que quieren probar Symfony2 lo más rápidamente posible antes de decidir si utilizan Symfony2 o no para su proyecto.
3. Instalación mediante un archivo comprimido sin vendors, es una forma ligeramente diferente de hacer lo mismo que en la instalación mediante Composer.

1.1.-Instalación de composer

Composer es un manejador de dependencias para PHP. Los proyectos PHP grandes, como por ejemplo las aplicaciones Symfony2, dependen a su vez de muchos otros proyectos.
Cuando envías por ejemplo un email, Symfony2 utiliza una librería externa llamada SwiftMailer. Así que para que tu aplicación funcione bien, Symfony2 necesita que todas esas librerías externas (llamadas ***dependencias***) se instalen correctamente.

Hay varias formas de instalar Composer, la primera y más sencilla si utilizas Windows es con el instalador Composer-Setup.exe, que podemos obtener en http://getcomposer.org/download/. Sólo tendremos que seguir el wizard y él mismo nos actualizará el valor de nuestra variable PATH, de esta forma, y tras reiniciar la consola, podremos ejecutar las órdenes deseadas.

Las siguientes instrucciones explican cómo instalar Composer de forma global en tu ordenador. Esto supone que la instalación es un poco más larga de lo normal, pero a cambio no tendrás que reinstalar Composer para cada nuevo proyecto.

1. Abre cualquier navegador y accede a la siguiente dirección http://getcomposer.org/installer

2. Guarda el contenido de la página anterior en un archivo llamado instalador.php (no importa el directorio donde lo guardes).

3. Abre la consola de comandos de Windows y entra en el directorio donde has guardado el archivo instalador.php. Por ejemplo:

```
C:\> cd "C:\Users\MiUsuario\Downloads"
```

4. Ejecuta el siguiente comando para instalar Composer:

```
C:\Users\MiUsuario\Downloads> php instalador.php
```

Si todo ha funcionado bien, verás un nuevo archivo composer.phar en ese mismo directorio. Ahora ya puedes borrar el archivo instalador.php.

5. Localiza el directorio donde tienes instalado el archivo php.exe. Si utilizas Wamp por ejemplo, este archivo seguramente se encuentra en C:\wamp\bin\php\php5.4.12\php.exe

6. Mueve el archivo composer.phar anterior al mismo directorio donde se encuentra php.exe.

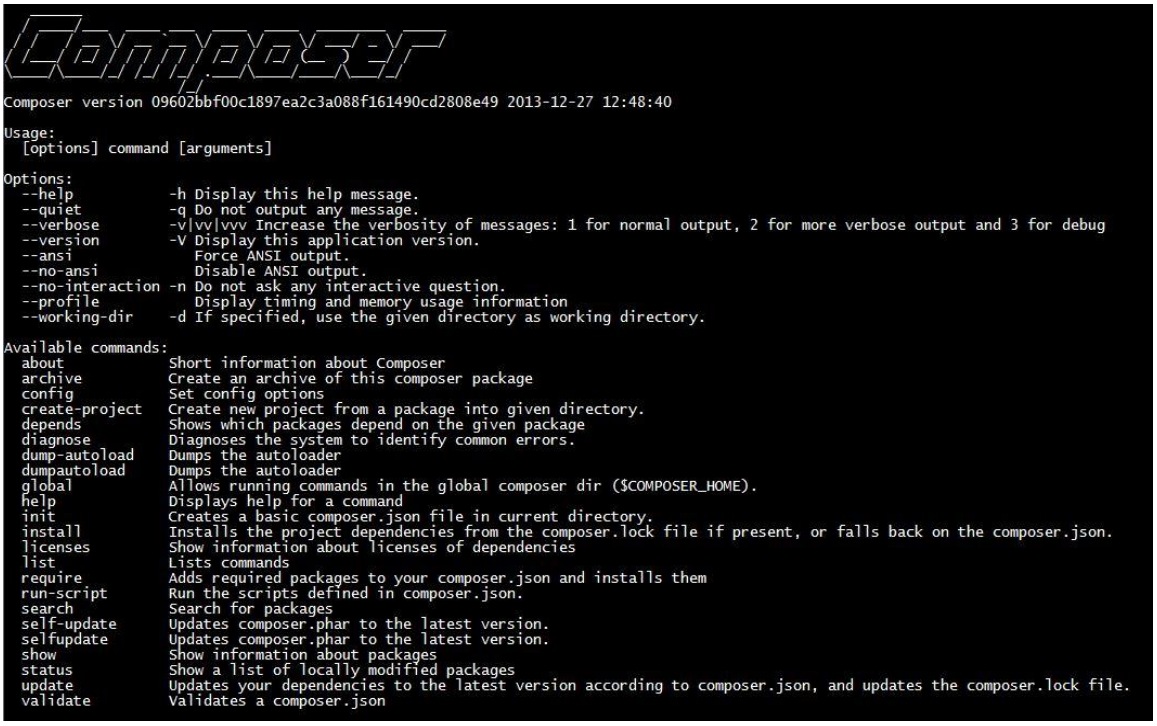
7. Por último, crea un archivo llamado composer.bat en el mismo directorio de php.exe y composer.phar y que contenga las dos siguientes líneas:

```
@ECHO OFF
php composer.phar %*
```

Después de todos estos pasos, abre una nueva consola de comandos y ejecuta el siguiente comando sin ninguna opción:

```
C:\> composer
```

Si todo ha funcionado bien, deberías ver algo como esto:



2.-Creación del proyecto

En primer lugar, elegimos el directorio donde vamos a crear nuestro proyecto Symfony2, y creamos una carpeta con el nombre del proyecto. A continuación, abrimos la consola de comandos de nuestro sistema operativo y ejecutamos lo siguiente:

Si tenemos composer instalado:

```
C:\> composer create-project symfony/framework-standard-edition <directorio>
```

(Cambiando el valor <directorio> por la ruta del directorio donde queremos crear el proyecto, por ejemplo: "C:/wamp/www/MiPFC").

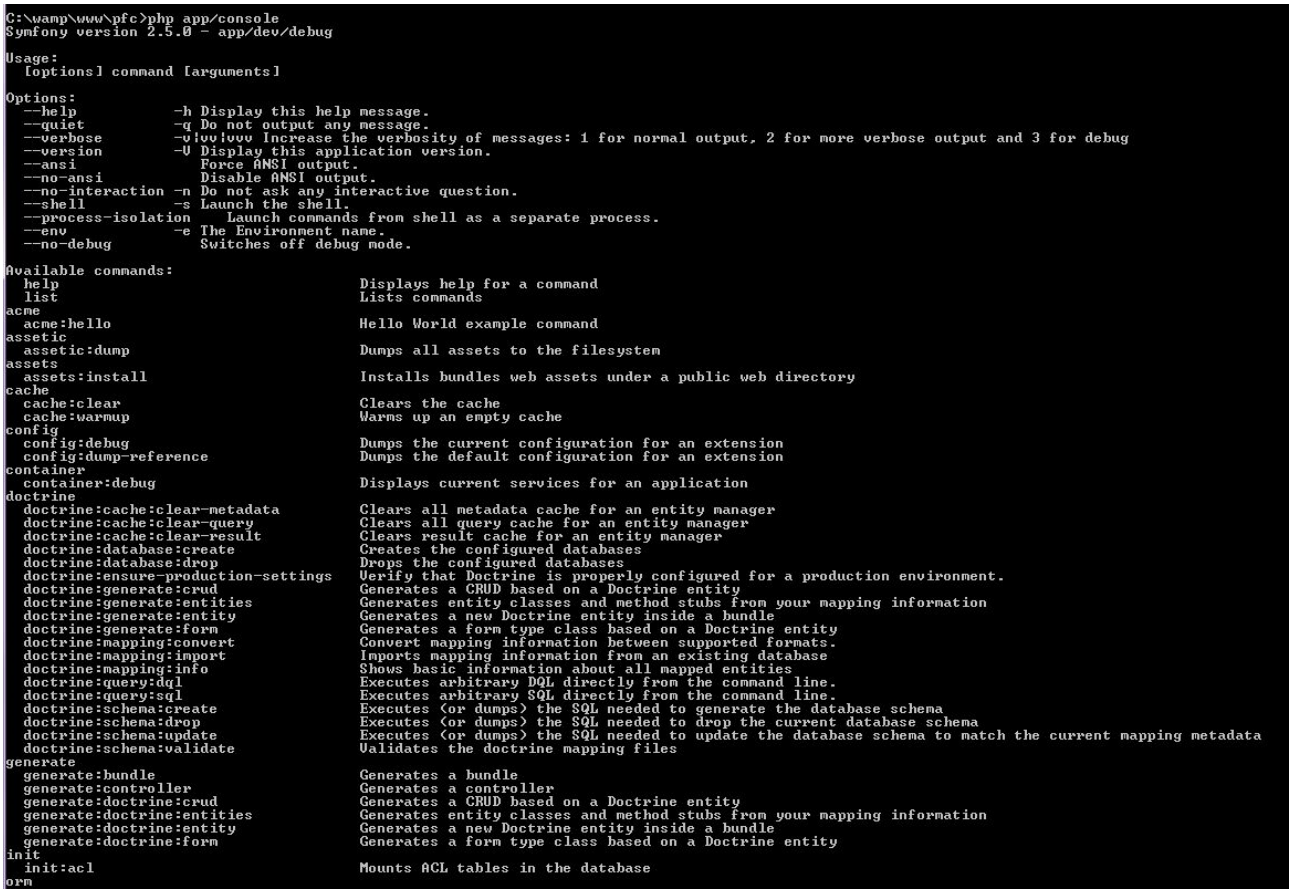
Una vez creado, para comprobar que todo ha ido bien escribimos en la consola:

```
C:\> cd C:\wamp\www\MiPFC
C:\wamp\www\MiPFC> php app/check.php
```

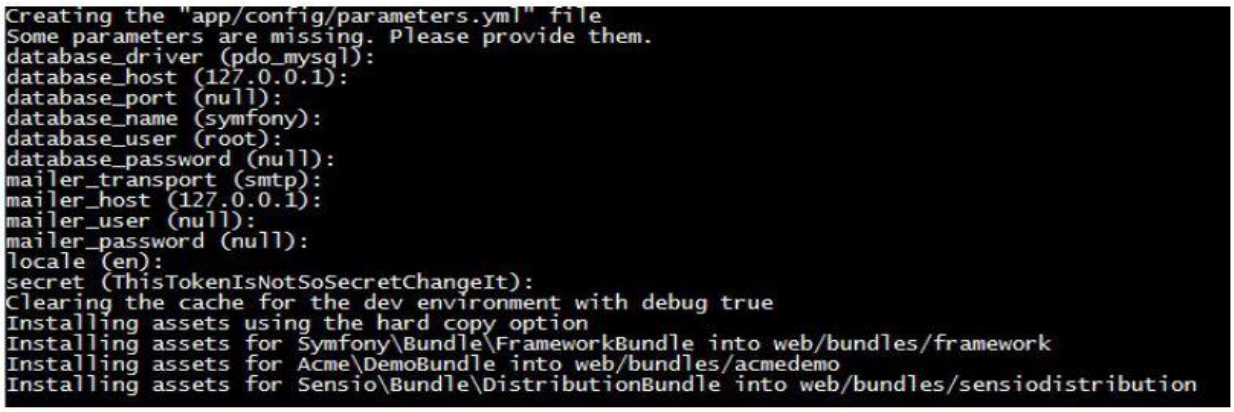
Después ejecutamos (para comprobar que symfony se ha instalado correctamente):

```
C:\wamp\www\MiPFC> php app/console
```

Si todo ha ido bien deberíamos ver una lista de comandos como esta, más completa:



Utilizando el comando composer create-project, no sólo creamos el proyecto, sino que también podemos configurar la base de datos de nuestro proyecto.
Después de haberse instalado todas las dependencias, se nos formulan unas preguntas sobre la configuración del proyecto, así como la base de datos, a la que nosotros responderemos con INTRO en todas para obtener el valor por defecto, el que está entre paréntesis, ya que lo configuraremos más adelante.



Si tras estos pasos revisamos el directorio de nuestro proyecto, veremos que se ha creado toda una estructura de ficheros y directorios.

3.-Configuración de Virtual hosts

3.1.-¿Qué es un virtual host y por qué lo vamos a utilizar?

Este término hace referencia a asignar una misma IP a sitios web con diferentes nombres y viceversa en una misma máquina.

Para nuestro proyecto hemos decidido utilizar uno para que la url de acceso al proyecto sea más simple. De esta forma, en lugar de tener que indicar la ruta exacta de acceso a nuestro proyecto, bastará con indicar "ruta del pryecto" en la barra de direcciones del navegador.

3.2.-Cómo configurar nuestro virtual host

Para este próposito deberemos editar algunos ficheros de configuración de Apache así como de Windows. Respecto a los de Apache:

En primer lugar, editaremos el contenido de C:/wamp/bin/apache/Apachev2/conf/httpd.conf descomentando la siguiente línea:

```
# Virtual hosts
Include conf/extra/httpd-vhosts.conf
```

De esta forma habilitaremos la opción de utilizar los virtual host con nuestro servidor Apache.

En el archivo C:/wamp/bin/apache/Apachev2/conf/extra/httpd-vhosts.conf añadiremos:

```
#####
## localhost
## DOMAINE principal
#####
NameVirtualHost localhost

<VirtualHost localhost>
   DocumentRoot C:/wamp/www/
   ServerName localhost
</VirtualHost>

NameVirtualHost 127.0.0.1
<VirtualHost 127.0.0.1>
   DocumentRoot "C:\wamp\www\MiPFC\web"
   DirectoryIndex app_dev.php
   ServerName proyecto.local
</VirtualHost>

<VirtualHost prueba.local>
   DocumentRoot "C:\wamp\www\MiPFC\web"
   DirectoryIndex app_dev.php
   ServerName proyecto.local
   ServerAlias www.proyecto.local
   <Directory "C:\wamp\www\MiPFC\web">
```

```
        Options Indexes FollowSymLinks Includes ExecCGI
        AllowOverride All
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

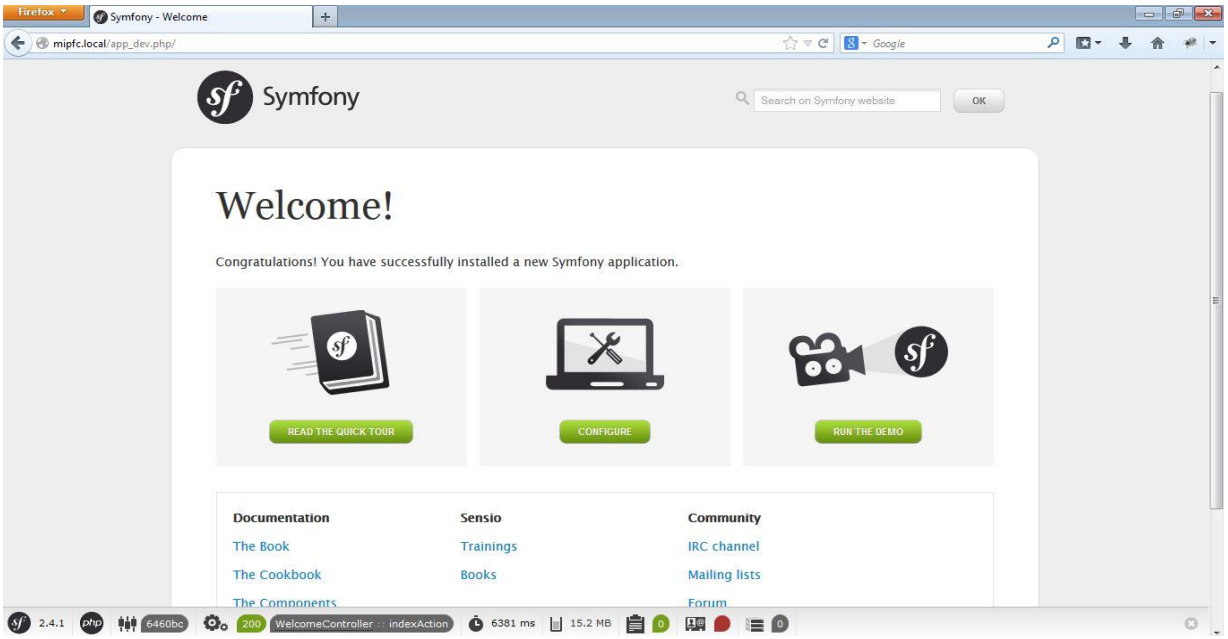
En C:\wamp\bin\php\php5.4.12\php.ini buscaremos la línea que contenga **;extension=php_openssl.dll** y la descomentaremos eliminando el **;** al pincipio de la línea. De esta forma podremos crear el proyecto desde consola utilizando composer y wamp.

En C:\wamp\www\MiPFC\web\.htaccess modificaremos las apariciones de app.php por app_dev.php. De esta forma cuando el servidor acceda al proyecto, lo hará a la ruta de la versión en desarrollo ("dev").

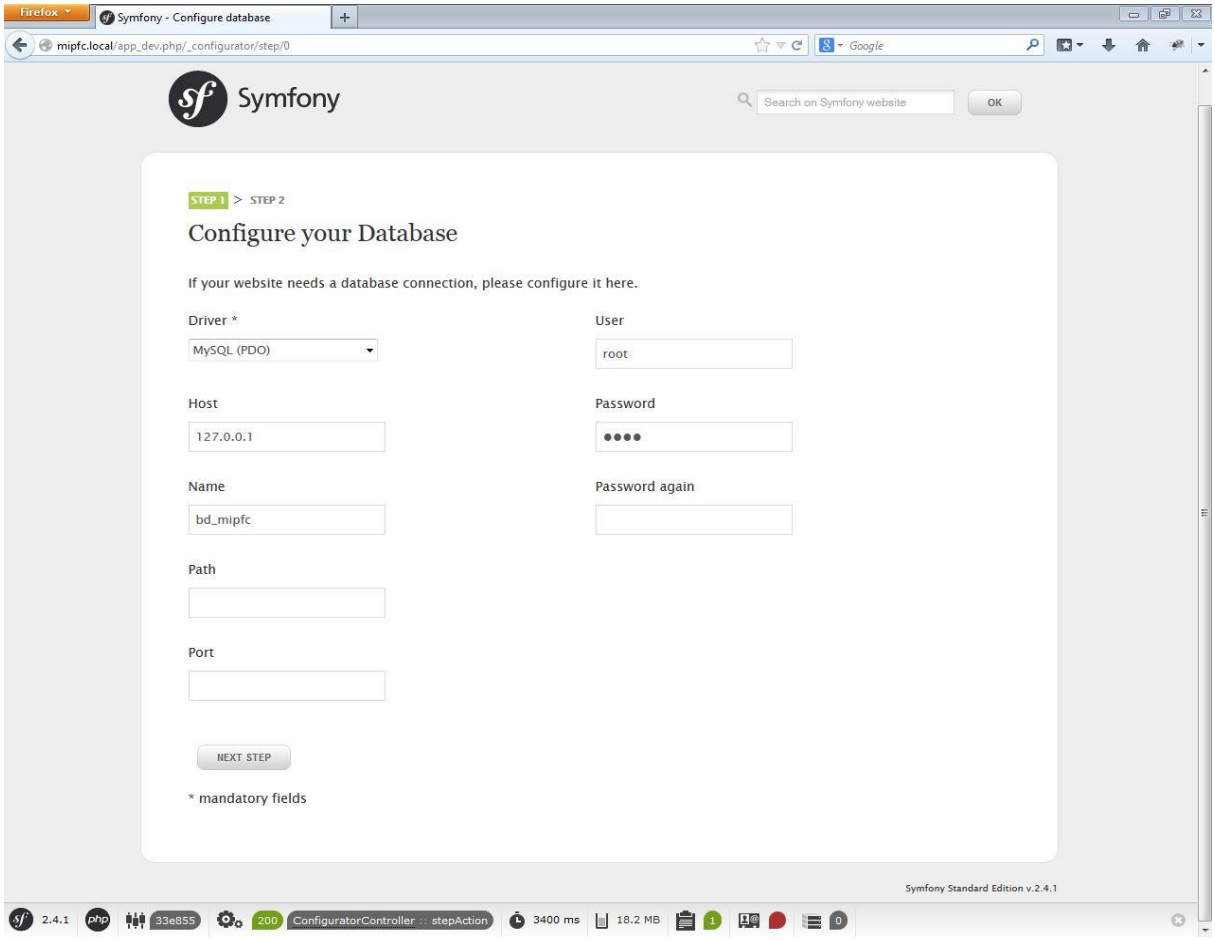
Respecto a los archivos de Windows. Editaremos el fichero C:\Windows\System32\drivers\etc\hosts, en el cual añadiremos el nombre de virtual host y dirección, en este caso:

```
        127.0.0.1   localhost
        127.0.0.1   mipfc.local
        127.0.0.1   www.mipfc.local
```

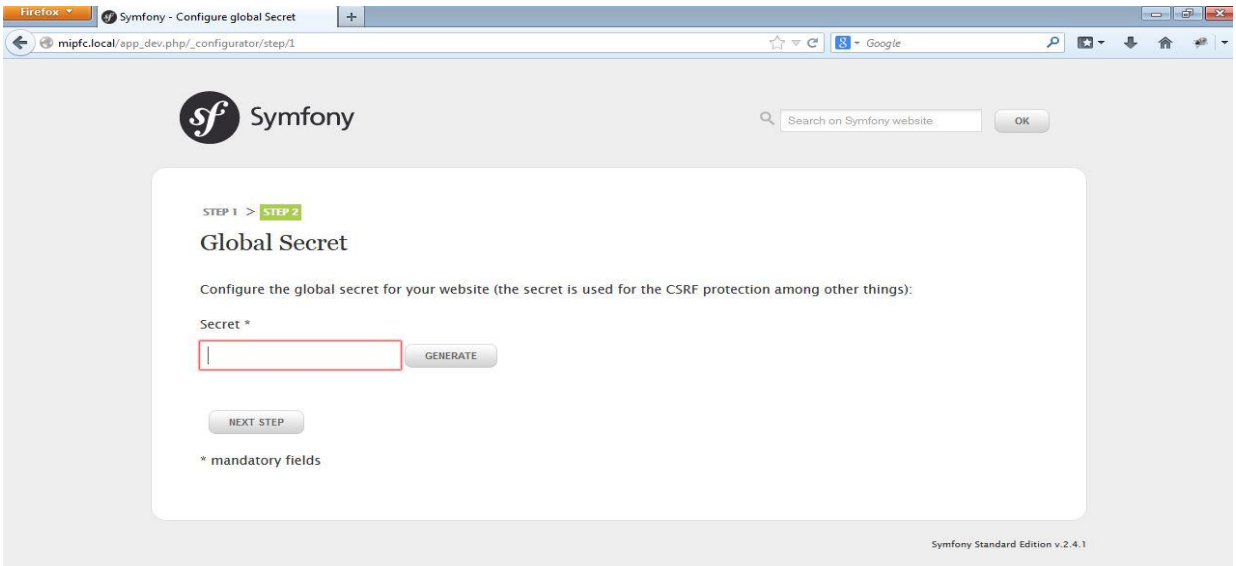
Si todo ha ido bien, al introducir mipfc.local en la barra de direcciones, nos aparecerá lo siguiente, que es la portada por defecto de Symfony2 :



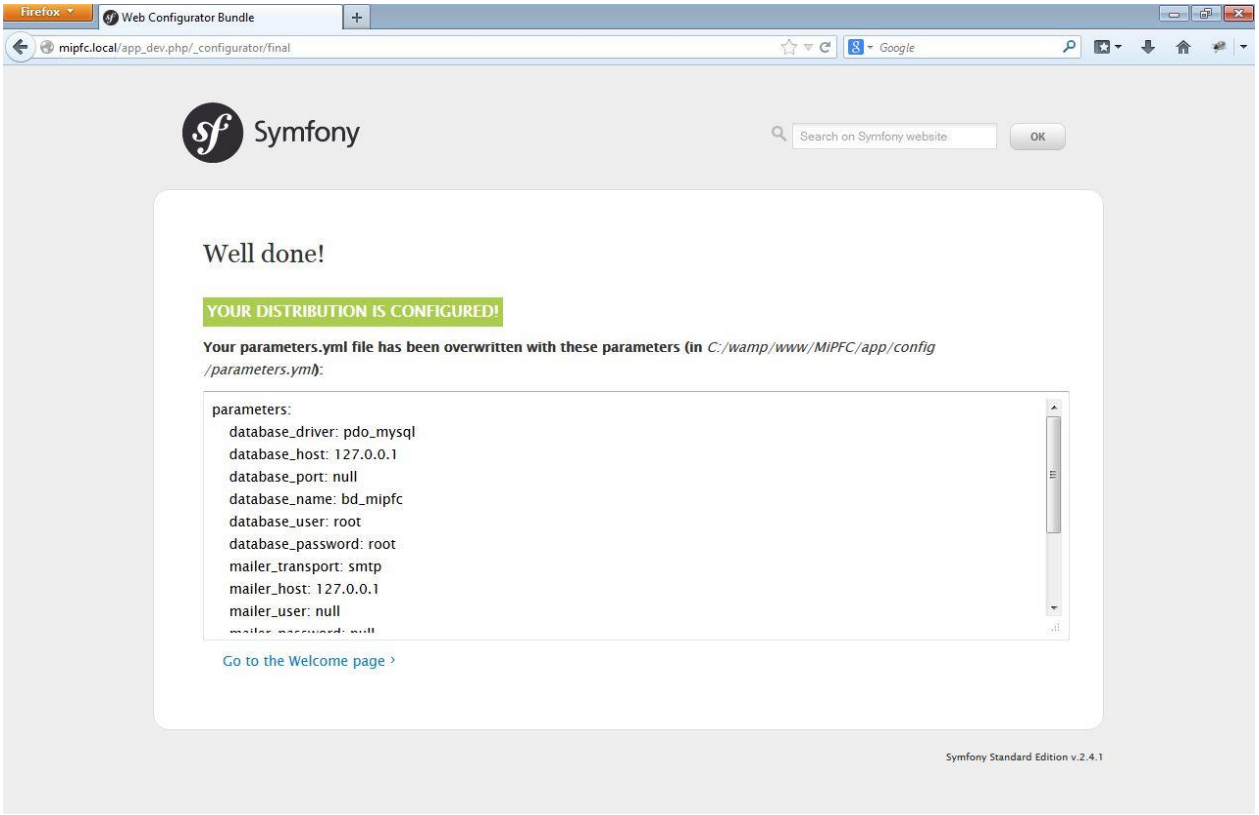
Si hacemos click en CONFIGURE, nos lleva a la siguiente página donde daremos nombre, usuario, contraseña y algunos parámetros más a nuestra base de datos:



La segunda página permite configurar el *"secreto global"*, que es una cadena de texto aleatoria utilizada en diversas partes del proyecto, como por ejemplo la protección de los formularios frente a los ataques de tipo CSRF. Pulsa varias veces el botón *GENERATE* para asegurarte de que sea un valor realmente aleatorio.



Cuando pulsemos el botón *NEXT STEP* terminaremos la configuración y obtendremos una página como esta:



La principal causa de errores al ejecutar la aplicación o los comandos de Symfony2 es la mala configuración de los permisos de los directorios app/cache/ y app/logs/, que son los únicos en los que escribe Symfony2. El problema es que en ocasiones escribe en ellos el servidor web y otras veces son los comandos de consola los que escriben.

Se deben cambiar los permisos de app/cache/ y app/logs/ para todos los usuarios a través de la pestaña *Seguridad* de las propiedades de esas carpetas. En caso de que estas carpetas no estuvieran creadas, debemos hacerlo.

La distribución estándar de Symfony2 también incluye un pequeño bundle de ejemplo llamado AcmeDemoBundle, que sirve para mostrar algunas de las buenas prácticas en el desarrollo con Symfony2. Después de probarlo y jugar con su código, se recomienda borrarlo. Para eliminar este bundle y todos sus rastros:

- Elimina el directorio src/Acme
- Quita las rutas _welcome, _demo y _demo_secured del archivo app/config/routing_dev.yml
- Borra el registro del bundle en el archivo app/AppKernel.php (quita la línea

$bundles[] = new Acme\DemoBundle\AcmeDemoBundle();)

4.-Creación de bundles

Un Bundle es básicamente una carpeta que contiene los archivos necesarios para un grupo de funcionalidades específicas, como por ejemplo un blog, un carrito de compras o incluso parte del frontend y backend de nuestra aplicación. La idea es que debería ser trasladable a otro proyecto y poder ser reutilizado.

Una aplicación en Symfony2 podrá contener todos los Bundles que queramos y necesitemos, simplemente debemos crearlos y registrarlos. Los Bundles que nosotros creemos deberán ir dentro de la carpeta src\ del proyecto mientras que los Bundles de terceros deberán ir dentro de la carpeta vendor\. Cómo hacer esta utilización de bundles de terceros se explicará más adelante.

Un Bundle tiene una estructura de carpetas y archivos definidos y un nombre identificador dentro de nuestro proyecto, este nombre lo utilizaremos para hacer referencia al mismo. Lo ideal es no guardar directamente los bundles dentro src\ sino dentro de una carpeta que represente al proyecto, empresa o a nosotros a la cual llamamos paquete, esto a fin de que si alguien más crea un Bundle con el mismo nombre no se confunda con el nuestro.

En nuestro proyecto vamos a utilizar 3 Bundles. Aquí explicaremos como generar el primero, UsuarioBundle.

Para no crearlos a mano usaremos una utilidad de Symfony llamada “console”. Abriremos un terminal de windows, cmd, y entraremos al directorio de nuestro proyecto con el siguiente comando:

```
C:\> cd wamp\www\MiPFC
```

Ejecutemos en el cmd lo siguiente:

```
C:\wamp\www\MiPFC> php app/console generate:bundle
```

Con este comando se ejecutará un generador que nos formulará cinco preguntas, como se muestra a continuación:

1. *Bundle namespace*, es el namespace bajo el que se creará el bundle y por tanto, determina el directorio en el que se guarda el código. El namespace se compone de tres partes: la primera es el nombre del proyecto o empresa que desarrolla el bundle (MiPFC en este caso), la segunda parte es opcional e incluye una o más categorías en las que se clasifica el bundle (casi siempre se deja vacío) y la última parte es el propio

nombre del bundle, que se puede elegir libremente pero siempre debe acabar en Bundle.
Respuesta: MiPFC/UsuarioBundle.

2. *Bundle name*, es el nombre con el que se hará referencia al bundle dentro de la aplicación. Se recomienda que el nombre sea simplemente el namespace sin barras separadoras (en este caso, MiPFCUsuarioBundle). No obstante, como el nombre de los bundles se escribe cientos de veces en el código de la aplicación, se recomienda que si los bundles son sólo para uso interno de tu aplicación, se utilice un nombre lo más corto posible, eliminando la primera palabra del bundle: UsuarioBundle.

3. *Target directory*, indica la ruta donde crear el bundle, debes aceptar el valor por defecto que se muestra, que coincide con el directorio src/ del proyecto. Respuesta recomendada: pulsar ENTER para elegir el valor por defecto.

4. *Configuration format*, determina el formato utilizado en los archivos de configuración del bundle. Puedes elegir entre YAML, XML, PHP o las anotaciones. Respuesta recomendada:
yml

5. *Do you want to generate the whole directory structure?*, por defecto Symfony2 genera una estructura básica de directorios. Respuesta recomendada: no

6. Después de contestar a esas preguntas, el comando muestra un resumen de tus respuestas y te pregunta si realmente quieres generar ese bundle. Si estás de acuerdo, contesta yes a la pregunta *Do you confirm generation?.*

7. *Confirm automatic update of your Kernel?*, contesta yes para que el bundle se active en la aplicación después de generarlo.

8. *Confirm automatic update of the Routing?*, contesta también yes para que el archivo de
enrutamiento del bundle se cargue automáticamente desde el archivo de enrutamiento
general de la aplicación.

Después de haber generado el bundle, se habrá creado una estructura como esta:

```
src/
└— MiPFC/
└— UsuarioBundle/
├— Usuario.php
├— Controller/
│  └— DefaultController.php
├— DependencyInjection/
│  ├— Configuration.php
│  └— OfertaExtension.php
├— Resources/
│  ├— config/
│  │  ├— routing.yml
│  │  └— services.yml
│  └— views/
│  └— Default/
│  └— index.html.twig
└— Tests/
└— Controller/
└— DefaultControllerTest.php
```

Para poder utilizar un bundle en una aplicación Symfony2, además de crearlo, se necesita activarlo. La activación se realiza en la clase más importante de Symfony, llamada AppKernel.php.

Cada vez que un usuario realiza una petición a un sitio web creado con Symfony, se instancia una clase para crear la aplicación Symfony2 que se encarga de responder a la petición del usuario.

Para activar el bundle UsuarioBundle, añade la siguiente línea al final del array $bundles del método registerBundles():

```
// app/AppKernel.php
<?php
use Symfony\Component\HttpKernel\Kernel;
use Symfony\Component\Config\Loader\LoaderInterface;

class AppKernel extends Kernel
{
        public function registerBundles()
        {
                $bundles = array(
                        new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
                        new Symfony\Bundle\SecurityBundle\SecurityBundle(),
                        new Symfony\Bundle\TwigBundle\TwigBundle(),
                        // ...
                        new MiPFC\UsuarioBundle\UsuarioBundle(),
                );

                // ...
        }
}
```

Cuando se genera el bundle con el comando generate:bundle esta línea de código se añade automáticamente. Pero si se instalan bundles desarrollados por terceros, hay que añadir esa línea manualmente.

5.-Generación de rutas

En primer lugar explicaremos brevemente qué son las URLs amigables y porque utilizarlas.

Con la llegada de los lenguajes de servidor (PHP, ASP…), las URL se tornaron complejas, del estilo de:

http://www.dominio.com/index.php?id=23&order=asc&item=32&cat=33

Los factores técnicos para indicar los datos a extraer de una base de datos y para seleccionar el tipo de página a mostrar tomaron el control frente a las hasta entonces ***URL***s, por defecto amigables sin necesidad de premeditarlas demasiado. Con ***URL***s de este tipo, la dirección web ya no aporta nada sobre el contenido de la página: los motores de búsqueda no saben qué es ese id23, ni esa sarta de parámetros colocados tras ellos. Los buscadores quieren direcciones interesantes, como:

http://www.dominio.com/usuario/proyecto

Para ayudar a esto, podemos hacer que cada vez que accedamos a algún fichero de nuestros bundles mediante la URL, se muestre la ruta que queramos para nuestro bundle. Con el objetivo de conseguir esto y una vez creado nuestro bundle, nos al fichero routing.yml (ubicado en \MiPFC\app\config\routing.yml), y añadimos el siguiente código:

```
usuario:
  resource: "@UsuarioBundle/Resources/config/routing.yml"
  prefix:   /Usuario
```

De esta forma, cada vez que se quiera generar una nueva ruta (o URL) se creará en el archivo routing.yml indicado y además tendrá el prefijo /Usuario.

6.-Creación de entidades

6.1.- ¿Qué es una entidad?

En la arquitectura MVC(Modelo, Vista, Controlador), los modelos son la representación en código de las tablas de las que se compone nuestro esquema de base de datos.
Para el framework Symfony, los modelos reciben el nombre de entidades. Éstas se componen de atributos que representan las características de los objetos y algunos métodos para tratar dichas características.

En este apartado trataremos la creación de entidades, así como la forma de establecer las relaciones entre éstas.

6.2.-¿Cómo crear una entidad?

En primer lugar, decir que la creación de entidades puede realizarse por consola o

manualmente. Para nuestro caso, nos serviremos de la primera opción.

Abrimos la consola y nos dirigiremos al directorio de nuestro proyecto, una vez allí ejecutaremos el siguiente comando:

```
C:\wamp\www\MiPFC> php app/console doctrine:generate:entity
```

Tras ejecutar esta orden, deberemos responder a una serie de preguntas que ayudan a la creación automática de la entidad. El proceso a seguir es el siguiente:

```
This command helps you generate Doctrine2 entities.

First, you need to give the entity name you want to generate.
You must use the shortcut notation like AcmeBlogBundle:Post.

The Entity shortcut name: PFCBundle:Proyecto
```

El *Entity shortcut name* es el nombre corto de la entidad, que se forma concatenando el nombre del *bundle* en el que se encuentra y el nombre de la clase de la entidad. Para generar una entidad Proyecto.php en el *bundle* PFCBundle, indica el nombre corto PFCBundle:Proyecto.

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

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

Las anotaciones son el formato recomendado para definir las entidades, pero Doctrine2 también permite definir esta información en archivos XML, YAML y PHP. Para seleccionar las anotaciones como formato, simplemente pulsa Enter. Después de indicar esta información básica, el comando solicita una por una la información de todas las propiedades. No es necesario crear la propiedad $id que actúa de clave primaria de la entidad porque el comando la genera automáticamente. Así que vamos a empezar por la primera propiedad de Proyecto, que es $titulo:

```
New field name (press <return> to stop adding fields): titulo
Field type [string]: <Enter>
Field length [255]: <Enter>
```

En primer lugar indica el nombre de la propiedad (en este caso, titulo). Después, hay que indicar el tipo de información que se guarda en la propiedad. Como por defecto el tipo es string, se puede pulsar Enter para seleccionarlo. Dependiendo del tipo de dato seleccionado, el comando puede plantear más preguntas. Para el tipo string por ejemplo se pide la longitud máxima de la cadena de texto. Para seleccionar el valor por defecto 255, pulsaremos Enter.

A continuación se crean el resto de propiedades de la entidad:

```
New field name (press <return> to stop adding fields): listaAlumnos
Field type [string]: <Enter>
Field length [255]: <Enter>

New field name (press <return> to stop adding fields): fechaInicio
Field type [string]: datetime

New field name (press <return> to stop adding fields): fechaPresentacion
Field type [string]: datetime

New field name (press <return> to stop adding fields): fechaAltaSistema
Field type [string]: datetime

New field name (press <return> to stop adding fields): profesor
Field type [string]: <Enter>
Field length [255]: <Enter>

New field name (press <return> to stop adding fields): listaHitos
Field type [string]: <Enter>
Field length [255]: <Enter>

New field name (press <return> to stop adding fields): descripcion
Field type [string]: <Enter>
Field length [255]: <Enter>

New field name (press <return> to stop adding fields): informacionAdicional
Field type [string]: <Enter>
Field length [255]: <Enter>

New field name (press <return> to stop adding fields): estado
Field type [string]: boolean

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

El comando doctrine:generate:entity no permite indicar el nombre de una entidad como tipo de dato (como los que necesitan las propiedades $listaAlumnos, $profesor y $listaHitos). En este caso, simplemente utilizamos el tipo string y se corrige después a

mano el código generado.

Para terminar de añadir propiedades, pulsaremos Enter como nombre de la propiedad. A continuación el comando pregunta si queremos generar un repositorio vacío. Contestaremos que no, ya que nuestro repositorio ya está creado (tal y como hemos explicado en la sección de cómo trabajar con Git):

```
Do you want to generate an empty repository class [no]? No

Summary before generation

You are going to generate a "PFCBundle:Proyecto" Doctrine2 entity
using the "annotation" format.

Do you confirm generation [yes]? yes
```

La entidad ya ha sido generada en el archivo src/MiPFC/PFCBundle/Entity/Proyecto.php. Si el directorio Entity/ no existía, el comando también lo genera. A continuación podemos ver las partes fundamentales de la entidad que se ha generado:

```php
use Doctrine\ORM\Mapping as ORM;

/**
 * Proyecto
 *
 * @ORM\Table()
 * @ORM\Entity
 */
class Proyecto
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

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

```

Cabecera y declaración del id y de un atributo de tipo String

```php
    /**
     * @var \DateTime
     *
     * @ORM\Column(name="fechaInicio", type="datetime")
     */
    private $fechaInicio;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="fechaPresentacion", type="datetime")
     */
    private $fechaPresentacion;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="fechaAltaSistema", type="datetime")
     */
    private $fechaAltaSistema;

    /**
     * @var Profesor
     *
     * @ORM\ManyToOne(targetEntity="PFC\UsuarioBundle\Entity\Profesor")
     */
    private $profesor;
```

Declaración de atributos de tipo fecha y tipos propios como Profesor

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

    /**
     * Set titulo
     *
     * @param string $titulo
     * @return Proyecto
     */
    public function setTitulo($titulo)
    {
        $this->titulo = $titulo;

        return $this;
    }

    /**
     * Get titulo
     *
     * @return string
     */
    public function getTitulo()
    {
        return $this->titulo;
    }
```

Getters y setters de cada atributo

```
220     /**
221      * Set profesor
222      *
223      * @param Profesor $profesor
224      * @return Proyecto
225      */
226     public function setProfesor(\PFC\UsuarioBundle\Entity\Profesor $profesor)
227     {
228         $this->profesor = $profesor;
229 
230         return $this;
231     }
232 
233     /**
234      * Get profesor
235      *
236      * @return Profesor
237      */
238     public function getProfesor()
239     {
240         return $this->profesor;
241     }
```

Declaración de los métodos get y set para el tipo Profesor

Antes de dar por concluida la entidad, corregiremos las propiedades $listaAlumnos, $profesor y $listaHitos:

```
/**
    * @var string
    *
    * @ORM\ManyToOne(targetEntity="MiPFC\UsuarioBundle\Entity\Alumno")
    */
   private $listaAlumnos;

...

   /**
    * Set listaAlumnos
    *
    * @param Alumno $listaAlumnos
    * @return Proyecto
    */
   public function setListaAlumnos(\MiPFC\UsuarioBundle\Entity\Alumno $listaAlumnos)
   {
      $this->listaAlumnos = $listaAlumnos;

      return $this;
   }

   /**
    * Get listaAlumnos
    *
    * @return Alumno
    */
   public function getListaAlumnos()
   {
      return $this->listaAlumnos;
   }
```

Para establecer la relación entre tablas hemos realizado tres cambios (marcados en verde):

1. Modificar el tipo de los valores que reciben/devuelven los métodos get y set. Antes el valor era de tipo String, mientras que ahora se ha especificado un tipo no primitivo, ya que el valor debía ser un objeto de los cuales se compone el sistema.

2. Modificar las propiedades de la variable a nivel global. Si nos fijamos en el comentario que acompaña a la variable veremos que incluye una etiqueta ***ManyToOne***. Esta etiqueta tiene como objetivo indicar a Doctrine que esta variable está relacionada con una entidad existente en el sistema. Dicha entidad se especifica pasándola como si se tratara de un parámetro a esta etiqueta.

3. Indicar a los métodos set el tipo de parámetro que recibirá. A pesar de que en PHP no es necesario indicar a qué tipo pertenece una variable pasada como parámetro, en Symfony los métodos set que hacen referencia a otro objeto sí deben incluirlo.

7.-Creación de la base de datos

7.1.-Mapeo

En los proyectos Symfony las bases de datos se gestionan mediante Doctrine, que básicamente es un mapeador de objetos relacional para PHP (consultar anexo para más información).

El mapeo de la base de datos se va a realizar siguiendo el sistema de anotaciones. Para lo cual indicaremos en cada entidad una serie de especificaciones que junto con una orden de consola, nos generará el esquema de la base de datos. Además debemos incluir una referencia al uso de Doctrine.

Si queremos que alguna de nuestras entidades se refleje en nuestra base de datos lo haremos de la siguiente forma:

```
3  namespace PFC\ProjectBundle\Entity;
4
5  use Doctrine\ORM\Mapping as ORM;
6
7  /**
8   * Proyecto
9   *
10  * @ORM\Table()
11  * @ORM\Entity
12  */
13 class Proyecto
14 {
```

En la imagen podemos ver la inclusión de Doctrine y unas anotaciones encima del nombre de la clase. Éstas indican que correspondencia deberá tener esta clase en la base de datos.

Para generar cada uno de los atributos de la entidad, las anotaciones deberán indicar el tipo de datos que contendrá cada atributo:

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

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

Si nos fijamos en el primer campo, las anotaciones no sólo hacen referencia al tipo de datos sino también a las características especiales del campo. En este caso se indica que el valor del atributo “id” se generará automáticamente. Por otro lado la anotación para el campo título indica el nombre que dicho campo tendrá en la base de datos, el tipo y ya que se trata de un string, la longitud máxima de este.

En caso de querer representar otros formatos como fechas utilizaríamos la siguiente notación:

```
    /**
     * @var Alumno
     *
     * @ORM\ManyToOne(targetEntity="PFC\UsuarioBundle\Entity\Alumno")
     */
    private $listaAlumnos;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="fechaInicio", type="datetime")
     */
    private $fechaInicio;
```

En la imagen podemos apreciar la creación de un campo con un tipo propiamente definido por el programador. Doctrine permite mediante la anotación hacer referencia al tipo de clase concreto que necesitemos. Este tipo de mapeo sirve para representar el concepto de claves foráneas en los modelos de bases de datos relacionales. Es importante tener bien claro el modelo de datos, ya que a la hora de definir claves foráneas en una entidad de Symfony, sólo es necesario expresar la relación en un sentido.

Una vez definidas todas las anotaciones sobre nuestras clases sólo debemos realizar dos pasos más para tener nuestra base de datos:

1.-Crear una nueva BD utilizando la siguiente instrucción, la cual toma los parámetros que definimos al crear nuestra aplicación y genera un modelo:

php app/console doctrine:database:create

2.-Ejecutar el siguiente comando:

php app/console doctrine:schema:update –force

Esta instrucción es de las más poderosas de Doctrine. Su función es la de comparar cada una de las anotaciones de nuestras entidades y en caso de que exista alguna diferencia con el modelo en la base de datos, actualiza dicho modelo, independientemente del cambio que sea.

7.2.-Generación de la relación de herencia

La herencia de tablas permite al programador crear tablas de base de datos que heredan de otras tablas, de la misma forma que las clases pueden heredar de otras clases en los lenguajes de programación orientados a objetos. La herencia de tablas es una forma sencilla de que dos o más tablas compartan información en una única tabla padre.

```php
/**
 * Usuario
 *
 * @ORM\Table()
 * @ORM\Entity
 * @ORM\InheritanceType("SINGLE_TABLE")
 * @ORM\DiscriminatorColumn(name="discr", type="string")
 * @ORM\DiscriminatorMap({"alumno"="Alumno", "profesor"="Profesor"})
 */
class Usuario
{
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    protected $id;
```

Para llevar a cabo esta estrategia se ha utilizado la técnica de herencia de una sola tabla (Single-Table Inheritance), la cual es una estrategia de mapeo donde todas las clases de una jerarquía se mapean en una sola tabla de una base de datos. Para distinguir qué filas representan a qué tabla en la jerarquía se usa una columna llamada discriminador.

En la imagen podemos ver que se ha definido una anotación para indicar la columna que actuará como discriminante y otra que indica el valor debe tener ese discriminante. De esta forma podemos almacenar en una sola tabla los distintos tipos de usuarios. Además las entidades hijas deben indicar que extienden, en este caso, de Usuario.

8.-Creación de vistas

8.1.-Uso de Twig

En una aplicación MVC las vistas son devueltas por los controladores. En el caso de nuestra aplicación Symfony se han utilizado plantillas Twig.

Twig es un motor de plantillas para PHP que permite utilizar conjuntamente código HTML y algunas estructuras de flujo de control.

A continuación se muestra un ejemplo:

```
<div class="col-lg-10">
    <label>Nombre</label> {{profesor.nombre}}<br>
    <label>Apellidos</label> {{profesor.apellido1}}<br>
    <label>Departamento</label> {{profesor.departamento.
    nombre}}<br>
</div>
```

En la imagen podemos ver que Twig utiliza la notación de dobles llaves para utilizar variables. Estas variables son las que se definen en el controlador junto con la vista al procesar el renderizado de la plantilla. Es decir que en nuestro ejemplo concreto hemos enviado un objeto profesor y estamos mostrando su nombre, apellido1 y el nombre del departamento al que pertenece. Respecto a este último caso cabe destacar la potencia de Twig a la hora de mostrar información de objetos relacionados con otros. Puesto que un profesor tiene un objeto de tipo Departamento, gracias a Twig sólo necesitamos devolver todo el objeto profesor obtenido en el controlador, y

una vez en la plantilla podemos acceder a los atributos de ese departamento tal y como se muestra en la imagen.

Por último destacar que el nombre de los archivos debe tener la siguiente nomenclatura:

fichaProfesor.html.twig

8.2.-Generando la plantilla base y sus derivadas

Dentro del directorio app/Resources/views (en caso de no existir deberemos crearlo) debemos crear una plantilla base. El principal motivo es debido a que la mayoría de las vistas de nuestra aplicación utilizan las mismas partes. Gracias a Twig podemos abstraer estas partes en una sola plantilla y extender el resto de ésta. De esta forma evitamos repetir trabajo gracias al sistema de herencia.

En nuestro caso completo nuestra plantilla base será la siguiente:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}
            {% stylesheets '@bootstrap_css' filter='cssrewrite' %}
                <link rel="stylesheet" href="{{ asset_url }}" />
            {% endstylesheets %}
        {% endblock %}
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
        {% block javascripts %}
            <script type="text/javascript" src="{{ asset('bundles/comun/js/jquery-1.11.1.min.js') }}"></script>
            {% javascripts '@bootstrap' %}
                <script src="{{ asset_url }}"></script>
            {% endjavascripts %}
        {% endblock %}
    </head>
    <body class="container">
        {% block body %}{% endblock %}
    </body>
</html>
```

Tal y como se muestra, la plantilla incluye el código HTML básico de la cabecera que se repetirá a lo largo de todos nuestras vistas. También podemos apreciar la definición de bloques, como por ejemplo {% block body %}{% endblock %}, con esto lo que conseguimos es que cuando otra plantilla herede de Base.html.twig podamos definir donde se va a sobreescribir el código. Es decir, si quiero que mi nueva vista tenga las mismas cabeceras que su padre pero quiero que el body sea distinto, bastaría con

reescribir todo el bloque body dentro de {% block body %}{% endblock %} en la vista hija.

Para hacer referencia a una plantilla padre simplemente basta con hacer esto:

```
{% extends '::base.html.twig' %}
    {% block title %}Listado mensajes profesor{% endblock %}
{% block body %}
```

8.3.-Uso de instrucciones de flujo en Twig

Entendemos por instrucciones de flujo las sentencias como “for”, “if”, etc. Twig nos permite utilizar este tipo de instrucciones. Su uso es realmente sencillo, basta con ver el siguiente ejemplo:

```
<form class="" method="POST" name="pepito" {{ form_enctype(formulario) }}>
    <div class="form-group row">
        <div class="col-lg-4">
            <label class="control-label">Remitente</label>
            <select class="form-control" id="user-receptor" name="receptor">
                {% for usuario in usuarios %}
                    {% if (usuario.id != user.id) %}
                        <option value="{{usuario.id}}">{{usuario.nombre}}</
                        option>
                    {% endif %}
                {% endfor %}
            </select>
        </div>
```

De esta forma dotamos a los diseñadores de más flexibilidad y resulta más rápido e intuitivo generar vistas.

9.-Implementación de un CRUD

9.1.-Estrategia a llevar a cabo

CRUD es el acrónimo de Create, Read, Update y Delete. Vamos a ver cómo realizar estas operaciones en nuestra aplicación. Muchas aplicaciones de mayor profundidad suelen utilizar los controladores para obtener los datos del modelo y unas clases intermedias llamadas servicios para gestionar la lógica con estos datos. En nuestro caso realizaremos ambas cosas en el controlador, pero la utilización de esta capa de servicio consiste en crear unas clases específicas y luego incluirlas en los controladores.

9.2.-Estructura y creación del controlador

Al crear la aplicación se nos crea automáticamente dentro de cada Bundle un directorio llamado Controller que contiene un archivo llamado “DefaultController.php”.

```php
<?php

namespace PFC\ProjectBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use PFC\ProjectBundle\Entity\Proyecto;
use PFC\UsuarioBundle\Entity\Alumno;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    public function indexAction($name)
    {
        return $this->render('ProjectBundle:Default:index.html.twig', array('name' => $name));
    }

```

En la imagen podemos ver como se define por defecto un controlador. En él vemos un método llamado indexAction. Cuando creamos un método este debe seguir la nomenclatura *Action, de esta forma esos métodos estarán disponibles para poder ser asignados al definir una URL.

9.3.-Create

Para crear nuevos registros lo primero que debemos hacer es definir una nueva instancia de la clase a insertar y obtener el entity manager de Doctrine (líneas 19 y 20). Este gestor nos sirve para interactuar con el modelo.

Lo siguiente que hemos hecho es crear una instancia de formulario para referenciar los datos y así saber qué valor corresponde con cada campo (líneas 22 a la 36, en los anexos explicaremos la gestión de formularios).

Si prestamos atención a la línea 39, podemos ver la forma alternativa de asignar valor a los campos de nuestro objeto sin necesidad de formularios.

Para insertar nuestro registro en la base de datos nos serviremos del método persist() que nos proporciona el entity manager. Este método detecta automáticamente el tipo de objeto que estamos pasándole e inserta en la tabla correspondiente. Para que estos datos se persistan en la base de datos debemos utilizar el método flush().

Por último devolvemos la vista correspondiente si todo ha ido bien.

```
17  public function nuevoProyectoAction(Request $request)
18  {
19      $proyecto = new Proyecto();
20      $em = $this->getDoctrine()->getManager();
21
22      $formulario = $this->createFormBuilder($proyecto)
23          ->add('titulo')
24          ->add('fechaInicio', 'date')
25          ->add('fechaPresentacion', 'date')
26          ->add('profesor')
27          ->add('descripcion', 'textarea')
28          ->add('informacionAdicional', 'textarea')
29          ->add('estado', 'choice', array(
30              'choices'   => array('0' => 'Inactivo', '1' => 'Activo'),
31              'required'  => false,
32              )
33          )
34          ->getForm();
35
36      $formulario->handleRequest($request);
37
38      if($formulario->isValid()){
39          $proyecto->setFechaAltaSistema(new \DateTime());
40          $em->persist($proyecto);
41          $em->flush();
42
43          return $this->redirect($this->generateUrl('ficha_profesor', array('id' => $proyecto->getProfesor()->getId())));
44      }
45
```

9.4.-Read

Esta operación también utiliza el entity manager. Tras obtenerlo utilizaremos el método getRepository() de éste. Este método recibe como parámetro un String que contiene el Bundle y el modelo del cual queremos leer, utilizando la siguiente notación : “NombreBundle:Entidad”. En la línea 126 observamos que una vez seleccionado el repositorio se llama al método que hace de selector, es decir el método al cual le pasaremos el parámetro por el que queramos buscar. Tras esta instrucción ya tendríamos nuestro registro leído.

```
122 public function fichaProyectoAction(Request $request)
123 {
124     $em = $this->getDoctrine()->getManager();
125
126     $proyecto = $em->getRepository("ProjectBundle:Proyecto")->findOneById($request->get('id_proyecto'));
127     $hitos = $em->getRepository("ProjectBundle:ProjectHito")->findBy(array('proyecto' => $proyecto->getId()));
128
129     return $this->render('ProjectBundle:Default:fichaProyecto.html.twig', array(
130                 'proyecto' => $proyecto,
131                 'hitos' => $hitos
132                 )
133             );
134 }
```

9.5.-Update

La operación es una mezcla entre Create y Read. Invocaremos al entity manager, tras lo cual leeremos el registro a editar (línea 81). Seguidamente nos ayudaremos de un formulario para referenciar los campos del objeto, y por último haremos el persist() y el flush() como si de un Create se tratara (líneas 100 y 101).

```php
public function editarProyectoAction(Request $request)
{
    $em = $this->getDoctrine()->getManager();
    $proyecto = $em->getRepository("ProjectBundle:Proyecto")->findOneBy(array('id' => $request->get('id')));

    $formulario = $this->createFormBuilder($proyecto)
        ->add('titulo')
        ->add('fechaInicio', 'date')
        ->add('fechaPresentacion', 'date')
        ->add('profesor')
        ->add('descripcion', 'textarea')
        ->add('informacionAdicional', 'textarea')
        ->add('estado', 'choice', array(
           'choices'   => array('0' => 'Inactivo', '1' => 'Activo'),
           'required'  => false,
           )
        )
        ->getForm();

    $formulario->handleRequest($request);

    if($formulario->isValid()){
        $em->persist($proyecto);
        $em->flush();

        return $this->redirect($this->generateUrl('index_profesor', array('id' => $proyecto->getProfesor()->getId())));
    }


    return $this->render('ProjectBundle:Default:editarProyecto.html.twig', array('formulario' => $formulario->createView()))
}
```

9.6.-Delete

Para eliminar registros nos serviremos del método remove() del entity manager de Doctrine. En primer lugar obtenemos el objeto que queramos eliminar, de la misma manera que en el Read (línea 81).

Tras lo cual invocamos al método remove() pasándole el objeto a eliminar. Cabe destacar que no haces ningún tipo de referencia a la integridad referencial porque los métodos internamente se encargan de gestionarlo. En caso de que alguna operación viole la integridad, la operación no se realiza y la aplicación lanzará una excepción. Por último llamamos a persist().

```php
public function eliminarProyectoAction(Request $request)
{
    $em = $this->getDoctrine()->getManager();
    $proyecto = $em->getRepository("ProjectBundle:Proyecto")->findOneBy(array('id' => $request->get('id')));

    $em->remove($proyecto);
    $em->flush();

    return $this->redirect($this->generateUrl('index_profesor', array('id' => $proyecto->getProfesor()->getId())));
}

```

VI.-CONCLUSIONES Y TRABAJO FUTURO

El proceso de desarrollo de un software es algo en constante evolución. Es común el hecho de mejorar un producto. A continuación se presenta una lista de mejoras que se podrían aplicar:

- Refactorización del modelo de datos: se podrían unir las dos clases que sirven para representar el Inbox de los usuarios en una sola. Bastaría con hacer una sola clase con un discriminante para distinguir si son mensajes de entrada o salida.
- Interfaces de usuario más prácticas: mejora que consistiría en incluir más vistas en una pantalla. Por ejemplo en las vistas utilizadas para listar elementos se podría dar la opción de editar campos de los elementos en esa propia lista. De esta forma se reducirían considerablemente los clicks que debe hacer el usuario en la aplicación.
- Control exhaustivo de errores: a nivel de código se deberían tratar los posibles errores mediante sentencias “try catch”, de esta forma controlaríamos los tipos de errores y evitaríamos flujos de acción inesperados por parte de la aplicación.
- Inclusión de un CMS: para tener un mayor control sobre la aplicación, se podría incluir un gestor de contenido para un usuario Administrador. De esta forma se podrían realizar más acciones cómo dar de alta Departamentos o titulaciones, o incluso tener un mayor sobre los usuarios registrados en la aplicación.

VII.-ANEXOS

1.-Incluir librerías/plugins de terceros

A continuación vamos a explicar cómo podemos incluir plugins o librerías desarrollados por terceros para poderlos utilizar en nuestra aplicación Symfony.

Una de las cosas que mayor versatilidad y funcionalidad aporta a nuestros proyectos es el uso de librerías y plugins. Todo proyecto en Symfony2 permite esta acción. En nuestro caso particular vamos a explicar cómo incluir dos librerías: JQuery y Bootstrap.

1.1.-¿Qué son?

Los plugins/librerías son extensiones de código que realizan una determinada funcionalidad. Pueden ser utilizados en cualquier aplicación, siempre y cuando dichas extensiones sean compaibles con el lenguaje de nuestro código. La mayoría de los plugins no requiere de otros para poder ser incluídos en un proyecto, no obstante existen extensiones. Debido al buen hacer de las distintas comunidades de desarrolladores, estas librerías suelen venir documentadas, probadas y perfectamente versionadas. Podemos encontrarlas en sitios web como Github, Bitbucket, etc.

1.2.-JQuery y Bootstrap

JQuery es una librería Javascript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

Bootstrap por su parte es un framework para desarrollo frontend que proporciona un interfaz gráfico vistoso así como dotar a un proyecto de las herramientas necesarias para que éste sea Responsive.

1.3.-Instalación

En primer lugar debemos incluír JQuery, ya que Bootstrap necesita de esta librería para funcionar. Para ello seguiremos los siguientes pasos:

1.-Dentro de nuestra carpeta \PFC\src\PFC\ComunBundleResources crearemos una carpeta llamada "Public". Esta carpeta es muy importante ya que sin ella el framework no podrá reconocer las librerías.

2.-Dentro de esta nueva carpeta creamos otra llamada "js", en ella introduciremos el código.

3.-Descargar JQuery desde su sitio web http://http://jquery.com/download (descargaremos la versión del enlace indicado como "Download the compressed, production jQuery 1.11.1") y guardarlo en la carpeta que acabamos de crear.

4.-En nuestro archivo \PFC\app\Resources\views\base.html.twig incluíremos la siguiente línea:

```
13          {% block javascripts %}
14              <script type="text/javascript" src="{{ asset('bundles/comun/js/jquery-1.11.1.min.js') }}"></script>
15              {% javascripts '@bootstrap' %}
16                  <script src="{{ asset_url }}"></script>
17              {% endjavascripts %}
18          {% endblock %}
```

5.-Tras esto nos vamos a la consola y en el directorio del proyecto ejecutamos el siguiente comando:

```
php app/console assets:install
```

Tal y como se ve en la imagen adjunta, el framework ha reconocido nuestros archivos.

```
C:\wamp\www\Symfony\PFC>php app/console assets:install
Installing assets using the hard copy option
Installing assets for Symfony\Bundle\FrameworkBundle into web/bundles/framework
Installing assets for PFC\ComunBundle into web/bundles/comun
Installing assets for Acme\DemoBundle into web/bundles/acmedemo
Installing assets for Sensio\Bundle\DistributionBundle into web/bundles/sensiodi
stribution
```

Si ahora inspeccionásemos el código fuente de la página veríamos que la librería aparece.

Una vez instalado JQuery vamos a añadir Bootstrap:

1.-Para la instalación de Bootstrap vamos a utilizar composer. Como se ha explicado anteriormente es un gestor de dependencias. En este caso nos permitirá añadir los ficheros necesarios modificando un fichero de configuración.

2.-Abrimos el archivo PFC\composer.json y añadimos la línea ""twitter/bootstrap": "3.1.1"" al final del apartado "require".

3.-Una vez hecha esta modificación ejecutamos en nuestra consola el comando "composer update", dentro del directorio de nuestro proyecto. Esto provocará que en nuestra carpeta vendors se añada una nueva llamada "Twitter", si la inspecionamos veremos el contenido que se ha descargado mediante composer.

4.-Debido a que Bootstrap se compone tanto de ficheros .css como .js, la forma de incluír su código será ligeramente diferente a la de JQuery. En este caso

abriremos el archivo PFC\app\config\config.yml y aquí definiremos una serie de variables globales para hacer referencia a los ficheros que debemos incluír. Para ello en la sección "assetics" de dicho fichero añadiremos el siguiente contenido:

```
assets:
    bootstrap:
        inputs:
            - '%kernel.root_dir%/../vendor/twitter/bootstrap/dist/js/bootstrap.js'
    bootstrap_css:
        inputs:
            - '%kernel.root_dir%/../vendor/twitter/bootstrap/dist/css/bootstrap.css'
```

NOTA: los nombres bootstrap y bootstrap_css son definidos por el propio usuario, por lo que podrían ser cualesquiera. Si nos fijamos en las dos rutas que hemos definido vemos que presentan una vriable del propio framework para definir el path de éste seguida del path de la carpeta vendor donde se encuentra el código referenciar.

5.-Como último paso nos vamos a \PFC\app\Resources\views\base.html.twig y añadiremos, respectivamente, dentro de los bloques stylesheets y javascript los siguientes fragmentos de código:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1, user-scalable=no">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}
            {% stylesheets '@bootstrap_css' filter='cssrewrite' %}
                <link rel="stylesheet" href="{{ asset_url }}" />
            {% endstylesheets %}
        {% endblock %}
        <link rel="icon" type="image/x-icon" href="{{ asset('favicon.ico') }}" />
        {% block javascripts %}
            <script type="text/javascript" src="{{ asset('bundles/comun/js/jquery-1.11.1.min.js') }}"></script>
            {% javascripts '@bootstrap' %}
                <script src="{{ asset_url }}"></script>
            {% endjavascripts %}
        {% endblock %}
    </head>
```

NOTA SOBRE EL CÓDIGO JAVASCRIPT: Como se ha dicho anteriormente, es importante que JQuery esté cargado antes que bootstrap para que éste pueda funcionar. Para ello antes de cargar el bloque de javascripts se debe llamar a JQuery como hemos explicado antes.

6.-Tras esto ejecutamos el comando php app/console assets:install. El resultado final de nuestro archivo base.html.twig sería el que hemos visto en la imahen superior.

2.-Utilización de formularios

Symfony permite definir los formularios en el backend de la aplicación. Esto supone que podemos especificar de qué campos estará formado nuestro formulario, las restricciones de éstos, así como el tipo de datos que van a contener.

```
19      $proyecto = new Proyecto();
20      $em = $this->getDoctrine()->getManager();
21
22      $formulario = $this->createFormBuilder($proyecto)
23          ->add('titulo')
24          ->add('fechaInicio', 'date')
25          ->add('fechaPresentacion', 'date')
26          ->add('profesor')
27          ->add('descripcion', 'textarea')
28          ->add('informacionAdicional', 'textarea')
29          ->add('estado', 'choice', array(
30              'choices'   => array('0' => 'Inactivo', '1' => 'Activo'),
31              'required'  => false,
32              )
33          )
34          ->getForm();
35
36      $formulario->handleRequest($request);
37
38      if($formulario->isValid()){
39          $proyecto->setFechaAltaSistema(new \DateTime());
40          $em->persist($proyecto);
41          $em->flush();
42
43          return $this->redirect($this->generateUrl('ficha_profesor', array('id' => $proyecto->getProfesor()->getId())));
44      }
45
```

En la imagen adjunta podemos ver como mediante el método createFormBuilder() vamos añadiendo los campos de nuestro formulario. Este método recibe un objeto del tipo del formulario que deseemos crear. De esta forma los campos tendrán las mismas características que tengan en su definición. Para tener este formulario disponible en la vista sólo tenemos que pasarlo en el método como si se tratara de una variable más.

```
return $this->render('ProjectBundle:Default:nuevoProyecto.html.twig', array('formulario' => $formulario->createView()));
```

Tras esto en nuestra plantilla Twig realizaremos el renderizado del formulario tal y como se indica a continuación:

```
6               <form class="form-horizontal" method="POST"  {{ form_enctype(formulario) }}>
7                   <fieldset>
8                       <legend>Información básica</legend>
9                       <div class="form-group row">
10                          <div class="col-lg-6">
11                              <label class="control-label">Título</label>
12                              {{ form_widget(formulario.titulo, {'attr': {'class': 'form-control'} }) }}
13                          </div>
14                      </div>
15                      <div class="form-inline">
16                          <div class="form-group col-lg-4">
17                              <label class="control-label">Inicio</label>
18                              <div class="">
19                                  {{ form_widget(formulario.fechaInicio, {'attr': {'class': 'form-control'
                                    } }) }}
20                              </div>
21                          </div>
```

En la línea 6 observamos que al declarar la etiqueta del formulario debemos llamar al método form_enctype() de Twig para que podamos visualizar los campos con su formato correspondiente. Además cada campo debe ser declarado dentro del método form_widget(), el cual acepta como primer parámetro el campo del formulario a renderizar y como segundo las características que deseemos que tenga,

entendiéndose por características aquellos atributos de HTML que tendría el input que se va a generar. Este segundo parámetro debe estar en formato JSON.

Además de estos campos se debe incluir el método form_rest(formulario), el cual es necesario para que cuando enviemos la información del formulario ésta se envíe toda. Esto es debido a que Symfony utiliza campos ocultos como un token de seguridad
Que todos los formularios llevan. Este campo es de funcionamiento a nivel interno y no debemos preocuparnos de declararlo ni gestionarlo, ya que la aplicación lo trata internamente.

3.-Novedades de la versión 2.4

Muchos artículos que encontramos en internet hacen referencia a la versión 2.3 de Symfony. Para este proyecto hemos utilizado la 2.4, a continuación se enumeran una serie de novedades que respecta dicho versión. Cabe destacar que el paso de 2.3 a 2.4 supone un proceso de retrocompatiblidad perfecta, por tanto el código debería funcionar perfectamente.

- Se ha añadido un nuevo componente llamado ExpressionLanguage (ver documentación)
- Se ha introducido un nuevo servicio llamado request_stack y que reemplaza al servicio request. A partir de esta versión, se desaconseja el uso del objeto Request en los servicios y se recomienda utilizar en su lugar el objeto RequestStack (detalles).
- Las plantillas Twig ahora permiten medir en detalle el tiempo que tardan en renderizar cada parte (detalles)
- Ahora es mucho más fácil definir tus propios ***user providers*** y ***authenticators*** (detalles) y también puedes asociar firewalls a hosts (detalles).
- Los logs de la aplicación ahora se pueden mostrar en la consola (detalles) y también se han añadido muchas pequeñas mejoras a la consola (detalles).
- Ahora es posible detener un proceso si lleva demasiado tiempo inactivo (detalles).
- Se ha añadido un panel de depuración de formularios en la barra de depuración (detalles) y se ha mejorado bastante el validador de imágenes (detalles).

4.-Información acerca de Doctrine

Es una librería de código abierto disponible en github. Una de las mejores definiciones acerca de esta librería sería la de Wikipedia: Doctrine 1.x se basa en el active record pattern para trabajar con datos, en los que una clase se corresponde con una tabla de base de datos. Por ejemplo, si un programador quiere crear un nuevo objeto "Usuario" en la base de datos, no tendrá que escribir ninguna sentencia **SQL**.

Una característica de Doctrine es el bajo nivel de configuración que necesita para empezar un proyecto. Doctrine puede generar clases a partir de una base de datos existente y después el programador puede especificar relaciones y añadir funcionalidad extra a las clases autogeneradas. No es necesario generar o mantener complejos esquemas XML de base de datos como en otros frameworks.

Otra característica importante de Doctrine es la posibilidad de escribir consultas de base de datos utilizando un dialecto de SQL denominado **DQL** (Doctrine Query Language) que está inspirado en Hibernate (Java).

Otras características notables de Doctrine son:

- Soporte para datos jerárquicos;
- Soporte para hooks (métodos que pueden validar o modificar las escrituras y lecturas de la base de datos) y eventos para manejar la lógica de negocio relacionada;
- Herencia;
- Un framework de caché que utiliza diversos motores como memcached, SQLite o APC;
- Transacciones ACID;
- Diversos comportamientos del modelo (conjuntos anidados, internacionalización, log, índice de búsqueda);
- Una función "compilar" que combina varios archivos PHP del framework en uno solo para evitar el descenso de rendimiento que provoca incluir varios archivos PHP.

5.-Sobre las tecnologías utilizadas

5.1.-JAVASCRIPT

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo

que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems.

-librosweb

5.2.-CSS

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

-librosweb

5.3.- jQuery

jQuery es una librería de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. jQuery es la librería de JavaScript más utilizada.

Es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos.

jQuery, al igual que otras librerías, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta librería se logran grandes resultados en menos tiempo y

espacio.

-wikipedia

5.4.-PHP

PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente en interpretación del lado del servidor (server-side scripting) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.
Es un acrónimo de "PHP: Hypertext Preprocessor", la mayor parte de su sintaxis es similar a C, Java y Perl, y es fácil de aprender. La meta de este lenguaje es permitir escribir a los creadores de páginas web, páginas dinámicas de una manera rápida y fácil, aunque se pueda hacer mucho más con PHP.
Fue creado originalmente por Rasmus Lerdorf en 1994; sin embargo la implementación principal de PHP es producida ahora por The PHP Group y sirve como el estándar de facto para PHP al no haber una especificación formal.
Este lenguaje es uno de los más populares en la actualidad. Ej: WordPress, Wikipedia, vBulletin, Facebook...

PHP no devuelve solamente código HTML al navegador, entre muchas de sus funciones también puede crear imágenes, generar miniaturas de las mismas, crear archivos PDF en base a contenido definido o crear archivos de tipo Flash en el instante y bajo demanda, entre muchas cosas más, todo depende del uso adecuado de librerías y el conocimiento del lenguaje.
En lo que se refiere a datos almacenados, PHP soporta una gran cantidad de bases de datos para interactuar con la información, son más de veinte distintas a las que podemos ingresar datos o extraer, entre las que se encuentran:
Oracle
Informix
MySQL
MS SQL Server
Sybase y muchas otras.

VII.-BIBLIOGRAFÍA Y WEBS CONSULTADAS

- **http://www.juntadeandalucia.es**
- **http://www.desarrolloweb.com**
- **http://www.maestrosdelweb.com**
- **http://symfony.es**
- **http://symfony.com**
- **http://showmethecode.es**
- **http://es.wikipedia.org**
- **Libro Desarrollo web ágil con Symfony2 de Javier Eguiluz**