

I.E.S ABASTOS



Desarrollo gestión de proyectos

Autor: **Jaime Aguilá Sánchez**
Ciclo Desarrollo de Aplicaciones Web
Memoria del Proyecto de DAW
IES Abastos. Curso 2015/16. Grupo 7S. 27 de Mayo de 2016
Tutor individual: Heike Bonilla

Non nova, sed nove.

Contenido

| | |
|---|----|
| 1. Introducción | 1 |
| 1.1. DEFINICIÓN DEL PROYECTO | 1 |
| 2. Objetivos y planificación | 2 |
| 2.1. OBJETIVOS | 2 |
| 2.2. PLANIFICACIÓN | 3 |
| 3. Diseño | 4 |
| 3.1. REQUISITOS MÍNIMOS SERVIDOR | 4 |
| 3.2. REQUISITOS MÍNIMOS DE LOS CLIENTES | 4 |
| 3.3. DIAGRAMA DE CASOS DE USO | 4 |
| 3.3.1. Usuario | 4 |
| 3.3.2. Alumno | 5 |
| 3.3.3. Profesor | 5 |
| 3.3.4. Administrador | 5 |
| 3.4. DIAGRAMA DE CLASES | 6 |
| 3.4.1. Diagrama de clases Usuarios, UsuariosRoles y Roles | 6 |
| 3.4.2. Diagrama de clases Auxiliares. | 6 |
| 3.4.3. Diagrama de clases Proyectos | 7 |
| 3.5. MOCKUPS | 9 |
| 4. Tecnologías y Herramientas | 10 |
| 4.1. PHP | 10 |
| 4.2. MVC SOBRE PHP | 10 |
| 4.3. PHPUNIT | 11 |
| 4.4. MYSQL | 11 |
| 4.5. IIS | 11 |
| 4.6. JQUERY | 12 |
| 4.7. FRAMEWORK EASYUI | 12 |
| 4.8. CONTROL DE VERSIONES | 12 |
| 4.8.1. Git | 13 |
| 4.8.2. GitHub | 13 |
| 5. Implementación | 14 |
| 5.1. METODOLOGÍA | 14 |
| 5.1.1. Metodología mixta | 14 |
| 5.1.2. TDD | 16 |
| 5.1.3. Refactorización | 17 |
| 5.2. PATRONES DE DISEÑO | 18 |
| 5.2.1. El patrón Singleton | 18 |
| 5.2.2. El patrón FrontController | 19 |
| 5.2.3. El patrón MVC . | 19 |
| 5.3. FUNCIONAMIENTO WEB | 24 |
| 5.3.1. Get - Indice | 24 |
| 5.3.2. Post - ObtenerDatos | 24 |

Desarrollo gestión de proyectos

| | |
|---|----|
| 5.3.3. <i>Get - Crear</i> | 24 |
| 5.3.4. <i>Post - Crear</i> | 24 |
| 5.3.5. <i>Get - Editar</i> | 24 |
| 5.3.6. <i>Post - Editar</i> | 24 |
| 5.3.7. <i>Post - Deleteld</i> | 24 |
| 5.4. ACCESO A ARCHIVOS | 25 |
| 5.5. SEGURIDAD | 26 |
| 5.5.1. <i>Control de acceso.</i> | 26 |
| 5.5.2. <i>Verbos HTTP</i> | 26 |
| 5.5.3. <i>Filtrado de parámetros.</i> | 26 |
| 5.5.4. <i>XSS</i> | 27 |
| 5.5.5. <i>Inyección SQL</i> | 27 |
| 5.5.6. <i>CSRF – Cross Site Request Forgery</i> | 27 |
| 5.6. URL FRIENDY | 28 |
| 5.8. GUI | 29 |
| 6. Resultados y conclusiones | 30 |
| 6.1. ESTADO ACTUAL | 30 |
| 6.2. CONCLUSIONES | 30 |
| 6.3. POSIBILIDADES FUTURAS | 30 |
| 6.4. VALORACIÓN PERSONAL | 31 |
| 7. Anexos | 32 |
| A. FRONTCONTROLLER | 32 |
| B. PLANTILLA.PHP | 33 |
| C. LISTADO DE MÓDULOS | 34 |
| D. Sección rewrite de web.Config | 35 |
| 8. Referencias y bibliografía | 36 |

Tabla de ilustraciones

| | |
|--|----|
| 1 Diagrama de Gantt..... | 3 |
| 2 Diagrama de casos de uso | 4 |
| 3 Diagrama de clases | 6 |
| 5 Diseño previo pantalla principal | 9 |
| 4 Diseño previo pantalla autenticación..... | 9 |
| 6 Diseño previo pantalla alta usuarios | 9 |
| 7 Diseño previo pantalla Listado usuarios..... | 9 |
| 8 Patrón MVC | 10 |
| 9 Apartado de Issues de Github..... | 13 |
| 10 Metodologías Agiles frente a tradicionales | 14 |
| 11 Ciclo TDD Rojo-Verde-Refactorizar | 16 |
| 12 Identificar calidad del código | 17 |
| 13 Implementacion patrón singleton | 19 |
| 14 Instanciación y llamada al método de acción del controlador final | 19 |
| 15 Estructura carpetas MVC | 20 |
| 16 Código básico de una acción de un controlador..... | 21 |
| 17 Código básico de un modelo..... | 21 |
| 18 Código del motor de vistas..... | 22 |
| 19 Código de un ejemplo de vista | 22 |
| 20 Código ObtenArchivo.php | 25 |
| 21 Pantalla principal | 29 |
| 22 Pantalla listado usuarios | 29 |
| 23 Pantalla autenticación | 29 |

1. Introducción

1.1. Definición del proyecto

Este proyecto surge como una propuesta de la profesora Heike Bonilla, dado que los proyectos de fin de ciclo se almacenan en cajas y no existe un sistema informático para clasificarlos y permitir acceder a sus datos, surgió la posibilidad de realizar una aplicación web para gestionar los proyectos de fin de ciclo.

Decidimos realizarlo en MVC sobre PHP, para poner en practica todo lo aprendido en el ciclo, y para acceder a la base de datos realizarlo por PDO parametrizando las consultas, además queríamos ver en más profundidad el uso de plantillas.

Para ello, decidimos ponernos unos límites, no usar ningún plugin de plantillas, para tener que ver que soporte nos daba PHP, para realizar nosotros el trabajo.

Y el segundo limite, y más importante, hay muchísimos frameworks de MVC en PHP, que son muy buenos, como **CodeIgniter**, **Laravel**, **CakePHP**, **Symfony**, **Yii**, pero decidimos no usar ninguno, queríamos aprender a realizar el proceso por nuestra cuenta, y no depender de un **framework**, que te lo da todo hecho, requiere mucho tiempo de aprendizaje y te ata a un producto en particular, además al ser tan completos suelen hacer muchas más cosas de las que necesitamos y suelen encontrarse agujeros de seguridad, obligándote a estar siempre pendiente del framework.

2. Objetivos y planificación

2.1. Objetivos

El objetivo principal es la creación de una aplicación web que nos permita almacenar los proyectos de fin de ciclo y acceder a sus datos, tanto desde los proyectos en si, como desde los alumnos.

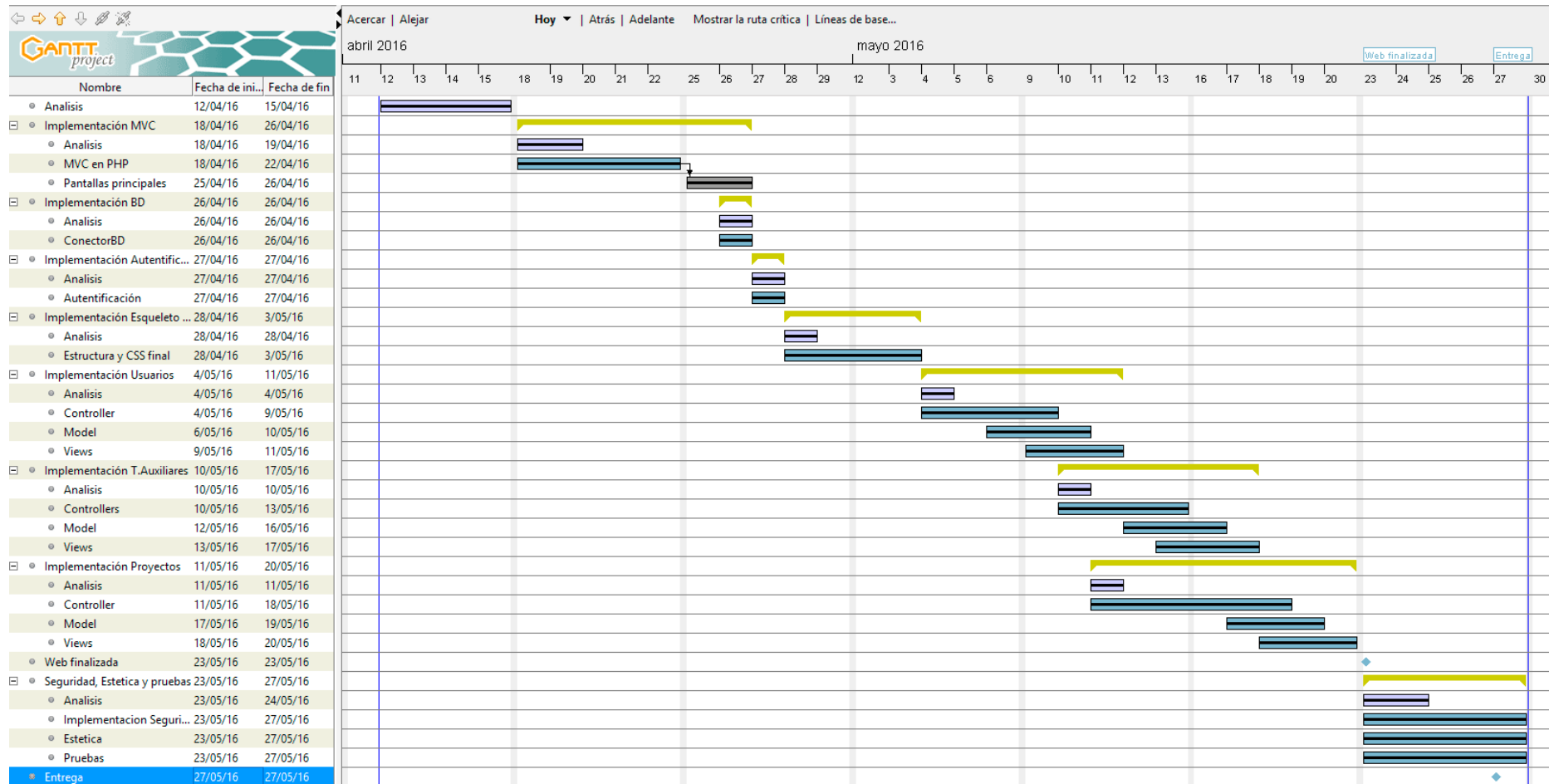
Específicamente la aplicación deberá tener las siguientes secciones:

- Un panel de entrada, que nos permita autenticarnos
- El acceso de tres roles distintos, administradores, profesores y alumnos
- Permitir dar de alta alumnos y profesores
- Permitir dar de alta y modificar los datos de un proyecto
- Permitirnos realizar búsquedas sobre los proyectos
- Realizar nuestro propio MVC en PHP.
- Aprender a usar o crear algún tipo de plantillas.

A nivel personal, con el proyecto se busca:

- Obtener el título de Técnico en Desarrollo de Aplicaciones Web.
- Aprender MVC sobre PHP
- Profundizar en la seguridad de las aplicaciones Web
- Obtener una aplicación web que nos permita mayor flexibilidad a la hora de gestionar los proyectos de fin de ciclo.

2.2. Planificación



1 Diagrama de Gantt

3. Diseño

3.1. Requisitos mínimos servidor

Para poder instalar la aplicación en el servidor tendremos unos requisitos mínimos, el proyecto se ha realizado sobre Windows con IIS, aunque se puede utilizar sin problemas Apache realizando unos mínimos cambios.

- Windows Server o Linux.
- Servidor web, IIS o Apache (el proyecto está desarrollado sobre IIS)
- Tener instalado MySQL
- Acceso a la maquina desde internet, a ser posible con un DNS fijo.

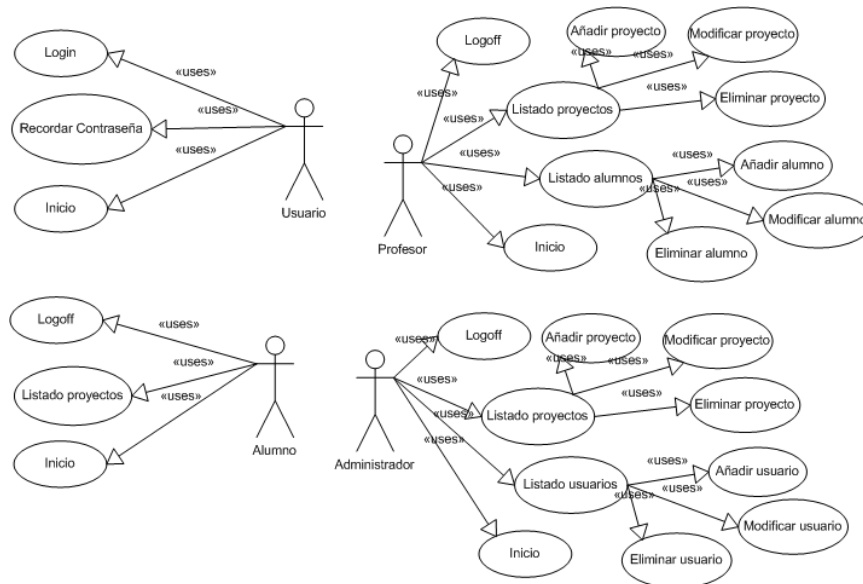
3.2. Requisitos mínimos de los clientes

Para que los clientes se puedan conectar a la aplicación web, se requiere lo siguiente:

- *Acceso a internet*
- *Un navegador moderno con JavaScript activado.*

3.3. Diagrama de casos de uso

En la aplicación existen tres roles diferenciados, los alumnos, los profesores y los administradores.



2 Diagrama de casos de uso

3.3.1. Usuario

Un usuario es cualquier persona que accede a la aplicación, antes de autenticarse en esta. Sus acciones posibles son las siguientes:

- **Login**, que le permitirá autenticarse en la aplicación.
- **Recordar contraseña**, para poder establecer la nueva contraseña a través de correo
- **Inicio**, será la pantalla principal de presentación.

3.3.2. Alumno

Los alumnos son el rol con menos privilegios del sistema. Las acciones posibles que pueden realizar son las siguientes:

- **Logoff**, que le permitirá salir de la aplicación.
- **Inicio**, será la pantalla principal de presentación.
- **Listado de proyectos**, que le mostrara todos los proyectos que hay en el sistema.

3.3.3. Profesor

Los profesores son el rol principal de la aplicación, y serán los que gestionen los proyectos. Las acciones posibles que pueden realizar son las siguientes:

- **Logoff**, que le permitirá salir de la aplicación.
- **Inicio**, será la pantalla principal de presentación.
- **Listado de proyectos**, que le mostrara todos los proyectos que hay en el sistema. Y desde aquí además podrán:
 - Añadir proyecto
 - Modificar proyecto
 - Eliminar proyecto
- **Listado de alumnos**, que le mostrara todos los alumnos que hay en el sistema. Y desde aquí además podrán:
 - Añadir alumno
 - Modificar alumno
 - Eliminar alumno

3.3.4. Administrador

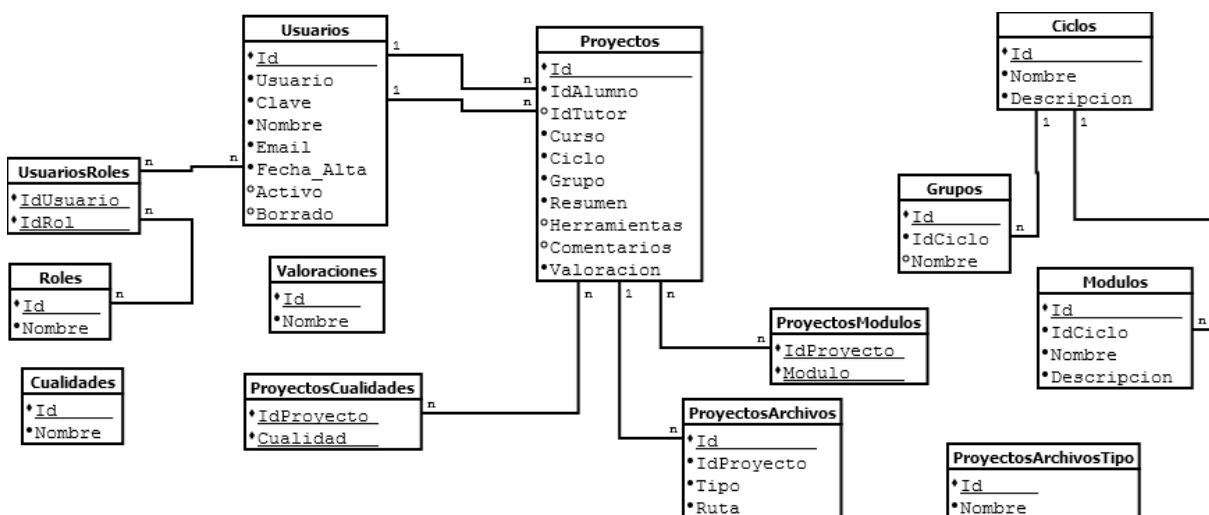
Los administradores, podrán hacer exactamente lo mismo que los profesores, con la única salvedad de que en vez de alumnos, accederán a usuarios que les permite añadir, modificar o eliminar usuarios, los usuarios podrán ser alumnos, profesores o administradores.

Las acciones posibles que pueden realizar son las siguientes:

- **Logoff**, que le permitirá salir de la aplicación.
- **Inicio**, será la pantalla principal de presentación.
- **Listado de proyectos**, que le mostrara todos los proyectos que hay en el sistema. Y desde aquí además podrán:
 - Añadir proyecto
 - Modificar proyecto
 - Eliminar proyecto
- **Listado de usuarios**, que le mostrara todos los usuarios que hay en el sistema. Y desde aquí además podrán:
 - Añadir usuarios
 - Modificar usuarios
 - Eliminar usuarios

3.4. Diagrama de clases

La base de datos de la aplicación se dividirá en tres bloques, las tablas relacionadas con los usuarios, las tablas auxiliares y las tablas de los proyectos, a continuación mostramos todo el esquema de la base de datos.



3 Diagrama de clases

3.4.1. Diagrama de clases Usuarios, UsuariosRoles y Roles

La tabla de Usuarios contendrá los datos personales y será la que se usará para autenticar a un usuario en la aplicación. Al mismo tiempo estos usuarios tendrán unos roles asignados mediante la tabla **UsuariosRoles**, que serán los que distinguirán al tipo de usuario en si (**Alumno, Profesor o Administrador**) hemos decidido permitir que un mismo usuarios pueda tener diversos roles, para dar mayor libertad a la aplicación, pudiendo un usuario ser tanto Administrador como Profesor, o incluso ser Alumno.

3.4.2. Diagrama de clases Auxiliares.

Serán todas las tablas auxiliares del sistema, los usuarios con rol administrador serán los encargados de mantenerlas y los únicos que podrán acceder a sus mantenimientos.

3.4.2.1. Diagrama de clases Ciclos

Almacena las siglas del ciclo y la descripción de este, por defecto ya viene cargada con los ciclos actuales.

- ASIR - Técnico Superior en Administración de Sistemas Informáticos en Red
- DAM - Técnico Superior en Desarrollo de Aplicaciones Multiplataforma
- DAW - Técnico Superior en Desarrollo de Aplicaciones Web

3.4.2.2. Diagrama de clases Grupos

Almacena los grupos relacionándolos con los ciclos, por defecto ya viene cargada con los grupos actuales.

- ASIR - 7L, 7M, 7P
- DAM – 7J, 7N, 7U
- DAW – 7K, 7S

3.4.2.3. Diagrama de clases Módulos

Almacena las siglas del módulo y la descripción de este, por defecto ya viene cargada con los módulos actuales. En el **Listado de módulos** aparecen la lista de módulos.

3.4.2.4. Diagrama de clases ProyectoArchivosTipo

Almacena el nombre para la lista de tipos de archivo asociados a los proyectos, por defecto ya viene cargado con los siguientes.

- Documento de memoria
- Video
- Presentación
- Código Fuente
- Otros

3.4.2.5. Diagrama de clases Cualidades

Almacena el nombre para la lista de cualidades que se pueden asociar a un proyecto, por defecto ya viene cargado con los siguientes.

- Actualidad
- Internet
- Seguridad
- Web
- Rendimiento
- Calidad
- Estetica

3.4.2.6. Diagrama de clases Valoraciones

Almacena el nombre para la lista de valoraciones que se pueden asignar a un proyecto, por defecto ya viene cargado con los siguientes.

- Normal
- Buena
- Muy Buena
- Excelente

3.4.3. Diagrama de clases Proyectos

Es la clase principal del proyecto, es la que almacena los datos de cada proyecto, cada proyecto va asociado a un Alumno, por el **IdAlumno**, y a un profesor, por **IdTutor**, que serán registros de la tabla **Usuarios**, tiene un **curso**, que será el año en cuestión del proyecto, y va asociado a un **Ciclo** y **Grupo**, estos campos se obtendrán de las tablas auxiliares **Ciclos** y **Grupos**, pero nos guardaremos directamente el texto, no será una relación, así aunque en el futuro cambien los datos de estas tablas, los proyectos que ya estén guardados seguirán conteniendo los datos originales.

Tiene tres campos de texto **Resumen**, será un resumen de que consiste el proyecto en sí, **Herramientas**, que servirá para especificar las herramientas, lenguajes y tecnologías empleadas en el desarrollo del proyecto y un campo **Comentarios**, que será el que rellenara el profesor con la evaluación del proyecto. Además tiene un campo **Valoración** en el cual se le podrá asignar una valoración sacada de la tabla **Valoraciones**.

Además los proyectos tienen una serie de documentos asociados mediante la tabla **ProyectosArchivos**, que servirá para guardar cada uno de los archivos asociados al proyecto. También contará con una lista de módulos asociados que será la tabla **ProyectosModulos**, para especificar con que módulos tiene más afinidad el proyecto, que se sacaran de la tabla **Modulos**. Y por último una lista de **Cualidades**, que se guardaran en la tabla **proyectosCualidades**, y se sacaran de la tabla **Cualidades**.

3.5. Mockups

A continuación se incluyen imágenes de los diseños previos de algunas pantallas de la aplicación.

Desarrollo gestión de proyectos

Home Login

Bienvenido a la pagina principal

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique, tortor mauris molestie elit, et lacinia ipsum quam nec dui. Quisque nec mauris sit amet elit iaculis pretium sit amet quis magna. Aenean velit odio, elementum in tempus ut, vehicula eu diam. Pellentesque rhoncus aliquam mattis. Ut vulputate eros sed felis sodales nec

Jaime Aguilá Sánchez

4 Diseño previo pantalla principal

Desarrollo gestión de proyectos

Home Login

Usuario

Contraseña

Entrar al sistema

Jaime Aguilá Sánchez

5 Diseño previo pantalla autenticación

Desarrollo gestión de proyectos

Home Usuarios Auxiliares ▼ Administrador del sistema Logoff

Listado de usuarios

Add Edit Del Q Buscar

| ▼ Usuario | ▼ Nombre | ▼ Correo | ▼ F.Alta | ▼ Activo |
|------------|--------------|--------------------------|------------|-------------------------------------|
| PruebaRana | Jaime Aguilá | jaguila@pymsolutions.com | 02/05/2016 | <input checked="" type="checkbox"/> |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

1 2 3 ... 7 8 9

Jaime Aguilá Sánchez

7 Diseño previo pantalla Listado usuarios

Desarrollo gestión de proyectos

Home Usuarios Auxiliares ▼ Administrador del sistema Logoff

Añadir nuevo usuario

Usuario

Clave

Nombre

Correo

F. Alta

Aceptar Cancelar

Jaime Aguilá Sánchez

6 Diseño previo pantalla alta usuarios

4. Tecnologías y Herramientas

4.1. PHP

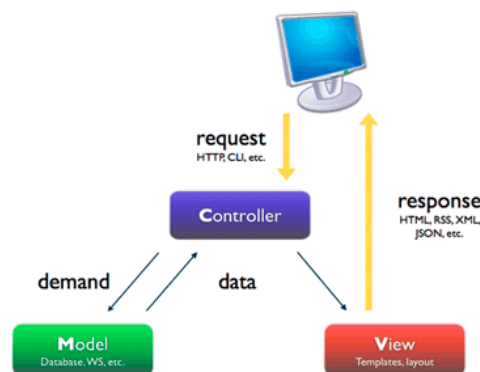
PHP es un lenguaje de programación de uso general de **código del lado del servidor** originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

PHP se considera uno de los lenguajes más flexibles, potentes y de alto rendimiento conocidos hasta el día de hoy, lo que ha atraído el interés de múltiples sitios con gran demanda de tráfico, como Facebook, para optar por el mismo como tecnología de servidor. Fue creado originalmente por Rasmus Lerdorf en **1995**. Actualmente el lenguaje sigue siendo desarrollado con nuevas funciones por el grupo PHP. Este lenguaje forma parte del software libre publicado bajo la licencia PHP, que es incompatible con la Licencia Pública General de GNU debido a las restricciones del uso del término PHP.

4.2. MVC sobre PHP

El modelo–vista–controlador (**MVC**) es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

En pocas palabras, el patrón de diseño MVC organiza el código en base a su función. De hecho, este patrón separa el código en tres capas:



8 Patrón MVC

- La capa del **modelo** define la lógica de negocio (la base de datos pertenece a esta capa).

- La **vista** es lo que utilizan los usuarios para interactuar con la aplicación (los gestores de plantillas pertenecen a esta capa).
- El **controlador** es un bloque de código que realiza llamadas al modelo para obtener los datos y se los pasa a la vista para que los muestre al usuario.

4.3. PHPUnit

PHPUnit es el estándar de facto para hacer test unitarios en proyectos PHP.

4.4. MySQL

MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation y está considerada como la base de datos open source más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server, sobre todo para entornos de desarrollo web.

MySQL fue inicialmente desarrollado por MySQL AB fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010, la cual ya era dueña desde 2005 de Innobase Oy, empresa finlandesa desarrolladora del motor InnoDB para MySQL.

Al contrario de proyectos como Apache, donde el software es desarrollado por una comunidad pública y los derechos de autor del código están en poder del autor individual, MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código. Esto es lo que posibilita el esquema de doble licenciamiento anteriormente mencionado. La base de datos se distribuye en varias versiones, una Community, distribuida bajo la Licencia pública general de GNU, versión 2, y varias versiones Enterprise, para aquellas empresas que quieran incorporarlo en productos privativos. Las versiones Enterprise incluyen productos o servicios adicionales tales como herramientas de monitorización y soporte oficial.

MySQL es usado por muchos sitios web grandes y populares, como Wikipedia, Google (aunque no para búsquedas), Facebook, Twitter, Flickr, y YouTube.

4.5. IIS

Internet Information Services o IIS es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows. Originalmente era parte del Option Pack para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003. Windows XP Profesional incluye una versión limitada de IIS. Los servicios que ofrece son: **FTP, SMTP, NNTP y HTTP/HTTPS**.

Este servicio convierte a un PC en un servidor web para Internet o una intranet, es decir que en los ordenadores que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente.

Se basa en varios módulos que le dan capacidad para procesar distintos tipos de páginas. Por ejemplo, Microsoft incluye los de Active Server Pages (**ASP**) y **ASP.NET**. También pueden ser incluidos los de otros fabricantes, como **PHP** o **Perl**.

Aunque vamos a usar IIS para el desarrollo del proyecto, se podría usar Apache como servidor web sin demasiados problemas.

4.6. JQuery

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, 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. Fue presentada el 14 de enero de 2006 en el BarCamp NYC.

jQuery es la biblioteca de JavaScript más utilizada.

jQuery 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 privados. jQuery, al igual que otras bibliotecas, 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 biblioteca se logran grandes resultados en menos tiempo y espacio.

4.7. Framework easyui

Es un pequeño framework, que funciona de forma conjunta con **jQuery**, y que nos provee de una serie de componentes listos para que utilicemos en nuestras pantallas web. Este framework a su vez se compone de un conglomerado de plugins muy utilizados, los mismos se encuentran integrados en un entorno de trabajo de muy simple utilización. Con tan solo unas líneas de código podemos ordenar tablas, crear pestañas, validar formularios y varias decenas de otras funcionalidades, todas con una simple llamada.

4.8. Control de versiones

Comenzaremos explicando que es un sistema de control de versiones y porque es necesario. Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión dando lugar a los llamados sistemas de control de versiones. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico). Ejemplos de este tipo de herramientas son entre otros: **CVS**, **Subversion**, **SourceSafe**, **Git**, **Mercurial**, o **Team Foundation Server**.

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente dando lugar a los sistemas de control de código fuente o SCM. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etc.

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar.
- Posibilidad de realizar cambios sobre los elementos almacenados.
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).

Aunque no es estrictamente necesario, suele ser muy útil la generación de informes con los cambios introducidos entre dos versiones, informes de estado, marcado con nombre identificativo de la versión de un conjunto de ficheros, etc.

4.8.1. Git

Git es un sistema de control de versiones distribuido lo que le aporta:


- Velocidad
- Diseño sencillo
- Fuerte apoyo al desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos (como el núcleo de Linux) de manera eficiente (velocidad y tamaño de los datos)


Desde su nacimiento en 2005, Git ha evolucionado y madurado para ser fácil de usar y aún conservar estas cualidades iniciales. Es tremendamente rápido, muy eficiente con grandes proyectos, y tiene un increíble sistema de ramificación (branching) para desarrollo no lineal, además en la actualidad se ha convertido en un estándar de facto, sobre todo en el desarrollo de software libre.

4.8.2. GitHub

GitHub es un alojamiento web para alojar proyectos utilizando el sistema de control de versiones **Git**. El código se almacena en repositorios de forma pública, aunque también se puede hacer de forma privada, creando una cuenta de pago.

Implementar funciones de sanitizacion de texto para que los modelos las puedan usar. #7

 **Open** PruebaRana opened this issue 3 minutes ago · 0 comments



PruebaRana commented 3 minutes ago

Owner +👤🔗

Aladir al fichero de utilidades funciones genericas para sanitizar codigo.
Sanitizar -> Eliminar codigo peligroso ("", "'", etc)
ClearJScript -> Eliminar
Normalizar -> Sustituir acentos y caracteres extendidos.
EliminaHTML -> Su nombre lo dice todo.
Capitalizacion -> Poner en mayuscular la primera letra de cada palabra

Labels

enhancement

Milestone

No milestone

Assign

Pr

9 Apartado de Issues de Github

5. Implementación

5.1. Metodología

Para el desarrollo del proyecto, se ha optado por usar metodologías ágiles, dado que las metodologías tradicionales (Métrica 3, Cascada, en V, ITIL) son muy rígidas frente a los cambios, se deben seguir unas políticas y normas muy estrictas, y obligan a ceñirse al análisis inicial del desarrollo.

Por el contrario las metodologías ágiles (XP¹, SCRUM, Kanban) tienen una mayor flexibilidad ante los cambios, los procesos están menos ceñidos a normas y permiten una mayor respuesta al cambio, y dado que el proyecto queremos que nos sea lo más útil posible, necesitábamos tener una mayor libertad para ir adaptándolo y modificándolo conforme fuéramos avanzando en él.

Los defensores de la XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se ha partido de un análisis genérico, para a continuación ir realizando cada sección en ciclos más cortos en los cuales se parte de un análisis, se buscan cada una de las tareas a realizar, se ordenan y se van implementando, en caso de encontrarnos con vacíos o mejoras a implementar, se añaden a la lista de tareas, para realizarlas en el siguiente ciclo.



10 Metodologías Ágiles frente a tradicionales

5.1.1. Metodología mixta

Dado que prácticamente todos los roles definidos tanto en Scrum o XP, los hemos acaparado, no tenía mucho sentido seguir una metodología ágil al pie de la letra, ya que las reuniones de equipo (clientes, programadores, analistas, testers) no tenían sentido, así que hemos acabado realizando una mezcla de desarrollo Agile.

En vez de desarrollar todo el proyecto en bloque y no tener nada funcional hasta los últimos pasos del desarrollo, hemos dividido el proyecto en pequeñas partes totalmente funcionales y estas a su vez las hemos dividido en tareas mucho más sencillas, realizando pequeñas iteraciones que incluían planificación, análisis de requisitos, diseño, codificación, pruebas y documentación.

¹ XP - eXtreme Programming o programación extrema, formulada por Kent Beck

Teniendo gran importancia el concepto de "Finalizado" (Done), ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar la finalización del producto, sino incrementar el valor por medio de "software que funciona" (sin errores).

Para ellos nos hemos apoyado totalmente en Github, su apartado de Issues, y en el uso de **PDA**s (Papel De Apuntar), cada tarea, mejora, error, adaptación, acababa apuntado en un papel, y en cada ciclo se agrupaban una serie de pequeñas tareas que tuvieran relación y se incluían en el proceso, una vez acabadas, se probaba su funcionamiento y se refactorizaba el código para darle un mínimo de calidad, si en algún momento se encontraban errores en otros apartados, o aparecía una deficiencia en el diseño que necesitara cambiar alguna parte ya realizada, acababan apuntados en papel, para que en el siguiente ciclo se tuvieran en cuenta. También hemos usado GitHub, en su apartado de Issues, para identificar estos ciclos, y tener una visión más global de la situación, ya que nos permite clasificar cada Issue por colores para identificar los que son mejoras, de lo que son corrección de errores, y ver rápidamente cuantos tenemos ya cerrados, y cuantos hay abiertos, como se puede ver en la siguiente imagen.

5.1.2. TDD

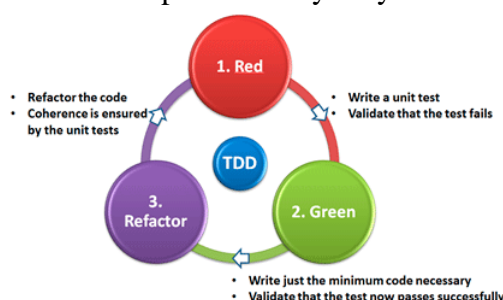
Aunque no hemos seguido una metodología totalmente TDD (Test-Driven Development) desarrollo guiado por pruebas, sí que hemos querido apoyarnos en los test unitarios, para poder realizar los procesos de refactorización teniendo mayor seguridad.

En TDD en primer lugar se debe definir una lista de requisitos y después se ejecuta el siguiente ciclo:

- **Elegir un requisito:** Se elige de una lista el requisito que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácilmente implementable.
- **Escribir una prueba:** Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces.
- **Verificar que la prueba falla:** Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.
- **Escribir la implementación:** Escribir el código más sencillo que haga que la prueba funcione. Se usa la expresión "Déjelo simple" ("Keep It small and simple"), conocida como principio **KISS**.
- **Ejecutar las pruebas automatizadas:** Verificar si todo el conjunto de pruebas funciona correctamente.
- **Eliminación de duplicación:** El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado. Se hace un pequeño cambio cada vez y luego se corren las pruebas hasta que funcionen.
- **Actualización de la lista de requisitos:** Se actualiza la lista de requisitos tachando el requisito implementado. Asimismo se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requisitos de diseño (P. ej que una funcionalidad esté desacoplada de otra).

Resumiendo en el TDD primero se escribe el Test, de forma que este al ejecutarse falla, después se escribe el código mínimo del programa para que el test no falle, después este código se refactoriza y se vuelve al principio para escribir nuevos test.

Es un enfoque que puede parecer contraproducente y muy lento al principio, pero que obliga a



11 Ciclo TDD Rojo-Verde-Refactorizar

ir conociendo la lógica del negocio de forma más profunda para ir implementando tests más específicos que acaben dando como resultado el código mínimo que implementa toda la funcionalidad básica de la aplicación a la par que la cobertura del código mediante test es del 100%, dando lugar a un código robusto y que está completamente comprobado por test, de forma que el mantenimiento y la ampliación sea más sencilla y segura.

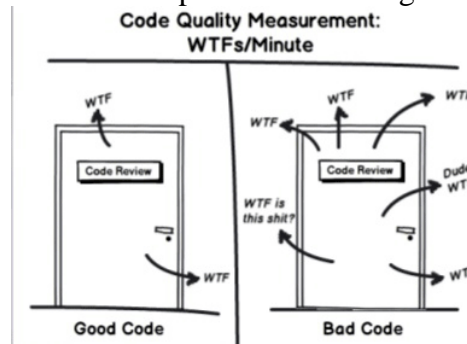
En nuestro caso, no hemos desarrollado la aplicación siguiendo TDD, sí que empleábamos el mismo concepto de ciclos, lo único que en cada ciclo no comenzábamos directamente escribiendo las pruebas, sino que iban a la par, mientras implementábamos el código íbamos escribiendo pruebas y estas a su vez nos hacían reevaluar la implementación, una vez finalizado el ciclo, refactorizábamos el código, y los test nos facilitaban mucho la tarea, ya que sin tener pruebas unitarias, la refactorización puede ser una lotería.

5.1.3. Refactorización

Refactorizar es realizar una transformación al código **preservando su comportamiento**, modificando sólo su estructura interna para **mejorarlo**.

Es muy importante tener muy claro este concepto, ya que la refactorización no debe de modificar el comportamiento del código, esto es **fundamental**. Refactorizar no es corregir errores, ni añadir funcionalidades, ni hacer optimizaciones de código para que sea más rápido, es mejorar el código para que sea más robusto y fácil de mantener, sin que su comportamiento cambie, lo que se conoce informalmente por limpiar el código.

El objetivo es mejorar la facilidad de comprensión del código o cambiar su estructura y diseño



12 Identificar calidad del código

y eliminar código muerto, para facilitar el mantenimiento en el futuro.

A finales de 1990, Kent Beck acuñó el término **“Code Smell”** conocido por **código que huele o apesta**. Una forma de saber si nuestro código necesita ser refactorizado es intentar detectar estos “olores”, conocidos como Bad Smell. Un Bad Smell es un indicio de que existe código de poca calidad, hay muchas formas distintas de que el código apeste y existen refactorizaciones específicas para cada uno de estos olores.

Aprender a identificar estos “malos olores”, y conocer cómo se suelen Refactorizar para evitar que apesten, nos puede ser muy útil en el desarrollo de software, y sobre todo a la hora de ampliar funcionalidades o mantener el código.

No vamos a detallar cada uno de estos olores, y cada una de las formas o métodos de Refactorizar.

5.2. Patrones de diseño

En programación a veces nos encontramos con ciertos problemas que se repiten a lo largo de nuestra experiencia, como puede ser el caso típico, hacer una clase que pueda controlar múltiples tipos de conexiones, o una clase que una vez instanciada (creado un objeto) no se puede volver a crear una nueva instancia.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su **efectividad** resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser **reutilizable**, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Fue a principios de la década de los 90, cuando los patrones de diseño tuvieron un gran éxito en el mundo de la informática a partir de la publicación del libro **Design Patterns** escrito por el grupo **Gang of Four (GoF)** compuesto por **Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides**, en el que se recogían 23 patrones de diseño comunes.

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Asimismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

No es obligatorio utilizar los patrones, solo es aconsejable en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. "Abusar o forzar el uso de los patrones puede ser un error". No vamos a detallar cada uno de estos patrones, simplemente vamos a comentar dos de ellos que hemos usado en el proyecto.

5.2.1. El patrón **Singleton**

Se encarga de garantizar la existencia de una única instancia de una clase y la creación de un mecanismo de acceso global a dicha instancia.

Tanto el objeto **ConexionBD** como **Config**, implementan el patrón singleton para que solo pueda haber una instancia en todo el ciclo de vida de la petición de un cliente, de esta forma el acceso a la BD se crea y se abre una única vez, cuando un modelo lo instancia, y puede ser usado varias veces desde distintos modelos compartiendo la instancia, mientras dure el proceso de la página.

```
<?php
class ConexionBD extends PDO
{
    private static $instance = null;
    public function __construct()
    {
        $config = Config::GetInstance();
        parent::__construct('mysql:host='.$config->get('db ... pass'));
        //Realiza el enlace con la BD en utf-8
        parent::exec("set names utf8");
        parent::setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, TRUE);
    }
    public static function GetInstance()
    {
        if( self::$instance == null )
        {
            self::$instance = new self();
        }
        return self::$instance;
    }
}
```

13 Implementacion patrón singleton

La implementación es muy sencilla, se crea una propiedad privada de tipo **static**, a la que llamaremos **\$instance**, y cuando necesitemos acceder al objeto, lo instanciamos llamando a **::GetInstance()**, en este método, se comprueba si nuestra propiedad **instance** está o no inicializada y si no lo esta se inicializa.

5.2.2. El patrón **FrontController**

Es un patrón de diseño que se basa en usar un controlador como punto inicial para la gestión de todas las peticiones, se encarga de realizar funciones como comprobación de restricciones de seguridad, manejo de errores, mapear y delegar las peticiones a otros componentes de la aplicación que se encargarán de generar la vista para el usuario.

Todas las peticiones se realizaran al fichero **index.php** que se encuentra en la raíz, este fichero lo único que hace es instanciar a **FrontController**, que es el que inicializa el funcionamiento del siguiente patrón **MVC**.

```
//Si todo esta bien, creamos una instancia del controlador y llamamos a la acción
$controller = new $controllerName();
$controller->$actionName();
```

14 Instanciación y llamada al método de acción del controlador final

El **FrontController** es el encargado de recibir todas las peticiones, incorpora los ficheros necesarios mediante **require_once**, y determina que controlador es el encargado de atender la petición y que acción es la que se requiere.

En el **Anexo A**, se adjunta el código de **FrontController**, para comprobar si el controlador es válido o si la acción es correcta, no lo hacemos en base a un array sino que comprobamos que el fichero en cuestión exista y que la acción dentro del controlador de destino pueda ser llamada desde PHP, así no tenemos que estar manteniendo un array de controladores con acciones, que puedan no coincidir con el código final.

5.2.3. El patrón **MVC**.

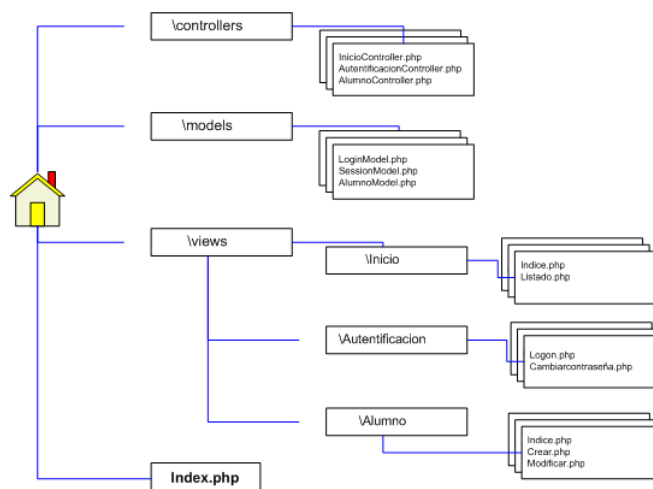
Ya se ha explicado en el punto **4.2 MVC sobre PHP**, en que consiste el patrón MVC, aquí vamos a explicar como lo hemos implementado.

Hay muchos frameworks en PHP que implementan directamente **MVC** como **CakePHP**, **Laravel**, **Symfony** o **CodeIgniter**, No hemos querido usar ninguno de estos frameworks, y hemos decidido implementarlo directamente por varios motivos:

- No queremos depender de un framework en particular y la necesidad de aprender a utilizarlo, ya que estos frameworks implementan muchas más funcionalidades que para el proyecto en cuestión no nos interesan.
- Queremos investigar el funcionamiento del patrón **MVC** dentro de PHP, aunque sea al nivel más básico posible.

Esta implementación ha sido lo que más nos ha costado desarrollar, no por la dificultad en sí, sino porque partíamos con la experiencia de uso de la implementación de **MVC sobre .NET**, y hemos partido de manías heredadas, que nos han encauzado a desarrollar el patrón de una forma muy parecida a la de .NET.

- Los controladores estarán dentro de la carpeta **\Controllers**, y el nombre del fichero y la clase de php será el nombre del controlador acabado con la palabra controller, **AlumnoController**.
- Los modelos estarán dentro de la carpeta **\Models**, y el nombre del fichero y la clase de php será el nombre del modelo acabado con la palabra Model, **AlumnoModel**.
- Las vistas estarán dentro de la carpeta **\Views**, y existirá una carpeta por cada controlador que necesite una vista, con el nombre del controlador, y dentro de esta carpeta estarán los ficheros finales de php de la vista en sí.



15 Estructura carpetas MVC

Existirán otras carpetas, para el funcionamiento de la web, que serán los siguientes:

- **Css:** Aquí estarán los ficheros de estilos
- **Includes:** Paginas PHP que se usaran como includes, ej cabeceras y pies.
- **Instalar:** Las paginas para realizar una instalación limpia, estas páginas generaran la base de datos en MySQL
- **Js:** Los ficheros de javascript usados en las páginas.
- **Libs:** Ficheros de PHP, con funcionalidades necesarias, útiles.php, conexión.php

5.2.3.1. Controladores

Los controladores serán nombrados en singular y acabados en **Controller**, y se guardaran en la carpeta **Controllers**.

Los controladores, internamente tienen un nombre, para identificarlos, una instancia al objeto **view**, que es un pequeño motor de plantillas, muy básico, y el método por el cual se ha llegado a la acción (**GET, POST, TRACE**)

En el método **__destruct()**, se liberan los objetos que se hayan instanciados.

En el método encargado de realizar la acción, incluiremos el fichero del modelo correspondiente, lo instanciaremos, realizaremos las acciones que correspondan, montaremos un Array **\$data**, con todos los datos que queremos pasar a la vista, y acabará llamando a nuestro motor de plantillas, para que llame a la vista final pasándole los datos de **\$data**.

```
//Incluye el modelo que corresponde
require_once 'models/LoginModel.php';
//Creamos una instancia de nuestro "modelo"
$item = new LogonModel();
// Se realizan las acciones necesarias
$item = $item.obtenerlista();
//Pasamos a la vista toda la información a representar
$data["Item"] = $item;
$data["mensaje"] = "";
//Finalmente presentamos nuestra plantilla
$this->view->show($this->_Nombre . "/login.php", $data);
```

16 Código básico de una acción de un controlador

Además y muy importante, a los controladores y a las acciones en sí, debemos de poder especificarles un nivel de acceso, para ello tenemos un método **EstaLogueado**, que puede recibir o no, una lista de roles (**Administrador, Profesor, Usuario**) es un método de salvaguarda, en el momento que se ejecuta, comprueba si la petición tiene un usuario logueado, y si le pasamos una lista de roles, comprueba que el usuario en cuestión tenga algún rol de los que le pasemos, en caso de no cumplirse algo, manda una cabecera redirect al cliente redirigiéndole a la página de login y cancela el resto del proceso.

De esta forma, si un controlador debe ser solo permitido para los Administradores, colocaremos **EstaLogueado("Administrador")**, en el constructor del controlador, y sabremos que todo el que acceda a cualquier método debe estar logueado y ser administrador, si por el contrario solo queremos permitir el acceso a una acción, a los profesores, incluiremos como primera llamada en ese método **EstaLogueado("Profesor")**, si simplemente queremos que el usuario este logueado sin más **EstaLogueado()**;

5.2.3.2. Modelos

Por convención todos los modelos serán nombrados en singular y acabados en **Model**, y se guardaran en la carpeta **Models**

Los modelos serán los únicos que se comunicaran directamente con la base de datos, obtendrán los datos, los añadirán y modificaran, para ello tendrán una propiedad que será una instancia a la conexión con la base de datos, nuestro objeto **ConexionBD**, esta propiedad se instanciara en el constructor, y al ser de tipo singleton, solo se inicializara una vez.

A parte contienen todas las propiedades, que se usaran en las vistas para pintar los datos, y todos los métodos necesarios que necesiten los controladores.

```
// Propiedad que es una instancia de nuestro objeto ConexionBD
protected $_db;

public function __construct()
{
    // Obtenemos la única instancia a la BD
    $this->_db = ConexionBD::GetInstance();
}
```

17 Código básico de un modelo

5.2.3.3. Motor de vistas

Las vistas son llamadas desde nuestro motor de vistas el cual es instanciado desde los controladores, este motor es muy básico pero nos da toda la funcionalidad que necesitamos.

```
class View
{
    function __construct()
    {
    }
    public function show($name, $vars = array())
    {
        // $name es el nombre de nuestra vista, por ej, listado.php
        // $vars es el contenedor de nuestras variables, Array llave => valor, opcional.

        // Traemos una instancia de nuestra clase de configuracion.
        $config = Config::GetInstance();

        // Montamos la ruta a la vista
        $path = $config->get('viewsFolder').$name;

        // Si no existe el fichero en cuestion, mostramos un 404
        if (file_exists($path) == false)
        {
            trigger_error ('Template `` . $path . `` does not exist.', E_USER_NOTICE);
            return false;
        }
        // Si hay variables para asignar, las pasamos una a una.
        if (is_array($vars))
        {
            foreach ($vars as $key => $value)
            {
                // Ojo al doble $, para crear la variable con el nombre de la variable
                $$key = $value;
            }
        }
        // Finalmente, incluimos la plantilla.
        include($path);
    }
}
```

18 Código del motor de vistas

Nuestro motor **View**, contiene un único método **show**, que es el encargado de llamar a una copia de nuestro fichero de configuración, comprobar si la vista existe, y si le pasamos un Array con datos, se encarga de recorrerlo y generar variables con los valores que contiene, de forma que desde el fichero de la vista, podamos acceder directamente a estos valores. Desde los ficheros finales de php, tendremos acceso directo a **\$config**, **\$vars** y a cada uno de los elementos de **\$vars** llamándolos directamente por su nombre, como si los hubiéramos declarado directamente en el propio fichero.

5.2.3.4. Vistas

Las vistas estarán todas dentro de una carpeta con el nombre del controlador al que pertenecen dentro de la carpeta **Views**, estarán nombradas como el nombre de la acción que representan, y contendrán el código php para pintar los datos. A continuación se presenta un ejemplo de su uso.

Mediante el uso de los buffers de PHP, definiremos tres secciones, la sección de los estilos **seccionCSS**, la sección del contenido **seccionContenido**, y la sección de los JS **seccionJS**, de esta forma cada vista podrá cargar el CSS o el JS que necesite, sabiendo que este se cargará en su sección correspondiente, estas dos secciones no tienen por qué estar, si una vista no contiene CSS o JS personal, no necesitarán definir estas secciones, la única necesaria sea la del contenido que será el HTML propio de esta vista.

Al final de la vista, se hará un include de la plantilla que se desea cargar, de esta forma podremos incluso tener diferentes plantillas y serán las vistas las encargadas de decidir que plantilla es la que usaran.

```
<?php
// Este bloque ira en la seccionCSS arriba del todo en la plantilla, junto a los CSS
// Servirá para cuando necesitemos cargar CSSs específicos para nuestra vista
ob_start()
?>
    <link rel="stylesheet" type="text/css" href="css/ejemplo1.css" media="all" />
    <link rel="stylesheet" type="text/css" href="css/ejemplo2.css" media="all" />
<?php $seccionCSS = ob_get_clean() ?>

<?php
// este bloque sera el contenido principal de nuestra vista
ob_start()
?>
    <div>
        Nuestro contenido en si
    </div>
<?php $seccionContenido = ob_get_clean() ?>

<?php
// este bloque ira en la seccionJS abajo del todo en la plantilla, junto a los JS
// Servirá para cuando necesitemos cargar JSs específicos para nuestra vista
ob_start()
?>
    <script src="js/ejemplo1.js" type="text/javascript"></script>
    <script src="js/ejemplo2.js" type="text/javascript"></script>
<?php $seccionJS = ob_get_clean() ?>

<?php
// Esta sera la plantilla que usara esta vista, será la encargada de pintar
// las tres secciones seccionCSS, seccionContenido y seccionJS
include 'views/plantilla.php'
```

5.2.3.5. Plantillas

En el **Plantilla.php**, está el código de nuestra plantilla por defecto.

En la parte del **HEAD**, cargara todos los CSS necesarios para todas las páginas y realizara la carga de seccionCSS, si la vista lo ha generado, llamando a `<?php echo obtenerParametro($seccionCSS);?>`

El contenido lo cargara en la sección contenido, después de cargar la cabecera, llamando a `<?php echo obtenerParametro($seccionContenido);?>`, después cargara el pie de la página.

Justo antes de cerrar el **BODY**, cargara todos los JS necesarios para todas las páginas y a continuación realizara la carga de seccionJS, si la vista lo ha generado, llamando a `<?php echo obtenerParametro($seccionJS);?>`

Las secciones se cargaran llamando al método **obtenParametro**, que está en el fichero de Utilidades, este método lo usamos para que no se generen errores de tipo **Undefined Variable** en el fichero de error de PHP. De esta forma, no tenemos que estar definiendo siempre las secciones tanto si las necesitamos como si no, así en las vistas solo se generara la sección contenido que es la necesaria, y las otras dos secciones solo se generaran cuando realmente la vista las necesite.

5.3. Funcionamiento Web

Todas las secciones implementan unas acciones mínimas a través de los controladores, para el funcionamiento de la web, estas son.

5.3.1. Get - Índice

Es una petición por **GET**, que carga la vista Índice, esta vista usa el framework **EasyUI**, para construir un grid con el listado de los registros, en principio es una página vacía, que mediante peticiones por **Ajax** a la acción **ObtenerDatos**, se encarga de rellenar el grid con datos, realizar las paginaciones, las ordenaciones y las búsquedas.

5.3.2. Post - ObtenerDatos

Es una petición por **POST**, que recibe los siguientes parámetros **page**, es la página que quiere obtener, **rows**, son los elementos que habrá por página, **sort**, el campo por el que quiere ordenar, **order**, si la ordenación es **ASC**endente o **DESC**endiente, **WhereCampo**, el campo para realizar la búsqueda, **WhereValor**, el valor a buscar. Todas las acciones de paginado, ordenación y búsquedas, se realizarán en el servidor, el cliente se limitará a pintar los datos. Estos datos se retornan en un elemento JSON que contiene un valor **Total**, se usa para que el cliente sepa cuantas páginas existen en total, y un array de **elementos**, con los datos pedidos.

5.3.3. Get - Crear

Es una petición por **GET**, que se encarga de cargar la vista para la creación de un registro.

5.3.4. Post - Crear

Todas las peticiones que realicen modificaciones sobre los datos, se realizarán por **POST**, esta acción se encargará de obtener los datos que se quieren crear, instanciar al modelo correspondiente, realizar las comprobaciones correspondientes y si todo es correcto guardar los datos.

5.3.5. Get - Editar

Es una petición por **GET**, que recibe el Id del registro a modificar y se encarga de cargar la vista para la edición del registro, con los datos de este.

5.3.6. Post - Editar

Todas las peticiones que realicen modificaciones sobre los datos, se realizarán por **POST**, esta acción se encargará de obtener los datos que se quieren modificar, instanciar al modelo correspondiente, realizar las comprobaciones correspondientes y si todo es correcto guardar los datos.

5.3.7. Post - Deleteld

Todas las peticiones que realicen modificaciones sobre los datos, se realizarán por **POST**, esta acción se encargará de obtener los datos que se quieren borrar, instanciar al modelo correspondiente, realizar las comprobaciones correspondientes y si todo es correcto eliminar los datos.

5.4. Acceso a archivos

El acceso a los archivos asociados a los proyectos no se realiza de forma directa, estas peticiones las procesa el archivo **ObtenArchivo.php**, Este fichero realiza las comprobaciones de seguridad, si la persona que está intentando acceder esta logueada, si el fichero al que intenta acceder existe, y si todo es correcto, retorna el fichero al cliente para su descarga.

```
<?php
//Incluimos algunas clases:
require_once "libs/Config.php";           //de configuracion
require_once 'libs/ConexionDB.php';       //Acceso a BD por PDO con singleton
require_once 'libs/Utiles.php';           //
require_once 'config.php';                //Archivo con configuraciones.

// Iniciamos la sesion
require_once 'models/SessionModel.php';
session_start();
$SESSION = unserialize(serialize($ SESSION));
global $usuario;
$usuario = obtenerParametroArray($_SESSION, "user", new SesionModel());
$IdUsuario = $usuario->Id;

// Obtenemos todos los parametros necesarios
$lsROOT = __DIR__ . "/Contenidos/";
$File = $_GET['File'];
$FicheroFinal = $lsROOT.$IdUsuario."/". $File;
$NombreFichero = basename($FicheroFinal);

// Comprobaciones
// 1 - Esta logueado
EstaLogueado();

// 2 - Comprobar que el fichero en cuestion existe
if ( !file_exists( $FicheroFinal ) ) {
    header('HTTP/1.0 404 Not Found');
    echo "<h1>Error 404 Not Found</h1>";
    echo "El fichero que esta intentando consultar no existe.";
    exit;
}

// Si todo lo anterior es correcto, obtener el fichero y mandarlo al cliente.
$len = filesize($FicheroFinal);
// header('Content-type: application/pdf');
header('Content-Disposition: attachment; filename="'. $NombreFichero. '");
header('Content-Length: ' . $len);
readfile($FicheroFinal);
?>
```

20 Código ObtenArchivo.php

5.5. Seguridad

Partimos desde un posicionamiento totalmente negativo, toda petición que provenga del cliente puede ser potencialmente peligrosa, e incluso, todos los datos que obtengamos de la base de datos pueden haber sido corrompidos y ser también dañinos.

Hemos consultado con mucho detalle la web de **OWSAP**, ya que es el **Proyecto abierto de seguridad de aplicaciones web**, y hay muchísimo contenido para intentar estar protegido. Desde esta perspectiva, estamos obligados a filtrar todos los datos, en ambos sentidos.

Además para evitar ciertos ataques, todas las páginas dinámicas de la web (**las paginas PHP**) estarán fuera del alcance de la web, en una carpeta llamada **APP**, y la web estará en una carpeta llamada **WEB**, en la parte de la **WEB**, solo habrá dos ficheros PHP, el **index.php**, que es el corazón de la web, el que llama a **FrontController.php**, que es el que realiza toda la magia, y **ObtenArchivo.php**, que es el encargado de dar acceso a los ficheros que se suben a los proyectos, estos ficheros tampoco estarán en la web, de forma que no se pueda acceder a ninguno de ellos si no es pasando por **ObtenArchivo.php**.

5.5.1. Control de acceso.

Tenemos un método **EstaLogueado()**, que nos permite redireccionar automáticamente a la pantalla de login a toda petición que no coincida con los requisitos requeridos por el método, de forma que todos los controladores, pueden granular el acceso mediante este método, pudiendo establecerlo a nivel del constructor, de forma que todo el controlador este restringido, o a nivel de acción, he incluso combinarlos, de forma que un controlador tenga permiso solo a las peticiones que estén logueadas, y determinadas acciones dentro de este controlador, restrinjan aún más el acceso, estableciendo unos roles mínimos, como es el caso del controlador **ProyectoController**.

5.5.2. Verbos HTTP

Cualquier petición que implique que tengamos que realizar alguna consulta parametrizada o alguna modificación en la base de datos, forzamos que deba ser realizada por **POST**, por **GET** solo se aceptan peticiones que no realicen ninguna acción en la base de datos, esto obliga a tener un form, y a no poder falsificar peticiones directamente desde la barra del navegador, aún pueden falsificarse peticiones, pero ya no de una forma directa. Los controladores, solo permiten por **GET**, las acciones de **Inicio**, que pinta la plantilla de la acción en blanco, y el **Crear y Editar**, para pintar la plantilla de edición.

Para obtener los registros de las paginaciones en las pantallas de inicio, obligamos a que sean peticiones por **POST**, y las acciones de **Crear, Editar y Borrar** para guardar los datos en la base de datos, también son por **POST**, además de que les obligamos a que lleven un **Token** (se explicara más adelante en el punto **5.5.6 CSRF**).

5.5.3. Filtrado de parámetros.

En el fichero de **Utilidades.php**, hay unos cuantos métodos encargados del filtrado de peticiones. Que son usados desde los controladores cuando se intenta acceder a algún parámetro que ha enviado el cliente, ya sea las peticiones de las paginaciones, o bien, los datos del formulario de envío para realizar cambios.

Los métodos más importantes son los siguientes.

5.5.3.1. Sanitizar

Recibe el texto a procesar y un parámetro booleano para saber si tiene que permitir que contenga etiquetas HTML o no, **sanitizar(\$asTexto, \$aswAllowHTML)** el valor por defecto es false, solo se establece a true en casos específicos (los campos de **Resumen, Herramientas y Comentarios** de los proyectos, si se permite HTML, se eliminarán todos los bloques **SCRIPT, OBJECT, APPLET y EMBED** del contenido, y dentro de las etiquetas HTML, se eliminarán todos los eventos (**onBlur, onError, onFocus, onLoad, onResize, onUnload, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp**), en caso de no permitir HTML, seremos más estrictos y llamaremos a **strip_tags**.

Este método, se llamara **SIEMPRE**, tanto para obtener parámetros del cliente, como para obtener parámetros de la base de datos.

5.5.3.2. CapitalizarPalabra, CapitalizarFrases y Normaliza

Estos métodos son auxiliares, y los emplean los modelos para dar cierto formato a determinados campos.

El de normalizar se usa para obtener los nombres de los ficheros, eliminando acentos y caracteres extendidos.

5.5.3.3. obtenParametroArray

Este método se llama siempre que queramos obtener un parámetro de un array, por ejemplo parámetros que nos llegan por GET o por POST, de los arrays **\$_GET** y **\$_POST**, de esta forma nos evitamos generar basura en el fichero de errores de PHP cuando intentamos llamar a un parámetro que no existe, además de permitir especificarle al método, que valor nos tiene que retornar en caso de que no exista el parámetro.

5.5.4. XSS

Este tipo de ataques suelen producirse por una tercera persona que intenta inyectar código en páginas web visitadas por el usuario, puede ser de forma directa o indirecta, para evitar este tipo de ataques, lo único que podemos hacer es filtrar siempre los contenidos. Y nunca guardar nada en la base de datos que no haya sido antes filtrado.

5.5.5. Inyección SQL

Parecido al anterior solo que en este caso no intentan inyectar javascript en la página, sino sentencias SQL para atacar a la base de datos.

Ante este ataque, tenemos dos medidas, una la citada antes, todo se filtra en ambos sentidos, y la segunda medida es el uso de PDO parametrizado para el acceso a la base de datos, todas las consultas van parametrizadas tanto las que guardan datos, como las que simplemente retornan registros, para ello hacemos uso de una clase **POJO** en PHP no sé cómo se llaman, una clase vacía que solo nos almacena dos parámetros **WherePDO**, la Where que queremos ejecutar y un array con todos los parámetros que usaremos. Esta clase la emplean los controladores y los modelos, para establecer los campos **WHERE**.

5.5.6. CSRF – Cross Site Request Forgery

Este tipo de ataques, se basa en que la víctima este logueada dentro de un sistema que sea vulnerable, mientras que se le muestran o envían enlaces de otras páginas que están falsificados, para que al pulsar en ellos, realice la acción sobre el sistema vulnerable, a este

ataque se lo conoce también como enlace hostil, ataque de un click, cabalgamiento de sesión, y ataque automático.

Para protegernos de este tipo de ataques hemos vuelto a mirar hacia el MVC de .NET, ya que implementa un sistema que es muy eficaz, el uso de un TOKEN doble, en los formularios, y la sesión, el **AntiForgeryToken**, se establece un hash aleatorio en la sesión y en un campo hidden del formulario este hash tiene una validez de 20min, toda petición que llegue al servidor, se comprobará que tiene su Token, si lo tiene se validará contra la copia de este que hay en sesión, para comprobar que coinciden, y además se comprobará que el token de sesión no haya caducado, si todo es correcto, se continuará con la petición, si alguna de estas comprobaciones falla, se cancelará la petición.

Este token se regenera con cada petición, de forma que incluso podamos invalidarnos a nosotros mismos si accedemos a la web, desde dos ventanas del navegador.

5.6. URL Friendly

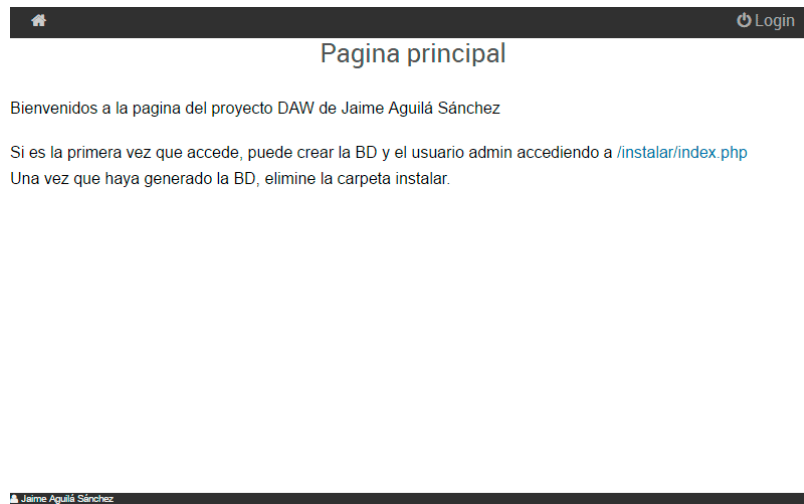
En el fichero **config.php**, hay un parámetro **\$config->set('urlFriendly', true);** que si lo establecemos a true, todos los enlaces que monta el sistema, son amigables, del estilo **/seccion/Inicio/índice** pero no obliga a que el servidor web donde este alojada la aplicación, se configure para aceptar este tipo de enlaces.

Nosotros hemos montado la aplicación dentro del IIS de Microsoft, y hemos configurado una regla de entrada para aceptar este tipo de enlaces, en caso de que se montara sobre apache, habría que replicar esta regla en el **htaccess** antes de establecer el parámetro a true.

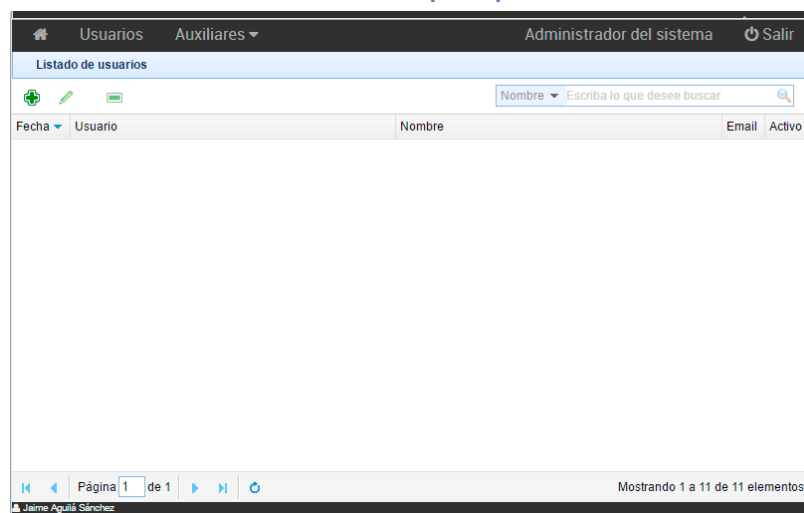
Si no lo establecemos a true, los enlaces serán todos del tipo **/index.php?controlador=inicio&accion=privado&esto=33**

5.8. GUI

A continuación se incluyen imágenes de algunas de las pantallas finales de la aplicación.



21 Pantalla principal



22 Pantalla listado usuarios

6. Resultados y conclusiones

6.1. Estado actual

La aplicación se ha realizado completamente, teniendo una aplicación totalmente funcional que nos cubre todos los objetivos iniciales.

- Un panel de entrada, que nos permita autenticarnos
- El acceso de tres roles distintos, administradores, profesores y alumnos
- Permitir dar de alta alumnos y profesores
- Permitir dar de alta y modificar los datos de un proyecto
- Permitirnos realizar búsquedas sobre los proyectos
- Realizar nuestro propio MVC en PHP.
- Aprender a usar o crear algún tipo de plantillas.

Personalmente, me puse el reto de aprender a realizar test unitarios y de interfaz, ya que aportan mucha seguridad y control al código desarrollado, y tengo que reconocer que he fracasado totalmente al intentar implementar una metodología basada en test, la intención inicial era usar **PHPUnit** para realizar test unitarios y si fuera posible realizar también test de interfaz, y no hemos podido realizar ninguno de los dos tipos de test.

El aprendizaje de **PHPUnit** y la realización de test se queda como deuda técnica que deberé de solventar en otro momento.

6.2. Conclusiones

Hemos obtenido un patrón **MVC** muy similar al de .NET, que implementa una gran facilidad para trabajar con él, ha quedado un producto bastante robusto y que nos da una buena seguridad.

El motor de vistas, junto con la gestión de plantillas, nos ahorra mucho trabajo a la hora de diseñar pantallas y el framework **easyUI**, nos permite realizar Grids de listado con muchísima funcionalidad, obteniendo los datos a través de AJAX.

Junto con las políticas de acceso por PDO parametrizado, el filtrado bidireccional de todos los datos y el uso de tokens en los formularios, hemos obtenido un juego de herramientas muy completo para el desarrollo dentro de PHP.

6.3. Posibilidades futuras

Debido a las limitaciones del proyecto, se ha tenido que sacrificar algunas opciones que queríamos realizar desde el principio, y mientras realizábamos el proyecto, nos han surgido otras ideas que tampoco tenían cabida realizarlas.

- Implementar un sistema similar al de NET para añadir a los modelos propiedades de validación, de forma que no tengamos que estar realizando las validaciones a mano.
- Implementar la localización de la aplicación para que el producto soporte el diversos idiomas.

6.4. Valoración personal

Personalmente estoy muy satisfecho con el proyecto en sí, he cubierto todos los objetivos iniciales, aunque he fallado en un objetivo personal a la hora de realizar test, que no ha sido posible, seguramente no por su dificultad en sí, sino porque es un proyecto académico con un margen de tiempo reducido, y hay que ser realistas en lo que se puede o no incluir. Lo que sí que he conseguido es obtener un patrón **MVC**, muy completo y con el que me siento muy cómodo para poder realizar otras aplicaciones web.

7. Anexos

A. FrontController

```
<?php
class FrontController
{
    static function main()
    {
        //Incluimos algunas clases:
        require_once "libs/Config.php";           //de configuración
        require_once 'libs/ConexionDB.php';       //Acceso a BD por PDO con singleton
        require_once 'libs/Utiles.php';           //

        require_once 'libs/View.php';             //Mini motor de plantillas
        require_once 'config.php';                 //Archivo con configuraciones.

        // Iniciamos la sesion
        require_once 'models/SessionModel.php';
        session_start();
        $SESSION = unserialize(serialize($ SESSION));
        global $usuario;
        $usuario = obtenerParametroArray($ SESSION, "user");

        // Por similitud con otros MVC los controladores terminan en Controller.
        // Por ej, la clase controladora Items, ser ItemsController
        // Obtenemos el Controlador o en su defecto, InicioController
        if(! empty($_GET['controlador'])) {
            $controllerName = $_GET['controlador'] . 'Controller';
        } else {
            $controllerName = CONTROLADOR_DEFECTO."Controller";
        }

        // Obtenemos la accion, si no hay acción tomamos indice como acción
        if(! empty($_GET['accion'])) {
            $actionName = $_GET['accion'];
        } else {
            $actionName = ACCION_DEFECTO;
        }

        $controllerPath = $config->get('controllersFolder') . $controllerName . '.php';

        //Incluimos el fichero que contiene nuestra clase controladora solicitada
        if(is_file($controllerPath)) {
            require $controllerPath;
        } else {
            // Lanzamos un 404
            die('El controlador no existe - 404 not found');
        }

        //Si no existe la clase que buscamos y su acción mostramos un error 404
        if (is_callable(array($controllerName, $actionName)) == false)
        {
            trigger_error ($controllerName.'->'.$actionName.' no existe', E_USER_NOTICE);
            return false;
        }

        //Si todo esta bien, creamos una instancia del controlador y llamamos a la acción
        $controller = new $controllerName();
        $controller->$actionName();
    }
}
?>
```


B. Plantilla.php

```
<!DOCTYPE html>
<html lang="es-ES">
  <head>
    <title><?php echo obtenerParametro($TituloPagina);?></title>
    <meta charset="utf-8" />
    <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
    <meta http-equiv="content-language" content="es" />
    <meta name="Lang" content="ES" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
    <meta http-equiv="x-dns-prefetch-control" content="on">
    <meta name="robots" content="NOINDEX,NOARCHIVE,NOFOLLOW" />
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />

    <link rel="stylesheet" type="text/css" href="js/easyui/themes/default/easyui.css" />
    <link rel="stylesheet" type="text/css" href="js/easyui/themes/icon.css" media="all" />
    <link rel="stylesheet" type="text/css" href="css/estilos.css" media="all" />
    <link rel="stylesheet" type="text/css" href="css/tema_defecto.css" media="all" />
    <link rel="stylesheet" type="text/css" href="css/buttons.css" media="all" />
    <?php echo obtenerParametro($seccionCSS);?>
  </head>
  <body>
    <!--[if lt IE 7]>
      <p class="chrome" >You are using an <strong>outdated</strong> browser. Please
        <a href="http://browsehappy.com/">upgrade your browser</a> or <a
          href="http://www.google.com/chrome" >activate Google Chrome
          Frame</a> to improve your experience.</p>
    <![endif]-->
    <header id="cabecera"><?php include("includes/_cabecera.php"); ?> </header>

    <section id="contenido">
      <?php echo obtenerParametro($seccionContenido);?>
    </section>

    <footer id="pie"><?php include("includes/_pie.php"); ?> </footer>

    <script src="js/jquery-1.8.3.min.js" type="text/javascript"></script>
    <script src="js/EasyUI/jquery.easyui.min.js" type="text/javascript"></script>
    <script src="js/EasyUI/locale/easyui-lang-es.js" type="text/javascript"></script>
    <script src="js/jquery.unobtrusive-ajax.min.js" type="text/javascript"></script>
    <script src="js/jquery.validate.min.js" type="text/javascript"></script>
    <script src="js/jquery.validate.unobtrusive.min.js" type="text/javascript"></script>
    <script src="js/jquery.validate.inline.min.js" type="text/javascript"></script>
    <script>
      var AltoFilasGrid = 30;
      var cantidadFilas = 20;
      <?php if (obtenerParametro($Mensaje) != null){ ?>
        $.messenger.show({ title:'Titulo', msg:'<?php echo $Mensaje; ?>', timeout:8000,
height:60, style:{
          left:'', right:0,
top:document.body.scrollTop+document.documentElement.scrollTop+45, bottom:'' }
        });
      <?php } ?>
      function IrA(asURL) {
        document.location.href = asURL;
      }
    </script>
    <?php echo obtenerParametro($seccionJS);?>
  </body>
</html>
```

C. Listado de módulos

| ASIR | |
|------|--|
| ISO | Implantación de sistemas operativos |
| PAD | Planificación y administración de redes |
| FHW | Fundamentos de hardware |
| SGBD | Sistemas Gestores de Bases de Datos |
| LGM | Lenguajes de marcas y sistemas de gestión de información |
| ASO | Administración de sistemas operativos |
| SRI | Servicios de red e Internet |
| AW | Implantación de aplicaciones web |
| GBD | Gestión de bases de datos |
| SAD | Seguridad y alta disponibilidad |
| EIE | Empresa e iniciativa emprendedora |
| ING | Ingles |
| FOL | Formación y orientación laboral |

| DAM | |
|------|--|
| SI | Sistemas informáticos |
| BD | Bases de datos |
| PRO | Programación |
| LGM | Lenguajes de marcas y sistemas de gestión de información |
| ED | Entornos de desarrollo |
| DWEC | Desarrollo web en entorno cliente |
| DWS | Desarrollo web en entorno servidor |
| DAW | Despliegue de aplicaciones web |
| DIW | Diseño de interfaces WEB |
| EIE | Empresa e iniciativa emprendedora |
| ING | Ingles |
| FOL | Formación y orientación laboral |

| DAW | |
|-----|--|
| SI | Sistemas informáticos |
| BD | Bases de Datos |
| PRO | Programación |
| LGM | Lenguajes de marcas y sistemas de gestión de información |
| ED | Entornos de desarrollo |
| AD | Acceso a datos |
| DI | Desarrollo de interfaces |
| PMM | Programación multimedia y dispositivos móviles |
| PSP | Programación de servicios y procesos |
| SGE | Sistemas de gestión empresarial |
| EIE | Empresa e iniciativa emprendedora |
| ING | Ingles |
| FOL | Formación y orientación laboral |

D. Sección rewrite de web.Config

```
<rewrite>
  <outboundRules>
    <preConditions>
      <preCondition name="ResponseIsHtml1">
        <add input="{RESPONSE_CONTENT_TYPE}" pattern="^text/html" />
      </preCondition>
    </preConditions>
  </outboundRules>
  <rules>
    <clear />
    <rule name="RewriteUserFriendlyURL1" stopProcessing="true">
      <match url="^contenidos/([^\/]*)/([^\/]*)" negate="false" />
      <conditions logicalGrouping="MatchAll" trackAllCaptures="false" />
      <action type="Rewrite"
url="ObtenArchivo.php?IdUser={R:1}&File={R:2}" />
    </rule>
    <rule name="URL Friendly" stopProcessing="true">
      <match url="^seccion/([^\/]*)/([^\/]*)" />
      <conditions logicalGrouping="MatchAll" trackAllCaptures="false">
        <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
        <add input="{REQUEST_FILENAME}" matchType="IsDirectory"
negate="true" />
      </conditions>
      <action type="Rewrite"
url="index.php?controlador={R:1}&accion={R:2}" />
    </rule>
  </rules>
</rewrite>
```

8. Referencias y bibliografía

- PHP Wiki - <https://es.wikipedia.org/wiki/PHP>
- PHP - <http://phpdevenezuela.github.io/php-the-right-way/#bienvenido>
- PHP - <http://php.net/manual/es/intro-what-is.php>
- MVC Wiki - <https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador>
- MVC en PHP - <http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/257>
- FrontController Wiki - https://en.wikipedia.org/wiki/Front_controller
- MySQL Wiki - <https://es.wikipedia.org/wiki/MySQL>
- PHP Pear Wiki - <https://es.wikipedia.org/wiki/PEAR>
- PHPUnit - <https://phpunit.de/>
- MySQL - <http://dev.mysql.com/doc/refman/5.7/en/>
- IIS Wiki - https://es.wikipedia.org/wiki/Internet_Information_Services
- JQuery Wiki - <https://es.wikipedia.org/wiki/JQuery>
- JQuery desarrollo web - <http://www.desarrolloweb.com/manuales/manual-jquery.html>
- EasyUI - <http://www.jeasyui.com/>
- XSS Wiki - https://es.wikipedia.org/wiki/Cross-site_scripting
- Inyección SQL Wiki - https://es.wikipedia.org/wiki/Inyecci%C3%B3n_SQL
- CSRF Wiki - https://es.wikipedia.org/wiki/Cross-site_request_forgery
- OWASP - https://www.owasp.org/index.php/PHP_Security_Cheat_Sheet
- NoCSRF.php - <https://github.com/BKcore/NoCSRF>