

## MEMORIA ESCRITA DEL PROYECTO

CFGS Desarrollo de Aplicaciones Web

### **Campus Virtual**

**Autor:** Rosa López Mas

**Tutor:** Pablo M. Díaz (Tever)

Fecha de entrega: 28/04/2023

**Convocatoria:** 2S – 2223

**Documentos del proyecto:** <https://drive.google.com/drive/folders/1TAcSzRgrmdObrpKOW-QPDDtKkFawuJjP?usp=sharing>



# Índice de contenido

1.	Introducción.....	3
1.1.	Motivación.....	3
1.2.	Abstract .....	3
1.3.	Objetivos propuestos (generales y específicos).....	3
2.	Metodología usada .....	5
3.	Tecnologías y herramientas utilizadas en el proyecto .....	6
4.	Estimación de recursos y planificación .....	8
5.	Análisis del proyecto.....	9
6.	Diseño del proyecto .....	11
6.1.	Base de Datos .....	11
6.2.	Prototipos.....	12
6.3.	Guía de estilo .....	12
6.4.	Estructura del proyecto (MVC) .....	12
6.5.	Crear/cerrar conexión BD, Registro, <i>Login</i> /cerrar sesión.....	13
6.6.	index/templateC/main.php .....	16
6.7.	Sidebar .....	17
6.8.	Módulos ( <i>body</i> ) .....	18
6.8.1.	Perfil .....	18
6.8.2.	Usuarios .....	19
6.8.3.	Gestor de Estudios .....	23
6.8.4.	Asignaturas.....	24
6.8.5.	Mensajes .....	26
6.8.6.	Calendario .....	27
6.8.7.	Expediente Alumno .....	30
6.8.8.	Modo Claro/Oscuro: .....	31
7.	Despliegue y pruebas .....	32
8.	Contexto laboral.....	34

<b>9. Instalación y configuración .....</b>	<b>35</b>
<b>10. Posicionamiento SEO .....</b>	<b>37</b>
<b>11. Conclusiones.....</b>	<b>38</b>
<b>12. Vías futuras .....</b>	<b>39</b>
<b>13. Glosario .....</b>	<b>40</b>
<b>14. Bibliografía/Webgrafía.....</b>	<b>41</b>
<b>15. Anexos.....</b>	<b>42</b>
<b>15.1. Anexos I.....</b>	<b>42</b>
<b>15.1.1. Anexos - Metodologías usadas .....</b>	<b>42</b>
<b>15.1.2. Anexos - Estimación de recursos y planificación .....</b>	<b>¡Error! Marcador no definido.</b>
<b>15.1.3. Anexos - Análisis del proyecto.....</b>	<b>43</b>
<b>15.1.3.1. Diagrama ER .....</b>	<b>43</b>
<b>15.1.3.2. Diagramas casos de uso .....</b>	<b>44</b>
<b>15.1.3.3. Diagrama de clases .....</b>	<b>45</b>
<b>15.1.4. Anexos - Diseño del proyecto.....</b>	<b>46</b>
<b>15.1.4.1. Base de datos (modelo relacional) .....</b>	<b>46</b>
<b>15.1.4.2. Prototipos y guía de estilo.....</b>	<b>47</b>
<b>15.1.4.3. Vistas de la aplicación .....</b>	<b>49</b>
<b>15.1.5. Anexos – Despliegue y pruebas.....</b>	<b>53</b>
<b>15.2 Anexo II .....</b>	<b>58</b>

En la normativa de proyectos vigente encontrarás una breve descripción de cada uno de estos apartados para saber qué información debes incluir en ellos

# 1. Introducción

## 1.1. Motivación

Debido a la pandemia hemos podido ver como la educación online ha incrementado e incluso la educación presencial ha optado por la utilización de herramientas vía online para que tanto profesores como alumnos las utilicen para subir/descargar recursos, tareas, etc. Estamos en plena era digital, donde empezamos a intercambiar herramientas de estudios como papel y bolígrafo por dispositivos como ordenadores, *tablets* o móviles. Es por ello que me interesó la idea de crear una aplicación web que encajase a la perfección en el ámbito educativo, una aplicación intuitiva para los usuarios, adaptable a los distintos niveles educativos (formación básica, formación profesional, universitaria) y, sobre todo, atractiva para el usuario, convirtiéndola así en una herramienta diaria durante el curso.

## 1.2. Abstract

The project consists of a virtual campus, an application for public or private educational entities. The application is focused on three types of users, each of them will be able to perform different actions. The administrator user will have the ability to create, read, update and delete (CRUD) the data in the database. The teacher user will be able to upload files and resources for his students, create events in the calendar visible to all users, grade his students and send/answer messages. Finally, the student user will be able to view the subjects, files and resources uploaded by his teacher, send private messages to the teacher or classmates, view his academic record and the events created.

The application will be designed in an attractive and responsive way, with very intuitive patterns for a comfortable usability, with the option of a dark mode to avoid problems of eyestrain.

## 1.3. Objetivos propuestos (generales y específicos)

Los **objetivos generales** son que profesores y alumnos tengan la aplicación como herramienta rutinaria para llevar a cabo las tareas durante el curso escolar. El sistema requerirá de un registro inicial del usuario, tras el registro deberán validar la cuenta mediante correo electrónico. Una vez validada la cuenta podrán acceder a la aplicación mediante un *login*, insertando el correo electrónico y la contraseña.

La aplicación va a estar compuesta por módulos. La vista de la misma estará estructurada por dos elementos principales, un cuerpo central que mostrará el contenido de cada módulo y un menú con el que se accederá a los distintos módulos. Los módulos que tendrá la aplicación serán los siguientes:

- Perfil: Módulo donde se podrá ver y editar los datos del usuario.

- Usuarios: Módulo para el administrador, donde desde ahí podrá editar, crear o eliminar los datos de los usuarios.
- Estudios: Módulo para administradores y profesores. Allí se podrá crear/editar/eliminar estudios (acciones solo para administrador) y se podrá acceder al listado de asignaturas que tiene cada curso y al listado de alumnos que cursan cada estudio. Así mismo, el profesor podrá acceder al expediente de cada alumno donde calificará cada asignatura.
- Mensajes: Módulo enfocado para la comunicación entre usuarios, donde se podrá enviar/recibir mensajes privados a otro usuario registrado.
- Calendario: Módulo con un calendario el cual el usuario podrá visualizar eventos de otros usuarios y crear/editar/eliminar sus propios eventos.
- Asignaturas: Módulo para alumnos y profesores, donde se mostrará un listado de las asignaturas del curso realizado y de cada asignatura el profesor pueda subir apuntes/recursos de la asignatura y el alumno pueda visualizarlos y descargarlos.
- Expediente: Espacio exclusivo para el alumno, donde podrá ver las notas de cada materia, pudiendo así descargarlo en formato PDF.

Otros objetivos que me gustaría lograr es que el usuario pudiese personalizar la interfaz de la aplicación con el modo claro/oscuro y que el menú lateral fuese desplegable.

Los **objetivos específicos** son, principalmente, **analizar** bien qué queremos que haga la aplicación y las acciones de cada usuario, estructurando las entidades necesarias de la aplicación y sus relaciones entre sí mediante el diagrama Entidad-Relación(E-R) y el diagrama de casos de uso para clarificar mejor qué acciones realizará cada usuario.

Posteriormente **diseñar** la aplicación, estructurar cada módulo, definir un diseño para que la aplicación tenga sinergia, con un mismo patrón. Una vez aclaradas las funciones de que desempeñará cada módulo el siguiente objetivo será definir el diagrama de clases, del cual nos guiaremos para **desarrollar** la parte lógica del proyecto estructurándolo con el patrón de diseño Modelo-Vista-Controlador (MVC), todo ello con sus pertinentes **pruebas** unitarias tras la finalización de cada módulo y pruebas de caja negra una vez terminada la aplicación para verificar el correcto funcionamiento de la misma. Por último, **documentarlo** todo en las próximas páginas de esta memoria.

## 2. Metodología usada

El ciclo de vida que mejor se adapta a este proyecto es el **modelo iterativo incremental**, ya que cada incremento estará formado por un módulo, el cual pasará por un análisis, un diseño, un desarrollo de código y unas pruebas, mientras no cerremos un módulo no pasaremos al siguiente. Por lo que, en caso de tener un cliente, podemos ir haciendo pequeñas entregas para recibir un *feedback* por parte del mismo y mejorar la aplicación. Este modelo nos permite ir definiendo sobre la marcha algunos objetivos y dejar abiertas las posibilidades a cambios en los próximos incrementos.

Como metodología ágil, la que mejor encaja en este caso es la **SCRUM**, ya que podremos ir haciendo *Sprints*, pequeñas entregas de los módulos(incrementos) finalizados a los clientes para su testeo. Cabe decir que como organización personal he utilizado un sistema de “tarjetas” creadas en la aplicación Trello divididas por las fases del ciclo de vida (análisis, diseño, codificación, pruebas...), dentro de cada tarjeta hay un listado de objetivos los cuales fui marcando según se finalizaba la tarea.

*[Imágenes en anexos.]*

### 3. Tecnologías y herramientas utilizadas en el proyecto

Las **tecnologías** empleadas son las siguientes:

- **HTML5** y **CSS3**, dadas en la asignatura **Lenguajes de Marcas**, tecnologías para maquetar y diseñar la parte cliente de la aplicación.
- **Bootstrap(v5)**, *framework* de código abierto aceptado en la normativa del proyecto que incluye elementos de diseño basados en HTML y CSS aportando así un diseño *responsive* ahorrándonos así mucho código CSS.
- **JavaScript** (JS), lenguaje estudiado en **Entorno-Cliente**, el cual nos aporta interactividad a la aplicación, utilizando a su vez AJAX con el fin de facilitar al usuario la experiencia con la aplicación libre de recargas.
- **jQuery**, librería de JS, estudiada en **Diseño de interfaces**, la cual nos proporciona un aspecto más animado a la aplicación.
- **PHP(v8)**, lenguaje *back-end*, estudiado en **Entorno-Servidor**, es el más empleado en esta aplicación, dado que es una aplicación basada en la metodología CRUD. Ha sido un lenguaje esencial para comunicar la parte del cliente con el servidor, tomando los datos pertinentes y enviándolos al servidor mediante consultas SQL.
- **MySQL**, estudiado en **Bases de Datos**, este lenguaje ha sido clave para crear nuestras tablas y para solicitar a la base de datos los datos pertinentes al servidor.
- Como extra he utilizado la librería **FullCalendar(v6.1.5)** de JavaScript. Esta librería reduce enormemente la codificación en la creación de un calendario. Tiene como función principal mostrar un calendario totalmente personalizable, con posibilidad de tener distintas vistas (mensual, semanal, diario), a su vez dispone métodos que nos permite almacenar, mostrar, editar y eliminar eventos creados por el usuario. El uso que le he dado es bastante básico para la magnitud de la librería. Explicaré en más detalle cómo funciona en el apartado de diseño.

Las **herramientas** utilizadas son las siguientes:

- **XAMPP** (v3.3.0) herramienta utilizada en **Despliegue de aplicaciones y entorno Servidor**, la cual dispone de un servidor *Apache* para poder lanzar la aplicación y del SGBD MySQL, donde desde el panel *PHPMyAdmin* hemos creado la base de datos para la aplicación.
- **Visual Studio Code**, IDE utilizado en varias asignaturas como **Programación** o **Entorno cliente**, lo utilizaremos como editor de código para nuestra aplicación.
- **Brave**, navegador principal para probar y visualizar los avances de la aplicación.

Otras herramientas utilizadas:

- **Boxicons:** librería de iconos, los cuales nos aportarán un aspecto más moderno a la aplicación.
- **Google-Fonts:** librería basada en CSS para dar estilo al texto de nuestra aplicación.
- **Trello:** herramienta en línea para la realización de *diagrama de Gantt* y tarjetas visuales (Kanban)
- **Figma:** herramienta en línea para la creación de los *wireframes* y guía de estilo.
- **Diagrams.net:** herramienta en línea para la realización de los diagramas de casos de uso, de clases y E-R.

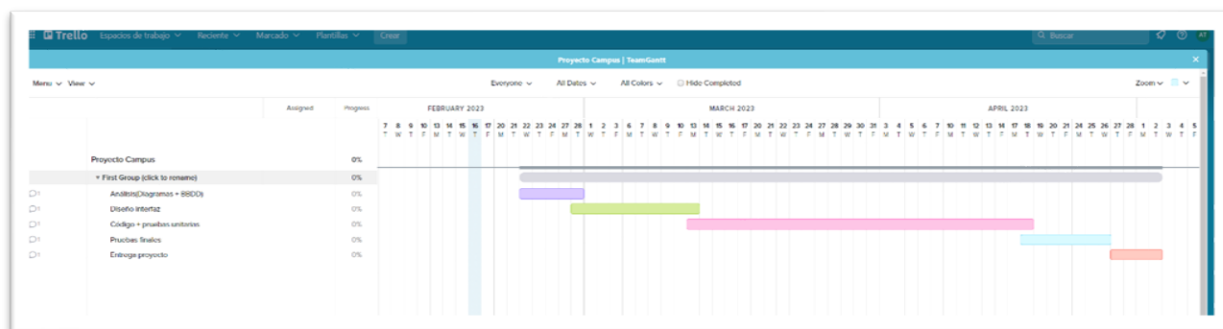


## 4. Estimación de recursos y planificación

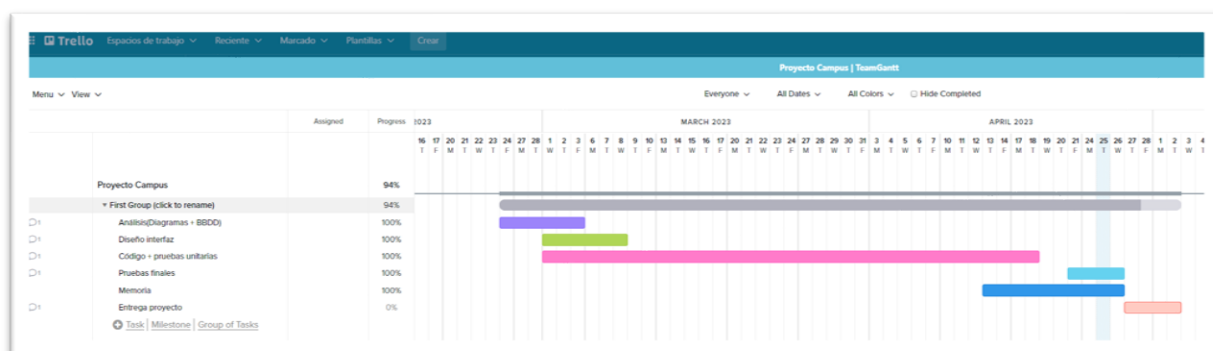
Para la estimación de recursos hemos realizado un *diagrama de Gantt*, donde teniendo en cuenta que sólo disponíamos de escasos dos meses (70 días desde la apertura para la entrega de la propuesta hasta el último día de la entrega del proyecto) hemos distribuido las fases del proyecto más o menos acorde al tiempo que se requerirá:

- Análisis: 7-8% del tiempo total dedicado.
- Diseño: 18%.
- Código + Pruebas unitarias: 58%
- Pruebas finales (caja negra): 10%.
- Documentación + entrega: 6%.

Es una estimación aproximada, pero claramente la fase de desarrollo de código y pruebas unitarias es la que más tiempo nos va a llevar, ya que es en la que más dificultades y errores vamos a encontrar.



Finalmente, los tiempos reales los hemos representado en otro diagrama de Gantt:



Podemos observar que inicialmente tenía previsto empezar a codificar a partir del 13 de marzo, pero viendo el poco tiempo del que disponía y el proyecto me iba a abarcar mucho trabajo decidí empezar a codificar pequeños bloques de código al mismo tiempo que diseñaba la interfaz.

## 5. Análisis del proyecto

### Requisitos Funcionales:

- El usuario podrá registrarse introduciendo su DNI (que será su *Primary Key* en la BBDD), Nombre, apellidos, correo electrónico, contraseña, seleccionará de un listado el curso a realizar y el tipo de usuario que será.
- Una vez registrado en la base de datos, el usuario podrá iniciar sesión mediante un *login* introduciendo su correo electrónico y su contraseña.
- Según el tipo de usuario conectado tendrá acceso a unos módulos u otros, por lo que el menú para acceder a dichos módulos será distinto para cada tipo de usuario.
- El usuario administrador tendrá acceso a la creación, edición y eliminación de todos los datos relacionados con cursos, usuarios y asignaturas.
- El usuario administrador y profesor podrán crear eventos y que sean visibles para todos los usuarios.
- Todos los usuarios podrán enviar y recibir mensajes.
- Todos los usuarios disponen de un módulo para visualizar los datos del perfil y editarlos en caso de cambios.
- El usuario profesor únicamente podrá editar las notas de sus alumnos, solamente a los que pertenecen al curso al que pertenece el profesor.
- El usuario profesor solo podrá consultar los datos relacionados con cursos, usuarios y asignaturas.
- El usuario profesor podrá subir documentos *pdf* y enlaces de terceros en el módulo de asignaturas.
- El usuario alumno podrá visualizar los eventos creados.
- El usuario alumno será el único que podrá visualizar su expediente con las asignaturas del curso realizado y sus respectivas notas.
- El usuario alumno podrá visualizar los documentos *pdf* y enlaces puestos por el profesor.
- Todos los usuarios disponen de la opción de poner un “modo oscuro” en la aplicación.

**Requisitos No Funcionales:**

- El formulario de registro tiene una validación de datos.
- Todas las consultas a la base de datos se realizarán vía programación orientada a objetos (PDO), con consultas preparadas y el uso de la función *bindParam()*, con el fin de proteger la web contra inyecciones SQL.
- Todos los CRUD son realizados únicamente por el usuario administrador, con el fin de evitar modificaciones no deseadas en la base de datos.
- Si se intenta acceder a zonas donde el usuario no tiene permiso será redirigido al inicio.
- Si se intenta acceder a la aplicación sin iniciar sesión, el usuario será redirigido al *login*.
- El registro de usuarios, la edición del perfil y la creación de usuarios por parte del administrador tendrán un sistema de encriptado de contraseñas mediante la función *password\_hash()* de PHP.
- La creación del fichero *.htaccess* (HyperText Access), con la directiva "*Options All -Indexes*", la cual impide el acceso desde el servidor a un listado de directorios sin haber especificado un fichero previamente.

**Diagrama Entidad-Relación**

*[Imagen adjunta en los anexos.]*

**Diagramas Casos de uso**

*[Imágenes adjuntas en los anexos.]*

**Diagrama de Clases**

*[Imagen adjunta en los anexos.]*

## 6. Diseño del proyecto

### 6.1. Base de Datos

La base de datos inicialmente la empecé a crear desde *MySQL Workbench*, sincronizándola con *PHPMyAdmin* mediante la asignación del mismo puerto (3306).

El problema vino cuando, al no utilizar un control de versiones, en el momento que el *Workbench* recibía una actualización, el MySQL de Xampp dejaba de funcionar, mostrándome un error de incompatibilidad de versiones, lo que me obligaba a tener que estar haciendo *backups* de la BBDD cada cierto tiempo. Finalmente decidí continuar creándola directamente desde PHPMyAdmin.

Como mencioné anteriormente, la metodología que empleé fue la interactiva incremental, por lo que en cada incremento fui creando la tabla pertinente al módulo.

Finalmente, y guiándome con el diagrama E-R, cree una BBDD llamada “Campus” con 8 tablas:

- **Usuarios**, tabla con 9 campos, donde se recogen los datos personales del usuario, el tipo de usuario, y dos de los campos que por falta de tiempo están orientados a vías futuras para validar el usuario mediante correo y *tokenizar* la sesión.
- **Módulos**, tabla con 2 campos donde almacena el nombre de los estudios impartidos junto a un identificador.
- **Asignaturas**, tabla con 5 campos donde se almacenan las asignaturas junto a un identificador, y como *foreign key* el identificador del módulo al que pertenecen.
- **Mensajes**, tabla con 7 campos donde almacenamos el título y el mensaje creado, junto a un identificador, una fecha y como *foreign key* los identificadores del usuario emisor y el usuario receptor del mensaje.
- **Eventos**, tabla con 6 campos, donde almacenamos los datos necesarios a los eventos, fecha de inicio, fecha fin, el título, descripción del evento junto a un identificador y como *foreign key* el identificador del usuario creador del evento.
- **Notas**, tabla con 4 campos que almacena la nota junto a un identificador y como *foreign key* el identificador de la asignatura y el identificador del usuario a la que pertenece la nota.
- **Archivos**, tabla con 6 campos donde almacenamos los datos del archivo subido, don un identificador y como *foreign key* el identificador de la asignatura a la que pertenece.
- **Enlaces**, tabla de 4 campos donde junto al identificador almacenamos el título, la *url* y el identificador de la asignatura a la que pertenece el enlace como *foreign key*.

[imagen adjunta en los anexos.]

## 6.2. Prototipos

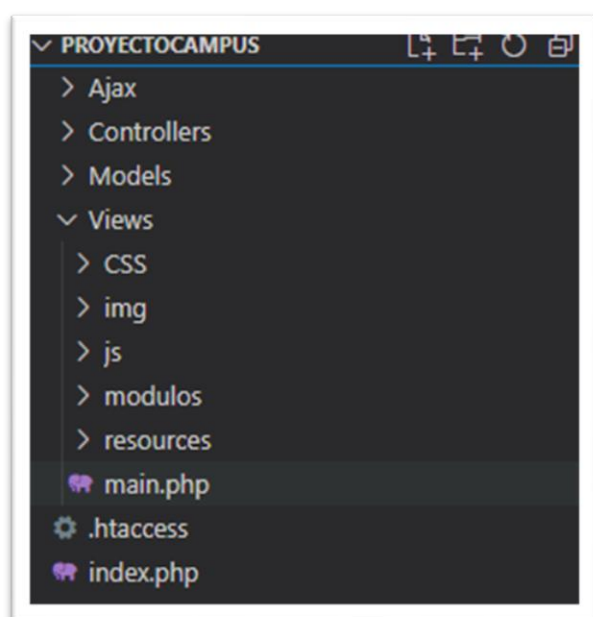
[Imágenes adjuntas en los anexos.]

## 6.3. Guía de estilo

[Imágenes adjuntas en los anexos.]

## 6.4. Estructura del proyecto (MVC)

La estructura de carpetas del proyecto es la que vemos en la imagen de la izquierda. Trabajamos con el patrón Modelo Vista Controlador (MVC), separando en tres capas la codificación del proyecto. Este patrón es habitual utilizarlo en aplicaciones donde se requiere el uso de interfaces de usuario, por lo tanto, es idóneo para nuestro proyecto. A parte de las tres carpetas concernientes al MVC, hemos creado una carpeta para los ficheros AJAX, el fichero *.htaccess* y por último el iniciador de la aplicación (*index.php*). En el Controlador (“*Controllers*”) vamos a encontrar ficheros *.php* que van a ser los encargados de tomar los datos pertinentes del lado cliente. En el Modelo (“*Models*”) encontraremos ficheros *.php* donde recogeremos los datos del controlador para realizar la sentencia SQL necesaria a la base de datos y recibir una respuesta por parte del servidor. Finalmente, en la Vista (“*Views*”) tenemos varias carpetas todas relacionadas con lo concerniente al lado cliente. Empezando por orden ascendente:



- “CSS”: incluye dos hojas de estilo, una destinada a la interfaz del registro y del login y la otra para la aplicación principal.
- “img”: carpeta donde se han incluido las imágenes utilizadas para la aplicación.
- “js”: donde encontraremos 4 ficheros js, uno para el registro, otro para la aplicación principal, y los dos restantes están vinculados al calendario, uno para la traducción en español y el otro con los métodos y eventos necesarios para su correcto funcionamiento.
- “módulos”: aquí es donde vamos a encontrar cada una de las vistas principales de la aplicación.
- “resources”: es donde se almacenarán los archivos almacenados por el usuario profesor y administrador para los alumnos.
- *main.php*: es el cuerpo de la aplicación, donde irán contenidas las vistas.



que contenga al menos una minúscula, una mayúscula, un dígito, un carácter especial y por supuesto sin espacios en blanco, así damos lugar a que los usuarios tengan contraseñas seguras,

además para asegurarnos de que el usuario introduce la contraseña que desea, ésta tendrá una validación donde tendrá que introducirla de nuevo. Mediante el método de JS *addEventListener* tomará el evento al hacer “submit” en el formulario donde validará las expresiones regulares con lo introducido por el usuario, si los datos introducidos son incorrectos mostrará un mensaje proporcionando una ayuda al usuario para introducir correctamente los

datos (imagen de la izquierda), si son correctos los datos el formulario se enviarán con éxito al controlador(*usuariosC.php*).

Dentro del método crearemos una variable(*\$hash*) que almacenará la contraseña de manera encriptada gracias al método de PHP “*password\_hash*” que podemos encontrar en la documentación oficial de PHP (incluida en la bibliografía de este documento). También captaremos en variables el nombre de la tabla a la que queremos almacenar los datos(*\$tablaBD*) y la variable *\$datos* que, mediante un array, almacenará todos los datos introducidos por el usuario.

```
public function RegistrarUsuarioC(){
    if(isset($_POST['userID'])){
        //Encriptamos la contraseña mediante la función de PHP password_hash
        $pass = $_POST['contraseña'];
        $hash = password_hash($pass, PASSWORD_DEFAULT);
        $tablaBD = "usuario";
        $datos = array("userID" => $_POST["userID"],
            "contraseña" => $hash,
            "nombre" => $_POST["nombre"],
            "apellidos" => $_POST["apellidos"],
            "correo" => $_POST["correo"],
            "moduloID_FK" => $_POST["moduloID"],
            "tipoUsuario" => $_POST["tipoUsuario"]);
        $result = UsuariosM::RegistrarUsuarioM($tablaBD,$datos);

        if($result == true){
            echo <script>
                window.location = "http://localhost/ProyectoCampus/login";
            </script>;
        }
    }
}
```

Los datos recogidos en el controlador serán enviados al modelo mediante la llamada al método “*RegistrarUsuarioM()*” situada dentro de la clase “*UsuariosM*”. Dentro de esta función crearemos la variable *\$pdo* la cual creará la conexión a la base de datos mediante el método ya creado

```
public static function RegistrarUsuarioM($tablaBD,$datos){
    $pdo = Conexion::conexionBD() -> prepare("INSERT INTO $tablaBD (userID, contraseña, nombre, apellidos, correo, tipoUsuario, moduloID_FK)
    VALUES(userID,:contraseña,:nombre,:apellidos,:correo,:tipoUsuario,:moduloID_FK)");
    $pdo -> bindParam("userID", $datos["userID"], PDO::PARAM_STR);
    $pdo -> bindParam("contraseña", $datos["contraseña"], PDO::PARAM_STR);
    $pdo -> bindParam("nombre", $datos["nombre"], PDO::PARAM_STR);
    $pdo -> bindParam("apellidos", $datos["apellidos"], PDO::PARAM_STR);
    $pdo -> bindParam("correo", $datos["correo"], PDO::PARAM_STR);
    $pdo -> bindParam("tipoUsuario", $datos["tipoUsuario"], PDO::PARAM_STR);
    $pdo -> bindParam("moduloID_FK", $datos["moduloID_FK"], PDO::PARAM_INT);
    if($pdo -> execute()){
        return true;
    }
    $pdo -> close();
    $pdo = null;
}
```

“*conexionBD()*” y preparará la sentencia SQL, en este caso para insertar los datos introducidos por el usuario a la base de datos. Mediante *bindParam* vincularemos cada parámetro de la sentencia al dato enviado en el array seguido del tipo de dato que

recogeremos (PDO::PARAM\_STR si es un *string*, PDO::PARAM\_INT si es un *integer*), si se ejecuta la función retornaremos un true, cerraremos conexión y vaciamos la variable *\$pdo*. Si retorna true recogeremos la respuesta del servidor en la variable *\$result* situada en la función del controlador, si retorna true nos enviará al *login* para poder iniciar sesión.



## Login:

Como hemos mencionado anteriormente una vez correcto el registro del usuario nos enviara

directamente a un formulario(*login.php*) donde precisaremos introducir el correo electrónico y la contraseña que hemos registrado. Tras enviar los datos llamamos al método (*IniciarSesionC()*) creado en el controlador (*usuariosC.php*) donde recogerá los datos introducidos por el usuario, en este caso solo enviaremos como dato el correo (ya que la contraseña al estar encriptada necesitaremos del método *password\_verify*) y la variable para la

tabla, solo enviamos el correo porque en la respuesta del servidor ejecutaremos la función *fetch()* para que nos devuelva toda la fila del registro correspondiente al correo enviado.

```
//Inicio Sesion
public function IniciarSesionC(){
    if(isset($_POST['correo'])){
        $tablaBD = "usuario";
        $datos = $_POST['correo'];

        $result = UsuariosM::IniciarSesionM($tablaBD, $datos);

        if($result['correo'] == $_POST['correo'] && password_verify($_POST['contrasena'],$result['contrasena'])){
            $_SESSION['IniciarSesion'] = true;
            $_SESSION['tipoUsuario'] = $result['tipoUsuario'];
            $_SESSION['userID'] = $result['userID'];
            $_SESSION['nombre'] = $result['nombre'];
            $_SESSION['apellidos'] = $result['apellidos'];
            $_SESSION['correo'] = $result['correo'];
            $_SESSION['contrasena'] = $result['contrasena'];
            $_SESSION['moduloID_fk'] = $result['moduloID_fk'];
            echo "<script>
                window.location = 'http://localhost/ProyectoCampus/inicio';
            </script>";
        }else{
            echo 'El correo o la contraseña con incorrectos';
        }
    }
}
```

```
//Inicio Sesion
public static function IniciarSesionM($tablaBD, $datos){
    $pdo = Conexion::conexionBD() -> prepare("SELECT * FROM $tablaBD WHERE correo = :correo");
    $pdo -> bindParam(":correo", $datos, PDO::PARAM_STR);
    $pdo -> execute();
    return $pdo -> fetch();
    $pdo -> close();
    $pdo = null;
}
```

Una vez retornado los datos haremos una doble comprobación de que el resultado(\$result) sea igual que los valores introducidos por el usuario, si es correcto crearemos las variables de sesión(\$\_SESSION) con los datos registrados por el usuario. En el caso de que los datos introducidos sean incorrectos nos mostrará un mensaje alertando del error.

He considerado el uso de las variables de sesión en lugar de las cookies, ya que al tratarse de información confidencial de datos personales es más seguro mantener dichos datos almacenados en el servidor.

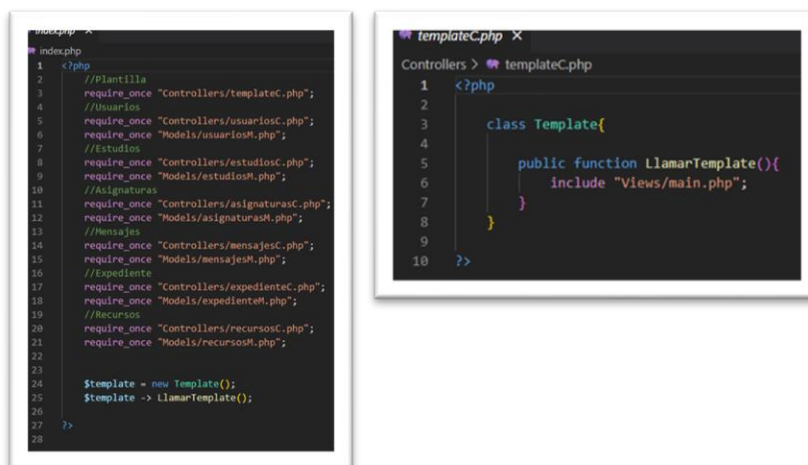


## 6.6. index/templateC/main.php

Una vez *logueado* el usuario tenemos 3 ficheros importantes para la ejecución de la aplicación en el lado del cliente.

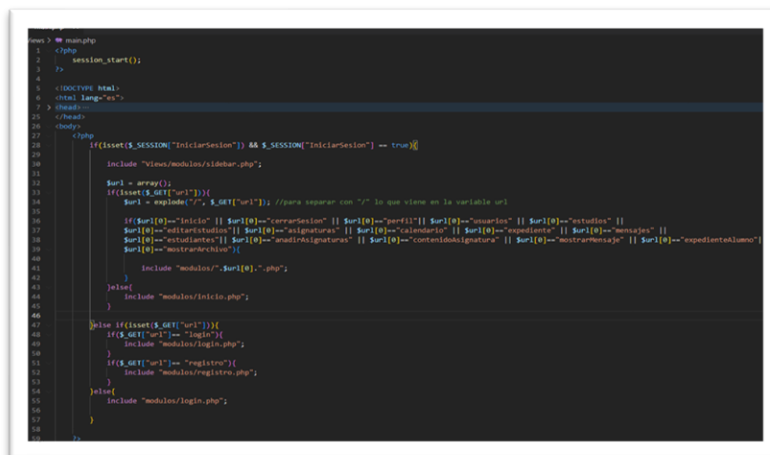
### index.php y templateC.php:

Este fichero es el lanzador de la aplicación, el que va a almacenar todas las llamadas a los ficheros del controlador y modelo mediante *require\_once* y también en la que crearemos el objeto de la clase *Template* en la que llamará a la función (*LlamarTemplate()*) situado en el fichero *templateC.php* donde llamaremos a la plantilla que hemos creado para la aplicación(*main.php*).



### main.php:

En este fichero es donde iniciaremos la función PHP *session\_start()* y donde mostraremos en el cuerpo de la aplicación todas las vistas creadas.

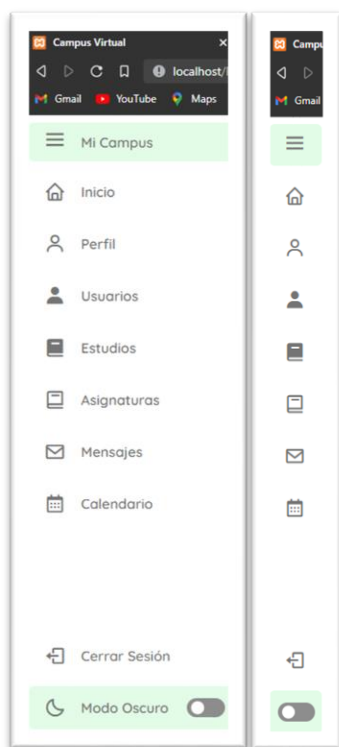


Para evitar que se pueda acceder a la aplicación sin estar *logueado* hacemos una comprobación(línea 28) de que solamente será visible el cuerpo de la aplicación si existe la variable de sesión `$_SESSION['iniciarSesion']`(explicada anteriormente en el *login.php*) y que además ésta

sea verdadera, en caso de no cumplirse una de las dos condiciones nos enviará directamente al *login* e incluso introduciendo directamente en la url la vista a la que queramos acceder, de esta forma aseguramos accesos maliciosos a la aplicación.

Una vez cumplidas ambas condiciones para el inicio de sesión incluiremos dos ficheros clave, uno de ellos será el menú(*sidebar.php*) que estará visible en toda la aplicación y los módulos para optimizar el código y evitar código “*espagueti*” los hemos almacenado en un array dentro de la variable `$url[ ]`, en la cual hemos empleado el método de PHP *explode()* para que mediante “/” separemos cada elemento de la *url*, en este caso los módulos estarán ubicados como el primer elemento (`$url[0]`).

## 6.7. Sidebar



En el lado izquierdo podemos ver un ejemplo de cómo está estructurado el menú visto desde el lado cliente, para la creación de éste hemos hecho uso de *Bootstrap5*. Está compuesto por enlaces

```
<li class="nav-link">
  <a href="http://localhost/ProyectoCampus/perfil">
    <i class="bx bx-user icon"></i>
    <span class="text nav-text">Perfil</span>
  </a>
</li>
```

que nos mostrará en la parte central el módulo seleccionado (Ejemplo de enlace en la imagen de arriba). Para que el menú dispusiese de interactividad, hemos hecho que tuviese dos vistas opcionales, una del menú desplegado y otra del menú reducido donde sólo se ven los iconos. Para crear esa animación hemos hecho uso de eventos en JavaScript aplicando la clase “close” *header* del menú cuando hacemos *click* sobre él y eliminándola cuando volvemos a *clickar*.

En la parte inferior podemos ver la opción de cerrar sesión y un interruptor que nos servirá para cambiar de modo claro a modo oscuro con un simple click sobre él. Al pulsar sobre Cerrar Sesión nos llevará a la función que destruirá la sesión(*session\_destroy*), redirigiéndonos al *login*.

```
cerrarSession.php
1 <?php
2
3 session_destroy();
4
5 echo '<script>
6   window.location = "http://localhost/ProyectoCampus/login";
7 </script>'
8
9 >
```

## 6.8. Módulos (*body*)

Aquí vamos a mostrar un poco más en detalle cada una de las funcionalidades principales que hemos creado para que nuestra aplicación cumpla con los objetivos que nos hemos propuesto.

En lo referente al CRUD, para evitar repetir contenido muy similar, mostraré un ejemplo de cada tipo de ejecución (*Insert, Select, Update y Delete*), ya que, variando por los datos enviados por parte del controlador, la estructura de las funciones es muy similar.

### 6.8.1. Perfil

La finalidad de este módulo es mostrar los datos del usuario conectado y a su vez poder editarlos. En el caso de la contraseña no mostramos el resultado por seguridad. Es una vista bastante sencilla e intuitiva a la vez que útil para que la institución que utilice la aplicación pueda tener los datos del usuario actualizados.

Los archivos implicados en el módulo:

- **Modelo:** usuariosM.php.
  - ✓ **Métodos:** MostrarDatosM(), ActualizarDatosM().
- **Controlador:** usuariosC.php.
  - ✓ **Métodos:** MostrarDatosC(), ActualizarDatosC().
- **Vista:** perfil.php.

## 6.8.2. Usuarios

Este módulo está configurado para solo ser visible para los roles de “*admin*” y “*profesor*”. Se trata de un panel compuesto por una tabla donde muestra todos los usuarios registrados, mostrando datos como el DNI, nombre, apellidos, correo electrónico, estudios y el tipo de usuario.

Archivos implicados en el módulo:

- **Modelo:** usuariosM.php.
  - ✓ **Métodos:** CrearUsuarioM(), MostrarUsuarioM(), ActualizarUsuarioM(), EliminarUsuarioM().
- **Controlador:** usuariosC.php.
  - ✓ **Métodos:** CrearUsuarioC(), MostrarUsuarioC(), ActualizarUsuarioC(), EliminarUsuarioC().
- **Vista:** usuarios.php.
- **Ajax:** EditarUsuariosA()

En caso de que el usuario sea de tipo “*profesor*” sólo tendrá acceso a visualizar las tablas a modo informativo.

DNI Usuario	Nombre	Apellidos	Correo Electrónico	Estudios	Tipo de Usuario
000000000	admin	admin	admin@admin.com	Usuario Administrador	Admin
11111111W	Joana	Perez Molina	joanaperez@hotmail.com	FPS Desarrollo Aplicaciones Web	Alumno
12121212f	Meme	meme	meme@meme.com	FPS Desarrollo Aplicaciones Web	Alumno
22222222K	Pedro Jaime	Jimenez Lopez	pedrojimenez@hotmail.com	FPS Desarrollo Aplicaciones Web	Profesor
33333333K	Profesor	Profesor	profe@profe.com	FPM Sistemas Microinformáticos y Redes	Profesor

Para el caso del usuario administrador va a poder realizar funciones CRUD, dejando a su disposición botones para añadir y editar que muestran un modal con un formulario y un botón para eliminar usuarios.

Gestor de Usuarios

Añadir Usuario

DNI Usuario	Nombre	Apellidos	Correo Electrónico	Estudios	Tipo de Usuario	Acciones
000000000	admin	admin	admin@admin.com	Usuario Administrador	Admin	 
11111111W	Joana	Perez Molina	joanaperez@hotmail.com	FPS Desarrollo Aplicaciones Web	Alumno	 
12121212f	Meme	meme	meme@meme.com	FPS Desarrollo Aplicaciones Web	Alumno	 
22222222K	Pedro Jaime	Jimenez Lopez	pedrojimenez@hotmail.com	FPS Desarrollo Aplicaciones Web	Profesor	 
33333333K	Profesor	Profesor	profe@profe.com	FPM Sistemas Microinformáticos y Redes	Profesor	 

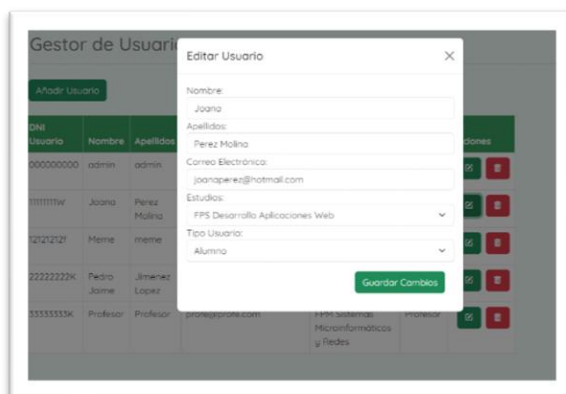
Este es un buen ejemplo para mostrar lo que hemos creado en el controlador y en el modelo porque además hemos hecho uso de AJAX/JQuery para poder mostrar en los modales los datos relacionados al registro(en el caso de editar) y el uso de JQuery para eliminar el registro.



conexión a la base de datos mediante el método estático *conexionBD()* de la clase Conexión y la almacenaremos en la variable *\$pdo*, acto seguido preparamos la sentencia SQL para insertar los datos y mediante *bindparam* vincularemos cada parámetro de la sentencia al dato del array seguido del tipo de dato que recogeremos. Ejecutamos la sentencia, cerramos la conexión y vaciamos la variable *\$pdo*.

Para **editar el usuario** como mencionamos anteriormente, también utilizaremos un modal.

En este caso podemos ver que se trata también de un formulario, pero esta vez con los datos del registro al que pertenece el modal.



El formulario en código es igual que para el caso de añadir usuarios, pero con un identificador diferente y añadiendo un identificador a cada input del formulario con el fin de poder utilizar dichos ID para asignarles unos valores mediante AJAX y JQuery.



En el lado izquierdo podemos ver el evento JQuery en el que al hacer “click” en el elemento con la clase “*.EditarUsuario*”(clase que hemos añadido al botón de editar), nos crea dos variables, una será donde



almacenaremos el atributo creado en el boton cuyo valor es el userID del registro y la variable “datos” que almacenará un nuevo objeto FormData() y le añadirá el valor “uID”.

A continuación, se realiza una petición AJAX al fichero *usuariosA.php* con el método POST y se establece el parámetro data al objeto FormData creado. Desde el fichero

*usuariosA.php* crearemos función *EditarUsuariosA()* donde almacenaremos las variables *\$columna* (la cual tomará el valor de la columna de la tabla necesaria userID) y *\$valor* la cual tomará el valor enviado mediante el objeto FormData(el atributo uID).

Las variables serán enviadas mediante la función estática *MostrarUsuariosC(\$columna,\$valor)* del controlador la cual enviará a su vez los datos al modelo añadiendo como parametro la variable

\$tablaBD.

Al enviarle valores no nulos en la variable \$columna ejecutaremos la primera condición, la cual devolverá los datos del registro al cual equivale la variable \$valor.

```

//Mostrar Usuarios
public static function MostrarUsuarios($columna, $valor){
    $tablaBD = "usuario";
    $result = UsuariosM::MostrarUsuariosM($tablaBD, $columna, $valor);
    return $result;
}

```

```

//Mostrar Usuarios
public static function MostrarUsuariosM($tablaBD, $columna, $valor){
    if($columna != null){
        $pdo = Conexion::conexionBD() -> prepare("SELECT * FROM $tablaBD WHERE $columna = :columna");
        $pdo -> bindParam(":columna", $valor, PDO::PARAM_STR);
        $pdo -> execute();
        return $pdo -> fetch();
    }else{
        $pdo = Conexion::conexionBD() -> prepare("SELECT * FROM $tablaBD");
        $pdo -> execute();
        return $pdo -> fetchAll();
    }
    $pdo -> close();
    $pdo = null;
}

```

Una vez todo ejecutado correctamente, damos los pasos hacia atrás, la respuesta del modelo que almacenará en la variable \$result del controlador que, a su vez se almacenará en formato JSON en la variable \$result de usuariosA.php. Los datos almacenados en el JSON los asignaremos como valores de los inputs del formulario del modal.

Por último, para el botón de eliminar registros, le asignaremos un atributo que almacene el valor del userID del registro y la clase ".EliminarUsuario".

```

<a href="#">
    <button class="btn btn-danger EliminarUsuario" uID="'. $value["userID"].'"><i class="bx '(.bx-trash)'."></i></button>
</a>

```

Mediante un simple código JQuery tomaremos el valor del atributo al hacer click sobre el botón, valor que nos servirá para enviar al controlador desde la url y así poder hacer desde el modal una sentencia SQL (Delete) donde le pasaremos por parámetro el atributo captado.

```

//Eliminar datos usuario
$( ".f" ).on( "click", ".EliminarUsuario", function() {
    let uID = $(this).attr("uID");

    window.location = "index.php?url=usuarios&uID="+ uID;
});

```

```

//Eliminar Usuario
public function EliminarUsuarioC(){
    if(isset($_GET["uID"])){
        $tablaBD = "usuario";
        $id = $_GET["uID"];
        $result = UsuariosM::EliminarUsuariosM($tablaBD,$id);
        if($result== true){
            echo '<script>
                window.location = "http://localhost/ProyectoCampus/usuarios"
            </script>';
        }
    }
}

```

```

//Eliminar Usuario
public static function EliminarUsuariosM($tablaBD,$id){
    $pdo = Conexion::conexionBD() -> prepare ("DELETE FROM $tablaBD WHERE userID = :userID");
    $pdo -> bindParam(":userID", $id, PDO::PARAM_STR);
    if($pdo -> execute()){
        return true;
    }
    $pdo -> close();
    $pdo = null;
}

```



### 6.8.3. Gestor de Estudios

Este módulo tiene varias finalidades, la gestión de los estudios impartidos por la institución que utilice la aplicación, gestionar las asignaturas de cada estudio, mostrar los alumnos pertenecientes a cada estudio y calificarlos.

El usuario administrador tendrá habilitadas las acciones de añadir, editar y eliminar estudios, sin embargo, el usuario profesor sólo podrá visualizar la lista de estudios. Además, tanto el administrador como profesores podrán acceder a dos vistas, la de las asignaturas que pertenecen al estudio listado en la tabla y a los alumnos matriculados en él.

La vista de las asignaturas por estudio será solo modo lectura para profesores, pero, para el usuario administrador tendrá habilitadas las acciones para añadir, editar y eliminar asignaturas.

La vista de los alumnos mostrará el DNI y el nombre completo del alumno y como acción, tanto profesores como alumnos, se accederá al expediente del alumno, lugar donde se le podrá calificar al alumno por asignatura. En caso de ya estar calificado, pero queramos modificar la nota tendremos la opción de editarla, si por el contrario el alumno aun no está calificado de una asignatura aparecerá como “No evaluado” y tendrá la opción de añadir una nota. *[imágenes de las vistas en anexos]*

Archivos implicados en el módulo:

- **Modelo:** estudiosM.php(panel Estudios), asignaturasM.php(panel Asignaturas), expedienteM()(calificar alumnos).
  - ✓ **Métodos:** CrearEstudiosM(), MostrarEstudiosM(), EditarEstudiosM(), ActualizarEstudiosM(), EliminarEstudiosM(), anadirAsignaturaM(), MostrarAsignaturaM(), editarAsignaturaM(), EliminarAsignaturaM(), MostrarExpedienteM(), anadirNotaM(), editarNotaM().
- **Controlador:** estudiosC.php(panel Estudios), asignaturasC.php(panel Asignaturas), expedienteC()(calificar alumnos).
  - ✓ **Métodos:** CrearEstudiosC(), MostrarEstudiosC(), EditarEstudiosC(), ActualizarEstudiosC(), EliminarEstudiosC(), anadirAsignaturaC(), MostrarAsignaturaC(), editarAsignaturaC(), EliminarAsignaturaC(), MostrarExpedienteC(), anadirNotac(), editarNotac().
- **Vista:** estudios.php, anadirAsignaturas.php(botón Asignaturas), estudiantes.php(botón estudiantes), expediente.php(botón dentro de estudiantes)



#### 6.8.4. Asignaturas

Este módulo está principalmente orientado para profesores y alumnos (aunque los administradores también tienen acceso a él). Es el espacio donde ambos usuarios pueden visualizar un listado de las asignaturas que se imparten por semestres, de cada asignatura va a ser posible acceder a una vista (*contenidoAsignatura*) donde el usuario Profesor podrá subir recursos (ficheros en pdf y enlaces) de interés para sus alumnos, y éstos poder visualizarlos desde la misma aplicación en caso de ser pdf o verlo en otra pestaña del navegador en caso de ser un enlace.

Archivos implicados en el módulo:

- **Modelo:** asignaturasM.php, recursosM.php.
  - ✓ **Métodos:** MostrarAsignaturasM(), MostrarAsignaturasUserM(), mostrarArchivosM(), subirArchivosM(), eliminarArchivosM(), mostrarEnlacesM(), subirEnlacesM(), eliminarEnlacesM().
- **Controlador:** asignaturasC.php, recursosC.php
  - ✓ **Métodos:** MostrarAsignaturasC(), MostrarAsignaturasUserC(), mostrarArchivosC(), subirArchivosC(), eliminarArchivosC(), mostrarEnlacesC(), subirEnlacesC(), eliminarEnlacesC().
- **Vista:** asignaturas.php, contenidoAsignatura.php.

De este módulo destacaré el método subirArchivos() ya que tuve que investigar cómo podría subir ficheros a la base de datos y que se me lo almacenasen en la carpeta “resources” situada dentro de “Views”. Descubrí desde la documentación oficial que PHP dispone de una variable para específica para subir ficheros HTTP, la variable *superglobal* \$\_FILES[], se trata de un array donde almacena los elementos relacionados con el fichero en cuestión.

\$\_FILES[name, type, tmp\_name, error, size]

En el [name] almacena el nombre (más extensión) del archivo (“ejemplo.pdf”),

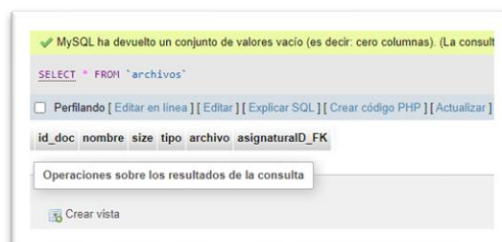
En el [type] almacena el tipo de fichero (“application/pdf”),

En el [tmp\_name] almacena la ruta temporal del archivo (/tmp/php/php432),

En el [error] almacena el mensaje de error UPLOAD\_ERR\_OK,

Y en [size] almacena el tamaño del archivo en bytes.

Sabiendo que para trabajar con ficheros vamos a necesitar de esos datos, cree la tabla de la *bbdd* en relación con los datos que me aportaba la variable S\_FILES, el nombre, el tamaño y el tipo.



Mediante un formulario en un modal introduciremos dos sencillos inputs, uno destinado para el nombre personalizado del archivo y otro de tipo “file” necesario para subir un archivo. Para lograr que nuestro archivo se envíe satisfactoriamente es necesario introducir en la etiqueta de apertura del formulario el elemento ‘*enctype="multipart/form-data"*’.

```

<div class="modal fade" id="subirarchivos">
  <div class="modal-dialog">
    <div class="modal-content">
      <form method="post" enctype="multipart/form-data">
        <div class="modal-header">
          <div class="modal-title">Subir Archivo</div>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="close"></button>
        </div>
        <div class="modal-body">
          <div class="form-group">
            <input type="text" class="form-control" name="nombre">
          </div>
          <div class="form-group">
            <div class="form-group">
              <input type="file" class="form-control" name="archivo" id="archivo">
            </div>
          </div>
          <div class="form-group">
            <input type="hidden" class="form-control" name="asignaturaID_FK" value="{$exp[1]}">
          </div>
        </div>
        <div class="modal-footer">
          <button type="submit" class="btn btn-success" data-dismiss="modal">Subir Archivo</button>
        </div>
      </form>
      <div class="modal-body">
        <div class="form-group">
          <input type="text" class="form-control" name="nombre">
        </div>
        <div class="form-group">
          <div class="form-group">
            <input type="file" class="form-control" name="archivo" id="archivo">
          </div>
        </div>
        <div class="form-group">
          <input type="hidden" class="form-control" name="asignaturaID_FK" value="{$exp[1]}">
        </div>
      </div>
      <div class="modal-footer">
        <button type="submit" class="btn btn-success" data-dismiss="modal">Subir Archivo</button>
      </div>
    </div>
  </div>
</div>

```

Una vez enviados los datos introducidos por el usuario es el momento de dar paso al controlador, donde al pulsar sobre el *submit* hará una serie de verificaciones y enviará los datos al servidor.

```

//Subir archivos
public function subirArchivosC(){
  //variables para tomar el id de la asignatura por URL
  $exp = explode("/", $_GET["url"]);
  $aID = $exp[1];
  //Comprobamos si pulsamos "submit"
  if(isset($_POST["submit"])){
    //variable con la ruta y nombre del archivo destino
    $file = "Views/resources/".$_FILES['archivo']['name'];
    //verificamos para solo permitir archivos en PDF
    $verificaPdf = ['application/pdf'];
    if(!in_array($_FILES['archivo']['type'], $verificaPdf)){
      echo '<script>
      alert("Solo se permiten archivos PDF.");
      </script>';
      return;
    }
    //Movemos el archivo temporal a la ruta destino
    move_uploaded_file($_FILES['archivo']['tmp_name'],$file);
    echo '<script>
    alert("archivo subido");
    </script>';
    //Subimos los datos a la base de datos
    $tablaBD = "archivos";
    $datos = array("nombre" => $_POST["nombre"],
                  "size" => $_FILES['archivo']['size'],
                  "tipo" => $_FILES['archivo']['type'],
                  "archivo" => $_FILES['archivo']['name'],
                  "asignaturaID_FK" => $_POST["asignaturaID_FK"]);
    $result = RecursosM::subirArchivosM($tablaBD,$datos);
    if($result==true){
      echo '<script>
      window.location = "../contenidoAsignatura/'.$aID.'";
      </script>';
    }
  }
}

```

Lo primero que crearemos es la variable \$file donde almacenaremos la ruta donde queremos situar el archivo dentro de nuestra carpeta contenedora del proyecto. Lo siguiente será crear una variable para verificar que sólo subirá el usuario archivos pdf, para ello hacemos la condición donde \$\_FILES[type] será igual a ['application/pdf'], en caso de no ser así, mostrará un mensaje de error indicando al usuario del tipo de archivo permitido.

Una vez verificado el tipo de archivo es momento del método interno de PHP “*move\_uploaded\_file()*” donde le pasaremos por parámetros la ubicación temporal del archivo

(\$\_FILES\_['tmp\_name']) y la ubicación donde queremos tener el archivo (\$file). Si todo a funcionado con éxito procederemos a subir los datos a la base de datos.

### 6.8.5. Mensajes

Este módulo se trata de un sistema interno de mensajería entre usuarios. Donde el usuario podrá enviar un mensaje nuevo mediante un modal a cualquier usuario registrado, recibir y visualizar los mensajes y eliminarlos. Es una interfaz sencilla, pero con una pequeña animación que distingue un mensaje leído de un mensaje no leído.


- **Modelo:** mensajesM.php.
  - ✓ **Métodos:** enviarMensajeM(), mostrarMensajeM(), eliminarMensajeM(), verMensajeM(), mensajeLeidoM().
- **Controlador:** mensajesC.php
  - ✓ **Métodos:** enviarMensajeC(), mostrarMensajeC(), eliminarMensajeC(), verMensajeC(), mensajeLeidoC().
- **Vista:** mensajes.php, mostrarMensajes.php.

De este módulo me gustaría hacer mención a la animación empleada para saber si el mensaje a sido o no leído.

Primero de todo mencionar la *bbdd*, donde hemos creado la tabla “mensajes” en la cual almacenamos los siguientes datos:

id	titulo	mensaje	remitente	destinatario	fecha	leido
----	--------	---------	-----------	--------------	-------	-------

El campo “leído” almacenará por defecto el valor ‘0’. Por lo tanto, un usuario al recibir un mensaje en su bandeja de entrada verá lo siguiente:

Mensajes				
Nuevos mensajes				
Leído	Remitente	Título	Fecha y Hora	Acción
	Josana Torres Molina	Pruveta	2023-05-05 23:35:55	 
	edman adrian	Pruveta de feedback	2023-05-05 23:39:54	 
	edman adrian	Muestra de feedback 2	2023-05-25 23:35:55	 
	edman adrian	El plan solo	2023-04-08 17:34:24	 
	edman adrian	Pruveta Leído	2023-04-24 20:05:16	 

Como podemos observar hay mensajes con un doble-check (leídos) y un mensaje con una campana animada (no leído).

En el momento en el que el usuario visualiza el mensaje, desde el modelo haremos un “*update*”

[illegible]

```

public static function mensajeIdiom($tablaBD, $id){
    $pdo = Conexion::conexionBD(); -> prepare("UPDATE $tablaBD SET leido = '1' WHERE id = :id");
    $pdo -> bindParam(':id', $id, PDO::PARAM_INT);
    if($pdo -> execute()){
        return true;
    }
    $pdo -> close();
    $pdo = null;
}

```

donde el valor “leído” de la *bbdd* lo cambiaremos por el valor ‘1’, mostrando así el doble-check.

## 6.8.6. Calendario

Este módulo se trata de un calendario el cual el usuario Administrador y Profesor podrá crear, editar y eliminar eventos y el usuario Alumno únicamente podrá visualizar los eventos.

Archivos implicados en el módulo:

- **Modelo:** eventosM.php.
  - ✓ **Métodos:** mostrarEventoM(), crearEventoM(), editarEventoM(), eliminarEventoM().
- **Controlador:** eventosC.php
  - ✓ **Métodos:** mostrarEventoC(), crearEventoC(), editarEventoC(), eliminarEventoC().
- **Vista:** calendario.php.
- **Script:** fullCalendar.js, es.global.js.

Para este módulo tuve que hacer una investigación más exhaustiva ya que era la primera vez que trabajaba con ella, además la documentación oficial (anclada en Bibliografía) es bastante escueta, tuve que hacer muchas búsquedas en páginas como *Stackoverflow* y videos de *Youtube* para llegar a comprender su funcionamiento.

### Puesta en marcha:

De las varias opciones que tiene la librería para poner en marcha el calendario me decanté por la del uso de “*Script Tags*”, me pareció la opción más sencilla ya que solo es necesario introducir un par de <script> en el <head>, uno para el calendario en sí y el otro necesario para los formatos de las fechas.

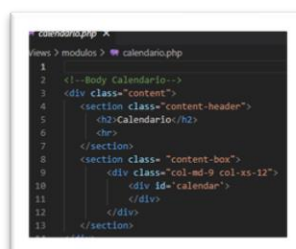
```
<script src='https://cdn.jsdelivr.net/npm/fullcalendar@6.1.5/index.global.min.js'></script>
<script src='https://cdn.jsdelivr.net/npm/moment@2.27.0/min/moment.min.js'></script>
```

Por otro lado, la inserción de dos ficheros .js, uno en el que haremos todos los manejos del DOM (FullCalendar.js) y otro predeterminado de la librería para su traducción al español (es.global.js).



La imagen situada en la izquierda corresponde al fichero fullCalendar.js, ese fragmento de código es el proporcionado por la librería para mostrarlo en nuestro módulo. En él crea la variable “calendarEl” donde toma el elemento con el id “calendar” correspondiente a la siguiente imagen,

la cual pertenece al fichero calendario.php (vista).



Posteriormente se crea otra variable “calendar” donde se crea el objeto de la librería y toma por parámetro la variable anterior y de ella insertaremos todas las opciones de configuración que hemos seleccionado para nuestra aplicación.

- **initialView:** ‘dayGridMonth’, Aquí especificamos el tipo de vista predeterminada que queremos mostrar, en este caso que muestre el calendario mensual.
- **Locale:** ‘es’, Aquí especificamos las siglas “es” para que muestre el calendario en español.
- **headerToolbar:** { }, Aquí podemos personalizar como queremos la barra de herramientas posicionando (*right*, *left* y *center*) donde queremos cada botón. En mi caso en el lado derecho situé los botones para poder seleccionar los tipos de vista del calendario (mensual, semanal y diario) en el centro el título y en el izquierdo los botones para pasar al mes anterior/siguiente o que se posicione en el día actual.
- **Height**, para adaptar el tamaño del calendario a nuestra aplicación le puse un ancho de 650px.
- **Events**, esta opción es la que recoge los eventos como un array, como queremos que recoja los eventos de nuestra *bbdd*, le insertaremos la ruta “*Controller/eventosC.php?accion=lista*” que es la encargada de mostrar los eventos en formato JSON. Explicaré más en detalle el controlador.
- **dateClick()**, este método es el que nos insertará eventos al hacer click en un día del calendario, en nuestro caso que nos muestre un modal para crear eventos y tome los valores de la fecha de inicio y fecha fin y nos la introduzca en el modal. También una función creada por nosotros en la que cada vez que hagamos click nos limpie los últimos datos escritos en el formulario para evitar enviar datos equivocados.
- **eventColor**, por defecto, los eventos se muestran de color azul, dado que nuestra aplicación tiene un patrón con tonos verdes, modificamos el color de los eventos por uno que se adapte a nuestra aplicación dándole armonía e integración.
- **eventClick()**, este método es el que nos mostrará la información del evento al hacer click sobre él mediante el modal.
- **calendar.render()**, este método es el más importante, es el encargado de renderizarnos el calendario, sin él no sería posible visualizarlo.

Una vez personalizado visualmente el calendario, es momento configurar el controlador y el modelo. Como mencioné antes, los eventos se almacenan en un array, por lo que necesitamos tratar los datos que nos retorna el servidor en un array en formato JSON.

```
//Mostrar eventos y lo almacenamos en un array en formato JSON
public function MostrarEventoC(){

    $tablaBD = "eventos";
    $datos = array();
    $result = EventosM::MostrarEventoM($tablaBD);
    foreach($result as $fila){
        $datos[] = array("id" => $fila["eventoID"],
                        "title" => $fila["titulo"],
                        "start" => $fila["finicio"],
                        "end" => $fila["ffin"],
                        "descripcion" => $fila["descripcion"],
                        "autor" => $fila["autor"]);
    }
    echo json_encode($datos);
}
//Crear Evento
```

```

101 require_once '../Models/Eventos.php';
102 //switch para llamar a las funciones desde AJAX
103 if(isset($_GET['accion'])){
104     switch($_GET['accion']){
105         case "listar":
106             $datos = new EventosC();
107             $datos -> MostrarEventosC();
108             break;
109         case "insertar":
110             $datos = new EventosC();
111             $datos -> crearEventoC();
112             break;
113         case "editar":
114             $datos = new EventosC();
115             $datos -> editarEventoC();
116             break;
117         case "eliminar":
118             $datos = new EventosC();
119             $datos -> eliminarEventoC();
120             break;
121     }
122 }
123 }

```

Para optimizar el código en un solo fichero controlador he optado por la creación de un `switch` case para cada tipo de acción dentro de controlador “eventosC.php”:

En el que para cada case creo un objeto de la clase EventosC y le asigno el método pertinente para cada uno.

Haremos uso de AJAX para tratar los datos recibidos por el servidor de forma asíncrona. Para ello crearemos funciones AJAX en el fichero `js` una por cada acción.

La función `agregarDatosForm()` va a ser la encargada de tomar el valor de los datos introducidos y los almacena en la variable “registro”.

```

//Función para agregar datos del formulario al registro
function agregarDatosForm(){
    let registro = {
        eventoID: $('#eventoID').val(),
        titulo: $('#titulo').val(),
        inicio: $('#inicio').val(),
        fin: $('#fin').val(),
        descripcion: $('#descripcion').val(),
        autor: $('#autor').val()
    }
    return registro;
}

```

El modal va a disponer de tres botones, uno para añadir cuando se ejecute el método `dateClick()`, es decir al pulsar en un día sin eventos. Y los otros dos botones, Añadir y Eliminar, visibles al hacer click sobre el evento(`eventClick()`).

Al hacer `click` sobre cada botón se ejecutará la función `agregarDatosForm()` introduciendo su valor en otra variable llamada “registro”, seguidamente se ejecutará la función AJAX correspondiente pasándole por parámetro la variable mencionada antes.

Las funciones AJAX tomarán como url la acción del case correspondiente y tomará como datos los datos almacenados

en la variable “registro” de la función `agregarDatosForm()`

una vez realizada con éxito la función AJAX se ejecutará el

método `refetchEvents()`, la cual refrescará el calendario mostrando los eventos actualizados de forma automática sin tener que refrescar el navegador.

[imágenes de las vistas en anexos]

```

//FUNCIONES
//funciones ajax
function agregarEvento(registro) {
    $.ajax({
        type: "POST",
        url: 'Controllers/eventosC.php?accion=insertar',
        data: registro,
        success: function(msg){
            calendar.refetchEvents();
        },
        error: function(error){
            alert("Error al agregar un evento: " + error);
        }
    });
}

function editarEvento(registro) {
    $.ajax({
        type: "POST",
        url: 'Controllers/eventosC.php?accion=editar',
        data: registro,
        success: function(msg){
            calendar.refetchEvents();
        },
        error: function(error){
            alert("Error al editar el evento: " + error);
        }
    });
}

function eliminarEvento(registro) {
    $.ajax({
        type: "POST",
        url: 'Controllers/eventosC.php?accion=eliminar',
        data: registro,
        success: function(msg){
            calendar.refetchEvents();
        },
        error: function(error){
            alert("Error al eliminar el evento: " + error);
        }
    });
}

```

```

//Eventos de botón
//Al pulsar "añadir"
$('#botonAgregar').click(function(){
    let registro = agregarDatosForm();
    agregarEvento(registro);
});
//Al pulsar "editar"
$('#botonEditar').click(function(){
    let registro = agregarDatosForm();
    editarEvento(registro);
});
//Al pulsar "eliminar"
$('#botonEliminar').click(function(){
    let registro = agregarDatosForm();
    eliminarEvento(registro);
});

```



### 6.8.7. Expediente Alumno

El módulo Expediente es donde el usuario Alumno va a poder visualizar la calificación recibida por el profesor del curso.

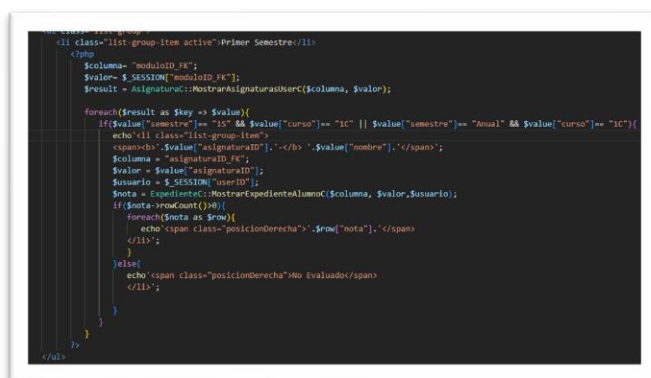
Archivos implicados en el módulo:

- **Modelo:** asignaturasM.php, expedienteM.php.
  - ✓ **Métodos:** MostrarAsignaturasUserM(), MostrarExpedienteAlumnoM().
- **Controlador:** asignaturasC.php, expedienteC.php.
  - ✓ **Métodos:** MostrarAsignaturasUserC(), MostrarExpedienteAlumnoC().
- **Vista:** expedienteAlumno.php.

Para este módulo se ha realizado un listado de las materias por semestres mediante el método *MostrarAsignaturasUserC()* pasándole por parámetro el valor de la variable de sesión equivalente al ID del estudio realizado por el usuario, de esta manera me mostrará todas las asignaturas del curso que está realizando el usuario.

Por otro lado, hemos creado la tabla notas, donde se almacenará un id, el ID del alumno como Foreign Key (FK), el ID de la asignatura como FK y la nota, de esta manera cada registro equivale a la nota de un alumno en una asignatura.

Para que nos muestre el resultado de las notas necesitamos pasarle al método *MostrarExpedienteAlumnoC()* los parámetros que equivalen al ID de la asignatura y el ID del alumno.



```

<li class="list-group-item active">Primer Semestre</li>
</li>
$columna = "moduloID_FK";
$valor = $_SESSION["moduloID_FK"];
$result = asignaturasM::MostrarAsignaturasUserC($columna, $valor);

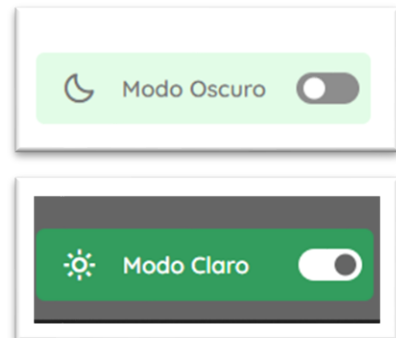
foreach($result as $key => $value){
    if($value["semestre"] == "1" && $value["curso"] == "1C" || $value["semestre"] == "2" && $value["curso"] == "1C"){
        echo<li class="list-group-item">
            <span><b> $value["asignaturaID"]</b></span> <b> $value["nombre"]</b></span>";
            $columna = "asignaturaID_FK";
            $valor = $value["asignaturaID"];
            $usuario = $_SESSION["userID"];
            $nota = ExpedienteC::MostrarExpedienteAlumnoC($columna, $valor, $usuario);
            if($nota->rowCount() > 0){
                foreach($nota as $row){
                    echo<span class="posicionDerecha"> $row["nota"]</span>";
                }
            }else{
                echo<span class="posicionDerecha"> No Evaluado</span>";
            }
        }
    }
}
</li>

```

[imágenes de las vistas en anexos]

### 6.8.8. Modo Claro/Oscuro:

Una funcionalidad que me interesaba implementar era el “modo claro” y el “modo oscuro” de la aplicación, ya que, personalmente soy partidaria del “modo oscuro”, porque al pasar mucho tiempo frente a la pantalla el “modo oscuro” ayuda a no sobrecargar la vista.



Mediante CSS cree el interruptor y mediante el manejador de eventos del DOM le di interactividad, donde al hacer *click* sobre el interruptor(*toggle*) aplico la clase “*dark*”, la cual convierte la paleta de colores creada en CSS a una con tonos negros.

Para preservar el modo seleccionado durante las cargas a otras vistas y evitar al usuario tener que seleccionar constantemente el modo deseado almacenamos en *localStorage* el modo seleccionado.



[imágenes de las vistas en anexos]



## 7. Despliegue y pruebas

Para el despliegue de la aplicación utilizamos el servidor web Apache y un sistema gestor de BBDD (SGBD) mediante el paquete de software XAMPP. Utilizaremos el puerto 80 para mostrar la aplicación en local y el puerto 3306 para el gestor de BBDD.

Una vez desplegada la aplicación en local en el navegador, en mi caso Brave, procedemos a la realización de pruebas de caja negra para verificar que la aplicación funciona correctamente y en caso de errores, localizarlos y corregirlos.

CU-01	Registro	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Sistema para introducir los datos personales de los usuarios	
<b>Precondición</b>	El usuario no está registrado	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	La aplicación pide ingresar DNI, Nombre, Apellidos, Correo, Contraseña, confirmación de contraseña, estudios a realizar y tipo de usuario.
	2	El usuario introduce los datos especificados en el formulario.
	3	Los datos introducidos son correctos, accede al login.
<b>Postcondición</b>	Usuario puede acceder a la aplicación desde el login	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	3	El DNI no dispone de 8 dígitos y una letra
	3	El correo no tiene una @ y finaliza con un "."+(2-3) letras
	3	La contraseña no tiene un tamaño de entre 5-10 caracteres y no contiene al menos una mayúscula, una minúscula, un dígito y un carácter especial.
	3	La contraseña y la confirmación de la contraseña no coinciden.

CU-01	<b>Inicio Sesión</b>	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Sistema de acceso a la aplicación	
<b>Precondición</b>	El usuario esta registrado	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El formulario pide correo y contraseña
	2	El usuario introduce los datos
	3	Los datos introducidos son correctos, se accede a la aplicación.
<b>Postcondición</b>	El usuario puede hacer uso de la aplicación	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	3	El correo o la contraseña no son correctos.

CU-01	<b>Editar Perfil</b>	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	El usuario modifica datos nombre, apellidos, correo y contraseña	
<b>Precondición</b>	El usuario está <i>logeado</i>	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	El formulario muestra los datos almacenados en la BD
	2	El usuario edita los datos.
	3	Se guardan los cambios.
<b>Postcondición</b>	Se muestran los datos actualizados en el formulario.	

## 8. Contexto laboral

Un escenario obvio para este proyecto es el sector educativo, instituciones educativas podrían utilizar Campus Virtual como herramienta para impartir clases en línea, como apoyo a las clases presenciales donde subir material extra de las clases, informar de exámenes, entregas de trabajos, entre otras cosas.

En este contexto, sería necesario un trabajo en equipo entre profesores, administradores del sistema y desarrolladores para implantar nuevas funcionalidades y mantener/mejorar las ya existentes.

Los profesores son los que se encargarían de darle a la aplicación la actividad necesaria para que los alumnos tengan la aplicación como herramienta diaria.

Los administradores son los que se encargarían del mantenimiento de datos, de la gestión de usuarios, estudios y asignaturas.

Los desarrolladores son los que mejorarán la aplicación, los que controlarán las versiones, los que se encargarán de gestionar errores y solventarlos.

A priori no precisa de un gran equipo de desarrolladores, pero al tratarse de un proyecto fácilmente escalable, estaría la posibilidad de precisar de un equipo más grande, acorde a lo que puede abarcar la aplicación.

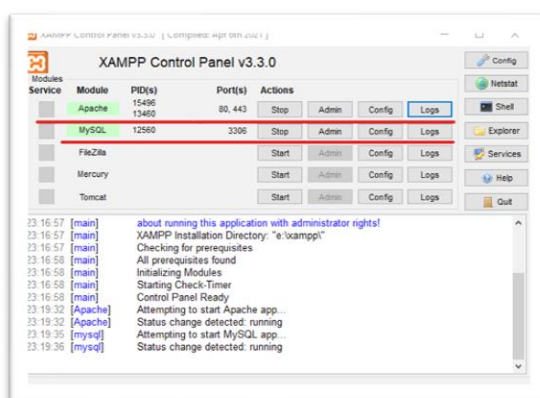
## 9. Instalación y configuración

Para hacer uso de la aplicación va a ser necesario un servidor, en este caso al tratarse de un proyecto sin dominio, vamos a desplegarlo en local. Para ello nos serviremos del paquete de software XAMPP, el cual ya dispone del servidor Apache y del Sistema Gestor de Base de Datos MySQL.

Lo primero que va a ser necesario es la instalación de XAMPP, en el siguiente enlace podrá instalarlo:

<https://www.apachefriends.org/download.html> (La versión con la que hemos trabajado es la 8.1.6).

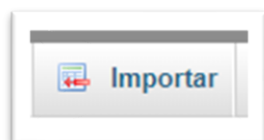
Una vez instalado y ejecutado se nos abrirá el panel de control de XAMPP donde podremos ver los distintos servicios que nos ofrece, en nuestro caso solo necesitaremos iniciar en primer lugar el servidor Apache y en segundo lugar MySQL.



Lo siguiente que necesitamos es importar la base de datos, para ello pulsaremos sobre “admin” de MySQL desde el panel de control o podemos acceder mediante el siguiente enlace:

<http://localhost/phpmyadmin/>

Se nos abrirá el panel para gestionar las bases de datos, en los botones de arriba tenemos la opción de “importar”, al clicar sobre ésta seleccionaremos el archivo “campus.sql” situado dentro de la carpeta del proyecto.



El siguiente paso será introducir la carpeta “ProyectoCampus”, que será la que contenga todo el código, dentro de la carpeta “htdocs” de Xampp.

Ahora ya, sin más dilación, procedemos al acceso a la aplicación, para ello tenemos dos vías, o bien desde el botón “explorer” buscamos la carpeta “htdocs” y localizamos la carpeta “ProyectoCampus”, o bien accedemos directamente desde el siguiente enlace:

<http://localhost/ProyectoCampus/login>

Tiene a su disposición tres tipos usuarios predeterminados registrados para probar la aplicación:

**Usuario administrador**

Correo: admin@admin.com

Contraseña: 12345

**Usuario profesor**

Correo: profe@profe.com

Contraseña: 12345

**Usuario administrador**

Correo: alumno@alumno.com

Contraseña: 12345

También puede registrarse como un nuevo usuario con los datos que desee.

## 10. PPosicionamiento SEO

Para el posicionamiento SEO de la aplicación vamos a tener en cuenta varios factores:

- Palabras clave: Las palabras clave podrán ir incrementando según la institución que utilizase la aplicación, depende del marketing que puedan ofrecer al usuario, pero así de forma genérica, teniendo en cuenta la finalidad del proyecto podríamos emplear las siguientes palabras clave:
  - ✓ “campus virtual”
  - ✓ “estudios en línea”
  - ✓ “formación online”
  - ✓ “formación a distancia”

En nuestro proyecto las mostraríamos de la siguiente manera:



```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="keywords" content="campus virtual, estudios en línea, formacion online, formacion a distancia">
  <!-- Bootstrap -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
  <!-- Boxicons -->

```

- Optimización de contenidos: proporcionándole al usuario un contenido de calidad, con las funciones de la aplicación bien definidas y con la información clara.
- Experiencia de usuario: proporcionándole al usuario una aplicación sencilla pero potente, muy intuitiva y con un diseño responsive.

## 11. Conclusiones

Como conclusión puedo decir que el resultado del proyecto ha sido satisfactorio, he logrado la gran mayoría de los objetivos propuestos al inicio. Además, aunque me he llevado muchos quebraderos de cabeza solucionando errores, puedo decir que he aprendido mucho, he mejorado mis habilidades con PHP y JS. Me he desenvuelto favorablemente en la solución de errores, localizando los problemas mediante la herramienta para desarrolladores del navegador o desde los logs de apache, buscando los tipos de errores y el por qué han surgido. Eso me ha permitido que conforme avanzaba en la aplicación, solucionaba mucho más rápido los problemas.

## 12. Vías futuras

Aunque estoy bastante contenta con el resultado de la aplicación, si es cierto que hay objetivos que me propuse que no he logrado alcanzar y otros que, conforme avanzaba en la aplicación y aumentaba mi experiencia, me hubiesen gustado implantarlos.

Estos son algunos ejemplos de mejoras que me gustaría implementar en un futuro:

- Hacer una **verificación de los usuarios mediante email**. El usuario al registrarse recibiría un correo electrónico en su bandeja de entrada con un enlace el cual verificaría al usuario modificando en el campo “estado” de la tabla usuarios, que por defecto es “1” por un “0”. Así de esa manera el administrador del sistema periódicamente podría hacer limpieza de usuarios cuyo estado es igual a “1”.
- **Generar pdf** de documentación como el expediente del alumno. Ahora mismo la aplicación sólo permite visualizar el expediente, pero para una aplicación en uso, es necesario que los usuarios puedan generar ese tipo de documentaciones importantes para emplearlas como certificados.
- **Tokenizar los registros**. Con el fin de aportar más seguridad a los usuarios, sería interesante una autenticación y una autorización de acceso mediante tokens.
- **Creación de cookies** si la carga se vuelve más pesada en el server. Aunque no son más seguras que la creación de variables de sesión, si es cierto que las cookies aportan una ventaja importante en el caso de que la aplicación tome más peso y la cantidad de usuarios que la utilicen sea cada vez mayor.
- **Mejorar el Calendario**. Si bien es cierto que ahora mismo es bastante funcional, pero no está al 100% de cómo me hubiese gustado. Me habría gustado que los usuarios pudieran ver quién es el usuario que ha creado el evento. Que los usuarios pudieran crear eventos “públicos” o “privados” eligiendo quien puede ver o no el evento.



## 13. Glosario

**Login:** De la palabra anglosajona “Log in” cuyo significado es “iniciar sesión”

**Feedback:** Palabra anglosajona que significa “retroalimentación”, “reacciones”, una opinión tras un hecho.

**MVC:** Siglas de Modelo-Vista-Controlador. Patrón de diseño de un proyecto.

**E-R:** Siglas de Entidad-Relación. Modelo de estructuración de una base de datos.

**SCRUM:** Metodología ágil.

**Sprint:** Etapas de la metodología ágil SCRUM.

**Primary Key (PK):** Clave Primaria. Identificador único e irrepitible de un registro en una tabla.

**Foreign Key (FK):** Clave foránea. Clave primaria de una tabla que se emplea en otra tabla.

**PDO :** Siglas de “**PHP Object Data**”. Programación de Objetos en PHP.

**BBDD:** Siglas de Bases de Datos.

**Framework:** Entorno de trabajo, conjunto estandarizado de conceptos.

**JS:** Siglas de JavaScript.

**AJAX:** Siglas de “*Asynchronous JavaScript And XML*”.

**CRUD:** Siglas de “*Create, Read, Update and Delete*”

**SQL:** Siglas de “*Structured Query Language*”

**SGBD:** Siglas de Sistema Gestor de Base de Datos.

**IDE:** Siglas “*Integrated Development Environment*”

## 14. Bibliografía/Webgrafía

### Diseño:

- Iconos: <https://boxicons.com/>
- Paleta de colores: <https://htmlcolorcodes.com/es/>
- Fuetes: <https://fonts.google.com/>
- Prototipos: <https://www.figma.com/>
- Diagramas: <https://app.diagrams.net/>
- Planificación y gestión de recursos: <https://trello.com/es>

### Documentación:

- Bootstrap5: <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- PHP: <https://www.php.net/docs.php>
- FullCalendar: <https://fullcalendar.io/docs/getting-started>
- W3Schools: <https://www.w3schools.com/>

### Video apoyo para Ajax + PHP:

- <https://www.youtube.com/watch?v=GeCNShiLdpc>

### Comprobación de expresiones regulares:

- <https://regexper.com/>

### Para solucionar el problema de MySQL:

- <https://www.youtube.com/watch?v=leL4XRTIUsw>
- <https://kinsta.com/knowledgebase/xampp-mysql-shutdown-unexpectedly/>

### Posicionamiento SEO:

- <https://es.godaddy.com/blog/que-es-posicionamiento-seo/>
- <https://miposicionamientoweb.es/guia-seo-para-principiantes/>

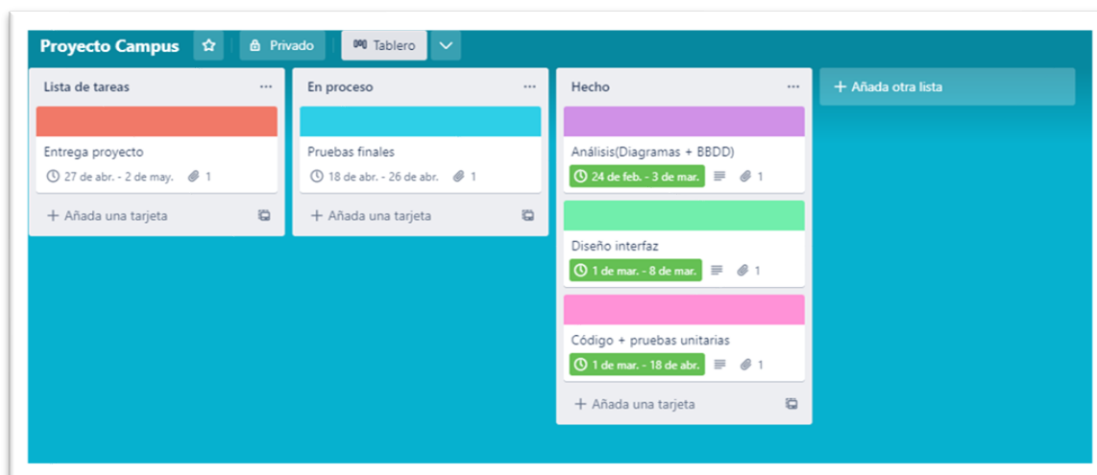
### Resolución de dudas varias:

- <https://stackoverflow.com/>

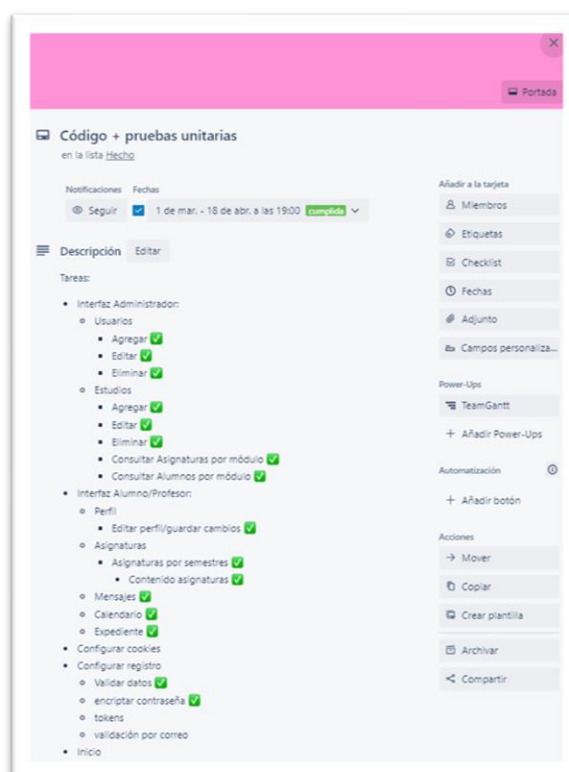
## 15. Anexos

### 15.1. Anexos I

#### 15.1.1. Anexos - Metodologías usadas



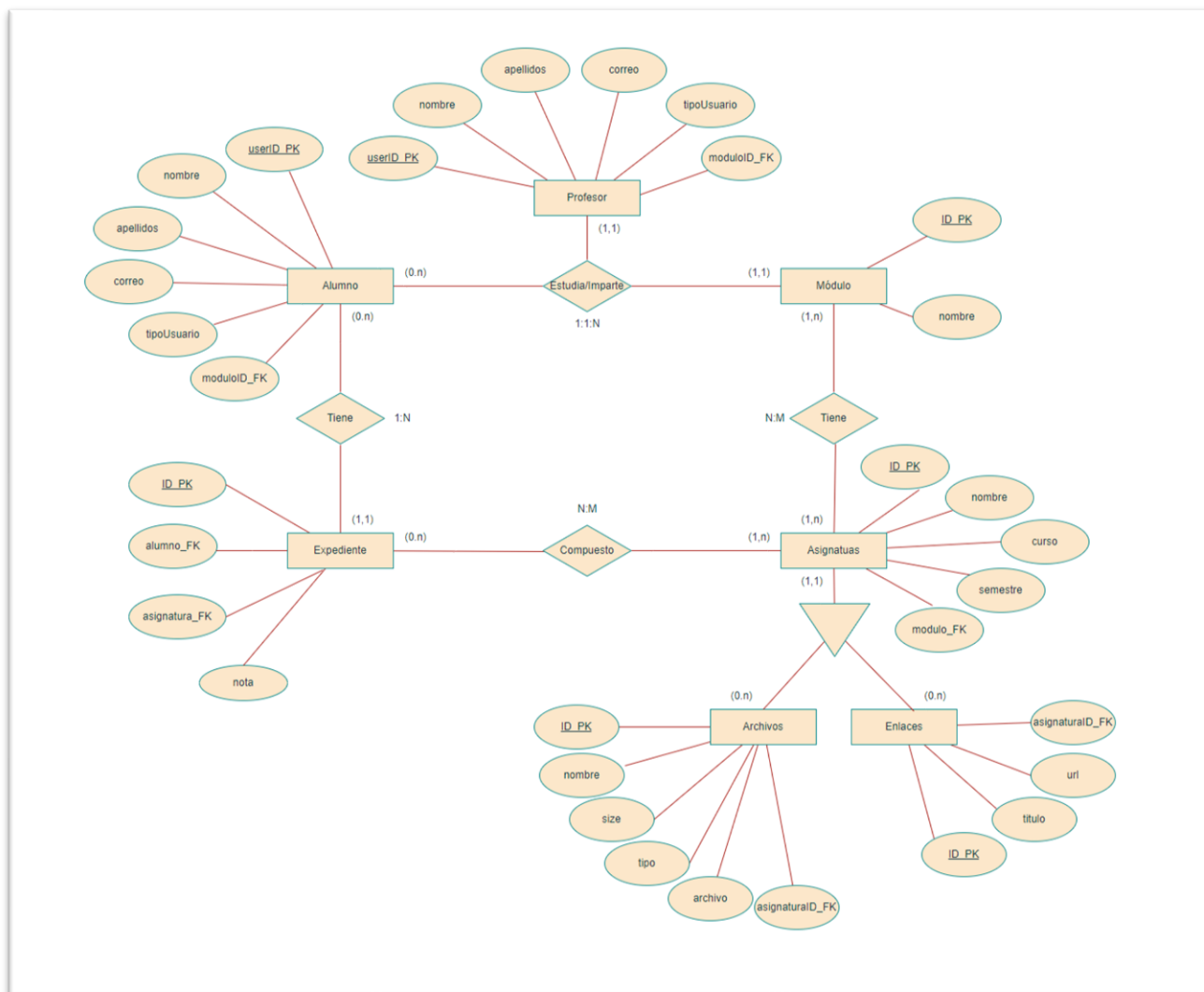
*Sistemas de tarjetas. Realizado en la aplicación Trello.*



*Ejemplo de lista de tareas dentro de una tarjeta. Realizado en la aplicación Trello.*

## 15.1.2. Anexos - Análisis del proyecto

### 15.1.2.1. Diagrama ER



*Diagrama Entidad-Relación. Realizado en la aplicación Diagrams.net*

### 15.1.2.2. Diagramas casos de uso

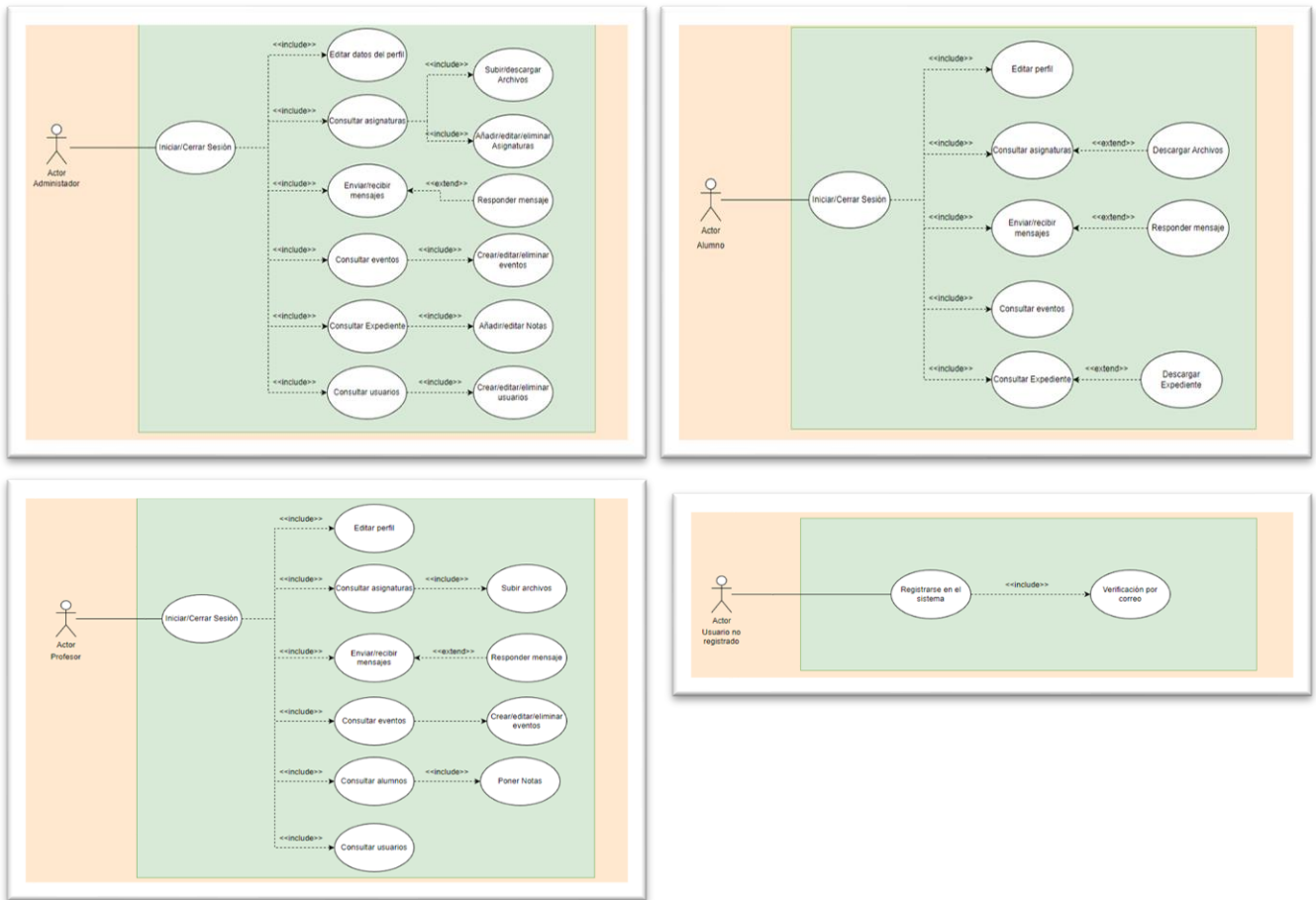


Diagrama Casos de uso. Diagrama realizado en la aplicación Diagrams.net

### 15.1.2.3. Diagrama de clases

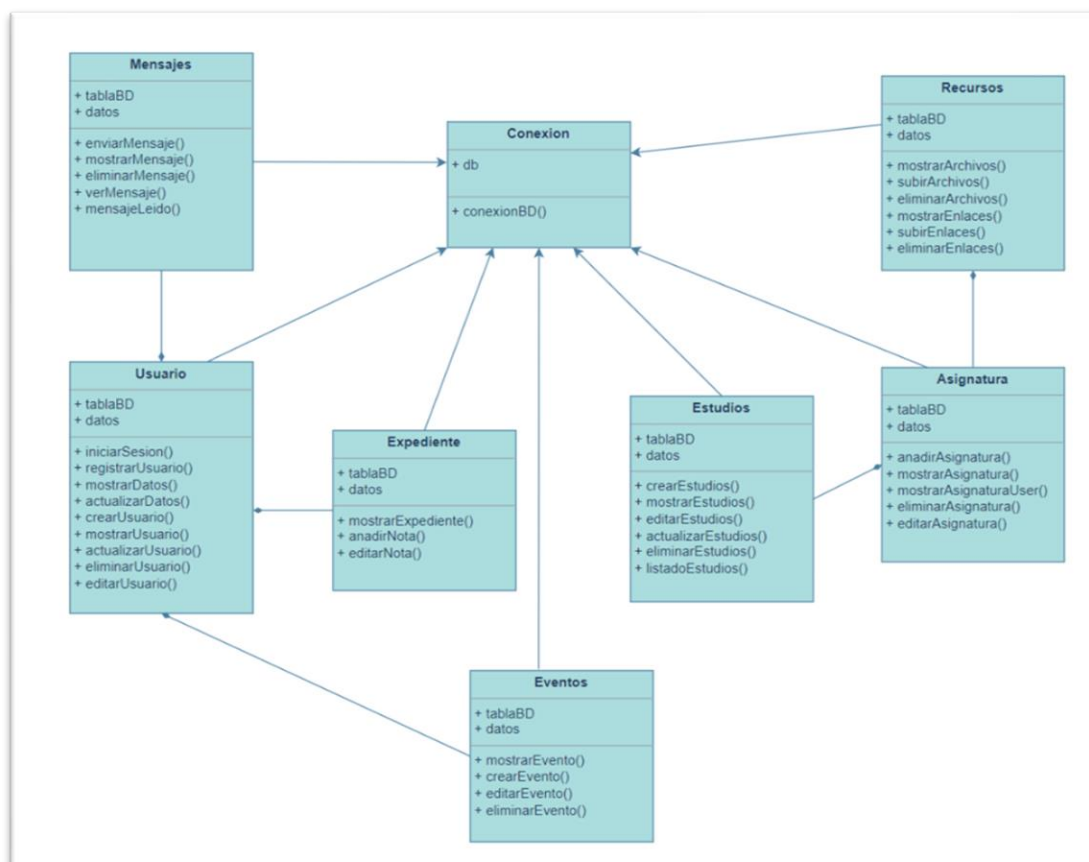
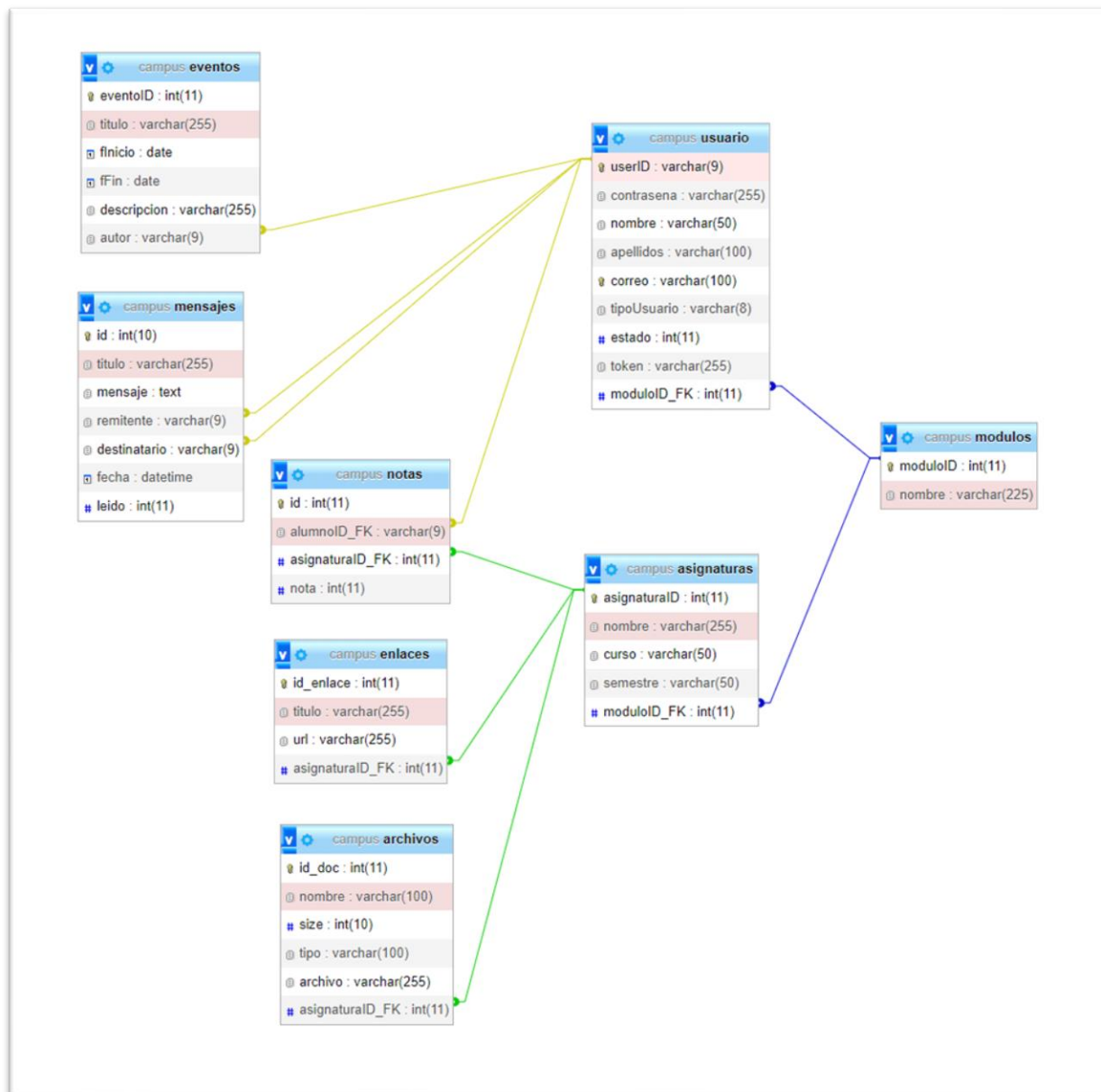


Diagrama de clases. Diagrama realizado en la aplicación *Diagrams.net*

### 15.1.3. Anexos - Diseño del proyecto

#### 15.1.3.1. Base de datos (modelo relacional)



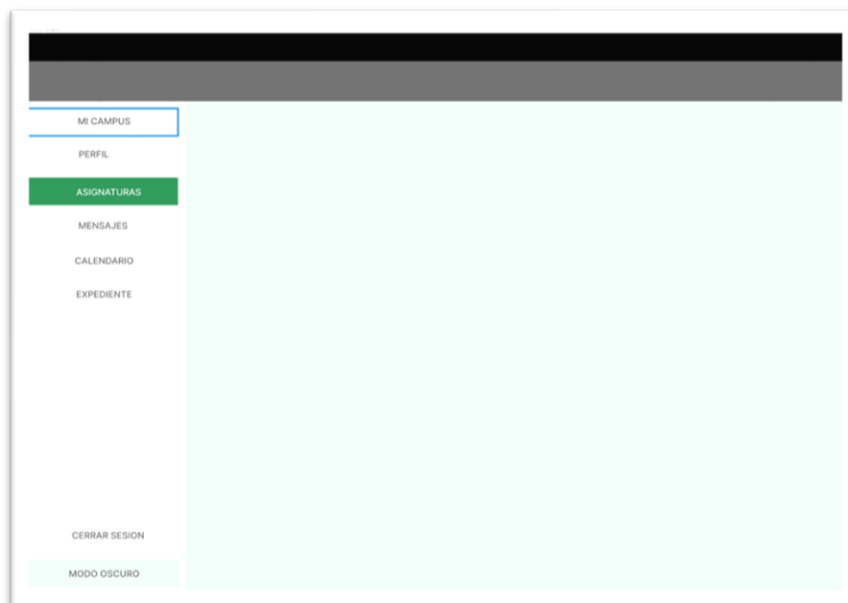
Modelo relacional de la base de datos. Captura extraída del modo diseñador de phpMyAdmin.



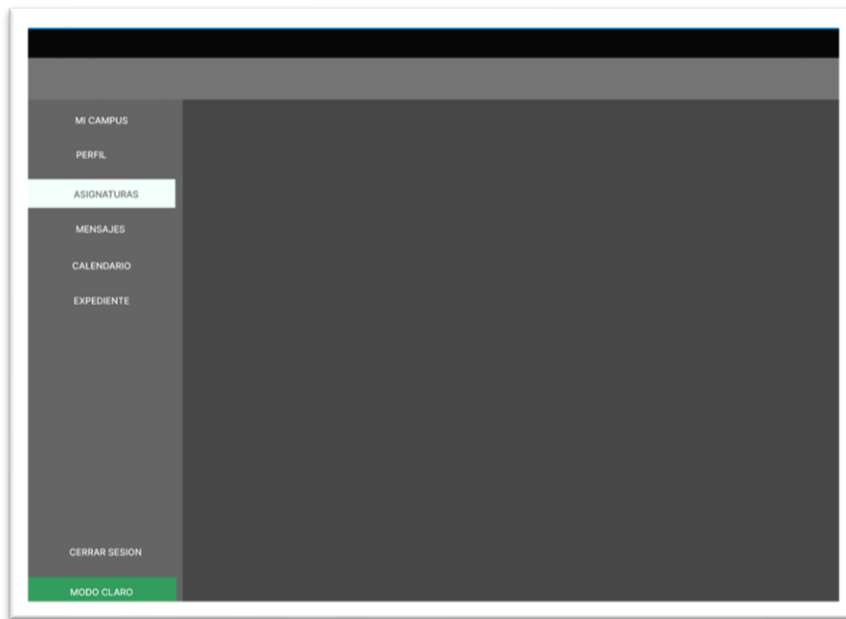
### 15.1.3.2. Prototipos y guía de estilo



*Prototipo interfaz para el registro/login. Realizado en la aplicación Figma.*



*Prototipo interfaz Campus "Modo Claro". Realizado en la aplicación Figma.*






Prototipo interfaz Campus "Modo Oscuro". Realizado en la aplicación Figma.



Guía de estilo. Realizado en la aplicación Figma.

### 15.1.3.3. Vistas de la aplicación

#### Gestor de Estudios:

Gestor de Estudios			
Ingresar Estudio			+
Código	Nombre	Acciones	
1	FPS Desarrollo Aplicaciones Web	 	Asignaturas Estudiantes
3	FPM Sistemas Microinformáticos y Redes	 	Asignaturas Estudiantes

*Gestor de estudios desde Administrador. Captura de la aplicación.*

Gestor de Estudios			
Código	Nombre	Acciones	
1	FPS Desarrollo Aplicaciones Web	 	Asignaturas Estudiantes
3	FPM Sistemas Microinformáticos y Redes	 	Asignaturas Estudiantes

*Gestor de estudios desde Profesor. Captura de la aplicación.*

Gestor de Asignaturas de: FPS Desarrollo Aplicaciones Web				
<a href="#">Añadir Asignatura</a>		<a href="#">← Volver</a>		
Código	Nombre	Curso	Semestre	Acciones
1013	Programación	1C	Anual	 
1014	Bases de Datos	1C	Anual	 
1015	Entornos de Desarrollo II	1C	2S	 
1021	Entorno Cliente	2C	2S	 
1022	Sistemas Informáticos	1C	1S	 
1024	Entorno Servidor	2C	1S	 

*Gestor de asignaturas desde Administrador. Captura de la aplicación.*

Gestor de Asignaturas de: FPS Desarrollo Aplicaciones Web

[← Volver](#)

Código	Nombre	Curso	Semestre
1013	Programación	1C	Anual
1014	Bases de Datos	1C	Anual
1015	Entornos de Desarrollo II	1C	2S
1021	Entorno Cliente	2C	2S
1022	Sistemas Informáticos	1C	1S
1024	Entorno Servidor	2C	1S

*Gestor de asignaturas desde Profesor. Captura de la aplicación.*

Estudiantes de: FPS Desarrollo Aplicaciones Web

[← Volver](#)

DNI	Alumno	Acciones
11111111W	Perez Molina, Joana	<a href="#">Expediente</a>
12121212f	meme, Meme	<a href="#">Expediente</a>

*Listado de estudiantes por estudio. Captura de la aplicación.*

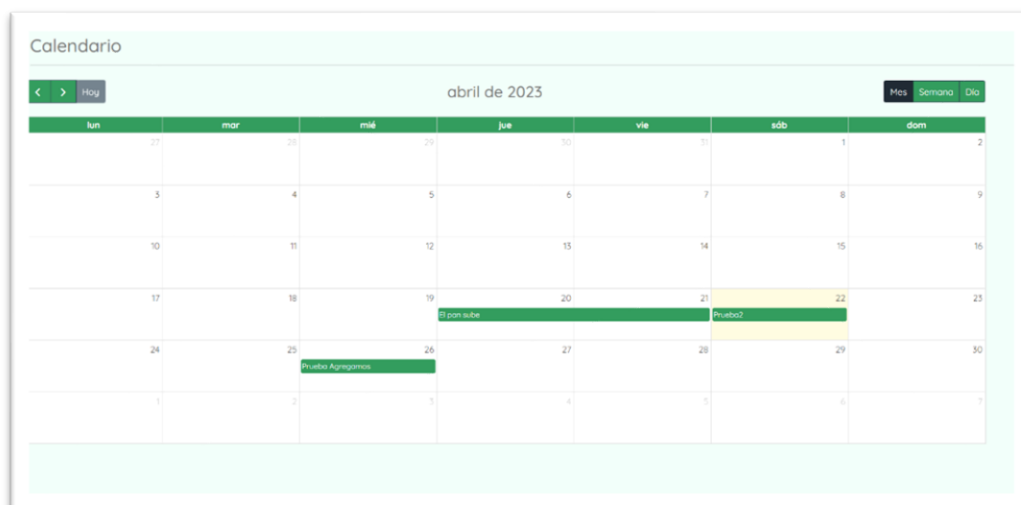
Expediente Académico de: Joana Perez Molina

[← Volver](#)

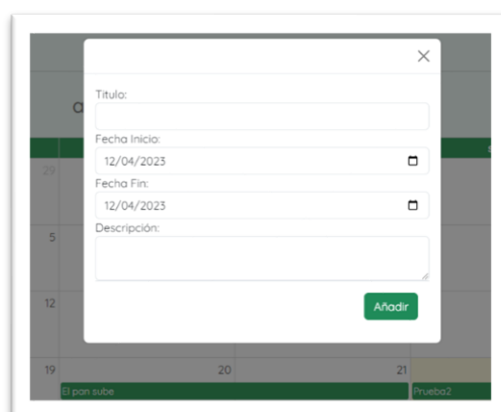
Código	Nombre	Curso	Semestre	Nota	Acción
1013	Programación	1C	Anual	5	<a href="#">📄</a>
1014	Bases de Datos	1C	Anual	8	<a href="#">📄</a>
1015	Entornos de Desarrollo II	1C	2S	7	<a href="#">📄</a>
1021	Entorno Cliente	2C	2S	5	<a href="#">📄</a>
1022	Sistemas Informáticos	1C	1S	7	<a href="#">📄</a>
1024	Entorno Servidor	2C	1S	No evaluado	<a href="#">+</a>

*Expediente académico de alumno. Captura de la aplicación.*

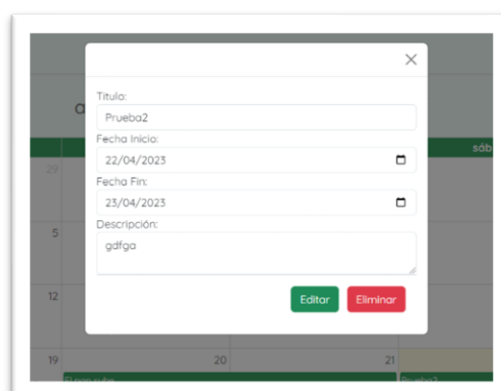
## Calendario:



*Vista general del calendario. Captura extraída de la aplicación.*



*Vista modal Añadir evento. Captura extraída de la aplicación.*













*Vista modal Editar/Eliminar evento. Captura extraída de la aplicación.*

## Expediente:











Expediente Académico de: Joana Perez Molina	
Primer Semestre	
1013 Programación	5
1014 Bases de Datos	8
1022 Sistemas Informáticos	7
Segundo Semestre	
1013 Programación	5
1014 Bases de Datos	8
1015 Entornos de Desarrollo II	7
Tercer Semestre	
1024 Entorno Servidor	No Evaluado
Cuarto Semestre	
1021 Entorno Cliente	5

*Vista Expediente de un alumno. Captura extraída de la aplicación.*

## Modo claro/oscuro:

Mi Campus		Gestor de Usuarios						
Inicio		Añadir Usuario						
Perfil		DNI Usuario	Nombre	Apellidos	Correo Electrónico	Estudios	Tipo de Usuario	Acciones
Usuarios		00000000	admin	admin	admin@admin.com	Usuario Administrador	Admin	 
Estudios		11111111	Joana	Perez Molina	joanaperez@hotmail.com	FPS Desarrollo Aplicaciones Web	Alumno	 
Asignaturas		12121212	Maria	maria	maria@maria.com	FPS Desarrollo Aplicaciones Web	Alumno	 
Mensajes		22222222	Pedro	Jimenez Lopez	pedrojimenez@hotmail.com	FPS Desarrollo Aplicaciones Web	Profesor	 
Calendario		33333333	Profesor	Profesor	profe@profe.com	FPM Sistemas Microinformáticos y Redes	Profesor	 
Cerrar Sesión								
Modo Oscuro		<input type="checkbox"/>						

*Vista "Modo Claro". Captura extraída de la aplicación.*

Mi Campus		Gestor de Usuarios						
Inicio		Añadir Usuario						
Perfil		DNI Usuario	Nombre	Apellidos	Correo Electrónico	Estudios	Tipo de Usuario	Acciones
Usuarios		00000000	admin	admin	admin@admin.com	Usuario Administrador	Admin	 
Estudios		11111111	Joana	Perez Molina	joanaperez@hotmail.com	FPS Desarrollo Aplicaciones Web	Alumno	 
Asignaturas		12121212	Maria	maria	maria@maria.com	FPS Desarrollo Aplicaciones Web	Alumno	 
Mensajes		22222222	Pedro	Jimenez Lopez	pedrojimenez@hotmail.com	FPS Desarrollo Aplicaciones Web	Profesor	 
Calendario		33333333	Profesor	Profesor	profe@profe.com	FPM Sistemas Microinformáticos y Redes	Profesor	 
Cerrar Sesión								
Modo Claro		<input checked="" type="checkbox"/>						

*Vista "Modo Oscuro". Captura extraída de la aplicación.*

#### 15.1.4. Anexos – Despliegue y pruebas

CU-01	Asignaturas (Administrador)	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Acceso a todas las asignaturas almacenados en la BD.	
<b>Precondición</b>	El usuario ha iniciado sesión	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Listado de todas las asignaturas.
	2	El usuario puede acceder a cada una de las asignaturas pinchando sobre ellas.
	3	Se accede al contenido de la asignatura, apuntes y recursos.
	4	El usuario puede subir apuntes y/o recursos

CU-01	Asignaturas (Profesor)	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Acceso solo a las asignaturas a las que pertenece los estudios impartidos.	
<b>Precondición</b>	El usuario ha iniciado sesión	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Listado de todas las asignaturas por semestres.
	2	El usuario puede acceder a cada una de las asignaturas pinchando sobre ellas.
	3	Se accede al contenido de la asignatura, apuntes y recursos.
	4	El usuario puede subir apuntes y/o recursos

CU-01	Asignaturas (Alumno)	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Acceso solo a las asignaturas a las que pertenece sus estudios.	
<b>Precondición</b>	El usuario ha iniciado sesión	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Listado de todas las asignaturas por semestres.
	2	El usuario puede acceder a cada una de las asignaturas pinchando sobre ellas.
	3	Se accede al contenido de la asignatura, apuntes y recursos.

CU-01	<b>Enviar Mensajes</b>	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Sistema de mensajería interna.	
<b>Precondición</b>	El usuario ha iniciado sesión.	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Visualiza listado de bandeja de entrada.
	2	Pulsa sobre “Nuevo mensaje”
	3	Rellena el formulario seleccionando el destinatario, inserta un título y redacta el mensaje.
	4	Pulsa sobre enviar.
<b>Postcondición</b>	El destinatario recibirá en su bandeja de entrada el mensaje.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	3	El usuario no ha rellenado los campos título y mensaje.

CU-01	<b>Crear Evento</b>	
<b>Versión</b>	1.0 (25/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Creación de un evento en el calendario	
<b>Precondición</b>	El usuario debe iniciar sesión como “admin” o “profesor”	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al calendario.
	2	Clickar sobre un día.
	3	Introducir Título y Descripción.
	4	Pulsar “Enviar”
<b>Postcondición</b>	El resto de los usuarios pueden ver el evento creado	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>



CU-01	Añadir Estudios	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Agregar un estudio a la base de datos	
<b>Precondición</b>	El usuario debe iniciar sesión como “admin”.	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Estudios”
	2	El usuario ingresa el nombre completo que quiere almacenar en la base de datos en el Input donde especifica “Ingresar Estudio”
	3	Pulsa sobre el botón “+”.
<b>Postcondición</b>	El estudio ingresado aparece en la tabla. Ahora se puede editar/eliminar el estudio.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>

CU-01	Añadir Asignaturas	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Agregar una asignatura a un Estudio	
<b>Precondición</b>	El usuario debe iniciar sesión como “admin”.	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Estudios”.
	2	Pulsa sobre el botón “Asignaturas”.
	3	Pulsa sobre el botón “Añadir Asignatura”.
	4	Ingresa en el formulario el nombre de la asignatura, selecciona el curso y el semestre.
	5	Envía el formulario.
<b>Postcondición</b>	La asignatura ingresada aparece en la tabla. Ahora se puede editar/eliminar la asignatura.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>

CU-01	Añadir Archivo	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Subir un archivo pdf a una asignatura	
<b>Precondición</b>	El usuario ha iniciado sesión como “admin” o como “profesor”.	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Asignaturas” del menú.
	2	Pulsar sobre una asignatura.
	3	Sobre “Apuntes” pulsa el botón para subir archivos.
	4	Introduce el nombre personalizado del archivo y selecciona el archivo desde el ordenador.
	5	Pulsa sobre “Añadir Archivo”.
<b>Postcondición</b>	Los usuarios que pertenecen al Estudio podrán ver/descargar en archivo.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>

CU-01	Añadir Enlace	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Subir un enlace a una asignatura	
<b>Precondición</b>	El usuario ha iniciado sesión como “admin” o como “profesor”.	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Asignaturas” del menú.
	2	Pulsar sobre una asignatura.
	3	Sobre “Recursos” pulsa el botón para subir enlaces.
	4	Introduce en el formulario el nombre personalizado del Enlace y la url del mismo.
	5	Pulsa sobre “Subir Enlace”.
<b>Postcondición</b>	Los usuarios que pertenecen al Estudio podrán pulsar sobre el enlace y será redirigido mediante otra pestaña.	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>

CU-01	Añadir nota	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Calificar a los alumnos	
<b>Precondición</b>	El usuario debe iniciar sesión como “admin” o “profesor”	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Estudios”.
	2	Acceder a “Estudiantes”
	3	Acceder a “Expediente”
	4	Pulsar sobre el botón de “Acción”
	6	Selecciona la nota.
	7	Pulsa sobre el botón de envío.
<b>Postcondición</b>	El alumno ya tiene una calificación	
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>
	5	Si el alumno ya está calificado, mostrará la nota y un botón para editar la nota.
	5	Si el alumno no está calificado, mostrará como “No evaluado” y un botón para añadir la nota.

CU-01	Ver expediente Alumno	
<b>Versión</b>	1.0 (23/04/2023)	
<b>Autor/es</b>	Rosa López Mas	
<b>Descripción</b>	Medio por el que el usuario “alumno” puede ver sus calificaciones.	
<b>Precondición</b>	El usuario ha iniciado sesión como “alumno”	
<b>Secuencia normal</b>	<b>Paso</b>	<b>Acción</b>
	1	Acceder al módulo “Expediente”.
	2	El usuario visualiza un listado de las asignaturas y su calificación.
<b>Postcondición</b>		
<b>Excepciones</b>	<b>Paso</b>	<b>Acción</b>

## 15.2 Anexo II

