

# **Proyecto Final de Grado**

## **Desarrollo de un CMS para la gestión de un blog de videojuegos**

*Ignacio de Loyola Romero Romero*

## Tabla de contenido

Análisis de necesidades del sector .....	3
¿Porque necesito una página web? .....	3
¿Qué puedo hacer a diferencia de otras páginas web de videojuegos? .....	3
Propuesta de diseño de un proyecto concreto que de respuesta a dichas necesidades .....	3
¿Cuáles son los objetivos del proyecto? .....	3
¿Qué es un CMS? .....	3
¿Cuáles son las herramientas software que voy a utilizar para el desarrollo del CMS? .....	4
Desarrollo del proyecto .....	10
Conectar con la base de datos con Medoo .....	10
Enlazando los archivos principales .....	11
Log-in de usuario hecho con Ajax .....	17
Creación de un formulario para subir publicaciones. ....	20
Formulario de Categorías .....	22
Subir las publicaciones .....	25
Mostrar las publicaciones .....	28
Mostrar las publicaciones individuales .....	29
Formulario de registro del usuario .....	31
Formulario de log in del Usuario .....	31
Registrar Usuarios .....	31
Log in de los usuarios .....	34
Cambio de menú al iniciar sesión como usuario .....	34
Marcar favorito la publicación .....	34
Mostrar todas las publicaciones favoritas .....	36
Vista para el administrador .....	39
Activar y desactivar las publicaciones en el Home .....	41
Valoración económica .....	43
Conclusión .....	43
Dificultades encontradas .....	43
Opciones de mejora .....	43

## Análisis de necesidades del sector

A continuación, hablaré de mi proyecto fin de grado, que será una página web de un blog de videojuegos. Mi objetivo será hacer una página de información hecha por mí, un periódico de noticias, análisis y cosas sobre el mundo de los videojuegos.

### ¿Porque necesito una página web?

Necesito una página web para que mi medio sea el más reconocido y que pueda haber gente que decida ir a mi blog antes que al de otros que son periódicos como por ejemplo son Vandal, Meristation, etc. Habrá noticias más recientes y un análisis más detallado de mis videojuegos favoritos.

### ¿Qué puedo hacer a diferencia de otras páginas web de videojuegos?

Al ser un periódico de videojuegos nuevo, hay mucha competencia. No podré hablar de los juegos más recientes por presupuesto, pero sí puedo hablar de juegos anteriores. Este no será un periódico como otro cualquiera pues mi intención es hablar de los juegos que me apetezcan y juegos de culto. No quiero hablar por ejemplo del último FIFA, quiero hablar por ejemplo sobre la saga de Pro Evolution Soccer.

## Propuesta de diseño de un proyecto concreto que de respuesta a dichas necesidades

### ¿Cuáles son los objetivos del proyecto?

Los objetivos se dividen en dos:

#### **Objetivos principales:**

- Un CMS (Content Management System) funcional programado en PHP, Ajax(jQuery) y como gestor de base de datos MySQL .

#### **Objetivos opcionales:**

- Subir Vídeos
- Que los usuarios marquen como favoritos sus publicaciones.

### ¿Qué es un CMS?

El término CMS proviene del inglés Content Management System, que significa Sistema de Gestión de Contenidos. Es un sistema online que nos permite poner en marcha un sitio web de forma práctica y rápida.

<https://rockcontent.com/es/blog/cms/>

¿Cuáles son las herramientas software que voy a utilizar para el desarrollo del CMS?

Los programas con los que voy a trabajar son:

- PhpStorm
- Putty.

¿Por qué voy a trabajar con estos programas?

PhpStorm es un entorno de desarrollo para PHP el cual ha sido utilizado en la formación impartida por el Instituto Superior De Escuela Profesional (Isep Ceu). Es un programa con el que me siento cómodo y que es perfecto para mi proyecto.

Putty me va a ser útil para conectar directamente con el servidor alojado en Amazon Web Services.

Aún sin nada hecho, ¿qué idea tengo pensada para hacer en mí página web?

Mis puntos más importantes son:

- Hacer un CMS funcional
- Subir Publicaciones
- Registrar Usuarios

### *Desarrollo del proyecto*

#### Servidor AWS y construcción de la base de datos

Lo primero que tuve que hacer fue hacerme una cuenta en Amazon Web Services y seguir todos los pasos indicados por el tutor siguiendo las indicaciones de este enlace:

<https://jairogarciarincon.com/clase/curso-instalacion-y-puesta-en-marcha-de-un-entorno-aws>

Siguiendo los pasos indicados logré instalar Ubuntu en una máquina alojada en AWS. Por último, configuramos el Putty para conectarme al servidor utilizando las claves creadas por el propio Amazon Web Services.

Una vez configurado el acceso instalamos PHP y MySQL siguiendo los pasos indicados en los enlaces que nos ha dejado el tutor:

<https://jairogarciarincon.com/clase/curso-instalacion-y-puesta-en-marcha-de-un-entorno-aws/instalacion-de-apache-2-mysql-5-y-php-7>

<https://jairogarciarincon.com/clase/bases-de-datos-en-php/introduccion-a-las-bases-de-datos>

Siguiendo los pasos del enlace mencionado anteriormente, vamos ahora a crear una base de datos para mi página web y lo haré desde el Putty.

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql> CREATE DATABASE myweb CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Ahora ya creada la base de datos, vamos a crear un usuario para acceder a la base de datos.

```
mysql> CREATE USER 'usuario-myweb'@'localhost' IDENTIFIED BY 'Atletico10.';  
Query OK, 0 rows affected (0.00 sec)  
mysql>
```

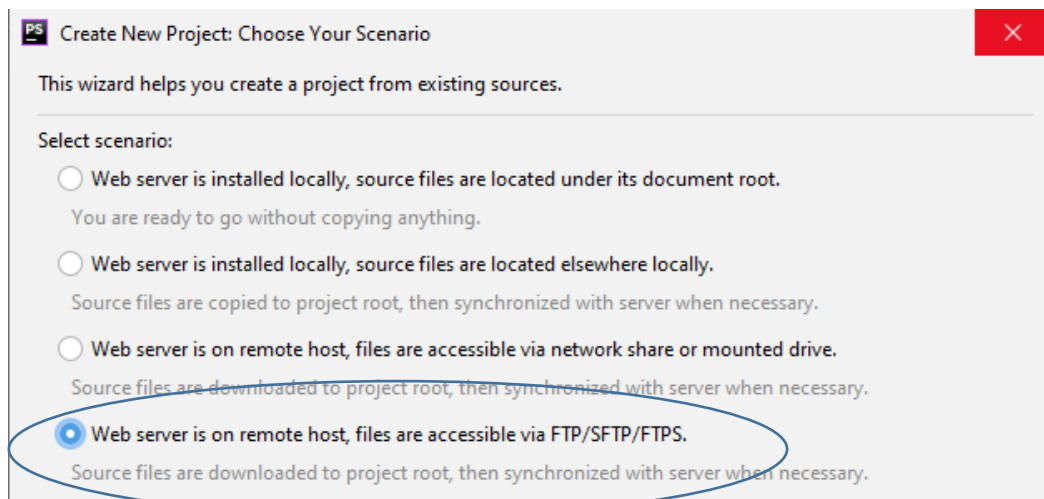
Ya creado el usuario, voy a darle permisos para el usuario pueda hacer las funciones básicas, es decir, insertar un registro, actualizar, borrar un registro, consultas, crear tablas, modificar tablas, borrar tablas, etc.

```
mysql> GRANT INSERT,SELECT,UPDATE,DELETE ON myweb.* TO 'usuario-myweb'@'localhos  
t';  
Query OK, 0 rows affected (0.00 sec)
```


Por último, cree la tabla de usuarios utilizando **use myweb** para lanzar consultas sobre la base de datos 'myweb' creada para ser la base de datos que utilizará el CMS.

Conectar PHPSTORM con el servidor y con nuestra base de datos para el desarrollo:

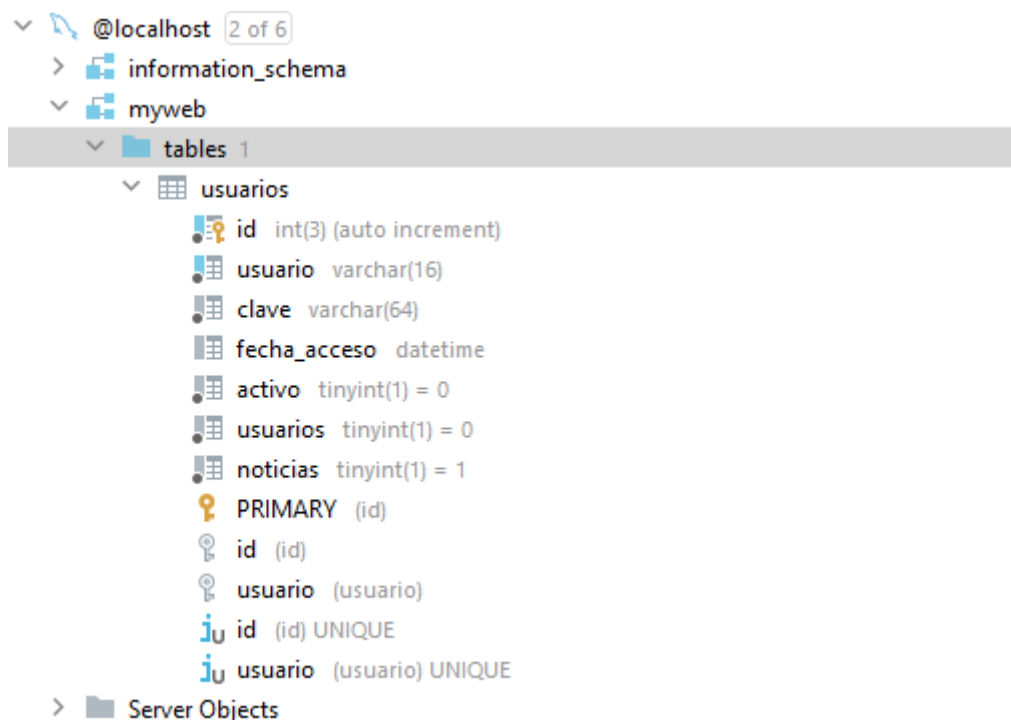
Abrimos el PhpStorm y vamos a la pestaña Files-New Project from Existing Files y nos saldrá una ventana que nos parecerá un par de opciones.



Le damos a siguiente y nos aparecerá los diferentes servidores remoto, en nuestro caso en la ip pública 18.134.74.30.

 `servidortfg` `sftp://18.134.74.30:990` `http://18.134.74.30`

Para conectar con la base de datos 'myweb' creada anteriormente tengo que darle a la pestaña plus y donde pone Schema, nos saldrá una ventana donde tengo que poner mi nombre de la base datos que se llama myweb. A continuación se muestra como aparece en el PHPSTORM esta conexión con la base de datos.



Se puede apreciar que habíamos creado previamente la tabla de usuarios con todos los campos y propiedades como primary keys etc.

Además de la tabla usuarios se han creado las siguientes tablas necesarias para el CMS:

- Publicaciones: Se almacena los datos necesarios para las publicaciones.
- Favoritos: Se registran las publicaciones favoritas de los distintos usuarios.
- Categorías: Se registra las categorías que se pueden utilizar para asociarlas a una publicación.
- Usuario: Usuarios que pueden marcar como favoritas publicaciones y acceder a su listado personalizado de publicaciones favoritas.

#### Publicaciones

```
create table publicaciones
(
    nombre          varchar(50)          not null,
    texto           text                  null,
    img_publicaciones varchar(100)        not null,
    id_publicaciones int auto_increment,
    creado          varchar(100)          not null,
    id              int(3)                 not null,
    id_categorias   int                   not null,
    extension       varchar(5)            not null,
    activo          tinyint(1) default 1 not null,
    constraint publicaciones_id_publicaciones_uindex
        unique (id_publicaciones)
);
```

Creo una constraint en id\_publicaciones para que ID\_publicaciones sea único, ya que ID es para almacenar el id de usuario que redacta la publicación.

#### Favoritos

```
create table favoritos
(
    id_favorito int auto_increment
        primary key,
    usuario varchar(16) not null,
    publicaciones int not null,
    constraint favoritos_usuario_uindex
        unique (usuario)
);
```

### Categorías

```
create table categorias
(
    id_categorias int auto_increment,
    categorias varchar(50) null,
    constraint categorias_id_categorias_uindex
        unique (id_categorias)
);
alter table categorias
    add primary key (id_categorias);
```

### Usuario



```

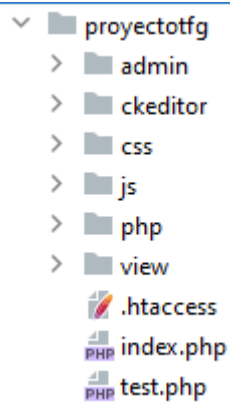
create table usuario
(
    id        int(3) auto_increment,
    usuario   varchar(16) not null,
    clave     varchar(255) not null,
    constraint id
        unique (id),
    constraint usuario
        unique (usuario)
);

alter table usuario
    add primary key (id);

```

### Estructura del proyecto

Voy a hacer una estructura basada en las cosas que necesito y son:



- CSS: Contiene los archivos CSS que utilizaré para mi CMS, así como los CSS del framework Semantic UI y archivos de CSS personalizados.
- Js: Contiene librerías de javascript como jQuery y javascripts personalizados que se utilizan para realizar peticiones ajax contra el backend en PHP.
- Admin: Contiene archivos PHP para el backend de las funciones que puede realizar un administrador.
- Php: Contiene archivos de PHP para las funciones de conectar la base de datos y consultas.

- View: Son archivos PHP que contienen HTML para mostrar las diferentes vistas del CMS como, los menús de admin y del home, inicios de sesión, publicaciones, etc.
- .htaccess: para hacer mis propias URL amigables.

### Librería PHP para realizar operaciones SQL contra la base de datos

Para conectar la base de datos, voy a utilizar Medoo, que es una librería que permite realizar operaciones SQL mediante funciones de PHP.

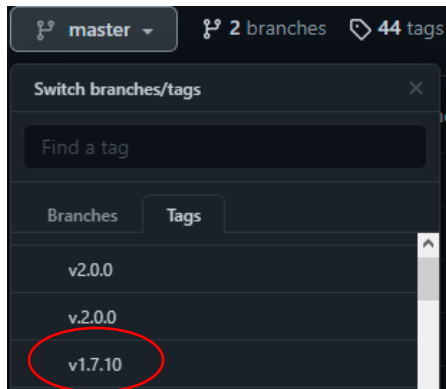
#### Instalación

Me descargue el archivo zip de su página web.

<https://medoo.in>

Al descargar la versión más reciente, no me funcionaba algunas funciones y tuve que descargarme su penúltima versión en github que es la 1.7.10 en vez 2.0.1

<https://github.com/catfan/Medoo>



Al descargar el archivo medoo.php, lo puse en mi carpeta de php.

### Desarrollo del proyecto

#### Conectar con la base de datos con Medoo

Creamos un archivo llamado init.php para configurar la conexión el cual contiene la clave y contraseña para conectar con MySQL.

```

<?php
session_start();
require 'Medoo.php';

use Medoo\Medoo;

try{
    $database = new Medoo([
        // [required]
        'database_type' => 'mysql',
        'server' => 'localhost',
        'database_name' => 'myweb',
        'username' => 'usuario-myweb',
        'password' => 'Atletico10.',
    ]);}catch(PDOException $e){
    echo "No se pudo conectar a la base de datos";
}
}

```

En el archivo init.php, cabe recordar que utilizo un require de la librería. Al arrancarlo con google Chrome, me salía una pantalla en blanco, al ver que no aparece ningún mensaje de error, quiere decir que conecta perfectamente con la base de datos. También decir que he creado un sesión\_start() para que inicie sesión en el CMS.

## Enlazando los archivos principales

En la carpeta php, creamos nuestro archivo **functions.php**. Dentro del archivo de **functions.php**, hago un require para llamar a la base de datos con el archivo **init.php**

```
require 'init.php';
```

Ahora creo las clases, que las voy a llamar, AccionesUsuario y AdminAcciones. Las clases de AccionesUsuario la creo para llamar a las funciones que haga un usuario normal, sin tener que ser administrador. Estas acciones se verán más adelante.

Por otro lado, tenemos la clase AdminAcciones es para las funciones del administrador del CMS.

Estas clases realizan operaciones SQL contra la base de datos para obtener publicaciones, actualizar una publicación, borrar una publicación, etc.

El archivo que gestiona cual es la información que hay que cargar y la vista que hay que utilizar en base a la ruta que se ponga en el navegador es el **index.php**. En el index.php se realiza primero un require de app\_top.php. En este archivo el cual carga el **functions.php**

```
require 'functions.php';
$usuario = new AccionesUsuarios();
```

cargamos los datos necesarios utilizando las clases AccionesUsuario o AdminAcciones. Y por último hacemos un require de la vista.

### Construcción de la maquetación básica del CMS

Para la maquetación del CMS, en vez de utilizar bootstrap, voy a usar Semantic UI porque quiero un diseño sencillo y algo rápido. Al ver que en **index.php** dentro de la carpeta proyectotfg al ponerle el enlace no me salía el enlace, tuve que descargarme un archivo zip

<https://semantic-ui.com/introduction/getting-started.html>

Comprí el archivo y lo puse en la carpeta css, luego en la de framework y descomprí el archivo zip, la ruta es proyectotfg\css\framework\semantic.

Me sitúo en el archivo de index.php y llamo a los estilos del semantic.

```
<linkrel="stylesheet"href=http://18.134.74.30/proyectotfg/css/framework/semantic/semantic.css>
```

Llamo desde el servidor de Amazon para que así me carguen todos los estilos en mi CMS.

Después llamo al archivo script del framework de Semantic UI

```
<script  
src="http://18.134.74.30/proyectotfg/css/framework/semantic/semantic.js"></script>.
```

Para que funcione, necesito una librería jQuery, voy a este enlace

<https://developers.google.com/speed/libraries#jquery>

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
```

Ahora en el archivo **index.php**, hago una llamada al menú del home para ello, hago una etiqueta php y pongo la etiqueta require

```
<?php  
require 'view/nav/main_nav.php';  
?>
```

Creó el archivo en la ruta que está puesta en la imagen y para el menú, me he ayudado del layout de ejemplo que viene en la documentación de Semantic UI.

<https://semantic-ui.com/examples/homepage.html>

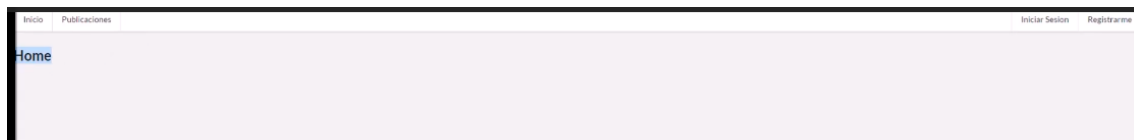


Obviamente, no tiene ningún enlace ya que todavía no he relacionado ningún archivo. Ahora me voy a **main.css** y le voy a poner al body un color blanco con

```
body { background-color: #FFFFFF};
```

Vuelvo al **index.php**, y hago un isset para comprobar si en la ruta me viene la sección home cargo la vista **home.php**.

```
if(!isset($_GET['section'])) {require 'view/home.php';}
```



Y vemos que nos sale en el index al iniciar.

Ahora voy a editar un poco el archivo home con php, voy a abrir un div con una clase para el framework

```
<div class="ui grid stackable container">
```

Quiero meterle 16 columnas a mi home para ello, abro otro div y lo llamo de la siguiente forma.

```
<div class="sixteen wide column">
```

He de comentar que Semantic UI es muy cómodo a la hora de montar un diseño sencillo, simplemente pones lo que he puesto arriba y tienes 16 columnas. Este framework lo descubrí gracias a este enlace que explicaba las ventajas y desventajas de Bootstrap y la ventajas y desventajas de Semantic.

<https://developerz.software/es/2020/03/29/semantic-ui-o-bootstrap-descubre-los-pros-y-contras-de-cada-framework/>

Luego modificando un poco el home poniendo

```
<h2>Name</h2>
<p>19-05-2021</p>
```

Me salía tal como voy a mostrar.



Vemos que mi interfaz le falta cosas, para ello voy a ir al **main.css** para retocarlo un poco. Para ello quiero poner que me muestre el título en el centro y las publicaciones, más de lo mismo para que me quede como quiero, he creado una clase para el body que se llama **body\_background** y lo he creado con un div. He utilizado las siguientes características para el **body\_background** en el **main.css**.

```
.body_background {
  width: 100%;
  height: 50vh;
  background-position: center;
}
```

El vh lo utilice ( que es altura de la ventana gráfica en height) para que me coja solo el 50 por ciento, en el caso de que quiera luego meterle una cabecera de imagen. Y el **background-position** lo utilizo para que se centre todo. Vi que el título no se me centra. Para ello tuve que crear una clase, llamada **main\_title** para que se me centrará

Quiero centrar tanto el título de mi artículo (no está creado aun en la base de datos), para ello creo dos clases que se llamarán **post\_name** para el nombre y **post\_data** para la fecha.

Voy a crear una etiqueta **img**, que lo pondré justo encima de del nombre, no pondremos ninguna imagen todavía, pero pondremos una de prueba gracias a este enlace que es para probar

<https://placeholder.com>

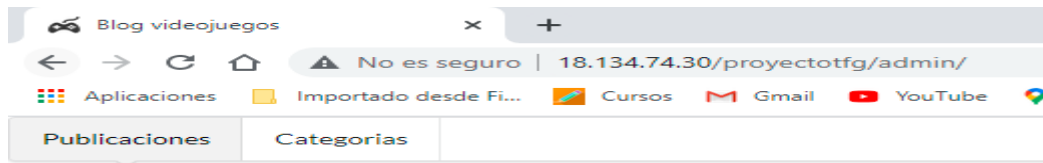
Le añado una clase a la imagen, llamándole **post\_img** y quedaría tal que así:



#### Vista para el administrador

Ahora me voy al archivo **index.php** y copio lo que lo todo el contenido de ese archivo y me lo llevo a otro **index.php** que sería el index de la carpeta admin. Cambio lo que es la ruta, es decir, le pongo **../** a todas las rutas, Modifico lo que es **isset** para cuando no este iniciado sesión, no pueda ir al archivo de **home.php**

```
If(
!isset($_SESSION['admin']))
{require '../view/admin/home.php';}
```



## Sesión Iniciada

Ahora haré lo mismo para el menú, es decir, que sea diferente. Hago lo mismo en el **index.php** y me creo un menú nuevo para el admin, me creo en el nav un archivo que se llamará **main\_admin\_nav.php**.

```
if(!isset($_SESSION['admin']))
{ require '../view/nav/main_admin_nav.php';}
```

El menu de **main\_admin\_nav.php**

```
<div class="ui pointing menu" style="...">
  <?php /*<a href="http://18.134.74.30/proyectotfg/view/admin/posts.php" class="active item">*/ ?>
  <a href="posts" class="active item">
    Publicaciones
  </a>
  <?php /*<a href="http://18.134.74.30/proyectotfg/view/admin/categories.php" class="active item"> */?>
  <a href="categories" class="item">
    Categorías
  </a>
  <div class="right menu">
    <a href="http://18.134.74.30/proyectotfg/php/Accionesusuario/logout.php" class="item">
      Cerrar Sesión
    </a>
  </div>
</div>
```

Me creo un archivo que lo llamo **login.php** y lo meto en la carpeta view.

```
<div class="login_register_container">
  <h2>Iniciar Sesión</h2>
  <p><b>Correo Electrónico/Usuario</b></p>
  <div class="ui input">
    <input type="text" class="txtEmailLogin" placeholder="CorreoElectrónico/Usuario">
  </div>
  <p><b>Contraseña</b></p>
  <div class="ui input">
    <input type="password" class="txtPassLogin" placeholder="Introduce tu contraseña">
  </div>
  <div class="ui basic blue button btnAdminLogin">Iniciar Sesión</div>
</div>
```

Le creo un div para el diseño y lo llamo **login\_register\_container** y el diseño es así como lo tengo.



## Iniciar Sesión

**Correo Electrónico/Usuario**

**Contraseña**

Log-in de usuario hecho con Ajax

Para esta parte, me he tenido que ayudar de dos videos para hacer logear mi usuario admin

<https://www.youtube.com/watch?v=-tkQzhEXozI&list=PLdWUlk20IaEW6IjbvKFJoigFpXkSc9Owv&index=9>

<https://www.youtube.com/watch?v=s7s2Ruj9NxQ&list=PLdWUlk20IaEW6IjbvKFJoigFpXkSc9Owv&index=10>

Siguiendo los pasos del primer enlace puesto, me creo un archivo de javascript en mi carpeta de javascript, y lo llamo **admin.js**.

```
$(document).ready(function (){
```

Luego me creo en el botón de iniciar sesión de **login.php** una clase CSS que en el vídeo la llama btnAdminLogon

```
<div class="ui basic blue button btnAdminLogon">Iniciar Sesión</div>
```

Pongo la clase abro un .on y le pongo un click para cuando le doy un click que me carga la función

```
$(".btnAdminLogon").on( types: "click", selector: function (){
```

Me creo dos variables que se llamarán usuario y clave, estas recogen los datos y para ello lo relaciono con sus respectivas clases CSS que cree en el **login.php** en el input type y le pongo val para que establezca el valor del campo input y trim para los espacios en blanco.

```
var usuario = $(".txtEmailLogin").val().trim(),
    clave = $(".txtPassLogin").val().trim();
```

```
$.ajax({ url: {
  type: "POST",
  url: root + "php/accionesAdmin/login.php",
  data: {
    usuario: usuario,
    clave: clave
  },

```

Abrimos un AJAX y le digo que lo haga a través del método POST le doy una url que he creado arriba con la ruta de mi directorio que se llama root con el enlace principal de mi CMS. Y pongo la ruta de la petición y le pongo la ruta del archivo **login.php** y le doy sus datos.

```
    success : function (data) {
      console.log(data);
      if (data == "true") {
        window.location.href = "http://18.134.74.30/proyectotfg/admin/";
      } else if (data == "false") {
        alert("Las credenciales no coinciden, por favor intentelo de nuevo.");
      }
    }
  });
});
```

Más adelante, abro un success con un function si le digo que si es cierto el usuario y la contraseña, que me lleve a la ventana administrador y si no me salta el mensaje de error. Y no me salta ya que todavía no he llamado a las tablas desde **functions.php**.

Ahora me voy al archivo **functions.php** y me creó una función para que me inicie sesión y que recoja los datos y me la creo en la clase de AdminAcciones .

```

class AdminAcciones
{
    public function login($user, $pass)
    {
        global $database;

        $data = $database->select( table: "usuarios", [
            "clave"
        ], [
            "OR" => [
                "usuario" => $user
            ]
        ]);
        $password_db = $data[0]["clave"];

        if ($pass == $password_db) {
            return true;
        } else {
            return false;
        }
    }
}

```

Llamo a la base de datos con un global, consulto los usuarios con un select utilizando Medoo. Para la password abro un if y si es cierto pues inicia sesión y si no pues nos salta el mensaje que puse en el **admin.js**.

En el menú de admin que lo llamo **main\_admin\_nav.php**, le pongo los href con los enlaces y funciona. Ahora un archivo en la carpeta de AccionesUsuario y le creo un archivo que le llamo un **logout.php**

```

<?php
session_start();

session_destroy();
header( header: "Location: http://18.134.74.30/proyectotfg/");
exit;
?>

```

Añado en el **index.php** de admin un isset para que muestre el **main.php**, es decir, en el caso de que el usuario inicie sesión, le saldrá la ventana de que su sesión está iniciada.

```

elseif (isset($_SESSION['admin']) && !isset($_GET['section']))
{ require './view/admin/main.php';}

```

Compruebo que funciona la ventana error

18.134.74.30 dice

Las credenciales no coinciden, por favor intentelo de nuevo.

Aceptar

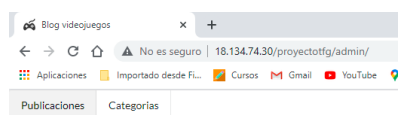
Correo Electrónico/Usuario

admi

Contraseña

.....

Iniciar Sesión



Sesión Iniciada

### Creación de un formulario para subir publicaciones.

Me creo un archivo **.htaccess**, ya que al poner las rutas, no me reconocían los estilos, ni los menús etc. Al hacer el archivo **.htaccess** no me funcionaba, así que tuve que meterme en el Putty y abrir y meterme en los permisos. Esto lo hice gracias a este enlace que encontré como hacerlo.

[https://www.digitalocean.com/community/tutorials/how-to-rewrite-urls-with-mod\\_rewrite-for-apache-on-ubuntu-16-04](https://www.digitalocean.com/community/tutorials/how-to-rewrite-urls-with-mod_rewrite-for-apache-on-ubuntu-16-04)

Al hacer eso, hice como había que hacerlo y ya solucioné eso.

```
#Rutas Administrador|
RewriteRule ^admin/posts$ admin/index.php?section=posts [QSA]
```

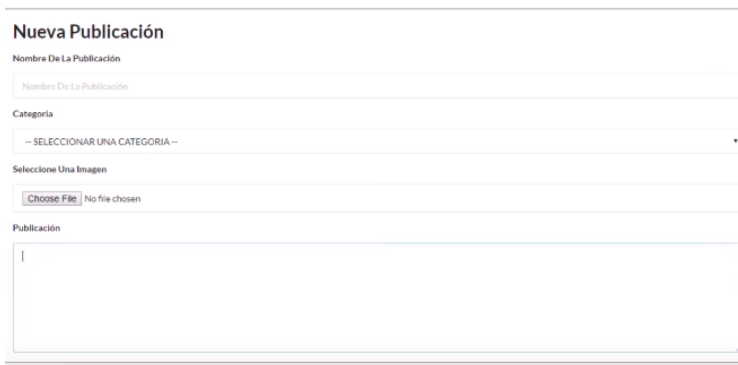
Le pongo URL amigable en el **.htaccess** para que me lleve a la categoría posts que serían las publicaciones de mi CMS.

Luego voy al **index.php** de admin y le pongo en el elseif le pongo un get con un section para que llame a posts que serían publicaciones y le pongo que requiera el archivo de **posts.php** dentro de la carpeta de admin.

```
elseif (isset($_SESSION['admin']) && isset($_GET['section']) && $_GET['section'] ==
"posts"){ require '../view/admin/posts.php';}
```

Creo el archivo **posts.php** dentro de la carpeta de view, y en el archivo me creo un div con una clase para luego retocarlo en el **main.css**. Me creo un form , es decir, un formulario, para que me recoja los campos de las tablas, en el formulario pongo los siguientes campos

- El nombre de la publicación
- Su categoría con el aspecto del select importado de Semantic UI
- La imagen del artículo
- El contenido de la publicación



Formulario de Nueva Publicación:

- Nombre De La Publicación:
- Categoría:
- Seleccione Una Imagen:  No file chosen
- Publicación:

Le doy gracias a un plugin llamado ckeditor que hace que tengas un editor de texto en el textarea y he utilizado la versión 4 de ckeditor y he cogido el standard pack

<https://ckeditor.com/ckeditor-4/download/>

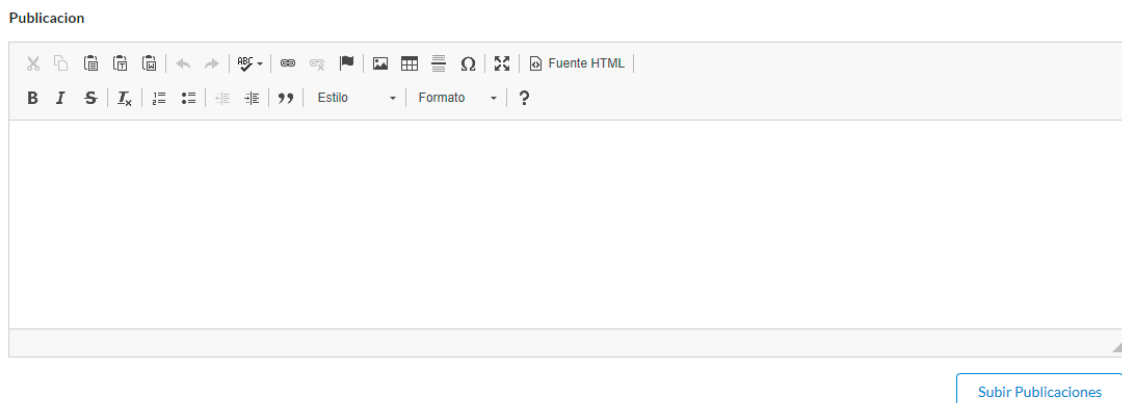
Para integrarlo lo hare a través de javascript.

```
try{ CKEDITOR.replace('txtDescription');}catch (e){}
```

Lo pongo después del var root para que recoja todos los campos.

Cabe recordar que descargué el zip del enlace mostrado con anterioridad.

Ya que no me lo cogía externamente desde el enlace. Pongo una id en el textarea y me lo reconoce.



Publicacion

CKEditor 4.7.3

Subir Publicaciones

## Formulario de Categorías

Voy a hacer un formulario de categorías para las diferentes plataformas de videojuegos. Para ello hago el mismo proceso que hice anteriormente, es decir, creo el archivo en la carpeta view/admin llamándole **categories.php** y lo mismo que tuve que hacer para el **index.php** de las vistas del posts, hago lo mismo para las categorías llamándole **categories**.

En el archivo, creó un formulario al igual que el anterior archivo.

```
<div id="new_post_container" class="ui form new_post_container">
  <h1>Categorías</h1>
  <p><b>Nombre De La Categoría</b></p>
  <div class="ui input">
    <input type="text" class="txtNameCategory" placeholder="Nombre De La Categoría">
  </div>
  <button class="ui blue basic button btnSaveCategory">Guardar Categoría</button>
  <p class="clearfix"></p>

  <h3>Categorías Existentes</h3>
  <table class="ui single line table tblCategories">
    <thead>
      <th>Nombre De Categoría</th>
      <th>Acción</th>
    </thead>
    <tbody>
      <?php foreach($categorias as $category): ?>
        <tr>
          <td><?php echo $category['categorias']; ?></td>
          <td><i class="fa fa-trash" btnRemoveCategory" aria-hidden="true" title="Eliminar Categoría" data-categoriaId="<?php echo $category['id_categorias']; ?>"></i> </td>
        </tr>
      <?php endforeach; ?>
    </tbody>
  </table>
</div>
```

En el archivo hago un foreach para que muestre el nombre de las categorías llamando al campo de las categorías y luego en la acción pongo una papelera, por si quiere borrarla. Para que pueda eliminar y añadir categorías, voy a **functions.php** y creo las funciones

```

public function getCategories(){
    global $database;

    $categories = $database->select( table: "categorias",
        "id_categorias",
        "categorias"
    );
    return $categories;
}

public function saveCategory($category)
{
    global $database;

    $database->insert( table: "categorias", [
        "categorias" => htmlentities($category)
    ]);
    return $database->id();
}

public function deleteCategory($category_id){
    global $database;

    $delete = $database->delete( table: "categorias", [
        "id_categorias" => $category_id
    ]);
    return $delete->rowCount();
}

```

Para que todo esto me funcione, tengo que hacer las funciones en el **admin.js** y crearme los botones de eliminar. En el javascript llamo al botón `btnSaveCategory` que he creado en el archivo **categories.php**. Al igual que con el botón de iniciar sesión como administrador, es una función que hay que darle click, creo una variable que llamaré `category` para que recoja los datos de las bases de datos en las funciones creadas en php. Hago un Ajax para que llame a la función, una URL con variable `root` que ya creamos antes y con la ruta que es `php/accionesAdmin/save_category.php` y que recoja los datos con un `data`.

```

$.ajax( url: {
    type: "POST",
    url: root + "php/accionesAdmin/save_category.php",
    data: {
        category: category
    },

```

Luego más abajo del código, lo que hago es el botón de guardar categoría, hay una clase llamada `loading`, que lo hace es que haga una animación para que cargue y nos salga un mensaje que nos dirá que lo tenemos guardado

```

        console.log(data);

        $('tblCategories tr:last').after('<tr><td>' + category + '</td></td><td><i class="fa fa-times btnRemoveCategory" data-categoriaId="' + data + '" title="Eliminar Categoria"
    }else{
        alert("Hubo un error.");
    }
},
error: function () {
    alert("Se ha producido un error");
}
});
});

```

Ahora me creo el archivo de **save\_category.php**. Ahí llamo a la función de saveCategory en **functions.php**.

```

<?php
require '../functions.php';
$obj = new AdminAcciones();

$save = $obj->saveCategory($_POST['category']);
echo $save;
?>

```

Para eliminar la categoría, hago la misma estructura que para guardar, con algunos cambios.

```

$('tblCategories').on( types: "click", selector: '.btnRemoveCategory', data: function () {
    var category_id = $(this).attr( name: "data-categoriaId"),
    self = this;

```

Aquí lo que hago es llamar a la función a través de un click con una clase que me he creado en un table y luego creo una variable para con el attr obtenga el valor de un atributo para el primer elemento en el conjunto de elementos coincidentes que lo quiero coger el id de la categoría que tengo en el i y luego le creo un self = this; donde quiero que me recoja todo más adelante

```

$.ajax( url: {
    type: "POST",
    url: root + "php/accionesAdmin/delete_category.php",
    data: {
        category_id: category_id
    },

```

Aquí llamo y hago lo mismo que hice en el anterior AJAX, pero aquí el archivo que tengo que crear es **delete\_category.php** para que llame a las funciones de PHP de borrar categorías.



```

        success: function (data) {
            if(data > 0) {
                $(self).parent().parent().remove();
                alert("Se ha eliminado correctamente.");
            }else{
                alert("Hubo un error.");
            }
        },
        error: function (){
            alert("Se ha producido un error");
        }
    });
});

```

Aquí lo que hago es que me recoja el tr y el td con parent y luego un remove para que me lo elimine.

Luego en el **app\_top\_admin.php** recojo los datos de las categorías usando la función de obtener categorías, que es lo mismo que hice en el **index.php** de usuario.

### Categorías

Nombre De La Categoría

Guardar Categoría

Categorías Existentes

Nombre De Categoría	Acción
PSS	
Xbox Series x	

## Subir las publicaciones

Para subir las publicaciones, lo primero que tuve que hacer en el **app\_top\_admin.php**, es que cuando el administrador, es decir, seleccione la vista, seleccione la vista posts y también que quiero obtener las categorías, luego más adelante, voy al **.htaccess** para redireccionar la ruta a donde quiero, es que la ventana para subir publicaciones. Luego en el archivo **posts.php**, hago un foreach para que me reconozca las categorías en el select.y luego debajo poner un option y abrir una etiqueta php para llamar a las al id de las categorías y las categorías de la base de datos.

A la hora de escribir una publicación en la caja de texto, no nos lo reconocía, tuve que meterme en la documentación de ckeditor para saber cómo.

[https://ckeditor.com/docs/ckeditor4/latest/guide/dev\\_savedata.html](https://ckeditor.com/docs/ckeditor4/latest/guide/dev_savedata.html)

Aquí buscando la página, decía que hay que llamar a la caja de texto en el script que he creado llamado **admin.js**. Tuve que crearme una clase en el botón de subir publicación para que reconociera la función y lo hice al igual que todas las funciones que se ejecutan cuando se realiza un click en un botón de acción.

```
$('.btnSavePost').on( types: "click", selector: function(e)
{ e.preventDefault();
  var description = CKEDITOR.instances.txtDescription.getData(),
    name = $('.txtNamePost').val().trim(),
    category_id = $('.txtCategoryPost').val().trim();
```

Le pongo un preventDefault porque quiero evitar la acción por defecto. Me creo tres variables que son:

- Description : Llamo al id de textarea y luego con un getData, recojo todo los datos.
- Name : Llamo a la clase que cree en el input para que reconozca el nombre de la publicación
- Category\_id : Llamo a la clase que cree en el select para que me conozca las categorías en el select.

```
if(description !== "" && name !== "" && category_id > 0){
  var formData = new FormData($("#new_post_container")[0]);
  formData.append( name: "description", description);
  console.log(formData);
```

Le digo que si la descripción, el nombre y el id de la categoría está vacía, que no publique nada. Luego con la variable, llamo al id que hice en el form para que coja todos los datos.

```
$.ajax( url: {
  xhr: function (){
    var xhr = new window.XMLHttpRequest();
```

Abro una etiqueta Ajax con un xhr que es una petición al http. Creo una variable para que recoja con XMLHttpRequest la información sin tener que recargar la página.

```
xhr.upload.addEventListener( type: "progress", listener: function (evt : ProgressEvent<XMLHttpRequestEventTarget> ){
    if(evt.lengthComputable){
        var percentComplete = evt.loaded / evt.total;
        percentComplete = parseInt( s: percentComplete * 100);

        console.log(percentComplete);
    }
}, options: false);

return xhr;
},
```

Con la variable de xhr, le pongo un upload con addEventListener que hace que registre un evento a un objeto en específico, pongo que esa de tipo progress y que sea una función con evt que es un evento en este caso lo que quiero es que cuando se suba una publicación se muestre un porcentaje( que no aparece en la web) en la consola, y para ello, me creo una variable con un lengthComputable que hace que devuelva la duración del progreso.

```
type: "POST",
url: root + "php/accionesAdmin/new_post.php",
data: formData,
processData: false,
contentType: false,
```

Aquí hago la recogida de datos con un post, la ruta del archivo que tengo que crear más la página web, los datos con la variable formData, se recogen, y el processData y el contentType como falsos.

Ahora me voy a crear el archivo de **new\_post.php**, donde recojo la función como hice en borrar la categoría o guardar la misma y hago lo mismo que hice con guardar y borrar categorías, es decir, llamar a la función,

```
<?php
require '../functions.php';
$obj = new AdminAcciones();

$save = $obj->savePost($_POST['category_id']);
|
echo $save;
?>
```

Luego en **functions.php**, creo la función de savePost y lo que quiero es insertar el nombre, la categoría, la imagen, el id de la publicación y cuando se ha creado y le pongo un time() en creado.

Volví al javascript, para que me recogiera las imágenes

```

    success: function(data){
        $('#txtNamePost,.image_file').val( value: "");
        CKEDITOR.instances['txtDescription'].setData("");
        alert("Se subio la publicacion");
    },
    error: function(){
        alert("Error...");
    }
}
```

Meto el nombre de clase de input en el nombre y en la clase que creada para subir la imagen. Vi que la imagen se me subía bien, ya que no me detectaba la imagen y aparecía sin el formato de imagen en el putty

```
-rw-r--r-- 1 www-data www-data 189198 May 27 19:54 60aff8da8c165
-rw-r--r-- 1 www-data www-data 10710 May 30 17:42 60b3ce6c6068f
```

Para que me subiera con formato de imagen, me busque un enlace que me ayudó a solucionar el problema.

<https://www.saotn.org/validate-mime-types-with-php-fileinfo/>

Me creé un campo en la base de datos llamado extensión, de varchar de 100, y en **new\_post.php**, lo ponía y también ponía todo lo relacionado a la imagen y ahora sí me salía con el formato en el putty.

```
-rw-r--r-- 1 www-data www-data 188468 May 30 17:56 60b3dlb87e91c.jpeg
-rw-r--r-- 1 www-data www-data 85670 May 30 18:36 60b3db2fe0ec4.jpeg
-rw-r--r-- 1 www-data www-data 96743 May 30 18:37 60b3db570e473.jpeg
-rw-r--r-- 1 www-data www-data 10710 May 31 09:47 60b4b0bc99740.jpeg
-rw-r--r-- 1 www-data www-data 188468 Jun 1 13:02 60b62fcd6c06c.jpeg
```

### Mostrar las publicaciones

Para mostrar las publicaciones, me creó en **functions.php** que muestre las publicaciones recientes y la llamaré getRecentPosts. Está función la creo AccionesUsuario, ya que esto lo puede ver cualquiera siendo o no usuario.

Recojo lo que es el id de la publicación, nombre, la imagen, quien lo creó y su extensión.

```
],[//Mostramos la publicaciones en pantalla
    "ORDER" => ["publicaciones.id_publicaciones" => "DESC"],
    "LIMIT" => "8"
]);
return $posts;
}
```

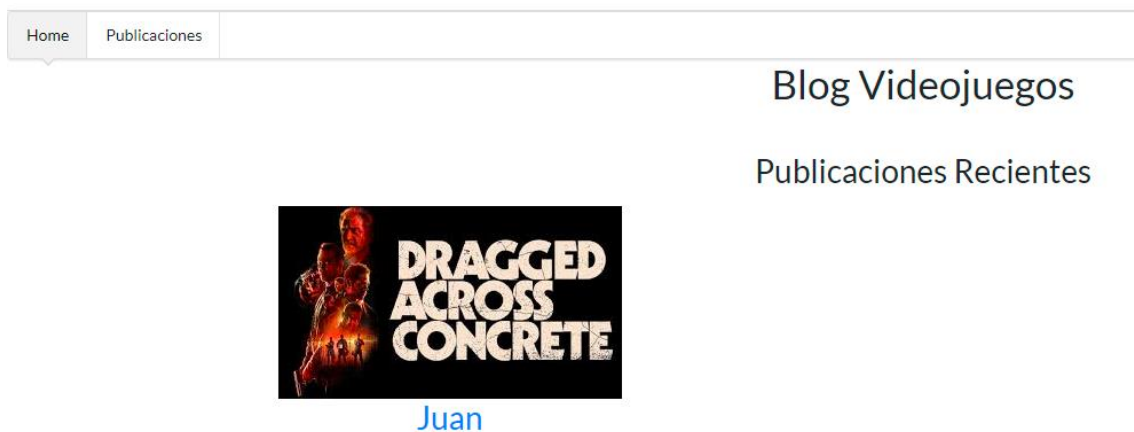
Con un order, le digo que muestre con una llamada sql que lo haga en orden descendente y que solo muestre 8 publicaciones.

Para llamar a la función, me voy al archivo **app\_top.php**, y ahí hago que llame a la función creada.

```
if(!isset($_GET['section'])){ $posts = $usuario->getRecentPosts();}
```

Ahora voy al **home.php** de view y con un foreach, recojo lo que es los datos y para ello, abro una etiqueta php y un echo, quiero que me recoja los siguientes datos:

- El id de la publicación
- La imagen y su extensión
- El nombre
- La fecha



[Mostrar las publicaciones individuales](#)

Voy a crear una ruta en el **.htaccess** que me lleve tanto a mi categoría como a publicación

```
RewriteRule ^post/([0-9]+)$ index.php?section=post&id_publicaciones=$1 [QSA]
```

Ahora me voy al **index.php** para poner la ruta con el isset y lo pondré la etiqueta post.

Me creo el archivo **post.php** pero antes, vuelvo al **index.php** para añadirle un http a los estilos para que no me pase lo mismo que me paso en el administrador.

Voy al **post.php** y lo que hago es recoger los campos con una etiqueta php en cada campo, es decir, una para el texto, otra para la imagen etc etc.

Al añadir los campos, tuve que ir al **app\_top.php** y me tuve que crear una variable para llamar a una función que tengo que crear para que me muestre los campos de forma individual en la publicación.

```
public function getPostInfo($id_publicaciones){
    global $database;

    $post = $database->select( table: "publicaciones",[
        "[>]categorias" => ["id_categorias" => "id_categorias"],
        "[>]usuarios" => ["id" => "id"],
    ],[
        "publicaciones.nombre",
        "publicaciones.texto",
        "publicaciones.img_publicaciones",
        "publicaciones.extension",
        "publicaciones.creado",
        "usuarios.usuario"
    ],[//Mostramos la publicaciones en pantalla
        "publicaciones.id_publicaciones" => $id_publicaciones
    ]);
    return $post;
}
```

Hago un join para llamar a las categorías y a los usuarios para que me recoja el campo usuario y la categoría.



## Spec Ops : The Line

05-06-2021 21:50:35 - admin

Spec Ops: The Line llega a las tiendas tras un largo proceso de desarrollo durante el que el juego llegó a estar desaparecido en combate. Mostrado por primera vez en el E3 de 2010, la idea de 2K Games y Yager es llevarnos a una ciudad de Dubái sumida en el caos tras una tormenta de arena: no una suave película, como las que se han visto en algunas películas ambientadas en la metrópoli emiratí, sino una verdaderamente devastadora, capaz de derribar edificios, destruir infraestructuras y convertir a esta ciudad tan de moda últimamente en un escenario casi post-apocalíptico, las calles desiertas y tomadas por los insurgentes y sus barricadas. En algún momento del suceso la unidad 33 del Ejército americano, liderada por el coronel Konrad, decidió cambiar su vocación de protección de los supervivientes por la de señores de la guerra, pero eso es algo que el Capitán Walker y los dos soldados a su mando todavía no saben cuando les envían a las ruinas de la ciudad tras recibir una señal de auxilio.

Reaparecido a finales del año pasado, Spec Ops ha evolucionado bastante desde aquel juego que vimos a mediados de 2010. La ambientación sigue siendo la misma y la arena juega un papel interesante en la jugabilidad, permitiendo usarla para acabar con los enemigos, pero parece que esta idea se ha dejado como anécdota para algunos momentos del juego y así centrar los esfuerzos en transmitir una historia que capture al jugador. Spec Ops se ha vuelto más oscuro, intentando arrastrar al jugador a la espiral de destrucción y locura en la que se ha convertido la ciudad.



## Formulario de registro del usuario

Me voy al **.htaccess** y pongo la ruta para que me lleve a la URL amigable de **register.php**

**RewriteRule** ^register\$ index.php?section=register [QSA]

Lo añadido en el menú y para el diseño, voy a coger el mismo diseño que hice en el admin. Pero cambiando los nombres de la clase por txtUserNameRegister y txtPassRegister.

## Formulario de log in del Usuario

Me creo una ruta en el **.htaccess** para llamar a la ventana de log-in de que el usuario inicie sesión, lo que hago es copiar el register y cambiando register por log-in. Y lo mismo para el **index.php** del inicio, hago un elseif y pongo log-in al final del get section. Para el diseño, pongo la misma ventana del admin, pero al igual que en el paso anterior, le cambio las clases de nombre poniendo LoginUser tanto al texto como al campo de la contraseña. Al archivo php lo llamo **register.php**

## Registrar Usuarios

Para el registro de usuarios me voy a crear una tabla llamada usuario, para que guarde los diferentes usuarios y un javascript llamado **main.js**

En el **register.php** me creo una clase en el botón de registrar usuario y así en el **main.js**, llamo al botón con un document ready function para luego hacer que llame a la función dándole un click y que recoja los campos de nombre y contraseña.

```

'''
$(".btnRegisterUser").on( types: "click", selector: function (){
    //Recojo los campos de cuando un usuario se registra
    var usuario = $(".txtUsernameRegister").val().trim(),
        clave    = $(".txtPassRegister").val().trim(),
        self     = this;

```

Para enviar los datos, hago lo mismo que he hecho en el **admin.js**.

```

if(usuario !== "" && clave !== ""){
    //Enviar los datos con Ajax
    $.ajax( url: {
        type: "POST",
        url: root + "php/Accionesusuario/register.php",
        data: {
            usuario : usuario,
            clave : clave
        },
        beforeSend: function (){
            $(self).addClass( value: "loading");
        },
        success: function (data){
            $(self).removeClass( value: "loading");
            $(".txtUsernameRegister, .txtUsernameRegister").val( value: "");
            alert(data);
        },
        error: function (){
            $(self).removeClass( value: "loading");
            alert("Error de servidor");
        }
    });
}
}
else{
    alert("Llene todos los campos vacios");
}

```

Me creo el archivo **register.php** para llamar a la función de que te registre el usuario y que también me valide la contraseña para ello utilizo un `preg_match`. Lo que quiero conseguir es que los usuarios cuando pongan sus contraseñas obligarles a que tenga al menos una mayúscula, una minúscula, un número, carácter especial y que tenga 8 caracteres.



```

<?php
require "../functions.php";
$user = new AccionesUsuarios();

// Given password
$password = $_POST['clave'];

// Validar la contraseña
$uppercase = preg_match( pattern: '@[A-Z]@', $password);
$lowercase = preg_match( pattern: '@[a-z]@', $password);
$number     = preg_match( pattern: '@[0-9]@', $password);
$specialChars = preg_match( pattern: '@[^\w]@', $password);

if($user->checkExistance($_POST['usuario'])){
    echo "Ya existe un usuario con el mismo nombre de cuenta.";
}

if(!$uppercase || !$lowercase || !$number || !$specialChars || strlen($password) < 8) {
    echo 'La contraseña debe tener al menos 8 caracteres de longitud y debe incluir al menos';
}
else
{
    $register = $user->Register($_POST['usuario'],$_POST['clave']);
    if($register){
        echo "Se ha registrado correctamente";
    }
}
?>

```

Me tengo que ir a **functions.php** a crearme dos funciones que son checkExistance y Register.

```

public function checkExistance($user){
    global $database;

    $usuarios = $database->count( table: "usuario", [
        "OR" => [
            "usuario" => $user
        ]
    ]);
    return $usuarios;
}

public function Register($user,$pass)
{
    global $database;
    if ($this->checkExistance($user) == 0) {
        $register = $database->insert( table: "usuario", [
            "usuario" => htmlentities($user),
            "clave" => password_hash($pass, algo: PASSWORD_BCRYPT)
        ]);
        return $database->id();
    } else {
        return false;
    }
}

```

Para chequear necesito que el campo usuario esté chequeado y luego en la función register, quiero que me chequee el usuario antes de insertarlo y también que la contraseña que esté encriptada para no poder verla en la base de datos.

#### Log in de los usuarios

Voy al **admin.js**, copio lo que es que el administrador logee sesión como admin, lo llevo al **main.js**, cambio lo campos y en donde pone window.location.href. quiero que me lleve a la raíz no al admin, cambio la ruta de accionesAdmin por Accionesusuario y llamo al archivo de **login-user.php** . Copio lo mismo que el archivo de **login.php**, le cambió admin por user y las acciones de administrador por usuario normal.

#### Cambio de menú al iniciar sesión como usuario

Para ello tuve que copiar el menú en el mismo **main\_nav.php** y que tenga dos menús. Le puse un isset para que cuando se inicie sesión, aparezca en el menú la pestaña de Mis publicaciones favoritas y que cierre sesión y para que cierre le pongo la ruta de **logout.php** para que me cierre.

#### Marcar favorito la publicación

Para que me salga el icono de una estrella, me tuve que ir a la documentación de Semantic UI y buscar un icono de estrella.

<https://semantic-ui.com/elements/icon.html>

La puse en el archivo **post.php** y luego para que me cheque si está accediendo un usuario, le pongo un if con sesión para el usuario. En el botón de star, le pongo un span con una clase para que me haga la función de favorito y llamo a esa función en el **main.js**. Lo hago igual que lo hacía para llamar a las categorías en el **admin.js**.

```

$(".btnMarkFavorite").on( types: "click", selector: function (){
    var post_id = $(this).attr( name: "data-idpublicaciones");

    $(this).remove();
    $.ajax( url: {
        type: "POST",
        url: root + "php/Accionesusuario/favorite.php",
        data:{
            post_id : post_id
        },
        success: function (data){
            if(data == "true"){
                alert("Artículo agregado a favoritos");
            }else{
                alert("Error...");
            }
        }
    });
});

```

### Favorite.php

```

<?php
require '../functions.php';
$user = new AccionesUsuarios();
//Obtener el perfil del usuario
$profile = $user->getProfile($_SESSION["user"]);
//Marcar como favorito
$favorite = $user->markAsFavorite($_POST["post_id"], $profile[0]["id"]);
if($favorite > 0) {
    echo "true";
}else{
    echo "false";
}
?>

```

Las funciones creadas en **functions.php**

```

public function markAsFavorite($id_publicaciones,$id){
    global $database;

    $database->insert( table: "favoritos",[
        "id" => $id,
        "id_publicaciones" => $id_publicaciones
    ]);
    return $database->id();
}

```

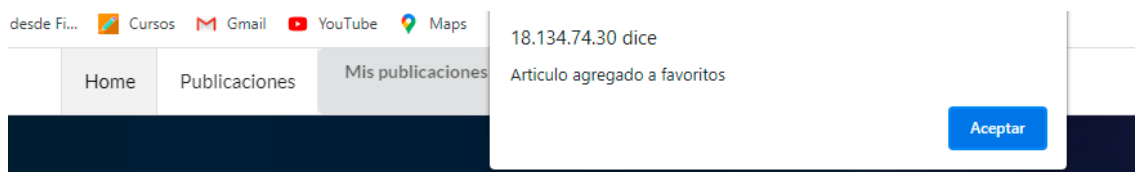
```

public function getProfile($user){
    global $database;

    $usuario = $database->select( table: "usuario",[
        "id"
    ],["OR" => [
        "usuario" => $user
    ]
    ]);
    return $usuario;
}

```

En get profile, lo que hago es que seleccione el id del usuario para que luego lo relacione con el usuario que creo el artículo que sería el admin.



## Yakuza 0

05-06-2021 21:30:02 - admin

### [Mostrar todas las publicaciones favoritas](#)

Para que me muestre las publicaciones, hago lo mismo que hice en **posts.php**, y luego en el **index.php** llamo a la ruta con un isset. y luego voy al **app\_top.php** donde quiero cargar las publicaciones favoritas, con una función creada en **functions.php** que realiza una consulta a la tabla de las publicaciones favoritas de los usuarios.

```

public function getMyFavorites($id){
    global $database;
    $posts = $database->select( table: "favoritos",[
        "[>]publicaciones" => ["id_publicaciones" => "id_publicaciones"]
    ], [
        "publicaciones.id_publicaciones",
        "publicaciones.nombre",
        "publicaciones.img_publicaciones",
        "publicaciones.creado",
        "publicaciones.extension",
        "favoritos.id_favoritos"
    ],[//Mostramos la publicaciones en pantalla
        "favoritos.id" => $id,
        "ORDER" => ["favoritos.id_publicaciones" => "DESC"],
        "LIMIT" => "8"
    ]);
    return $posts;
}



```

## App\_top.php

```

if(isset($_SESSION["user"])){
    //Obtener el perfil del usuario
    $profile = $usuario->getProfile($_SESSION["user"]);
    //Obtener publicaciones favoritas
    $posts = $usuario->getMyFavorites($profile[0]["id"]);
}
}

```

Home	Publicaciones	Mis publicaciones favoritas
<div>  <p><b>DarkSiders</b> 05-06-2021 21:45:44</p> <p>Eliminar Publicación</p> </div> <div>  <p><b>Yakuza 0</b> 05-06-2021 21:30:02</p> <p>Eliminar Publicación</p> </div>		

Para evitar que el usuario cheque la publicación más de una vez, me voy al **main.js** y le pongo un remove llamando a la función this, antes de que me coja todos los datos.

```
$(".btnMarkFavorite").on( types: "click", selector: function (){  
    var post_id = $(this).attr( name: "data-idpublicaciones");  
  
    $(this).remove();  
    $.ajax( url: {  
        type: "POST",  
        url: root + "php/Accionesusuario/favorite.php",
```

```
//Chequear que la publicación visitada ya este en favoritos del usuario  
$check = $usuario->checkFavorites($profile[0]["id"],$_GET["id_publicaciones"]);
```

En el **app\_top.php** hago que me cheque las funciones y para ello, me la tengo que crear en **functions.php**.

```
public function checkFavorites($id,$id_publicaciones){  
    global $database;  
  
    $usuarios = $database->count( table: "favoritos",[  
        "AND" => [  
            "id" => $id,  
            "id_publicaciones" => $id_publicaciones  
        ]  
    ]);  
    return $usuarios;  
}
```

Vuelvo ahora al **post.php** para ponerle una variable (que la llamo igual que el **app\_top.php**) para que recoja el chequeo.

```
//Chequear que la publicación visitada ya este en favoritos del usuario  
if(isset($_SESSION["user"]))  
{  
    $check = $usuario->checkFavorites($profile[0]["id"], $_GET["id_publicaciones"]);  
}else{  
    $check = false;  
}
```

Para eliminar una publicación de favoritos, fui a **main.js** y hacer exactamente lo que hice para añadir las publicaciones. Me creo un archivo en AccionesUsuario que se llamará **delete\_favorite.php** y que a través del id de favoritos, me lo elimine.

```
<?php
require '../functions.php';
$user = new AccionesUsuarios();

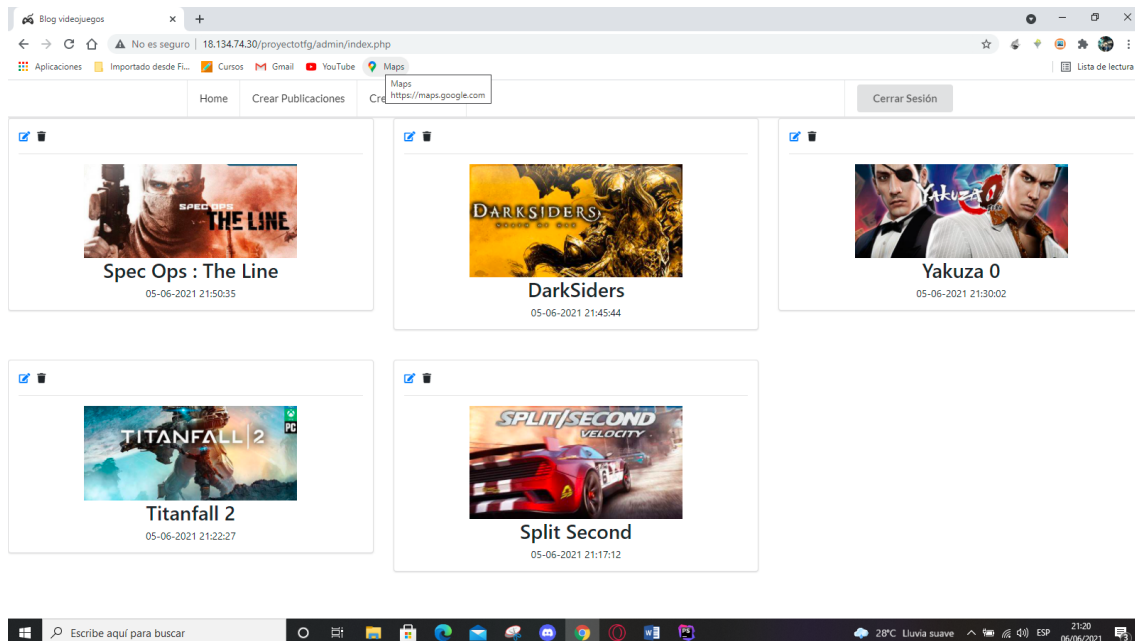
//Eliminar como favorito
$delete = $user->deletefavorite($_POST["favorite_id"]);
if($delete > 0){
    echo "true";
}else{
    echo "false";
}
?>
```

```
public function deletefavorite($favoritosid){
    global $database;

    $delete = $database->delete( table: "favoritos", [
        "id_favoritos" => $favoritosid
    ]);
    return $delete->rowCount();
}
```

[Vista para el administrador](#)

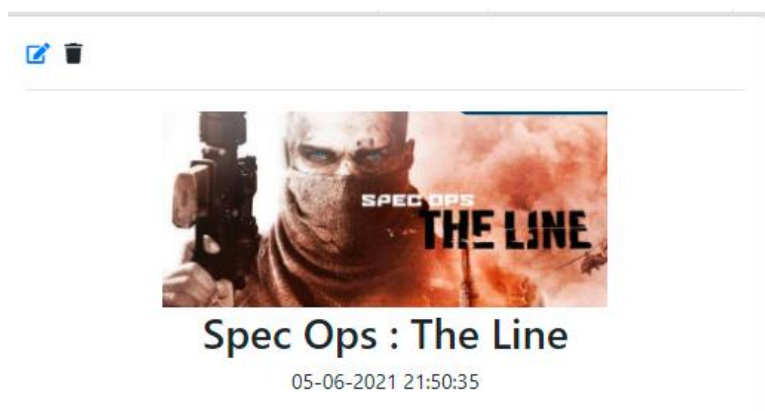
Para la vista de administrador, copio y pego el **home.php** y ya lo tendría.



### Editar y eliminar una publicación

Ahora que me crea una publicación, me crea una categoría y todo está funcional, voy a meterle más cosas a las publicaciones para que puedan hacer más funciones.

Para eliminar la publicación, en el Javascript de **admin.js** cogí el **btnSavePost** y que hiciera exactamente lo mismos pasos pero que ahora se llamará **btnDeletePost** y que tuve que crearme en accionesAdmin un archivo que se llama **delete\_post.php**. En el archivo creado, hice lo mismo que en **delete\_category.php**, pero en este caso que lo eliminará con el id de la publicación. En **functions.php** hago que me elimine por completo la publicación con un **delete**. Arriba de la publicación, me aparece el icono de borrar





Ahora voy con editar, que es más o menos igual, aquí en el **main.php** de admin, tuve que hacerme un id para que recogiera todos los campos de la publicación y también que el botón lo relacionara.

```
<a href="edit-post/<?php echo $post['id_publicaciones'];?>"><i class="edit icon"></i></a>
```

Le pongo un edit-post para que me lo reconozca con el id de la publicación y después me voy al **.htaccess** para que me coja el id y el editar.

```
RewriteRule ^admin/edit-post/([0-9]+)$ admin/index.php?section=edit-post&id_publicaciones=$1 [QSA]
```

Ahora voy al **admin.js** y copio el código de lo que es el borrar categoría y edito la ruta.

Ahora me creo el **edit\_post.php** en view/admin. En el view, simplemente lo que hago es copiarme lo que tenía en **posts.php** de view/admin y ya. Ahora en el archivo de AccionesAdmin me creó el **update\_post.php** y copio lo mismo que **new\_post.php**

## Activar y desactivar las publicaciones en el Home

Voy a **functions.php** y en las funciones `getPost` (tanto de admin como de usuario), `getMyFavorites` y `getRecentPosts` le pongo el campo activo con 1, .

En **functions.php**, me creo dos funciones que se llamarán `disablePost` y `Activepost` para que desactive y active las publicaciones en el Home. Me lo creo en la clase `AdminAcciones`.

```
public function disablePost($id_publicaciones)
{
    global $database;
    try {
        $data = $database->update( $table: "publicaciones", [
            "activo" =>0
        ],
        ["id_publicaciones" => $id_publicaciones]);
    } catch (PDOException $exception) {
        echo $exception->getMessage();
    }
    return $data->rowCount();
}

public function enablePost($id_publicaciones)
{
    global $database;
    try {
        $data = $database->update( $table: "publicaciones", [
            "activo" =>1
        ],
        ["id_publicaciones" => $id_publicaciones]);
    } catch (PDOException $exception) {
        echo $exception->getMessage();
    }
    return $data->rowCount();
}
```

Luego en el **main.php** , de view/admin le pongo dos iconos de ojos con el Semantic UI, le pongo un if en el caso de que se desactiva, se muestre el ojo y gracias a una clase del Semantic UI, puedo poner que se vea la imagen borrosa y cuando se active, se vea más clara.

```
<?php if($post['activo']){?>

    <i class="eye slash icon btnViewDisable" data-publicacionesId="<?php echo $post['id_publicaciones'];?>"></i>
    ">

<?php } else { ?>
    <i class="eye icon btnViewActive" data-publicacionesId="<?php echo $post['id_publicaciones'];?>"></i>
    ">

<?php } ?>
```

Me creo en los botones un data-publicacionesId para que me recoja el id de las publicaciones. Me voy al **admin.js** donde hago la llamada a las clases de activar y desactivar en el javascript. Hago lo mismo que en hice en categorías, tanto para activar como desactivar. Me creo para desactivar en la carpeta de AdminAcciones, el archivo llamado **disable\_post.php** y para activar lo llamo **enable\_post.php**. Tanto en **enable\_post.php** como en **disable\_post.php** recojo el id de la publicación.

Compruebo que funciona



Desactivado



Activado

Lo que hace es cambiar el botón.

## Valoración económica

<https://kaira.es/cual-es-el-precio-hora-de-un-programador/>

20 euros por hora

60 horas

6 horas documentación X 20 e

6 horas puesta en marcha del servidor, base de datos y ajustes de permisos x 20 e

48 horas de desarrollo x 20

1200 euros

## Conclusión

Ha sido un proyecto en donde he puesto en práctica todo lo aprendido en estos dos años en la formación profesional. Me ha servido de mucho este proyecto para darme cuenta de cosas que había aprendido, que pensé que eran inútiles, me han servido de mucho.

### Dificultades encontradas

Las dificultades, han sido en cuanto a logear los usuarios y lo relacionado **.htaccess** por lo demás ha sido repetir el mismo proceso más o menos. También he tenido problemas con lo relacionado con los vídeos, no he podido añadirlo por falta de tiempo y de probarlo con más certeza.

### Opciones de mejora

Las opciones que podría haber mejorado, serían lo relacionadas, pese a que es un diseño que si bien es sencillo y funcional, creo que debería haber mejorado más el diseño.

