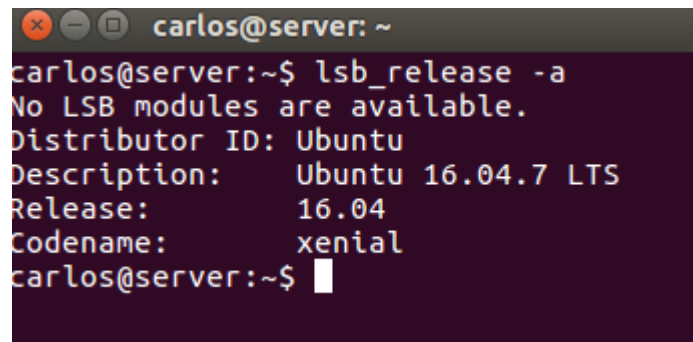


IMPLEMENTACIÓN

El primer punto de la memoria se lo quiero dedicar a la implementación que he utilizado para desarrollar el proyecto. Básicamente he montado un servidor WEB en un PC de sobremesa que tiene unos 15 años. Es un PENTIUM 4 a 2,8 GHZ con 3 GB de memoria RAM a 667 GHZ. Como se puede ver una maravilla de PC, pero aún así la verdad opiniones a parte funciona mejor que mi otro equipo ACER I5 con 8 GB de memoria RAM DDR3. Serán cosas de Windows ...

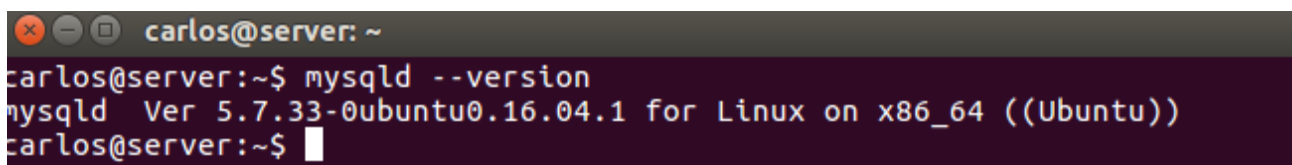
Pongo una captura de la versión de Ubuntu que estoy usando.

A terminal window titled 'carlos@server: ~' showing the output of the 'lsb_release -a' command. The output lists the distributor ID as Ubuntu, description as Ubuntu 16.04.7 LTS, release as 16.04, and codename as xenial.

```
carlos@server:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.7 LTS
Release:        16.04
Codename:       xenial
carlos@server:~$
```

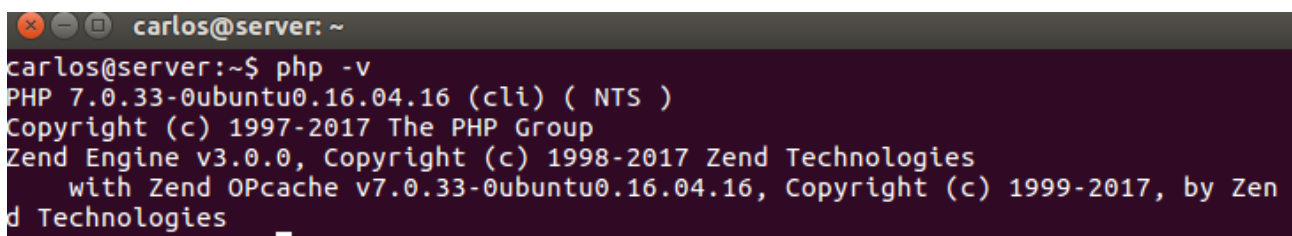
Después instale Apache2, PHP 7 y MYSQL :

```
sudo apt-get install apache2
sudo apt install mysql-server
```

A terminal window titled 'carlos@server: ~' showing the output of the 'mysql --version' command. The output indicates MySQL Ver 5.7.33-0ubuntu0.16.04.1 for Linux on x86_64 ((Ubuntu)).

```
carlos@server:~$ mysql --version
mysql Ver 5.7.33-0ubuntu0.16.04.1 for Linux on x86_64 ((Ubuntu))
carlos@server:~$
```

```
sudo apt-get install php libapache2-mod-php php-mysql
```

A terminal window titled 'carlos@server: ~' showing the output of the 'php -v' command. The output displays PHP 7.0.33-0ubuntu0.16.04.16 (cli) (NTS), copyright information for The PHP Group and Zend Technologies, and details about the Zend Engine and Zend OPcache.

```
carlos@server:~$ php -v
PHP 7.0.33-0ubuntu0.16.04.16 (cli) ( NTS )
Copyright (c) 1997-2017 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2017 Zend Technologies
    with Zend OPcache v7.0.33-0ubuntu0.16.04.16, Copyright (c) 1999-2017, by Zen
d Technologies
carlos@server:~$
```

Luego cree un VirtualHost para el dominio NO-IP:

```
VirtualHost *:8080>
# The ServerName directive sets the request scheme, hostname and port to
# the server uses to identify itself. This is used when creating
# redirection URLs. In the context of virtual hosts, the ServerName
# specifies what hostname must appear in the request's Host: header to
# match this virtual host. For the default virtual host (this file) this
# value is not decisive as it is used as a last resort host regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

ServerAdmin webmaster@localhost
ServerName changes.ddns.net
ServerAlias www.changes.com
DocumentRoot /var/www/html/Changes/index.php

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# virtual hosts, using the
# LogLevel <level> directive
```

Para poder acceder desde el exterior de mi red a la web he tenido que abrir los puertos al router. Aquí si nos fijamos en el VirtualHost veremos que también modifique el puerto de escucha de Apache al 8080, esto es por que el 80 lo usa mi proveedor de ISP para acceder a mi router.

Mapping Name	WAN Name	Internal Host	External Host	Enable
--	1_TR069_VOIP_IPTV_INTERNET_R_VID_100	192.168.1.100	--	Enable

Type: ☒ User-defined ☐ Application Select...

Enable Port Mapping: ☒

Mapping Name:

WAN Name: 1_TR069_VOIP_IF


Internal Host: 192.168.1.100 server

External Source IP Address: --

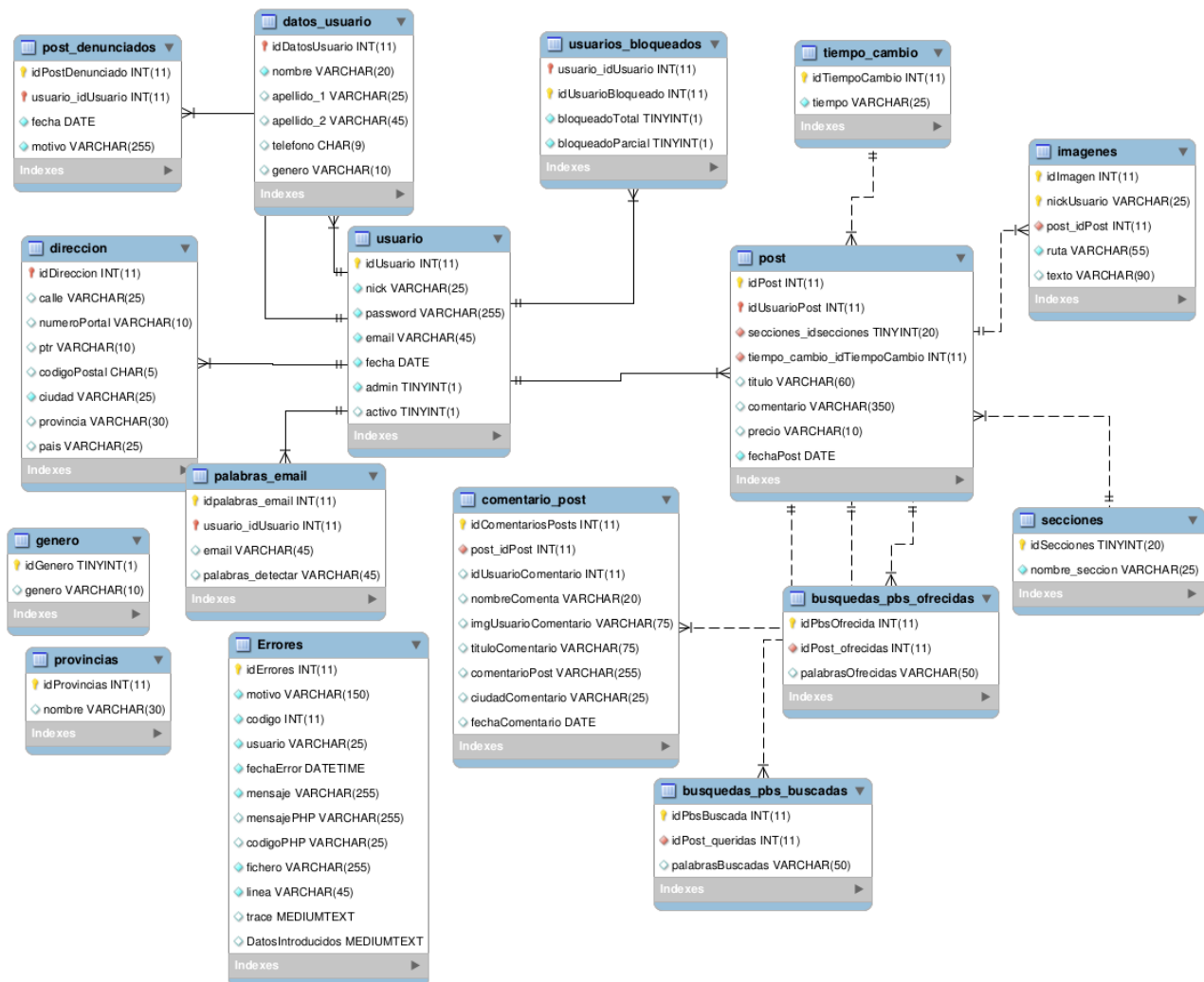
Protocol: TCP/UDP Internal port number: 8080 -- 8080

External port number: 8080 -- 8080 External source port number: --

Por último cree una cuenta en NO-IP para redirigir un dominio gratuito a mi IP

Hostname	Last Update	IP / Target	Type
 changes.ddns.net Active	Nov 4, 2021 09:05 PDT	37.221.239.142:8080/Changes/Vista/index.php	URL

BASE DE DATOS



Como se puede apreciar la base de datos esta compuesta por 2 entidades principales que son Usuario y Post. Estas dos entidades tienen otras entidades débiles y relaciones que veremos adelante. Al mismo tiempo estas entidades están relacionadas entre si de 1:N. También se puede apreciar tres tablas independientes que son genero, provincias y Errores. Esta claro que genero debería ser una relación 1:1 con datos_usuario y provincias el mismo tipo con la tabla dirección para tener un grado de normalización de tipo 3FN pero tampoco se buscaba llegar a ese nivel. Luego esta la tabla Errores que básicamente guarda los errores que se producen para luego ser analizados.

Comenzaremos describiendo a la **entidad usuario**. Vemos que esta relacionada con dos tablas 'datos_usuario y dirección' de 1:1. Estas dos entidades son débiles y **con dependencia de identificación** por lo que necesitan de la clave primaria de la entidad usuario. Aquí tenemos una normalización de tipo **2FN**, podemos ver

claramente que de esta forma estas tablas nos servirían también por ejemplo para ‘trabajadores, empleados, etc etc’.

También vemos que la **entidad usuario** tiene relaciones con las tablas ‘usuarios_bloqueados, post_denunciados y palabras_email’. Estas relaciones son de 1:N por que un usuario puede bloquear a otros tantos, puede denunciar varios post y guardar en la tabla palabras_email muchas palabras. Esta última tabla es donde un usuario ingresa unas palabras que serán comparadas con las que usen el resto de usuarios cada vez que publican un Post para recibir un email si hay coincidencia. Esto se vera más adelante. Hay que decir que la relación usuarios_bloqueados es una relación reflexiva.

Continuamos ahora con la **entidad Post**. Como se puede ver esta entidad esta relacionada con otras dos entidades aparte de usuario que son débiles y dependen de esta para existir. La relación con imágenes es a través de la clave FK de la entidad de post. Luego tenemos la relación con comentarios que básicamente es el mismo tipo de relación. Luego tenemos una relación con la tabla secciones que es una relación 1:N pero esta vez inversa. Osea el post puede tener únicamente una sección y una sección puede tener muchos post, por eso tenemos la clave primaria de la tabla sección en la tabla post y no a la inversa. Por último la entidad post tiene una relación con dos tablas que son ‘busquedas_pbs_buscadadas y busquedas_pbs_ofrecidas’ que son dos tablas donde se almacenan unas frases que describen el objeto que se va cambiar y por lo que se quiere cambiar.

Quiero dejar un detalle aquí, y es que a las tablas ‘ busquedas_pbs_buscadadas, busquedas_pbs_ofrecidas y sección’ **no las he tratado como entidades**. Esto es por que no lo son, ya que no dejan de ser atributos de la entidad Post.

Por último quiero decir en COLLATION de la bbdd. Hasta hace poco usaba el típico UTF-8 hasta que leyendo por internet descubrí que existía el utf8mb4_unicode_ci. ¿Que diferencia hay con UTF-8? Que este admite 4 bytes por campo en lugar de 3 como el UTF-8. Por lo que se pueden usar emoticonos y admite todos los caracteres del mundo.

Referencias:

<https://gestionbasesdatos.readthedocs.io/es/latest/Tema2/Teoria.html>

https://ikastaroak.birt.eus/edu/argitalpen/backupa/20200331/1920k/es/DAMDAW/BD/BD02/es_DAMDAW_BD02_Contenidos/website_31_tipos_fuertes_y_dbiles.html

<https://www.vivablogger.com/convertir-utf8-a-utf8mb4/>

https://docs.moodle.org/all/es/MySQL_soporte_unicode_completo/

ARQUITECTURA

La arquitectura del sistema ha sido MVC, separada cada sección por directorios. Empezaremos describiendo la capa de Modelo.

Esta capa que es la encargada de la funcionalidad del sistema y acceder a los datos, tiene 3 clases. La principal DataObject que es abstracta y extienden 2 de las subclases que son Usuario, Post. He decir que no es necesario para que el sistema funcione obligatorio crear esta clase, pero a nivel educativo me pareció interesante. Básicamente tiene un constructor y un método que nos devuelve el valor de un campo de un array. En un contesto más formal esta clase debería de tener los métodos comunes a todas las subclases.

La clase usuario tiene un única propiedad que es el array de la clase DataObject que contiene todos las propiedades de la clase Usuario. No voy a entrar a describir los métodos que tiene cada clase pero son los típicos insert, delete, update y obtener algún campo específico como Id o Nick etc.

La clase Post utiliza el mismo sistema de instanciación. Dispone de sus métodos propios que veremos adelante según vallamos explicando la funcionalidad del sistema.

Por último tenemos una clase que Email que es la encargada de mandar los emails según sea para registrarse, darse de baja etc . Esta usa PHPMailer para hacer su trabajo.

La capa Vista que es la encargada de hacer de interfaz con el usuario, en este trabajo son formularios y mostrar anuncios. Básicamente hay dos páginas principales que son index.php que es donde se muestran todos los anuncios y registrarse.php que sirve para registrarse o actualizar los datos del usuario.

La capa Controlador trabaja de intermediario entre Modelo y Vista. Esta recoge los datos de los formularios mostrados en la capa Vista. Hay varias opciones además de mostrar los anuncios como dar de alta usuario, actualizar, hacer un comentario, bloquear un usuario, etc. Cada opción tiene dos archivos. El primero recoge los datos que el usuario introduce en los formularios y luego hace una petición JSON a un archivo con el mismo nombre .php. Este archivo es el que accede a la BBDD para trabajar los datos si fuera necesario o devuelve los datos si hubiese para mostrarlos.

Luego tenemos un directorio llamado Sistema donde encontramos las constantes utilizadas en todo el sistema, PHPMailer, una clase directorios con

métodos para crear, copiar, borrar archivos etc. Básicamente esta clase es la encargada de trabajar con el Sistema de archivos del SO.

También hay una clase que tiene dos métodos, uno para generar el Hash de la contraseña del usuario y otro para compararlos.

REGISTRAR UN USUARIO

El proceso de registro sigue los pasos comunes. Es un formulario en varios pasos donde se piden datos, se validan y si son correctos se pasa al siguiente. El trabajo de validación es realizado tanto por JQUERY como PHP. Con el primero se realiza una validación del tipo campos vacíos, obligatorios, etc. Con PHP volvemos a hacer este trabajo 'es una decisión mía' pues me siento más seguro con PHP. Además este es el encargado de comprobar si un nick, cuenta de correo etc ya existe accediendo a la BBDD. Los archivos encargados de esto son formulario_reg.js y validoForm.php. Esta última implementa una interfaz que obliga a implementar varios métodos. Una vez más esta claro que no es obligatorio para el sistema y lo hice a nivel de estudio.

En el primer paso del formulario nos pide introducir un nick, password y correo. Se valida que ningún campo este vacío o que el nick o correo ya existan en el sistema.

2	34	pedro	\$2y\$09\$Vq6p76Fd3kzTvmfbbqcJ4.h0jicXFPHO...	carlosneiraweb@gmail.com	2021-12-03	0	1
---	----	-------	---	--------------------------	------------	---	---

En el segundo paso se nos pide los datos propios de usuario, como nombre apellidos etc. Aquí solo es obligatorio los campos nombre y teléfono.

34	pedro	garcia	serrano	666214536	hombre
----	-------	--------	---------	-----------	--------

El tercer paso se nos piden los datos de la dirección. En este caso los únicos campos obligatorios son ciudad y código postal.

2	34	c/serrania	15	5	46500	Valencia	No importa
---	----	------------	----	---	-------	----------	------------

El cuarto paso se puede subir una imagen de perfil. No es obligatorio pero si no de hace se pone una por 'default'. Aquí se controla que la imagen sea de tipo .jpg, si no es así nos avisa.



Upss !!!

Hemos tenido un problema.
Únicamente aceptamos imagenes .jpg

Aceptar

Si todo ha ido bien crearemos tres directorios en el sistema con el nick de usuario en los directorios 'Videos, photos y datos_usuario'. También se podría haber usado el ID del usuario.

El quinto paso nos sale un mensaje si todo ha ido bien y ya podemos usar el portar, además recibimos un correo con nuestros datos.

Te lo cambio

Miles de personas compartiendo te están en esperando.

Enhorabuena pedro por registrarte en **Te Lo Cambio**

Ahora podrás cambiar con nuestro usuarios.

Recuerda que tú usuario es: pedro

Y tu password es: 987654321

ACTUALIZAR UN USUARIO

En el menú de usuario hay una opción que nos permite actualizar nuestros datos. Podemos cambiar la foto de perfil, nick de usuario, password, direccion etc. Lo primero que solicitamos es el nick y password actuales para empezar con el proceso. Esto lo hacemos en el mismo menú que llama al método 'pedirDatosParaActualizar' del archivo actualizarDatos.js. Si son correctos entonces se llama al método 'actualizarDatosUsuario' del mismo archivo que hace una instancia del objeto usuario

en una variable de sesión que se llama ‘\$_SESSION[‘actualizo’]’. Esta variable contiene todos los datos del usuario.

```
$_SESSION['actualizo']['nick'] = $tmpAct[0][0];
$_SESSION['actualizo']['correo'] = $tmpAct[0][1];
$_SESSION['actualizo']['calle'] = $tmpAct[0][2];
$_SESSION['actualizo']['portal'] = $tmpAct[0][3];
$_SESSION['actualizo']['puerta'] = $tmpAct[0][4];
$_SESSION['actualizo']['codigoPostal'] = $tmpAct[0][5];
$_SESSION['actualizo']['ciudad'] = $tmpAct[0][6];
$_SESSION['actualizo']['provincia'] = $tmpAct[0][7];
$_SESSION['actualizo']['pais'] = $tmpAct[0][8];
$_SESSION['actualizo']['nombre'] = $tmpAct[0][9];
$_SESSION['actualizo']['primerApellido'] = $tmpAct[0][10];
$_SESSION['actualizo']['segundoApellido'] = $tmpAct[0][11];
$_SESSION['actualizo']['tlf'] = $tmpAct[0][12];
$_SESSION['actualizo']['gn'] = $tmpAct[0][13];

echo PHP_EOL;
```

Con echar un vistazo al formulario registrar.php veremos como vamos a ir mostrando los datos actuales al usuario para que pueda ir modificándolos.

```
echo '<input type="text" name="ciudad" id="ciudad" placeholder="Nombre de tu Localidad" maxlength="25" value=' .
; if(isset($_SESSION['usuario']['ciudad']) && (!isset($_SESSION['actualizo']))) {echo $_SESSION['usuario']['ciudad'];}

if(isset($_SESSION['actualizo']['ciudad'])) {echo $_SESSION['actualizo']['ciudad'];} echo ">";
```

Hemos de tener en cuenta que ahora vamos a trabajar con dos variables de sesión. La primera es ‘\$_SESSION[‘usuario’]’ que es la que va recogiendo los datos que el usuario va escribiendo en el formulario al registrarse. Por eso podemos ir hacia atrás por los distintos pasos en el formulario y no perder la información. Pero cuando usamos la opción de actualizar lo que hacemos es no mostrar un formulario vacío sino ya relleno con los datos actuales que tiene el usuario que es la variable ‘\$_SESSION[‘actualizo’]’. Los nuevos datos que introduzca el usuario se guardan en la variable ‘\$_SESSION[‘usuario’]’ que al finalizar el proceso en lugar de llamar al método de la clase usuario insertar se llama al método actualizar.

Lo primero que hacemos antes de nada es conservar los datos que tenemos del usuario por si hubiese algún error en el proceso. Esto se hace tanto a nivel de BBDD utilizando **una transacción con commit y rollback**. En caso de error al actualizar cualquier tabla nos devolvería al punto inicial. La segunda parte es hacer una copia temporal de las imágenes, vídeos y foto personales por si hubiera un error. Si todo finaliza bien esta copia es eliminada.

Voy a explicar aquí, aunque su uso se hace en todos los archivos, dos clases llamadas *'MisExcepciones que extiende MetodosInfoExcepciones'*. Estas clases recogen los típicos errores que todos conocemos de PHP cuando estamos haciendo un programa y cometemos un error. En pantalla nos sale escrito cuando PHP no está en modo seguro. Pero cuando lo configuramos y lo dejamos sin el **Modo Seguro** y ponemos **Display errors = off** solo nos queda ir al archivo `/var/log/apache/error.log` para ver los errores. Básicamente estas dos clases lo que hacen es crear un objeto con dos propiedades que tienen un código personal y una descripción personal del error. Además **reciben la variable \$ex de la clase Exception de php** con los típicos `$ex→getMessage`, `$ex→getCause` etc, etc. Además recogemos la información que el usuario ha ido introduciendo y lo almacenamos en la tabla Errores de la bbdd.

El objetivo de esto, aunque no está implementado era poder hacer consultas sobre donde se estaban produciendo los errores. Cada día, semana, etc y ver los datos que estaban introduciendo los usuarios.

Iré poniendo ejemplos en capturas de esta tabla provocando yo errores.

Primero hacemos una copia de todos los archivos del usuario. Esta se guarda en el directorio Sistema/TMP_ACTUALIZAR. Pongo una captura en caso de error para ver lo explicado anteriormente, está claro que yo provoque el error aposta :

Form Editor Navigate: 1 / 2 Edit:

IdErrores: 1

Error al crear directorios TMP

Motivo:

Codigo: 11

Usuario: pedro

FechaError: 2021-12-07 18:05:19

Mensaje: No se pudo crear el directorio padre TMP

MensajePHP: Hubo un problema al crear los directorios TMP al actualizar. Posiblememnte ya existia

CodigoPHP: 0

Fichero: /var/www/html/Changes/Sistema/Directorios.php

Linea: 884

Trace: #0 /var/www/html/Changes/Controlador/Validar/ControlErroresSistemaEnArchivosUsuarios.php(141): Directorios::crearDirectorioPadreTMP('../Sistema/TMP...')
#1 /var/www/html/Changes/Controlador/Validar/ControlErroresSistemaEnArchivosUsuarios.php(172): crearDirectoriosTMP()
#2 /var/www/html/Changes/Controlador/Validar/ControlErroresSistemaEnArchivosUsuarios.php(473):

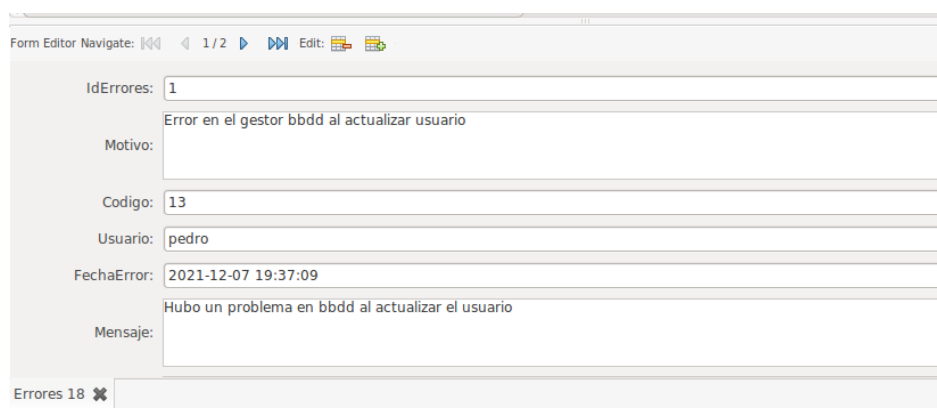
Datos antes de la actualizacion
Error al crear directorios TMPnick => pedrocorreo => carlosneiraweb@gmail.comcalle => aaaaaaaaaaaaaportal => aaaaaaaaaaapuerta => aaaaaaaaaaacodigoPostal => 44444ciudad => valenciaprovincia => Ciudad Realpais => aaaaaaaaaaaaaaaaanombre => pedroprimerApellido => aaaaaaaaaaaaaasegundoApellido => aaaaaaaaaaaaaaaaatlf => 666666666gn => hombreDatos introducidos por el usuario al actualizar.
nick =>

Como se ve se añade dos mensajes personalizados como se ha comentado y se recogen los errores internos de PHP. Esto se aplica sobre todo a los principales métodos del Sistema. Además en caso de error siempre se intenta dejar el sistema libre de basura. Y restaurando los directorios que el usuario tenía antes de la actualización.

Si todos los directorios se han renombrado entonces iremos a la clase usuario para hacer **un update** de las tablas correspondientes. Llamaremos a este método desde el mismo archivo registrarse.php desde el método ingresar, que según exista o no la variable ‘\$_SESSION["userTMP"]’ recordemos que se instancia con un objeto usuario cuando inicia sesión un usuario llama al metodo insert o update de la clase usuario.

Este método hace el update de las tablas necesarias con una transacción. En caso de error devuelve su antiguo valor a las tablas y llama a los métodos adecuados para restaurar los antiguos directorios y eliminar los posibles nuevos que se hayan podido crear. De esta manera se evita que el sistema se llene de basura y acabe inservible.

Pongo captura de ejemplo de error en la BBDD al actualizar:



Form Editor Navigate: 1 / 2 Edit: [Icons]

IdErrores:	1
Motivo:	Error en el gestor bbdd al actualizar usuario
Codigo:	13
Usuario:	pedro
FechaError:	2021-12-07 19:37:09
Mensaje:	Hubo un problema en bbdd al actualizar el usuario

Errores 18 ✖

MensajePHP:	SQLSTATE[HY093]: Invalid parameter number: parameter was not defined
CodigoPHP:	HY093
Fichero:	/var/www/html/Changes/Modelo/Usuarios.php
Linea:	688

```
#0 /var/www/html/Changes/Modelo/Usuarios.php(688): PDOStatement->execute()
#1 /var/www/html/Changes/Vista/registrarse.php(488): Usuarios->actualizoDatosUsuario()
#2 /var/www/html/Changes/Vista/registrarse.php(616): ingresarUsuario()
#3 /var/www/html/Changes/Vista/registrarse.php(154): processFormRegistro(Array, 'step4')
```

al actualizar.
nick => pedro2password => 987654321email => carlosneirawebssssssssss@gmail.comnombre => pedroapellido_1 => qqqqqqqqqqqqqqqqqqqqqqqqapellido_2 => qqqqqqqqqqqqqqqqqqqqqqqqtelefono => 777777777genero => mujercalle => qqqqqqqqqqqqqqqqqqqqqqnumeroPortal => qqqqqqqqqqqptr => qqqqqqqqqqciudad => qqqqqqqqqqqqqqqqqqqqqqqqcodPostal => 44444provincia => No importapais => qqqqqqqqqqqqqqq

SUBIR UN POST

El proceso de subir un post sigue la misma metodología que el de registrarse. Es un formulario dividido en varios pasos. Para poder publicar hay que estar registrado y haberse logueado.

El primer paso se nos pide un título para el anuncio, elegir una sección, hacer un comentario, un precio aproximado del producto a cambiar, por cuánto tiempo se va a cambiar, unas 4 pequeñas frases para describir por el producto que se quiera cambiar y 4 pequeñas frases que describan el producto a cambiar. Aquí los únicos campos obligatorios son el título, la descripción y el precio.

Cuando pasamos al siguiente paso se crea en el directorio `photos/nombre_usuario` un directorio donde se almacenara las imágenes que se vayan a subir. Para crear este directorio usamos el archivo `'ControlErroresSistemaEnArchivosPost'` que como su nombre indica es el encargado de llamar a los métodos de clase `Directorios` para crear, copiar y mover los archivos cuando se sube un Post.

Hay que tener en cuenta que trabajamos con dos sistemas. El primero es el Sistema de archivos del OS y el segundo el de la BBDD. Este archivo es el encargado de asegurar que no se corrompa el sistema en caso de un error. Más tarde se explicara con más detalle.

Llamamos al método crear subdirectorio:

```
function crearSubdirectorio(){
    /**/
    $nickUsu = $_SESSION['userTMP']->getValue('nick');
    //[0] nombre usuario
    $_SESSION['nuevoSubdirectorio'][0] = $nickUsu;
    //[1] numero subdirectorio ejemplo "1"
    $_SESSION['nuevoSubdirectorio'][1] = Directorios::crearSubdirectorio(' ../photos/' . $nickUsu, "crearSubdirectorio");
    // echo 'responde ' . " ../photos/" . $_SESSION['nuevoSubdirectorio'][0] . '/' . $_SESSION['nuevoSubdirectorio'][1];
    Directorios::copiarFoto("../photos/demo.jpg", " ../photos/" . $_SESSION['nuevoSubdirectorio'][0] . '/' . $_SESSION['nuevoSubdirectorio'][1]);
    //fin crearSubdirectorio
}
```

‘\$_SESSION[‘userTMP’] → getValue(‘nick’)’;

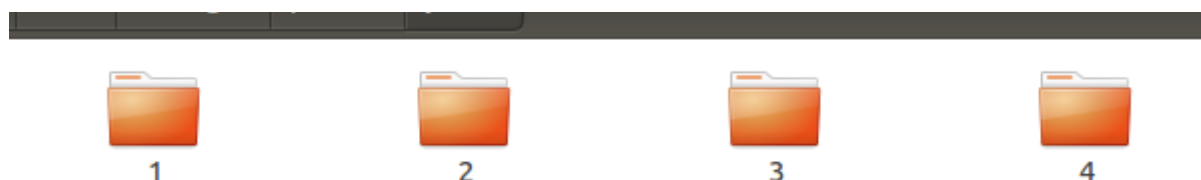
Esta variable se instancia cuando el usuario se ‘loguea’ para entrar en el portal este paso me lo salte por que no tiene más importancia. Lo importante es que esta variable de sesión contiene un objeto de usuario que es una instancia del array de la clase abstracta DataObject.

¿Os acordáis del método de esta clase que nos permitía acceder a cualquier campo del array data? Aquí hay un ejemplo, además de acceder con el típico ‘\$this → ’ en la clase del objeto usuario podemos acceder a cualquier propiedad de un objeto usuario o Post con el método de la clase abstracta.

\$_SESSION[‘nuevoSubdirectorio’]

Aquí almacenaremos el directorio donde se guardara el Post. El primer campo es el nick y para saber el segundo utilizamos el método crearSubdirectorio de la clase Directorios. Esta clase lo único que hace es contar los directorios que tiene ya creados en la directorio ../photos/nickUsuario. Por ejemplo si un usuario tiene 3 Posts y quiere publicar un cuarto devolvera un 4. Aquí también se podría haber usado el ID del Post para poner de nombre al directorio. Pero los pasos deberían ser al revés, primero ingresar el Post en la BBDD para saber el ID y luego crear los directorios con el ID del Post.

Ejemplo para usuario Pedro que tenia 3 Post publicados : ../photos/pedro/4.



Directorios::copiarFoto("../photos/demo.jpg", " ../photos/" .
\$_SESSION[‘nuevoSubdirectorio’][0] . '/' . \$_SESSION[‘nuevoSubdirectorio’]
[1] . "/demo.jpg", "copiarDemoSubirPost");

Aquí lo que hacemos es subir una imagen ‘demo’ por si el usuario no quiere subir ninguna imagen. Tras realizar este paso insertamos en la BBDD el Post. Para ello llamamos al método ‘ingresarPost’ que instancia un objeto Post con las variables ‘\$_POST[“tituloSubirPost”]’ del formulario. Una vez tenemos la instancia llamamos al método ‘insertPost’ de la clase Post para que haga la inserción.

Este método hace un ‘commit’ a las tablas correspondientes y inserta en la BBDD en la tabla imágenes la ruta de la imagen demo.

#	idImagen	post_idPost	nickUsuario	ruta	texto
1	43	14	carlos	1/1	vvvvv
2	44	14	carlos	1/2	vvvv
3	46	15	carlos	2/1	hfda adad
4	48	16	carlos	3/1	tha a aht
*	NULL	NULL	NULL	NULL	NULL

Veamos como como se suben las imágenes.

Cuando un usuario en el segundo paso sube una imagen lo primero que hacemos es validarla. Para ello usamos el método ‘Directorios::validarFoto’. Este método valida que se sube una imagen, el formato y el tamaño. Si todo es correcto movemos la imagen al directorio correcto y eliminamos la imagen ‘demo’ que habíamos subido del directorio.

Después renombramos la imagen.¿Por que? Por que para mostrarla más tarde necesitamos saber su nombre. Por eso las vamos renombrando ‘1.jpg,2.jpg,3.jpg...’.



Durante el proceso siempre podemos eliminar una imagen ya subida como una pequeña descripción que se puede subir junto a la imagen. Si se pincha sobre una imagen se añade un formulario creado con JQUERY que permite modificar el texto de la descripción de la imagen o eliminarla completamente.

Por último ingresamos en la BBDD la ruta de la imagen para poder mostrarla después. Para ello llamamos al método ‘ingresarImagenes’ que crea un objeto Post con la ruta de la imagen y el texto que la describe. Una vez tenemos la instancia del objeto Post llamamos al método de la clase Post ‘insertarFotos’.

Para saber la ruta de la imagen nos valemos de la variable de sesion ‘\$_SESSION['idImagenIngresar']’ que contiene la ruta, ejemplo:

‘../photos/jose/2/1.jpg ‘

Pero a nosotros solo nos interesa insertar en la BBDD esta parte ‘jose/2/1’. Por lo que con un método llamado ‘prepararRuta’ eliminamos lo que no nos interesa.

NEGOCIO DEL SISTEMA

Vamos a explicar las distintas opciones que tiene el sistema.

BUSCADOR

He implementado un pequeño buscador que tiene varias opciones. Este se añade a index.php desde el archivo buscador.js. Es un formulario creado en JSON. Este te ofrece varias opciones de búsqueda, cosas que tú buscas o cosas que tú ofreces para comparar con la de otros usuarios. Luego puedes elegir la provincia, un precio aproximado y el tiempo por lo que quieres hacer el cambio.

Una vez elegido las opciones, al ir introduciendo letras en el buscador se hace una llamada a un método anónimo del archivo buscador.js que hace una llamada al archivo busquedas.php que es donde se ejecuta la sentencia SQL y devuelve los resultados.

Por ejemplo empezamos a buscar por cosas ofrecidas sin marcar ninguna opción y escribimos una ‘c’, la petición JSON sería:

opcion=BUSCADOR&BUSCAR=c&tabla=ofrezco&buscarPorProvincia=1&buscarPorPrecio=0&buscarPorTiempoCambio=0

El resultado sería:

Proyecto	
JSON	Datos sin procesar Cabeceras
Guardar	Copiar Contraer todo Expandir todo
▼ 0:	
0:	"cartucho tinta"
palabras:	"cartucho tinta"
▼ 1:	
0:	"cuchillo"
palabras:	"cuchillo"
▼ 2:	
0:	"coche familiar"
palabras:	"coche familiar"

Como se puede ver hace una consulta a la tabla 'busquedas_pbs_ofrecidas' que son las 4 frases que se nos pide cuando subimos un Post que describimos por lo que estaríamos interesados en cambiarlos.

Cuando pinchamos sobre una de las frases nos devuelve un conjunto de Posts que contienen esas frases. Ejemplo de petición al pinchar sobre **cartucho de tinta** .

opcion=ENCONTRADO&ENCONTRAR=cartucho tinta&tabla=ofrezco&inicio=0

La salida para esta consulta sería un unico Post:

```
▼ 0:
  ▼ totalRows:
    0: "1"
    totalRows: "1"
▼ 1:
  0: "31"
  1: "lola"
  2: "37"
  3: "Cantabria"
  4: "08-12-2021"
  5: "ggggggg"
  6: "lola"
  7: "1/1"
  8: "gtggg\r\nggggg\r\nggggggg"
  9: "Por un par de dias"
  10: "0"
  idPost: "31"
  nick: "lola"
  idUsu: "37"
  provincia: "Cantabria"
  fecha: "08-12-2021"
  titulo: "ggggggg"
  nickUsuario: "lola"
  ruta: "1/1"
  comentario: "gtggg\r\nggggg\r\nggggggg"
  tiempoCambio: "Por un par de dias"
```

Veamos ahora si se selecciona alguna opción del buscador por ejemplo la ciudad o el tiempo de cambio. Aquí lo interesante es como se monta la SQL. La muestro en bruto:

```

$sqlBuscador= "select distinct $columnaId
from post p
inner join usuario u on u.idUsuario = p.idUsuarioPost
inner join direccion dire on dire.idDireccion = u.idUsuario".
" inner join tiempo_cambio tmc on tmc.idTiempoCambio = p.tiempo_cambio_idTiempoCambio ".
"(isset($buscarPorTiempoCambio) ? " and tmc.tiempo = '$buscarPorTiempoCambio' : "").
" inner join ".$tabla." pbs on $columnaId = p.idPost ".
(isset($buscarPorProvincia) ? " and dire.provincia = '$buscarPorProvincia' : "").
" and $columnaPalabra like :buscar ".(isset($buscarPorPrecio) ? $pvp : ""). " order by $columnaId DESC limit 5;";
}
//echo $sqlBuscador;

```

Para no tener varias SQL según la opción estas se escriben utilizando el operador ternario de PHP. Por ejemplo si añadimos a la búsqueda la provincia la SQL queda así:

*opcion=BUSCADOR&BUSCAR=c&tabla=ofrezco&buscarPorProvincia=Barcelona
&buscarPorPrecio=0&buscarPorTiempoCambio=0*

```

select distinct idPost_ofrecidas
from post p
inner join usuario u on u.idUsuario = p.idUsuarioPost
inner join direccion dire on dire.idDireccion = u.idUsuario inner join tiempo_cambio
tmc on tmc.idTiempoCambio = p.tiempo_cambio_idTiempoCambio inner join
busquedas_pbs_ofrecidas pbs on idPost_ofrecidas = p.idPost and dire.provincia =
'Barcelona' and palabrasOfrecidas like :buscar order by idPost_ofrecidas DESC
limit 5;

```

Ahora vamos a añadir tiempo de cambio una semana:

*opcion=BUSCADOR&BUSCAR=c&tabla=ofrezco&buscarPorProvincia=Barcelona
&buscarPorPrecio=0&buscarPorTiempoCambio=Por una semana*

La SQL se le añade :

```

select distinct idPost_ofrecidas
from post p
inner join usuario u on u.idUsuario = p.idUsuarioPost
inner join direccion dire on dire.idDireccion = u.idUsuario inner join
tiempo_cambio tmc on tmc.idTiempoCambio = p.tiempo_cambio_idTiempoCambio
and tmc.tiempo = 'Por una semana' inner join busquedas_pbs_ofrecidas pbs on
idPost_ofrecidas = p.idPost and dire.provincia = 'Barcelona' and palabrasOfrecidas
like :buscar order by idPost_ofrecidas DESC limit 5;

```


Como se puede ver tenemos una SQL básica y la hacemos dinámica.

El resultado es mostrar todos los Post que cuando se han subido se han puesto en las palabras ofrecidas las palabras que coinciden en la búsquedas. Ocurre lo mismo con las palabras que uno escribe en el formulario cuando te piden por lo que estarías interesado en cambiar.

MENÚ USUARIO

El menú de usuario es otro formulario que se añade a 'index.php' desde el archivo 'mostrarMenuUsuario.js' con JQUERY. Tienes 3 opciones implementadas, darse de baja, cambiar datos y bloquear usuarios.

Darse de baja

Esta opción te permite darte de baja definitivamente o bien parcial. La diferencia radica en que si lo haces totalmente se eliminara tú cuenta y Posts del sistema. Si lo haces parcialmente no puedes loguearte ni recibirás correos del portal.

Estas opciones son fáciles de implementar. La primera desde el menú se llama al método 'darseBajaDefinitivamente' del archivo 'darBajaUsuario.js' que hace una petición JSON al archivo 'darBajaUsuario.php' con la opción **definitivamente**. Básicamente hace un **delete** de la tabla usuario y **en cascada** se ven afectada el resto de tablas. Finalmente se llama al método de la clase 'Directorios' :

```
Directorios::eliminarDirectoriosSistema("../..../photos/"$nickEliTotal,"eliminarDirectoriosBajaUsuario");
```

Este método recibe la ruta con todos los directorios del usuario y los elimina del sistema.

Finalmente se le manda un correo para informarle que la baja se ha cursado con éxito.

La segunda opción utiliza los mismos archivos **se hace un update** al campo de la tabla usuario 'activo' a un 0. Cuando un usuario se registra se inserta un 1. Cuando un usuario intenta loguearse este campo es consultado. Si se intenta acceder se le informa:

Introduzca sus datos

Formulario de ingreso

Introduce nombre de usuario:

carlos

Introduce tú password

Tú cuenta no esta activa.
Ponte en contacto con un administrador

aceptar

salir

Actualizar Datos

Esta opción sirve para actualizar los datos de un usuario. Como se ha explicado más arriba no voy a entrar en detalles.

Bloquear Usuarios

Esta opción permite bloquear a un usuario de dos maneras distintas.

TOTALMENTE

Esto quiere decir que cuando un usuario bloquea a otro este último no verá sus Posts. Aunque para la demostración del funcionamiento del sistema se ha cambiado por mostrar un aviso. Veamos un ejemplo:

El usuario Carlos bloquea a Pedro totalmente. Cuando Pedro entre en el Portal y salga un Post de Carlos este verá:

ESTO PINTA MAL PARA TI !!!

En realidad como se ha comentado simplemente al mostrar los Post lo que se hace **es consultar si la persona que ha colgado el Post tiene bloqueada a la persona que esta logueada en ese momento. Si es así en el bucle que muestra los Post se hace a ese en concreto un continue.** Pero para verse más claro el funcionamiento he añadido esto.

Todo esto se realiza en el archivo 'json.php' que es el encargado de mostrar los Posts. Este archivo recibe una petición desde el archivo 'principal.js' como su nombre indica es el encargado de iniciar todo el sistema. No voy a entrar en explicar la

paginación en este punto, lo explicaremos más tarde. Vamos a ver como vemos que usuarios están bloqueados.

Primero pedimos los Posts, una cantidad en concreto, supongamos 5. Cuando tenemos los Post **en sus datos incluye el id del usuario que lo subió**. Como tenemos el id del usuario logueado en cada Post comparamos que el id del usuario logueado no está en la tabla de usuarios bloqueados por el usuario que subió el Post. Si es así se añade una bandera al array con los datos del Post.

Pongo la consulta:

```
if (isset($_SESSION['userTMP'])) {  
    $usuBloqueados = $usuBloqueo->devuelveUsuariosBloqueados($tmp[2]);  
  
    //var_dump($usuBloqueados);  
    $totalUsuarioBloqueado = count($usuBloqueados);  
    // Si el usuario que ha colgado el Post ha bloqueado  
    // algun usuario se verifica que no sea el que esta logueado  
    //Se le impide ver este Post  
  
    if ($totalUsuarioBloqueado > 0) {  
        for ($i = 0; $i < $totalUsuarioBloqueado; $i++) {  
            if (($usuLogeado == $usuBloqueados[$i][0]) and ($usuBloqueados[$i]['bloqueadoTotal'] == 1)) {  
                $tmp['coment'] = 2;  
            } else if (($usuLogeado == $usuBloqueados[$i][0]) and ($usuBloqueados[$i]['bloqueadoParcial'] == 1)) {  
                //Agregamos un testigo para cuando se  
                //muestre en JAVASCRIPT el POST  
                //Se inavilite el boton de comentar  
                $tmp['coment'] = 1;  
            }  
        }  
    }  
}
```

Como se ve se consulta los dos tipos de bloqueos el total y el parcial. Veamos la respuesta JSON a un usuario llamado Pedro que ha sido bloqueado por Carlos:

```
totalRows: "4"  
1:  
  0: "30"  
  1: "carlos"  
  2: "35"  
  3: "No importa"  
  4: "08-12-2021"  
  5: "gfdjhjdj"  
  6: "carlos"  
  7: "5/demo"  
  8: "fffgfdfgfgfg\r\nfgfg\r\nff"  
  9: "Por una temporada"  
 10: "0"  
idPost: "30"  
nick: "carlos"  
idUsu: "35"  
provincia: "No importa"  
fecha: "08-12-2021"  
titulo: "gfdjhjdj"  
nickUsuario: "carlos"  
ruta: "5/demo"  
comentario: "fffgfdfgfgfg\r\nfgfg\r\nff"  
tiempoCambio: "Por una temporada"  
coment: 2  
2:
```

Si nos fijamos en el último campo aparece **coment : 2**. Veamos cuando se muestran los Post en el archivo 'mostrarPosts.js' lo que ocurre:

```
//Si el usuario ha sido bloqueado parcialmente
//eliminamos el boton de comentar con JAVASCRIPT
if(objPost[i].coment == 1){
    $("."+objPost[i].idPost).hide();//.attr('disabled',true);
};

if(objPost[i].coment == 2){
    $("#"+objPost[i].idPost).empty();
    $("#"+objPost[i].idPost).prepend($('

',{
        class : 'cont_post',
    })).append($('

# ',{ text : "Esto pinta mal para ti !!!" }))); };


```

Esta claro que esto **esta en un bucle y si en el array JSON en el campo coment hay un 2 en lugar de mostrar el Post muestra la captura de antes. Aunque como digo en verdad ahí va un continue que lo que hace es no mostrar ese Post.**

PARCIALMENTE

El proceso es el mismo y utiliza los mismos archivos. La diferencia es que si te bloquean parcialmente puedes ver los anuncios pero no puedes añadir comentarios. Esto lo conseguimos eliminando el botón de comentar del anuncio como se ve en la captura de arriba. Pongo captura de usuario que no esta bloqueado parcialmente y otro que si lo esta.

Sin bloquear:

Tiempo que durara el cambio: Para siempre

[Comentar](#)

Total Comentarios : 1

Bloqueado parcial:

Tiempo que durara el cambio: Por un par de dias

3

HERRAMIENTAS EXTERNAS

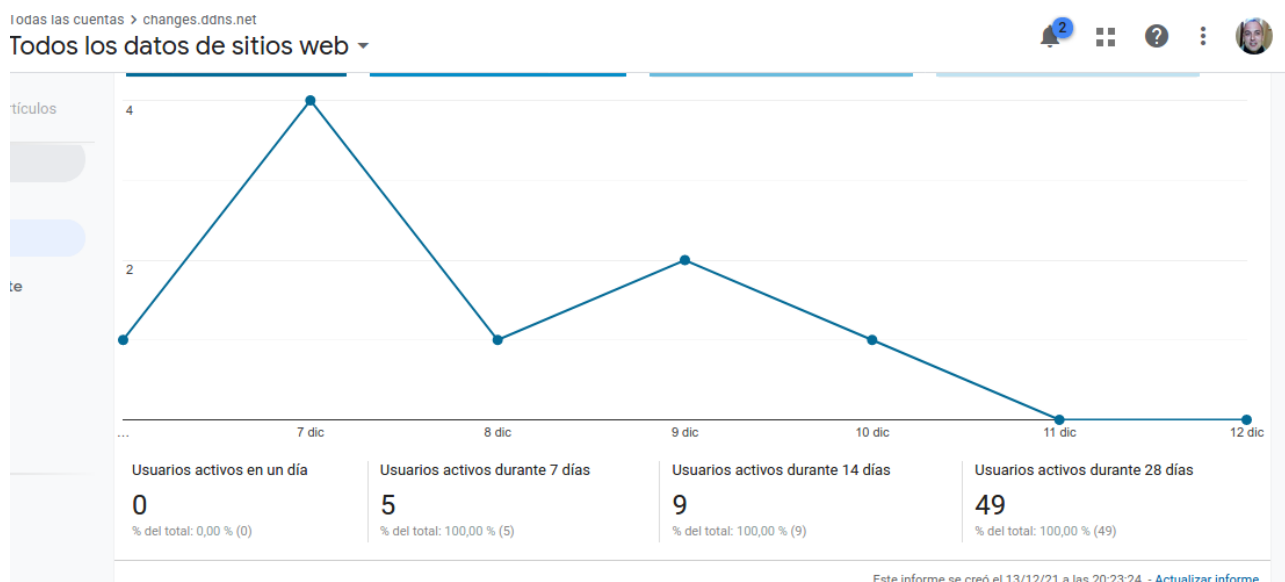
Voy a añadir dos herramientas que he usado para el proyecto que me parecen interesantes. La primera es **GIT**, un sistema de control de versiones diseñado por **Linus Torvalds**. Pongo una captura de mi cuenta en **GITHUB**.

The screenshot shows a GitHub repository page for 'carlosneiraweb/pretutoria'. The repository has 21 commits and was last updated 5 hours ago. The file list includes folders like 'Controlador', 'DescripcionPortal', 'Modelo', 'Sistema', 'Vista', 'css', 'datos_usuario', 'fonts', 'img', 'mysql', 'nbproject', and 'photos'. The right sidebar shows repository statistics: 0 stars, 1 watching, 0 forks, and no releases or packages published.

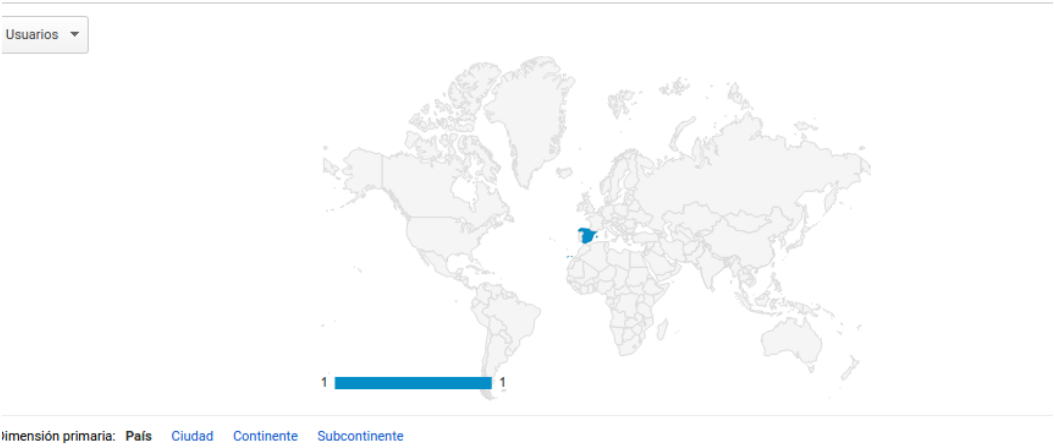
File/Folder	Commit Message	Time Ago
Controlador	pretutoria	5 hours ago
DescripcionPortal	pretutoria	5 hours ago
Modelo	BUSCADOR	5 days ago
Sistema	pretutoria	5 hours ago
Vista	pretutoria	5 hours ago
css	EXCEPCIONES ACTUALIZAR_3	6 days ago
datos_usuario	Ajustes	5 days ago
fonts	ANTES CSS	2 months ago
img	ANTES CSS	2 months ago
mysql	BUSCADOR	5 days ago
nbproject	ANTES CSS	2 months ago
photos	pretutoria	5 hours ago

La segunda herramienta que he usado es **GOOGLE ANALYTICS**. Con esta herramienta que nos ofrece **GOOGLE** podemos hacer un seguimiento de nuestra página en muchos sentidos. Por ejemplo desde que país nos ven, que edad o sexo tienen nuestros usuarios, que sistema operativo o tipo de dispositivo usan, que navegador, tiempo medio de visita, nuevos usuarios, etc.

Pongo un par de capturas:



País donde nos ven:



Datos Generales:

