APLICACIONES WEB

Memoria del Proyecto

Práctica 2 - Arquitectura y prototipo del proyecto del grupo UniTools

Miembros del grupo :
Luis Cepeda, Carlos Bilbao, Hugo Ribeiro, Daniel Canseco, Fernando Ruiz, Bruno Mayo

Índice general

1. Descripción general de la práctica

En esta memoria, describiremos cómo hemos decidido estructurar nuestra página web llegado este punto de su desarollo. Se trata de una práctica importante para nosotros, ya que su desarollo y las decisiones que tomáramos iban a condiciar el resto de las prácticas. Es decir, la estructura general, así como la Base de Datos y las abstraciones que empleamos van a servirnos como base en sucesivas versiones.

Durante esta práctica hemos centrado nuestros esfuerzos en dos aspectos:

1. Organizar nuestra página web haciendo uso de **un esquema que divide en cuatro partes (cabecera, navegación ó menú, contenido y pie)**, siendo el contenido el único actualizado dinámicamente, en función de las peticiones recbidas.

   A continuación, hemos dividido entre scripts de vista, de apoyo y de lógica. Esta distribución ha resultado ser muy provechosa y nos ha permitido avanzar más agilmente. Además, hemos ahorrado muchas líneas de código gracias a ella.

2. Desarrollar tres funcionalidades principales. Para comenzar a darle forma a nuestra web, hemos elegido por un lado funcionalidades esenciales: Es decir, sin ellas lo demás no sería posible, este es el caso de **Login y Registro**.

   Al mismo tiempo, hemos desarrollado otras dos funcionalidades que, sin depender otras de ellas, planteaban también un reto técnico. Estas han sido Herramientas **y más importante, Foro**.

Todo esto se ha hecho utilizando el patrón DAO (dao/), mediate clases.

Hemos seguido la estructura de carpetas explicada en la teoría de la asignatura. Hemos creado un usuario extra para la Base de Datos (aparte del propio root, con total acceso) con diferentes permisos y privilegios, así como una tabla extra en la Base de Datos, como log de los accesos fallidos en el Login. La contraseña se ha encriptado con Bcrypt y un hash para mayor seguridad.

A la hora de dividir el trabajo, decidimos dejar a cada miembro del equipo encargado de dirigir cada una de las partes. Dicho lo cual, hemos hecho un esfuerzo por realizar videollamadas semanalmente para coordinar el trabajo y asegurarnos de que todos estábamos al corriente del funcionamiento de lo demás. Hemos utilizado GitHub para gestionar las versiones y commits del proyecto. **El código mostrado en esta memoria, podría no estar completamente actualizado debido a pequeños cambios para pulir detalles, no obstante la idea general explicada será la misma en cualquier caso**. Nótese que hay un archivo .htaccess oculto.

2. Scripts para las vistas

Las vistas posibles de esta versión son entre las que permite navegar el Menú de la página web, aunque algunas de ellas no mostrarán ningún contenido si el usuario no ha sido previamente registrado.

- (scripts/perfil.php) Perfil incluye ya toda la información del usuario, incluido si se trata de un usuario Premium.
- (scripts/proyectos.php) Proyectos permitirá a los estudiantes subir código de sus proyectos.
- (scripts/foro.php) **Foro permite a los usuarios postear y ver mensajes**.
- (scripts/mensajes.php) Mensajes requerirá estar loggeado.
- (scripts/herramientas.php) **Herramientas incluye algunas utilidades para los estudiantes** a modo de funcionalidades. En esta versión hemos añadido varias, que serán explicadas luego.
- (scripts/login.php) Login permite iniciar sesión al usuario, tal y como se explicará más tarde.
- (scripts/registro.php) Registrar permite crear un nuevo usuario, mediante una conexión a la Base de Datos del DAO, como se explicará a continuación.

En realidad, en nuestro portal sólo se accede a una página, que es index.php, cuyo código se muestra a continuación. Se ha incluido un caso condicional para hacer *session_start()*, para que en el caso que el usuario se acabe de conectar ó su sesión haya caducado, se puedan restaurar las variables globales.

Después, utilizando *require()* se incluyen las partes que toda vista debe tener: cabecera (estructura/cabecera.php), navegación (estructura/menu.php), un contenido (del que ahora se hablará) y un pie de página (estructura/pie.php).

```
<?php
    if(!isset($_SESSION))
    {
        session_start();
    }
?>
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="../css/hoja.css">
    <title>INDEX</title>
    <meta charset="UTF-8">
```

```
</head>
<body>
<div id="contenedor">
    <?php
        require("cabecera.php") ;
    require("navegacion.php") ;
  ?>
  <div id="contenido">
     <?php require("contenido.php");?>
  </div>
  <?php
    require("pie.php") ;
  ?>
</div> <!-- Fin del contenedor -->
</body>
```

Por tanto, lo único que cambiará según cual sea la vista es el contenido, administrado en estructura/contenido.php, el cual no solo se encarga de cambiar el contenido en función de la petición que le llegue, sino que además realiza el control de seguridad para comprobar que el usuario ha sido correctamente loggeado, y en caso contrario no mostrarle dicho contenido.

Para ilustrar esto, asumamos que alguien que no está ni siquiera loggeado intenta acceder a un contenido sólo disponible para administradores, manipulando la petición en la url para acceder a admin.php (Es decir, escribe *index.php?page=admin*). En ese caso, tal y como se muestra en el siguiente código, contenido.php no le dejará acceder tras comprobar con los parámetros de la variable global SESSION que, efectivamente, ni era administrador ni había iniciado sesión.

```
<?php

if(isset($_GET["page"])) {
    (...)

  else if($_GET["page"] == "admin") {

    if ((!isset($_SESSION["login"]))&&(!isset($_SESSION["esAdmin"]))) {
            echo "<p>No puedes ver este contenido, tienes que estar loggeado y ser Administrador para visualizarlo.</p>";
    }
```

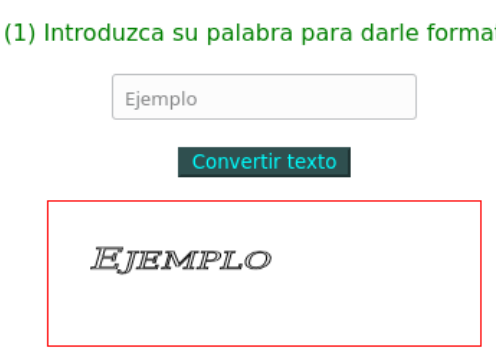
Otra funcionalidad que hemos implementado es **Herramientas**, en *script/herramientas.php*. Se trata de una serie de utilidades sencillas que quisimos añadir para prácticar nuestra programación en JavaScript y al mismo tiempo proveer de alguna herramienta que puedan encontrar nuestros compañeros interesante.

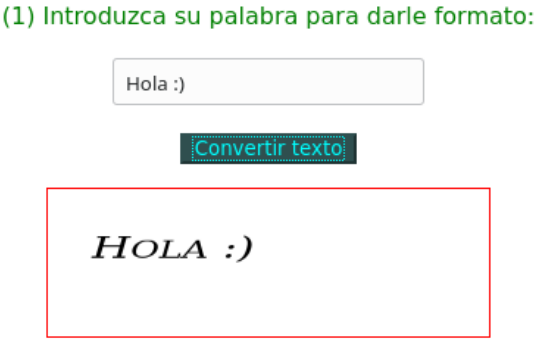
Por ahora, se han implementado cuatro herramientas:

1. Un formateador de palabras.
2. Un contador de palabras.
3. Un conversor de decimal a hexadecimal.
4. Un conversor de decimal a binario.
5. Un conversor de binario a decimal.

Todos salvo el primero son sencillas funciones de JavaScript cuyo desarrollo no merece la pena explicar. Lo único reseñable es que las tres últimas se han implementado, en realidad, con la misma función pero distintos parámetros.

La herramienta más interesante es la primera, ya que para ella se ha hecho uso de **Canvas**, que permite dibujar gráficas y similares en JavaScript como se muestra.






```
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.font = "italic small-caps 30px Verdana";
ctx.strokeText("Ejemplo",30,70);
function texto(){
        $str = document.getElementById('valor_d4').value;
        ctx.clearRect(0,0,c.width,c.height).fillText($str, 30, 70);
        alert("Palabra cambiada!");
}
```

La otra funcionalidad y la principal de esta práctica es el Foro, gestionado mediante el patrón de diseño DAO. Una vez un usuario se ha registrado e iniciado sesión, puede **tanto revisar la lista de Posts publicados como escribir un nuevo Posts**, con scripts/post.php.

Para mostrar los posts del foro, basta con usar los DAO, que realizan unas queries a la Base de Datos que ordenan las entradas por antigüedad, cosa que se puede hacer sin problema gracias a los id, que ascienden automáticamente. Una vez se tienen los Transfer Object de las entradas, estas pueden asociarse al TO del usuario, para luego crear una tabla con su información (id y Nombre de Usuario).

```php
<?php
        include_once("dao/dao_user.php");
        include_once("dao/dao_post.php");

        $foro_data = new TOUpost();
        $dao_post = new DAOpost();
        $dao_user = new DAOUsuario();
        $res = $dao_post->show_all_data();

        while(!empty($res)){
            $curr_post = array_shift($res);
            $user_id = $curr_post->get_user(); // id del usuario
            $usuario = $dao_user->search_userId($user_id);
            (...)
            echo "<table class=\"posts\">";
            echo "<tbody>";
            echo "<tr>";
            echo "<td>ID del post:".$curr_post->get_id()."</td>";
            (...)
            echo "</table>";
        }
        $dao_user->disconnect();
        $dao_post->disconnect();
    ?>
```

Publicar entradas en el foro es un poco más intrincado, ya que requiere coger el texto escrito por el usuario y realizar una inserción en la Base de Datos. El DAO hace:

```php
$sql = sprintf("INSERT INTO posts(user,title,content,category)
        VALUES ('$user', '$title', '$content', '$category')");
$result = $this->ejecutarConsulta($sql);
```

La conexión a la Base de Datos tendrá que hacerse con los permisos necesarios para insertar, por lo que hemos decidido dar al usuario no root permisos de inserción. De todo ello se hablará más detenidamente en el Capítulo dedicado a la BBDD. El "padre"de los DAO es el encargado de gestionar la conexión.

```php
<?php

class DAO {

 public $conn;

   public function __construct() {

  if (!$this->conn){

    $this->conn = mysqli_connect("localhost", "root", "", "
   unitoolsdb");

    if( mysqli_connect_error ()){
       die ("Conexion con la base de datos fallida : " .
   mysqli_connect_error());
    }
    (...)
?>
```

Y luego las clases DAO heredan de éste:

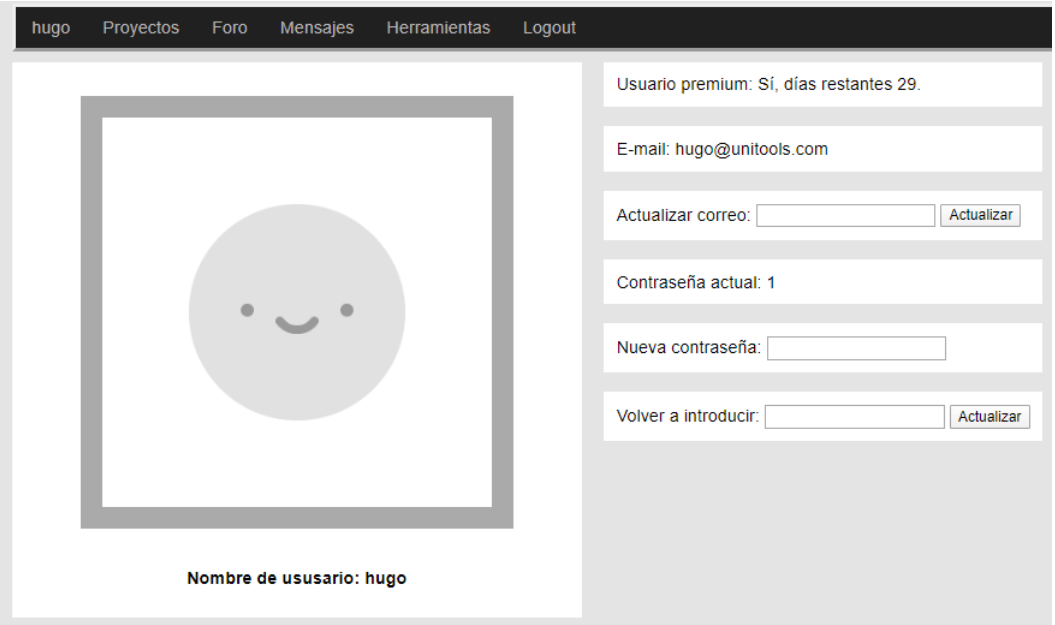
```php
<?php

include_once('DAO.php');
include_once('user_class.php');

/* Data Access Object */
class DAOUsuario extends DAO {

  public function __construct(){
    parent::__construct();
  }

  public function insert_User($TOUser){
    $mail = $TOUser->get_email();
        $pass = $TOUser->get_password();
        $username = $TOUser->get_username();
        $premium =  $TOUser->get_premium();
```

```
18      $sql = "INSERT INTO user SET email='$mail' , password='$pass',
        username='$username', premium='$premium'";
19
20      if (!$this->insertarConsulta($sql))
21        return false;
22      else
23      {
24        return true;
25      }
26    }
```

Y utilizan los Transfer Object como "moneda de cambio"de la información:

```
1  <?php
2
3  /* Transfer Object */
4  class TOUser {
5
6    private $userId;
7    private $email;
8    private $password;
9    private $user_name;
10   private $premium;
11
12   function __construct($userId='',$email='',$password='',$user_name
      ='',$premium=''){
13     $this->userId = $userId;
14     $this->email = $email;
15     $this->password = $password;
16     $this->user_name = $user_name;
17     $this->premium = $premium;
18   }
19   (...)
20
21   public function get_user(){
22     // Devuelve un array con todos los datos de usuario
23     $columna = [
24         "email"     => $this->email,
25         "password"  => $this->password,
26         "user_name" => $this->user_name,
27         "premium"   => $this->premium
28     ];
29     return $columna;
30   }
```

Además, ya hemos montado la versión básica del Perfil, tal y como se muestra a continuación. Esta incluye información como los días que quedan de Premium, así como la contraseña, y la posibilidad de actualziar correo y/o contraseña.



Un archivo no accesible por las vistas (ya que en sus propias reglas se define como protegido) es includes/.htaccess, que hemos generado teniendo en cuenta:

- Redirección de errores a archivos de /scripts/err
- Bloqueo de bots.
- Prevención de visualización de .htaccess.
- Prevención de listado de los directorios.

```
Options +FollowSymLinks
RewriteEngine on
RewriteCond %{HTTP_HOST} ^index.php[nc]
RewriteRule ^(.*)$ http://www.index.php/$1 [r=301,nc]
ErrorDocument 400 /scripts/err/400.php
RewriteEngine On
RewriteCond %{HTTP_USER_AGENT} ^BlackWidow [OR]
<Files .htaccess>
order allow,deny
deny from all
(...)
```

3. Scripts adicionales

Dos scripts que es importante mencionar son el de Login y el de Registro, ambos estrechamente ligados con la Base de Datos que será después descrita.

El contenido de scripts/registrar.php es simplemente un form sencillo, cuya action es procesarRegistro.php, en la misma carpeta. A continuación se describirán algunas de las partes más importantes de este script.

Primero, se crean variables que serán utilizadas para la conexión con la Base de Datos por parte de los DAO. Estas son los campos de la tabla usuario, que fueron previamente rellenados en registro.php. Por t, se utilizan las funciones *trim* y *strip_tags* para hacer "sanitize"del input. Esto incluye eliminar espacios, e **impedir que el usuario introduzca código malicioso en las entradas**. Por otro lado, se utilizan funciones extra para comprobar la validez de los campos introducidos, que de fallar darán lugar a mensajes de error en rojo. Como se ve en la línea 19, las contraseñas se almacenan en la BBDD cifradas con un hash.

```
1  include_once('../dao/dao_user.php');
2  $id_user = "";
3  $username =  htmlspecialchars(trim(strip_tags($_REQUEST["username"
       ])));
4  $email = htmlspecialchars(trim(strip_tags($_REQUEST["email"])));
5  $password = $_REQUEST["password"];
6  $password2 = $_REQUEST["password2"];
7  $premium = 0;
8
9  if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
10     $_SESSION['error_registro'][] = "El email introducido no es
       valido";
11 }
12 (...)
13 $dao_usuario = new DAOUsuario();
14
15 if ($dao_usuario->search_username($username)) {
16         $_SESSION['error_registro'][]="Usuario insertado existe";
17 }
18
19 $encrypted = password_hash($password,PASSWORD_BCRYPT);
20 $user = new TOUser($id_user,$email,$encrypted,$username, premium);
21 (...)
```

Acabado esto, toca conectarse a la Base de Datos, utilizando eL DAO. En este script se intentará conectar a la Base de Datos, utilizando los valores proporcionados por

el TO del Post. Esta función se encuentra en /dao/dao_post.php.

```
public function insert_Post($TOUpost){
    $user = $TOUpost->get_user();
    $title = $TOUpost->get_title();
    $content = $TOUpost->get_content();
    $category =  $TOUpost->category();
    $sql = sprintf("INSERT INTO posts(user,title,content,category)
        VALUES ('$user', '$title', '$content', '$category')");
    $result = $this->ejecutarConsulta($sql);

    if (count($result) > 0) {
      $post = new TOUpost($result['id_post'], $result['user'],
   $result['title'], $result['content'], $result['id_cat']);
      return $post;
    }
    return null;
  }
```

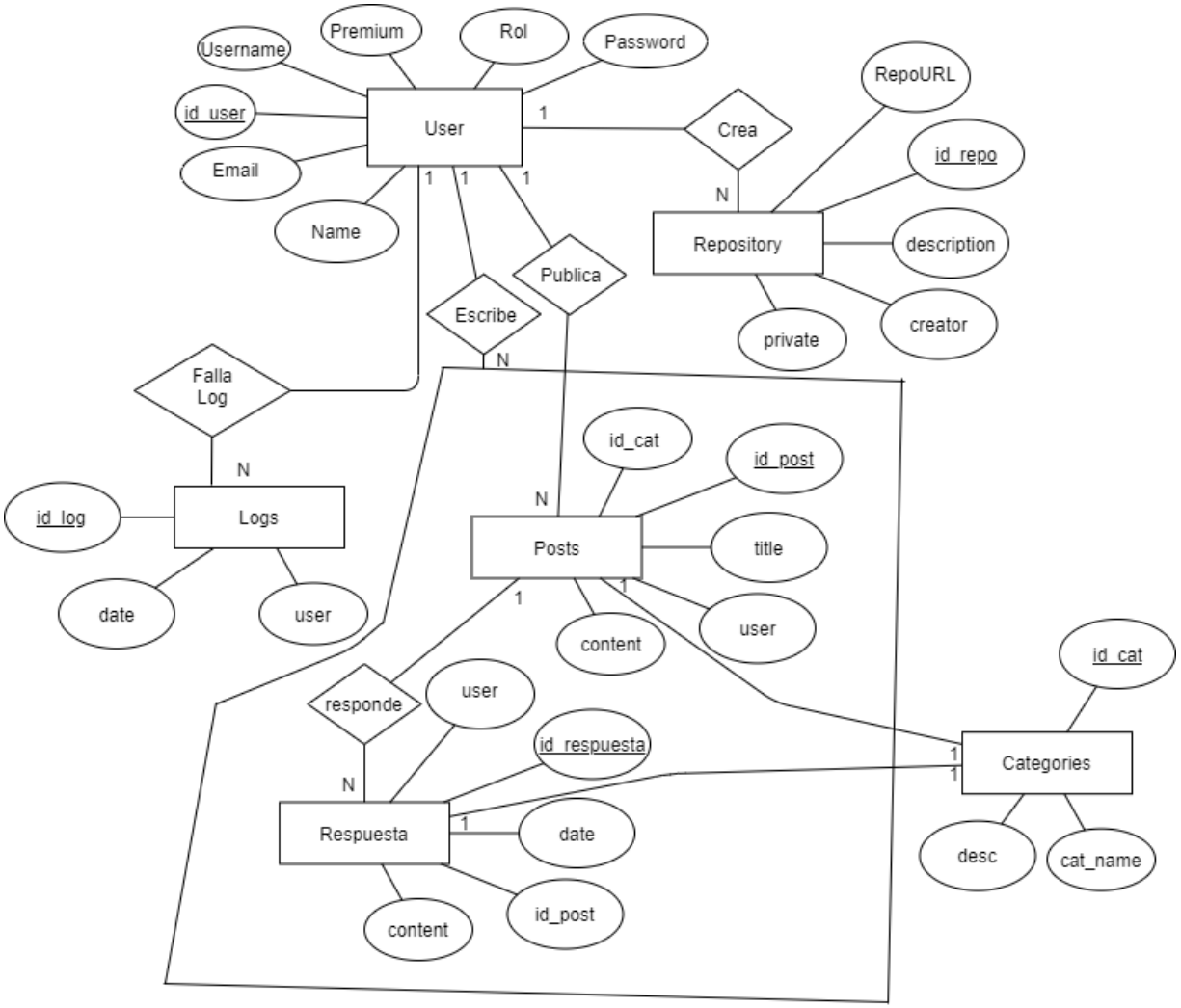
Como se puede observar, la petición SQL se trata de una sencilla inserción en la tabla user, utilizando todas las variables que fueron previamente revisadas.

El script de procesarLogin.php es muy similar. Se reciben los valores que han sido introducidos en el form de login.php y estos se utilzan para realizar la query en el DAO, compartidos por el TOU. Eso si, como era de esperar en esta ocasión la petición SQL no es una inserción, sino un SELECT que comprueba que los valores son idénticos.

```
public function search_username($username){
    $sql = sprintf("SELECT * FROM user WHERE username = '" .
   $username. "'");
    if (!$this->ejecutarConsulta($sql))
      return null;
    else
    {
      $result = $this->ejecutarConsulta($sql);
      $user = new TOUser($result['id_User'],$result['email'],
   $result['password'],$result['username'],$result['premium']);
      return $user;
    }
  }
```

4. Estructura de la Base de Datos

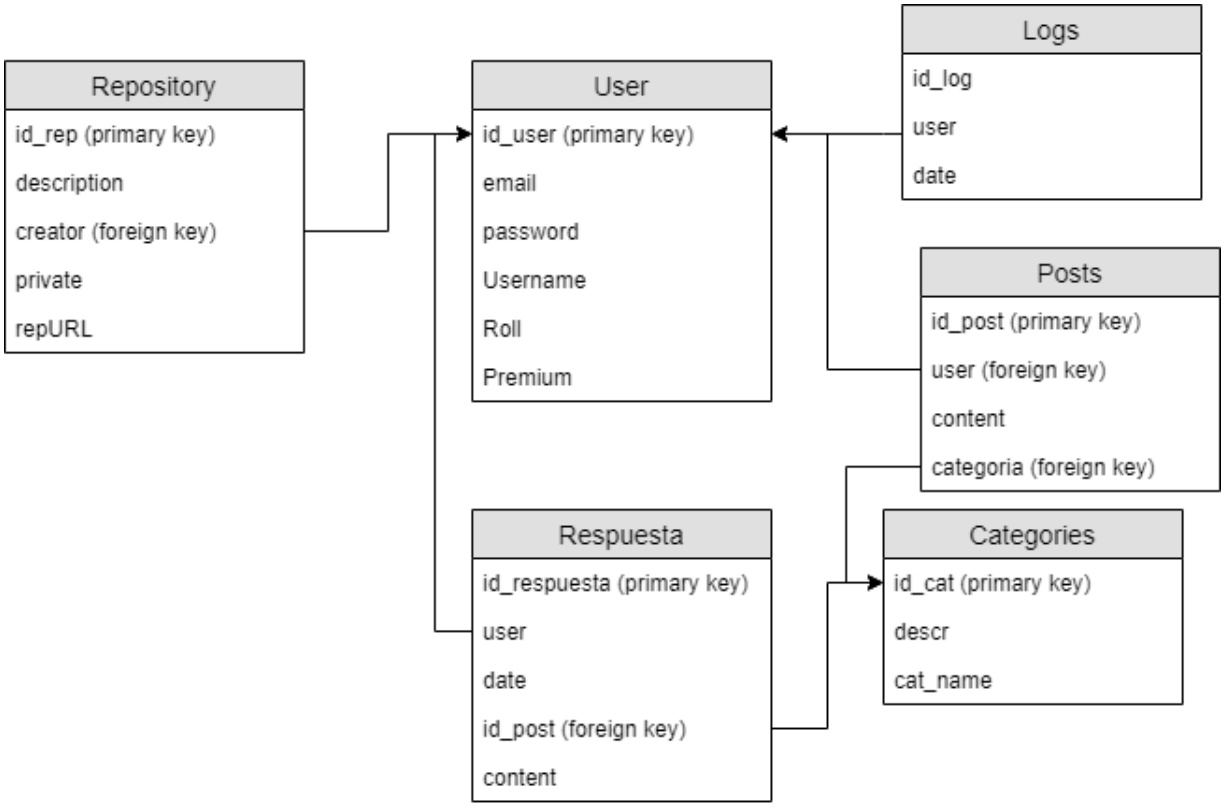
A continuación repasaremos la estructura que hemos seguido para nuestra Base de Datos inicial, apoyándonos en PhpMyAdmin. Para ello, hemos decidido diseñar e incluir dos módelos muy utilizados para la definición de las BBDD, el Modelo Relacional y el Modelo Entidad Relación, de más alto nivel. También hablaremos de cómo hemos gestionado los permisos.



Como se puede ver, la Base de Datos consta de seis tablas: User, Repository, Logs (de accesos fallidos), Posts (en el Foro), Respuesta (del Foro) y Categories (para el Foro). Cada una de ellas con su clave primaria y relacionada con otra (u otras, como en el caso de User).

Además, se pueden apreciar las restricciones de cantidad. Un usuario puede crear N repositorios, así como generar N logs de fallos y publicar N posts en un foro. Casi todas las relaciones son para con el Usuario y de tipo (1,N).

Desgraciadamente, **esta información se pierde al traducir el Modelo ER a un modelo relacional**, por culpa de las limitaciones inherentes al segundo. Este segundo esquema quedaría entonces como se muestra a continuación.



Estas seis tablas luego son pobladas en nuestra Base de Datos, los ids son incrementados por orden de apariencia y estos no pueden ser nulos, tal y como se puede apreciar en el siguiente código.

Además, como puede apreciarse en el código mostrado abajo, se han añadido restricciones para las foreign key y políticas CASCADE para las actualizaciones.

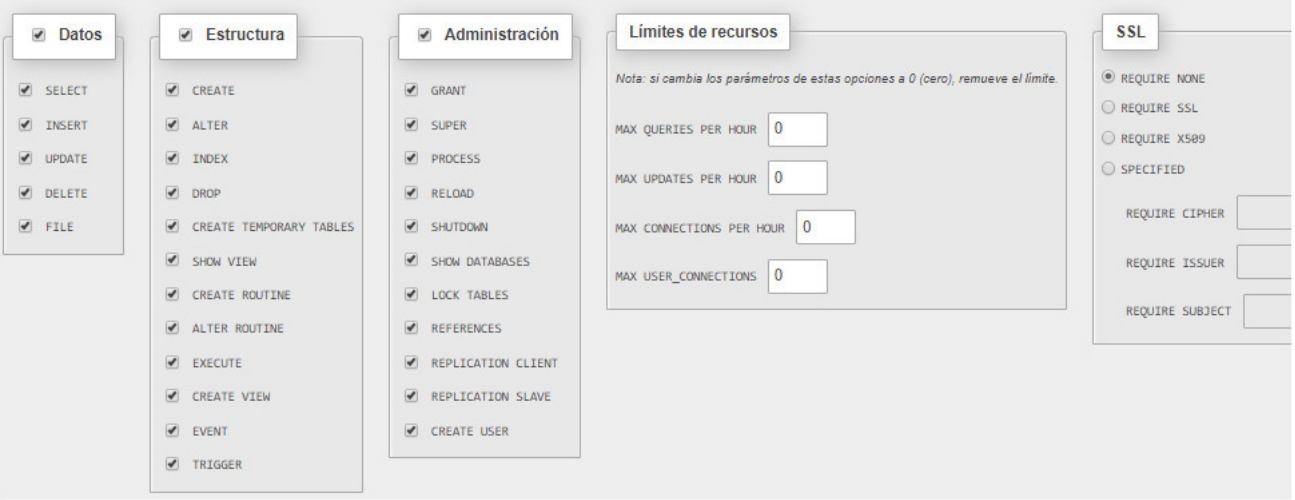
```
--
-- AUTO_INCREMENT de la tabla `repository`
--
ALTER TABLE `repository`
  MODIFY `id_rep` int(11) NOT NULL AUTO_INCREMENT;
```

```
(...)
--
-- Filtros para la tabla `repository`
--
ALTER TABLE `repository`
  ADD CONSTRAINT `repository_ibfk_1` FOREIGN KEY (`creator`)
   REFERENCES `user` (`id_User`) ON UPDATE CASCADE;
COMMIT;
```

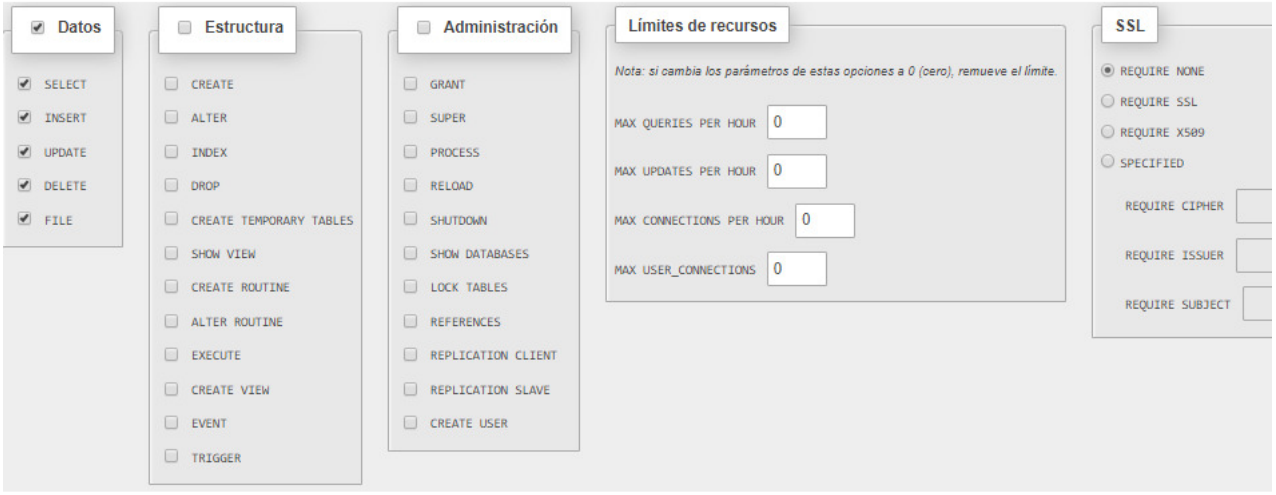
Además, hemos tenido que tener en mente los permisos para hacer una Base de Datos segura y restrictiva. Para eso, hemos decidido usar **dos cuentas de usuarios**.

Los permisos del usuario root son los que eran de esperar, totales. La siguiente imagen muestra una captura de pantalla de los parámetros utilizados en PhpMyAdmin para añadir al usuario root con total privilegio y permisos de acceso.

Como se puede ver, el usuario root tiene todos los permisos tanto desde un punto de vista de Administración , como de Estructura (crear,alterar entradas...), y Datos para realizar queries a la Base de Datos (Seleccionar, Insertar, Actualizar, Eliminar, etc).



Por otro lado, los privilegios del usuario normal (**usuario1**) son mucho más limitados, como se puede ver debajo. En este caso, **la contraseña que usamos para el usuario normal es unitoolsdb**.



La BBDD (disponible en ***DataBase/unitoolsdb.sql***) se complicará y crecerá en sucesivas prácticas. Dicho esto, consideramos que la estructura inicial, así como el sistema de privilegios por el que hemos optado, son un buen punto de partida y no esperamos grandes cambios ni re-estructuraciones completas.