CÓMO SE HIZO

Resumen de la práctica 2

Descripción breve

Describe la memoria de la práctica 2 de la asignatura de Programación Web

Emilio Chica Jiménez
emiliocj@correo.ugr.es

Indice

Introducción

La práctica se ha llevado a cabo usando la estructura de HMTL y CSS de la práctica 1 como base. Mi trabajo lo he dividido del siguiente modo:

1. Estudié la arquitectura necesaria para el proyecto y a parte del diagrama de clases incluido en este documento en el que se puede ver la estructura básica de la aplicación, he elegido una arquitectura MVC (Modelo Vista Controlador) que me ha servido para organizar el trabajo. El estudio de esta arquitectura me llevó 2 días, el primero para pensarla y el segundo para organizar el proyecto en carpetas.
2. En el día 3 empecé a crear modelos para las clases del diagrama de clases y fui confeccionando los DAO (Data Access Object) que me servirían de puente entre los modelos y la base de datos y a crear la validación de los formularios en el archivo script.js.
3. El cuarto día lo dediqué a convertir mi archivo class.BaseDatos.php que tenía orientado a mysqli a PDO pues observé que era más sencillo y reutilizable para hacer consultas seguras con prepare, ya que mysqli ponía problemas a la hora de hacer un prepare con un array de elementos.
4. El 5 y 6 día empecé con los archivos controladores que en este caso son index.php y portada.php, para los cuales empecé con una estructura básica y fui ampliándola conforme creaba secciones del menú ejemplo: bibliografía, fotos, información.
5. El 7 y 8 día continuaba con los controladores a la par que iba creando las vistas que necesitaba para cada elemento.
6. Por último, los 3 últimos días los dediqué a corregir errores y a probar que todo funcionase.
7. Las claves para cualquier usuario de la aplicación, véase: maria, pepe, jose, jonas son: Xpw_76652127!

Arquitectura de la aplicación

En la figura 1 podemos ver el diagrama de clases de la aplicación, donde he necesitado una clase **Amigo** para relacionar los usuarios entre sí, aunque en las especificaciones de la práctica no era necesario lo he creído conveniente pues así aumenta la funcionalidad de la aplicación.

Como se puede ver también he incluido una propiedad en el usuario llamada **enlinea** que, aunque no la actualizo a la hora de iniciar sesión pues no lo pedía la práctica, sí que la uso para filtrar los usuarios que están en línea en la aplicación.

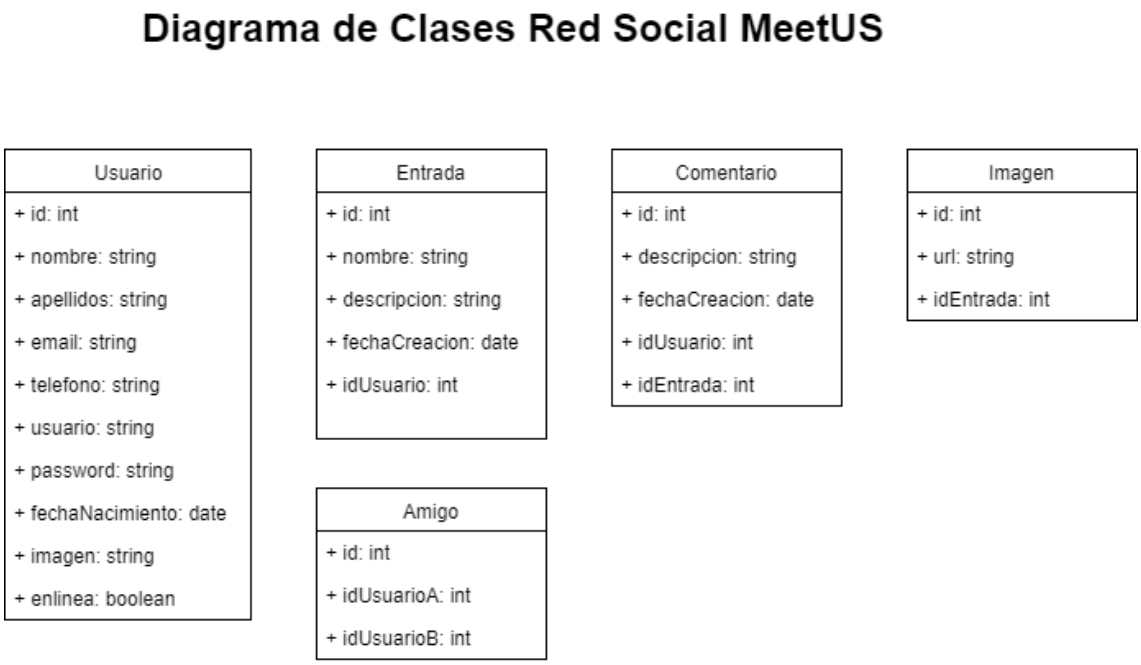

Figura 1: Diagrama de clases de la aplicación

La imagen de la figura 2 muestra la arquitectura que he seguido a la hora de hacer la aplicación, donde tengo:

- **Modelos**: Usuario, Entrada, Comentario, Imagen, BaseDatos, Sesion y Constantes
- **Vistas**: VistaUsuarios, VistaEntradas, VistaComentarios, VistaImagenes, VistaIndex, VistaActivos, VistaAmigos, VistaHeader, VistaHeadFooter, VistaMenu, VistaPerfil, VistaPublicar.
- **Controladores**: index.php, portada.php
- **DAO**: Usuarios, Entradas, Comentarios e Imágenes
- **Funcionalidades**: funciones.php

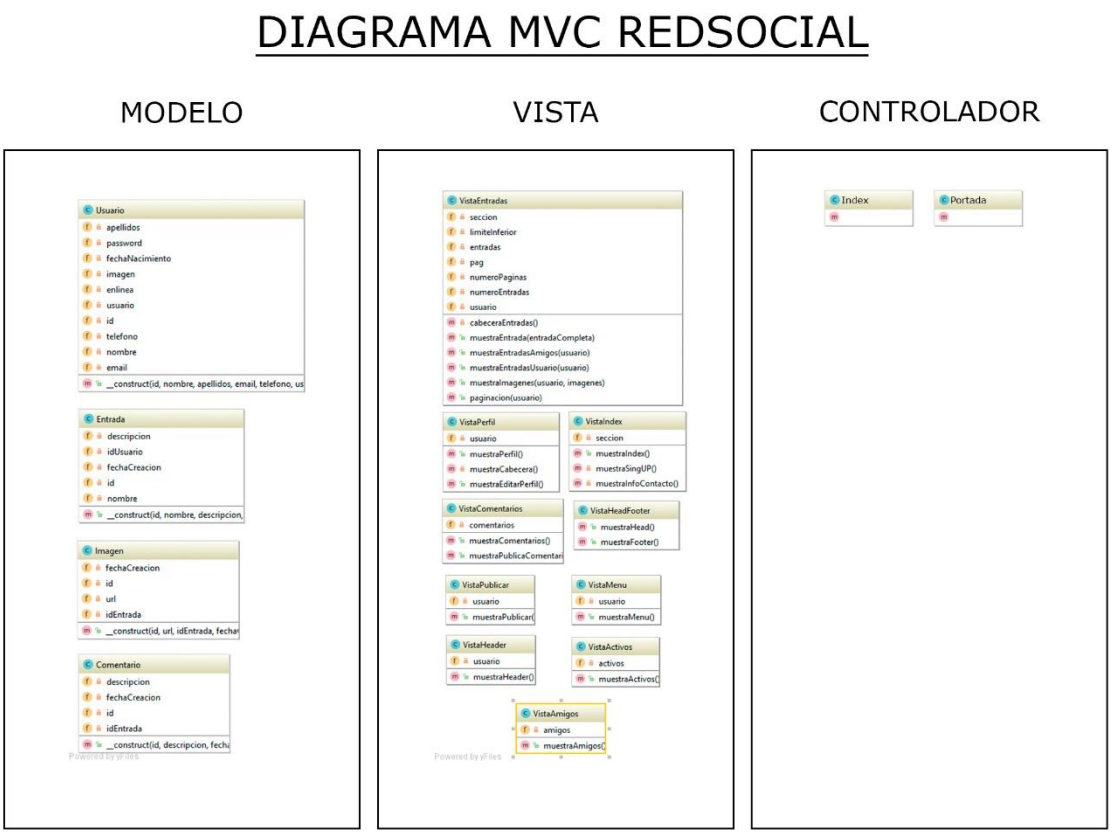

Figura 2: Diagrama MVC de la aplicación

Como en la arquitectura no eran necesarios los modelos de BaseDatos, Sesion y Constantes no los he reflejado en el diagrama de la figura 2.

He seguido esta arquitectura, pero podía haber implementado una API REST para hacer la aplicación más extensible y más accesible, pero no he implementado esta por falta de tiempo.

Desarrollo del proyecto

Para el desarrollo de este proyecto he requerido de dos lenguajes de programación PHP y Javascript exclusivamente, no he usado frameworks de ningún tipo pues en el documento de la práctica no especificaba que se pudiesen usar, por lo que todo está resuelto con estos dos lenguajes.

PHP

El planteamiento de la arquitectura que he seguido (MVC) me ha llevado a usar clases PHP en casi todo el proyecto, salvo en los archivos controladores como index.php y portada.php, pues carecía de sentido crear una clase de estos.

Otros dos archivos para los que no he usado clases son los archivos que me proveen de funcionalidades, que serían funciones.php y obtenerEntradasAjax.php.

Empezando por funciones.php la decisión de que no sea una interfaz que la implementen index.php y portada.php es debido a que los controladores no los he planteado como clases por lo que no pueden implementar una interfaz. Este archivo contiene una serie de funcionalidades comunes para evitar la sobrecarga de los archivos index.php y portada.php y para favorecer la reutilización.

Funcionalidades del archivo **funciones.php**:

```php
/**
 * Filtra los datos de un formulario
 * @param $datos
 * @return string
 */
function filtrado($datos)
{
    if (gettype($datos) == "string") {
        $datos = trim($datos); // Elimina espacios antes y después de los datos
        $datos = stripslashes($datos); // Elimina backslashes \
        $datos = htmlspecialchars($datos); // Traduce caracteres especiales en entidades HTML
    }
    return $datos;
}
```

Código 1: Función de filtrado para los datos de un formulario

```
function  validarTelefono($telefono){
    return preg_match("/^((\+?34([ \t|\-])?)?[9|6|7|8]((\d{1}([
\t|\-])?[0-9]{3})|(\d{2}([ \t|\-])?[0-9]{2}))([ \t|\-])?[0-9]{2}([
\t|\-])?[0-9]{2})/",$telefono);
}
```

Código 2: Ejemplo de validación para el teléfono

Estas funciones de validación pueden parecer redundantes con respecto a que también se realizan en el lado del cliente con Javascript pero no lo son, pues una persona puede construirse un formulario idéntico al nuestro sólo que sin validación de Javascript por lo que cuando lleguen los datos al servidor estarán sin validar y podemos crear una situación de riesgo.

```
/**
 * Crea el comentario de una entrada
 * @param $bd
 * @return bool
 */
function crearComentario($bd,$usuario){
    //Hago un filtrado previo por si hay valores indeseables en el
array
    $arrayFiltrado=array();
    foreach ($_POST as $k => $v)
        $arrayFiltrado[$k] = filtrado($v);
    $entrada = new Entrada($arrayFiltrado["entrada"]);
    $comentario = new
Comentario(NULL,$arrayFiltrado["comentario"],$arrayFiltrado["fechac
reacion"]);
    $daoComentario = new Comentarios($bd);
    $daoComentario->addComentario($comentario,$entrada,$usuario);
    return true;
}
```

Código 3: Ejemplo de función de creación de un comentario

Como la función del código 3 hay para elementos como Entrada, Imágen y Usuario, en este último también pueden ser modificados sus datos desde el formulario que se muestra en la sección de Información.

```php
/**
 * Carga las entradas de de un usuario
 * @param $usuario
 * @param $bd
 * @return array
 */
function cargaEntradasUsuario($usuario, $bd){
    $daoPublicaciones = new Entradas($bd);
    return $daoPublicaciones->getEntradasUsuario($usuario);
}
```

Código 4: Ejemplo de función para cargar entradas de un usuario.

Como la función del código 4 hay para elementos como Comentarios, Imágenes y Usuarios, con sus correspondientes datos.

```php
/**
 * Comprueba el intento de login del usuario
 * @param $bd
 * @return bool|Usuario
 */
function compruebaLoginUsuario($bd){
    //Hago un filtrado previo por si hay valores indeseables en el
array
    $arrayFiltrado=array();
    foreach ($_POST as $k => $v)
        $arrayFiltrado[$k] = filtrado($v);

    //Utilizo los objetos DAO para añadir el usuario a la BD
    $daoUsuario = new Usuarios($bd);
    $usuario = new Usuario();
    $usuario->setPassword($arrayFiltrado["password"]);
    $usuario->setUsuario($arrayFiltrado["usuario"]);
    $resultado = $daoUsuario->compruebaCredenciales($usuario);
    //Compruebo si tengo usuario
    if($resultado!=false) {
        $usuario->setPassword(null);
        $usuario->setId($resultado["id"]);
        $usuario->setImagen($resultado["imagen"]);
        return $usuario;
    }
    else
        return false;
}
```

Código 5: Función para la comprobación de las credenciales del usuario

Esta función al terminar devuelve un objeto Usuario para guardarlo en la sesión si ha podido iniciar sesión correctamente sino devuelve false.

```php
/**
 * Sube un archivo al servidor con el nombre propio del archivo en
la ruta del
 * usuario ej: pepe/nombrearchivo.jpg
 * @param $bd
 * @param $usuario
 * @param $entrada
 * @return
 */
function subidaFichero($bd,$usuario,$entrada){
    if(isset($_FILES['imagen']['name']) && strlen
($_FILES['imagen']['name'])>0) {
        $dir_subida = $usuario . "/";
        ////////////NO TENGO PERMISOS DE CREAR CARPETAS!!!!!!!
        if ( !is_dir($dir_subida) && is_writable("../redsocial")) {
            mkdir($dir_subida,0755, true);
        }else
            $dir_subida="img/";
        $path = $_FILES['imagen']['name'];
        $ext = pathinfo($path, PATHINFO_EXTENSION);
        $fichero_subido = $dir_subida . $usuario .
date("Ymd_Hm").".".$ext;
        /////NO TENGO PERMISOS PARA SUBIR ARCHIVOS!!!!
        if(is_writable($dir_subida)) {
            if (!move_uploaded_file($_FILES['imagen']['tmp_name'],
$fichero_subido)) {
                echo "Problema de ataque de subida de
ficheros!.\n";
            }
        }
        $imagen = new Imagen(null, $fichero_subido, $entrada-
>getId());
        $daoImagenes = new Imagenes($bd);
        $daoImagenes->addImagen($imagen, $entrada);
        return $fichero_subido;
    }

}
```

Código 6: Funcionalidad para subir imágenes al servidor

Aunque la práctica decía que no era necesario que se subiesen imágenes al servidor he incluido dicha funcionalidad, aunque claro el servidor está configurado de manera que el usuario apache no pueda subir archivos o imágenes al servidor por lo que no funciona esta funcionalidad para este servidor.

Funcionalidad del archivo **obtenerEntradasAjax.php:**

La funcionalidad específica de este archivo es proporcionar los nombres de las entradas de un Usuario concreto en forma de archivo JSON lo que se asemejaría a una pequeña funcionalidad añadida de una API REST. Este archivo se usa cuando se quiere consultar las entradas de un Usuario pasando el ratón por encima mediante una llamada en AJAX para cargar las entradas en un ToolTip que se mostrará sobre la imagen.

En cuanto a los otros archivos como da DAO, modelos y vistas merece la pena poner algún ejemplo sobre algún archivo DAO.

Funcionalidad del archivo **class.Usuarios.php:**

```
/**
 * Comprueba las credenciales de un usuario
 * @param $usuario
 * @return mixed
 */
function compruebaCredenciales($usuario){
    $this->bd->setConsultaParametrizada("SELECT
u.id,u.usuario,u.imagen FROM usuario as u WHERE u.usuario=? AND
u.password=?",array($usuario->getUsuario(),sha1($usuario-
>getPassword())));
    return $this->bd->getFila();
}
```

Código 7: Comprueba las credenciales del usuario mediante una consulta parametrizada

Cabe la pena destacar que esta funcionalidad esta consulta está resuelta con prepare por ello es parametrizada, para evitar posibles ataques de SQLInjection.

```
/**
 * Obtiene los amigos del usuario logueado
 * @param $usuario
 * @return array
 */
function getAmigos($usuario){
    $this->bd->setConsultaParametrizada("SELECT
u.id,u.usuario,u.imagen FROM usuario as u INNER JOIN amigo as a ON
u.id=a.idUsuarioB OR u.id=a.idUsuarioA WHERE u.id!=? AND
(a.idUsuarioB=? OR a.idUsuarioA=?) GROUP BY
u.nombre",array($usuario->getId(),$usuario->getId(),$usuario-
>getId()));
    $usuarios = array();
    $i=0;
    while ($fila = $this->bd->getFila()){
        $usuarios[$i] = new Usuario();
        $usuarios[$i]->setId($fila["id"]);
        $usuarios[$i]->setUsuario($fila["usuario"]);
        $usuarios[$i]->setImagen($fila["imagen"]);
        $i++;
    }
    return $usuarios;
}
```

Código 8: Ejemplo de consulta de los amigos de un usuario

Como vemos se devuelve un array de objetos Usuario con los amigos de un usuario.

En cuanto a los controladores del proyecto vale la pena destacar algunas funcionalidades del controlador portada.php.

Funcionalidad del archivo **portada.php:**

```
///COMENTARIO
//Creo el comentario si hay datos por el POST que han sido validados
if(isset($_POST["comentar"],$_POST["comentario"],$_POST["fechacreaci
on"],$_POST["entrada"]) && $_SERVER["REQUEST_METHOD"] == "POST"){
    if(crearComentario($bd,$login)){
       // echo "<script type='text/javascript'>alert('Comentario
publicado!')</script>";
    }
}
```

Código 9: Ejemplo de validación de datos en PHP con la creación de un comentario por parte del usuario que ha iniciado sesión $login

```
if (isset($seccion))
    switch ($seccion) {
        case "portada":
            //Muestra los amigos del usuario actual
            $vistaAmigos->muestraAmigos();
            $vistaEntradas = new
VistaEntradas(cargaEntradasAmigos($user, $bd), $seccion,$pag);
            $vistaEntradas->muestraEntradasAmigos($user);
            $vistaActivos->muestraActivos();
            break;
//////////////////////AQUÍ HAY MÁS CASE…//////////////////////////
        case preg_match('/entrada_([0-9])+/', $seccion) ? true :
false:
            //Muestra los amigos del usuario actual
            $vistaAmigos->muestraAmigos();
            $vistaEntradas = new
VistaEntradas(cargaEntradasAmigos($user, $bd), $seccion,$pag);
            $entradaCompleta=cargaEntrada($seccion,$bd);
            $vistaEntradas->muestraEntrada($entradaCompleta);
            $comentarios =
cargaComentarios($entradaCompleta[0],$bd);
            $vistaComentarios = new VistaComentarios($comentarios);
            $vistaComentarios-
>muestraPublicaComentario($login,$entradaCompleta[0]);
            if($comentarios!=null && isset($comentarios) &&
!empty($comentarios))
                $vistaComentarios->muestraComentarios();
            $vistaActivos->muestraActivos();

            break;
    }
```

Código 10: Controla la vista dependiendo de la sección y el usuario tenga.

Javascript

La parte de Javascript se centra prácticamente en la validación y para ello he utilizado métodos como:

```
/**
 * Valida el telefono con los caracteres posibles que puede
introducir el usuario
 * @returns {boolean}
 */
function validarTelefono(){
    var telef;
    telef=document.getElementById("telefono");
   var er=/^((\+?34([ \t|\-])?)?[9|6|7|8]((\d{1}([ \t|\-])?[0-
9]{3})|(\d{2}([ \t|\-])?[0-9]{2}))([ \t|\-])?[0-9]{2}([ \t|\-])?[0-
9]{2})$/ ;
    if(!er.test(telef.value)){
      telef.className=telef.className + " error";
        telef.innerHTML="";
        telef.placeholder="Incorrecto";
        document.getElementById("errorT").style.display="block";
        return false;
    }
    return true;
}
/**
 * Limpia el campo de telefono de los errores cuando el usuario
entra en el campo para modificarlo
 */
function limpiarT(){
   var telef;
    telef=document.getElementById("telefono");
   telef.className=telef.className.replace(" error", "");
   telef.innerHTML="";
    telef.placeholder="Telefono";
    document.getElementById("errorT").style.display="none";
}
```

Código 11: Ejemplo de validación del campo teléfono con su función que limpia el error cuando deja de producirse o se va a escribir.

La única funcionalidad de Javascript que se sale de la validación ha sido la de mostrar los ToolTip con las entradas del usuario cuando se pase el ratón por una la imagen de un usuario. Para ello he usado una pequeña librería llamada ToolTip y AJAX para cargar las entradas asíncronamente, como se muestra en el código 12.

```
/**
 * Funcion para cargar el fichero con las entradas
 */
function cargarTextoAjax(id) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var jsonObj = JSON.parse(this.responseText);
            var entradas = "";
            var data = jsonObj.ent;
            for(var i in data)
                entradas+=data[i].entrada+"<br/>";


            mostrarTooltip(entradas);
        }
    };
    xhttp.open("GET", "funciones/obtenerEntradasAjax.php?id="+id,
true);
    xhttp.send();
}

/**
 * Muestra el popup con el contenido
 * @param contenido
 */
function mostrarTooltip(contenido){
    tooltip.show(contenido);
}
```

Código 12: Función AJAX que carga las entradas y muestra un ToolTip

Conclusión

He incluido funcionalidades nuevas como por ejemplo la de que como Usuario logueado vea en su Muro las Entradas suyas y las de los amigos de este, por lo que he incluido una nueva sección llamada **Portada (a esta sección se accede por el enlace del logo)** para cada usuario donde el usuario logueado verá en esta sección las entradas comentadas antes pero no podrá publicar, en la sección **Biografía** sin embargo sí puede publicar entradas y sólo tiene visibles las suyas propias no la de sus amigos**. ¡Importante!** La funcionalidad de añadir a mis amigos no está implementada a causa de no ser necesaria para la práctica, por lo que SÓLO los usuarios que ya están registrados previamente son los que tendrán amigos pues se han insertado las relaciones en la base de datos.

Además, si accedemos a un usuario amigo nuestro en la sección de amigos ahora sólo veremos los amigos de este que hemos seleccionado y las entradas de este permitiéndonos publicar

una entrada para este amigo.

Con todas estas mejoras y la práctica completa puedo decir que la práctica ha sido interesante y me ha servido para organizar mi código para futuros proyectos de una manera eficiente.

Referencias

[1] w3schools, «w3schools.com,» [En línea]. Available: https://www.w3schools.com/css/css_rwd_images.asp .

[2] D. Lázaro, «https://diego.com.es,» [En línea]. Available: https://diego.com.es/formularios-en-php.

[3] PHP, «http://php.net,» [En línea]. Available: http://php.net/manual/es/features.file-upload.post-method.php.

[4] bourbaki, «http://stackoverflow.com,» [En línea]. Available: http://stackoverflow.com/questions/4043741/regexp-in-switch-statement.

[5] jonniespratley, «http://snipplr.com/,» [En línea]. Available: http://snipplr.com/view/17419/javascript--create-excerpt/.

[6] S. Editorial, «http://www.scriptiny.com,» [En línea]. Available: http://www.scriptiny.com/2008/06/javascript-tooltip/.