PRÁCTICA: LOGIN Y REGISTRO CON PHP

Se deberán crear dos documentos .html: un registro y un login (usuario-contraseña). Ambos deberán estar vinculados entre sí, para poder ir de uno a otro y viceversa.
Se recomienda utilizar bootstrap u otro framework similar.

REGISTRO

En este caso, se pedirán al usuario los siguientes datos:

- Nombre
- Correo electrónico
- Tipo de usuario: profesor, alumno o administrador (lista desplegable)
- Usuario
- Contraseña
- Contraseña (volver a escribirla)

TIPOS DE USUARIO

0- elige tu rol	sin tipo de usuario (ERROR)
1- administrador/a	tendrá acceso a todos los datos de la base de datos.
2- profesor/a	tiene acceso a modificar sus datos personales y a ver los datos de sus alumnos, excepto la contraseña
3- estudiante	solo tiene acceso a ver y modificar sus datos personales, excepto la contraseña

OLVIDO CONTRASEÑA

En el caso de que algún usuario olvide su contraseña, se dirigirá a una página en la que se pida el nombre de usuario y la nueva contraseña repetida dos veces (comprobar).

LOGIN

Esta es la página de index.html; mediante un formulario, se pedirán al usuario dos datos: el usuario y la contraseña.

Se comprueban la coincidencia contra la base de datos, y si todo va bien, van a una página principal.php

BASE DE DATOS

La base de datos BDUSUARIOS, tendrá 2 tablas: USUARIOS Y TIPOSUSUARIO

USUARIOS(id, nombre, email, tipoUsuario, usuario, pwd)

TIPOSUSUARIO(id, tipoUsuario)

Ambas se relacionarán por el campo USUARIOS.tipoUsuario con TIPOSUSUARIO.id, no hace falta relacionarlos de momento.

FASE I: SIMPLEMENTE FUNCIONA

En la página de REGISTRO, insertar usuarios dentro de la tabla USUARIOS utilizando php con mysqli.

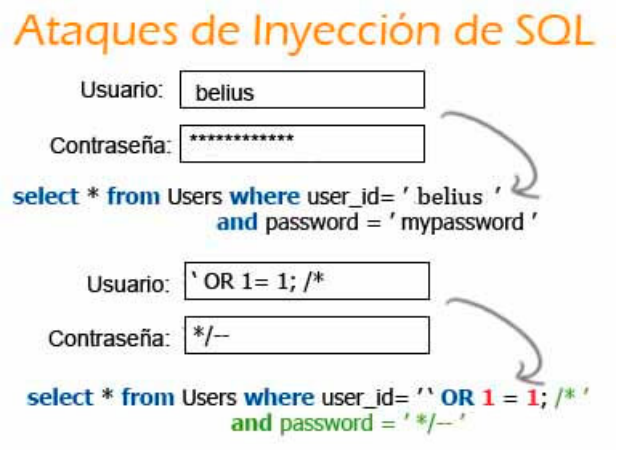
En la página de LOGIN, comprobar si existe el usuario y la contraseña en la base de datos, y si es así, enviar al usuario a una página principal.php. Si no coincide, informar del error.

En la página de OLVIDO CONTRASEÑA, comprobar que existe el usuario, y si es así, sobreescribir la PWD en la tabla USUARIOS, y luego enviar a la página de LOGIN.

INYECCIONES SQL:

Una vez que todo funciona correctamente, los datos se guardan, se recuperan, etc. intentaremos realizar una inyección de código SQL en la que se inserten datos de un usuario fake y otra en la que se borre toda la información de las tablas.

Se aporta un ejemplo sobre su funcionamiento:



FASE II: APORTANDO SEGURIDAD

1. VALIDACIÓN EN EL LADO DEL CLIENTE:

Es importante validar el formulario en cliente antes de enviar los datos al servidor, y validar también los datos que llegan al servidor, para evitar Inyecciones SQL, entre otras cosas.

Estas validaciones se realizarán con expresiones regulares o indicando el tipo de input (html), o con scripts (js).

- El **nombre** irá con apellidos, pero no permitirá números ni caracteres especiales.
- El **email** tendrá el típico formato de un correo electrónico.
- El **tipo de usuario** será una lista desplegable que cogerá los datos de la tabla TIPOSUSUARIO de nuestra base de datos.
- El **usuario** deberá ser una sola palabra (token) y sólo admitirá letras, números y la barra baja (sin ñ ni acentos). Tendrá un máximo de 12 caracteres.
- La **contraseña**, ha de tener al menos un número, una letra, y uno de los siguientes caracteres ($. , - #). Tendrá un mínimo de 8 caracteres y un máximo de 12.

Para la validación en cliente, es importante el atributo onsubmit en la etiqueta form, que permite o no cambiar de página según sea true o false. Por ejemplo, si tengo

```
<form action=”principal.php” onsubmit="return x()">

…

</form>
```

Y hago las comprobaciones en JS:

```
  <script>
     function x() {

…

if (a == b) {
                return false;
          } else{
                return true;
          }
}
  </script>
```

De esta manera, el formulario solo nos dejaría acceder a la página especificada en el action si los datos son los correctos.

2. HASH DE LA CONTRASEÑA

Es importante guardar un hash de la contraseña, y no la contraseña en sí; esto aporta un extra de seguridad en cuanto a que ni siquiera el administrador de la base de datos la puede adivinar. (muchas veces el factor humano es de lo más inseguro en entornos digitales)

Para ello, existe en php la función hash. Ejemplo: hash('sha256','Hola');

sha256 es el algoritmo de cifrado utilizado

Hola es el mensaje a hashear

SALIDA: e633f4fc79badea1dc5db970cf397c8248bac47cc3acf9915ba60b5d76b0e88f (64 caracteres)

3. VALIDACIÓN EN EL LADO DEL SERVIDOR

En el **formulario de registro**, validaremos en servidor los siguientes datos:

Solo se podrá ser administrador/a si el mail termina en xunta.gal. En caso contrario, si un usuario escoge ser admin y su mail no cumple el requisito, se guardará en la base de datos como un usuario sin tipo.

El usuario no se podrá repetir, por lo deberá comprobarse si ya existe un usuario igual.

En ambos casos, deberemos notificar al usuario de todos estos errores.

Una vez se guarden los datos del usuario y todo esté correcto, se enviará a éste al formulario de login (index.html)

Para validar los datos exactos que el usuario debe escribir en el **formulario de login**, vamos a tener que validar en el lado del servidor, y comparar con los datos que están guardados en la base de datos.

Si todo es correcto, se enviará al usuario a la página principal.php.

4. UTILIZACIÓN DE SESIONES

Para garantizar que solo los usuarios autorizados puedan acceder a cierta información o a ciertas páginas (principal.php), vamos a tener que utilizar las variables de sesión.

Sesiones en PHP (diego.com.es)

RÚBRICA

CRITERIO	DESCRIPCIÓN	PUNTOS MÁX
1. Estructura	- el código está bien organizado en archivos/ carpetas y separado adecuadamente HTML, CSS, JS y PHP	0.5
	- gestión eficiente de la base de datos (id autoincremental, usuario UNIQUE)	0.5
2. Funcionalidad	-login/registro completamente funcional	1
	-existe una página funcional para recuperar la contraseña	1
	- se utilizan correctamente variables de sesión para usuarios logueados	0.5
	-las variables de sesión, así como las conexiones expiran adecuadamente por inactividad o cierre de sesión	0.5
3. Corrección	-hash de la contraseña en la base de datos	0.5
	-control de usuarios admin válidos	0.5
	-validación de datos tanto en cliente como en servidor	0.5
	-gestión correcta de errores en el servidor	0.5
	-documento .pdf de inyecciones SQL con capturas	0.5
4. Presentación	-código bien presentado con identación, comentarios y nombres de variable descriptivos	0.5
	-estética de la interfaz: diseño atractivo, tipografía legible, uso coherente de colores, disposición adecuada de los elementos y diseño responsive. Uso consistente de estilos en toda la aplicación	0.5
	-buena usabilidad: la aplicación es intuitiva y fácil de navegar, envía mensajes de error claros y fáciles de entender	0.5
5. Esfuerzo	- implementa el sistema de roles de usuario, mostrando diferentes datos o accesos según el tipo de usuario que se loguea	0.5
	- se pueden modificar, borrar o crear usuarios, según el tipo de usuario que se ha logueado	0.5
	- inclusión de una foto de perfil de usuario (carga y almacenamiento de la imagen	0.5
	- limita el número de intentos en login para evitar ataques de fuerza bruta	0.5