



3. DESARROLLO WEB AVANZADO CON PHP



CONTENIDOS

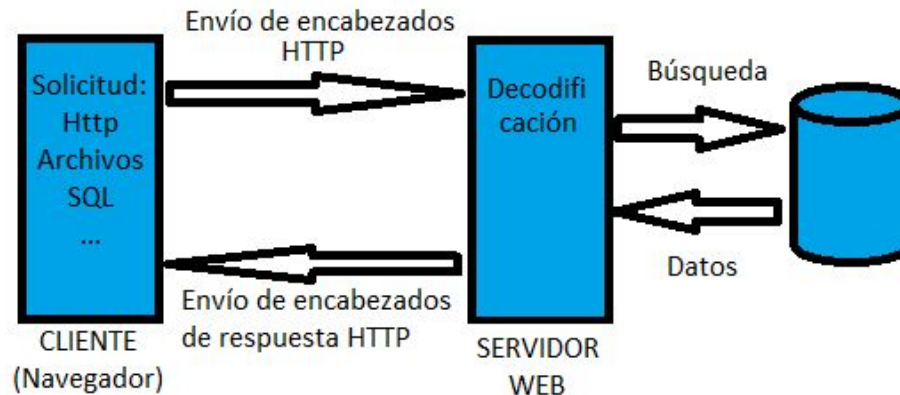
- Paso de parámetros y mantenimiento de estado.
- Formularios. Envío de información.
- Formulario de login para la autenticación de usuarios.
- Autenticación y autorización de usuarios.
- Cookies
- Sesiones
- Envío de correo electrónico
- Ficheros
- Acceso a bases de datos (pdo (jdbc) o driver específico (mysqli))
- Pruebas y depuración

PASO DE PARÁMETROS Y MANTENIMIENTO DE ESTADO

HTTP permite realizar comunicaciones con distintos métodos (8 en total) que nos servirá para tener el mismo estado entre peticiones:

- <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>

De todos ellos, los que más nos interesan son **GET** (obtener) y **POST** (enviar).



PASO DE PARÁMETROS Y MANTENIMIENTO DE ESTADO

MÉTODOS SEGUROS
SIN ACCIÓN EN EL SERVIDOR

GET IMPLEMENTADO DESDE HTTP/1.1

HEAD INSPECCIÓN DE RECURSOS DE LA CABECERA

PUT ALMACENAMIENTO DE DATOS AL SERVIDOR - INVERSO A GET

POST ENVÍO DE DATOS DE ENTRADA PARA SER PROCESADOS

PATCH MODIFICACIÓN PARCIAL DE UN RECURSO

TRACE MENSAJE RECIBIDO DE ECHO

OPTIONS CAPACIDADES DEL SERVIDOR

DELETE ELIMINAR UN RECURSO - NO GARANTIZADO

MENSAJES CON "BODY"
ENVÍAN DATOS AL SERVIDOR

PASO DE PARÁMETROS

https://en.wikipedia.org/wiki/HTTP#Request_methods

Request method ↕	RFC ↕	Request has payload body ↕	Response has payload body ↕	Safe ↕	Idempotent ↕	Cacheable ↕
GET	RFC 9110	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 9110	Optional	No	Yes	Yes	Yes
POST	RFC 9110	Yes	Yes	No	No	Yes
PUT	RFC 9110	Yes	Yes	No	Yes	No
DELETE	RFC 9110	Optional	Yes	No	Yes	No
CONNECT	RFC 9110	Optional	Yes	No	No	No
OPTIONS	RFC 9110	Optional	Yes	Yes	Yes	No
TRACE	RFC 9110	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

PASO DE PARÁMETROS

GET VS POST

Con **GET** se solicitan páginas web a un servidor, por ejemplo al pulsar un vínculo o insertando una dirección en el navegador. Al utilizar éste método se pueden pasar parámetros a través de la **URL**. Para ello se accede a la página utilizando el carácter “?” y cada parámetro es una dupla “nombre=valor” pudiendo enlazar varios parámetros con el carácter “&”. En **PHP** se podrá leer con la variable **\$_GET**.

abc.com/web.php?v1=1&v2=2

En este caso se pide el recurso “web.php” con las variables “v1” y “v2” que tienen como valores 1 y 2.

PASO DE PARÁMETROS



GET VS POST

POST se utiliza para enviar formularios al servidor que es la forma más habitual de enviar información a un servidor.

En los **formularios** el usuario tiene la opción de rellenar distintos tipos de campos o controles y lo envía todo junto al pulsar un botón de envío.

En **PHP** podemos acceder a dicha información a través del array **\$_POST** de manera parecida a **\$_GET** aunque en este caso no podemos verlos reflejados junto a la URL.



PASO DE PARÁMETROS

Desde nuestro recurso **PHP** podemos acceder a los parámetros introducidos por GET con la variable “**\$_GET**” que es un array cuyas claves son todos los parámetros y así podremos acceder a los valores correspondientes.

Para evitar que haya parámetros en blanco y obtengamos un error podemos usar las funciones “empty()” o “is_null()” que prácticamente hacen lo mismo salvo porque funcionan diferente si hay parámetro sin valor (“web.php?nombre”).

PASO DE PARÁMETROS. EJERCICIOS

Para los ejercicios siguientes comprobar que existen los parámetros y que son del tipo correcto.

Ejercicio 1: realiza una página PHP que reciba por GET un parámetro nombre y realice el “hola xxxxx” con dicho nombre.

Ejercicio 2: realiza una página PHP que reciba por GET dos parámetros base y altura y calcule su área.

Ejercicio 3: realiza una página PHP que reciba por GET tres parámetros nombre, apellidos y edad los muestre diciendo “Eres xxxxx zzzzzz y tienes XX años”



PASO DE PARÁMETROS

Desde un **formulario** básico HTML podemos indicar el recurso al que se enviarán los datos (“**action**”) para ser procesados y el método utilizado (“**method**”).

Podemos utilizar tanto el método GET como POST aunque lo habitual es utilizar el segundo.

En nuestro script **PHP** el atributo “**name**” que aparece en las etiquetas del formulario serán las que tengamos que utilizar para extraer los datos.



PASO DE PARÁMETROS. FORMULARIO

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Formulario básico</title>
  </head>
  <body>
    <form action="web.php" method=POST>
      <input name="usuario" type="text">
      <input name="clave" type="password">
      <input type="submit">
    </form>
  </body>
</html>
```



PASO DE PARÁMETROS. FORMULARIO

Los datos se pueden recoger mediante la variable global **\$_POST** de manera muy parecida a como se hacía con GET.

En este caso no veremos los datos en la **URL** aunque existen herramientas para poder verlos.

```
<?php
```

```
//Mostramos lo que envía en el POST
```

```
$user = $_POST["usuario"];
```

```
$password = $_POST["clave"];
```

```
echo "Usuario: " . $user . " <br>;
```

```
echo "Clave: " . $password . " <br>;
```



PASO DE PARÁMETROS. EJERCICIOS

Volver a hacer los ejercicios siguientes comprobando que existen los parámetros y que son del tipo correcto.

Ejercicio 1: realiza una página PHP que reciba por POST un parámetro nombre y realice el “hola xxxxx” con dicho nombre.

Ejercicio 2: realiza una página PHP que reciba por POST dos parámetros base y altura y calcule su área.

Ejercicio 3: realiza una página PHP que reciba por POST tres parámetros nombre, apellidos y edad los muestre diciendo “Eres xxxxx zzzzzz y tienes XX años”



EJERCICIO PARA ENTENDER EL PASO DE PARÁMETROS

Con los siguientes 2 ficheros se puede practicar para entender bien el acceso a nuestra aplicación web

- En el **primer ejemplo** tenemos varios enlaces para acceder a nuestra aplicación con varios parámetros.
- En el **segundo ejemplo** tenemos 2 formularios para hacer envíos mediante GET en el primer formulario y POST en el segundo.

Crea la aplicación “miaplicacion.php” que reciba los datos en cada caso.

EJERCICIO PARA ENTENDER EL PASO DE PARÁMETROS

```
<!DOCTYPE html>
<html><head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
</head><body>
    <a target="_blank" href=miaplicacion.php>
        Acceso a mi aplicación sin parámetros</a><br/>
    <a target="_blank" href=miaplicacion.php?arg1=1>
        Acceso a mi aplicación con un parámetro</a><br/>
    <a target="_blank" href=miaplicacion.php?arg1=1&arg2=2>
        Acceso a mi aplicación con dos parámetros</a><br/>
    <a target="_blank" href=miaplicacion.php?arg3=3&arg4=4>
        Acceso a mi aplicación con otros dos parámetros diferentes</a><br/>
</body></html>
```

EJERCICIO PARA ENTENDER EL PASO DE PARÁMETROS

```
<!DOCTYPE html>
<html><head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
</head><body>
    <form target="_blank" action = "miaplicacion.php" method = "GET">
        Arg1: <input name = "arg1" type = "text"><br/>
        Arg2: <input name = "arg2" type = "text"><br/>
        <input type = "submit" value="Formulario con GET"><br/>
    </form><br/>
    <form target="_blank" action = "miaplicacion.php" method = "POST">
        Arg1: <input name = "arg1" type = "text"><br/>
        Arg2: <input name = "arg2" type = "text"><br/>
        <input type = "submit" value="Formulario con POST"><br/>
    </form>
</body></html>
```


FORMULARIOS. ENVÍO DE INFORMACIÓN.

- En general los **formularios** sirven para **enviar** datos al servidor.
- Se pone a disposición del usuario diferentes **campos** que deberá rellenar antes de enviar.
- El servidor procesa dichos datos y genera una **respuesta**.
- Existen distintos **objetivos** para hacer formularios como son:
 - Formulario de login.
 - Formulario en un sólo fichero.
 - Formulario de subida de ficheros.



FORMULARIOS. HEADER

The diagram illustrates the structure of an HTTP response. It shows a sequence of lines representing the response, with arrows indicating the grouping of headers. The lines are: HTTP/1.1 200 OK, Access-Control-Allow-Origin: *, Connection: Keep-Alive, Content-Encoding: gzip, Content-Type: text/html; charset=utf-8, Date: Wed, 10 Aug 2016 13:17:19 GMT, Etag: "d9b3b01a9a0dcff22e22c43e30f65c11f6b71b", Keep-Alive: timeout=5, max=999, Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT, Server: Apache, Set-Cookie: cerftoken=, Transfer-Encoding: chunked, Vary: Cookie, Accept-Encoding, X-Frame-Options: DENY, and (body). The headers are grouped into three categories: Response headers (Access-Control-Allow-Origin, Connection, Content-Encoding, Content-Type, Date, Etag, Keep-Alive, Last-Modified, Server, Set-Cookie, Transfer-Encoding, Vary, X-Frame-Options), Representation headers (Content-Type, Content-Encoding), and General headers (Date, Last-Modified, Server, Set-Cookie, Transfer-Encoding, Vary, X-Frame-Options).

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:19 GMT
Etag: "d9b3b01a9a0dcff22e22c43e30f65c11f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: cerftoken=
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY
(body)
```

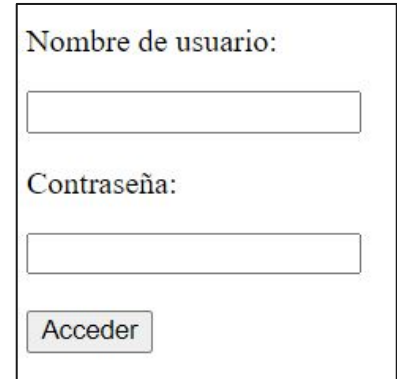
Siempre que tengamos que **hacer una redirección** tras un formulario (por ejemplo en el login) debemos tener en cuenta lo siguiente:

- Hay que enviar las **cabeceras** antes de empezar con el cuerpo de una respuesta.
- Esto implica utilizar la función “**header()**” incluso antes de que se empiece a escribir la salida.
- La función “**header()**” no se puede utilizar después de hacer cualquier envío de información, por ejemplo un “echo”, ya que dará error por no ser el primer dato enviado.

FORMULARIO DE LOGIN PARA LA AUTENTIFICACIÓN DE USUARIOS

- Con este tipo de **formularios** vamos a comprobar que con los datos introducidos podemos tener acceso al resto del sistema.
- En este caso debe haber una **comprobación** y redirigir hacia la aplicación o hacia un mensaje de error:

```
if ($_POST["clave"] === "1234"){  
    header("Location:bienvenido.html");  
} else {  
    header("Location:error.html");  
}
```



Nombre de usuario:

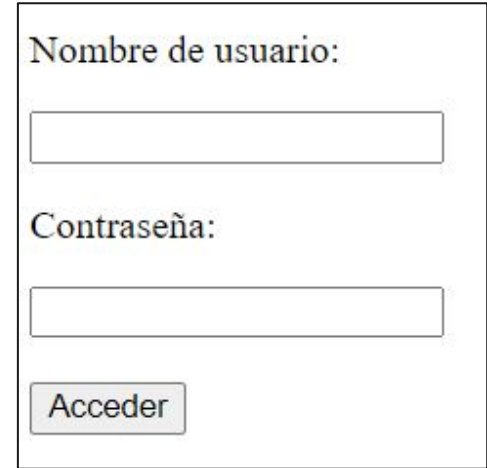
Contraseña:

Acceder

- Modificando un poco el formulario anterior a continuación obtendremos un formulario para nuestro **login**.

FORMULARIO DE LOGIN PARA LA AUTENTIFICACIÓN DE USUARIOS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulario de login</title>
    <meta charset = "UTF-8">
  </head>
  <body>
    <form action = "login.php" method = "POST">
      <p> Nombre de usuario:</p>
      <input name = "usuario" type = "text"><br/>
      <p> Contraseña:</p>
      <input name = "clave" type = "password"><br/><br/>
      <input value = "Acceder" type = "submit">
    </form>
  </body>
</html>
```



Visual representation of the login form:

Nombre de usuario:

Contraseña:

Acceder



PRÁCTICA ENTREGABLE SOBRE FORMULARIOS POST

Hacer los apartados siguientes en el mismo fichero PHP comprobando que existen los parámetros y que son del tipo correcto:

- Si recibe por POST un parámetro, nombre, realice el “hola xxxxx” con dicho nombre.
- Si recibe por POST dos parámetros, base y altura, calcule su área y lo muestre
- Si recibe por POST tres parámetros, nombre, apellidos y edad, los muestre diciendo “Eres xxxxx zzzzzz y tienes XX años”

La aplicación mostrará previamente el número de parámetros existentes (count) y lo que va a hacer.

FORMULARIO EN UN SÓLO FICHERO

- Este tipo de formularios integran en el **mismo fichero** el **formulario** HTML y el **procesamiento** con PHP de dicho formulario, es decir, se envía y es procesado por el mismo fichero.
- Normalmente al abrir la página (desde un enlace) o con el método **GET** (un “header(Location:)”) se accede a rellenar el formulario mientras que si se accede con el método **POST** es porque ya se ha rellenado y se tiene que procesar.
- Podemos saber cómo se ha llegado a nuestra aplicación consultado la variable `$_SERVER["REQUEST_METHOD"]`.





FORMULARIO EN UN SÓLO FICHERO

Algunas cosas que podemos utilizar para facilitar esta comunicación son:

- Uso de una variable (\$err) para mostrar un mensaje de error.
- Uso de una variable (\$usuario) para mantener el usuario introducido.
- Uso de la variable \$_SERVER["PHP_SELF"] para redireccionar el formulario a la propia página sin saber el nombre del fichero.
- Se debe utilizar junto con la función htmlspecialchars que filtra los caracteres por seguridad,
“htmlspecialchars(\$_SERVER["PHP_SELF"]);”.
- Usando \$_SERVER["REQUEST_METHOD"] sólo validamos el formulario si se han introducido los datos mediante POST.



FORMULARIO EN UN SÓLO FICHERO

```
<?php /* si va bien redirige a principal.php si va mal, mensaje de error */
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if ($_POST['usuario'] === "usuario" and $_POST["clave"] === "1234"){
        header("Location: acceso.php");
    }else{
        $err = true;
        $usuario=$_POST['usuario'];
    }
}
}??>
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de login</title>
        <meta charset = "UTF-8">
    </head>
```

Usuario	<input type="text"/>	
Clave	<input type="text"/>	<input type="button" value="Enviar"/>

Revise usuario y contraseña		
Usuario	<input type="text" value="usuario"/>	
Clave	<input type="text"/>	<input type="button" value="Enviar"/>



FORMULARIO EN UN SÓLO FICHERO

```
<body>
  <?php if(isset($err)){
    echo "<p> Revise usuario y contraseña</p>";
  }?>
  <form action = "<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"
    method = "POST">
    <label for = "usuario">Usuario</label>
    <input value = "<?php if(isset($usuario))echo $usuario;?>"
      id = "usuario" name = "usuario" type = "text">
    <label for = "clave">Clave</label>
    <input id = "clave" name = "clave" type = "password">
    <input type = "submit">
  </form>
</body>
</html>
```



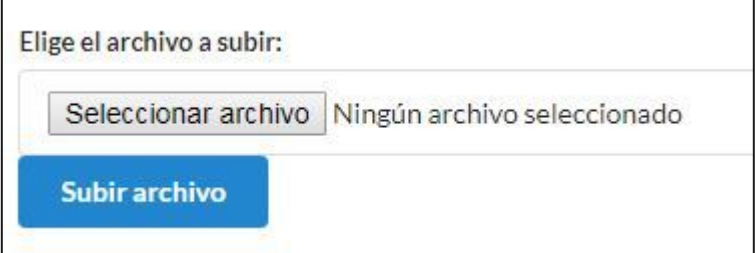
PRÁCTICA ENTREGABLE. FORMULARIO ALL IN ONE

Crea un único fichero PHP (formulario todo en un fichero) que realice las siguientes opciones:

- La primera vez que se pida, debe mostrar un formulario de login.
- El formulario de login seguirá mostrándose con un mensaje de error si no se accede correctamente.
- Si se accede correctamente aparecerá otro formulario (estará implementado en el mismo PHP) con los siguientes características:
 - Tendrá una subida de ficheros.
 - Tendrá un aviso del máximo en esa subida de ficheros.
 - Tendrá un campo que será el nombre con el que guardar el archivo en servidor.
- Al enviar un fichero se informará al usuario del proceso realizado y los diferentes nombres del fichero (de subida y de guardado si se hizo correctamente)

FORMULARIO DE SUBIDA DE FICHEROS

- Este es un caso especial de formulario en el que el objetivo es evidentemente **subir un fichero**.
- Se necesita un **atributo especial** en el formulario:
“enctype=“multipart/from-data”” junto al método **POST**.
- En el formulario tendremos un **input** de tipo “**file**” que abre una ventana para seleccionar el fichero.
- Al enviarlo se rellena la variable **\$_FILES** con los datos del fichero enviado. [Referencia](#).



Elige el archivo a subir:

Seleccionar archivo Ningún archivo seleccionado

Subir archivo

FORMULARIO DE SUBIDA DE FICHEROS

A screenshot of a web form for uploading files. It has a title "Elige el archivo a subir:" followed by a text input field containing "Seleccionar archivo" and a status message "Ningún archivo seleccionado". Below these is a blue button labeled "Subir archivo".

Elige el archivo a subir:	
Seleccionar archivo	Ningún archivo seleccionado
<button>Subir archivo</button>	

La variable `$_FILES` contiene un **array** con todos los ficheros que se hayan enviado a través del POST.

Para cada uno de los ficheros se puede acceder a las siguientes **propiedades**:

- “name”: `$_FILES[“fichero”][“name”]`
- “size” (en bytes): `$_FILES[“fichero”][“size”]`
- “type” (MIME): `$_FILES[“fichero”][“type”]`
- “tmp_name” (nombre en el servidor): `$_FILES[“fichero”][“tmp_name”]`
- “error” (error en la subida): `$_FILES[“fichero”][“error”]`

FORMULARIO DE SUBIDA DE FICHEROS

A screenshot of a web form for file upload. At the top, it says "Elige el archivo a subir:". Below this is a text input field with the placeholder "Seleccionar archivo" and the text "Ningún archivo seleccionado" to its right. At the bottom of the form is a blue button with the text "Subir archivo".

Elige el archivo a subir:	
Seleccionar archivo	Ningún archivo seleccionado
Subir archivo	

El fichero se guarda **temporalmente** en el directorio temporal del servidor y se podrá mover (sino será eliminado del servidor) con la función “**move_uploaded_file(\$fichero, \$destino)**” devolviendo un booleano falso si no se ha podido mover.

El fichero `$fichero` será el nombre temporal de uno de los ficheros enviados a través del POST mientras que `$destino` será una dirección válida (y que debe contener el nombre final).

Por ejemplo:

```
move_uploaded_file(
    $_FILES["fichero"]["tmp_name"],
    "subidas/". $_FILES["fichero"]["name"]);
```

FORMULARIO DE SUBIDA DE FICHEROS

```
<!DOCTYPE html>
<html>
  <body>
    <form action="subir.php" method="post"
      enctype="multipart/form-data">
      Seleccionar el fichero para subir
      <input type="file" name="fichero">
      <input type="submit" value="Subir fichero">
    </form>
  </body>
</html>
```



Elige el archivo a subir:

Seleccionar archivo Ningún archivo seleccionado

Subir archivo

FORMULARIO DE SUBIDA DE FICHEROS

```
<?php
    $tam = $_FILES["fichero"]["size"];
    if($tam > 256 * 1024){
        echo "<br>Demasiado grande";
        return;
    }
    echo "Nombre del fichero: " . $_FILES["fichero"]["name"];
    echo "<br>Nombre temporal del fichero en el servidor: " .
        $_FILES["fichero"]["tmp_name"];
    $res = move_uploaded_file($_FILES["fichero"]["tmp_name"], "subidas/" .
        $_FILES["fichero"]["name"]);
    if($res){
        echo "<br>Fichero guardado";
    } else {
        echo "<br>Error";
    }
}
```

Elige el archivo a subir:

Seleccionar archivo

Ningún archivo seleccionado

Subir archivo



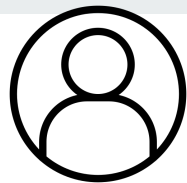
FORMULARIOS. EJERCICIOS

Realizar los siguientes formularios comprobando que se envían los datos correctamente:

Ejercicio 1: Envío de 3 campos (nombre y apellidos) con GET y POST en 2 formularios diferentes pero que llegue a la misma página y sea capaz de mostrar los 3 campos.

Ejercicio 2: Envío de un número. Redirigir a páginas diferentes si el número es par o impar.

Ejercicio 3: Crea un formulario para subir un fichero y que deje cambiar su nombre cuando se suba al servidor.



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

En general, cuando hablamos de aplicaciones web, tendremos que realizar 2 tareas, la **autenticación** de los usuarios y la **autorización** de sus permisos.

En el primer paso se debe hacer “**login**” ya sea con un formulario o a través de algún servicio externo, por ejemplo a través de **Oauth** como puede ser Google. En este tipo de servicios se entabla una comunicación para utilizar las credenciales en otra aplicación.

Una vez nuestro usuario haya accedido a nuestro sistema se deben comprobar los **permisos** de los que dispone.



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

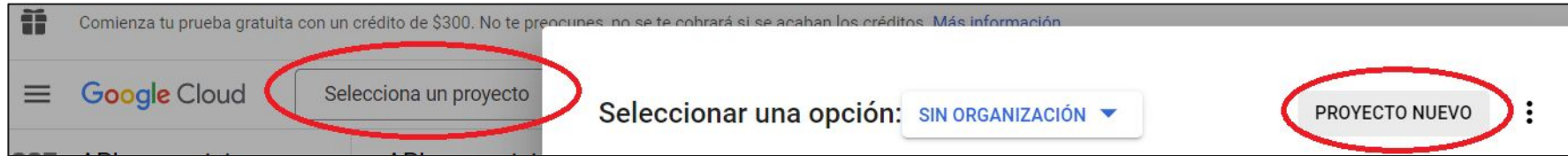
Como se ha mencionado, con **Oauth** lo que se pretende es que nuestra aplicación haga una petición, en este caso será a **Google**, para que allí se realice la validación del usuario, y una vez validado se obtiene dicha verificación en forma de un objeto (habitualmente llamado “**token**”) y que implica que el usuario ya ha sido autenticado.

Antes de conseguir una autenticación a través de Oauth con Google debemos habilitar un nuevo proyecto en el Dashboard de Google:

1. Accedemos a <https://console.developers.google.com/>
2. Allí creamos un nuevo proyecto para utilizar nuestra aplicación.



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS



Nombre del proyecto *
pruebaPHP

ID de proyecto: pruebaphp-387421. No se podrá cambiar más tarde. [EDITAR](#)

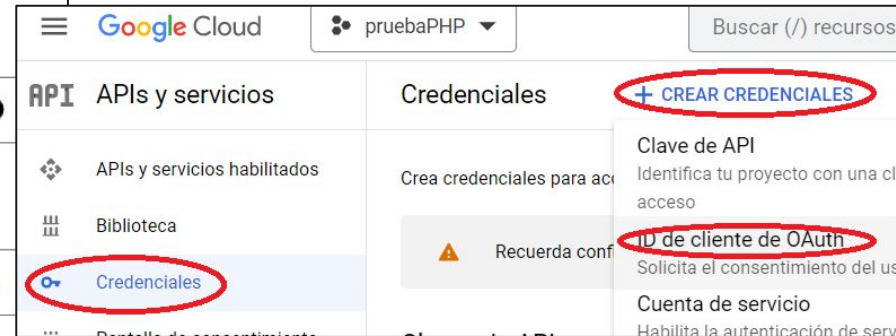
Organización *
iesjulianmarias.es

Selecciona una organización para vincularla a un proyecto. No podrás cambiar esta selección más adelante.

Ubicación *
iesjulianmarias.es

Organización o carpeta superior

CREAR CANCELAR



Crear ID de cliente de OAuth

Un ID de cliente se usa con el fin de identificar una sola app para los servidores de OAuth de Google. Si la app se ejecuta en varias plataformas, cada una necesitará su propio ID de cliente. Consulta [Configura OAuth 2.0](#) para obtener más información. [Obtén más información](#) sobre los tipos de clientes de OAuth.

⚠ Para crear un ID de cliente de OAuth, primero debes configurar la pantalla de consentimiento

CONFIGURAR PANTALLA DE CONSENTIMIENTO

User Type

☒ Interno ?

Solo está disponible para los usuarios de tu organización. No necesitarás enviar tu app para verificarla. [Obtén más información sobre el tipo de usuario](#)

☐ Externos ?

Está disponible para cualquier usuario de prueba con una Cuenta de Google. Tu app se iniciará en modo de prueba y solo estará disponible para los usuarios que agregues a la lista de usuarios de prueba. Una vez que la app esté lista para enviarse a producción, puede que debas verificarla. [Obtén más información sobre el tipo de usuario](#)

CREAR

Nombre de la aplicación *

pruebaPHP

El nombre de la aplicación que solicita el consentimiento

Correo electrónico de asistencia del usuario *

luis.gonzalez@iesjulianmarias.es

Para que los usuarios puedan contactarte si tienen problemas al dar consentimiento

Logotipo de la app

Este es tu logotipo. Ayuda a que los usuarios identifiquen tu app durante el consentimiento de OAuth.
Después de subir un logotipo, deberás configurarlo solo para uso interno o externo. [Obtén más información](#)

Vista previa del logotipo de la app

Información de contacto del desarrollador

Direcciones de correo electrónico *

luis.gonzalez@iesjulianmarias.es

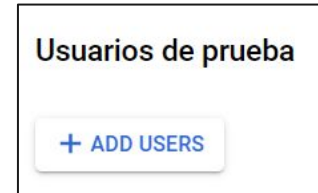
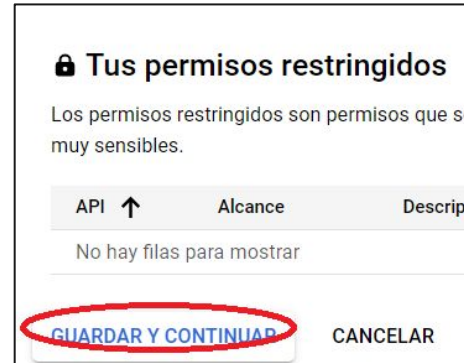
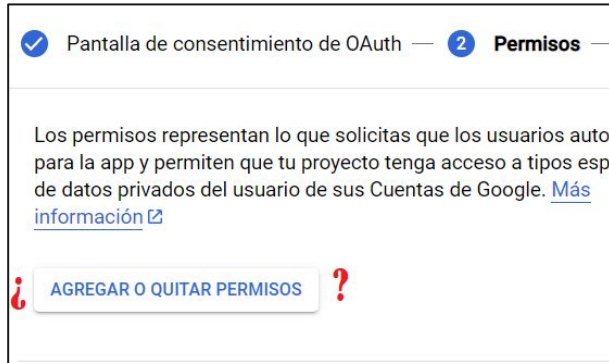
Google enviará notificaciones sobre cualquier cambio de correo electrónico.

GUARDAR Y CONTINUAR CANCELAR



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

En este punto se pueden limitar los permisos de los usuarios.

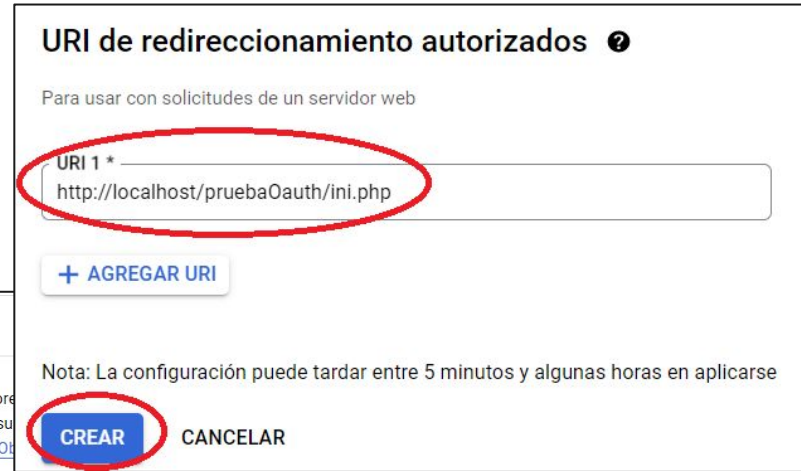
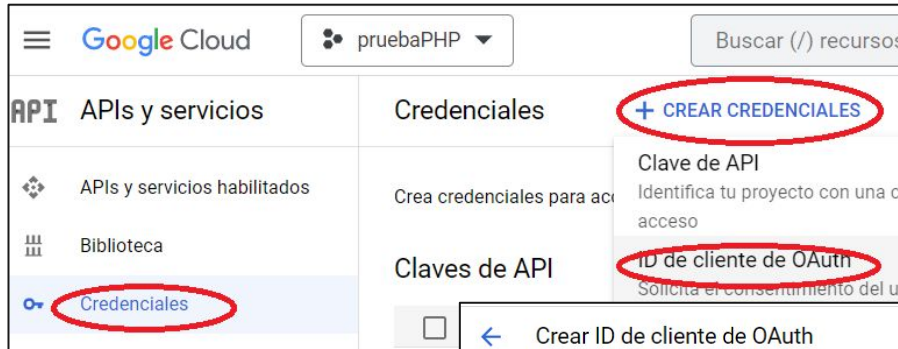


Además, en nuestro caso no haría falta porque tenemos “Tipo de usuario **interno**” pero si fuera **externo** hace falta crear un grupo de usuarios que puedan conectarse a nuestra aplicación mientras esté en período de prueba (en la pantalla de Resumen).



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

Ahora ya podemos crear el ID de autenticación:



Se creó el cliente de OAuth



[↓ DESCARGAR JSON](#)

☐

1

22 may 2023

39



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

Una vez creado el proyecto en Google y descargado el archivo .json, podemos crear nuestra aplicación.

- Creamos una nueva carpeta en nuestro sistema y nos situamos en ella para instalar el SDK para PHP de Google mediante composer (podría hacerse manualmente mediante descarga):

composer require google/apiclient

```
C:\pruebaOAuth>composer require google/apiclient
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^2.15 for google/apiclient
```

- A continuación se debe crear un fichero php con el código de las páginas siguientes en el cuál se redirige a la autenticación de Google o se muestran los datos del usuario.



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

<?php

```
require_once 'vendor/autoload.php';
$client = new Google_Client();
$client->setAuthConfigFile('claves.json');
// Parámetros para conseguir permisos sin el usuario presente
$client->setAccessType('offline');
$client->setApprovalPrompt('force');
// En este caso solicitamos permisto para el email y el perfil
// Más en https://developers.google.com/identity/protocols/googlescopes
$client->addScope("email");
$client->addScope("profile");
// URL que tenemos autorizada en nuestro proyecto Google
$client->setRedirectUri('http://localhost/pruebaOauth/ini.php');
```

Nombre

vendor

claves.json

composer.json

composer.lock

ini.php

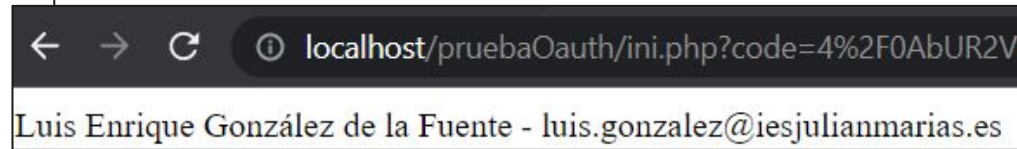
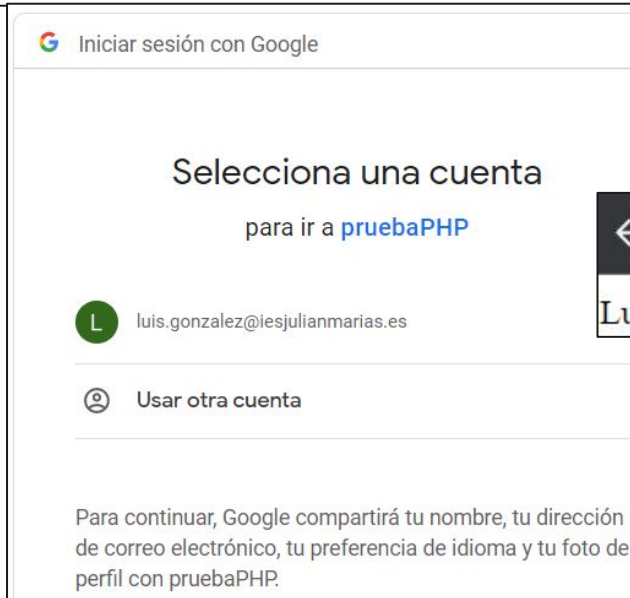
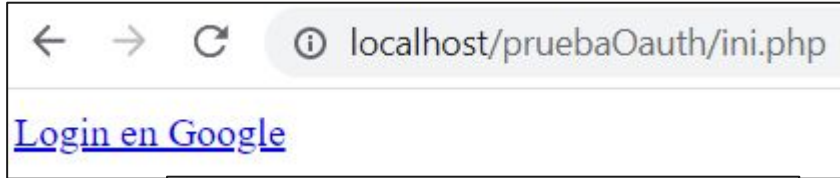


AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS

```
// Comprobamos si se ha identificado con anterioridad
if(!isset($_GET['code'])){ //Como no lo ha hecho redirigimos a Google
    echo "<a href='\".$client->createAuthUrl().\"'>Login en Google</a>";
}else{ //Ya está autenticado
    $token = $client->fetchAccessTokenWithAuthCode($_GET['code']);
    $client->setAccessToken($token['access_token']);
    // Guardamos el token para otros accesos
    $google_oauth = new Google_Service_Oauth2($client);
    $google_account_info = $google_oauth->userinfo->get();
    $email = $google_account_info->email;
    $name = $google_account_info->name;
    echo $name." - ".$email;
} // El resto del código ya tiene al usuario logueado
```



AUTENTIFICACIÓN Y AUTORIZACIÓN DE USUARIOS





PRÁCTICA ENTREGABLE. LOGIN

Crea una página que permita elegir la forma en la que hacer login:

- Opción “login local”: con un usuario y contraseña establecido en el código de tu aplicación
- Opción “login remoto”: para hacer un login a través de Google.

En ambos casos se debe pasar a una página de perfil de usuario con datos básicos y poder cerrar la sesión.

COOKIES



Las **cookies** son ficheros dejados por el servidor web en los ordenadores de los clientes que pueden almacenar información como la fecha de la última visita o las preferencias de idioma.

Cuando el **cliente** realiza la petición envía las cookies pertenecientes a dicho **servidor**.

Se pueden manejar las cookies mediante la función **setcookie()** que crea una nueva cookie que recibe varios valores como son el nombre, el valor o la fecha en la que expira la cookie.



COOKIES. ¿QUÉ SON?

¿QUÉ SON?

- Son un sistema de almacenamiento que usan los navegadores para que los sitios web “nos recuerden”.
- Cada vez que se entra a un sitio web, se genera un archivo de texto y se almacena en el navegador.
- El navegador envía dicho archivo al servidor en cada acceso nuevo.

BENEFICIOS:

- Permiten a los sitios web identificar a los usuarios y mejorar su experiencia y los tiempos de carga.
- Sólo pueden ser leídas por el sitio web que las ha creado.



COOKIES. ¿QUÉ SON?

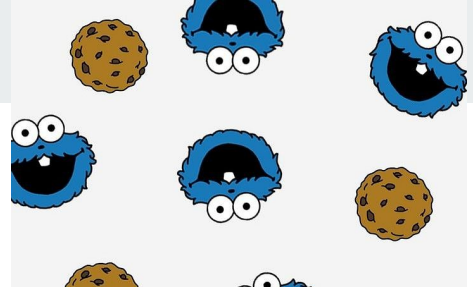
¿QUÉ INFORMACIÓN GUARDAN?

- Nuestra sesión de usuario.
- Nuestro comportamiento en esa web
- El idioma del navegador.
- Nuestras preferencias de usuario.

TIPOS DE COOKIES:

- **Propias:** las emite el sitio web que visitamos.
- De **terceros:** cuando son emitidas por una web externa de la que estamos.

COOKIES EN PHP

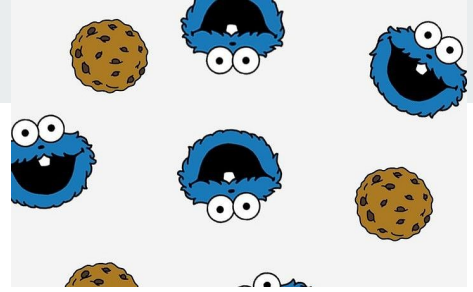


`setcookie("nombre", "valor", "tiempo", "ruta", "dominio", "seguridad", "httponly")`

- “nombre”: el nombre dado a la cookie.
- “valor”: el valor que le asignamos.
- “tiempo”: la fecha en la que expirará la cookie. Segundos pasados desde el 1 de enero de 1970. Por ejemplo `time() + 3600*24` para dar 1 día de validez.
- “ruta”: en la que se guardará la cookie dentro del servidor.
- “dominio”: el dominio en el que estará disponible, por ejemplo `php.com`.
- “seguridad”: true si sólo se quieren enviar a través de HTTPS.
- “httponly”: true si sólo se quieren enviar a través de HTTP.

Se puede destruir una cookie con `setcookie(nombre, valor, time()-100)`

COOKIES EN PHP



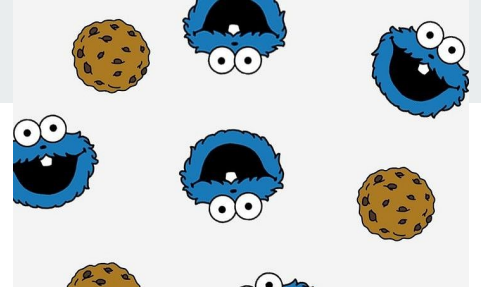
Referencia: <https://www.php.net/manual/es/function.setcookie.php>

Hay que tener presente que la llamada a **setcookie** se debe hacer **antes** que cualquier otra escritura desde PHP ya que se envía en las cabeceras de las peticiones HTTP (como el head).

Para acceder a los valores de la cookie se tiene el array que contiene la variable **\$_COOKIE** y usando como clave el nombre que se dio al crearla.

Cada cookie tendrá validez a partir de la siguiente petición del cliente.

COOKIES EN PHP. EJEMPLO



```
<?php
```

```
    if(!isset($_COOKIE["ejemplo"])){ // Caso de no existir
        setcookie("ejemplo", "1", time()+3600*24);
        echo "Bienvenida en el primer acceso.";
    } else {// Caso de existir
        $contador=(int)$_COOKIE["ejemplo"];
        $contador++;
        setcookie("ejemplo", $contador, time()+3);
        echo "Bienvenida en el acceso número $contador.";
    }
```

// ¿Qué significa el “+3” del tiempo? ¿Cómo se puede mejorar?
//¿Cómo se podría borrar la cookie añadiendo un enlace a otra página?



PRÁCTICA ENTREGABLE. FORMULARIO CON COOKIES

Crea una página con un formulario que de a elegir entre rojo y amarillo.

- Al seleccionarlo la misma página escribirá un título diciendo el color elegido (se permite poner el color de fondo o lo que se os ocurra).
- El resto de veces que se acceda a la página si hay cookie definida no se debe mostrar el formulario.

SESIONES



Las **sesiones** ayudan a asociar las distintas peticiones HTTP que realiza un cliente a un servidor.

El servidor asigna un **identificador de sesión** al usuario que en las siguientes peticiones debe ser enviado para que se le pueda tratar de igual manera.

Para mantener dicha sesión **el servidor utiliza una cookie en el cliente** por lo que si dicha cookie es borrada se perderá la sesión.

Para **crear** una sesión podemos usar el comando “**start_session()**”.

SESIONES



Al ejecutar “`start_session()`”:

- Si **no existe** una sesión anterior, **se crea**.
- Si hay una **sesión activa** la función que ejecuta el comando **se une** a dicha sesión pudiendo acceder a “`$_SESSION`” como array compartido entre todos los que acceden a dicha sesión.
 - `$_SESSION[“nombre”]=valor;`
- El procedimiento habitual es que tras loguearse en una aplicación web se inicie una sesión para compartir datos.
- Para **eliminar** una sesión se debe usar el comando `session_destroy()` pero además debemos eliminar la cookie y la variable de sesión.

SESIONES. EJEMPLO SIMPLE



```
<?php
    session_start();
    if(!isset($_SESSION["cuenta"])){
        $_SESSION["cuenta"]=0;
    }else{
        $_SESSION["cuenta"]++;
    }
    echo "En sesiones 1: " . $_SESSION["cuenta"];
    echo "<br><a target='_blank' href='sesiones2.php'>Sesiones2</a>";
```

// Segunda página que usa la sesión

```
<?php
    session_start();
    echo "En sesiones 2: " . $_SESSION["cuenta"];
```

SESIONES. EJEMPLO MÁS COMPLEJO

//Con la misma primera página.
// Segunda página que usa la sesión

```
<?php
    session_start();
    if(!isset($_SESSION['cuenta'])) {
        header("Location: sesiones1.php");
    }
    echo "En sesiones 2: ". $_SESSION["cuenta"];
    echo "<br><a href = 'sesiones3.php'> Salir <a>";
```



SESIONES. EJEMPLO MÁS COMPLEJO



//Tercera página que usa la sesión

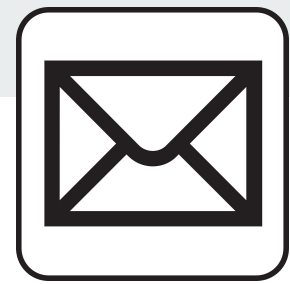
```
<?php
    session_start();
    if(!isset($_SESSION['cuenta'])) {
        header("Location: sesiones1.php");
    }
    $_SESSION = array(); // Borramos la variable de sesión
    session_destroy();    // Eliminar la sesion
    setcookie(session_name(), 123, time() - 1000); // Eliminar la cookie
    header("Location: sesiones1.php");
```




PRÁCTICA ENTREGABLE. SESIONES

Crea una página que haga lo mismo que con las cookies, es decir, un formulario que de a elegir entre rojo y amarillo.

Al seleccionarlo la misma página escribirá un título diciendo el color elegido (se permite poner el color de fondo o lo que se os ocurra) y el resto de veces que se acceda a la página si existe la sesión definida no se debe mostrar el formulario pero si el color elegido.



ENVÍO DE CORREO ELECTRÓNICO

- La función “**mail()**” permite enviar correos directamente pero dependemos del sistema:
 - <https://www.php.net/manual/es/function.mail.php>
- Esta función utiliza la **configuración local del servidor** (sendmail en **Linux**) pero pueden ser filtrados como SPAM.
- Además para **Windows** esta opción nos dará problemas al no tener configurado un servidor de correo.
- En este caso, vamos a ver la existencia de una **librería, PHPMailer** que nos permite ocuparnos de más detalles de formato a la hora de realizar el envío.

INSTALAR PHPMAILER

Para usar **PHPMailer** basta con tener al alcance de nuestra aplicación los ficheros de la librería:

- Accede a la página GITHUB de PHPMailer:
 - <https://github.com/PHPMailer/PHPMailer>
- Descarga una copia de la rama “master” desde “Code”, “Download ZIP”.
- Descomprime la carpeta “src” dentro del proyecto en donde quieras usar ésta librería. Puedes cambiar el nombre por ejemplo por “PHPMailer”.
- Si se utiliza **Gmail** (nuestro caso) se **debe** permitir el uso de aplicaciones poco seguras en <https://myaccount.google.com/lesssecureapps>
- Completa el código en tu proyecto teniendo en cuenta todas los ficheros necesarios antes de configurar los detalles del correo.

INSTALAR PHPMAILER

The screenshot shows the GitHub repository page for PHPMailer. The repository is public and has 25 issues, 7 pull requests, 13 security advisories, and 77 tags. The repository structure is as follows:

| File/Folder | Description |
|-------------|--------------------------------|
| .github | GH Actions: bust the cache sen |
| .phan | Don't analyse dependency test |
| docs | Docs links |
| examples | D'oh - forgot to add the exam |
| language | Updating author doc |
| src | Avoid POP3 client error messag |

The 'Code' dropdown menu is open, showing the following options:

- Clone (with a question mark icon)
- HTTPS (selected) GitHub CLI
- https://github.com/PHPMailer/PHPMailer.git (copy icon)
- Use Git or checkout with SVN using the web URL.
- Open with GitHub Desktop
- Download ZIP (highlighted in yellow)

EJEMPLO CON PHPMAILER

```
<?php
```

```
use PHPMailer\PHPMailer\PHPMailer;  
use PHPMailer\PHPMailer\Exception;  
require 'PHPMailer/Exception.php';  
require 'PHPMailer/PHPMailer.php';  
require 'PHPMailer/SMTP.php';
```

```
$mail = new PHPMailer();  
$mail->isSMTP();  
$mail->SMTPDebug = 2; // cambiar a 0 para no ver mensajes de error  
$mail->SMTPAuth = true;  
$mail->SMTPSecure = "tls";  
$mail->Host = "smtp.gmail.com";  
$mail->Port = 587;
```

EJEMPLO CON PHPMAILER

```
$mail->Username = ""; // Usuario de google
$mail->Password = ""; // Clave
$mail->SetFrom('user@gmail.com', 'Test');
$mail->Subject = "Correo de prueba";
$mail->MsgHTML('Prueba');
// $mail->addAttachment("fichero.pdf"); // Adjuntos
$address = "correo@gmail.com"; // Destinatario
$mail->AddAddress($address, "Test");
$resul = $mail->Send(); // Se envía
if(!$resul) { // Se comprueba según se haya enviado
    echo "Error" . $mail->ErrorInfo;
} else {
    echo "Enviado";
}
```



PRÁCTICA ENTREGABLE. CORREO ELECTRÓNICO

Realiza un formulario cuyo objetivo sea enviar un correo electrónico con adjuntos.

Debe incluir todos los datos para poder ser rellenado por el usuario.

Utiliza el ejemplo anterior pero con los datos incluidos en el formulario por el usuario.

FICHEROS



Para acceder a **ficheros** desde PHP podemos utilizar la **librería “Filesystem”** que tiene una sintaxis muy parecida a la utilizada en C o la librería **“Simple-XML”** para trabajar con ficheros XML.

Se debe tener en cuenta que cuando PHP accede al sistema de ficheros lo haremos dentro de los **directorios accesibles** en el servidor.

El orden habitual para trabajar con ficheros es **abrir** el fichero (por ejemplo con “fopen”), se hacen los **recorridos** oportunos (usando “feof” para saber cuando se acaba el fichero por ejemplo) y **cerrarlo**.

<https://www.php.net/manual/es/book.filesystem.php>

FICHEROS



Para **abrir** el fichero utilizamos “**fopen**(“rutaFichero”, “modo”);” que devuelve FALSE si no se ha podido abrir. Los modos posibles son:

- Sólo lectura de fichero existente (“r”).
- Lectura y escritura de fichero existente (“r+”).
- Sólo escritura que si ya existe sobrescribe (“w”).
- Lectura y escritura que si ya existe sobrescribe (“w+”).
- Sólo escritura al final (“a”).
- Lectura y escritura al final (“a+”).

Se debe comprobar por si hay problemas con la apertura del fichero y después de utilizarlo se debe **cerrar** con “**fclose**”.

FICHEROS. APERTURA



Una vez abierto el fichero se genera un “**indicador**” con la posición en la que nos encontramos leyendo o escribiendo según el modo elegido.

En caso de elegir un modo “a” o “a+” se establecerá al final, mientras que si es otro modo será al inicio del fichero.

```
<?php
    $fich = fopen("fichero.txt", "r");
    if ($fich === False){
        echo "No se encuentra fichero.txt<br>";
    }else{
        echo "fichero.txt se abrió con éxito<br>";
    }
}
```

FICHEROS. RECORRIDO `fgetc`



Para realizar una **lectura** básica podemos utilizar la función “**fgetc**” que devuelve un carácter cada vez que se lee por lo que se debe utilizar junto con la función “**feof**” para saber hasta donde se puede leer.

Cuando se lee o escribe un carácter se avanza el indicador hasta la última posición leído o escrito.

La función “**feof(\$fich)**” devolverá TRUE cuando el indicador se encuentre en la última posición.

También podemos utilizar la función “**fscanf**” para leer líneas con un determinado formato.

FICHEROS. RECORRIDO fgetc



```
<?php
    $fich = fopen("fichero.txt", "r+");
    if ($fich === False){
        echo "No se encuentra el fichero o no se pudo leer<br>";
    }else{
        echo "Se procede a leer el fichero: ";
        while( !feof($fich) ){
            $car = fgetc($fich);
            echo $car;
        }
    }
    fclose($fich);
```

FICHEROS. RECORRIDO fscanf



En el caso de utilizar “fscanf” podemos utilizarla de 2 formas distintas.

Por un lado, con 2 parámetros para indicar el fichero y el formato de las líneas devuelve un array de los datos leídos. Los formatos son ya conocidos en PHP: <https://www.php.net/manual/es/function.sprintf.php>

```
$array = fscanf ($fich, $formato)
```

```
//Para líneas que contienen 4 números separados por espacios
```

```
while( !feof($fich) ){
```

```
    $numeros = fscanf($fich, "%d %d %d %d");
```

```
    echo $numeros[0] . " . " . $numeros[1] . " . " . $numeros[2] . " . " . $numeros[3];
```

```
}
```

FICHEROS. RECORRIDO fscanf



La otra forma es indicar con parámetros adicionales las variables en las que se almacenarán las lecturas.

```
$array = fscanf ($fich, $formato, $var1, $var2...)
```

Es necesario **cerrar** los ficheros con “**fclose**” para que no haya problemas.

```
//Para líneas que contienen 4 números separados por espacios  
while( !feof($fich) ){  
    $numeros = fscanf($fich, "%d %d %d %d", $n1, $n2, $n3, $n4);  
    echo $n1 . ", " . $n2 . ", " . $n3 . ", " . $n4;  
    echo "\n<br>";  
}
```

FICHEROS. FUNCIONES



Existen **más formas** de abrir y recorrer los ficheros así como otras funciones interesantes:

- “file_get_contents” que devuelve una cadena con el contenido del fichero
- “file_put_contents” que escribe en un fichero.
- “fgets” para leer líneas completas.
- “fputs” y “fwrite” para escribir en el fichero.
- “rewind” para poner el indicador de lectura en la primera posición.
- “copy” para realizar copias de un fichero.
- “is_file” e “is_dir” para saber si son fichero o directorio respectivamente.
- “rename” para renombrar.

FICHEROS. file_get/put_contents



Con estas funciones **no es necesario** abrir y cerrar los ficheros ya que se usan directamente con las rutas de los ficheros.

```
$contenido = file_get_contents("fichero.txt");  
echo "Contenido del fichero: $contenido<br>";  
$res = file_put_contents("nuevoFichero.txt", "Creado con file_put_contents");  
if($res){  
    echo "Fichero creado con éxito";  
}else{  
    echo "Error al crear el fichero";  
}
```




FICHEROS. FICHEROS XML

Para trabajar específicamente con estos ficheros podemos usar la librería **Simple-XML** que puede usarse por defecto en PHP.

1. Con “`simplexml_load_file(ruta)`” leeremos un fichero de su ruta y obtendremos un objeto del tipo “SimpleXMLElement” que podremos manipular.
2. Con un foreach sobre el objeto que se ha generado podremos realizar un recorrido.
3. Con la función “Xpath” del objeto en cuestión podemos seleccionar elementos del XML (<https://es.wikipedia.org/wiki/XPath>).



FICHEROS. Ejemplo XML

```
<?php
    $datos = simplexml_load_file("ficheroXML.xml");
    if($datos===false){
        echo "Error al leer el fichero";
    }
    foreach($datos as $valor){
        print_r($valor);
    }
    $nombres = $datos->xpath("//Nombre");
    foreach($nombres as $valor){
        print_r($valor);
        echo "<br>";
    }
}
```



FICHEROS. FICHEROS XML. FUNCIONES

Otras **funciones** que podemos usar con ficheros **XML** son:

- “schemaValidate” para validar un esquema XSD a través de un objeto DOMDocument.
- “transformToXml” para realizar una transformación también a través de un objeto DOMDocument y un XSLTProcessor.
 - Se crean 2 DOMDocument para el fichero a transformar y el XSLT.
 - Se crea un XSLTProcessor y se le carga el DOM XSLT anterior con “importStylesheet”.
 - Se usa “transformToXml” desde el procesador y sobre el DOM del fichero XML original.



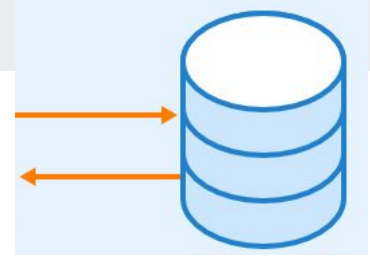
PRÁCTICA ENTREGABLE. FICHEROS

* No hace falta tener un fichero en la parte de servidor.

Realiza los siguientes apartados en un fichero PHP.

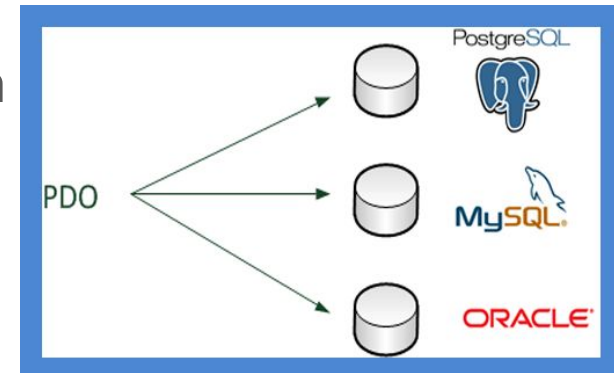
- Escritura en un nuevo fichero de los números del 1 al 10 teniendo 2 en cada línea ("1 2" en la primera línea, "3 4" en la segunda y así sucesivamente) con el comando "fputs".
- Lectura del fichero anterior con "fgets".
- Realiza una copia del fichero anterior con el comando "copy".
- Renombra el nuevo fichero con el comando "rename".

ACCESO A BASES DE DATOS



Al igual que en otros lenguajes de programación, en PHP existe la posibilidad de manejar **diversos sistemas gestores de bases de datos**.

Además, en PHP existe una manera diferente para proveer una **capa de abstracción** sobre la base de datos, los **PHP data Objects** o **PDO** que ofrecen un interfaz común sea cual sea la base de datos por lo que el código será independiente de la base de datos que se use.





ACCESO A BASES DE DATOS

Lo primero que se debe hacer para trabajar con una base de datos es realizar una **conexión** a la base de datos.

Para ello se crea un objeto de la clase PDO:

```
$bd = new PDO($conexion, $usuario, $clave, $ops);
```

Referencia: <https://www.php.net/manual/es/class.pdo.php>

- Existe la posibilidad de añadir **opciones** en un array opcional (\$ops).



ACCESO A BASES DE DATOS

- **\$conexion**: cadena que especifica qué driver hay que utilizar, la localización y el nombre de la base de datos. Para MySQL podemos utilizar “mysql:dbname=nombre;host=direccion” siendo “nombre” la base de datos a conectar en el servidor indicado por “direccion”.
- **\$usuario**: es el usuario con el que queremos conectar.
- **\$clave**: es la clave del usuario anterior.

En caso de necesitar cambiar el puerto en \$conexion, se puede realizar de 2 formas diferentes:

- Utilizando un nuevo parámetro “host=127.0.0.1;port=3306”.
- Usando la forma “127.0.0.1:3306” para la dirección.



ACCESO A BASES DE DATOS

Si tenemos éxito en la generación del objeto PDO se podrá manejar la base de datos con dicho objeto, pero en caso contrario se generará una **excepción**.

Una conexión a una base de datos se debe **cerrar** con “`$bd->close();`”.

También es posible crear **conexiones persistentes** en el tiempo con la opción “`PDO::ATTR_PERSISTENT => true`” en el array de opciones.

```
$bd = new PDO($conexion, $usuario, $clave,  
array(PDO::ATTR_PERSISTENT => true));
```




ACCESO A BASES DE DATOS

Lo primero que debemos hacer es instalar el **driver** oportuno para la base de datos que vayamos a utilizar. Podemos ver algunos de los disponibles en: <https://www.php.net/manual/es/pdo.drivers.php>, aunque no implica que no se deban instalar.

Para ver los que hay instalados podemos ejecutar:

```
print_r(PDO::getAvailableDrivers());
```

Lo normal en nuestro entorno será tener instalados los de MySQL y SQLite.

EJEMPLO DE ACCESO A MySQL

EJEMPLO

```
<?php
    $cadena_conexion = 'mysql:dbname=usuarios;host=127.0.0.1';
    $usuario = 'usuario';
    $clave = 'clave';
    print_r(PDO::getAvailableDrivers());
    try {
        echo "Vamos a intentar conectar con el usuario: ".$usuario."\n<br>";
        $bd = new PDO($cadena_conexion, $usuario, $clave);
        echo "Se ha conectado.\n<br>";
    } catch (PDOException $e) {
        echo 'Error con la base de datos: ' . $e->getMessage();
    }
}
```



EJECUCIÓN DE SQL

Una vez que hemos conectado y obtenemos el objeto PDO podemos utilizar la función “**query**” de dicho objeto para **ejecutar sentencias**. Dicha función recibe como cadena la SQL en cuestión.

```
$res = $bd->query($sql);
```

Se devuelve FALSE si hubo algún error mientras que si la ejecución es correcta en \$res obtenemos un objeto **PDOStatement** que contiene las filas del resultado recorribles a través de un bucle foreach y donde cada fila es una array de los nombres de la consulta como claves y los valores serán los encontrados en la base de datos.

EJEMPLO DE ACCESO A MySQL

EJEMPLO

Dentro del bloque try-catch y después haber obtenido un PDO correctamente podemos ejecutar el siguiente ejemplo para obtener los usuarios y sus claves.

```
$bd = new PDO($cadena_conexion, $usuario, $clave);  
echo "Se ha conectado.\n<br>\n";  
$sql = 'SELECT nombre, clave FROM users';  
$usuarios = $bd->query($sql);  
echo "Número de usuarios: " . $usuarios->rowCount() . "<br>\n";  
foreach ($usuarios as $usu) {  
    print "Nombre : " . $usu['nombre'];  
    print " Clave : " . $usu['clave'] . "<br>\n";  
}
```

SENTENCIAS PREPARADAS

Es posible “preparar” sentencias con algún valor en forma de incógnita que luego se ejecuten varias veces pasando dicho valor como argumento.

Puede hacerse **por orden**, en donde la incógnita se escribe como una ? al preparar la sentencia y luego se ejecuta pasando un array con ese número de incógnitas.

```
$res = $bd->prepare("select nombre, clave from users  
                    where nombre = ? OR nombre = ?");  
$res->execute( array("User1", "User2"));  
echo "Contando claves de usuario: " . $res->rowCount() . "<br>\n";  
foreach ($res as $usu) {  
    print $usu['nombre'] . " : " . $usu['clave'] . "<br>\n";  
}
```

SENTENCIAS PREPARADAS

O puede hacerse **por nombre**, en donde la incógnita se escribe con un nombre en particular que debe respetarse cuando se ejecuta pasando un array con parejas nombre - valor.

Hay que fijarse bien en la sintaxis y escribir bien los “:” delante del nombre.

```
$res = $bd->prepare("select nombre, clave from users  
                    where nombre = :nombre1 OR nombre = :nombre2");  
$res->execute( array(':nombre1' => "User1", ':nombre2' => "User2"));  
echo "Contando claves de usuario: " . $res->rowCount() . "<br>\n";  
foreach ($res as $usu) {  
    print $usu['nombre'] . " : " . $usu['clave'] . "<br>\n";  
}
```

INSERCIÓN, BORRADO Y ACTUALIZACIÓN

El sistema que hay que seguir para realizar cualquier **operación** de **inserción**, **borrado** o **actualización** es muy similar al descrito para las sentencias, incluso también se puede hacer a través de sentencias preparadas.

Al utilizar la función “**query**” enviaremos una sentencia SQL de inserción, borrado o actualización.

```
$insercion = "insert into users(nombre, clave) values('User4', '1111');";  
$res = $bd->query($insercion);  
if($res) {  
    echo "La inserción es correcta y se han insertado " . $res->rowCount() . " filas.<br>\n";  
}else {  
    print_r( $bd -> errorinfo());  
}
```

INSERCIÓN, BORRADO Y ACTUALIZACIÓN

En todos los casos, la opción **autocommit** debe estar activada (SELECT @@AUTOCOMMIT;)

```
$actualizacion = "update users set clave = '1234' where nombre = 'User4'";
$res = $bd->query($actualizacion);
if($res){
    echo "La actualización es correcta y se han actualizado " . $res->rowCount() . " filas.<br>\n";
}else {
    print_r( $bd -> errorinfo());
}
$borrado = "delete from users where nombre = 'User4'";
$res = $bd->query($borrado);
if($res){
    echo "El borrado es correcto y se han borrado " . $res->rowCount() . " filas.<br>\n";
}else {
    print_r( $bd -> errorinfo());
}
```


TRANSACCIONES

El proceso de las **transacciones** se debe realizar más conscientemente que nunca desde PHP, para ello se deben usar las siguientes funciones del objeto PDO creado con la conexión:

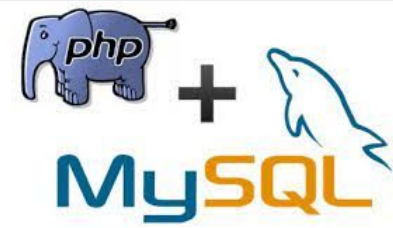
- “**beginTransaction()**”: que marca el comienzo de la transacción.
- “**commit()**”: que da por finalizada la transacción y por tanto para a salvarse el estado de la base de datos.
- “**rollBack()**”: se cancela la transacción y se deben deshacer los cambios.

Todas las operaciones realizadas en ese objeto PDO tras realizar el “beginTransaction” no serán guardadas en la base de datos hasta recibir el “commit” correspondiente.

EJEMPLO DE TRANSACCIONES

EJEMPLO

```
$bd->beginTransaction();
$insertar = "insert into users(nombre, clave) values('User5', '1111')";
$res = $bd->query($insertar);
//Si ponemos true se termina la transacción, y si es false se hace rollback
$continuar=false;
if($continuar){
    // Si todo va bien se consolida la transacción
    echo "Se termina la transacción<br>\n";
    $bd->commit();
}else{
    echo "Transacción anulada<br>\n";
    $bd->rollback(); //Se deshace la transacción
}
```



ACCESO A BASES DE DATOS CON MySQLi

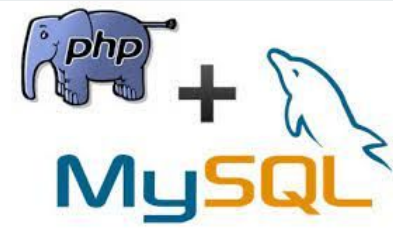
PHP ofrece además la posibilidad de conectarnos a una base de datos MySQL mediante el **driver específico MySQLi**.

<https://www.php.net/manual/es/book.mysql.php>

La conexión es parecida mediante la siguiente instrucción:

```
$mysqli = new mysqli($host, $usuario, $clave, $bd);
```

Los parámetros habituales son host, usuario, clave y base de datos e incluso podemos añadir otro parámetro para el puerto como se puede ver en la referencia <https://www.php.net/manual/es/class.mysql.php>



ACCESO A BASES DE DATOS CON MySQLi

Una vez realizada la conexión el funcionamiento es muy parecido al seguido con PDO.

- Mediante la función “**query**” ejecutamos una sentencia SQL (ya sea de consulta, de inserción, etc.) en el objeto mysqli creado durante la conexión. Esta función devuelve “false” si hay algún problema o el resultado de la instrucción si todo ha ido bien.
- En las consultas “SELECT” haremos uso de las siguientes funciones:
 - “`$res->data_seek(0);`” para colocarnos al inicio y poder recorrer los resultados.
 - “`$res->fetch_assoc()`” que devuelve la siguiente fila de la respuesta.

EJEMPLO CON MySQLi

EJEMPLO

```
<?php
```

```
//Creamos la conexión
```

```
$mysqli = new mysqli("localhost", "root", "1234", "usuarios", 3306);
```

```
//Si da error sacamos un mensaje y no se ejecuta nada más
```

```
if ($mysqli->connect_errno) {
```

```
    echo "Fallo al conectar a MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
```

```
}else{
```

```
    //Si no da error ejecutamos el resto
```

```
    echo "Se ha conectado: " . $mysqli->host_info . "<br>\n";
```

```
    //En la siguiente línea mediante 3 sentencias creamos e insertamos datos en una nueva tabla
```

```
    if (!$mysqli->query("DROP TABLE IF EXISTS usuarios2")) ||
```

```
        !$mysqli->query("CREATE TABLE usuarios2(dni INT, nombre VARCHAR(20))") ||
```

```
        !$mysqli->query("INSERT INTO usuarios2(dni, nombre) VALUES (1,'User1'), (2,'User2')")) {
```

```
        //Si alguna de ellas devuelve false, error de ejecución, entonces sacaremos este mensaje
```

```
        echo "Falló la creación de la tabla: (" . $mysqli->errno . ") " . $mysqli->error;
```

```
}
```

EJEMPLO CON MySQLi

EJEMPLO

```
else{
    //Se lanza una sentencia para leer todos los datos introducidos.
    $res = $mysqli->query("SELECT dni FROM usuarios2 ORDER BY dni ASC");
    echo "Resultados: <br>\n";
    //Se coloca el apuntador en la fila 0
    $res->data_seek(0);
    //El bucle recoge la siguiente fila con el fetch
    while ($fila = $res->fetch_assoc()) {
        echo " DNI = " . $fila['dni'] . "<br>\n";
    }
}
```

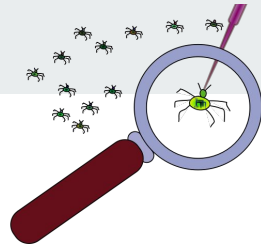


PRÁCTICA ENTREGABLE. ACCESO A BASE DE DATOS

Realiza una aplicación PHP que acceda a una base de datos con una tabla "personas" con datos "dni", "nombre", "apellidos" y "fecha de nacimiento"

En dicha aplicación se debe dar las siguientes posibilidades:

- Mediante un formulario enviar datos que serán introducidos en la tabla "personas" anterior.
- Hacer un volcado de los datos de la tabla "personas" en un fichero.



PRUEBAS Y DEPURACIÓN

El último apartado de este tema hace referencia a las **pruebas** y a la **depuración**.

Para el apartado de pruebas nos centraremos en las pruebas unitarias (de funciones o clases de manera individual) y daremos un vistazo rápido a **PHPUnit**, similar a JUnit para Java.

En el lado de la depuración haremos un vistazo rápido a 2 opciones relativamente fáciles de probar durante nuestros desarrollos.

PRUEBAS Y DEPURACIÓN

Al igual que se ha hecho con Java, podemos realizar pruebas unitarias sobre nuestro proyecto PHP mediante “**PHPUnit**”

<https://phpunit.de/getting-started/phpunit-10.html>

La última **versión**, versión 10, permite hacer una instalación global en nuestro sistema y también permite hacer instalaciones **proyectos**.

En la siguiente diapositiva vamos a ver una breve instalación del segundo tipo pero a través de composer, para lo cuál lo tendremos que tener instalado en nuestro ordenador.

PRUEBAS Y DEPURACIÓN

1. Desde Windows (o desde Linux que el proceso es similar) nos vamos a la consola y creamos una carpeta para situarnos en ella.
2. Ejecutamos “composer init” para crear el proyecto y vamos añadiendo los datos de proyecto que queramos.
3. Cuando lleguemos a la pregunta (cuidado que hay 2 parecidas y es la 2ª) “Would you like to define your **dev** dependencies (**require-dev**) interactively [yes]?” presionamos enter, y escribimos “phpunit/phpunit” para instalar PHPUnit. (El resto de pasos básicamente es dar a ENTER y también generará el directorio “src” para nuestros códigos).
4. Con esto se genera la estructura necesaria para ejecutar PHPUnit y el composer.json con toda la configuración (página siguiente).

PRUEBAS Y DEPURACIÓN

El composer.json podría ser:

```
{
    "name": "luis/prueba-test1",
    "require-dev": {
        "phpunit/phpunit": "^10.1"
    },
    "autoload": {
        "psr-4": {
            "Luis\\PruebaTest1\\": "src/"
        }
    },
    "require": {}
}
```

Además podemos comprobar si se ha instalado, situándonos en el directorio “vendor/bin” y ejecutando “phpunit --version”.

```
C:\PHPUnit\pruebaTest1\vendor\bin>phpunit --version
PHPUnit 10.1.3 by Sebastian Bergmann and contributors.
```

PRUEBAS Y DEPURACIÓN

Ya tendremos creada la **carpeta** “**src**” donde vamos a poder dejar nuestros ficheros php. En nuestro ejemplo vamos a generar una clase en el fichero “**Ej.php**” que dejaremos en dicho directorio. El código está en la página siguiente y básicamente es una clase con un atributo, un constructor y un método. (Cuidado con el namespace)

Para las pruebas nos falta crear una nueva **carpeta** “**tests**” y allí, para probar la clase “Ej”, creamos el fichero “**EjTest.php**”, con el código que hay en las páginas siguientes. Vemos que hay 2 métodos, uno para probar el constructor (y el tipo de la variable “\$nombre”) y otro para probar el método de la clase “Ej” llamado “decirHola” (en el que se comprueba que devuelve un String y que contiene el nombre asignado). (Cuidado con el namespace)

PRUEBAS Y DEPURACIÓN

//En el directorio “src” tendremos la siguiente clase “Ej.php”

```
<?php
    namespace Luis\PruebaTest1;
    class Ej {
        public string $nombre;
        public function __construct(string $name) {
            $this->nombre = $name;
        }
        public function decirHola(): string {
            return "Hola " . $this->nombre . ".";
        }
    }
```

PRUEBAS Y DEPURACIÓN

//En el directorio “tests” tendremos la siguiente clase “EjTest.php”

<?php

```
namespace Luis\PruebaTest1;
use PHPUnit\Framework\TestCase;
final class EjTest extends TestCase{
    public function testClassConstructor(){
        $ej = new Ej('Luis');
        $this->assertSame('Luis', $ej->nombre);
    }
    public function testDecirHola(){
        $ej = new Ej('Luis');
        $this->assertIsString($ej->decirHola());
        $this->assertStringContainsStringIgnoringCase('Luis', $ej->decirHola());
    }
}
```

PRUEBAS Y DEPURACIÓN

En el ejemplo anterior podemos ver las siguientes cosas:

- Habitualmente tendremos una **carpeta “tests”** para las pruebas de nuestro código que estará en la **carpeta “src”**.
- Normalmente las pruebas serán clases **herederas** de la clase `PHPUnit\Framework\TestCase`.
- En la prueba, cada **prueba individual** es un **método** público nombrado con el prefijo “test”. Por ejemplo, para probar el método “decirHola” en la clase “Ej” el método sería “testDecirHola”.
- Dentro del método de prueba, “testDecirHola”, se usa el método de PHPUnit “**assertSame**” para comprobar que devuelve el valor esperado.

PRUEBAS Y DEPURACIÓN

Ahora ya podemos **ejecutar** las **pruebas** con “phpunit carpetaTests”:

```
C:\PHPUnit\pruebaTest1\vendor\bin>phpunit ../../tests
PHPUnit 10.1.3 by Sebastian Bergmann and contributors.

Runtime:      PHP 8.1.6

..                                                    2 / 2 (100%)

Time: 00:00.011, Memory: 6.00 MB

OK (2 tests, 3 assertions)
```

En esta imagen vemos como se han pasado 2 casos de prueba con 3 aserciones (pruebas reales), la duración de la prueba y la memoria utilizada.

PRUEBAS Y DEPURACIÓN

Ya podemos crear más pruebas utilizando una serie de métodos llamados “**assert**” de la clase “**TestCase**” que devuelven un valor booleano (true/false) si la prueba ha sido exitosa o no. Algunos de ellos son los siguientes aunque hay muchos más:

- “**assertSame**”: para comprobar que son del mismo tipo.
- “**assertEquals**”: para comprobar que el valor es igual
- “**assertRegExp**”: para comprobar una expresión regular.
- “**assertStringContainsString**”: para comprobar que contiene un String.
- “**assertArrayHasKey**”: para comprobar la existencia dentro de un array.
- “**assertJsonStringEqualsJsonString**”: para comprobar el Json devuelto.

Además se pueden generar 2 métodos, “**setUp()**” y “**tearDown()**” que se ejecutan antes y después de cualquier prueba que pueden englobar código repetido.



PRÁCTICA ENTREGABLE. PRUEBAS UNITARIAS

Realiza una clase PHP que tenga:

- 2 atributos, los que tu quieras.
- 2 constructores, con y sin los atributos.
- 2 métodos que mezclen los atributos anteriores.

Añade una clase de pruebas para probar todos los anteriores, tanto los tipos como los contenidos.

PRUEBAS Y DEPURACIÓN



En cuanto a la depuración, PHP viene con el depurador interactivo **phpdbg**, aunque se pueden utilizar depuradores externos.

<https://www.php.net/manual/es/book.phpdbg.php>

Con este depurador podemos acceder mediante consola a la interfaz para gestionar nuestro php, añadir puntos de parada, ver sus opcodes, etc.

```
C:\xampp\htdocs>phpdbg debug.php
[Welcome to phpdbg, the interactive PHP debugger, v8.1.6]
To get help using phpdbg type "help" and press enter
[Please report bugs to <http://bugs.php.net/report.php>]
[Successful compilation of C:\xampp\htdocs\debug.php]
prompt> help

phpdbg is a lightweight, powerful and easy to use debugging platform for PHP5.4+
It supports the following commands:

Information
list      list PHP source
info      displays information on the debug session
print     show opcodes
frame     select a stack frame and print a stack frame summary
generator show active generators or select a generator frame
back      shows the current backtrace
help      provide help on a topic

Starting and Stopping Execution
```

PRUEBAS Y DEPURACIÓN

También podríamos ejecutar desde la propia consola para no tener que entrar en el depurador interactivo. Podemos ver los opcodes (códigos básicos de las operaciones) de un programa mediante el comando “phpdbg -p fichero.php”.

```
C:\xampp\htdocs>phpdbg -p debug.php

$_main:
; (lines=22, args=0, vars=2, tmps=4)
; C:\xampp\htdocs\debug.php:1-9
L0002 0000 EXT_STMT
L0002 0001 ASSIGN CV0($str) string("ola")
L0007 0002 EXT_STMT
L0007 0003 INIT_FCALL 1 96 string("str_split")
L0007 0004 SEND_VAR CV0($str) 1
L0007 0005 V3 = DO_FCALL
L0007 0006 V4 = FE_RESET_R V3 0020
L0007 0007 FE_FETCH_R V4 CV1($char) 0020
L0008 0008 EXT_STMT
L0008 0009 ECHO CV1($char)
L0008 0010 EXT_STMT
L0008 0011 ECHO string(": ")
L0008 0012 EXT_STMT
L0008 0013 INIT_FCALL 1 112 string("ret_ord")
L0008 0014 SEND_VAR CV1($char) 1
L0008 0015 V5 = DO_FCALL
L0008 0016 ECHO V5
L0008 0017 EXT_STMT
L0008 0018 ECHO string("<br>")
L0007 0019 JMP 0007
L0007 0020 FE_FREE V4
L0009 0021 RETURN int(1)
[Script ended normally]
```

PRUEBAS Y DEPURACIÓN



En los dos casos anteriores el **programa php** es el siguiente:

```
<?php
    $str = "hola";
    function devuelveNumero( $c ){
        return ord( $c );
    }

    foreach ( str_split( $str ) as $char ){
        echo $char, ": ", devuelveNumero( $char ), "<br>\n";
    }
```



PRUEBAS Y DEPURACIÓN

Otra herramienta conocida para hacer el debug de aplicaciones PHP se llama **Xdebug**, aunque tal vez sea poco amigable para el programador y sea mejor instalarla junto con algún framework o IDE que ya lo lleve integrado.

Para la instalación manual se deben seguir los siguientes pasos:

- Acceder al “phpInfo()” del servidor que mostrará información de PHP.
- Seleccionaremos todo el texto, lo copiamos (ctrl+c) y presionamos el botón “Analyse my phpinfo() output”.
- Accedemos a <https://xdebug.org/wizard> y lo copiamos en el cuadro.
- Ahora ya solo queda seguir las instrucciones que se nos ofrecen.



PRUEBAS Y DEPURACIÓN

En nuestro caso las instrucciones son sencillas, básicamente descargar un fichero “.dll”, guardarlo en una carpeta de nuestro servidor xampp y actualizar el fichero de configuración inicial antes de reiniciar el servidor.

Instructions

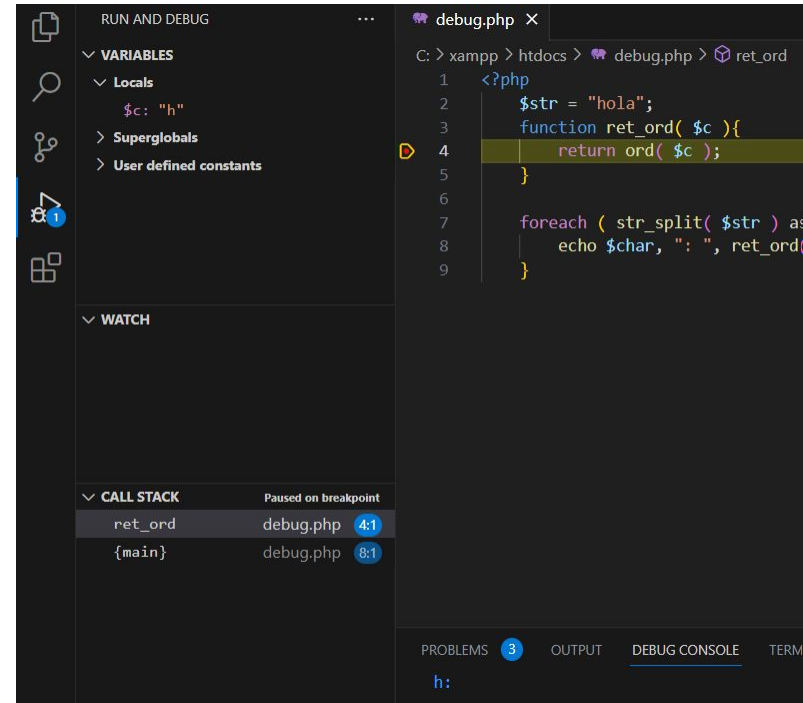
1. Download `php_xdebug-3.2.1-8.1-vs16-x86_64.dll`
2. Move the downloaded file to `C:\xampp\php\ext`, and rename it to `php_xdebug.dll`
3. Update `C:\xampp\php\php.ini` to have the line:
`zend_extension = xdebug`
4. Restart the Apache Webserver

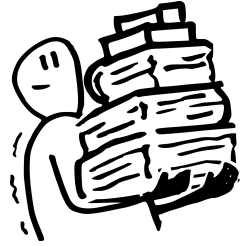


PRUEBAS Y DEPURACIÓN

El siguiente paso es añadir alguna extensión a nuestro editor de código, por ejemplo “Visual Studio Code”:

- Instalamos la **extensión** de «PHP Debug» en el editor.
- Es posible que tengamos que reiniciar “Visual Code” y ya tendremos disponibles las herramientas de depuración.





WEBGRAFÍA

- <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>
- <https://www.php.net/manual/es/reserved.variables.files.php>
- <https://github.com/PHPMailer/PHPMailer>
- <https://www.php.net/manual/es/book.filesystem.php>
- <https://es.wikipedia.org/wiki/XPath>
- <https://www.php.net/manual/es/class.pdo.php>
- <https://www.php.net/manual/es/book.mysql.php>
- <https://phpunit.de/>
- <https://docs.phpunit.de/en/10.0/assertions.html>
- <https://cosasdedevs.com/posts/testar-aplicaciones-php-phpunit/>
- <https://www.php.net/manual/es/book.phpdbg.php>
- <https://xdebug.org>
- Imágenes CC: <https://pixabay.com/>, <https://freeicons.io> y <https://www.pexels.com>



PROYECTO



Realiza una aplicación para realizar valoraciones de películas.

Todo el mundo podrá acceder a ver las valoraciones pero para subir una valoración (nota y mensaje sobre la película) se debe estar logueado.

- Base de datos que soporte el sistema:
 - Peliculas(Título, Año, Duración, idDirector).
 - Directores (id, nombre).
 - Valoraciones (idUsuario, idPelicula, valorNumerico, comentario).
 - Usuarios (correoElectrónico, clave, nombre)

Todos los que empiezan por id se suponen claves foráneas aunque podéis modificar la estructura a vuestro parecer según os convenga.

PROYECTO



Página de bienvenida. Opciones disponibles:

- **Listar películas** que haya en el sistema (pueden estar por defecto en la página de bienvenida o listarse en otra).
- Información de la película. **Página diferente** que reciba un id de **película** y saque los datos de la BD para mostrar una plantilla genérica con Título, Año, Duración y Director además de las distintas valoraciones que se han hecho.
- **Listar directores** que haya en el sistema (pueden estar por defecto en la página de bienvenida o listarse en otra) y al acceder a uno se deben ver todas sus películas (otra **página diferente** que recibe el id de **director** y accede a la BD).
- En cada página debe haber una forma de **volver** a la página de bienvenida.
- Opción para iniciar **login** (se valida directamente sobre la base de datos o con Google).
- Debe aparecer un **contador** del número de accesos que ha tenido cada página, para ello se debe guardar en un **fichero en servidor**.

PROYECTO



Página de usuario registrado:

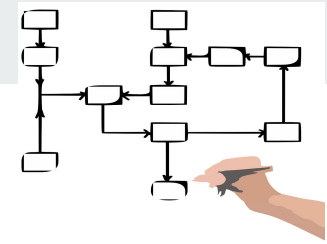
- En un lugar preferente debe aparecer la **última valoración** insertada desde el cliente en cuestión (**cookie**).
- **Formulario** para dar de **alta** una nueva **película** con los datos necesarios. Cuando se hace, se envía un **correo** a todos los usuarios registrados informando del nuevo alta.
- **Formulario** para crear una nueva **valoración** en una **página diferente**. Para llegar a ella puede haber un selector de películas en la página de usuario o se puede llegar desde la película en particular. De no enviar alguno de los datos (valor numérico o comentario) se introducirá algún valor por defecto.
- Entre las páginas se debe **mantener** al **usuario**, por ejemplo con una **sesión**.
- Se debe **mantener** la **sesión entre accesos** (**cookie**) hasta que el usuario salga de la sesión (eliminar).

PROYECTO. ENTREGA

Se deben entregar los siguientes elementos:

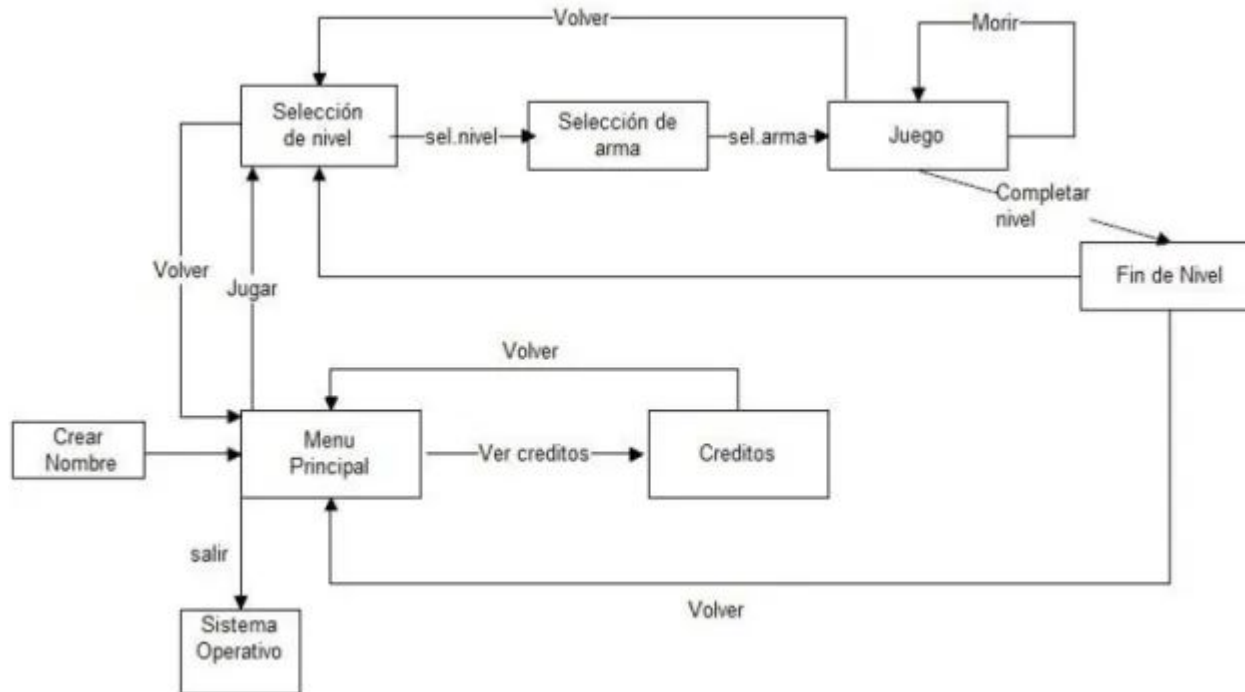
- **Script** para la creación y carga inicial de la base de datos.
- **Comprimido** con los ficheros PHP, HTML, etc. oportunos siguiendo la estructura del proyecto (si hubiera carpetas) listo para descomprimir en un servidor web con PHP (por ejemplo httdocs de Xampp) y que funcionase.
- **Fichero** de instrucciones sobre el funcionamiento básico (por ejemplo en Word) con el flujo de datos (variables \$_... usadas) entre los ficheros y las pantallas que saldrán al usuario (no manual de instrucciones con imágenes), sólo explicación de lo que se ha diseñado. (Tabla con fichero, descripción, parámetros y posibles redirecciones).
 - Por ejemplo “*Habrà un enlace ‘Valorar’ que enviarà a una página que recibe por GET la película y abre el formulario de valoraciones*”.



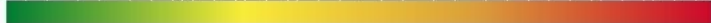


PROYECTO. ENTREGA

Éste podría ser un ejemplo de navegación entre páginas con el evento correspondiente:




PROYECTO. RÚBRICA

| | | | | | | | |
|-------------------------------------------------------------------------------------|---|--------------------------|---|------------------------|---|---------------------|---|
| Excellent | 4 | Good | 3 | Satisfactory | 2 | Needs to impr.. | 1 |
|  | | | | | | | |
| Student use a clear | | Student's voice is clear | | Student's voice is low | | Student mumbles and | |

- Script base de datos: 0,5 puntos.
- Página bienvenida: correctamente configurada 0,5 puntos.
- Listado de películas: ya sea en la misma página o en otra 0,5 puntos.
- Página de información de película: si se muestran los datos 0,5 puntos y si tiene las valoraciones 1 punto.
- Mostrar página de directores con todas sus películas: 0,5 puntos.
- Página de películas de un director: 0,5 puntos.
- Hacer el login: 1 punto.
- Contador del número de accesos (ficheros): 0,5 puntos.
- Última valoración (cookie): 0,5 puntos.

PROYECTO. RÚBRICA

| | | | | | | | |
|-------------------------------------------------------------------------------------|---|--------------------------|---|------------------------|---|---------------------|---|
| Excellent | 4 | Good | 3 | Satisfactory | 2 | Needs to impr.. | 1 |
|  | | | | | | | |
| Student use a clear | | Student's voice is clear | | Student's voice is low | | Student mumbles and | |

- Alta películas y correo cuando se da de alta películas: 0,5 si sólo se da de alta y 1 punto si se envían los correos.
- Alta de nueva valoración: 0,5 puntos.
- Mantenimiento de usuario entre páginas (Sesión): 0,5 puntos.
- Mantenimiento de usuario entre accesos (Cookies): 0,5 puntos.
- Eliminar sesión de usuario cookie y sesión: 0,5 puntos.
- Documento explicativo: 0,5 o 1 punto según documento.
- Si no hay comentarios y explicaciones: -1 punto.



PROYECTO ALTERNATIVO

Realiza una aplicación para realizar compras y valorarlas.

Todo el mundo podrá acceder a ver las valoraciones pero para subir una valoración (nota y mensaje) se debe estar logueado y haberlo comprado.

- Base de datos que soporte el sistema:
 - Productos(Nombre, Precio).
 - Compras (id, idCliente, idProducto, fecha).
 - Valoraciones (idCliente, idProducto, valorNumerico, comentario).
 - Cliente (correoElectrónico, nombre, clave).

Todos los que empiezan por id se suponen claves foráneas aunque podéis modificar la estructura a vuestro parecer según os convenga.