

Desarrollo Web en entorno servidor

Unidad 3:

Aplicaciones Web en PHP

Formularios

Tabla de contenido

1.	Depurador, errores y excepciones.	2
1.1.	Instalación de Xdebug	2
1.2.	Errores	6
1.3.	Excepciones	7
2.	Formularios en HTML	9
2.1.	Características de GET y POST	9
2.1.1.	Ejemplo del método GET	9
2.1.2.	¿Cuándo usar GET?	11
2.1.3.	Ejemplo del método POST	11
2.1.4.	¿Cuándo usar POST?	12
2.2.	Validación de datos en el mismo formulario	12

1. Depurador, errores y excepciones.

A lo largo de esta sección vamos a configurar nuestro IDE para trabajar con un depurador del lenguaje PHP, así como comentaremos los diferentes tipos de errores generados por PHP y cómo podemos manejarlos.

1.1. Instalación de Xdebug

A la hora de buscar y detectar errores en el código que producimos es fundamental el uso de algún depurador que nos facilite esta tarea.

Uno de los depuradores PHP más extendidos para Visual Studio Code es Xdebug. Para instalarlo, buscamos en la pestaña de extensiones de Visual Studio, PHP Debug.

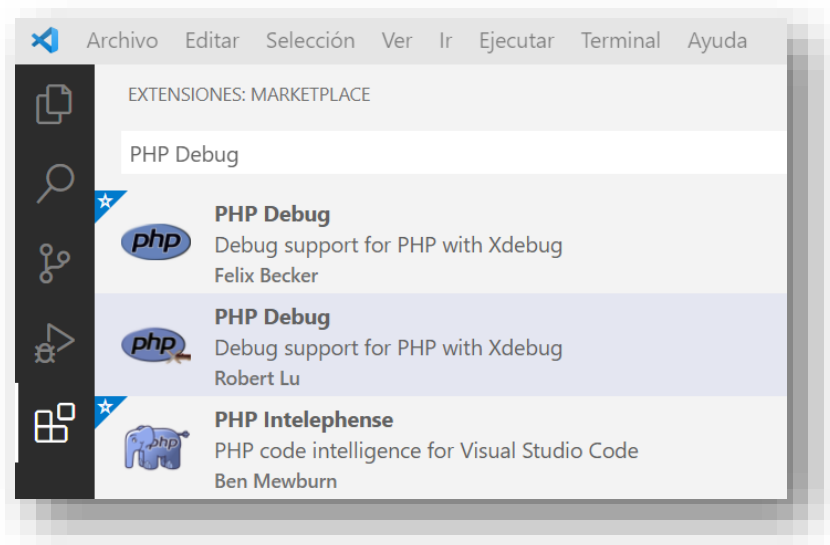


Figura 1. – Pestaña de extensiones

Seleccionamos la versión de Felix Becker y hacemos clic en instalar

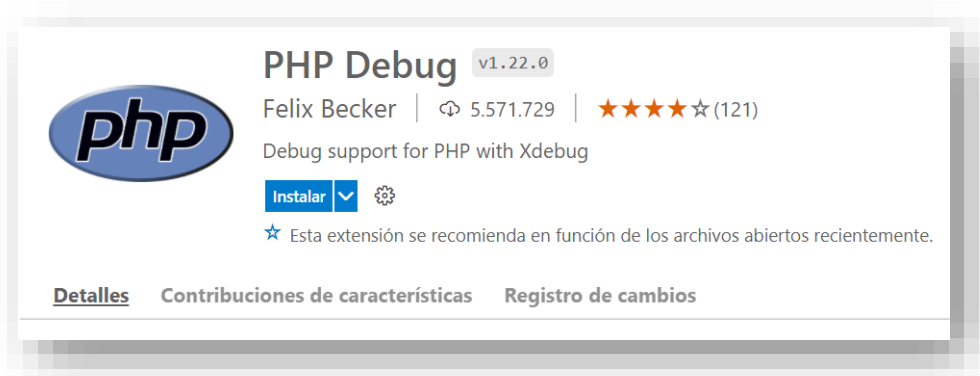


Figura 2 – Extensión Xdebug para PHP

Una vez instalado hay que configurar adecuadamente nuestro IDE para poder utilizarlo.

Para ello, accedemos a la información de nuestra versión PHP instalada haciendo clic en el siguiente enlace, <http://localhost/dashboard/phpinfo.php>

Seleccionamos el contenido completo de la página con **Ctrl+A**, lo cortamos y lo pegamos en el cuadro de texto destinado para tal fin en el siguiente enlace: [Xdebug: Support — Tailored Installation Instructions](#)

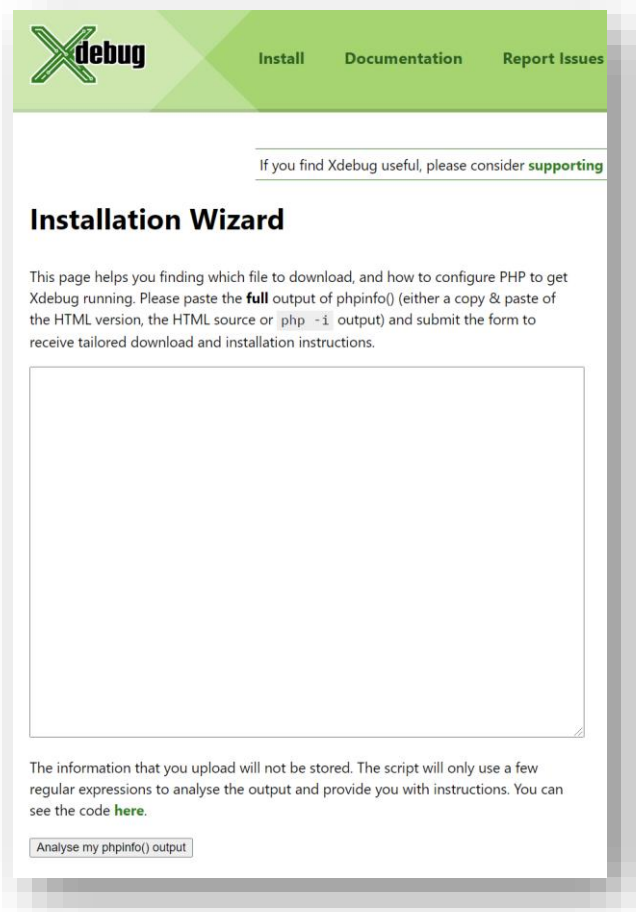


Figura 3 – Cuadro de ayuda a la configuración de Xdebug

Una vez copiado, hacemos clic en el botón inferior izquierdo para enviar la información y nos aparecerán las instrucciones que hemos de seguir para terminar la configuración.

Las instrucciones habituales son:

1. Descargar [php_xdebug-3.1.1-8.0-vs16-x86_64.dll](#)
2. Mover el archive descargado a la carpeta `C:\xampp\php\ext`, y renombrarlo como `php_xdebug.dll`
3. Actualizar el archivo `C:\xampp\php\php.ini`, abriéndolo con el bloc de notas, añadiendo, al final del mismo la siguientes líneas:
`zend_extension = xdebug`
`xdebug.mode = debug`
`xdebug.start_with_request = yes`
4. Reiniciar el servidor de Apache y Visual Studio.

Por último, abrimos Visual Studio y accedemos a la paleta de comandos pulsando F1. Escribimos “open settings” y hacemos clic en la opción “Open Settings (JSON)”.



Figura 4 – Archivo settings.json

Añadimos las dos líneas siguientes, teniendo en cuenta que la última línea no termina en coma.

```
"php.validate.executablePath": "C:\\xampp\\php\\php.exe",
"php.debug.executablePath": "C:\\xampp\\php\\php.exe",
```

Guardamos la nueva configuración y reiniciamos Visual Studio.

Por último, accedemos a la pestaña “Ejecución y depuración” y hacemos clic en “cree un archivo launch.json”, guardamos el archivo “launch.json” que se abre por defecto y hemos terminado con la configuración de Visual Studio para trabajar con Xdebug.

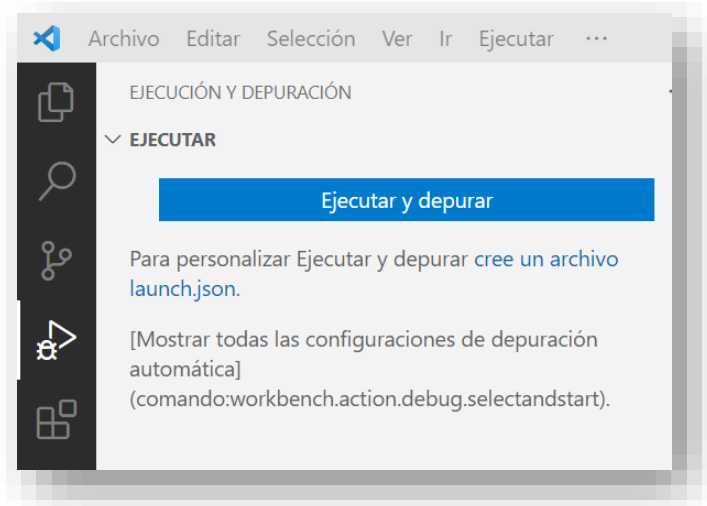


Figura 5 - Pestaña ejecución y depuración

Para ver cómo se trabaja con el depurador, abrimos cualquier script previo que tuviéramos, por ejemplo “break_anidado.php” y accedemos de nuevo a la pestaña “Ejecución y depuración”

Si hacemos clic a la izquierda de los números que marcan las líneas de código vemos que aparece un punto rojo, en el ejemplo están marcadas las líneas 5 y 13. Estos puntos, llamados puntos de interrupción, indican los puntos donde la ejecución del script se va a detener.

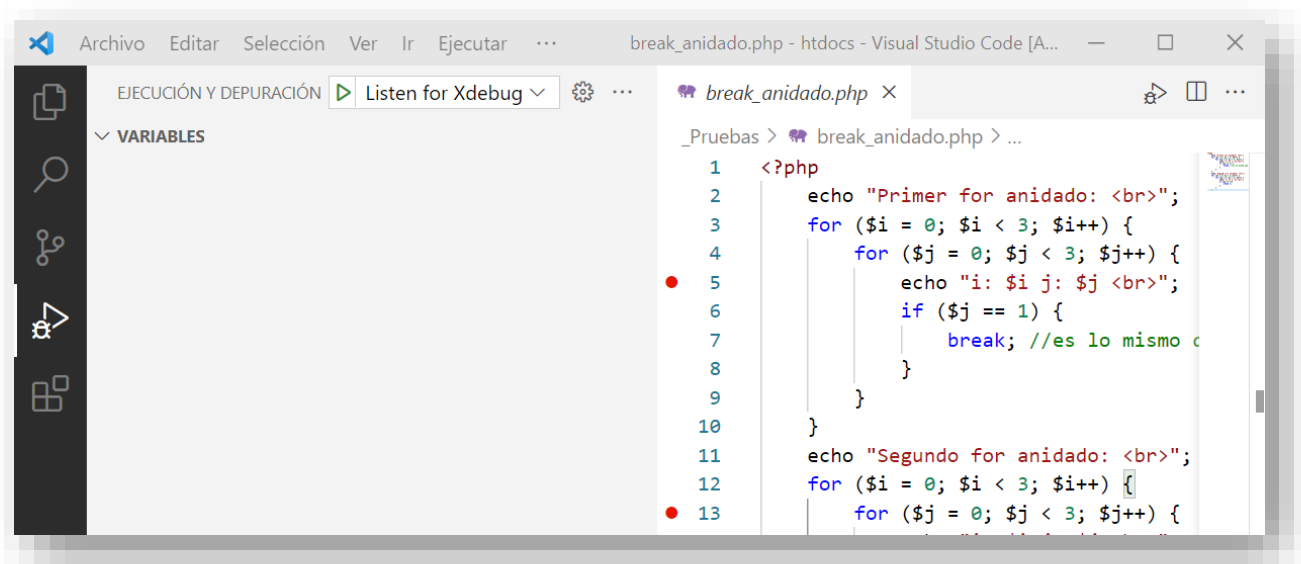


Figura 6 – Puntos de interrupción

Para iniciar el depurador, hacemos clic en el icono verde a la izquierda de “Listen for Xdebug”. Nos aparecerá una barra con varios iconos que nos permiten controlar cómo nos movemos entre los diferentes puntos de interrupción seleccionados cuando lancemos el script.

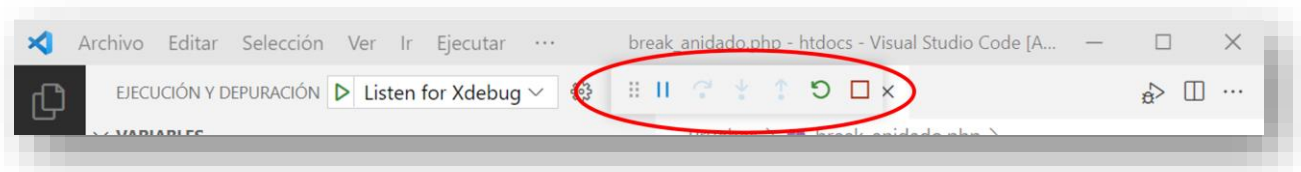


Figura 7 – Barra del depurador

Por último, abrimos el script “break_anidado.php” desde el navegador y volvemos a Visual Studio.

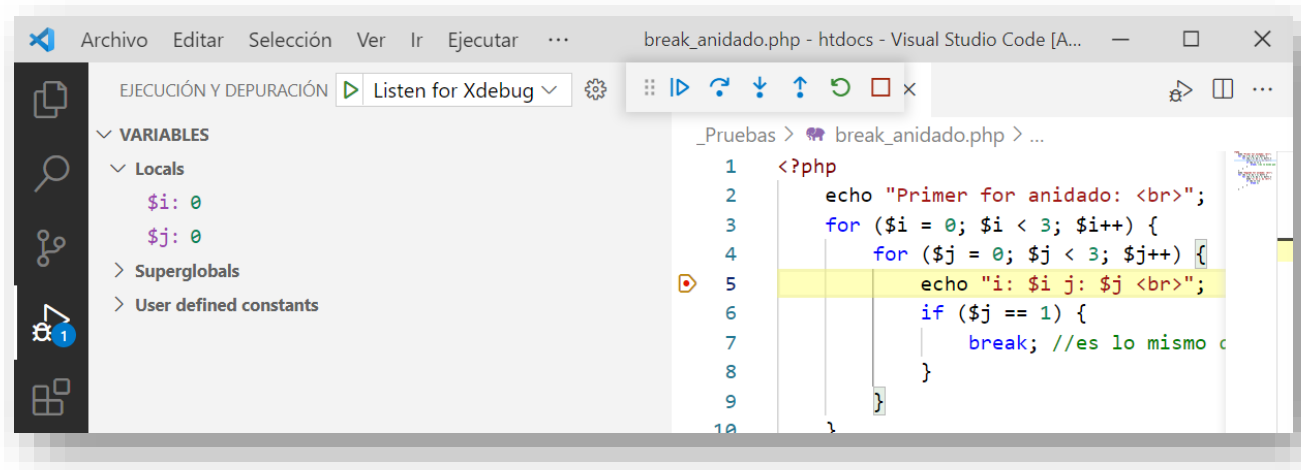


Figura 8 – Script detenido en el primer punto de interrupción.

Vemos cómo el script se ha detenido en la línea marcada y podemos ver, entre otras cosas el estado actual de las variables locales y superglobales. Podemos controlar la ejecución del mismo desde la barra del depurador para ver cómo va evolucionando.

1.2. Errores

Sabemos que ante determinadas condiciones (por ejemplo, utilizar una variable no inicializada) PHP genera un error. Algunos de estos errores detienen la ejecución del script y otros simplemente nos son notificados.

En PHP hay diferentes tipos de errores, cada uno asociado con un número y una constante predefinida. Estos errores son:

Código	Constante	Descripción
1	E_ERROR	Error fatal en tiempo de ejecución. El script se detiene
2	E_WARNING	Advertencia en tiempo de ejecución. El script no se detiene
4	E_PARSE	Error de sintaxis al compilar
8	E_NOTICE	Notificación. Puede indicar error o no
16	E_CORE_ERROR	Error fatal al iniciar PHP
32	E_CORE_WARNING	Advertencia al iniciar PHP
64	E_COMPILE_ERROR	Error fatal al compilar
128	E_COMPILE_WARNING	Advertencia fatal al compilar
256	E_USER_ERROR	Error generado por el usuario
512	E_USER_WARNING	Advertencia generada por el usuario
1024	E_USER_NOTICE	Notificación generada por el usuario
2048	E_STRICT	Sugerencias para mejorar la portabilidad
4096	E_RECOVERABLE_ERROR	Error fatal capturable
8192	E_DEPRECATED	Advertencia de código obsoleto
16384	E_USER_DEPRECATED	Como la anterior, generada por el usuario
32767	E_ALL	Todos los errores

Figura 9 - Tipos de error

Se puede controlar cómo se comporta PHP antes los errores mediante tres directivas del fichero php.ini:

- **error_reporting**: indica qué errores deben reportarse. Lo normal es utilizar E_ALL, es decir, todos. La configuración por defecto en php.ini es:

error_reporting=E_ALL & ~E_DEPRECATED & ~E_STRICT

- **display_errors**: señala si los mensajes de error deben aparecer en la salida del script. Esta opción es apropiada durante el desarrollo, pero no en producción. La configuración por defecto en php.ini es:

display_errors=On

- **log_errors**: indica si los mensajes de error deben almacenarse en un fichero. Es especial mente útil en producción, cuando no se muestran los errores en la salida. La configuración por defecto en php.ini es:

log_errors =On

- **error_log**: si la directiva anterior está activada, es la ruta en la que se guardan los mensajes de error. La configuración por defecto en php.ini es:

error_log="C:\xampp\php\logs\php_error_log"

Si necesitamos modificar los errores que se han de reportar en tiempo de ejecución, podemos hacer uso de la función [error_reporting\(\)](#), la cual permite cambiar el valor de la directiva `error_reporting` en tiempo de ejecución.

También es posible definir una función propia para que se encargue de los errores utilizando `set_error_handler()`. La función que se ocupe de los errores tendrá que tener la siguiente cabecera:

```
bool handler (int $errno, string $errstr [, string $errfile [, int $errline [, array $errcontext ]]]);
```

El ejemplo `error_handler.php` muestra cómo utilizar `set_error_handler()` para manejar los errores con una función propia.

```
1:  <?php
2:      function manejadorErrores($errno, $str, $file, $line){
3:          echo "Ocurrió el error: $errno <br>";
4:          echo "Nombre del error: $str <br>";
5:          echo "Archivo del error: $file <br>";
6:          echo "Línea del error: $line <br>";
7:      }
8:      set_error_handler("manejadorErrores");
9:      $a = $b; // causa error, $b no está inicializada
```

La salida será:

Ocurrió el error: 2

Nombre del error: Undefined variable \$b

Archivo del error: C:\xampp\htdocs\DWES\cap2\error_handler.php

Línea del error: 10

Muchos más ejemplos están disponibles en la documentación de PHP, [set_error_handler\(\)](#).

1.3. Excepciones

Otra opción para indicar un error es lanzar una excepción. Para controlar las excepciones se utilizan bloques `try/catch/finally`. Cuando se lanza una excepción y no es capturada por un bloque `catch`, la ejecución del programa se detiene. Si es capturada, se ejecuta el código del bloque correspondiente.

Para capturar una excepción se introduce la instrucción que puede causarla dentro de un bloque `try` y se añade el bloque `catch` correspondiente. Se puede añadir un bloque `finally`, que se ejecuta después del `try/catch`, haya habido excepción o no.

```
try{
    instrucciones;
}catch(Exception e){
    instrucciones;
}finally{
    instrucciones;
}
```

En el ejemplo `excepciones.php` se utiliza una función que recibe dos argumentos y lanza una excepción si el segundo es cero. Para lanzar una excepción se utiliza [throw](#), que recibe como argumento un objeto de clase [Exception](#) o de alguna subclase.

Cuando una excepción es lanzada, el código siguiente a la declaración no será ejecutado, y PHP intentará encontrar el primer bloque `catch` coincidente. Si una excepción no es capturada, se emitirá un Error Fatal de PHP con un mensaje "Uncaught Exception ..." ("Excepción No Capturada"), a menos que se haya definido un manejador con `set_exception_handler()`.

En las líneas 12 y 20 definimos un nuevo objeto de la clase `Exception`, `$e`. Uno de los métodos de esta clase es `getMessage()`, al cual hacemos referencia en las mismas líneas.

```
1:  <?php
2:      function dividir($a, $b){
3:          if ($b==0){
4:              throw new Exception('El segundo argumento es 0');
5:          }
6:          return $a/$b;
7:      }
8:      try{
9:          $resul1 = dividir(5, 0);
10:         echo "Resul 1 $resul1". "<br>";
11:     }catch(Exception $e){
12:         echo "Excepción: ". $e->getMessage(). "<br>";
13:     }finally{
14:         echo "Primer finally<br>";
15:     }
16:     try{
17:         $resul2 = dividir(5, 2);
18:         echo "Resul 2 $resul2". "<br>";
19:     }catch(Exception $e){
20:         echo "Excepción: ". $e->getMessage(). "<br>";
21:     }finally{
22:         echo "Segundo finally";
23:     }
```

La salida de este ejemplo es:

```
Excepción: El segundo argumento es 0
Primer finally
Resul 2 2.5
Segundo finally
```

En la primera llamada a `dividir()`, línea 9, se produce una excepción. Por tanto, el resto del bloque `try`, el `echo`, no se ejecuta. Si se ejecuta el `catch` y el `finally` correspondientes. En la segunda llamada a `dividir()`, línea 17, no se produce excepción. En este caso se ejecutan el `echo` del `try` y a continuación el del bloque `finally`. El bloque `catch` no se ejecuta.

2. Formularios en HTML

La forma natural para hacer llegar a la aplicación web los datos del usuario desde un navegador, es utilizar formularios HTML.

Los formularios HTML van encerrados siempre entre las etiquetas `<FORM>` `</FORM>`. Dentro de un formulario se incluyen los elementos sobre los que puede actuar el usuario, principalmente usando las etiquetas `<INPUT>`, `<SELECT>`, `<TEXTAREA>` y `<BUTTON>`.

```
<form action = "" method = "">
```

```
...
```

```
</form>
```

El atributo `action` del elemento `FORM` indica la página a la que se le enviarán los datos del formulario.

Por su parte, el atributo `method` especifica el método usado para enviar la información. Este atributo puede tener dos valores: `GET` y `POST`, siendo `GET` el valor por defecto.

2.1. Características de GET y POST

Tanto `GET` como `POST` crean un array asociativo. En este array las claves son los nombres de los controles de formulario y los valores son los datos de entrada del usuario.

Tanto `GET` como `POST` son variables superglobales, y se accede a su contenido con las sentencias `$_GET` y `$_POST`. Por tanto, siempre son accesibles, independientemente del ámbito donde nos encontremos, pudiendo acceder a ellos desde cualquier función, clase o archivo sin tener que hacer nada especial.

La diferencia fundamental entre ambas es cómo se pasan al script:

`$_GET` es un array de variables que se pasan al script actual a través de los parámetros de URL.

`$_POST` es un array de variables que se pasan al script actual a través del método HTTP `POST`.

2.1.1. Ejemplo del método GET

Para explicar cómo usar el método `GET` vamos a utilizar el formulario `form_get.html`

```
1: <!DOCTYPE html>
2: <html>
3:   <head>
4:     <title>Formulario usando get</title>
5:     <meta charset = "UTF-8">
6:   </head>
7:   <body>
8:     <section>
9:       <h2>Sin comprobar nombre</h2>
10:      <form action = "hola_nombre.php" method = "get">
11:        <label for="nombre"> Introduce tu nombre: </label>
12:        <input id="nombre" name = "nombre" type = "text">
```

```

13:         <input type = "submit">
14:     </form>
15: </section>
16: <section>
17:     <h2>Comprobando el nombre</h2>
18:     <form action = "hola_comprobacion.php" method = "get">
19:         <label for="nombre"> Introduce tu nombre: </label>
20:         <input id="nombre" name = "nombre" type = "text">
21:         <input type = "submit">
22:     </form>
23: </section>
24: </body>
25: </html>

```

En los campos del formulario que se vayan a enviar hay que usar el atributo name, este será el nombre de la variable a recoger con el método GET (o con el método POST).

Cuando se usa el método GET los parámetros se pasan al servidor en la URL. A la ruta normal para acceder a una página se le añade el carácter “?” como indicador de que empieza la lista de parámetros.

Cada parámetro tiene un nombre, a la izquierda del igual, y un valor, a la derecha del igual. Los argumentos están separados entre sí por el carácter ampersand.

Por ejemplo, si se accede con el navegador a:

`http://localhost/.../hola_nombre.php?nombre=Ana`

Se solicita al servidor el fichero `/.../hola_nombre.php` del servidor localhost y se le pasa un parámetro llamado "nombre" con valor "Ana".

Con la siguiente URL se añadiría otro parámetro para el apellido:

`http://localhost/.../hola_nombre.php?nombre=Ana&apellido=Luna`

Se puede acceder a los argumentos de la URL desde un bloque PHP usando el array superglobal `$_GET`, el cual tiene un elemento por cada argumento presente en la URL. El nombre del argumento será la clave del elemento del array.

El fichero `hola_nombre.php` muestra un mensaje personalizado usando el valor del parámetro nombre.

```

1: <?php
2:     echo "Hola: " . $_GET["nombre"];

```

Si se accede con la ruta.

`http://localhost/.../hola_nombre.php?nombre=Ana`

se obtendrá el mensaje "Hola Ana". Si se accede con

`http://localhost/.../hola_nombre.php`

se obtendrá:

Warning: Undefined array key "nombre" in C:\xampp\htdocs\...\hola_nombre.php on line 2
Hola:

Para controlar si los parámetros se han pasado correctamente se pueden utilizar las funciones `empty()` o `is_null()`. Las dos devuelven TRUE cuando el parámetro no está presente en la URL. La diferencia está en los parámetros presentes, pero sin valor, por ejemplo:

`http://localhost/.../hola_nombre.php?nombre`

En este caso, `empty($_GET["nombre"])` devuelve TRUE, pero `is_null($_GET["nombre"])` devuelve FALSE.

El ejemplo `hola_comprobacion.php` mejora el anterior para mostrar un mensaje de error si no se pasa el parámetro nombre.

```
1: <?php
2:     if (empty($_GET["nombre"])) {
3:         echo "Error, falta el parámetro nombre";
4:     }else {
5:         echo "Hola " . $_GET["nombre"];
6:     }
```

Ejercicio 1

Crear un formulario en el que podamos escribir dos valores y que, una vez pulsado el botón enviar, muestre su suma. La función utilizada para sumar debe comprobar que los dos valores existan y sean números.

2.1.2. ¿Cuándo usar GET?

La información enviada desde un formulario con el método GET es visible para todos. Todos los nombres y valores de las variables se muestran en la URL.

GET también tiene límites en la cantidad de información para enviar. La limitación es de unos 2000 caracteres. Sin embargo, debido a que las variables se muestran en la URL, es posible crear un marcador de la página en un navegador. Esto puede resultar útil en algunos casos.

GET se puede utilizar para enviar datos no confidenciales.

¡NUNCA se debe utilizar GET para enviar contraseñas u otra información confidencial!

2.1.3. Ejemplo del método POST

Para explicar el método post vamos a ver hacer uso del archivo `login_basico.html`.

```
1: <!DOCTYPE html>
2: <html>
3:     <head>
4:         <title>Formulario de login</title>
5:         <meta charset = "UTF-8">
6:     </head>
7:     <body>
8:         <form action = "login_basico.php" method = "POST">
```

```

9:         <input name = "usuario" type = "text">
10:        <input name = "clave" type = "password">
11:        <input type = "submit">
12:    </form>
13: </body>
14: </html>

```

Al igual que con el método GET, vemos que en el atributo `action` de la línea 8 se especifica la ruta del script al que se enviará el formulario para que se procese, `login_basico.php`. El cual contiene el siguiente código:

```

1: <?php
2:     echo "Usuario introducido: " . $_POST['usuario']. "<br>";
3:     echo "Clave introducida: " . $_POST['clave'];

```

Al usar el método POST los parámetros no se muestran en la URL, pero se pueden consultar desde la consola del navegador accediendo al menú de desarrollo mediante el atajo de teclado Ctrl+Mayus+I y seleccionando la pestaña de Red o Network.

Una vez abierta, muestra las peticiones que se realizan desde el navegador.

Si se selecciona la correspondiente al envío del formulario, se pueden consultar los parámetros solicitados.

2.1.4. ¿Cuándo usar POST?

La información enviada desde un formulario con el método POST es invisible para los demás. Todos los nombres y valores están incrustados en el cuerpo de la solicitud HTTP. No tiene límites en la cantidad de información a enviar.

Además, POST admite funciones avanzadas, como poder cargar varios archivos de forma simultánea en el servidor.

Sin embargo, debido a que las variables no se muestran en la URL, no es posible crear un marcador de la página con los datos del formulario.

Los desarrolladores prefieren POST para enviar datos de formularios.

2.2. Validación de datos en el mismo formulario

Siempre que sea posible, es preferible validar los datos que se introducen en el navegador antes de enviarlos. Para ello deberemos usar código en lenguaje Javascript.

Si por algún motivo hay datos que se tengan que validar en el servidor, por ejemplo, porque necesites comprobar que los datos de un usuario no existan ya en la base de datos antes de introducirlos, será necesario hacerlo con código PHP en la página que figura en el atributo `action` del formulario.

En este caso, una posibilidad que podemos tener en cuenta es usar la misma página que muestra el formulario como destino de los datos. Si tras comprobar los datos éstos son correctos, se reenvía a otra página. Si son incorrectos, se rellenan los datos correctos en el formulario y se indican cuáles son incorrectos y por qué.

Para hacerlo de este modo, hemos de comprobar si la página recibe datos (hay que mostrarlos y no generar el formulario), o si no recibe datos (hay que mostrar el formulario). Esto se puede hacer utilizando la función `isset`¹ con una variable de las que se deben recibir (por ejemplo, poniéndole un nombre al botón de enviar y comprobando sobre él).

En el ejemplo `validación_1.php` hemos englobado el formulario dentro de una sentencia `else` (línea 16) para que sólo se genere si no se reciben datos en la página.

Además, para enviar los datos a la misma página que contiene el formulario hemos usado `$_SERVER['PHP_SELF']` (línea 18) para obtener su nombre; esto hace que no se produzca un error, aunque la página se cambie de nombre.

```
1: <!DOCTYPE html>
2: <html>
3: <head>
4: <title>Validación de datos en el propio formulario</title>
5: <meta charset="UTF-8">
6: </head>
7: <body>
8: <?php
9:   if (isset($_POST['enviar'])) {
10:     $nombre = $_POST['nombre'];
11:     $modulos = $_POST['modulos'];
12:     print "Nombre: " . $nombre . "<br />";
13:     foreach ($modulos as $modulo) {
14:       print "Modulo: " . $modulo . "<br />";
15:     }
16:   } else {
17:     ?>
18:     <form name="input" action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
19:       Nombre del alumno: <input type="text" name="nombre" /><br />
20:       <p>Módulos que cursa:</p>
21:       <input type="checkbox" name="modulos[]" value="DWES" />
22:       Desarrollo web en entorno servidor<br />
23:       <input type="checkbox" name="modulos[]" value="DWEC" />
24:       Desarrollo web en entorno cliente<br />
25:       <br />
26:       <input type="submit" name="enviar" value="Enviar" />
27:     </form>
28:     <?php
29:   }
30:   ?>
31: </body>
32: </html>
```

A continuación, modificamos el ejemplo anterior para conseguir revisar los datos antes de mostrarlos. Vamos a comprobar que el nombre no está vacío y que se haya seleccionado alguno de

¹ `isset` — Determina si una variable está definida y no es `null`

los módulos. Además, si faltase algún dato, se generará el formulario indicando que datos no se ha rellenado y mostrando los ya rellenados.

En el siguiente código de ejemplo, `validacion_2.php` se muestra cómo hacerlo.

```
1: <!DOCTYPE html>
2: <html>
3: <head>
4: <title>Validación de datos en el propio formulario</title>
5: <meta charset="UTF-8">
6: </head>
7: <body>
8: <?php
9: if (!empty($_POST['modulos']) && !empty($_POST['nombre'])) {
10:     $nombre = $_POST['nombre'];
11:     $modulos = $_POST['modulos'];
12:     print "Nombre: " . $nombre . "<br />";
13:     foreach ($modulos as $modulo) {
14:         print "Modulo: " . $modulo . "<br />";
15:     }
16: } else {
17:     ?>
18:     <form name="input" action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
19:         Nombre del alumno:
20:         <input type="text" name="nombre" value="<?php
21:             if(isset($_POST['nombre']))
22:                 echo $_POST['nombre'];
23:             ?>" />
24:         <?php
25:         if (isset($_POST['enviar']) && empty($_POST['nombre']))
26:             echo "<span style='color:red'> &lt;-- Debe introducir un nombre.</span>"
27:             ?><br />
28:         <p>Módulos que cursa:
29:         <?php
30:         if (isset($_POST['enviar']) && empty($_POST['modulos']))
31:             echo "<span style='color:red'> &lt;-- Debe escoger al menos un módulo.</span>"
32:             ?>
33:         </p>
34:         <input type="checkbox" name="modulos[]" value="DWES" <?php
35:             if (isset($_POST['modulos']) && in_array("DWES", $_POST['modulos']))
36:                 echo 'checked="checked"';
37:             ?> />
38:         Desarrollo web en entorno servidor
39:         <br />
40:         <input type="checkbox" name="modulos[]" value="DWE" <?php
41:             if (isset($_POST['modulos']) && in_array("DWE", $_POST['modulos']))
42:                 echo 'checked="checked"';
43:             ?> />
44:         Desarrollo web en entorno cliente<br />
```

```

45:     <br />
46:     <input type="submit" name="enviar" value="Enviar" />
47: </form>
48: <?php
49: }
50: ?>
51: </body>
52: </html>

```

En la línea 9 comprobamos que el nombre no esté vacío y que se haya seleccionado como mínimo uno de los módulos. Como ninguna de estas condiciones se cumple al acceder al formulario por primera vez, pasamos a la línea 18, generando el formulario para solicitar los datos.

Cada vez que hacemos clic en enviar, hacemos una llamada al propio formulario y mostramos una advertencia en aquellos campos que no hubiéramos seleccionado o introducido ningún valor previamente, líneas 26 y 31.

Los nombres de los checkbox, líneas 34 y 40, coinciden, esto es debido a que sus valores se guardarán en el array `modulos[]`

Actividad Obligatoria 1

Programa en una única página web en PHP una aplicación para mantener los datos de los clientes de una empresa.

**Los datos que se han de almacenar de cada cliente son: su nombre y su correo electrónico.
No podrá haber nombres repetidos en los datos de los clientes.**

Dividiendo la página web de forma horizontal, en la parte de superior figurará un formulario con dos cuadros de texto, uno para el nombre y otro para el correo. En la parte inferior se mostrarán los datos de los clientes.

Cada vez que se envíe el formulario:

- Si el nombre está vacío, se mostrará una advertencia.
- Si el nombre que se introdujo no existe en los datos de los clientes, y el correo no está vacío, se añadirá a la agenda.
- Si el nombre que se introdujo ya existe en los datos de los clientes y el correo no está vacío, se actualizará el correo.
- Si el nombre que se introdujo ya existe en los datos de los clientes y no se indica el correo electrónico, se eliminarán los datos del cliente.