# TEMA 4

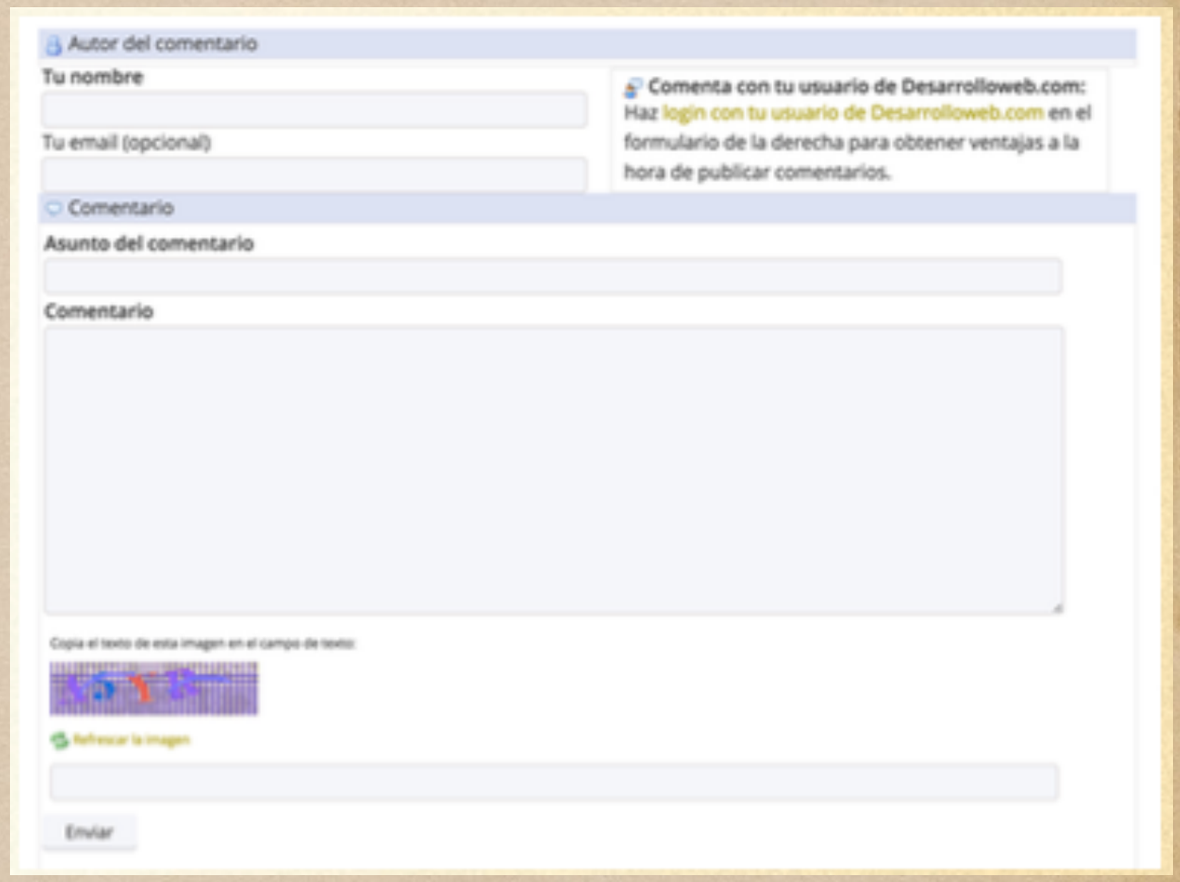
# GET & POST

## $_GET y $_POST en PHP

# Utilidad de GET & POST

En el desarrollo de aplicaciones web, usamos estas interfaces para:

- Pasar datos entre páginas.
- Entrada de datos por parte del usuario




```
<a href="ver.php?id=34">Ver producto</a>
```

# ¿Qué son $_GET y $_POST

Ya vimos en el tema anterior que son:

- Variables superglobales del sistema.
- Arrays asociativos.

Se utilizan para el paso de variables entre scripts, normalmente por medio de formularios.

Las podemos usar como:

```
$nombre = $_POST["nombre"];

$producto = $_GET["id"];
```

# Diferencias

Según el método de envío de los datos:

- GET es más "sucio" (los datos se ven en la URL).
- POST es más transparente.
- POST es más "seguro" (sólo aparentemente).

En principio, lo anterior hace más deseable a POST, pero:

- Requiere tener un formulario (GET no)
- El "refresh" de un POST produce un mensaje "horrible".

# GET

# Formato de los datos en la URL

example.com/buscar.php?q=algo&src=otra

Pares clave/valor

Al recibir esos datos

$_GET[“q”] tendrá el valor “algo”

$_GET[“src”] tendrá el valor “otra”

# Confianza cero en el usuario

Nunca estamos seguros de que el dato lo vamos a recibir tal y como se espera.

Por tanto, como no damos por hecho la recepción de la variable, debemos de preguntar si existe, lo cual haremos con la función **isset()**.

```
if (isset($_GET["q"])) {

  //entonces hacemos lo previsto con $_GET["q"]

}
```

# urlencode()

Recordemos que los datos en la URL deben ir escritos con una codificación especial. Hemos de llevar mucho cuidado con caracteres especiales, como &, el espacio en blanco, $ y otros. Las versiones modernas de los navegadores ya arreglan parte de estos problemas pero no todos, y los antiguos, ninguno.

No nos podemos fiar que el navegador vaya a codificar por nosotros la URL, aunque en muchas ocasiones lo haga. Por ejemplo:

**www.midominio.com/dibujos.php?preferido=Tom & Jerry**

Solución:

```
www.midominio.com/dibujos.php?preferido=<?php urlencode(Tom & Jerry) ?>
```

# POST

# POST

Para poder utilizar el array asociativo $_POST necesitamos usar formularios. Es un método aparentemente más seguro, pues los datos que enviamos al script no se ven en la URL como nos ocurría con $_GET.

Pero sólo aparentemente, pues con las herramientas para desarrolladores que nos proporcionan los diferentes navegadores que usamos, podemos cambiar o alterar el formulario que envía la información.

Es una forma más de insistir en que nunca nos debemos fiar de los usuarios.

# Formularios

```
<form action="post.php" method="post">
   <p>
      <label for="nombre">Nombre</label>
      <input type="text" name="nombre" value="">
   </p>
   <p>
      <input type="submit" value="Enviar">
   </p>
</form>
```

Definimos el comportamiento de este formulario

Se asignan nombres a los campos del formulario

# Formularios

- **action:** Indica el archivo al que se van a enviar los datos del formulario. Si queremos enviarlo al mismo archivo, se puede quedar en blanco (action="") o usar $_SERVER["PHP_SELF"].

- **method:** Indica si el método es por GET o POST. Si no se dice nada, por defecto es GET.

- **name:** Indica el nombre de los campos. Esos serán las claves en el array (ya sea $_GET o $_POST).

Cuando el script "post.php" reciba estos datos por POST, podemos acceder a ellos como

**$_POST["nombre"]**

y tendrá el valor del texto escrito por el usuario en el campo del formulario.

# Recibiendo datos

Para saber que estamos recibiendo datos desde el formulario podemos usar:

```
if ($_POST) {

   //estoy recibiendo datos por POST

}
```

Pero nos pueden cambiar los nombres de las variables, por lo que se recomienda, además, comprobar los diferentes campos:

```
if (isset($_POST[“nombre”])) {

   //La variable nombre esta siendo enviada por POST

}
```

# El botón de submit

Si tenemos un formulario con varios botones de enviar y queremos saber cual de ellos se ha pulsado, debemos asignarle un nombre a ese campo, mediante el atributo name.

```
<input type="submit" value="Editar" name="accion">

<input type="submit" value="Borrar" name="accion">
```

# Caracter "." y " " en nombres de campos

En los nombres de los campos de formulario (el atributo name) si se encuentra un carácter "." o un espacio en blanco " ", PHP los sustituye por un carácter de subrayado.

```
<input type="text" name="num dias">

<input type="text" name="nombre.apellidos">
```

Se accede mediante:

```
$_POST["num_dias"];

$_POST["nombre_apellidos"];
```

# Datos en un array

Podemos agrupar datos de un formulario desde varios campos, o en un campo <select>, con la notación de array.

```
<input type="password" name="clave[original]">

<input type="password" name="clave[repetida]">

<select multiple name="categorias[]">
```

Y accedemos a ellos como:

```
$_POST["clave"]    //Esto es un array

$_POST["clave"]["repetida"]

$_POST["categorias"]   //Esto también es un array
```

# Cuidados elementales

Recordemos:

**Toda entrada de datos por parte del usuario es potencialmente insegura.**

Debemos siempre cuidar que:

- Los datos existen.
- Contienen lo que se espera.
- Y en el formato que se espera.

# Funciones para verificar los tipos

PHP ofrece diversas funciones para comprobar los tipos de las variables. Son muy usadas para comprobar los datos que nos llegan por GET/POST.

- `is_int()`
- `is_numeric()`
- `is_array()`
- ...

# Cuidado con lo que se hace

Veamos dos problemas:

```
if (isset($_GET["seccion"])) {
  include $_GET["seccion"] . ".php";
}
```

---

```
<?php
  foreach($_POST as $key => $value) {
    $$key = $value;
  }
?>
```