Clase 08

Análisis de Sistemas

Materia:
Programación Web II

Docente contenidista: ROLDÁN, Hernán

Revisión: Coordinación

Contenido

Clase 8

¡Te damos la bienvenida a la materia **Programación Web II**!

En esta clase vamos a ver los siguientes temas:

- Formularios: finalidad y uso.
- Creación de un formulario HTML básico.
- Seguridad en el envío de formularios y prevención de ataques de tipo Cross-Site Scripting (XSS).

Formularios: finalidad y uso

Los formularios en PHP son una parte fundamental de la interacción entre los usuarios y las aplicaciones web. Permiten a los usuarios enviar datos al servidor para su procesamiento y almacenamiento.

Un formulario es una colección de elementos de entrada, como campos de texto, casillas de verificación, botones de opción, listas desplegables, etc., que se utilizan para recopilar información del usuario. Los usuarios pueden ingresar datos en estos elementos y luego enviarlos al servidor a través del formulario.

Cuando un usuario envía un formulario, los datos ingresados se envían al servidor web para su procesamiento. PHP es uno de los lenguajes de programación más populares para procesar datos de formularios en el lado del servidor.

Algunas de las funciones principales de los formularios en PHP son:

- ## Recopilación de datos

  Los formularios permiten recopilar datos ingresados por los usuarios, como nombres, direcciones, números de teléfono, comentarios, etc.
  Estos datos son esenciales para realizar diversas acciones, como registrarse en un sitio web, enviar mensajes, realizar pedidos, completar encuestas, etc.

- ## Validación y verificación de datos

  Los formularios en PHP permiten validar y verificar los datos ingresados por los usuarios. Esto implica comprobar si los datos cumplen con ciertos criterios o restricciones, como la longitud mínima/máxima, el formato correcto (por ejemplo, una dirección de correo electrónico válida), la selección de opciones obligatorias, etc.
  La validación de datos es importante para garantizar la integridad y la seguridad de los datos.

- ## Procesamiento y almacenamiento de datos

  Después de que los datos del formulario se envían al servidor, PHP puede procesarlos y realizar acciones en consecuencia. Esto puede incluir el almacenamiento de datos en una base de datos, el envío de correos electrónicos, la generación de informes, la actualización de registros, etc.
  El procesamiento de datos permite que los formularios sean interactivos y funcionales.

En resumen, los formularios en PHP son una forma crucial de interactuar con los usuarios en aplicaciones web.
Permiten recopilar datos, validarlos, procesarlos y almacenarlos en el servidor. Esto facilita la interacción y la comunicación entre los usuarios y las aplicaciones, lo que permite una amplia gama de funcionalidades en línea.

Creación de un formulario HTML básico

archivo "formulario.php"

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Formularios con PHP</title>
    <!-- Add Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.
min.css" rel="stylesheet">
</head>
<body>
    <div class="container mt-5">
        <div class="row justify-content-center">
            <div class="col-lg-6">
                <h1 class="text-center">Formulario de Registro</h1>

                <form action="file2.php" method="POST">
                    <div class="form-group mb-3">
                        <label for="nombre">Nombre:</label>
                        <input type="text" name="nombre" id="nombre"
class="form-control" required placeholder="Ingrese su nombre">
                    </div>

                    <div class="form-group mb-3">
                        <label for="apellido">Apellido:</label>
                        <input type="text" name="apellido"
id="apellido" class="form-control" required placeholder="Ingrese su
apellido">
                    </div>

                    <div class="form-group mb-3">
                        <label for="usuario">Usuario:</label>
                        <input type="text" name="usuario" id="usuario"
class="form-control" required placeholder="Ingrese su nombre de
usuario">
                    </div>
                    <div class="form-group mb-3">
                        <label for="clave">Clave:</label>
```

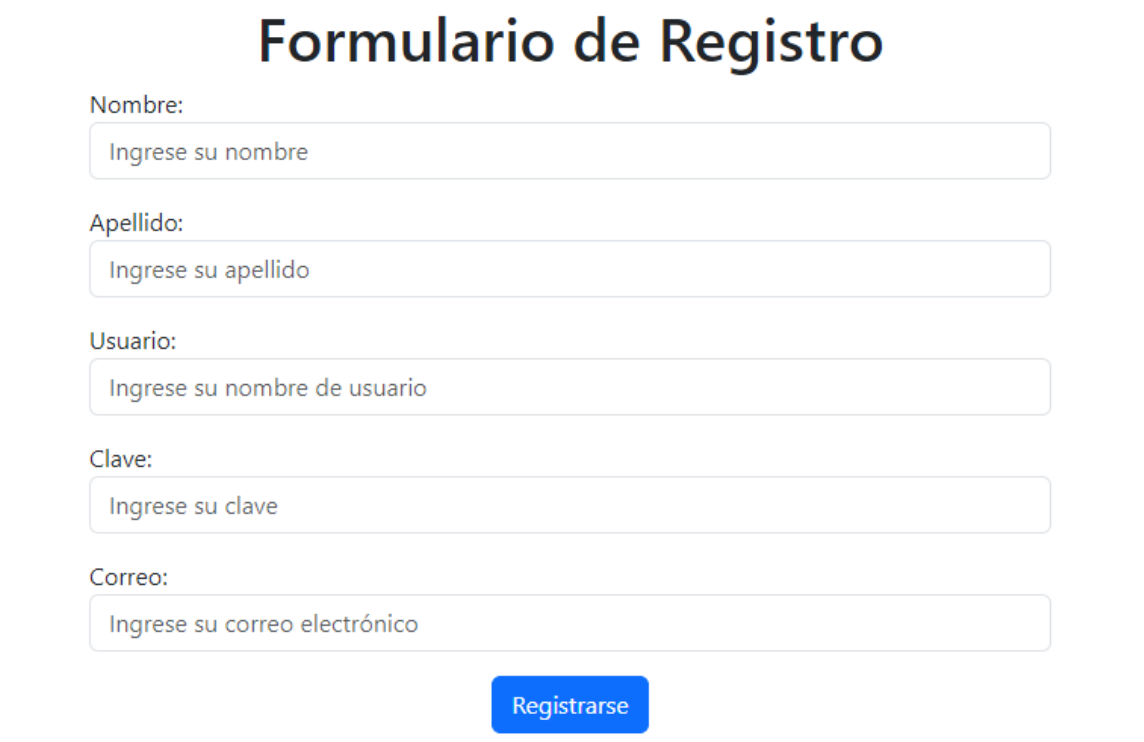
```
                        <input type="password" name="clave" id="clave"
class="form-control" required placeholder="Ingrese su clave">
                    </div>

                    <div class="form-group mb-3">
                        <label for="correo">Correo:</label>
                        <input type="email" name="correo" id="correo"
class="form-control" required placeholder="Ingrese su correo
electrónico">
                    </div>

                    <div class="text-center">
                        <input type="submit" value="Registrarse"
class="btn btn-primary">
                    </div>
                </form>
            </div>
        </div>
    </div>

    <!-- Add Bootstrap JavaScript (optional) -->
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bu
ndle.min.js"></script>
</body>
</html>
```

Vista en el navegador



Envío de datos a través del método POST

archivo "procesar_datos.php"

```php
<?php

  // Verificamos si la solicitud se realizó mediante el método POST
  if($_SERVER["REQUEST_METHOD"] == "POST"){
    // Verificamos si los campos del formulario están definidos y no
son NULL
    if(isset($_POST['nombre']) && isset($_POST['apellido']) &&
isset($_POST['usuario']) && isset($_POST['clave']) &&
isset($_POST['correo'])){
    // Recuperamos los datos del formulario
    $nombre = $_POST['nombre'];
    $apellido = $_POST['apellido'];
    $usuario = $_POST['usuario'];
    $clave = $_POST['clave'];
    $correo = $_POST['correo'];
      // Validamos los datos ingresados por el usuario.
      if(empty($nombre) || empty($apellido) || empty($usuario) ||
empty($clave) || empty($correo)){
        echo "Error, todos los campos son requeridos. Por favor,
verifica los datos ingresados.";
      }
      elseif(!filter_var($correo, FILTER_VALIDATE_EMAIL)){
        echo "El correo ingresado no es válido. Por favor, verifica el
correo ingresado.";
      }
      else{
      // Si los datos ingresados son válidos, se muestra un mensaje de
éxito.
        echo "¡Registro exitoso!";
      }
    }
    else{
```

```
      echo "Por favor, completa todos los campos.";
    }
  }
  else{
    echo "Error al enviar el formulario.";
  }

?>
```

Código explicado:

1. Primero, se verifica si la solicitud se realizó mediante el método POST utilizando ***$_SERVER["REQUEST_METHOD"] == "POST"***. Esto asegura que el código dentro de esta condición se ejecute sólo cuando se envíe el formulario.

2. A continuación, se utiliza la función isset para verificar si los campos del formulario están definidos en $_POST. Esto se hace mediante ***isset($_POST['nombre'])***, ***isset($_POST['apellido'])***, ***isset($_POST['usuario'])***, ***isset($_POST['clave'])*** y ***isset($_POST['correo'])***. Si algún campo está ausente o no está definido, se mostrará el mensaje de error correspondiente.

3. Si todos los campos están definidos, se procede a recuperar los datos del formulario utilizando ***$_POST['nombre']***, ***$_POST['apellido']***, ***$_POST['usuario']***, ***$_POST['clave']*** y ***$_POST['correo']*** y se asignan a variables respectivas: ***$nombre***, ***$apellido***, ***$usuario***, ***$clave*** y ***$correo***.

4. Luego, se realiza la validación de los datos. Se utiliza la función empty para verificar si algún campo está vacío, utilizando ***empty($nombre)***, ***empty($apellido)***, ***empty($usuario)***, ***empty($clave)*** y ***empty($correo)***. Si algún campo está vacío, se muestra el mensaje de error correspondiente.

5. Además, se utiliza la función filter_var junto con la opción FILTER_VALIDATE_EMAIL para validar el campo de correo electrónico, utilizando ***filter_var($correo, FILTER_VALIDATE_EMAIL)***. Si el campo de correo electrónico no tiene un formato válido, se muestra el mensaje de error correspondiente.

6. Si todos los campos pasan las validaciones, se ejecutan las acciones adicionales que desees realizar (por ejemplo, guardar los datos en una base de datos). En este ejemplo, simplemente se muestra un mensaje de "Registro exitoso".

7. Si alguno de los campos no está definido o si la solicitud no se realizó mediante el método POST, se mostrará el mensaje de "Acceso inválido".

Ejemplo de envío de datos a través del método GET en PHP:

Supongamos que tenemos una página llamada *"saludo.php"* que recibe un parámetro "nombre" a través del método GET y muestra un saludo personalizado:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Saludo Personalizado</title>
</head>
<body>

    <?php

      if(isset($_GET['nombre'])){
        $nombre = $_GET['nombre'];
        echo "<h1>¡Hola, $nombre!</h1>";
      }
      else{
        echo "<h1>¡Hola, visitante!</h1>";
      }

    ?>

</body>
</html>
```

Para probar este ejemplo, puedes acceder a la página *"saludo.php"* añadiendo el parámetro "nombre" en la URL, de la siguiente manera: *saludo.php?nombre=Juan*.
La página mostrará el saludo "¡Hola, Juan!".

El método GET se utiliza comúnmente para enviar datos a través de la URL.
Los datos enviados mediante el método GET son visibles en la barra de direcciones del navegador y se representan como parámetros de la URL.

Por ejemplo, en el caso del ejemplo anterior, el parámetro "nombre" se muestra en la URL como saludo.php?nombre=Juan.

Las principales **ventajas de utilizar el método GET** para el envío de datos son:

- **Simplicidad y facilidad de uso:**
  Es sencillo agregar parámetros a la URL y, por lo tanto, es fácil enviar datos a través del método GET.
- **Visibilidad de los datos:**
  Los datos enviados mediante GET son visibles en la URL, lo que facilita su inspección y depuración. Esto es útil para fines de desarrollo y pruebas.
- **Accesibilidad desde enlaces y marcadores:**
  Al utilizar GET, es posible compartir enlaces que contengan los parámetros necesarios para acceder directamente a una página o realizar una acción específica.
- **Cacheo de datos:**
  Los navegadores pueden almacenar en caché las respuestas de las solicitudes GET, lo que puede mejorar el rendimiento y la velocidad de carga de páginas repetidas.

Sin embargo, también hay algunas **consideraciones importantes** al utilizar el método GET:

- **Seguridad:**
  Los datos enviados a través de GET son visibles en la URL, lo que significa que no es adecuado para enviar información confidencial como contraseñas, ya que pueden quedar expuestas.
  Longitud de URL:
  La longitud de una URL está limitada, y enviar grandes cantidades de datos a través de GET puede ocasionar problemas si se supera el límite establecido por el servidor o el navegador.

Recordá que el método GET es ideal para enviar datos que no son sensibles y que deben ser visibles en la URL, como parámetros de búsqueda o identificadores, pero cuando se requiere mayor seguridad o se necesitan enviar grandes cantidades de datos, es más apropiado utilizar el método POST.

Seguridad en el envío de formularios y prevención de ataques de tipo Cross-Site Scripting (XSS)

La seguridad en el envío de formularios es un aspecto crítico en el desarrollo web, y uno de los riesgos más comunes es el ataque de tipo Cross-Site Scripting (XSS).

El XSS es un tipo de ataque en el que un atacante inserta código malicioso (scripts) en los datos enviados desde un formulario o cualquier otra fuente de entrada del usuario.

Cuando estos datos se muestran en una página web sin ser debidamente sanitizados, se ejecutan los scripts maliciosos en el navegador del usuario, lo que permite al atacante robar información, secuestrar sesiones o incluso modificar el contenido de la página.

Para prevenir los ataques XSS, es esencial aplicar técnicas de sanitización y validación adecuadas en los datos recibidos del formulario antes de mostrarlos en la página.

Algunas prácticas para prevenir XSS incluyen:

- **Sanitización de Entradas:** Antes de utilizar los datos recibidos del formulario, es importante eliminar cualquier contenido malicioso. Esto se puede lograr mediante funciones de sanitización específicas que eliminen etiquetas y caracteres peligrosos, como htmlspecialchars en PHP.
- **Validación de Entradas:** Es crucial validar los datos enviados desde el formulario para asegurarse de que cumplan con el formato y tipo de datos esperado. De esta manera, se evita que datos no válidos o potencialmente dañinos lleguen al sistema.
- **Uso de Contextos Seguros:** Al mostrar los datos en la página, es recomendable utilizar contextos seguros según el contenido que se va a mostrar. Por ejemplo, si se muestran datos como texto, se debe utilizar htmlspecialchars para evitar que se interpreten como etiquetas HTML. Si se muestran URLs, se puede utilizar urlencode para codificarlas de forma segura.

Ejemplo de formulario seguro para evitar ataques XSS:

Supongamos que tenemos un formulario de contacto que solicita el nombre y el mensaje del usuario, y luego muestra el mensaje en la página de agradecimiento.
Para hacerlo seguro, podemos implementar medidas de prevención de XSS.

archivo "formulario_de_contacto.php"

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Formulario de Contacto Seguro</title>
</head>
<body>
    <h1>Formulario de Contacto</h1>
    <form action="procesar_formulario.php" method="POST">
        <div>
            <label for="nombre">Nombre:</label>
            <input type="text" name="nombre" id="nombre" required>
        </div>
        <div>
            <label for="mensaje">Mensaje:</label>
            <textarea name="mensaje" id="mensaje" rows="4"
required></textarea>
        </div>
        <button type="submit">Enviar Mensaje</button>
    </form>
</body>
</html>
```

archivo "procesar_formulario.php"

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Agradecimiento</title>
</head>
<body>
    <h1>Gracias por tu mensaje</h1>
    <?php
    if (isset($_POST['nombre']) && isset($_POST['mensaje'])) {
        $nombre = trim($_POST['nombre']); // Eliminamos espacios en blanco al principio y al final del nombre
        $mensaje = trim(stripslashes($_POST['mensaje'])); // Eliminamos espacios y barras diagonales invertidas del mensaje

        // Prevenimos ataques XSS utilizando htmlspecialchars
        $nombre = htmlspecialchars($nombre);
        $mensaje = htmlspecialchars($mensaje);

        echo "<p>Nombre: $nombre</p>";
        echo "<p>Mensaje: $mensaje</p>";
    }
    ?>
</body>
</html>
```

En el ejemplo anterior, se implementan las siguientes prácticas de seguridad:

- Se utiliza el atributo ***required*** en los campos para asegurarse de que estén completos antes de enviar el formulario.
- Al recibir los datos en el servidor, se realiza una validación para asegurarse de que los datos sean válidos y no contengan contenido malicioso.
- Al mostrar el mensaje en la página de agradecimiento (*procesar_formulario.php*), se utiliza ***htmlspecialchars*** para evitar que el contenido se interprete como etiquetas HTML y, en su lugar, se muestre como texto seguro.
- La función ***trim()*** en PHP se utiliza para eliminar los espacios en blanco (espacios, tabulaciones y saltos de línea) al principio y al final de una cadena de texto. Esta función es útil para limpiar los datos de entrada y eliminar cualquier espacio adicional que el usuario pueda haber introducido por accidente. También es importante destacar que ***trim()*** no afecta a los espacios en blanco dentro del contenido del mensaje, solo los elimina en los extremos.
- Función ***stripslashes()***: La función ***stripslashes()*** en PHP se utiliza para eliminar las barras diagonales invertidas (\) que se agregan automáticamente a ciertos caracteres cuando se envían datos desde un formulario. Estas barras diagonales invertidas son agregadas por PHP como parte del proceso de escapado de caracteres para proteger los datos y evitar problemas de interpretación en ciertas situaciones.

Con la incorporación de las funciones trim() y stripslashes(), el formulario se vuelve más seguro y robusto.
Recordá que trim() asegura que los espacios en blanco no afecten la entrada de datos y evita que los usuarios ingresen espacios adicionales por accidente, y stripslashes() garantiza que las barras diagonales invertidas se eliminen del contenido del mensaje antes de mostrarlo, lo que previene problemas de interpretación y mejora la presentación del mensaje.

Hemos llegado así al final de esta clase en la que vimos:

- Formularios: finalidad y uso.
- Creación de un formulario HTML básico
- Seguridad en el envío de formularios y prevención de ataques de tipo Cross-Site Scripting (XSS).

Te esperamos en la **clase en vivo** de esta semana.
No olvides realizar el **desafío semanal**.

¡Hasta la próxima clase!

Bibliografía

Documentación Oficial de PHP:
https://www.php.net/manual/es/index.php

W3Schools: PHP Tutorial:
https://www.w3schools.com/php/default.asp

Cabezas Granado, L., González Lozano, F., (2018). Desarrollo web con PHP y MySQL. Anaya Multimedia.

Heurtel, O., (2016). Desarrolle un sitio web dinámico e interactivo. Eni Ediciones.

Welling, L., Thompson, L., (2017). Desarrollo Web con PHP y MySQL. Quinta Edición. Editorial Anaya.

Para ampliar la información

PHP: htmlspecialchars

PHP: trim

PHP: stripcslashes

PHP: The Right Way

Todo sobre PHP

PHP Programming Language Tutorial - Full Course

PHP Full Course for non-haters

CURSO de php desde cero

MDN: Introducción al lado Servidor

Guía de HTML