Universidad Politécnica de Tlaxcala Región Poniente

Ingeniería en Sistemas Computacionales

Materia: Programación WEB

Docente: Ing. Vanesa Tenopala Zavala

Alumno: Isaac Brandon Martínez Ramírez

Matricula: 22SIC008

Tema: WWW

Ciclo escolar: septiembre - diciembre 2024

Fecha: 8 de noviembre de 2024

Índice

Reporte de la Práctica 1

1. Descripción del Programa

La práctica 1 incluye dos archivos HTML, inicio.html y otra.html, que permiten enviar y mostrar datos mediante un enlace con parámetros en la URL (método GET). En inicio.html, se crea un enlace que envía el nombre y correo de un usuario a otra.html, donde JavaScript extrae los valores de la URL y los muestra en la página.

2. Objetivos

Enviar datos mediante un enlace: Crear un enlace en inicio.html que envíe el nombre y el correo del usuario a otra.html.

Mostrar los datos recibidos: Utilizar JavaScript en otra.html para leer los parámetros de la URL y mostrar la información en pantalla.

3. Componentes del Programa

3.1. Archivos HTML

- **inicio.html**: Página que contiene un enlace hacia otra.html enviando parámetros mediante la URL.
- **otra.html**: Página que recibe y valida los parámetros de la URL y muestra los datos utilizando JavaScript.

3.2. Funcionalidad JavaScript en otra.html

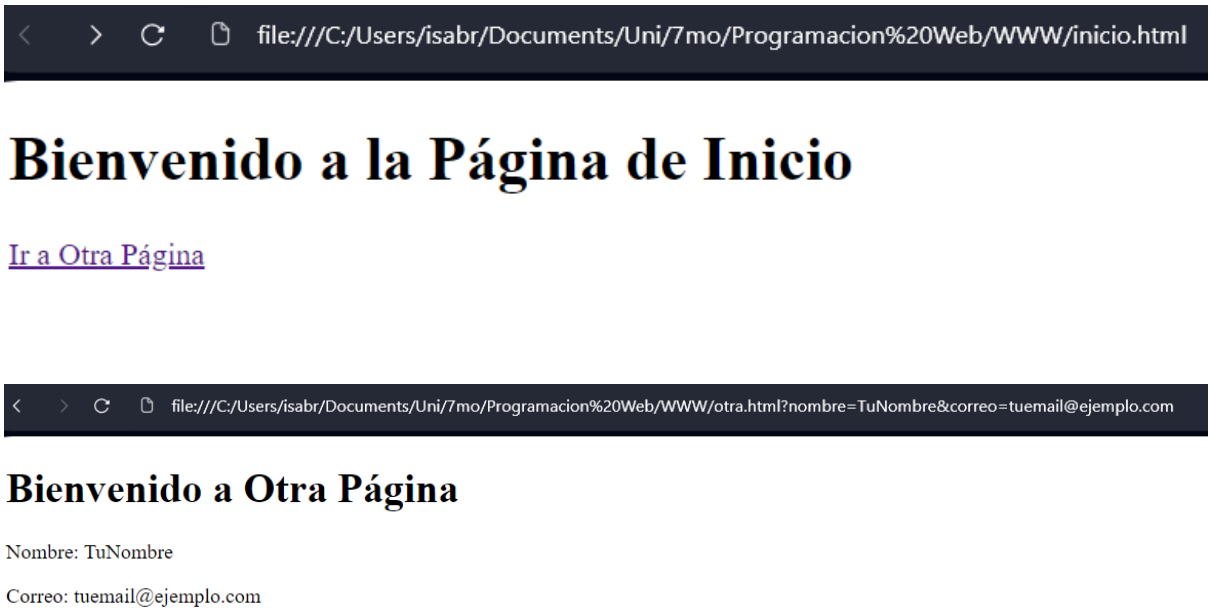
- **JavaScript**: Un script que toma los parámetros nombre y correo de la URL usando URLSearchParams y los inserta en elementos para mostrarlos al usuario.

4. Funcionamiento del Programa

1. **inicio.html**: Contiene un enlace con parámetros en la URL que envía los datos nombre y correo a otra.html. Al hacer clic en el enlace, el usuario es redirigido con los valores en la URL.

2. **otra.html**: Cuando la página se carga, el script JavaScript extrae los valores de nombre y correo de la URL y los muestra en pantalla en los elementos especificados.

5. Resultado



6. Conclusión

Esta práctica permite comprender cómo enviar información entre páginas mediante query strings y cómo extraer y mostrar los datos en el cliente. Es una técnica útil para enviar datos no sensibles y mantener una navegación simple entre páginas.

Reporte de la Práctica 2

1. Descripción del Programa

La práctica 2 incluye dos archivos HTML, formulario.html y registro.html, que permiten al usuario ingresar su nombre y correo en un formulario y enviarlos a otra página mediante el método GET. registro.html utiliza JavaScript para mostrar los datos recibidos de la URL en la página.

2. Objetivos

- Capturar datos de entrada: Crear un formulario en formulario.html donde el usuario pueda ingresar su nombre y correo electrónico.
- Enviar datos a otra página: Utilizar el método GET para enviar los datos a registro.html y mostrar la información recibida.

3. Componentes del Programa

3.1. Archivos HTML

- formulario.html: Página que contiene un formulario con campos para ingresar nombre y correo. Al enviar el formulario, los datos se envían a registro.html mediante el método GET.
- registro.html: Página que recibe los datos a través de la URL, extrae los valores con JavaScript y los muestra en pantalla.

3.2. Funcionalidad JavaScript en formulario.html

- **JavaScript**: Un script que utiliza URLSearchParams para extraer los valores de nombre y correo de la URL y mostrarlos en elementos .

4. Funcionamiento del Programa

1. **formulario.html**: Presenta un formulario con campos para el nombre y el correo. Al enviarlo, se redirige a registro.html y los datos se incluyen en la URL mediante el método GET.

2. **registro.html**: Al cargar la página, el script JavaScript extrae y muestra los valores de nombre y correo de la URL en elementos .

5. Resultado

6. Conclusión

Esta práctica es útil para entender la transmisión de datos a través de formularios y la manipulación de datos en URLs mediante JavaScript. Utilizar GET permite ver y manipular los datos fácilmente en la URL, lo cual es adecuado para datos no confidenciales.

Diferencias entre Enviar Formularios HTML con el Método GET y el Método POST

1. Introducción a los Métodos GET y POST

Al enviar datos desde un formulario HTML a un servidor, el método de envío especifica cómo se transmitirá esa información. Los métodos más comunes en formularios HTML son **GET** y **POST**.

- **GET**: Envía los datos del formulario como parte de la URL en un formato de cadena de consulta (query string).
- **POST**: Envía los datos del formulario en el cuerpo de la solicitud HTTP, no en la URL.

Ambos métodos tienen ventajas y desventajas dependiendo de la naturaleza de los datos que se envían y las necesidades de la aplicación.

2. Diferencias Principales entre GET y POST

2.1 Visibilidad de los Datos

- **GET**: Los datos se envían en la URL, lo que hace que sean visibles para cualquier persona que tenga acceso al enlace. Por ejemplo, un formulario que envía los datos mediante GET podría tener una URL como:

https://ejemplo.com/resultado?nombre=Juan&edad=30

Esto es adecuado para datos que no sean sensibles, como preferencias de búsqueda o parámetros de navegación. Sin embargo, no es seguro para información confidencial, como contraseñas o números de tarjetas de crédito.

- **POST**: Los datos se envían en el cuerpo de la solicitud HTTP, por lo que no son visibles en la URL. Esto es preferible para datos sensibles y permite un mayor nivel de privacidad.

2.2 Tamaño de los Datos Enviados

- **GET**: La longitud de los datos está limitada por el tamaño máximo de la URL, que varía según el navegador, pero generalmente es de aproximadamente 2048 caracteres. Esto limita la cantidad de datos que se pueden enviar y lo hace inadecuado para formularios grandes.
- **POST**: No tiene una restricción en la longitud de los datos, ya que estos se envían en el cuerpo de la solicitud. Esto lo hace ideal para enviar grandes cantidades de datos, como información de registros o archivos.

2.3 Caching (Almacenamiento en Caché)

- **GET**: Las solicitudes GET son almacenadas en la memoria caché del navegador, lo que facilita que el usuario pueda regresar a la página anterior con los mismos datos. Esto es útil para búsquedas y filtros de datos.
- **POST**: Generalmente, las solicitudes POST no se almacenan en caché, ya que cada vez que se envía una solicitud POST se trata como una acción nueva. Esto es adecuado para formularios de registro o pagos, donde no queremos que los datos se reutilicen de manera no intencional.

2.4 Idempotencia

- **GET**: Es un método idempotente, lo que significa que una misma solicitud GET repetida no debe alterar el estado del servidor. Es decir, llamar varias veces a una URL con GET debería producir el mismo resultado sin efectos secundarios.
- **POST**: No es idempotente, ya que puede cambiar el estado del servidor (por ejemplo, al insertar datos en una base de datos). Esto es relevante para formularios que crean, actualizan o eliminan datos.

3. Ejemplos Prácticos

Ejemplo 1: Formulario con GET

Este formulario envía el nombre del usuario en la URL. Es útil para búsquedas o datos no sensibles.

HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <title>Formulario con GET</title>
</head>
<body>
   <form action="resultado.php" method="GET">
      <label for="nombre">Nombre:</label>
      <input type="text" id="nombre" name="nombre">
      <button type="submit">Enviar</button>
   </form>
</body>
</html>
```

Si el usuario ingresa "Juan" como nombre, la URL sería:

https://ejemplo.com/resultado.php?nombre=Juan

Ejemplo 2: Formulario con POST

Este formulario es adecuado para datos sensibles, como el correo electrónico y la contraseña.

HTML

```
<!DOCTYPE html>
<html lang="es">
<head>
   <meta charset="UTF-8">
   <title>Formulario con POST</title>
</head>
```

```
<body>
    <form action="login.php" method="POST">
        <label for="correo">Correo:</label>
        <input type="email" id="correo" name="correo">

        <label for="password">Contraseña:</label>
        <input type="password" id="password" name="password">

        <button type="submit">Ingresar</button>
    </form>
</body>
</html>
```

En este caso, los datos no serán visibles en la URL, lo que ofrece mayor privacidad y seguridad.

4. Cuándo Usar GET y POST

- **Usar GET** para:
    - Enviar datos no sensibles.
    - Realizar búsquedas o filtros en los que la visibilidad en la URL es útil.
    - Solicitudes que necesitan ser cacheadas o almacenadas en los favoritos.
- **Usar POST** para:
    - Enviar datos sensibles (como contraseñas o información personal).
    - Formularios que generan un cambio en el servidor (registros, actualizaciones).
    - Formularios que requieren el envío de archivos adjuntos o grandes cantidades de datos

5. Conclusiones

Cada método de envío en formularios HTML, **GET** y **POST**, tiene características únicas que los hacen adecuados para diferentes escenarios. GET es ideal para transmitir datos no sensibles y permite ver los parámetros en la URL, lo que facilita la creación de enlaces compartibles y el almacenamiento en caché de resultados. Sin embargo, su límite de tamaño y visibilidad lo hacen menos seguro para información privada o grandes cantidades de datos.

Por otro lado, POST es la opción preferida para enviar datos confidenciales y cantidades significativas de información, ya que utiliza el cuerpo de la solicitud y evita exponer los datos en la URL. La elección entre GET y POST no solo depende de la cantidad y sensibilidad de los datos, sino también de las necesidades de interacción de la aplicación web, como la idempotencia y el almacenamiento en caché. En resumen, comprender las diferencias y usos óptimos de GET y POST permite a los desarrolladores construir aplicaciones web más seguras, eficientes y adaptadas a las necesidades de los usuarios.

Manipulación de Headers con PHP

1. Introducción a los Headers en PHP

Los **headers HTTP** son metadatos que se envían con la solicitud o la respuesta entre el servidor y el cliente. En PHP, se puede usar la función header() para enviar headers específicos al cliente, controlando aspectos como la redirección, el tipo de contenido y la autenticación.

2. Ejemplos Comunes de Manipulación de Headers en PHP

2.1 Redirección de Página

La redirección permite enviar al usuario a otra página. Para ello, se usa el código de estado Location, generalmente acompañado por un código de estado 301 (redirección permanente) o 302 (redirección temporal).

PHP:

```
<?php
// Redirección a otra página
header("Location: https://ejemplo.com/nueva-pagina.php");
exit;
?>
```

2.2 Controlar el Tipo de Contenido

El header Content-Type especifica el tipo de contenido que el servidor está enviando al cliente. Esto es útil para archivos como JSON, XML o descargas de archivos.

PHP:

```
<?php
// Enviar un archivo JSON
```

```
header("Content-Type: application/json");
echo json_encode(["nombre" => "Juan", "edad" => 30]);
?>
```

2.3 Descarga de Archivos

Podemos forzar la descarga de archivos indicando el Content-Disposition como attachment, lo cual hará que el navegador descargue el archivo en lugar de visualizarlo.

PHP:

```
<?php
// Forzar la descarga de un archivo
header("Content-Type: application/octet-stream");
header("Content-Disposition: attachment; filename=\"archivo.pdf\"");
readfile("ruta/al/archivo.pdf");
?>
```

2.4 Controlar la Caché

El manejo de headers de caché permite definir si el contenido debe almacenarse en caché y por cuánto tiempo.

PHP:

```
<?php
// Desactivar caché
header("Cache-Control: no-cache, no-store, must-revalidate"); // HTTP 1.1.
header("Pragma: no-cache"); // HTTP 1.0.
header("Expires: 0"); // Proxies.
?>
```

3. Ejemplo de Aplicación Completa con Headers

Supongamos que tenemos una aplicación que, al recibir un formulario, redirige al usuario si los datos son válidos, devuelve un archivo PDF si así lo requiere y evita el cacheo de datos sensibles.

PHP:

```
<?php
// Validar datos del formulario (ejemplo simplificado)
if ($_SERVER['REQUEST_METHOD'] == 'POST' && isset($_POST['descargar_pdf'])) {
    // Evitar caché
    header("Cache-Control: no-cache, no-store, must-revalidate");
    header("Pragma: no-cache");
    header("Expires: 0");

    // Forzar descarga de archivo PDF
    header("Content-Type: application/pdf");
    header("Content-Disposition: attachment; filename=\"reporte.pdf\"");
    readfile("ruta/al/reporte.pdf");
    exit;
} else {
    // Redirigir a la página de error si no es una solicitud válida
    header("Location: error.php");
    exit;
}
?>
```

4. Consideraciones y Mejores Prácticas

- **Enviar Headers Antes del Contenido**: Los headers deben enviarse antes de cualquier contenido HTML o salida en PHP.

- **Usar exit Después de una Redirección**: Para asegurar que el script se detenga después de una redirección.
- **Configurar Headers de Caché en Información Sensible**: Especialmente en aplicaciones de banca o datos personales.

5. Conclusiones

La manipulación de headers en PHP es una herramienta poderosa que permite a los desarrolladores controlar el comportamiento y la seguridad de las aplicaciones web en varios aspectos. Desde la redirección de usuarios hasta la configuración de caché y el control de tipos de contenido, el uso adecuado de headers mejora significativamente la experiencia del usuario y la gestión de datos.

La capacidad de enviar headers específicos en diferentes situaciones permite una personalización avanzada de la comunicación entre cliente y servidor. Esto es particularmente útil en aplicaciones que requieren la descarga de archivos, el envío de respuestas en formato JSON, o la protección de datos sensibles mediante configuraciones de caché. Comprender cómo y cuándo usar headers con PHP brinda a los desarrolladores un mayor control sobre la respuesta del servidor y fortalece la seguridad y funcionalidad de las aplicaciones.