Tema 2. Inserción de Código en Páginas Web

Módulo: Desarrollo Web en Entorno Servidor – 2º DAW

2.1. Introducción

El desarrollo de aplicaciones web del lado del servidor se basa en el uso de **lenguajes de programación y tecnologías específicas**, ejecutados en un servidor para dar servicio a múltiples clientes.

Dependiendo del lenguaje elegido:

- Cambia la **distribución de la lógica de la aplicación** (dónde se ejecutan las distintas partes del software).
- Varía la forma de **interactuar con la información del cliente** (formularios, cookies, sesiones, etc.).
- Se modifica la **gestión del flujo de trabajo** de la aplicación.
- Es diferente la **configuración del entorno del servidor**.

En este tema se estudiarán:

- Tipos de servidores web y sus características.
- Principales lenguajes del lado servidor.
- El flujo de información cliente-servidor.
- Ejemplos prácticos con PHP como lenguaje de referencia.

2.2. Objetivos

1. Reconocer la **arquitectura de aplicaciones web** del lado servidor en función del lenguaje utilizado.
2. Aprender a generar **código dinámico** que será procesado en el servidor y mostrado al cliente en formato HTML.
3. Conocer la **sintaxis y etiquetas propias** de los lenguajes de servidor.
4. Dominar la **declaración de variables**, tipos de datos simples y conversiones entre ellos.
5. Comprender la importancia del **ámbito de las variables** en el desarrollo de aplicaciones.
6. Valorar la relevancia de **separar lógica de negocio y presentación** en las aplicaciones web modernas.

2.3. Lenguajes y Tecnologías de Servidor

- El servidor web es un **programa cuya misión principal es entregar documentos HTML** al cliente.
- Estos documentos no son necesariamente estáticos, pueden contener: enlaces, imágenes, formularios, botones, animaciones, reproductores de sonido o vídeo, etc.
- El **intercambio de datos** entre cliente y servidor se realiza mediante protocolos, principalmente **HTTP**, usando normalmente el puerto 80 (o el 443 si es HTTPS).

Ejemplo práctico:
Cuando escribimos https://www.ieslosremedios.org, el navegador se comunica con el servidor web correspondiente, que procesa la petición, ejecuta código si es necesario (por ejemplo en PHP), y finalmente devuelve un documento HTML que el navegador interpreta y muestra.

2.4. Secuencia de Comunicación Cliente/Servidor

1. El navegador solicita a un servidor DNS la **traducción de la URL** en una dirección IP.
2. Una vez obtenida la IP, se realiza una **petición HTTP** al servidor web.
3. Dado que HTTP es un protocolo **sin estado (stateless)**, cada petición se procesa de forma independiente.
    - Con la versión **HTTP/1.1** se introdujo la posibilidad de conexiones persistentes, reduciendo el número de conexiones necesarias.
4. El servidor procesa la solicitud y, si es necesario, ejecuta código en el lado servidor.
5. El servidor devuelve como respuesta el **documento HTML** al cliente.
6. El navegador interpreta el documento y lo muestra al usuario.

2.5. Protocolo

- Un **protocolo** es un conjunto de reglas que regulan la comunicación entre entidades en una red.
- En Internet, el más utilizado es **TCP/IP**, que incluye diversos subprotocolos:
    - HTTP → transacciones web.
    - FTP → transferencia de archivos.
    - SMTP / POP3 / IMAP → correo electrónico.

El protocolo HTTP

- Pertenece a la **capa de aplicación** de TCP/IP.

- Cada petición HTTP abre una conexión con el servidor que se cierra tras la respuesta.
- Es **sin estado**, por lo que el servidor no recuerda interacciones previas. (Empezamos de nuevo)
- Para mantener información entre peticiones, se usan mecanismos como **cookies**, **sesiones** o **tokens**.

2.6. Clasificación de Servidores Web

Los servidores web se pueden clasificar según cómo procesan las peticiones y optimizan la ejecución:

1. **Basados en procesos**.
2. **Basados en hilos**.
3. **Dirigidos por eventos**.
4. **Implementados en el núcleo del sistema**.

2.6.1. Servidores Basados en Procesos

En este modelo, cada vez que un cliente (navegador) realiza una petición al servidor, el **proceso principal** del servidor crea una **copia de sí mismo** para atenderla.

El concepto de *fork*

- En sistemas operativos tipo UNIX/Linux, la función fork() se utiliza para crear un **nuevo proceso** duplicando el proceso padre.
- Este nuevo proceso es casi idéntico al original (mismo código, mismas variables iniciales), pero tiene su propio espacio de memoria y se ejecuta de forma independiente.
- Gracias al *fork*, el servidor puede atender varias peticiones simultáneamente, ya que cada proceso hijo gestiona de forma aislada una petición concreta.

Ejemplo esquemático

1. El proceso principal escucha en el puerto 80.
2. Llega una petición HTTP de un cliente.
3. El proceso principal ejecuta fork().
4. Se crea un proceso hijo que atiende esa petición.
5. El proceso padre sigue escuchando nuevas conexiones.

Ventajas:

- Seguridad y aislamiento: si un proceso falla, no afecta a los demás.
- Sencillez de implementación.

Inconvenientes:

- Alto consumo de memoria (cada proceso mantiene su propia copia de recursos).
- Menor rendimiento en sistemas con muchas conexiones simultáneas.

2.6.2. Servidores Basados en Hilos

Este modelo es una alternativa más ligera que los procesos. En lugar de crear **un proceso completo** por cada petición, se crean **hilos** dentro de un mismo proceso.

El concepto de *hilo*

- Un **hilo (thread)** es una **unidad de ejecución** dentro de un proceso.
- Varios hilos pueden ejecutarse en paralelo compartiendo la **misma memoria y recursos** del proceso padre.
- Crear un hilo es menos costoso que crear un proceso, ya que no es necesario duplicar toda la memoria ni la información de control del proceso.

Ejemplo esquemático

1. El proceso principal del servidor web recibe una petición.
2. En lugar de hacer un *fork*, crea un **nuevo hilo**.
3. Este hilo se encarga de procesar la petición, mientras otros hilos atienden otras conexiones.

Ventajas:

- Menor consumo de memoria en comparación con procesos.
- Mayor rendimiento en sistemas con muchas peticiones concurrentes.

Inconvenientes:

- **Riesgo de seguridad**: todos los hilos comparten la misma memoria.
    - Si un hilo modifica una variable global, todos los demás verán el cambio.
    - Un error en un hilo puede afectar a todo el servidor.

2.6.3. Servidores Dirigidos por Eventos

Este modelo no utiliza múltiples procesos ni múltiples hilos, sino un **único proceso e hilo** que atiende varias conexiones mediante un sistema de **eventos** y **sockets**.

El concepto de *socket*

- Un **socket** es un punto final de comunicación entre dos programas que se ejecutan en red (cliente y servidor).
- Funciona como un **canal de comunicación** identificado por:

    - Una dirección IP.
    - Un número de puerto.
- Permite que dos programas intercambien datos, incluso si están en máquinas diferentes.

Operaciones asíncronas

- Una operación es **asíncrona** cuando no bloquea la ejecución del programa mientras espera una respuesta.
- En lugar de detenerse, el servidor sigue atendiendo otros eventos y, cuando llega la respuesta, se activa una **notificación** para continuar la ejecución.
- Esto permite manejar miles de conexiones de manera eficiente sin necesidad de un proceso/hilo por cliente.

Comunicación bidireccional

- La comunicación mediante sockets es **bidireccional**:
    - El cliente puede enviar datos al servidor.
    - El servidor puede enviar datos al cliente en la misma conexión.
- Esto resulta esencial en aplicaciones que requieren **interactividad en tiempo real**, como chats, juegos online o servicios de streaming.

Ventajas:

- Gran rendimiento con pocos recursos, especialmente con muchas conexiones simultáneas.
- Ideal para aplicaciones que requieren alta escalabilidad.

Inconvenientes:

- Mayor complejidad de programación.
- La concurrencia es "simulada": aunque parece que atiende a muchos clientes a la vez, en realidad un único hilo va gestionando los eventos de cada socket.

2.6.4. Servidores en el Núcleo del Sistema

- Estrategia avanzada donde parte del servidor web se ejecuta en el **kernel** del sistema operativo.
- Muy eficiente porque elimina pasos intermedios entre aplicación y sistema operativo.

Ventajas:

- Gran rapidez en la gestión de peticiones.
- Reduce la latencia.

Inconvenientes:

- Riesgo muy alto: un fallo en el servidor afecta a todo el sistema operativo.
- Dificultad de configuración y mantenimiento.

2.7. Servidores Web más utilizados

Los servidores web actuales suelen:

- Soportar múltiples lenguajes de programación (PHP, Java, Python, Perl, Ruby...).
- Incluir extensiones y módulos capaces de responder a peticiones de distintos servicios.
- Trabajar con lenguajes **interpretados** y **compilados**.
- Permitir configuración avanzada para optimizar rendimiento y seguridad.

El servidor web no es solo el encargado de servir páginas HTML, también puede:

- Gestionar sesiones de usuarios.
- Controlar accesos mediante autenticación.
- Proporcionar estadísticas y registros (logs) detallados.
- Servir como **proxy inverso** o **balanceador de carga**.

A continuación se describen los servidores más importantes y utilizados en el ámbito profesional.

2.7.1. Apache HTTP Server

- Servidor **multiplataforma** disponible en Windows, Linux, macOS y otros.
- Es de **código abierto** bajo licencia GPL y cuenta con una gran comunidad de desarrolladores.
- Modular y altamente configurable: se pueden habilitar/deshabilitar módulos según las necesidades.
- Puede funcionar tanto **basado en procesos** como **basado en hilos** (mediante diferentes modelos de ejecución llamados *MPM – Multi-Processing Modules*).
- Soporte para:
    - Acceso a bases de datos.
    - Autenticación con usuario y contraseña.
    - Páginas de error personalizadas.
    - Registro de logs en múltiples formatos.
- Seguridad robusta y amplia documentación.

Por defecto se configura en hilos

Ejemplo de uso real:

- Gran parte de los proveedores de hosting compartido utilizan Apache, debido a su flexibilidad y compatibilidad con **PHP y MySQL**, la base de aplicaciones como **WordPress, Joomla o Drupal**.

2.7.2. Microsoft IIS (Internet Information Services)

- Servidor web de Microsoft, integrado en sistemas Windows Server.
- Orientado principalmente a aplicaciones en la plataforma **.NET** y al uso de **ASP (Active Server Pages)**.
- Interfaz gráfica de configuración sencilla e integración con otros servicios de Windows.
- Soporte para añadir módulos que permiten ejecutar otros lenguajes como **PHP**.
- Funciones destacadas:
    - Administración remota.
    - Despliegue de aplicaciones multimedia.
    - Integración directa con Active Directory y otros servicios de red corporativa.

Ejemplo de uso real:

- Empresas que trabajan con soluciones empresariales basadas en **Microsoft .NET** suelen usar IIS para integrar fácilmente aplicaciones web con sus sistemas internos.

2.7.3. Sun Java System Web Server (actualmente parte de Oracle)

- Servidor web de **alto rendimiento, seguro y escalable**.
- Soporta contenido **dinámico y estático**.
- Orientado a entornos empresariales donde se requiere un gran rendimiento con aplicaciones Java.
- Optimizado para ejecutar **JSP (JavaServer Pages)**, **Servlets** y otras tecnologías Java.
- Permite instalar módulos para soportar otros lenguajes: PHP, Python, Ruby, Perl, etc.
- Multiplataforma (Windows, Linux, Solaris...).

Ejemplo de uso real:

- Empresas que tienen aplicaciones críticas desarrolladas en **Java EE** lo utilizan como servidor de aplicaciones para garantizar estabilidad y escalabilidad.

2.7.4. Nginx

- Servidor web ligero y de alto rendimiento.
- Arquitectura **asíncrona basada en eventos**, lo que le permite manejar un gran número de conexiones simultáneas con bajo consumo de memoria.
- Funciona también como **servidor proxy inverso** y **balanceador de carga**.
- Multiplataforma y de **código abierto**.
- Estabilidad y fácil configuración.

Ejemplo de uso real:

- Grandes compañías de Internet lo usan en producción: **Netflix, WordPress, GitHub, Facebook** en determinadas secciones.
- Muchas veces se combina con Apache:
    - Nginx como proxy inverso que recibe todas las peticiones.
    - Apache para procesar las páginas dinámicas.

2.7.5. Lighttpd

- Servidor optimizado para entornos donde se requiere **alta velocidad y bajo consumo de recursos**.
- Menor uso de CPU y memoria que Apache en escenarios con muchas conexiones concurrentes.
- Se distribuye bajo licencia BSD.
- Utiliza un solo proceso con varios hilos, pero **no puede crear nuevos hilos según la demanda**.
- Compatible con Windows y Linux.

Ejemplo de uso real:

- Muy utilizado en aplicaciones web que requieren **gran cantidad de peticiones pequeñas**, como servicios de streaming de música o vídeo.

2.8. Obtención del Código Enviado por el Cliente

En esta sección nos centraremos en **PHP** como ejemplo de lenguaje de servidor.

- PHP (*Hypertext Preprocessor*) es un lenguaje de **scripting del lado del servidor**.
- Código abierto, multiplataforma y con gran comunidad de desarrolladores.
- Puede **insertarse dentro del HTML** para generar contenido dinámico.

- Nació en 1994 y desde su versión 4 (año 2000) alcanzó gran popularidad.
- Su éxito se debe a:
    - Facilidad de aprendizaje.
    - Integración sencilla con bases de datos (especialmente MySQL).
    - Gran cantidad de frameworks (Laravel, Symfony, CodeIgniter).

Ejemplo sencillo de inserción en HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo PHP</title>
</head>
<body>
    <h1>Hola, <?php echo "mundo"; ?>!</h1>
</body>
</html>
```

En este ejemplo, el servidor ejecuta el código PHP y devuelve al navegador únicamente el HTML generado.

2.9. Arquitectura de un Servidor con PHP

Un servidor que ejecute PHP se estructura en varias capas:

1. **Servidor Web**
    - Recibe las peticiones del cliente.
    - Ejemplo: Apache, Nginx.
2. **Capa SAPI (Server Abstraction API)**
    - Intermediario entre PHP y el servidor web.
    - Se encarga de inicializar PHP, gestionar cookies, formularios y datos enviados por POST o GET.
3. **Núcleo PHP**
    - Gestiona la configuración del entorno de ejecución.
    - Define variables globales.
    - Ofrece interfaces para entrada/salida, transformación de datos y carga de extensiones.
4. **Motor Zend**

- Encargado de **analizar, compilar y ejecutar** el código PHP.
- Permite extensiones para modificar su funcionalidad.

2.10. Ejecución de un Script en el Motor Zend

El motor Zend actúa como una **máquina virtual** que procesa los scripts PHP.

Fases principales

1. **Análisis léxico (Lexer)**
   - Convierte el código PHP en un conjunto de **tokens** (piezas básicas).
   - Ejemplo: <?php echo "Hola"; ?> → tokens: echo, "Hola", ;.
2. **Análisis sintáctico (Parser)**
   - Convierte los tokens en un **árbol sintáctico**.
   - Traduce el código a instrucciones intermedias entendibles por la máquina.
3. **Compilación a código intermedio**
   - El árbol sintáctico se traduce a **código intermedio optimizado**.
   - Esta fase facilita la ejecución y permite mejorar el rendimiento.
4. **Ejecución**
   - El **ejecutor de Zend** interpreta las instrucciones intermedias.
   - Se procesan funciones, estructuras de control y operaciones.
   - El resultado final es enviado al navegador como HTML.

Características del motor Zend

- Implementa más de 150 instrucciones específicas y optimizadas.
- Admite extensiones para mejorar el rendimiento (ejemplo: **OPcache** para almacenar scripts compilados en memoria).
