Unidad 7 - Programación de servicios web

Introducción

La **Arquitectura Orientada a Servicios (SOA)** permite el desarrollo de sistemas escalables que facilitan la interacción entre distintas aplicaciones, dentro o fuera de una organización. Los **servicios web** (Web Services) permiten este intercambio de información mediante tecnologías como **SOAP** y **WSDL**.

En las primeras aplicaciones web, su diseño dependía del navegador desde el cual se accedían. Actualmente, el enfoque está en la exposición de contenido a través de servicios web, utilizando estándares como **XML**, lo que facilita la interoperabilidad entre sistemas. Un ejemplo de servicio web simple es **RSS**, que proporciona información actualizada con frecuencia, como noticias o clima.

La arquitectura SOA implica que una aplicación web actúe como cliente de servicios web proporcionados por un servidor. Estos servicios permiten la reutilización de código y la programación modular, independientemente de la tecnología utilizada en cada aplicación.

Arquitectura Web Services y protocolos

La arquitectura Web Services, es un estándar definido por W3C y es el que utilizaremos nosotros.

El estándar de **Web Services**, definido por **W3C**, se basa en varios protocolos esenciales:

- **Intercambio de mensajes →** SOAP (basado en XML)
- **Descripción de servicios →** WSDL (basado en XML)
- **Descubrimiento de servicios →** UDDI (poco utilizado)
- **Comunicación →** HTTP (también FTP, SMTP, etc.)

Implementación de servicios web con PHP

Para desarrollar servicios web en **PHP**, se recomienda usar **NUSOAP**, una API que permite tanto el consumo como la provisión de servicios web.

Pasos para la implementación:

1. Descargar e instalar **NUSOAP** desde SourceForge.
2. Ubicar la carpeta **Lib** en el directorio raíz del proyecto web.
3. Crear un servicio web sencillo en **servicio.php**
4. Generar automáticamente el archivo **WSDL** con la clase `nusoap_server`.
5. Verificar el servicio con herramientas como **wizdl**.

El **WSDL** (Web Services Description Language) describe un servicio web en XML y consta de secciones clave:

- **Descripción:** Define los espacios de nombres.

- **Types:** Define los tipos de datos usados.
- **Message:** Especifica parámetros y respuestas.
- **PortType:** Configura las funciones del servicio.
- **Binding:** Define la transmisión de datos.
- **Service:** Lista las URLs de acceso al servicio.

Una vez probado el servicio, se puede consumir desde otra aplicación mediante un cliente PHP (`cliente.php`).

Symfony - Framework PHP

Symfony es un framework basado en **MVC** que optimiza el desarrollo de aplicaciones web en PHP.

Instalación y configuración

1. Instalar **Composer**, el gestor de dependencias de PHP.
2. Crear un nuevo proyecto Symfony:

```
composer create-project symfony/skeleton .
```

Estructura de un proyecto Symfony

- `bin/` → Ejecutables, incluyendo la consola de Symfony.
- `config/` → Archivos de configuración.
- `migrations/` → Scripts para la base de datos.
- `public/` → Archivos accesibles desde la URL (CSS, JS, etc.).
- `src/` → Código principal (controladores, lógica de negocio).
- `templates/` → Vistas con Twig.
- `var/` → Caché y logs.
- `vendor/` → Librerías gestionadas por Composer.

Rutas y controladores

Las rutas pueden definirse en:

- Archivos `.yaml`
- **Anotaciones** dentro de los controladores

Para crear un controlador, ejecutar:

```
php bin/console make:controller NombreControlador
```

Todas las rutas pasan por el `index.php` principal, actuando como **front controller**.

Gestión de bases de datos

Pasos para la integración con MySQL:

1. Crear la base de datos con codificación **utf8mb4_general_ci**.
2. Mapear las tablas a entidades PHP usando **Doctrine ORM**.
3. Configurar la conexión en el archivo `.env`.
4. Generar entidades automáticamente con:

   ```
   php bin/console make:entity
   ```

5. Agregar métodos `get` y `set` a las entidades.

Plantillas y estilos

Para estructurar la presentación se utiliza **Twig**.

- Se recomienda crear un archivo `base.html.twig` como plantilla principal.
- Los estilos pueden gestionarse con **Bootstrap**:

  ```
  composer require twbs/bootstrap
  ```

Los assets se organizan en la carpeta `public/assets/css/styles.css`.

Dependencias y gestión con Composer

El archivo `composer.json` almacena todas las dependencias del proyecto.

Si se clona un proyecto desde **GitHub**, solo es necesario ejecutar:

```
composer install
```