TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

SISTEMAS DISTRIBUTIVOS

SERVICIOS WEB Y ARQUITECTURA REST

SERVICIOS WEB Y ARQUITECTURA REST

En el contexto de los sistemas distribuidos modernos, los servicios web se han consolidado como una de las herramientas más utilizadas para lograr la interoperabilidad entre aplicaciones heterogéneas a través de una red. Estos servicios permiten que diferentes sistemas intercambien información mediante protocolos estándares, independientemente del lenguaje de programación, sistema operativo o plataforma tecnológica.

Un servicio web expone funcionalidades o recursos a través de una interfaz pública, accesible mediante protocolos de red. Su principal objetivo es permitir la comunicación entre máquinas, ya sea en una red local o a través de Internet, utilizando formatos estructurados como XML o JSON.

Tipos de servicios web

Existen dos estilos arquitectónicos predominantes en el diseño de servicios web:

- **SOAP (Simple Object Access Protocol):** basado en XML, es un protocolo formal con reglas estrictas de contrato y mensajería. Aunque robusto, su complejidad lo ha hecho menos popular en aplicaciones web ligeras.
- **REST (Representational State Transfer):** un estilo arquitectónico más sencillo, ampliamente adoptado por su simplicidad, escalabilidad y compatibilidad con HTTP.

En este desarrollo se hace énfasis en REST, por su relevancia práctica en la construcción de APIs modernas.

1. Arquitectura REST

REST, definido por Roy Fielding en su tesis doctoral en el año 2000, no es un protocolo, sino un conjunto de principios arquitectónicos para diseñar servicios web que aprovechan plenamente las capacidades del protocolo HTTP (Coulouris et al., 2012).

En un sistema RESTful, cada recurso (por ejemplo, usuarios, productos, transacciones) se representa mediante una URL única, y se interactúa con él utilizando los métodos HTTP estándar:

Tabla 1. Métodos HTTP

Método HTTP	Acción asociada	Ejemplo en un recurso productos
GET.	Obtener datos.	GET /productos.
POST.	Crear recurso.	POST /productos.
PUT.	Actualizar recurso.	PUT /productos/123.
DELETE.	Eliminar recurso.	DELETE /productos/123.
DdoS.	Saturación del sistema con tráfico masivo.	Rate limiting, proxies inversos, protección en la nube.

Este modelo convierte a REST en una arquitectura predecible, escalable y sencilla de consumir, incluso por aplicaciones móviles, microservicios y plataformas web.

Principios claves de REST

Para que un servicio se considere verdaderamente RESTful, debe cumplir con varios principios fundamentales:

- Identificación de recursos mediante URIs. Cada recurso tiene una dirección única y estable.
- Uso de métodos HTTP estándar. Cada operación debe corresponder con el verbo HTTP adecuado.
- Representación de recursos. Los recursos pueden representarse en diferentes formatos como JSON, XML o YAML.
- Statelessness (sin estado). Cada solicitud HTTP es independiente. El servidor no almacena contexto entre peticiones del cliente.
- Cacheabilidad. Las respuestas pueden ser almacenadas temporalmente, mejorando el rendimiento.
- Interfaz uniforme. Todos los recursos se acceden de forma homogénea, lo que facilita el uso y desarrollo.

Ejemplo práctico de API RESTful

A continuación, se muestra un ejemplo de cómo se estructuraría un servicio RESTful para una entidad llamada libros:

- Obtener todos los libros: GET /api/libros.
- Obtener un libro específico: GET /api/libros/10.
- Crear un nuevo libro: POST /api/libros.

Figura 1. Body POST libros

```
{
  "titulo": "Cien años de soledad",
  "autor": "Gabriel García Márquez"
}
```

- Actualizar un libro existente: PUT /api/libros/10.
- Eliminar un libro: DELETE /api/libros/10.

Este enfoque permite que los clientes interactúen con los datos de forma coherente, utilizando estructuras uniformes y bien definidas.

Ventajas y desventajas de REST

Ventajas:

- **Ligereza:** utiliza HTTP sin necesidad de protocolos adicionales.
- **Escalabilidad:** ideal para sistemas distribuidos y arquitecturas de microservicios (Coulouris et al., 2012).
- **Independencia del cliente y servidor:** permite múltiples tipos de clientes (web, móviles, IoT).
- **Interoperabilidad:** gracias al uso de formatos estándar como JSON o XML.

Desventajas:

- **Limitado control en operaciones complejas:** no se adapta bien a transacciones múltiples o operaciones de múltiples pasos.
- **Ausencia de contratos formales:** puede dificultar la validación automática sin herramientas adicionales como OpenAPI.
- **Seguridad manual:** REST no incluye un esquema de seguridad predeterminado, se debe integrar con OAuth, JWT, etc.

Tabla 2. REST vs SOAP: Comparación breve

Característica	REST	SOAP
Estilo arquitectónico.	Ligero, basado en HTTP.	Protocolo rígido.
Formato de datos.	JSON, XML, YAML.	XML solamente.
Transacciones.	No soportadas directamente.	Soporta operaciones transaccionales.
Facilidad de consumo.	Alta.	Menor, requiere herramientas.
Rendimiento.	Mejor.	Menor, más pesado.

El enfoque RESTful ha transformado la forma en que se diseñan los servicios web en sistemas distribuidos. Su simplicidad, escalabilidad y compatibilidad con tecnologías modernas lo han convertido en un estándar de facto en la construcción de interfaces entre sistemas (Coulouris et al., 2012). Comprender sus principios y aplicarlos correctamente permite desarrollar soluciones robustas, modulares y orientadas al futuro, fundamentales en entornos distribuidos dinámicos y de gran escala.