3. Servicios Web

Grado en Ingeniería del Software · Ingeniería Web

Resumen con navegación y enlaces cruzados

Índice

1 3.1 Conceptos básicos

1.1 Sistemas distribuidos

Conjunto de agentes software que cooperan y se comunican mediante una pila de protocolos de red. Los Servicios Web (WS) son una forma de organizar estos sistemas. *véase* Arquitecturas de servicios.

1.2 Servicio web (definición W3C)

Sistema software para interacción máquina-a-máquina con interfaz *machine-processable* (WSDL) y mensajes SOAP sobre HTTP con serialización XML.

1.3 Ventajas de WS

Reduce complejidad frente a RPC/RMI/CORBA, mejora interoperabilidad entre lenguajes y evita problemas de firewall usando HTTP/80.

1.4 Anatomía y estructura

Consumidor y proveedor conectados por HTTP: proxies del servicio, interfaz e implementación; comunicación por paso de mensajes sin acoplar conocimientos internos. *véase* HTTP y CRUD.

1.5 Arquitecturas de servicios

Plano conceptual: endpoints y mensajes; plano técnico: **SOAP** y **RESTful**.

1.5.1 Servicios SOAP

Protocolo de intercambio con mensajes XML. Interfaz descrita con **WSDL**.

WSDL Contrato servidor–cliente: operaciones, parámetros, mensajes, ubicación y protocolo.

Listing 1: WSDL: operación suma

```
<definitions ... name=ServicioEjemplo targetNamespace=http://servicio.app/>
  <types>
    <xsd:schema>
      <xsd:import namespace=http://servicio.app/
                  schemaLocation=http://localhost:8080/.../ServicioEjemplo?xsd=1/>
    </xsd:schema>
  </types>
  <message name=suma><part name=parameters element=tns:suma/></message>
  <message name=sumaResponse><part name=parameters element=tns:sumaResponse/></message>
  <portType name=ServicioEjemplo>
    <operation name=suma>
      <input message=tns:suma/><output message=tns:sumaResponse/>
    </operation>
  </portType>
  <binding name=ServicioEjemploPortBinding type=tns:ServicioEjemplo>
    <soap:binding style=document transport=http://schemas.xmlsoap.org/soap/http/>
  </binding>
  <service name=ServicioEjemplo>
    <port name=ServicioEjemploPort binding=tns:ServicioEjemploPortBinding>
      <soap:address location=http://localhost:8080/.../ServicioEjemplo/>
    </port>
  </service>
</definitions>
```

1.5.2 Servicios RESTful

Recursos invocados por URI, con estado representado y transferido como representaciones (JSON, XML, HTML). HTTP se usa como API uniforme.

2 3.2 Arquitecturas REST

2.1 REST: estilo arquitectónico

Definido por Fielding: hipermedia como motor del estado de aplicación, navegación por hiperenlaces entre recursos.

2.2 Recursos y URIs

Identificación con URI; múltiples representaciones y formatos; se opera sobre representaciones, no sobre el recurso directo.

2.3 Principios REST

Cliente–servidor, *stateless*, *cacheable*, interfaz uniforme, sistema en capas, código bajo demanda (opcional) y HATEOAS (opcional). *véase* HATEOAS.

2.4 HTTP como interfaz

Mensajes autodescriptivos, intermediarios posibles, *GET* cacheable.

2.4.1 Métodos HTTP y semántica

- **GET**, **HEAD**, **OPTIONS**: seguros; GET/HEAD idempotentes.
- **POST**: crea en colección.
- **PUT**/**PATCH**: actualizan; idempotencia en PUT.

- **DELETE**: elimina; idempotente.

Evitar usar GET/POST para acciones de otros métodos.

2.4.2 CRUD y rutas REST

Mapeo operaciones–recursos: `GET/POST /recurso`, `GET/PUT/DELETE /recurso/{id}`.

2.4.3 Estructura de mensajes, estados y parámetros

Cabeceras y cuerpo; códigos 2xx/3xx/4xx/5xx; plantillas de URI, *query* y cuerpo para parámetros.

2.4.4 HATEOAS

Respuestas incluyen enlaces para descubrir acciones y navegación. Implementaciones: HAL, JSON-LD, JSON:API.

2.5 Buenas prácticas de APIs REST

Nombres en plural, subrecursos, no usar verbos en la ruta, uso de *query* para filtros, ordenación, campos, paginación; versionado, cabeceras `Content-Type/Accept`, HTTPS y compresión.

2.6 Documentación del API

URI, métodos, parámetros, formatos, códigos y límites; ejemplos de petición y respuesta.

3 3.3 Frameworks Web

Panorama de frameworks REST: Python (Django, Flask, FastAPI), Java (Spring, JAX-RS), Node.js (Express, Fastify), Ruby (Rails), PHP (Laravel).

3.1 FastAPI

ASGI, tipos con anotaciones y pydantic, OpenAPI automático; ejemplo:

Listing 2: FastAPI: endpoint mínimo

```
from fastapi import FastAPI
app = FastAPI()

@app.get(/hola)
async def hola():
    return {message: Hola,␣mundo}
```

3.2 Flask y Jinja2

Flask: microframework WSGI; Jinja2: plantillas con herencia e *HTML escaping*. Incluye ejemplos de rutas, conexión a MongoDB y plantillas con {{ ... }}.

3.3 Django

Full-stack MVC, ORM, enrutado, autenticación.

3.4 Spring / Spring Boot

IoC, beans, acceso a datos, gestión de transacciones; Boot simplifica arranque.

3.5 Express y Laravel

Express: minimalista para Node.js; Laravel: PHP con MVC, Blade, ORM.

4 3.4 Microservicios

4.1 Introducción y problema

De monolito a servicios pequeños, poco acoplados, con comunicación ligera (REST/HTTP).

4.2 Características clave

Despliegue independiente, escalado por servicio, equipos por *dominio de negocio*, cultura DevOps, *producto* vs *proyecto*, diversidad tecnológica, tamaño y número de servicios.

4.3 Ejemplos y casos de éxito

E-commerce, movilidad; Netflix y Amazon como referentes.

4.4 Ventajas e inconvenientes

Modularidad y cohesión vs latencia, consistencia distribuida y complejidad operativa.

4.5 Patrones relacionados

4.5.1 API Gateway

Punto único de entrada, adaptación por cliente, autenticación, monitorización, caché. Riesgo de cuello de botella y punto único de fallo.

4.5.2 Instancia por contenedor

Empaquetar cada servicio en contenedores; réplica sin estado; pros y contras operativos.

5 3.5 OpenAPI y Swagger

5.1 OpenAPI Initiative

Especificación estándar para describir servicios REST en JSON/YAML; origen en Swagger; versión 3.x.

5.2 Estructura básica (YAML)

Listing 3: OpenAPI 3.0: estructura mínima

```
openapi: 3.0.0
info: {version: 1.0.0, title: Sample API, description: ...}
servers: [{ url: https://example.io/v1 }]
paths:
  /items:
    get:
      description: Returns a list of all items
      responses:
        '200': { description: Successful response }
```

5.3 Paths, components y parámetros

Rutas con operaciones y respuestas; esquemas en `components.schemas`; parámetros en `path` y `query`.

5.4 Generación automática y editor

Esqueleto de código desde especificación; frameworks que exponen `/openapi.json` y `/docs`; Swagger Editor.

5.5 Ejemplos en OpenAPI

Añadir ejemplos en YAML o vía decoradores en FastAPI.

6 3.6 JavaScript y Ajax

6.1 Concepto y flujo

Comunicación asíncrona navegador–servidor sin recargar la página.

6.2 Peticiones y respuestas

Listing 4: XMLHttpRequest: POST y GET

```
var x = new XMLHttpRequest();
x.open(POST,https://api/acme/customers/+id,true);
x.setRequestHeader(Content-type,application/x-www-form-urlencoded);
x.send('{nombre:Juan,numeros:[2,4]}');

var y = new XMLHttpRequest();
y.open(GET,https://api/acme/customers/+id,true);
y.setRequestHeader(Accept,application/json);
y.send(null);
```

Listing 5: Manejo de estados

```
x.onreadystatechange = function(){
  if (x.readyState===4 && x.status===200) {
    document.getElementById(myDiv).innerHTML = x.responseText;
  }
};
```

6.3 JSON en Ajax

Listing 6: Parseo de JSON

```
const obj = JSON.parse(myJSONtext); // preferible a eval(...)
```

6.4 Referencias útiles

DOM, eventos, creación de nodos, imágenes.

7 3.7 Mashups e integración de servicios

7.1 Definición y roles

Integración de contenidos/servicios de múltiples proveedores en una sola app; roles: proveedor, componente, API, integrador.

7.2 Tipos

Lado cliente vs lado servidor: ventajas y desventajas de rendimiento, caché, seguridad y concurrencia.

7.3 Mapas y geocodificación

OpenStreetMap con OpenLayers/Leaflet; geocodificación directa e inversa con Nominatim o MapQuest.

7.4 Google Maps APIs

7.4.1 JavaScript API

Listing 7: Inicialización básica de mapa

```
<script src=https://maps.googleapis.com/maps/api/js?v=3.exp&key=API_KEY></script>
<div id=map-canvas style=width:640px;height:480px></div>
<script>
function showMap(){
  const opts = { zoom:16, center: new google.maps.LatLng(36.715219,-4.477676) };
  new google.maps.Map(document.getElementById('map-canvas'), opts);
}
window.onload = showMap;
</script>
```

Marcadores directos o vía GeoJSON; *véase* Ajax para integrar datos dinámicos.

7.4.2 REST, Java y Python

Servicios de geocoding y clientes oficiales.

8 Véase también

- Métodos y semántica HTTP: § 3.2.4.
- Buenas prácticas de diseño de API: § 3.2.1 y § 3.3.
- Documentación y contratos: § 3.5 y § 3.1.5.
- Integración cliente rica: § 3.6 con mapas § 3.7.3.
- Despliegue a escala: § 3.4.