SOA vs. Microservicios. API RESTful y JSON

CÓRDOBA CREA FUTURO

Introducción

Los usuarios, en general, consumimos distintos servicios según la necesidad de cada caso. Por ejemplo, podemos consultar el clima en una ciudad determinada o el estado de vuelo de una aeronave civil. Si deseamos integrar alguno de estos u otros servicios en nuestro propio sistema informático, podemos hacerlo mediante una comunicación con la plataforma que los proporciona. De esta manera, el sistema puede automatizar el procedimiento: el usuario accede a la plataforma y recibe los servicios del clima, del estado de vuelos y de cualquier otro servicio que se haya integrado, mejorando la eficiencia y la experiencia de uso.

Figura 1. Sistema solicitando y consumiendo servicios

Fuente: elaboración propia

En la figura 1, se representa el concepto de una aplicación que consume e integra uno o varios servicios en su sistema. Estos servicios pueden encontrarse en una o varias plataformas, y la forma de solicitarlos se explica en esta lectura.

Una plataforma puede ofrecer uno o varios servicios, con o sin cargo; generalmente, después de contratarlos, el desarrollador los integra a su aplicación mediante un *token* (código *hash*) acompañado de sus credenciales de acceso.

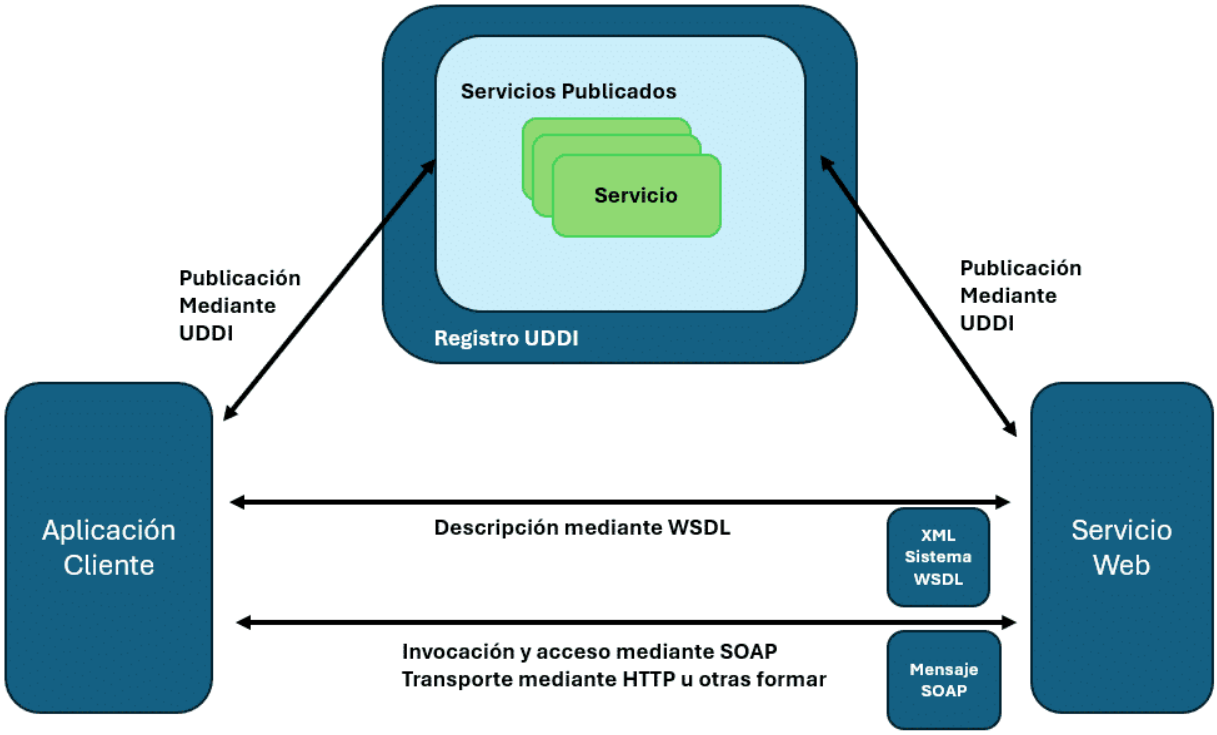
En la figura 2, se ilustra cómo, desde la perspectiva de un desarrollador, se integran componentes de *software* denominados servicios en una aplicación empresarial. Estos servicios se comunican entre sí mediante interfaces de servicio

que utilizan estándares comunes, lo que facilita su integración en nuevas aplicaciones sin afectar al sistema existente.

Este ejemplo describe particularmente la arquitectura SOA (arquitectura orientada a servicios), usando un registro distribuido llamado UDDI (*universal description, discovery, and integration*), que permite publicar y descubrir servicios web para su reutilización, funcionando como un directorio de servicios similar a un directorio telefónico.

La especificación estándar UDDI utiliza XML (lenguaje de marcado extensible) para definir cómo se describen, localizan e integran los servicios web en una arquitectura SOA. El acceso al registro UDDI se puede realizar de varias maneras: a través de la interfaz de usuario del registro, mediante las *interfaces de programación de aplicaciones* (API) o utilizando la API de Java™ para registros XML (JAXR).

Figura 2. Arquitectura distribuida SOA



Fuente: elaboración propia

PARA TERMINAR DE COMPRENDER LA FIGURA 2, SE DEFINEN SEGUIDAMENTE DOS TÉRMINOS:

- ***XML*** es el acrónimo de Lenguaje de Marcado Extensible (Logto blog), una tecnología que usa etiquetas para estructurar, almacenar y transportar datos de forma que sean legibles tanto por humanos como por máquinas. Su principal propósito es facilitar el intercambio de información entre diferentes sistemas, como sitios web y aplicaciones, ya que define cómo se deben marcar los datos y permite que otros sistemas los interpreten correctamente.

- ***SOAP*** (Simple Object Access Protocol) es un protocolo de mensajería para el intercambio de información en servicios web, basado en XML y con reglas de comunicación estrictas. Permite la comunicación entre diferentes sistemas y plataformas, ya que es independiente del lenguaje de programación y del sistema operativo. Utiliza otras tecnologías como HTTP para el transporte de mensajes y está definido por estándares como WSDL y WS-Security

PARA TERMINAR DE COMPRENDER LA FIGURA 2, SE DEFINEN SEGUIDAMENTE DOS TÉRMINOS:

PATRONES DE DISEÑO DE ARQUITECTURA EN CAPAS

SOA (arquitectura orientada a servicios) y los microservicios son dos enfoques que descomponen aplicaciones en componentes. Sin embargo, los microservicios son más pequeños, independientes y específicos, cada uno con su propia base de datos, mientras que SOA tiene un alcance más amplio a nivel empresarial y los servicios suelen estar más acoplados.

En la **tabla 1** se describen las características generales de SOA y microservicios. Se observa que las API RESTful y el formato JSON se utilizan en microservicios —y también en SOA—, ya que REST es un estilo arquitectónico para API que resulta rápido, escalable y flexible

en el formato de datos, mientras que JSON es ligero, eficiente y ampliamente adoptado para el intercambio de información.

Tabla 1. SOA vs. microservicios

Característica	SOA (arquitectura orientada a servicios)	Microservicios
Alcance	Empresarial, más amplio	Aplicación, más específico
Tamaño del servicio	Más grande, con capacidades empresariales completas	Más pequeño, especializado en una sola tarea
Independencia	Los servicios son más acoplados	Servicios independientes, cada uno con su propia base de datos
Comunicación	Puede usar SOAP o REST, a menudo	Usualmente, usa API RESTful, más simples y ligeras

	más complejos (con XML)	
Escalabilidad	Más difícil de escalar	Fácil de escalar debido a su naturaleza sin estado y modularidad

Fuente: elaboración propia

CONTINUAR

1. SOA (arquitectura orientada a servicios)

SOA, o arquitectura orientada a servicios, define una forma de hacer que los componentes de software sean reutilizables e interoperables mediante interfaces de servicio. Los servicios utilizan estándares de interfaz comunes y siguen un patrón arquitectónico que permite incorporarlos rápidamente a nuevas aplicaciones.

Cada servicio en una arquitectura orientada a servicios incluye el código y los datos necesarios para ejecutar una función empresarial completa y específica (por ejemplo, consultar el crédito de un cliente, calcular la cuota mensual de un préstamo o procesar una solicitud de hipoteca). Las interfaces de servicio proporcionan un acoplamiento flexible, lo que significa que pueden invocarse con poco o ningún conocimiento sobre la implementación del servicio subyacente. Esto reduce las dependencias entre aplicaciones.

Esta interfaz funciona como un contrato de servicio entre el proveedor y el consumidor. Las aplicaciones que se

encuentran detrás de la interfaz pueden estar desarrolladas en Java, Microsoft .NET, Cobol u otros lenguajes de programación. También pueden ser soluciones empaquetadas por un proveedor (como SAP), aplicaciones SaaS (como Salesforce CRM) o productos de código abierto.

WSDL

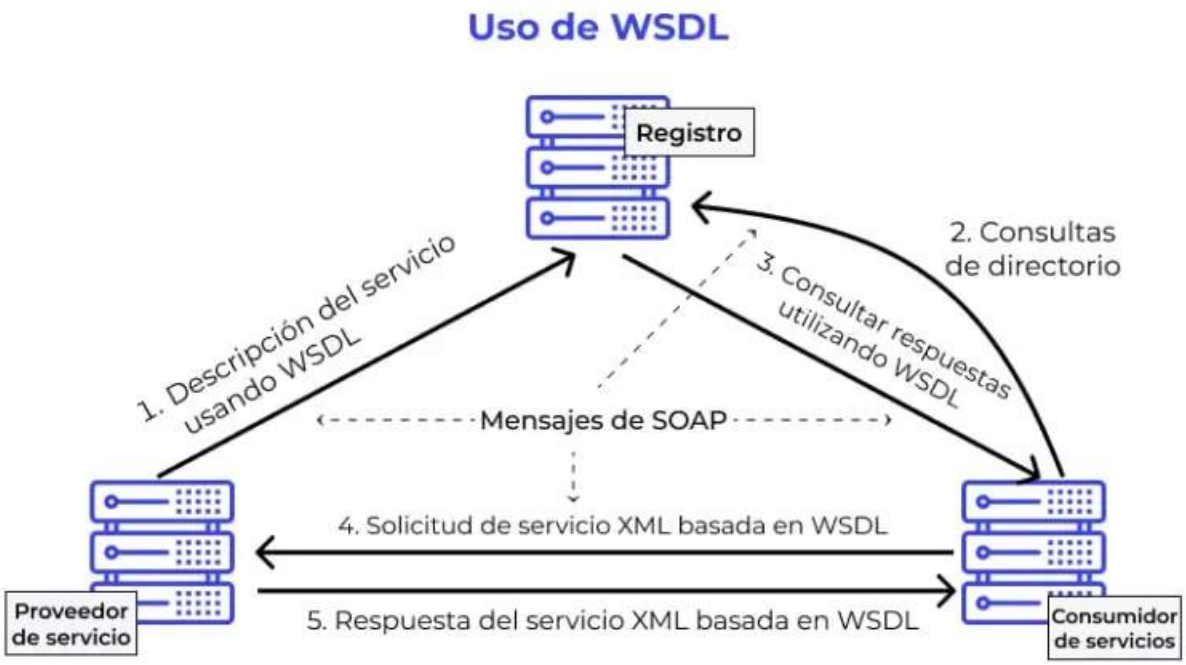
Las interfaces de servicio se definen, con frecuencia, mediante el lenguaje de descripción de servicios web (*WSDL*, por sus siglas en inglés: *web services description language*), una estructura de etiquetas estándar basada en XML (lenguaje de marcado extensible) que describe la funcionalidad de los servicios web, incluyendo cómo interactuar con ellos.

WSDL proporciona una descripción que un cliente puede interpretar para realizar llamadas a un servicio web. Detalla los procedimientos disponibles, los mensajes que deben enviarse y las respuestas esperadas.

Un archivo WSDL funciona como un contrato o manual de instrucciones que el servicio web pone a disposición. Permite a los desarrolladores conocer qué funciones están disponibles en el servidor, qué datos deben enviarse para que una operación se ejecute correctamente y qué formato tendrán los

resultados. Esto resulta fundamental para lograr una integración eficaz entre sistemas dentro de arquitecturas de servicios web.

Figura 3. WSDL

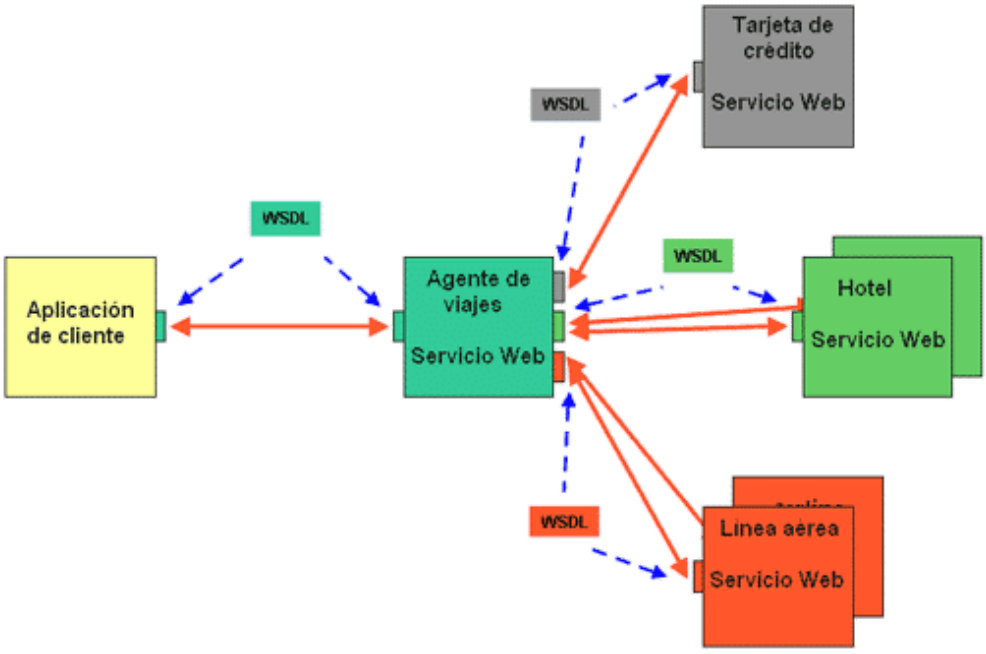


Fuente: Wallarm, s.f., https://short.do/PsXrkh

A continuación, se presenta un ejemplo para ilustrar el funcionamiento de una arquitectura orientada a servicios: un usuario abona en una agencia de viajes un paquete que incluye un boleto aéreo y una reserva de hotel. El proceso

informático que ocurre detrás de esa transacción puede describirse del siguiente modo:

Figura 4. Compra en una agencia de viajes y uso WSDL



Fuente: [imagen sin título sobre compra en una agencia de viajes y uso WSDL], (s.f.), https://short.do/oII6WF

También es posible exponer los servicios mediante protocolos de red estándar, como el protocolo simple de acceso a objetos (SOAP) sobre HTTP o RESTful HTTP con formato JSON, para enviar solicitudes de lectura o modificación de datos.

A continuación, se describen los conceptos mencionados:

SOAP/HTTP —

SOAP y HTTP son tecnologías distintas que suelen utilizarse en conjunto dentro de los servicios web. SOAP (*simple object access protocol*) es un protocolo basado en XML que permite el intercambio de mensajes entre aplicaciones, mientras que HTTP (*hypertext transfer protocol*) es un protocolo de transporte utilizado para enviar y recibir esos mensajes a través de Internet. HTTP funciona con métodos como *GET*, *POST*, *PUT* y *DELETE* para interactuar con recursos web.

SOAP representa el «qué»: define el formato del mensaje y el protocolo para el intercambio de datos. HTTP representa el «cómo»: el medio a través del cual se transportan esos mensajes por la red.

JSON —

JSON (*JavaScript Object Notation*) es un formato de texto ligero y legible para humanos, utilizado para el intercambio de datos. Es independiente del lenguaje de programación y se usa comúnmente en aplicaciones web para enviar información entre un servidor y un navegador. Su simplicidad lo hace comprensible tanto para personas como para máquinas.

JSON representa los datos principalmente de dos formas:

- **Objetos**. Colección de pares clave-valor, definidos entre llaves {}. Cada par tiene una clave (una cadena entre comillas dobles), seguida de dos puntos : y un valor. Los pares están separados por comas.
- ***Arrays***: colección ordenada de valores, definidos entre corchetes [], también separados por comas.

Veamos un ejemplo:

```
{
 "nombre": "Ejemplo",
 "edad": 30,
 "estaActivo": true,
 "hobbies": ["leer", "programar", "viajar"],
 "direccion": {
   "calle": "Calle Falsa 123",
   "ciudad": "Metropolis"
 }
}
```

- **RESTful HTTP (JSON/HTTP)**

Representational state transfer (REST) es un patrón de diseño para interactuar con recursos almacenados en un servidor. Cada recurso

tiene una identidad, un tipo de datos y permite un conjunto definido de acciones. El patrón de diseño RESTful suele utilizarse junto con HTTP, y comúnmente intercambia datos en formato JSON, por su eficiencia y compatibilidad.

WSDL: ejemplo

El lenguaje de descripción de servicios web (*WSDL*, por sus siglas en inglés) permite detallar los protocolos necesarios para describir y localizar servicios web. La versión 2.0 es actualmente la recomendada por el W3C. Este lenguaje está basado en XML y se utiliza en tecnologías de implementación como SOAP para establecer conexiones entre servicios web.
Es importante señalar que WSDL separa la funcionalidad ofrecida por el servicio de sus detalles técnicos, como el protocolo de red o el formato del mensaje (por ejemplo, SOAP, HTTP o MIME). Esta separación corresponde a lo que se denomina «parte abstracta».

La estructura básica de un archivo WSDL contempla las siguientes definiciones:

```
<?xml version=»1.0"?>
<definitions>
  <types> ... </types>
```

```
  <message> ... </message>

  <portType> ... </portType>

  <binding> ... </binding>

</definitions>
```

Tabla 2. Estructura y elementos principales de un archivo WSDL

Elemento WSDL	Descripción
<?xml version=”1.0">	Etiqueta básica de inicio del archivo WSDL, tal como es utilizado en los archivos XML
<definitions>	Inicia del documento y agrupa a todos los demás elementos.
<types>	Tipos de datos de los mensajes.
<message>	Define los métodos y parámetros para ejecutar la operación y puede estar compuesto de cualquier <types>.
<portType>	Aquí se definen las operaciones que pueden ser ejecutadas. Esta es una parte

	muy importante. Además, los mensajes de petición y el de respuesta.
<binding>	Aquí se definen el formato del mensaje y detalles del protocolo para cada *portType*.

Fuente: elaboración propia

A continuación se describen las principales etiquetas que componen un archivo WSDL, junto con sus funciones:

- **<definitions>.** Es la etiqueta raíz del archivo WSDL. Define el nombre del servicio web y el espacio de nombres (*namespace*) asociado. Puede incluir atributos como los siguientes:
  - xmlns. Espacio de nombres del WSDL, por ejemplo: http://schemas.xmlsoap.org/wsdl/
  - name: nombre del servicio definido en el archivo.
- **<types>.** Contiene la definición de los tipos de datos que se utilizan en los mensajes. Puede incluir esquemas XML para describir estructuras complejas.
- **<message>.** Define los mensajes que se intercambian entre el cliente y el servicio. Cada mensaje puede tener una o más partes (parámetros), que se basan en los tipos definidos anteriormente.

- **<portType>.** Describe las operaciones (métodos) disponibles en el servicio web, así como los mensajes de entrada y salida de cada una.
- **<binding>.** Establece el formato y el protocolo de comunicación que se utilizará (por ejemplo, SOAP). Define cómo se deben transmitir los mensajes definidos en el portType.
- **<service>.** Agrupa uno o varios puertos (<port>) que indican dónde está disponible el servicio (*endpoint*). Dentro de <service> podemos encontrar:
    - name. Nombre del servicio.
    - documentation: una descripción opcional del servicio.
    - <port>: especifica la dirección y el protocolo de acceso, por ejemplo:

<soap:address location="http://localhost:8082/MiServicio/Servicio1.wsdl" />
<http:address location="http://localhost:8082/MiServicio/wsdl/Servicio.jsp" />

Figura 5. Ejemplo completo de archivo WSDL con anotaciones explicativas


```
<definitions name="StockQuote"
          targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
             xmlns="http://www.w3.org/2000/10/XMLSchema">
          <element name="TradePriceRequest">
             <complexType>
                 <all>
                     <element name="tickerSymbol" type="string"/>
                 </all>
             </complexType>
          </element>
          <element name="TradePrice">
             <complexType>
                 <all>
                     <element name="price" type="float"/>
                 </all>
             </complexType>
          </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Fuente: [imagen sin título sobre ejemplo de archivo WSDL], (s.f.), https://short.do/7YkkyR

¿Qué es REST?

La transferencia de estado representacional (REST) es una arquitectura de *software* que impone condiciones sobre cómo debe funcionar una API. Brinda visibilidad y portabilidad entre plataformas a cualquier sistema de API.

Los desarrolladores de API pueden diseñar API por medio de varias arquitecturas diferentes. Las API que siguen el estilo arquitectónico de REST se llaman **API REST**. Los servicios web que implementan una arquitectura de REST son llamados servicios web **RESTful**. El término API RESTful suele referirse a las API web RESTful. Sin embargo, los términos API REST y API RESTful se pueden utilizar de forma intercambiable.

SOAP VS. API REST: COMPARACIÓN | REST VS. SOAP: CASOS DE USO

Tanto SOAP como la API REST son tecnologías válidas, cada una con características que permiten su aplicación en distintos casos de uso. Mientras que REST se destaca por ser más flexible, ligera y fácil de implementar, SOAP ofrece mayor robustez, seguridad y confiabilidad, especialmente en entornos empresariales complejos.

A continuación, se presenta una tabla comparativa que resume las principales diferencias entre ambas:

Tabla 3. Comparación entre SOAP y API REST

Características	REST API	SOAP API
Protocolo.	Estilo arquitectónico (utiliza HTTP).	Protocolo estricto (utiliza XML).
Formato de datos	JSON, XML, HTML	Solo XML
Seguridad	Sin seguridad integrada (requiere HTTPS)	Seguridad integrada (WS-Security)
Rendimiento (Performance)	Más rápido y ligero	Más lento debido a la sobrecarga de XML
Escalabilidad	Altamente escalable	Menos escalable
Gestión de errores	Gestión de errores personalizada	Gestión de errores integrada
Flexibilidad	Más flexible, ligero	Más rígido y

		estructurado
Casos de uso	Web, aplicaciones móviles, servicios de IoT	Finanzas, salud, aplicaciones empresariales

Fuente: elaboración propia

La elección entre REST y SOAP depende de los requisitos específicos del proyecto. Mientras que REST es preferido cuando se busca flexibilidad, rapidez y simplicidad, SOAP se destaca en contextos donde se requiere una estructura más formal y altos niveles de seguridad y confiabilidad.

Casos de uso de las API REST

- **Servicios web:** REST es ideal para construir servicios web ligeros y escalables.
- **Aplicaciones móviles:** es ampliamente utilizado por su eficiencia y rapidez en el intercambio de datos.
- **Internet de las cosas (IoT):** su naturaleza sin estado y escalabilidad lo hacen ideal para dispositivos conectados.

- **API públicas:** muchas plataformas, como Twitter y GitHub, utilizan REST para ofrecer interfaces abiertas a desarrolladores externos.

Casos de uso de las API SOAP

- **Servicios financieros:** SOAP es la opción preferida para transacciones bancarias y procesamiento de pagos, donde la seguridad es clave.
- **Sector sanitario:** se utiliza para el intercambio seguro de historiales médicos y datos sensibles.
- **Servicios gubernamentales:** SOAP es común en entornos donde se requiere alta fiabilidad, autenticación y trazabilidad.
- **Aplicaciones empresariales:** los sistemas críticos de gran escala, que necesitan escalabilidad y seguridad avanzada, suelen implementar SOAP.

En resumen, **REST** es más adecuado cuando se busca una solución ligera, flexible y escalable, mientras que **SOAP** es la mejor opción cuando se requiere **mayor seguridad, integridad de los datos y cumplimiento estricto de protocolos**. Conocer las fortalezas y limitaciones de cada enfoque te permitirá tomar una mejor decisión para tu proyecto.

Se sugiere explorar diferentes interacciones en el siguiente enlace: https://www.w3schools.com/js/js_json_syntax.asp. Este recurso permite observar cómo funciona la sintaxis de JSON.

Además, para comprender el proceso de conversión de datos entre un servidor y una aplicación cliente, puede utilizarse el conversor disponible en https://www.site24x7.com/es/tools/json-a-java.html.

En este sitio se visualiza cómo transformar un objeto JSON en una clase Java, como se muestra en la figura 6.

Figura 6. Conversión de un objeto JSON a clase Java utilizando una herramienta en línea

Fuente: captura de pantalla de Site 24x7 (https://www.site24x7.com/es/tools/json-a-java.html)

Asimismo, se sugiere realizar una prueba de conversión utilizando otro conversor disponible en https://json2csharp.com/code-converters/json-to-pojo. Para ello, puede utilizarse el siguiente fragmento de código como ejemplo.

```
{
  "arrayColores":[{
        "nombreColor":"rojo",
        "valorHexadec":"#f00"
    },
    {
        "nombreColor":"verde",
        "valorHexadec":"#0f0"
    },
    {
        "nombreColor":"azul",
        "valorHexadec":"#00f"
    },
    {
        "nombreColor":"cyan",
```

```
        "valorHexadec":"#0ff"
    },
    {
        "nombreColor":"magenta",
        "valorHexadec":"#f0f"
    },
    {
        "nombreColor":"amarillo",
        "valorHexadec":"#ff0"
    },
    {
        "nombreColor":"negro",
        "valorHexadec":"#000"
    }
]
```

¿Qué es un ESB?

Un *enterprise service bus* (ESB), o bus de servicios empresariales, es un patrón de arquitectura de software que actúa como intermediario para facilitar la comunicación y el intercambio de datos en tiempo real entre distintas aplicaciones dentro de una organización. Opera como un componente centralizado encargado de enrutar mensajes,

transformar datos y traducir protocolos, permitiendo que aplicaciones heterogéneas —que suelen emplear distintos formatos y tecnologías— puedan comunicarse de forma eficiente.

Si bien es posible implementar una arquitectura orientada a servicios (SOA) sin un ESB, en ese caso simplemente se contaría con un conjunto de servicios independientes. Cada aplicación tendría que conectarse directamente a los servicios que necesite, gestionando de forma manual las transformaciones de datos requeridas para cumplir con las interfaces de cada uno.

Microservicios

La arquitectura de microservicios es un estilo arquitectónico orientado al desarrollo de aplicaciones. Permite dividir el funcionamiento interno de una aplicación en pequeños componentes, que pueden modificarse, escalarse y administrarse de forma independiente. Este enfoque no define cómo deben comunicarse las aplicaciones entre sí; en ese caso, se retoma el enfoque de nivel empresarial mediante el uso de interfaces de servicio, tal como lo propone la arquitectura orientada a servicios (SOA).

La arquitectura de microservicios surgió y ganó relevancia con el auge de la virtualización, la computación en la nube, el desarrollo ágil

y las prácticas *DevOps*. Gran parte de sus ventajas en estos entornos proviene del desacoplamiento de los componentes, lo que permite optimizar diversos aspectos:

- **Agilidad y productividad para desarrolladores**. Los microservicios permiten incorporar nuevas tecnologías en partes específicas de la aplicación sin afectar al sistema completo. Cada componente puede modificarse, probarse e implementarse de forma independiente, lo que acelera los ciclos de iteración.
- **Escalabilidad**: esta arquitectura aprovecha plenamente la escalabilidad de la nube. Cada microservicio puede escalarse de forma individual para responder de manera más eficiente a la demanda y optimizar el uso de los recursos disponibles.
- **Resiliencia**: el desacoplamiento permite que el fallo de un microservicio no afecte al funcionamiento de los demás. Además, cada componente puede gestionarse con sus propios niveles de disponibilidad, sin necesidad de que toda la aplicación se ajuste a los estándares de disponibilidad más exigentes.

Los microservicios constituyen un enfoque arquitectónico verdaderamente nativo de la nube, que suele operar mediante contenedores. Esto los hace más escalables y portables para la creación de servicios independientes. Al igual que en la arquitectura orientada a servicios (SOA), los sistemas basados en microservicios se componen de componentes especializados, reutilizables y

débilmente acoplados, que por lo general funcionan de manera autónoma.

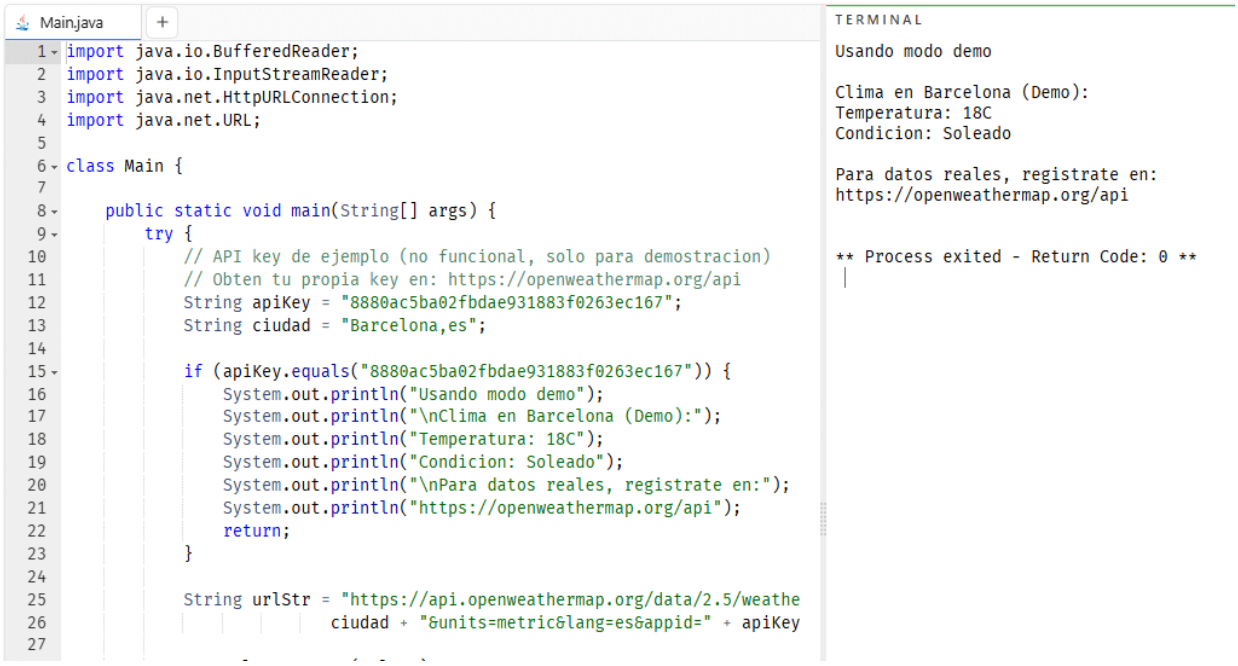
Ejemplo API

Se sugiere acceder al siguiente enlace, donde puede observarse un ejemplo de código en Java que realiza una llamada a un servicio para obtener el estado del tiempo en una ciudad: https://www.online-java.com/VLsuwSj5kz

El ejemplo simula el clima de la ciudad de Barcelona, España. Para que la ejecución sea funcional y en tiempo real, es necesario contratar el servicio correspondiente y reemplazar el token (*API key*) en la línea 12 del código.

Al hacer clic en el enlace, se visualizará el contenido mostrado en la figura 7.

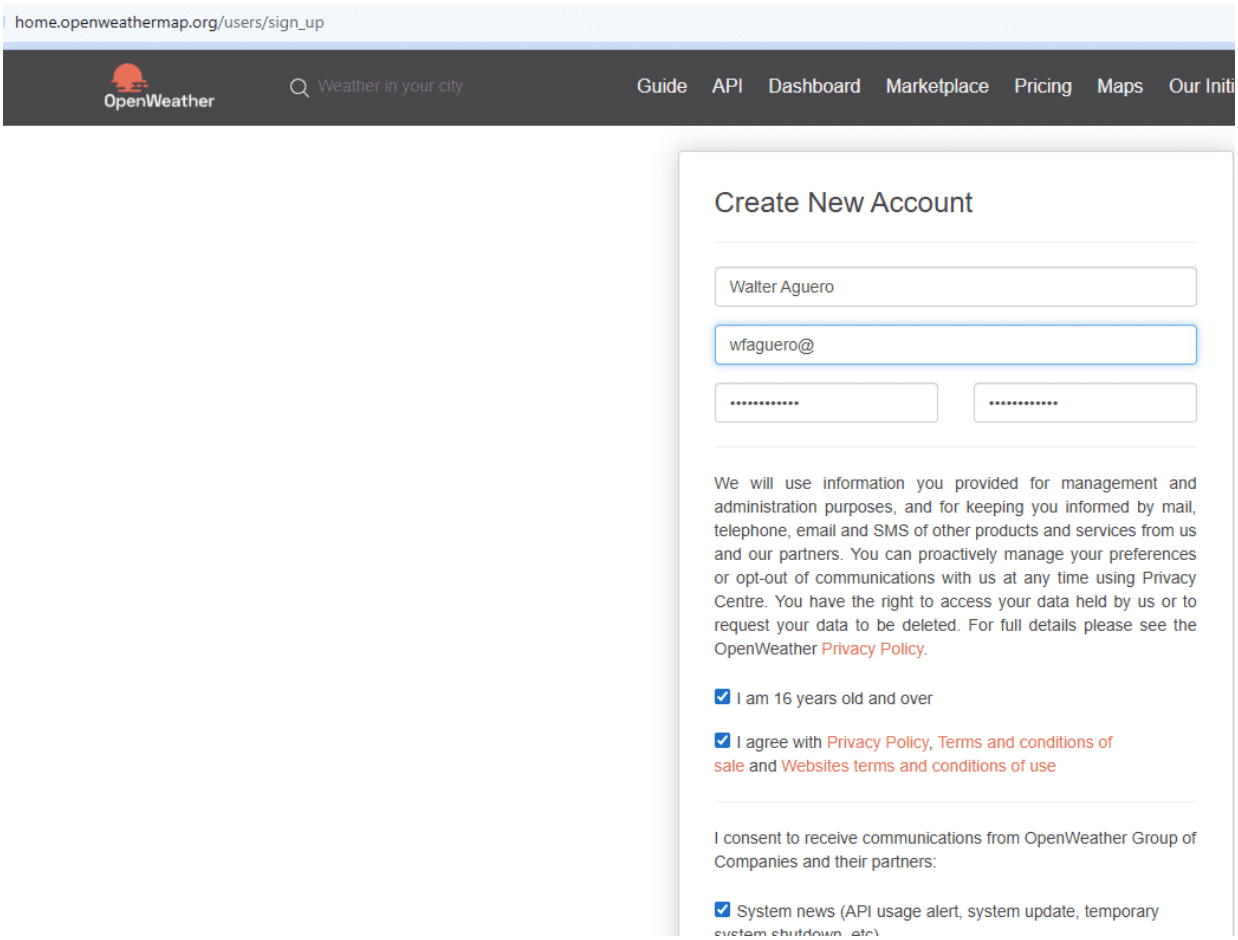
Figura 7. Ejemplo de API



Fuente: captura de pantalla de Online Java (https://www.online-java.com/VLsuwSj5kz)

Para obtener un *token* real, es necesario contratar el servicio correspondiente. En este caso, se puede gestionar el acceso desde el siguiente sitio web: https://home.openweathermap.org/users/sign_up.

Figura 8. Registro de usuario en la plataforma OpenWeather para obtener una API key



Fuente: captura de pantalla de Open weather Map (https://home.openweathermap.org/users/sign_up.)

CONTINUAR

Referencias

Imagen sin título sobre compra en una agencia de viajes y uso WSDL], (s.f.). http://www.hipertexto.info/documentos/serv_web.htm

[Imagen sin título sobre ejemplo de archivo WSDL], (s.f.). https://tecnomaniadsd.wordpress.com/2014/08/06/que-es-wsdl/

Wallarm, (s.f.). *¿Qué es el lenguaje de descripción de servicios web (WSDL)?* https://lab.wallarm.com/what/que-es-el-lenguaje-de-descripcion-de-servicios-web-wsdl/?lang=es

CONTINUAR