­

**Servicios REST**

Luis Alberto García Reyes

# zIntroducción a los servicios REST

Transferencia de estado representacional (en inglés representational state transfer) o REST es un **conjunto de principios arquitectónicos** para diseñar servicios web escalables, ligeros y mantenibles. Se basa en el protocolo HTTP y utiliza los métodos GET, POST, PUT, DELETE y otros para interactuar con los recursos. Algunas de sus características principales son:

* Arquitectura cliente-servidor sin estado.
* Utiliza operaciones estándar de HTTP para interactuar con los recursos (verbos GET, POST, PUT, DELETE).
* Los recursos se identifican mediante URIs.
* Transferencia de representaciones de recursos (generalmente JSON o XML).
* Interfaz uniforme y desacoplada del cliente y servidor.

SOAP (Simple Object Access Protocol) es otro protocolo para servicios web basado en XML. A diferencia de REST, SOAP es un protocolo más complejo y rígido, que **requiere más ancho de banda** y **sobrecarga**. REST, por otro lado, es **más ligero, flexible y escalable**. Además, REST utiliza **directamente los métodos HTTP**, mientras que SOAP los envuelve en un protocolo adicional.

## Ventajas

**Simplicidad y ligereza:** Al utilizar HTTP y formatos de datos comunes (JSON, XML), son más sencillos de implementar y mantener.

**Escalabilidad:** Al ser servicios sin estado, son más escalables.

Independencia de lenguaje y plataforma: Al usar protocolos y formatos estándar, pueden ser consumidos desde diferentes lenguajes y plataformas.

**Cacheable:** Las respuestas pueden ser cacheadas, lo que mejora el rendimiento.

## CASos de uso

* APIs públicas y privadas para aplicaciones web y móviles.
* Integración de sistemas y servicios distribuidos.
* Transferencia de datos en aplicaciones de Internet de las Cosas (IoT).
* Servicios en la nube y arquitecturas de microservicios.

# Fundamentos de HTTP

Los fundamentos HTTP son esenciales para comprender la comunicación entre los servidores y los clientes en una arquitectura basada en REST.

**Los métodos HTTP** son acciones que se pueden realizar en un recurso identificado por una URL. Algunos de los métodos HTTP más comunes son:

* **GET**: Solicita una representación del recurso especificado. Solo recupera datos.
* **POST**: Envía datos para ser procesados por el recurso identificado. Usualmente utilizado para crear nuevos recursos.
* **PUT**: Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la solicitud.
* **DELETE**: Borra el recurso especificado.
* **PATCH**: Aplica modificaciones parciales a un recurso.

**Los códigos de estado HTTP** son indicadores numéricos que se devuelven en las respuestas de las solicitudes HTTP. Algunos códigos de estado comunes son:

* **200 OK**: La solicitud ha tenido éxito.
* **201 Created**: La solicitud ha sido cumplida y se ha creado un nuevo recurso como resultado.
* **400 Bad Request**: La solicitud no se pudo entender debido a una sintaxis incorrecta.
* **401 Unauthorized**: El cliente debe autenticarse para obtener la respuesta solicitada.
* **404 Not Found**: El servidor no pudo encontrar el recurso solicitado.

**Los encabezados HTTP** son información adicional enviada junto con las solicitudes y las respuestas HTTP. Son importantes en el contexto de REST porque permiten especificar metadatos sobre el cuerpo de la solicitud o la respuesta, así como controlar el comportamiento de la comunicación HTTP. Algunos encabezados comunes incluyen:

* **Content-Type:** Indica el tipo de medio del cuerpo del mensaje.
* **Accept:** Indica los tipos de medios que el cliente está dispuesto a recibir.
* **Authorization:** Proporciona credenciales para autenticar al usuario.
* **Cache-Control:** Controla cómo se almacenan en caché las respuestas.

# API REST

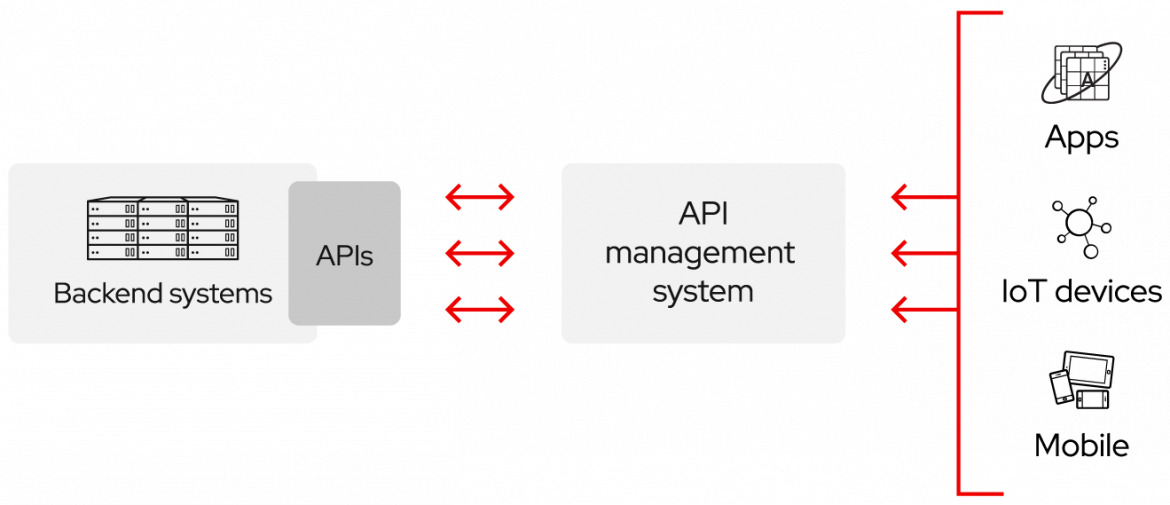
Una API REST (Representational State Transfer) es una **interfaz de programación de aplicaciones** que sigue los principios y convenciones de la arquitectura REST. Se utiliza para permitir la comunicación entre diferentes sistemas de software a través de la web. Una API REST se basa en el protocolo HTTP y utiliza los métodos estándar de HTTP (GET, POST, PUT, DELETE, etc.) para realizar operaciones en recursos.

Las API permiten la comunicación entre productos y servicios sin la necesidad de conocer su **implementación interna**. Esto simplifica el desarrollo de aplicaciones, ahorrando tiempo y dinero. Ofrecen flexibilidad, simplifican el diseño y la gestión de aplicaciones, y fomentan la innovación al facilitar la creación de nuevas herramientas y productos.

A menudo, las API se consideran **contratos**, con documentación que establece un acuerdo entre las partes. Esto define cómo se deben estructurar las solicitudes y respuestas para que el software funcione de manera coherente.

Facilitan la colaboración entre los equipos comerciales y de TI al integrar elementos de nuevas aplicaciones en arquitecturas existentes.

Las API también permiten compartir datos con clientes y usuarios externos. Las API públicas amplían las conexiones con socios y pueden generar ingresos al monetizar los datos (por ejemplo, la API de Google Maps).



# Endpoints

Es una URL que actúa como punto de contacto entre un cliente API y un servidor API. Los clientes API envían solicitudes a los puntos finales de API para acceder a la funcionalidad y los datos de la API.

Una API REST típica tiene muchos **endpoints** que corresponden a sus recursos disponibles. Las solicitudes a los puntos finales de API deben incluir un método que indique la operación a realizar, así como los encabezados, parámetros, credenciales de autenticación y datos del cuerpo necesarios

¿Cómo funcionan?

Los endpoints de una API funcionan conectando clientes y servidores API y manejando la transferencia de datos entre ellos. Un buen diseño de una API debe tener puntos finales claros e intuitivos que proporcionen una forma predecible para que los clientes interactúen con los recursos del servidor.

Diagrama

Descripción generada automáticamente

# Anotaciones en Spring Boot

Las anotaciones en Spring Boot son **etiquetas especiales** que se aplican a clases, métodos o campos para **proporcionar metadatos** adicionales que Spring utiliza para configurar y administrar la aplicación, reduce la necesidad de configuración manual y simplifica el desarrollo de aplicaciones.

Algunas de las anotaciones más comunes en Spring Boot son:

* **@RestController:** Esta anotación es una combinación de @Controller y @ResponseBody. Marca una clase como un controlador especializado en servicios web RESTful. Cada método en un @RestController devuelve directamente un objeto serializado en la respuesta HTTP, en lugar de depender de una vista para generar la respuesta.
* **@Controller:** Marca una clase como un controlador en una aplicación Spring MVC tradicional. Los métodos de un controlador pueden devolver una vista o un objeto que se resolverá en una vista por medio de un ViewResolver.
* **@RequestMapping**: Esta anotación se utiliza para mapear solicitudes HTTP a métodos específicos de un controlador. Puede aplicarse a nivel de clase o de método y permite especificar la URL y el método HTTP que se deben utilizar para acceder a ese método.
* **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping:** Son variantes especializadas de @RequestMapping que especifican métodos HTTP específicos. Por ejemplo, @GetMapping mapea una solicitud HTTP GET, @PostMapping mapea una solicitud HTTP POST, y así sucesivamente. Estas anotaciones proporcionan una forma más específica y concisa de mapear solicitudes HTTP a métodos en un controlador.
* **@RequestBody**: Esta anotación se utiliza para vincular el cuerpo de una solicitud HTTP a un objeto en un método del controlador. Spring Boot se encarga automáticamente de convertir el cuerpo de la solicitud (como JSON o XML) en un objeto Java.
* **@ResponseBody**: Esta anotación se utiliza para indicar que el valor de retorno de un método del controlador debe escribirse directamente en el cuerpo de la respuesta HTTP. Spring Boot se encarga automáticamente de convertir el objeto Java en el cuerpo de la respuesta (como JSON o XML).

# Manipulación de datos

Las operaciones CRUD (Crear, Leer, Actualizar, Eliminar), la interacción con bases de datos y la validación de datos de entrada son algunos ejemplos de la manipulación de datos mas utilizadas con los servicios REST.

## Operaciones CRUD (Crear, Leer, Actualizar, Eliminar):

Estas operaciones son fundamentales en cualquier API REST que trabaje con datos.

* **Create**: Se refiere a la capacidad de la API para agregar nuevos registros a la base de datos.
* **Read:** Implica la capacidad de recuperar información de la base de datos.
* **Update**: Permite modificar registros existentes en la base de datos.
* **Delete**: Permite eliminar registros de la base de datos

## Interacción con bases de datos (JPA, Hibernate, etc.):

* Las API REST generalmente necesitan interactuar con una base de datos para almacenar y recuperar datos.
* Se utilizan tecnologías como JPA (Java Persistence API), Hibernate, o JDBC (Java Database Connectivity) para interactuar con la base de datos.
* Estas tecnologías permiten mapear objetos Java a entidades de base de datos y realizar operaciones CRUD de manera eficiente.

## validación de datos de entrada

Es importante validar los datos que ingresan a la API para garantizar la integridad y seguridad de la aplicación.

Se pueden utilizar diferentes técnicas de validación, como validación de campos obligatorios, validación de formato, validación de rangos, etc.

Frameworks de validación como Hibernate Validator en combinación con anotaciones en las clases de entidad pueden ayudar a implementar la validación de datos de manera efectiva.