

# ➤ La interoperabilidad entre los sistemas

---

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?



# REPASO CLASE ANTERIOR

En la clase anterior trabajamos :

- ✓ *La importancia de la interoperabilidad entre sistemas*
- ✓ *Conceptos e implementación de protocolos de intercambio de datos*

# LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

6.5

Start! 🏁

**La interoperabilidad  
entre los sistemas**

Estado Representacional  
de Transferencia (REST)

RESTful

Qué es el Estado Representacional de Transferencia (REST)

# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?



***Aprender el concepto e implementación de Estado Representacional de Transferencia (REST)***



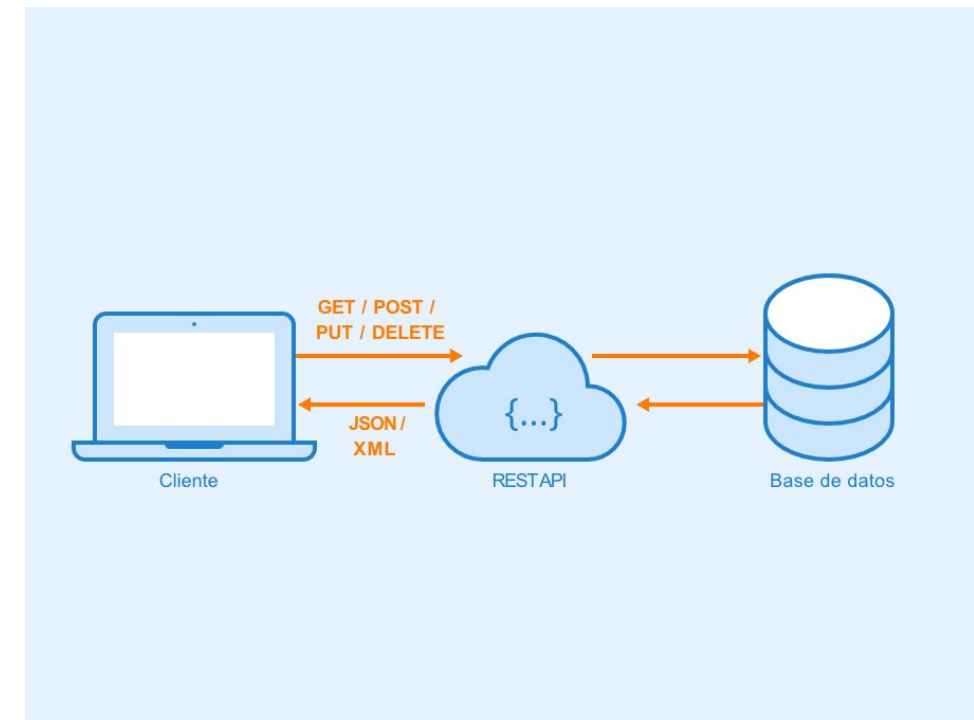
# ➤ Estado Representacional de Transferencia (REST)



# Estado Representacional de Transferencia (REST)



El Estado Representacional de Transferencia, comúnmente conocido como REST, ha emergido como un paradigma fundamental en el diseño de sistemas distribuidos y la construcción de servicios web. Este enfoque arquitectónico, conceptualizado por Roy Fielding en su tesis de doctorado en 2000, ha ganado una amplia aceptación debido a su simplicidad, escalabilidad y capacidad para facilitar la interoperabilidad entre sistemas heterogéneos.



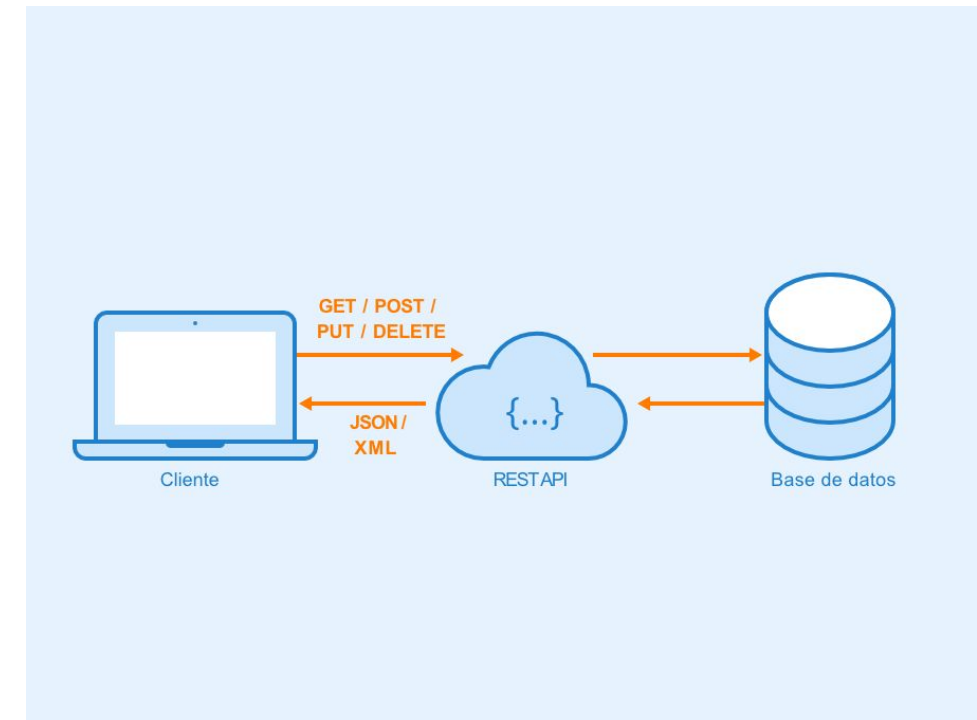


# Estado Representacional de Transferencia (REST)



## Principios de REST

REST es un estilo arquitectónico que guía el diseño de servicios web sencillos, ligeros, almacenables en caché y escalables. Aprovecha todas las ventajas de utilizar el protocolo HTTP y anima a los sistemas a comunicarse a través de una interfaz común simplificada a una estructura URL combinada con métodos HTTP.







# Estado Representacional de Transferencia (REST)



Este protocolo es popular debido a ventajas como la sencillez, la flexibilidad, la escalabilidad y el rendimiento. En lugar de mensajería basada en XML, **REST utiliza una estructura de URL + método HTTP** para lograr algunas ventajas importantes:



- **Ligereza:** no hay largos documentos XML, lo que mejora la eficiencia a través del cable.
- **Almacenamiento en caché:** las cabeceras HTTP permiten el almacenamiento en caché.
- **Escalabilidad:** la naturaleza sin estado mejora la escalabilidad.
- **Flexibilidad:** Son posibles muchos formatos, como JSON, no sólo XML.
- **Compatibilidad heredada:** funciona bien con la arquitectura de Internet existente.

# › Principios REST



# Principios REST

## 1- Separación cliente-servidor



Los sistemas REST **tienen una separación explícita** entre las preocupaciones del cliente y las del servidor.



- **Los clientes no se preocupan por el almacenamiento** de datos o los detalles de implementación del servidor: sólo necesitan saber cómo interactuar con las capacidades del servidor a través de la interfaz.
- **El servidor no se ocupa de las capacidades del cliente** ni de los detalles de implementación: sólo necesita saber cómo satisfacer las peticiones y servir las respuestas correspondientes.



Esto permite que las arquitecturas de servidor y cliente evolucionen de forma independiente **sin un fuerte acoplamiento**.



# Principios REST



## 2- Sin estado

En la web, HTTP es un protocolo sin estado: **el servidor no almacena el estado del cliente entre peticiones.** REST se basa en este concepto y separa aún más el estado del cliente del estado del servidor para aumentar la escalabilidad.



- **El servidor no necesita gestionar sesiones de clientes**, sólo gestionar peticiones individuales de forma aislada. Si el contexto del cliente es necesario para gestionar una solicitud, **el cliente proporciona el contexto con cada solicitud individual.**





# Principios REST



## 3- Almacenamiento en caché

REST pretende aprovechar una de las principales ventajas del uso de HTTP: el almacenamiento en caché **a través de las cabeceras HTTP y los códigos de estado.**



- **Los recursos son identificables únicamente a través de URL**, sin que el contexto del cliente se almacene en el servidor. Esto permite almacenar en caché las respuestas, ya que la misma URL produce la misma respuesta.
- **No es necesario gestionar sesiones de cliente en el servidor**: basta con almacenar en caché las respuestas anteriores para mejorar el rendimiento y la escalabilidad.






# Principios REST

## 4- Sistema por capas



La arquitectura de un sistema REST se compone de capas jerárquicas por función o propósito. Esto permite el equilibrio de carga, el almacenamiento en caché compartido, las capas de seguridad y mejora la flexibilidad y la escalabilidad.



- El nivel de presentación o **capa de interfaz de usuario** se encarga de la interfaz con el usuario final.
- Por debajo de ella, la **capa de aplicación** se encarga de la lógica del proceso funcional y de la validación.
-  Por debajo, el **nivel de datos** se encarga del almacenamiento y el acceso a bases de datos o sistemas de archivos.

**La capa cliente no necesita conocer todas las capas del sistema, sólo la capa inmediata con la que interactúa.**



# Principios REST



## 5- Interfaz uniforme

REST pretende obtener ventajas basándose en el estándar HTTP existente en lugar de crear nuevas pilas de protocolos. Mediante el uso de verbos HTTP, URL, cabeceras y códigos de estado, REST **aprovecha la sintaxis existente del protocolo HTTP en lugar de inventar nuevos estándares basados en XML.**



Esto simplifica la arquitectura general del sistema para ofrecer un modelo de desarrollo unificado en todos los sistemas y la web.



# ➤ Recursos REST






# Recursos REST



**La pieza central de REST son los recursos abstractos** que modelan objetos de información, datos entidades, o conceptos específicos. Un recurso es cualquier cosa que pueda ser nombrada e identificada individualmente. Por ejemplo:



- Un documento o archivo
- Un objeto de datos relacional como una fila en una base de datos
- Un servicio de procesamiento específico
- Una colección o contenedor de otros recursos
-  Una entidad conceptual no textual



# Recursos REST



Cada recurso tiene una identidad única y permanente definida por un identificador universal, típicamente un **URI** (Uniform Resource Identifier).



Por ejemplo, un recurso que modela el concepto de "cliente" en un dominio empresarial determinado puede identificarse de forma única como:

<http://api.domain.com/customers/1234>





# Recursos REST



## Manipulación de recursos a través de representación



Los clientes no acceden directamente a los recursos mismos, sino a una representación temporal de su estado capturado en cierto momento. Las representaciones transfieren el estado en formatos estandarizados como JSON o XML conteniendo atributos de datos u otros metadatos.





# Recursos REST

## Manipulación de recursos a través de representación



Por ejemplo, la representación de JSON para el recurso de cliente anterior puede ser:

```
{  
  "id": 1234,  
  "name": "Juan Perez",  
  "address": "Calle 123"  
}
```



Esto desacopla el recurso en sí del formato de representación específico. Incluso las interacciones en medios no textuales como video o audio se consideran "representaciones" de recurso.



# Recursos REST



## Accediendo recursos a través de identificadores

Los clientes acceden, manipulan y transfieren estas representaciones de recursos de forma uniforme a través de sus identificadores estándares. Esto típicamente utiliza URIs HTTP estándar en combinación con métodos HTTP estándar como GET, POST, PUT, DELETE que manipulan el recurso de forma predecible.





# Recursos REST

## Accediendo recursos a través de identificadores

Las URIs identifican y ubican recursos pero también determinan acción esperada cuando se accede a través de un verbo HTTP:



```
GET http://api.domain.com/customers/1234  
PUT http://api.domain.com/customers/1234  
DELETE http://api.domain.com/customers/1234
```



Estos patrones determinan comportamiento uniforme estándar sin más especificación formal gracias a construcciones web existentes.



# Seguridad en REST

## Seguridad en REST



Como REST opera sobre el protocolo HTTP existente, puede implementar los mismos mecanismos de seguridad subyacentes:

- SSL/TLS para cifrado de tráfico
- Autenticación HTTP para validar usuarios
- Tokens OAuth para delegación de acceso
- Listas de control de acceso para autorizar recursos
- Políticas de uso para prevenir abusos



Al aprovechar totalmente credenciales y controles de seguridad HTTP, no se requieren mecanismos inventados personalizados para proteger una API REST.

# › Ventajas y Desventajas de REST





# Ventajas de REST



REST y sus principios guía ofrecen varias ventajas clave sobre otros estilos arquitectónicos web:



- 1- **Desacoplamiento mejorado:** La separación de preocupaciones entre cliente y servidor sin estado compartido mejora modularidad. Los clientes solo conocen URIs de recursos, pero no conocen la implementación del servidor. Esta abstracción desacopla frontends y backends, permitiendo evolución independiente.
- 2- **Simplicidad inherente:** REST aprovecha standards web asistentes como URIs, HTTP y JSON que son simples, ubicuos y familiares. No se requieren nuevas convenciones o regímenes complejos.



# Ventajas de REST



3- **Entrega optimizada de recursos:** Las URIs REST desreferencian recursos grandes permitiendo entrega en etapas. Las cachés y redes de distribución de contenido optimizan entrega utilizando mecanismos web sin cambios.



4- **Rendimiento y escalabilidad:** Sin estado y almacenamiento en caché inherentes permiten escalar horizontalmente. Los servidores REST simples procesan más solicitudes, más rápido y con menos recursos que los servicios RPC encapsulados.

5- **Portabilidad y longevidad:** REST es independiente del lenguaje y la plataforma, permitiendo integrarse ampliamente entre ecosistemas y generaciones de tecnología a medida que evolucionan.



# Desventajas de REST



Sin embargo, REST también tiene debilidades inherentes a considerar:

- 1- **Complejidad inicial de modelado de recursos:** Modelar todas las capacidades como recursos con representaciones transferibles requiere análisis de dominio avanzado. Es más difícil que simplemente exponer objetos y métodos.
- 2- **No funciona bien para llamadas multidominio:** REST para un solo recurso funciona bien. Pero las operaciones que abarcan múltiples recursos requieren abstracción adicional para coordinar confirmaciones transaccionales.
- 3- **Rigidez en URIs difícil de iterar:** Las URIs que identifican recursos se vuelven frágiles ante cambios. Agregar nuevos parámetros o estructuras de datos requiere nuevas URIs que pueden interferir con el cliente.





# Desventajas de REST



Por lo tanto, si bien REST ofrece ventajas en caching, desacoplamiento, longevidad de APIs y escalamiento horizontal, también viene con un mayor esfuerzo inicial de diseño y limitaciones operativas. Los ingenieros deben sopesar compromisos al elegir un estilo arquitectónicos por sobre otros.



Momento: ✚

# Time-out!

🕒 15 min.





# **Ejercicio N° 1**

# **RESTful**



# Preguntas Frecuentes

## Contexto: 🙌

Vamos a repasar los conceptos más importantes respondiendo algunas preguntas frecuentes sobre el protocolo REST.

## Consigna: ✍️ Respondan levantando la mano!

- 1- ¿Qué significa la abreviatura "REST" y cuáles son sus principios fundamentales?
- 2- ¿Por qué REST se considera "sin estado" y cuáles son las ventajas de esta característica?
- 3- **Investigar:** ¿Cómo se puede implementar la arquitectura por capas en un sistema basado en REST y cuáles son los beneficios de este enfoque?

**Tiempo** 🕒: 15 minutos

○

# ¿Alguna consulta?

+





# RESUMEN

¿Qué logramos en esta clase?

- ✓ **Comprender el concepto y las características del Estado Representacional de Transferencia (REST)**



# #WorkingTime

Continuemos ejercitando

**¡Antes de cerrar la clase!** Te invitamos a: 📌 📌 📌

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
  - a. *Lectura Modulo 6, Lección 4: páginas 3 - 4*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 🙌

