



Modulo 3:

Proyecto de aplicación de consulta de clima, aplicación de consola con consumo de servicios REST externos



Sesión 1.

Fundamentos de servicio REST



REST

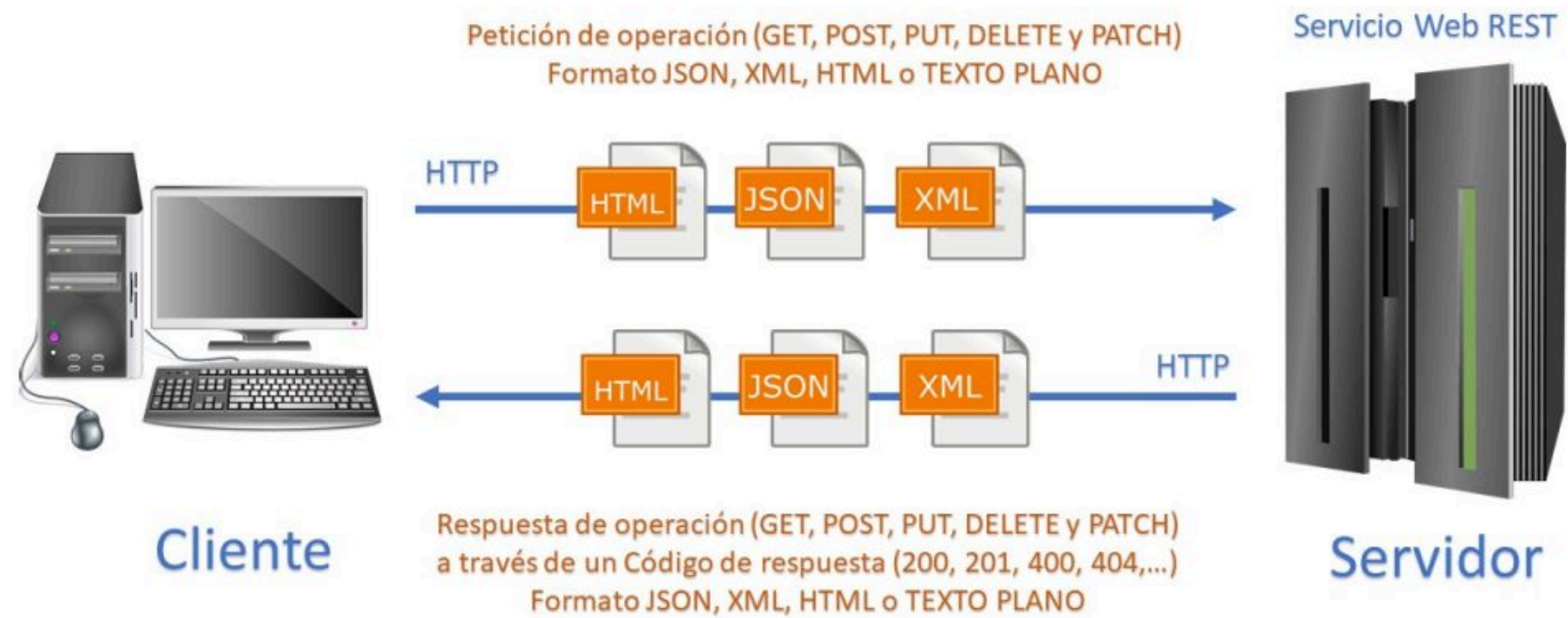
REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP (uno de los protocolos más antiguos) y nos sirve para obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.



REST

El formato más usado en la actualidad es el formato JSON, ya que es más ligero y legible en comparación al formato XML. Elegir uno será cuestión de la lógica y necesidades de cada proyecto.

Esquema Web Service REST





REST

REST deriva de "REpresentational State Transfer", que traducido vendría a ser "transferencia de representación de estado", REST no tiene estado (es stateless), lo que quiere decir que, entre dos llamadas cualesquiera, el servicio pierde todos sus datos. Esto es, que no se puede llamar a un servicio REST y pasarle unos datos (p. ej. un usuario y una contraseña) y esperar que "nos recuerde" en la siguiente petición.

Principios de REST

- Recursos y Representaciones
- Identificación de Recursos
- Mensajes Autocontenidos
- Stateless (Sin Estado)
- Interfaces Uniformes
- Hipermedia como el Motor del Estado de la Aplicación
- Cacheable



Recursos y representaciones




- En REST, un recurso es cualquier cosa que pueda ser nombrada, identificada o direccionada. Por ejemplo, un recurso podría ser un documento, una imagen, un servicio temporal (como una consulta actual del clima), una colección de otros recursos, etc.
- Cada recurso puede tener una o más representaciones. Una representación de un recurso es típicamente un archivo (como JSON, XML, HTML) que muestra el estado actual del recurso. La interacción con un recurso se realiza a través de sus representaciones.



Identificacion de recursos


Los recursos son identificados mediante URLs (Uniform Resource Locators). Cada URL única apunta a un recurso específico. Por ejemplo, `http://api.ejemplo.com/usuarios/123` podría ser la URL que identifica al recurso del usuario con ID 123.





Mensajes autocontenidos


En REST, cada solicitud y respuesta HTTP debe ser autocontenida. Esto significa que cada mensaje debe contener toda la información necesaria para entender la solicitud o respuesta sin depender de un estado previo (statelessness).





Stateless


Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no debe almacenar ninguna información sobre el estado del cliente entre las solicitudes. Esto facilita la escalabilidad del sistema.





Interfaces uniformes

La uniformidad de la interfaz es una de las características clave de REST y significa que hay una forma estándar de interactuar con los recursos, independientemente de las características internas de la implementación. Esta uniformidad se logra a través de:

- a. Identificación de recursos en las solicitudes.
 - b. Manipulación de recursos a través de sus representaciones.
 - c. Mensajes autodescriptivos.
 - d. Hipermedia como el motor del estado de la aplicación (HATEOAS).
- 



Hipermedia como el Motor del Estado de la Aplicación

Un cliente REST interactúa con la aplicación completamente a través de hipermedios proporcionados dinámicamente por las aplicaciones. Esto significa que la navegación y la interacción con la API se basa en enlaces que son parte de las representaciones de los recursos.



Cachable

Las respuestas deben definir explícitamente si se pueden almacenar en caché o no. El uso adecuado de la caché puede mejorar significativamente el rendimiento y la escalabilidad de la aplicación.



METODOS EN REST

GET: Recupera una representación de un recurso. No modifica el recurso y es idempotente (múltiples solicitudes tienen el mismo efecto que una sola solicitud).

POST: Envía datos al servidor para crear un nuevo recurso. Puede modificar el estado del servidor.

PUT: Actualiza un recurso existente o crea uno nuevo si no existe. Es idempotente.

DELETE: Elimina un recurso especificado. Es idempotente.

PATCH: Aplica modificaciones parciales a un recurso.





idempotente


significa que realizar una operación varias veces tendrá el mismo efecto que realizarla una sola vez. En otras palabras, una operación es idempotente si, independientemente de cuántas veces se ejecute, el resultado será siempre el mismo.



Metodos http idempotentes

GET Es idempotente porque una solicitud GET a una URL no cambia el estado del recurso en el servidor. Obtener un recurso varias veces no tiene efectos secundarios adicionales.

PUT Es idempotente porque si envías la misma solicitud PUT varias veces, el estado del recurso será el mismo que si la envías una sola vez. Este método se usa para actualizar o crear un recurso en una ubicación específica.






Metodos http idempotentes

DELETE Es idempotente porque eliminar un recurso varias veces produce el mismo resultado que eliminarlo una sola vez. Si el recurso no existe, el servidor simplemente puede indicar que el recurso no se encuentra.

POST No es idempotente porque una solicitud POST crea un nuevo recurso. Si envías la misma solicitud POST varias veces, se crearán múltiples recursos.





Beneficios de REST

Escalabilidad: Gracias a su arquitectura sin estado y uso de caché, REST es altamente escalable.

Flexibilidad: Puede usar cualquier lenguaje de programación y formato de datos.

Interoperabilidad: Las APIs RESTful pueden ser consumidas por diferentes sistemas y plataformas.



RESTful


Es un adjetivo que describe una aplicación, servicio o sistema que implementa y cumple con los principios y restricciones del estilo arquitectónico REST. En otras palabras, cuando decimos que un servicio es "RESTful", estamos indicando que sigue los principios de REST.

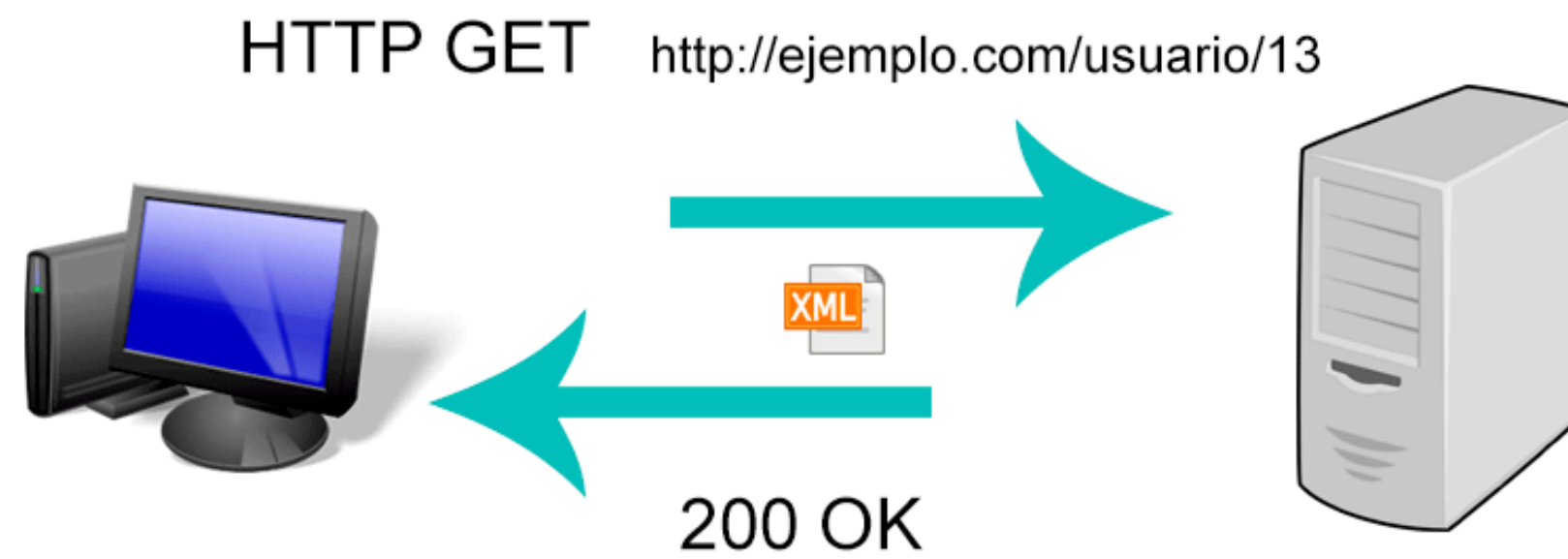
Un API RESTful, por ejemplo, es una interfaz de programación que sigue las reglas y principios de REST. Esto significa que expone recursos que pueden ser manipulados utilizando los métodos HTTP estándar y sigue una estructura predecible y uniforme en sus URLs.



Como entender el estado

En el servicio arquitectonico de REST El servidor no almacena ninguna información sobre el estado de la interacción del cliente entre solicitudes. Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud por sí misma.






El uso de códigos de estado HTTP, como "200 OK", no se refiere al concepto de estado en el contexto de "stateless" en REST. Los códigos de estado HTTP son una forma de que el servidor comuniqué el resultado de una solicitud al cliente.



Codigos de estado HTTP

Los códigos de estado HTTP son respuestas del servidor a las solicitudes del cliente que indican el resultado de la solicitud. Estos códigos son parte del protocolo HTTP y son usados por las APIs RESTful para comunicar si la solicitud fue exitosa, si hubo un error, y de qué tipo.





Codigos HTTP

200 OK: La solicitud fue exitosa y el servidor devuelve la información solicitada.


201 Created: La solicitud fue exitosa y resultó en la creación de un nuevo recurso.

400 Bad Request: La solicitud es inválida o malformada.

401 Unauthorized: La solicitud requiere autenticación.

404 Not Found: El recurso solicitado no se pudo encontrar.

500 Internal Server Error: Hubo un error en el servidor al procesar la solicitud



Conclusion

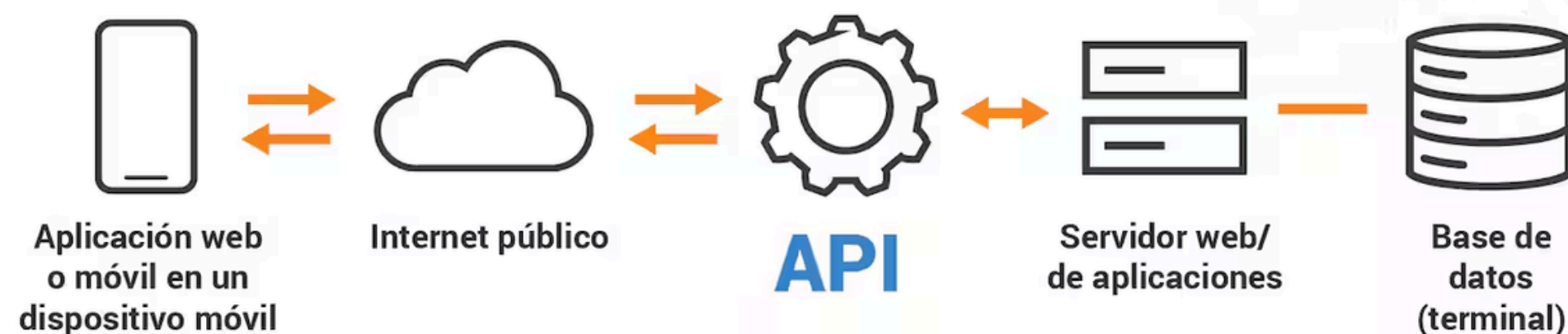
REST es un estilo arquitectónico para diseñar servicios web que utiliza HTTP para interactuar con recursos. Su simplicidad y uso de estándares abiertos lo hacen muy popular para construir APIs escalables y mantenibles. Comprender los principios fundamentales de REST, como la identificación de recursos, el statelessness, y el uso de métodos HTTP, es crucial para diseñar y consumir servicios RESTful de manera efectiva.

API

Una API (Application Programming Interface, o Interfaz de Programación de Aplicaciones) es un conjunto de reglas y definiciones que permite a las aplicaciones comunicarse entre sí. Es una forma estandarizada en la que los programas pueden interactuar y compartir datos y funcionalidades.

API

Una API es una interfaz definida que permite la interacción entre diferentes sistemas de software. Actúa como un intermediario que permite que dos aplicaciones hablen entre sí mediante un conjunto predefinido de reglas.





Componente clave de una API

End Points

Son las URL a las que las solicitudes pueden ser enviadas. Cada end point está asociado con una función específica.

- Ejemplo: <https://api.ejemplo.com/usuarios>
- 