Consumo de API REST

API REST (Parte II)

{desafío} latam_

Construir una aplicación Android que consume un servicio REST actualizando la interfaz de usuario, acorde al lenguaje Kotlin y a la librería Retrofit

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *REST, RESTful, RESTless*
- *Estándares REST para clientes*

¿Has escuchado REST/RESTful anteriormente?
¿Qué crees que significa?

/* REST, RESTful, RESTless */

REST y RESTful

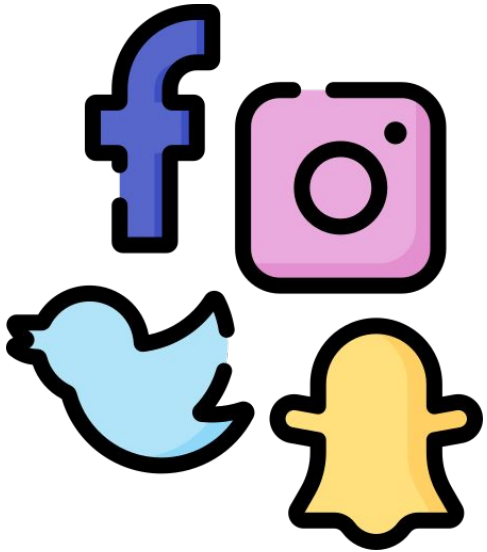
REST (Representational State Transfer) es un estilo arquitectónico para crear servicios web. Un servicio REST es un servicio web que se adhiere a los principios de la arquitectura REST.

Los servicios REST se basan en el protocolo HTTP y están diseñados para funcionar con la infraestructura existente de la web. Utilizan métodos HTTP estándar (como GET, POST, PUT y DELETE) para interactuar con los recursos, que se identifican mediante URI.

En un servicio RESTful, los recursos son el concepto fundamental y se accede a ellos mediante URI. Los recursos se pueden considerar como entidades u objetos que tienen un identificador único y se pueden manipular mediante métodos HTTP estándar.

Los servicios REST también usan códigos de estado HTTP estándar para indicar el resultado de una solicitud y pueden devolver datos en una variedad de formatos, como JSON o XML.

REST y RESTful



- Los servicios REST se pueden usar para crear operaciones CRUD (Crear, Leer, Actualizar, Eliminar) simples en recursos, también se pueden usar para crear sistemas más complejos, como una aplicación de redes sociales que usa servicios RESTful para manejar la autenticación de usuarios, datos almacenamiento y paso de mensajes.

- En general, los servicios REST son livianos, flexibles y escalables y pueden ser utilizados fácilmente por una amplia variedad de clientes, incluidos navegadores web, dispositivos móviles y otros servidores.

- Es importante tener en cuenta que REST no es un protocolo, es un estilo arquitectónico y no existen reglas estrictas para implementar un servicio RESTful, sino un conjunto de pautas y mejores prácticas a seguir.

RESTful - ejemplo

Un ejemplo de un servicio RESTful podría ser un servicio web que permita a los clientes interactuar con una base de datos de libros. El servicio tendría URI que identifican recursos como libros y autores, y usaría métodos HTTP estándar como GET, POST, PUT y DELETE para interactuar con esos recursos.

Por ejemplo, un cliente podría enviar:

- una solicitud `GET` al URI "`https://example.com/books`" para recuperar una lista de todos los libros en la base de datos
- una solicitud GET al URI "`https://example.com/books/123`" para recuperar información sobre un libro específico con ID 123,
- una solicitud POST al URI "https://example.com/books" para agregar un nuevo libro a la base de datos y
- una solicitud DELETE al URI "https ://example.com/books/123" para eliminar un libro específico de la base de datos.

RESTful - ejemplo

El servicio devolvería códigos de estado HTTP estándar, como 200 OK, 201 Creado y 404 No encontrado, para indicar el resultado de la solicitud y devolverá datos en un formato estándar como JSON.

Otro ejemplo podría ser un servicio web que permita a los clientes interactuar con una plataforma de redes sociales, tendría recursos como usuarios, publicaciones y comentarios, y usaría métodos HTTP estándar para interactuar con esos recursos, como GET para recuperar datos. , POST para crear nuevos datos, PUT para actualizar datos y DELETE para eliminar datos.

Es importante tener en cuenta que, estos son solo ejemplos, hay muchas otras formas de implementar un servicio RESTful y la estructura y el comportamiento del servicio pueden variar según el caso de uso y los requisitos.

¿Cuál es la diferencia entre REST y RESTful?

REST y RESTful a menudo se usan indistintamente, pero tienen significados ligeramente diferentes.

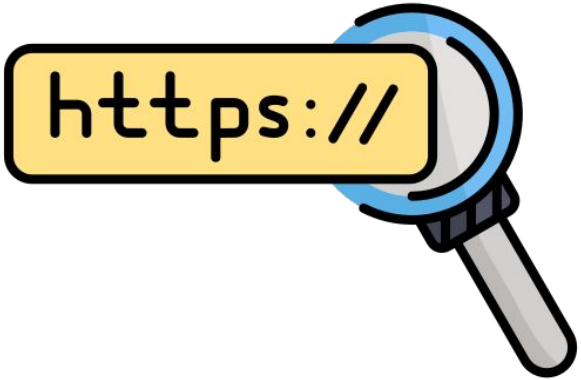
REST (Representational State Transfer) es un estilo arquitectónico para crear servicios web. No es un protocolo ni un estándar, sino un conjunto de pautas y mejores prácticas para crear servicios web que una amplia variedad de clientes pueden utilizar fácilmente. Los servicios RESTful son servicios web que se adhieren a los principios de la arquitectura REST.	RESTful, por otro lado, es un adjetivo que describe un servicio o aplicación que se adhiere a los principios de la arquitectura REST. Un servicio RESTful es un servicio web que se origina empleando los principios de REST. Es un servicio que está diseñado para funcionar con la infraestructura existente de la web y utiliza métodos HTTP estándar (como GET, POST, PUT y DELETE) para interactuar con los recursos, que se identifican mediante URI.

¿Cuál es la diferencia entre REST y RESTful?

En resumen, REST es un estilo arquitectónico, es un conjunto de pautas y mejores prácticas para construir servicios web, mientras que RESTful es un adjetivo que describe un servicio o aplicación que se adhiere a los principios de la arquitectura REST.

Todos los servicios RESTful están basados en REST, pero no todos los servicios basados en REST son RESTful.

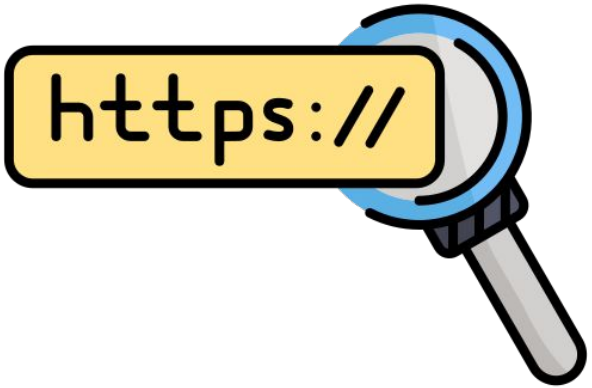
RESTless



Un servicio **RESTless** es un servicio web que no se adhiere completamente a los principios de la arquitectura REST (Representational State Transfer).

Los servicios RESTful son servicios web que se adhieren a los principios de la arquitectura REST, incluido el uso de métodos HTTP estándar (como GET, POST, PUT y DELETE) para interactuar con recursos identificados por URI, devolviendo datos en formato estándar y ser apátrida.

RESTless



Un servicio **RESTless**, por otro lado, puede no seguir todos o algunos de estos principios y puede no ser consumido tan fácilmente por una amplia variedad de clientes. Por ejemplo, un servicio RESTless puede usar métodos no estándar o no devolver datos en un formato estándar. Además, no puede ser apátrida y puede retener información sobre el cliente entre solicitudes.

Es importante tener en cuenta que el término servicio RESTless no se utiliza mucho y no es un término técnico en la arquitectura REST. Sin embargo, puede usarse para describir un servicio que no sigue todos los principios de REST y puede considerarse menos óptimo en comparación con un servicio RESTful.

RESTless - ejemplo

- Un ejemplo de un servicio RESTless podría ser un servicio web que utiliza métodos no estándar para interactuar con los recursos. Por ejemplo, un servicio web que usa un método "SEARCH" personalizado en lugar del método GET estándar para recuperar recursos, dificultará que los clientes comprendan y procesen la solicitud, y podría causar problemas de compatibilidad con otros sistemas que esperan utilizar métodos estándar.

- Otro ejemplo de un servicio RESTless podría ser un servicio web que no devuelva datos en un formato estándar. Por ejemplo, un servicio web que devuelve datos en un formato binario propietario. Esto dificultará que los clientes analicen y comprendan los datos, y podría causar problemas de compatibilidad con otros sistemas que esperan datos en un formato estándar.

- Un servicio RESTless también puede ser uno que no siga el principio de no tener estado y retenga información sobre el cliente entre solicitudes, por ejemplo, usando cookies de sesión o almacenando información del cliente en el lado del servidor. Esto podría causar problemas de escalabilidad y puede dificultar el manejo de varias solicitudes simultáneamente.

Demostración:
“Consumir un servicio REST”

Consumir un servicio REST - Request

Request con Kotlin y OkHttp:

```
val client = OkHttpClient()
val request = Request.Builder()
      .url("https://imdb8.p.rapidapi.com/auto-complete?q=game")
      .get()
      .addHeader("X-RapidAPI-Key", "LxSeM9ihtXmshRiNGpFiSic7OqQmp16KACojsn58F57sZ05HEP")
      .addHeader("X-RapidAPI-Host", "imdb8.p.rapidapi.com")
      .build()

val response = client.newCall(request).execute()
```

Request con curl:

```
curl --request GET \
      --url 'https://imdb8.p.rapidapi.com/auto-complete?q=game' \
      --header 'X-RapidAPI-Host: imdb8.p.rapidapi.com' \
      --header 'X-RapidAPI-Key: LxSeM9ihtXmshRiNGpFiSic7OqQmp16KACojsn58F57sZ05HEP'
```

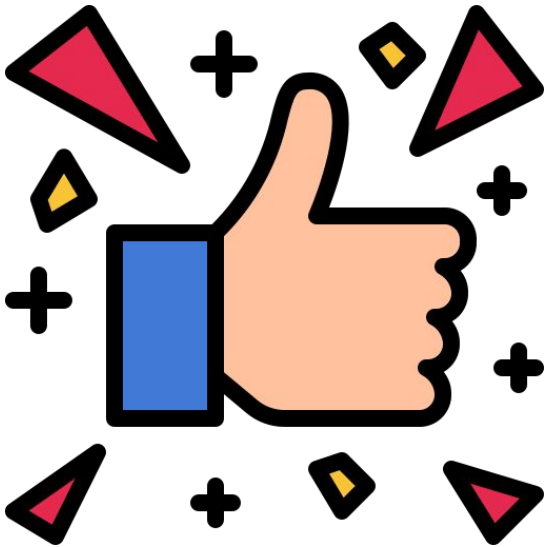
Consumir un servicio REST - Headers

Headers:


```
{
  "access-control-allow-credentials": "true",
  "access-control-allow-headers": "ver",
  "access-control-allow-methods": "GET, POST",
  "access-control-allow-origin": "*",
  "connection": "keep-alive",
  "content-length": "4012",
  "content-type": "application/json",
  "date": "Wed, 16 Feb 2022 10:30:28 GMT",
  "server": "RapidAPI-1.2.8",
  "x-rapidapi-region": "AWS - ap-southeast-1",
  "x-rapidapi-version": "1.2.8",
  "x-ratelimit-requests-limit": "500",
  "x-ratelimit-requests-remaining": "499",
  "x-ratelimit-requests-reset": "2332481"
}
```

/* Estándares REST para clientes */

Estándares REST para clientes



REST (Representational State Transfer) es un estilo arquitectónico para crear servicios web y no tiene estándares específicos para los clientes. Sin embargo, existen algunos principios generales y mejores prácticas que los clientes deben seguir al consumir servicios RESTful.

Siguiendo estos principios, los clientes pueden consumir servicios RESTful de manera consistente y predecible y pueden integrarse fácilmente con diferentes servidores y servicios.

Es importante señalar que estas son recomendaciones generales y algunas de ellas pueden no ser aplicables o factibles en algunos casos.

Recomendaciones

1. Use métodos HTTP estándar: los clientes deben usar métodos HTTP estándar (como GET, POST, PUT y DELETE) para interactuar con los recursos en el servidor. Estos métodos tienen un significado y un comportamiento estándar, lo que facilita que los clientes y los servidores comprendan y procesen las solicitudes.

2. Siga los códigos de estado HTTP estándar: los clientes deben poder interpretar los códigos de estado HTTP estándar devueltos por el servidor, como 200 OK, 201 Creado y 400 Solicitud incorrecta. Estos códigos indican el resultado de la solicitud y brindan información adicional sobre el estado del recurso solicitado.

Recomendaciones

3. Use URI para identificar recursos: los clientes deben usar URI para identificar recursos en el servidor. Los URI deben utilizarse para apuntar a un solo recurso y deben usarse para identificarlo.

4. Utilice formatos de datos estándar: los clientes deben poder procesar los datos devueltos por el servidor en formatos estándar, como JSON o XML. Esto facilita que los clientes analicen y comprendan los datos devueltos por el servidor.

5. Sin estado (Stateless): los clientes no deben conservar ninguna información sobre el servidor entre solicitudes y no deben suponer que el servidor conserva información sobre el cliente.

¿Cuál es la diferencia entre los servicios web REST, RESTful y RESTless? ¿Cómo se relacionan con los principios de statelessness y escalabilidad?

Próxima sesión...

- *Instalar Postman*
- *Hacer requests HTTP usando Postman*
- *Guardar los requests HTTP en Postman*
- *Exportar e importar en Postman*
- *Cómo encontrar APIs REST*

{desafío}
latam_

Academia de
talentos digitales