Metodos HTTP y APIs REST

El acrónimo “REST” significa Transferencia de Estado Representacional. Antes de explicar este término con más detalle, discutamos primero los métodos HTTP y algo de terminología.

En una arquitectura cliente/servidor, las aplicaciones están compuestas por uno o más servicios que residen en los servidores. Estos servicios contienen recursos. El cliente realiza una solicitud de un recurso a través de un objeto de solicitud utilizando una ruta que tiene un punto final dentro del servicio. La aplicación envía un objeto de respuesta de vuelta en respuesta al cliente para honrar esa solicitud.

Un objeto de solicitud contiene tres partes: una URL, un encabezado de solicitud y un cuerpo de solicitud. El servidor utiliza la URL para identificar el servicio y el punto final dentro del servicio que se está utilizando. La URL contiene cuatro partes: un protocolo, un nombre de host, un camino y una cadena de consulta. El encabezado de solicitud contiene metadatos sobre el recurso del cliente que realiza la solicitud, como el agente, host, tipo de contenido, longitud de contenido, y qué tipo de datos el cliente debería esperar en la respuesta.

El servidor responde con un objeto de respuesta que consiste en un encabezado, un cuerpo y un código de estado. El cuerpo del objeto de respuesta a menudo contiene una carga útil en JSON para proporcionar los datos de vuelta al cliente.

Hay una serie de métodos HTTP que se pueden utilizar en la API REST que permiten la interacción entre el cliente y un servicio. Los métodos más comunes son GET, POST, PUT, DELETE y PATCH.

El nombre del método describe lo que sucede con el recurso cuando se aplica el método. Los métodos PUT y DELETE resultan en datos idempotentes si se llama al mismo método API varias veces.

HTTP tiene tres formas de pasar parámetros: el parámetro de ruta de URL, el parámetro de consulta de URL y el parámetro de encabezado. Los parámetros de ruta y consulta se pasan como parte de la URL, pero el parámetro de encabezado se pasa directamente por el navegador al servicio.

Cuando el servicio completa una solicitud, devuelve una respuesta. Un código de estado HTTP debería ser parte de esa respuesta. El código de estado HTTP indica si la respuesta se ha completado o no. Las categorías de códigos de respuesta se muestran en la siguiente tabla.

Rango de Código de Estado	Significado
200-299	Todo está bien
300-399	El recurso se ha movido
400-499	Error del lado del cliente
500-599	Error del lado del servidor

1. **La API aprovecha una arquitectura cliente-servidor compuesta por recursos que son gestionados y entregados a través de HTTP.**
2. **La comunicación entre el cliente y el servidor es sin estado.**
3. **Los datos son cacheables para mejorar el rendimiento del lado del cliente.**
4. **La interfaz se transfiere en un formato estándar de tal manera que los recursos solicitados almacenados en el servidor están separados de la representación enviada al cliente. La representación enviada al cliente contiene datos suficientes para que el cliente pueda manipular la representación.**
5. **Las solicitudes y respuestas se comunican a través de diferentes capas, como middleware. El cliente y el servidor a menudo no se comunican directamente entre sí.**
6. **(Opcional) Los recursos son generalmente estáticos, pero también pueden contener código ejecutable. El código solo debe ejecutarse cuando el cliente lo solicita.**

Un @app.route() método se utiliza al definir un servicio RESTful. Este método toma dos parámetros: la ruta desde la URL al servicio que se está utilizando y un parámetro opcional de método HTTP como POST, GET, etc. El parámetro de ruta puede incluir variables, como <username>. La raíz de una ruta se representa por '/'. Por ejemplo, si deseas que la ruta sea www.mywebsite.com/accounts, solo necesitas especificar '/accounts' como el parámetro de ruta en la función @app.route().

Las APIs REST tienen las siguientes características:

- Basadas en recursos; es decir, describen conjuntos de recursos
- Solo contienen sustantivos, no verbos
- Usan sustantivos en singular al referirse a un recurso singular o sustantivos en plural al referirse a una colección de recursos
- Siempre se identifican por URLs

APIs NO RESTful	Equivalentes RESTful
GET http://api.myapp.com/getUser/123	GET http://api.myapp.com/users/123
POST http://api.myapp.com/addUser	POST http://api.myapp.com/users
GET http://api.myapp.com/removeUser/123	DELETE http://api.myapp.com/users/123

Directrices de formato de URL

- Debe usar una barra '**/**' para denotar una relación jerárquica en la estructura del directorio
- Debe evitar usar una barra final, por ejemplo, **/resource/**
- Debe usar guiones, no camel case, por ejemplo, **/my-resource**, no **/myResource**
- No debe usar un guion bajo '**_**' en la URL, por ejemplo, **/my-resource**, no **/my_resource**
- Debe usar minúsculas
- No debe usar un punto '.' en una URL
- Puede contener múltiples recursos subordinados e IDs en la URL, por ejemplo, **GET /resource/{id}/subordinate/{id}**

Término Descripción

1.Servicio Un componente de software que forma parte de una aplicación y cumple un propósito específico. Generalmente, un servicio toma entrada de un cliente u otro servicio y produce una salida**.**

2.HTTP El protocolo utilizado por una arquitectura cliente-servidor en internet para recuperar o intercambiar datos de recursos.

3.API Una Interfaz de Programación de Aplicaciones es un conjunto de definiciones y protocolos que permiten a dos servicios comunicarse entre sí. La API solicita datos que pueden ser intercambiados entre un recurso y los resultados que se devuelven de ese recurso.

4.REST Un conjunto de directrices arquitectónicas que describen cómo escribir una interfaz (API) entre dos componentes, generalmente un cliente y un servidor, que describen cómo se comunican estos componentes entre sí. REST significa Transferencia de Estado Representacional. REST describe una forma estándar de identificar y manipular recursos. REST asegura que los mensajes enviados entre el cliente y el servidor sean autodescriptivos y definen cómo el cliente interactúa con el servidor para acceder a los recursos en el servidor.

5.Recurso Un recurso es el concepto fundamental de una API RESTful. Es un objeto que tiene un tipo definido, datos asociados, relaciones con otros recursos y un conjunto de métodos que pueden operar sobre él. Un recurso se define comúnmente en formato JSON, pero también puede ser XML.

6.Solicitud Una solicitud es realizada por un cliente a un host en un servidor para acceder a un recurso. El cliente utiliza partes de una URL para determinar la información necesaria del recurso. Los métodos de solicitud más comunes incluyen GET, POST, PUT, PATCH y DELETE, pero también incluyen HEAD, CONNECT, OPTIONS, TRACE y PATCH.

7.Objeto de Solicitud Contiene los datos de la solicitud HTTP. Contiene tres partes: una URL, un encabezado y un cuerpo.

8.Ruta La combinación de un método HTTP y la ruta al recurso desde la raíz de la ruta.

9.Punto final La ubicación del recurso especificado por una API REST que se está accediendo en el servidor. Generalmente se identifica a través de la URL en el método HTTP de la API.

10.Objeto de Respuesta Contiene los datos de respuesta HTTP en respuesta a una solicitud. Contiene un encabezado, un cuerpo y un estado.

11.Respuesta Una respuesta es realizada por un servidor y enviada a un cliente para proporcionar al cliente el recurso solicitado, informar al cliente que la acción solicitada se ha completado, o hacer saber al cliente que ha habido un error al procesar la solicitud.

12.URL Un "Identificador de Recurso Uniforme" se utiliza de manera intercambiable con el término URL. Son parte de una API RESTful que localiza el punto final del recurso solicitado y contiene los datos sobre cómo debe ser manipulado ese punto final. El cliente emite una solicitud HTTP utilizando el URI/URL para manipular el recurso. Deben consistir en cuatro partes: el nombre del host, la ruta, el encabezado y una cadena de consulta.

13.Encabezado de Solicitud Información enviada al servidor sobre el recurso recuperado o el cliente que solicita. Ejemplos incluyen:

Método con punto final: POST /car-reviews

User-agent: El tipo de navegador que está utilizando el cliente.

Host: Una computadora en una red que se comunica con otros hosts.

ContentType: El tipo de medio de un recurso como texto, audio o una imagen.

Longitud de contenido: El número de bytes de datos que se envían en una respuesta.

Accept-Encoding: Formato de datos de retorno esperado, p. ej., application/json

Información de conexión

14.Cuerpo de Solicitud Proporciona los datos que se envían al servidor.

15.Protocolo Indica al servicio cómo se deben transferir los datos entre el servidor y el cliente.

16.Nombre de host El nombre de un dispositivo en una red, también llamado comúnmente nombre del sitio.

17.Ruta La ruta identifica la ubicación del recurso en el servicio y su punto final. Por ejemplo: https://www.customerservice/customers/{customer_id}

18.Cadena de Consulta La parte de una URL que contiene la consulta.

19.User-agent El tipo de navegador que está utilizando el cliente.

20.Host Una computadora en una red que se comunica con otros hosts.

21.Tipo de contenido El tipo de medio de un recurso como texto, audio o una imagen.

22.Longitud de contenido El número de bytes de datos que se envían en una respuesta.

23.Encabezado de Respuesta Contiene metadatos sobre la respuesta, como una marca de tiempo, control de caché, información de seguridad, tipo de contenido y el número de bytes en el cuerpo de la respuesta.

24.Cuerpo de Respuesta Los datos del recurso solicitado se envían de vuelta al cliente.

25.Estado de Respuesta El código de retorno que comunica el resultado del estado de la solicitud al cliente.

26.JSON "Notación de Objetos de JavaScript" es un formato para almacenar y transportar datos, generalmente como una forma de enviar datos desde un servicio en un servidor al cliente. Consiste en pares clave-valor y es autodescriptivo. El formato de los datos JSON es el mismo que el código para crear objetos de JavaScript, lo que facilita convertir estos datos en objetos de JavaScript, pero puede ser escrito en cualquier lenguaje de programación. JSON tiene tres tipos de datos: escalares (números, cadenas, booleanos, nulo), arreglos y objetos.

27.Payload El payload son los datos en el cuerpo de una respuesta que se transportan desde un servidor al cliente debido a una solicitud de API.

28.GET El método GET se utiliza como una solicitud que recupera una representación de un recurso. GET() nunca debe modificar un recurso y solo devolver una representación del recurso solicitado.

29.POST Método HTTP que envía datos al servidor para crear un recurso y debe devolver el código de estado 201_CREATED.

30.PUT Método HTTP que actualiza un recurso o reemplaza uno existente. Llamar a PUT múltiples veces en fila no tiene efectos secundarios, mientras que POST sí. Debe devolver un código 200_OK si el recurso existe y puede ser actualizado o devolver un código 404_NOT_FOUND si el recurso no existe.

31.DELETE Método HTTP que elimina un recurso y devuelve 204_NO_CONTENT si el recurso existe y puede ser eliminado por el servidor o si el recurso no puede ser encontrado, lo que significa que ya ha sido eliminado.

32.PATCH Método HTTP que aplica modificaciones parciales a un recurso.

33.Idempotente Describe un elemento de un conjunto que permanece sin cambios al hacer múltiples solicitudes idénticas. Los métodos PUT y DELETE resultan en datos idempotentes si se llama al mismo método de API múltiples veces.

34.Parámetro de Ruta de URL Pasado a la operación por el cliente como una variable en la ruta de la URL.

35.Parámetro de Consulta de URL Contiene pares clave-valor, generalmente en formato JSON, y se separan de la ruta por un '?'. Si hay múltiples pares clave-valor, deben separarse por un '&'. La consulta puede usarse para pasar un filtro que se aplicará a los resultados devueltos por la operación.

36.Parámetro de Encabezado Contiene metadatos adicionales sobre la consulta, como identificar al cliente que está llamando a la operación.

37.Sin estado Todas las solicitudes de un cliente a un servidor para recursos ocurren de manera aislada entre sí. El servidor no es consciente del estado de la aplicación en el cliente, por lo que esta información debe ser pasada con cada solicitud.

38.Cachable La capacidad de almacenar datos en el cliente para que los datos puedan ser utilizados en una solicitud futura.

39.Middleware Software que se sitúa entre aplicaciones, bases de datos o servicios y permite que esas diferentes tecnologías se comuniquen.