Arquitectura REST

¿Qué es REST?

La transferencia de estado representacional (*representational state transfer*) es un estilo de arquitectura para el diseño de software de servicios web. Básicamente es un conjunto de reglas que buscan que los diferentes sistemas de la web funcionen entre sí.

Los principios de REST son:

1. Arquitectura Cliente-Servidor: la arquitectura debe separar las acciones del cliente y del servidor, ya que ambos son entidades independientes y tienen sus responsabilidades específicas (uno realiza solicitudes y el otro recibe las peticiones).
2. *Stateless*: cada solicitud del cliente al servidor debe contener **toda la información necesaria** para comprender y procesar esta solicitud y el servidor **no debe almacenar ningún estado** (información del cliente, como sus datos de autenticación) sobre el cliente entre las solicitudes.
3. Interfaz uniforme: la interfaz cliente/servidor debe ser coincidente y consistente; es decir, las operaciones sobre los recursos deben ser identificables mediante URL (*Uniform Resource Locators*), los métodos HTTP estándar (`GET`, `POST`, `PUT`, `DELETE`) y la representación de los recursos debe ser coherente, los cuales usualmente se encuentran en formato JSON, XML o YAML.
4. Sistema en capas: esta arquitectura puede ser compuesta por varias capas intermedias entre el cliente y el servidor, lo cual permite la escalabilidad y modularidad del sistema dado a que cada capa puede realizar sus propias funciones que no afecten directamente al cliente o al servidor.
5. Código bajo demanda: los servidores pueden extender o personalizar temporalmente la funcionalidad del cliente transfiriendo a este código de programación del *software*.
6. Almacenamiento en caché: admiten el almacenamiento en caché, que es el proceso de almacenar algunas respuestas en la memoria caché del cliente para mejorar el tiempo de respuesta del servidor.

¿Qué es una API?

Una interfaz de programación de aplicaciones (API) es un mecanismo y también el conjunto de definiciones y protocolos que permite a dos componentes de software comunicarse entre sí sin necesidad de que el usuario que las consume sepa cómo recibe el recurso ni de dónde proviene.

Podemos ver a la API como un mediador entre un usuario y el recurso web que quiere obtener.

¿Qué es una API REST?

También conocida como API RESTful, es una API que sigue los principios y convenciones de REST para darle una funcionalidad. Una API REST se ajusta a los límites de esta arquitectura y permite la interacción con los servicios web.

Los beneficios que ofrece son:

- Escalabilidad: los sistemas que implementan API REST pueden escalar de manera eficiente dadas las ventajas mencionadas de esta arquitectura.
- Flexibilidad: dado el principio Cliente-Servidor, permiten simplificar y desacoplar varios componentes del servidor, lo cual permite evolucionar cada parte de manera independiente sin afectar al cliente ni al servidor como se mencionó anteriormente.
- Independencia: son independientes de la tecnología utilizada. Esto quiere decir que se pueden escribir aplicaciones del lado del cliente y del servidor en diversos lenguajes de programación sin afectar el diseño de la API.

¿Cómo funcionan las API REST?

Cuando un cliente requiere un recurso de un sitio web, primero se pone en contacto con el servidor mediante la API; los desarrolladores de la API consumida explican cómo el cliente debe utilizarla en su documentación; el servidor autentica al cliente y confirma si tiene derecho a obtener la información dada la petición para después procesarla y devolver una respuesta.

Como bien mencionamos antes, las operaciones sobre los recursos son identificables mediante URL, los métodos HTTP (`GET`, `POST`, `PUT`, `DELETE`, entre otros) y la representación de los recursos en formato JSON, XML o YAML.

Dicho ésto, es mediante un *endpoint* (una URL) donde los clientes envían las solicitudes para realizar una acción y obtener una respuesta; en otras palabras, para acceder y utilizar la API REST. Por ejemplo, para solicitar un producto en específico, puede representarse así: "/productos/2", donde estamos pidiendo el segundo producto en la lista de productos.

En cuanto a los métodos HTTP mencionados anteriormente, éstos corresponden a las operaciones básicas del almacenamiento persistente (como las bases de datos), las cuales reciben el nombre CRUD (*Create*, *Read*, *Update* y *Delete*), las cuales hacen lo siguiente:

- `POST`: corresponde a *Create*. Se utiliza para crear un nuevo recurso en el servidor.
- `GET`: corresponde a *Read*. Se utiliza para obtener un recurso disponible en el servidor.
- `PUT`: corresponde a *Update*. Se utiliza para actualizar completamente un recurso en el servidor, donde se espera que el cuerpo de la solicitud tenga la representación completa del recurso. También se suele utilizar el método `PATCH` para este fin, con la diferencia de que actualiza parcialmente un recurso ya existente, es decir, el cuerpo de la solicitud sólo contendrá los cambios concretos que se desean actualizar.
- `DELETE`: corresponde a *Delete*. Se utiliza para eliminar un recurso existente del servidor.

Ya que mencionamos HTTP, también aquí se define un estándar de códigos de estado, el cuál se muestra acorde al resultado de la petición del cliente. Se dividen en cinco categorías:

- 1XX - informativa: la solicitud fue recibida y el servidor la está procesando.

  Ejemplos:

  - 100 (Continuar): el servidor recibió la solicitud y el cliente puede continuar.
  - 101 (Cambiando protocolos): el servidor está cambiando los protocolos acorde a la petición hecha.

- 2XX - exitosa: la solicitud fue recibida y aceptada correctamente.

  Ejemplos:

  - 200 (OK): la solicitud tuvo éxito.
  - 201 (Creado): se ha creado un nuevo recurso de acuerdo a la petición del cliente.

- 3XX - redirección: el cliente debe realizar acciones adicionales para completar la solicitud.

  Ejemplos:

  - 301 (Movido permanentemente): el recurso ha sido movido a una nueva ubicación.
  - 302 (Encontrado): el recurso se ha encontrado temporalmente en otra ubicación.

- 4XX - error del cliente: hubo un error por parte del cliente al realizar la solicitud.

  Ejemplos:

  - 400 (Solicitud incorrecta): la solicitud es incorrecta, por lo cual el servidor no puede procesarla.
  - 403 (Prohibida): el cliente no tiene permiso para acceder al recurso solicitado.
  - 404 (No encontrado): el servidor no puede encontrar el recurso solicitado.

- 5XX - error del servidor: ocurrió un error en el servidor al intentar procesar la solicitud del cliente.

  Ejemplos:

  - 500 (Error interno del servidor): el servidor se encuentra en una situación anormal que le impide cumplir con la solicitud.
  - 503 (Servicio no disponible): el servidor no está listo para manejar la solicitud debido a una sobrecarga temporal o por mantenimiento.

Referencias

- *¿Qué es una API REST?* (s. f.). https://www.redhat.com/es/topics/api/what-is-a-rest-api
- *¿Qué es una API de RESTful? - Explicación de API de RESTful - AWS*. (s. f.). Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/restful-api/
- Wikipedia contributors. (2004, 17 agosto). *REST*. Wikipedia. https://en.wikipedia.org/wiki/REST
- Wikipedia contributors. (2023, 26 octubre). *Create, read, update and delete*. Wikipedia. https://en.wikipedia.org/wiki/Create,_read,_update_and_delete