



API - API-Driven Architecture

13-01-2023

Leandro Baier

Metas

Temas a revisar:

1. Definition of API in IT nowadays.
2. API Application & Resource scope.
3. Place of Consumers in API-driven Architecture.
4. API Versioning Strategies.
5. API Resource Modeling.
6. REST Concepts & Principles.
7. Scope of Consumers in API-driven Architecture.

1. Definition of API in IT nowadays.

Una API (Application Programming Interfaces) es una interfaz que nos permite exponer funcionalidad abstrayendo al cliente de la implementación de la lógica, permitiéndonos encapsular funcionalidades para que sean reutilizables.

2. API Application & Resource scope.

Un recurso es aquello que una API se encarga de administrar, por ejemplo si tengo una colección de autos, para permitirle a los clientes consultar disponibilidad del recurso y administrar los recursos que estos posean podemos exponer funcionalidades como *getCarsByClient* o *buyCar*.

3. Place of Consumers in API-driven Architecture.

Dentro de una arquitectura orientada a API los clientes pueden existir dentro y fuera del ecosistema, podemos tener por ejemplo otras APIs o WebApps que consuman nuestra funcionalidad como podemos consumir internamente funcionalidades entre APIs del mismo ecosistema.

Al día de hoy tenemos tres tipos de clientes: REST, RPC y SOAP.

4. API Versioning Strategies.

Versionado a través de la URI

Se utiliza la URI para indicar la version de la API:

- localhost:8080/api/v1.2/cars/1
- Pros: Los clientes pueden almacenar en caché los recursos fácilmente
- Contras: Esta solución tiene un footprint bastante grande en la base del código, ya que la introducción de cambios importantes implica la ramificación de toda la API.

Versionado a través de Query Parameters

Se envía la versión de la API como query parameter:

- localhost:8080/cars/1?version=1.2
- Pros: Es una forma sencilla de crear versiones de una API, y es fácil cambiar a la última versión por defecto
- Contras: Los parámetros de consulta son más difíciles de usar para enrutar solicitudes a la versión de API adecuada

Versionado a través de Custom Headers

Se utilizan headers para indicar la versión de la API:

- curl -H "version: 1.2" localhost:8080/cars/1
- Pros: No satura el URI con información de versiones
- Contras: Requiere headers custom

Versionado a través de Negociación de Contenido

Se envía la versión de la API a través del contenido esperado:

- curl -H "Accept: application/json; version=1.2" localhost:8080/cars/1
- Pros: Nos permite versionar una sola representación de recursos en lugar de versionar toda la API, lo que nos brinda un control más granular sobre el control de versiones. Crea un footprint más pequeño. No requiere implementar reglas de enrutamiento de URI.
- Contras: Requerir encabezados HTTP con tipos de medios hace que sea más difícil probar y explorar la API usando un navegador

5. API Resource Modeling.

Asdasd


6. REST Concepts & Principles.

Conceptos:

- Cliente: El cliente es un hardware o software que utiliza la API accesible a través de un servidor.
- Recurso: Un recurso puede ser cualquier objeto sobre el que la API pueda ofrecer información.
- Servidor: Un servidor es cualquier sistema que contiene recursos que el cliente desea. Cuando recibe solicitudes de clientes, proporciona el contenido al cliente mediante la interfaz API. El servidor solo otorgará un estado representativo de la fuente y no acceso completo al cliente.

Principios:

- Client-Server: Este principio REST funciona según el concepto de que el cliente y el servidor deben estar aislados entre sí y permitir que se desarrollen de forma independiente.
- Stateless: Las API no tienen estado, lo que significa que las llamadas se pueden realizar de forma independiente. Además, cada llamada incluye los datos esenciales para completarse de manera efectiva.
- Cacheable: Como una API sin estado puede aumentar la sobrecarga de solicitudes mediante la gestión de grandes cargas de llamadas entrantes y salientes, un diseño de API REST debe almacenar datos almacenables en caché. De acuerdo con este principio de diseño de API, los datos dentro de una respuesta deben ser indirectamente o categorizados como almacenables en caché o no almacenables en caché.
- Uniform Interface: Para desacoplar un cliente del servidor, debe tener una interfaz unificada que permita el desarrollo autónomo de la aplicación sin vincular estrechamente sus servicios, modelos y acciones a la propia capa de la API.
- Layered System: Un sistema de arquitectura de API REST en capas tiene una mayor estabilidad porque restringe el rendimiento de los componentes para que cada componente no pueda "ver" más allá de la capa inmediata con la que se entremezcla.
- Code on Demand: Este principio REST permite que la codificación o los applets se comuniquen a través de la API utilizada dentro de la aplicación. Una definición de



API REST permite ampliar la funcionalidad del cliente descargando e implementando codificación en forma de applets o scripts. Esto agiliza a los clientes al disminuir la cantidad de funciones esenciales que deben implementarse previamente. La mayoría de las veces, un servidor devuelve una representación de recursos estáticos en formato XML o JSON. Pero cuando sea necesario, los servidores pueden entregar código ejecutable al cliente.

7. Scope of Consumers in API-driven Architecture.

Independientemente del cliente, el alcance está limitado a la funcionalidad expuesta por la interfaz. El cliente está abstraído de cómo se realiza la solicitud, sólo conoce el request que realiza, y la respuesta que obtiene.