Tanmay Deshpande Seguir
23 de julio de 2018 · 5 min de lectura ★

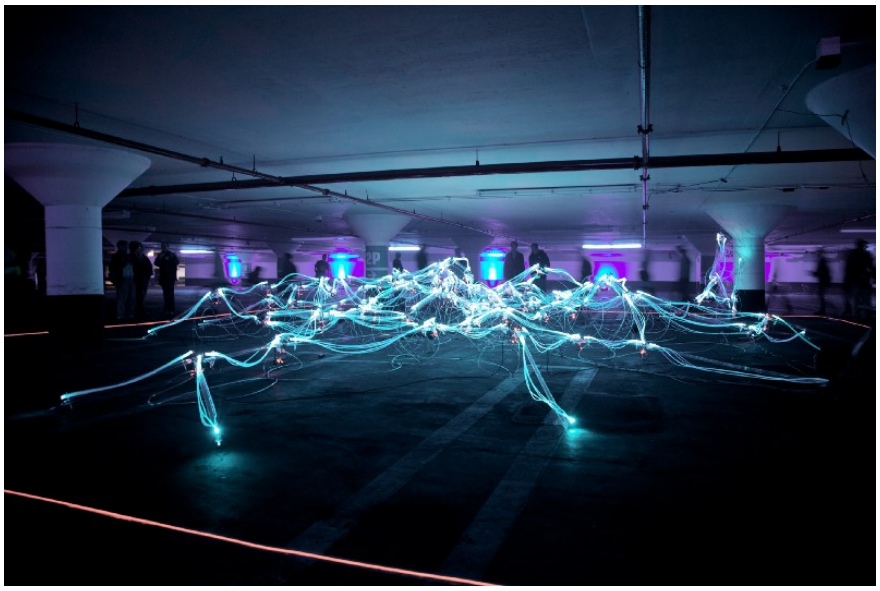

Foto de Marius Masalar en Unsplash.

Como desarrolladores de software, la mayoría de nosotros usamos o creamos API REST en el día a día. Las API son el medio predeterminado de comunicación entre los sistemas. Amazon es el mejor ejemplo de cómo las API pueden usarse de manera eficiente para la comunicación.

En este artículo, voy a hablar sobre cómo diseñar mejor las API REST para evitar errores comunes.

. . .

Mandato de Jeff Bezos (Clave del éxito)

Algunos de ustedes podrían haber estado conscientes del mandato de Jeff Bezos para los desarrolladores en Amazon. Si nunca tuvo la oportunidad de escucharlo, los siguientes puntos son el quid de esto.

1. En adelante, todos los equipos expondrán sus datos y funcionalidad a través de las interfaces de servicio.
2. Los equipos deben comunicarse entre sí a través de estas interfaces.
3. No se permitirá ninguna otra forma de comunicación entre procesos, ni enlaces directos, ni lecturas directas del almacén de datos de otro equipo, ni modelo de memoria compartida, ni puertas traseras de ningún tipo. La única comunicación permitida es a través de llamadas de interfaz de servicio a través de la red.
4. No importa qué tecnología usen. HTTP, Corba, Pubsub, protocolos personalizados, no importa. A Bezos no le importa.
5. Todas las interfaces de servicio, sin excepción, deben diseñarse desde cero para ser externalizables. Es decir, el equipo debe planificar y diseñar para poder exponer la interfaz a los desarrolladores en el mundo exterior. Sin excepciones.
6. **Quien no haga esto será despedido.**

Eventualmente, esto resultó ser la clave para el éxito de Amazon. Amazon podría crear sistemas escalables y, posteriormente, también podría ofrecerlos como servicios como los servicios **web de Amazon** .

. . .

Principios de diseño de APIs RESTful

Ahora entendamos los principios que debemos seguir al diseñar las API RESTful.

Mantenlo simple

Necesitamos asegurarnos de que la URL base de la API es simple. Por ejemplo, si queremos diseñar API para productos, debería diseñarse como:

```
/ productos
/ productos / 12345
```

La primera API es obtener todos los productos y la segunda es obtener un producto específico.

Usa sustantivos y no los verbos.

Muchos desarrolladores cometen este error. En general, se olvidan de que tenemos métodos HTTP para describir mejor las API y terminamos usando verbos en las URL de la API. Por ejemplo, la API para obtener todos los productos debe ser:

```
/ productos
```

y *no* como se muestra abajo

```
/ getAllProducts
```

Algunos patrones de URL comunes, que he visto hasta ahora.

Uso de los métodos HTTP correctos

Las API RESTful tienen varios métodos para indicar el tipo de operación que vamos a realizar con esta API.

- **GET** - Para obtener un recurso o colección de recursos.
- **POST** - Para crear un recurso o colección de recursos.
- **PUT / PATCH** - Para actualizar el recurso existente o la colección de recursos.
- **ELIMINAR** : para eliminar el recurso existente o la colección de recursos.

Debemos asegurarnos de que usamos el método HTTP correcto para una operación determinada.

Usar plurales

Este tema es un poco discutible. A algunas personas les gusta mantener la URL del recurso con nombres plurales, mientras que a otras les gusta mantenerla singular. Por ejemplo -

```
/ productos
```

```
/producto
```

Me gusta mantenerlo en plural, ya que evita la confusión sobre si estamos hablando de obtener un solo recurso o una colección. También evita agregar cosas adicionales como adjuntar todo a la URL base, por ejemplo, `/product/all`

Puede que a algunas personas no les guste esto, pero mi única sugerencia es mantenerlo uniforme en todo el proyecto.

Usar parametros

A veces necesitamos tener una API que debería contar más historias que solo por ID. Aquí deberíamos hacer uso de los parámetros de consulta para diseñar la API.

- `/products?name='ABC'` debería preferirse a `/getProductsByName`
- `/products?type='xyz'` debería preferirse a `/getProductsByType`

De esta manera puede evitar las URL largas con simplicidad en el diseño.

Usa los códigos HTTP apropiados

Tenemos muchos códigos HTTP . La mayoría de nosotros solo terminamos usando dos - 200 y 500! Esto ciertamente no es una buena práctica. A continuación se presentan algunos códigos HTTP de uso común.

- **200 OK** : este es el código HTTP más utilizado para mostrar que la operación realizada es exitosa.
- **201 CREADO** : se puede usar cuando usa el método POST para crear un nuevo recurso.
- **202 ACEPTADO** : se puede usar para confirmar la solicitud enviada al servidor.
- **400 BAD REQUEST** : se puede usar cuando falla la validación de entrada del lado del cliente.
- **401 NO AUTORIZADO / 403 PROHIBIDO** : se puede usar si el usuario o el sistema no están autorizados para realizar una determinada operación.

- **404 NO ENCONTRADO** : se puede usar si está buscando un recurso determinado y no está disponible en el sistema.
- **500 ERROR DE SERVIDOR INTERNO** : nunca debe lanzarse explícitamente, pero puede ocurrir si el sistema falla.
- **502 BAD GATEWAY** : se puede usar si el servidor recibió una respuesta no válida del servidor ascendente.

Versiones

La versión de las API es muy importante. Muchas compañías diferentes usan versiones de diferentes maneras. Algunos usan versiones como fechas, mientras que otros usan versiones como parámetros de consulta. En general me gusta mantenerlo prefijado al recurso. Por ejemplo:

```
/ v1 / products
/ v2 / products
```

También me gustaría evitar el uso `/v1.2/products` , ya que implica que la API cambiaría con frecuencia. Además, los puntos (.) Pueden no ser fácilmente visibles en las URL. Así que manténlo simple.

Siempre es una buena práctica mantener la compatibilidad con versiones anteriores para que, si cambia la versión de la API, los consumidores tengan el tiempo suficiente para pasar a la próxima versión.

Usar paginación

El uso de la paginación es una necesidad cuando se expone una API que puede devolver datos enormes, y si no se realiza el balanceo de carga adecuado, el consumidor podría terminar con la interrupción del servicio. Siempre debemos tener en cuenta que el diseño de la API debe ser una prueba completa y una prueba infalible.

El uso de `limit` y `offset` se recomienda aquí. Por ejemplo `/products?limit=25&offset=50` ,. También se recomienda mantener un límite predeterminado y un desplazamiento predeterminado.

Formatos soportados

También es importante elegir cómo responde su API. La mayoría de las aplicaciones de hoy en día deberían devolver respuestas JSON, a menos

que tenga una aplicación heredada que aún necesite obtener una respuesta XML.

Use los mensajes de error apropiados

Siempre es una buena práctica mantener un conjunto de mensajes de error que la aplicación envía y responder a eso con la identificación correcta. Por ejemplo, si utiliza las API de gráficos de Facebook, en caso de errores, devuelve un mensaje como este:

```
{
  "error": {
    "mensaje": "(# 803) Algunos de los alias que solicitó no
existen: productos",
    "tipo": "OAuthException",
    "código": 803,
    "fbtrace_id": "FOXX2AhLh80"
  }
}
```

También he visto algunos ejemplos en los que las personas devuelven una URL con un mensaje de error, que le brinda más información sobre el mensaje de error y cómo manejarlo.

Uso de las especificaciones de OpenAPI.

Para que todos los equipos de su empresa cumplan con ciertos principios, el uso de la especificación OpenAPI puede ser útil. OpenAPI le permite diseñar sus API primero y compartirlas con los consumidores de una manera más fácil.

Conclusión

Es bastante evidente que si desea comunicarse mejor, las API son el camino a seguir. Pero si están mal diseñados, podría aumentar la confusión. Así que ponga su mejor esfuerzo en diseñar bien, y el resto es solo la implementación.

. . .

Gracias por leer

Si descubrió algunas formas mejores de diseñar API, siéntase libre de compartirlas en la sección de comentarios. Todos los comentarios son bienvenidos!