Publicado el día **6 de octubre de 2017** .

Mejores prácticas para su API REST

Como desarrollador de Android, casi siempre necesito solicitar información a una API REST, ya sea para obtener datos del servidor o enviarlos. A lo largo de mi experiencia laboral, he tenido que integrarme con las API que hemos desarrollado y otros terceros con los que tuvimos que integrarnos, pero he visto una lista de errores que todos deberían evitar al desarrollar una API REST.

Evite devolver siempre el mismo código de estado HTTP

He visto algunas API REST que siempre devolvieron un código de estado 200. Como no se devolvieron errores en esta API, podría pensar que todo fue perfecto, ¿verdad? Pero no fue así, fue un desastre de hecho. ¿Cómo puede diferenciar las respuestas exitosas con las respuestas de error provenientes del cliente o del servidor? ¿Respecto al cuerpo? Esta lógica debería estar en el backend, ¡nunca en el cliente! El cliente siempre debe ser estúpido: cuanto más, mejor. No debe contener lógica sobre la API del servidor. Si se cambia el servidor, puede estar causando errores en el cliente si no se actualiza al mismo tiempo; hecho que es muy común con las aplicaciones móviles.

Mi recomendación es utilizar códigos de estado HTTP para las respuestas del servidor. Puede encontrar los siguientes códigos de estado según RFC7321

1xx Respuesta informativa

100 Continuar

101 Cambio de protocolos

102 Procesamiento

2xx Respuesta exitosa

200 OK

201 Creado

202 Aceptado

204 Sin contenido

etc.

Redirección 3xx

300 opciones múltiples

302 Encontrado

304 No modificado

305 Usar proxy

etc

4xx Error en el cliente

400 Petición Incorrecta

401 No autorizado

403 prohibido

404 No encontrado

410 desaparecido

etc.

5xx Error en el servidor

Error interno de servidor 500

501 No implementado

502 Puerta de enlace no válida

503 Servicio no Disponible

etc

No utilizar formatos de respuesta de error diferentes

En relación con lo que describí anteriormente, si mantiene el mismo formato de error, simplificará la integración de los clientes con su API. Por ejemplo, puede definir una estructura como sigue y tiene cualquier middleware en su servidor que detecte respuestas de error y envuelva el error en una respuesta como

```
Response status code: 403
Response body
{
    code: 5,
    message: "Error message"
}
```

Además, los servidores no deben devolver el seguimiento de la pila de errores, deben manejarse.

Algunos ingenieros comienzan a desarrollar API REST con un único método HTTP, incluso codificando algunos servicios web protegidos mediante autenticación. He encontrado 2 casos, cuando todas las solicitudes implementan el método GET enviando información de sesión como parámetro de consulta o cuando todas las solicitudes implementan el método POST con información de datos de sesión incluida en el cuerpo. No están bien diseñados, tenemos diferentes métodos para realizar diferentes acciones.

HTTP define una serie de métodos estandarizados . Permítanme resumir los más utilizados con la definición HTTP:

GET: transfiere una representación actual del recurso de destino.

POST: realiza un procesamiento específico de recursos en la carga útil de la solicitud.

PUT: Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la solicitud.

ELIMINAR: Elimina todas las representaciones actuales del recurso de destino.

Ok, te estarás preguntando, ¿qué significa? ¿Cómo puedo aplicar a mi API REST? No se preocupe, les daré un ejemplo.

Imagina que estamos desarrollando un recurso de "usuarios", necesitamos crear 5 servicios web básicos:

	Método HTTP	CAMINO	CUERPO	Respuesta exitosa
Obtener todos los usuarios	OBTENER	/ usuarios		Lista de representaciones de usuarios
Consigue un usuario	OBTENER	/ users / <userId>		Representación de usuarios
Agregar un usuario	ENVIAR	/ usuarios	Representación de usuarios	Representación de usuarios
Editar un usuario	PONER	/ users / <userId>	Representación de usuarios	Representación de usuarios
Eliminar un usuario	ELIMINAR	/ users / <userId>		

Como puede ver, es muy claro saber qué hace cada recurso.

Control de versiones de su API

Hoy en día, las empresas suelen querer haber publicado sus aplicaciones en Google Play o AppStore y continuamente lanzan nuevas versiones de su aplicación, pero ¿qué pasa con el servidor? También está aumentando en funcionalidades. El problema surge cuando los usuarios no actualizan cuando hay una nueva actualización, sucede con frecuencia. No puede permitirse tener un servidor por versión de la aplicación, entonces, ¿qué puedo hacer? La única solución es tener diferentes versiones en tu API.

Además, recuerde que cuando tenga diferentes versiones de su API, debe pensar en la compatibilidad con versiones anteriores. No rompa nada que esté funcionando actualmente después de una nueva versión.

Si solo tuvieras que soportar un sitio web, podrías olvidar este consejo, pero estamos en un mundo donde las aplicaciones móviles están en todas partes; por lo que tarde o temprano, su producto se publicará en los mercados de aplicaciones.

Incluir token de sesión en encabezados

implementando todas las solicitudes con el método POST para enviar el token de sesión como un parámetro en el cuerpo. No tiene sentido implementar el servicio web "Obtener todos los usuarios" con un método POST. Como ha aprendido anteriormente, este método debería ser GET.

Por lo tanto, admita el encabezado HTTP " *Authorization* " en su API y envíelo en todas las solicitudes con las credenciales de usuario para autenticarse con el servidor. Puede ser flexible y puede adaptarse fácilmente a su protocolo de autenticación.

Internacionaliza tu API

Es muy común devolver mensajes de error, exitosos o informativos en nuestra API. Como he comentado anteriormente, el cliente no debe contener ninguna lógica de servidor, ni siquiera para mensajes. A veces, necesitamos mostrar esos mensajes en el cliente. Luego, debemos enviar internacionalizados según el dispositivo del cliente. Pero, ¿cómo puede el cliente solicitar mensajes en un idioma?

Admite el encabezado " *Accept-Language* " en nuestra API y los clientes son responsables de enviar su idioma en cada solicitud utilizando este encabezado de solicitud. Por ejemplo: configurando " *Accept-Language*: es " debería recibir mensajes en español.

Realmente le recomiendo que use cualquier biblioteca i18n para admitir esto, aunque actualmente solo admite un idioma. En el futuro, deberá admitir más de uno y será una pesadilla reemplazar cadenas codificadas de forma rígida por tokens i18n.

Pagina tus resultados

Implemente la paginación en todos los servicios web que crea que devolverá demasiados datos. Reducirás el tiempo de respuesta y los datos enviados a los clientes. Si ese cliente es nuestro dispositivo móvil, piense que puede que no tenga una buena conexión a Internet, como consecuencia, como se devuelve menos tiempo e información, será mejor.

Responde lo que estás solicitando

Si está solicitando un recurso, devuelva su representación o una lista de ellos; Evite responder con algo diferente. Puede pensar que lo está haciendo bien, pero le mostraré un ejemplo común cuando la gente se equivoca.

Si llamamos al servicio web para buscar a todos los usuarios, algunas API devuelven una respuesta como la siguiente:

```
{
    "users": [
        {
            "name": "John",
            "surname": "Smith",
            "age": 25
        },
        {
            "name": "Peter",
            "surname": "Potter",
            "age": 47
        }
    ]
}
```

```
[
    {
        "name": "John",
        "surname": "Smith",
        "age": 25
    },
    {
        "name": "Peter",
        "surname": "Potter",
        "age": 47
    }
]
```

Devolver tipos de datos correctos

Intente devolver los tipos de datos correctos y aproveche los objetos nulos en caso de que no tenga esa información.

Por ejemplo, imagine que queremos devolver información del usuario. Los campos que tenemos de los usuarios son nombre, apellido y edad.

```
{
    "name": <string>,
    "surname": <string>,
    "age": <int>
}
```

Si la edad no fuera obligatoria, podría ser nula. En ese caso, evite usar age como cadena y devuelva una cadena vacía (“”), devuelva nulo cuando sea desconocido.

```
{
    "name": "John",
    "surname": "Smith",
    "age": null
}
```

La integridad del modelo podría verse afectada y evitará conversiones de tipos de datos inútiles.

Pon a prueba tu backend con pruebas automatizadas

Las pruebas deberían ser obligatorias en todos los proyectos pero aún más con la parte que tiene toda la lógica del producto. Le ayudarán a no estropear o incluir nuevos errores en las funciones que están funcionando actualmente. Entonces, la próxima vez que comience a desarrollar una nueva función en su backend, incluya algunas pruebas de integración y unidades, se lo agradecerá.

Utilice HTTPS sobre HTTP y certificados válidos según sus clientes

Hoy en día, si desea tener una API segura, primero debe ejecutarla con un esquema HTTPS, este hecho es conocido por la mayoría de las personas. Pero tenga cuidado al elegir el certificado que desea usar en su aplicación. Evite los certificados autofirmados o los algoritmos de cifrado antiguos. Si tienes alguna duda, lee los algoritmos de encriptación que soportan tus clientes (dispositivos móviles, navegadores, etc) y elige uno según tus necesidades.

¿Podemos hacer? ¡Por supuesto que puede! Uno de mis ejemplos favoritos es usar el encabezado de solicitud *If-Modified-Since* . ¿Por qué? Porque podemos almacenar en caché la fecha de la última respuesta que recibimos y solicitar cambios desde esa fecha. Esta es una muy buena práctica para dispositivos móviles debido a que los usuarios generalmente no tienen planes de datos ilimitados.

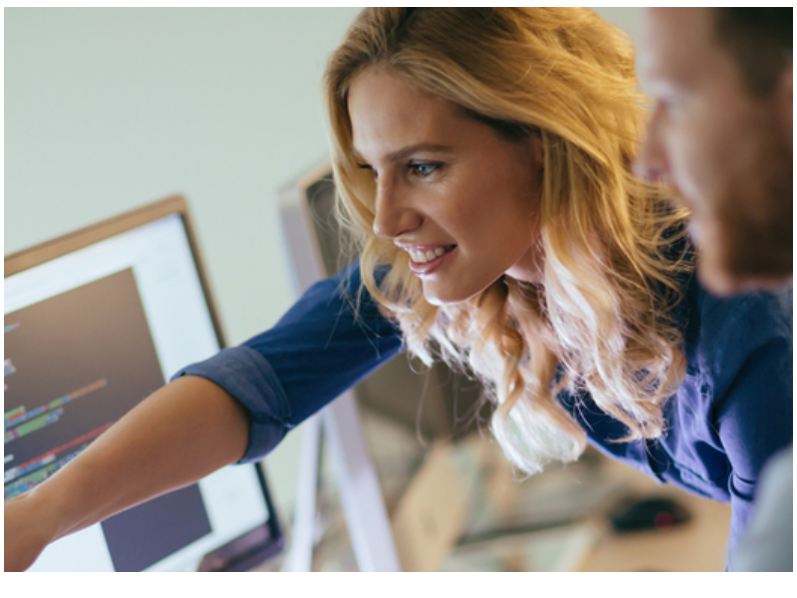
¿Más prácticas?

Por supuesto que podría olvidar más buenas prácticas y ¡vamos! ¡No seas tímido y coméntalos! Me encanta aprender algo nuevo y podemos ampliar este artículo o crear uno nuevo.

También te pueden interesar estos contenidos:

La Web3 no es el futuro

Automatización de procesos en empresas, ¿es rentable?



La gestión de proyectos en el ámbito comercial y marketing

Scandit: Integración de Matrix Scan