**URLs**

* Regla: *Nombrado de recursos*
* Identificador: *URL001*

**Descripción del estándar**

El nombrado de los recursos debe ser mediante sustantivos en forma plural, no mediante verbos.

**Buenas prácticas**

* GET /api/v1/magazines
* GET /api/v1/clientes/123456

**Malas prácticas**

* Uso de sustantivos en singular
* GET /magazine
* GET /magazine/1234
* Uso de verbos
* GET /magazines/1234/create

**Descripción detallada**

Una API REST se centra en las entidades del dominio del problema que expone, en lugar de un conjunto de funciones que manipulan esas entidades, la forma natural de referirse a estas entidades es mediante el uso de sustantivos.

Utilice la versión plural de un nombre de recurso (entidad), a menos que el recurso en cuestión sea un *singleton* dentro del sistema.

**Referencias**

* https://geemus.gitbooks.io/http-api-design/content/en/requests/resource-names.html
* https://apiguide.readthedocs.io/en/latest/build\_and\_publish/use\_RESTful\_urls.html

**URLs**

* Regla: *Plantillas de URI*
* Identificador: *URL002*

**Descripción del estándar**

Evite requerir URI de recursos más complejos que ***colección / elemento / colección****.*

**Buenas prácticas**

* https://dogtracker.com/persons/98765432
* https://dogtracker.com/persons/98765432/dogs

**Malas prácticas**

* http://adventure-works.com/customers/1/orders/99/products

**Descripción detallada**

Un API permite a un cliente realizar una serie de operaciones específicas en sus recursos. La técnica común para exponer estas funciones en una API es definir una plantilla de URI que el cliente puede usar para construir URLs a partir de información en los recursos (si no está familiarizado con las plantillas de URI, son URLs con variables que pueden ser reemplazadas con valores para formar una URL utilizable).

Cuando los diseñadores de API definen las plantillas de URI para su API, están diseñando un lenguaje de consulta personalizado para la API. Los diseñadores tienen una gran libertad en la forma en que definen este lenguaje de consulta, pero cuanto más ejercen esta libertad, mayor será la carga del cliente API para aprender lo que se puede hacer a partir de estas plantillas.

**Referencias**

* https://tools.ietf.org/html/rfc6570
* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
* https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf

**URLs**

* Regla: *Filtrar colecciones*
* Identificador: *URL003*

**Descripción del estándar**

Utilizar parámetros de consulta si desea clasificar o filtrar elementos.

**Buenas prácticas**

* https://dogtracker.com/dogs?color=red&state=running&location=park
* https://dogtracker.com/persons/5678/dogs?color=red

**Algunos otros ejemplos.**

* + /users?sort\_by=asc(email)
  + /users?sort\_by=desc(email)
  + /app/items?limit=30&marker=08ec231f6d9a43dda97d4b950c3393df
  + /photos?limit=5
  + /photos?limit=5&order=size
  + /comprobantes?fecha-desde=2017&orden=DESC&pagina=1
  + GET /users?sort\_by=asc(email)
  + GET /users?sort\_by=desc(email)
  + GET /users?sort\_by=+email
  + GET /users?sort\_by=-email
  + GET /users?sort\_by=email.asc
  + GET /users?sort\_by=email.desc
  + GET /users?sort\_by=email&order\_by=asc
  + GET /users?sort\_by=email&order\_by=desc
  + GET /items?limit=20&after\_id=20

**Malas prácticas**

* + /photos/order/size/limit/5
  + /photos/limit/5/order/size
  + /comprobantes/orden/desc/fecha-desde/2007/pagina/1

**Descripción detallada**

Mantenga su API intuitiva al simplificar las asociaciones entre los recursos. Este estilo de consulta URL puede expresar fácilmente las consultas que son difíciles de expresar con variables *de ruta o plantillas de URI*.

El uso de URL de consulta basadas en rutas y parámetros de cadena de consulta se puede combinar.

Como práctica recomendada, casi todos los desarrolladores recomiendan lo siguiente:

Si desea identificar un recurso, debe usar variable de ruta. Pero si desea clasificar o filtrar elementos, entonces debe usar el parámetro de consulta.

**Referencias**

* https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf

**URLs**

* Regla: *Uso de sub recursos*
* Identificador: *URL004*

**Descripción del estándar**

Utilice sub recursos para relaciones.

**Buenas prácticas**

* https://dogtracker.com/persons/1/dogs

Devuelve una lista de los perros que pertenecen a la persona #1

* https://dogtracker.com/persons/1/dogs/1

Devuelve detalles del perro #1 del propietario #1

**Consideraciones**

* https://dogtracker.com/search?type=Dog&owner={personId}​

Aunque no es considerada una mala práctica hay una serie de razones por las cuales el estilo propuesto como buena práctica es preferido.

**Descripción detallada**

El objetivo de esta regla es mostrar cómo puede diseñar URL de consulta que son regulares y predecibles para que un usuario que conoce el modelo de los datos expuesto por su API puede predecir principalmente el formato de las URL de consulta sin que usted se las explique.

* Muchos desarrolladores de aplicaciones que consumen la API encuentran el primer estilo más legible e intuitivo.
* Los desarrolladores de API que implementan la API generalmente encuentran que el primer estilo es más fácil de implementar.
* Este estilo de URL de consulta puede expresar fácilmente consultas transversales de gráficos que son difíciles de expresar en el estilo de parámetro de consulta.

**Referencias**

* https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf

**URLs**

* Regla: Tamaño máximo de URL en la solicitud HTTP GET
* Identificador: *URL005*

**Descripción del estándar**

Limitar la cantidad de información que viaja en la URI. Si bien el estándar HTTP no define un límite de tamaño de URL y los encabezados, la mayoría de los clientes y servidores HTTP tienen un límite práctico entre 2 kB y 8 kB.

**Buenas prácticas**

* https://maps.googleapis.com/maps/api/geocode/json?latlng=32.677649,-97.465539&key=1234567890-8OFbk-Q-0os1dfghaaLOPp86HBeTnSuo

**Malas prácticas**

* No debemos usar las solicitudes GET para que los datos se publiquen en el servidor, solo se deben usar, como su nombre indica, para obtener un recurso particular del servidor.
* Las URL muy largas no se recomiendan en absoluto.

**Descripción detallada**

Las solicitudes GET se realizan para solicitar un recurso particular del servidor. Cualquier información necesaria para obtener el recurso deseado se pasa como parámetros de cadena de consulta.

Para que una aplicación basada en web sea compatible con todos los navegadores, el método Get no debe usarse para pasar cadenas muy largas, que contengan más de 2000 caracteres. Si se enfrenta a una situación en la que necesita usar URL para contener una gran cantidad de caracteres, entonces debería diseñar nuevamente su API.

**Referencias**

* https://technomanor.wordpress.com/2012/04/03/maximum-url-size/
* https://blogs.dropbox.com/developers/2015/03/limitations-of-the-get-method-in-http/

**URLs**

* Regla: Envio de información sensible
* Identificador: *URL006*

**Descripción del estándar**

El envío de información sensible debe ser enviada en el cuerpo o en un encabezado personalizado de la solicitud.

**Buenas prácticas**

POST /oauth2/client\_credential/authorizationcode HTTP/1.1

Host: fgo084-eval-test.apigee.net

User-Agent: curl/7.61.1

Accept: \*/\*

Content-Length: 0

client\_id: P4OgdfQX3bVtJfZWqHjVJzKRmkq4GGAz

response\_type: code

redirect\_uri: test

**Malas prácticas**

POST /oauth2/client\_credential/authorizationcode?client\_id=P4OgdfQX3bVtJfZWqHjVJzKRmkq4GGAz&response\_type=code&redirect\_uri=test

HTTP/1.1

Host: fgo084-eval-test.apigee.net

User-Agent: curl/7.61.1

Accept: \*/\*

Content-Length: 0

**Descripción detallada**

Aunque pueden pasar los valores como parámetros de consulta como se muestra arriba, es una buena práctica pasarlos como una cadena codificada en un encabezado.

**Referencias**

* https://docs.apigee.com/api-platform/security/oauth/oauth-20-client-credentials-grant-type

**Verbos HTTP**

* Regla: *Método GET*
* Identificador: *VRB001*

**Descripción del estándar**

Recuperar una representación del recurso en el URI especificado.

**Buenas prácticas**

|  |  |
| --- | --- |
| **Recurso** | **GET** |
| /customers | Recuperar a todos los clientes |
| /customers/1 | Recuperar los detalles para el cliente 1 |
| /customers/1/orders | Recuperar todos los pedidos para el cliente 1 |

* **Ejemplo. Listar todas las revistas contenidas en el recurso /magazines**

*Request*

GET /magazines HTTP/1.1  
Host: example.gov.au  
Accept: application/json, text/javascript

*Response*

* 200 (Indica que la solicitud ha tenido éxito)

HTTP/1.1 200 OK  
Cache-Control: no-cache  
Content-Type: text/javascript  
{...}

* 404 (Indica que el recurso no existe

HTTP/1.1 404 Not Found  
Cache-Control: no-cache  
Content-Type: text/javascript

**Malas prácticas**

No use el método GET para cambios de estado.

**Descripción detallada**

El método **GET** hace referencia a la acción de **Leer** (**Read**) del CRUD. Las solicitudes GET (**GET requests**) se consideran **seguras**, lo que significa que no deben alterar el estado del servidor. Las solicitudes GET se consideran **idempotentes**, lo que significa que deben ser repetibles sin causar efectos secundarios. Las solicitudes GET no deben tener un cuerpo (body request).

**Referencias**

* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/

**Verbos HTTP**

* Regla: *Método POST*
* Identificador: *VRB002*

**Descripción del estándar**

Crear un nuevo recurso en el URI especificado.

**Buenas prácticas**

|  |  |  |
| --- | --- | --- |
| **Recurso** | **POST** | |
| /customers | | Crea un nuevo cliente |
| /customers/1/orders | | Crear un nuevo pedido para el cliente 1 |

* **Ejemplo. Crear nueva revista dentro del recurso / magazines.**

*Request*

POST /magazines HTTP/1.1  
Host: example.gov.au  
Accept: application/json, text/javascript  
{...}

*Response*

* Código de estado 201 (Indica que la solicitud ha sido cumplida)

HTTP/1.1 201 Created  
Vary: Accept  
Content-Type: text/javascript  
{ "id": "1234" }

* Código de estado 405 (Indica que el cliente intentó usar un método que el servidor no admite, la solicitud no tuvo efecto debido a que el recurso ya existe)

HTTP/1.1 405 Method Not Allowed  
Vary: Accept  
Content-Type: text/javascript  
{  
  "developerMessage" : "Unable to create a magazine with ID of 1234 because a magazine with that ID already exists",  
  "userMessage" : "Unable to create duplicate magazine 1234",  
  "errorCode" : "444444",  
  "moreInfo" : "http://api.example.gov/v1/documentation/errors/444444.html"  
}

**Malas prácticas**

|  |  |
| --- | --- |
| **Recurso** | **POST** |
| /customers/1 | Error |

**Descripción detallada**

El método **POST** hace referencia a la acción de **Crear** (**Create**) del CRUD. Las solicitudes POST (**POST requests**) se consideran **Inseguras** ya que alteran el estado del servidor. Se consideran **no idempotentes** ya que las solicitudes repetidas darán como resultado la creación de varios recursos. Las solicitudes POST deben tener un cuerpo (**body request**).

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

**Verbos HTTP**

* Regla: *Método DELETE*
* Identificador: *VRB003*

**Descripción del estándar**

Eliminar el recurso en el URI especificado.

**Buenas prácticas**

|  |  |  |
| --- | --- | --- |
| **Recurso** | **DELETE** | |
| /customers | | Eliminar a todos los clientes | |
| /customers/1 | | Eliminar cliente 1 | |
| /customers/1/orders | | Eliminar todos los pedidos para el cliente 1 | |

* **Ejemplo. Eliminar el recurso especificado /magazines/1234.**

*Request*

DELETE /magazines/1234 HTTP/1.1  
Host: example.gov.au  
Accept: application/json, text/javascript

*Response*

* Código de estado 200 (Indica que la solicitud ha tenido éxito)

HTTP/1.1 200 OK  
Vary: Accept  
Content-Type: text/javascript

* Código de estado 204 (Indica que la solicitud ha tenido éxito pero no hay contenido adicional para enviar en el cuerpo de respuesta)

HTTP/1.1 204 No Content  
Vary: Accept  
Content-Type: text/javascript

**Descripción detallada**

El método **DELETE** hace referencia a la acción de **Eliminar** (**Delete**) del CRUD. Las solicitudes DELETE (**DELETE requests**) se consideran Inseguras ya que alteran el estado del servidor. Sin embargo, se consideran idempotentes, ya que al repetir esta acción DELETE, no debe tener ningún efecto secundario (es decir si por casualidad vuelves a invocar al endpoint con el verbo DELETE, no va eliminar un recurso que ya fue eliminado). Una solicitud DELETE no debe tener un cuerpo (body request).

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

**Verbos HTTP**

* Regla: *Método PUT*
* Identificador: *VRB004*

**Descripción del estándar**

Crear o reemplazar el recurso en el URI especificado.

**Buenas prácticas**

|  |  |  |
| --- | --- | --- |
| **Recurso** | **PUT** | |
| /customers | | Actualización masiva de clientes |
| /customers/1 | | Actualice los detalles del cliente 1 si existe |
| /customers/1/orders | | Actualización masiva de pedidos para el cliente 1 |

* **Ejemplo. Reemplaza el recurso /magazines/1234 con la representación en la solicitud**

*Request*

PUT /magazines/1234 HTTP/1.1  
Host: example.gov.au  
Accept: application/json, text/javascript  
[ ... ]

*Response*

* Código de estado 200 (Indica que la solicitud ha tenido éxito)

HTTP/1.1 200 OK  
Vary: Accept  
Content-Type: text/javascript  
{ "id": "1234" }

* Código de estado 204 (Indica que la solicitud ha tenido éxito pero no hay contenido adicional para enviar en el cuerpo de respuesta)

HTTP/1.1 204 No Content  
Vary: Accept  
Content-Type: text/javascript

**Descripción detallada**

El método **PUT** hace referencia a la acción de **Actualizar** (**Update**) del CRUD. Se considera **Inseguros** ya que altera el estado del servidor. Por lo general se considera **Idempotent**, es decir si invocamos a la misma solicitud varias veces debe tener el mismo resultado. Esto es cierto en la situación en la que está proporcionando propiedades de recursos y valores que se reemplazarán en un mismo recurso. Sin embargo, si su servicio tiene la capacidad de modificar una propiedad, como incrementar una propiedad, estas solicitudes se convertirían en no idempotentes. El método PUT debe tener un cuerpo.

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

**Verbos HTTP**

* Regla: *Método PATCH*
* Identificador: *VRB005*

**Descripción del estándar**

Realizar una actualización parcial de un recurso.

**Buenas prácticas**

|  |  |  |
| --- | --- | --- |
| **Recurso** | **PATCH** | |
| /customers | | Actualización masiva de clientes |
| /customers/1 | | Actualice los detalles del cliente 1 si existe |
| /customers/1/orders | | Actualización masiva de pedidos para el cliente 1 |

* **Ejemplo. Actualiza de manera parcial el recurso /magazines/1234 con la representación en la solicitud**

*Request*

PUT /magazines/1234 HTTP/1.1  
Host: example.gov.au  
Accept: application/json, text/javascript  
[ ... ]

*Response*

* Código de estado 200 (Indica que la solicitud ha tenido éxito)

HTTP/1.1 200 OK  
Vary: Accept  
Content-Type: text/javascript  
{ "id": "1234" }

* Código de estado 204 (Indica que la solicitud ha tenido éxito pero no hay contenido adicional para enviar en el cuerpo de respuesta)

HTTP/1.1 204 No Content  
Vary: Accept  
Content-Type: text/javascript

**Descripción detallada**

El método **PATCH** hace referencia a la acción de **Actualizar** (**Update**) del CRUD. Se considera **Inseguros** ya que altera el estado del servidor. Por lo general se considera **Idempotent**, es decir si invocamos a la misma solicitud varias veces debe tener el mismo resultado. Esto es cierto en la situación en la que está proporcionando propiedades de recursos y valores que se reemplazarán en un mismo recurso. Sin embargo, si su servicio tiene la capacidad de modificar una propiedad, como incrementar una propiedad, estas solicitudes se convertirían en no idempotentes. El método PATCH debe tener un cuerpo.

Estas solicitudes se usan para realizar **actualizaciones parciales**. Sólo se establecerán las propiedades proporcionadas en el cuerpo de solicitud, las propiedades no listadas conservarán su valor existente. Algunas API sólo admiten PUT o PATCH para realizar cualquier actualización.

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

**Verbos HTTP**

* Regla: *Método HEAD*
* Identificador: *VRB006*

**Descripción del estándar**

Obtener encabezados de un recurso.

**Buenas prácticas**

* HEAD /employees HTTP/1.1

**Descripción detallada**

La solicitud**HEAD** **se utiliza para obtener sólo los headers** (encabezados) de un recurso. Estos headers pueden contener **metadatos** útiles como cuando el recurso debe haber caducado. Si el costo de recuperar un recurso es alto.

Por lo general, devuelve los mismos encabezados que GET. La especificación HTTP requiere que ni la solicitud ni la respuesta contengan un cuerpo.

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://blog.4psa.com/rest-best-practices-choosing-http-methods/

**Verbos HTTP**

* Regla: *Método OPTIONS*
* Identificador: *VRB007*

**Descripción del estándar**

Consultar los métodos HTTP disponibles para la manipulación de recursos.

**Buenas prácticas**

* OPTIONS /docs HTTP/1.1

**Descripción detallada**

La solicitud OPTIONS es útil para servicios expuestos a un navegador, por ejemplo, un navegador cliente puede enviar una solicitud OPTIONS y el servidor responde con headers que especifican que métodos HTTP puede utilizar el navegador y que otras funciones son soportadas por el servidor web.

Tenga en cuenta que la solicitud no contiene un cuerpo, pero es libre de devolver la representación de un recurso en la respuesta.**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://blog.4psa.com/rest-best-practices-choosing-http-methods/

**Códigos de estado HTTP**

* Regla: *Respuestas informativas*
* Identificador: *CDE001*

**Descripción del estándar**

Solicitar respuesta informativa del servidor.

**Buenas prácticas**

* 100 Continue

Informa a los destinatarios que el cliente está a punto de enviar un cuerpo de mensaje (presumiblemente grande) en esta solicitud y desea recibir una respuesta provisional de 100 (Continue).

Un cliente envía una solicitud con un encabezado Expect y espera a que el servidor responda antes de enviar el cuerpo del mensaje.

El servidor ahora comprueba los encabezados de solicitud y puede responder con una respuesta 100 (Continue) para indicar al cliente que siga adelante y envíe el cuerpo del mensaje, o enviará un estado 417 (Expectation Failed) si alguna de las expectativas no puede cumplirse.

También permite que el servidor de origen responda inmediatamente con un mensaje de error, como 401 (no autorizado) o 405 (método no permitido), antes de que el cliente comience a llenar datos innecesarios.

Request:

PUT /v1/products/31001?apikey=g8dStp1mMmPHSZBrKieBGuz8CSuzmlkA HTTP/1.1

Host: fgo084-eval-test.apigee.net

Accept: \*/\*

User-Agent: Rigor API Tester

Expect: 100-continue

Content-Length: 1234567890987

Response:

HTTP/1.1 100 Continue

Date: Fri, 28 Sep 2018 23:12:34 GMT

Content-Type: application/json; charset=utf-8

Content-Length: 36

Connection: keep-alive

Server: nginx

Access-Control-Allow-Origin: \*

Cache-Control: no-cache

Strict-Transport-Security: max-age=31556926; includeSubDomains; preload

**Descripción detallada**

**1xx: Respuestas informativas**

Petición recibida, continuando proceso. Esta respuesta significa que el servidor ha recibido los encabezados de la petición, y que el cliente debería proceder a enviar el cuerpo de la misma (en el caso de peticiones para las cuales el cuerpo necesita ser enviado; por ejemplo, una petición Hypertext Transfer Protocol). Si el cuerpo de la petición es largo, es ineficiente enviarlo a un servidor, cuando la petición ha sido ya rechazada, debido a encabezados inapropiados.

**Referencias**

* https://developer.mozilla.org/es/docs/Web/HTTP/Status/100
* https://tools.ietf.org/html/rfc7231#section-5.1.1
* https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP

**Códigos de estado HTTP**

* Regla: *Peticiones correctas*
* Identificador: *CDE002*

**Descripción del estándar**

Validar solicitudes recibidas correctamente, entendidas y aceptadas.

**Buenas prácticas**

* 201 Created

El código de estado 201 siempre debe venir con un encabezado de ubicación que proporcione la URL del recurso recién creado.

HTTP/1.1 201 Created

Location: https://dogtracker.com/dogs/1234567

* 200 OK

La respuesta a un GET exitoso no es solo un código de estado 200 y algunos datos, también incluye un paquete estándar de encabezados.

HTTP/1.1 200 OK

Content-Location: https://dogtracker.com/dogs/1234567

Content-Type: application/json

ETag: 1437080173827

Content-Length: nnnn

// body goes here //

**Descripción detallada**

**2xx: Peticiones correctas**

Permite al cliente validar la respuesta a su solicitud, tales como respuestas a peticiones correctas, peticiones que han sido completas, aceptadas, peticiones que se han completado con éxito pero no tiene contenidos, etc.

El significado de un éxito depende del método de solicitud HTTP:

El resultado exitoso de un PUT o un DELETE a menudo no es un 200 OK, sino un 204 Sin contenido o un 201 Creado cuando el recurso se carga por primera vez.

**Referencias**

* https://developer.mozilla.org/es/docs/Web/HTTP/Status/200
* https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP

**Códigos de estado HTTP**

* Regla: *Redirecciones*
* Identificador: *CDE003*

**Descripción del estándar**

Considerar acciones adicionales en respuesta a una solicitud.

**Buenas prácticas**

* [301 Moved Permanently](https://es.wikipedia.org/wiki/HTTP_301)

Lo recomendable es proporcionar información de ayuda y regresar un HTTP status 301 e incluir la correcta URL en la location header.

HTTP/1.1 301 Moved Permanently

Location: https://api.yourdomain.com/v2/customers

{

"deprecated": true,

"message": "use /v2/customers"

}

* 302 Found

Este código de respuesta proporcionará una URL en el campo del encabezado Location. Esta es una invitación al agente de usuario para realizar una segunda solicitud.

Es común hacer uso de este código cuando solicitas algún código de autorización como en el ejemplo siguiente:

POST /oauth2/client\_credential/authorizationcode HTTP/1.1

Host: email-eval-test.apigee.net

User-Agent: curl/7.61.1

Accept: \*/\*

Content-Length: 0

client\_id: TTY0gdfQX3bVtJfZWqHjVJzKRmkq4GGAz

response\_type: code

redirect\_uri: test

HTTP/1.1 302 Found

Date: Thu, 18 Oct 2018 15:10:28 GMT

Content-Length: 0

Connection: keep-alive

Location: test?code=jvwRaYyU

* 303 See Other

La respuesta a la solicitud se puede encontrar en otro URI utilizando el método GET. Cuando se recibe en respuesta a un POST (o PUT / DELETE), el cliente debe suponer que el servidor ha recibido los datos y debe emitir una nueva solicitud GET al URI dado.

Es muy común hacer uso de este código de respuesta cuando haces uso de paginaciones como en el ejemplo siguiente:

Request:

GET /dogs HTTP/1.1

Host: dogtracker.com

Accept: application/json

Response:

HTTP/1.1 303 See Other

Location: https://dogtracker.com/dogs?limit=25,offset=0

* 304 Not Modified

El cliente envió un encabezado if-modified-since con la solicitud para indicar al servidor que envíe la respuesta solo si se actualizó después de la fecha en este encabezado.

Request:

GET /v1/customers/ABC123 HTTP/1.1

Host: api.yourdomain.com

If-Modified-Since: Wed, 21 Dec 2016 08:12:31 MDT

Response:

HTTP/1.1 304 Not Modified

Last-Modified: Wed, 21 Dec 2016 08:12:31 MDT

**Descripción detallada**

**3xx: Redirecciones**

Esta clase de código de estado indica que una acción subsecuente necesita efectuarse por el agente de usuario para completar la petición. La acción requerida puede ser llevada a cabo por el agente de usuario sin interacción con el usuario si y solo si el método utilizado en la segunda petición es GET o HEAD. El agente de usuario no debe redirigir automáticamente una petición más de 5 veces, dado que tal funcionamiento indica usualmente un Bucle infinito.

**Referencias**

* https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP

**Códigos de estado HTTP**

* Regla: *Errores del cliente*
* Identificador: *CDE004*

**Descripción del estándar**

La solicitud contiene sintaxis incorrecta o no puede procesarse.

**Buenas prácticas**

* 400 Bad Request

Indica que el servidor no puede o no procesará la solicitud debido a algo que se percibe como un error de cliente (por ejemplo, sintaxis de solicitud mal formada, solicitud no válida) enmarcado de mensajes, o enrutamiento de solicitudes engañosas)

POST /users

{"name":"John Doe"}

400 Bad Request

{"error": "invalid\_user", "error\_description": "A user must have an email adress"}

* 401 Unauthorized

Indica que la solicitud no se ha aplicado porque carece de credenciales de autenticación válidas para el recurso de destino.

Este estado se envía con un encabezado WWW-Authenticate que contiene información sobre cómo autorizar correctamente.

Este estado es similar al 403, pero en este caso, la autenticación es posible.

Request:

PUT /v1/employees/41f1f77b-8922-11e8-86ee-021e63aadcc4 HTTP/1.1

Host: cosafinity-prod.apigee.net

Accept: \*/\*

User-Agent: Rigor API Tester

Content-Length: 524

Response:

HTTP/1.1 401 Unauthorized

Date: Fri, 14 Sep 2018 17:47:46 GMT

Content-Type: application/json

Content-Length: 210

Connection: keep-alive

Access-Control-Allow-Origin: \*

Server: Apache

Set-Cookie: rememberMe=deleteMe; Path=/; Max-Age=0; Expires=Thu, 13-Sep-2018 17:47:46 GMT

* [403 Forbidden](https://es.wikipedia.org/wiki/HTTP_403)

La solicitud fue legal, pero el servidor rehúsa responderla dado que el cliente no tiene los privilegios para hacerla. En contraste a una respuesta 401 No autorizado, la autenticación no haría la diferencia.

GET /users/1234

403 Forbidden (Existe el usuario pero no es accesible)

*Nota: Considere enviar como código de respuesta 404 para todas las solicitudes GET esto evitara a un actor malicioso conocer la existencia de un usuario.*

* [404 Not Found](https://es.wikipedia.org/wiki/HTTP_404)

Recurso no encontrado. Se utiliza cuando el servidor web no encuentra la página o recurso solicitado

GET /users/1234

404 Not Found

* 405 Method Not Allowed

Indica que el cliente intentó usar un método que el servidor no admite, el recurso dado debe emparejarse con un encabezado Allow que indica qué métodos son compatibles con el recurso.

HTTP/1.1 405 Method Not Allowed

Allow: GET, DELETE, PATCH

**Descripción detallada**

**4xx Errores del cliente**

La intención de la clase de códigos de respuesta 4xx es para casos en los cuales el cliente parece haber errado la petición. Excepto cuando se responde a una petición HEAD, el servidor **debe** incluir una entidad que contenga una explicación a la situación de error, y si es una condición temporal o permanente. Estos códigos de estado son aplicables a cualquier método de solicitud (como GET o POST). Los agentes de usuario *deben* desplegar cualquier entidad al usuario. Estos son típicamente los códigos de respuesta de error más comúnmente encontrados.

**Referencias**

* https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP
* https://tools.ietf.org/html/rfc7231#section-6.5.1

**Códigos de estado HTTP**

* Regla: *Errores de servidor*
* Identificador: *CDE005*

**Descripción del estándar**

El servidor falló al completar una solicitud aparentemente válida..

**Buenas prácticas**

* 500 Internal Server Error

GET /users

500 Internal server error

Content-Type: application/json

{"error":”server\_error", "error\_description":"Oops! Something went wrong..."}

**Descripción detallada**

**5xx Errores de servidor**

Indica que el servidor encontró una condición inesperada que le impidió cumplir con la solicitud.

Esta respuesta de error es una respuesta genérica de "catch-all". A veces, los administradores del servidor registran respuestas de error como el código de estado 500 con más detalles sobre la solicitud para evitar que el error vuelva a ocurrir en el futuro.

El cliente no puede realmente hacer nada al respecto.

**Referencias**

* https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos\_de\_estado\_HTTP
* https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/500

**Paginación**

* Regla: *Paginación*
* Identificador: *PAG001*

**Descripción del estándar**

Limitar la cantidad de datos devueltos por una sola solicitud.

**Buenas prácticas**

**Request:**

GET /dogs HTTP/1.1

Host: dogtracker.com

Accept: application/json

**Response:**

HTTP/1.1 303 See Other

Location: https://dogtracker.com/dogs?limit=25,offset=0

**Request:**

GET /dogs?limit=25,offset=0 HTTP/1.1

Host: dogtracker.com

Accept: application/json

**Response:**

HTTP/1.1 200 OK

Content-Type: application/json

Content-Location: https://dogtracker.com/dogs?limit=25,offset=0

Content-Length: 23456

{

"self": "https://dogtracker.com/dogs?limit=25,offset=0",

"kind": "Page",

"pageOf": "https://dogtracker.com/dogs",

"next": "https://dogtracker.com/dogs?limit=25,offset=25",

"contents": [

{"self": "https://dogtracker.com/dogs/12344",

"kind": "Dog",

"name": "Fido",

"furColor": "white"

},

{

"self": "https://dogtracker.com/dogs/12345",

"kind": "Dog",

"name": "Rover",

"furColor": "brown"

},

… (23 more)

]

}

**Malas prácticas**

Si una API no es compatible con la paginación desde el principio, su soporte posterior es problemático porque al agregar la paginación se rompe el comportamiento de la API.

**Descripción detallada**

Las solicitudes GET sobre los recursos de recopilación pueden devolver una gran cantidad de elementos. Debe diseñar una API web para limitar la cantidad de datos devueltos por una sola solicitud. Considere la posibilidad de admitir cadenas de consulta que especifiquen la cantidad máxima de elementos para recuperar y un desplazamiento inicial en la colección.

**Referencias**

* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
* https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf

**Selección de campos**

* Regla: *Selección de campos*
* Identificador: FLD*001*

**Descripción del estándar**

Proporcione al consumidor de API la posibilidad de elegir campos devueltos.

**Buenas prácticas**

**Request:**

GET /cars?fields=manufacturer,model,id,color HTTP/1.1

Host: dogtracker.com

Accept: application/json

**Response:**

HTTP/1.1 200 OK

Content-Type: application/json

{

…

}

**Malas prácticas**

La mayoría de los clientes móviles muestra solo pocos atributos en una lista, por lo que no implementar la posbilidad de elegir los campos requeridos incrementaría el trafico de red.

**Descripción detallada**

Es fácil de leer; un desarrollador puede seleccionar solo la información que necesita una aplicación en un momento dado; reduce los problemas de ancho de banda, lo que es importante para las aplicaciones móviles.

**Referencias**

* https://apigee.com/about/blog/technology/restful-api-design-can-your-api-give-developers-just-information-they-need

**Encabezados**

* Regla: *Encabezados personalizados*
* Identificador: *ENC001*

**Descripción del estándar**

Crear encabezados personalizados HTTP

**Buenas prácticas**

* Se recomienda simplemente nombrar encabezados personalizados con algo de relevancia sin el prefijo X.

Proxy-Connection: keep-alive

Status: 200 OK

**Malas prácticas**

* No debe prefijar sus nombres de parámetros con "X-" o construcciones similares.

**X-API-Version: 1**

**X-Powered-By: PHP/5.2.17**

X-Request-ID: f058ebd6-02f7-4d3f-942e-904344e8cde5

* Evitar los encabezados que proliferan.

OpenStack-Nova-API-Version: 2.1

**Descripción detallada**

Los encabezados HTTP personalizados generalmente están destinados a proporcionar información específica entre el cliente y el servidor o con fines de resolución de problemas.

Siempre que sea posible, debe usar un nombre de encabezado **más genéricos** y ubicar los detalles en el valor.

**Referencias**

* http://www.solocodigoweb.com/blog/2017/06/27/introduccion-al-diseno-de-una-api-http-parte-1-de-4-http-requests/
* https://tools.ietf.org/html/rfc2616.html
* https://tools.ietf.org/html/rfc6648.html
* https://specs.openstack.org/openstack/api-wg/guidelines/headers.html
* https://www.keycdn.com/support/custom-http-headers/
* https://en.wikipedia.org/wiki/List\_of\_HTTP\_header\_fields#access-control-response-headers

**Versionamiento**

* Regla: *Nomenclatura de versiones*
* Identificador: *VER001*

**Descripción del estándar**

Establecer un identificador de versión en un segmento de ruta de URL.

**Buenas prácticas sugeridas**

* Aunque el control de versiones de las API continúa siendo un tema dividido ya que la adopción de una a otra varía en opiniones. El patrón o la práctica más común es probablemente la que tiene un identificador de versión en un segmento de ruta de URL, esto hace más claro para los desarrolladores conocer cuál es la última versión.

GET /<domain>/v1/<resource>/{id}

* https://cosafinity-prod.apigee.net/v1/employees
* https://cosafinity-prod.apigee.net/v1/employees/41f1f77b-8922-11e8-86ee-021e63aadcc4
* La recomendación actual es apoyar el versionamiento a través de números de versión directamente en los URI de recursos. Eso hace que la versión sea visible y una API versionada sea más fácil de comprender y utilizar correctamente.

Los números de versión en URI deben ser altos en la jerarquía de nodos, preferiblemente como el primer nodo, para ejemplo:

* bancoazteca.com.mx/v1/usuarios
* bancoazteca.com.mx/api/v2/usuarios

**Enfoques de versionamiento no sugerido**

* Nunca libere una API sin una versión y haga que la versión sea obligatoria.
* Marca de tiempo en la URL. ​

​Incluir la fecha no es una garantía de que eventualmente esta versión sea depreciada.

/2010-04-01/Accounts/

* Versiones de cadenas de consulta

**​**Puede especificar la versión del recurso utilizando un parámetro dentro de la cadena de consulta. Pero depende del código que maneja la solicitud para analizar la cadena de consulta y devolver la respuesta HTTP adecuada.

http://adventure-works.com/customers/3?version=2

* Versiones de encabezado

**​**Establecer el número de versión dentro de un header personalizado, esto es conveniente pero no es fácil de explorar con herramientas comunes.

GET http://adventure-works.com/customers/3 HTTP/1.1  
Custom-Header: api-version=1

* Versiones media type

**​**Es posible definir tipos de medios personalizados que incluyen información que permite a la aplicación del cliente indicar qué versión de un recurso está esperando, pero si el encabezado no especifica ningún tipo de medio conocido, el servidor web podría generar un mensaje de respuesta HTTP 406 (No aceptable) o devolver un mensaje con un tipo de medio predeterminado.

GET http://adventure-works.com/customers/3 HTTP/1.1  
Accept: application/vnd.adventure-works.v1+json

* La estrategia de "no versionar" para versionar

https://api.acmecorp.com/orders/123456

https://api2.acmecorp.com/orders/123456

**Descripción detallada**

El control de versiones permite que una API web indique las características y recursos que expone.

Cuando selecciona una estrategia de control de versiones, también debe considerar las implicaciones en el rendimiento, especialmente el almacenamiento en caché en el servidor web. Algunos navegadores web más antiguos y proxies web no almacenan en caché las respuestas para las solicitudes que incluyen una cadena de consulta en el URI.

* Consideraciones de versionamiento
  + Major: Cambio drástico, esta cambio no es compatible con versiones inferiores.
  + Minor: Se añaden características nuevas al desarrollo y/o modifica funcionalidad existente, este cambio sigue siendo compatible con el código existente.
  + Patch: Solución de bugs siendo cambios retro compatibles.

**Referencias**

* https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
* https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf

**Seguridad**

* Regla: *Selección de protocolo*
* Identificador: *SEG001*

**Descripción del estándar**

Elegir el protocolo de seguridad correcto.

**Buenas prácticas**

* Utilizar el protocolo de autorización OAuth 1.0a o OAuth2 (preferiblemente MAC).

**Malas prácticas**

* Deben evitarse los protocolos de autenticación de la API personalizada, a menos que realmente sepa realmente lo que está haciendo y comprenda completamente todas las complejidades de las firmas digitales criptográficas.
* Evitar en lo posible Basic Authentication

**Descripción detallada**

Hoy en día, casi todas las principales API web utilizan **OAuth2**. Algunos ejemplos son PayPal, Twitter, Google, Facebook, GitHub y muchos más.

Los protocolos de autenticación estándar de la industria ayudan a reducir el esfuerzo de asegurar su API. Se pueden usar protocolos de seguridad personalizados, pero solo bajo circunstancias muy específicas.

* Listado de principales protocolos de seguridad
  + Basic API Authentication w/ TLS
  + OAuth1.0a
  + OAuth2
  + OAuth2 MAC
  + Protocolos personalizados

**Referencias**

* https://stormpath.com/blog/secure-your-rest-api-right-way
* https://docs-apis.apigee.io/files/Web-design-the-missing-link-ebook-2016-11.pdf
* https://es.slideshare.net/stormpath/secure-your-rest-api-the-right-way

**Seguridad**

* Regla: *API Key*
* Identificador: *SEG002*

**Descripción del estándar**

Utilizar API Key

**Buenas prácticas**

* Utilizar API Keys para evitar el uso malicioso o el abuso de la API.

GET https://developer.nrel.gov/api/alt-fuel-stations/v1.json?limit=1&api\_key=YOUR\_KEY\_HERE

**Descripción detallada**

La API Key a menudo actúa como un identificador único y un token secreto para la autenticación, y generalmente tendrá un conjunto de derechos de acceso en la API asociada a él.

Las API Keys proporcionan

* Identificación del proyecto: identifique la aplicación o el proyecto que realiza una llamada a esta API.
* Autorización de proyecto: compruebe si se le ha otorgado acceso a la aplicación de llamada a la API.

**Referencias**

* https://en.wikipedia.org/wiki/Application\_programming\_interface\_key
* https://cloud.google.com/endpoints/docs/openapi/when-why-api-key