

Proposed RESTful API Standards

The purpose of this document is to provide an effective and consistent development/usage experience when working with API endpoints.

Defintions:

Idempotent Methods: A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent.

Safe Methods: Request methods are considered "safe" if their defined semantics are essentially read-only; i.e., the client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource. Likewise, reasonable use of a safe method is not expected to cause any harm, loss of property, or unusual burden on the origin server.

| Method | Safe | Idempotent | Reference | Cachable |
|---------|------|------------|-------------------------------|----------------|
| GET | yes | yes | Section 4.3.1 | By Default |
| PUT | no | yes | Section 4.3.4 | Never |
| POST | no | no | Section 4.3.3 | Not by Default |
| DELETE | no | yes | Section 4.3.5 | Never |
| PATCH | no | yes | N/A | Not by Default |
| CONNECT | no | no | Section 4.3.6 | Never |
| HEAD | yes | yes | Section 4.3.2 | By Default |
| OPTIONS | yes | yes | Section 4.3.7 | Never |
| TRACE | yes | yes | Section 4.3.8 | Never |

This table references [IETF Tools](#) for RFCs

TLDR;

TLDR; - Do's:

- Do use HTTP **GET** method to request a representation of some resource. GETs should only be used to retrieve data
- Do use HTTP **POST** method to submit an entity to the specified resource. Often will change the state on the server
- Do use HTTP **PUT** to replace all current representations of the target resource
- Do use HTTP **DELETE** to delete the specified resource (or inactivate)
- Do use HTTP **PATCH** to apply a partial modification to some resource
- Do use **kebab-case** in the URI path of an endpoint
- Do use nouns for resources in the URI path
- Do order resources in a URI path hierarchically
- Do use a route prefix in controllers
- Do denote resource collections by being plural
- Do denote single resource records by being singular
 - This likely won't occur a lot of the time
- Do organize controller files/folders relative to endpoint URI paths
 - URI Template: [**api-metadata/**]**<file-name-prefix>/.../<folder-name>**
 - For example if we had the endpoint URI: **api/v3/users/{userId}/notaries**:
 - The containing folder for the controller is **Notary**
 - The containing file for the controller/endpoint is **UserController**
 - If, for example, we have a endpoint URI path that contains only one resource (e.g. **api/v1/users**)
 - The containing folder for the controller is **User**
 - The containing file for the controller/endpoint is **UserController**
- Do use query parameters to filter collections
- Do [only if needed] use a verb at the end of the URI path if the HTTP Methods are not adequate.
Example:
 - **api/v5/users/{userId}/cart/checkout**
 - You would likely provide the "cart" to be checked out
 - It might return whether or not the cart was successfully checked out
 - Like mentioned in the **Do Not's**: Don't use CRUD function names here
 - Should be thought of as modeling a procedural concept. Should be executed like function, having an input and output

TLDR; - Do Not's:

- Do not use idempotent (e.g. GET, PUT, DELETE) methods where the effect on the server is inconsistent with other such requests(non-idempotent)
 - This is very important for GETs, as they are both safe and idempotent
- Do not use PascalCase or snake_case in URI paths
- Do not use UPPER CASE characters in URI paths
- Do not use trailing slashes (/) in URI paths
- Do not use CRUD function names in URI paths. e.g.:
 - create
 - read
 - get
 - update
 - delete
 - remove

Do's

Do use the appropriate HTTP methods for the action being performed

GET

GET should only be used to request a representation of some resource. The results on this request should be filterable with the use of query parameters. The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching described in [section 13](#)

- **Idempotent:** Yes
- **Safe:** Yes
- **Response:** 200 (OK) with the queried collection/record.

PUT

PUT should only be used to replace all current representations of the target resource. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.

- **Idempotent:** Yes
- **Safe:** No
- **Response:** If a new resource is created, the origin server **MUST** inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes **SHOULD** be sent to indicate successful completion of the request.

POST

POST should only be used to submit an entity to the specified resource. Will change the state on the server. Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields.

- **Idempotent:** No
- **Safe:** No
- **Response:** The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result. If a resource has been created on the origin server, the response **SHOULD** be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header (see [section 14.30](#)).

DELETE

DELETE should only be used to delete the specified resource (or inactivate). The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server **SHOULD NOT** indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

- **Idempotent:** Yes
- **Safe:** No
- **Response:** A successful response **SHOULD** be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.

PATCH

PATCH should only be used to apply a partial modification to some resource. Should be treated as a partial PUT.

- **Idempotent:** Yes
- **Safe:** No
- **Response:** See PUT

For the definitions of idempotent and safe, see the Definitions section of this document

If an error/exception occurs, be sure to return a valuable response (HTTP status code/message)

Many of these specifications were derived from RFCs for [w3's Hypertext Transfer Protocol](#)

For a list of HTTP status codes, check out the [Mozilla Developer docs](#)

Do use **kebab-case** in the URI path of an endpoint

To make URIs easy for people to scan and interpret, use the hyphen (-) character to improve the readability of names in long path segments.

Do use nouns for resources in the URI path

RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb) because nouns have properties which verbs do not have – similar to resources have attributes. Some examples of a resource are:

- Users of the system
- User accounts
- A comment in a forum
- etc.

Do order resources in a URI path hierarchically

This improves the readability, and makes it easier to think about. Some examples might be:

```
api/v1/users/{id} → a user  
api/v1/users/{id}/accounts → accounts for some user  
api/v1/forum-threads/{id} → a forum thread  
api/v1/forum-threads/{id}/comments → comments from some forum thread
```

Do use a route prefix in controllers

Provided in the example below is how to route prefix. This provides a more direct context as to what each controller method does.

```
[RoutePrefix("api/web-user")] // [Route] in .NET Core  
public class WebUserController : ApiController  
{  
    [HttpGet]  
    [Route("{id}/accounts")]  
    public IHttpActionResult GetAccount(int id)  
    {  
        // Do stuff...  
    }  
  
    // ...  
}
```

Do denote resource collections by being plural

Example:

```
api/v3/users → all users  
api/v3/users/{id} → a specific user from the collection of users  
api/v3/user/{id}/accounts → all accounts for the specified user
```

Do denote single resource records by being singular

Example: `api/v1/users/{id}/cart`

This technique should be used sparingly. Very rarely will a resource not be a part of a collection.

References:

- <https://tools.ietf.org/html/rfc7231> for RFCs
- <https://restfulapi.net/resource-naming/>
- <https://www.w3.org/Protocols/rfc2616/rfc2616.html>
- <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

Naming convention presented by [Restful API .net](#):

For more clarity, let's divide the resource archetypes into four categories (document, collection, store and controller) and then you should always target to put a resource into one archetype and then use its naming convention consistently. For uniformity's sake, resist the temptation to design resources that are hybrids of more than one archetype.

document)

A document resource is a singular concept that is akin to an object instance or database record. In REST, you can view it as a single resource inside resource collection. A document's state representation typically includes both fields with values and links to other related resources.

Use "singular" name to denote document resource archetype.

```
http://api.example.com/device-management/managed-devices/{device-id}
http://api.example.com/user-management/users/{id}
http://api.example.com/user-management/users/admin
```

collection)

A collection resource is a server-managed directory of resources. Clients may propose new resources to be added to a collection. However, it is up to the collection to choose to create a new resource, or not. A collection resource chooses what it wants to contain and also decides the URIs of each contained resource.

Use "plural" name to denote collection resource archetype.

```
http://api.example.com/device-management/managed-devices
http://api.example.com/user-management/users
http://api.example.com/user-management/users/{id}/accounts
```

store)

A store is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them. A store never generates new URIs. Instead, each stored resource has a URI that was chosen by a client when it was initially put into the store.

Use "plural" name to denote store resource archetype.

```
http://api.example.com/cart-management/users/{id}/carts
http://api.example.com/song-management/users/{id}/playlists
```

controller)

A controller resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs.

Use “verb” to denote controller archetype.

<http://api.example.com/cart-management/users/{id}/cart/checkout> <http://api.example.com/song-management/users/{id}/playlist/play>