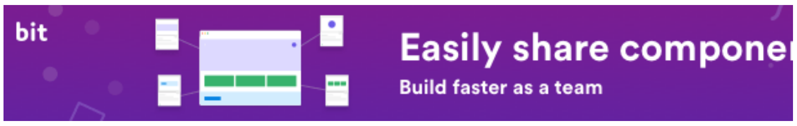
Fork me on GitHub



Using HTTP Methods for RESTful Services

Quick-Tips | Resource Naming

The HTTP verbs comprise a major portion of our “uniform interface” constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others.

Below is a table summarizing recommended return values of the primary HTTP methods in combination with the resource URIs:

HTTP Verb	**CRUD**	**Entire Collection (e.g. /customers)**	**Specific Item (e.g. /customers/{id})**
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Below is a more-detailed discussion of the main HTTP methods. Click on a tab for more information about the desired HTTP method.

POST GET PUT PATCH DELETE

PUT is most-often utilized for **update** capabilities, PUT-ing to a known resource URI with the request body containing the newly-updated representation of the original resource.

However, PUT can also be used to create a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. Again, the request body contains a resource representation. Many feel this is convoluted and confusing. Consequently, this method of creation should be used sparingly, if at all.

Alternatively, use POST to create new resources and provide the client-defined ID in the body representation—presumably to a URI that doesn't include the ID of the resource (see POST below).

On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. A body in the response is optional—providing one consumes more bandwidth. It is not necessary to return a link via a Location header in the creation case since the client already set the resource ID.

PUT is not a safe operation, in that it modifies (or creates) state on the server, but it is idempotent. In other words, if you create or update a resource using PUT and then make that same call again, the resource is still there and still has the same state as it did with the first call.

If, for instance, calling PUT on a resource increments a counter within the resource, the call is no longer idempotent. Sometimes that happens and it may be enough to document that the call is not idempotent. However, it's recommended to keep PUT requests idempotent. It is strongly recommended to use POST for non-idempotent requests.

Examples:

- *PUT http://www.example.com/customers/12345*
- *PUT http://www.example.com/customers/12345/orders/98765*
- *PUT http://www.example.com/buckets/secret_stuff*

REST API Tutorial