HTTP Methods

REST APIs enable you to develop any kind of web application having all possible CRUD (create, retrieve, update, delete) operations. REST guidelines suggest using a specific HTTP method on a particular type of call made to the server (though technically it is possible to violate this guideline, yet it is highly discouraged).

Use below-given information to find a suitable HTTP method for the action performed by API.

Table of Contents

Use GET requests **to retrieve resource representation/information only** – and not to modify it in any way. As GET requests do not change the state of the resource, these are said to be **safe methods**. Additionally, GET APIs should be **idempotent**, which means that

making multiple identical requests must produce the same result every time until another API (POST or PUT) has changed the state of the resource on the server.

If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

For any given HTTP GET API, if the resource is found on the server, then it must return HTTP response code `200 (OK)` – along with the response body, which is usually either XML or JSON content (due to their platform-independent nature).

In case resource is NOT found on server then it must return HTTP response code `404 (NOT FOUND)`. Similarly, if it is determined that GET request itself is not correctly formed then server will return HTTP response code `400 (BAD REQUEST)`.

HTTP POST

Use POST APIs **to create new subordinate resources**, e.g., a file is subordinate to a directory containing it or a row is subordinate to a database table. When talking strictly in terms of REST, POST methods are used to create a new resource into the collection of resources.

Ideally, if a resource has been created on the origin server, the response SHOULD be HTTP response code `201 (Created)` and contain an entity which describes the status of the request and refers to the new resource, and a Location header.

Many times, the action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either HTTP response code `200 (OK)` or `204 (No Content)` is the appropriate response status.

Responses to this method are **not cacheable**, unless the response includes appropriate Cache-Control or Expires header fields.

HTTP PUT

Use PUT APIs primarily **to update existing resource** (if the resource does not exist, then API may decide to create a new resource or not). If a new resource has been created by the PUT API, the origin server MUST inform the user agent via the HTTP response code `201 (Created)` response and 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.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

The difference between the POST and PUT APIs can be observed in request URIs. POST requests are made on resource collections, whereas PUT requests are made on a single

resource.

HTTP DELETE

As the name applies, DELETE APIs are used **to delete resources** (identified by the Request-URI).

A successful response of DELETE requests SHOULD be HTTP response `code 200 (OK)` if the response includes an entity describing the status, `202 (Accepted)` if the action has been queued, or `204 (No Content)` if the action has been performed but the response does not include an entity.

DELETE operations are **idempotent**. If you DELETE a resource, it's removed from the collection of resources. Repeatedly calling DELETE API on that resource will not change the outcome – however, calling DELETE on a resource a second time will return a 404 (NOT FOUND) since it was already removed. Some may argue that it makes the DELETE method

non-idempotent. It's a matter of discussion and personal opinion.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are **not cacheable**.

HTTP PATCH

HTTP PATCH requests are **to make partial update on a resource**. If you see PUT requests also modify a resource entity, so to make more clear – PATCH method is the correct choice for partially updating an existing resource, and PUT should only be used if you're replacing a resource in its entirety.