HTTP methods, status codes and response types

Introduction

You already know that HTTP methods and status codes play an essential role in REST APIs. It is important to keep to the conventions for HTTP methods and status codes for two reasons. Firstly, if you follow conventions from the developing phase, it will be much easier to avoid bugs in your code and deploy the final project on the production server. Secondly, it makes it easier for other developers to use your APIs. This reading highlights the most important HTTP methods, status codes and API response types that you will use when working on API projects.

HTTP methods

In the world of REST APIs, one endpoint can perform multiple tasks. It can deliver a resource, create a new resource, update or delete it. The endpoint remains the same while the action varies. When a client invokes an API, how does the API developer know which of the multiple actions should be performed? This is where HTTP methods come in.

HTTP methods or request types tell the API endpoint what it should do with the resources. It defines the action. The API developer makes decisions and manipulates resources appropriately based on the HTTP methods in an HTTP request. Here is a list of the most used HTTP methods and which action you should initiate for those calls.

HTTP method	Action
`GET`	Returns the requested resource. If not found, returns a `404 Not Found` status code.
`POST`	Creates a record. The `POST` request always comes with an HTTP request body containing JSON or Form URL encoded data, which is also called a payload. If the data is valid, the API endpoint will create a new resource based on these data. Although you can create multiple resources with a single `POST` call, it is not considered a best practice to do so.
`PUT`	Instructs the API to replace a resource. Like a `POST` request, the `PUT` request also comes with data. A `PUT` request usually supplies all data for a particular resource so that the API developer can fully replace that resource with the provided data. A `PUT` request deals with a single resource.

HTTP method	Action
`PATCH`	Tells the API to update a part of the resource. Note the difference between a `PUT` and a `PATCH` call. A `PUT` call replaces the complete resource, while the `PATCH` call only updates some parts. A `PATCH` request also deals with a single record.
`DELETE`	Instructs the API to delete a resource.

Example calls

HTTP method	Sample endpoints	Query string / payload

`GET`	`/api/menu-items`
	`/api/meu-items/1`
	`/api/menu-items?category=appetizers`
	`/api/menu-items?perpage=3&page=2`
		A `GET` call doesn't need a payload. However, `GET` calls can be accompanied by query string parameters and their values to filter the API output.

`POST`	`/api/menu-items`
	`/api/orders`
		Here's a sample JSON payload for the `/api/menu-items` endpoint to create a new resource:
		`{`
		`"title":"Beef Steak",`
		`"price": 5.50,`
		`"category":"main",`
		`}`

`PUT`	`/api/menu-items/1`
	`/api/orders/1`
		Here's a sample JSON payload for this endpoint `/api/menu-items/1` to completely replace it. Note that you need to supply all data for a `PUT` request.
		`{`
		`"title":"Chicken Steak",`
		`"price": 2.50,`
		`"category":"main",`
		`}`

`PATCH`	`/api/menu-items/1`
	`/api/orders/1`
		Here's a sample JSON payload for this endpoint `/api/menu-items/1` to partially update this resource
		`{`
		`"price": 3.00`
		`}`

HTTP method	Sample endpoints	Query string / payload

`DELETE`	`/api/menu-items`
	`/api/menu-items/1`
	`/api/orders`
	`/api/orders/1`
		When the `DELETE` call is sent to a collection endpoint, like `/api/menu-items` the API developer should delete the entire collection. When it is sent to a particular resource, like this, `/api/menu-items/1`, then the API developer should delete only that resource.

Status codes

Sending appropriate status codes with every API response is essential. And as a developer, you should not just pick any code. Every status code has meaning, so you should choose the most appropriate one based on the situation. Here's a list of the status code ranges and their purposes.

Status code range	Purpose
`100-199`	This range is mainly used to pass on some information. For example, sometimes an API needs time to process the request and it can't instantly deliver the result. In such a case, the API developer can set it to keep returning `102 – Processing` until the result is ready. This way, the client understands that the result isn't ready and should be checked again.
`200-299`	These are the success codes. If the client requests something and the API acts successfully, it should deliver the output with one of these status codes. For example, for a `PUT`, `PATCH`, or `DELETE` call, you can return `200 – Successful` if the operation was successful. For a successful `POST` call, you can set it to return a `201 – Created` status code when the resource has been created successfully.
`300-399`	These are the redirection codes. Suppose as an API developer, you changed the API endpoint from `/api/items to api/menu-items`. If the client makes an API call to `/api/items`, then you can redirect the client to this new endpoint `/api/menu-items` with a `301 – Permanently moved` status code so that the client can make new calls to that endpoint next time.

Status code range	Purpose
`400–499`	4xx status codes are used in the following situation: if the client requests something that does not exist, sends an invalid payload with insufficient data, or wants to perform an action that the client is not authorized for. For the above scenarios, the appropriate status codes will be: · **`404 - Not Found`** if the client requests something that doesn't exist, · **`400 - Bad Request`** if a client sends an invalid payload with insufficient data, · **`401 - Unauthorized`**, · **`403 - Forbidden`** if the client tries to perform an action it's not authorized for.
`500–599`	These alarming status codes are usually automatically generated on the server side if something goes wrong in the code, and the API developer doesn't write code to deal with those errors. For example, a client requests a non-existing resource, and the API developer tries to display that resource without adequately checking if that resource exists in the database. Or if the API developer didn't validate the incoming data and attempted to create a new resource with invalid or insufficient data. You, as an API developer, should always avoid 5xx errors.

Response types

These days, the most common response types involved with REST APIs are JSON, XML, plain text, and sometimes YAML. Frameworks like DRF come with built-in renderer classes that can convert the data into an appropriate format and display it correctly.

There are also third-party renderers available for this job. While making an API call, the client can specify its desired response format with the **`Accept`** HTTP header. And that header should be considered to deliver the result in that format using the render classes. Here's a list of HTTP headers for different response types.

Response type	Request header
HTML	**`Accept: text/html`**
JSON and JSONP	**`Accept: application/json`**
XML	**`Accept: application/xml`** **`Accept: text/xml`**
YAML	**`Accept: application/yaml`** **`Accept: application/x-yaml`** **`Accept: text/yaml`**

Conclusion

In this reading, you learned about different types of HTTP methods, status codes, and API response types.

Mark as completed