# 9 HTTP Methods

Stay ahead of the game with these **essential HTTP methods**!

swipe >

# HTTP Protocol

HTTP stands for Hyper Text Transfer Protocol. Communication between client and web servers is done by sending HTTP Requests and receiving HTTP Responses.

HTTP Protocol is used to perform

CRUD operations (Create, Read, Update, Delete) by sending HTTP requests with different HTTP methods.

HTTP methods are also called as HTTP verbs.

# HTTP Request Structure

Every HTTP request should have a URL or URI address and a method.

(Highlighted in purple)

Example URI

http://api.example.com/products?name=laptop&available=true

The format of a request message includes

- A request-line
- Zero or more header field(s) followed by CRLF (Carriage Return, Line Feed)
- An empty line
- A message-body (optional)

# HTTP Methods in REST

REST

REpresentational State Transfer

It is a set of architectural principles for designing web services.

Web services based on REST Architecture are known as RESTful web services.

REST makes it easy to share data between clients and servers.

REST applications use HTTP methods like GET, POST, DELETE, PUT, etc., to do CRUD operations.

swipe >

# Categories of HTTP Methods

You can divide HTTP methods into two main categories :

- Safe HTTP Methods
- Idempotent Methods

## Safe HTTP Methods

It doesn't change data on the server. It always returns the same response, no matter how many times it gets called.

Example - GET, HEAD

## Idempotent Methods

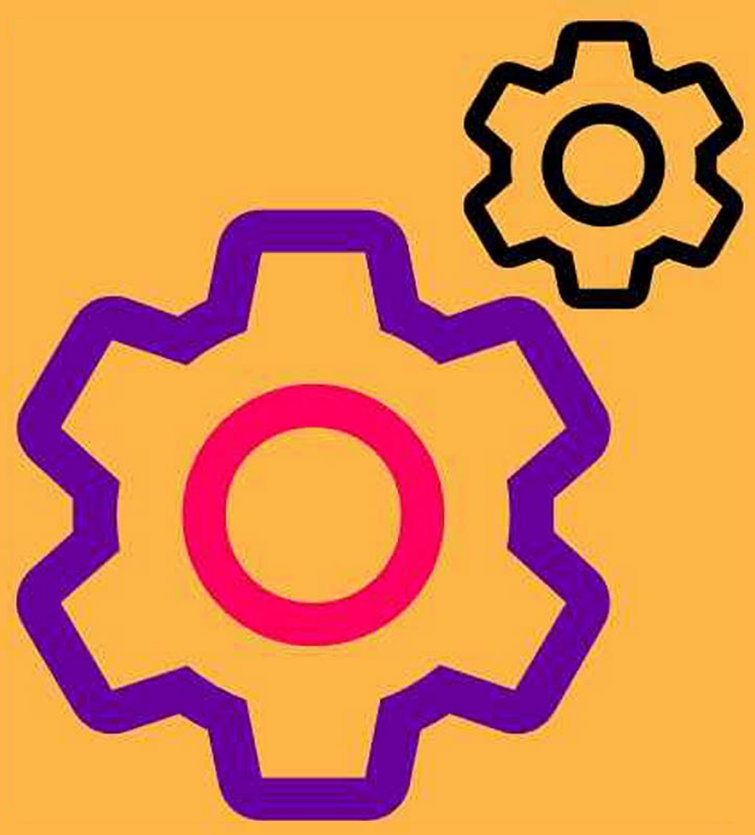
Example - GET, HEAD, PUT, DELETE, TRACE

It may change data on the server. It always returns the same response, no matter how many times it gets called.

All safe methods are also idempotent, but not all idempotent methods are safe.

swipe >

# The 9 HTTP Methods



- GET Method
- POST Method
- PUT Method
- PATCH Method
- DELETE Method
- HEAD Method
- OPTIONS Method
- TRACE Method
- CONNECT Method

Let us now see them in detail.

# GET Method

A GET Request is used to request information from a resource such as a website, a server, or an API.

Example

GET /api/employees/{employee-id}

Returns a specific employee by employee id.

GET /api/employees

Returns a list of all employees.

Since the GET method should never change the data on the resources and just read them(read-only), it is considered a safe method.

The GET method is also idempotent.

# Test an API with a GET Method

When we want to test an API, the most popular method that we would use is the GET method.

Therefore, We expect the following to happen

- If the resource is accessible, the API returns the 200 Status Code, which means OK.
- Along with the 200 Status Code, the server usually returns a response body in XML or JSON format. So, for example, we expect the [GET] /members endpoint to return a list of members in XML or JSON.
- If the server does not support the endpoint, the server returns the 404 Status Code, which means Not Found.
- If we send the request in the wrong syntax, the server returns the 400 Status Code, which means Bad Request.

swipe >

# POST Method

It creates a new resource on the backend (server). We send data to the server in the request body.

Example

POST /api/employees/department

Creates a department resource.

POST /api/employees/232/department/114/department-items

Creates a department item using the employee id and department id.

Two identical POST requests will create two new equivalent resources with the same data and different resource ids. We don´t get the same result every time.

It is neither a safe nor an idempotent method

# Testing a POST Endpoint

Since the POST method creates data, we must be cautious about changing the data. Testing all the POST methods in APIs is highly recommended.

Here are some suggestions that we can do for testing APIs with POST methods

- Create a resource with the POST method, and it should return the 201 Status Code.
- Perform the GET method to check if it created the resource was successfully created. You should get the 200 status code, and the response should contain the created resource.
- Perform the POST method with incorrect or wrong formatted data to check if the operation fails.

# PUT Method

Using this, we can update an existing resource by sending the updated data as the content of the request body to the server.

Example

PUT /api/employees/123

Update employee by employee id

If it applies to a collection of resources, it replaces the whole collection, so be careful using it. The server will return the 200 or 204 status codes after updating.

The PUT method is idempotent but not safe.

# Test an API with a PUT Method

The PUT method is idempotent, and it modifies the entire resources.

Make sure to do the following operations

- Send a PUT request to the server many times, and it should always return the same result.
- When the server completes the PUT request and updates the resource, the response should come with 200 or 204 status codes.
- After the server completes the PUT request, make a GET request to check if the data is updated correctly on the resource.
- If the input is invalid or has the wrong format, the resource must not be updated.

# PATCH Method

Similar to PUT, PATCH updates a resource, but it updates data partially and not entirely.

Example

```
PATCH /api/employees/123
{
"name" : "Brij"
}
```

Updates name for employee id 123.

The PATCH method updates the provided fields of the employee entity. In general, this modification should be in a standard format like JSON or XML.

It is neither a safe nor an idempotent method

# Test an API with a PATCH Method

To test an API with the PATCH method, follow the steps for the testing API with the PUT and the POST methods.

Consider the following results

- Send a PATCH request to the server. The server will return the 2xx HTTP status code, which means, the request is successfully received, understood, and accepted.
- Perform the GET request and verify that the content is updated correctly.
- If the request payload is incorrect or ill-formatted, the operation must fail.

# DELETE Method

The DELETE method deletes a resource. Regardless of the number of calls, it returns the same result.

Example

DELETE /api/employees/235

Delete employee by employee id.

Most APIs always return the 200 status code even if we try to delete a deleted resource but in some APIs, If the target data no longer exists, the method call would return a 404 status code.

The DELETE method is idempotent but not safe.

# Testing a DELETE Endpoint

When it comes to deleting something on the server, we should be cautious. We are deleting data, and it is critical.

Then perform the following actions

- Call the POST method to create a new resource. Never test DELETE with actual data. For example, first, create a new employee and then try to delete the employee you just created.
- Make the DELETE request for a specific resource. For example, the request [DELETE] /employees/{employee-id} deletes a employee with the specified employee id.
- Call the GET method for the deleted employee, which should return 404, as the resource no longer exists.

# HEAD Method

The HEAD method is similar to the GET method. But it doesn't have any response body, so if it mistakenly returns the response body, it must be ignored.

Example

HEAD /api/employees

Similar to GET, but it does not return the list of emloyees.

Before requesting the GET endpoint, we can make a HEAD request to determine the size (Content-length) of the file or data that we are downloading.

The HEAD method is safe and idempotent.

# Testing a HEAD Endpoint

One of the advantages of the HEAD method is that we can test the server if it is available and accessible as long as the API supports it.

The API can be tested as follows

- It is much faster than the GET method because it has no response body.
- The status code we expect to get from the API is 200.
- Before every other HTTP method, we can first test API with the HEAD method.

# OPTIONS Method

This method is used to get information about the possible communication options (permitted HTTP methods) for the given URL or an asterisk to refer to the entire server.

Example

OPTIONS /api/main.html/1.1

Returns permitted HTTP method in this URL

OPTIONS * HTTP/1.1

Returns all permitted methods

Various browsers widely use the OPTIONS method to check whether the CORS (Cross-Origin resource sharing) operation is restricted on the targeted API or not.

The OPTIONS method is safe and idempotent

# Testing an OPTIONS Endpoint

Depending on whether the server supports the OPTIONS method, we can test the server for the times of FATAL failure with the OPTIONS method.

To try it, consider the following

- Make an OPTIONS request and check the header and the status code that returns.
- Test the case of failure with a resource that doesn't support the OPTIONS method.

# TRACE Method

The TRACE method is for diagnosis purposes. It creates a loop-back test with the same request body that the client sent to the server before, and the successful response code is 200 OK.

Example

TRACE /api/main.html

Responds the exact request that client sent.

The TRACE method could be dangerous because it could reveal credentials. A hacker could steal credentials, including internal authentication headers, using a client-side attack.

The TRACE method is safe and idempotent

# Test an API with a TRACE Method

- Make a standard HTTP request like a GET request to /api/status
- Replace GET with the TRACE and send it again.
- Check what the server returns. If the response has the same information as the original request, the TRACE ability is enabled in the server and works correctly.

# CONNECT Method

The CONNECT method is for making end-to-end connections between a client and a server. It makes a two-way connection like a tunnel between them.

Example

CONNECT www.example.com:443 HTTP/1.1

Connects to the URL provided.

For example, we can use this method to safely transfer a large file between the client and the server.

It is neither a safe nor an idempotent method

These are the 9 HTTP methods, their uses and a guide on how to test them. Hope you learned about them in detail.

swipe >

brijpandeyji

# For More Interesting Content

brijpandeyji

## Follow Me On LinkedIn

https://www.linkedin.com/in/brijpandeyji/