HTTP Request Lifestyle

.NET CORE

HTTP sessions consist of three phases.
1. The client establishes a (usually) TCP connection.
2. The client sends its request and waits for the answer.
3. The server processes the request, sending back its answer, with a status code and data.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

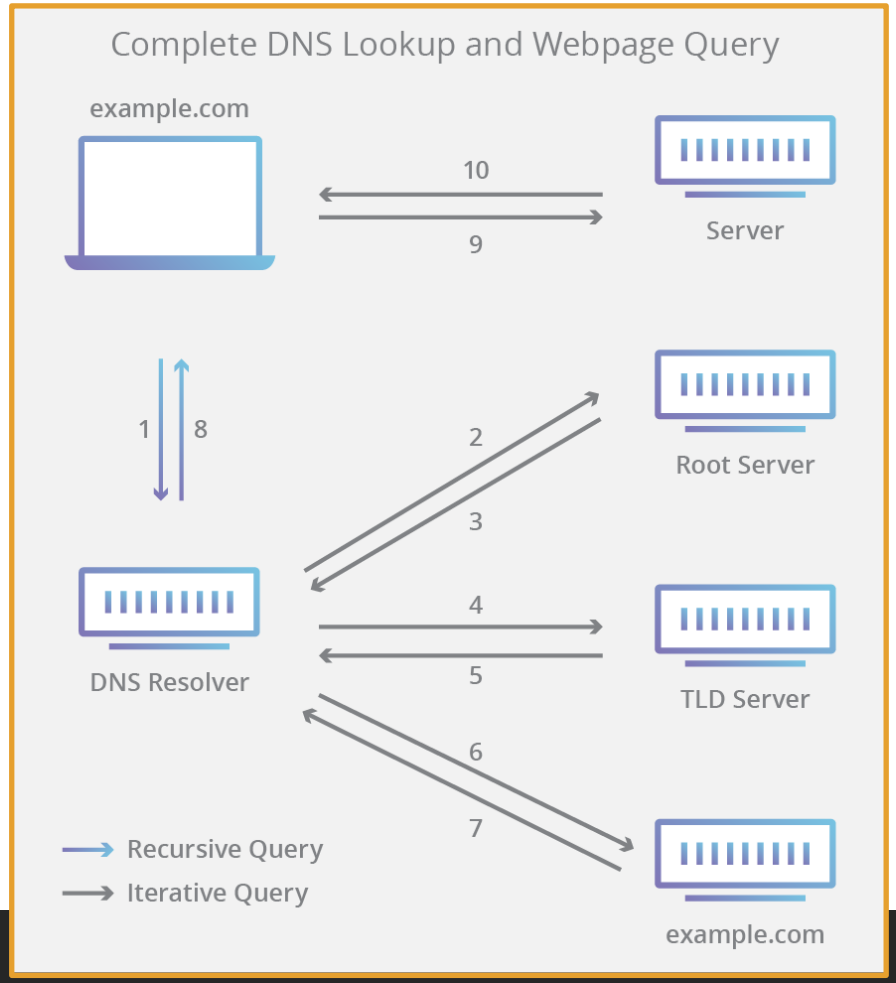
DNS (Domain Name System)

https://en.wikipedia.org/wiki/Domain_Name_System

The ***Domain Name System (DNS)*** is essentially a directory of names and IP addresses.

The DNS translates domain names (www.revature.com) to numerical IP addresses (255.255.255) for locating and identifying computer services and devices over the web.

The DNS also associates identifying data with the unique domain names assigned to each connected entity.

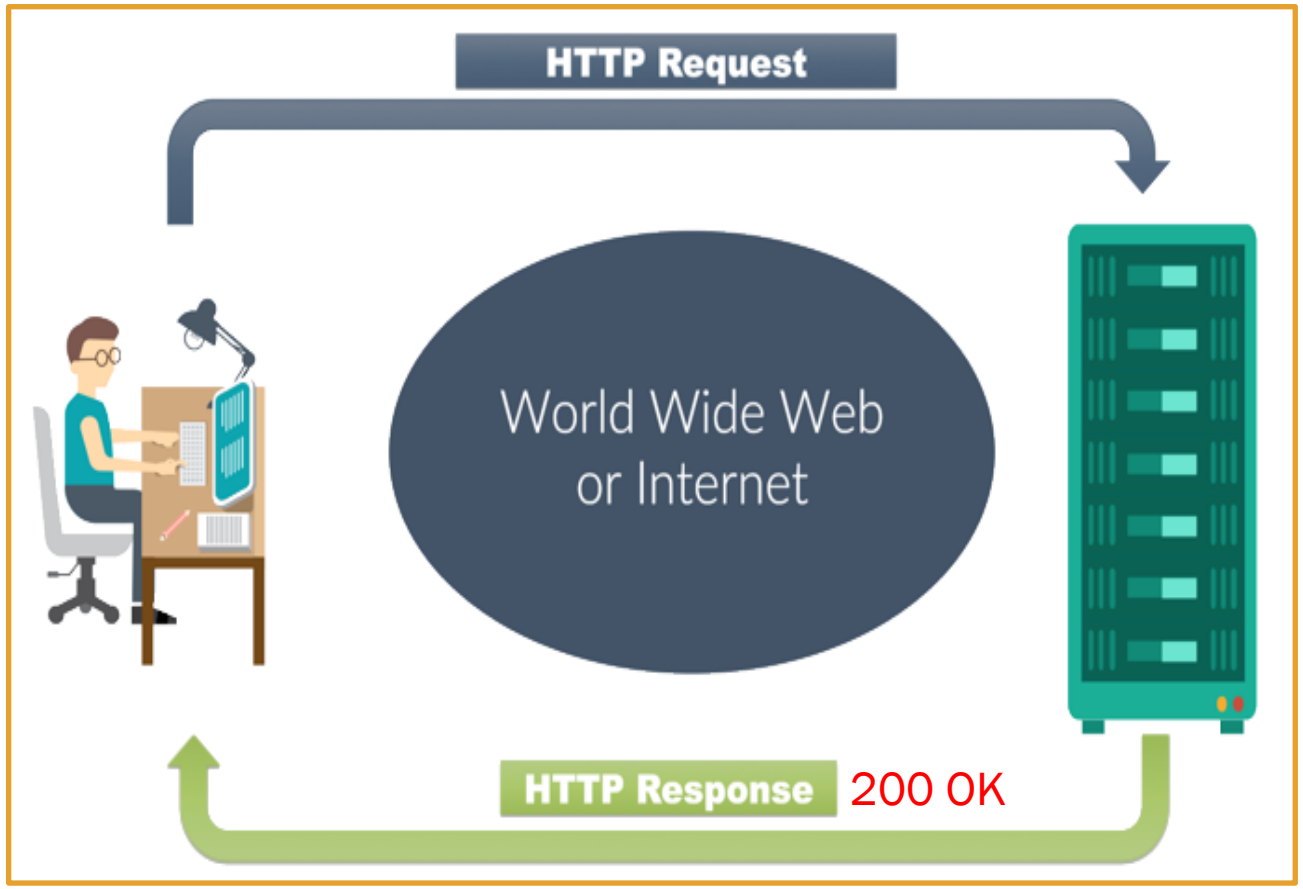


HTTP Request LifeCycle

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

HTTP sessions consist of three basic phases:

1. The client establishes a TCP connection.
2. The client sends its ***Request*** and waits for the ***Response***.
3. The server processes the request and sends back its ***Response*** with a ***status code*** and appropriate data.



Establish a connection

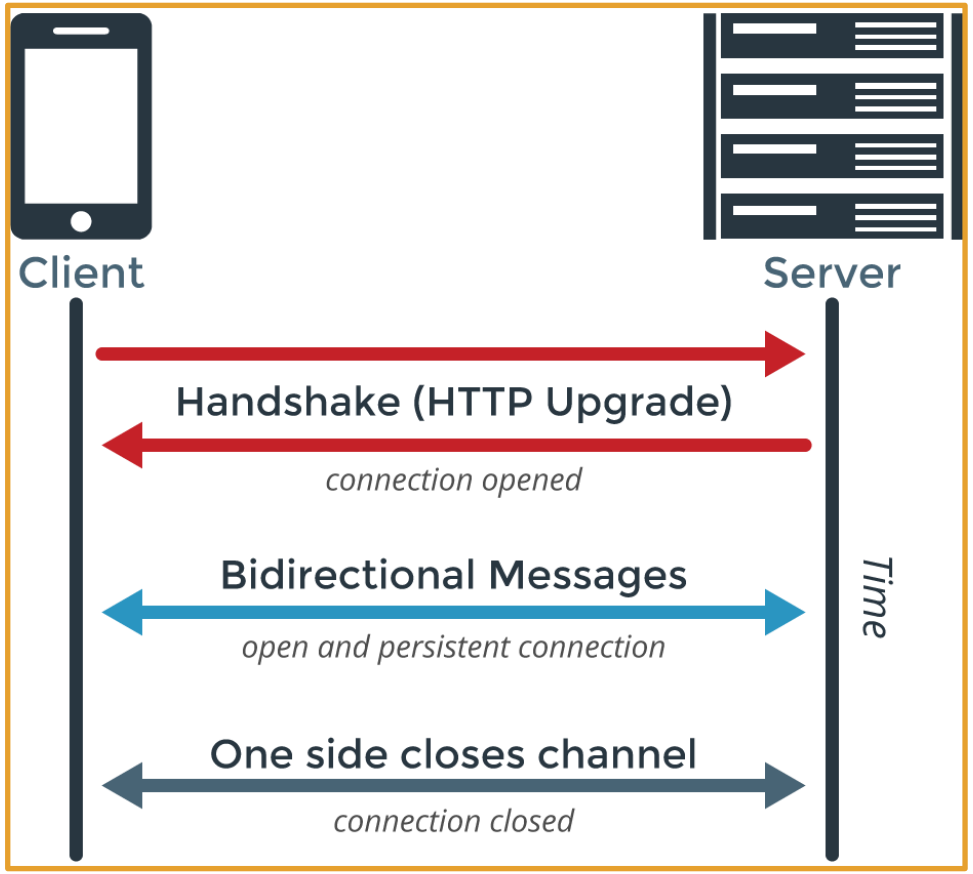
https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

1. The client establishes the connection.
   - The HTTP default port is port :80.
2. The client sends its request and waits for the answer.
   - The ***URL*** of a page to fetch contains both the domain name, and the port number.
3. The server processes the request, sending back its answer, providing a status code and appropriate data.

The client-server model requires an explicit ***Request***.

Workarounds to this limitation are:

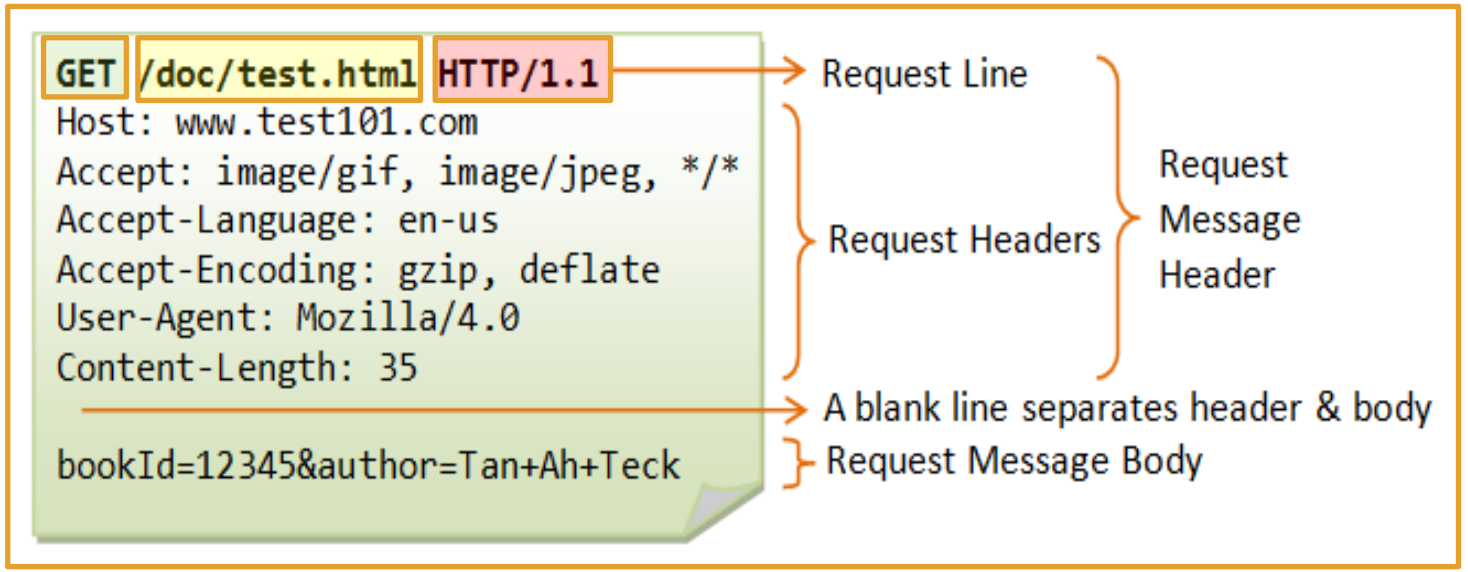
- ping the server periodically via the XMLHTTPRequest,
- Fetch APIs,
- using the WebSockets API



Connecting and Sending A Request

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

- The client always establishes the connection. The client-server model does not allow the server to send data to the client without an explicit Request.
- The HTTP default port is port :80.
- The URL of a page to fetch contains both the domain name, and the port number.



On connection, the web browser sends the request. A client request consists of text directives, separated by CRLF (Carriage Return, Line Feed), divided into three blocks:

Line 1 - request method followed by the absolute URL doc path without the protocol or domain name and the HTTP protocol version.

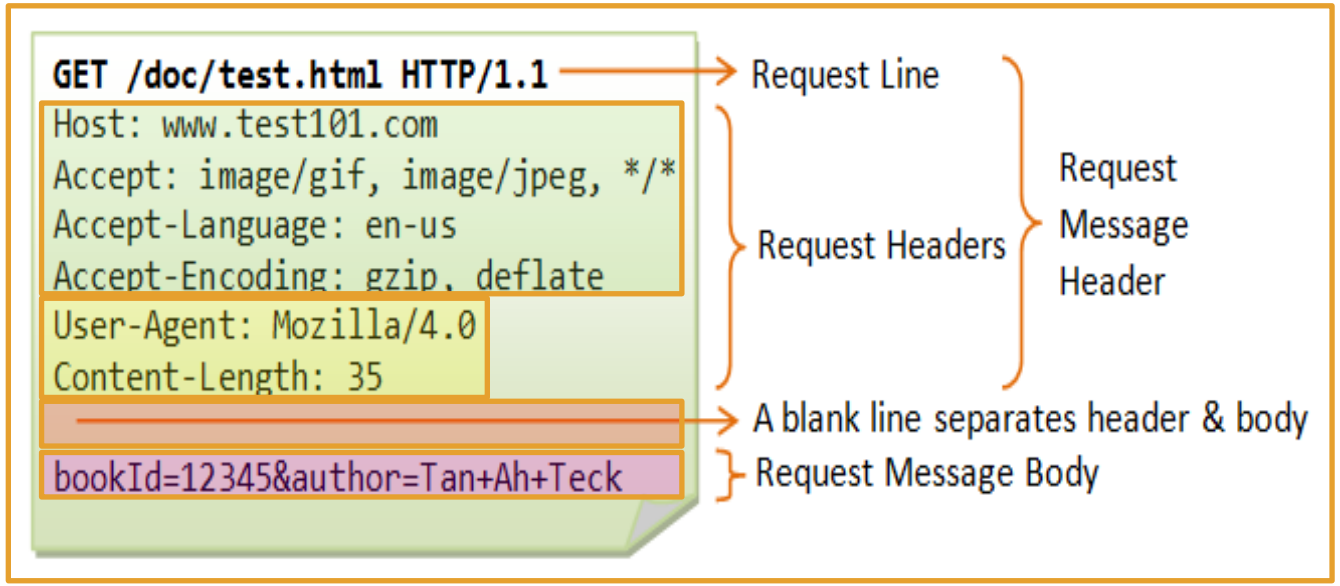
Sending A Request

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

An HTTP header gives the server:

- information about what type of data is appropriate (e.g., what language, what MIME types, etc)
- other data which informs and may alter its behavior (e.g., not sending an answer if it is already cached).

Then, there's an empty line and an (optional) **data** block, mainly used by the POST method which contains further data.



```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Connection: keep-alive
Cache-Control: s-maxage=300, public, max-age=0
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT
ETag: "2e77ad1dc6ab0b53a2996dfd4653c1c3"
Server: meinheld/0.6.1
Strict-Transport-Security: max-age=63072000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding,Cookie
Age: 7
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>A simple webpage</title>
</head>
<body>
  <h1>Simple HTML5 webpage</h1>
  <p>Hello, world!</p>
</body>
</html>
```

Response

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

The server processes the ***Request*** and returns a ***Response***. A server ***Response*** is formed of text directives, separated by CRLF, divided into three blocks:

Line 1 (the status line) :an acknowledgment of the HTTP version used, and a request ***status code*** (and its meaning).

Subsequent lines represent specific HTTP headers, giving the client information like data type and size, compression algorithm used, hints about caching, etc. It ends with an empty line.

The final block is a data block which contains the (optional) data

Request Methods

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

Request ***methods*** (***HTTP verbs***) indicate the desired action to be performed on a resource. The most common requests are ***GET*** and ***POST***. There are also ***PUT***, ***DELETE***, ***TRACE***, ***HEAD***, ***CONNECT***, ***OPTIONS***.

- The ***POST*** method sends data to a server. ***POST*** is used mainly for HTML ***Forms***.
- The ***GET*** method requests/retrieves data.

```
1  POST /contact_form.php HTTP/1.1
2  Host: developer.mozilla.org
3  Content-Length: 64
4  Content-Type: application/x-www-form-urlencoded
5
6  name=Joe%20User&request=Send%20me%20one%20of%20your%20catalogue
```

```
1  GET / HTTP/1.1
2  Host: developer.mozilla.org
3  Accept-Language: fr
```

Request Methods

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

HTTP Verb	Description
*GET	Requests a representation of the specified resource. GET should only retrieve data.
HEAD	Asks for a response identical to that of a GET request, but without the response body.
*POST	Used to submit an entity to the specified resource.
*PUT	Replaces all current representations of the target resource with the request payload.
*DELETE	Deletes the specified resource.
CONNECT	Establishes a tunnel to the server identified by the target resource.
OPTIONS	Describes the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.
PATCH	Applies partial modifications to a resource.

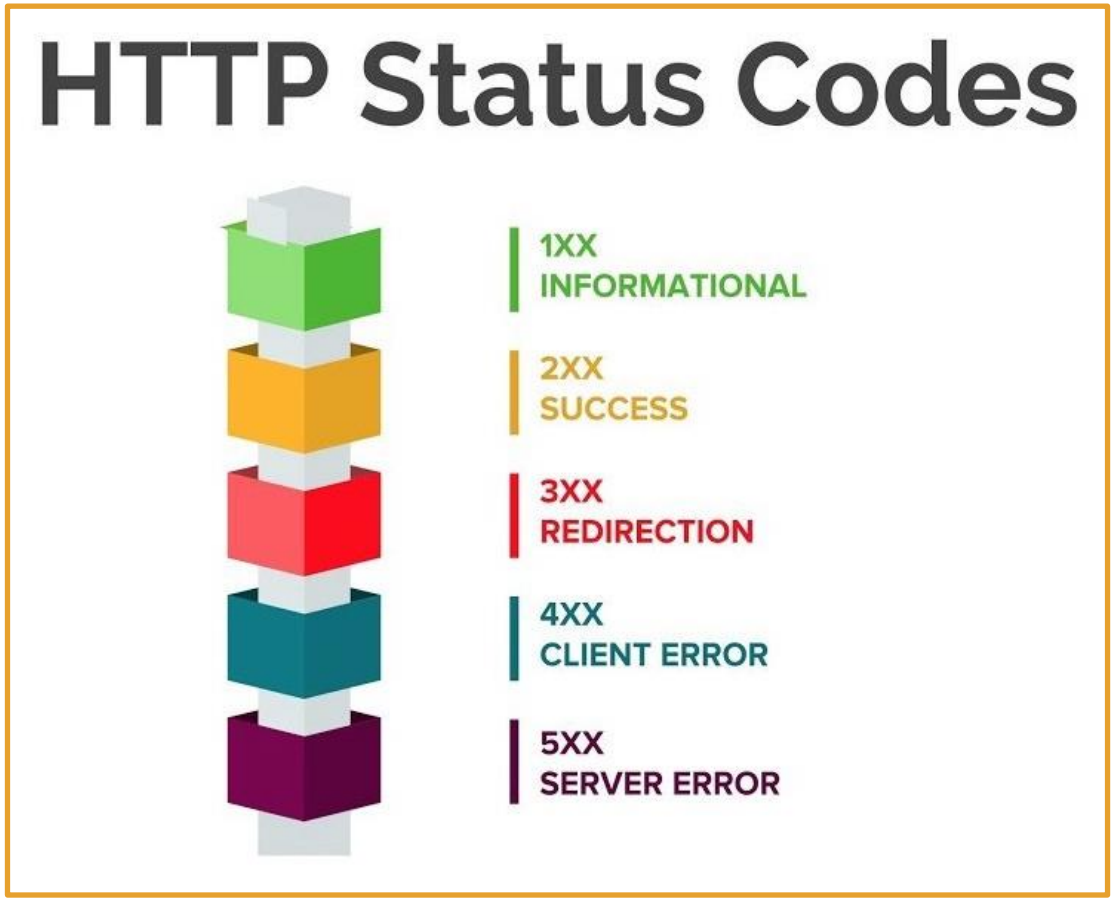
*most common

Response Status Codes

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
https://tools.ietf.org/html/rfc2616#section-10

HTTP response status codes give the result of an HTTP request. Responses are grouped in five classes:

- Informational responses (100–199),
- Successful responses (200–299),
- Redirects (300–399),
- Client errors (400–499),
- and Server errors (500–599).
- Cheat Sheet



Response Status Codes

https://www.smartlabsoftware.com/ref/http-status-codes.htm
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Code number	Code Meaning
200 OK, 201	The request has succeeded., request has been fulfilled resulting in new resource(s) created.
300 Multiple Choices	The requested resource has different choices and cannot be resolved into one.
301 Moved Permanently	The requested resource has been assigned a new permanent URI
304 Not Modified	Client performed a conditional GET request. Access is allowed. The document is unmodified
307 Temporary Redirect	The requested resource resides temporarily under a different URI.
400 Bad Request	The request could not be understood by the server due to malformed syntax.
401 Unauthorized	The request requires user authentication.
403 Forbidden	The server understood the request but is refusing to fulfill it.
404 Not Found	The server has not found anything matching the Request-URI.
410 Gone	The requested resource is no longer available at the server and no forwarding address is known.
500 Internal Server Error	The server encountered an unexpected condition which prevented it from fulfilling the request.
501 Not Implemented	The server does not support the functionality required to fulfill the request.
503 Service Unavailable	Your web server is unable to handle your HTTP request at the time.
550 Permission Denied	Your account does not have permission to perform the action you are attempting.

Safe and Idempotent

https://developer.mozilla.org/en-US/docs/Glossary/safe
https://developer.mozilla.org/en-US/docs/Glossary/Idempotent

An HTTP method is ***safe*** if it doesn't alter the state of the server. In other words, a method is ***safe*** if it leads to a read-only operation. Several common HTTP methods are ***safe***: ***GET***, ***HEAD***, or ***OPTIONS***.

All ***safe*** methods are also ***idempotent***, but not all ***idempotent*** methods are ***safe***. For example, ***PUT*** and ***DELETE*** are both ***idempotent*** but unsafe.

idempotent - an identical request can be made once or several times in a row with the same effect while leaving the server in the same state. Implemented correctly, the ***GET***, ***HEAD***, ***PUT***, and ***DELETE*** method are ***idempotent***, but not the ***POST*** method. All ***safe*** methods are ***idempotent***.

- Description of common idempotent methods: `GET`, `HEAD`, `PUT`, `DELETE`, `OPTIONS`, `TRACE`
- Description of common non-idempotent methods: `POST`, `PATCH`, `CONNECT`

Additional Resources

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#HTTP_Messages

https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design