HTTP

Written by [Febogamedev]

HTTP, an acronym for Hypertext Transfer Protocol, is an application-layer protocol used to exchange information on the Web (Fielding et al., 2022c).
Although its name refers to hypertext, HTTP does not transfer only HTML documents. It can be used to request and transmit various Web resources (images, CSS stylesheets, fonts, etc.) (Fielding et al., 2022c).

HTTP is a family of protocols based on a request-response communication model: a client sends a request and a server returns a response (Fielding et al., 2022c).
The different versions retain the same fundamental concepts, but represent and transport messages in different ways (Fielding et al., 2022a; Thomson & Benfield, 2022; Bishop, 2022).

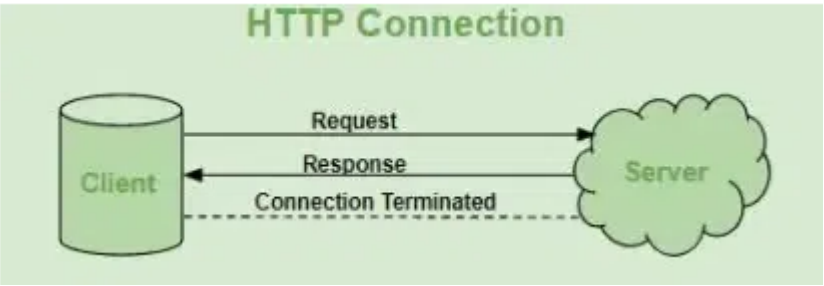
Today, HTTP is normally used through *HTTPS*.
HTTPS indicates *HTTP* communication secured through *TLS*, a cryptographic protocol that protects the transmitted data (Fielding et al., 2022c; Rescorla, 2026).
The browser verifies the identity of the remote service, and the data exchanged is protected against interception and modification during transfer (Rescorla, 2026).

HTTP is the main protocol for transmitting information on the Web in a client-server architecture: the client makes a request and the server returns the response (Fielding et al., 2022c).

A service accessed through *HTTP* normally uses the default port 80, while an *HTTPS* service normally uses port 443 (Internet Assigned Numbers Authority, 2026).

There are two types of HTTP messages (Fielding et al., 2022c):

- request messages: the client makes the request to the server.
- response messages: the server responds to the client.



Connections can be kept open and reused for several requests (persistent connections), or closed when they are no longer needed (Fielding et al., 2022a).

It is important to note that if a Web page consists of images, videos or other external resources, the browser interpreting that page makes an *HTTP* request for each resource (WHATWG, 2026).

The HTTP request and response are examined below. For simplicity, we will refer to the representation used in *HTTP/1.1 (Fielding et al., 2022a)*.

HTTP request

An HTTP request is a message through which a client asks a server to perform an operation relating to a resource (Fielding et al., 2022c).

The client can ask, for example, to:
- Obtain a Web resource.
- Send form data.
- Create or modify a resource.
- Delete a resource.
- Obtain information without transferring the full content.

A request can be represented textually through (Fielding et al., 2022a):
- Request line.
- Headers.
- Blank line (the final blank line separates the headers from any request content).
- Optional body.

Request line

The first line is called the request line (Fielding et al., 2022a).
It consists of: METHOD TARGET VERSION (Fielding et al., 2022a).

Example
GET /product?id=123 HTTP/1.1

Method
The HTTP method indicates which operation the client asks to apply to the target resource (Fielding et al., 2022c).
Standard method names are written in uppercase (Fielding et al., 2022c).

GET
GET requests a current representation of the resource from the server.

GET is considered a safe method in HTTP semantics: the client is not asking the server to change the state of the resource. This does not prevent secondary activities such as recording the request in logs, but the requested operation must be essentially read-only (Fielding et al., 2022c).

HEAD
HEAD is similar to GET, but the server does not return the content of the representation.

It can be used to obtain metadata such as:
- media type.
- size.
- date of last modification.
- cache information.

The server should return, as far as possible, the same headers that it would have returned in response to a GET request (Fielding et al., 2022c).

POST
POST asks the target resource to process the data contained in the request according to its own rules.

It can be used, for example, to:
- Submit a form.
- Create a new comment.
- Place an order.
- Upload a file.
- Start a processing operation.

The result depends on the function defined by the application for that specific *URL (Fielding et al., 2022c)*.

PUT
PUT asks to create or replace the state of the resource identified by the URL.
In this case, the client already knows the exact address of the resource it wants to create or replace.

PUT does not necessarily mean “upload a file”: the content can represent any state recognised by the application (Fielding et al., 2022c).

PATCH
PATCH is used to apply partial modifications to a resource.

For example, instead of sending an entire user profile again, a PATCH request could ask to change only the image or display name (Dusseault & Snell, 2010).

DELETE
DELETE asks to remove the association between the URL and the current resource.
How the application performs this operation depends on the server. The data could be deleted, archived or simply marked as no longer available (Fielding et al., 2022c).

Request target

The second part of the line indicates the request target, that is, the part of the request that identifies or locates its recipient/target; in its most common form, it includes the path and any query (Fielding et al., 2022a).

For example, in the request “/product?id=123”, we can distinguish:

- Path: /product.
- Query: id=123.

Note that the URL fragment, that is, the part beginning with #, is not sent to the server: it is reserved for processing by the client (Berners-Lee et al., 2005).

Protocol version
In the request, the third part indicates the protocol version, in our case HTTP/1.1.

Request headers

Headers, or header fields, convey additional information about the request and the context in which it is made.
Each field consists of a name and a value: Field-Name: value (Fielding et al., 2022a).

Host
This field identifies the host and any port of the requested resource.
It is essential when the same infrastructure manages several domain names.
A request without Host must be considered invalid by the server (Fielding et al., 2022a).

User-Agent
Identifies the client program that sent the request.

It can contain the product name and version, but should not be considered a complete and consistently reliable description of the user’s device (Fielding et al., 2022c).

Accept
Indicates the media types preferred by the client for the response.
Relative preferences can also be expressed.
For example: Accept: text/html, application/json;q=0.8
The value q=0.8 indicates a lower preference than the type text/html, to which the default value 1 is applied because q is not specified (Fielding et al., 2022c).

Content-Type
When the request has content, Content-Type describes its format (Fielding et al., 2022c):
For example: Content-Type: application/json.

Request body

The body contains any data sent by the client to the server.
The meaning of the content depends on the method and the declared media type (Fielding et al., 2022c).

It can contain:

- Form fields.
- JSON data.
- An image.
- A document.
- A file.
- Instructions for modifying a resource.

The body is frequently used with POST, PUT and PATCH, but its presence does not depend solely on the method name: the method's semantics determine how it should be interpreted (Fielding et al., 2022c).

Request example

```
GET /product?id=123 HTTP/1.1
Host: www.example.com
User-Agent: ExampleBrowser/1.0
Accept: text/html
```

Analysis:

- Method: GET, requests a representation of the resource.
- Path: /product.
- Query: id=123.
- Version: HTTP/1.1.
- Host: www.example.com.
- User-Agent: identifies the client program.
- Accept: indicates that the client prefers to receive HTML.

HTTP response

An *HTTP* response is the message through which a server communicates the outcome of a request to the client (Fielding et al., 2022c).
The response can contain:

- The result of the operation.
- A representation of the requested resource.
- Information useful to the client.
- A redirect.
- A description of an error.

For example, the server can return:

- A Web resource.
- Confirmation that a form has been submitted.
- An indication that the resource is at another URL.
- A message reporting a problem.

A response is represented textually as follows (Fielding et al., 2022a):

- Status line
- Headers
- Blank line
- Optional body

Status line

The first line of the response is called the status line.
It consists of: VERSION CODE PHRASE (Fielding et al., 2022a).

Example
HTTP/1.1 404 Not Found

Status codes
The status code is a three-digit number that describes the result of the request.

The first digit indicates the general category:

Category	General meaning
1xx	Interim information
2xx	Request processed successfully
3xx	Redirection or use of a previous response
4xx	Problem attributed to the client's request
5xx	Problem encountered by the server

The code does not describe every detail of the application, but communicates a general meaning valid across different HTTP systems (Fielding et al., 2022c).

Common codes:
- 200 OK: Indicates that the request has been processed successfully. The precise meaning of the content depends on the method used. In response to a GET, the body normally contains a representation of the requested resource (Fielding et al., 2022c).
- 201 Created: Indicates that the request has been processed successfully and has led to the creation of a new resource. The response can use the Location header to indicate the URL of the created resource (Fielding et al., 2022c).
- 204 No Content: Indicates that the request has been processed successfully, but that the response contains no body. It can be used, for example, after saving a change when there is no need to send new content to the client (Fielding et al., 2022c).
- 301 Moved Permanently: Indicates that the resource has been assigned a new permanent URL. The new destination is normally indicated through the Location header. Future links should use the new address (Fielding et al., 2022c).

- 302 Found: Indicates that the resource is temporarily at a different URL. The client should continue to use the original URL in future requests, because the redirect could change. Browsers normally follow the redirect automatically (Fielding et al., 2022c).
- 304 Not Modified: Indicates that the client can reuse a response already stored in the cache because the resource has not changed. The response does not contain a new complete copy of the resource. This reduces the amount of data transferred (Fielding et al., 2022b; Fielding et al., 2022c).
- 400 Bad Request: Indicates that the server cannot or will not process the request because of a problem attributed to the client, for example invalid syntax, incorrect message structure, misleading or inconsistent routing data (Fielding et al., 2022c).
- 401 Unauthorized: Indicates that the request does not have valid authentication credentials (Fielding et al., 2022c).
- 403 Forbidden: Indicates that the server has understood the request but refuses to fulfil it. The user could be authenticated but lack the necessary permissions (Fielding et al., 2022c).
- 404 Not Found: indicates that the server has not found an available resource for the requested address or does not wish to disclose that it exists. An incorrect URL is one possible cause, but not the only one (Fielding et al., 2022c).
- 429 Too Many Requests: Indicates that the client has sent too many requests within a certain period of time. It can be used by services to limit abuse or overload (Nottingham & Fielding, 2012).
- 500 Internal Server Error: Indicates that the server encountered an unexpected condition that prevented it from completing the request. It does not specify the actual problem: it could depend on the application code, configuration or a service used by the server (Fielding et al., 2022c).
- 502 Bad Gateway: Indicates that a server acting as a gateway or proxy received an invalid response from another server involved in processing (Fielding et al., 2022c).
- 503 Service Unavailable: Indicates that the service is temporarily unable to handle the request, for example because of overload, maintenance or temporary unavailability. The response can contain the Retry-After header, which suggests when to try again (Fielding et al., 2022c).
- 505 HTTP Version Not Supported: Indicates that the server does not support, or refuses to support, the major version of HTTP used in the request. It is a valid code, but considerably less common in users' everyday experience (Fielding et al., 2022c).

Reason phrase
The reason phrase, such as "OK" or "Not Found", is primarily intended for human readability, but does not constitute a reliable channel for conveying information.
It can be changed, translated, omitted or removed during conversion between different versions of *HTTP*.
The client must therefore interpret the numeric code rather than rely on the textual phrase (Fielding et al., 2022a).

Response headers

Response headers convey additional information about the response, the content and instructions intended for the client.
Each field consists of a name and a value: Field-Name: value.

Headers are separated from one another by new lines. A blank line indicates the end of the header section (Fielding et al., 2022a).

Content-Type
Describes the media type of the content in the body.

It can indicate, for example:
- Content-Type: image/webp
- Content-Type: application/json
- Content-Type: application/pdf

The browser uses this information to understand how to interpret the data received (Fielding et al., 2022c).

Content-Length
When present and applicable, it indicates the length of the content in bytes.
However, it is not mandatory in every response.
It may be absent, for example, when the length is not known at the time the headers are sent or when the protocol uses other mechanisms to delimit the data.
In some responses, such as 204 No Content, it must not be sent (Fielding et al., 2022c).

Location
Indicates another URL related to the result of the response.

It is mainly used:
- In redirects.
- To indicate a newly created resource (Fielding et al., 2022c).

Cache-Control
Communicates rules concerning the storage and reuse of the response in the cache.

It can, for example, indicate how long the response can be reused before requesting validation from the server (Fielding et al., 2022b).

Content-Encoding
Indicates that the content was encoded or compressed before transmission.
The browser must apply the corresponding decoding before using the content (Fielding et al., 2022c).

Set-Cookie
Asks the browser to store a cookie according to the specified conditions.
The cookie can be sent in subsequent requests when the rules concerning domain, path and security allow it (Barth, 2011).

Server
Can indicate information about the software that handled the request.

It is an optional field: the server may omit it or provide generic information. Exposing the software's name and detailed version is not necessary for the response to function (Fielding et al., 2022c).

Response body

The response body contains any data sent by the server to the client.

It can contain:

- HTML code.
- CSS.
- JavaScript.
- An image.
- JSON data.
- A PDF file.
- Audio or video.
- An explanation of the error.

The format is normally described by the Content-Type header (Fielding et al., 2022c).
Not all responses contain a body.

For example:

- A response to HEAD does not return the content;
- 204 No Content contains no content;
- 304 Not Modified indicates that the stored copy should be reused;
- Some informational 1xx responses do not contain final content.

The presence of the body therefore depends on the request method, the status code and the rules of the specification for *HTTP (Fielding et al., 2022c)*.

Response example

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=3600

<!doctype html>
<html lang="it">

<head>
  <meta charset="utf-8">
  <title>Welcome</title>
</head>

<body>
  <h1>Hello, world!</h1>
  <p>Welcome to the example page.</p>
```

```
</body>

</html>
```

Analysis:

- Version: HTTP/1.1.
- Status code: 200.
- Reason phrase: OK.
- Content-Type: the body contains HTML encoded in UTF-8.
- Cache-Control: the response can be reused according to the indicated rule.
- Body: contains the HTML document.
- Content-Length: it has not been included because it is not necessary to understand the example and should correspond exactly to the bytes transmitted.

Supplementary material

Exercises

Building the correct request

A Web application must allow a user to:

- View the product with ID 123.
- Obtain only the metadata of the same resource.
- Submit a new comment.
- Change only the display name in their profile.
- Completely replace a resource already identified by a precise URL.
- Delete a resource.

For each operation:

- Choose the most appropriate method from GET, HEAD, POST, PUT, PATCH and DELETE.
- Justify your choice.
- Indicate in which cases the presence of a body in the request is plausible.
- To view the product, construct a possible request line using “/product?id=123”.
- Break down the resulting line into method, request target and version.
- Distinguish the path and query in the request target.
- Explain why a fragment such as #recensioni should not appear in the request sent to the server.
- Add at least three appropriate headers from Host, User-Agent, Accept and Content-Type.
- Explain the difference between Accept and Content-Type.
- Finally, explain why GET is considered a safe method in HTTP semantics.

Diagnosing server responses

A browser makes several requests and receives these responses:

- Case A: 200 OK
- Case B: 201 Created with a Location header.
- Case C: 301 Moved Permanently.
- Case D: 304 Not Modified.
- Case E: 401 Unauthorized.
- Case F: 403 Forbidden.
- Case G: 404 Not Found.
- Case H: 429 Too Many Requests.
- Case I: 500 Internal Server Error.
- Case L: 502 Bad Gateway.
- Case M: 503 Service Unavailable.

For each case:

- Indicate which category it belongs to among 2xx, 3xx, 4xx and 5xx.
- Explain the general meaning of the code.
- Indicate which cases primarily represent a problem attributed to the client.

- Indicate which cases represent a server-side or infrastructure problem.
- Explain the difference between 401 Unauthorized and 403 Forbidden.
- Explain the difference between 301 Moved Permanently and 302 Found.
- Explain why 304 Not Modified can reduce the amount of data transferred.
- Explain in which situation 201 Created can use the Location header.
- Explain the conceptual difference between 500, 502 and 503.
- Explain why a program should rely primarily on the numeric code rather than on the reason phrase, such as OK or Not Found.

Analysing a complete HTTP communication

Consider this request:

```
GET /catalogo?categoria=libri HTTP/1.1
Host: shop.example.com
User-Agent: ExampleBrowser/1.0
Accept: text/html, application/json;q=0.8
```

The server responds:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Cache-Control: max-age=3600
Set-Cookie: sessione=abc123

<!doctype html>
<html lang="it">
...
</html>
```

Analyse the entire exchange.

- Identify the method, path, query, version and host in the request.
- Explain what the Accept header communicates.
- Interpret the value q=0.8.
- Explain why the request does not necessarily contain a body.
- Identify the version, status code and reason phrase in the response.
- Explain what Content-Type: text/html; charset=utf-8 communicates.
- Explain what function Cache-Control: max-age=3600 performs.
- Explain what Set-Cookie asks the browser to do.
- Identify the response body.
- Explain why not all HTTP responses necessarily have to contain a body, giving at least three examples from the handout.
- Imagine that the browser finds images, CSS and JavaScript in the HTML. Explain why the page can generate further separate HTTP requests.
- Explain why an HTTP connection can be kept open and reused for subsequent requests.
- Conclude by reconstructing the complete cycle: request → server-side processing → response → client interpretation → possible new requests.

Bibliography

Barth, A. (2011). *HTTP State Management Mechanism (RFC 6265)*. RFC Editor.

Berners-Lee, T., Fielding, R., & Masinter, L. (2005). *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986)*. RFC Editor.

Bishop, M. (2022). *HTTP/3 (RFC 9114)*. RFC Editor.

Dusseault, L., & Snell, J. (2010). *PATCH Method for HTTP (RFC 5789)*. RFC Editor.

Fielding, R., Nottingham, M., & Reschke, J. (2022a). *HTTP/1.1 (RFC 9112)*. RFC Editor.

Fielding, R., Nottingham, M., & Reschke, J. (2022b). *HTTP Caching (RFC 9111)*. RFC Editor.

Fielding, R., Nottingham, M., & Reschke, J. (2022c). *HTTP Semantics (RFC 9110)*. RFC Editor.

Internet Assigned Numbers Authority. (2026). *Service Name and Transport Protocol Port Number Registry*. IANA.

Nottingham, M., & Fielding, R. (2012). *Additional HTTP Status Codes (RFC 6585)*. RFC Editor.

Rescorla, E. (2026). *The Transport Layer Security (TLS) Protocol Version 1.3 (RFC 9846)*. RFC Editor.

Thomson, M., & Benfield, C. (2022). *HTTP/2 (RFC 9113)*. RFC Editor.

WHATWG. (2026). *Fetch Standard*.

Further reading

- David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy, HTTP: The Definitive Guide, O'Reilly Media, 2002 — Probably the most directly relevant book for this handout. It systematically examines HTTP requests and responses, methods, status codes, headers, message bodies, connections, caching, authentication, and content negotiation. It is particularly useful for understanding how HTTP messages are constructed and interpreted. Although its coverage of protocol versions is outdated, its treatment of HTTP/1.1 remains valuable for studying the fundamental communication model.
- James F. Kurose and Keith W. Ross, Computer Networking: A Top-Down Approach, 9th ed., Pearson, 2026 — Provides the broader networking foundations needed to understand HTTP as an application-layer protocol. It examines client–server communication, persistent and non-persistent connections, HTTP, DNS, web caching, and the relationship between application protocols and transport services. It is particularly suitable for connecting the handout's explanation of HTTP messages to the underlying network infrastructure.
- Leonard Richardson, Mike Amundsen, and Sam Ruby, RESTful Web APIs, O'Reilly Media, 2013 — An excellent complement for understanding the semantics of HTTP methods and their application to web services. The book examines resources, representations, GET, POST, PUT, PATCH, DELETE, idempotence, status codes, headers, and content negotiation. It is particularly relevant to the handout's distinction between retrieving, creating, modifying, and deleting resources, showing why HTTP methods express different operations rather than simply transporting data.
- Ilya Grigorik, High Performance Browser Networking, O'Reilly Media, 2013 — Focuses on HTTP communication, connection management, latency, caching, and

web performance. It explains how network conditions and protocol behaviour affect the time required to retrieve web resources. It is especially useful for extending the handout's discussion of persistent connections and multiple HTTP requests generated while loading a web page. Some discussions of newer HTTP versions are dated, but the networking principles remain relevant.

- Malcolm McDonald, Web Security for Developers, No Starch Press, 2020 — Provides a practical introduction to web security, encryption, authentication, sessions, permissions, and vulnerabilities affecting HTTP-based applications. It is particularly useful for understanding why HTTPS is necessary, how session management relates to cookies, and why protecting network communication is only one component of securing a web application.

Editorial notes

In producing these materials, I have used generative artificial intelligence tools to support the writing process, particularly to improve the form of the text, reorganize content, refine wording, and speed up certain editorial tasks.

As I work independently on the production of these materials, I try to automate any activities that can reasonably be automated, so that I can devote more time to research, design, and content development.

However, artificial intelligence does not determine the content of this work: the choice of topics, structure, ideas, interpretations, examples, and teaching approach are developed by me. AI is therefore used primarily as a tool to support production and formal revision, while authorship and responsibility for the design and content remain mine.

Unless otherwise indicated, this material is distributed under the Creative Commons Attribution–NonCommercial–ShareAlike 4.0 International license (CC BY-NC-SA 4.0).

You are therefore free to share, redistribute, adapt, and create derivative works based on this material, provided that appropriate credit is given, the material is not used for commercial purposes, and any modified or derivative versions are distributed under the same license.

