

# Metodi HTTP - Nello Specifico

---

## Metodo GET

---

Con GET tipicamente effettuiamo un'operazione di lettura, quindi recuperiamo una risorsa con il supporto delle linee di intestazione per specificare meglio la richiesta.

Quando usiamo GET la **request-line** è quella più significativa. Il **body** di un messaggio di richiesta GET è **vuoto**; si può violare questa indicazione, ma da specifica il corpo dovrebbe essere vuoto, questo perchè se richiediamo delle risorse non abbiamo bisogno di passare nessun corpo al server.

Attraverso il secondo token specifichiamo la risorsa di cui abbiamo bisogno; dopo il nome della risorsa possiamo trovare una stringa che segue la sintassi: `GET /risorsa.html?cognome=biondo&nome=peppe HTTP/1.1`

Il '?' indica le coppie nome-valore che sono i parametri che impieghiamo per caratterizzare meglio la richiesta; ad esempio vogliamo effettuare una richiesta più specifica con dei parametri. Ad esempio condizioniamo la richiesta (in questo caso) con due parametri di nome **cognome e nome**, ed i valori sono **biondo e peppe**.

Come abbiamo detto il metodo GET è sia **safe** che **idempotente** (vedi lezione precedente); può essere assoluto, ovvero quando è usato in maniera assoluta la richiesta deve essere trasferita sul server se presente, o può essere **condizionale**: quando si effettua una richiesta di tipo GET si può dire che il server deve spedire la risorsa al client **solo se sono verificate delle condizioni**;

Quando la richiesta è condizionale, nelle linee di intestazione troviamo qualcosa simile ai campi: `if-Modified-Since`, `if_match`, `if-Non-Match`.

## Metodo HEAD

---

Questo metodo è simile al metodo GET; quì il messaggio di risposta **non contiene corpo**. Quindi la richiesta è simile al GET, ma non avremo un corpo, mentre le intestazioni saranno le stesse.

HEAD viene usato per effettuare **delle verifiche**, come, ad esempio, per verificare l'esistenza di una risorsa.

Anche in questo caso HEAD è sia safe che idempotente.

## Metodo POST

---

Il metodo POST è leggermente più complesso; è un metodo che consente di trasferire dati dal client al server. I dati sono presenti nel **body del messaggio di richiesta**, quindi quando usiamo questo metodo, il messaggio di richiesta deve contenere il corpo.

L'URI che viene specificata dopo il metodo POST è l'uri della risorsa alla quale si vogliono inviare i dati presenti nel corpo. Si sottopongono i dati presenti nel corpo **alla risorsa identificata nell'URI**.

**Esempio di POST:** `POST /dati.jsp HTTP/1.1` + **corpo**.

Siccome non possiamo indicare la risorsa che vogliamo sia aggiornata, POST non è un metodo idempotente.

# Metodo PUT

---

E' un altro metodo usato per effettuare **scritture sul server**, quindi anche in questo caso troviamo un body, ma a differenza di POST questo metodo prevede un'interpretazione dell'URI che segue il metodo;

mentre con POST specifichiamo l'uri della risorsa sul server alla quale vogliamo inviare i dati, con il PUT l'URI che segue il metodo, è l'uri che vogliamo sia creata sul server, con i dati presenti nel body.

Quindi, se esiste già una risorsa sul server con quel nome, essa viene **aggiornata**.

Questo è importante perchè rende il PUT un metodo **non safe**, ma idempotente. Se invochiamo ripetutamente il metodo PUT, andiamo ad aggiornare successivamente quella risorsa, quindi non cambiamo nulla (rendendo PUT idempotente).

# Messaggi di risposta HTTP

---

Il protocollo HTTP deve essere visto come un *linguaggio* da usare poi per lo sviluppo di applicazioni in web. Anche quando usiamo delle librerie faremo riferimento ai metodi visti in precedenza.

La differenza sostanziale con i messaggi di richiesta, sta nella **start-line**, che prende il nome di **status-line**.

## Struttura del messaggio di risposta

---

### Status line

Nella prima linea della status line troviamo 3 token:

```
<http protocol version> <response code> <description>
```

Il primo token ci dice quale protocollo usiamo per la costruzione del messaggio di risposta, il secondo token è un codice numerico che ci dice se l'interazione è andata a buon fine o meno; questo codice è uno dei famosi codici numerici che a volte ci vengono restituiti sulle pagine web, come ad esempio 404 - risorsa non trovata, o 200 - OK. Il terzo token è una stringa che dà significato al codice numerico.

### Linee di risposta header

Dopo la linea di intestazione, abbiamo le status line che contengono le coppie **chiave:valore** del tipo:

- Content-Type: <tipo del contenuto>
- Content-Length: <lunghezza del body>
- Server: <implementazione del server>

### Linea vuota

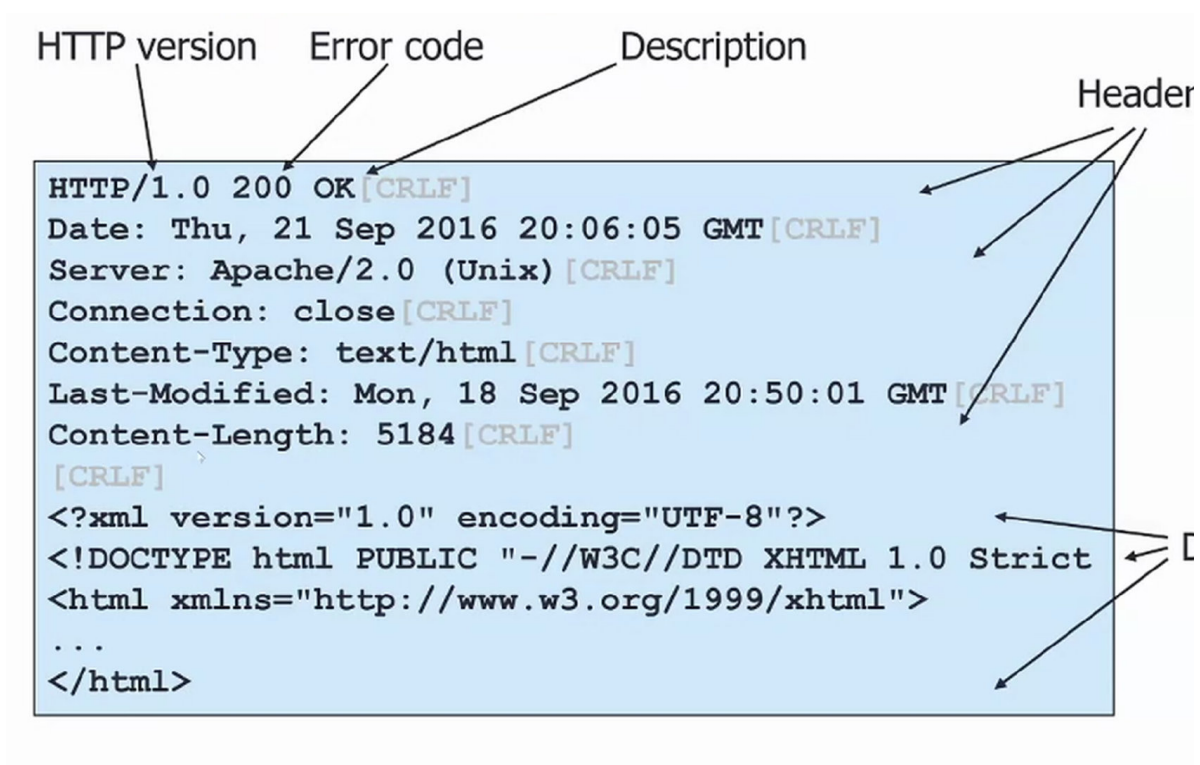
Questa linea è lasciata vuota

### Data

Questa sezione contiene il corpo (body) del messaggio.

## Esempio di messaggio di risposta

---



## Test sul campo - Telnet

Usiamo telnet per inviare delle stringhe ad un server, che li interpreterà come se fossero comandi. Telnet è stato sostituito da SSH, che usa una connessione protetta.

Telnet prevede l'uso della porta 23, stessa porta usata per la chat multi-utente durante l'esercitazione 6.

Per prima cosa troviamo l'indirizzo ip del server con: `dig www.unisannio.it`

```
giulianoranauro — -bash — 80x24
MBP-di-Giuliano:~ giulianoranauro$ dig www.unisannio.it

;; <<>> DiG 9.10.6 <<>> www.unisannio.it
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6668
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:;, udp: 1220
;; QUESTION SECTION:
;www.unisannio.it.                IN      A

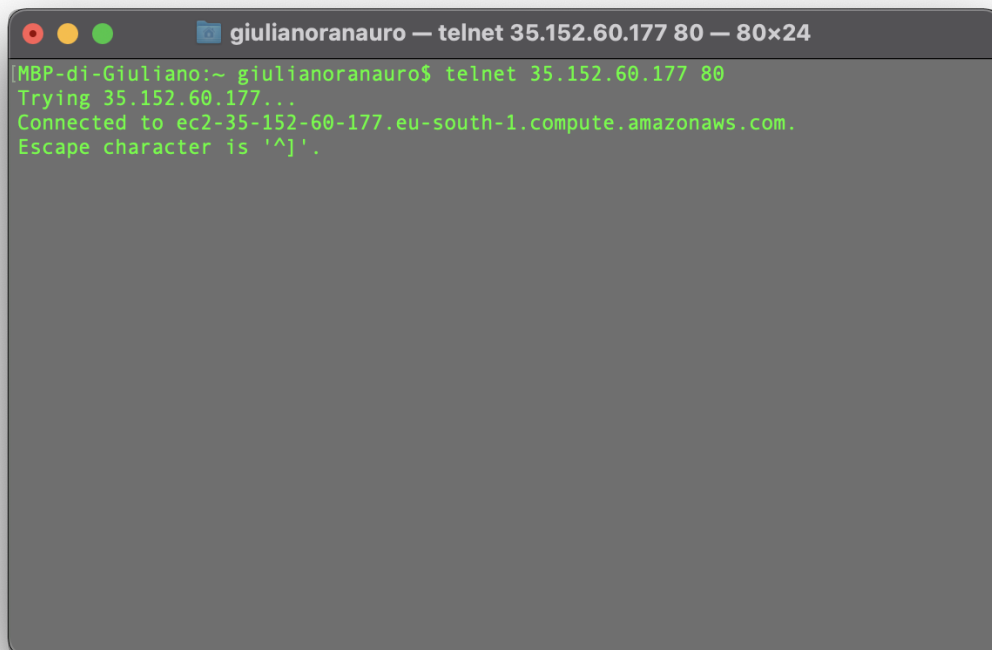
;; ANSWER SECTION:
www.unisannio.it.                166981  IN      A      35.152.60.177

;; Query time: 21 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Thu Feb 03 15:06:50 CET 2022
;; MSG SIZE rcvd: 61

MBP-di-Giuliano:~ giulianoranauro$
```

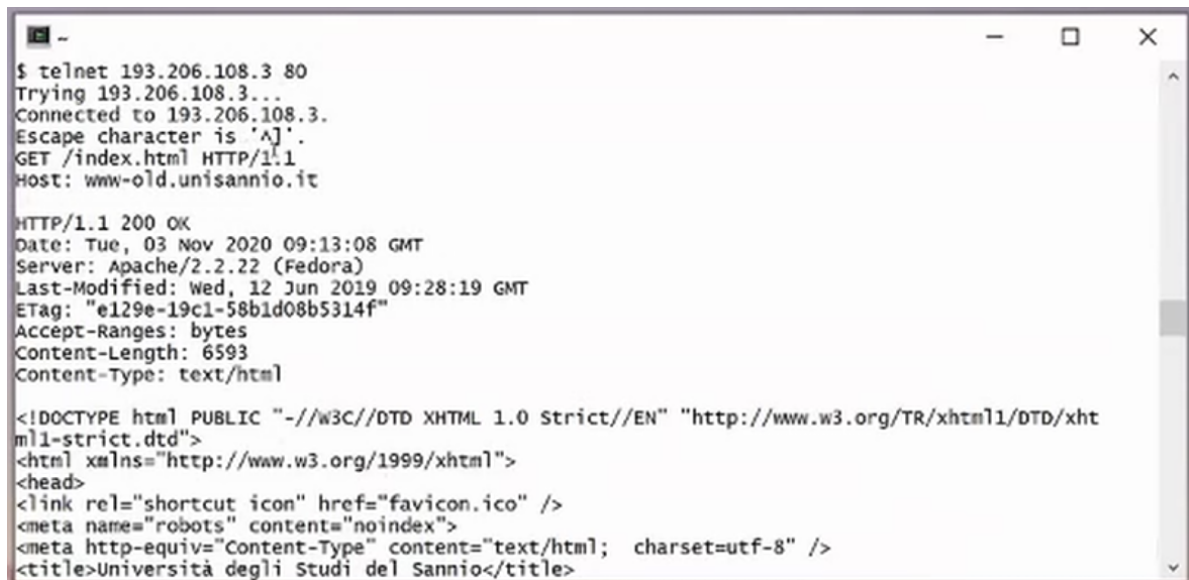
Successivamente digitiamo `telnet 35.152.60.177 80`; se non abbiamo installato telnet, digitiamo `brew install telnet` su MacOS, per eseguire questo comando avremo bisogno di avere installato il gestore di pacchetto **brew**.

Otteniamo quindi:

A terminal window titled "giulianoranauro — telnet 35.152.60.177 80 — 80x24". The prompt is "MBP-di-Giuliano:~ giulianoranauro\$". The user enters "telnet 35.152.60.177 80". The output shows the connection process: "Trying 35.152.60.177...", "Connected to ec2-35-152-60-177.eu-south-1.compute.amazonaws.com.", and "Escape character is '^['".

```
MBP-di-Giuliano:~ giulianoranauro$ telnet 35.152.60.177 80
Trying 35.152.60.177...
Connected to ec2-35-152-60-177.eu-south-1.compute.amazonaws.com.
Escape character is '^['.
```

Per poter proseguire dobbiamo creare la nostra richiesta:

A terminal window showing a telnet connection to 193.206.108.3 80. The user enters "telnet 193.206.108.3 80". The output shows the connection process: "Trying 193.206.108.3...", "Connected to 193.206.108.3.", and "Escape character is '^['". The user then enters "GET /index.html HTTP/1.1". The output shows the HTTP response: "Host: www-old.unisannio.it", "HTTP/1.1 200 OK", "Date: Tue, 03 Nov 2020 09:13:08 GMT", "Server: Apache/2.2.22 (Fedora)", "Last-Modified: Wed, 12 Jun 2019 09:28:19 GMT", "ETag: \"e129e-19c1-58b1d08b5314f\"", "Accept-Ranges: bytes", "Content-Length: 6593", "Content-Type: text/html". The response body is an HTML document with a title "Università degli Studi del Sannio".

```
$ telnet 193.206.108.3 80
Trying 193.206.108.3...
Connected to 193.206.108.3.
Escape character is '^['.
GET /index.html HTTP/1.1
Host: www-old.unisannio.it

HTTP/1.1 200 OK
Date: Tue, 03 Nov 2020 09:13:08 GMT
Server: Apache/2.2.22 (Fedora)
Last-Modified: Wed, 12 Jun 2019 09:28:19 GMT
ETag: "e129e-19c1-58b1d08b5314f"
Accept-Ranges: bytes
Content-Length: 6593
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="shortcut icon" href="favicon.ico" />
<meta name="robots" content="noindex">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Università degli Studi del Sannio</title>
```

Non ho potuto effettuare la richiesta perchè evidentemente la struttura del sito è cambiata

Siccome il sito unisannio non funziona tanto bene, e noi ci siamo abituati, proviamo ad effettuare gli stessi passaggi sul [mio sito](#) hostato su GitHub:

```

MBP-di-Giuliano:~ giulianoranauro$ dig www.follen99.github.io
; <<>> DiG 9.10.6 <<>> www.follen99.github.io
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26177
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags: udp: 1220
;; QUESTION SECTION:
;www.follen99.github.io.                IN      A

;; ANSWER SECTION:
www.follen99.github.io. 3600    IN      A      185.199.109.153
www.follen99.github.io. 3600    IN      A      185.199.108.153
www.follen99.github.io. 3600    IN      A      185.199.111.153
www.follen99.github.io. 3600    IN      A      185.199.110.153

;; Query time: 30 msec
;; SERVER: 192.168.1.1#53(192.168.1.1)
;; WHEN: Thu Feb 03 15:28:23 CET 2022
;; MSG SIZE rcvd: 115

MBP-di-Giuliano:~ giulianoranauro$ telnet 185.199.109.153 80
Trying 185.199.109.153...
Connected to cdn-185-199-109-153.github.com.
Escape character is '^['.
GET /ranauro.giuliano HTTP/1.1
Host: www.follen99.github.io

HTTP/1.1 404 Not Found
Server: GitHub.com
Content-Type: text/html; charset=utf-8
permissions-policy: interest-cohort=()
ETag: "5f765aa7-239b"
Content-Security-Policy: default-src 'none'; style-src 'unsafe-inline'; img-src data;; connect-src 'self'
X-GitHub-Request-Id: 6582:E86D:10DC9A7:1140794:61FBE6AE
Content-Length: 9115
Accept-Ranges: bytes
Date: Thu, 03 Feb 2022 14:29:02 GMT
Via: 1.1 varnish
Age: 0
Connection: keep-alive
X-Served-By: cache-mxp6973-MXP
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1643898542.997184,V50,VE92
Vary: Accept-Encoding
X-Fastly-Request-ID: 44af8a303f45ec453202d9053157450ef5b219c4

<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <meta http-equiv="Content-Security-Policy" content="default-src 'none'; styl
e-src 'unsafe-inline'; img-src data;; connect-src 'self'">

```

Per prima cosa ricaviamo l'indirizzo IP del server che hosta la pagina web con `dig` `www.fo11en99.github.io`; una volta ottenuto l'indirizzo IP ci colleghiamo con telnet usando il comando: `telnet 185.199.109.153 80`.

Una volta collegati dobbiamo costruire la richiesta HTTP GET con

```
GET /ranauro.giuliano HTTP/1.1
Host:www.follen99.github.io
```

Il server ci risponderà con il file **.html** della pagina.

# Autenticazione

---

Vediamo cosa succede quando una richiesta di tipo GET non può essere immediatamente trasmessa perchè **protetta**; in questo caso il server chiede che il client si autentichi per essere autorizzato alla risorsa.

Il tentativo di accesso alla risorsa mediante una GET **fallisce**, ed il messaggio di risposta dal server non conterrà il codice 200, e nemmeno la risorsa richiesta. Troveremo invece un messaggio di risposta di tipo **401 - authorization required**.

Questo significa che il client deve autenticarsi prima di poter accedere alla risorsa. Quando il server risponde con **401** (errore dovuto ad una carenza del messaggio di richiesta), chiede anche (attraverso il messaggio di risposta) che il client **faccia autenticare l'utente**.

Il server esplicita questa richiesta attraverso il campo chiamato **WWW-Authenticate**, seguito da uno schema: `WWW-Authenticate: <scheme>`.

Quando il client riceve questa risposta contenente questa linea di intestazione, il browser propone al client un **form di autenticazione** (il client è programmato preventivamente per fornire il form), e le credenziali che l'utente inserisce saranno successivamente inviate al server, attraverso la linea di intestazione presente nel messaggio del tipo: `Authorization: <credentials>`. Le credenziali sono codificate in **base64**, che consente di impiegare 64 caratteri che non interferiscono con caratteri speciali. Queste credenziali **vengono spedite in chiaro**!

Il server invia anche un'altra informazione che dice per quali **contenuti l'autenticazione fornita vale**. Per tutte le risorse che appartengono al sottoinsieme il client non chiederà all'utente di autenticarsi nuovamente.

## Gestione della sessione

---

La sessione è un'intervallo temporale durante il quale si sviluppa il dialogo tra client e server.

🚩 1:51

L'autenticazione effettuata nella prima interazione, può essere usata nelle interazione successiva? Ovviamente sì. I client tendono a memorizzare le informazioni di accesso per evitare che l'utente debba inserire le sue informazioni continuamente; inoltre queste informazioni non vengono memorizzate sul server.

Questo anche perchè l'HTTP è **stateless**, quindi non solo le informazioni di accesso, ma anche altro tipo di informazioni di sessione non vengono memorizzate lato server.

Quello che il client fa (senza che l'utente lo noti) è mantenere le informazioni di sessione (prima parlavamo di credenziali di accesso) in modo che il client non debba continuamente inserire dati.

## Stato della sessione: Cookie

---

I cookie sono informazioni che vengono inviate dal server al client, perchè poi il client le rispedisca al server nel momento in cui è necessario. Servono per far sì che l'utente non inserisca dati che **ha già inserito in precedenza**. Serve quindi al server per capire da chi proviene una determinata richiesta.

Il protocollo HTTP per supportare i cookie prevede due campi di intestazione:



## Set-Cookie

Viene usato nei messaggi di risposta con l'obiettivo di costruire un legame con il messaggio di richiesta successivo, quindi tra due interazioni.

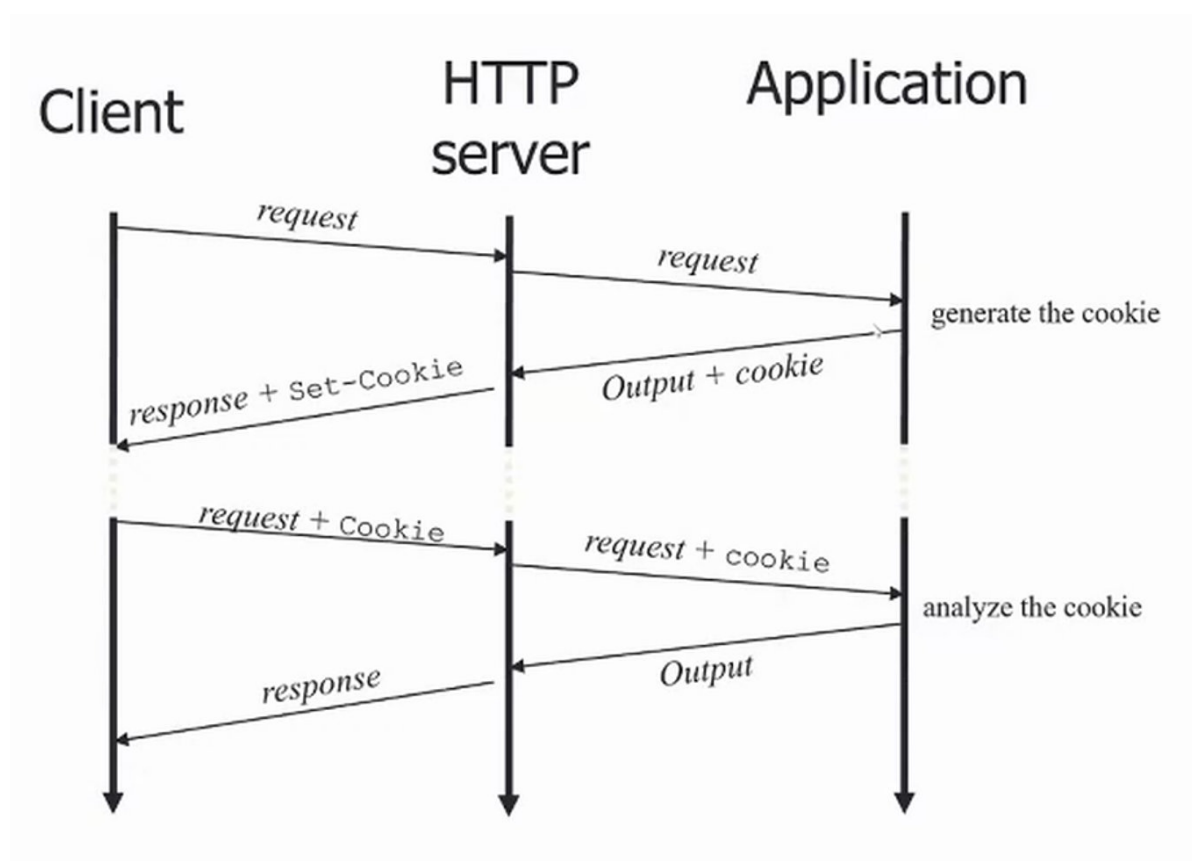
Con set-cookie il server spedisce al client il cookie con un valore che può assumere la forma:

Set-Cookie: name=value; Domain=www.domain.com; Path=/protected; Expires=Wed, 22-Mar-2020 21:10:01 GMT; Secure; HttpOnly

Il server ci dice il nome del cookie, che è costituito da una stringa; il Dominio; il percorso (il cookie sarà utile non per tutte le risorse ma per alcune risorse); scadenza del cookie, ovvero per quanto tempo il client deve mantenere l'informazione; "secure", ovvero ci dice se il cookie deve essere usato solo in interazioni sicure, ovvero una comunicazione non in chiaro ma cifrato; "HttpOnly" ci dice che il cookie deve essere spedito solo con HTTP, quindi senza il supporto di script.

## Cookie

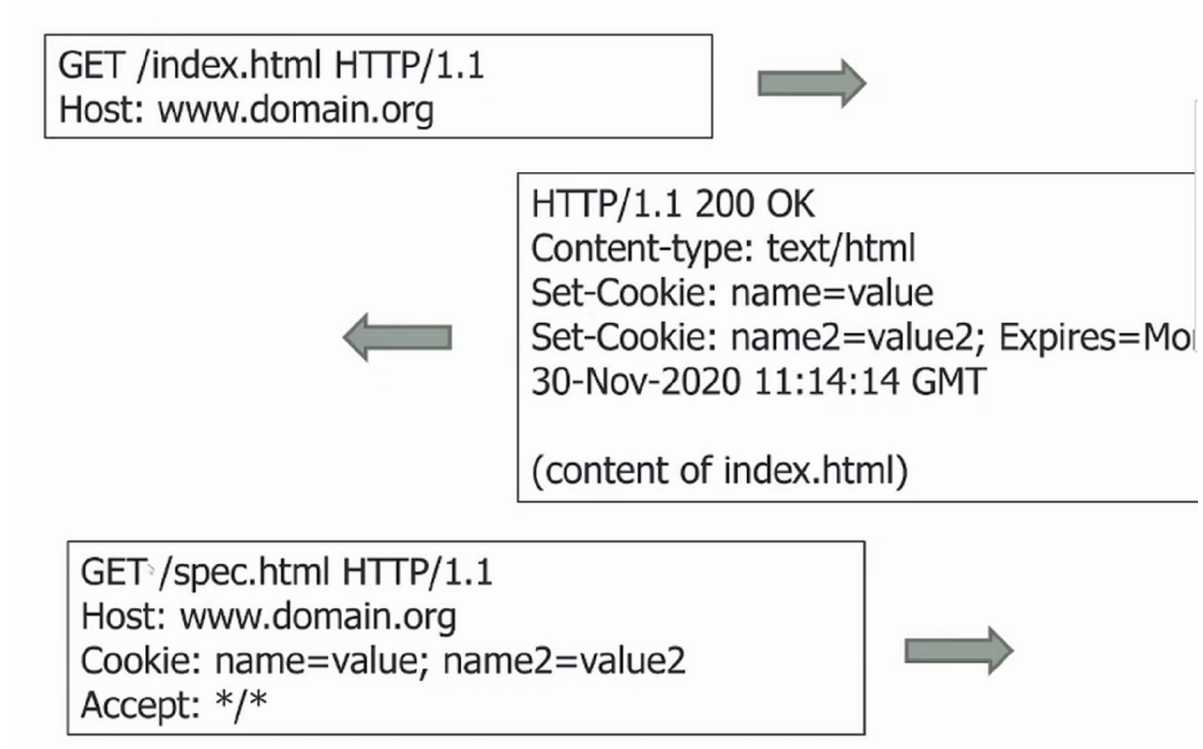
Il client che riceve il cookie, spedirà nelle interazioni successive con un altro campo.



Abbiamo un messaggio di richiesta da parte del client che arriva al server, la parte applicativa del server genera un cookie che viene associato nella linea di impostazione della risposta, ed il client memorizza il cookie.

Nelle interazioni successive il client utilizzerà il campo cookie con il valore ricevuto in precedenza ed il server lo utilizzerà per le sue cose.

## Message flow con i cookie



Come possiamo vedere, abbiamo prima la richiesta del client, e nella risposta del server, tra le linee di intestazione del messaggio, sono presenti proprio dei campi **set-cookie** con delle informazioni dei cookie.

Quando il client invia un nuovo messaggio, nell'intestazione saranno presenti i cookie ricevuti precedentemente.