

# HTTP

# HYPERTEXT TRANSFER PROTOCOL

# Storia HTTP

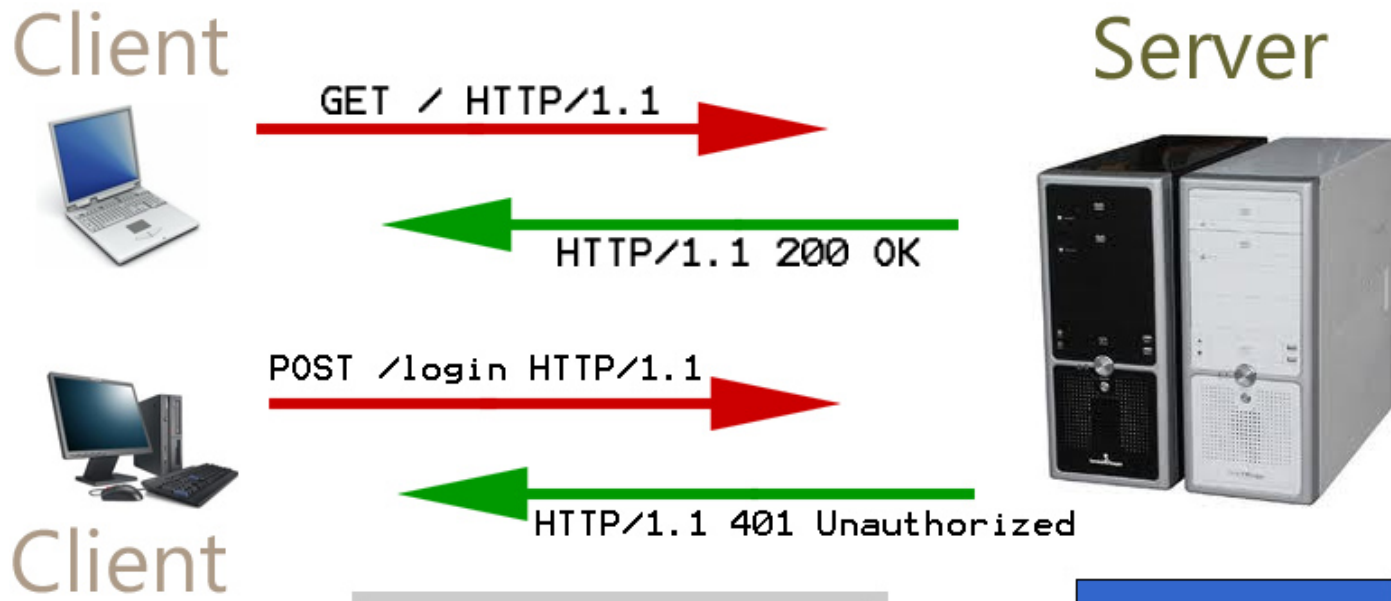
- HTTP originale (anche noto come HTTP/0.9)
  - fine anni '80
  - uso quasi solo sperimentale
- HTTP/1.0
  - **1991** proposto nel 1996 standard
  - RFC 1945
- HTTP/1.1
  - RFC 2068 poi 2616 poi 7230-7237
  - nel **1997**
- HTTP/2
  - RFC 7540 maggio **2015**
  - migliore efficienza di trasmissione
- HTTP/3
  - RFC 9114 giugno **2022**
  - **QUIC/UDP**

# Caratteristiche HTTP

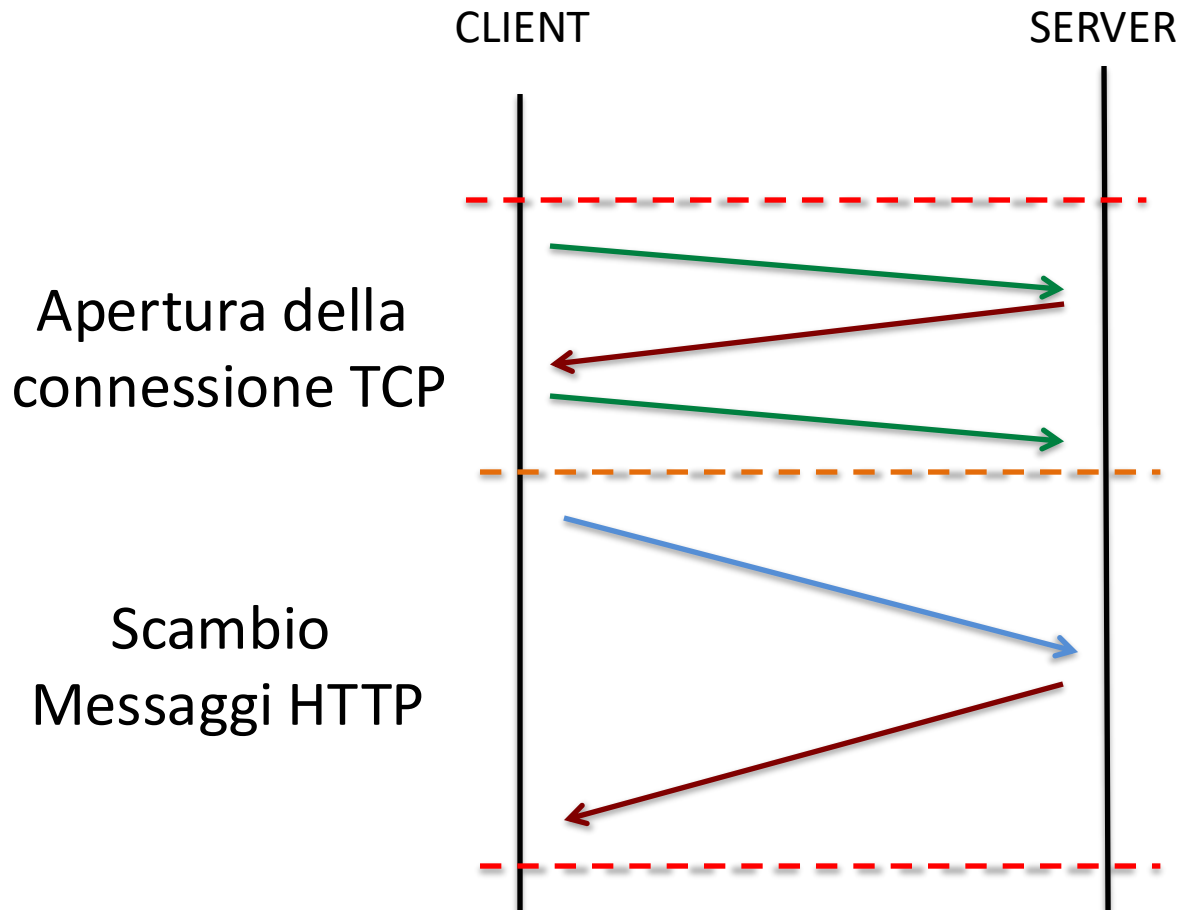
- Application level protocol
  - Lo usano gli applicativi per comunicare
- Scambio di messaggi di richiesta e risposta
- Stateless (senza stato o senza memoria)
  - Ogni richiesta HTTP è indipendente da quelle precedenti e si conclude al momento della chiusura della connessione
- Media independent
  - Può trasportare ogni tipo di dato

# Protocollo

- Client
  - Messaggi di Richiesta – **HTTP Request**
- Server
  - Messaggi di Risposta – **HTTP Response**



# Transazione HTTP



HTTP è l'insieme di regole che definisce il formato ed il contenuto della conversazione fra web client e server

# MESSAGGI HTTP

# Richiesta GET HTTP

Linea della Richiesta



Headers



Riga vuota



```
GET /pw HTTP/1.1 [CRLF]
Host: ppl.eln.uniroma2.it [CRLF]
Connection: keep-alive [CRLF]
Pragma: no-cache [CRLF]
Cache-Control: no-cache [CRLF]
Accept-Encoding: gzip, deflate, sdch [CRLF]
[CRLF]
```

# Risposta HTTP

Linea di Stato



```
HTTP/1.1 200 OK [CRLF]
```

Headers



```
Date: Fri, 19 May 2017 07:11:37 GMT [CRLF]  
Server: Apache/2.2.14 (Ubuntu) [CRLF]  
X-Powered-By: PHP/5.3.10-1~lucid+2uwsgi2 [CRLF]  
Vary: Accept-Encoding [CRLF]  
Content-Encoding: gzip [CRLF]  
Content-Length: 3722 [CRLF]  
Content-Type: text/html; charset=UTF-8 [CRLF]
```

Riga vuota



```
[CRLF]
```

Body



```
<!DOCTYPE html>  
<html lang="en-US">  
<head>  
...
```



# Richiesta POST HTTP

Linea della Richiesta →

Headers →

Riga vuota →

Body →

```
POST /pw/esf/selfpr.php HTTP/1.1 [CRLF]
Host: ppl.eln.uniroma2.it [CRLF]
Content-Length: 53 [CRLF]
Pragma: no-cache [CRLF]
Cache-Control: no-cache [CRLF]
Origin: http://ppl.eln.uniroma2.it [CRLF]
Upgrade-Insecure-Requests: 1 [CRLF]
Content-Type: application/x-www-form-
urlencoded [CRLF]
Referer:
http://ppl.eln.uniroma2.it/pw/esf/selfpr.php [
[CRLF]
[CRLF]
username=Pierpaolo&email=pierpaolo.loreti%40g
mail.coma
```

# Risposta HTTP

Linea di Stato

→ **HTTP/1.1 201 OK**[CRLF]

Headers

Date: Fri, 19 May 2017 06:54:31 GMT [CRLF]  
Server: Apache/2.2.14 (Ubuntu) [CRLF]  
X-Powered-By: PHP/5.3.10-  
1~lucid+2uwsgi2 [CRLF]  
Vary: Accept-Encoding [CRLF]  
Content-Encoding: gzip [CRLF]  
Content-Length: 129 [CRLF]  
Content-Type: text/html [CRLF]  
[CRLF]

Riga vuota

<pre>

Body

```
$_GET array(0) {}

$_POST array(2) { ["username"]=>string(9)
"Pierpaolo" ["email"]=>
string(26) "pierpaolo.loreti@gmail.com"}
```

# Formato Messaggi

- **Start-line**
  - Request-line o Status-line
- **Header fields**
  - uno per riga seguiti da CRLF
  - quattro tipi: general, request, response, entity
- **Riga vuota**
  - per indicare la fine dell'header
  - contiene solo CRLF
- **Body**
  - opzionale

# METODI

# HTTP metodi

- GET
- HEAD
- POST

## HTTP 1.1

- PUT
- PATCH
- DELETE
- OPTIONS

# Metodi delle richieste HTTP

- GET
  - Recupera le informazioni specificate dalla URI di richiesta.
  - La risposta può essere memorizzata in cache
- HEAD
  - la risposta deve essere identica a quelle che restituirebbe il GET
  - il server non deve restituire il corpo del messaggio nella risposta
  - La risposta può essere memorizzata in cache
- POST
  - È usato per richiedere che il server accetti l'entità inclusa nella richiesta
  - il modo in cui la funzione è eseguita dipende dal server
  - I codici di risposta variano a seconda dell'azione (200, 204, 201)
  - In generale le risposte non sono soggette a cache

# Metodi delle richieste HTTP

- PUT

- Richiede che l'entità inclusa sia memorizzata sotto l'URI di richiesta fornita.
- se esistente
  - l'entità inclusa dovrebbe essere considerata come una versione modificata di quella esistente
  - Risposte 200 o 204
- Se non esistente
  - il server può creare la risorsa con quella URI
  - Risposta 201
- Nella POST la URI identifica la risorsa che gestirà l'entità inclusa nella PUT la URI nella richiesta PUT identifica l'entità inclusa

- PATCH

- Richiede di applicare delle modifiche all'entità a cui la URI di riferisce
- Una richiesta PATCH è considerata un insieme di istruzioni su come modificare una risorsa. La PUT è una rappresentazione completa di una risorsa.
- Risposte 200 o 204

# Metodi delle richieste HTTP

- DELETE

- Richiede la cancellazione dal server di origine della risorsa identificata dalla URI di richiesta.
- La risposta di successo dovrebbe essere
  - 200 (OK) se include un'entità che descrive lo stato
  - 202 (Accepted) se l'azione non è stata ancora attivata
  - 204 (No Content) se l'azione è stata attivata, ma non c'è un'entità inclusa nella risposta
- Le risposte a DELETE non sono soggette a cache

- OPTIONS

- Rappresenta una richiesta di informazioni riguardanti le opzioni di comunicazione disponibili per una URI
- Le risposte non sono soggette a cache
- Se la URI è un asterisco ("\*") la richiesta si intende applicata al server



# Risposte HTTP

- **1xx Informational:** indicano che la richiesta è stata ricevuta e il processo è in corso.
- **2xx Success:** confermano che la richiesta è stata ricevuta, capita ed elaborata con successo.
- **3xx Redirection:** segnalano che il client deve effettuare ulteriori operazioni per completare la richiesta.
- **4xx Client Error:** indicano che la richiesta contiene errori da parte del client (ad esempio, una risorsa inesistente).
- **5xx - Server Error:** segnalano che il server ha incontrato un problema interno durante l'elaborazione della richiesta.

# Response Status Code

- Status-codes 1xx - Informational
  - Riservato per usi futuri
- Status-codes 2xx - Success
  - L'azione è stata ricevuta con successo, capita e accettata
    - 200 OK
    - 201 POST command successful
    - 202 Request accepted
    - 203 GET or HEAD request fulfilled
    - 204 No content

# Response Status Code

- Status-codes 3xx - Redirection

- Per completare la richiesta è necessaria un'altra azione
  - 300 Resource found at multiple locations
  - **301 Resource moved permanently**
  - **302 Resource moved temporarily**
  - 304 Resource has not modified (since date)

- Status-codes 4xx - Client error

- La richiesta contiene una sintassi errata o non può essere servita
  - 400 Bad request from client
  - **401 Unauthorized request**
  - 402 Payment required for request
  - 403 Resource access forbidden
  - **404 Resource not found**
  - 405 Method not allowed for resource
  - 406 Resource type not acceptable

# Response Status Code

- Status-codes 5xx - Server error
  - Il server non riesce a servire una richiesta apparentemente valida
    - 500 Internal server error
    - 501 Method not implemented
    - 502 Bad gateway or server overload
    - 503 Service unavailable / gateway timeout
    - 504 Secondary gateway / server timeout

# Header fields

- Righe testuali free-format che specificano caratteristiche
  - generali della trasmissione (**header generali**)
  - dell'entità trasmessa (**header di entità**)
  - della richiesta effettuata (**header di richiesta**)
  - della risposta generata (**header di risposta**)

# Header generali

**Si applicano sia a richieste che a risposte.**

Contengono metadati generali sulla comunicazione.

Esempi:

- **Date:** data e ora della comunicazione.
- **Connection:** mantiene o chiude la connessione (keep-alive, close).

# Header di Entità

**Descrivono il corpo (body) del messaggio HTTP.**

Informano sul tipo di contenuto o sulla sua lunghezza.

Esempi:

- **Content-Type:** tipo di contenuto (es. application/json, text/html).
- **Content-Length:** lunghezza in byte del corpo.
- **Content-Encoding:** compressione usata (gzip, deflate).
- **Last-Modified:** ultima modifica

# Header di Richiesta

**Usati dal client** per fornire dettagli sulla richiesta.

Possono indicare la lingua preferita, il tipo di contenuto accettato o informazioni sul client.

Esempi:

- **User-Agent**: identifica il client (browser, app, ecc.).
- **Accept**: tipi di contenuto accettati (text/html, application/json).
- **Authorization**: per l'invio di credenziali di accesso.
- **Accept-Language**: preferenze linguistiche dell'utente.



# Esempi Header di Richiesta

- From
  - From: pierpaolo.loreti@uniroma2.it
- User-Agent
  - User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_9\_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36
- If-Modified-Since
  - If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
- Referer
  - Referer: http://www.uniroma2.it/
- Authorization
  - Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l

# Header di Risposta

**Usati dal server** per fornire informazioni sulla risposta.

Possono specificare dati sul server, la tecnologia usata, o la lingua della risposta.

Esempi:

- **Server:** indica il software del server (es. Apache, nginx).
- **Content-Language:** lingua della risposta.
- **WWW-Authenticate:** chiede autenticazione (es. Basic realm="Area Riservata").
- **Location:** usato per le redirect

# Esempi Header di Risposta

- **Server**
  - Server: Apache 1.3.20
- **Location**
  - Location: <http://web.uniroma2.it/>
- **WWW-Authenticate**
  - WWW-Authenticate: Basic realm="Area Privata"

**INVIARE DATI**

# Metodo GET

- I dati vengono passati **nella URL**, come **query string**.

```
GET /search?q=telemedicina HTTP/1.1  
Host: www.portalesanitario.com
```



Vantaggi: semplice e visibile nella barra del browser.



Svantaggi: **limite di lunghezza** della URL

# Metodo POST

- I dati vengono inviati nel **corpo della richiesta** (body), non nella URL.

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

username=admin&password=1234
```



Più sicuro (dati non visibili in URL), adatto per **form**, login, file, ecc.

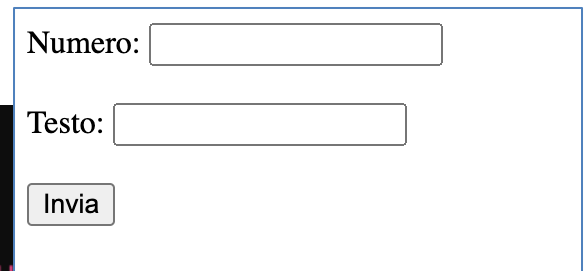


Non memorizzabile in cache, meno “trasparente” per l’utente.

# Forme di invio dati lato client

- Query string (con GET)
- Form HTML (con POST)
- Fetch/AJAX in JavaScript (API REST)

```
<form action="process.php" method="post">  
  <label for="numero">Numero:</label>  
  <input type="number" id="numero" name="numero" required>  
  <br><br>  
  <label for="testo">Testo:</label>  
  <input type="text" id="testo" name="testo" required>  
  <br><br>  
  <button type="submit">Invia</button>  
</form>
```



Numero:

Testo: