

Programmazione per il Web

Alessio Marini, 2122855

Appunti presi durante il corso di **Programmazione per il Web** nell'anno **2025/2026** del professore Mattia Samory.

Gli appunti li scrivo principalmente per rendere il corso più comprensibile **a me** e anche per imparare il linguaggio Typst. Se li usate per studiare verificate sempre le informazioni 🙏 .

Contatti:

📧 [alem1105](#)

✉ marini.2122855@studenti.uniroma1.it

September 27, 2025

Indice

1. HTTP (HyperText Transfer Protocol)	3
1.1. Metodi HTTP	4
1.1.1. PUT	4
1.1.2. GET	5
1.1.3. POST	5
1.1.4. DELETE	5
1.1.5. Altri metodi	5
1.2. Codici di Stato	5
1.2.1. 2xx Successful	6
1.2.2. 3xx Redirection	6
1.2.3. 4xx Client Error	6
1.2.4. 5xx Server Error	6
2. API Pubbliche	7

1. HTTP (HyperText Transfer Protocol)

E' un protocollo a livello applicazione nello stack TCP / IP. La sua variante più sicura si chiama **HTTPS**.

Si basa sul concetto di **client e server**, il client è chi richiede dei servizi o risorse mentre il server è chi le offre.

User Agent (UA)


E' una qualsiasi applicazione del client che avvia una richiesta, ad esempio il browser web, un'app ecc...

Origin Server (O)

Un programma che può originare risposte autorevoli per una data risorsa, ad esempio un sito web.

Esempio di Richiesta

```
1 GET /hello.txt HTTP/1.1
2 User-Agent: curl/7.64.1
3 Host: [www.example.com](https://www.example.com)
4 Accept-Language: en, it
```

 HTTP

- La prima è la **linea di richiesta** dove viene indicato il metodo HTTP, in questo caso GET, l'URI della risorsa e la versione del protocollo.
- Vari campi di intestazione, ad esempio **User-Agent**.
- Corpo del messaggio opzionale.

Esempio di Risposta

```
1 HTTP/1.1 200 OK
2 Date: Mon, 27 Jul 2009 12:28:53 GMT
3 Server: Apache
4 Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
5 ETag: "34aa387-d-1568eb00"
6 Accept-Ranges: bytes
7 Content-Length: 51
8 Vary: Accept-Encoding
9 Content-Type: text/plain
10 Hello World! My content includes a trailing CRLF.
```

 HTTP

- In questo caso abbiamo un codice che indicato lo stato della richiesta.
- Contenuto, opzionale.

Ci sono altri due elementi che possiamo trovare in una connessione HTTP:

Intermediari

Sono altri nodi presenti tra il client e il server, ad esempio dei proxy.

Cache

E' una archivio di vecchi messaggi di risposta. Quindi se ad esempio un nodo che si trova in mezzo ad una comunicazione già conosce la risposta ad una richiesta può subito inviarla.

Per fare in modo che i client memorizzino le risposte il server deve inviare delle risposte con l'header **cacheable**.

Dispositivi come i proxy possono memorizzare risposte mentre i **tunnel** no.

1.1. Metodi HTTP

I metodi che possiamo utilizzare con il protocollo sono:

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

I metodi hanno delle proprietà:

- **SAFE**: Un metodo che non ha effetti collaterali sulla risorsa, agisce in sola lettura. Può comunque cambiare lo stato dei server ad esempio creando nuovi file (log). GET, HEAD, OPTIONS e TRACE sono metodi SAFE.
- **IDEMPOTENT**: Richieste identiche multiple con quel metodo hanno lo stesso effetto di una sola richiesta. Ad esempio PUT e DELETE sulle stesse risorse sono idempotenti.
- **CACHEABLE**: Sono i metodi che permettono ad una cache di memorizzare una risposta. GET, HEAD e POST sono CACHEABLE (ma non sempre).


1.1.1. PUT

Serve a creare una nuova risorsa nel server, va specificata nella richiesta. Se questa risorsa già esiste allora la sovrascrive.

Come detto prima è IDEMPOTENTE quindi qualsiasi PUT identica e successiva ad un'altra non modifica la risorsa.

Non è sicuro e non è nemmeno salvabile in cache.


```
1 PUT /course-descriptions/web-and-software-architecture
```

 HTTP

1.1.2. GET

Richiede una risorsa dal server.

```
1 GET /course-descriptions/web-and-software-architecture
```


 HTTP

E' safe dato che non modifica la risorsa, cacheable quindi non serve che il server la rispedisca ad ogni richiesta e questa memorizzazione può avvenire sotto determinate condizioni come ad esempio dei timer. Non è idempotente, quindi richieste successive anche se identiche modificano la risorsa.

1.1.3. POST

Serve ad inviare dati al server o ad aggiornare quelli già presenti.

```
1 POST /announcements/  
2 POST /announcements/{id}/comments/  
3 POST /users/{id}/email
```


 HTTP

La risposta può essere memorizzata in cache, ma non è sicuro nè idempotente.

1.1.4. DELETE

Si invia al server una richiesta per cancellare una determinata risorsa.

```
1 DELETE /courses/web-and-software-architecture
```

 HTTP

Non è safe ma è idempotente.


1.1.5. Altri metodi

- HEAD: Funziona come il GET ma non trasferisce il contenuto della richiesta, soltanto gli header.
- CONNECT: Stabilisce un tunnel verso il server indicato dalla risorsa target.
- OPTIONS: Descrive le opzioni di comunicazione per la risorsa target.
- TRACE: Esegue un test di loop-back del messaggio in lungo tutto il percorso verso la risorsa target.

1.2. Codici di Stato

Negli esempi precedenti abbiamo visto una risposta che iniziava con:

```
1 HTTP/1.1 200 OK
```

 HTTP

Il codice serve a descrivere il risultato della richiesta. Tramite questo possiamo capire, ad esempio:

- Se la richiesta ha avuto successo
- Se ci sono contenuti allegati

I codici sono formati da 3 cifre nell'intervallo 100-599, la prima cifra indica la categoria generale.

- **1xx** (Informational): La richiesta è stata ricevuta ed è in esecuzione.

- **2xx** (Successful): La richiesta è stata ricevuta, compresa ed eseguita con successo.
- **3xx** (Redirection): Sono necessarie ulteriori azioni per completare la richiesta.
- **4xx** (Client error): La richiesta contiene sintassi errata oppure non può essere soddisfatta.
- **5xx** (Server error): Il server non è riuscito a soddisfare una richiesta apparentemente valida.

1.2.1. 2xx Successful

- **200 OK** : In una richiesta GET la risposta conterrà la risorsa richiesta. In una POST la risposta conterrà qualcosa che descrive il risultato dell'azione.
- **201 Created** : La richiesta è stata soddisfatta.
- **204 No Content** : Il server ha elaborato con successo la richiesta ma non sta restituendo dati.

1.2.2. 3xx Redirection

- **301 Moved Permanently** : Questa richiesta ma anche le successive dovrebbe essere reindirizzato all'URI fornito.
- **302 Found** : Visita un'altra URL.

1.2.3. 4xx Client Error

- **400 Bad Request** : Errore del client
- **401 Unauthorized** : Serve un'autenticazione
- **403 Forbidden** : La richiesta è stata compresa dal server e quindi è valida ma l'azione richiesta non è permessa.
- **404 Not Found** : La risorsa non è stata trovata.
- **405 Method not Allowed** : Il metodo richiesto non è supportato.

1.2.4. 5xx Server Error

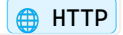
- **500 Internal Server Error** : Errore del server
- **501 Not Implemented** : Il metodo richiesto non è riconosciuto o il server non è in grado di soddisfare la richiesta.
- **502 Bad Gateway** : Un gateway o un proxy hanno ricevuto una risposta non valida dal server.
- **503 Service Unavailable** : Server sovraccarico o spento.
- **504 Gateway Timeout** : Il server non ha ricevuto una risposta in tempo dal server a monte.

2. API Pubbliche

Ci permettono di capire come inviare delle richieste ad un server.

Tramite il comando `curl` possiamo effettuare richieste tramite il terminale. Se non indichiamo un metodo di default verrà utilizzato il GET.

```
1 curl https://swapi.dev/api/people/1/
```



Se vogliamo ad esempio effettuare un POST:

```
1 curl -X POST -H "Content-Type:  
2 application/json" -d '{"title": "Test"}'  
3 https://jsonplaceholder.typicode.com/posts
```

