Message-oriented communication

Web e messaggi HTTP

Architettura del Web

Il Web supporta l'interazione tra client e server via HTTP.

- Il **client** è realizzato da un “Browser” o “User-Agent”
- Il **server** è realizzato da un “Web Server” o “HTTP Server”

Browser

Il browser è l’applicazione per il Web sul lato del client .
Un web browser (detto User-Agent) è un programma che consente la navigazione nel Web da parte di un utente .
La funzione primaria di un browser è quella di interpretare il codice con cui sono espresse le informazioni (pagine web) e visualizzarlo (operazione di rendering) in forma di ipertesto.

I browser moderni hanno funzioni più avanzate per trattati altri tipi di dati (ad esempio Multimedialità, RSS, XML, JSON ...) e ospitare ed eseguire altri applicazioni (ad esempio JavaScript).

L'operazione di rendering dipende dai dispositivi utilizzati, anche "non visuali", per esempio per supportare utenti non vedenti (ad esempio sintesi vocale, alfabeto Braille).

Web Page e Web Server

Una **pagina web** (web page, o anche documento) è concettualmente un file, cioè una sequenza di dati (in formato digitale, cioè una sequenza di byte) che definisce una risorsa, che è identificato da una URL (cioè un indirizzo univoco per la risorsa) .
Risorsa: un oggetto (uno studente, un conto corrente, …) che può avere diverse rappresentazioni, cioè diversi profili, che possono essere resi disponibili.
Esempi di risorsa: testo, immagini, musica, …

Storicamente, una pagina web è un file HTML che definisce la struttura e i contenuti della pagina, testo più altri elementi (immagini, elementi grafici, …)
HTML: **HyperText Markup Language**, è il linguaggio con cui si scrivono gliipertesti (si

definisce la struttura di una pagina, i collegamenti, si inseriscono immagini, …).
Ipertesto: insieme di testi o pagine che possono essere letti in maniera non sequenziale secondo percorsi definiti dagli hyperlink (o più semplicemente link, cioè collegamenti). Compongono una rete raggiata o variamente incrociata di informazioni organizzate secondo criteri paritetici o gerarchici.

Definizione di Web Server

Un web server è un software o un'infrastruttura hardware che fornisce risorse o servizi su richiesta agli utenti tipicamente su internet o su una rete locale.

In altre parole, un web server è un server specializzato progettato per gestire le richieste di siti web, pagine web, file multimediali e altre risorse web da parte dei client, come i browser web. Quando un client, come ad esempio un browser web, fa una richiesta a un web server per accedere a una risorsa, il server risponde inviando la risorsa richiesta attraverso la rete al client.

Differenza tra URI, URL e URN

URI: *Uniform Resource Identifier*, stringa di caratteri che identifica in modo univoco una risorsa su Internet o in una rete di computer.

URL: *Uniform Resource Locator*, identifica un oggetto nella rete e specifica il protocollo da usare per ricevere/inviare dati. Ha cinque componenti principali:
1. nome del protocollo
2. indirizzo dell'host
3. porta del processo (la controparte)
4. percorso nell'host
5. Identificatore della risorsa

```
protocollo://indirizzo_IP[:porta]/cammino/risorsa
    1.              2.          3.      4.      5.
```

Esempi: ftp://www.adobe.com/download/acroread.exe,
http://www.biblio.unimib.it/go/Home/Home-English/Services

URN: *Uniform Resource Name*, fornisce un nome univoco per una risorsa, ma non include informazioni su come accedervi o dove si trova. Gli URN sono concepiti per essere persistenti e indipendenti dalla posizione, il che significa che l'identificativo URN della risorsa rimarrà lo stesso anche se la risorsa viene spostata o se il metodo per accedervi cambia.

In breve, mentre URI è il termine generico che include entrambi URL e URN, URL fornisce un modo per localizzare una risorsa specifica su Internet, mentre URN fornisce un nome univoco per una risorsa senza necessariamente fornire informazioni su dove trovare la risorsa.

Linguaggi del Web

I **dati testuali** sono espressi in linguaggi standard:

- *HTML* (per definire la struttura dei contenuti e la loro impaginazione)
  può contenere CSS (per gestire la presentazione, cioè il rendering), …
- *XML* (focalizzato sui dati e la loro struttura)
  XSL, RDF, …
- *JSON* (focalizzato sui dati e la loro struttura)

I **dati non testuali** (immagini, audio, video)
Encoding MIME (definisce il formato dei contenuti)
Multipurpose Internet Mail Extension, qualifica i dati inviati via Internet (in http, qualifica il tipo dei dati del body). Sistema di codifica utilizzato per convertire i dati binari in una forma di testo compatibile con i protocolli Internet.
Definito da 5 categorie che possono essere specializzate con sottotipi.
- Text
Esempi di sottotipi: plain, html
- Image
Esempi di sottotipi : jpeg, gif
- Audio
Esempi di sottotipi : basic (8-bit mu-law encoded), 32kadpcm (32 kbps coding)
- Video
Esempi di sottotipi : mpeg, quicktime
- Application
Dati che devono essere processati da un' applicazione prima di essere "visibili"
Esempi di sottotipi : msword, octetstream

Le pagine Web possono contenere del codice espresso in linguaggi di scripting per arricchire l'**interazione e rendere le pagine attive**
JavaScript, VBScript, Java/Applet, Adobe Flash …

Protocollo HTTP

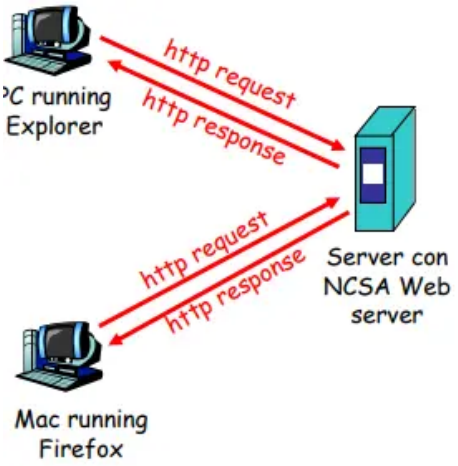
Definizione di protocollo

I protocolli definiscono il formato, l'ordine di invio e di ricezione dei messaggi tra i dispositivi, il tipo dei dati e le azioni da eseguire quando si riceve un messaggio.
Per poter capire le richieste e formulare le risposte i due processi devono concordare un protocollo.

Esempi: HTTP, FTP, SMTP...

Il protocollo HTTP è un protocollo di livello applicativo per il Web. Utilizza il modello client/server:

- Client: Browser che richiede, riceve e "mostra" oggetti Web
- Server: Web server che invia oggetti in risposta alle richieste



HTTP è stateless

Il termine "stateless" (letteralmente "senza stato") nel contesto di HTTP si riferisce al fatto che ogni richiesta HTTP viene trattata indipendentemente dalle altre richieste precedenti o successive. Questo significa che il server HTTP non mantiene alcuna informazione sullo stato (state) della sessione tra le richieste da parte dello stesso client.

In un protocollo stateless come HTTP, ogni richiesta HTTP viene gestita separatamente, senza che il server ricordi lo stato delle richieste precedenti del client. Ciò significa che il server non mantiene informazioni sulle sessioni utente, sullo stato di autenticazione o sulle azioni precedenti eseguite dal client. Ogni richiesta è autonoma e indipendente dalle altre e deve contenere tutte le informazioni necessarie per la sua esecuzione (self-contained).

Questa caratteristica è una delle ragioni per cui HTTP è considerato un protocollo leggero e scalabile, poiché non richiede al server di mantenere grandi quantità di informazioni sullo stato delle sessioni dei client.

I protocolli che mantengono informazione di stato sono complessi (es. TCP).

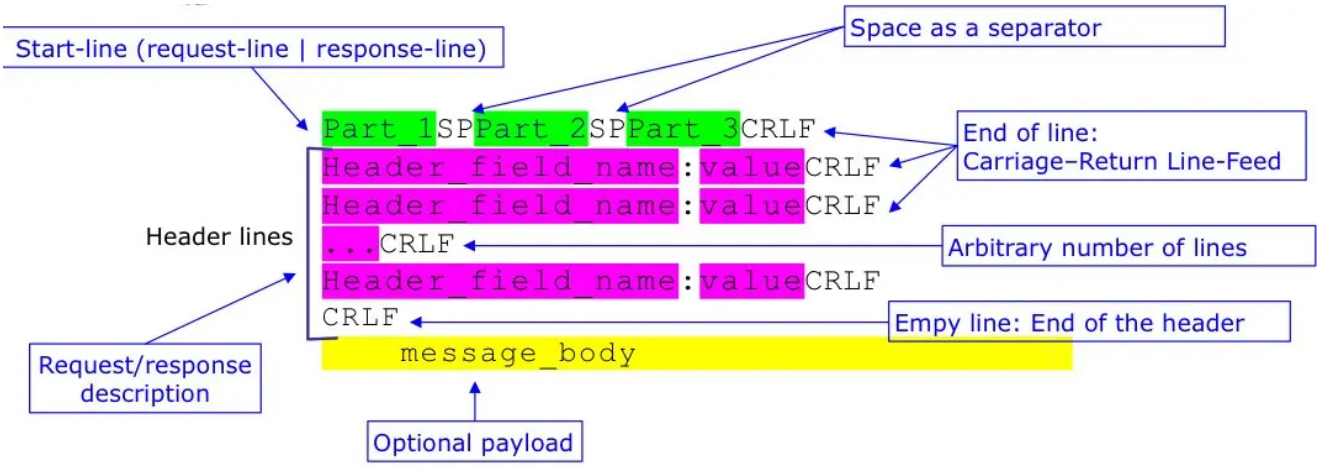
HTTP usa TCP

HTTP utilizza TCP come protocollo di trasporto per la comunicazione tra client e server su Internet.

1. Il client inizia una connessione TCP (crea una socket) verso il server sulla porta 80
2. Il server accetta la connessione TCP dal client
3. Vengono scambiati messaggi http (messaggi del protocollo di livello applicativo) tra il browser (client http) e il Web server (server http)

Formato dei messaggi

Due tipi di messaggi http: ***request, response***
- ASCII (formato testo leggibile)
- Hanno la stessa struttura (start-line, headers e body)

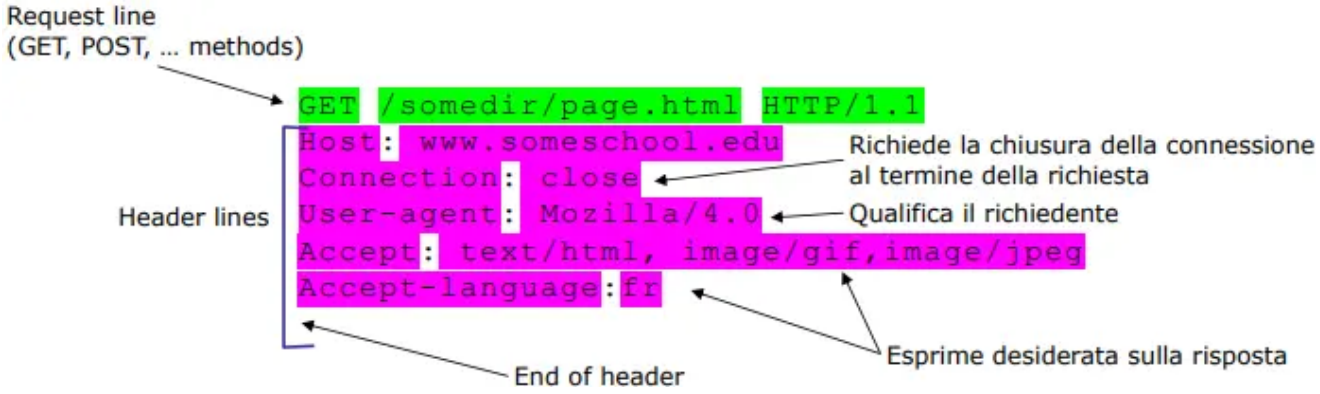


Richiesta HTTP

- **Linea di richiesta (Start-line)** : La prima riga di una richiesta contiene il metodo HTTP utilizzato (come GET, POST, PUT, DELETE, ecc.), l'URI della risorsa richiesta e la versione del protocollo HTTP (ad esempio, HTTP/1.1).
- **Intestazioni (Headers)**: Le intestazioni contengono informazioni aggiuntive sulla richiesta, come le informazioni sull'host, i tipi di contenuto accettati, le credenziali di autenticazione, ecc.
- **Corpo del messaggio (Body)**: Facoltativo. Contiene i dati inviati al server, ad esempio nel caso di richieste POST o PUT.

Serve un Carriage-Return Line Feed alla fine (CRLF).

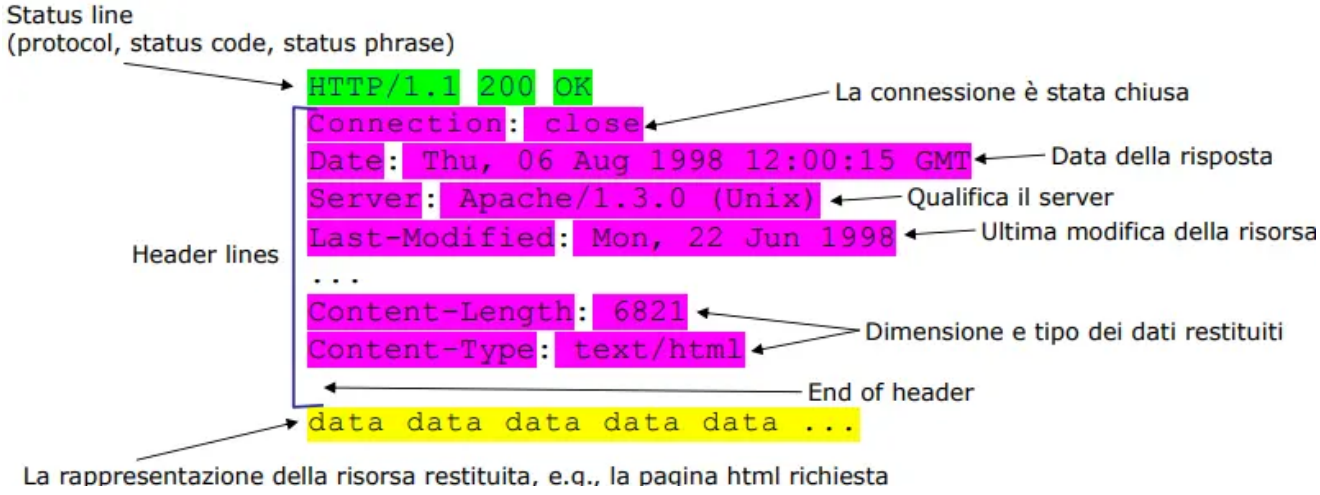
Esempio di richiesta HTTP



Risposte HTTP

- **Linea di stato (Start-line)**: La prima riga di una risposta contiene la versione del protocollo HTTP utilizzato e uno status code che indica se la richiesta è stata eseguita con successo, ha riscontrato un errore o ha avuto altri risultati.
- **Intestazioni (Headers)**: Le intestazioni contengono informazioni aggiuntive sulla risposta, come la data, il tipo di contenuto restituito, la lunghezza del corpo della risposta, ecc.
- **Corpo del messaggio (Body)**: Contiene i dati restituiti dal server al client, ad esempio il contenuto di una pagina web o di un file.

Esempio di risposta HTTP



HTTP 1.0: Server chiude connessione al termine della richiesta
HTTP 1.1: mantiene aperta la connessione oppure chiude se la richiesta contiene Connection: close

Domanda: Perché sono state inserite le informazioni sulla dimensione e il tipo dei dati restituiti?

1. **Interpretazione corretta dei dati**: Specificare il tipo dei dati restituiti consente al client di interpretare correttamente il contenuto della risposta. Ad esempio, se il tipo di contenuto è "text/html", il client sa che può trattare il corpo della risposta come HTML e visualizzarlo correttamente nel browser. Se il tipo di contenuto è "application/json", il

client sa che il corpo della risposta contiene dati in formato JSON che possono essere interpretati e utilizzati in base alle specifiche del formato JSON.
2. **Gestione delle risorse**: Conoscere la dimensione dei dati restituiti consente al client di allocare risorse di memoria adeguate per gestire la risposta. Ad esempio, un client può allocare un buffer di dimensioni adeguate per memorizzare il corpo della risposta senza rischiare di esaurire la memoria disponibile.
3. **Ottimizzazione delle prestazioni**: Conoscere la dimensione dei dati restituiti può aiutare sia il client che il server a ottimizzare le prestazioni.

In breve, specificare la dimensione e il tipo dei dati restituiti nei messaggi di risposta HTTP è fondamentale per consentire una corretta interpretazione e gestione dei dati da parte del client, nonché per ottimizzare le prestazioni e l'efficienza delle comunicazioni tra client e server.

Codici di stato

Codice	Tipologia	Informazione
1xx	Informational	Request received; server is continuing the process.
2xx	Success	The request was successfully received, understood, accepted and serviced.
3xx	Redirection	Further action must be taken in order to complete the request.
4xx	Client Error	The request contains bad syntax or cannot be understood.
5xx	Server Error	The server failed to fulfill an apparently valid request.

Esempi:

- *200 OK*
  Successo, se richiesto l’oggetto è contenuto nel messaggio
- *301 Moved Permanently*
  L’oggetto richiesto è stato spostato. Il nuovo indirizzo è specificato nell’header (Location: …)
- *400 Bad Request*
  Richiesta incomprensibile al server
- *404 Not Found*
  Il documento non è stato trovato sul server
- *505 HTTP Version Not Supported*

Metodi HTTP

		cache	safe	idempotent
OPTIONS	represents a request for information about the communication options available on the request/response chain identified by the Request-URI			✓
GET	means retrieve whatever information (in the form of an entity) is identified by the Request-URI	✓	✓	
HEAD	identical to GET except that the server MUST NOT return a message-body in the response	✓	✓	
POST	is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line			
PUT	requests that the enclosed entity be stored under the supplied Request-URI			✓
DELETE	requests that the origin server delete the resource identified by the Request-URI			✓
TRACE	is used to invoke a remote, application-layer loop- back of the request message			✓

Metodo Safe: Un metodo HTTP è considerato "safe" se non modifica lo stato del server o delle risorse sul server durante l'esecuzione. In altre parole, una richiesta di un metodo sicuro non dovrebbe avere effetti collaterali sullo stato del server. Questo significa che una richiesta "safe" non modifica, elimina o aggiunge risorse sul server.

Metodo Idempotente: Un metodo HTTP è considerato "idempotente" se l'applicazione ripetutamente esegue lo stesso metodo con gli stessi parametri e lo stato del server rimane invariato. In termini più semplici, una richiesta idempotente può essere ripetuta più volte senza causare cambiamenti diversi dal primo tentativo. Questo è particolarmente importante quando si considerano le possibili ripetizioni delle richieste a causa di errori di rete o di altre condizioni transitorie.

L'importanza di questi concetti risiede nel fatto che consentono agli sviluppatori e agli architetti di progettare e implementare servizi web in modo coerente e affidabile.
Ad esempio, quando si sviluppa un'applicazione web, è importante capire se una determinata richiesta modificherà lo stato del server (non sicura), se può essere ripetuta senza conseguenze (idempotente) e se può essere eseguita senza modificare lo stato del server (sicura). Queste proprietà aiutano a garantire che le applicazioni interagiscano in modo prevedibile con i servizi web e che le operazioni di rete possano essere gestite in modo affidabile.

Metodo GET

- Restituisce una rappresentazione di una risorsa.
- Include eventuali parametri in coda alla URL della risorsa.
- È safe: l'esecuzione non ha effetti sul server => la risposta può essere gestita con una cache dal client
- Uso tipico: ottenere dati in formato di pagine html e immagini, dati XML o JSON

La GET include eventuali parametri in coda alla URL della risorsa (detti query)

```
GET resource[?key=value{&key=value}] HTTP1.1
{header lines}
```

Esempio: dammi tutti gli ordini di marzo per il prodotto con codice Q2345

```
/myCompany/orders?item="Q2345"&date="2022/03"
```

Metodo POST

- Comunica dei dati da elaborare lato server o crea una nuova risorsa subordinata all'URL indicata.
- L'input segue come documento autonomo (body)
- Non è idempotente: ogni esecuzione ha un diverso effetto => La risposta NON può essere gestita con una cache dal client
  Uso tipico: processare FORM e modificare dati in un DB

La POST prevede che i dati vengano messi in coda come documento autonomo (detto body)

```
POST resource HTTP1.1
{header lines}

body
```

Esempio: aggiorna i codici di un prodotto in tutti gli ordini di aprile POST

```
POST /myCompany/orders HTTP1.1
Content-Length: 59
Content-Type: application/x-www-form-urlencoded
update=true&oldItem="Q2345"&newItem="Q68254"&date="2022/04"
```

Metodo HEAD

Simile al metodo GET ma viene restituito solo l'Head della pagina Web. Spesso usato in fase di debugging.

Applicazioni Web

GET condizionale

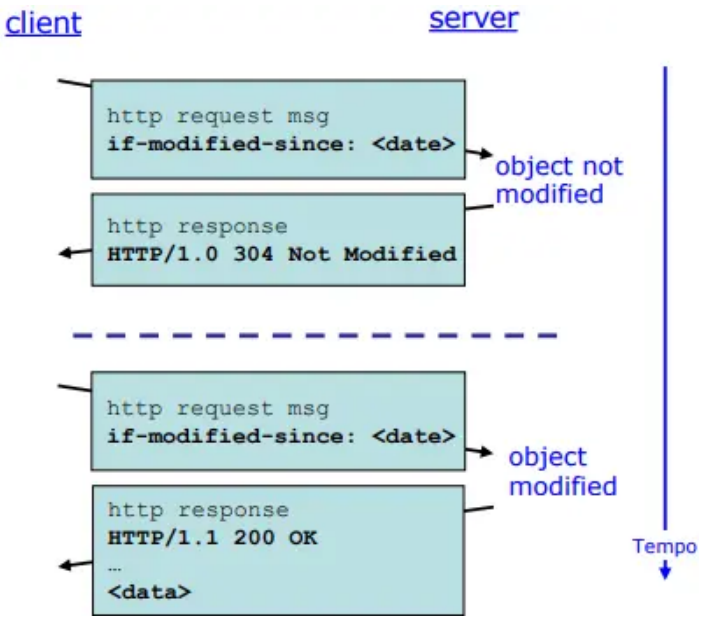
Obiettivo: non re-inviare oggetti che il client ha già in cache

Client: header con la data dell'oggetto memorizzato in cache

```
if-modified-since: <date>
```

Server: la risposta non ha body se l'oggetto in cache è aggiornato

```
HTTP/1.0 304 Not Modified
```



Cookie

Obiettivo: associare un identificatore alla conversazione per creare uno "stato"

Server: invia un “cookie” con la risposta

```
set-cookie: 1678453
```

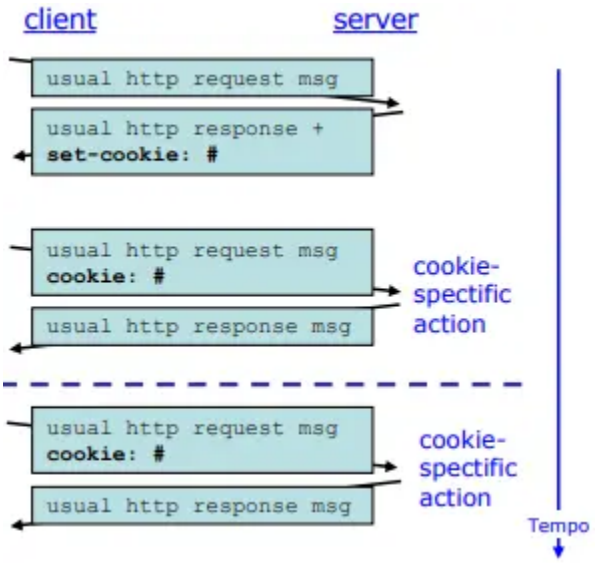
Client: include il cookie nei messaggi

```
cookie: 1678453
```

Il server controlla il cookie per

- Autenticazione
- Sessione di lavoro

NB: viola le regola stateless



Autenticazione

Obiettivo: controllare l'accesso alle risorse
Http è un protocollo stateless: il client deve autenticare ogni richiesta

Server: rifiuta la connessione e invia

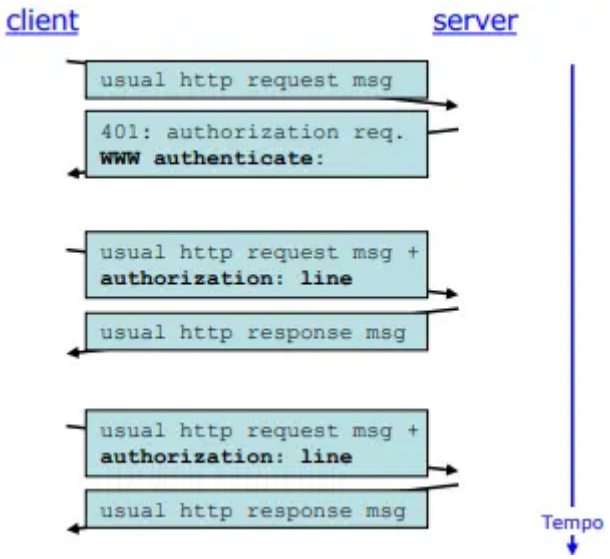
```
401 authorization required

WWW authenticate:
```

Client: include login e password nell'header del messaggio di richiesta

```
authorization: line
```

NB: Il client memorizza i dati in modo che l'utente non debba digitarli ogni volta



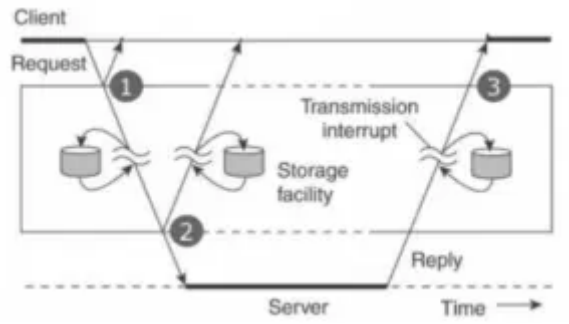
Tipi di comunicazione

Esistono tre tipi di comunicazione:

1. **Comunicazione sincrona**: il mittente invia un messaggio al destinatario e attende una risposta immediata prima di procedere ulteriormente. Il mittente e destinatario sono sincronizzati nel tempo.
2. **Comunicazione asincrona**: il mittente invia un messaggio al destinatario senza aspettare una risposta immediata. Il mittente può continuare a eseguire altre attività senza bloccarsi in attesa della risposta del destinatario e quest'ultimo quando riceve il messaggio può elaborarlo e rispondere in un secondo momento, senza essere vincolato a un tempo specifico.

Sincronizzazione

In una comunicazione, la sincronizzazione può avvenire in tre punti:



1. **Request submission**: processo di invio di una richiesta (request) da parte di un client a un server attraverso un protocollo di comunicazione (HTTP). Viene preso in carico dal middleware;
2. **Request delivery**: processo attraverso il quale la richiesta inviata dal client viene ricevuta e recapitata correttamente al server designato (middleware ricevente);
3. **Request processing**: processo attraverso il quale un server riceve, analizza e risponde alle richieste inviate da un client attraverso un protocollo di comunicazione (HTTP).

La sincronizzazione può essere:

- **Transiente**: Se il destinatario non è connesso, i dati vengono scartati.
- **Persistente**: Il middleware memorizza i dati fino alla consegna del messaggio al destinatario. Non è necessario che i processi siano in esecuzione prima e dopo l'invio/ricezione dei messaggi.

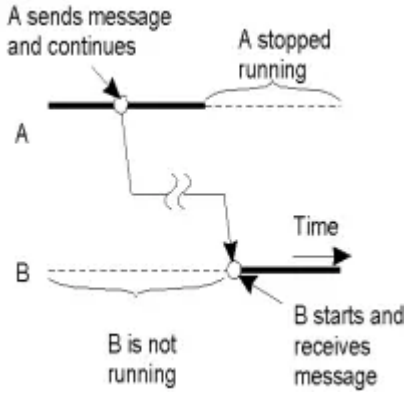
Persistenza e sincronicità in una comunicazione

Comunicazione persistente asincrona

Modello di comunicazione in cui i messaggi inviati sono conservati nel middleware anche dopo l'invio e il mittente e il destinatario non sono sincronizzati nel tempo, consentendo al

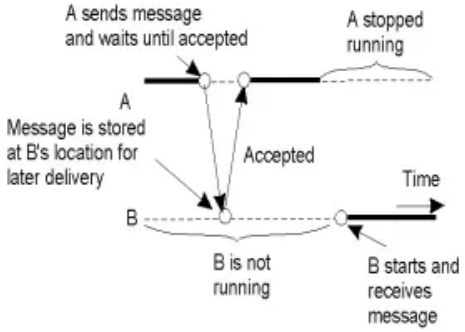
destinatario di ricevere e rispondere ai messaggi in modo indipendente dal mittente e senza la necessità di rispondere immediatamente.

Spesso utilizzato in scenari in cui è necessario garantire che i messaggi non vadano persi anche in caso di disconnessione temporanea o di indisponibilità del destinatario. Nell'esempio, infatti, B inizialmente non è attivo.



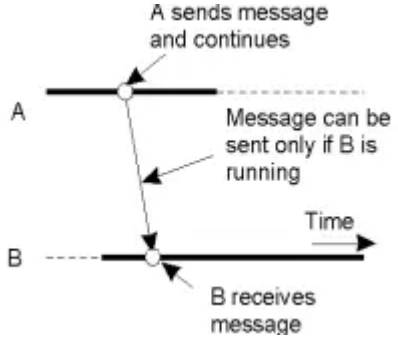
Comunicazione persistente sincrona

Il messaggio inviato dal mittente A è conservato nel sistema di comunicazione anche dopo che è stato inviato, ma il mittente A aspetta comunque una risposta immediata dal middleware del destinatario B.



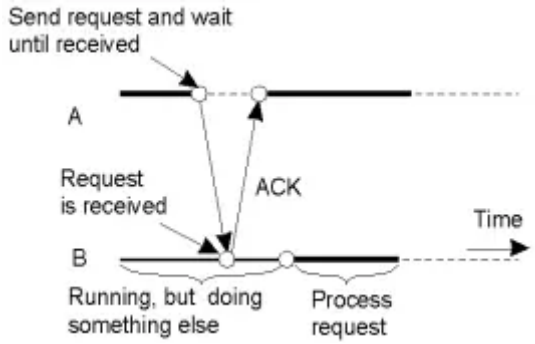
Comunicazione transiente asincrona

Il mittente non deve attendere una risposta immediata dal destinatario. I messaggi possono essere elaborati o gestiti dal destinatario quando è conveniente, senza la necessità di rispondere immediatamente. Inoltre, i messaggi trasmessi possono essere rimossi dal sistema una volta che sono stati elaborati o consegnati con successo al destinatario. Se B non è connesso, i messaggi vengono scartati.



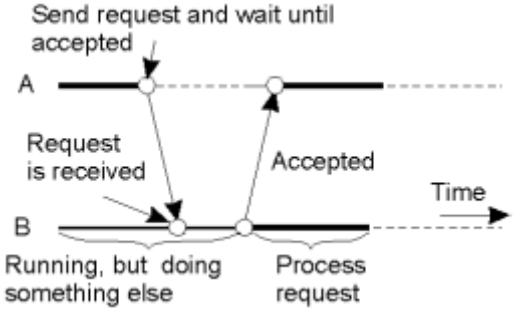
Comunicazione transiente sincrona basata su ricevuta

Il mittente invia un messaggio al destinatario e successivamente aspetta una ricevuta (ACK) di conferma dal destinatario. Questo significa che il mittente è sincronizzato nel tempo con il destinatario fino a quando non riceve la ricevuta. Una volta ricevuta la conferma, il mittente può procedere ulteriormente con altre attività.
Fa diventare persistente la comunicazione, se il sistema non lo fa, è il programma che mantiene il dato non un middleware. Inoltre B non processa subito, legge solo inviando un ack, e processa più avanti.



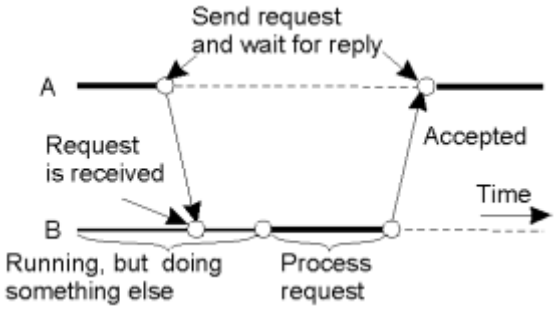
Comunicazione transiente sincrona basata su consegna

Il mittente invia un messaggio al destinatario e rimane in attesa finché non riceve conferma che il messaggio è stato consegnato con successo. Questo tipo di comunicazione richiede un feedback rapido sullo stato della consegna del messaggio. Infatti B manda un accept solo una volta che inizia ad elaborare la richiesta.



Comunicazione transiente sincrona basata su risposta

il mittente invia un messaggio al destinatario e rimane in attesa finché non riceve una risposta. Questo tipo di comunicazione richiede una sincronizzazione temporale tra mittente e destinatario, in quanto il mittente non procederà ulteriormente fino a quando non avrà ricevuto una risposta.



Comunicazione persistente

Message-Queuing Model

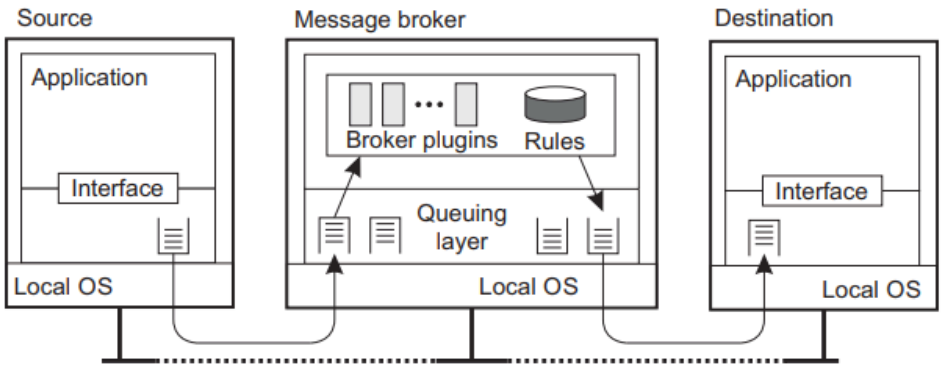
Offrono capacità di archiviazione a medio termine per i messaggi, senza richiedere né al mittente né al destinatario di essere attivi durante la trasmissione dei messaggi.
Esistono quattro modalità in base allo stato del sender o del receiver in quanto possono essere **running** o **passive**.

Primitive

Primitive	Meaning
Put	Aggiunge un messaggio a una coda
Get	Blocca fino a quando la coda specifica non è vuota e rimuove il primo messaggio
Poll	Controlla una coda specifica per messaggi e rimuove il primo. Non blocca.
Notify	Installa un gestore da chiamare quando viene inserito un messaggio nella coda

Message Brokers

I "message brokers" sono componenti software che fungono da intermediari nella trasmissione di messaggi tra diverse applicazioni o sistemi all'interno di un'architettura di comunicazione distribuita. Il broker instrada i messaggi in base a determinati criteri e gestiscono le code di messaggi, garantendo l'affidabilità della trasmissione.
Utilizzo dei protocolli **Publish and subscribe**.



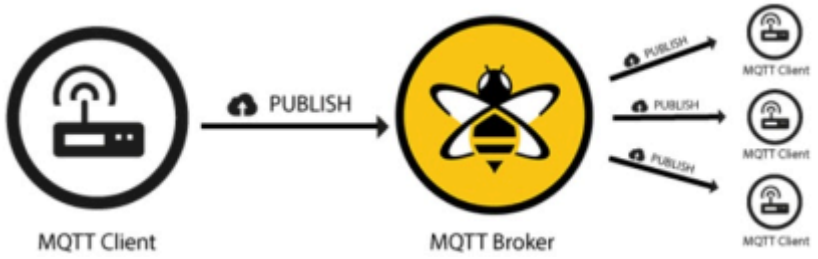
Publish and subscribe pattern

Disaccoppiamento tra mittente (publisher) e destinatario (subscriber)
Un broker fornisce un servizio di smistamento messaggi (dati o eventi) appartenenti a un certo argomento (topic)
• I mittenti si registrano dichiarando: «voglio inviare messaggi su un certo argomento»
• I destinatari si registrano dichiarando: «voglio ricevere messaggi su un certo argomento»

Caratteristiche

- Il mittente e il destinatario non comunicano direttamente
  Indipendenza tra le componenti e concorrenza, con conseguente scalabilità
- È possibile realizzare una comunicazione molti-a-molti
- È possibile realizzare una comunicazione persistente

Le politiche per il mantenimento dei dati del broker possono essere diverse.



MQTT è l'implementazione più utilizzata del pattern Publish-and-Subscribe.