Tecnologie Web
Il protocollo HTTP

WWW = URL + HTTP + HTML

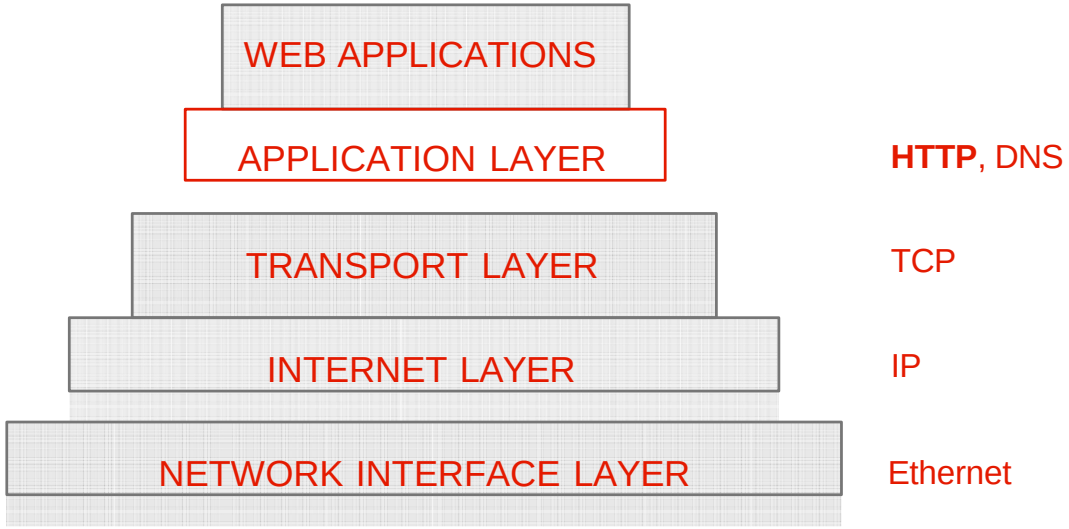
- **HTTP** è l'acronimo di **H**yper**T**ext **T**ransfer **P**rotocol
- E' il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) il server e il client
- Gestisce sia le **richieste** (URL) inviate al server che le **risposte** inviate al client (pagine)
- E' un protocollo **stateless**: né il server né il client mantengono, a livello di protocollo, informazioni relative ai messaggi precedentemente scambiati
- Ci sono state tre versioni di HTTP: 0.9, 1.0 e 1.1

HTTP: Terminologia

- **Client**: Programma applicativo che stabilisce una connessione al fine di inviare delle richieste
- **Server**: Programma applicativo che accetta connessioni al fine di ricevere richieste ed inviare specifiche risposte con le risorse richieste.
- **Connessione**: circuito virtuale stabilito a livello di trasporto tra due applicazioni per fini di comunicazione
- **Messaggio**: è l'unità base di comunicazione HTTP, è definita come una specifica sequenza di byte concettualmente atomica.
  - **Request**: messaggio HTTP di richiesta
  - **Response**: messaggio HTTP di risposta
  - **Resource**: Oggetto di tipo dato univocamente definito
  - **URI**: Uniform Resource Identifier–identificatore unico per una risorsa.
- **Entity**: Rappresentazione di una risorsa, può essere incapsulata in un messaggio.

HTTP nello stack TCP/IP

HTTP si situa a livello **application** nello stack TCP/IP



- **E’ un protocollo basato su TCP**
- **Sia le richieste al server che le risposte ai client sono trasmesse usando stream TCP**
- **Segue uno schema di questo tipo:**
  - Il server rimane in ascolto, tipicamente sulla porta 80
  - Il client apre una connessione TCP sulla porta 80
  - Il server accetta la connessione
  - Il client manda una richiesta
  - Il server invia la risposta e chiude la connessione

Esempio HTTP (1.0)

- Ipotizziamo di volere richiedere una pagina composta da un file HTML e 10 immagini JPEG:

1. Il client http inizia una connessione TCP verso il server http sull' host sulla porta 80
2. Il server http è "in ascolto" sulla porta 80. "Accetta" la richiesta di connessione e ne dà conferma al client
3. Il client http invia un messaggio di richiesta http (**request message**) contenente la URL
4. Il server http riceve il messaggio di richiesta, costruisce un messaggio di risposta (**response message**) contenente l'oggetto richiesto e e lo invia
5. Il client http riceve il messaggio di risposta contenente il file html, visualizza la pagina html. Analizzando il file html, il browser trova i riferimenti a 10 oggetti jpeg
6. Il server http chiude la connessione TCP.
7. I passi 1-6 sono ripetuti per ciascuno dei 10 oggetti jpeg

Differenze fra HTTP 1.0 e 1.1

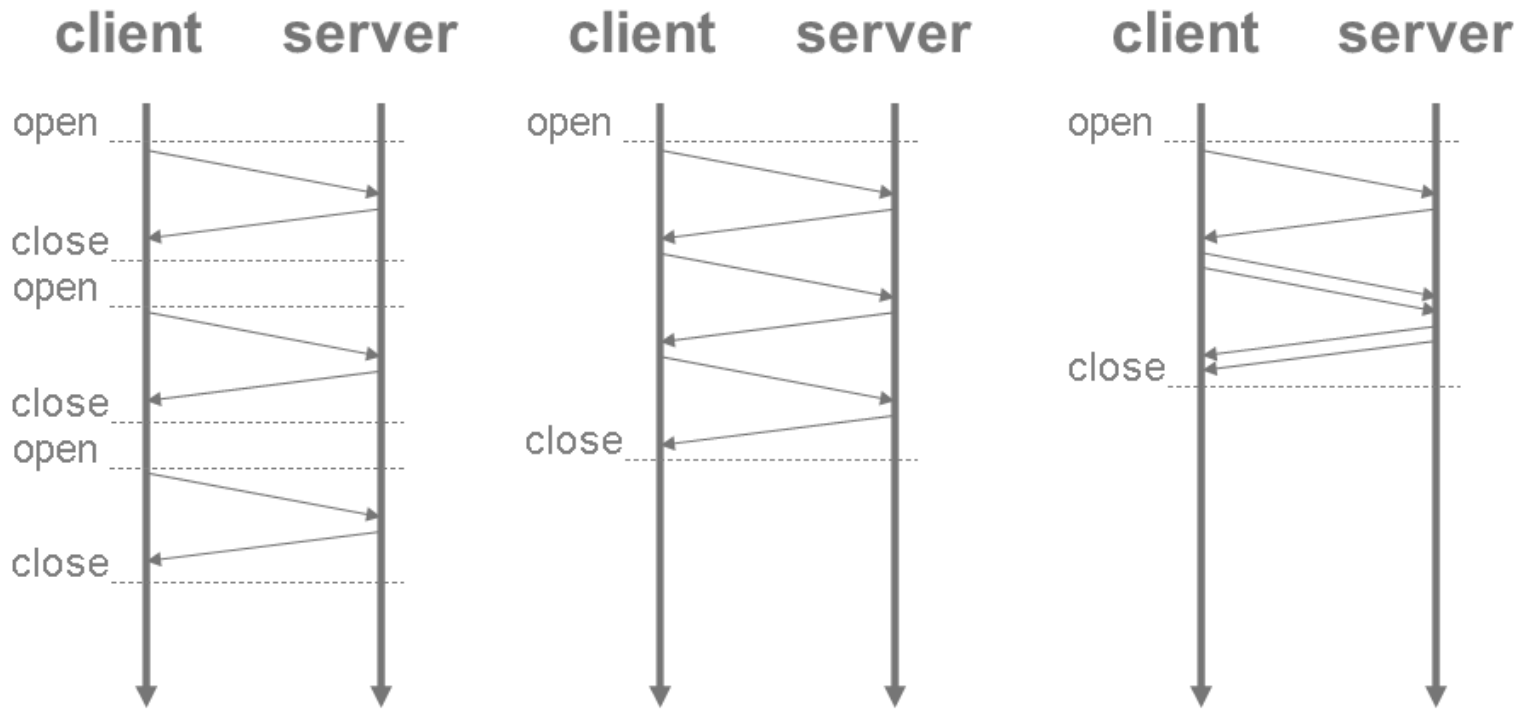
- La connessione HTTP è composta da una serie di richieste ed una serie corrispondente di risposte.
- La differenza principale tra HTTP 1.0 e 1.1 è la possibilità di specificare coppie multiple di richiesta e risposta nella stessa connessione.
- Le connessioni 1.0 vengono dette **non persistenti** mentre quelle 1.1 vengono definite **persistenti**
- Il server lascia aperta la connessione TCP dopo aver spedito la risposta e può quindi ricevere le richieste successive sulla stessa connessione.
- Nell'esempio precedente l'intera pagina web (file HTML e 10 immagini) possono essere inviate nella stessa connessione TCP

  Il server HTTP chiude la connessione quando viene specificato nell'header del messaggio oppure quando non è usata da un certo tempo (time out)

HTTP 1.1 e pipelining

- Per migliorare ulteriormente le prestazioni si può usare usa la tecnica del **pipelining**
- Il pipelining consiste nell'invio di molteplici richieste da parte del client prima di ricevere le risposte
- Le risposte debbono però essere date nello stesso ordine delle richieste, poiché non è specificato un metodo esplicito di associazione tra richiesta e risposta

Confronto fra tipi di connessione



Messaggi

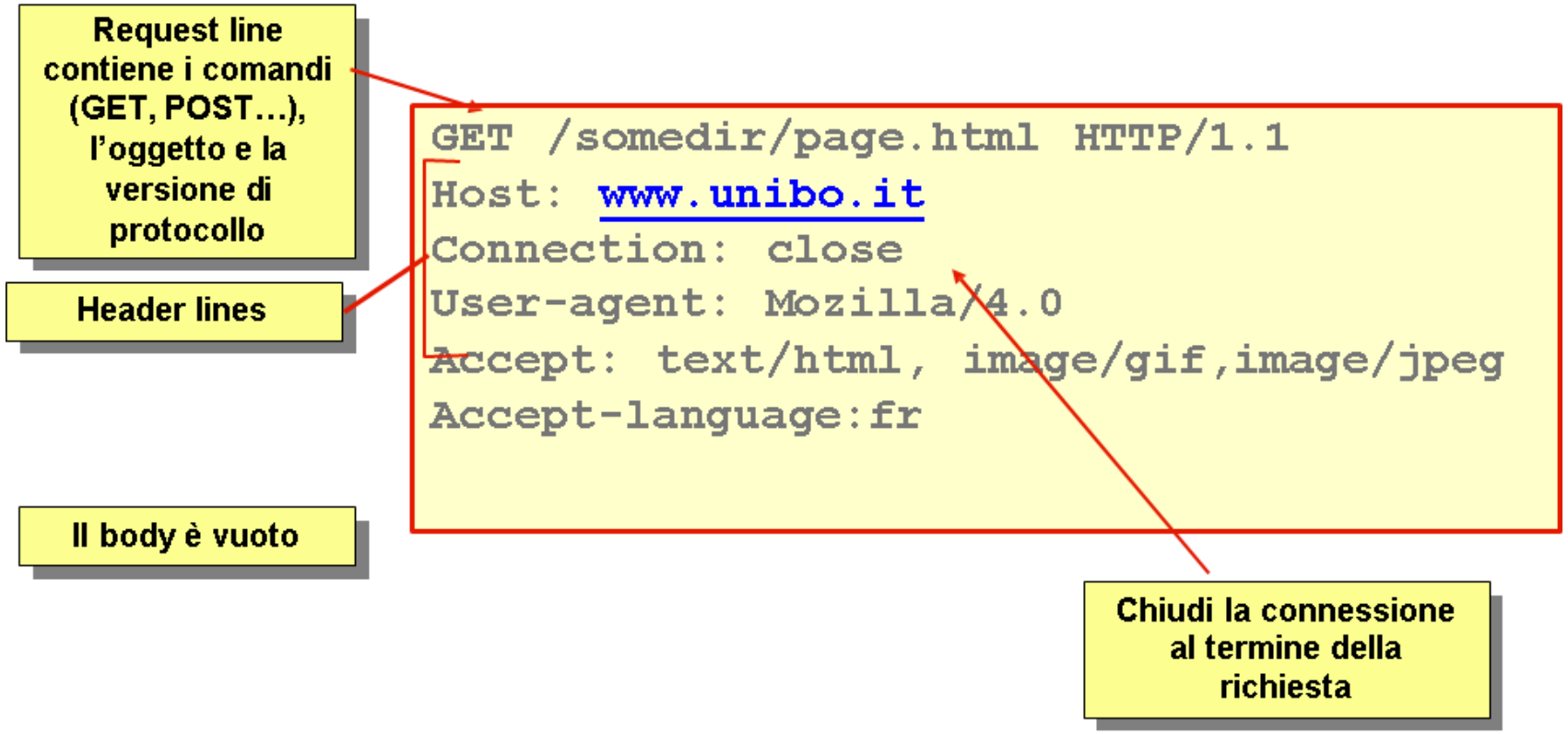
- Un messaggio HTTP è definito da due strutture:
  - Message Header: Contiene tutte le informazioni necessarie per la identificazione del messaggio (più in generale tutte le intestazioni del messaggio)
  - Message Body: Contiene i dati trasportati dal messaggio.
- Esistono schemi precisi per ogni tipo di messaggio relativamente agli header ed ai body
- I messaggi di Response contengono i dati relativi alle risorse richieste (tipicamente una pagina html)
- I dati sono codificati secondo il formato specificato nell'header
- Solitamente sono in formato MIME (Multipurpose Internet Mail Extensions)

Header HTTP

- **Gli header sono righe (nome: valore) che specificano caratteristiche del messaggio trasmesso o ricevuto:**
  - Header generali della trasmissione
    - Data, codifica, versione, tipo di comunicazione, ecc.
  - Header dell'entità trasmessa
    - Content-type, Content-Length, data di scadenza, ecc.
  - Header della richiesta effettuata
    - Chi fa la richiesta, a chi viene fatta la richiesta, che tipo di caratteristiche il client è in grado di accettare, che autorizzazione puo' portare, ecc.
  - Header della risposta generata
    - Che server dà la risposta, che tipo di autorizzazione è necessaria, ecc.

Messaggi HTTP: esempio di richiesta

- Il protocollo utilizza messaggi in formato ASCII (testo leggibile)
- Esempio di messaggio http request:



Un esempio un po' più complesso

```
GET /search?q=Introduction+to+XML+and+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.2)
  Gecko/20040803
Accept: text/xml,application/xml,application/xhtml+xml,
  text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: da,en-us;q=0.8,en;q=0.5,sw;q=0.3 Accept-
Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 Keep-
Alive: 300
Connection: keep-alive
Referer: http://www.google.com/
```

I comandi della richiesta - GET

- **GET**
  - **Serve per richiedere una risorsa ad un server**
  - **E' il metodo più frequente: è quello che viene attivato facendo click su un link ipertestuale di un documento HTML, o specificando un URL nell'apposito campo di un browser.**
  - **E' previsto il passaggio di parametri (la parte `<query>` dell'URL)**
  - **La lunghezza massima di un URL è limitata**

I comandi della richiesta - POST

- **POST**
  - Serve per richiedere una risorsa
  - A differenza del GET i dettagli per la identificazione e la elaborazione della risorsa stessa non sono nell'URL ma sono contenuti nel body del messaggio
  - Non ci sono limiti di lunghezza nei parametri di una richiesta
  - POST viene usato per esempio per sottomettere i dati di una form HTML ad un'applicazione CGI sul server.
  - Si ha una trasmissione di informazioni che però non porta alla creazione di una risorsa sul server

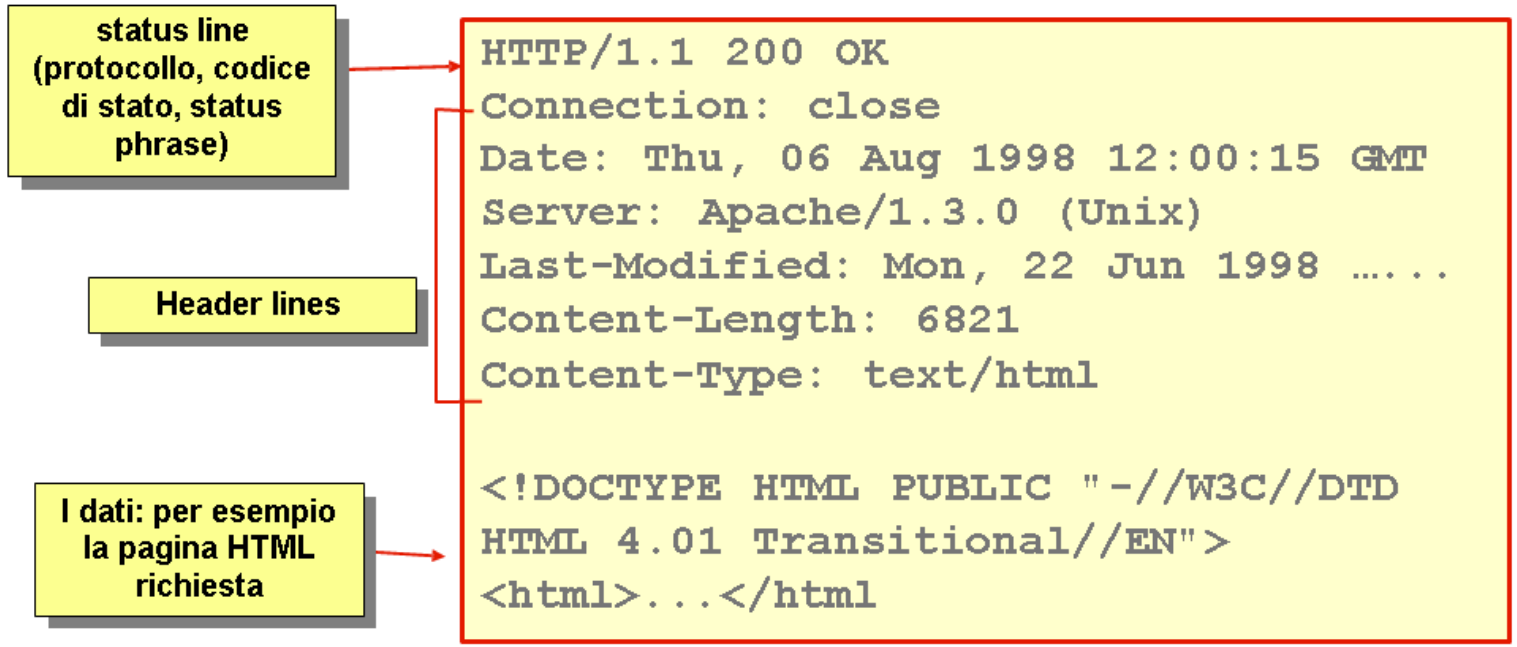
I comandi della richiesta - PUT e DELETE

- PUT
  - Chiede la memorizzazione sul server di una risorsa all'URL specificato
  - Il metodo PUT serve quindi per trasmettere delle informazioni dal client al server
  - A differenza del Post però si ha la creazione di una risorsa (o la sua sostituzione se esisteva già).
  - L'argomento del metodo PUT è la risorsa che ci si aspetta di ottenere facendo un GET in seguito con lo stesso nome.
- DELETE
  - Richiede la cancellazione della risorsa riferita dall'URL specificato..
- Sono normalmente disabilitati sui server pubblici

I comandi della richiesta – HEAD, OPTIONS e TRACE

- **HEAD:** è simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza il corpo.
- Viene usato per verificare un URL
  - Validità: la risorsa esiste e non è di lunghezza zero
  - Accessibilità: non è richiesta autenticazione
- **OPTIONS:** serve per richiedere informazioni sulle opzioni disponibili per la comunicazione.
- **TRACE:** è usato per invocare il loop-back remoto a livello applicativo del messaggio di richiesta.
- Consente al client di vedere cosa è stato ricevuto dal server: viene usato nella diagnostica e nel testing dei servizi web.

Il formato della risposta



- HTTP 1.0: Il server chiude la connessione al termine della richiesta
- HTTP 1.1: il server mantiene aperta la connessione oppure la chiude se si mette la clausola: `Connection: close`

I codici di stato

- **Lo status code è un numero di tre cifre, di cui la prima indica la classe della risposta, e le altre due la risposta specifica.**
- **Ci sono 5 classi:**
  - 1xx: Informational. **Una risposta temporanea alla richiesta, durante il suo svolgimento (sconsigliata a partire da HTTP 1.0)**
  - 2xx: Successful. **Il server ha ricevuto, capito e accettato la richiesta**.
  - 3xx: Redirection. **Il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta.**
  - 4xx: Client error. **La richiesta del client non può essere soddisfatta per un errore da parte del client (errore sintattico o richiesta non autorizzata).**
  - 5xx: Server error. **La richiesta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno (suo o di applicazioni CGI).**

Esempi di codici di stato

- 100 Continue (se il client non ha ancora mandato il body)
- 200 Ok (GET con successo)
- 201 Created (PUT con successo)
- 301 Moved permanently (URL non valida, il server conosce la nuova posizione
- 400 Bad request (errore sintattico nella richiesta)
- 401 Unauthorized (manca l'autorizzazione)
- 403 Forbidden (richiesta non autorizzabile)
- 404 Not found (URL errato)
- 500 Internal server error (tipicamente un CGI mal fatto)
- 501 Not implemented (metodo non conosciuto dal server)

I cookie

- Parallelamente alle sequenze request/response, il protocollo prevede una struttura dati che si muove come un token, dal client al server e viceversa: i **cookie.**
- I cookie possono essere generati sia dal client che dal server, dopo la loro creazione vengono sempre passati ad ogni trasmissione di request e response.
- Hanno come scopo quello di fornire un supporto per il mantenimento di uno stato in un protocollo come http che è essenzialmente **stateless.**

Struttura dei cookie

- **I cookie sono una collezione di stringhe:**
  - Key: identifica univocamente un cookie all'interno di un dominio:path
  - Value: valore associato al cookie (è una stringa di max 255 caratteri)
  - Path: posizione nell'albero di un sito al quale è associato (di default /)
  - Domain: dominio dove è stato generato
  - Max-age: (opzionale) numero di secondi di vita (permette la scadenza di una sessione)
  - Secure: (opzionale) non molto usato. Questi cookie vengono trasferiti se e soltanto se il protocollo è sicuro (https)
  - Version: identifica la versione del protocollo di gestione dei cookie

Autenticazione

- Esistono situazioni in cui si vuole restringere l'accesso alle risorse ai soli utenti abilitati
- Tecniche comunemente utilizzate
  - Indirizzo IP
  - Form per la richiesta di username e password
  - HTTP Basic
  - HTTP Digest

Riconoscimento dell'indirizzo IP

- Basare l'autenticazione sull'indirizzo IP del client è una soluzione che presenta vari svantaggi:
  - Non funziona se l'indirizzo non è pubblico (vedi esempio dei NAT)
  - Non funziona se l'indirizzo IP è assegnato dinamicamente (p.es DHCP)
  - Esistono tecniche che consentono di presentarsi con un IP fasullo (spoofing)
- L'autenticazione HTTP Digest è caduta in disuso negli ultimi anni
- Normalmente si usano
  - Form
  - HTTP Basic

Autenticazione HTTP Basic

- **Challenge:**

```
HTTP/1.1 401 Authorization Required
WWW-Authenticate: Basic realm="The Doe Family Site"
```

- **Response:**

```
Authorization: Basic emFjaGFyaWFzOmFwcGxlcGllcg==
```

Autenticazione Form

- Normalmente si usa il metodo POST
- Analoghe considerazioni a quelle fatte per HTTP Basic

Sicurezza

- Proprietà desiderabili
  - Confidenzialità
  - Integrità
  - Autenticità
  - Non Ripudio

SSL/TSL

- SSL: **Secure Sockets Layer**
- TLS: **Transport Layer Security**

SSL/TSL

- Viene posto un livello che si occupa della gestione di confidenzialità, autenticità ed integrità della comunicazione fra HTTP e TCP
- **Accediamo tramite** `https://...`
- Basato su crittografia a chiave pubblica
  - private key + public key
  - certificato (in genere usato per autenticare il server)

Architetture avanzate per il Web

- Proxy: Programma applicativo in grado di agire sia come Client che come Server al fine di effettuare richieste per conto di altri Clienti. Le Request vengono processate internamente oppure vengono ridirezionate al Server. Un proxy deve interpretare e, se necessario, riscrivere le Request prima di inoltrarle
- Gateway: Server che agisce da intermediario per altri Server. Al contrario dei proxy, il gateway riceve le request come se fosse il server originale ed il Client non è in grado di identificare che la Response proviene da un gateway. Detto anche reverse proxy.
- Tunnel: Programma applicativo che agisce come “blind relay” tra due connessioni. Una volta attivo (in gergo “salito”) non partecipa alla comunicazione http

Caching

- Idea di base: memorizzare copie temporanee di documenti web (es. pagine HTML, immagini) al fine di ridurre l'uso della banda ed il carico sul server.
- Una web cache memorizza i documenti che la attraversano. L'obiettivo è usare i documenti in cache per le successive richieste qualora alcune condizioni siano verificate.
- Tipi di web cache
  - User Agent Cache
  - Proxy Cache

User Agent Cache

- Lo user agent (tipicamente il browser) mantiene una cache delle pagine visitate dall'utente.
- L'uso delle user agent cache era molto importante in passato quando gli utenti non avevano accesso a connessioni di rete a banda larga
- Questo modello di caching è ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente. Nuovi strumenti, es. Google Gears, si basano su questo concetto.

Proxy Cache

- **Forward Proxy Caches**
  - Servono per ridurre le necessità di banda
  - Es. rete locale aziendale, Università, etc.
  - Il proxy intercetta il traffico e mette in cache le pagine
  - Successive richieste non provocano lo scaricamento di ulteriori copie delle pagine al server
- **Reverse Proxy Caches**
  - Gateway cache
  - Operano per conto del server e consentono di ridurre il carico computazionale delle macchine.
  - I client non sono in grado di capire se le pagine arrivano dal server o dal gateway
  - Internet Caching Protocol per il coordinamento fra diverse cache. Base per le content delivery networks.

HTTP e Cache

- HTTP definisce vari meccanismi per la gestione delle cache
  - **Freshness**:
  - **Validation**: può essere usato per controllare se un elemento in cache è ancora corretto, per esempio, nel caso in cui sia in cache da molto tempo
  - **Invalidation**: è normalmente un effetto collaterale di altre request che hanno attraversato la cache. Se per esempio viene mandata una POST, una PUT o una DELETE ad una URL il contenuto della cache deve essere invalidato