**Appunti TSW:**

**URI (Uniform Resource Identifier):** forniscono un meccanismo semplice ed estensibile per identificare una risorsa. Con il termine **risorsa** intendiamo qualunque entità abbia una identità (come per esempio: documenti, immagini e servizi). In particolare, questo tipo di identificatore non fa per forza riferimento a risorse presenti in internet.

Proprietà:

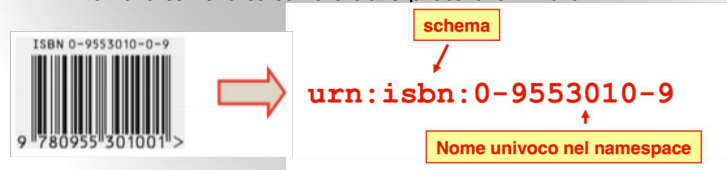
* **Uniformità:** gli identificatori sono uniformi, tale proprietà offre diversi vantaggi, essa ci permette infatti di utilizzare delle convenzioni sia sintattiche che semantiche comuni.

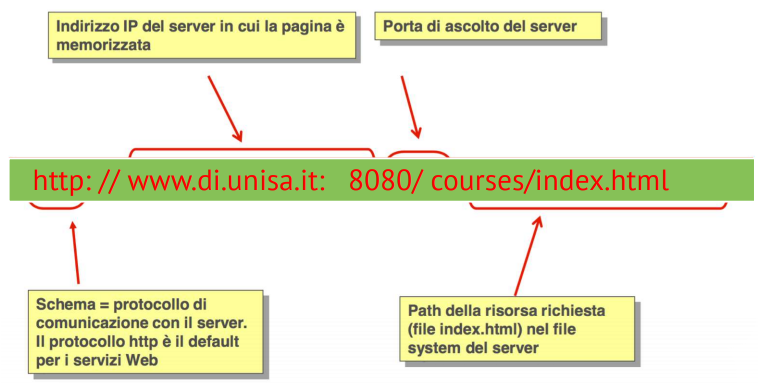
**Sintassi degli URI:** nel caso degli **URI** gli identificatori sono stringhe con una sintassi definita che può essere generalmente cosi presentata:

Esiste però un sottoinsieme di **URI** che utilizza la seguente sintassi:

Dove A parte **<scheme>,** le altre parti possono talora essere omesse, come nei casi in cui non è inclusa la componente **<authority>** o non è inclusa la componente **<query>.**

Esistono inoltre due specializzazioni del concetto di **URL:**

**Uniform Resource Name (URN):** identifica una risorsa per mezzo di un “**nome**” che deve essere globalmente unico e restare valido anche se la risorsa diventa non disponibile o cessa di esistere. Esempio:

**Uniform Resource Locator (URL):** identifica una risorsa per mezzo del suo meccanismo di accesso primario (es. locazione nella rete) piuttosto che sulla base del suo nome o dei suoi attributi.

Esempio:

Generalizzando, in uno schema **HTTP-like** l’**URL** utilizza la seguente sintassi:

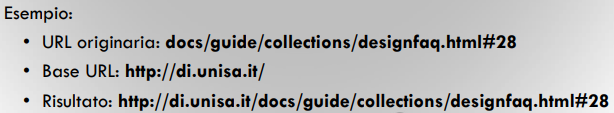
**<protocol>://[<username>:<password>@] <host>[:<port>] [/<path> [? <query>] [#fragment]]**

In particolare, i principali componenti di questo schema sono:

* **<protocol>:** Descrive il protocollo da utilizzare per l'accesso al server (HTTP, HTTPS, FTP, MMS, ...).
* **<username>:<password>@:** credenziali per l'autenticazione.
* **<host>:** indirizzo server su cui risiede la risorsa. Può essere un indirizzo IP logico o fisico (www.unisa.it o 193.205.160.20).
* **<port>:** definisce la porta da utilizzare (TCP come protocollo di trasporto per HTTP). Se non viene indicata, si usa porta standard per il protocollo specificato (per HTTP è 80).
* **<path>:** percorso (pathname) che identifica la risorsa nel file system del server. Se manca, tipicamente si accede alla risorsa predefinita (es. home page à index.html)
* **<query>:** una stringa di caratteri che consente di passare al server uno o più parametri. Di solito ha questo formato:
  + **parametro1=valore&parametro2=valore2...**
  + **fragment:** è una breve stringa di caratteri che si riferisce a una risorsa che è subordinata a un’altra risorsa primaria

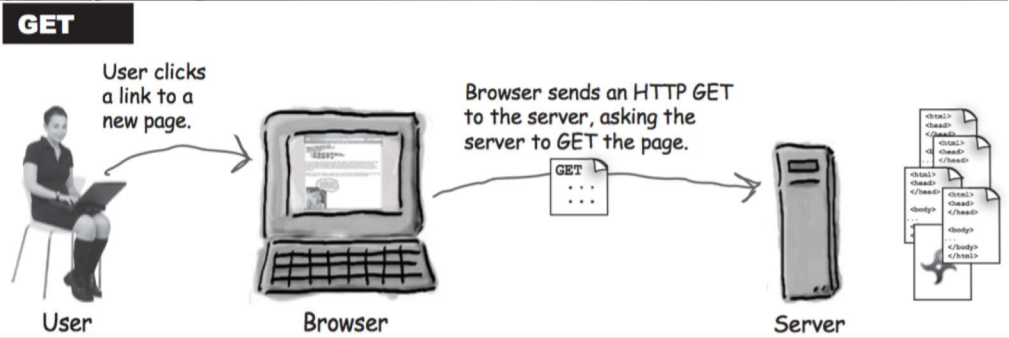
Un ulteriore classificazione che si fa sugli **URL** è la seguente:

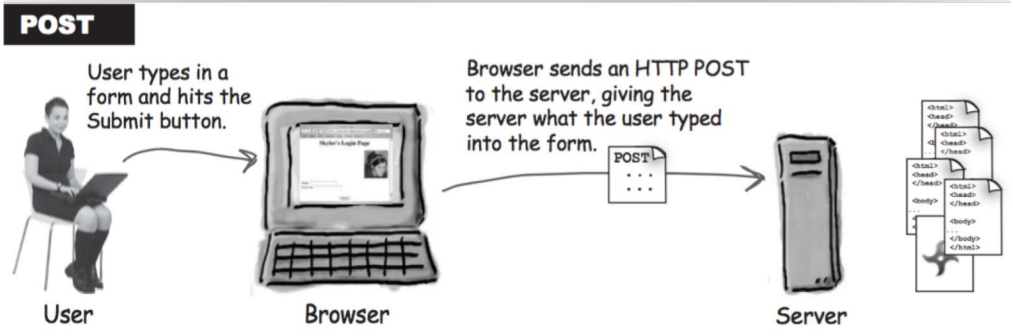
* **URL opaca:** non è soggetta a ulteriori operazioni di parsing.
  + **mailto:francese.rita@unisa.it**
* **URL gerarchica:** è soggetta a ulteriori operazioni di parsing, per esempio per separare l’indirizzo del server dal percorso all’interno file system.
  + **http://di.unisa.it/docs/guide/collections/designfaq.html#28../../../lab/examples/ant/build.xml** [**file:///~/calendar**](file:///~/calendar)
  + **Normalizzazione:** processo di rimozione dei segmenti "." e “. " (e altri caratteri speciali) dal path di una URI gerarchica. Si applica solo a URI gerarchiche, su URI opache non ha effetto.
  + **Risoluzione:** è il processo che a partire da una URI originaria porta all’ottenimento di una URI risultante. La URI originaria viene risolta basandosi su una terza URI, detta base URI.
  + **Relativizzazione:** è il processo inverso alla risoluzione.



**(HyperText Transfer Protocol) HTTP:** È il protocollo di livello applicativo utilizzato per trasferire le risorse Web (pagine o elementi di pagina) da server a client. Gestisce sia le richieste (URL) inviate al server che le risposte inviate al client (pagine). Principali terminologie di questo protocollo:

* **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 (Client à Server).
* **Response:** messaggio HTTP di risposta (Server à Client).
* **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, tipicamente di risposta.

**Metodo GET:** Serve per richiedere una risorsa ad un server. È 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. È previsto il passaggio di parametri (la parte <query> dell’URL) e la lunghezza massima di un URL è limitata.

**Metodo POST:** Progettato come il messaggio per richiedere una risorsa. A differenza di GET, i dettagli per identificazione ed elaborazione della risorsa stessa non sono nell’URL, ma sono contenuti nel body 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 sul server (lo vedremo presto...) Si ha una trasmissione di informazioni client à server che però non porta alla creazione di una risorsa sul server.

**Metodo 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 con lo stesso nome in seguito.

**Metodo DELETE:** Richiede la cancellazione della risorsa riferita dall’URL specificato.

**Metodo HEAD:** è simile al metodo GET, ma il server deve rispondere soltanto con gli header relativi, senza body.

* Viene usato per verificare un URL
* **Validità:** la risorsa esiste e non è di lunghezza zero
* **Accessibilità:** non è richiesta autenticazione

**Metodo OPTIONS:** serve per richiedere informazioni sulle opzioni disponibili per la comunicazione.

**Metodo TRACE:** è usato per invocare il loop-back remoto a livello applicativo del messaggio di richiesta

* Consente al client di vedere che cosa è stato ricevuto dal server: viene usato nella diagnostica e nel testing dei servizi Web.

**Codici di stato:** è un numero di tre cifre, di cui la prima indica la classe della risposta e le altre due la risposta specifica, esistono 5 classi:

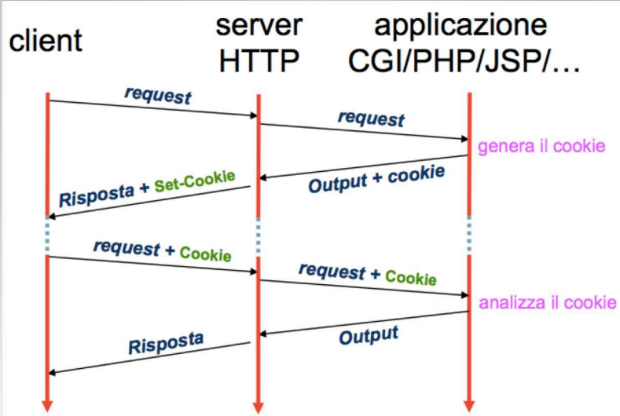
* **1xx: Informational:** Una risposta temporanea alla richiesta, durante il suo svolgimento (sconsigliata a partire da HTTP 1.0)
* **2xx: Successful:** Il server ha ricevuto, capito e accettato la richiesta
* **3xx: Redirection:** Il server ha ricevuto e capito la richiesta, ma sono necessarie altre 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 (suo o di applicazioni CGI (Common Gateway Interface))

Alcuni esempi:

* **100 Continue** (se il client non ha ancora mandato il body, deprecated da HTTPv1.0)
* **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**. Essi possono essere generati sia dal client che dal server e hanno scopo quello di fornire un supporto per il **mantenimento di stato** in un protocollo come HTTP che è essenzialmente **stateless**. Essi si presentano essenzialmente come una collezione di stringhe:

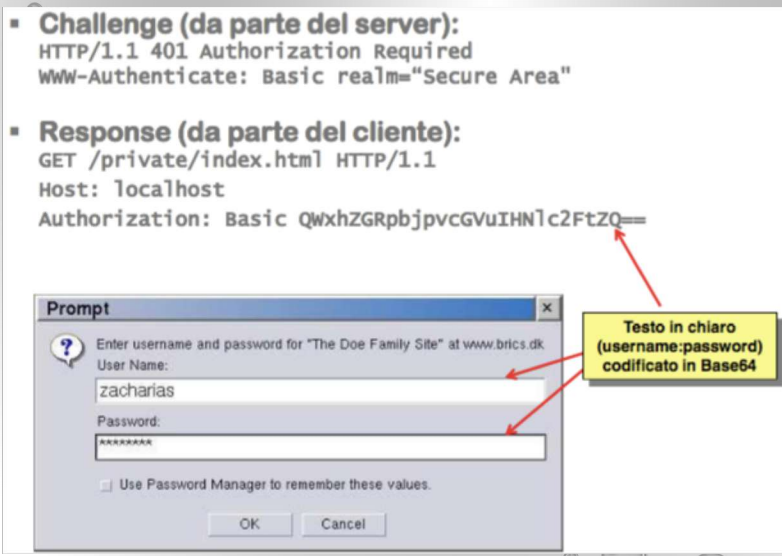
* **Key:** identifica univocamente un cookie all’interno di un **domain: path**
* **Value:** valore associato al cookie (è una stringa di max 255 caratteri)
* **Path:** posizione nell’albero di un sito al quale è associato (di default /)
* **Domain:** dominio dove è stato generato
* **Max-age:** (opzionale) numero di secondi di vita
* **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

I cookies usano due header, uno per la risposta, ed uno per richieste successive:

* **Set-Cookie:** header della risposta, il client può memorizzarlo e rispedirlo alla prossima richiesta.
* **Cookie:** header della richiesta. Il client decide se spedirlo sulla base del nome del documento, dell’indirizzo IP del server, e dell’età del cookie.

**Autenticazione:** Basarel’autenticazione sull’indirizzo IP del client è una soluzione che presenta vari svantaggi:

* Non funziona se l’indirizzo non è pubblico
* Non funziona se l’indirizzo IP è assegnato dinamicamente
* Esistono tecniche che consentono di presentarsi con un IP fasullo

Normalmente si usano

* **HTTP Basic**
* **FORM**

**Sicurezza:** proprietà desiderabili:

* **Confidenzialità**
* **Integrità**
* **Autenticità**
* **Non ripudio della sorgente:** prova chi è il mittente dei dati in una transazione.
* **Non ripudio della destinazione:** prova che i dati sono arrivati ad uno specifico destinatario.

Per gestire la sicurezza del canale di trasporto esistono due principali protocolli:

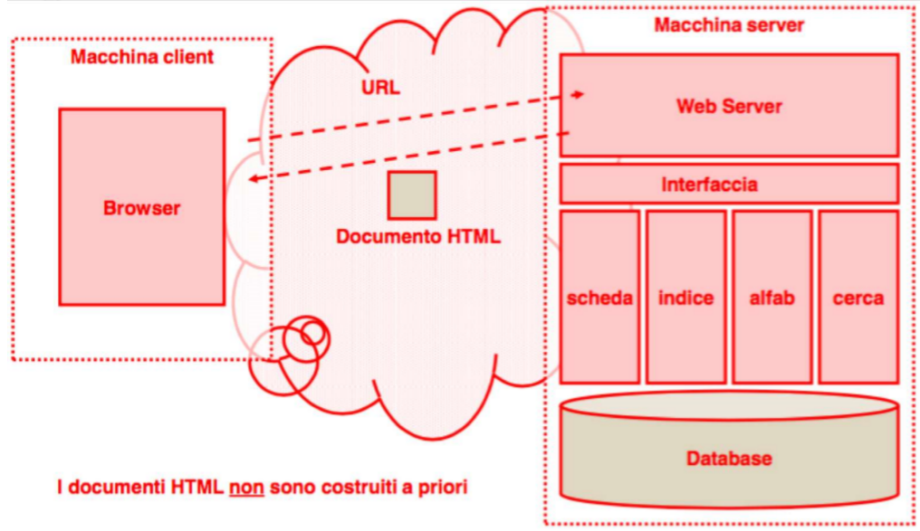
* **SSL (Secure Sockets Layer):** Viene aggiunto un livello che si occupa della gestione di confidenzialità, autenticità ed integrità della comunicazione fra HTTP e TCP. Si accede tramite https://... Basato su crittografia a chiave pubblica:
  + private key + public key
  + certificato (in genere usato per autenticare il server)
* **TLS (Trasporto Layer Security)**
  + Sostituisce **SSL**
  + È alla base di **HTTPS**

**User Agent Cache:** Lo user agent (tipicamente il browser) mantiene una cache delle pagine visitate dall’utente, Questo modello di caching è ora molto rilevante per i dispositivi mobili al fine di consentire agli utenti di lavorare con connettività intermittente.

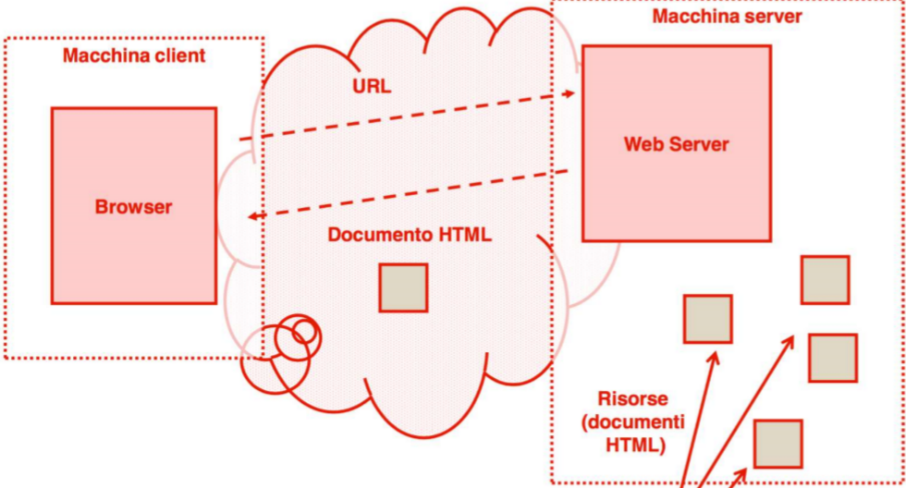
**HTTP e Cache:** HTTP definisce vari meccanismi per la gestione delle cache

* **Freshness:** controllata lato server da Expires response header e lato cliente da direttiva Cache-Control: max-age.
* **Validation**: può essere usato per controllare se un elemento in cache è ancora corretto, ad es. 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 a un URL il contenuto della cache deve essere e viene automaticamente invalidato.

**Differenza fra web statico e web dinamico:**

****Modello web dinamico:

Modello web statico:



**Stato delle applicazioni:** Esistono principalmente due tipi di iterazioni tra client e server:

* **Stateful:** esiste stato dell’interazione e quindi l’n-esimo messaggio può essere messo in relazione con gli n-1 precedenti. Esempio: autenticazione in un sito web.
* **Stateless:** non si tiene traccia dello stato, ogni messaggio è indipendente dagli altri.

**Diversi stati in un’applicazione web:**

* **Stato di esecuzione:** rappresenta un avanzamento in una esecuzione; per sua natura è uno stato volatile; può essere mantenuto in memoria lato server come stato di uno o più oggetti.
* **Stato di sessione:** la sessione viene gestita di solito in modo unificato attraverso l’uso di istanze di oggetti specifici (supporto a oggetti sessione).
* **Stato informativo persistente:** viene normalmente mantenuto in una struttura persistente come un database.

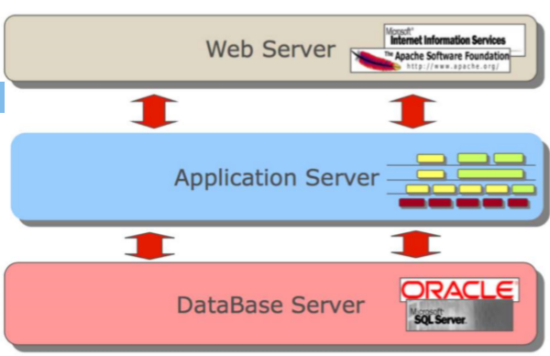
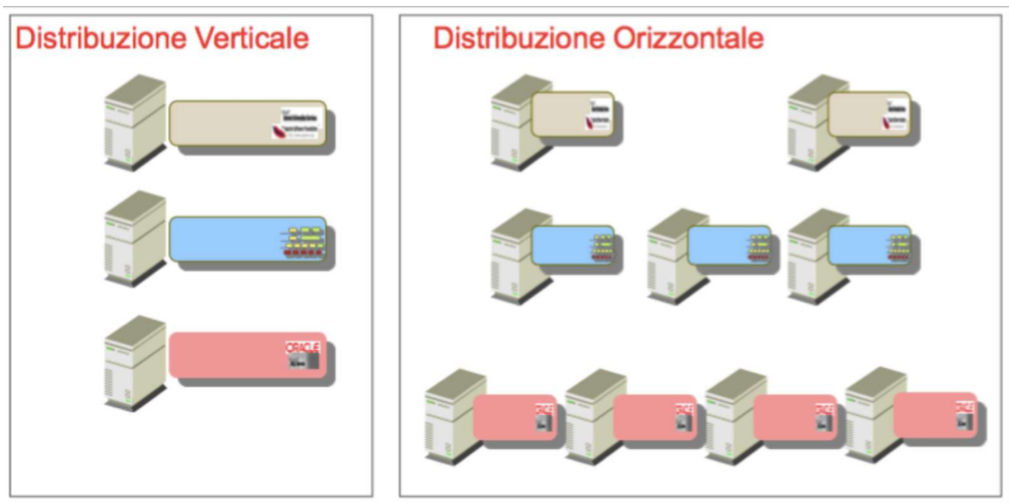
**Concetto di sessione:** La sessione rappresenta lo stato associato ad una sequenza di pagine visualizzate da un utente:

* Contiene tutte le informazioni necessarie durante l’esecuzione
  + **Informazioni di sistema**: IP di provenienza, lista delle pagine visualizzate, ...
  + **Informazioni di natura applicativa:** nome e cognome, username, quanti e quali prodotti ha inserito nel carrello per un acquisto, ...

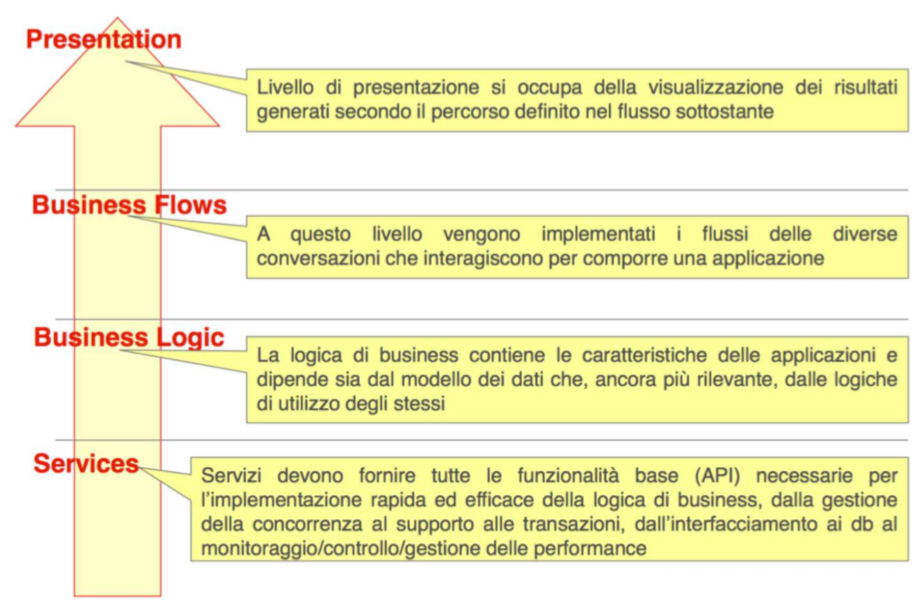
Lo scope di sessione è dato da:

* **Tempo di vita** della interazione utente (lifespan).
* **Accessibilità:** usualmente concesso alla richiesta corrente e a tutte le richieste successive provenienti dallo stesso processo browser.

La **conversazione** rappresenta una sequenza di pagine di senso compiuto (ad esempio l’insieme delle pagine necessarie per comperare un prodotto).

**Architettura a three-tier:** La suddivisione dell'applicazione in tre diversi moduli o strati dedicati rispettivamente alla **interfaccia utente**, alla **logica funzionale** e alla **gestione dei dati persistenti**.

Nel tempo si è affermata una classificazione indipendente dalla implementazione tecnologica, basata su una struttura a 4 livelli principali:



**Servizi:** I servizi realizzano le funzioni di base per sviluppo di applicazioni:

* Accesso e gestione risorse.
* Gestione transazioni.
* Gestione sicurezza.
* Accesso e gestione delle sorgenti dati.

**Business logic:** È l’insieme di tutte le funzioni offerte dall’applicazione

**Business flow:** Una conversazione è realizzata da un insieme di pagine collegate in un flusso di successive chiamate, Il **business flow** raccoglie l’insieme delle chiamate necessarie per realizzare una conversazione.

**Presentazione:** Come abbiamo già detto, business flow è in grado di fornire i dati di output necessari. Il livello di presentazione ha il compito di interpretare questi dati e generare **l’interfaccia grafica** per la visualizzazione dei contenuti (rendering).