Applicazioni web e cloud

appunti, a.a. 2021/2022

Area 1 - Web, HTML e CSS

Introduzione: World Wide Web

www = (World Wide Web) è definito da Tim Berners-Lee nel 1990 al CERN di Ginevra. E' l'applicazione internet più usata **open service**, è un insieme di **ipertesti** (**testo** con **link** e **media** come immagini, video e suoni, hanno una struttura non sequenziale) multimediali distribuiti su molti **nodi** (pagine web).

www non è internet.

www è distribuito e scalabile (è la capacità di mantenere prestazioni stabili anche nel caso di variazioni notevoli della mole o della tipologia dei dati trattati).

FTP = File Transfer Protocol, trasferimento di file usato prima di www.

L'architettura web si divide in:

- **client** = si connette al server, fa submit della richiesta, aspetta la risposta.
- **server** = programma "in ascolto" che fornisce un servizio. E' più potente del client perché deve gestire più traffico (più client contemporaneamente) e perché deve garantire privacy e sicurezza.

I concetti base che fanno parte del web sono:

- **URL** = Uniform Resource Locator, è il nome/path della pagina web.
- **URI** = Uniform Resource Identifier, sintassi universale per identificare risorse di rete, indipendente dal protocollo che in aggiunta a identificare una risorsa fornisce anche un meccanismo per localizzarla.
- **HTTP** = HyperText Transfer Protocol, è il canale di trasporto per spostare la richiesta dal client verso il server (e la risposta dal server verso il client).
- **HTML** = HyperText Markup Language.

La **sintassi URI** è:

scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

Nello specifico lo URI è composto da:

- scheme
- user:password = informazione utente
- host = nome del dominio a cui ci riferiamo
- port
- path
- query, #fragment = eventuali altre query (opzionali)

Esempio: foo://example.com:8042/over/there?name=ferret#nose

L'**URL** può assumere 2 diversi formati:

- **assoluto** = contiene una specifica completa dell'URL.
- **relativo** = omette l'indirizzo del server per ragioni di ottimizzazione.

HTTP

E’ un protocollo di trasporto semplice e stabile. E’ basato su TCP e viene usato tra browser e web server. Il suo funzionamento è diviso in 3 fasi principali:

1. **handshaking** = setup della connessione.
2. **scambio dati** = HTTP request e reply.
3. **chiusura della connessione**

HTTP è un protocollo del livello applicativo dello stack ISO/OSI (livello 7).
Non fornisce alcun tipo di affidabilità o ritrasmissione (queste caratteristiche vengono fornite dai protocolli di livello rete/network).
E’ un protocollo stateless (senza stato), il server non mantiene lo stato della connessione, non c’è collegamento tra due richieste consecutive (si aggiunge memoria tramite il meccanismo dei cookies).
Permette trasferimenti bidirezionali (es.: form), la negoziazione delle capability (l’utente può richiedere opzioni come il set di caratteri o la localizzazione), il supporto per il caching (memorizzazione delle risorse web per il loro riutilizzo) e i server proxy.

HTTP/1

La prima versione fu la 1.0 (ideato per la prima versione del web) e a seguire ci fu la 1.1 (va bene anche oggi). Quest'ultima versione viene migliorata nella 2 per questioni di performance.

I metodi di HTTP/1.0 sono:

- **GET** → per ottenere un documento (es.: GET URL HTTP/version number) (da terminale -X GET indirizzo)
- **POST** → si spedisce un contenuto/informazione (simile a GET, ma codifica gli input in modo diverso)
- **HEAD** → richiede solo informazioni dell’header (utile per controllare se un documento esiste e quanto è recente) (da terminale -I indirizzo)

Si aggiungono a http/1.1 i metodi:

- **PUT** → trasferisce un documento al server
- **DELETE** → elimina un documento dal server

Ci sono 3 distinzioni di header http (o riga di intestazione): richiesta, risposta o quelli applicabili a entrambi.
Formato **richiesta HTTP**:

1. riga richiesta
    - metodo
    - url
    - versione http che si sta usando
2. righe di intestazione
    - header file name
    - value

3. riga vuota
4. corpo entità

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

I campi della **header di richiesta** (o riga di intestazione della richiesta) possono essere:

- User-agent = versione del browser
- Referer = da dove proviene l'utente
- From = indirizzo email dell'utente (generalmente non usato per motivi di privacy)
- Authorization = può inviare il nome utente e password (usato con documenti che richiedono autorizzazione)
- If-Modified-Since = invia il documento solo se è più recente della data specificata (usato per memorizzare nella cache)

Formato **richiesta HTTP**:

1. riga di stato
    - versione
    - codice stato
    - frase
2. riga di intestazione
3. riga vuota
4. corpo entità

I campi <codice stato> e <frase> sono delle informazione che il server da al client, comunicando se è riuscito a portare a buon fine o meno la richiesta, a seconda del suo range:

- 200-299 → conferma la buona riuscita della richiesta (OK 200)
- 300-399 → contenuto ricercato non si trova (Moved 301=Il documento è stato spostato permanentemente. Not modified 304= La versione nella cache è aggiornata, il browser visualizza la versione della pagina già salvata in cache)

- 400-499 → la richiesta non va bene (Bad request 400=Errore di sintassi nella richiesta del client. Forbidden 403=Al cliente non è consentito l'accesso, cioè è protetto. Not found 404=Il file non è stato trovato)
- 500-509 → errore da parte del server (Internal error 500=Il server ha incontrato una condizione inaspettata. Service unavailable 503=Il server è sovraccarico)

L'**header di risposta** (o riga di intestazione della risposta) possono essere:
- Date = tempo di risposta
- Server = informazioni sull'identificazione del server
- Last-modified = ora in cui è stato modificato l'ultima volta
- Content-length = dimensione documento in byte
- Content-type = formato file
- Expires = evita al browser la memorizzazione di pagine nella cache oltre la data di scadenza

Ci sono due tipi di connessioni:
- **non persistenti** = usato con HTTP/1.0. Per ogni contenuto che si scambia si crea una nuova connessione.
- **persistenti** = usato con HTTP/1.1. La connessione rimane attiva e si possono scambiare più oggetti sulla stessa connessione. Le richieste vengono messe in pipeline e vengono mandate una dopo l'altra senza aspettare la risposta. La connessione può chiudersi in modo esplicito, sia da server che da client, mandando l'header di connection-close.

Nelle connessioni persistenti si risparmia tempo (non c'è troppa latenza) e risorse (continuo scambio di messaggi di stato) rispetto alle connessioni non persistenti. L'unico svantaggio delle connessioni persistenti è che bisogna identificare inizio e fine di ogni oggetto inviato sulla rete Internet. Ci sono 2 soluzioni:
- mandare una lunghezza seguita da un oggetto
- inviare un valore sentinella che identifica la fine di un oggetto

La gestione delle connessioni persistenti in HTTP/1.1 è però problematica in quanto:
- molti server non implementavano il pipelining (molta latenza)
- c'è comunque una sequenzialità dei messaggi

Con HTTP/2 si ha la possibilità di fare **multiplexing**, cioè spezzare il messaggio e mandare diversi pezzi in maniera non sequenziale. HTTP/1.1 implementa il **chunking**, prende un messaggio e lo fa a pezzetti.

virtual hosting = permette di dare accesso a diversi siti web memorizzati sulla stessa macchina server, questo è possibile perché il browser manda il nome del sito a cui connettersi come parte integrante della richiesta. I v.h. sono configurati e gestiti dall'amministratore del servizio, non dallo sviluppatore web.

negoziazione = l'header è usato per negoziare il formato della pagina, in termini di linguaggio, encoding e funzionamento/capability. La negoziazione può essere guidata sia da server (versione più efficiente) che da client.
Il browser può specificare rappresentazioni che sono accettabili con un valore di preferenza per ognuna esistono diversi Accept-header che corrispondono ai Content-headers:

- Accept-Encoding
- Accept-Charset
- Accept-Language

caching = tramite il browser si può fare cache dei contenuti visitati dall'end-system, in modo da facilitarne la fruizione. Se il contenuto non è nella cache viene generata la richiesta GET.

proxy server = è un servizio di un determinata comunità, scelto all'interno o vicino a tale comunità. Fa **caching** dei contenuti per tutta la comunità (caching a livello di organizzazione), cioè da qualunque end-system parta una richiesta per procurare un determinato contenuto web, il proxy server ne mantiene una copia per un certo periodo di tempo per tutti gli utenti. Il problema è stabilire questo periodo di tempo.
Il server stabilisce se una risorsa può essere salvata o meno in cache e il tempo massimo.

La cache è controllata dai response-header (header che danno informazioni riguardo l'utilizzo della cache):

- **Cache-control** = opzioni di gestione della cache come:
  - No-cache = indica che il contenuto non sarà inserito nella cache
  - Private = il contenuto può essere memorizzato solo per l'utente che ha fatto richiesta
  - Public = il contenuto è memorizzato sia nelle cache pubbliche che private.
- **Last-modified** = Permette di sapere l'ultima modifica della pagina che è stata acceduta Mantenuto nella cache e usato per controllare se la pagina deve essere ritrasferita
- **Expires Data** = di scadenza Il browser utilizza la pagina fino alla scadenza senza contattare il server

richiesta condizionale = La richiesta contiene un header con alcune condizioni . Se le condizioni non sono soddisfatte, il server non ritorna l'oggetto richiesto. Un esempio è l'header *if-Modified-Since*, che chiede se la pagina è stata modificata da una certa data e nel caso di non inviarla. Il server risponde con Not modified 304 o OK 200, e in quest'ultimo caso rinvia l'intera risorsa.

HTTP/2

I **problemi di HTTP/1.1** sono:

- **Concurrent connection limit** = (limite di connessioni concorrenti) c'è un limite al numero di connessioni che possono connettere uno stesso client a uno stesso

server, nonostante l'uso delle connessioni persistenti, questo per evitare una rapida saturazione il server con un numero basso di client. Per colpa di questo limite si ritorna a delle connessioni sequenziali, aumentando la latenza.

- **Head of line blocking** = A volte ci sono delle dipendenze tra oggetti che non possono essere spezzati. Si ritorna alla sequenzialità (HTTP/1.1 definisce il pipelining ma molti non lo applicano).
- **TCP slow start** = la connessione è inizialmente lenta e, poco a poco, aumenta la quantità di dati trasmessi, fino ad occupare la banda che ha a disposizione (è una partenza lenta, per evitare sovraccarichi della rete). Questa lentezza porta dei grandi svantaggi nel caso si debbano effettuare tante connessioni.
- **Latenza – page load time** = i ritardi si accumulano.

Per risolvere questi problemi nasce **HTTP/2 SPDY**, proposto da google nel 2012 come estensione di HTTP, per migliorare le performance. E' la base per HTTP/2 standard (HTTP/2 SPDY era usato per fare prove di affidabilità e funzionalità).

HTTP/2 implementa le seguenti funzionalità:

- Consente di utilizzare in modo **più efficiente** le risorse di rete e di ridurre la percezione della latenza. Introduce **compressione del campo header** e consente **più scambi simultanei** sulla stessa connessione.
- Pipelining, "head of line blocking", connessione singola vengono ottimizzati in una soluzione che consente di intervallare messaggi di richiesta e di risposta sulla stessa connessione (**multiplexing**) e utilizza una codifica efficiente per i campi di intestazione HTTP. Consente inoltre di **prioritizzare le richieste**, permettendo di completare più rapidamente le richieste più importanti, migliorando ulteriormente le prestazioni.
- Mantiene la semantica di HTTP/1.1 (Metodi, codici di stato, campi dell'header, URI).

Le novità di HTTP/2 hanno come risultato:

- Meno connessioni TCP in confronto a HTTP/1.0 e HTTP/1.1 (HTTP/1.x)
- Meno concorrenza con altri flussi di dati e connessioni a lungo termine
- Migliore utilizzazione della capacità di rete disponibile
- Elaborazione più efficiente dei messaggi tramite l'uso di framing dei messaggi binar

le caratteristiche di HTTP/2 sono:

- Usa una connessione singola e multiplex e quindi il numero massimo di connessioni per dominio può essere ignorato.
- Comprime i dati nello header e li invia in un formato binario compatto e quindi migliora il formato basato su testo usato in precedenza.
- Non necessita degli accorgimenti usati per migliorare le performance di HTTP/1.1.

binary framing layer = si cambia la codifica del dato. Si prendono i messaggi, si suddividono e si scambiano, non più come testo, ma come formato binario. Il binary framing layer si occupa di queste traduzioni. Si aggiungono i concetti di:

- **frame** = unità singola di comunicazione.
- **messaggio** = una sequenza completa di frame che mappano a una richiesta logica o un messaggio di risposta (un oggetto singolo).
- **stream** = flusso di dati da un client a un server e viceversa. All'interno possono passare più messaggi. Ogni singola connessione TCP contiene tutte le comunicazioni con un numero di stream qualsiasi.

multiplexing = ovvero mescolare i frame, anche se bisogna essere guidati da vincoli di priorità e dipendenza.

compressione header = l'header HTTP/1 introducono 500-800 byte di overhead per trasferimento che raggiunge più kilobyte con i cookie HTTP (tanti dati). In HTTP/2 introduce il formato di compressione HPACK = codifica dei campi header tramite **static Huffman encoding**, una compressione dei singoli valori. Il client e il server condividono un elenco indicizzato di header già scambiati, gli indici usati per codificare tali header a un nuovo invio. Come ulteriore ottimizzazione si usa una tabella **statica** (elenco di header HTTP comune che tutte le connessioni possono utilizzare (RFC7541)) e una **dinamica** (inizialmente vuota e aggiornata in base ai valori scambiati all'interno di una connessione).

server push = in HTTP/1 il server, dopo aver inviato l'HTML, aspetta richieste per altre risorse. In HTTP/2, il server fa push delle risorse comuni, tramite PUSH_PROMISE indicano l'intenzione del server di eseguire push delle risorse descritte al client e devono essere consegnate prima dei dati. La promessa manda solo l'header, se il client non dice niente nell'ACK che fa, il server manda la risorsa, ma se il client ha già la risorsa in cache può dire al server che non la deve inviare.

Confronto tra protocollo HTTP/1 e HTTP/2:

- HTTP/1 usa connessioni multiple (6 in parallelo)
- HTTP/2 usa una connessione

HTTP/3

usa QUIC, un protocollo a livello di "trasporto" per migliorare il multiplexing (native in QUIC) riducendo l'impatto dei pacchetti persi ai soli stream coinvolti (HTTP/3 vuole incrementare le prestazioni).

Il supporto attuale per HTTP/3 è più del 10% dei server.

HTML

Introduzione

HTML (HyperText Markup Language) non è un linguaggio di programmazione ma un linguaggio di **markup** (oltre a rendere il testo più leggibile, il markup permette anche di specificare ulteriori usi del testo).
Viene usato per la creazione di pagine web e descrive come la pagina deve essere **strutturata** (prima dell'avvento di CSS definiva anche come doveva essere visualizzata, poi si è capito che mettere tutto su un unico file era scomodo). **HTML ha poche e semplici regole sintattiche**.

Il browser ha come obiettivo interpretare la pagina, non solo colorando il testo e applicando gli stylesheets, ma anche eseguendo degli script JavaScript.

Il browser è composto da diversi engine:

- **Layout Engine** = (detto anche Browser Engine) riceve input dal browser (URL bar, search box, mouse clicks and key presses) e li passa al rendering engine.
- **Rendering Engine** = riceve il codice HTML e lo interpreta visualizzandolo a video. Ad esempio, testo in bold.
- **User Interface** = controlli del browser, ad esempio, bottoni per andare avanti e indietro, bookmark, etc..
- **JavaScript Engine** = engine che si prende cura di parsare ed eseguire codice JavaScript per poi ritornare il risultato dell'esecuzione.
- **Network Layer** = gestisce funzioni di rete, ad esempio, criptazione, richieste http e tutte le configurazioni di rete

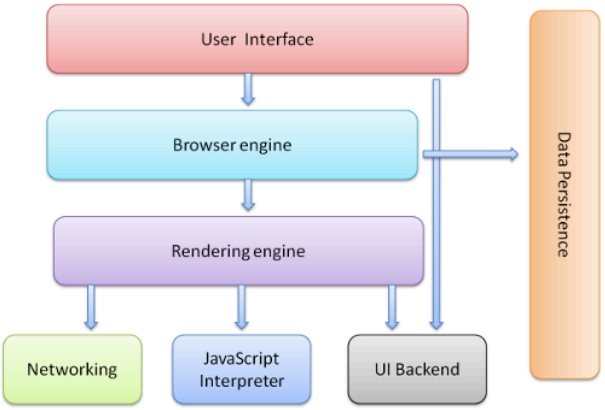


Figure : Browser components

- **Storage** = porzione di memoria dove il browser memorizza cached file, cookie e oggetti e dati create con JavaScript
- **Operating System Interface** = componente che interagisce con il sistema operativo per disegnare diversi elementi e per gestire la finestra (chiudi, massimizza, minimizza)

Gli sviluppatori devono sviluppare web in modalità **multi-browser**, cioè il sito web deve essere visualizzabile nella stessa maniera nei principali browser.

XHTML è un HTML (non molto usato) **dove le regole sintattiche diventano importanti** (se c'è un errore in XHTML l'intera pagina non viene visualizzata). Era nato dal fatto che molte pagine scritte in HTML erano scritte in modo scorretto (spreco di risorse).

Struttura della pagina HTML

HTML è formato da **tag**, dei marcatori che permettono di identificare delle porzioni di testo. Su queste porzioni di testo possiamo imporre delle regole di visualizzazione. Il nome del tag è collegato alla sua funzione. Un tag è contenuto, insieme ad eventuali attributi, tra parentesi triangolari. C'è sempre un tag di inizio e un tag di fine, quest'ultimo contraddistinto dal carattere "/" dopo "<".

```
 <tag attributi>contenuto</tag>
 <font color="blue">testo</font>
```

Il contenuto è quella porzione di testo su cui viene applicata la funzione del tag.

```
<img src="miaImmagine.jpg">
```

Questo è un tag di tipo immagine senza contenuto, al posto di questo tag il browser deve visualizzare "miaImmagine.jpg" (spesso al posto di un file nel file system viene inserito un link).

La pagina HTML ha una struttura del tipo:

- **testa** = header che contiene una descrizione dei metadati della pagina. Contiene informazioni non immediate, descrive come il documento deve essere letto e interpretato. Contiene meta-tag, script, stili ed è racchiuso tra il tag <head>.
- **corpo** = contenuto vero e proprio. Contiene tutti i tag per la realizzazione del sito Web (ad esempio <font ...>) ed è racchiuso tra il tag <body>.

Tutto parte con una riga di intestazione.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//IT">
```

Indica le specifiche W3C adottate per lo sviluppo della pagina.

il tag <html> indica l'inizio di una pagina HTML (i tag sono annidati, ovvero uno dentro l'altro).

```
<html>
  <head> <!--contiene meta-tag, script e stili-->
  </head>
  <body> Hello                    ciao <br>
 World! </body>
</html>
```

HTML e case unsesitive e non è sensibile agli spazi e alle linee vuote, per andare a capo esiste un tag apposito
 (XHTML è case sensitive).

Testa <head>

Contiene informazioni/metadati relativi al documento e non al suo contenuto (ad esempio, titolo, keyword, altri dati non considerati contenuto del documento). I browser di

solito non presentano elementi che compaiono nella testa (HEAD) se non il titolo che viene usato come nome dalla scheda/finestra.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//IT">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html"; charset="iso-8859-1">
    <!-- Scriveremo qui -->
    <title>Pagina di prova</title>
  </head>
  <body> Hello World! </body>
</html>
```

<title> = Indica il titolo della pagina
<meta> = Definisce informazioni relative alla pagina. Contiene tre attributi principali:

- **Name** = contiene il nome della proprietà: Author, Description, Copyright, Generator, Language, Keywords.
- **content** = contiene il valore della proprietà.
- **http-equiv** = specifica le caratteristiche del HTTP response header.

Esempio:

```
<meta name="author" content="Greta">
```

I meta tag keywords sono le chiavi di ricerca e servono per indicizzare le pagine.

```
<meta name="keywords" content="calcio soccer tennis
basket pallacanestro pallavolo volley inter milan juventus
roma lazio...">
```

L'attributo HTTP-EQUIV contiene informazioni sulla comunicazione server-browser e può essere usato al posto dell'attributo name. Fornisce un header HTTP per informazioni/valore dell'attributo content. Può essere usato per simulare un HTTP response header e gli HTTP server usano questo attributo per ottenere informazioni circa l'HTTP response header.

```
<meta http-equiv="Content-Type" content="text/html; charset=iso- 8859-1">
```

Corpo <body>

Contiene il contenuto vero e proprio. Il browser presenta il contenuto in diversi modi. Contiene tutte le informazioni per la visualizzazione della pagina.
Alcuni elementi HTML nel body sono:

- **block-level** = impongono una spaziatura tra loro(elemento block-level) e il testo.
- **inline** = (text level) testo classico che fluisce in maniera sequenziale.

Content model:

- Elementi block-level possono contenere elementi inline e altri elementi block-level
- Elementi inline contengono solo dati e altri elementi inline

Formatting:

- Elementi block-level sono formattati in maniera differente dagli elementi inline
- Elementi block-level iniziano su una linea nuova, elementi inline no

Tag contenitori di testo definisce degli elementi block-level (unica eccezione) che hanno un comportamento predefinito.

```
<h1>titolo</h1><h2>sottotitolo</h2>
```

Tag che usano font differenti in base al numero di header, sono previste sei grandezze predefinite per i titoli, si va da <h1> che è la grandezza maggiore fino ad <h6> che è la grandezza minore. Manda a capo il contenuto successivo.

```
<p>paragrafo</p>
```

Unità base di suddivisione del testo. Lascia una riga vuota prima e dopo il testo.

```
<div>blocco1</div>
```

Blocco contenitore. Serve per dare una struttura alla pagina web. Non ha una struttura predefinita. Blocco testo va a capo ma non lascia righe di spazio.

```
<span>contenitore1</span>
```

Contenitore generico per applicare stylesheet. Come non ha una struttura predefinita. E' un elemento inline, continua sulla stessa riga del tag che lo contiene (ad es.) e non va a capo.

```
<h2>Titolo della pagina</h2>
<p>paragrafo 1</p>
<p>paragrafo 2</p>
<div>blocco 1</div>
<div>blocco 2  <b>ciao<b>
  <span>contenitore 1</span>
  <span>contenitore 2</span>
</div>
```

Ci sono due tipi di fogli di stile (tutti elementi inline):

- **Fisici** = definiscono lo stile grafico.
  - <b> bold: formatta testo in grassetto
  - <i> italic: formatta il testo in corsivo
  - <u> underline: sottolinea il testo
  - <strike>: testo barrato (usato per correzioni)
- **Logici** = forniscono informazioni sul ruolo svolto dal testo (opzionale: impongono uno stile grafico).

- <sup> apice: E=mc² (utili per la scrittura di formule matematiche)
- <sub> pedice: H₂0
- <abbr> abbreviazione: non comporta nessun cambiamento grafico
- <address> indirizzo
- <cite> citazioni: testo in corsivo
- <samp> esempio: testo a spaziatura fissa

Font = colore, dimensione e tipo di carattere

```
<font face="verdana, arial, helvetica, sans- serif ">testo</font>
```

L'attributo face è il tipo di font che si va ad utilizzare e sono messi in ordine di preferenza.
L'attributo color è il colore è può essere espresso: con il suo nome, in HEX o in RGB (es.: aqua/#00ffff/0,255,255).

```
<font color="#00FFFF" face="calibri">testo aqua</font>
```

L'attributo size indica la dimensione del font. Ci sono diversi modi per impostare la dimensione:

- Valore tra 1 e 7

```
<font size="3">Dimensione 3</font>
```

- Valori relativi alla dimensione base del font

```
  <font size="+2">Dimensione +2</font>
```

- %di pixel, "num"em dove em è l'unità di misura che dipende dall'ambiente e dalla font size originale.

La sintassi di un elenco è:
<elenco>
    <elemento> primo elemento
    <elemento> secondo elemento
</elenco>
Ci sono tre tipi di elenchi:

- **ordinati** =
    - tag elenco ordinato <ol> (ordered list)
    - tag elemento <li> (list item)
- **non ordinati** = bullet list
    - tabella tag <ul> (unordered list)
    - tag elemento <li> (list item)
- **di definizioni** =
    - elenco di definizione <dl> (definition list)
    - termine da definire <dt> (definition term)
    - definizione del termine <dd> (definition description)

Ipertestualità (Link)

Insieme non lineare di documenti con informazioni di varia natura (testi, immagini, brani musicali, filmati), collegati l'uno all'altro per mezzo di connessioni logiche e rimandi (link) che consentono all'utente di costruire di volta in volta un autonomo percorso di lettura.
I link sono formati da due componenti:

- il contenuto (testo o immagine) che nasconde il **collegamento**
- la **risorsa puntata**

```
Clicca <a href="indirizzo"> qui </a> per visualizzare il collegamento.
```

L'indirizzo/destinazione può essere di diverso tipo (.gif, .jpeg, .doc, .pdf, .zip, .exe, un indirizzo email...).
Un indirizzo ha diversi attributi:

- <body link="red"> cambia il colore dei **link**
- <body vlink="green"> cambia il colore dei **link visited**
- <body alink="yellow"> cambia il colore dei **link attivi**

Ci sono due tipi di percorsi per i link:

- **assoluti** = viene indicato il percorso per esteso

```
Leggi l'esempio <a href="http://www.miosito.it/cartella/miofile.html"> qui </a>
Leggi l'esempio <a href="c:\cartella\miofile.html"> qui </a>
```

- **relativi** = fanno riferimento alla posizione del file a cui si riferisce al link a partire dalla posizione della pagina che stiamo sviluppando

```
<a href="link.html">Clicca qui</a>  <!--Indica al browser di cercare il file
nella stessa directory -->
<a href="sottodir/sottolink.html">Clicca qui</a>  <!--Indica al browser di
cercare il file nella sottodirectory sottodir-->
<a href="../superlink.html">Clicca qui</a>  <!--.. Indica al browser di cercare
il file nella directory padre-->
```

Ci possono essere anche link interni (ancore):

- Ancora

```
 <a name="primo"> àncora </a> <!--si identifica l'oggetto con un nome-->
```

- Riferimento all'àncora

```
<a href="#primo"> vai all'àncora </a>
```

- Riferimento ad inizio pagina

```
  <a href="#"> torna su </a>
```

Le immagini hanno una sintassi tipo:

```
<img src="pic.jpg" alt="description" width="128" height="128">
```

dove gli attributi sono:

- src: sorgente del file (percorso di memorizzazione file)
- alt: descrizione mostrata al passaggio sull'immagine

- width e height: dimensioni immagine

Tabelle (table)

Una tabella è definita tramite l'elemento <table> e ogni riga è definita con il tag <tr> contenuto in table. Ogni cella è definita con il tag <td> contenuto in <tr>.
Una riga può essere anche divisa in table heading usando l'elemento <th>. Di solito sono mostrati centrati e in grassetto.
Elemento <caption> per definire la caption della tabella.

```
<table>
  <tr>
    <th>Company</th>
    <th>Contact</th>
    <th>Country</th>
  </tr>
  <tr>
    <td>Alfreds Futterkiste</td>
    <td>Maria Anders</td>
    <td>Germany</td>
  </tr>
  <tr>
    <td>Centro comercial Moctezuma</td>
    <td>Francisco Chang</td>
    <td>Mexico</td>
  </tr>
</table>
```

Attributo border = (Se non specificato nessun bordo)

```
<table border="1" style="width:100%">
```

Per fondere due celle o due colonne in un'unica sola si usano gli attributi rowspan e colspan.

Form

Form usate per collezionare input utente, è basato sull'elemento <form>. Specifica diversi elementi come: input, checkbox, radio button, submit button, etc.
Nel form l'elemento input è il più importante, ci sono diverse varianti che dipendono dall'attributo **type**:

- **Text** = definisce input testo normale
- **Radio** = definisce un input radio button (per selezionare sola una scelta tra diverse opzioni)
- **Submit** = definisce un bottone submit (per sottomettere la form)

```
<input type="text">
```

Definisce un campo testo di una linea

```
<input type="password">
```
un tipo speciale di campo testo dove il valore inserito viene nascosto

```
<form>
First name:<br>
<input type="text" name="firstname">
<br>
Last name:<br>
<input type="text" name="lastname">
</form>
```

```
<textarea>
<textarea name="message" rows="10" cols="30">
The cat was playing in the garden.
</textarea>
```
Definisce un campo testo a linee multiple

```
<input type="radio">
<form>
<input type="radio" name="sex" value="male" checked> Male
<br>
<input type="radio" name="sex" value="female"> Female
</form>
```
Definisce un radio button che permette una scelta tra diverse opzioni

```
<input type="checkbox">
<form>
 <input type="checkbox" name="vehicle1" value="Bike"> I have a bike
 <br>
 <input type="checkbox" name="vehicle2" value="Car"> I have a car
 </form>
```
Definisce una serie di checkbox che permettono una scelta multipla tra diverse opzioni

Drop-down list con il tag <select>. Definisce un menù a tendina, l'elemento <option> definisce delle possibili opzioni.

```
<select name="cars">
<option value="volvo">Volvo</option>
<option value="saab">Saab</option>
<option value="fiat" selected>Fiat</option>
<option value="audi">Audi</option>
</select>
```

Submit button con il tag <input type="submit">. Definisce un bottone per sottomettere la form ad un **handler** = una pagina lato server che processa i dati della form, è definito con l'attributo action.

```
<form action="action_page.php">
First name:<br>
<input type="text" name="firstname" value="Mickey">
<br>
Last name:<br>
<input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
 </form>
```

Permette anche di eseguire semplici azioni come:

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

Action attribute = definisce le azioni che devono essere fatte quando la form è inviata. La sottomissione viene fatta solitamente con il **botton submit**. La pagina viene inviata a una pagina web lato server. Ad esempio uno script lato server gestisce la form inviata

```
<form action="action_page.php">
```

Se l'attributo action è omesso si considera la pagina corrente di default.

Method attribute = Specifica l'operazione HTTP da usare per inviare la form.
Le operazioni possono essere:

- **GET** = (default) usata per invio di form passive (ad es., search engine query) e senza informazioni sensibili (esempio: I dati inviati tramite la form sono visibili all'indirizzo *action_page.php?firstname=Mickey&lastname=Mouse*). GET ottima per piccole quantità di dati. GET è usata per invio di form che aggiornano dati.
- **POST** = fornisce maggiore sicurezza perchè I dati non sono visibili nell'indirizzo (ad es., password)

```
<form action="action_page.php" method="get">
<form action="action_page.php" method="post">
```

Name attribute è necessario perché ogni campo input deve avere un nome.

```
<form action="action_page.php">
 First name:<br>
<input type="text" value="Mickey">
 <br>
 Last name:<br>
<input type="text" name="lastname" value="Mouse">
 <br><br>
<input type="submit" value="Submit">
</form>
```

Fieldset raggruppa campi in una form tramite elemento <fieldset>. Si può definire una caption per la form <legend>.

```
<form action="action_page.php">
 <fieldset>
<legend>Personal information:</legend>
 First name:<br> <input type="text" name="firstname" value="Mickey">
 <br>
 Last name:<br> <input type="text" name="lastname" value="Mouse">
<br><br>
<input type="submit" value="Submit">
</fieldset>
</form>
```

Attributo value specifica il valore iniziale.

```
<input type="text" name="firstname" value="John">
```

Attributo readonly specifica che il valore non può essere cambiato.

```
<input type="text" name="firstname" value="John" readonly>
```

Attributo disabled specifica un campo disabilitato e non utilizzabile (non viene inviato).

```
<input type="text" name="firstname" value="John" disabled>
```

Attributo size specifica la dimensione in caratteri del campo

```
<input type="text" name="firstname" value="John" size="40">
```

Attributo maxlength specifica la dimensione massima del campo

```
<input type="text" name="firstname" maxlength="10">
```

XHTML

XHTML = **EX**tensible **H**yper**T**ext **M**arkup **L**anguage. XHTML è quasi identico a HTML ma è più stringente di HTML. XHTML è supportato dai principali browser.
Molte pagine web contengono HTML errato, ma viene visualizzato bene nella maggior parte dei casi anche se non segue le regole HTML.

Diversi browser che eseguono su diversi device
Per i device più piccoli spesso è difficile interpretare un markup sbagliato a causa di mancanza di risorse o potenza computazionale
XML è un linguaggio di markup dove i documenti devo essere ben formati (markup corretto).
XHTML combina HTML e XML, XHTML è HTML ridisegnato come XML, XTHML è definito per essere visualizzabile dai browser per HTML

Struttura documento
XHTML DOCTYPE è obbligatorio
L'attributo xmlns in <html> è obbligatorio
<html>, <head>, <title>, e <body> sono obbligatori

Gli Elementi XHTML devono :

- essere innestati correttamente
- sempre essere chiusi
- essere minuscoli
- avere un'unico elemento radice

Attributi XHTML
Nomi di attributi devono essere minuscoli
Valori di attributi devono essere quoted
Minimizzazione degli attributi vietata

Come convertire HTML in X-HTML:

- Aggiungere un XHTML <!DOCTYPE> nella prima linea di ogni pagina
- Aggiungere un attributo xmlns all'elemento html di ogni pagina
- Cambiare tutti i nomi di elemento in minuscolo
- Chiudere tutti gli elementi vuoti
- Cambiare tutti i nomi di attributo in minuscolo
- Quotare tutti i valori di attributi

W3C fornisce un meccanismo per la validazione delle pagine web http://validator.w3.org

CSS

Introduzione

CSS = **C**ascading **S**tyle **S**heet (fogli di stile a cascata). Impone delle regole sulla visualizzazione della pagina. Quando con HTML 3.2 sono stati introdotti elementi tipo <font>, lo sviluppo di pagine web è diventato un incubo per gli sviluppatori, viene personalizzata ogni singola pagina che aggiunge complessità e costi, andando contro il principio della **scalabilità**. Serve una separazione netta tra struttura/contenuto e stile di visualizzazione/formattazione.
I fogli di stile servono per una gestione del sito facilitata infatti:

- riducono il carico di lavoro
- permettono di controllare il layout di pagine web multiple in un colpo solo

Il risultato sono pagine più leggere e facili da modificare (milioni di byte di banda risparmiati).

Meccanismi, costrutti e selettori

Una regola CSS è composta di due parti:

- h1= **selettore**, definisce la parte del documento cui verrà applicata la regola
- {...} = **blocco delle dichiarazioni**, formato da coppie:
  - **proprietà** = (seguito da ":") definiscono un aspetto dell'elemento da modificare (margini, colore di sfondo, …) secondo il valore espresso
  - **valore** (seguito da ";")

```
h1 {
  color: white;
  background: red;
}
```

Per ogni elemento è possibile definire una sintassi singola

```
div {
margin-top: 10px;
margin-right: 5px;
margin-bottom: 10px;
margin-left: 5px;
}
```

e una abbreviata

```
div {margin: 10px 5px 10px 5px;}
```

Alcune combinazioni di selettori possono essere:
elemento1 elemento2: seleziona tutti gli elemento2 contenuti (**tutti** i discendenti) in elemento1

```
ul li { background-color: yellow; }
```

elemento1 > elemento2: seleziona tutti gli elemento2 che hanno come **padre** un elemento1

```
div > p { background-color: yellow; }
```

elemento1 + elemento2: seleziona tutti gli elemento2 che sono piazzati immediatamente dopo un elemento1

```
div + p { background-color: yellow; }
```

Alcune selettori di attributi:
[attribute]: seleziona tutti gli elementi con attributo attribute

```
a[target] { background-color: yellow; }
```

[attribute=value]: seleziona tutti gli elementi con la coppia (attribute,value)

```
a[target=_blank] { background-color: yellow; }
```

[attribute~=value]: seleziona tutti gli elementi che hanno un attributo il cui valore è una sequenza di parole separate da spazio e una delle parole è value

```
img[title~="xyz"] { background-color: yellow; }
```

[attribute|=value]: uguale al precedente tranne per il fatto che il valore dell'attributo è una sequenza di parole separate da "–"

Si possono fare anche raggruppamenti e passare da

```
h1 {background: white;}
h2 {background: white;}
h3 {background: white;}
```

a

```
h1, h2, h3 {background: white;}
```

C'è anche un selettore universale con il carattere "*" per tutti gli elementi.

I fogli di stile possono essere esterni, interni e in linea. Nello specifico:

- È **esterno** un foglio di stile definito in un file separato dal documento (editabili anche con il Blocco Note con estensione .css)

```
<link rel="stylesheet" type="text/css" href="stile.css">
```

oppure

```
<style> @import url(stile.css); </style>
```

- È **interno** un foglio di stile il cui codice è compreso in quello del documento HTML

```
...<head>
<style type="text/css">
body { background: #FFFFCC; }
</style>
</head>...
```

- È **in linea** quando lo stile è interno al tag HTML target

```
<h1 style="color: red; background: black;">...</h1>
```

CSS in linea sovrascrive CSS interno, CSS interno sovrascrive CSS esterno.

Attributi <link>:

- **rel** = tipo di relazione tra documento e file collegato (obbligatorio) due possibili valori: stylesheet e alternate stylesheet
- **href** = definisce l'URL assoluto o relativo del foglio di stile (obbligatorio)
- **type** = identifica il tipo di dati da collegare: text/css (obbligatorio)
- **media** = supporto (schermo, stampa, etc) cui applicare un particolare foglio di stile (opzionale) (ad es., screen)

```
<html>
<head>
 <link href="stile.css" type="text/css" rel="stylesheet">
</head>
<body> ... </body>
</html>
```

Esistono dei **selettori speciali**:

- Class, usato per un set di elementi
- Id, usato per selezionare uno specifico elemento

I due selettori devono essere associati ad una stylesheet altrimenti perdono di significato (si possono combinare ricorsivamente class e id).

selettore class = Selettori che non mappano direttamente a uno specifico tag (ad esempio cambiare il colore di una parola all'interno di un tag paragrafo). Il nome del selettore class è preceduto da "."

```
<style type="text/css">
.testorosso {
font: 12px Arial, Helvetica, sans-serif;
color: #FF0000;
}
</style>
<p class="testorosso">  Il testo all'interno del paragrafo è colorato di rosso
</p>
```

Esiste un secondo tipo di sintassi (più restrittivo):
<elemento>.nome_della_classe

```
p.testorosso {color: red;}
```

lo stile verrà applicato solo ai paragrafi che presentino l'attributo class="testorosso".

Classi multiple:

```
p.testorosso.grassetto {color:red; font-weight:bold;}
```

Regola applicherà a tutti gli elementi in cui siano presenti (in qualunque ordine) i nomi delle classi definiti, aumenta la difficoltà di comprensione della pagina.
Avranno dunque il testo rosso e in grassetto il paragrafo ... ma non questo ... .

Pseudo classi = usate per definire lo stato di un certo elemento.

- **:active** è usato per selezionare e impostare lo stile di elementi «attivi», lo stile cambia al click del mouse. Usato soprattutto con i link (anche se non esclusivo), per i link si usano.
- **:link** per pagine non visitate,
- **:visited** per pagine visitate,

- **:hover** quando il mouse passa sopra il link

```
p:active, a:active { background-color: yellow; }
a:hover { background-color: yellow; }
```

selettore id = Identificano lo stile di un singolo elemento all'interno della pagina HTML. Usato quando si è sicuri che quell'elemento comparirà **una sola volta**. Il nome del selettore è preceduto da "#"

```
<style type="text/css">
#testorosso {
font: 12px Arial, Helvetica, sans-serif;
color: #FF0000;
} </style>
  <p id="testorosso"> Il testo all'interno del paragrafo è colorato di rosso
</p>
```

Pseudo elementi = Usate per specificare parti di un elemento

:after o ::after inserisce del contenuto con un certo stile dopo un elemento

```
p::after { content: " - Remember this"; }
```

:before o ::before inserisce del contenuto con un certo stile prima di un elemento

```
p::before { content: " Remember this - "; }
```

:first-child seleziona l'elemento solo se primo figlio

```
 p:first-child { background-color: yellow; }
```

:first-letter seleziona la prima lettera dell'elemento

```
p::first-letter { font-size: 200%; color: #8A2BE2; }
```

::first-line seleziona la prima linea dell'elemento

```
p::first-line { background-color: yellow; }
```

Un elemento contenuto in un altro (child) eredita tutte le proprietà di stile dal suo elemento padre. Tuttavia, se il figlio (o discendente) ha proprietà in conflitto con le proprietà definite dall'elemento padre, il conflitto viene risolto in favore dell'elemento figlio (la proprietà più specifica vince).

Ereditarietà esplicita = Associare a una proprietà il valore **inherit** per indicare che la proprietà ereditarà il valore dall'elemento padre. Sintassi "proprietà: inherit".

```
 span { color: blue; border: 1px solid black; }
```

```
.extra span { color: inherit; }
```

Regole CSS - **Valutazione a cascata**
User agent deve applicare un algoritmo per stabilire i valori di una combinazione di elementi e proprietà:

- Per prima cosa bisogna selezionare tutte le dichiarazioni che applicano a un elemento/proprietà per il media type richiesto. Le dichiarazioni si applicano se il selettore è consistente con l'elemento considerato. Il Media type definisce lo stile in base al tipo di media scelto.

```
@media only screen and (max-width: 600px) {
 body { background-color: lightblue; }
}
```

1. Il primo ordinamento è fatto in base a **peso e origine.** Per dichiarazioni normali, gli style sheet dell'autore della pagina sovrascrivono gli style sheet dell'utente che sovrascrivono quelli di default.
2. Il secondo ordinamento è fatto in base alla **specificity**.
3. Il terzo ordinamento è fatto in base **all'ordine**.

Cosa succede se nessuna dichiarazione applica a una proprietà di un elemento? Generalmente, valore ereditati dall'elemento antenato più vicino che ha un valore per quella proprietà. Se nessun antenato ha un valore (o la proprietà non supporta ereditarietà), il CSS definisce un valore iniziale che viene usato

Formattare testo con CSS

Nel design di pagine web si può formattare testo in maniera simile alle applicazioni di word processing. Necessità di specificare un font che è disponibile nel browser lato utente, altrimenti si avranno problemi di visualizzazione (il browser sostituisce il font con un altro). I font disponibili su (quasi) tutti i browser sono: arial, verdana, georgia, times new roman, courier, trebuchet, lucida, tahoma, impact.

font stack = la soluzione per la mancanza di font comuni è l'utilizzo di alternative font, il font stack definisce un insieme di font che possono essere utilizzati per stampare il testo a video
font-family:"Helvetica Neue", Helvetica, Arial, serif;
Serif permette di usare qualunque font serif come times new roman, georgia

font-size = Definisce la dimensione del testo (importante perché i monitor hanno risoluzione diversa.
Ci sono diverse opzioni di definizione come:

- **Absolute-size** = un insieme di keyword che definiscono dimensioni predefinite
  Scala di dimensione crescente in accordo alle preferenze utente: xx-small, x-small, small, medium, large, x-large, xx-large.

```
body {
```

```
font-family:"Trebuchet MS", Tahoma, Arial, sans-serif;
font-size:small;
}
```

- **Length** = un numero seguito da una unità di misura:
  - assoluta (cm, mm, in, pt, pc) Point pt non va bene per il web (è fatto soprattutto per stampanti e si applica male a monitor)
  - relativa (em, ex, px) Pixel px sembrano perfetti (unità di misura dei monitor e loro grafica) Alcuni browser non ridimensionano caratteri in pixel se sovrascrivono i valori di default

```
body {
font-family:"Trebuchet MS", Tahoma, Arial, sans-serif;
font-size:10px;
}
```

- **Percentage** = un intero seguito da un "%".Il valore è la percentuale della dimensione del font dell'oggetto padre. Definisce la dimensione come la dimensione di default del browser

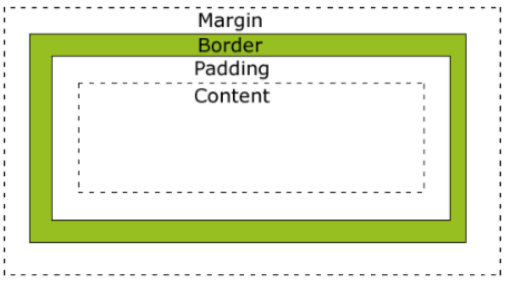
```
body {
font-family:"Trebuchet MS", Tahoma, Arial, sans-serif;
font-size:100%;
}
```

- **Relative size** = Un insieme di keyword interpretate come relative alla dimensione del font dell'oggetto padre. Possibili valori includono larger e smaller.

Property	Possible values
`font-style`	`normal` (initial value), `italic` (more cursive than normal), or `oblique` (more slanted than normal).
`font-weight`	`bold` or `normal` (initial value) are standard values, although other values can be used with font families having multiple gradations of boldness (see CSS2 [W3C-CSS-2.0] for details).
`font-variant`	`small-caps`, which displays lowercase characters using uppercase glyphs (small uppercase glyphs if possible), or `normal` (initial value)

CSS box model = Tutti gli elementi HTML sono delle box. Il CSS box model racchiude ogni elemento in un box e vi associa:

- **Content** = il contenuto della box, dove testo e immagini risiedono
- **Padding** = libera un'area attorno al content (È trasparente)
- **Border** = un bordo che circonda padding e content



- **Margin** = libera un'area attorno al bordo (È trasparente)

border = border-* può avere da uno a 4 valori:

- **border-style: dotted solid double dashed**; top border è dotted right border è solid bottom border è double left border è dashed
- **border-style: dotted solid double**; top border è dotted right and left borders sono solid bottom border è double
- **border-style: dotted solid**; top and bottom borders sono dotted right and left borders sono solid
- **border-style: dotted**; Tutti e quattro borders sono dotted

"Border" può essere utilizzato come attributo scorciatoia. Racchiude in una definizione: border-width, border-style (required) e border-color

```
p{ border: 5px solid red; }
```

margin = Definisce lo spazio tra gli elementi (cioè lo spazio vuoto **fuori** dai bordi). Top, right, bottom, e left margin possono essere cambiati indipendentemente. Una scorciatoia con l'attributo margin può essere usata per cambiare i margini in una volta sola (al posto di usare i 4 possibili valori dell'attributo border).

```
p{
margin-top: 100px;
 margin-bottom: 100px;
margin-right: 150px;
margin-left: 50px;
}
p{ margin: 100px 50px; }
```

padding = Definisce lo spazio tra gli elementi border e content (cioè lo spazio vuoto **all'interno** dei bordi). Top, right, bottom, e left margin possono essere cambiati indipendentemente. Una scorciatoia con l'attributo padding può essere usata per cambiare i margini in una volta sola (al posto di usare i 4 possibili valori dell'attributo padding).

```
p{
padding-top: 25px;
padding-right: 50px;
padding-bottom: 25px;
padding-left: 50px;
}
p{ padding: 25px 50px;}
```

```
<!DOCTYPE html>
<html> <head>
<style>
div {
background-color: lightgrey;
```

```
width: 300px;
padding: 25px;
border: 25px solid navy;
margin: 25px;
}
</style>
</head>
<body> <div>TESTO</div> </body>
</html>
```

Line-height = L'altezza della line box è data da line-height. Definisce spaziatura tra linee di testo. Il valore iniziale è normal (cioè cell height).

Font-weight = stabilisce lo spessore dei caratteri (es.: font-weight:normal; font-weight:lighter;)
Text-transform = agisce e converte i caratteri (es.: text-transform:uppercase;)
Letter-spacing = definisce la spaziatura tra i caratteri del testo (es.: letter-spacing:0.2em)

Property	Values
`text-decoration`	`none` (initial value), `underline`, `overline`, `line-through`, or space-separated list of values other than `none`.
`letter-spacing`	`normal` (initial value) or a length representing additional space to be included between adjacent letters in words. Negative value indicates space to be removed.
`word-spacing`	`normal` (initial value) or a length representing additional space to be included between adjacent words. Negative value indicates space to be removed.
`text-transform`	`none` (initial value), `capitalize` (capitalizes first letter of each word), `uppercase` (converts all text to uppercase), `lowercase` (converts all text to lowercase).
`text-indent`	length (initial value 0) or percentage of box width, possibly negative. Specify for block elements and table cells to indent text within first line box.
`text-align`	`left` (initial value for left-to-right contexts), `right`, `center`, or `justified`. Specify for block elements and table cells.
`white-space`	`normal` (initial value), `pre`. Use to indicate whether or not white space should be retained.

css e liste html = Si possono modificare indentazione, spaziature usando margin e padding, si possono customizzare i bullet e si possono modificare i colori dello sfondo del carattere.

Flow processing

= il modo in cui gli oggetti sono in relazione l'uno con l'altro.
In **normal flow processing**[1], ogni elemento ha una box corrispondente. L'elemento HTML si riferisce a una box chiamata **initial containing block** (corrisponde all'intero documento).
Box degli elementi figli sono contenuti in box degli elementi padri
Elementi **block** = fratelli sono messi uno sopra l'altro.
Elementi **inline** = fratelli sono messi uno di fianco all'altro.

[1] Normal Flow - Learn web development | MDN

Proprietà **display** = Controlla il layout e dice se e come un elemento deve essere mostrato. Valore di default block o inline, si può selezionare none.

block = Un elemento con proprietà display e valore block inizia sempre su una riga nuova. Lo style sheet dello user agent (non è CSS) specifica valori di default.
Elementi block tradizionali includono: html, body, p, pre, div, form, ol, ul, dl, hr, h1-h6. Molti degli altri elementi eccetto li e quelli relative alle tabelle hanno la proprietà display con valore inline.
Quando elementi block sono impilati, i margini adiacenti sono collassati nel margine più grande.
Valore iniziale della proprietà width è auto. Elementi block occupano l'area di contenuto più larga possibile. Tutto fatto rispettando margini e padding.
Possibile specificare la lunghezza usando CSS width. Possibile specificare la lunghezza come percentuale del contenuto dell'elemento padre

inline = Box che corrispondono a celle di caratteri o elementi inline (display: inline) sono messe uno in fianco alle altre in line box. Elementi inline iniziano sempre sulla stessa riga e prendono la larghezza necessaria. Line box sono messe una sopra le altre.
Padding/border/margin influenzano la larghezza ma non l'altezza delle box inline.
Per posizionare elementi inline all'interno delle line box si usa la proprietà vertical-align.

Valore display none indica che l'elemento **non** deve essere mostrato. L'elemento è nascosto e la pagina visualizzata come se non ci fosse. L'elemento e i suoi discendenti non vengono mostrati. Non influenza il normal flow.
Visibility = significa che gli elementi non vengono visualizzati ma ci sono, influenza il normal flow.

proprietà **position**[2] = specifica il tipo di posizionamento che può essere:

- **static** = valore di default. Elemento posizionato in base al normal flow.L'elemento div ha position static.

```
div.static {
position: static;
border: 3px solid #8AC007;
}
```

- **fixed** = elemento posizionato nello stesso punto anche quando la pagina viene scorsa. Proprietà top, bottom, left, right usate per posizionare l'elemento.

```
div.fixed {
position: fixed;
bottom: 0;
right: 0;
width: 300px;
border: 3px solid #8AC007;
}
```

[2] CSS Positioning and Normal Flow

- **relative** = permette agli elementi di essere posizionati al di fuori del normal flow. Posizionamento relativo alla posizione tradizionale (quella con position: static).
- **absolute** = Posizionamento box relativo alla posizione dell'elemento antenato (quello con position: relative). Se non esistono antenati si considera il body e l'elemento mantiene la posizione quando la pagina viene scorsa.

Proprietà top, bottom, right, left hanno risultati diversi in base al valore di position.

z-index = specifica la gestione di elementi che si sovrappongono: chi sta sopra e chi sta sotto. Si possono definire valori positivi e negative.

Proprietà **float** = Permette di posizionare testo intorno a un'immagine (es.: text wrap o runaround). CSS ottiene questo effetto permettendo a elementi che seguono un elemento float in HTML di circondare l'elemento, cambiando la loro posizione. Float permette anche di creare pagine a colonne assumendo valore left, right, none.
Quando un elemento è annotato con float è tolto dal normale flusso HTML, quindi spesso possono esserci errori di visualizzazione se non si sta attenti. Una soluzione è quella di forzare i due elementi parenti uno a sinistra e l'altro a destra. Nessuna dipendenza dall'ordine dei tag HTML nel body della pagina. Come alternative:

- Forzarli entrambi a sinistra in catena (stesso risultato)
- Indicare il primo elemento con float:right (oppure float:left)

Proprietà **clear** = Se si vuole forzare un elemento a stare da solo senza elementi a lato si usa la proprietà clear. Specifica su quale lato di un elemento non possono essere visualizzati elementi float. Si possono evitare elementi:

- A sinistra clear: left
- A destra clear: right
- A sinistra e destra clear: both

Originariamente griglie di elementi erano realizzate con i float. Supporto per resize automatico rispetto alla dimensione del browser.
Display: **inline-block** = Come elemento inline ma con height e width.

Struttura della pagina

Come definire la struttura della pagina? Originariamente veniva usata la struttura tabella di HTML Tag <table>. Ai tempi era l'unico modo possibile, spesso si usavano tabelle innestate l'una nell'altra. Difficile la manutenzione e la possibilità di modificare il layout.

Area 2 - JavaScript

Introduzione

Risorsa in rete identificata tramite URL è di tre tipi:

- **Pagina statica** (HTML) il cui contenuto è fisso, definito nel momento in cui la pagina è stata scritta (es.: http://www.example.com/biblioteca.html)
- **Pagina "dinamica"** (per esempio PHP, ASP, o JSP) il cui contenuto viene generato (selezionato, composto) al momento della richiesta (o della visualizzazione) (es.: http://www.example.com/biblioteca.php)
- **Indirizzo di un programma** (ad es., Java Servlet) il cui compito è quello di generare dinamicamente una pagina Web, in base alla richiesta del client (es.: http://www.di.unimi.it/ardagna/exServlet)

Applicazione web = software sviluppato e utilizzato attraverso tecnologie web come:

- linguaggi di:
    - **mark-up** = descrivono documenti strutturati (contenuto + struttura + aspetto/formattazione). Producono pagine web statiche interpretate dal browser. (es.: HTML e CSS)
    - **scripting** = tipo particolare di linguaggio di programmazione (es.: PHP, JavaScript), per scrivere **script** = un piccolo programma che inseriamo nella pagina web.
    - **programmazione** = descrivono programmi come sequenza di istruzioni.
- tecnologie client-side e server-side

I linguaggi di programmazione e scripting sono formati da:

- Programma = insieme di istruzioni
- Linguaggi di programmazione = linguaggi per scrivere programmi (istruzioni) composto da:
    - Alfabeto = insieme di simboli con cui si possono costruire i termini del linguaggio (lessico = keywords" del linguaggio)
    - Frasi ben formate = istruzioni (programmi)
    - Sintassi = definita da una grammatica che fornisce le regole di composizione dei termini in frasi ben formate del linguaggio
    - Semantica = definisce il significato delle frasi ben formate del linguaggio (esecuzione del programma)
- Analizzatore sintattico (parser) = analizza frasi e decide se sono frasi ben formate del linguaggio o no

Il programmatore scrive il programma sorgente utilizzando un linguaggio "ad alto livello". Occorre tradurre il programma sorgente in un programma composto da istruzioni in linguaggio macchina (programma eseguibile). Tecniche per effettuare questa traduzione:

- Compilazione: traduzione effettuata da strumenti chiamati compilatori (ad es., C)
- Interpretazione: traduzione effettuata da strumenti chiamati interpreti (ad es., JavaScript, Ruby)
- Approccio misto (ad es., Java, Python)

Traduzione eseguita da **compilatore** = Specifica per una data macchina. Tradurre il programma sorgente nel linguaggio macchina della macchina specifica. Genera un file eseguibile (.exe)

- Vantaggi: efficiente (esecuzione veloce)
- Svantaggi: poco flessibile (per eseguire il programma su macchine diverse è necessario ri-compilare i sorgenti)

Traduzione eseguita da **interprete** = Interpreta un programma (traduzione simultanea). Ogni istruzione viene tradotta in un insieme di istruzioni nel linguaggio macchina della piattaforma specifica ed eseguita Interprete specifico per la piattaforma

- Vantaggi: flessibile (il sorgente è direttamente eseguibile su macchine diverse, dato l'interprete)
- Svantaggi: poco efficiente (esecuzione lenta)

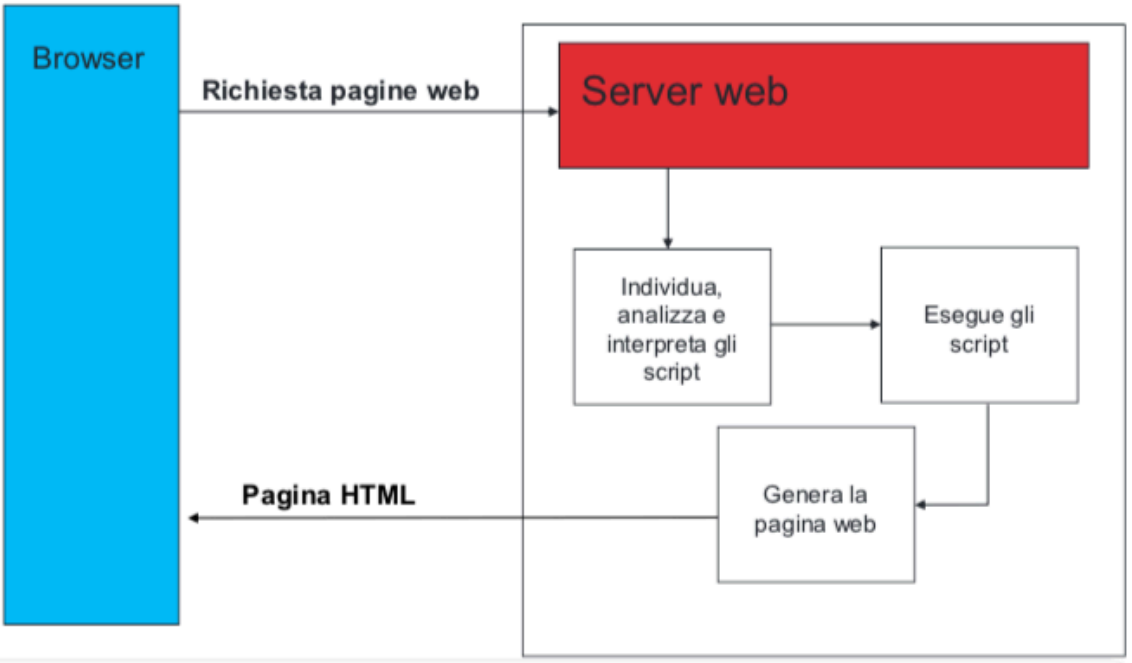
Approccio misto = Programma sorgente tradotto in formato intermedio (bytecode) indipendente dalla macchina (compilatore) Programma in formato intermedio viene interpretato (interprete)
L'approccio misto somma i vantaggi dei due approcci:

- Efficiente (il linguaggio intermedio è “molto vicino” al linguaggio macchina e la sua interpretazione è veloce)
- Flessibile (perché posso eseguire il programma compilato su macchine diverse)

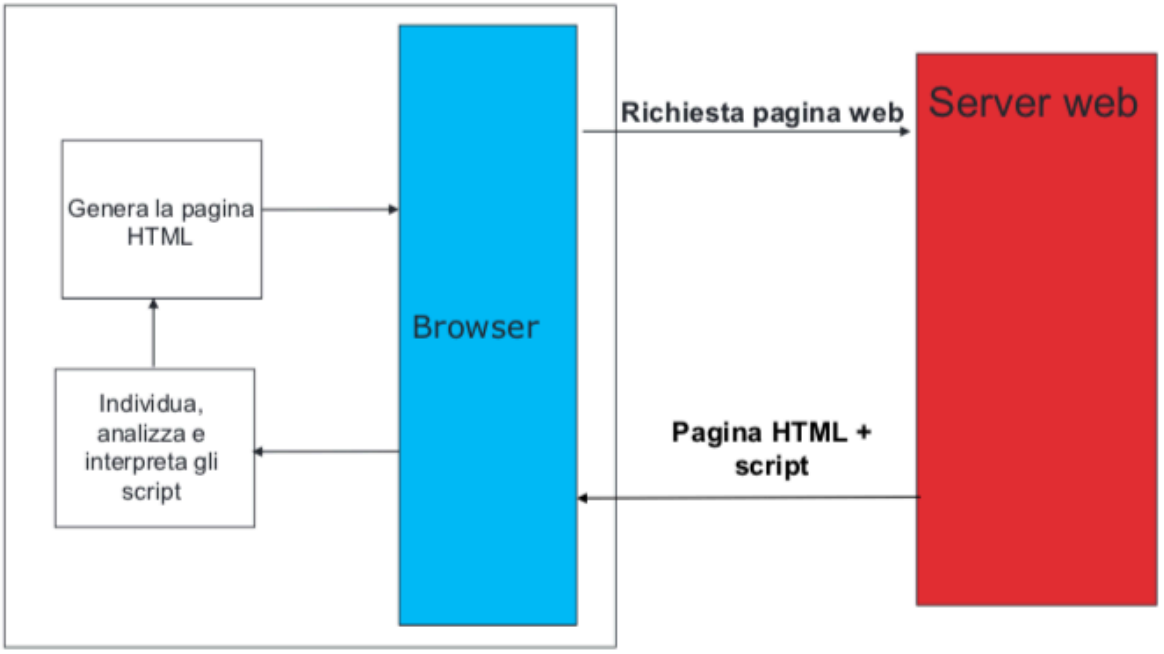
Pagine Web "dinamiche" = Il contenuto viene generato al momento della richiesta/visualizzazione. Si basano su linguaggi di programmazione e scripting e possono essere formate da tecnologie:

- **client-side** (pagine "debolmente" dinamiche). Elaborazione lato client (browser). Scripting client-side (JavaScript), Java Applet, Adobe Flash.
- **server-side** (pagine "autenticamente" dinamiche). Elaborazione lato server (web server). Scripting server-side (PHP, Active Server Pages, Java Server Page), programmi (Java Servlet), Ruby, Python, Perl, (JavaScript)

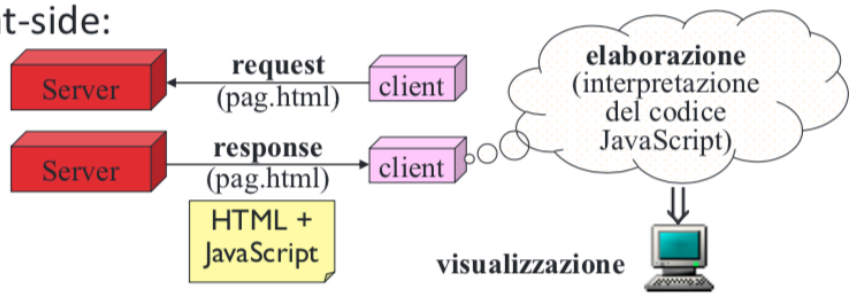
Architettura server-side



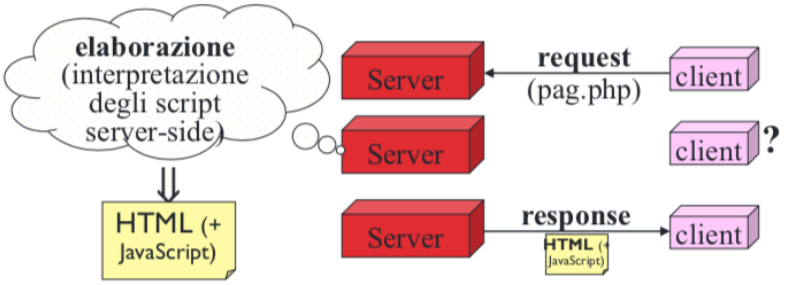
Architettura client-side



- Client-side:



- Server-side:



Si sviluppano su tre livelli logici:

- Presentazione → HTML, CSS
- Intermedio → JavaServlet, JSP, PHP, ASP, JavaScript, Flash (XSLT)
- Dati → RDBMS, XML, JSON

Fondamenti

JavaScript è un linguaggio di **scripting** non un linguaggio di programmazione.

Nonostante la somiglianza nel nome, è un linguaggio completamente distinto da Java. Come tutti i linguaggi di scripting, è interpretato, il sorgente **non deve essere compilato** per essere eseguito, l'interprete di JavaScript è generalmente contenuto all'interno del **browser**.

A differenza di HTML, JavaScript è case-sensitive. Purtroppo possono esserci incompatibilità e differenze tra i diversi browser.
Si basa su due concetti principali:

- DOM (Document Object Model)
- Eventi (script event-driven)

JavaScript è un **linguaggio debolmente tipato** Il tipo delle variabili (e dei parametri/argomenti delle funzioni) non viene dichiarato esplicitamente, ma definito implicitamente al primo assegnamento
JavaScript converte automaticamente i tipi durante l'esecuzione (quando possibile)
Tipi principali in JavaScript:

- **Number** = interi e decimali (virgola mobile); ad es: 7, 7.7
- **Boolean** = valori booleani (vero/falso); ad es: true, false
- **String** = sequenze di caratteri; ad es: "ciao

Non è obbligatorio l'uso della keyword var quando si dichiarano le variabili, sono senza tipo. È obbligatorio l'uso della keyword var per le dichiarazioni delle variabili locali. Le inizializzazioni sono facoltative (ma è buona norma farle all'atto della definizione e prima di utilizzare le variabili). Tutte le istruzioni JavaScript devono terminare con ";".

loosely typed = è possibile assegnare a una stessa variabile prima un valore stringa, poi un numero, poi altro ancora. Ad esempio:
alfa = 10 beta = "Claudio" alfa = "Erika" /* tipo diverso.*/
Sono consentiti incrementi, decrementi e operatori di assegnamento estesi (++, --, +=, …).

Due possibili scope:

- **Globale**, per le variabili definite fuori da funzioni
- **Locale**, per le variabili definite esplicitamente dentro a funzioni (compresi i parametri ricevuti)

Un blocco non delimita uno scope! Tutte le variabili definite fuori da funzioni, anche se dentro a blocchi innestati, sono globali.

Operatore typeof ritorna il tipo di una espressione.
typeof(10/2) = number
Il tipo è **dinamico**, rappresenta il tipo in quel momento temporale e corrispondente al valore attuale della variabile (o dell'oggetto…).
Le istruzioni sono separate da una fine riga o da un punto e virgola.

Le istruzioni devono essere separate da un fine riga o da un punto e virgola (simile al Pascal). Ad esempio:
Istruzione1 // fine riga
istruzione2; istruzione3
istruzione4

Le costanti numeriche sono sequenze di caratteri numerici non racchiuse da virgolette o apici (tipo number). Le costanti booleane sono true e false (tipo boolean). Altre costanti sono null, NaN e undefined (indica un valore indefinito).

I commenti in JavaScript sono come in Java:
// commento su riga singola
/* commento su + righe*/

Gli operatori javascript possono essere:

- Aritmetici → +, -, *, /, ++, --
- Di confronto → ==, != (numeri e stringhe) >, >=, <, <= (numeri) === (fa anche il controllo del tipo)
- Booleani: && (AND), || (OR), ! (NOT)

- Concatenazione (di stringhe): +
- Assegnamento: =

Le espressioni sono simili a quelle Java:
- Espressioni numeriche: somma, sottrazione, prodotto, divisione (sempre fra reali), modulo, shift, ...
- Espressioni condizionali con ? ... :
- Espressioni stringa: concatenazione con +
- Espressioni di assegnamento: con =

Esempi:
- document.write(a/b)
- document.write(a%b)
- document.write("a" + 'b')

condizioni booleane = Una condizione booleana è un'espressione che ha valore vero (true) o falso (false). Le condizioni booleane sono espressioni composte da: costanti, variabili, operatori di confronto e operatori logici.

Stringhe = delimitate sia da virgolette sia da apici singoli. Per annidare virgolette e apici, occorre alternarli. Concatenazione con + document.write("paolino" + 'paperino'). Concatenazione fra stringhe e numeri comporta la conversione automatica del valore numerico in stringa. Le stringhe JavaScript sono oggetti dotati di proprietà (ad es., length) e metodi (ad es., substring(first,last)).

Il codice di un programma JavaScript viene incluso in un file HTML per mezzo del tag <SCRIPT language="JavaScript">. Sono possibili più programmi nella stessa pagina, ovvero una pagina HTML può contenere più tag <script>. L'interprete HTML lo invia all'interprete JavaScript.
Le definizioni delle funzioni vengono generalmente incluse nella sezione <head>, ma viene anche incluso nel campo <body>.
I richiami alle funzioni (predefinite nel linguaggio o definite dal programmatore) avvengono dove occorre, nel body della pagina HTML.

Costrutto if-else = per esprimere un'azione condizionale
if (condizione) { sequenza di azioni }

Liste = sequenza ordinata di elementi (es.: elenco di link in una pagina Web, elenco degli iscritti alle liste elettorali). Le operazioni sulle liste sono:
- Calcolo lunghezza
- Stampa di tutti gli elementi
- Aggiunta di un elemento
- Cancellazione di un elemento

Array = lista rappresentata come array con indice a partire da 0. Le celle di un array JavaScript non hanno il vincolo di omogeneità in tipo: ogni cella può contenere indistinta-mente numeri, stringhe, oggetti, altri array, etc.
Creazione e inizializzazione di un array:

- Array vuoto
  - var lista = new Array();
  - lista[0] = "Claudio"; ...
- Array inizializzato in fase di definizione
  - lista = new Array("Claudio", "Erika", "Denise");
- Elencare la sequenza, racchiusa fra parentesi quadre, di valori iniziali separati da virgole
  - vett = [1, -2, "tre"]

Inserimento valori: lista[i] = "stringa";
Lettura contenuto in posizione i: var elem = lista[i];

Lunghezza di in un array (attributo length) var lunghezza = lista.length;
Per (sovra)scrivere (Ogni scrittura sovrascrive l'elemento che era memorizzato in precedenza) l'ultimo elemento dell'array: lista[lunghezza-1]="stringa";
Per leggere l'ultimo elemento dell'array: var elem = lista[lunghezza-1];

cicli = Strutture di controllo per l'accesso sequenziale a un set di elementi (ad es., lista). I cicli sono generalmente basati sul concetto di indice che scorre lungo la lista indicando, via via, posizioni successive. JavaScript supporta for, while, do/while, for... in..., with.

ciclo for = è composto da:

- Inizio = la posizione iniziale dell'indice
- Test = condizione che vera (true) fa sì che il ciclo prosegua; falsa (false) provoca l'uscita dal ciclo
- Incremento = incremento dell'indice ad ogni ciclo

Istruzioni eseguite a partire da inizio, finché il test è vero, avanzando ad ogni passo di quanto è indicato da incremento.
for (inizio; test; incremento) { istruzioni }

ciclo while = Finché la condizione è vera esegui le istruzioni.
while (condizione) { istruzioni }

I cicli for e while si equivalgono, per scorrere una lista si può usare l'uno o l'altro. Generalmente più semplice il for, il while è più versatile e può essere usato per scopi diversi dalla gestione delle liste (ad esempio quando non si conosce il numero di cicli).

Oggetti DOM, eventi, finestre, nodi

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."
È uno standard W3C (World Wide Web Consortium) che definisce uno **standard per accedere documenti**.
Separato in tre parti:

- Core DOM – modello standard per tutti i tipi di documenti
- XML DOM – modello standard per documenti XML
- HTML DOM – modello standard per documenti HTML

HTML DOM = È uno standard object model e programming interface per HTML. Definisce:

- Elementi HTML come oggetti
- Proprietà degli elementi HTML
- I metodi per accedere agli elementi HTML
- Gli eventi per tutti gli elementi HTML

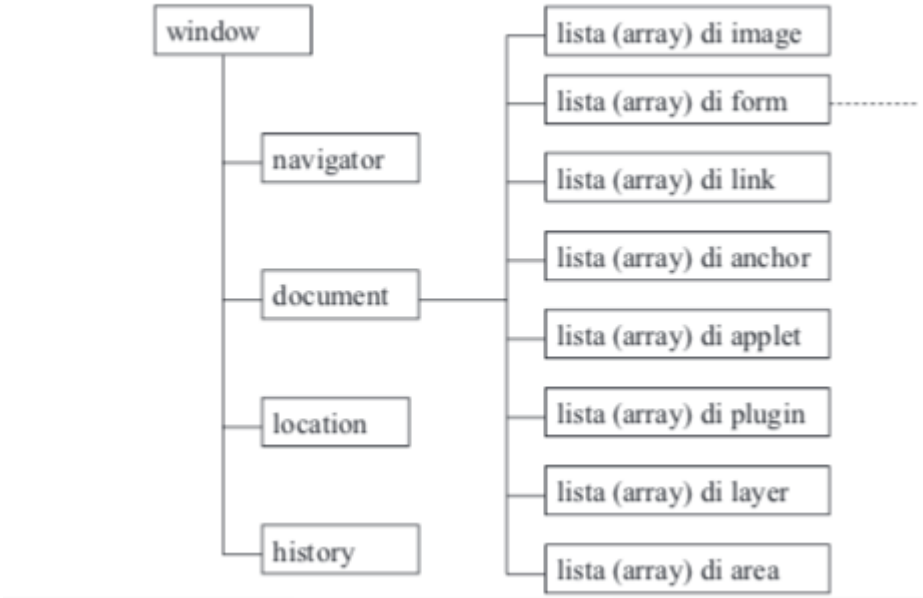
In altre parole, è uno standard che definisce come ottenere, cambiare, aggiungere o modificare elementi HTML.

JavaScript usa HTML DOM per modificare tutti gli elementi di una pagina web. Quando una pagina è caricata il browser crea il DOM della pagina, la pagina viene rappresentata come un albero. DOM è definito dal browser (Definizione è fatta separatamente per Explorer, Mozilla, etc. e possono nascere incompatibilità).

Tramite il modello DOM a oggetti, JavaScript può:

- Cambiare tutti gli elementi e attributi HTML di una pagina
- Cambiare tutti gli stili CSS
- Rimuovere elementi e attributi HTML
- Aggiungere elementi e attributi HTML
- Reagire a eventi nella pagina
- Creare nuovi eventi

Il DOM è organizzato secondo una gerarchia:



window = è la radice della gerarchia DOM e **rappresenta la finestra corrente del browser**. I figli dell'oggetto window sono:

- **navigator** = il browser (in quanto applicazione). Ad esempio, per sapere quale browser si sta utilizzando (Explorer/ Netscape).
- **document** = il contenuto della finestra.
- **location** = informazioni sull'URL corrente. Ad esempio, per caricare nella finestra un URL differente.
- **history** = elenco degli URL visitati di recente. Ad esempio, per tornare alla pagina Web precedente.

La window fornisce diversi metodi come:

- **alert**: window.alert(messaggio) propone una finestra di avviso con messaggio pop-up, è usabile anche in un link e non restituisce nessun valore.
- **confirm**: fa apparire una finestra di conferma con il messaggio dato. Restituisce true o false.
- **prompt**: fa apparire una finestra di dialogo per immettere un valore. Restituisce il valore string.

document = rappresenta il documento corrente: il contenuto della pagina web attuale. Da non confondere con la finestra corrente! Le componenti principali di document sono:

- forms (array) = lista dei moduli (form) contenute nella pagina
- elements (array di Buttons, Checkbox, etc etc...)
- anchors (array)
- links (array)
- images (array) = lista delle immagini contenute nella pagina
- applets (array)
- embeds (array)

Il document ha anche diversi metodi disponibili come il metodo write che stampa un valore (stringhe, numeri, immagini, etc…) a video.

oggetto = è una collezione di:

- **proprietà** (variabili): sono a loro volta oggetti. Es.: window.status = 'hello!'; (window = nome oggetto e status = nome proprietà)
- **funzioni** (metodi, operazioni). Es.: window.print();

L'invocazione di una funzione apparentemente senza alcun oggetto si riferisce all'oggetto window. Un riferimento all'oggetto document non preceduto da un riferimento all'oggetto window, è equivalente al caso in cui document è preceduto da window (implicito).

L'accesso agli oggetti contenuti in una pagina può avvenire tramite liste delle immagini (<IMG...) (o dei moduli (<FORM...)) contenute in una pagina. Es.: window.document.images[i]

I campi di testo sono oggetti dotati di nome posti all'interno di un oggetto form pure esso dotato di nome. Il campo di testo è caratterizzato dalla proprietà value.

```
<FORM name="myform">
 <INPUT type="text" name="cognome" size=20>
<INPUT type="button" name="bottone" value="Show"
onClick="alert(document.myform.cognome.value)">
</FORM>
```

Metodo getElementById(idname) = Ritorna l'elemento con uno specifico id. Ritorna null se non esiste l'elemento.

eventi = i programmi Javascript sono tipicamente guidati dagli eventi (event-driven). Il gestore di eventi: event-handler. Il DOM fornisce una serie di gestore di eventi predefiniti. (Esempio: <a href="#" onClick = "window.print()" > Attributo onClick = evento click innesca il gestore. window.print() = un codice JavaScript che viene eseguito dal gestore)

L'accesso agli oggetti di una pagina avviene tramite nodi infatti ogni oggetto è un nodo. La relazione tra nodi viene definita attraverso funzioni: parent, child, sibling.

Funzioni, modello a oggetti

Object-**based** language usa l'idea di incapsulare stato e operazioni all'interno di oggetti. Non applica i concetti di ereditarietà e sottotipi. JavaScript a volte riferito come prototype-based language, non esistono classi, ma gli oggetti ereditano codice e dati da altri oggetti template.

Le **funzioni** sono definite tramite la keyword **function** e sempre racchiuse in un blocco. Possono essere considerate sia procedure (non ha istruzione return) sia funzioni in senso proprio (non esiste la keyword void). I parametri formali sono senza dichiarazione di tipo.

Dichiarazione delle variabili è:

- Implicita o esplicita per variabili globali (la dichiarazione implicita è sempre e solo per variabili globali)
- esplicita, per variabili locali (la dichiarazione esplicita ha un effetto che dipende da dove si trova la dichiarazione)

Dichiarazione esplicita (keyword var):

- var pippo = 10 // dichiarazione esplicita
- pippo = 10 // dichiarazione implicita

Un **oggetto** JavaScript è una collezione di dati dotata di nome, ogni dato è interpretabile come una proprietà. Per accedere alle proprietà si usa la "dot notation" (es.: nomeOggetto.nomeProprietà). Tutte le proprietà sono accessibili.
Un oggetto JavaScript è costruito tramite un **costruttore** che stabilisce la struttura dell'oggetto e quindi le sue proprietà. I costruttori sono invocati mediante l'operatore **new.**
In JavaScript non esistono classi, sottoclassi, il nome del costruttore è a scelta dell'utente.

Area 3 - HTML5, CSS3

HTML5

Introduzione

HTML ha molte **limitazioni**. Ha bisogno di flash/quicktime per eseguire audio/video, non può memorizzare dati lato client se non con linguaggi di scripting o altre tecnologie, non ha un formato nativo per il disegno: grafica o animazioni fornite come immagini o con Flash, Java, Silverlight. HTML limita l'evoluzione del web.

HTML5:

- è uno **standard per strutturare e presentare il contenuto del World Wide Web**.
- è una cooperazione tra World Wide Web Consortium (W3C) e Web Hypertext Application Technology Working Group (WHATWG).
- è **pragmatico**, parte dalla considerazione che i browser decidono cosa implementare e cosa no (non ha senso tradurre in specifiche qualcosa che non viene utilizzato).
- fornisce **supporto** per tutto ciò che è **JavaScript**.
- aggiunge **semantica**, migliore definizione delle componenti delle pagine web, i tag HTML forniscono informazioni aggiuntive utili per motori di ricerca.

Lista di **funzionalità** sotto il cappello "HTML5":

1. **Nuovi elementi strutturali**, semantica (ad es., <header>, <footer>, <nav>,)
2. **Form 2.0** e validazione client-side
3. Supporto nativo dei browser per audio e video (<video>, <audio>)
4. Canvas API e SVG
5. **Web storage** = memoria che si trova dentro il browser che permette di memorizzare i dati come coppia (chiave, valore).
6. Offline application
7. Geolocation
8. **Drag & Drop**
9. Web Workers
10. Nuove API di comunicazione (Server Sent Events, Web Sockets, ...)

HTML5 non è una, ma un gruppo di tecnologie.

Concetti base

La sintassi è:

- più **permissive** di XHTML per garantire compatibilità.
- **case <u>in</u>sensitive**, maiuscole e minuscole non interferiscono con la validazione della pagina
- I tag di chiusura <u>non</u> sono richiesti.
- Le virgolette e i valori sono opzionali per gli attributi.

HTML5 introduce 28 nuovi elementi: <section>, <article>, <aside>, <hgroup>, **<header>**,**<footer>**, **<nav>**, <figure>, <figcaption>, <video>, <audio>, <source>, <embed>,

<mark>,<progress>, <meter>, <time>, <ruby>, <rt>, <rp>, <wbr>, <canvas>, <command>, <details>,<summary>, <datalist>, <keygen> and <output>.

Semantic markup =
- Definizione e scelta dell'elemento migliore per l'oggetto che si vuole rappresentare.
- Sintassi che da l'idea di cosa si sta facendo sia a un **uomo** che a una **macchina** (browser).
- Associa un **significato** al contenuto.
- Pagine facili da capire e modificare per sviluppatore, adatte per mostrare il contenuto in maniera **comprensibile** a utenti con disabilità.

Categorie di contenuti HTML5:
- **Metadata** = configura la modalità di presentazione e il comportamento del contenuto della pagina.
- **Flow** = il vero contenuto della pagina. Contiene tutti gli elementi usati per marcare il contenuto.
- **Sectioning** = Nuova categoria di HTML5, include quattro elementi: <article>, <aside>, <nav> e <section>.
- **Heading** = Contiene tutti gli elementi standard HTML 4.0 per heading.
- **Phrasing** = è il testo del documento. Includono elementi per marcare testo interno a paragrafi.
- **Embedded** = importa un'altra risorsa nella pagina, ad esempio, audio e video.
- **Interactive** = elementi con il compito di gestire l'interazione con l'utente.

HTML5 web storage

HTML5 supporta memorizzazione dei dati ottenuti da internet nella propria macchina desktop o smartphone tramite due modalità:
- **application caching** = salva la logica applicativa e l'interfaccia utente. Caching classico dei contenuti.
- **offline storage** = cattura dati generati dall'utente e risorse di interesse per l'utente.

I dati memorizzati sono connessi alla loro origine. Di solito i dati possono essere acceduti solo da un dominio e possono essere acceduti da tutte le applicazioni di quel dominio.

Web Storage API definisce uno standard per salvare dati localmente su un computer utente o su un device.
Prima dell'avvento dello standard Web Storage, gli sviluppatori web memorizzavano le informazioni in **cookie**, o tramite **plugin**, con Web Storage abbiamo un meccanismo standardizzato per memorizzare fino a 5MB di dati creati dal sito web o dall'applicazione web.
Web Storage è importante per lo sviluppo di Offline Web Application infatti c'è la necessità di memorizzare i dati utente mentre si lavora offline, e il Web Storage fornisce tale spazio.

Ci sono due tipi di offline storage:
- **session storage** = storage di sessione. Permette di memorizzare dati specifici a una finestra/tab. Permette di **isolare informazioni in ogni finestra**, se l'utente visita lo stesso sito in due finestre diverse, ogni finestra avrà il suo session storage

individuale con dati separati e distinti. **Non è persistente**. Viene mantenuto solo per la durata della sessione utente in uno specifico sito, viene mantenuto per il tempo in cui la finestra/tab del browser è aperta e visualizza il sito.
- **local storage** = A differenza del session storage, il local storage permette di salvare dati persistenti sul computer dell'utente, attraverso il browser. Quando l'utente rivisita il sito successivamente, ogni dato salvato nel local storage può essere **recuperato**.

Dati memorizzati come coppie chiave-valore (simili ai cookie).

Differenza tra local storage e cookie. I local storage possono essere scambiati a una prima occhiata per HTTP cookie ma ci sono alcune differenze chiave:
- Cookie sono letti **lato server**, mentre local storage sono disponibili solo **lato client.**
- Se l'esecuzione del codice lato server dipende sul valore di alcune variabili, i cookie sono la soluzione ideale. I cookie inviati con ogni richiesta al server portano **overhead sostanziali** in termini di banda usata.
- Local storage risiede sull'hard disk dell'utente, e viene letto a **zero costo**.
- Local storage fornisce molto spazio di memorizzazione, i Cookie memorizzano fino a 4KB di informazione mentre il Local storage memorizza fino a 5MB di informazione.

i metodi principali del local storage sono:
- **setItem(key,value)** = Salva un valore con una nuova chiave o ne aggiorna uno con una chiave esistente
- **getItem(key)** = Accede a un valore con una determinata chiave
- **removeItem(key)** = Cancella la entry con chiave key dal data store
- **clear()** = Cancella tutte le entry dal datastore per l'applicazione corrente
- **length** = Proprietà readonly che ritorna il numero di chiavi nel data store

Drag & Drop

Permette di spostare un elemento in un'altra posizione. Ogni elemento può essere trascinato. Attributo draggable="true" rende un elemento trascinabile.

CSS3

Introduzione

CSS fondamentale per supportare il passaggio verso HTML5 dove la rappresentazione della struttura e la semantica sono separati da elementi stilistici.
CSS3 cambia il modo di operare:
- Specifica monolitica sostituita da approccio modulare
- Ogni modulo tratta un tema diverso
- Ciascun tema sviluppato con tempistiche diverse

Strumenti tipografici avanzati infatti si ha maggiore controllo sul testo, una gestione avanzata della disposizione di oggetti su più colonne, opacità elementi e bordi con angoli smussati. Si aggiungono effetti grafici come transizioni, trasformazioni e animazioni.

Buona risposta da parte dei fornitori di browser.

Nuovi selettori/pseudoclassi

Nuovi selettori di attributi e pseudo classi permettono di intervenire in maniera puntuale, ad esempio, posso selezionare elementi che contengono solo una parte di un valore di un attributo.
Nuovi selettori di attributi e pseudo classi permettono di intervenire in maniera discreta (non invasiva). La regola CSS può essere applicata con class o id senza modificare la pagina HTML.
Pulizia del codice elevata e costi di manutenzione minori. Maggior indipendenza tra il documento che riceve lo stile e lo stile stesso, maggior riutilizzo delle due componenti.

Selettori di attributi = Regola si applica a un elemento HTML che contiene un determinato attributo o ha un certo valore per l'attributo.

+ Modalità di selezione puntuale e discreta
- Performance ridotte
- Calcolare quanti e quali elementi sono coinvolti è oneroso
- Utilizzare solo se necessari

Pseudo-classi struttura = Regole per assegnare uno stile a un elemento in base alla posizione nel documento HTML.

@media rule

@media rule è usato per definire diverse regole di stile in base ai diversi media type/device. In CSS2, venivano chiamati media types, in CSS3 vengono chiamate media queries. Media queries guardano alle **capacità del device** e possono essere usate per controllare diversi aspetti di come viene visualizzata una pagina.

Trasformazioni

Un altro modulo CSS3 permette di applicare trasformazioni a un elemento HTML. Interviene sulle coordinate che ne determinano la posizione.
Applicazione di trasformazioni alle coordinate bidimensionali .
Applicazione di funzione di trasformazione come:

- rotate = ruota l'elemento in senso orario
- skew = produce un'inclinazione di elemento HTML
- scale = modifica la dimensione di un elemento HTML

XML

Introduzione

Programmable Web è simile al web con la differenza è che usa documenti XML o simili.
Si basa su http e XML.

XML

eXtensible Markup Language = A set of rules for defining and representing information as structured documents for applications on the Internet; a restricted form of SGML.

XML è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio marcatore basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.
XML è stato creato per strutturare, memorizzare, descrivere e trasmettere i dati. Non ha dei tag predefiniti, si inventano. **XML è uno strumento per trasmettere informazioni, indipendente dalla piattaforma, dal software e dall'hardware**.

La sintassi di XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <nota>
 <a>Gianni</a>
<da>Monica</da>
 <titolo>Promemoria</titolo>
<corpo>Ricordati di passare a prendermi domani</corpo>
</nota>
```

Con XML i dati vengono memorizzati separatamente dai documenti HTML. La prima riga, la dichiarazione XML, definisce la versione di XML e il tipo di codifica dei caratteri.

Gli elementi in un documento XML possono essere estesi per rappresentare più informazioni.

Gli elementi XML possono avere degli attributi nel tag iniziale. Gli attributi forniscono maggiori informazioni sul tag. Il valore dell'attributo deve essere compreso tra doppi apici oppure tra singoli apici.

Alcuni problemi che si possono incontrare usando gli attributi Possono contenere un solo valore Non sono estendibili in caso di future revisioni Non possono descrivere strutture Sono più difficili da manipolare da parte delle applicazioni

Validazione di documenti XML

Schema e DTD = (Document Type Definition) sono documenti che definiscono la struttura di un documento XML. Gli schemi sono a loro volta documenti XML. I DTD hanno una sintassi un po' più complessa e sono obsoleti.

XML Schema è uno standard per la validazione dei file XML, scritto in XML. Definisce tipo e cardinalità degli elementi Basato sui concetti di tipi semplici e complessi

JSON

Introduzione

(JavaScript Object Notation) È un formato adatto all'interscambio di dati fra applicazioni client-server. JSON è **text-based open standard** leggero disegnato per lo **scambio di dati** human-readable.

JSON è usato quando si scrivono applicazioni basate su JavaScript. Usato per serializzare e trasmettere **dati strutturati** su una connessione di rete. Primariamente usata per trasmettere dati tra server e applicazioni web. Web Service e API usano JSON per fornire dati pubblici. Può essere usato con linguaggi moderni.
JSON è:

- una sorta di dialetto XML
- facile da leggere e scrivere
- formato di scambio basato su testo e **molto leggero (a differenza di XML)**
- indipendente dal linguaggio

La sintassi JSON è considerata come un sottoinsieme della sintassi JavaScript:

- i dati sono rappresentati come coppie nome/valore
- le parentesi graffe contengono oggetti e ogni nome è seguito dal : (colon), le coppie nome/valore sono separate da , (comma)
- Parentesi quadre definiscono array e sono separate da , (comma)

```
{ "book": [
{ "id":"01",
"language": "Java",
"edition": "third",
"author": "Herbert Schildt" },
{ "id":"07",
"language": "C++",
"edition": "second"
"author": "E. Balagurusamy" }
]}
```

JSON supporta due strutture dati:

- Insieme di coppie nome/valore
- Lista ordinate di valori (array)

JSON Schema è una specifica per la definizione/validazione dei dati JSON.

JSON vs XML.

- JSON è simile a XML ma più leggero
- sono entrambi formati human readable e indipendenti dal linguaggio
- ogni valore in XML richiede apertura e chiusura di un tag, in JSON solo il valore.

Possibile confrontare JSON e XML sulla base di tre fattori principali:

- **verbosità** = XML più verboso di JSON. Più veloce scrivere JSON per esseri umani.
- **utilizzo degli array** = XML usato per descrivere dati strutturati che non includono array. JSON include array.
- **parsing** = attività di analisi sintattica che permette di interpretare una stringa. Il metodo eval di Javascript fa il parsing di JSON. Quando applicato a JSON, eval ritorna l'oggetto descritto.

in sintesi:
JSON:

- è leggero, quindi semplice da leggere e scrivere
- supporta gli array

- i file JSON sono più leggibili dall'uomo
- non ha capacità di visualizzazione
- fornisce tipi di dati scalari e la capacità di esprimere dati strutturati tramite array e oggetti
- supporto per oggetti nativi
- XML è meno semplice di JSON

XML:
- non supporta array
- i file XML sono meno leggibili
- fornisce la capacità di visualizzare i dati perchè è un linguaggio di markup
- non fornisce alcuna nozione di tipi di dati. Bisogna fare affidamento a XML Schema per aggiungere informazioni sul tipo
- gli oggetti devono essere espressi per convenzioni, spesso attraverso un misto di attributi ed elementi.

somiglianze tra JSON e XML, entrambi:
- sono semplici e aperti
- supportano unicode
- rappresentano dati autodescrittivi
- sono interoperabili e indipendenti della lingua

Javascript e XML

XMLHttpRequest Object = usato per scambiare dati con il server in background.
Permette di:
- Aggiornare una pagina senza ricaricarla
- Richiedere i dati dal server dopo che la pagina è stata caricata
- Ricevere i dati dal server dopo che la pagina è stata caricata
- Inviare i dati al server in background xmlhttp = new XMLHttpRequest();

Nonostante il nome, tramite questo oggetto è possibile ricevere dati anche in **formati diversi** dall'XML, come il **JSON** o il formato binario.
XML Parser = converte un documento XML in un oggetto XML DOM, manipolabile con javascript.

Javascript e JSON

JSON è un'alternativa leggera a XML ed è un meccanismo semplice e **indipendente dal linguaggio** per formattare strutture dati
JSON.parse
JSON.stringify

Area 4 - Virtualizzazione e Cloud

Virtualizzazione

Introduzione

Virtualizzazione = attività mirata alla creazione di sostituti (risorse virtuali) per le risorse reali, che hanno le stesse funzioni e interfacce esterne delle proprie controparti, ma attributi diversi (dimensione, prestazioni e costo).

Emulazione vs Simulazione vs Virtualizzazione. Sono termini in relazione ma non sono la stessa cosa:

- **Emulazione** = emula il comportamento del sistema reale, **esegue codice non modificato**, **accurato** e flessibile, **costoso**. Rappresenta esattamente il sw originale, però è lento, infatti ogni istruzione del sw originale deve essere tradotta nell'hw che si sta utilizzando.
- **Simulazione** = **approssima** il comportamento del sistema reale, r**ichiede riscrittura del software**, **poco costoso** e flessibile, **perde accuratezza**. Rapida ma meno precisa.
- **Virtualizzazione** = virtualizza il comportamento esatto del sistema reale, cross-platform, **accurato** e flessibile, **costoso**.

Perché virtualizzare: **problemi**

- Troppi server, poco lavoro
- Hardware vecchio non funziona
- Requisiti infrastrutturali sempre maggiori
- Poca flessibilità in ambienti condivisi
- Hardware sottoutilizzato
- Costi e necessità elevati
- Ambienti eterogenei

Perché virtualizzare: **benefici**

- Consolidamento e riduzione del costo dell'hardware
- Ottimizzazione dei carichi di lavoro
- Flessibilità e capacità di risposta IT
- Ambienti esecutivi multipli
- Gestione semplificata
- Maggiori performance, trasparenza, eterogeneità, portabilità, green IT

Componenti della virtualizzazione:

- Hypervisor: la componente di sistema che realizza la virtualizzazione
- Virtual Machine Monitor: la componente applicativa che realizzala virtualizzazione
- Sistema (SO) guest: il sistema virtualizzato (sinonimo di virtual machine)
- Sistema (SO) host: il sistema reale
- Management Server: piattaforma di virtualizzazione

- Management Console: fornisce accesso all'interfaccia di gestione della virtualizzazione
- Network components: permettono lo sviluppo di reti virtuali
- Virtualized Storage: fornisce i componenti di astrazione dello storage fisico

Grado di emulazione dell'hardware:
- **Emulazione completa** = permette di emulare tutti gli aspetti di un calcolatore (hw, so, app). Per eseguire un SO guest non modificato su un'architettura host completamente diversa
- **Emulazione parziale** =
    - Virtualizzazione completa: VMM emula l'hardware necessario da permettere l'esecuzione isolata di un SO guest
    - Virtualizzazione assistita dall'hardware: VMM permette l'esecuzione isolata di un SO guest non modificato. VMM mette a disposizione dei SO guest hardware vero tramite alcune estensioni hardware del processore
    - Para virtualizzazione: VMM non emula necessariamente l'hardware, mette a disposizione una API per estendere il SO guest. L'estensione consiste nella implementazione di hypercall

La virtualizzazione può essere effettuata a diversi livelli:
- **Sistema operativo** = VMM ha il compito di partizionare le risorse hardware del SO host fra i vari guest
- **Applicazione** = meccanismo per eseguire i programmi in maniera portabile su diverse architetture hardware/software
- **Risorse**

La virtualizzazione può essere:
- **nativa** (bare-metal) = se un sistema di virtualizzazione prevede l'utilizzo esplicito di un unico SO.
- **non nativa** (hosted) = se un sistema di virtualizzazione prevede l'utilizzo di un SO host e di un SO guest.

Cloud

Introduzione

Il cloud è un'integrazione di quattro oggetti:
- hardware
- tecnologie internet
- sistemi distribuiti
- system management

Sistema distribuito = n processori autonomi (siti): n amministratori, n sistemi operativi, n flussi di dati/controllo. Una rete d'interconnessione.
- Visione utente: un solo sistema (virtuale).
- Visione sviluppatore: client-server

SOA (Service-Oriented Architecture) = un'architettura software adatta a supportare l'uso di servizi Web per garantire l'interoperabilità tra diversi sistemi così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business e soddisfare le richieste degli utenti in modo integrato e trasparente.

- Sfruttano la soluzione di delivery dei Web Services
- Implementano il concetto di sistema e computing distribuito fornendo un sistema debolmente accoppiato, standard e indipendente dal protocollo
- Forniscono risorse software come servizi con interfacce pubbliche

Sistema parallelo = 1 computer, n nodi: un solo sistema operativo, un solo amministratore. Memoria: dipende (distributed vs shared). Visione sviluppatore: una sola macchina che esegue codice parallelo.

Cluster computing = uso di PC interconnessi da una rete ad alte prestazioni come macchina parallela. Approcci principali: rete dedicata, rete general purpose.

Network computing = estensione del cluster computing alle WAN. Insieme di macchine distribuite su una MAN/WAN che eseguono codice parallelo.

Meta computing = un meta computer è un insieme di risorse distribuite in grado di eseguire collaborativamente il codice. Una macchina virtuale eseguita su un sistema distribuito.

Grid computing = condivisione di risorse coordinata e problem solving in organizzazioni virtuali dinamiche e multi-istituzionali. Permette aggregazione di risorse distribuite e accesso trasparente.

Cloud computing

Utility computing = on demand delivery of infrastructure, applications, and business processes in a security- rich, shared, scalable, and standard-based computer environment over the Internet for a fee.

Autonomic computing = sistemi autonomi con self-management. Forniscono meccanismi di adaptation, riducono il coinvolgimento degli utenti. Automazione dei data center.

Cosa è una cloud? Paradigma che si basa sui concetti di utility computing, autonomic computing, sistemi distribuiti.

Partendo dalla definizione del NIST spiegare il cloud computing.
Cloud computing = definizione del NIST: Il cloud computing è un **modello** per consentire **l'accesso** conveniente e diffuso di rete on-demand(su richiesta) a un **pool condiviso di risorse informatiche configurabili** (ad es. reti, server, storage, applicazioni e servizi) di cui è possibile eseguire rapidamente un rifornimento e il rilascio con il minimo sforzo di gestione o interazione del fornitore di servizi. Questo modello cloud è composto da cinque caratteristiche essenziali, tre modelli di servizio e quattro modelli di distribuzione.

Si indica un paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili.

caratteristiche essenziali del cloud computing:

- **On-demand self service** = Un cliente può procurarsi risorse, server time e network storage automaticamente a secondo del bisogno senza interazione «umana» con il service provider
- **Broad network access** = risorse disponibili attraverso la rete, accesso tramite protocolli standard, supporto per tutti i tipi di device e piattaforme.
- **Rapid elasticity** = capacità di scalare le risorse elasticamente in base ai **reali bisogni**. La sensazione del cliente è di avere risorse infinite, anche se non è assolutamente vero. Nessuno spreco di risorse tipiche dei sistemi on premises.
- **Measured service** = la cloud controlla e ottimizza le risorse dinamicamente usando funzionalità di metering (paghi solo per quello che consumi). Utilizzo di risorse viene monitorato, controllato, loggato fornendo trasparenza per il provider e il cliente dei servizi cloud.
- **Resource pooling** = le risorse sono ripartite per servire i clienti tramite un modello multi-tenant. Risorse fisiche e virtuale (ri)assegnate dinamicamente a seconda del bisogno. Indipendenza dalla locazione.

Caratteristiche aggiuntive: lower cost, ease of utilization, quality of service (QoS), reliability, outsourced IT management.
Multi-tenancy = capacità di condivisione di una singola istanza condivisa di un software tra più organizzazioni/client (tenant). Dati degli utenti sono separati virtualmente, non fisicamente.

- Vantaggi: costi ridotti per il cloud provider, dinamicità di accesso a risorse condivise.
- Svantaggi: utenti potrebbero essere in grado di accedere a dati di altri utenti, backup e restore dei dati difficoltoso.

Modelli di servizio Cloud Computing

IaaS (Infrastructure as a Service)

- Utente gestisce interamente CPU, memoria, storage, rete, e risorse di computazione aggiuntive.
- Utente in grado di installare ed eseguire codice generico incluso sistemi operativi e applicazioni.
- Utente non gestisce o controlla l'infrastruttura Cloud, mentre controlla sistemi operativi, storage e applicazioni installate.
- Offre risorse virtualizzate on demand.
- Fornisce diversi server con diversi sistemi operativi e uno stack software ad hoc.

PaaS (Platform as a Service)

- Utente installa e crea applicazioni sviluppate tramite linguaggi di programmazione, librerie, tools, e servizi supportati dal provider.
- Utente mantiene il controllo sulla piattaforma installata.
- Utente non gestisce o controlla l'infrastruttura sottostante che include rete, sistemi operativi, server o storage.

- Utente mantiene il controllo delle applicazioni installate e delle configurazioni dell'ambiente per hosting di applicazioni
- Piattaforma installata sull'infrastruttura dove sviluppatori creano e installano applicazioni

SaaS (Software as a Service)
- Utente utilizza le applicazioni di un cloud provider che sono eseguite sull'infrastruttura cloud.
- Il cliente non gestisce o controlla l'infrastruttura cloud, incluso rete, server, sistema operativo, storage o anche funzionalità specifiche dell'applicazione

i 4 Deployment Models:
- **Private cloud** = fornita per uso esclusivo di una singola organizzazione che comprende utenti multipli.
- **Community cloud** = fornita per uso esclusivo da parte di specifiche comunità di utenti
- **Public cloud** = fornita per uso comune e al pubblico (es. Google)
- **Hybrid cloud** = combinazione di due o più infrastrutture cloud distinte (private, community, o public).

Migrazione e Cloudonomics

Perché migrare da sistemi on premise a cloud:
- On-demand resourcing
- Scalability
- Economy of scale
- Flexibility and elasticity
- Growth
- Utility based metering
- Shared infrastructure
- High availability
- Security

5 livelli di migrazione:
1. application,
2. code,
3. design,
4. architecture,
5. usage.

7-step model:
1. **Cloud Migration Assessment** = Valutazione delle problematiche ai 5 livelli di migrazione. Considera tool da usare, test case, funzionalità, configurazione.
2. **Isolate the dependencies** = Isolare le dipendenze sistemistiche e ambientali dei componenti applicativi nel data center locale. Modellare il livello di complessità della migrazione.
3. **Map the messaging and the environment** = Mappare cosa deve rimanere locale e cosa deve essere migrato.

4. **Re-architect and implement the lost functionalities** = Identificare cosa deve essere riprogettato e ri-implementato. In alcuni casi alcune funzionalità possono essere perse.
5. **Leverage cloud functionalities & features** = Usare funzionalità cloud per implementare il servizio e se possibile migliorarlo
6. **Test the migration** = Testare il corretto funzionamento del servizio sulla cloud. Identificare i rischi della migrazione
7. **Iterate and optimize** = Se il test produce risultati contrastanti, iterare il processo per ottimizzare la migrazione dell'applicazione sulla cloud. Gestire i rischi della migrazione identificati al passo 6.

Le 10 leggi della **Cloudonomics:**

1. Utility services cost less even though they cost more = Il costo per unità di tempo è maggiore. **Accesso on demand alle utility riduce il costo totale**.
2. On-demand trumps forecasting = **Capacità di allocazione e deallocazione quasi istantanea**. Previsioni spesso sbagliate, capacità di reagire prontamente permette grande guadagno.
3. The peak of the sum is never greater than the sum of the peaks = **La quantità di risorse è la somma dei picchi**. La cloud installa meno risorse (riallocazione delle risorse).
4. Aggregate demand is smoother than individual = **L'aggregazione di richieste da diversi clienti tende a ridurre le variazioni.** Cloud ottiene miglior efficienza.
5. Average unit costs are reduced by distributing fixed costs over more units of output = **Costi fissi sono maggiormente distribuiti.** Diminuisce il costo per unità in diversi ambiti: storage, bandwidth, etc.
6. Superiority in numbers is the most important factor in the result of a combat = **Superiorità numerica permette di vincere le battaglie**. Attacco DoS più difficile nella cloud.
7. Space-time is a continuum = **Cloud scalability permette decision making più rapido**.
8. Dispersion is the inverse square of latency = **Ridurre la latenza è fondamentale** per molte applicazioni. Più semplice in cloud.
9. Don't put all your eggs in one basket = Maggior affidabilità. **Replica su data center distribuiti**.
10. An object at rest tends to stay at rest = **I data center** aziendali sono installati nella sede dell'azienda. Nella cloud **sono installati dove è più vantaggioso**.

Un'altra caratteristica che influenza i costi del cloud computing sono i **pattern di crescita**:

- **Constant growth** = Numero di utenti cresce mese per mese Il numero di server cresce proporzionalmente per gestire il numero di richieste.
- **Seasonal growth** = Crescite e contrazioni previste durante l'anno.
- **Lifecycle growth** = Aziende che osservano crescite temporanee durante il lancio di nuovi prodotti e attività commerciali.

i pattern poi possono essere:

- **Permanent pattern** = Il pattern persiste, quando applicato cambia il numero di risorse da usare.

- **Temporary pattern =** Il pattern ha una durata temporale, alla fine della finestra temporale l'utilizzo della risorsa ritorna al numero originale.

TCO (Total Cost of Ownership): stima dei costi di utilizzo di un prodotto per il suo ciclo di vita
SLA (Service-Level Agreement): stabiliscono accordi tra cloud provider e utenti

Area 5 - Servizi REST

REST (**RE**presentational **S**tate **T**ransfer) E' un tipo di architettura software per i sistemi di ipertesto distribuiti come il World Wide Web. Alternativa alle architetture software basate su **SOAP** (**S**imple **O**bject **A**ccess **P**rotocol).

Gestiscono un tipo di contenuto variabile che può essere XML, ma principalmente è JSON.

REST è Moderna architettura web descritta attraverso un **architectural style** = Architectural style is a named, coordinated set of architectural constraints. Il design architetturale considera un sistema nella sua interezza, senza vincoli (constraint). Identifica e applica vincoli incrementalmente agli elementi del sistema. Differenzia lo spazio del design e permette ai parametri che influenzano il comportamento del sistema di fluire naturalmente, in armonia con il sistema.

ROA (**R**esource-**O**riented **A**rchitecture)
- REST è un'astrazione di elementi architetturali in un sistema hypermedia distribuito
- REST ignora i dettagli implementativi dei componenti e la sintassi dei protocolli
- REST si focalizza sul ruolo dei componenti, vincoli per interazione con altri componenti e l'interpretazione dei dati

RESTFul Web Service È una configurazione di URI, HTTP, XML/JSON che lavora come il resto del web.

La natura e lo stato dei **data element** è un aspetto essenziale di REST (Dovuto alla natura degli hypermedia distribuiti). Quando un link è selezionato l'informazione fluisce dalla locazione dove è memorizzata a quella dove verrà usata.
Ci sono 3 possibilità di comunicazione (REST supporta un ibrido delle tre)
1. Fare rendering del dato dove si trova e mandare un'immagine a formato fisso al ricevente.
2. Incapsulare il dato con l'engine di rendering e inviarlo al ricevente.
3. Mandare il dato all'utente che farà il rendering.

REST fornisce una versione ibrida delle tre opzioni:
- comprensione condivisa di dati e metadati
- limita lo scope di cosa viene rivelato all'interfaccia standard

Componenti REST trasferiscono una risorsa usando una delle rappresentazioni standard:
- Rappresentazione selezionata in base ai desideri del ricevente e alla natura della risorsa
- Conoscenza della rappresentazione originale nascosta dietro le interfacce
- La rappresentazione consiste di istruzioni in un formato standard relative a un engine di rendering

REST fornisce
- Separation of concern
- Scalabilità

- Permette information hiding attraverso interfacce generiche che supportano encapsulation e service evolution
- Fornisce un set di funzionalità attraverso un engine di feature scaricabile

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Resource = è ogni informazione che può avere un nome, ogni concetto che può essere target di riferimento ipertestuale. Ogni resource è una membership function che al tempo t viene associata a un set di entità/valori. I valori sono la rappresentazione della risorse e/o gli identificativi. Le risorse possono essere:

- vuote,
- statiche, ad esempio il libro XYZ
- dinamiche, ad esempio il mio libro preferito

L'unica cosa sempre statica è la semantica della risorsa

Resource identifier = l'autorità di naming deve mantenere la validità semantica del mapping tra identificatore e risorsa. URI, URN

Addressability = un'applicazione è indirizzabile se:

- espone i suoi dati come risorse,
- espone un URI per ogni informazione di interesse,
- il file system è indirizzabile.

Statelessness = **ogni richiesta http è eseguita in isolation**. Il server non usa informazioni da richieste precedenti, il client inserisce tutto ciò che è necessario a soddisfare la richiesta. Nonostante il principio di statelessness, spesso il web è implementato stateful.
Perché statelessness?

- Lo stato renderebbe le richieste molto più semplici, ma la gestione del client HTTP sarebbe molto più complicata.
- Senza stato si rimuovono gran parte delle condizioni di fallimento.

Dal punto di vista server è utile perché:

- Facile distribuire applicazioni tra server in load balancing
- Scaling up fatto semplicemente aggiungendo server senza preoccuparsi dello stato
- Applicazioni facili da cachare

Dal punto di vista client è utile perché:

- Sessione può essere aggiunta a livello applicativo

Representation = usata per ottenere lo stato di una risorsa come documenti, file, http message entity.

Link e connettività = link e form all'interno di una pagina web, connettività.

Visione architetturale

- Process view = mostra le interazioni tra componenti rivelando il percorso dei dati nel sistema
- Connector view = si concentra sulle comunicazioni tra componenti, specialmente i vincoli che definiscono interfacce di risorse generiche
- Data view = rivela lo stato dell'applicazione al fluire dell'informazione

RESTful Services

Nonostante REST sia un architectural style per networked hypermedia application, è usato principalmente per costruire lightweight, maintainable, e scalable Web services.
Un servizio basato su REST è chiamato **RESTful service**.
REST non dipende da nessun protocollo, anche se quasi tutti i RESTful service usano HTTP.

REpresentational State Transfer dal punto di vista del Client

Visione astratta del client REST:

- Definire informazioni da aggiungere all'HTTP request: HTTP method, URI, HTTP header, documenti da mandare nel body
- Creare l'HTTP request e mandarla al server HTTP
- Parsare la risposta (codice, header, entity body) e fornire le informazioni al codice REST

Il client usa HTTP library per lettura/scrittura di messaggi HTTP

- Feature obbligatorie: supporto HTTPS/SSL, supporto HTTP method, possibilità di customizzare header e body del messaggio, accesso alla risposta (compreso header e response code), HTTP proxy
- Feature opzionali: supporto per dati in forma compressa, cache delle risposte, supporto dell'autenticazione HTTP (Basic, Digest, WSSE), supporto per HTTP redirect, capacità di interpretare e creare cookie

Il client usa XML parser per interpretare la risposta

- Document-based model = DOM, Tree-style parser, struttura data annidata accessibile e modificabile tramite XPath e CSS. Richiede caricamento di tutto il documento in memoria.
- Event-based strategy o pull = SAX ,trasforma il documento in un flusso di eventi (inizio/fine tag, commenti XML, dichiarazione di entità). Gestisce un evento alla volta e può fermare il parsing su richiesta.

Client REST possono essere sviluppati in molti linguaggi.

REpresentational State Transfer dal punto di vista del Server

Visione astratta del server REST:
- Accettare HTTP request
- Parsare l'HTTP request ed estrarre le informazioni importanti
- Spedire la risposta la risposta al client

Anche il server può utilizzare HTTP library.

REST accede ai dati usando URI, i suoi 4 principi mostrano il successo e la scalabilità del protocollo HTTP:
- Resource Identification attraverso URI
- Uniform Interface per tutte le risorse
    - **GET** (Query the state, idempotent, can be cached): ottiene una rappresentazione della risorsa
    - **POST** (Create a child resource): crea una nuova risorsa
    - **PUT** (Update, transfer a new state): crea o aggiorna una risorsa
    - **DELETE** (Delete a resource): cancella una risorsa
- "Self-Describing" Message attraverso metadati e rappresentazione di risorse multiple
- Hyperlink per definire le transizioni di stato dell'applicazione e le relazioni tra risorse

Metodi HTTP aggiuntivi della uniform interface:
- HEAD: riceve solo i medati di una risorsa
- Options: controlla quali metodi sono supportati dalla risorsa
    - Ritorna un HTTP allow
    - Allow: GET, HEAD (risorsa in sola lettura)

Safety:
- GET, HEAD
- Richieste safe, solo lettura, non causano danni catastrofici
- Side effect: incremento un contatore, loggo la richiesta

Idempotenza
- GET, HEAD, PUT, DELETE
- L'esecuzione di un commando una o più volte ha lo stesso effetto

I RESTful Web Service usano risorse come figure, video, pagine web, business information, o qualunque cosa possa essere rappresentata in un computer-based system.
Obiettivo: fornire un hook attraverso cui il client può accedere queste risorse.
Proprietà e feature:
- Representations
- Messages
- URIs
- Uniform interface
- Stateless
- Links between resources
- Caching

Representations:

- REST si focalizza su risorse e come accedere a risorse
- Prima cosa da fare identificare le risorse e poi rappresentarle
- Possono essere usate diverse rappresentazioni (ad es., JSON/XML)
- La selezione del formato della risposta dipende dal tipo di cliente e da parametri della richiesta.

Una buona rappresentazione ha le seguenti caratteristiche:

- Client e server conoscono il formato della rappresentazione
- Una rappresentazione rappresenta la risorsa nella sua interezza.Se la risorsa ha rappresentazioni parziali può essere divisa in risorse figlie, con vantaggi computazionali e di performance.
- Permette di collegare diverse risorse, inserendo URI or unique ID di altre risorse nella rappresentazione

URI (Uniform Resource Identifier) = Standard Internet per naming e identification di risorse (nasce nel 1994, rivisto fino al 2005). Stringa che identifica univocamente una risorsa generica che può essere un indirizzo Web, un documento, un'immagine, un file, un servizio, un indirizzo di posta elettronica, ecc.

buone pratiche per l'uso del URI:

- Preferire sostantivi (plurali) a verbi
- Non usare spazi
- Mantenere le URI corte
- Usare uno schema di convenzione conosciuto in modo che il client possa costruirle da programma

REST richiede un'URI per ogni risorsa. RESTful service usa una human readable URI per indirizzare le risorse.

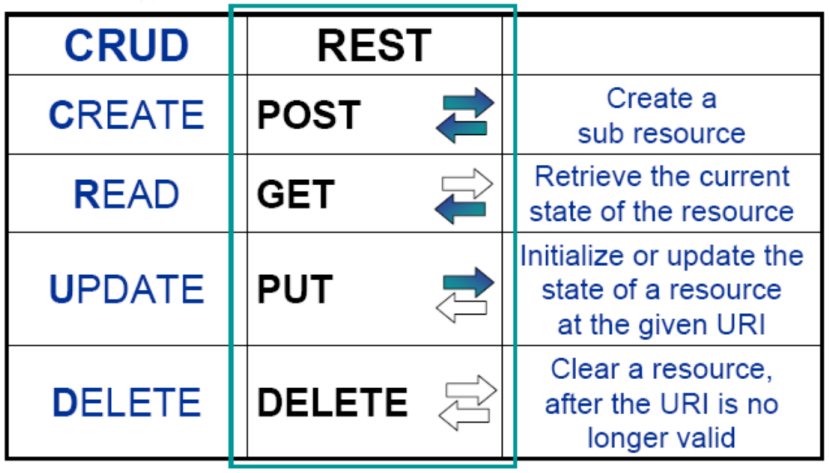
- Gerarchia basata sul concetto di directory
- URI identifica una risorsa o collezione di risorse
- L'operazione è determinata dall'HTTP method
- L'URI non dovrebbe dire nulla riguardo operazione/azione in modo da poter chiamare un'URI con diversi HTTP method per eseguire diverse operazioni.

REST permette di usare URI costruite con l'aiuto di query parameter.

Uniform Interface Principle: esempio CRUD

- Create → POST: create a sub resource
- Read → GET: retrieve the current state of the resource
- Update → PUT: initialize or update the state of a resource at the given URI
- Delete → DELETE: clear a resource, after the URI is no longer valid

Con la PUT il **client** decide il nome della risorsa, con la POST il **server** decide il nome della risorsa.



CRUD	REST	
CREATE	POST	Create a sub resource
READ	GET	Retrieve the current state of the resource
UPDATE	PUT	Initialize or update the state of a resource at the given URI
DELETE	DELETE	Clear a resource, after the URI is no longer valid

POST vs PUT:

- POST usata per creare risorse subordinate, PUT usata per creare/modificare risorse
- Con la PUT il client decide il nome della risorsa.
- Con la POST il server decide il nome della risorsa per assicurare che l'id sia univoco.

Statelessness = RESTful service è stateless, non mantiene lo stato dell'applicazione per il client. Stateless service è più facile da deployare, mantenere e scalare, fornisce miglior tempo di risposta.

Nonostante sia semplice identificare e navigare attraverso un servizio RESTful, un minimo livello di documentazione è molto utile.

High REST vs Low REST:
High REST = Utilizzo di "nice" URI viene raccomandato. Utilizzo di quattro verbi: GET, POST, PUT e DELETE. Risposte codificate come Plain Old XML.
Low REST = http GET per richieste idempotenti, POST per tutte le altre. Risposte in qualsiasi MIME type (ad esempio, XHTML).

REST: Vantaggi

- Semplicità
- http/POX è ubiquo (attraversa i firewall)
- Interazione Stateless/sincrona
- Scalabilità
- Facilità di utilizzo e adozione (infrastruttura leggera)
- Utilizzato da tutte le applicazioni web 2.0

REST: Svantaggi

- Confusione (High REST vs Low REST)
- Il mapping della semantica sincrona del REST-style al di sopra dei sistemi di backend esistenti crea disallineamenti nel design
- Non può rilasciare enterprise-style "-ilities" oltre http/ssl
- Difficile identificare e localizzare le risorse in maniera appropriata in tutte le applicazioni

- Apparente mancanza di standard
- Descrizione semantica/sintattica molto informale

RESTful Web Service: Design Methodology
- Identificare risorse che possono essere esposte come servizi (ad es., yearly risk report, book catalog, purchase order, open bug, polls and votes)
- Definire “nice” URL per indirizzare i servizi
- Capire il significato di GET, POST, PUT, DELETE su una data URI di una risorsa
- Definire e documentare la rappresentazione delle risorse
- Modelli di relazione (ad es., contenimento, riferimento, transizione di stato) tra risorse con hyperlink che possono essere seguite per ottenere più dettagli (o effettuare transizioni di stato)
- Implementare e deployare un Web server T
- estare i servizi

Approfondimenti

POST vs GET:
- GET è un’operazione read-only: può essere ripetuta senza influenzare lo stato delle risorse e può essere cachata
- POST è un’operazione read-write: può cambiare lo stato della risorsa e provocare alcuni side effect lato server

Come creare una risorsa (inizializzarne lo stato)?
PUT/resource/{id}
Problema: come assicurare che {id} sia univoco?
POST /resource
201 Created
Location: /resource/{id}
Risorse create da client multipli e concorrenti Soluzione: lasciare al server il dovere di calcolare l’id univoco

Negoziazione del contenuto = il client elenca una lista di formati che capisce. Il server sceglie la più appropriata per la risposta.
Negoziazione del contenuto **forzata** = URI specifica punta ad un formato di rappresentazione specifico usando il postfix.

Idempotent vs Unsafe
richieste idempotenti = possono essere processate diverse volte senza side effect (lo stato del server non cambia).
GET/book
PUT/order/x
DELETE/order/y
 Se qualcosa va male (server down, server internal error), la richiesta può essere semplicemente rimandata fino a quando il server non ritorna attivo.

richieste unsafe = modificano lo stato del server e non possono essere ripetute senza ulteriori effetti. Richieste unsafe richiedono gestione delle situazioni eccezionali. In alcuni casi le API possono essere ridefinite per usare operazioni idempotenti