Pagina Web: Documento digitale identificato da una URL (Uniform Resource Locator) e normalmente visualizzato mediante un browser che deve scoprire dove si trova, chiedere che ci venga inviato, controllare e verificare ciò che riceviamo e mostrarlo.

Internet è una sistema di dispositivi elettronici interconnessi IP: Internet Protocol
Il World Wide Web (WWW) è uno dei modi con cui le informazioni possono essere condivise. il WWW è un insieme di pagine web ed altri documenti collegati tra loro usando i collegamenti ipertestuali (link) Il protocollo principale per scambiare documenti è HTTP (HyperText Transfer Protocol)

All'uomo è comodo dare un nome ai server, ai computer serve un indirizzo numerico per comunicare con le altre macchine. Esistono dei computer che traducono nomi in indirizzi IP (Domain Name Server)

Server Web: Computer che rispondono a delle richieste di documenti
In realtà chi offre il servizio è un programma: il server web, Aspetta una richiesta, cerca un documento e lo invia al richiedente, Il server web deve parlare HTTP, spesso lo chiamiamo Server HTTP, I computer server non sono particolari, ogni computer può diventare un server

Client Web: I software che fanno richieste HTTP ai server si chiamano client, I più comuni client web sono i Browser
- Richiesta di un documento: L'utente inserisce un indirizzo, Il browser chiede un documento al server, Il server glielo spedisce, Il browser lo mostra all'utente

I browser parlano HTTP come i server web. Sia le richieste che le risposte sono gestite mediante HTTP. Richiedono documenti di qualsiasi tipo: testo, immagini, video, audio, script, etc. I browser visualizzano i documenti richiesti al server in un finestra.

HTML
Hypertext Markup Language (HTML): markup semantico per il contenuto delle pagine web
Linguaggio per strutturare documenti ipertestuali. E' uno standard del W3C. Abbiamo dei tag che possono essere annidati e avere degli attributi come ad esempio “class”
Gli elementi e quindi i tag formano un albero.

Una pagina HTML ha:
L'elemento <html> è la radice dell'albero e contiene tutti gli altri
L'elemento <head> fornisce informazioni circa il documento: Contiene metadati relativi alla pagina, descrizione, charset, autore, etc. Generalmente il contenuto contenuto all'interno di questo tag non viene visualizzato nella pagina web. Contiene il <title> che definisce il titolo della pagina, visualizzato dal browser nella toolbar, è usato dai motori di ricerca. Tale elemento è obbligatorio nelle pagine, ci può essere un solo elemento <title>.
L'elemento <body> contiene il contenuto visibile

La dichiarazione <!DOCTYPE> serve a dire al browser il tipo di documento HTML, ci sono diversi tipi di documento. Il browser deve sapere il tipo e la versione dell'HTML da visualizzare.

Lo scopo dell'HTML5 è quello di aggiungere significato e struttura al contenuto, non è pensato per fornire istruzioni su come il contenuto deve essere presentato.
Markup semantico: Scegliere l'elemento HTML che fornisce la miglior descrizione del contenuto (Es. il titolo più importante all'inizio del documento deve essere circondato con tag h1, senza preoccuparsi di come viene visualizzato di default)

Block element – Dopo alcuni elementi il browser inserisce un accapo e spazio intorno
Inline element – elementi che non iniziano in una nuova linea ma rimangono nel flusso del paragrafo

Le URL messe all’interno degli attributi src, href, .. sono fatte di tre parti – Il protocollo – Il nome del server – Il path della risorsa, (relative definite rispetto alla posizione attuale del file e assolute partono dalla root del server)

CSS (CASCADING STYLE SHEETS) Standard W3C, Definisce la presentazione del documento HTML (o in generale XML) (cioè come viene presentato) CSS è un linguaggio indipendente con la sua sintassi

```
selettore {
proprietà: valore;
}
```

Possiamo avere:
External style sheet, collegato mediante il tag <link rel=”stylesheet” ...>
Internal style element <style>...</style> all’interno del body

Relazioni nel DOM
Il DOM (Document Object Model) è una rappresentazione strutturata del contenuto di una pagina web. Possiamo immaginarlo come un albero gerarchico, in cui ogni parte del

documento HTML (come paragrafi, titoli, link, immagini, ecc.) è rappresentata da un nodo. L'elemento radice dell'albero è chiamato document, ed è l'oggetto che rappresenta l'intera pagina web caricata nel browser. Da questo oggetto partono tutti gli altri elementi: html poi body, head, e così via.
Grazie al DOM, JavaScript può interagire con il contenuto della pagina, ad esempio per modificare il testo, aggiungere elementi, cambiare stili, o rispondere a eventi (come un click)

I selettori CSS sono le "chiavi" che dicono al browser a quali elementi HTML applicare un certo stile.
1. Parent (genitore) : Elementi direttamente sopra.
2. Child (figlio) : Discendenti diretti e viceversa si dice genitore (parent).
3. Descendant (discendente) : Gli elementi contenuti in un elemento sono i suoi discendenti.
4. Sibling (fratello) : due elementi che stanno allo stesso livello, dentro lo stesso genitore.
5. Ancestor (antenati): Gli elementi sopra nell'albero.

Discendente (spazio)
```
div p {
color: blue;
}
```
Colpisce tutti i <p> dentro un , anche se sono annidati più in profondità. È il più "largo" e generale.

Figlio diretto (>)
```
div > p {
list-style-type: square;
}
```
Colpisce solo i <p> che sono figli diretti di , non altri.

Fratello immediato (+)
```
h1 + p {
margin-top: 0;
}
```
Colpisce il primo <p> subito dopo un <h1> , se sono fratelli diretti.

Fratelli generici (~)
```
h1 ~ p {
color: red;
}
```
Colpisce tutti i paragrafi fratelli successivi di un <h1>.

Una pseudo-classe viene utilizzata per definire uno stato speciale di un elemento.
```
selector:pseudo-class {
property:value;
}
```

es:
```
a:visited{
color: red;
```

}
cosi come :hover, :first-child :last-child

Uno pseudo-elemento viene utilizzato per dare uno stile alle parti specifiche di un elemento. Disegna la prima lettera o riga di un elemento, Inserire un contenuto prima o dopo un elemento
selector::pseudo-element {
property:value;
}

es:
p::first-letter {
color: #ff0000;
font-size: xx-large;
}
cosi come ::after ::first-line ::selection

È possibile impostare lo stile si elementi HTML con attributi o valori di attributo specifici, è possibile selezionare parte del valore:
[attribute]{
property: value;
}
[attribute=value]{
property: value;
}

es:
img[alt]{
background-color: grey;
}
a[target="_blank"]{
color: red;
}

Alcune proprietà sono ereditate dai discendenti:
Gli stili relativi ai font e al testo sono ereditati dai descendant (font-size, family, color, visibility)

I conflitti di dichiarazione sono la normalità, domina quello quello inserito per ultimo nel css, Se non voglio che una regola venga sovrascritta la posso dichiarare !important.

Il Cascade "cascata" è un algoritmo che definisce come combinare valori di proprietà provenienti da fonti diverse.
Stili:
– browser - user-agent stylesheets
– author - quello che scriviamo noi
– reader - the user of the browser, may have a custom style sheet to tailor its experience.

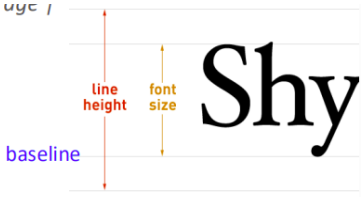
Calcolo della specificità:
Ad ogni dichiarazione è attribuita una specificità misurata con quattro valori [a,b,c,d], ogni valore è indipendente dagli altri, "a" è il valore più importante e "d" il meno importante, si confrontano i valori più importanti e se uguali si passa a quelli successivi altrimenti termina
Es. 0,1,0,0 è più specifico di 0,0,5,5

Calcolo dei valori
a. 1 se la dichiarazione è inline, 0 altrimenti
b. numero di selettori id
c. numero di selettori di classe, attributo o pseudo-classe
d. numero di selettori elemento o pseudo elemento
html body div#pagewrap ul#summer-drinks li.favorite
specificità = 0 2 1 5

Per il font del testo: tutti i font cominciano con la lettera maiuscola, tranne i font generici (sans-serif), sono separati da virgole, se il loro nome contiene spazi vanno fra virgolette
Sans-serif è la famiglia di font senza “grazie” mentre il serif ha le “grazie”
Monospace ogni lettera occupa lo stesso spazio.

Non tutti i font sono disponibili, dipendono dal sistema operativo se sono installati nel pc, si specifica quindi il font desiderato e delle opzioni alternative simili. Possiamo distinguere principalmente tra font di sistema e web font.
I font di sistema sono preinstallati sui dispositivi degli utenti e utilizzati direttamente dal sistema operativo. Esempi comuni includono Arial, Times New Roman e Verdana.
I web font sono caratteri font scaricati da un server al momento del caricamento della pagina web. (grazie ad essi è possibile utilizzare font personalizzati)
Il font stack è un elenco di font specificati nella proprietà font-family in ordine di preferenza. Se il primo non è disponibile, il browser prova ad usare il secondo, poi il terzo etc…

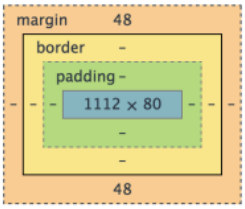
Unità di misura relative:
- px pixel
- em a unit of measurement equal to the current font size
- % relative to the parent element
- rem root em, equal to the em size of the root element (html)
- vw viewport width unit, equal to 1/100 of the current viewport (browser window) width
- vh viewport height unit, equal to 1/100 of the current viewport height
- vm viewport minimum unit, equal to the value of vw or vh, whichever is smaller



font-family: font utilizzato, e.g. Helvetiva, Arial
font-size: dimensione del testo, e.g. 60px, 3em
font-weight: Quanto è grassetto il testo, e.g. bold
font-style: Che stile è il testo e.g. italic

line-height: spaziatura tra le righe, e.g. 2em
color: colore del testo, e.g. #000. #abedef
text-decoration: imposta effetti sul testo, e.g. underline, overline, none
text-align: Come è allineato il testo, e.g. center
letter-spacing: spaziatura tra le lettere, e.g. 5px
text-indent: rientro della prima riga, e.g. 2em
text-transform: trasforma il testo, e.g. upper-case, lowercase, capitalize
vertical-align: Allineare rispetto alla linea di base, e.g. text-top

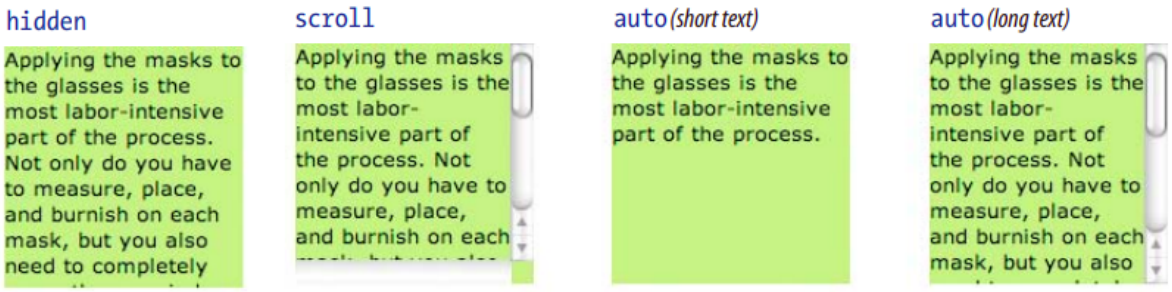
Sizing
Ogni elemento html è contenuto in un box rettangolare. Le proprietà css sono applicate al box – content (width, height), padding, borders, e margins



box-sizing:
border-box – include nel size padding e border
content-box – solo contenuto

Margin
Gli elementi inline ignorano il margine top e bottom.
Ogni elemento ha dei margini, due margini adiacenti (bottom e top di due elementi) collassano in uno solo: il massimo dei due. Non collassano se elemento float o absolute. Si può impostare un valore negativo al margine per sovrapporre

Overflow



Background
background-color, opacity, background-image, background-repeat (per la ripetizione dell immagine), background-position, background-attachment (l immagine se scrolla nello sfondo)

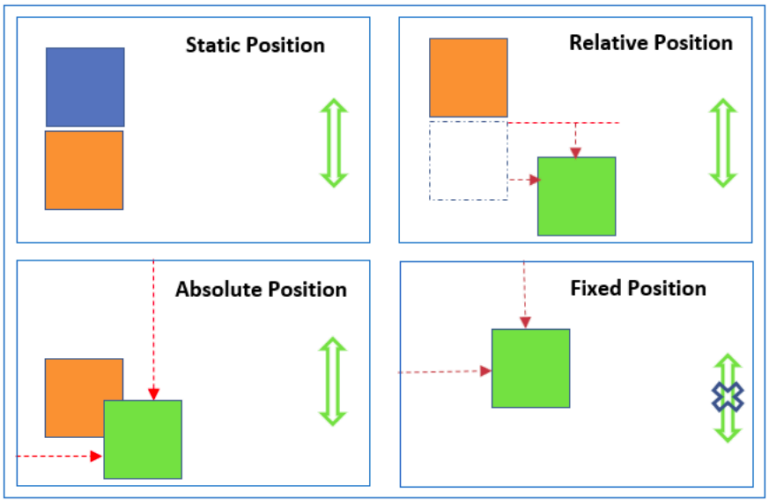
Liste
list-style-type: (decimal, latin, roman, disc, circle), list-style-position (se il numero sta dentro o fuori la lista inside o outside), list-style-image (mettere immagine al posto di disc)

Punti specifici di una pagina
Identificazione dell'ancora, ogni id può essere usato come ancora
<h1 id="startH">H</h1>
Si usa come # ed il nome dell'identificatore
... F | G | H | I | J ...

se si clicca sul link si andrà sull'ancora, se l'id non è presente funziona comunque come un link normale
il link target="blank" dice di aprire il link in una nuova finestra. Si possono fare anche link a mail (mailto:email) o link a telefono (tel:nr)

Specifici dei link
:link – link non visitati
:visited – link già visitati dall'utente (cache del browser)
:focus – l'elemento è selezionato
:hover – il mouse è sopra l'elemento, deve essere dichiarato dopo :link e :visited
:active – nel momento del click, deve essere dichiarato dopo :hover

Posizione Elementi
tag position
static: posizione predefinita, viene posizionato secondo il normale flusso
relative: viene posizionato secondo il normale flusso del documento ma è posizionabile rispetto al suo posto originale (top left bottom right),
absolute: l'elemento viene rimosso dal flusso del documento ed è posizionabile rispetto al suo contenitore (sempre tramite top left …)
fixed: rimosso dal flusso del documento e posizionabile rispetto al viewport

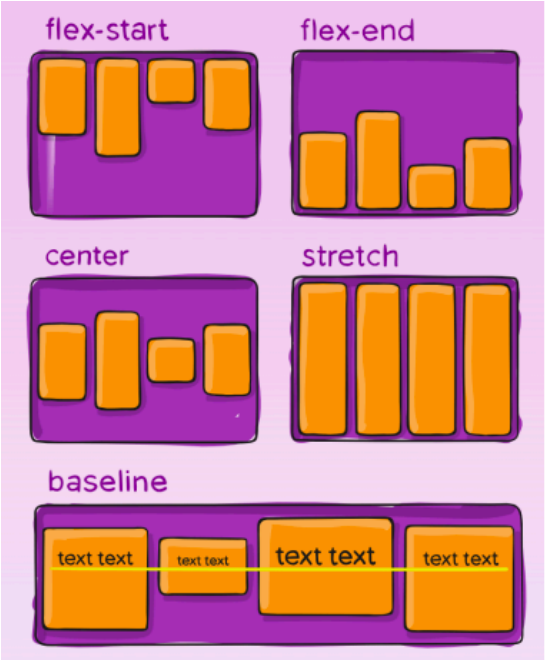


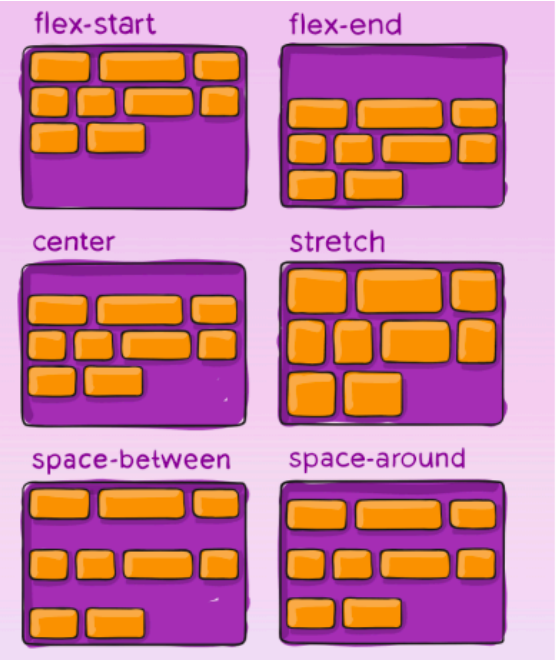
tag top, right, left, bottom
Specifica la distanza rispetto all'elemento o blocco contenitore, deve essere contenuto in un contenitore con position non static (o il body)

tag z-index
per gestire sovrapposizioni

tag float
muove un elemento tutto a destra o tutto a sinistra (right o left) permettendo agli altri elementi di circondarlo. Si staccano dal flusso normale ma influenzano il contenuto dei blocchi intorno. Sono contenuti nell'area del contenuto dell'elemento che li contiene. I margini sono mantenuti,
Nell'elemento contenitore alcune volte devo dichiarare overflow auto o hidden per far si che si allunghi in modo da contenere il float.

FlexBox Display
display: flex (o inline-flex)
flex-direction: per la direzione del flex se a righe o colonne
flex-wrap: per capire se forzare tutto all'interno della stessa riga o poter andare anche a capo
justify-content: per la posizione orizzontale, come ad esempio centrarlo, spaziarlo equivalentemente (space-between), farlo partire dal fondo e cosi via
align-items & align-content: il primo per allinearli verticalmente il secondo per allineare tutto il contenuto in un certo modo (flex-end cerca di mettere tutto alla fine, center lo centra sia verticalmente che orizzontalmente)
order: ne indica l ordine
align-self: allinea un singolo elemento in un determinato modo
flex-grow: Indica quanto un elemento può crescere per occupare lo spazio disponibile.
flex-shrink: Indica quanto un elemento può restringersi se non c'è spazio.
flex-basis Definisce la dimensione iniziale dell'elemento.
flex: Shorthand per flex-grow, flex-shrink e flex-basis.





Menu
navigazione verticale o orizzontale si applica uno stile ad una lista ul

Layout delle pagine
Nel web design, il layout determina come i contenuti si distribuiscono nella pagina può essere:
Fluid: proporzionale alla larghezza del browser
Fixed: larghezza indipendente dalla finestra del browser

Layout Fluido
Vantaggi: Sembra più vicino alla natura dei borwser, Elimina spazi potenzialmente vuoti, L'utente può controllare la pagina, Non ho scrollbar orizzontali.
Svantaggi: Se il monitor è grande ho righe troppo lunghe, limitare la dimensione, Non è facile predire la posizione degli elementi e l'effetto grafico finale, Problemi nei browser piccoli, I conti sono un poco più complicati

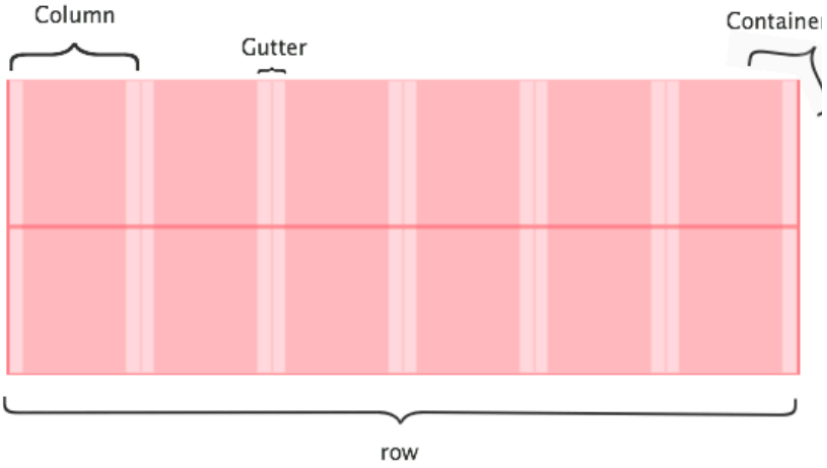
Layout Fixed
Vantaggi: Controllo del numero delle righe, è semplice da realizzare, è una visualizzazione comune per i desktop.
Svantaggi: se la finestra è piccola la parte destra viene coperta, se lo schermo è grande ho spazi a dx ed sx da gestire, se ho carattere grandi ho problemi di righe corte, la pagina non è controllabile dall'utente

Responsive Web Design
Fornire layout diversi per schermi diversi. Il layout si adatta in automatico alla dimensione dello schermo. Stessi contenuti, ma presentazione ottimizzata per ogni dispositivo. Un solo file HTML, ma con CSS variabili grazie alle media query
Il CSS responsive di può scrivere in due modi: Mobile First o Desktop First
In entrambi i casi dobbiamo:
1. Controllare il viewport:
da inserire nell'head:
<meta name="viewport" content="width=device-width, initial-scale=1">
2. Gestire Layout con media queries:
definisco degli stili per determinati media e device, in particolare si usano quando certe condizioni sono attive, si possono combinare tra di loro.
@media (max-width: px){proprietà}
@media (min-width: 30em) and (orientation: landscape) { ... }
Mobile First e Desktop-First
L'idea è quella di scrivere il CSS per gli schermi piccoli, poi espandere mediante media query con min-width per schermi più grandi, viceversa con desktop-first che si usa max-width
Breakpoints
I breakpoint sono punti su una scala ideale di larghezza del viewport in cui si verifica una qualche modifica al layout della pagina. (grandezza della pagina tipo 800px)
I breakpoint si definiscono con valori numerici nelle media query.
3. Usare dei Media "fluidi": usare quindi max-width e min-width al posto di width generale

In CSS si possono usare delle custom properties che possono essere globali o locali, globali le metto nel root altrimenti usando la stessa sintassi le inserisco nei singoli selettori in modo che lo possa usare nei figli
:root{

-- blue: #444;
}

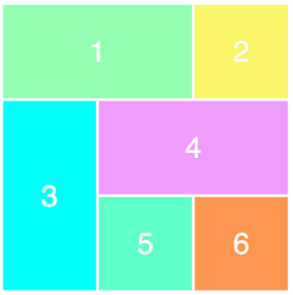
Grid Layout e Bootstrap



si metterà display: grid e i tag grid-template-columns e grid-template-rows
grid-template-rows: 320px auto 320px; (tre righe da tot px)
grid-template-columns: 100px 100px 100px 100px; (quattro colonne da tot px)
il posizionamento si fa con grid-column-start e grid-column-end o semplicemente grid-column e si vanno a indicare quante righe o colonne voglio occupare o da dove partire

```
.item1 {
        grid-column-start: 1;
        grid-column-end: 3;
}

.item3 {
        grid-row-start: 2;
        grid-row-end: 4;
}
.item4 {
        grid-column-start: 2;
        grid-column-end: 4;
}
```



Bootstrap (framework)
Facile da usare, chiunque abbia anche solo una conoscenza di base di HTML e CSS può iniziare a utilizzare Bootstrap
Funzionalità responsive
Il CSS responsive di Bootstrap si adatta a telefoni, tablet e desktop con approccio mobile-first
Compatibilità con i browser
Bootstrap è compatibile con tutti i browser moderni (Chrome, Firefox, Internet Explorer, Safari e Opera)

Form
Un insieme di elementi in una pagina web con cui l'utente interagisce per inviare informazioni ad uno script. All interno del tag form abbiamo vari input.
Un elemento <input> può essere visualizzato in molti modi, a seconda dell'attributo type:
text: Campo di testo a una riga
password: Campo di testo con caratteri nascosti

email: Campo per inserire un indirizzo email (valida formato @)
number: Campo per numeri (può avere min/max)
tel: Campo per numero di telefono
url: Campo per un indirizzo web
date: Selettore per una data (con calendario)
time: Selettore per un orario
checkbox: Casella da spuntare (scelte multiple)
radio: Pulsante di scelta esclusiva tra opzioni range
file: Campo per caricare file
submit: Pulsante per inviare la form
reset: per resettare tutto il form
hidden: inviare informazione che non provengono dall'utente (es. referral)

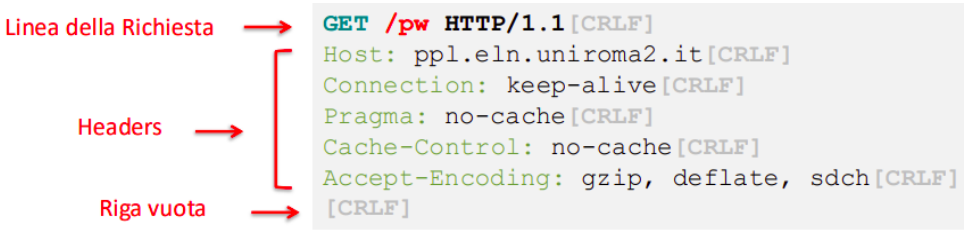
Ogni elemento nel form definisce una variabile che ha un nome, l'utente interagendo con il form varia il contenuto delle variabili che vengono inviate al server
Attributo name: id della variabile
Attributo value: valore di default (opzionale)

Il tag definisce un'etichetta per gli elementi form. Due modalità per associarli: Circondare l'elemento da descrivere (implicita) o usare l'attributo for che deve essere uguale all'attributo id di <input> (esplicita).
L'attributo action contiene la URL dello script che elaborerà i dati Se non presente si assume che il suo valore sia quello della pagina corrente.
L'attributo method Specifica la modalità di invio dei dati (es. GET per i dati nell'url o POST per i dati nel body della richiesta)

tag textarea per identificare un area di input testuale multiriga
tag select per un menu a tendina dove le opzioni vengono messe all'interno delle varie option (come per la lista ul e li)

Tabelle
tag <table> per la creazione di tabelle con th per gli header, tr per le righe e td per le signole celle

HTTP
HTTP è un protocollo che usano gli applicativi per comunicare, tramite scambio di messaggi richiesta e risposta. È Stateless (senza stato/memoria) cioè ogni richiesta è indipendente da quelle precedenti e si conclude al momento della chiusura della connessione, può trasportare ogni tipo di dato.
HTTP è l'insieme di regole che definisce il formato ed il contenuto della conversazione fra web client e server, prima dello scambio di messaggi tra client e server bisogna prima aprire la connessione tramite TCP.
Formato Richiesta:

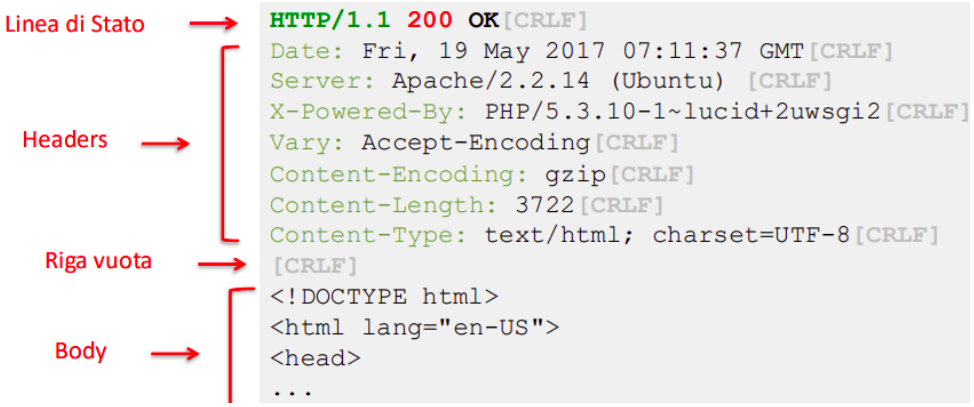


Nelle richieste POST è presente anche il body, nel quale ci sono i dati da inviare al server, di solito in formato JSON (JS Object Notation)

I Metodi di richiesta che possiamo avere in HTTP sono:
GET: Recupera le informazioni specificate dalla URI di richiesta, la risposta può essere memorizzata in cache. I dati vengono inviati nella URL come query string, è semplice e visibile nella barra del browser ma ha un limite di lunghezza
HEAD: la risposta deve essere identica a quelle che restituirebbe il GET, il server non deve restituire il body del messaggio nella risposta, la risposta può essere memorizzata in cache
POST: Usato dal client per inviare dati al server, dati che sono passati nel body delal richiesta. Più sicuro (dati non visibili nella url) vengono inviati nel corpo della richiesta, è adatto per form, ma non memorizzabile in cache.
PUT: usato dal client per modificare o creare (se non esiste gia quella risorsa) una risorsa sul server.
PATCH: per modifiche una risorsa a cui la URI di riferisce
DELETE: cancellazione dal server di origine della risorsa identificata dalla URI di richiesta.
OPTIONS: Rappresenta una richiesta di informazioni riguardanti le opzioni di comunicazione disponibili per una URI

Abbiamo vari Headers come:
User-Agent: identifica il client (browser, app, ecc.).
Accept: tipi di contenuto accettati (text/html, application/json).
Authorization: per l'invio di credenziali di accesso.
Accept-Language: preferenze linguistiche dell'utente

Formato Risposta:



Risposte HTTP:
1xx Informational: indicano che la richiesta è stata ricevuta e il processo è in corso.
2xx Success: confermano che la richiesta è stata ricevuta, capita ed elaborata con successo.
3xx Redirection: segnalano che il client deve effettuare ulteriori operazioni per completare la

richiesta.
4xx Client Error: indicano che la richiesta contiene errori da parte del client (ad esempio, una risorsa inesistente).
5xx - Server Error: segnalano che il server ha incontrato un problema interno durante l'elaborazione della richiesta.

Esistono vari Headers come:
Server: indica il software del server (es. Apache, nginx).
Content-Language: lingua della risposta.
WWW-Authenticate: chiede autenticazione (es. Basic realm="Area Riservata").
Location: usato per le redirect

Altri Header del Body:
Content-Type: tipo di contenuto (es. application/json, text/html).
Content-Length: lunghezza in byte del corpo.
Content-Encoding: compressione usata (gzip, deflate).
Last-Modified: ultima modifica

I dati possono essere inviati al server tramite:
Query string (con GET)
Form HTML (con POST)
Fetch/AJAX in JavaScript (API REST)

HTTPS = HTTP+Security
I messaggi vengono cifrati, si aggiunge il protocollo TLS/SSL porta 443 non più 80, quindi prima della comunicazione vera e propria si fa il TLS handshake.
Per la cifratura i certificati si installano nel server, includono una chiave pubblica. I messaggi vengono cifrati con la chiave pubblica e mandati al server che li decifrerà con la chiave privata (da non condividere)

JS
Nel front-end, JavaScript serve per:
• Interagire con la pagina HTML (DOM)
• Gestire eventi (click, input, caricamento, ecc.)
• Manipolare stili, classi e contenuti dinamicamente
• Comunicare con server (es. via fetch)
• Controllare il flusso logico e rispondere alle azioni dell'utente.

Un transpiler (da transformation compiler) è uno strumento che permette di tradurre codice da un linguaggio ad un altro linguaggio simile. Questi sono molto utilizzati in particolare per tradurre codice Javascript moderno in codice di versioni meno recenti, in questo modo è possibile rendere il nuovo codice compatibile anche con ambienti che non supportano ancora le nuove funzionalità di javascript (come vecchi browser). I Transpiler permettono anche il passaggio da altri linguaggi a JS come ad esempio a partire da TypeScript..
Un polyfill è una libreria js (piccolo script js) che adattano del codice HTML, JS, CSS per renderlo retrocompatibile con standard "vecchi" dei browser.

Mentre il transpiler traduce il codice prima dell'esecuzione, il polyfill lo esegue in runtime aggiungendo la funzionalità mancante.

Quali sono le caratteristiche di JavaScript?
- JavaScript è dinamico (dynamic): javascript non è compilato, ma interpretato direttamente dal browser tramite un motore javascript.
- Case-Sensitive: JS fa distinzione tra maiuscole e minuscole (es. Variabile != variabile)
- Loosely typed (tipizzazione debole): non è necessario specificare il tipo delle variabili.
- Altre caratteristiche: Event-Driven (reagisce ad eventi es. click, submit etc…), asincrono (supporta codice non bloccante con callback, Promise, async/await)

Il garbage collector è si occupa di liberare automaticamente la memoria occupata da variabili, oggetti, funzioni che non sono più utilizzati, quindi quando non abbiamo riferimenti ad essi.
Il Garbage Collector in JS è automatico ma non infallibile, esistono infatti casi in cui non riesce a liberare la memoria anche se l'oggetto non serve più logicamente. Questi casi si chiamano "memory leak" e si verificano quando qualcosa in memoria resta referenziato per errore (Riferimenti Circolari).

Sia let che const servono per dichiarare variabili, l'unica differenza è che con const la variabile non può essere riassegnata mentre con let si.
Tuttavia con const si può comunque modificare il contenuto di oggetti e array, infatti const blocca il riferimento ma non il contenuto dell'oggetto o array.
In js si può convertire una variabile (stringa - intero) o in modo automatico o tramite il parseInt()
Mentre let ha come scope l'enclosing block più vicino (se lo definisco in un blocco for sarà visibile solo in quel blocco), var ha come scope il functional block più vicino (se lo definisco in un blocco for sarà visibile comunque in tutta la funzione).
(=== permette di vedere anche l'uguaglianza del dato non solo il valore)

Lo scope rappresenta la visibilità di una variabile o funzione, cioè dove nel codice posso accedere a quella variabile e dove no. I tipi principali di scope sono:
- Global Scope: si tratta di una variabile definita fuori da tutte le funzioni o blocchi, ed è quindi visibile ovunque nel file js.
- Function Scope: le variabili definite dentro una funzione sono visibili solo lì dentro
- Block Scope: le variabili dichiarate con let o const dentro { } (come if, for…) sono visibili solo dentro quel blocco.

Arrow functions è un sistema più sintetico di specificare una funzione (funzione() => {})

Un oggetto è una lista di di coppie "proprietà":"valore", racchiuse in { } (dichiarazione diretta let x = {}, sennò anche tramite costruttore), le proprietà possono essere solo stringhe o simboli, un valore può essere un tipo primitivo, un array, un altro oggetto o una funzione.
Quando copiamo un oggetto A e B faranno dei riferimenti allo stesso oggetto, dobbiamo prima istanziare a e dopo uguagliarlo a b
Un metodo in js e più in generale in programmazione a oggetti è una funzione che appartiene ad un oggetto (ciò che l'oggetto può fare).
This puo' essere utile nei metodi riferirci ad altre proprietà dell'oggetto, a differenza

this nel lato function rappresenta l'outer scope (l'oggetto che lo contiene).
Se la funzione viene chiamata come metodo di un oggetto (come in questo caso persona.saluta()) -> this è proprio persona, fa riferimento a quell'oggetto.
Invece, fuori da un oggetto this normalmente sarà "window" (si riferisce all'oggetto globale) oppure sarà "undefined" nel caso si usi la strict mode.

Per creare oggetti uguali o simili possiamo usare delle funzioni come il costruttore esso rappresenta la funzione speciale utilizzata per creare oggetti con la stessa struttura.
Quando viene chiamato un costruttore con new: Viene creato un oggetto vuoto e assegnato a this che punta a questo nuovo oggetto creato, all'interno della funzione, si settano le proprietà dell'oggetto e gli si imposta il prototipo e una volta terminata l'esecuzione della funzione, l'oggetto che è stato riempito viene restituito (implicitamente) (grazie all'uso della parola chiave new) e viene assegnato alla variabile.

Ogni funzione costruttore in JS ha una proprietà di default chiamata .prototype. Questo oggetto verrà usato come prototipo per tutti gli oggetti creati con new da quella funzione costruttore, JS fa questo per evitare la duplicazione dei metodi. Quando chiamiamo una proprietà di un oggetto, prima si cerca tra le proprietà dell'oggetto, poi tra le proprietà del prototipo e infine tra le proprietà del prototipo del prototipo etc... [Prototype chain].

Prototipi di funzione: È l'istanza di un oggetto che diventerà il prototipo per tutti gli oggetti creati usando la funzione come costruttore
Prototipi di oggetto: È l'istanza dell'oggetto dal quale l'oggetto è ereditato

Possiamo definire funzioni dentro altre funzioni. La funzione "nested" può accedere allo scope della funzione che la include. L'inner function può accedere allo scope delle outer functions. L'outer function non può accedere allo scope delle inner functions.
Una closure è una funzione che ha accesso alle variabili del suo scope esterno anche dopo che lo scope esterno è terminato.

```
function salutatore(name) {
    let text = 'Ciao' + name; // Local variable
    let diCiao = function() { alert(text); }
    return diCiao;
}
let s = salutatore('Lorenzo');
s(); // alerts "Ciao Lorenzo"
```

s non memorizza solo il return della funzione salutatore, ma anche le variabili appartenenti al suo scope. Uno dei principali utilizzi delle closure è quello di simulare la programmazione ad oggetti. Non avendo modificatori di accesso come public o private, possiamo incapsulare tutto all'interno di una closure.

Il nodo document ha una serie di proprietà e metodi predefiniti che permettono di accedere e manipolare i vari elementi.
Ad esempio:
- document.getElementById("titolo") permette di selezionare un elemento con id "titolo".
- document.body restituisce l'elemento della pagina.
- document.querySelector(".classe") cerca il primo elemento con quella classe.

In sintesi, il DOM è la struttura che collega JavaScript alla pagina HTML, e ci permette di modificarla dinamicamente

Il browser quando legge <script ...> interrompe la creazione della pagina: scarica lo script e lo esegue. Se mettiamo gli script in cima alla pagina, gli script possono non vedere gli elementi della pagina, gli script possono rallentare il rendering. Se mettiamo gli script in fondo potremmo attendere troppo.
L'attributo defer dice al browser di NON aspettare lo script, che verrà scaricato in background ed eseguito quando il DOM è costruito
Mentre async è come defer, ma lo script viene eseguito appena è pronto.

Cross Origin Resource Sharing (CORS)
Le CORS (Cross-Origin Resource Sharing) sono un meccanismo di sicurezza fondamentale, servono per controllare l'accesso tra risorse che provengono da origini diverse, ad esempio la condivisione di risorse tra due siti diversi.
È uno standard W3C per condividere risorse tra domini diversi. Prevede la richiesta di autorizzazione client e l'autorizzazione server.
In generale, un browser permette normalmente agli script di una pagina web di accedere ai dati contenuti in un'altra pagina solo se hanno la stessa origin, dove una origin è definita dal protocollo (es. HTTP, HTTPS), il dominio (es. esempio.com) e la porta (es. :3000).
Utilizzando CORS si proteggono i dati degli utenti e del server da richieste non autorizzate fatte da pagine malintenzionate.
CORS si basa sull'uso di header HTTP che vengono aggiunti alle richieste e alle risposte.
Esistono due casi principali:
- Caso 1: Richieste semplici. Una richiesta è considerata semplice se usa solo metodi GET, POST, HEAD e usa solo header standard del tipo Accept, Accept-Language, Content-Language, Content-Type con solo alcuni valori specifici.
- Caso 2: Richieste preflight. Se la richiesta non è semplice (PUT/DELETE) dove viene inviata una richiesta di tipo OPTIONS, detta richiesta preflight, prima di mandare la richiesta vera e propria.

JS Assíncrono e Síncrono
Sincrono: Viene eseguito riga per riga, nell'ordine in cui è scritto. Ogni istruzione blocca l'esecuzione del programma finché non è completata. Diventa un problema quando se un'operazione richiede tempo, tipo la lettura da un file.
Asincrono: Permette a certe operazioni di avviarsi e continuare in background, senza bloccare il resto del codice.
È importante perché se dovesse aspettare ogni operazione lunga (es. richiesta di rete, accesso a file, attesa dell'utente), l'intera pagina resterebbe bloccata.

Event Loop
L'event loop è un meccanismo che controlla l'ordine in cui il codice viene eseguito.
Il suo compito è gestire l'esecuzione del codice, delle callback, delle promises di tutto ciò che è asincrono.
Essendo single-threaded, eseguo solo una cosa alla volta, ed è qui che entra in gioco l'evento loop. Per parlare di evento loop dobbiamo prima parlare dei spazi di esecuzione:
1. Call Stack: è dove il motore JS esegue il codice. Ogni funzione chiamata viene messa qui e poi tolta quando ha finito.

2. Web API/Node API: sono funzioni fornite dall'ambiente e vengono delegate fuori dal thread principale.
3. Task Queue (Callback Queue): in questa coda vanno a finire le callback o le funzioni da eseguire dopo che le WEB APIs hanno finito.
4. Microtask Queue: una coda speciale usata principalmente per le promises. Le funzioni in questa coda hanno priorità più alta rispetto alla task queue.

Dunque, l' evento loop funziona nel seguente modo:
1. Prende il primo task dalla call task ed esegue.
2. Se trova un'operazione asincrona, la passa alle WEB APIs.
3. Quando l'operazione è completata (dopo il tempo indicato, o dopo una risposta), la relativa funzione viene messa nella task queue o nella microtask queue.
4. L'event loop aspetta che il call stack sia vuoto. Appena è vuoto esegue tutti i microtask (es. .then()) e poi prende un task dalla task queue e lo esegue.

Callback
funzione che viene passata come argomento a un'altra funzione, e che viene chiamata (invocata) in un secondo momento, spesso dopo che un'operazione è stata completata.

Promises
Una promise è un oggetto che rappresenta il comportamento (o il fallimento) di un'operazione asincrona. In pratica, promette che prima o poi verrà completata con un valore (se ha successo) o con un errore (se fallisce).
Una Promise può trovarsi in 3 stati:
- Pending: La promessa è stata creata ma l'operazione non è ancora completata.
- Fulfilled: L'operazione è andata a buon fine e la promessa ha un valore.
- Rejected: Qualcosa è andato storto e la promessa ha un errore.
Si creano con il costruttore promise = new Promise((resolve, reject) => {}).
Una volta creata, si può agire sul risultato con: .then() se ha avuto successo, .catch() se ha fallito, .finally() per codice che dovrà essere eseguito in ogni caso.

```
function loginUtente(username) {
    fetch(`/api/login?user=${username}`)
        .then((risposta) => {
            if (!risposta.ok) {
                // Lancia manualmente un errore per entrare nel catch
                throw new Error("Errore di rete");
            }
            return risposta.json();
        })
        .then((dati) => {
            console.log('Benvenuto,', dati.nome);
        })
        .catch((errore) => {
            console.error('Errore durante il login:', errore);
        });
}
```

Microtask

La coda dei microtask è un meccanismo interno di JS in cui vengono accodati i compiti asincroni “leggeri”, come i gestori .then() delle Promise, che devono essere eseguiti SUBITO DOPO il codice sincrono attuale ma PRIMA di qualsiasi altra cosa (come timer o eventi).

In parole più semplici: js esegue il codice riga per riga (sincrono), quando incontra una Promise non la esegue immediatamente ma inserisce il codice dentro .then(), .catch(), .finally() in una coda speciale: la Microtask Queue. Non appena il motore JS finisce di eseguire il codice sincrono, esegue le microtask nella coda secondo la logica FIFO (prima vengono eseguite le più vecchie fino alle più recenti) e ciò avviene prima ancora di gestire i timer (setTimeout) e altri macrotask. Ha quindi priorità rispetto alla coda di callback

Fetch e AJAX sono due modi per effettuare richieste HTTP da una pagina web verso un server. Sono usati per ottenere dati, inviarli, aggiornare contenuti senza ricaricare la pagina. In altre parole rende le applicazioni web dinamiche e interattive.
- AJAX (Asyncronous Javascript And XML): rappresenta la tecnica di programmazione che consente a una pagina web di comunicare con un server in modo asincrono senza dover ogni volta ricaricare l’intera pagina. Si basa sull'oggetto XMLHttpRequest per la comunicazione.
- Fetch: è un'API introdotta nei browser per semplificare le chiamate HTTP. È basata sulle promises, è piu leggibile e pulita.

Poiché concatenando Promises, il codice diventa poco leggibile, sono state introdotte due keywork: await e async .
- async trasforma una funzione in una funzione asincrona che restituisce sempre una Promise.
- await rende il codice asincorno e aspetta la risposta finché la promise non è completata. Si usa solo all'interno di funzioni async.
Insieme permettono di scrivere codice asincrono come se fosse sincrono, migliorando chiarezza, leggibilità e manutenzione.

Errori
Per la gestione dell’errore c'è il try and catch.

NodeJS
NodeJS è un ambiente di esecuzione per js che permette di eseguire codice js fuori dal browser (lato server). Per quel che riguarda l’architettura, Node.js è basato su:
- Single-threaded event loop: ossia usa un solo thread principale per gestire tutte le richieste
- Event-driven e asincrono: si basa su eventi e operazioni non bloccanti (event loop=
- Può essere considerato come ambiente runtime per JavaScript, basato sul motore V8 di Google, fornisce un contesto dove si può scrivere codice js su qualsiasi piattaforma dove Node.js può essere installato, l’ambiente ideale dove usare Node.js è il server.
- Internamente usa una libreria libuv, per gestire thread in background (thread pool)

Oltre al thread principale per js, Node.js ha una serie di altri thread in background (detta Thread Pool e gestita da libuv) per operazioni particolarmente pesanti o bloccanti (come accesso ai file, DNS etc…)

Node.js esegue fino a 4 thread separati in parallelo nella thread pool, ma questi non eseguono codice JS direttamente.
Quindi Node.js single-threaded per js, ma può usare thread secondari per operazioni costose.
Node utilizza vari moduli, pezzi riusabili di codice (messi all'interno di require), questi possono essere core, locali o di terze parti.
Le IIFE sono funzioni che si eseguono subito dopo la dichiarazione. In Node venivano usate per isolare codice ed evitare conflitti di variabili.
Express
Express è un framework per NodeJSm progettato per la semplificare la creazione di server web e API.
Permette di definire e gestire in modo facile e flessibile le route, aggiungere middlerware etc... .

Routing (livello applicazioni web, non di rete)
Il routing è il modo in cui un'applicazione risponde alle richieste HTTP su determinati percorsi (URL) con metodi specifici (GET, POST, ecc.). (quindi risponde alle richieste su vari endpoint).
I route parameters (o parametri dinamici) sono segmenti della URL che fungono da segnaposto per valori variabili. Si usano per creare rotte dinamiche in Express.

Un middleware in Express è una funzione che viene eseguita durante il ciclo di gestione di una richiesta HTTP.
Serve per intercettare, modificare o gestire la richiesta o la risposta, prima di arrivare all'handler finale (cioè alla risposta vera e propria).
Un middleware è una funzione che riceve tre argomenti, sempre in questo ordine:
- req == oggetto richiesta (request)
- res == oggetto risposta (response)
- next == callback da chiamare per passare il controllo al middleware successivo.

I middleware sono disposti in cascate cosi quando un Middleware non deve mandare la risposta al cliente, si invoca next() per passare a quelle successive nella catena in modo da continuare ad elaborare la richiesta fino eventualmente a mandare la risposta.

Metodo Express	Tipo di risposta	Descrizione
`res.send()`	testo / oggetto / buffer	Risposta generica, si adatta al tipo
`res.json()`	JSON	Risposta in formato JSON
`res.sendFile()`	file	Invia un file al client
`res.redirect()`	redirect HTTP	Reindirizza verso un altro URL
`res.status()`	imposta codice HTTP	Imposta lo stato della risposta

Per gestire i file statici (immagini, file CSS e file JavaScript) si utilizza la funzione middleware integrata express.static in Express.

REST (REpresentational State Transfer)
È un insieme di linee guida o principi per la realizzazione di una architettura di sistema, non si riferisce ad un sistema concreto, non si tratta di uno standard, È un architettura Client-Server e Stateless e ha la cache. Ha diversi principi tra cui:
- Identificazione delle risorse: ogni risorsa (es. utente, prodotto, ordine) è identificata da un URI univoco. (tipicamente in formato JSON o XML) Una risorsa è qualsiasi entità significativa per il sistema.
- Uso esplicito dei metodi HTTP: le operazioni sulle risorse usano i metodi HTTP standard.
- Risorse autodescrittive: ogni risorsa contiene tutte le informazioni necessarie per essere compresa e gestita dal client.
- Collegamenti tra risorse (Hypermedia): le risorse possono contenere link verso altre risorse correlate, seguendo il principio HATEOAS.
- Comunicazione stateless (senza stato): ogni richiesta client al server deve contenere tutte le informazioni necessarie per essere compresa e gestita, senza mantenere stato tra una richiesta e l'altra. La principale ragione di ciò è la scalabilità (mantenere uno stato consuma tante risorse per il server)

Nelle API REST posso eseguire le operazioni CRUD (create, read, update, delete) per farlo farò una richiesta con rispettivi metodi HTTP (POST, GET, PUT, DELTE) e si ha un unico punto di ingresso API

API (application program interface) Un'API è un'interfaccia che consente a due applicazioni o servizi di comunicare tra loro, seguendo regole ben definite per lo scambio di dati e comandi. Rest non è un'API ma un modo per progettare API secondo i principi visti prima.

CSR e SSR
CSR (Client Side Rendering): usa JavaScript per modificare dinamicamente il DOM nel browser.
– Vantaggi: interfacce interattive (es. social).
– Svantaggi: prestazioni scarse con grandi volumi di dati.
SSR (Server Side Rendering): il server genera l'HTML completo prima di inviarlo.
– Vantaggi: più semplice, ottimo per SEO.
– Usato nei marketplace (Amazon, eBay…).

Templating
tecnica che permette di generare pagine web dinamiche combinando codice HTML con codice JavaScript. Vantaggi:
– Riduzione del codice duplicato: evita la ripetizione di strutture comuni come header, footer e menu di navigazione.
– Facilità di manutenzione: modifiche a componenti comuni si riflettono su tutte le pagine.
– Velocità di sviluppo: consente di concentrarsi sul contenuto specifico di ogni pagina senza riscrivere il layout generale.

EJS è un motore di template per Node.js che permette di inserire codice JavaScript direttamente in HTML.