Appunti Tecnologie Web

Appunti estratti degli argomenti delle domande teoriche più frequenti

1. Le Fondamenta del Web

1.1 I 3 Standard alla Base del Web

Il World Wide Web, come lo conosciamo oggi, si fonda su tre pilastri tecnologici che ne permettono il funzionamento.

- **HTML (HyperText Markup Language):** È il linguaggio di marcatura standard per la creazione di pagine web. Non è un linguaggio di programmazione, ma un linguaggio descrittivo che definisce la **struttura** e il **contenuto** di un documento web attraverso l'uso di "tag".
  - **Contesto Storico:** Nato nel 1991 da un'idea di Tim Berners-Lee al CERN, si è evoluto attraverso varie versioni (HTML 4, XHTML) fino all'attuale standard, HTML5, che ha introdotto enormi miglioramenti.
- **HTTP/S (HyperText Transfer Protocol / Secure):** È il protocollo di comunicazione a livello applicativo che definisce le regole per lo scambio di messaggi tra un client (es. un browser) e un server web.
  - **HTTP:** Opera su un modello richiesta-risposta (request-response). Il client invia una richiesta (es. `GET /pagina.html`) e il server risponde con lo stato della richiesta e, se ha successo, con la risorsa richiesta.
  - **HTTPS:** È la versione sicura di HTTP. Aggiunge un livello di crittografia (TLS - Transport Layer Security, successore di SSL) per proteggere i dati scambiati da intercettazioni e manomissioni. È oggi lo standard de facto per qualsiasi sito web.
- **URI (Uniform Resource Identifier):** È un identificatore univoco per una risorsa sul web. L'URI ha due sottotipi principali:
  - **URL (Uniform Resource Locator):** È il tipo di URI più comune e specifica la "posizione" di una risorsa (es. `https://www.esempio.it/contatti`), includendo il protocollo, l'host e il percorso.
  - **URN (Uniform Resource Name):** Identifica una risorsa tramite un nome univoco, ma non necessariamente la sua posizione (es. `urn:isbn:978-0140449136` per un libro).

1.2 Scambio di Informazioni Client-Server

Il processo di visualizzazione di una pagina web è una danza coordinata tra client e server.

1. **Richiesta del Client:** L'utente digita un URL nel browser (il client). Il browser traduce l'URL in una richiesta HTTP/S.
2. **Risoluzione DNS:** Il browser contatta un server DNS (Domain Name System) per tradurre il nome di dominio (es. `www.esempio.it`) nell'indirizzo IP numerico del server che ospita il sito.
3. **Connessione e Invio Richiesta:** Il browser stabilisce una connessione TCP/IP con il server all'indirizzo IP ottenuto e invia la richiesta HTTP/S. La richiesta contiene:

- **Metodo:** `GET` (per leggere una risorsa), `POST` (per inviare dati), ecc.
- **URI della Risorsa:** `/pagina.html`
- **Header:** Informazioni aggiuntive come il tipo di browser (User-Agent), le lingue accettate (Accept-Language) e i **cookie** precedentemente salvati da quel server.

4. **Elaborazione del Server:** Il web server (es. Apache, Nginx) riceve la richiesta. Può:
   - Reperire un file statico (HTML, CSS, immagine).
   - Passare la richiesta a un'applicazione lato server (es. uno script PHP, Node.js) che genera dinamicamente l'HTML.
5. **Risposta del Server:** Il server invia una risposta HTTP/S al client, che contiene:
   - **Status Code:** `200 OK` (successo), `404 Not Found` (risorsa non trovata), `500 Internal Server Error`, ecc.
   - **Header:** Informazioni sulla risposta, come il tipo di contenuto (Content-Type: `text/html`), e istruzioni per impostare nuovi cookie.
   - **Body (Corpo):** Il contenuto della risorsa richiesta (es. il codice HTML della pagina).
6. **Rendering del Browser:** Il browser riceve la risposta, interpreta (esegue il "parsing") del codice HTML per costruire la struttura della pagina (DOM - Document Object Model), scarica le risorse collegate (CSS, JavaScript, immagini) e infine "disegna" (renderizza) la pagina visibile all'utente.

1.3 Codifica dei Caratteri (Character Encoding)

Per rappresentare il testo in formato digitale, ogni carattere deve essere associato a un codice numerico. La codifica dei caratteri è lo standard che definisce questa associazione.

Codifica	Descrizione	Vantaggi	Svantaggi
ASCII	Utilizza 7 bit (128 caratteri).	Molto compatto, standard storico per l'inglese.	Non può rappresentare caratteri accentati, simboli di valuta o alfabeti non latini.
ISO-8859-1 (Latin-1)	Utilizza 8 bit (256 caratteri).	Estende ASCII per includere i caratteri accentati dell'Europa occidentale.	Non è universale, non può rappresentare caratteri cirillici, greci o asiatici.
UTF-8	Utilizza un numero variabile di byte (da 1 a 4).	**Standard universale del web.** Compatibile all'indietro con ASCII (i primi 128 caratteri usano 1 solo byte). Può rappresentare qualsiasi carattere Unicode.	Leggermente meno efficiente in termini di spazio per lingue non latine (es. asiatiche) rispetto a codifiche specifiche come UTF-16.

Best Practice: Dichiarare sempre la codifica nel documento HTML per evitare che il browser la interpreti in modo errato, causando la visualizzazione di caratteri incomprensibili (effetto "mojibake").

```
<head>
    <meta charset="UTF-8">
</head>
```

Punti Chiave: UTF-8 è la codifica da utilizzare per qualsiasi progetto web moderno. Risolve il problema della compatibilità internazionale del testo.

2. HTML5: Struttura Semantica e Nuove API

2.1 Cosa ha introdotto HTML5

HTML5 non è solo un aggiornamento, ma una profonda revisione dello standard che ha modernizzato il web.

- **Sintassi Flessibile:** Abbandona la rigidità di XHTML, avvicinandosi alla sintassi più permissiva di HTML4. L'obiettivo è la robustezza: i browser sono progettati per interpretare il codice anche se non è perfettamente valido. Il DOCTYPE è stato semplificato drasticamente: `<!DOCTYPE html>`.
- **Elementi Semantici:** Introduce tag che descrivono il *significato* del loro contenuto, non solo la loro presentazione.
- **Supporto Multimediale Nativo:** I tag `<audio>` e `<video>` permettono di integrare contenuti multimediali senza dipendere da plugin esterni come Flash.
- **Grafica e Animazioni:**
  - `<canvas>`: Fornisce un'area di disegno su cui è possibile renderizzare grafica 2D (e 3D con WebGL) tramite API JavaScript.
  - **SVG (Scalable Vector Graphics):** Anche se preesistente, la sua integrazione diretta in HTML5 è stata migliorata.
- **API JavaScript Avanzate:**
  - **Geolocation:** Per ottenere la posizione geografica dell'utente (con il suo consenso).
  - **Web Storage (localStorage e sessionStorage):** Un'alternativa moderna e più capiente ai cookie per salvare dati lato client.
  - **Web Workers:** Per eseguire script in background senza bloccare l'interfaccia utente.
  - **Drag and Drop:** API native per implementare funzionalità di trascinamento.
- **Moduli Migliorati:** Nuovi tipi di input (`date`, `email`, `number`, `range`) che offrono validazione nativa e interfacce utente migliori (specialmente su mobile).

2.2 Elementi Semantici: Il Perché e il Come

Prima di HTML5, le pagine erano spesso costruite con un abuso di `<div>` generici (il cosiddetto "div-itis"). Gli elementi semantici danno un significato strutturale al documento.

- **`<header>`:** Contiene l'intestazione di una pagina o di una sezione (logo, titolo, navigazione principale).
- **`<nav>`:** Raggruppa i link di navigazione principali.
- **`<main>`:** Identifica il contenuto principale e unico della pagina. Dovrebbe essercene solo uno per documento.
- **`<article>`:** Rappresenta un contenuto autonomo e auto-conclusivo (es. un post di un blog, un articolo di giornale, un commento).
- **`<section>`:** Raggruppa contenuti tematicamente correlati. A differenza di `<article>`, non è necessariamente autonomo.
- **`<aside>`:** Contiene informazioni correlate ma tangenziali al contenuto principale (es. una barra laterale con link correlati, biografia dell'autore).
- **`<footer>`:** Contiene il piè di pagina di un documento o di una sezione (informazioni di copyright, contatti, link secondari).

Perché usare la semantica?

1. **Accessibilità (a11y):** Gli screen reader usano questi tag per permettere agli utenti con disabilità visive di navigare la pagina in modo più efficiente (es. "salta alla navigazione", "vai al contenuto principale").
2. **SEO (Search Engine Optimization):** I motori di ricerca comprendono meglio la struttura e la gerarchia dei contenuti, migliorando il posizionamento.
3. **Manutenibilità:** Un codice semanticamente corretto è più facile da leggere, capire e modificare per gli sviluppatori.

Domanda di Approfondimento: Qual è la differenza pratica tra `<section>` e `<article>`?

- Un `<article>` potrebbe vivere da solo in un feed RSS e avere ancora senso. Una `<section>` è una parte della pagina che raggruppa contenuti; se presa da sola, potrebbe non avere un significato completo.

3. CSS: Stile, Layout e Design Adattivo

3.1 La Cascata (Cascade) in CSS

CSS sta per **Cascading Style Sheets**. Il termine "cascading" (a cascata) è fondamentale: descrive l'algoritmo con cui il browser determina quale regola di stile applicare a un elemento quando ci sono più

regole in conflitto.

L'ordine di priorità della cascata è il seguente:

1. **Origine e Importanza (Origin and Importance):**
   - **Transizioni e Animazioni CSS** (massima priorità).
   - **Regole Utente con `!important`:** Stili definiti dall'utente nel proprio browser con `!important`.
   - **Regole Autore con `!important`:** Stili definiti dallo sviluppatore nel CSS del sito con `!important`.
   - **Regole Autore:** Gli stili normali del sito.
   - **Regole Utente:** Stili normali definiti dall'utente.
   - **Stili di Default del Browser (User-Agent):** Lo stile predefinito del browser (es. link blu, titoli in grassetto).
2. **Specificità (Specificity):** A parità di origine, vince la regola con il selettore più specifico. La specificità è calcolata come un valore a 4 cifre (es. `1,0,0,0`):
   - **Stili Inline (es. `<div style="...">`):** 1-0-0-0 (massima specificità).
   - **ID (es. `#mioId`):** 0-1-0-0.
   - **Classi, pseudo-classi e attributi (es. `.miaClasse`, `:hover`, `[type="text"]`):** 0-0-1-0.
   - **Elementi e pseudo-elementi (es. `div`, `::before`):** 0-0-0-1.
3. **Ordine nel Codice (Source Order):** A parità di origine e specificità, vince l'ultima regola dichiarata nel codice. Se si importa un foglio di stile esterno, le regole definite *dopo* l'importazione avranno la precedenza.

3.2 Fogli di Stile: Esterni, Interni e Inline

Tipo	Descrizione	Vantaggi	Svantaggi
Esterno	Gli stili sono in un file `.css` separato, collegato tramite `<link rel="stylesheet" href="style.css">`.	**Best Practice.** Separazione tra contenuto (HTML) e presentazione (CSS). Riutilizzabile su più pagine. Facile da manutenere. Sfrutta la	Richiede una richiesta HTTP aggiuntiva (impatto minimo con HTTP/2).

Tipo	Descrizione	Vantaggi	Svantaggi
		cache del browser.	
Interno	Gli stili sono inseriti nell' `<head>` del documento HTML all'interno di un tag `<style>` .	Utile per stili specifici di una singola pagina o per rapidi test. Non richiede richieste HTTP aggiuntive.	Non riutilizzabile. Mescola struttura e presentazione. Aumenta il peso del file HTML.
Inline	Gli stili sono applicati direttamente a un elemento tramite l'attributo `style="..."` .	Massima specificità, utile per sovrascrivere regole specifiche in modo rapido (es. via JavaScript) o per HTML email.	**Da evitare.** Viola completamente la separazione dei concetti. Difficilissimo da mantenere. Non sfrutta la cache. Aumenta il disordine nel codice HTML.

3.3 Novità di CSS3

CSS3 ha rappresentato una svolta, passando da una specifica monolitica a un approccio **modulare**. Questo significa che diversi aspetti del CSS (colori, selettori, layout) possono evolvere indipendentemente, permettendo un'innovazione più rapida.

Le principali introduzioni includono:

- **Media Queries:** La tecnologia fondamentale per il Responsive Design. Permettono di applicare blocchi di regole CSS solo se determinate condizioni sono verificate (es. la larghezza dello schermo).

```css
/* Stili di base (Mobile First) */
.container { width: 100%; }

/* Stili per schermi più grandi di 768px */
@media (min-width: 768px) {
    .container {
        width: 750px;
        margin: 0 auto;
    }
}
```

- **Selettori Avanzati:** Maggiore potere per selezionare elementi senza aggiungere classi o ID (es. `p:nth-child(2n)` per selezionare tutti i paragrafi pari).
- **Miglioramenti Visuali:**
  - `border-radius` : Per creare angoli arrotondati.
  - `box-shadow` e `text-shadow` : Per aggiungere ombre a box e testo.
  - **Gradienti:** Creazione di sfumature di colore (`linear-gradient` , `radial-gradient`) senza usare immagini.
  - **Sfondi Multipli:** Possibilità di applicare più immagini di sfondo a un singolo elemento.
- **Moduli di Layout Moderni:**
  - **Flexbox (Flexible Box Layout):** Un modello di layout monodimensionale ottimizzato per distribuire lo spazio tra gli elementi in un contenitore, anche quando la loro dimensione è sconosciuta o dinamica. Ha risolto innumerevoli problemi di allineamento verticale e orizzontale.
  - **Grid Layout:** Un potente modello di layout bidimensionale (righe e colonne), che permette di creare griglie complesse e responsive con un controllo senza precedenti.
- **Transizioni e Animazioni:**
  - `transition` : Permette di animare in modo fluido il passaggio di una proprietà CSS da un valore all'altro (es. un cambio di colore al passaggio del mouse).
  - `animation` e `@keyframes` : Per creare animazioni complesse e multi-step.
- **Web Fonts (`@font-face`):** La possibilità di incorporare font personalizzati direttamente nel CSS, liberando i designer dalla dipendenza dai pochi "font sicuri per il web".

3.4 Codifica dei Colori

CSS offre diversi modi per definire i colori, ognuno con i suoi casi d'uso.

Metodo	Esempio	Descrizione
Parole Chiave (Keywords)	`red` , `blue` , `tomato`	Nomi predefiniti per i colori più comuni. Semplici ma limitati.

Metodo	Esempio	Descrizione
Esadecimale (HEX)	`#FF0000` (rosso) `#F00` (shorthand)	Rappresenta i valori di Rosso, Verde e Blu (RGB) in base 16. È il formato più compatto e comune nelle specifiche di design.
RGB / RGBA	`rgb(255, 0, 0)` `rgba(255, 0, 0, 0.5)`	Definisce il colore tramite i suoi componenti di Rosso, Verde e Blu (da 0 a 255). La versione **RGBA** aggiunge un canale **Alpha** (trasparenza) da 0.0 (trasparente) a 1.0 (opaco).
HSL / HSLA	`hsl(0, 100%, 50%)` `hsla(0, 100%, 50%, 0.5)`	Definisce il colore tramite **Tonalità (Hue)**, **Saturazione (Saturation)** e **Luminosità (Lightness)**. È molto più intuitivo per l'occhio umano: per scurire un colore basta diminuire la luminosità, senza dover ricalcolare tre valori RGB. La versione **HSLA** include il canale Alpha.

Punto Chiave: HSL/HSLA è estremamente utile per creare palette di colori dinamiche (es. creare la versione "hover" di un pulsante schiarendolo del 10%) in modo programmatico.

4. Progettazione Centrata sull'Utente (User-Centered Design - UCD)

4.1 Filosofia del Design User-Centered

Il Design User-Centered (UCD) è un approccio progettuale che pone i bisogni, i desideri e i limiti degli utenti finali al centro di ogni fase del processo di design e sviluppo. L'obiettivo non è costringere l'utente ad adattarsi al prodotto, ma progettare il prodotto perché si adatti all'utente.

Il processo UCD è tipicamente **iterativo** e si basa su un ciclo di:

1. **Ricerca e Analisi:** Capire chi sono gli utenti e qual è il loro contesto d'uso.
2. **Progettazione:** Creare soluzioni basate sui dati raccolti (es. wireframe, prototipi).
3. **Test e Valutazione:** Mettere i prototipi nelle mani di utenti reali per raccogliere feedback.
4. **Affinamento:** Modificare il design in base al feedback e ripetere il ciclo.

4.2 UX (User Experience) vs. Usabilità

Questi due termini sono spesso confusi, ma descrivono concetti diversi, sebbene correlati.

- **Usabilità:** È un **attributo qualitativo** dell'interfaccia. Riguarda la facilità con cui un utente può raggiungere un obiettivo specifico. È pragmatica e misurabile. **Domanda chiave: "L'utente riesce a farlo?"**
- **User Experience (UX):** È un concetto **olistico e soggettivo**. Comprende **tutte** le percezioni e le reazioni di un utente che derivano dall'uso (o dall'aspettativa d'uso) di un prodotto o servizio. Include l'usabilità, ma anche l'estetica, le emozioni, le performance, il branding e il piacere d'uso. **Domanda chiave: "Che esperienza ha vissuto l'utente?"**

In sintesi, **l'usabilità è una componente fondamentale di una buona UX, ma non è l'intera UX.** Un sito può essere usabile ma avere una pessima UX (es. è funzionale ma brutto, lento e frustrante).

4.3 Usabilità secondo la Norma ISO 9241

La norma internazionale **ISO 9241-11** fornisce una definizione formale e ampiamente accettata di usabilità:

> "Il grado in cui un prodotto può essere usato da specifici utenti per conseguire specifici obiettivi con **efficacia**, **efficienza** e **soddisfazione** in uno specifico **contesto d'uso**."

Analizziamo i componenti:

- **Efficacia:** La capacità degli utenti di raggiungere i propri obiettivi con accuratezza e completezza. (L'utente ci riesce?)
- **Efficienza:** Le risorse impiegate per raggiungere l'obiettivo (tempo, sforzo cognitivo, numero di click). (Quanto velocemente/facilmente ci riesce?)
- **Soddisfazione:** La percezione soggettiva dell'utente riguardo la piacevolezza e l'accettabilità del sistema. (All'utente è piaciuto usarlo?)
- **Contesto d'Uso:** L'usabilità non è una proprietà assoluta, ma dipende da chi sono gli utenti, quali compiti svolgono e in quale ambiente (fisico e tecnico) operano.

4.4 Strumenti di Progettazione: Personas, Scenarios, Focus Group

Questi sono strumenti pratici utilizzati nel processo UCD per comprendere e rappresentare gli utenti.

- **Personas:**
  - **Cosa sono:** Archetipi di utenti fittizi, ma basati su dati di ricerca reali. Sono descrizioni dettagliate che rappresentano un gruppo di utenti con comportamenti e obiettivi simili. Non sono persone reali, ma rappresentazioni credibili.

    - **Scopo:** Dare un volto umano ai dati demografici. Aiutano il team di sviluppo a empatizzare con gli utenti e a prendere decisioni di design focalizzate sui loro bisogni ("Cosa farebbe 'Marco, il professionista impegnato' in questa situazione?").
- **Scenarios (Scenari):**
    - **Cosa sono:** Brevi storie narrative che descrivono come una *Persona* utilizza il prodotto per raggiungere un obiettivo specifico. Includono il contesto, le motivazioni e le possibili distrazioni.
    - **Scopo:** Contestualizzare l'interazione. Aiutano a definire i flussi utente e i requisiti funzionali, rispondendo alla domanda "Come e perché l'utente userà questa feature?".
- **Focus Group:**
    - **Cosa sono:** Sessioni di discussione moderate con un piccolo gruppo di utenti target (6-9 persone).
    - **Scopo:** Raccogliere opinioni, atteggiamenti e percezioni su un prodotto, un'idea o un concept. Sono ottimi per la fase esplorativa della ricerca.
    - **Limitazione Importante:** I focus group rivelano ciò che le persone **dicono** di fare o pensare, che può essere diverso da ciò che **fanno** realmente. Per valutare il comportamento effettivo, si preferisce il **test di usabilità** individuale.

5. Accessibilità Web (a11y)

5.1 Definizione di Accessibilità

L'accessibilità web (spesso abbreviata come **a11y**, dove "11" rappresenta le lettere tra la "a" e la "y") è la pratica di progettare e sviluppare siti e applicazioni web in modo che possano essere utilizzati da **tutti**, comprese le persone con disabilità.

Questo non riguarda solo la disabilità visiva (cecità, ipovisione), ma un ampio spettro di condizioni:

- **Uditiva:** Sordità o ipoacusia.
- **Motoria:** Difficoltà nell'uso di mouse o tastiera.
- **Cognitiva:** Difficoltà di apprendimento, di concentrazione o di memoria.
- **Situazionale:** Un utente senza disabilità permanenti ma che si trova in una situazione limitante (es. usare lo smartphone sotto il sole, avere un braccio ingessato, non poter attivare l'audio in un luogo pubblico).

Accessibilità vs. Usabilità:

- **Accessibilità:** Si concentra sulla rimozione delle barriere per le persone con disabilità. È un prerequisito.
- **Usabilità:** Si concentra sulla facilità d'uso per tutti. Un sito può essere tecnicamente accessibile ma difficile da usare (quindi con scarsa usabilità). L'obiettivo è renderlo **accessibile E usabile**.

5.2 WCAG 2.0 e Confronto con la Legge Stanca (04/2004)

- **WCAG (Web Content Accessibility Guidelines):**
  - **Cosa sono:** Lo standard internazionale di riferimento per l'accessibilità web, pubblicato dal W3C. La versione 2.0 (e le successive 2.1 e 2.2) è la più rilevante.
  - **Struttura (Principio POUR):** Sono organizzate gerarchicamente su 4 principi fondamentali. Un sito deve essere:
    a. **Percepibile:** Le informazioni e i componenti dell'interfaccia devono essere presentati in modi che gli utenti possano percepire (es. testo alternativo per le immagini, sottotitoli per i video).
    b. **Utilizzabile (Operable):** I componenti dell'interfaccia e la navigazione devono essere utilizzabili (es. tutte le funzionalità disponibili da tastiera, tempo sufficiente per leggere i contenuti).
    c. **Comprensibile (Understandable):** Le informazioni e le operazioni dell'interfaccia devono essere comprensibili (es. linguaggio chiaro, istruzioni semplici, navigazione coerente).
    d. **Robusto (Robust):** Il contenuto deve essere abbastanza robusto da poter essere interpretato in modo affidabile da un'ampia varietà di user agent, comprese le tecnologie assistive.
  - **Livelli di Conformità:** Per ogni linea guida, ci sono dei "criteri di successo" verificabili, classificati in tre livelli di conformità:
    - **A (il più basso):** Requisiti essenziali.
    - **AA (intermedio):** Raccomandato per la maggior parte dei siti. È il livello richiesto dalle normative di molti paesi (inclusa l'UE).
    - **AAA (il più alto):** Requisiti più stringenti, non sempre applicabili a tutti i contenuti.
- **Legge Stanca (Legge 04/2004):**
  - **Cosa è:** La normativa italiana che "favorisce l'accesso dei soggetti disabili agli strumenti informatici". Si applica principalmente alla Pubblica Amministrazione e a enti che ricevono finanziamenti pubblici.
  - **Relazione con WCAG:** Inizialmente, la Legge Stanca aveva dei propri requisiti tecnici. Tuttavia, con gli aggiornamenti successivi (in particolare con le direttive europee), la normativa italiana si è **allineata allo standard WCAG 2.1, livello AA**. Pertanto, oggi, per essere conformi alla Legge Stanca, è necessario essere conformi alle WCAG 2.1 livello AA. Il confronto tra i vecchi requisiti della Legge Stanca e le WCAG è quindi diventato un esercizio storico più che pratico.

6. Responsive Design e Mobile First

6.1 Graceful Degradation vs. Progressive Enhancement

Queste sono due filosofie di progettazione che affrontano il problema della diversità di browser e dispositivi.

- **Graceful Degradation (Degrado Grazioso):**
  - **Punto di Partenza:** Il sito viene progettato e sviluppato per i browser più moderni e potenti (desktop), con tutte le funzionalità e gli effetti grafici.
  - **Processo:** Successivamente, si lavora per "degradare" l'esperienza in modo che il sito rimanga funzionale, anche se meno ricco, sui browser più vecchi o sui dispositivi meno capaci. Si rimuovono o si sostituiscono le funzionalità non supportate.
  - **Focus:** Garantire che "funzioni ancora" su dispositivi inferiori.
- **Progressive Enhancement (Miglioramento Progressivo):**
  - **Punto di Partenza:** Si progetta e sviluppa un'esperienza di base solida, che funzioni su **qualsiasi** browser e dispositivo, concentrandosi solo sul contenuto e sulle funzionalità essenziali.
  - **Processo:** Si aggiungono progressivamente strati di complessità e arricchimento (layout più complessi, animazioni, funzionalità avanzate tramite JavaScript) solo per i browser che li supportano.
  - **Focus:** Garantire un'esperienza di base solida per tutti e un'esperienza migliore per chi ha dispositivi più moderni. **Questo è il principio alla base del Mobile First.**

6.2 Dal Responsive Design al Mobile First

- **Responsive Web Design (RWD):** È una tecnica di sviluppo che utilizza layout flessibili (con percentuali), immagini flessibili e Media Queries per creare un sito che si adatta a diverse dimensioni dello schermo. Nelle sue prime incarnazioni, il RWD seguiva spesso un approccio di *Graceful Degradation*: si progettava la versione desktop e poi si "restringeva" per gli schermi più piccoli.
- **Mobile First:** È l'applicazione pratica del principio del *Progressive Enhancement* al RWD.
  - **Processo:** Si inizia a progettare e sviluppare per il dispositivo più piccolo e con più vincoli: lo smartphone. Si definisce l'HTML e il CSS di base per garantire che il contenuto sia leggibile e le funzionalità principali siano accessibili. Poi, usando le Media Queries con `min-width`, si aggiungono layout più complessi e funzionalità avanzate man mano che lo spazio sullo schermo aumenta.
  - **Vantaggi:**

a. **Focus sull'Essenziale:** Costringe a dare priorità al contenuto e alle funzionalità più importanti.
b. **Performance Migliori:** I dispositivi mobili scaricano solo il CSS di base, che è più leggero, invece di scaricare tutto il CSS desktop per poi sovrascriverlo.
c. **Codice più Pulito:** È più facile aggiungere regole che toglierle o sovrascriverle, portando a un CSS più snello e manutenibile.

Riassunto Concettuale: Il Responsive Design è *l'obiettivo* (un sito che funziona ovunque). Mobile First è la *strategia migliore* per raggiungerlo.

7. Sviluppo Web: Architettura e Sicurezza

7.1 Solution Stack: LAMP vs. MEAN

Un "solution stack" è una combinazione di tecnologie software usate per creare e far funzionare un'applicazione web completa. LAMP e MEAN sono due degli stack più noti, ma rappresentano filosofie molto diverse.

Caratteristica	LAMP Stack	MEAN Stack
Componenti	**L**inux (OS), **A**pache (Web Server), **M**ySQL (Database), **P**HP (Linguaggio Server)	**M**ongoDB (Database), **E**xpress.js (Backend Framework), **A**ngular (Frontend Framework), **N**ode.js (Runtime Server)
Linguaggio	Principalmente PHP lato server e JavaScript lato client.	**JavaScript ovunque** (frontend, backend, e anche nel database con query JSON-like).
Modello Dati	**SQL (Relazionale):** Dati strutturati in tabelle con schemi rigidi. Ottimo per coerenza e integrità dei dati (transazioni ACID).	**NoSQL (Document-Oriented):** Dati archiviati in documenti JSON-like (chiamati BSON). Schema flessibile, ottimo per dati non strutturati e scalabilità orizzontale.
Architettura	Tradizionalmente usata per siti web "monolitici", dove il server genera e invia pagine HTML complete.	Ottimizzata per **Single Page Applications (SPA)**, dove il frontend (Angular) gestisce gran parte della logica e scambia dati con il backend tramite API RESTful.

Caratteristica	LAMP Stack	MEAN Stack
Vantaggi	Tecnologia matura, stabile, vastissima community e documentazione. Hosting economico e diffuso.	Isomorfismo del linguaggio (un solo linguaggio da imparare). Sviluppo rapido di prototipi. Performance elevate per applicazioni real-time grazie a Node.js.
Svantaggi	Modello di sviluppo più "classico". Meno adatto a SPA e applicazioni real-time complesse.	Meno maturo di LAMP. La mancanza di schemi rigidi in MongoDB richiede più disciplina a livello applicativo per mantenere la coerenza dei dati.

Varianti: Esistono molte varianti, come **MERN** (con React al posto di Angular) o **MEVN** (con Vue.js). La controparte multipiattaforma di LAMP è **XAMPP** (Cross-platform, Apache, MariaDB/MySQL, PHP, Perl).

7.2 Scambio Dati Asincrono tra JavaScript e PHP (AJAX)

La richiesta di "scambiare valori tra JavaScript e PHP" si traduce, in un contesto moderno, nell'implementazione di chiamate **AJAX (Asynchronous JavaScript and XML)**. Questo approccio permette alla pagina web di comunicare con il server in background, senza ricaricare la pagina.

Flusso di Funzionamento Moderno (con `fetch` e JSON):

1. **Evento nel Browser (Client):** Un'azione dell'utente (es. click su un pulsante, inserimento di testo in un campo) scatena un evento JavaScript.
2. **Chiamata `fetch` (JavaScript):** JavaScript usa l'API `fetch` per inviare una richiesta HTTP asincrona a uno script sul server (es. `api.php`). I dati possono essere passati nell'URL (metodo `GET`) o nel corpo della richiesta (metodo `POST`).
3. **Elaborazione (PHP - Server):** Lo script `api.php` riceve la richiesta. Legge i parametri (`$_GET` o `$_POST`), esegue la logica necessaria (es. interroga il database, fa dei calcoli) e prepara i dati per la risposta.
4. **Formattazione della Risposta (PHP):** Invece di generare HTML, lo script PHP formatta i dati di risposta in formato **JSON (JavaScript Object Notation)**, uno standard leggero e facilmente interpretabile da JavaScript.
5. **Invio Risposta (PHP):** Lo script imposta l'header `Content-Type: application/json` e invia la stringa JSON come corpo della risposta.
6. **Gestione della Risposta (JavaScript):** Il codice JavaScript, tramite le *Promises* della `fetch`, riceve la risposta. La converte da stringa JSON a un oggetto JavaScript e la utilizza per aggiornare dinamicamente il DOM (Document Object Model), mostrando i nuovi dati all'utente senza alcun refresh della pagina.

Esempio Pratico:

File `index.html` (con JavaScript):

```
<button id="loadUserBtn">Carica Utente</button>
<p>Nome utente: <span id="userName">...</span></p>

<script>
document.getElementById('loadUserBtn').addEventListener('click', () => {
  fetch('api.php?id=1') // 1. Chiamata GET a api.php
    .then(response => {
      if (!response.ok) { throw new Error('Errore di rete'); }
      return response.json(); // 6. Converte la risposta JSON
    })
    .then(data => {
      // 6. Aggiorna il DOM con i dati ricevuti
      document.getElementById('userName').textContent = data.name;
    })
    .catch(error => console.error('Errore:', error));
});
</script>
```

File `api.php` (sul server):

```
<?php
// 3. Elaborazione della richiesta
$userId = isset($_GET['id']) ? (int)$_GET['id'] : 0;
// In un caso reale, qui ci sarebbe una query al database
$users = [ 1 => ['name' => 'Alice'], 2 => ['name' => 'Bob'] ];
$user = $users[$userId] ?? ['name' => 'Sconosciuto'];

// 4. Impostazione dell'header per la risposta JSON
header('Content-Type: application/json');

// 5. Invio della risposta codificata in JSON
echo json_encode($user);
?>
```

7.3 Principi di Sicurezza per la Trasmissione Dati

Garantire che la trasmissione di dati sensibili sia "sicura" richiede un approccio a più livelli, che va oltre il semplice uso di HTTPS.

1. **Crittografia del Canale (Protezione in Transito):**
    - **HTTPS (HTTP su TLS):** È il fondamento. Cripta i dati tra client e server, proteggendoli da intercettazioni (*Man-in-the-Middle*). Deve essere abilitato su tutto il sito, non solo sulle pagine di login.
    - **HSTS (HTTP Strict Transport Security):** Un header inviato dal server che obbliga il browser a comunicare solo via HTTPS per un dato periodo, prevenendo attacchi che tentano di forzare un downgrade a HTTP.
2. **Protezione dei Dati a Riposo (Protezione sul Server):**
    - **Hashing delle Password:** Le password non devono **mai** essere salvate in chiaro o in modo reversibile. Utilizzare algoritmi di hashing moderni e lenti come **Bcrypt** o **Argon2** (disponibili in PHP tramite `password_hash()` e `password_verify()`). Questi algoritmi includono automaticamente il **salting** per prevenire attacchi basati su *rainbowtable*.
3. **Validazione e Sanificazione dell'Input (Non fidarsi del Client):**
    - **Validazione Lato Server:** La validazione JavaScript sul client è utile per la UX, ma è trivialmente aggirabile. **Ogni dato ricevuto dal client deve essere validato e sanificato sul server.**
    - **Protezione da SQL Injection:** Utilizzare **Prepared Statements** (con PDO o MySQLi in PHP). Questo approccio separa il comando SQL dai dati, rendendo impossibile per un utente malintenzionato alterare la logica della query.
    - **Protezione da Cross-Site Scripting (XSS):** Quando si visualizzano dati forniti dall'utente, effettuarne sempre l'escape (es. con `htmlspecialchars()` in PHP). Questo converte caratteri speciali come `<` e `>` nelle loro entità HTML (`<`, `>`), impedendo l'iniezione di script malevoli nella pagina.
4. **Prevenzione di Attacchi alle Funzionalità:**
    - **Cross-Site Request Forgery (CSRF):** Un attacco che inganna un utente autenticato per fargli compiere un'azione a sua insaputa. Si previene usando **token anti-CSRF**: per ogni azione che modifica dati (POST, DELETE), il server genera un token unico e lo inserisce nel form. Al momento della sottomissione, il server verifica che il token ricevuto sia valido.
    - **Logica Applicativa sul Server:** Non eseguire mai calcoli o logiche sensibili (es. calcolo del prezzo di un carrello, verifica dei permessi) lato client con JavaScript. Questa logica deve risiedere **esclusivamente sul server**, dove non può essere manipolata.

Punti Chiave e Domande di Approfondimento

Consulta anche le domande e risposte di approfondimento.