Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Applicazione delle metodologie di Responsive Web Design e Progressive Enhancement al sito del Museo Virtuale di Informatica

Relatore:
Chiar.mo
Prof. Giorgio Casadei

Presentata da:
Andrea De Carolis

II Sessione Anno Accademico 2015/2016

Indice

Introduzione

Da sempre istituzioni museali, biblioteche e archivi storici costituiscono dei bacini di informazioni che soddisfano necessità sia didattiche, sia di ricerca, sia di semplici curiosità culturali.

Il particolare, un museo virtuale, essendo per sua natura virtuale e quindi accessibile e consultabile a prescindere da una sua dimensione meramente fisica, può soddisfare necessità legate a diversi livelli di approfondimento e chiavi di lettura, a seconda del fruitore, delle esigenze e diversamente da quanto accade per i "centri culturali" propriamente detti. Ad esempio potrebbe risultare utile per i livelli di approfondimento richiesti da classi di scuola primaria, licei, ricercatori e, genericamente, potrebbe rivelarsi un utile strumento per soddisfare la curiosità delle persone colte o anche, semplicemente, appassionate della materia.

Per fare in modo che l'accesso a questi bacini di informazioni sia espresso nella maniera migliore, è opportuno che la fruizione dei contenuti, insieme alla gestione degli stessi, sia efficiente e quanto più versatile e adattabile alle flessibilità di internet. Con questo intendiamo dire che da un lato i contenuti devono essere visualizzabili in modo ottimale su quanti più dispositivi possibili (smartphone, tablet, computer desktop), dall'altro che sia possibile una gestione (aggiornamento, estensione e riorganizzazione) efficiente dei contenuti.

L'obiettivo di questa tesi è di implementare una versione aggiornata del Museo Virtuale di Informatica, utilizzando strumenti aggiornati per garantire le prestazioni flessibili sopra elencate.

Posizione del problema

L'esplosione della navigazione tramite dispositivi mobili come tablet e soprattutto smartphone ha cambiato radicalmente il modo di fruire e di conseguenza realizzare siti web, indipendentemente dalla complessità e dal livello di interazione con l'utente che essi prevedono.

L'idea di rendere facilmente accessibile e consultabile il materiale che si trova all'interno del Museo Virtuale dell'Informazione rappresenta un contributo piccolo ma tangibile per fornire il Museo di una struttura moderna, sicura, e accessibile a quante più persone.

L'evoluzione di dispositivi e di tecnologie ha trovato un punto fermo negli standard W3C riguardo HTML, CSS e JavaScript e allo stesso tempo ha decretato il declino di tecnologie proprietarie come Adobe Flash.

Forti dell'esperienza riguardo questa evoluzione, ri-progettando oggi il sito del MVdI si presenta l'opportunità di aggiornare la base tecnologica dello stesso, permettendo così una facile consultazione da qualsiasi dispositivo (mobile o fisso) e consentendo di allargare il bacino d'utenza.

Il *Responsive Web Design* (d'ora in avanti RWD) è un approccio figlio di questa realtà: utilizzando una serie di pratiche relative al CSS e al HTML, consiste nel fare in modo che il layout del sito si adatti al dispositivo che lo naviga, sia esso piccolo come un telefono o largo come uno schermo desktop.

Questa miglioria però da sola non basta: mentre da un lato HTML e CSS ci permettono grandi passi avanti dal punto di vista grafico e da un punto di vista d'interazione (tutto supportato nativamente dai browser moderni), la grande varietà di dispositivi in circolazione con le più differenti potenzialità ci richiede un ulteriore sforzo perché che questi contenuti siano accessibili universalmente. Il *progressive enhancement* è un approccio alla progettazione del front end che va proprio in questa direzione: ci permette di garantire l'accesso alle informazioni al numero massimo di utenti mentre, al contempo, assicura un'esperienza più ricca (*enhanced*, per l'appunto) ai browser più moderni.

A corollario di questi due approcci ci sono una serie di tecniche e pratiche per progettare i contenuti in modo da ridurre lo sforzo cognitivo per accedervi (*architettura delle informazioni*), per progettare l'interfaccia in modo uniforme tra tutte le larghezze

dei viewport (*mobile first*), per rendere i contenuti e le risorse veloci da scaricare e da visualizzare sul dispositivo (*web performance*).

Infine, ma non per questo meno importante, la gestione dei contenuti richiede uno strumento potente e flessibile a tal punto da poter gestire la complessità intrinseca di tante informazioni organizzate. A questo fine installeremo l'ultima versione di *WordPress* con un tema sviluppato sulla base di quanto appena descritto.

Obiettivo della tesi

Il contributo che, insieme al prof Casadei, è necessario dare al Museo Virtuale di Informatica consiste nell'aggiornare sia la parte di front end, per rendere i contenuti fruibili nel migliore dei modi dalla grande vastità di dispositivi, sia la parte di back end del sito, in modo che quegli stessi contenuti siano gestibili nel tempo e nelle modalità.

Possiamo quindi sintetizzare l'obiettivo di questa tesi in tre punti:

1. **Aggiornare la struttura del sito del Museo Virtuale di Informatica**. Sia dal un punto di vista dell'organizzazione dei contenuti sia dal punto di vista della presentazione (e quindi della leggibilità) degli stessi.

2. **Permetterne la visualizzazione e la consultazione dai dispositivi mobili**. Obiettivo che, come vedremo, influenzerà diversi aspetti della fase di progettazione e della fase di implementazione.

3. Permettere **l'aggiunta, l'aggiornamento e la modifica dei contenuti** del MVdI in modo autonomo tramite un CMS (*content management system*).

Struttura della tesi

La tesi è strutturata in capitoli: il primo capitolo contiene le **metodologie e le scelte progettuali** che guideranno tutte le fasi di sviluppo (capitolo 1). Successivamente andremo ad elencare alcuni **strumenti** che utilizzeremo per mettere in pratica le metodologie appena descritte (capitolo 2). Infine andremo ad **applicare le metodologie**, tramite gli strumenti al Museo Virtuale di Informatica. Questo accadrà secondo due

fasi: la prima relativa **alla parte di front end** (visualizzazione e fruizione dei contenuti, capitolo 3.3), e poi alla parte di **back end** (aggiornamento e gestione dei contenuti, capitolo 3.4).

Capitolo 1

Metodologie e scelte progettuali

In questo capitolo sono elencati i criteri di sviluppo e le modalità di progettazione che utilizzeremo durante tutto il progetto. La scelta di un criterio rispetto ad un altro è spinta da motivazioni di carattere tecnico, motivazioni relative all'accessibilità, motivazioni relative all'organizzazione dietro il Museo Virtuale di Informatica.

1.1 Responsive web design

1.1.1 Origini del termine

Nel mondo dell'architettura da dieci anni a questa parte si è sviluppata una disciplina di progettazione chiamata "*responsive architecture*". Pur essendo un approccio relativamente giovane, ha destato molto interesse per la sua natura più interattiva rispetto alle discipline più classiche dell'architettura.

Un'architettura responsive è un'architettura che risponde, tramite sensori, a degli stimoli provenienti dall'esterno.

Le applicazioni sono le più varie: dall'artista che sperimenta superfici che reagiscono al suono della voce [4] o spazi che si adattano agli occupanti [5], al vetro che diventa opaco una volta che si raggiunge una certa densità di persone all'interno di una stan-

za [6], fino ad arrivare ad un muro flessibile che si piega man mano che le persone si avvicinano creando più o meno spazio a seconda delle necessità [7].

Figura 1.1: Attraverso l'uso di inchiostri termosensibili e fotosensibili, il colore degli interni della sede "Office for Robotic Architectural Media & The Bureau for Responsive Architecture" a Chicago, diventa più chiaro nei giorni caldi e più scuro nei giorni freddi.

Nello stessa maniera dell'architettura, il Responsive Web Design è un approccio che fa adattare il layout di una pagina web sulla base del contesto dove essa viene visualizzata.

1.1.2 Responsive web design

Il termine *Responsive Web Design* (d'ora in poi RWD) è stato coniato da Ethan Marcotte in un articolo pubblicato sul sito *A List Apart* del 2010 [1] e ripreso successivamente più nel dettaglio in suo breve saggio del 2011 [2].

Tecnicamente parlando, un sito sviluppato secondi il RWD presenta tre aspetti del CSS:

1. utilizza un layout con griglie fluide,
2. le immagini sono flessibili (quindi con la larghezza definita in percentuale),
3. utilizza le media query per rilevare la larghezza del browser.

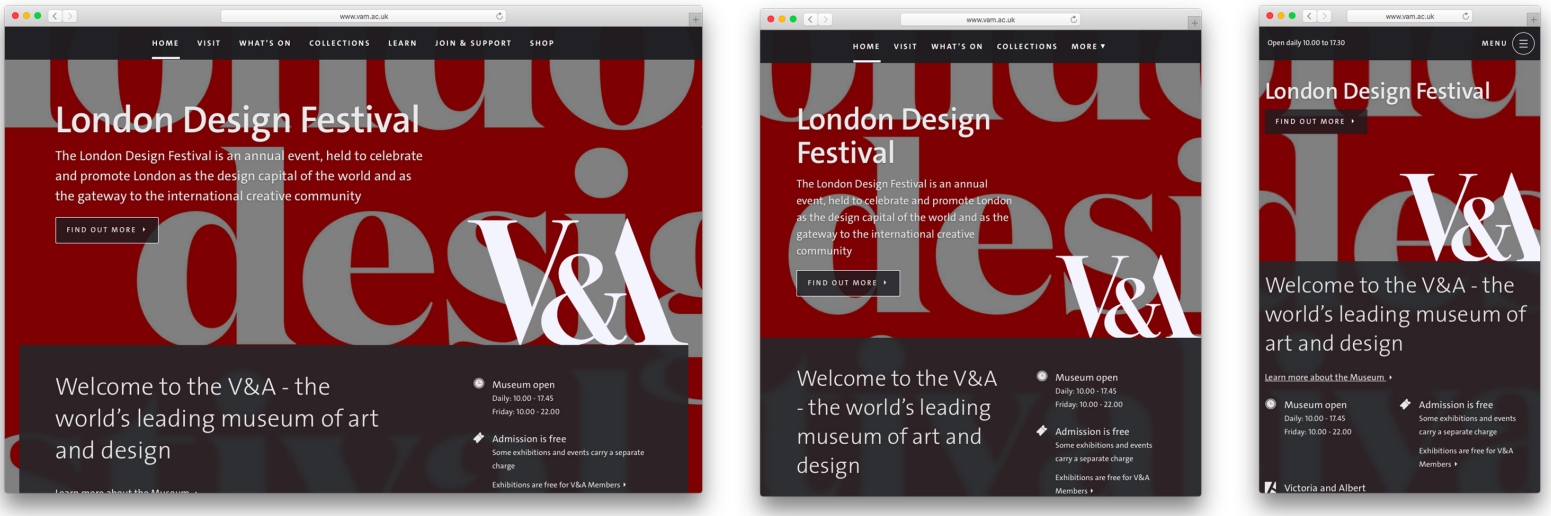


Figura 1.2: Il sito del Victoria & Albert Museum di Londra realizzato con il RWD. La larghezza del contenuto si adatta alla larghezza del contenitore.

Applicando queste tre caratteristiche si otterrà un sito web ottimizzato per il display che lo visualizza, che in particolare avrà:

- il layout della pagina che si adatterà di volta in volta in modo da garantire una leggibilità e una comprensione della stessa (ad esempio tenendo conto della lunghezza delle righe di testo [24]);
- la grandezza dei testi (font-size) sarà grande a sufficienza per garantire un buon livello di leggibilità;

- le aree sensibili (link, bottoni, elementi di input) saranno grandi abbastanza da poter essere utilizzate non esclusivamente con un puntatore (mouse) ma anche da puntatori meno precisi (es. dita);
- il sito sarà consultabile comodamente da dispositivi mobili, senza richiedere azioni di zoom, *pinch* o simili.

Cambio di paradigma

Il RWD ha messo in luce un modo di fare web design che permette di creare pagine che funzionano in modo ottimale a prescindere dal dispositivo che le visualizza, spostando l'attenzione dalla soluzione grafica, o meglio *hack* grafico, all'aspetto centrale del web: i contenuti.

Togliendo infatti i vincoli delle larghezze fisse, la progettazione deve tener conto di più layout da presentare e, rispetto al passato, può far affidamento solo **su alcune delle caratteristiche tecniche** dei dispositivi (ad esempio i puntatori del mouse).

Questo cambio di paradigma nella progettazione ha portato alla crescita e diffusione di alcune discipline come l'**architettura delle informazioni**, il **progressive enhancement**, riportando inoltre attenzione su temi universali come l'**accessibilità** e le **performance**.

1.1.3 Accessibilità

La soluzione ultima per rendere una pagina accessibile è sottoporla ad una (o più) sessioni di accessibilità. Tematica che non verrà approfonditamente trattata in questa tesi ma che terremo in considerazione per l'oggetto del nostro studio.

Faremo dunque quanto più possibile per garantire che i contenuti del MVdI siano accessibili dal più vasto pubblico possibile, a prescindere dal dispositivo che utilizzano, dal browser che utilizza e dal fatto che gli utenti del sito utilizzino tecnologie assistive o meno.

Markup

Per raggiungere il nostro scopo di accessibilità, l'aspetto sul quale possiamo fare più affidamento e concentrare maggiormente le nostre risorse è il markup HTML.

Il *World Wide Web Consortium* (W3C) è l'ente che definisce le specifiche per l'accessibilità sul web attraverso il suo *Web Accessibility Initiative* (WAI). Inoltre le linee guida per l'accessibilità (*Web Content Accessibility Guidelines* - WCAG 2.0) forniscono sia principi di alto livello sia una checklist di specifiche tecniche per assicurarsi che un sito sia effettivamente accessibile. Seguendo le direttive espresse da questi enti è possibile scrivere del markup HTML semantico.

Il markup scritto in modo semantico, con al suo interno informazioni aggiuntive ed eventuali feedback che contribuiscano all'usabilità generale del sito, sarà usabile dagli **screen reader** e dalle tecnologie assistive, sarà **indicizzabile** dai motori di ricerca e ovviamente navigabile dagli utenti con i dispositivi e browser più diffusi. Infine, rispettando queste caratteristiche, possiamo essere ragionevolmente sicuri che il codice rispetterà a sua volta le specifiche di **legge in vigore** nel nostro paese [21].

Regole WAI-ARIA

WAI-ARIA (*Accessible Rich Internet Applications*) è una specifica W3C che include raccomandazioni per applicare significato al markup utilizzato nelle interfacce a prescindere dal fatto che siano statiche o dinamiche.

Esse prevedono un set di attributi, chiamati *landmark roles*, che possono essere assegnati agli elementi base di HTML.

Queste regole permettono di identificare le maggiori sezioni della pagina quali *header*, *footer*, contenuto principale (*main content*) e altre ancora agli *screen reader* (ad esempio in JAWS) e permettono agli utenti di navigare tra queste sezioni utilizzando la tastiera.

Le regole che identificano le sezioni statiche del contenuti includono:

- **banner** — per l'*header* generale che appare in tutte le pagine del sito;
- **complementary** — per contenuti di supporto, come le barre laterali (*sidebar*);
- **contentinfo** — informazioni riguardo il contenuto della pagina, come il *footer* o i *copyright*;
- **main** — per il contenuto primario della pagina (ad esempio il corpo di una notizia);
- **navigation** — per la lista dei link di navigazione;
- **search** — per il form di ricerca.

Si può assegnare una regola WAI-ARIA ad un segmento della pagina utilizzando gli attributi di HTML. Ad esempio:

```
<ul role="navigation">
       <li><a href="home.html">Home</a></li>
       <li><a href="about.html">About the Company</a></li>
       <li><a href="contact.html">Contact Us</a></li>
</ul>
```

Altre regole possono essere aggiunte ai contenuti generati dinamicamente ma per lo scopo di questo progetto basteranno quelle appena elencate.

Web Content Accessibility Guidelines (WCAG)

Attualmente alla versione 2.0, le WCAG sono state create dal WCAG working group e riviste e approvate dal World Wide Web Consortium (W3C), l'organismo standard che produce e mantiene le specifiche di markup e CSS.

Così come sintetizzato da Scott Jehl [23], in modo volutamente generico le WCAG promuovono la creazione di codice che rispetti quattro concetti, ovvero che sia:

- **percepibile**,
- **usabile**,
- **comprensibile**,
- **robusto**.

Nella pratica, questi concetti sono ottenibili seguendo le regole che il WCAG ha redatto:

- Fornire **testo alternativo** per ogni contenuto non testuale. In questo modo il contenuto può cambiare forma per le persone che potrebbero averne bisogno. Esempi di forme alternative possono essere Braille, vocali, simboli, linguaggio più semplice o semplicemente più grande.
- Creare contenuti che possono essere **presentati in modi differenti** (ad esempio con un layout più semplice) senza per questo perdere informazioni o la struttura dei documenti.
- Facilitare nella **visualizzazione** dei contenuti (ad esempio separare visivamente i contenuti in primo piano dallo sfondo).
- Prevedere tutte le funzionalità, prime tra tutte la **navigazione**, disponibili anche **da tastiera**.
- Fornire un modo per aiutare gli utenti a navigare, a trovare i contenuti e a capire dove si trovano all'interno del sito.
- Rendere il codice **leggibile e comprensibile**.
- Fare in modo che le pagine appaiano e si comportino nel modo previsto.
- Massimizza la **compatibilità** con gli attuali e i futuri user agents, compresi quelli di **tecnologie assistive**.

La lista intera delle regole con le spiegazioni dettagliate è presente nel sito del WCAG [20].

1.1.4 Mobile first

Il termine *Mobile first* è stato coniato da Luke Wroblewski nel suo blog nel 2009 [8] e poi approfondito in un saggio del 2011 [9]. È un approccio alla progettazione che pone l'attenzione in ordine prima sui dispositivi mobili e successivamente anche su quelli desktop.

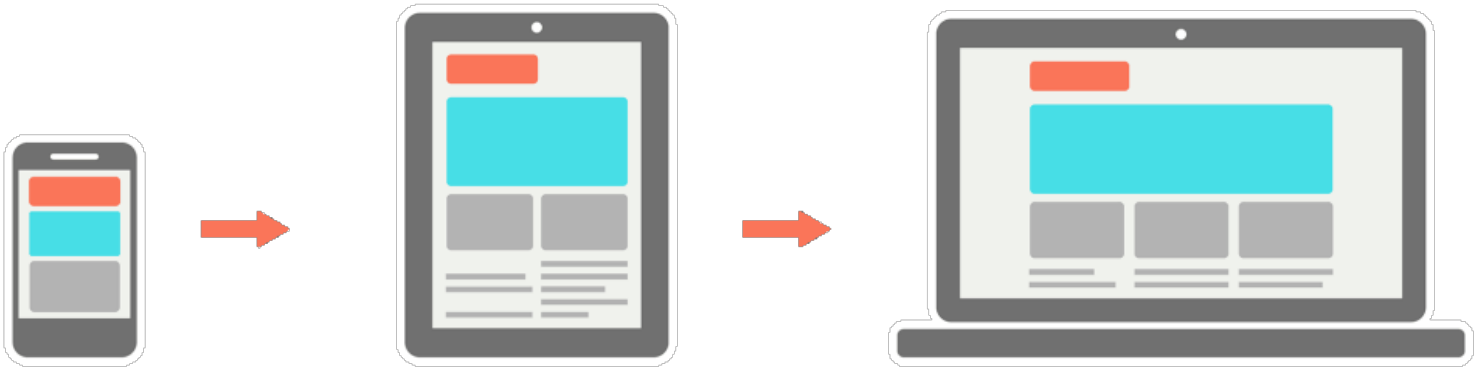


Figura 1.3: Un mockup di esempio che raffigura l'organizzazione del contenuto prima per i device più piccoli (mobile) e poi via via per quelli più grandi (computer desktop)

L'idea alla base è che occorre gestire per prima cosa quelli che sono i contenuti e le parti interattive (prima fra tutti la navigazione) in modo efficiente e considerando quello che potrebbe essere lo scenario di fruizione peggiore. In secondo luogo si è obbligati a dare **il giusto spazio ai contenuti**, eventualmente limitando o eliminando i contenuti secondari che sono, per tanto, non fondamentali. In questo modo ne giovano anche **le performance** e lo sforzo cognitivo nella comprensione del contenuto della pagina.

Open device lab

Uno dei consigli di Luke Wroblewski riguardo il testing dei design, è quello di effettuare i test, per quanto possibile, sui dispositivi reali piuttosto che su un emulatore. Questo perché l'esperienza utente, per quanto simulata, vede per sua natura dei limiti imposti dalla fisicità dello schermo, della risoluzione dello stesso, dalla gestualità delle dita sullo schermo e altro ancora.

Gli *open device lab* (`https://opendevicelab.com/`) sono dei laboratori con raccolte di telefoni che la comunità di sviluppatori web mette a disposizione gratuitamente agli altri sviluppatori. L'idea alla base di questo movimento è che, dato l'elevato tasso di riciclo e aggiornamento dei dispositivi, tanto vale "prestare" alla comunità i vecchi dispositivi in modo che possano aiutare nello sviluppo di applicazioni e siti web.

Nel nostro caso ci avvarremo del device lab di Bologna per effettuare i test dei vari design.

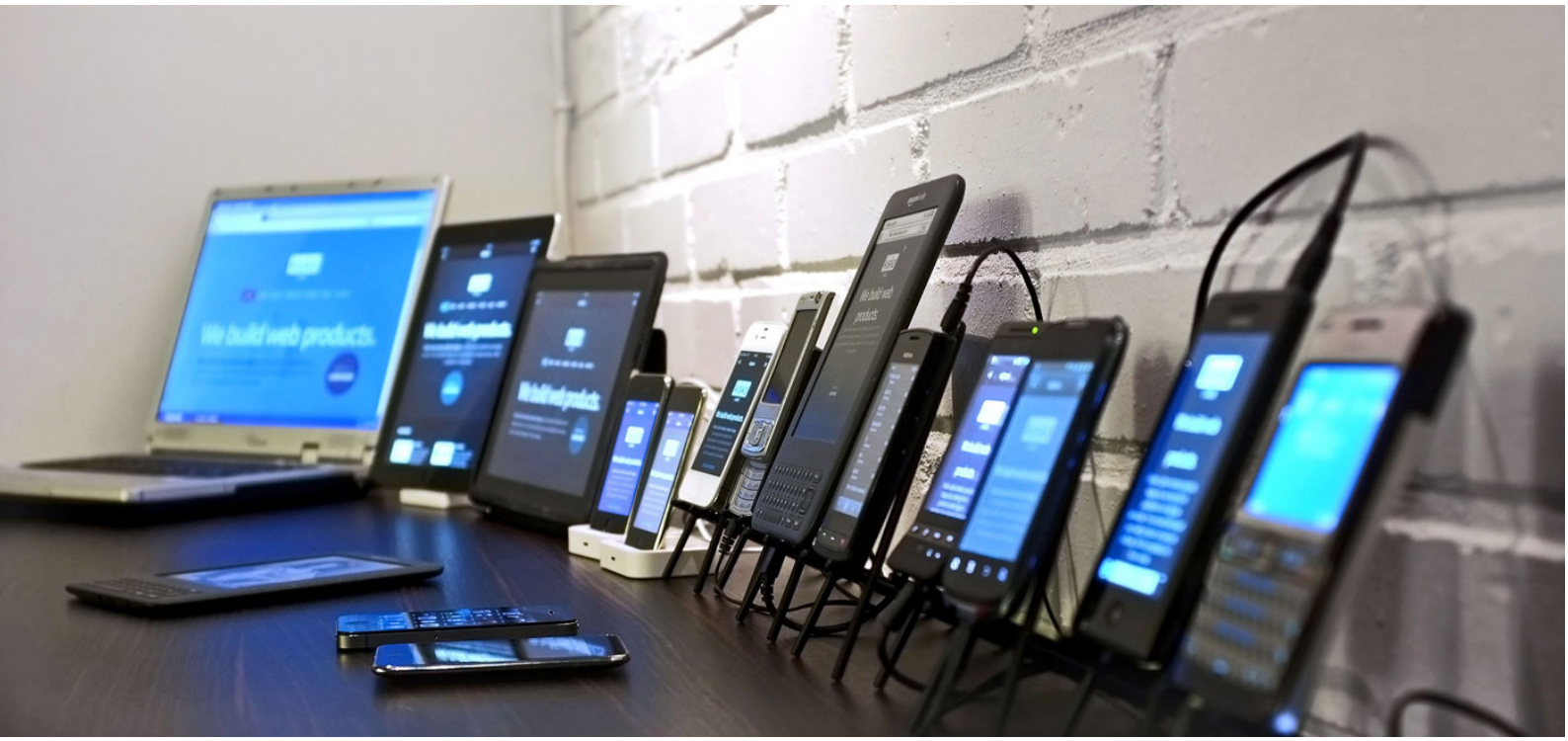


Figura 1.4: Il device lab di Bologna ad una conferenza di Front End [33]

1.1.5 Web performance

Nonostante il tema delle web performance esuli dallo scopo di questa tesi, il suo impatto sull'esperienza utente è talmente elevato che non può non essere citato come uno degli elementi imprescindibili del discorso.

Come nota Steve Souders nel suo sito ([10]), in media più dell'80% della responsabilità legata alla lentezza di un sito dipende da come è organizzato il front end di quello stesso sito. Per questo motivo andremo a stilare una serie di *best practice* al fine di ottimizzare la pagina da questo punto di vista.

Ridurre al minimo le risorse esterne

Pur mantenendo la netta divisione tra contenuto (HTML), presentazione (CSS) e comportamento (JavaScript), è importante ridurre al minimo le chiamate a risorse esterne. Questo data la **natura del protocollo HTTP**. Nella versione largamente utilizzata oggi (1.1), le risorse possono essere scaricate in parallelo in un limitato numero (la specifica del protocollo indica due come numero massimo di connessioni parallele [11] anche se i browser più diffusi non la rispettano più [12]).

Caricare i file CSS all'interno del tag *<head>*

Mettendo i CSS all'interno del tag *<head >*, il browser riesce a **renderizzare la pagina progressivamente**. Il tempo totale di caricamento non è detto che verrà migliorato, ma di sicuro ci sarà una percezione di velocità maggiore (come suggerito dalle linee guida di Yahoo [13]).

Caricare i file JavaScript alla fine della pagina

A differenza dei CSS che permettono alla pagina di caricarsi progressivamente, mentre il browser carica il codice JS blocca la renderizzazione e allo stesso tempo non scarica altre risorse. Per questo motivo i file JavaScript vanno caricati appena sopra la chiusura del tag *</body>* [14]. Inoltre, quando possibile, è consigliabile caricare il javascript in modo asincrono, in modo da bloccare il DOM il minor tempo possibile [19].

Utilizzare i giusto formati di immagini

In media circa il 64% del peso di una pagina web dipende dalle immagini [15]. Le modalità per ottimizzare questo aspetto sono molteplici e variano in base alle necessità: utilizzare **immagini vettoriali** (come il formato SVG) quando possibile, usare i **nuovi**

formati creati appositamente per il web quando supportati (come APNG e WEBP) [16]. e in generale utilizzare le *responsive images*.

Abilitare le compressioni gzip

Per tutte le risorse testuali (HTML, CSS, JavaScript) è auspicabile l'utilizzo della compressione gzip. Questa compressione una volta abilitata permette un **risparmio di byte trasmessi** tra il 60 e il **90%** rispetto alle dimensioni delle risorse non compresse [17].

Quando possibile, salvare in cache le risorse che cambiano meno nel tempo

È bene mettere nella cache del browser le risorse che vengono utilizzare spesso. Questo permette di evitare il download di una risorsa ad ogni accesso di pagina, migliorando sensibilmente le performance sia reali, sia percepite [18].

1.2 Progressive enhancement

Come spiega bene Scott Jehl in "Responsible Responsive Web Design" [22], il layout è solo una delle diverse variabili che dobbiamo tenere in considerazione quando costruiamo siti e applicazioni multi-dispositivo. Dobbiamo assicurarci di essere "responsive" anche dal punto di vista delle **performance**, dell'**usabilità** e dell'**accessibilità**. E non sono sulla base della larghezza del dispositivo, ma bensì anche sulle diverse capacità che presentano i browser.

1.2.1 Differenziare le esperienze

Il *progressive enhancement* è una metodologia di sviluppo web che garantisce il massimo dell'accessibilità dividendo l'esperienza utente in **due livelli**: un primo livello dove vengono serviti al browser solo le **risorse essenziali** e dove si utilizzano solo gli standard web largamente supportati, in modo da mantenere il contenuto accessibile e navigabile dalla maggior parte degli utenti. C'è poi un secondo livello per gli utenti che usano browser che supportano un set di tecnologie più recenti. A questi si fornisce un'**esperienza**

arricchita delle ultime tecnologie più evolute ed aggiornate, forti del fatto che verranno supportate (sempre restando all'interno degli standard web).

Questo approccio permette di essere maggiormente pronti ai cambiamenti di condizioni che si potrebbero presentare in futuro (*future proof*), non essendo vincolati a fornire a tutti i browser la stessa esperienza utente.

Contenuti sempre accessibili

Il web si basa su tre livelli distinti che rappresentano rispettivamente il contenuto, il livello presentazionale, e il comportamento.

Il contenuto è il cardine del web ed è per questo motivo che il linguaggio HTML è creato con un meccanismo di *fault tolerance*. Questo garantisce che in caso di errori, si tratti di errori di sintassi o di elementi non riconosciuti dal browser, l'elaborazione della pagina web non venga bloccata. Lo stesso approccio è valido per i fogli di stile CSS: nel caso dovessero esserci errori o sintassi non riconosciute, la caratteristica di *fault tolerance* permette di ignorare solo la regola o il blocco di regole non riconosciute e di continuare nella lettura del CSS.

Esperienza arricchita

JavaScript, a differenza di HTML e CSS, non presenta la caratteristica di *fault tolerance*. Questo lo rende un layout meno stabile al quale affidare la funzionalità basilare del sito. Il JavaScript quindi farà parte, insieme alle ultime regole standard CSS, dell'esperienza "arricchita", che verrà fornita ai browser più moderni.

1.2.2 Funzionamento tecnico

Implementare questo approccio vuol dire servire "di default" una versione semplice del sito in HTML e CSS, con tutti i requisiti necessari affinché il sito sia accessibile e navigabile. Insieme al contenuto base del sito, aggiungiamo un file JavaScript che ha la funzione di testare le capacità del browser. Sia nel caso in cui il browser fosse moderno, sia in quello in cui il browser fosse vecchio e pertanto non in grado di supportare le capacità

richieste per la versione arricchita, salviamo il risultato su un cookie; in questo modo non dovremo effettuare i test ad ogni caricamento di pagina. Se invece il browser ha le capacità per ottenere una corretta visualizzazione della versione arricchita, dopo aver salvato il risultato su un cookie, andiamo ad aggiungere una classe *enhanced* all'elemento *<html>* e a caricare eventualmente le risorse aggiuntive.

[esempio pratico con galleria di immagini?]

Capitolo 2

Strumenti

Andiamo qui a raccogliere una serie di pratiche e di strumenti che utilizzeremo per applicare il *responsive web design* e il *progressive enhancement* al sito del Museo Virtuale di Informatica. Sono elencate sia discipline di analisi e progettazione, sia strumenti meramente tecnici che aiutano nella produzione del codice.

2.1 Architettura delle informazioni

L'architettura delle informazioni (*Information Architecture* — IA) è la struttura organizzativa logica e semantica delle informazioni, dei contenuti, dei processi e delle funzionalità di un sistema o ambiente informativo. Da un punto di vista cognitivo, possiamo affermare che una buona architettura delle informazioni è necessaria per rendere i contenuti accessibili e utilizzabili.

Per implementare al meglio il *responsive web design* diventa cruciale l'organizzazione delle informazioni e organizzare in modo coerente e funzionale le informazioni, sia da un punto di vista generale (navigazione e alberatura del sito), sia da un punto di vista più dettagliato (disposizione degli elementi in pagina).

L'architettura delle informazioni è una disciplina che mira a gestire le informazioni che ci circondano al fine di limitare il più possibile lo sforzo cognitivo a cui siamo sottoposti

per usufruirne.

Si tratta infatti di un'attività, prima ancora che di una disciplina, che si occupa della relazione uomo-informazione. Consiste nel:

- classificare, organizzare e raggruppare i contenuti;
- rendere le informazioni trovabili, comprensibili e usabili;
- progettare i sistemi di navigazione, di ricerca e di condivisione dei contenuti.

Spesso viene definita come una "disciplina ponte" perché sono molte le conoscenze coinvolte nel momento della sua applicazione: scienze dell'informazione, teorie della classificazione, logica, linguistica, user experience, user centered design, web design, psicologia cognitiva ed ergonomia.

Strumenti - Wireframing

Per avere un riferimento visivo del nuovo sito, sono stati creati dei mockup (bozze grafiche a bassa fedeltà) dello stesso attraverso Balsamiq (https://balsamiq.com). Per ogni tipologia di pagina è stato creato un mockup, in questo modo è stato possibile capire la complessità del progetto grafico e quali erano i punti ai quali prestare maggiore attenzione.

2.1.1 Applicazione

Oggi l'architettura delle informazioni si confronta con la progettazione di ecosistemi di comunicazione che comprendono numerosi strumenti, ambiti multimediali e interattivi (mobile devices, social networks, apps) che comunicano tra di loro. In questa tesi l'applicazione dell'IA è relativa all'organizzazione dei contenuti e dei meccanismi di navigazione del sito del MVdI. Questo su diversi livelli: dalla macro navigazione del sito alle informazioni presenti nel layout di pagina.

Più nel dettaglio, ci concentreremo su tutte le operazioni di:

- inventario dei contenuti e delle tipologie dei contenuti disponibili,
- organizzazione dei contenuti secondo una semantica definita,

- analisi e progettazione della navigazione del sito,
- presentazione dei contenuti in pagina,
- organizzazione dei processi per la progettazione e lo sviluppo di altri aspetti del sito.

Tralasceremo volontariamente tutto quello che riguarda la progettazione condivisa, il co-design, il service design in quanto non inerenti a questo progetto.

2.2 HTML

Con riferimento a quanto già scritto riguardo l'accessibilità (1.1.3), andiamo a definire un set di tool e di risorse utili al fine di scrivere codice markup semantico, con l'obiettivo di tenere il livello di accessibilità più elevato possibile.

Il markup del documento è sicuramente la base sulla quale si fonda la durabilità e l'accessibilità dello stesso, risulta dunque utile riportare una serie di regole e alcuni criteri da rispettare per tale scopo.

- Tutti i template del sito dovranno superare la **validazione HTML5** effettuata con il **W3C Validator** (https://validator.w3.org/);
- All'interno dei documenti, dove necessario, saranno utilizzate le **regole WAI-ARIA** (https://www.w3.org/TR/wai-aria/) per aumentare l'accessibilità dei contenuti dinamici e delle diverse parti dell'interfaccia utente;
- La navigazione dovrà essere garantita anche utilizzando solo la **tastiera**;
- Un ulteriore controllo sarà fatto con tool automatici come ***WAVE*** *Chrome Extension* [25], un plugin di Google Chrome che effettua dei test automatici sulla struttura del documento.

2.2.1 HTML5

Le novità introdotte dall'HTML5 rispetto all'HTML4 sono finalizzate soprattutto a migliorare il **disaccoppiamento fra struttura** (markup), **presentazione grafica** (tipo

di carattere, colori, etc definite dalle direttive di stile) e contenuti di una pagina web, definiti dal testo vero e proprio.

Tra le diverse migliorie che HTML5 ha portato rispetto ai suoi predecessori, ci sono gli elementi semantici (<*header*>, <*footer*>, <*section*>, <*article*>) che, insieme alle nuove funzionalità introdotte, possono permettere di creare siti internet più interattivi e connessi. Ad esempio si posso includere video e audio senza ricorrere a plugin esterni, sono previsti diversi modi di interagire con altre pagine e applicazioni online e offline, si possono infine aggiungere dati attraverso un attributo dedicato (*data-*).

2.3 CSS

Da un punto di vista più operativo, lo sviluppo del codice CSS richiederà alcuni accorgimenti sia nella scelta degli strumenti per produrlo, sia nella scelta dei nomi delle classi (namespace).
Approcci come *BEM* e *Atomic design* risolvono i limiti del linguaggio CSS definendo alcune regole relative a come dichiarare i selettori. Infine si eviterà la creazione materiale del CSS in favore del metalinguaggio SASS, che una volta compilato andrà a generare codice CSS.

2.3.1 Atomic design

L'idea alla base dell'Atomic design (espressione coniata per la prima volta da Brad Frost nel suo blog [26]) è che lo sviluppo front end, più che uno sviluppo di template, è uno sviluppo di sistemi di moduli che, raggruppati tra di essi, danno vita ai template e quindi alle pagine vere e proprie di un sito.

L'approccio prevede di "stilare" i singoli elementi che, una volta affiancati e montati su strutture sempre più complesse, vanno a comporre una pagina.

Con un (parziale) parallelo con il mondo della fisica, gli elementi di una sito vengono così suddivisi:

1. Atomi: gli atomi sono gil elementi più semplici dell'interfaccia, come bottoni, elementi di input, etichette (label).

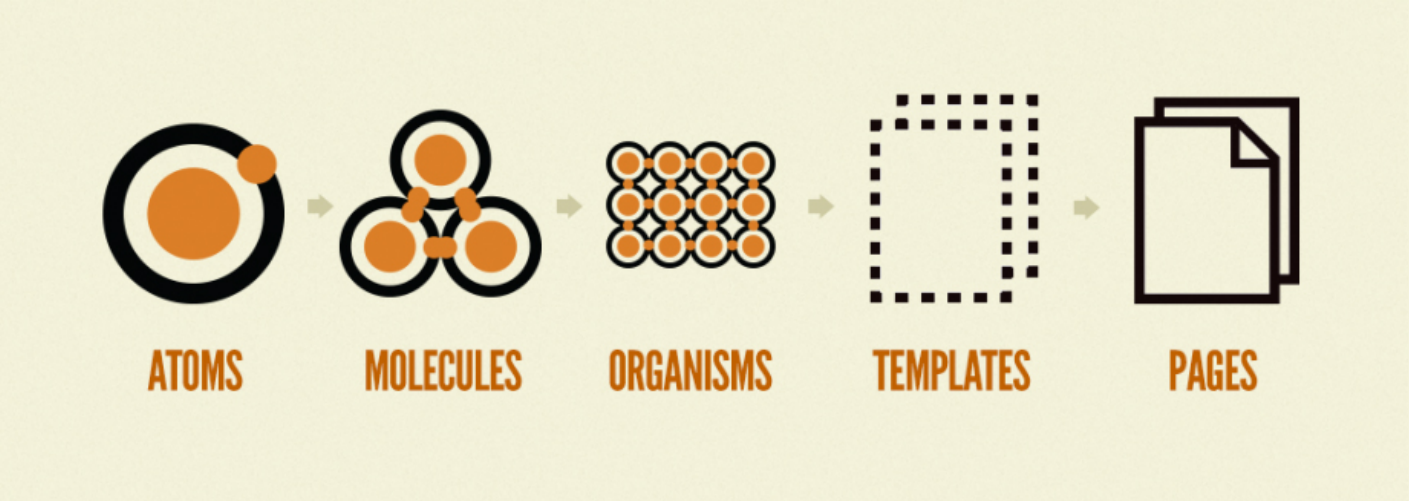


Figura 2.1: Dal sito di Brad Frost `http://bradfrost.com/blog/post/atomic-web-design/`

2. Molecole: quando si iniziano a combinare gli atomi tra di loro si dà vita alle molecole. Ad esempio un elemento di input più un'etichetta e un bottone posso dare vita a una casella di ricerca.

3. Organismi: più molecole insieme danno vita agli organismi. L'esempio più calzante potrebbe essere rappresentato dall'header di una pagina che al suo interno contiene, tra le altre cose, la casella di ricerca di cui sopra.

4. Template: più organismi (header, footer, sezioni varie) vanno a comporre i template interi.

5. Pagine: i template, una volta popolati con i contenuti andranno a comporre le pagine vere e proprie del sito.

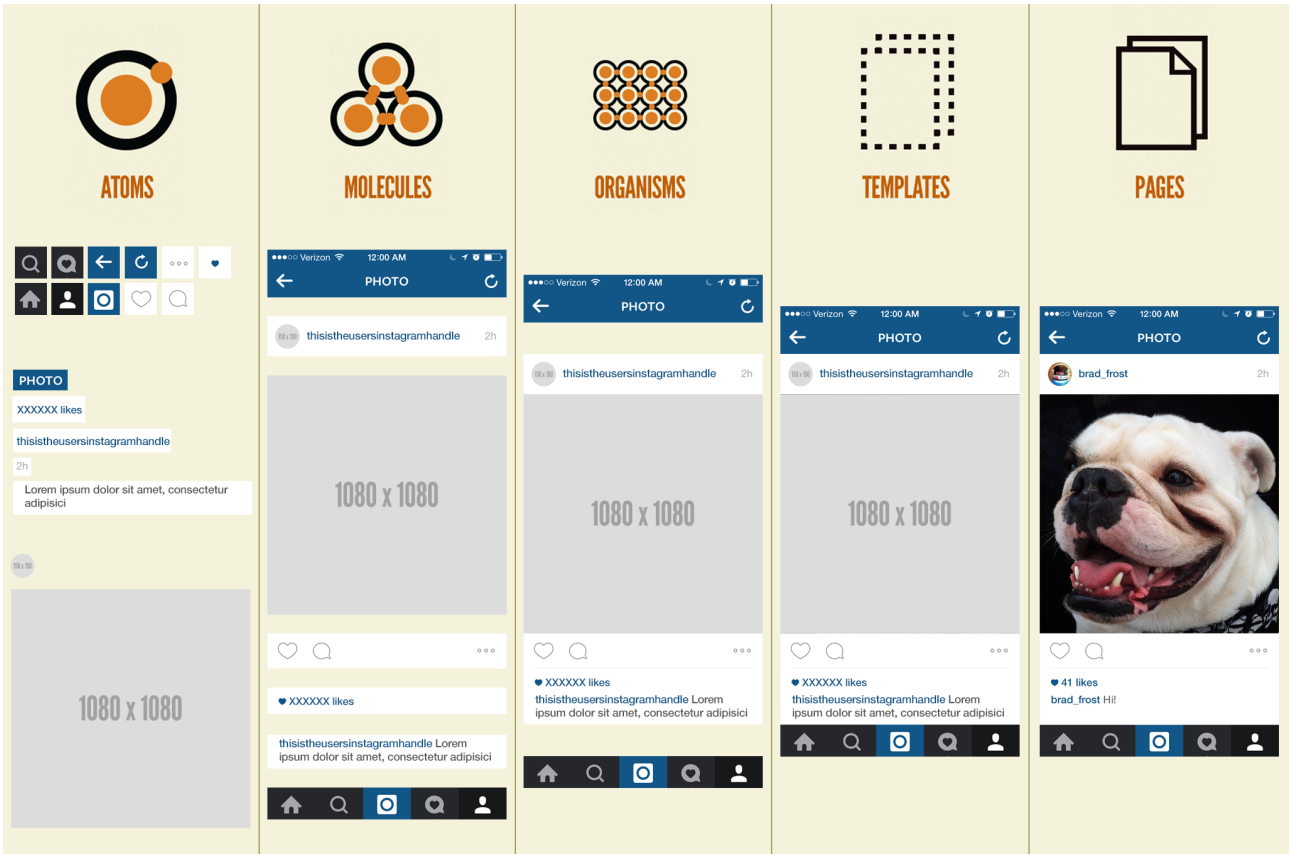


Figura 2.2: La pagina di Instagram analizzata secondo l'atomic design

2.3.2 BEM

Creato dal team di sviluppo di Yandex, BEM è l'acronimo di *Block Element Modifier* ed è un approccio allo sviluppo basato sui componenti [27]. L'idea è di dividere l'interfaccia grafica in blocchi indipendenti così da poter riutilizzazione quanto più il codice e sviluppare in modo più facile e veloce le interfacce.

Un esempio di un semplice modulo di ricerca:

```
<form class="search-form">
    <!-- input element in the search-form block -->
    <input class="search-form__input">
```

```
    <!-- button element in the search-form block -->
    <button class="search-form__button">Search</button>
</form>
```

2.3.3 SASS

Per generare il codice CSS utilizzeremo il metalinguaggio SASS (http://sass-lang.com/). Questo metalinguaggio ci permetterà di avere a disposizione diverse funzionalità che il codice CSS nativo non presenta. Avremo quindi a disposizione variabili, funzioni, estensioni, nesting e altre caratteristiche non presenti nel linguaggio nativo [28].

Utilizzando SASS inoltre, andremo ad organizzare i file in modo che essi siano separati e organizzati secondo le logiche citate in precedenza (2.3.2 e 2.3.1) andando a produrre comunque un unico output finale (1.1.5).

2.4 JavaScript

Come anticipato nella sezione *progressive enhancement* (1.2), nel nostro caso le funzionalità di JavaScript rappresentano un miglioria e un arricchimento dell'esperienza utente. Più in particolare, andremo ad utilizzare JavaScript per arricchire l'esperienza gestendo gallerie di immagini e caricando, quando possibile, progressivamente i file CSS al fine di migliorare le performance del sito ed apportare altre piccole migliorie.

2.5 Living style guide

Per documentare il lavoro svolto, in particolare la parte relativa al front end, andremo a redigere una *(living) style guide*.

Una *living style guide* è un documento che mostra tutti i moduli disponibili relativi al front end e come questi sono realizzati in HTML, CSS (SCSS) e JavaScript. Al suo interno ci sono esempi relativi a come usare un modulo, alla tipologia di varianti dello stesso, al codice che lo genera, alle possibilità di utilizzo ad esso legate ed, eventualmente, è anche presente della documentazione aggiuntiva.

Questo documento diventa a tutti gli effetti il punto di riferimento per gli sviluppi futuri, garantendo così uniformità e manutenibilità del codice nel tempo.

Nel web si trovano molti esempi di style guide, diversi per complessità e per scopi di documentazione. Di seguito un elenco di alcune ottime style guide dal punto di vista della completezza e della complessità:

- **Gov.uk**, la style guide per il governo inglese:
  `http://govuk-elements.herokuapp.com/`;
- **Code for America**:
  `http://codeforamerica.clearleft.com/`;
- **Rizzo**, la style guide di Lonely Planet:
  `http://rizzo.lonelyplanet.com/styleguide/`;
- **GEL**, la style guide della BBC:
  `http://www.bbc.co.uk/gel`.

2.5.1 Testing

Per l'attività di testing sul desktop, abbiamo effettuato i test su tutti i browser installabili sulla principale macchina di sviluppo (quindi Safari, Chrome, Firefox, Opera su un MacBook Air) e su IE11 eseguito su una macchina virtuale (`https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`).

Per quanto riguarda invece l'attività di testing sui dispositivi mobili abbiamo fatto riferimento sia a dispositivi reali sia a interfacce su dispositivi reali. Per i dispositivi reali abbiamo utilizzato il *Open Device Lab* di Bologna. Per i test emulati, sempre grazie all'Open Device Lab che ha messo a disposizione il proprio account, abbiamo utilizzato il software BrowserStack [37].

Open Device Lab

Un *Open Device lab* (ODL) è uno spazio dove chi sviluppa siti internet può testare gratuitamente quello che sta creando su tablet e cellulari messi a disposizione da altre persone.

L'idea originale viene da un post di **Jeremy Keith** [38], socio fondatore di una delle agenzie web più rispettate al mondo [39], dove confermando che è meglio fare test su dispositivi reali piuttosto che su emulatori, notava quanto fosse difficile procurarsene.

Avendo molti telefoni in disuso lui stesso, ha pensato di portarli nel suo studio per usarli nei suoi test e ha invitato tutti gli sviluppatori locali a usarli gratuitamente e a contribuire nello stesso modo. Da questo piccolo gesto è nato un movimento di sviluppatori / studi / agenzie che condividono gratuitamente dispositivi e spazi con altri sviluppatori in tutto il mondo.

Una volta raggruppati tutti gli ODL nel mondo [40], i produttori stessi di cellulari si sono interessati al movimento e hanno iniziato a fare donazioni e a supportare in qualche modo questa nuova comunità. Tra questi c'è il sito **Browserstack** che ha messo a disposizione di tutti gli ODL di un account illimitato del loro servizio.

A Bologna c'è un Open Device Lab ed è stato aperto da Andrea De Carolis (il sottoscritto) e Sandro Stefanelli a marzo del 2014. Proprio questo device lab ha fornito gli strumenti per effettuare dei test più approfonditi sul MVdI.

Browser e dispositivi

Su ambiente desktop siamo riusciti, grazie anche ad una macchina virtuale, a testare i seguenti ambienti:

- Su ambiente Mac:
  - Firefox / gecko, versione 30+;
  - Safari / webkit, versione 6+;
  - Chrome / webkit, versione 20+;
  - Opera / webkit, versione 15+;

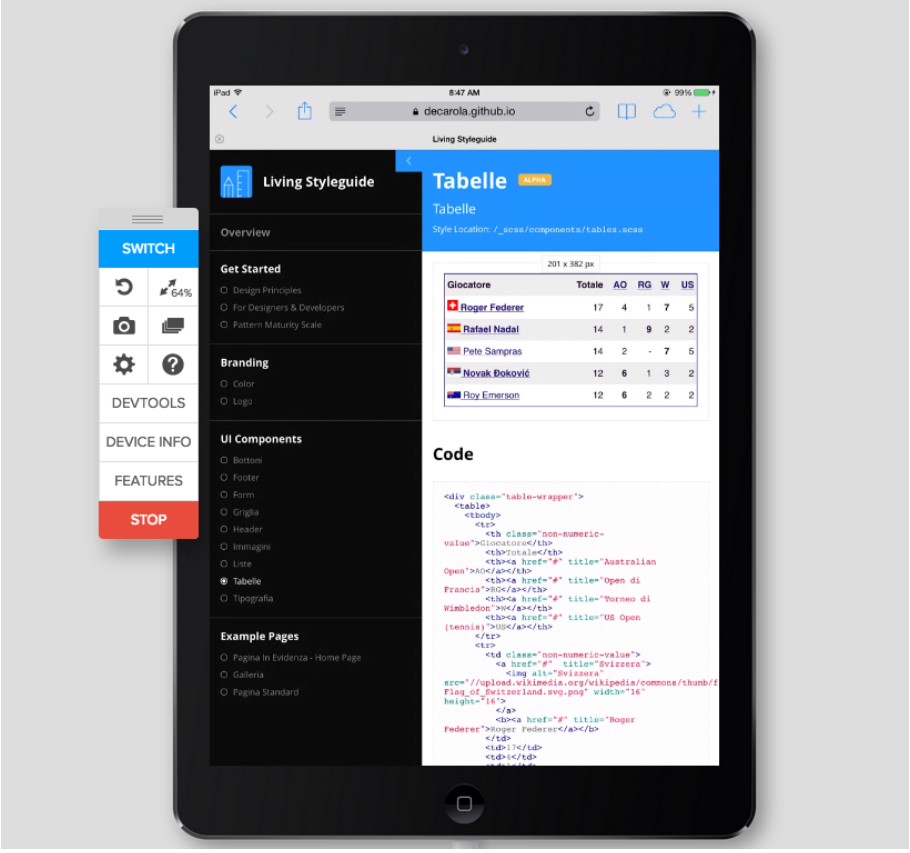


Figura 2.3: Una fase di testing con Browserstack

- Su ambiente Windows (8.1, emulato):
  - Internet Explorer / Trident, versione 11+.
  - Chrome / webkit, versione 20+;
  - Firefox / gecko, versione 30+;

Per tablet e smartphone, abbiamo testato sui dispositivi del ODL:

- Apple iPad 2 (iOS 7);
- Apple iPhone 6s (iOS 10);
- Nokia X3 (Simbian);

- Htc Desire C (Android 4.0.3);
- Samsung GT-S5360 (Android 2.3.5);
- Samsung Galaxy Nexus (Android 4.3);
- Sony Xperia U (Android 2.3.7).

2.6 WordPress

WordPress è uno dei CMS (*Content Management System*) più utilizzati al mondo (ad aprile 2016 le statistiche lo vedevano installato sul 26% dei siti del mondo [29]) ed ha al suo seguito una delle più grandi comunità di sviluppatori. Nato nel 2003 dall'intuizione proprio di due sviluppatori, rilasciato sotto licenza GPLv2, WordPress è cresciuto negli anni e rappresenta una soluzione elegante e con un'ottima architettura basata su PHP e MySQL.

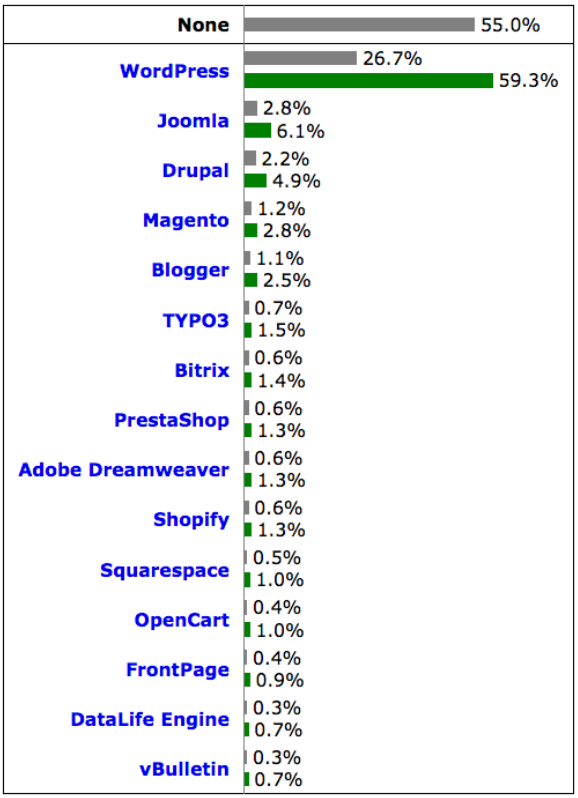


Figura 2.4: I primi 15 risultati del sondaggio effettuato da W3techs relativo ad aprile 2016 indica che WordPress è utilizzato sul 26% dei siti mondiali e con una quota di mercato rispetto ai sistemi di gestione dei contenuti del 59,3% (fonte: `https://w3techs.com/`)

2.6.1 Caratteristiche

Sono molteplici le caratteristiche che rendono WordPress uno dei CMS più usati nel web. Quelle che più sono utili al nostro scopo sono:

- La separazione netta, tramite **i temi**, tra il motore di WordPress e la parte di front end;
- La possibilità di estenderne le funzionalità tramite **plugin**. Questo aspetto può essere particolarmente utile per le future estensioni del MVdI;
- La gestione delle pagine con un sistema di **template**;

- La disponibilità di una vasta quantità di **temi gratuiti** (tra cui temi "skeleton" [30] con solo la struttura HTML completa dalla quale estendere la parte di templating);
- I ***permalink***, ovvero link facilmente leggibili (e indicizzabili) per far riferimento alle risorse del sito (pagine, post, etc);
- La gestione di una tassonomia basata su **categorie** e **tag**;
- Un **editor WYSIWYG** per la formattazione dei testi;
- Il supporto **multiutente** con diversi livelli di permessi;
- Il **log** degli utenti che visitano il blog;
- Un sistema di **aggiornamento** automatico.

2.6.2 Tema per MVdI

Quello che faremo sarà creare un tema ad hoc che funzionerà come base per le future estensioni funzionali del Museo Virtuale di Informatica.

Più in particolare, andremo a creare un'estensione di *unserscores*, il tema "skeleton" sviluppato dagli stessi sviluppatori di WordPress che contiene il minimo set indispensabile con tutte le informazioni e le componenti di un sito completo ([30]).

Capitolo 3

Riprogettazione del Museo Virtuale di Informatica

In questo capitolo documenteremo la riprogettazione del sito del Museo Virtuale di Informatica.

Prima lavoreremo sugli aspetti puramente progettuali quali architettura delle informazioni (3.1) e design (3.2), successivamente sugli aspetti relativi al codice front end (3.3) e back end / CMS (3.4).

3.1 Architettura delle informazioni

In questa prima sezione ci focalizzeremo sull'approccio **contenutistico**, andando a decidere come organizzare i contenuti all'interno del sito e all'interno della pagina.

3.1.1 Inventario

Per comprendere meglio la complessità e la diversità dei contenuti del sito faremo un **inventario** di quest'ultimi. Questo inventario ha il compito di mettere in evidenza:

- la **macro organizzazione** dei contenuti in relazione a tutto il sito;
- la diversità e il numero di **template** utilizzati per ogni pagina del sito;

- le **URL** attualmente in uso.

Analizziamo dunque ogni punto più nel dettaglio.

Macro organizzazione

Il MVdI è attualmente organizzato con una navbar che contiene collegamenti sia a pagine di contenuto, sia a pagine amministrative, sia a pagine non più attive. In particolare, oltre alla *Home page* abbiamo:

1. **Tour virtuale**;
2. **Cronologia**;
3. **Glossario**;

Più altre due pagine non necessarie all'utente finale:

4. **Dove siamo** (inattiva);
5. **Admin**;

Fortunatamente il numero dei documenti attivi permette una riorganizzazione manuale degli stessi.

Mentre le pagine *Admin* (pagina funzionale per accedere al CMS, da nascondere all'utente finale) e *Dove siamo* (pagina inattiva) non necessitano di un approfondimento, le pagine *Cronologia*, *Glossario* e *Tour virtuale* presentano i contenuti importanti per il sito e necessitano quindi di un approfondimento.

Il **tour virtuale** contiene le *stanze* che rappresentano i diversi periodi temporali significativi per la storia dell'informatica:

1. Dal big bang fino al 5000 a.C. (l'invenzione del linguaggio scritto);
2. Dal 5000 a.C fino al 1600 d.C. (la nascita della scienza moderna);

3. Dal 1600 fino al 1945 (la realizzazione del primo elaboratore moderno);

4. Dal 1945 fino al 1979 (la nascita del personal computer);

5. Dal 1979 fino al 1994 (la diffusione del www);

6. Dal 1994 fino a oggi (WiMax, IEEE 802.16);

7. Da oggi a domani.

All'interno di ogni stanza del tour virtuale ci sono 5 sezioni: i **temi principali** raccolgono degli argomenti di rilievo per quel periodo; le **teche** contengono del materiale multimediale (principalmente immagini) relativo ad oggetti o componenti di rilievo; i **punti interattivi** raccolgono al loro interno animazioni e parti interattive (spesso si tratta di piccole applicazioni realizzate in Adobe Flash o Java che mostrano oggetti altrimenti non reperibili come l'abaco giapponese o l'abaco cinese); La **cronologia della sezione** ospita una cronologia più dettagliata con gli eventi salienti relativi al periodo storico; infine, gli **approfondimenti** contengono ulteriori esplorazioni di di interesse e in generale ogni argomento di rilievo.

La **cronologia** è organizzata secondo lo schema delle "stanze" (anche in questo caso sette), come accade nella pagina del *Tour Virtuale*. Sempre come nelle singole *cronologia della sezione* del *Tour virtuale*, si ripercorrono gli eventi salienti relativi ad un periodo storico.

Infine il **glossario** del museo contiene circa duecentoventi voci con relative definizioni. La pima è *Algoritmo* e l'ultima è *WWW*.

Nei limiti del possibile, compatibilmente con i vincoli della struttura dati suggerita da WordPress (3.4), cercheremo di mantenere la struttura dei contenuti in modo da assicurare una **continuità** con quello che sarà il vecchio sito.

Figura 3.1: La pagina Home page del MVdI

3.1.2 Lista dei template

I template attualmente attivi contengono, oltre alla testata e al footer, un contenitore centrale che ospita gli elementi standard HTML (testi, immagini, elenchi puntati e tabelle) con poche varianti significative.

Andando più nel dettaglio relativamente ai template utilizzati, definiamo i tipi di layout presenti all'interno del sito. Gli stessi che poi verranno implementati dal backend e dal CMS.

Di seguito la lista delle pagine differenziate in base al contenuto che presentano:

- Home page (1);
- Tour virtuale (2);
  - Singola stanza (2);
  - Temi principali (2);
  - Teche (3);
- Cronologia;
  - Singola cronologia (2);
- Glossario (4).

Da questa lista possiamo dedurre la **lista dei template** da generare:

1. Home page / Pagina di evidenza
2. Pagina standard
3. Galleria multimediale
4. Glossario

Di conseguenza quest'ultima lista sarà la lista dei template che andremo di volta in volta a popolare con i contenuti delle pagine e dei post.

Data l'organizzazione asciutta dei contenuti, cercheremo di dare proprio a quegli stessi contenuti un'enfasi maggiore rispetto a quella riservata al loro contenitore. Ad esempio, per facilitare la lettura del testo (che compone la maggior parte del contenuto del sito) la lunghezza delle righe non supererà mai i 60-70 caratteri [24].

URL attive

Pur non essendoci nessun obbligo a riguardo, per garantire il servizio continuativo di una risorsa web nel tempo, è buona pratica tenere attivo l'URL anche dopo che questa risorsa viene spostata, sostituita o rimossa. Come progettisti possiamo dire di avere l'obbligo morale di tenere in vita i link che puntano alle risorse ed evitare il più possibile improvvise segnalazioni di errori 404.

Il CMS attualmente nel MVdI presenta delle URL con alcune caratteristiche non ottimali: in primo luogo è espresso in chiaro il linguaggio utilizzato per generare le pagine del sito (PHP) e a seguire è esplicito l'identificativo univoco del risorsa (ad esempio *page_id*).

Ecco ad esempio alcune URL attive:

Stanza del museo	**Indirizzo URL**
Home page	/files/viewpage.php?page_id=1
Tour virtuale	/files/viewpage.php?page_id=2
Sezione "Dal big bang fino al 5000 a.C. (l'invenzione del linguaggio scritto)."	/files/articles.php?article_id=1
Teca sezione 1 "Dal big bang fino al 5000 a.C."	/files/photogallery.php?album_id=1

Nuova versione con URL "parlanti"

Pur non avendo una provata valenza semantica, abbiamo ritenuto utile riscrivere le URL in modo che siano "parlanti" e quanto più possibili vicine al linguaggio umano e al contenuto che contengono.

Alcune url di esempio:

Stanza del museo	Indirizzo URL
Home page	/
Tour virtuale	/tour-virtuale
Sezione “Dal big bang fino al 5000 a.C. (l’invenzione del linguaggio scritto).”	/tour-virtuale/dal-bing-bang-fino-al-5000-ac
Teca sezione 1 “Dal big bang fino al 5000 a.C.”	/tour-virtuale/dal-bing-bang-fino-al-5000-ac/teca

È bene ricordare ancora una volta che prima di effettuare la migrazione dalle vecchie URL alle nuove occorre assicurarsi che ci sia un mapping tra il vecchio URL e il nuovo URL.

Redirect vecchie URL

Dato che potrebbero essere attivi dei link in ingresso per le vecchie URL, gestiremo i redirect attroverso il file .htaccess del webserver Apache.

Il redirect permanente, codice HTTP 301 ”Moved Permanently”, indica ai client HTTP, tipicamente browser, o agli spider che l’URL richiesta è stata trasferita in maniera permanente verso un nuovo indirizzo.

Un esempio del codice di redirect:

```
Redirect 301 /files/viewpage.php?page_id=1 http://www.MVdI.com/tour-virtuale
```

3.1.3 Alberatura e macro organizzazione

Andiamo quindi a rappresentare un’**alberatura** dei contenuti del sito, raggruppati per tipologia. Questa rappresentazione ci aiuta a capire come sono organizzati i contenuti e quanto sono complessi.

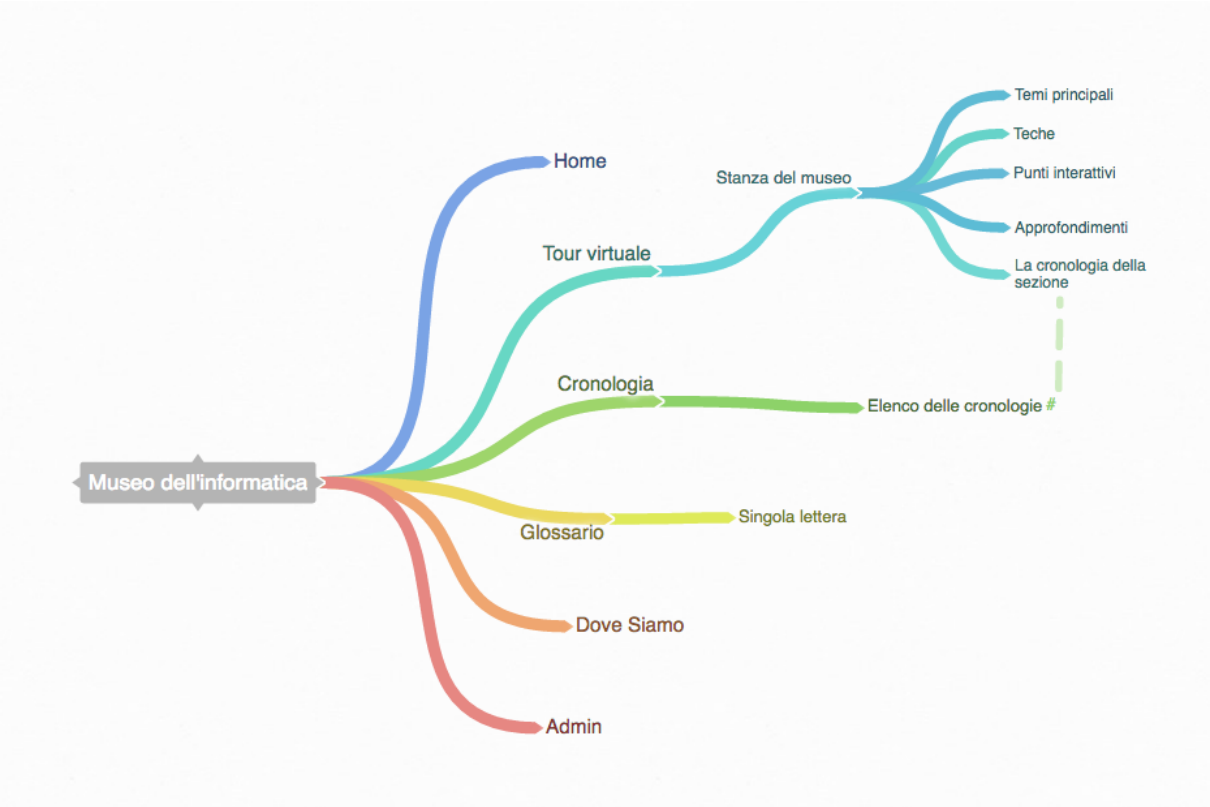


Figura 3.2: L'alberatura dell'attuale MVdI

Non stravolgeremo l'organizzazione attuale dei contenuti ma piuttosto andremo a riorganizzarla e a snellirla: terremo nella navigazione principale (quella presente nell'header) le pagine strettamente necessarie alla consultazione del sito. Le altre pagine invece saranno visualizzate solo nella navigazione secondaria (nel footer).

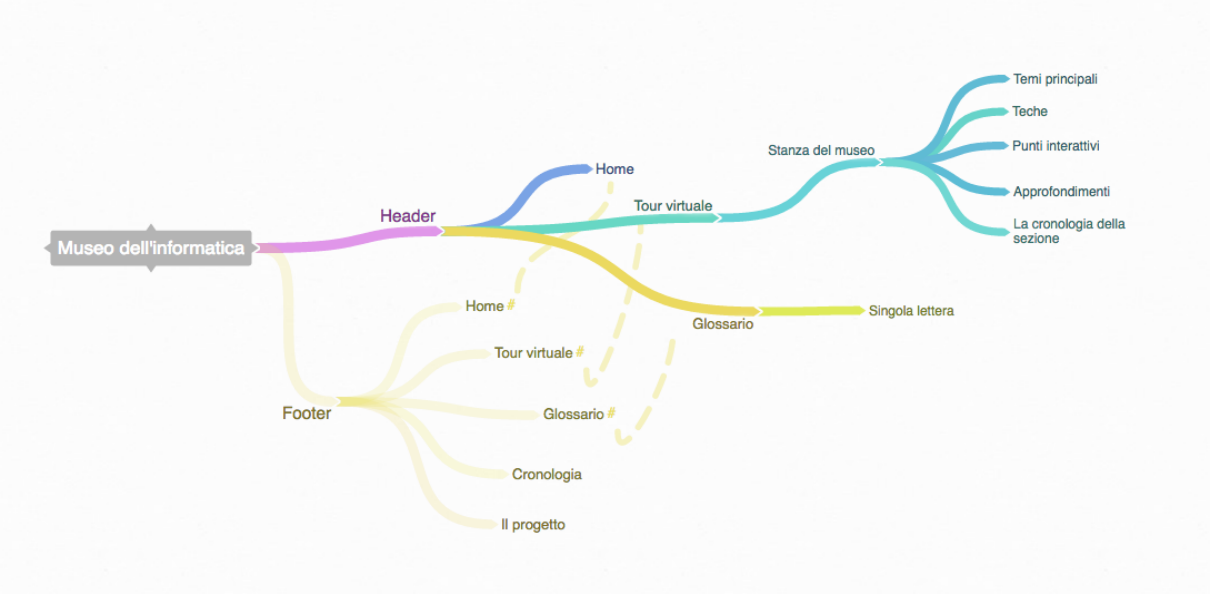
Quindi avremo una navigazione principale così organizzata:

- Home page,
- Tour virtuale,
- Glossario.

Mentre nella parte bassa del sito avremo la navigazione completa, sostituendo la pagina *dove siamo* con una pagina di descrizione del progetto:

- Home page,
- Tour virtuale,
- Glossario,
- Cronologia,
- Il progetto.

All'interno delle sezioni *tour virtuale* manterremo le stesse voci di menù andando a rimuovere solo le sezioni che non hanno contenuti al loro interno. Di conseguenza l'alberatura del sito risulterà così aggiornata:



Questa struttura andrà applicata al CMS e alle strutture dati relative ad esso: mentre l'implementazione fisica della struttura dati (sia essa via database o salvata su file) ci interessa relativamente per i nostri fini, la rappresentazione logica è molto importante.

3.2 Immagine coordinata

In questa sezione andiamo a documentare le scelte grafiche e stilistiche. Pur non essendo l'aspetto puramente grafico negli obiettivi del lavoro svolto, abbiamo valutato gli elementi grafici come parti di interazione con l'interfaccia grafica, assicurando e prediligendo quindi gli aspetti di leggibilità e di accessibilità rispetto a quelli puramente estetici.

Raccogliamo quindi le regole che definiscono l'immagine coordinata. Queste regole assicurano la leggibilità e di accessibilità del risultato finale e garantiscono uno sviluppo futuro omogeneo a ciò che abbiamo fatto fino a questo momento. Il set minimo di regole di immagine coordinata da conoscere chiaramente per poter descrivere l'identità grafica di un sito internet riguarda la **tipografia**, i **colori**, il **logo** e le **forme e decorazioni**.

3.2.1 Colori

Per garantire un livello più alto di accessibilità dal punto di vista della lettura, sono stati utilizzati diversi tool che monitorano il grado di contrasto tra i colori:

- Random A11y Color Palettes (http://randoma11y.com) è un sito che genera accoppiate di colori che hanno tra di loro un giusto grado di contrasto;
- Contrast ratio (http://leaverou.github.io/contrast-ratio/) è un tool creato da Lea Verou (W3C Invited Experts al CSS Working Group) che fornisce informazioni riguardo il grado di accessibilità di una coppia di colori;

Figura 3.3: La palette cromatica attualmente in uso.

All'interno della versione attuale del MVdI, sono utilizzati tre colori principali. La tonalità bordeaux (#AD2B50) e la tonalità grigia (#555) per i testi e per i link, il bianco (#FFF) come colore di sfondo.

Le prime due varianti (bordeaux e grigio), quando utilizzate su sfondo bianco o come sfondo per una scritta bianca, hanno un rapporto di contrasto sufficiente (rispettivamente 6.5 e 7.5). Purtroppo il rapporto di contrasto crolla quando le utilizziamo insieme (1.1), come si può facilmente vedere da questo esempio:

Figura 3.4: La palette cromatica applicata ai testi.

Nuovi colori

Per la scelta dei nuovi colori si è fatto riferimento alle specifiche W3c sul contrasto (https://www.w3.org/TR/WCAG/#visual-audio-contrast) e, tramite gli strumenti appena citati, si è scelto il seguente abbinamento di ottano (#508894), grigio (#23050B, questa volta derivato dal rosso), e bianco (#fff):

Figura 3.5: La nuova palette cromatica attualmente in uso.

Questa scelta di colori permette l'utilizzo degli stessi in ogni possibile combinazione, essendo il contrasto derivante dall'accoppiamento pari a:

- #EFEFEF + #23050B = 16.6 (Livello AAA per tutti i testi)
- #508894 + #23050B = 4.8 (Livello AA per i testi di tutte le dimensioni e livello AAA per i testi superiori a 18pt o in grassetto superiori a 14pt);
- #508894 + #EFEFEF = 3.5 (Livello AA: per testi grandi - superiori a 18pt o in grassetto superiori a 14pt);

Figura 3.6: La nuova palette cromatica applicata ai testi.

La sicurezza di avere tonalità cromatiche che riescano a garantire un sufficiente livello di contrasto in qualsiasi abbinamento ci fornisce un set di coppie di colori che si possono usare con eventuali nuovi moduli, senza doverci preoccupare ulteriormente di questo aspetto di usabilità.

3.2.2 Tipografia

Attualmente tutti i testi sono in **Verdana**, con le grandezze dei font che variano da 13px per i corpo testo (utilizzati spesso dentro le tabelle invece che all'interno di tag <*p*>) a 19.5px per i titoli.

Possiamo apportare migliorie dal punto di vista della leggibilità anche solo aumentando la grandezza dei font. Proseguiamo dunque in questa direzione aggiungendo anche un web font.

I web font sono una delle novità portate dal CSS3. Da ormai diversi anni infatti si possono includere nelle pagine web dei font esterni a quelli presenti sulla macchina del browser. **Google font** è un'ottima libreria di web font rilasciati sotto licenza libera dalla quale attingere.

Roboto by Google

Per la tipografia abbiamo scelto di utilizzare il font **Roboto**. Roboto è un font sviluppato da **Google** per i suoi sistemi Android ed è stato rilasciato sotto licenza Apache ([31]). Il fatto di essere un font progettato per l'utilizzo sullo schermo, di essere completo con più di sei pesi differenti, di essere rilasciato con una licenza che ne permette l'utilizzo liberamente e infine di essere disponibile gratuitamente nella versione per il web, rende il Roboto un ottimo candidato per i nostri scopi.

Figura 3.7: Esempi di utilizzo del font Roboto

3.2.3 Logo

Attualmente non è presente un vero e proprio logo del museo. Ciò che, allo stato attuale, può richiamare maggiormente l'idea di logo, sono una serie di scritte animate (realizzate con Adobe Flash) che però non rendono chiara l'identità grafica del Museo. Oltre alla scritta "Museo dell'Informatica", sono presenti delle Clip art e delle animazioni sulla destra.

Il nuovo logo è la trasposizione tipografica della dicitura "Museo dell'informatica". Le uniche accortezze utilizzate per la creazione del logo sono:

- Deve restare leggibile su sfondo chiaro/scuro
- Per le versioni più piccole si utilizzeranno le iniziali
- Non è previsto un pittogramma

Figura 3.8: Il nuovo logo del MVdI

Figura 3.9: La versione con solo le iniziali

3.2.4 Forme e decorazioni

L'unica forma di decorazione è rappresentata da alcune barre orizzontali grigie con del testo bianco al loro interno (nell'header e nel footer). Il testo al loro interno è di difficile lettura per via della sua dimensione (10px). Oltre a queste piccole forme, ci sono le decorazioni di default di HTML (liste puntate, link evidenziati).

Figura 3.10: L'attuale header contiene un testo con grandezza 10px

Atmosfera del sito

Con Atmosfera del sito, come suggerito da Andy Clarke [32], intendiamo le forme, gli spazi e altri aspetti visivi che contribuiscono a definire più nel dettaglio l'identità del sito.

Includiamo in questa definizione le ombreggiature, le spaziature intorno agli elementi di interattività (bottoni, form) e intorno al logo, eventuali pattern utilizzati negli sfondi, particolari scelte tipografiche.

Nel nostro caso ci limiteremo a:

- **Non usare ombre** (sempre a favore della leggibilità);
- Utilizzare (quando sarà necessario) un bordo con spessore pari a 4px;
- Usare variazioni di colori e di colori di sfondo che rispecchino i parametri di accessibilità che abbiamo citato in precedenza (sez. 3.2.1).

3.3 Front end

Adesso che abbiamo definito sia i contenuti che avremo a disposizione, sia in quale forma saranno visualizzati in pagina, possiamo iniziare a definire il codice front end affinché il sito venga correttamente visualizzato all'interno del browser.

Alla fine di questa sezione sarà presente un rimando alla *living style guide* che avrà la funzione di documentare dettagliatamente il lavoro fatto.

3.3.1 HTML

Il sito attuale è stato sviluppato usando *XHTML 1.0 Transitional* e non presente ottimizzazioni per la semantica. Interverremo per migliorare la semantica del documento, eliminando le tabelle utilizzate per ovviare limiti grafici, aggiungendo metadati per semplificare l'utilizzo tramite *screen reader* (regole WAI-ARIA) e infine assicurandoci che il layer di markup HTML sia quanto più separato da quello presentazionale (CSS).

Vediamo un esempio pratico. Questo è un estratto della navbar all'interno dell'header:

```
<table>
  [...]
  <tr>
    <td>
      <ul>
        <li><a href="viewpage.php?page_id=1">Home</a></li>
        <li><a href="viewpage.php?page_id=2">Tour virtuale</a></li>
        <li><a href="articles.php?article_id=74">Cronologia</a></li>
        <li><a href="articles.php?article_id=64">Glossario</a></li>
        <li><a href="http://www.cs.unibo.it/risorse/museum/csr_sdi.html">Dove
            Siamo</a></li>
        <li><a href="login.php">Admin</a></li>
      </ul>
    </td>
  </tr>
  [...]
</table>
```

Come si nota è stata utilizzata una tabella per poter visualizzare gli elementi in riga. Escamotage facilmente evitabile utilizzando regole CSS.

Lo stesso succede all'interno del **corpo pagina**, dove i contenuti sono ospitati per la quali totalità delle volte all'interno di tabelle:

```
<table>
  <tbody>
    <tr>
      <td class="main-body">
        <h2>Le stanze del museo</h2>
        Il percorso [...] riportati tra parentesi.
        <ol>
        <li><a href="articles.php?article_id=1">Dal big [...] </a></li>
          <li><a href="articles.php?article_id=2">Dal 5000 [...] </a></li>
          <li><a href="articles.php?article_id=3">Dal 1600 [...] </a></li>
          <li><a href="articles.php?article_id=4">Dal 1945 [...] </a></li>
          <li><a href="articles.php?article_id=5">Dal 1979 [...] </a></li>
          <li><a href="articles.php?article_id=6">Dal 1994 [...] </a></li>
          <li><a href="articles.php?article_id=7">Da oggi [...] </a></li>
        </ol>
        [...]
      </td>
    </tr>
    [...]
  </tbody>
</table>
```

Questa organizzazione del markup rende molto complessa sia la lettura del contenuto da parte di screen reader, sia la manutenzione e l'aggiornamento del sito, sia l'indicizzazione da parte dei motori di ricerca.

Nuova versione

La nuova versione dei template utilizzerà gli elementi semantici di **HTML5** insieme alle **WAI-ARIA rules**, permettendo così di ottenere una struttura di layout molto più snella e al tempo stesso molto più accessibile.

Qui di seguito un estratto del codice della navbar con gli accorgimenti appena elencati:

```
<nav role="navigation">
  <ul class="nav__pages">
    <li><a href="/">Home page</a></li>
    <li><a href="/tour-virtuale/">Tour virtuale</a></li>
    <li><a href="/glossario/">Glossario</a></li>
  </ul>
</nav>
```

Lo stesso approccio verrà utilizzato per i contenuti della pagina, dove andremo a sostituire gli elementi *<table>* con i relativi elementi semantici di HTML5.

```
<main role="main">
  <h2>Le stanze del museo</h2>
  <p>Il percorso [...] riportati tra parentesi.</p>
  <ol>
    <li><a href="articles.php?article_id=1">Dal big [...] </a></li>
    <li><a href="articles.php?article_id=2">Dal 5000 [...] </a></li>
    <li><a href="articles.php?article_id=3">Dal 1600 [...] </a></li>
    <li><a href="articles.php?article_id=4">Dal 1945 [...] </a></li>
    <li><a href="articles.php?article_id=5">Dal 1979 [...] </a></li>
    <li><a href="articles.php?article_id=6">Dal 1994 [...] </a></li>
    <li><a href="articles.php?article_id=7">Da oggi [...] </a></li>
  </ol>
</main>
```

3.3.2 CSS

Riscrivendo completamente gli stili, abbiamo l'occasione di implementare il più possibile gli approcci di *progressive enhancement* e *mobile first* elencati nella prima sezione.

Per l'organizzazione completa dei file SCSS che andranno a comporre gli stili del sito, seguiremo un'architettura basata su "Sass Guidelines", una guida scritta da Hugo Giraudel [35] e successivamente sintetizzata e riorganizzata da Matthew Elsom [36]. Questa consiste in un set di regole e *best practice* per mantenere il codice granulare al livello giusto, leggibile e ordinato.

Naming convention

Sia per rispettare le pratiche che abbiamo elencato (2.3.2 e 2.3.1), sia per non ripetere dichiarazioni (*don't repeat yourself* - DRY), applicheremo le seguenti convenzioni:

1. **Selezioneremo** gli elementi **facendo** sempre **riferimento alla classe** e non all'elemento di HTML. Questo ci permetterà di riutilizzare le struttura dati anche con elementi contenutistici diversi. Inoltre utilizzando le classi al posto degli ID non siamo vincolati dall'unicità del selettore ID.

2. Implementeremo la naming convention basata su **BEM** (2.3.2).

3. Per alcuni aspetti puramente funzionali, faremo uso di prefissi specifici per le classi:

   (a) Le classi che sono utilizzate come **trigger JavaScript** avranno un prefisso "*.j-*" (es. j-navbar-menu).

   (b) Le classi che indicano un **cambio di stato** avranno un prefisso "*.is-*" (es. is-open).

4. Come regola generica, consideriamo che un elemento non sa quale elemento può contenere, ma sa da chi è contenuto. Questo vuol dire che nel caso di un elemento contenuto all'interno di un altro elemento, il file di stile dell'elemento contenuto avrà il **riferimento al contenitore**.

3.3.3 JavaScript

In alcune sezioni particolari del MVdI, come ad esempio all'intero dei **punti interattivi** nelle stanze del tour virtuale, JavaScript rappresenta un arricchimento dell'esperienza fondamentale (questi punti interattivi dovranno comunque fornire un'alternativa

testuale per essere pienamente accessibili). Tuttavia, nella consultazione e nella semplice navigazione dei contenuti, JavaScript deve rappresentare un aspetto trascurabile.

Per i nostri scopi iniziali, andremo ad utilizzare JavaScript per un set limitato di obiettivi:

- *Feature detection* per capire se un browser supporta particolari caratteristiche del CSS (specie quelle più avanzate);
- Loading progressivo di fogli di stile, al fine di migliorare le performance della pagina;
- Miglioramento la fruizione di contenuti fotografici, in particolare delle gallerie di foto;
- Piccole migliorie relative alla navigazione su schermi piccoli (*toggle* della navbar);

Analizziamone dunque alcune:

Feature detection

Utilizzeremo Modernizr per controllare il supporto da parte di un browser di una particolare caratteristica del CSS. Modernizr è una libreria JavaScript che esegue una serie di controlli all'inizio del caricamento della pagina (in modo asincrono per non rallentare il caricamento della stessa) e, una volta terminati questi controlli, aggiunge una serie di classi all'elemento $<html>$ per segnalare quali feature sono supportate e quali no.

Ad esempio, se si configura Modernizr per controllare il supporto di JavaScript (in primis), di **flexbox**, e delle **CSS Transforms**, ad ogni esecuzione verranno aggiunte all'elemento $<html>$, in base al supporto, le classi *js*, *flexbox*, *csstransforms* in caso affermativo; le classi *no-js* (che sarà attiva di default ed eventualmente sostituita in automatico), *no-flexbox*, *no-csstransforms* in caso negativo. Di conseguenza potremo organizzare i CSS in modo da soddisfare entrambi i casi e fornire le soluzioni più adatte sia se nel caso vengano supportate feature specifiche, sia nel caso in queste non lo siano.

Gallerie fotografiche

Per le gallerie fotografiche, in particolare per la pagina *teca* delle stanze virtuali, andremo ad utilizzare una piccola libreria jQuery supportata sui dispositivi mobili e compatibile fino a IE8: Featherlight [41].

Il funzionamento della galleria di immagini di default prevede che il link dell'immagine apra la pagina della singola teca. Con Featherlight attivo, al posto di essere reindirizzati sulla pagina del progetto, vedremo aprirsi un *lightbox* con il contenuto del progetto.

Per garantire la possibilità per l'utente di disattivare questa funzionalità, ogni volta che attiviamo il plugin allo stesso tempo aggiungiamo pulsante per disabilitarlo.

3.3.4 Documentazione e style-guide

Per non essere troppo vincolanti per gli sviluppi futuri, terremo la style guide del MVdI quanto più sintetica. In particolare essa avrà al suo interno queste aree (in ordine):

1. **Identity**. In questa sezione si documenteranno le scelte di identità grafica, quindi le palette cromatiche e il logo nelle sue varie versioni.

2. **Base elements**. Qui si documenteranno gli stili degli elementi base di HTML. Quindi tutti gli elementi di tipografia (heading ed elementi inline), le immagini, le tabelle.

3. **Components**. Dove si documentano gli elementi formati da più elementi case, come navbar, header, footer e sistemi di icone.

4. **Layout**. All'interno di questa sezione saranno documentati i vari layout di pagina disponibili.

5. **Examples**. Quest'ultima sezione includerà gli esempi veri e propri di pagina completa.

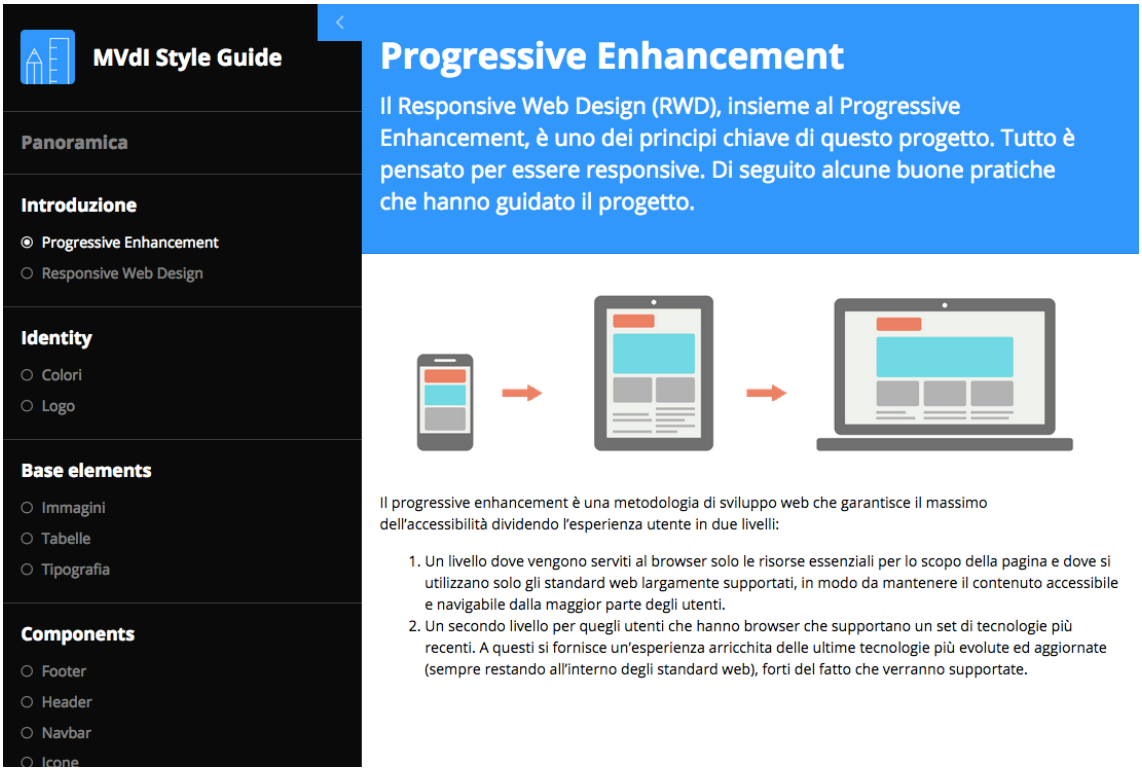


Figura 3.11: Una pagina della style guide.

Jekyll

Per riscrivere la style guide è stato utilizzato un generatore di siti statici: Jekyll [42]. Questo strumento ha permesso, da un punto di vista di markup, di poter definire dei template di pagina e di includere blocchi di codice che si ripetono nel sito.

Al risultato prodotto da Jekyll verrà poi aggiunto del JavaScript aggiuntivo per la gestione dei contenuti reali (WordPress). Esso sarà il riferimento statico di quello che poi dovrà sarà il risultato finale.

3.4 Back end

Infine, nella terza ed ultima parte andremo a documentare come applicheremo quanto sviluppato nel front end al **back end**, ovvero al relativo tema di WordPress, e dichiare-

remo quali miglioramenti andremo ad applicare per avere migliori rendimenti in termini di performance e usabilità.

3.4.1 WordPress

Una volta completato il setup inziale del database e del CMS, abbiamo preparato l'ambiente per iniziare ad importare i dati.

Impostazioni iniziali

Tra le modifiche fatte alle impostazioni iniziali, abbiamo pensato di togliere i riferimenti alle **date dai permalink**, sia per motivi di leggibilità delle URL (3.1.2) sia perché la data di creazione del post nel nostro caso ha un'importanza secondaria. Per quanto riguarda la **tassonomia**, abbiamo utilizzato l'organizzazione nativa di *categorie* e *tag* già integrata in WordPress, facendo riferimento ad un dualismo tra periodo storico e categorie, e tag e tipologia di contenuto. Per l'import dei dati abbiamo preferito utilizzare un approccio manuale data la non uniformità dei dati. A questo punto abbiamo a disposizione i contenuti finali del sito e possiamo cominciare la fase di templating.

Tema per il MVdI

Per lo sviluppo del tema, data la complessità che ha raggiunto il sistema di template di WordPress nel corso degli anni [([34]], abbiamo iniziato a lavorare partendo da un template *skeleton*, ovvero un template che presenta solo lo "scheletro" essenziale per poter funzionare. In particolare abbiamo utilizzato **Underscore**, creato e mantenuto dal team di Atomattic (l'azienda che mantiene e coordina WordPress e la relativa comunità).

Una volta creati i - andiamo ad applicare i moduli della style guide - lista dei template saranno i template di wordpress - il css sarà semplicemente importato, così come il js - andiamo ad aggiungere funzionalità di wordpress come gli snippet di codice in modo da avere le classi css a disposizione

3.4.2 Applicazione della style guide

Una volta che i contenuti sono gestiti dal CMS, andiamo finalmente ad applicare gli stili e i comportamenti definiti nella style guide. Questa operazione consiste, una volta assicurati che siano presenti i file CSS e JavaScript, nel aggiungere classi e attributi ai vari elementi del markup generato da WordPress.

3.4.3 Plug in di WordPress

Infine, per completare la preparazione del sito del MVdI, andiamo ad aggiungere alcuni (pochi) plugin di WordPress:

- **Glossario**. Per la gestione del glossario utilizzeremo un plugin dedicato. Questo plugin, dato un vocabolario di termini, trova le parole del vocabolario e crea un link attorno ad esse alla relativa voce del glossario [43].

- **Sicurezza**. Utilizzeremo un plugin per effettuare controlli di sicurezza sulle visite del sito (Firewall, blocco degli IP malevoli etc) [44].

- **Cache**. Essendo in sito relativo alla consultazione dei contenuti, per tenere le performance della generazione della pagina alte come quelle relative al front end (1.1.5, aggiungeremo un plugin per gestire una cache dei contenuti. In questo modo le pagine saranno generate solo la prima volta e poi salvate nella cache per ogni accesso successivo [45].

Bibliografia

[1] *Responsive Web Design* di Ethan Marcotte. 25 maggio 2010
`http://alistapart.com/article/responsive-web-design`

[2] *Responsive Web Design* di Ethan Marcotte. A book apart, Prima edizione 7 giugno 2011; Seconda edizione 2 dicembre 2014
`https://abookapart.com/products/responsive-web-design`

[3] Ricerca di comScore Inc `https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S`

[4] Installazione artistica *4D-Pixel* di Daan Roosegaarde.
`https://vimeo.com/14899669` di Studio Roosegaarde

[5] Installazione artistica *Liquid - interactive cocoon* di Daan Roosegaarde.
`https://vimeo.com/14899445` di Studio Roosegaarde

[6] Smart glass di Smartglass International, Irlanda
`http://www.smartglassinternational.com/`

[7] Interactive wall di Festo Bionic, Germania
`https://vimeo.com/4661618`

[8] "Mobile first" by Luke Wroblewski
`http://www.lukew.com/ff/entry.asp?933`

[9] "Mobile first" by Luke Wroblewski. Edito da A book apart, 2011
`https://abookapart.com/products/mobile-first`

[10] “the Performance Golden Rule” by Steve Souders
https://www.stevesouders.com/blog/2012/02/10/
the-performance-golden-rule/

[11] “Hypertext Transfer Protocol – HTTP/1.1”, capitolo 8, Connections
https://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html

[12] Da un test di Browserscope
http://www.browserscope.org/?category=network&v=top

[13] “Best Practices for Speeding Up Your Web Site”, Yahoo! Inc.
https://developer.yahoo.com/performance/rules.html#css_top=

[14] “High Performance Web Sites: Rule 6 Move Scripts to the Bottom” di Steve Souders
https://developer.yahoo.com/blogs/ydn/high-performance-sites-rule-6-move-scripts-
html

[15] HTTP Archive - Stats
http://httparchive.org/interesting.php#bytesperpage

[16] “Progressive Web App Road Show 2016” di Google
https://developers.google.com/web/fundamentals/performance/
optimizing-content-efficiency/image-optimization

[17] “Progressive Web App Road Show 2016” di Google
https://developers.google.com/web/fundamentals/performance/
optimizing-content-efficiency/optimize-encoding-and-transfer#
compressione-del-testo-con-gzip

[18] PageSpeed Insight di Google
https://developers.google.com/speed/docs/insights/
LeverageBrowserCaching

[19] PageSpeed Insight di Google
https://developers.google.com/speed/docs/insights/BlockingJS

[20] http://www.w3.org/TR/WCAG

[21] *Disposizioni per favorire l'accesso dei soggetti disabili agli strumenti informatici* anche nota come "Legge Stanca"
http://www.camera.it/parlam/leggi/04004l.htm

[22] "Responsible Responsive Web Design" di Scott Jehl. Edit. A book apart, 2014 https://abookapart.com/products/responsible-responsive-design

[23] "Design with progressive enhancement" di Todd Parker, Patty Toland, Scott Jehl, and Maggie Costello Wachs. Edit. New Riders, 2010 https://www.filamentgroup.com/dwpe/

[24] *Regole editoriali, tipografiche & redazionali* di Fabrizio Serra. Fabrizio Serra editore. Prima edizione luglio 2004.

[25] Plugin di Google Chrome che controlla in automatico le funzionalità standard riguardo l'accessibilità come le descrizioni alternative per le immagini, il contrasto tra i colori degli elementi, etichette nelle form
http://wave.webaim.org/extension/

[26] *Atomic design* di Brad Frost
http://bradfrost.com/blog/post/atomic-web-design/

[27] BEM (Block, Element, Modifier) is a component-based approach to web development.
https://en.bem.info/methodology/

[28] Sass: Syntactically Awesome Style Sheets
http://sass-lang.com/

[29] "Usage of content management systems for websites"
https://w3techs.com/technologies/overview/content_management/all/

[30] "Introduction To Underscores: A WordPress Starter Theme With Konstantin Obenland"
https://wptavern.com/introduction-to-underscores-a-wordpress-starter-theme-with-k

[31] The Roboto family of fonts
https://github.com/google/roboto/

[32] "Designing atmosphere" da "Hardboiled Web Design — Fifth Anniversary Edition" di Andy Clarke.

[33] *Open Device Lab!* di Andrea De Carolis
http://blog.decaro.la/2014/03/03/open-device-lab/

[34] "Template Hierarchy" dal Codex di WordPress
https://developer.wordpress.org/themes/basics/template-hierarchy/

[35] "An opinionated styleguide for writing sane, maintainable and scalable Sass."
https://sass-guidelin.es/

[36] "A Simple SCSS Architecture, and Best Practice Playbook" di Matthew Elsom.
http://matthewelsom.com/blog/simple-scss-playbook.html

[37] http://browserstack.com/

[38] *Left to our own devices*, dal blog personale di Jeremy Keith.
http://adactio.com/journal/5433/

[39] Clearleft agency, Brighton.
http://clearleft.com/

[40] Il sito di tutti gli Open Device Lab nel mondo, gestito da Andre Jay Meissner.

urlhttps://opendevicelab.com/

[41] Featherlight su GitHub.
https://github.com/noelboss/featherlight/

[42] Jekyll
https://jekyllrb.com

[43] Glossario di Daniele Scasciafratte
https://it.wordpress.org/plugins/glossary-by-codeat/

[44] Wordfence plugin
https://it.wordpress.org/plugins/wordfence/

[45] W3 Total Cache
https://it.wordpress.org/plugins/w3-total-cache/