Sic et Simpliciter

Guida Pratica al Progetto di Tecnologie Web

Mariano Sciacco

2018 — 2019

marianosciacco.it — *Informatica, UniPD*

Indice

1 Prefazione

Questa guida viene scritta con l'intento di dare uno spunto da cui partire e su cui basarsi con il progetto di Tecnologie Web a livello organizzativo, e raccoglie quanti più possibili accorgimenti, trucchi e consigli da seguire per avere meno criticità possibili a seguito della revisione finale dei professori.

Tutto ciò che viene riportato in questa guida proviene da un'esperienza di progetto reale. Ovviamente non tutte le pratiche descritte sono perfette, pertanto è opportuno confrontarsi direttamente con i professori in caso di dubbio, dando uno sguardo anche alle slide del corso.
Detto questo, buona lettura e mi auguro che vi possa essere utile!

Mariano

2 Introduzione

2.1 Front-end e Back-end

Per prima cosa, è opportuno differenziare due terminologie importanti che verranno usate spesso in questa guida: **front-end** e **back-end**.

Front-end: riguarda lo stile e l'area di *presentazione* del sito. In generale, tutta la parte con cui un utente si interfaccia con il sito e in cui l'elaborazione avviene nel client.
Fanno parte i seguenti linguaggi: `html, css, javascript`.

Back-end: riguarda l'area di *elaborazione e gestione dati*. Di questa parte se ne occupa il server e il client invia solamente dati.
Nel nostro caso, fanno parte i seguenti linguaggi: `php` e `sql`.

2.2 Premesse sulla valutazione

La valutazione dei professori si basa sull'analisi completa del sito sia a livello di programmazione che a livello di progettazione. In particolare, dei 34 punti predisposti per il progetto:

- 9 punti alla relazione
- 6 punti alla progettazione
- 3 punti all'accessibilità
- 2 punti per Javascript
- 3 punti per PHP
- i rimanenti punti sono partizionati tra HTML, CSS, Sito Mobile / versione di stampa e funzionalità del sito

In generale, è facile comprendere di come la parte **Front-end** sia quella vincente per una valutazione migliore. Ovviamente la disposizione dei voti può variare ogni anno, però l'idea di fondo non cambia.

2.3 Punti forti e punti deboli di un progetto

Durante le lezioni, i professori hanno espresso più volte di come la parte di usabilità e accessibilità sia importante per un sito. Nella parte di **valutazione**, però, si dà moltissimo peso a come si è lavorato a livello di programmazione. Pertanto, nonostante il sito sia bellissimo e ricco di funzioni, con un minimo errore si può portare alla perdita di punti fondamentali, specie se compaiono **errori durante la validazione del progetto**.

Questo implica che il progetto debba essere sicuramente in un buono stato di presentazione, ma ancora di più è importante concentrarsi sul come sono stati scritti i file e sulla correttezza sintattica e formale degli standard, specie per HTML e CSS.

L'ideale è tirare fuori un **numero contenuto di funzionalità** su cui concentrarsi e in cui perderci poco tempo, visto che gran parte della valutazione si basa sulla parte front-end.

Riassumendo `(TL;DR)`:

- Scegliere un giusto numero di funzionalità, senza esagerare.
- Le funzionalità più essenziali dovranno essere ben curate.
- Concentrarsi molto di più sui linguaggi *front-end*.
- Un piccolo errore può essere fatale nella valutazione.

3 Gruppo e Ambienti di lavoro

3.1 Scelta del gruppo e del coordinatore

La fase di composizione del gruppo è cruciale. A prescindere che tutti i membri del gruppo si conoscano o meno, è importante stabilire fin da subito le politiche generali con tutti. Queste politiche riguardano soprattutto la disponibilità per incontrarsi, l'ambiente di lavoro comune e l'idea generale del sito che si vuole realizzare.

E' semplice immaginare di come durante tutto l'arco di tempo capiteranno *incomprensioni tra i vari membri, flame gratuito e opinioni completamente differenti*, ma non c'è da preoccuparsi: tutto questo è normale.

Dal momento che questo avverrà, è bene fin da subito delineare il *referente principale* del gruppo che dovrà assumere il ruolo di **coordinatore generale**. In linea di massima, potrebbe essere la persona con un po' più di esperienza sul campo e/o in generale una figura che sia abbastanza estroversa e con un minimo di doti di leadership.

3.2 Approccio di lavoro e suddivisione dei ruoli

Per affrontare questo progetto ci sono due approcci:

1. Il primo approccio si basa sul pricipio di "**tutti fanno un po' di tutto**", in questo modo si cerca di garantire partecipazione di tutti, e ad ognuno viene semplicemente assegnato un ruolo di responsabilità a meno del coordinatore generale: **responsabile front-end, responsabile back-end e responsabile accessibilità & database**. Questo approccio risulta il più democratico possibile, ma meno focalizzato sul singolo argomento, perché tutti devono avere conoscenze trasversali sull'argomento e idealmente tutti toccano ogni linguaggio di programmazione, sia front-end che back-end. L'approccio più corretto sarebbe questo, dal momento che ci si scambiano più opinioni consapevolmente tra i vari membri del gruppo, costruendo il sito insieme.

2. Il secondo approccio si basa su una **divisione netta dei compiti**. Ad ognuno viene assegnato (o ci si auto-assegna) un compito da fare in cui ci si sente più pronti e, dopo un breve consulto con il gruppo per ciascun compito, si procede al completamento. Questo approccio è più focalizzato su un singolo argomento, ma talvolta dispersivo poiché ci si può ritrovare da soli e senza solidi confronti da parte degli altri membri.

Scegliere il giusto approccio di lavoro dipende dal gruppo e dalle personalità dei singoli. E' chiaro che con il primo, tutti devono far tutto, per cui l'apprendimento sarà più lungo e le opinioni saranno tante, talvolta diverse ma comunque costruttive, mentre col secondo si lascia più spazio ai singoli membri, ma aumenta il rischio di dover portare avanti anche il lavoro di qualcun altro o di lasciare indietro parti di progetto.

L'ideale sarebbe bilanciarsi su ciò che bisogna fare e chi lo deve fare, e in questo il coordinatore deve fare scelte opportune e quanto più in accordo con tutto il gruppo. Le scelte implementative alla fine dovranno sempre essere giustificate, soprattutto per stare dentro con i tempi di sviluppo.

3.3 Organizzare gli incontri

Personalmente, consiglio di organizzare in maniera anticipata gli incontri con il gruppo, più volte alla settimana con sessioni che variano dalle 2 alle 4 ore. Ovviamente il tutto deve essere concordato in base agli impegni di ciascuno ed è **opportuno trovarsi almeno in 3 persone su 4**, così da non lasciare indietro nessuno.

Consiglio: conviene predisporre un **gruppo Telegram** e un **gruppo Slack**, il primo per comunicazioni veloci e cazzeggio, il secondo per comunicazioni serie. Per il resto, consiglio di trovarsi su **Discord o Skype** (se esiste ancora) per fare *chiamate di gruppo*, qualora non possiate trovarvi di persona.

3.4 Github e Versionamento

Siccome siamo più informatici che farmacisti, per chi ancora non ha iniziato, consiglio caldamente l'utilizzo di **Github**. Questo semplice strumento garantisce velocità e comodità di programmazione e soprattutto permette di sapere lo storico di OGNI file e soprattutto di chi ha compiuto un eventuale misfatto.

Inoltre, con Github è disponibile un **Issue Tracking System** integrato fatto con i fiocchi, che permette di coordinarsi autonomamente sul da farsi, gestendo *features implementation* e *bug tracking* durante le fasi del progetto, senza il bisogno di avere fogli di Google Docs volanti.

- Guida per Issue Tracking System di Github
- Consiglio di leggersi le slides dei **laboratori di Tecnologie Open-Source (2018)** scritte dal *prof. Bertazzo* per poter padroneggiare bene Git e Github. Si trovano anche su MEGA.

Consiglio: meglio perdere un paio di incontri per capire bene come funziona Git e Github, piuttosto che versionare tramite MEGA o Google Drive. Il tempo usato per questo verrà ripagato con un tempo di coordinamento minore e meno incomprensione generale.

Consiglio 2: E' possibile richiedere il **Github Student Pack** registrandosi e facendo domanda con l'email universitaria. Questo permette di avere il **Premium** e di avere illimitati collaboratori nelle repo private (invece di solo 3).

Consiglio 3: Fate aprire la Repo a chi sa già usare meglio Git e Github o altrimenti al coordinatore generale.

3.5 Configurare l'ambiente di lavoro

L'ambiente di lavoro è la parte più importante di tutto il progetto. Sapendo che ognuno ha il proprio computer con il proprio sistema operativo bello o

brutto che sia, chiunque può stare certo che - fortunatamente - nel campo di Tecnologie Web configurare l'ambiente di lavoro è un gioco da ragazzi (o quasi).

In generale, per lavorare è sufficiente un tris di programmi che nel gergo si chiama **AMP** `Apache, Mysql, PHP` Con queste tre componenti si possono fare siti web del calibro richiesto per il progetto di TecWeb.

Con grande fantasia, per i tre sistemi operativi più usati (Windows, Linux, MacOs) troverete relativamente *WAMP, LAMP e MAMP*. Se avete Chrome OS o altri sistemi operativi Cloud Based, potete seguire questo link.

Vi riporto i seguenti programmi che vi consiglio per esperienza e che sono facili da installare:

Windows: personalmente consiglio vivamente **easyPHP**. E' un ambiente semplice, veloce e facile da installare. Una volta installato, vi basta aggiungere la directory in cui è presente il vostro progetto (direttamente dal pannello web, `add directory`) e avrete subito configurato il vostro sito in locale a modi `localhost/edsa-SITOBELLO`.

MacOs: per chi usa un Mac, consiglio di usare **MAMP**, la versione gratuita. La semplicità è paragonabile a EasyPHP, ma in generale una volta configurata la cartella del progetto dal menù `preferenze` avrete allo stesso modo `localhost/SITOBELLO`

Linux: per i più smanettoni consiglio **PHPStorm** che è a pagamento, tuttavia se ci si registra con l'email universitaria e si fa richiesta è gratuito. Bisogna configurare diversi parametri ed eventuali librerie che possono essere utili nel progetto, ma poi dovrebbe funzionare tutto senza molti problemi. In alternativa, **XAMPP** è una buona piattaforma alternativa, più leggera di PHPStorm.

Consiglio: come IDE la scelta è libera. Usate quello che preferite, personalmente consiglio **Sublime Text** visto che è molto leggero e ben integrato per PHP e HTML. Eventualmente poi potete installarci pacchetti aggiuntivi con Package Control.

3.6 Database remoto, Auto-deploy e sito online

Per poter avere sempre risultati concreti e per poter testare di volta in volta il proprio sito al di fuori dell'ambiente di lavoro, si consiglia di hostarlo su una piattaforma online (es: `altervista.org`, 000Webhost, vps personale) o nei server predisposti dell'università. Questo sarà molto utile soprattutto nella parte di validazione.

Per chi possiede un VPS personale, può utilizzare un semplice script di auto-deploy tramite un Webhook che manda una richiesta ad uno specifico file, e il server, una volta aggiunta una chiave SSH-RSA che lo autentica per il Deploy su Github, esegue dei comandi di `git fetch & pull` per scaricare le modifiche.

Faccio notare che con il **Github student pack** si hanno 50$ di voucher su DigitalOcean, che è un servizio che offre VPS a tempo di utilizzo, molto affidabile. E' sufficiente usare una taglia piccola di VPS (5$ al mese va bene) e volendo con una applicazione preinstallata come il pannello hosting *VestaCP* o l'ambiente *LAMP*.

In generale, però, se non volete addentrarvi su cose troppo avanzate, potete semplicemente installarvi in questo VPS un **Database MYSQL (accessibile da IP) con pannello PHPMyAdmin** (sempre tra le applicazioni preinstallate) da usare come database comune per tutti i membri del gruppo (ricordatevi di aprire le porte del MYSQL dal firewall del VPS). In alternativa anche `db4free.net` può essere una alternativa, solo che va a scadenza settimanale da rinnovare e può essere parecchio instabile, nonché lento (morale: se scegliete di usarlo, fatevi spesso backup).

Consiglio molto questa soluzione visto che il PHP permette di connettersi in remoto ai database, oltre che in locale, avendo così un database uguale ed unico per tutti, sempre se questo vi possa essere comodo (altrimenti lo si installa in locale, però va aggiornato manualmente da ognuno ogni qualvolta ci siano cambiamenti sulle tabelle).

Nota: usare il database in remoto, richiede la porta per MYSQL aperta. A casa o con le reti mobili non c'è problema, ma con le reti wireless dell'università ciò non funziona, a meno che non siate connessi con una VPN

(vostra o pubblica) sulla porta 443 (TCP). Volendo potreste sempre configurare una VPN sul server che avete predisposto, installando con uno script automatico di OpenVPN e far connettere ciascun membro del gruppo.

4 Le fasi del progetto

4.1 Progettazione

Una volta definito bene il gruppo, l'ambiente di lavoro e l'approccio da usare, si può iniziare con la parte di progettazione. Di seguito verranno descritti tutti i tips & tricks per questa fase, con i relativi consigli e suggerimenti da seguire.

4.1.1 Su cosa fare il sito?

Obiettivamente, conviene fare il sito pensando ad un approccio di tipo aziendale, pertanto sarebbe opportuno fare un sito che possa avere effettivamente un fine. Il prodotto, in ogni caso, può basarsi sull'interesse comune dei vari membri. Sconsiglio di fare siti ricchi funzionalità, troppo lunghi e complessi, e di concentrarsi piuttosto su qualcosa di semplice (perché *semplice è bello*) come blog, portfolio, mini social network o sito vetrina.

Quello su cui punteranno i professori è *analizzare profondamente la parte visitatori e utenti*: assicuratevi che questa parte sia abbastanza solida e con un minimo di funzionalità. Per la parte visitatori, ad esempio, si può pensare di implementare qualcosa come commenti su un post, sistema like o reaction, funzioni di ricerca, visualizzazioni di uno o più elementi, form di contatto, registrazione a newsletter, ecc. Allo stesso modo, per la parte utenti, si può integrare un'area con registrazione e login, insieme a sezioni private che integrano cambio impostazioni, password, ecc.

4.1.2 Approccio Bottom-Up o Top-Down?

L'approccio **Bottom-Up** si basa sullo svolgere prima la parte di Back-end, per poi riadattarla alla parte di presentazione, Front-end. In questo modo si ottiene una parte di elaborazione dati più avanzata, senza avere però feedback visivo di come verrà il sito.

L'approccio **Top-Down**, invece, parte con lo svolgere una prima parte di Front-end per avere uno scheletro del sito, cui verrà poi sviluppata e applicata la parte Back-end, perdendo potenzialmente di complessità.

Personalmente, tra i due approcci consiglierei una via di mezzo, che riguarda il fare inizialmente uno scheletro del sito con *Header e Footer* in cui poi si procede in parallelo per ciascuna funzionalità e si riadatta PHP e HTML in base alle necessità, sviluppando ciascuna feature in maniera **modulare** (ossia, come funzione o metodo).

Con il secondo approccio, potrebbe capitare di trovare una **ripetizione di codice PHP** usato per implementare una particolare funzionalità. Analogamente, col primo approccio potrebbe capitare nel caso di **HTML e classi CSS**. Per ovviare a questo problema è opportuno comunicare tra i vari membri del gruppo e parlarne sempre prima di integrare qualsiasi cosa, così da mantenere codice pulito ed estensibile.

Dall'esperienza si è visto che *chi fa una parte in Front-end, poi sarebbe opportuno la completasse anche in Back-end* per **mantenere coerenza nel lavoro**. Chiaramente anche un approccio del tipo "*io faccio solo Back-end, tu fai solo Front-end*" può andare bene, ma l'importante è che ci sia coerenza nel singolo modulo o nella parte di codifica di un singolo linguaggio.

4.1.3 Iniziamo dalla Base (di dati)

Una volta trovato il tema del sito e scelto l'approccio di sviluppo, si può procedere ad una fase più pratica. Il punto di partenza su cui si consiglia di iniziare è sicuramente il **database**.

La base di dati dovrà essere uguale per tutti, pertanto è fondamentale configurarla bene affinché tutti possano avere lo stesso database nel proprio ambiente di lavoro, così da usare e testare il sito in maniera completa.

Qui è sufficiente rifarsi alle proprie conoscenze del progetto di Database. *Sconsiglio di integrare una parte relazionale con chiave esterne esplicitate, triggers o funzioni*, al fine di evitare problemi. Da come è stato detto a lezione, è sufficiente che la base di dati faccia il suo dovere nella maniera più

semplice possibile.

4.1.4 Model View Cosa?

Per l'organizzazione dei file e del lavoro, seppur non sia richiesto, è possibile adottare l'approccio *Model View Controller (MVC)*.

Personalmente lo consiglio se tutti i membri del gruppo lo hanno già adottato in precedenza e se sanno come usarlo, altrimenti si può far fede ad un approccio più diretto, con meno file, ma con più righe di codice sullo stesso file ed eventualmente con più linguaggi mischiati nello stesso file.

Si riporta come esempio il *pattern model-view* che è il più semplice e veloce da implementare, nonché da gestire.

Con questo pattern, i file che vengono acceduti a modi `http://miosito.it/index.php` rappresenteranno i **modelli**. All'interno di un modello, ci dovrà essere esclusivamente la parte *Back-end* che si occuperà del reperimento, inserimento, modifica e/o cancellazione dei dati.

I modelli dovranno richiamare (o, per meglio dire, includere) i file di **vista** (o view) in cui si concentrerà la parte *Front-end*, principalmente HTML e integrazione di variabili PHP, ove necessario.

Nelle sezioni successive verrà illustrato qualche esempio pratico, qualora si decida di scrivere secondo questo pattern.

4.1.5 Funzioni od Oggetti?

Se siete tutti esperti di PHP potete provare a buttarvi sulla programmazione ad oggetti in PHP. Di contro, non lo consiglierei pienamente per chi non ha mai usato PHP prima d'ora visto e considerato che richiede più tempo di sviluppo e di test, nonostante l'organizzazione sia decisamente più pulita e potenzialmente estensibile. Inoltre, ai fini del progetto il PHP vale relativamente pochi punti, quindi anche un approccio semplificato va più che bene.

Nota: se si sceglie di fare tutto a funzioni, è possibile fare al più una classe a oggetti (come la classe database ad esempio). Chiedere comunque ai professori per sicurezza, dal momento che vogliono o tutto a oggetti o tutto a funzioni.

4.1.6 La cartella di progetto

La cartella di progetto avrà al suo interno diverse sottocartelle che conterranno i file per il progetto.

Di seguito illustro un possibile modo in cui si possono organizzare i files del progetto in modo tale che tutto sia ordinato e le risorse facilmente importabili.

```
_database
  db.sql

_docs/
  documentazione.md

includes/
  class/
    db.class.php
  functions/
    funz1.php
    funz2.php
    ...
  config.php
  variables.php
  resources.php

styles/
  css/
    general.css
    ...
  js/
    core.js
    ...
  img/
    immagine1.jpg
    ...
  resource.css
```

```
29
30 views/
31   template/
32     header.php
33     footer.php
34   showIndex.php
35   showAbout.php
36   ...
37
38 index.php
39 about.php
40 contacts.php
41 ...
```

Snippet 1: Esempio di struttura della cartella di progetto

In questo esempio, le prime due cartelle non saranno direttamente utili al progetto e serviranno solamente per salvarsi su Github documentazione e database:

- La cartella *_database* contiene uno o più file .sql aggiornati con struttura e con eventualmente dati per il popolamento del database.
- La cartella *_docs/* contiene i documenti scritti `markdown` quali note scritte di progettazione.

Si passa quindi al vero core del sito web, in cui si organizzano tutte le risorse disponibili:

- La cartella *includes/* contiene tutti i file di funzioni, configurazione del sito e variabili di ambiente. Il file `resources.php` sarà un semplice file che richiama in ordine con dei `require_once 'file.php'` o `include('file.php')` tutti i file php presenti nella cartella, partendo prima dal `config.php e variables.php` e poi importando tutti i file di funzione e/o di classe presenti nelle sottocartelle.
  In questo modo, all'inizio di ogni file php presente nella root, si potrà richiamare direttamente solo il file `resources.php`, mantenendo il file leggero e pulito.

- La cartella *styles/* è molto auto-esplicativa e simile alla cartella `includes/`. Anche qui, per comodità, `resources.css` funge da file per importare tutti i file CSS presenti in `css/`. In questo modo, nel `<head>` sarà sufficiente richiamare tramite il `<link rel="..">` quel singolo file CSS.
- La cartella *views/* raccoglie tutti i file in cui è stato scritto codice HTML che verrà servito e adattato dinamicamente con la pagina richiesta nella root directory.
- La cartella *views/template/*, infine, racchiude quelle parti di template che si ripetono sempre nel sito, come la testata e il piè di pagina, e che dovranno essere richiamati singolarmente in ciascun file php presente nella root, dopo il `resources.php`.

Concludendo, i file nella root che sono quelli che verranno acceduti a modi `http://localhost/index.php`. Per inciso, nella terminologia *MVC*, questi file potrebbero rappresentare i controller. I modelli, invece, andrebbero inseriti in una apposita cartella `models/`.

Consiglio: andatevi a vedere la differenza tra `require`, `require_once`, `includes`. In generale, conviene usare `require_once`.

Consiglio 2: i file `config.php e variables.php` sono opzionali ma molto consigliati in quanto contengono delle variabili di configurazione del sito.

- Nel `config.php` si possono assegnare variabili con valori costanti (`define`) o di default (es: nome del sito, dati di connessione al database in un array, versione del sito, ecc.).
- Nel `variables.php` si consiglia invece di mettere le variabili di ambiente che verranno usate più spesso nel sito.

Ad esempio, variabili come `id, error, action`, in genere si ripetono spesso tra i modelli, e la loro funzione è, rispettivamente, di ritornare un ID, un messaggio di errore e/o una azione tramite GET. In questo file, dunque, si potrebbe inserire

```
if(isset($_GET['id'])) $id = (int)$_GET['id']; else $id = 0;
```

così che la variabile `$id` non si debba dichiarare ogni volta in ogni singolo modello che la usa, conformando tutti gli sviluppatori del sito a questo standard comune e rendendo più leggibile il codice.

4.2 Sviluppo del sito

In questa fase, si comincia ad applicare tutto ciò di cui si è discusso e si comincia a scrivere codice, testare e validare ciò che è stato fatto.

4.2.1 Apprendere i linguaggi HTML, CSS, JS e PHP

In generale, si consiglia quanto più possibile da attingere alle slides dei professori e agli incontri di laboratorio.

In generale, però, per apprendere a fondo questi linguaggi è opportuno fare molte ricerche su internet e soprattutto avere fonti attendibili. Di seguito si fa riferimento ad alcuni siti che sono molto utili sia ai fini di apprendimento, sia per reperire risorse da implementare nel sito:

- W3School è il sito più completo e affidabile che ci sia. Fornisce esempi, documentazione e guide per tutti e cinque i linguaggi trattati per TecWeb.
- php.net è il sito ufficiale di PHP con una documentazione completa. Per ogni funzione, classe o argomento trattato, al di sotto di esso si possono trovare i commenti degli utenti con degli esempi scritti di applicazione dell'argomento selezionato.
- StackOverflow permette di fare domande e ancor prima di trovare risposte fatte nel corso di decenni in ambito di siti web e molto altro. Ovviamente conviene cercare il problema sempre in inglese, e gran parte delle soluzioni ai vostri problemi le troverete lì.
- ColorZilla con questo piccolo e potente tool si può creare del codice CSS per ottenere gradienti e sfumature di colore.
- CSS-Tricks raccoglie buoni consigli per usare al meglio il CSS e per dare un tocco di moderno al sito.
- W3.org raccoglie tutta la documentazione sul HTML che è bene controllare prima di fare qualsiasi tipo di implementazione e, possibilmente, da confrontare con W3school.

Per il resto, si consiglia sempre di fare ricerche Google in *inglese* correlate a ciò che si vuole fare.

4.2.2 Quale versione di HTML?

La scelta di quale versione di HTML usare è senza ombra di dubbio la più dibattuta. Nelle specifiche dell'anno corrente (2018-2019) è possibile usare HTML5 con XML (che tradotto sarebbe xHTML 5), lo standard più "moderno" fino ad ora accettato, specie per la retrocompatibilità con IE9. Usarlo potrebbe rivelarsi un rischio, soprattutto per la valutazione finale, poiché si deve stare molto attenti nella parte di validazione di conformità del codice allo standard ed è facile trovare documentazioni sbagliate in giro per il web che non si adattano alla versione richiesta dai professori.

Personalmente consiglierei comunque la versione più recente di HTML usabile nel progetto, facendo attenzione però a possibili tag nuovi non compatibili tra browser e alla sintassi che potrebbe presentare piccole variazioni rispetto alle versioni precedenti.

Nota: nella fase di relazione, sarà opportuno motivare un minimo questa scelta, argomentando perché si è scelta una versione al posto di un'altra.

4.2.3 Consigli per HTML

HTML sarà la parte più semplice da scrivere, ma non è da sottovalutare dal momento che molti punti del progetto si concentrano su di esso. La cosa più importante da tenere a mente è che la *struttura DEVE essere il più semplice possibile* e talvolta, quando si parla di stile, riuscire a ridurre il numero di classi da creare in CSS per tenere il codice pulito sarebbe l'ideale.

Analogamente, è opportuno cercare di *usare sempre lo stesso HTML per gli elementi* sia nella versione Desktop, che nella versione Mobile, altrimenti si tende ad appesantire inutilmente la pagina e a dover riscrivere più volte lo stesso codice.

In generale, vi lascio questi tips:

- Fare attenzione ai `<div>` annidati e alla chiusura dei tag (di cui fortunatamente il compilatore segnala errore).
- Attenzione alla retrocompatibilità dei nuovi tag in HTML5 che non tutti sono compatibili sui browser più vecchi (IE9).
- Alcuni attributi, sempre in HTML5, sono segnati come errore dai validatori poichè obsoleti. Tuttavia, alcuni di essi, se necessari, si possono usare e giustificare in relazione. Ad esempio, `longdesc="..."`, che può essere utile per le immagini, è consigliato dagli stessi professori nelle slide e, sebbene segnato come errore dal validatore, comunque permesso.
- La struttura portante (o template) del sito deve essere la prima cosa da fare e da perfezionare fin da subito, così da poter proseguire agilmente nel lavoro e adattare bene la parte del BODY (che sarà quella più dinamica) alle parti di HEADER e FOOTER (che sono più statiche, in genere).
- Fare molta attenzione alla sintassi in base al tipo di HTML che usate (specie xHTML), perché il semplice fatto di scrivere qualche tag / attributo / valore in MAIUSCOLO al posto del minuscolo può portarvi alla perdita di punti.

Una piccola nota anche sulla disposizione delle informazioni in un sito web, che in genere si studia a *Web Information Management*. Fare attenzione quando si vogliono mettere tante immagini e soprattutto alla quantità di testo.
Less is more, quindi poche immagini ma messe nei punti giusti sarebbe il giusto mezzo. Cercate poi di trovare un buon rapporto tra la quantità di parole e la grandezza del testo. Piccole e brevi frasi, ma con significato (!= ad effetto), vincono sui wall of text. Ai professori non dovrebbe cambiare molto, ma a livello psicologico e a livello di impressione generale del sito può fare una grossa differenza e magari portarvi qualche punto in più.

4.2.4 Consigli per il CSS

Il CSS è - in genere - la parte più ardua da fare, anche per chi se ne intende molto di programmazione, poiché ci vuole molta pratica per poterlo capire a fondo e comprenderne al volo i problemi e le soluzioni attuabili.

Per padroneggiare bene il CSS si ritiene opportuno vedere tutte le tecniche spiegate in classe e riportate nelle slide; in particolare, capire bene `margin, padding` e il funzionamento di `position e z-index`.

Dopodiché, la maggior parte delle volte la stesura del CSS dovrà avvenire per *WYSIWYG*, utilizzando *Ispeziona elemento* del browser per testare in tempo reale le modifiche ad un elemento. Una volta testati gli attributi, questi si possono riportare direttamente nel file CSS. A tal proposito si consiglia di usare browser quali Firefox e Chrome, che sono ottimi per questo genere di cose, senza contare l'ampia disponibilità di plugin disponibili.

Consiglio: Fare attenzione alla compatibilità cross-browser e alla retrocompatibilità con IE9. Molti attributi sono troppo nuovi per i browser vecchi, oppure alcuni sono fatti solamente per uno specifico browser (in genere questi sono segnati con prefissi quali `moz- o webkit-`). Si consiglia quindi di controllare la retrocompatibilità su W3School di ciascun attributo.

Consiglio 2: L'organizzazione dei file CSS e sulla quantità di files da produrre è a libera scelta. Idealmente sarebbe bene avere pochi files CSS che raccolgono caratteristiche per tutte le pagine, ma anche fare un CSS per ogni pagina e poi includerlo in un simil `resources.css` può rivelarsi una buona soluzione.

Consiglio 3: Non si consiglia di usare Bootstrap a meno che non si sappia molto bene come implementarlo, a detta dei professori. Se si vuole usare, si deve comunque avere una buona produzione di CSS proprio per il template del sito e per altre funzioni come ad esempio il file CSS per la stampa.

Consiglio 4: Usate le media query messe a disposizione da CSS 3 direttamente nel file css ed evitate i media su HTML nel `<link rel="...">`.

Nota: nei browser sopracitati c'è anche la versione scura di `Ispeziona elemento`.

4.2.5 Consigli su Javascript

Javascript da quest'anno pesa relativamente poco sui punti finali, ma, al contrario di quello che pensano molti, scriverne tanto usando magari particolari librerie come `jQuery`, non serve praticamente a **niente**.

I professori lasciano intendere che la parte di JS debba essere usata principalmente per fare un **controllo nella parte input degli utenti** (principalmente le pagine con form cui hanno accesso i visitatori).

A tal proposito, va sviluppata una parte JS che controlli in un form se, dati certi valori negli input, questi siano validi e se si possa procedere a mandare le informazioni al server (il quale, a sua volta, avrà anche lato Back-end medesimi controlli, qual'ora JS sia disattivato).
Purtroppo, sebbene i `required` o `type="email"` di HTML5 siano presenti, questi non sono sufficienti come controllo Front-end.

Ovviamente, se si volesse implementare cose come Slider o caricamenti asicroni con `xmlHttpRequest` questi sarebbero comunque ben accetti, ma peserebbero comunque poco rispetto a quello che viene richiesto.

Consiglio: Non perdere troppo tempo per fare animazioni o cose simili nel progetto, perché conterebbero poco niente. Concentrarsi sui controlli e sulla sicurezza dei valori di input è meglio.

Consiglio 2: Nel caso in cui ci sia uno stile o delle parti di sito che si rompono in assenza di JS, potete usare `<noscript>` in head per aggiungere un file css apposito (tipo `nojs.css` con stili alternativi e/o di fix) oppure direttamente nell'html per fare un elemento di replace che compare solo con JS disattivato.

4.2.6 Consigli su PHP

Il PHP è un linguaggio molto semplice da imparare, non tipizzato e abbastanza versatile. Per apprenderlo pienamente conviene leggersi molti esempi online in base a ciò che bisogna fare e il mio consiglio è di impararsi i 3 concetti di fondo che vengono spiegati anche a lezioni:

- Usare e printare le variabili, soprattutto quelle globali per i form, `$_GET[] e $_POST[]`
- Il funzionamento delle sessioni, vedendosi `session_start()`, `$_SESSION[]`, `session_unset()`, `session_destroy()`
- La connessione e gestione dei dati del database con `mysqli`

Sconsiglio vivamente di provare a implementare `$_COOKIE` dal momento che è abbastanza complicato e sicuramente non richiesto per nulla nel progetto.

Vi lascio di seguito una lista di funzioni utili che ritroverete spesso:

```
error_reporting() // per visualizzare o meno certi errori come warning o notice
str_replace() // per eseguire replace di una sottostringa
explode() // divide in un array sottostringhe delimitate da uno o piu' caratteri presenti in una stringa
preg_match(), preg_replace() // utilizza le espressioni regolari per trovare pattern in una stringa
hash() // per hashare le password, si consiglia sha256 almeno
empty() // se la variabile o l'array e' vuoto
isset() // se la variabile esiste e con valore
strlen() // lunghezza della stringa
count() // conta gli elementi di un array
```

Snippet 2: Funzioni PHP utili

Per il resto, gli operatori sono sempre gli stessi degli altri linguaggi di programmazione, con l'unica nota della presenza dell'operatore `===` che fa match anche sul tipo oltre che sul valore. A tal proposito sono presenti anche i cast espliciti, come ad esempio `$x = (int)$_GET['id']` oppure per gli array `$x = (object) array("roba" => "x", "obar" => 2); echo $x->obar;`

4.2.7 Esempio di sviluppo del sito

Nella gerarchia riportata nello scorso capitolo sulla progettazione, si è menzionata la cartella `template/` che dovrebbe contenere `header.php` e `footer.php`.

Nel caso di *Model-View*, queste parti verranno sempre richiamate insieme ad una o più view che comporranno nel modello il file HTML finale. E' chiaro che i meta tag dovranno essere aggiornati per ogni pagina, quindi per ciascun modello è sufficiente usare delle variabili php nominate sempre allo stesso modo e riportate con degli `echo` o `<?=$var;?> (short echo)` nei file di template.

Di seguito si lascia un esempio molto semplice per far capire intuitivamente come verrebbe sviluppato il sito a livello di organizzazione e composizione dei file.

```
<!DOCTYPE html>
<html>
  <head>
    <title><?=$page_title;?></title>
    <meta name="description" content="<?=$page_description;?>">
    <link rel="stylesheet" type="text/css" href="styles/resource
    .css">
     ...
  </head>
  <body>
    <header>
      <ul class="menu">
        <li> Link 1 </li>
         ...
      </ul>
       ...
    </header>
     ...
    <div class="body-container"> <!-- apro body container -->

    <!-- Fine heading template -->
```

Snippet 3: Esempio file views/template/header.php

```
1
2     <p> Questo e' un paragrafo di esempio nel corpo della pagina
      </p>
3     <p> La data di oggi e': <?=$data;?> </p>
4
```

Snippet 4: Esempio file views/showIndex.php

```
1
2     </div><!-- chiudo body container -->
3
4     <footer>
5       ...
6     </footer>
7   </body>
8 </html>
```

Snippet 5: Esempio file views/template/footer.php

```
1 <?php
2   // Risorse e configurazione PHP
3
4   require_once 'includes/resources.php';
5
6
7   // Configurazione della pagina
8
9   $page_title = "Homepage";
10  $page_description = "Questa e' la homepage";
11
12  // Caricamento dati
13
14  $data = date("d/m/Y");
15
16  ...
17
18  // Composizione della pagina
19
20  require_once 'views/template/header.php';
21
22  require_once 'views/template/showIndex.php';
23
24  require_once 'views/template/footer.php';
25
26 ?>
```

Snippet 6: Esempio file index.php

```php
<?php

// File di risorse PHP

require_once 'config.php';
require_once 'variables.php';
require_once 'functions/funz1.php';
require_once 'functions/funz2.php';
...
```

Snippet 7: Esempio file includes/resources.php

```
// File di risorse CSS

@import url("css/general.css");
@import url("css/layout.css");
@import url("css/file2.css");

...
```

Snippet 8: Esempio file styles/resources.php

C'è da tenere in considerazione che le scelte progettuali poi possono essere riadattate nella maniera che uno ritiene più comoda. Ad esempio, si potrebbe fare un file apposito per un *menù di amministrazione* che viene richiamato con un `if(isAdmin()) require_once 'views/adminMenu.php'`, se nel sito si è loggati come admin.

Le possibilità sono molte, basta avere un po' di creatività e trovare il giusto approccio che possa andare bene per tutti i membri del gruppo.

4.3 Validazione e Accessibilità

In questa sezione si tratteranno i consigli per quanto riguarda la validazione e l'accessibilità. Si preannuncia che molte delle cose non citate saranno facilmente reperibili dalle slide della *prof.ssa Gaggi*. Dal momento che **è richiesto un sito online per eseguire la maggior parte delle validazioni**, si consiglia di avere online il progetto come precedentemente suggerito nella parte di progettazione.

4.3.1 Test di conformità del sito

Dopo aver eseguito gran parte dello sviluppo del sito, si può passare ad una fase di test veloce delle funzionalità del sito. In questo caso, si consiglia di far provare il sito anche a persone esterne al progetto per capire se ci siano punti critici da eventualmente correggere.

E' di estrema importanza controllare che la parte adibita ai visitatori, soprattutto in caso di input form sia SOLIDA e con tutti i controlli del caso. Vi riporto gli **errori più frequenti nei form** che possono capitare:

- Spazi vuoti dentro un input passano il controllo
- Email a modi *a@a.it* è valida
- Input vuoto richiesto è valido
- Numeri dentro un campo come "Nome e Cognome" è valido
- Input richiesto diverso da quello aspettato ma comunque valido

Per sistemare questi errori è opportuno eseguire sia controlli con JS che con PHP, talvolta identici, così da proteggersi da eventuali malintenzionati e, nella maggior parte dei casi, per ridurre il carico server affidandosi ai controlli JS lato client. Da non sottovalutare i controlli PHP in caso di JS disattivato.

A scalare, le parti che richiedono più controlli sono (in ordine decrescente):

1. Zona visitatori
2. Zona utenti
3. Zona admin

Fortunatamente, si trovano molti script in giro per il web che aiutano a garantire la conformità di un input atteso. L'importante è controllare tutti i risultati possibili, senza tralasciare nulla.

4.3.2 Validazione del codice

La parte di validazione del codice è molto criticata nella valutazione, specie qualora compaiano errori inaspettati. Al fine di evitare questo genere di problematica è bene affidarsi ai validatori che troviamo online e che sono stati consigliati a lezione nelle slides.

In particolare si consigliano tutti i validatori trattati a lezione e in particolare quelli ufficiali:

- Validatore HTML W3.org,
- Validatore CSS W3.org
- Total Validator (è presente xHTML5)

Per il modello *MV / MVC*, al fine di validare il codice è sufficiente copiarsi la sorgente della pagina generata dal *modello / controller PHP* ed eseguire quindi la validazione.

Il *JS* non è necessario validarlo, piuttosto è importante che i browser non emettano un errore quando viene eseguito. Per testare il JS, dunque, è sufficiente usare la console di `Ispeziona Elemento` nel browser.

Allo stesso modo *PHP* non è necessario validarlo, ma è sufficiente controllare a runtime che non ci siano errori e, possibilmente, anche warnings (`error_reporting(0)`: *well yes, but actually no*).

4.3.3 Scremare gli errori di validazione

Come detto in precedenza, alcuni errori di validazione possono essere permessi se concordati con i professori (come il `longdesc` per HTML5). Ovviamente al 95% tutti gli errori dovranno essere corretti o la valutazione si ridurrà parecchio.

Fare attenzione agli attributi del genere `disabled="disabled"` che si ripetono, e che da HTML5 non è necessario ripeterli (basta solo `disabled`, come riportano tutte le documentazioni), ma al contrario in xHTML5 è obbligatorio ripeterli.

Infine, una volta concluso l'iter di validazione, non dimenticarsi di inserire i banner del codice validato per HTML, CSS e accessibilità.

4.3.4 Punti importanti sull'accessibilità

La parte di accessibilità della *prof.ssa Gaggi* riporta alla perfezione nelle sue slide tutte le regole da usare al fine di non avere problemi di navigazione con persone ipovedenti, garantendo tutte le funzionalità che il sito offre.
Di seguito riporto una lista delle cose più importanti da ricordarsi di inserire od omettere nel sito e da menzionare per sicurezza nella relazione, se opportuno:

Torna Su: un link o un pulsante, adibito soprattutto per il mobile, per tornare su nella pagina (`<a href="#">Torna su</a>`).

Val al contenuto: link da porre in maniera invisibile solo per gli screen reader subito dopo il tag `<body>` con riferimento a dopo la testata / menù e inizio del contenuto.

Alt nelle immagini: gli alt nelle immagini vanno riportati per qualunque immagine (sebbene per immagini di contorno e non utili a fine informativo non sia necessario; confrontarsi comunque coi professori per sicurezza di questo punto)

Longdesc nelle immagini: il longdesc va riportato solamente nel caso in cui ci sia una immagine complessa e importante per il contenuto del

sito che non ha descrizione esplicita (o se la ha, conviene metterci l'ID `#descImmagine1` al paragrafo che contiene la descrizione).

Tabindex: non è necessario se tutta la struttura dei form di input funziona in automatico, va comunque giustificato il mancato utilizzo.

AccessKey: idem, da non usare; inoltre, alcuni tasti potrebbero andare in conflitto con hotkey native dei browser, dunque meglio evitare.

xml:lang e lang: sono importanti e vanno aggiunti per ogni singola parola o frase diversa dalla lingua riportata per il documento
`<html lang="it" xml:lang="it">`

Abbreviazioni e Acronimi: ricordarsi di usare il tag apposito in caso di acronimi o parole abbreviate per farle leggere in modo completo e corretto agli screen reader.

Sitemap: non è necessaria, ma va giustificata la presenza o meno in relazione. Se venisse messa, si ritiene che sia facile perdersi nel sito, ma questo permetterebbe di capire se effettivamente per gli utenti la navigazione è sufficiente o meno. Se non la si mette allora si ritiene che il sito sia già facilmente navigabile.

Breadcrumb: questo è opportuno inserirlo così da dire all'utente dove si trova correntemente e anche per agevolare la navigazione.

Link circolari: vanno evitati completamente. L'ideale è rimuovere completamente il tag `<a href="...">` e metterci testo semplice oppure uno `<span>`.

Cache manifest: non necessario, troppo complicato da implementare.

File CSS minimizzati: si consiglia di minimizzare ogni file CSS, così da rimuovere spazi e commenti superflui. Per farlo, vi sono dei tool automatici online e gratuiti detti *css minifier*. Ovviamente, alla consegna DEVE essere allegata anche la versione normale dei file CSS nel progetto.

Tabelle: le tabelle vanno rese accessibili come visto in laboratorio. E' opportuno far combaciare testata con cella tramite `id, scope, headers` ed elementi affini.

Cookie consent: si può inserire come no, l'importante è giustificare che se non viene inserito è perché si tratta di un progetto accademico, ma in caso di un prodotto rilasciato in produzione andrebbe messo.

4.3.5 Strumenti di validazione del grado di accessibilità

Per requisiti di progetto, è richiesto il grado AA minimo di accessibilità. Ovviamente, a detta della professoressa, più è alto il grado di accessibilità, meglio è. Renderlo alto, fortunatamente non è complicato, ma richiede parecchi accorgimenti che verranno detti dal validatore.

Di seguito si elencano i principali validatori e strumenti usati per l'analisi di accessibilità, con tanto di prove di colori e contrasto:

- AChecker: strumento di validazione grado di accessibilità (usare WCAG 2.1)
- Wave: altro strumento di validazione grado di accessibilità
- TopTal: strumento per il colortest, utile nella parte di relazione di accessibilità.

Anche in questo caso, ricordarsi di riportare il relativo banner, una volta convalidato il sito.

4.4 Relazione

La relazione di progetto riassume brevemente tutto quello che è stato fatto, dalla progettazione allo sviluppo, fino ai test e alla validazione.

In generale non c'è molto da dire a tal riguardo se non di come predisporre i vari argomenti, basandosi molto anche su quello che hanno fatto i gruppi negli anni precedenti.

1. Frontpage (con le intestazioni richieste nelle specifiche di progetto)
2. Introduzione (Abstract)
3. Analisi dei requisiti (target di utenza, obbiettivi, requisiti di funzionamento, ..)
4. Organizzazione del lavoro e progettazione (approcci utilizzati, gestione dei files ..)
5. Implementazioni lato front-end (sezioni implementate, funzionalità per visitatori, utenti e admin, sicurezza degli input con javascript ..)
6. Accessibilità e usabilità (studio dei colortest e implementazioni effettuate)
7. Implementazioni lato back-end (funzioni di aggiunta, edit e cancellazione, descrizione sistema utente e utilizzo, database ..)
8. Motivazione delle scelte implementative e parti critiche del progetto*
9. Note conclusive
10. (Opzionale) Implementazioni future

In generale questa può essere una possibile scaletta degli argomenti. Poi si possono comunque disporre gli argomenti in modo migliore, ma una presentazione di questo calibro dovrebbe essere ben accetta ai professori, in quanto completa.

*** Qualunque cosa che possa rappresentare una criticità va riportata nella relazione, al fine di attenuare eventualmente la perdita di punti.**

Concludendo, si consiglia di stare tra le 15 e le 25 pagine e di ricontrollare quanto è stato scritto affinché sia coerente nei vari punti della relazione. Si consiglia di usare Google Drive con una cartella condivisa, se si intende scrivere la relazione in più persone, dividendosi ciascun argomento da trattare.

Presentare il PDF in LaTeX è consigliato per i più smanettoni e per essere più ordinati, ma non obbligatorio.

5 Conclusione

5.1 Note conclusive

Giungiamo quindi alla conclusione di questa guida. La maggior parte delle informazioni qui riportate sono frutto di una esperienza di progetto, nonché di esperienza personale. Mi scuso per eventuali errori grammaticali, ma alla fine è stato scritto tutto molto velocemente e di getto, riordinando per quanto possibile le informazioni nella maniera più comoda.

Bisogna dire che il **lavoro da fare per questo progetto è molto** (visto che si estende per almeno 3-4 mesi), però il fatto di averlo concluso e portato a termine sarà positivo dal momento che, a prescindere dal voto che si prenderà, è comunque una nota aggiuntiva nel proprio curriculum. Purtroppo il voto non corrisponderà il più delle volte alle proprie aspettive, nonostante il duro lavoro. Questo perché il sito perfetto è utopia e lavorando in 4 persone è facile fare errori o sbagliare scelta implementativa.

Per questo vi consiglio di trarre quanto più possibile da questa guida, *prendendo comunque ogni cosa con un minimo di consapevolezza (e non come se fosse legge)*. Inoltre, soprattutto per i novizi in tecnologie web, consiglio di confrontarsi molto sia nel gruppo Telegram che con gli altri gruppi, magari di persona. Oltre a questo, domandate spesso anche ai professori che sono sempre molto disponibili e per esperienza inserite tutti e tre come destinatari della email (come suggeriscono loro stessi), così da ricevere una risposta più veloce e coerente.

Detto questo, **date frutto alla vostra creatività per questo progetto e vedrete che verrà fuori un'esperienza costruttiva.**

Buona fortuna e buon lavoro per il progetto!

Lista di Snippets