Università degli Studi di Torino, A.A. 2018-2019

Tecnologie Web (6 cfu) - MFN0634

Specifiche progetto finale

Obiettivi generali

Il **progetto finale** deve essere consegnato almeno tre giorni prima della data d'appello scelta dallo studente e deve essere accompagnato da una **relazione** che descriva la struttura e le funzionalità principali del sito che si è realizzato.

Scopo del presente documento è quello di fornire le specifiche sia della relazione di progetto che del sito di cui si chiede la realizzazione.

Il **tema del sito è libero ed a scelta dello studente.** Ciononostante, il sito dovrà essere strutturato in modo tale da seguire le specifiche indicate. Parte importante della realizzazione del sito è la relazione di accompagnamento che descrive tecnicamente il modo in cui sono state implementate le varie funzionalità, la struttura del back-end, quella del front-end ed il modo in cui è stata gestita la comunicazione tra i vari livelli dell'architettura.

Lo studente potrà utilizzare tutte le **tecnologie viste durante il corso:** HTML(5), CSS(3), PHP7.2 (sessioni, connessione a db, Object Oriented), JavaScript (Eventi JS, DOM), librerie JS (Prototype/Scriptaculous, JQuery), Ajax, scambio dati (XML o JSON), servizi web e cosí via. I problemi che sono stati analizzati e per i quali sono state discusse le soluzioni (usabilità, separazione presentazione/contenuto/comportamento, sicurezza del web, etc.) dovranno essere affrontati: scopo dell'elaborato finale è quello di consentire la connessione di tutti gli argomenti discussi con lo scopo di realizzare un sito web professionale.

Saranno considerati ammissibili anche strumenti che non abbiamo avuto modo di approfondire durante il corso, quali ad esempio: Bootstrap, pattern MVC, Angular.js, etc. Si consideri che comunque queste tecnologie saranno valutate solamente in quanto *opzionali* e non obbligatorie.

Requisiti

Segue la descrizione di massima delle funzionalità generali che devono essere integrate nel sito finale; quindi, sarà spiegato cosa ci si aspetta dalla relazione che deve essere consegnata congiuntamente al sito.

Il sito

Come precedentemente chiarito, il tema del sito lo dovete decidere voi. Potete simulare un sito di commercio elettronico, un semplice sito di intrattenimento, un sito informativo ed interattivo, un'estensione/rivisitazione di uno dei tanti esercizi visti durante il laboratorio, etc.

La cosa importante è che il sito abbia una determinata **struttura architetturale** e che presenti alcune determinate **funzionalità**. In questo documento presenteremo i requisiti riguardo a questi aspetti.

Struttura - L'architettura del sito deve essere così strutturata:

- **front-end:** L'utente accede al sito tramite una **home-page**, che corrisponde all'index della cartella "**progetto**". Tutto il sito deve essere coerente stilisticamente a quanto mostrato nella pagina principale. A questo livello bisogna stare molto attenti a dividere quanto più possibile presentazione (CSS), contenuto (HTML) e comportamento (JS, Prototype/Scriptaculous, JQuery). È consentito l'uso di Boostrap, libreria jQuery, etc. È importante organizzare bene le dipendenze (considerare l'inclusione dei fogli di stile e delle funzioni JS tramite file esterni e non incorporati direttamente nel file html). I file, a loro volta, devono essere organizzati per cartelle (ad esempio tutte le immagini dentro una cartella **img**, i fogli di stile dentro una cartella **css**, così come il codice javascript andrebbe sotto una cartella **js**, etc.). Evitate codice ridondante: prevedete di usare file separati per parti di codice che si ripetono in pagine diverse (header, footer, menù, etc.).

- **back-end:** tutto quello che l'utente non vede direttamente sul browser è stato sviluppato e mantenuto lato server. Separate logicamente le parti che necessitano di un'intelligenza "server-side". Usate PHP e mantenete i dati con MySQL (eventualmente aiutandovi con lo strumento phpMyAdmin). È molto probabile che il numero di pagine php superino in numero quelle html: in linea di principio non avete bisogno di nessun file html, anche se qualche porzione di codice HTML di supporto (es., header, footer) in alcuni casi potrebbe non contenere codice php. Anche in questo caso organizzate molto bene le informazioni: il database deve seguire i principi dell'approccio relazionale, le **query SQL** devono essere il più possibile efficienti e ottimizzate (es., evitare molte JOIN annidate su tabelle grosse). Potete realizzare il vostro database da zero usando dati fittizi, così come potete partire da uno dei database che abbiamo usato durante il corso o da altri a vostra disposizione (in questo caso, controllate bene i diritti che avete sullo sfruttamento di quei dati). Infine, organizzate bene anche le cartelle dove andrete a memorizzare i file php: anche se non seguite alla lettera un approccio MVC (che siete liberi comunque di adottare), sarebbe bene memorizzare le funzioni e le classi php in file che abbiano nomi significativi e che siano anche organizzati in base alle loro logiche

funzionali. Ad es., i file che riguardano la gestione dei profili utente potrebbero essere immagazzinati in una cartella **users**, quelli che riguardano la gestione del carrello della spesa in una cartella **shop**, etc. - se invece seguite il pattern MVC sarebbe meglio usare i nomi delle cartelle **model**, **view**, **controller** per poi creare sottocartelle con i nomi delle classi php che definite per ognuna delle categorie MVC.
Importante (vedi punto successivo): minimizzate gli output in html! Privilegiate output XML e/o JSON.

- **comunicazione back/front-end:** prevedete di scambiare i dati tra le parti client e server del vostro sito in modo asincrono usando la tecnologia **Ajax.** Questo rende l'aggiornamento dinamico della pagina molto più efficiente. Non usate un formato di dati qualsiasi. Scegliete **XML** oppure **JSON:** questo vi farà imparare a fare le cose per bene, ma soprattutto avrete a disposizione le tante librerie (Prototype o JQuery) che vi faciliteranno la gestione del file ricevuto dal server e il successivo aggiornamento del **DOM HTML.** L'idea, pertanto, è che il vostro application server realizzato in php funzioni secondo i principi dei **web service:** i programmi PHP accedono ai dati lato server, li convertono nel formato scelto (XML e JSON) e li inviano al client sul canale aperto da AJAX. I metodi HTTP che potete usare per realizzare queste chiamate possono essere sia GET che POST. In breve: **evitare** di generare output HTML con script PHP e di abusare del pattern click-wait-refresh che spesso non è necessario.

Funzionalità - Il sito deve offrire le seguenti funzionalità:

- **login/logout:** l'utente che arriva all'home page trova soltanto una pagina che chiede l'inserimento delle opportune credenziali. Non si può usare il resto del sito se non vengono inserire i dati di un utente registrato.
Consiglio: ripassate la lezione appresa con il caso di studio associato al capitolo 14

- **registrazione:** deve essere possibile per un utente nuovo potersi registrare al sito. Non realizzate una registrazione complicata: non chiedete l'attivazione da parte di un amministratore. O la registrazione và a buon fine, oppure fallisce: gestite entrambe le situazioni, fornendo un opportuno riscontro all'utente.

- **contenuto generato dall'utente:** deve essere possibile per l'utente registrato accedere a delle pagine che permettano l'esplorazione di contenuto esistente (es. articoli da acquistare, con la possibilità di visualizzare dettagli quali prezzo, descrizione tecnica, etc.). Deve inoltre esistere la possibilità di creare/organizzare il contenuto in modo personalizzato (es. scrivere recensioni dell'articolo; creare una "wish list" di articoli che si vorrebbero acquistare e di condividerli con qualche amico; inserire gli articoli in un carrello della spesa).

Caratteristiche - Il sito deve possedere le seguenti caratteristiche:

- **usabilità:** il sito deve seguire i principi di usabilità che abbiamo visto a lezione. Ad esempio, ad ogni azione dell'utente deve essere fornito un feedback (sia in caso di successo che di fallimento), i messaggi devono essere chiari, ma non dare dettagli sulla struttura del sito o del codice (incomprensibili per gli utenti normali e pericolosi da mostrare agli utenti malintenzionati); non usate finestre di pop-up (semplicemente: non usate le finestre di pop-up); usate combinazioni di colori adeguate ed evitate sfondi

che possono infastidire la lettura; etc.
Consiglio: ripassate le lezioni apprese nel capitolo 07.

- **interazione/animazione:** prevedete una sezione del sito che mostri un'animazione interattiva all'utente. Considerate qualcosa di difficoltà paragonabile al gioco del 15 (puzzle game) che abbiamo proposto durante il corso. Questa componente deve essere comunque coerente con il resto del sito (drag&drop di oggetti da acquistare nel carrello della spesa, film da guardare in una sorta di playlist virtuale, un giochino - semplice! - al termine del completamento di un determinato questionario scientifico, etc.). Siate creativi! Ma tenete anche ben saldi i piedi per terra. Importante: questi elementi interattivi "avanzati" devono essere funzionali al sito!
  Consiglio: ripassate la lezione appresa con il compito lab06

- **sessioni:** gestite lo stato lato server usando le sessioni php. Ricordate di aprirle e chiuderle sempre al momento giusto (e nel posto giusto a livello di codice...).

- **interrogazione del database:** considerate la creazione di opportuni form che permettono un'interrogazione d'alto livello (e coerente con gli scopi del sito) di dati che sono presenti lato server. Ad esempio, potreste voler vedere gli acquisti che avete effettuato nel passato, la vostra lista della spesa attuale, la lista dei desideri, gli altri utenti con il profilo più simile al vostro, i film diretti da un determinato regista, tutte le ricette che contengono il vostro ingrediente preferito, etc.

- **validazione dati input:** i dati immessi dagli utenti devono essere sempre validati, sia lato client per migliorare l'usabilità che lato server per evitare attacchi.

- **sicurezza:** attenzione agli attacchi! Non deve essere possibile compiere attacchi XSS ed HTML/SQL Injection.

- **presentazione:** la presentazione del sito deve importa sia "bella". Senza che il design vada a scapito dell'usabilità, il sito deve essere chiaro, facile da usare e da leggere, con elementi ben distinti tra di loro, etc. Considerate l'adozione di un layout flessibile, responsive, etc., preferendo uno schema a due o tre colonne invece che a pagina piena.

Termini e scadenze: il sito deve essere caricato su GitLab2 **entro 3 (tre) giorni** prima della data d'appello. Il docente clonerà i progetti degli studenti registrati all'esame subito dopo che i termini di iscrizione all'appello saranno scaduti. Dopo quella data le modifiche fatte al codice o alla relazione **non saranno più considerate.**

Relazione di progetto

Scopo della relazione è di presentare gli aspetti progettuali e tecnici del sito che si consegna, disegnando la struttura funzionale del sito, presentando l'idea di interaction design, come è stato progettato il database, quali le componenti di front-end che consentono la dinamicità del sito, quale la logica di interazione con le funzioni server-side (es., quali i parametri da usare nel GET o POST, quale l'output restituito, sintassi XML/JSON dell'output, etc.).

Andando nello specifico, la relazione che dovete presentare deve descrivere, in cinque pagine al massimo (facendo riferimento ad un formato simile a quello del presente documento).

La prima mezza pagina (al massimo) deve contenere queste informazioni:

- **il tema del sito:** descrizione generale del contenuto e degli obiettivi del sito.

- **le sezioni principali:** considerate una suddivisione in macro-sezioni, ognuna delle quali può corrisponde ad una voce del menù principale. Ogni sezione deve avere uno scopo e può essere associata ad una o più funzionalità descritte nella sezione precedente

La parte rimanente della relazione, invece, dovrà soffermarsi sui seguenti aspetti:

funzionalità:

Facendo riferimento a **login/logout**, **registrazione** e **gestione del contenuto generato dall'utente**, la relazione deve esplicitare come queste funzioni sono state implementate nel progetto specifico.

caratteristiche:

La relazione deve descrivere in modo concreto come e dove sono state curate le caratteristiche richieste, quali **usabilità**, **interazione/animazione**, **sessioni**, **interrogazione del db**, **validazione dei dati di input**, **sicurezza**, **presentazione.**

front-end:

Dovete descrivere come sono stati trattate le seguenti problematiche a livello di front-end (non in generale, ma nello specifico del vostro progetto):

- separazione presentazione/contenuto/comportamento (stile unobtrusive)

- soluzioni cross-platform

- organizzazione file e cartelle di progetto

- soluzioni html/css/javascript degne di nota

back-end e comunicazione front/back-end:

Dovete descrivere come sono stati trattate le seguenti problematiche a livello di back-end e di comunicazione tra front/back-end (non in generale, ma nello specifico del vostro progetto):

- architettura generale classi/funzioni php

- schema del db

- descrizione delle funzioni remote (come se fosse una API)

Ad esempio, se realizzate una funzione user.php che serve a verificare se un dato nome utente esiste, potreste specificare:

- il modo in cui usare la funzione, es., metodo GET su http://localhost/User/user.php?name=pippo

- la sintassi dell'output, es., file JSON:

```
{
      "name": pippo;
      "id": 12132435;
      "reg_date": "May 1, 2016, 1:04 AM";
}
```

- condizioni di errore: come sono gestite

- funzioni di callback lato javascript/ajax

- etc.

Importante: il livello di dettaglio deve essere **tecnico**. No fuffa!

Sviluppo e valutazione

Consigli per lo sviluppo, deploy e test

Avete imparato diverse tecniche per fare **debugging.** Riassumiamole brevemente:

- verificate che il vostro ambiente **gitlab2** sia ben configurato. Se qualcosa non funziona per bene, considerate la possibilità di resettare tutto ed iniziare da capo con un nuovo ambiente prima di cominciare lo sviluppo del nuovo sito.

- se sapete usare **Eclipse** o **NetBeans** o altro ambiente di sviluppo (**Atom**, **Visual Studio Code**, etc.), avete una marcia in più. Considerate la possibilità di impararli cogliendo l'occasione di questo progetto. In particolare, saper usare i debugger integrati ed eseguire passo-passo il codice grazie ad opportuni breakpoint rende la vita del programmatore molto più semplice.

- usate i validatori ufficiali w3c per **html/css**

- usate gli **strumenti per lo sviluppatore** dei vostri browser. La console è la vostra migliore amica, ma ricordate anche anche il debugger built-in è molto utile, per non considerare la rappresentazione del box-model, la possibilità di analizzare le risorse di rete, quella di modificare proprietà css e codice html "al volo" per valutarne l'effetto, etc. Provate su browser diversi!

- per poter fare debugging del codice lato server, valutate di configurare opportunamente **XDebug** nel vostro ambiente di sviluppo e di laboratorio.

- considerate la possibilità di usare **PHPUnit** testing.

Implementazione e valutazione

Durante la correzione del progetto gli errori conteranno davvero. State attenti ai problemi grossi, ma la cosa migliore è partire da un codice pulito. Sotto sono riassunti i consigli principali che abbiamo dato durante il corso. La **valutazione del vostro progetto** dipenderà dalla **completezza** dal progetto e dalla sua **correttezza.** La correttezza è legata al modo in cui rispetterete le specifiche descritte in questa sezione.

HTML, CSS: Ricordare di usare un HTML(5) validato con W3C HTML(5) validator. Non inserite informazioni di stile nel HTML5, come fogli di stile inline o tag HTML come b o font.

Inserite tutte le informazioni di stile in uno o più file CSS. Per fare tutto correttamente, il vostro CSS deve superare la validazione con W3C CSS (jigsaw). Per esempio, se dovete usare spesso le stesse regole di stile, non definitele più volte nel foglio di stile. Se si usa lo stesso colore o tipo di font per più elementi, occorre raggruppare questi elementi in una singola regola di stile nel CSS. Scegliete di usare i selettori per evitare di definire più id e class. Limitate l'uso di posizioni fixed o absolute. Non usate le tabelle di HTML per definire il layout.

Formattate il vostro HTML e CSS in modo da renderlo più leggibile possibile. Ponete un commento nella <head> di ogni file con il vostro nome, cognome, breve descrizione del sito e il contenuto del file. Usate in modo opportuno spazi bianchi e indentazione. Per rendere la lunghezza delle righe maneggevole, non ponete più di un blocco di elementi in una linea, o vai a capo dopo massimo 100 caratteri.

Cercate di strutturare il vostro codice in un modo simile agli esempi visti in classe, facendo particolare riferimento ai diversi casi di studio che abbiamo analizzato al termine di ogni capitolo. Usate spazi bianchi ed intestazioni in modo appropriato. Non mettete più di un elemento blocco per ogni linea della vostra pagina HTML.

Non usate tabelle HTML per definire il layout, ma solo per contenere dati ed informazioni (<table> serve per organizzare il contenuto, non per organizzare la presentazione e lo stile).

Quando usate i form HTML, scegliete i controlli corretti ed i loro attributi di conseguenza.

Scegliete GET e POST per spedire i dati al server nel modo corretto.

PHP, form e DB: Il vostro codice PHP non deve generare **warning** o **errori** (possibilmente neanche **notice**), quindi controllate opportunamente i vostri **file di log** prima di consegnare il progetto **(questo è <u>importante</u>: troppo spesso capita che lo studente cada dalle nuvole quando gli si chiede di localizzare il file di log!).** Minimizzate l'uso di variabili globali (a volte sono necessarie, ma abituatevi ad usare funzioni parametrizzate), usate correttamente l'indentazione e la spaziatura, evitate linee più lunghe di cento caratteri che rendono il codice difficile da leggere. Usate il materiale prodotto durante le settimane precedenti e quello allegato ai vari capitoli del libro di testo (il materiale è reperibile su moodle).

Non filtrate a posteriori i vostri dati recuperati da DB con PHP: usate le giuste query SQL piuttosto. Se fate così, potrete trattare i vostri dati molto più semplicemente ed efficientemente. Ad esempio, è decisamente un esempio di pessimo stile di programmazione recuperare tutti i dati di una tabella dal db e poi eseguire complesse operazioni sull'array ottenuto per cercare le informazioni che corrispondono alla richiesta dell'utente. SQL è un linguaggio potente: usatelo.

Per fare tutto alla perfezione, riducete la quantità di codice PHP nel mezzo delle vostro codice HTML. Il codice ridondante può essere inserito in una funzione, da dichiarare dentro un file che può essere incluso in altri file. Ricordatevi di limitare il più possibile le istruzioni PHP print ed echo: usate piuttosto le **espressioni blocco** <?= ... ?>, così come abbiamo visto in classe.

Un altro aspetto importante è legato all'uso dei ***commenti* PHP.** Ad esempio, all'inizio di ogni file, di ogni funzione e di ogni blocco PHP, ricordatevi di mettere dei commenti descrittivi.

JS, DOM, librerie: Il (o i) file .js deve(devono) essere il più possibile privo(i) di errori (usate gli strumenti dello sviluppatore del vostro browser anche per il debugging).

Usate poche variabili globali ed evitate codice ridondante, impiegando anche opportuni parametri e la restituzione di valori in modo appropriato. Le operazioni più frequenti devono diventare funzioni, per evitare che il codice diventi troppo complicato e lungo.

Sono consentite **alcune variabili globali**, ma non troppe: si devono usare soprattutto variabili locali. Se una variabile è usata spesso, dichiaratela come "costante" definendola in lettere maiuscole (es: IN_UPPER_CASE) e usandola nel codice. Non salvate come variabili globali gli oggetti DOM, come nel caso dei valori selezionati con le funzioni $, $$ o document.getElementById.

Commentate in modo adeguato il codice JavaScript. All'inizio del file descrivere il programma e inoltre descrivete le sezioni più complicate del vostro codice.

Separate la parte di **content** (HTML), dalla **presentation** (CSS), e dal **behavior** (JS). Il codice JS dovrebbe usare stili e classi del CSS, piuttosto che settare ogni proprietà in JS.

Usate in modo corretto JavaScript in modo da non avere altro codice, né funzioni onclick o altro, incluso nel codice HTML: adottate uno stile JS "discreto" (**unobtrusive**).

Gestite gli eventi e l'attraversamento del DOM con le librerie JS che abbiamo visto, adottando la "famiglia" Prototype/Scriptaculous oppure JQuery (a vostra scelta).

Importante: il codice usato negli esercizi che viene riciclato integralmente o quasi non sarà considerto nella valutazione finale. I progetti che copiate da altri compagni di corso saranno considerati nulli. La collaborazione è benvenuta, ma i progetti finali devono essere profondamente diversi

Consegna del progetto e iscrizione all'esame

Frequentanti

Se avete frequentato il corso, avete già consegnato la url gitlab2 del vostro progetto. Non dovete fare altro: quando vi iscriverete all'esame, il docente scaricherà tutti i vostri esercizi ed il vostro progetto d'esame (con relazione). il docente scaricherà il vostro progetto e la vostra relazione **tre giorni prima** del giorno dell'appello al quale vi sarete iscritti.

Non Frequentanti

Il progetto e la relazione dovrà essere consegnato esclusivamente su gitlab2. Dovrete usare quindi una cartella di consegna su moodle dove andrete a specificare il vostro progetto (assicurandovi prima che sia pubblico e che comunque il docente ed i tutor del corso possano accedervi). Ad esempio, dovrete consegnare una url che somiglia alla seguente e dove al posto del nome del docente ci sarà il vostro username gitlab2:

https://gitlab2.educ.di.unito.it/ruffo/tweb.git

Anche in questo caso dovete iscrivervi all'appello in tempo: il docente scaricherà il vostro progetto e la vostra relazione **tre giorni prima** del giorno dell'appello al quale vi sarete iscritti.