Relazione Tecnologie Web 2014

Andrea Giacomo Baldan, Alberto De Agostini

18 Giugno 2014

- E-mail referenti
  - *E-mail: a.g.baldan@gmail.com*
  - *E-mail: miniotta@hotmail.it*
- Specifiche
  - *URL sito: http://tecnologie-web.studenti.math.unipd.it/tecweb/~utente*
  - *Nome utente:*
  - *Password:*

1 Abstract

L' attivita sportiva

2 Analisi Utenza

Essendo un sito di un centro sportivo dotato di attrezzature per la maggior parte degli sport piu' piscina, palestra e corsi per arti marziali abbiamo pensato che tutte le persone sono possibili clienti, dai bambini piccoli che possono iniziare fin da piccoli corsi di nuoto fino agli anziani che per tenersi in salute possono fare diverse attivita' sportive. Tuttavia la feature piu' significativa e su cui ci siamo piu' concentrati e' la possibilita' di prenotare un campo da gioco per alcuni sport percio' il sito e' piu' improntato sulla gestione risorse per la prenotazione che sugli altri possibili sport quali piscina o altri corsi di palestra per i quali il sito sarebbe stato piu' solo di vetrina e di informazione.

3 Ambiente di Lavoro

Per la cooperazione nello sviluppo del progetto si e' scelto di usare **github**. Il progetto e' stato quindi messo in una repository **git** in modo da poter essere condiviso da entrambi i componenti del gruppo. Tuttavia per la suddivisione dei compiti e per le parti realizzate in cooperativa abbiamo lavorato piu' giorni assieme nel laboratorio universitario in Paolotti. Per quasi la totalita' del progetto si e' lavorato su ambiente **linux** con qualche eccezione per i vari test sia per l'accessibilita' sia per layout su diverso OS. Alcuni dei programmi utilizzati per lo sviluppo sono:

- per la scrittura del codice e' stato utilizzato **emacs** e **sublime text**
- come server locale per la prova e lo sviluppo da casa **apache2**
- per la gestione e modifica delle immagini **gimp**
- per la relazione **LateX**

4 suddivisioneRuoli

Inizialmente il gruppo era composto da 3 studenti, tuttavia uno degli individui ha deciso di abbandonare cosi' abbiamo continuato e concluso il progetto in due. Ci siamo resi conto durante lo sviluppo del progetto che il carico di lavoro per due studenti era abbastanza elevato e questo ci ha procurato qualche problema. La suddivisione dei compiti e' stata decisa assieme per paralelizzare il lavoro e procedere piu' velocemente tuttavia ci siamo trovati in varie occasioni in laboratorio per decidere aspetti comuni e abbiamo lavorato molto assieme. in linea di massima la suddivisione e' stata fatta cosi':

- **Andrea Giacomo Baldan** si e' occupato di:
  - Scripting lato server (contenuti dinamici `Perl/CGI`)
  - Validazioni e controlli `Javascript`
  - Sistema `Templating` e `routes`
  - Creazione e mantenimento delle pagine di amministrazione del sito
- **Alberto de Agostini** si e' occupato di:
  - Layout (CSS)
  - Definizione della struttura dei file `XML` del database;
  - Definizione degli schemi associati ai file `XML`;
  - Test sulla validazione del codice `XHTML`;
  - Test sulla validazione del codice `CSS`;

Entrambi abbiamo contribuito alla stesura della relazione e assieme abbiamo pensato alla struttura del progetto, del sito e alle funzionalita' che esso offre.

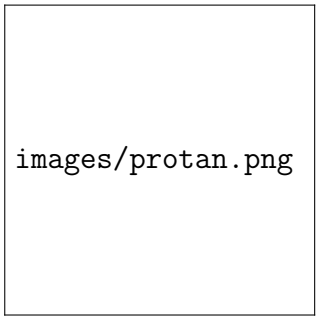


Figure 1: a) protanopia

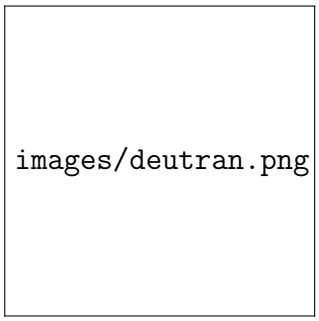


Figure 2: b) deutranopia

5 accessibilita

Abbiamo fatto diversi test in internet per testare il nostro sito per le persone daltoniche, qui sotto possiamo vedere qualche immagine filtrata tramite qualche sito per fare i test

to mare vacca

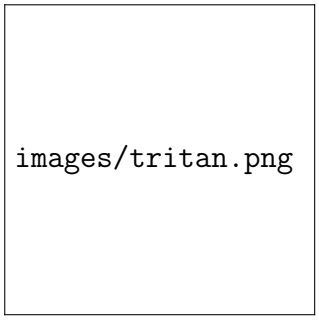


Figure 3: c) tritanopia

6 CSS

Per lo sviluppo di un sito web il ramo della presentazione risulta molto importante. Abbiamo cercato di creare un layout fluido, semplice ed intuitivo per tutti considerando che ogni persona e' un potenziale cliente come gia' mezionato nel capitolo di analizi utenza. Il sito e' diviso fondamentalmente in 6 sezioni:

- **Header** sempre nella parte superiore della pagina contiene il Titolo del centro sportivo e **Path** che risulta utile quando si accede alla pagina per la registrazione e durante la prenotazione di un campo poiche' sono 2 pagine di 'secondo livello'
- **Navigation** consiste nel menu' con tutti i link utili per la navigazione nel sito
- **Content** sempre al centro della pagina consiste nel contenuto del sito, contiene tutte le informazioni utili, la pagina di registrazione e tutti i form per la prenotazione dei campi da gioco.
- **Login** presente in ogni pagina contiene un piccolo form per effettuare l'accesso al sito o un link per andare alla registrazione per un nuovo utente.
- **News** e' un piccolo riquadro con le varie news inserite dall'amministratore
- **footer** posto sempre nella parte inferiore del sito contiene le immagini delle varie validazioni.

6.1 Differenze su diversi terminali

Abbiamo utilizzato le **mediaquery** per ottenere diversi layouts su vari terminali. qui sotto riportiamo delle immagini per mostrare alcune di queste differenze:

Su dispositivi con risoluzioni alte abbiamo deciso di usare un layout a 3 colonne mentre come possiamo notare su dispositivi con media risoluzione come tablet abbiamo deciso di spostare semplicemente la parte Login e News sotto per lasciare piu' spazio al contenuto essendo di maggiore importanza. Su dispositivi mobile abbiamo messo tutto in colonna.

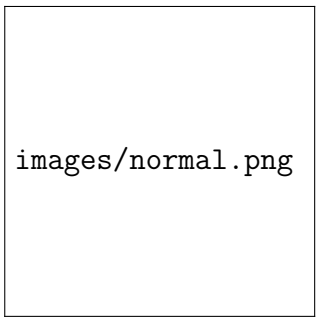


Figure 4: a) style

Figure 5: b) tablet

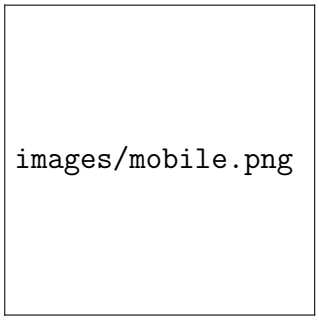
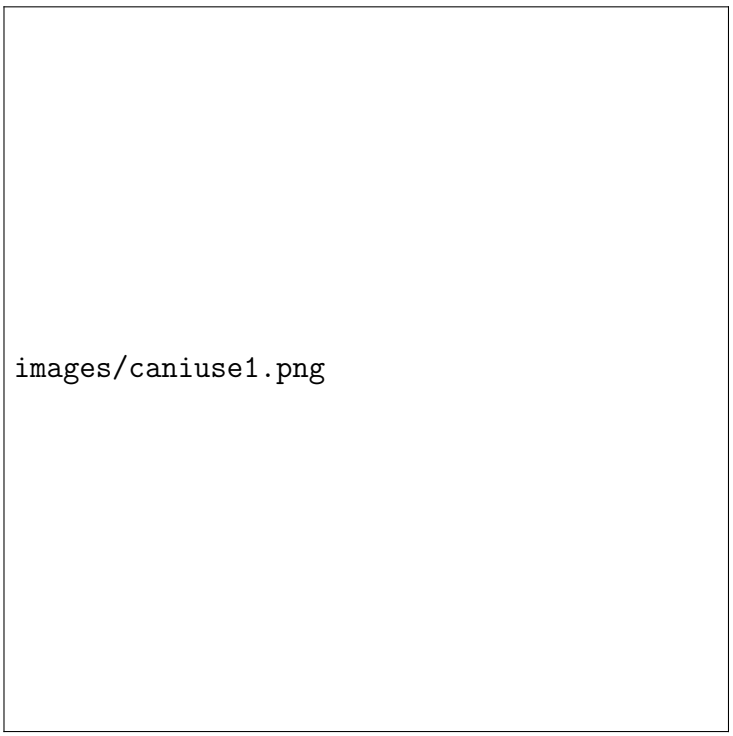


Figure 6: c) mobile

6.2 Compatibilita'

Nello sviluppo della parte di presentazioni abbiamo usato qualche regola CSS non compatibile con tutti i browsers o non con tutte le versioni. Per un maggior supporto abbiamo utilizzato diversi **vendor prefix** quali **-moz**,**-o**,**-webkit** Per la compatibilita' di queste regole abbiamo effettuato delle ricerche online e basandoci su alcuni siti come **http://caniuse.com/**. possiamo ad esempio vedere la compatibilita' di 'Transition' nei vari browser cosi':



Altre regole sono:

- rgba che ha un 90% di compatibilita' globale
- border-radius usato per arrotondare header, footer e varie immagini e' supportato da tutti i browser eccetto opera mini, da IE9 e da firefox con prefisso -moz fino alla versione 3.6 anche senza successivamente.
- box-shadow supportato da tutte le versione recenti di tutti i browser eccetto opera mini, mentre per alcune versioni vecchie servono prefissi come -webkit per chrome fino a versione 9

In ogni caso anche con versioni che non supportano queste regole css abbiamo visto che c'e' sempre un fallback elegante, il sito risulta essere meno

accattivante tuttavia il funzionamento e la dispozione generale del sito non cambiano rimanendo cosi' accettabile con ogni browser.
Per i test di compatibilita' sono stati effettuati con tutti i browser che sono presenti nel laboratorio Paolotti che sono i seguenti:

- chrome versione 34.0 nessun problema
- chromium versione 34.0 nessun problema
- firefox versione 28.0 nessun problema
- konqueror versione 4.5 con questo browser la navigazione e' pressoche' impossibile, si riscontrano molti problemi quali il non supporto a nessuna delle regole css precedentemente elencate, non rispetta il ridimensionamento delle immagini, alcuni link addirittura risultano non funzionanti. Tuttavia dopo alcune ricerche in internet abbiamo deciso deliberatamente di tralasciarlo viste le bassissime percentuali di utilizzo nel mondo.
- Opera versione 12.16 salvo qualche differenza come il colore dei placeholder tutto risulta funzionare normalmente.
- Internet Explorer
- Safari

Abbiamo anche effettuato test con Chrome disabilitando tutti gli stili o eliminando tutte le immagini e la navigazione sul sito non risultava compromessa.

7 Perl

7.1 Organizzazione

Trattandosi di un sito con una buona quantità di contenuti dinamici, è stato studiato un approccio quanto più modularizzato possibile, in modo da garantire maggior chiarezza e manutenibilità, una sorta di *pattern MVC*, dove le *view* sono rappresentate da templates (`.tmpl`) raccolti in una directory completamente separata dal codice, modelli e controller sono contenuti in 3 file contenenti inoltre le funzioni principali necessarie al popolamento dinamico del sito, si è quindi resa necessaria la suddivisione di esse in una gerarchia formata da tre moduli:

- `UTILS`: classe padre principale, raccoglie le funzioni di uso generale per il funzionamento e la popolazione delle varie pagine, caricamento ed interfaccia dei vari database XML (Model)
- `UTILS::Admin`: classe figlio di UTILS, raccoglie le funzioni strettamente necessarie al backend dell'applicazione, funzionalità di login e mantenimento delle sessioni
- `UTILS::UserService`: classe figlio di UTILS, raccoglie le funzioni necessarie al compimento delle operazioni strettamente legate all'utente (e.g CRUD delle proprie generalità), prenotazione risorse

La directory *cgi-bin* contiene solo ed esclusivamente script e moduli *perl*, mediante la variabile \$ENV{HTML_TEMPLATE_ROOT} = "../public_html/templates"; e' stato possibile mantenere i *templates* in una directory separata (*public_html/templates/*.

I principali moduli utilizzati sono:

- `CGI`: modulo per la gestione dei parametri input degli script *CGI*
- `CGI::Session`: modulo utilizzato per la gestione delle sessione e lo scambio di dati fra pagine
- `HTML::Template`: modulo per la gestione dei contenuti statici e la popolazione dei contenuti dinamici
- `XML::LibXML`: modulo per la gestione e l'interfacciamento dei file xml utilizzati per la raccolta delle informazioni e risorse
- `DateTime`: modulo utilizzato nella gestione delle prenotazioni, facilita la manipolazioni delle date e timestamp

Alcune funzioni all'interno di questi moduli sono state "privatizzate", in quanto funzioni di utilità non direttamente finalizzate all'utilizzo da parte dell'utente (e.g. creazione scheletro tabelle, calcolo e conversione dei giorni della settimana etc.). Esse sono state implementate come `subroutine` anonime assegnate a variabili, in modo che possano essere richiamate solo all' interno dei moduli in cui sono dichiarate; nella fattispecie, e' stata utilizzata una funzione protetta per la creazione delle tabelle di prenotazione dinamiche. In particolare ognuno di questi moduli fa da appoggio a rispettivi script utilizzati per effettuare le varie operazione per mezzo di dispatch tables, che consentono di risparmiare un gran numero di operazioni ridondanti e di automatizzare il piu possibile le operazioni da eseguire, aumentando inoltre la separazione tra codice e contenuto, avvicinandosi ad un approccio MVC:

- `load.cgi`: si appoggia ad `UTILS` ed è il motore di popolamento principale del sito, ogni pagina accessibile è generata e popolata da questo script, per mezzo di dispatch tables
- `admin.cgi`: si appoggia ad `UTILS::Admin`, controparte backend di `load.cgi`, ogni pagina della parte amministrativa è generata da questo script
- `process.pl`: script necessario alle basilari operazioni di modifica/popolamento risorse/pagine (CRUD)
- `user_jobs.pl`: controparte frontend di `process.pl`, tutte le operazioni che l'utente può effettuare sono gestite da questo codice

Vi sono infine `login.pl`, `login.cgi` e `logout.pl`, piccoli script atti solo all'autenticazione dell'utente, *frontend* e *backend* ed alla chiusura di eventuali sessioni aperte. `vbooked.pl` è infine lo script utilizzato per visualizzare le tabelle di prenotazione via AJAX senza il bisogno di effettuare *refresh* della pagina.

7.2 Sistema di popolamento templates

Ogni *route* richiama il *dispatcher* da `UTILS` e passa un *hash* contenente i parametri necessari al popolamento del template richiamato, che inoltre possiede lo stesso nome della *route* appunto.
Da `load.cgi` attraverso l'oggetto `$utils` e la dispatch table viene automaticamente richiamato e popolato il template corretto:

Dispatch table all'interno di `load.cgi`:

```
my %routes = (
  'home'          => \&index,
  'impianti'      => \&impianti,
  'contatti'      => \&contatti,
  'corsi'         => \&corsi,
  'prenotazioni'  => \&prenotazioni,
  'registrazione' => \&registrazione,
  'personale'     => \&personale,
  'prenota'       => \&prenota,
  'edit_personal' => \&edit_personal
);

if( grep { $page eq $_} keys %routes){
  $routes{$page}->();
}
```

Funzione `corsi` associata alla route corsi:

```
sub corsi {
  my @loop_prices = $utils->list_prices;
  my @loop_scheduling = $utils->list_scheduling;
  my %params = (
     title => 'Centro sportivo - Corsi',
     page      => 'corsi',
     path      => 'Corsi',
     courses_price => \@loop_prices,
     courses_scheduling => \@loop_scheduling,
     LOGIN     => $sess_params{is_logged},
     USER      => $sess_params{profile},
     attempt   => $sess_params{attempt}
     );
  $utils->dispatcher('corsi', %params);
}
```

Funzione `dispatcher` all'interno di `UTILS.pm`, avendo per convenzione `$route` il nome del template a cui la route è associata, esso viene richiamato e popolato con i parametri contenuti in `%params` settati nella funzione `corsi` in `load.cgi`:

```
sub dispatcher {
  my $self = shift;
  my $route = shift;
  my %params = @_;
  my $template = HTML::Template->new(filename => $route.".tmpl", utf8 => 1);
  foreach(keys %params){
     $template->param($_ => $params{$_});
  }
  my @loop_news = $self->getNews;
  foreach(@loop_news){
     delete $_->{N_ID};
  }
  $template->param(NEWS => @loop_news);
  print "Content-Type: text/html\n\n", $template->output;
}
```

Ogni input e' stato validato oltre che lato client via javascript, anche lato server, utilizzando funzioni di validazione appositamente create, in modo da garantire procedure di login, registrazione, prenotazione e modifica prive di spiacevoli errori.