POLITECNICO DI TORINO

COLLEGIO DI INGEGNERIA INFORMATICA, DEL CINEMA E MECCATRONICA

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Sistema di creazione automatica di moduli per la registrazione dati su piattaforme on-line

Relatore
prof. Guido Perboli

Laureando
Davide Piga
matricola: 160191

Supervisore aziendale
Risolviamo s.r.l.
dott. ing. Gianluca Cumani

Aprile 2018

Indice

Elenco delle figure

Introduzione

Fin dal principio, l'uomo ha sempre cercato di ampliare la sua conoscenza. Proprio la conoscenza ha un ruolo fondamentale nell'esistenza degli esseri umani, e si riflette in ogni ambito della vita. La ricerca ha bisogno di sempre più conoscenze per fare nuove scoperte, appassionati di tutto il mondo cercano costantemente di sapere qualcosa in più sui propri interessi; soprattutto, però, le aziende hanno bisogno di conoscere gusti e abitudini di vecchi, nuovi e potenziali clienti per poter migliorare i propri prodotti.

E in un mondo dove, grazie alla recente diffusione di internet e della digitalizzazione, tutto ha subito una forte accelerazione, si potrebbe rischiare di rimanere indietro. L'aspetto positivo è che se in passato raccogliere un certo tipo di informazioni poteva essere scomodo e complicato, oggi, grazie alla tecnologia, è diventato un processo quasi banale. Per questo la maggior parte (se non tutti) i perseguitori della conoscenza hanno puntato gli occhi su questi nuovi canali di informazione.

Un'azienda che voglia fare delle ricerche di mercato, può tranquillamente scordarsi dei vecchi tempi nei quali aveva solo un ridotto campione da intervistare e le invece miriadi di successive scartoffie da compilare; ora si può raggiungere un campione molto più ampio e la parte di catalogazione che ne segue è per la maggior parte automatizzata. A patto però che si sia utilizzato uno strumento correttamente creato per la raccolta delle informazioni. Lo

strumento al quale si fa riferimento è il form, non solo semplice controparte del classico questionario cartaceo ma sua vera e propria evoluzione. Il form è un documento HTML che si pone all'interno di pagine e servizi web per essere, appunto, compilato da un campione potenzialmente globale con le più svariate informazioni. Tali form sono dunque alla base del sistema di raccolta delle informazioni e per questo sono molto diffusi, basti pensare al classico form di login che spesso rappresenta la prima interfaccia tra utente e servizio. Esistono quindi migliaia di siti e applicazioni che aiutano l'utente medio a creare in maniera sempre più semplice e immediata dei form personalizzati che soddisfino i propri bisogni, ed è proprio qui che questo progetto, realizzato in collaborazione con la Risolviamo s.r.l. si va a collocare: la corretta creazione dei form.

L'obiettivo quindi che si è cercato di raggiungere è quello di offrire agli utenti interessati un innovativo metodo per la creazione dei suddetti form, che sia facilmente utilizzabile da utenti esperti e non, e che permetta una corretta immissione delle informazioni ottenute all'interno delle basi di dati con conseguente facilità di recupero delle stesse.

Allo scopo di evitare incomprensioni all'interno del presente elaborato si ritiene necessario fare una distinzione tra utenti, fruitori principali del servizio offerto di creazione dei form, e utente finale, che invece compila realmente il form creato.

Capitolo 1

Stato dell'arte

Come anticipato, le applicazioni che offrono all'utente la possibilità di creare dei form personalizzati sono una quantità notevole. I metodi da loro utilizzati per fare ciò sono svariati ma alla fine la presentazione del form completo sarà una combinazione di due specifici linguaggi: HTML e CSS. Questi linguaggi si collocano alla base di ogni applicazione visiva a cui è possibile accedere tramite la rete.

1.1 Strumenti usati comunemente

1.1.1 HTML

Nato agli albori di internet, il linguaggio HTML (HyperText Markup Language) ha lo scopo di creare e stabilire la struttura di pagine e applicazioni web. Nel corso degli anni sono state sviluppate diverse versioni di HTML, ciascuna delle quali finalizzata all'introduzione di innovazioni nella stesura delle pagine, in linea con lo sviluppo delle tecnologie e delle tendenze emerse dal web. Le prime versioni si limitavano a consentire la creazione di semplici pagine di testo con collegamenti ipertestuali, riducendo al minimo l'attenzio-

ne dedicata all'impaginazione grafica e allo sviluppo multimediale. Proprio la necessità di introdurre contenuti grafici più complessi e un'ampia gamma di elementi multimediali, ha condotto ad un'evoluzione del linguaggio HTML, fino alla sua versione più recente, HTML5.

Tra gli aggiornamenti e novità previsti da HTML5, rispetto alle versioni precedenti, si riscontrano:

- estensione a tutti i tag degli attributi indirizzati all'accessibilità, sinora limitati solo ad alcuni di essi;
- rielaborazione delle regole per la strutturazione del testo riguardante sezioni, paragrafi e capitoli.
- supporto di elementi "canvas"(estensione dell'HTML standard che permette il rendering dinamico di immagini bitmap gestibili attraverso un linguaggio di scripting) abbinati a JavaScript per la gestione di grafica vettoriale e la realizzazione di animazioni;
- supporto della geolocalizzazione, per considerare anche la diffusione di sistemi operativi mobile;
- inserimento di "HTML5 Storage" come alternativa più efficiente ai cookie;
- impiego dei cosiddetti "Web Workers", i quali sono in grado di rendere più coerente l'esecuzione di JavaScript da parte dei browser;
- estensione del supporto agli elementi dedicati al controllo dei moduli elettronici;

Un documento HTML è sostanzialmente composto da una serie di elementi, eventualmente annidati, delimitati da una coppia di tag, detti di apertura e

di chiusura. Un tag è una particolare sequenza di caratteri che indicano al parser HTML cosa farne di ciascun elemento e, nel caso del linguaggio HTML iniziano sempre col carattere "<" e finiscono con ">"; all'interno solitamente contengono il nome che definisce il tag più eventuali attributi che possono modificarne l'aspetto tipico.

La struttura di un documento HTML si compone principalmente dei seguenti elementi fondamentali:

- html, il cui scopo è quello di indicare che il file che si sta scrivendo è realizzato in linguaggio HTML. Tutto il testo, comprensivo degli altri tag, dovrà essere racchiuso tra i suoi tag di inizio e fine, <`html`> e </`html`>;

- head, ovvero la sezione di intestazione, la quale inizia con il tag <`head`> e termina con </`head`>. La presente sezione contiene almeno il titolo del documento racchiuso tra i tag <`title`> e </`title`>; il titolo dovrà contenere una descrizione del documento che sia breve, per essere efficacemente visualizzato dai programmi di navigazione nella barra del titolo della finestra, e significativa, perché viene utilizzato dai programmi che catalogano i documenti di Internet per creare indici di ricerca;

- body, ovvero la sezione corpo, la quale è racchiusa tra i tag <`body`> e </`body`> e contiene la parte del documento che viene visualizzata dai browser quando viene aperta la pagina.

1.1.2 CSS

Il CSS (Cascading Style Sheets) si occupa di curare l'aspetto meramente grafico degli appena citati documenti HTML. Inizialmente ciò avveniva all'interno delle pagine HTML, rendendole sempre più caotiche via via che venivano

introdotte nuove caratteristiche. Per questo si è deciso di creare un nuovo standard che potesse snellire il contenuto di un file HTML, lasciando solamente istruzioni necessarie al corretto funzionamento della pagina, rendendone inoltre il funzionamento più facile da capire per l'uomo. Grazie a questa separazione si possono realizzare più velocemente pagine web, ad esempio lavorando sui due aspetti in parallelo, e riutilizzare più semplicemente il codice.

Il CSS consente quindi la creazione di un "foglio di stile", ovverosia un set di regole stilistiche che descrivono come un documento HTML verrà presentato all'utente. Visto in quest'ottica, HTML viene usato per descrivere la struttura del documento e le sue varie parti incurandosi del suo aspetto, di come verrà presentato al lettore che dovrà leggere il documento. CSS in seguito descrive come gli elementi della pagina HTML verranno presentati al lettore del documento stesso. le principali proprietà che contraddistinguono le regole stilistiche, di cui il CSS si compone, sono:

- Ereditarietà: Secondo questa proprietà le impostazioni stilistiche,applicate ad un elemento, ricadono anche sui suoi discendenti, almeno fino a quando, per un elemento discendente, non si imposti esplicitamente un valore diverso per quella proprietà.

- Specificità: La specificità, come il nome può suggerire, descrive il peso relativo delle varie regole all'interno di un foglio di stile. Esistono regole ben precise per calcolarla e sono quelle che applica lo user agent di un browser quando si trova davanti ad un CSS.

- Il concetto di cascata: Nei CSS, più di un foglio di stile può influenzare la presentazione simultaneamente. Ci sono due ragioni principali per questa caratteristica:

  - Modularità, per cui un autore di fogli di stile può combinare diversi (parziali) fogli di stile per ridurre la ridondanza e l'equilibrio autore/utente.

  - Equilibrio autore/utente, per cui sia l'utente che l'autore possono influenzare la presentazione attraverso i fogli di stile. Per fa ciò, essi usano lo stesso linguaggio di fogli di stile riflettendo così una fondamentale caratteristica del web: tutti possono pubblicare. L'User Agent è libero di scegliere la tecnica per riferirsi a fogli di stile personale. A volte possono nascere conflitti fra fogli di stile che influenzano la presentazione. La risoluzione del conflitto è basato su ogni regola che ha un peso. Di default, il peso delle regole dell'utente è minore del peso delle regole dell'autore del documento. Per esempio, se nascono conflitti fra i fogli di stile di un documento e lo stile personale dell'utente, verranno utilizzate le regole dell'autore. Sia le regole dell'utente che quelle dell'autore sovrascrivono i valori di default dell'User Agent. Anche fogli di stile importati cadono a cascata con ogni altro, nell'ordine in cui vengono importati, in accordo con le regole di cascata definite prima. Ogni regola specificata nel foglio di stile sovrascrive le regole nei fogli di stile importati. Cioè, i fogli di stile importati hanno un ordine di cascata più basso rispetto alle regole previste nel foglio di stile stesso. I fogli di stile importati possono a loro volta importare e sovrascrivere altri fogli di stile, e via di seguito.

- Importanza: Se una dichiarazione viene accompagnata dalla parola chiave !important essa balza al primo posto nell'ordine di applicazione a prescindere da peso, origine, specificità e ordine.

1.1.3 Linguaggi di scripting

Il linguaggio HTML crea pagine web statiche, quindi l'utilizzo dei soli linguaggi appena citati è spesso limitante e per questo si tende ad associarli ad altri strumenti come, ad esempio, i linguaggi di scripting JavaScript (più indirizzato al lato client) e PHP (più server-side oriented).

JS è un linguaggio di script che, nonostante il nome, non ha nulla a che vedere con il linguaggio Java. Il suo scopo principale è quello di migliorare l'esperienza che si ha delle pagine web rendendole dinamiche, ad esempio con l'aggiunta di effetti interattivi basati su determinate azioni compiute dall'utente, ma anche

- validazione degli input di un form prima di inviarli al server per assicurarsi che essi siano nel formato corretto;
- caricamento di nuove pagine e invio al server di dati, tramite Ajax, senza il bisogno di ricaricare la pagina corrente;
- animazione degli elementi di una pagina, ad esempio spostandoli, ingrandendoli/rimpicciolendoli, facendoli apparire o sparire, e così via;
- riproduzione dicontenuti interattivi come audio e video;
- trasmissione di informazioni riguardo l'utente e le sue attività online, per statistiche, annunci personalizzati etc.

Ormai è praticamente diventato uno standard de-facto ed è presente in quasi tutte le pagine e applicazioni web.

PHP (inizialmente acronimo di Personal Home Page, ma in seguito modificato con l'attuale significato ricorsivo di PHP: Hypertext Preprocessor) è un linguaggio di scripting più evoluto, chè è nato per lo sviluppo di pagine

web ma che può essere anche utilizzato in maniera più generica. Proprio per questo suo più ampio spettro di utilizzo, ha bisogno di un motore dedicato per girare, in grado di interpretare il codice scritto in PHP e produrre un risultato adeguato, a differenza dei documenti HTML che vengono interpretati in automatico da qualsiasi browser. Esistono comunque svariati programmi gratuiti in grado di interpretare il codice PHP, quindi questo non comporta un ostacolo nella realizzazione di un progetto. Parti di codice PHP possono essere inserite all'interno di documenti HTML e viceversa. L'utilizzo del primo caso è consigliato, in quanto basta aprire il tag apposito con `<?php`, inserire le varie parti PHP e poi chiuderlo con `?>`. Il caso contrario richiede invece di mandare in output ogni comando HTML attraverso il comando PHP `echo`.

1.1.4 Database

I database (anche base di dati o banca dati) sono gli archivi di tutta la conoscenza digitale. Tutto viene salvato al loro interno ed è importante che ciò avvenga correttamente, pena l'impossibilità futura di estrapolare efficacemente i dati richiesti.

1.2 Confronto di servizi esistenti

In base alla qualità del servizio richiesto è possibile riscontrare una notevole varietà nell'offerta proposta. In particolare si può distinguere tra servizi gratuiti e a pagamento. Sebbene i servizi a pagamento siano più professionali e talvolta più completi, quelli gratuiti offrono comunque un'ampia gamma di azioni possibili dalle quali attingere. Tipicamente però, l'utilizzatore si trova davanti ad un menù da cui scegliere diverse strutture di blocchi da riempire

e personalizzare, e che verranno poi inseriti nel form. Per fare questo, le varie applicazioni offrono svariati modi ma tendono ad essere tutte un po' macchinose e, delle volte, complicate se non addirittura fastidiose.

Pagamento:

Come, ad esempio, Google Surveys[13]; questo servizio offerto da Google sembra offrire la possibilità di far compilare i propri form ad un target ben preciso, si possono includere persone in base all'età, sesso, posizione geografica, ma anche selezionare utilizzatori di un certo tipo servizio (come social media o servizi di streaming). Per poter realizzare quest'ultima parte però,

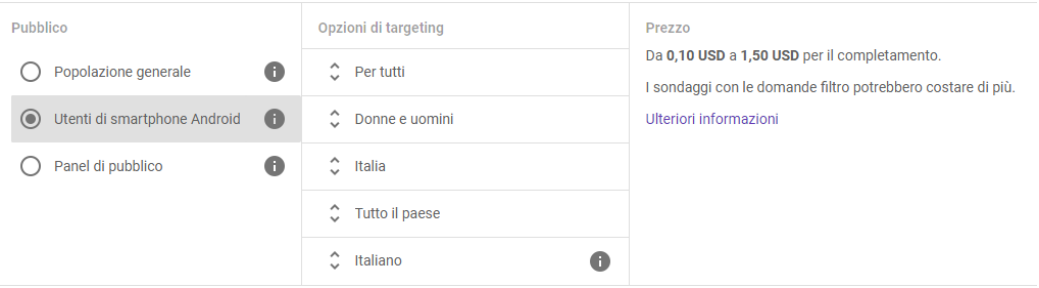

Figura 1.1: Google Surveys, opzioni preliminari

gli sviluppatori hanno creato un'app chiamata Opinion Rewards[12], che promette di ricompensare con credito Google Wallet gli utenti che la utilizzano. Essi dovranno rispondere innanzitutto a delle domande generali atte a etichettare ciascun utente come adatto o non adatto ai vari questionari, e poi riceveranno direttamente sulla loro applicazione i vari sondaggi da completare. Questa forma di ricompensa è spesso usata dalle varie società che creano sondaggi per invogliare gli utenti più disparati a fornire loro preziose informazioni rispondendo a tali sondaggi.

Particolari richieste da parte dell'utente che crea il sondaggio però, fanno rapidamente aumentare il prezzo del servizio offerto da Google Surveys, il

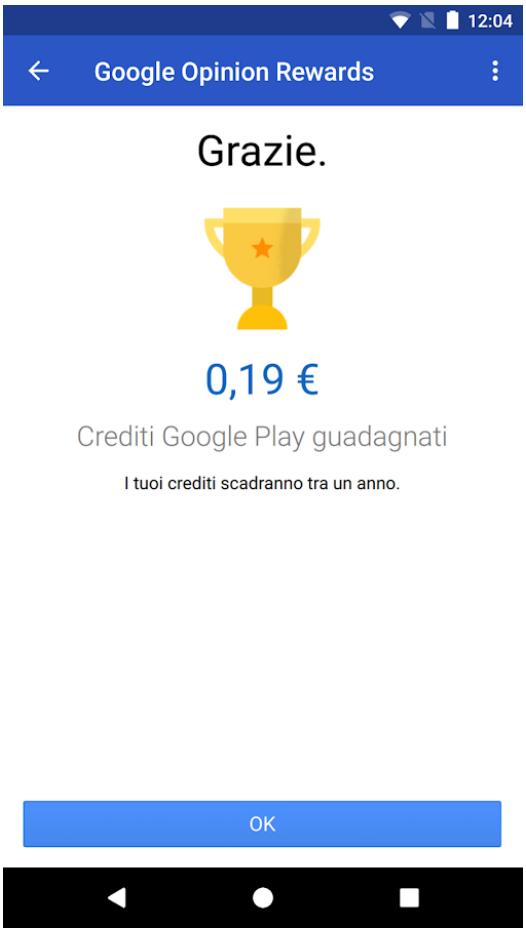

Figura 1.2: Opinion Rewards, ricompensa per l'utente finale

quale può facilmente raggiungere cifre non trascurabili, soprattutto per le piccole/medie imprese.

Nonostante sia un servizio a pagamento, non è data la piena libertà all'utilizzatore di creare form a lui più congeniali. Durante la fase di creazione infatti può capitare di imbattersi in qualche vincolo come, ad esempio, nella schermata di creazione di una domanda aperta con risposta di tipo testo, dove compare il seguente messaggio:

Le domande di testo aperte richiedono tempo e impegno supplementari da parte degli intervistati, per cui supportiamo solo

Survey type	1 question survey	2-10 question survey
General population or Android-smartphone users, no screening questions	$0.10 per completed response	$1.00 per completed response
Age, gender, location targeting, no screening questions	$0.15 per completed response	$1.50 per completed response
Android audience panels, no screening questions	$0.30 per completed response	$3.00 per completed response
User-list targeted Surveys 360 only. Contact Surveys Sales if interested.	$1.00 per completed response	$10.00 per completed response
Zip-code targeted Surveys 360 only. Contact Surveys Sales if interested.	$0.60 per completed response	$6.00 per completed response

Figura 1.3: Google Surveys, tabella dei costi

due domande di testo aperte per sondaggio.

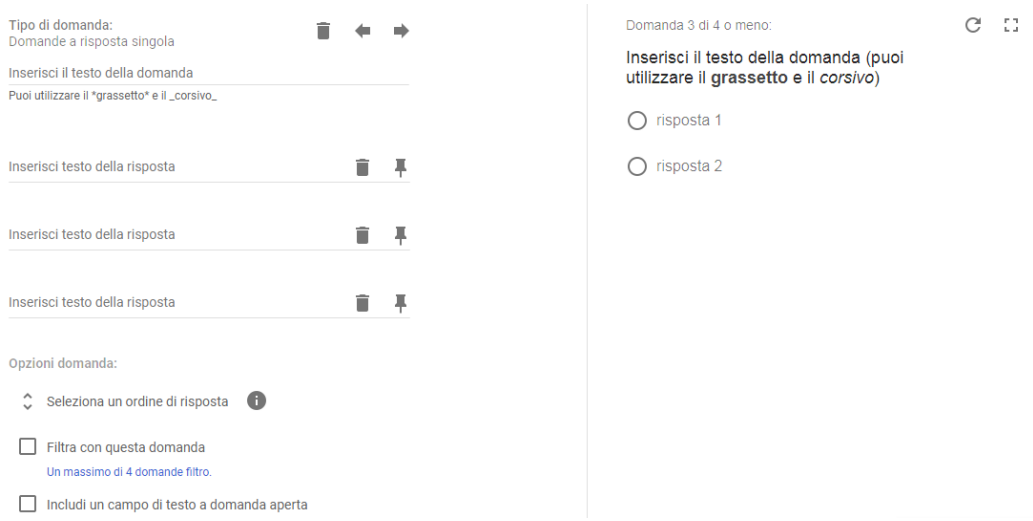

Figura 1.4: Google Surveys, esempio creazione domanda

Gratuiti:

Esistono tante possibilità di scelta per chi cerca un servizio gratuito che possa

aiutarlo nella realizzazione di un form. Una delle tante è Google Forms[11], creata ugualmente da Google, ma sicuramente più simile al prodotto realizzato in questo progetto rispetto al precedente esempio a pagamento.

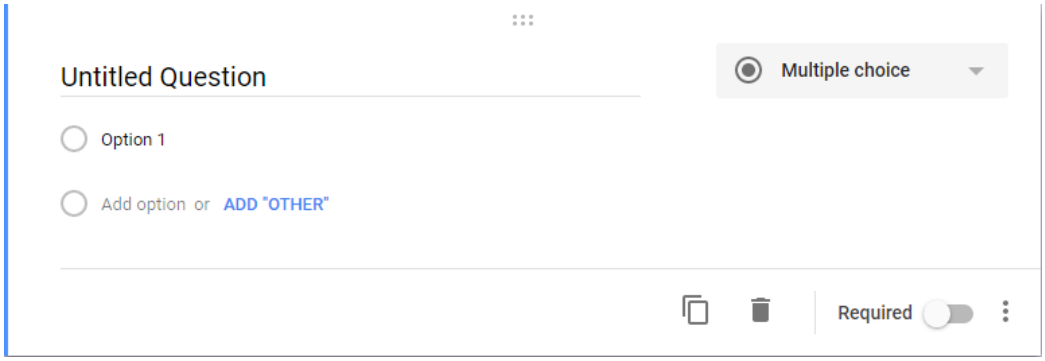

Figura 1.5: Google Forms, esempio creazione domanda

Questo servizio offre la possibilità di creare un form secondo le proprie esigenze (non si sono riscontrate particolari restrizioni) e permette naturalmente di controllare le risposte ricevute, sia individualmente sia di estrapolarne una visione di insieme, grazie all'ausilio di alcuni grafici. Essendo Google Forms un servizio offerto dal pacchetto Google Drive, tali grafici sono realizzati attraverso un altro servzio disponibile all'interno del pacchetto: Google Spreadsheets (l'equivalente di Microsoft Excel, del pacchetto Microsoft Office). Anche la lista di risposte può essere facilmente importata in un documento Spreadsheet o può, eventualmente essere esportata come file.csv, estensione appunto utilizzata per il salvataggio di fogli di calcolo. Non è quindi data possibilità di salvare i dati all'interno di un database e, sebbene sia possibile importare un file.csv, non è detto che tutti gli utenti siano in grado di farlo.

Esiste un'ulteriore fascia composta da servizi a pagamento ma che offrono una parte limitata di azioni che possono essere eseguite gratuitamente. Tra i

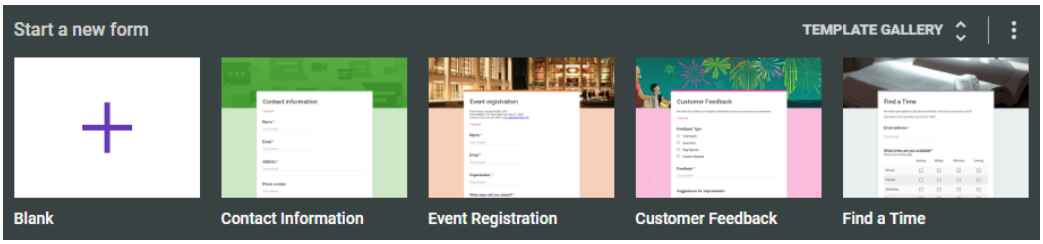

Figura 1.6: Google Forms, selezione template precompilati

servizi osservati, MailChimp[22] offre alcune delle caratteristiche che si sono tenute in considerazione per la realizzazione di questo progetto come, ad esempio, la possibilità di creare un form da includere poi nel proprio sito.

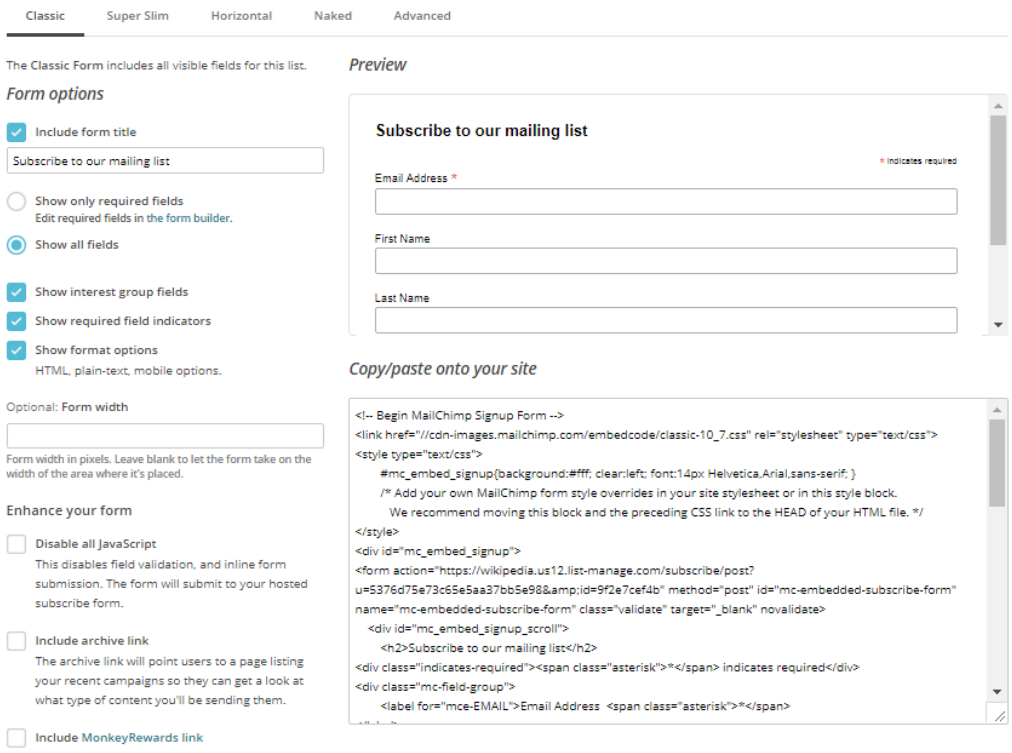

Figura 1.7: MailChimp, creazione form precompilato

Purtroppo, però, essendo MailChimp un sito non specializzato per questo compito ma che offre vari servizi, la scelta di questi form (almeno per gli

account gratuiti) è molto scarna.

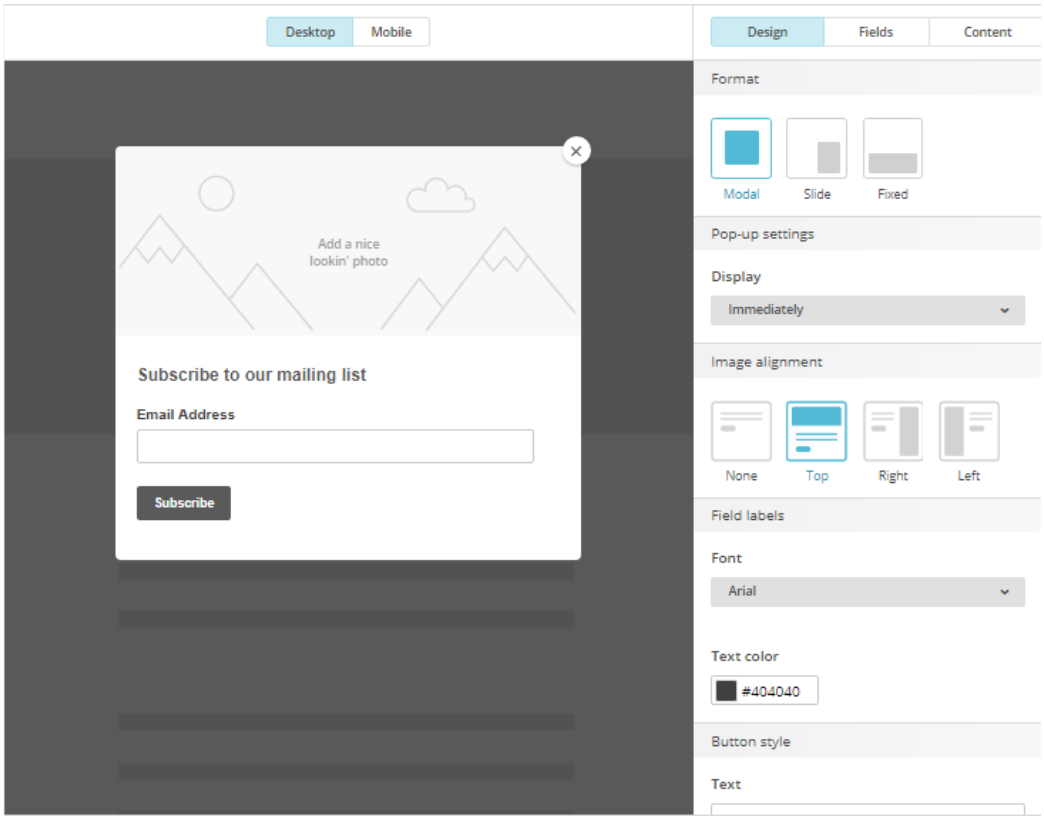

Figura 1.8: MailChimp, creazione form pop-up

Capitolo 2

Problematiche

In seguito all'osservazione dei servizi disponibili on-line per la creazione di vari form, appare chiaro che tali servizi presentano uno o più aspetti nei quali sono carenti. Un prodotto può offrire determinate caratteristiche ma non delle altre, che magari sono presenti in un prodotto concorrente, e non è certo comodo ricorrere all'utilizzo di più servizi per la creazione di un singolo form, miscelando poi parti dei vari risultati ottenuti, in una sorta di chimera digitale, nel tentativo di ottenere il risultato sperato.

Come se non bastasse, il problema principale sta nel fatto che tutti questi form creati sono fini a sé stessi, in quanto nessun servizio offre la possibilità di riutilizzarli, né per intero né in parte, causando un non trascurabile spreco di tempo e risorse. Questa problematica rappresenta un grosso ostacolo, soprattutto per le aziende, sia dal punto di vista dei costi relativi all'utilizzo del tempo per la stessa risorsa, sia dal punto di vista del desiderio di evoluzione della tecnica dell'azienda stessa.

Si sente quindi la mancanza di un sistema che permetta di recuperare velocemente parti precedentemente svolte, e che permetta di farlo in maniera semplice, così da dare la possibilità anche agli utenti inesperti di fruirne.

Capitolo 3

Soluzioni proposte

Come si è visto quindi, non esiste la possibilità di poter realizzare un qualcosa di facilmente riutilizzabile.

Dopo una comparazione dei prodotti sul mercato e una seguente analisi dei punti chiave dei quali deficitano, ci si è effettivamente resi conto della possibilità di creare un servizio più avanzato rispetto alla concorrenza.

Si è quindi palesata la necessità di un'evoluzione, che portasse ad una maggiore rapidità nella costruzione di form e ad una migliore riutilizzabilità di quanto creato, senza intaccarne però la qualità. A tal fine è stato realizzato un nuovo metalinguaggio, che potesse sopperire a tale mancanza e permettesse di portare a compimento il progetto.

3.1 Cos'è un metalinguaggio

Il termine metalinguaggio può assumere diverse sfaccettature in base al campo nel quale viene usato ma, grosso modo, indica un insieme di termini, simboli, gesti (quindi non necessariamente un vero e proprio linguaggio) che manifestino un qualche tipo di informazione riguardo a un altro linguaggio.

L'uso dei metalinguaggi in campo informatico viene anche detto metaprogrammazione, dato che ogni linguaggio di programmazione può essere inteso come metalinguaggio rispetto al linguaggio macchina (o ad altri linguaggi di basso livello). L'utilizzo di questi metalinguaggi, a fronte di un periodo iniziale di apprendimento, porta spesso alla riduzione del tempo di sviluppo, che è anche uno degli obiettivi che si pone questo progetto.

Per poter utilizzare questi linguaggi però, bisogna descrivere in modo formale e senza ambiguità le sue regole (anche dette grammatica); tale insieme prende anche il nome di metasintassi. La BNF (Backus-Naur form) è quella che viene solitamente utilizzata per raggiungere questo scopo. Ne esiste anche un'evoluzione, chiamata Extended Backus-Naur form, dove col termine "estesa" si intende un miglioramento della leggibilità e della sinteticità e che quindi, in realtà, è più compatta della precedente.

Per necessità correlate al funzionamento ottimo del programma, si è scelto di descriverne la grammatica utilizzando la classica BNF e non la sua versione estesa.

3.2 Utilizzo del metalinguaggio

Il metalinguaggio realizzato per questo progetto va utilizzato per creare dei file di testo che verranno dati in pasto ad un programma, spiegato più avanti, il quale analizzerà tali file e restituirà un documento HTML contenente il form desiderato, pronto per essere utilizzato.

La struttura di questi file di input è organizzata in blocchi, e possono essere indipendenti o raggruppati. Intuitivamente, il creatore del file utilizzerà un blocco per ogni domanda che desidera ritrovare poi nel form finale. Prima di ilustrare nel dettaglio come si creano e utilizzano i vari blocchi, è bene far

sapere che il metalinguaggio realizzato è caseless; questo significa che tutte le parole chiave del linguaggio possono essere scritte indifferentemente in MAIUSCOLO, minuscolo, Proper o come più comodo all'utilizzatore. Nel corso della spiegazione, le parole chiave saranno scritte in MAIUSCOLETTO, mentre le parti che andranno sostituite con dei valori a discrezione dell'utilizzatore saranno in *corsivo*, per permetterne un immediato riconoscimento rispetto al resto del testo.

3.3 Creazione di un file / Tipi di blocco

I blocchi utilizzabili si dividono sostanzialmente in 3 categorie:

- blocchi base,
- blocchi complessi (o raggruppati),
- blocchi caricati.

I blocchi base sono, per l'appunto, la base del metalinguaggio. Essi vengono richiamati anche dagli altri tipi di blocchi; è possibile creare un form senza utilizzare gruppi di blocchi o caricare file esterni, ma non è possibile non utilizzare i blocchi base.

3.3.1 Blocco base

Dal punto di vista della creazione dei form, i blocchi base sono delle unità atomiche; dal punto di vista analitico però, essi possono essere ulteriomente suddivisi in più parti per poter essere analizzati meglio.
La struttura generica di un blocco base, suddivisa per righe, è come compare in figura 3.1.

Innanzitutto, ogni blocco base inizia, ed è quindi definito, con la parola chiave BLOCK. Questo termine deve per forza comparire all'inizio di ogni

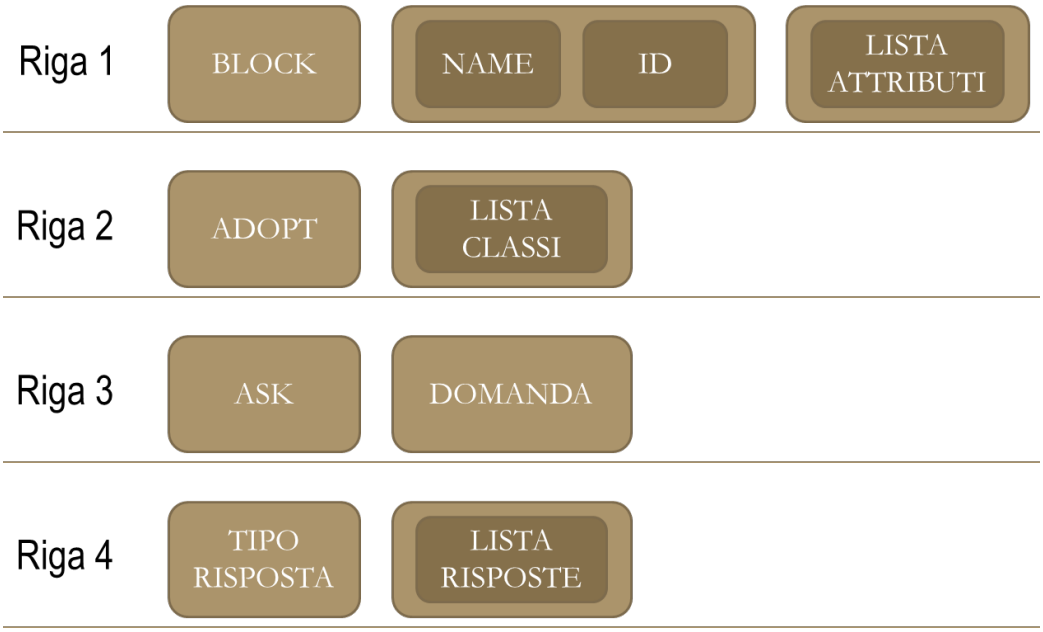

Figura 3.1: Organizzazione di un blocco base riga per riga

blocco base, non può essere omesso.
Subito dopo, seguono il *nome* e l'*ID* del blocco, ciascuno racchiuso tra doppi apici.

```
BLOCK "name" "id"
```

ex. `BLOCK "pippo" "pippo1"` ⇒ `name="pippo" id="pippo1"`

All'interno di un documento HTML è possibile avere più parti con lo stesso nome, ma non è invece lecito avere più parti con lo stesso ID[1]; per questo motivo si è ritenuta necessaria una differenziazione. Ciononditméno, è possibile omettere uno dei due termini se si vuole assegnare ad entrambi lo stesso valore.

```
BLOCK "nameid"
```

[1]ID sta per identificatore e, in campo informatico, spesso sottintende anche il termine univoco

ex. `BLOCK "pippo"` ⇒ `name="pippo" id="pippo"`

Infine, è possibile omettere entrambi ed il programma, automaticamente, attribuirà loro un valore univoco.

```
BLOCK
```

ex. `BLOCK` ⇒ `name="autonomeunivoco" id="autonomeunivoco"`

In caso sia l'utente a decidere i vari *ID* per i blocchi, egli deve accertarsi di non immettere dei duplicati. Il programma esegue per sicurezza un controllo, ma l'unica cosa che può fare è restituire un messaggio d'errore, non è in grado di sostituire autonomamente i duplicati ed è giusto che sia così perché, se l'utente inserisce un ID personalizzato, evidentemente ha un valido motivo per farlo (ad esempio viene richiamato da un'altra parte) e dare la possibilità al programma di modificarlo direttamente, magari senza avvisare il creatore, potrebbe portare più problemi che benefici.

3.3.1.1 Attributi

A seguire, possono essere aggiunti eventuali attributi, quali REQUIRED, OPTIONAL e HIDDEN.

REQUIRED è l'opzione più comune, indica che l'utente finale che compilerà il form deve obbligatoriamente compilare/rispondere a questa domanda.

```
BLOCK "nameid" REQUIRED
```

Essendo l'ozione più comune, essa è stata inoltre selezionata come opzione di default ed è quella che il programma utilizzerà in caso di assenza di attributi.

```
BLOCK "nameid"
```

Le due righe mostrate qui sopra sono quindi equivalenti e produranno lo stesso risultato.

OPTIONAL è, come si può intuire, il caso contrario a REQUIRED e indica che non è necessario rispondere a tale domanda.

```
BLOCK "nameid" OPTIONAL
```

Non è possibile utilizzare contemporaneamente OPTIONAL e REQUIRED sullo stesso blocco.

~~`BLOCK "nameid" REQUIRED OPTIONAL`~~

HIDDEN è infine l'ultimo attributo selezionabile, e indica un blocco che non è inizialmente visibile.

```
BLOCK "nameid" HIDDEN
```

Se si vuole rendere visibile tale blocco in seguito, è necessario aggiungere le parole chiave SHOW IF (o SHOWIF) seguite dall'*id* del blocco e dal *valore* che questo blocco deve assumere per far comparire il blocco nascosto.

```
BLOCCO "nameid" HIDDEN SHOW IF other_block_id=a_certain_value
```

Opzionalmente, in questo caso, il termine HIDDEN può essere omesso.

```
BLOCCO "nameid" SHOW IF other_block_id=a_certain_value
```

Dato che un blocco marcato HIDDEN non sarà visibile al compilatore del form, a meno che non venga soddisfatta una determinata condizione, questo tipo di blocco è intrisecamente un blocco opzionale; avere una domanda nascosta ma a cui dover rispondere obbligatoriamente porterebbe all'impossibilità di poter terminare la compilazione del form. Per questo motivo non è possibile utilizzare l'attributo HIDDEN in coppia con REQUIRED perché si creerebbe un paradosso logico. Non è invece necessario specificare l'attributo OPTIONAL per il corretto funzionamento del parametro HIDDEN in quanto ciò viene fatto in automatico.

~~`BLOCK "nameid" REQUIRED HIDDEN`~~

ADOPT è la parola chiave che serve ad indicare quali classi il blocco deve adottare. Essa va posta su una nuova riga e deve essere seguita dall'elenco delle classi da utilizzare, separate da una virgola.

```
ADOPT class1, class2, classN
```

Nel caso in cui non si vogliano far ereditare al blocco le classi generali (spiegate più avanti, alla sezione 3.4, pag. 32), bisogna aggiungere l'opzione ONLY prima dell'elenco delle classi

```
ADOPT ONLY class1, class2, classN
```

3.3.1.2 Domande e risposte

Nella riga successiva va inserita la domanda da porgere all'utente finale, preceduta dalla parola chiave ASK: (anche senza i due punti):

```
ASK: Come ti chiami?
```

La stringa inserita dall'utente assumerà il ruolo di label per il campo destinato alla risposta. È eventualmente possibile omettere completamente questa riga se lo si desidera.

L'ultima riga è dedicata al tipo di risposta che l'utente finale può dare, con eventuale elenco di opzioni dalle quali scegliere. La prima parola della riga deve essere il tipo di risposta, che può essere scelta tra uno dei tipi base offerti dal linguaggio HTML:

Risposte aperte tra le quali spiccano TEXT, TEXTAREA (o AREA), PASSWORD; esse non hanno quindi bisogno di una lista di opzioni ma possono eventualmente essere seguite da un suggerimento (o placeholder, letteralmente: segnaposto)

```
TEXT Inserisci qui il tuo nome
```

o perfino da una risposta precompilata compresa tra parentesi tonde.

```
TEXT (Davide)
```

Nel primo caso otterremo quindi un suggerimento destinato ad aiutare l'utente finale con la compilazione del form, mentre nel secondo caso otterremo un campo testo con un valore precompilato al suo interno che è a tutti gli effetti una risposta già pronta per l'utente finale, il quale può decidere di volta in volta se tenere la risposta proposta o modificarla a suo piacimento.

Figura 3.2: Campo di testo con suggerimento interno

Figura 3.3: Campo di testo con valore precompilato

Le principali differenze fra TEXT e TEXTAREA sono che la textarea ha uno spazio più grande di input e, in caso non dovesse bastare, può essere ingrandito a piacere dall'utente finale tramite la maniglia presente nell'angolo in basso a destra.

In linea con le innovazioni introdotte da HTML5, altri tipi di risposta possibili (ma che, almeno per il momento, accettano come unico paramentro solamente un eventuale valore preimpostato) sono: BUTTON, COLOR, DATE, DATETIME, DATETIMELOCAL, EMAIL, FILEUPLOAD, HIDDEN, IMAGE, MONTH, NUMBER, PASSWORD, RANGE, RADIO, RESET, SEARCH, SUBMIT,

Figura 3.4: Comparazione tra il blocco textarea e il blocco text

TEXT, TIME, URL, WEEK. [2]

La maggior parte di questi è del tutto simile al campo TEXT ma processa i dati ricevuti in maniera differente; si riportano alcuni esempi di quelli che potrebbero essere i tipi più utilizzati:

Risposte chiuse come RADIOBUTTON (o RADIO), CHECKBOX (o CHECK), SELECT (o DROPDOWN o DROP), nelle quali l'utilizzatore che sceglie una di queste tipologie deve anche dare una lista di una o più opzioni dalla quale l'utente finale può attingere. Tali opzioni vanno elencate di seguito, separate da un pipe "|" e possono essere racchiuse all'interno di parentesi quadre.

Nel caso di utilizzo del comando SELECT otterremo il cosiddetto menù a tendina, come mostrato in figura 3.9. In questo tipo di input, l'utente può selezionare un solo tipo di risposta.

```
SELECT [ Opzione 1 | Opzione 1 | Opzione 3 | Opzione 4]
```

Come si evince dalla figura, il sistema selezionerà in automatico la prima opzione; in caso si volesse scegliere di mostrare un'altra risposta a chi compilerà il form, senza però sconvolgere l'ordine delle risposte, è possibile indicare al programma quale mostrare racchiudendola tra parentesi tonde.

[2] Alcuni di questi tipi di input potrebbero non essere compatibili con Internet Explorer 9 (e versioni precedenti) o Safari

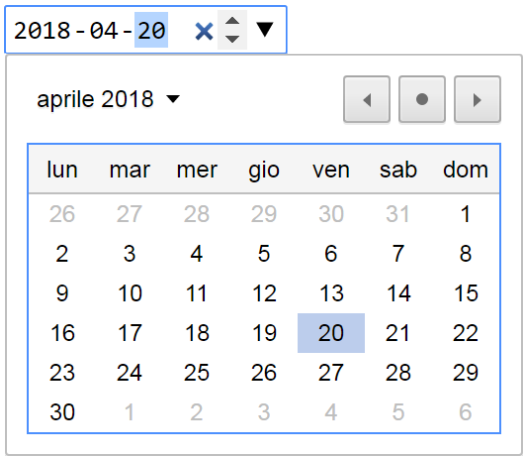

Figura 3.5: Calendario ottenibile tramite l'input di tipo DATE

Figura 3.6: L'input di tipo NUMBER permette di selezionare un numero con gli appositi tasti sul lato destro

Figura 3.7: L'input di tipo PASSWORD maschera i dati immessi

Figura 3.8: L'input di tipo RANGE permette di ottenere uno slider

```
SELECT [ Opzione 1 | Opzione 1 | (Opzione 3) | Opzione 4]
```

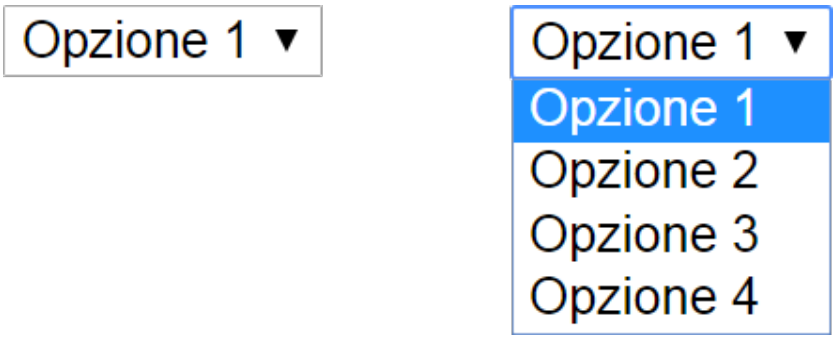

Figura 3.9: Come si presenta un blocco di input di tipo select inizialmente e durante l'interazione da parte dell'utente finale

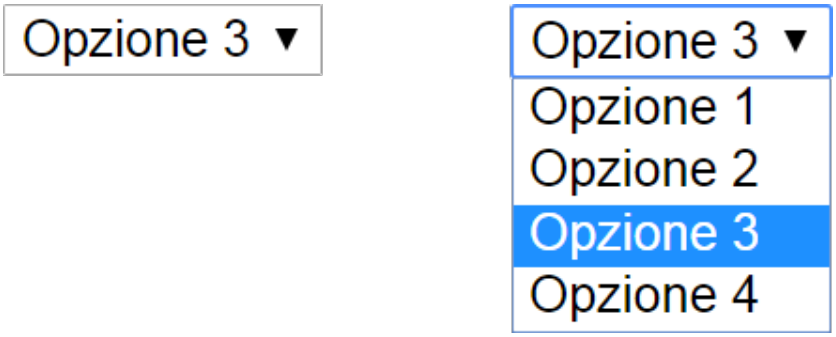

Figura 3.10: Un blocco di input di tipo select con una differente opzione selezionata come prima scelta

Alternativa al tipo SELECT è il tipo RADIO, nel quale l'utente può ugualmente selezionare una sola risposta ma, a differenza della select, le opzioni disponibili vengono mostrate direttamente.

```
RADIO [ Opzione A | Opzione B | Opzione C ]
```

In caso si volesse scegliere una risposta predefinita per l'utente, essa andrà racchiusa tra parentesi tonde

```
RADIO [ Opzione A | (Opzione B) | Opzione C ]
```

La figura 3.11 mostra la forma finale di entrambi i casi.

Mentre la selezione tramite RADIOBUTTON e SELECT ammette al più una sola risposta, quella realizzata tramite CHECKBOX prevede la possibilità di selezionare multiple risposte, potenzialmente anche tutte.

```
CHECKBOX [ Opzione A | Opzione B | Opzione C | Opzione D]
```

Anche in questo caso si possono indicare le eventuali risposte predefinite per l'utente racchiudendole tra parentesi tonde

```
CHECKBOX [ Opzione A | (Opzione B) | (Opzione C) | Opzione D]
```

Le differenze sono mostrate in figura 3.12.

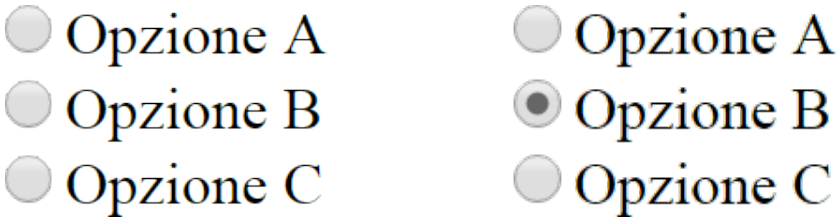

Figura 3.11: Blocco di input di tipo radio senza e con risposta preselezionata

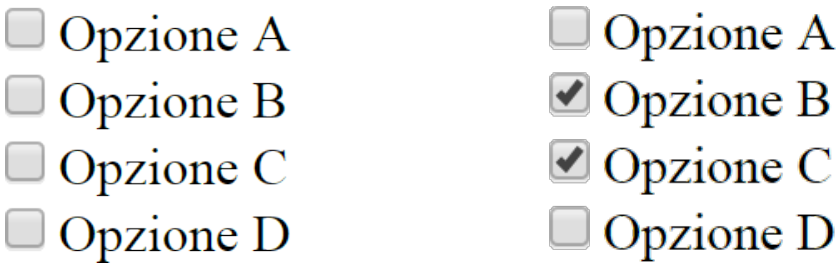

Figura 3.12: Blocco di input di tipo checkbox senza e con risposte preselezionate

3.3.2 Blocchi complessi

La maggior parte delle volte sarà probabilmente necessario utilizzare più blocchi assieme per raggiungere lo scopo desiderato; è quindi possibile raggruppare più blocchi tramite il costrutto NEW GROUP (o solo GROUP) seguito dai blocchi voluti e terminato dalle parole chiave END GROUP (o solo END). Tale procedimento è ricorsivo e possiamo quindi avere gruppi all'interno di altri gruppi.

```
NEW GROUP
   blocco #1
   blocco #2
   NEW GROUP
      blocco #3
      blocco #4
   END GROUP
   ...
   blocco #N
END GROUP
```

In caso di creazione di gruppi contigui e di utilizzo della sintassi breve, è necessario porre END (del primo gruppo) e GROUP (del secondo) su due linee diverse per evitare interpretazioni errate da parte del programma.
È inoltre possibile far seguire NEW GROUP da eventuali attributi, proprio come un blocco base.
Operativamente, la coppia di comandi (NEW GROUP, END) non fa altro che racchiudere i propri sotto-blocchi, una volta trasformati nel loro corrispettivo HTML, all'interno di un tag HTML .

3.3.3 Caricamento di un blocco

La vera comodità di questo sistema risiede nella possibilità di salvare dei blocchi creati per poterli facilmente riutilizzare in seguito. Per fare ciò si utilizza la parola chiave LOAD, seguita dal nome del blocco richiesto

```
LOAD "nome_del_blocco_da_caricare"
```

Questa è la peculiarità che manca agli servizi disponibili online al momento. Grazie a questo sistema si possono riutilizzare i (gruppi di) blocchi già creati in passato, basta richiamarli e il programma li carica per voi. Una volta creati i blocchi necessari, in teoria si potrebbe creare un nuovo form semplicemente tramite una serie di LOAD, o comunque apportando solamente minime modifiche.

Per personalizzare di volta in volta i blocchi, è possibile passare dei parametri utilizzando il termine SET seguito dal nome del blocco e dall'id che si vuole assegnare a tale blocco.

```
LOAD "blocco$\alpha$" SET bloccoX AS nuovoid1, bloccoY AS nuovoid2
```

Al fine di garantire l'unicità degli ID anche in caso di blocchi richiamati più volte in diverse parti del form, le stringhe `bloccoX` e `bloccoY` conterrano in realtà tutta la catena dei blocchi richiamati (e quindi dei file aperti) separati da un underscore(_).

Esempio:

Si supponga di caricare un insieme di file annidati, con una struttura simile alla seguente:

alpha

↳ bravo

↳ charlie

↳ delta

ognuno dei file viene caricato e, ad un certo punto, al suo interno compare l'istruzione per caricare un nuovo file.

Ora, si supponga inoltre che all'interno del file delta sia presente un blocco con id "pippo"; a questo punto per sostituire pippo con, ad esempio, "pluto", bisognerebbe procedere nei seguenti modi:

- se ci si trovasse all'interno del file Alpha (e si volesse caricare quindi il file bravo), la sintassi sarebbe

```
LOAD "bravo"
SET bravo_charlie_delta_pippo AS pluto
```

- se invece ci si trovasse all'interno del file Bravo (e si volesse caricare quindi il file charlie), la sintassi sarebbe

```
LOAD "charlie"
SET charlie_delta_pippo AS pluto
```

- se, infine, ci si trovasse all'interno del file Charlie, la sintassi sarebbe

```
LOAD "delta"
SET delta_pippo AS pluto
```

3.4 Finalizzazione di un form

Quanto visto finora rappresenta la parte più consistente del processo di creazione di un form completo, ma è necessario aggiungere ancora alcuni dettagli. Innanzitutto, la creazione di un form è indicata dal comando START FORM. Dato che ogni form dovrà salvare i dati immessi dagli utenti finali in un database, è possibile passare una action da far eseguire al form, attraverso il comando DO seguito dalla action stessa tra doppi apici. Infine si può indicare il metodo di trasmissione dei dati tramite il comando THROUGH seguito dal metodo scelto; si ricorda che i metodi possibili sono due: GET e POST. In caso di omissione di quest ultimo comando, il programma selezionerà in automatico il metodo POST.
Opzionalmente si può inoltre indicare la fine del form inserendo nell'ultima riga del file le parole END FORM.

```
START FORM DO "action" THROUGH POST
    *Lista dei blocchi*
END FORM
```

Al di sotto di questa prima riga è possibile definire delle classi generali, come già accennato parlando della creazione dei blocchi singoli. Anche la sintassi è simile: si usa ugualmente il termine ADOPT seguito dall'elenco delle classi, vero, ma in questo caso il tutto va preceduto dal termine ALL e il tipo di blocco.

```
ALL text ADOPT classX, classY, classZ
```

L'esempio qui sopra sta ad indicare che tutti i blocchi di tipo text erediteranno le classi chiamate classX, classY e classZ. In caso non si volessero far

ereditare queste classi ad un determinato blocco, si ricorda, bisognerà aggiugere a quel blocco l'opzione `ADOPT ONLY` seguita dall'elenco di classe che dovrà ereditare.

Infine, è necessario aggiungere un blocco di tipo SUBMIT col quale l'utente finale possa interagire, una volta compilato il form, per poter permettere l'invio dei dati.

```
BLOCK SUBMIT
```

Figura 3.13: Il pulsante submit, con l'etichetta di default automaticamente tradotta nella lingua dell'utilizzatore

La sintassi prevede come parametro opzionale solamente la possibilità di modificare quanto appare scritto sul bottone.

```
BLOCK SUBMIT "Form compilato"
```

Figura 3.14: Il pulsante submit, con l'etichetta personalizzata dal creatore del form

Si presentano ora alcuni esempi di confronto tra metalinguaggio e corrispettivo codice HTML per chiarire meglio i concetti appena descritti.

Esempio 1 Blocco minimo funzionale. Questo esempio riporta il numero minimo di termini appartenenti al metalinguaggio da dover utilizzare per avere un equivalente blocco HTML funzionante e correttamente definito.
Ecco cosa è necessario scrivere col metalinguaggio: `BLOCK TEXT`
Ecco l'equivalente HTML: `<input type="text" name="block1" id="block1" required />`

Figura 3.15: Esempio 1a: Blocco HTML funzionante ottenuto scrivendo il minor quantitativo possibile di metalinguaggio

Il blocco appena esposto non dà molte informazioni all'utente finale che compilerà il form, quindi è consigliabile aggiungere almeno un suggerimento:
Metalinguaggio: `BLOCK TEXT Nome`
Equivalente HTML: `<input type="text" name="block1" id="block1" placeholder="Nome" required />`

Figura 3.16: Esempio 1b: Blocco HTML funzionante ottenuto scrivendo il minor quantitativo possibile di metalinguaggio, con aggiunta di suggerimento

Esempio 2 Un semplice blocco atto a raccogliere le informazioni anagrafiche di un utente potrebbe essere, ad esempio, così creato:

Metalinguaggio:
```
NEW GROUP "Anagrafica"
BLOCK "Name"
ASK Nome
TEXT
BLOCK "Surname"
ASK Cognome
TEXT
BLOCK "Gender"
ASK Sesso
RADIO [M|F]
END GROUP
```

Ed ecco il corrispettivo HTML:

```
<div name="Anagrafica" id="Anagrafica">
    <label for="Name">Nome</label>
    <input type="text" name="Name" id="Name" required />
    <label for="Surname">Cognome</label>
    <input type="text" name="Surname" id="Surname" required />
    <label for="Gender">Sesso</label>
    <input type="radio" name="Gender" id="Gender1" value="M" required/>
    <label for="Gender1">M</label>
    <input type="radio" name="Gender" id="Gender2" value="F" required/>
    <label for="Gender2">F</label>
</div>
```

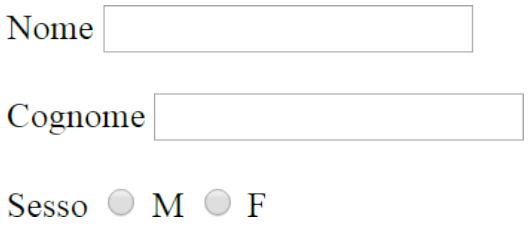

Figura 3.17: Esempio 2

Esempio 3 Mentre gli esempi precedenti riguardavano solo la creazione di blocchi, in questo esempio si mostrerà la realizzazione di un form completo, seppur banale, sfruttando la vera forza di questo progetto, ovvero la riutilizzabilità di blocchi già creati. Si supponga di voler creare un form di registrazione per nuovi utenti, e di aver precedentemente creato e salvato col nome "Anagrafica" il gruppo di blocchi presentati nell'esempio 2. A questo punto non resta che aggiungere, ad esempio, un campo per inserire l'e-mail, uno per inserire una password e richiamare infine il file Anagrafica.

Metalinguaggio:
```
START FORM DO "/action_page.php"
LOAD "Anagrafica"
BLOCK "email"
ASK email
EMAIL
BLOCK "password"
ASK Password
PASSWORD
BLOCK SUBMIT
END FORM
```

Ecco cosa bisognerebbe scrivere invece in HTML per ottenere lo stesso risultato, mostrato in figura 3.18:

```
<form action="/action_page.php" method="POST">
<div name="Anagrafica" id="Anagrafica">
    <label for="Name">Nome</label>
    <input type="text" name="Name" id="Anagrafica_Name" required
/>
    <label for="Surname">Cognome</label>
    <input type="text" name="Surname" id="Anagrafica_Surname" required
/>
    <label for="Gender">Sesso</label>
    <input type="radio" name="Gender" id="Anagrafica_Gender1" value="M"
required/>
    <label for="Gender1">M</label>
    <input type="radio" name="Gender" id="Anagrafica_Gender2" value="F"
required/>
    <label for="Gender2">F</label>
</div>
<input type="email" name="email" id="email" required />
<label for="email">email</label>
<label for="password">Password</label>
<input type="password" name="password" id="password" required />
<input type="submit" />
</form>
```

Grazie a questo esempio si iniziano a vedere chiaramente quali vantaggi porta l'utilizzo di questo progetto: un riutilizzo dei blocchi già creati (in questo caso si è scelto un blocco piccolo ma, naturalmente, niente vieta di

Figura 3.18: Esempio 3

riutilizzare blocchi più complessi) e una maggiore velocità nella scrittura del codice, conseguenza della riutilizzazione. Si può anche notare come la parte scritta col metalinguaggio sia più comprensibile ad un utente poco pratico, dato che ricorda più una lingua parlata (l'inglese) rispetto al codice HTML.

3.5 Dal metalinguaggio al documento HTML

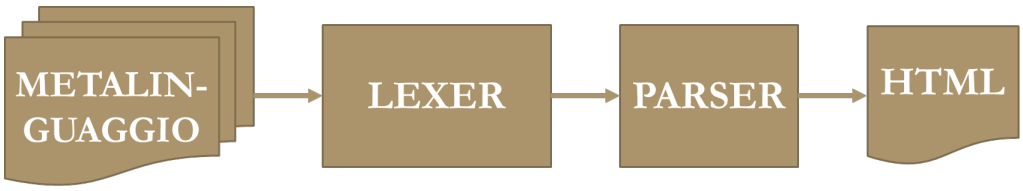

Figura 3.19: Schema logico del processo di trasformazione da metalinguaggio a HTML

Si è già parlato di HTML e di altri linguaggi ad esso collegati; ebbene, l'output che si vuole ottenere dal programma in oggetto è proprio un documento HTML, in modo da poterlo inserire facilmente all'interno di un

documento HTML più complesso.
Per fare ciò, il programma fa uso di due strumenti: un analizzatore lessicale e un analizzatore sintattico. Il primo riceverà in input il file creato servendosi del metalinguaggio e restituirà un output che verrà preso in consegna dal secondo per poi restituire il documento HTML. Di seguito vengono brevemente descritti quali strumenti sono stati utilizzati per ottenere tale risultato.

3.5.1 Analizzatore lessicale (Lexer)

Il lexer (anche noto come scanner) è il primo componente ad intervenire sul file composto col metalinguaggio; esso è stato creato con Lex e il suo compito è quello di scansionare i file in ingresso alla ricerca di parole chiave (o token), definite in una serie di regole al suo interno, e passare in seguito tali token al parser.

> **Lex** è l'abbreviazione di Lexical Analyzer e serve, appunto, per la creazione di analizzatori lessicali, anche noti come scanner o lexer. Esistono diversi tipi di scanner, per questo progetto ne è stato realizzato uno di tipo bottom-up. Nel tempo sono state inoltre create varie versioni di Lex; qui è stata utilizzata l'originale, scelta spesso privilegiata da molti sistemi UNIX ancora oggi e basata sul linguaggio C.

Un lexer è tipicamente diviso in tre sezioni, separate tra loro dal token %%, e sono chiamate sezione delle definizioni, delle regole (anche detta grammatica) e del codice.

3.5.1.1 Sezione delle definizioni

In questa prima parte del file vanno inserite tutte le inizializzazioni volte a far comunicare il lexer con il parser e a permettere una corretta interpretazione del file in ingresso. Per l'elenco completo delle definizioni si rimanda all'appendice A.

3.5.1.2 Sezione delle regole

Le regole definite all'interno dell'analizzatore lessicale fanno uso di espressioni regolari per estrarre i token dal testo. Le regex sono uno strumento molto potente per la ricerca (ed eventualmente modifica) nei testi; esse fanno uso di un insieme di simboli per riuscire a trovare un determinato pattern nel testo che analizzano. Esse sono, tra l'altro, un ottimo esempio di metalinguaggio ma, purtroppo, sono anche un argomento molto complesso con la grave carenza di uno standard per la definizione univoca della loro sintassi, indi per cui si preferisce evitare di entrare nello specifico e lasciare al lettore l'eventuale compito di approfondire l'argomento.

Per la realizzazione di questo progetto, si è preferito mantenere uno scanner alquanto semplice, il cui scopo fosse quello di assegnare alle varie parti del metalinguaggio i token corrispettivi, inoltrare tali token al parser e demandare a quest ultimo tutto il lavoro di creazione del documento HTML.

```
form          {  return FORM;                }
:             {  return DP;                  }
check(box)?   {  return CHECK;               }
[0-9]+        {  yylval.ival = atoi(yytext);
                 return INT;                 }
```

Negli esempi qui sopra, tratti direttamente dal lexer del progetto, si possono distinguere regole semplici, come ad esempio le prime due, che cercano un pattern specifico ("form" e ":") e restituiscono il token associato, e regole più complesse, come la terza, che utilizza un'espressione regolare per restituire il token CHECK sia in caso trovi nel testo la parola "check", sia trovi "checkbox"; l'ultima, infine, non solo restituisce un token INT per ogni numero che trova[3] ma vi associa anche il numero vero e proprio, così da poter essere usato dal parser.

3.5.1.3 Sezione del codice

Nella presente sezione andrebbe inserito eventuale codice C per rendere il lexer capace di svolegere compiti più complessi, ma in questo caso, sempre per la decisione di mantenere questo analizzatore il più semplice possibile, la sezione del codice è vuota e tutto il lavoro è demandato al parser.

3.5.2 Analizzatore sintattico (Parser)

Il parser, creato con Yacc, è sostanzialmente composto da una serie di regole alle quali vengono assegnate delle azioni, scritte in C, in questo progetto volte a produrre blocchi scritti in HTML a seconda di quali token e in quale ordine vengono incontrati. Per poterne realizzare tutte le funzionalità però, l'utilizzo di Yacc non è risultato sufficiente e si è dovuto quindi migrare all'utilizzo di Bison.

[3] Il pattern 0-9 vuol dire "qualsiasi cifra tra 0 e 9" (compresi); il pattern [ ]+ vuol dire "una o più ripetizioni di ciò che appartiene a questo gruppo" (le cifre).

Yacc è l'acronimo di Yet Another Compiler-Compiler ed è uno strumento che genera parser di tipo shift-reduce. Bison è una sua evoluzione, la cui differenza sostanziale da Yacc (e il motivo principale dello scambio) sta nella possibilità di poter creare parser rientranti. Un parser rientrante permette sia ad una funzione di poter richiamare sé stessa, sia il corretto utilizzo di thread e processi figli, mentre utilizzando Yacc (che produce solamente parser non-rientranti) non c'è modo di ritornare al processo padre una volta che i figli hanno terminato il proprio lavoro. Gli analizzatori lessicali vengono di solito accoppiati ad un parser per poter analizzare testi più complessi, cosa che non sarebbe possibile con la sola grammatica (o non senza aumentarne esponenzialmente la complessità). Come loro, anche Yacc si basa sul C, e questo rende molto semplice l'integrazione tra le due parti.

Per meglio capire il funzionamento di un parser, si può pensare allo stato iniziale della grammatica come alla radice di un albero, ai simboli non terminali come ai rami, e agli stati terminali come fossero le foglie.
Come per gli scanner, anche la struttura di un parser è divisa in tre sezioni (definizioni, regole e codice) separate dal token %%

3.5.2.1 Sezione delle definizioni

In questa parte sono presenti non solo le inizializzazioni e le dichiarazioni utili per le azioni da intraprendere per la conversione del metalinguaggio in

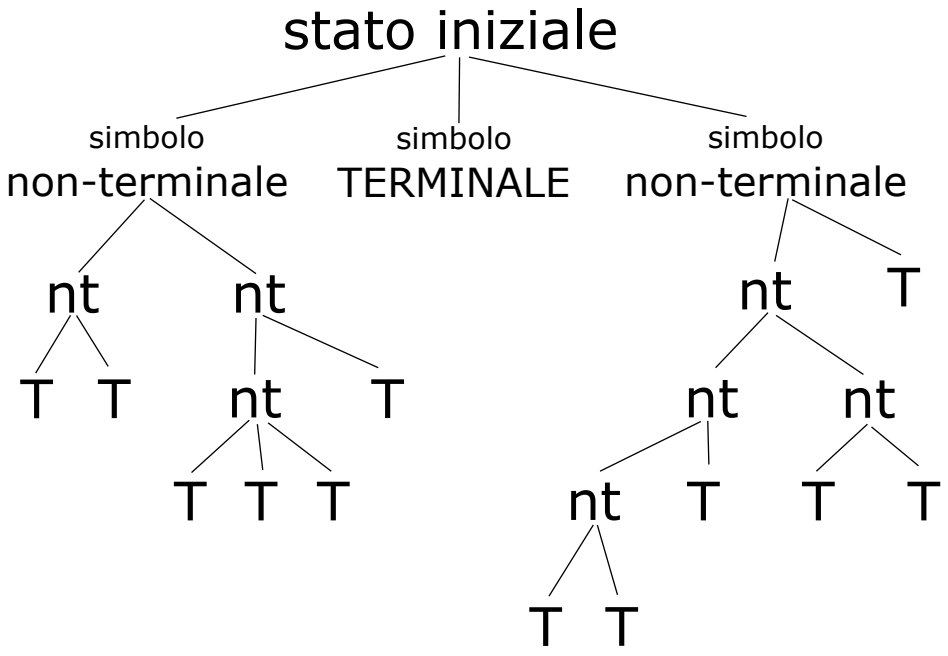

Figura 3.20: Esempio di schema ad albero del passaggio dallo stato iniziale ai simboli terminali, attraverso simboli non-terminali

HTML, ma anche l'elenco dei token terminali ricevuti dallo scanner. Per l'elenco completo si rimanda all'appendice B.

3.5.2.2 Sezione delle regole

Questo è il nócciolo del programma: la grammatica presenta un insieme di regole composte da simboli terminali e non terminali. Anche se possono essere scritti allo stesso modo a livello implementativo, è prassi differenziarli per una più rapida distinzione da parte di un eventuale osservatore umano; si è soliti scrivere i simboli terminali in maiuscolo (ex. TERMINALE) e quelli non terminali in minuscolo (ex. nonterminale). Tale regola è stata naturalmente osservata nella realizzazione di questo lavoro.

A livello sintattico, però, i due tipi di simboli presentano una sostanziale differenza: per ogni simbolo terminale della grammatica si potrà trovare

un corrispondente simbolo terminale nell'elenco dei token provenienti dal lexer, mentre un simbolo non terminale sta invece ad indicare un insieme di simboli terminali e non. Quando il parser quindi incontra uno di questi simboli non terminali, scorre la grammatica alla ricerca di una regola che definisca con cosa può essere sostituito e, se nell'elenco è presente uno o più altri simboli non terminali, il processo si ripete, finché non otterrà una serie di soli simboli terminali. A quel punto, sempre seguendo le regole della grammatica, potrà iniziare la fase di semplificazione (riduzione) fino a raggiungere lo stato iniziale.

Lo stato iniziale di questo progetto è indicato dal simbolo non terminale goal:

```
goal    :    head endls block endls tail endls
        |    block
        ;
```

Questo è il punto di partenza del parser, dal quale si tenterà di ottenere la suddetta lista di simboli terminali (non più espandibili). Formalmente, al nome dello stato iniziale si fanno seguire i due punti (:) seguiti dalla regola che lo espande. Eventuali regole alternative vanno separate con un pipe (|), simbolo dell'operatore logico OR. Il blocco va chiuso mettendo un punto e virgola finale (;). La formattazione qui presentata non serve ai fini del funzionamento del programma (il tutto potrebbe stare su una singola riga) ma, di nuovo, è utile ad un osservatore umano per capire più semplicemente cosa succede. Come si può notare, il simbolo goal può assumere due possibili valori, ma solo uno dei due alla volta.
Si supponga di voler espandere la prima regola e che venga selezionato il simbolo tail come prossimo elemento da espandere; la regola che definisce tale simbolo non terminale è

```
tail    :    END FORM
        ;
```

In questo caso, il parser troverà come unica opzione due simboli terminali; ciò significa che è stata raggiunta la fine di quel ramo dell'albero e non vi è altro lavoro da fare (esplorare); potrà quindi ridurre la regola e continuare ad espandere eventuali rami non ancora visitati.

Il simbolo non-terminale block, presente anch'egli nella prima regola dello stato iniziale goal, si presenta invece sotto questa forma:

```
block  :  %empty
       |  block loaded
       |  block base
       |  block custom
       |  block endls
       ;
```

Salta sicuramente agli occhi il fatto che tale simbolo è presente non solo sulla parte sinistra ma anche tra le regole a destra che lo definiscono, in altre parole, cioè, il simbolo non terminale block è composto in parte da sé stesso. L'azione di richiamare sé stesso prende il nome di ricorsione. Grazie alla ricorsione, un programma è in grado di risolvere un problema complicato affrontandone (e risolvendone) un pezzetto alla volta e andando man mano avanti, mantenendo coscienza di ciò che ha già fatto. È inoltre presente il costrutto %empty, che sta ad indicare una regola vuota, utile per uscire dal ciclo altrimenti infinito creato dalla ricorsione.
Per una maggiore chiarezza e visione d'insieme, si riporta la grammatica ripulita dal codice C/C++:

```
goal          :       head endls block endls tail endls
              |       block
```

```
                ;
head            :       START FORM nameid action method class
                ;
action          :       %empty
                |       SEND DA STRING DA
                ;
method          :       %empty
                |       THROUGH POST
                |       THROUGH GET
                ;
tail            :       END FORM
                ;
block           :       %empty
                |       block loaded
                |       block base
                |       block custom
                |       block endls
                ;
custom          :       NEW tabs endls contents endls END endls
                ;
contents        :       %empty
                |       contents custom
                |       contents base
                |       contents loaded
                ;
base            :       BLK tabs bname endls attribute endls question endls answer endls tabs
                ;
bname           :       %empty
                |       DA STRING DA
                ;
question        :       ASK STRING tabs
                ;
answer          :       tabs anstype tabs
                ;
attribute       :       %empty
                |       attribute hidden
                |       attribute REQUIRED
                |       attribute OPTIONAL
                ;
hidden          :       when STRING EQ STRING
                ;
when            :       SHOW IF
                |       SHOWIF
                ;
anstype         :       CHECK checklist
                |       AREA tabs
                |       DROP tabs droplist
                |       RADIO radiolist
                |       TEXT tabs txt
                ;
txt             :       %empty
                |       TA STRING TC
                |       STRING
                ;
checklist       :       QA STRING tabs cfill labend c QC
                |       STRING tabs cfill labend c
                |       QA tabs cfill def labend c QC
                |       tabs cfill def labend c
                ;
c               :       %empty
                |       c OR STRING tabs cfill labend
                |       c OR tabs cfill def labend
                ;
radiolist       :       QA STRING tabs rfill labend r QC
                |       STRING tabs rfill labend r
                |       QA tabs rfill def labend r QC
                |       tabs rfill def labend r
                ;
r               :       %empty
                |       r OR STRING tabs rfill labend
                |       r OR tabs rfill def labend postdef
                ;
postdef         :       %empty
                |       postdef OR STRING tabs rfill labend
                ;
droplist        :       QA STRING tabs d QC
                |       STRING tabs d
                |       QA tabs sel d QC
                |       sel d
                ;
d               :       %empty
                |       d OR STRING tabs
```

```
                |       d OR tabs sel postsel
                ;
postsel         :       %empty
                |       postsel OR STRING tabs
                ;
sel             :       TA STRING TC
                ;
def             :       TA STRING TC
                ;
loaded          :       LOAD DA STRING DA endls tabs
                ;
nameid          :       %empty
                |       DA STRING DA DA STRING DA
                |       DA STRING DA
                ;
class           :       %empty
                |       ADOPT class
                |       class STRING
                ;
cfill           :       %empty
                ;
labend          :       %empty
                ;
dfill1          :       %empty
                ;
rfill           :       %empty
                ;
tabs            :       %empty
                ;
endls           :       %empty
                |       endls ENDL
                |       ENDL
                ;
```

3.5.2.3 Sezione del codice

In questa sezione è stata inserita la funzione C main(), che è il punto d'ingresso del programma. Essa si occupa di aprire in lettura il file composto col metalinguaggio che gli verrà indicato al momento del lancio del programma e, eventualmente, anche aprire in scrittura un secondo file (sempre indicatogli dall'utilizzatore) dove inserirà il documento HTML risultante. Fatto questo, si potrà procedere con la vera e propria fase di parsing.

Per comodità, sia qui che nella precedente sezione si è fatto uso di una combinazione dei linguaggi C e C++, dato che quest ultimo permette di svolgere certi compiti con più facilità rispetto al suo antenato.

C/C++ sono due linguaggi di programmazione per alcuni aspetti simili ma per altri molto diversi. Simili perché, in fin dei conti, C++ deriva dal C e voleva essere un suo miglioramento; diversi per molti aspetti: ad esempio, C è di natura procedurale mentre C++ è orientato agli oggetti. Questa differenza alla base del loro pensiero ha portato col tempo a generare delle incompatibilità tra i due linguaggi, ma molte sezioni sono ancora in grado di comunicare tra loro.

3.6 Interazione tra i vari componenti

Il programma realizzato riceve in ingresso un file di testo, lo analizza, trasforma, e infine manda in output il risultato.
Il primo ad operare sul file è l'analizzatore lessicale, che lo analizza e marchia i vari token sulla base delle proprie espressioni regolari; in seguito, l'analizzatore sintattico prenderà questo elenco di token e cercherà di farli combaciare con le regole presenti nella propria grammatica, eseguendo del codice C/C++ che produrrà frammenti di codice HTML man mano che le produzioni vengono ridotte.

Il programma è facilmente espandibile per l'aggiunta di nuove funzionalità. Tuttavia, il progetto è stato realizzato su piattaforma linux quindi, in caso di modifica su altre piattaforme, i comandi potrebbero variare.

UNIX/Linux I sistemi operativi Linux si basano su UNIX; è importante far notare che tutti i programmi utilizzati nella realizzazione di questo lavoro sono gratuiti e open-source, dando quindi a tutti la possibilità sia di utilizzare semplicemente il prodotto finito, sia di utilizzare i sopra-descritti programmi per apportare delle modifiche al prodotto al fine di espanderlo/personalizzarlo.

Al momento della compilazione dell'analizzatore lessicale, viene creato un file addizionale, solitamente chiamato `lex.yy.c`, nel quale è presente una versione in C delle regole presenti nel lexer e che servirà quindi al programma per riconoscere i token. In caso di modifica al lexer, bisogna aggiornare il suddetto file C per poter utilizzare le nuove aggiunte, lo si fa tramite il comando

```
lex meta.lex
```

L'analizzatore sintattico invece genera due file ausiliari, solitamente chiamati `y.tab.c` e `y.tab.h`, che permettono la comunicazione tra lexer e parser e, come nel caso precedente, contengono la grammatica del parser e permettono al programma di utilizzarla. Il comando per aggiornare tali file, a seguito di eventuali modifiche, è

```
yacc -d m2h.y
```

Una volta modificato uno dei due tra lexer e parser, è necessario aggiornare anche l'eseguibile del programma, e lo si fa eseguendo il comando

```
g++ y.tab.c lex.yy.c -lfl -o m2h
```

L'appendice C offre una copia del file Makefile per la generazione automatica di tali comandi. Una volta ottenuto l'eseguibile, esso può essere lanciato da shell col comando

```
./m2h file_di_input file_di_output
```

Il file di output può essere omesso e in tal caso il risultato sarà proiettato a schermo.

Capitolo 4
Sviluppi futuri

Nonostante quanto visto finora, il programma realizzato è lontano dal considerarsi completo, viste anche le continue innovazioni, specialmente in campo informatico.

Una peculiarità di questo progetto è che il suo codice può essere visto da chiunque e, essendo stato realizzato con programmi non a pagamento, è possibile apportare delle modifiche in base ai propri bisogni.

Una delle prime aggiunte di cui il software potrebbe giovare è sicuramente la possibilità di maneggiare file multimediali, come immagini e audio/video. Non è infatti raro trovare tali file in sondaggi creati per ricerche di mercato.

Un ulteriore passo che sicuramente può rivelarsi utile è la realizzazione di un'interfaccia grafica per la creazione dei blocchi, di modo da poter agevolare soprattutto gli utenti che utilizzano dispositivi portatili, quali smartphone e tablet.

Sarebbe inoltre utile allestire uno spazio online dedicato per il conservamento e la condivisione di blocchi già creati. Una sorta di libreria attraverso la quale gli utenti meno esperti possano attingere dai lavori di quelli più esperti.

Capitolo 5
Conclusioni

Alla luce di quanto visto, si è positivi nell'affermare di essere riusciti nell'intento di creare un sistema aperto che sia in grado di consegnare al proprio utilizzatore componenti che siano al contempo facilmente riutilizzabili e scalabili, ma anche di qualità. Qualità che non si limita solamente al rispetto degli standard imposti dal consorzio W3C per documenti HTML, ma che comprende anche una corretta catalogazione delle informazioni, per una efficiente memorizzazione e conseguente facilità di recupero di queste ultime dalle basi di dati. Il fatto che questo sistema sia aperto non pone vincoli o limitazioni all'utilizzatore ma, anzi, invita all'apporto di modifiche/migliorie e allo scambio di informazioni e conoscenza, come spesso avviene in ambienti open-source.

Attraverso questo progetto si è inoltre creata la base per un sistema object-oriented. Tale sistema separa infatti le diverse aree di lavoro, permettendo così a differenti tipi di utenti di concentrarsi su aspetti diversi della creazione dei suddetti form. Si possono distinguere sostanzialmente 3 macro-aree nelle quali operare:

- un livello più basso, al quale opereranno prevalentemente esperti e programmatori, i quali potranno creare nuovi blocchi con funzionalità di base e si accerteranno che la comunicazione tra i vari componenti del

sistema avvenga correttamente (non solo la comunicazione tra lexer e scanner, del quale ci si è occupati in questo lavoro, ma anche altri tipi di comunicazione come, ad esempio, quella tra form e database).

- un livello intermedio, nel quale agirà prevalentemente il cosiddetto "utente medio", fruitore principale del servizio qui proposto; non necessariamente una persona che conosce il funzionamento di HTML (o di quant'altro si trovi realmente dietro le quinte per far sì che tutto funzioni) e non è nemmeno richiesto che egli lo sappia, nella stragrande maggioranza dei casi infatti gli basterà utilizzare quanto gli viene messo a disposizione dagli utenti che operanno al livello sottostante per creare un form a lui congeniale.

- un livello più alto, al quale verranno creati i componenti di contorno al form, come ad esempio i fogli di stile CSS e le classi, che potranno essere in seguito richiamate indipendentemente all'interno di uno o un altro form.

Appendices

Il file meta.lex

```
%option noyywrap
%option caseless
%option yylineno

%{
#include <iostream>
#include "y.tab.h"

int line_num = 1;
int inout = 0;
%}

%x trapici
%x outapici
%x domanda
%x predef
%x LEXING_ERROR
%x COMMENTS

ASK    ask:?
SHOWIF    ("show if"|"showif")

%%

[ \s\t]     ;

adopt    { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return ADOPT    ; }
all        { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return ALL    ; }
and        { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return AND    ; }
as         { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return AS    ; }
{ASK}    { BEGIN(domanda);
           char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return ASK    ; }
block    { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return BLK    ; }
do         { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return DO    ; }
end        { char *res = new char[strlen(yytext) + 1];
           strcpy(res, yytext);
           yylval.sval = res;
           return END    ; }

form    { char *res = new char[strlen(yytext) + 1];
```

```
          strcpy(res, yytext);
          yylval.sval = res;
          return FORM    ; }
group    { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return GROUP    ; }
    /*
if         { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return IF    ; }
    */
load     { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return LOAD    ; }
new        { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return NEW    ; }
only     { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return ONLY    ; }
send     { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return SEND    ; }
set        { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return SET    ; }
send     { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return SEND    ; }
    /*
show     { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return SHOW    ; }
    */
{SHOWIF} { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return SHOWIF    ; }
start    { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return START    ; }
through    { char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return THROUGH; }

\"         {
        /*
          if(inout==0)
            BEGIN(trapici);
          if(inout==1)
            BEGIN(outapici);
        */
          return DA    ; }

:         { return DP    ; }
\n             { ++line_num;
          return ENDL    ; }
=         { return EQ    ; }
\|         { return OR    ; }
\[         { return QA    ; }
\]         { return QC    ; }
\(         {
        /*
          BEGIN(predef) ;
        */
```

```
          return TA    ; }
\)        { return TC    ; }

get        { yylval.sval = "get";
          return GET    ; }
post    { yylval.sval = "post";
          return POST    ; }


required    { yylval.sval = "required";
              return REQUIRED;}
optional    { yylval.sval = "optional";
              return OPTIONAL;}
hidden        { yylval.sval = "hidden";
              return HIDDEN    ; }
advanced    { yylval.sval = "advanced";
              return ADVANCED;}

    /* I seguenti valori sono stati castati per compatibilità con la map nel parser */

    /*    ATTENZIONE!                                              */
    /*    Bisogna passare il valore di *tutte* le parole chiave,    */
    /*    altrimenti l'utente non le potrà usare                     */
    /*    (problema per i form in inglese)                          */
    /*        Soluzione alternativa:                                 */
    /*     forzare le parole chiave a inizio riga "^"               */
    /*    bisogna fare il trim di eventuali spazi/tab "[\s\t]*"    */
    /*     soluzione che non va bene per i termini REQUIRED etc.    */

submit        { yylval.sval = "submit";
              return SUBMIT;             }
area        { yylval.sval = "area";
              return AREA;                 }
textarea    { yylval.sval = "textarea";
              return AREA;                 }
longtext    { yylval.sval = "longtext";
              return AREA;                 }
radio        { yylval.sval = "radio";
              return RADIO;                }
radiobutton    { yylval.sval = "radiobutton";
              return RADIO;                }
check        { yylval.sval = "check";
              return CHECK;              }
checkbox    { yylval.sval = "checkbox";
              return CHECK;              }
text        { yylval.sval = "text";
              return TEXT;                }
shorttext    { yylval.sval = "shorttext";
              return TEXT;                }
select        { yylval.sval = "select";
              return DROP;                 }
drop        { yylval.sval = "drop";
              return DROP;                 }
dropdown    { yylval.sval = "dropdown";
              return DROP;                 }

(button|color|date|datetime|datetimelocal|email|fileupload|hidden|image|month|number|password|range
|reset|search|submit|time|url|week)    {    char *res = new char[strlen(yytext) + 1];
               strcpy(res, yytext);
              yylval.sval = res;
              return OTHERINPUTTYPES;
            }

    /* un approccio migliore, probabilmente, è quello di separare ogni riga in condition (vedi <domanda>) differenti,
    soprattutto in previsione di aggiunte di feature future,
    così da limitare le parole chiave ad essere parole chiave solo in quel punto,
     ed essere trattate come parole normali nelle altre righe */

<predef>[a-zA-Z0-9_\-\ ]+        {
        // we have to copy because we can't rely on yytext not changing underneath us:
          char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return STRING;
        }
<predef>\)        BEGIN(INITIAL);
```

```
<domanda>([^\r\n]|" ")+    {
        // we have to copy because we can't rely on yytext not changing underneath us:
          char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return STRING;
        }
<domanda>\n        BEGIN(INITIAL);


<trapici>[^"]+      {
        // we have to copy because we can't rely on yytext not changing underneath us:
          char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
            inout = 1;
          return STRING;
        }
<trapici>\"        BEGIN(INITIAL);

<outapici>\" {inout = 0;
        }
<outapici>[^"] BEGIN(INITIAL);

    /* ATTENZIONE! */
    /* Per ora è necessario separare i termini inseriti dall'utente
     dai simboli token per evitare di includerli. */
    /* Questo permette all'utente di inserire qualsiasi stringa egli voglia
     ma può portare ad un errore di compilazione a seconda di dove si omette lo spazio */
    /* *Fondamentale* separare l'ultimo carattere delle stringhe dai token che seguono! */
    /* ex:  [A|B]       ERRORE!
            [A |B]      ERRORE!
            [A|B ]      Sintatticamente corretto ma genera un'unica opzione chiamata A|B (quindi ERRORE!)
            [ A|B ]     come sopra (ERRORE!)
            [ A | B ]   OK :) Opzione suggerita
            [A |B ]     also ok :) ma meglio la precedente,
             per evitare fraintendimenti e vedere al volo eventuali errori
    */
[a-zA-Z0-9][^" "\n\r]*    {
        // we have to copy because we can't rely on yytext not changing underneath us:
          char *res = new char[strlen(yytext) + 1];
          strcpy(res, yytext);
          yylval.sval = res;
          return STRING;
        }

[0-9]+      { yylval.ival = atoi(yytext);
          return INT    ; }

\/\*    { // start of a comment: go to a 'COMMENTS' state.
        BEGIN(COMMENTS);
    }

<COMMENTS>\*\/    {   // end of a comment: go back to normal parsing.
            BEGIN(INITIAL);
        }
<COMMENTS>\n      { ++line_num; }   // still have to increment line numbers inside of comments!
<COMMENTS>.       ;   // ignore every other character while we're in this state

.                 { BEGIN(LEXING_ERROR); yyless(1); }

<LEXING_ERROR>.+  { fprintf(stderr,
                            "Invalid character '%c' found at line %d,"
                            " just before '%s'\n",
                            *yytext, yylineno, yytext+1);
                  exit(1);
                }

%%
```

Il file m2h.y

```
%{
        #include <iostream>
        #include <fstream>
        #include <string>
        #include <sys/wait.h>
        #include <stdlib.h>
        #include <unistd.h>
        #include <stdio.h>
        #include <string.h>
        #include <sys/types.h>
        #include <map>
        #include <vector>

        extern int yylex();
        extern FILE *yyout;
        extern int line_num; //utilizzato per indicare la linea dell'errore

        bool tofile=0;                //output su schermo/file; mi sa non utilizzato
        bool isreq = true;        //serve per aggiungere "required" ai vari input
        bool ishid = false; // is HIDDEN
        bool shif = false;        // SHOW IF
        bool noglobal = false;        //utilizzato da ADOPT ONLY
         per evitare che un blocco adotti eventuali classi globali.

        int maincount=0;        //numero di blocchi, temporaneamente usato per IDs,
         forse con reentrant si potrà rimuovere
        int subcount=0;                //numero blocchi dentro un gruppo, al momento non è utilizzato
        int blockcount=0;        //same as maincount?
        int groupcount=0;        //numero gruppi (e quindi <div>)
        int nestlevel=1;        //serve per indentare i blocchi; parte da uno così è tutto indentato rispetto a <form>
        int anscount = 0;        //numero opzioni risposte chiuse;
        utilizzato negli ID delle risposte "..._choice_anscount"

        std::map <std::string, std::string> globalclasses; //ALL input_type ADOPT classlist --> <input_type, classlist>

        std::string blockname, blockid;        //di volta in volta contengono name e ID di ogni blocco
//        std::string thisid; //usato per creare l'ID delle varie scelte delle risposte multiple
        std::string file2read, file2write;        //file di input e output
        std::string label, label1, label2; //utilizzati per creare correttamente le label degli input
        std::string strwspaces;        //usata per concatenare i vari token STRING separati da spazi,
         dato che le condition nello scanner non funzionano come dovrebbero
        std::string blocktype, localclasses; //utilizzati per attribuire le classi ai blocchi
        std::string shif_id2chk, shif_val2cmpr;

        std::vector<std::string> pathchain;        //vettore contenente i nomi dei file aperti;
        utilizzato per stampare percorso file e ID univoci

        void ReadNewFile(std::string file2read);
        void yyerror(const char *s);

        std::streambuf *psbuf, *backup; //
        std::ofstream filestr;
%}

%union {
        int ival;
        char *sval;
}

%token <sval>START <sval>FORM <sval>END
%token <sval>DO <sval>THROUGH <sval>GET <sval>POST
%token <sval>NEW <sval>GROUP <sval>ENDGRP <sval>LOAD
%token <sval>ASK <sval>BLK
```

```
%token <sval>ALL <sval>ADOPT <sval>ONLY <sval>SEND
%token <sval>OPTIONAL <sval>REQUIRED <sval>ADVANCED
%token <sval>HIDDEN /*<sval>SHOW <sval>IF*/ <sval>SHOWIF
%token <sval>SET <sval>AS <sval>AND
//%token AREA CHECK DROP RADIO SUBMIT TEXT
%token DA DP ENDL EQ OR QA QC TA TC


%token  <ival>INT
%token  <sval>STRING
%token  <sval>OTHERINPUTTYPES
%token         <sval>AREA <sval>CHECK <sval>DROP <sval>RADIO <sval>SUBMIT <sval>TEXT
%type        <sval>inputtype
//%type         <sval>words

%%

goal        :                {
                                        std::cout << std::endl;
                                }
                        head endls gclasses endls block endls tail endls
                                {        std::cout.rdbuf(backup); /*reset to standard output again */
                                        std::cout << std::endl << "COMPLETE" << std::endl;
                                }
                |                //{         std::cout.rdbuf(psbuf); /*assign streambuf to cout*/ }
                        block
                                {        std::cout.rdbuf(backup);
                                        std::cout << "<!-- Parsing file "
                                        << file2read << " terminato correttamente -->"
                                        <<std::endl;
                   // std::cout.rdbuf(backup); /*reset to standard output again */
                                        // std::cout << std::endl << "block loaded correctly" << std::endl;
                                }
                ;
/* * * * * *
 * H E A D *
 * * * * * */
head        :        START FORM
                        {
                                        std::cout << "<form";
                                }
                           fnamid config endls
                                {
                                        std::cout << ">" << std::endl;
                                }
                ;
fnamid  :   %empty
                                 {
                                        std::cout << " name=\"form" << "\" id=\"form" <<"\"";
                                }
                |        DA STRING DA DA STRING DA
                                {
                                        std::cout << " name=\""<< $2 << "\" id=\"" << $5 <<"\"";
                                }
                |        DA STRING DA
                                {
                                        std::cout << " name=\""<< $2 << "\" id=\"" << $2 <<"\"";
                                }
                ;

config  :   %empty
        |   config action
        |   config method
        |   config class
                |        config expert
        |   config endls
        ;
action        :        %empty
                |        DO DA STRING DA
                {
                                        std::cout << " action=\""<< $3 <<"\"";
                                }
                ;
method        :        %empty
                {
                                        std::cout << " method=\"post\"";
                                }
                |        THROUGH POST
                {
                                        std::cout << " method=\"post\"";
                                }
```

```
                |        THROUGH GET
                {
                                        std::cout << " method=\"get\"";
                                }
                ;
gclasses:   %empty
                |        gclasses ALL inputtype ADOPT DA words DA endls
                        {
                                globalclasses.insert(std::pair <std::string, std::string> ($3, strwspaces));
                        }
        ;
inputtype:        AREA {$$=$1;} | CHECK {$$=$1;} | DROP {$$=$1;} | RADIO {$$=$1;} | SUBMIT {$$=$1;} | TEXT {$$=$1;}
| OTHERINPUTTYPES {$$=$1;}
        ;
expert        :        ADVANCED words
                ;
tail        :        %empty
                        {
                                        std::cout << std::endl << "</form>";
                                }
/*         Decidere se END FORM ci vuole o no.. al momento disattivato.
        Trovare un modo per farlo opzionale perché così genera errore */
                |        END FORM
                        {
                                        /*        questo può andare anche dentro tail volendo
                                                (magari si può rendere tail obbligatorio) */
                                        std::cout << std::endl << "</form>";
                                }
                ;
/* * * * * * *
 * B L O C K *
 * * * * * * */
block        :        %empty
                |          block loaded
                |        block base
                |        block custom
                |        block endls
                ;
loaded        :        LOAD DA STRING DA endls
/*
        ATTENZIONE!
        non accetta filename con spazi!
*/
                                {
                                        std::string file2read = $3;
                                        std::string loading = "./m2h " + file2read + " " + file2write;
                                        system(loading.c_str());
                                        //pathchain.pop_back();
                                }
                ;
custom        :        NEW GROUP
                {
                    nestlevel++;
                                        maincount++;
                                        subcount++;
                                        /* fix IDs.. is ‘if(level==1)maincount++‘ ok? */
                                        std::cout << "<div" ;
                }
                                        /* aggiungere attributi anche qui? */
                         gnamid attribute endls
                                {
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                         std::cout << ">" << std::endl ;
                                }
                         block
                         endls
                         END GROUP
                                {
                                        subcount--;
                                        nestlevel--;
                                }
                          endls
                                 {
                                        std::cout << "</div>" << std::endl;
                                }
                ;
gnamid  :   %empty
                                 {
                                        std::cout << " name=\"group_"<< maincount
                                        << "\" id=\"group_" << maincount <<"\"";
```

```
                        }
                |       DA STRING DA DA STRING DA
                        {
                                std::cout << " name=\""<< $2 << "\" id=\"" << $5 <<"\"";
                        }
                |       DA STRING DA
                        {
                                std::cout << " name=\""<< $2 << "\" id=\"" << $2 <<"\"";
                        }
                ;
/*contents:       %empty
                |       contents custom
                |       contents base
                |       contents loaded
                |       contents endls
                ;
*/
base            :       BLK
                {
                        //std::cout << "new block found" << std::endl;
                    maincount++;
                      }
                        bnamid endls {/*std::cout<<std::endl<<"sono qui"<<std::endl;*/} attribute endls
                                {
                                }
                        class endls
                                {
                                }
                        ansorsub
                ;
ansorsub:       question endls answer endls
                                {
                                        if(shif==true)
                                        {
                           std::cout        << "<script>\n"
                                            << "\t$(function() {\n"
                                            << "\t\t$('#" << shif_id2chk <<"').change(function(){\n"
                                            << "\t\t\t$('#" << blockid << "').hide();\n"
                                            << "\t\tif($(this).val()=='" << shif_val2cmpr << "')\n"
                                            << "\t\t\t$('#" << blockid << "').show();\n"
                                            << "\t\t});\n"
                                            << "\t});\n"
                                            << "</script>\n";
                                        shif = false;
                                        }
                                        std::cout <<"<br/>"<< std::endl;
                                        /*         questo <br/> è stato aggiunto per la discussione,
                                                manda a capo dopo ogni blocco. con CSS forse non necessario.
                                                endl invece dovrebbe rimanere */
                                }
                |       SUBMIT
                                {
                                        std::cout << "<input type=\"submit\"";
                                        std::cout << " name=\"" << blockname <<"\" id=\"" << blockid <<"\"";
                                }
                        buttxt
                                {
                                        std::cout << " />";
                                }
                ;
buttxt          :       %empty
                |       STRING
                                {
                                        std::cout << " value=\"" << $1 << "\"";
                                }
                ;
bnamid          :       %empty
                         {
                    blockname = "block"; blockname += std::to_string(maincount);
                    blockid = "block"; blockid += std::to_string(maincount);
                                        //std::cout << " name=\"block_"<< maincount
                                        << "\" id=\"block_" << maincount <<"\"";
                                }
                |       DA STRING DA DA STRING DA
                                {
                    blockname = $2;
                    blockid = $5;
                                        //std::cout << " name=\""<< $2 << "\" id=\"" << $5 <<"\"";
                                }
                |       DA STRING DA
                                {
```

```
                    blockname = $2;
                    blockid = $2;
                                    //std::cout << " name=\""<< blockname << "\" id=\"" << blockid <<"\"";
                                }
                ;
attribute:        %empty
                                {
                                    isreq=true;
                                    ishid=false;
                    //std::cout << " required";
                                }
                |       attribute HIDDEN when
                                {
                                    isreq=false;
                                    ishid=true;
                                }
                |       attribute REQUIRED
                                {
                                    isreq=true;
                                    ishid=false;
                    //std::cout << " required";
                                }
                |       attribute OPTIONAL
                                {
                                    isreq=false;
                                    ishid=false;
                    //std::cout << " required";
                                }
                ;
when        :         %empty
                |       SHOWIF STRING EQ STRING
                                {
                                    shif=true;
                                    shif_id2chk = $2;
                                    shif_val2cmpr = $4;
                                    std::cout << "mostrare solo se blocco " << $2 << " = " << $4 << std::endl;
                                }
                ;
class        :        %empty
                |        ADOPT DA words DA
                                {
                                    localclasses = strwspaces;
/*                                    std::cout << " class=\"";
                                    // something like cout << gclassmap(type)
                                    std::cout << " " << $3 << "\"";
                                    //wut?
                                    //std::cout << " class=\"";} class {std::cout << "\""; */
                                }

                |       ADOPT ONLY DA words DA
                                {
                                    noglobal = true;
                                    localclasses = strwspaces;
//                                    std::cout << " class=\"" << $4 << "\"";
                                }
                ;
question:        %empty
/*
        Qui l'inclusione degli spazi viene gestita correttamente nello scanner
        per cui STRING è a posto così, non bisogna fare nulla
*/
        |   ASK STRING
                                {
                    std::cout << "<label for=\"" << blockid << "\"";
                                    if(ishid)
                                            std::cout << " style=\"display:none\"";
                                    std::cout << ">" << $2 << "</label>" << std::endl;
                                    /* OLD ver
                    label1 = "\t<label for=\"";
                                    label = $3;
                                    label2 = "\">"+ label + "</label>";
                    */
                                }
                ;
answer        :       //
                {
                    //std::cout << "\t<div class=\"answer\">" << std::endl;
                }
                        anstype         //{       std::cout << "\t</div>" << std::endl;}
```

```
                ;
anstype         :       CHECK {         blocktype="check";        } checklist
                |         AREA
                                {
                                         blocktype="area";
                                        std::cout << "<textarea";
                                         std::cout << " name=\""<<blockname<<"\"";
                                         std::cout << " id=\""<<blockid<<"\"";
                    }
                insertclasses xtra
                    {
                        if(isreq)
                            std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                        std::cout << "></textarea>" << std::endl;
                                }
                    |        DROP
                                 {
                                         blocktype="drop";
                                        std::cout << "<select";
                                        std::cout << " name=\"" << blockname << "\"";
                                        std::cout << " id=\"" << blockid << "\"";
                                }
                                        insertclasses
                                {
                        if(isreq)
                            std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << ">" << std::endl;
                                        std::cout << "<option value=\"\"></option>" << std::endl;
                                }
                            droplist
                                {
                                        std::cout << "</select>" << std::endl;
                                        anscount=0;
                                }
                    |        RADIO       { blocktype="radio"; }        radiolist
                    |        TEXT  //DA STRING DA
                                {
                                        /* OLD ver
                        std::cout << label1 << $4 << label2 << std::endl;
                        */
                                         blocktype="text";
                                        std::cout << "<input type=\"text\"";
                                         std::cout << " name=\""<<blockname<<"\"";
                                         std::cout << " id=\""<<blockid<<"\"";
                                }
                            insertclasses xtra
                                {
                        if(isreq)
                            std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << " />" << std::endl;
                                }
            |   OTHERINPUTTYPES
                    {
                                         blocktype=$1;
                                        std::cout << "<input type=\"" << $1 << "\"";
                                         std::cout << " name=\""<<blockname<<"\"";
                                         std::cout << " id=\""<<blockid<<"\"";
                                }
                            insertclasses xtra
                                {
                        if(isreq)
                            std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << " />" << std::endl;
                    }
                    ;
xtra            :        %empty
                    |        TA
                                 {
                                        std::cout << " value=\"";
                                }
                            words TC
                                 {
```

```
                                        std::cout << "\"";
                                }
                |                 {
                                        std::cout << " placeholder=\"";
                                }
                        words
                                 {
                                        std::cout << "\"";
                                }
                ;
checklist:        QA copt QC {anscount=0;}
                ;
copt        :         %empty
                |
                                {
                                        std::cout << "\t<input type=\"checkbox\"";
                                }
                        words insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    /* checklist can't be required! a script is needed to achieve this */
                                        /* if(isreq)
                        std::cout << " required"; */
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid
                                        <<"\">" << strwspaces <<"</label>" << std::endl;
                                }
                |        copt OR words
                                {
                                        std::cout << "\t<input type=\"checkbox\"";
                                }
                        insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    /*if(isreq)
                        std::cout << " required";*/
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                         << strwspaces <<"</label>" << std::endl;
                                }
                |        TA words TC
                                {
                                        std::cout << "\t<input type=\"checkbox\"";
                                }
                        insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    /*if(isreq)
```

```
                        std::cout << " required";*/
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << " checked";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                         << strwspaces <<"</label>" << std::endl;
                                }
                |        copt OR TA words TC
                                {
                                        std::cout << "\t<input type=\"checkbox\"";
                                }
                        insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    /*if(isreq)
                        std::cout << " required";*/
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << " checked";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                         << strwspaces <<"</label>" << std::endl;
                                }
                ;
radiolist:       QA ropt rchk ropt QC {anscount=0;}
                ;
ropt        :        %empty
                |        ropt OR
                |        words
                                {
                                        std::cout << "\t<input type=\"radio\"";
                                }
                        insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    if(isreq)
                        std::cout << " required";
                                        std::cout << ">";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                        << strwspaces <<"</label>" << std::endl;
                                }
                |        ropt OR words
                                {
                                        std::cout << "\t<input type=\"radio\"";
                                }
                        insertclasses
                                {
                                        anscount++;
```

```
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    if(isreq)
                        std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                        << strwspaces <<"</label>" << std::endl;
                                }
                ;
rchk            :         %empty
                |          TA words TC
                                {
                                        std::cout << "\t<input type=\"radio\"";
                                }
                        insertclasses
                                {
                                        anscount++;
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                        thisid += std::to_string(anscount);
                                        std::cout << " name=\"" << blockid << "\"";
                                        std::cout << " id=\"" << thisid << "\"";
                                        std::cout << " value=\"" << strwspaces << "\"";
                    if(isreq)
                        std::cout << " required";
                                        if(ishid)
                                                std::cout << " style=\"display:none\"";
                                        std::cout << " checked";
                                        std::cout << ">";
                                        std::cout << std::endl;
                                }
                        //indent
                                {
                                        std::string thisid; thisid = blockid; thisid += "_choice";
                                         thisid += std::to_string(anscount);
                                        std::cout << "\t<label for=\"" << thisid <<"\">"
                                        << strwspaces <<"</label>" << std::endl;
                                }
                ;
droplist:       QA words
                                {
                                         std::cout << "\t<option value=\"" << strwspaces << "\">"
                                          << strwspaces <<"</option>" << std::endl;
                                }
                        d QC
                |        words
                                {
                                        std::cout << "\t<option value=\"" << strwspaces << "\">"
                                        << strwspaces << "</option>" << std::endl;
                                }
                        d
                |        QA
                                {
                                        std::cout << "\t<option ";
                                }
                        selected
                                {
                                        std::cout <<"</label>" << std::endl ;
                                }
                        d QC
                |
                                {
                                        std::cout << "\t<option " ;
                                }
                        selected
                                {
                                        std::cout << "</label>" << std::endl ;
                                }
                        d
```

```
                ;
d               :         %empty
                |        d OR words
                                {
                                        std::cout << "\t<option value=\"" << strwspaces << "\">"
                                        << strwspaces << "</option>" << std::endl;
                                }
                |        d OR
                                {
                                        std::cout << "\t<option ";
                                }
                        selected
                                {
                                        std::cout <<" </option>" << std::endl ;
                                }
                        postsel
                ;
dfill1          :        %empty
                                {
                                        std::cout ;
                                }
                ;
selected:         TA words TC
                                {
                                        std::cout << "value=\"" << strwspaces << "\" selected>" << strwspaces ;
                                }
                ;
postsel         :        %empty
                |         postsel OR words
                                {
                                        std::cout << "\t<option value=\"" << strwspaces << "\">"
                                        << strwspaces << "</option>" << std::endl;
                                }
                ;
labend          :        %empty
                                {
                                        std::cout << "</label>" << std::endl;
                                }
                ;
words           :       %empty
                |        STRING
                                {
                                        strwspaces = $1;
                                }
                |        words STRING
                                {
                                        strwspaces += " "; strwspaces += $2;
                                        // append $2 to $1 and make that result the production's value
                                        //$$=$1;
                                        //$$=(char *)realloc($$,strlen($$)+strlen($2)+1);
                                        //strcat($$,$1);
                                        //std::cout << $$;
                                }
/* il seguente gruppo di regole serve in caso l'utente inserisca una di queste parole chiave
, altrimenti si pianta tutto.*/
/* MOOOOLTO probabilmente, a sto punto, bisogna farlo per *tutte* le parole chiave */
                |        AREA {strwspaces  = $1;} | CHECK {strwspaces  = $1;} | DROP {strwspaces = $1;}
                | RADIO {strwspaces = $1;} | SUBMIT {strwspaces = $1;} | TEXT {strwspaces = $1;}
                | OTHERINPUTTYPES {strwspaces = $1;}
                |        words AREA {strwspaces += " "; strwspaces += $2;}
                | words CHECK {strwspaces += " "; strwspaces += $2;}
                | words DROP {strwspaces += " "; strwspaces += $2;}
                | words RADIO {strwspaces += " "; strwspaces += $2;}
                | words SUBMIT {strwspaces += " "; strwspaces += $2;}
                | words TEXT {strwspaces += " "; strwspaces += $2;}
                | words OTHERINPUTTYPES {strwspaces += " "; strwspaces += $2;}
                ;
insertclasses:        {
                                        std::cout << " class=\"";
                                        if(noglobal==false)
                                        {
                                                if(globalclasses.count(blocktype) == 1)
                                                // Alternately my_map.find( key ) != my_map.end() works too
                                                {
                                                        std::cout << globalclasses[blocktype];
                                                        if(localclasses.length() > 0) std::cout << " ";
                                                }
```

```
                                        noglobal=false;
                                }
                                std::cout << localclasses;
                                std::cout << "\"";
                                localclasses.clear();
                        }
                ;
indent          :       %empty
                        {
                         for(int i=0;i<nestlevel;i++)
                                std::cout << "\t";
                        }
                ;
endls           :       %empty
                |          endls
                        {
                                //std::cout<<"sono in endls, ";
                        }
                          ENDL
                |          ENDL
                ;

%%


std::ofstream results;
        extern int yyparse();
        extern FILE *yyin;
void PrintFilesChain();

int main(int argc, char** argv)
{
        backup = std::cout.rdbuf(); //back up cout's streambuf
        yyin = fopen(argv[1], "r");
        if (!yyin)
        {
                std::cout << "I can't open the indicated file!" << std::endl;
                return -1;
        }
/*                std::streambuf *psbuf, *backup;
                std::ofstream filestr;
*/
        file2read = argv[1];
        if(argc==3)
        {

   if(tofile==0)
{     filestr.open(argv[2], std::fstream::app);
        //tofile=1;
        file2write=argv[2];
}
                psbuf = filestr.rdbuf();        //get file's streambuf
                std::cout.rdbuf(psbuf);                //assign streambuf to cout

/*
                yyout = fopen(argv[2], "w");
            std::ofstream out("out.txt");
            std::streambuf *coutbuf = std::cout.rdbuf(); //save old buf
            std::cout.rdbuf(out.rdbuf()); //redirect std::cout to out.txt!
*/
        }
std::cout << std::endl << "<!-- Parsing file " << file2read << " in corso... -->" << std::endl;
        do {
                yyparse();
        } while (!feof(yyin));

//        std::cout.rdbuf(backup);
        filestr.close();

        return 0;
}


/*
void yyerror(char *s) {
        std::cout << "EEK, parse error on line " << line_num << "!  Message: " << s << std::endl;
        exit(-1);
}*/
void yyerror(const char *s)
```

```
{
        std::cout.rdbuf(backup);
        std::cout << std::endl << "Parsing error on file \"" << file2read <<   "\", at line " << line_num;
        std::cout << std::endl << "Message: " << s << std::endl;
        remove(file2write.c_str());
        // might as well halt now:
        exit(-1);
}}
```

Il file Makefile

```
y.tab.c y.tab.h: m2h.y
	yacc -d m2h.y

lex.yy.c: meta.lex y.tab.h
	lex meta.lex

m2h: lex.yy.c y.tab.c y.tab.h
	g++ -std=c++11 y.tab.c lex.yy.c -lfl -Wl,--no-as-needed -o m2h
```

Bibliografia

[1] *TEX - LATEX Stack Exchange. Q&A for users of TEX, LATEX, ConTeXt, and related typesetting systems.* URL: https://tex.stackexchange.com/ (visitato il 14/03/2018).

[2] *A Quickstart Guide to Debugging C Programs with gdb.* URL: http://teaching.csse.uwa.edu.au/units/CITS2230/resources/gdb-intro.html (visitato il 14/03/2018).

[3] *Aquamentus. Lex & yacc tutorial.* URL: http://aquamentus.com/tut_lexyacc.html (visitato il 14/03/2018).

[4] *Boost C++ Libraries.* URL: http://www.boost.org/ (visitato il 14/03/2018).

[5] *C++ reference.* URL: http://cppreference.com (visitato il 14/03/2018).

[6] *Codecademy.* URL: https://www.codecademy.com/ (visitato il 14/03/2018).

[7] *Codecademy Forums.* URL: https://discuss.codecademy.com/ (visitato il 14/03/2018).

[8] *cplusplus.com. The C++ Resources Network.* URL: http://www.cplusplus.com (visitato il 14/03/2018).

[9] *CTAN. The Comprehensive TEX Archive Network.* URL: http://www.ctan.org (visitato il 14/03/2018).

[10] *DI Management. Converting from lex & yacc to flex & bison.* URL: https://www.di-mgt.com.au/converting_from_lex_and_yacc.html (visitato il 14/03/2018).

[11] *Google Forms.* URL: https://forms.google.com (visitato il 14/03/2018).

[12] *Google Play. Google Opinion Rewards - App Android su Google Play.* URL: https://play.google.com/store/apps/details?id=com.google.android.apps.paidtasks (visitato il 14/03/2018).

[13] *Google Surveys.* URL: https://surveys.google.com (visitato il 14/03/2018).

[14] *GuIT. Gruppo Utilizzatori Italiani TEX e LATEX.* URL: http://www.guit.sssup.it/ (visitato il 14/03/2018).

[15] *IBM Knowledge Center. Generating a parser using yacc.* URL: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.bpxa600/genyac.htm (visitato il 14/03/2018).

[16] *ISO C++. News, Status & Discussion about Standard C++.* URL: https://isocpp.org/ (visitato il 14/03/2018).

[17] *Lemoda. Make a reentrant parser with Flex and Bison.* URL: https://www.lemoda.net/c/reentrant-parser/ (visitato il 14/03/2018).

[18] John R. Levine. *Flex & Bison: Text Processing Tools.* O'Really Media, 2013.

[19] *Lexical Analysis With Flex, for Flex 2.6.2.* URL: http://westes.github.io/flex/manual/index.html (visitato il 14/03/2018).

[20] *Linux man pages.* URL: https://linux.die.net/man/ (visitato il 14/03/2018).

[21] *M.I.T. - The Knowledge Base.* URL: http://kb.mit.edu/confluence/display/home/The+Knowledge+Base (visitato il 14/03/2018).

[22] *Mail Chimp.* URL: https://mailchimp.com/ (visitato il 14/03/2018).

[23] *man7. Linux man pages online.* URL: http://man7.org/linux/man-pages/index.html (visitato il 14/03/2018).

[24] *Microsoft Docs. Il linguaggio C/C++ e librerie Standard.* URL: https://docs.microsoft.com/it-it/cpp/cpp/c-cpp-language-and-standard-libraries (visitato il 14/03/2018).

[25] *Mozilla Developer Network. Tecnologie web per sviluppatori.* URL: https://developer.mozilla.org/it/docs/Web (visitato il 14/03/2018).

[26] Robin Nixon. *Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5.* O'Really Media, 2015.

[27] C.H. Pappas e W.H. Murray. *C/C++ Programmer's Guide.* Ziff-Davis Press, 1995.

[28] *PHP Documentation.* URL: http://php.net/docs.php (visitato il 14/03/2018).

[29] *Pluto. Italian Linux Documentation Project.* URL: http://www.pluto.it/ildp (visitato il 14/03/2018).

[30] *Samplify. Samplify® by Research Now: DIY Market Research Solution.* URL: https://www.researchnow.com/products-services/samplify/ (visitato il 14/03/2018).

[31] Abraham Silberschatz, Peter Baer Galvin e Greg Gagne. *Sistemi operativi: concetti ed esempi.* Pearson Paravia, 2006.

[32] *Software Engineering Stack Exchange. Q&A for professionals, academics, and students working within the systems development life cycle.* URL: https://softwareengineering.stackexchange.com/ (visitato il 14/03/2018).

[33] *Stack Overflow. Where Developers Learn, Share, & Build Careers*. URL: https://stackoverflow.com/ (visitato il 14/03/2018).

[34] Bjarne Stroustrup. *The C++ Programming Language*. Pearson, 2013.

[35] *SysTutorials. Linux User and Programmer's Manual - Manpages*. URL: https://www.systutorials.com/docs/linux/man/ (visitato il 14/03/2018).

[36] *The Lex & Yacc Page. Lex, Yacc, Flex, Bison: Overview, Documentation, Tools, Pointers*. URL: http://dinosaur.compilertools.net/ (visitato il 14/03/2018).

[37] *The Linux man-pages project*. URL: https://www.kernel.org/doc/man-pages/ (visitato il 14/03/2018).

[38] *Watermelon Research*. URL: http://www.watermelonresearch.com/ (visitato il 14/03/2018).

[39] *World Wide Web Consortium (W3C)*. URL: https://www.w3.org (visitato il 14/03/2018).