

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE, MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

Pakkery: progettazione e sviluppo di un sistema di authoring basato su templating

Relatore:
Dott. Angelo Di Iorio

Presentata da:
Antonio Conteduca

Sessione II
Anno Accademico 2015/2016

Indice

Introduzione	5
1 Stato dell'arte	8
1.1 Template	8
1.2 Transclusione	9
1.3 Wiki Template	10
1.4 Documenti form-based	12
1.5 Transclusioni in un ambiente HTML	13
1.5.1 Architettura del Sistema	15
1.5.2 Creazione di una transclusione	16
1.5.3 Recupero di una transclusione	17
2 Architettura e Design	18
2.1 Glossario	18
2.2 Analisi dei Requisiti	20
2.3 Formato Dati	21
2.3.1 Template	21
2.3.2 Istanza	22
2.3.3 Variabile	23
2.3.4 Documento	23
2.4 Interfaccia ed Utilizzo	24
2.4.1 Login	24
2.4.2 Homepage	24
2.4.3 Info	25
2.4.4 My template	25
2.4.5 My docs	26
2.4.6 Nuovo Documento	26
2.4.7 Modifica variabile	28
2.4.8 Nuovo Template	28
2.4.9 Aggiornamento di un'istanza	30
2.5 Interazioni	31

2.5.1	Caricamento di un Documento	31
2.5.2	Aggiornamento di un'istanza	32
3	Implementazione	33
3.1	TinyMCE	33
3.2	Gestione degli eventi	33
3.2.1	click: "#createVarModalButton"	34
3.2.2	click: "#applyEditVariable"	36
3.2.3	click: ".applyTemplate"	37
3.3	Configurazione Editor	38
3.4	Funzioni e variabili	40
3.4.1	viewChanges	40
3.4.2	applyChanges	42
3.4.3	removeVar	43
3.4.4	editVars	43
4	Valutazione	45
4.1	Obiettivi	45
4.2	Risultati	46
	Conclusioni	49
	Bibliografia	51

Elenco delle figure

1.1	Transclusione del documento B nel documento A	10
1.2	Modifica del template Letteratura con i dieci campi	11
1.3	Rendering del Template Letteratura	12
1.4	Interfaccia per la creazione di un articolo	14
1.5	Interfaccia che visualizza un articolo contenente una transclusione	15
1.6	Componenti server-side	16
1.7	Oggetto <code><transclusion></code> con relativo esempio	17
2.1	Esempio di template	18
2.2	Utilizzo di una stessa porzione di testo usata in due contesti diversi con le relative modifiche alle variabili	19
2.3	Inclusione del template <i>Stazione</i> nei documenti <i>Milano</i> e <i>Roma</i>	19
2.4	Struttura template <i>Stazione</i>	22
2.5	Struttura istanza basata sul template <i>Stazione</i>	22
2.6	Struttura variabile <i>semplice</i>	23
2.7	Struttura variabile <i>complessa</i>	23
2.8	Struttura documento <i>Milano</i>	24
2.9	Homepage	25
2.10	Editor per la creazione di un nuovo documento dal titolo <i>Lista della spesa</i>	26
2.11	modale che visualizza i template disponibili e la relativa descrizione	27
2.12	View del documento <i>Lista della spesa</i> dopo l'inclusione del template <i>Lorem Ipsum</i>	27
2.13	modale per la modifica delle variabili relative all'istanza del template <i>Lorem Ipsum</i>	28
2.14	Editor per la creazione di un nuovo template dal titolo <i>Lorem Ipsum</i> con tre variabili inizializzate	29
2.15	Modal per la creazione della variabile dal nome <i>variabile-prova</i>	29
2.16	Alert per la presenza di modifiche nelle istanze contenute nel documento	30
2.17	modale per la preview del documento <i>Lista della spesa</i> dove viene visualizzato la modifica al template <i>Lorem Ipsum</i>	31
2.18	Caricamento di un documento	32

2.19	Aggiornamento di una specifica istanza	32
4.1	Media delle valutazioni in relazione ai task	47
4.2	Media delle valutazioni in relazione alle domande	47

Introduzione

I moderni Word Processor (fra tutti Microsoft Word) permettono agli utenti di creare e modificare una varietà di documenti. Altre applicazioni permettono di realizzare una varietà di presentazioni come articoli, notiziari, brochures, volantini pubblicitari, carta intestata e altro. Alcuni word processor consentono di selezionare documenti pre-impostati che hanno una struttura già pronta per essere riempita secondo le necessità dell'utente; tali documenti vengono chiamati **template**. Per esempio, un catalogo potrebbe essere provvisto di template da cui l'utente può selezionare quello più idoneo per preparare una brochure, un modulo, una lettera. In altri sistemi, tali cataloghi sono preparati in anticipo dagli sviluppatori e presentati all'utente come una collezione da cui si può selezionare un particolare template. Selezionato ed aperto un template, l'utente può effettuare modifiche cambiando i colori o gli stili del template pre-impostato. Oltre alla comodità di avere template pre-impostati, il riuso di parti comuni di documento o la creazione di nuovi template permette all'utente di poter conformarsi ad alcuni standard senza perdersi troppo negli stili e alle norme di un determinato modulo e quindi facilitando e velocizzando la fase di scrittura. L'utilizzo di template migliora la qualità e la coerenza dei documenti in quanto i template garantiscono che i contenuti siano gli stessi e che gli aggiornamenti delle parti incluse siano propagati correttamente.

Ma cosa succede quando si utilizza un template che, una volta incluso nei documenti degli utenti, viene modificato? Ipotizziamo che un'azienda abbia cambiato il proprio direttore: tale cambiamento porterà delle conseguenti modifiche su determinati documenti. La soluzione proposta è quella di utilizzare il concetto descritto in tale elaborato e dimostrato con il software *Pakkery* che permette di modificare un unico documento e trasmettere queste modifiche sui documenti interessati. Inoltre può capitare che non tutti gli utenti vogliano applicare gli aggiornamenti ai propri documenti e quindi si ha bisogno di un meccanismo che faccia decidere all'utente se applicare o meno tali modifiche.

Quindi unendo questa necessità al bisogno di utilizzare template abbiamo progettato il software *Pakkery* che si ispira al concetto di transclusione e quello di *Warm Link*, ossia fornire all'utente la possibilità di confermare l'aggiornamento del documento. La **transclusione**, come definito da Nelson nel 1963 [1], è un processo attraverso il quale è possibile includere il contenuto di una pagina in un'altra. In altre parole, due o più documenti vengono usati per generare un terzo nuovo e diverso: la terza pagina visualizza

il contenuto delle prime due, e se in queste il testo viene modificato, viene modificato il risultato anche nella terza pagina. Questo meccanismo viene utilizzato per modificare i dati in un solo posto, ed avere poi automaticamente le modifiche disponibili in tutte le altre pagine dove i dati sono visualizzati. La differenza tra *Pakkery* e l'idea di Nelson sta nel concetto di link: infatti Nelson, nel progetto *Xanadu* [8], includeva porzioni di altri documenti attraverso il concetto di *Hot Link* quindi, quando il testo sorgente veniva modificato, cambiava anche la relativa copia nel documento in cui era incluso; mentre *Pakkery* offre all'utente la possibilità di scegliere se applicare le modifiche sul proprio documento (*Warm Link*).

Le transclusioni e il modello *xanalogo* hanno ispirato i primi sistemi ipertestuali e il Web. Uno dei sistemi disponibili nelle rete è quello che prevede la creazione di documenti mediante l'utilizzo di moduli web composti da form facilitando l'utente in quanto lo guidano alla scrittura. Un altro meccanismo è quello dei *Wiki Template* che permettono all'utente, che crea una pagina wiki, di non perdere troppo tempo nella creazione della struttura ma di focalizzare l'attenzione sul contenuto della pagina utilizzando parti di testo provenienti da altre fonti in modo da avere informazioni sempre aggiornate. Ma oltre alla struttura vogliamo evidenziare l'importanza di avere un procedimento che permetta di includere testi comuni evitando il semplice "copia e incolla" così da avere un collegamento diretto con il testo originario, uno dei punti chiave del concetto delle *transclusioni*. Uno dei lavori che più si avvicina a tale elaborato e sfrutta gli aspetti delle transclusioni, è quello proposto da Kolbitsch e Maurer [10] che avanzarono un progetto, web-based, che permette all'utente di costruire documenti elettronici utilizzando, attraverso il semplice link, documenti disponibili sul web. La comodità di avere un collegamento con il testo originale è dovuta al fatto che la modifica sul testo d'origine può essere propagata sul testo in cui c'è stata l'inclusione e quindi evitare di modificare tutti i documenti che includono quella specifica porzione di testo.

Sono molti gli studi e i progetti che hanno tentato di implementare questo tipo di meccanismo e sebbene ci sia ancora qualche aspetto da integrare, il software che andremo a descrivere presenta tutte le caratteristiche elencate prima: riuso, aggiornamento e creazione, tutto sviluppato in un ambiente web-based. Il **riuso di risorse** viene implementato attraverso l'utilizzo di template che possono essere scelti da un "modale" da cui l'utente può decidere di inserire uno o più template mantenendo in questo modo un collegamento diretto con questo, come con le *transclusioni*; quindi, quando si modificherà un template ci sarà un meccanismo che identifica gli **aggiornamenti** da effettuare e l'utente sarà libero di scegliere se applicare o meno tali modifiche, secondo il concetto del *warm link*.

Prima di partire con la descrizione di *Pakkery* andremo ad analizzare il contesto del tema in oggetto, ossia verranno espone nozioni base e lo studio di alcuni lavori precedenti, già accennati prima, in modo da avere le idee chiare su quanto verrà descritto nei

capitoli successivi. Nello specifico andremo ad analizzare in concetto di *transclusione*, fondamentale per trattare l'argomento del riuso di documenti.

Il secondo capitolo analizza l'architettura, gli elementi, le interazioni tra di essi e l'utilizzo di alcune funzionalità importanti con un basso livello di dettaglio in modo che l'utente "meno avanzato" e curioso possa comprendere i meccanismi chiave di tale lavoro.

Il terzo capitolo è rivolto alla fase implementativa. Questo capitolo descrive gli algoritmi che permettono l'esecuzione delle funzionalità di *Pakkery*.

Infine l'ultimo capitolo mira alla valutazione del software da parte degli utenti in modo da poter apportare modifiche che siano conformi alle esigenze dell'utente medio, al fine di migliorare la qualità del software.

Capitolo 1

Stato dell'arte

Prima di introdurre i concetti e l'implementazione di *Pakkery* è doveroso inquadrare il problema attraverso lo studio di punti fondamentali e di qualche lavoro simile.

1.1 Template

Per iniziare dobbiamo chiarire cosa intendiamo con il termine “template”.

Un Template è un modello predefinito che consente di creare o inserire contenuti di diverso tipo in un documento o in una pagina web.

Per fare un esempio grossolano ma efficace, citeremo la comune Carta Intestata considerandola come il template per le nostre lettere. Una serie di indicazioni comuni e ricorrenti (nome, logo ed indirizzo del mittente) sono prestampate nel modello. Quando vogliamo scrivere una nuova lettera, prendiamo il template (un foglio di carta intestata nell'esempio) e lo completiamo della parte **variabile** (destinatario, oggetto, data e testo della lettera) mentre i dati ricorrenti sono già presenti e non servirà riscriverli ogni volta.

Il template è quindi un meccanismo di riuso di documenti usati per riempirne altri. Esistono varie interpretazioni in letteratura di questo termine ma nel contesto in esame, ossia quello dei documenti, sarà questa l'interpretazione assegnata. Questo meccanismo di riuso di documenti in un'epoca come questa risulta assai utile per lo scambio di informazioni. Infatti l'avanzare delle tecnologie ha rivoluzionato l'accesso, il trasferimento, la presentazione e l'uso di informazioni. Questo avanzare ha permesso di accedere, recuperare e condividere in maniera indipendente un gran numero di informazioni da una varietà di piattaforme. Tuttavia, tutta questa accessibilità non è formattata nel giusto modo. Sebbene i formati di tali dati permettano la loro lettura, essi perdono la loro abilità di comunicare il vero significato che tali dati vogliono esprimere. Sfortunatamente la formattazione dei documenti è tipicamente persa quando i documenti vengono caricati nella rete.

Quindi si é tentato di conformare il formato delle informazioni messe in rete. Per esempio, HyperText Markup Language (HTML) é un semplice sistema di Markup per documenti usato per strutturarli. L'HTML é usato per descrivere il testo, cosi come paragrafi, liste e citazioni. Tuttavia l'HTML é limitato in questo perché non permette ai dati di essere visti in maniera ottimale per uno specifico device.

Per colmare questi limiti, alcuni hanno introdotto il concetto di “What You See Is What You Get” (**WYSIWYG**). Tale concetto si riferisce al problema di ottenere sulla carta e/o immagine che abbiano una disposizione grafica uguale a quella visualizzata sullo schermo del computer. Un esempio di software con questo formato é tinyMCE che permette di creare un documento pensando di produrlo su un comune *Word Processor* (Microsoft Word). Sebbene questo sia un problema, con questo lavoro vogliamo porre maggiormente l'attenzione sulle comodità di riutilizzare documenti, o porzioni di essi, più che sulla loro struttura.

1.2 Transclusione

Per capire ciò che è alla base di questo elaborato dobbiamo affrontare il concetto di transclusione anche attraverso lo studio di alcuni lavori che utilizzano tale meccanismo in maniera differente. Infatti come vedremo nelle sezioni successivi: i *wiki template* (1.3) mirano al riuso mentre i *documenti form-based* (1.4) al rendere la scrittura di documenti da parte dell'utente più semplice.

Molti documenti sono composti da una varietà di frammenti provenienti da altri documenti. Infatti i documenti digitali non sono unità isolate. Molti di loro contengono collegamenti a risorse esterne, riferimenti a precedenti revisioni e altro ancora. [7]

Tale meccanismo, chiamato *Transclusione*, fu proposto da Ted Nelson (1963) nel suo “Xanadu Project” [8]. Secondo Nelson la *Transclusione* é una forma avanzata di inclusione che permette di incorporare il contenuto di una pagina in un'altra, ossia due o più documenti possono essere usati per generarne un terzo nuovo e diverso. Questo terzo documento visualizza i contenuti dei primi due e se il testo delle due pagine “sorgenti” viene modificato, i cambiamenti si ripercuotono anche sul terzo documento. Esso é quindi un meccanismo che permette di modificare i dati in un singolo posto ed avere le modifiche disponibili in tutte le altre pagine dove i dati sono visualizzati.

L'idea di Nelson era quella di creare documenti dove ogni frammento poteva essere sempre identificato, recuperato e aggregato all'interno del sistema. L'informazione su chi, quando e come ogni frammento è stato modificato permette agli utenti di manipolare i documenti correlati in maniera efficiente. La permanente connessione tra il contenuto incluso e la sua originale sorgente è un'altra caratteristica molto importante del meccanismo delle transclusioni.

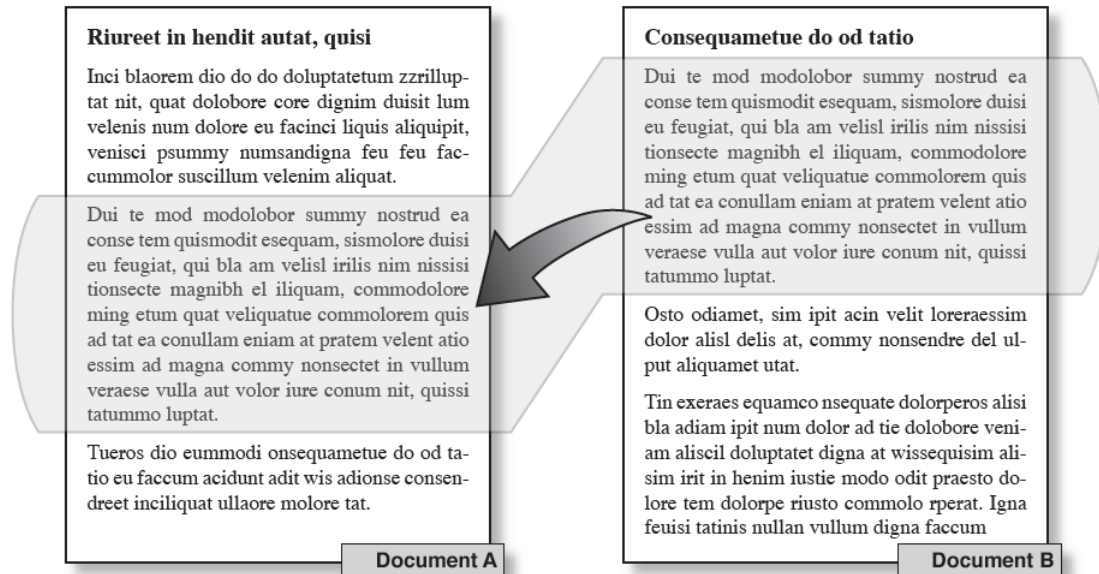


Figura 1.1: Transclusione del documento B nel documento A

Esistono due modi di vedere e modificare un documento contenente inclusioni:

- accedendo al documento trattando le inclusioni come se fossero trasparenti
- evidenziando le inclusioni e differenziando le varie informazioni relative ad esso; tale approccio è interessante nel caso in cui gli utenti siano consapevoli di chi, quando e come ogni frammento è stato editato. Così facendo è possibile identificare i singoli contributi nel documento modificato, mostrare le diverse versioni evidenziando ogni modifica.

Quindi sono tante le funzionalità che possono essere costruite dalla connessione tra l'inclusione in un documento e la sua copia originale.

1.3 Wiki Template

Uno dei tanti lavori che implementano il meccanismo delle transclusioni è il Wiki Template. Un Wiki è un software web che permette ai propri utenti di aggiungere, modificare o cancellare contenuti attraverso l'utilizzo di un browser web (l'esempio più conosciuto di Wiki è senz'altro "Wikipedia"). Il termine fu coniato da Ward Cunningham nel Marzo del 1995 [2] fondando il primo Wiki nella storia del web: "Portland Pattern Repository". I Wiki sono applicati in molte applicazioni web e da questo concetto sono stati sviluppati molti motori per la costruzione di template [6]. Una parte del successo dei Wiki è basata sulla loro totale libertà, facilità di accesso e mancanza di struttura [4]. In tal modo si possono esprimere idee e condividere informazioni velocemente. Un altro vantaggio è

senza dubbio la possibilità di creare collegamenti ipertestuali in maniera rapida da parte dell'utente. Quindi un Wiki può essere considerato come uno strumento attraverso il quale aiutare l'utente ad esprimere e condividere le proprie idee.

Definito cos'è un wiki fissiamo il concetto di *wiki template*:

Un Wiki template è un meccanismo che permette all'utente di determinare la struttura e l'aspetto di una pagina Wiki.

Alla creazione di una pagina, l'autore può selezionare un template dall'insieme dei template disponibili e assegnarlo alla nuova pagina che sta creando. Questo determinerà come la pagina sarà visualizzata all'utente e come i campi della pagina saranno compilati dall'autore. Per chiarire consideriamo l'esempio proposto nello studio [3]: una maestra crea un template per "letteratura" (fig. 1.2) e lo aggiunge all'insieme dei template disponibili. A partire da ciò gli autori potranno selezionare questo template per definire il comportamento delle loro pagine. Il template "letteratura" è composto da dieci differenti elementi che hanno il compito di assegnare significato alla struttura.

The screenshot shows a web-based form for editing a template. The form is titled "Diploma Thesis: Two-level Tailoring Support for CSCL". It has a navigation bar at the top with a search box and several icons. The main content area is divided into several sections, each with a text input field:

- Bibliographic Data:** This section contains several input fields:
 - Author: Bounimi, M., Haake, J., Lendgrot, B., Schummer, T. & Haake, A.
 - Title: Two-level Tailoring Support for CSCL
 - In: Proceedings of CRMWG03, Lecture Notes in Computer Science, Volume 280
 - Place: Autrans, France
 - Year: 2003
 - Pages: 74-81
 - URL: http://www.springerlink.com/link.asp?id=420c&wvwp039v4
- Summary:** A large text area for the summary.
- Research Questions:** A text area containing a list of questions:
 - How do templates affect the options for tailoring?
 - What are relevant levels for tailoring?
 - What level of complexity can the users handle?
- Research Methods:** A text area containing a list of methods:
 - The authors first report on interviews made with users. Based on the interviews, they present a theory for tailoring on different levels. This theory is validated by:
 - providing a prototypical implementation
 - providing anecdotal experiences with power users who tailored the system.
- Results:** A text area containing a list of results:
 - The authors claim that
 - users have the need for tailoring,
 - users are not able to handle complex tailoring languages, and
 - tailoring should be performed on different levels.

However, they point out that additional evaluation of their tailoring language is needed.

At the bottom of the form, there are "Save" and "Cancel" buttons. The form also has a navigation bar at the bottom with a search box and several icons.

Figura 1.2: Modifica del template Letteratura con i dieci campi

Un aspetto molto importante, dal punto di vista dell'utente, è quello di permettere a questo di selezionare solo ciò che il template propone. Inoltre l'utente che usa un template non ha bisogno di comprendere il testo sorgente che lo compone ma solo i campi che lo

compongono. In seguito per visualizzare la pagina si utilizza il template in uso in modo da definire come i campi saranno interpretati (fig.1.3).

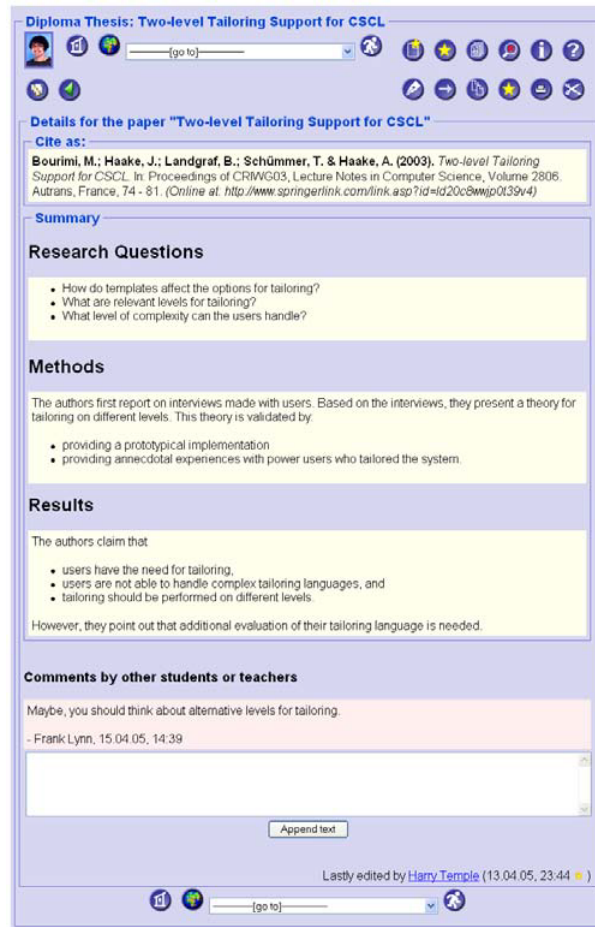


Figura 1.3: Rendering del Template Letteratura

1.4 Documenti form-based

Altri lavori mirano ad utilizzare il concetto delle transclusionione per guidare l'utente alla scrittura di documenti. Infatti, come anche affermato da Ritter nel suo progetto *Template-based creation of electronic document* [9], molti sistemi permettono agli utenti di poter generare documenti elettronici. I seguenti sono esempi di questa situazione: un operatore telefonico che interagisce con i clienti genera degli ordini di vendita basati su queste interazioni; oppure un segretario riempie moduli per un dipartimento; un utente che compila un modulo elettronico sul web. Se l'utente è nuovo o relativamente inesperto nella compilazione di moduli, tipicamente l'interfaccia provvede a guidarlo. Tuttavia, l'utente che ha usato un template, o qualcosa di simile, ha confidenza con la struttura e il contenuto. L'utente potrebbe conoscere, prima di visualizzare il template, quali campi sono da completare e l'ordine. Questi utenti "avanzati" potrebbero trovare scomodo inserire necessariamente informazioni usando la struttura del template che l'interfaccia fornisce.

Inoltre l'interfaccia potrebbe richiedere all'utente di effettuare compiti ripetitivi per compilare tutti i campi. Per esempio, i campi per l'inserimento di informazioni essenziali potrebbero essere sparsi su tutto il modulo. A meno che l'utente sia disposto a muoversi tra i vari campi, le informazioni devono essere inserite seguendo l'ordine dell'interfaccia. Gli svantaggi in queste situazioni sono dovuti al fatto che un utente ben informato potrebbe sentire il sistema lento e poco configurato in quanto non permette all'utente di inserire le informazioni essenziali in maniera veloce. Per risolvere questo problema l'utente può lavorare con diversi template che sono strettamente correlati al compito per cui sono stati creati.

1.5 Transclusioni in un ambiente HTML

Sebbene le transclusioni furono descritte all'inizio del 1960 esse non sono state ancora rese disponibili agli utenti del World Wide Web. In un loro articolo [10], Kolbitsch e Maurer descrissero un prototipo di implementazione di un sistema che permetteva all'utente di scrivere articoli che potevano contenere transclusioni. Il sistema è web-based e attraverso un semplice bottone dell'interfaccia l'utente può inserire transclusioni provenienti da pagine HTML disponibili dalla rete. A differenza di altri approcci utilizzati per implementare questo concetto, che prevedevano l'introduzione di nuove "regole" per l'HTML, il sistema offerto da i due precedentemente citati utilizza solamente tecniche che provengono dall'ambiente HTML. JavaScript, DOM, PHP, HTTP, e lo stesso HTML sono le tecnologie principali utilizzate per lo sviluppo del prototipo.

L'HTML è un linguaggio relativamente facile per descrivere il contenuto di pagine ipertestuali indipendenti dalla piattaforma [13]. L'ambiente che si sta per descrivere mira a facilitare il riuso di informazioni disponibili nella rete. Gli obiettivi principali sono stati:

- **facilità di utilizzo:** il sistema deve permettere di creare transclusioni in maniera semplice come un tradizionale copia-incolla;
- **utilizzo di qualsiasi documento:** qualunque pagina web può essere sorgente di una qualche transclusione;
- **livello di granularità:** qualunque porzione di testo può essere incluso, da un singolo carattere ad una pagina per intero.

Il sistema non richiede alcun plugin per il browser e nessun tool aggiuntivo. Come scritto nei paragrafi precedenti, le uniche tecnologie di cui fa uso il sistema sono:

- **HTML:** le transclusioni possono essere costruite da qualunque documento HTML formattato reperibile dalla rete;

- **Javascript, DOM:** la maggior parte dei browser web rappresentano le pagine HTML come un albero di oggetti. La tecnologia alla base di questo concetto è il *Document Object Model*[11] [12]. Javascript è utilizzato per accedere agli elementi del DOM della pagina oggetto della transclusione [13] [14];
- **HTTP:** i documenti contengono transclusioni che sono trasmesse ai lettori usando il protocollo di trasferimento ipertestuale [15]

Elencati gli strumenti, passiamo ora alla descrizione del sistema in questione. Nel prototipo si possono distinguere due fondamentali azioni: la creazione di una transclusione ossia la creazione di un articolo e la sua valutazione nel momento in cui questo è mostrato. Quando un utente vuole creare un articolo con una transclusione il browser mostra un'interfaccia divisa in due parti (fig. 1.4). La prima parte contiene una convenzionale textarea, in cui l'utente può inserire contenuto in HTML, e due bottoni che permettono il salvataggio dell'articolo e l'inserimento di una transclusione. Quando l'utente preme quest'ultimo bottone viene caricata la pagina del relativo URL inserito. La fase di caricamento avviene mediante un'applicazione HTTP ed il relativo contenuto viene mostrato nella seconda parte dell'interfaccia. Caricato il documento da transcludere l'utente può selezionare una porzione di testo dal secondo frame e premere il bottone che ne permette la transclusione con il successivo inserimento nell'articolo che si sta creando. Questo bottone richiama una funzione javascript che determina la posizione iniziale e finale della selezione.

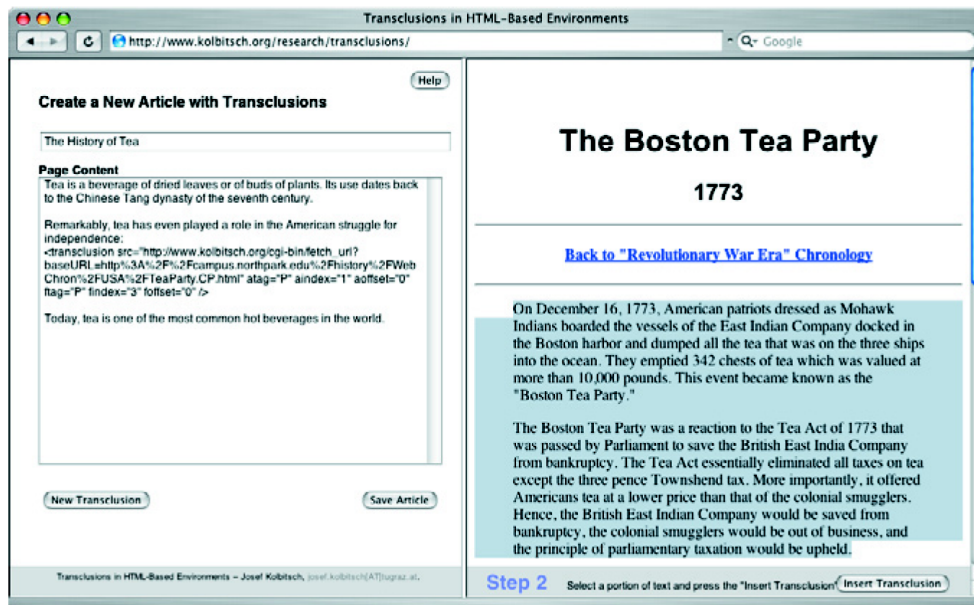


Figura 1.4: Interfaccia per la creazione di un articolo

Conclusa la fase di creazione dell'articolo l'utente può scegliere di salvare il documento. Il contenuto della textarea, con inclusa la transclusione, è inviata al server attraverso uno

script CGI. Il server archivia il testo “statico” e i valori forniti dalla transclusione (tag e parametri). Quando viene richiesto un articolo contenente una transclusione é invocato un altro script CGI che recupera il contenuto ed i parametri delle transclusione dal database. I parametri che definiscono la transclusione sono usati per caricare la pagina sorgente dalla sua originale locazione. Se non ci sono modifiche, la porzione di testo transclusa é estratta dalla pagina originale e una volta combinata con il testo “statico” viene inviata al browser (fig. 1.5).

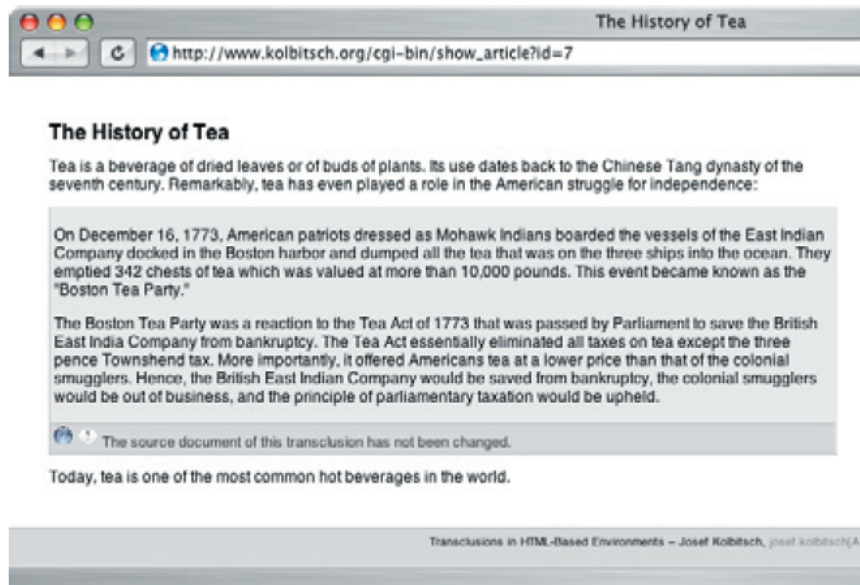


Figura 1.5: Interfaccia che visualizza un articolo contenente una transclusione

1.5.1 Architettura del Sistema

L'implementazione segue una classico paradigma client-server. Un convenzionale server HTTP, una database relazionale, una serie di script CGI, un proxy HTTP e codice Javascript sono i componenti principali. Come si nota dalla figura 1.6, esistono tre tipi di script CGI:

- script “Crea e Salva”: riceve i dati presentati dall'utente, ne analizza il contenuto, estrae transclusioni e salva questo ed il contenuto dell'articolo nel database interno al sistema;
- script “Estrai e Unisci”: legge il contenuto di un articolo insieme alle informazioni sulle transclusioni dal database, recupera il documento sorgente della transclusione, assembla l'articolo e lo invia all'utente;
- script “Recupera”: utilizzato durante la creazione per caricare le pagine soggette a transclusioni.

Per quanto riguarda il database relazionale, esso é composto di sole due tabelle: una contenente la parte “statica” dell'articolo e l'altra contenente le informazioni dettagliate sulle transclusioni.

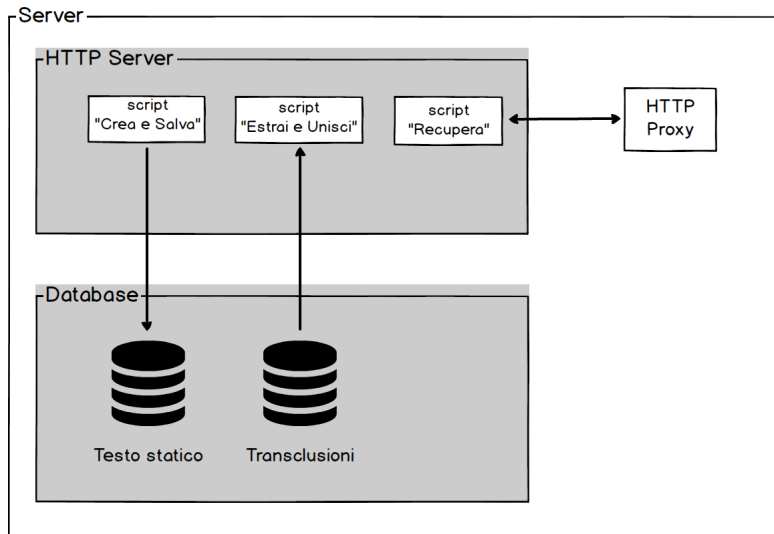


Figura 1.6: Componenti server-side

1.5.2 Creazione di una transclusione

Come descritto prima, l'interfaccia per la creazione di un nuovo articolo (fig. 1.4) é divisa in due parti. Quando l'utente desidera inserire una transclusione seleziona una porzione di testo con il puntatore del mouse e clicca sul bottone fornito dall'interfaccia. Questo click richiama una funzione Javascript che accede al DOM del documento esterno per determinare il punto iniziale e finale della selezione fatta dall'utente. Inoltre la funzione client-side genera un tag intermedio `<transclusion>` (fig. 1.7) che viene inserito nell'articolo. Questo nuovo tag é composto da sette parametri che individuano le informazioni necessarie per determinare, in maniera precisa, l'*anchor* e il *focus* della selezione. Questi sette parametri sono:

- *src*: URL del documento da transcludere;
- *atag*, *ftag*: i nomi dei tag in cui la transclusione inizia e finisce;
- *aindex*, *findez*: indici dei tag in cui la transclusione inizia e finisce;
- *aoffset*, *foffset*: indicano rispettivamente l'inizio e la fine della transclusione.

Quando l'articolo con l'inclusione é salvato dall'utente, i dati sono inviati al server e viene invocato lo script “Crea e Salva” che ne estrae le transclusioni dall'articolo, determina gli attributi e scrive le informazioni ricavate nel database. É importante sottolineare che il tag `<transclusion>` é utilizzato unicamente nella fase di creazione ed é, quindi, visibile solo all'interno del sistema e non all'utente.

```
<transclusion src="{url}"
  atag="{tag}" aindex="{int}" aoffset="{int}"
  ftag="{tag}" findex="{int}" foffset="{int}" />

<transclusion src="http://www.kolbitsch.org/about/"
  atag="H1" aindex="1" aoffset="0"
  ftag="P" findex="4" foffset="29" />
```

Figura 1.7: Oggetto *<transclusion>* con relativo esempio

1.5.3 Recupero di una transclusione

Quando viene richiesto un articolo, il suo “corpo” è analizzato con lo scopo di cercare le transclusioni. Il componente che si occupa di questo è, come già detto, lo script CGI “Estrai e Unisci”. Per ogni oggetto transclusione eventualmente trovato vengono effettuati i seguenti steps:

- risoluzione dell’oggetto e recupero delle informazioni sia sulla transclusione che sul documento situato nel database;
- verifica del caricamento del documento attraverso l’URL della risorsa;
- recuperata un inclusione bisogna verificare se i metadati (data creazione e/o modifica) sono stati modificati;
- se i dati inerenti al documento sono verificati si recupera la risorsa e si estrae la porzione di testo determinata dai valori degli attributi contenuti nel tag *<transclusion>*;
- si ricopia l’oggetto transclusione nell’articolo con il relativo contenuto;
- se qualche operazione non è andata a buon fine viene segnalata come messaggio di errore.

Ogni transclusione è formattata in modo da permettere all’utente/lettore di poter distinguere il contenuto proprio da quello che appartiene alla transclusione (nella fig. 1.5 lo sfondo della transclusione è di un colore più scuro rispetto al resto del testo).

Questa implementazione ha mostrato la nozione di ipertesto di Ted Nelson e uno dei suoi primi concetti (transclusioni). Sebbene l’HTML sia stato influenzato dai principi delle transclusioni per le inclusioni più complesse, come le immagini, non si è ancora trovata un’implementazione regolare e soddisfacente. Perciò esistono numerose implementazioni proposte che sviluppano le transclusioni con la tecnologia disponibile in questo momento. Una possibile implementazione, anche se descritta in poche pagine, è quella proposta in questa tesi.

Capitolo 2

Architettura e Design

Questo capitolo offre una panoramica del software introducendo le relazioni tra l'ambiente e i principi di sviluppo utilizzati nel design dei componenti. Inoltre si procederà a chiarire concetti e strutture fondamentali per il corretto apprendimento sull'utilizzo di *Pakker*.

2.1 Glossario

Per evitare ambiguità o confusioni su termini specifici che incontreremo durante la descrizione di questo capitolo è bene chiarire una terminologia base:

- **Template:** documento già pronto che può essere incluso in ulteriore documento; composto da parte variabile e parte fissa; nella figura 2.1 la parte in rosso identifica la variabile mentre quelle in nero la parte fissa;

Template: Stazione

La stazione di **Bologna Centrale** è la principale stazione ferroviaria della città di **Bologna** ed è la **quinta** in Italia per dimensioni e volume di traffico viaggiatori (circa **78.000 m²** attraversati da **58.000.000** di viaggiatori all'anno).

Figura 2.1: Esempio di template

- **Istanza:** l'inclusione di un template all'interno del documento; nella figura 2.3 viene incluso il template *Stazione* nel documento *Milano e Roma* con la sostituzione della parte variabile.
- **parte fissa:** porzione di testo appartenente al template o istanza; un aggiornamento del template sovrascrive le eventuali modifiche fatte dall'utente su questa porzione preservando, come detto al punto precedente, la parte variabile;

- **variabile**: porzione di testo di un template o istanza che può essere modificata; il contenuto della variabile non verrà sovrascritto nella fase di aggiornamento del template; ogni variabile è identificata da un nome e nella figura 2.2 sono evidenziate di rosso.

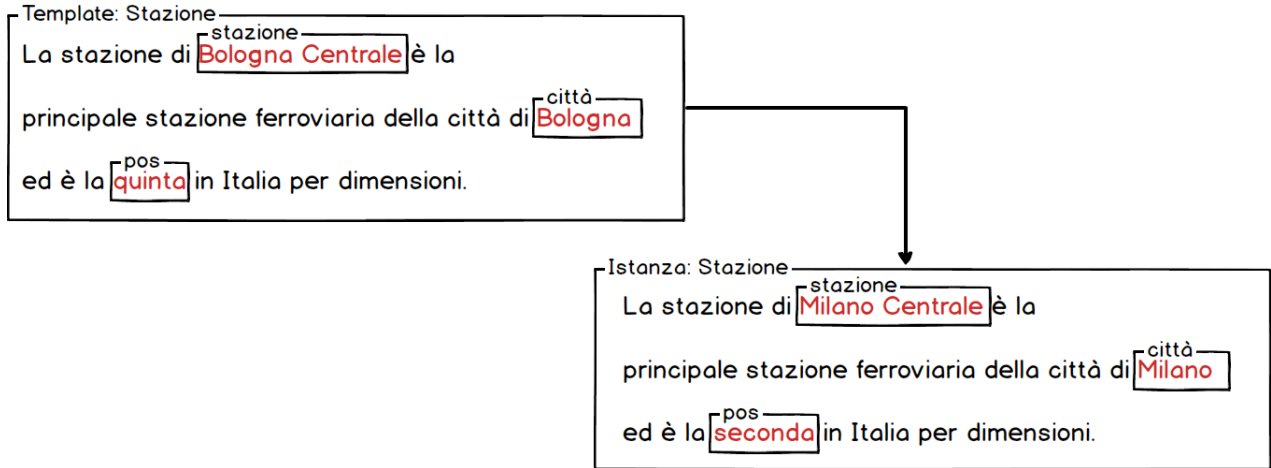


Figura 2.2: Utilizzo di una stessa porzione di testo usata in due contesti diversi con le relative modifiche alle variabili

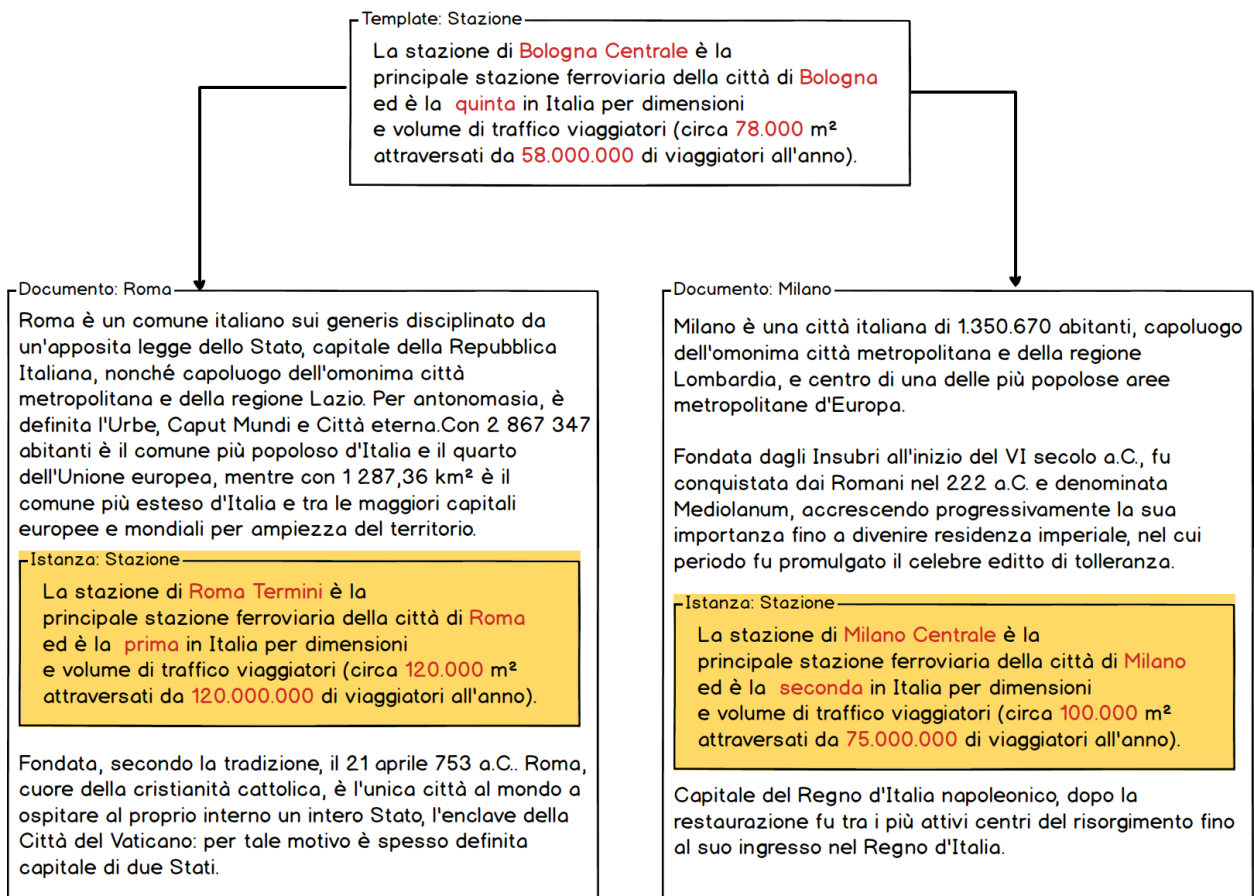


Figura 2.3: Inclusione del template *Stazione* nei documenti *Milano* e *Roma*

2.2 Analisi dei Requisiti

Partiamo con l'analisi dei requisiti ossia la descrizione di ciò che il software deve o dovrà fare. Questa fase è molto importante nella produzione di software in quanto capita spesso che né il programmatore né il cliente abbiano una conoscenza completa di tutte le caratteristiche del sistema nelle prime fasi del processo di sviluppo. È necessario, quindi, adottare delle attività che permettano ad entrambe le parti di poter sfruttare la propria crescente competenza durante il processo di sviluppo. Capire i requisiti e redigerli in maniera chiara e concisa è un buon punto di partenza per avere una buona manutenzione e impedire la scrittura di codice che non soddisfa le intenzioni del cliente.

L'approccio usato per stilare i requisiti è stato quello di utilizzare la formula delle *user story* [18] ossia una breve descrizione dell'interazione di un utente o sistema esterno col sistema. La formula delle *user story* sembra molto facile rispetto ad altri metodi di "cattura" dei requisiti. Eppure nella sua semplicità si rivela efficace per descrivere ciò che dev'essere realizzato. Queste sono state ricavate dal team di sviluppo per avere un'idea chiara sulle necessità dei clienti. Con le *User Story*, le specifiche sono scritte ad alto livello in modo che tutti i partecipanti possano comprenderle, a prescindere dalle conoscenze tecniche.

Andiamo ora ad analizzare i requisiti fondamentali, e tutti implementati, di Pakkery attraverso la formula delle *user story*.

1. Accesso:

- Come utente iscritto, vorrei loggarmi (con user e password) senza inserire ogni volta la password così da non doverla ricordare.

2. Gestione documentale:

- Come utente, vorrei visualizzare la lista dei miei documenti;
- Come utente, vorrei modificare i miei documenti;
- Come utente, vorrei avere la possibilità di visualizzare informazioni riguardanti i miei documenti (titolo, template utilizzati);
- Come utente, vorrei poter filtrare un documento per fare risaltare le parti dei template;
- Come utente che crea un documento, vorrei aggiungere istanze nel mio documento;
- Come utente che usa un istanza di template, vorrei essere avvisato nel caso in cui esso venga modificato;
- Come utente che usa un istanza di template modificata, vorrei vedere un'anteprima della modifica e decidere se applicarla al documento che usa quell'inclusione modificata.

3. Gestione template:

- Come utente, vorrei visualizzare la lista dei template disponibili;
- Come utente, vorrei creare template;
- Come utente che crea un template, vorrei inserire una breve descrizione del template che sto creando;
- Come utente, vorrei modificare i miei template;
- Come utente, vorrei creare variabili nei miei template;
- Come utente, vorrei eliminare variabili nei miei template;

2.3 Formato Dati

Per la realizzazione di tutti i requisiti elencati nella sezione precedente abbiamo creato delle strutture indipendenti ed autonome tra di loro in modo da rendere più semplice la loro implementazione. Le strutture descritte di seguito sono trasparenti all'utente e facilitano il reperimento dei dati al sistema in quanto ogni informazione relativa ad istanze, documenti, template e variabili é contenuta nella struttura stessa (*self contain*). Queste informazioni sono inserite nei tag attraverso lo speciale attributo *data-** di HTML5 che consente di memorizzare dati personalizzati [20].

2.3.1 Template

La struttura scelta per il template si compone di quattro attributi quali:

- *data-template*, specifica l'URI della risorsa ossia il template;
- *data-template-info*, breve descrizione del template;
- *data-template-version*, versione del template, al momento della creazione di un nuovo template il valore di default è 1 e ad ogni modifica tale valore viene incrementato di uno;
- *data-template-name*, nome che l'utente assegna al template, è l'ultima parte che compone l'URI.

La figura mostra un template con cinque variabili. Gli attributi all'interno del tag *div* identificano l'URL del template, ossia la risorsa è collocata nella sottocartella "template" di "pakkery", la descrizione del template, il nome del template e la versione che al momento della creazione di un template viene inizializzata a 1.

```

<div data-template-instance="http://localhost/Tesi/pakkery/template/Stazione"
    data-template-name="Stazione"
    data-template-instance-version="1" >

    La stazione di <span data-template-variable="stazione">Bologna centrale</span>
    è la principale stazione ferroviaria della città di
    <span data-template-variable="city">Bologna</span> ed è la
    <span data-template-variable="position">quinta</span> in Italia
    per dimensioni e volume di traffico viaggiatori (circa
    <span data-template-variable="sup">78.000</span>m2
    attraversati da<span data-template-variable="pass">58.000.000</span>
    di viaggiatori all'anno).

</div>

```

Figura 2.4: Struttura template Stazione

2.3.2 Istanza

La struttura scelta per l'istanza si compone di tre attributi quali:

- *data-template-instance*, specifica l'URI del template la cui istanza fa riferimento
- *data-template-name*, specifica il nome del template
- *data-template-instance-version*, specifica la versione al momento della creazione dell'istanza

```

<div data-template-instance="http://localhost/Tesi/pakkery/template/Stazione"
    data-template-name="Stazione"
    data-template-instance-version="1" >

    La stazione di <span data-template-variable="stazione">Milano centrale</span>
    è la principale stazione ferroviaria della città di
    <span data-template-variable="city">Milano</span> ed è la
    <span data-template-variable="position">seconda</span> in Italia
    per dimensioni e volume di traffico viaggiatori (circa
    <span data-template-variable="sup">100.000</span>m2
    attraversati da<span data-template-variable="pass">75.000.000</span>
    di viaggiatori all'anno).

</div>

```

Figura 2.5: Struttura istanza basata sul template Stazione

2.3.3 Variabile

Una variabile è sempre un frammento di HTML che appartiene al template e può essere un semplice testo

```
<span data-template-variable="nome_variabile">
  contenuto della variabile
</span>
```

Figura 2.6: Struttura variabile *semplice*

o un frammento di HTML più articolato (lista, tabella, ...).

```
<div data-template-variable="nome_variabile">
  Questo è una variabile il cui contenuto è una lista:
  <li>Item Uno </li>
  <li>Item Due </li>
  <li>Item Tre </li>
  <li>Item Quattro </li>
</div>
```

Figura 2.7: Struttura variabile *complessa*

Come si nota dalle due ultime figure, la differenza principale tra questi due tipi di variabili è il tag che li racchiude. Infatti si tratta di uno *span* per la variabile *semplice* e di un *div* per la variabile *complessa*. Inoltre l'unico attributo all'interno del tag che racchiude la variabile è il *data-template-variable* che identifica il nome della variabile.

2.3.4 Documento

L'ultima struttura descritta è quella relativa al documento. In esso si ritrovano le strutture riguardati un'istanza e una variabile con i conseguenti attributi. L'unico attributo all'interno del tag che rappresenta il documento è *data-document* che identifica l'URI del documento. La seguente figura mostra la struttura del documento con l'inclusione del template *Stazione* (istanza di figura 2.5) con numero di versione *uno*: questo significa che l'istanza è stata creata utilizzando la versione numero uno del template; inoltre le variabili hanno un contenuto differente rispetto al template di figura 2.4.

```

<div data-document="http://localhost/Tesi/MasterProject/documents/Milano" >

Milano è una città italiana di 1.350.670 abitanti, capoluogo dell'omonima
città metropolitana e della regione Lombardia, e centro di una delle più
popolose aree metropolitane d'Europa.
Fondata dagli Insubri all'inizio del VI secolo a.C., fu conquistata dai
Romani nel 222 a.C. e denominata Mediolanum, accrescendo progressivamente
la sua importanza fino a divenire residenza imperiale, nel cui periodo fu
promulgato il celebre editto di tolleranza.

<div data-template-instance="http://localhost/Tesi/MasterProject/template/Stazione"
data-template-name="Stazione"
data-template-instance-version="1" >

La stazione di <span data-template-variable="stazione">Milano centrale</span>
è la principale stazione ferroviaria della città di
<span data-template-variable="city">Milano</span> ed è la
<span data-template-variable="position">seconda</span> in Italia
per dimensioni e volume di traffico viaggiatori (circa
<span data-template-variable="sup">100.000</span>m2
attraversati da<span data-template-variable="pass">75.000.000</span>
di viaggiatori all'anno).

</div>

Capitale del Regno d'Italia napoleonico, dopo la restaurazione fu tra
i più attivi centri del risorgimento fino al suo ingresso nel Regno d'Italia.

</div>

```

Figura 2.8: Struttura documento *Milano*

2.4 Interfaccia ed Utilizzo

Per la costruzione dell'interfaccia è stato utilizzato il bootstrap "AdminLTE" [19]. AdminLTE è un template Web, open source, per la realizzazione di siti gestionali. È scritto in HTML e con l'utilizzo del framework *CSS Bootstrap 3* ed è basato su una struttura modulare che lo rende facile da personalizzare.

2.4.1 Login

L'accesso all'applicazione avviene mediante una classica schermata di login che richiede *user* e *password*. Per adesso non è presente nessuna gestione a livello di database e quindi qualunque utente, anche non iscritto, ha la possibilità di interagire con le funzionalità dell'applicazione.

2.4.2 Homepage

Effettuato il login, l'utente si ritrova la seguente schermata (fig. 2.9) con la sidebar (pannello a sinistra) che mostra le operazioni che si possono effettuare. Il pannello utente

posto nella parte superiore della sidebar permette all'utente di uscire dall'applicazione con la successiva eliminazione dei *cookie*¹. Senza questo passaggio l'utente può accedere all'applicazione in un secondo momento saltando la fase di login.

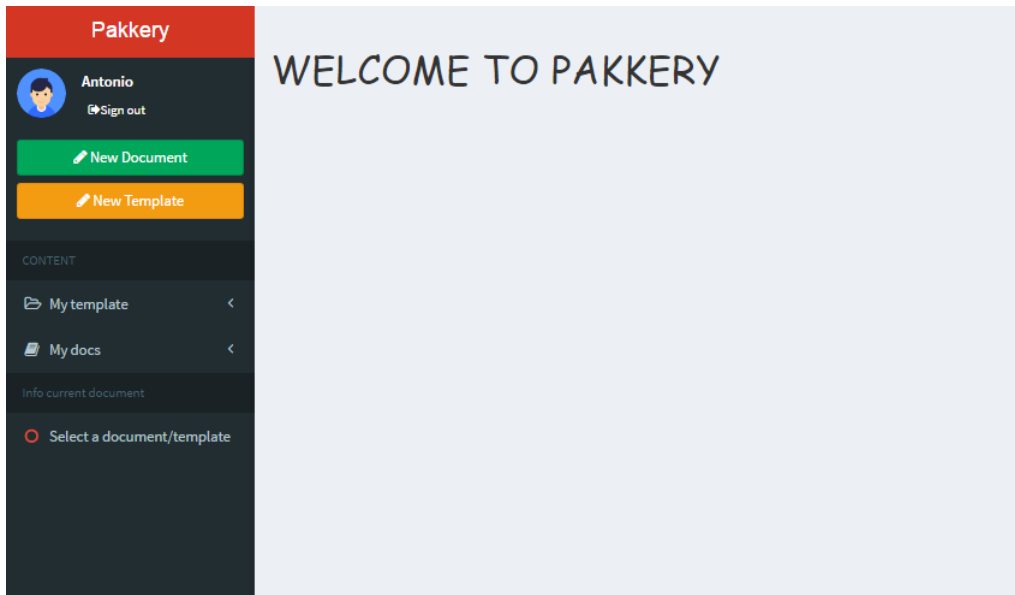
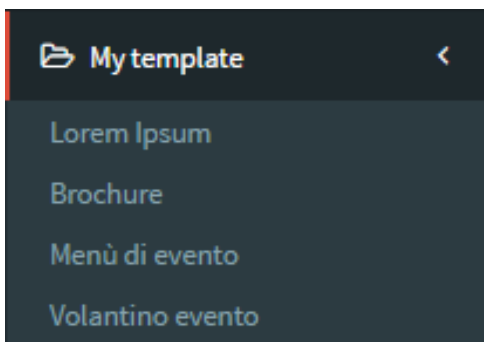


Figura 2.9: Homepage

2.4.3 Info

La sezione “info” fornisce metadati sull’oggetto che si sta visualizzando. Nel caso di un template questo mostrerà il titolo del template, la descrizione e la versione. Per quanto riguarda un documento, invece, le informazioni, oltre al titolo, saranno riguardanti i template utilizzati al suo interno.

2.4.4 My template

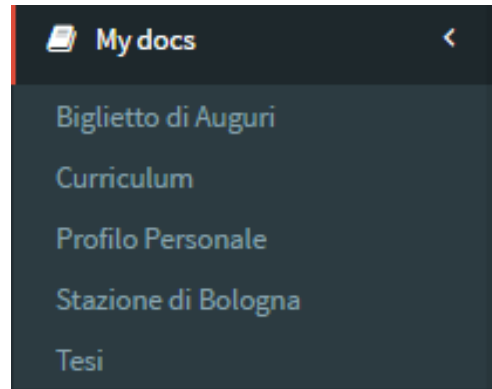


Nella sezione “My template”, come facile intuire, si possono trovare i template creati e disponibili. Il “click” su un elemento della lista permetterà la visualizzazione del template con la possibilità, dopo aver visualizzato il template, di modificarlo e di creare nuove variabili all’interno del template. Tutti i template presenti nella seguente lista potranno essere inclusi, a seconda delle necessità, nei documenti che l’utente ha intenzione di creare.

¹Meccanismo tipicamente utilizzato per implementare meccanismi di identificazione di un client presso un server. Contiene brevi informazioni che possono essere salvate sul computer dell’utente quando il browser richiama un determinato sito web [21].

2.4.5 My docs

La sezione analoga a quella “My template” è la sezione “My docs”. In questa sezione si possono trovare i documenti creati dall’utente. Il “click” su un elemento della lista permetterà la visualizzazione del documento con la possibilità, dopo aver visualizzato il documento, di modificarlo e di aggiungere istanze di template. C’è anche la possibilità di ricevere informazioni sul documento corrente, ossia titolo e template utilizzati, cliccando sull’oggetto *Info* posto subito dopo la lista dei documenti.



2.4.6 Nuovo Documento

La creazione di un nuovo documento porta l’utente ad interagire con l’editor Tiny. Al “click” del bottone “new Document”, nella sidebar, l’interfaccia, attraverso l’uso di un modale, propone all’utente di inserire il campo obbligatorio “titolo”. Una volta inserito il titolo apparirà la seguente interfaccia:

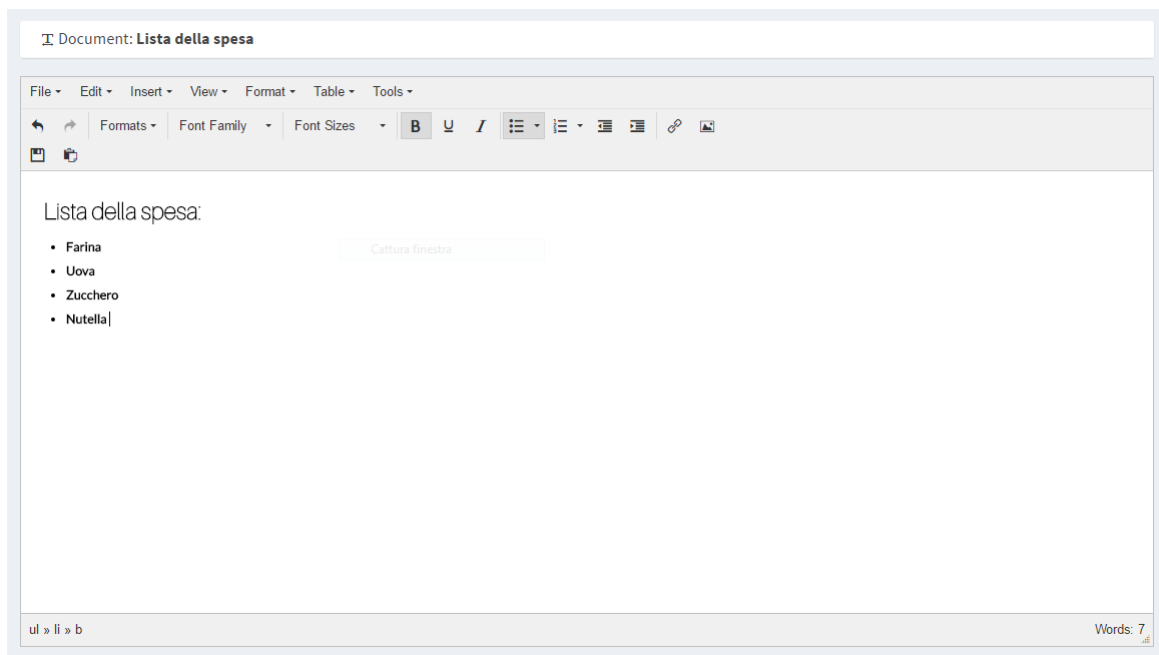


Figura 2.10: Editor per la creazione di un nuovo documento dal titolo *Lista della spesa*

L’editor fornisce, oltre a quella di tiny, una *toolbar* ad hoc per le funzionalità di *Pakery* ossia la possibilità di salvare il documento (📁) e di aggiungere istanze di template ad esso (📄). L’aggiunta di un’istanza al documento avviene mediante la selezione del template scelto dal modale seguente:

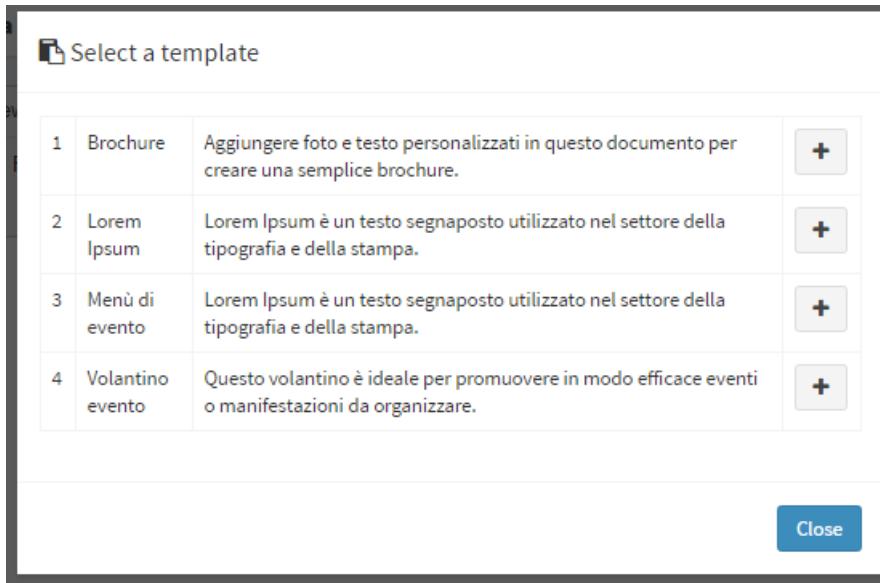


Figura 2.11: modale che visualizza i template disponibili e la relativa descrizione

Il documento con l'istanza appena inserita (si è scelti di includere il template *Lorem Ipsum*) apparirà nel seguente modo:

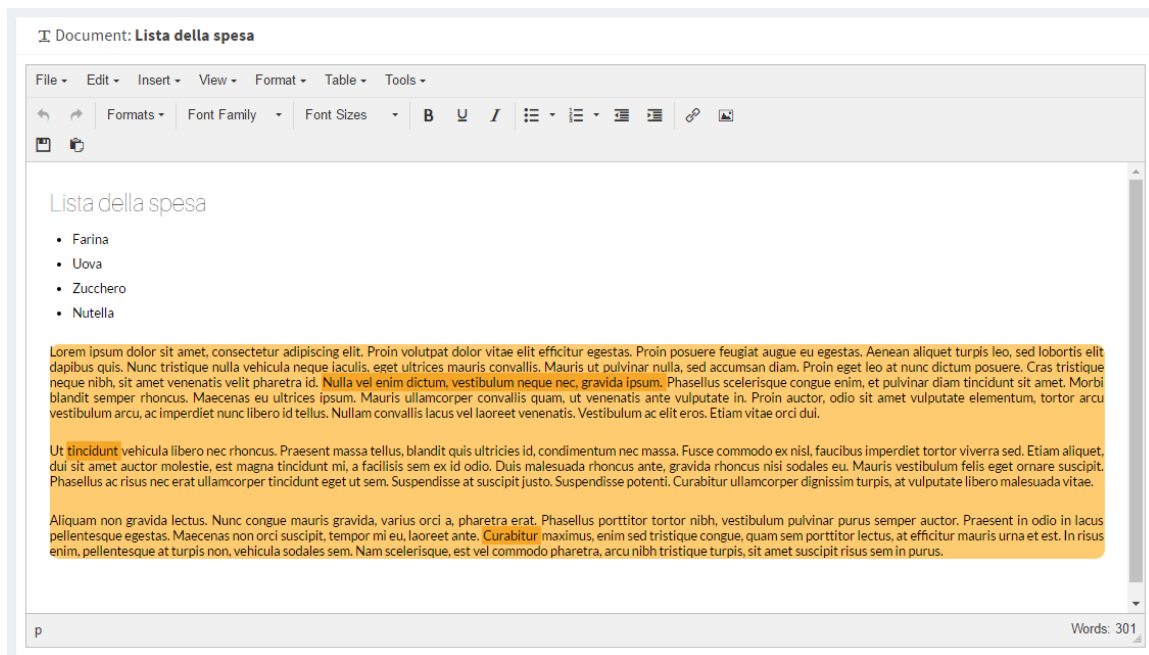
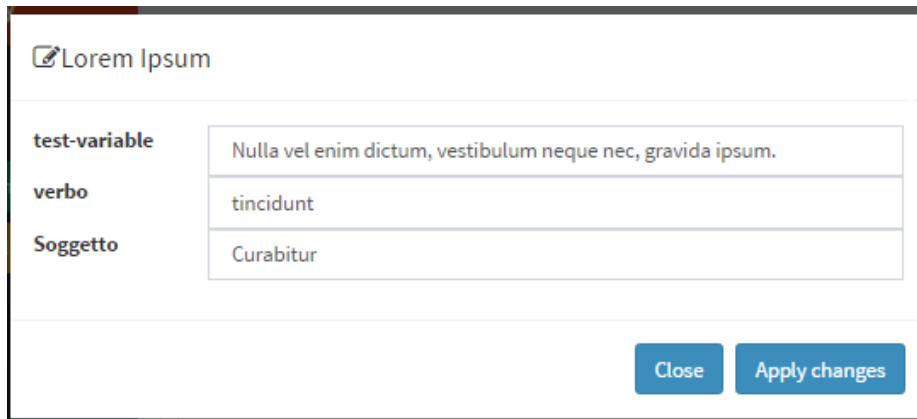


Figura 2.12: View del documento *Lista della spesa* dopo l'inclusione del template *Lorem Ipsum*

Come si nota dalla figura 2.12 il template è evidenziato con colore diverso dal resto del testo così da rendere più facile ed intuitivo il riconoscimento delle istanze. Inoltre la parte variabile è di un colore più scuro rispetto al colore usato per il template, il motivo è sempre lo stesso ossia evidenziare la parte “variabile” dalla parte “fissa”.

2.4.7 Modifica variabile

Una delle funzionalità principali di *Pakkery* è senza dubbio la modifica di una variabile. Ipotizziamo di modificare una variabile contenuta nell'istanza del template aggiunto *Lorem Ipsum*. Cliccando con il tasto destro del mouse sull'istanza, verrà mostrato un menù a tendina con l'oggetto *Edit Variables* ed un successivo click su questo elemento porterà l'utente al seguente modale che contiene la lista delle variabili contenute in quella specifica istanza selezionata:



The screenshot shows a modal window titled "Lorem Ipsum" with a pencil icon. It contains a table of variables:

test-variable	Nulla vel enim dictum, vestibulum neque nec, gravida ipsum.
verbo	tincidunt
Soggetto	Curabitur

At the bottom right, there are two buttons: "Close" and "Apply changes".

Figura 2.13: modale per la modifica delle variabili relative all'istanza del template *Lorem Ipsum*

Si procede al click di *Apply changes* in modo da rendere effettive le modifiche sul documento. Conclusa la fase di modifica o creazione del documento si può procedere al salvataggio del documento che potrà essere reperibile dalla lista dei documenti vista in precedenza.

2.4.8 Nuovo Template

Un template non è altro che un documento su cui si possono definire delle *variabili*. Anch'esso ha bisogno di un titolo per la creazione e in questo caso è anche richiesta una descrizione riguardo lo specifico utilizzo del template che si sta per creare. Questa modalità di inserimento di dati viene fatta, anche qui, utilizzando un modale. Ipotizziamo di voler creare il template *Lorem Ipsum* utilizzato nella sezione precedente ed incluso nel documento *Lista della spesa*. L'editor che viene mostrato è il seguente:

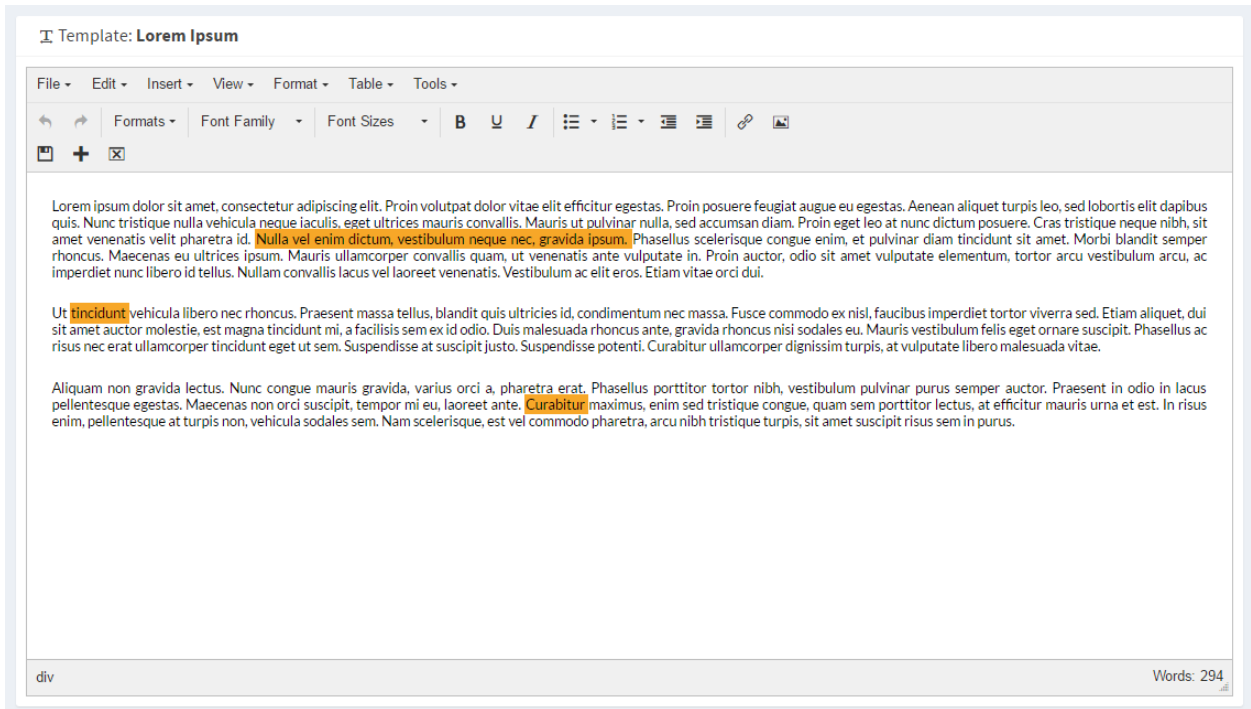


Figura 2.14: Editor per la creazione di un nuovo template dal titolo *Lorem Ipsum* con tre variabili inizializzate

La toolbar dedicata alle funzionalità di Pakkery offre, oltre al pulsante per il salvataggio, due pulsanti per la gestione delle variabili: l'aggiunta (**+**) e la rimozione (**✕**). La rimozione avviene semplicemente posizionando il focus sulla variabile che si vuole rimuovere (si rimuove il ruolo di variabile non il testo che identifica la variabile) e cliccando il bottone relativo alla rimozione oppure attraverso il menù a tendina che viene mostrato al click destro del mouse. L'aggiunta di una variabile, invece, avviene mediante la selezione nel testo della parte che si vuole "promuovere" a variabile e cliccando sul tasto **+**. Tale click porta l'utente ad interagire con il seguente modale che semplicemente chiede il nome della variabile con relativo controllo sul nome per verificarne l'unicità nel template:

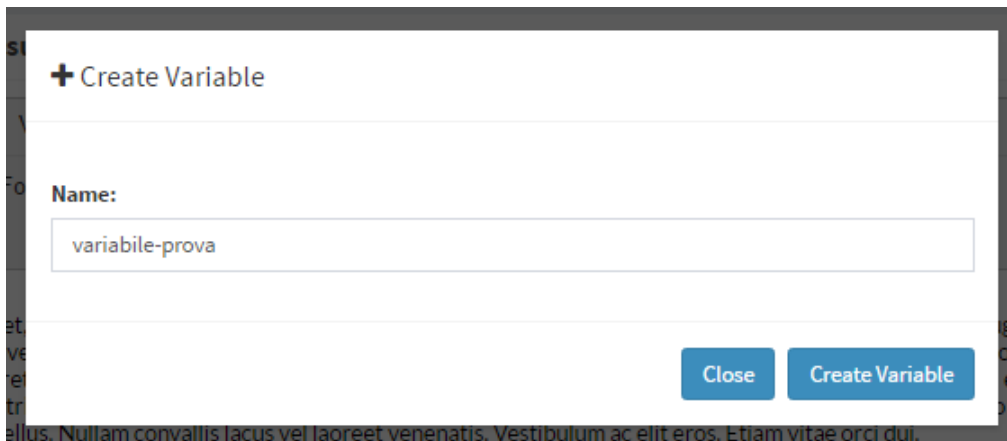


Figura 2.15: Modal per la creazione della variabile dal nome *variabile-prova*

Scelto il nome ed effettuato il controllo su esso, la variabile viene creata ed identificata visivamente da un colore di sfondo arancio. D’ora in avanti, quando l’utente che sceglie di utilizzare questo template e di modificare il contenuto della variabile appena creata dovrà essere consapevole che il contenuto che inserirà nella parte “variabile” sarà salvato nelle eventuali modifiche future mentre ciò che non fa parte della parte “variabile”, ossia la parte “fissa”, sarà perso nell’aggiornamento alla nuova versione del template.

2.4.9 Aggiornamento di un’istanza

Quando un template viene modificato (viene aggiunta o rimossa una variabile, un immagine, un testo, ...) tale modifica può essere propagata nei documenti in cui l’istanza relativa al template in oggetto prende parte. Perciò si ha bisogno di un meccanismo che avvisi l’utente di una modifica e quindi far decidere a questi la propagazione della modifica sul proprio documento. Immaginiamo di aver modificato il template *Lorem Ipsum* usato nel documento precedentemente creato, *Lista della spesa*. L’utente seleziona tale documento e l’interfaccia mostra, attraverso un “alert” (fig. 2.16) la presenza di modifiche e permette un’anteprima di queste.

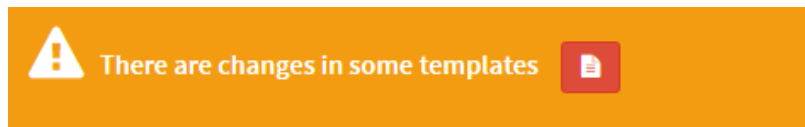



Figura 2.16: Alert per la presenza di modifiche nelle istanze contenute nel documento

Ciò che permette all’utente di poter visualizzare le modifiche è il click sul *button* situato nell’*alert* (). Il successivo click sul *button* appena descritto mostrerà all’utente un modale che visualizza il documento per intero con la differenza che le istanze che necessitano di modifiche sono evidenziate rispetto al resto del testo. Sono presenti dei bottoni che hanno il compito di applicare, e salvare, gli aggiornamenti. Se l’utente che ha usato il template ha inserito contenuti personali nella parte *variabile* questi contenuti verranno presi e salvati nella nuova versione, mentre per quanto riguarda le modifiche fatte dall’utente sull’istanza, ossia ciò che non rientra nella parte *variabile*, verrà sovrascritto dalla nuova versione del template. Nell’esempio seguente (fig. 2.17) è presente un solo template aggiornabile ma nel caso i template utilizzati nel documento fossero più di uno è possibile aggiornarli tutti mediante il click di *Apply all changes* (anche in questo caso il contenuto di ogni parte variabile verrà sostituito e l’aggiornamento sarà immediatamente salvato).

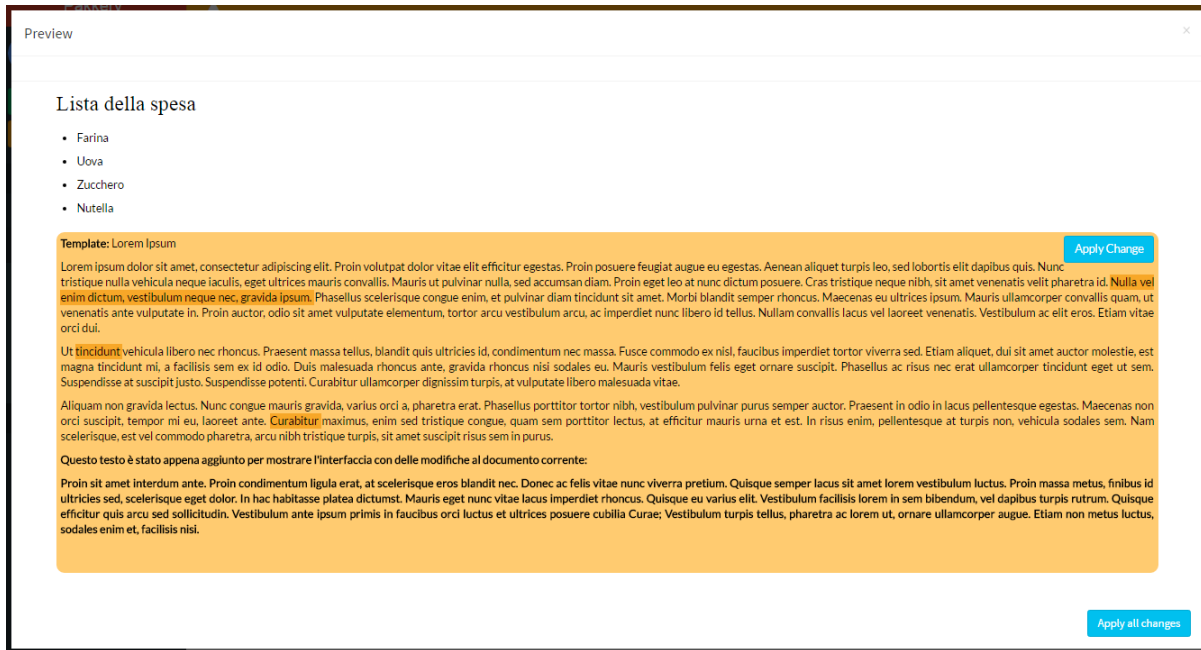


Figura 2.17: modale per la preview del documento *Lista della spesa* dove viene visualizzato la modifica al template *Lorem Ipsum*

2.5 Interazioni

L'applicazione è strutturata su tre livelli quali:

- **Interfaccia**, tutto ciò con cui l'utente può interagire ossia codice HTML e CSS;
- **Elaborazioni e richieste**, o per meglio dire il core dell'applicativo; questa parte verrà trattata più nel dettaglio nel capitolo 3;
- **Storage**, livello che permette il salvataggio e il recupero delle risorse. È composto da due cartelle principali che hanno lo scopo di contenere i documenti e i template.

Di seguito sono descritti, con un basso livello di dettaglio, i passaggi necessari per alcune operazioni fondamentali.

2.5.1 Caricamento di un Documento

La seguente illustrazione (fig. 2.18) mostra i passaggi necessari, compiuti dal sistema, per il caricamento del documento scelto. La parte più elaborata consiste, in caso di modifiche nei template, nell'applicare i cambiamenti alle istanze presenti nel documento. Controlli e sostituzioni sono effettuati a livello client-side e questo meccanismo permette all'utente di poter decidere quali istanze aggiornare. Lo *storage* viene interrogato due volte:

- per ottenere il documento che si è scelti di visualizzare;

- per caricare i template ed eseguire controlli, dopo che il sistema ha individuato le istanze all'interno del documento.

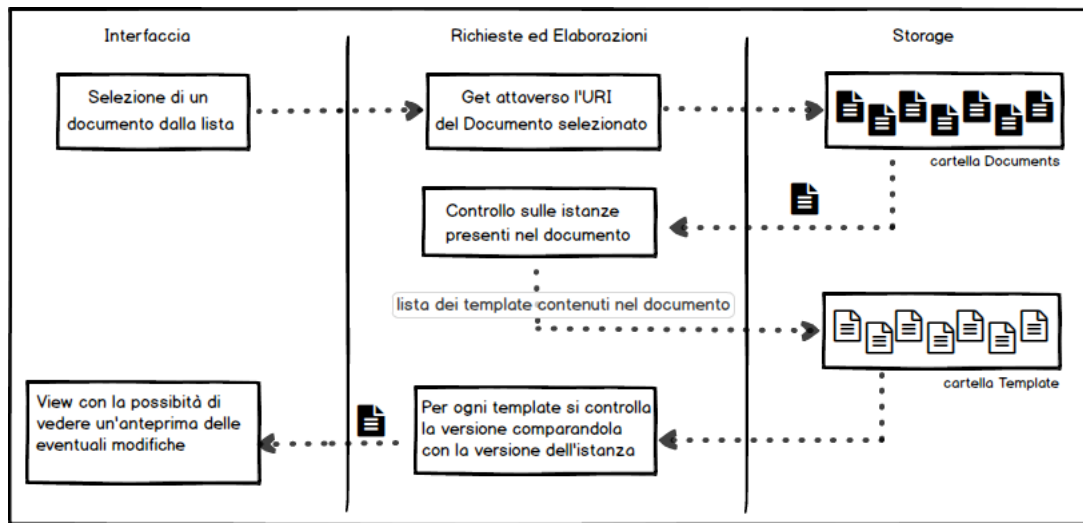


Figura 2.18: Caricamento di un documento

2.5.2 Aggiornamento di un'istanza

Complementare alla fase precedentemente descritta è l'aggiornamento di un'istanza attraverso il modale dedicato (fig. 2.17). La parte sostanziale dell'operazione avviene a livello *Richieste ed Elaborazioni* dove, una volta che l'utente ha scelto quale istanza aggiornare, questa viene, attraverso il nome del template relativo, comunicata a livello *javascript* che ne richiede il caricamento in modo da poter sostituire le variabili dell'istanza nel template completando l'aggiornamento. Ottenuto l'oggetto aggiornato si procede al caricamento dello stesso nell'interfaccia e al suo salvataggio nello *storage*.

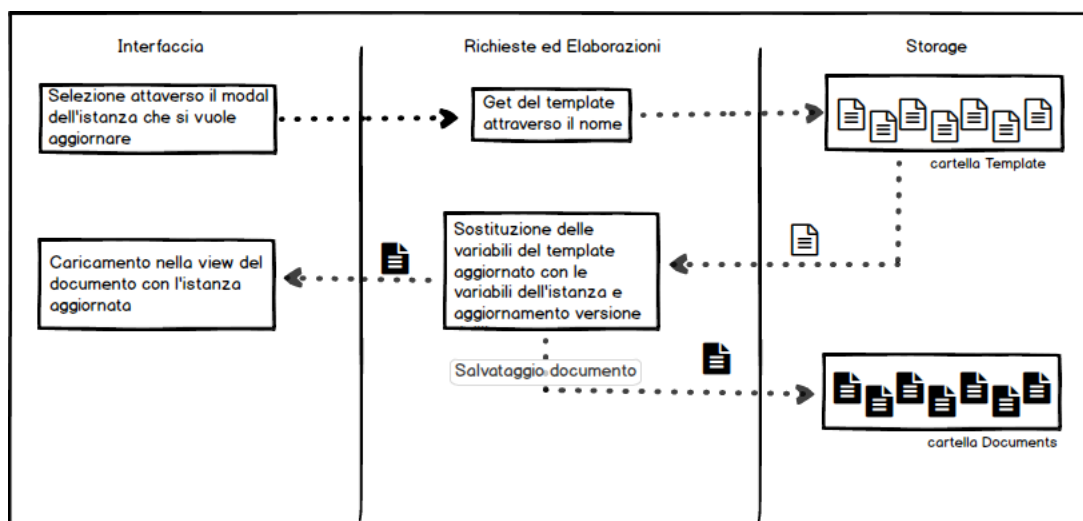


Figura 2.19: Aggiornamento di una specifica istanza

Capitolo 3

Implementazione

Conclusa la parte riguardante l'interfaccia e le sue funzionalità passiamo a spiegare come queste funzionalità siano possibili. Passiamo dunque alla spiegazione del codice sorgente. Il codice è interamente scritto nel linguaggio client-side JavaScript con l'utilizzo del framework jQuery, mentre per la fase di storage si è usato il linguaggio server-side PHP.

3.1 TinyMCE

L'elemento principale di *Pakkery*, senza il quale lo sviluppo di corrente progetto sarebbe risultato più ostico e difficile, è l'editor TinyMCE.

TinyMCE (Tiny Moxiecode Content Editor) è una piattaforma indipendente basata sul concetto di WYSIWYG, “quello che vedi è quello che otterrai”. È interamente scritto in JavaScript e diffuso come software libero sotto la licenza LGPL da Ephox. Questo software ci consente di convertire un campo textarea dell'HTML, o altri elementi di esso, in un'istanza di editor. L'architettura di Tiny è molto semplice e per inizializzare un'istanza basta inserire nel codice di configurazione il selettore dell'elemento che si è scelti di convertire in editor. Fortunatamente la documentazione al riguardo è abbastanza esaustiva per le necessità di un programmatore e la realizzazione di qualsiasi software che lo utilizzi [16] [17].

3.2 Gestione degli eventi


Con i moduli *main.js*, *editDoc.js*, *viewDocument.js* e *viewTemplate.js* si gestiscono i vari eventi che si possono verificare durante l'esecuzione dell'applicazione come, ad esempio, il click di un bottone per la creazione di un Documento.

Per indicare un evento su un determinato elemento si userà la seguente sintassi:

nomeEvento: “selettoreCSS”

Quindi *click*: “.open” indica il click su un elemento con classe “open”.

3.2.1 click: “#createVarModalButton”

Selezionata la porzione di testo che si è scelti di “promuovere” a variabile, l’utente applica questo cambiamento attraverso il button  situato nell’editor per la creazione/modifica del template (fig. 2.14). Effettuato il click, l’utente si ritrova il modale di figura 2.15 ed inserito in nome della variabile con successivo click si solleva l’evento che nelle seguenti righe andremo a descrivere. Preso il nome della variabile ed effettuato il controllo che non sia una stringa vuota oppure il nome non sia già in uso da un’altra variabile contenuta nell’istanza corrente (attraverso la funzione `checkName(name)`), si prosegue con l’identificazione del nodo di inizio e fine della selezione.

```
var name = $("#nameVar").val();
if(name == "") return;
if(checkName(name)) {
    $("#label.alert_error").html("Name already used");
    $("#nameVar").val("");
    return;
}
$("#label.alert_error").html("");
var start = tinymce.activeEditor.selection.getStart();
var end = tinymce.activeEditor.selection.getEnd();
```

Dal nodo `start` e dal nodo `end` possiamo controllare il tipo di variabile: *semplice* o *complessa*. Una variabile *semplice* è una variabile che ha nodo `start` e `end` che coincidono e quindi:

```
if(start == end) {
    var text = tinymce.activeEditor.selection.getContent();
    if(text && text.length > 0) {
        tinymce.activeEditor.selection.setContent(
            "<span style='height: 1px'></span><span title='" +
            name + "' data-template-variable='" + name + "'> " +
            text + " </span><span style='height: 1px'></span> ");
    }
}
```

Identificato il testo selezionato si costruisce la struttura descritta nel capitolo precedente aggiungendo degli *span* all’inizio e alla fine in modo da permettere l’aggiunta di testo ai lati della selezione. Se i due nodi non coincidono, siamo nel caso di variabile *complessa*. Per prima cosa si identifica il `tagName` del nodo iniziale appartenente alla selezione. Nel caso in cui il tag fosse un P, LI o UL si parte dall’`anchorNode` e si concatena il testo di tutti i fratelli di questo nodo fino a giungere al `focusNode`. Ricavato il testo si passa alla rimozione dei nodi coinvolti nella selezione e alla loro sostituzione con la struttura annessa di testo appena elaborato.

```
else {
    var x;
```

```

var textVar;
var elem = tinyMCE.activeEditor.selection.getSel().anchorNode
    .parentElement.nodeName;
if(elem == "P" || elem == "LI" || elem == "UL") {
    var sel = tinyMCE.activeEditor.selection.getSel();
    var anchor = sel.anchorNode.parentElement;
    var text = anchor.outerHTML;
    var focus = sel.focusNode.parentElement;
    var sibling = sel.anchorNode.parentElement.nextSibling;
    while(sibling.outerHTML !== focus.outerHTML) {
        text = text + sibling.outerHTML;
        prevSibling = sibling;
        sibling = sibling.nextSibling;
        tinyMCE.activeEditor.dom.remove(prevSibling);
    }
    text = text + focus.outerHTML;
    tinyMCE.activeEditor.dom.remove(anchor);
    tinyMCE.activeEditor.dom.remove(focus);
    tinyMCE.activeEditor.insertContent(
        "<p style='height: 1px'></p><div title='" + name +
        "' data-template-variable='" + name + "'> " + text +
        " </div><p style='height: 1px'></p>");
}

```

Invece quando il *tagName* è un elemento della tabella si risale il DOM alla ricerca del nodo che ha come *tagName* TABLE. In tal modo si può estrarre il contenuto all'interno della tabella e utilizzarlo per la creazione della variabile con successiva sostituzione.

```

else if(elem == "TD" || elem == "TR" || elem == "TBODY" ||
    elem == "TABLE") {
    x = tinyMCE.activeEditor.selection.getSel().anchorNode
        .parentElement;

    while(1) {
        if(x.nodeName == "TABLE") break;
        x = x.parentElement;
    }
    textVar = x.outerHTML;
    tinyMCE.activeEditor.dom.remove(x);
    tinyMCE.activeEditor.insertContent(
        "<p style='height: 1px'></p><div title='" + name +
        "' data-template-variable='" + name + "'> " + textVar +
        " </div><p style='height: 1px'></p>");
}
}

```

Le ultime due istruzioni servono a resettare il campo input relativo al nome e a nascondere il modale alla fine dell'elaborazione.

```
$("#nameVar").val("");
```

```
$("#createVar").modal("hide");
```

3.2.2 click: “#applyEditVariable”

È l'evento che permette di applicare le modifiche alle singole variabili. Grazie alla funzione *editVars*, che costruisce il modale per la modifica con i vari input, ne parleremo più avanti, al click del button si risale il DOM fino ad incontrare o un elemento con id (caso newDoc) o con attributo *data-template-document* (caso editDoc).

```
var x = tinymce.get(idCurrentEditor).selection.getSel()
    .extentNode.parentElement;
while(1) {
    if(x.attributes.length !== 0 && ( x.attributes[0].nodeName == "id" ||
        x.attributes[0].nodeName == "data-document")) break;
    x = x.parentElement;
}
```

In entrambi i casi trovato il nodo si individuano le due possibili tipologie di variabili:

- semplici: (*itemSimple*) ossia quelle variabili il cui tag è uno span e quindi hanno come casella di input un semplice input-text identificati con la classe *.inputVar*
- complesse: (*itemComplex*) quelle variabili che possono essere dei frammenti di HTML più strutturati con un editor inline di tiny e identificati con la classe *.tiny_editVar*

I due casi vanno affrontati in maniera diversa in quanto hanno complessità diversa. Infatti, mentre per l'input-text basta un semplice *.val()* per prenderne il valore contenuto, per gli *itemComplex* serve l'id dell'editor.

```
var doc = "<div>" + $(x).html() + "</div>";
var data_doc = urlDoc + currentDoc;
var docDom = $.parseHTML(doc);
var nameInstance = $(this).attr("template");
var itemSimple = $(".inputVar");
var itemComplex = $(".tiny_editVar");
var newVar = [];
if(itemSimple.length > 0) {
    for(var i = 0; i < itemSimple.length; i++) {
        var name = $(itemSimple[i]).attr("name");
        var content = $(itemSimple[i]).val();
        var item = {
            nameVar: name,
            contentVar: content
        };
        newVar.push(item);
    }
}
```

Perciò attraverso la classe `.tiny_editVar` si ricava il nome della variabile e l'id dell'editor che ne permetteva la modifica. Infatti tiny associa ad ogni istanza di editor un id che non è altro che la concatenazione della stringa `mce_` con un numero progressivo che conta il numero di editor istanziati fino a quel momento. Per il ricavare il contenuto dei vari editor si è quindi proceduto in questo modo: per ogni elemento con la classe `.tiny_editVar` e `name` uguale al nome della variabile che si sta esaminando si estrae l'id relativo e di conseguenza si estrarre il contenuto dall'editor.

```

if (itemComplex.length > 0) {
  for (var j = 0; j < itemComplex.length; j++) {
    var name = $(itemComplex[j]).attr("name");
    var idVarTiny = $(".tiny_editVar[name='" + name + "']").attr("id");
    var content = tinymce.get(idVarTiny).getContent();
    var item = {
      nameVar: name,
      contentVar: content
    };
    newVar.push(item);
  }
}

```

Conclusa la fase di immagazzinare le variabile (nome e contenuto) si è passati alla sostituzione nel documento

```

var instance = $(docDom).find("[data-template-name='" +
  nameInstance + "']");
for (var z = 0; z < newVar.length; z++) {
  $(instance).find("[data-template-variable='" +
    newVar[z].nameVar + "']").html(newVar[z].contentVar);
}
$(docDom).find("[data-template-name='" + nameInstance + "']")
  .html($(instance).html());
tinymce.get(idCurrentEditor).setContent(
  "<div data-document='" + data_doc + "'>"
  + $(docDom).html() + "</div>");

```

3.2.3 click: “.applyTemplate”

Quando l'utente decide di applicare gli aggiornamenti di un template all'istanza contenuta nel proprio documento lo fa cliccando il bottone situato nel modale “Preview” (fig.2.17). Questo click solleva un evento che, grazie all'attributo “template” situato all'interno del tag del bottone, permette di applicare le modifiche all'istanza desiderata. Quindi per prima cosa si ricava il nome del template che si è scelti di aggiornare ed insieme all'oggetto DOM del documento corrente (sia l'oggetto aggiornato che quello originale, ossia senza le

modifiche), si richiama la funzione *applyChanges* (ne discuteremo più avanti) che ritorna l'HTML del documento correntemente visualizzato.

```
var template = $(this).attr("template");
var res = applyChanges(docObject.domOriginal,
    docObject.domMod, template);
```

Tale HTML sarà ciò che andrà a sovrascrivere il contenuto degli editor (con e senza la visione dei template). Si ricavano perciò gli *id* dei due editor per permettere il *setContent* del nuovo contenuto con successivo salvataggio sul file system del documento aggiornato e si “ripulisce” il contenuto del modale “Preview” attraverso la rimozione del button che applica l'aggiornamento all'istanza e l'attributo classe che evidenzia l'istanza da aggiornare.

```
$(this).parent().removeAttr("class");
$(this).remove();
var idWithoutTemplate =
    $(".tiny_viewDocWithoutTemplate").attr("id");
tinymce.get(idWithoutTemplate).setContent(res);
var idWithTemplate =
    $(".tiny_viewDocWithTemplate").attr("id");
tinymce.get(idWithTemplate).setContent(res);
saveDocWithUpdate(idWithTemplate);
```

L'ultima operazione da effettuare è un semplice decremento delle modifiche da effettuare e nel caso in cui dopo questa sottrazione il numero è pari a zero si nasconde il modale e l'alert.

```
numChanges--;
if (numChanges == 0) {
    $("#viewModDoc").modal('hide');
    $(".areaAlert").hide();
}
```

3.3 Configurazione Editor

Con *pakkery.js* si è voluto gestire tutto ciò che fosse inerente alla parte dell'editor tinyMCE, ossia le diverse configurazioni dei vari editor: da quello per la view di un documento a quello per la modifica di un template. Come scritto nella sezione inerente alla descrizione di TinyMCE, questo editor inizializza un determinato elemento dell'HTML in un'istanza di Tiny. Per fare ciò si indica il selettore dell'elemento oggetto di tale istanza. Per una documentazione più dettagliata riguarda alla configurazione di TinyMCE vi rimando alla documentazione disponibile sul sito e archivio [16] [17]. Per l'implementazione di *PAkkery* sono serviti otto diversi editor, elencati di seguito:

- **div.tiny_newTemplate**: inizializza l'editor relativo alla creazione di un nuovo Template; tale editor ha tutte le funzionalità dell'editor per la creazione dei documenti ma in aggiunta ha la possibilità di “promuovere” una parte del template a variabile. Questo meccanismo per la promozione prevede di “avvolgere” il testo selezionato in base alla forma stabilita. La variabile appena creata avrà uno stile che la identificherà dal resto del testo del template. Si può modificare tale stile attraverso la modifica del css *newTemplate.css*;
- **div.tiny_editTemplate**: inizializza l'editor riguardante la modifica di un template; segue le stesse funzionalità dell'editor descritto al punto precedente;
- **div.tiny_viewTemplate**: inizializza l'editor per la visualizzazione dei template. Questa volta l'editor è in sola lettura (campo `readonly: true`) e i campi "variabile" sono evidenziati attraverso un certo stile modificabile nel file css: *viewTemplate.css*;
- **div.tiny_viewDocWithTemplate**: inizializza l'editor per la visualizzazione del documento con in sola lettura (campo `readonly: true`). Le parti istanze di template e le variabili sono evidenziate (attraverso un certo stile modificabile nel file css *viewDocument.css*) per essere notate da chi legge il documento;
- **div.tiny_viewDocWithoutTemplate**: è sempre usato per inizializzare l'editor in modalità lettura del documento ma non è applicato nessuno stile per evidenziare parti del documento;
- **div.tiny_newDoc**: inizializza l'editor per la creazione di un nuovo documento; è un “sottocaso” dell'editor utilizzato per la creazione di un nuovo template con l'aggiunta della funzionalità di poter inserire istanza di template all'interno del documento che si sta creando. Anche qui le parti più significative, vale a dire le istanze e le variabili, sono contrassegnate con uno stile diverso di css modificabile dal file *viewDocumentWithTemplate.css*;
- **div.tiny_editDoc**: l'editor per la modifica di un documento ha le stesse caratteristiche dell'editor descritto nel punto precedente;
- **div.tiny_editVar**: è inizializzato all'interno del modale per permettere la modifica di variabili più strutturate (complesse).

Il numero elevato di editor è motivato dal fatto che si voleva dare un compito specifico ad ogni componente in modo da non avere conflitti e di poter modificare l'interfaccia dell'editor senza dipendere da altro.

3.4 Funzioni e variabili

Nei moduli *global.js* e *function.js* si raggruppano rispettivamente tutte le variabili globali e le funzioni utilizzate durante la fase di implementazione. In particolare si possono gestire i vari messaggi di notifica attraverso l'aggiunta di codice HTML nell'oggetto *multiNotify* (in *global.js*) e attraverso la chiamata della funzione *openNotify* (in *function.js*).

3.4.1 viewChanges

```
function viewChanges(doc)
```

Una delle funzioni principali e più corpose è la *viewChanges* che prende in input l'URI del documento e fornisce in output un oggetto composto da sette campi. Grazie a questa è possibile verificare se le istanze contenute all'interno del documento passato come parametro richiedono un aggiornamento, ossia se la versione del template è differente da quella dell'istanza e nel caso ci siano aggiornamenti provvede a costruire l'HTML del documento aggiornato. L'oggetto output di *viewChanges* fornisce le "informazioni" contenute nella seguente tabella:

Campo	Valore
mod	(booleano) identifica la presenza di aggiornamenti nelle istanze
bodyOriginal	HTML originale del documento che si è scelti di caricare
bodyModified	HTML del documento che contiene le eventuali modifiche
domOriginal	oggetto DOM del documento senza modifiche
domMod	oggetto DOM del documento con le modifiche
instance	array che contiene i nomi dei template le cui istanze nel documento fanno riferimento
template	array che contiene i template che richiedono aggiornamento

Analizziamo il codice di questa funzione. Si ricava il DOM del documento e si inizializza l'oggetto di ritorno.

```
var docHtml = $.parseHTML(get(doc));
var ret = {
  mod: false,
  bodyOriginal: $(docHtml).html(),
  bodyModified: "",
  domOriginal: $(docHtml),
```

```

domMod: "",
instance: [],
template: []
};

```

Si individuano tutte le istanze presenti nel documento.

```

var instance = $(docHtml).find("[data-template-instance]");
var numInstance = instance.length;

```

Per ogni istanza individuata si confronta la versione dell'istanza con il suo relativo template.

```

for (var i = 0; i < numInstance; i++){
    var uriTemplate = $(instance[i]).attr("data-template-instance");
    var nameTemplate = $(instance[i]).attr("data-template-name");
    (ret.instance).push(nameTemplate);
    var currentVersion = $(instance[i]).attr("data-template-instance-version");
    var templateHtml = $.parseHTML(get(uriTemplate));
    var versionTemplate = $(templateHtml).attr("data-template-version");

```

e se le due versioni sono differenti, ossia il template la cui istanza fa riferimento ha subito delle modifiche, si passa ad individuare le variabili (nome e contenuto) presenti nel documento in modo da poterle salvare e poi andare a sostituire nella versione aggiornata dell'istanza.

```

if (versionTemplate != currentVersion){
    ret.mod = true;
    (ret.template).push(nameTemplate);
    $(instance[i]).addClass("updated");
    var instanceVariable = $(instance[i]).find(
        "span[data-template-variable]");
    var numInstanceVariable = instanceVariable.length;
    var arrayInstanceVariable = [];
    for (j = 0; j < numInstanceVariable; j++){
        var variableName = $(instanceVariable[j]).attr(
            "data-template-variable");
        var variableContent = $(instanceVariable[j]).text();
        arrayInstanceVariable[j] = {
            nameVar: variableName,
            contentVar: variableContent
        };
    }
}

```

Costruito l'array (*arrayInstanceVariable*) contenente le variabili della vecchia istanza si procede con la sostituzione ed applicazione del “button” che sarà visibile nel modale che visualizza le modifiche e che permetterà la realizzazione dell'aggiornamento sul documento.

```

for (var j = 0; j < numInstanceVariable; j++){

```

```

    var templateVariables = $(templateHtml);
    $(templateVariables).find("span[data-template-variable='" +
        arrayInstanceVariable[j].nameVar + "']");
    $(templateVariables).text(arrayInstanceVariable[j].contentVar);
}
var newInstance = $(templateHtml).html();
$(instance[i]).html(newInstance);
$(instance[i]).attr("data-template-instance-version",
    versionTemplate);
var oldInstance = $(docHtml).find("div[data-template-name='" +
    nameTemplate + "']");
$(oldInstance).attr("data-template-instance-version", versionTemplate);
$(oldInstance).html(newInstance);
$(oldInstance).prepend("<div class='info-changes'><b>Name Template: </b>" +
    nameTemplate + "<button style='float:right' template='" + nameTemplate +
    "' class='applyTemplate btn btn-info'>Apply Change</button>");
arrayInstanceVariable = null;
}
}
ret.bodyModified = $(docHtml).html();
ret.domMod = $(docHtml);
return ret;

```

3.4.2 applyChanges

```
function applyChanges(doc, docMod, template)
```

Se *viewChanges* permettere di verificare la presenza di modifiche, la funzione che le applica è *applyChanges* che viene richiamata al click del bottone *apply Change* nel modale *preview*. Tale funzione prende in input l'oggetto DOM del documento originale, l'oggetto DOM del documento con applicate le modifiche e il nome del template la cui istanza si vuole aggiornare.

Dopo aver individuato l'istanza nel DOM aggiornato (*newInstance*) e l'istanza nel DOM non aggiornato (*oldInstance*) nel documento originale, se ne ricava la versione ultima del template in modo da aggiornare il numero che indica in numero di aggiornamento.

```

    var newInstance = $(docMod).find("div[data-template-name='"+template + "']");
    var versione = $(newInstance).attr("data-template-instance-version");
    var oldInstance = $(doc).find("div[data-template-name='"+template + "']");
    $(oldInstance).attr("data-template-instance-version", versione);
    $(oldInstance).html($(newInstance).html());

```

Si rimuove l'attributo *class* dalla nuova istanza per distinguere che questo template è stato modificato e quindi il cambio di colore conferma l'avvenuto aggiornamento.

```

$(newInstance).removeAttr("class");
$(docMod).find(".info-changes").remove();

```

```
return $(docMod).html();
```

3.4.3 removeVar

```
function removeVar(ed)
```

Creata una variabile si ha la possibilità di rimuoverla attraverso *removeVar* che in input prendere l'oggetto editor su cui viene chiamata la funzione. La rimozione avviene risalendo il DOM fino a trovare un nodo che abbia come secondo attributo *data-template-variable* che appunto identifica una variabile secondo la forma descritta nei capitoli precedenti.

```
var x = ed.selection.getSel().extentNode.parentElement;
while(1) {
    if(x.attributes[1].nodeName == "data-template-variable")
        break;
    x = x.parentElement;
}
```

In tal modo, una volta trovato il nodo variabile si procede con il salvare il frammento di HTML contenuto in questo nodo e alla rimozione dello stesso. In fine si inserisce nello stesso punto il frammento prelevato prima della cancellazione del nodo.

```
var text = x.innerHTML;
ed.dom.remove(x);
tinyMCE.activeEditor.insertContent(text);
```

3.4.4 editVars

```
function editVars(ed)
```

L'*editVars* oltre che permettere la modifica di variabili ha il compito di costruire il modale con l'elenco delle variabili all'interno dell'istanza. Il metodo utilizzato è più o meno quello utilizzato per la rimozione ma in questo caso si risale il DOM fino a trovare un elemento che abbia come primo attributo. *data-template-instance*

```
$("#contentEditVar").empty();
var x = ed.selection.getSel().extentNode.parentElement;
while(1) {
    if(x.attributes[0].nodeName == "data-template-instance")
        break;
    x = x.parentElement;
}
```

Individuato il nodo si procede con il selezionare le variabili al suo interno e a settare l'attributo *template* del bottone che apporterà le modifiche. Per ogni variabile trovate e in base all'elemento che individua la variabile si costruisce l'area per la modifica: se la variabile è "avvolta" da uno *span* abbiamo pensato bastasse un semplice input-text mentre in altri casi, con strutture più complesse di HTML (tabelle, liste, ...), si inizializza

un editor in versione *inline* che ha come classe il nome della variabile in modo da poterne rilevare l'id dell'editor relativo.

```

var nameInstance = $(x).attr("data-template-name");
$("#applyEditVariable").attr("template",nameInstance);
$("#nomeTemplate").html(nameInstance);
var vars = $(x).find("[data-template-variable]");
var numVars = vars.length;
for (var i = 0; i<numVars; i++){
    var nameVar = $(vars[i]).attr("data-template-variable");
    var contentVar = $(vars[i]).html();
    var tipo = $(vars[i]).prop("tagName");
    var content;
    if(tipo == "SPAN"){
        content = "<input name='"+nameVar+"' class='form-control inputVar '"+
            "type='text ' value='"+contentVar+"'>";
    }else{
        content = "<div name='"+nameVar+"' class='tiny_editVar'>"+
            contentVar+"</div>";
    }
    $("#contentEditVar").append("<div class='row'><div class='col-md-1'>"
        "<label>"+nameVar + "</label></div><div class='col-md-11'>"+
        content+"</div></div>");
}
$("#contentEditVar").append("<script src='myFiles/js/tinyEditVar.js'>"+
    "</script >");
$("#modalVarEdit").modal('show');

```

Non abbiamo discusso tutte le caratteristiche di *Pakkery*, in termini di implementazione, in quanto alcune sono ovvie e quindi non necessitano di una descrizione approfondita ma il lettore potrà reperire il codice presso attraverso la repository **GitHub** al seguente link: <https://github.com/antonioconte/pakkery>.

Capitolo 4

Valutazione

Quella che presentiamo in questo capitolo è una valutazione preliminare del software *Pakkery*. È molto importante, prima di rilasciare il software, far testare lo stesso ad utenti con diverse esperienze e abilità in modo da poter stilare una serie di modifiche da effettuare e quindi rendere l'applicativo più conforme agli usi e alle abitudini dell'utente medio.

4.1 Obiettivi

I soggetti in esame sono stati dieci, divisi in due gruppi di cinque. Analizzando i dieci soggetti possiamo distinguere due tipi di abilità: “avanzata” (4), ossia quegli utenti che hanno esperienza con il mondo informatico, e “intermedia” (6), utenti che hanno come unico strumento di scrittura di documenti il software *Word*. La divisione nei due gruppi era così formata: due componenti “avanzati” e tre “intermedi” per ogni gruppo.

I compiti sono stati eseguiti non in contemporanea ma in successione in quanto la terminazione di un compito serviva a far iniziare il successivo. I compiti eseguiti sono stati cinque, ognuno eseguito due volte, prima da un gruppo e poi dall'altro, in modo da avere una valutazione più precisa per ogni compito. Segue una lista dei compiti assegnati:

1. *Crea un nuovo template dal titolo “Profilo” in cui ti descrivi e costruisci delle variabili riguardanti il tuo nome e cognome;*
2. *Crea un nuovo documento dal titolo “Eccomi” ed inserisci il template “Profilo” modificando le variabili in base ai tuoi dati;*
3. *Crea un nuovo template dal titolo “Famiglia” e costruisci una tabella con campi di intestazione “nome”, “cognome”, “ruolo” e crea la variabile “table” che si riferisce alla tabella;*
4. *Modifica il template “Profilo” aggiungendo delle esperienze personali (hobby, ambizioni, ...) ed elimina la variabile “nome” all'interno del template;*

5. *Seleziona il documento “Eccomi!” e applica le eventuali modifiche, dopodiché passa alla creazione di un nuovo documento ed inserisci il template dal nome “Famiglia” modificando la tabella con i dati della tua famiglia.*

Al termine del compito gli utenti hanno compilato un questionario, in forma anonima, composto da sette domande in cui gli veniva chiesto di dare un giudizio (da uno a cinque) al compito assegnato e all'utilizzo di *Pakkery*. Di seguito sono riportate le domande presenti sul questionario compilato dai soggetti del test:

1. *Quanto valuti la facilità con cui hai portato a termine il tuo task? (1 molto difficile, ..., 5 abbastanza semplice)*
2. *Quanto valuti l'usabilità dell'interfaccia?*
3. *Quanto valuti la comodità di avere dei template?*
4. *Gli strumenti utilizzati per la creazione del documento sono stati sufficienti per dare stile al documento creato? (1 assolutamente no, ..., 5 assolutamente si)*
5. *È stato facile creare/modificare una variabile? (1 molto difficile, ..., 5 abbastanza semplice)*
6. *Useresti questo per la creazione dei tuoi documenti? (1 assolutamente no, ..., 5 assolutamente si)*
7. *Consigliaresti ad altri di usare questo applicativo? (1 assolutamente no, ..., 5 assolutamente si)*

Oltre a queste sette domande, il questionario chiedeva all'utente le funzionalità che si sarebbero potute aggiungere per rendere l'applicativo più conforme alle loro esigenze. La domanda era a risposta aperta ed opzionale.

4.2 Risultati

Dai risultati ottenuti abbiamo notato che gli utenti hanno risposto in maniera positiva alle domande del questionario. Infatti la media delle valutazioni è circa 3,1. Un valore discreto ma senz'altro migliorabile. La media delle valutazioni in base ai task (fig.4.1) ha fatto notare che tutti questi sono stati valutati in maniera positiva (per ogni task il valore medio della valutazione è superiore, o uguale, al 3). Nello specifico gli utenti che hanno eseguito i task numero 1 e 5 l'hanno valutato con un valore medio pari rispettivamente a 3,2 e 3,53.

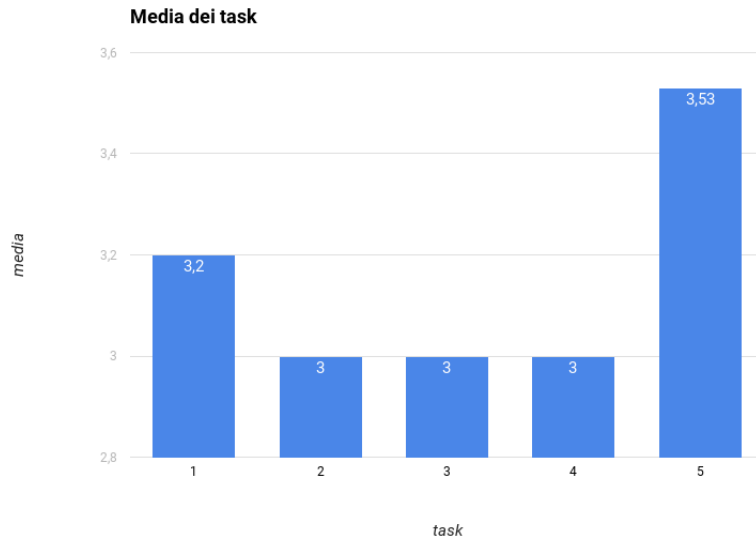


Figura 4.1: Media delle valutazioni in relazione ai task

Per quanto riguarda le valutazioni in base alle domande (fig.4.2), gli utenti hanno risposto positivamente alla domanda numero 4, ossia quella relativa agli strumenti messi a loro disposizione per la creazione del loro documento. Dal grafico si nota che gli utenti non hanno ben inteso l'utilizzo dei template (domanda numero 5) ed un loro possibile ruolo all'interno dei documenti.

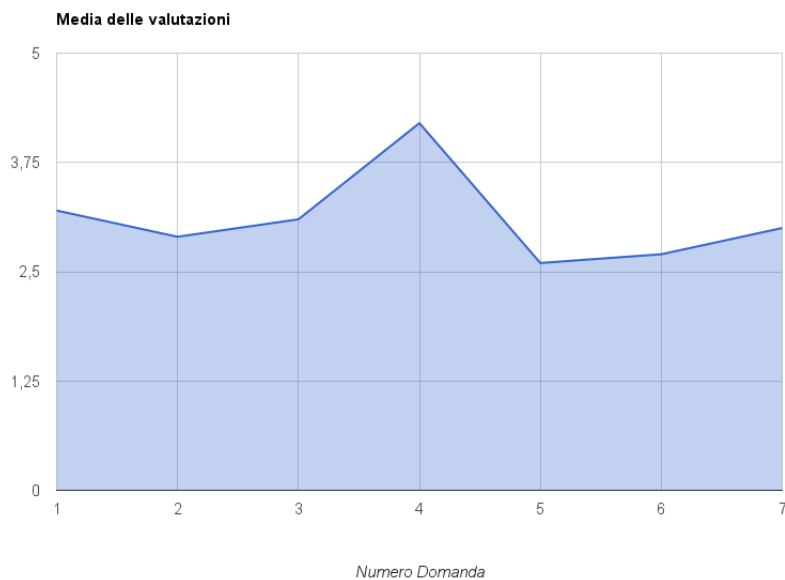


Figura 4.2: Media delle valutazioni in relazione alle domande

La tabella seguente mostra le valutazioni dei dieci soggetti. Ogni cella contiene due valutazioni (ogni task è stato svolto due volte, un componente per gruppo) rispetto al relativo task e alla domanda.

Domanda	Task				
	1	2	3	4	5
1 - Facilità	3 - 2	4 - 2	3 - 3	5 - 4	3 - 3
2 - Usabilità	3 - 3	4 - 3	3 - 2	3 - 3	2 - 3
3 - Comodità	5 - 2	4 - 3	3 - 3	3 - 2	4 - 2
4 - Strumenti	4 - 4	5 - 4	4 - 4	4 - 5	4 - 4
5 - Variabile	4 - 2	3 - 2	2 - 2	3 - 3	2 - 3
6 - Usi futuri	2 - 1	2 - 3	3 - 3	3 - 3	4 - 3
7 - Usi futuri	2 - 3	3 - 3	3 - 3	4 - 2	4 - 3

Tabella 4.1: Valutazioni dei dieci soggetti del test

Riguardo la domanda aperta, gli utenti hanno evidenziato un pó di difficoltà nel trovare i bottoni per l'aggiunta di template o la creazione di una variabile ed hanno suggerito di colorare o ingrandire questi bottoni in modo da poterli identificare meglio al primo utilizzo. Ci sono stati anche consigli riguardo la modalità di aggiornamento dell'istanza in cui veniva suggerito di renderla più intuitiva. Altri hanno sollevato la necessità di avere più strumenti per la creazione di documenti e hanno fatto confusione con il termine *template* considerandolo come un modello di *Bootstrap* ossia un template che riguarda più la presentazione che il contenuto. Questa ambiguità può essere stata causata da una cattiva spiegazione del termine.

Anche se non sufficientemente elevata, questa valutazione preliminare pone i primi aspetti da modificare e migliorare. Andrebbe fatta una valutazione considerando un arco di tempo maggiore e un numero di utenti più elevato.

Conclusioni

La preparazione di documenti richiede molto tempo nel trovare informazioni e recuperarle da parti già scritte che possono essere inserite. Con questo lavoro abbiamo voluto presentare una possibile implementazione del concetto di **riutilizzo controllato** di documenti, partendo dall'idea delle transclusioni concepita da Nelson, e abbiamo proposto una tecnica per facilitare la sua messa in opera. Se da un lato il riuso di documenti o di template, nel caso specifico di questo lavoro, facilita la condivisione di informazioni e velocizza la scrittura di documenti, dall'altro lato esistono problemi riguardanti la gestione degli aggiornamenti successivi ad una modifica sul template utilizzato. *Come ci si comporta quando un template utilizzato in un documento viene rimosso?* Senz'altro c'è stata una modifica e quindi il sistema la mostrerà all'utente. Questo esempio banale e facile da risolvere, perchè basterebbe porre un controllo sull'esistenza del template e poi passare alla ricerca di modifiche, serve a far capire al lettore che il punto principale di tale lavoro non era tanto la gestione delle risorse, ma l'interazione tra di loro.

Nel corso del primo capitolo abbiamo mostrato due possibili implementazioni per la costruzione di documenti: una soluzione proponeva di utilizzare un modulo (1.4) che guidava l'utente alla scrittura e questo approccio, in alcuni casi, poteva rallentare l'utente più "esperto"; l'altra soluzione proposta è stata quella di includere porzioni di testo, attraverso l'URL del documento da inserire, e modificarle tramite una *textarea* (1.5). Da questi due studi abbiamo implementato *Pakkery* in modo da avere entrambi gli aspetti: la modifica di variabile tramite form (2.4.7) oppure tramite la modifica nel testo. La differenza tra *Pakkery* ed il lavoro descritto in 1.5 è sostanzialmente quella che permette all'utente di decidere di applicare le modifiche a seguito di un cambiamento del template utilizzato, infatti quando il sistema *Pakkery* rileva una modifica, permette all'utente di visualizzarla attraverso un'anteprima (2.4.9) e successivamente l'utente può decidere di applicarla nel documento. Uno degli obiettivi futuri che ci prefiggiamo è quello di migliorare la ricerca di un template secondo le necessità dell'utente. Infatti, bisognerebbe fornire più **metadati** riguardo ogni risorsa: per adesso di un template le informazioni disponibili sono quelle relative alla versione, descrizione e nome. Quindi si potrebbero arricchire con ulteriori informazioni quali: parole chiave, autore del template, data creazione e/o modifica, un indice che rappresenta il numero di volte che quel template è stato utilizzato in modo che

l'utente abbia un *feedback* riguardo la scelta dell'inclusione. Sebbene, per adesso, l'inclusione di template riguardi solo i documenti si vorrebbe includere questa funzionalità anche nella creazione di template e quindi facilitare di conseguenza la loro scrittura. Un'altra funzionalità potrebbe essere quella di permettere all'utente di importare ed esportare documenti e template permettendone così un salvataggio in locale oppure dare la possibilità, al creatore, di specificare l'utilizzo del template ossia rendendolo pubblico o privato.

Inoltre abbiamo la necessità di rendere i concetti più intuitivi e semplici per l'utente: a seguito dei test e delle valutazioni svolte abbiamo notato che molti utenti hanno avuto un pó di difficoltà nel comprendere i concetti per l'utilizzo dell'applicativo come ad esempio il significato di variabile oppure di template che in alcuni utenti veniva identificato come un modello per la creazione di siti web ossia una soluzione che mira più alla presentazione che al contenuto. I suggerimenti forniti da questa valutazione devono essere il punto di partenza per miglioramenti futuri per rendere il software, come già detto, più intuitivo agli usi degli utenti.

Bibliografia

- [1] Wikisource, Transclusionione. <https://it.wikisource.org/wiki/Aiuto:Transclusionione>.
- [2] Leuf, Bo, and Ward Cunningham. *The Wiki Way: Quick Collaboration on the Web*. Boston: Addison-Wesley, 2001.
- [3] Haake, Anja, Stephan Lukosch, and Till Schümmer. *Wiki-templates: adding structure support to wikis on demand*, Proceedings of the 2005 international symposium on Wikis. ACM, 2005.
- [4] Rick, Jochen, et al. *Collaborative learning at low cost: CoWeb use in English composition*. Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community. International Society of the Learning Sciences, 2002.
- [5] Meyrowitz, Norman. *Hypertext—does it reduce cholesterol, too?*. From Memex to hypertext. Academic Press Professional, Inc., 1991.
- [6] Wiki choicetree. <http://c2.com/cgi/wiki?WikiChoicetree>. Maggio, 2005.
- [7] Di Iorio, Angelo, and John Lumley. *From XML inclusions to XML transclusions*. Proceedings of the 20th ACM conference on Hypertext and hypermedia. ACM, 2009.
- [8] Nelson, Theodor H. *Literary Machines: The Report On, and Of, Project Xanadu concerning Word Processing, Electronic Publishing, Hypertext, Thinkertoys, Tomorrow's Intellectual Revolution, and Certain Other Topics including Knowledge, Education and Freedom*. Swarthmore, PA: Theodor H. Nelson, 1987.
- [9] Ritter, G. *Template-based creation of electronic document*. Google Patents, US Patent App. 11/368,841, 2007.
- [10] Maurer, Hermann, and Josef Kolbitsch. *Transclusions in an html-based environment*. CIT. Journal of computing and information technology 14.2 (2006): 161-173.
- [11] WOOD, Lauren et al. *Document Object Model (DOM) Level 1 Specification Version 1.0*. <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>. Ottobre, 1998.

- [12] Le Hegaret, Philippe et al. *Document Object Model (DOM)*. <http://www.w3.org/DOM/>. Gennaio, 2005.
- [13] Raggett, Dave et al. *HTML 4.01 Specification*. <http://www.w3.org/TR/html4/>. Dicembre, 1999.
- [14] Netscape Communications. *Javascript Developer Central*. <http://developer.netscape.com/tech/javascript/>, 2004.
- [15] Fielding, R. et al. *Hypertext Transfer Protocol - HTTP/1.1, Request for comments 2616*, 1999.
- [16] Ephox. *TinyMCE Documentation*. <https://www.tinymce.com/docs/>.
- [17] Ephox. *TinyMce Documentation and Wiki*. <http://archive.tinymce.com/wiki.php/TinyMCE>.
- [18] Gayatri, Venkata, and Krishnamurty Pammi. *Agile User Stories*. (2013).
- [19] Almsaeed Studio. *Documentation of AdminLTE bootstrap*. <https://almsaeedstudio.com/themes/AdminLTE/documentation/>.
- [20] W3school, *HTML data-* Attributes*. http://www.w3schools.com/tags/att_global_data.asp.
- [21] Wikipedia, *Cookie*. <https://it.wikipedia.org/wiki/Cookie>.