ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI BOLOGNA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

Corso di Laurea in Informatica per il Management

Un ambiente Web per la conversione di documenti TEI in formato RASH: progettazione ed implementazione

Relatore:

Prof.

Angelo Di Iorio

Presentato da:

Marco Ballardini

Sessione III

Anno Accademico 2021-2022

INDICE

INDICE DELLE FIGURE

Capitolo 1

1 Introduzione

Questa tesi nasce dall'esigenza di permettere ad un utente di poter caricare documenti in formato TEI [1] sulla piattaforma DocuDipity [2], uno strumento interattivo basato sul Web per supportare l'esplorazione e l'analisi di documenti eterogenei; fornisce un'interfaccia di lettura sequenziale con visualizzazioni alternative come SunBurst e viste basate su alberi.

Affinché l'utente possa usufruire di tali funzionalità è necessario introdurre un intermediario in grado di modulare i documenti utente al formato e alla struttura che DocuDipity è in grado di interpretare.

Nello specifico DocuDipity espone un'API[1] di caricamento per collezioni di documenti, quest'ultime affinché possano essere caricate e manipolate è necessario che presentino documenti in formato RASH [3] ospitati in apposite sottodirectories, e un file Manifest in formato JSON[2] contenente metadati referenzianti informazioni atte a caratterizzare la collezione stessa.

Perché il formato RASH? RASH è un formato Web-First per la scrittura di articoli scientifici basati su HTML. Prevede un numero ristretto di elementi e l'impiego di ciascuno di essi viene regolamentato mediante una grammatica formale volta a rimuoverne le ambiguità e a conferirgli una specifica semantica. Di conseguenza l'accessibilità e la manipolazione del contenuto di documenti in tale formato risulta essere estremamente agevolata.

Per adempiere a tale necessità è stata realizzata tei2rash, un'applicazione WEB che permette di convertire documenti in formato TEI nel formato RASH e di impacchettarli in maniera tale che possano essere caricati su DocuDipity.

[1] API è l'acronimo di Application Programming Interface, si tratta di un intermediario Software che permette la comunicazione tra applicazioni diverse.
[2] JSON è l'acronimo di Javascript Object Notation, è un formato utilizzato per l'interscambio di dati tra applicazioni.

In particolare, l'applicazione permette all'utente di creare dei workspace: directories accompagnate da un file Manifest in formato JSON, popolabili con collezioni di documenti, ognuno ospitato in una apposita sottodirectory, previa conversione in formato RASH. In questo modo, le collezioni ottenute rispettano la struttura e i formati richiesti dall'API di caricamento di DocuDipity.

Di seguito è riportata l'organizzazione complessiva della tesi:

- Il secondo capitolo evidenzia il contesto tecnologico in cui si inserisce l'applicazione web tei2rash realizzata.

  In primo luogo, viene descritto il formato TEI seguito da una breve digressione sul catalogo Bibit della biblioteca digitale Biblioteca Italiana. Successivamente, viene descritto il formato RASH ed alternativi allo stesso. Viene poi fornita una panoramica dello strumento DocuDipity ed infine ho ritenuto necessario menzionare, per una maggiore completezza informativa, l'editor WYSIWYG[3] RAJE[4] per la scrittura di documenti RASH.

- Il terzo capitolo si apre con la spiegazione del problema da affrontare, nonché obbiettivo sotteso dall'applicazione stessa riassumibile in tre macro-problemi: conversione documento dal formato TEI al formato HTML, conversione del documento dal formato HTML al formato RASH e per ultimo l'impacchettamento seguito dall'aggiornamento del file JSON Manifest.

  Successivamente, viene fornita una panoramica sull'architettura logica dell'applicazione realizzata ed infine si descrivono le soluzioni adottate per sopperire ai 3 marco-problemi sopra citati.

- Il quarto capitolo descrive i dettagli implementativi dell'applicazione, tale descrizione seguirà una pipeline logica (caricamento/ottenimento TEI,

[3] WYSIWYG è l'acronimo di What You See Is What You Get.

[4] RAJE è l'acronimo Rash Javascript Editor, si tratta di un editor in grado di produrre documenti RASH ben formattati e validi.

conversione XHTML[5], conversione RASH, impacchettamento e aggiornamento Manifest) così da garantire una maggiore comprensione dell'infrastruttura tecnologica implementata.

- Il quinto capitolo andrà a trattare i test e le valutazioni effettuate sull'applicazione realizzata. Inoltre, delinea i limiti dell'applicazione oltre che a fornire tutta una serie di sviluppi futuri.

[5] XHTML è l'acronimo di eXtensible HyperText Markup Language, è un linguaggio di marcatura di ipertesti estendibile.

Capitolo 2

2 Contesto

2.1 TEI

TEI [1] (Text Encoding Initiative) è un sistema di codifica SGML/XML per testi, nato e sviluppato con l'intento di dare la possibilità di trascrivere in forma elettronica, un documento che originariamente è disponibile solo in forma cartacea.

In particolare, si tratta di un DTD [4] o modello di codifica che vuole contemplare tutta la serie di fenomeni di interesse 'umanistico' e trovare, per ciascuno di essi, un vocabolario unico al fine di arrivare ad una formalizzazione utile a normalizzare i criteri, le modalità e il lessico del markup, di fronte alla polisemia del linguaggio naturale.

A partire dal 1987 le tre maggiori associazioni mondiali di studiosi di scienze umane attraverso metodologie informatiche, la Association for Computer and Humanities (AHC), la Association for Computational Linguisitic (ACL) e la Association for Literary and Linguistic Computing (ALLC) hanno avviato un progetto internazionela per sviluppare un modello di codifica normalizzato, tale progetto è stato denominato Text Encoding Initiative[6] (TEI).

Il lavoro delle commissioni, iniziato nel 1989, ha condotto alla realizzazione di una vasta e complessa DTD (Document Type Definition), le cui caratteristiche sono state presentate per la prima volta nel 1994, con il titolo Guidelines for Electronic Text Encoding an Interchange (TEI P3). Nel 2007 è stata poi rilasciata la TEI P5, che ha portato ad una sostanziale revisione delle Guidelines, compresa la realizzazione di una versione di TEI basato su XML Schema che si affianca alla DTD.

Le Guidelines[7] TEI [1] forniscono uno strumento per rendere esplicite determinate caratteristiche di un testo in modo tale da facilitarne l'elaborazione mediante programmi informatici basati su tecnologie web. Definiamo questo processo di esplicitazione

[6] Sito ufficiale TEI: http://www.tei-c.org/
[7] http://www.tei-c.org/Guidelines

marcatura o codifica, qualsiasi rappresentazione di un testo digitale usa una qualche forma di codifica; TEI è stato creato sia per ovviare all'eccessiva proliferazione di schemi di codifica mutualmente incompatibili ostacolanti la ricerca scientifica, sia per adattarsi al crescente numero di applicazioni scientifiche ormai individuate per i testi in formato elettronico.

Ad oggi TEI [5] è uno dei progetti più influenti nel campo delle Digital Humanities[8]. È considerato lo standard de facto per la codifica dei testi, in quanto si ripone di contemplare tutti i fenomeni di interesse umanistico, fornendo per ciascuno un vocabolario unico di elementi non ambigui che sia quanto più possibile completo ed esaustivo.

L'utilizzo dello standard TEI facilità la portabilità dei documenti digitali favorendo la condivisione e l'interoperabilità ai fini della ricerca scientifica.
Per questi motivi si candida come principale formato d'origine per i documenti elaborati dall'applicazione TEI2HTML realizzata.

[8] La Digital Humanities o informatica umanistica è un campo di studi che nasce dall'interrogazione di procedure computazionali nelle discipline umanistiche.

Figura 1: Struttura interna documento TEI catalogo Bibit

In Figura 1 viene mostrata la struttura interna di un documento in formato TEI. Ogni testo TEI ha una testata che offre informazioni analoghe a quelle fornite dal frontespizio di un testo a stampa. La testata è introdotta e delimitata dall'elemento teiHeader ed è composta da quattro parti principali:

- L'elemento teiDesc contiene una descrizione bibliografica completa di un file digitale,
- L'elemento encodingDesc documenta le relazioni tra un documento elettronico e la fonte, o le fonti, da cui è derivato,
- L'elemento profileDesc contiene una descrizione dettagliata degli aspetti non bibliografici di un testo, quali le lingue o i dialetti usati, le circostanze in cui è stato prodotto e i partecipanti.

- L'elemento revisionDesc riassume la storia di revisioni di un documento elettronico.

L'elemento teiHeader è seguito dall'elemento text, quest'ultimo contiene un testo unico di qualsiasi tipo, unitario o composto, ad esempio una poesia o un dramma, una raccolta di saggi, un romanzo, un dizionario o un campione di corpus. L'elemento text si divide, a sua volta, in quattro elementi: body, front, group e back; il primo è obbligatorio mentre gli altri sono opzionali. All'interno dell'elemento body ci saranno una serie di suddivisioni ulteriori, necessarie a scandire la struttura interna del testo vero e proprio.

2.2 Biblioteca Italiana, il catalogo Bibit

Biblioteca italiana [6] è una biblioteca digitale di testi rappresentativi della tradizione culturale e letteraria dal Medioevo al Novecento, che conta nel proprio catalogo più di 3500 titoli.

Promossa sin dal 1996 ad opera del Centro Interuniversitario Biblioteca italiana Telematica (CIBIT), ad oggi ha conquistato un'ampia diffusione presso la comunità di studiosi, degli studenti e degli appassionati della letteratura italiana, attestandosi come biblioteca digitale più importante per dimensione e affidabilità per i documenti archiviati.

Bibit è la sezione centrale della biblioteca digitale, presenta oltre 1600 opere in formato testo, in edizione integrale, fondate su edizioni scientifiche di riferimento, codificate in XML/TEI e tutte liberamente accessibili ed interrogabili.

Bibit diviene la sorgente di documenti XML/TEI utilizzati per il testaggio dell'applicazione. Sicuramente uno sviluppo futuro di TEI2HTML prevede l'integrazione delle API di Bibit, così da permettere all'utente, qualora interessato, di selezionare direttamente l'opera letteraria senza dover disporre dell'URL[9] o della risorsa TEI in locale.

[9] URL è l'acronimo di Uniform Resource Locator, è un identificatore univoco di risorsa sul web.

2.3 Formati HTML per Articoli Accademici

2.3.1 Scholarly HTML

Scholarly HTML[10] [3] [7] definisce una serie di regole descrittive per l'adozione di un sottoinsieme definito di HTML per descrivere i metadati e il contenuto di articoli accademici.

Non si presenta come una grammatica formale, ma come un insieme di regole descrittive che consente di specificare solo una quantità ridotta di tag HTML per descrivere i metadati e il contenuto di un articolo accademico. È il principale formato intermedio utilizzato in ContentMine per descrivere la conversione del contenuto da PDF in HTML.

Inoltre, esiste un altro progetto che porta lo stesso nome, appunto Scholarly HTML, realizzato dalla società science.ai[11] con l'intento di fornire un formato dati specifico del dominio ma basato su standard aperti, al fine di consentire lo scambio interoperabile di articoli accademici.

Il formato non possiede alcuna grammatica formale ma gode di una documentazione ben definita che insegna come utilizzare il set di tag HTML accompagnato da uno schema.org[12] per realizzare documenti in tale formato.

2.3.2 HTMLBook

HTMLBook[13] [3] [8] è una specifica di O'Reilly per la creazione di documenti HTML utilizzando un sottoinsieme di elementi (X)HTML5, in particolare, si tratta di uno standard aperto basato su XHTML5 per l'authoring e la produzione di libri cartacei e digitali.

[10] http://scholarlyhtml.org/
[11] https://github.com/scienceai/scholarly.vernacular.io
[12] https://schema.org/
[13] https://github.com/oreillymedia/HTMLBook/

HTMLBook non contiene elementi o attributi aggiuntivi al di fuori della specifica XHTML5, è adattato semanticamente alla struttura di un libro, inclusi i contenuti più complessi utilizzati nei documenti tecnici e di riferimento.

HTMLBook è definito e può essere convalidato rispetto a uno schema XML, inoltre, i suoi fogli di stile sono scritti in CSS.

2.3.3 PubCSS

PubCSS [3] [9] è una libreria di fogli stile CSS e modelli HTML per la formattazione di pubblicazioni accademiche per la stampa e il web. PubCSS è un progetto che mira a promuovere l'uso di HTML + CSS per scrivere articoli accademici.

Non definisce una grammatica formale per l'elemento HTML impostato da utilizzare, piuttosto, fornisce alcuni modelli HTML in base a quattro stili CSS, che imitano i quattro stili LaTeX[14] per articoli informatici, ad esempio ACM SIG Proceedings, ACM SIGCHI Proceedings, ACM SIGCHI Extended Abstracts e IEE Conference Proceedings. Utilizzando il software Prince è possibile trasformare gli articoli PubCSS in PDF.

2.3.4 RASH

Rash [3] o Research Articles in Semplified HTML è un linguaggio di Markup che limita l'uso di elementi HTML[15] a soli 32 elementi per la scrittura di articoli di ricerca accademica. Consente agli autori di utilizzare annotazioni RDF incorporate, e segue rigorosamente il Digital Publishing WAI-ARIA Module 1.0[16] per esprimere la semantica strutturale sui vari elementi di Markup utilizzati.

Lo sviluppo RASH è partito dall'intera grammatica HTML5, e ha proceduto rimuovendo l'uso particolare degli elementi HTML, per renderli sufficientemente espressivi nell'ottica di rappresentare le strutture su cui poggiano gli articoli accademici

[14] https://www.latex-project.org/
[15] http://www.w3.org/TR/html5/
[16] https://www.w3.org/TR/dpub-aria-1.0/

e per avere un linguaggio totalmente conforme alla teoria sui modelli strutturali per Documenti XML[17].

L'idea alla base di tale teoria è che ogni elemento di un linguaggio di markup dovrebbe rispettare uno e un solo modello strutturale. I benefici che si traggono da questa restrizione sono una disambiguazione del linguaggio e l'ottenimento di una struttura ben definita che facilita fortemente il processo di accessibilità e di conversione tra linguaggi.

Rash è accompagnato da una grammatica[18] formale sviluppata mediante RelaxNG[19], che è un linguaggio schema semplice, facile da imparare ed estremamente potente per XML. Quest'ultima è stata organizzata in quattro blocchi logici di regole sintattiche, definendo rispettivamente elementi, attributi, modelli di contenuto per gli elementi e le relative liste di attributi.

Rash, a differenza degli altri formati, fornisce alternative pratiche alla gestione e all'implementazione delle formule matematiche. È noto che MathML sia il modo più accessibile per scrivere formule matematiche, d'altro canto, però, presenta una sintassi estremamente prolissa che lo ostacola per un'adozione diretta. Per agevolare il creatore a gestire le formule, Rash aggiunge due ulteriori modi. Il primo prevede l'impiego di un'immagine mediante l'elemento img, tale approccio risulta essere estremamente inaccessibile in quanto i vari elementi costituenti la formula non sono opportunamente contrassegnati per una distinzione corretta. L' alternativa prevede l'impiego di LaTeX che risulta essere il modo più comune per scrivere formule in articoli scientifici. Indipendentemente dalla modalità adottata, il rendering è implementato utilizzando MathJax[20], un motore di visualizzazione Javascript di formule matematiche funzionante nella maggior parte dei Browser moderni.

L'idea alla base di Rash è quella di consentire agli autori di continuare ad utilizzare gli elaboratori di testi e i formati con cui hanno una maggiore familiarità nel redigere i propri documenti, è innegabile che non tutti gli autori di articoli sono in grado di scrivere articoli in HTML. Affinché tale formato possa raggiungere un'ampia adozione nell'ambito autoriale è necessario supportarlo mediante interfacce e strumenti appropriati.

[17] https://asistdl.onlinelibrary.wiley.com/doi/10.1002/asi.23088
[18] https://raw.githubusercontent.com/essepuntato/rash/7ef4c2f2ea63575fb32f17e826d60333543eda67/grammar/rash.rng
[19] https://relaxng.org/spec-20011203.html
[20] https://www.mathjax.org/

A tal proposito è stato realizzato il Rash Framework[21] [10]: un insieme di specifiche e strumenti di struttura/conversione/estrazione per redigere articoli in Rash. Ad oggi il framework permette di convertire documenti nel formato Rash a partire dal formato ODT e DOCX, questa tesi propone un'applicazione web in grado di convertire documenti in formato Rash a partire da un formato d'origine non ancora trattato all'interno del Framework, il TEI.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml" prefix="schema: http://schema.org/ prism: http://prismstandard.org/namespaces/basic/2.0/ dcterms: http://purl.org/dc/terms/">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title property="dcterms:title">Rime</title>
    <meta name="author" content="Marco Ballardini" />
  </head>
  <body>

    <section class="body_div1">

      <h1>Rime</h1>

      <blockquote class="body_div1_divHead_opener">

        <p>DANTE ALIGHIERI A DANTE DA MAIANO</p>
        </blockquote>

      <blockquote class="body_div1_divBody_div2_lgouter">

        <p>Savete giudicar vostra ragione,</p>

        <p>o om che pregio di saver portate</p>

        <p>per che, vitando aver con voi quistione</p>

        <p>com so rispondo a le parole ornate.</p>
        </blockquote>
      </section>
  </body>
</html>
```

Figura 2: Struttura interna documento rash risultato della trasformazione del sorgente TEI mostrato in Figura 1

2.4 Editor WYSIWYG per Articoli RASH

L'acronimo WYSIWYG significa letteralmente What You See Is What You Get che può essere tradotto in "quello che vedi sullo schermo è quello che ottieni". Gli editor di questo tipo sono tipicamente caratterizzati da un'interfaccia visuale, l'utente può creare documenti nel linguaggio per cui l'editor è pensato senza esporsi alla complessità dello stesso.

[21] https://github.com/essepuntato/rash

2.4.1 RAJE

RAJE [11] o Rash Javascript Editor è un elaboratore di testi WYSIWYG per la scrittura di articoli accademici in HTML mediante il formato RASH. In particolare, RAJE permette agli autori di scrivere documenti di ricerca in HTML in modo nativo per mezzo di un'interfaccia user-friendly, invece, che scrivere un markup grezzo con un IDE, editor di testo o elaboratori esterni.

RAJE è stato sviluppato per consentire l'impiego di un formato alternativo, l'HTML, al PDF, che presenta importanti svantaggi come la mancanza di interattività, il fatto che la sua struttura monolitica non è adatta alla condivisione di contenuti sul web e le sue note difficoltà relative all'accessibilità dei contenuti fa da parte di persone con disabilità.

Per concludere, RAJE garantisce ai propri utenti i vantaggi di un word processor uniti a quelli dati da un formato basato su HTML, ovvero, interattività, accessibilità e facilità di elaborazione da parte della macchina.

2.5 DocuDipity

Esistono vari modi per leggere articoli scientifici e, in generale, i documenti. La lettura sequenziale del documento può essere integrata con visualizzazioni alternative come viste ad albero, grafici o mappe.

DocuDipity [12] [13] è un'applicazione web in grado di integrare agevolmente tecniche di infoview in un ambiente coordinato che aiuta i ricercatori a leggere e ad analizzare articoli scientifici in maniera interattiva e dinamica.

È in grado di estrarre e manipolare informazioni relative alla struttura e al contenuto del set di documenti forniti in input, l'interfaccia è composta da quattro parti principali, come mostrato nella Figura 3. La barra di navigazione in alto permette di selezionare il documento d'interesse, del quale viene mostrato il contenuto nelle due forme diverse ma coordinate poste nell'area centrale. In particolare, a sinistra viene fornita una vista basata sulla tecnica SunBurst che permette di catturare un riepilogo sulla struttura del documento, a destra viene fornita una visualizzazione ipertestuale del

contenuto del documento selezionato, mentre, in basso l'utente può selezionare regole di colorazione predefinite per evidenziare caratteristiche rilevanti del documento.

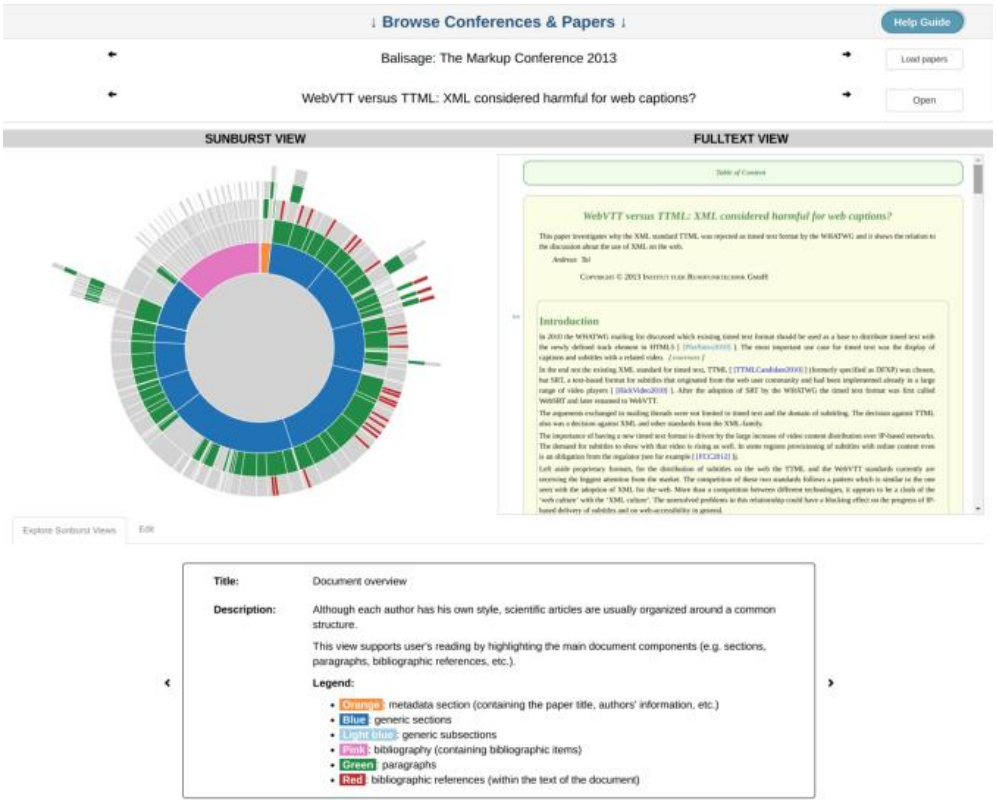

Figura 3: Overview dell'interfaccia DocuDipity

Affinché tali collezioni possano esservi "iniettate", DocuDipity espone un API di caricamento per collezioni di documenti. È necessario, però, che siano impacchettate secondo una struttura e un formato ben preciso. L'applicazione web TEI2RASH, oggetto di questa tesi, propone una soluzione puntuale a tale necessità; mediante una serie di conversioni ed elaborazioni è in grado di impacchettare i documenti d'interesse nella forma richiesta dall'API di caricamento sulla piattaforma.

Capitolo 3

3 DAL PROBLEMA ALLA SOLUZIONE

La piattaforma DocuDipity espone un'API di caricamento per collezioni di documenti, per potervi interagire è necessario rispettare alcuni requisiti di struttura e formato, come mostrato nella Figura 4.

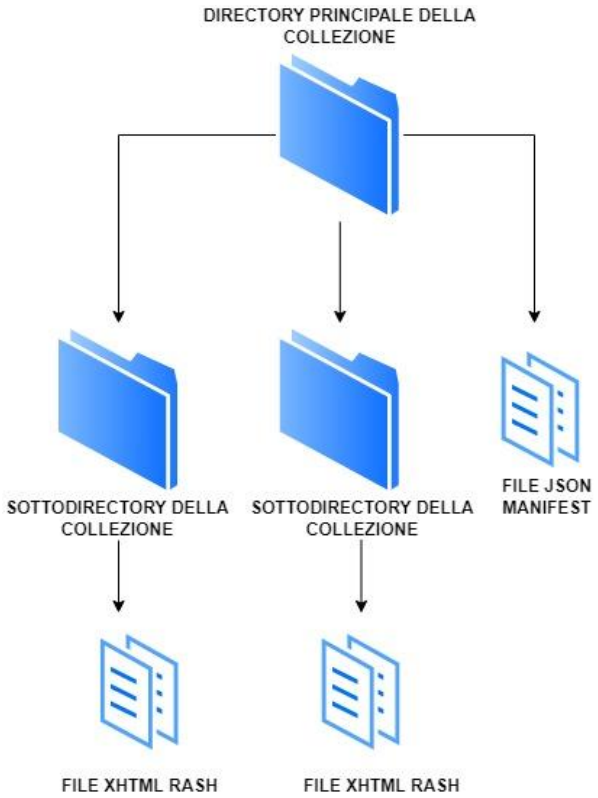

Figura 4: Diagramma API DocuDipity caricamento collezione

La Figura 4 mostra un diagramma che riassume i requisiti richiesti dall'API di DocuDipity per il caricamento di collezioni di documenti. La collezione deve essere contenuta all'interno di una directory principale, ogni documento deve essere in formato XHTML RASH e ospitato in una sottodirectory, infine, la collezione deve essere

accompagnata da un file Manifest in formato JSON contente metadati esponenti informazioni sulla collezione stessa.

Per ottenere una struttura che rispetti tali requisiti è stata realizzata l'applicazione web tei2rash, quest'ultima verte su due principali aspetti: la gestione dei workspace e la conversione del documento dal formato TEI al formato XHTML RASH.

Con il termine workspace mi riferisco ad una collezione di documenti idonea ai requisiti dell'API di DocuDipity.

3.1 Panoramica dell'applicazione tei2rash

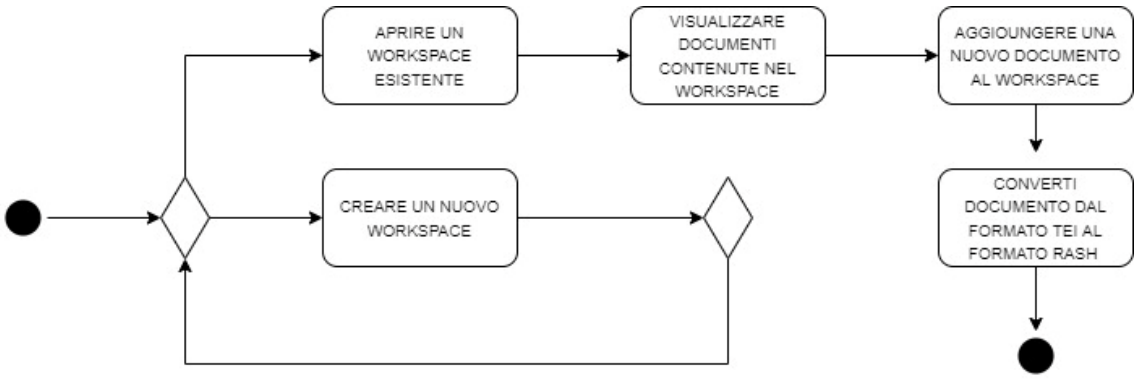

Figura 5: Pipeline logica funzionamento applicazione TEI2RASH

La Figura 5 mostra un digramma che definisce la pipeline logica d'interazione fra l'utente utilizzatore e l'applicazione tei2rash, di seguito andrò a dettagliare ciascuna attività definita.

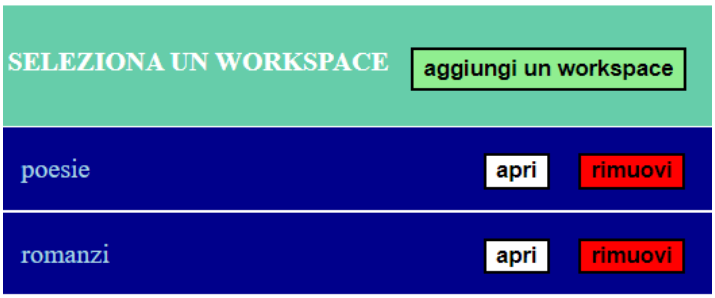

Figura 6: Pannello gestione workspace applicazione TEI2RASH

L’applicazione web presenta un layout estremamente semplice quanto intuitivo, in primo luogo, permette all’utente di aggiungere un nuovo workspace, di aprirne uno esistente, o, eventualmente cancellarlo. La cancellazione di un workspace esistente viene compiuta con successo solamente se non contiene alcuna opera al suo interno.

Figura 7: Pannello gestione opere workspace applicazione TEI2RASH

A seguito dell’apertura di un workspace, l’applicazione fornisce un listato delle opere contenute al suo interno, ognuna accompagnata dalle informazioni identificative più rilevanti: titolo, anno e autore.

L’utente ha la possibilità di aggiungere una nuova opera al workspace, o, di rimuoverne una già esistente. Il pannello presenta anche un bottone “CARICA SU DOCUDIPITY” la cui funzionalità non è ancora stata realizzata, risulta essere un ottimo punto di partenza per implementazioni future volte a migliorare l’applicazione.

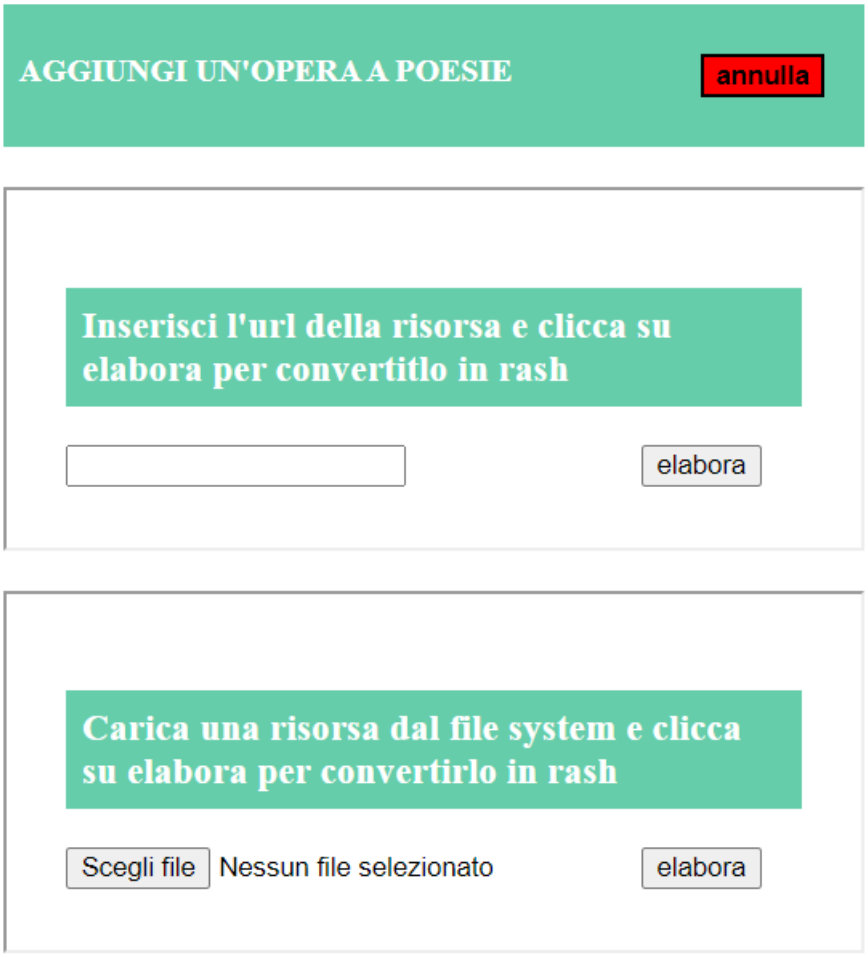

Figura 8: Pannello aggiungi opera applicazione TEI2RASH

Mediante questo pannello, l'utente può aggiungere un nuovo documento al workspace, o, inserendo l'URL della risorsa TEI o caricando un file TEI direttamente dal file system. Indipendentemente dalla modalità adottata, l'applicazione compie una prima conversione dal formato TEI al formato XHTML e successivamente una conversione dal formato XHTML al formato RASH. Se il processo complessivo di conversione avviene con successo, l'applicazione procede a validare il documento secondo la grammatica RASH e, qualora valido, aggiorna il file Manifest, impacchetta il documento, e fornisce un feedback a video come mostrato nella Figura 9.

Figura 9: Esempio feedback di successo applicazione TEI2RASH

Successivamente, l'opera sarà presente nel listato proprio del workspace in questione.

3.2 Le 3 macro-fasi del funzionamento

In Figura 10 viene illustrato un diagramma in cui viene sintetizzata la pipeline logica sequenziale delle 3 macro-fasi del funzionamento.

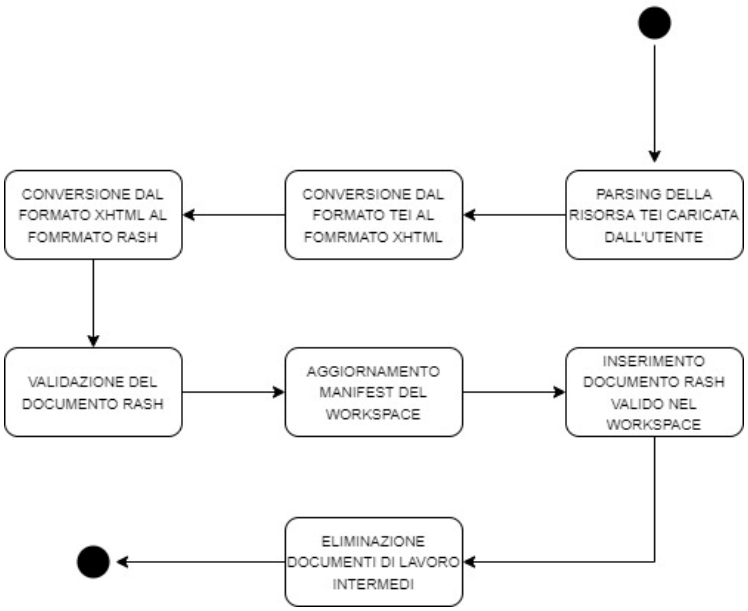

Figura 10: Pipeline logica di funzionamento delle 3 macro-fasi

Affinché il lettore possa comprendere meglio quanto andrò a descrivere nelle sezioni successive, fornisco una breve introduzione al linguaggio XSLT accompagnandola con un esempio pratico.

3.2.1 XSLT

XSLT o EXtensible Stylesheet Language Transformations [14] [15] è un linguaggio di trasformazione: dato un documento XML è possibile generare un altro documento XML derivato applicando delle regole di trasformazione specificate nel foglio di stile.

Un foglio di stile XSLT è un documento XML che utilizza un DTD i cui elementi hanno senso noto al motore XSLT. Un foglio di stile XSLT è composto sostanzialmente di template di costruzione, che permettono di riscrivere una selezione di elementi del documento XML d'origine in altri elementi del documento di destinazione, inoltre è possibile definire strutture di controllo, variabili e funzioni.

Ogni template individua un pattern da ricercare nel documento di partenza, e vi associa un blocco di elementi e testo da inserire nel documento di destinazione. XSLT si basa fondamentalmente su xPath per la definizione dei pattern.

Esistono due filosofie di riscrittura disponibili in XSLT, che vengono dette pull e push:

- Pull: basata su template, viene usata tipicamente per trasformare dati. In un documento preformattato per l'output, si vanno ad inserire le parti di documento tratte dal file XML d'origine.
- Push: basata su regole, usata tipicamente per trasformare documenti. Per ogni elemento del documento di input, si cerca la regola più appropriata e la si usa per scrivere il risultato.

```
<xsl:template match = "section">
    </hr> <h2><xsl:value-of select="title"/></h2>
    <xsl:apply-templates />
</xsl:template>
```

Figura 11: Esempio sintassi foglio di stile XSLT di tipo push

Per fornire maggiore chiarezza su quanto detto sino ad ora, andrò ad analizzare l'estratto di codice XSLT fornito nella Figura 11. L'elemento xsl: template è la regola da applicare se l'elemento del file XML d'origine corrisponde al valore dell'attributo match

definito mediante un'istruzione xPath[22]; xsl: apply-templates spinge a cercare, tra i figli dell'elemento selezionato dall'attributo match del template padre, se esistono altri template applicabili. È il modo per far ripartire ricorsivamente la ricerca di altri template; xsl: value-of preleva il contenuto dell'elemento XML selezionato dall'attributo select.

In particolare, il template definito nella Figura 11 viene applicato agli elementi section presenti nel documento XML d'origine, oggetto della trasformazione. Introduce un elemento html br seguito da un elemento h2, all'interno del quale andrà a porre il contenuto dell'elemento identificato dalla select dell'elemento xsl: value-of. Successivamente andrà a richiamare ricorsivamente tutti i template presenti nel foglio di stile sugli elementi figli dell'oggetto section del documento XML d'origine.

3.2.2 Conversione dei documenti dal formato TEI al formato XHTML

Per il primo processo di conversione ho utilizzato l'utility tei2html [16] disponibile online su GitHub[23]. Si tratta di una raccolta di fogli di stile xslt 3.0 per trasformare un documento codificato in base alla DTD TEI in HTML, sviluppati per generare un documento HTML monolitico, ma possono generare anche file ePub[24]. La trasformazione supporta i seguenti elementi[25] tipici del formato TEI:

- Testo normale e stili di testo,
- Frontespizi,
- Tabelle,
- Liste,
- Sommario,
- Poesie e opere teatrali,
- Illustrazioni e immagini.

[22] https://www.w3schools.com/xml/xpath_intro.asp
[23] GitHub è un servizio di hosting per progetti Software.
[24] ePub è uno standard aperto specifico per la pubblicazione di libri digitali nel formato XML.
[25]https://www.wwp.northeastern.edu/outreach/seminars/_current/handouts/elementList.xhtml#gi_quote

3.2.2.1 Panoramica tei2html

Il progetto presenta un foglio di stile principale, punto d'ingresso per la conversione da TEI a HTML, tei2html.xsl. Al suo interno troviamo molteplici inclusioni di dipendenze ad altri fogli di stile xslt ognuno adempiente ad una specifica funzionalità. Inoltre, vengono definite alcune variabili globali assegnate mediante istruzioni xPath; quest'ultime costituiscono i valori degli attributi dei metadati propri del documento HTML di output.

3.2.2.2 Le inclusioni

Le dipendenze incluse all'interno del file principale tei2html.xsl sono ospitate nella directory "modules", quanto segue è una breve descrizione delle funzioni realizzate da alcune delle dipendenze.

Il foglio di stile figures.xsl gestisce l'elemento figure del documento TEI, utilizzato per contenere immagini, didascalie e descrizioni testuali delle stesse. Tale elemento viene tradotto nell'elemento HTML img nel documento di output, specificandone i valori degli attributi, name, alt, src, width, height e class.

Il foglio di stile tables.xsl gestisce l'elemento table del documento TEI, utilizzato per contenere testo visualizzato in forma tabellare, in righe e colonne; quest'ultimo ospita al suo interno gli elementi figli head, row e cell. Table viene tradotto nell'elemento HTML table; head viene tradotto nell'elemento HTML caption; row viene tradotto nell'elemento HTML tr, mentre cell viene tradotto nell'elemento HTML td.

Il foglio di stile lists.xsl gestisce l'elemento list del documento TEI, utilizzato per contenere una qualsiasi sequenza di voci organizzate in una lista; quest'ultimo ospita al suo interno l'elemento figlio item. List, a seconda del valore dell'attributo rend, viene tradotto nell'elemento HTML ol o ul; item viene tradotto nell'elemento li e il suo contenuto testuale viene wrappato nell'elemento HTML inline span.

Il foglio di stile divisions.xsl gestisce gli elementi divisione del documento TEI, utilizzati per definire delle partizioni testuali. Quest'ultimi si suddividono in numerati e

non numerati, le divisioni numerate sono denominate div1, div2, ecc., dove il numero indica il livello di profondità della divisione all'interno della gerarchia, mentre, le divisioni non numerate sono semplicemente chiamate div e possono essere nidificate l'una dentro l'altra. Indipendentemente dalla tipologia di divisione, queste vengono tradotte nell'elemento HTML div, il nome della divisione e il valore dell'attributo type divengono il valore dell'attributo class dell'elemento generato.

Il foglio di stile header.xsl produce lo scheletro del documento HTML di output, definisce l'elemento html che ingloba gli elementi header di cui specifica gli elementi title, meta e style; e l'elemento body da cui richiama ricorsivamente tutti i template sugli elementi figli dell'elemento text del documento TEI.

3.2.2.3 L'integrazione in tei2rash

Mediante il comando Git[26] checkout[27] ho scaricato il repository online. Per poter eseguire la trasformazione è necessario un processore XSLT 3.0 sul server invocato come processo separato. Come processore XSLT ho adottato Saxon[28], in particolare ho scaricato la versione SaxonHE11-3J[29]. SaxonJ[30] è processore XSLT scritto e basato su Java[31], di conseguenza, è necessario avere una Java Runtime Environment[32] installato nel sistema.

Il comando[33] per compiere la trasformazione è il seguente [java -cp"" net.sf.saxon.Tranform -s"" -xsl"" -o""]; il parametro -cp deve essere seguito dal path in cui il jar del Software Saxon risiede, il parametro -s deve essere seguito dal path del file sorgente xml su cui vogliamo compere la trasformazione, il parametro -xsl deve essere seguito dal path del foglio di stile xslt principale, mentre il parametro -o deve essere seguito dal path in cui vogliamo salvare l'output risultato della trasformazione.

---

[26] Git è un software per il controllo di versione distribuito, utilizzato da interfaccia a riga di comando.
[27] https://www.atlassian.com/git/tutorials/using-branches/git-checkout#:~:text=The%20git%20checkout%20command%20lets,new%20commits%20on%20that%20branch.
[28] https://www.saxonica.com/welcome/welcome.xml
[29] https://github.com/Saxonica/Saxon-HE/
[30] https://www.saxonica.com/html/documentation12/about/gettingstarted/gettingstartedjava.html
[31] Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica.
[32] JRE o Java Runtime Environment è un ambiente di esecuzione per applicazioni scritte nel linguaggio Java.
[33] https://www.saxonica.com/html/documentation10/using-xsl/commandline/

Essendo TEI2RASH un'applicazione Python[34] tale comando viene richiamato dal server come processo separato mediante l'impiego del modulo Subprocess [17], del quale viene richiamata la funzione call.

L'unica problematica riscontrata è relativa al formato del documento di output risultato della trasformazione. Essendo che si tratta di una trasformazione intermedia, il documento di output dovrà essere sottoposto da un'ulteriore trasformazione xslt. Di conseguenza è necessario modificare il formato di output da HTML a XHTML così che possa essere passato come valore di input del parametro -o. Per modificare il formato di output ho cambiato il valore dell'elemento xsl: output da html a xhtml.

3.2.2.4 Convertitori alternativi a tei2html

3.2.2.4.1 Teic

Teic [18] è una famiglia di fogli di stile XSLT 2.0 per trasformare documenti TEI XML in vari formati, tra cui XHTML, LaTeX, JSON, DOCS e ODT. Presenta limitazioni, in quanto, non è in grado di parsare ogni elemento TEI e i corrispettivi attributi, e non fornisce alcun listato degli elementi che è in grado di trasformare. Inoltre, il pacchetto richiede l'installazione dello strumento aggiuntivo Apache Ant[35] 1.9+ e JAVA 1.6+, oltre, che ad un qualsiasi processore XSLT, se Saxon dalla versione 9.0 e successive.

Il Repository è scandito in molteplici directories, ognuna dedicata ad una specifica conversione dal formato TEI. Al loro interno contengono fogli di stile xslt e un file xml build di Ant. All'interno del file build xml è definito, tra i tag target, il comando necessario a compiere la trasformazione xslt. Per eseguirlo da linea di comando è necessario spostarsi dentro la directory dedicata alla trasformazione xslt di nostro interesse, in questo caso html, ed eseguire [Ant teitohtml fileinput fileoutput]; teitohtml è il nome del comando, definito tra i tag target, interno al file build xml, fileinput e fileoutput

[34] Python è un linguaggio di programmazione di alto livello, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.
[35] https://ant.apache.org/

costituiscono rispettivamente il path del file TEI d'origine e il path del file HTML in cui si riflette la trasformazione xslt.

3.2.2.4.2 Tei Boilerplate

Tei Boilerplate [19] fornisce fogli di stile xslt per convertire documenti dal formato TEI al formato HTML5. Nella directory css fornisce fogli di stile CSS predefiniti per i documenti output della trasformazione, in alternativa, l'utente stesso può aggiungerne.

La documentazione evidenzia che l'utility è in grado di parsare solamente un numero molto ridotto di elementi e attributi TEI, in particolare detiene una collezione predefinita di temi specificanti la struttura generale del documento sorgente che l'utente può trasformare con successo, e specifica che è in grado di interpretare i soli attributi rend, rendition e style.

Per compiere la trasformazione è necessario solamente un processore XSLT, non è richiesta l'installazione di pacchetti addizionali nel sistema. Tei Boilerplate è pensato per essere un'alternativa leggera e semplice alle soluzioni XSLT più complesse.

3.2.3 Conversione dei documenti dal formato XHTML al formato RASH

Il primo processo di conversione ci fornisce un documento xhtml in output, diviene ora necessario convertirlo in rash. Essendo sia xhtml sia rash formati basati su xml, anche il secondo processo di trasformazione avviene mediante l'applicazione di un foglio di stile xslt.

3.2.3.1 Sviluppo

In questo caso, era mio compito realizzare il foglio di stile xslt incaricato della conversione dal formato xhtml a rash.

Inizialmente, mi sono soffermato sull'analisi della grammatica rash così da comprendere meglio i vincoli sintattici e strutturali da riflettere nel documento di output, risultato della trasformazione finale.

Successivamente, ho analizzato l'organizzazione interna degli elementi dei documenti xhtml, output della prima conversione, al fine di individuare pattern[36] strutturali ricorrenti comuni.

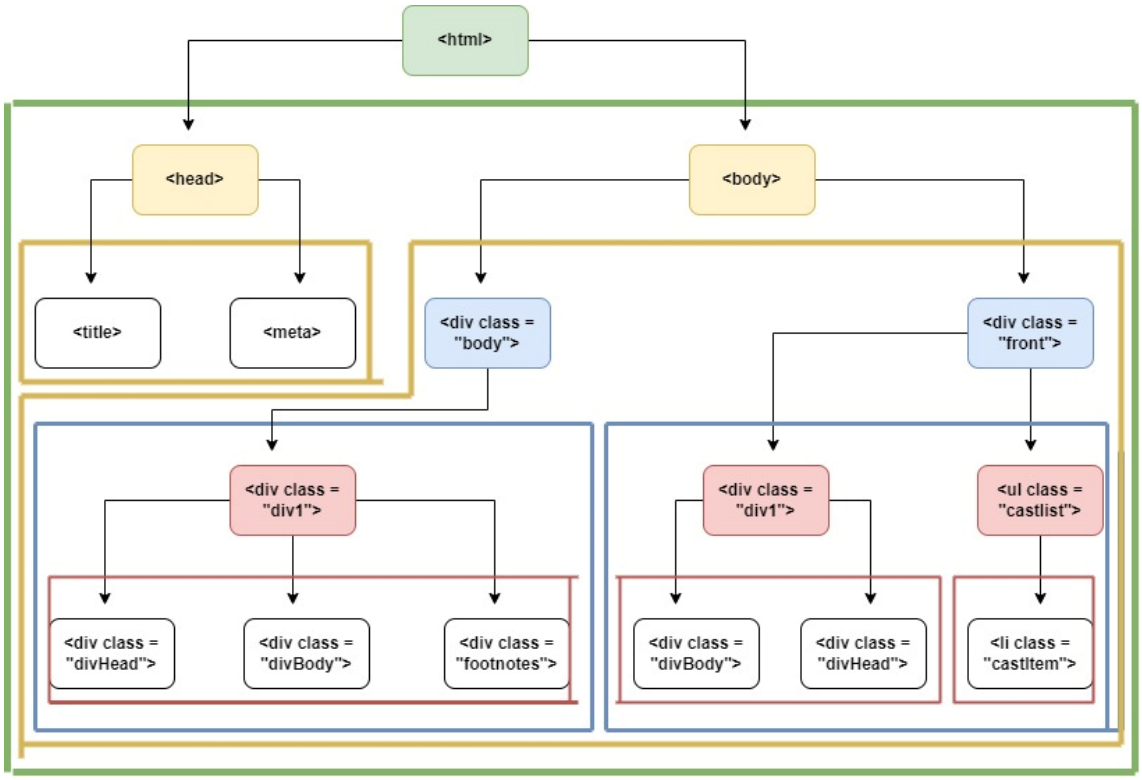

Figura 12: Diagramma semplificato struttura gerarchica interna documenti xhtml output prima trasformazione xslt

[36] Con pattern faccio riferimento ad una collezione di elementi identificata da un elemento "primario" la cui organizzazione interna è tale da produrre, nel documento di output, una struttura equivalente in termini di contenuto rispettando i requisiti sintattici strutturali dettati dalla grammatica Rash.

In Figura 12 viene mostrato il diagramma semplificato rappresentante la struttura gerarchica complessiva, suddivisa in pattern, risultato dello studio dell'organizzazione interna di molteplici documenti di output derivanti dalla prima conversione. L'elemento html evidenziato in verde costituisce la radice del documento, l'espansione gerarchica viene evidenziata in molteplici livelli, per ogni livello vi è uno o più elementi primari, evidenziati con lo stesso colore del pattern che identificano.

L'elemento div di classe body contiene l'intero corpo di un testo unitario, mentre, l'elemento div di classe front contiene qualsiasi prefazione che si trova all'inizio di un documento, prima del corpo principale.

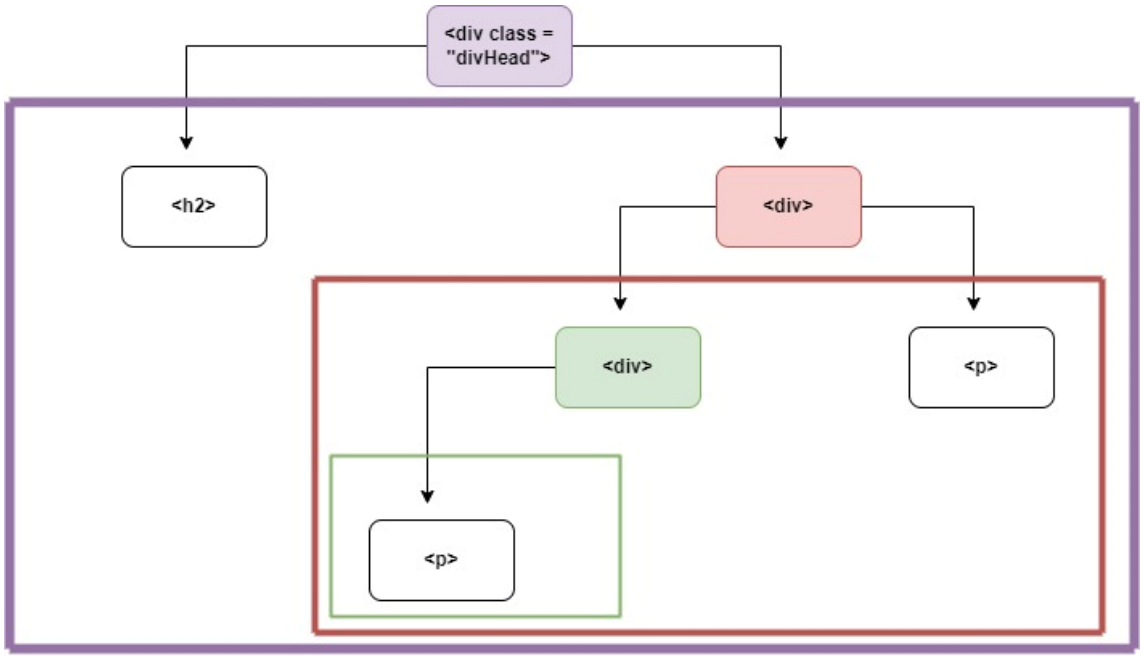

Figura 13: Diagramma struttura interna semplificata dell' espansione gerarchica elemento div di classe divHead

In Figura 13 viene mostrata la struttura interna, suddivisa in pattern, dell'espansione gerarchica dell'elemento div di classe divHead mostrato nella Figura 12. Tipicamente al suo interno troviamo un'intestazione e un'introduzione alla sezione che lo segue, il div con classe divBody.

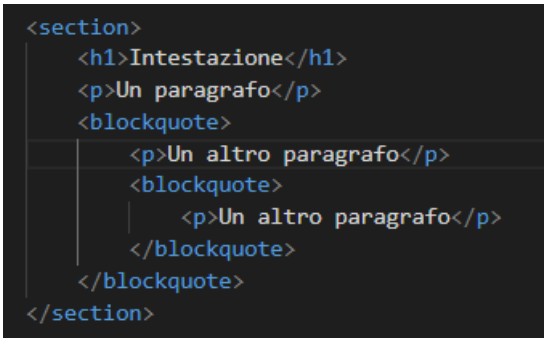

Figura 14: Risultato trasformazione xslt pattern strutturale mostrato in Figura 13

In Figura 14 viene mostrata la struttura interna rash valida risultato della trasformazione xslt del pattern in Figura 13. In particolare, il div di classe divHead viene omesso e l'elemento h2 viene tradotto in un elemento h1 d'intestazione della sezione complessiva racchiudente la struttura, quest'ultima deriva della traduzione dell'elemento div di classe div1 padre. Se il div di classe divHead non contiene alcuna intestazione, allora, ne viene aggiunta una fittizia.

Differente è la logica di trasformazione per il div evidenziato in rosso, in questo caso, se l'elemento è seguito da un elemento d'intestazione, allora, viene anch'esso trasformato in un elemento xhtml section, altrimenti, viene trasformato in un elemento blockquote. In definitiva, ogni template del foglio di stile andrà ad implementare delle regole di trasformazione differenti a seconda degli elementi figli e dalla posizione gerarchica coperta dell'elemento primario che è corrisposto dal valore match del template stesso, nell'ottica di ottenere una trasformazione sintatticamente e strutturalmente conforme con la grammatica rash.

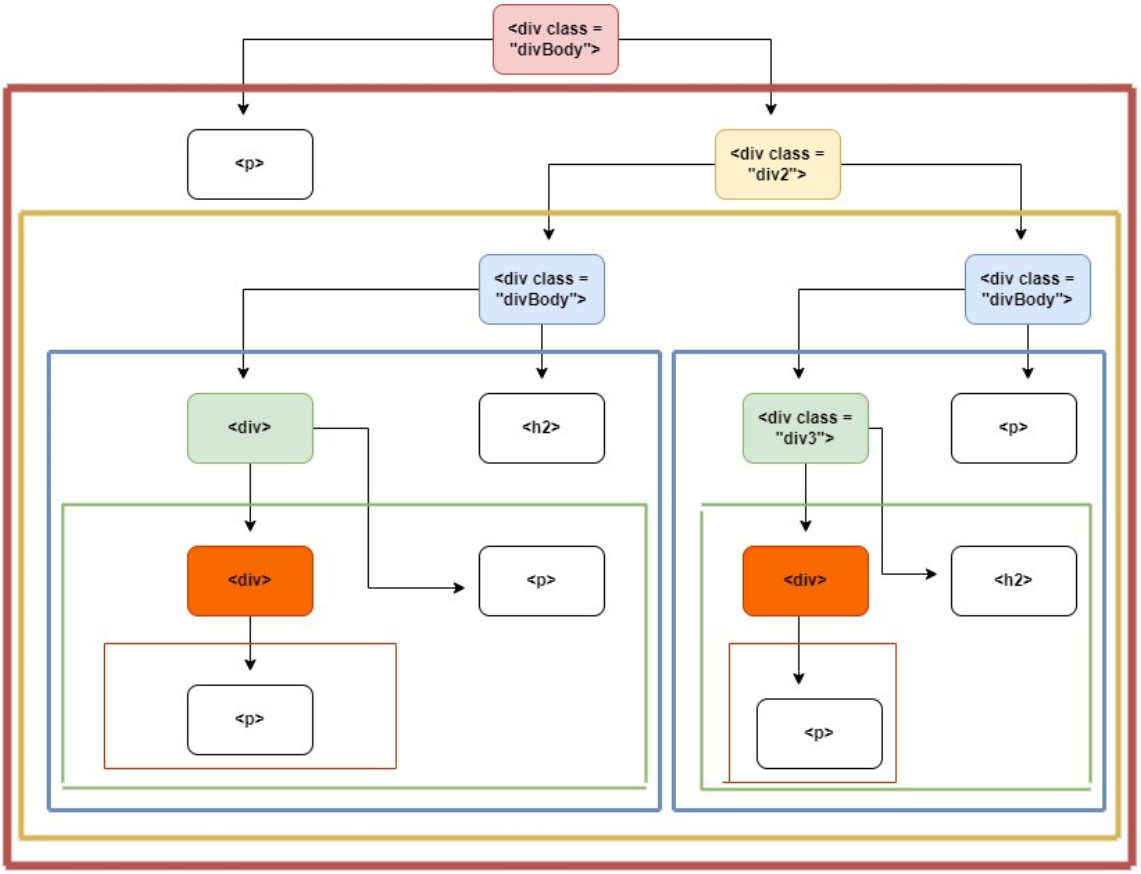

Figura 15: Diagramma struttura interna semplificata dell'espansione gerarchica elemento div di classe divBody

L'elemento div di classe divBody mostrato in Figura 15 tipicamente ospita il contenuto di un singolo capitolo, strofa o canto, in generale, il contenuto del singolo elemento incluso nel div di cui è figlio. Il suo contenuto viene introdotto dall'intestazione presente nel div che lo precede di classe divHead.

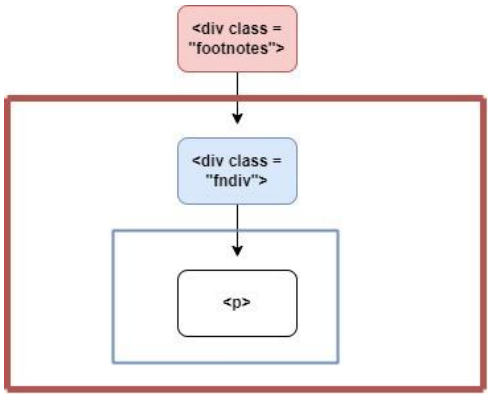

Figura 16: Diagramma struttura interna semplificata dell'espansione gerarchica elemento div di classe footnotes

L’elemento div di classe footnotes mostrato in Figura 16 contiene le note a piè di pagina, ogni nota è divisa dal div di classe fndiv e contiene paragrafi.

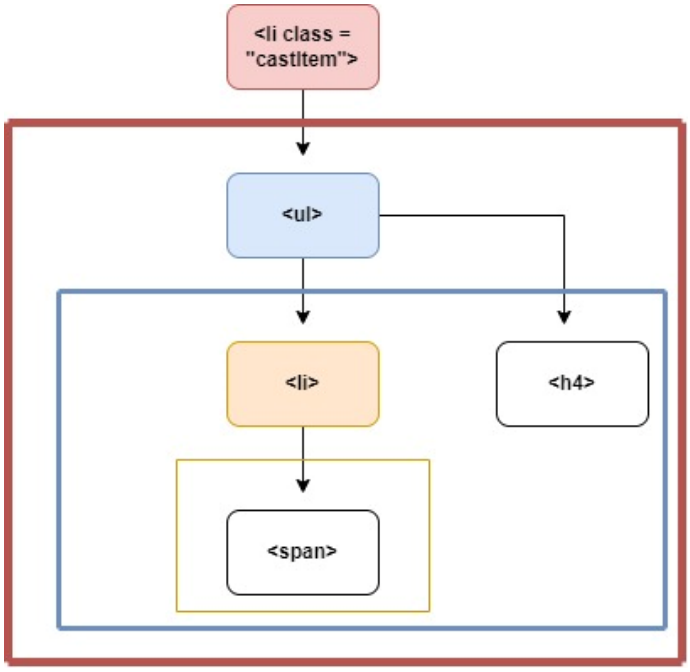

Figura 17: Diagramma struttura interna semplificata dell'espansione gerarchica elemento div di classe castItem

L’elemento li di classe castItem mostrato in Figura 17 contiene la lista dei personaggi presenti all’interno dell’opera, per ognuno vi è un’intestazione seguita da una breve descrizione.

Una tale strutturazione gerarchica interna mi ha permesso di scindere il documento in molteplici pattern intermedi e ricorrenti, ogni elemento appartenente a tali pattern sarà caratterizzato a seconda del ruolo gerarchico che ricopre e della classe di appartenenza, di particolare importanza sono i nodi “primari” evidenziati con lo stesso colore del pattern in cui risiedono.

In definitiva, ogni pattern ha una caratteristica comune dettata dall’elemento “primario” che lo identifica. Ogni elemento primario del singolo pattern può essere identificato da un’espressione xPath a seconda della posizione gerarchica, del tipo o dalla classe a cui appartiene. Questo mi ha permesso di implementare un foglio di stile xslt di tipo push, l’attributo match di ogni template definito selezionerà l’elemento primario identificatore del singolo pattern e lo tradurrà nel documento di output in maniera tale che

sia coerente con i vincoli sintattici e strutturali dettati dalla grammatica rash ed equivalente in termini di contenuto. Ogni template, oltre che a definire la struttura xhtml di output andrà a invocare al suo interno anche l'istruzione xsl: apply-template, così da richiamare, su ogni figlio dell'elemento corrisposto dal match, tutti i template definiti all'interno del foglio stile. In questo modo a partire dall'elemento radice del documento tutti gli elementi figli verranno parsati e opportunamente tradotti nel documento di output.

3.2.4 Impacchettamento e realizzazione del Manifest

A seguito dell'applicazione di entrambe le trasformazioni xslt avremo in output un documento xhtml da validare rispetto alla grammatica rash. Se tale validazione ha successo sarà possibile procedere con l'impacchettamento del documento e l'aggiornamento del Manifest, altrimenti, l'utente verrà informato dell'invalidità e l'applicazione cancellerà il documento e tutti i file intermedi di lavoro creati.

La realizzazione del Manifest avviene ancora prima che si verifichino le trasformazioni xslt sul documento d'interesse dell'utente, in particolare, come mostrato nella Figura 6, l'utente può creare un nuovo workspace cliccando sull'opportuno bottone. A seguito del click verrà fornita un'area di testo compilabile con il nome del nuovo workspace che si intende creare. Ecco che nel momento in cui l'utente conferma la creazione, l'applicazione andrà a creare nel file System sia la directory referenziante il workspace sia il file JSON Manifest proprio della collezione stessa. Tale file sarà denominato con lo stesso nome della collezione.

Figura 18: Struttura interna file JSON Manifest di una singola collezione

In Figura 18 viene mostrata la struttura interna del file Manifest contenente una serie di metadati referenzianti informazioni caratteristiche della collezione. Se il documento xhtml, output delle due trasformazioni xslt, risulta essere conforme alla grammatica rash, è necessario impacchettarlo e aggiornare il Manifest.

Per impacchettarlo, l'applicazione prende il nome dell'opera contenuta nel documento, lo parsa trasformando i caratteri in ASCII, rimuovendo gli spazi vuoti e trasformando i caratteri in minuscolo, e infine crea una sottodirectory con tale nome, interna alla directory referenziante la collezione, in cui scrive il documento xhtml rash valido derivante dalle trasformazioni xslt, denominandolo "index.xhtml".

Successivamente, procede con l'aggiornamento del file Manifest, in particolare, crea un nuovo oggetto JSON con la stessa struttura interna di quelli mostrati nell'array docs nella Figura 18. Di seguito, preleva dal documento TEI iniziale i valori relativi a titolo, anno e autore dell'opera e popola le proprietà dell'oggetto. Il valore della proprietà path corrisponde al nome della sottodirectory, in cui risiede il documento, preceduta da "./". Infine, inserisce il nuovo oggetto in coda all'array docs e riscrive il file.

Se il processo di impacchettamento del documento e di aggiornamento del Manifest si completa con successo, allora, l'utente verrà informato e potrà visualizzare la nuova opera tra l'elenco del workspace. In questo modo, otteniamo una struttura complessiva in linea con i requisiti dell'API DocuDipity.

Capitolo 4

4 DETTAGLI IMPLEMENTATIVI

In questo capitolo andrò a fornire alcuni dettagli sull'implementazione dell'applicazione web tei2rash. Inizialmente, verrà fornita una descrizione sull' organizzazione e la struttura interna complessiva e di seguito verranno illustrate, nel dettaglio, le principali funzioni client e server side implementate.

4.1 Struttura e organizzazione interna

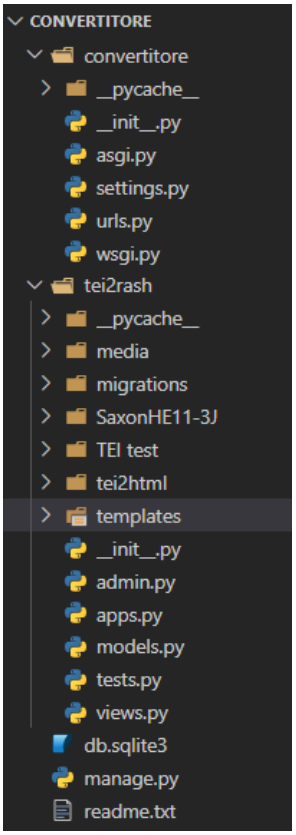

Figura 19: Struttura complessiva del progetto

Tei2rash è un'applicazione web scritta in python realizzata mediante l'impiego del framework Django[37], di conseguenza sarà ospitata all'interno di un progetto Django. In Figura 19, viene mostrata la struttura interna complessiva, gli script Python di configurazione del progetto vengono memorizzati nella directory convertitore, al suo interno troviamo:

- /settings.py è un file contenente tutte le informazioni relative alle impostazioni e alla configurazione dell'intero progetto.

- /urls.py è un file in cui viene definito il modulo URLconf, quest'ultimo è scritto in puro codice Python e permette una mappatura tra gli url delle richieste client side e le corrispettive callback da invocare server side per adempiere al servizio richiesto. Quando un utente compie una richiesta, Django carica il modulo URLconf[38] leggendolo dal valore della property ROOT_URLCONF definita nel file settings.py e successivamente parsa il contenuto della variabile urlpatterns. Quest'ultima è un elenco di funzioni path del modulo django.urls costituite da due argomenti: un URL pattern e una funzione di callback. Django confronta in successione ordinata i pattern URL con l'url della richiesta stessa e a fronte di una corrispondenza viene invocata la corrispettiva funzione di callback. Alla funzione di callback vengono implicitamente passati sia la richiesta sia i parametri definiti tra '<>' dell'URL pattern, terminato il comportamento ritornano una risposta http al Client. In Figura 20 viene mostrato l'URLconf del progetto Django realizzato.

[37] https://www.djangoproject.com/
[38] https://docs.djangoproject.com/en/4.1/topics/http/urls/

```python
from django.contrib import admin
from django.urls import path
from tei2rash import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('elabora_opera/', views.get_tei),
    path('aggiungi_workspace/', views.aggiungi_workspace),
    path('get_workspaces/', views.get_all_warkspace),
    path('remove_workspace/<str:workspace>', views.remove_workspace),
    path('get_info_workspace/<str:workspace>/', views.get_info_workspace),
    path('rimuovi_opera/', views.remove_opera),
    path('elabora_opera_file/', views.elabora_opera_file)
]
```

Figura 20: URLconf del progetto definito nel file convertitore/urls.py

L'applicazione web tei2rash è contenuta all'interno della directory tei2rash, sotto modulo del progetto Django. Al suo interno troviamo:

- /media è una directory di lavoro in cui sono contenuti il foglio di stile xslt file_processor.xsl (risultato del processo definito nella Sezione 3.2.3.1), lo schema xml RELAX NG rash.rng contenente la grammatica rash e la directory RASH in cui vengono salvati i workspace creati dall'utente utilizzatore.

- /SAXONHE11-3J è una directory che contiene il jar del processore xslt SaxonJ necessario ad eseguire le trasformazioni xslt.

- /tei2html è la directory contenente la collezione di fogli di stile xslt necessari a compiere il processo di trasformazione intermedia dal formato TEI al formato XHTML.

- /templates è la directory contenente la pagina web HTML index.html dell'applicazione renderizzata all'utente per potervi interagire.

- /views.py è il file contenente l'intera business logic dell'applicazione, al suo interno è implementato il comportamento di tutte le funzioni di callback definite nel URLconf mostrato in Figura 20. Tali funzioni chiamate viste assumono in input una richiesta http, eventuali parametri, e ritornano in output una risposta http. Inoltre, contiene una serie di funzioni ausiliarie richiamate all'interno delle callback per adempiere ai servizi richiesti dall'utente.

Infine, a partire dalla root directory CONVERTITORE del progetto troviamo i file:

- /manage.py è un file costituente una utility da riga di comando che consente di interagire con il progetto stesso in vari modi.

- /readme.txt è un file di testo in cui vengono definiti i requisiti di sistema e i passi da seguire per lanciare l'applicazione tei2rash mediante il server di sviluppo Django.

4.2 Tecnologie e le librerie adottate

Di seguito verrà fornita una breve descrizione sulle tecnologie e le librerie adottate per la realizzazione dell'applicazione.

- HTML o HyperText Markup Language è il linguaggio di markup utilizzato per realizzare il sorgente della pagina web reindirizzata all'utente utilizzatore. Tale linguaggio "marca": la struttura del documento, le informazioni di presentazione, i collegamenti ipertestuali e le risorse multimediali; il browser è in grado di interpretare tali informazioni e visualizzare la pagina web.

- CSS o Cascading Style Sheet[39] è il linguaggio di stylesheet utilizzato per definire le caratteristiche tipografiche e di layout della pagina web.

- Javascript[40] è il linguaggio di script adottato client-side per fornire dinamicità ed interattività alla pagina web, in particolare viene implementato asincronicamente associando gli script al verificarsi di eventi sul documento, al completamento di un'operazione di rete o a un time-out.

[39] https://www.w3.org/Style/CSS/Overview.en.html
[40] https://www.javascript.com/

- AJAX[41] o Asynchronous Javascript And XML è una tecnica per la creazione di applicazioni web interattive, permette l'aggiornamento asincrono di porzioni di pagine HTML. È un termine che indica l'utilizzo di una combinazione di tecnologie comunemente utilizzate sul web: XHTML e CSS, DOM[42] modificato attraverso Javascript per la manipolazione dinamica dei contenuti, XMLHttpRequest per lo scambio di messaggi asincroni fra browser e web server e XML o JSON come metalinguaggi dei dati scambiati.

- JQuery[43] è una libreria Javascript che ho adottato client-side per semplificarmi il processo di manipolazione e personalizzazione delle chiamate AJAX asincrone e del DOM.

- Python è il linguaggio di programmazione realizzante l'intero modulo server side dell'applicazione.

- Aiohttp[44] è un framework client/server http asincrono python, viene adottato server side, in combinazione alla propria libreria Session, per realizzare la richiesta get http all'uri fornito in input dall'utente.

- Lxml[45] è una libreria ricca di funzionalità per l'elaborazione di documenti XML e HTML in python, viene utilizzata server side per la validazione[46] del documento xhtml, output della seconda trasformazione, rispetto alla grammatica rash.

- Unidecode[47] è una libreria che fornisce la funzione unidecode che assume in input una stringa Unicode e la trasforma in ASCII; tale funzionalità viene adoperata server side per ottenere il nome della directory in cui viene salvato il file rash valido a seguito delle trasformazioni xslt.

[41] https://www.w3schools.com/xml/ajax_intro.asp
[42] https://www.w3.org/TR/REC-DOM-Level-1/introduction.html
[43] https://jquery.com/
[44] https://docs.aiohttp.org/en/stable/
[45] https://lxml.de/
[46] https://lxml.de/api/lxml.etree.RelaxNG-class.html
[47] https://pypi.org/project/Unidecode/

- Beautifulsoup4 è una libreria che fornisce metodi e idiomi python per parsare, navigare e analizzare documenti html e xml, le sue funzionalità vengono adoperate server side per estrapolare i dati dai documenti d'origine TEI, così da popolare opportunamente il Manifest a fronte di una conversione avvenuta con successo.

4.3 Le funzioni

In questa sezione verrà fornita una descrizione delle principali funzioni implementate client e server side.

4.3.1 Client Side

```
function elabora_tei(workspace){
    var input = document.getElementById('url')
    if(!input.checkValidity()){
        display_feed_utente('url non valido', false)
        return
    }

    var data = new FormData();

    data.append("url", $("input[id^='url']")[0].value);
    data.append("workspace", workspace);
    data.append("csrfmiddlewaretoken", "{{ csrf_token }}");

    $.ajax({
        method: "post",
        url: "/elabora_opera/",
        processData: false,
        contentType: false,
        mimeType: "multipart/form-data",
        data: data,
        success: (res)=>{
            var obj = JSON.parse(res)
            if(obj.status==true){
                var text = "conversione avvenuta con successo"
                display_feed_utente(text, true)
                get_info_workspace(workspace)
                var section_form = document.getElementById("load")
                section_form.innerHTML = ""
                document.querySelectorAll(".btn_info_work").forEach(e => e.disabled = false)
                document.querySelectorAll(".btn_rmv").forEach(e => e.disabled = false)
                document.querySelectorAll(".btn_work_del").forEach(e => e.disabled = false)
            }else{
                var text = `${obj.status}`
                display_feed_utente(text, false)
                get_info_workspace(workspace)
                var section_form = document.getElementById("load")
                section_form.innerHTML = ""
                document.querySelectorAll(".btn_info_work").forEach(e => e.disabled = false)
                document.querySelectorAll(".btn_rmv").forEach(e => e.disabled = false)
                document.querySelectorAll(".btn_work_del").forEach(e => e.disabled = false)
            }
        },
        error: (err)=>{
            //alert(err)
            display_feed_utente("Errore", false)
        }
    })
}
```

Figura 21: Codice funzione elabora_tei

In Figura 21 viene mostrato il codice della funzione `elabora_tei` incaricata di scatenare il processo di trasformazione xslt server side sulla risorsa TEI dell'utente:

- La funzione assume in input il `workspace` in cui l'utente utilizzatore vuole aggiungere il nuovo documento.

- Definisce una variabile `input` a cui assegna l'elemento html input con attributo type url identificato mediante l'attributo id.

- Mediante il metodo Javascript `checkValidity` verifica che il valore dell'url sottoforma di stringa dell'elemento html `input` sia valido.

- Se l'url inserito dall'utente è valido, viene istanziato un oggetto FormData `data`, costituisce un set compilabile di coppie chiave/valore trasmissibile con un oggetto XMLHttpRequest. Tale set viene popolato con il valore dell'url, il valore del workspace e un token CSRF[48] per superare il middleware di protezione Django da attacchi cross-site request forgery.

- Di seguito, viene utilizzato il metodo JQuery $.ajax({opzioni})[49] che permette di effettuare una richiesta Ajax http asincrona. La richiesta Ajax è composta da una serie di opzioni definite nella forma chiave: valore che ne permettono la configurazione. Il valore di method definisce il metodo http della richiesta Ajax; la chiave url contiene la stringa referenziante l'URL a cui inviare la richiesta; la chiave data contiene i dati da inoltrare al server, in questo caso si tratta di un oggetto e affinché JQuery non lo trasformi in stringa è necessario settare la chiave processData a false; la chiave success assume come valore una funzione di callback invocata se la richiesta ha esito positivo, la callback assume in input i dati ritornati in risposta dal server. In particolare, crea un oggetto JSON con i dati della risposta, a seconda dell'esito della conversione informa l'utente richiamando la funzione ausiliaria `display_feed_utente` e ri visualizza il workspace

[48] https://docs.djangoproject.com/en/4.1/ref/csrf/
[49] http://api.jquery.com/jquery.ajax/

corrente, con l'eventuale nuovo documento, invocando la funzione `get_info_workspace`.

Vi è la funzione `elabora_tei_file` che compie le stesse funzionalità descritte per la funzione `elabora_tei` ma, anziché, passare nel campo data il valore stringa dell'url passa un oggetto File ottenuto dall'input con attributo type file.

```
function get_info_workspace(workspace){
    $.ajax({
        type: "get",
        url: "/get_info_workspace/"+workspace,
        success: (res)=>{
            if(res.status==true){
                var documenti = res.documenti
                var titolo = res.titolo
                var section_documenti = document.getElementById('display_opere');
                section_documenti.innerHTML = "";
                var div_option = document.createElement('div')
                div_option.setAttribute('id', "div_option")
                var btn_add = document.createElement('button')
                btn_add.innerHTML = "aggiungi un'opera"
                btn_add.setAttribute('id', 'btn_opt')
                btn_add.setAttribute('onclick', `aggiungi_opera("${workspace}")`)
                var titolo_sec = document.createElement('h3');
                titolo_sec.setAttribute('id', 'titolo_opt')
                titolo_sec.innerHTML = titolo
                div_option.appendChild(titolo_sec)
                div_option.appendChild(btn_add)
                section_documenti.appendChild(div_option)
                for(const documento of documenti){
                    var div_documento = document.createElement('div');
                    var btn_documento = document.createElement('button');
                    btn_documento.innerHTML = "rimuovi"
                    btn_documento.setAttribute('class','btn_rmv')
                    btn_documento.setAttribute('onclick', `rimuovi_opera("${workspace}", "${documento.title}")`)
                    div_documento.setAttribute('class', 'menu');
                    div_documento.innerHTML = ` Titolo: ${documento.title} <br>
                                                Anno: 1${documento.year} <br>
                                                Autore: ${documento.authors}`;
                    div_documento.appendChild(btn_documento);
                    section_documenti.appendChild(div_documento);
                }
                var btn_docudipity = document.createElement('button')
                btn_docudipity.setAttribute('id','btn_docudipity')
                btn_docudipity.setAttribute('onclick','carica_workspace()')
                btn_docudipity.innerHTML = `CARICA SU DOCUDIPITY`
                section_documenti.appendChild(btn_docudipity)
            } else{
                display_feed_utente("Errore nel caricamento delle opere", false)
            }
        },
        error: (err)=>{
            //alert(err)
            display_feed_utente("Errore", false)
        }
    })
}
```

Figura 22: Codice funzione get_info_workspace

In Figura 22 viene mostrato il codice della funzione `get_info_workspace` incaricata della visualizzazione dinamica ed interattiva delle opere contenute in un singolo workspace:

- La funzione assume in input il `workspace` d'interesse per l'utente.

- Di seguito viene compiuta una chiamata http ajax asincrona, in questo caso essendo una richiesta con metodo http get non è necessario specificare gran parte delle opzioni di configurazione viste nella descrizione della funzione mostrata in Figura 21.
- Anche in questo caso, a fronte di richiesta compiuta con esito positivo, nella chiave success viene definita una funzione di callback implementante il seguente comportamento: assegna alla variabile `documenti` la lista dei documenti presenti nel workspace d'interesse e nella variabile `titolo` il nome del workspace, entrambi i valori vengono prelevati dall'oggetto json ritornato in risposta dal server; seleziona l'elemento html section dedicato alla visualizzazione dei documenti del singolo workspace e lo assegna alla variabile `section_documenti`; crea un elemento html div in cui inserisce un bottone dedicato all'inserimento di un nuovo documento e lo aggiunge in testa a `section_documenti` mediante la funzione appendChild; scorre la lista documenti e per ogni documento crea un elemento html div in cui inserisce il contenuto testuale referenziante le informazioni caratteristiche del singolo documento mediante la proprietà innerHTML e vi aggiunge un bottone dedicato all'eliminazione del documento stesso dal workspace; le funzionalità dei bottoni vengono definite mediante il valore dell'attributo evento onclick; ogni div dedicato al singolo documento viene aggiunto a `section_documenti` e infine viene aggiunto il bottone incaricato di richiamare la funzione `carica_wokspace` dedica al caricamento dell'intero workspace su DocuDipity, la cui funzionalità non è ancora stata implementata.

Di seguito andrò a descrivere il comportamento di massima delle altre funzioni definite la cui implementazione risulta essere similare alle due precedentemente descritte.

- `elimina_workspace` è la funzione incaricata di scatenare il processo di eliminazione del singolo workspace sul server, viene invocata quando l'utente utilizzatore interagisce cliccando sul bottone "rimuovi" mostrato in Figura 6.

- `display_aggiungi_workspace` è la funzione incaricata di visualizzare il campo di testo compilabile dall'utente con il nome del nuovo workspace, e due

bottoni rispettivamente per confermare o annullare l'aggiunta del nuovo workspace.

- `aggiungi_workspace` è la funzione incaricata di scatenare il processo di creazione di un nuovo workspace sul server, viene invocata quando l'utente clicca sul bottone per confermare l'aggiunta di un nuovo workspace. Prima di procedere al compimento della richiesta http ajax asincrona controlla che il nome del workspace sia valido; quindi, che non contenga caratteri proibiti per nominare una directory.

- `annulla_aggiungi_workspace` è la funzione che annulla l'operazione di creazione di un nuovo workspace invocata quando l'utente clicca sul bottone per annullare l'aggiunta di un nuovo workspace. Elimina il nuovo campo di testo e richiama la funzione `get_all_workspace`.

- `get_all_workspace` è la funzione incaricata di recuperare dal server tutti i workspace creati dall'utente, anche in questo caso viene compiuta una richiesta http ajax e il contenuto della risposta viene parsato per poterlo visualizzare a video come mostrato in Figura 6.

- `rimuovi_opera` è la funzione incaricata di scatenare il processo di rimozione della singola opera appartenente ad uno specifico workspace sul server. Compie una chiamata http ajax a cui passa come valori nel campo data, oltre al token CSRF, anche il titolo del documento e il corrispettivo workspace di appartenenza.

- `display_feed_utente` è la funzione incaricata di fornire feedback a video sullo stato di successo o insuccesso delle operazioni compiute dall'utente utilizzatore.

4.3.2 Server Side

Per quanto riguarda la componente server side andrò a dettagliare le funzioni di callback definite nell'URLconf del progetto, mentre fornirò una descrizione di massima per le funzioni ausiliarie secondarie.

```python
async def elabora_opera_file(request):
    tei_file = request.FILES['file']
    workspace = request.POST.get('workspace')
    start_time = time.time()

    xml = gestione_file(tei_file)

    is_xml_valid = check_body_response(xml)
    if is_xml_valid==False:return JsonResponse({"status": "documento xml non valido"})
    out_path = define_out_path(xml)

    if out_path == '':return JsonResponse({"status": "documento xml non valido"})
    is_write = write_xml_file(out_path, xml)
    if is_write==False:return JsonResponse({"status": "documento xml non valido"})
    out_name = out_path.split('/')[3].replace('xml', 'xhtml')

    #check duplicato tei2html
    cartella = get_nome_cartella(unquote(out_name))

    if cartella == '' : return JsonResponse({"status": "impossibile convertire l'xml in html"})
    path_to_check = f"tei2rash/media/RASH/{workspace}/{cartella}/index.xhtml"

    if(os.path.exists(path_to_check)) : return JsonResponse({"status": "risorsa rash già presente in questo workspace"})

    tei2html = elabora_tei(out_path, out_name)
    if tei2html==False : return JsonResponse({"status": "impossibile convertire l'xml in html"})

    #html2rash
    out = "tei2rash/media/HTML/"+out_name
    out = unquote(out)
    #html2rash = elabora_html(out, out_name, workspace, cartella)
    html2rash = elabora_html(out, path_to_check)
    if html2rash==False: return JsonResponse({"status": "impossibile convertire l'html in rash"})

    #rashvalidation e update manifest
    path_to_check = unquote(path_to_check)
    is_rash_valid = check_rash_validator(path_to_check)
    if is_rash_valid == False :
        rimuovi_rash_opera(unquote(out_name), workspace)
        rimuovi_fogli_di_lavoro(out_path, out)
        return JsonResponse({"status": "il documento prodotto non è rash valido."})

    update_manifest = aggiorna_manifest(out, cartella, workspace)
    if update_manifest == False :
        rimuovi_rash_opera(unquote(out_name), workspace)
        rimuovi_fogli_di_lavoro(out_path, out)
        return JsonResponse({"status": "impossibile aggiornare il manifest"})

    rimuovi_fogli_di_lavoro(out_path, out)

    total_time = time.time() - start_time
    print(total_time)

    return JsonResponse({"status": True})
```

Figura 23: Codice funzione elabora_opera_file

In Figura 23 viene mostrato il codice della funzione `elabora_opera_file` incaricata di aggiungere un nuovo documento al workspace d'interesse dell'utente. Il

comportamento complessivo della funzione costituisce l'intero processo definito dal diagramma mostrato in Figura 10.

- La funzione assegna alla variabile `tei_file` un oggetto UploadedFile[50]. L'istruzione request.FILES[51] ritorna un dizionario di oggetti UploadedFile, ogni oggetto al suo interno è identificato da una chiave il cui valore corrisponde alla chiave con cui il file è stato aggiunto al FormData, contenuto del campo data della richiesta http Ajax eseguita client side. Mentre, alla variabile `workspace` viene assegnato il nome del workspace a cui l'utente vuole aggiungere il documento. L'istruzione request.POST[52] permette di accedere al campo data della richiesta http assunta in input e tramite la get preleva il valore dell'opportuna chiave. Ho dovuto utilizzare due istruzioni differenti in quanto request.POST non include le informazioni sul caricamento di file.

- Mediante la funzione `gestione_file` viene assegnata alla variabile `xml` il contenuto stringato dell'oggetto UploadedFile.

- `is_xml_valid` è una variabile booleana che assume valore True se il contenuto stringato costituisce un documento xml ben formato, altrimenti, False. L'analisi della stringa viene compiuta dal metodo `check_body_response` mediante la funzione fromstring del modulo xml.etree.ElementTree[53].

- `out_path` è una variabile a cui viene assegnato il path del file xml TEI input del primo processo di trasformazione xslt.

- La funzione `write_xml_file` assume in input il path e il contenuto stringato del file xml TEI, in particolare, mediante la funzione integrata[54] open, con parametro settato a 'w', crea un nuovo file nel file system al path di input e vi scrive il contenuto definito dalla stringa.

[50] https://docs.djangoproject.com/en/4.1/ref/files/uploads/#django.core.files.uploadedfile.UploadedFile
[51] https://docs.djangoproject.com/en/4.1/ref/request-response/#django.http.HttpRequest.FILES
[52] https://docs.djangoproject.com/en/4.1/ref/request-response/
[53] https://docs.python.org/3/library/xml.etree.elementtree.html
[54] https://docs.python.org/3/library/functions.html

- Alla variabile `out_name` viene assegnato il nome del file xml TEI con estensione sostituita da '.xml' a '.xhtml'.

- Alla variabile `cartella` viene assegnato il nome della directory in cui verrà salvato il file xhtml rash valido, output del processo complessivo di trasformazione xslt, mentre, il path relativo di tale file viene assegnato alla variabile `path_to_check`. Il modulo os.path[55] permette di interagire con il file system, di tale modulo viene richiamata la funzione exists, a cui viene passato in input la variabile `path_to_check`, per verificare che il workspace corrente non contenga già il documento al suo interno.

- La funzione `elabora_tei` è incaricata di eseguire il primo processo di trasformazione xslt sul documento xml TEI. Il comando per compiere la trasformazione xslt è descritto nella Sezione 3.2.2.3. La funzione `elabora_tei` assume in input due parametri `out_path` e `out_name`; `out_path` viene passato come valore del parametro '-s' al comando, mentre, `out_name` viene utilizzato per definire il path del documento xhtml risultato della trasformazione xslt, nonché valore del parametro '-o'.

- Alla variabile `out` viene assegnato il path del documento xhtml intermedio risultato del primo processo di trasformazione realizzato dalla funzione `elabora_tei`.

- La funzione `elabora_html` è incaricata di eseguire il secondo processo di trasformazione xslt sul documento xhtml output della prima trasformazione. Quest'ultima assume in input due parametri `out` e `path_to_check`; `out` viene passato come valore del parametro '-s' del comando di trasformazione xslt, mentre, `path_to_check` viene passato come valore del parametro '-o'.

[55] https://docs.python.org/3/library/os.html

- La funzione `check_rash_validator` è incaricata di verificare che il documento xhtml derivante dal processo di trasformazione xslt complessivo sia conforme alla grammatica rash; ritorna True o False a seconda che il documento sia valido rispetto allo schema RelaxNG.

- La funzione `rimuovi_rash_opera` elimina dal file system il documento xhtml finale qualora non risulti essere valido o si verifica un errore durante l'aggiornamento del Manifest, mentre, `rimuovi_fogli_di_lavoro` elimina i documenti intermedi necessari al compimento del processo complessivo di trasformazione xslt.

- La funzione `aggiorna_manifest` aggiorna il file Manifest del workspace corrente andando ad aggiungere all'array `docs` l'oggetto JSON contenente i metadati referenzianti le informazioni caratteristiche del nuovo documento xhtml rash valido aggiunto.

Vi è la funzione `get_tei` che implementa lo stesso comportamento della funzione `elabora_opera_file`, ma, anziché operare sul documento xml TEI caricato dall'utente, opera sul documento ottenuto compiendo una richiesta get http asincrona all'url specificato dall'utente nel FormData passato come valore del campo data nella richiesta http ajax client side.

```
def aggiungi_workspace(request):
    if request.method == 'POST':
        new_workspace = request.POST.get('workspace')
        is_add = add_new_workspace(new_workspace)
        if is_add : return JsonResponse({'data' : True})
        else : return JsonResponse({'data' : False})
    return JsonResponse({'data': False})
```

Figura 24: Codice funzione aggiungi_workspace

In Figura 24 viene mostrato il codice della funzione `aggiungi_workspace` incaricata di aggiungere un nuovo workspace per l'utente utilizzatore.

- Alla variabile `new_workspace` viene assegnato il nome del nuovo workspace che si vuole realizzare, anche in questo caso, ottenuto accedendo all'opportuna chiave del campo data della richiesta http request, input della funzione.

- Di seguito, viene richiamata funzione `add_new_workspace` incaricata di creare nel file system la nuova directory padre della collezione. La funzione assume in input il nome del nuovo workspace, utilizza la funzione exists del modulo os.path per verificare che non ci sia già un workspace con tale nome. Se già presente ritorna False, altrimenti, crea la directory e richiama la funzione `add_manifest_to_workspace` che crea il file Manifest all'interno della directory.

```
def get_all_warkspace(request):
    dirs = os.listdir(default_path)
    directories = []
    i = 0
    for dir in dirs:
        directories.append(dir)

    return JsonResponse({"workspaces": directories})
```

Figura 25: Codice funzione get_all_workspace

In Figura 25 viene mostrato il codice della funzione `get_all_workspace` incaricata di fornire al chiamante client side la lista complessiva dei workspace creati.

- La funzione listdir del modulo os ritorna in output i nomi di tutte le directory del path assunto in input. Tale collezione viene scandita mediante un ciclo for e ogni nome viene aggiunto all'array `directories` ritornato in risposta.

```python
def remove_workspace(request, workspace):
    folder_path = default_path + workspace
    if os.path.exists(folder_path):
        if len(os.listdir(folder_path)) == 1:   #solo il manifest
            is_removed = remove_manifest(workspace)
            if is_removed : os.rmdir(folder_path)
        else: return JsonResponse({"status": False})
    return JsonResponse({"status": True})
```

Figura 26: Codice funzione remove_workspace

In Figura 26 viene mostrato il codice della funzione `remove_workspace` incaricata di rimuovere un workspace.

- La funzione controlla che il workspace sia privo di documenti rash; quindi, che contenga solo il file Manifest; richiama `remove_manifest` che elimina il file Manifest dal file system mediante il metodo remove del modulo os; e infine, rimuove la directory referenziante il workspace.

```python
def get_info_workspace(request, workspace):
    path_manifest = default_path +f"/{workspace}/{workspace}.json"
    if os.path.exists(path_manifest):
        with open(f'tei2rash/media/RASH/{workspace}/{workspace}.json', 'r', encoding='utf-8') as json_f:
            data = json.load(json_f)
            documenti = data["docs"]
            to_return = []
            for doc in documenti:
                to_return.append(doc)
            print(to_return)
            return JsonResponse({"status": True, "documenti": to_return, "titolo":workspace})
    else: return JsonResponse({"status": False})
```

Figura 27: Codice funzione get_info_workspace

In Figura 27 viene mostrato il codice della funzione `get_info_workspace` incaricata di fornire al chiamante client side le informazioni dei documenti xhtml rash validi contenuti nel singolo workspace.

- La funzione utilizza la funzione integrata open, con parametro settato a 'r', per leggere il contenuto del file Manifest del workspace d'interesse per l'utente utilizzatore. La variabile `data` è un dizionario[56] il cui contenuto è assegnato dal

[56] https://docs.python.org/3/tutorial/datastructures.html

metodo load del modulo json[57]. Alla variabile `documenti` viene assegnato il valore dell'array `docs` prelevato dal dizionario data. Infine, ogni oggetto contenuto in `docs` viene inserito nell'array `to_return` ritornato in risposta.

```
def remove_opera(request):
    opera = request.POST.get('opera')
    workspace = request.POST.get('workspace')
    is_remove_rash = rimuovi_rash_opera(opera, workspace)
    if is_remove_rash == False : JsonResponse({"status": "opera rash non trovata"})
    rimuovi_opera_manifest(opera, workspace)
    return JsonResponse({"status": "opera eliminata"})
```

Figura 28: Codice funzione remove_opera

In Figura 28 viene mostrato il codice della funzione `remove_opera` incaricata di eliminare un documento xhtml rash da un workspace.

- Alle variabili `opera` e `workspace` vengono assegnati rispettivamente i nomi del documento e del workspace. La funzione `rimuovi_rash_opera` rimuove il documento xhtml rash identificandolo dal path costruito con i valori delle variabili `opera` e `workspace`.

- Di seguito, viene richiamata la funzione `rimuovi_opera_manifest` incaricata di rimuovere il documento in questione dall'array `docs` del file Manifest proprio del workspace.

[57] https://docs.python.org/3/library/json.html

Capitolo 5

5 VALUTAZIONE E CONCLUSIONI

Dopo aver ampiamente discusso l'implementazione complessiva del Software, è necessario fornire un'analisi dei test compiuti volti a verificarne il funzionamento, dei limiti riscontrati ed eventuali sviluppi futuri per migliorare l'applicazione tei2rash.

5.1 Testing e analisi

L'applicazione tei2rash è stata testata su una collezione di documenti xml TEI prelevati dal catalogo Bibit di Biblioteca Italiana. Tali documenti sono contenuti nella directory TEI test al path convertitore/tei2rash/TEI test. La directory contiene, oltre alla collezione di documenti, anche un file txt test_url in cui sono ospitati gli url di ciascun documento, così da poter testare l'applicazione sia mediante l'upload di file dal file system, sia mediante il caricamento dell'url della risorsa remota. I documenti derivanti dal processo complessivo di trasformazione xslt sono stati validati rispetto alla grammatica formale RASH[58] definita in RELAX NG.

Documento file system	Dimensione in KB	Tempo di elaborazione in s	Rash valido
Vita nuova.xml	122 KB	4.332 s	☑
Telesilla.xml	49 KB	3.958 s	☑
Sul Romanticismo. Lettera al marchese Cesare D'Azeglio.xml	128 KB	3.993 s	☑
Storia d'Italia.xml	3835 KB	6.889 s	☑
Senilità.xml	414 KB	4.469 s	☑
Rime.xml	79 KB	4.119 s	☑
L'unità dell'Italia e la quadratura del circolo.xml	8 KB	3.374 s	☑
Lettera al Casanova.xml	22 KB	3.520 s	☒
La via del rifugio.xml	62 KB	4.021 s	☑
La Feroniade.xml	85 KB	4.340 s	☑
Il podere.xml	271 KB	4.421 s	☑
Deifira.xml	51 KB	3.454 s	☑
Consolatoria a Pino de' Rossi.xml	53 KB	3.638 s	☑
Canti di Castelvecchio.xml	173 KB	4.941 s	☑
Ajace.xml	114 KB	4.282 s	☒

Figura 29: Esiti test upload file dal file system

[58] https://github.com/essepuntato/rash/blob/master/grammar/rash.rng

Documento url	Dimensione in KB	Tempo di elaborazione in s	Rash valido
Vita nuova.xml	122 KB	4.446 s	☑
Telesilla.xml	49 KB	4.300 s	☑
Sul Romanticismo. Lettera al marchese Cesare D'Azeglio.xml	128 KB	4.139 s	☑
Storia d'Italia.xml	3835 KB	7.099 s	☑
Senilità.xml	414 KB	4.595 s	☑
Rime.xml	79 KB	4.810 s	☑
L'unità dell'Italia e la quadratura del circolo.xml	8 KB	3.646 s	☑
Lettera al Casanova.xml	22 KB	3.671 s	☒
La via del rifugio.xml	62 KB	4.708 s	☑
La Feroniade.xml	85 KB	4.491 s	☑
Il podere.xml	271 KB	4.720 s	☑
Deifira.xml	51 KB	3.535 s	☑
Consolatoria a Pino de' Rossi.xml	53 KB	3.838 s	☑
Canti di Castelvecchio.xml	173 KB	5.393 s	☑
Ajace.xml	114 KB	4.381 s	☒

Figura 30: Esiti test upload url risorsa remota

Documento file system		Documento url	
Dimensione media in KB	Tempo medio di elaborazione in s	Dimensione media in KB	Tempo medio di elaborazione in s
Dim > 116 KB	**4.761 s**	**Dim > 116 KB**	**4.967 s**
Dim < 116 KB	**3.803 s**	**Dim < 116 KB**	**4.124 s**

Figura 31: Analisi di confronto dei test eseguiti

In Figura 29, sono mostrati i dati derivanti dai test compiuti mediante il caricamento del documento xml TEI dal file system, mentre, in Figura 30 sono mostrati i dati derivanti dai test compiuti mediante il caricamento dell'url della risorsa xml TEI remota. Indipendentemente dalla modalità di caricamento, l'applicazione tei2rash fornisce documenti di output xhtml rash validi nel 86,7% dei casi di test. I fallimenti derivano da annidamenti di tag xhtml non individuati durante l'analisi della struttura complessiva dei documenti, output della prima trasformazione xslt. Questo si ripercuote nella realizzazione del foglio di stile xslt file_processor, i cui template non saranno in grado di trasformarli in strutture conformi alla grammatica rash, nel documento di output xhtml. Entrando nel dettaglio pratico, in Ajace, si verifica la seguente struttura: vi è un div di classe div1 padre di un div di classe divHead che è padre di uno span. Il template catturante il nodo padre div di classe div1 definisce istruzioni xsl per cui il div viene tradotto in una section, all'interno della quale viene richiamata l'istruzione xsl: apply-template che andrà ad invocare tutti i templates sugli elementi figli, quindi anche su divHead. Il template catturante il div di classe divHead lo omette, e se presente un nodo d'intestazione lo traduce in un elemento h1, altrimenti, inserisce un'intestazione fittizia; di seguito va a richiamare l'istruzione xsl: apply-templates. Qui, si riscontra l'errore in

quanto la section padre, derivante della traduzione del div di classe div1, avrà tra i figli diretti un elemento span, risultato della traduzione del template applicato allo stesso. Come detto in precedenza, tale problematica deriva dall'analisi delle strutture interne ricorrenti dei documenti xhtml, output del primo processo di trasformazione xslt; sarà necessario andare a modificare il comportamento del template catturante l'elemento span, in funzione del nodo padre di cui è figlio. In Figura 31, sono mostrati i dati derivanti dalle due modalità di caricamento a confronto, sia per documenti con dimensione superiore alla media, sia per documenti con dimensione inferiore alla media, la modalità di caricamento dell'url richiede un tempo di elaborazione complessivo superiore, questo deriva dalla necessità di compiere una richiesta get http all'url specificato per reperire la risorsa xml TEI remota. Il tempo che intercorre tra l'invio della richiesta e la ricezione della risposta comporta un tempo addizionale al complessivo d'elaborazione.

5.2 Conclusioni, limiti e sviluppi futuri

La presente tesi si pone come obbiettivo primario l'implementazione di un sistema che permetta ad un utente di caricare documenti in formato TEI sulla piattaforma DocuDipity, a tal proposito è stata realizzata l'applicazione web tei2rash. TEI è stato scelto come formato d'origine per i molteplici vantaggi descritti nella Sezione 2.1. Affinché il processo di elaborazione ed organizzazione dei documenti sia funzionale all'obbiettivo da raggiungere, bisogna soffermarsi sull'analisi dei requisiti imposti dall'API di caricamento della piattaforma. Solamente dopo aver compreso i vincoli strutturali e di formato che la collezione di documenti deve rispettare è possibile implementare un sistema in grado di adempiere a tali necessità. L'applicazione verte su due principali aspetti: la conversione del documento dal formato TEI al formato RASH e l'organizzazione complessiva delle collezioni di documenti. Il primo aspetto viene scandito in due procedure sequenziali, in cui l'output della prima conversione diviene l'input della seconda. La prima prevede la conversione del documento dal formato TEI al formato xhtml, mentre, la seconda prevede la conversione dal formato xhtml al formato rash. Essendo sia TEI, sia xhtml, sia rash formati basati su XML i processi di conversione avvengono mediante l'applicazione di fogli di stile xslt. La prima conversione impiega

l'utility tei2html reperibile online, l'intero processo d'integrazione viene descritto nella Sezione 3.2.2.3; la seconda conversione utilizza il foglio di stile xslt file_processor la cui realizzazione è descritta nella Sezione 3.2.3.1. Il secondo aspetto si concentra sull'organizzazione gerarchica della collezione di documenti. Vengono realizzate le funzionalità per l'estrapolazione delle informazioni caratteristiche della collezione e dei singoli documenti per la realizzazione e il popolamento del file Manifest e la funzionalità necessaria alla manipolazione del documento derivante dal processo complessivo di trasformazione, in particolare, viene validato rispetto alla grammatica rash, e se valido viene inserito in una sottodirectory figlia della directory padre della collezione. L' implementazione ti tali funzionalità è avvenuta in concomitanza alla realizzazione dell'interfaccia utente d'interazione. A seguito del completamento dell'applicazione si è proceduto a testarla su una collezione di documenti TEI, i risultati derivanti dai test sono poi stati analizzati ed interpretati. Di seguito andrò a descrivere quelli che sono i limiti dell'applicazione e alcuni sviluppi futuri la cui realizzazione rende l'applicazione più completa e funzionale; sia per i limiti, sia per gli sviluppi futuri verrà fornito uno spunto implementativo al fine di agevolarne la realizzazione.

Un limite dell'applicazione tei2rash risiede nella mancata gestione delle immagini nel foglio di stile xslt file_processor, il cui processo di realizzazione è descritto nella Sezione 3.2.3.1. Il processo di trasformazione intermedio, mediante l'utility tei2html, è in grado di catturare le immagini dal file xml TEI d'origine e di proiettarle nel documento xhtml, output della trasformazione. Per far sì che le immagini vengano correttamente tradotte anche nel documento xhtml rash finale, è necessario definire un ulteriore template al foglio di stile file_processor. Tale template dovrà avere come valore dell'attributo match un'espressione xPath in grado di catturare l'elemento contenente le immagini nel documento xhtml intermedio, e una serie di istruzioni xslt tali da produrre una struttura complessiva di output conforme alla grammatica RASH. Inoltre, sarà necessario introdurre un meccanismo server side per la gestione del file immagine. L'idea è di prelevare il valore dell'attributo src del tag html img dal documento derivante dalla prima trasformazione xslt, compiere una richiesta http get all'url specificato dal valore prelevato e salvare la risorsa riornata in risposta in una directory figlia della directory in cui viene impacchettato il file xhtml rash valido. Di seguito, bisogna andare a modificare il valore dell'attributo src del tag html img nel file xhtml rash valido, affinché punti al nuovo path

in cui è stata salvata l'immagine. Qualora, la richiesta get all'url della risorsa immagine fallisca si potrebbe introdurre un'immagine di default alternativa.

Una funzionalità addizionale futura dell'applicazione risiede nell'integrazione dell'API del catalogo Bibit di Biblioteca Italiana. Uno spunto d'implementazione è il seguente: al caricamento del DOM della pagina web viene implementata una funzione che, mediante una richiesta get http, preleva dal catalogo Bibit tutti i nomi delle opere xml TEI contenute. Tale set di nomi di opere verrà salvato in una collezione, di seguito andremo a creare un campo testo editabile dall'utente a completamento automatico. Per farlo, possiamo usufruire del modulo Autocomplete della libreria JQuery UI[59]; tale funzionalità migliora l'esperienza di navigazione e ricerca, consente all'utente di trovare e selezionare rapidamente da un elenco precompilato di valori durante la digitazione del testo. A seguito della selezione dell'opera, l'utente potrà caricarla nel proprio workspace con le funzioni server side già presenti. In generale, tale funzionalità permetterà all'utente di poter caricare documenti nel workspace d'interesse senza dover essere a conoscenza dell'url della risorsa remota o di dover avere il documento salvato in locale nel file system.

Un altro sviluppo futuro prevede l'integrazione dell'API di caricamento del workspace sulla piattaforma DocuDipity. Al momento, è presente solamente il bottone necessario a scatenare tale funzionalità, ma l'implementazione concreta non è stata affrontata. A seguito dell'integrazione, l'utente potrà usufruire di tutte le funzionalità di infoview fornite dalla piattaforma stessa sui documenti caricati.

Per risolvere la problematica relativa ai fallimenti dell'applicazione nel produrre documenti rash validi, si potrebbe fare utilizzo del validatore rash-validator[60] contenuto nel RASH framework. Si tratta di uno script python, richiamabile da linea di comando, che consente all'utente di validare i documenti rispetto alla grammatica rash e, qualora, il documento non fosse valido fornisce dettagli sull'errore. L'idea è quella di andare ad analizzare l'errore sollevato, risalire al o ai tag xhtml che lo hanno prodotto e andare a correggere il file xslt file_processor in maniera tale che produca un documento di output conforme alla grammatica rash a fronte di tali situazioni.

[59] https://jqueryui.com/autocomplete/
[60] https://github.com/essepuntato/rash/tree/master/tools/rash-validator

Inoltre, si potrebbe introdurre un sistema di registrazione e login, così che ogni utente possa usufruire di un'area riservata con cui interagire con i propri workspace. Tale sistema può essere integrato sfruttando il modulo d'autenticazione e permessi[61] built-in di Django. Inoltre, si può aggiungere la possibilità di creare workspace annidati catalogandoli per caratteristiche comuni, come ad esempio l'autore degli stessi o la corrente letteraria d'appartenenza.

Infine, si potrebbe compiere un'attività di refactoring[62] dell'intero progetto Django, affinché sposi il pattern architetturale Model View Template per cui è stato pensato.

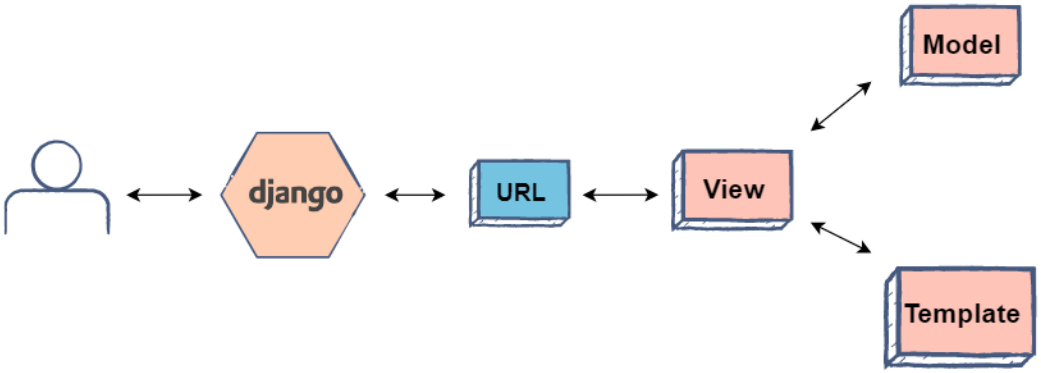

Figura 32: Struttura del pattern architetturale Model View Template

In Figura 32, viene mostrato un diagramma referenziante la struttura del pattern MVT:

- Il Model[63] (tei2rash/models.py) è una classe necessaria a mantenere i campi e i metodi essenziali di un oggetto che si vuole memorizzare. L'idea è quella di definire una classe Model per ciascun file (xml TEI, xhtml, xhtml rash) caratterizzante il processo complessivo di trasformazione xslt e di inserirvi i metodi necessari alla trasformazione e manipolazione degli stessi. Inoltre, sarà necessario creare la classe Model referenziante il singolo workspace ed inserivi i metodi necessari, così che l'utente possa manipolarlo.

[61] https://docs.djangoproject.com/en/4.1/topics/auth/default/#permissions-and-authorization
[62] Con refactoring si intende la ristrutturazione del codice volta a renderlo più pulito e manutenibile senza modificarne il comportamento, tale attività permette di orientare la struttura allo sviluppo e all'integrazione di nuove feature, migliorandone le prestazioni.
[63] https://docs.djangoproject.com/en/4.1/topics/db/models/

- Il Template[64] (tei2rash/templates) è un file che definisce la struttura o il layout dell'interfaccia utente.

- La View[65] (tei2rash/views.py) è una funzione di gestione che assume in input una richiesta http e restituisce una risposta http. I dati necessari a soddisfare la richiesta vengono prelevati interrogando le classi del Model e vengono visualizzati all'utente mediante i Template. Per esempio, quando l'utente richiede i documenti contenuti all'interno di un workspace, la view dovrà: interrogare la classe Model del workspace recuperando i dati dei documenti in esso contenuti tramite i metodi della classe, e passare tali dati al template ritornato in risposta http così che possa visualizzarli all'utente richiedente.

[64] https://docs.djangoproject.com/en/4.1/topics/templates/
[65] https://docs.djangoproject.com/en/4.1/topics/http/views/

6 Bibliografia

[1] L. Burnard, «TEI: tei5_it,» 2004. [Online]. Available: https://tei-c.org/Vault/P4/Lite/teiu5_it.html. [Consultato il giorno 15 Febbraio 2023].

[2] A. D. Iorio, S. Peroni, F. Poggi, F. Vitali e P. Ciancarini, «Exploiting coordinated views for scholarly reading and analysis. In Proceedings-DMSVIVA 2019: 25th International DMS Conference on Visualization and Visual Languages (Vol. 2019, pp. 113-124). Knowledge Systems Institute Graduate School, KSI Research Inc..,» 2019. [Online]. [Consultato il giorno 15 Febbraio 2023].

[3] S. Peroni, F. Osborne, A. D. Iorio, A. G. Nuzzolese, F. Poggi, F. Vitali e E. Motta, «Research Articles in Simplified HTML: a Web-first format for HTML-based scholarly articles,» 2017. [Online]. Available: https://peerj.com/articles/cs-132/. [Consultato il giorno 16 Febbraio 2023].

[4] D. Buzzetti, «La Codifica XML/TEI,» 2011. [Online]. Available: http://web.dfc.unibo.it/buzzetti/corsoSFI/25febbraio/tomasi.pdf. [Consultato il giorno 15 Febbraio 2023].

[5] L. Ricci, «TEI - Text Ecnoding Initiative - Cliomatica - Digital History,» 2021. [Online]. Available: http://lhs.unb.br/cliomatica/index.php/TEI_-_Text_Encoding_Initiative. [Consultato il giorno 15 Febbraio 2023].

[6] S. Asperti, A. Quondam e B. Alfonzetti, «Biblioteca Italiana,» 2003. [Online]. Available: http://www.bibliotecaitaliana.it/. [Consultato il giorno 15 Febbraio 2023].

[7] T. Siegman e R. Berjon, «Scholarly HTML,» 2014. [Online]. Available: https://w3c.github.io/scholarly-html/. [Consultato il giorno 15 Febbraio 2023].

[8] S. Kleinfeld, «HTMLBook,» 2016. [Online]. Available: https://oreillymedia.github.io/HTMLBook/#:~:text=HTMLBook%20is%20an%20open%2C%20XHTML5,in%20digital%20or%20print%20form.. [Consultato il giorno 15 Febbraio 2023].

[9] T. Park, «GitHub-PUBcss,» 2017. [Online]. Available: https://github.com/thomaspark/pubcss/. [Consultato il giorno 15 Febbraio 2023].

[10] A. D. Iorio, A. G. Nuzzolese, F. Osborne, S. Peroni, F. Poggi, M. Smth, F. Vitali e J. Zhao, «The RASH Framework: enabling HTML + RDF submission in scholarly venues,» 2015. [Online]. Available:

https://iris.unimore.it/handle/11380/1199160?mode=complete. [Consultato il giorno 15 Febbraio 2023].

[11] G. Spinaci, S. Peroni, A. D. Iorio, F. Poggi e F. Vitali, «RASH Javascript Editor (RAJE),» 2017. [Online]. Available: https://essepuntato.it/papers/raje-doceng2017.html#section3. [Consultato il giorno 15 Febbraio 2023].

[12] K. Maiani, «Progettazione e sviluppo di un ambiente di lettura e confronto tra articoli scientifici,» 2021. [Online]. Available: https://amslaurea.unibo.it/23292/1/maiani_tesi%20.pdf. [Consultato il giorno 15 Febbraio 2023].

[13] S. Bonfanti, «DocuDipity: un ambiente per esplorare tendenze nella scrittura di articoli scientifici,» 2016. [Online]. Available: https://amslaurea.unibo.it/11956/. [Consultato il giorno 15 Febbraio 2023].

[14] W3School, «Introduzione XSLT,» 2001. [Online]. Available: https://www.w3schools.com/xml/xsl_intro.asp. [Consultato il giorno 15 Febbraio 2023].

[15] V. Fabio, «XSLT,» 2000. [Online]. Available: http://www.cs.unibo.it/~fabio/corsi/ium00/slides/20-XSLT/20-XSLT.pdf. [Consultato il giorno 16 Febbraio 2023].

[16] J. Hellingman, «tei2html,» 2009. [Online]. Available: https://github.com/jhellingman/tei2html. [Consultato il giorno 15 Gennaio 2023].

[17] Python, «subprocess --- Subprocess Management,» 2023. [Online]. Available: https://docs.python.org/3/library/subprocess.html. [Consultato il giorno 16 Febbraio 2023].

[18] TEIC, «TEIC Stylesheet,» 2009. [Online]. Available: https://github.com/TEIC/Stylesheets. [Consultato il giorno 22 Febbraio 2023].

[19] J. Walsh, G. Simpson e S. Moaddeli, «Tei-Boilerplate,» 2011. [Online]. Available: https://github.com/TEI-Boilerplate/TEI-Boilerplate. [Consultato il giorno 22 Febbraio 2023].

[20] D. Giacomini, «a2 --- Introduzione a TEI,» 2007. [Online]. Available: https://wwwcdf.pd.infn.it/AppuntiLinux/introduzione_a_tei.htm. [Consultato il giorno 15 Febbraio 2023].