Scuola universitaria professionale della Svizzera italiana
Dipartimento tecnologie innovative

SUPSI

DocTT — Document Tagging Tool

Studente/i

Cristian Spozio

Denys Vitali

Relatore

Daniele Puccinelli

Correlatore

Salvatore Vanini

Corso di laurea

Ingegeneria Informatica

Modulo

M02042 - Progetto di semestre

Anno

2019

Data

14 maggio 2019

Indice

Elenco delle figure

Abstract

DocTT (Document Tagging Tool) is a tagging tool (corpus annotation tool) which allows its users to tag text documents such as press conferences, conference calls or pitches. *DocTT* satisfies the need to switch from outdated tools such as *UAM Corpus Tool* in order to provide an easy, open source, multi-platform, lightweight and customizable tool to replace these outdated tools.
The tags take form in colored rectangles surrounding the text and allowing a fast and easy visual recognition of a tag. Tags are defined through a pre-uploaded tree by the user. The files involved (document files and tree files) follow the XML standard and their parsing is managed by the tool.

Riassunto

DocTT è uno strumento di tagging di corpus testuale che utilizzato per suddividere parti di testo ed assegnare delle etichette a parti di discorsi. Il suo impiego principale è il tagging di conference calls, pitches o documenti testuali in generale.
DocTT soddisfa il bisogno di cambiamento motivato dalla relativamente scarsa user-friendliness di tool come *UAM Corpus Tool* e fornisce un'alternativa open source multi-piattaforma, leggera e personalizzabile per rimpiazzare questi strumenti.

I tag sono rappresentati da rettangoli smussati e colorati che circondano il testo, in modo da fornire un veloce riconoscimento di parti del discorso. I tag sono definiti mediante un albero caricato precedentemente dall'utente. I file utilizzati (file di documento e dell'albero) seguono lo standard XML e le convenzioni di *UAM Corpus Tool*. La loro interpretazione è gestita dal nostro strumento.

Capitolo 1

Introduzione

1.1 Struttura

DocTT si divide in 4 sezioni ognuna adibita ad una funzione specifica:

- Home
- Document Upload
- Tree View
- Tree Upload

1.1.1 Home

La **Home** di *DocTT* è la sezione in cui viene presentata la lista dei documenti caricati, dalla quale è possibile accedere ai documenti singoli e quindi alla loro modifica.

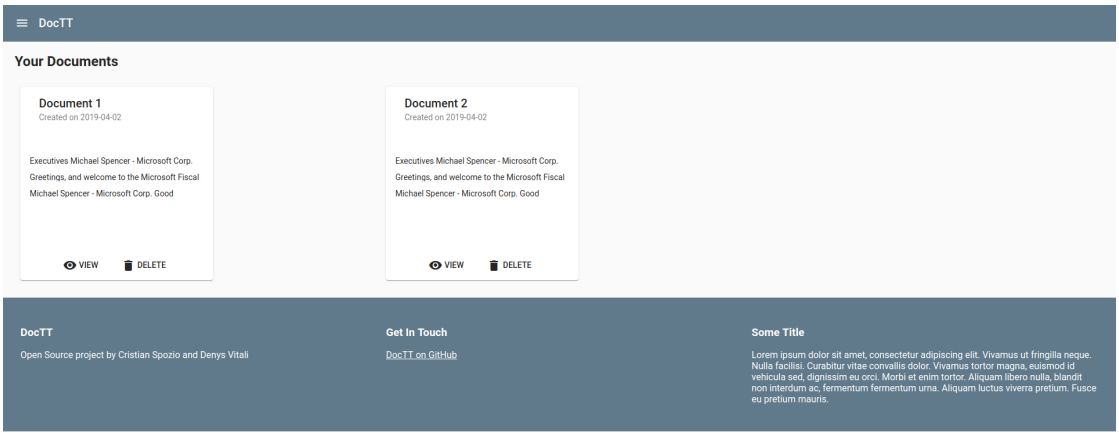


Figura 1.1: Home section

1.1.2 Document Upload

La sezione **Document Upload** di *DocTT* è quella che permette l'upload dei documenti da taggare ed il loro salvataggio nello storage del browser.

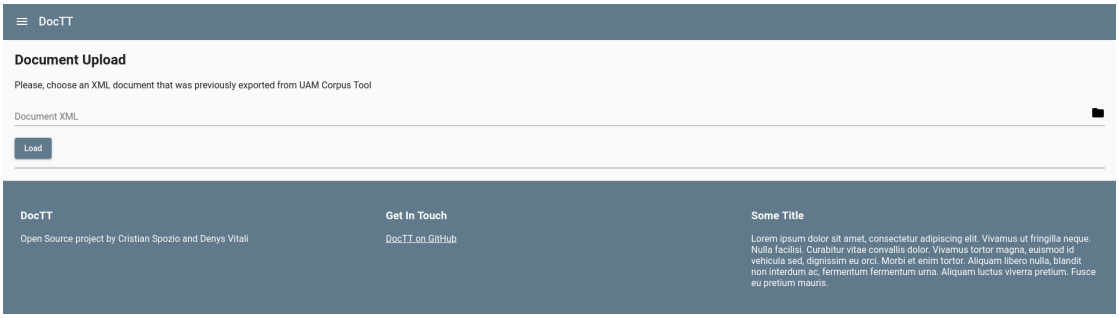


Figura 1.2: Document Upload section

1.1.3 Tree View

La sezione **Tree View** di *DocTT* è la parte che mostra l'albero di tagging attualmente in uso come un elenco puntato espandibile per ogni sottosezione presente.

Figura 1.3: Tree View section

1.1.4 Tree Upload

La sezione **Tree Upload** di *DocTT* è il componente che permette l'upload dell'albero di tagging. Offre inoltre un'anteprima dell'albero attualmente in uso (nel caso ci fosse), secondo la stessa visualizzazione della sezione **Tree View**.

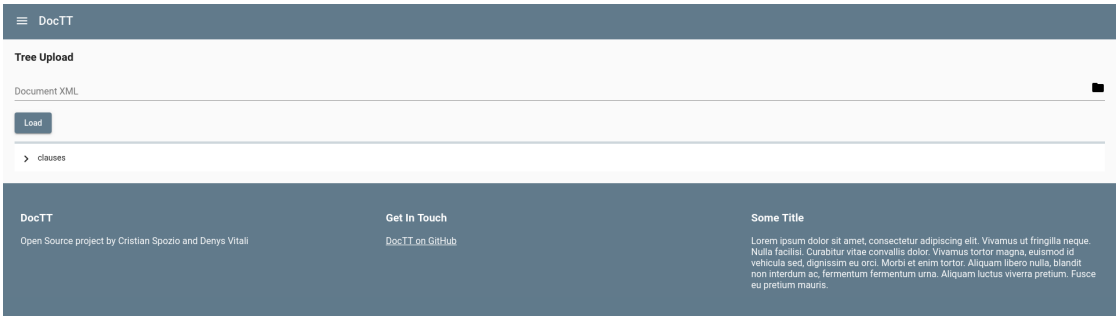


Figura 1.4: Tree Upload section

1.2 Tecnologie

DocTT è stato sviluppato in **Angular**[1] secondo le direttive fornite dal relatore. È quindi stata divisa *app* nelle sottosezioni directives, models, modules, services e shared. I *componenti Angular* sono stati quindi divisi in 3 parti: struttura, stile e funzione rispettivamente secondo i linguaggi **HTML5**, **SCSS** e **TypeScript**.

[1] Angular: Framework JavaScript per la creazione di applicazioni web

Capitolo 2

Funzionamento

2.1 Processo di tagging

Il processo di tagging può venire effettuato solo dopo il caricamento di un albero di tagging e dopo la selezione di un documento, operazioni eseguibili nelle apposite sezioni del tool.

2.1.1 Floating menu

La procedura effettiva di tagging viene effettuata selezionando del testo, utilizzando il cursore: al completamento di questa operazione apparirà un menu alla fine della selezione, contenente l'albero di tagging dal quale l'utente potrà scegliere quale tag attribuire alla selezione.

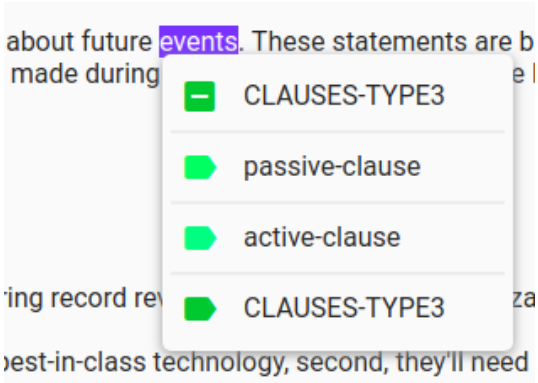


Figura 2.1: Floating Tagging Menu

2.1.2 Nested tags

Il tagging può essere effettuato a più livelli, quindi un tag (o più) possono essere contenuti in un altro tag (o più).

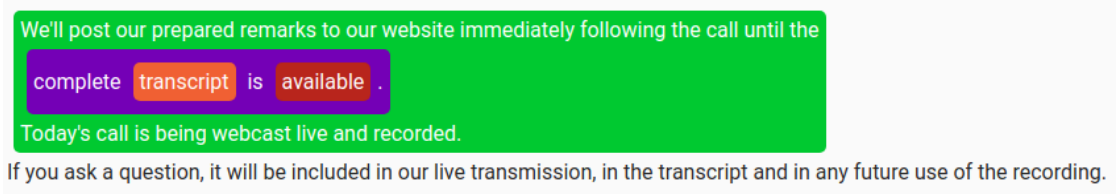


Figura 2.2: Nested Tags Example

2.1.3 Tag View

Una volta taggata una parte di testo, è possibile vedere quale tag è applicato in quella sezione semplicemente portando il cursore sopra il suddetto tag.

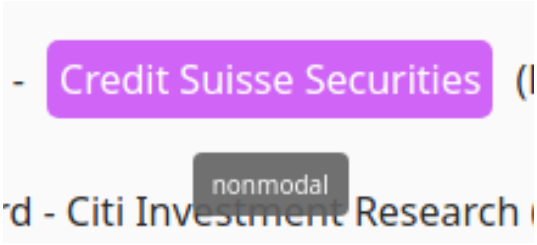
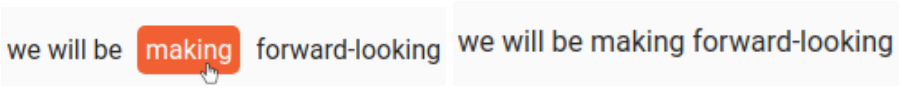


Figura 2.3: Tag View

2.1.4 Rimozione di tag

I tag sono facilmente rimovibili semplicemente cliccando su di essi.



(a) Before the click (b) After the click

Figura 2.4: Tag Removal Example

Capitolo 3

Specifiche tecniche

Qui sotto verranno elencate le specifiche tecniche utilizzate durante lo sviluppo del tool.

3.1 JavaScript/TypeScript Enviroment

Durante lo sviluppo del tool il team si è trovato a confronto con l'ambiente di sviluppo di *JavaScript/TypeScript* e quindi con le rispettive tecnologie, qui elencate le più significative.

3.1.1 Angular

Come già citato in precedenza, **Angular** è un *web application framework* basato su *TypeScript*. Il concetto principale dietro *Angular* è quello di definire una struttura di base ed una gerarchia di componenti, che verranno poi inseriti all'interno del componente principale *app* come il riempimento di una cornice vuota.

3.1.1.1 App structure

Essendo *app* il componente principale la sua struttura è fondamentale. Noi abbiamo optato per suddividere il tutto, secondo convenzione di *Angular*, in 5 parti:

Directives Le *direttive Angular* sono strumenti che una volta definiti cambiano aspetto e/o comportamento di un elemento del DOM[1].

Models I *modelli Angular* sono templates che definiscono la struttura di classi o interfacce.

Components I *componenti Angular* sono l'elemento base per la costruzione di una UI[2] e sottostanno sempre ad una direttiva e sono spesso costruiti con dei templates. Sono definiti da tre segmenti: struttura, stile e funzione.

Services I *servizi Angular* sono dei componenti senza struttura né stile, utilizzati in tutto il complesso per le loro funzioni e le operazioni che offrono. Possono essere visti come della classi statiche di utilità in **Java**.

[1] DOM (Document Object Model): È una forma di rappresentazione di una pagina web come albero gerarchico

[2] UI: User Interface (Interfaccia Utente)

Modules I *moduli Angular* sono una raccolta di componenti, servizi, direttive, controller, filtri e informazioni sulla configurazione. Possono essere visti come le frazioni principali di una *app*.

3.1.1.2 Routing

Il routing è una parte fondamentale di una qualunque applicazione front-end che si possa definire tale. Grazie ad esso è possibile manovrare un'applicazione in esecuzione attraverso la richiesta di specifici path nell'URL. Questo permette di spostarsi attraverso l'applicazione senza continuare a passare da una pagina web all'altra, ma semplicemente andando a cambiare i contenuti della pagina attuale secondo le risorse richieste. Ciò comporta meno richieste al server ed un più rapido processo di caricamento e *switching* tra pagine/componenti.

3.1.2 npm

npm o *Node.js package manager* è un package manager per *JavaScript*. Esso consiste in un client da CLI [3] che permette all'utente di utilizzare e/o distribuire moduli *JavaScript*.

3.1.3 Webpack

Webpack è un *module bundler JavaScript* open-source. Esso permette di definire *loaders, plugins*, ecc., utilizzato per un approccio modulare alle *web applications*.

3.2 Compilazione e Test

3.2.1 Compilazione

3.2.1.1 npm

npm offre anche la possibilità di definire degli *scripts* utilizzabili nel suo client per utilizzi vari, nel nostro caso la compilazione. Difatti il nostro progetto è compilabile tramite l'utilizzo del comando **npm run build:dev** sul client *npm*.

3.2.2 Test

3.2.2.1 Jest

Jest è un framework di Testing *JavaScript* basato sulla semplicità di configurazione ed utilizzo. Permette di definire delle *test suites*, delle collezioni di test che vadano a verificare il corretto funzionamento e comportamento delle funzionalità del tool.

[3] CLI (Command Line Interface): console

3.2.2.2 npm

Anche qui *npm* si rende utile, potendo appunto definire uno *script* di testing che lancerà le varie *test suites* per il controllo tramite il comando **npm test**.

Capitolo 4

Problemi Riscontrati

4.1 Nuovo ambiente

Un grosso problema è stato il 'nuovo ambiente' con cui ci si è dovuti confrontare quale il mondo di *Angular*. Essendo un requisito imposto abbiamo dovuto adattarci ed imparare il framework nel minor tempo possibile per essere subito operativi e concentrarci sulla funzionalità del tool e le features richieste.

4.2 Comprensione differente

All'inizio del progetto ci sono state piccole incomprensioni che hanno portato ad un'errata implementazione, come anche lo scoprire a progetto già inoltrato che l'albero di tagging avesse delle condizioni di tagging come restrizioni di tipo simil-logico sul concetto di *either-or* o *if-then*.

4.3 Mancanza di specifiche

Essendo il tool UAM Corpus Tool (tool di riferimento dal quale dovevamo effettuare l'integrazione) mal documentato e closed-source, abbiamo avuto notevoli difficoltà nel capire certi tipi di scelte effettuate dagli sviluppatori del tool. Non essendo disponibile una specifica, il nostro tentativo è una specie di reverse-engineering basato sull' analisi dei file esportati mediante UAM Corpus Tool. La compatibilità completa non è quindi garantita, specie con versioni precedenti alla 3.3.

Capitolo 5

Piani futuri

5.1 Miglioramenti e Bug fix

5.1.1 Errore newline

Se nella selezione compare una *newline* a se stante la selezione riscontra problemi perché il carattere non viene rilevato come testo e quindi non mostra il **floating menu**.

5.1.2 Tag removal

Se nel documento sono presenti per parsing errato o malformazione del documento delle sezioni non etichettate dalla classe *span-container* il componente non viene rilevato e quindi non viene rimosso il tag.

5.2 Features

5.2.1 Tagging completo

Come accennato nel capitolo precedente la scoperta delle restrizioni simil-logiche hanno portato problemi durante lo sviluppo, uno dei piani futuri è appunto quello di implementare correttamente questa feature.

5.2.2 Salvataggio su localStorage ed esportazione in XML

Sfortunatamente per una questione di tempo non ci è stato possibile implementare una funzionalità di salvataggio dei documenti una volta taggati, conseguentemente i cambiamenti al cambio della pagina non vengono salvati.

In uno sviluppo futuro è senz'altro necessario implementare un salvataggio dei documenti almeno nel localStorage, dopodiché inserire un layer di esportazione in XML (compatibile con UAM Corpus Tool) ed in un formato future-proof e meglio organizzato (come per esempio JSON / Protobuf).

5.2.3 Interfaccia con IA

Durante lo sviluppo abbiamo avuto l'occasione di relazionarci con un masterando partecipante al progetto che stava lavorando su una rete neurale capace di effettuare il tagging in automatico dei documenti. Un'aggiunta possibile sarebbe quella di interfacciare il tool con il lavoro del masterando e quindi al momento dell'upload di un documento effettuare questo tagging automatico iniziale sul quale poi l'utente andrà a lavorare.