ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CORSO DI LAUREA IN

INFORMATICA PER IL MANAGEMENT

PROGETTAZIONE E SVILUPPO DI UN SISTEMA PER ANNOTARE TESTI INTEGRABILE IN APPLICAZIONI TYPESCRIPT

Relatore

Prof. Angelo Di Iorio

Presentata da

Marco Dal Pian

Sessione dicembre 2024

Anno Accademico 2023/2024

Yes, truth is a thing. It's the truth of who we are and what we do.
And you have to face that, and accept it.

Harry Dean Stanton

Indice

Introduzione

L’obiettivo della tesi è realizzare un sistema che permetta di aggiungere annotazioni ai testi, consultare le annotazioni degli altri utenti e aggiungerci commenti. Esso deve essere pronto ad essere facilmente integrato con il framework TypeScript di DocuDipity [19], un sistema che permette all’utente di consultare i testi attraverso varie visualizzazioni, ma che non permette nessuna aggiunta di contenuti o la possibilità di commentare documenti. Il progetto nasce con l’obiettivo di superare questa limitazione. Il sistema in questione, chiamato Text Annotator, è un'applicazione web che permette di visualizzare il funzionamento dello stesso, senza che debba prima essere integrato in un'applicazione terza. Nella sezione 6 viene spiegato come integrare Text Annotator in DocuDipity, o in una qualsiasi applicazione con framework TypeScript. Grazie a Text Annotator, DocuDipity disporrà di un nuovo sistema che permette di interagire coi testi. Gli utenti avranno la possibilità di aggiungere il loro contributo e la loro opinione riguardo ai testi presenti nel sistema, e riguardo ai commenti pubblicati da altri utenti. Gli studiosi iscritti al sito disporranno quindi di un meccanismo che gli permette di confrontarsi, e che è direttamente collegato alla visualizzazione testuale degli articoli scientifici presenti nell’applicazione.

Contesto: DocuDipity

Esistono diversi modi per leggere articoli scientifici e, in generale, documenti. La lettura sequenziale del flusso di testo può essere completata con visualizzazioni alternative come viste ad albero, mappe e grafici. In alcuni casi gli studiosi devono esaminare attentamente i documenti, esaminando attentamente ogni singola parola e carattere; in altri, è sufficiente una visuale a volo d'uccello: sfogliano le pagine, cercando parti rilevanti, oppure saltano a sezioni specifiche, partendo dall'indice. L'idea di base di DocuDipity è quella di migliorare l'approccio overview+detail combinando visualizzazioni coordinate in modo da aiutare i ricercatori a scoprire tratti degli articoli scientifici, senza necessariamente leggerne l'intero contenuto. DocuDipity combina il layout di flusso ipertestuale dell'articolo, in cui i lettori possono scorrere il contenuto sequenziale del documento e leggerlo, con un nuovo modo di visualizzare

documenti basati su alberi, in particolare quelli XML, basato sulla visualizzazione SunBurst. SunBurst è una forma di visualizzazione radiale progettata per rappresentare e riassumere in una forma molto compatta anche gerarchie di informazioni complesse. Vale la pena notare che DocuDipity è uno strumento generale che può funzionare su qualsiasi documento XML, senza richiedere alcuna informazione di base sul formato in cui sono scritti i documenti. Infatti, il motore DocuDipity si basa sulla teoria dei pattern strutturali. Sfruttando l'algoritmo di identificazione dei pattern, DocuDipity è in grado di estrarre informazioni rilevanti sulla struttura del set di documenti forniti come input e di utilizzarle per visualizzare i documenti e per fornire un framework di analisi per ispezionarne facilmente il contenuto. L'applicazione web DocuDipity è stata inizialmente implementata dal DASPLab (Di- gital and Semantic Publishing Lab), un gruppo di ricerca dell'Università di Bologna, successivamente sono state aggiunte altre funzionalità grazie al contributo di tre tesi di laurea. Vediamo quindi un esempio, Figura 1, riguardante la funzionalità principale di Docudipity, l'interazione tra due diversi tipi di visualizzazione:

- Sulla sinistra viene mostrata la *ReadingView*, è la visualizzazione testuale base di un articolo scientifico.
- Sulla destra viene mostrata *Document Components*, è la forma di visualizzazione radiale che rappresenta e riassume in una forma compatta le gerarchie dell'articolo scientifico.

Cliccando su una sezione di *Document Components* viene fatta scorrere la *ReadingView*, e viene evidenziata la relativa porzione di testo. Allo stesso modo, passando il puntatore su una porzione di testo, viene evidenziata la relativa sezione di *Document Components*. In questo caso il paragrafo selezionato corrisponde alla fetta blu della visualizzazione a destra, che gerarchicamente è figlia delle due fette viola più interne.

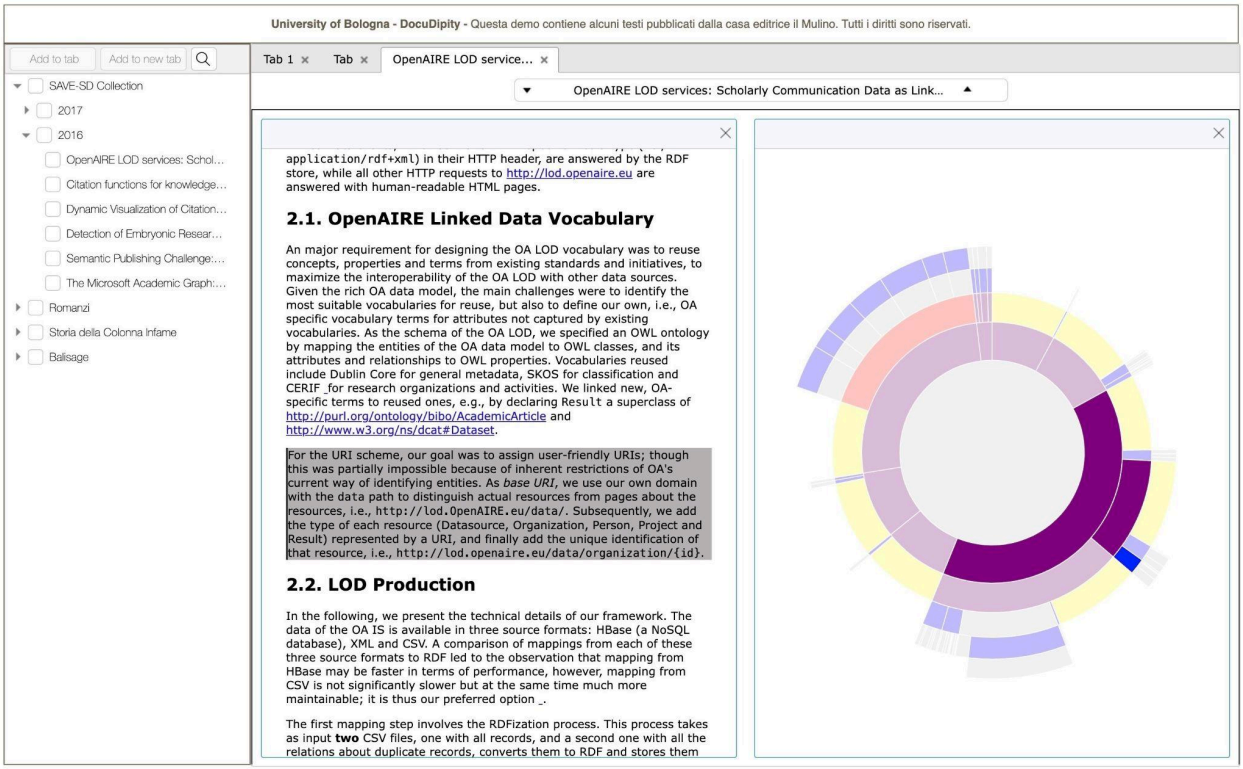

Figura 1: Esempio di una interazione tra due visualizzazioni di testo dell'applicazione DocuDipity.

Capitolo 1

Librerie di annotazione

Sistemi come Annotate the Web [13], Hypothesis [2] o Ziflow [29] permettono di aggiungere annotazioni ad una pagina web, ma non sono ciò di cui necessita Docudipity. Il sistema di cui ha bisogno deve essere su misura, in modo da avere il pieno controllo sul funzionamento dello stesso, e che si allinei alle sue necessità. Per realizzare un'applicazione di questo tipo, che offra la possibilità di selezionare porzioni di testo alle quali aggiungere annotazioni, mi sono immerso nella ricerca di una libreria compatibile che mi aiutasse nel processo di sviluppo. Ho trovato varie librerie relative all'annotazione di immagini e grafici, ma non erano ovviamente pertinenti al sistema che volevo implementare. Infine ho trovato tre librerie possibilmente adatte, che vengono spiegate nelle Sezioni 1.1, 1.2 e 1.3.

1.1 react-text-annotation

react-text-annotation [16] è un componente React per annotazione testo scritto in linguaggio TypeScript, inoltre supporta implementazioni basate su linguaggio JavaScript. È stato progettato per essere personalizzato indipendentemente dal framework di stile utilizzato, con l'obiettivo di assicurare affidabilità per utilizzo di produzione e compatibilità cross-browser, facile da usare e che copre molti casi d'uso. Nella Figura 2 viene mostrato un esempio di testo annotato utilizzando questa libreria.

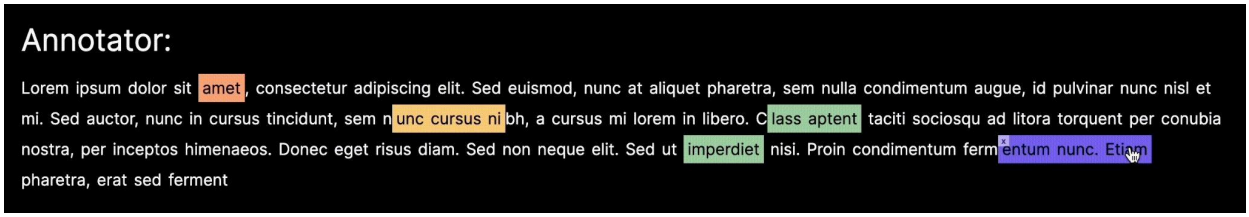

Figura 2: Output del componente *TextAnnotator* di react-text-annotation.

Utilizzo della libreria

Nella Figura 3 viene mostrato un esempio di renderizzazione del componente *TextAnnotator* di react-text-annotation.

```
<TextAnnotator
  value={annotations}
  onChange={handleAnnotate}
  content={text}
  category={selectedCategory}
/>
```

Figura 3: Componente *TextAnnotator* di react-text-annotation.

Le proprietà che definisce sono le seguenti:

- value, contiene la lista degli oggetti *Annotation* presenti sul testo.
- onChange, contiene la funzione che viene lanciata quando l'utente seleziona una porzione di testo.
- content, contiene il testo che deve essere annotato.
- category, contiene l'oggetto *Category* attualmente selezionato che verrà usato per annotare il testo.

Proprietà dell'oggetto *Annotation*:

- start, numero della posizione del cursore all'inizio della selezione del testo.
- end, numero della posizione del cursore alla fine della selezione del testo.
- text, porzione di testo inclusa nella selezione.
- category, oggetto *Category* selezionato per l'annotazione.

Proprietà dell'oggetto *Category*:

- id, numero identificativo dell'oggetto *Category*.
- color, colore in formato HEX con il quale vengono colorate le relative note.

Inizialmente questa libreria sembrava perfetta per l'uso che volevo farne, e sarebbe stato semplice integrarla. Il componente che mette a disposizione prende in input un parametro di tipo string, lo renderizza, e permette di selezionarne una porzione per colorarla di uno specifico

colore. Avrei poi dovuto implementare un sistema di aggiunta annotazioni alle relative porzioni di testo. Purtroppo non si poteva fornirle in input un file HTML, indipendentemente dal modo o formato, quindi ho deciso di scartarla.

1.2 annotate

annotate [21] è una libreria per annotare e pubblicare testo sul web, che può essere usata per annotare qualsiasi documento di testo. È stata sviluppata utilizzando HTML, css e JavaScript. Nella Figura 4 viene mostrato un esempio di documento annotato utilizzando questa libreria.

Figura 4: Output di un documento annotato utilizzando la libreria annotate.

Utilizzo della libreria

Copiare i file *index.html*, *styles.css*, e *annotate.js* messi a disposizione dalla libreria nel progetto. L'unico file da modificare è *index.html*, a meno che non si voglia modificare lo stile o il comportamento JavaScript. Nella Figura 5 viene mostrata la struttura di base che ogni sezione del documento deve seguire.

```
<section class="group">
  <div class="content quote">
    Text that's being <mark data-annotation-id="1" aria-details="unique-comment-id">annotated</mark>.
  </div>
  <div class="content note">
    <div class="annotation" role="comment" data-annotation-id="1" id="unique-comment-id">
      <div class="commenter">Commenter name</div>
      Comment text.
    </div>
  </div>
</section>
```

Figura 5: Struttura di un documento annotato utilizzando la libreria annotate.

Ogni sezione di testo si trova all'interno di una riga con una sezione di sinistra e una di destra. L'elemento <*section class="group"*> rappresenta questa riga. Ogni sezione contiene un <*div*> con la classe *content quote* o *content note*. *quote* corrisponde al testo annotato e *note* corrisponde all'annotazione. Ogni porzione di testo annotato è racchiusa dentro un <*mark*>, che contiene l'id della relativa annotazione.

La tecnica di utilizzo di questa libreria non era perfettamente compatibile con il progetto che volevo realizzare. Avrei dovuto implementare un event handler per catturare le porzioni di testo selezionate, per poi aggiungere le annotazioni direttamente sui documenti HTML. Lo sviluppo dell'applicazione usando questa libreria sarebbe stato complicato, così ho deciso di scartarla.

1.3 recogito-js

RecogitoJS [23] è una libreria JavaScript per annotazioni di testo. Può essere usata per aggiungere funzionalità ad una pagina web, o come strumento per sviluppare un annotation app completamente personalizzata. Nella figura 6 viene mostrato un esempio di testo annotato utilizzando questa libreria.

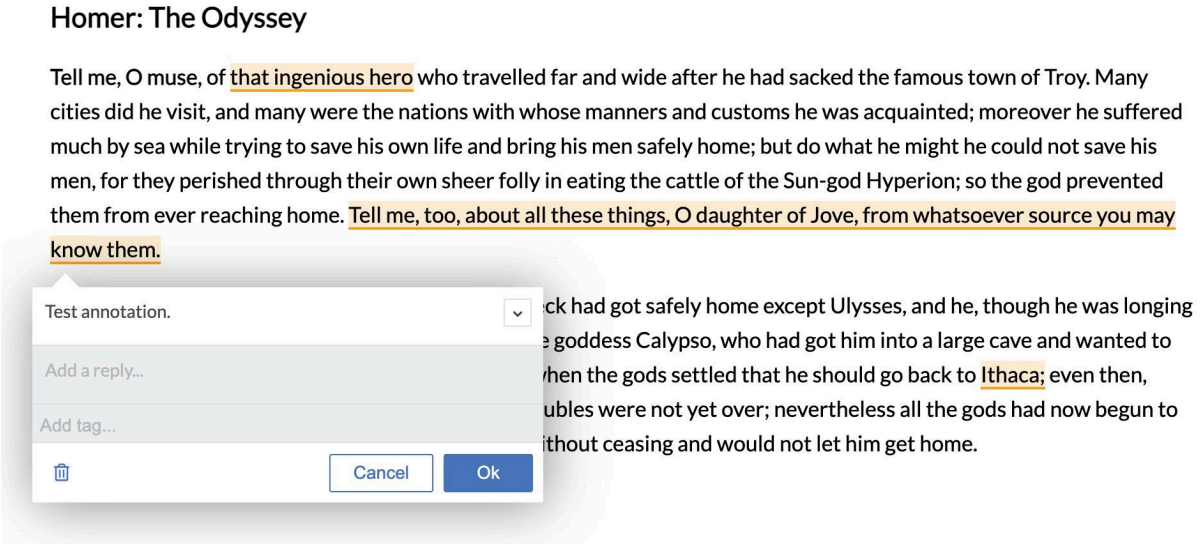

Figura 6: Testo annotato utilizzando la libreria recogito-js

Utilizzo della libreria

Nella Figura 7 viene mostrato come inizializzare un'istanza RecogitoJS su un elemento della pagina.

```
1  const r = new Recogito({ content: 'my-content' });
```

Figura 7: Inizializzazione di un'istanza RecogitoJS

my-content corrisponde all'id del contenitore dell'elemento che si vuole annotare. Il passo successivo consiste nel definire il comportamento degli event handler listati di seguito (nella Figura 8 viene mostrato un esempio):

- createAnnotation
- updateAnnotation
- deleteAnnotation
- selectAnnotation

```
1  r.on('selectAnnotation', function(annotation, element) {
2    console.log(annotation, element);
3  });
```

Figura 8: handler di un evento *selectAnnotation* della libreria recogito-js.

Le varie funzioni e event handler messi a disposizione da RecogitoJS, quindi la possibilità di modificare il comportamento della libreria in modo semplice, mi hanno portato a scegliere questa libreria per realizzare Text Annotator.

Limitazioni

Ho incontrato tre limitazioni durante l'utilizzo di questa libreria:

- Mostrare il nome dell'autore nella finestra di creazione/modifica/eliminazione nota.
- Assegnare diversi colori alle porzioni di testo che contengono annotazioni.
- Selezionare una porzione di testo che finisce con un "a capo".
- Selezionare un'immagine che non sia preceduta e succeduta da testo.

Il primo limite l'ho risolto aggiungendo all'inizio del corpo delle annotazioni il nome dell'autore, nascondendolo dalle visualizzazioni che non lo richiedevano, e rendendo impossibile la modifica dell'autore in caso di modifica della nota. Il secondo limite l'ho risolto aggiungendo agli oggetti Note un parametro *color*, che al momento di renderizzazione delle annotazioni viene letto per poi essere assegnato agli HTMLElements corrispondenti alle note presenti sul testo. Gli ultimi due limiti non erano risolvibili, ma comunque non erano abbastanza gravi da dover scartare la libreria, quindi ho deciso di allertare l'utente di questi vincoli attraverso un paragrafo di avvertimento posto in capo ai testi annotabili.

Capitolo 2

Terminologia e analisi dei requisiti

DocuDipity offre una collezione chiusa di documenti, per ognuno di essi sarà possibile aggiungere annotazioni e commenti grazie a Text Annotator. Il sistema prende in input le informazioni relative al testo da annotare, e permette all'utente di creare le proprie note all'interno di un contenitore chiamato sessione. Per ogni documento presente in DocuDipity esiste una relativa sessione per ogni utilizzatore. Essa è inizialmente privata, così che l'utente possa modificare le proprie annotazioni prima di renderle pubbliche. Quando ritiene che le proprie note siano finalizzate, deve salvare e chiudere la sessione corrente. Di conseguenza diventa pubblica, dando la possibilità agli altri utenti di visionare le annotazioni che contiene, e aggiungerci commenti. Di seguito la terminologia inerente a Text Annotator:

- Annotazione/nota: rappresenta un commento riguardo una porzione di testo. Modello dati nella Tabella 1.

Dato	Descrizione
ID	Stringa identificativa univoca relativa alla nota.
type	= "draft" o "final", nota privata o pubblica.
value	Corpo della nota.
userID	Stringa identificativa univoca relativa all'autore della nota.
username	Nome utente dell'autore della nota.

bookID	Stringa identificativa univoca del testo relativo all'annotazione.
session	Sessione che contiene la nota.
exact	Porzione di testo relativa alla nota.
start	Numero della posizione del cursore all'inizio della selezione del testo relativa alla nota.
end	Numero della posizione del cursore alla fine della selezione del testo relativa alla nota.
isComment	= false, definisce la nota come nota.

Tabella 1: Modello dati Annotazione.

- Commento: rappresenta una considerazione, giudizio, opinione o parere riguardo un'annotazione. Modello dati nella Tabella 2.

Dato	Descrizione
ID	Stringa identificativa univoca relativa al commento.
type	= "draft" o "final", commento privato o pubblico.
value	Corpo del commento.
userID	Stringa identificativa univoca relativa all'autore del commento.

username	Nome utente dell'autore del commento.
bookID	Stringa identificativa univoca del testo relativo al commento.
session	Sessione che contiene il commento.
exact	Porzione di testo relativa al commento.
start	Numero della posizione del cursore all'inizio della selezione del testo relativa al commento.
end	Numero della posizione del cursore alla fine della selezione del testo relativa al commento.
isComment	= true, definisce la nota come commento.

Tabella 2: Modello dati Commento.

- Sessione: rappresenta un ambiente all'interno del quale un utente può creare annotazioni che sono visibili solo a lui. In questo modo l'utente può creare un set di note, per poi in un secondo momento aggiungerne, modificarle o eliminarle, e successivamente, quando sono ultimate, pubblicarle. Modello dati nella Tabella 3.

Dato	Descrizione
name	Nome univoco della sessione.
description	Descrizione della sessione.

color	Colore assegnato alle note contenute nella sessione.
date	Data di creazione della sessione, quando la sessione viene chiusa si aggiorna in data di pubblicazione.
type	= “draft” o “final”, sessione privata o pubblica.

Tabella 3: Modello dati Sessione.

- Testo/articolo/libro: rappresenta un documento HTML relativo ad un articolo scientifico presente nel sistema. Modello dati nella Tabella 4.

Dato	Descrizione
ID	Stringa identificativa univoca del testo.
title	Titolo del testo.
content	Codice HTML del testo.

Tabella 4: Modello dati Testo.

Di seguito i requisiti funzionali dell’applicazione, modellati in base agli obiettivi definiti nell’Introduzione e raggruppati in base alla terminologia:

- Testo:
    - Annotazione testo: Ogni testo presente nel sistema può essere annotato.
    - Evidenziazione testo: Le porzioni di testo che contengono un’annotazione sono evidenziate.
    - View testi: Ogni testo nel sistema presenta una view che mostra il corpo dello stesso, dalla quale è possibile creare annotazioni.

- Annotazione:
    - View annotazioni: Ogni testo nel sistema presenta una view contenente tutte le annotazioni con i relativi commenti.
    - Presentazione annotazioni: Le annotazioni e relativi commenti vengono mostrati attraverso una finestra pop-up quando si clicca su una porzione di testo evidenziata.
    - Creazione annotazioni: L'utente può selezionare una porzione di testo e aggiungere un'annotazione ad essa.
    - Modifica annotazioni: L'utente può selezionare una propria annotazione e modificarne il contenuto.
    - Eliminazione annotazioni: L'utente può selezionare una propria annotazione ed eliminarla dal sistema.
- Commento:
    - Aggiunta commenti: L'utente può aggiungere commenti alle proprie annotazioni e alle annotazioni di altri utenti.
    - Modifica commenti: L'utente può selezionare un proprio commento e modificarne il contenuto.
    - Eliminazione commenti: L'utente può selezionare un proprio commento ed eliminarlo dal sistema.
- Sessione:
    - Creazione sessione: Quando l'utente crea una nuova nota, essa viene salvata in una nuova sessione (o aggiunta alla sessione già creata) relativa al testo che si sta annotando. Le note contenute nella sessione sono visualizzabili soltanto dall'autore delle stesse.
    - Salvataggio sessione: L'utente può salvare e chiudere la propria sessione dopo aver inserito le relative informazioni obbligatorie: nome e colore. Le note contenute nella sessione saranno visualizzabili da tutti gli utenti iscritti al sistema, ed una nuova sessione sarà creata nel momento di creazione di una nuova annotazione.
    - Eliminazione sessione: L'utente può eliminare la propria sessione, le note contenute nella stessa vengono cancellate.

- Requisito aggiuntivo:
    - Filtro: Nella view annotazioni sono presenti due menu utili al filtraggio delle note pubblicate: filtro autore e filtro sessione.

Capitolo 3

Text Annotator: interfaccia e funzionalità

Text Annotator presenta una schermata, Figura 9, che è divisa in due sezioni:

- Sulla sinistra si trova un riepilogo di tutte le informazioni relative alle note create per il testo selezionato.
- Sulla destra vi è una sezione che contiene:
    - Il titolo del testo selezionato.
    - Informazioni e istruzioni che spiegano all'utente cosa può e non può fare.
    - Il corpo del testo in questione.

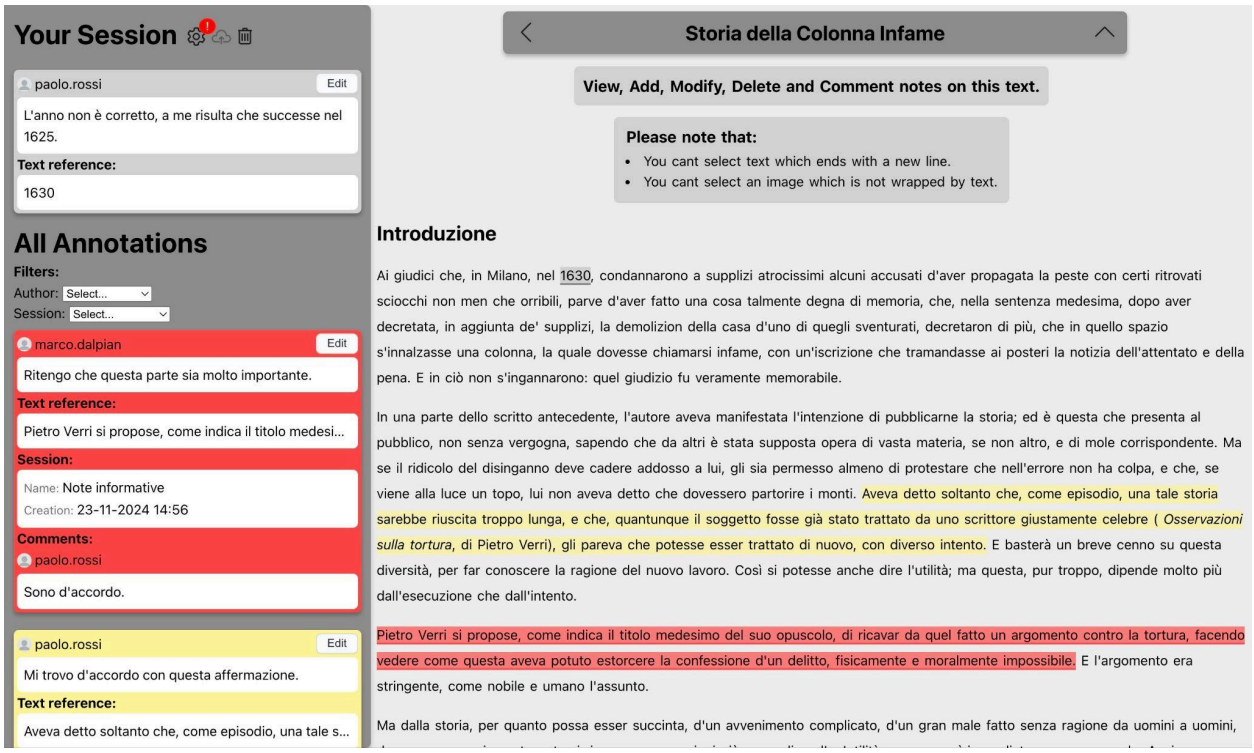

Figura 9: Schermata di Text Annotator.

3.1 Creazione annotazione e sessione

Selezionando una porzione di testo, viene mostrata una finestra, Figura 10, che permette di inserire un'annotazione.

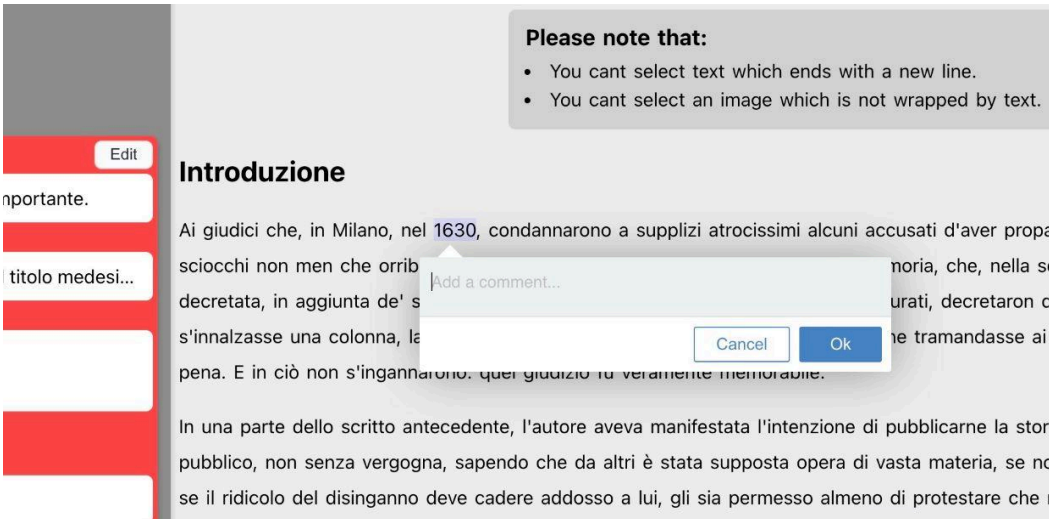

Figura 10: Finestra di annotazione di Text Annotator.

Dopo aver confermato, l'annotazione viene salvata tra le note visibili solamente dall'utente che le ha create, e quindi consultabile nella sezione *Your session*, mostrata in Figura 11.

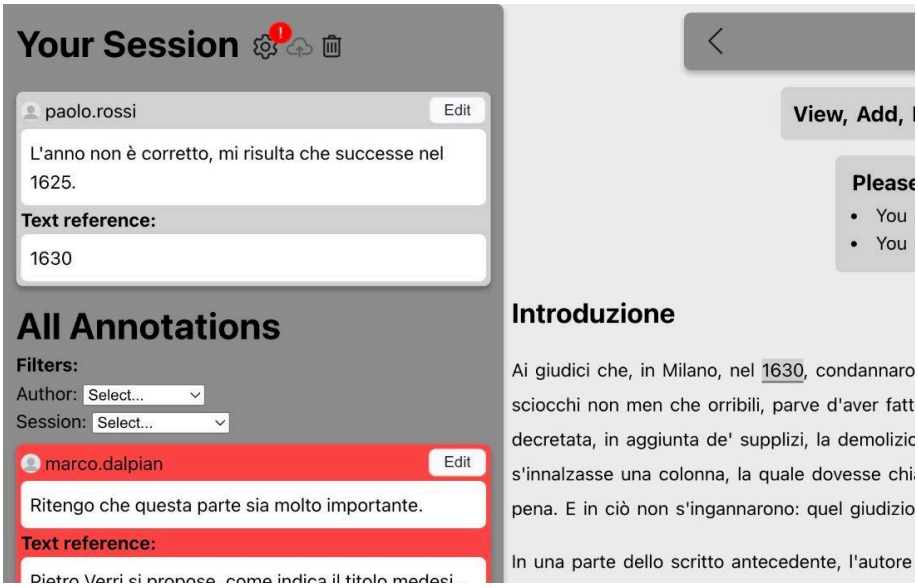

Figura 11: Sezione *Your Session* di Text Annotator.

3.2 Salvataggio annotazione e sessione

Per confermare le note che ha creato, quindi renderle disponibili a tutti gli utenti iscritti al sistema, l'utente deve cliccare sull'icona a nuvoletta che si trova sopra alle note nella sua sessione. Se l'utente non ha ancora inserito le informazioni obbligatorie relative alla sessione, non sarà in grado di pubblicare la stessa, il bottone di salvataggio sarà disabilitato e verrà mostrato un avviso in corrispondenza dell'icona ad ingranaggio. L'utente deve quindi cliccare tale bottone, ed inserire le informazioni richieste contrassegnate da un avviso nella Figura 12.

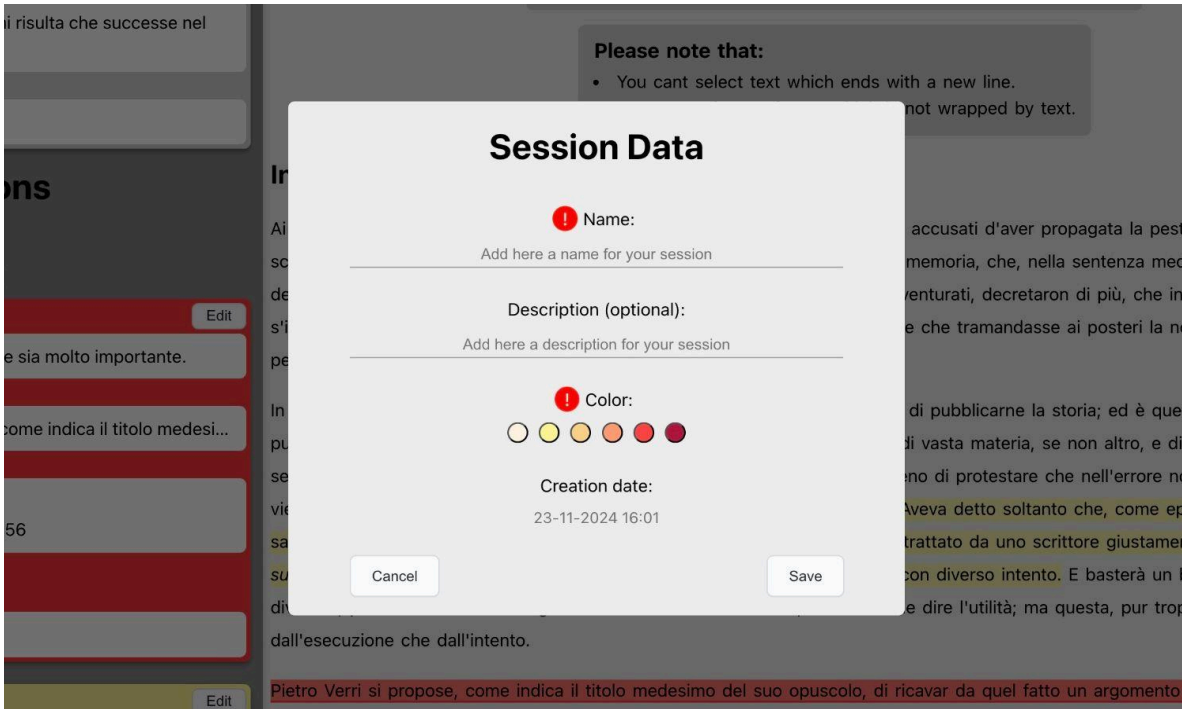

Figura 12: Schermata per l'inserimento delle informazioni relative alla sessione di Text Annotator.

Se l'utente seleziona un colore già utilizzato per un'altra sessione pubblicata, viene avvisato tramite una finestra mostrata nell'angolo in basso a destra, Figura 13, questo però non gli impedirà di scegliere comunque quel colore.

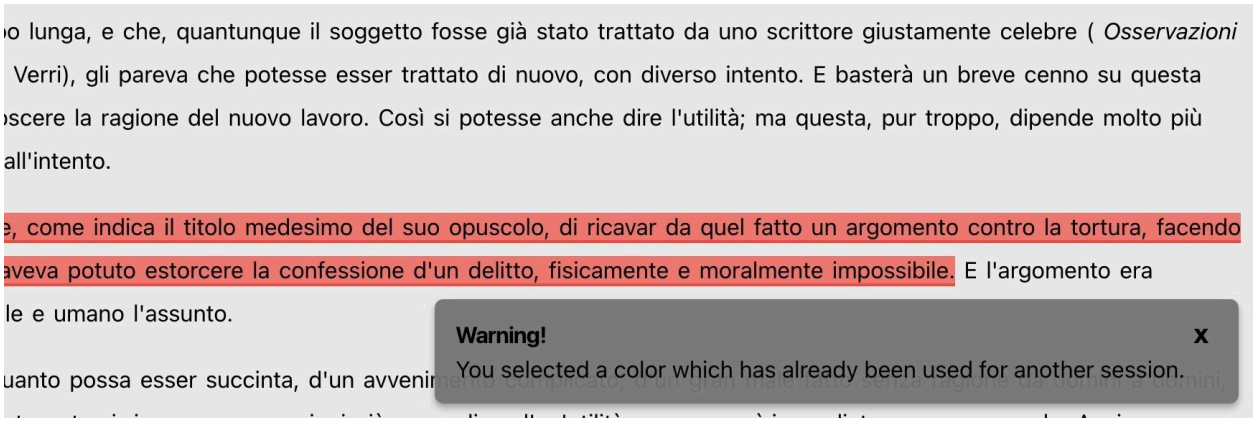

Figura 13: Messaggio di avvertimento per il colore della sessione già utilizzato.

Una volta inserito il nome e assegnato un colore alla sessione, le relative note verranno evidenziate con lo stesso colore e sarà possibile salvare e chiudere la sessione corrente. Cliccando sull'icona a nuvoletta, viene mostrata una finestra che richiede una conferma, Figura 14.

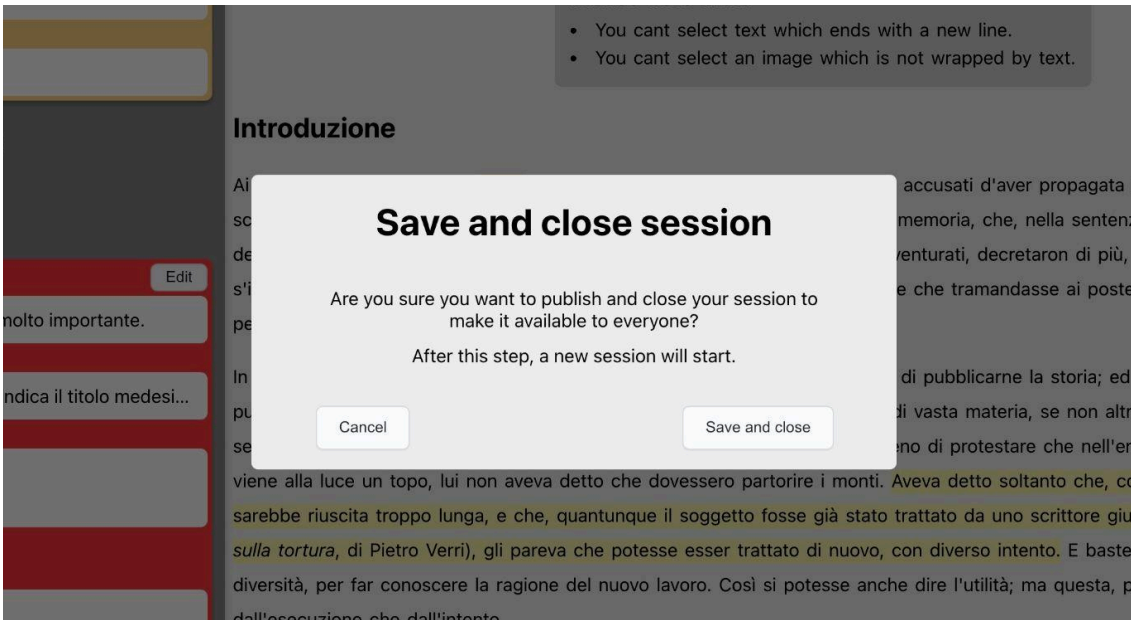

Figura 14: Finestra di salvataggio e chiusura sessione.

Cliccando su *Save and close*, tutte le note presenti nella sessione dell'utente vengono rese disponibili a tutti gli utenti iscritti al sistema, e presenteranno anche tutte le informazioni relative

alla sessione. Quando l'utente creerà una nuova annotazione sullo stesso testo, verrà creata una nuova sessione.

Se l'utente prova a salvare una sessione con un nome già presente tra le sessioni pubblicate, viene avvisato tramite una finestra, Figura 15, mostrata nell'angolo in basso a destra.

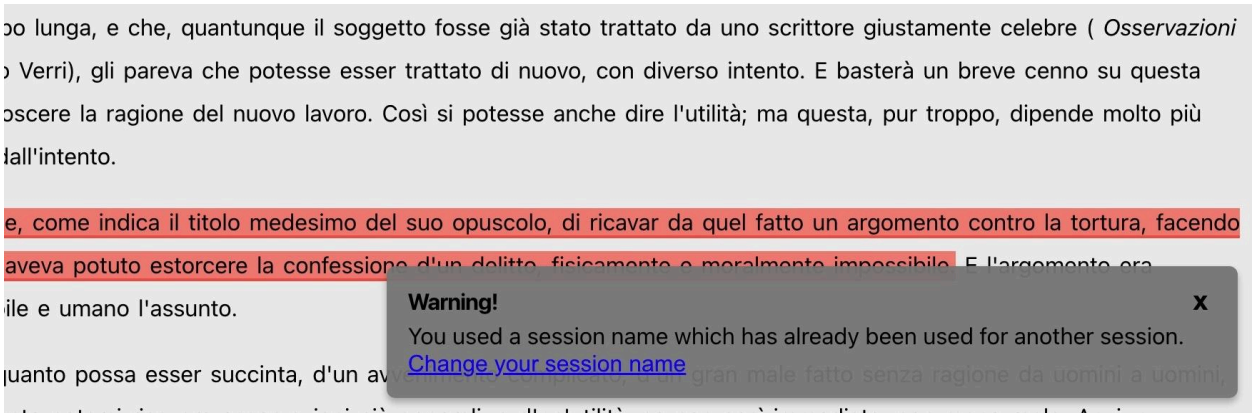

Figura 15: Messaggio di avvertimento per nome sessione già utilizzato.

Nel momento in cui viene effettuato l'accesso alla schermata di annotazione di un testo in cui sono presenti note non ancora pubblicate, l'utente viene avvisato tramite una finestra, nella Figura 16, mostrata nell'angolo in basso a destra.

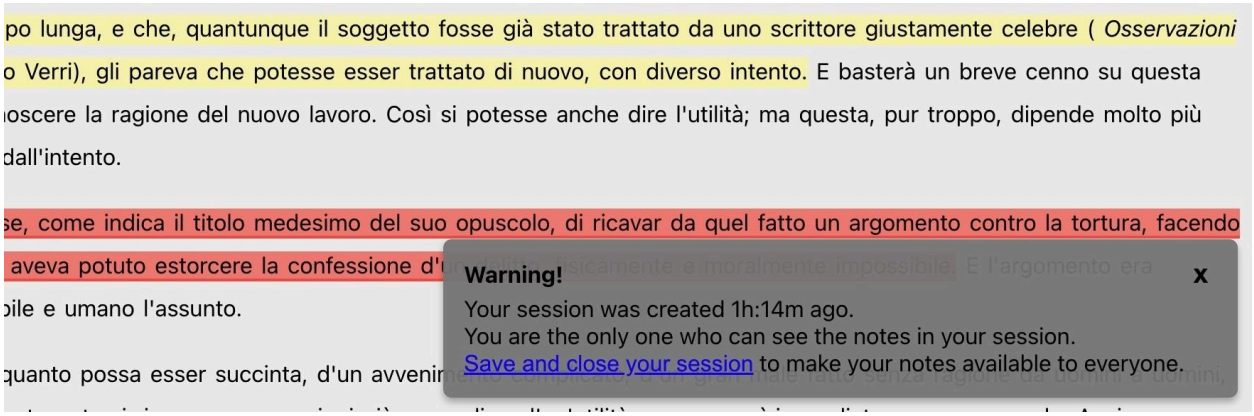

Figura 16: Messaggio di avvertimento per una sessione che deve ancora essere salvata e chiusa.

3.3 Creazione commento

L'utente ha la possibilità di commentare le annotazioni nella sua sessione e le annotazioni pubblicate da qualsiasi utente. Cliccando su una porzione di testo evidenziata, viene mostrata una finestra, Figura 17, contenente l'annotazione principale e tutti i commenti relativi ad essa. Da questa finestra l'utente può inserire un commento.

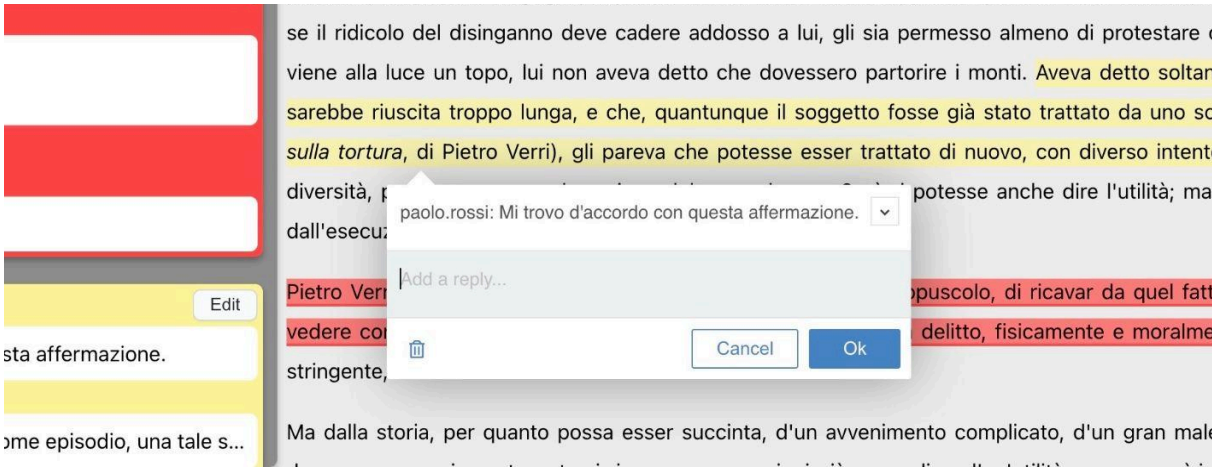

Figura 17: Finestra per l'inserimento di commenti.

Se l'utente commenta una nota presente nella propria sessione, il commento viene salvato nella relativa nota, che resta tra le annotazioni non ancora pubblicate. Se l'utente commenta una nota già pubblicata, essa viene direttamente salvata nella relativa nota pubblica, senza bisogno di ulteriori conferme.

3.4 Modifica e eliminazione

L'utente può modificare o eliminare le sue annotazioni o commenti, cliccando la freccia sulla destra, e selezionando *Edit* o *Delete*, Figura 18.

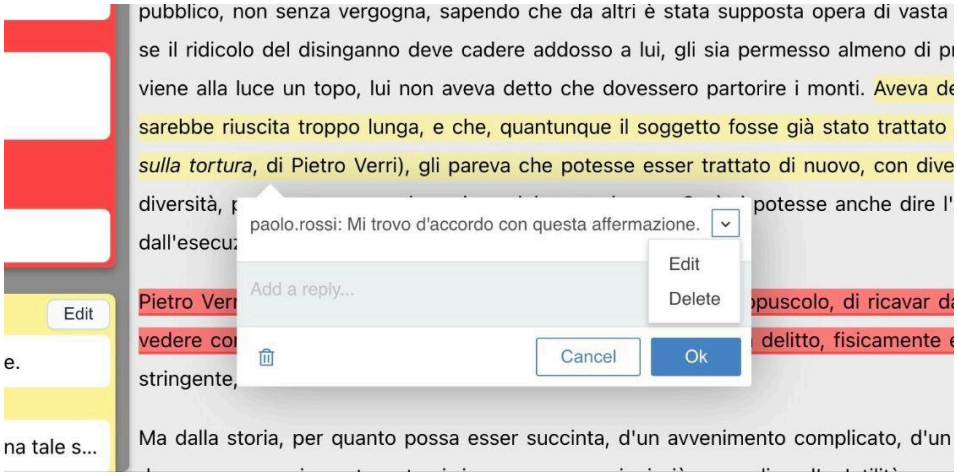

Figura 18: Finestra per la modifica o eliminazione di note o commenti.

Nel caso in cui l'utente tentasse di modificare o eliminare una nota non creata dallo stesso, viene avvisato tramite una finestra, Figura 19, mostrata nell'angolo in basso a destra.

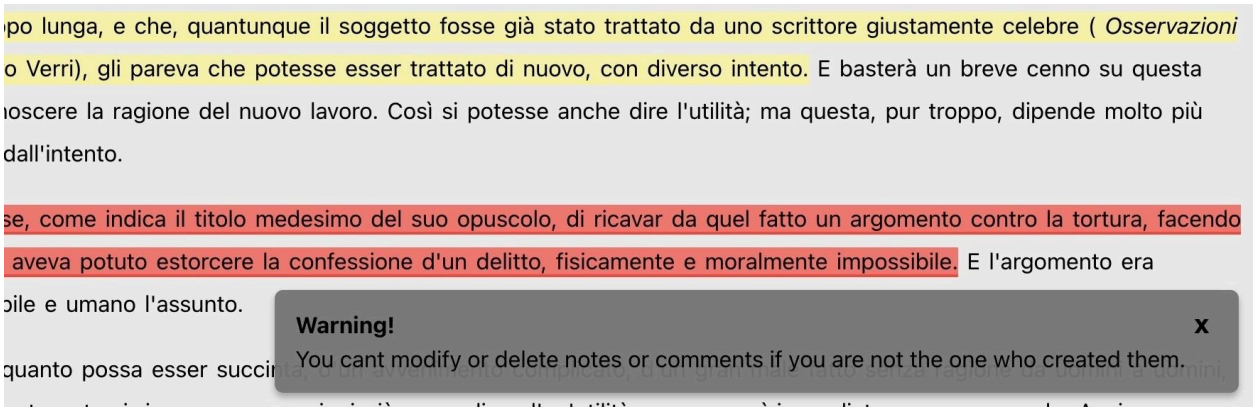

Figura 19. Messaggio di avvertimento per un tentativo di modifica o cancellazione di una nota o commento non creato dall'utente corrente.

L'utente può eliminare tutte le note non pubblicate relative alla sua sessione cliccando l'icona a cestino che si trova di fianco a *Your Session*. Viene quindi mostrata una finestra, Figura 20, che richiede una conferma.

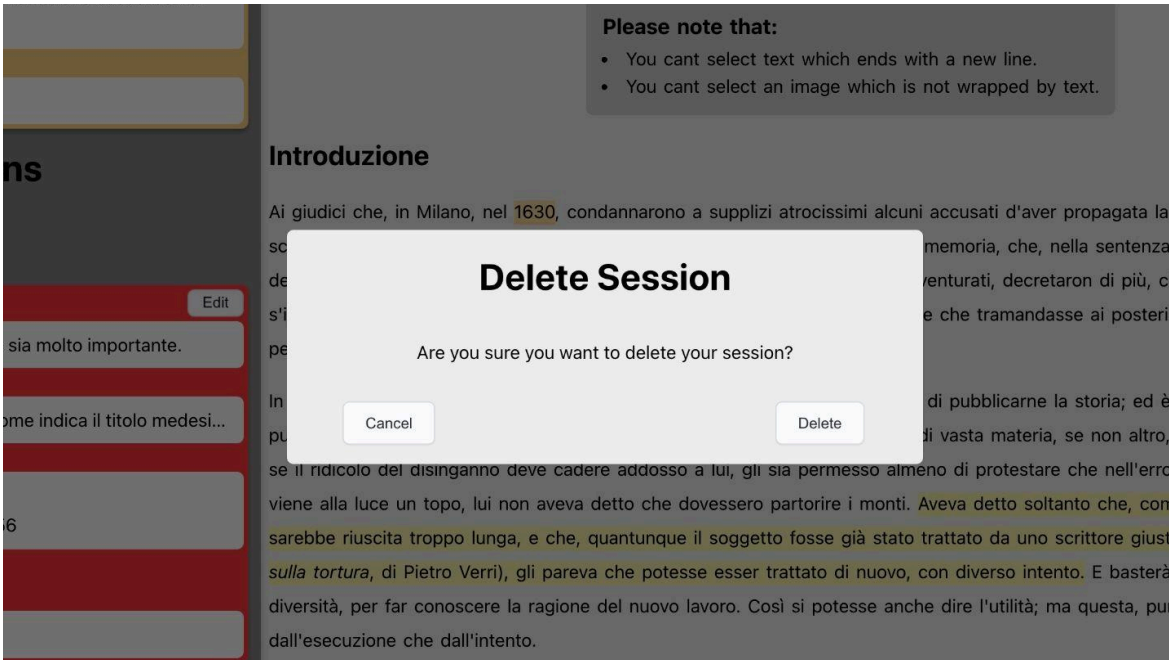

Figura 20: Finestra di conferma per l'eliminazione della sessione

3.5 Filtro e navigazione

L'utente può filtrare le note pubblicate da tutti gli utenti registrati al sistema selezionando il valore desiderato dai due dropdown menu *Author* e *Session*. Quando l'utente seleziona un valore da uno o da entrambi i menu, vengono mostrate sul testo e nella sezione annotazioni solamente le note che rientrano nei parametri scelti, Figura 21. Si possono rimuovere i filtri cliccando il bottone *Clear filters*.

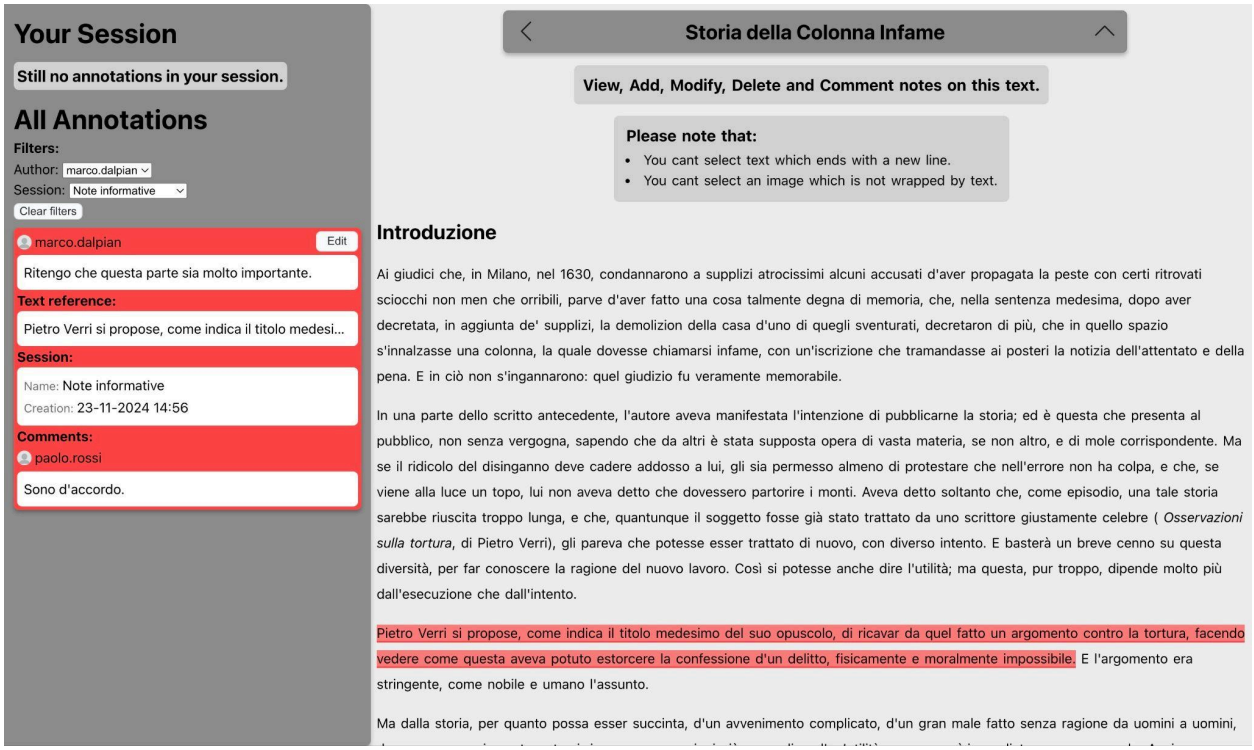

Figura 21: Note mostrate dopo aver impostato i filtri.

Nel caso in cui il testo presenti un'alta densità di note, l'utente può aiutarsi nell'individuazione di una specifica nota cliccando il bottone *Edit*, presente sulle annotazioni nella sezione a sinistra. Quest'ultimo aprirà sul testo la finestra, Figura 22, dedicata a:

- Modifica nota.
- Eliminazione nota.
- Aggiunta commenti.
- Modifica commenti.
- Eliminazione commenti.

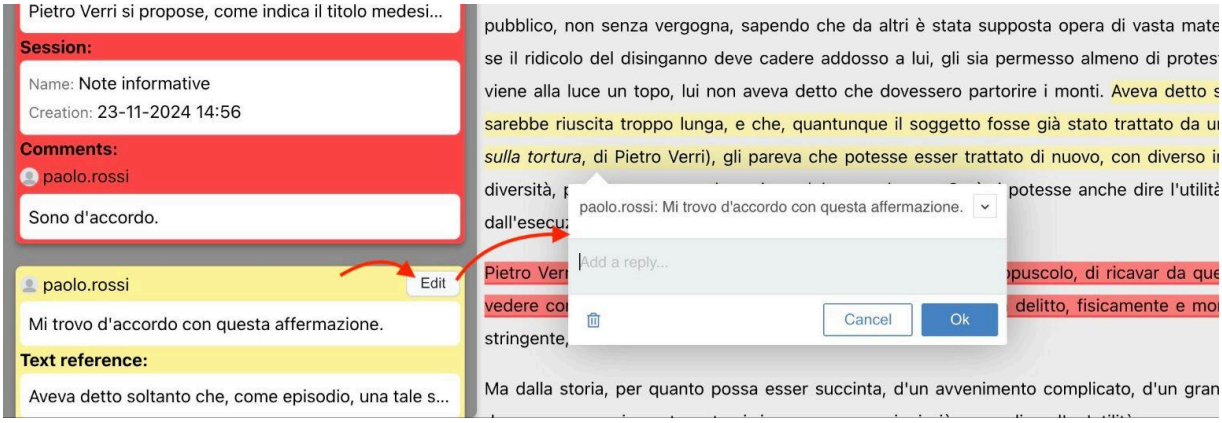

Figura 22: Finestra mostrata dopo aver cliccato sul bottone *Edit*.

Nel menu fissato in cima alla pagina, contenente il titolo del testo che si sta consultando, sono presenti due bottoni utili alla navigazione, Figura 23. Quello sulla sinistra permette all'utente di chiudere la visualizzazione del testo corrente, e quello sulla destra scorre il testo fino ad arrivare in cima allo stesso.

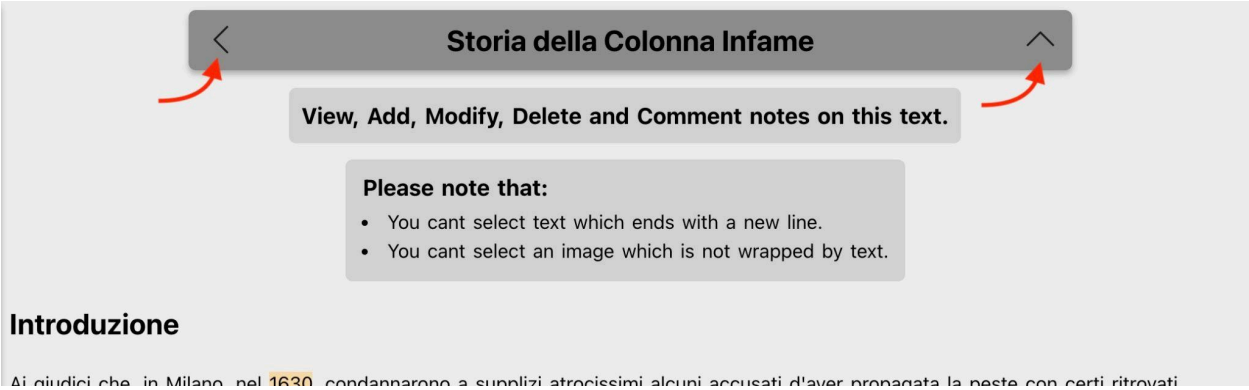

Figura 23: Bottoni utili alla navigazione.

3.6 Schermate esterne

La versione demo presenta due schermate aggiuntive rispetto al componente core di Text Annotator, sono state realizzate con lo scopo di mostrarne il funzionamento senza che debba essere prima integrato in un'applicazione web. Quando l'utente accede alla versione demo, viene mostrata una schermata di login che richiede solamente l'inserimento dell'username. È stata presa questa scelta perchè quando Text Annotator verrà integrato, le informazioni relative

all’utente verranno richieste nella schermata di login di DocuDipity, e fornite a Text Annotator come parametri props. In seguito alla conferma dell’username, viene mostrata una schermata che elenca tre testi, Figura 24. Cliccando su uno di questi, l’utente verrà reindirizzato alla pagina di annotazione del relativo testo, consultabile nella Figura 9. Anche in questo caso i parametri relativi al testo che si vuole annotare verranno forniti a Text Annotator come props. L’utente può tornare alla schermata di login cliccando il bottone “<” presente nell’angolo in alto a sinistra.

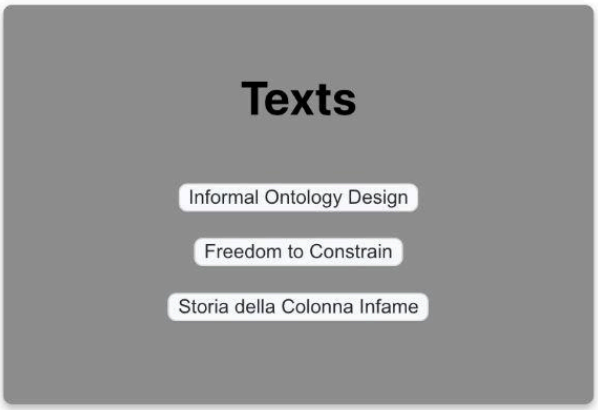

Figura 24: Menu utile alla navigazione tra i vari testi messi a disposizione dal server di Text Annotator.

Capitolo 4

Il front-end di Text Annotator

Text Annotator è composto da due parti:

- Client-side: parte di codice che viene eseguita dal browser dell'utente quando visita la pagina web relativa all'applicazione. Essa permette al sito web di avere un'interfaccia grafica con cui l'utente può interagire.
- Server-side: parte di codice che viene eseguita dal client ogni volta che ha bisogno di mostrare, aggiungere, modificare o eliminare le informazioni che vengono presentate nella pagina web.

Nelle Sezioni 4.1, 4.2 e 4.3 vengono descritte le caratteristiche del lato client di Text Annotator, mentre nelle Sezioni 5.1, 5.2 e 5.3 vengono descritte le caratteristiche del lato server di Text Annotator.

4.1 Framework

TypeScript [26] è il framework utilizzato per lo sviluppo del lato client di Text Annotator. È stato scelto perché è il framework utilizzato in DocuDipity, così da facilitare l'integrazione. Si tratta di un framework basato su JavaScript, offre le stesse funzionalità e aggiunge un ulteriore sistema di gestione dei types. JavaScript offre types come *string* e *number*, ma non controlla che vengano assegnati in modo consistente, TypeScript si occupa di effettuare questo controllo. Un'addizionale livello di controllo rende il codice più sicuro, di conseguenza il beneficio principale di questo framework è che può evidenziare comportamenti inattesi nel codice, abbassando la possibilità di generazione di bug.

4.2 Librerie

Nella Figura 25 viene mostrata la lista delle *dependencies* del lato client di Text Annotator.

```
"dependencies": {
    "@recogito/recogito-js": "^1.8.4",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.18.108",
    "@types/react": "^18.3.6",
    "@types/react-dom": "^18.3.0",
    "ajv": "^8.17.1",
    "ajv-keywords": "^5.1.0",
    "axios": "^1.7.7",
    "buffer": "^6.0.3",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-html-parser": "^2.0.2",
    "react-scripts": "5.0.1",
    "typescript": "^4.9.5",
    "uuid": "^11.0.3",
    "web-vitals": "^2.1.4"
  }
```

Figura 25: Lista delle *dependencies* del lato client di Text Annotator.

Di seguito la descrizione delle *dependencies*:

- @recogito/recogito-js [23], libreria principale di Text Annotator, permette di creare e gestire le annotazioni. Nella Figura 26 viene mostrato un esempio di inizializzazione di un'istanza RecogitoJS, "my-content" corrisponde all'id del contenitore dell'elemento che si vuole annotare.

```
const r = new Recogito({ content: 'my-content' });
```

Figura 26: Inizializzazione di un istanza RecogitoJS.

- @testing-library/jest-dom [11], libreria che estende la libreria Jest, rende i test più dichiarativi, chiari da leggere e da gestire.
- @testing-library/react [22], servizi utili al testing del React DOM, che incoraggiano l'uso di buone pratiche di testing.
- @testing-library/user-event [20], libreria che simula interazioni dell'utente inviando gli eventi che si verificherebbero se l'interazione avesse luogo in un browser.
- @types/jest, pacchetto che contiene le definizioni dei type per Jest.
- @types/node, pacchetto che contiene le definizioni dei type per Node.
- @types/react, pacchetto che contiene le definizioni dei type per React.
- @types/react-dom, pacchetto che contiene le definizioni dei type per react-dom.
- ajv [6], libreria che permette di implementare complesse logiche di validazione dati, utilizzando schemi dichiarativi per dati JSON, senza il bisogno di scrivere codice.
- ajv-keywords [8], libreria che che mette a disposizione custom JSON-Schema keywords per il validatore ajv.
- axios [15], libreria utile alla creazione di richieste HTTP a risorse esterne. Nella Figura 27 viene mostrato un esempio di utilizzo della libreria.

```
await axios.get('http://localhost:3000/annotator/'+props.bookID)
  .then(response => {
    const notesAPI = response.data;
    console.log('List of notes:', notesAPI);
    setNotes(notesAPI);
});
```

Figura 27: Esempio di richiesta GET utilizzando la libreria axios.

- buffer [1], libreria di manipolazione dati utile al corretto funzionamento di react-html-parser.
- react [17], libreria JavaScript che semplifica lo sviluppo di interfacce utente.
- react-dom [17], libreria che offre metodi che vengono usati ad alto livello nell'applicazione per renderizzare il componente padre della stessa, o per offrire una

forma di uscita di emergenza dal modello React, qualora ce ne fosse bisogno. Nella Figura 28 viene mostrato un esempio di utilizzo della libreria.

```
const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);

root.render(
  <App />
);
```

Figura 28: Esempio di renderizzazione del componente padre di un'applicazione web utilizzando la libreria react-dom.

- react-html-parser [18], libreria che converte stringhe HTML, quindi tutti gli elementi HTML, attributi e stili contenuti, nei loro equivalenti React. Nella Figura 29 viene mostrato un esempio di utilizzo della libreria.

```
        <div id="bookDiv">{ReactHtmlParser(props.bookContent)}</div>
```

Figura 29: Esempio di renderizzazione di un documento HTML utilizzando la libreria react-html-parser.

- react-scripts [4], libreria utile all'inizializzazione di un'applicazione React.
- typescript [12], framework su cui si basa l'applicazione.
- uuid [24], libreria utile alla creazione di RFC9562 UUIDs (Universally Unique IDentifiers). Nella Figura 30 viene mostrato un esempio di utilizzo della libreria.

```
1  const newID = uuidv4();
```

Figura 30: Esempio di creazione di un identificatore univoco utilizzando la libreria uuid.

- web-vitals [27], libreria utile alla misura di tutte le metriche Web Vitals (user experience e performance dell'applicazione) su veri utenti, in un modo che combacia con come sono misurate da Chrome.

4.3 Struttura client-side

Di seguito un elenco relativo alla struttura lato client di Text Annotator:

- *index.tsx*: entry point dell'applicazione, renderizza App.tsx
- *App.tsx*: si occupa di renderizzare Login.tsx, TextsList.tsx o TextAnnotator.tsx
- *Login.tsx*: la prima schermata visibile all'utente quando accede all'applicazione, richiede l'inserimento dell'username.
- *TextsList.tsx*: si accede a questa schermata dopo aver confermato il proprio username, da qui è possibile selezionare il testo che si vuole annotare, oppure tornare alla schermata di login.
- *TextAnnotator.tsx*: componente principale dell'applicazione:
    - Crea un oggetto RecogitoJS.
    - Definisce gli event handler di RecogitoJS.
    - Definisce le chiamate axios:
        - *fetchNotes*: per ottenere tutte le note relative al testo che si vuole annotare.
        - *postNote*: per salvare la nota appena creata.
        - *modifyNote*: per modificare una nota.
        - *deleteNote*: per eliminare una nota.
        - *deleteSession*: per eliminare una sessione.
    - Si occupa di renderizzare:
        - *NotesView.tsx*
        - *BookView.tsx*

        - *Modal.tsx*
        - *Warning.tsx*
- *NotesView.tsx*: gestisce la sezione di sinistra di Text Annotator:
    - Definisce le funzioni utili alla gestione dei filtri sulle note.
    - Renderizza la lista di note nella sessione.
    - Renderizza la lista di note pubbliche.
    - Renderizza gli elementi utili al filtraggio delle note pubbliche.
- *BookView.tsx*: gestice la sezione di destra di Text Annotator:
    - Renderizza informazioni e istruzioni che spiegano all'utente cosa può e non può fare.
    - Renderizza il corpo del testo che si vuole annotare.
- *Modal.tsx*: componente che gestisce la visibilità delle schermate a modale di Text Annotator.
- *Warning.tsx*: componente che gestisce la visibilità dei messaggi di avviso di Text Annotator.
- *Cartella css*:
    - *app.css*: proprietà css comuni alla maggior parte dei componenti dell'applicazione.
    - *notesView.css*: proprietà css relative al componente NotesView.tsx
    - *bookView.css*: proprietà css relative al componente BookView.tsx
    - *modal.css*: proprietà css relative al componente Modal.tsx
    - *warning.css*: proprietà css relative al componente Warning.tsx

Capitolo 5

Back-end a supporto di Text Annotator

Il lato server di Text Annotator è stato sviluppato in modo da riprodurre un API REST. Le API (Application Programming Interface) [3] sono meccanismi che consentono a due componenti software di comunicare tra loro usando una serie di definizioni e protocolli. Nello specifico, le API REST (Representational State Transfer), tra le più diffuse e flessibili sul Web ad oggi, definiscono una serie di funzioni GET (per recuperare un record), POST (per creare un record), PUT (per aggiornare un record) e DELETE (per eliminare un record) che i client possono utilizzare per accedere ai dati del server. Client e server scambiano dati tramite HTTP. La caratteristica principale delle API REST è l'assenza di stato. Per assenza di stato si intende che i server non salvano i dati del client tra le richieste. Le richieste del client al server sono simili agli URL che vengono digitati nel browser per visitare un sito Web. Le risposte del server sono dati semplici, senza il rendering grafico tipico di una pagina Web.
Nelle Sezioni 5.1, 5.2 e 5.3 vengono descritte le caratteristiche del lato server di Text Annotator.

5.1 Framework

Il lato server di Text Annotator è stato sviluppato utilizzando Node e Express. Node (o Node.js) [10] è un ambiente di esecuzione cross-platform che permette agli sviluppatori di creare applicazioni lato server utilizzando JavaScript. Questo ambiente è progettato per essere utilizzato al di fuori del contesto di un browser, quindi eseguendo direttamente su un computer o su un sistema operativo server. Pertanto, l'ambiente esclude le API JavaScript specifiche del browser e aggiunge il supporto per le API più tradizionali del sistema operativo, comprese le librerie per HTTP e file system. Rende quindi possibile la creazione di un server web utilizzando il pacchetto HTTP di Node. Express [9] è il più popolare framework per lo sviluppo di applicazioni lato server con Node. Fornisce vari meccanismi per:

- Creare handlers per richieste con diversi verbi HTTP in diversi URL.

- Integrare con motori di rendering *view* per generare risposte inserendo dati nei modelli.
- Impostare impostazioni comuni per applicazioni web, come la porta da utilizzare per la connessione, e la posizione dei modelli utilizzati per il rendering della risposta.
- Aggiungere middleware di elaborazione delle richieste aggiuntive in qualsiasi punto all'interno della pipeline di gestione delle richieste.

Express è un framework minimalista, infatti vari sviluppatori hanno creato dei pacchetti middleware compatibili per la gestione dei problemi dello sviluppo web. Sono disponibili librerie per gestire cookies, sessioni, login, parametri URL, dati POST, e molte altre. Per questi motivi Express è stato scelto come framework per lo sviluppo del lato server di Text Annotator.

5.2 Librerie

Nella Figura 31 viene mostrata la lista delle *dependencies* del lato server di Text Annotator.

```
  "devDependencies": {
    "@types/cors": "^2.8.17",
    "@types/express": "^5.0.0",
    "@types/node": "^22.7.3",
    "express": "^4.21.0",
    "ts-node": "^10.9.2",
    "typescript": "^5.6.2"
  },
  "dependencies": {
    "@types/uuid": "^10.0.0",
    "cors": "^2.8.5",
    "jsdom": "^25.0.1",
    "uuid": "^10.0.0"
  }
```

Figura 31: Lista delle *dependencies* del lato server di Text Annotator.

Di seguito la descrizione delle *dependencies*:

- @types/cors, pacchetto che contiene le definizioni dei type per CORS.
- @types/express, pacchetto che contiene le definizioni dei type per Express.
- @types/node, pacchetto che contiene le definizioni dei type per Node.js.

- express [14], framework su cui si basa l'applicazione.
- ts-node [6] [25], motore di esecuzione TypeScript per Node.js.
- typescript [12], framework su cui si basa l'applicazione.
- @types/uuid, pacchetto che contiene le definizioni dei type per uuid.
- cors [28], pacchetto node.js che fornisce un middleware Express che può essere usato per abilitare CORS con varie opzioni. Nella Figura 32 viene mostrato un esempio di utilizzo della libreria.

```
const app = express();

const allowedOrigins = ['http://localhost:3001'];
const options: cors.CorsOptions = {
  origin: allowedOrigins
};

app.use(cors(options));
app.use(express.json());
```

Figura 32: Esempio di utilizzo della libreria CORS per permettere a determinate origini di accedere al server.

- jsdom [4], libreria usata per estrarre specifici frammenti da file HTML. Nella Figura 33 viene mostrato un esempio di utilizzo della libreria.

```
const bookString = fs.readFileSync(path.resolve(__dirname, './books/book1.html'), 'utf8');
const bookDOM = new jsdom.JSDOM(bookString);
let title: string = bookDOM.window.document.querySelector("head").textContent;
```

Figura 33: Esempio di utilizzo della libreria jsdom per estrarre il titolo di un documento HTML.

- uuid [24], libreria utile alla creazione di RFC9562 UUIDs (Universally Unique IDentifiers). Nella Figura 30 viene mostrato un esempio di utilizzo della libreria.

5.3 Struttura server-side

Il server definisce due interfacce:

- Note, contenente tutti i parametri richiesti dalla libreria RecogitoJS, e alcuni campi aggiuntivi utili alla gestione delle note e delle sessioni, Figura 31.

```
export interface Note {
  '@context': string;
  body: [{id: string, purpose: string, type: string, value: string, userID: string,
    username: string, isComment: boolean}];
  id: string;
  target: Selector[];
  type: string;
  bookID: string;
  session: {name: string, description: string, color: string, date: Date, type: string};
}

interface Selector{
  selector:[{exact: string, type: string}, {end: number, start: number, type: string}];
}
```

Figura 31: Interfaccia Note.

- Book, contenente le informazioni utili al salvataggio dei testi che vengono messi a disposizione dal server, Figura 32.

```
export interface Book {
  id: string;
  title: string;
  content: string;
}
```

Figura 32: Interfaccia Book.

Il server, utile alla demo di Text Annotator, mette a disposizione tre testi, che sono conservati nella cartella *books* del server. Ogni file HTML relativo ai testi viene salvato in una variabile string e viene estratto il titolo. I testi vengono poi salvati all'interno di un array di oggetti *Book*, e ad ognuno viene assegnata una stringa identificativa univoca, Figura 33.

```
const bookString1 = fs.readFileSync(path.resolve(__dirname, './books/book1.html'),
'utf8');
let book1: string = bookString1.substring(bookString1.indexOf("<body>")+6,
bookString1.lastIndexOf("</body>"));
const bookDOM1 = new jsdom.JSDOM(bookString1);
let title1: string = bookDOM1.window.document.querySelector("head").textContent;

let books: Book[] = [
  {id: uuidv4(), title: title1, content: book1},
  {id: uuidv4(), title: title2, content: book2},
  {id: uuidv4(), title: title3, content: book3}
];
```

Figura 33: Popolazione dell'array contenente i testi del server di Text Annotator.

Di seguito una lista riportante le chiamate messe a disposizione dal server:

- Testi:
  - http://localhost:3000/annotator/books (GET), per richiedere le informazioni riguardanti tutti i testi.
  - http://localhost:3000/annotator/:bookID (GET), per richiedere tutta la lista di note relative ad un singolo testo. *bookID* corrisponde alla stringa identificativa univoca del testo richiesto.
- Note e sessioni:
  - http://localhost:3000/annotator (POST), per creare una nuova nota. Il body della richiesta deve contenere l'oggetto Annotazione da salvare nel server.
  - http://localhost:3000/annotator/:id (PUT), per aggiornare una nota. *id* corrisponde alla stringa identificativa univoca della nota da modificare. Il body della richiesta deve contenere l'oggetto Annotazione contenente le modifiche da salvare nel server. Modifiche all'annotazione disponibili attraverso questa richiesta:
    - modificare un annotazione.
    - modificare un commento.
    - cancellare un commento.
    - modificare le informazioni di sessione.
  - http://localhost:3000/annotator/:id (DELETE), per eliminare una nota. *id* corrisponde alla stringa identificativa univoca dell'annotazione da eliminare dal server.

- http://localhost:3000/annotator/session/:userID/:bookID (DELETE), per eliminare una sessione. *userID* corrisponde alla stringa identificativa univoca dell'autore della sessione da eliminare dal server. *bookID* corrisponde alla stringa identificativa univoca del testo che corrisponde alla sessione da eliminare dal server.

Capitolo 6

Come integrare Text Annotator in DocuDipity

In questo capitolo viene descritta la procedura da seguire per integrare il sistema Text Annotator in DocuDipity, o in un'altra applicazione con framework TypeScript:

- Sezioni 6.1, 6.2 e 6.3, istruzioni utili per l'integrazione del componente client principale e delle sue dipendenze.
- Sezioni 6.4, 6.5 e 6.6, istruzioni utili per l'integrazione degli endpoint che devono essere messi a disposizione al client per poter richiedere e gestire i dati relativi alle annotazioni, commenti, sessioni e testi che deve presentare.

Il codice relativo alle applicazioni client e server di Text Annotator è disponibile all'indirizzo: [https://github.com/marcodalpian/text-annotator].

6.1 Client-side - Librerie

Nella Figura 25 viene mostrata la lista delle *dependencies* necessarie per il funzionamento del lato client di Text Annotator. Eseguire i tre passaggi seguenti per installare le librerie nell'applicazione:

- Copiare le librerie che non sono già presenti nel file *package.json* dell'applicazione, ed aggiungerle alla lista.
- Eseguire il comando *npm install* per effettuare l'installazione delle nuove librerie.
- Eseguire il comando *npm audit fix* per risolvere le eventuali vulnerabilità rilevate nel progetto e nelle sue dipendenze.

6.2 Client-side - Files

Nella Figura 34 viene mostrata la directory dell'applicazione lato client di Text Annotator.

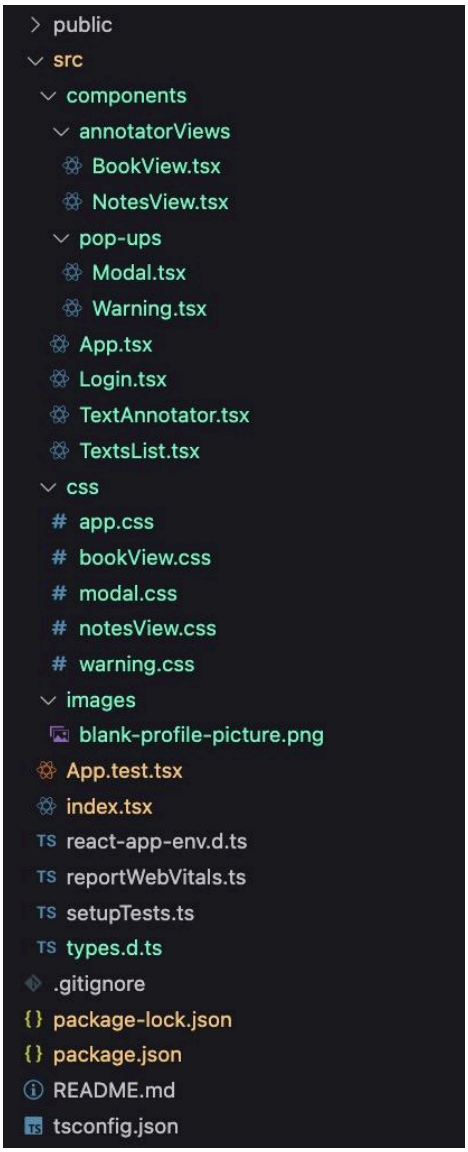

Figura 34: Directory dell'applicazione client di Text Annotator.

Copiare nell'applicazione i seguenti files, i restanti sono files necessari al funzionamento di Text Annotator all'interno della sua applicazione web completa:

- Cartella src:
    - Cartella components:
        - cartella annotatorViews:

            - BookView.tsx
            - NotesView.tsx
        - cartella pop-ups:
            - Modal.tsx
            - Warning.tsx
        - TextAnnotator.tsx
    - Cartella css:
        - app.css
        - bookView.css
        - modal.css
        - notesView.css
        - warning.css
    - Cartella images:
        - blank-profile-picture.png
    - types.d.ts

6.3 Client-side - Render e parametri

TextAnnotator.tsx è il componente principale dell'applicazione, quindi per mostrare il Text Annotator nell'applicazione bisogna renderizzare questo componente fornendogli i seguenti parametri come props(nella Figura 35 viene mostrata l'interfaccia di Text Annotator):

- bookID, stringa identificativa univoca del testo che si vuole annotare, fornita dal server contenente la collezione di documenti annotabili.
- title, titolo del testo che si vuole annotare.
- bookContent, codice HTML del testo che si vuole annotare, racchiuso in una variabile di tipo string.
- userID, stringa identificativa univoca dell'utente loggato che vuole annotare il testo.
- username, nome dell'utente loggato che vuole annotare il testo.
- renderPage, funzione che permette all'utente di chiudere il Text Annotator, e quindi mostrare la pagina precedente. Questa funzione viene richiamata dal bottone con l'icona "<" presente nel menu fissato in alto, alla sinistra del titolo del testo.

```
interface TextAnnotatorProps {
  bookID: string;
  title: string;
  bookContent: string;
  userID: string;
  username: string;
  renderPage: Function;
}
```

Figura 35: Interfaccia di Text Annotator contenente i parametri props.

Nella Figura 36 viene mostrato un esempio di renderizzazione di TextAnnotator.tsx.

```
import TextAnnotator from './TextAnnotator';

export default function Test() {
  return (
          <TextAnnotator bookID={bookID} title={title} bookContent={content}
userID={userID} username={username} renderPage={renderPage} />
  );
}
```

Figura 36: Esempio di renderizzazione del componente Text Annotator.

6.4 Server-side - Librerie

Nella Figura 31 viene mostrata la lista delle *dependencies* necessarie per il funzionamento del lato server di Text Annotator. Eseguire i tre passaggi seguenti per installare le librerie nel server:

- Copiare le librerie che non sono già presenti nel file *package.json* del server, ed aggiungerle alla lista.
- Eseguire il comando *npm install* per effettuare l'installazione delle nuove librerie.
- Eseguire il comando *npm audit fix* per risolvere le eventuali vulnerabilità rilevate nel progetto e nelle sue dipendenze.

6.5 Server-side - Modelli e endpoint

Nella Figura 37 viene mostrata la directory dell'applicazione lato server di Text Annotator.

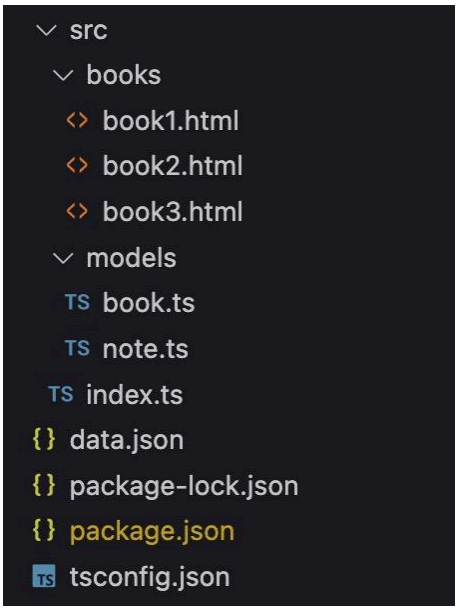

Figura 37: Directory dell'applicazione server di Text Annotator.

Copiare nel server la cartella *models*, contenente le interfacce che definiscono ed esportano gli oggetti *Note* e *Book*. Successivamente integrare gli endpoint presenti nel file *index.ts*(descritti nella Sezione 5.3):

- app.get('/annotator/books', …
- app.get('/annotator/:bookID', …
- app.post('/annotator', …
- app.put('/annotator/:id', …
- app.delete('/annotator/:id', …
- app.delete('/annotator/session/:userID/:bookID', ...

6.6 Server-side - Testi e annotazioni

I testi di Docudipity dovranno essere inseriti all'interno di un array di oggetti *Book*(descritto nella Sezione 2), seguendo la modalità che viene proposta nel file *index.ts* del server di Text Annotator, Figura 38.

```
let books: Book[] = [
  {id: uuidv4(), title: title1, content: book1},
  {id: uuidv4(), title: title2, content: book2},
  {id: uuidv4(), title: title3, content: book3}
];
```

Figura 38: Inserimento di testi in un array di oggetti *Book* del server di Text Annotator.

Ogni oggetto *Book* inserito deve contenere:

- id: stringa identificativa univoca del testo, fornita dal server contenente la collezione di documenti annotabili.
- title: titolo del testo.
- content: stringa contenente il codice HTML del testo.

Le annotazioni verranno inserite all'interno di un array di oggetti *Note*(descritto nella Sezione 2), presente nel file *index.ts* del server di Text Annotator, Figura 39.

```
let notes: Note[] = [];
```

Figura 39: Array di oggetti *Note* del server di Text Annotator.

Capitolo 7

Conclusioni

L’elaborato in questione tratta di un percorso di progettazione e sviluppo di un’applicazione web utile all’annotazione di testi che ho chiamato Text Annotator. Ho deciso di creare questo sistema con l’obiettivo che possa essere integrato nell’applicazione web DocuDipity. Si tratta di un progetto che ha come idea principale quella di combinare visualizzazioni coordinate in modo da aiutare i ricercatori a scoprire tratti degli articoli scientifici, senza necessariamente leggerne l'intero contenuto. Quando Text Annotator sarà integrata in DocuDipity, essa aggiungerà una nuova ed importante funzionalità al sito web, che al momento non presenta nessun meccanismo che permetta di aggiungere contenuti o commentare documenti. Gli studiosi e ricercatori iscritti ad esso, mentre consulteranno le visualizzazioni degli articoli, potranno confrontarsi ed interagire utilizzando questo sistema di annotazione testi, oppure potranno semplicemente scrivere delle note private direttamente sugli articoli, senza dover per forza renderle pubbliche.
DocuDipity è stata realizzata utilizzando un framework TypeScript. Di conseguenza ho dovuto sviluppare Text Annotator utilizzando il medesimo framework, così che potesse essere facilmente integrabile. Per realizzare un sistema che offra la possibilità di selezionare porzioni di testo alle quali aggiungere annotazioni, mi sono immerso nella ricerca di una libreria compatibile JavaScript o Typescript che mi potesse aiutare nel processo di sviluppo. Durante la mia ricerca ho trovato tre librerie possibilmente adatte all’utilizzo che volevo farne. Due di esse non erano perfettamente compatibili col progetto che volevo sviluppare. La terza libreria, chiamata RecogitoJS, era proprio quello che stavo cercando. Ovviamente le funzionalità rese disponibili da essa non erano sufficienti per soddisfare i requisiti funzionali dell’applicazione che volevo sviluppare, così ho dovuto integrarla in un’applicazione che potesse modificarne il comportamento e allo stesso tempo aggiungere molte altre funzionalità, come la possibilità di colorare le note o la creazione di sessioni di annotazioni. In questo modo sono riuscito a sviluppare l’applicazione lato client di Text Annotator, che aveva bisogno di un lato server per poter salvare tutte le annotazioni create dagli utenti. Ho realizzato il back-end utilizzando

l’ambiente di esecuzione Node ed il framework Express. In questo modo ho dato vita ad un API di tipo REST, che quindi presenta un’architettura ad elevate prestazioni ed affidabilità. Anche il back-end è stato realizzato con lo scopo che potesse essere facilmente integrabile con il server di DocuDipity.

Per quanto riguarda gli sviluppi futuri, si potrebbe espandere la funzionalità di sessione, con la possibilità per ogni utente di avere molteplici sessioni aperte simultaneamente, così da poter conservare una collezione di note private, e allo stesso tempo pubblicare altre annotazioni. Inoltre, la possibilità di aggiungere note su immagini o grafici presenti nei documenti di Docudipity, sarebbe un'ulteriore funzionalità che renderebbe annotabile qualsiasi elemento contenuto nei testi.

Il mio auspicio è che Text Annotator possa un giorno facilitare le attività lavorative quotidiane di studiosi e ricercatori che utilizzano DocuDipity.

Sitografia

[1] Feross Aboukhadijeh, Jesse Tane, …(69 contributors): buffer. https://github.com/feross/buffer, 2023.

[2] Annotation Unlimited: Hypothesis - Web & PDF Annotation. https://chromewebstore.google.com/detail/hypothesis-web-pdf-annota/bjfhmglciegochdpefhhlphglcehbmek, 2024.

[3] Cos'è un'API (Application Programming Interface)?, https://aws.amazon.com/it/what-is/api/

[4] Dan, Joe Haddad, Ian Sutherland, Ian Schmitz, Kristofer Giltvedt Selbekk, …(899 contributors): create-react-app. https://github.com/facebook/create-react-app, 2022.

[5] Domenic Denicola, Elijah Insua, Zirro, Sebastian Mayr, …(296 contributors): jsdom. https://github.com/jsdom/jsdom, 2022.

[6] Blake Embrey, Andrew Bradley, …(111 contributors): ts-node. https://github.com/TypeStrong/ts-node, 2023.

[7] Evgeny, Erik Brinkman, spaced4ndy, …(201 contributors): ajv. https://github.com/ajv-validator/ajv, 2024.

[8] Evgeny, Will Farrell, …(23 contributors): ajv-keywords. https://github.com/ajv-validator/ajv-keywords, 2021.

[9] Express, Framework web veloce, non categorico e minimalista per Node.js, https://expressjs.com/it/

[10] Express/Node introduction, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

[11] Ernesto García, Seth, John Gozde, Nicky McCurdy, Leandro Lourenci, …(113 contributors): jest-dom. https://github.com/testing-library/jest-dom, 2024.

[12] Anders Hejlsberg, Sheetal Nandi, Nathan Shively-Sanders, Daniel Rosenwasser, …(790 contributors): TypeScript. https://github.com/microsoft/TypeScript, 2024.

[13] H. Pleitez: Annotate the Web. https://chromewebstore.google.com/detail/annotate-the-web/hkdbcemiphlcpmcphjgfaepgciimcoef, 2022.

[14] TJ Holowaychuk, Douglas Wilson, jongleberry, …(321 contributors): express. https://github.com/expressjs/express, 2024.

[15] Jay, Matt Zabriskie, Dmitriy Mozgovoy, Nick Uraltsev, …(455 contributors): axios. https://github.com/axios/axios, 2024.

[16] Vladimir Kokovic, Jessica Feng: react-text-annotation. https://github.com/vlddlv/react-text-annotation, 2024.

[17] Paul O'Shannessy, Dan, Sebastian Markbåge, Andrew Clark, Sophie Alpert, Joseph Savona, …(1668 contributors): react. https://github.com/facebook/react, 2024.

[18] peternewnham, Kevin Huang, MJJ: react-html-parser. https://github.com/peternewnham/react-html-parser, 2017.

[19] Poggi F.; Ciancarini P.; Di Iorio A.; Peroni S.; Vitali F., Exploiting coordinated views for scholarly reading and analysis, in: Proceedings - DMSVIVA 2019: 25th International DMS Conference on Visualization and Visual Languages, Knowledge Systems Institute Graduate School, KSI Research Inc., 2019, 2019, pp. 113 - 124. https://doi.org/10.18293/DMSVIVA2019-021, 2019.

[20] Giorgio Polvara, Philipp Fritsche, Kent C. Dodds, …(98 contributors): user-event. https://github.com/testing-library/user-event, 2023.

[21] Morten Rand-Hendriksen, Molly White, Kees: annotate. https://github.com/molly/annotate, 2022.

[22] Sebastian "Sebbie" Silbermann, Kent C. Dodds, Alex Krolick, …(173 contributors): react-testing-library. https://github.com/testing-library/react-testing-library, 2024.

[23] Rainer Simon, Aapo Virta, Richard Eckart de Castilho, Robin Tissot, Ben Silverman, The Gitter Badger, Robert Peters, Fejes Botond: RecogitoJS. https://github.com/recogito/recogito-js, 2023.

[24] Christoph Tavan, Robert Kieffer, Ignat Prokopovich, …(73 contributors): uuid. https://github.com/uuidjs/uuid, 2024.

[25] Rich Trott, Ryan Dahl, Ben Noordhuis, Isaacs, …(3542 contributors): node. https://github.com/nodejs/node, 2024.

[26] TypeScript for JavaScript Programmers, https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

[27] Philip Walton, Barry Pollard, Brendan Kenny, …(31 contributors): web-vitals. https://github.com/GoogleChrome/web-vitals, 2024.

[28] Douglas Wilson, Troy Goode, …(51 contributors): cors. https://github.com/expressjs/cors, 2024.

[29] Ziflow: Ziflow. https://www.ziflow.com/, 2024.

Ringraziamenti

Ringrazio il prof. Angelo Di Iorio per la disponibilità e l'entusiasmo mostrati durante lo svolgimento della tesi.

Un grande ringraziamento va anche a tutta la mia famiglia e i miei amici, che hanno sempre creduto in me e mi hanno sostenuto durante il percorso di laurea.