Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Dipartimento di Informatica - Scienze e Ingegneria

SVILUPPO DI "PLUGINIZER" PER L'EDITAZIONE DI DOCUMENTI STRUTTURATI DI MICROSOFT WORD

Laurea triennale in Informatica

RELATORE:
Chiar.mo
Prof. Fabio Vitali

CANDIDATO:
Andrea Napoli

II Sessione

Anno Accademico 2022/2023

A tutti coloro

che mi sono stati

vicini e hanno creduto

sempre in me

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il progetto di tesi consiste nello sviluppare ed implementare un software di utility per la progettazione e la modifica di documenti scritti all'interno del programma Microsoft Word.
Durante la fase di tirocinio abbiamo costruito una base solida dalla quale partire per espandere ulteriormente il mio elaborato finale.

L'obiettivo di questa tesi di laurea è stato quindi sviluppare e implementare il plug-in Pluginizer in modo che sia di supporto agli utenti, con lo scopo di ottimizzare l'organizzazione e la strutturazione di un documento, introducendo funzionalità avanzate, non presenti all'interno dell'editor di testo, utilizzate per semplificare e migliorare la creazione di contenuti.
I documenti a cui è indirizzato Pluginizer sono quelli strutturati, come quelli legali o giudiziari. All'interno di molti uffici del nostro paese, per la modifica e la creazione di questa tipologia di documenti, viene utilizzato Microsoft Word, che purtroppo non è il software adatto e di

conseguenza genera documenti non soddisfacenti.
È qui che entra in gioco Pluginizer: permetterà di editare documenti, utilizzando Microsoft Word, ma in maniera controllata, rispettando gli standard, come quello di Akoma Ntoso[1], uno standard XML sviluppato per facilitare la generarazione di documenti strutturati efficienti fornendo uno schema standardizzato.
Inizialmente, avevamo pensato di integrare nel progetto un metodo diretto per la conversione del documento Word in Akoma Ntoso una volta che il documento fosse stato completato. Tuttavia, abbiamo constatato che questo approccio avrebbe richiesto un impegno e una complessità notevolmente superiore, portandoci al di fuori dei confini dei temi e degli obiettivi principali del progetto. Infatti, avremmo dovuto creare un intero servizio autonomo, completamente separato dal contesto del progetto che stavamo attualmente sviluppando.

Un plug-in è un modulo software che viene integrato all'interno di un'applicazione, o di un programma, per estendere delle funzionalità già esistenti o fornire nuove capacità. Sono progettati per interagire con il programma principale, sfruttando le sue interfacce di programmazione e consentendo agli sviluppatori di introdurre funzionalità personalizzate tramite diversi linguaggi di programmazione: noi abbiamo utilizzato JavaScript affiancato dal framework React e sfruttato le API di Office.js per manipolare il DOM di Word.
Un plug-in opera come un modulo separato, mantenendo l'isolamento dal codice dell'applicazione ospite. Questo comporta che gli add-in possono essere sviluppati indipendentemente e poi distribuiti e installati

senza influire sulle funzionalità principali dell'applicazione.

Come già ribadito in precedenza, la piattaforma che ospiterà il nostro plug-in è Microsoft Word. Microsoft Word è uno dei programmi di elaborazione testi più conosciuti e più utilizzati a livello mondiale.
Si distingue per la sua interfaccia user-friendly, la quale permette la creazione e la modifica di documenti da parte di utenti di qualsiasi livello di esperienza. Inoltre la sua interfaccia è di tipo WYSIWYG (What You See Is What You Get), ovvero che il testo nel risultato finale apparirà esattamente come si vede nell'editor, permettendo di visualizzare istantaneamente il risultato ottenuto.

L'interfaccia di Pluginizer è molto chiara ed intuitiva e attraverso le sue sezioni è possibile eseguire semplici operazioni già presenti in Word come la creazione di liste e l'allineamento del testo, ma è possibile svolgere anche operazioni inedite in quest'ambiente come la creazione e l'utilizzo di nuovi stili e il salvataggio di informazioni associate ad un testo.

Ho deciso di sviluppare questo tipo di progetto poiché rappresentava per me un ambiente completamente inedito e sconosciuto nel quale non avevo mai programmato. La mancanza di esperienza ha portato ad una serie di sfide affascinanti e stimolanti. Ogni nuova funzionalità da implementare richiedeva una fase preliminare di studio approfondito, durante la quale ho dovuto comprendere i meccanismi su come Word sviluppasse i propri metodi, e cercare di capire come crearne altri non presenti sulla piattaforma ma che rispettassero le esigenze degli utenti.

Dopo la fase di studio, il passo successivo consisteva nel tradurre le competenze acquisite in soluzioni pratiche ed efficienti, sfruttando al meglio le varie opzioni messe a disposizione da Microsoft. Tra le sfide più stimolanti, ci sono quelle situazioni in cui Microsoft non forniva direttamente metodi o strumenti specifici per affrontare determinate problematiche, come per esempio l'espansione della selezione automatica fino alla fine della parola, la ricerca di tutte le istanze di un determinato testo all'interno del documento, oppure la creazione di nuovi stili personalizzati. In tali circostanze, ho dovuto suddividere il problema in sottoproblemi più gestibili e cercare di risolverli applicando metodologie alternative. Successivamente, ho dovuto unire queste soluzioni parziali per giungere infine a quella completa e definitiva.

Un altro aspetto importante da considerare è stato il processo di testing e validazione del plug-in. Dopo averlo sviluppato e caricato online attraverso un servizio di hosting, è stato fondamentale sottoporre il progetto ad un processo di test per verificarne la robustezza e l'efficacia. Questa fase ha implicato la collaborazione con un gruppo di persone, che hanno utilizzato il plug-in in situazioni reali per identificare eventuali bug o miglioramenti necessari. L'interazione con gli utenti è stata preziosa per raccogliere opinioni e risolvere i vari problemi riscontrati da quest'ultimi.

E' importante sottolineare che il plug-in è stato progettato per essere altamente personalizzabile, permettendo agli utenti di configurare le impostazioni in base alle proprie esigenze, e versatile, in quanto può

essere utilizzato su qualsiasi sistema operativo e sulla versione online di Microsoft Word; per fare ciò sono stati implementati dei controlli in modo da poter gestire l'interfaccia in base alla piattaforma sulla quale il plug-in viene utilizzato. Di conseguenza è stato necessario creare una documentazione chiara per gli utenti, con lo scopo di fornire le istruzioni su come poter installare il software sulla propria macchina.

Tutti questi aspetti, insieme al processo di sviluppo e alle varie funzionalità implementate, verranno approfonditi nel corso della tesi, fornendo una panoramica completa del progetto.

Capitolo 2

Contesto scientifico e tecnologico

Come già preannunciato nell'introduzione, questo progetto si trova all'interno di una sfida comunitaria volta a sviluppare un meccanismo per annotare documenti Word utilizzando vocabolari predefiniti, come ad esempio Akoma Ntoso.

2.1 Introduzione ad Akoma Ntoso

Akoma Ntoso è uno standard XML utilizzato per la rappresentazione strutturata di documenti legali e normativi. È stato sviluppato per facilitare la creazione, la gestione e l'interscambio di testi legislativi, consentendo l'automazione di processi legati alla produzione e alla pubblicazione di leggi, regolamenti e altri documenti giuridici.
Akoma Ntoso[2] fornisce uno schema standardizzato per la rappresentazione dei documenti giuridici, consentendo di organizzare il testo in modo gerarchico e dettagliato. Ciò include la definizione di sezioni, articoli, paragrafi, note e altre unità strutturali presenti nei testi legislativi. Permette inoltre un markup aggiuntivo dei documenti, creando la strut-

tura e i componenti semantici dei documenti legislativi digitali, permettendo loro di essere completamente machine-readable[1] e di poter far parte del Web Semantico.
Ad ogni documento sono, infatti, associate informazioni e dati aggiuntivi che ne specificano il contesto semantico.
Alla base di Akoma Ntoso troviamo il supporto alle seguenti funzionalità[3]:

- Orientamento giuridico-legislativo: lo standard fornisce una descrizione della struttura principale di documenti legali e legislati
- Affidamento a standard esistenti: Akoma Ntoso fa uso di standard efficiente e già largamente utilizzato come XML, URIs e RDF
- Ontologia: qualsiasi informazione che specifichi il contesto giuridico o istituzionale in cui un documento svolge un ruolo.

2.2 Approcci e Problemi

Per lo sviluppo di questo progetto ho esaminato ulteriori soluzioni precedenti implementate da altri programmatori per apprendere più approfonditamente il funzionamento interno di questi software. Questi tentativi, sebbene validi, mostravano molte limitazioni che ne hanno reso l'efficacia limitata: in presenza di documenti estesi o complessi, la gestione di stili può diventare un compito molto suscettibile di errori umani, oppure un cambiamento nella struttura del documento spesso richiede una revisione manuale approfondita rallentando il flusso del

[1] si intende che questo standard XML è progettato in modo che i computer possano analizzare e comprendere in modo automatico la struttura e il significato dei documenti legali formattati secondo tale standard senza la necessità di interpretazione umana

lavoro.

Dopo aver visto le limitazioni che presentavano gli altri software, abbiamo cercato di svilupparne uno molto efficiente, che non ricadesse in questi errori e che funzionasse in maniera praticamente perfetta.

2.3 Architettura e tecnologie utilizzate

Un plug-in progettato per Microsoft Word funziona in modo simile a qualsiasi altro tipo di plug-in. La sua struttura è solitamente ben definita e organizzata per garantire una corretta integrazione con l'applicazione ospite.
I componenti principali per il funzionamento di un plug-in sono l'interfaccia, utilizzata dagli utenti per interagire con esso, la logica di business, che si occupa del coordinamento del processo di esecuzione ed è responsabile dell'elaborazione dati e dell'interazione con il documento Word, ed infine è presente l'integrazione con il documento Word, in cui vengono utilizzate API messe a disposizione dall'applicazione madre per comunicare con essa.

Adesso andremo ad analizzare nello specifico le varie tecnologie utilizzate che ci permettono di svolgere correttamente i comportamenti appena elencati.

2.3.1 Funzionamento di un plug-in per Word

Come già anticipato nell'introduzione di questa sottosezione, per il corretto funzionamento del plug-in sono neccessari due elementi chiave: Of-

fice.js, che ci ha permesso di instaurare una comunicazione tra il plug-in e l'applicazione Word, e il file Manifest con il quale il plug-in viene distribuito agli utenti una volta terminato.

Office.js

Office.js rappresenta una libreria JavaScript fornita da Microsoft che offre una vasta gamma di metodi e API[4] (Application Programming Interface) per accedere in modo avanzato ai documenti, interagire con i dati e gestire gli eventi. Questa libreria si è dimostrata estremamente preziosa, soprattutto per la gestione degli eventi, grazie alla sua capacità di rispondere in tempo reale ai cambiamenti all'interno dei documenti.
La sua stretta compatibilità con React è stata fondamentale per la creazione di un prodotto finale di alta qualità, ed inoltre la grande versatilità del plug-in è dovuta all'amplificata compatibilità di Office.js con le diverse piattaforme. Ciò consente al nostro software di funzionare perfettamente in qualsiasi ambiente, offrendo un'esperienza uniforme e ottimale agli utenti.

Manifest.xml

Uno dei componenti principali all'interno dell'architettura di un add-in è il file di Manifest, ovvero un file in formato XML, che contiene le informazioni necessarie per registrare, distribuire e configurare l'add-in all'interno dell'applicazione.
Il file di manifest è suddiviso in diverse parti:

- **Metadata:** comprendono il nome dell'add-in, l'autore, la descrizione, l'icona e altre informazioni di identificazione. Questi det-

tagli aiutano gli utenti a capire cosa fa l'add-in e da chi è stato sviluppato.

- **Punti di Ingresso:** il manifest specifica i punti di ingresso dell'add-in, cioè URL o percorsi dei file che l'applicazione ospite deve caricare e visualizzare quando l'add-in è attivato. Questi punti di ingresso possono includere componenti dell'interfaccia utente e script JavaScript.
- **Autorizzazioni:** se un add-in richiede l'accesso a risorse o funzionalità specifiche dell'applicazione ospite, il manifest può essere utilizzato per specificare le autorizzazioni necessarie. Questo processo è fondamentale per garantire che l'add-in possa accedere solo alle risorse consentite, contribuendo così a migliorare la sicurezza. Nel nostro caso, abbiamo assegnato all'add-in il permesso *ReadWriteDocument*. Ciò significa che il plugin ha il permesso di leggere e modificare i documenti di Word. In situazioni diverse, ad esempio quando si sviluppa un'applicazione per uno smartphone, è possibile utilizzare il file di manifest per concedere autorizzazioni specifiche, come l'accesso alla posizione, alle notifiche, e così via, in base alle necessità dell'applicazione.
- **Configurazioni:** in alcuni casi, il file di manifest può contenere configurazioni specifiche per l'add-in, come le impostazioni predefinite o i parametri personalizzabili. Queste configurazioni influenzano il comportamento dell'add-in quando è attivo.

Il file manifest contenente il plug-in sviluppato si trova all'indirizzo link specificato qui[5].

Figura 2.1: Schema del funzionamento di un file manifest

2.3.2 Sviluppo di un plug-in per Word

Il plug-in sviluppato si basa su un approccio web-based, il che significa che è trattato come una pagina web e Word si occupa di integrarlo all'interno della sua applicazione.
L'approccio web-based ha semplificato il processo di distribuzione e di aggiornamento, in quanto si possono effettuare modifiche e cambiamenti all'interno del software senza richiedere un'ulteriore installazione all'utente.
Inoltre questo tipo di approccio mi era molto familiare grazie a progetti svolti in passato. Questa conoscenza del paradigma web-based mi ha

permesso di sfruttare tecnologie e competenze precedentemente acquisite in modo efficace. In particolare, sono state sfruttate tecnologie come NPM e Yeoman, che hanno semplificato la gestione delle dipendenze e la struttura del progetto, e per lo sviluppo dell'intero progetto abbiamo sfruttato la potenza di React integrato con la libreria MUI.
Per creare il primo add-in su cui lavorare abbiamo seguito la guida presente su Microsoft[6].

Node Package Manager (NPM)

Nel processo di definizione dell'architettura del mio software, ho sfruttato NPM (Node Package Manager) come strumento principale per gestire le varie dipendenze del progetto.
NPM è il gestore di pacchetti predefinito per l'ecosistema Node.js e JavaScript che consente di installare e gestire pacchetti software. Le dipendenze necessarie vengono definite nel file package.json, nel quale vengono anche inclusi i pacchetti utilizzati per lo sviluppo, le librerie di terze parti e gli strumenti di test. E' stato utilizzato per installare Yeoman Generator e per ottenere i diversi componenti come gli elementi della libreria MUI.

Inoltre NPM offre la possibilità di creare un ambiente di sviluppo grazie a comandi personalizzati definiti nel package.json. Questi comandi possono per esempio automatizzare la compilazione del codice, per l'esecuzione dei test e l'avvio del server di sviluppo.

Yeoman

Yeoman è uno strumento open-source e un framework che facilita la generazione automatica di codice e la creazione di strutture di progetto. Fornisce una serie di generatori predefiniti che permettono agli sviluppatori di iniziare nuovi progetti seguendo un modello predefinito e ottimizzato.

Un generatore che abbiamo utilizzato è Yeoman generator, ovvero uno script automatizzato, la cui funzione principale è quella di generare automaticamente la struttura di base di un progetto, insieme a file, configurazioni e altre risorse iniziali.
In sostanza, grazie a questo strumento è possibile risparmiare tempo, avviando dei progetti con una base già consolidata.

Durante la fase iniziale del progetto abbiamo avuto modo di sfruttare al massimo le potenzialità di questo tool; dopo aver completato l'installazione di Yeoman attraverso l'utilizzo di NPM, abbiamo installato anche uno Yeoman generator che ci ha permesso di ottenere un plug-in basilare ma funzionante da cui partire.

React e MUI

Per la realizzazione dell'interfaccia del mio progetto di tesi, ho fatto affidamento su React e MUI.

React[7] è un framework JavaScript sviluppato da Meta che si è affermato come una delle opzioni più popolari per la creazione di interfacce

dinamiche e reattive. L'utilizzo di React ha apportato numerosi vantaggi alla mia implementazione.

L'integrazione di MUI all'interno di React è stata una scelta cruciale per lo sviluppo dell'elaborato. MUI (Material-UI)[8] è una libreria JavaScript open source che offre componenti di interfaccia utente reattivi e stilizzati seguendo le linee guida del design Material Design di Google. La scelta di MUI è stata motivata dalla sua naturale sinergia con React; la libreria offre una vasta gamma di componenti UI predefiniti che si integrano perfettamente nelle applicazioni React. Inoltre, i componenti di MUI UI sono implementati nello stesso modo dei componenti React, il che significa che possono essere facilmente incorporati nell'architettura senza problemi di compatibilità.
Inizialmente, all'interno del progetto, i componenti erano sviluppati attraverso l'utilizzo di Fluent UI[9], un'altra libreria di componenti sviluppata da Microsoft. Tuttavia, durante lo sviluppo del progetto, ho notato che questa libreria era piuttosto limitata e aveva poca scelta tra i vari componenti, Di conseguenza ho trovato MUI, che risulta perfetta per React ed è adatta per lo sviluppo su quest'ambiente.

Una delle caratteristiche fondamentali di React è la sua architettura basata su componenti. Questa struttura consente di suddividere l'interfaccia in componenti riutilizzabili e autonomi, ciascuno responsabile della gestione del proprio stato. Questo approccio non solo rende più semplice la gestione di interfacce complesse, ma favorisce anche una chiara separazione dei compiti, migliorando la manutenibilità comples-

siva del progetto.

Un aspetto distintivo di React è l'uso della virtual DOM (Document Object Model). Questa rappresentazione leggera e virtuale del DOM reale consente a React di ottimizzare l'aggiornamento dell'interfaccia utente, calcolando solo le modifiche necessarie tra il nuovo stato e quello precedente, così da essere in grado di apportare aggiornamenti efficienti e mirati, riducendo al minimo l'impatto sulle prestazioni complessive dell'applicazione. Durante lo sviluppo del mio progetto, questa caratteristica si è dimostrata estremamente utile, soprattutto considerando la frequenza con cui sono state apportate modifiche all'interfaccia; quindi, avere un aggiornamento praticamente istantaneo ha sicuramente velocizzato il processo di sviluppo.

La reattività dei dati è un altro pilastro su cui si basa React, in quanto semplifica notevolmente il monitoraggio e la gestione delle modifiche ai dati, permettendo di aggiornare automaticamente l'interfaccia utente in risposta a modifiche nel documento o azioni compiute dall'utente, fornendo così un'esperienza fluida e intuitiva.

Tutte queste caratteristiche rendono React altamente compatibile con la libreria Office.js, offrendo una combinazione molto potente per lo sviluppo di applicazione nell'ambiente di Office.

Capitolo 3

Interfaccia e funzionamento di Pluginizer

In questo capitolo andremo a spiegare e ad analizzare nello specifico ogni aspetto dell'interfaccia utente e spiegheremo i vari meccanismi presenti dietro le diverse funzionalità dell'applicazione.

Esamineremo come gli utenti possano utilizzare tutte le funzionalità dell'add-in per creare e modificare documenti in maniera organizzata e strutturata, come sfruttare i nuovi stili implementati e come vengano gestite e memorizzate le diverse informazioni relative al testo.

3.1 Da Word ad Akoma Ntoso

Lo scopo principale di Pluginizer è consentire agli utenti di impiegare Microsoft Word per la modifica di documenti strutturati. Attraverso le funzionalità del plug-in, le modifiche verranno eseguite in totale sintonia con gli standard di Akoma Ntoso, come se il documento fosse stato

originariamente creato utilizzando questo standard.
Per un corretto uso di Akoma Ntoso, le regole da rispettare sono:

- **Attributi e metadati**: poter aggiungere delle informazioni al documento permette una migliore descrizione dello stesso. Quindi è importante includere gli attributi e i metadati richiesti per descrivere i dettagli del documento, come titolo, data, autore, fonte, e altre informazioni pertinenti. Questa parte è gestibile tramite la sezione "Documents" di Pluginizer
- **Struttura gerarchica**: assicurarsi che il documento abbia una struttura gerarchica ben definita, includendo sezioni, articoli, paragrafi e altre unità strutturali conformi allo standard Akoma Ntoso. Originariamente, all'interno di Pluginizer, era prevista una sezione denominata "Structures" incaricata di gestire queste informazioni, che purtroppo non siamo riusciti ad implementare.
- **Elementi semantici**: Utilizzare gli elementi semantici previsti da Akoma Ntoso per rappresentare in modo appropriato le informazioni contenute nel documento, ad esempio, utilizzare gli elementi corretti per definire titoli, riferimenti, citazioni e note. Questo compito è precisamente ciò che la sezione "Inlines" di Pluginizer si propone di realizzare: suddividere gli elementi del testo in stili e categorie distinti, consentendo al tempo stesso la memorizzazione delle relative informazioni.
- **Formattazione consistente**: mantenere una formattazione coerente per garantire che la struttura e l'aspetto del documento siano in conformità con gli standard di Akoma Ntoso.

3.2 Schermata Principale

Come possiamo osservare nella figura 3.1, in alto è presente il logo del plug-in insieme ad un pulsante denominato *About* che consente di accedere alle informazioni iniziali di presentazione. L'add-in è principalmente costituito da un menù a tendina noto come *Accordion* il quale fa parte dei componenti utilizzati in React.
Ogni sezione dell'Accordion è suddivisa in due parti:

- **Accordion Summary:** viene definito ciò che deve essere visualizzato quando la sezione è chiusa. Nel nostro caso abbiamo scelto di visualizzare il nome della sezione
- **Accordion Details:** viene definito il contenuto da visualizzare quando la sezione è espansa

Dalla figura possiamo notare le quattro sezioni principali.

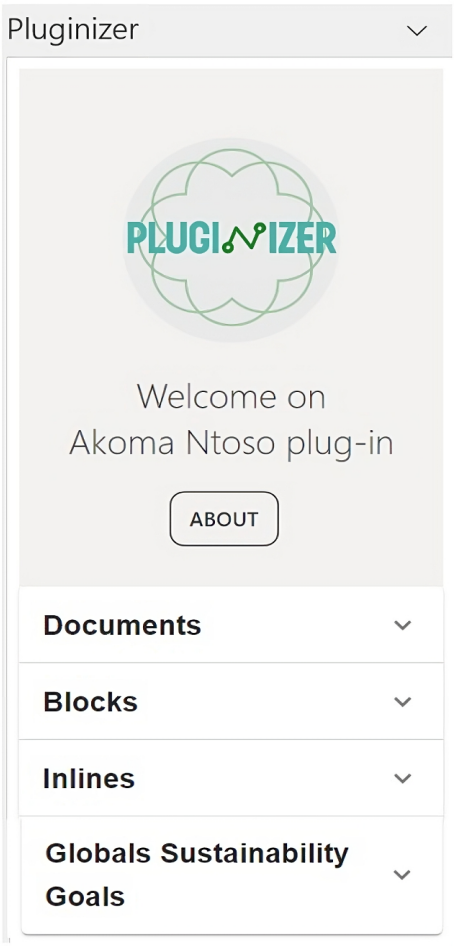


Figura 3.1: Schermata principale

3.3 Documents

All'interno di questa sezione, a differenza delle altre, sono presenti dei campi da compilare. La natura di questi campi varierà in base al tipo di documento che abbiamo creato.

L'obiettivo principale è migliorare la gestione e l'archiviazione dei vari documenti creati. Questo approccio ci consente di avere un controllo dettagliato su ogni fase del ciclo di vita dei documenti, garantendo che il

processo di creazione, revisione e pubblicazione sia svolto con precisione e chiarezza.
Le informazioni che verranno salvate aiuteranno gli utenti a identificare immediatamente il tipo di documento e le procedure da seguire, contribuendo così a una gestione più efficiente dei documenti.
Adesso guarderemo nel dettaglio che tipi di informazione è possibile salvare all'interno del documento.

Attraverso il campo *Document Type* è possibile identificare la categoria a cui appartiene il documento in fase di modifica. La sua compilazione è obbligatoria, pertanto abbiamo stabilito un valore predefinito nel caso in cui l'utente ometta di specificarlo. Le opzioni disponibili includono Decisions, Report, Agenda o Generic Document.

Il campo *Language* offre la possibilità di selezionare la lingua in cui è scritto il documento. La sua compilazione non è obbligatoria, ma può essere utile per chiarire la lingua di riferimento del testo.

Official Identifier è un campo opzionale, ma un elemento fondamentale per la condivisione e l'accessibilità. All'interno viene specificato l'URL ufficiale tramite il quale si può accedere direttamente al documento. Inoltre permette di condividere il documento con altri utenti in modo rapido e semplice.

L'inserimento del campo *Alias* è completamente facoltativo. È importante notare che è possibile avere più alias associati allo stesso documento. La finalità principale di questo campo è quella di consentire la creazione di parole chiave o riferimenti alternativi per identificare il documento in modo rapido e intuitivo. Ciò favorisce una migliore organizzazione e accessibilità ai documenti, semplificando la ricerca e la localizzazione di specifici contenuti in un contesto più ampio.

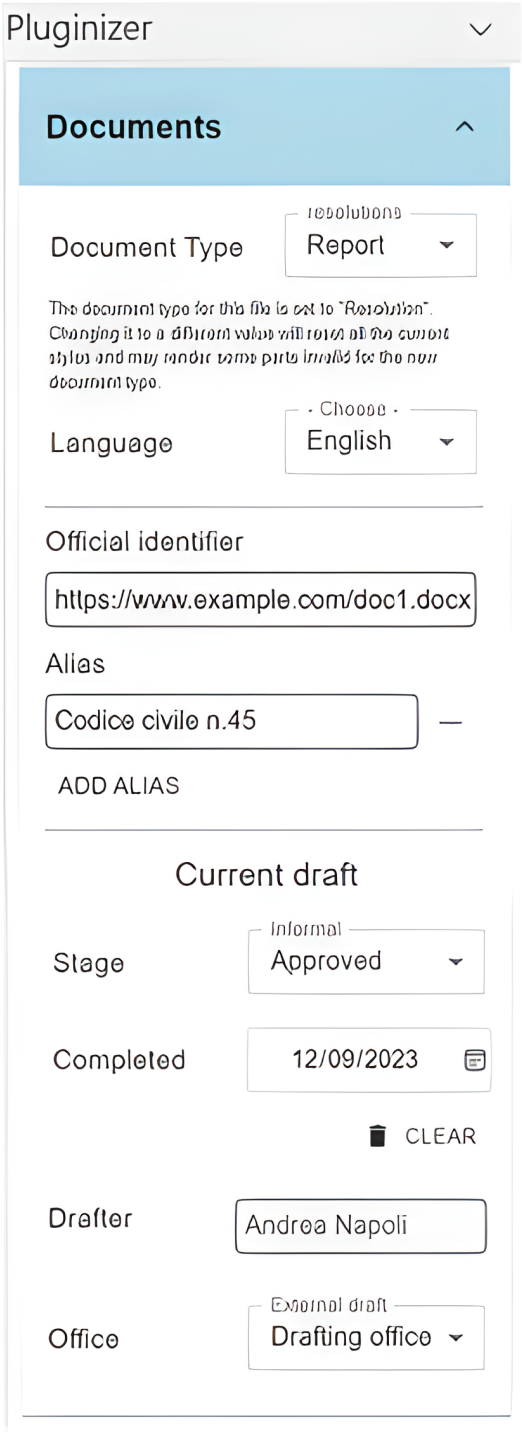


Figura 3.2: Schermata Documents

Il campo *Stage* è obbligatorio. E' possibile scegliere tra "First Draft",

se si sta creando una versione iniziale o una bozza, “Under revision”, se il documento è stato completato e attualmente è in fase di revisione e verifica, ed infine “Approved” per indicare che è stato approvato. E’ un’informazione fondamentale in quanto consente agli utenti, compresi quelli esterni che accedono al documento, di comprendere immediatamente lo stato in cui si trova il documento e quali procedure o azioni sono adatte.

L’aggiunta del campo *Completed* è facoltativo ed è utile per indicare la data in cui il documento è stato completato. E’ un’informazione importante se il documento deve rispettare determinate scadenze.
Il campo *Drafter* indica semplicemente l’autore che sta modificando attualmente il documento. E’ un campo che viene compilato in automatico prelevando il nome dall’account collegato e la sua modifica è disabilitata per l’utente.

Infine è presente il campo obbligatorio *Office* utile per specificare l’ufficio responsabile delle modifiche apportato al documento. E’ possibile scegliere tra "Drafting Office" e "Publication Office".

3.4 Blocks

La sezione Blocks è stata la prima ad essere stata sviluppata nel mio plug-in. È stata una scelta strategica in quanto offre funzionalità simili a quelle già presenti in Word, ma il suo scopo principale è quello di iniziare a prendere familiarità con l’ambiente e con i metodi principali offerti dalla libreria Office.js.

Le funzionalità implementate al suo interno sono:

- Allineamento del testo a destra, sinistra, al centro e testo giustificato.
- Creazione di liste numerate e non numerate.
- Spostamento del testo selezionato verso destra e sinistra.
- Possibilità di aumentare l'interlinea tra le varie righe del testo.

Uno dei principali obiettivi era quello di permettere agli utenti di modificare facilmente l'allineamento del testo nei loro documenti, cosa che Word permette di fare nativamente. Tuttavia, attraverso il plug-in, è possibile farlo in modo più personalizzato e flessibile.

Sviluppando questa sezione, ho avuto l'opportunità di apprendere i principali metodi e strumenti offerti dalla libreria Office.js, in particolare il metodo di selezione del testo. Questo mi ha permesso di acquisire una conoscenza fondamentale dell'ambiente di sviluppo e di mettere le basi per l'implementazione di ulteriori funzionalità più complesse all'interno del plug-in.

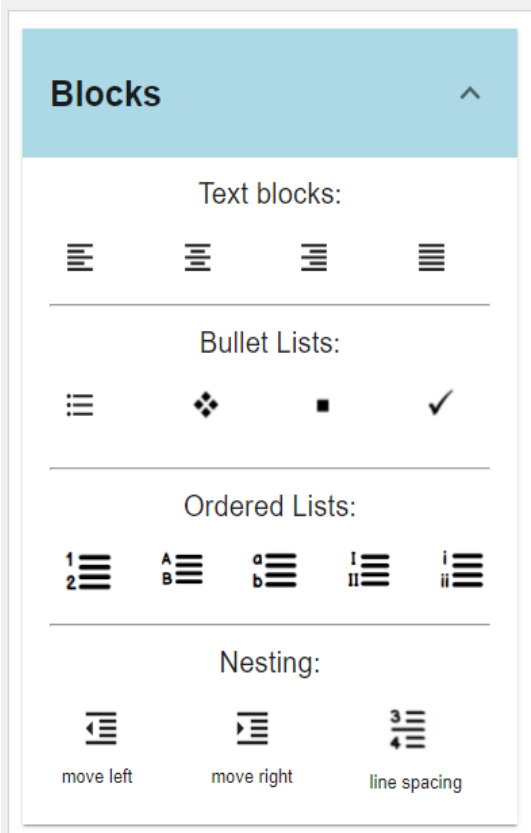


Figura 3.3: Schermata Blocks

3.5 Inlines

Questa sezione dell'add-in riveste un ruolo di primaria importanza, nonché quella che ha richiesto più impegno e tempo investito in quanto gestisce le funzionalità chiave.
A sua volta è formata da 4 parti che adesso andremo ad analizzare.

3.5.1 Information

E' un componente che non è sempre visibile all'interno della sezione. Quando il software identifica la presenza di testo selezionato, esso pre-

leverà in automatico il contenuto e lo mostrerà a schermo insieme alle informazioni associate. Se il testo non dispone ancora di informazioni, verrà comunque mostrato, ma senza alcuna definizione correlata.

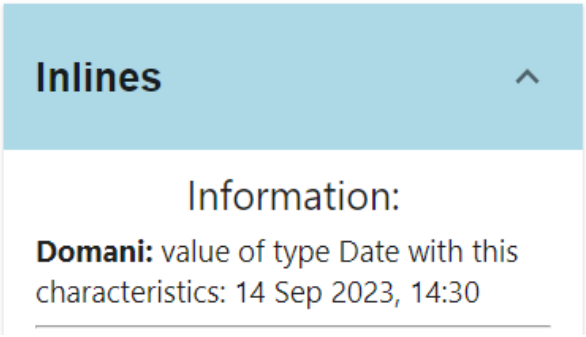


Figura 3.4: Schermata di Information

3.5.2 Default Style

In questa parte vengono impiegati gli stili preesistenti nell'applicazione Word per la formattazione del testo selezionato.

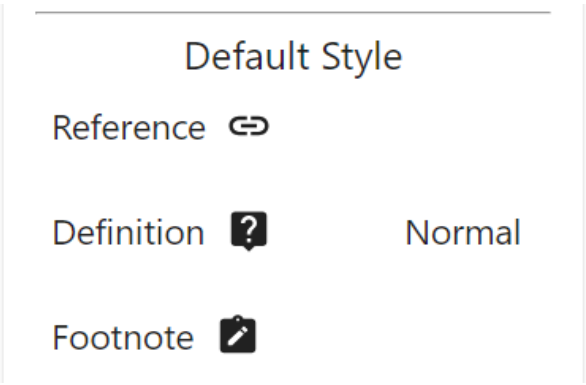


Figura 3.5: Schermata di Default Style

Come si può vedere dalla figura, sono disponibili 4 opzioni:

- **Reference:** Formatta il testo utilizzando lo stile "IntenseReference". Questo tipo di riferimento viene utilizzato per evidenziare al-

l'interno del documento gli elementi che si riferiscono allo stesso oggetto. Un esempio tipico potrebbe essere: "All'interno della costituzione verranno modificati solo gli articoli 13, 14, 17 e 19 della legge 574". In questo caso, gli articoli 13, 14, 17 e 19 fanno riferimento allo stesso oggetto, ossia la legge 574.
Reference offre tre diverse opzioni attraverso una finestra di dialogo:

- **Ref (Single Reference)**: utilizzato quando è presente un solo elemento che si riferisce a un oggetto. Ad esempio, quando è presente un unico articolo che fa riferimento a una legge.
- **MRef (Multiple Reference)**: utilizzato quando sono presenti più elementi che si riferiscono ad uno stesso oggetto. Un esempio potrebbe essere quello fatto in precedenza.
- **RRef (Riferimento consecutivo)**: usato quando sono presenti più elementi consecutivi che fanno riferimento allo stesso oggetto. Ad esempio, "dall'articolo 12 all'articolo 15 della legge 574".

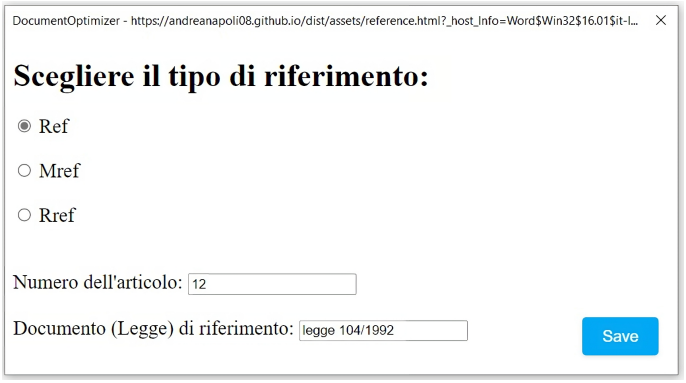


Figura 3.6: Reference di tipo Ref

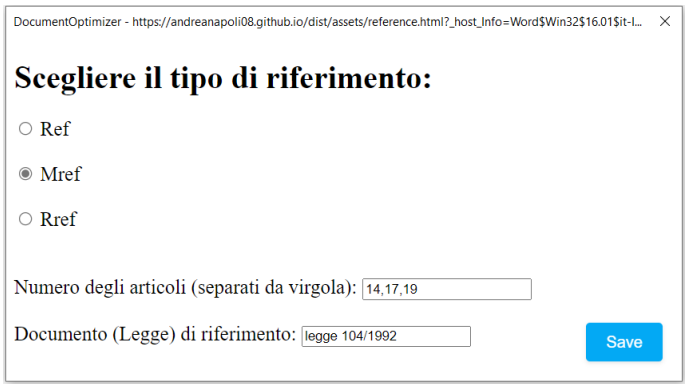


Figura 3.7: Reference di tipo MRef

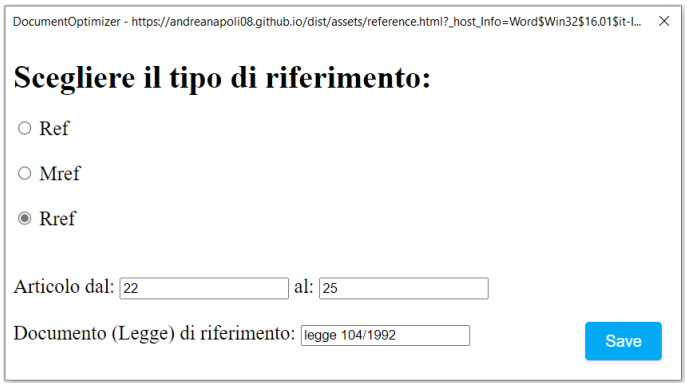


Figura 3.8: Reference di tipo RRef

- **Definition:** utilizza lo stile presente in word chiamato "Heading8". Come si può intuire dal nome, viene utilizzato per evidenziare nel documento tutte quelle frasi che spiegano dei concetti, forniscono definizioni oppure specificano un determinato concetto.
  L'utilizzo di questa formattazione non utilizza nessuna finestra di dialogo, in quanto l'utente non ha bisogno di inserire nessuna informazione aggiuntiva, poiché il suo unico obiettivo è quello di migliorare la leggibilità del documento strutturando ed evidenziando in maniera diversa i vari elementi.

- **Footnote:** utilizza lo stile "IntenseEmphasis. E' progettato per creare una footnote, o meglio nota a piè di pagina, all'interno del documento. La nota a piè di pagina ha lo scopo di fornire ulteriori informazioni o dettagli relativi a una parola, frase, pagina o un elemento specifico nel testo principale. È comunemente utilizzato per spiegazioni, citazioni o riferimenti.
  Quando viene applicato lo stile *Footnote* tramite il plug-in, si aprirà una finestra di dialogo che consentirà all'utente di inserire il testo che sarà collegato al contenuto selezionato nel documento. Questo testo aggiuntivo verrà quindi visualizzato in una nota in fondo alla pagina, permettendo ai lettori di accedere facilmente a ulteriori dettagli quando necessario.

Figura 3.9: Schermata di un Footnote

- **Normal:** questo comando permette di ripristinare rapidamente lo stile di formattazione predefinito, eliminando tutte le modifiche apportate dall'utente al testo. Inoltre, quando si applica lo stile *Normal*, è importante notare che qualsiasi informazione nascosta

nel testo verrà automaticamente cancellata. Questa funzionalità è particolarmente utile quando si desidera eliminare tutte le personalizzazioni e tornare al formato standard del documento, consentendo al contempo di conservare la possibilità di inserire ulteriori informazioni o modifiche in un secondo momento.

Di seguito presentiamo un'immagine che illustra un esempio pratico dell'applicazione degli stili precedentemente descritti all'interno di un documento. Questo esempio è stato creato per dimostrare come la formattazione e l'organizzazione del testo possano essere migliorati utilizzando gli stili all'interno del plug-in.

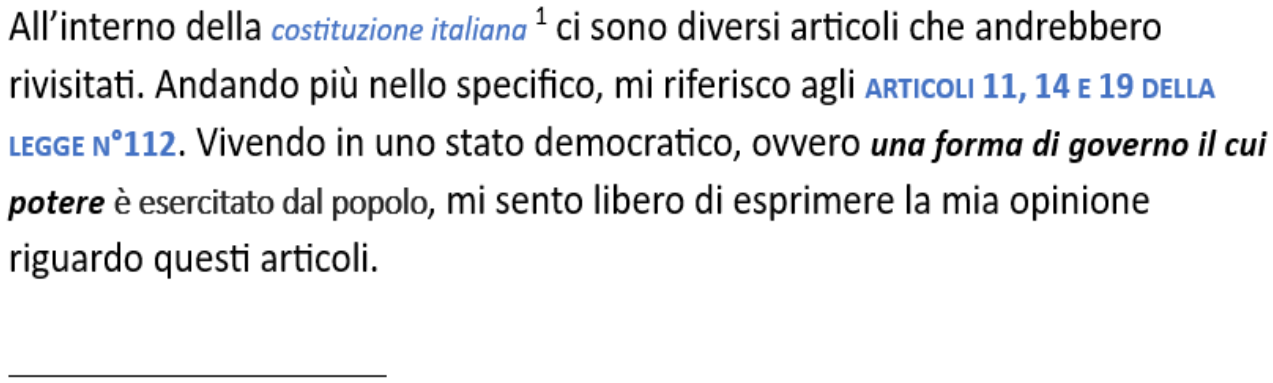


Figura 3.10: Esempio pratico di un testo

3.5.3 Entities

La porzione relativa alle *Entities* si distingue a causa dell'utilizzo esclusivo di stili personalizzati completamente innovativi all'interno dell'applicazione Word. Questi stili sono stati progettati con l'obiettivo di suddividere e organizzare in modo ancora più efficace il testo, offrendo

una struttura visuale chiara e intuitiva.

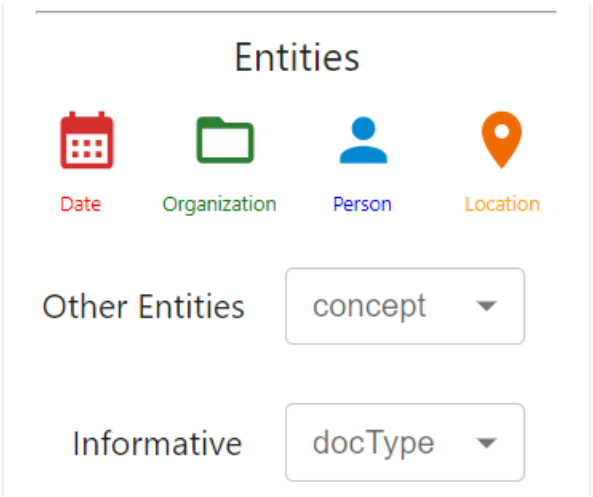


Figura 3.11: Schermata delle Entità

Questa sezione è ulteriormente suddivisa in tre sottosezioni distinte:

- **Important Entities:** in questa sottosezione, troviamo entità di grande rilevanza, tra cui Date, Organization, Person e Location. Queste rappresentano le categorie di entità più frequentemente utilizzate all'interno del documento. Per semplificare il processo di formattazione, è stato creato un apposito pulsante diretto per ciascuna di queste entità. Inoltre, è stata sviluppata una finestra di dialogo dedicata per ciascuna entità. Queste finestre di dialogo consentono all'utente di inserire facilmente nuove informazioni relative all'entità selezionata.
  - **Date:** permette di inserire il giorno e l'orario a cui si fa riferimento. L'inserimento del giorno è obbligatorio, mentre quello dell'orario no.

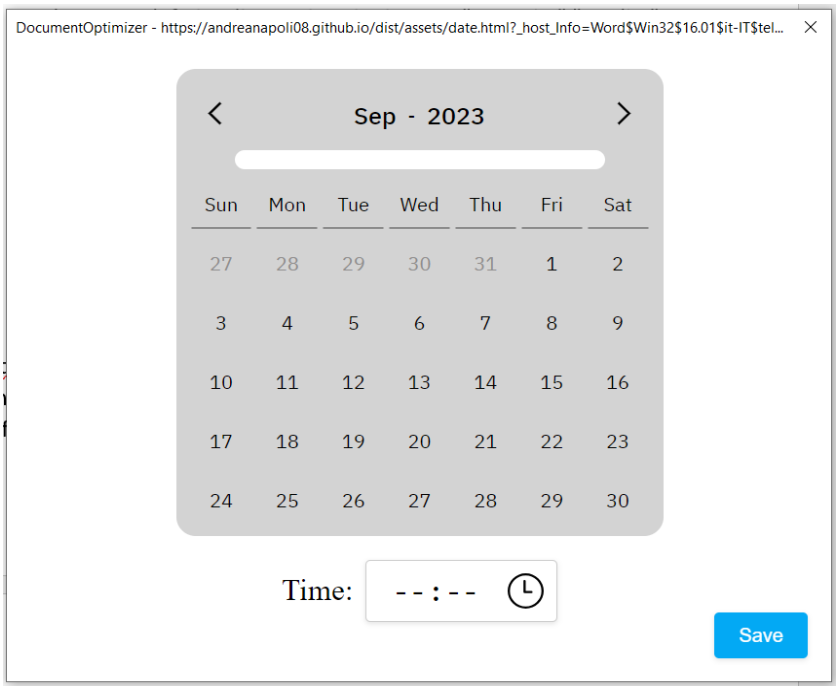


Figura 3.12: Finestra stile Date

– **Organization:** in questa finestra l'utente può scegliere tra un elenco predefinito di organizzazioni, come "Negozio," "Stadio," "Centro Commerciale," oppure è possibile inserire una propria organizzazione personalizzata utilizzando il campo di testo.

Figura 3.13: Finestra stile Organization

– **Person:** questa finestra funziona in modo simile a quella delle organizzazioni. L'utente può selezionare una persona da un elenco predefinito, come "Impiegato," "Direttore," "Maestro," o inserirne una nuova attraverso il campo di testo.

Figura 3.14: Finestra stile Organization

– **Location:** anche questa finestra offre la possibilità di scegliere tra un elenco predefinito di location, come "Torre di Pisa," "Colosseo," "Arena di Verona," o inserire una nuova location personalizzata attraverso il campo di testo.

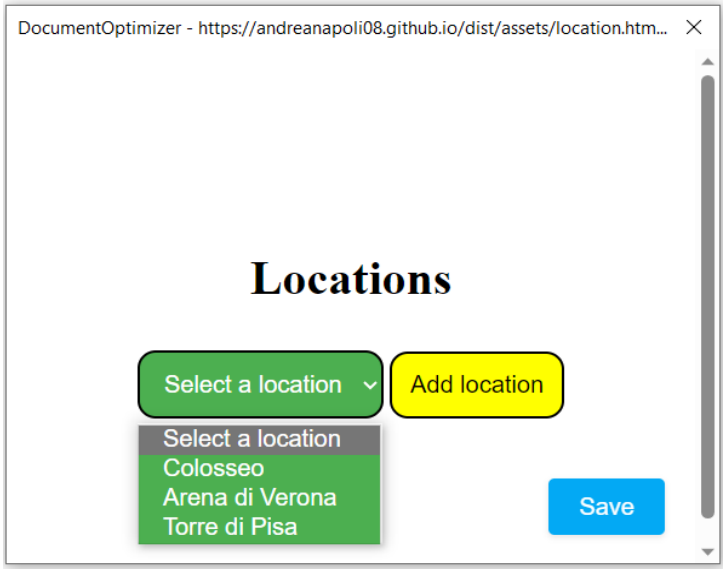


Figura 3.15: Finestra stile Organization

- **Other Entities:** Questa parte è dedicata alle entità secondarie, le quali, sebbene meno frequentemente utilizzate rispetto a quelle precedentemente elencate, rivestono un ruolo più specifico. Le entità incluse in questa categoria sono: Object, Event, Process, Role, Term e Quantity. L'obiettivo principale di questa sezione è quello di arricchire il testo con ulteriori dettagli organizzativi e, allo stesso tempo, di facilitare l'inserimento di collegamenti ipertestuali. Quando si seleziona uno stile, si aprirà una finestra di dialogo che consentirà di definire il link a cui il testo farà riferimento e di decidere se mantenere il testo originale selezionato o modificarlo. A seguire, è possibile visualizzare un esempio di finestra di dialogo

per questo tipo di formattazione che consente di inserire le informazioni necessarie (le finestre di queste entità sono tutte simili tra di loro).

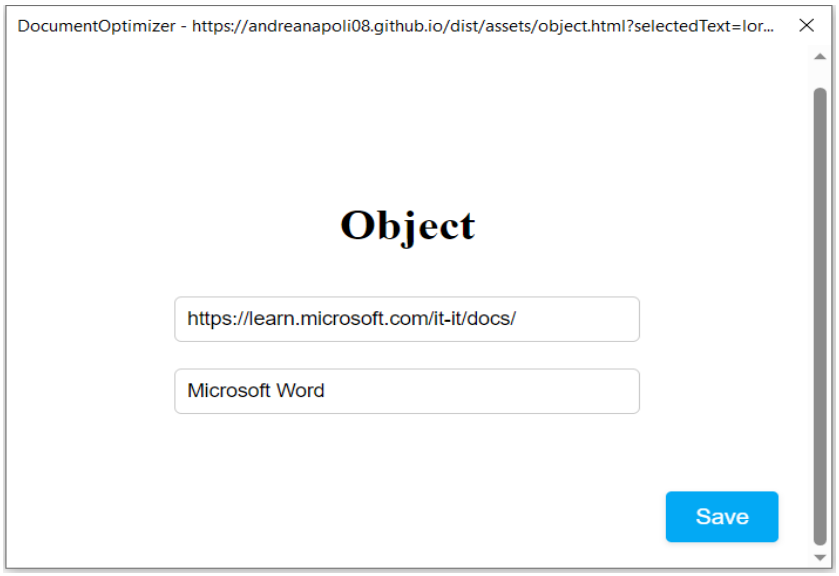


Figura 3.16: Finestra stile Object

- **Informative:** Questo campo ha un unico scopo, ovvero migliorare l'aspetto visivo del contenuto senza richiedere ulteriori informazioni dall'utente. La sua funzione principale è quella di suddividere ulteriormente gli elementi presenti nel testo. Le opzioni disponibili in questo settore sono varie e dipendono dalle preferenze dell'utente. Alcuni esempi includono: "docTitlte," "docDate," "shortTitle," "docStage," e molte altre. Questo stile è particolarmente utile quando le informazioni nel testo sono già chiare e non richiedono integrazioni, ma desideriamo comunque distinguere questa parte dal resto del testo.

Come abbiamo fatto per la sottosezione precedente, mostriamo un esem-

pio di testo in cui risultano utili gli stili appena elencati implementati all'interno del plug-in.

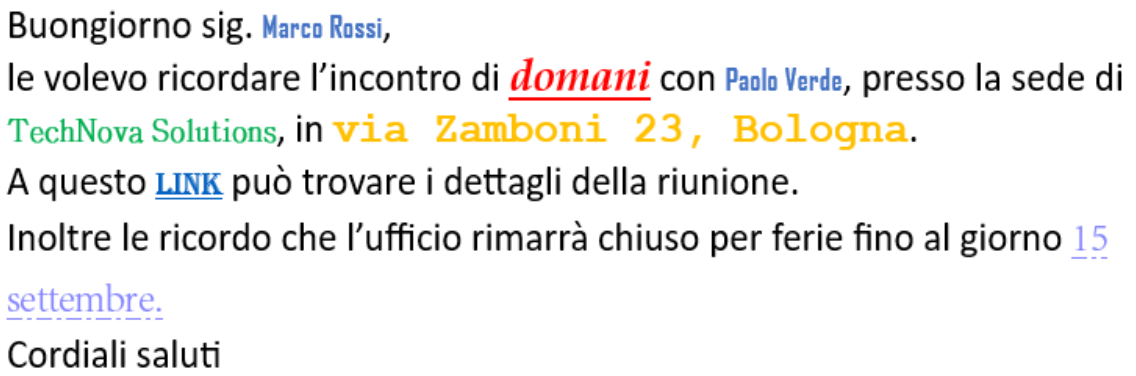


Figura 3.17: Esempio pratico di un testo con entità

In questo esempio possiamo trovare le seguenti entità:

- **Marco Rossi:** stile di tipo Person con caratteristica aggiuntiva "Impiegato"
- **Domani:** stile di tipo Date con caratteristica aggiuntiva: 7 settembre, ore 18
- **Paolo Verde:** stile di tipo Person con caratteristica aggiuntiva "Responsabile del settore tecnico"
- **TechNova Solutions:** stile di tipo Organization con caratteristica aggiuntiva "Azienda informatica"
- **Via Zamboni 23, Bologna** stile di tipo Location, con caratteristica aggiuntiva "Ufficio TechNova"
- **Link:** stile di tipo Event presente nelle entità secondarie con collegamento ipertestuale ad un file con i dettagli della riunione

- **15 settembre:** stile di tipo docDate presente nel campo Informative in quanto non ci sono informazioni da aggiungere

3.5.4 Tipography

In questa porzione del software è possibile svolgere funzionalità di tipografia che Word possiede già, infatti anche questa parte è stata sviluppata all'inizio per prendere familiarità con l'ambiente Office. In aggiunta abbiamo due funzionalità che invece non sono presenti nell'applicazione di Office:

- **Espansione della selezione:** se l'utente seleziona il testo tagliando le parole, cioè senza selezionarle completamente, e la funzione è abilitata attraverso la checkbox, il sistema automaticamente riconoscerà il testo completo da includere ed espanderà la selezione all'inizio e alla fine della parola. Di default questa funzione è abilitata.
- **Applicazione a tutte le istanze:** di base questa funzionalità è disabilitata. Quando l'utente attiva questa opzione e seleziona del testo, il sistema identificherà automaticamente tutte le occorrenze del testo selezionato all'interno del documento e applicherà la formattazione desiderata a ciascuna di esse.

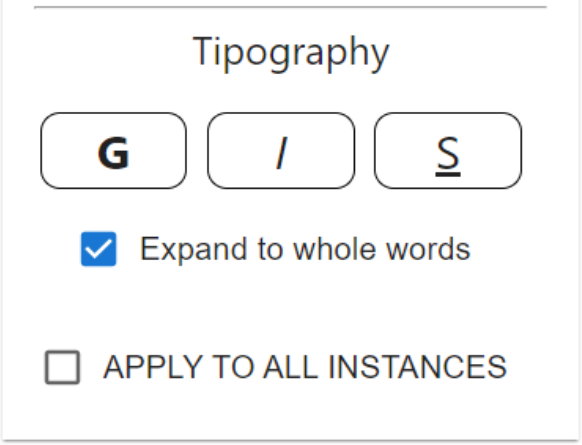


Figura 3.18: Schermata di Tipography

Nel capitolo successivo, illustreremo il funzionamento di entrambi gli algoritmi e spiegheremo come siamo riusciti a raggiungere una soluzione, considerando che la libreria Office.js non fornisce metodi diretti per implementare tali funzionalità.

3.6 Global Sustainability Goals (GSG)

L'ascesa della consapevolezza ambientale e sociale ha portato a una crescente attenzione verso la necessità di integrare i principi di sviluppo sostenibile nelle attività quotidiane, compreso il modo in cui creiamo e gestiamo documenti digitali. In risposta a questa esigenza, abbiamo deciso di dedicare una sezione del plug-in chiamata *Global Sustainability Goals*.

E' stata sviluppata con l'obiettivo di migliorare la struttura e l'organizzazione dei documenti, contribuendo così a promuovere una comunicazione più chiara e un'impronta ecologica ridotta.

I Global Sustainability Goals sono una serie di 17 obiettivi di sviluppo sostenibile stabiliti dalle Nazioni Unite. Questi obiettivi sono stati adottati nel settembre 2015 come parte dell'Agenda 2030 per lo Sviluppo Sostenibile. In questa sezione del plug-in, ciascun obiettivo è associato a uno stile unico che colora il testo selezionato con lo stesso colore dell'icona che rappresenta l'obiettivo corrispondente.

(a) GSG da 1 a 8 **(b)** GSG da 9 a 17

Figura 3.19: Schermate di GSG

Capitolo 4

Esplorando il codice di Pluginizer

In questo capitolo andremo ad analizzare tutti gli aspetti relativi al codice. Spiegheremo i dettagli e le scelte implementative adottate per sviluppare l'elaborato, come abbiamo deciso di strutturare il progetto e i vari problemi riscontrati con le rispettive soluzioni.

4.1 Struttura del codice e funzionamento interno

La struttura adottata per il progetto è abbastanza semplice: tutti gli elementi sono contenuti all'interno di taskpane, ovvero una cartella che è stata creata all'interno del progetto base grazie a yeoman generator.

All'interno di questa cartella, è presente un file denominato *taskpane.html*, che costituisce la pagina HTML principale per il pannello delle attività dell'add-in di Word. Questa pagina funge da contenitore per l'applicazione e definisce l'aspetto base dell'interfaccia utente. Quando l'utente aprirà l'add-in, Word si occuperà di caricare questa pagina nel pannello delle attività.

Tuttavia, al momento del caricamento, il contenuto HTML è statico e non interattivo. La vera interattività e la logica dell'applicazione entrano in gioco successivamente quando il file *index.tsx* viene eseguito all'interno di *taskpane.html* tramite l'inclusione di uno script.

Il file *index.tsx* è il nucleo del sistema ed è responsabile di tutte le operazioni preliminari e dell'inizializzazione del plug-in. Non appena viene eseguito, avvia l'applicazione React, tramite una funzione creata appositamente chiamata render() che prende il controllo del contenitore identificato dall'ID *container* all'interno di *taskpane.html*. Da questo punto in poi, l'applicazione prende il controllo del contenitore all'interno di *taskpane.html* e sarà completamente funzionale e può gestire le interazioni dell'utente, fornendo tutte le funzionalità previste per l'add-in.

All'interno di index.tsx ci sono tre funzioni principali che permettono il funzionamento e l'avviamento di tutta l'applicazione:

- `render()`: come già annunciato in precedenza, si occupa di reindirizzare un componente che gli viene passato come parametro, in questo caso verrà passato "App".
  Il lavoro principale lo svolge ReactDOM.render() che viene utilizzata per effettuare il rendering del componente *App* all'interno del DOM (Document Object Model) dell'HTML. Questa funzione richiede in ingresso due argomenti: il primo è il componente da reindirizzare, nel nostro caso "App", mentre il secondo contiene l'elemento del DOM in cui si desidera inserire il componente, e nel

nostro caso ci riferiamo al container creato appositamente all'interno del file taskpane.html.
Prima però di reindirizzare *App*, vengono utilizzati due wrapper.
Un wrapper è un componente o un elemento che avvolge un altro componente, consentendo di personalizzarne il comportamento senza modificarlo direttamente. Il suo scopo è quello di nascondere l'implementazione effettiva delle funzionalità e presentarle attraverso un'interfaccia ben definita.
I wrapper che vengono utilizzati sono:

- **AppContainer**: fornisce funzionalità di hot-reloader. La funzione di hot-reloader in React è una caratteristica che consente di apportare modifiche al codice durante lo sviluppo senza ricaricare l'intera applicazione. Quando si modifica un componente React, il hot-reloader aggiorna solo la parte interessata, accelerando il processo di sviluppo.
- **ThemeProvider:** viene utilizzato per applicare il tema di MUI all'interfaccia dell'add-in.

```
const render = (Component) => {
  ReactDOM.render(
    <AppContainer>
      <ThemeProvider>
        <Component
          title={title}
          isOfficeInitialized={isOfficeInitialized}
        />
      </ThemeProvider>
    </AppContainer>,
    document.getElementById("container")
  );
};
```

Figura 4.1: Codice di render()

- `Office.onReady()`: è fondamentale per determinare quando l'ambiente di Office è completamente avviato e pronto per l'interazione con l'add-in. Questo punto di ingresso consente di avviare in modo sicuro le operazioni legate a Office
- `Word.run()`: è stata la funzione principale all'interno del progetto e viene utilizzata da ogni componente. Serve per definire ed eseguire operazioni specifiche all'interno del documento Word, come l'accesso a contenuti del documento, la manipolazione delle proprietà del documento, la ricerca e la sostituzione di testo, e altro ancora.

Inoltre, alla fine del file index, c'è un controllo per l'hot-reloading, che permette di caricare dinamicamente i moduli dell'applicazione senza necessità di un aggiornamento manuale della pagina, semplificando il processo di sviluppo e debug dell'add-in.

Come anticipato all'inizio di questa sezione, è importante notare che i file del progetto risiedono all'interno della cartella denominata *taskpane*. Inoltre, abbiamo organizzato ulteriormente questa struttura creando una cartella separata chiamata *components*. All'interno di questa cartella è presente una serie di sottocartelle, ciascuna dedicata a una specifica sezione del plug-in. Ogni sottocartella contiene i vari componenti e sottocomponenti utilizzati nel contesto di quella sezione del software.

4.2 DOM (Document Object Model)

Il DOM di Word è una rappresentazione strutturata e gerarchica di un documento Microsoft Word all'interno dell'applicazione Word stessa. Questo modello consente di accedere, manipolare e automatizzare il contenuto, la formattazione e altre proprietà di un documento Word utilizzando JavaScript, tramite le API di Office.js.
Le API di Office.js modificano il DOM tramite chiamate e metodi JavaScript. Inizialmente, è necessario individuare gli elementi da modificare all'interno del DOM attraverso specifici metodi all'interno della funzione `Word.run()`. Una volta individuati gli elementi, è possibile apportare le modifiche desiderate utilizzando ulteriori metodi come quello di inserimento del testo.
Va sottolineato che, durante l'esecuzione di `Word.run()`, le modifiche

non diventano visibili direttamente nel documento Word, ma vengono conservate temporaneamente in un'area di staging dedicata.
Per rendere effettive le modifiche e sincronizzarle tra il DOM e il documento Word, è necessario invocare `context.sync()` alla conclusione di ciascuna funzione. Questa chiamata comunica con l'applicazione Word, estrae le modifiche dall'area temporanea e le applica al documento Word, rendendole visibili e interattive per l'utente.

Il DOM di HTML è una rappresentazione simile ma specifica per le pagine web HTML. React, essendo una libreria JavaScript per la creazione di interfacce utente, manipola il DOM di HTML per mostrare i suoi componenti all'interno del plug-in

In sostanza, il DOM di Word consente di manipolare e automatizzare i documenti Word, mentre il DOM di HTML permette di creare un'interfaccia utente interattiva per il plug-in e di consentire agli utenti di interagire con il documento Word attraverso di essa.

4.3 Deployment con GitHub Pages

Questa fase del progetto riveste un ruolo fondamentale in quanto consente a qualsiasi utente di accedere e utilizzare agevolmente il plug-in sul proprio dispositivo. Per fare ciò sono stati eseguiti una serie di passaggi.

Inizialmente, abbiamo avviato il processo di build del progetto mediante l'utilizzo della tecnologia npm e il comando `npm run build`.
Eseguire la build del progetto vuol dire preparare il codice sorgente e

trasformarlo in codice eseguibile o distribuibile.

Una volta completata questa operazione, all'interno del progetto viene creata la directory dist, la cui abbreviazione sta per "distribution", in cui sono presenti i file che sono stati preparati e ottimizzati durante il processo di build per essere distribuiti agli utenti finali o per essere utilizzati in un ambiente di produzione.

Il passo successivo ha coinvolto il caricamento del contenuto della directory *dist* su un servizio di hosting. La nostra scelta è stata GitHub Pages, un servizio di hosting web gratuito fornito da GitHub. GitHub Pages ci ha fornito l'opportunità di pubblicare il nostro sito web direttamente dalla repository GitHub, offrendoci un URL pubblico accessibile a chiunque.

A questo punto, all'utente non rimane che collegarsi all'URL designato (per questo progetto è https://andreanapoli08.github.io/dist/manifest.xml) e procedere con il download del file di manifest.

4.4 Importazione del plug-in

Come abbiamo già osservato nel capitolo 2, il plug-in viene importato nell'applicazione Word tramite il file Manifest.xml e ci sono diverse procedure da seguire in base alla piattaforma in cui desideriamo utilizzarlo.

Se ci troviamo su Microsoft Word Desktop in Windows bisogna svolgere i seguenti passaggi:

1. Spostare il Manifest all'interno di una cartella
2. Condividere la cartella in rete
3. Copiare il percorso di rete che permette di raggiungere la cartella
4. Aprire Microsoft Word Desktop
5. Andare su: File > Opzioni > Centro protezione > Impostazioni Centro protezione
6. Spostarsi all'interno della voce "Cataloghi di componenti aggiuntivi attendibili"
7. Inserire il percorso della cartella di rete nell'apposito campo di testo
8. Premere su aggiungi
9. Abilitare la funzione "Mostra nel menù"
10. Salvare e riavviare Word
11. Una volta riaperto Word premere su Inserisci > Miei componenti aggiuntivi
12. Selezionare cartella condivisa e selezionare il proprio plug-in

Se lo si vuole utilizzare su Word online la procedura è più semplice:

1. Aprire il proprio documento Word
2. Premere su Insert > Add-ins
3. Cliccare su "Upload my add-in"
4. Selezionare il file Manifest scaricato grazie al link

Infine se ci si trova su un dispositivo Macintosh:

1. Andare all'interno di questo percorso:

   `/Users/<username>/Library/Containers/com.microsoft.Word/Data/Documents/wef`

   Se la cartella wef non esiste, è necessario crearla
2. Copiare il file Manifest.xml all'interno della cartella wef
3. Riavviare Word e aprire il documento
4. Andare su Insert > My add-ins e nel menù a discesa dovrebbe comparire il plug-in
5. Selezionarlo e in automatico si aprirà nel pannello delle attività pronto per l'utilizzo

Capitolo 5

Problemi riscontrati e soluzioni

Durante lo sviluppo dell'elaborato, abbiamo riscontrato numerosi problemi, alcuni di piccola entità mentre altri di maggiore rilevanza. Questi problemi ci hanno obbligato a modificare le funzionalità che stavamo cercando di implementare, e in alcuni casi, a rinunciare a implementarle del tutto. È importante notare che tutti questi problemi derivano dalle limitazioni dei metodi disponibili all'interno della libreria Office.js, poiché non offre i metodi necessari per sviluppare le funzionalità previste in maniera diretta. In questa sezione, esamineremo dettagliatamente i principali problemi riscontrati e spiegheremo la soluzione adottata e il processo che ci ha portato a essa.

5.1 Creazione degli stili personalizzati

Il primo grande problema che abbiamo affrontato riguardava la creazione di stili personalizzati. Dopo giorni di ricerca, abbiamo avuto la conferma anche da Rick Kirkham, un utente che lavora all'interno della community di Microsoft, sostenendo che non esisteva ancora un modo

per creare stili personalizzati per Word utilizzando il codice e le API[10]. Questa funzionalità è fondamentale e non potevamo tralasciarla, quindi siamo stati costretti a trovare una soluzione.

Poiché Word consente la creazione di nuovi stili tramite la sua interfaccia, abbiamo deciso di creare un template[11] che comprendesse tutti gli stili necessari. In aggiunta abbiamo inserito una proprietà denominata *AKN template* con valore *vero*. Il documento è stato quindi salvato con l'estensione .dotx, ovvero l'estensione utilizzata da Word per i documenti di template.

Successivamente abbiamo convertito il file in base64 e lo abbiamo incluso nella directory del nostro progetto.
A questo punto, quando il plug-in viene avviato, il software esegue automaticamente una verifica: se il documento possiede la proprietà "AKN template" significa che stiamo già lavorando su un file con gli stili personalizzati. In caso contrario, l'add-in aprirà automaticamente il file di template utilizzando un metodo apposito fornito da Office.js e il contenuto del file che l'utente stava modificando verrà trasferito nel file di modello, prelevando il contenuto XML del vecchio documento e inserendolo in quello nuovo. Questo processo avviene in meno di un secondo, quindi l'utente noterà semplicemente l'apertura di un nuovo file che include i nuovi stili.

```
Word.run(async (context) => {
      var customDocProps = context.document.properties.customProperties;
      context.load(customDocProps);
      await context.sync();
      if(customDocProps.items.length == 0 ||
        !customDocProps.items[0].value ||
        customDocProps.items[0].key != "AKN Template"){
        if (Office.context.platform !== Office.PlatformType.OfficeOnline) {
          const myNewDoc = context.application.createDocument(templateDocument);
          context.load(myNewDoc);
          await context.sync();
          const body = context.document.body;
          const bodyXML = body.getOoxml();
          return context.sync().then(async function () {
              myNewDoc.body.insertOoxml(bodyXML.value, 'End');
              await myNewDoc.save();
              myNewDoc.open();
              await context.sync()
          });
        }
      }
    });
```

Figura 5.1: Inserimento file template

5.2 Espansione della selezione

L'espansione della selezione è una funzionalità che Word non possiede e che risulta molto utile per l'utente, in quanto, se è attivata tramite la checkbox, permette di espandere la selezione all'intera parola o frase. La complessità di questa funzione sta nel fatto che, secondo la documentazione di Microsoft, non esiste un metodo diretto che permetta di svol-

gere il comportamento appena descritto. Inoltre, anche sulla community di Stack Overflow, un utente ha chiesto come si potesse implementare una funzione simile, ma non ci sono state risposte positive[12].
La funzione che abbiamo implementato è molto articolata, in quanto durante lo sviluppo, sono sorti dei piccoli bug all'interno del documento che hanno richiesto ulteriori controlli. Ma adesso andiamo a spiegare nel dettaglio il suo funzionamento.

Iniziamo naturalmente verificando se la checkbox è abilitata, confermando così l'intenzione dell'utente di espandere la selezione.
Da qui la funzione si suddivide in due parti:

- **Espansione in avanti:** la selezione viene estesa verso la fine della parola tramite il metodo `selection.expandTo()`, che, come suggerisce il nome, si occupa di aggiornare il contenuto della selezione. Il metodo `selection.getTextRanges()`, invece, è utilizzato per determinare fino a dove estendere la selezione. Quest'ultimo metodo richiede un set di caratteri come parametro; noi includiamo tutti i segni di punteggiatura e lo spazio, i quali indicano quando arrestare l'espansione. È possibile specificare se includere o meno l'ultimo carattere.

  Come preannunciato, abbiamo dovuto aggiungere un ulteriore controllo, in quanto se nella selezione erano presenti dei paragrafi o righe vuote, l'espansione non funzionava correttamente. Quindi abbiamo controllato l'eventuale presenza di questi, per poi rimuoverli dal suo contenuto.

```
let text = selection.text;
let spaceCount = text.split(" ").length;
let paragraphCount = selection.paragraphs.items.length;
let emptyParagraph = 0;
for (let i = 0; i < paragraphCount; i++) {
    if (selection.paragraphs.items[i].text == "") {
        emptyParagraph++;
    }
}
const nextCharRanges = selection.getTextRanges([" ", ".", ",", ";",
 "!", "?", ":", "\n", "\r"], true);
nextCharRanges.load("items");
await context.sync();
if (nextCharRanges.items.length > 0) {
    if (paragraphCount > 1) {
        spaceCount = spaceCount + paragraphCount - 1 - emptyParagraph;
    }
    for (let i = 0; i < spaceCount; i++) {
        selection = selection.expandTo(nextCharRanges.items[i]);
    }
}
selection.load("text");
await context.sync();
```

Figura 5.2: Codice selezione in avanti

- **Espansione all'indietro:** questa parte del processo funziona in modo diverso, poiché i metodi descritti precedentemente possono essere utilizzati solo in avanti all'interno del testo[13]. In questo caso, torniamo all'inizio del paragrafo a cui appartiene la selezione e individuiamo tutte le parole che la precedono. Aggiungiamo quindi in essa l'ultima parola trovata, poiché questa rappresenterà la parola da estendere.

```
if (isLetterOrNumber(charBefore)) {
    let paragraph = selection.paragraphs.getFirst();
    paragraph.load("text");
    await context.sync();

    let rangeToSelect = paragraph.getRange("Start").expandTo(selection);
    let textBeforeSelection = rangeToSelect.getTextRanges([" ", ".", ",", ";"], true);
    textBeforeSelection.load("items");
    await context.sync();
    let lastItem = textBeforeSelection.items[textBeforeSelection.items.length - spaceCount];
    let rangeToExpand = lastItem.getRange("Start");
    selection = selection.expandToOrNullObject(rangeToExpand);
    await context.sync();
}
```

Figura 5.3: Codice selezione all'indietro

Una volta calcolata la selezione corretta, utilizziamo il metodo `selection.select()` per garantire una corretta espansione della selezione anche in maniera visiva.

5.3 Ricerca di tutte le istanze

La funzione *AllInstances* rappresenta la seconda opzione che gli utenti possono controllare tramite una checkbox. A causa di una scelta implementativa, che approfondirò dettagliatamente nella sezione 5.4, questa funzionalità può essere disattivata solo per gli attributi di tipografia, come il grassetto, il corsivo e il sottolineato. Mentre, per tutti gli altri stili, questa funzione rimarrà sempre attiva.

```
const range = context.document.body.getRange();
await context.sync();
const searchResults = range.search(selection.text,
    { matchCase: false, matchWholeWord: false });
searchResults.load("items");
await context.sync();
const occurrences = searchResults.items;
```

Figura 5.4: Ricerca di tutt le istanze

Adesso non resta che iterare su `occurrences` per applicare la formattazione desiderata.

5.4 Salvataggio delle informazioni

Il salvataggio delle informazioni relative al testo ha un ruolo cruciale, poiché consente all'utente di associare qualsiasi informazione necessaria al testo una volta che lo stile è stato selezionato. Inizialmente, avevamo considerato l'implementazione tramite l'uso dei campi di Word, ma successivamente abbiamo compreso che questa non era la soluzione ottimale. I campi di Word sono progettati per rendere il documento più dinamico rispetto ai cambiamenti e non sono adatti per il salvataggio di informazioni aggiuntive.
Dopo un'approfondita fase di studio, abbiamo scoperto che le XML Custom Parts rappresentavano l'opzione perfetta per realizzare questa funzionalità.

- `insertInformation()`: viene utilizzato per inserire una nuova informazione nascosta relativo al testo tramite il metodo addAsync() delle CustomParts. Prende in ingresso due parametri:
  - **context**: viene utilizzato per eseguire operazioni asincrone come `await()`
  - **xmlData**: è una stringa che rappresenta una parte di un documento XML contenente le informazioni da inserire. Un esempio di xmlData potrebbe essere il seguente:

```
const xmlData = `<root xmlns="${NAMESPACE_URI}">
                        <data id="${uniqueId}"
                            style="${entity}"
                            text="${selectedText.toLowerCase()}"
                        >
                            ${JSON.stringify(messageFromDialog)}
                        </data>
                    </root>`;
```

Figura 5.5: Esempio stringa xmlData

    - ∗ <**root**> è la radice del documento e viene identificata attraverso l'attributo xmlns
    - ∗ <**data**> è un elemento che serve per contenere i dati da salvare. E' distinto principalmente da due attributi: un id univoco per identificare l'informazione e l'attributo text che serve per associare l'informazione al testo. A volte può esse-

re presente un terzo attributo che specifica il tipo dell'entità di cui stiamo salvando l'informazione.

* all'interno del tag data, viene passato il contenuto dei dati da salvare in formato JSON.

```
const insertInformation = async (context, xmlData) => {
        Office.context.document.customXmlParts.addAsync(xmlData, (result) => {
            if (result.status === Office.AsyncResultStatus.Succeeded) {
                console.log("Dati personalizzati aggiunti con successo");
            } else {
                console.error("Errore durante l'aggiunta dei dati personalizzati");
            }
        });
        await context.sync();
    }
```

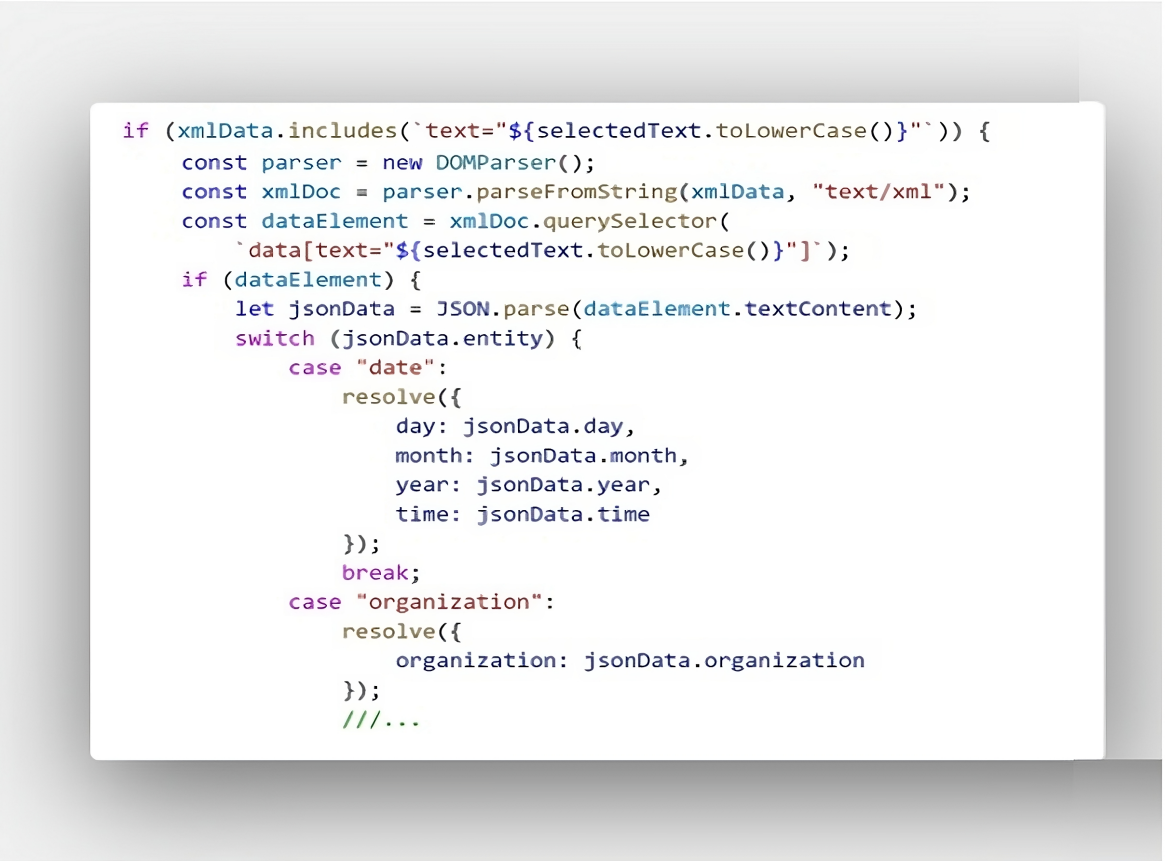
Figura 5.6: Funzione inserimento informazione

- `deleteInformation()`: è un metodo il cui scopo è eliminare un'informazione nel documento. Inizialmente vengono prelevate tutte le informazioni presenti all'interno del documento tramite il metodo `getByNamespaceAsync()`. Successivamente ricerca l'informazione in base al testo che viene passato come parametro. Una volta individuata, viene utilizzato il metodo `deleteAsync()` per rimuoverla.

```
const deleteInformation = async (context, NAMESPACE_URI, selectedText) => {
        Office.context.document.customXmlParts.getByNamespaceAsync(NAMESPACE_URI, (result) => {
            if (result.status === Office.AsyncResultStatus.Succeeded) {
                const xmlParts = result.value;
                for (const xmlPart of xmlParts) {
                    xmlPart.getXmlAsync(asyncResult => {
                        if (asyncResult.status === Office.AsyncResultStatus.Succeeded) {
                            const xmlData = asyncResult.value;
                            if (xmlData.includes(`text="${selectedText.toLowerCase()}"`)) {
                                xmlPart.deleteAsync();
                            }
                        } else {
                            console.error("Errore nel recupero dei contenuti personalizzati");
                        }
                    });
                }
            } else {
                console.error("Errore nel recupero dei contenuti personalizzati");
            }
        });
        await context.sync();
    }
```

Figura 5.7: Funzione eliminazione informazione

- `getInformation()`: è un metodo utilizzato per recuperare le informazioni associate a un testo specifico, che viene passato come parametro. Inizialmente esegue una ricerca equivalente a quella del metodo di eliminazione. Una volta individuata l'informazione, viene prelevato l'intero oggetto <data> associato all'informazione, quindi tutti i suoi attributi e il suo contenuto. L'oggetto ottenuto viene convertito in formato JSON ed infine sono presenti vari controlli per determinare a che tipo di entità si riferisce l'informazione per costruire in maniera corretta il JSON da ritornare contenente le informazioni.



```
if (xmlData.includes(`text="${selectedText.toLowerCase()}"`)) {
    const parser = new DOMParser();
    const xmlDoc = parser.parseFromString(xmlData, "text/xml");
    const dataElement = xmlDoc.querySelector(
        `data[text="${selectedText.toLowerCase()}"]`);
    if (dataElement) {
        let jsonData = JSON.parse(dataElement.textContent);
        switch (jsonData.entity) {
            case "date":
                resolve({
                    day: jsonData.day,
                    month: jsonData.month,
                    year: jsonData.year,
                    time: jsonData.time
                });
                break;
            case "organization":
                resolve({
                    organization: jsonData.organization
                });
                ///...
```

Figura 5.8: Funzione prelevamento informazione

Tuttavia questa funzionalità presenta una limitazione: poiché le informazioni vengono inserite associandole al testo selezionato dall'utente, non è possibile salvare informazioni diverse per la stessa parola. Ad esempio, se la parola "domani" appare due volte nel documento, è necessario assegnarle la stessa informazione, poiché si tratta della stessa parola. Da qui deriva la scelta implementativa di applicare lo stesso stile a tutte le occorrenze presenti all'interno del documento.

5.5 Differenze tra Word Desktop e Word Online

Durante tutto il processo di sviluppo, abbiamo effettuato test sia su Word Desktop che su Word online, e sono state proprio queste fasi di testing a evidenziare notevoli differenze tra le due piattaforme.
Una delle prime differenze emerse è la seguente: su Word online, non è possibile recuperare il nome dell'utente che sta apportando modifiche al documento, mentre su Word Desktop questa funzionalità è disponibile. Inoltre su Word Online non è possibile applicare la sottolineatura a parole che al momento dell'apertura del documento si trovano già al suo interno.

Tuttavia, la differenza più significativa riguarda la possibilità di creare un nuovo documento. Su Word online, tale operazione non è consentita a causa di restrizioni nelle API, come suggerisce l'errore all'interno della console quando il plug-in viene avviato, mentre su Word Desktop funziona perfettamente. Questa problematica ha comportato una significativa revisione della gestione del progetto. Abbiamo dovuto introdurre controlli specifici per la piattaforma su cui il plug-in è in esecuzione. Se l'utente utilizza Word Online, per le entità e i GSG utilizziamo gli stili predefiniti di Word, ma ne personalizziamo la formattazione (colore, dimensione del carattere, grassetto, ecc.). Invece, se si utilizza Word Desktop, utilizziamo normalmente gli stili personalizzati che abbiamo creato nel file di modello.

L'unico modo per applicare gli stili personalizzati su Word Online è caricare manualmente e aprire il file di modello, quindi spostare il con-

tenuto dal vecchio al nuovo documento. In altre parole, ciò che il plug-in effettua in modo completamente automatizzato su Word Desktop richiede un intervento manuale da parte dell'utente su Word Online.

Il motivo principale per cui le API possono funzionare in modo diverso su Word Desktop rispetto a Word Online è legato all'ambiente in cui ciascuna di queste versioni di Word opera. Word Desktop è un'applicazione autonoma installata sul tuo computer, mentre Word Online è una versione basata su web che funziona all'interno di un browser web.

Queste differenze architetturali hanno un impatto significativo sulla capacità delle API di eseguire determinate operazioni. Word Desktop ha accesso diretto al sistema operativo e alle risorse locali del computer, il che consente un'interazione più completa e veloce con le API. D'altra parte, Word Online è limitato dall'ambiente sandbox del browser, che impone restrizioni sulla sicurezza e sull'accesso alle risorse locali. Di conseguenza, alcune operazioni che potrebbero essere gestite facilmente da Word Desktop potrebbero risultare più complesse o limitate su Word Online.

Inoltre, Word Desktop è un'applicazione standalone con funzionalità avanzate, mentre Word Online è progettato per essere leggero e accessibile da qualsiasi luogo tramite il web. Questa differenza nella complessità delle due versioni influisce sulla disponibilità di alcune funzionalità e API avanzate.

Capitolo 6

Conclusioni

Al termine di questo progetto e dopo mesi di duro lavoro, posso affermare con grande soddisfazione di aver raggiunto i risultati sperati. Nonostante gli ostacoli e i numerosi problemi che si sono presentati lungo il percorso, ho sempre trovato una soluzione ottimale e funzionante.

Ci sono due realizzazioni di cui sono particolarmente fiero.
La prima è la funzione di espansione della selezione, che è stata un vero e proprio desiderio personale da realizzare in quanto si trattava di una skill che Word non possedeva. Poiché non esisteva un metodo predefinito per farlo, ho dedicato tempo ed energie per sviluppare un approccio efficiente, risolvendo con successo i vari bug che si sono manifestati durante il processo di sviluppo.

La seconda realizzazione di cui sono orgoglioso riguarda la gestione e la creazione degli stili personalizzati. Questa parte del progetto ha richiesto molti giorni di studio e ricerca, poiché inizialmente non sembrava

esserci una soluzione diretta. Tuttavia, abbiamo trovato una soluzione creativa attraverso l'utilizzo di un file di template, il che ha richiesto ulteriori sforzi ma alla fine si è dimostrata efficace.

Quindi sono particolarmente orgoglioso per queste due realizzazioni, per il tanto impegno e tempo dedicato, ma soprattutto per il risultato finale dell'elaborato, per com'è stato gestito e per come si è concluso.

Essendo un progetto abbastanza sostanzioso e lavorandoci da solo sicuramente sono presenti ancora dei bug che purtroppo non sono riuscito a risolvere completamente, come per esempio la mancanza di automazione nell'utilizzo del file di template sulla piattaforma online di Word, oppure migliorare la gestione delle informazioni multiple per lo stesso testo.
In un futuro spero di poter dedicare ulteriore tempo e risorse a questo progetto per poterlo migliorare ed espandere. So che un progetto simile è stato sviluppato in diverse parti del mondo ed è ancora in fase di sviluppo. Spero di aver dato il mio contributo per poterlo iniziare anche nel nostro paese e che magari un giorno venga distribuito a tutti gli utenti per poterlo utilizzare nella propria vita quotidiana. Sarebbe un grande onore sapere che un progetto da me iniziato è stato sviluppato con grande successo.

Bibliografia

[1] "Definizione akoma ntoso." https://it.wikipedia.org/wiki/Akoma_ntoso.

[2] F. Vitali and M. Palmirani, "Akoma ntoso: Flexibility and customization to meet different legal traditions," vol. 24, 2019. https://cris.unibo.it/retrieve/handle/11585/710681/517270/balisageVol24-2019.pdf.

[3] A. Zecca, "Sviluppo di un editor di metadati per simplex," 2022.

[4] "Microsoft word api documentation." https://learn.microsoft.com/en-us/javascript/api/word?view=word-js-preview.

[5] "File manifest of plug-in." https://andreanapoli08.github.io/manifest.xml.

[6] "Create first add-in." https://learn.microsoft.com/en-us/office/dev/add-ins/quickstarts/word-quickstart?tabs=yeomangenerator.

[7] "Documentation of react." https://react.dev/learn.

[8] "Mui react." https://mui.com/material-ui/getting-started.

[9] "Fluent ui react." https://react.fluentui.dev/?path=/docs/concepts-introduction--page.

[10] "Impossible create custom styles." https://stackoverflow.com/questions/68629625/how-to-select-the-whole-word-by-office-js.

[11] "Add template file." https://stackoverflow.com/questions/68629625/how-to-select-the-whole-word-by-office-js.

[12] "Problem with whole word." https://stackoverflow.com/questions/68629625/how-to-select-the-whole-word-by-office-js.

[13] "Problem with backward expansion." https://stackoverflow.com/questions/58357313/getting-a-ranges-surrounding-text-in-office-js.

Ringraziamenti

Dopo questi anni di studio e sacrifici sono giunto alla fine di questo percorso, un percorso che mi ha lasciato tanto a livello professionale ma che soprattutto mi ha aiutato a crescere e a maturare. In questi anni si sono susseguiti alti e bassi, ma grazie alle persone che sto per nominare, i momenti più duri non sono stati così difficili da affrontare.

Vorrei partire da una persona che c'è da quando sono al mondo, mio fratello, una persona a cui devo essere molto grato e che nei momenti di studio è riuscito sempre a strapparmi un sorriso (spesso perché lui non voleva studiare e non sapeva cosa fare); è riuscito a togliermi sempre i dubbi che avevo e a consigliarmi durante tutto il percorso, soprattutto dicendo come passare l'esame senza seguire le lezioni visto che lui è un esperto.

Ringrazio tutta la mia famiglia, i miei amici, la fiorefamily che è sempre stata pronta ad interessarsi e a supportarmi prima di ogni esame, ai miei nonni, che, anche se lontani, mi hanno sempre mostrato la loro vicinanza.

Non posso non ringraziare il mio migliore amico Manuel, la persona

con cui ho intrapreso l'università. Abbiamo condiviso tutto: libri, appunti, ma soprattutto progetti. In questi anni è sempre stato al mio fianco, ogni singolo giorno, soprattutto grazie alle chiamate su teams che in teoria dovevano durare solo 5 minuti. E' riuscito a farmi ridere anche nei momenti più intensi e difficili, ma anche ad illudermi per tutte le volte in cui ha urlato "Andre so io perché non va" e dopo puntualmente continuava a non andare. Ma nonostante questo non posso non essergli grato perchè senza di lui non ce l'avrei mai fatta.

Ringrazio la mia dolce metà, colei che c'è sempre stata in questi anni ed è riuscita a risollevarmi il morale ogni volta che ne avevo bisogno. La ringrazio per tutto l'impegno che mostrava per aiutarmi ad affrontare le varie problematiche che si presentavano e per tutte le volte che si è seduta al mio fianco per aiutarmi ad organizzare lo studio e a scegliere i corsi da seguire. Ma soprattutto la ringrazio per aver sopportato tutti i miei scleri ogni qualvolta un programma non andava, e vi garantisco che sono stati molti.

Per ultimo ho voluto lasciare le persone più importanti della mia vita, i miei genitori. Grazie a loro sono diventato l'uomo che vedete oggi. Ogni consiglio è stato prezioso per me; hanno saputo ascoltarmi e appoggiarmi in tutte le scelte che ho fatto e anche se alcune non erano condivise, hanno mostrato sempre il loro sostegno. Spero di avervi reso orgogliosi di me, proprio come io sono grato di avere voi come genitori.

Grazie.