Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica

Ristrutturazione ed estensione di un'applicazione web per l'annotazione di documenti pdf

Tesi di laurea triennale

Relatore

Prof. Luigi De Giovanni

Laureando

Stefano Rizzo

matricola: 1193464

Anno Accademico 2021-2022

Stefano Rizzo: *Ristrutturazione ed estensione di un'applicazione web per l'annotazione di documenti pdf* , Tesi di laurea triennale, © Luglio 2022.

Sommario

L'obiettivo principale dello stage è stato ristrutturare una parte del *front end* della piattaforma *RiskApp*, una web app che offre una serie di strumenti atti ad automatizzare e semplificare il lavoro degli intermediari del mondo assicurativo. In particolare, è stata eseguita la ristrutturazione di uno strumento dedito all'annotazione di polizze assicurative in formato PDF/A.
Questo strumento permette all'utilizzatore di RiskApp di caricare una *polizza assicurativa*, ovvero un documento in formato pdf, associato ad un cliente. Successivamente, questo documento potrà essere consultato ed annotato in modo da poter reperire rapidamente ed efficacemente le informazioni in esso contenute.
La ristrutturazione effettuata durante lo stage ha permesso di correggere e rivedere la parte logica e grafica del *front end* e, in alcuni casi, aggiornare o sostituire le tecnologie precedentemente adottate, ponendo particolare attenzione nel semplificare la *manutenibilità* e *l'usabilità* del prodotto, dato che dovrà essere usato anche da utenti non necessariamente familiari con tecnologie digitali.

Organizzazione del testo

Il primo capitolo descrive il contesto aziendale nel quale si è svolto il tirocinio.

Il secondo capitolo descrive il tirocinio descrivendone gli obiettivi.

Il terzo capitolo effettua una panoramica del tool di annotazione prima della ristrutturazione.

Il quarto capitolo descrive il lavoro svolto durante il tirocinio.

Il quinto capitolo effettua una retrospettiva sul periodo di stage.

Riguardo la stesura del testo, relativamente al documento sono state adottate le seguenti convenzioni tipografiche:

* gli acronimi, le abbreviazioni e i termini ambigui o di uso non comune menzionati vengono definiti nel glossario, situato alla fine del presente documento;
* per la prima occorrenza dei termini riportati nel glossario viene utilizzata la seguente notazione: *parola*[g];
* i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

"Life is really simple, but we insist on making it complicated"

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Luigi De Giovanni, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori Franca e Marco per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Luglio 2022 Stefano Rizzo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il contesto aziendale

Nel corso di questo capitolo viene presentata l'azienda in cui è stato svolto il progetto di stage e le metodologie di lavoro addottate.

1.1 L'azienda

RiskApp s.r.l. è una azienda del Padovano fondata nel 2016, che si occupa di sviluppo software per il mondo assicurativo. Il *core business* di RiskApp è lo sviluppo ed il mantenimento dell'omonima piattaforma, che viene costantemente aggiornata ed estesa con nuove funzionalità per meglio aderire alle necessità dei suoi clienti. Distribuita come Software as a Service[g]e pensata per intermediari del mondo assicurativo come agenti e broker, è finalizzata alla vendita di prodotti assicurativi nel settore delle piccole e medie imprese.
Il principale punto di forza di questa piattaforma è quello di stimare le possibili perdite economiche di un'impresa attraverso un algoritmo proprietario che, anche attraverso l'uso di intelligenza artificiale, valuta il rischio raccogliendo e combinando una moltitudine di dati da diverse fonti. Altre funzionalità messe a disposizione dalla piattaforma sono legate principalmente alla consulenza ed alla gestione dei clienti.
Ad oggi, RiskApp ha intrapreso numerose relazioni commerciali con importanti player del mondo della finanza a livello nazionale e internazionale come UnipolSai [46], Fidelidade [16], P&V [27] e molti altri, oltre alle numerose collaborazioni con università e acceleratori d'impresa [40].

1.2 Metodologie utilizzate

RiskApp è una realtà aziendale piccola e giovane: al momento dello svolgimento dello stage contava solamente sei dipendenti, ognuno dei quali incaricato di sviluppare, mantenere ed estendere una parte della piattaforma. Proprio visto l'esiguo quantitativo di personale, l'azienda mira ad avere un modo di lavorare che si avvicini il più possibile a quello *agile*[22], in modo da massimizzare il valore prodotto e minimizzare lo sforzo ed il rischio legato all'utilizzo di modelli di *project management* tradizionali.
Per integrare il modello di sviluppo *agile* nella propria metodologia di lavoro, RiskApp adotta le seguenti strategie:

* per pianificare il lavoro da fare e tenere sotto controllo quanto fatto, l'azienda adotta un approccio simile a *SCRUM*[41], di cui ne riprende i principi base quali trasparenza, controllo ed adattamento, ma permettendosi talvolta di non aderire strettamente al processo nella sua essenza più formale, per esempio, senza effettuare veri e propri *daily scrum*, ma prediligendo una comunicazione continua in ufficio o attraverso messaggistica istantanea, oppure attraverso riunioni meno frequenti (una o due volte a settimana) ma più approfondite, in contesti di lavoro a distanza o in modalità ibrida;

* per lo sviluppo del codice, RiskApp adotta un modello *iterativo*: le varie componenti del software vengono continuamente sviluppate e migliorate nel tempo. In questa maniera, il software può essere distribuito ai propri clienti in uno stato funzionante in tempi relativamente brevi (da uno a tre mesi) per poi migliorarlo ed estenderlo in base alle necessità riscontrate attraverso la comunicazione con i clienti;

* per versionare il codice e tener traccia dei cambiamenti, in RiskApp la codebase[g]è organizzata in due repository *GitHub*: uno per il *front end* ed uno per il *back end*. Viene utilizzata una strategia di *branching* di tipo *feature branch* per permettere la scrittura di software in maniera collaborativa e senza conflitti;

* per testare e distribuire il software, si segue un approccio di tipo Continuous Integration[g]e Continuous Delivery[g], in maniera da fornire ai propri clienti software testato e funzionante attraverso rilasci frequenti e continui. Per evitare interruzioni di servizio, inaccettabili in un contesto di produzione, la piattaforma RiskApp è *distribuita* su due server diversi: uno di sviluppo aggiornato diverse volte al giorno, ed uno di produzione, in cui viene rilasciato il software quando le funzionalità sono state interamente sviluppate e reputate mature;

* per mantenere un ambiente di lavoro produttivo ed efficace, all'interno dell'azienda si utilizzano diversi software, tra cui Gmail, Slack e Teams, rispettivamente per gestire posta elettronica, lo scambio di messaggi in maniera istantanea e per effettuare videochiamate. Un ulteriore software non strettamente legato allo sviluppo di codice che ho avuto modo di usare, è Figma: un tool utilizzato da designer di User Interface (UI) e grafici per la creazione di wireframe e mockup di applicazioni e siti web;

* per lo sviluppo del software, non sono state espresse preferenze riguardo all'Integrated Developement Enviroment (IDE)[g]. La maggior parte degli sviluppatori in azienda utilizza PyCharm [34] o Intellij Idea [20], utilizzati per lo sviluppo delle parti di *back end* in Python e in Java, rispettivamente. Questi due IDE prodotti da JetBrains sono senza dubbio tra i più popolari in commercio ed offrono un grande numero di funzionalità, in grado di aiutare lo sviluppatore ad effettuare *debug* e *refactoring* del codice. Inoltre, semplificano il controllo di versione integrando al loro interno i principali software di versionamento le cui funzionalità sono rese semplici ed intuitive attraverso l'uso di interfacce grafiche. Nel caso dello stage, dove è stato richiesto di sviluppare la parte di *front end* della piattaforma, si è preferito utilizzare Visual Studio Code, un *editor di testo open source*, che può essere esteso nelle sue funzionalità installando estensioni attraverso il suo market. In questa maniera è possibile personalizzare l'editor creando un ambiente di sviluppo in grado di rispondere perfettamente alle esigenze personali di ogni programmatore.

1.3 Tecnologie adottate

La piattaforma RiskApp è una web application, e adotta le tecnologie principalmente usate nello sviluppo web. Di seguito verranno elencate le tecnologie impiegate a livello di front end, tra cui:

* **CSS**: è il linguaggio utilizzato per definire lo *stile* ed il *layout* dei componenti della UI. In RiskApp si adotta la versione 3;
* **HTML**: è il *linguaggio di markup* che definisce le proprietà e i contenuti di una pagina web. In RiskApp si adotta la versione 5;
* **JavaScript**: è il linguaggio di programmazione con cui è scritto l'intero *front end* e parte del *back end*, attraverso Node.js. Risulta essere estremamente versatile, permettendo l'utilizzo di diversi *paradigmi* di programmazione, come quello ad *oggetti* (attraverso l'utilizzo di *prototipi*), ad *eventi*, *imperativo* e *dichiarativo* [21]. Al giorno d'oggi, è il linguaggio di programmazione più popolare secondo i report forniti da *GitHub Octoverse* [17]. In RiskApp si adotta la versione *ECMAscript 6*, che comprende numerosi aggiornamenti del linguaggio al fine di ottenere uno stile di scrittura del codice più *dichiarativo* possibile;
* **jsx**: è una estensione della *sintassi* di JavaScript che permette di definire elementi HTML al cui interno compaiono espressioni JavaScript. Viene ampiamente utilizzata in React per descrivere la UI;
* **Node.js**: fornisce un *web server open source e multipiattaforma*, in cui può essere eseguito codice JavaScript in maniera *asincrona*. Insieme a Node viene anche usato il suo packet manager *npm* per installare e gestire semplicemente le *dipendenze* di un progetto [25]. La combinazione di un web server open source particolarmente efficiente, che permette allo sviluppatore web di non dover imparare ulteriori linguaggi di programmazione per lo sviluppo back end, ed un packet manager completo e semplice da usare come npm, ha portato Node.js a diventare, secondo *StackOverflow* [44], la sesta tecnologia web più utilizzata nel 2021;
* **React**: è la libreria JavaScript adottata per la costruzione della UI. Si basa sull'uso dei componenti, ossia funzioni JavaScript che descrivono *logica* e *comportamento* di parti di interfaccia grafica che possono successivamente essere riusate e composte tra di loro. In React, si adotta un *paradigma* di programmazione di tipo *dichiarativo*, che aiuta a semplificare la scrittura e rendere prevedibile il mantenimento del codice [37];
* **Redux**: è un *framework* la cui funzione è quella di fare da *state container* in applicazioni di grandi dimensioni, in modo da gestire in maniera prevedibile lo *stato* dei componenti ed il comportamento dell'applicazione al cambio di quest'ultimo [39];
* **Webpack**: attraverso la creazione di un grafo delle dipendenze rappresentante le diverse parti di codice e le loro relazioni, permette di compilare un'applicazione JavaScript in una o più parti indipendenti chiamate *bundle o moduli*. In questa maniera si può eseguire code splitting[g]e fornire al client code on demand[g], migliorando le prestazioni [47].

Capitolo 2

Descrizione dello stage

Il capitolo presenta il progetto di stage e gli obiettivi, sia personali che aziendali, che questo progetto si prefissa di raggiungere. Il capitolo successivamente tratta la pianificazione effettuata per lo sviluppo del progetto e gli obiettivi identificati.

2.1 Motivazione

L'idea della proposta di stage è nata dalla necessità di ristrutturare il *front end* della piattaforma RiskApp, nello specifico, quella del tool che permette di visualizzare ed annotare le polizze assicurative. Questo tool, infatti, possedeva un'interfaccia grafica che non era stata attentamente progettata, rendendo il prodotto software non in linea con gli standard estetici e di usabilità richiesti da RiskApp. Un altro obiettivo del tirocinio è stato relativo alla ricerca di nuove tecnologie adottabili per la visualizzazione e l'annotazione della polizza assicurativa. La parte della piattaforma oggetto della ristrutturazione, infatti, è stata sviluppata inizialmente nel 2018 e non è mai stata integrata in RiskApp in quanto è stato preferito concentrare lo sviluppo su altre funzionalità, reputate più importanti dal *business*.
Ripreso lo sviluppo del tool solo recentemente, si è sollevata la necessità di approfondire se le tecnologie usate per visualizzare ed annotare la polizza fossero ancora adatte allo scopo oppure, se potessero essere aggiornate e/o rimpiazzate con altre in grado di rendere più agevole lo sviluppo ed il mantenimento del software.

2.2 Obiettivi

2.2.1 Obiettivi aziendali

La ristrutturazione del tool di annotazione delle polizze permetterebbe a RiskApp di ottenere un'interfaccia utente più semplice ed efficace, introducendo così nella sua piattaforma una nuova funzionalità, al giorno d'oggi già a buon punto nello sviluppo, capace di portare nuovo valore al prodotto software.
Inoltre, dal punto di vista del team di sviluppo, la ristrutturazione del codice e l'eventuale cambio di tecnologia porterebbe ad un più semplice mantenimento della codebase, oltre che ad un incremento in prestazioni ed ad una riduzione di bug.

2.2.2 Obiettivi personali

Ho scelto di svolgere il tirocinio in RiskApp per diversi motivi.
Innanzitutto, lo sviluppo web è un settore dell'informatica che ho sempre considerato interessante e della quale in futuro mi piacerebbe approfondire diversi aspetti.
Le tecnologie, come *JavaScript* e *React*, richieste dal progetto di stage sono tecnologie che volevo approfondire e studiare da tempo, e lo stage non solo mi ha dato la possibilità di farlo, ma mi ha anche dato modo di vederle usate in contesti diversi rispetto a quanto già visto, per esempio durante il corso di Tecnologie Web, e più complessi rispetto a quanto avrei potuto affrontare attraverso lo studio autonomo.
In secondo luogo, fin dal primo colloquio avvenuto attraverso l'evento *Stage-IT* con il tutor aziendale, ho avuto l'idea che l'azienda fosse una realtà giovane e dinamica, in cui fosse facile inserirsi.

2.3 Pianificazione del progetto

Precedentemente all'inizio del tirocinio sono stati effettuati diversi incontri con i responsabili di RiskApp, con cui, a grandi linee, sono state definite le richieste relative allo sviluppo del progetto. Grazie a questi incontri, è stato possibile capire quali erano le aspettative da parte dell'azienda rispetto al progetto di stage in modo da poterle assecondare correttamente. Inoltre, sono state date informazioni circa le tecnologie utilizzate nella piattaforma, in modo da poter organizzare la quantità di studio in maniera autonoma.
Fissati gli obiettivi, il lavoro è stato organizzato in circa 320 ore di tirocinio, suddivise in otto settimane da 40 ore lavorative.
Il periodo di stage comprende, suddiviso a grandi linee, tre periodi:

* **primo periodo (durata tre settimane)**: studio autonomo delle tecnologie utilizzate e dell'infrastruttura del software preesistente. Ricerca e studio di fattibilità circa l'adozione di nuove librerie da utilizzare per la visualizzazione e l'annotazione di documenti pdf;

* **secondo periodo (durata due settimane)**: ristrutturazione del codice esistente;

* **terzo periodo (durata tre settimane)**: progettazione ed implementazione della nuova UI.

Questa pianificazione, insieme ad un continuo confronto con il tutor aziendale, si è rivelata essere efficace, in quanto, come dettagliato al Capitolo 5, durante lo svolgimento del tirocinio non si sono verificati né slittamenti della pianificazione né revisioni dei requisiti individuati.
Nella Tabella 2.1 viene esposta la pianificazione settimanale come definita nel Piano di Lavoro.
Nella Tabella 2.2 viene riportata la ripartizione oraria per attività, come definita nel Piano di Lavoro.

Ore settimanali	Attività svolte
Prima settimana - 40 ore	Incontro con persone coinvolte nel progetto per discutere i requisiti e le richieste relativamente al sistema da sviluppare; verifica credenziali e strumenti di lavoro assegnati; presa visione dell'infrastruttura esistente; inizio della formazione su React e Redux.
Seconda settimana - 40 ore	Studio dell'infrastruttura esistente; studio di React e Redux.
Terza settimana - 40 ore	Ricerca di eventuali nuove librerie in grado di apportare miglioramenti all'annotazione e visualizzazione dei file di tipo pdf; studio delle nuove librerie; studio di React e Redux.
Quarta settimana - 40 ore	Implementazione delle nuove librerie; revisione ed implementazione di modifiche alle componenti che gestiscono le annotazioni.
Quinta settimana - 40 ore	Implementazione delle modifiche alle componenti del frontend; definizione e scrittura dei test; produzione documentazione relativa alle modifiche apportate.
Sesta settimana - 40 ore	Revisione della UI; produzione dei wireframe relativi alla nuova UI; inizio del refactoring dei componenti della UI.
Settima settimana - 40 ore	Implementazione delle modifiche ai componenti della UI; progettazione e sviluppo dei componenti necessari per implementare la shortcut relativa alla funzionalità di annotazione.
Ottava settimana - 40 ore	Sviluppo dei test relativi ai componenti oggetto di refactoring o scritti ex-novo; produzione documentazione relative alle modifiche apportate alla UI ed ai componenti aggiunti; collaudo e verifica del lavoro prodotto.

Tabella 2.1: Piano di lavoro.

Durata in ore	Descrizione dell'attività
72	**Formazione**
18 18 36	Formazione React Formazione Redux Ricerca e studio di nuove librerie per la manipolazione dei documenti pdf
64	**Analisi**
16 16 16 10 6	Analisi del software preesistente Progettazione delle modifiche ai componenti preesistenti Progettazione della nuova UI Progettazione delle nuove funzionalità Stesura documentazione relativa ad analisi e progettazione
146	**Refactoring**
72 64 10	Implementazione modifiche ai componenti revisione della UI produzione dei test
146	**Collaudo finale e attività conclusive**
38 64 10	Collaudo Verifica documentazione finale Presentazione della piattaforma agli stakeholders e live demo
Totale ore	**320**

Tabella 2.2: Ripartizione Oraria.

2.4 Obiettivi identificati

Insieme al tutor aziendale sono stati concordati una serie di obiettivi a cui si farà riferimento secondo la seguente notazione:

OB[**classificazione**][**numero incrementale**]

Per **classificazione** dell'obiettivo si intende:

- **O**: obbligatoro;
- **D**: desiderabile;
- **F**: facoltativo.

Le sigle precedentemente indicate saranno seguite da una coppia sequenziale di numeri, identificativo dell'obiettivo.
Si prevede lo svolgimento degli obiettivi indicati in Tabella 2.3.

Obiettivi	Descrizione
O01	Le componenti che gestiscono le annotazioni devono essere ristrutturate per funzionare con il nuovo schema dati adottato nel backend;
O02	Le componenti ristrutturate devono essere riscritte utilizzando sintassi e buone pratiche che ne semplifichino l'implementazione e ne migliorino leggibilità e manutenibilità;
O03	Devono essere riviste collocazione e aspetto visivo della toolbar per la navigazione del pdf con le varie annotazioni;
O04	Le modifiche effettuate alla UI devono mantenerla responsiva, accessibile e semplice da utilizzare per l'utilizzatore finale;
D01	Deve essere aggiunta una shortcut per l'attivazione della funzionalità di annotazione del pdf;
D02	Devono essere riviste le librerie adottate per la gestione dei pdf, se ne conseguono benefici a livello di prestazioni e/o di manutenibilità del software;
D03	Deve essere fornita documentazione delle scelte implementative adottate nel refactoring di componenti e UI;
F01	devono essere presenti dei test a livello di frontend.

Tabella 2.3: Tabella di tracciamento degli obiettivi.

Capitolo 3

Panoramica del tool di annotazione esistente

Dal momento che lo scopo del progetto di tirocinio è stato ristrutturare un'applicazione preesistente, prima di procedere con il resoconto del lavoro svolto, è necessario fare una panoramica del software al momento dell'inizio dello stage. Verranno quindi illustrate le principali funzionalità del tool di annotazione, come sia stato originariamente implementato e le problematiche ad esso legate.

3.1 Le funzionalità del tool di annotazione

Le funzionalità previste dal tool possono essere suddivise in due macrocategorie: consultazione ed annotazione.

3.1.1 Consultazione

Dal momento che la polizza assicurativa è un documento in formato pdf, l'utente deve poterlo visualizzare e navigare agevolmente, in maniera del tutto analoga a come farebbe con un qualunque visualizzatore di documenti pdf. Deve quindi poter accedere direttamente ad una data pagina del documento, *zoomare* ed effettuare *scrolling verticale* in modo da poterlo consultare nella sua interezza.
Inoltre, un'altra funzionalità chiave di questo strumento è la funzione di *sintesi*: non è un raro caso che le polizze assicurative siano documenti da molte decine di pagine. Questa funzione permette all'utente, attraverso un click, di sintetizzare in forma tabellare l'intero documento, grazie ad un sistema basato sull'intelligenza artificiale, in grado di leggere la polizza ed estrarne automaticamente i punti chiave. Durante il tirocinio, questa funzionalità non era ancora stata completamente sviluppata da RiskApp a livello *back end* ed è stata quindi implementata nel *front end* dell'applicativo attraverso un mockup.

3.1.2 Annotazione

Il tool deve permettere all'utente di selezionare, attraverso l'uso del mouse, *aree* o *parti di testo* del documento, ed associarle ad un insieme di dati che ne definiscono le proprietà. In particolare, sono presenti due diverse modalità di annotazione: l'annotazione *normale* e quella *rapida*. In particolare,

* nell'*annotazione normale*, l'utente seleziona del testo o un'area del documento. Al termine della selezione, attraverso la comparsa di un *pop-up*, può selezionare le proprietà dell'annotazione per poi salvarla;

* nell'*annotazione rapida*, l'utente imposta a *priori* le proprietà dell'annotazione, per poi proseguire con la selezione di un'area o di testo. Terminata la selezione, è possibile salvare l'annotazione effettuata attraverso la comparsa di un tooltip[g]. In questa maniera, si riduce il numero di volte in cui l'utente deve impostare i parametri delle annotazioni, velocizzando la procedura di annotazione, nel frequente caso in cui ci siano da annotare più parti che condividono le stesse proprietà. È possibile, inoltre, resettare le proprietà precedentemente impostate.

La selezione del testo salva nell'annotazione anche il testo selezionato, mentre la selezione di un'area salva uno *screenshot* dell'area selezionata. Al salvataggio di un'annotazione, l'area o il testo annotato si evidenziano, ed attraverso un click è possibile aggiungere un commento all'annotazione oppure modificarne le proprietà. Inoltre, una volta salvata un'annotazione, questa viene aggiunta al pannello delle annotazioni, in modo da potervi accedere rapidamente. Fondamentale è anche la possibilità di *filtrare* le annotazioni in base alle loro proprietà, per poter effettuare semplici azioni di ricerca. È quindi importante stabilire quali siano le proprietà che vengono associate alle annotazioni, in quanto permetterà di definire una terminologia che verrà usata più volte nel futuro uso dell'applicazione. Questi parametri sono: *categoria dell'annotazione*, *tipo di sezione* ed *unità di rischio*.
Verranno ora spiegati nel dettaglio i ruoli di tali informazioni:

* **tipo di sezione**: in ogni polizza assicurativa, esistono diverse sezioni in cui vengono definite determinate clausole. RiskApp ne ha individuate sei e sono: condizioni aggiuntive, condizioni generali, condizioni speciali, definizioni, esclusioni e garanzie;

* **categoria dell'annotazione**: in una polizza assicurativa, esistono diverse categorie di clausole. RiskApp ne ha identificate cinque e sono: limite, massimale, partita, scoperto e wording. Ognuna di queste è associata ad un colore, che definisce il colore dell'annotazione in modo che sia facile per l'utente distinguerle immediatamente a colpo d'occhio;

* **unità di rischio**: sono i diversi tipi di rischio su cui vengono definite le clausole. Ogni unità di rischio è associata ad una categoria di rischio. Per esempio, l'unità di rischio "Atti vandalici dei ladri" oggetto di una particolare clausola della polizza, è associata alla categoria di rischio "Furto e Rapina". Questa gerarchia permette di organizzare le diverse unità di rischio in modo che siano facilmente raggruppabili e ricercabili dall'utente. Le unità di rischio in una polizza possono essere molteplici e dipendono unicamente dal contenuto del contratto. Pertanto, poiché sarebbe impossibile fornire un insieme esaustivo di unità di rischio, viene dato modo all'utente di inserire solamente quelle che ritiene necessarie. L'inserimento di una

unità di rischio può avvenire in due modi: o importandola da un set di unità preesistenti sviluppato da RiskApp, oppure, creando una nuova unità di rischio ed associandola ad una categoria di rischio preesistente.

3.2 Analisi dell'interfaccia grafica del tool

Verrà analizzata ora l'interfaccia grafica del tool di annotazione come definita prima dello stage.
Come si può vedere in Figura 3.1, il tool di annotazione ha un'interfaccia grafica organizzata in quattro pannelli:

1. una toolbar posizionata nella parte superiore della pagina;
2. una sidebar sulla sinistra, che verrà chiamata "pannello delle annotazioni";
3. una sidebar sulla destra, che verrà chiamata "pannello di selezione rapida";
4. un pannello centrale in cui è possibile visualizzare il contenuto della polizza assicurativa e le annotazioni effettuate.

3.2.1 Toolbar

La toolbar, mostrata in Figura 3.2, contiene diverse funzionalità. Da sinistra a destra si trovano:

1. il pulsante per andare alla pagina precedente;
2. il pulsante per accedere al modale di aggiunta e modifica delle unità di rischio;
3. il pulsante per il filtraggio delle annotazioni;
4. il pulsante per visualizzare la sintesi del contratto;
5. un input per andare ad una pagina specifica all'interno della polizza;
6. i pulsanti per effettuare lo zoom del documento;
7. un pulsante per attivare la selezione dell'area;
8. un pulsante che apre un pop up che funge da legenda.

3.2.2 Pannello delle annotazioni

Il pannello delle annotazioni (vedi Figura 3.3), permette di visualizzare le diverse annotazioni effettuate sotto forma di card. Attraverso il click sul pulsante ad hamburger, si può aprire o chiudere il pannello per recuperare spazio nella visualizzazione del pdf. Ogni card (ad esempio, quella in Figura 3.4) mostra al suo interno :

1. le proprietà dell'annotazione;
2. un eventuale commento associato all'annotazione;
3. il testo selezionato o lo screenshot dell'area selezionata, in base al tipo di selezione effettuata;
4. l'indicazione della pagina in cui si trova l'annotazione;
5. i pulsanti di modifica ed eliminazione dell'annotazione.

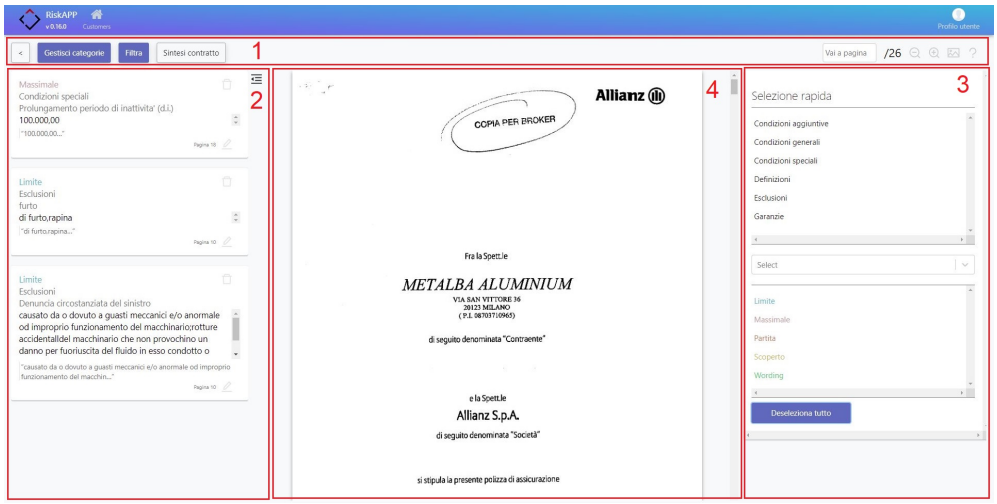

Figura 3.1: UI del tool di annotazione.

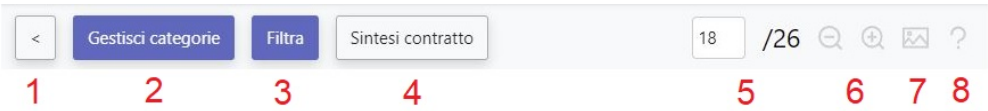

Figura 3.2: La toolbar del tool di annotazione.

3.2.3 Pannello di selezione rapida

Il pannello di selezione rapida, mostrato in figura 3.5, viene utilizzato per impostare o resettare i parametri associati all'annotazione rapida. È stato organizzato nella seguente maniera:

1. pannello per la scelta del tipo di sezione;
2. dropdown per la scelta dell'unità di rischio. Questo componente è stato realizzato estendendo le funzionalità dei dropdown di Bootstrap permettendo la ricerca da tastiera dell'unità di rischio;
3. pannello per la scelta della categoria di appartenenza dell'annotazione;
4. pulsante per il reset del pannello.

3.3 Analisi dell'implementazione del tool

3.3.1 Organizzazione del codice

Il codice del *front end* dell'applicativo è suddiviso per funzionalità: ogni cartella all'interno della cartella *src/* contiene il codice necessario al funzionamento dell'omonimo *tool*. Nel caso del *tool di annotazione delle polizze*, il codice si trova dentro la cartella *contractscomparison*, al cui interno si trovano le sottodirectory *actions*, *components* e *reducers*, oltre che al file *index.js* usato per esportare i *moduli* dichiarati all'interno delle suddette cartelle. La cartella *components* contiene al suo interno tutti i componenti

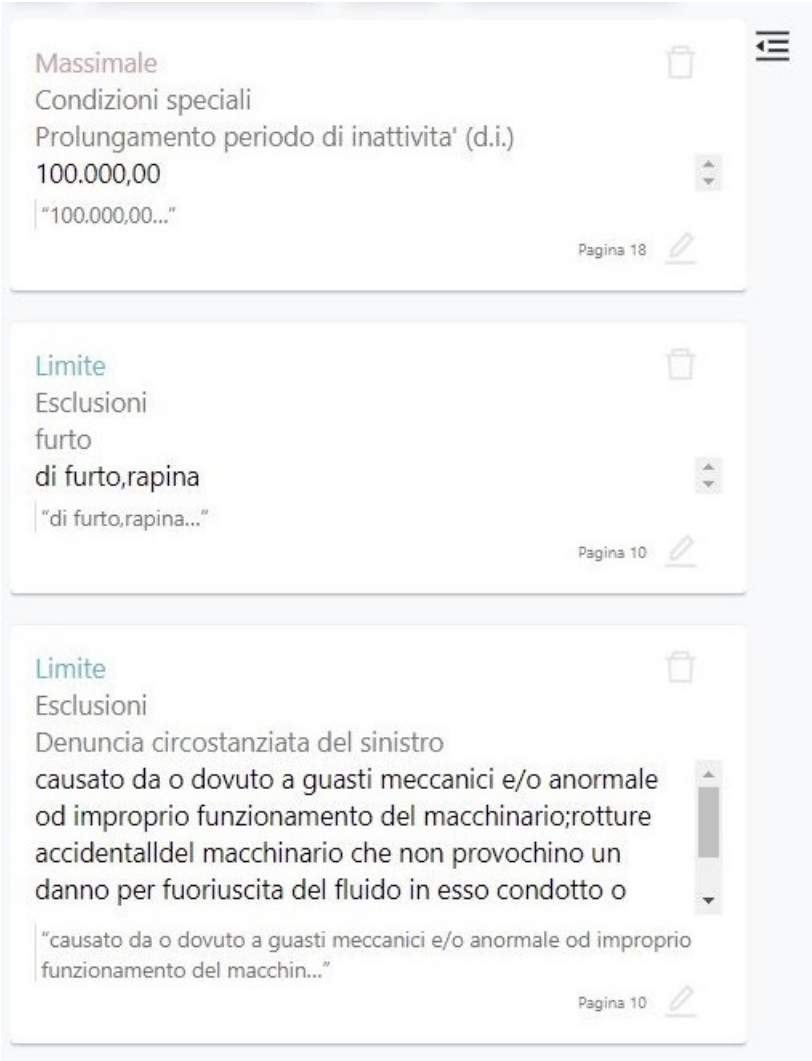

Figura 3.3: Il pannello delle annotazioni.

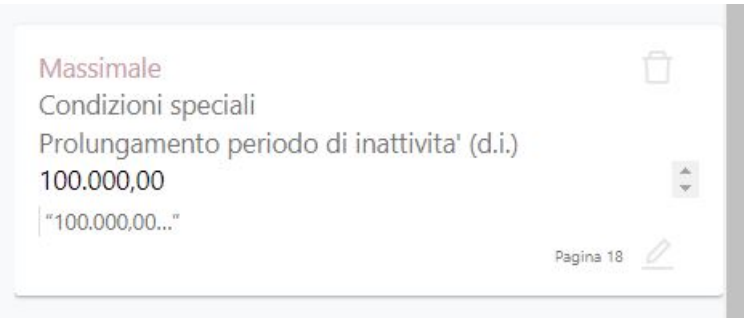

Figura 3.4: Un esempio di card, rappresentante una annotazione di testo a cui è stato aggiunto un commento.

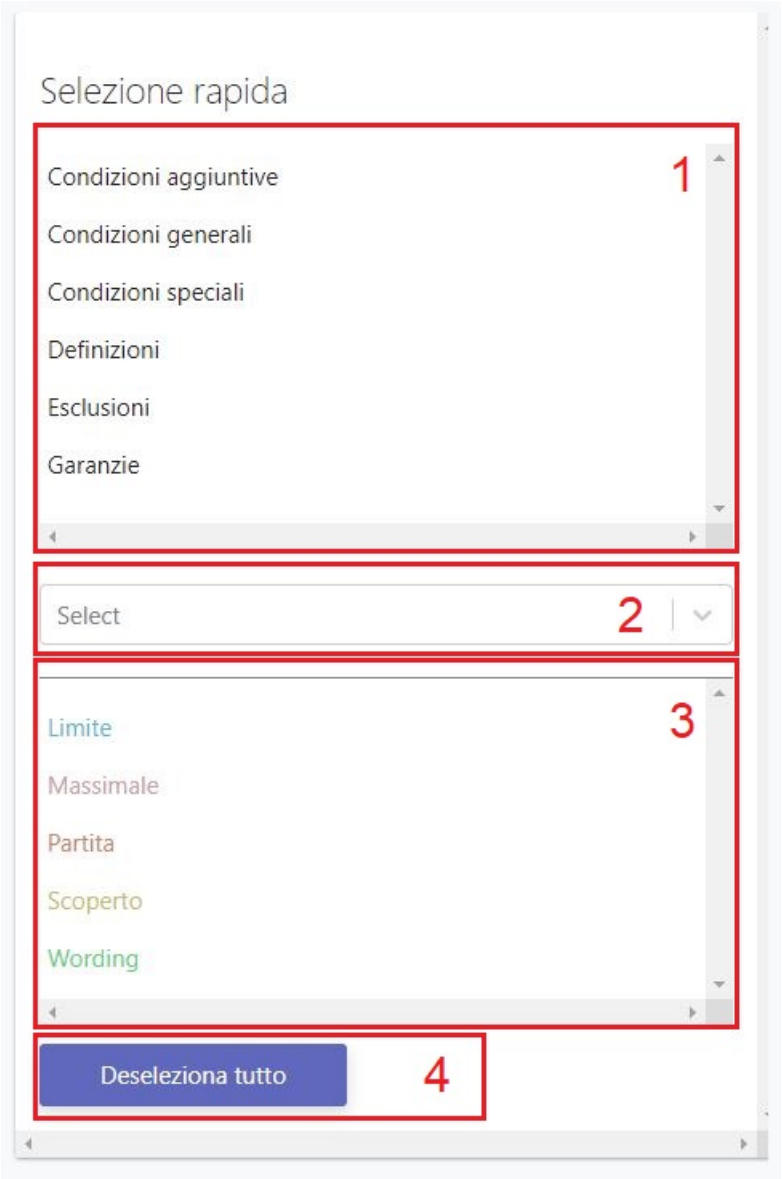

Figura 3.5: Il pannello di selezione rapida.

React usati per la creazione della UI, oltre che i *fogli di stile* e le librerie usate per la visualizzazione e l'annotazione dei pdf. Le cartelle *actions* e *reducers* al loro interno contengono rispettivamente le implementazioni delle *action* e dei *reducer* necessari per l'uso di Redux.
Il codice è organizzato nella seguente gerarchia:

```
-src
|-contractscomparison
| |-actions
| | |-index.js
| |-components
| | |-lib
| | |-SingleContractPage
| | | |-AddUnitType.jsx
| | | |-AnnotationCard.jsx
| | | |-DeleteModal.jsx
| | | |-Dropdowns.jsx
| | | |-EditUnitType.jsx
| | | |-FastAnnotationSidebar
| | | |-filterPdfPage.js
| | | |-ManageUnitTypesModal.jsx
| | | |-ModalEditAnnotation.jsx
| | | |-Sidebar.jsx
| | | |-SingleContractPage.jsx
| | | |-Toolbar.jsx
| | | |-...
| | |-style
| | |-...
| |--reducers
| |--index.js
|- ...
..
```

Dove i componenti all'interno di *SingleContractPage* hanno il seguente scopo:

* **AddUnitType**: componente che definisce logica e presentazione all'aggiunta di unità di rischio nel modale di gestione delle categorie;

* **AnnotationCard**: componente che definisce la presentazione di un'annotazione sotto forma di card;

* **DeleteModal**: componente che definisce la logica necessaria all'eliminazione di un'annotazione;

* **Dropdowns**: componente che definisce logica e presentazione delle dropdown usate nel filtraggio delle annotazioni;

* **EditUnitType**: componente che definisce logica e presentazione alla modifica delle unità di rischio nel modale di gestione delle categorie;

* **FastAnnotationSidebar**: componente che definisce logica e presentazione del pannello di annotazione rapida;

* **filterPdfPage**: file JavaScript usato per effettuare il filtraggio delle annotazioni;
* **ManageUnitTypesModal**: componente che definisce la presentazione del modale di gestione delle categorie;
* **ModalEditAnnotation**: componente che definisce logica e presentazione del modale di modifica delle annotazioni;
* **Sidebar**: componente che definisce logica e presentazione del pannello delle annotazioni;
* **SingleContractPage**: componente padre che richiama tutti gli altri;
* **Toolbar**: componente che definisce la presentazione della toolbar.

3.3.2 Componenti React

L'applicativo consiste in un insieme di componenti *React*, ognuno dei quali incapsula la logica, il comportamento e la presentazione di una parte dell'interfaccia. Un componente può essere visto come una funzione *JavaScript* che prende in input dei parametri chiamati *props*, possiede uno stato interno chiamato *state*, ed un metodo *render*, che si occupa di definire l'aspetto del componente. Lo stato interno del componente è un oggetto inizializzato dal costruttore alla creazione del componente e modificabile solo attraverso il metodo *useState()* messo a disposizione da *React*. In questa maniera, all'aggiornamento dello stato interno di un componente, il *framework* si occupa di aggiornare, in maniera *efficiente* ed *asincrona*, il componente stesso e tutti quelli che vi dipendono. A volte è necessario per un componente padre condividere informazioni con i componenti figli. Per questo motivo si utilizzano i *props*: proprietà che vengono passate dal padre ai figli al momento della loro invocazione ed ai quali è possibile accedere attraverso il campo *this.props*. Attraverso il passaggio di funzioni di callback[g]nei *props* ai componenti figli è possibile far eseguire a quest'ultimi funzioni in grado di modificare lo stato interno del componente padre [38].
In React sono presenti i "lifecycle methods", ovvero un insieme di metodi che ogni componente React eredita e che vengono invocati dalla libreria in momenti specifici del ciclo di vita dell'applicativo, come esplicitato dalla Figura 3.6. Attraverso la loro sovrascrittura permettono di definire il comportamento di un componente quando viene creato, aggiornato e distrutto.

3.3.3 Utilizzo di Redux

Talvolta può capitare che ci siano diversi componenti nella web app che abbiano bisogno di accedere e modificare lo stato di un certo componente. Per poter far fronte a questa necessità, bisognerebbe *elevare lo stato del componente* in oggetto fino al componente padre comune di tutti i componenti che necessitano di accedere allo stato, per poi propagarlo ai livelli inferiori della gerarchia attraverso l'uso dei props. *Redux* è una libreria che mette a disposizione un contenitore di stato, che permette di centralizzare in un posto solo lo stato della web app. Sarà poi possibile, iscrivendo i componenti a parti di questo store, accedere e modificare le informazioni lì contenute. Alla modifica da parte di un componente di una parte di store, tutti i componenti *React* iscritti a quella parte modificata vengono aggiornati. In questa maniera, si può utilizzare lo stato di *React* per memorizzare solamente le informazioni necessarie al funzionamento della UI, mentre la memorizzazione delle informazioni che caratterizzano la *logica di*

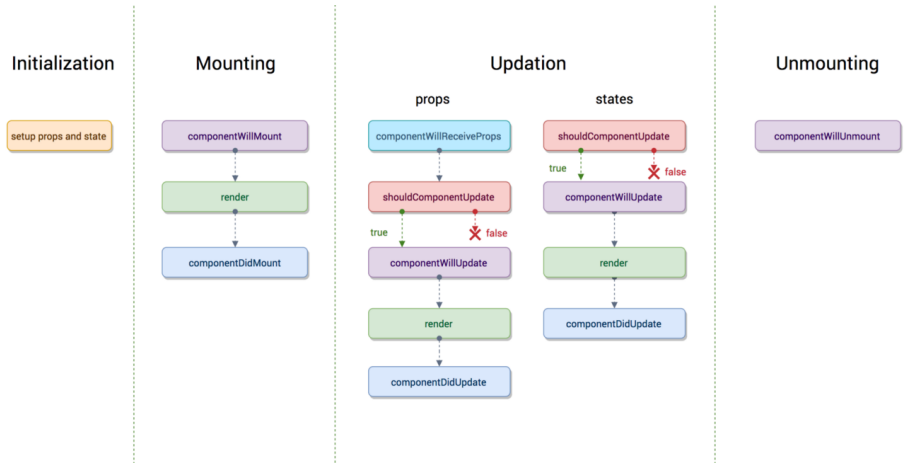

Figura 3.6: Esempio di ciclo di vita di un componente React. [23].

business possono essere delegate a Redux, separando la *logica di presentazione* da quella di *modello*. Nel caso del tool di annotazione, lo *store Redux* viene utilizzato per memorizzare le annotazioni, oltre che ad altri dati come, ad esempio, le unità di rischio. Lo store, per *Redux*, è *immutabile*, e pertanto non può essere modificato direttamente. La modifica, quindi, può avvenire solamente attraverso il *dispatch* di una azione, che viene processata da un *reducer*: ovvero, una funzione pura che dati lo stato attuale ed un'azione, è in grado di calcolare il nuovo stato. Una rappresentazione grafica del *workflow* di redux si può vedere in Figura 3.7.

3.3.4 Annotazione della polizza

Per l'implementazione delle funzionalità di annotazione, l'applicazione RiskApp si basa su *PDF.js*. *PDF.js* è una libreria *JavaScript open source* il cui sviluppo è supportato da *Mozilla* e che consente di visualizzare documenti pdf all'interno di un elemento *canvas HTML5*. Con più di un milione di download settimanali ed in costante crescita [26], questa tecnologia è lo standard di fatto per la visualizzazione di documenti pdf in ambito web. È importante sottolineare come *PDF.js* sia una libreria finalizzata alla visualizzazione dei documenti e non alla loro modifica. La libreria *PDF.js* è organizzata in tre livelli: *Core*, *Display* e *Viewer*, il cui ruolo risulta importante da comprendere per avere un'idea del suo funzionamento.

* **Core**: il nucleo di PDF.js. Il suo compito è quello di interpretare il documento secondo lo standard ISO 19005[g]PDF/A.

* **Display**: espone le diverse API[g]necessarie al rendering del documento. La lista completa delle API esposte si trova al riferimento sitografico [28]. Questo livello si occupa di generare e mentenere aggiornato il *text layer*, il cui ruolo è spiegato di seguito.

* **Viewer**: questo livello mette a disposizione il visualizzatore che permette di mostrare a schermo il documento ed una collezione minimale di funzionalità per interagirvi, come zoom, ricerca e simili.

La funzionalità principale del tool di annotazione va oltre la visualizzazione di un documento pdf: deve essere in grado di evidenziare parti di questo documento. In

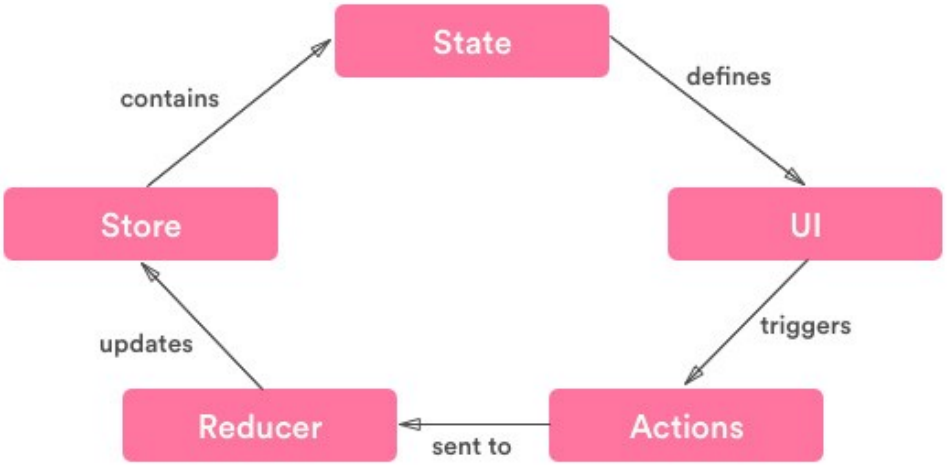

Figura 3.7: Workflow del framework react. [24].

questo caso, la libreria PDF.js non è di aiuto e risulta quindi necessario per RiskApp impiegare altre tecnologie, come ad esempio *react-pdf-highlighter* [36], che permette di aggiungere in maniera relativamente semplice questa funzionalità. Questa libreria si interfaccia a *PDF.js* attraverso il suo *text layer*.

```
▼<div class="textLayer" style="width: 794px; height: 1122px;">
   <span style="left: 587.067px; top: 31.08px; font-size: 26.4933px; font-family: sans-serif; transform: scaleX(1.21182);">Allianz</span>
   <span style="left: 353.28px; top: 94.6133px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.08562);">COPIA </span>
   <span style="left: 405.413px; top: 91.28px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.08151);">PER </span>
   <span style="left: 442.347px; top: 87.0133px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.07746);">BROKER</span>
   <span style="left: 335.04px; top: 349.827px; font-size: 15.4533px; font-family: sans-serif; transform: scaleX(0.745232);">Fra </span>
   <span style="left: 357.173px; top: 349.293px; font-size: 15.4533px; font-family: sans-serif; transform: scaleX(0.758055);">la </span>
   <span style="left: 370.64px; top: 349.867px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(0.94741);">Spett.le</span>
   <span style="left: 212.16px; top: 399.333px; font-size: 25.9467px; font-family: sans-serif; transform: scaleX(1.15873);">METALBA </span>
   <span style="left: 358.56px; top: 397.947px; font-size: 25.8667px; font-family: sans-serif; transform: scaleX(1.30426);">ALUMINIUM</span>
   <span style="left: 311.04px; top: 437.347px; font-size: 10.4933px; font-family: sans-serif; transform: scaleX(1.3917);">VIA </span>
   <span style="left: 338.373px; top: 436.8px; font-size: 11.04px; font-family: sans-serif; transform: scaleX(1.12101);">SAN </span>
   <span style="left: 366.507px; top: 436.267px; font-size: 11.04px; font-family: sans-serif; transform: scaleX(1.19887);">VITTORE </span>
   <span style="left: 427.44px; top: 436.267px; font-size: 11.04px; font-family: sans-serif; transform: scaleX(1.05571);">36</span>
   <span style="left: 331.68px; top: 452.16px; font-size: 11.04px; font-family: sans-serif; transform: scaleX(1.0628);">20123 </span>
   <span style="left: 368.613px; top: 452.16px; font-size: 11.04px; font-family: sans-serif; transform: scaleX(1.23601);">MILANO</span>
```

Figura 3.8: Esempio di text layer generato.

Il text layer è la parte di document object model[g], o *DOM*, in cui il contenuto del documento pdf viene renderizzato dinamicamente sotto forma di elementi *span* html, successivamente posizionati attraverso l'uso di *css inline* e della proprietà *css transform*. È possibile vedere come il *text layer* renderizza il documento pdf in Figura 3.8. Questo componente di *PDF.js* è fondamentale perché è l'unico che consente a librerie come *react-pdf-highlighter* di interagire con il contenuto del documento pdf e aggiungere funzionalità oltre a quelle nativamente previste da *PDF.js*.

Per far ciò, *react-pdf-highlighter* crea un ulteriore livello *PdfLoader*, come è possibile vedere in Figura 3.9, posizionato al di sopra del *text layer* di *PDF.js*, all'interno del quale vengono registrati eventi del mouse come il movimento, la pressione e il rilascio del tasto sinistro, che permettono di definire una selezione. In particolare, una volta terminata una selezione, si ottengono delle coordinate con cui è possibile determinare un'area di documento di cui recuperare il testo presente nel *text layer*,

```
▼<div class="PdfLoader-container-with-notes">
  ▼<div touch-action="auto">
    ▼<div class="PdfHighlighter">
      ▼<div class="pdfViewer removePageBorders" style="padding-bottom: 120px;">
        ▼<div class="page" data-page-number="1" style="width: 794px; height: 1122px;" data-loaded="true">
          ▶<div class="canvasWrapper" style="width: 794px; height: 1122px;">…</div>
          ▼<div class="textLayer" style="width: 794px; height: 1122px;">
             <span style="left: 587.067px; top: 31.08px; font-size: 26.4933px; font-family: sans-serif; transform: scaleX(1.21182);">Allianz</span>
             <span style="left: 353.28px; top: 94.6133px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.08562);">COPIA </span>
             <span style="left: 405.413px; top: 91.28px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.08151);">PER </span>
             <span style="left: 442.347px; top: 87.0133px; font-size: 14.3467px; font-family: sans-serif; transform: scaleX(1.07746);">BROKER</span>
```

Figura 3.9: Il livello PdfLoader di react-pdf-highlighter viene posizionato al di sopra del text-layer.

```
▼<div class="PdfHighlighter__highlight-layer">
  ▼<div>
    ▼<div>
      ▼<div class="Highlight ">
        ▼<div class="Highlight__parts">
           <div id="0ed8bf8d-0276-4411-8afb-bcb731a521ce0" class="Highlight__part" style="left: 210.734px; top: 235.625px; width: 57.0916px; height: 12px;
           color: rgb(181, 125, 133); background: rgb(255, 209, 216);"></div>
```

Figura 3.10: Esempio di annotazione salvata nell'highlight layer.

oppure uno *screenshot* della corrispondente area selezionata. L'annotazione, in seguito viene aggiunta in coda al *text-layer* nell'*highlight layer*, come evidenziato dalla Figura 3.10.

3.4 Problematiche del tool di annotazione

Fin dal primo utilizzo, risulta chiaro come il tool di annotazione soffra di diversi problemi: alcuni tecnici ed altri che evidenziano una progettazione superficiale. Tra i problemi tecnici troviamo:

* l'interfaccia che non è sviluppata in maniera *responsive*. Nonostante RiskApp non sia pensata per l'uso mobile, ma piuttosto per un uso desktop in ufficio, l'applicazione dovrebbe adattarsi alle diverse ampiezze di schermo senza rompersi;

* alcune funzionalità non funzionano come inteso o non funzionano affatto. Per esempio: l'aggiunta di nuove unità di rischio segna il successo della procedura di aggiunta, ma le unità di rischio non vengono aggiornate; non sempre è possibile eliminare i filtri necessari per il filtraggio delle annotazioni; il click sulla card dell'annotazione non sempre porta all'annotazione;

* per realizzare alcune funzionalità vengono utilizzati i componenti sbagliati. Per esempio, per realizzare la funzionalità "vai a pagina ..." del documento, viene utilizzato un

  ```
  <input type=''text''>
  ```

  e non un

  ```
  <input type=''number''>
  ```

  Questo porta alla necessità di aggiungere maggiori controlli sull'input oltre che dare un significato semantico sbagliato al componente;

* mancanza dell'adozione di buone pratiche in merito ad accessibilità e usabilità. Mancano i tag WAI-ARIA[g], le *label* associate ai campi dei *form*, i *title* associati ai bottoni, e la maggior parte del testo nell'applicativo è *hard-coded* nonostante negli altri tool della piattaforma venga usato il *framework Polyglot.js* per effettuare le traduzioni;

* mancanza di buone pratiche di programmazione, ad esempio, presenza di codice duplicato, mancanza di adesione agli stili di codifica proposti dai *framework* usati, problemi di prestazioni, promiscuità tra la parte di presentazione e quella di logica dei componenti *React*.

Tra i problemi di progettazione, elenchiamo:

* studio approssimativo dei casi d'uso. Basti pensare al fatto che l'annotazione normale e quella rapida svolgono esattamente la stessa funzione, e di fatto, l'unico tipo di annotazione che viene usata è quella rapida;

* l'interfaccia risulta poco immediata e di difficile comprensione per l'utente. Inoltre, la suddivisione in pannelli risulta essere troppo "ingombrante" e poco scalabile, visto che toglie una quantità importante di schermo alla polizza assicurativa, e tanto più è grande l'area o la selezione del testo, tanto più grande risulta essere la card dell'annotazione, la quale non ha limiti alla sua crescita.

Capitolo 4

Ristrutturazione e ampliamento del tool di annotazione

Nel corso di questo capitolo viene presentato il lavoro svolto durante il progetto di stage.

4.1 Studio di fattibilità sul cambio delle tecnologie

Uno dei punti chiave del progetto di stage è stata la ricerca di nuove tecnologie adottabili per la visualizzazione e l'annotazione della polizza assicurativa.
Come anticipato nella Sezione 2.1, il tool di annotazione è stato sviluppato inizialmente nel 2018 per poi venire accantonato ed essere ripreso solo in tempi più recenti.
Questo significa che le scelte tecnologie effettuate a suo tempo da RiskApp potrebbero non essere più le scelte migliori per continuare lo sviluppo dell'applicativo. Inoltre, come spiegato nella Sezione 3.3.4, PDF.js tende ad essere "limitante" per quanto riguarda lo sviluppo di funzionalità di modifica dei documenti pdf. È stato quindi necessario ricercare quali tecnologie avrebbero potuto sostituire quelle attualmente usate e come quelle attualmente usate siano cambiate dal momento in cui si è fermato lo sviluppo del tool, in modo da poter effettuare una scelta ponderata: decidere se mantenere le tecnologie attuali o migrare verso soluzioni differenti.

4.1.1 Alternative a PDF.js

Per la visualizzazione dei documenti PDF in browser, attualmente non esistono molte alternative a PDF.js. Le poche presenti nel mercato, infatti, si suddividono in due categorie: quelle basate su PDF.js e quelle che, invece, utilizzano interpreti proprietari per lo standard PDF/A. Le prime si occupano di incapsulare PDF.js in una libreria proprietaria in modo da mettere a disposizione ulteriori funzionalità, ma delegando l'interpretazione ed il rendering del documento a PDF.js. Le seconde, invece, utilizzano soluzioni proprietarie in grado di interpretare, renderizzare e modificare documenti in diversi formati, oltre che offrire molte altre funzionalità in grado di aumentare la flessibilità della tecnologia. Per lo studio di fattibilità, oltre a PDF.js sono state considerate altre due librerie proprietarie della *software house* PDFTron: PDF.js Express e l'omonimo PDFTron, alcune alternative a PDF.js tre le più affermate nel mercato [31]. Verranno ora analizzati pregi e difetti di queste tecnologie.

PDF.js

PDF.js è la libreria attorno alla quale è stato originariamente sviluppato il *tool* di annotazione. La versione adottata è la *2.4.456* rilasciata il 19 marzo 2020, mentre l'ultima versione disponibile è la *2.13.216* rilasciata il 27 febbrario 2022.
I motivi che hanno spinto RiskApp ad andare in cerca di una soluzione diversa da PDF.js sono principalmente legati alla difficoltà di annotare il documento pdf ed alle prestazioni, oltre che a problemi di fedeltà nel rendering, che portano alla generazione di parti di testo errato nel caso in cui il documento sia stato scansionato attraverso una fotocamera di bassa qualità.
Consultando i changelog[g]disponibili nella sezione *releases* del progetto *Github* di PDF.js, si può notare come tra le due versioni citate siano stati effettuati numerosi aggiornamenti che hanno apportato miglioramenti alle prestazioni, alla stabilità ed al rendering del testo [30], che erano alcune delle motivazioni che, in primo luogo, hanno spinto RiskApp a cercare tecnologie alternative a PDF.js.
Tra i difetti di PDF.js [32] annoveriamo:

* PDF.js si appoggia al browser per il *rendering* dei font e delle immagini, e questa caratteristica può portare a possibili comportamenti inaspettati tra browser diversi;
* PDF.js non ha come priorità il rendering dei documenti pdf ad alta fedeltà;
* inefficienza nella visualizzazione di documenti di grosse dimensioni;
* libreria pensata per la sola visualizzazione dei documenti;
* mancanza di documentazione esaustiva sulla libreria;
* mancanza di supporto alle gesture[g]mobile.

Tra i pregi invece, troviamo [32]:

* progetto *open source* distribuito con licenza Apache 2.0;
* popolare e mantenuto con rilasci frequenti ancora dopo anni;
* *tool* di annotazione già implementato con questa libreria.

Riassumendo, PDF.js è una libreria che mette a disposizione un visualizzatore di documenti pdf in browser semplice da usare, anche se questa semplicità ha un costo: si è costretti a scendere a compromessi in ambito di performance, affidabilità e qualità del *rendering*. Questo esclude PDF.js dalle tecnologie utilizzabili quando l'alta fedeltà di documenti "pesanti" e la necessità di fare modifiche al documento è una priorità del business, come in contesti di computer grafica. In quel caso è necessario rivolgersi ad altre tecnologie.

PDF.js Express

PDF.js Express è una soluzione commerciale offerta dalla *software house PDFTron* basata su PDF.js, la quale, offre i seguenti upgrade su PDF.js:

* opzioni di ricerca multiple;
* interfaccia utente curata e personalizzabile;

* night mode;
* supporto cross-browser[g]ed alle gesture nelle tecnologie *mobile*;
* documentazione esaustiva e professionale;
* possibilità di aggiungere ai pdf disegni, compilare *form*, usare strumenti di misura ed effettuare firme digitali.

Queste ulteriori funzionalità, chiaramente, vengono ad un costo aggiuntivo che oscilla intorno ai 400$/mese. PDF.js Express mette a disposizione una soluzione interessante e semplice da integrare per chi volesse sviluppare un applicativo il cui scopo principale è visualizzare e gestire in maniera completa documenti in formato pdf, offrendo funzionalità richieste da molto tempo dalla community di PDF.js ma mai implementate nella libreria.

PDFTron

PDFTron è la versione interamente proprietaria di PDF.js Express. Infatti, invece di usare l'interprete di PDF.js per leggere i documenti pdf, PDFTron utilizza un interprete proprietario in grado di leggere anche formati di file diversi, come ad esempio, i file *word*. Oltre a questo, PDFTron offre:

* rendering ad alta fedeltà;
* possibilità di aggiungere annotazioni;
* scalabile attraverso elaborazione serverless[g];
* integrato con gli altri software PDFTron;
* supporto diretto, consulenza e formazione.

Riguardo i costi, questa tecnologia offre la possibilità di una prova gratuita per poi passare ad un piano tariffario personalizzato in base alle esigenze richieste. Questa tecnologia offre il maggior numero di funzionalità, e si presta ad essere usata per lo sviluppo di suite per la produttività da ufficio.

4.1.2 Scelte di progetto

Alla luce dell'analisi effettuata, è stato scelto di mantenere ed aggiornare PDF.js in quanto è la tecnologia che meglio si addice alle esigenze di RiskApp.
Bisogna far presente come PDF.js sia "maturata" notevolmente in questi ultimi anni, offrendo una tecnologia più completa e decisamente più affidabile rispetto a quella offerta fino a qualche anno prima.
È vero che le alternative proprietarie come PDF.js Express e PDFTron offrono più funzionalità rispetto a PDF.js, però è anche vero che in RiskApp, tutte quelle funzionalità aggiuntive non sono richieste e vanno al di fuori dello *scope* dello strumento di annotazione: ovvero la creazione di annotazioni secondo la definizione data nella precedente Sezione 3.1.2, piuttosto che la creazione di una *suite* completa per la gestione di documenti pdf.
PDF.js Express, inoltre, essendo basato su PDF.js, ne eredita tutti i limiti tecnici e, dal momento che incapsula PDF.js al suo interno, dipende da almeno un'altra

organizzazione per la risoluzione di eventuali *bug*. Un punto a favore di PDF.js Express è il miglior supporto alle tecnologie *mobile*. La piattaforma RiskApp però, è sviluppata e venduta come soluzione desktop, per cui non è una priorità del business il supporto alle *gesture mobile*. Inoltre, se si volesse sviluppare il tool di annotazione in un'ottica mobile-friendly[g], bisognerebbe rivedere completamente il funzionamento e l'organizzazione dell'interfaccia grafica, per adattare l'esperienza d'uso del tool ad uno schermo di dimensioni ridotte.
Altra motivazione importante, è che attualmente il tool di annotazione funziona, eccezione fatta per diversi bug che saranno oggetto della ristrutturazione. Pertanto, si è preferito aggiornare le tecnologie già utilizzate risparmiando tempo, e concentrando gli sforzi nella manutenzione e sviluppo dell'interfaccia grafica piuttosto che nella migrazione ad un nuovo *set* tecnologico.

4.1.3 Aggiornamento della tecnologia

Com'era prevedibile, l'aggiornamento di PDF.js ha introdotto dei breaking changes[g], che è stato necessario risolvere prima di poter proseguire con la ristrutturazione del codice. La risoluzione di queste problematiche, seppure semplice, è stata complicata dal fatto che la documentazione di PDF.js non esplicitasse chiaramente l'introduzione di alcuni cambiamenti nell'uso della libreria. Verranno ora presentate le problematiche riscontrate e le conseguenti soluzioni adottate dopo l'aggiornamento della libreria.

Compilazione del codice

* **Problematica**: il codice non viene compilato, in quanto non viene interpretato correttamente dal compilatore.
* **Causa**: il codice di PDF.js è stato convertito in formato *ES2020* (*ECMAScript2020*) ed alcuni operatori (chaining operator[g]) non vengono riconosciuti dal compilatore.
* **Soluzione adottata**: è stato importato il codice in formato *ES6* dalla cartella *legacy/* di PDF.js al posto di quello in formato *ES2020*.

Nuova gestione degli eventi

* **Problematica**: il documento pdf non viene caricato.
* **Causa**: per permettere la presenza di più visualizzatori pdf all'interno di una stessa pagina, PDF.js ha cambiato la gestione degli eventi. JavaScript è un linguaggio orientato agli eventi: questo significa che allo scatenarsi di un evento (come il click del mouse su un bottone, il ridimensionamento della finestra del browser, o anche un evento lanciato dal sistema operativo), è possibile associare l'esecuzione di codice. Per far questo, si effettua il binding[g]di un evento ad una funzione [15].
  Anche PDF.js lancia degli eventi: in questo modo lo sviluppatore può effettuare il binding agli eventi scatenati da PDF.js per definire il comportamento del Visualizzatore. Più nello specifico, la libreria, nella versione precedente, lanciava degli eventi globali a cui era possibile effettuare il binding del Viewer (il componente dell'architettura di PDF.js incaricato di visualizzare il documento pdf) attraverso *document.addEventListener()*. Nella nuova versione, la libreria

è passata ad una gestione separata degli eventi per ogni *Viewer*, attraverso un oggetto *eventBus* passato come parametro nella costruzione di un *Viewer*. In questa maniera, è possibile risalire al *Viewer* in cui è stato scatenato l'evento ed agire di conseguenza.

* **Soluzione adottata**: viene aggiunta alla classe JavaScript incaricata di costruire il visualizzatore pdf, un oggetto *eventBus* sul quale effettuare binding dei metodi per gli eventi *"pagesinit", "textlayerrendered" e "pagechanghing"* scatenati da PDF.js, necessari per definire il comportamento del *Viewer*.

Posizionamento del Viewer

* **Problematica**: il documento pdf non viene caricato.
* **Causa**: la libreria lancia un errore in console se il contenitore del *Viewer* non è posizionato assolutamente.
* **Soluzione adottata**: attraverso codice *JavaScript* viene impostata il posizionamento assoluto del nodo contenente il *Viewer*.

4.2 Ristrutturazione del front end

La ristrutturazione del *front end* si è svolta in due fasi: nella prima, sono state effettuate delle modifiche alla codebase preesistente (refactoring) con lo scopo di migliorarne la leggibilità e correggere alcuni dei problemi tecnici esposti alla Sezione 3.4, mentre nella seconda fase è avvenuto il vero e proprio *restyling* della UI.

4.2.1 Refactoring del codice esistente

Una parte dello stage è stata rivolta all'individuazione e alla soluzione di diverse problematiche relative alla struttura del codice esistente. In questa sezione verranno trattate solamente le maggiori problematiche riscontrate durante il processo di ristrutturazione e della loro risoluzione.

Promiscuità tra presentazione e logica del pannello di annotazione rapida

* **Problematica**: come si può vedere in Figura 4.1, nel pannello di annotazione rapida i menù per la selezione dei tipi di categoria e di sezione utilizzano il colore di sfondo dell'elemento cliccato per stabilire quale sia la categoria e il tipo di sezione correntemente attivo. Questo avrebbe portato a comportamenti non previsti nel momento del *restyling* della UI[g]e nel caso l'utente utilizzi estensioni che vanno a modificare i colori mostrati in browser, per esempio, per aumentare il contrasto tra testo e sfondo. Inoltre, è importante notare come venga usato in maniera estensiva il metodo *getElementsByClassName()* per recuperare dal DOM elementi HTML su cui poi effettuare controlli di logica. Questa non è una buona pratica di programmazione, in quanto le classi CSS non dovrebbero essere usate con la funzione di identificatore del ruolo di un elemento.
* **Soluzione adottata**: il codice, come si può vedere in Figura 4.2, è stato riscritto adottando una variabile che tiene in memoria l'ultimo elemento selezionato, ed utilizzando *getElementsByName()* al posto di *getElementsByClassName()*.

```
1. loadCategories = () => {
2.     //…
3.     <li
4.         key={index}
5.         onClick={(event) => {
6.             if (
7.                 document.getElementsByClassName(
8.                     "quicknote-principal-categories"
9.                 )[index].style.backgroundColor === "rgb(255, 255, 255)" ||
10.                index !== this.state.indexOfSelectedCategory
11.                ) {
12.                    this.handleClickCategory(event, index, cat.uuid, color);
13.                }
14.                //…
15.    </li>
16.    //…
17. };
```

Figura 4.1: Esempio di codice problematico, che a riga 9 esegue controlli sul colore di sfondo.

```
1. loadCategories = () => {
2.     //...
3.     <li
4.         name={"quicknote-categories-list"}
5.         value={0}
6.         onMouseDown={(event) => {
7.             const category = document.getElementsByName(
8.                 "quicknote-categories-list"
9.                 )[index];
10.            if (category.value === 0) {
11.                document
12.                    .getElementsByName("quicknote-categories-list")
13.                    .forEach((c) => {
14.                        c.value = 0;
15.                    });
16.                category.value = 1;
17.                this.handleClickCategory(event, index, cat.uuid, color);
18.            }
19.        //...
20.        };
21.    //...
```

Figura 4.2: Esempio di codice corretto: esegue controlli solamente sul valore del campo booleano value (vedi riga 13), non utilizza le funzioni getElementsByClassName() e mantiene la coerenza del valore attivo al click del mouse

```
1. filteredHighlights: new Annotations(this.props.annotations)
2.          .filterByCategoryType(this.props.categoryTypeFilters)
3.          .filterBySectionType(this.props.sectionTypeFilters)
4.          .filterByUnitType(this.props.unitTypeFilters)
5.          .toArray();
```

Figura 4.3: Esempio di filtraggio delle annotazioni eseguito in maniera dichiarativa.

```
1. <p>Sto caricando i dati</p>
```

Figura 4.4: Esempio di testo hard-coded impossibile da tradurre.

Refactoring del codice adibito al filtraggio delle annotazioni

* **Problematica**: il filtraggio delle annotazioni viene inutilmente effettuato in maniera estremamente complicata (non verrà riportato il codice originale in quanto eccessivamente prolisso) ed in certi casi, errata.
* **Soluzione adottata**: è stata aggiunta una nuova classe JavaScript *Annotations* che fornisce diversi metodi per accedere e filtrare le annotazioni in maniera dichiarativa. È stato necessario, al fine di mantenere la compatibilità con il resto del codice, dotare la classe del metodo *toArray()*, metodo che invocato al termine delle operazioni, permette di ritornare l'array di annotazioni com'era originariamente concepito. Il filtraggio delle annotazioni può essere quindi effettuato semplicemente come illustrato nel codice riportato in Figura 4.3.

Traduzione automatica non funzionante

* **Problematica**: sono presenti stringhe di testo *hard-coded* che non permettono di tradurre il contenuto della pagina in base alle preferenze del browser. Un esempio di questo problema è riportato in Figura 4.4.
* **Soluzione adottata**: ogni stringa di testo deve essere tradotta ed inserita sia in italiano che in inglese in un file *translation.JSON* che associa ad un identificativo le due stringhe di testo, ed in base alle preferenze del browser viene caricata quella opportuna. In Figura 4.5 si trova un esempio di codice caricato dinamicamente in base alla lingua del browser. Un esempio del file JSON usato per le traduzioni si trova in Figura 4.6. Consultare la documentazione di *polyglot.js* per maggiori informazioni [33].

Scomparsa della selezione di testo

* **Problematica**: la selezione del testo effettuata scompare quando si clicca dentro la barra di annotazione rapida, togliendo all'utente un riferimento visivo che lo aiutasse a ricordare che parte di documento aveva selezionato.

```
1. <p>{this.props.p.t("contractscomparison.loading_data")}</p>
```

Figura 4.5: Esempio di testo caricato dinamicamente da polyglot.js .

```
1. {
2.   "en": {
3.     "contractscomparison": {
4.       ...
5.       "loading_data": "Loading data",
6.       ...
7.     },
8.   },
9.   "it": {
10.    "contractscomparison": {
11.      ...
12.      "loading_data": "Sto caricando i dati",
13.      ...
14.    },
15.  }
16. }
```

Figura 4.6: Esempio di file translation.js.

* **Causa**: la causa è da ricercarsi nel fatto che la selezione di elementi di testo all'interno di una pagina web esiste finché la selezione possiede il *focus* della pagina. Nel momento in cui il *focus* viene perso, per esempio perché si clicca un pulsante, la selezione smette di esistere.

* **Soluzione adottata**: per ovviare a questo problema, è stato necessario riscrivere le componenti all'interno del pannello di annotazione rapida basandosi su elementi html nativi (e non componenti provenienti da React-Bootstrap) e utilizzando il metodo *preventDefault()* per bloccare il bubbling[g]degli eventi ai livelli superiori del DOM. È stato necessario sviluppare i componenti usando elementi html nativi in quanto il comportamento dei componenti React-Bootstrap relativo agli eventi si è rivelato diverso rispetto a quello di default degli elementi html. Per esempio, il *bubbling* dell'evento legato alla selezione su un elemento *<select>* può essere bloccato attraverso il metodo *preventDefault()*, diversamente da quello legato alla selezione del corrispondente componente React *<Dropdown />*. La documentazione in merito non fornisce spiegazioni, ed attualmente è aperta una *issue* nel progetto React-Bootstrap relativa al problema in attesa di essere risolta.

Posizionamento dei tooltip

* **Problematica**: errato posizionamento verticale dei tooltip per la conferma della selezione.

* **Causa**: PDF.js inserisce all'interno del *text layer* degli elementi senza contenuto, invisibili all'utente, con dimensioni *width=0* e *height>0*.

* **Soluzione adottata**: vengono filtrati suddetti elementi per ottenere un set di coordinate che corrisponda effettivamente alla porzione di documento selezionata.

Mancanza di fluidità nell'uso dell'applicazione

* **Problematica**: animazioni a scatti nel momento in cui si fa collassare il pannello delle annotazioni.

* **Causa**: per far collassare il pannello delle annotazioni, veniva modificata la larghezza del container del pannello, che attraverso un'animazione, veniva portata dal valore originale a 0. Questa però, non è una buona tecnica da usare quando si vuole far scomparire un pannello, in quanto il browser è costretto, per ogni frame dell'animazione, a ricalcolare il box model[g]e renderizzare il pannello con tutte le card contenute al suo interno, provocando numerosi layout shift[g]che rendono il tutto poco fluido.

* **Soluzione adottata**: invece di ridurre la larghezza del pannello delle annotazioni è stato deciso di farlo scomparire dallo schermo posizionandolo assolutamente a sinistra di una quantità negativa di pixel attraverso una animazione. In questa maniera l'animazione risulta essere decisamente più fluida e gradevole. Per mantenere il tutto accessibile è importante aggiornare il campo *visibility* del pannello, in modo da rimuoverlo/aggiungerlo dagli elementi accessibili da uno *screen reader*. La stessa tecnica è stata usata in maniera analoga per il pannello di annotazione rapida.

Annotazione inesistente

* **Problematica**: a volte il click del mouse sulla card di un'annotazione non esegue alcuna azione al posto di portare al punto del documento in cui l'annotazione si trova.

* **Causa**: la causa è da ricercarsi nel modo in cui PDF.js carica dinamicamente il *text layer*. Quest'ultimo infatti, non viene caricato interamente al caricamento del documento, ma viene generato in maniera *lazy* solo quando la relativa pagina viene renderizzata. Questo significa che se PDF.js non ha ancora caricato il text layer della pagina in cui si trova l'annotazione su cui viene eseguito il click, l'azione di effettuare lo scrolling fino al *div* in cui si trova l'annotazione non può essere eseguita in quanto il *div* è *undefined*.

* **Soluzione adottata**: è stato possibile risolvere il bug memorizzando, alla creazione di una nuova annotazione, anche la posizione all'interno del documento. In questo modo non si è costretti ad aspettare che il text layer venga generato da PDF.js.

4.2.2 Restyling della UI

Il *restyling* della UI è avvenuto in tre fasi: una prima in cui si è analizzato il funzionamento e l'interfaccia del software con l'obiettivo di semplificarli, una seconda in cui sono stati creati una serie di wireframe[g]con *Figma*, che successivamente alla loro validazione da parte del tutor aziendale sono serviti come linee guida per la terza fase: la ristrutturazione dei componenti della UI.

Analisi criticità dei componenti esistenti

Riportiamo di seguito, per i diversi componenti esistenti, un'analisi delle criticità riscontrate.

* **Toolbar**: la prima cosa che si nota con l'utilizzo del software è la toolbar e come questa racchiuda al suo interno diverse funzionalità con scopi diversi. Per esempio, il filtraggio delle annotazioni viene effettuato nella toolbar, anche se esiste il

pannello delle annotazioni in cui ci si aspetterebbe di trovare tale funzionalità.
Inoltre, la toolbar, quando la finestra viene ridimensionata e vengono mostrati i *dropdown* usati per filtrare le annotazioni, si rompe: i *dropdown* si sormontano con gli altri componenti e l'altezza della toolbar aumenta notevolmente, togliendo molto spazio alla visualizzazione della polizza.
Questo fa capire come la toolbar debba essere rivista non solo nelle funzionalità e nella sua organizzazione, ma anche in maniera tale da degradare gradevolmente al variare della dimensione della pagina, senza togliere troppo spazio alla polizza.

* **Pannello delle annotazioni e relative card**: il pannello delle annotazioni è un altro componente che necessita di una ristrutturazione abbastanza profonda. Dal pulsante ad hamburger, posizionato alla destra del pannello, incaricato di aprire e chiudere quest'ultimo, si evince che quel componente avrebbe voluto avere un comportamento analogo a quello di un *Offcanvas* di Bootstrap, una barra laterale a comparsa che funge da sistema di navigazione per siti web orientati all'uso *mobile*. In questo caso, però, invece che basarsi sul componente *Offcanvas* fornito da *React-Bootrstrap* per la realizzazione del pannello, questo è stato sviluppato *"from scratch"*. Questo approccio allo sviluppo ha portato ai seguenti problemi:
  - alla chiusura e all'apertura del pannello delle annotazioni l'intera pagina esegue molteplici *layout shift*;
  - il pannello ha grandezza fissa proporzionale alla larghezza della pagina, rendendo il *layout* non scalabile e togliendo molto spazio alla polizza;
  - le card contenute al suo interno non hanno limiti alla loro grandezza che dipende unicamente dal loro contenuto;
  - il contenuto delle card e come questo viene presentato deve essere rivisto, in quanto si vuole aumentare il numero di card presentate riducendo la quantità di scrolling verticale e minimizzare lo spazio utilizzato per la visualizzazione.

* **Pannello di selezione rapida**: durante questa ristrutturazione è stato rivisto completamente il funzionamento dell'annotazione: come accennato nel Capitolo 3, la selezione normale e quella rapida all'atto pratico non solo non hanno sostanziali differenze che le caratterizzino, ma rendono anche meno immediato l'utilizzo della funzione di annotazione all'utente. Infatti, non è previsto un "pulsante" che permetta di passare da selezione normale a quella rapida, bensì, il passaggio tra queste due modalità di annotazione avviene quando l'utente imposta completamente o resetta (anche solo parzialmente) le impostazioni di annotazione. Questo non solo complica l'utilizzo del software all'utente, ma lo complica anche a chi il software lo deve sviluppare e mantenere: mantenere due modalità di annotazione, infatti, per come funziona *React-Pdf-Highlighter*, obbliga ad avere codice duplicato e numerosi problemi di perdita dell'evidenziazione del testo, per fare alcuni esempi. È stata quindi discussa con il tutor aziendale una modalità alternativa di annotazione in modo tale da rendere l'esperienza d'uso più lineare e semplificare il mantenimento del software.
È stato deciso di eliminare l'annotazione normale e quella rapida, sostituendole con un'unica modalità di annotazione. È stato inoltre rivisto il flusso delle attività che deve fare l'utente per annotare.
Quando l'utente accede al tool di annotazione l'interfaccia è in modalità di "consultazione". In questa modalità le uniche operazioni che è possibile eseguire sono quelle che permettono la consultazione della polizza, come definito nel

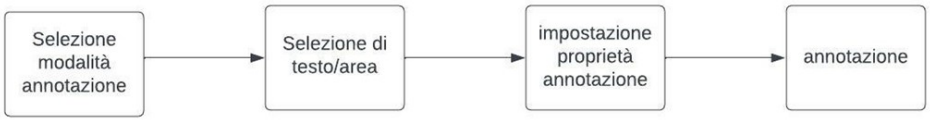

Figura 4.7: Rappresentazione grafica del flusso di azioni che deve compiere l'utente per effettuare una annotazione.

Capitolo 3. Successivamente, l'utente potrà scegliere di entrare in modalità di annotazione, ed una volta entrato in tale modalità, comparirà sulla destra il pannello di annotazione rapida. Di default sarà attiva la selezione del testo e sarà possibile *switchare* tra la modalità di selezione di testo o di aree attraverso un opportuno bottone. Una volta selezionata la modalità di selezione desiderata, l'utente potrà annotare il documento in maniera analoga all'annotazione rapida. In caso l'utente non abbia impostato la modalità di selezione oppure non abbia impostato i parametri dell'annotazione, viene notificato l'errore, in modo da poterlo correggere. Il flusso di attività diventa quindi quello esposto in Figura 4.7.

Design dei wireframe con Figma

Successivamente alla fase di analisi descritta nella Sezione 4.2.2, è stato necessario creare una serie di *wireframe*, in modo da poter trasformare le idee per la nuova interfaccia grafica in qualcosa di concreto, che potessero essere presentate e validate dal tutor aziendale prima di cominciare lo sviluppo del software. Per far ciò è stato usato *Figma*, un popolare software usato da web designer e grafici che permette anche di lavorare a distanza in maniera collaborativa attraverso un sistema analogo a quello usato da *Google Documents*. Il processo di sviluppo dei wireframe è stato iterativo: sono state effettuate una serie di proposte che hanno permesso di affinare il layout della nuova interfaccia, fino a che il layout proposto non è stato reputato maturo e validato dal tutor aziendale.

Ristrutturazione dei componenti esistenti

La ristrutturazione dei componenti è stata un'attività relativamente semplice dopo aver sviluppato i *wireframe*, ed ha avuto come output l'interfaccia grafica presentata in Figura 4.8. Infatti, ci sono stati pochi scostamenti da quest'ultimi e generalmente sono state modifiche minori che non hanno impattato in maniera significativa lo sviluppo.
L'idea alla base di questa ristrutturazione è stata suddividere i vari componenti in pannelli "flottanti" sopra il *Viewer* di PDF.js con posizionamento fisso all'interno della finestra e con la possibilità di essere nascosti quando non utilizzati. In questa maniera è stato possibile dedicare la maggior parte dello schermo alla polizza, differentemente dalla precedente implementazione, dove la maggior parte dello schermo era costantemente occupata dai pannelli delle annotazioni e di annotazione rapida. Questa interfaccia, inoltre, scala molto meglio della precedente quando la si ridimensiona: infatti il software rimane tranquillamente usabile anche quando non viene usato a finestra intera.
Si vedrà ora maggiormente nel dettaglio come sono stati ristrutturati i principali

componenti.

* **La toolbar** è stata spostata in un pannello flottante in basso a destra e contiene al suo interno una serie di pulsanti che consentono di accedere a tutte le funzionalità di consultazione della polizza oltre che a quelle necessarie per entrare in modalità di annotazione e per poter scegliere la tipologia di selezione. Così è sempre chiaro all'utente che tipologia di annotazione sta effettuando. Come si può notare dalla Figura 4.9, alcune icone risultano disabilitate: questo perché non tutte le funzionalità possono essere usate in ogni momento: per esempio, se il tool non è in modalità di annotazione, non è possibile selezionare il tipo di selezione, oppure se il tool sta mostrando la pagina di sintesi del contratto, non è possibile effettuare lo zoom sul documento pdf. La toolbar, da sinistra a destra, contiene:
  - un <input type="number"> per navigare ad una pagina specifica all'interno della polizza;
  - i pulsanti per effettuare lo zoom del documento;
  - il pulsante per entrare in modalità annotazione;
  - il pulsante per effetttuare l'annotazione di aree;
  - il pulsante per effettuare l'annotazione di testo;
  - il pulsante per visualizzare la sintesi del contratto;
  - il pulsante per accedere al modale di modifica delle unità di rischio;
  - il pulsante per mostrare la legenda pop up.
* **Il pannello delle annotazioni**, come evidenziato in Figura 4.10, ora contiene anche un *Collapsable React-Bootrstrap* che può essere aperto con un click per visualizzare le *dropdown* per il filtraggio. In questa maniera si risparmia schermo quando non si sta utilizzando attivamente il filtraggio. Il pannello di filtraggio si può vedere in Figura 4.11.
  Per rendere sempre evidenti i filtri applicati, quando questi sono presenti ed il *Collapsable* del filtraggio è in forma ridotta, vengono aggiunti al di sopra del suddetto *Collapsable* dei *toast* che indicano quali filtri sono correntemente attivi. Un esempio di ciò lo si può vedere in Figura 4.12.
* **Le card**, di cui se ne può vedere un esempio in Figura 4.13, sono state riviste riducendo leggermente il font e i margini tra gli elementi. I campi relativi al testo originale e allo screen dell'area selezionata dell'annotazione ora compaiono all'interno di componenti *Collapsable* che possono essere aperti attraverso il click del mouse. Il commento associato all'annotazione, se presente, viene sempre mostrato sotto le proprietà dell'annotazione. Se il commento risulta essere più lungo di cinque righe, allora questo viene troncato alla quinta riga e può essere visualizzato nella sua interezza attraverso un click del mouse. Questa funzionalità è fornita dalla libreria *ReadMoreReact*, a cui viene delegata la responsabilità dell'accessibilità.
* **Il pannello di annotazione rapida**, mostrato in Figura 4.14, ha subito pochi cambiamenti rispetto all'implementazione originale. Il principale cambiamento è che ora il pannello di annotazione dipende da *Toolbar* in quanto è quel componente che è incaricato di mostrare il pannello di annotazione in base all'indicazione se il tool è in modalità di annotazione. In questa maniera, se non si sta annotando

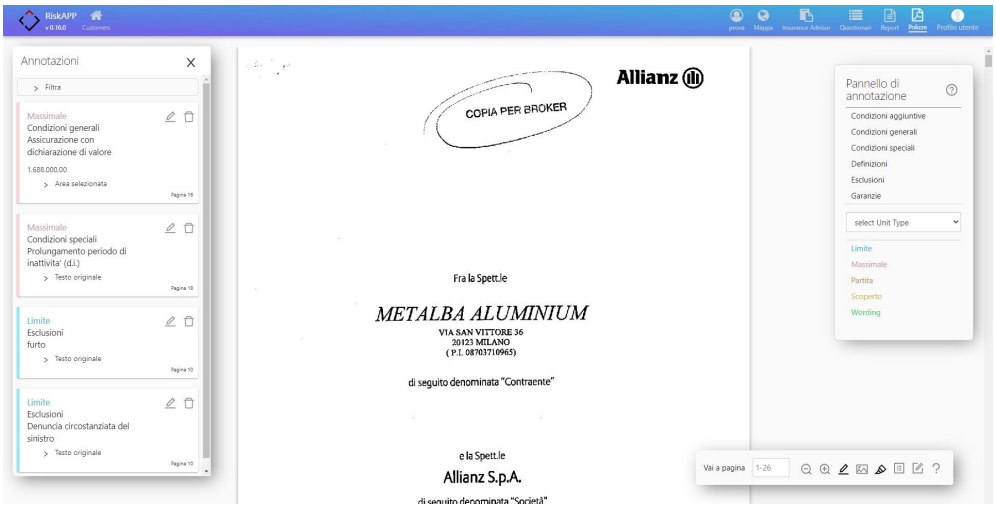

Figura 4.8: La nuova interfaccia grafica frutto della ristrutturazione.

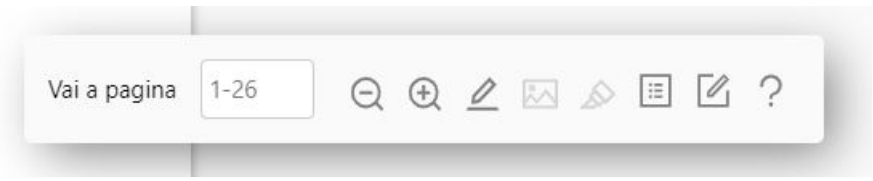

Figura 4.9: La nuova toolbar.

la polizza, il pannello viene automaticamente nascosto, risparmiando spazio a schermo.

* **Il modale di aggiunta delle unità di rischio**, mostrato in Figura 4.15, ha mantenuto la struttura che aveva nella precedente versione del software, a cui sono state aggiunte le opportune label per renderne più chiaro l'utilizzo. Una volta scelta una nuova unità di rischio, compare un pulsante che permette di confermare l'unità di rischio. È stato aggiunto al modale un pop up di aiuto che l'utente può visualizzare per avere maggiori spiegazioni su come aggiungere o modificare una unità di rischio.

* **Il modale di modifica delle unità di rischio**, mostrato in Figura 4.16, ora consente di selezionare direttamente l'unità di rischio che si intende modificare. È possibile effettuare un filtraggio delle unità per sezione di rischio in modo da velocizzarne la ricerca. Una volta selezionata una unità di rischio, è possibile rinominarla e confermare la modifica.

4.2.3 Considerazione sull'accessibilità

Un aspetto chiave dello sviluppo web è senza dubbio la creazione di interfacce grafiche che siano inclusive per ogni categoria di utente. Nella ristrutturazione di questo software, questo principio è stato ampiamente preso in considerazione, e per raggiungere lo scopo si è cercato di seguire le linee guida per l'accessibilità e le *best practices* raccomandate

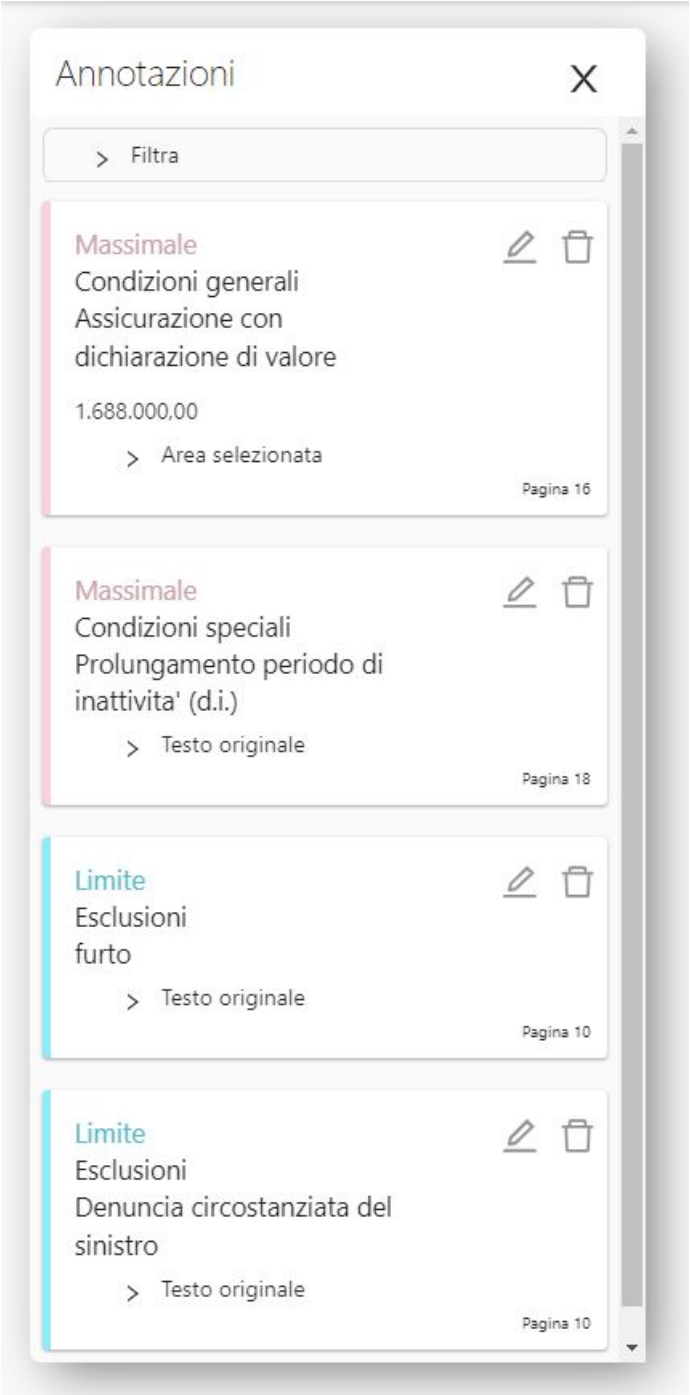

Figura 4.10: Il nuovo pannello delle annotazioni.

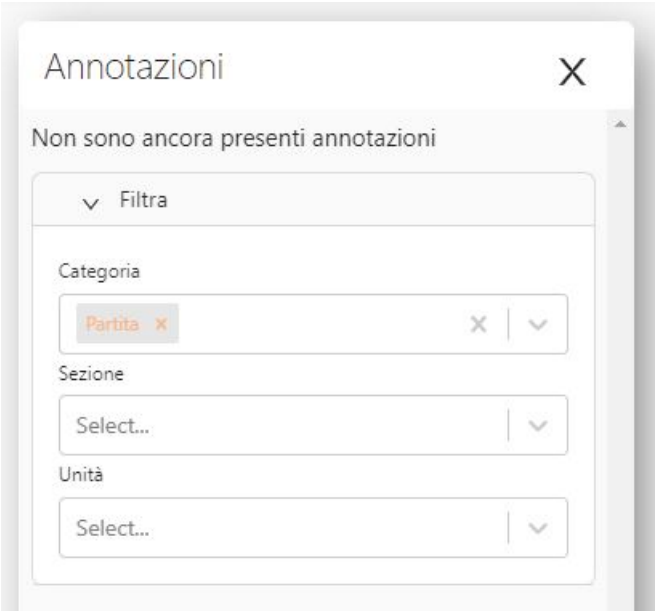

Figura 4.11: Come viene presentato il menù di filtraggio delle annotazioni.

Figura 4.12: Toast rappresentanti i filtri attivi.

Figura 4.13: Card riviste.

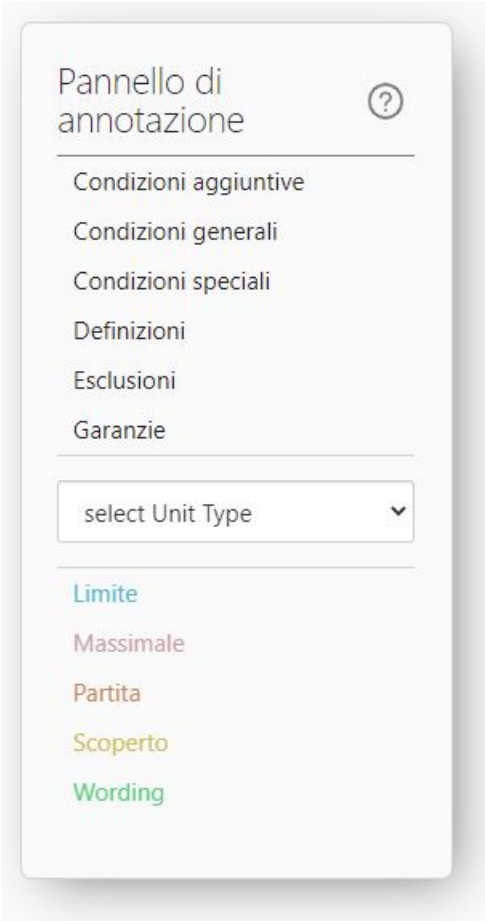

Figura 4.14: Il nuovo pannello di annotazione rapida.

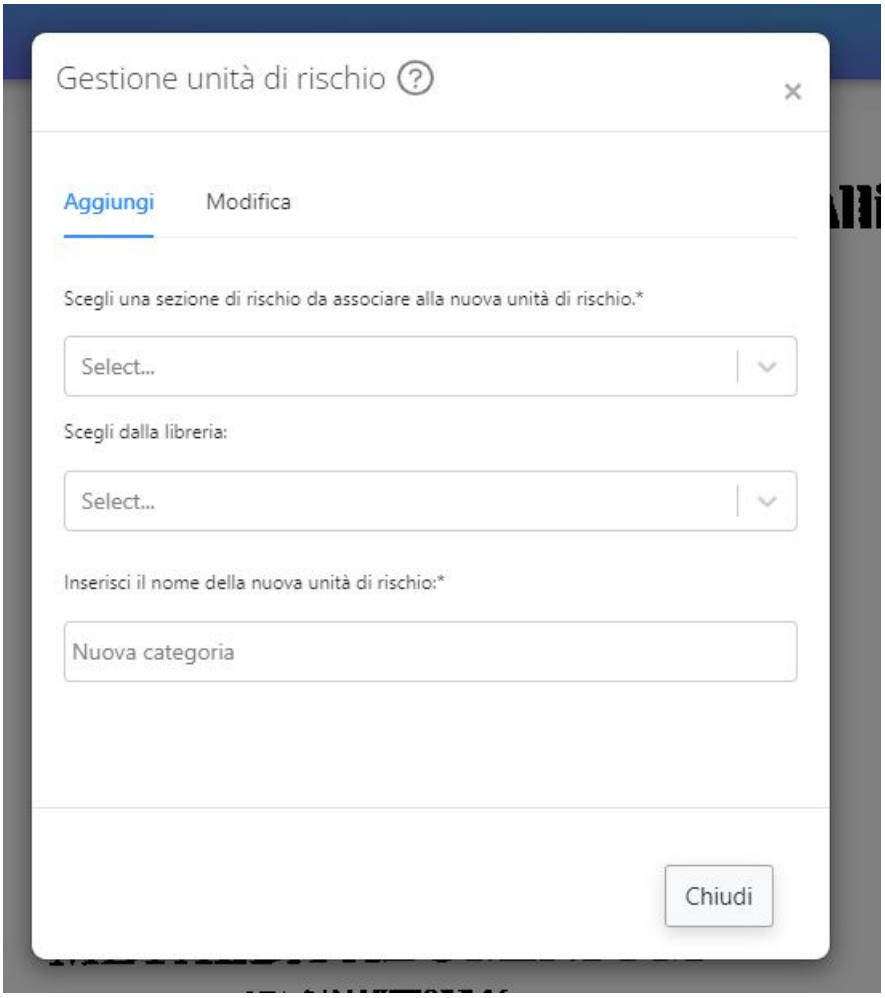

Figura 4.15: Il nuovo modale per l'aggiunta di una unità di rischio.

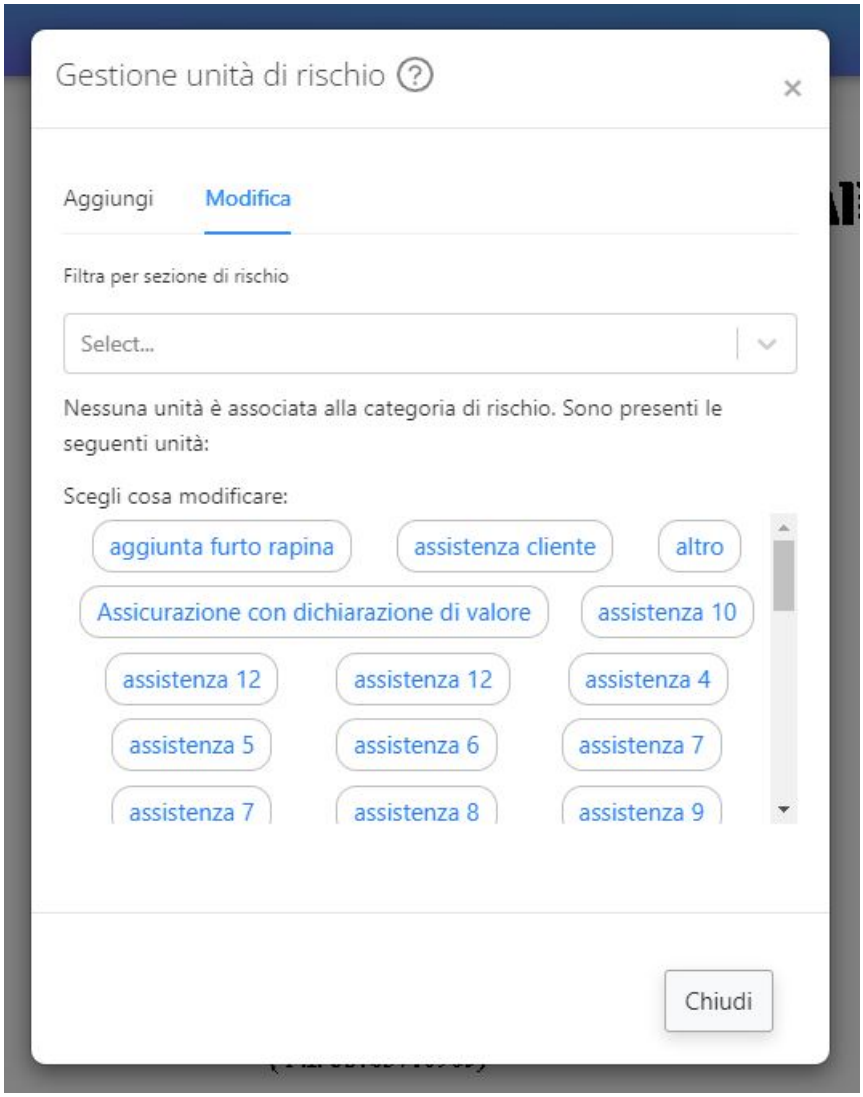

Figura 4.16: Il nuovo modale di modifica delle unità di rischio.

nella documentazione di ogni tecnologia adottata. L'accessibilità del tool è stata valutata con estensioni come *AXE* [3] e *Google Lighthouse* [19], in modo da controllare che i colori avessero contrasto adeguato, e che la struttura della pagina fosse navigabile attraverso uno screen reader. È necessario fare una precisazione sui colori: come si può facilmente notare in Figura 4.14, i colori usati per identificare le diverse sezioni di rischio non rispecchiano il contrasto minimo necessario per superare i test dei tool per la verifica dell'accessibilità. Questo perchè quei colori sono gestiti a livello backend e sarebbe sufficente modificarli attraverso una *query* al database. Non disponendo però dell'accesso al database e quindi non potendo eseguire tale modifica personalmente, ho segnalato al tutor aziendale il problema ma nessuna azione correttiva è stata eseguita prima della fine del periodo di tirocinio. Il software, esclusa la questione colori sopracitata, non può essere considerato accessibile al 100% in quanto, la navigazione all'interno del Viewer di PDF.js e l'annotazione della polizza non possono essere effettuate tramite tastiera. In ogni caso, è importante far notare come l'accessibilità sia solo uno dei tanti fattori che rendono un software usabile, e che ci sia molto altro da tenere in considerazione quando si vuole sviluppare una UI di qualità. La semplicità d'uso, l'immediatezza e la capacità per un'azienda di fornire un software che risponda in maniera efficace alle esigenze dei propri clienti sono fattori che contribuiscono in egual misura alla buona riuscita di un prodotto software.

4.2.4 Testing e documentazione della UI

Per testare l'interfaccia, il tutor aziendale ha consigliato di seguire un approccio *manuale*, sia per mantenere la compatibilità con i tempi previsti dal Piano di Lavoro, sia perchè i test a livello di front-end sono classificati come obiettivi facoltativi per il raggiungimento del progetto di tirocinio. Il testing manuale consiste nell'effettuare una sequenza di operazioni stabilita a priori per verificare che il software si comporti nella maniera attesa. Come accennato nella Sezione 4.2.3, sono stati usati tool di validazione come *AXE* e *Google Lighthouse* per valutare l'accessibilità dell'interfaccia. Per lo sviluppo ed il refactoring del codice JavaScript sono stati usati *linter* e *prettifier*, software che aiutano lo sviluppatore a scrivere codice che rispetti le *best practices* riguardo leggibilità e manutenibilità. Il codice dell'interfaccia grafica è stato accompagnato da numerosi commenti che spiegano il ruolo di ogni componente e le motivazioni delle scelte implementative effettuate.

Capitolo 5

Conclusioni

In questo capitolo viene presentata una valutazione retrospettiva sullo stage.

5.1 Consuntivo orario

Il progetto di stage prevede una durata di 320 ore lavorative per un totale di undici crediti formativi universitari. Lo stage è cominciato il giorno 8 marzo 2022 ed è terminato il giorno 8 maggio 2022. Il numero di ore di lavoro effettivamente svolte ammonta a 320, come si può evincere dalla Tabella 5.1. Le ore effettuate comprendono l'attività di ricerca, di sviluppo e produzione della documentazione a corredo del codice. Il lavoro è stato svolto principalmente da casa con incontri settimanali in azienda, dove è stato possibile avere un confronto diretto su quanto prodotto durante la settimana. La pianificazione effettuata inizialmente si è rivelata efficace, in quanto è risultata aderire precisamente con le tempistiche richieste per lo svolgimento del progetto. L'unico scostamento orario rispetto a quanto preventivato nel Piano di Lavoro è quello relativo alla produzione dei test, che, come spiegato nella Sezione 4.2.4, sono stati esclusivamente manuali, rendendo possibile dedicare più tempo all'attività di revisione della UI.

5.2 Raggiungimento degli obiettivi

Tutti gli obiettivi obbligatori e desiderabili fissati nel Piano di Lavoro alla Sezione 2.4 sono stati completati con successo, diversamente da quello facoltativo, in quanto i test a livello di front end sono stati effettuati esclusivamente con un approccio manuale, come spiegato nella Sezione 4.2.4.
Un rescoonto del raggiungimento degli obiettivi si può trovare alla Tabella 5.2.

5.3 Valutazioni personali

Terminato lo stage, posso reputarmi pienamente soddisfatto del risultato ottenuto. Dal punto di vista aziendale, sono riuscito a portare a termine un progetto che rispecchia i desideri dell'azienda, nonostante la difficoltà del doversi approcciare ad un software complesso, privo di documentazione e dalle molteplici problematiche che hanno richiesto diverse ore di debug per poter essere risolte. Mi reputo soddisfatto anche dal punto

Descrizione attività	**Durata prevista**	**Durata effettiva**
Formazione React	18	18
Formazione Redux	18	18
Ricerca e studio di nuove librerie per la manipolazione dei documenti pdf	36	36
Analisi del software preesistente	16	16
Progettazione delle modifiche ai componenti preesistenti	16	16
Progettazione della nuova UI	16	16
Progettazione delle nuove funzionalità	10	10
Stesura documentazione relativa ad analisi e progettazione	6	6
Implementazione modifiche ai componenti	82	82
Revisione della UI	64	74
Produzione dei test	10	0
Collaudo	38	38
Verifica documentazione finale	64	64
Presentazione della piattaforma agli stakeholders e live demo	10	10
	Ore totali previste	**Ore totali effettive**
	320	320

Tabella 5.1: Consuntivo orario.

Obiettivi	Descrizione	Raggiunto
O01	Le componenti che gestiscono le annotazioni devono essere ristrutturate per funzionare con il nuovo schema dati adottato nel backend;	Sì
O02	Le componenti ristrutturate devono essere riscritte utilizzando sintassi e buone pratiche che ne semplifichino l'implementazione e ne migliorino leggibilità e manutenibilità;	Sì
O03	Devono essere riviste collocazione e aspetto visivo della toolbar per la navigazione del pdf con le varie annotazioni;	Sì
O04	Le modifiche effettuate alla UI devono mantenerla responsiva, accessibile e semplice da utilizzare per l'utilizzatore finale;	Sì
D01	Deve essere aggiunta una shortcut per l'attivazione della funzionalità di annotazione del pdf;	Sì
D02	Devono essere riviste le librerie adottate per la gestione dei pdf, se ne conseguono benefici a livello di prestazioni e/o di manutenibilità del software;	Sì
D03	Deve essere fornita documentazione delle scelte implementative adottate nel refactoring di componenti e UI;	Sì
F01	devono essere presenti dei test a livello di frontend.	No

Tabella 5.2: Tabella di raggiungimento degli obiettivi.

di vista personale: ho avuto modo di mettermi alla prova in un ambito, quello dello sviluppo web, in cui vorrei specializzarmi nel periodo post-laurea e verso il quale nutro una grande passione. Inoltre, l'ambiente giovane e dinamico che ho trovato in RiskApp è stato particolarmente piacevole e stimolante, e quando mi è stato offerto un posto di lavoro come sviluppatore front-end ho avuto la conferma di aver svolto un lavoro apprezzato.

5.4 Conoscenze acquisite

Il tirocinio mi ha permesso di acquisire una buona conoscenza relativa alle moderne tecnologie web ed alla maniera in cui queste vengono utilizzate per lo sviluppo di web app. Questa è stata un'ottima esperienza complementare a corsi come Tecnologie Web, Ingegneria del Software, Tecnologie Open Source ed Altri Paradigmi di Programmazione. I corsi sopracitati hanno permesso di approcciarmi a questo progetto di stage con un'ottica diversa, che certamente non avrei avuto se non li avessi frequentati: grazie al progetto di Ingegneria del Software, dove ho avuto modo di effettuare numerose pianificazioni, sono riuscito ad organizzare il lavoro da svolgere in maniera realistica ed efficace; grazie al corso di Tecnologie Web, ho potuto sviluppare un'interfaccia grafica sapendo che l'accessibilità di un software avviene per costruzione e non per revisioni successive; grazie al corso di Tecnologie Open Source, ho potuto utilizzare con competenza gli strumenti necessari allo sviluppo di software in maniera collaborativa; attraverso quanto visto durante il corso di Altri Paradigmi di Programmazione, sono riuscito ad approcciarmi facilmente all'uso delle più recenti versioni di JavaScript. Oltre alle conoscenze tecniche, però, con questo stage ho avuto modo di sviluppare anche quelle che vengono chiamate *soft skills*. Ho potuto "toccare con mano" quanto la comunicazione e la collaborazione all'interno di un ambiente lavorativo siano importanti per portare a buon fine un progetto.

Glossario

API Application Programming Interface. Sono funzioni che fungono da interfaccia tra due software. La chiamata di una API permette al software chiamante di usare la funzionalità esposta dalla API del software chiamato. 19, 47

binding Binding, letteralmente "legare", un evento JavaScript ad una funzione, permette di invocare suddetta funzione ogni volta che viene scatenato un determinato evento [15]. 26, 47

box model Nello sviluppo web, il modello box CSS si riferisce a come gli elementi HTML sono modellati nei motori del browser e come le dimensioni di quegli elementi HTML sono derivate dalle proprietà CSS [4]. 31, 47

breaking changes Modifica ad un software che ne provoca il malfunzionamento [5]. 26, 47

bubbling Proprietà degli eventi JavaScript che consiste nel risalire il DOM fino all'elemento radice o fino a che non incontrano un elemento presente nel cammino che porta all'elemento radice, legato a quell'evento e che lo blocca [14]. 30, 47

callback In programmazione, una callback è, in genere, una funzione, o un "blocco di codice" che viene passata come parametro ad un'altra funzione. In particolare, quando ci si riferisce alla callback richiamata da una funzione, la callback viene passata come argomento ad un parametro della funzione chiamante. In questo modo la chiamante può realizzare un compito specifico (quello svolto dalla callback) che non è, molto spesso, noto al momento della scrittura del codice [13]. 18, 47

chaining operator È un operatore che permette allo sviluppatore di leggere il valore di una proprietà che è posizionata all'interno di una catena di prototipi collegati tra loro senza controllare che ogni riferimento nella catena sia valido [7]. 26, 47

changelog È il registro che contiene tutte le modifiche apportate a un progetto. 24, 47

code on demand È una tecnologia che consente ad un server di inviare codice eseguibile su richiesta ad un client [8]. 3, 47

code splitting È il processo di suddivisione dell'insieme del codice che compone un software in un insieme di bundle o moduli indipendenti [9]. 3, 47

Codebase In informatica, con *codebase* si intende l'intera collezione di codice sorgente usata per costruire una particolare applicazione o un particolare componente [10]. 2, 5, 27, 47

CD In ingegneria del software, la consega continua o continuous delivery è una pratica che consiste nel rilascio frequente di software, assicurandosi che quest'ultimo sia rilasciabile in ogni momento e senza doverlo fare manualmente [11]. 2, 48

CI In ingegneria del software, l'integrazione continua o continuous integration è una pratica che consiste nell'allineamento frequente degli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso [12]. 2, 48

cross-browser Capacità di visualizzare e godere a pieno di tutte le funzionalità di un applicativo web in qualsiasi browser. 25, 48

DOM In informatica il Document Object Model (spesso abbreviato come DOM), letteralmente modello a oggetti del documento, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti. È lo standard ufficiale del W3C per la rappresentazione di documenti strutturati in maniera da essere neutrali sia per la lingua che per la piattaforma. È inoltre la base per una vasta gamma di interfacce di programmazione delle applicazioni, alcune di esse standardizzate dal W3C [13]. 20, 48

gesture un gesto effettuato sul dispositivo di puntamento (tipicamente negli schermi touchscreen) che permette l'utilizzo di determinate funzionalità di un applicativo. 24, 25, 48

IDE Integrated Developement Enviroment. È un software che integra al suo interno diversi strumenti che facilitano lo sviluppo e il debug del codice. 2, 48

ISO 19005 Questo standard definisce un formato (PDF/A) per l'archiviazione nel lungo periodo di documenti elettronici ed è basato sulla versione 1.4 del formato PDF di Adobe, implementato in Adobe Acrobat versione 5 e successive [45]. 19, 48

layout shift Effetto visivo che consiste nel ridimensionamento delle parti di una pagina. Questo effetto può risultare particolarmente sgradevole se avviene inaspettatamente e disorienta l'utente. 31, 48

mobile-friendly Ottimizzato per i dispositivi mobile. 26, 48

serverless Con il termine serverless (dall'inglese senza server) si intende un modello di esecuzione cloud dove il provider del servizio cloud alloca le risorse della macchina appena queste vengono richieste. Quando un app non è in uso, nessuna risorsa viene consumata e il prezzo è basato solo sulle risorse utilizzate [42]. 25, 48

Software as a Service è un modello di servizio del software applicativo dove un produttore di software sviluppa, opera (direttamente o tramite terze parti) e gestisce un'applicazione web. Con l'utilizzo di SaaS, il software utilizzato non è installato localmente, ma viene messo a disposizione dei clienti tramite una connessione Internet (previo eventuale abbonamento) [43]. 1, 48

tooltip Breve messaggio a comparsa. 12, 48

UI User Interface. 2, 27, 48

WAI-ARIA Acronimo per "Web Accessibility Initiative - Accessible Rich Internet Applications suite of web standards". Sono un insieme di specifiche che definiscono come aumentare l'accessibilità dei contenuti dinamici e dei componenti per interfacce utente in ambito web [2]. 22, 49

wireframe Nel web design il wireframe rappresenta il modello iniziale di rappresentazione di un sito web [48]. 31, 49

Bibliografia

Siti web consultati

[1] *Apache 2.0: home page.* URL: https://www.apache.org/licenses/LICENSE-2.0.

[2] *ARIA standar guidelines: definition.* URL: https://it.wikipedia.org/wiki/Wai-aria (cit. a p. 49).

[3] *Axe accessibility: home page.* URL: https://www.deque.com/axe/browser-extensions/ (cit. a p. 41).

[4] *Box model: definition.* URL: https://en.wikipedia.org/wiki/CSS_box_model (cit. a p. 47).

[5] *Breaking change: definition.* URL: https://en.wiktionary.org/wiki/breaking_change (cit. a p. 47).

[6] *Callback: definition.* URL: https://it.wikipedia.org/wiki/Callback.

[7] *Chaining Operator: documentation.* URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Optional_chaining (cit. a p. 47).

[8] *Code on demand: definition.* URL: https://en.wikipedia.org/wiki/Code_on_demand#:~:text=In%20distributed%20computing%2C%20code%20on%20demand%20is%20any,language%20for%20the%20Flash%20Player%2C%20and%20JavaScript.%20 (cit. a p. 47).

[9] *Code splitting: definition.* URL: https://reactjs.org/docs/code-splitting.html (cit. a p. 47).

[10] *Codebase: definition.* URL: https://tinyurl.com/46paj8et (cit. a p. 48).

[11] *Continuous Delivery: definition.* URL: https://tinyurl.com/3s4n95jd (cit. a p. 48).

[12] *Continuous Integration: definition.* URL: https://tinyurl.com/55zwwrmx (cit. a p. 48).

[13] *DOM: definition.* URL: https://it.wikipedia.org/wiki/Document_Object_Model (cit. alle pp. 47, 48).

[14] *Event bubbling: documentation.* URL: https://developer.mozilla.org/en-US/docs/Web/API/Event/bubbles?retiredLocale=it (cit. a p. 47).

[15] *Event handling: documentation*. URL: https://developer.mozilla.org/en-US/docs/Web/Events/Event_handlers (cit. alle pp. 26, 47).

[16] *Fidelidade: home page*. URL: https://www.fidelidade.pt/EN/fidelidade/about-us/Paginas/about-us.aspx (cit. a p. 1).

[17] *GitHub Octoverse: top languages over the years*. URL: https://octoverse.github.com/#top-languages-over-the-years (cit. a p. 3).

[18] *Github: home page*. URL: https://github.com/.

[19] *Google Lighthouse: home page*. URL: https://github.com/GoogleChrome/lighthouse (cit. a p. 41).

[20] *Intellij Idea: home page*. URL: https://www.jetbrains.com/idea/ (cit. a p. 2).

[21] *JavaScript: Mozilla documentation*. URL: https://developer.mozilla.org/en-US/docs/Web/javascript (cit. a p. 3).

[22] *Manifesto Agile*. URL: http://agilemanifesto.org/iso/it/ (cit. a p. 1).

[23] *medium.com: lifecycle methods*. URL: https://medium.com/@saharshgoyal/react-16-new-life-cycle-methods-inside-out-fdd269c43ccd (cit. a p. 19).

[24] *medium.com: what Redux state management is*. URL: https://segunsubair.medium.com/redux-react-state-management-23be854799d2 (cit. a p. 20).

[25] *Node: home page*. URL: https://nodejs.org/en/ (cit. a p. 3).

[26] *npmjs.com: PDF.js-dist*. URL: https://www.npmjs.com/package/pdfjs-dist (cit. a p. 19).

[27] *p&v: home page*. URL: https://www.pv.be/fr/home (cit. a p. 1).

[28] *PDF.js: Display layer apy documentation*. URL: https://github.com/mozilla/pdf.js/blob/master/src/display/api.js (cit. a p. 19).

[29] *PDF.js: Github project*. URL: https://github.com/mozilla/pdf.js.

[30] *PDF.js: releases*. URL: https://github.com/mozilla/pdf.js/releases?page=1 (cit. a p. 24).

[31] *PDFTron: home page*. URL: https://www.pdftron.com/ (cit. a p. 23).

[32] *pdftron.com: Comparison between PDF.js and proprietary technologies*. URL: https://www.pdftron.com/blog/pdf-js/pdf-js-build-vs-buy/ (cit. a p. 24).

[33] *Polyglot.js: home page*. URL: https://airbnb.io/polyglot.js/ (cit. a p. 29).

[34] *Pycharm: home page*. URL: https://www.jetbrains.com/pycharm/ (cit. a p. 2).

[35] *React-Bootstrap: home page*. URL: https://react-bootstrap.github.io/.

[36] *React-Pdf-Highlighter: Github project*. URL: https://github.com/agentcooper/react-pdf-highlighter (cit. a p. 20).

[37] *Reactjs: home page*. URL: https://reactjs.org/ (cit. a p. 3).

[38] *reactjs.org: Thinking in React*. URL: https://it.reactjs.org/docs/thinking-in-react.html (cit. a p. 18).

[39] *Redux: home page*. URL: https://redux.js.org/ (cit. a p. 3).

[40] *Riskapp: home page*. URL: https://www.riskapp.it/ (cit. a p. 1).

[41] *Scrum: what is scrum*. URL: https://www.scrum.org/resources/what-is-scrum/ (cit. a p. 2).

[42] *Serverless: definition*. URL: https://it.wikipedia.org/wiki/Serverless (cit. a p. 48).

[43] *Software as a Service: definition*. URL: https://it.wikipedia.org/wiki/Software_as_a_service (cit. a p. 48).

[44] *StackOverflow: most popular programming languages in 2021*. URL: https://insights.stackoverflow.com/survey/2021#programming-scripting-and-markup-languages (cit. a p. 3).

[45] *Standard ISO 19005: definition*. URL: it.wikipedia.org/wiki/PDF/A (cit. a p. 48).

[46] *UnipolSai: home page*. URL: https://www.unipolsai.it/homepage (cit. a p. 1).

[47] *Webpack: home page*. URL: https://webpack.js.org/ (cit. a p. 3).

[48] *Wireframe: definition*. URL: https://it.wikipedia.org/wiki/Wireframe_(web_design) (cit. a p. 49).