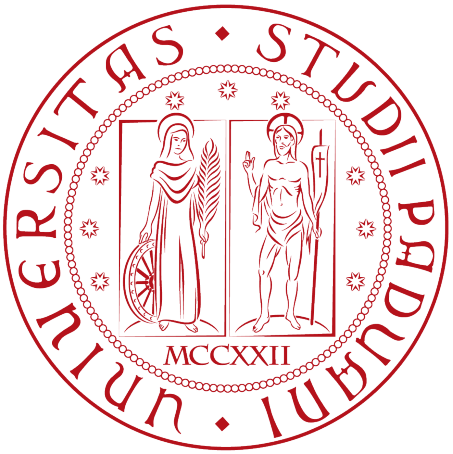
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in Informatica



Realizzazione di una piattaforma di gestione della knowledge base in un contesto di generative search

Tesi di laurea

Relatore

Prof. Lamberto Ballan

Laureando

Samuel Scarabottolo

Anno Accademico 2022-2023

We Can't Change What's Done, We Can Only Move On
— Arthur Morgan

Dedicato ai miei cari genitori

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di circa trecento ore, dal laureando Samuel Scarabottolo presso l'azienda Siav S.p.A di Rubano (PD). Gli obbiettivi da raggiungere erano molteplici.
In primo luogo era richiesto di inquadrare il problema con la stesura di un documento di specifica dei requisiti, in cui venivano descritti i requisiti funzionali del sistema da realizzare. In secondo luogo era richiesta la progettazione e l'implementazione di *store* per la conservazione di documenti, che implementa funzionalità per recuperare documenti rilevanti per una certa *query* e la gestione degli accessi per gli utenti a documenti diversi. Il tutto di quanto descritto doveva essere gestito tramite microservizi per l'esposizione delle funzionalità citate precedentemente tramite servizi REST. Terzo ed ultimo obbiettivo era la stilatura della documentazione tecnica e utente per la corretta comprensione del sistema realizzato.

"Endure and Survive"

— Joel Miller

Ringraziamenti

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

Desidero ringraziare inoltre i miei genitori, che mi hanno accompagnato e sostenuto per questi meravigliosi 3 anni. Senza di loro tutto questo non sarebbe mai stato possibile e per questo dedico loro questa tesi, frutto anche dei loro sacrifici.

Vorrei esprimere la mia gratitudine anche al mio tutor aziendale Luca Corò per l'aiuto e disponibilità fornitami e l'azienda Siav S.p.A che ha reso possibile questa esperienza formativa.

Ringrazio i miei compagni di università per tutti i bellissimi anni passati insieme, complici di svariate avventure e serate passate a studiare.

Infine vorrei ringraziare tutti i miei amici che mi hanno sostenuto e aiutato per tutto questo percorso durato 3 anni, insieme, come una famiglia.

Padova, Settembre 2023 Samuel Scarabottolo

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Socrate è una *web application*[1] che permette l'amministrazione ed interrogazione di *chatbot* la cui Knowlege Base può essere situata in sistemi eterogenei[2], con due casistiche principali, determinate dalla natura del frontend impiegato:

- *frontend* integrato in sistema terzi (Archiflow, Silloge, Localstore)
- *frontend standalone* (es. bot Teams)

e altri due scenari determinati dalla modalità di indicizzazione dei documenti necessari a creare il contesto su cui il bot elabora la propria risposta:

- **Indicizzazione off-line:** i documenti vengono indicizzati in *batch* prima di attivare il *chatbot*;
- **Indicizzazione on-line on-the-fly:** dalla quale si possono identificare ulteriori due scenari:
  - l'utente interroga l'intera base documentale. In questo caso è necessaria una fase di *search* perliminare dei documenti rilevanti presenti nell'intera base documentale;
  - l'utente, all'interno del sistema di gestione documentale, seleziona una scheda ed interroga i documenti presenti in essa. Per questa casistica non è necessaria una fase di *search*.

Per questo progetto è stata affidata l'implementazione di **Localstore** API.

1.1 L'azienda

Siav S.p.A è un'azienda informatica specializzata nella dematerializzazione, nella gestione elettronica dei documenti e nei processi digitali. Fondata nel 1990 a Rubano (PD) dall'attuale presidente Alfieri Voltan, ad oggi Siav è la prima azienda

[1] web application: applicazione software eseguibile sul web ospitate su server remoti. Gli utenti possono interagire con esse attraverso il browser, senza dover installare nulla sul proprio dispositivo

[2] sistemi eterogenei: ambienti in cui diversi componenti o dispositivi hanno caratteristiche, piattaforme o configurazioni diverse

italiana nel settore dell'*Enterprise Content Management*. Offre *software*, soluzioni *cloud* e servizi *outsourcing*[3] per la gestione elettronica dei documenti, il protocollo informatico, il *workflow management*[4], la fatturazione elettronica e la conservazione digitale. Inoltre Siav conta oltre 4000 clienti nel mercato pubblico e privato, ed è presente con le proprie sedi a Padova, Milano, Genova, Bologa e Roma, opera anche in Svizzera e Romania.

Figura 1.1: L'azienda: Siav S.p.A

1.2 Organizzazione del testo

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;
- alcuni termini specifici sono corredati di spiegazione a piè di pagina;
- i termini in lingua straniera o facenti parti del gergo tecnico sono evidenziati con il carattere *corsivo*.

1.3 Struttura del documento

Di seguito viene mostrata la struttura del documento. Ogni capitolo viene accompagnato da una piccola descrizione rappresentativa di quanto contiene:

1. Introduzione: in questo capitolo vengono introdotti lo scopo dello stage, l'azienda che ha reso possibile ciò e come viene organizzato il testo all'interno del documento;
2. Lo stage nel dettaglio: in questo capitolo vengono spiegati tutti i dettagli riguardanti lo stage e dunque il progetto affidato;
3. Strumenti e tecnologie: in questo capitolo vengno elencati tutti gli strumenti utilizzati e descritte nel dettaglio tutte le tecnologie utili per la codifica del prodotto. In particolare ci si sofferma sui vantaggi e svantaggi di ognuna;
4. Analisi dei requisiti: in questo capitolo vengono elencati i requisiti suddivisi per soggetto a cui fanno riferimento;

[3] outsourcing: pratica per la quale un'azienda affida alcune sue funzioni ad esterni

[4] workflow management: consiste nell'assegnare compiti, risorse e processi in un flusso logico, garantendo l'efficienza e la tracciabilità delle operazioni

5. Progettazione e codifica: in questo capitolo vengono spiegate tutte le scelte progettuali e conseguentemente come è stata affrontata la fase di codifica sia del *backend* che del *frontend*;

6. Verifica e validazione: in questo capitolo vengono spiegate le fasi di verifica e validazione del codice sviluppato;

7. Conclusione: in questo capitolo si fa un resoconto di quanto è stata l'esperienza di stage.

Capitolo 2

Lo stage nel dettaglio

2.1 Motivazione dello stage

Essendo Siav S.p.A una delle principali aziende informatiche specializzate nella dematerializzazione e gestione elettronica dei documenti, ed inoltre sempre più lampante l'avanzamento tecnologico sotto l'aspetto dell'intelligenza artificiale, sono sorte diverse esigenze per stare al passo con i tempi. Uno dei problemi più comuni del lavoratore è proprio la ricerca all'interno dei documenti delle informazioni necessarie. Talvolta risulta quasi impossibile trovare le risposte e si passano ore per cercarle. Nasce dunque "Socrate", una *web application* in grado di permettere l'amministrazione di *chatbot* la cui *knowledge base* può essere situata in sistemi eterogenei. Per questo progetto si è prefissato di sviluppare "Localstore API", ovvero uno store *standalone*[1] che conserva direttamente i documenti caricati dall'utente e gli store nei quali essi sono inseriti. I chatbot fungono da veri e propri "servizi cliente" che rispondono in linguaggio umano ad ogni domanda che viene posta, sui documenti inseriti.

2.2 Descrizione del progetto di stage

Lo stage si è collocato all'interno di un progetto più ampio per la realizzazione di una *web application*, la quale permette l'amministrazione ed interrogazione di *chatbot* in grado di rispondere alle domande degli utenti. Ogni qualvolta si vuole creare un *chatbot* la cui *knowledge base* sia formata da documenti provenienti da fonti miste (ad esempio, come in questo caso, documenti caricati manualmente da frontend oppure presi da wiki, file su share di rete ecc.) risulta opportuno appoggiarsi ad uno strato di persistenza[2] (es. Weaviate, Elsaticsearch, Solr ecc.), pertanto, dovendo mettere a disposizione una serie di funzionalità tramite un'interfaccia comune, abbiamo bisgogno di implementare uno specifico "wrapper[3]" a seconda dell'opzione

[1] standalone: applicazione o un dispositivo che è completamente autosufficiente e non richiede connessioni esterne o dipendenze da altri sistemi per funzionare.

[2] strato di persistenza: (o livello di persistenza) e' un componente, o un'astrazione all'interno di un'applicazione software che gestisce la persistenza dei dati

[3] wrapper: componente o una struttura che avvolge o racchiude un'altra entità, fornendo un'interfaccia semplificata o un nuovo livello di astrazione per interagire con essa.

di persistenza scelta. Nella sua prima implementazione è stato usato il vector database Weaviate per la persistenza dei vettori, relativi ai documenti caricati. Per questo progetto è stato possibile, alla luce di uno studio approfondito, realizzare un'interfaccia per esporre le funzionalità di creazione, condivisione, eliminazione, popolazione ed interrogazione delle *knowledge bases*.

2.3 Organizzazione dello stage

Di seguito vengono descritte tutte le informazioni riguardanti l'organizzazione oraria con la suddivisione dei lavori e le varie fasi di progettazione

2.3.1 Suddivisione oraria

Di seguito viene mostrato come il lavoro è stato suddiviso per attività e ore attraverso un Diagramma di Gantt:

#	Attività	Settimane									
		1	2	3	4	5	6	7	8	9	10
1	Studio progetto complessivo e concetti base										
2	Analisi dei requisiti										
3	Studio tecnologie										
4	Progettazione										
5	Formazione contestuale alla progettazione										
6	Implementazione del *software*										
7	Test e sperimentazione del *software*										
8	Documentazione										

Figura 2.1: Diagramma di Gantt per la suddivisione delle attività

Sono state previste 3 settimane part-time per un totale di 60 ore, le rimanenti 6 settimane, invece, a tempo pieno per un totale (tra part-time e full-time) di circa 300 ore, suddivise in diversi segmenti:

- Studio progetto complessivo e concetti base
- Analisi dei requisiti;
- Studio tecnologie;
- Progettazione;
- Formazione contestuale alla progettazione;
- Implementazione software;
- Test e sperimentazione del software;
- Documentazione.

La suddivisione temporale con l'avanzamento della codifica ha subito delle leggere modifiche dovute da imprevisti con l'installazione di alcune tecnologie richieste per l'implementazione di diverse funzionalità. Ciononostante non ha causato ritardi.

2.3.2 Traccia cronologica delle attività svolte

Per quanto riguarda il mantenimento delle informazioni di quanto svolto è stata creata una nota Evernote condivisa, dedicata alla sola cronologia delle attività svolte. Di seguito una foto esplicativa:

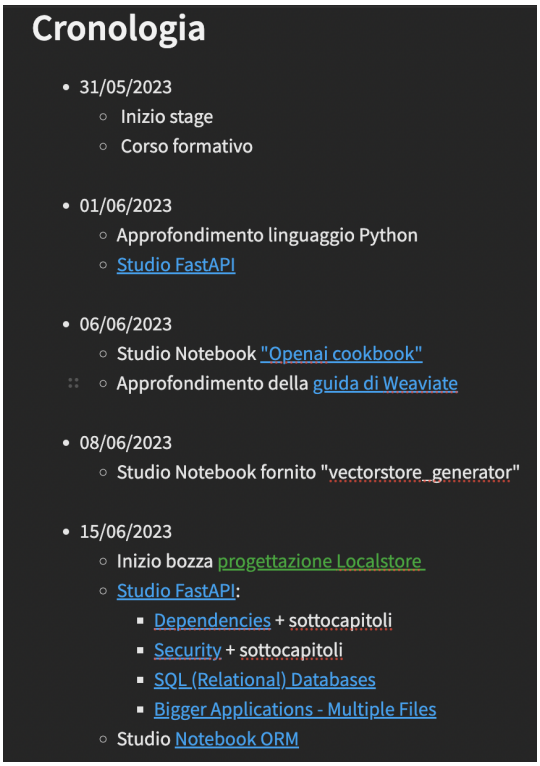


Figura 2.2: Cronologia delle attività svolte

2.4 Obiettivi prefissati

Di seguito vengono elencati tutti gli obiettivi ritenuti obbligatori e non, al fine della corretta implementazione del prodotto:
Obbligatori

- **O01**: inquadramento del problema con stesura del documento di specifica;
- **O02**: analisi di casi d'uso e requisiti;
- **O03**: progettazione ed implementazione di store per la conservazione di documenti (e relativi *embedding*) che consente di recuperare documenti rilevanti per una certa *query* e la gestione di più utenti con accesso a documenti differenti;
- **O04**: progettazione ed implementazione di un microservizio per l'esposizione delle funzionalità sopra citate tramite servizi REST

- **O05**: redazione della documentazione specifica per la descrizione degli strumenti e delle tecniche utilizzate, descrizione dell'architettura del sistema, documentazione del progetto per lo sviluppatore e dei passi necessari per installare ed eseguire il tutto.

Facoltativi

- **F01**: *frontend* che prevede le funzionalità di *upload* dei documenti, gestione utenti e la possibilità di porre domande sulle *knowledge base* create;
- **F02**: microservizio REST da integrare all'esistente sistema per la gestione e l'aggiunta di ulteriori sistemi *target* interrogabili (uno dei quali sarà lo *store* implementato negli obiettivi obbligatori);
- **F03**: integrazione con Microsoft Teams.

2.5 Pianificazione del lavoro

Ogni attività, inizialmente, è stata conseguita con l'aiuto del tutor aziendale, il quale dava le basi per avanzare al meglio con il lavoro da svolgere. Per ogni singola scelta progettuale sono state create delle note su Evernote, condivise tra vari utenti all'interno dell'azienda, ciascuna delle quali conteneva ogni informazione utile a comprendere le motivazioni di ogni decisione presa.

2.5.1 Studio progetto complessivo e concetti base

Ad inizio stage è stata fornita una nota contenente una spiegazione generale del progetto complessivo che l'azienda stava già sviluppando, ovvero la *web application* "Socrate". Il compito principale in questa fase è stato quello di leggere attentamente il documento per capirne i singoli dettagli da implementare ed afferrare tutti i concetti base, quali: *knowledge base* , strato di persistenza, API, Localstore API, Weaviate ed Elasticsearch.

2.5.2 Analisi dei requisiti

È stato creato un documento nel quale sono stati inseriti tutti i requisiti fondamentali e non che il progetto doveva soddisfare per una più completa implementazione di esso.

2.5.3 Studio tecnologie

Per questa fase sono state prese le guide web delle principali tecnologie utilizzate per la codifica del prodotto. In quanto totalmente nuove allo stagista, il loro studio ha inciso notevolmente sull'organizzazione delle attività che inizialmente si era prefissato. Nel capitolo successivo verranno introdotte tutte le tecnologie che sono state usate durante il periodo di stage.

2.5.4 Formazione contestuale alla progettazione

Durante la fase progettuale è stato necessario un ulteriore periodo di formazione sui linguaggi di programmazione da utilizzare nel corso della codifica del prodotto.

2.5.5 Implementazione software

Una volta completata la fase di progettazione ed approvata dal tutor aziendale si è entrati in quella di implementazione del *software*, dove si è andati a codificare ciò che si è stipulato durante la progettazione.

2.5.6 Test e sperimentazione software

Questa fase è avvenuta diverse volte nel corso della fase di codifica del prodotto (sia *backend* che *frontend*). In questo periodo ci sono stati *meeting* con il tutor, dove si andava a fare un controllo del codice e si cercavano possibili errori o bachi all'interno del codice.

2.5.7 Documentazione

Questa fase è concorrente a tutte quelle sopracitate, in quanto attività in linea a tutte le scelte progettuali e di codifica.

Capitolo 3

Strumenti e teconologie

In questo capitolo vengono elencate le tecnologie e strumenti che sono stati utilizzati durante l'intero periodo di stage. In più viene spiegata nel dettaglio la fase di progettazione e codifica del prodotto.

3.1 Strumenti utilizzati

3.1.1 Microsoft Teams

Microsoft Teams è una piattaforma di collaborazione e comunicazione aziendale sviluppata da Microsoft. È parte dell'ecosistema Microsoft 365 (in passato noto come Office 365) ed è progettato per migliorare la comunicazione e la produttività nelle aziende, consentendo ai team di lavorare insieme in modo efficiente, sia in remoto che in presenza. Nel periodo di stage, infatti, è stato fondamentale per la comunicazione con il tutor aziendale ed altri dipendenti all'interno dell'ambiente di lavoro, con i quali si sono tenuti meeting settimanali di aggiornamento e risoluzione dei problemi.

Figura 3.1: Microsoft Teams

3.1.2 Evernote

Evernote è un'applicazione di organizzazione e gestione delle informazioni molto popolare e versatile, disponibile per diverse piattaforme, tra cui *desktop*, web e dispositivi mobili. La sua caratteristica principale è quella di permettere agli utenti

di catturare, organizzare e sincronizzare note, appunti, immagini, pagine web e altri contenuti in modo efficace e accessibile ovunque. Nello specifico è stato creato un taccuino condiviso con il tutor aziendale ed altri responsabili, nel quale sono state salvate tutte le note relative alle scelte progettuali e ai fini organizzativi di ogni settimana lavorativa.

Figura 3.2: Evernote

3.1.3 Microsoft Outlook

Outlook è un *software* di gestione delle informazioni e di comunicazione sviluppato da Microsoft. Esso è principalmente noto come client di posta elettronica, ma offre anche una vasta gamma di funzionalità per la gestione delle attività, il calendario, i contatti e molto altro. Outlook è compatibile con diverse piattaforme, tra cui Windows, macOS, dispositivi mobili e la versione web. Sono state fornite le credenziali per accedere alla piattaforma, utilizzata per le comunicazioni più importanti e organizzazione di meeting settimanali.

Figura 3.3: Microsoft Outlook

3.1.4 Visual Studio Code

Visual Studio Code (VS Code) è un editor di codice sorgente gratuito e *open source* sviluppato da Microsoft. È progettato per essere leggero, veloce e altamente

personalizzabile, e offre un'esperienza di sviluppo efficiente per una vasta gamma di linguaggi di programmazione e tecnologie. VS Code è disponibile per Windows, macOS e Linux. Per questo progetto è stato utilizzato per l'intervo sviluppo dell'applicativo, con l'appoggio di diverse estensioni:

- **Ruff**: è un *linter*, utile per l'analisi del codice, il quale segnala errori, problemi di stile e potenziali bug;
- **SQLite Viewer**: estensione che permette di visualizzare le tabelle SQLite;
- **Live Share**: estensione che permette la condivisione in tempo reale di progetti in VS Code, facendo in modo che più persone possano codificare contemporaneamente;
- **autoDocstring - Python Docstring generator**: per la creazione istantanea di *docstrings*[1] python;
- **Jupyter**: per la creazione, visualizzazione e avvio di Jupyter Notebook, utili per il pre-testing del codice da sviluppare.

Figura 3.4: Visual Studio Code

3.1.5 Draw.io

Draw.io è un'applicazione web gratuita e *open source* per la creazione di diagrammi e schemi, sviluppata da JGraph Ltd. Questo strumento è ampiamente utilizzato per disegnare diagrammi di flusso, organigrammi, diagrammi UML, mappe concettuali, diagrammi di rete, wireframe e molti altri tipi di grafici e rappresentazioni visive. Per questo progetto è stato utilizzato per la creazione del modello Entità-Relazionare per lo sviluppo del Database e gli UML per la struttura del prodotto.

[1] docstring: stringa di documentazione incorporata nel codice sorgente di un programma o di una funzione, utile a fornire una spiegazione, una descrizione o una documentazione del codice circostante.

Figura 3.5: Draw.io

3.1.6 Check Point VPN

Check Point VPN(Virtual Private Network) è una soluzione di rete sicura fornita da Check Point Software Technologies Ltd., un'azienda specializzata nella sicurezza informatica. La soluzione VPN di Check Point consente di creare una connessione sicura tra dispositivi remoti o utenti e una rete aziendale o una rete privata, utilizzando un tunnel crittografato attraverso internet. È stato offerto accesso alla VPN tramite l'utilizzo delle credenziali fornite ad inizio stage, tramite l'attivazione di questa è stato possibile l'utilizzo della piattaforma GitLab aziendale.

Figura 3.6: Check Point VPN

3.1.7 GitLab

GitLab è una piattaforma di sviluppo *software* che offre un set completo di strumenti per la gestione del ciclo di vita del *software* e la collaborazione tra *team* di sviluppo. È basato su Git, un sistema di controllo versione distribuito, ed è progettato per supportare lo sviluppo di progetti software in modo efficiente e collaborativo. GitLab è disponibile in due versioni principali: GitLab Community Edition (CE) e GitLab Enterprise Edition (EE), quest'ultima con funzionalità aggiuntive orientate all'uso aziendale.

Figura 3.7: GitLab

3.1.8 Docker Desktop

Docker Desktop è un'applicazione per *desktop* che semplifica la gestione e l'utilizzo di *container*[2] Docker su sistema operativo locale. È progettato per fornire un'esperienza *user-friendly* per sviluppatori, devops e altri professionisti IT che vogliono creare, testare e distribuire applicazioni in ambienti di containerizzati. Per questo progetto (verranno spiegati nel dettaglio nelle tecnologie utilizzate) sono stati configurati due server: Tika per l'estrazione e *chunking* del testo dei documenti caricati e Weaviate per lo *storing* dei *chunk*.

Figura 3.8: Docker Desktop

Tika Server

Tika Server è un'applicazione basata su Java che fornisce servizi di estrazione di testo e metadati da una vasta gamma di formati di *file*. È un componente del progetto Apache Tika ed è progettato per essere eseguito come un *server* autonomo.

Figura 3.9: Tika Server

Weaviate

Weaviate è un motore di ricerca semantica *open-source* che utilizza tecnologie di *embedding* vettoriale per consentire la ricerca e il recupero avanzato di dati in base alla loro similarità semantica.

Figura 3.10: Weaviate

[2]container: tecnologia di virtualizzazione leggera che consente di eseguire applicazioni e servizi in un ambiente isolato e autosufficiente.

3.2 Tecnologie utilizzate

3.2.1 Python

Python è un linguaggio di programmazione interpretato, ad alto livello, versatile e orientato agli oggetti. È stato creato da Guido van Rossum e il suo sviluppo ha avuto inizio alla fine degli anni '80. Python è noto per la sua semplicità e leggibilità del codice, che lo rende una scelta popolare per principianti e professionisti nel campo dello sviluppo software.

Figura 3.11: Python

Pro

- **Semplicità e leggibilità del codice**: la sintassi di Python è molto chiara e semplificata, rendendo il codice più facile da scrivere, leggere e mantenere;
- **Vasta comunità e risorse**: Python ha una delle comunità di sviluppatori più grandi al mondo, di conseguenza, ci sono numerose risorse, documentazioni, librerie e *framework* disponibili che semplificano e velocizzano lo sviluppo di progetti;
- **Multi-piattaforma**: Python è supportato su diverse piattaforme, tra cui Windows, macOS e Linux, consentendo di sviluppare applicazioni *cross-platform*;
- **Ampia libreria standard**: Python offre una vasta libreria *standard* che fornisce funzioni per una varietà di attività comuni, come gestione *file*, *networking*, elaborazione di testo e altro ancora;
- **Estensibilità**: Python è altamente estensibile grazie alla possibilità di integrare codice scritto in C/C++ e altre lingue, consentendo di migliorare le prestazioni di parti critiche di un'applicazione;
- **Adatto per l'apprendimento**: la semplicità e la leggibilità del codice lo rendono uno dei linguaggi più adatti per principianti che desiderano imparare la programmazione.

Contro

- **Prestazioni**: Python nonostante sia potente e versatile, è noto per essere meno performante rispetto a linguaggi come C o C++, poiché è interpretato. Potrebbe essere dunque meno veloce in alcune situazioni, specialmente per applicazioni ad alte prestazioni o con computazioni intensive;
- GIL: caratteristica che limita l'esecuzione del codice Python su un singolo thread alla volta. Ciò potrebbe influire sulle prestazioni in applicazioni multithreading;
- **Dimensioni dell'eseguibile**: le applicazioni Python tendono ad avere dimensioni dell'eseguibile maggiori rispetto a quelle scritte in linguaggi compilati, poiché richiedono l'interpretazione del codice a *runtime*[3];
- **Concorrenza**: sebbene Python supporti la programmazione concorrente, la gestione di *thread* e processi potrebbe essere più complessa rispetto ad alcuni altri linguaggi.

3.2.2 SQLAlchemy

SQLAlchemy è una libreria Python che offre un *framework* Object-Relational Mapping per interagire con database relazionali utilizzando il linguaggio di programmazione Python. SQLAlchemy semplifica la gestione delle operazioni di database, consentendo agli sviluppatori di scrivere query SQL in modo più intuitivo utilizzando oggetti Python, senza dover scrivere query SQL direttamente.

Figura 3.12: SQLAlchemy

Pro

- **Portabilità**: SQLAlchemy è compatibile con diversi *database* relazionali, inclusi MySQL, PostgreSQL, SQLite, Oracle e altri. Ciò consente di scrivere codice che funzioni su diversi *database* senza dover apportare modifiche significative;
- **Sicurezza**: utilizzare l'ORM di SQLAlchemy contribuisce a prevenire le vulnerabilità comuni come le SQL injection, in quanto i parametri vengono gestiti in modo sicuro all'interno del *framework*;
- **Query flessibili**: SQLAlchemy offre una vasta gamma di metodi per eseguire *query* complesse, con espressioni, filtri, aggregazioni e altro ancora;

[3]runtime: durante il quale un programma o un'applicazione software è in esecuzione su una macchina o un dispositivo.

- **Transazioni e controllo delle transazioni**: SQLAlchemy gestisce automaticamente le transazioni e offre un controllo flessibile sulle transazioni per garantire l'integrità dei dati nel *database*;
- **Estensibilità**: SQLAlchemy offre una serie di estensioni e strumenti per supportare funzionalità avanzate, come la gestione di *database* spaziali o la creazione di indici personalizzati.

Contro

- **Complessità iniziale**: SQLAlchemy potrebbe richiedere una curva di apprendimento iniziale per gli sviluppatori che non hanno esperienza con ORM. La complessità può aumentare con operazioni avanzate o complesse sul *database*;
- **Overhead**: l'utilizzo di un ORM come SQLAlchemy può introdurre un certo *overhead* [4] rispetto all'uso diretto delle *query* SQL, poiché l'ORM deve interpretare e mappare gli oggetti Python al *database*.
- **Performance**: in alcune situazioni, l'uso di SQLAlchemy potrebbe non essere ottimale per operazioni ad alta intensità di lettura/scrittura o con enormi quantità di dati. In questi casi, scrivere *query* SQL personalizzate potrebbe risultare più efficiente;
- **Aggiornamenti**: gli aggiornamenti e le modifiche delle versioni potrebbero richiedere modifiche al codice esistente, poiché SQLAlchemy continua a evolversi e migliorare nel tempo.

3.2.3 FastAPI

FastAPI è un *framework web* moderno, veloce e ad alte prestazioni per lo sviluppo di API RESTful in Python. È stato creato per offrire un'esperienza di sviluppo rapida ed efficiente, consentendo agli sviluppatori di costruire API potenti e scalabili con una sintassi semplice e intuitiva. Esso si basa su Python 3.6+ e utilizza il tipo di annotazioni e la validazione dei tipi per semplificare la gestione dei dati e garantire la sicurezza.

Figura 3.13: FastAPI

[4] overhead: costo aggiuntivo o a una quantità di risorse (tempo, memoria, potenza di calcolo, larghezza di banda, etc.) che vengono impiegate per eseguire attività o processi accessori che non sono direttamente correlati all'obiettivo principale.

Pro

- **Elevata velocità**: FastAPI è noto per le sue prestazioni eccezionali e la velocità di esecuzione, grazie all'utilizzo di Python 3.6+ e dell'approccio di tipo annotazioni, FastAPI può eseguire il codice molto più rapidamente rispetto ad altri *framework* Python per la costruzione di API.
- **Sintassi semplice ed intuitiva**: dato che FastAPI utilizza la sintassi di tipo annotazioni Python, lo rende molto leggibile e facile da comprendere.
- **Validazione dei tipi**: FastAPI offre una potente validazione dei tipi, consentendo agli sviluppatori di specificare i tipi di dati attesi in *input* e *output* delle API. Questa caratteristica aiuta a prevenire errori e garantisce la sicurezza dei dati.
- **Supporto per i WebSocket**: FastAPI supporta anche i WebSocket, consentendo di creare API *real-time* interattive e comunicazioni bidirezionali tra *client* e *server*.
- **Documentazione automatica**: FastAPI genera automaticamente documentazione interattiva per le API create, rendendo facile per gli sviluppatori e gli utenti esplorare, testare e comprendere l'utilizzo delle API.
- **Integrazione con librerie Python esistenti**: FastAPI si integra bene con altre librerie Python, consentendo di utilizzare librerie di terze parti per aggiungere funzionalità aggiuntive alle API.

Contro

- **Curva di apprendimento iniziale**: sebbene la sintassi di tipo annotazioni di Python renda FastAPI leggibile, potrebbe richiedere un po' di tempo per gli sviluppatori che non hanno esperienza con questo approccio.
- **Supporto della comunità**: nonostante FastAPI sia diventato molto popolare, potrebbe avere meno risorse e documentazione rispetto a *framework* più consolidati e più anziani.
- **Ancora in fase di sviluppo**: FastAPI è un *framework* relativamente giovane, quindi potrebbe essere soggetto a cambiamenti e aggiornamenti frequenti.

3.2.4 Httpx

HTTPX è una libreria Python per richieste HTTP che si concentra sulla flessibilità, le prestazioni e l'esperienza dell'utente. È progettata per sostituire la libreria *standard requests* e offre funzionalità avanzate come il supporto nativo per richieste asincrone e la compatibilità con le specifiche HTTP/1.1, HTTP/2 e WebSocket.

Figura 3.14: Httpx

Pro

- **Richieste Asincrone**: HTTPX supporta richieste asincrone grazie all'utilizzo di *coroutines* (async/await). Questo significa che è possibile eseguire richieste HTTP in modo parallelo, il che può migliorare significativamente le prestazioni in scenari ad alta concorrenza;
- **Gestione Automatico dei Pool di Connessione**: HTTPX gestisce automaticamente i *pool* di connessione per migliorare l'efficienza e le prestazioni delle richieste HTTP, riducendo al minimo il sovraccarico delle connessioni;
- **Supporto per Autenticazione**: HTTPX supporta diverse forme di autenticazione, come l'autenticazione base e l'autenticazione con *token*.

Contro

- **Curva di apprendimento**: anche se l'interfaccia è simile a *requests*, potrebbe comunque essere necessario un periodo di adattamento per abituarsi alle specifiche funzionalità e convenzioni di HTTPX;
- **Dimensione del Pacchetto**: a causa delle sue funzionalità avanzate, HTTPX potrebbe avere una dimensione del pacchetto leggermente maggiore rispetto ad altre librerie di richieste HTTP più leggere.

3.2.5 Streamlit

Streamlit è un *framework open-source* per la creazione di applicazioni web interattive e di *data science* in modo semplice e veloce utilizzando Python. Con Streamlit, gli sviluppatori possono trasformare facilmente *script* Python in app web completamente funzionali, senza dover imparare nuovi linguaggi o tecnologie. Streamlit è progettato per semplificare il processo di sviluppo e consentire agli utenti di creare rapidamente e facilmente applicazioni web per visualizzare dati, eseguire analisi, costruire dashboard e molto altro ancora.

Figura 3.15: Streamlit

Pro

- **Facilità d'Uso**: Streamlit è noto per la sua semplicità e facilità d'uso. Gli sviluppatori possono creare applicazioni web interattive con poche righe di codice, riducendo la curva di apprendimento per chi non ha esperienza nello sviluppo web;
- **Pythonic**: Streamlit utilizza Python come linguaggio principale per la creazione di applicazioni. Questo è un vantaggio per gli sviluppatori Python, in quanto possono utilizzare le loro competenze esistenti senza dover imparare nuovi linguaggi o *framework*;
- **Prototipazione Rapida**: Streamlit è ideale per la prototipazione rapida di applicazioni. Gli sviluppatori possono creare rapidamente dashboard e interfacce utente per visualizzare e interagire con i dati.
- **Interattività**: Streamlit offre un'ampia gamma di componenti interattivi predefiniti, come grafici, widget e pannelli di controllo, che consentono agli utenti di interagire con i dati e personalizzare la visualizzazione.
- **Ampio Supporto per Librerie di Visualizzazione**: Streamlit integra facilmente con librerie di visualizzazione popolari come Matplotlib, Plotly e Altair, consentendo agli sviluppatori di creare visualizzazioni accattivanti.

Contro

- **Limitato a Python**: Anche se è un vantaggio per gli sviluppatori Python, chiunque sia abituato ad altri linguaggi potrebbe sentirsi limitato dalla necessità di utilizzare Python per creare applicazioni con Streamlit.
- **Flessibilità Limitata**: Streamlit è progettato per semplificare la creazione di applicazioni web, ma questo può limitare la flessibilità per progetti più complessi o personalizzati che richiedono funzionalità più avanzate.
- **Limitazioni di Personalizzazione**: Sebbene Streamlit offra molti componenti predefiniti, la personalizzazione avanzata dell'aspetto e del comportamento delle applicazioni potrebbe richiedere più sforzi rispetto a un *framework* di sviluppo web completo.
- **Gestione delle Sessioni**: Streamlit è inizialmente progettato per essere eseguito localmente durante lo sviluppo. Quando si tratta di distribuire applicazioni in produzione o gestire le sessioni degli utenti, potrebbero sorgere sfide aggiuntive.

Capitolo 4

Analisi dei requisiti

Questo capitolo propone di elencare e specificare, dopo un'attenta analisi conseguita con il tutor aziendale, tutti i requisiti che sono stati individuati per l'implementazione corretta e completa del prodotto.

4.1 Tracciamento dei requisiti

È stato creato un documento nel quale sono stati inseriti tutti i requisiti fondamentali e non che il progetto doveva soddisfare per una più completa implementazione di esso. Di seguito vengono elencati tutti i requisiti che sono stati trovati dopo un'attenta attività di studio. Il codice dei requisiti è stato strutturato nella seguente maniera: R(N/D)(U/S/D/C) dove:

R = requisito

N = obbligatorio (necessario)

D = desiderabile

Z = opzionale

U = utente

S = store

D = documento

C = chunk

4.1.1 Requisiti relativi agli utenti

Questa tabella contiene tutti i requisiti che consentono la gestione corretta degli utenti registrati all'interno dell'applicativo.

Requisito	Descrizione
RNU-1	L'utente deve poter essere creato ed inserito all'interno del database;
RNU-2	L'utente deve poter eseguire l'accesso tramite le credenziali fornite;
RNU-3	L'utente può possedere i privilegi di "superuser";
RNU-4	L'utente che ha eseguito correttamente il login può modificare le proprie credenziali;
RNU-5	Gli utenti superuser possono visualizzare tutti gli utenti registrati all'interno della piattaforma;
RNU-6	Gli utenti superuser possono visualizzare le informazioni di qualsiasi utente registrato all'interno della piattaforma;
RNU-7	Due utenti diversi non possono avere lo stesso username;
RNU-8	Ogni utente che ha eseguito correttamente l'accesso deve poter creare uno store;
RNU-9	La creazione di un nuovo utente è compito esclusivo di qualsiasi utente dotato di privilegi da superuser;
RDU-10	Se l'utente che esegue il login inserisce le credenziali in modo errato allora deve esere restituito un messaggio d'errore;
RDU-11	Se un utente qualsiasi prova ad accedere alle informazioni di un utente registrato all'interno del sistema, deve venire restituito un messaggio d'errore;

Tabella 4.1: Tabella del tracciamento dei requisti relativi agli Utenti

4.1.2 Requisiti relativi agli store

Questa tabella elenca tutti i requisiti che devono essere rispettati (o desiderati) per la corretta gestione degli store all'interno dell'applicativo.

Requisito	Descrizione
RNS-1	Lo store deve essere associato univocamente ad un solo utente;
RNS-2	Ci possono essere più store con il medesimo nome;
RNS-3	Uno store può essere eliminato solamente dall'utente che lo ha creato, oppure da un utente dotato dei privilegi da superuser;
RNS-4	Si deve dare la possibilità all'utente richiedente di visualizzare tutti gli store che esso ha creato;
RNS-5	Si deve dare la possibilità all'utente richiedente di visualizzare lo store che lui stesso ha creato;
RNS-6	L'utente superuser deve avere la possibilità di visualizzare la lista completa di tutti gli store nel database;
RNS-7	L'utente dotato dei privilegi da superuser deve avere la possibilità di visualizzare qualsiasi store;
RNS-8	Si deve dare la possibilità di condividere uno store creato dall'utente con altri utenti;
RNS-9	Gli utenti che hanno ricevuto in condivisione gli store possono accedervi (sola visualizzazione);
RNS-10	si deve dare la possibilità all'utente che ha condiviso uno store di revocarlo verso l'utente che desidera;
RNS-11	Si deve dare la possibilità all'utente che ha condiviso uno store di revocarlo verso tutti gli utenti con cui lo ha condiviso;
RNS-12	L'eliminazione di uno store ne comporta anche la rimozione di tutti i documenti ad esso contenuti e la cartella associata;
RNS-13	L'utente corrente deve avere la possibilità di visualizzare tutti gli store che gli sono stati condivisi;
RDS-14	Se un utente prova ad eliminare un store che non ha creato, viene lanciato un messaggio d'errore;
RDS-15	Se un utente prova ad accedere ad uno store che non ha creato o non gli è stato condiviso, viene lanciato un messaggio d'errore;

Tabella 4.2: Tabella del tracciamento dei requisti relativi agli Store

4.1.3 Requisiti relativi ai Documenti

Questa tabella elenca tutti i requisiti individuati al fine di gestire correttamente l'inserimento dei documenti all'interno di store, con i corretti output per le richieste effettuate dall'utente.

Requisito	Descrizione
RND-1	Ogni documento viene identificato univocamente dal proprio id, lo store di appartenenza e l'utente che ha effettuato l'upload;
RND-2	Deve venire salvato nel database il nome effettivo del file caricato;
RND-3	Possono venire caricati file identici in diversi store;
RND-4	Si deve dare la possibilità di eliminare un file all'interno di uno store (creato dall'utente);
RND-5	Si deve dare la possibilità di eliminare tutti i file all'interno di uno store (creato dall'utente) con una sola richiesta;
RND-6	Gli utenti dotati superuser possono eliminare qualsiasi file contenuto in qualsiasi store;
RND-7	Gli utenti che ricevono degli store via condivisione possono accedere ai file al loro interno (sola visualizzazione e interrogazione) ;
RDD-8	Se un utente cerca di eliminare o visualizzare un file che non gli appartiene viene lanciato un messaggio di errore;
RDD-9	Se un utente tenta di eliminare un file appartenente ad uno store che gli è stato condiviso viene lanciato un messaggio d'errore;
RDD-10	Se un utente cerca di acccedere ad un file di uno store che non gli appartiene o non gli è stato condiviso, viene lanciato un messaggio d'errore;
RDD-11	Se un utente prova ad aggiungere un file in uno store che non gli appartiene o non gli è stato condiviso, viene lanciato un messaggio d'errore;

Tabella 4.3: Tabella del tracciamento dei requisti relativi ai Documenti

4.1.4 Requisiti relativi ai chunk

Questa tabella elenca tutti i requisiti che sono stati individuati al fine di gestire al meglio la suddivisione dei documenti in chunk, i quali poi verranno ritornati qualora l'utente dovesse effettuare una query su uno store di cui il documento relativo al chunk appartiene.

Requisito	**Descrizione**
RNC-1	Ogni chunk viene identificato univocamente dal documento da cui è stato estratto;
RNC-2	Per ogni chunk deve essere specificata la pagina di inizio e quella di fine;
RNC-3	Se un utente elimina un documento che ha caricato, conseguentemente vengono eliminati tutti i chunk associati;
RNC-4	Quando un documento viene caricato, esso viene suddiviso per l'appunto in sezioni (chunk) e caricate nel vectorstore Weaviate;
RNC-5	Se un utente elimina uno store, di conseguenza vengono eliminati tutti i documenti contenuti in esso e con questi anche i chunk associati;
RNC-6	L'interrogazione ad uno store ritorna tutti i chunk più inerenti con la query che l'utente ha effettuato;
RNC-7	L'estrazione del testo avviene tramite Tika server;

Tabella 4.4: Tabella del tracciamento dei requisti relativi ai Chunk

Capitolo 5

Progettazione e codifica

In questo capitolo vengono spiegate le fasi di progettazione e codifica dell'applicativo. Le spiegazioni per entrambe le fasi sono state suddivise per *backend* e *frontend*.

5.1 Progettazione

Per la fase di progettazione sono stati usati diversi strumenti, tra i quali:

- **Evernote**: per tenere traccia delle scelte progettuali e contenere tutte le informazioni necessarie alla comprensione del prodotto;
- **Draw.io**: per la creazione dell'UML esplicativo delle classi da implementare ed il modello Entità-Relazione per la progettazione del database.

5.1.1 Sintesi dell'architettura

Di seguito viene mostrata la sintesi dell'architettura del prodotto per come è stato pensato. Ciò che è stato preso in considerazione per questo stage è l'implementazione di Localstore API, di cui il funzionamento è stato spiegato precedentemente.

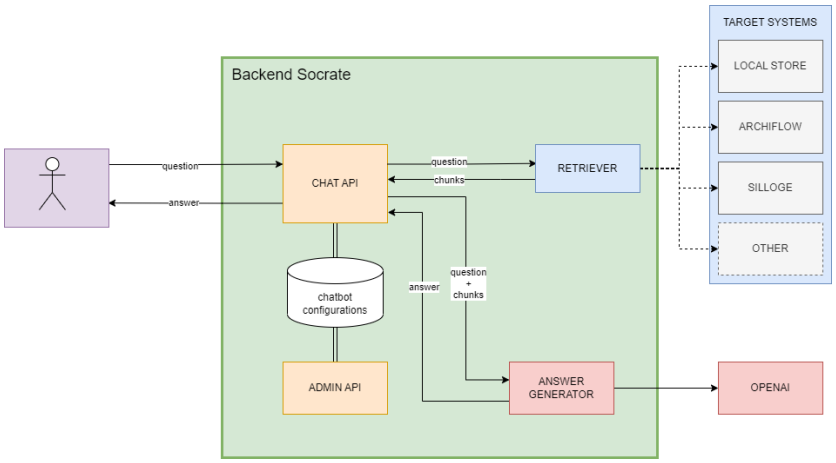


Figura 5.1: Sintesi dell'architettura

Da questa immagine possiamo notare diversi attori, tra i quali:

- **Chat API**: servizio che coordina le logiche implementate in *retriever*[1] ed *answer generator* (usa retriever per recuperare il contesto, interroga answer generator per ottenere la risposta, espone le chiamate rest necessarie, gestisce i log, etc.). Dovrà anche gestire l'accesso ai differenti *chatbot*, controllando che l'utente specifico abbia visibilità sul *chatbot* che vuole interrogare.
- **Retriever**: si tratta dell'oggetto che recupera da un dato sistema *target* i *chunk* rilevanti rispetto alla domanda posta. Tale oggetto deriverà da diverse implementazioni della classe Retriever, ognuna che gestirà un dato sistema *target*
- **AnswerGenerator**: a fronte di una domanda e delle porzioni (i.e., chunk) rilevanti di documenti reperite dal *retriever*, produce la risposta alla domanda. Implementazioni diverse di AnswerGenerator si interfacciano a servizi che mettono a disposizione LLM diversi (e.g., OpenAI o self-hosted). In funzione della tipologia di LLM applica le strategie più adeguate (prompt , lunghezza contesto totale formato da più chunk, ecc.)
- **Admin API**: servizio che si occupa di gestire e popolare la configurazione per i vari *chatbot* gestiti da Socrate. Nelle configurazioni vengono specificate le informazioni necessarie a connettersi ai vari sistemi (filtri da applicare, indici in cui cercare, etc.) per reperire i documenti corretti.
- **Sistemi target**: si tratta dei sistemi dove è memorizzata la *knowledge base* che alimenta i *chatbot* (es. documenti Archiflow o Silloge, chunk Weaviate ecc.)
  - LocalStore: è uno *store* che conserva direttamente le porzioni dei documenti e ne definisce le modalità di accesso. E' necessario il suo impiego ogni qualvolta si vuole creare un *chatbot* la cui *knowledge base* è formata da documenti provenienti da fonti diverse (e.g., Confluence, documentazione presente in altri sistemi, ecc.). Nella sua prima implementazione utilizzerà il *vector database* Weaviate per la persistenza e il recupero dei vettori relativi ai chunk di testo caricati;
  - Archiflow/Silloge: implementano già le funzionalità per gestire visibilità, reperire e caricare documenti.

5.1.2 Inserimento di un documento

Quando un utente desidera caricare un documento su uno *store* di cui ha il controllo, subentra Tika Server, il quale preleverà il testo del documento caricato e tramite le funzioni di codice sviluppate verrà suddiviso in *chunk*. Questi verranno poi inviati

[1] retriever: componente di un sistema di recupero delle informazioni o un motore di ricerca, che ha il compito di trovare e recuperare documenti o dati pertinenti da una vasta collezione di informazioni. Il suo obiettivo è restituire risultati di ricerca rilevanti in risposta a una query o una richiesta dell'utente.

ad OpenAI, che ne calcolerà i relativi *embedding* di ciascuno. Successivamente, ogni *chunk* con il relativo *embedding* verrà inserito nel *vector database* Weaviate per il mantenimento delle informazioni e la ricerca dei *chunk* più rilevanti in base alle *query* poste dall'utente.

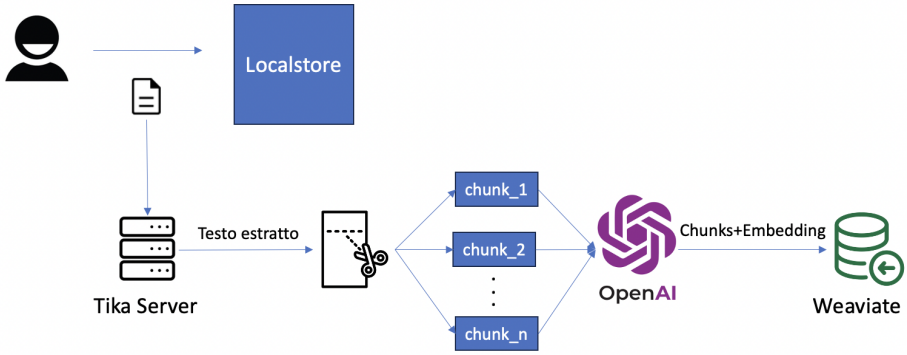


Figura 5.2: Inserimento di un documento

5.1.3 Interrogazione della knowledge base

L'utente che desidera interrogare uno *store*, invia una *query* alla knowledge base che ha selezionato. Ne verrà dunque calcolato l'*embedding* di essa tramite OpenAI, la quale verrà inviata a Weaviate per la ricerca dei *chunk* più rilevanti in base alla richiesta. Infine, una volta individuati, verranno restituiti all'utente richiedente.

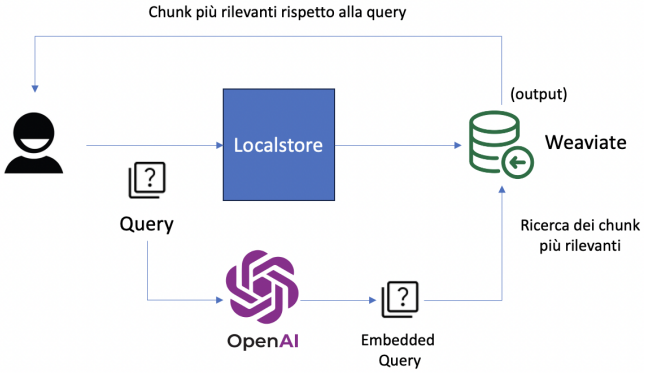


Figura 5.3: Interrogazione della knowledge base

5.1.4 Progettazione Backend

Per la progettazione del *backend* è stato fatto uno schema simil-UML con le informazioni fornite per ciò che Localstore API doveva svolgere, nel dettaglio:

```
LOCALSTORE API
    LOGIN(USERNAME, PASSWORD) -> TOKEN
    CREATE_USER(USER) -> USER
    DELETE_USER(USERNAME)
```

```
UPDATE_USER(USER) -> USER
LIST_USERS()
CREATE_STORE(STORE_ID, USER_ID, EMBEDDING_MODEL) -> STORE
DELETE_STORE(STORE_ID, USER_ID)
SHARE_STORE(STORE_ID, USER_ID, RECEIVER_IDS)
REVOKE_STORE(STORE_ID, USER_ID, RECEIVER_IDS)
UPSERT(FILE, METADATA, STORE_ID, USER_ID) -> DOCUMENT_ID
DELETE(DOCUMENT_ID, STORE_ID, USER_ID)
QUERY(QUESTION, STORE_ID, USER_ID) -> CHUNKS
LIST_STORES(USER_ID)
CHECK_ACCESS(USERNAME, STORE_ID) -> STATUS
```

Il concetto di STORE è da intendere come una "classe" di Weaviate (analogamente agli "indici" in Elastic Search) dove vengono caricati i documenti relativi ad un singolo *chatbot* (ogni STORE è praticamente la *knowledge base* per un *chatbot*). Lo schema che è stato creato in base a ciò che è stato elencato sopra, è il seguente:

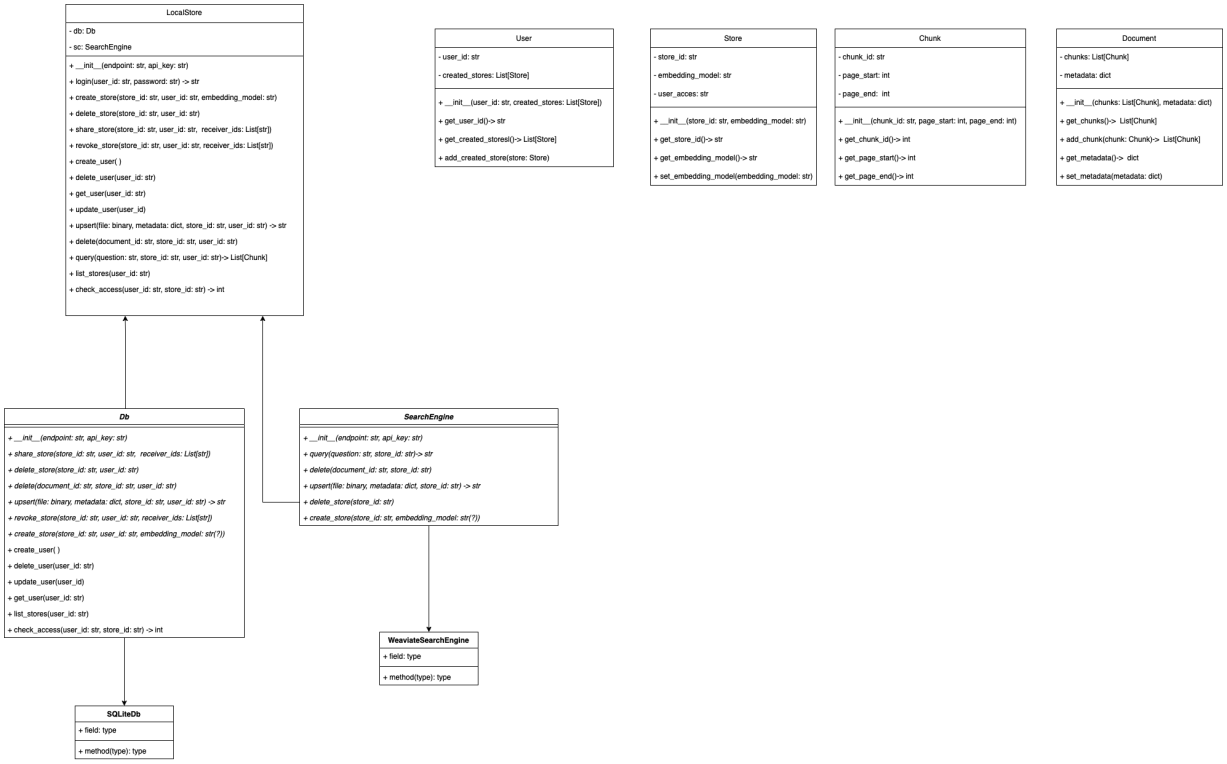


Figura 5.4: Backend

5.1.5 Database

Per la conservazione delle informazioni relative agli utenti, store, documenti e *chunk* di essi si è pensato di progettare e configurare un *database*.

Modello Entità-Relazione non risolto

Per la progettazione del database si è costruito un modello Entità-Relazione, il quale ha la seguente struttura:

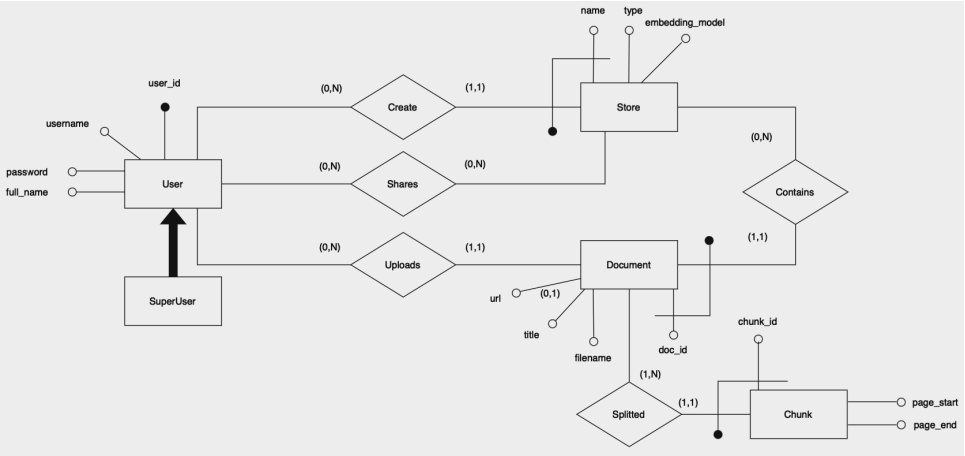


Figura 5.5: Modello Entità-Relazione (non risolto)

Come si può notare vi sono 4 Entità principali ed una generalizzazione, quest'ultima totale poichè un'utente deve essere obbligatoriamente base o *superuser*.

Modello Entità-Relazione risolto

Di seguito viene illustrato il modello Entità-Relazione con risoluzione della generalizzazione, con l'incorporamento dell'entità *"Superuser"* in *User* tramite il campo opzionale *"superuser"*:

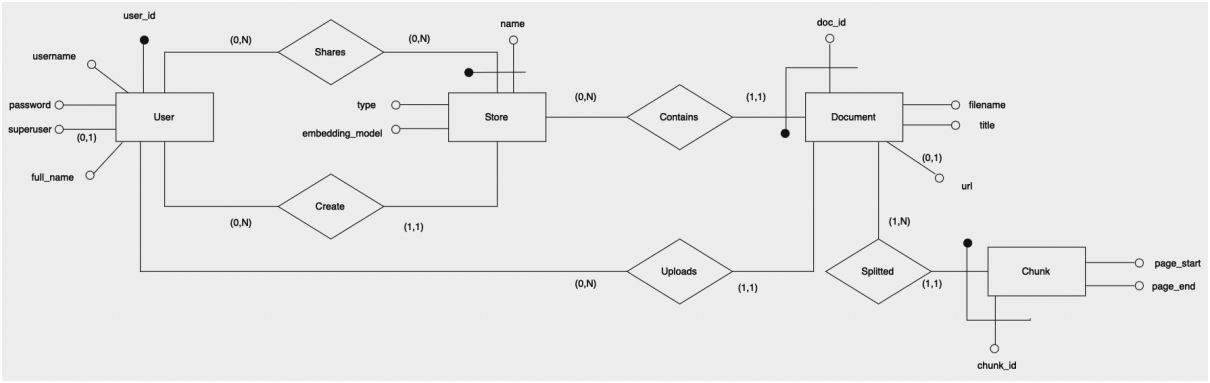


Figura 5.6: Progettazione: Modello Entità-Relazione (risolto)

Identificatori univoci

Per ogni entità viene data una spiegazione sulla scelta dell'identificativo univoco:

- **User**: entità che contiene le credenziali di ogni utente registrato all'interno del sistema e se esso ha i permessi di *superuser*
  - PK: è stata scelta come chiave primaria un identificativo auto incrementante denominato "user_id". Il quale per l'appunto ha l'unico scopo di rendere univoca la singola tupla relativa all'utente.
- **Store**: entità che ha lo scopo di contenere le informazioni relative allo store creato da un'utente
  - PK: la chiave primaria è una chiave composta dall'attributo "name" in *store* e la chiave esterna (FK) verso *User* per l'identificazione dell'utente associato allo *store*.
- **Document**: entità per tenere traccia delle informazioni (non il contenuto) dei documenti che ogni utente carica negli *store* da loro creati
  - PK: per questa entità, la chiave primaria è composta dall'attributo "doc_id" e le chiavi dell'entità *Store*. Così facendo si tiene traccia anche dello *store* a cui appartiene il documento e di conseguenza chi ha caricato quest'ultimo.
- **Chunk**: entità che contiene tutte le informazioni dei *chunk* relativi ai documenti che sono stati caricati
  - PK: ogni *chunk* viene identificato dal proprio id e le chiavi primarie del documento associato. Facendo così riusciamo a risalire a che documento appartiene il sigolo *chunk*, chi ha caricato il documento e lo *store* di appartenenza.

5.1.6 Progettazione Frontend

Anche se il *frontend* era appartenente alla lista dei requisiti non obbligatori, ed essendo in anticipo con i tempi, si è riusciti a progettarne una bozza ed implementarne un Proof of Concept per illustrare il funzionamento a livello utente del prodotto. Inizialmente si è pensato di strutturare il frontend in base a questo Mockup:

Come si vedrà nel capitolo relativo alla codifica del *frontend*, questo *mockup* è servito più come una base da seguire. Infatti il prodotto "finito" risulta leggermente diverso da quello che si è pensato di implementare.

5.2 Codifica

La fase di codifica è stata conseguita in concomitanza con lo studio delle tecnologie da utilizzare, tramite le *user guide* ufficiali fornite, tra le quali troviamo:

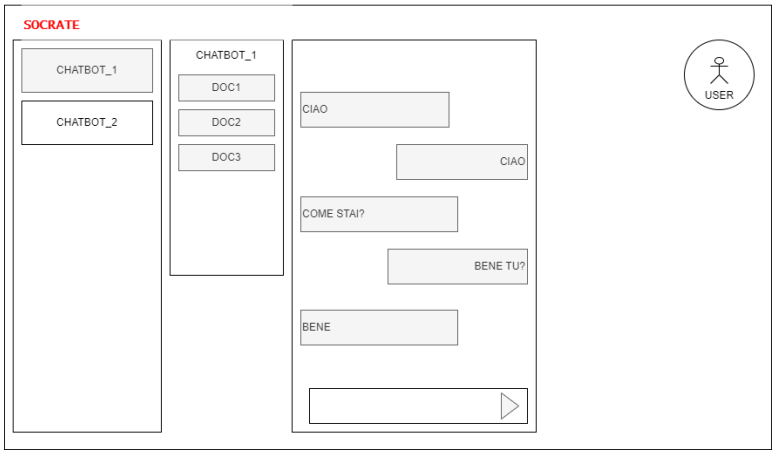


Figura 5.7: Mockup Frontend

- Guida FastAPI per la creazione di API basate su Python;
- OpenAI Cookbook, il quale descrive le principali funzionalità basate su OpenAI API;
- Guida Weaviate, *open-source vector database*;
- Guida creazione ORM in SQLAlchemy;
- Guida Streamlit per la codifica del *frontend* basato su Python.

5.2.1 Codifica Backend

Per quanto riguarda il *backend*, si è adottato un lavoro basato su *feature branch* in GitLab, con un susseguirsi di *commit* "ridotti e mirati" allo sviluppo di determinate funzionalità, così da facilitare il versionamento del prodotto.

Lista dei branch

Di seguito elenchiamo la lista di *branch* strutturata all'interno della *repository* relativa al *backend*:

- **Master**: *branch* contenente il prodotto finito;
- **Develop**: *branch* "d'appoggio" prima che venga effettuato un *push* di tutto il codice in Master;
- **ORM**: *branch* dedicato alla codifica delle tabelle che si sono pensate per l'applicativo;
- **Auth**: *branch* dedicato alla codifica dell'autenticazione per gli utenti;
- **Stores**: *branch* dedicato alla codifica degli *endpoint* per gli *store* (successivamente utilizzato per la codifica generale degi *endpoint* per tutti i modelli).

5.2.2 Codifica degli endpoint

Gli *endpoint* sono punti finali o URL all'interno di un'applicazione web o di un'API che corrispondono a determinate azioni o risorse. Parlando più in generale, essi consentono al *client* di inviare richieste HTTP ad un *server* e ottenere una risposta in base all'azione richiesta. Di seguito vengono descritte le principali funzionalità implementate per modello:

Login

POST:

- /api/login/access-token
  - Viene effettuato il *login* dell'utente, il quale una volta che ha inserito correttamente le proprie credenziali gli verrà ritornato il *token* d'accesso
- /api/login/test-token
  - Serve a testare se il *token* associato all'utente, una volta inserito, ritorna correttamente le informazioni dell'utente stesso

Users

POST:

- /api/users/create
  - Per la creazione di un nuovo utente e l'inserimento di esso all'interno del *database*. Richiede che l'utente loggato al momento della richiesta sia dotato dei privilegi di *superuser*
- /api/users/update
  - Per modificare le informazioni di base dell'utente. Solo l'utente loggato può modificare se stesso.

GET:

- /api/users/create
  - Per estrapolare le informazioni relative all'utente attualmente loggato.
- /api/users/list
  - Per ritornare una lista delle informazioni generali di tutti gli utenti presenti nel *database*, richiede che l'utente loggato al momento della richiesta sia dotato dei privilegi di *superuser*.
- /api/users/{username}
  - Per ritornare le informazioni specifiche dell'utente con *username* specificato nell'URL dell'*endpoint*. Richiede che l'utente loggato al momento della richiesta sia dotato dei privilegi di *superuser*.

DELETE:

- /api/users/{username}
  - Per eliminare l'utente di *username* specificato. Richiede che l'utente loggato al momento della richiesta sia dotato dei privilegi di *superuser*.

Stores

POST:

- /api/stores/create
  - Per la creazione di un nuovo *store*.
- /api/stores/share
  - Per la condivisione di un determinato *store* con altri utenti.

GET:

- /api/stores/{store_id}
  - Per ritornare le informazioni dello *store* specificato. Solo l'utente superuser potrà visualizzare anche gli *store* che non ha creato lui stesso.
- /api/stores/list
  - Per visualizzare la lista di tutti gli *store* che l'utente loggato ha creato. Solo l'utente *superuser* potrà visualizzare anche il resto di *store* da lui non creati.
- /api/stores/list-shares
  - Per visualizzare la lista di *store* condivisi dall'utente attualmente loggato.
- /api/stores/list-received
  - Per visualizzare la lista di *store* ricevuti in condivisione all'utente attualmente loggato.

DELETE:

- /api/stores/{store_id}
  - Per eliminare lo *store* specificato dal proprio id tra quelli creati dall'utente attualmente loggato. Se quest'ultimo è *superuser* potrà eliminare qualsiasi *store*. L'eliminazione di uno *store* comporta anche l'eliminazione di ogni documento in esso contenuto e la rispettiva cartella.
- /api/stores/revoke-all/{store_id}
  - Per revocare lo *share* di un determinato *store* con tutti gli utenti il quale esso è stato condiviso.

Documents

POST:

- /api/documents/upload
  - Per la pubblicazione di un *file* all'interno di uno *store* identificato dal proprio id, se non è già presente viene creata una *directory* nominata con l'id dello *store* nella quale al suoi interno viene salvato il *file*;
  - *Chunking* del documento inserito con estrapolazione del testo attraverso tika, l'inserimento dei vari pezzi di testo viene fatta attraverso il *vector database* Weaviate.

GET:

- /api/documents/document/{store_id}/{document_id}/
  - Per ritornare il documento sotto forma di *attachment* da scaricare. Solo gli utenti *superuser* possono accedere a qualsiasi documento di qualsiasi *store*.
- /api/documents/document-url/{document_id}
  - Per ritornare l'URL associato al documento specificato dal proprio id.
- /api/documents/list
  - Per ritornare la lista di documenti di un dato *store* creato dall'utente. Solo gli utenti *superuser* potranno visualizzare il contenuto di qualsiasi *store*.

DELETE:

- /api/documents/{document_id}
  - Per eliminare un dato documento identificato dal proprio id appartenente ad uno *store*. Solo gli utenti *superuser* potranno eliminare qualsiasi documento appartenente a qualsiasi *store*
- /api/documents/delete-all/{store_id}
  - Per eliminare tutti i documenti appartenenti ad uno *store* identificato dal proprio id. Solo gli utenti *superuser* potranno eliminare il contenuto di qualsiasi *store*.

Query

GET:

- /api/query
  - Per inviare una *query* ed interrogare lo *store* specificato.

5.2.3 Endpoint su Swagger

Si è usato Swagger, il quale è uno strumento ampiamente utilizzato per la documentazione, la visualizzazione e il *testing* delle API. È particolarmente utile quando si sviluppano API con *framework* come FastAPI, poiché semplifica la generazione automatica di documentazione interattiva e offre un'interfaccia utente per esplorare ed eseguire le richieste API direttamente dal *browser*. In figura 5.6 si mostra un piccolo scorcio di alcuni *endpoint* implementati e come appaiono nell'interfaccia di Swagger:

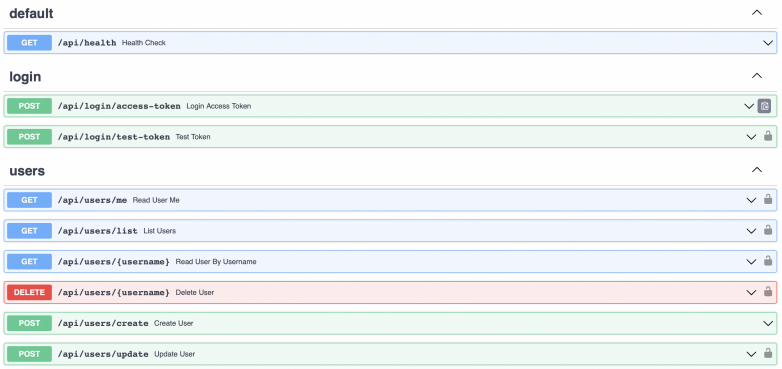


Figura 5.8: Interfaccia Swagger

Come si può notare, gli *endpoint* vengono suddivisi per soggetto di richiesta. Ad esempio, tutti gli *endpoint* relativi alla gestione degli utenti sono sotto il soggetto *"users"*. Ogni *endpoint* viene identificato poi anche per il tipo di richiesta HTTP, di seguito una breve spiegazione di quelle utilizzate:

- **GET**: viene utilizzata per recuperare dati o risorse dal *server*. È il tipo di richiesta utilizzato quando si inserisce un URL nel browser;
- **POST**: viene utilizzata per inviare dati al *server* per elaborazione. È comunemente utilizzata quando si inviano dati da un modulo HTML o quando si inviano dati JSON a un'API;
- **DELETE**: viene utilizzata per rimuovere una risorsa dal *server*.

Ogni *endpoint* può presentare il simbolo di un lucchetto, questo sta a significare che, affinchè la richiesta possa essere effettuata, l'utente deve essere autenticato correttamente.

5.2.4 Codifica Frontend

Per quanto riguarda la codifica del *frontend*, come già accennato poc'anzi, si è tenuto come riferimento il *mockup* fornito modificandone l'estetica in corso d'opera. Per la codifica si è deciso di utilizzare Streamlit, ovvero un *framework open-source* sviluppato per semplificare il processo di trasformazione dei modelli di *machine learning* e degli *script* Python in app interattive che possono essere condivise e utilizzate da chiunque senza richiedere competenze di sviluppo web.
Per prima cosa se si vuole sviluppare in Streamlit bisogna importare la libreria:

```
import streamlit as st
```

in questo modo, attraverso il modulo st si possono chiamare le principali funzioni di Streamlit.
Per quanto riguarda la configurazione del *client*, è stato utilizzato HTTPX, ovvero una libreria Python ad alte prestazioni per effettuare richieste HTTP. Nel dettaglio, la configurazione di esso è la seguente:

```
client = httpx.Client(
        base_url="http://localhost:8080/api/",
        headers={}
```

dove in "base_url" per l'appunto viene salvata la *path* usata come base per gli *endpoint*, la quale verrà completata in base alla funzionalità selezionata dall'utente.
Esempio:
Si vuole creare un nuovo utente, per fare ciò abbiamo bisogno della *path*

```
"http://localhost:8080/api/users/create"
```

Sapendo che "base_url" contiene "http://localhost:8080/api" avremo bisogno solo della parte rimanente, la quale si aggiunge nella seguente maniera:

```
def create_new_user(client: httpx.Client, username:str, full_name: str,
password: str, superuser: bool)->bool:
if st.button("Create New User"):
    r = client.post("users/create", json={"username": username,
    "full_name": full_name, "password": password, "superuser": superuser})
    if r.status_code == 200:
        st.write("User created")
        st.experimental_rerun()
    else:
        st.write("Something went wrong")
```

Questa sopra è una porzione di codice del *frontend* per la creazione di un nuovo utente, il quale svolge il compito di fare una richiesta HTTP di tipo POST solamente nel momento in cui l'utente clicca il bottone "Create New User". Nel dettaglio, sapendo che client contiene l'url base, nel momento in cui si chiama il metodo post (essendo la creazione dell'utente una richiesta di tipo POST) basta passare il rimanente dell'URL relativo alla richiesta, in questo caso "users/create". Importante anche l'inserimento degli elementi richiesti attraverso il parametro json.
Per quanto riguarda la struttura dell'applicativo si è deciso di mantenere un menù a comparsa laterale che permette di selezionare diverse pagine:

Home

Pagina che si apre di *default* non appena l'utente ha effettuato correttamente il *login*. In essa vi è possibile creare *store*, condividerli, eliminarli, aggiungere e rimuovere documenti allo *store* selezionato. Nel caso in cui l'utente loggato avesse i permessi di *superuser* allora potrà eliminare anche gli *store* che non ha creato, oltre ad eliminarne il contenuto.

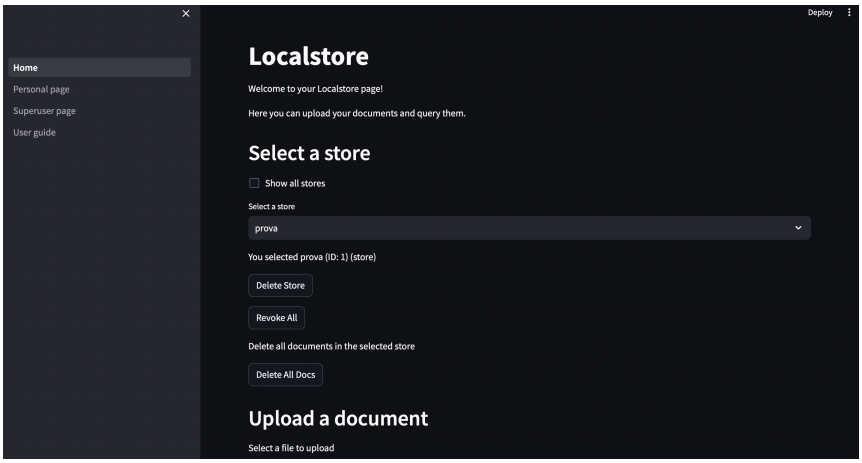


Figura 5.9: Home

Personal page

Pagina che presenta tutte le informazioni personali dell'utente che ha effettuato il *login*. Da questa è possibile modificarle, visualizzare tutti gli *store* creati dall'utente loggato, visualizzare gli *store* che ha condiviso e quelli ricevuti. Nel caso in cui l'utente loggato fosse *superuser* allora ha la possibilità di visualizzare tutti gli *store* presenti nella piattaforma.

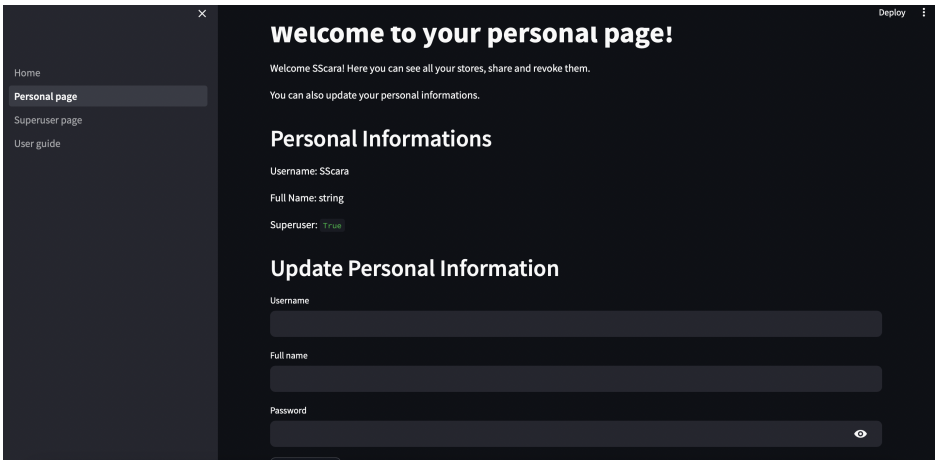


Figura 5.10: Personal Page

Superuser page

Pagina consultabile solamente se l'utente loggato ha i permessi di *superuser*. Questa contiene le funzionalità di creazione di un nuovo utente ed eliminazione dell'utente selezionato.

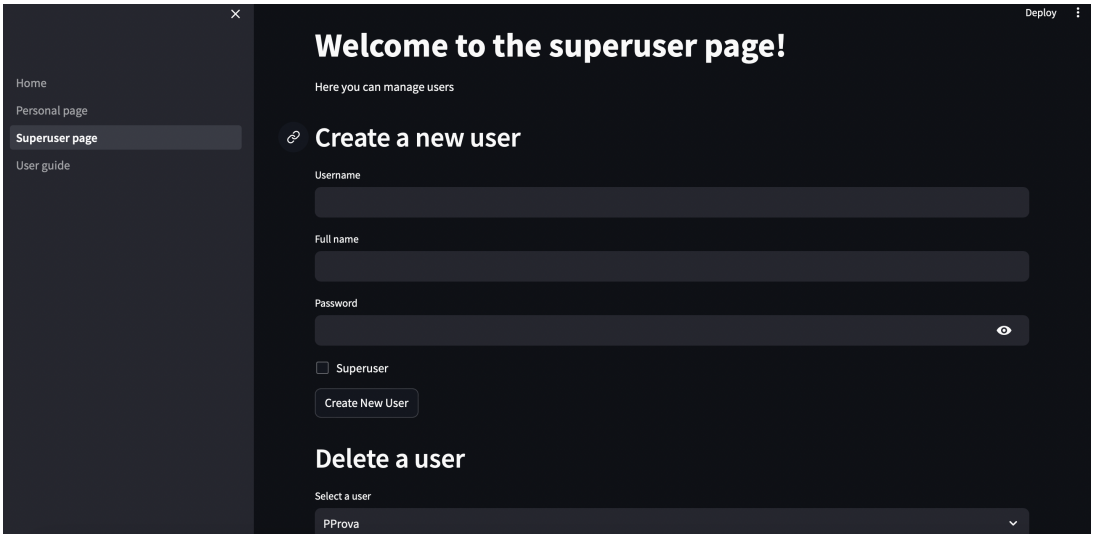


Figura 5.11: Superuser Page

User guide

Pagina consultabile anche da chi non ha eseguito il login. Contiene tutte le istruzione per l'utilizzo dell'applicativo.

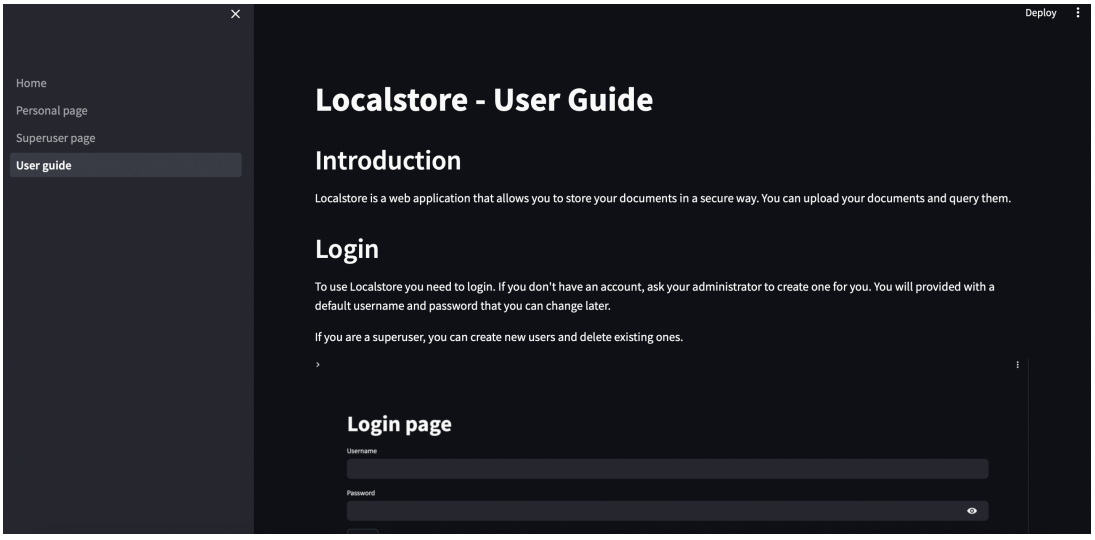


Figura 5.12: User guide

Capitolo 6

Verifica e validazione

Al termine di ogni fase del progetto si è proceduti con l'attività di verifica, per assicurarsi che quanto implementato fosse privo di *bug* e rispettasse i requisiti individuati. In questa sezione vengono riportate:

- **La strategia di verifica**, ovvero l'insieme degli obiettivi di qualità che si vogliono soddisfare, associati alle metriche utilizzate per quantificare tali obiettivi;
- Le **modalità di testing** con le quali è stato verificato il codice prodotto.

6.0.1 Strategie di verifica

Settimanalmente, assieme al *tutor* aziendale, sono stati svolti dei *meeting* su Teams per il controllo del corretto funzionamento delle funzioni implementate. Per fare ciò, si è utilizzata l'interfaccia Swagger con la quale, uno per uno, si andavano a testare gli *endpoint* sviluppati, di seguito un esempio di corretto funzionamento per la visualizzazione delle informazioni dell'utente per *username* inserito:

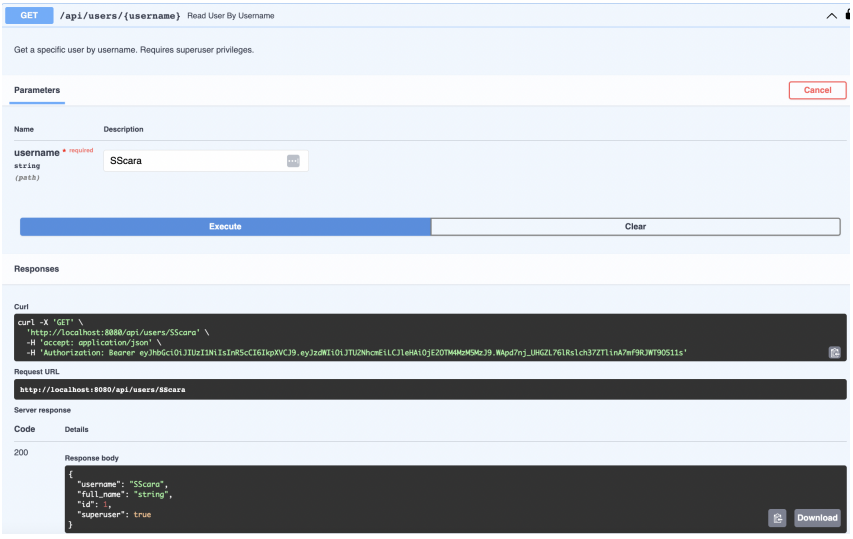


Figura 6.1: Test su Swagger

Per facilitare l'individuazione degli errori, sono stati inseriti dei blocchi di codice volti al lancio di eccezioni in base al problema riscontrato, ad esempio:

```
async def check_if_user_exists(db: AsyncSession, username: str) -> User:
"""Check if user exists."""
user = await crud.user.get_by_username(db, username=username)
if not user:
    raise HTTPException(
        status_code=status.HTTP_404_NOT_FOUND, detail="User not found"
    )
return user
```

Come si può notare, l'eccezione che viene lanciata non è altro che un codice HTTP. Di seguito viene elencata una lista di tutti i codici HTTP che sono stati utilizzati per questo progetto:

- **400 BAD REQUEST**: restituito da un *server* quando la richiesta inviata dal *client* non può essere elaborata perché è malformata, non valida o non conforme agli standard definiti per la comunicazione HTTP;
- **401 UNAUTHORIZED**: restituito da un *server* quando la richiesta inviata dal *client* richiede un'azione che necessita l'autenticazione o autorizzazione, ma le credenziali fornite dal client sono insufficienti o non valide;
- **404 NOT FOUND**: restituito da un *server* quando la risorsa richiesta dal *client* non è stata trovata sul *server*;
- **409 CONFLICT**: restituito da un *server* quando una richiesta del *client* provoca un conflitto con lo stato attuale delle risorse sul *server*.

Nel caso in cui uno tra questi codici è stato lanciato, si è passa ti fare un *debug* degli *endpoint* vittima di qualche *bug*.

6.0.2 Validazione

Per quanto riguarda la fase di validazione, si è entrati in questa solamente nel caso in cui nessuna eccezione è stata lanciata e dunque, apparentemente, il codice non dovrebbe aver riscontrato alcun problema. A seguito di ciò sono stati fatti dei controlli agli *output* resituiti dalle richieste per verificarne il corretto funzionamento. Se questi presentavano dei problemi si ritornava a fare le verifiche di rito tramite *debugging* del codice che generava problemi.

Capitolo 7

Conclusioni

7.1 Consuntivo finale

Il progetto di *stage* ha impiegato all'incirca poco più di 300 ore, in linea con quanto previsto. Come è possibile vedere dalla seguente tabella, ogni attività ha impiegato lo stesso numero di ore previste ed effettive:

7.1.1 Rapporto tra ore previste ed effettive

Attività	**Preventivo**	**Consuntivo**
Studio dell'architettura del progetto	24	24
Analisi dei requisiti	16	16
Studio delle tecnologie	32	32
Progettazione delle componenti richieste	40	40
Formazione aggiuntiva	16	16
Implementazione del *software*	140	140
Test e sperimentazione del *software*	24	24
Documentazione	48	48

Tabella 7.1: Tabella rappresentante il rapporto tra le ore previste ed effettive per ogni attività

Per quanto riguarda la fase di documentazione, non è stata una fase a se stante, bensì è stata svolta con l'avanzare del progetto. Qualsiasi modifica o scelta progettuale veniva redatta in appositi documenti.

7.2 Raggiungimento degli obiettivi

Nella sezione denominata Obiettivi prefissati, sono stati elencati tutti gli obiettivi obbligatori al fine di trarre un prodotto il più completo possibile. Di seguito viene mostrata una tabella elencante gli esiti per ognuno di essi:

7.2.1 Resoconto degli obiettivi raggiunti

ID	Priorità	Completamento
O01	Obbligatorio	Soddisfatto
O02	Obbligatorio	Soddisfatto
O03	Obbligatorio	Soddisfatto
O04	Obbligatorio	Soddisfatto
O05	Obbligatorio	Soddisfatto
F01	Facoltativo	Soddisfatto
F02	Facoltativo	Non soddisfatto
F03	Facoltativo	Non soddisfatto

Tabella 7.2: Tabella rappresentante il completamento dei vari obiettivi prefissati

7.3 Conoscenze acquisite

Durante l'esperienza di *stage*, lo stagista ha avuto modo di approfondire più nel dettaglio i LLM ed il loro funzionamento. Lo studio del linguaggio di programmazione Python, non è mai stato così dettagliato come in questo *stage*, dove si è lasciata piena libertà di gestione nell'apprendimento, ed un continuo affiancamento da parte del tutor aziendale, il quale contribuiva nel dare consigli e materiale sul quale approfondire. L'utilizzo di strumenti come Docker Desktop ha contribuito ad arricchire le conoscenze dello stagista sotto gli aspetti di configurazione di *server* locali in grado di far avviare applicativi e per la gestione documentale, con appoggio di *vector database* per il contenimento delle porzioni di testo estratte.

7.4 Valutazione personale

Valuto il mio periodo di *stage* molto positivamente, in quanto la maggior parte delle tecnologie e strumenti utilizzati durante questa esperienza non sono mai stati approfonditi prima. Il tutto dunque ha contribuito senz'altro ad arricchire il mio bagaglio personale delle conoscenze nell'ambito informatico e non solo, anche quello relativo alla comunicazione in un ambito aziendale o più in generale in quello lavorativo. La valutazione positiva è stata sicuramente influenzata da un ambiente di lavoro molto giovanile e alla mano, fin da subito si sono andati ad instaurare dei rapporti lavorativi molto solidi e funzionali che hanno reso lo *stage* una vera e propria esperienza più che un obbligo. Grazie a questa opportunità ho avuto modo di capire cos'è veramente il mondo del lavoro, il quale molto diverso dal pensiero medio dello studente; tutti gli impegni improrogabili, tempistiche da rispettare e le fatiche sono state delle sfide difficili da superare, ma il loro superamento ha contribuito ad arricchire il mio bagaglio culturale.

Glossario

Application Programming Interface (API)

Si tratta di un insieme di regole e protocolli che permettono a diversi *software* di comunicare tra loro. Le API definiscono i metodi e i formati di dati che le applicazioni possono utilizzare per richiedere e scambiare informazioni. Sono fondamentali nello sviluppo software moderno per consentire l'integrazione tra sistemi diversi e la riusabilità del codice. Le API nascondono la complessità interna delle applicazioni e forniscono un'interfaccia standardizzata per l'interazione tra *software*.

Knowledge Base (KB)

E un archivio strutturato di informazioni, dati e conoscenze relative a un particolare argomento o dominio specifico. La *knowledge base* è progettata per immagazzinare, organizzare e rendere facilmente accessibili le informazioni utili a una particolare azienda, organizzazione o sistema.

Vector Database

È un tipo di *database* specializzato progettato per l'archiviazione e la gestione di dati rappresentati come vettori matematici. Questi vettori possono rappresentare oggetti, documenti o dati complessi e sono utilizzati per calcolare la similarità tra gli elementi del *database*. Un esempio di *database* vettoriale è Weaviate.

Diagramma di Gantt

Strumento di gestione dei progetti utilizzato per pianificare e visualizzare le attività in un progetto nel tempo. È costituito da una tabella a doppia entrata in cui le attività sono elencate verticalmente lungo l'asse delle ordinate e il tempo è rappresentato orizzontalmente lungo l'asse delle ascisse. Le attività sono rappresentate da barre orizzontali all'interno del diagramma, la cui lunghezza rappresenta la durata prevista delle attività.

Servizi REST

Sono servizi web basati su architettura REST (Representational State Transfer) che consentono a *client* e *server* di comunicare attraverso internet. Utilizzano gli standard HTTP per l'accesso, la modifica e la gestione delle risorse, che sono identificate da URL. Questi servizi seguono principi di progettazione chiari, come l'uso di verbi HTTP standard (GET, POST, PUT, DELETE) e la rappresentazione delle risorse in formato JSON o XML. Vengono ampiamente utilizzati per l'integrazione tra sistemi software distribuiti e applicazioni web, offrendo una comunicazione efficiente e scalabile.

Jupyter Notebook

Jupyter Notebook è un'applicazione *open source* ampiamente utilizzata nell'ambito della scienza dei dati, dell'analisi dei dati e della programmazione interattiva. Consente agli utenti di creare e condividere documenti interattivi chiamati *"notebook"* che integrano testo, codice, grafici e *output* in un'unica interfaccia.

Unified Modeling Language (UML)

È un linguaggio di modellazione visuale utilizzato nell'ingegneria del *software* e in altre discipline per rappresentare, documentare e comunicare il design e la struttura dei sistemi complessi. È uno *standard* industriale riconosciuto e ampiamente utilizzato.

Virtual Private Network (VPN)

È una tecnologia che crea una connessione sicura e crittografata tra due o più dispositivi o reti attraverso Internet. La principale funzione di una VPN è quella di garantire la *privacy* e la sicurezza delle comunicazioni online, consentendo agli utenti di navigare in modo anonimo e proteggere i propri dati da accessi non autorizzati.

Git

È un sistema di controllo di versione distribuito, ampiamente utilizzato per la gestione del codice sorgente e la collaborazione nello sviluppo *software*. Creato da Linus Torvalds nel 2005, Git è noto per la sua velocità, efficienza e flessibilità ed è utilizzato da sviluppatori di tutto il mondo per tenere traccia delle modifiche apportate al codice e gestire progetti *software* di varie dimensioni.

Global Interpreter Lock (GIL)

Caratteristica specifica dell'interprete CPython di Python, che è la versione più comune di Python utilizzata. Il GIL è un meccanismo di blocco che consente a solo

un thread di Python di eseguire codice Python alla volta, anche su sistemi con più *core* CPU.

Object-Relation Mapping (ORM)

Tecnica di programmazione che consente di mappare oggetti di un linguaggio di programmazione orientato agli oggetti (come Python o Java) a tabelle in un *database* relazionale. L'obiettivo principale dell'ORM è semplificare l'interazione tra il codice dell'applicazione e il *database*, consentendo agli sviluppatori di manipolare i dati del *database* utilizzando oggetti e operazioni orientate agli oggetti invece di scrivere *query* SQL manualmente.

WebSocket

Protocollo di comunicazione bidirezionale che consente l'interazione in tempo reale tra un *server* e un *client* su Internet. A differenza dei protocolli tradizionali basati su HTTP, WebSocket offre una connessione persistente e a bassa latenza che permette al *server* di inviare dati al *client* non appena sono disponibili, senza la necessità di richieste periodiche da parte del client.

Large Language Models (LLM)

Sono una classe di modelli di intelligenza artificiale (IA) progettati per comprendere e generare linguaggio naturale su vasta scala. Questi modelli utilizzano reti neurali profonde con decine o centinaia di miliardi di parametri per apprendere la struttura e il significato del linguaggio umano.

Proof of Concept (PoC)

È una dimostrazione o un esperimento che mira a dimostrare la fattibilità o la validità di un'idea, di un concetto o di una tecnologia. In sostanza, il PoC è una prova preliminare che verifica se un progetto, un prodotto o un sistema può essere sviluppato e implementato con successo.

Mockup

È una rappresentazione visiva o un prototipo di un prodotto o di un sistema, spesso utilizzato nelle prime fasi di sviluppo di un progetto per visualizzare l'aspetto e il funzionamento previsti. I *mockup* sono utilizzati per comunicare un'idea o un concetto a un pubblico o a un team di sviluppo, consentendo loro di comprendere come apparirà e funzionerà il prodotto finale.