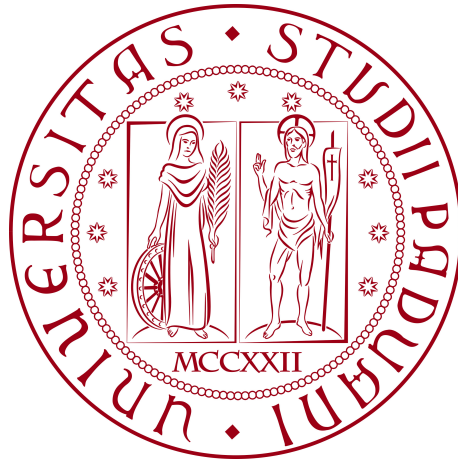


Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Una web app basata sulla tecnica  
"Retrieval-Augmented Generation"

*Tesi di Laurea Triennale*

*Relatore*

Prof. Baldan Paolo

*Laureando*

Zaccone Rosario

Matricola 2043680

---

ANNO ACCADEMICO 2023-2024



# Ringraziamenti

Desidero esprimere la mia gratitudine al professor Baldan Paolo, mio relatore, per l'aiuto e il sostegno che mi ha dato durante la stesura dell'elaborato.

Vorrei anche ringraziare, con affetto, i miei genitori e i miei nonni per il loro sostegno, e il team di Halue per l'esperienza di tirocinio svolta presso la loro azienda.

Padova, Settembre 2024

*Zaccone Rosario*

# Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage dal laureando Rosario Zaccone, presso l'azienda Halue srl. La finalità dello stage era implementare un'applicazione che consentisse all'utente di ottenere informazioni riguardo un noto brand, utilizzando le più recenti tecniche di intelligenza artificiale. Lo scopo del documento è fornire una panoramica sul prodotto realizzato e sull'analisi, progettazione, codifica e testing che hanno portato alla realizzazione di esso.

# Indice

<b>Glossario</b>	<b>xii</b>
<b>1 Introduzione</b>	<b>1</b>
1.1 Azienda e progetto . . . . .	1
1.2 Esplorazione tecnologica . . . . .	2
1.2.1 Approccio . . . . .	2
1.2.2 Tecnica AI . . . . .	3
1.2.3 Framework RAG . . . . .	5
1.3 Prodotto ottenuto . . . . .	6
1.4 Organizzazione del testo . . . . .	7
<b>2 Analisi dei requisiti</b>	<b>8</b>
2.1 Casi d'uso . . . . .	8
2.1.1 UC01: Upload file . . . . .	10
2.1.1.1 UC01.1 Caricamento file nello storage . . . . .	11
2.1.1.2 UC01.2 Inserimento fonte . . . . .	11
2.1.1.3 UC01.3 Caricamento file nel database . . . . .	12
2.1.2 UC02: Upload file txt . . . . .	12
2.1.3 UC03: Upload file PDF . . . . .	13
2.1.3.1 UC03.1: Selezione testo estratto dal PDF . . . . .	14
2.1.3.2 UC03.2: Modifica testo estratto dal PDF . . . . .	15
2.1.3.3 UC03.3: Selezione immagini estratte dal PDF . . . . .	15
2.1.3.4 UC03.4: Inserimento descrizione immagini estratte dal PDF . . . . .	15
2.1.4 UC04: Upload file immagine . . . . .	16
2.1.4.1 UC04.1: Inserimento descrizione immagine . . . . .	17

2.1.5	UC05: Chat . . . . .	17
2.1.5.1	UC05.1: Inserimento messaggio . . . . .	18
2.1.5.2	UC05.2: Visualizzazione messaggio . . . . .	18
2.1.6	UC06: Configurazione ambiente . . . . .	19
2.1.6.1	UC06.1: Configurazione LLM . . . . .	20
2.1.6.2	UC06.2: Configurazione embedding model . . . . .	20
2.1.6.3	UC06.3: Configurazione temperatura . . . . .	20
2.1.6.4	UC06.4: Configurazione prompt . . . . .	21
2.1.7	UC07: Visualizzazione file caricati . . . . .	21
2.1.7.1	UC07.1: Visualizzazione lista file . . . . .	22
2.1.7.2	UC07.2: Visualizzazione informazioni singolo file . . . . .	22
2.1.8	UC08: Visualizzazione embedding estratti dai file . . . . .	23
2.1.8.1	UC08.1: Visualizzazione lista embedding . . . . .	23
2.1.8.2	UC08.2: Visualizzazione informazioni singolo embedding . . . . .	24
2.1.9	UC09: Ricerca embedding . . . . .	24
2.1.10	UC10: Ricerca embedding per file . . . . .	25
2.1.10.1	UC10.1: Inserimento nome del file per la ricerca . . . . .	25
2.1.10.2	UC11.1: Inserimento testo embedding per la ricerca . . . . .	26
2.1.11	UC12: Eliminazione embedding . . . . .	27
2.1.12	UC13: Modifica embedding . . . . .	27
2.1.12.1	UC13.1: Aggiornamento contenuto testuale . . . . .	28
2.1.13	UC14: Visualizzazione errore upload nello storage . . . . .	28
2.1.14	UC15: Visualizzazione errore upload nel db vettoriale . . . . .	29
2.1.15	UC16: Visualizzazione errore configurazione . . . . .	29
2.1.16	UC17: Visualizzazione errore eliminazione embedding . . . . .	29
2.1.17	UC18: Visualizzazione errore modifica embedding . . . . .	30
2.2	Requisiti . . . . .	30
2.2.1	Requisiti funzionali . . . . .	30
2.2.2	Requisiti non funzionali . . . . .	37
<b>3</b>	<b>Progettazione</b>	<b>38</b>
3.1	Tecnologie e strumenti . . . . .	38

---

3.1.1	Pinecone . . . . .	38
3.1.2	MariaDB . . . . .	38
3.1.3	LangServe . . . . .	39
3.1.4	PyMuPDF . . . . .	39
3.1.5	PyTest . . . . .	40
3.1.6	Remix . . . . .	40
3.1.6.1	React . . . . .	40
3.1.6.2	Tailwind CSS . . . . .	40
3.1.7	Git e GitHub . . . . .	40
3.2	Architettura . . . . .	41
3.2.1	Database vettoriale . . . . .	44
3.2.2	Database relazionale . . . . .	45
3.2.2.1	Analisi . . . . .	45
3.2.2.2	Progettazione concettuale . . . . .	46
3.2.2.3	Progettazione logica . . . . .	46
3.2.3	Data Access Layer . . . . .	46
3.2.4	Business Logic Layer . . . . .	48
3.2.4.1	Schemi di validazione . . . . .	48
3.2.5	Presentation Layer . . . . .	50
3.2.5.1	Componenti . . . . .	50
3.2.6	Design Pattern . . . . .	52
3.2.6.1	Singleton . . . . .	52
3.2.6.2	Facade . . . . .	53
3.2.6.3	RAII . . . . .	53
3.2.6.4	Dependency Injection . . . . .	54
3.2.6.5	MVC . . . . .	54
<b>4</b>	<b>Realizzazione e testing</b>	<b>56</b>
4.1	Codifica . . . . .	56
4.1.1	Backend . . . . .	56
4.1.2	Frontend . . . . .	60
4.2	Testing . . . . .	63

4.3	Prodotto finale . . . . .	64
4.3.1	Home . . . . .	65
4.3.2	Upload . . . . .	66
4.3.2.1	Upload file testo . . . . .	67
4.3.2.2	Upload file immagine . . . . .	68
4.3.2.3	Upload file PDF . . . . .	69
4.3.3	Chat . . . . .	70
4.3.4	Configurazione . . . . .	71
4.3.5	Knowledge . . . . .	72
<b>5</b>	<b>Conclusioni</b>	<b>74</b>
5.1	Analisi critica . . . . .	74
5.2	Conoscenze acquisite . . . . .	75
	<b>Bibliografia</b>	<b>i</b>
	<b>Sitografia</b>	<b>ii</b>

# Elenco delle figure

1.1	Schema della RAG . . . . .	4
2.1	Diagramma UML di tutti i principali use case . . . . .	9
2.2	Diagramma UML dello use case UC01 (Upload file) . . . . .	10
2.3	Diagramma UML dello use case UC03 (Upload file PDF) . . . . .	13
2.4	Diagramma UML dello use case UC04 (Upload file immagine) . . . . .	16
2.5	Diagramma UML dello use case UC05 (Chat) . . . . .	17
2.6	Diagramma UML dello use case UC06 (Configurazione ambiente) . . . . .	19
2.7	Diagramma UML dello use case UC07 (Visualizzazione file caricati) . . . . .	21
2.8	Diagramma UML dello use case UC08 (Visualizzazione embedding estratti dai file) . . . . .	23
2.9	Diagramma UML dello use case UC10 (Ricerca embedding per file) . . . . .	25
2.10	Diagramma UML dello use case UC11 (Ricerca embedding per testo) . . . . .	26
2.11	Diagramma UML dello use case UC13 (Modifica embedding) . . . . .	27
3.1	Diagramma della Three-Layer Architecture . . . . .	42
3.2	Diagramma dell'architettura usata nel progetto . . . . .	43
3.3	Endpoint esposti dall'API . . . . .	44
3.4	Esempi di embedding visti dal web client di Pinecone . . . . .	45
3.5	Comparazione di performance fra MariaDB e Pinecone . . . . .	45
3.6	Diagramma ER . . . . .	46
3.7	Diagramma UML delle classi del data access layer . . . . .	47
3.8	Diagramma UML delle classi del business logic layer . . . . .	48
3.9	Schemi di validazione . . . . .	49
3.10	Schemi di validazione . . . . .	49

3.11	Schemi di validazione . . . . .	50
3.12	Diagramma UML del pattern Singleton . . . . .	53
3.13	Diagramma UML del pattern Facade . . . . .	53
3.14	Diagramma del MVC . . . . .	55
4.1	Schermata di home . . . . .	65
4.2	Menu e sezione renderizzata . . . . .	66
4.3	Schermata di caricamento nello storage . . . . .	67
4.4	Schermata di successo di caricamento nello storage . . . . .	67
4.5	Schermata di upload di file di testo completato . . . . .	68
4.6	Schermata di upload di file immagine . . . . .	69
4.7	Schermata di upload di file PDF, selezione testo . . . . .	70
4.8	Schermata di upload di file PDF, selezione immagine . . . . .	70
4.9	Schermata di chat . . . . .	71
4.10	Schermata di configurazione . . . . .	72
4.11	Schermata di gestione file . . . . .	72
4.12	Schermata di gestione vettori . . . . .	73

## Elenco delle tabelle

2.1	Tabella del tracciamento dei requisiti funzionali, sezione di upload . . . . .	32
2.2	Tabella del tracciamento dei requisiti funzionali, sezione di chat . . . . .	33
2.3	Tabella del tracciamento dei requisiti funzionali, sezione di configurazione . . . . .	34
2.4	Tabella del tracciamento dei requisiti funzionali, sezione di knowledge . . . . .	36
2.5	Tabella del tracciamento dei requisiti di vincolo . . . . .	37

# Elenco dei codici sorgenti

4.1	Parte del codice della classe <code>DatabaseAccess</code> . . . . .	57
4.2	Parte del codice della classe <code>ChatService</code> . . . . .	58
4.3	Parte del codice della classe <code>Utilities</code> . . . . .	59
4.4	Parte del codice dell'API . . . . .	60
4.5	Codice del componente <code>Chat</code> . . . . .	61
4.6	Codice del componente <code>Chat</code> . . . . .	62
4.7	Esempio di action Remix . . . . .	63
4.8	Esempio di classe di testing . . . . .	64

# Glossario

**agile** Nella gestione dei progetti e nello sviluppo software, la metodologia agile è un insieme di pratiche e principi che enfatizzano la flessibilità, la collaborazione, e il miglioramento continuo. Il suo obiettivo principale è di rispondere in modo rapido ed efficiente ai cambiamenti e alle esigenze dei clienti attraverso cicli di sviluppo iterativi e incrementali.. [1](#)

**API** Un' Application Programming Interface (API) è un insieme di regole, protocolli e funzioni che permettono a diverse applicazioni software di comunicare tra loro per scambiare dati.. [1](#), [6](#), [38–40](#), [42](#), [43](#), [56](#), [75](#)

**chain** In LangChain, una chain è una sequenza strutturata di passaggi modulari che vengono eseguiti in successione per completare un compito complesso. Ogni passaggio della catena può coinvolgere diversi componenti come modelli linguistici, template di prompt, parser di output o integrazioni con strumenti esterni. Le chain permettono di costruire flussi di lavoro articolati, dove l'output di un passaggio diventa l'input del successivo, creando così pipeline efficienti per l'elaborazione e l'analisi dei dati.. [6](#), [39](#)

**community** Nell'informatica, una community è un gruppo di utenti che collaborano, condividono conoscenze, risolvono problemi e contribuiscono allo sviluppo e miglioramento di software o tecnologie.. [2](#), [5](#), [6](#)

**cosine** In Pinecone, e in generale nella data analysis, la cosine similarity è tecnica euristica per misurare similitudine tra due vettori calcolando il coseno dell'angolo compreso tra di due.. [44](#)

**embedding** Nel machine learning, gli embedding sono vettori creati da modelli di apprendimento automatico a partire da uno specifico oggetto, con lo scopo di catturarne i suoi dati significativi.. [3](#), [4](#), [10](#), [12](#), [14](#), [16](#), [19](#), [20](#), [22–38](#), [44](#), [45](#), [47](#), [51](#), [52](#), [72](#)

**KNN** Nel machine learning, il K-nearest neighbors (KNN) è un algoritmo utilizzato per compiti di classificazione o rigressione.. [3](#), [75](#)

**endpoint** In un'API, un endpoint è un URL che funge da punto di contatto tra un client e un server API. Il client invia richieste agli endpoint API per accedere alle sue funzionalità.. [6](#), [43](#)

**framework** Struttura di base su cui si costruiscono e organizzano componenti o sistemi complessi. Nel contesto dell'ingegneria del software e dello sviluppo di applicazioni, un framework fornisce un insieme standard di funzionalità e regole che semplificano il processo di sviluppo, promuovendo la riusabilità del codice e l'adozione di pratiche standard. I framework possono includere librerie, strumenti e linee guida per supportare gli sviluppatori nel creare applicazioni robuste ed efficienti.. [2](#), [5](#), [6](#), [37](#), [39](#), [40](#), [64](#), [75](#)

**hook** In React, funzione che permette di utilizzare lo stato e altre funzionalità nei componenti funzionali, senza dover utilizzare classi. Gli hook consentono di gestire lo stato, eseguire effetti collaterali, e rendere i componenti più modulari e riutilizzabili.. [50](#), [54](#), [60](#)

**knowledge** Insieme di documenti e immagini che fornisce informazioni e contesto riguardo un brand.. [1](#), [3](#), [6](#), [7](#), [21–28](#), [30](#), [33](#), [34](#), [37](#), [38](#), [50–52](#)

**LLM** Un Large Language Model (LLM) è un modello di linguaggio di grandi dimensioni addestrato su una vasta quantità di dati testuali per comprendere e generare linguaggio naturale. Gli LLM sono capaci di eseguire una varietà di compiti linguistici, come traduzione, completamento di testo, risposta a domande e molto altro, grazie alla loro struttura avanzata e alla capacità di apprendere pattern complessi presenti nei dati di addestramento.. [3–5](#), [19](#), [20](#), [33](#), [48](#), [51](#)

**ORM** L'Object-Relational Mapping (ORM) è una tecnica di programmazione che consente di convertire dati tra sistemi di tipo oggetto e sistemi di database relazionali. Utilizzando un ORM, gli sviluppatori possono interagire con il database utilizzando oggetti del linguaggio di programmazione piuttosto che comandi SQL, semplificando l'accesso e la gestione dei dati. Questa tecnica mappa le classi a tabelle del database e gli attributi delle classi a colonne, facilitando operazioni come il recupero, l'inserimento, l'aggiornamento e la cancellazione dei dati.. [46](#), [47](#)

**PoC** Nell'ingegneria del software, un Proof of Concept (PoC) è una demo che viene utilizzata per studiare la fattibilità di un'idea e per esplorare le funzionalità di una tecnologia oggetto di studio.. [2](#), [5](#)

**production readiness** Nell'informatica, condizione di un software in cui è pronto per essere rilasciato per l'utente finale.. [2](#)

**RAG** La Retrieval-Augmented Generation (RAG) è una tecnica di intelligenza artificiale che combina la ricerca e il recupero di informazioni di uno specifico dominio con la generazione di testo. La RAG utilizza modelli di recupero per trovare e fornire contenuti rilevanti da un database o da fonti esterne, che vengono poi utilizzati da modelli di generazione per produrre risposte o testi più informati e pertinenti. Questo approccio migliora la qualità e la precisione delle risposte, integrando conoscenze specifiche e aggiornate nel processo di generazione del linguaggio.. [3-7](#), [38](#), [70](#), [75](#)

**RAII** Resource Acquisition Is Initialization (RAII) è un'idioma di programmazione utilizzato principalmente in C++, in cui l'acquisizione di una risorsa, come memoria o file, è legata al ciclo di vita di un oggetto. In RAII, una risorsa viene acquisita durante la costruzione di un oggetto e rilasciata automaticamente alla sua distruzione.. [52](#), [53](#), [56](#)

**RDBMS** Il Relational Database Management System (RDBMS) è un tipo di sistema di gestione di database che utilizza un modello relazionale per gestire i dati.. [38](#)

**REST** Representational State Transfer (REST) è un'architettura per la progettazione di servizi web che utilizza il protocollo HTTP per la comunicazione tra client e server.

È basata su principi come l'uso di URL univoci per identificare risorse e metodi HTTP (GET, POST, PUT, DELETE) per manipolarle.. [43](#), [75](#)

**UML** Unified Modeling Language (UML) è un linguaggio di modellazione standardizzato utilizzato per specificare, visualizzare, realizzare, modificare e documentare gli artefatti di un sistema software. Per artefatto si intende qualunque prodotto tangibile del progetto legato al sistema, come codice, documentazione, tabelle del database.. [8](#), [38](#)

**use case** Scenario o insieme di scenari che descrivono come un sistema, un'applicazione o un servizio verrà utilizzato da un utente o da un altro sistema. Nel contesto dell'ingegneria del software, un use case rappresenta una descrizione formale delle interazioni tra un attore (utente o sistema esterno) e il sistema stesso, con l'obiettivo di raggiungere un determinato risultato. Gli use case sono utilizzati per definire i requisiti funzionali e fornire una base per la progettazione e lo sviluppo.. [8](#), [30](#)

**web app** Applicazione software accessibile tramite un browser web. [1](#), [6](#), [40](#), [42](#)



# Capitolo 1

## Introduzione

### 1.1 Azienda e progetto

L'azienda che ha ospitato lo stage è *Halue*, che si occupa di consulenza tecnologica specializzata nello sviluppo di soluzioni software per l'e-commerce. L'attività di stage è stata svolta in modalità mista, alcuni giorni in presenza ed altri in smart-working, seguendo i principi della metodologia *agile<sub>G</sub>*. *Colnago* ha avuto un ruolo attivo, partecipando periodicamente alle riunioni di stato avanzamento lavori.

Il progetto parte dalla richiesta di un cliente di *Halue*, *Colnago*, che ha chiesto la realizzazione di un chatbot o un sistema di Q&A, basato sulle recenti tecniche di intelligenza artificiale, che potesse fornire all'utente la possibilità di chiedere e ricevere informazioni riguardo il brand in questione. A partire da ciò, si è pensato di realizzare una *web app<sub>G</sub>*, per l'utilizzo lato amministrativo, che presenta delle sezioni per caricare la *knowledge<sub>G</sub>* aziendale (foto, documenti, cataloghi) e per testare il chatbot.

Durante la fase di pianificazione sono stati individuati tre obiettivi principali. Il primo è lo sviluppo di un'*API<sub>G</sub>* che offre le funzionalità basate sull'intelligenza artificiale e relative alla gestione dell'intero sistema, come l'upload della *knowledge<sub>G</sub>* e l'uso della chat. Il secondo obiettivo riguarda l'implementazione di una *web app<sub>G</sub>* che offra un'interfaccia utente per utilizzare il sistema attraverso queste *API<sub>G</sub>*. Infine, è stata considerata la possibilità di integrare nell'applicazione un'interfaccia per la gestione della *knowledge<sub>G</sub>* aziendale.

## 1.2 Esplorazione tecnologica

Le prime settimane dello stage sono state dedicate all'esplorazione tecnologica, per comprendere bene il dominio del problema e individuare le tecnologie più adatte. L'esplorazione si è focalizzata principalmente sull'individuare una tecnica basata sull'intelligenza artificiale su cui basare l'intero sistema e di seguito individuare un *framework<sub>G</sub>* che potesse semplificarne la realizzazione. Di seguito è descritta la metodologia seguita in questo periodo di stage, insieme alle nuove tecnologie analizzate.

### 1.2.1 Approccio

L'approccio seguito per individuare e studiare le possibili tecnologie è il seguente:

- Studio del dominio del problema e dell'ambito;
- Individuazione delle tecnologie principali, attraverso call con esperti del settore a contatto con l'azienda e attraverso consultazioni di discussioni su forum famosi come *StackOverflow*;
- Realizzazione di piccoli *PoC<sub>G</sub>* per testare come le tecnologie analizzate si comportino nell'ambito di studio e le funzionalità che offrono;
- Valutazione delle tecnologie, basata sull'esperienza personale e sulla consultazione di articoli a tema;
- Discussione con il tutor interno per prendere una scelta finale.

I criteri presi in considerazione per prendere una scelta tecnologica sono stati i seguenti:

- Attinenza delle funzionalità offerte della tecnologia con il progetto da realizzare;
- *community<sub>G</sub>*;
- *production readiness<sub>G</sub>*;
- Curva di apprendimento;
- *Pricing*.

## 1.2.2 Tecnica AI

Il primo passo dell'esplorazione tecnologica è consistito nello scegliere la tecnica di base su cui basare il progetto, partendo dall'idea di aggiungere la *knowledge<sub>G</sub>* aziendale ad un modello già esistente in modo che potesse soddisfare le richieste dell'utente riguardo il brand. Sono state individuate due tecniche principali, il *fine-tuning* e la *RAG<sub>G</sub>*, ed è stato consultato l'articolo [7] per approfondire le loro differenze.

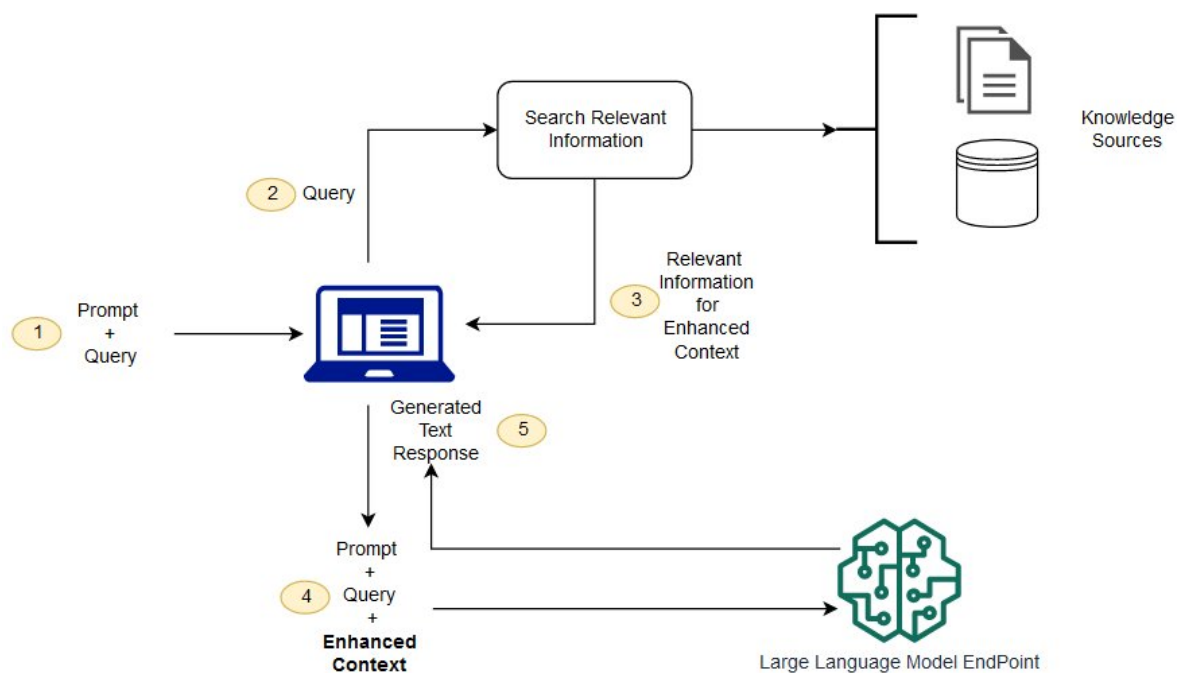
Il *fine-tuning*, come descritto in [15], è un processo di ottimizzazione che si applica a modelli di apprendimento automatico preaddestrati. Consiste nel prendere un modello già addestrato su un grande set di dati generici e adattarlo a un compito specifico utilizzando un set di dati più piccolo e pertinente al contesto desiderato.

Il *fine-tuning* migliora le prestazioni del modello focalizzandolo su compiti specifici, permette di incorporare nuove conoscenze nel modello, rendendolo più esperto in un determinato dominio e non richiede prompt di grandi dimensioni. A suo discapito, però, è un processo intensivo in termini di risorse, difatti richiede una notevole potenza computazionale in fase di training.

La *RAG<sub>G</sub>* è una tecnica avanzata che combina la ricerca e il recupero della *knowledge<sub>G</sub>* con la generazione di testo. Questa metodologia prevede la popolazione di un database vettoriale con la *knowledge<sub>G</sub>* specifica di un dominio. Quando viene ricevuta una richiesta, il sistema effettua una ricerca nel database per identificare il contesto pertinente e crea un prompt arricchito da sottoporre a un *LLM<sub>G</sub>*. Quest'ultimo utilizza le informazioni fornite per generare la risposta finale. La tecnica è illustrata nell'immagine 1.1, estratta da [3].

In dettaglio, dai documenti e dalle diverse risorse vengono estratti dei vettori, chiamati *embedding<sub>G</sub>*, e caricati nel database vettoriale. Gli *embedding<sub>G</sub>* possono essere arricchiti con metadati, quali il file di origine, la rappresentazione testuale, la fonte, il numero di pagina e l'autore del documento. Quando viene sottoposta una query, anche essa viene convertita in *embedding<sub>G</sub>* utilizzando modelli appositi. Successivamente, un algoritmo di classificazione, come il *KNN<sub>G</sub>*, viene utilizzato per identificare gli *embedding<sub>G</sub>* relativi alla *knowledge<sub>G</sub>* più rilevante. A questo punto, viene costruito un prompt seguendo un tem-

plate definito dallo sviluppatore, che combina la query originale con i contenuti estratti dal database vettoriale (principalmente metadati, poiché la loro rappresentazione vettoriale è impiegata esclusivamente nella fase di ricerca). Il prompt viene quindi inviato a un  $LLM_G$ , il quale elabora una risposta utilizzando le sue capacità generative e il contesto estratto. È possibile specificare nel prompt se il  $LLM_G$  debba utilizzare solo il contesto fornito o anche la sua conoscenza preesistente.



**Figura 1.1:** Schema della RAG

La  $RAG_G$  migliora l'accuratezza del modello, incorporando nel prompt del contesto rilevante, presenta costi più bassi del *fine-tuning*, poiché creare  $embedding_G$  è più economico rispetto ad addestrare un modello. È altamente scalabile poiché non richiede un ampio riaddestramento del modello in caso di aggiunta di nuova conoscenza. A suo discapito presenta dei prompt con dimensioni molto grandi, andando a consumare quindi un alto numero di *token*. Fatte queste premesse, si è ritenuta la  $RAG_G$  la tecnica ideale per il progetto in questione, in quanto capace di dare una buona accuratezza contestuale a un costo iniziale contenuto, offrendo anche una buona scalabilità.

### 1.2.3 Framework RAG

Per implementare la  $RAG_G$  si è deciso di utilizzare un  $framework_G$ , come è solito fare in progetti del genere. Ne sono stati individuati due  $framework_G$ : LangChain e LlamaIndex.

Alcuni pregi e difetti sono stati rilevati grazie ad una discussione su StackOverflow [4], e ne è stata testata la loro validità con lo sviluppo di piccoli  $PoC_G$ .

Come descritto in [2], «LangChain è un framework open source per la creazione di applicazioni basate su modelli linguistici di grandi dimensioni (LLM). Gli LLM sono grandi modelli di deep learning pre-addestrati su grandi quantità di dati in grado di generare risposte alle query degli utenti, ad esempio rispondere a domande o creare immagini da prompt basati su testo. LangChain fornisce strumenti e astrazioni per migliorare la personalizzazione, l'accuratezza e la pertinenza delle informazioni generate dai modelli. Ad esempio, gli sviluppatori possono utilizzare i componenti LangChain per creare nuove catene di prompt o personalizzare i modelli esistenti. LangChain include anche componenti che consentono agli LLM di accedere a nuovi set di dati senza riqualificazione.»

Come descritto in [9], LlamaIndex è un  $framework_G$  che facilita l'integrazione di modelli linguistici avanzati con dati aggiuntivi. Fornisce strumenti per connettere  $LLM_G$  a diverse fonti di dati, come database e documenti, permettendo di costruire applicazioni personalizzate e potenti.

Entrambi sono molto utilizzati dalla comunità, supportano i linguaggi di programmazione Python e JavaScript, ma presentano delle differenze.

Per quanto riguarda la  $community_G$ , LlamaIndex ha raccolto un discreto seguito su GitHub con 34.000 stelle. LangChain, invece, vanta un seguito ancora più vasto, con oltre 90.000 stelle. La documentazione di entrambi è discreta, ma LlamaIndex si distingue per una documentazione molto gradevole e intuitiva, che include un chatbot basato sull'intelligenza artificiale per assistere l'utente nella navigazione e comprensione dei contenuti. D'altra parte, LangChain offre una documentazione che si concentra molto sugli esempi pratici, in particolare sui casi d'uso del progetto di stage. Un altro aspetto fondamentale riguarda il loro scopo d'uso. LlamaIndex non è pensato per essere un  $framework_G$  *general-purpose*, mentre LangChain è progettato per un uso più ampio e versatile, rendendolo adatto a una varietà di scenari e applicazioni. LlamaIndex eccelle nell'ottimizzazione

della ricerca e del recupero dei dati, rendendolo una scelta eccellente per progetti che richiedono un accesso rapido e preciso alle informazioni. LangChain, invece, offre una serie di servizi avanzati, come LangServe e LangSmith, che semplificano il deployment e il testing delle applicazioni.

Il *framework<sub>G</sub>* adatto al progetto è stato ritenuto LangChain. I fattori che hanno portato a ciò sono stati i servizi aggiuntivi che semplificano deployment e testing, una *community<sub>G</sub>* più ampia, le molteplici funzionalità che offre che lo rendono adatto ad un progetto che in futuro si pone ad avere una natura *general-purpose*. Inoltre il concetto di *chain<sub>G</sub>* su cui si basa astrae elegantemente il concetto di *RAG<sub>G</sub>* e ne aiuta a capire il funzionamento per chi è alle prime armi con essa.

LangChain è disponibile sia in Python che in Javascript/Typescript: è stata scelta la sua versione Python, in quanto la corrispondente in Javascript risulta essere più indietro con le funzionalità, con alcune feature necessarie al progetto in beta e con una *community<sub>G</sub>* minore. Inoltre LangServe è disponibile solo in Python.

### 1.3 Prodotto ottenuto

Sono state realizzate due applicazioni, che vanno a costituire un sistema in grado di testare il chatbot e caricare e gestire la *knowledge<sub>G</sub>*:

- ***API<sub>G</sub>***: *API<sub>G</sub>* che fornisce gli *endpoint<sub>G</sub>* per realizzare le principali operazioni correlate al pattern *RAG<sub>G</sub>*, come il caricamento della *knowledge<sub>G</sub>* del brand, l'interazione con il chatbot e la gestione dei contenuti caricati. È stata realizzata come servizio separato dalla *web app<sub>G</sub>* per esigenze tecniche e per una maggiore estensibilità, dato che queste funzionalità serviranno in futuro anche al chatbot finale presente sul sito del brand;
- ***web app<sub>G</sub>***: *web app<sub>G</sub>* che fornisce un'interfaccia grafica per l'utente in modo da poter utilizzare le funzionalità offerte dall'*API<sub>G</sub>*. Il suo utilizzo è destinato a *Colnago*. Presenta 4 sezioni principali: *Upload* per il caricamento della *knowledge<sub>G</sub>*, *knowledge<sub>G</sub>* per la gestione della stessa, *Configurazione* per la configurazione dell'intero sistema, *Chat* per il test del chatbot.

I punti deboli della *RAG<sub>G</sub>* si concentrano tutti sulla scarsa qualità dei dati contestuali caricati nel database vettoriale, ma per sopperire a ciò si è pensato di includere nel progetto una sezione che consente al brand di fare upload della *knowledge<sub>G</sub>* aziendale in maniera intelligente, aggiungendo metadata ai documenti e selezionando accuratamente le informazioni estratte da questi ultimi, apportando modifiche a queste se necessario. Le miglorie date da questa scelta sono notevoli poiché, per esempio, nel caricamento di un file PDF consente di andare a selezionare solo il testo e le immagini utili, andando ad escludere contenuto non desiderato, come immagini di poco valore informativo, e andando a migliorare i frammenti di testo estratti effettuando delle correzioni in modo da aumentare la loro qualità.

### 1.4 Organizzazione del testo

**Il secondo capitolo** approfondisce l'Analisi dei Requisiti;

**Il terzo capitolo** approfondisce la progettazione del sistema;

**Il quarto capitolo** tratta della codifica e del testing del software, facendo anche una panoramica sul prodotto finale;

**Nel quinto capitolo** vengono tratte le conclusioni finali sull'esperienza svolta.

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;
- i nomi di variabili, funzioni, classi e quant'altro, sono evidenziati con lo stile *monospace*;
- i termini in lingua straniera non comuni nel linguaggio italiano o facenti parti del gergo tecnico sono evidenziati con lo stile *corsivo*.

# Capitolo 2

## Analisi dei requisiti

Il seguente capitolo descrive l'Analisi dei Requisiti del progetto. Per individuare i requisiti, si è partiti con l'individuazione degli *use case<sub>G</sub>*, rappresentati formalmente con i classici diagrammi *UML<sub>G</sub>* e in maniera testuale. Per maggiori dettagli sui diagrammi *UML<sub>G</sub>*, si rimanda a [11]. A partire da essi, sono stati redatti i requisiti, suddivisi in due categoria: requisiti funzionali e requisiti non funzionali.

### 2.1 Casi d'uso

Nella figura 2.1 è mostrata una panoramica di tutti gli *use case<sub>G</sub>*. Nelle varie sottosezioni viene discusso ogni *use case<sub>G</sub>* in maniera approfondita, illustrando i principali scenari e mostrando diagrammi per i relativi *sub-use case<sub>G</sub>*

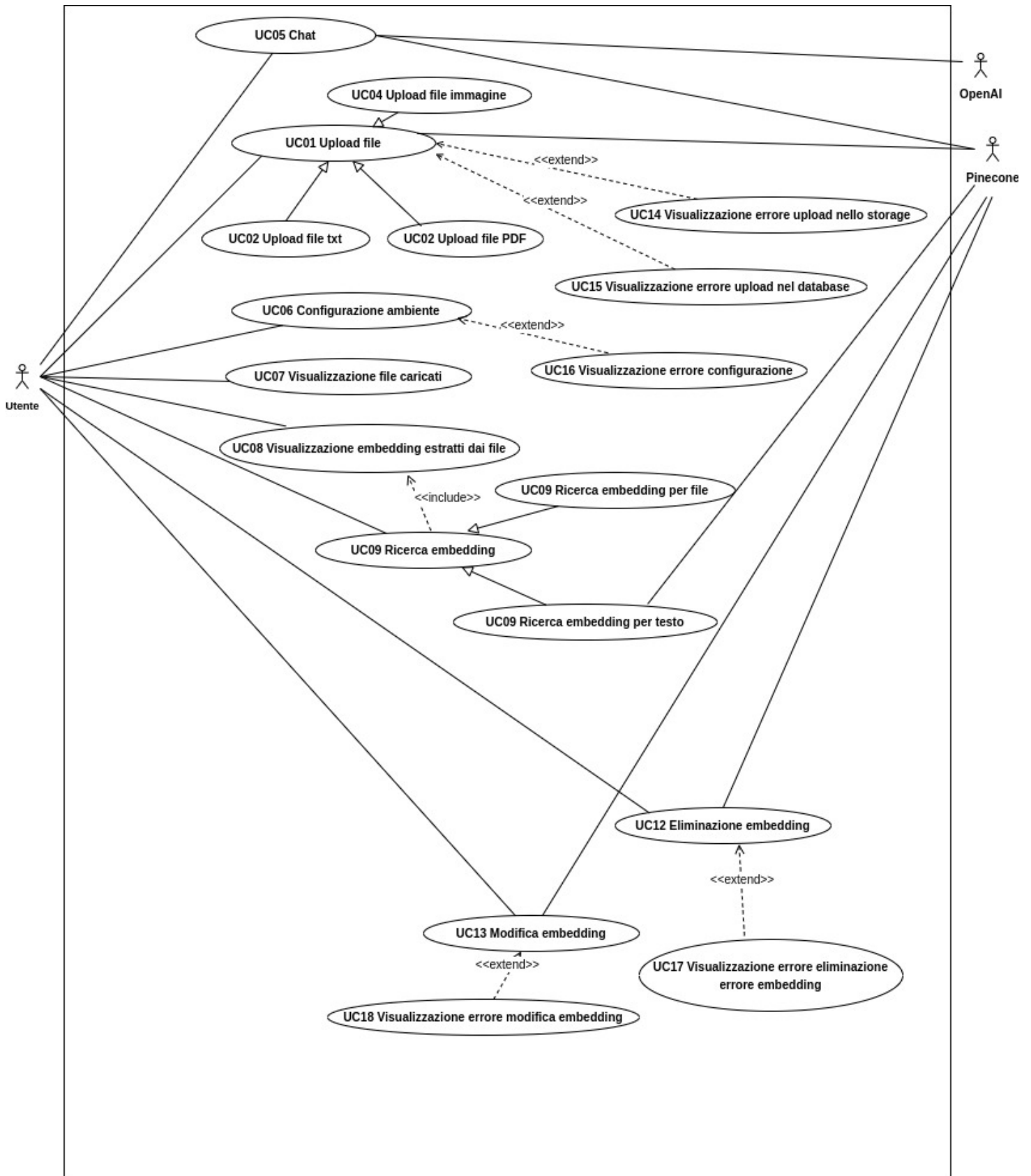


Figura 2.1: Diagramma UML di tutti i principali use case

### 2.1.1 UC01: Upload file

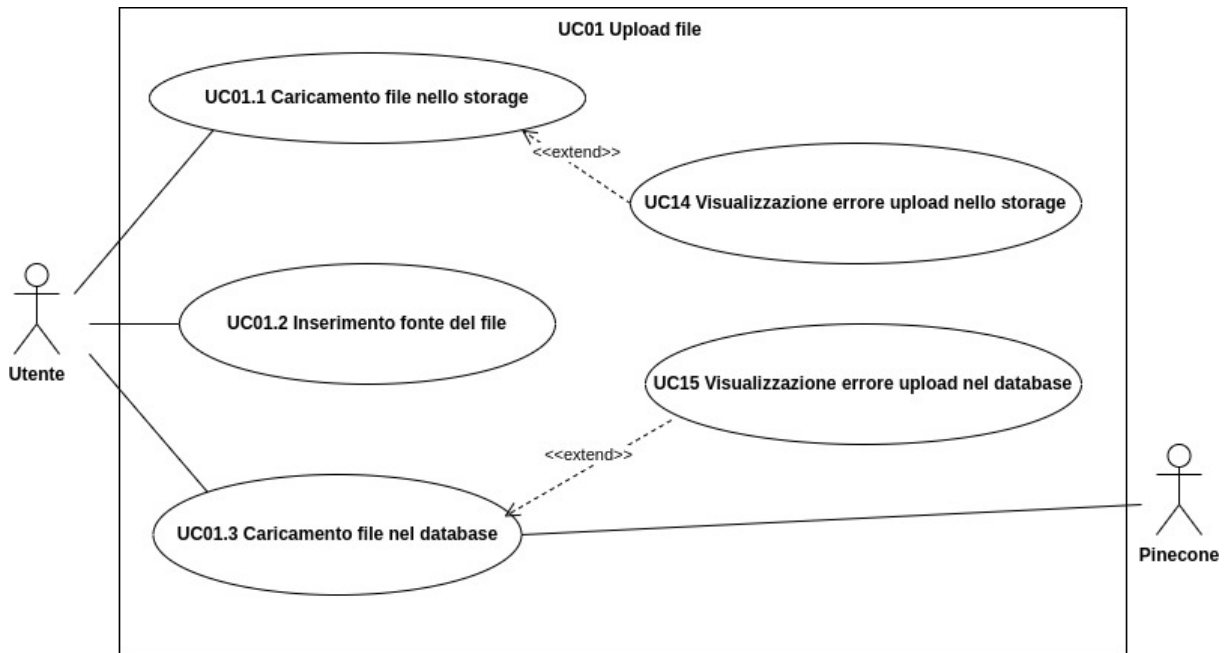


Figura 2.2: Diagramma UML dello use case UC01 (Upload file)

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione di upload

**Post-condizioni:** Gli *embedding<sub>G</sub>* estratti dal file sono caricati nel database vettoriale, il file è caricato nello storage, informazioni su file ed *embedding<sub>G</sub>* sono caricate nel database relazionale

**Scenario principale:**

- 1 (UC01.1) Il file viene caricato nello storage
- 2 (UC01.2) L'utente inserisce la fonte
- 3 (UC01.3) Pinecone carica al suo interno gli *embedding<sub>G</sub>* estratti dal file e le relative informazioni, insieme a quelle del file, vengono caricate nel database relazionale

**Scenari alternativi:**

- Scenario alternativo 1

1.a (UC14) Viene visualizzato un messaggio di errore legato al caricamento del file nello storage

– Scenario alternativo 2

1 (UC01.1) Il file viene caricato nello storage

2 (UC01.2) L'utente inserisce la fonte

3.a (UC15) Viene visualizzato un messaggio di errore legato al caricamento del file nei database

### 2.1.1.1 UC01.1 Caricamento file nello storage

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload

**Post-condizioni:** Il file è caricato nello storage

**Scenario principale:**

- 1 L'utente seleziona il file da caricare
- 2 Il sistema carica il file nello storage]

**Scenari alternativi:**

– Scenario alternativo 1

- 1 L'utente seleziona il file da caricare
- 1.a Il sistema prova a caricare il file nello storage
- 1.b (UC14) Viene visualizzato un messaggio di errore legato al caricamento del file nello storage

### 2.1.1.2 UC01.2 Inserimento fonte

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload

**Post-condizioni:** La fonte viene inserita

**Scenario principale:**

- 1 L'utente inserisce la fonte del file

### 2.1.1.3 UC01.3 Caricamento file nel database

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, il file è stato caricato nello storage e la fonte è stata inserita

**Post-condizioni:** Il file è caricato nel database vettoriale e le relative informazioni nel database relazionale

**Scenario principale:**

- 1 Pinecone carica al suo interno il file
- 2 Informazioni sul file e sui suoi relativi *embedding<sub>G</sub>* sono caricate nel database relazione

**Scenari alternativi:**

- Scenario alternativo 1

1.a (UC15) Viene visualizzato un messaggio di errore legato al caricamento del file nei database

### 2.1.2 UC02: Upload file txt

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, L'utente si trova nella sezione di upload

**Post-condizioni:** Gli *embedding<sub>G</sub>* estratti dal file sono caricati nel database vettoriale, il file è caricato nello storage, informazioni su *embedding<sub>G</sub>* e file sono caricate nel database relazionale

**Scenario principale:**

- 1 (UC01.1) Il file viene caricato nello storage
- 2 (UC01.2) L'utente inserisce la fonte
- 3 (UC01.3) Pinecone carica al suo interno gli *embedding<sub>G</sub>* estratti dal file e le informazioni, insieme a quelle del file, vengono caricati nel database relazionale

**Scenari alternativi:**

– Scenario alternativo 1

1.a (UC14) Viene visualizzato un messaggio di errore legato al caricamento del file nello storage

– Scenario alternativo 2

1 (UC02.1) Il file viene caricato nello storage

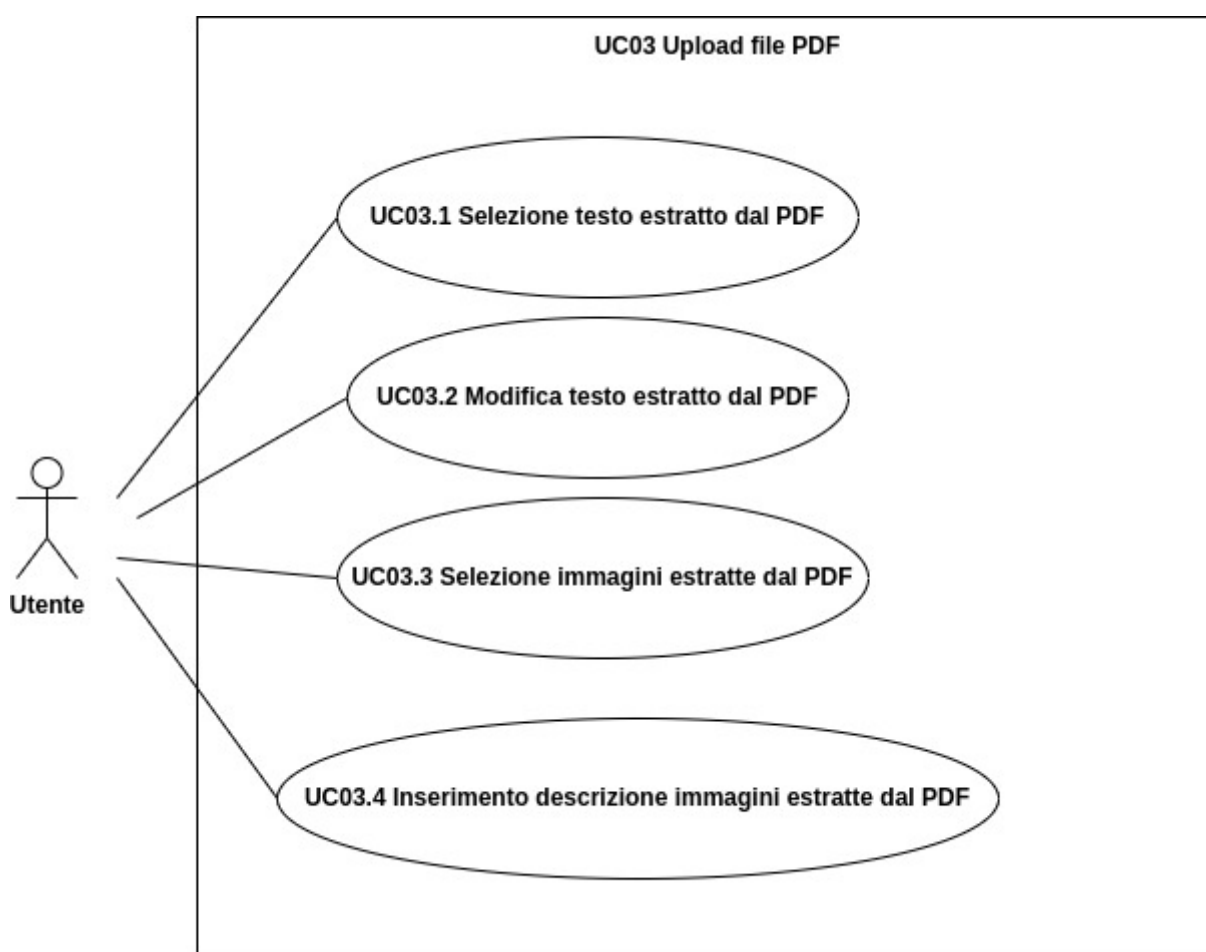
2 (UC02.2) L'utente inserisce la fonte

3.a (UC15) Viene visualizzato un messaggio di errore legato al caricamento del file nei database

**Eredita da: UC01**

La descrizione dei sotto casi d'uso è omessa, in quanto identica a quella di UC01.

### 2.1.3 UC03: Upload file PDF



**Figura 2.3:** Diagramma UML dello use case UC03 (Upload file PDF)

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione di upload

**Post-condizioni:** Gli *embedding<sub>G</sub>* estratti dal file sono caricati nel database vettoriale, il file caricato nello storage, informazioni su file ed *embedding<sub>G</sub>* sono caricate nel database relazionale

### Scenario principale:

- 1 (UC01.1) Il file viene caricato nello storage
- 2 (UC01.2) L'utente inserisce la fonte
  - 3 Il sistema estrae frammenti di testo e immagini dal file PDF
- 4 (UC03.1) L'utente seleziona i frammenti di testo estratti dal PDF
- 5 (UC03.2) L'utente modifica il contenuto dei frammenti di testo estratti dal PDF
- 6 (UC03.3) L'utente seleziona le immagini dal PDF
- 7 (UC03.4) L'utente inserisce una descrizione per le immagini selezionate
- 8 (UC01.3) Pinecone carica al suo interno gli *embedding<sub>G</sub>* estratti dal file e le informazioni, insieme a quelle del file, vengono caricati nel database relazionale

### Scenari alternativi:

– Scenario alternativo 1

- 1.a (UC14) Viene visualizzato un messaggio di errore legato al caricamento del file nello storage

– Scenario alternativo 2

- 1 (UC01.1) Il file viene caricato nello storage
  - 2 .. 7 Stessi passi dello scenario principale
- 8.a (UC15) Viene visualizzato un messaggio di errore legato al caricamento del file nei database

**Eredita da:** UC01

#### 2.1.3.1 UC03.1: Selezione testo estratto dal PDF

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, il PDF è stato caricato nello storage, sono stati estratti testo e immagini dal PDF

**Post-condizioni:** Il frammento di testo da caricare è selezionato

### **Scenario principale:**

- 1 L'utente seleziona il frammento di testo estratto dal PDF

### **2.1.3.2 UC03.2: Modifica testo estratto dal PDF**

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, il PDF è stato caricato nello storage, l'utente ha selezionato il frammento di testo da modificare

**Post-condizioni:** Il frammento di testo selezionato è aggiornato

### **Scenario principale:**

- 1 L'utente modifica il frammento di testo estratto dal PDF, aggiornandone il suo contenuto

### **2.1.3.3 UC03.3: Selezione immagini estratte dal PDF**

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, il PDF è stato caricato nello storage, sono stati estratti testo e immagini dal PDF

**Post-condizioni:** L'immagine da caricare è selezionata

### **Scenario principale:**

- 1 L'utente seleziona l'immagine estratta dal PDF

### **2.1.3.4 UC03.4: Inserimento descrizione immagini estratte dal PDF**

**Attore principale:** Utente

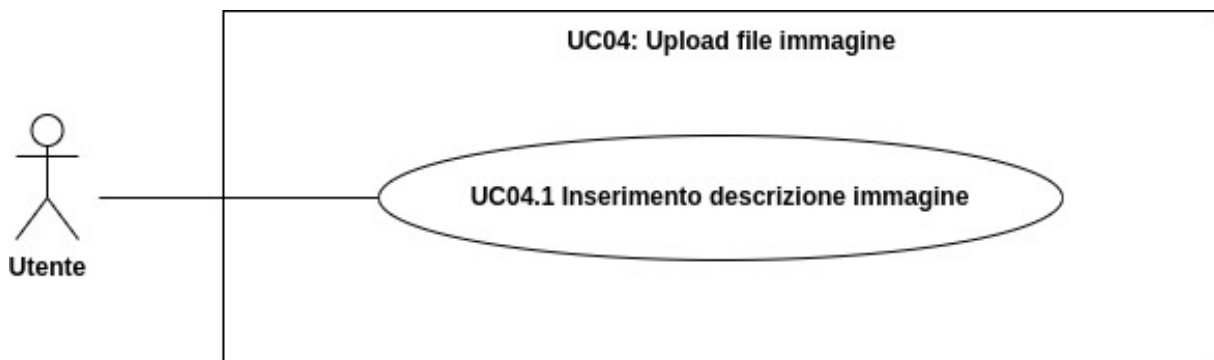
**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, il PDF è stato caricato nello storage, l'utente ha selezionato l'immagine

**Post-condizioni:** Viene inserita una descrizione per l'immagine

### **Scenario principale:**

- 1 L'utente inserisce una descrizione per l'immagine estratta dal PDF

### 2.1.4 UC04: Upload file immagine



**Figura 2.4:** Diagramma UML dello use case UC04 (Upload file immagine)

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione di upload

**Post-condizioni:** Gli *embedding<sub>G</sub>* estratti dal file sono caricati nel database vettoriale, il file è caricato nello storage, informazioni su file ed *embedding<sub>G</sub>* sono caricate nel database relazionale

**Scenario principale:**

- 1 (UC01.1) Il file viene caricato nello storage
- 2 (UC01.2) L'utente inserisce la fonte
- 3 (UC04.1) L'utente inserisce la descrizione dell'immagine
- 4 (UC01.3) Pinecone carica al suo interno gli *embedding<sub>G</sub>* estratti dal file e le informazioni, insieme a quelle del file, vengono caricati nel database relazionale

**Scenari alternativi:**

– Scenario alternativo 1

- 1.a (UC14) Viene visualizzato un messaggio di errore legato al caricamento del file nello storage

– Scenario alternativo 2

1 .. 3 Come nello scenario principale

- 4.a (UC15) Viene visualizzato un messaggio di errore legato al caricamento del file nei database

**Eredita da:** UC01

### 2.1.4.1 UC04.1: Inserimento descrizione immagine

**Attore principale:** Utente

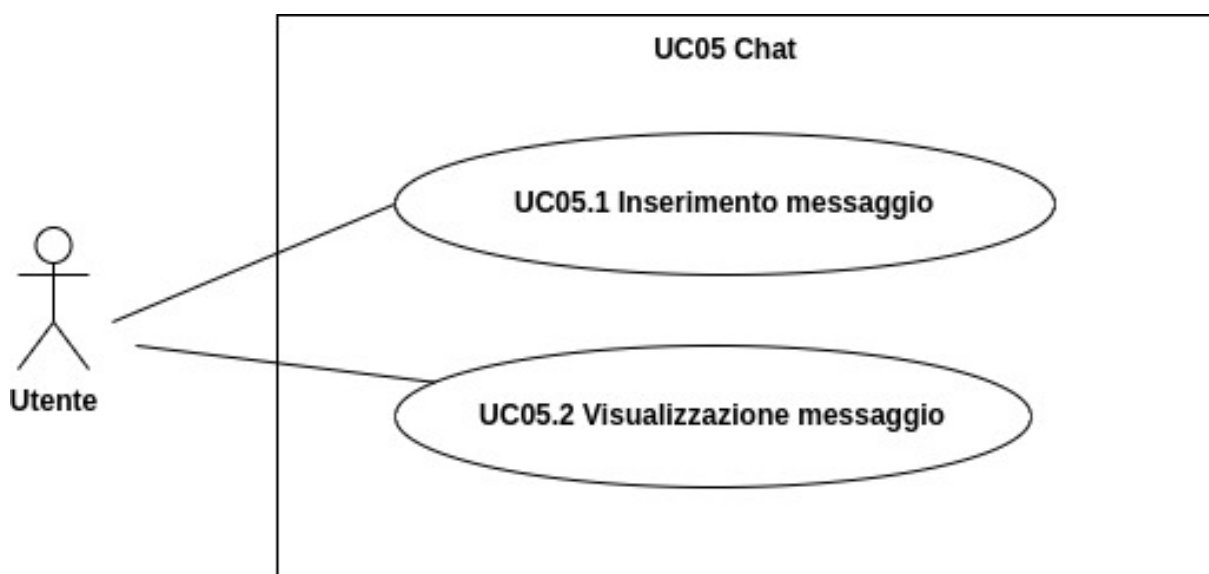
**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, l'immagine è stata caricata nello storage

**Post-condizioni:** La descrizione dell'immagine è inserita nel sistema

**Scenario principale:**

- 1 L'utente inserisce la descrizione dell'immagine

### 2.1.5 UC05: Chat



**Figura 2.5:** Diagramma UML dello use case UC05 (Chat)

**Attore principale:** Utente

**Attori secondari:** Pinecone, OpenAI

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, i servizi OpenAI sono operativi, l'utente si trova nella sezione di chat

**Post-condizioni:** L'utente tiene una conversazione con il sistema

**Scenario principale:**

- 1 (UC05.1) L'utente inserisce un messaggio
  - 2 Il sistema elabora il messaggio e ne fornisce un output
- 3 (UC05.2) L'utente visualizza il messaggio in output

### 2.1.5.1 UC05.1: Inserimento messaggio

**Attore principale:** Utente

**Attori secondari:** Pinecone, OpenAI

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, i servizi OpenAI sono operativi, l'utente si trova nella sezione di chat

**Post-condizioni:** L'utente inserisce un messaggio nel sistema

**Scenario principale:**

- 1 L'utente inserisce un messaggio

### 2.1.5.2 UC05.2: Visualizzazione messaggio

**Attore principale:** Utente

**Attori secondari:** Pinecone, OpenAI

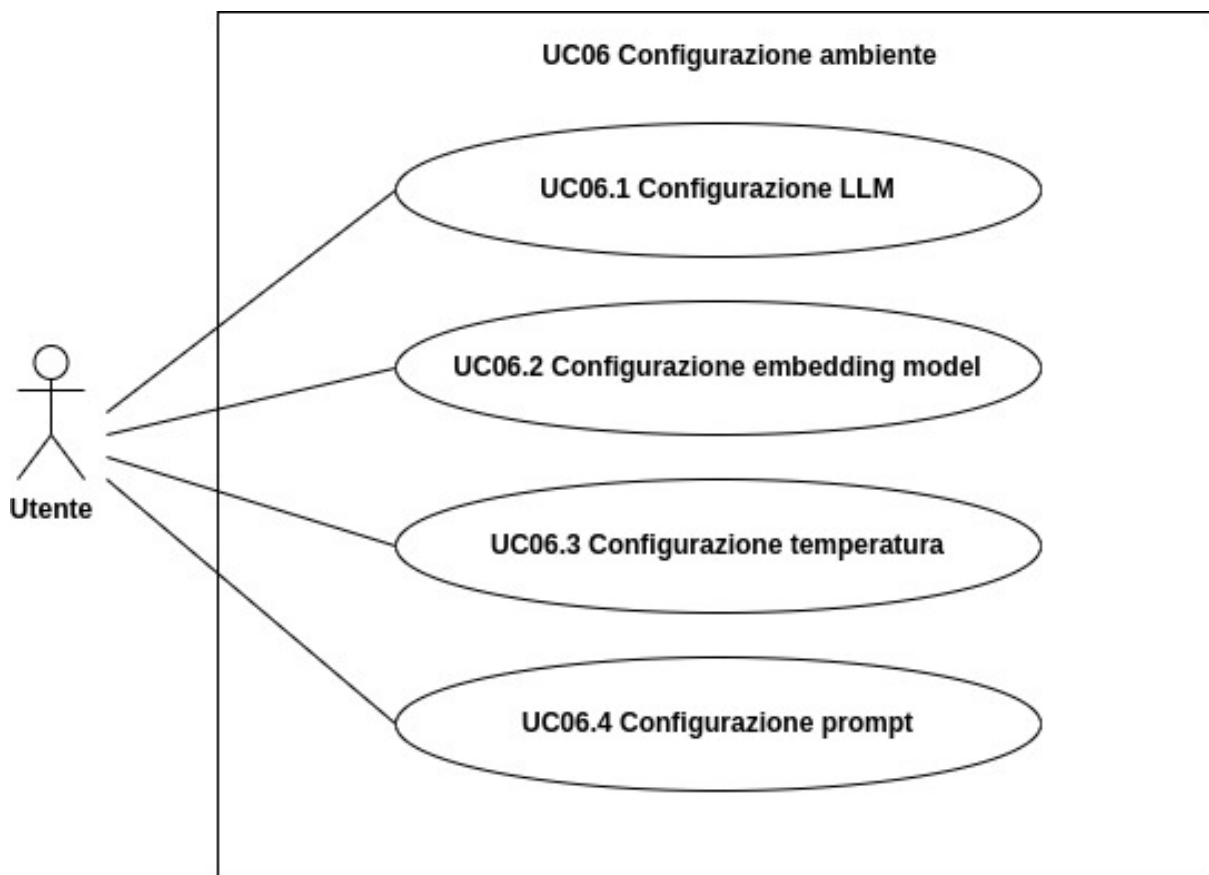
**Precondizioni:** Il sistema è operativo, Pinecone è operativo, i servizi OpenAI sono operativi, l'utente si trova nella sezione di chat

**Post-condizioni:** L'utente visualizza la risposta del sistema in output

**Scenario principale:**

- 1 L'utente visualizza la risposta del sistema

## 2.1.6 UC06: Configurazione ambiente



**Figura 2.6:** Diagramma UML dello use case UC06 (Configurazione ambiente)

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione

**Post-condizioni:** L'utente aggiorna la configurazione del sistema

**Scenario principale:**

- 1 (UC06.1) L'utente configura il *LLM<sub>G</sub>*
- 2 (UC06.2) L'utente configura il modello per generare gli *embedding<sub>G</sub>*
- 3 (UC06.3) L'utente configura la temperatura
- 4 (UC06.4) L'utente configura il prompt
- 5 Il sistema aggiorna la sua configurazione

**Scenari alternativi:**

– Scenario alternativo 1

5.a (UC16) Viene visualizzato un messaggio di errore legato alla configurazione del sistema

### 2.1.6.1 UC06.1: Configurazione LLM

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione

**Post-condizioni:** L'utente inserisce un nuovo *LLM<sub>G</sub>*

**Scenario principale:**

1 L'utente inserisce un nuovo *LLM<sub>G</sub>*

### 2.1.6.2 UC06.2: Configurazione embedding model

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione

**Post-condizioni:** L'utente inserisce un nuovo *embedding<sub>G</sub>* model

**Scenario principale:**

1 L'utente inserisce un nuovo *embedding<sub>G</sub>* model

### 2.1.6.3 UC06.3: Configurazione temperatura

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione

**Post-condizioni:** L'utente imposta il valore della temperatura per il modello

**Scenario principale:**

1 L'utente inserisce il valore della temperatura desiderata

#### 2.1.6.4 UC06.4: Configurazione prompt

**Attore principale:** Utente

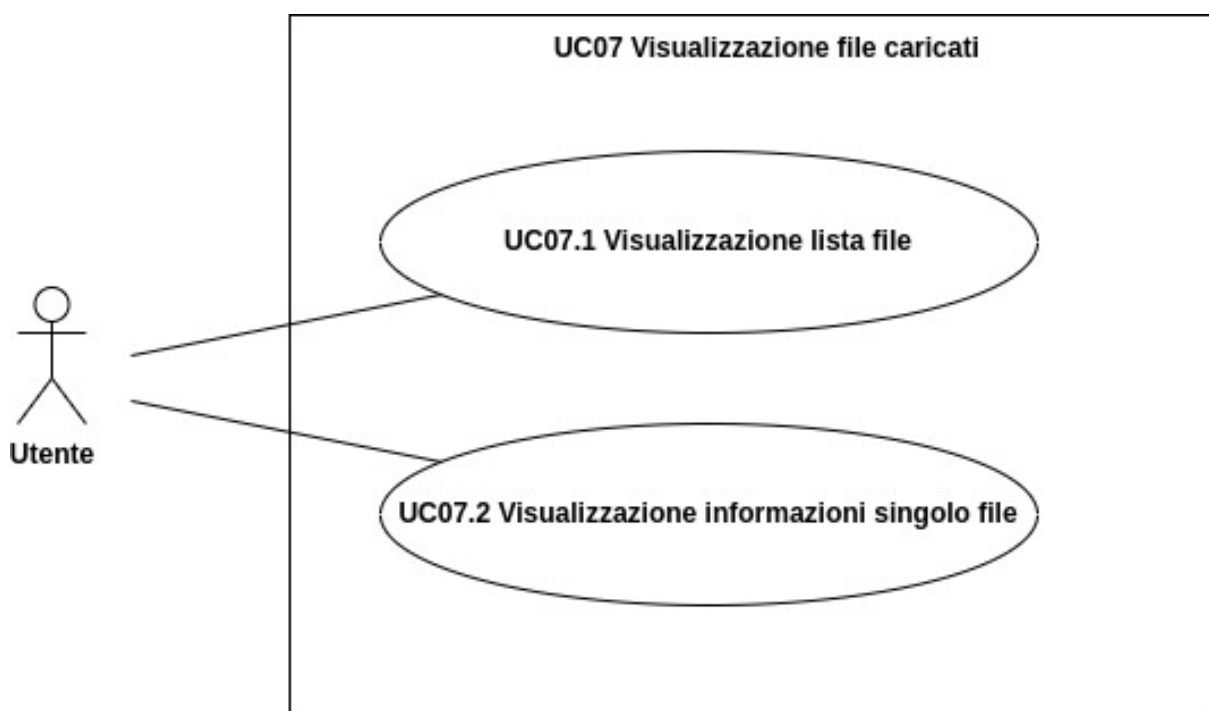
**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione

**Post-condizioni:** L'utente inserisce un nuovo prompt per il modello

**Scenario principale:**

- 1 L'utente inserisce un nuovo prompt

#### 2.1.7 UC07: Visualizzazione file caricati



**Figura 2.7:** Diagramma UML dello use case UC07 (Visualizzazione file caricati)

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di *knowledge*

**Post-condizioni:** L'utente visualizza informazioni sui file caricati

**Scenario principale:**

1 (UC07.1) L'utente visualizza la lista dei file caricati

2 (UC07.2) L'utente può visualizzare le informazioni relative a un singolo file

### 2.1.7.1 UC07.1: Visualizzazione lista file

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza la lista dei file caricati

**Scenario principale:**

- 1 L'utente visualizza la lista dei file caricati

### 2.1.7.2 UC07.2: Visualizzazione informazioni singolo file

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza le informazioni relative ad un file selezionato

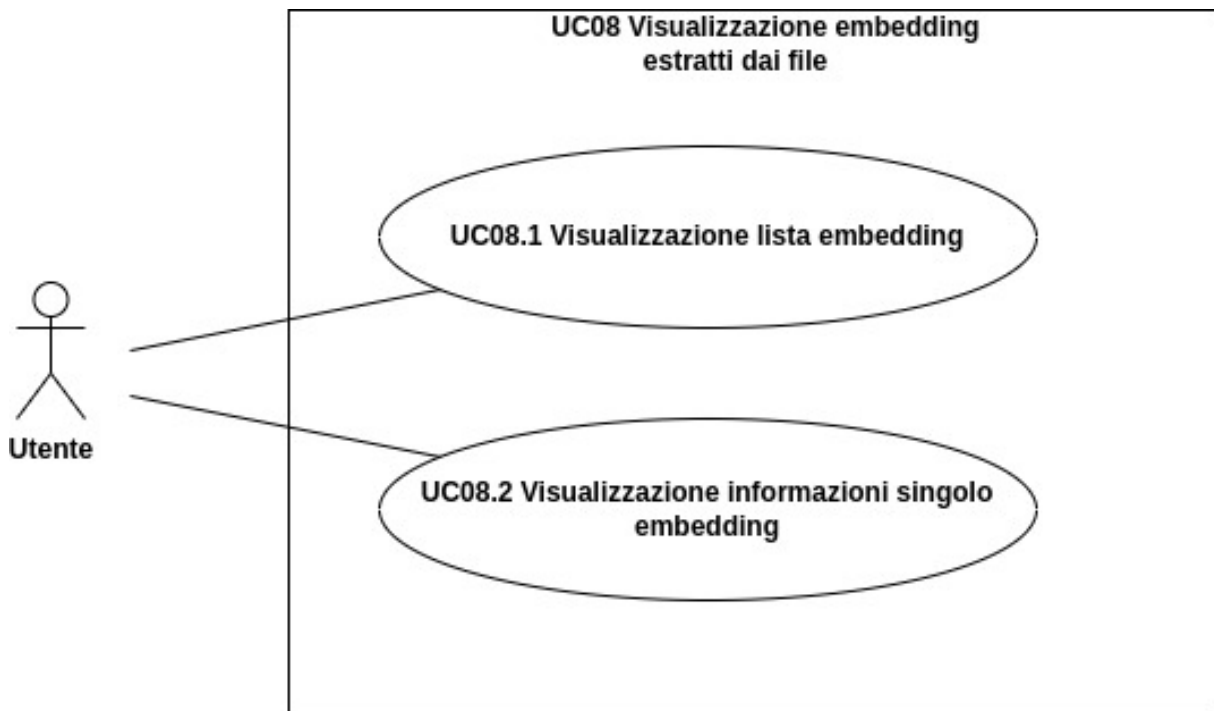
**Scenario principale:**

1 (UC07.2.1) L'utente visualizza il nome del file

1 (UC07.2.2) L'utente visualizza il tipo del file

1 (UC07.2.3) L'utente visualizza il numero di *embedding<sub>G</sub>* associati al file

## 2.1.8 UC08: Visualizzazione embedding estratti dai file



**Figura 2.8:** Diagramma UML dello use case UC08 (Visualizzazione embedding estratti dai file)

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione File, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza informazioni sugli *embedding<sub>G</sub>*

**Scenario principale:**

- 1 (UC07.1) L'utente visualizza la lista degli *embedding<sub>G</sub>*
- 2 (UC07.2) L'utente può visualizzare le informazioni relative a un singolo *embedding<sub>G</sub>*

### 2.1.8.1 UC08.1: Visualizzazione lista embedding

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione File, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza la lista degli *embedding<sub>G</sub>* caricati

### Scenario principale:

- 1 L'utente visualizza la lista degli *embedding<sub>G</sub>* caricati

### 2.1.8.2 UC08.2: Visualizzazione informazioni singolo embedding

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza le informazioni relative ad un *embedding<sub>G</sub>* selezionato

### Scenario principale:

- 1 (UC08.2.1) L'utente visualizza l'id dell'*embedding<sub>G</sub>*
- 1 (UC08.2.2) L'utente visualizza il file di origine dell'*embedding<sub>G</sub>*
- 1 (UC08.2.3) L'utente visualizza il contenuto testuale relativo all'*embedding<sub>G</sub>*

### 2.1.9 UC09: Ricerca embedding

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

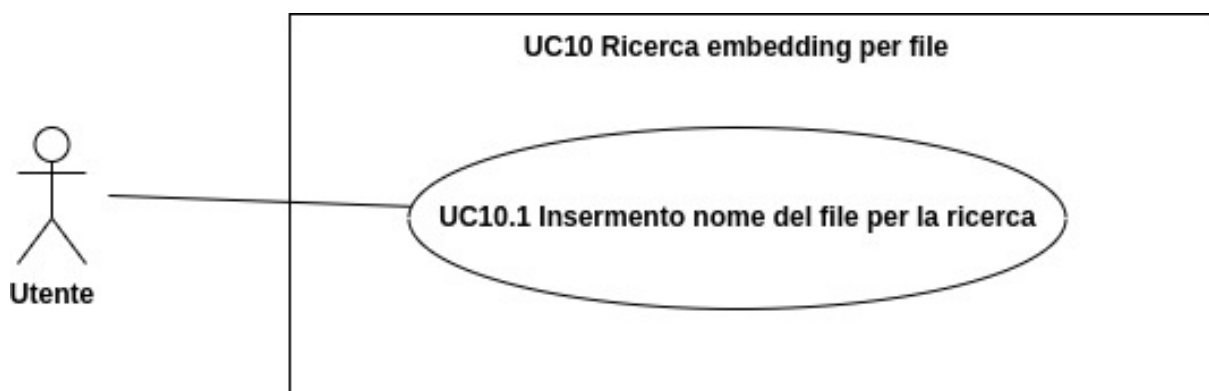
**Post-condizioni:** L'utente visualizza i risultati della ricerca sugli *embedding<sub>G</sub>*

### Scenario principale:

- 1 L'utente effettua una ricerca sugli *embedding<sub>G</sub>*
- 2 (UC07) L'utente visualizza gli *embedding<sub>G</sub>*

**Include:** UC7

## 2.1.10 UC10: Ricerca embedding per file



**Figura 2.9:** Diagramma UML dello use case UC10 (Ricerca embedding per file)

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza i risultati della ricerca sugli *embedding<sub>G</sub>*

**Scenario principale:**

- 1 (UC10.1) L'utente inserisce il nome del file
- 2 (UC07) L'utente visualizza gli *embedding<sub>G</sub>* relativi al filename inserito

**Eredita da:** UC09

### 2.1.10.1 UC10.1: Inserimento nome del file per la ricerca

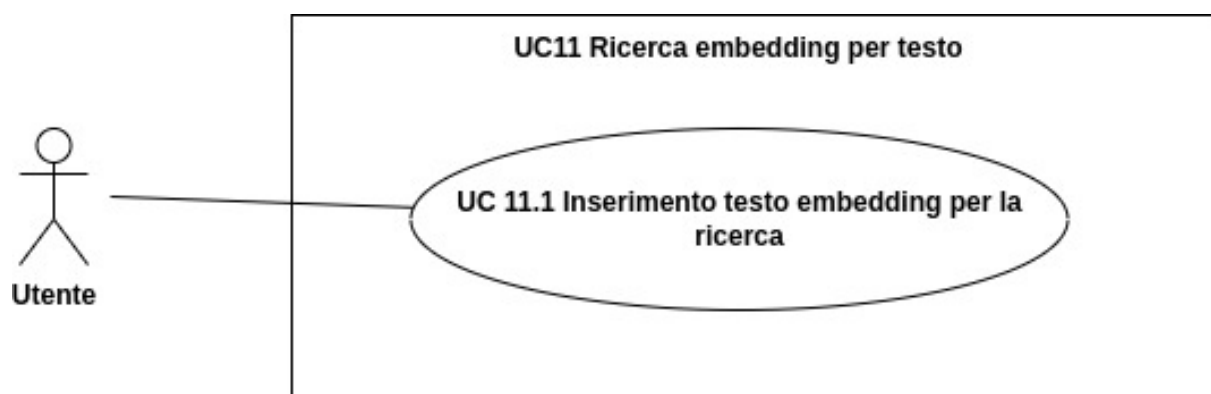
**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** Il nome nel file è inserito nel sistema per la ricerca

**Scenario principale:**

- 1 L'utente inserisce il nome del file



**Figura 2.10:** Diagramma UML dello use case UC11 (Ricerca embedding per testo)

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'utente visualizza i risultati della ricerca sugli *embedding<sub>G</sub>*

**Scenario principale:**

- 1 (UC10.1) L'utente inserisce il contenuto testuale da ricercare negli *embedding<sub>G</sub>*
- 2 (UC07) L'utente visualizza gli *embedding<sub>G</sub>* che soddisfano la ricerca

**Eredita da:** UC09

### 2.1.10.2 UC11.1: Inserimento testo embedding per la ricerca

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** Il testo è inserito nel sistema per la ricerca

**Scenario principale:**

- 1 L'utente inserisce il testo

### 2.1.11 UC12: Eliminazione embedding

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'*embedding<sub>G</sub>* selezionato viene eliminato dal sistema

**Scenario principale:**

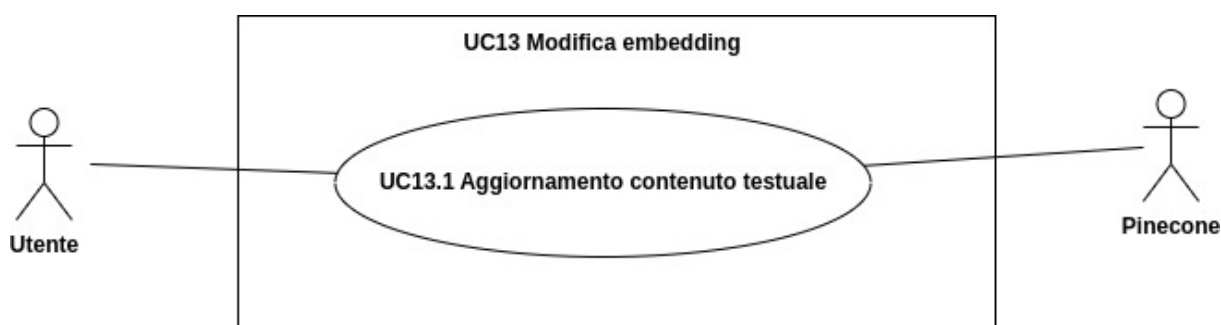
- 1 L'utente seleziona l'*embedding<sub>G</sub>* da eliminare e preme il bottone per la rimozione
- 2 L'*embedding<sub>G</sub>* viene eliminato dal sistema

**Scenari alternativi:**

– Scenario alternativo 1

- 1 L'utente seleziona l'*embedding<sub>G</sub>* da eliminare e preme il bottone per la rimozione
- 2.a (UC17) Viene visualizzato un messaggio di errore legato alla procedura di eliminazione

### 2.1.12 UC13: Modifica embedding



**Figura 2.11:** Diagramma UML dello use case UC13 (Modifica embedding)

**Attore principale:** Utente

**Attore secondario:** Pinecone

**Precondizioni:** Il sistema è operativo, Pinecone è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*

**Post-condizioni:** L'*embedding<sub>G</sub>* selezionato viene aggiornato

**Scenario principale:**

- 1 L'utente seleziona l'*embedding<sub>G</sub>* da aggiornare e preme il bottone per la modifica
- 2 (UC13.1) L'utente inserisce le nuove informazioni sull'*embedding<sub>G</sub>*
- 3 Il sistema aggiorna l'*embedding<sub>G</sub>*

**Scenari alternativi:**

- Scenario alternativo 1
  - 1 .. 2 Come nello scenario principale
- 3.a (UC18) Viene visualizzato un messaggio di errore legato alla procedura di modifica

### 2.1.12.1 UC13.1: Aggiornamento contenuto testuale

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, sottosezione di *knowledge<sub>G</sub>*, l'utente ha selezionato l'*embedding<sub>G</sub>* da aggiornare

**Post-condizioni:** Viene inserito nel sistema il nuovo contenuto testuale

**Scenario principale:**

- 1 L'utente apporta le modifiche volute al contenuto testuale associato all'*embedding<sub>G</sub>*

### 2.1.13 UC14: Visualizzazione errore upload nello storage

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, si è verificato un errore relativo all'upload nello storage

**Post-condizioni:** Il sistema visualizza sul display un messaggio di errore

**Scenario principale:**

- 1 Il sistema stampa sul display un messaggio di errore che viene visualizzato dall'utente

### 2.1.14 UC15: Visualizzazione errore upload nel db vettoriale

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di upload, si è verificato un errore relativo all'upload nel database vettoriale

**Post-condizioni:** Il sistema visualizza sul display un messaggio di errore

**Scenario principale:**

- 1 Il sistema stampa sul display un messaggio di errore che viene visualizzato dall'utente

### 2.1.15 UC16: Visualizzazione errore configurazione

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione di configurazione, si è verificato un errore relativo all'aggiornamento della configurazione del sistema

**Post-condizioni:** Il sistema visualizza sul display un messaggio di errore

**Scenario principale:**

- 1 Il sistema stampa sul display un messaggio di errore che viene visualizzato dall'utente

### 2.1.16 UC17: Visualizzazione errore eliminazione embedding

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, si è verificato un errore relativo all'eliminazione di un' *embedding<sub>G</sub>*

**Post-condizioni:** Il sistema visualizza sul display un messaggio di errore

**Scenario principale:**

- 1 Il sistema stampa sul display un messaggio di errore che viene visualizzato dall'utente

### 2.1.17 UC18: Visualizzazione errore modifica embedding

**Attore principale:** Utente

**Precondizioni:** Il sistema è operativo, l'utente si trova nella sezione Vettori, si è verificato un errore relativo alla modifica di un' *embedding*<sub>G</sub>

**Post-condizioni:** Il sistema visualizza sul display un messaggio di errore

**Scenario principale:**

- 1 Il sistema stampa sul display un messaggio di errore che viene visualizzato dall'utente

## 2.2 Requisiti

Gli *use case*<sub>G</sub> descritti in precedenza sono stati usati come punto di partenza per individuare i requisiti del sistema, che sono di seguito elencati.

Ogni requisito ha un codice così strutturato:

$$\mathbf{R}[\mathbf{W}][\mathbf{T}]-[\mathbf{XYZ}] \quad (2.1)$$

dove:

**R:** identifica il codice come requisito;

**W:** indica la priorità, M per obbligatorio, D per desiderabile e O per opzionale;

**T:** indica la tipologia del requisito, F per funzionale e V per requisito di vincolo;

**XYZ:** numero per identificare il requisito;

### 2.2.1 Requisiti funzionali

Nella tabella 2.1 sono elencati i requisiti funzionali relativi all'upload della *knowledge*<sub>G</sub>. In sintesi, l'utente deve poter caricare, attraverso un form, diversi tipi di file e specificare relativi metadata. Inoltre, per i file PDF, deve essere presente un processo di estrazione che estrae testo e immagini, in modo che l'utente attraverso un secondo form possa selezionare quelle volute. Il sistema memorizza i contenuti caricati in uno storage,

nel caso del progetto uno storage locale, ne estrae gli *embedding<sub>G</sub>* e li carica nel database vettoriale. Inoltre carica informazioni su file ed *embedding<sub>G</sub>* in un database relazionale. In caso di errori, viene visualizzato un messaggio di errore che informa l'utente del malfunzionamento.

Requisito	Descrizione	Fonte
RMF-001	L'utente deve poter caricare un file txt nello storage	UC02, UC01.1
RMF-002	L'utente deve poter indicare la fonte del file txt	UC02, UC01.2
RMF-003	L'utente deve poter caricare gli <i>embedding<sub>G</sub></i> relativi al file txt nel database vettoriale	UC02, UC01.3
RMF-004	Il sistema deve caricare nel database relazionale le informazioni relative al file txt caricato nello storage e ai suoi <i>embedding<sub>G</sub></i> memorizzati nel database vettoriale	UC02, UC01.3
RMF-005	L'utente deve poter caricare un file PDF nello storage	UC03, UC01.1
RMF-006	L'utente deve poter indicare la fonte del file PDF	UC03, UC01.2
RMF-007	L'utente deve poter selezionare le parti di testo del PDF da cui verranno estratti gli <i>embedding<sub>G</sub></i> da caricare nel database vettoriale	UC03, UC03.1
RMF-008	L'utente deve poter modificare il contenuto delle parti di testo del PDF da cui verranno estratti gli <i>embedding<sub>G</sub></i> da caricare nel database vettoriale	UC03, UC03.2
RMF-009	L'utente deve poter selezionare le immagini del PDF da caricare nello storage	UC03, UC03.3
RMF-010	L'utente deve poter inserire una descrizione per le immagini estratte dal PDF	UC03, UC03.4
RMF-011	L'utente deve poter caricare gli <i>embedding<sub>G</sub></i> relativi ai frammenti di testo e alle immagini estratte nel database vettoriale	UC03, UC01.3
Continua nella prossima pagina...		

Tabella 2.1 – Continuo della tabella

Requisito	Descrizione	Fonte
RMF-012	Il sistema deve caricare nel database relazionale le informazioni relative al file PDF caricato nello storage e ai suoi <i>embedding<sub>G</sub></i> memorizzati nel database vettoriale	UC03, UC01.3
RMF-013	L'utente deve poter caricare un file immagine nello storage	UC04, UC01.1
RMF-014	L'utente deve poter indicare la fonte del file immagine	UC04, UC01.2
RMF-015	L'utente deve poter inserire una descrizione testuale per l'immagine da inserire	UC04, UC04.1
RMF-016	L'utente deve poter caricare gli <i>embedding<sub>G</sub></i> relativi al file immagine nel database vettoriale	UC04, UC01.3
RMF-017	Il sistema deve caricare nel database relazionale le informazioni relative al file immagine caricato nello storage e ai suoi <i>embedding<sub>G</sub></i> memorizzati nel database vettoriale	UC04, UC01.3
RMF-018	L'utente deve poter visualizzare un messaggio di errore in caso di errori nel caricamento di un file nello storage	UC14
RMF-019	L'utente deve poter visualizzare un messaggio di errore in caso di errori nel caricamento degli <i>embedding<sub>G</sub></i> nel database vettoriale	UC15

Tabella 2.1: Tabella del tracciamento dei requisiti funzionali, sezione di upload

Nella tabella 2.2 sono elencati i requisiti funzionali relativi alla chat. In sintesi, l'utente deve poter porre domande al chatbot, e deve riceverne risposte coerenti, che possono contenere anche immagini.

Requisito	Descrizione	Fonte
RMF-020	L'utente deve poter conversare con un chatbot capace di fornire informazioni sulla <i>knowledge<sub>G</sub></i> caricata	UC05
RMF-021	L'utente deve poter inserire un messaggio nel chatbot capace di fornire informazioni sulla <i>knowledge<sub>G</sub></i> caricata	UC05.1
RMF-022	L'utente deve poter visualizzare la risposta del chatbot, capace di fornire informazioni sulla <i>knowledge<sub>G</sub></i> caricata, riguardo alla richiesta effettuata	UC05.2

**Tabella 2.2:** Tabella del tracciamento dei requisiti funzionali, sezione di chat

Nella tabella 2.3 sono elencati i requisiti funzionali relativi alla configurazione del sistema. In sintesi, l'utente deve poter modificare il *LLM<sub>G</sub>* usato, il modello utilizzato per generare gli *embedding<sub>G</sub>*, la temperatura (fattore che indica la creatività nelle risposte del chatbot) e il prompt utilizzato. In caso di errori, deve essere visualizzato un messaggio che riporti il malfunzionamento.

Requisito	Descrizione	Fonte
RMF-023	L'utente deve poter aggiornare la configurazione del sistema	UC06
RMF-024	L'utente deve poter aggiornare il <i>LLM<sub>G</sub></i> usato	UC06.1
RMF-025	L'utente deve poter aggiornare il modello utilizzato per generare gli <i>embedding<sub>G</sub></i>	UC06.2
RMF-026	L'utente deve poter aggiornare la temperatura usata	UC06.3
RMF-027	L'utente deve poter aggiornare il prompt usato	UC06.4
Continua nella prossima pagina...		

Tabella 2.3 – Continuo della tabella

Requisito	Descrizione	Fonte
RMF-028	L'utente deve poter visualizzare un messaggio di errore in caso di configurazione fallita	UC16

Tabella 2.3: Tabella del tracciamento dei requisiti funzionali, sezione di configurazione

Nella tabella 2.4 sono elencati i requisiti funzionali relativi alla gestione della *knowledge<sub>G</sub>*. In sintesi, l'utente deve poter visualizzare le informazioni relative ai file e agli *embedding<sub>G</sub>* caricati nel sistema. Non visualizza il contenuto degli *embedding<sub>G</sub>*, ma solamente metadata, poiché si tratta di vettori numerici di dimensione elevata, a cui normalmente un utente non è interessato. Oltre alla possibilità di visualizzare le informazioni, l'utente deve poter effettuare una ricerca, per file o per contenuto testuale, sugli *embedding<sub>G</sub>*, modificare il loro contenuto ed eliminarli.

Requisito	Descrizione	Fonte
RMF-029	L'utente deve poter visualizzare le informazioni sui file caricati nello storage	UC07
RMF-030	L'utente deve poter visualizzare la lista dei file caricati nello storage	UC07.1
RMF-031	L'utente deve poter visualizzare le informazioni sui singoli file caricati nello storage	UC07.2
RMF-032	L'utente deve poter visualizzare il nome del file caricato nello storage	UC07.2.1
RMF-033	L'utente deve poter visualizzare il tipo del file caricato nello storage	UC07.2.2
RMF-034	L'utente deve poter visualizzare il numero di <i>embedding<sub>G</sub></i> caricati nel database vettoriale associati ad un dato file caricato nello storage	UC07.2.3
Continua nella prossima pagina...		

Tabella 2.4 – Continuo della tabella

Requisito	Descrizione	Fonte
RMF-035	L'utente deve poter visualizzare le informazioni sugli <i>embedding<sub>G</sub></i> caricati nel database vettoriale	UC08
RMF-036	L'utente deve poter visualizzare la lista degli <i>embedding<sub>G</sub></i> caricati nel database vettoriale	UC08.1
RMF-037	L'utente deve poter visualizzare le informazioni sui singoli <i>embedding<sub>G</sub></i> caricati nel database vettoriale	UC08.2
RMF-038	L'utente deve poter visualizzare l'id dell' <i>embedding<sub>G</sub></i> caricato nel database vettoriale	UC08.2.1
RMF-039	L'utente deve poter visualizzare il file di origine relativo all' <i>embedding<sub>G</sub></i> caricato nel database vettoriale	UC08.2.2
RMF-040	L'utente deve poter visualizzare il contenuto testuale relativo all' <i>embedding<sub>G</sub></i> caricato nel database vettoriale	UC08.2.3
RMF-041	L'utente deve poter effettuare una ricerca fra gli <i>embedding<sub>G</sub></i>	UC09
RMF-042	L'utente deve poter effettuare una ricerca fra gli <i>embedding<sub>G</sub></i> per file	UC10
RMF-043	L'utente deve poter inserire il nome del file relativo alla ricerca degli <i>embedding<sub>G</sub></i> per file	UC10.1
RMF-044	L'utente deve poter effettuare una ricerca fra gli <i>embedding<sub>G</sub></i> per testo	UC11
RMF-045	L'utente deve poter inserire il testo relativo alla ricerca degli <i>embedding<sub>G</sub></i> per testo	UC11.1
RDF-046	L'utente deve poter eliminare un <i>embedding<sub>G</sub></i>	UC12
RDF-047	L'utente deve poter aggiornare un <i>embedding<sub>G</sub></i>	UC13
Continua nella prossima pagina...		

Tabella 2.4 – Continuo della tabella

Requisito	Descrizione	Fonte
RDF-048	L'utente deve poter aggiornare il contenuto testuale relativo all' <i>embedding<sub>G</sub></i>	UC13.1
RDF-049	L'utente deve poter visualizzare un messaggio di errore in caso di problemi con l'eliminazione di un <i>embedding<sub>G</sub></i>	UC17
RDF-050	L'utente deve poter visualizzare un messaggio di errore in caso di problemi con l'aggiornamento di un <i>embedding<sub>G</sub></i>	UC18

**Tabella 2.4:** Tabella del tracciamento dei requisiti funzionali, sezione di knowledge

### 2.2.2 Requisiti non funzionali

Nella tabella 2.5 sono elencati i requisiti non funzionali, in particolare i requisiti di vincolo. È richiesto che il database vettoriale sia *serverless*, ovvero che il server e l'infrastruttura sottostante siano interamente gestiti dal provider, liberando così lo sviluppatore da attività operative e di manutenzione. L'interfaccia grafica dell'applicazione deve essere sviluppata in Remix, *framework<sub>G</sub>* usato dall'azienda ospitante lo stage, mentre il fetch di file e *embedding<sub>G</sub>*, compresa la ricerca per file, nella sezione di *knowledge<sub>G</sub>* dell'applicazione, deve coinvolgere solamente il database relazionale per ragioni prestazionali, su richiesta interna dell'azienda.

Requisito	Descrizione	Fonte
RMV-001	Il database vettoriale deve essere <i>serverless</i>	Interno
RMV-002	Fetch di file e <i>embedding<sub>G</sub></i> , ricerca degli <i>embedding<sub>G</sub></i> per file devono avvenire coinvolgendo solamente il database relazionale, per ragioni di efficienza prestazionale	Interno
RMV-003	La UI dell'applicazione deve essere realizzata con il <i>framework<sub>G</sub></i> Remix	Interno

**Tabella 2.5:** Tabella del tracciamento dei requisiti di vincolo

# Capitolo 3

## Progettazione

Questo capitolo illustra il periodo di progettazione, andando a illustrare le tecnologie, l'architettura adottata e le diverse componenti architetture, utilizzando sia diagrammi tipici dell'ingegneria del software, come i diagrammi *UML<sub>G</sub>*, che un linguaggio meno formale e tecnico per un lettore meno esperto.

### 3.1 Tecnologie e strumenti

Di seguito viene data una panoramica delle tecnologie e strumenti utilizzati, sono omesse la *RAG<sub>G</sub>* e LangChain poiché sono state discusse in 1.2.

#### 3.1.1 Pinecone

Pinecone è un database vettoriale *serverless* utilizzato per costruire applicazioni nell'ambito dell'intelligenza artificiale. Le ottime *API<sub>G</sub>* offerte e la sua natura cloud lo hanno reso perfetto per le esigenze del progetto. All'interno del progetto, viene utilizzato per memorizzare gli *embedding<sub>G</sub>* relativi ai file caricati e facenti parte della *knowledge<sub>G</sub>*.

#### 3.1.2 MariaDB

MariaDB è un *RDBMS<sub>G</sub>* nato da una *fork* di MySQL. All'interno del progetto viene utilizzato per memorizzare informazioni su file e *embedding<sub>G</sub>* in modo da ottimizzare operazioni di ricerca, che sarebbero state più lente se si fosse solamente utilizzato un database vettoriale. Tale *RDBMS<sub>G</sub>* è stato proposto dall'azienda.

### 3.1.3 LangServe

Langserve è uno strumento utilizzato per facilitare il deployment di applicazioni Python basate su LangChain. Utilizza diverse tecnologie chiave per fornire una soluzione robusta e scalabile:

- **LangChain**: illustrato in [1.2](#);
- **FastAPI**: Come riportato in [\[8\]](#), «FastAPI è un web framework moderno e veloce (a prestazioni elevate) che serve a creare API con Python 3.6+ basato sulle annotazioni di tipo di Python.»;
- **Pydantic**: Una libreria Python per la validazione dei dati e la gestione dei tipi, utilizzata da FastAPI per garantire che i dati in entrata e in uscita siano correttamente tipizzati e validati;
- **Docker**: Una piattaforma per la containerizzazione delle applicazioni, utilizzata per distribuire Langserve in un ambiente isolato e portabile;
- **Uvicorn**: Un server ASGI utilizzato da FastAPI per gestire le richieste HTTP in modo efficiente;
- **Poetry**: Un gestore di dipendenze e uno strumento di packaging per Python, che consente di dichiarare, gestire e installare le dipendenze dei progetti in modo semplice e coerente.

LangServe è stato scelto principalmente perché integrava in un unico *framework<sub>G</sub>* servizi come FastAPI, Docker, Uvicorn e Poetry; le sue funzionalità legate a LangChain, che consistono nella definizione specifica di route che eseguono le *chain<sub>G</sub>*, non sono state utilizzate poiché ci si è preferito affidare al metodo di definizione delle route di FastAPI, ritenuto più completo e personalizzabile. All'interno del progetto viene usato per semplificare la creazione e deployment dell'*API<sub>G</sub>* creata.

### 3.1.4 PyMuPDF

PyMuPDF è una libreria Python per l'estrazione, l'analisi, la conversione e la manipolazione di documenti PDF. All'interno del progetto viene usata per estrarre testo e

immagini dai documenti PDF; è stata scelta rispetto alle altre librerie disponibili per la sua ottima capacità nel lavorare sia con il testo che con le immagini, come riportato dallo studio descritto in [1].

### 3.1.5 PyTest

PyTest è un *framework*<sub>G</sub> per il testing di applicazioni Python, è stato utilizzato per testare alcune classi su cui si basa l'*API*<sub>G</sub>.

### 3.1.6 Remix

Remix è un *framework*<sub>G</sub> per lo sviluppo full-stack di *web app*<sub>G</sub>. È stato utilizzato per realizzare la parte di View e di Controller dell'applicazione. È stata una tecnologia imposta dall'azienda, poiché utilizzata comunemente nei loro progetti e nel sito del brand che dovrà ospitare una versione perfezionata e ultimata dell'applicativo.

#### 3.1.6.1 React

React è una libreria JavaScript per costruire interfacce utente, si basa sui *componenti*, funzioni o classi che rappresentano un pezzo di interfaccia grafica e consentono la gestione del suo stato. È la libreria fondamentale su cui è costruito Remix.

#### 3.1.6.2 Tailwind CSS

Tailwind CSS è un *framework*<sub>G</sub> di CSS che offre una serie di classi utility che possono essere combinate per dare uno stile all'interfaccia grafica. È stato utilizzato poiché supportato di default da Remix.

### 3.1.7 Git e GitHub

Git è un sistema di controllo di versione distribuito progettato per tracciare le modifiche nel codice sorgente durante lo sviluppo di software. Esso consente la gestione di diverse versioni di un progetto e facilita la collaborazione tra sviluppatori, mantenendo un registro dettagliato delle modifiche e agevolando l'integrazione di varianti del codice. GitHub è una piattaforma web che offre servizi di hosting per *repository* Git.

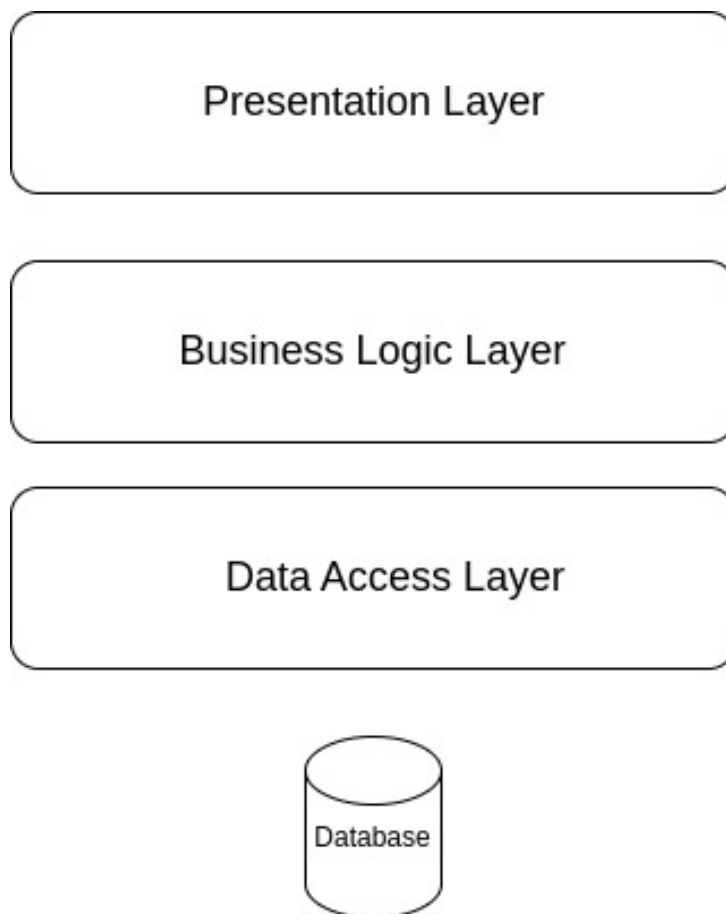
## 3.2 Architettura

L'architettura di implementazione è una Three-Layer Architecture, una tipica architettura a strati come quelle descritte in [12], dove sono presenti tre livelli:

- **Data Access Layer:** strato che si occupa dell'accesso ai dati;
- **Business Logic Layer:** strato che contiene la business logic;
- **Presentation Layer:** strato che si occupa dell'interfaccia grafica e dell'interazione con l'utente.

Ogni strato dipende dallo strato sottostante, mentre non ha dipendenze con quello sopra di esso.

È stata scelta questa architettura per separare il frontend dal backend dell'applicazione, in modo da rendere il tutto più mantenibile e applicare il principio di *separation of concerns*, secondo il quale il backend deve essere completamente indipendente dal frontend. Un'architettura che applica anche questo principio è quella esagonale, simile a quella a strati con la differenza che la business logic è totalmente indipendente dagli altri strati, ma è stato preferito non implementarla per via dell'overhead cognitivo che essa avrebbe introdotto. Nella figura 3.1 è illustrata una classica architettura a tre strati, mentre nella 3.2 l'architettura adattata al progetto.

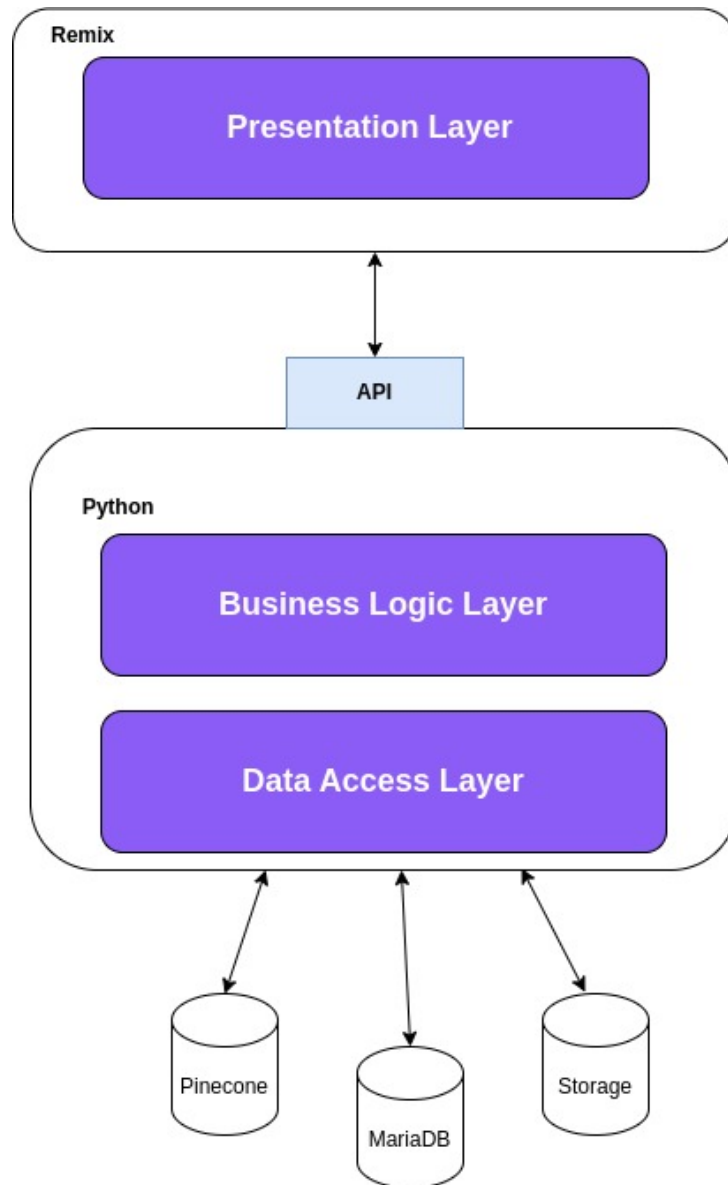


**Figura 3.1:** Diagramma della Three-Layer Architecture

Sono presenti due applicazioni:

- una che si occupa dell'interfaccia grafica e dell'interazione con essa, scritta in JavaScript utilizzando il framework Remix;
- una che si occupa della logica di business e dell'interazione con i dati, scritta in Python.

Le due applicazioni comunicano fra loro grazie a un' *API* realizzata con FastAPI, mentre l'utente accede alla *web app* da browser.



**Figura 3.2:** Diagramma dell'architettura usata nel progetto

L'*API<sub>G</sub>* segue lo standard *REST<sub>G</sub>*, nella figura 3.3 sono illustrati i vari *endpoint<sub>G</sub>* esposti. È stato scelto questo standard per via della sua semplicità, rispetto a tecnologie più complesse come SOAP o GraphQL.

POST	/chat/invoke	Chatbot	▼
POST	/image	Upload Image	▼
POST	/text	Upload Text	▼
POST	/pdf/extraction	Extract Content	▼
POST	/pdf/extraction/load	Upload Batch	▼
GET	/configuration	Get Configuration	▼
PUT	/configuration	Update Configuration	▼
POST	/storage	Store Content	▼
POST	/storage/batch	Store Batch	▼
GET	/storage/{filetype}/{filename}/	Read Content	▼
GET	/files	Get Files	▼
GET	/files/{filename}/vectors	Get Vectors By File	▼
GET	/vectors	Get Vectors	▼
DELETE	/vectors/{vector_id}	Delete Vector	▼
PUT	/vectors/{vector_id}	Update Vector	▼
POST	/vectors/search/content	Search Vectors	▼

**Figura 3.3:** Endpoint esposti dall'API

### 3.2.1 Database vettoriale

In Pinecone è stato creato un indice, che è l'unità organizzativa di livello più alto dei dati vettoriali in Pinecone. Accetta e memorizza vettori, gestisce le query su di essi e svolge altre operazioni vettoriali sui suoi contenuti, come viene descritto in [13].

L'indice è stato configurato con una dimensione di 1536 e una metrica *cosine<sub>G</sub>*, per mantenere compatibilità con i modelli utilizzati per generare gli *embedding<sub>G</sub>*, cioè `text-embedding-ada-002` e `text-embedding-3-small`.

Sono creati *embedding<sub>G</sub>* di tue tipologie:

- **Testo:** rappresentano frammenti di testo, fra i loro metadata contengono il contenuto testuale che ha originato i valori del vettore, il file da cui proviene, la fonte;
- **Immagine:** rappresentano un'immagine, i valori del vettore sono originati a partire dalla descrizione testuale dell'immagine e non dalla foto stessa. Fra i metadata contengono la descrizione testuale, la fonte, il file da cui provengono e un url utilizzato per il loro recupero.

Tutti gli *embedding<sub>G</sub>* hanno un id che consente la loro identificazione. Nella figura 3.4 è mostrata la struttura degli *embedding<sub>G</sub>* in Pinecone.

ID	VALUES
7	efa69a0a-f9... 0.0343404375, -0.0503324196, -0.0208650753, 0.0175705906, 0.0455737188, 0.0143676177, -0.0402201787, 0.0262643714, -0.00938013...
SCORE	-0.0138
METADATA	file: "colnago2010.pdf" source: "Catalogo Colnago 2010" text: "Foto della bicicletta Colnago ACE" type: "image" uri: "http://localhost:8000/storage/img/colnago2010.pdf_image_3.png"
8	5e2917d7-c2... 0.00573667418, 0.000698930235, 0.00884174, -0.0352972485, -0.0102359699, -0.021877788, -0.0220869221, 0.0614855289, 0.0313469...
SCORE	-0.0165
METADATA	file: "colnago.txt" source: "wikipedia" text: "colnago è un'azienda italiana."

Figura 3.4: Esempi di embedding visti dal web client di Pinecone

## 3.2.2 Database relazionale

### 3.2.2.1 Analisi

La presenza di un database relazionale è necessaria per conservare la configurazione del sistema e le diverse informazioni su file e *embedding<sub>G</sub>*.

Perché conservare queste informazioni anche in un database relazionale se sono presenti anche in quello vettoriale? È vero, viene introdotta una ridondanza, ma si migliorano le prestazioni: le informazioni su tutti gli *embedding<sub>G</sub>* sono leggibili da Pinecone tramite algoritmi di ricerca randomica, mentre utilizzando delle tabelle per rappresentare file e *embedding<sub>G</sub>* si può accedere alle informazioni su tutti gli *embedding<sub>G</sub>* in maniera più efficiente. Nella figura 3.5 è presente una comparativa fra i due approcci. Inoltre utilizzando un database relazionale è possibile implementare facilmente la paginazione dei risultati, non ancora supportata da Pinecone. Anche se nel progetto di stage i risultati delle query da database non sono mai paginati, su richiesta dell'azienda, si è preferito adottare l'approccio con database relazionale per ragioni di estensibilità, nel caso in cui la paginazione verrà aggiunta in futuro.

```
Tempo impiegato per fare il fetch di tutti i vettori da Pinecone: 0.9183502197265625 seconds
Tempo impiegato per fare il fetch di tutti i vettori dalla tabella di MariaDB: 0.001718759536743164 seconds
```

Figura 3.5: Comparazione di performance fra MariaDB e Pinecone

### 3.2.2.2 Progettazione concettuale

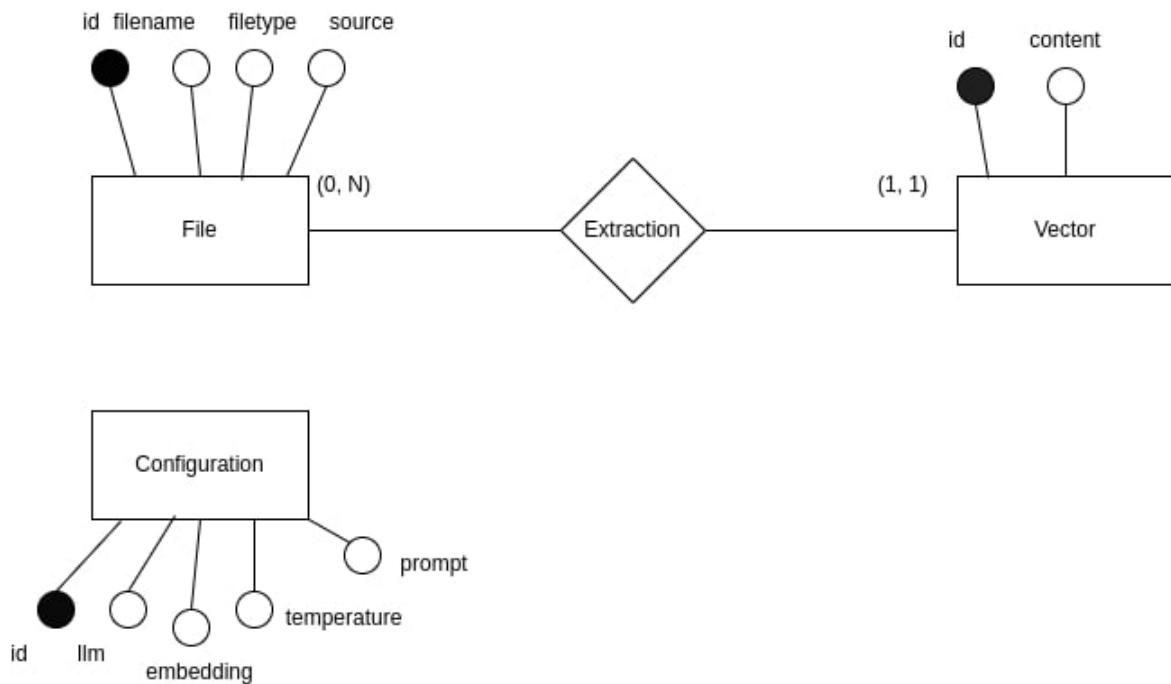


Figura 3.6: Diagramma ER

### 3.2.2.3 Progettazione logica

- `configuration(id, embedding, llm, temperature, prompt)`
- `file(id, filetype, filename, source)`
- `vector(id, file_id*, content)`
  - `file_id` chiave esterna su `file`

### 3.2.3 Data Access Layer

Questo strato contiene le classi che si occupano dell'interazione con il database vettoriale, con il database relazionale e con lo storage. È stata presa in considerazione di utilizzare un *ORM* per gestire l'interazione con MariaDB, ma l'opzione è stata scartata poiché avrebbe portato ad un *over-engineering* del problema. Nella figura 3.7 è presentato il relativo diagramma della classi, mentre di seguito sono elencate le classi individuate:

- **Storage**: classe astratta che fornisce un'interfaccia per leggere e scrivere su uno storage, necessaria per leggere e scrivere file. I file vengono salvati per consentire l'estrazione degli *embedding<sub>G</sub>* e il recupero delle immagini da parte del chatbot quando l'utente chiede di visualizzare delle immagini. Seppur lo storage utilizzato è soltanto uno, si è preferito creare questa astrazione per facilitare un possibile cambio di storage in futuro;
- **LocalStorage**: classe figlia di **Storage**, rappresenta uno storage locale;
- **AwsStorage**: classe figlia di **Storage**, rappresenta uno storage Aws S3.
- **DatabaseAccess**: classe che si occupa di interfacciarsi con il database relazionale, le query da eseguire sono incapsulate nei suoi metodi. Viene usata come approccio alternativo agli *ORM<sub>G</sub>* in Python;
- **SingletonMeta**: classe necessaria per implementare il pattern Singleton nella classe **Database**;
- **VFile**: classe che rappresenta un file;
- **Vector**: classe che rappresenta un *embedding<sub>G</sub>*;
- **PineconeVectorStore**: classe di LangChain, utilizzata per accedere a Pinecone.

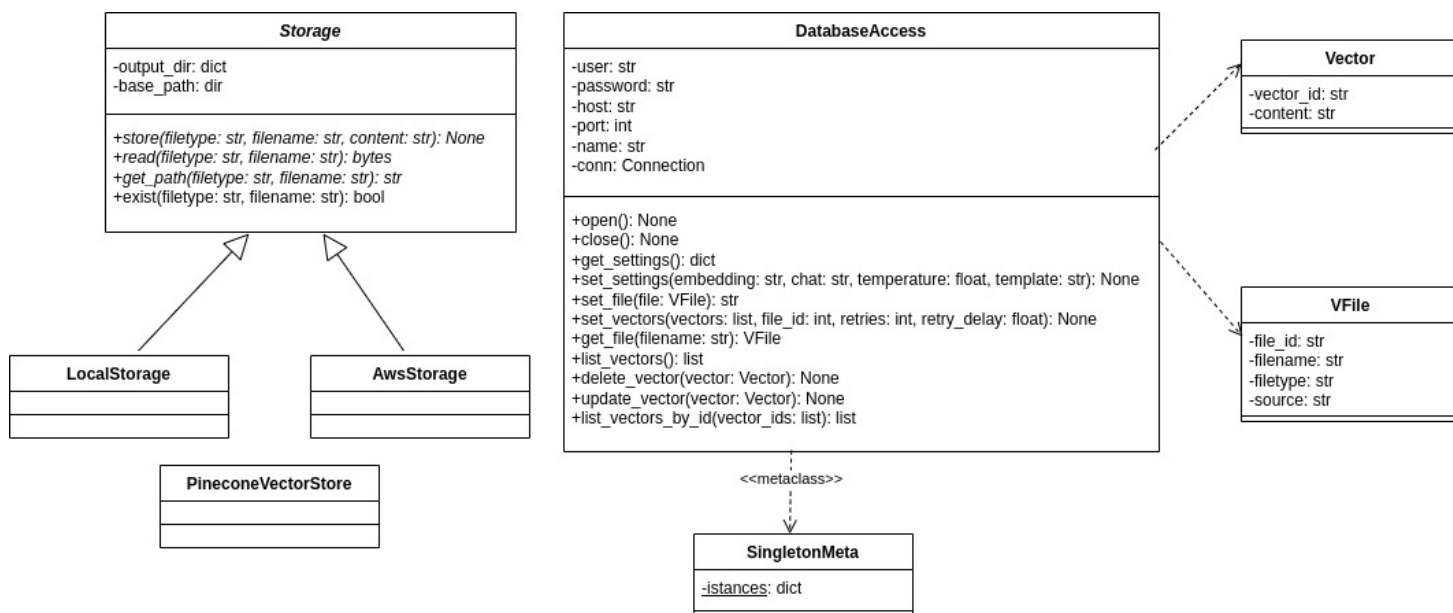
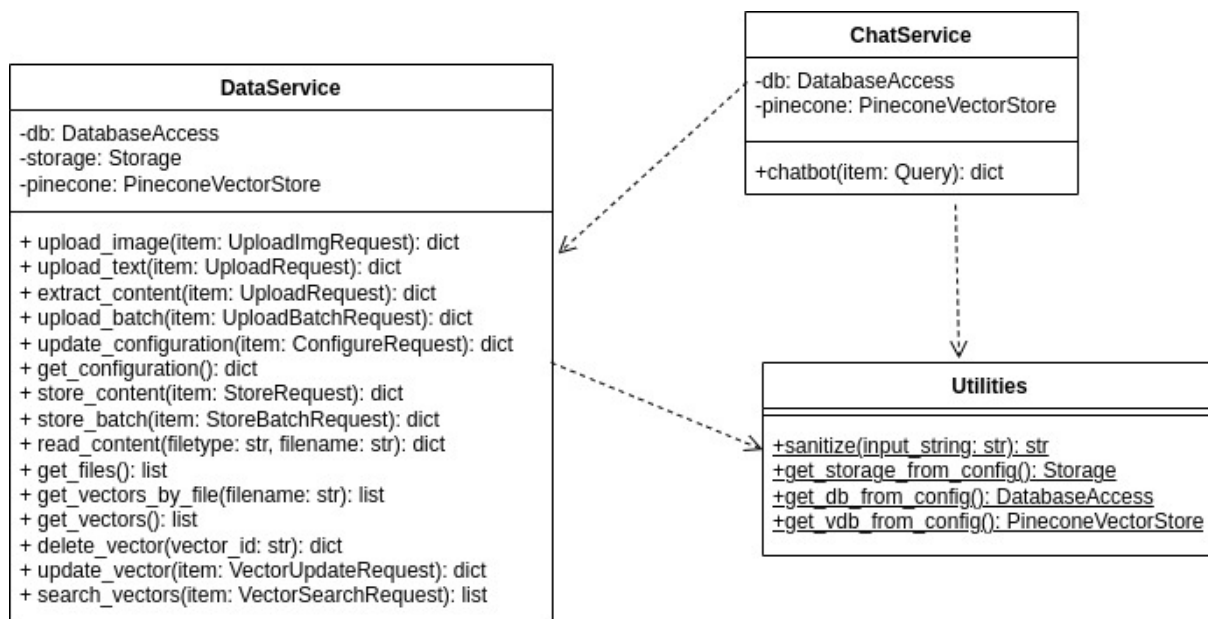


Figura 3.7: Diagramma UML delle classi del data access layer

### 3.2.4 Business Logic Layer

Questo strato contiene le classi che si occupano della business logic, ossia l'insieme delle regole e dei processi che governano il funzionamento e l'elaborazione dei dati all'interno del sistema. Nella figura 3.7 è presentato il relativo diagramma della classi, mentre di seguito sono elencate le classi individuate:

- **DataService**: classe che si occupa della business logic correlata ai dati;
- **ChatService**: classe che si occupa della business logic correlata al chatbot, recupera il contesto utile alla domanda e utilizza un *LLM<sub>G</sub>* per elaborare la risposta finale;
- **Utilities**: classe che fornisce metodi statici di utilità, sostanzialmente per fare un sanitize dei dati, e costruire gli oggetti di persistenza a partire da un file di configurazione.



**Figura 3.8:** Diagramma UML delle classi del business logic layer

#### 3.2.4.1 Schemi di validazione

Per la validazione lato backend è stato utilizzato Pydantic, che permette di costruire degli schemi di validazione in modo da verificare la correttezza dei dati in input e in output. Gli schemi a livello implementativo sono delle classi, dove gli attributi specificano il tipo

dei dati da validare, e i metodi specificano le operazioni di validazione. Di seguito, nelle figure 3.9, 3.10 e 3.11 sono riportati gli schemi utilizzati, la seguente documentazione è stata generata da FastAPI.

```
ConfigureRequest ^ Collapse all object
embedding* string matches ^(text-embedding-3-small|text-embedding-ada-002)$
llm* string matches ^(gpt-4o|gpt-3.5-turbo)$
temperature* number [0.1, 1]
prompt* string

HTTPValidationError ^ Collapse all object
detail ^ Collapse all array-object>
  Items ^ Collapse all object
    loc* > Expand all array<(string | integer)>
    msg* string
    type* string

Query ^ Collapse all object
input* string

ResponseFile ^ Collapse all object
content* string
extension* string

ResponseMessage ^ Collapse all object
message* string

StoreBatchRequest ^ Collapse all object
filename* > Expand all array<any>
content* > Expand all array<any>
```

**Figura 3.9:** Schemi di validazione

```
StoreRequest ^ Collapse all object
filetype* string
filename* string
content* string

UploadBatchRequest ^ Collapse all object
filename* string
filetype* string
text_content* > Expand all array<string>
images_desc* > Expand all array<string>
images_name* > Expand all array<string>
metadata* string

UploadingRequest ^ Collapse all object
filename* string
description* > Expand all string >= 1 characters
source* string
file* string

UploadRequest ^ Collapse all object
filetype* string
filename* string
metadata* string
```

**Figura 3.10:** Schemi di validazione

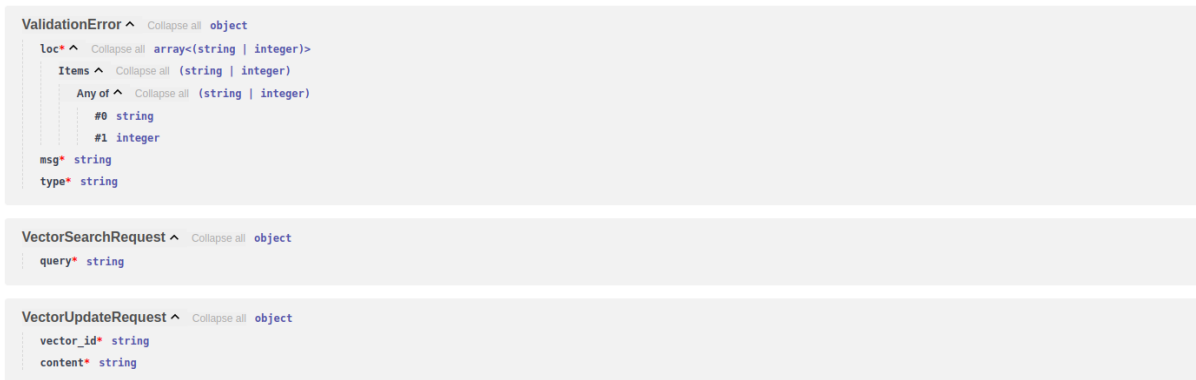


Figura 3.11: Schemi di validazione

### 3.2.5 Presentation Layer

Il presentation layer si occupa di presentare i dati all'utente e permette un'interazione con essi. I componenti sono stati creati utilizzando React, il routing è stato gestito con React Routing, mentre la parte relativa alla comunicazione con lo strato di business logic e alla gestione degli input dell'utente è stata gestita con le action e i loader, *hook*<sub>G</sub> resi disponibili da Remix. Le action sono utilizzate per la gestione dei form, mentre i loader per il caricamento dei dati provenienti dagli strati sottostanti.

#### 3.2.5.1 Componenti

- **App:** Root dell'applicazione, al suo interno vengono renderizzate le route relative all'upload dei file, alla chat, alla configurazione e alla *knowledge*<sub>G</sub>.
  - **Index:** Index della Root principale;
  - **Upload:** Componente che si occupa dell'upload dei file, contiene la sezione relativa all'upload del file nello storage. Per la parte di upload nel database vettoriale, viene renderizzato un componente diverso in base al tipo di file caricato, poiché le informazioni da inserire sono differenti in base all'estensione del file;
    - \* **ImgUpload:** Componente renderizzato nella route relativa all'upload di un'immagine. Contiene un form per inserire la fonte e la descrizione testuale della foto;

- \* **TextUpload:** Componente renderizzato nella route relativa all'upload di un file di testo. Contiene un form per inserire la fonte del file;
- \* **PdfUpload:** Componente renderizzato nella route relativa all'upload di un file PDF. Dal PDF vengono estratti immagini e frammenti di testo, attraverso un form l'utente può decidere quali informazioni caricare, modificare il loro contenuto e aggiungere descrizioni testuali per le immagini. Come nei componenti precedenti, è possibile inserire la fonte del file PDF, interpretata come il titolo del documento.
  - **ChooseForm:** Componente renderizzato all'interno di PdfUpload, fornisce un form che permette all'utente di selezionare le immagini e i frammenti di testo da caricare, inserendo anche i relativi metadati;
  - **Text:** Componente renderizzato all'interno di ChooseForm, rappresenta un frammento di testo estratto dal PDF, modificabile dall'utente;
  - **Image:** Componente renderizzato all'interno di ChooseForm, rappresenta un'immagine estratta dal PDF, e consente all'utente di inserire la relativa descrizione.
- **Chat:** Componente renderizzato nella route relativa alla chat, contiene un form attraverso il quale l'utente può inserire le sue domande, e una sezione dove viene visualizzata la risposta elaborata dal sistema, contenente sia testo che immagini;
  - \* **Gallery:** Componente renderizzato all'interno della chat, utilizzato per rappresentare l'insieme delle immagini nella risposta del chatbot.
- **Settings:** Componente renderizzato nella route relativa alla configurazione del sistema, contiene un form attraverso il quale l'utente può selezionare il modello utilizzato per generare gli *embedding<sub>G</sub>*, il *LLM<sub>G</sub>* utilizzato, la temperatura e inserire il prompt di base;
- **Context:** Componente renderizzato nella route relativa alla *knowledge<sub>G</sub>*, contiene due sottoroute: una utilizzata per i file e l'altra per gli *embedding<sub>G</sub>*.

- \* **Files:** Componente renderizzato nella route relativa alla gestione dei file della *knowledge<sub>G</sub>*. Permette di visualizzare le informazioni su ogni file caricato;
  - **VFile:** Componente renderizzato all'interno di Files, rappresenta un file caricato dall'utente, contiene informazioni riguardo il nome del file, il tipo e il numero di vettori associati.
- \* **Vectors:** Componente renderizzato nella route relativa alla gestione degli *embedding<sub>G</sub>* della *knowledge<sub>G</sub>*. Permette di visualizzare le informazioni su ogni *embedding<sub>G</sub>*.
  - **Vector:** Componente renderizzato all'interno di Vectors, rappresenta un *embedding<sub>G</sub>*, riporta informazioni riguardo il suo id, il file da cui è stato estratto e il suo contenuto testuale. Inoltre contiene form per la modifica delle informazioni dell'*embedding<sub>G</sub>* e per la sua rimozione dal sistema.

### 3.2.6 Design Pattern

I design pattern sono delle soluzioni sistematiche a problemi ricorrenti nel design del software. Sono stati utilizzati i design pattern Singleton e Facade, illustrati in [6], *RAII<sub>G</sub>*, discusso in [10], e Dependency Injection, descritto in [5].

#### 3.2.6.1 Singleton

Il Singleton è un design pattern creazionale che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a quell'istanza. È usato normalmente per le classi che gestiscono accesso al database. È stato utilizzato nell'implementazione della classe `DatabaseAccess`. Nella figura 3.12 è mostrato il classico diagramma di questo design pattern.

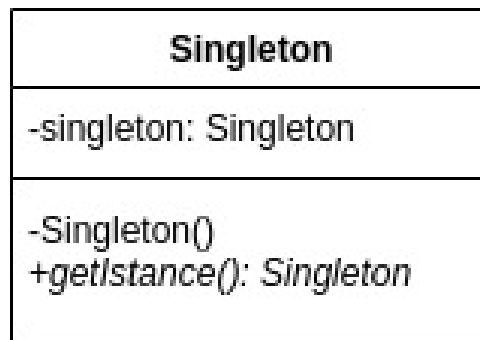


Figura 3.12: Diagramma UML del pattern Singleton

### 3.2.6.2 Facade

La Facade è un design pattern strutturale utilizzato per fornire un'interfaccia semplificata a un complesso insieme di classi. Viene implementato dalle classi `DataService` e `ChatService`, che forniscono un'interfaccia comoda che raggruppa diverse chiamate ai metodi di alcune classi del sistema. Ad esempio, per il caricamento di un file esso deve essere salvato sullo storage, sul database relazionale e su quello vettoriale, e le classi sopracitate presentano dei metodi che si occupano di raggruppare le varie interazioni con questi tre sistemi di memoria. Nella figura 3.13 è mostrato il classico diagramma di questo design pattern.

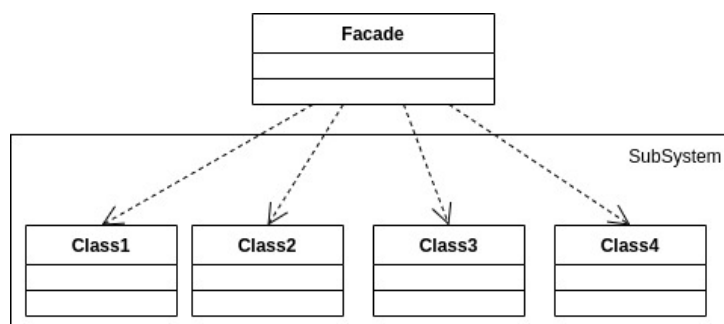


Figura 3.13: Diagramma UML del pattern Facade

### 3.2.6.3 RAII

*RAII* è un design pattern di gestione delle risorse, che specifica che la classe che rappresenta l'accesso ad una risorsa, debba acquisire quest'ultima nel suo costruttore e debba rilasciarla nel suo distruttore. Viene utilizzato dalla classe `DatabaseAccess`, per

gestire l'accesso al database relazionale. Tale pattern permette un accesso alle risorse sicuro e comodo per lo sviluppatore, che non deve preoccuparsi di acquisire o rilasciare la risorsa manualmente.

#### 3.2.6.4 Dependency Injection

Il pattern Dependency Injection è un design pattern strutturale. Seguendo questo pattern la creazione delle dipendenze di un oggetto non deve essere eseguita dall'oggetto esterno, ma deve essere relegata ad un'altra entità. Viene utilizzato per gestire le dipendenze delle classi `DataService` e `ChatService`: queste ultime hanno come attributi oggetti di tipo `DatabaseAccess`, `Storage` e `PineconeVectorStore`, ma questi non vengono istanziati nel costruttore delle classi `Service` ma vengono create dalla classe `Utilities`, e iniettate attraverso il passaggio di argomenti al costruttore. Riduce l'accoppiamento fra classi e ne facilita l'estensibilità.

#### 3.2.6.5 MVC

Il MVC (Model-View-Controller) è un pattern architetturale utilizzato solitamente nel presentation layer. Questo pattern presenta tre unità:

- **Modello:** unità che fornisce una rappresentazione dei dati, spesso tramite strutture dati specifiche;
- **Vista:** unità che rappresenta la parte grafica, si occupa di presentare a schermo i dati rappresentati dal modello;
- **Controller:** unità che si occupa di catturare gli input provenienti dall'utente e di conseguenza aggiornare il modello.

Ha molte varianti e interpretazioni, nel progetto viene usato nel presentation layer da Remix, dove i dati provenienti dagli strati sottostanti e lo stato dei componenti rappresentano il modello, i vari pezzi di HTML generati da React le viste, e i vari `hook_G` (`action` e `loader`) i controller. Nella figura 3.14, tratta da [14], ne è mostrato un tipico diagramma.

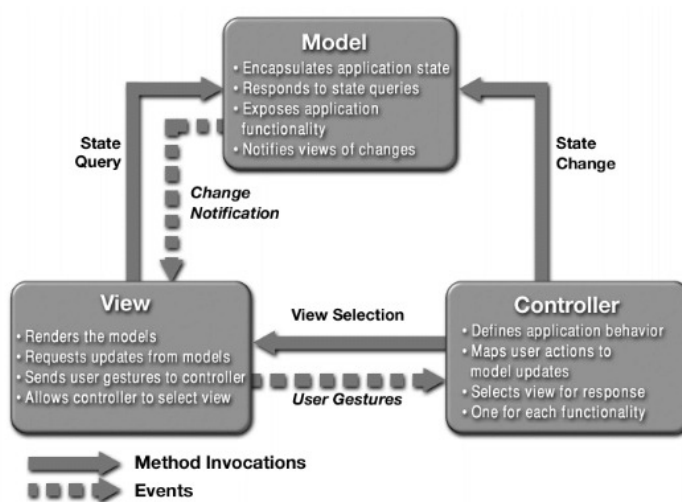


Figura 3.14: Diagramma del MVC

# Capitolo 4

## Realizzazione e testing

In questo capitolo vengono mostrati alcuni esempi di codice del software prodotto, vengono discusse le tecniche di testing adottate e viene fornita una panoramica finale sul prodotto.

### 4.1 Codifica

#### 4.1.1 Backend

Di seguito sono mostrati alcuni esempi di codice:

- Il listato [4.1](#) mostra parte dell'implementazione della classe `DatabaseAccess`, mostrando come vengono gestite le query e come viene implementato il pattern *RAII<sub>G</sub>*, gestendo la risorsa database nel costruttore e nel distruttore;
- Il listato [4.2](#) mostra parte dell'implementazione della classe `ChatService`, che implementa la business logic legata all'elaborazione delle risposte a partire dalle domande dell'utente. È possibile vedere l'implementazione del pattern Facade;
- Il listato [4.3](#) mostra parte del codice della classe `Utilities`, mostrando i metodi utilizzati per realizzare la Dependency Injection, permettendo di iniettare oggetti creati a partire da informazioni presenti su file;
- Il listato [4.4](#) mostra parte del codice dell'*API<sub>G</sub>*, mostrando come definire una route in FastAPI.

```
class DatabaseAccess(metaclass=SingletonMeta):
    def __init__(self, user: str, password: str, host: str, port: int, name: str) ->
    ↪ None:
        self._user = user
        self._password = password
        self._host = host
        self._port = port
        self._name = name
        self._conn = mariadb.connect(
            user=self._user, password=self._password, host=self._host,
            port=int(self._port), database=self._name
        )

    def get_file(self, filename: str) -> VFile:
        cur = self._conn.cursor()
        cur.execute("SELECT vector.id, vector.content FROM vector INNER JOIN file ON
    ↪ vector.file_id = file.id WHERE file.filename = ?",
            (filename, ))
        vectors = []
        res = cur.fetchall()
        for row in res:
            vectors.append(Vector(row[0], row[1]))

        cur.execute("SELECT filetype, source, id FROM file WHERE filename = ?",
            (filename, ))
        row = cur.fetchone()
        filetype = row[0]
        source = row[1]
        file_id = row[2]
        self._conn.commit()
        return VFile(filetype, filename, source, file_id, vectors)

#codice omissis

    def __del__(self) -> None:
        if self._conn:
            self._conn.close()
```

Codice 4.1: Parte del codice della classe DatabaseAccess

```
class ChatService:
    def __init__(self, db: DatabaseAccess, pinecone: PineconeVectorStore)
    ↪ -> None:
        self._db = db
        self._pinecone = pinecone

    def chatbot(self, item: Query) -> dict:
        settings = self._db.get_settings()
        chat = ChatOpenAI(model=settings["llm"],
    ↪ temperature=settings["temperature"])
        template = settings["prompt"] + ChatService.og_template

        parser = JsonOutputParser(pydantic_object=Answer)
        retriever = self._pinecone.as_retriever(k=4)
        prompt = PromptTemplate(
            template=template,
            input_variables=["context", "question"],
        )

        setup_and_retrieval = RunnableParallel(
            {"context": retriever, "question": RunnablePassthrough()}
        )

        chain = setup_and_retrieval | prompt | chat | parser
        answer = chain.invoke(Utilities.sanitize(item.input))
        data = DataService(self._db, Utilities.get_storage_from_config(),
    ↪ Utilities.get_vdb_from_config())
        for index, url in enumerate(answer["urls"]):
            answer["urls"][index] = data.read_content(url.split("/")[-2],
    ↪ url.split("/")[-1])
        answer["srcs"] = answer.pop("urls")
        return answer
```

```
class Utilities:
    @staticmethod
    def get_db_from_config() -> DatabaseAccess:
        db = DatabaseAccess(user=os.getenv("DB_USER"),
                             password=os.getenv("DB_PASSWORD"),
                             host=os.getenv("DB_HOST"),
                             port=os.getenv("DB_PORT"),
                             name=os.getenv("DB_NAME"))

        return db
```

**Codice 4.3:** Parte del codice della classe `Utilities`

```
@app.post("/chat/invoke", status_code=200)
async def chatbot(item: Query):
    chat = ChatService(Utilities.get_db_from_config(),
        → Utilities.get_vdb_from_config())
    return chat.chatbot(item)

@app.post("/image", status_code=201, response_model=ResponseMessage)
async def upload_image(item: UploadImgRequest):
    data = DataService(Utilities.get_db_from_config(),
        → Utilities.get_storage_from_config(),
        → Utilities.get_vdb_from_config())
    return data.upload_image(item)

@app.post("/text", status_code=201, response_model=ResponseMessage)
async def upload_text(item: UploadRequest):
    data = DataService(Utilities.get_db_from_config(),
        → Utilities.get_storage_from_config(),
        → Utilities.get_vdb_from_config())
    return data.upload_text(item)
```

**Codice 4.4:** Parte del codice dell'API

## 4.1.2 Frontend

Di seguito sono mostrati alcuni esempi di codice:

- Il listato [4.5](#) mostra il codice del componente `Chat`, da cui si può vedere la struttura base di un componente React, l'utilizzo di Tailwind per dare uno stile grafico e l'uso di *hook* `G`;
- Il listato [4.7](#) mostra un esempio di action Remix, che fa da controller nel presentation layer.

```
export default function Chat() {  
  const actionData = useActionData();  
  const navigation = useNavigation();  
  const isSubmitting = navigation.state === "submitting";  
  return (  
    <div className="flex flex-col items-center gap-4">  
      <Form method="POST" className="text-gray-700 bg-white  
        ↪ shadow-md rounded px-8 pt-6 pb-8 mb-4 w-6/12">  
        <legend className="block font-bold mb-2 py-4  
          ↪ text-lg">Cosa vuoi sapere?</legend>  
          <label className="text-indent block  
            ↪ text-sm font-bold mb-2  
            ↪ htmlFor="question">  
            Cosa vuoi sapere?  
          </label>  
        <div className="flex gap-4">  
          <input  
            className=" block w-full px-3 py-2 border  
            ↪ rounded-lg focus:outline-none  
            ↪ focus:ring-violet-500  
            ↪ focus:border-violet-500"  
            id="question"  
            type="text"  
            name="question"  
            placeholder="Inserisci una richiesta..."  
          />  
        </div>  
      </Form>  
    </div>  
  );  
}
```

**Codice 4.5:** Codice del componente Chat

```
        <button
            className="bg-violet-500 text-white font-bold py-2 px-4
                ↪ rounded-lg hover:bg-violet-600 focus:outline-none
                ↪ focus:shadow-outline"
            type="submit"
            disabled={isSubmitting}
            >{isSubmitting ? "Invio..." : "Invia" }</button>
        </div>
    </Form>
    <div className="mx-auto max-w-screen-md bg-white shadow-md rounded
        ↪ px-8 pt-6 pb-8 mb-4 w-6/12">
    <p className="mb-4">
        {actionData ? actionData.text : ""}
    </p>
    <Gallery srcs={actionData?.srcs ?? []} />
    </div>
</div>
);
}
```

Codice 4.6: Codice del componente Chat

```
export async function action({ request }) {
  const host = process.env.HOST;
  const body = await request.formData();
  const endpoint = host + "/chat/invoke/"
  const data = {
    "input": body.get("question"),
  }
  let response;
  try {
    response = await fetch(endpoint, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(data)
    });
  } catch (error) {
    return json({ error: "Errore nel fetch" });
  }
  if (response.status !== 200) {
    return json({ error: "Errore nel server" });
  } else
    return await response.json();
}
```

Codice 4.7: Esempio di action Remix

## 4.2 Testing

In questa sezione vengono illustrate le tecniche di testing automatico e manuale adottate.

Il testing automatico consente di eseguire dei test tramite un'applicazione, permettendo

di sostituire test manuali e ottenere una migliore affidabilità. Consiste nello scrivere una classe di test per ogni classe implementata, che presenta dei metodi che vanno a confrontare i risultati prodotti dalla classe da testare con quelli aspettati. Nel progetto i test automatici sono solamente utilizzati in parte, poiché scrivere test per ogni classe presente avrebbe portato via una notevole quantità di tempo, che si è preferito invece dedicare all'esplorazione tecnologica e alla progettazione del software.

Le classi di test sono realizzate tramite PyTest, *framework*<sub>G</sub> di testing molto utilizzato in Python. Di seguito, in figura 4.8, è riportato parte del codice della classe `TestLocalStorage`, utilizzata per il testing automatico della classe `LocalStorage`.

```
class TestLocalStorage:
    def setup_method(self):
        self.storage = LocalStorage(os.getenv("BASE"))

    def test_read(self):
        s1 = self.storage.read("txt", "sample.txt")
        s2 = self.storage.read("pdf", "sample.pdf")
        s3 = self.storage.read("img", "sample.png")
        assert s1 == TestLocalStorage.expected["txt"]
        assert s2 == TestLocalStorage.expected["pdf"]
        assert s3 == TestLocalStorage.expected["img"]
```

**Codice 4.8:** Esempio di classe di testing

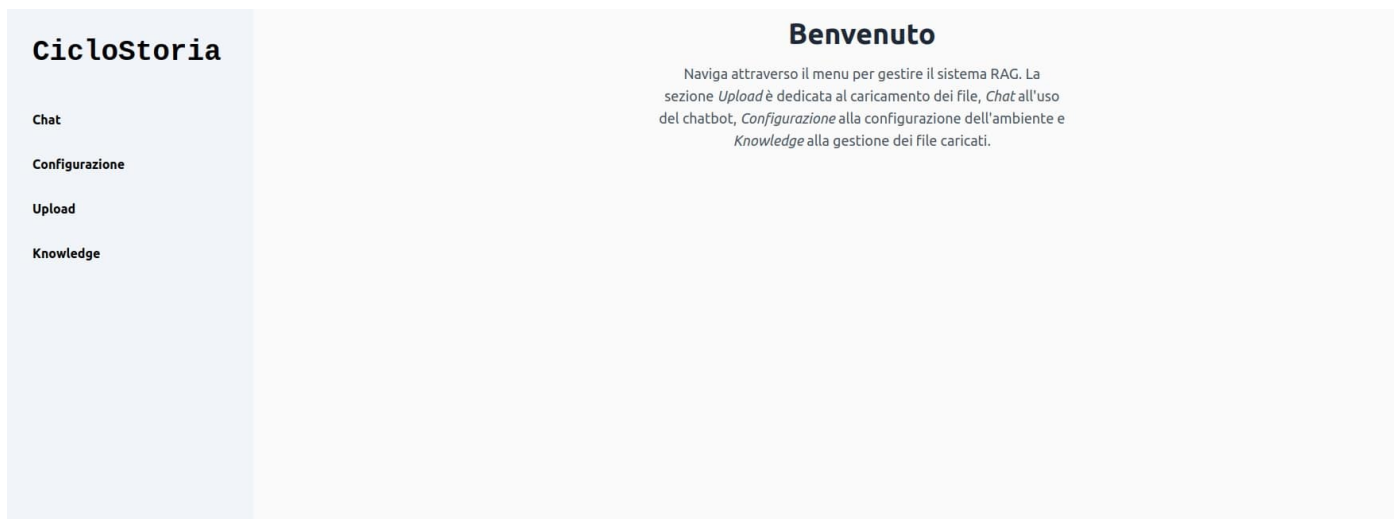
Le classi del backend e il codice del frontend non testato tramite testing automatico è stato testato manualmente, conducendo diverse prove e confrontando i risultati ottenuti con quelli attesi.

## 4.3 Prodotto finale

In questa sezione si offre una panoramica completa sul prodotto finale.

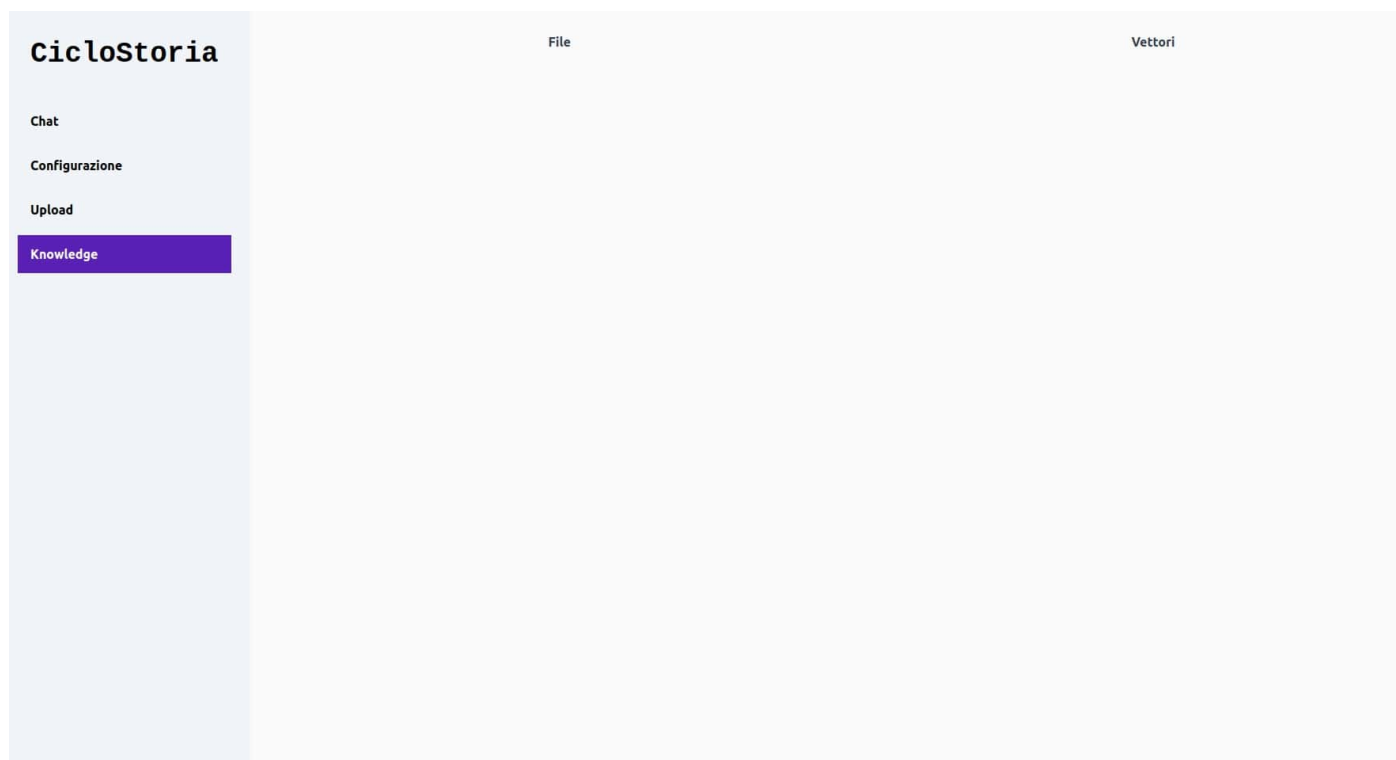
### 4.3.1 Home

Appena entrati nell'applicazione viene visualizzata la schermata di home, come mostrato nella figura 4.1.



**Figura 4.1:** Schermata di home

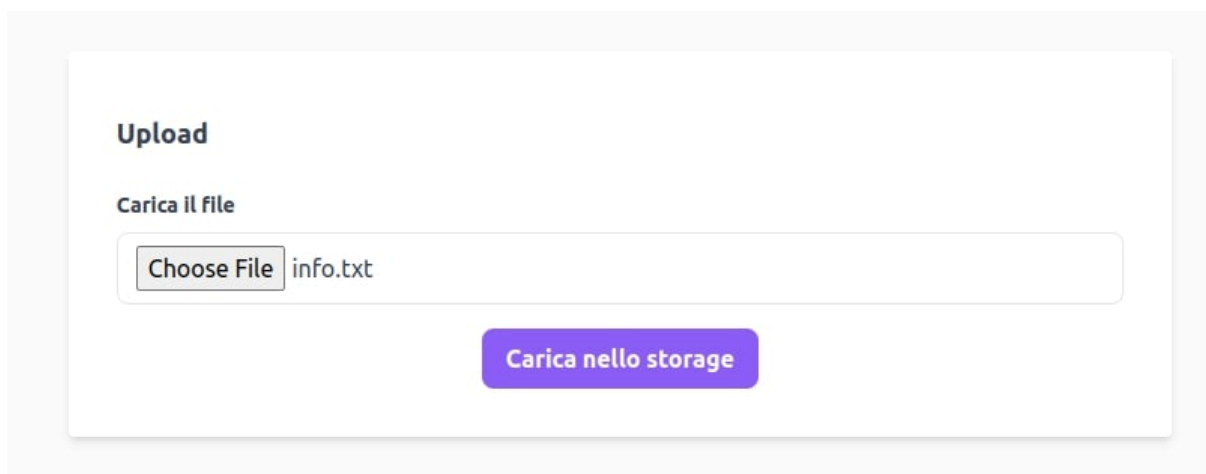
A sinistra è presente un menu che permette la navigazione dentro l'applicazione, e rimane fisso sullo schermo mentre sulla destra vengono renderizzate le rispettive sezioni selezionate. Nella figura 4.2 ne è mostrato un esempio, dove è stata selezionata la sezione di *Knowledge*.



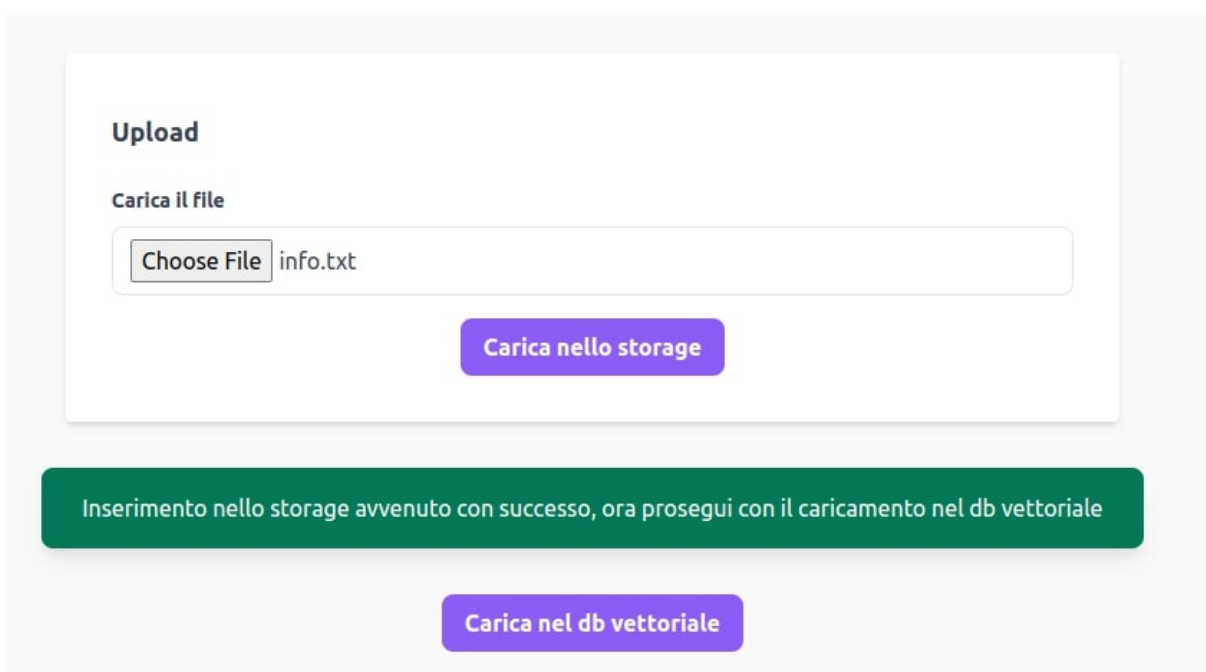
**Figura 4.2:** Menu e sezione renderizzata

### 4.3.2 Upload

La sezione *Upload* permette il caricamento di diversi file nel sistema. Appena si entra viene chiesto di selezionare il file da caricare, che inizialmente verrà memorizzato solamente nello storage, come mostrato nella figura 4.3; una volta completato il caricamento, viene visualizzata una notifica di successo, come mostrato nella figura 4.4, e successivamente vengono chieste altre informazioni da inserire, in base al tipo di file, e viene completato il caricamento anche sul database relazionale e vettoriale.



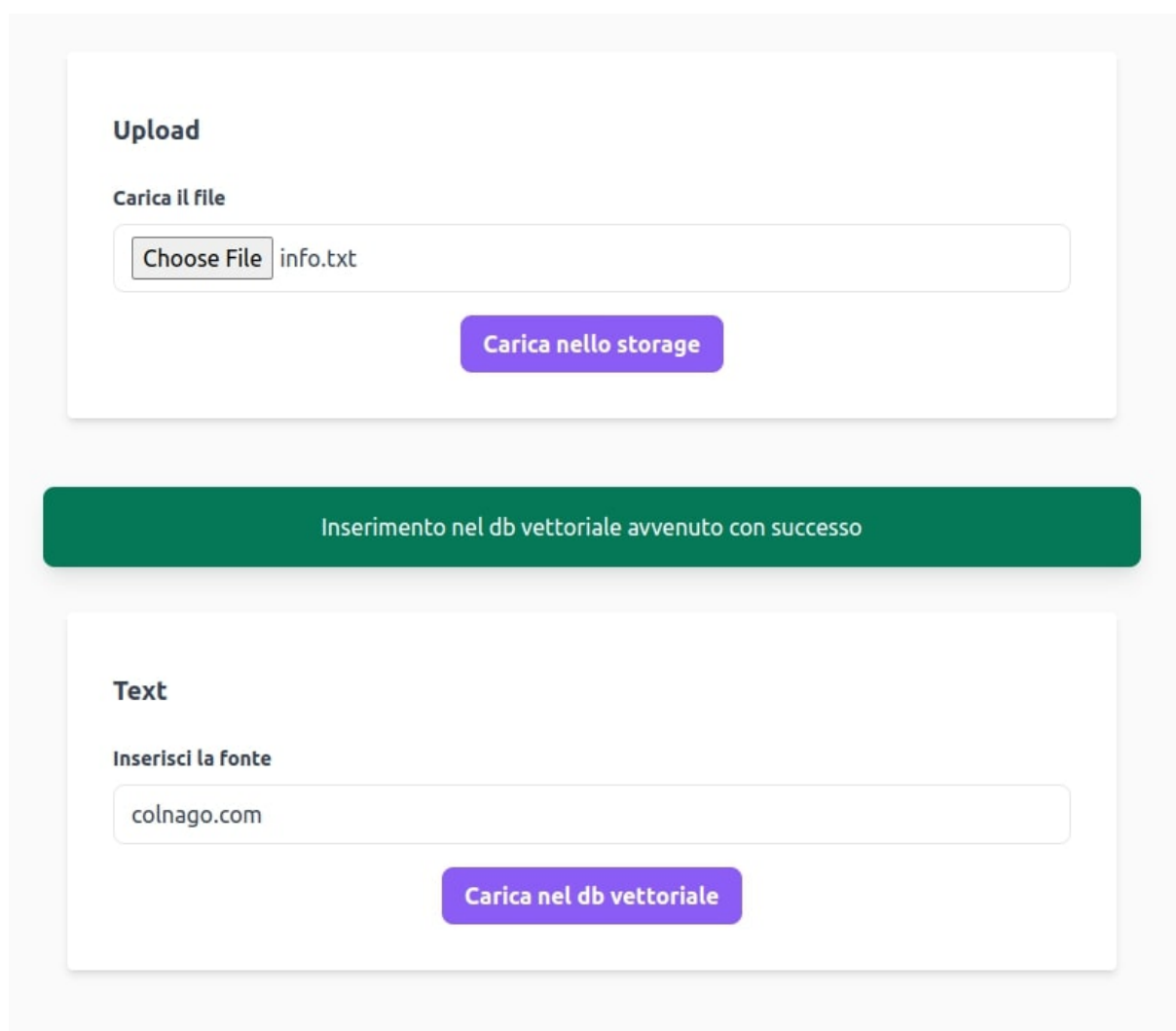
**Figura 4.3:** Schermata di caricamento nello storage



**Figura 4.4:** Schermata di successo di caricamento nello storage

#### 4.3.2.1 Upload file testo

Se il file selezionato è un file di testo, viene chiesta come informazione aggiuntiva solamente la sua fonte, come mostrato nella figura 4.5.



**Figura 4.5:** Schermata di upload di file di testo completato

#### 4.3.2.2 Upload file immagine

Se il file selezionato è un file immagine (jpg, png, gif), viene chiesta come informazione aggiuntiva la sua fonte e una sua descrizione testuale, come mostrato nella figura 4.6.

The screenshot shows a web interface for uploading an image. It is divided into two main sections:

- Upload:** Contains a heading "Upload", a sub-heading "Carica il file", a file selection input field with a "Choose File" button and the filename "g4x.jpg", and a purple button labeled "Carica nello storage".
- Immagine:** Contains a heading "Immagine", a sub-heading "Inserisci una descrizione dell'immagine", a text area with the description "Foto della bicicletta da gravel Colnago G4-X.", a sub-heading "Inserisci la fonte", an input field with the source "colnago.com", and a purple button labeled "Carica nel db vettoriale".

**Figura 4.6:** Schermata di upload di file immagine

### 4.3.2.3 Upload file PDF

Se il file selezionato è un file PDF, vengono estratti da esso frammenti di testo e immagini, che l'utente può selezionare. L'utente può inoltre modificare i frammenti di testo e inserire descrizioni testuali per le immagini.

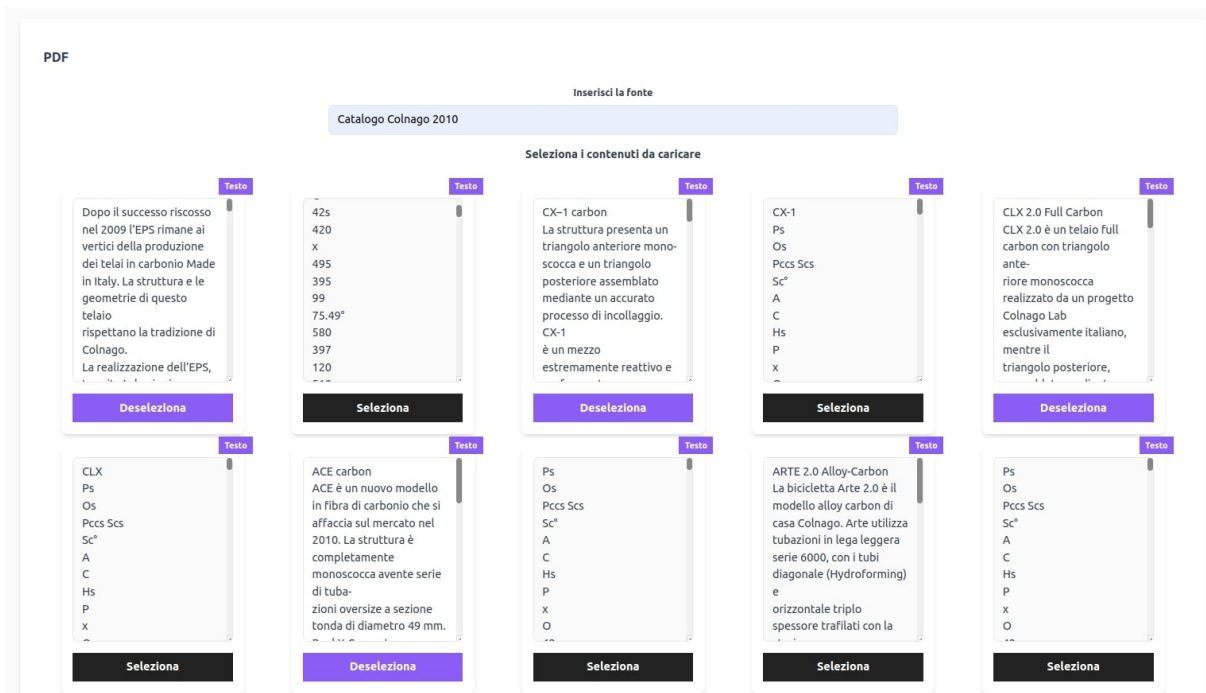


Figura 4.7: Schermata di upload di file PDF, selezione testo

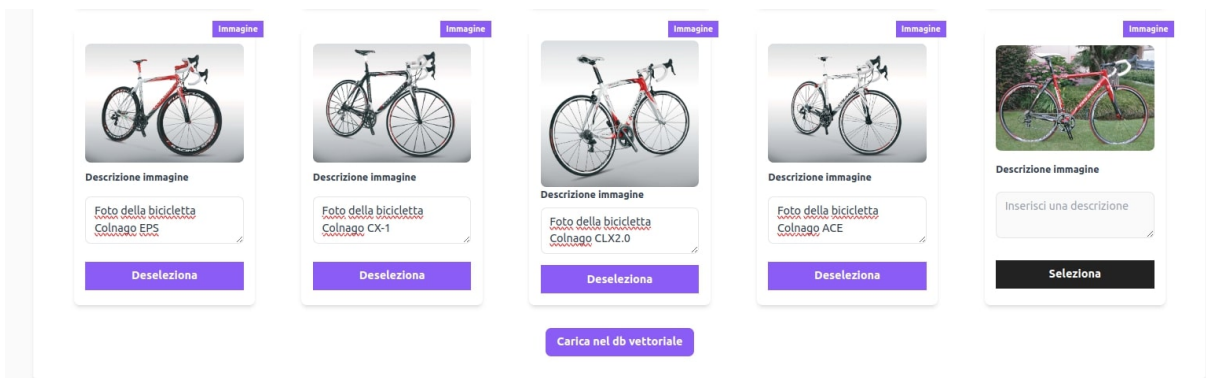
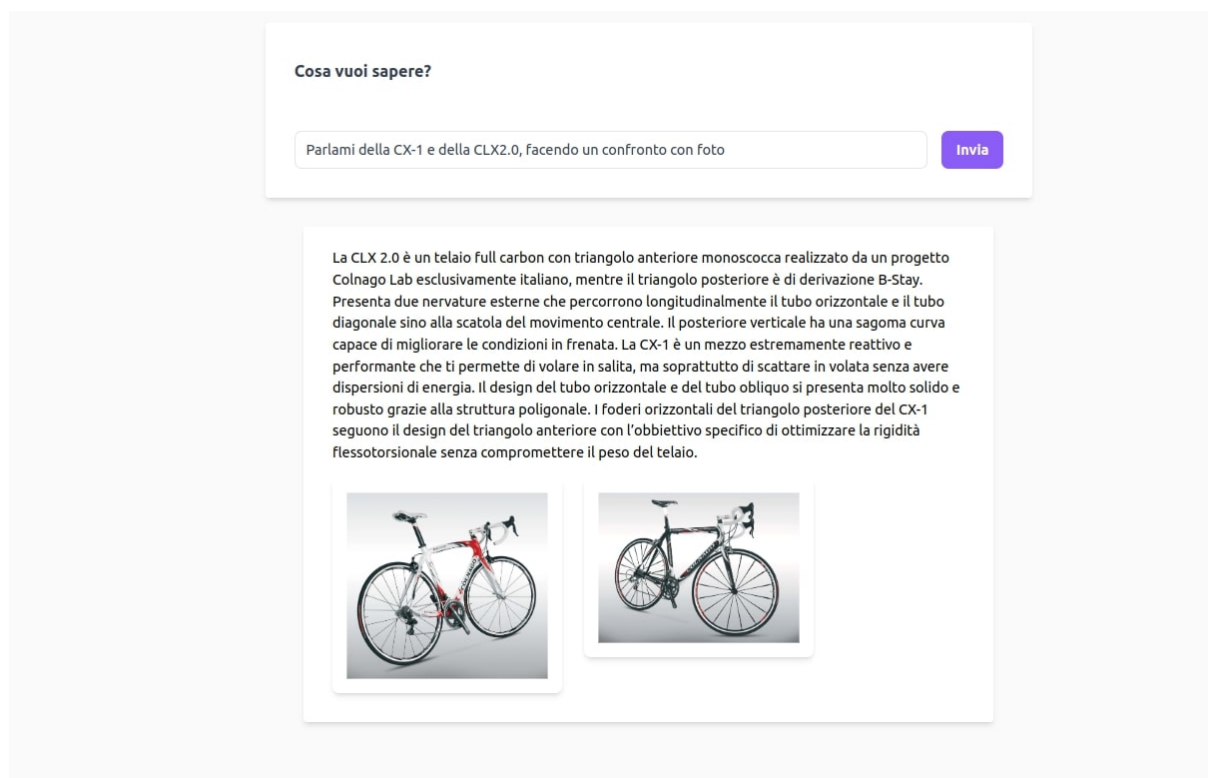


Figura 4.8: Schermata di upload di file PDF, selezione immagine

### 4.3.3 Chat

Nella sezione *Chat* è presente il chatbot che utilizza la tecnica *RAG<sub>G</sub>*, un esempio di domanda e risposta è mostrato nella figura 4.9.



**Figura 4.9:** Schermata di chat

### 4.3.4 Configurazione

Nella sezione *Configurazione* si possono configurare i principali parametri del sistema, come mostrato nella figura 4.10.

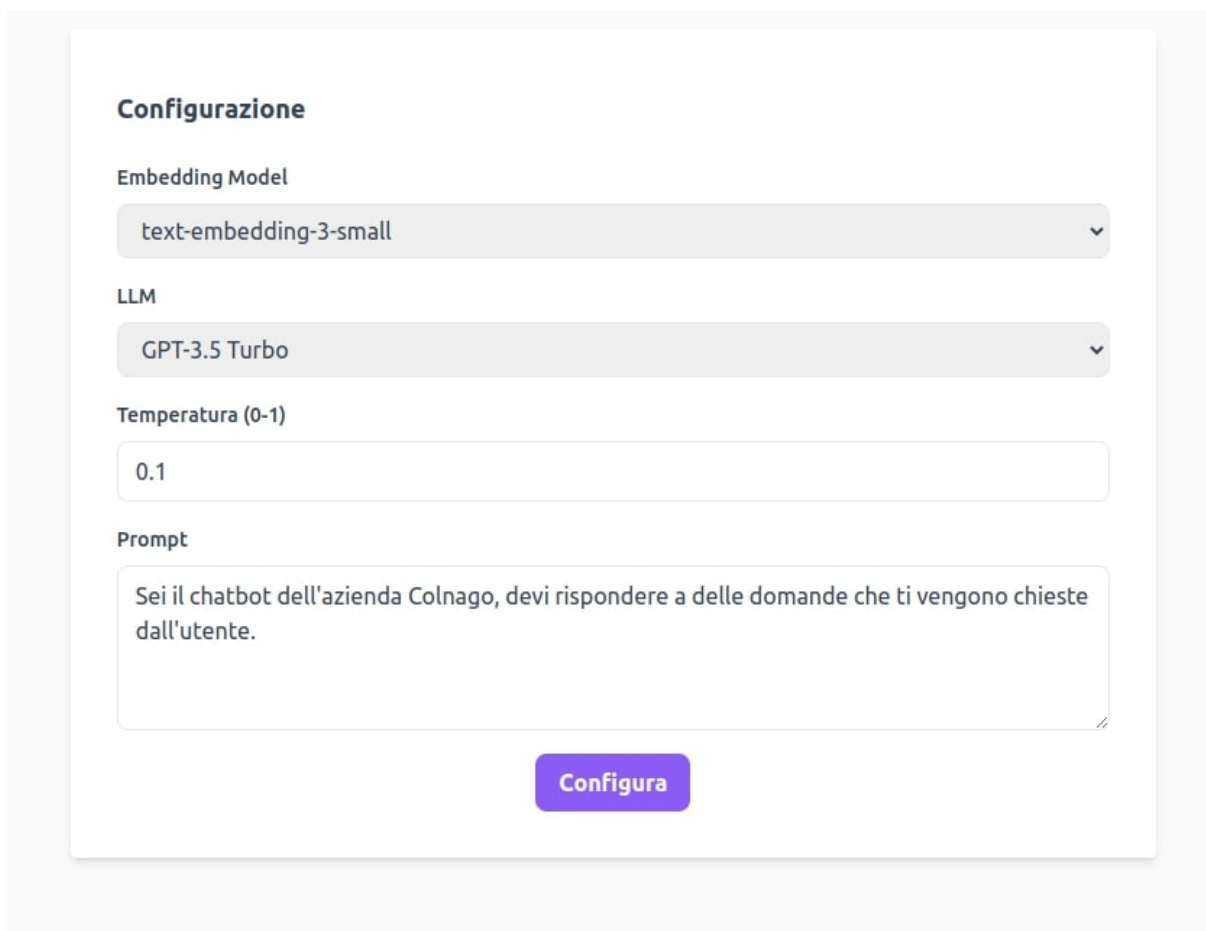


Figura 4.10: Schermata di configurazione

### 4.3.5 Knowledge

Nella sezione *Knowledge* si possono gestire i file caricati e gli *embedding<sub>G</sub>* estratti da essi. Si possono visualizzarne le informazioni principali e, per gli *embedding<sub>G</sub>*, modificare il loro contenuto o eliminarli.

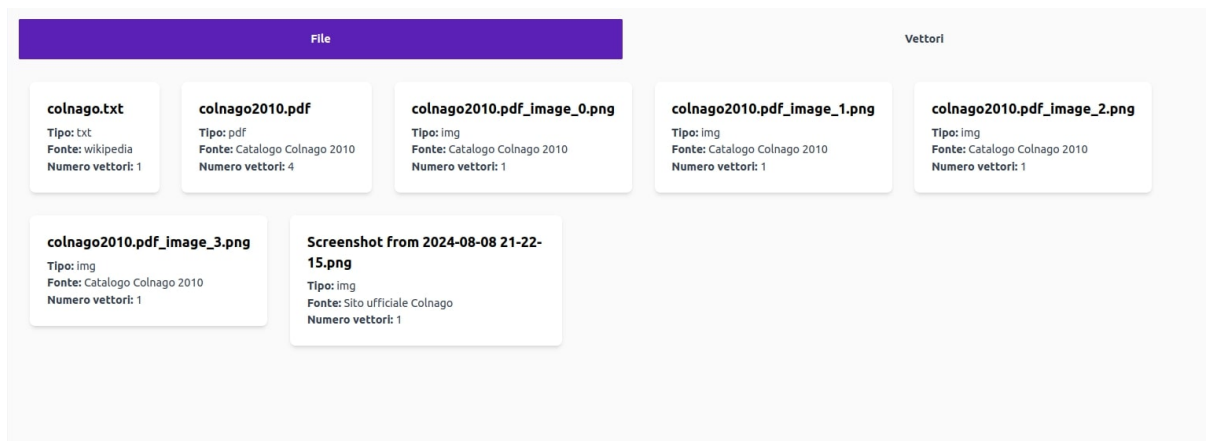


Figura 4.11: Schermata di gestione file

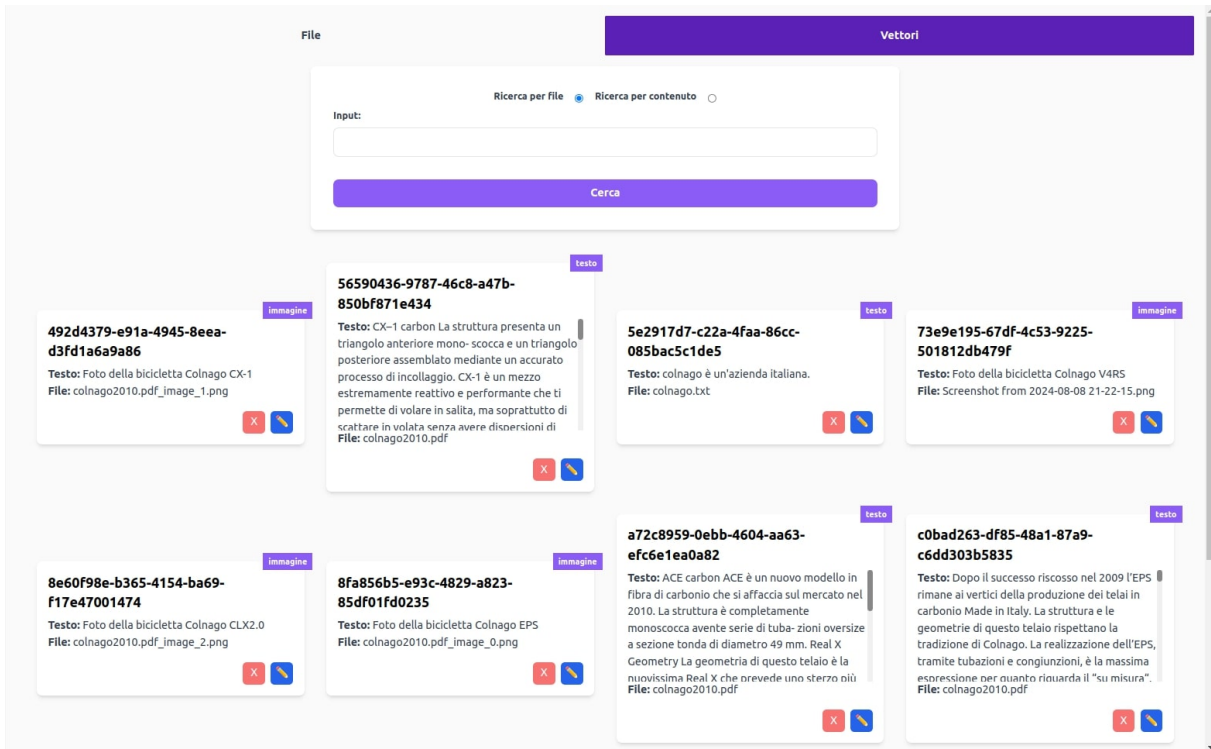


Figura 4.12: Schermata di gestione vettori

# Capitolo 5

## Conclusioni

### 5.1 Analisi critica

Il software prodotto soddisfa gli obiettivi prefissati e i requisiti individuati durante l'Analisi dei Requisiti. È stato effettuato un ridimensionamento degli obiettivi stabiliti in seguito al periodo di esplorazione tecnologica, poiché lo studio del problema ha portato delle modifiche, e soprattutto delle aggiunte, all'idea iniziale descritta in fase di pianificazione, cose oltretutto normale per un progetto prototipale ed esplorativo.

Il prodotto finale è utilizzato dall'azienda come punto di partenza per lo sviluppo dell'applicazione da mandare in produzione, lavoro che verrà fatto nei prossimi mesi andando ad aggiungere funzionalità al chatbot e integrarlo sul sito ufficiale del brand.

Fra le possibili migliorie che potevano essere applicate per uno sviluppo migliore e più consono ai dettami dell'ingegneria del software è presente sicuramente una maggiore implementazione di test automatici, per andare a costruire la base di un processo di *continuous integration* e *continuous deployment*, oltre che una validazione più completa dell'input dell'utente anche sul lato frontend.

Possibili estensioni applicabili sono:

- implementazione di una memoria per il chatbot, con tanto di cronologia messaggi e la possibilità di iniziare una nuova conversazione;
- inclusione della fonte dei contenuti utilizzati per costruire la risposta nell'output del chatbot;

- gestione ottimizzata delle tabelle, punto debole dei database vettoriali;
- caricamento di intere cartelle di file e supporto a più formati esistenti;
- stesura di un prompt di base più completo, punto cruciale della *RAG<sub>G</sub>*, difatti spesso arriva a raggiungere pagine di lunghezza;
- migliore gestione degli errori lato database e storage;
- supporto basato sull'intelligenza artificiale per la generazione delle descrizioni delle immagini da inserire nel sistema.

## 5.2 Conoscenze acquisite

Tale progetto ha portato all'acquisizione di nuove conoscenze, oltre alla consolidazione e approfondimento di conoscenze precedenti. Innanzitutto sono state studiate le più recenti tecniche legate all'intelligenza artificiale, con particolare attenzione alla *RAG<sub>G</sub>*, insieme ai relativi *framework<sub>G</sub>* che ne facilitano l'implementazione e agli algoritmi, come il *KNN<sub>G</sub>*, che ne fanno da fondamenta. È stata conseguita conoscenza sui database vettoriali, e quindi studiato un approccio alternativo ai classici database relazionali. È stata acquisita la competenza di creazione di *API<sub>G</sub>* e delle relative architetture utilizzate per implementarle, come *REST<sub>G</sub>*. È stata approfondita la conoscenza di React e studiato un *framework<sub>G</sub>* recente come Remix.

Oltre a ciò è stato possibile fare esperienza lavorativa, osservando sul campo come si lavora in un'azienda di software, il *way of working* adottato e le dinamiche di relazione con il cliente finale.

# Bibliografia

## Testi

- [6] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994 (cit. a p. [52](#)).
- [10] Scott Meyers. *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*. 3rd. Addison-Wesley Professional, 2005 (cit. a p. [52](#)).
- [11] Russ Miles e Kim Hamilton. *Learning UML 2.0*. O'Reilly, 2006 (cit. a p. [8](#)).
- [12] Mark Richards. *Software Architecture Patterns*. O'Reilly, 2015 (cit. a p. [41](#)).

## Articoli

- [7] Aman Gupta et al. «RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture». In: *arXiv preprint arXiv:2401.08406* (2024) (cit. a p. [3](#)).

# Sitografia

- [1] Pradeep Bansal. *A Comparison of python libraries for PDF Data Extraction for text, images and tables*. URL: <https://pradeepundefned.medium.com/a-comparison-of-python-libraries-for-pdf-data-extraction-for-text-images-and-tables-c75e5dbcfef8> (cit. a p. 40).
- [2] *Che cos'è LangChain? | IBM*. URL: <https://www.ibm.com/it-it/topics/langchain> (cit. a p. 5).
- [3] *Che cos'è RAG? - Spiegazione della IA Retrieval-Augmented Generation - AWS*. URL: <https://aws.amazon.com/it/what-is/retrieval-augmented-generation/> (cit. a p. 3).
- [4] *Differences between Langchain & LlamaIndex*. URL: <https://stackoverflow.com/questions/76990736/differences-between-langchain-llamaindex> (cit. a p. 5).
- [5] Martin Fowler. *Inversion of Control Containers and the Dependency Injection pattern*. 2004. URL: <https://www.martinfowler.com/articles/injection.html> (cit. a p. 52).
- [8] *Index - FastAPI*. URL: <https://fastapi.tiangolo.com/it/> (cit. a p. 39).
- [9] *LlamaIndex, Data Framework for LLM Applications*. URL: <https://www.llamaindex.ai/> (cit. a p. 5).
- [13] *Understanding indexes - Pinecone Docs*. URL: <https://docs.pinecone.io/guides/indexes/understanding-indexes> (cit. a p. 44).

- [14] *Understanding the MVC Pattern*. URL: <https://www.oracle.com/technical-resources/articles/javase/mvc.html> (cit. a p. 54).
- [15] *What is fine-tuning?* URL: <https://www.ibm.com/topics/fine-tuning> (cit. a p. 3).