⟨CodeHex16⟩

unipd.codehex16@gmail.com

Specifica Tecnica

Data	21/03/2025
Versione	1.0.0

Sommario

Specifica tecnica

Ruoli

Matteo Bazzan	Redattore, Verificatore
Luca Ribon	Redattore, Verificatore
Filippo Sabbadin	Redattore, Verificatore
Luca Rossi	Redattore

Registro delle Versioni

Versione	Data	Autore	Cambiamenti	Verificatore
1.0.0	17/05/2025	Luca Ribon	Aggiornamento requisiti e revisione finale	Filippo Sabbadin
0.6.0	16/05/2025	Luca Ribon, Matteo Bazzan	Completamento parte di Database-API	Filippo Sabbadin
0.5.0	14/05/2025	Luca Ribon, Matteo Bazzan	Completamento parte di LLM-API	Filippo Sabbadin
0.4.0	05/05/2025	Matteo Bazzan	Grafici e parti mancanti	Luca Ribon
0.3.0	20/04/2025	Luca Ribon	Architettura	Matteo Bazzan
0.2.0	10/04/2025	Filippo Sabbadin	Stesura sezioni iniziali	Luca Ribon
0.1.0	21/03/2025	Luca Rossi	Bozza iniziale struttura	Luca Ribon

Indice

Lista di immagini

Lista di tabelle

1. Introduzione

1.1. Scopo del prodotto

Il progetto consiste nella realizzazione di un **chatbot*** basato su modelli linguistici (**LLM***) pensato per i **fornitori** di beni, come bevande o alimenti, da offrire ai propri clienti. Questo sistema consente ai clienti di ottenere in modo semplice e immediato informazioni dettagliate sui prodotti o servizi disponibili, senza la necessità di contattare direttamente un operatore dell'azienda.

Il chatbot si integra con un'interfaccia dedicata al fornitore*, che permette di:

- Gestire clienti, faq* e documenti contenenti le informazioni di riferimento utilizzate dal modello linguistico per generare risposte accurate e personalizzate.
- Personalizzare graficamente la piattaforma tramite l'inserimento del logo aziendale e la selezione di una palette colori.

Per garantire la massima compatibilità e facilità d'uso, il chatbot è accessibile tramite un'interfaccia web*, che può essere utilizzata su qualsiasi dispositivo su cui è installato un browser. I linguaggi principali usati nella webapp* sono HTML*, CSS*, JavaScript*, TypeScript* e Python*, linguaggi ampiamente supportati da molti dispositivi.

1.2. Scopo del documento

Lo scopo del documento è fornire una panoramica dettagliata delle scelte progettuali e tecniche adottate per lo sviluppo del sistema. Qui verranno forniti i diagrammi UML* delle classi e le scelte architetturali, oltre a una descrizione delle tecnologie utilizzate e delle API* implementate.

1.3. Glossario

Per facilitare la comprensione di questo documento, viene fornito un glossario che chiarisce il significato dei termini specifici utilizzati nel contesto del progetto. Ogni termine di glossario è contrassegnato con un asterisco «*» in apice e collegato direttamente alla pagina web del glossario, permettendo così di accedere immediatamente alla definizione completa del termine. Le definizioni sono disponibili nel documento: `Glossario.pdf`
e nella seguente pagina web: `https://codehex16.github.io/glossario`.

1.4. Riferimenti

1.5. Riferimenti normativi

- Capitolato C7 - Assistente Virtuale Ergon:
  `https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C7.pdf`
  (ultima consultazione: 16-05-2025);
- Regolamento del progetto didattico:
  `https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf`
  (ultima consultazione: 16-05-2025);
- Norme di progetto:
  `https://codehex16.github.io/docs/3%20-%20PB/Norme-di-Progetto.pdf`
  (versione 2.0.0);

1.6. Riferimenti informativi

- Analisi dei requisiti:
  `https://codehex16.github.io/docs/3%20-%20PB/Analisi-dei-Requisiti.pdf`
  (versione 2.0.0);
- Diagrammi delle classi (UML):
  `https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf`
- Slide sui pattern architetturali del prof. Cardin *(ultima consultazione: 16-05-2025)*:
  - introduzione ai pattern:
    `https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf`
  - pattern creazionali:
    `https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf`
  - pattern strutturali:
    `https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf`
  - pattern comportamentali:
    `https://drive.google.com/file/d/1cpi6rORMxFtC91nI6_sPrG1Xn-28z8eI/view?usp=sharing`

- pattern Model View Controller:
  https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf

2. Tecnologie

In questa sezione vengono descritte le tecnologie utilizzate per lo sviluppo del sistema, suddivise in base alla loro funzione e al loro ambito di applicazione. Ogni tecnologia è accompagnata da una breve descrizione e dalle motivazioni che hanno portato alla sua scelta.

2.1. Tecnologie per la codifica

2.1.1. Linguaggi

Linguaggio	Motivazione	Versione
HTML	Utilizzato per la creazione della struttura e del contenuto delle pagine web. È il linguaggio di markup standard per la creazione di pagine web.	5
CSS	Utilizzato per la formattazione e lo stile delle pagine web. Permette di separare il contenuto dalla presentazione, migliorando la manutenibilità del codice.	3
JavaScript	Utilizzato per la creazione di interazioni dinamiche e reattive nelle pagine web.	ECMAScript 2024
Python	Scelto per la sua versatilità e facilità d'uso, è il linguaggio principale per lo sviluppo del back-end. Inoltre supporta e integra esaustivamente tutti i componenti esterni come gli LLM e i database vettoriali e non; inoltre la documentazione relativa a queste integrazioni è ampia e ben strutturata.	3.12
TypeScript	Scelto per la sua tipizzazione statica, migliora la qualità del codice e facilita la manutenzione. È utilizzato in combinazione con Svelte per lo sviluppo del front-end.	5.8.3

Tabella 1: Linguaggi utilizzati

2.1.2. Framework

Nome	Motivazione	Versione
Svelte	Scelto per la sua semplicità e leggerezza, è il framework utilizzato per il rendering delle pagine del front-end. Permette di creare UI e UX gradevoli con una struttura e semantica del codice che il gruppo ha preferito rispetto ad altri framework.	5.30.1
SvelteKit	Scelto per la sua integrazione con Svelte, è un framework per lo sviluppo di applicazioni web. Permette di gestire implementare funzioni come ottimizzazione delle build, routing in modo semplice.	2.21.0
FastAPI	Framework scelto per la sua facilità nell'implementazione di API REST. È utilizzato per il back-end per far comunicare tra loro i componenti software.	0.115.12

Tabella 2: Framework utilizzati

2.1.3. Librerie

PYTHON

Nome	Motivazione	Versione
langchain	Libreria scelta per l'integrazione tra modelli AI e database, permette di gestire documenti, contesto e query rivolte all'LLM integrato.	0.3.25
passlib	Libreria che implementa diversi algoritmi di hashing che sono stati utilizzati per la gestione delle password.	1.7.4
pydantic_core	Libreria utile per la modellazione di schemi utilizzati nella trasferimento di dati tramite chiamate API.	2.33.2
bcrypt	Libreria per l'hashing delle password, specificamente implementa l'algoritmo bcrypt, noto per la sua robustezza.	4.3.0

Nome	Motivazione	Versione
motor	Driver asincrono per MongoDB, permette di interagire con il database in modo non bloccante, ideale per applicazioni web moderne (es. con FastAPI).	3.7.0
python-jose	Libreria per la gestione di token JWT (JSON Web Tokens) e altre specifiche JOSE (Javascript Object Signing and Encryption), utile per l'autenticazione e la trasmissione sicura di informazioni.	3.4.0
requests	Libreria HTTP per Python, utilizzata per inviare richieste HTTP/1.1 di tutti i tipi (GET, POST, PUT, ecc.).	2.32.3
pymongo	Libreria utilizzata per la manipolazione del database MongoDB.	4.12.1
uvicorn	Server ASGI (Asynchronous Server Gateway Interface) ad alte prestazioni, comunemente utilizzato per eseguire applicazioni web asincrone Python come quelle basate su FastAPI o Starlette.	0.34.2
jwt	Libreria per la codifica e decodifica di JSON Web Tokens (JWT), utilizzata per l'autenticazione e lo scambio sicuro di informazioni.	1.3.1
fastapi-mail	Estensione per FastAPI che facilita l'invio di email, supportando l'invio asincrono e la gestione di template.	1.4.2
pytz	Libreria per la gestione accurata dei fusi orari in Python, basata sul database Olson tz.	2025.2
starlette	Framework ASGI (Asynchronous Server Gateway Interface) leggero e toolkit, su cui sono costruiti framework più completi come FastAPI. Fornisce le basi per costruire servizi web asincroni.	0.46.2
openai	Libreria client ufficiale di OpenAI per interagire con le loro API	1.77.0

Nome	Motivazione	Versione
chromadb	Database vettoriale open-source progettato per applicazioni AI, facilita la memorizzazione, l'interrogazione e la gestione di embedding per funzionalità come la ricerca semantica.	0.6.3
bson	Libreria per la codifica e decodifica di BSON (Binary JSON), il formato di serializzazione dei dati utilizzato da MongoDB.	0.5.10
pypdf	Libreria Python pura per la manipolazione di file PDF: permette di dividere, unire, ritagliare, crittografare, decrittografare pagine PDF e estrarre testo.	5.4.0
python-multipart	Libreria per il parsing di dati multipart/form-data, comunemente usata nei web framework per gestire upload di file e dati di form complessi.	0.0.20
fastapi	Core framework per la creazione di API RESTful, scelto per la sua velocità e semplicità d'uso.	0.115.12
pydantic	Libreria per la validazione dei dati e la gestione delle impostazioni, utilizzata da FastAPI.	2.11.4
langchain-openai	Integrazione specifica di LangChain per utilizzare i modelli forniti da OpenAI.	0.3.16
langchain_chroma	Integrazione specifica di LangChain per l'interazione con il database vettoriale ChromaDB.	0.2.3
langchain_community	Collezione di integrazioni e componenti della comunità per LangChain.	0.3.23
httpx	Client HTTP asincrono per Python, utilizzato per effettuare richieste HTTP in modo non bloccante.	0.28.1

Tabella 3: Librerie utilizzate

TypeScript

Nome	Motivazione	Versione
tailwindcss	Un framework CSS utility-first per costruire rapidamente interfacce utente personalizzate direttamente nel markup HTML.	3.4.9
vite	Uno strumento di build per il frontend moderno che offre un'esperienza di sviluppo estremamente veloce e bundle ottimizzati per la produzione.	6.0.0
lucide-svelte	Libreria di icone SVG (basate su Lucide) per componenti Svelte.	0.468.0
marked	Un parser Markdown per convertire Markdown in HTML.	15.0.7
mode-watcher	Utility per Svelte per rilevare e reagire ai cambiamenti della preferenza di tema del sistema operativo (chiaro/scuro) o per gestirla manualmente.	0.5.0
lodash-es	Libreria di utilità che fornisce funzioni helper modulari e performanti per la manipolazione di dati.	4.17.21
chalk	Libreria per la stilizzazione delle stringhe nel terminale, utile per migliorare la leggibilità degli output durante lo sviluppo.	5.4.1

Tabella 4: Librerie TypeScript utilizzate

2.1.4. Strumenti e servizi

Strumento	Motivazione	Versione
Git	Utilizzato per il versionamento del codice sorgente, permette di tenere traccia delle modifiche e collaborare con altri membri del team	2.49.0
GPT-4o mini	Il modello utilizzato per il chatbot, scelto in base al prezzo e qualità delle risposte	-
Docker	Utilizzato per suddividere ed eseguire in container l'applicazione, rendendola facilmente distribuibile e scalabile in diversi ambienti	28.1.1

Strumento	Motivazione	Versione
MongoDB	Database NoSQL utilizzato per memorizzare documenti, cronologia delle conversazioni e utenti. Scelto perché rende più facile e diretto memorizzare i file come formato json	8.0
ChromaDB	Database vettoriale per memorizzare e recuperare embedding dei documenti, consentendo ricerche semantiche rapide per fornire risposte contestuali. Scelto per la sua velocità e facilità d'uso, è in grado di gestire grandi volumi di dati e query complesse.	1.0.4

Tabella 5: Strumenti e servizi utilizzati

2.2. Tecnologie per i test

Tecnologia	Motivazione	Versione
Github Actions	Utilizzato per l'integrazione continua e il testing automatico del codice. Permette di eseguire test e controlli di qualità ogni volta che viene effettuata una modifica al codice sorgente o prima di un merge con un altro branch	-
Pytest	Utilizzato per il testing del codice Python, permette di scrivere test in modo semplice e intuitivo. È stato scelto per la sua facilità d'uso e per la sua integrazione con FastAPI	8.3.5
Vitest	È un framework di testing per applicazioni TypeScript e, in particolare, adatto anche per essere utilizzato con SvelteKit. Permette di scrivere test di integrazione e unitari ed è integrato con il sistema di build Vite che utilizziamo nel progetto.	

Tabella 6: Tecnologie per il testing utilizzate

3. Architettura

In questa sezione viene presentata l'architettura del sistema, suddivisa in due parti principali: il front-end e il back-end. Ogni parte è descritta in dettaglio, evidenziando le tecnologie di codifica utilizzate e le scelte architetturali adottate.

Il sistema adotta un'**architettura a microservizi**, composta da un frontend (*Suppl-AI*), un'API per la gestione della persistenza dei dati (*Database-API*) e un'API interfacciarsi con l'LLM (*LLM-API*).

I motivi che ci hanno portato a scegliere questa architettura sono:
- **Modularità**: ogni microservizio è responsabile di una funzionalità specifica, facilitando la manutenzione e l'evoluzione del sistema; infatti eventuali sviluppi futuri considerabili dal proponente, come un'applicazione mobile nativa, vengono semplificati dalla struttura a microservizi.
- **Diversità delle tecnologie**: ogni microservizio può essere sviluppato utilizzando tecnologie diverse. Infatti, utilizzando SvelteKit per il frontend siamo riusciti a separare il codice Svelte/TypeScript da quello Python utilizzato dai servizi backend*.
- **Principio di singola responsabilità**: ogni microservizio ha responsabilità e scopo ben definiti, facilitando la comprensione e la gestione del codice.

3.1. Flusso del sistema

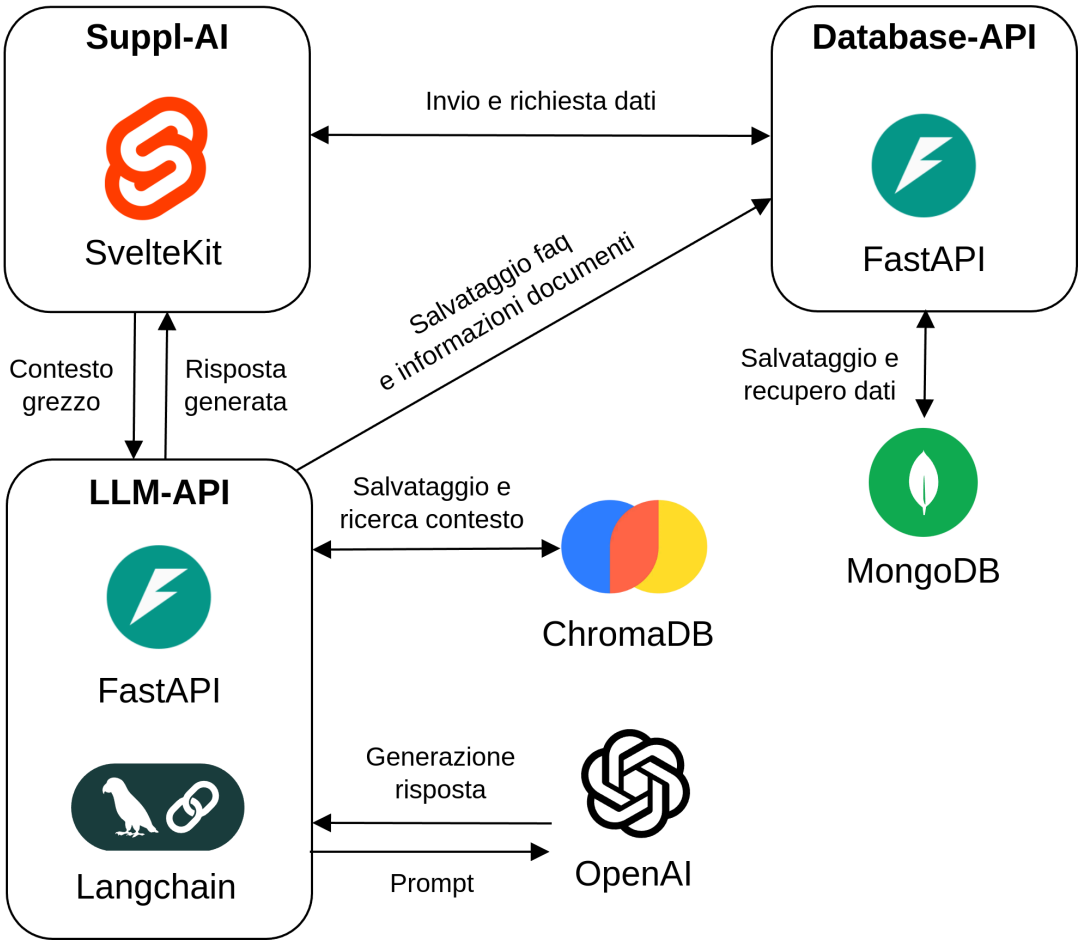

Figura 1: Flusso del sistema

- **Invio contesto**: tramite il client Suppl-AI* l'utente può inviare documenti e FAQ al chatbot che verranno memorizzati nel database vettoriale.
- **Interazione con il chatbot**:
  - **Richiesta generazione risposta**: l'utente invia una domanda al chatbot tramite il client *Suppl-AI*; la domanda viene inviata, insieme allo storico della chat ad *LLM-API*.
  - **Generazione prompt**: *LLM-API* recupera il contesto da *ChromaDB* tramite la domanda e lo storico della chat, e genera un prompt per l'LLM. Il prompt contiene la domanda dell'utente, lo storico della chat e il contesto recuperato.
  - **Generazione risposta**: *LLM-API* si interfaccia con le API di *OpenAI* per generare una risposta in base al prompt costruito. La risposta generata viene poi restituita al client *Suppl-AI*.
- **Interazione con il database**:

- **Autenticazione**: il client può inviare le credenziali di accesso a *Database-API* per autenticarsi. *Database-API* verifica le credenziali e restituisce un token JWT al client, che viene utilizzato per autenticare le richieste successive.
- **Invio dei dati**: tramite il client, l'utente può inviare chat, documenti, FAQ e utenti al *Database-API*.
- **Salvataggio dei dati**: *Database-API* riceve i dati e li manipola per salvarli correttamente nel database *MongoDB*.
- **Richiesta dei dati**: il client può richiedere chat, documenti, FAQ e utenti al *Database-API*.
- **Recupero dei dati**: *Database-API* riceve la richiesta e recupera i dati dal database *MongoDB* per poi restituirli al client.

3.2. Architettura Frontend

3.2.1. Suppl-AI

Suppl-AI è il nome del frontend della webapp. Per il suo sviluppo sono stati utilizzati ***Svelte***, ***SvelteKit*** e ***TypeScript***.

SvelteKit è un framework* frontend che integra Svelte e consente di creare interfacce utente reattive e performanti, mentre **TypeScript** è un superset* di JavaScript che aggiunge tipizzazione statica al linguaggio, viene utilizzato per la logica di gestione e manipolazione dei dati visualizzati.

3.2.1.1. Struttura del codice

Ogni pagina della webapp è composta da un file Svelte `+page.svelte`, e da un file TypeScript `+page.server.ts`.

3.2.1.1.1. File Svelte

I file `+page.svelte` contengono il codice HTML, CSS e TS, utilizzato per la gestione del layout della pagina. In aggiunta alla sintassi HTML, Svelte include funzionalità aggiuntive come l'uso dei costrutti `{#if}`, `{#each}` e `{#await}` per la gestione della logica di rendering condizionale, cicli e promesse; in più implementa anche un sistema di reattività che consente di aggiornare automaticamente l'interfaccia utente quando c'è un cambiamento di stato dei componenti.

Inoltre da la possibilità di creare e riutilizzare componenti, che possono essere importati in altri file `.svelte`.

3.2.1.1.2. File Typescript

I file TypeScript `+page.server.ts` contengono la logica di gestione dei dati contenuti nella pagina; questi file sono eseguiti lato server. Sono responsabili anche della gestione delle chiamate API al backend.

3.2.1.2. Design pattern utilizzati

3.2.1.2.1. Model View Controller

Ogni pagina della webapp è composta da un file `+page.svelte`, e da un file `+page.server.ts`. Questa struttura segue il pattern Model-View-Controller* (MVC), in cui il file Svelte rappresenta la *View*, mentre il file TypeScript rappresenta il *Controller*, il *Model* invece è rappresentato dalle backend API.

3.2.1.2.2. Composite

I file Svelte sono strutturati tramite il pattern *composite*, in quanto sono composti da più componenti che possono essere riutilizzati in altre pagine.

3.3. Architettura Backend

Per lo sviluppo delle API backend sono stati utilizzati ***FastAPI*** e ***Python***.

Python è il linguaggio principale utilizzato per lo sviluppo del backend del prodotto. È stato scelto per l'integrazione ottimale, tramite librerie, di LLM e database NoSQL e vettoriali.

FastAPI è un framework per la creazione di API RESTful, progettato per essere veloce e semplice da usare. È stato scelto perché risulta essere più essenziale con le funzionalità di cui avevamo bisogno rispetto ad altri framework.

Inoltre, FastAPI supporta la tipizzazione statica delle richieste, che aiuta a prevenire errori e rende il codice più leggibile e manutenibile. Infine genera automaticamente la documentazione delle API, rendendo più semplice la comprensione e l'utilizzo delle stesse per tutti gli sviluppatori del progetto.

3.3.1. Database API

Database-API è il nome del API backend che si occupa della gestione della persistenza dei dati. Per l'interazione con il database ***MongoDB*** sono state importate diverse librerie, tra cui ***motor*** e ***pymongo***.

Motor è un driver asincrono per MongoDB, progettato per funzionare con framework asincroni come FastAPI. Permette di eseguire operazioni di database in modo non bloccante, migliorando le prestazioni e la reattività dell'applicazione.

Pymongo è una libreria Python per interagire con MongoDB. È stata utilizzata per alcune operazioni di database che non richiedevano l'asincronia, come la gestione della connessione al database.

3.3.1.1. Struttura del codice

Il codice di Database-API è strutturato in ***router***, ***repository*** e ***service***.

3.3.1.1.1. Router

I router sono responsabili della gestione delle richieste HTTP e dell'instradamento delle stesse alle funzioni appropriate. Ogni router è dedicato a una delle seguenti funzionalità specifica del sistema:

- `auth.py`;
- `chat.py`;
- `document.py`;
- `faq.py`;
- `setting.py`;
- `user.py`;

3.3.1.1.2. Repository

I repository sono responsabili dell'interazione con il database. Gestiscono le operazioni CRUD (Create, Read, Update, Delete) e forniscono un'interfaccia per accedere ai dati. I repository sono suddivisi in:

- `chat_repository.py`;
- `document_repository.py`;
- `faq_repository.py`;
- `setting_repository.py`;
- `user_repository.py`;

3.3.1.2. Design pattern utilizzati

3.3.1.2.1. Repository

Il pattern *repository* è stato utilizzato per separare la logica di accesso ai dati dalla gestione delle chiamate API. Questo consente di mantenere il codice più manutenibile.

Infatti le classi repository implementano le operazioni CRUD e vengono utilizzate dai vari router per interagire con il database.

3.3.1.2.2. Singleton

Il pattern *singleton* è stato utilizzato per garantire che ci sia una sola istanza del database in tutta l'applicazione. Questo è importante per evitare conflitti e garantire la coerenza dei dati.

3.3.1.2.3. Dependency Injection

Il pattern ***dependency injection*** è un pattern strutturale che consente di includere le dipendenze necessarie in una classe, invece che crearle all'interno della classe stessa. Questo pattern permette di ridurre l'accoppiamento tra le classi e di rendere il codice più modulare.

3.3.1.2.4. Service

I service sono delle classi che offrono funzionalità di business logic, come la gestione dell'autenticazione e l'invio di email. I service sono:

- `auth_service.py`;
- `email_service.py`;

3.3.1.3. Diagramma delle classi

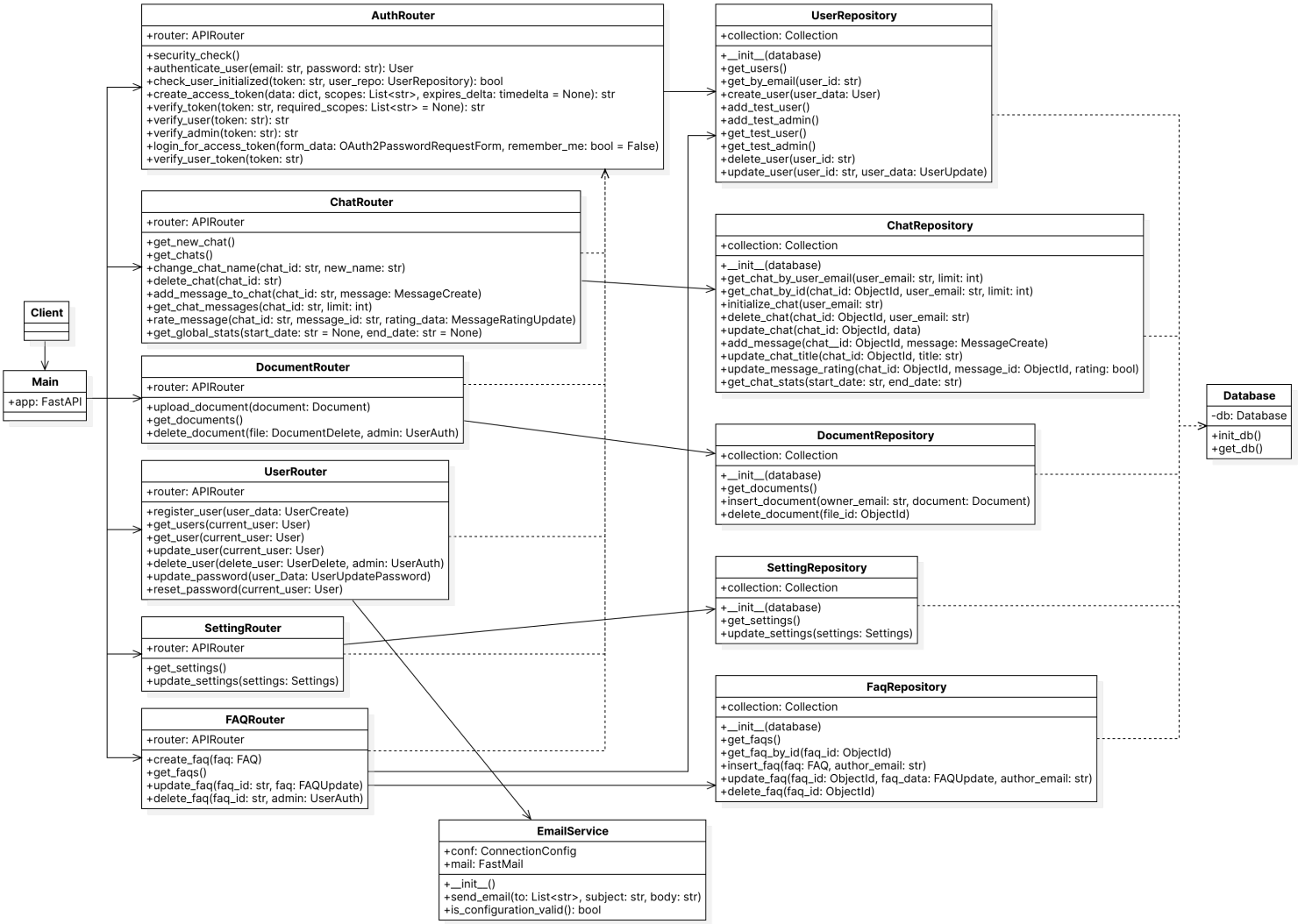

Figura 2: Diagramma delle classi di Database-API

3.3.1.3.1. DTO

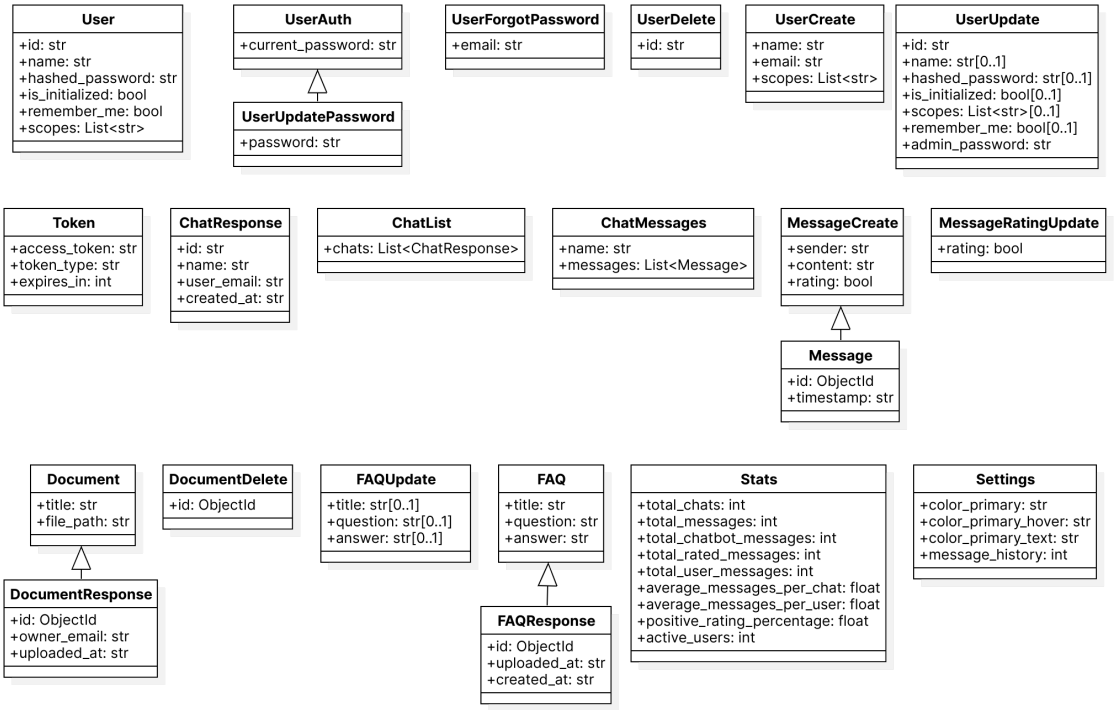

Figura 3: Diagramma delle classi dei DTO di Database-API

I DTO (Data Transfer Object) sono oggetti utilizzati per trasferire dati tra le diverse parti del sistema, in particolare tra il client e il server Database-API. Semplificano la comunicazione e la validazione dei dati.

3.3.1.3.1.1. User

Rappresenta un utente nel sistema.

- `+id: str`: l'indirizzo email dell'utente, utilizzato come identificativo univoco nel database;
- `+name: str`: il nome dell'utente;
- `+hashed_password: str`: la password dell'utente, memorizzata come hash
- `+is_initialized: bool`: indica se l'utente ha completato la configurazione iniziale (cambio password al primo accesso). Default: `False`;
- `+remember_me: bool`: indica se l'utente ha scelto di essere ricordato per accessi futuri. Default: `False`;
- `+scopes: List[str]`: lista dei permessi associati all'utente (es. «user», «admin»). Default: `["user"]`;

3.3.1.3.1.2. UserAuth

Utilizzato per l'autenticazione e operazioni che richiedono la password corrente.
- `+current_password: str`: la password attuale dell'utente;

3.3.1.3.1.3. UserUpdatePassword

Generalizzazione della classe UserAuth. Utilizzato per l'aggiornamento della password dell'utente.
- `+password: str`: la nuova password scelta dall'utente. Deve rispettare i criteri di complessità definiti;

3.3.1.3.1.4. UserForgotPassword

Utilizzato per la richiesta di recupero password.
- `+email: str`: l'indirizzo email dell'utente che ha dimenticato la password;

3.3.1.3.1.5. UserUpdate

Utilizzato per aggiornare i dati di un utente. Tutti i campi sono opzionali.
- `+id: str`: l'identificativo dell'utente da aggiornare;
- `+name: Optional[str]`: il nuovo nome dell'utente;
- `+password: Optional[str]`: la nuova password dell'utente;
- `+is_initialized: Optional[bool]`: il nuovo stato di inizializzazione;
- `+remember_me: Optional[bool]`: il nuovo stato per «ricordami»;
- `+scopes: Optional[List[str]]`: la nuova lista di permessi;
- `+admin_password: Optional[str]`: la password dell'amministratore, richiesta per alcune modifiche;

3.3.1.3.1.6. UserDelete

Utilizzato per eliminare un utente.
- `+id: str`: l'identificativo dell'utente da eliminare;

3.3.1.3.1.7. UserCreate

Utilizzato per la creazione di un nuovo utente.
- `+name: str`: il nome del nuovo utente;
- `+email: str`: l'indirizzo email del nuovo utente;
- `+scopes: Optional[List[str]]`: la lista di permessi per il nuovo utente;

3.3.1.3.1.8. Token

Rappresenta un token di accesso JWT.
- `+access_token: str`: il token JWT;
- `+token_type: str`: il tipo di token (solitamente «bearer»);
- `+expires_in: Optional[int]`: la durata di validità del token in secondi;

3.3.1.3.1.9. ChatResponse

Rappresenta i dati di una chat restituiti in una lista.
- `+id: str`: l'identificativo univoco della chat;
- `+name: str`: il nome della chat;
- `+user_email: str`: l'email dell'utente proprietario della chat;
- `+created_at: Optional[str]`: la data e ora di creazione della chat;

3.3.1.3.1.10. ChatList

Contiene una lista di chat.
- `+chats: List[ChatResponse]`: la lista degli oggetti ChatResponse;

3.3.1.3.1.11. Message

Rappresenta un singolo messaggio all'interno di una chat.
- `+id: ObjectId`: l'identificativo univoco del messaggio;
- `+sender: str`: il mittente del messaggio (es. «user» o «chatbot»);
- `+content: str`: il contenuto testuale del messaggio;
- `+timestamp: str`: la data e ora di invio del messaggio;
- `+rating: Optional[bool]`: la valutazione del messaggio (True per positivo, False per negativo, None se non valutato);

3.3.1.3.1.12. ChatMessages

Rappresenta una chat con la sua lista di messaggi.
- `+name: str`: il nome della chat;
- `+messages: List[Message]`: la lista dei messaggi della chat;

3.3.1.3.1.13. MessageCreate

Utilizzato per creare un nuovo messaggio.
- `+content: str`: il contenuto del messaggio;
- `+sender: str`: il mittente del messaggio. Default: «user»;
- `+rating: Optional[bool]`: la valutazione iniziale del messaggio;

3.3.1.3.1.14. MessageRatingUpdate

Utilizzato per aggiornare la valutazione di un messaggio.
- `+rating: Optional[bool]`: la nuova valutazione del messaggio;

3.3.1.3.1.15. Document

Rappresenta le informazioni base di un documento.
- `+title: str`: il titolo del documento;
- `+file_path: str`: il percorso del file del documento nel sistema;

3.3.1.3.1.16. DocumentResponse

Generalizzazione della classe Document. Rappresenta un documento con informazioni aggiuntive restituito dall'API.
- `+id: ObjectId`: l'identificativo univoco del documento ;
- `+owner_email: EmailStr`: l'email del proprietario del documento;
- `+uploaded_at: str`: la data e ora di caricamento del documento;

3.3.1.3.1.17. DocumentDelete

Utilizzato per specificare l'ID di un documento da eliminare.
- `+id: str`: l'identificativo del documento da eliminare;

3.3.1.3.1.18. FAQ

Rappresenta una singola FAQ (Frequently Asked Question).
- `+title: str`: il titolo della FAQ (massimo 30 caratteri);
- `+question: str`: la domanda della FAQ;
- `+answer: str`: la risposta alla FAQ;

3.3.1.3.1.19. FAQUpdate

Utilizzato per aggiornare una FAQ esistente. Tutti i campi sono opzionali.
- `+title: Optional[str]`: il nuovo titolo della FAQ (massimo 30 caratteri);
- `+question: Optional[str]`: la nuova domanda della FAQ;
- `+answer: Optional[str]`: la nuova risposta della FAQ;

3.3.1.3.1.20. FAQResponse

Generalizzazione della classe FAQ. Rappresenta una FAQ con informazioni aggiuntive restituita dall'API.
- `+id: ObjectId`: l'identificativo univoco della FAQ nel database;

- `+created_at: str`: la data e ora di creazione della FAQ;
- `+updated_at: str`: la data e ora dell'ultimo aggiornamento della FAQ;

3.3.1.3.1.21. EmailSchema

Utilizzato per inviare una lista di indirizzi email.
- `+email: List[str]`: una lista di indirizzi email;

3.3.1.3.1.22. Settings

Rappresenta le impostazioni di personalizzazione della piattaforma. Tutti i campi sono opzionali.
- `+color_primary: Optional[str]`: il colore primario dell'interfaccia;
- `+color_primary_hover: Optional[str]`: il colore primario al passaggio del mouse;
- `+color_primary_text: Optional[str]`: il colore del testo per elementi con colore primario;
- `+message_history: Optional[int]`: la durata (in giorni o numero di messaggi, da definire) per cui conservare lo storico dei messaggi;

3.3.1.3.1.23. Stats

Rappresenta le statistiche di utilizzo della piattaforma.
- `+total_chats: int`: numero totale di chat;
- `+total_messages: int`: numero totale di messaggi;
- `+total_chatbot_messages: int`: numero totale di messaggi inviati dal chatbot;
- `+total_rated_messages: int`: numero totale di messaggi valutati;
- `+total_user_messages: int`: numero totale di messaggi inviati dagli utenti;
- `+average_messages_per_chat: float`: media di messaggi per chat;
- `+average_messages_per_user: float`: media di messaggi per utente;
- `+positive_rating_percentage: float`: percentuale di valutazioni positive;
- `+active_users: int`: numero di utenti attivi;

3.3.1.3.2. AuthRouter

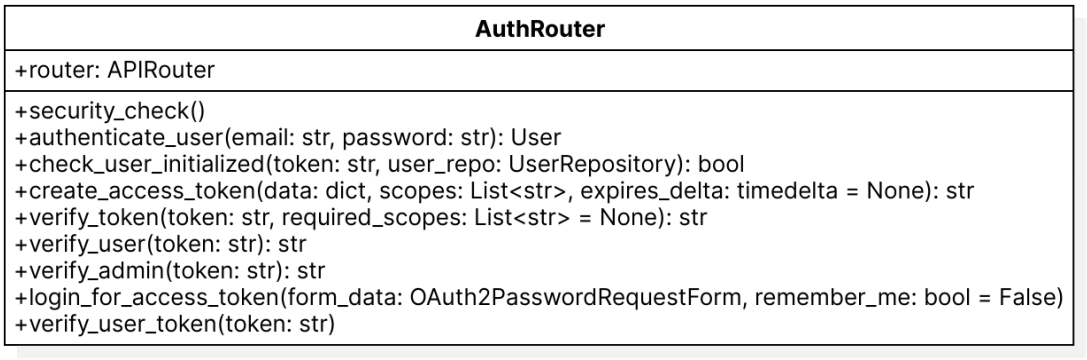

Figura 4: Diagramma delle classi di AuthRouter

La classe `AuthRouter` gestisce le operazioni di autenticazione e autorizzazione degli utenti. Utilizza il pacchetto `fastapi.security` per implementare l'autenticazione basata su token JWT. Le operazioni principali includono la registrazione, il login, il recupero della password e la modifica della password. I metodi della classe UserRepository vengono utilizzati tramite l'oggetto restituito dal metodo `UserRepository.get_user_repository()` (**Dependency Injection**).

3.3.1.3.2.1. Attributi

- `router: APIRouter`: l'oggetto del modulo fastapi che permette di definire le API route;

3.3.1.3.2.2. Metodi

- `+security_check()`: verifica se è stata impostata la SECRET_KEY_JWT utilizzata per la generazione dei token JWT;
- `+authenticate_user(email: str, password: str) -> User`: autentica un utente in base all'email e alla password fornita; restituisce l'oggetto User se l'autenticazione ha successo;
- `+check_user_initialized(token: str, user_repo: UserRepository) -> bool`: restituisce True se l'utente ha completato la configurazione iniziale (cambio password al primo accesso), altrimenti false;
- `+create_access_token(data: dict, scopes: List[str], expires_delta: Optional[timedelta] = None) -> str`: ritorna un token di accesso JWT che contiene i dati forniti e durata uguale a expires_delta se specificata;
- `+verify_token(token: str, required_scopes: List[str] = None) -> str`: verifica la validità di un token JWT e restituisce i dati contenuti nel token se valido;
- `+verify_user(token: str) -> str`: verifica se il token è corretto e, se valido, restituisce il contenuto del token;

- `+verify_admin(token: str) -> str`: verifica se il token è corretto e, se valido, verifica che il token contenga l'amministratore tra i ruoli; se il token è quello di un amministratore, restituisce il contenuto del token;
- `+login_for_access_token(form_data: OAuth2PasswordRequestForm, remember_me: bool = False)`: metodo accessibile tramite API che verifica le credenziali dell'utente, inizializza i permessi, aggiorna la flag `remember_me` nel database e, se l'autenticazione ha successo, restituisce un token JWT con durata che dipende dalla flag `remember_me`;
- `+verify_user_token(token: str)`: metodo accessibile tramite API che verifica la validità del token JWT e restituisce i permessi dell'utente se il token è valido;

3.3.1.3.2.3. ChatRouter

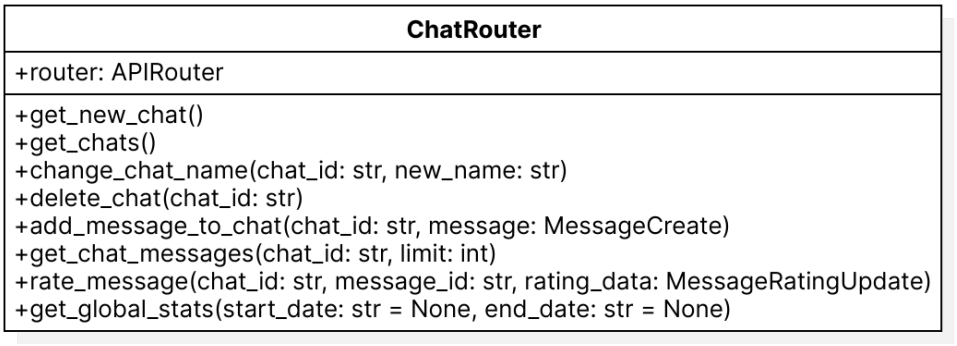

Figura 5: Diagramma delle classi di ChatRouter

La classe `ChatRouter` gestisce le operazioni relative alle chat, inclusa la creazione, il recupero e l'aggiornamento delle chat e dei messaggi. Le operazioni di creazione e aggiornamento richiedono l'autenticazione dell'utente (`Depends(verify_user)`), mentre le operazioni di eliminazione richiedono privilegi di amministratore (`Depends(verify_admin)`).
Le interazioni con il database sono demandate all'oggetto `ChatRepository` che viene ottenuto tramite il metodo `ChatRepository.get_chat_repository()` (**Dependency Injection**).

3.3.1.3.2.4. Attributi

- `router: APIRouter`: l'oggetto del modulo `fastapi` che permette di definire le API route;

3.3.1.3.2.5. Metodi

- `+get_new_chat()`: crea una nuova chat, per l'utente autenticato, di cui restituisce l'id;
- `+get_chats()`: ritorna la lista delle chat dell'utente autenticato;
- `+change_chat_name(chat_id: str, new_name: str)`: cambia il nome della chat `chat_id` con il nuovo nome `new_name`;
- `+delete_chat(chat_id: str)`: elimina la chat con l'id `chat_id`;

- `+add_message_to_chat(chat_id: str, message: MessageCreate)`: aggiunge un nuovo messaggio alla chat con l'id `chat_id`; i dati del messaggio sono forniti tramite l'oggetto `message`;
- `+get_chat_messages(chat_id: str, limit: int)`: ritorna i messaggi della chat con l'id `chat_id`; se `limit` è definito ritorna gli ultimi n messaggi della chat, dove n è il valore di `limit`;
- `+rate_message(chat_id: str, message_id: str, rating: MessageRatingUpdate)`: aggiorna la valutazione del messaggio con l'id `message_id` nella chat `chat_id`; la valutazione è fornita tramite l'oggetto `rating`, può essere `True` (valutazione positiva), `False` (valutazione negativa) o `None` (non valutato);
- `+get_global_stats(start_data: str = None, end_date: str = None)`: ritorna le statistiche globali del sistema, come il numero totale di chat, messaggi, valutazioni; solo gli admin possono utilizzare questo metodo;

3.3.1.3.3. DocumentRouter

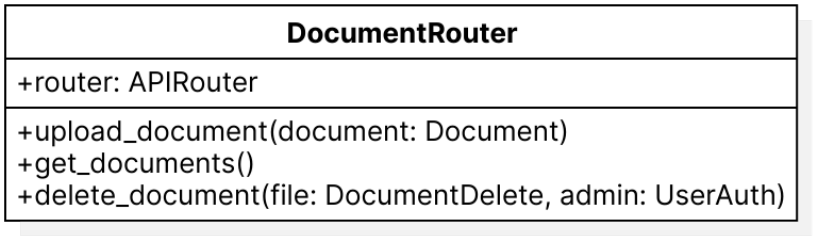

Figura 6: Diagramma delle classi di DocumentRouter

La classe `DocumentRouter` gestisce le operazioni relative ai documenti, come il caricamento, il recupero e l'eliminazione. Le operazioni di caricamento e recupero dei documenti richiedono privilegi di amministratore (`Depends(verify_admin)`). L'eliminazione di un documento richiede privilegi di amministratore e la conferma della password dell'amministratore. Le interazioni con il database per i metadati dei documenti sono demandate a `DocumentRepository`, mentre la verifica delle credenziali dell'amministratore è gestita tramite `UserRepository` e la funzione `authenticate_user`.

3.3.1.3.3.1. Attributi

- `router: APIRouter`: l'oggetto del modulo `fastapi` che permette di definire le API route;

3.3.1.3.3.2. Metodi

- `+upload_document(document: Document)`: carica un nuovo documento. Richiede privilegi di amministratore;
- `+get_documents() -> List[Document]`: restituisce una lista di tutti i documenti. Richiede privilegi di amministratore;

- `+delete_document(file: DocumentDelete)`: elimina un documento specificato. Richiede privilegi di amministratore e la password dell'amministratore che esegue l'operazione;

3.3.1.3.4. UserRouter

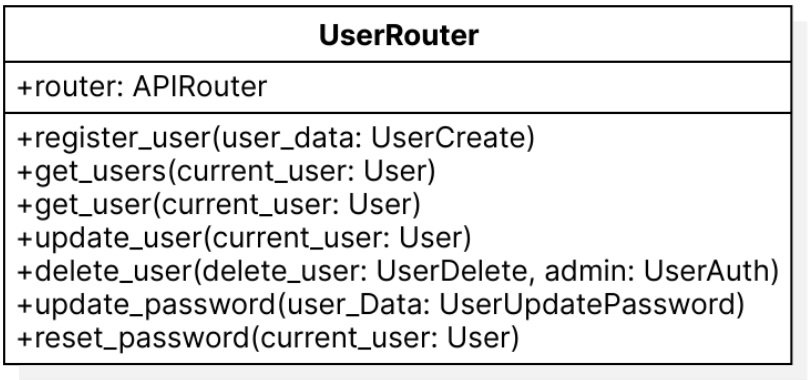

Figura 7: Diagramma delle classi di UserRouter

La classe `UserRouter` gestisce le chiamate alle operazioni relative agli utenti, inclusa la registrazione, il recupero dei dati, l'aggiornamento e l'eliminazione. Gestisce anche le funzionalità di modifica e reset della password. Molte operazioni richiedono privilegi di amministratore, verificati tramite `Depends(verify_admin)`, mentre altre richiedono solo l'autenticazione dell'utente (`Depends(verify_user)`). Le interazioni con il database sono demandate a `UserRepository`.

3.3.1.3.4.1. Attributi

- `router: APIRouter`: l'oggetto del modulo `fastapi` che permette di definire le API route;

3.3.1.3.4.2. Metodi

- `+register_user(user_data: UserCreate)`: registra un nuovo utente. Richiede privilegi di amministratore. Invia un'email con una password temporanea all'utente registrato, per farlo utilizza `EmailService`;
- `+get_users() -> List[User]`: restituisce una lista di tutti gli utenti registrati. Richiede privilegi di amministratore;
- `+get_user() -> User`: restituisce i dati dell'utente che effettua la richiesta (utente autenticato);
- `+update_user(user_new_data: UserUpdate)`: aggiorna i dati di un utente specificato. Richiede privilegi di amministratore e la password dell'amministratore che esegue l'operazione;
- `+delete_user(delete_user: UserDelete, admin: UserAuth)`: elimina un utente specificato. Richiede privilegi di amministratore e la password dell'amministratore che esegue l'operazione;

- `+update_password(user_data: UserUpdatePassword)`: aggiorna la password dell'utente autenticato. Richiede la password corrente dell'utente;
- `+reset_password(current_user: User)`: reimposta la password per l'utente identificato tramite l'email. Invia un'email con la nuova password temporanea;

3.3.1.3.5. SettingRouter

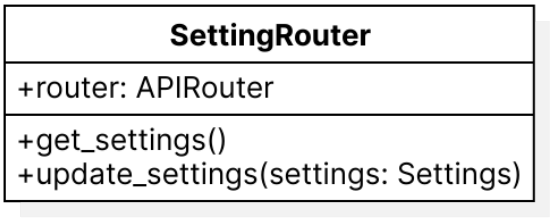

Figura 8: Diagramma delle classi di SettingRouter

La classe `SettingRouter` gestisce le operazioni relative alle impostazioni globali dell'applicazione. Permette di recuperare le impostazioni correnti e di aggiornarle. La lettura delle impostazioni è accessibile pubblicamente, mentre la modifica richiede privilegi di amministratore (`Depends(verify_admin)`). Le operazioni sui dati sono demandate a `SettingRepository`.

3.3.1.3.5.1. Attributi

- `router: APIRouter`: gestisce le chiamate all'oggetto del modulo `fastapi` che permette di definire le API route;

3.3.1.3.5.2. Metodi

- `+get_settings() -> Settings`: restituisce le impostazioni correnti dell'applicazione;
- `+update_settings(settings: Settings)`: aggiorna le impostazioni dell'applicazione. Richiede privilegi di amministratore;

3.3.1.3.6. FAQRouter

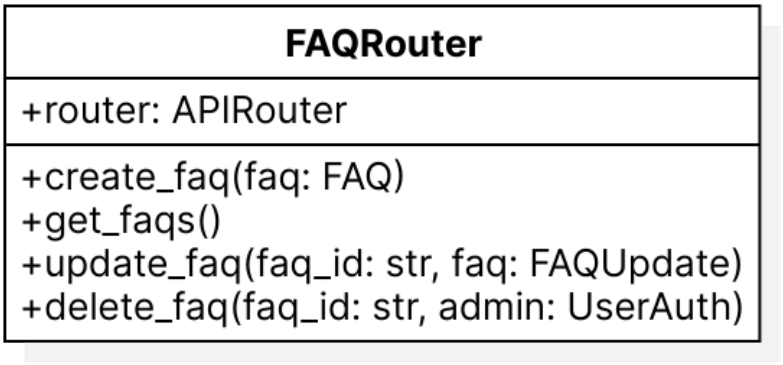

Figura 9: Diagramma delle classi di FaqRouter

La classe `FAQRouter` gestisce le chiamate alle operazioni per l'accesso, la modifica e l'eliminazione delle FAQ. L'accesso per la lettura delle FAQ richiede l'autenticazione dell'utente (`Depends(verify_user)`), mentre le operazioni di creazione, modifica ed eliminazione richiedono privilegi di amministratore (`Depends(verify_admin)`). Interagisce con `FaqRepository` per la persistenza dei dati e con `UserRepository` per la verifica delle credenziali dell'amministratore durante l'eliminazione.

3.3.1.3.6.1. Attributi

- `router: APIRouter`: l'oggetto del modulo `fastapi` che permette di definire le API route;

3.3.1.3.6.2. Metodi

- `+create_faq(faq: FAQ)`: crea una nuova FAQ. Richiede privilegi di amministratore;
- `+get_faqs()`: restituisce una lista di tutte le FAQ. Richiede l'autenticazione dell'utente;
- `+update_faq(faq_id: str, faq: FAQUpdate)`: aggiorna una FAQ esistente, identificata dal suo `faq_id`. Richiede privilegi di amministratore;
- `+delete_faq(faq_id: str, admin: UserAuth)`: elimina una FAQ esistente, identificata dal suo `faq_id`. Richiede privilegi di amministratore e la password dell'amministratore che esegue l'operazione;

3.3.1.3.7. UserRepository

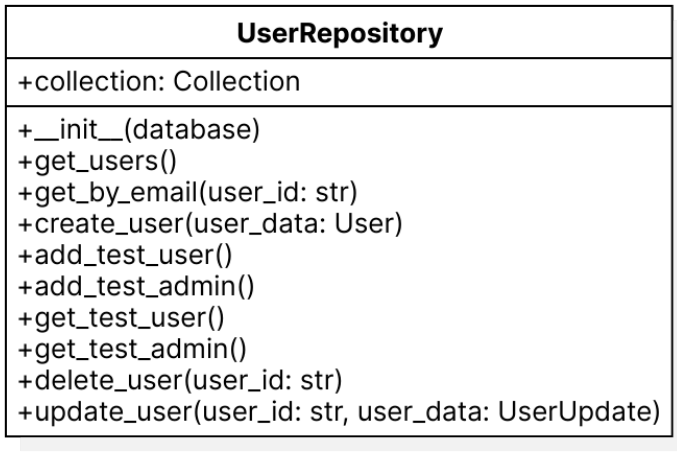

Figura 10: Diagramma delle classi di UserRepository

La classe `UserRepository` è responsabile della gestione delle operazioni CRUD (Create, Read, Update, Delete) per gli utenti nel database. Interagisce con la collezione «users» in MongoDB. L'istanza del database viene ottenuta tramite *Dependency Injection*.

3.3.1.3.7.1. Attributi

- `collection: Collection`: la collezione «users» del database;

3.3.1.3.7.2. Metodi

- `+__init__(database)`: inizializza il repository con una connessione al database;
- `+get_users() -> List[User]`: recupera tutti gli utenti presenti nel database;
- `+get_by_email(user_id: str) -> User`: recupera un utente specifico in base al suo indirizzo email (che funge da `_id`);
- `+create_user(user_data: User)`: inserisce un nuovo utente nel database;
- `+delete_user(user_id: str)`: elimina un utente dal database in base al suo indirizzo email. Solleva un'eccezione `HTTPException` se l'utente non viene trovato o se si verifica un errore durante l'eliminazione;
- `+update_user(user_id: str, user_data: schemas.UserUpdate) -> UpdateResult`: aggiorna i dati di un utente esistente. Solleva un'eccezione `HTTPException` se l'utente non viene trovato, se non vengono forniti dati per l'aggiornamento, o se i dati forniti corrispondono a quelli esistenti (nessuna modifica);
- `+add_test_user()`: (Metodo di utilità) Aggiunge un utente di test predefinito al database;

- `+add_test_admin()`: (Metodo di utilità) Aggiunge un utente amministratore di test predefinito al database, utilizzando variabili d'ambiente per le credenziali se disponibili;
- `+get_test_user() -> User`: (Metodo di utilità) Recupera l'utente di test;
- `+get_test_admin() -> User`: (Metodo di utilità) Recupera l'utente amministratore di test;

3.3.1.3.8. ChatRepository

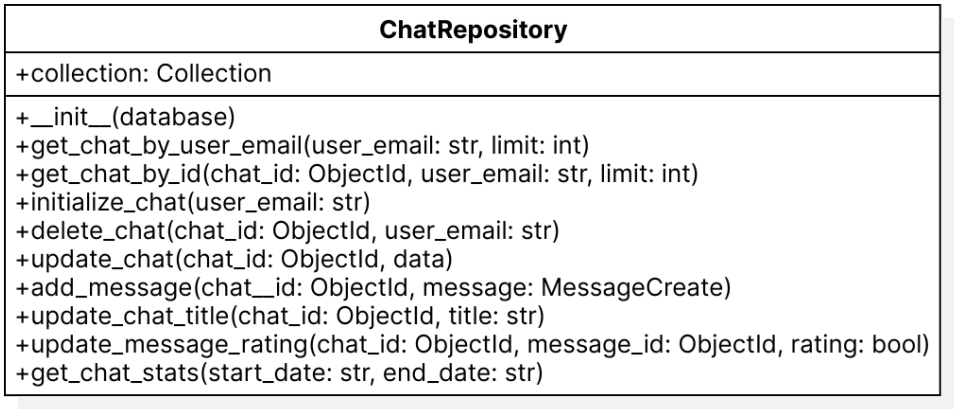

Figura 11: Diagramma delle classi di ChatRepository

La classe `ChatRepository` gestisce tutte le operazioni relative alle chat e ai messaggi nel database, interagendo con la collezione «chats» in MongoDB. L'istanza del database viene ottenuta tramite *Dependency Injection*.

3.3.1.3.8.1. Attributi

- `collection: Collection`: la collezione «chats» del database;

3.3.1.3.8.2. Metodi

- `+get_chat_by_user_email(user_email: str, limit: int = 100) -> List[Chat]`: recupera tutte le chat associate a un indirizzo email specifico, con un limite opzionale sul numero di chat restituite;
- `+get_chat_by_id(chat_id: str, user_email: str, limit: int = 100) -> Chat`: recupera una chat specifica in base al suo ID e all'email dell'utente. È possibile specificare un limite per il numero di messaggi da includere nella chat restituita (gli ultimi `limit` messaggi);
- `+initialize_chat(user_email: str) -> Chat`: crea una nuova chat per l'utente specificato, inizializzandola con un messaggio di benvenuto predefinito dal bot. Il nome della chat è impostato a «Chat senza nome» e viene registrato il timestamp di creazione;
- `+delete_chat(chat_id: str, user_email: str)`: elimina una chat specifica in base al suo ID e all'email dell'utente;

- `+update_chat(chat_id: str, data: dict)`: aggiorna i dati di una chat esistente;
- `+add_message(chat_id: str, message: schemas.MessageCreate) -> dict`: aggiunge un nuovo messaggio a una chat esistente. Il messaggio include un ID univoco, il mittente, il contenuto, un timestamp e una valutazione iniziale (None). Restituisce i dati del messaggio aggiunto;
- `+update_chat_title(chat_id: str, title: str)`: aggiorna il titolo di una chat specifica;
- `+update_message_rating(chat_id: ObjectId, message_id: ObjectId, rating: bool)`: aggiorna la valutazione (positiva/negativa) di un messaggio specifico all'interno di una chat, solo se il messaggio è stato inviato dal bot;
- `+get_chat_stats(start_date: Optional[str] = None, end_date: Optional[str] = None) -> Stats`: calcola e restituisce statistiche aggregate sull'utilizzo delle chat. È possibile filtrare le statistiche per un intervallo di date. Le statistiche includono: numero totale di chat, numero totale di messaggi, messaggi inviati dal chatbot, messaggi valutati, messaggi inviati dagli utenti, media messaggi per utente, media messaggi per chat, percentuale di valutazioni positive e numero di utenti attivi;

3.3.1.3.9. DocumentRepository

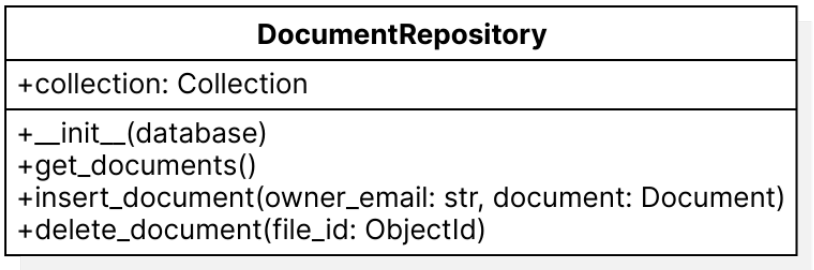

Figura 12: Diagramma delle classi di DocumentRepository

La classe `DocumentRepository` è responsabile della gestione delle operazioni CRUD per i metadati dei documenti (come titolo, percorso del file, proprietario e data di caricamento) nel database. Interagisce con la collezione «documents» in MongoDB. L'istanza del database viene ottenuta tramite *Dependency Injection*.

3.3.1.3.9.1. Attributi

- `collection: Collection`: la collezione «documents» del database;

3.3.1.3.9.2. Metodi

- `+get_documents()`: recupera tutti i documenti;
- `+insert_document(owner_email: str, document: Document)`: inserisce le informazioni di un nuovo documento nel database;

- `+delete_document(file_id: str)`: elimina le informazioni di un documento dal database in base al suo ID;

3.3.1.3.10. SettingRepository

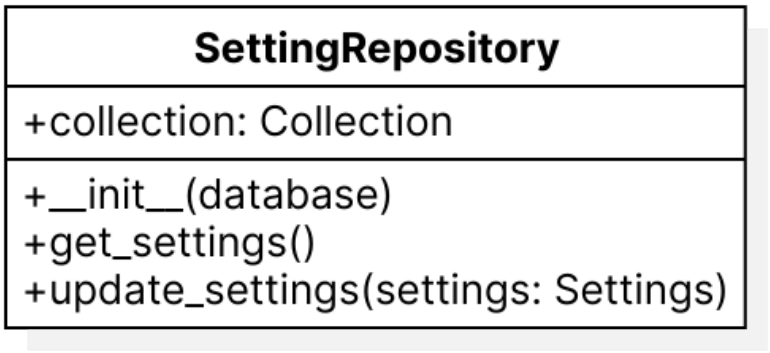

Figura 13: Diagramma delle classi di SettingRepository

La classe `SettingRepository` gestisce le impostazioni di personalizzazione globali della piattaforma. Interagisce con la collezione «settings» in MongoDB. L'istanza del database viene ottenuta tramite *Dependency Injection*. Al momento dell'inizializzazione, crea le impostazioni predefinite se non esistono;

3.3.1.3.10.1. Attributi

- `collection: Collection`: la collezione «settings» del database;

3.3.1.3.10.2. Metodi

- `+get_settings() -> schemas.Settings`: recupera le impostazioni globali dell'applicazione;
- `+update_settings(settings: Settings)`: aggiorna le impostazioni globali dell'applicazione. Solleva un'eccezione `HTTPException` se le impostazioni non vengono trovate o se i dati forniti corrispondono a quelli esistenti;

3.3.1.3.11. FaqRepository

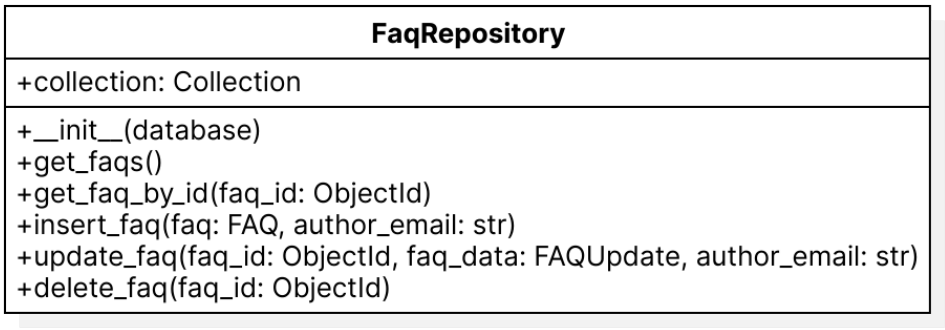

Figura 14: Diagramma delle classi di FaqRepository

La classe `FaqRepository` è responsabile della gestione delle operazioni CRUD per le FAQ (Frequently Asked Questions) nel database. Interagisce con la collezione «faq» in MongoDB. L'istanza del database viene ottenuta tramite *Dependency Injection*.

3.3.1.3.11.1. Attributi

- `collection: Collection`: la collezione «faq» del database;

3.3.1.3.11.2. Metodi

- `+get_faqs()`: recupera tutte le FAQ presenti nel database;
- `+get_faq_by_id(faq_id: ObjectId)`: recupera una FAQ specifica in base al suo ID;
- `+insert_faq(faq: FAQ, author_email: str) -> ObjectId`: inserisce una nuova FAQ nel database, associandola all'email dell'autore e registrando timestamp di creazione e aggiornamento. Restituisce l'ID della FAQ inserita;
- `+update_faq(faq_id: ObjectId, faq_data: FAQUpdate, author_email: str)`: aggiorna una FAQ esistente. Solleva un'eccezione `HTTPException` se la FAQ non viene trovata o se i dati forniti corrispondono a quelli esistenti. Aggiorna l'email dell'autore e il timestamp di aggiornamento;
- `+delete_faq(faq_id: ObjectId)`: elimina una FAQ dal database in base al suo ID;

3.3.1.3.12. EmailService

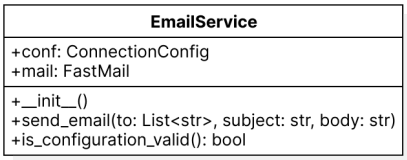

Figura 15: Diagramma delle classi di EmailService

La classe `EmailService` è responsabile dell'invio di email. Utilizza la libreria `fastapi-mail` e configura i parametri di connessione al server SMTP tramite variabili d'ambiente.

3.3.1.3.12.1. Attributi

- `-conf: ConnectionConfig`: oggetto di configurazione per `fastapi-mail` contenente i dettagli del server SMTP;
- `-mail: FastMail`: istanza di `FastMail` utilizzata per inviare i messaggi;

3.3.1.3.12.2. Metodi

- `+__init__()`: costruttore della classe, inizializza `conf` leggendo le variabili d'ambiente (con valori di default) e l'oggetto `mail`;
- `+send_email(to: List[str], subject: str, body: str) -> None`: invia un'email ai destinatari specificati. Solleva un'eccezione `ValueError` se la configurazione dell'email non è valida;
- `+is_configuration_valid() -> bool`: controlla se le variabili d'ambiente necessarie per l'invio delle email sono state configurate;

3.3.1.3.13. Database

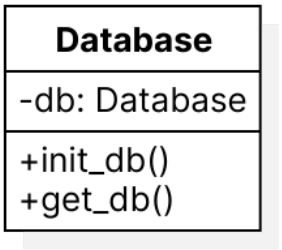

Figura 16: Diagramma delle classi di Database

La classe `Database` è responsabile della gestione della connessione al database MongoDB. Utilizza il pattern Singleton per garantire che esista una singola istanza di connessione al database durante l'intero ciclo di vita dell'applicazione.

3.3.1.3.13.1. Attributi

- `-db`: istanza della connessione al database.

3.3.1.3.13.2. Metodi

- `+init_db()`: Inizializza la connessione al database. Questa funzione deve essere chiamata all'avvio dell'applicazione;
- `+get_db() -> AsyncIOMotorDatabase`: Restituisce l'istanza del database precedentemente inizializzata. Se `init_db()` non è stata ancora chiamata (e quindi `db` è `None`), solleva una `RuntimeError` per indicare che il database non è pronto per l'uso. Questo meccanismo assicura che tutte le parti dell'applicazione accedano alla stessa istanza di connessione al database; (**Singleton**)

3.3.2. LLM API

LLM-API è il nome del API backend che si occupa della gestione dell'interazione con l'LLM. Per l'interazione con gli LLM sono state importate diverse librerie, tra queste quella principale è ***LangChain***.

LangChain è una libreria progettata per semplificare l'integrazione e l'interazione con diversi LLM. Fornisce strumenti e astrazioni per costruire applicazioni che utilizzano LLM, facilitando l'interazione con i modelli, la gestione del contesto e la generazione di prompt. Inoltre semplifica l'intercambiabilità tra i diversi modelli e provider.

3.3.2.1. Struttura del codice

Il codice di LLM-API è strutturato in ***router*** e ***service***.

3.3.2.1.1. Router

I router sono responsabili della gestione delle richieste HTTP e dell'instradamento delle stesse alle funzioni appropriate. Ogni router è dedicato a una delle seguenti funzionalità specifica del sistema:

- `document.py`;
- `faq.py`;
- `llm.py`;

3.3.2.1.2. Service

I service sono delle classi che rappresentano la business logic principale; queste gestiscono i seguenti aspetti dell'interazione con l'LLM:

- `embeddings_service.py`: fornisce i metodi per vettorializzare il contesto grezzo fornito dall'utente, trattando gli embedding provider in modo generico;
- `file_manager_service.py`: fornisce i metodi per gestire e manipolare i diversi tipi di file forniti dall'utente, trattandoli in modo generico;

- `llm_response_service.py`: fornisce i metodi per ottenere le risposte generate dall'LLM in base al contesto e prompt forniti;
- `llm_service.py`: permette di operare con diversi LLM e provider di LLM rendendoli facilmente intercambiabili;
- `vector_database_service.py`: fornisce i metodi per interagire con il database vettoriale, permettendo di memorizzare e recuperare i dati vettorializzati, semplificando l'intercambiabilità tra diversi database vettoriali;

3.3.2.2. Design pattern utilizzati

3.3.2.2.1. Strategy

Il pattern ***strategy*** è un pattern comportamentale che consente di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Questo pattern permette di separare l'algoritmo dalla sua implementazione, consentendo di modificare il comportamento del sistema senza alterare il codice esistente.

Abbiamo utilizzato questo pattern per:
- dare la possibilità di implementare e utilizzare diversi provider di LLM e diversi LLM;
- dare la possibilità di utilizzare diversi provider di funzioni di embedding;
- dare la possibilità di utilizzare database vettoriali diversi;
- dare la possibilità di estendere i diversi tipi di file gestiti dal sistema.

3.3.2.2.2. Singleton

Il pattern ***singleton*** è un pattern creazionale che garantisce che una classe abbia una sola istanza e fornisce un punto di accesso globale a essa. Questo pattern è utile quando è necessario controllare l'accesso a una risorsa condivisa, come un database o un file di configurazione.

Abbiamo utilizzato il pattern singleton per gestire l'accesso al database vettoriale in modo che la stessa istanza del database possa essere utilizzata in tutta l'applicazione, evitando conflitti e garantendo la coerenza dei dati.

3.3.2.2.3. Dependency Injection

Il pattern ***dependency injection*** è un pattern strutturale che consente di includere le dipendenze necessarie in una classe, invece che crearle all'interno della classe stessa. Questo pattern permette di ridurre l'accoppiamento tra le classi e di rendere il codice più modulare.

3.3.2.3. Diagramma delle classi

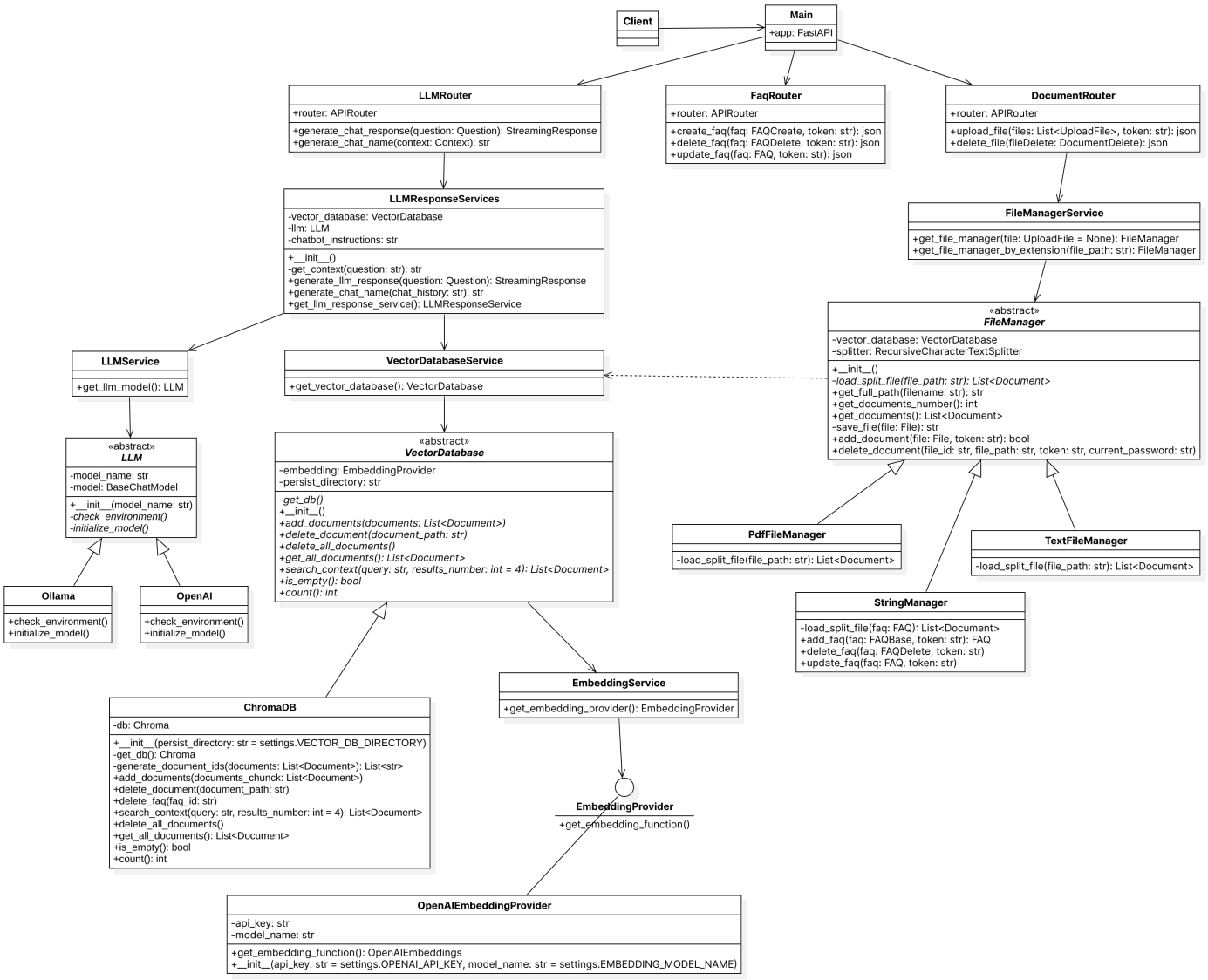

Figura 17: Diagramma delle classi di LLM-API

3.3.2.3.1. DTO

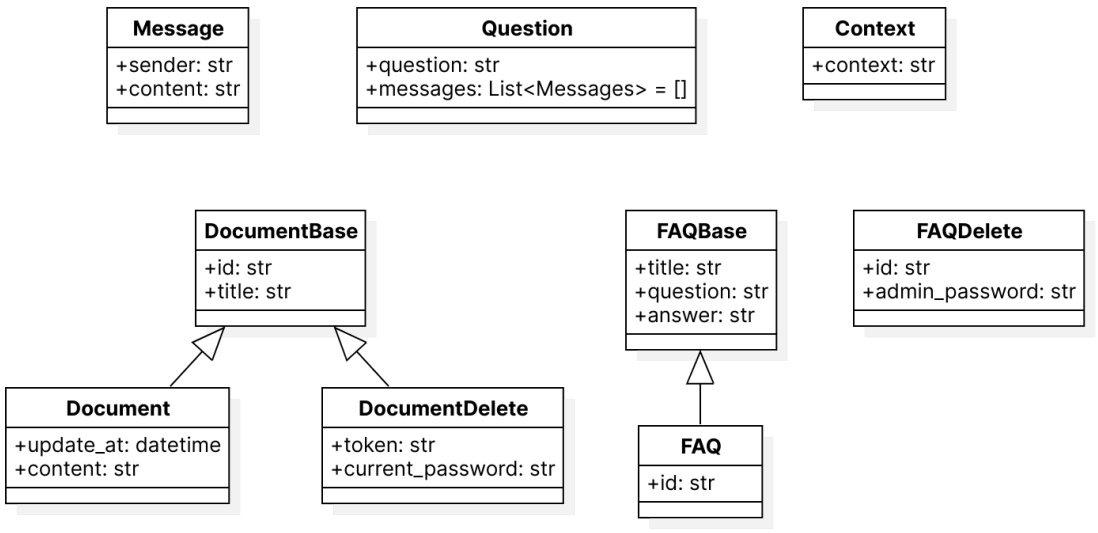

Figura 18: Diagramma delle classi dei DTO di LLM-API

I DTO (Data Transfer Object) sono oggetti utilizzati per trasferire dati tra le diverse parti del sistema, in particolare tra il client e il server LLM-API. Semplificano la comunicazione e la validazione dei dati.

3.3.2.3.1.1. Message

- `+sender: str`: rappresenta il mittente del messaggio, che può essere l'utente o il chatbot;
- `+content: str`: rappresenta il contenuto del messaggio;

3.3.2.3.1.2. Question

- `+question: str`: rappresenta la domanda posta dall'utente al chatbot;
- `+messages: List[Messages]`: rappresenta il contesto fornito dall'utente al chatbot;

3.3.2.3.1.3. Context

- `+context: str`: rappresenta il contesto fornito dall'utente al chatbot;

3.3.2.3.1.4. DocumentBase

- `+id: str`: rappresenta l'identificativo del documento;
- `+title: str`: rappresenta il nome del documento;

3.3.2.3.1.5. Document

Generalizzazione della classe DocumentBase.
- `+update_at: datetime`: rappresenta la data di aggiornamento del documento;
- `+content: str`: rappresenta il contenuto del documento;

3.3.2.3.1.6. DocumentDelete

Generalizzazione della classe DocumentBase.
- `+token: str`: rappresenta il token JWT di autenticazione dell'utente;
- `+current_password: str`: rappresenta la password dell'utente;

3.3.2.3.1.7. FAQBase

- `+title: str`: rappresenta il titolo della FAQ;
- `+question: str`: rappresenta la domanda della FAQ;
- `+answer: str`: rappresenta la risposta della FAQ;

3.3.2.3.1.8. FAQ

Generalizzazione della classe FAQBase.
- `+id: str`: rappresenta l'identificativo della FAQ;

3.3.2.3.1.9. FAQDelete

- `+id: str`: rappresenta l'identificativo della FAQ;
- `+admin_password: str`: rappresenta la password dell'utente;

3.3.2.3.2. LLMRouter

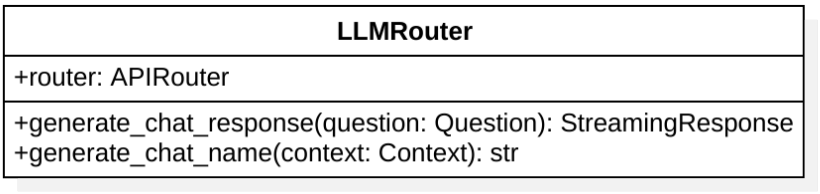

Figura 19: Diagramma delle classi di LLMRouter

La classe `LLMRouter` è responsabile della gestione delle richieste HTTP relative alle query che verranno reindirizzate all'LLMResponseService.

L'oggetto LLMResponseService viene importato tramite il metodo `LLMResponseService.get_llm_response_service()` che restituisce l'istanza del servizio LLMResponseService (**Dependency Injection**).

3.3.2.3.2.1. Attributi

- `+router: APIRouter`: oggetto del modulo fastapi che permette di definire le API route;

3.3.2.3.2.2. Metodi

- `+generate_chat_response(question: Question) -> StreamingResponse`: restituisce la risposta generata, in base al prompt fornito, tramite uno stream di dati; per ottenere la risposta generata viene chiamato il metodo `LLMResponseService.generate_llm_response()`;
- `+generate_chat_name(context: Context) -> str`: restituisce il nome della chat generato in base al contesto fornito; per ottenere il nome generato viene chiamato il metodo `LLMResponseService.generate_llm_chat_name()`;

3.3.2.3.3. FaqRouter

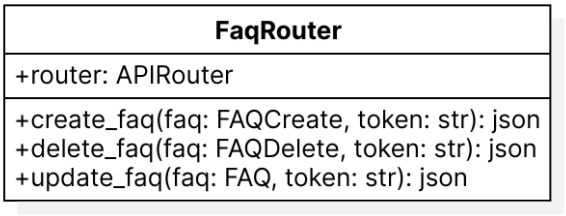

Figura 20: Diagramma delle classi di FaqRouter

La classe `FaqRouter` è responsabile della gestione delle richieste HTTP relative a salvataggio, modifica e cancellazione delle FAQ nel database vettoriale.
I metodi della classe FileManager vengono utilizzati tramite l'oggetto restituito dal metodo `FileManagerService.get_file_manager_by_extension()` (**Dependency Injection**).

3.3.2.3.3.1. Attributi

- `+router: APIRouter`: oggetto del modulo fastapi che permette di definire le API route;

3.3.2.3.3.2. Metodi

- `+create_faq(faq: FAQBase, token: str) -> json`: chiama il metodo `FileManager.add_faq()`, il token viene utilizzato per verificare che l'utente che esegue la richiesta sia un admin; restituisce i dati della faq creata e l'esito della richiesta;

- `+delete_faq(faq: FAQDelete) -> json`: chiama il metodo `FileManager.delete_faq()`, il token viene utilizzato per verificare che l'utente che esegue la richiesta sia un admin; restituisce l'esito della richiesta;
- `+update_faq(faq: FAQBase) -> json`: chiama il metodo `FileManager.update_faq()`, il token viene utilizzato per verificare che l'utente che esegue la richiesta sia un admin; restituisce i dati della faq aggiornata e l'esito della richiesta;

3.3.2.3.4. DocumentRouter

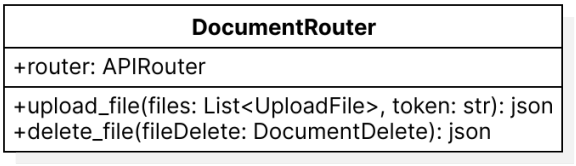

Figura 21: Diagramma delle classi di DocumentRouter

La classe `DocumentRouter` è responsabile della gestione delle richieste HTTP relative a salvataggio, modifica e cancellazione dei documenti nel database vettoriale.
I metodi della classe FileManager vengono utilizzati tramite l'oggetto restituito dal metodo `FileManagerService.get_file_manager_by_extension()` (**Dependency Injection**).

3.3.2.3.4.1. Attributi

- `+router: APIRouter`: oggetto del modulo fastapi che permette di definire le API route;

3.3.2.3.4.2. Metodi

- `+upload_file(files: List[UploadFile], token: str) -> json`: chiama il metodo `FileManager.add_document()` per ogni file contenuto nella lista files, il token viene utilizzato per verificare che l'utente che esegue la richiesta sia un admin; restituisce l'esito della richiesta;
- `+delete_file(fileDelete: DocumentDelete) -> json`: chiama il metodo `FileManager.delete_document()`, il token viene utilizzato per verificare che l'utente che esegue la richiesta sia un admin; restituisce l'esito della richiesta;

3.3.2.3.5. LLMResponseService

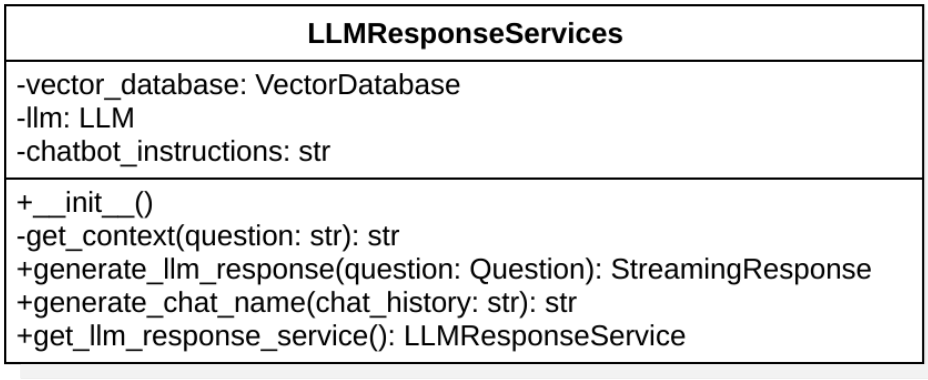

Figura 22: Diagramma delle classi di LLMResponseService

La classe `LLMResponseService` si occupa di orchestrare l'ottenimento del contesto, la preparazione della richiesta per l'LLM e, infine, il ritorno della risposta generata.

3.3.2.3.5.1. Attributi

- `-vector_database: VectorDatabase`: oggetto VectorDatabase che permette di recuperare il contesto; l'oggetto viene istanziato al momento della costruzione dell'LLMResponseService e viene ottenuto tramite il metodo `VectorDatabaseService.get_vector_database()` (**Dependency Injection**);
- `-llm: LLM`: oggetto LLM che permette di generare la risposta; l'oggetto viene istanziato al momento della costruzione dell'LLMResponseService e viene ottenuto tramite il metodo `LLMService.get_llm()` (**Dependency Injection**);
- `-chatbot_instruction: str`: stringa che contiene il prompt iniziale con le istruzioni per il chatbot;

3.3.2.3.5.2. Metodi

- `+__init__()`: costruttore della classe, inizializza gli oggetti vector_database, llm e il prompt iniziale;
- `-get_context(question: str) -> str`: restituisce il contesto in base alla domanda fornita; per ottenere il contesto viene chiamato il metodo `VectorDatabase.search_context()`;
- `+generate_llm_response(question: Question) -> StreamingResponse`: tramite uno stream di dati restituisce la risposta, generata in base al prompt fornito;
  per ottenere il contesto viene chiamato il metodo `LLMResponseService.get_context()` e per ottenere la risposta generata viene chiamato il metodo `LLM.generate_response()`;

- `+generate_llm_chat_name(chat_history: str) -> str`: restituisce il nome della chat generato in base al contesto fornito;
  per ottenere il nome generato viene chiamato il metodo `LLM.generate_chat_name()` a cui viene passato lo storico della chat e un prompt standard che fornisce le istruzioni per l'LLM;
- `+get_llm_response_service() -> LLMResponseService`: restituisce l'istanza del servizio LLMResponseService;

3.3.2.3.6. LLMService

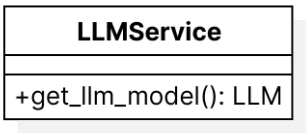

Figura 23: Diagramma delle classi di LLMService

La classe `LLMService` permette di ottenere l'istanza dell'oggetto LLM adatto ad interagire con l'LLM specifico scelto dall'amministratore della piattaforma.

3.3.2.3.6.1. Metodi

- `+get_llm_model() -> LLM`: restituisce l'oggetto LLM adatto basandosi sulle variabili d'ambiente LLM_MODEL e LLM_PROVIDER;

3.3.2.3.7. LLM

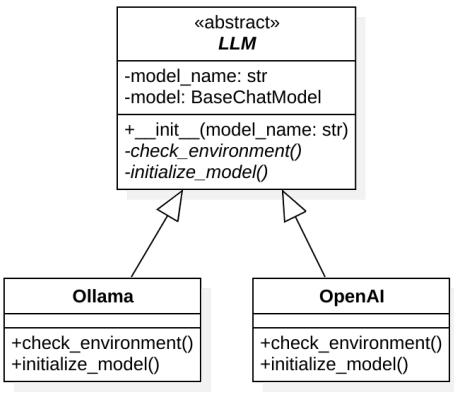

Figura 24: Diagramma delle classi di LLM

La classe `LLM` è un'astrazione che rappresenta un LLM generico. Lo scopo è quello di inizializzare l'oggetto LLM in modo che, le funzionalità di langchain per interagire con i modelli, siano utilizzabili in modo generico, quindi senza dipendere dall'LLM scelto.

3.3.2.3.7.1. Attributi

- `-model_name: str`: nome del modello LLM;
- `-model: BaseChatModel`: oggetto della classe `langchain_core.language_models.chat_models.BaseChatModel` che permette di invocare il modello LLM;

3.3.2.3.7.2. Metodi

- `+__init__(model_name: str)`: costruttore della classe, inizializza gli oggetti model_name e model, il secondo viene inizializzato tramite il metodo `initialize_model()`; controlla che le variabili d'ambiente necessarie siano impostate correttamente tramite il metodo `check_environment()`;
- `-check_environment()`: controlla che le variabili d'ambiente necessarie siano impostate correttamente;
- `-initialize_model() -> BaseChatModel`: inizializza l'oggetto model in base al modello scelto;

3.3.2.3.8. Ollama

Classe concreta della classe LLM che permette di utilizzare `Ollama` come provider di LLM.

3.3.2.3.8.1. Metodi

- `-check_environment()`: controlla che le variabili d'ambiente necessarie all'utilizzo del modello scelto di Ollama siano impostate correttamente;
- `-initialize_model() -> BaseChatModel`: inizializza l'oggetto model per essere utilizzato con il modello scelto di Ollama;

3.3.2.3.9. OpenAI

Classe concreta della classe LLM che permette di utilizzare `OpenAI` come provider di LLM.

3.3.2.3.9.1. Metodi

- `-check_environment()`: controlla che le variabili d'ambiente necessarie all'utilizzo del modello scelto di OpenAI siano impostate correttamente;
- `-initialize_model() -> BaseChatModel`: inizializza l'oggetto model per essere utilizzato con il modello scelto di OpenAI;

3.3.2.3.10. VectorDatabaseService

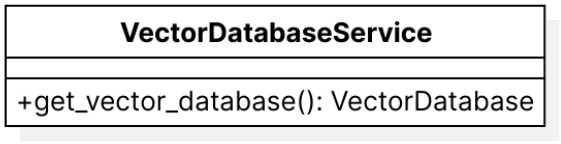

Figura 25: Diagramma delle classi di VectorDatabaseService

La classe `VectorDatabaseService` permette di ottenere l'istanza dell'oggetto VectorDatabase adatto ad interagire con il database vettoriale scelto dall'amministratore della piattaforma.

3.3.2.3.10.1. Metodi

- `+get_vector_database() -> VectorDatabase`: restituisce l'oggetto VectorDatabase adatto basandosi sulla variabile d'ambiente VECTOR_DATABASE_PROVIDER;

3.3.2.3.11. VectorDatabase

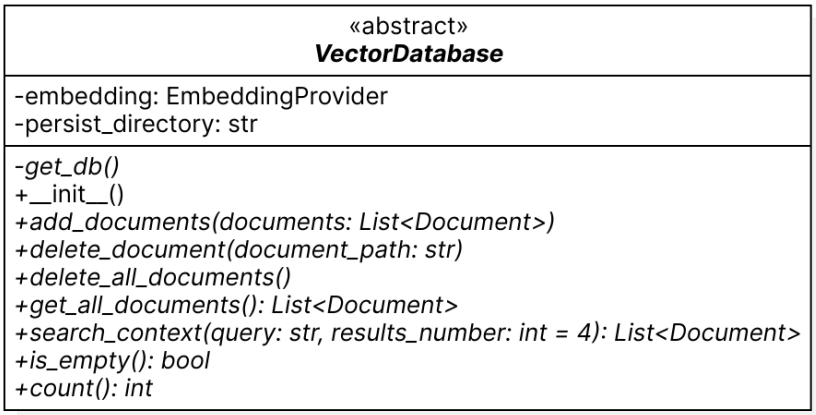

Figura 26: Diagramma delle classi di VectorDatabase

La classe `VectorDatabase` è un'astrazione che rappresenta un database vettoriale generico. Lo scopo è quello di inizializzare oggetti delle classi concrete, figlie di VectorDatabase, in modo che le funzionalità di langchain per interagire con i database vettoriali siano utilizzabili in modo generico, quindi senza dipendere dal database scelto (**Strategy**).

3.3.2.3.11.1. Attributi

- `-embedding: EmbeddingProvider`: oggetto della classe EmbeddingProvider che permette di accedere diverse funzioni di embedding in base all'embedding provider scelto; l'attributo viene inizializzato tramite il metodo `EmbeddingService.get_embedding_provider()` (**Dependency Injection**);
- `-persist_directory: str`: percorso della directory in cui sono memorizzati i dati vettorializzati; viene inizializzata tramite la variabile d'ambiente VECTOR_DB_DIRECTORY;

3.3.2.3.11.2. Metodi

- `+__init__()`: costruttore della classe, inizializza gli oggetti embedding e persist_directory;
- `-get_db()`: restituisce l'istanza del database vettoriale contenuta nell'oggetto VectorDatabase (**Singleton**);
- `+add_documents(documents: List<Document>)`: aggiunge i documenti passati al database vettoriale;
- `+delete_document(document_path: str)`: elimina il documento, che si trova al percorso passato, dal database vettoriale;
- `+delete_all_documents()`: elimina tutti i documenti dal database vettoriale;
- `+get_all_documents() -> List[Document]`: restituisce tutti i documenti presenti nel database vettoriale;
- `+search_context(query: str, results_number: int = 4) -> List[Document]`: restituisce il contesto in base alla domanda fornita;
- `+is_empty() -> bool`: restituisce True se il database vettoriale è vuoto, False altrimenti;
- `+count() -> int`: restituisce il numero di documenti presenti nel database vettoriale;

3.3.2.3.12. ChromaDB

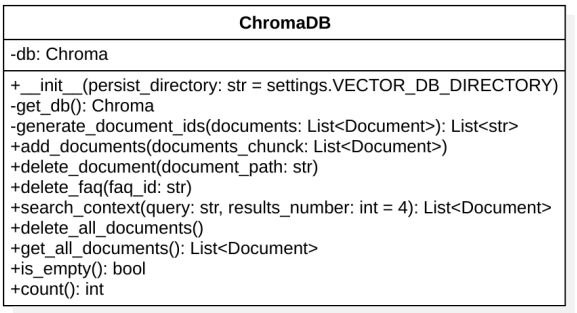

Figura 27: Diagramma delle classi di ChromaDB

La classe `ChromaDB` è un'implementazione concreta della classe VectorDatabase che permette di utilizzare ChromaDB come database vettoriale.

3.3.2.3.12.1. Attributi

- `-db: Chroma`: oggetto della classe `langchain_chroma.vectorstores.Chroma` che permette di interagire con il database vettoriale ChromaDB;

3.3.2.3.12.2. Metodi

- `+__init__(persist_directory: str = settings.VECTOR_DB_DIRECTORY)`: inizializza gli oggetti embedding, persist_directory utilizzando la variabile d'ambiente VECTOR_DB_DIRECTORY e db; l'oggetto embedding viene inizializzato tramite il metodo `EmbeddingService.get_embedding_provider()` (**Dependency Injection**);
- `-get_db() -> Chroma`: restituisce l'istanza del database vettoriale ChromaDB contenuta nell'oggetto campo db della classe ChromaDB (**Singleton**);
- `-generate_document_ids(documents: List[Document]) -> List[str]`: ritorna gli identificativi dei documenti da aggiungere al database vettoriale; gli identificativi vengono generati tramite il metodo `uuid.uuid3()` in modo che il contenuto del documento sia univoco.
- `+add_documents(documents: List[Document])`: aggiunge i documenti passati al database vettoriale ChromaDB; per ogni documento viene generato un identificativo tramite il metodo `ChromaDB.generate_document_ids()` e viene aggiunto al database vettoriale tramite il metodo `langchain_chroma.vectorstores.Chroma.add_documents()`;
- `+delete_document(document_path: str)`: elimina il documento, che si trova al percorso passato, dal database vettoriale ChromaDB; per eliminare il documento viene utilizzato il metodo `langchain_chroma.vectorstores.Chroma.delete()`;
- `+delete_faq(faq_id: str)`: elimina la FAQ, che si trova all'interno del database vettoriale ChromaDB, con l'id passato come parametro; per eliminare la FAQ viene utilizzato il metodo `langchain_chroma.vectorstores.Chroma.delete()`;
- `+search_context(query: str, results_number: int = 4) -> List[Document]`: restituisce il contesto in base alla domanda fornita; per ottenere il contesto viene utilizzato il metodo `langchain_chroma.vectorstores.Chroma.similarity_search()` che restituisce i documenti più simili alla query passata come parametro;
- `+delete_all_documents()`: elimina tutti i documenti dal database vettoriale ChromaDB; per eliminare tutti i documenti viene utilizzato il metodo `langchain_chroma.vectorstores.Chroma.reset_collection()`;
- `+get_all_documents() -> List[Document]`: restituisce tutti i documenti presenti nel database vettoriale ChromaDB; per ottenere tutti i documenti viene utilizzato il metodo `langchain_chroma.vectorstores.Chroma.get()` che restituisce tutti i documenti presenti nel database vettoriale;
- `+is_empty() -> bool`: restituisce True se il database vettoriale ChromaDB è vuoto, False altrimenti;
- `+count() -> int`: restituisce il numero di documenti presenti nel database vettoriale ChromaDB;

3.3.2.3.13. EmbeddingService

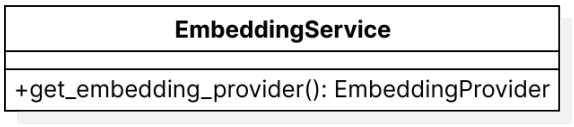

Figura 28: Diagramma delle classi di EmbeddingService

La classe `EmbeddingService` permette di ottenere un'istanza del provider configurato tramite le impostazioni, semplificando l'integrazione e l'intercambiabilità tra diversi provider.

3.3.2.3.13.1. Metodi

- `+get_embedding_function()`: metodo astratto che restituisce la funzione di embedding fornita dal provider;

3.3.2.3.14. EmbeddingProvider

Figura 29: Diagramma delle classi di EmbeddingProvider

La classe `EmbeddingProvider` è un'astrazione che rappresenta un provider di embedding generico. Lo scopo è quello di inizializzare oggetti delle classi concrete, figlie di EmbeddingProvider, in modo che le funzionalità di langchain per interagire con i provider di embedding siano utilizzabili in modo generico, quindi senza dipendere dal provider scelto (**Strategy**).

3.3.2.3.14.1. Metodi

- `+get_embedding_provider() -> EmbeddingProvider`: restituisce un'istanza del provider di embedding configurato;

3.3.2.3.15. OpenAIEmbeddingProvider

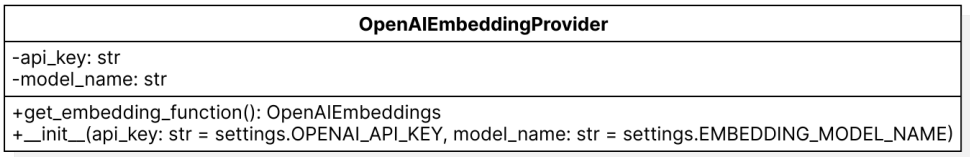

Figura 30: Diagramma delle classi di OpenAIEmbeddingProvider

La classe `OpenAIEmbeddingProvider` è un'implementazione concreta della classe EmbeddingProvider che utilizza le API di OpenAI per fornire funzioni di embedding. Questa classe consente di utilizzare le funzioni di embedding che sfruttano modelli di OpenAI.

3.3.2.3.15.1. Attributi

- `-api_key: str`: chiave API utilizzata per autenticarsi con il servizio OpenAI;
- `-model_name: str`: nome del modello di embedding utilizzato;

3.3.2.3.15.2. Metodi

- `+__init__(api_key: str, model_name: str)`: costruttore della classe, inizializza gli attributi api_key tramite la variabile d'ambiente OPENAI_API_KEY e model_name tramite la variabile d'ambiente EMBEDDING_MODEL_NAME;
- `+get_embedding_function() -> OpenAIEmbeddings`: restituisce la funzione di embedding fornita da OpenAI;

3.3.2.3.16. FileManagerService

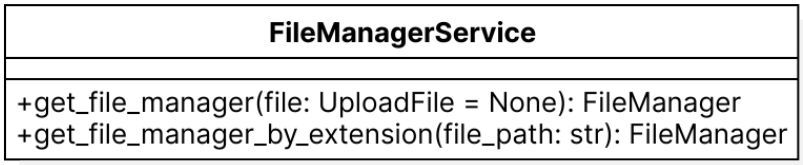

Figura 31: Diagramma delle classi di FileManagerService

La classe `FileManagerService` permette di ottenere l'istanza dell'oggetto FileManager adatto ad interagire con il file manager scelto dall'amministratore della piattaforma.

3.3.2.3.16.1. Metodi

- `+get_file_manager(file: UploadFile) -> FileManager`: restituisce l'oggetto FileManager adatto basandosi sul tipo di file passato come parametro; se non esiste un file manager per il tipo di file passato, viene sollevata un'eccezione; Se invece non viene passato un file, viene restituito lo StringManager;
- `+get_file_manager_by_extension(file_extension: str) -> FileManager`: restituisce l'oggetto FileManager adatto basandosi sull'estensione del file passato come parametro; se non esiste un file manager per l'estensione passata, viene sollevata un'eccezione;

3.3.2.3.17. FileManager

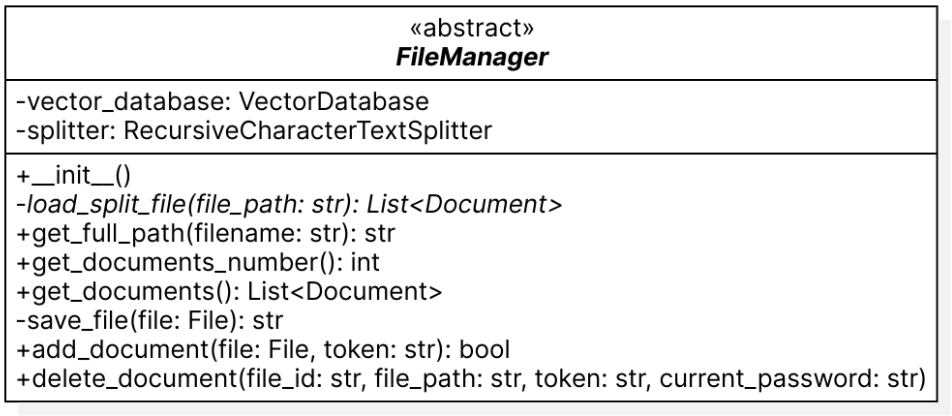

Figura 32: Diagramma delle classi di FileManager

La classe `FileManager` è un'astrazione che rappresenta un file manager generico. Lo scopo è quello di inizializzare oggetti delle classi concrete, figlie di FileManager, in modo che le funzionalità di langchain per interagire con i file siano utilizzabili in modo generico, quindi senza dipendere dal tipo di file scelto (**Pattern Strategy**).

3.3.2.3.17.1. Attributi

- `-vector_database: VectorDatabase`: oggetto VectorDatabase che permette di interagire con il database vettoriale; l'oggetto viene istanziato al momento della costruzione del FileManager e viene ottenuto tramite il metodo `VectorDatabaseService.get_vector_database()` (**Dependency Injection**);
- `-splitter: RecursiveCharacterTextSplitter`: oggetto della classe `langchain_core.text_splitter.RecursiveCharacterTextSplitter` che permette di dividere il testo in chunck di dimensioni stabilite; l'oggetto viene inizializzato al momento della costruzione del FileManager;

3.3.2.3.17.2. Metodi

- `+__init__()`: costruttore della classe, inizializza gli oggetti vector_database e splitter;
- `-load_split_file(file_path:str) -> List[Document]`: metodo astratto che carica il file dal percorso passato come parametro, lo divide in chunck e restituisce una lista di Document;
- `+get_full_path(filename:str) -> str`: restituisce il percorso completo del file passato come parametro, unendo il percorso della directory persistente e il nome del file;
- `+get_documents_number() -> int`: restituisce il numero di documenti presenti nel database vettoriale;
- `+get_documents() -> List[Document]`: restituisce tutti i documenti presenti nel database vettoriale;
- `-save_file(file: File) -> str`: metodo astratto che salva il file passato come parametro nella directory persistente e restituisce il percorso completo del file salvato;
- `+add_document(file: UploadFile, token:str) -> bool`: aggiunge il file passato come parametro al database vettoriale; per farlo, chiama il metodo `FileManager.save_file()` per salvare il file nella directory persistente, chiama il metodo `FileManager.load_split_file()` che suddivide il file in chunck e poi chiama il metodo `VectorDatabase.add_documents()` per aggiungere i chunck al database vettoriale; poi effettua una richiesta POST a *Database-API* per salvare le informazioni del documento nel database; restituisce True se l'operazione è andata a buon fine, False altrimenti;
- `+delete_document(file_id: str, file_path:str, token:str, current_password:str) -> bool`: elimina il documento passato come parametro dal database vettoriale; per farlo, chiama il metodo `VectorDatabase.delete_document()`, poi effettua una richiesta DELETE a *Database-API* per eliminare le informazioni del documento dal database; restituisce True se l'operazione è andata a buon fine, False altrimenti;

3.3.2.3.18. TextFileManager

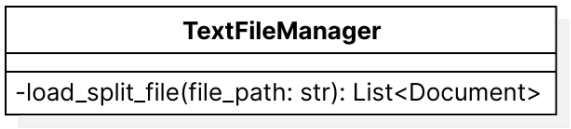

Figura 33: Diagramma della classe TextFileManager

La classe `TextFileManager` è un'implementazione concreta della classe FileManager che si occupa specificamente della gestione dei file di testo (.txt).
Questa classe utilizza il TextLoader di langchain per caricare e processare i file di testo.

3.3.2.3.18.1. Metodi

- `load_split_file(file_path: str) -> List[Document]`: implementa il metodo astratto della classe padre per caricare un file di testo dal percorso specificato; utilizza TextLoader con encoding UTF-8 per caricare il file, poi applica il text splitter per dividere il contenuto in chunk di dimensione appropriata; restituisce una lista di Document contenenti i chunk del testo;

3.3.2.3.19. PdfFileManager

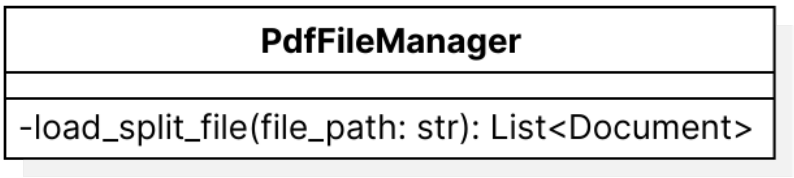

Figura 34: Diagramma della classe PdfFileManager

La classe `PdfFileManager` è un'implementazione concreta della classe FileManager specializzata nella gestione dei file PDF.
Utilizza PyPDFLoader di langchain per caricare e processare i documenti PDF.

3.3.2.3.19.1. Metodi

- `load_split_file(file_path: str) -> List[Document]`: implementa il metodo astratto della classe padre per caricare un file PDF dal percorso specificato; utilizza PyPDFLoader in modalità «single» per caricare il file PDF mantenendo il contesto tra le pagine, poi applica il text splitter per dividere il contenuto in chunk di dimensione appropriata; restituisce una lista di Document contenenti i chunk del PDF.

3.3.2.3.20. StringManager

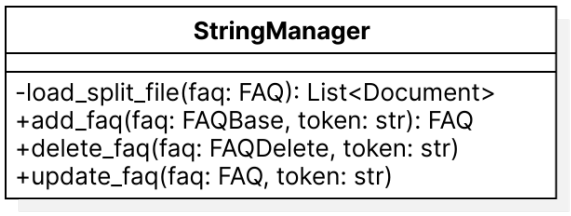

Figura 35: Diagramma della classe StringManager

La classe `StringManager` è un'implementazione concreta della classe FileManager che, a differenza delle altre implementazioni, non gestisce file fisici ma contenuti testuali come le FAQ. È responsabile della gestione delle FAQ nel sistema, inclusa la loro aggiunta, aggiornamento ed eliminazione dal database vettoriale.

3.3.2.3.20.1. Metodi

- `load_split_file(faq: FAQ) -> List[Document]`: implementa il metodo astratto della classe padre per trasformare una FAQ in un Document; crea un Document con contenuto formattato come «Domanda: {faq.question} Risposta: {faq.answer}» e con metadata che include la fonte («faqs») e l'ID della FAQ; applica il text splitter per dividere eventualmente il contenuto in chunk; restituisce una lista di Document.
- `add_faq(faq: FAQBase, token: str) -> FAQ`: aggiunge una nuova FAQ al sistema; invia una richiesta POST a *Database-API* per salvare la FAQ nel database relazionale, ottenendo indietro l'ID assegnato; crea un oggetto FAQ completo con l'ID ricevuto, chiama il metodo `load_split_file` per trasformare la FAQ in Document e lo aggiunge al database vettoriale tramite `VectorDatabase.add_documents()`; restituisce l'oggetto FAQ completo di ID.
- `delete_faq(faq: FAQDelete, token: str) -> None`: elimina una FAQ dal sistema; invia una richiesta DELETE a *Database-API* per rimuovere la FAQ dal database relazionale, verificando la password dell'admin; se l'operazione ha successo, rimuove la FAQ dal database vettoriale chiamando `VectorDatabase.delete_faq()`.

4. Requisiti

Di seguito è riportata la tabella con i requisiti funzionali, presenti nel documento *Analisi dei requisiti*, in cui viene indicato se sono stati soddisfatti o meno. Si ricorda che i requisiti hanno la seguente nomenclatura:

R - [numero] - [tipo] - [priorità]

con:
- **Numero**: numero progressivo che identifica il requisito, parte da 01.
- **Tipo**: può essere
  - **F**: requisito funzionale, indica una funzionalità del sistema;
  - **Q**: requisito di qualità, definisce le caratteristiche della qualità del prodotto, come un sistema deve essere o come il sistema deve esibirsi, per soddisfare le esigenze dell'utente;
  - **V**: requisito di vincolo, ovvero limiti e restrizioni imposte dal capitolato;
- **Priorità**: può essere
  - **O**: Obbligatorio, viene richiesto dal proponente* ed è necessario per considerare il prodotto completo;
  - **D**: Desiderabile, non è strettamente necessario ma è un valore aggiunto;

4.1. Tracciamento dei requisiti

ID Requisito	Descrizione	Stato
R-01-F-O	L'utente deve poter accedere alla piattaforma	Soddisfatto
R-02-F-O	Nel caso in cui il fornitore acceda per la prima volta alla piattaforma, deve aggiornare la password	Soddisfatto
R-03-F-O	Nel caso in cui il cliente acceda per la prima volta alla piattaforma, deve aggiornare la password	Soddisfatto
R-04-F-O	Sia cliente che fornitore devono poter cambiare la password liberamente	Soddisfatto

ID Requisito	Descrizione	Stato
R-05-F-D	L'utente non autenticato deve poter scegliere se reinserire le credenziali ad ogni accesso o se farle memorizzare alla piattaforma anche dopo il termine della sessione	Soddisfatto
R-06-F-O	L'utente non autenticato deve poter recuperare la password nel caso in cui la dimentichi	Soddisfatto
R-07-F-O	L'utente non autenticato deve essere notificato in caso di errore nell'inserimento delle credenziali	Soddisfatto
R-08-F-O	Sia cliente che fornitore devono avere la possibilità di uscire dal proprio account	Soddisfatto
R-09-F-O	Sia cliente che fornitore devono avere la possibilità di vedere la lista di tutte le chat in loro possesso, delle quali verrà mostrato il titolo	Soddisfatto
R-10-F-O	Sia cliente che fornitore devono avere la possibilità di creare una nuova chat con un contesto pulito, che verrà aggiunta alla lista di quelle già presenti	Soddisfatto
R-11-F-D	Sia cliente che fornitore devono avere la possibilità di modificare il titolo di una chat già esistente	Non soddisfatto
R-12-F-O	Sia cliente che fornitore devono avere la possibilità di aprire una chat singola dalla lista di tutte le chat in loro possesso, una volta aperta visualizzeranno il titolo della chat e i messaggi scambiati con il chatbot	Soddisfatto
R-13-F-O	Sia cliente che fornitore devono poter scrivere messaggi per comunicare con il chatbot	Soddisfatto

ID Requisito	Descrizione	Stato
R-14-F-O	L'utente che digita il messaggio deve essere avvisato nel caso in cui il messaggio scritto sia troppo lungo	Soddisfatto
R-15-F-O	Sia cliente che fornitore devono avere la possibilità di scrivere messaggi tramite FAQ preimpostate dal fornitore	Soddisfatto
R-16-F-O	Sia cliente che fornitore per comunicare con il chatbot devono poter trasmettere il messaggio scritto, e successivamente visualizzarlo nella chat	Soddisfatto
R-17-F-O	Sia cliente che fornitore devono ricevere la risposta elaborata dal chatbot in seguito all'invio di un messaggio; durante l'elaborazione della risposta l'utente deve ricevere un feedback che indica l'elaborazione della risposta	Soddisfatto
R-18-F-O	Il cliente deve avere la possibilità di valutare la risposta ricevuta dal chatbot tramite l'opzione 'Pollice su/giù'	Soddisfatto
R-19-F-O	Sia cliente che fornitore devono avere la possibilità di eliminare una chat presente nella lista di tutte le chat	Soddisfatto
R-20-F-D	L'utente deve avere la possibilità di scegliere tra tema scuro e tema chiaro dell'interfaccia	Soddisfatto
R-21-F-O	Il fornitore deve avere la possibilità di modificare il numero massimo di messaggi visualizzati all'interno delle chat	Soddisfatto
R-22-F-D	Il fornitore deve avere la possibilità di caricare il proprio logo per personalizzare la propria piattaforma fornita ai clienti	Soddisfatto

ID Requisito	Descrizione	Stato
R-23-F-D	Il fornitore deve avere la possibilità di cambiare il colore primario dell'interfaccia della propria piattaforma	Soddisfatto
R-24-F-D	Il fornitore deve avere la possibilità di visualizzare le statistiche relative alle interazioni con il chatbot	Soddisfatto
R-25-F-D	Il fornitore deve avere la possibilità di filtrare le statistiche visualizzate	Soddisfatto
R-26-F-O	Il fornitore deve avere la possibilità di aggiungere gli account per i propri clienti	Soddisfatto
R-27-F-O	Il fornitore deve essere avvisato nel caso in cui stia aggiungendo un account cliente già esistente	Soddisfatto
R-28-F-O	Il fornitore deve avere la possibilità di eliminare un account cliente solo dopo aver autorizzato l'eliminazione tramite la propria password	Soddisfatto
R-29-F-O	Il fornitore deve avere la possibilità di inserire documenti aziendali in modo da fornire ulteriore contesto all'chatbot	Soddisfatto
R-30-F-O	Il fornitore deve sapere quando un file caricato, sia come logo che come documento aziendale, non è nel formato corretto	Soddisfatto
R-31-F-O	Il fornitore deve avere la possibilità di visualizzare la lista dei documenti aziendali caricati nella piattaforma	Soddisfatto
R-32-F-O	Il fornitore deve avere la possibilità di eliminare un documento aziendale dalla piattaforma solo dopo aver autorizzato l'eliminazione tramite la propria password	Soddisfatto

ID Requisito	Descrizione	Stato
R-33-F-O	Il fornitore deve avere la possibilità di aggiungere delle domande preimpostate nella piattaforma	Soddisfatto
R-34-F-O	Il fornitore deve avere la possibilità di visualizzare la lista delle domande preimpostate inserite nella piattaforma	Soddisfatto
R-35-F-O	Il fornitore deve avere la possibilità di modificare le domande preimpostate già inserite nella piattaforma	Soddisfatto
R-36-F-O	Il fornitore deve avere la possibilità di eliminare le domande preimpostate dalla piattaforma solo dopo aver autorizzato l'eliminazione tramite la propria password	Soddisfatto
R-37-F-O	Un utente non autenticato o il cliente che sta utilizzando la piattaforma deve essere avvisato nel caso in cui il sistema non sia raggiungibile, possibilmente specificando il motivo del malfunzionamento	Soddisfatto
R-38-F-O	Un utente non autenticato o il cliente che sta utilizzando la piattaforma deve essere avvisato nel caso in cui la richiesta che ha inviato contenga dati mancanti o errati; nel caso sia pertinente deve anche essere specificato il problema	Soddisfatto
R-39-F-O	Il fornitore deve avere la possibilità di modificare gli account degli utenti già inseriti nella piattaforma	Soddisfatto

Tabella 7: Tracciamento dei requisiti

4.2. Resoconto

Di seguito il grafico che rappresenta il numero di requisiti funzionali soddisfatti e non soddisfatti:

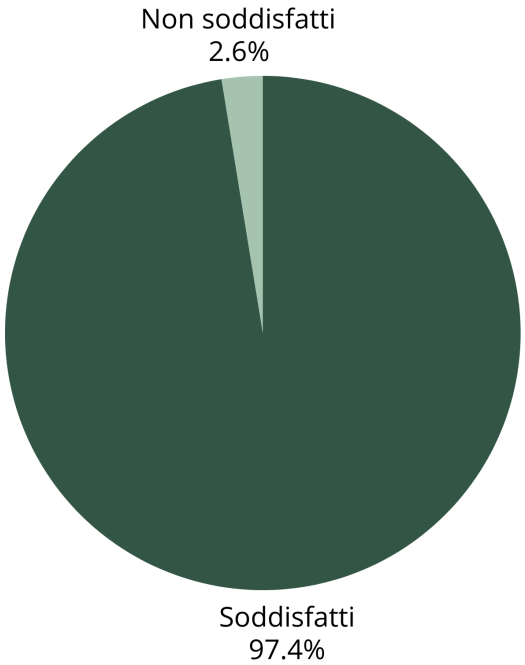

Figura 36: Grafico dei requisiti soddisfatti e non soddisfatti

L'unico requisito funzionale non soddisfatto è il requisito R-11-F-D, un requisito desiderabile, che prevede la possibilità di modificare il titolo di una chat già esistente. Questo perché, in accordo con il proponente, si è deciso di fare generare il titolo della chat all'LLM basandosi sui messaggi inviati, rendendo quindi questo requisito non necessario.