Specifica Tecnica

Gruppo Argo — ***Progetto ChatSQL***

Informazioni sul documento	
Versione	0.0.7
Approvazione	TODO
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gruppo Argo

UNIVERSITÀ DEGLI STUDI DI PADOVA

Registro delle modifiche

Ver.	Data	Redazione	Verifica	Descrizione
0.0.7	2024-08-03	Raul Pianon	Riccardo Cavalli, Sebastiano Lewental, Mattia Zecchinato	Aggiunta sezione architettura esagonale
0.0.6	2024-08-02	Martina Dall'Amico, Raul Pianon	Riccardo Cavalli	Aggiunta sezione Database
0.0.5	2024-07-31	Riccardo Cavalli	Raul Pianon	Progettazione dei componenti front-end, struttura back-end
0.0.4	2024-07-30	Riccardo Cavalli	Raul Pianon	Struttura cartelle front-end
0.0.3	2024-07-29	Riccardo Cavalli	Raul Pianon	Panoramica generale dell'architettura
0.0.2	2024-07-26	Riccardo Cavalli	Raul Pianon	Completata sezione scelte tecnologiche
0.0.1	2024-07-20	Riccardo Cavalli	Tommaso Stocco, Mattia Zecchinato	Prima stesura del documento

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di descrivere le scelte tecnologiche e progettuali alla base dello sviluppo del prodotto ChatSQL. Verranno quindi illustrati gli stili e i pattern architetturali adottati dal team.

1.2 Riferimenti

Il presente documento si basa su normative elaborate dal team, dall'ente proponente o da entità esterne, oltre a includere materiali informativi. Tali riferimenti sono elencati di seguito.

1.2.1 Riferimenti normativi

- *Norme di Progetto v1.0.1*;
- Regolamento del progetto didattico:
  https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/PD2.pdf;
- Capitolato C9 - ChatSQL:
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9.pdf
    (Ultimo accesso: 2024-07-02);
  - https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9p.pdf
    (Ultimo accesso: 2024-07-02).

1.2.2 Riferimenti informativi

- *Analisi dei Requisiti v1.0.2*;
- *Glossario v1.0.1*;
- Verbali interni:
  - 2024-04-03;
  - 2024-04-10;
  - 2024-04-16;
  - 2024-04-20;
  - 2024-04-25;
  - 2024-05-02;
  - 2024-05-07;
  - 2024-05-16;
  - 2024-05-23;
  - 2024-05-28;

  - 2024-06-03;
  - 2024-06-14;
  - 2024-06-22;
  - 2024-07-06;
  - 2024-07-10;
  - 2024-07-18.
- Verbali esterni:
  - 2024-04-09;
  - 2024-05-06;
  - 2024-05-22;
  - 2024-06-07;
  - 2024-07-09.

1.3 Glossario

Allo scopo di evitare incomprensioni relative al linguaggio utilizzato nella documentazione di progetto, viene fornito un *Glossario*, nel quale ciascun termine è corredato da una spiegazione che mira a disambiguare il suo significato. I termini tecnici, gli acronimi e i vocaboli ritenuti ambigui vengono formattati in corsivo all'interno dei rispettivi documenti e marcati con una lettera $_{G}$ in pedice. Tutte le ricorrenze di un termine definito nel *Glossario* subiscono la formattazione sopracitata.

2 Tecnologie utilizzate

Nella sezione seguente sono documentate le scelte tecnologiche del team, la cui validità è stata dimostrata mediante lo sviluppo di un *Proof of Concept*$_G$. Per ciascuna tecnologia, il team riporta le motivazioni che ne hanno determinato la scelta.

2.1 Backend

2.1.1 FastAPI

FastAPI è un *framework*$_G$ web moderno per la creazione di *API*$_G$ con Python.

Motivazioni

- **Flessibilità**: FastAPI offre ampia libertà agli sviluppatori e può essere facilmente integrato con librerie esterne; contrariamente a Django, non impone strutture rigide, consentendo agli sviluppatori di scegliere gli strumenti più adatti al loro caso d'uso;
- **Prestazioni**: FastAPI assicura prestazioni elevate grazie all'integrazione nativa con Starlette e Pydantic;
- **Orientato alla produttività**: FastAPI è progettato per massimizzare la produttività degli sviluppatori, riducendo il tempo necessario per la configurazione dell'ambiente e lo sviluppo delle API;
- **Documentazione API**: FastAPI genera automaticamente la documentazione interattiva delle API, a differenza di Flask e Django, che richiedono l'integrazione di librerie esterne per ottenere funzionalità simili;
- **Validazione automatica dei tipi**: FastAPI utilizza Pydantic per la validazione automatica dei dati;
- **Modernità**: FastAPI è un framework moderno, che sfrutta le funzionalità più recenti di Python, come le annotazioni di tipo.

2.1.2 Python

Python è un linguaggio di programmazione ad alto livello ampiamente utilizzato in applicazioni di machine learning ed elaborazione del linguaggio naturale (NLP).

Motivazioni

- **Leggibilità**: Python ha una sintassi intuitiva e pulita, che agevola la leggibilità, la manutenzione e la collaborazione;
- **NLP**: Python è il linguaggio di riferimento per applicazioni di NLP (elaborazione del linguaggio naturale);
- **Librerie**: Python dispone di un'ampia gamma di librerie e moduli di terze parti;

- **Orientato agli oggetti**: Python è un linguaggio orientato agli oggetti; pertanto, consente di organizzare il codice seguendo i principi del design *SOLID*$_G$;
- **Comunità**: Python dispone di una vasta comunità di sviluppatori che forniscono supporto e soluzioni a problemi comuni.

2.1.3 txtai

txtai$_G$ è un database di embeddings sviluppato in Python e progettato per ottimizzare la ricerca semantica.

Motivazioni

- **Comunicazione con la Proponente**: txtai è stato proposto dalla *Proponente*$_G$, sia nel capitolato che durante le riunioni esterne, per facilitare la comunicazione e le revisioni congiunte;
- **Query SQL-like**: txtai supporta query SQL-like per la ricerca semantica;
- **Configurazione minima**: txtai richiede una configurazione minima e fornisce una suite completa di strumenti di intelligenza artificiale, inclusa la traduzione automatica tra lingue;
- **Persistenza degli indici**: txtai semplifica il processo di memorizzazione e ripristino degli *indici*$_G$;
- **Debug**: txtai mette a disposizione funzioni specifiche per il debug del processo di generazione di un *prompt*$_G$.

2.1.4 SQLAlchemy

SQLAlchemy è lo strumento principale per l'interazione con database *SQL*$_G$ in Python.

Motivazioni

- **Portabilità**: SQLAlchemy supporta i principali *DBMS*$_G$ e agevola la transizione;
- **Performance e sicurezza**: SQLAlchemy massimizza l'efficienza e la sicurezza delle transazioni attraverso il pattern "Unit of Work";
- **SQL injection**: SQLAlchemy include una protezione integrata contro l'SQL injection;
- **Leggibilità e mantenibilità**: SQLAlchemy utilizza classi e oggetti Python per rappresentare le relazioni di un *database*$_G$, rendendo il codice intuitivo e autoesplicativo;
- **Flessibilità**: SQLAlchemy consente di costruire query SQL in linguaggio nativo o di utilizzare un *ORM*$_G$ per interagire con il database.

2.1.5 JWT

JSON Web Token (JWT) uno standard web per lo scambio di dati.

Motivazioni

- **Stateless**: JWT consente di autenticare le richieste senza necessità di sessioni sul server, migliorando la scalabilità e riducendo il carico sul server;
- **Sicurezza**: I JWT sono firmati digitalmente, il che garantisce l'integrità e l'autenticità del token;
- **Formato standard**: JWT è supportato da una vasta gamma di linguaggi e framework, tra cui FastAPI.

2.1.6 pytest

pytest è un *framework*$_G$ di test Python che può essere utilizzato per scrivere test di unità, test di integrazione e test end-to-end.

Motivazioni

- **Semplicità**: pytest consente la definizione dei test come normali funzioni in Python, senza la necessità di definire classi apposite a meno che non siano volute. Inoltre, non vi è presenza di codice boilerplate;
- **Concisione**: pytest dispone di fixtures per i test, ovvero un modo conciso ed esplicito per aggiungere codice di configurazione alle funzioni di test che richiedono passaggi di preparazione;
- **Parametrizzazione nativa**: pytest permette nativamente la parametrizzazione dei test, così da poter eseguire lo stesso test con input e risultati attesi diversi in base alle esigenze;
- **Estensibilità**: pytest presenta un ricco ecosistema di plugin per favorire lo sviluppo dei test in circostanze specifiche senza dover alterare il codice;
- **Accuratezza degli output**: pytest fornisce di default un elevato grado di dettaglio nell'analisi degli output dei test, come i tracebacks e le assertion introspection, semplificando l'analisi degli errori;
- **Facilità d'integrazione**: pytest si integra facilmente con le pipeline CI/CD, strumenti di coverage e diverse interfacce di testing.

2.2 Frontend

2.2.1 Vue.js

Vue.js$_G$ è un framework JavaScript utilizzato per la creazione di interfacce utente reattive e dinamiche.

Motivazioni

- **Component-based**: Vue.js è un framework basato su componenti, suddivisi in tre sezioni: template (HTML), stile (CSS) e logica (JavaScript);

- **Reattività**: Vue.js offre un sistema reattivo che permette di aggiornare automaticamente la vista dell'utente in base ai cambiamenti dello stato dell'applicazione;
- **Integrazione**: Vue.js può essere facilmente integrato con librerie esterne, come Axios e PrimeVue;
- **Prestazioni**: Vue.js implementa il concetto, introdotto da React, di DOM virtuale, riducendo i re-render non necessari e migliorando le prestazioni dell'applicazione;
- **MVVM**: Vue.js implementa il pattern MVVM (Model-View-ViewModel), con un focus sul livello ViewModel.

2.2.2 Axios

Axios è una libreria JavaScript utilizzata per connettersi con le API di *back-end*$_G$ e gestire le richieste effettuate tramite il protocollo HTTP.

Motivazioni

- **Semplicità**: Axios è un libreria semplice e intuitiva per effettuare richieste HTTP;
- **Gestione dati JSON**: Axios converte automaticamente i dati da e in JSON;
- **Compatibilità e integrazione**: Axios è compatible e facilmente integrabile con Vue 3 e la Composition API;
- **Promise-based**: Axios utilizza un'interfaccia Promise-based per gestire le richieste asincrone.

2.2.3 TypeScript

TypeScript è un linguaggio di programmazione che estende JavaScript, aggiungendo caratteristiche come i tipi di dato.

Motivazioni

- **Compatibilità retroattiva** (backward compatibility): Qualsiasi programma scritto in JavaScript può funzionare in TypeScript senza richiedere modifiche;
- **Robustezza**: TypeScript fornisce un sistema di type-checking basato sulla tipizzazione statica, che permette di rilevare errori di tipo durante la fase di sviluppo;
- **Documentazione automatica**: La tipizzazione funge da documentazione incorporata, semplificando la manutenzione e la collaborazione;
- **Supporto IDE**: TypeScript è supportato dai principali editor, come Visual Studio Code, che forniscono funzionalità avanzate di debugging e formattazione.

2.2.4 PrimeVue

PrimeVue è una libreria di componenti per Vue.js.

Motivazioni

- **Integrazione**: PrimeVue è stato progettato appositamente per Vue.js e, pertanto, offre un'integrazione nativa con il *framework*$_G$;
- **Completezza**: PrimeVue offre una vasta gamma di componenti, temi e stili personalizzabili;
- **Funzionalità avanzate**: PrimeVue fornisce componenti per la creazione di interfacce utente complesse e interattive. Inoltre, supporta le caratteristiche e funzionalità avanzate di Vue.js;
- **Documentazione e supporto**: La documentazione e i materiali di supporto sono orientati all'integrazione con Vue.js.

2.2.5 Jest

Jest è un framework di test per JavaScript e TypeScript. Offre una suite completa di strumenti per il testing automatizzato.

Motivazioni

- **Semplicità**: Jest richiede una configurazione iniziale minima, specialmente con TypeScript. L'integrazione con ts-jest consente un supporto TypeScript senza processi di configurazione complicati;
- **Completezza**: Jest è una soluzione completa che include un test runner, una libreria di asserzioni e il mocking integrato. Ciò riduce la necessità di installare e configurare librerie aggiuntive;
- **Prestazioni**: Jest esegue i test in parallelo per impostazione predefinita, migliorando significativamente le prestazioni e riducendo i tempi di esecuzione dei test;
- **Mocking**: Jest consente agli sviluppatori di mockare funzioni, moduli e timer senza richiedere librerie aggiuntive;
- **Accuratezza degli output**: Jest fornisce messaggi di errore chiari e informativi, facilitando l'identificazione e la risoluzione dei problemi. La modalità watch integrata supporta il debug dei test durante lo sviluppo;
- **Estensibilità**: Jest è accompagnato da una ricca suite di plugin che consente di estenderne le funzionalità. In questo modo è possibile soddisfare specifiche esigenze di testing senza alterazioni al codice;
- **Facilità d'integrazione**: Jest è ben supportato dalle piattaforme di integrazione e distribuzione continua come Travis CI, CircleCI e GitHub Actions, facilitando l'automazione fluida del processo di testing.

2.3 Altri strumenti e tecnologie

2.3.1 Docker

Docker$_G$ è una piattaforma open-source per la creazione, la distribuzione e l'esecuzione di applicazioni in contenitori leggeri e portabili.

Motivazioni

- **Isolamento delle applicazioni**: Docker consente di isolare le applicazioni in contenitori, garantendo che ciascuna applicazione abbia un ambiente di esecuzione indipendente;
- **Collaborazione**: La condivisione delle librerie e delle dipendenze semplifica la collaborazione e riduce il rischio di conflitti;
- **Portabilità**: I contenitori possono essere eseguiti su qualsiasi sistema che supporti Docker, indipendentemente dall'ambiente di sviluppo o di produzione;
- **Scalabilità**: Docker permette di aggiungere o rimuovere container in maniera rapida ed efficiente;
- **CI/CD**: Docker può essere utilizzato per automatizzare i processi di continuous integration e continuous delivery.

2.3.2 SQLite

SQLite$_G$ è una libreria di gestione di database relazionali integrata nella maggior parte dei linguaggi di programmazione.

Motivazioni

- **Leggerezza**: SQLite è estremamente leggero e richiede una quantità inferiore di risorse rispetto ai DBMS tradizionali;
- **Autosufficienza**: SQLite non richiede un processo separato per funzionare ed è progettato per essere integrato direttamente nelle applicazioni;
- **Compatibilità**: SQLite supporta la quasi totalità delle funzionalità dei database SQL standard;
- **Portabilità**: SQLite memorizza l'intero database in un singolo file, semplificando il trasferimento dei dati tra sistemi e piattaforme differenti;
- **ACID Compliance**: SQLite supporta le transazioni ACID (Atomicità, Coerenza, Isolamento e Durabilità), garantendo l'integrità dei dati anche in caso di malfunzionamenti;
- **Performance**: SQLite è ottimizzato per gestire database di piccole e medie dimensioni.

2.3.3 Hugging Face

Hugging Face$_G$ è una piattaforma open-source che fornisce strumenti e risorse per lavorare su progetti di NLP (elaborazione del linguaggio naturale).

Motivazioni

- **Modelli pre-addestrati**: Hugging Face mette a disposizione una vasta collezione di modelli pre-addestrati;
- **Modelli open-source**: Hugging Face fornisce un'ampia selezione di modelli open-source;
- **Modelli locali**: Hugging Face offre la possibilità di scaricare i *modelli*$_G$ in locale per l'esecuzione offline;
- **Comunità**: La piattaforma dispone di una comunità attiva di sviluppatori e ricercatori che contribuiscono al miglioramento continuo dei modelli;
- **Supporto per diverse lingue**: Hugging Face offre una varietà di modelli multilingue e modelli per la traduzione.

2.3.4 ChatGPT

ChatGPT$_G$ (Chat Generative Pre-trained Transformer) è un ChatBOT basato su intelligenza artificiale e apprendimento automatico, sviluppato da OpenAI.

Motivazioni

- **Popolarità**: ChatGPT è uno dei ChatBOT più diffusi nel campo informatico, e l'adeguatezza delle sue risposte è stata dimostrata in numerosi contesti;
- **Integrazione**: ChatGPT offre API per l'integrazione con applicazioni esterne;
- **Supporto Multilingue**: I modelli alla base di ChatGPT supportano diverse lingue, tra cui l'italiano e l'inglese.

2.3.5 LMSYS Chatbot Arena

LMSYS Chatbot Arena è uno spazio online dedicato al *benchmarking*$_G$ di LLM.

Motivazioni

- **Benchmark**: LMSYS Chatbot Arena permette di confrontare modelli diversi e di selezionare il miglior output in ciascuna iterazione;
- **Disponibilità**: LMSYS Chatbot Arena consente di testare i modelli più diffusi, inclusi ChatGPT, Gemini, Claude e LLaMA.

2.3.6 LM Studio

LM Studio$_G$ è un'applicazione desktop che semplifica il processo di testing di modelli linguistici di grandi dimensioni (LLM).

Motivazioni

- **Semplicità**: LM Studio offre un'interfaccia intuitiva e user-friendly per testare i modelli LLM;
- **Test offline**: LM Studio consente di scaricare le versioni quantizzate dei modelli per l'esecuzione offline;
- **Debug**: LM Studio fornisce strumenti per il debug e l'analisi dei risultati;
- **Server OpenAI-like**: Le interazioni con il server locale di LM Studio seguono il formato API di OpenAI;
- **Prestazioni**: LM Studio fornisce un'opzione per abilitare l'accelerazione GPU e velocizzare i processi di inferenza dei modelli.

2.4 Panoramica delle tecnologie

Di seguito è fornita una panoramica generale delle tecnologie utilizzate dal team, suddivise nelle seguenti categorie:

- **Framework**;
- **Linguaggi**;
- **Librerie**;
- **Strumenti**.

2.4.1 Framework

Tabella 2.1: Framework utilizzati

Nome	Versione	Descrizione
FastAPI	0.110.0	Framework web moderno per la creazione di API con Python.
Vue.js	3.4.21	Framework JavaScript per la creazione di interfacce utente reattive e dinamiche.
pytest	8.3.2	Framework di test per Python. Pytest consente di scrivere test di unità, di integrazione e di sistema.
Jest	29.7.0	Framework di test per JavaScript e TypeScript. Jest offre una suite completa di strumenti per il testing automatizzato.
Cypress	13.6.0	Framework di test che può essere utilizzato per testare l'intera applicazione simulando l'interazione dell'utente.

2.4.2 Linguaggi

Tabella 2.2: Linguaggi utilizzati

Nome	Versione	Descrizione
HTML	5	Linguaggio di markup utilizzato per definire la struttura delle pagine web.
CSS	3	Linguaggio usato per definire lo stile e l'aspetto estetico delle pagine web.
TypeScript	5	Espansione del linguaggio JavaScript, progettata per migliorare lo sviluppo di pagine web dinamiche e interattive.
Python	3.10.12	Linguaggio di programmazione ad alto livello e orientato agli oggetti.

2.4.3 Librerie

Tabella 2.3: Librerie utilizzate

Nome	Versione	Descrizione
txtai	7.0.0	Database di embeddings sviluppato in Python e progettato per ottimizzare la ricerca semantica.
SQLAlchemy	2.0.31	Libreria open-source che fornisce un toolkit SQL e un ORM per le interazioni con database.
jsonschema	4.23.0	Libreria per la validazione di JSON Schema in Python.
Pydantic	2.8.2	Libreria Python per la validazione dei dati mediante le annotazioni di tipo.
PyJWT	2.8.0	Libreria Python per la gestione dei JSON Web Token.
parameterized	0.9.0	Libreria Python utilizzata per creare test parametrizzati, ovvero test che vengono eseguiti più volte con diversi set di parametri.
openai	1.37.1	Libreria Python per l'interazione con i servizi di OpenAI.

Continua nella prossima pagina

Tabella 2.3: Librerie utilizzate (continua)

Nome	Versione	Descrizione
aiofiles	24.1.0	Libreria Python per eseguire operazioni di I/O su file in modo asincrono.
PrimeVue	3.53.0	Libreria di componenti per Vue.js.
Vue I18n	9.0.0	Libreria per la creazione di interfacce utente multilingue in applicazioni Vue.js.
Vuex (oppure Pinia)	3.10.12	Libreria di gestione dello stato in applicazioni Vue.js.
Axios	3.10.12	Libreria JavaScript utilizzata per connettersi con le API di back-end e gestire le richieste effettuate tramite il protocollo HTTP.
Vue Test Utils	2.4.6	Libreria per il testing di componenti in applicazioni Vue.js. Vue Test Utils supporta sia test di unità che test di integrazione.

2.4.4 Strumenti

Tabella 2.4: Strumenti utilizzati

Nome	Versione	Descrizione
Git	2.45.2	Sistema di controllo di versione distribuito utilizzato principalmente nello sviluppo software.
pip	24.0	Sistema di gestione dei pacchetti per Python.
npm	10.7.0	Sistema di gestione dei pacchetti per JavaScript e per l'ambiente di esecuzione Node.js.
ESLint	9.5.0	Strumento di analisi statica che mira a individuare errori di programmazione, problemi stilistici e costrutti sospetti nel codice JavaScript.
Prettier	3.3.0	Formattatore di codice per JavaScript, TypeScript, JSON, HTML, CSS, Vue e YAML.
Pylint	3.2.6	Strumento di analisi statica per Python.

Continua nella prossima pagina

Tabella 2.4: Strumenti utilizzati (continua)

Nome	**Versione**	**Descrizione**
Vue Test Utils	3.0.0	Libreria per il testing di componenti in applicazioni Vue.js.
Docker	/	Piattaforma per l'esecuzione di applicazioni in contenitori isolati. Docker può essere utilizzato in combinazione con GitHub Actions per automatizzare i processi di build, test e distribuzione.
SQLite	/	Sistema di gestione di database relazionali open-source, leggero e facilmente portabile.
Hugging Face	/	Piattaforma di hosting di modelli e set di dati di machine learning.
ChatGPT	/	ChatBOT basato su intelligenza artificiale e apprendimento automatico.
LMSYS Chatbot Arena	/	Spazio online dedicato al benchmarking di LLM.
LM Studio	/	Applicazione desktop per il testing di LLM in locale.

3 Architettura

3.1 Introduzione

Il prodotto ChatSQL è basato su un'architettura client-server. Il client è l'interfaccia attraverso la quale gli utenti interagiscono con il sistema, come ad esempio un browser web. In altre parole, il client è il componente che richiede risorse o servizi. Il server è l'applicazione che riceve ed elabora le richieste provenienti da uno o più client, fornendo risposte appropriate.

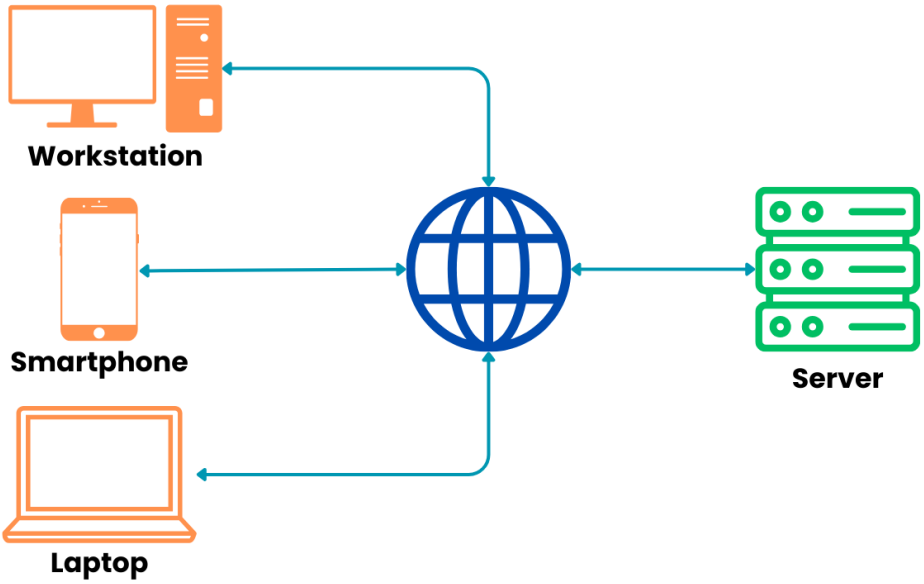

Figura 3.1: Architettura client-server

Per migliorare lo sviluppo collaborativo, la modularità e la manutenibilità, il sistema è stato suddiviso in due componenti principali:

- **Front-end**: è la porzione di un sistema che l'utente visualizza e con cui può interagire. Il front-end è sviluppato utilizzando il framework *Vue.js*$_G$ ed è responsabile dell'interfaccia grafica, che deve essere intuitiva, funzionale e accattivante. Trasmette le richieste dell'utente al back-end e visualizza i risultati ottenuti;
- **Back-end**: è il segmento che gestisce la logica di business, l'elaborazione dei dati e la comunicazione con i database e altri servizi. Il back-end è sviluppato utilizzando il framework *FastAPI*$_G$.

La comunicazione tra il front-end e il back-end avviene tramite chiamate *API*$_G$. Il team segue le linee guida e i principi definiti da REST (representational state transfer), uno stile architetturale che impone condizioni sul funzionamento di un'API. Le REST API (o RESTful API) sono stateless, il che significa che ogni richiesta HTTP deve includere tutte le informazioni necessarie per elaborarla. Questo riduce il carico sul

server e migliora la scalabilità. Inoltre, l'approccio stateless agevola l'implementazione di sistemi di caching, migliorando le prestazioni complessive. Una REST API è simile a un sito web in esecuzione in un browser con funzionalità HTTP integrata. Le operazioni sono basate su metodi HTTP standard come GET, POST, PUT e DELETE.

La persistenza delle informazioni dei dizionari (nome, descrizione, ecc.) è garantita dalla presenza di un database, che memorizza anche gli operatori registrati nel sistema. Il database è implementato utilizzando SQLite. Di seguito è riportata l'architettura ad alto livello della web app.

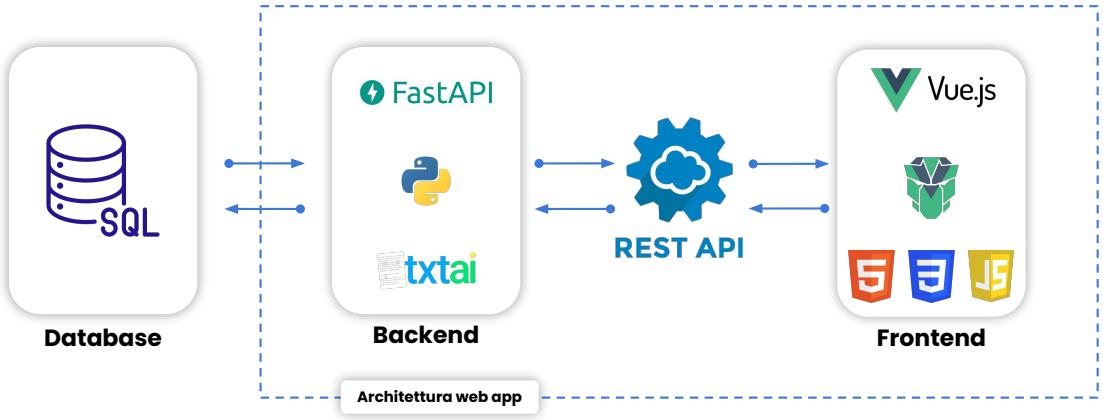

Figura 3.2: Architettura web app

3.2 Assemblaggio dei componenti

Docker Compose viene utilizzato per gestire applicazioni multi-container, permettendo di assemblare diversi servizi che compongono un'applicazione. Nel contesto di ChatSQL, il team ha creato i seguenti container Docker:

- **backend**: espone l'interfaccia di backend sulla porta 8000. All'indirizzo *localhost:8000/docs* è possibile consultare la documentazione interattiva delle API. Inoltre, sono disponibili dettagli sui Data Transfer Objects (DTO);
- **frontend**: espone l'interfaccia utente sulla porta 5173.

3.3 Struttura del sistema

3.3.1 Front-end

Vue.js implementa il pattern architetturale MVVM (Model-View-ViewModel), una declinazione del pattern Model-View-Controller (MVC). L'MVVM viene integrato nativamente attraverso il modo in cui Vue gestisce i dati, la logica e l'interfaccia utente. Il ViewModel è un oggetto che sincronizza il Model e la View. Ogni istanza di Vue è un ViewModel.

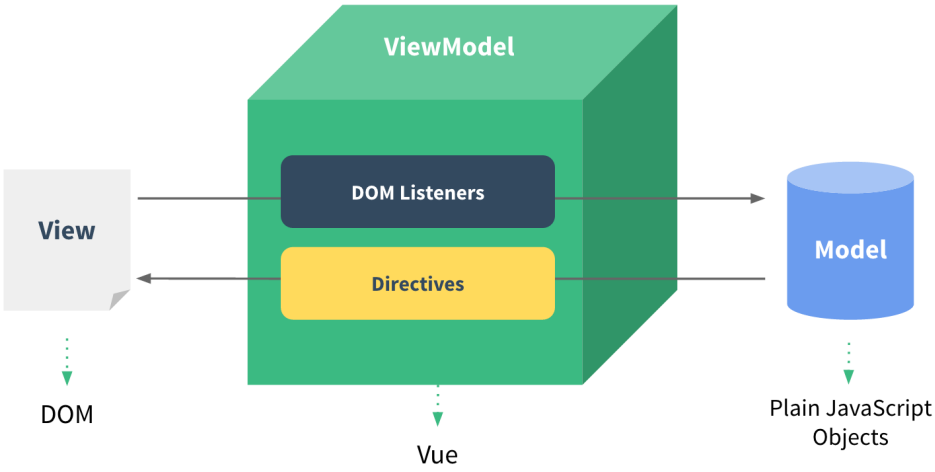

Figura 3.3: Model-View-ViewModel (MVVM)

Il pattern MVVM comprende tre layer, ciascuno con un ruolo specifico:

- **Model**: è responsabile della gestione dei dati e della logica di business. In Vue.js, il Model è tipicamente rappresentato da semplici oggetti JavaScript (plain JavaScript objects) o data model objects (oggetti più strutturati) che diventano reattivi quando utilizzati dalle istanze di Vue. Inoltre, Vue offre soluzioni per la gestione centralizzata dello stato, come Vuex e Pinia;
- **View** (Presentation layer): è responsabile della presentazione dei dati all'utente. Descrive la struttura del DOM (Document Object Model). La View è rappresentata dal template, che utilizza una sintassi HTML arricchita con binding e direttive specifiche per riflettere i cambiamento di stato. L'interfaccia viene aggiornata dinamicamente quando cambiano i dati del modello;
- **ViewModel**: è il layer che collega il Model e la View. Ha il compito di gestire le interazioni dell'utente e sincronizzare i dati con la View. Il ViewModel è rappresentato dalle istanze di Vue. Oltre a sincronizzare il Model e la View, il ViewModel garantisce una chiara separazione tra i dati e la logica di presentazione.

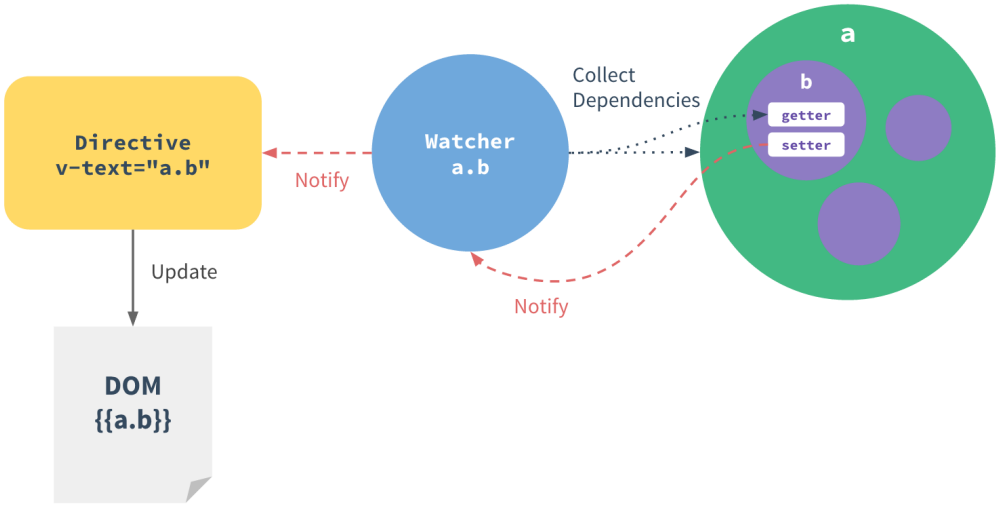

Figura 3.4: Funzionamento del pattern MVVM

Vue.js implementa due concetti fondamentali:

- **Reattività**: la reattività è un meccanismo che permette di aggiornare dinamicamente la View quando cambiano i dati del Model. I dati reattivi sono oggetti che vengono osservati da Vue. La differenza rispetto ai normali oggetti JavaScript è che Vue è in grado di intercettare l'accesso e la modifica di tutte le proprietà di un oggetto reattivo;
- **Composition API**: permette di suddividere la logica di un componente in funzioni riutilizzabili. La Composition API migliora la riusabilità, l'organizzazione e la manutenibilità del codice. Inoltre, la Composition API risolve alcune limitazioni della Options API, che emergono quando la logica di un singolo componente supera una determinata soglia di complessità.

La struttura organizzativa del front-end segue le convenzioni definite dal framework Vue.js, con alcune modifiche e personalizzazioni per migliorare la riusabilità, la manutenibilità e l'usabilità. Il front-end è suddiviso nelle seguenti cartelle:

```
frontend
├── public
│   ├── themes
│   └── icons
└── src
    ├── assets
    ├── components
    │   └── layout
    ├── composables
    ├── locales
    ├── router
    ├── store
    ├── services
    ├── types
    └── views
```

Di seguito è riportata una breve descrizione delle cartelle principali:

- **public**: contiene le icone e i temi utilizzati nell'applicazione;
- **src**: contiene il codice sorgente e gli assets.

La cartella *src* è suddivisa nelle seguenti sottocartelle:

- **assets**: contiene i file statici, come immagini, font e fogli di stile;
- **components**: contiene i componenti Vue riutilizzabili da più viste;
- **composables**: contiene le funzioni riutilizzabili;
- **locales**: contiene i file $JSON_G$ di traduzione;
- **router**: contiene le definizioni delle route dell'applicazione. La definizione di una route comprende l'URL o il percorso che l'utente deve inserire nel browser per accedere alla route. Inoltre, specifica quale componente deve essere caricato quando l'utente accede a quel percorso;
- **store**: contiene i moduli per la gestione dello stato condiviso tra i componenti;
- **services**: contiene funzioni di utilità e servizi per la comunicazione con il back-end;
- **types**: contiene le definizioni dei tipi e delle interfacce per un'API basata su OpenAPI;
- **views**: contiene le viste dell'applicazione. Una vista è un componente principale che può essere suddiviso in sottocomponenti, disponibili nella cartella *components*. In genere, le view rappresentano pagine o sezioni centrali e sono gestite dal router di Vue.

3.3.2 Back-end

Il modello architetturale scelto dal gruppo è il modello ad **architettura esagonale**. Questo modello trova le sue basi nell'architettura a livelli, dalla quale cerca di pren-

derne le basi e di far fronte ai problemi di dipendenza stretta che si creavano tra i livelli.

L'architettura è composta da 3 parti principali:

- **Core**: il core è la sezione centrale dell'architettura, in cui risiede la business logic dell'applicazione. È indipendente da qualsiasi interfaccia utente o servizio esterno. Ciò viene fatto per mantenerla isolata e disaccoppiata dal resto dell'architettura, in modo che possa agire indipendentemente;
- **Ports**: costituiscono dei punti di comunicazione tra il core e altri parti del sistema o servizi esterni. Esistono due tipi principali di porte:
  - Inbound port: consentono al nucleo di essere invocato da componenti esterne;
  - Outbound port: consentono al nucleo di reperire informazioni dall'esterno.
- **Adapters**: sono pattern architetturali che permettono di adattare i dati in modo che possano essere ricevuti dall'esterno e passare per le porte o vice versa. Costituiscono la sezione più esterna dell'architettura e consentono pertanto il passaggio e il monitoraggio dei dati.

Questa architettura è rappresentata a forma esagonale per diversi motivi; da una parte l'architettura vuole ricordare la cella di un alveare. Avendo tale forma, la cella si può incastrare con altre celle allo scopo di definire un alveare. Allo stesso modo un'architettura esagonale può interfacciarsi con altre strutture con le stesse caratteristiche per definire un sistema più grande. Un altro motivo della scelta è la visualizzazione della simmetria della struttura, la quale viene visualmente divisa a metà, dove una metà si interfaccia con l'esterno per ricevere gli input, mentre l'altra si interfaccia con l'esterno per inviare informazioni in output.

L'architettura esagonale offre numerosi vantaggi, tra i quali:

- **Flessibilità**: i componenti dell'architettura agiscono indipendentemente tra di loro e di conseguenze si possono adattare con facilità;
- **Scalabilità**: una cella ad architettura esagonale può essere vista come un microservizio e per tanto risulta più facilmente replicabile;
- **Testabilità**: il core è isolato e disaccoppiato dal resto dell'architettura;
- **Manutenibilità**: in quanto il core è isolato e disaccoppiato dal resto dell'architettura.

Questo tipo di architettura non è immune a difetti, tra i quali si possono trovare:

- **Complessità**: l'architettura richiede una progettazione puntuale e una buona conoscenza dei pattern architetturali;
- **Dipendenza** che permane tra il core e le porte: cambiamenti sostanziali del core possono richiedere una manutenzione delle porte;
- **Difficoltà di debugging**: il core è separato dal resto dell'architettura, di conseguenza è possibile introdurre bug_G nell'integrazione con servizi esterni.

Il gruppo ha concordato di adottare questo modello architetturale, poiché ritiene che il core dell'applicazione sia sufficientemente robusto da non dover creare problemi di manutenzione nel tempo. Inoltre, la scalabilità e la flessibilità di questo modello architetturale sono state considerate un vantaggio significativo per gestire la varietà dei contesti di applicazione del sistema. Infine, l'architettura si sposa bene con le esigenze dell'applicazione di interfacciarsi con servizi esterni quali $database_G$ o LLM_G e API_G.

La struttura organizzativa del back-end segue i principi dello stile architetturale scelto, al fine di semplificare il processo di traduzione della progettazione in codice. Il back-end è suddiviso nelle seguenti cartelle:

```
backend
├── TODO
└── TODO
```

4 Progettazione ad alto livello dei componenti

4.1 Front-end

4.1.1 AppLayout

4.1.1.1 Descrizione

AppLayout è il componente che definisce la struttura generale dell'applicazione.

4.1.1.2 Sottocomponenti

AppLayout è composto dai seguenti sottocomponenti:

- **AppTopbar**: componente condiviso da tutte le pagine dell'interfaccia. La topbar permette di accedere ai menu dell'applicazione. È situata nella zona superiore dello schermo e include il seguente componente (puramente visivo):
  - **AppLogo**: logo dell'applicazione.
- **LoginDialog**: finestra per l'autenticazione dell'utente;
- **MenuSidebar**: barra laterale che contiene il menu di navigazione principale;
- **ConfigSidebar**: barra laterale che contiene il menu per la configurazione del sistema e l'apertura del dialog di login.

4.1.1.3 Tracciamento dei requisiti

Tabella 4.1: Tracciamento dei requisiti per il componente AppLayout

ID	Componente
RF.O.1	LoginDialog
RF.O.1.1	LoginDialog
RF.O.1.2	LoginDialog
RF.O.2	LoginDialog
RF.O.60	ConfigSidebar
RF.D.61	ConfigSidebar
RF.OP.62	ConfigSidebar

4.1.2 ChatView

4.1.2.1 Descrizione

ChatView è la pagina di generazione del $prompt_c$. Espone le seguenti funzionalità:

- Selezione di un *dizionario dati*$_G$;
- Visualizzazione di un'anteprima del dizionario dati;
- Selezione della lingua di richiesta;
- Selezione di un *DBMS*$_G$;
- Invio di una richiesta al ChatBOT;
- Visualizzazione della risposta del ChatBOT;
- Copia del prompt generato;
- Visualizzazione del messaggio di *debug*$_G$ associato a ciascun prompt;
- Download di un file di *log*$_G$;
- Eliminazione del contenuto della chat.

4.1.2.2 Sottocomponenti

ChatView è composto dai seguenti sottocomponenti:

- **ChatMessage**: rappresenta i messaggi inviati e/o restituiti all'interno della chat;
- **ChatDeleteBtn**: pulsante per eliminare il contenuto della chat;
- **DictPreview**: mostra un'anteprima del dizionario dati selezionato.

4.1.2.3 Tracciamento dei requisiti

Tabella 4.2: Tracciamento dei requisiti per il componente ChatView

ID	Componente
RF.O.3	ChatView
RF.O.4	ChatView
RF.O.5	ChatView
RF.O.6	ChatMessage
RF.D.7	ChatView
RF.O.8	ChatMessage
RF.O.9	ChatView
RF.O.9.1	ChatView
RF.O.10	ChatView
RF.O.10.1	ChatView
RF.O.11	ChatView
Continua nella prossima pagina	

Tabella 4.2: Tracciamento dei requisiti per il componente ChatView (continua)

ID	Componente
RF.O.14	DictPreview
RF.O.14.1	DictPreview
RF.O.14.2	DictPreview
RF.O.14.3	DictPreview
RF.O.14.3.1	DictPreview
RF.O.14.3.1.1	DictPreview
RF.O.14.3.1.2	DictPreview
RF.O.25	ChatView
RF.O.25.1	ChatMessage
RF.O.25.1.1	ChatMessage
RF.O.25.1.2	ChatMessage
RF.O.26	ChatView
RF.O.27	ChatDeleteBtn
RF.O.35	ChatMessage
RF.O.36	ChatView
RF.O.41	ChatMessage
RF.O.45	ChatMessage
RF.O.47	ChatView
RF.O.48	ChatView
RF.O.49	ChatView
RF.O.50	ChatView
RF.O.51	ChatView
RF.D.52	ChatView

4.1.3 DictionariesListView

4.1.3.1 Descrizione

DictionariesListView è la pagina di gestione $CRUD_G$ dei dizionari dati. Espone le seguenti funzionalità:

- Visualizzazione della lista dei *dizionari dati*$_G$;
- Creazione, modifica e cancellazione di un dizionario dati;
- Download di un dizionario dati;
- Persistenza degli indici associati ai dizionari dati.

4.1.3.2 Sottocomponenti

DictionariesListView è composto dai seguenti sottocomponenti:

- **CreateUpdateDictionaryModal**: finestra per la creazione e modifica di un dizionario dati.

4.1.3.3 Tracciamento dei requisiti

Tabella 4.3: Tracciamento dei requisiti per il componente DictionariesListView

ID	Componente
RF.O.9	DictionariesListView
RF.O.9.1	DictionariesListView
RF.O.10	DictionariesListView
RF.O.10.1	DictionariesListView
RF.O.10.3	DictionariesListView
RF.O.13	CreateUpdateDictionaryModal
RF.O.15	CreateUpdateDictionaryModal
RF.O.16	CreateUpdateDictionaryModal
RF.O.17	CreateUpdateDictionaryModal
RF.O.18	DictionariesListView
RF.O.19	CreateUpdateDictionaryModal
RF.O.20	CreateUpdateDictionaryModal
RF.O.21	DictionariesListView
RF.O.24	DictionariesListView
RF.O.28	CreateUpdateDictionaryModal
RF.O.29	CreateUpdateDictionaryModal
RF.O.30	CreateUpdateDictionaryModal
RF.O.31	CreateUpdateDictionaryModal
Continua nella prossima pagina	

Tabella 4.3: Tracciamento dei requisiti per il componente DictionariesListView (continua)

ID	Componente
RF.O.31.1	CreateUpdateDictionaryModal
RF.O.31.2	CreateUpdateDictionaryModal
RF.O.32	CreateUpdateDictionaryModal
RF.O.33	CreateUpdateDictionaryModal
RF.O.34	CreateUpdateDictionaryModal
RF.O.36	DictionariesListView
RF.O.37	DictionariesListView
RF.O.39	CreateUpdateDictionaryModal
RF.D.40	CreateUpdateDictionaryModal
RF.O.55	CreateUpdateDictionaryModal
RF.O.56	CreateUpdateDictionaryModal
RF.O.57	CreateUpdateDictionaryModal

4.1.4 MenuSidebar

4.1.4.1 Descrizione

MenuSidebar è la barra laterale che consente di navigare tra le seguenti pagine:

- ChatView;
- DictionariesListView.

4.1.4.2 Sottocomponenti

MenuSidebar è composto dai seguenti sottocomponenti:

- **AppMenu**: rappresenta il menu di navigazione principale;
- **AppFooter**: rappresenta il footer dell'applicazione. Include il seguente componente (puramente visivo):
  - **AppLogo**: logo dell'applicazione.

4.1.5 AppMenu

4.1.5.1 Descrizione

AppMenu è il menu di navigazione principale dell'applicazione.

4.1.5.2 Sottocomponenti

AppMenu è composto dai seguenti sottocomponenti:

- **AppMenuItem**: rappresenta un singolo elemento del menu di navigazione.

4.1.6 ChatMessage

4.1.6.1 Descrizione

ChatMessage è il componente che rappresenta i messaggi inviati e/o restituiti all'interno della chat.

4.1.6.2 Sottocomponenti

AppMenu è composto dai seguenti sottocomponenti:

- **StringDataModal**: componente per visualizzare stringhe di testo all'interno di una finestra modale.

4.1.7 StringDataModal

4.1.7.1 Descrizione

StringDataModal consente di visualizzare stringhe di testo all'interno di una finestra modale.

4.1.7.2 Sottocomponenti

StringDataModal è composto dai seguenti sottocomponenti:

- **DebugMessage**: rappresenta un messaggio di *debug*$_G$.

4.1.7.3 Tracciamento dei requisiti

Tabella 4.4: Tracciamento dei requisiti per il componente DebugMessage

ID	Componente
RF.O.22	DebugMessage
RF.O.23	DebugMessage
RF.O.24	DebugMessage

4.2 Back-end

4.2.1 get_all_dictionaries

4.2.1.1 Descrizione

La funzione `get_all_dictionaries` recupera tutti i dizionari dal database e restituisce una risposta strutturata.

4.2.1.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionariesResponseDto`;
- **Nome**: `getAllDictionaries`;
- **Dependency Injection**:
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.

4.2.1.3 Implementazione

- Recupera tutti i dizionari dal database utilizzando `crud.get_all_dictionaries(db)`;
- Restituisce un oggetto `DictionariesResponseDto` con i dati dei dizionari e lo stato `ResponseStatusEnum.OK`.

4.2.2 get_dictionary

4.2.2.1 Descrizione

La funzione `get_dictionary` recupera un dizionario specifico dal database utilizzando l'ID fornito e restituisce una risposta strutturata.

4.2.2.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/{id}`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionaryResponseDto`;
- **Nome**: `getDictionary`;
- **Dependency Injection**:
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario da recuperare.

4.2.2.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`;
- Se il dizionario viene trovato, restituisce un oggetto `DictionaryResponseDto` con i dati del dizionario e lo stato `ResponseStatusEnum.OK`;
- Se il dizionario non viene trovato, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`.

4.2.3 get_dictionary_file

4.2.3.1 Descrizione

La funzione `get_dictionary_file` recupera un file di dizionario specifico dal database utilizzando l'ID fornito e restituisce una risposta strutturata.

4.2.3.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/{id}/file`;
- **Tags**: `[tag]`;
- **Nome**: `getDictionaryFile`;
- **Dependency Injection**:
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario di cui recuperare il file.

4.2.3.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`;
- Se il dizionario viene trovato, restituisce un `FileResponse` con il nome del file generato tramite `__generate_schema_file_name(id)`;
- Se il dizionario non viene trovato, restituisce un oggetto `ResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`.

4.2.4 get_dictionary_preview

4.2.4.1 Descrizione

La funzione `get_dictionary_preview` recupera un'anteprima di un dizionario specifico dal database utilizzando l'ID fornito e restituisce una risposta strutturata.

4.2.4.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/{id}/dictionary-preview`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionaryResponseDto`;
- **Nome**: `getDictionaryPreview`;
- **Dependency Injection**:
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario di cui recuperare l'anteprima.

4.2.4.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`.
- Se il dizionario viene trovato:
  - Estrae un sotto-schema del dizionario utilizzando `SchemaMultiExtractor.-extract_preview(id)`;
  - Crea un oggetto `DictionaryPreviewDto` con i dettagli del sotto-schema;
  - Restituisce un oggetto `DictionaryResponseDto` con i dati dell'anteprima del dizionario e lo stato `ResponseStatusEnum.OK`;
- Se il dizionario non viene trovato, restituisce un oggetto `ResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`.

4.2.5 create_dictionary

4.2.5.1 Descrizione

La funzione `create_dictionary` permette di creare un nuovo dizionario nel database utilizzando i dati forniti e un file di schema. La funzione include controlli di validità e verifica l'unicità del nome del dizionario.

4.2.5.2 Dettagli dell'Endpoint

- **HTTP Method**: `POST`;
- **Endpoint**: `/`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionaryResponseDto`;
- **Dependencies**:

  - `Depends(JwtBearer())`: Assicura che l'endpoint sia protetto da un meccanismo di autenticazione JWT.
- **Nome**: `createDictionary`;
- **Dependency Injection**:
  - **file (Annotated[UploadFile, File()])**: Il file di schema del dizionario da caricare;
  - **dictionary (DictionaryDto)**: I dati del dizionario, ottenuti tramite `Depends()`;
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.

4.2.5.3 Implementazione

- Verifica che il nome e la descrizione del dizionario, oltre al file, siano presenti;
- Recupera un dizionario esistente con lo stesso nome utilizzando `crud.get_dictionary_by_name(db, name=dictionary.name)`;
- Se il dizionario esiste già, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.CONFLICT`;
- Crea un nuovo dizionario utilizzando `crud.create_dictionary(db=db, dictionary=dictionary)`;
- Legge il contenuto del file e valida il formato dello schema del dizionario utilizzando `DictionaryValidator.validate(Utils.string_to_json(content))`;
- Se lo schema non è valido, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`;
- Scrive il contenuto del file in un nuovo file utilizzando `aiofiles.open(__generate_schema_file_name(new_dic.id), "wb")`;
- Crea un indice `txtai` utilizzando `manager.create_index(new_dic.id)`;
- Restituisce un oggetto `DictionaryResponseDto` con i dati del nuovo dizionario e lo stato `ResponseStatusEnum.OK`;
- Se il file non è presente, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`;
- Se il nome e la descrizione del dizionario non sono presenti, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`.

4.2.6 update_dictionary_file

4.2.6.1 Descrizione

La funzione `update_dictionary_file` permette di aggiornare il file di schema di un dizionario esistente nel database utilizzando l'ID fornito e un nuovo file di schema. La funzione include controlli di validità dello schema.

4.2.6.2 Dettagli dell'Endpoint

- **HTTP Method**: `PUT`;
- **Endpoint**: `/{id}/file`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionaryResponseDto`;
- **Dependencies**:
  - `Depends(JwtBearer())`: Assicura che l'endpoint sia protetto da un meccanismo di autenticazione JWT.
- **Nome**: `updateDictionaryFile`;
- **Dependency Injection**:
  - **file (Annotated[UploadFile, File()])**: Il nuovo file di schema del dizionario da caricare;
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario di cui aggiornare il file.

4.2.6.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`;
- Se il dizionario non viene trovato, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`;
- Legge il contenuto del file e valida il formato dello schema del dizionario utilizzando `DictionaryValidator.validate(Utils.string_to_json(content))`;
- Se lo schema non è valido, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`;
- Scrive il contenuto del file in un nuovo file utilizzando `aiofiles.open(__generate_schema_file_name(id), "wb")`;
- Aggiorna l'indice `txtai` utilizzando `manager.create_index(found_dic.id)`;
- Restituisce un oggetto `DictionaryResponseDto` con i dati del dizionario aggiornato e lo stato `ResponseStatusEnum.OK`.

4.2.7 update_dictionary_metadata

4.2.7.1 Descrizione

La funzione `update_dictionary_metadata` permette di aggiornare i metadati di un dizionario esistente nel database utilizzando l'ID fornito e un oggetto `DictionaryDto`. La funzione include controlli di validità e verifica l'unicità del nome del dizionario.

4.2.7.2 Dettagli dell'Endpoint

- **HTTP Method**: `PUT`;
- **Endpoint**: `/{id}`;
- **Tags**: `[tag]`;
- **Response Model**: `DictionaryResponseDto`;
- **Dependencies**:
  - `Depends(JwtBearer())`: Assicura che l'endpoint sia protetto da un meccanismo di autenticazione JWT.
- **Nome**: `updateDictionaryMetadata`;
- **Dependency Injection**:
  - **dictionary (DictionaryDto)**: I nuovi metadati del dizionario;
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario di cui aggiornare i metadati.

4.2.7.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`;
- Se il dizionario non viene trovato, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`;
- Se il nome o la descrizione del dizionario sono assenti, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`;
- Verifica se esiste un altro dizionario con lo stesso nome utilizzando `crud.get_dictionary_by_name(db, dictionary.name)`;
- Se esiste un altro dizionario con lo stesso nome e l'ID è diverso, restituisce un oggetto `DictionaryResponseDto` con `data=None`, un messaggio di errore e lo stato `ResponseStatusEnum.CONFLICT`;
- Aggiorna il dizionario utilizzando `crud.update_dictionary(db, dictionary)`;
- Restituisce un oggetto `DictionaryResponseDto` con i dati del dizionario aggiornato e lo stato `ResponseStatusEnum.OK`.

4.2.8 delete_dictionary

4.2.8.1 Descrizione

La funzione `delete_dictionary` permette di eliminare un dizionario esistente dal database utilizzando l'ID fornito. La funzione elimina anche il file di schema associato e l'indice `txtai`.

4.2.8.2 Dettagli dell'Endpoint

- **HTTP Method**: `DELETE`;
- **Endpoint**: `/{id}`;
- **Tags**: `[tag]`;
- **Response Model**: `ResponseDto`;
- **Nome**: `deleteDictionary`;
- **Dependency Injection**:
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.
- **Parametri**:
  - **id (int)**: ID del dizionario da eliminare.

4.2.8.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, id)`;
- Se il dizionario non viene trovato, restituisce un oggetto `ResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`;
- Elimina il dizionario dal database utilizzando `crud.delete_dictionary(db, id)`;
- Se il file di schema esiste, lo elimina utilizzando `os.remove(__generate_schema_file_name(id))`;
- Elimina l'indice `txtai` utilizzando `manager.delete_index(found_dic.id)`;
- Restituisce un oggetto `ResponseDto` con lo stato `ResponseStatusEnum.OK`.

4.2.9 login

4.2.9.1 Descrizione

La funzione `login` permette di autenticare un utente utilizzando le credenziali fornite. Se le credenziali sono corrette, viene restituito un token JWT.

4.2.9.2 Dettagli dell'Endpoint

- **HTTP Method**: `POST`;
- **Endpoint**: `/`;
- **Tags**: `[tag]`;
- **Response Model**: `AuthResponseDto`;
- **Nome**: `login`;
- **Dependency Injection**:
  - **data (LoginDto)**: I dati di login dell'utente;

- **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.

4.2.9.3 Implementazione

- Recupera l'utente dal database utilizzando `crud.get_user_by_username(db, data-.username)`;
- Se l'utente non viene trovato, restituisce un oggetto `AuthResponseDto` con `data=-None` e lo stato `ResponseStatusEnum.NOT_FOUND`;
- Se la password non è corretta, restituisce un oggetto `AuthResponseDto` con `data-=None`, un messaggio di errore e lo stato `ResponseStatusEnum.BAD_CREDENTIAL`;
- Genera un token JWT utilizzando `JwtHandler.sign(query.id)`;
- Restituisce un oggetto `AuthResponseDto` con il token generato e lo stato `Response-StatusEnum.OK`.

4.2.10 generate_prompt

4.2.10.1 Descrizione

La funzione `generate_prompt` genera un prompt utilizzando un dizionario specifico e una query forniti dall'utente. La funzione verifica la validità dei dati di input e restituisce il prompt generato.

4.2.10.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/`;
- **Tags**: `[tag]`;
- **Response Model**: `StringDataResponseDto`;
- **Nome**: `generatePrompt`;
- **Dependency Injection**:
  - **dictionary_id (Annotated[int, Query(alias="dictionaryId")])**: L'ID del dizionario da utilizzare;
  - **query (str)**: La query fornita dall'utente;
  - **dbms (str)**: Il sistema di gestione del database (DBMS);
  - **lang (str)**: La lingua da utilizzare;
  - **db (Session)**: Dipendenza risolta tramite `Depends(get_db)`.

4.2.10.3 Implementazione

- Recupera il dizionario dal database utilizzando `crud.get_dictionary_by_id(db, dictionary_id)`;
- Se il dizionario non viene trovato, restituisce un oggetto `StringDataResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`;

- Se la query è vuota o nulla, restituisce un oggetto `StringDataResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.BAD_REQUEST`;
- Carica l'indice del dizionario utilizzando `manager.load_index(found_dic.id)`;
- Genera il prompt utilizzando `manager.prompt_generator(found_dic.id, query, lang, dbms, activate_log=True)`;
- Restituisce un oggetto `StringDataResponseDto` con il prompt generato e lo stato `ResponseStatusEnum.OK`.

4.2.11 generate_prompt_debug

4.2.11.1 Descrizione

La funzione `generate_prompt_debug` restituisce il contenuto del file di log per scopi di debug. Se il file di log non viene trovato, restituisce un messaggio di errore.

4.2.11.2 Dettagli dell'Endpoint

- **HTTP Method**: `GET`;
- **Endpoint**: `/debug`;
- **Tags**: `[tag]`;
- **Response Model**: `StringDataResponseDto`;
- **Nome**: `generatePromptDebug`.

4.2.11.3 Implementazione

- Tenta di aprire il file di log situato in `/opt/chatsql/logs/chatsql_log.txt`;
- Se il file viene trovato, legge il contenuto e restituisce un oggetto `StringDataResponseDto` con i dati del file e lo stato `ResponseStatusEnum.OK`;
- Se il file non viene trovato, restituisce un oggetto `StringDataResponseDto` con un messaggio di errore e lo stato `ResponseStatusEnum.NOT_FOUND`.

4.2.12 Tracciamento dei requisiti

Tabella 4.5: Tracciamento dei requisiti back-end

ID	Route
RF.O.1	`login`
RF.O.2	`login`
RF.O.3	`generate_prompt`
RF.O.6	`generate_prompt`, `generate_prompt_with_debug`

Continua nella prossima pagina

Tabella 4.5: Tracciamento dei requisiti back-end (continua)

ID	Route
RF.O.9	`get_all_dictionaries`
RF.O.9.1	`get_all_dictionaries`
RF.O.10	`get_all_dictionaries`, `get_dictionary`
RF.O.10.1	`get_all_dictionaries`, `get_dictionary`
RF.O.10.2	`get_all_dictionaries`, `get_dictionary`
RF.O.10.3	`get_all_dictionaries`, `get_dictionary`
RF.O.11	`generate_prompt`, `generate_prompt_with_debug`
RF.O.13	`create_dictionary`
RF.O.14	`get_dictionary_preview`
RF.O.14.1	`get_dictionary_preview`
RF.O.14.2	`get_dictionary_preview`
RF.O.14.3	`get_dictionary_preview`
RF.O.14.3.1	`get_dictionary_preview`
RF.O.14.3.1.1	`get_dictionary_preview`
RF.O.14.3.1.2	`get_dictionary_preview`
RF.O.15	`create_dictionary`
RF.O.16	`create_dictionary`
RF.O.17	`create_dictionary`
RF.O.18	`delete_dictionary`
RF.O.19	`create_dictionary`, `update_dictionary_file`
RF.O.20	`update_dictionary_file`
RF.O.21	`delete_dictionary`
RF.O.22	`generate_prompt_with_debug`
RF.O.28	`create_dictionary`, `update_dictionary_file`
RF.O.29	`update_dictionary_metadata`
RF.O.30	`update_dictionary_metadata`
RF.O.31	`create_dictionary`, `update_dictionary_metadata`
RF.O.31.1	`create_dictionary`, `update_dictionary_metadata`

Continua nella prossima pagina

Tabella 4.5: Tracciamento dei requisiti back-end (continua)

ID	Route
RF.O.31.2	`create_dictionary`, `update_dictionary_metadata`
RF.O.32	`create_dictionary`, `update_dictionary_metadata`
RF.O.33	`create_dictionary`, `update_dictionary_file`
RF.O.34	`create_dictionary`, `update_dictionary_file`
RF.O.35	`generate_prompt`, `generate_prompt_with_debug`
RF.O.37	`get_dictionary_file`
RF.O.39	`create_dictionary`, `update_dictionary_file`
RF.D.40	`create_dictionary`, `update_dictionary_file`
RF.O.41	`generate_prompt`, `generate_prompt_with_debug`
RF.OP.42	`generate_prompt`, `generate_prompt_with_debug`
RF.O.45	`generate_prompt`, `generate_prompt_with_debug`
RF.O.46	`generate_prompt_with_debug`
RF.O.47	`generate_prompt`, `generate_prompt_with_debug`
RF.D.52	`generate_prompt`, `generate_prompt_with_debug`
RF.O.55	`create_dictionary`, `update_dictionary_file`
RF.O.56	`create_dictionary`, `update_dictionary_file`
RF.O.57	`create_dictionary`, `update_dictionary_file`

5 Progettazione di dettaglio

5.1 Front-end

5.1.1 AppLayout

Elementi chiave

- Layout Wrapper: aggiorna lo stile dell'applicazione in base alle configurazioni del sistema;
- Outside Click Listener: i menu vengono chiusi quando viene effettuato un clic al di fuori di questi elementi;
- Scroll To Top: pulsante per tornare rapidamente all'inizio della pagina;
- Toast Message: messaggio di notifica per fornire feedback all'utente.

5.1.2 ChatView

Elementi chiave

- Selected Dictionary Name: nome ed estensione del *dizionario dati*$_G$ selezionato dall'utente;
- Toggle Select View: pulsante per visualizzare o nascondere il form di selezione del dizionario dati, della lingua e del *DBMS*$_G$;
- Select Dictionary Dropdown: menu a discesa per selezionare un dizionario dati;
- Toggle Details: pulsante per visualizzare o nascondere l'anteprima di un dizionario dati;
- Select DBMS Dropdown: menu a discesa per selezionare un DBMS;
- Select Language Dropdown: menu a discesa per selezionare una lingua;
- Chat Container: contenitore per i messaggi della chat;
- Chat Textarea: campo di testo per inserire una richiesta da inviare al ChatBOT;
- Request Button: pulsante per inviare la richiesta;
- Loading Spinner: indica che la generazione del *prompt*$_G$ è ancora in corso.

5.1.3 DictionariesListView

Elementi chiave

- Create Dictionary Button: pulsante per aprire la finestra di inserimento di un *dizionario dati*$_G$;
- Dictionary Data Table: tabella contenente i seguenti campi:
  - Dictionary Name: nome del dizionario dati;
  - Dictionary Description: descrizione del dizionario dati;

- Update Metadata Button: pulsante per aprire la finestra di modifica dei metadati (nome e descrizione);
- Update File Button: pulsante per aprire la finestra di modifica del file;
- Download File Button: pulsante per scaricare il file relativo a un dizionario dati;
- Delete Dictionary Button: pulsante per eliminare un dizionario dati.

5.1.4 LoginDialog

Elementi chiave

- Dialog: finestra per l'autenticazione dell'utente. La finestra interattiva appare sopra il contenuto principale dell'interfaccia (in overlay);
- Login Form: modulo per l'inserimento delle credenziali di accesso;
- Username Input: campo di testo per l'inserimento dello username;
- Password Input: campo di testo per l'inserimento della password;
- V-Model: crea un binding bidirezionale tra i campi di input e le rispettive variabili;
- Login Button: pulsante per richiedere l'accesso all'area di amministrazione;
- Status Message: messaggio per notificare all'utente che:
  - Il login è avvenuto con successo;
  - L'utente non è autorizzato ad accedere all'area di amministrazione;
  - Le credenziali inserite non sono corrette;
  - Il server non è raggiungibile.

5.1.5 ChatMessage

Props

- Message: testo del messaggio;
- IsSent: flag per indicare se il messaggio è stato inviato o ricevuto dall'utente;
- Debug (opzionale): contenuto del messaggio di *debug*$_G$.

Elementi chiave

- Message Card: contenitore per il messaggio. La disposizione e il colore del contenitore variano in base al mittente del messaggio;
- Avatar: icona del mittente. L'avatar è posizionato a sinistra o a destra del messaggio in base al mittente;
- Message Text: testo del messaggio;
- Copy Button: pulsante per copiare il testo del messaggio negli Appunti di sistema;

- Debug Button: pulsante per aprire la finestra modale contenente il messaggio di debug. Il pulsante è disponibile solo per il profilo Tecnico.

5.1.6 StringDataModal

Elementi chiave

- Modal: finestra modale responsive;
- Close Dialog: pulsante per chiudere la finestra modale;
- String Data: stringa di testo da visualizzare nel dialog.

5.1.7 DebugMessage

Props

- Message: testo del messaggio.

Elementi chiave

- Message Text: testo del messaggio;
- Download Button: pulsante per scaricare un file di log_G.

5.1.8 DictPreview

Props

- DetailsVisible: flag per visualizzare o nascondere l'anteprima del *dizionario dati*$_G$;
- DictionaryPreview: anteprima del dizionario dati, contenente le seguenti informazioni:
  - Database Name: nome del dizionario dati;
  - Database Description: descrizione del dizionario dati;
  - Lista delle tabelle:
    - Table Name: nome della tabella;
    - Table Description: descrizione della tabella.

Elementi chiave

- Preview Card: contenitore per l'anteprima del dizionario dati;
- Expand Button: pulsante per aumentare la dimensione della Preview Card. Il pulsante di espansione migliora l'accessibilità e l'usabilità;
- Collapse Button: pulsante per ridurre la dimensione della Preview Card.

5.1.9 ChatDeleteBtn

Props

- Messages: contenuto della chat da eliminare;
- Loading: flag per indicare se il ChatBOT sta elaborando un nuovo messaggio.

Emits

- ClearMessages: evento per notificare il componente genitore che i messaggi devono essere cancellati.

Elementi chiave

- Toggle: il pulsante di cancellazione deve essere nascosto se:
  - Non ci sono messaggi da cancellare;
  - Il ChatBOT sta elaborando un nuovo messaggio.

5.1.10 CreateUpdateDictionaryModal

Elementi chiave

- Close Dialog: pulsante per chiudere la finestra modale;
- Dictionary Name Input: campo di testo per inserire o modificare il nome di un *dizionario dati*$_G$;
- Dictionary Description Input: campo di testo per inserire o modificare la descrizione di un dizionario dati;
- File Uploader: componente per caricare il file relativo a un dizionario dati;
- Clear File Button: pulsante per annullare il caricamento del file;
- Submit Button: pulsante per richiedere il salvataggio dei dati;
- Loading Spinner: indica che il salvataggio del dizionario dati e del rispettivo *indice*$_G$ è ancora in corso.

5.1.11 AppTopbar

Elementi chiave

- Logo: logo dell'applicazione, affiancato dal nome del progetto (per migliorare l'accessibilità);
- Open Main Nav Button: pulsante per aprire il menu di navigazione principale;
- Open Config Menu Button: pulsante per aprire il menu di configurazione del sistema;
- Open Settings Button: pulsante per aprire la barra laterale di impostazioni;
- Open Login Dialog Button: pulsante per aprire la finestra di autenticazione;

- Logout Button: pulsante per effettuare il logout;
- Confirmation Dialog: finestra per confermare o rifiutare il logout;
- Outside Click Listener: i menu vengono chiusi quando viene effettuato un clic al di fuori di questi elementi.

5.1.12 MenuSidebar

Elementi chiave

- Close Main Nav Button: pulsante per chiudere il menu di navigazione principale;
- App Menu: menu di navigazione principale;
- App Footer: footer dell'applicazione.

5.1.13 ConfigSidebar

Elementi chiave

- Decrement Scale Button: pulsante per diminuire la dimensione degli elementi nell'interfaccia;
- Increment Scale Button: pulsante per ingrandire la dimensione degli elementi nell'interfaccia;
- Change Theme Switch: interruttore per alternare il tema dell'interfaccia tra la modalità chiara e quella scura;
- Change Language Dropdown: menu a discesa per selezionare la lingua dell'interfaccia.

5.1.14 AppMenu

Elementi chiave

- Layout Menu: contenitore per il menu di navigazione principale;
- Menu Items: voci di menu per accedere alle diverse sezioni dell'applicazione;
- Menu Icon: stringa contenente la classe da applicare all'icona della voce di menu;
- Menu Label: stringa contenente l'etichetta della voce di menu;
- Menu Link: stringa contenente il percorso della voce di menu.

5.1.15 AppMenuItem

Props

- Item: oggetto che rappresenta una voce di menu;
- Index: indice della voce di menu;
- Root: flag per indicare se la voce di menu è la radice del menu di navigazione;

- Parent Item Key: chiave per identificare la voce di menu genitore.

Elementi chiave

- Menu Item: contenitore per una voce di menu;
- Menu Icon: icona per identificare la voce di menu;
- Menu Label: etichetta per identificare la voce di menu;
- Menu Link: collegamento esterno (non gestito da Vue Router) o interno (gestito da Vue Router). Un link può essere anche un elemento cliccabile per aprire un sotto-menu;
- Sub Menu: contenitore per le voci di menu figlie.

5.1.16 AppLogo

Props

- Width: variabile per impostare la larghezza del logo;
- Height: variabile per impostare l'altezza del logo;
- Path: variabile per impostare il percorso dell'immagine.

Elementi chiave

- Filter: funzionalità per invertire il colore del logo da bianco a nero (in base al tema attivo).

5.1.17 AppFooter

Elementi chiave

- Logo: logo dell'applicazione;
- Version: numero di versione dell'applicazione;
- Copyright section: sezione contenente le seguenti informazioni:
  - Year: anno corrente;
  - Group Name: nome del team di sviluppo;
  - Copyright: prova di diritto d'autore.

5.2 Back-end

5.2.1 Database

Il diagramma E-R (entità-relazione) riportato a seguito illustra la struttura e le relazioni all'interno del database.

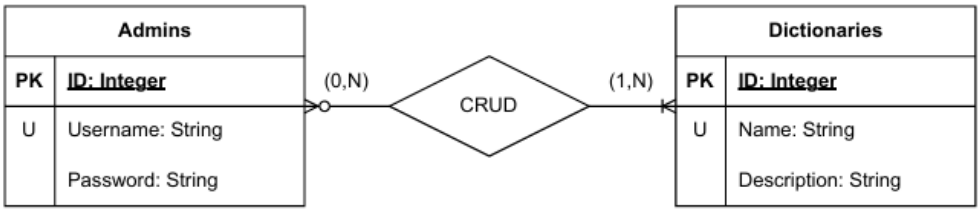

Figura 5.1: Diagramma E-R dell'interazione tra Admins (Tecnici) e *dizionari dati*$_G$ per le operazioni *CRUD*$_G$

5.2.2 Descrizione

Il diagramma utilizza la notazione standard per le descrizioni dei database relazionali; le entità sono rappresentate da rettangoli, mentre le relazioni sono le linee che collegano le entità. All'interno del diagramma sono presenti le seguenti sigle:

- **PK**: chiave primaria;
- **U**: attributi, o campi, della tabella.

Le linee terminano con dei simboli che ne indicano la cardinalità. Le tabelle sono inoltre collegate a un rombo che rappresenta la relazione tra di esse: questa relazione non è una tabella, ma serve solo a scopo illustrativo.

5.2.2.1 Tabella Admins La tabella **Admins** contiene i seguenti campi:

- **PK ID**: un identificatore unico per ogni amministratore, di tipo integer;
- **U Username**: il nome utente dell'amministratore, di tipo string;
- **Password**: la password dell'amministratore, di tipo string.

Ogni entry della tabella descrive un Admin, cioè un Tecnico che può eseguire operazioni *CRUD*$_G$ sui dizionari dati.

5.2.2.2 Tabella Dictionaries La tabella **Dictionaries** include i seguenti campi:

- **PK ID**: un identificatore unico per ogni dizionario, di tipo integer;
- **U Name**: il nome del dizionario, di tipo string;
- **Description**: la descrizione del dizionario, di tipo string.

5.2.2.3 Relazione Il diagramma mostra anche le operazioni CRUD (Create, Read, Update, Delete), ossia le interazioni tra i Tecnici e i *dizionari dati*$_G$.

- Relazione **0-N**: un admin può eseguire operazioni CRUD su 0 o N dizionari;

- Relazione **1-N** : un dizionario dati può essere gestito da 1 o N admin.