

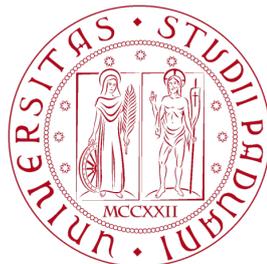
ARGO

Specifica Tecnica

Gruppo Argo – Progetto ChatSQL

Informazioni sul documento

Versione	1.0.0
Approvazione	Sebastiano Lewental
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo Argo



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Registro delle modifiche

Ver.	Data	Redazione	Verifica	Descrizione
1.0.0	2024-09-14	Sebastiano Lewental	Sebastiano Lewental	Approvazione e rilascio del documento
0.1.2	2024-09-12	Raul Pianon, Riccardo Cavalli	Marco Cristo, Tommaso Stocco, Mattia Zecchinato	Aggiornamento sezione classi
0.1.1	2024-09-10	Riccardo Cavalli, Raul Pianon	Marco Cristo, Mattia Zecchinato	Aggiornamento diagramma delle classi
0.1.0	2024-09-08	Riccardo Cavalli, Marco Cristo	Raul Pianon	Revisione completa del documento
0.0.13	2024-09-05	Riccardo Cavalli	Raul Pianon	Aggiornamento sezione DTO
0.0.12	2024-08-31	Raul Pianon	Riccardo Cavalli	Stesura sezione DTO
0.0.11	2024-08-25	Mattia Zecchinato, Raul Pianon	Riccardo Cavalli, Marco Cristo	Stesura sezione design pattern
0.0.10	2024-08-20	Raul Pianon	Martina Dall'Amico, Mattia Zecchinato	Stesura sezione classi
0.0.9	2024-08-13	Marco Cristo	Martina Dall'Amico, Mattia Zecchinato	Stesura progettazione ad alto livello back-end
0.0.8	2024-08-10	Martina Dall'Amico, Raul Pianon	Mattia Zecchinato	Caricamento diagrammi delle classi
0.0.7	2024-08-03	Raul Pianon	Riccardo Cavalli, Sebastiano Lewental, Mattia Zecchinato	Aggiunta sezione architettura esagonale
Continua nella prossima pagina				



Ver.	Data	Redazione	Verifica	Descrizione
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, Marco Cristo	Raul Pianon	Completata sezione scelte tecnologiche
0.0.1	2024-07-20	Riccardo Cavalli	Tommaso Stocco, Mattia Zecchinato	Prima stesura del documento

Indice

1	Introduzione	11
1.1	Scopo del documento	11
1.2	Riferimenti	11
1.2.1	Riferimenti normativi	11
1.2.2	Riferimenti informativi	11
1.3	Glossario	14
2	Tecnologie utilizzate	15
2.1	Back-end	15
2.1.1	FastAPI	15
2.1.2	Python	15
2.1.3	txtai	16
2.1.4	SQLAlchemy	16
2.1.5	JWT	16
2.1.6	pytest	17
2.2	Front-end	17
2.2.1	Vue.js	17
2.2.2	Axios	18
2.2.3	TypeScript	18
2.2.4	PrimeVue	18
2.2.5	Jest	19
2.3	Altri strumenti e tecnologie	20
2.3.1	Docker	20
2.3.2	SQLite	20
2.3.3	Hugging Face	21
2.3.4	ChatGPT	21
2.3.5	LMSYS Chatbot Arena	21
2.3.6	LM Studio	21
2.4	Panoramica delle tecnologie	22
2.4.1	Framework	22
2.4.2	Linguaggi	23
2.4.3	Librerie	23
2.4.4	Strumenti	24
3	Architettura	26
3.1	Introduzione	26
3.2	Assemblaggio dei componenti	27
3.3	Struttura del sistema	27
3.3.1	Front-end	27
3.3.2	Back-end	30
4	Progettazione ad alto livello dei componenti	33
4.1	Front-end	33
4.1.1	AppLayout	33
4.1.1.1	Descrizione	33
4.1.1.2	Sottocomponenti	33
4.1.1.3	Tracciamento dei requisiti	33



4.1.2	ChatView	34
4.1.2.1	Descrizione	34
4.1.2.2	Sottocomponenti	34
4.1.2.3	Tracciamento dei requisiti	34
4.1.3	DictionariesListView	36
4.1.3.1	Descrizione	36
4.1.3.2	Sottocomponenti	36
4.1.3.3	Tracciamento dei requisiti	36
4.1.4	MenuSidebar	37
4.1.4.1	Descrizione	37
4.1.4.2	Sottocomponenti	37
4.1.5	AppMenu	38
4.1.5.1	Descrizione	38
4.1.5.2	Sottocomponenti	38
4.1.6	ChatMessage	38
4.1.6.1	Descrizione	38
4.1.6.2	Sottocomponenti	38
4.1.7	StringDataModal	38
4.1.7.1	Descrizione	38
4.1.7.2	Sottocomponenti	38
4.1.7.3	Tracciamento dei requisiti	38
4.2	Back-end	39
4.2.1	get_all_dictionaries	39
4.2.1.1	Descrizione	39
4.2.1.2	Dettagli dell'endpoint	39
4.2.1.3	Implementazione	39
4.2.2	get_dictionary	39
4.2.2.1	Descrizione	39
4.2.2.2	Dettagli dell'endpoint	40
4.2.2.3	Implementazione	40
4.2.3	get_dictionary_file	40
4.2.3.1	Descrizione	40
4.2.3.2	Dettagli dell'endpoint	40
4.2.3.3	Implementazione	41
4.2.4	get_dictionary_preview	41
4.2.4.1	Descrizione	41
4.2.4.2	Dettagli dell'endpoint	41
4.2.4.3	Implementazione	41
4.2.5	create_dictionary	41
4.2.5.1	Descrizione	41
4.2.5.2	Dettagli dell'endpoint	42
4.2.5.3	Implementazione	42
4.2.6	update_dictionary_file	42
4.2.6.1	Descrizione	42
4.2.6.2	Dettagli dell'endpoint	42
4.2.6.3	Implementazione	43
4.2.7	update_dictionary_metadata	43
4.2.7.1	Descrizione	43
4.2.7.2	Dettagli dell'endpoint	43



4.2.7.3	Implementazione	44
4.2.8	delete_dictionary	44
4.2.8.1	Descrizione	44
4.2.8.2	Dettagli dell'endpoint	44
4.2.8.3	Implementazione	44
4.2.9	login	44
4.2.9.1	Descrizione	44
4.2.9.2	Dettagli dell'endpoint	45
4.2.9.3	Implementazione	45
4.2.10	generate_prompt	45
4.2.10.1	Descrizione	45
4.2.10.2	Dettagli dell'endpoint	45
4.2.10.3	Implementazione	46
4.2.11	generate_prompt_with_debug	46
4.2.11.1	Descrizione	46
4.2.11.2	Dettagli dell'endpoint	46
4.2.11.3	Implementazione	46
4.2.12	Tracciamento dei requisiti	47
5	Progettazione di dettaglio front-end	49
5.1	Componenti	49
5.1.1	AppLayout	49
5.1.2	ChatView	49
5.1.3	DictionariesListView	49
5.1.4	LoginDialog	50
5.1.5	ChatMessage	50
5.1.6	StringDataModal	51
5.1.7	DebugMessage	51
5.1.8	DictPreview	51
5.1.9	ChatDeleteBtn	52
5.1.10	CreateUpdateDictionaryModal	52
5.1.11	AppTopbar	53
5.1.12	MenuSidebar	53
5.1.13	ConfigSidebar	53
5.1.14	AppMenu	54
5.1.15	AppMenuItem	54
5.1.16	AppLogo	54
5.1.17	AppFooter	55
5.2	Interfacce e tipi	55
5.2.1	OpenAPI	55
5.2.2	Wrapper	55
5.3	Gestione degli errori	56
6	Progettazione di dettaglio back-end	57
6.1	Diagramma delle classi	57
6.2	Descrizione delle classi	58
6.2.1	Introduzione	58
6.2.2	Ports - Incoming	58
6.2.2.1	AuthenticationUseCase	58



6.2.2.2	DictionaryUseCase	59
6.2.2.3	PromptUseCase	59
6.2.2.4	SchemaValidatorUseCase	60
6.2.2.5	Ports - Outcoming	60
6.2.2.6	EmbeddingsAbstractFactory	60
6.2.2.7	IndexManagerPort	61
6.2.2.8	PromptManagerPort	62
6.2.2.9	SearchAlgorithmPort	62
6.2.2.10	DbManagerAbstractFactory	63
6.2.2.11	AuthenticationRepository	63
6.2.2.12	DictionaryRepository	64
6.2.2.13	FileRepository	64
6.2.3	Servizi	65
6.2.3.1	AuthenicationService	65
6.2.3.2	DictionaryService	66
6.2.3.3	PromptManagerService	67
6.2.4	Adapters - Incoming	68
6.2.4.1	SchemaValidatorFactory	68
6.2.4.2	JsonSchemaValidatorAdapter	68
6.2.5	Outcoming - Adapters	69
6.2.5.1	DbManagerFactory	69
6.2.5.2	SqlAlchemyDbManagerFactory	69
6.2.5.3	SqlAlchemyAuthenticationRepositoryAdapter	70
6.2.5.4	SqlAlchemyDictionaryRepositoryAdapter	71
6.2.5.5	EmbeddingsManagerFactory	71
6.2.5.6	TxtaiEmbeddingsManagerFactory	72
6.2.5.7	TxtaiIndexManagerAdapter	73
6.2.5.8	TxtaiPromptManagerAdapter	74
6.2.5.9	TxtaiSearchAlgorithmAdapter	75
6.2.5.10	FileFactory	76
6.2.5.11	JsonFileAdapter	76
6.2.6	Database	77
6.2.6.1	Descrizione	77
6.2.6.2	Tabella Admins	77
6.2.6.3	Tabella Dictionaries	78
6.2.6.4	Relazione	78
6.2.7	Design pattern	78
6.2.7.1	Introduzione	78
6.2.7.2	Abstract Factory	78
6.2.7.3	Dependency Injection	79
6.2.7.4	Port e Adapter (architettura esagonale)	80
6.2.7.5	Service Layer	80
6.2.7.6	Repository	80
6.2.7.7	Strategy	81
6.2.8	DTO (Data Transfer Objects)	81
6.2.8.1	Introduzione	81
6.2.8.2	ResponseStatusEnum	81
6.2.8.3	ResponseDto	82
6.2.8.4	StringDataResponseDto	82



6.2.8.5	AuthResponseDto	82
6.2.8.6	DictionaryResponseDto	83
6.2.8.7	DictionariesResponseDto	83
6.2.8.8	PromptResponseDto	83
6.2.8.9	LoginDto	84
6.2.8.10	AdminDto	84
6.2.8.11	DictionaryDto	84
6.2.8.12	DictionaryPreviewDto	85
6.2.8.13	TableDto	85
6.2.8.14	PromptDto	86

Elenco delle tabelle

2.1	Framework utilizzati	22
2.2	Linguaggi utilizzati	23
2.3	Librerie utilizzate	23
2.4	Strumenti utilizzati	24
4.1	Tracciamento dei requisiti per il componente AppLayout	33
4.2	Tracciamento dei requisiti per il componente ChatView	34
4.3	Tracciamento dei requisiti per il componente DictionariesListView	36
4.4	Tracciamento dei requisiti per il componente DebugMessage	39
4.5	Tracciamento dei requisiti back-end	47

Elenco delle figure

3.1	Architettura client-server	26
3.2	Architettura web app	27
3.3	Model-View-ViewModel (MVVM)	28
3.4	Funzionamento del pattern MVVM	29
6.1	Diagramma delle classi	57
6.2	Diagramma dell'interfaccia AuthenticationUseCase	58
6.3	Diagramma dell'interfaccia DictionaryUseCase	59
6.4	Diagramma dell'interfaccia PromptUseCase	59
6.5	Diagramma dell'interfaccia SchemaValidatorUseCase	60
6.6	Diagramma dell'interfaccia EmbeddingsAbstractFactory	60
6.7	Diagramma dell'interfaccia IndexManagerPort	61
6.8	Diagramma dell'interfaccia PromptManagerPort	62
6.9	Diagramma dell'interfaccia SearchAlgorithmPort	62
6.10	Diagramma dell'interfaccia DbManagerAbstractFactory	63
6.11	Diagramma dell'interfaccia AuthenticationRepository	63
6.12	Diagramma dell'interfaccia DictionaryRepository	64
6.13	Diagramma dell'interfaccia FileRepository	64
6.14	Diagramma della classe AuthenticationService	65
6.15	Rappresentazione della classe DictionaryService	66
6.16	Rappresentazione della classe PromptManagerService	67
6.17	Diagramma della classe SchemaValidatorFactory	68
6.18	Diagramma della classe JsonSchemaValidatorAdapter	68
6.19	Diagramma della classe DbManagerFactory	69
6.20	Diagramma della classe SQLAlchemyDbManagerFactory	69
6.21	Diagramma della classe SQLAlchemyAuthenticationRepositoryAdapter	70
6.22	Diagramma della classe SQLAlchemyDictionaryRepositoryAdapter	71
6.23	Diagramma della classe EmbeddingsManagerFactory	71
6.24	Diagramma della classe TxtaiEmbeddingsManagerFactory	72
6.25	Diagramma della classe TxtaiIndexManagerAdapter	73
6.26	Diagramma della classe TxtaiPromptManagerAdapter	74
6.27	Diagramma della classe TxtaiSearchAlgorithmAdapter	75
6.28	Diagramma della classe FileFactory	76
6.29	Diagramma della classe JsonFileAdapter	76
6.30	Diagramma E-R dell'interazione tra Admins (Tecnici) e <i>dizionari dati</i> , per le operazioni <i>CRUD</i> .	77
6.31	Diagramma della classe ResponseStatusEnum	81
6.32	Diagramma della classe ResponseDto	82
6.33	Diagramma della classe StringDataResponseDto	82
6.34	Diagramma della classe AuthResponseDto	82
6.35	Diagramma della classe DictionaryResponseDto	83
6.36	Diagramma della classe DictionariesResponseDto	83
6.37	Diagramma della classe PromptResponseDto	83
6.38	Diagramma della classe LoginDto	84
6.39	Diagramma della classe AdminDto	84
6.40	Diagramma della classe DictionaryDto	84
6.41	Diagramma della classe DictionaryPreviewDto	85



6.42 Diagramma della classe TableDto	85
6.43 Diagramma della classe PromptDto	86



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 v2.0.0*;
- Slide PD2 - Corso di Ingegneria del Software - 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: 2024-09-12);
 - <https://www.math.unipd.it/~tullio/IS-1/2023/Progetto/C9p.pdf>
(Ultimo accesso: 2024-09-12).

1.2.2 Riferimenti informativi

- *Analisi dei Requisiti v2.0.0*;
- Corso di Ingegneria del Software - Dependency injection:
https://www.math.unipd.it/~rcardin/swea/2022/Design_Pattern_Architetturali-Dependency_Injection.pdf;
- Corso di Ingegneria del Software - Object-oriented programming (OOP):
https://www.math.unipd.it/~rcardin/swea/2023/Object-Oriented_Programming_Principles_Revised.pdf;
- Corso di Ingegneria del Software - Diagrammi delle classi:
https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi_delle_Classi.pdf;
- Corso di Ingegneria del Software - Design pattern architetturali:
https://www.math.unipd.it/~rcardin/swea/2022/Software_Architecture_Patterns.pdf;
- Corso di Ingegneria del Software - Design pattern creazionali:
https://www.math.unipd.it/~rcardin/swea/2022/Design_Pattern_Creazionali.pdf;



- Corso di Ingegneria del Software - Design pattern strutturali:
https://www.math.unipd.it/~rcardin/swea/2022/Design_Pattern_Strutturali.pdf;
- Corso di Ingegneria del Software - Principi SOLID:
https://www.math.unipd.it/~rcardin/swea/2021/SOLID_Principles_of_Object-Oriented_Design_4x4.pdf;
- Repository GitHub Ingegneria del Software:
<https://github.com/rcardin/swe-imdb>;
- Repository GitHub Architettura esagonale:
<https://github.com/rcardin/hexagonal>;
- Corso di Ingegneria del Software - T6 - Progettazione software:
<https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T6.pdf>;
- Corso di Ingegneria del Software - T7 - Qualità del software:
<https://www.math.unipd.it/~tullio/IS-1/2023/Dispense/T7.pdf>;
- Pattern Strategy:
<https://refactoring.guru/design-patterns/strategy>
(Ultimo accesso: 2024-09-05);
- Struttura del pattern Strategy:
https://it.wikipedia.org/wiki/Strategy_pattern
(Ultimo accesso: 2024-09-05);
- Abstract factory:
https://it.wikipedia.org/wiki/Abstract_factory
(Ultimo accesso: 2024-08-25);
- Abstract Factory in Python:
<https://shanenullain.medium.com/abstract-factory-in-python-with-generic-typing-b9ceca2bf89e>
(Ultimo accesso: 2024-09-05);
- Interfacce e classi astratte in Python:
<https://medium.com/@shashikantrbl123/interfaces-and-abstract-classes-in-python-understanding-the-differences-3e5889a0746a>
(Ultimo accesso: 2024-08-20);
- Diagrammi delle classi - Notazioni:
<https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/diagrammi-di-classe-con-uml/>
(Ultimo accesso: 2024-08-20);
- Guida alla creazione di diagrammi UML:
<https://www.drawio.com/blog/uml-class-diagrams>
(Ultimo accesso: 2024-08-10);
- Service e repository pattern:
<https://medium.com/@ankitpall181/service-repository-pattern-802540254019>
(Ultimo accesso: 2024-09-05);

- *Glossario v2.0.0;*
- 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;
 - 2024-07-26;
 - 2024-08-01;
 - 2024-08-08;
 - 2024-08-14;
 - 2024-08-19;
 - 2024-08-27;
 - 2024-09-08;
 - 2024-09-16.
- Verbali esterni:
 - 2024-04-09;
 - 2024-05-06;
 - 2024-05-22;
 - 2024-06-07;
 - 2024-07-09;
 - 2024-09-09.



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 \mathcal{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*. Per ciascuna tecnologia, il team riporta le motivazioni che ne hanno determinato la scelta.

2.1 Back-end

2.1.1 FastAPI

FastAPI è un *framework* web moderno per la creazione di API 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_e*;
- **Comunità:** Python dispone di una vasta comunità di sviluppatori che forniscono supporto e soluzioni a problemi comuni.

2.1.3 txtai

txtai_e 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_e*, 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_e*;
- **Debug:** *txtai* mette a disposizione funzioni specifiche per il *debug_e* della ricerca semantica.

2.1.4 SQLAlchemy

SQLAlchemy è lo strumento principale per l'interazione con database *SQL_e* in Python.

Motivazioni

- **Portabilità:** SQLAlchemy supporta i principali *DBMS_e* 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 manutenibilità:** SQLAlchemy utilizza classi e oggetti Python per rappresentare le relazioni di un *database_e*, rendendo il codice intuitivo e autoesplicativo;
- **Flessibilità:** SQLAlchemy consente di costruire query SQL in linguaggio nativo o di utilizzare un *ORM_e* per interagire con il database.

2.1.5 JWT

JSON Web Token (JWT) è 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* di test Python che può essere utilizzato per scrivere test di unità e test di integrazione.

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 Front-end

2.2.1 Vue.js

Vue.js 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* 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 compatibile 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*;
- **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 complessi;
- **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 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 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 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* 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 (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'adequatezza 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* 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 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.
openai	1.37.1	Libreria Python per l'interazione con i servizi di OpenAI.
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.

Continua nella prossima pagina

Tabella 2.3: Librerie utilizzate (continua)

Nome	Versione	Descrizione
Axios	3.10.12	Libreria JavaScript utilizzata per connettersi con le API di back-end e gestire le richieste effettuate tramite il protocollo HTTP.

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.
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.
Continua nella prossima pagina		



Tabella 2.4: Strumenti utilizzati (continua)

Nome	Versione	Descrizione
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.

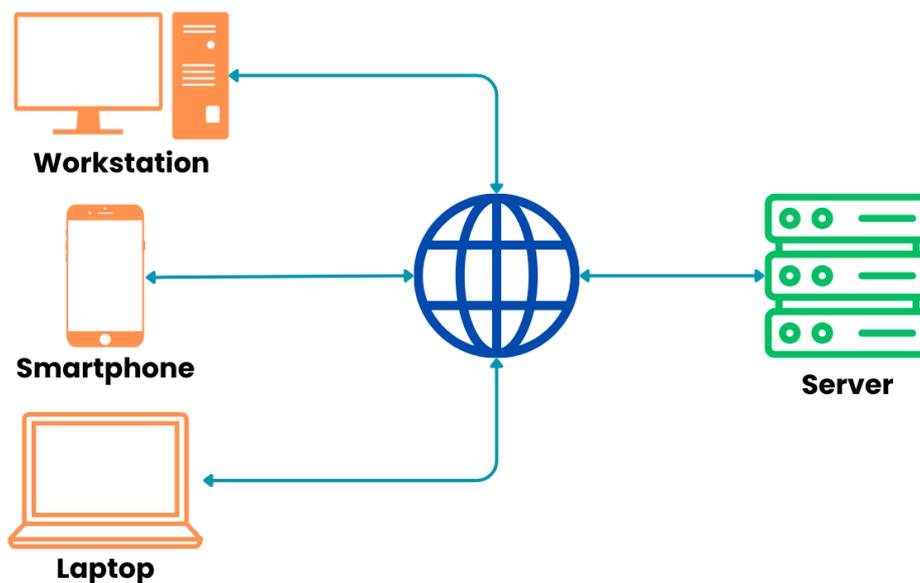


Figura 3.1: Architettura client-server

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*, 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*.

La comunicazione tra il front-end e il back-end avviene tramite chiamate *API*. 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.

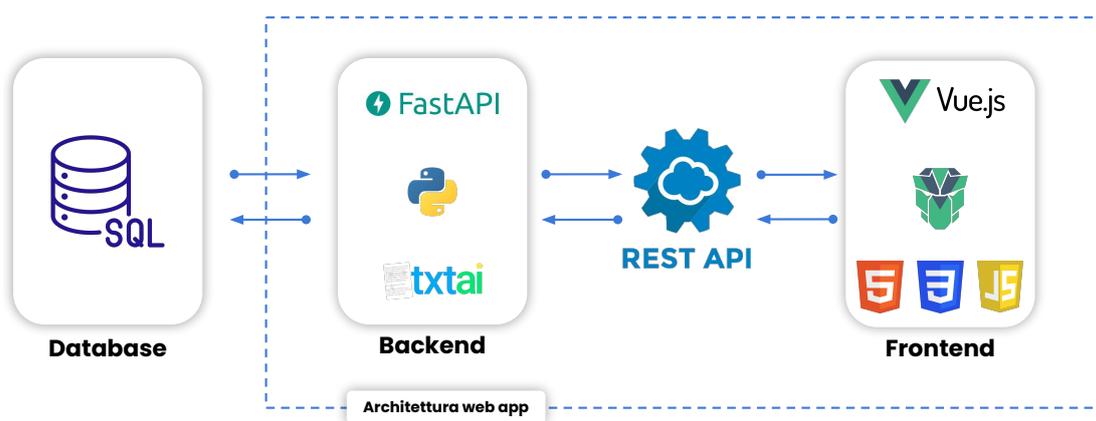


Figura 3.2: Architettura web app

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.

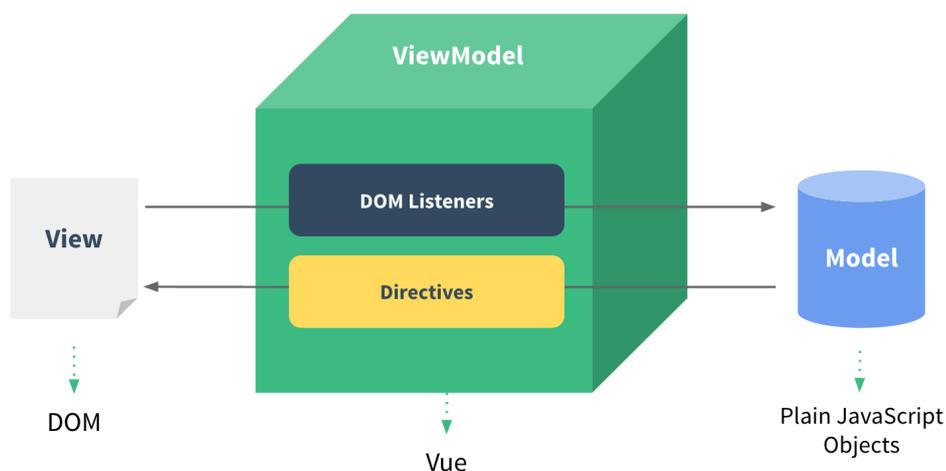


Figura 3.3: Model-View-ViewModel (MVVM)

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 cambiamenti 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.

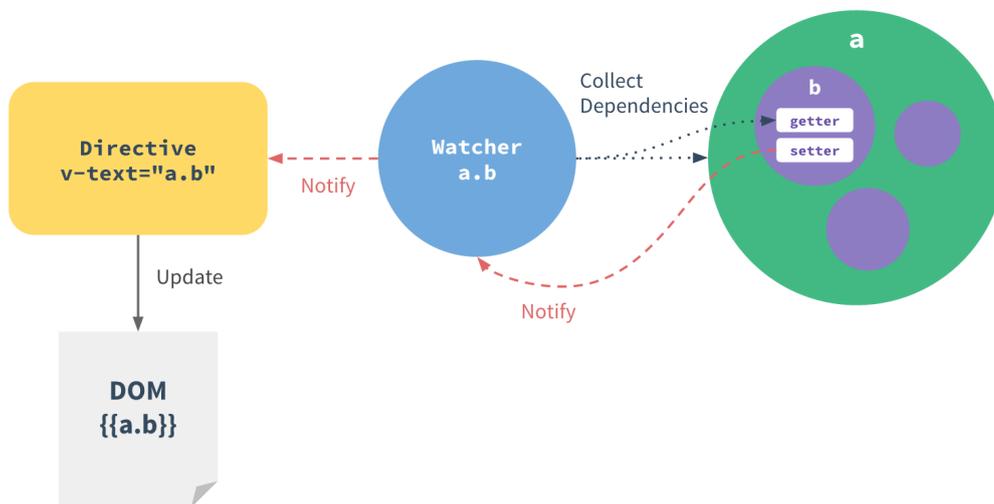


Figura 3.4: Funzionamento del pattern MVVM

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
│   ├── 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` 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;
- **services**: contiene funzioni di utilità e servizi per la comunicazione con il back-end;
- **types**: contiene le definizioni dei tipi e delle interfacce;
- **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 gruppo ha adottato l'**architettura esagonale** come modello architetturale. Questo modello trova le sue basi nell'architettura a livelli, ma mira a superarne i limiti, in particolare la stretta dipendenza tra i livelli. L'architettura è rappresentata in forma esagonale per diversi motivi. Da un lato, richiama la struttura delle celle di un alveare, dove ogni cella può collegarsi ad altre strutture simili, contribuendo alla definizione di un sistema più ampio e modulare. Dall'altro, la simmetria dell'esagono



consente di visualizzare l'architettura divisa a metà, facilitando la **suddivisione delle responsabilità**.

L'architettura è composta da 3 segmenti principali:

- **Core:** il core è la sezione centrale dell'architettura, in cui risiede la logica di business dell'applicazione. È indipendente da qualsiasi interfaccia utente o servizio esterno;
- **Port:** le porte costituiscono punti di comunicazione tra il core e le infrastrutture esterne. Esistono due tipi principali di porte:
 - Inbound port: definiscono le interfacce attraverso le quali il core riceve input dai componenti esterni. Queste porte fungono da contratti che specificano come i dati devono essere formattati;
 - Outbound port: definiscono i contratti per l'interazione del core con le infrastrutture esterne, come database e servizi. Queste porte consentono al core di richiedere dati o servizi senza preoccuparsi di come questi vengono forniti.
- **Adapter:** gli adapter agiscono come intermediari tra il core dell'applicazione e le infrastrutture esterne, trasformando i dati in un formato comprensibile per ciascun destinatario.

L'architettura esagonale consente di mantenere il core isolato dal resto del sistema, facilitando l'integrazione con servizi esterni tramite porte e adattatori. L'obiettivo è mantenere il core dell'applicazione indipendente dai dettagli implementativi.

Di seguito sono elencati alcuni dei vantaggi offerti da questo modello architetturale:

- **Separazione delle responsabilità:** la logica di business risiede nel core dell'applicazione, mentre le porte definiscono i contratti per la gestione delle interazioni con le dipendenze esterne. Questi contratti vengono poi implementati dagli adapter. Tale approccio promuove una chiara separazione delle responsabilità e migliora sia la modularità che la manutenibilità del sistema;
- **Flessibilità:** la separazione delle responsabilità e l'isolamento del nucleo rendono l'applicazione più flessibile e adattabile. In altre parole, l'aggiunta, la modifica o l'eliminazione di funzionalità non richiede una revisione dell'intera applicazione;
- **Scalabilità:** l'architettura esagonale favorisce l'aggiunta di nuove funzionalità o servizi esterni senza richiedere modifiche al core. È possibile integrare nuovi adapter per supportare tecnologie differenti, mantenendo la scalabilità del sistema. Il modello esagonale offre una base solida per scalare un'applicazione sia orizzontalmente che verticalmente;
- **Testabilità:** il core è isolato e disaccoppiato dal resto del sistema, favorendo l'utilizzo di *mock* o *stub* per gli adapter e le porte;
- **Manutenibilità:** le modifiche a un componente, come un cambiamento nella logica di business o nella modalità di interazione con le infrastrutture, non influiscono sugli altri componenti.

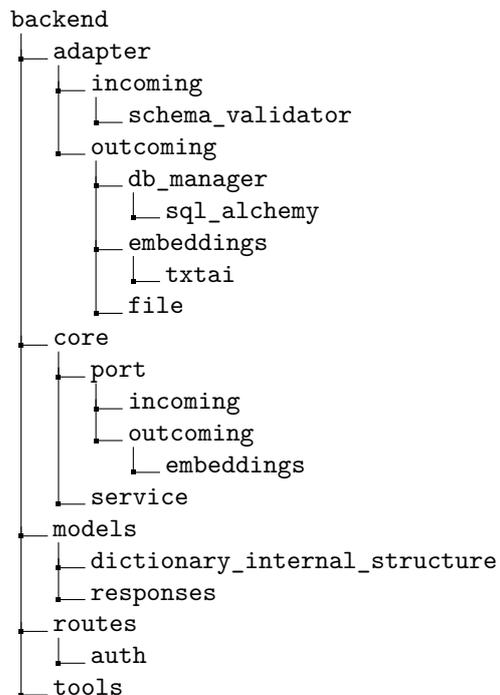


Questo tipo di architettura non è immune a difetti, tra i quali si possono trovare:

- **Complessità:** l'architettura richiede una progettazione puntuale e una buona conoscenza dei pattern architetturali;
- **Difficoltà di debugging:** il core è separato dal resto dell'architettura, di conseguenza è possibile introdurre *bug* 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 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*, o *LLM* e *API*.

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:



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 di impostazioni.

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.12	AppTopbar
RF.O.59	AppLayout
RF.O.60	ConfigSidebar
RF.O.61	ConfigSidebar
RF.D.62	ConfigSidebar

4.1.2 ChatView

4.1.2.1 Descrizione

ChatView è la pagina di generazione del *prompt_e*. Espone le seguenti funzionalità:

- Selezione di un *dizionario dati_e*;
- Visualizzazione di un'anteprima del dizionario dati;
- Selezione della lingua di richiesta;
- Selezione di un *DBMS_e*;
- Invio di una richiesta al ChatBOT;
- Visualizzazione della risposta del ChatBOT;
- Copia del prompt generato;
- Visualizzazione del messaggio di *debug_e* associato a ciascun prompt;
- Download di un file di *log_e*;
- Visualizzazione della lista di messaggi;
- Scorrimento della lista di messaggi;
- Eliminazione del contenuto della chat.

4.1.2.2 Sottocomponenti

ChatView è composto dai seguenti sottocomponenti:

- **ChatMessage**: rappresenta i messaggi inviati dall'utente o dal ChatBOT 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
Continua nella prossima pagina	

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

ID	Componente
RF.O.8	ChatMessage
RF.O.9	ChatView
RF.O.9.1	ChatView
RF.O.10	ChatView
RF.O.10.1	ChatView
RF.O.10.2	ChatView
RF.O.11	ChatView
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.48	ChatView
RF.O.49	ChatView
RF.O.50	ChatView
RF.O.51	ChatView

4.1.3 DictionariesListView

4.1.3.1 Descrizione

DictionariesListView è la pagina di gestione *CRUD*_e dei *dizionari dati*_e. Espone le seguenti funzionalità:

- Visualizzazione della lista dei dizionari;
- Ricerca dei dizionari per nome;
- Ricerca dei dizionari per descrizione;
- Creazione, modifica e cancellazione di un dizionario;
- Download di un dizionario;
- Persistenza degli indici associati ai dizionari.

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
Continua nella prossima pagina	

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

ID	Componente
RF.O.21	DictionariesListView
RF.O.24	DictionariesListView
RF.O.28	CreateUpdateDictionaryModal
RF.O.29	CreateUpdateDictionaryModal
RF.O.30	CreateUpdateDictionaryModal
RF.O.31	CreateUpdateDictionaryModal
RF.O.31.1	CreateUpdateDictionaryModal
RF.O.31.2	CreateUpdateDictionaryModal
RF.O.32	CreateUpdateDictionaryModal
RF.O.33	CreateUpdateDictionaryModal
RF.O.34	CreateUpdateDictionaryModal
RF.D.36	DictionariesListView
RF.D.37	DictionariesListView
RF.O.38	DictionariesListView
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 una singola voce del menu di navigazione.

4.1.6 ChatMessage

4.1.6.1 Descrizione

ChatMessage è il componente che rappresenta i messaggi inviati all'interno della chat.

4.1.6.2 Sottocomponenti

ChatMessage 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*.

4.1.7.3 Tracciamento dei requisiti

Tabella 4.4: Tracciamento dei requisiti per il componente DebugMessage

ID	Componente
RF.D.22	DebugMessage
RF.D.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 presenti nel sistema 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:**
 - **dictionary_service (DictionaryService):** dipendenza risolta tramite `Depends (get_dictionary_service)`.

4.2.1.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per recuperare l'elenco completo dei dizionari;
- Restituisce un oggetto `DictionariesResponseDto` con lo stato della risposta e, in caso di successo, i dati richiesti.

4.2.2 get_dictionary

4.2.2.1 Descrizione

La funzione `get_dictionary` recupera un dizionario specifico 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:**
 - **dictionary_service (DictionaryService):** dipendenza risolta tramite `Depends (get_dictionary_service)`.
- **Parametri:**
 - **id (int):** ID del dizionario da recuperare.

4.2.2.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per recuperare i dettagli del dizionario specificato;
- Restituisce un oggetto `DictionaryResponseDto` con lo stato della risposta e, in caso di successo, i dati richiesti.

4.2.3 get_dictionary_file

4.2.3.1 Descrizione

La funzione `get_dictionary_file` recupera il file associato a un dizionario specifico utilizzando l'ID fornito.

4.2.3.2 Dettagli dell'endpoint

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

4.2.3.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per recuperare il file associato al dizionario specificato;
- Se il dizionario viene trovato, restituisce il file come parte di una risposta HTTP (`FileResponse`);
- Se il dizionario non viene trovato, restituisce un oggetto `ResponseDto` con lo stato della risposta.

4.2.4 `get_dictionary_preview`

4.2.4.1 Descrizione

La funzione `get_dictionary_preview` recupera l'anteprima di un dizionario 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:**
 - **dictionary_service (DictionaryService):** dipendenza risolta tramite `Depends (get_dictionary_service)`.
- **Parametri:**
 - **id (int):** ID del dizionario di cui recuperare l'anteprima.

4.2.4.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per recuperare l'anteprima del dizionario specificato;
- Restituisce un oggetto `DictionaryResponseDto` con lo stato della risposta e, in caso di successo, i dati richiesti.

4.2.5 `create_dictionary`

4.2.5.1 Descrizione

La funzione `create_dictionary` inserisce un nuovo dizionario nel sistema utilizzando i dati forniti e un file di configurazione.

4.2.5.2 Dettagli dell'endpoint

- **HTTP Method:** POST;
- **Endpoint:** /;
- **Tags:** [tag];
- **Response Model:** DictionaryResponseDto;
- **Dependencies:**
 - `Depends(JwtBearer())`: autenticazione JWT necessaria per l'accesso all'endpoint.
- **Nome:** `createDictionary`;
- **Dependency Injection:**
 - **dictionary_service (DictionaryService)**: dipendenza risolta tramite `Depends(get_dictionary_service)`.
- **Parametri:**
 - **file (Annotated[UploadFile, File()])**: il file associato al dizionario;
 - **dictionary (DictionaryDto)**: i metadati del dizionario.

4.2.5.3 Implementazione

- Verifica la presenza del file e legge il contenuto, se disponibile;
- Effettua una chiamata al servizio di gestione dei dizionari per salvare un nuovo dizionario nel sistema;
- Restituisce un oggetto `DictionaryResponseDto` con lo stato della risposta e, in caso di successo, i dati del dizionario;

4.2.6 update_dictionary_file

4.2.6.1 Descrizione

La funzione `update_dictionary_file` aggiorna il file associato a un dizionario esistente utilizzando l'ID fornito e un nuovo file di configurazione.

4.2.6.2 Dettagli dell'endpoint

- **HTTP Method:** PUT;
- **Endpoint:** `/{id}/file`;
- **Tags:** [tag];
- **Response Model:** DictionaryResponseDto;
- **Dependencies:**
 - `Depends(JwtBearer())`: autenticazione JWT necessaria per l'accesso all'endpoint.



- **Nome:** `updateDictionaryFile`;
- **Dependency Injection:**
 - **dictionary_service (DictionaryService):** dipendenza risolta tramite `Depends (get_dictionary_service)`.
- **Parametri:**
 - **id (int):** ID del dizionario da aggiornare;
 - **file (Annotated[UploadFile, File()]):** il file associato al dizionario.

4.2.6.3 Implementazione

- Verifica la presenza del file e legge il contenuto, se disponibile;
- Effettua una chiamata al servizio di gestione dei dizionari per modificare il file associato al dizionario specificato;
- Restituisce un oggetto `DictionaryResponseDto` con lo stato della risposta e, in caso di successo, i dati del dizionario.

4.2.7 update_dictionary_metadata

4.2.7.1 Descrizione

La funzione `update_dictionary_metadata` aggiorna i metadati di un dizionario esistente utilizzando l'ID fornito e un oggetto `DictionaryDto`.

4.2.7.2 Dettagli dell'endpoint

- **HTTP Method:** `PUT`;
- **Endpoint:** `/id`;
- **Tags:** `[tag]`;
- **Response Model:** `DictionaryResponseDto`;
- **Dependencies:**
 - `Depends(JwtBearer())`: autenticazione JWT necessaria per l'accesso all'endpoint.
- **Nome:** `updateDictionaryMetadata`;
- **Dependency Injection:**
 - **dictionary_service (DictionaryService):** dipendenza risolta tramite `Depends (get_dictionary_service)`.
- **Parametri:**
 - **id (int):** ID del dizionario da aggiornare;
 - **dictionary (DictionaryDto):** i nuovi metadati del dizionario.

4.2.7.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per modificare i metadati del dizionario specificato;
- Restituisce un oggetto `DictionaryResponseDto` con lo stato della risposta e, in caso di successo, i dati del dizionario.

4.2.8 delete_dictionary

4.2.8.1 Descrizione

La funzione `delete_dictionary` elimina un dizionario dal sistema utilizzando l'ID fornito.

4.2.8.2 Dettagli dell'endpoint

- **HTTP Method:** DELETE;
- **Endpoint:** `/id`;
- **Tags:** `[tag]`;
- **Response Model:** `ResponseDto`;
- **Dependencies:**
 - `Depends(JwtBearer())`: autenticazione JWT necessaria per l'accesso all'endpoint.
- **Nome:** `deleteDictionary`;
- **Dependency Injection:**
 - **dictionary_service (DictionaryService)**: dipendenza risolta tramite `Depends(get_dictionary_service)`.
- **Parametri:**
 - **id (int)**: ID del dizionario da eliminare.

4.2.8.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dizionari per eliminare il dizionario specificato;
- Restituisce un oggetto `ResponseDto` con lo stato della risposta.

4.2.9 login

4.2.9.1 Descrizione

La funzione `login` autentica un utente utilizzando le credenziali fornite (username e password). Se le credenziali sono corrette, restituisce una risposta strutturata contenente un token di autenticazione JWT.

4.2.9.2 Dettagli dell'endpoint

- **HTTP Method:** POST;
- **Endpoint:** /;
- **Tags:** [tag];
- **Response Model:** AuthResponseDto;
- **Nome:** login;
- **Dependency Injection:**
 - **authentication_service (AuthenticationService):** dipendenza risolta tramite Depends (get_authentication_service).
- **Parametri:**
 - **data (LoginDto):** i dati di autenticazione dell'utente.

4.2.9.3 Implementazione

- Effettua una chiamata al servizio di autenticazione per autorizzare l'utente attraverso le credenziali fornite;
- Restituisce un oggetto AuthResponseDto con lo stato della risposta e, in caso di successo, il token di autenticazione JWT.

4.2.10 generate_prompt

4.2.10.1 Descrizione

La funzione `generate_prompt` genera un *prompt*_g utilizzando i parametri forniti dall'utente, tra cui un dizionario, una richiesta, una lingua e un *DBMS*_g.

4.2.10.2 Dettagli dell'endpoint

- **HTTP Method:** GET;
- **Endpoint:** /;
- **Tags:** [tag];
- **Response Model:** StringDataResponseDto;
- **Nome:** generatePrompt;
- **Dependency Injection:**
 - **prompt_manager_service (PromptManagerService):** dipendenza risolta tramite Depends (get_prompt_manager_service).
- **Parametri:**
 - **dictionary_id (Annotated[int, Query(alias="dictionaryId")]):** l'ID del dizionario;
 - **query (str):** la richiesta dell'utente in linguaggio naturale;

- **dbms (str)**: il sistema di gestione del database (DBMS);
- **lang (str)**: la lingua di richiesta.

4.2.10.3 Implementazione

- Effettua una chiamata al servizio di gestione dei prompt per generare un prompt in base ai parametri forniti;
- Restituisce un oggetto `StringDataResponseDto` con lo stato della risposta e, in caso di successo, il prompt generato.

4.2.11 generate_prompt_with_debug

4.2.11.1 Descrizione

La funzione `generate_prompt_with_debug` genera un *prompt_e* e include informazioni dettagliate di *debug_e*, fornendo dati aggiuntivi per la risoluzione di problemi.

4.2.11.2 Dettagli dell'endpoint

- **HTTP Method**: GET;
- **Endpoint**: /debug;
- **Tags**: [tag];
- **Response Model**: `PromptResponseDto`;
- **Dependencies**:
 - `Depends(JwtBearer())`: autenticazione JWT necessaria per l'accesso all'endpoint.
- **Nome**: `generatePromptWithDebug`;
- **Dependency Injection**:
 - **prompt_manager_service (PromptManagerService)**: dipendenza risolta tramite `Depends (get_prompt_manager_service)`;
- **Parametri**:
 - **dictionary_id (Annotated[int, Query(alias="dictionaryId")])**: l'ID del dizionario;
 - **query (str)**: la richiesta dell'utente in linguaggio naturale;
 - **dbms (str)**: il sistema di gestione del database (DBMS);
 - **lang (str)**: la lingua di richiesta.

4.2.11.3 Implementazione

- Effettua una chiamata al servizio di gestione dei prompt per generare un prompt con informazioni di debug relative ad esso;

- Restituisce un oggetto `PromptResponseDto` con lo stato della risposta e, in caso di successo, i dati richiesti.

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.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
Continua nella prossima pagina	

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

ID	Route
RF.O.21	delete_dictionary
RF.D.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
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.38	get_dictionary_file
RF.O.39	create_dictionary, update_dictionary_file
RF.D.40	create_dictionary, update_dictionary_file
RF.OP.42	generate_prompt, generate_prompt_with_debug
RF.O.45	generate_prompt, generate_prompt_with_debug
RF.D.46	generate_prompt_with_debug
RF.O.47	generate_prompt, generate_prompt_with_debug
RF.OP.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 front-end

5.1 Componenti

5.1.1 AppLayout

Elementi chiave

- Layout Wrapper: aggiorna lo stile dell'applicazione in base alla configurazione del sistema;
- Outside Click Listener: i menu vengono chiusi quando viene effettuato un clic al di fuori di essi;
- 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*_e selezionato dall'utente;
- Toggle Select View: pulsante per visualizzare o nascondere il form di selezione del dizionario dati, della lingua e del *DBMS*_e;
- Select Dictionary Dropdown: menu a discesa per selezionare un dizionario dati;
- View Details: pulsante per visualizzare 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;
- Scroll To Bottom: pulsante per scorrere fino all'ultimo messaggio 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*_e è ancora in corso.

5.1.3 DictionariesListView

Elementi chiave

- Create Dictionary Button: pulsante per aprire la finestra di inserimento di un *dizionario dati*_e;
- Search By Name Input: campo di testo per filtrare i dizionari dati in base al nome;
- Search By Description Input: campo di testo per filtrare i dizionari dati in base alla descrizione;

- 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.
- Status Message: messaggio per notificare all'utente che:
 - L'eliminazione del dizionario dati è avvenuta con successo;
 - L'eliminazione del dizionario dati è fallita.

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*.

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*.

5.1.8 DictPreview

Props

- DetailsVisible: flag per visualizzare o nascondere l'anteprima del *dizionario dati*;
- 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.

Emits

- HideDetails: evento per notificare al componente genitore che l'anteprima del dizionario dati deve essere chiusa.

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;
- Close Button: pulsante per chiudere l'anteprima del dizionario dati.

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 al 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* è ancora in corso;
- Status Message: messaggio per notificare all'utente che:
 - L'inserimento del dizionario è avvenuto con successo;
 - L'inserimento del dizionario è fallito;
 - La modifica dei metadati del dizionario è avvenuta con successo;
 - La modifica dei metadati del dizionario è fallita;
 - La modifica del file associato al dizionario è avvenuta con successo;
 - La modifica del file associato al dizionario è fallita.

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 essi.

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 (opzionale): variabile per impostare la larghezza del logo;
- Height (opzionale): 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 Interfacce e tipi

5.2.1 OpenAPI

Il team ha creato un file *openapi.d.ts* che definisce i tipi *TypeScript*_e per un client generato da *OpenAPI*_e. La Specifica OpenAPI è una specifica per file di interfaccia leggibile dalle macchine per descrivere, produrre, consumare e visualizzare servizi web RESTful. Il file viene generato automaticamente e facilita la comunicazione type-safe con un'API. All'interno del file sono definite una serie di interfacce e tipi che descrivono le strutture dati e le operazioni supportate dall'API.

Attraverso la descrizione degli endpoint, il file *openapi.d.ts* consente agli sviluppatori *front-end*_e di lavorare con tipi rigorosamente definiti quando effettuano richieste API con *Axios*_e. Questo garantisce che i dati delle richieste e delle risposte aderiscano al formato previsto, riducendo il tasso di errori e migliorando la leggibilità e la manutenibilità del codice. Il file prevede due namespace principali:

- *Components.Schemas*: contiene le definizioni delle strutture dati (*DTO*_e) che rappresentano le richieste e le risposte dell'API;
- *Paths*: contiene le definizioni degli endpoint dell'API, inclusi i parametri di richiesta e le risposte attese.

5.2.2 Wrapper

Il file *wrapper.ts* definisce interfacce e tipi personalizzati per gestire in modo tipizzato diversi aspetti dell'applicazione web, tra cui:

- Configurazione dei messaggi di stato in base al tipo di operazione eseguita;
- Gestione dei messaggi inviati e ricevuti all'interno della chat;
- Modellazione della struttura del menu di navigazione;



- Assegnazione dinamica delle classi CSS_e ;
- Visualizzazione dei $DBMS_e$ e delle lingue disponibili.

Nel file *wrapper.ts*, il team ha aggiunto dei contenitori per le interfacce e i tipi definiti in *openapi.d.ts*. Questa organizzazione semplifica l'importazione dei tipi nei file dell'applicazione.

5.3 Gestione degli errori

Il gruppo ha introdotto una gestione centralizzata degli errori nel *front-end_e*. Il file *status-messages.ts* si occupa della traduzione e della generazione di messaggi di stato personalizzati. Il team ha definito un set di messaggi per operazioni comuni come:

- Autenticazione;
- Generazione di *prompt_e*;
- Creazione, aggiornamento, cancellazione e recupero dei *dizionari dati_e*.

Il file *status-messages.ts* utilizza le interfacce e i tipi definiti in *wrapper.ts*, in particolare *StatusMessages*, che svolge due compiti principali:

- Rappresenta un dizionario di messaggi organizzati in base ai codici di stato (analoghi a quelli delle risposte HTTP) definiti nell'enumerazione presente in *openapi.d.ts*;
- Contiene funzioni che generano messaggi specifici basati sulle opzioni fornite.

6 Progettazione di dettaglio back-end

6.1 Diagramma delle classi

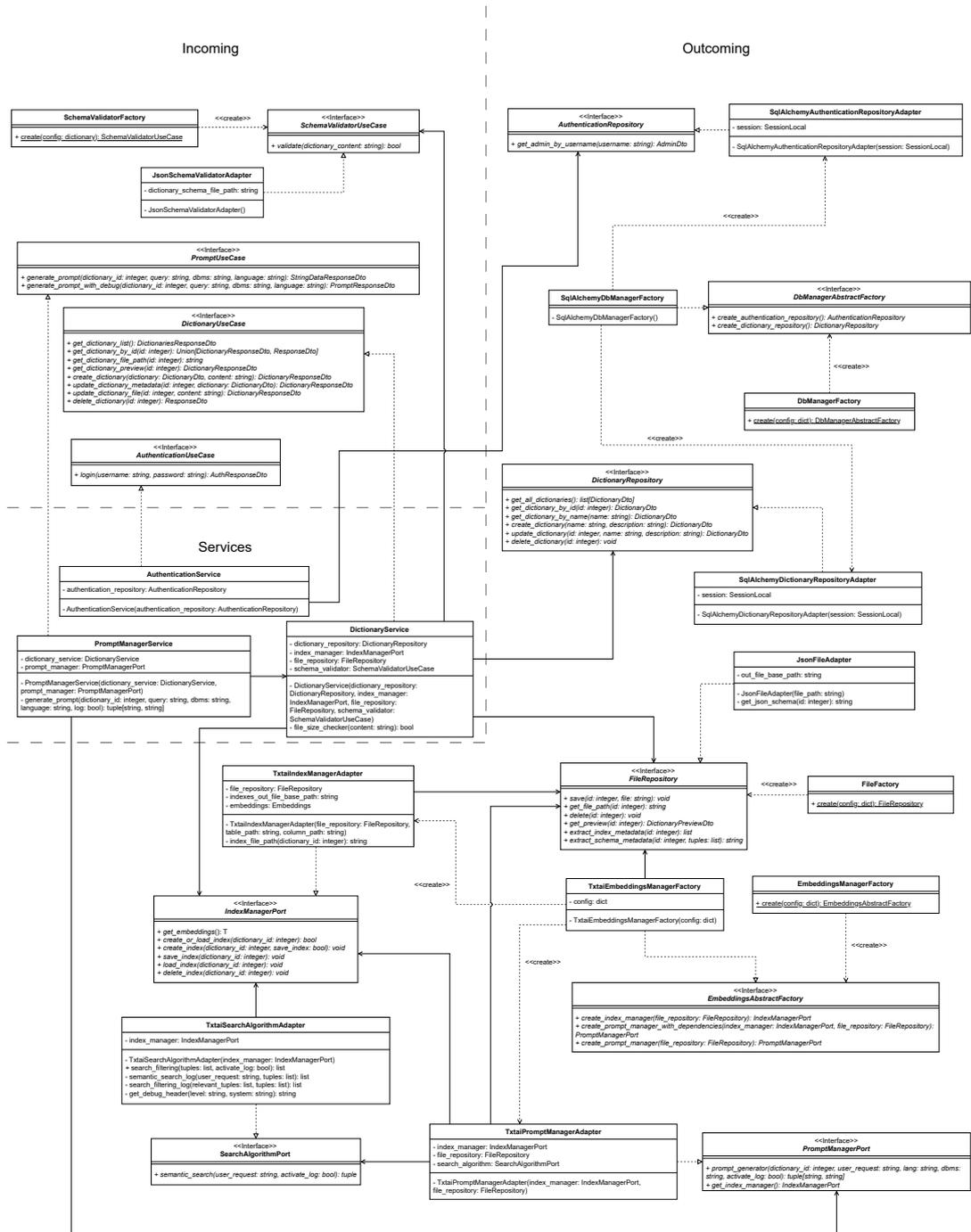


Figura 6.1: Diagramma delle classi

6.2 Descrizione delle classi

6.2.1 Introduzione

Il team ha creato un file di configurazione che utilizza il pattern *Singleton*, per gestire le impostazioni dell'applicazione. La classe *Configuration* è responsabile del caricamento e della gestione delle configurazioni, che vengono lette da un file *JSON* esterno. All'interno di questo file, è possibile definire le dipendenze esterne, tra cui *txtai* e *SQLAlchemy*, che saranno gestite dagli adapter. Inoltre, è previsto il supporto per parametri aggiuntivi, come i percorsi dei modelli di ricerca semantica per *txtai*. La classe *Configuration* offre un punto centrale di accesso alle configurazioni e ai servizi dell'applicazione, semplificando la gestione e l'organizzazione del codice.

La sezione seguente descrive le classi che compongono il *back-end* dell'applicazione. L'elenco segue la composizione dell'architettura, fornendo una descrizione degli attributi e dei metodi che caratterizzano ciascuna classe, insieme alle interfacce implementate e alle dipendenze verso altre classi del sistema.

La sezione principale è suddivisa in due sottosezioni:

- **Incoming:** classi che preparano i dati per il core dell'applicazione;
- **Outcoming:** classi che interagiscono con servizi esterni.

Le sezioni *incoming* e *outcoming* sono ulteriormente suddivise per distinguere tra porte, adapter e servizi. Le porte contengono le interfacce che vengono poi implementate dagli adapter o dai servizi.

6.2.2 Ports - Incoming

6.2.2.1 AuthenticationUseCase

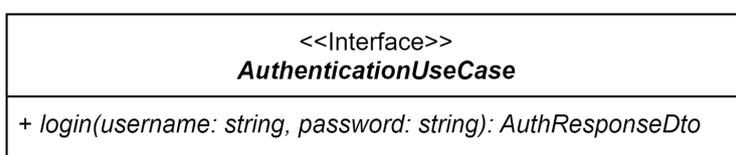


Figura 6.2: Diagramma dell'interfaccia AuthenticationUseCase

- **Descrizione:** AuthenticationUseCase definisce il contratto per i casi d'uso relativi all'autenticazione degli utenti;
- **Operazioni:**
 - Autenticazione con username e password.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.



6.2.2.2 DictionaryUseCase

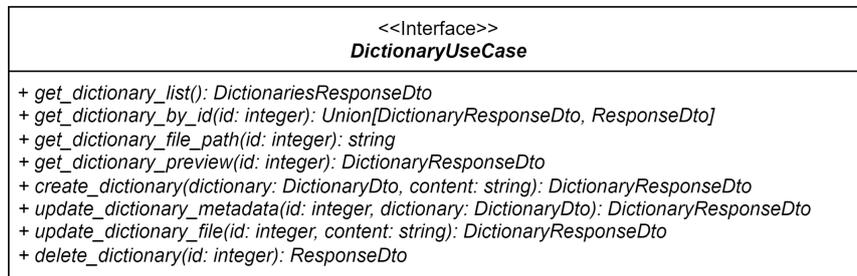


Figura 6.3: Diagramma dell'interfaccia DictionaryUseCase

- **Descrizione:** DictionaryUseCase definisce il contratto per i casi d'uso relativi alla gestione dei *dizionari dati*;
- **Operazioni:**
 - Recupero delle informazioni di tutti i dizionari;
 - Recupero delle informazioni di un dizionario;
 - Recupero del percorso del file di un dizionario;
 - Anteprima di un dizionario;
 - Creazione di un dizionario;
 - Aggiornamento dei metadati di un dizionario;
 - Aggiornamento del file di un dizionario;
 - Eliminazione di un dizionario.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.2.3 PromptUseCase

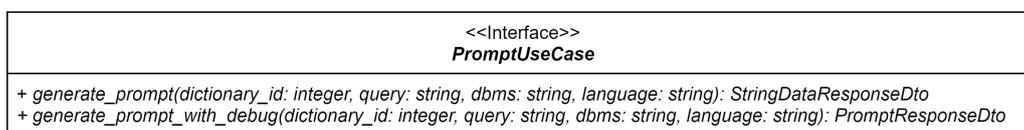


Figura 6.4: Diagramma dell'interfaccia PromptUseCase

- **Descrizione:** PromptUseCase definisce il contratto per i casi d'uso relativi alla generazione del *prompt*;
- **Operazioni:**



- Generazione del prompt a partire da una richiesta, un dizionario, una lingua e un $DBMS_e$;
- Generazione del prompt con informazioni di debug associate ad esso.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.2.4 SchemaValidatorUseCase

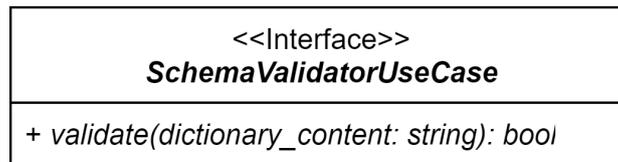


Figura 6.5: Diagramma dell'interfaccia SchemaValidatorUseCase

- **Descrizione:** SchemaValidatorUseCase definisce il contratto per i casi d'uso relativi alla validazione dello schema dei *dizionari dati_e*;
- **Operazioni:**
 - Validazione dello schema di un dizionario dati.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.2.5 Ports – Outcoming

6.2.2.6 EmbeddingsAbstractFactory

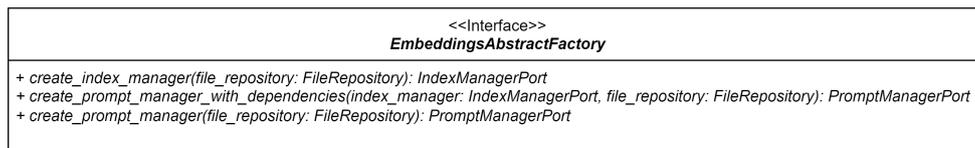


Figura 6.6: Diagramma dell'interfaccia EmbeddingsAbstractFactory

- **Descrizione:** EmbeddingsAbstractFactory fornisce una struttura per la creazione dei componenti relativi agli *embeddings_e* e alla generazione di *prompt_e*;
- **Operazioni:**
 - Creazione di un'istanza del gestore degli *indici_e*;
 - Creazione di un'istanza del gestore dei prompt.

- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete;
- **Dipendenze:**
 - FileRepository;
 - IndexManagerPort;
 - PromptManagerPort.

6.2.2.7 IndexManagerPort

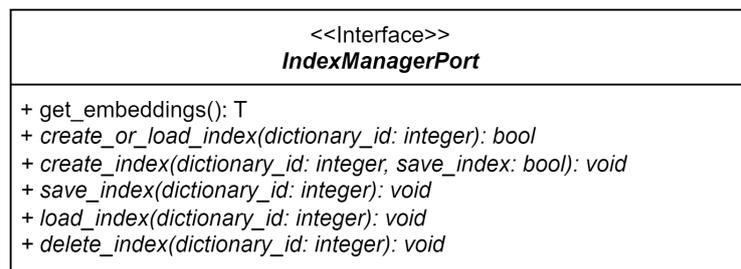


Figura 6.7: Diagramma dell'interfaccia IndexManagerPort

- **Descrizione:** IndexManagerPort fornisce un'interfaccia per gestire le operazioni $CRUD_e$ relative agli $embeddings_e$;
- **Operazioni:**
 - Recupero degli embeddings;
 - Creazione di un $indice_e$;
 - Salvataggio di un indice;
 - Ripristino di un indice;
 - Eliminazione di un indice.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.2.8 PromptManagerPort

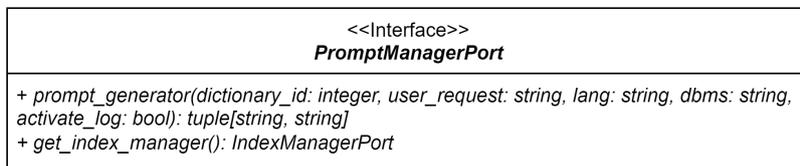


Figura 6.8: Diagramma dell'interfaccia PromptManagerPort

- **Descrizione:** PromptManagerPort definisce il contratto per la generazione di *prompt_e*;
- **Operazioni:**
 - Generazione del prompt;
 - Recupero dell'istanza del gestore degli *indici_e*.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete;
- **Dipendenze:**
 - IndexManagerPort.

6.2.2.9 SearchAlgorithmPort

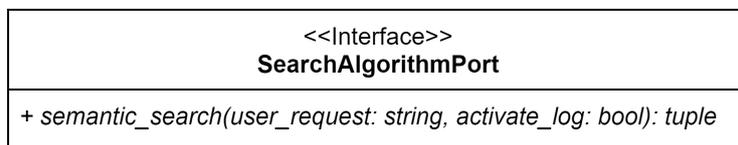


Figura 6.9: Diagramma dell'interfaccia SearchAlgorithmPort

- **Descrizione:** SearchAlgorithmPort definisce il contratto per le operazioni di ricerca semantica;
- **Operazioni:**
 - Ricerca semantica;
 - *Debug_e* del processo di ricerca semantica.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.



6.2.2.10 DbManagerAbstractFactory

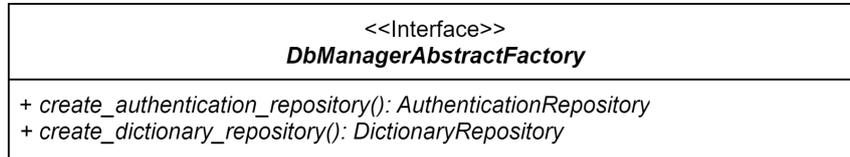


Figura 6.10: Diagramma dell'interfaccia DbManagerAbstractFactory

- **Descrizione:** DbManagerAbstractFactory fornisce una struttura per la creazione dei componenti relativi alla gestione dei dizionari e degli utenti;
- **Operazioni:**
 - Creazione di un'istanza della classe per la gestione degli utenti;
 - Creazione di un'istanza della classe per la gestione dei dizionari.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete;
- **Dipendenze:**
 - AuthenticationRepository;
 - DictionaryRepository.

6.2.2.11 AuthenticationRepository

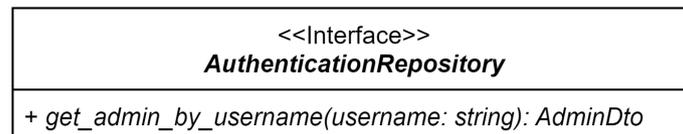


Figura 6.11: Diagramma dell'interfaccia AuthenticationRepository

- **Descrizione:** AuthenticationRepository definisce un contratto per l'accesso e la gestione dei dati di autenticazione degli utenti;
- **Operazioni:**
 - Recupera i dettagli di un admin tramite lo username.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.



6.2.2.12 DictionaryRepository

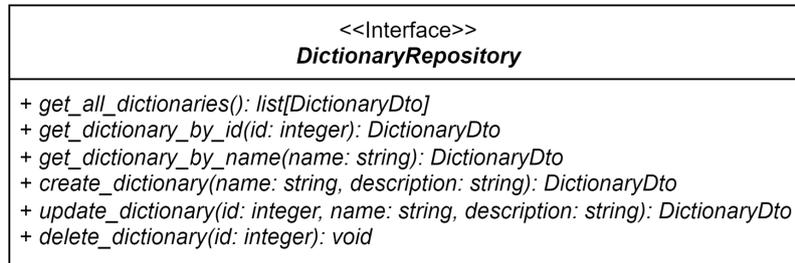


Figura 6.12: Diagramma dell'interfaccia DictionaryRepository

- **Descrizione:** DictionaryRepository definisce un contratto per la gestione dei dizionari all'interno del sistema;
- **Operazioni:**
 - Recupero delle informazioni di tutti i dizionari;
 - Recupero delle informazioni di un dizionario tramite il suo ID;
 - Recupero delle informazioni di un dizionario tramite il suo nome;
 - Creazione di un dizionario;
 - Aggiornamento dei metadati di un dizionario;
 - Eliminazione di un dizionario.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.2.13 FileRepository

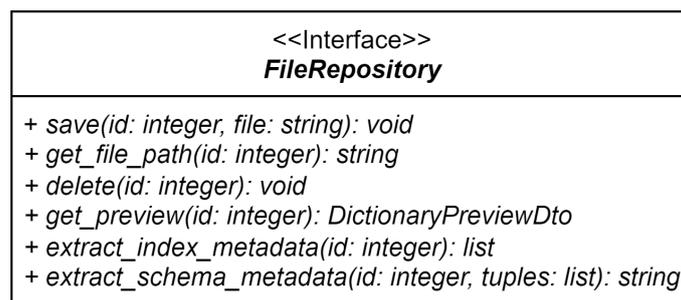


Figura 6.13: Diagramma dell'interfaccia FileRepository



- **Descrizione:** FileRepository definisce un contratto per la gestione delle operazioni *CRUD*₆ sui file associati ai dizionari;
- **Operazioni:**
 - Salvataggio di un file;
 - Recupero del percorso di un file;
 - Eliminazione di un file;
 - Estrazione di un'anteprima del contenuto di un file;
 - Estrazione dei metadati di un file per l'*indicizzazione*₆;
 - Estrazione dei metadati di un file in forma di *prompt*₆.
- **Implementazione:** i dettagli dell'implementazione sono riportati nelle classi concrete.

6.2.3 Servizi

6.2.3.1 AuthenticationService

AuthenticationService
- authentication_repository: AuthenticationRepository
- AuthenticationService(authentication_repository: AuthenticationRepository)

Figura 6.14: Diagramma della classe AuthenticationService

- **Descrizione:** AuthenticationService implementa il caso d'uso di autenticazione;
- **Interfaccia implementata:** AuthenticationUseCase;
- **Attributi:**
 - authentication_repository: istanza della classe utilizzata per gestire gli utenti nel sistema.
- **Metodi:**
 - AuthenticationService(authentication_repository: AuthenticationRepository): costruttore della classe;
 - + login(username: string, password: string): autentica un utente basandosi sullo username e sulla password forniti. Restituisce un oggetto di risposta che include un messaggio di stato e, in caso di successo, il token di autenticazione.
- **Dipendenze:**
 - AuthenticationRepository.



6.2.3.2 DictionaryService

DictionaryService
- dictionary_repository: DictionaryRepository - index_manager: IndexManagerPort - file_repository: FileRepository - schema_validator: SchemaValidatorUseCase
- DictionaryService(dictionary_repository: DictionaryRepository, index_manager: IndexManagerPort, file_repository: FileRepository, schema_validator: SchemaValidatorUseCase) - file_size_checker(content: string): bool

Figura 6.15: Rappresentazione della classe DictionaryService

- **Descrizione:** DictionaryService è responsabile della gestione dei *dizionari dati*_e;
- **Interfaccia implementata:** DictionaryUseCase;
- **Attributi:**
 - dictionary_repository: istanza della classe utilizzata per gestire i dati dei dizionari;
 - index_manager: istanza del gestore degli *indici*_e;
 - file_repository: istanza del gestore dei file;
 - schema_validator: istanza della classe utilizzata per validare lo schema dei dizionari.
- **Metodi:**
 - DictionaryService(dictionary_repository: DictionaryRepository, index_manager: IndexManagerPort, file_repository: FileRepository, schema_validator: SchemaValidatorUseCase): costruttore della classe;
 - + get_dictionary_list(): recupera la lista di tutti i dizionari;
 - + get_dictionary_by_id(id: integer): recupera un dizionario tramite il suo id;
 - + get_dictionary_file_path(id: integer): recupera il percorso del file tramite il suo id;
 - + get_dictionary_preview(id: integer): recupera l'anteprima del dizionario tramite il suo id;
 - + create_dictionary(dictionary: DictionaryDto, content: string): crea un nuovo dizionario e salva il file associato;
 - + update_dictionary_metadata(id: integer, dictionary: DictionaryDto): aggiorna i metadati di un dizionario esistente;
 - + update_dictionary_file(id: integer, content: string): aggiorna il file di un dizionario esistente;



- + `delete_dictionary(id: integer)`: elimina un dizionario esistente, inclusi il file e l'indice associati;
- - `file_size_checker(content: string)`: controlla se la dimensione del file rientra nei limiti consentiti.

- **Dipendenze:**

- DictionaryRepository;
- IndexManagerPort;
- FileRepository;
- SchemaValidatorUseCase.

6.2.3.3 PromptManagerService

PromptManagerService
- dictionary_service: DictionaryService - prompt_manager: PromptManagerPort
- PromptManagerService(dictionary_service: DictionaryService, prompt_manager: PromptManagerPort) - generate_prompt(dictionary_id: integer, query: string, dbms: string, language: string, log: bool): tuple[string, string]

Figura 6.16: Rappresentazione della classe PromptManagerService

- **Descrizione:** PromptManagerService si occupa di gestire le operazioni relative alla generazione di *prompt_e*;
- **Interfaccia implementata:** PromptUseCase;
- **Attributi:**
 - `dictionary_service`: istanza del servizio di gestione dei dizionari;
 - `prompt_manager`: istanza del gestore dei prompt.
- **Metodi:**
 - `PromptManagerService(dictionary_service: DictionaryService, prompt_manager: PromptManagerPort)`: costruttore della classe;
 - + `generate_prompt(dictionary_id: integer, query: string, dbms: string, language: string)`: genera un prompt a partire da una richiesta, un dizionario, una lingua e un *DBMS_e*;
 - + `generate_prompt_with_debug(dictionary_id: integer, query: string, dbms: string, language: string)`: genera un prompt con informazioni di *debug_e* relative ad esso;
 - - `generate_prompt(dictionary_id: integer, query: string, dbms: string, language: string, log: bool)`: metodo interno per attivare la generazione del prompt utilizzando il prompt manager.
- **Dipendenze:**



- DictionaryService;
- PromptManagerPort.

6.2.4 Adapters - Incoming

6.2.4.1 SchemaValidatorFactory



Figura 6.17: Diagramma della classe SchemaValidatorFactory

- **Descrizione:** SchemaValidatorFactory si occupa della creazione di istanze di validatori di schema, come JsonSchemaValidatorAdapter, basandosi sulla configurazione fornita;
- **Metodi:**
 - + create(config: dict): restituisce un'istanza del validatore in base al valore associato alla chiave "validation_file_type".
- **Dipendenze:**
 - SchemaValidatorUseCase.

6.2.4.2 JsonSchemaValidatorAdapter

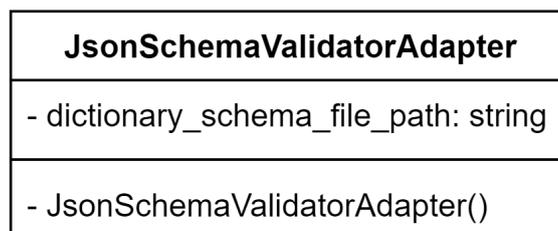


Figura 6.18: Diagramma della classe JsonSchemaValidatorAdapter

- **Descrizione:** JsonSchemaValidatorAdapter implementa il caso d'uso di validazione dello schema dei dizionari in formato *JSON_G*;
- **Interfaccia implementata:** SchemaValidatorUseCase;
- **Attributi:**



- `dictionary_schema_file_path`: percorso del file *JSON* che contiene lo schema dei dizionari dati.

• **Metodi:**

- `JsonSchemaValidatorAdapter()`: costruttore della classe;
- `+ validate(dictionary_content: string)`: valida il dizionario dati rispetto a uno schema predefinito.

• **Dipendenze:**

- `Utils`.

6.2.5 Outcoming - Adapters

6.2.5.1 DbManagerFactory

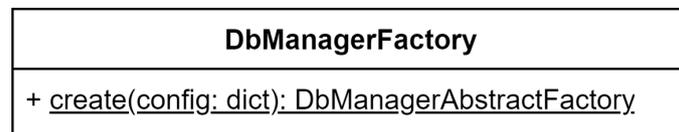


Figura 6.19: Diagramma della classe `DbManagerFactory`

- **Descrizione:** `DbManagerFactory` si occupa della creazione di istanze di gestori di *database* specifici, come `SqlAlchemyDbManagerFactory`, basandosi sulla configurazione fornita;

• **Metodi:**

- `+ create(config: dict)`: restituisce un'istanza del gestore di database in base al valore associato alla chiave "db_manager".

• **Dipendenze:**

- `DbManagerAbstractFactory`.

6.2.5.2 SqlAlchemyDbManagerFactory



Figura 6.20: Diagramma della classe `SqlAlchemyDbManagerFactory`

- **Descrizione:** SQLAlchemyDbManagerFactory si occupa della creazione di istanze di classi per la gestione degli utenti e dei dizionari utilizzando *SQLAlchemy*;
- **Interfaccia implementata:** DbManagerAbstractFactory;
- **Metodi:**
 - + create_authentication_repository(): restituisce un'istanza della classe SQLAlchemyAuthenticationRepositoryAdapter;
 - + create_dictionary_repository(): restituisce un'istanza della classe SQLAlchemyDictionaryRepositoryAdapter.
- **Dipendenze:**
 - AuthenticationRepository;
 - DictionaryRepository.

6.2.5.3 SQLAlchemyAuthenticationRepositoryAdapter

SQLAlchemyAuthenticationRepositoryAdapter
- session: SessionLocal
- SQLAlchemyAuthenticationRepositoryAdapter(session: SessionLocal)

Figura 6.21: Diagramma della classe SQLAlchemyAuthenticationRepositoryAdapter

- **Descrizione:** SQLAlchemyAuthenticationRepositoryAdapter interagisce con un database utilizzando *SQLAlchemy* per la gestione degli utenti;
- **Interfaccia implementata:** AuthenticationRepository;
- **Attributi:**
 - session: sessione di connessione al database.
- **Metodi:**
 - SQLAlchemyAuthenticationRepositoryAdapter(session: SessionLocal): costruttore della classe;
 - + get_admin_by_username(username: string): recupera un admin dal database tramite il suo username.

6.2.5.4 SQLAlchemyDictionaryRepositoryAdapter

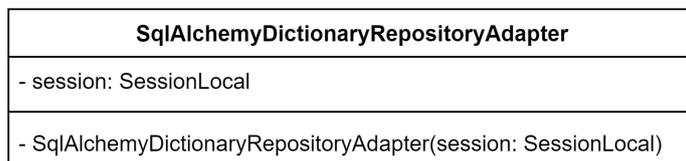


Figura 6.22: Diagramma della classe SQLAlchemyDictionaryRepositoryAdapter

- **Descrizione:** SQLAlchemyDictionaryRepositoryAdapter interagisce con un database utilizzando *SQLAlchemy* per la gestione dei dizionari;
- **Interfaccia implementata:** DictionaryRepository;
- **Attributi:**
 - session: sessione di connessione al database.
- **Metodi:**
 - SQLAlchemyDictionaryRepositoryAdapter(session: SessionLocal): costruttore della classe;
 - + get_all_dictionaries(): recupera tutti i dizionari dal database;
 - + get_dictionary_by_id(id: integer): recupera un dizionario tramite il suo ID;
 - + get_dictionary_by_name(name: string): recupera un dizionario tramite il suo nome;
 - + create_dictionary(name: string, description: string): crea un nuovo dizionario nel database;
 - + update_dictionary(id: integer, name: string, description: string): aggiorna i dati di un dizionario nel database;
 - + delete_dictionary(id: integer): elimina un dizionario dal database.

6.2.5.5 EmbeddingsManagerFactory



Figura 6.23: Diagramma della classe EmbeddingsManagerFactory



- **Descrizione:** EmbeddingsManagerFactory si occupa della creazione di istanze di gestori di *embeddings_e*, come TxtaiEmbeddingsManagerFactory, basandosi sulla configurazione fornita;
- **Metodi:**
 - + `create(config: dict)`: restituisce un'istanza del gestore di embeddings in base al valore associato alla chiave "embeddings_type".
- **Dipendenze:**
 - EmbeddingsAbstractFactory.

6.2.5.6 TxtaiEmbeddingsManagerFactory

TxtaiEmbeddingsManagerFactory
- config: dict
- TxtaiEmbeddingsManagerFactory(config: dict)

Figura 6.24: Diagramma della classe TxtaiEmbeddingsManagerFactory

- **Descrizione:** TxtaiEmbeddingsManagerFactory si occupa della creazione di istanze di gestori degli *indici_e* e dei *prompt_e* utilizzando *txtai_e*;
- **Interfaccia implementata:** EmbeddingsAbstractFactory;
- **Attributi:**
 - `config`: dizionario di configurazione del sistema.
- **Metodi:**
 - `TxtaiEmbeddingsManagerFactory(config: dict)`: costruttore della classe;
 - `create_index_manager(file_repository: FileRepository)`: restituisce un'istanza della classe `TxtaiIndexManagerAdapter`, impostando parametri specifici per *txtai*;
 - `create_prompt_manager_with_dependencies(index_manager: IndexManagerPort, file_repository: FileRepository)`: restituisce un'istanza della classe `TxtaiPromptManagerAdapter`;
 - `create_prompt_manager(file_repository: FileRepository)`: restituisce un'istanza della classe `TxtaiPromptManagerAdapter`, creata dopo aver generato un'istanza di `TxtaiIndexManagerAdapter`.
- **Dipendenze:**
 - FileRepository;
 - IndexManagerPort;
 - PromptManagerPort.

6.2.5.7 TxtailIndexManagerAdapter

TxtailIndexManagerAdapter
- file_repository: FileRepository - indexes_out_file_base_path: string - embeddings: Embeddings
- TxtailIndexManagerAdapter(file_repository: FileRepository, table_path: string, column_path: string) - index_file_path(dictionary_id: integer): string

Figura 6.25: Diagramma della classe TxtailIndexManagerAdapter

- **Descrizione:** TxtailIndexManagerAdapter gestisce le operazioni *CRUD*_e sugli indici utilizzando *txtai*_e;
- **Interfaccia implementata:** IndexManagerPort;
- **Attributi:**
 - file_repository: istanza del gestore dei file;
 - indexes_out_file_base_path: percorso della cartella contenente gli indici;
 - embeddings: istanza di Embeddings utilizzata per la gestione degli embeddings.
- **Metodi:**
 - TxtailIndexManagerAdapter(file_repository: FileRepository, table_path: string, column_path: string): costruttore della classe;
 - + get_embeddings: recupera l'istanza di Embeddings;
 - + create_or_load_index(dictionary_id: integer): crea o ripristina l'indice per un dizionario;
 - + create_index(dictionary_id: integer, save_index: bool): crea un nuovo indice per un dizionario;
 - + save_index(dictionary_id: integer): salva l'indice associato a un dizionario;
 - + load_index(dictionary_id: integer): ripristina l'indice associato a un dizionario;
 - + delete_index(dictionary_id: integer): elimina l'indice associato a un dizionario;
 - - index_file_path(dictionary_id: integer): restituisce il percorso completo dell'indice associato a un dizionario.
- **Dipendenze:**



- FileRepository.

6.2.5.8 TxtaiPromptManagerAdapter

TxtaiPromptManagerAdapter
- index_manager: IndexManagerPort - file_repository: FileRepository - search_algorithm: SearchAlgorithmPort
- TxtaiPromptManagerAdapter(index_manager: IndexManagerPort, file_repository: FileRepository)

Figura 6.26: Diagramma della classe TxtaiPromptManagerAdapter

- **Descrizione:** TxtaiPromptManagerAdapter gestisce la generazione di *prompt_e* utilizzando *txtai_e*;
- **Interfaccia implementata:** PromptManagerPort;
- **Attributi:**
 - index_manager: istanza del gestore degli *indici_e*;
 - file_repository: istanza del gestore dei file;
 - search_algorithm: istanza della classe utilizzata per gestire le operazioni di ricerca semantica.
- **Metodi:**
 - TxtaiPromptManagerAdapter(index_manager: IndexManagerPort, file_repository: FileRepository): costruttore della classe;
 - + prompt_generator(dictionary_id: integer, user_request: string, lang: string, dbms: string, activate_log: bool): carica l'indice associato a un dizionario e genera un prompt basato sulla richiesta dell'utente;
 - + get_index_manager(): restituisce l'istanza del gestore degli indici.
- **Dipendenze:**
 - IndexManagerPort;
 - FileRepository;
 - SearchAlgorithmPort.



6.2.5.9 TxtaiSearchAlgorithmAdapter

TxtaiSearchAlgorithmAdapter
- index_manager: IndexManagerPort
- TxtaiSearchAlgorithmAdapter(index_manager: IndexManagerPort) + search_filtering(tuples: list, activate_log: bool): list - semantic_search_log(user_request: string, tuples: list): list - search_filtering_log(relevant_tuples: list, tuples: list): list - get_debug_header(level: string, system: string): string

Figura 6.27: Diagramma della classe TxtaiSearchAlgorithmAdapter

- **Descrizione:** TxtaiSearchAlgorithmAdapter fornisce funzionalità di ricerca semantica e di *logging* del processo di ricerca;
- **Interfaccia implementata:** SearchAlgorithmPort;
- **Attributi:**
 - index_manager: istanza del gestore degli *indici*.
- **Metodi:**
 - TxtaiSearchAlgorithmAdapter(index_manager: IndexManagerPort): costruttore della classe;
 - + semantic_search(user_request: string, activate_log: bool): restituisce i risultati della ricerca semantica effettuata con *txtai*;
 - + search_filtering(tuples: list, activate_log: bool): restituisce le tabelle considerate rilevanti dall'algoritmo di filtraggio implementato dal team;
 - - semantic_search_log(user_request: string, tuples: list): restituisce il *log* della ricerca semantica effettuata con *txtai*;
 - - search_filtering_log(relevant_tuples: list, tuples: list): restituisce il log dell'algoritmo di filtraggio implementato dal team;
 - - get_debug_header(level: string, system: string): costruisce l'intestazione del log.
- **Dipendenze:**
 - IndexManagerPort.

6.2.5.10 FileFactory



Figura 6.28: Diagramma della classe FileFactory

- **Descrizione:** FileFactory si occupa della creazione di istanze di gestori di file, come JsonFileAdapter, basandosi sulla configurazione fornita;
- **Metodi:**
 - + create(config: dict): restituisce un'istanza del gestore di file in base al valore associato alla chiave "file_type".
- **Dipendenze:**
 - FileRepository.

6.2.5.11 JsonFileAdapter

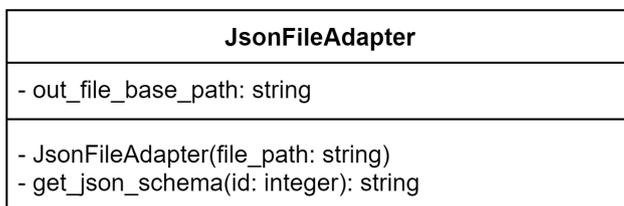


Figura 6.29: Diagramma della classe JsonFileAdapter

- **Descrizione:** JsonFileAdapter gestisce le operazioni *CRUD* sui file associati ai *dizionari dati*;
- **Interfaccia implementata:** FileRepository;
- **Attributi:**
 - file_path: percorso della cartella contenente i file dei dizionari dati;
- **Metodi:**
 - JsonFileAdapter(file_path: string): costruttore della classe;
 - + save(id: integer, file: string): salva il file;
 - + get_file_path(id: integer): restituisce il percorso del file;



- + `delete(id: integer)`: elimina il file;
- + `get_preview(id: integer)`: recupera un'anteprima del file;
- + `extract_index_metadata(id: integer)`: estrae i metadati per l'*indicizzazione*_e;
- + `extract_schema_metadata(id: integer, tuples: list)`: estrae i metadati in forma di *prompt*_e;
- - `get_json_schema(id: integer)`: restituisce lo schema del file *JSON*_e.

• **Dipendenze:**

- Utils.

6.2.6 Database

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

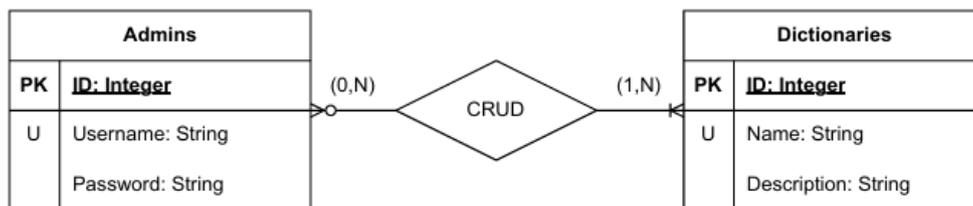


Figura 6.30: Diagramma E-R dell'interazione tra Admins (Tecnici) e *dizionari dati*_e per le operazioni *CRUD*_e.

6.2.6.1 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**: campo univoco.

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

6.2.6.2 Tabella Admins

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

- **ID (chiave primaria)**: un identificatore univoco per ogni amministratore, di tipo integer;
- **Username**: lo username univoco 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*_e sui dizionari dati.

6.2.6.3 Tabella Dictionaries

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

- **ID (chiave primaria)**: un identificatore univoco per ogni dizionario, di tipo integer;
- **Name**: il nome univoco del dizionario, di tipo string;
- **Description**: la descrizione del dizionario, di tipo string.

6.2.6.4 Relazione

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

- 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.

6.2.7 Design pattern

6.2.7.1 Introduzione La seguente sezione illustra i design pattern utilizzati durante lo sviluppo dell'architettura. Ciascun pattern è corredato da:

- Una breve descrizione;
- Le motivazioni che hanno spinto il gruppo ad adottarlo;
- I segmenti di codice in cui il pattern è stato utilizzato.

6.2.7.2 Abstract Factory

- **Descrizione**: il pattern *Abstract Factory* è un design pattern creazionale che fornisce un'interfaccia per creare famiglie di oggetti correlati o interdipendenti senza dover specificare le loro classi concrete. Il suo obiettivo principale è raggruppare la creazione di oggetti appartenenti alla stessa famiglia, garantendo la coerenza tra gli oggetti prodotti. Il processo di creazione viene incapsulato in una classe dedicata, nascondendo i dettagli dell'implementazione al codice client. Questo consente di cambiare facilmente l'implementazione di una famiglia di prodotti senza modificare il codice client;



- **Utilizzo:** questo pattern è stato adottato per astrarre la creazione dei componenti chiave del sistema, offrendo maggiore flessibilità e semplificando l'integrazione di nuove implementazioni. In particolare, sono state utilizzate due factory per gestire la creazione di `database`, ed `embeddings` manager.
 - `DbManagerFactory`: utilizzata per astrarre la creazione dei diversi tipi di database manager;
 - `EmbeddingsManagerFactory`: gestisce la creazione degli embeddings manager, con supporto per diversi algoritmi di embeddings.

6.2.7.3 Dependency Injection

- **Descrizione:** il pattern *Dependency Injection* è un design pattern architetturale che consente di iniettare le dipendenze di un oggetto dall'esterno, anziché crearle internamente. Questo approccio facilita il testing, migliora la manutenibilità e promuove il riuso del codice, poiché separa la logica di un componente dalla risoluzione delle sue dipendenze. L'iniezione delle dipendenze può avvenire attraverso vari meccanismi, tra cui costruttori o metodi setter, permettendo così una gestione più flessibile e modulare delle dipendenze all'interno del sistema;
- **Utilizzo:**
 - Route di `FastAPI` con `JwtBearer`: `FastAPI` utilizza `Dependency Injection` per gestire la validazione del token di autenticazione tramite `JwtBearer`. Le route di `FastAPI` iniettano automaticamente il token `JWT`, che viene poi utilizzato per autenticare e autorizzare le richieste degli utenti. Questo approccio consente di mantenere la logica di autenticazione separata dalla logica di business, semplificando la gestione della sicurezza e rendendo più agevole l'aggiornamento delle politiche di autenticazione;
 - Iniezione del servizio in `FastAPI`: un altro esempio di `Dependency Injection` in `FastAPI` è l'iniezione di servizi nelle route;
 - Classi `SqlAlchemyAuthenticationRepositoryAdapter` e `SqlAlchemyDictionaryRepositoryAdapter`:
 - * La classe `SqlAlchemyAuthenticationRepositoryAdapter`, che implementa l'interfaccia `AuthenticationRepository`, utilizza `Dependency Injection` per ottenere una sessione di database (`SessionLocal`) da `SQLAlchemy`. Questo permette di accedere ai dati relativi agli utenti autenticati senza dover gestire direttamente la configurazione del database;
 - * La classe `SqlAlchemyDictionaryRepositoryAdapter`, che implementa l'interfaccia `DictionaryRepository`, segue lo stesso principio, iniettando la sessione di database nel costruttore. Questo approccio consente di isolare la logica di accesso ai dati e semplifica il testing, poiché le sessioni possono essere facilmente sostituite con `mock` o altre implementazioni;
 - * L'iniezione della sessione di database attraverso il costruttore mantiene il codice pulito ed evita la creazione di dipendenze rigide, migliorando così la manutenibilità e testabilità del sistema.



- Embeddings manager: le dipendenze sono dichiarate come parametri del costruttore;
- Services: le dipendenze sono dichiarate come parametri del costruttore.

6.2.7.4 Port e Adapter (architettura esagonale)

- **Descrizione:** il pattern *Ports and Adapters*, noto anche come *Hexagonal Architecture*, è un design pattern architetturale che separa le interfacce di input e output dal core dell'applicazione. Questo pattern promuove la separazione delle responsabilità e mantiene il core isolato e robusto, agevolando la manutenibilità e il testing;
- **Utilizzo:** le porte e gli adapter incoming vengono utilizzati per validare i dati in ingresso e prepararli per il core dell'applicazione. Al contrario, le porte e gli adapter outgoing gestiscono le interazioni tra il core applicativo e le dipendenze esterne.
 - SchemaValidatorUseCase: interfaccia per la validazione dei dati in ingresso;
 - PromptManagerPort, SearchAlgorithmPort, IndexManagerPort: interfacce per l'interazione con i servizi esterni relativi agli *embeddings*;
 - AuthenticationRepository, DictionaryRepository, FileRepository: interfacce per l'interazione con i sistemi di persistenza dei dati.

6.2.7.5 Service Layer

- **Descrizione:** il pattern *Service Layer* è un design pattern architetturale che incapsula la logica di business all'interno di uno strato dedicato ai servizi. Questi servizi implementano i *casi d'uso* dell'applicazione e fungono da intermediari tra i controller e gli adapter, assicurando modularità, testabilità e manutenibilità del codice;
- **Utilizzo:**
 - DictionaryService: implementa i casi d'uso di gestione dei dizionari;
 - PromptManagerService: implementa i casi d'uso di generazione del *prompt*;
 - AuthenticationService: implementa i casi d'uso di autenticazione.

6.2.7.6 Repository

- **Descrizione:** il pattern *Repository* è un design pattern che funge da intermediario tra la logica di business e l'archiviazione dei dati. Questo pattern fornisce un'interfaccia simile a una collezione per accedere ai dati provenienti da *database* o altre fonti esterne, mantenendo il core dell'applicazione indipendente dai dettagli implementativi;
- **Utilizzo:**
 - DictionaryRepository: gestisce le informazioni relative ai dizionari;



- `AuthenticationRepository`: gestisce la persistenza e il recupero dei dati di autenticazione;
- `FileRepository`: gestisce le operazioni sui file associati ai dizionari.

6.2.7.7 Strategy

- **Descrizione:** il pattern *Strategy* è un design pattern comportamentale che definisce una famiglia di algoritmi, li incapsula singolarmente e li rende intercambiabili. Questo pattern permette di selezionare dinamicamente l'algoritmo da utilizzare al momento dell'esecuzione. Nel contesto di ChatSQL, il design pattern *Strategy* consente di cambiare l'algoritmo di ricerca semantica senza dover modificare il codice che lo utilizza;
- **Utilizzo:**
 - `SearchAlgorithmPort`.

6.2.8 DTO (Data Transfer Objects)

6.2.8.1 Introduzione

Questa sezione descrive le classi che rappresentano *DTO* (Data Transfer Objects), ossia design pattern utilizzati per il trasferimento dei dati tra gli strati dell'applicazione. I DTO sono dei contenitori semplici e privi di logica di business, progettati per aggregare dati e facilitare la comunicazione tra i componenti del sistema.

6.2.8.2 ResponseStatusEnum

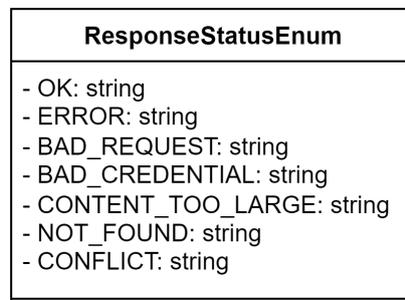


Figura 6.31: Diagramma della classe `ResponseStatusEnum`

`ResponseStatusEnum` è un'enumerazione che rappresenta i possibili stati di risposta del sistema. Le coppie chiave-valore sono basate sui codici di stato HTTP.

6.2.8.3 ResponseDto

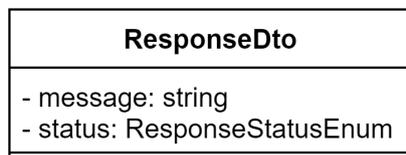


Figura 6.32: Diagramma della classe ResponseDto

ResponseDto è un DTO utilizzato per rappresentare una risposta standardizzata del sistema. La classe contiene un messaggio esplicativo e un'istanza di ResponseStatusEnum.

6.2.8.4 StringDataResponseDto



Figura 6.33: Diagramma della classe StringDataResponseDto

StringDataResponseDto è un DTO utilizzato per rappresentare una risposta del sistema che include dati testuali. La classe estende ResponseDto.

6.2.8.5 AuthResponseDto

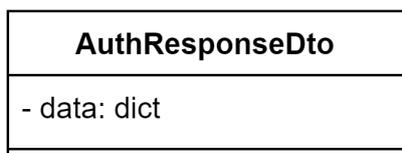


Figura 6.34: Diagramma della classe AuthResponseDto

AuthResponseDto è un DTO utilizzato per rappresentare la risposta del sistema a una richiesta di autenticazione. La classe estende ResponseDto.

6.2.8.6 DictionaryResponseDto

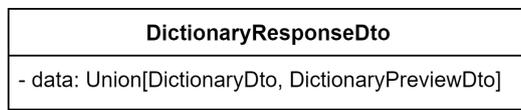


Figura 6.35: Diagramma della classe DictionaryResponseDto

DictionaryResponseDto è un DTO utilizzato per rappresentare una risposta del sistema che include dati relativi a uno specifico dizionario. Le informazioni possono includere le caratteristiche di un dizionario o un'anteprima del suo contenuto. La classe estende ResponseDto.

6.2.8.7 DictionariesResponseDto

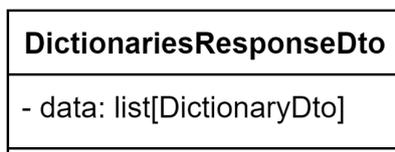


Figura 6.36: Diagramma della classe DictionariesResponseDto

DictionariesResponseDto è un DTO utilizzato per rappresentare una risposta del sistema contenente una lista di dizionari con le loro caratteristiche specifiche. La classe estende ResponseDto.

6.2.8.8 PromptResponseDto



Figura 6.37: Diagramma della classe PromptResponseDto

PromptResponseDto è un DTO utilizzato per rappresentare una risposta del sistema che include dati relativi a un *prompt*. La classe estende ResponseDto.

6.2.8.9 LoginDto

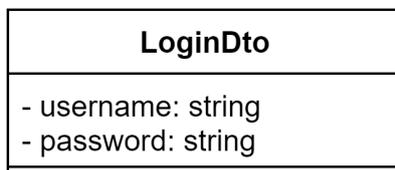


Figura 6.38: Diagramma della classe LoginDto

LoginDto è un DTO utilizzato per gestire i dati di autenticazione durante il processo di login. Le credenziali di accesso includono lo username e la password.

6.2.8.10 AdminDto

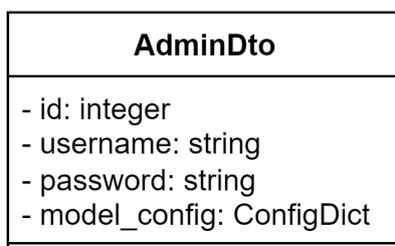


Figura 6.39: Diagramma della classe AdminDto

AdminDto è un DTO utilizzato per modellare i dati degli utenti con privilegi di amministratore nel sistema. Oltre a username e password, il DTO contiene un identificatore numerico univoco.

6.2.8.11 DictionaryDto

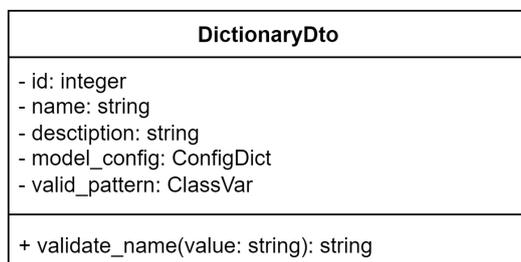


Figura 6.40: Diagramma della classe DictionaryDto

DictionaryDto è un DTO utilizzato per modellare le informazioni relative a un dizionario, inclusi il nome e la descrizione. La classe contiene un metodo di validazione per il campo "name".

6.2.8.12 DictionaryPreviewDto

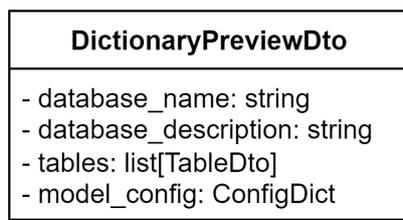


Figura 6.41: Diagramma della classe DictionaryPreviewDto

DictionaryPreviewDto è un DTO utilizzato per fornire una panoramica di un *dizionario dati*. L'anteprima include il nome e la descrizione del *database*, insieme a una lista di tabelle con le loro informazioni.

6.2.8.13 TableDto

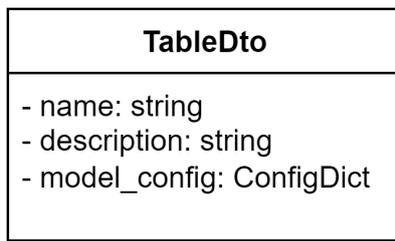


Figura 6.42: Diagramma della classe TableDto

TableDto è un DTO utilizzato per rappresentare i dati relativi a una tabella contenuta all'interno di un *dizionario dati*. Ogni tabella è corredata da un nome e da una descrizione.

6.2.8.14 PromptDto

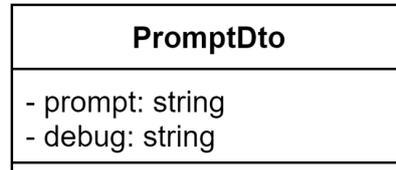


Figura 6.43: Diagramma della classe PromptDto

PromptDto è un DTO utilizzato per modellare il *prompt* generato dal sistema. La classe contiene due campi di tipo stringa: il testo del prompt e le informazioni di *debug* relative ad esso.