torchlight.swe2324@outlook.com

Specifica Tecnica

Versione 1.0.0

Redattori	Cappellari Marco
	Filippini Giovanni
	Meneghini Fabio
	Ye Tao Ren Federico
Verifica	Agafitei Ciprian
	Cappellari Marco
	Filippini Giovanni
	Meneghini Fabio
	Pluzhnikov Dmitry
Approvazione	De Laurentis Arianna Pia
Uso	Esterno
Destinatari	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Zucchetti S.p.A.

Registro delle Modifiche

Ver.	Data	Descrizione	Autore	Verifica
1.0.0	2024/04/09	Verifica finale e convalida del documento per la PB	Pluzhnikov Dmitry	Pluzhnikov Dmitry
0.3.2	2024/04/09	Stesura e completamento della sezione §5 e della sezione §5.2	Capellari Marco	Meneghini Fabio
0.3.1	2024/04/08	Stesura e completamento della sottosezione §3.1	De Laurentis Arianna Pia	Meneghini Fabio
0.3.0	2024/04/08	Verifica e validazione dello stato del documento	Ye Tao Ren Federico	Ye Tao Ren Federico
0.2.4	2024/03/07	Completata la stesura di §4 con il completamento della sottosezione §4.3	Meneghini Fabio	Agafitei Ciprian
0.2.3	2024/03/06	Completata la stesura di §2 con l'aggiunta di Docker all'interno della sottosezione §2.1	Meneghini Fabio	Agafitei Ciprian
0.2.2	2024/03/05	Terminata la stesura della sottosezione §4.2	Agafitei Ciprian	Capellari Marco
0.2.1	2024/03/04	Terminata la stesura della sottosezione §4.1	Agafitei Ciprian	Capellari Marco

Ver.	Data	Descrizione	Autore	Verifica
0.2.0	2024/03/04	Verifica e validazione dello stato del documento	Pluzhnikov Dmitry	Pluzhnikov Dmitry
0.1.5	2024/03/04	Continuata la stesura di §4.1	Meneghini Fabio	De Laurentis Arianna Pia
0.1.4	2024/03/03	Iniziata la stesura della sezione §4	Meneghini Fabio	De Laurentis Arianna Pia
0.1.4	2024/03/03	Terminata la stesura della sezione Introduzione §1	Meneghini Fabio	De Laurentis Arianna Pia
0.1.3	2024/04/02	Inizio della stesura dell'architettura	Agafitei Ciprian	De Laurentis Arianna Pia
0.1.2	2024/03/01	Iniziata la stesura della sezione Tecnologie §2	Agafitei Ciprian	Filippini Giovanni
0.1.1	2024/03/31	Definizione della struttura e delle sezioni e iniziata la stesura della sezione §1	Filippini Giovanni	Meneghini Fabio
0.1	2024/02/29	Creazione del documento	Filippini Giovanni	Pluzhnikov Dmitry

Tabella 1: Registro delle modifiche

Indice

1 Introduzione **1**

1.1 Scopo del documento . . . 1

1.2 Scopo del prodotto . . . 1

1.3 Glossario . . . 1

1.4 Miglioramenti al documento . . . 2

1.5 Riferimenti . . . 2

1.5.1 Riferimenti normativi . . . 2

1.5.2 Riferimenti informativi . . . 2

2 Tecnologie coinvolte **3**

2.1 Tecnologie utilizzate per la codifica . . . 3

2.1.1 Linguaggi . . . 3

2.1.2 Librerie . . . 3

2.1.3 Strumenti di supporto . . . 4

2.2 Strumenti per l'analisi del codice . . . 5

2.2.1 Strumenti per l'analisi statica . . . 5

2.2.2 Strumenti per l'analisi dinamica . . . 5

3 Architettura **6**

3.1 Architettura di dettaglio . . . 8

3.1.1 Architettura dell'autenticazione . . . 8

3.1.2 Generazione della risposta . . . 9

3.1.3 Architettura della gestione dei dizionari . . . 10

3.2 Pattern utilizzati . . . 11

3.2.1 Strategy . . . 11

3.2.2 Adapter . . . 12

4 Descrizione delle classi **13**

4.1 Model . . . 13

4.1.1 DictionarySchemaVerifierService . . . 13

4.1.2 JsonSchemaVerifierService . . . 13
4.1.3 UploadService . . . 14
4.1.4 SelectionService . . . 14
4.1.5 DeleteService . . . 14
4.1.6 AuthenticationService . . . 15
4.1.7 ChatService . . . 15
4.1.8 UserResponse . . . 15
4.1.9 TechnicianResponse . . . 16
4.1.10 Embedder . . . 16
4.2 View . . . 17
4.2.1 LoginWidget . . . 17
4.2.2 LogoutWidget . . . 17
4.2.3 SelectWidget . . . 18
4.2.4 UploadWidget . . . 18
4.2.5 DeleteWidget . . . 19
4.2.6 ChatWidget . . . 19
4.3 Controller . . . 20
4.3.1 AuthenticationController . . . 21
4.3.2 SelectionController . . . 21
4.3.3 UploadController . . . 21
4.3.4 DeleteController . . . 21
4.3.5 LogoutController . . . 22
4.3.6 ChatController . . . 22

5 Stato dei requisiti funzionali 23
5.1 Requisiti funzionali . . . 23
5.2 Requisiti soddisfatti . . . 30

Elenco delle figure

1 Rappresentazione dell'MVC Pull 6
2 Architettura del servizio d'autenticazione 8
3 Architettura della generazione delle risposte 9
4 Architettura della gestione dei dizionari 10

Elenco delle tabelle

1 Registro delle modifiche iii
2 Linguaggi utilizzati per la codifica 3
3 Librerie utilizzate per la codifica 3
4 Strumenti di supporto per la codifica 4
5 Strumenti utilizzati per l'analisi statica del codice 5
6 Strumenti utilizzati per l'analisi dinamica del codice 5
7 Requisiti funzionali 30

1 Introduzione

1.1 Scopo del documento

Questo documento ha lo scopo di servire da linea guida per gli sviluppatori che andranno ad estendere o mantenere il prodotto. Di seguito lo sviluppatore troverà nel documento tutte le informazioni riguardanti i linguaggi e le tecnologie utilizzate, l'architettura del sistema e le scelte progettuali effettuate per il prodotto.

1.2 Scopo del prodotto

Nell'ultimo anno vi è stato un cambiamento repentino nello sviluppo e nell'applicazione dell'*Intelligenza Artificiale*$_G$, passando dall'elaborazione e raccomandazione dei contenuti alla generazione di essi, come immagini, testi e tracce audio.

Il *capitolato*$_G$ C9, "ChatSQL", pone come obiettivo la realizzazione di un'applicazione che permetta, partendo dalla descrizione di un *database*$_G$ la generazione di un *prompt*$_G$ in risposta ad un'interrogazione in linguaggio naturale. Questo processo dovrà essere gestito da un *LLM*$_G$ (Large Language Model) specificatamente addestrato in questo scopo.

Tale applicazione sarà fruibile attraverso una *web app*$_G$, dove l'utente potrà caricare un file in formato *JSON*$_G$ e richiedere, con linguaggio naturale, la produzione di prompt riguardanti il documento caricato.

1.3 Glossario

Al fine di evitare possibili ambiguità relative al linguaggio utilizzato nei documenti, viene fornito un *Glossario* (attualmente alla sua versione *2.0.0*), nel quale sono contenute le definizioni di termini aventi uno specifico significato.
Tali termini sono segnati in corsivo e marcati con il simbolo "$_G$" a pedice (per esempio: *Way of Working*$_G$) alla prima occorrenza in ogni capitolo.

1.4 Miglioramenti al documento

La maturità e i miglioramenti sono aspetti fondamentali nella stesura di un documento. Questo permette di apportare agevolmente modifiche in base alle esigenze concordate tra i membri del gruppo e il *proponente*$_G$, nel corso del tempo. Di conseguenza, questa versione del documento non può essere considerata definitiva o completa, poiché è soggetta a evoluzioni future.

1.5 Riferimenti

1.5.1 Riferimenti normativi

- Norme di progetto (v 2.0.0)
- Capitolato d'appalto C9 - ChatSQL
- Slide PD2 del corso di Ingegneria del Software - Regolamento del Progetto Didattico

1.5.2 Riferimenti informativi

- Slide del corso di Ingegneria del Software - Diagrammi delle classi
- Wikipedia - Model–view–controller design pattern
- Documentazione Streamlit - API Reference
- Documentazione Streamlit - Understanding Streamlit's client-server architecture

2 Tecnologie coinvolte

Questa sezione fornisce un'analisi esaustiva delle tecnologie impiegate nel progetto in questione, comprendendo le procedure, gli strumenti e le librerie necessarie per lo sviluppo, il testing e la distribuzione del prodotto. Saranno discusse le tecnologie utilizzate per implementare sia il front-end che il back-end, la gestione del dati e l'integrazione con i servizi previsti.

2.1 Tecnologie utilizzate per la codifica

2.1.1 Linguaggi

Nome	Versione	Descrizione
Python	3.11.6	Linguaggio di programmazione ad alto livello, interpretato, orientato agli oggetti e multiparadigma.

Tabella 2: Linguaggi utilizzati per la codifica

2.1.2 Librerie

Nome	Versione	Descrizione
Streamlit	1.29.0	Libreria Python open source per la creazione di web app personalizzabili.
Txtai	6.2.0	Libreria open source per l'elaborazione del linguaggio naturale (NLP) in Python.
JSONSchema	4.20.0	Libreria Python usata per verificare la pertinenza di un file in formato JSON con un dato schema
Psutil	5.9.7	Libreria per il recupero di informazioni sui processi in esecuzione e sull'utilizzo del sistema
Keyboard	0.13.5	Permette di avere accesso completo alla tastiera, full control of your keyboard with this small Python library

Tabella 3: Librerie utilizzate per la codifica

2.1.3 Strumenti di supporto

Nome	Versione	Descrizione
Git	2.43.0	Sistema di controllo versione distribuito utilizzato per il tracciamento delle modifiche apportate al codice sorgente durante lo sviluppo del prodotto.
pip	24.0	sistema di gestione dei pacchetti, utilizzato per installare e gestire i pacchetti software per ampliare le capacità del programma
Docker	20.0.3	Sistema di controllo versione distribuito utilizzato per il tracciamento delle modifiche apportate al codice sorgente durante lo sviluppo del prodotto.

Tabella 4: Strumenti di supporto per la codifica

2.2 Strumenti per l'analisi del codice

2.2.1 Strumenti per l'analisi statica

Nome	Versione	Descrizione
Pylint	3.1.0	Pylint è un analizzatore di codice statico per , e permette di analizza il codice senza eseguirlo. Controlla la presenza di errori, applica uno standard di codifica e cerca dare suggerimenti su come il codice potrebbe essere sistemato.
Sonarlint	4.4.2	Si tratta di un'estensione per IDE che aiuta a rilevare e correggere i problemi di qualità durante la scrittura del codice, individuando i difetti in modo che possano essere corretti prima del commit del codice.

Tabella 5: Strumenti utilizzati per l'analisi statica del codice

2.2.2 Strumenti per l'analisi dinamica

Nome	Versione	Descrizione
pytest	8.1.1	Framework di test Python estremamente versatile che permette di essere adattato alle esigenze del team. Inoltre, è integrato con Stremalit e permette di tesate anche le interazione con l'applicazione.

Tabella 6: Strumenti utilizzati per l'analisi dinamica del codice

3 Architettura

L'architettura del programma ChatSQL è basata sul pattern architetturale Model-View-Controller (MVC_G), scrivendo e strutturando il pattern da zero, senza l'utilizzo di librerie e/o framework esterni, fornendo una struttura organizzativa chiara e separando le responsabilità all'interno dell'applicazione per una gestione più efficiente del codice e una migliore manutenibilità del software.

La variante *Pull* di MVC si distingue per la direzione del flusso di dati tra i componenti del pattern.

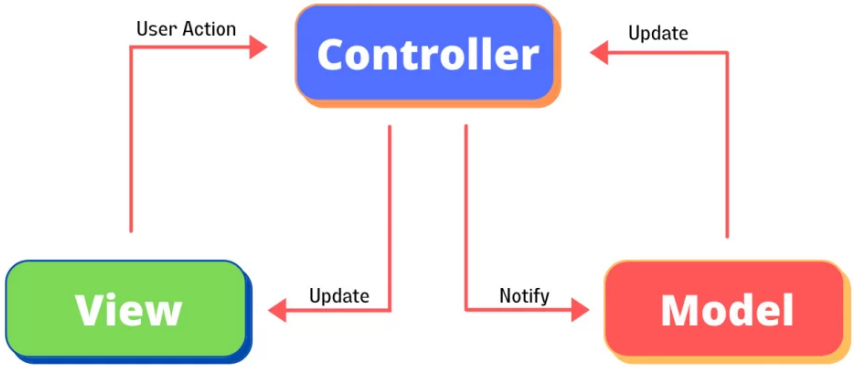

Figura 1: Rappresentazione dell'MVC Pull

Nello specifico, il pattern architetturale è così strutturato:

- **Modello (Model)**: Il Modello rappresenta la parte del sistema che gestisce i dati e la logica di business dell'applicazione. Esso è responsabile della manipolazione dei dati, dell'accesso ai dati persistenti, dell'applicazione delle regole di business e della gestione degli stati dell'applicazione. Il Modello, nel pattern MVC Pull, non è direttamente coinvolto nella comunicazione con la Vista: è passivo e non conosce nulla della Vista o del Controller.
- **Vista (View)**: La Vista rappresenta l'interfaccia utente dell'applicazione e visualizza graficamente i dati provenienti dal Modello. La Vista non manipola direttamente i dati ma li richiede al Modello quando è necessario aggiornare la visualizzazione.

- **Controller**: Il Controller funge da intermediario tra la Vista e il Modello. Esso risponde alle azioni dell'utente e coordina le interazioni tra la Vista e il Modello: quando l'utente compie un'azione nell'interfaccia utente, il Controller riceve l'input, determina come rispondere e, se necessario, richiede i dati al Modello, e una volta ottenuti, li passa alla Vista per essere visualizzati nell'interfaccia grafica.
- **Flusso di dati**: Nel MVC Pull, il flusso di dati avviene principalmente in direzione Controller `->` Modello `->` Vista. In particolare:
  - Quando l'utente interagisce con l'interfaccia utente (Vista), il Controller riceve l'input e prende le decisioni appropriate;
  - Il Controller può richiedere i dati al Modello, che li fornisce senza sapere nulla della Vista;
  - La Vista, in base alle istruzioni del Controller, può richiedere i dati al Modello per aggiornare la visualizzazione.

3.1 Architettura di dettaglio

3.1.1 Architettura dell'autenticazione

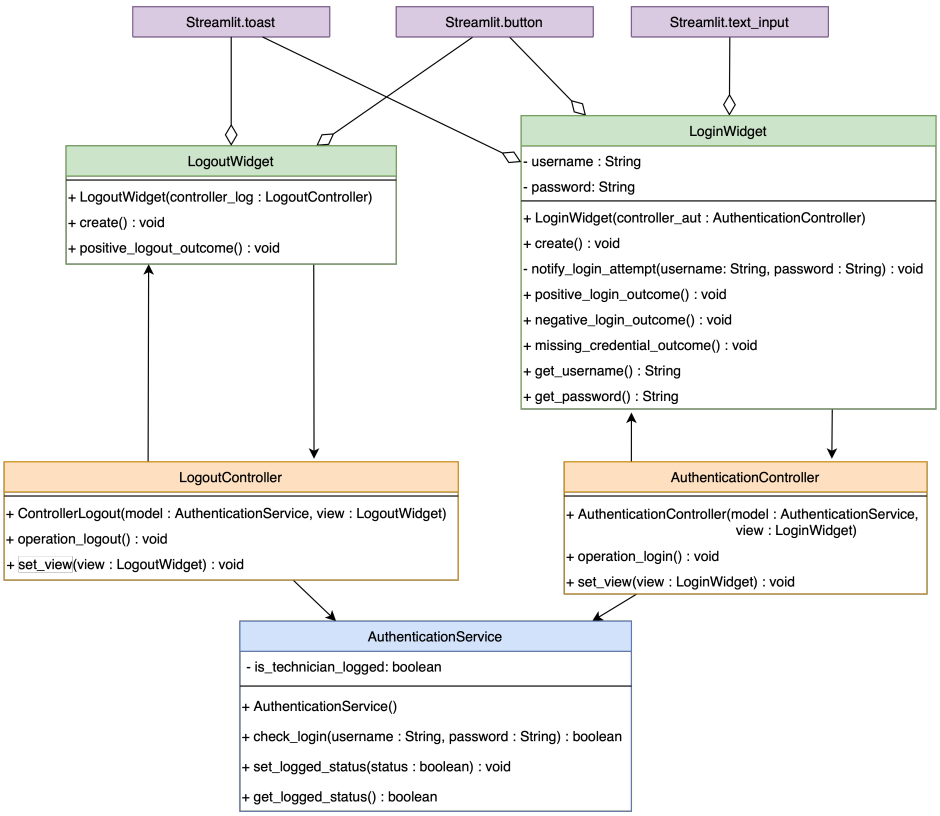

Figura 2: Architettura del servizio d'autenticazione

L'UML in questione rappresenta un sistema che facilita l'interazione degli utenti con il `LoginWidget` , il quale gestisce l'interfaccia di accesso consentendo agli utenti di inserire le proprie credenziali (username e password) tramite campi dedicati e un pulsante di conferma. Quando il Cliente preme il pulsante di conferma, tale azione viene notificata al `AuthenticationController` , responsabile della gestione della logica di autenticazione. Il `AuthenticationController` quindi recupera tali credenziali necessarie per passarle all' `AuthenticationService` , il quale si occupa di verificare la loro correttezza.

Il

`LogoutWidget` , come suggerito dal nome, contiene il widget per svolgere il logout a interfaccia. Esso presenta un pulsante dedicato all'effettuazione del logout. Una volta premuto il pulsante, il `LogoutWidget` notifica il

`LogoutController` , il quale comunica all' `AuthenticationService` di modificare lo stato di autenticazione memorizzato dentro di essa.

3.1.2 Generazione della risposta

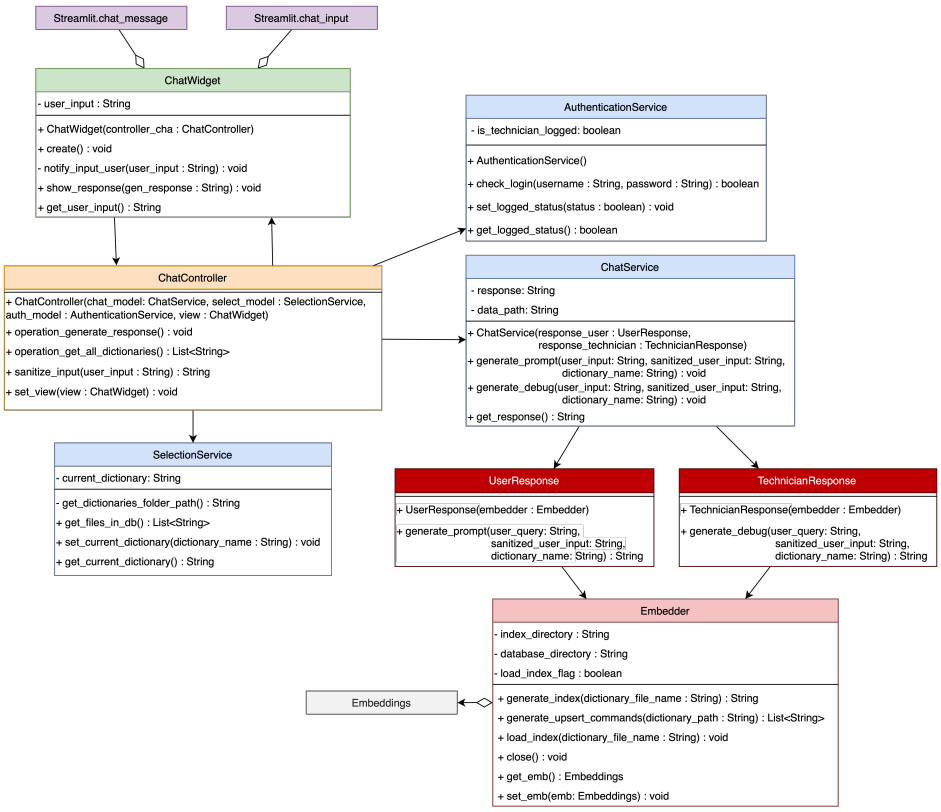

Figura 3: Architettura della generazione delle risposte

Il componente Model include il `ChatService`, responsabile dell'elaborazione dell'input dell'Utente generico e della generazione di risposte sia per il Cliente che per il Tecnico. Il `ChatService` si basa su due elementi chiave: i Dizionari, che rappresentano la struttura di un database, e gli `Embeddings`, vettori numerici che associano le parole a significati specifici per facilitare la generazione di risposte pertinenti.

Altri componenti del Model includono il `SelectionService`, che gestisce la selezione del dizionario più adatto, e l'`AuthenticationService`, che verifica le credenziali dei tecnici per l'accesso a funzionalità avanzate. Modelli come `UserResponse` e `TechnicianResponse` definiscono risposte modellate rispettivamente per il Cliente e il Tecnico. Il componente View è rappresentato dal `ChatWidget`, l'interfaccia utente che permette all'Utente generico di inse-

rire input e visualizzare le risposte del sistema.

Il componente Controller, in particolare il `ChatController`, funge da orchestratore dell'interazione tra utente e modelli. Coordina le attività del `ChatService`, `SelectionService`, `AuthenticationService` e `ChatWidget` per garantire un flusso fluido di informazioni tra l'utente e il sistema.

3.1.3 Architettura della gestione dei dizionari

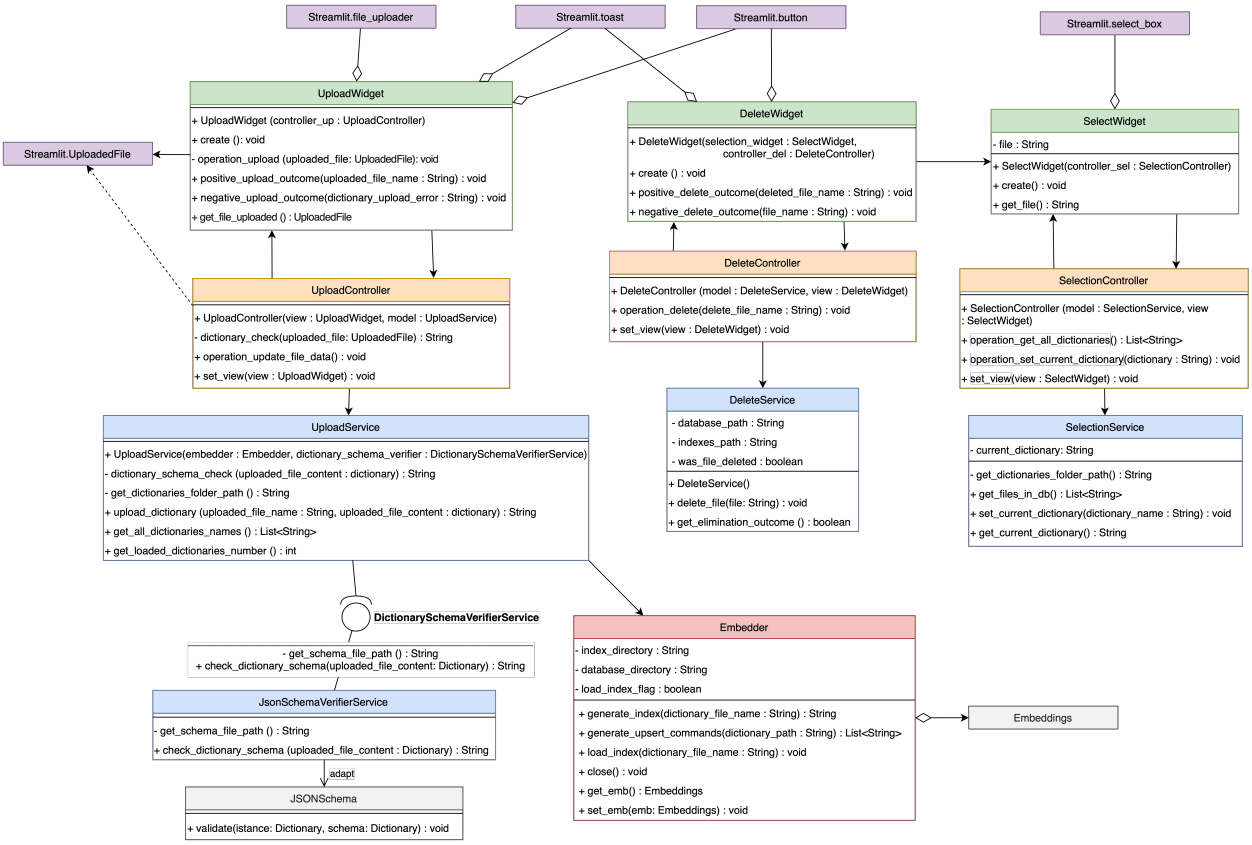

Figura 4: Architettura della gestione dei dizionari

Il sistema di gestione dizionari dati è suddiviso in diverse componenti, raggruppabili per funzionalità specifica che svolgono, ovvero le operazioni di caricamento, selezione e cancellazione dei dizionari dati. Per caricare un file, l'utente interagisce con la componente grafica di caricamento file disponibile a interfaccia, contenuta dentro la classe

`UploadWidget` , la quale notifica l'

`UploadController` dell'input dell'user. Il controller andrà a recuperare tale file e svolgere dei primi controlli di validazione, per esempio se sia in formato accettabile o se è già stato

salvato un file con lo stesso nome. Superati tali controlli, il file verrà passato all' `UploadService` e, se viene superata l'ultima verifica della struttura del contenuto svolta dalle interfacce concrete di `DictionarySchemaVerifierService` , la salverà a sistema, altrimenti comunicherà del fallimento. La classe controller riceverà una notifica sul risultato del processo di caricamento e comunicherà all'interfaccia come aggiornarsi per rispecchiare tale esito.
Per cancellare un documento, l'Utente generico seleziona il dizionario desiderato dalla lista, conferma l'intenzione di eliminare premendo l'apposito pulsante e il documento verrà rimosso dal sistema, sempre rispettando l'architettura MVC pull, perciò sarà `DeleteController` a recuperare il nome del file da cancellare e comunicare alla classe model `DeleteService` di eliminare il file e l'indice con tale nome, infine la vista viene aggiornata con un messaggio che comunica l'esito dell'eliminazione.
Per la selezione l'Utente generico semplicemente seleziona il dizionario su cui vuole porre le sue interrogazioni nella lista mostrata da una `select_box` il flusso delle informazioni è sempre lo stesso, ovvero è la classe `SelectionController` a comunicare alle classi delle View e del Model di aggiornarsi e modificarsi. La selezione di un nuovo dizionario corrente consiste nel modificare una variabile dove viene appunto salvata. Questo cambiamento avviene come conseguenza anche quando viene svolto un caricamento di un nuovo dizionario, il quale diviene il nuovo dizionario corrente, e quando viene svolta un'operazione di eliminazione seleziona l'ultimo dizionario caricato.

3.2 Pattern utilizzati

3.2.1 Strategy

Il pattern *Strategy* è stato adottato per permettere di gestire più file JSON di conformità. In particolare, è stato implementato il componente `DictionaryServiceVerifier` in modo da permettere la selezione di più file. Questa scelta architetturale ha permesso di separare la logica di verifica delle regole di conformità in diverse strategie, ognuna rappresentata da un'implementazione specifica del `DictionaryServiceVerifier`.

Questo approccio permette di avere una maggiore flessibilità nell'aggiunta e nella gestione

di nuove regole di conformità. Inoltre, grazie al pattern Strategy, è possibile migliorare la manutenibilità del sistema, in quanto ogni strategia di verifica è stata implementata in modo separato e coeso, facilitando eventuali modifiche o estensioni future.

Grazie a `DictionaryServiceVerifier` e al pattern Strategy, il sistema è in grado di gestire in modo efficiente e scalabile la complessità legata alle varie JSONSchema di conformità richieste, garantendo al contempo un alto livello di coerenza e coesione nell'implementazione della logica di verifica.

3.2.2 Adapter

L'utilizzo del pattern *Adapter* è particolarmente utile dal momento che, per la corretta strutturazione dell'applicativo, è necessario far aderire la classe `JSONSchema`, che fa parte di una libreria esterna, all'interfaccia `DictionarySchemaVerifier`: ciò è ottenuto grazie alla classe adapter intermedia `JsonSchemaVerifierService`, che ne implementa correttamente i metodi. In particolare, è stata usata la variante *Object Adapter*, in quanto permette di accoppiare la classe adapter e la classe adattata con una relazione di composizione, che comporta una dipendenza minore rispetto all'ereditarietà, che sarebbe stata necessaria nella variante *Class Adapter*.

4 Descrizione delle classi

La suddivisione in classi del progetto deriva dalla struttura del pattern architetturale *MVC*$_G$, descritto nella sezione precedente. In questa sezione verranno approfonditi i comportamenti e la logica di ciascuna classe facente parte dell'applicativo.

4.1 Model

Il Model rappresenta i dati dell'applicazione e le logiche di business associate. Il Modello è costituito dalle seguenti classi:

4.1.1 DictionarySchemaVerifierService

Questa è l'interfaccia che devono implementare tutte le classi che realizzano una versione dell'algoritmo di verifica di correttezza del dizionario dati che si vuole caricare nel server. Espone le signature dei metodi `get_schema_file_path(): str` e `check_dictionary_schema(uploaded_file_content: dictionary): str` . Questa classe permette di realizzare il design pattern *Strategy*$_G$, in quanto il team si riserva la possibilità di cambiare con facilità l'implementazione del metodo esposto dall'interfaccia, ovvero il recupero del percorso dello schema richiesto e l'algoritmo di verifica di correttezza del file caricato, dato che il formato del file accettato potrebbe cambiare o aumentare in futuro.

4.1.2 JsonSchemaVerifierService

Si tratta di una classe concreta che implementa l'interfaccia `DictionarySchemaVerifierService` . In particolare, questa classe realizza, tramite la sua implementazione del metodo `check_dictionary_schema(uploaded_file_content: dictionary): str` , il controllo di validità dei dizionari dati in formato *JSON*$_G$, cioè verifica che la struttura del file sia coerente con lo schema e che non presenti errori.

Per effettuare la verifica, tale metodo chiama prima il metodo `get_schema_file_path()` per recuperare il percorso dello schema JSON e poi esegue `validate(instance: dictionary, schema: dictionary)` della libreria `JSONSchema` ,

adattando così tale libreria all'interfaccia `DictionarySchemaVerifierService` secondo il design pattern *Adapter*$_G$.

4.1.3 UploadService

La classe `UploadService` si occupa di effettuare il caricamento dei dizionari dati e della generazione dei relativi file di *index* tramite il metodo `upload_dictionary(uploaded_file_name: str, uploaded_file_content:dictionary): str` . Prima di effettuare il caricamento effettivo viene verificata la validità del file tramite una chiamata al metodo `check_dictionary_schema` dell'attributo `dictionary_schema_verifier`.

Questa classe inoltre espone il metodo `get_loaded_dictionaries_number(): int` , che restituisce il numero di dizionari dati caricati, e il metodo `get_all_dictionaries_names()` , che restituisce la lista dei nomi di tutti i dizionari dati caricati.

4.1.4 SelectionService

Questa classe permette di tenere traccia del dizionario dati attualmente selezionato dall'Utente generico attraverso il metodo `set_current_dictionary(dictionary: str)`, che salva il nuovo dizionario corrente dentro all'attributo della classe `_current_dictionary` ogni qualvolta questo cambi dopo una nuova selezione, un caricamento o un'eliminazione di un dizionario. Possiede inoltre un metodo per recuperare tutti i dizionari attualmente salvati, `get_files_in_db()` , restituendoli ordinati dal più recentemente caricato a quello meno recente.

4.1.5 DeleteService

La classe `DeleteService` si occupa dell'eliminazione dei file dizionario dati caricati nel server. Per fare ciò, tale classe espone i metodi `delete_file(file: str)` , che permette di eliminare effettivamente il file specificato come parametro, e `get_elimination_outcome(): bool` , che restituisce l'esito dell'ultima operazione di eliminazione.

4.1.6 AuthenticationService

È la classe che si occupa dell'autenticazione. In particolare, il metodo `check_login(username: str, password: str): bool` permette di verificare la correttezza di username e password: in caso di esito positivo, viene modificato lo stato impostando a `True` l'apposito attributo booleano `_is_technician_logged`, inizialmente settato a `False`. Questa classe permette inoltre di effettuare il logout dalla sezione tecnico attraverso il metodo `set_logged_status(status): bool`, con valore False per il parametro `status`, impostando così `_is_technician_logged` a `False`. Per recuperare lo stato di autenticazione, la classe si avvale del metodo `get_logged_status(): bool`.

4.1.7 ChatService

Questa classe permette di generare il testo in risposta all'interrogazione fornita dall'Utente generico. Essa espone i seguenti tre metodi:

- `generate_prompt(user_input: str, sanitized_user_input: str, dictionary_name: str): str`: serve per generare il prompt per il Cliente. Per fare ciò, esso sfrutta l'omonimo metodo dell'attributo `_response_user` di tipo `UserResponse` di cui è composto, nel quale avviene l'effettivo algoritmo di generazione del prompt;
- `generate_debug(user_input: str,sanitized_user_input: str, dictionary_name: str): str`: questo metodo serve per generare il prompt con le informazioni di debug per il Tecnico. Analogamente al metodo descritto in precedenza, anch'esso sfrutta l'omonimo metodo del suo attributo `_response_technician`, di tipo `TechnicianResponse`, per produrre in modo effettivo la risposta;
- `get_response(): str`: viene usato per recuperare il prompt generato.

4.1.8 UserResponse

Questa classe presenta il metodo `generate_prompt(user_query, sanitized_user_input, dictionary_name)`, il quale esegue una ricerca delle tabelle e campi più rilevanti con l'interrogazione in linguaggio naturale fornita dal Cliente all'interno del vettore di embedding

relativo al dizionario dati di riferimento (attributo `_emb`, di tipo `Embedder`). Una volta ottenuti tali tabelle e campi, l'algoritmo procede con la costruzione effettiva della risposta da restituire al Cliente. Il prompt finale generato, contiene i nomi delle tabelle e dei campi più rilevanti con relativa descrizione, e nel caso siano coinvolte più tabelle, vengono incluse anche le varie relazioni che sussistono tra di esse.

4.1.9 TechnicianResponse

Questa classe presenta il metodo `generate_debug(user_query, sanitized_user_input, dictionary_name)`, che in modo analogo al metodo della classe descritta precedentemente, permette di generare in modo effettivo la risposta all'interrogazione fornita dal Tecnico. La ricerca all'interno del vettore di embedding funziona allo stesso modo, l'unica differenza è il formato della risposta, che include anche alcune informazioni di debug, come ad esempio i punteggi di similarità dei vari campi selezionati dalla ricerca, cioè un valore tra 0 e 1 che indica il livello di pertinenza di ciascun campo con l'input del Tecnico.

4.1.10 Embedder

Questa classe gestisce la generazione e il caricamento degli *indexes* per i dizionari dati. I metodi di `Embedder` sono i seguenti:

- `generate_index(dictionary_file_name: str)`: genera un *index* in cui vengono inseriti tutti i comandi di *upsert* relativi al dizionario dati il cui nome è specificato come parametro. Tale index viene salvato nel server al percorso indicato dall'attributo `_index_directory`;
- `generate_upsert_commands(dictionary_path: str)`: viene usato dal metodo precedente per generare i comandi di *upsert* relativi al dizionario dati il cui percorso è passato come parametro;
- `load_index(dictionary_file_name: str)`: carica l'index relativo al dizionario dati specificato come parametro;
- `close()`: chiude l'*index* corrente, ovvero quello il cui percorso è presente nell'attributo `_index_directory`;

- `get_emb()`: restituisce il valore dell'attributo `_emb`. In caso esso sia nullo, prima ne viene creato uno nuovo;
- `set_emb(emb: Embeddings)`: imposta un nuovo valore per l'attributo `_emb`.

4.2 View

La View del nostro applicativo è composta da sei classi, ognuna delle quali rappresenta un diverso *widget* dell'interfaccia grafica. Ciascuna di esse interagisce con le classi del Modello usando ciascuno un Controller dedicato che fa da tramite. Le classi che compongono la View sono le seguenti:

4.2.1 LoginWidget

Questa classe si occupa di visualizzare la maschera di input per le credenziali per effettuare il login necessario all'accesso alla sezione riservata al Tecnico. In particolare, tramite il metodo `create()` vengono visualizzati i due *input box* per inserire username e password e il pulsante per effettuare il login, ciascuno dei quali viene creato tramite la chiamata ad un opportuno metodo della libreria *Streamlit*$_G$. La classe `LoginWidget` espone inoltre tre metodi per notificare l'esito dell'operazione di login, in particolare sono esposti:

- `positive_login_outcome()`: segnala al Cliente il successo di tale operazione;
- `negative_login_outcome()`: segnala al Cliente il fallimento dell'operazione;
- `missing_credential_outcome()`: avvisa il Cliente del mancato inserimento di una delle due informazioni necessarie per il login.

Quando il Cliente preme il tasto per effettuare il login, il Controller di questo widget effettuerà opportuni controlli sulle credenziali, eventualmente utilizzando il Modello, per decidere l'esito dell'operazione e chiamare di conseguenza uno dei tre metodi presentati sopra.

4.2.2 LogoutWidget

`LogoutWidget` è la classe che permette di visualizzare il pulsante per effettuare il logout dalla sezione del tecnico. Tramite il metodo `create()` viene creato (attraverso un metodo della libreria *Streamlit*) e successivamente visualizzato il tasto per il logout. Quando il

Tecnico preme tale tasto, il relativo Controller, dopo aver modificato lo stato nel Modello (da Tecnico autenticato a non autenticato), chiama il metodo `positive_logout_outcome()` per notificare il Tecnico della corretta esecuzione di tale operazione.

4.2.3 SelectWidget

Questa classe permette di visualizzare a schermo il menù a tendina che permette all'Utente generico di selezionare il dizionario dati a cui fare riferimento. Una volta selezionato un dizionario dati tra quelli caricati nel server tramite questo widget, il sistema farà riferimento ad esso per la generazione della risposta in seguito ad un'interrogazione in linguaggio naturale fornita dall'Utente generico. Nel caso ci trovassimo nella sezione tecnico, sarà possibile anche eliminare il dizionario dati selezionato tramite un apposito widget (`DeleteWidget`), il cui funzionamento verrà descritto in seguito. Il widget viene creato tramite il metodo `create()`, che a sua volta chiama un metodo della libreria *Streamlit* per creare in modo effettivo e visualizzare il *select box*. La lista di tutti i dizionari dati che è possibile selezionare viene recuperata dal Controller del `SelectWidget` , che fa da tramite per accedere ai dati nel Modello. Quando l'Utente generico seleziona un nuovo dizionario dati, il Controller modifica nel Modello il valore della variabile che tiene traccia del dizionario dati corrente.

4.2.4 UploadWidget

La classe `UploadWidget` permette di mostrare nell'interfaccia grafica il pulsante di selezione di un file dal *file system*$_G$ del proprio sistema e il pulsante per confermare il caricamento del file scelto. Ciò permette di selezionare e caricare nel server un nuovo dizionario dati. Questo widget viene creato e visualizzato grazie al metodo `create()`, il quale chiama due metodi della libreria Streamlit per creare rispettivamente il *file uploader* e il bottone. Quando il Tecnico preme il bottone per caricare il dizionario inserito, l'`UploadWidget` invoca un metodo del suo Controller che effettuerà alcune verifiche sul file che si vuole caricare. In particolare, il Controller considera il file valido se e solo se rispetta le seguenti condizioni: il file non deve essere nullo, deve avere formato *JSON*$_G$, non deve essere già presente all'interno del server, devono essere già caricati al massimo altri tre file. In caso il file che si vuole caricare passi tutte le verifiche da parte del Controller, quest'ultimo chiamerà il metodo `upload_dictionary(uploaded_file_name: str,`

`uploaded_file_content: dictionary): str` del modello per effettuare l'upload effettivo del file. In base all'esito dell'operazione, il Controller invoca uno dei due metodi `positive_upload_outcome(uploaded_file_name: str)` e `negative_upload_outcome(dictionary_upload_error: str)` dell'`UploadWidget`, usati rispettivamente per notificare il Tecnico del successo o del fallimento di essa.

4.2.5 DeleteWidget

Questa classe consente di visualizzare la componente dell'interfaccia grafica che permette di eliminare il dizionario dati attualmente selezionato. Tale widget è composto da un `SelectWidget` , utile a selezionare il dizionario dati che si vuole eliminare tra quelli caricati nel server, e un pulsante che viene usato per effettuare in modo effettivo la cancellazione. Per creare e visualizzare nell'interfaccia questo widget viene usato il metodo `create()`, il quale al suo interno costruisce il `SelectWidget`, chiamando il suo metodo `create()`, e costruisce il bottone tramite un metodo della libreria `Streamlit`.

Quando l'utente preme il pulsante di eliminazione, il `DeleteWidget` chiama il metodo `operation_delete(delete_file_name: str)` del suo Controller; esso invocherà il metodo del Modello per eliminare in modo effettivo il file, e in base all'esito dell'operazione di eliminazione esso andrà a chiamare un metodo tra i seguenti due presenti all'interno della classe `DeleteWidget`, `positive_delete_outcome(delete_file_name: str)` e `negative_delete_outcome(delete_file_name)` , usati rispettivamente per mostrare a schermo un messaggio di successo o di fallimento dell'operazione appena eseguita.

4.2.6 ChatWidget

È la classe che rappresenta il widget relativo alla *chat*. Esso viene creato usando il suo metodo `create()`: per prima cosa esso visualizza a schermo tutti i messaggi che l'Utente generico e il `chatbot` si sono scambiati, chiamando i metodi `chat_message(name: str)` e `code(body: SupportsStr, language: str, line_numbers: bool)` per ogni messaggio salvato nella variabile di sessione `session_state.chat`. A questo punto, viene visualizzato un `input_box` per la *chat* su cui è possibile inserire l'interrogazione in linguaggio naturale e un

bottone per inviare tale interrogazione. Tale `input_box` sarà disabilitato qualora nel server non sia presente nessun dizionario dati.

Alla pressione del tasto "Invio", oppure dell'analogo tasto nell'interfaccia grafica, il `ChatWidget` chiama il metodo `operation_generate_response(user_input: str)` del suo Controller, il quale, dopo aver sanificato l'input inserito dall'Utente generico (ovvero dopo aver sostituito tutte le occorrenze degli apostrofi con degli spazi), andrà a recuperare dal Modello lo stato di autenticazione e in base ad esso deciderà se far generare al Modello il testo del prompt (nel caso il Cliente sia non autenticato) oppure il testo con le informazioni di debug (in caso il Tecnico sia autenticato).

Tali risposte vengono generate dal Modello grazie ai suoi metodi `generate_prompt(user_input: str, sanitized_user_input: str, current_dictionary: str)` e `generate_debug(user_input: str, sanitized_user_input: str, current_dictionary: str)`. Dopo aver generato la risposta, il Controller la recupera dal Modello e chiama il metodo `show_response(gen_response: str)` dell'oggetto di tipo `ChatWidget`, il quale salva il prompt appena generato nella variabile di sessione `session_state.chat` e lo mostra a schermo grazie ad una chiamata a `code(body: SupportsStr, language: str, line_numbers: bool)`.

4.3 Controller

Il Controller funge da intermediario tra la Vista e il Modello, garantendo una separazione chiara delle responsabilità e facilitando la gestione delle interazioni utente e dei dati. Esso riceve le richieste dell'utente (come ad esempio la pressione di un pulsante), le interpreta e decide quali azioni intraprendere in risposta. Il Controller, per esempio, può recuperare delle informazioni dal Modello oppure modificarne lo stato e di conseguenza aggiornare la vista. Nel nostro progetto, ciascuna classe della Vista (quindi le classi widget) hanno un relativo Controller, che gestisce appunto la comunicazione tra di esse e le classi del Modello. Le classi del Controller sono le seguenti:

4.3.1 AuthenticationController

È la classe che gestisce la comunicazione tra le classi `LoginWidget` e `AuthenticationService` per la funzionalità di *login*. Utilizza il metodo `operation_login(username: str, password: str)` per verificare la validità delle credenziali fornite e per effettuare il login, aggiornando la Vista in modo opportuno.

4.3.2 SelectionController

È il Controller che gestisce la comunicazione tra le classi `SelectionService` e `SelectWidget`, realizzando la funzionalità di selezione di un dizionario dati tra quelli attualmente presenti nel server. Esso utilizza i metodi:

- `operation_get_all_dictionaries()` per recuperare dal Modello tutti i file caricati;
- `operation_get_current_dictionary()` per recuperare dal Modello il dizionario dati a cui sta attualmente facendo riferimento;
- `operation_set_current_dictionary(dictionary: str)` per impostare nel Modello un nuovo dizionario dati a cui fare riferimento;

4.3.3 UploadController

Si occupa della gestione del flusso di comunicazione tra le classi `UploadWidget` e `UploadService` .
Attraverso il metodo `operation_update_file_data()`, questo Controller verifica la validità del dizionario dati che il Tecnico vuole caricare nel server. In caso il file sia valido, esso viene caricato dal Modello usando il metodo `upload_dictionary(uploaded_file_name, uploaded_file_content): str`. Successivamente, in base all'esito dell'operazione, la vista viene aggiornata mostrando un opportuno messaggio di feedback al Tecnico.

4.3.4 DeleteController

`DeleteController` permette di gestire la comunicazione tra le classi `DeleteWidget` e `DeleteService` per realizzare la funzionalità di eliminazione di un dizionario dati. Tramite il suo metodo `operation_delete(delete_file_name: str)`, esso tenta l'eliminazione del

file dal Modello: se l'operazione è andata a buon fine aggiornerà la vista mostrando un messaggio di feedback positivo, altrimenti mostrerà un messaggio di errore.

4.3.5 LogoutController

Funge da intermediario tra le classi `LogoutWidget` e `AuthenticationService`. Attraverso il metodo `operation_logout()`, questa classe Controller imposta nel modello lo stato di *login* dell'utente a `False` per poi aggiornare la vista mostrando un messaggio di avvenuto *logout*.

4.3.6 ChatController

Questo Controller coordina la comunicazione tra le classi `ChatWidget` e `ChatService` realizzando così la funzionalità di chat con il *chatbot*. Il metodo `operation_generate_response(user_input: str)` viene usato per generare una risposta all'interrogazione in linguaggio naturale inserita dall'Utente generico. In particolare, l'input dell'Utente generico viene sanificato rimuovendo tutti gli apostrofi, in quanto possono causare problemi durante la generazione del prompt, poi il Controller recupera lo stato di autenticazione dal Modello per decidere se far generare a quest'ultimo un semplice prompt (nel caso il Cliente non abbia svolto il login) oppure una risposta con le informazioni di *debug* (nel caso il Tecnico è autenticato). In ogni caso, la risposta generata viene poi recuperata dal Modello e viene mostrata a schermo aggiornando la Vista.

5 Stato dei requisiti funzionali

5.1 Requisiti funzionali

Codice	Rilevanza	Descrizione	Stato
ROF 1	Obbligatorio	Il Cliente deve fare l'accesso per accedere alla sezione tecnico	Soddisfatto
ROF 1.1	Obbligatorio	Per fare l'accesso alla sezione tecnico bisogna inserire l'username	Soddisfatto
ROF 1.2	Obbligatorio	Per fare l'accesso alla sezione tecnico bisogna inserire la password	Soddisfatto
ROF 2	Obbligatorio	Il sistema deve mostrare un messaggio di errore in caso di credenziali sbagliate	Soddisfatto
ROF 3	Obbligatorio	Il Tecnico deve poter uscire dalla sezione ad esso dedicata	Soddisfatto
ROF 4	Obbligatorio	Il Tecnico deve poter caricare un *dizionario dati*$_G$, ma solo una volta autenticato come tale	Soddisfatto
ROF 5	Obbligatorio	Deve essere presente un sistema di archiviazione del file di descrizione del *database*$_G$	Soddisfatto
ROF 6	Obbligatorio	Il sistema deve comunicare l'esito del caricamento del dizionario dati	Soddisfatto
ROF 7	Obbligatorio	Il sistema deve mostrare un messaggio di errore esplicativo nel caso in cui il caricamento del dizionario fallisca	Soddisfatto
ROF 8	Obbligatorio	È possibile caricare molteplici file contenenti la descrizione di diversi database	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
ROF 9	Obbligatorio	Deve essere possibile per il Tecnico cancellare i dizionari dati caricati nel sistema	Soddisfatto
ROF 10	Obbligatorio	L'Utente generico può selezionare il dizionario dati sul quale vuole porre la propria interrogazione dalla lista di quelli caricati	Soddisfatto
ROF 11	Obbligatorio	L'Utente generico può visualizzare la lista dei dizionari attualmente salvati nel sistema	Soddisfatto
ROF 11.1	Obbligatorio	L'Utente generico può visualizzare un dizionario nella lista di quelli attualmente salvati nel sistema	Soddisfatto
ROF 11.1.1	Obbligatorio	L'Utente generico può visualizzare il nome del dizionario nella lista di quelli attualmente presenti nel sistema	Soddisfatto
ROF 12	Obbligatorio	Deve esserci una maschera con il campo di testo dove l'Utente generico inserisce la sua interrogazione in linguaggio naturale	Soddisfatto
RDF 13	Desiderabile	L'Utente generico non deve poter inserire l'interrogazione se non è presente alcun dizionario dati	Soddisfatto
ROF 14	Obbligatorio	Nella sezione tecnico deve essere possibile eseguire il *debugging*$_G$ del file contenente la descrizione del database inserendo interrogazioni in linguaggio naturale	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
ROF 15	Obbligatorio	il Cliente deve ottenere un $prompt_{G}$ contenente le informazioni e i comandi necessari affinché, una volta fornito a ChatGPT o ad un altro LLM, questo possa generare la $query_{G}$ SQL_{G} equivalente all'interrogazione inserita	Soddisfatto
ROF 16	Obbligatorio	Il sistema deve visualizzare un messaggio di errore se la ricerca semantica non trova similarità nel dizionario dati	Soddisfatto
ROF 17	Obbligatorio	Il Tecnico deve poter visualizzare tutti i messaggi di debug generati finora come lista di messaggi in una chat, ognuno contenente informazioni utili riguardo le tabelle e i suoi campi presenti all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto
ROF 17.1	Obbligatorio	Il Tecnico deve poter visualizzare un messaggio di debugging contenente informazioni utili riguardo le tabelle e i suoi campi presenti all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto
ROF 17.1.1	Obbligatorio	Il Tecnico deve poter visualizzare informazioni utili riguardo le tabelle e i suoi campi presenti all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
ROF 17.1.1.1	Obbligatorio	Il Tecnico deve poter visualizzare informazioni utili riguardo una singola tabella e i suoi campi presente all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto
ROF 17.1.1.1.1	Obbligatorio	Il Tecnico deve poter visualizzare il nome delle tabelle presenti all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto
ROF 17.1.1.1.2	Obbligatorio	Il Tecnico deve poter visualizzare la lista dei campi delle tabelle presenti all'interno del dizionario selezionato al termine del processo di debugging	Soddisfatto
ROF 17.1.1.1.2.1	Obbligatorio	Il Tecnico deve poter visualizzare le informazioni riguardo un singolo campo delle tabelle tra i risultati del processo di debugging	Soddisfatto
ROF 17.1.1.1.2.1.1	Obbligatorio	Il Tecnico deve poter visualizzare il nome dei campi per ogni tabella nei risultati del processo di debugging	Soddisfatto
ROF 17.1.1.1.2.1.2	Obbligatorio	Il Tecnico deve poter visualizzare la descrizione dei campi per ogni tabella nei risultati del processo di debugging	Soddisfatto
ROF 17.1.1.1.2.1.3	Obbligatorio	Il Tecnico deve poter visualizzare il punteggio di similarità dei campi per ogni tabella nei risultati del processo di debugging	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
ROF 18	Obbligatorio	Il Cliente deve poter visualizzare i prompt generati all'interno di una casella apposita, organizzati in una lista di messaggi, ognuno dentro un messaggio apposito	Soddisfatto
ROF 18.1	Obbligatorio	Il Cliente deve poter visualizzare il prompt generato all'interno di una casella apposita, dentro un messaggio apposito	Soddisfatto
ROF 18.1.1	Obbligatorio	Il prompt deve contenere una lista di tutte le tabelle inerenti, visualizzando per ognuna delle informazioni utili per la generazione della $query_G$ SQL_G	Soddisfatto
ROF 18.1.1.1	Obbligatorio	Il prompt deve contenere singole tabelle inerenti all'interno di una lista, visualizzando per ognuna delle informazioni utili per la generazione della query SQL	Soddisfatto
ROF 18.1.1.1.1	Obbligatorio	Il prompt deve contenere il nome di ognuna delle tabelle inerenti	Soddisfatto
ROF 18.1.1.1.2	Obbligatorio	Il prompt deve contenere la lista dei campi per ognuna delle tabelle inerenti	Soddisfatto
ROF 18.1.1.1.2.1	Obbligatorio	Il prompt deve contenere i singoli campi all'interno di una lista per ognuna delle tabelle inerenti, mostrando informazioni utili per la generazione della query SQL	Soddisfatto
ROF 18.1.1.1.2.1.1	Obbligatorio	Il prompt deve contenere il nome di ognuno dei campi delle tabelle inerenti	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
ROF 18.1.1.1.2.1.2	Obbligatorio	Il prompt deve contenere la descrizione di ognuno dei campi delle tabelle inerenti	Soddisfatto
ROF 18.1.2	Obbligatorio	Il prompt deve contenere una lista delle descrizioni delle relazioni tra le tabelle inerenti utili per la generazione della query SQL	Soddisfatto
ROF 18.1.2.1	Obbligatorio	Il prompt deve contenere all'interno di un lista le singole descrizioni delle relazioni tra le tabelle inerenti	Soddisfatto
ROF 18.1.2.1.1	Obbligatorio	Il prompt deve contenere per ogni relazione il nome del campo con relazione esterna	Soddisfatto
ROF 18.1.2.1.2	Obbligatorio	Il prompt deve contenere per ogni relazione il nome del campo referenziato	Soddisfatto
ROF 18.1.3	Obbligatorio	Il prompt deve contenere l'interrogazione inserita dal Cliente per generarlo	Soddisfatto
ROF 19	Obbligatorio	Il Cliente deve poter copiare il testo del prompt generato	Soddisfatto
ROF 20	Obbligatorio	L'applicazione è in grado di supportare interrogazioni in inglese	Soddisfatto
RDF 21	Desiderabile	L'applicazione è in grado di supportare interrogazioni in italiano	Soddisfatto
RDF 22	Desiderabile	L'applicazione è in grado di supportare interrogazioni in cinese	Soddisfatto
RDF 23	Desiderabile	L'applicazione è in grado di supportare interrogazioni in rumeno	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
RDF 24	Desiderabile	L'applicazione è in grado di supportare interrogazioni in russo	Soddisfatto
ROF 25	Obbligatorio	Il *dizionario dati*$_G$ caricato deve avere la descrizione di tutte le parti del *database*$_G$: descrizione delle tabelle, dei loro campi e delle relazioni tra le tabelle	Soddisfatto
ROF 26	Obbligatorio	Deve esserci un sistema di filtraggio durante la ricerca di similarità nel dizionario per contenere la dimensione del *prompt*$_G$	Soddisfatto
ROF 27	Obbligatorio	Il sistema mostra un messaggio di errore specifico in caso viene caricato un file non in formato JSON	Soddisfatto
RDF 28	Desiderabile	Il sistema mostra un messaggio di errore specifico in caso viene caricato un file formato JSON di dimensioni troppo grandi	Non soddisfatto
ROF 29	Obbligatorio	Il sistema mostra un messaggio di errore specifico in caso viene caricato un file formato JSON che non rispetta lo schema imposto	Soddisfatto
RDF 30	Desiderabile	Il sistema mostra un messaggio di errore specifico in caso viene raggiunto il limite di dizionari salvabili sull'applicativo	Soddisfatto
RDF 31	Desiderabile	Il sistema mostra un messaggio di errore specifico in caso viene aggiunto un dizionario già salvato a sistema	Soddisfatto

Codice	Rilevanza	Descrizione	Stato
RDF 32	Desiderabile	Il sistema elimina i messaggi di debug una volta fatto il logout dall'area tecnico	Soddisfatto
RDF 33	Desiderabile	Il sistema elimina i messaggi di prompt una volta fatto il login con successo nell'area tecnico	Soddisfatto

Tabella 7: Requisiti funzionali

5.2 Requisiti soddisfatti

Tutti i 50 Requisiti Funzionali Obbligatori sono stati soddisfatti. Dei Requisiti Funzionali Desiderabili sono stati soddisfatti 10 su 11, in particolare l'*RDF 28* non è stato completato poiché l'implementazione di questa funzionalità in *Stremlit*$_{\boldsymbol{G}}$ ha presentato diverse problematiche.