Specifica Tecnica

6Coders
6Coders.unipd@gmail.com

11 Marzo 2024

Anno Accademico
2023/2024

Corso di Ingegneria del Software

Registro delle Modifiche - Changelog

Ver.	Data	Autore	Verificatore	Descrizione
1.0	14/05/2024		Lovato Yuri	Approvazione
0.9	12/05/2024	Lovato Yuri Marchiorato Pietro	Chelhaoui Osama	Sviluppata sezione "Design Pattern" Completata sezione "Back-end"
0.8	10/05/2024	Florian Edoardo	Bilinski Eleonora	Stesura sezione "Back-end"
0.7	08/05/2024	Lovato Yuri	Marchiorato Pietro	Completata sezione "Front-end"
0.6	07/05/2024	Bilinski Eleonora	Marchiorato Pietro	Sezione "Requisiti funzionali soddisfatti"
0.5	02/05/2024	Lovato Yuri	Marchiorato Pietro	Sviluppata sottosezione "Manager"
0.4	20/04/2024	Chelhaoui Osama	Marchiorato Pietro	Impostata la struttura dell'architettura
0.3	13/03/2024	Chelhaoui Osama	Marchiorato Pietro	Sviluppata sezione Tecnologie
0.2	12/03/2024	Chelhaoui Osama	Bilinski Eleonora	Sviluppata sezione Introduzione
0.1	11/03/2024	Lovato Yuri	Bilinski Eleonora	Creata la struttura iniziale del documento

Tabella 1: Versionamento documento

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

L'obiettivo del documento "Specifica Tecnica" consiste nell'illustrare in maniera esaustiva l'architettura del prodotto, delineando le tecnologie utilizzate e i requisiti necessari per l'utilizzo della *Web Application*$_G$. La struttura del prodotto viene presentata attraverso diagrammi delle classi, inoltre vengono approfonditamente spiegati e giustificati i *design pattern*$_G$ adottati.

1.2 Scopo del prodotto

Vista la notevole diffusione, specie negli ultimi anni dell'Intelligenza Artificiale, e l'abilità che i *Large Language Model*$_G$ hanno dimostrato nel generare codice in diversi linguaggi di programmazione, il *Proponente*$_G$, Zucchetti S.p.A., con il *Capitolato*$_G$ C9 - ChatSQL, richiede di sviluppare un'applicazione che consenta, a partire dalla struttura dati di un *database*$_G$ (*dizionario dati*$_G$), la generazione di un prompt in risposta a interrogazioni formulate in *linguaggio naturale*$_G$. Tale *prompt*$_G$ sarà quindi sottoposto a un *LLM*$_G$ (Large Language Model) come *ChatGPT*$_G$, il quale produrrà la *query*$_G$ in linguaggio *SQL*$_G$ corrispondente alla domanda posta dall'utente. Questa applicazione, sviluppata come *web application*$_G$, sarà accessibile da tutti i principale *browser*$_G$ (Chrome, Firefox, Edge e Safari) e permetterà agli utenti di richiedere, previo caricamento di un file *JSON*$_G$ che descriva il database in questione, la produzione di un prompt per poter poi far generare la query SQL necessaria ad effettuare l'interrogazione sulla base di dati, il tutto a partire da una richiesta inserita da un utente in linguaggio naturale.

1.3 Glossario

Il documento "Glossario V2.0" è parte della documentazione di progetto dove è possibile reperire definizioni chiare e precise dei vocaboli non comuni utilizzati nei documenti prodotti. La presenza di un determinato vocabolo in "Glossario" viene segnata con la lettera G a pedice (es. *Vocabolo*$_G$).

1.4 Riferimenti

1.4.1 Riferimenti Normativi

- Norme di Progetto V2.0;
- Capitolato [C9] - ChatSQL (Zucchetti SpA).

1.4.2 Riferimenti Informativi

- Slide T8 - Corso di Ingegneria del Software - Progettazione software;
- Slide P4 - Corso di Ingegneria del Software - Diagrammi delle classi;

- Slide P1 - Corso di Ingegneria del Software - Gestione delle dipendenze;
- Slide P9 - Corso di Ingegneria del Software - Pattern Model-View;
- Slide P12 - Corso di Ingegneria del Software - Programmazione SOLID.

2 Tecnologie

Di seguito vengono riportate le tecnologie utilizzate per lo sviluppo del prodotto, suddivise per categorie:

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

2.1 Linguaggi

Tecnologia	Versione	Descrizione
HTML	5	*Linguaggio di markup*$_G$ creato per definire gli elementi dell'interfaccia
CSS	3	Linguaggio di stile utilizzato per definire l'aspetto e la formattazione di documenti $HTML_G$ e XML_G
JavaScript	ES6	Linguaggio di programmazione ampiamente utilizzato per aggiungere interattività e dinamicità alle pagine web
Python	3.11	Linguaggio di programmazione ad alto livello utilizzato in una vasta gamma di applicazioni, dall'automazione agli sviluppi web

Tabella 2: Linguaggi utilizzati

2.2 Framework

Tecnologia	Versione	Descrizione
Vue.js	3.4.21	*Framework*$_G$ *JavaScript*$_G$ progressivo per la costruzione di interfacce utente dinamiche e reattive
Flask	3.0.2	Framework web basato su *Python*$_G$ per la creazione di applicazioni web leggere e scalabili

Tabella 3: Framework utilizzati

2.3 Librerie

Tecnologia	Versione	Descrizione
Axios	1.6.7	Libreria JavaScript per effettuare richieste $HTTP_G$ sia lato $client_G$ che lato $server_G$ in modo semplice ed efficiente
Transformers	4.38.2	Libreria Python per l'apprendimento automatico che offre modelli preaddestrati per il trattamento del $linguaggio\ naturale_G$ (NLP_G) e l'elaborazione del testo
PyTorch	2.2.1	Framework per l'apprendimento automatico e il calcolo scientifico che facilita lo sviluppo e l'implementazione di modelli di intelligenza artificiale
Bootstrap	5.3.3	Framework $frontend_G$ per la progettazione di siti e applicazioni web
Pinia	2.1.7	Libreria di store che facilita la condivisione dello stato tra i componenti Vue
OS	0.1.1	Libreria Python per l'interazione con il sistema operativo, ad esempio per la gestione dei file e delle $directory_G$
Pickle	4.0.1	Libreria Python per la serializzazione e deserializzazione di oggetti Python
JSON	2.0.9	Libreria Python per la codifica e decodifica di oggetti Python in formato $JSON_G$
Wekzueg	2.0.1	Libreria Python per la gestione delle richieste HTTP
Shutil	1.0.1	Libreria Python per la manipolazione di file e directory
NumPy	1.21.5	Libreria Python per il calcolo scientifico che fornisce supporto per array e matrici multidimensionali
Scikit Learn	1.4.2	Libreria Python per il calcolo della cosine similarity tra i vettori che rappresentano gli embedding delle frasi in linguaggio naturale.

Tabella 4: Librerie utilizzate

2.4 Strumenti

Tecnologia	Versione	Descrizione
Git	2.44.0	Sistema di controllo delle versioni distribuito ampiamente utilizzato per gestire il codice sorgente e la collaborazione nello sviluppo software
NPM	10.4.0	Gestore di pacchetti ampiamente utilizzato per installare, gestire e distribuire librerie e moduli JavaScript
PIP	24.0	Sistema di gestione dei pacchetti utilizzato per installare e gestire librerie e moduli Python

Tabella 5: Strumenti utilizzati

2.5 Analisi Statica

Tecnologia	Versione	Descrizione
ESLint	8.57.0	Strumento di analisi statica per JavaScript che aiuta a individuare e correggere errori nel codice, applicare uno stile di codifica coerente e identificare potenziali problemi di sicurezza
Pylint	3.1.0	Strumento di analisi statica per Python che identifica errori, violazioni dello stile di codifica e altre problematiche nel codice sorgente Python

Tabella 6: Strumenti di analisi statica utilizzati

2.6 Analisi Dinamica

Tecnologia	Versione	Descrizione
Github Actions	-	Funzionalità integrata di $GitHub_G$ che consente di automatizzare il flusso di lavoro dello sviluppo software, inclusi $test_G$, $compilazioni_G$, $rilasci_G$ e altro ancora, direttamente dal $repository_G$ GitHub
Vue Test Utils	1.3.6	Libreria ufficiale di $Vue.js_G$ che fornisce utilità per scrivere $test\ unitari_G$ e di integrazione per componenti Vue.js
Jest	27.5.1	Framework di $testing_G$ per JavaScript che semplifica la scrittura di test unitari
Unittest	3.9.0	Framework di $testing_G$ per Python che fornisce un insieme di strumenti per scrivere e gestire test unitari in Python
Cypress	9.6.1	Framework di $testing_G$ end-to-end per applicazioni web che consente di scrivere e eseguire test end-to-end per le applicazioni web

Tabella 7: Strumenti di analisi dinamica utilizzati

3 Architettura

Il prodotto si basa su un'architettura di tipo $Client - Server_G$. Il $Client_G$ è rappresentato dal $browser_G$ dell'utente, mentre il $Server_G$ è un'applicazione dedicata alla gestione delle richieste degli utenti e all'elaborazione delle informazioni, fornendo quindi le risposte ai Client. Per una migliore organizzazione e suddivisione delle responsabilità, il sistema è stato logicamente separato in due componenti principali:

- **Front-end**: Questo è stato sviluppato utilizzando il framework Vue.js. Il frontend è la parte dell'applicazione con cui gli utenti interagiscono direttamente attraverso il loro browser. Gestisce l'interfaccia utente e comunica con il backend per inviare richieste e ricevere dati;
- **Back-end**: Questo è stato sviluppato utilizzando il framework Flask. Il backend gestisce la logica di business dell'applicazione, elaborando le richieste ricevute dai client, accedendo ai dati necessari e generando le risposte appropriate. Si occupa anche della sicurezza e dell'autenticazione degli utenti, se necessario.

La comunicazione tra il frontend e il backend avviene tramite chiamate di tipo $REST\ API_G$ (Interfacce di Programmazione delle Applicazioni basate su REST). Le REST API consentono al frontend di inviare richieste al backend, consentendo una comunicazione efficiente e scalabile tra i due componenti dell'applicazione.

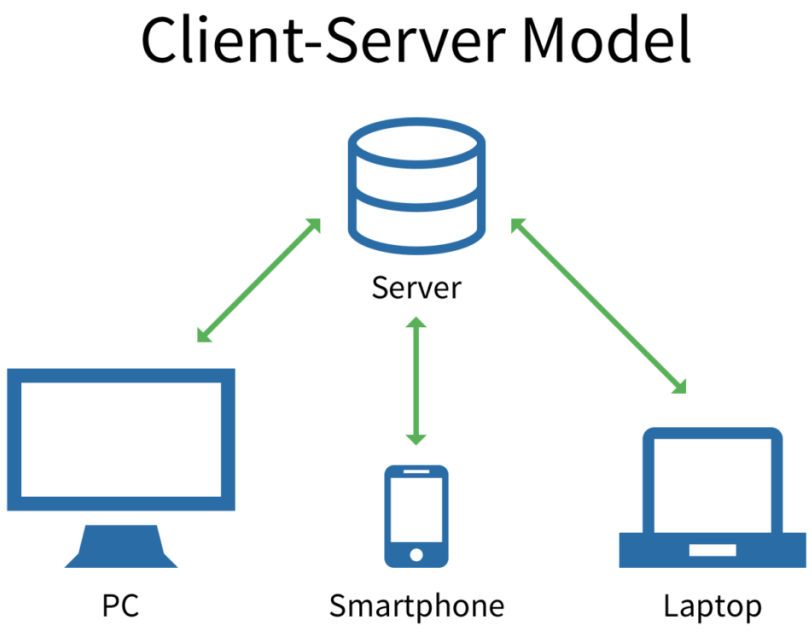

Figura 1: Architettura Client-Server

3.1 Front-end

3.1.1 Architettura del Front-end

Il frontend è responsabile della presentazione dell'interfaccia utente e dell'interazione con l'applicazione. A tale scopo è stato utilizzato il framework Vue.js. Durante la progettazione e lo sviluppo, abbiamo adottato il pattern architetturale Model-View-ViewModel ($MVVM_G$), implementato di default da Vue.js.

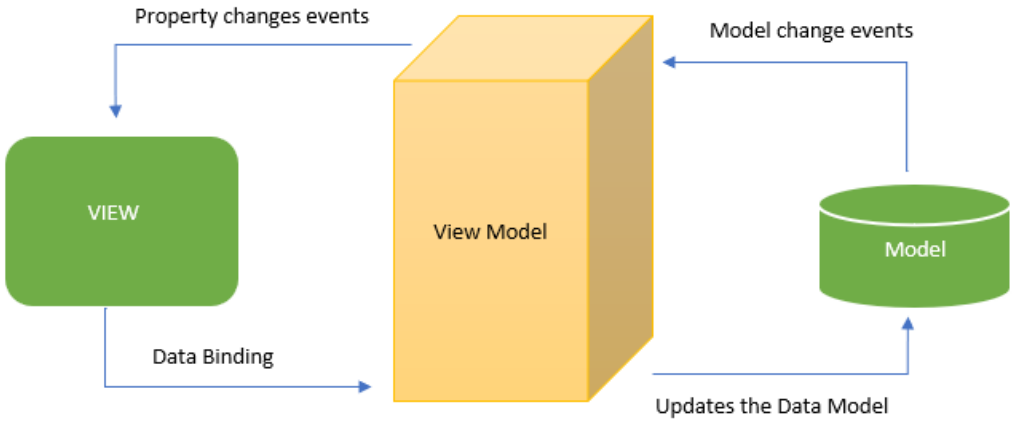

Figura 2: Model-View-ViewModel (MVVM)

- **Model**: Nel contesto di Vue.js, il modello rappresenta i dati e lo stato dell'applicazione. Questo è il cuore dell'applicazione e contiene le informazioni che verranno visualizzate e manipolate dall'utente. Il modello è spesso rappresentato da oggetti JavaScript o dati ottenuti da una fonte esterna, come un Server;
- **View**: In Vue.js la vista è la parte dell'interfaccia utente (UI_G) che viene visualizzata dall'utente. La vista è rappresentata tramite il *markup* $HTML_G$ e utilizza le direttive di Vue per collegare dinamicamente i dati del modello alla vista. La vista reagisce ai cambiamenti nel modello e riflette automaticamente le modifiche all'interfaccia utente;
- **ViewModel**: Nel contesto di Vue.js il ViewModel funge da intermediario tra il modello e la vista. Si occupa di gestire la logica di presentazione e di fornire metodi e funzionalità necessarie per la manipolazione dei dati. Il ViewModel è rappresentato da componenti Vue che contengono la logica di presentazione e le funzioni che operano sui dati del modello. Il ViewModel si assicura che la vista sia sempre sincronizzata con lo stato del modello, mantenendo al contempo una separazione chiara tra logica di visualizzazione e dati.

Di rilevante importanza per lo sviluppo del prodotto, sono state:

- **Composition API**: L'utilizzo della *Composition API*$_G$ di Vue.js per organizzare la logica all'interno dei componenti Vue in modo flessibile e riutilizzabile. Questo approccio consente di estrarre parti della logica del componente in funzioni riutilizzabili, mantenendo il codice pulito e comprensibile, e di evitare la duplicazione della logica;
- **Reactive function**: La funzione reactive è utilizzata per creare un oggetto reattivo, questo permette a Vue.js di tracciare le dipendenze tra i dati reattivi e i componenti che li utilizzano, potendo aggiornare automaticamente i componenti quando cambiano i dati reattivi. La funzione ref è utilizzata per creare un singolo valore reattivo ed è utile quando si ha bisogno di un singolo valore reattivo al di fuori di un oggetto;
- **Bootstrap**: Le pagine sono stilate utilizzando il framework Bootstrap, che offre una vasta gamma di componenti e stili predefiniti per la progettazione di interfacce utente responsive e moderne.

3.1.2 Diagrammi delle classi del Front-end

Nella pagina seguente viene rappresentato il diagramma delle classi per la struttura della parte frontend in sintassi UML_G. Di seguito le sezioni successive andranno a illustrare sezioni singole della *web application*$_G$, suddivise per le seguenti pagine:

- Home;
- Request;
- Manager.

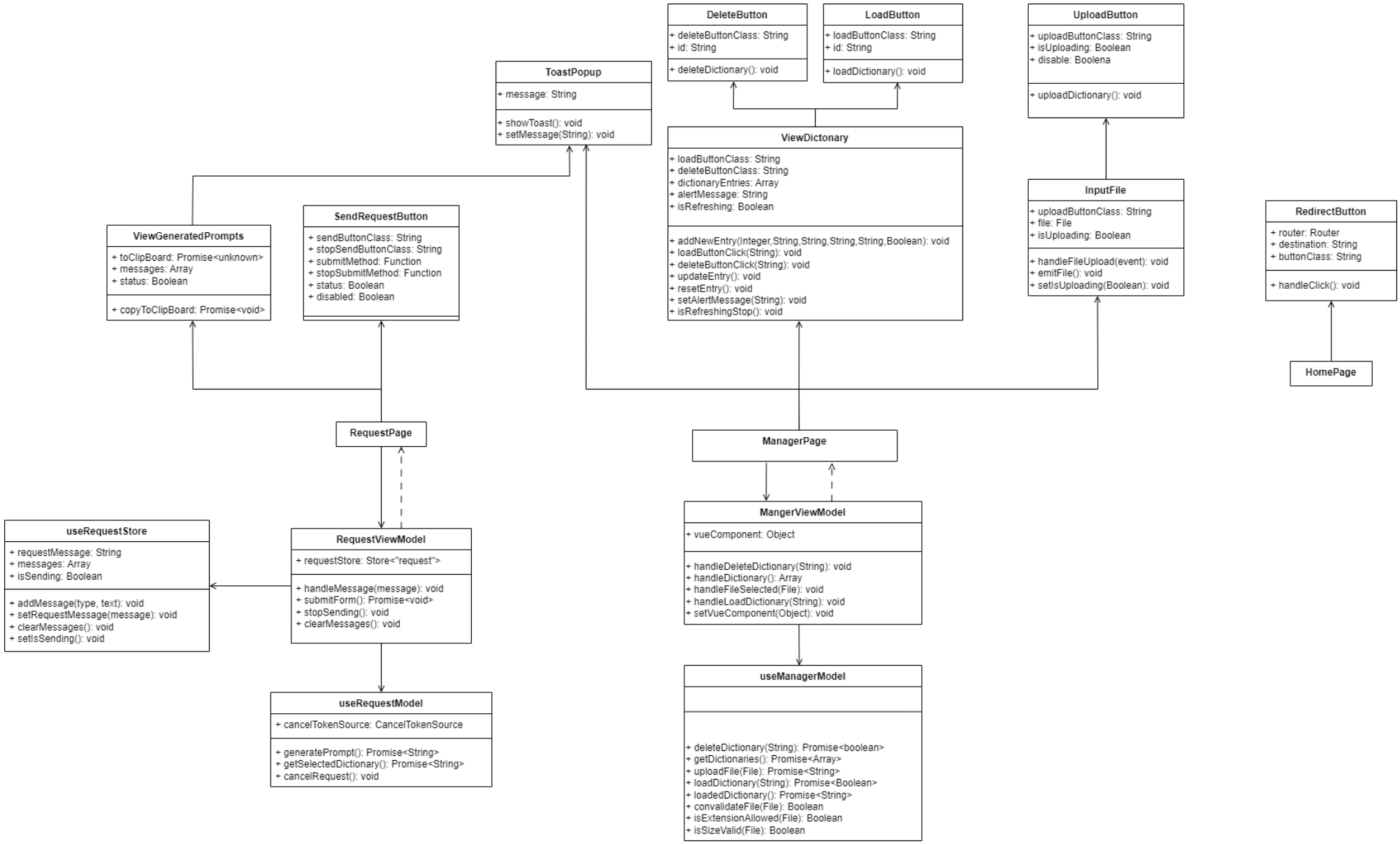

Figura 3: Diagramma delle classi del frontend

3.1.3 Home

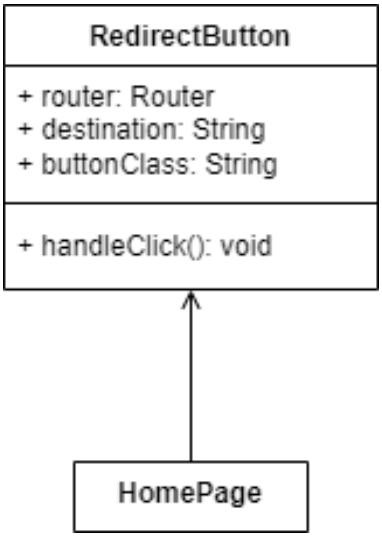

Figura 4: Diagramma delle classi di HomePage

La **HomePage** è la prima pagina ad essere visualizzata quando si accede alla web application, essa non espone nessuna funzionalità se non quella di anticipare la funzione delle altre due pagine della web application. HomePage monta un componente principale:

- **RedirectButton**: Questo bottone permette il reindirizzamento dell'utente verso una $destination_G$ definita che nel contesto della nostra applicazione è **ManagerPage** e **RequestPage**.

3.1.4 Request

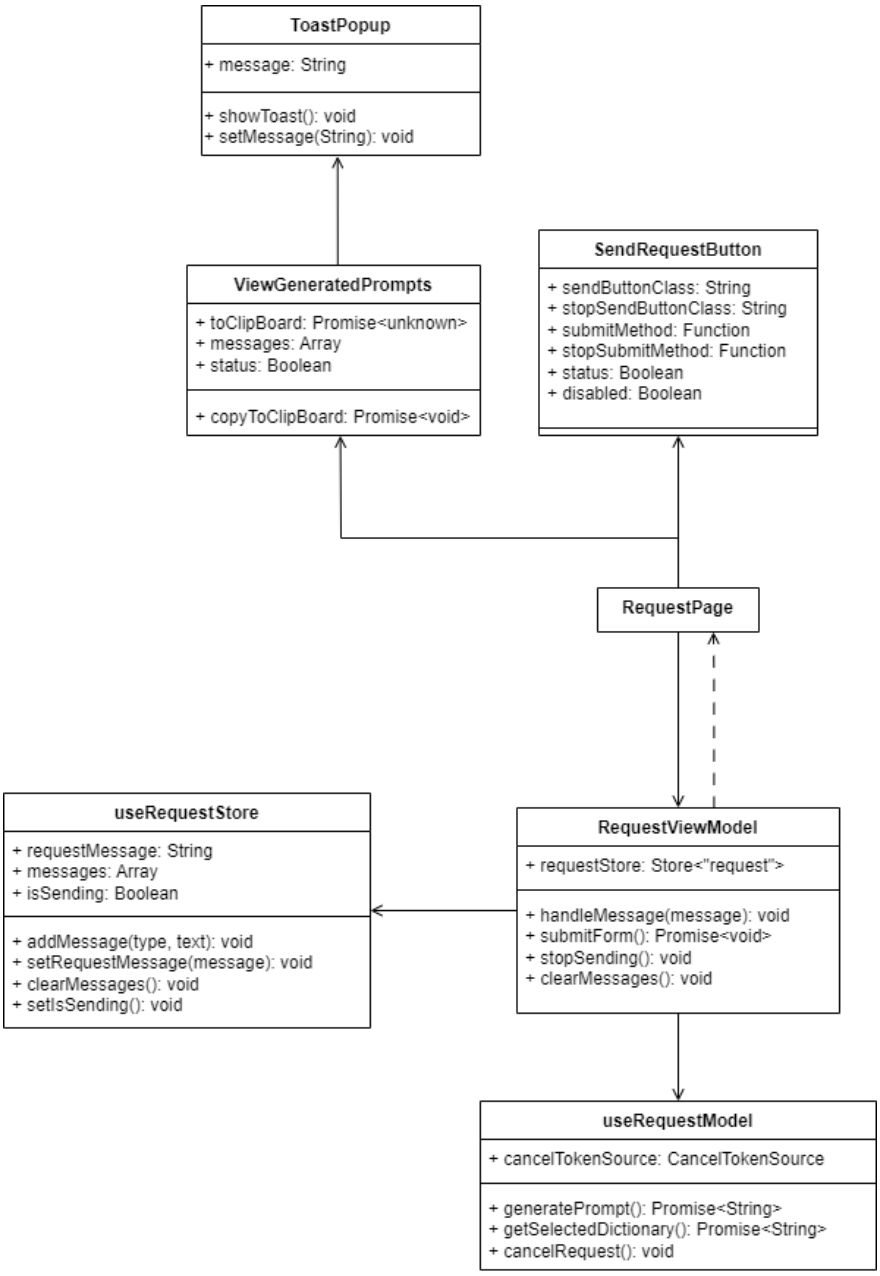

Figura 5: Diagramma delle classi di RequestPage

La **RequestPage** rappresenta il fulcro dell'applicazione: essa permette, a partire da una frase scritta in linguaggio naturale, di ottenere il prompt corrispondente secondo un dizionario dati caricato.

3.1.4.1 View

RequestPage monta i seguenti componenti:

- **SendRequestButton**: Permette alla frase scritta in linguaggio naturale di essere emessa per la generazione del prompt corrispondente;
- **ViewgeneratedPrompts**: Si occupa della visualizzazione delle richieste e dei corrispondenti prompt generati dal backend. Esso monta il componente **ToastPopUp** , esso espone un avviso di tipo $PopUp_G$ all'interno della pagina , per notificare all'utente di eventuali errori.

Questa suddivisione è stata realizzata per separare le responsabilità a livello di singoli componenti, al fine di rendere il sistema facilmente testabile e integrabile con altri componenti. Ogni componente ha un compito specifico e ben definito, consentendo una chiara separazione delle funzionalità. Ciò facilita il testing unitario, in quanto ciascun componente può essere testato indipendentemente, isolando le possibili fonti di errori e semplificando il processo di $debug_G$. Inoltre, questa struttura modulare rende più agevole l'integrazione di nuove funzionalità e la manutenzione del sistema nel tempo, in quanto i cambiamenti possono essere apportati in modo mirato a livello di singoli componenti, senza influire sul resto dell'applicazione.

3.1.4.2 ViewModel

Il ViewModel, residente nel modulo **RequestViewModel**, agisce come tramite tra la Vista e il Modello, sfruttando il meccanismo di **useRequestStore**. Quest'ultimo è stato implementato tramite la libreria Vue chiamata $Pinia_G$, che facilita la condivisione dello stato tra vari componenti e pagine.

3.1.4.3 Model

Il Model risiede nel modulo **useRequestModel** e si focalizza principalmente sulla gestione delle interazioni con il backend, sia per l'invio che per la ricezione delle informazioni riguardanti l'inoltro della richiesta e la creazione del prompt. In particolare:

- **generatePrompt()**: Tale metodo permette l'invio della richiesta in linguaggio naturale , la ricezzione e gestione del prompt corrispondente dal backend;
- **getSelectedDictionary()**: Permette di recuperare il nome del dizionario dati attualmente caricato per effettuare le interrogazioni.

La comunicazione con il backend avviene utilizzando la libreria $Axios_G$. Questa libreria semplifica la gestione delle richieste HTTP, consentendo di inviare e ricevere dati in modo asincrono. Tali funzioni restituiscono un oggetto di tipo $Promise_G$ che verrà risolto quando la risposta dal Server è completa, fornendo accesso ai dati restituiti o gestendo eventuali errori che si verificano durante la richiesta. Pertanto, queste funzioni sono di tipo asincrono, il che significa che vengono eseguite in modo non bloccante. Ciò comporta che, anziché attendere il completamento di una richiesta prima di proseguire con l'esecuzione del codice successivo, il programma può continuare ad eseguire altre istruzioni mentre aspetta la risoluzione della richiesta. Quando la richiesta viene completata e la Promise viene risolta, il codice associato

all'operazione asincrona verrà eseguito. Questo modello di programmazione asincrono è particolarmente utile nel mantenere l'interattività dell'interfaccia utente nelle applicazioni web, consentendo al contempo il recupero di dati dal Server senza bloccare l'intera esecuzione del programma.

3.1.5 Manager

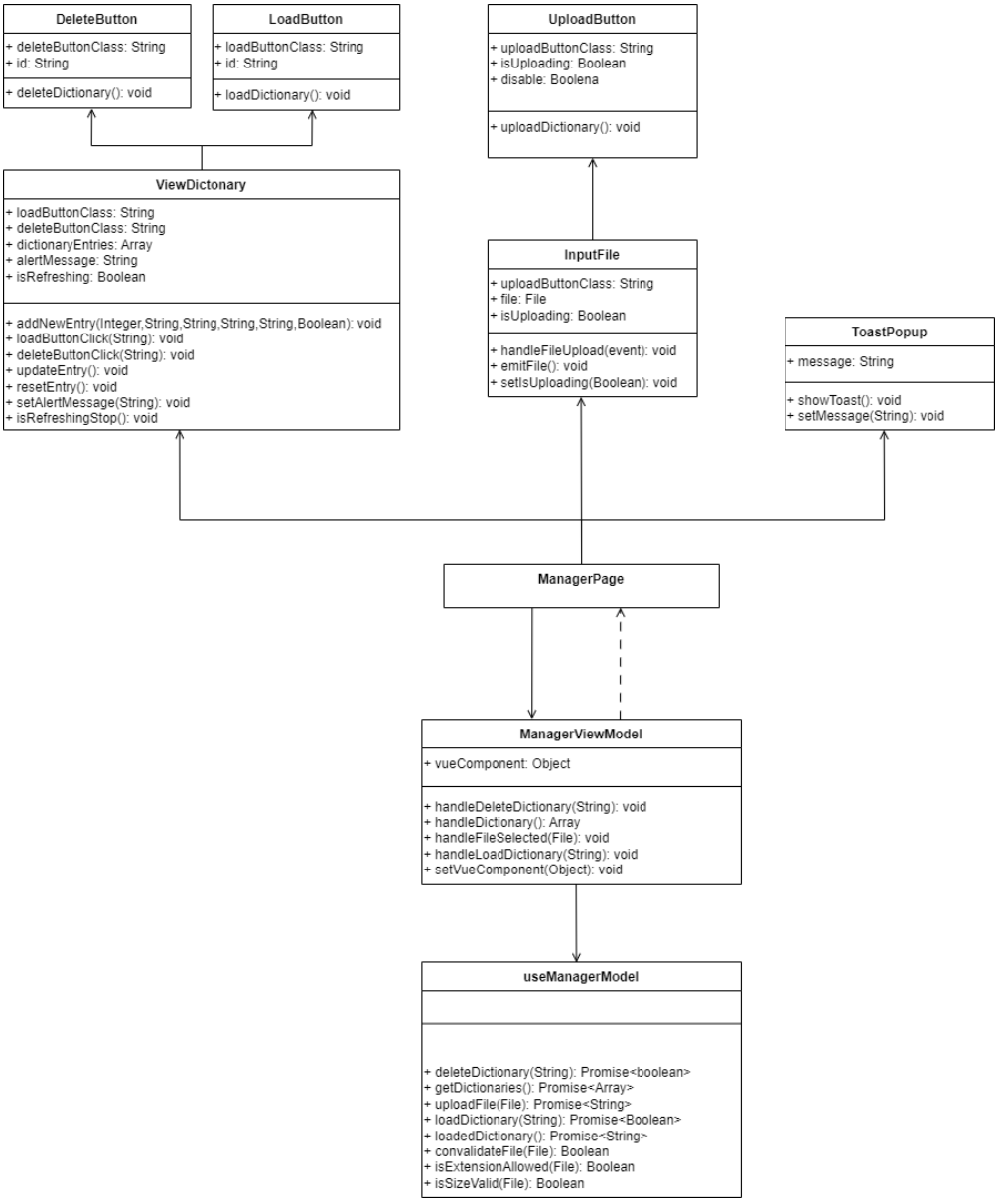

Figura 6: Diagramma delle classi di ManagerPage

ManagerPage è dedicata alla gestione dei dizionari dati e offre una serie di funzionalità che vanno dal caricamento iniziale dei dizionari al loro utilizzo per l'interrogazione dei dati.

3.1.5.1 View

Il nucleo di ManagerPage è costituito da due componenti principali:

- **InputFile**: Questo componente gestisce il processo di caricamento dei nuovi dizionari dati nel sistema. È composto dal componente **UploadButton** che definisce un pul-

sante di caricamento e consente agli utenti di selezionare e caricare i file contenenti i dati del dizionario. Una volta che l'utente ha selezionato il file da caricare, il sistema elabora il file e lo aggiunge al database dei dizionari disponibili;

- **ViewDictionary**: Questo componente offre un'interfaccia per la visualizzazione dei dizionari presenti nel sistema. Oltre alla semplice visualizzazione, consente anche di aggiornare i dati dei dizionari e di eseguire operazioni come il caricamento di un dizionario per l'interrogazione e l'eliminazione di dizionari non più necessari. È composto da due pulsanti principali:
  - **LoadButton**: Questo pulsante consente agli utenti di selezionare un dizionario presente nel sistema per l'interrogazione dei dati. Una volta selezionato, il dizionario viene caricato nell'ambiente di interrogazione;
  - **DeleteButton**: Questo pulsante fornisce agli utenti la possibilità di eliminare un dizionario dal sistema. Quando viene attivato, il dizionario selezionato viene rimosso definitivamente dal sistema.
- **ToastPopUp**: Questo componente permette la visualizzazione di un PopUp all'interno della pagina per informare l'utente di eventuali errori o successi delle operazioni.

3.1.5.2 ViewModel

Il ViewModel riesiede nel modulo **ManagerViewModel**. Questa classe funge da intermediaria tra il Model, definito nella classe **useManagerModel**, e la View, definita in **ManagerPage**. Il suo scopo principale è quello di fornire un'astrazione dei dati e della logica di presentazione dal Model alla View, consentendo una separazione chiara e una migliore gestione delle responsabilità. Il ManagerViewModel incapsula la logica di presentazione e le operazioni sui dati necessarie per soddisfare i requisiti dell'interfaccia utente. Interagisce con il Model per ottenere e aggiornare i dati, quindi esporre questi dati in una forma adatta alla View. Inoltre, gestisce anche la comunicazione tra la View e il Model, traducendo gli input dell'utente in azioni sul Model e aggiornando la View con le modifiche apportate ai dati.

3.1.5.3 Model

Il Model ha la sua sede nel modulo **useManagerModel**. La sua responsabilità principale è di gestire l'elaborazione delle informazioni provenienti dalla Vista, includendo la validazione e l'aggiornamento. Inoltre, il Modello è incaricato di inviare le richieste al backend e di ricevere le relative risposte. In particolare:

- **deleteDictionary()**: Comunica con il backend per eliminare un dizionario dati presente a sistema;
- **getDictionary()**: Richiede la lista dei dizionari dati presenti a sistema;
- **uploadFile()**: Invia al backend un nuovo dizionario dati da inserire a sistema;

- **loadDicitionary()**: Richiede al backend di selezionare un determinato dizionario dati da utilizzare per la generazione del prompt;
- **loadedDictionary()**: Richiede al backend il nome del dizionario caricato attualmente.

Tali funzioni restituiscono un oggetto di tipo Promise che verrà risolto quando la risposta dal Server è completa, fornendo accesso ai dati restituiti o gestendo eventuali errori che si verificano durante la richiesta. Pertanto, queste funzioni sono di tipo asincrono, il che significa che vengono eseguite in modo non bloccante. Ciò comporta che, anzichè attendere il completamento di una richiesta prima di proseguire con l'esecuzione del codice successivo, il programma può continuare ad eseguire altre istruzioni mentre aspetta la risoluzione della richiesta.

Il dizionario dati da inviare al backend subisce una convalida che ne verifica la correttezza e la coerenza, in particolare:

- **convalidateFile()**: Si occupa della validazione del file da caricare ed utilizza:
  - **isExtensionAllowed()**: Verifica che l'estensione del file sia corretta;
  - **isSizeValid()**: Verifica che la dimensione del file sia inferiore a quella massima consentita.

3.2 Back-end

3.2.1 Architettura del Back-end

Il backend è responsabile dell'elaborazione delle informazioni provenienti dal frontend, nonché dell'elaborazione e gestione dei dizionari dati, delle richieste e dei prompt. Il backend è stato realizzato utilizzando il framework Flask ed il linguaggio Python.
Il pattern archietturale utilizzato è stato *hexagonal architecture*$_G$, esso divide l'applicazione in tre parti principali:

- **Domain**: Il dominio rappresenta il nucleo dell'applicazione, dove risiede la logica di business. Include le regole e le operazioni che definiscono il comportamento dell'applicazione. Il codice del dominio dovrebbe essere puro e non dovrebbe dipendere da dettagli implementativi o da infrastrutture esterne;
- **Port**: Le porte rappresentano le interfacce attraverso le quali il dominio comunica con l'esterno. Possono essere viste come contratti o interfacce che definiscono come il dominio può essere interrogato o utilizzato. Le porte sono implementate come interfacce o classi astratte nel dominio. L'implementazione concreta delle porte è delegata agli adattatori;
- **Adapter**: Gli adattatori sono componenti che collegano il dominio con le infrastrutture esterne o con i dettagli implementativi. Gli adattatori traducono le richieste e le risposte dal formato del dominio a quello specifico dell'infrastruttura, garantendo che il dominio rimanga isolato e indipendente da dettagli implementativi esterni.

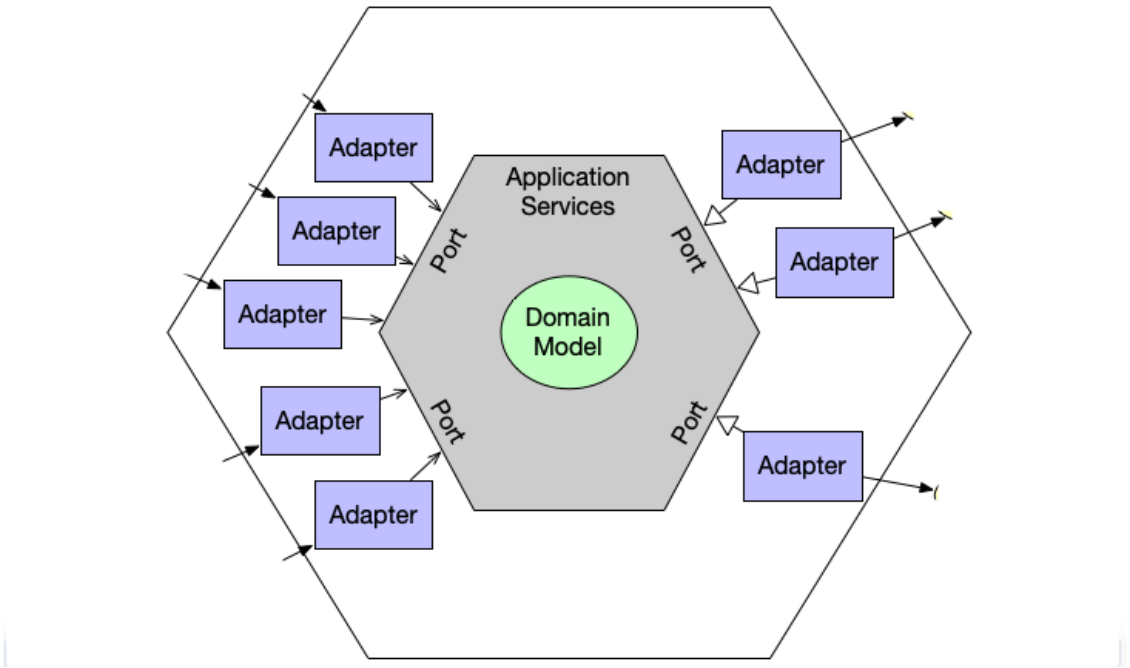

Figura 7: Hexagonal Architecture

La scelta di questo *design pattern*$_G$ è stata guidata da cinque fattori:

- **Separazione delle responsabilità**: L'architettura esagonale promuove una chiara separazione delle responsabilità tra le varie parti dell'applicazione. Il dominio si occupa della logica di business, le porte definiscono le interfacce attraverso le quali il dominio interagisce con l'esterno e gli adattatori collegano il dominio alle infrastrutture esterne. Questa separazione rende l'applicazione più modulare e più facile da comprendere, estendere e mantenere nel tempo;
- **Indipendenza dalla tecnologie esterne**: Poiché il dominio è isolato dalle infrastrutture esterne, l'applicazione diventa meno dipendente da tecnologie specifiche. Questo facilita la sostituzione o l'aggiornamento di parti dell'infrastruttura senza dover modificare il nucleo dell'applicazione;
- **Testing**: L'architettura esagonale rende più semplice il testing dell'applicazione, in quanto il dominio è isolato e non dipende da infrastrutture esterne;
- **Flessibilità**: Con la separazione delle responsabilità e l'indipendenza dalle tecnologie esterne, l'architettura esagonale rende l'applicazione più flessibile e adattabile ai cambiamenti. È più facile aggiungere nuove funzionalità, sostituire parti dell'infrastruttura o adottare nuove tecnologie senza dover riscrivere l'intera applicazione;
- **Scalabilità**: L'architettura esagonale favorisce la scalabilità dell'applicazione, consentendo di distribuire le varie parti su più nodi o di utilizzare diverse tecnologie per gestire carichi di lavoro diversi. Inoltre, la separazione delle responsabilità rende più semplice identificare e ottimizzare le parti critiche dell'applicazione.

3.2.2 Diagrammi delle classi del Back-end

Di seguito viene rappresentato il diagramma delle classi per la struttura della parte backend in sintassi UML:

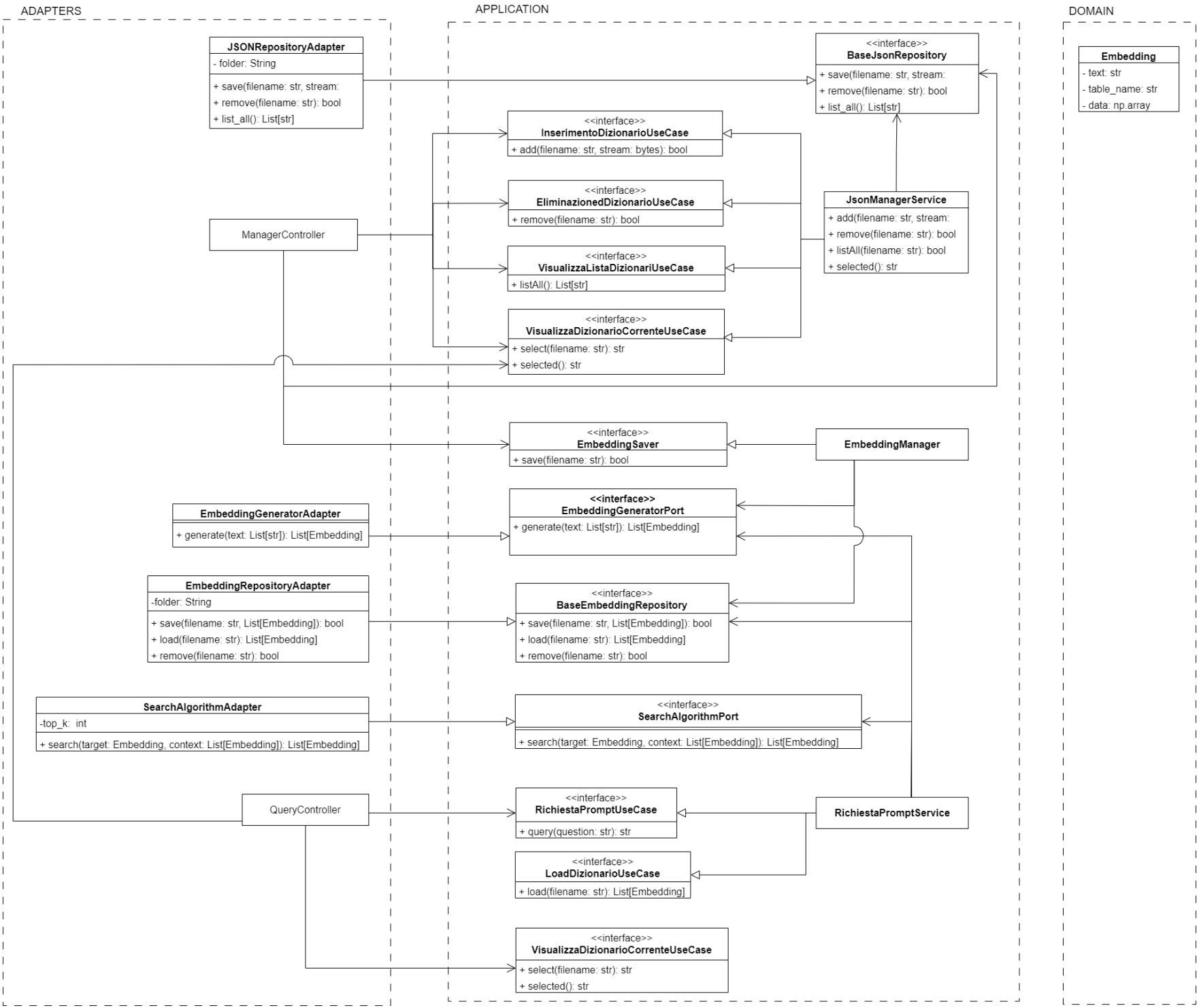

Figura 8: Diagramma delle classi del backend

3.2.3 Dominio

La nostra applicazione nel suo dominio contiene la classe **Embedding**.

3.2.3.1 Embedding

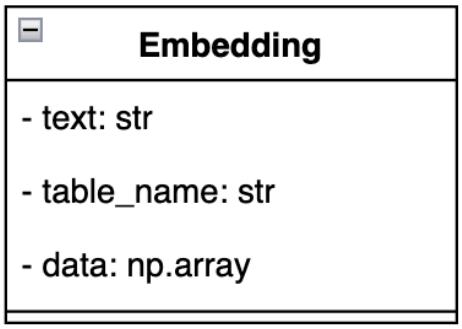

Figura 9: Embedding Class

Gli embedding sono dei vettori di valori creati da modelli di machine learning che rappresentano un oggetto e che quindi possono essere utilizzati per misurare la similitudine fra i vari oggetti. In particolare il campo "data" è un array di $NumPy_G$ ($Numerical\ Python_G$) che constente di effettuare operazioni su grandi quantità do dati in modo efficiente in termini di memoria e prestazioni , esso verrà utilizzato per memorizare gli embedding generati per ogni tabella. La proprietà "text" è una stringa per memorizzare il testo in linguaggio naturale rappresentato dagli embedding, la proprietà "table_name" invece è la stringa contenente il nome della tabella di riferimento.

3.2.4 Application e Adapters

3.2.4.1 ManagerController - Parte JSON

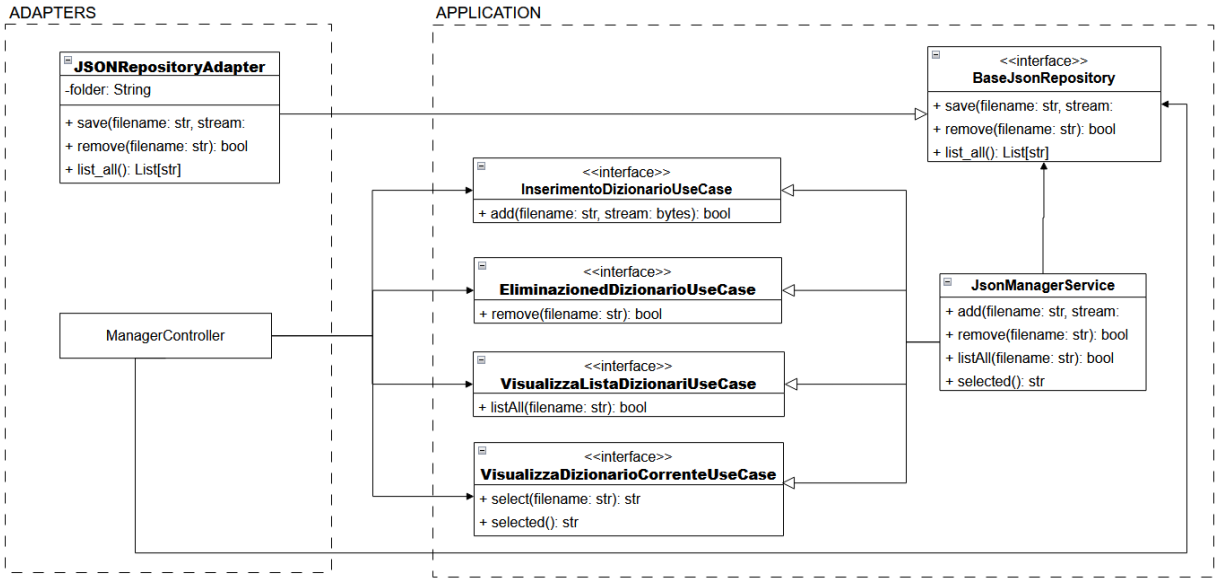

Figura 10: Diagramma classi Manager Controller - JSON

ManagerController si occupa di gestire le richieste provenienti dalla pagina Manager della parte front-end. Per la gestione e salvataggio dei dizionari dati utilizza le seguenti interfacce:

- **InserimentoDizionarioUseCase**: Espone la $signature_G$:
  - **add()**: Salva il dizionario dati caricato sotto forma di file JSON in una cartella apposita.
- **EliminazioneDizionarioUseCase**: Espone la signature:
  - **remove()**: Data una stringa, rimuove il file con il nome corrispondente da un'apposita cartella.
- **VisualizzaListaDizionariUseCase**: Espone la signature
  - **listAll()**: Ritorna la lista di tutti i nomi dei file caricati nell'apposita cartella.
- **VisualizzaDizionarioCorrenteUseCase**: Espone le signature:
  - **select()**: Data una stringa, seleziona il dizionario dati col nome corrispondente e lo imposta come attualmente in uso;
  - **selected()**: Ritorna il nome del dizionario dati attualmente in uso.

JSONManagerService è una classe concreta che implementa tali interfacce.

BaseJSONRepository è un'interfaccia che espone le seguenti signature:

- **add()**: Si occupa di salvare il dizionario dati caricato sotto forma di file JSON in una cartella apposita;
- **remove()**: Si occupa di rimuovere un file JSON rappresentante del dizionario dati dalla cartella apposita;
- **listAll()**: Si occupa di ritornare il nome di tutti i file JSON salvati nella cartella apposita.

questi metodi sono poi implementati dalla classe concreta **JSONRepositoryAdapter**.

3.2.4.2 ManagerController - Parte Embeddings

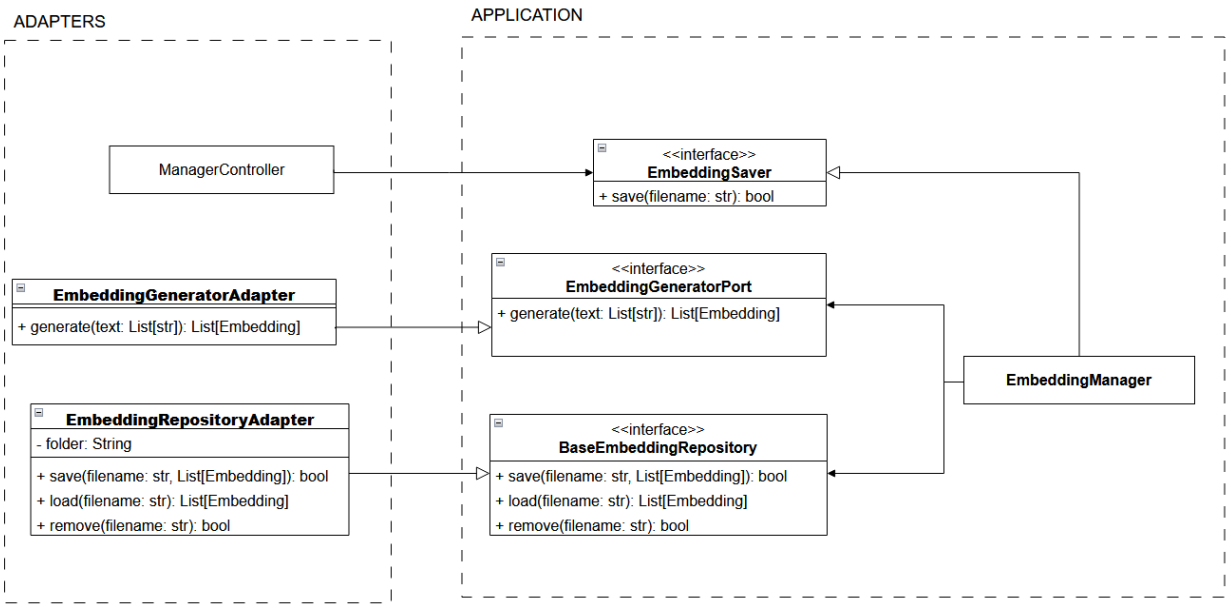

Figura 11: Diagramma classi Manager Controller - Embeddings

ManagerController si occupa anche della generazione e del salvataggio degli embedding, per questo motivo la classe ha una dipendenza con l'intefaccia EmbeddingSaver.

EmbeddingSaver: Espone la signature:

- **save()**: Prende in input il nome di un file JSON contenente un dizionario dati, genera gli embeddings per quel dizionario e li salva in un file in una cartella specifica.

EmbeddingManager implementa l'interfaccia EmbeddingSaver, essa si occupa di generare e salvare gli embedding dei dizionari dati.

EmbeddingGeneratorPort è un'interfaccia che espone la signature:

- **generate()**: Prende in input delle stringhe e restituisce gli embeddings corrispondenti.

EmbeddingGeneratorAdapter implementa l'interfaccia EmbeddingGeneratorPort e si occupa di generare embeddings dati dalle stringhe.

BaseEmbeddingRepository è un'interfaccia che espone le signature:

- **save()**: Accetta il nome di un file e una lista di embeddings come input, poi salva gli embeddings all'interno del file, collocandolo in una cartella specifica;

- **load()**: Prende in input il nome di un file e restituisce gli embeddings salvati all'interno di quel file;
- **remove()**: Prende in input il nome di un file contenente degli embeddings e lo elimina.

EmbeddingRepositoryAdapter implementa l'interfaccia BaseEmbeddingRepository e si occupa di gestire il salvataggio, il caricamento e la rimozione degli embeddings.

3.2.4.3 QueryContoller

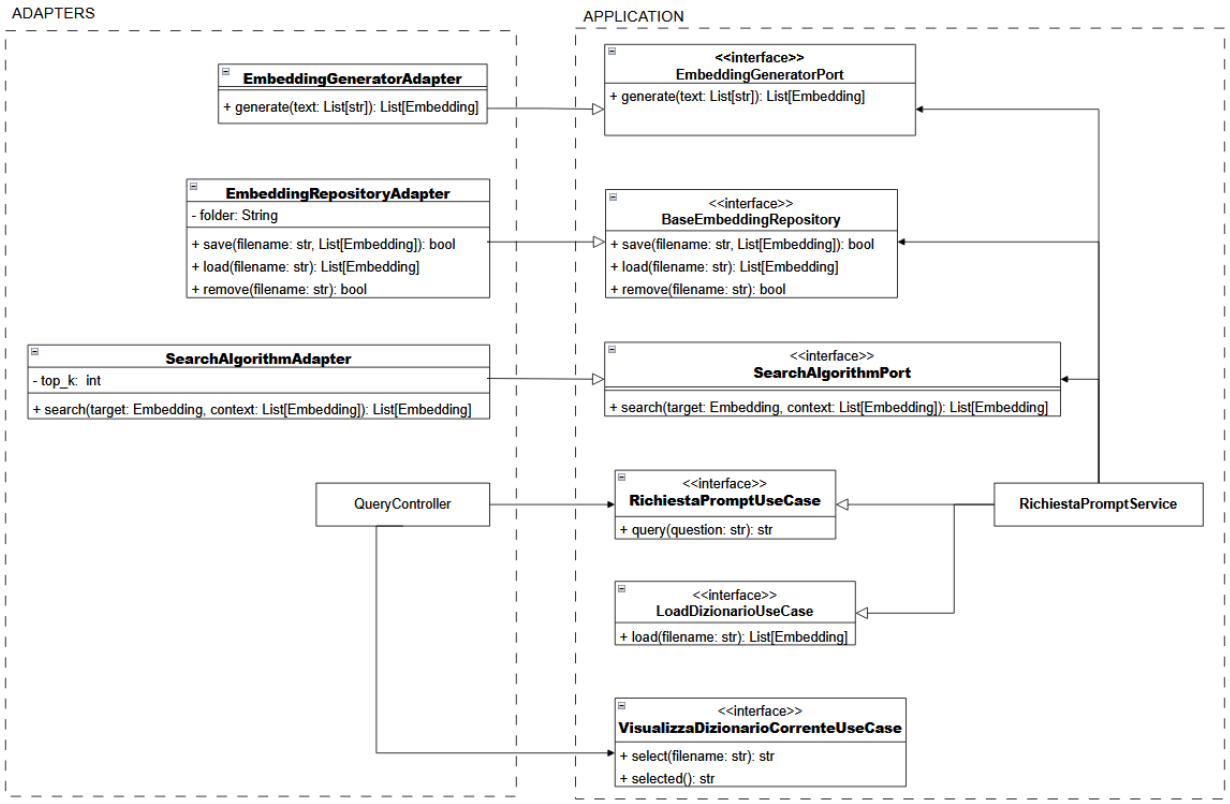

Figura 12: Diagramma classi Query Controller

QueryController si occupa di gestire le richieste provenienti dalla pagina Request della parte front-end. In particolare ritorna il dizionario dati attualmente utilizzato e gestisce della generazione del prompt.

VisualizzaDizionarioCorrenteUseCase è un'interfaccia che che espone la signature:

- **selected()**: Ritorna il dizionario dati attualmente in uso.

LoadDictionaryUseCase è un'interfaccia che espone la signature:

- **load()**: Ritorna gli embeddings contenuti all'interno di un file.

RichiestaPromptUseCase è un'interfaccia che espone la signature:

- **query()**: Prende in input una stringa che contiene una rischiesta in linguaggio naturale e restituisce la stringa prompt corrispondente.

EmbeddingGeneratorPort è un'interfaccia che espone la signature:

- **generate()**: Ritorna degli embeddings date delle stringhe

EmbeddingGeneratorAdapter è una classe concreta che implementa EmbeddingGeneratorPort.

BaseEmbeddingRepository è un'interfaccia che espone le signature:

- **save()**: Accetta il nome di un file e una lista di embeddings come input, poi salva gli embeddings all'interno del file, collocandolo in una cartella specifica;
- **load()**: Prende in input il nome di un file e restituisce gli embeddings salvati all'interno di quel file;
- **remove()**: Prende in input il nome di un file contenente degli embeddings e lo elimina.

EmbeddingRepositoryAdapter implementa l'interfaccia BaseEmbeddingRepository e si occupa di gestire il salvataggio, il caricamento e la rimozione degli embeddings.

SearchAlgorithmPort è un'interfaccia che espone la seguente signature:

- **search()**: Restituisce gli embeddings del dizionario dati attualmente caricato più simili agli embeddings generati dalla richiesta in linguaggio naturale

SearchAlgorithmAdapter è una classe concreta che implementa SearchAlgorithmPort.

RichiestaPromptService implementa le interfacce **LoadDictionaryUseCase** e **RichiestaPromptUseCase** per farlo utilizza:

- **EmbeddingGeneratorAdapter**
- **EmbeddingRepositoryAdapter**
- **SearchAlgorithmAdapter**

3.3 Design Pattern

I desing pattern utilizzati nel progetto sono stati scelti per garantire una struttura solida e flessibile, che permetta di adattarsi facilmente ai cambiamenti e di mantenere , essi sono stati :

- **Strategy**: Utilizzato per implementare l'algoritmo di ricerca degli embeddings più simili;
- **Adapter**: Utilizzato per adattare le interfacce del dominio alle interfacce delle infrastrutture esterne.

Per leggibilità sono stati omessi dal diagramma delle classi nella sezione 3.2.2 i dettagli delle classi che implementano tali design pattern , ma verranno esposti di seguito.

3.3.1 Strategy

Il design pattern $Strategy_G$ è stato utilizzato per implementare l'algoritmo di ricerca degli embeddings più simili. Questo pattern permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. In questo modo, è possibile variare l'algoritmo di ricerca degli embeddings più simili senza dover modificare il codice che lo utilizza.

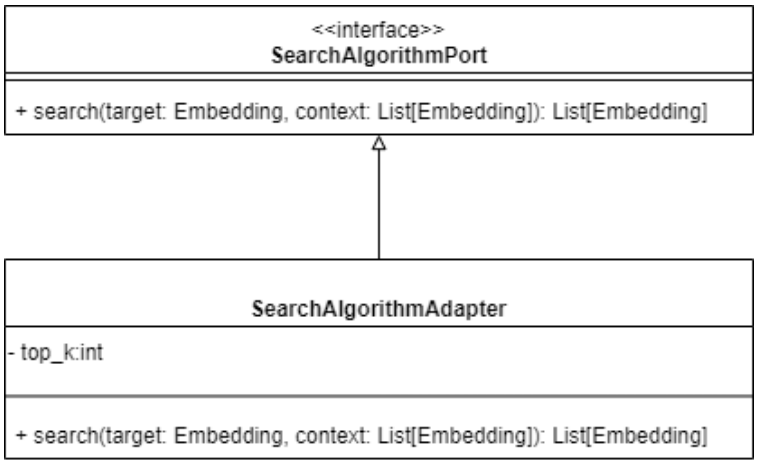

Figura 13: Design Pattern Strategy

In particolare **SearchAlgorithmAdapter** implementa l'intefaccia **SearchAlgorithmPort** e si occupa di implementare l'algoritmo di ricerca degli embeddings più simili tramite knn_G $(k - nearest\ neighbors_G)$. Questo permette di variare l'algoritmo di ricerca senza dover modificare il codice che lo utilizza.

3.3.2 Adapter

Il design pattern $Adapter_G$ è stato utilizzato nella modalità $Object\ Adapter_G$ per adattare le interfacce del dominio alle interfacce delle infrastrutture esterne. Questo pattern permette di convertire l'interfaccia di una classe in un'altra interfaccia che il client si aspetta.

3.3.2.1 EmbeddingRepository Adapter

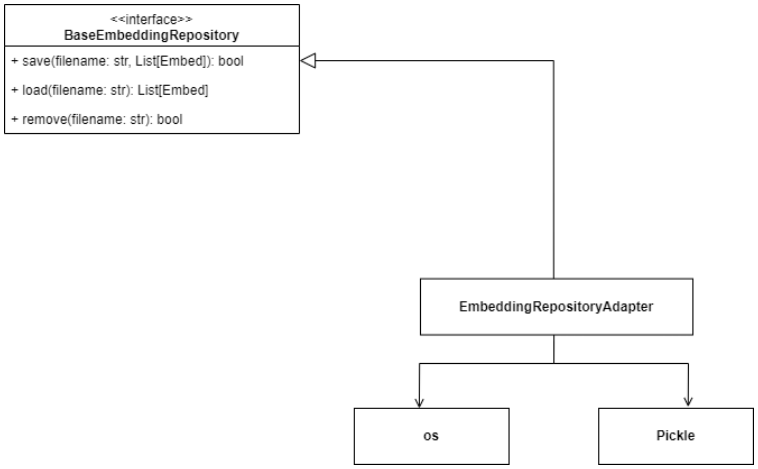

Figura 14: Design Pattern Adapter EmbeddingRepository

EmbeddingRepository implementa l'interfaccia **BaseEmbeddingRepository** ed effettua un $wrap_G$ delle librerie esterne OS_G e $Pickle_G$ per la gestione del salvataggio e recupero degli embedding.

3.3.2.2 EmbeddingGenerator Adapter

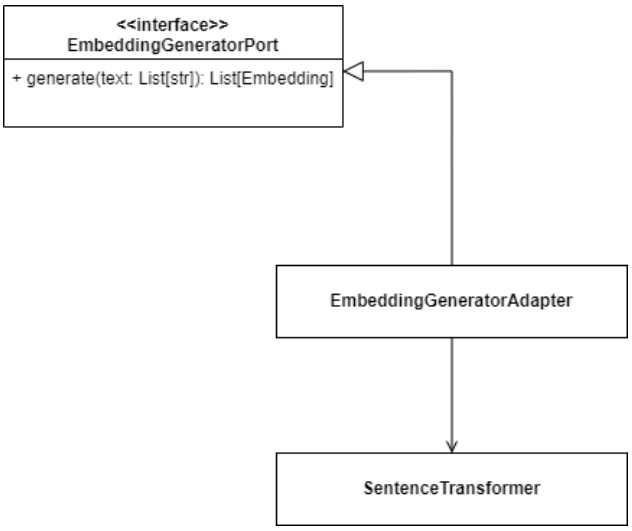

Figura 15: Design Pattern Adapter EmbeddingGenerator

EmbeddingGenerator implementa l'interfaccia **EmbeddingGeneratorPort** ed effettua un wrap della libreria esterna $SentenceTransformer_G$ per la generazione degli embedding.

3.3.2.3 JSONRepository Adapter

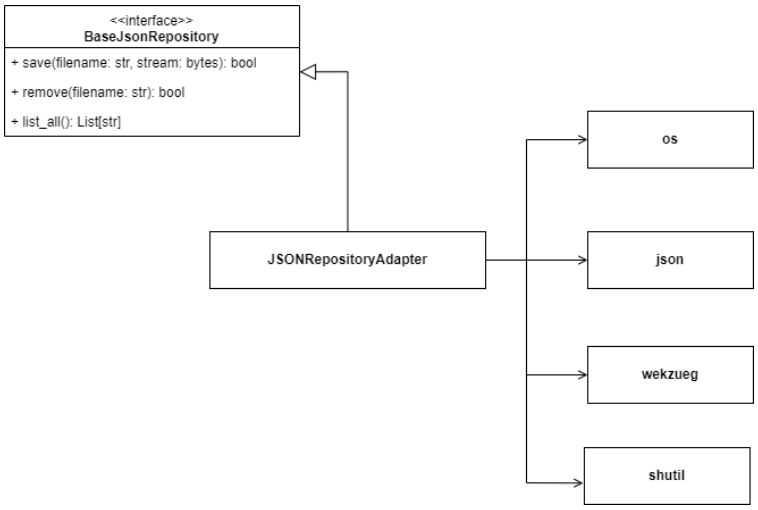

Figura 16: Design Pattern Adapter JSONRepository

JSONRepository implementa l'interfaccia **BaseJSONRepository** ed effettua un wrap delle librerie esterne OS_G , $JSON_G$, $Wekzueg_G$ e $Shutil_G$ per la gestione del salvataggio e recupero dei dizionari dati.

4 Requisiti funzionali soddisfatti

In questa sezione viene esposto il grado di adempimento dei requisiti funzionali indicati nel documento Analisi dei requisiti V2.0.

4.1 Tabella dei requisiti funzionali soddisfatti

Codice	Descrizione	Classificazione	Stato
RF-1	L'utente deve poter inserire un nuovo dizionario dati nel sistema.	Obbligatorio	Soddisfatto
RF-2	L'utente deve poter rimuovere un dizionario dati tra quelli caricati a sistema	Obbligatorio	Soddisfatto
RF-3	L'utente deve poter selezionare e caricare un dizionario dati, tra quelli inseriti a sistema, per effettuare l'interrogazione	Obbligatorio	Soddisfatto
RF-4	L'utente deve visualizzare la lista dei dizionari dati inseriti a sistema	Obbligatorio	Soddisfatto
RF-5	L'utente deve poter visualizzare il nome dei dizionari inseriti a sistema	Obbligatorio	Soddisfatto
RF-6	L'utente deve poter visualizzare l'estensione dei dizionari inseriti a sistema	Obbligatorio	Soddisfatto
RF-7	L'utente deve poter visualizzare la data di inserimento dei dizionari inseriti a sistema	Obbligatorio	Soddisfatto
RF-8	L'utente deve poter visualizzare la dimensione dei dizionari inseriti a sistema	Obbligatorio	Soddisfatto
RF-9	L'utente deve poter visualizzare il dizionario dati caricato	Obbligatorio	Soddisfatto
RF-10	L'utente deve poter visualizzare l'icona check per il dizionario dati caricato	Obbligatorio	Soddisfatto
RF-11	L'utente deve poter visualizzare il nome del dizionario dati caricato in verde	Obbligatorio	Soddisfatto
RF-12	L'utente deve poter inserire la richiesta in linguaggio naturale tramite un'interfaccia dedicata	Obbligatorio	Soddisfatto

Codice	Descrizione	Classificazione	Stato
RF-13	Il sistema deve processare la richiesta dell'utente inserita in linguaggio naturale e restituire un prompt per guidare chatbot alla generazione della corrispondente query SQL	Obbligatorio	Soddisfatto
RF-14	Il prompt generato dal sistema dev'essere contenuto all'interno di una casella di testo selezionabile per permettere all'utente di copiarne il testo	Obbligatorio	Soddisfatto
RF-15	Il sistema deve segnalare un messaggio di errore quando fallisce l'inserimento a sistema di un nuovo dizionario dati	Obbligatorio	Soddisfatto
RF-16	Il sistema deve segnalare un messaggio di errore nell'inserimento del dizionari dati se quest'ultimo non ha estensione JSON	Obbligatorio	Soddisfatto
RF-17	Il sistema deve segnalare un messaggio di errore nell'inserimento del dizionario dati se quest'ultimo contiene errori o non definito correttamente	Obbligatorio	Soddisfatto
RF-18	Il sistema deve segnalare un messaggio di errore nell'inserimento del dizionario dati se il dizionario dati inserito ha dimensione superiore a 500 KB	Obbligatorio	Soddisfatto
RF-19	Il sistema deve segnalare un messaggio di errore nell'inserimento del dizionario dati se il server non è raggiungibile	Obbligatorio	Soddisfatto
RF-20	Il sistema deve visualizzare un messaggio di errore nel caso vi sia un errore nella generazione del prompt	Obbligatorio	Soddisfatto
RF-21	Se non vi è una corrispondenza tra la richiesta in linguaggio naturale ed il dizionario dati caricato, l'utente visualizza un messaggio d'errore	Obbligatorio	Soddisfatto
RF-22	Se l'utente richiede la generazione del prompt corrispondente alla richiesta in linguaggio naturale ma non è possibile raggiungere il server, l'utente visualizza un messaggio d'errore	Obbligatorio	Soddisfatto

Codice	Descrizione	Classificazione	Stato
RF-23	L'utente deve poter visualizzare la lista delle richieste inserite	Obbligatorio	Soddisfatto
RF-24	L'utente deve poter visualizzare la lista dei prompt generati	Obbligatorio	Soddisfatto
RF-25	L'utente deve poter eliminare la lista delle richieste e dei prompt generati	Obbligatorio	Soddisfatto
RF-26	L'utente deve poter copiare il prompt generato	Obbligatorio	Soddisfatto
RF-27	Il sistema deve visualizzare un messaggio di errore nel caso in cui ci sia un errore nell'eliminazione di un dizionario dati	Obbligatorio	Soddisfatto
RF-28	L'utente visualizza un messaggio di errore se prova ad eliminare un dizionario dati non presente a sistema	Obbligatorio	Soddisfatto
RF-29	L'utente visualizza un messaggio di errore se prova ad eliminare un dizionario dati ma il server non è raggiungibile	Obbligatorio	Soddisfatto
RF-30	Il sistema deve visualizzare un messaggio di errore in caso vi sia un errore nel caricamento per l'interrogazione di un dizionario dati	Obbligatorio	Soddisfatto
RF-31	L'utente visualizza un messaggio di errore se prova a caricare un dizionario dati per l'interrogazione non presente a sistema	Obbligatorio	Soddisfatto
RF-32	L'utente visualizza un messaggio di errore se prova a caricare un dizionario dati per l'interrogazione ma il server non è raggiungibile	Obbligatorio	Soddisfatto
RF-33	L'utente deve poter aggiornare la lista dei dizionari dati	Obbligatorio	Soddisfatto
RF-34	L'utente deve poter visualizzare l'ora di aggiornamento della lista dei dizionari dati	Obbligatorio	Soddisfatto
RF-35	L'utente deve poter selezionare un dizionario dati	Obbligatorio	Soddisfatto

Codice	Descrizione	Classificazione	Stato
RF-36	L'utente deve poter inserire una richiesta in linguaggio naturale	Obbligatorio	Soddisfatto
RF-37	L'utente deve poter decidere se vuole gestire l'inserimento, il load o l'eliminazione dei dizionari dati o generare un prompt nel dizionario dati già selezionato	Obbligatorio	Soddisfatto
RF-38	Il sistema deve salvare i dizionario dati ed i rispettivi vettori di embeddings caricati dagli utenti	Obbligatorio	Soddisfatto
RF-39	Il dizionario dati caricato, il quale contiene la struttura del database, deve conetere descrizioni sulle tabelle, sui relativi campi e gli indici del database	Obbligatorio	Soddisfatto
RF-40	L'utente deve essere informato se il dizionario dati da inserire è già presente a sistema	Obbligatorio	Soddisfatto
RF-41	Il sistema deve poter generare un prompt a partire da una richiesta in lingua inglese	Obbligatorio	Soddisfatto
RF-42	L'utente non può inserire la richiesta se non c'è un dizionario dati caricato	Obbligatorio	Soddisfatto
RF-43	La scelta del dizionario dati deve essere semplice e immediata anche dalla pagina di generazione del prompt	Desiderabile	Non Soddisfatto
RF-44	L'utente visualizza, oltre al prompt, anche la frase SQL prodotta da un sistema di AI_G	Opzionale	Non Soddisfatto
RF-45	Il sistema deve assicurarsi che la frase SQL generata dal sistema di AI sia corretta sia dal punto di vista sintattico che semantico	Opzionale	Non Soddisfatto
RF-46	Il prompt generato deve essere utilizzabile con LLM alternativi a chatGPT	Opzionale	Non Soddisfatto
RF-47	Il sistema di generazione del prompt è funzionante anche con richieste inserite in lingua italiana	Opzionale	Non Soddisfatto

Codice	Descrizione	Classificazione	Stato
RF-48	La richiesta in linguaggio naturale deve poter essere inserita anche tramite input vocale	Opzionale	Non Soddisfatto

Tabella 8: Stato dei Requisiti Funzionali

4.2 Grafici dei requisiti funzionali soddisfatti

Sono stati soddisfatti 42 requisiti obbligatori, 0 requisiti desiderabili e 0 requisiti opzionali con una copertura del 100% di requisiti obbligatori ed una copertura di 87,5% di tutti i requisiti funzionali.

Tipologia	**Numero Totale**	**Numero soddisfatti**	**Percentuale**
Obbligatori	42	42	100%
Desiderabili	1	0	0%
Opzionali	5	0	0%
Totale	**48**	**42**	**87,5%**

Tabella 9: Tabella dei requisiti funzionali soddisfatti