
# Rod2Cod

# Specifica Tecnica

Il documento ha lo scopo di definire l'**architettura** e il **design** dell'**applicazione** che si andrà a sviluppare, includendo inoltre eventuali **API** e **tecnologie** utilizzate.

*Progetto di Ingegneria del Software*
*A.A. 2024/2025*

## Informazioni

| | |
|---|---|
| **Versione** | 1.0 |
| **Uso** | Esterno |
| **Data** | 11/04/2025 |
| **Destinatari** | Gruppo Rod2Cod |
| | Zucchetti |
| | Tullio Vardanega |
| | Riccardo Cardin |
| **Responsabile** | Alberto Maggion |
| **Amministratore** | Annalisa Egidi |
| **Verificatori** | Alberto Maggion |
| | Annalisa Egidi |
| | Filippo Bellon |
| | Luca Calzetta |
| | Michele Nesler |
| **Autori** | Alberto Maggion |
| | Annalisa Egidi |
| | Filippo Bellon |
| | Luca Calzetta |
| | Michele Nesler |

# Specifica Tecnica

| Versione | Data | Descrizione | Autore | Verificatore | Validatore |
|---|---|---|---|---|---|
| 1.0 | 2025-04-10 | | | | Michele Nesler |
| 0.17 | 2025-04-10 | Sezione Struttura - Aggiunto tracciamento requisiti su classi front-end | Filippo Bellon | Alberto Maggion | |
| 0.16 | 2025-04-09 | Sezione Struttura - Aggiunta sezione diagrammi delle classi frontend | Luca Calzetta | Alberto Maggion | |
| 0.15 | 2025-04-09 | Sezione Struttura - Aggiunto design pattern MVVM | Filippo Bellon | Alberto Maggion | |
| 0.14 | 2025-04-08 | Sezione Struttura - Aggiunte immagini diagrammi delle classi | Michele Nesler | Annalisa Egidi | |
| 0.13 | 2025-04-08 | Sezione Architettura - Architettura Logica - Aggiunta sezione architettura VVM | Luca Calzetta | Alberto Maggion | |
| 0.12 | 2025-04-08 | Sezione Tecnologie - Aggiornate librerie e linguaggi | Filippo Bellon | Alberto Maggion | |
| 0.11 | 2025-04-07 | Sezione Struttura - Aggiunto tracciamento requisiti su classi back-end | Filippo Bellon | Alberto Maggion | |
| 0.10 | 2025-04-07 | Sezione struttura, aggiunti risultato test e esecuzione test | Alberto Maggion | Filippo Bellon | |
| 0.9 | 2025-04-06 | Sezione API - Aggiunto endpoint Status Test, Sezione | Luca Calzetta | Filippo Bellon | |

| | | | | | |
|---|---|---|---|---|---|
| | | Tecnologie - Aggiunte librerie e framework frontend | | | |
| 0.8 | 2025-04-05 | Sezione Struttura - Aggiunta descrizione diagrammi delle classi *Elemento Domanda*$_G$ | Filippo Bellon | Michele Nesler | |
| 0.7 | 2025-04-03 | Sezione Struttura e Tecnologie - Aggiunto Database e aggiornate tecnologie back-end | Filippo Bellon | Luca Calzetta | |
| 0.6 | 2025-03-31 | Sezione API - Aggiunte API e relative tabelle | Filippo Bellon | Michele Nesler | |
| 0.5 | 2025-03-27 | Sezione Struttura - Aggiunti altri Design Patterns Utilizzati | Filippo Bellon | Luca Calzetta | |
| 0.4 | 2025-03-23 | Sezione Struttura - Aggiunti altri Design Patterns Utilizzati | Filippo Bellon | Luca Calzetta | |
| 0.3 | 2025-03-19 | Sezione Struttura - Aggiunti Design Patterns Utilizzati | Filippo Bellon | Annalisa Egidi | |
| 0.2 | 2025-03-13 | Aggiunta sezione Struttura | Filippo Bellon | Luca Calzetta | |
| 0.1 | 2025-03-05 | Prima stesura | Filippo Bellon | Luca Calzetta | |

# Indice

# 1 Introduzione

## 1.1 Glossario

Questo documento, come anche molti altri all'interno del progetto, viene affiancato dal documento **Glossario v1.0.0** presente all'interno della documentazione, contenente una definizione di tutti i termini specifici utilizzati, identificati da una G a pedice.

## 1.2 Riferimenti

### 1.2.1 Riferimenti normativi

- Way of Working
- Regolamento del progetto didattico: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf

### 1.2.2 Riferimenti informativi

- Capitolato C1: https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C1.pdf
- Analisi dei requisiti
- Verbali Interni
- Verbali Esterni
- Progettazione: Le dipendenze fra le componenti: https://www.math.unipd.it/~rcardin/swea/2022/Dependency%20Management%20in%20Object-Oriented%20Programming.pdf
- Progettazione e programmazione: Diagrammi delle classi (UML): https://www.math.unipd.it/~rcardin/swea/2023/Diagrammi%20delle%20Classi.pdf
- Progettazione: I pattern architetturali: https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf
- Progettazione software: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T06.pdf
- Progettazione: Il pattern Dependency Injection https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Architetturali%20-%20Dependency%20Injection.pdf
- Progettazione: il pattern Model-View-Controller e derivati: https://www.math.unipd.it/~rcardin/sweb/2022/L02.pdf
- Progettazione, i pattern creazionali (GoF): https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf
- Progettazione, i pattern strutturali (GoF): https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf
- Progettazione, i pattern di comportamento (GoF): https://drive.google.com/file/d/1cpi6rORMxFtC91nI6_sPrG1Xn-28z8eI/view?usp=sharing
- Programmazione: SOLID Programming: https://drive.google.com/file/d/1o1Xun2dVVc3mDiaGyN0FrDJhhoO3lfLQ/view?usp=sharing

# 2 Tecnologie

## 2.1 Introduzione

Questa sezione contiene informazioni riguardo le tecnologie utilizzate per la realizzazione dell'applicativo.

## 2.2 Elenco delle tecnologie

## 2.3 Linguaggi

| Nome | Descrizione | Versione |
|---|---|---|
| Python | Linguaggio di programmazione interpretato, ad alto livello e orientato agli oggetti. | >=3.13.0 |
| Javascript | Linguaggio di programmazione interpretato, orientato agli oggetti e basato su eventi. | ES14 |

## 2.4 Framework

| Nome | Descrizione | Versione |
|---|---|---|
| Vue JS | Framework JavaScript open-source per la creazione di interfacce utente e applicazioni web. | 3.5.13 |
| Bootstrap | Framework CSS per gestire la grafica all'interno del frontend | 5.3.3 |
| Flask | Micro framework per Python, progettato per lo sviluppo di applicazioni web. | 3.1.0 |

## 2.5 Librerie

| Nome | Descrizione | Versione |
|---|---|---|
| pytest | Libreria per il testing in Python, che consente di scrivere test in modo semplice e intuitivo. | 8.3.5 |
| pytest-cov | Plugin per pytest che fornisce report di copertura del codice. Utilizzato per i test all'interno dell'editor Visual Studio Code. | 6.0.0 |
| pytest-xdist | Plugin per pytest che consente di eseguire test in parallelo e distribuire i test su più processi o macchine. Utilizzato per i test all'interno dell'editor Visual Studio Code. | 3.6.1 |
| Flask | Micro framework per Python, progettato per lo sviluppo di applicazioni web. | 3.1.0 |
| Flask-SQLAlchemy | Estensione di Flask per l'integrazione con SQLAlchemy. | |
| coverage | Strumento per misurare la copertura del codice in Python. Utilizzato assieme a pytest all'interno delle *Github*$_G$ Actions. | 7.7.0 |
| radon | Strumento per l'analisi della complessità ciclomatica del codice Python. | 4.1.0 |
| xenon | Strumento complementare a radon per l'analisi della complessità ciclomatica, permette al comando di fallire se sotto una certa soglia. | 0.9.3 |
| dependency-injector | Libreria per effettuare l'"iniezione" delle dipendenze in Python. | 4.46.0 |
| psycopg2 | Driver PostgreSQL per Python, utilizzato per la connessione e l'interazione con il database PostgreSQL. | 2.9.10 |
| sacrebleu | Libreria utilizzata per il calcolo delle metriche BLEU, TER e CHRF nello strumento di valutazione delle risposte. | 2.5.0 |
| rouge-score | Libreria utilizzata per il calcolo della metrica ROGUE nello strumento di valutazione delle risposte. | 0.1.0 |

| | | |
|---|---|---|
| bert-score | Libreria utilizzata per il calcolo della metrica BERT nello strumento di valutazione delle risposte. | 0.3.10 |
| sentence-transformers | Libreria utilizzata per il calcolo della similitudine nello strumento di valutazione delle risposte. | 4.0.0 |
| pandas | Libreria per la manipolazione e l'analisi dei dati in Python. | 2.2.3 |
| joblib | Libreria che facilita l'elaborazione parallela, la memorizzazione in cache dei risultati e la distribuzione delle attività in Python. | 1.3.2 |
| flask-Cors | Estensione di Flask per gestire le richieste CORS. | 4.0.0 |
| axios | HTTP client per chiamate API | 1.8.3 |
| vue-router | Libreria di Vue JS per gestire le rotte in un'app *Single Page Application*$_G$ | 4.5.0 |
| vite | Vite è un build tool e dev server per frontend | 6.0.11 |
| vitest | Vitest è un test runner ottimizzato per Vite | 3.0.5 |
| jsdom | Simula il *DOM*$_G$ nel contesto Node.js per testare componenti vue | 26.0.0 |
| @ionic/vue | È il pacchetto ufficiale di Ionic per Vue 3. Utilizzato per inserire delle icone nel frontend. | 8.5.0 |
| configparser | Libreria per la gestione dei file di configurazione di tipo INI in Python. | 7.2.0 |
| waitress | Server WSGI per l'esecuzione di applicazioni Flask. Utilizzato per la fase di deployment. | 3.0.2 |
| hf-xet | Libreria utilizzata per il download e l'upload di modelli da Hugging Face. | 1.0.3 |

### 2.6 Strumenti

| Nome | Descrizione | Versione |
|---|---|---|
| Visual Studio Code | Editor di codice gratuito e open-source sviluppato da Microsoft. Supporta vari linguaggi di programmazione, estensioni personalizzabili e altri strumenti per lo sviluppo. | 1.98.2 |
| Docker | Piattaforma per la creazione, distribuzione e gestione di applicazioni in container. | 24.0.1 |
| PostgreSQL | Database relazionale open source basato su *SQL*$_G$. Utilizzato all'interno di un docker container. | 16.0 |
| *pgAdmin*$_G$ | Interfaccia grafica per la gestione di database PostgreSQL. Utilizzato all'interno di un docker container. | 9.1 |

## 3 API

### 3.1 *Elemento Domanda*$_G$

#### 3.1.1 POST Add *Elemento Domanda*$_G$

**Descrizione:** Questo endpoint consente di aggiungere un nuovo *elemento domanda*$_G$ inviando all'interno del body della richiesta, una domanda e la sua relativa risposta.

- **endpoint:** `/domande`

- **Metodo:** `POST`
- **Body:** formato `application/json` = `{"domanda": "stringa", "risposta": "stringa"}`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **201** | JSON | Date domanda e risposta, l'elemento viene creato con successo. |
| Negativo | **400** | `{"message": "..."}` | Non vengono caricati tutti i parametri richiesti, oppure alcuni presentano un formato non valido. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.1.2 GET Get *Elemento Domanda*$_G$

**Descrizione:** Questo endpoint consente di ottenere un *elemento domanda*$_G$ specifico avente un determinato id.

- **endpoint:** `/domande/{id}`
- **Metodo:** `GET`
- **Parametri:**
  - `id` = id dell'*elemento domanda*$_G$ da ottenere

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | L'elemento viene trovato e restituito con successo. |
| Negativo | **400** | `{"message": "..."}` | L'elemento non esiste oppure ne viene fornito un id in formato non valido. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.1.3 GET Get All Elementi Domanda

**Descrizione:** Questo endpoint consente di ottenere tutti gli elementi domanda presenti nel sistema.

- **endpoint:** `/domande`
- **Metodo:** `GET`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | Tutti gli elementi vengono restituiti con successo. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.1.4 POST Delete Elementi Domanda

**Descrizione:** Questo endpoint consente di eliminare una lista di elementi domanda specificando i rispettivi id.

- **endpoint:** `/domande/delete`
- **Metodo:** `POST`
- **Body:** formato `application/json` = `{"ids": [1, 2, 3]}`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | `{"message": "..."}` | Gli elementi vengono eliminati con successo. |
| Negativo | **400** | `{"message": "..."}` | Uno o più parametri non vengono forniti oppure presentano un formato non valido. |

| | | | |
|---|---|---|---|
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.1.5 POST Update *Elemento Domanda*$_G$

**Descrizione:** Questo endpoint consente di aggiornare un *elemento domanda*$_G$ specificando il suo id e i nuovi valori per domanda e risposta.

- **endpoint:** `/domande/{id}`
- **Metodo:** `PUT`
- **Body:** formato `application/json` = `{"domanda": "stringa", "risposta": "stringa"}`
- **Parametri:**
  - `id` = id dell'*elemento domanda*$_G$ da aggiornare

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | `{"message": "..."}` | L'elemento viene aggiornato con successo. |
| Negativo | **400** | `{"message": "..."}` | Uno o più parametri non vengono forniti oppure presentano un formato non valido. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

## 3.2 Esecuzione Test

#### 3.2.1 POST Execute Test

**Descrizione:** Questo endpoint consente di avviare l'esecuzione di un test sulle domande presenti nel sistema.

- **endpoint:** `/executeTest`
- **Metodo:** `POST`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | `{"message": "Test avviato con successo"}` | L'esecuzione del test viene avviata con successo. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.2.2 GET Status Test

**Descrizione:** Questo endpoint consente di ricavare lo stato del test attualmente in esecuzione.

- **endpoint:** `/executeTest/status`
- **Metodo:** `GET`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | Lo status del test in corso viene restituito con successo. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

## 3.3 Risultato Test

#### 3.3.1 GET Get Risultato Test

**Descrizione:** Questo endpoint consente di ottenere il risultato del test eseguito, dato il suo id.

- **endpoint:** `/risultati/{id}`
- **Metodo:** `GET`
- **Parametri:**

- `id` = id del risultato del test da ottenere

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | Il risultato del test viene trovato e restituito con successo. |
| Negativo | **400** | `{"message": "..."}` | Il risultato del test non esiste oppure ne viene fornito un id in formato non valido. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.3.2 GET Get All Risultati Test

**Descrizione:** Questo endpoint consente di ottenere tutti i risultati dei test eseguiti.

- **endpoint:** `/risultati`
- **Metodo:** `GET`

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | Tutti gli elementi vengono restituiti con successo. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

#### 3.3.3 GET Get Risultato Singola Domanda

**Descrizione:** Questo endpoint consente di ottenere il risultato di una singola domanda di un test, dato il suo id.

- **endpoint:** `/risultati/domande/{id}`
- **Metodo:** `GET`
- **Parametri:**
  - `id` = id del risultato della singola domanda da ottenere

| Esito | Codice HTTP | Body | Descrizione |
|---|---|---|---|
| Positivo | **200** | JSON | Il risultato della singola domanda del test viene trovato e restituito con successo. |
| Negativo | **400** | `{"message": "..."}` | Il risultato della singola domanda del test non esiste oppure ne viene fornito un id in formato non valido. |
| Negativo | **500** | `{"message": "..."}` | Si è verificato un errore interno del server. |

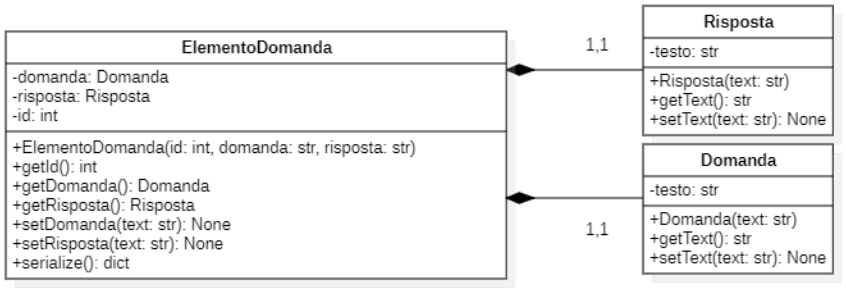
# 4 Struttura

## 4.1 Diagrammi delle classi

### 4.1.1 Backend

#### 4.1.1.1 Domain Model

##### 4.1.1.1.1 *Elemento Domanda*$_G$




I componenti principali del dominio di *elemento domanda*$_G$ sono:

###### 4.1.1.1.1.1 Domanda

- **Attributi**
  - `testo`: stringa che rappresenta il testo della domanda
- **Metodi**
  - `getText()`: restituisce il testo della domanda
  - `setText(text: string)`: imposta il testo della domanda

###### 4.1.1.1.1.2 Risposta

- **Attributi**
  - `testo`: stringa che rappresenta il testo della risposta
- **Metodi**
  - `getText(): str`
    - **Descrizione**:
      - restituisce il testo della risposta
    - **Input**:
      - nessuno
    - **Output**:
      - stringa che rappresenta il testo della risposta
  - `setText(text: str)`
    - **Descrizione**:
      - imposta il testo della risposta
    - **Input**:
      - `text`: stringa che rappresenta il testo della risposta
    - **Output**:
      - nessuno

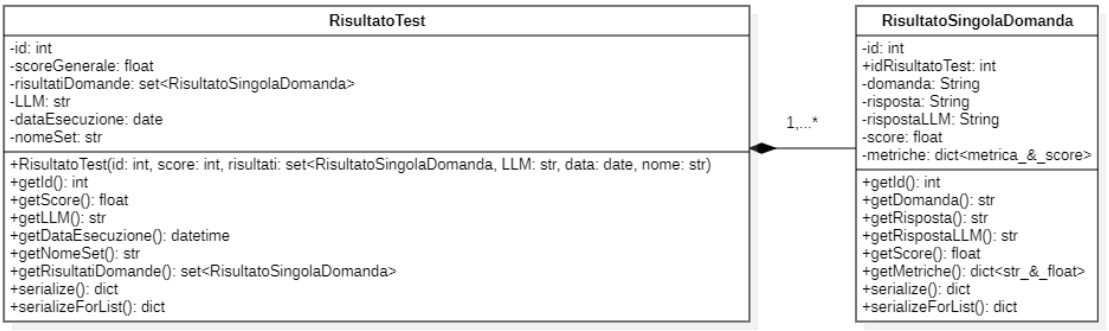
###### 4.1.1.1.1.3 ElementoDomanda

- **Attributi**
  - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
  - `domanda`: oggetto di tipo `Domanda` che rappresenta la domanda

    - `risposta`: oggetto di tipo `Risposta` che rappresenta la risposta

- **Metodi**
    - `getId(): int`
        - **Descrizione**:
            - restituisce l'identificativo univoco dell'*elemento domanda*$_G$
        - **Input**:
            - nessuno
        - **Output**:
            - intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - `getDomanda(): Domanda`
        - **Descrizione**:
            - restituisce l'oggetto `Domanda` associato all'*elemento domanda*$_G$
        - **Input**:
            - nessuno
        - **Output**:
            - oggetto di tipo `Domanda` che rappresenta la domanda
    - `getRisposta(): Risposta`
        - **Descrizione**:
            - restituisce l'oggetto `Risposta` associato all'*elemento domanda*$_G$
        - **Input**:
            - nessuno
        - **Output**:
            - oggetto di tipo `Risposta` che rappresenta la risposta
    - `setDomanda(domanda: str)`
        - **Descrizione**:
            - imposta il testo della domanda
        - **Input**:
            - `domanda`: stringa che rappresenta il testo della domanda
        - **Output**:
            - nessuno
    - `setRisposta(risposta: str)`
        - **Descrizione**:
            - imposta il testo della risposta
        - **Input**:
            - `risposta`: stringa che rappresenta il testo della risposta
        - **Output**:
            - nessuno
    - `serialize(): dict`
        - **Descrizione**:
            - restituisce un `dict` con i dati dell'*elemento domanda*$_G$
        - **Input**:
            - nessuno
        - **Output**:
            - `dict` con i dati dell'*elemento domanda*$_G$

#### 4.1.1.1.2 Risultato Test




Il componente principale del dominio di risultato test è:

##### 4.1.1.1.2.1 RisultatoSingolaDomanda

- **Attributi**
  - `id`: intero che rappresenta l'identificativo univoco del risultato della domanda
  - `domanda`: string che rappresenta la domanda
  - `risposta`: stringa che rappresenta la risposta attesa
  - `rispostaLLM`: stringa che rappresenta la risposta fornita dall'$LLM_G$
  - `score`: float che rappresenta il punteggio di similitudine tra la risposta attesa e la risposta fornita dall'$LLM_G$
  - `metriche`: dizionario contenente le metriche di valutazione della risposta
- **Metodi**
  - `getId(): int`
    - **Descrizione**:
      - restituisce l'identificativo univoco del risultato della domanda
    - **Input**:
      - nessuno
    - **Output**:
      - intero che rappresenta l'identificativo univoco del risultato della domanda
  - `getDomanda(): str`
    - **Descrizione**:
      - restituisce una stringa contenente la domanda posta all'$LLM_G$
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `str` che rappresenta la domanda
  - `getRisposta(): str`
    - **Descrizione**:
      - restituisce una stringa contenente la risposta attesa
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `str` che rappresenta la risposta
  - `getRispostaLLM(): str`
    - **Descrizione**:
      - restituisce una stringa contenente la risposta fornita dall'$LLM_G$
    - **Input**:

- nessuno
- **Output**:
  - oggetto di tipo `str` che rappresenta la risposta fornita dall'$LLM_G$

▸ `getScore(): float`
- **Descrizione**:
  - restituisce un float contenente il punteggio di similitudine tra la risposta attesa e la risposta fornita dall'$LLM_G$
- **Input**:
  - nessuno
- **Output**:
  - oggetto di tipo `float` che rappresenta il punteggio di similitudine tra la risposta attesa e la risposta fornita dall'$LLM_G$

▸ `getMetriche(): dict`
- **Descrizione**:
  - restituisce un dizionario contenente le metriche di valutazione della risposta con il relativo punteggio
- **Input**:
  - nessuno
- **Output**:
  - oggetto di tipo `dict` contenente le metriche di valutazione della risposta e il relativo punteggio

▸ `serialize(): dict`
- **Descrizione**:
  - restituisce un `dict` con tutti i dati del risultato della domanda
- **Input**:
  - nessuno
- **Output**:
  - `dict` con i dati del risultato della domanda

▸ `serializeForList(): dict`
- **Descrizione**:
  - restituisce un `dict` con i dati principali del risultato della domanda, senza le metriche
- **Input**:
  - nessuno
- **Output**:
  - `dict` con i dati del risultato della domanda, senza le metriche

#### 4.1.1.1.2.2 RisultatoTest

- **Attributi**
  - ▸ `id`: intero che rappresenta l'identificativo univoco del risultato del test
  - ▸ `score`: float che rappresenta il punteggio medio del test
  - ▸ $LLM_G$: stringa che rappresenta il nome dell'$LLM_G$ utilizzato
  - ▸ `dataEsecuzione`: oggetto di tipo `datetime` che rappresenta la data di esecuzione del test
  - ▸ `nomeSet`: stringa che rappresenta il nome del set di domande utilizzato
  - ▸ `risultatiDomande`: set di oggetti di tipo `RisultatoSingolaDomanda` che rappresentano i risultati delle domande del test
- **Metodi**
  - ▸ `getId(): int`
    - **Descrizione**:

    - restituisce l'identificativo univoco del risultato del test
  - **Input**:
    - nessuno
  - **Output**:
    - intero che rappresenta l'identificativo univoco del risultato del test
- `getScore(): float`
  - **Descrizione**:
    - restituisce il punteggio medio del test
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `float` che rappresenta il punteggio medio del test
- `getLLM(): str`
  - **Descrizione**:
    - restituisce il nome dell'$LLM_G$ testato
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `str` che rappresenta il nome dell'$LLM_G$ testato
- `getDataEsecuzione(): datetime`
  - **Descrizione**:
    - restituisce la data di esecuzione del test
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `datetime` che rappresenta la data di esecuzione del test
- `getNomeSet(): str`
  - **Descrizione**:
    - restituisce il nome del set di domande utilizzato
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `str` che rappresenta il nome del set di domande utilizzato
- `getRisultatiDomande(): set[RisultatoSingolaDomanda]`
  - **Descrizione**:
    - restituisce un set di oggetti di tipo `RisultatoSingolaDomanda` che rappresentano i risultati delle domande del test
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `set[RisultatoSingolaDomanda]` che rappresenta i risultati delle domande del test
- `serialize(): dict`
  - **Descrizione**:
    - restituisce un `dict` con tutti i dati del risultato del test
  - **Input**:
    - nessuno
  - **Output**:
    - `dict` con tutti i dati del risultato del test

- `serializeForList(): dict`
  - **Descrizione**:
    - restituisce un `dict` con i dati principali del risultato del test, senza i risultati delle domande
  - **Input**:
    - nessuno
  - **Output**:
    - `dict` con i dati principali del risultato del test, senza i risultati delle domande

#### 4.1.1.2 Application

##### 4.1.1.2.1 *Elemento Domanda*$_G$

Le dipendenze dei servizi di *elemento domanda*$_G$ sono:

- **ElementoDomanda**: rappresenta il dominio di *elemento domanda*$_G$
- **ElementoDomandaUseCase**: rappresentano i casi d'uso implementati dai servizi
- **ElementiDomandaPorts**: rappresentano le porte utilizzate dai servizi

###### 4.1.1.2.1.1 AddService


###### 4.1.1.2.1.1.1 AddElementoDomandaService

- **Attributi**
  - `port`: porta utilizzata per salvare l'*elemento domanda*$_G$
- **Metodi**
  - `addElementoDomanda(domanda: str, risposta: str): ElementoDomanda`
    - **Descrizione**:
      - aggiunge un *elemento domanda*$_G$ a partire da domanda e risposta
    - **Input**:
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena creato
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione della domanda o della risposta

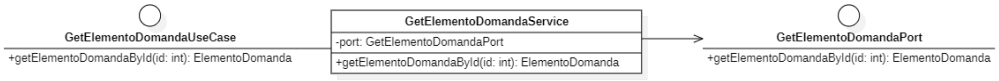
###### 4.1.1.2.1.1.2 AddElementoDomandaUseCase

- **Metodi**
  - `addElementoDomanda(domanda: str, risposta: str): ElementoDomanda`
    - **Descrizione**:
      - aggiunge un *elemento domanda*$_G$ a partire da domanda e risposta
    - **Input**:
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - oggetto di tipo `ElementoDomanda`

#### 4.1.1.2.1.1.3 SaveElementoDomandaPort

- **Metodi**
  - `saveElementoDomanda(elementoDomanda: ElementoDomanda): ElementoDomanda`
    - **Descrizione**:
      - salva un *elemento domanda*$_G$ nel database
    - **Input**:
      - `elementoDomanda`: oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ da salvare
    - **Output**:
      - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena salvato nel database

### 4.1.1.2.1.2 GetService




#### 4.1.1.2.1.2.1 GetElementoDomandaService

- **Attributi**
  - `port`: porta utilizzata per ottenere l'*elemento domanda*$_G$
- **Metodi**
  - `getElementoDomandaById(id: int): ElementoDomanda`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - **Output**:
      - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena ottenuto
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione dell'id

#### 4.1.1.2.1.2.2 GetElementoDomandaUseCase

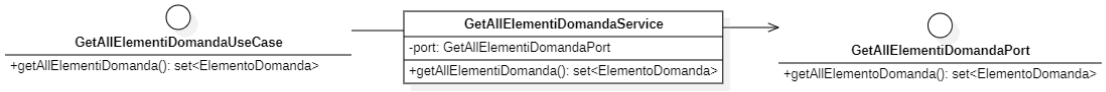
- **Metodi**
  - `getElementoDomandaById(id: int): ElementoDomanda`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - **Output**:
      - oggetto di tipo `ElementoDomanda`

#### 4.1.1.2.1.2.3 GetElementoDomandaPort

- **Metodi**
  - `getElementoDomandaById(id: int): ElementoDomanda`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ dal database, a partire dal suo id
    - **Input**:

- `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
  - **Output**:
    - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena ottenuto dal database

#### 4.1.1.2.1.3 GetAllService




##### 4.1.1.2.1.3.1 GetAllElementiDomandaService

- **Attributi**
  - `port`: porta utilizzata per ottenere tutti gli elementi domanda
- **Metodi**
  - `getAllElementiDomanda(): Set[ElementoDomanda]`
    - **Descrizione**:
      - ritorna un set di tutti gli elementi domanda
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appena ottenuti

##### 4.1.1.2.1.3.2 GetAllElementiDomandaUseCase

- **Metodi**
  - `getAllElementiDomanda(): Set[ElementoDomanda]`
    - **Descrizione**:
      - ritorna un set di tutti gli elementi domanda
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `ElementoDomandaù`

##### 4.1.1.2.1.3.3 GetAllElementiDomandaPort

- **Metodi**
  - `getAllElementiDomanda(): Set[ElementoDomanda]`
    - **Descrizione**:
      - ritorna un set di tutti gli elementi domanda presenti nel database
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appena ottenuti dal database

#### 4.1.1.2.1.4 DeleteService

#### 4.1.1.2.1.4.1 DeleteElementiDomandaService

- **Attributi**
  - `port`: porta utilizzata per eliminare gli elementi domanda
- **Metodi**
  - `deleteElementiDomanda(ids: Set[int]): bool`
    - **Descrizione**:
      - elimina gli elementi domanda a partire da un set di id
    - **Input**:
      - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di eliminazione
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione degli id

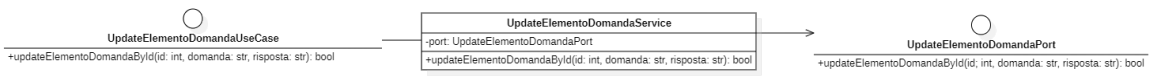
#### 4.1.1.2.1.4.2 DeleteElementiDomandaUseCase

- **Metodi**
  - `deleteElementiDomanda(ids: Set[int]): bool`
    - **Descrizione**:
      - elimina gli elementi domanda a partire da un set di id
    - **Input**:
      - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di eliminazione

#### 4.1.1.2.1.4.3 DeleteElementiDomandaPort

- **Metodi**
  - `deleteElementiDomanda(ids: Set[int]): bool`
    - **Descrizione**:
      - elimina gli elementi domanda nel database, a partire da un set di id
    - **Input**:
      - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di eliminazione nel database

#### 4.1.1.2.1.5 UpdateService




#### 4.1.1.2.1.5.1 UpdateElementoDomandaService

- **Attributi**
  - `port`: porta utilizzata per aggiornare gli elementi domanda
- **Metodi**
  - `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
    - **Descrizione**:

    - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
  - **Input**:
    - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - `domanda`: stringa che rappresenta il testo della domanda
    - `risposta`: stringa che rappresenta il testo della risposta
  - **Output**:
    - booleano che rappresenta il risultato dell'operazione di aggiornamento
  - **Eccezioni**:
    - `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione dell'id, della domanda o della risposta

#### 4.1.1.2.1.5.2 UpdateElementoDomandaUseCase

- **Metodi**
  - `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
    - **Descrizione**:
      - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di aggiornamento

#### 4.1.1.2.1.5.3 UpdateElementoDomandaPort

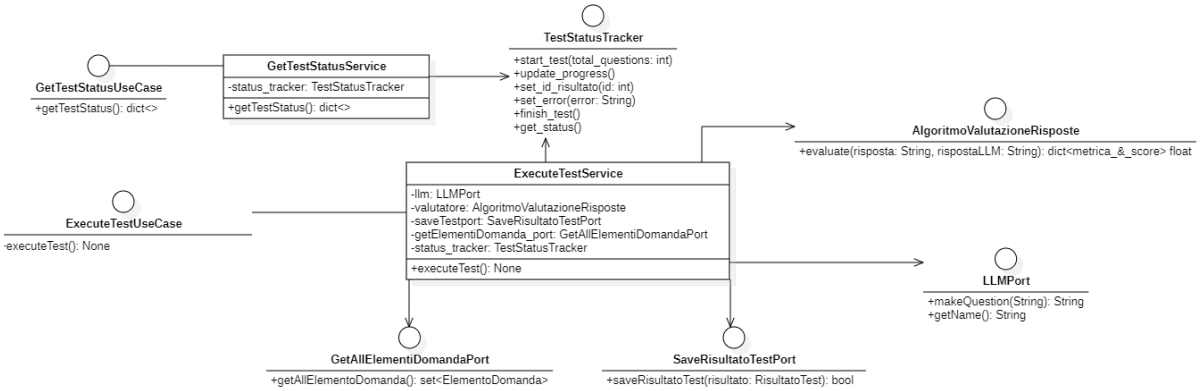
- **Metodi**
  - `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
    - **Descrizione**:
      - aggiorna domanda e risposta di un *elemento domanda*$_G$ nel database, a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di aggiornamento nel database

#### 4.1.1.2.2 Esecuzione Test

Le dipendenze dei servizi di esecuzione test sono:

- **ExecuteTestUseCase**: rappresentano i casi d'uso per eseguire il test
- **GetTestStatusUseCase**: rappresenta il caso d'uso per ottenere lo stato del test
- **LLMPort**: rappresenta la porta per comunicare con l'$LLM_G$
- **SaveRisultatoTestPort**: rappresenta la porta per salvare il risultato del test
- **GetAllElementiDomandaPort**: rappresenta la porta per ottenere tutti gli elementi domanda presenti nel sistema
- **AlgoritmoValutazioneRisposte**: rappresenta la classe responsabile della valutazione delle risposte
- **TestStatusTracker**: rappresenta la classe responsabile del tracciamento dello stato del test
- **RisultatoTest**: rappresenta il dominio di risultato test
- **RisultatoSingolaDomanda**: rappresenta il dominio di risultato per singola domanda

#### 4.1.1.2.2.1 ExecuteService




##### 4.1.1.2.2.1.1 ExecuteTestService

- **Attributi**
  - `status_tracker`: oggetto responsabile del tracciamento dello stato del test
  - $llm_G$: porta utilizzata per comunicare con l'$LLM_G$
  - `valutatore`: oggetto responsabile della valutazione delle risposte
  - `saveRisultatoTestPort`: porta utilizzata per salvare il risultato del test
  - `getElementiDomandaPort`: porta utilizzata per ottenere gli elementi domanda da testare
- **Metodi**
  - `executeTest(): None`
    - **Descrizione**:
      - esegue il test a partire da tutti gli elementi domanda presenti nel sistema, utilizzando l'$LLM_G$ per ottenere le risposte da confrontare. Si occupa anche di valutare le risposte ottenute e di salvare il risultato del test
    - **Input**:
      - nessuno
    - **Output**:
      - nessuno

##### 4.1.1.2.2.1.2 ExecuteTestUseCase

- **Metodi**
  - `executeTest(): None`
    - **Descrizione**:
      - esegue il test e si occupa del salvataggio del risultato
    - **Input**:
      - nessuno
    - **Output**:
      - nessuno

##### 4.1.1.2.2.1.3 LLMPort

- **Metodi**
  - `makeQuestion(domanda: str): str`
    - **Descrizione**:
      - ritorna la risposta fornita dall'$LLM_G$ a partire dalla domanda
    - **Input**:
      - `domanda`: stringa che rappresenta la domanda da porre all'$LLM_G$
    - **Output**:

        - oggetto di tipo `str` che rappresenta la risposta fornita dall'$LLM_G$
    - `getName(): str`
        - **Descrizione**:
            - ritorna il nome dell'$LLM_G$ in uso
        - **Input**:
            - nessuno
        - **Output**:
            - oggetto di tipo `str` che rappresenta il nome dell'$LLM_G$ in uso

#### 4.1.1.2.2.1.4 AlgoritmoValutazioneRisposte

- **Metodi**
    - `evaluate(risposta_attesa: str, risposta_llm: str): tuple[dict[str, float], float]`
        - **Descrizione**:
            - valuta la risposta fornita dall'$LLM_G$ rispetto alla risposta attesa e ritorna le metriche di valutazione e il punteggio di similitudine
        - **Input**:
            - `risposta_attesa`: stringa che rappresenta la risposta attesa
            - `risposta_llm`: stringa che rappresenta la risposta fornita dall'$LLM_G$
        - **Output**:
            - oggetto di tipo `tuple` contenente un dizionario con le metriche di valutazione e un float con il punteggio di similitudine

#### 4.1.1.2.2.1.5 SaveRisultatoTestPort

- **Metodi**
    - `saveRisultatoTest(risultato: RisultatoTest): RisultatoTest`
        - **Descrizione**:
            - salva il risultato del test nel database
        - **Input**:
            - `risultato`: oggetto di tipo `RisultatoTest` che rappresenta il risultato del test da salvare
        - **Output**:
            - oggetto di tipo `RisultatoTest` che rappresenta il risultato del test appena salvato nel database

#### 4.1.1.2.2.1.6 GetAllElementiDomandaPort

- **Metodi**
    - `getAllElementiDomanda(): Set[ElementoDomanda]`
        - **Descrizione**:
            - ritorna un set di tutti gli elementi domanda presenti nel sistema
        - **Input**:
            - nessuno
        - **Output**:
            - set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appena ottenuti

#### 4.1.1.2.2.1.7 GetTestStatusService

- **Attributi**
    - `status_tracker`: oggetto responsabile del tracciamento dello stato del test
- **Metodi**

- `getTestStatus(): dict`
  - **Descrizione**:
    - ritorna lo stato del test in corso
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `dict` che rappresenta lo stato del test in corso

#### 4.1.1.2.2.1.8 GetTestStatusUseCase

- **Metodi**
  - `getTestStatus(): dict`
    - **Descrizione**:
      - ritorna lo stato del test in corso
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `dict` che rappresenta lo stato del test in corso

#### 4.1.1.2.3 Risultato Test

Le dipendenze del servizio di risultato test sono:

- **GetRisultatoTestPort**: rappresenta la porta per ottenere il risultato del test
- **GetAllRisultatiTestPort**: rappresenta la porta per ottenere tutti i risultati del test
- **GetRisultatoSingolaDomandaPort**: rappresenta la porta per ottenere il risultato di una singola domanda
- **GetRisultatoTestUseCase**: rappresenta il caso d'uso per ottenere il risultato del test
- **GetAllRisultatiTestUseCase**: rappresenta il caso d'uso per ottenere tutti i risultati del test
- **GetRisultatoSingolaDomandaUseCase**: rappresenta il caso d'uso per ottenere il risultato di una singola domanda
- **RisultatoTest**: rappresenta il dominio di risultato test
- **RisultatoSingolaDomanda**: rappresenta il dominio di risultato per singola domanda

#### 4.1.1.2.3.1 GetRisultatoTest


##### 4.1.1.2.3.1.1 GetRisultatoTestService

- **Attributi**
  - `port`: porta utilizzata per ottenere il risultato del test
- **Metodi**
  - `getRisultatoTest(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna il risultato del test con l'id specificato
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato del test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato del test appena ottenuto
    - **Eccezioni**:

- `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione dell'id

#### 4.1.1.2.3.1.2 GetRisultatoTestUseCase

- **Metodi**
  - `getRisultatoTest(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna il risultato del test con l'id specificato
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato del test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato del test appena ottenuto

#### 4.1.1.2.3.1.3 GetRisultatoTestPort

- **Metodi**
  - `getRisultatoTest(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna il risultato del test con l'id specificato
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato del test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato del test con l'id specificato

### 4.1.1.2.3.2 GetAllRisultatiTest


#### 4.1.1.2.3.2.1 GetAllRisultatiTestService

- **Attributi**
  - `port`: porta utilizzata per ottenere tutti i risultati del test
- **Metodi**
  - `getAllRisultatiTest(): set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati del test
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `RisultatoTest` che rappresentano i risultati del test appena ottenuti

#### 4.1.1.2.3.2.2 GetAllRisultatiTestUseCase

- **Metodi**
  - `getAllRisultatiTest(): set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati del test
    - **Input**:

    - nessuno
  - **Output**:
    - set di oggetti di tipo `RisultatoTest` che rappresentano i risultati del test appena ottenuti

#### 4.1.1.2.3.2.3 GetAllRisultatiTestPort

- **Metodi**
  - `getAllRisultatiTest(): set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati del test presenti nel sistema
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `RisultatoTest` che rappresentano tutti i risultati di test presenti nel sistema

#### 4.1.1.2.3.3 GetRisultatoSingolaDomanda


##### 4.1.1.2.3.3.1 GetRisultatoSingolaDomandaService

- **Attributi**
  - `port`: porta utilizzata per ottenere il risultato di una specifica domanda di un test
- **Metodi**
  - `getRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna il risultato di una specifica domanda di un test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato della domanda di uno specifico test
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato della domanda appena ottenuto
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui ci siano problemi di validazione dell'id

##### 4.1.1.2.3.3.2 GetRisultatoSingolaDomandaUseCase

- **Metodi**
  - `getRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna il risultato di una specifica domanda di un test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato della domanda di uno specifico test
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato della domanda appena ottenuto

##### 4.1.1.2.3.3.3 GetRisultatoSingolaDomandaPort

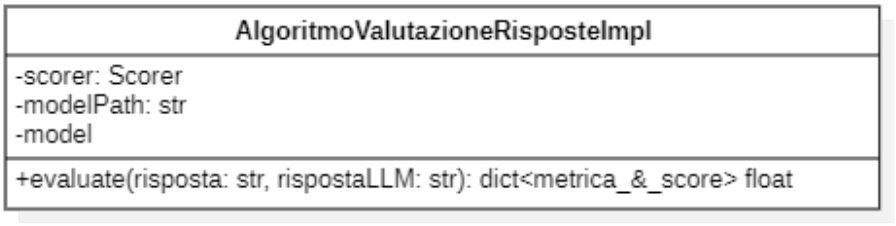
- **Metodi**
  - `getRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna il risultato di una specifica domanda di un test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato della domanda di uno specifico test
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato della domanda appena ottenuto

#### 4.1.1.2.4 Evaluation

Le dipendenze dei servizi di valutazione sono:

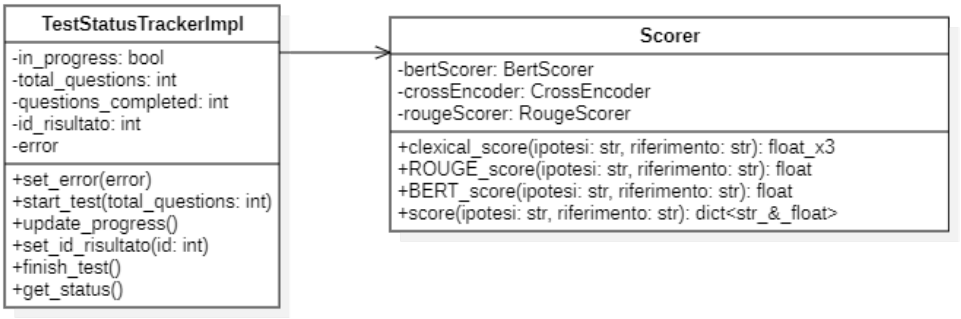
- **AlgoritmoValutazioneRisposte**: rappresenta l'algoritmo di valutazione delle risposte
- **TestStatusTracker**: rappresenta il tracciamento dello stato del test

##### 4.1.1.2.4.1 AlgoritmoValutazioneRisposteImpl




- **Attributi**
  - `scorer`: classe che si occupa di calcolcare i punteggi parziali di valutazione
  - `model`: classe che si occupa di calcolare i punteggi di similitudine a partire dai punteggi parziali
- **Metodi**
  - `evaluate(risposta_attesa: str, risposta_llm: str): tuple[dict[str, float], float]`
    - **Descrizione**:
      - valuta la risposta fornita dall'$LLM_G$ rispetto alla risposta attesa e ritorna le metriche di valutazione e il punteggio di similitudine
    - **Input**:
      - `risposta_attesa`: stringa che rappresenta la risposta attesa
      - `risposta_llm`: stringa che rappresenta la risposta fornita dall'$LLM_G$
    - **Output**:
      - oggetto di tipo `tuple` contenente un dizionario con le metriche di valutazione e un float con il punteggio di similitudine
  - `transformInput(data: dict): DataFrame`
    - **Descrizione**:
      - trasforma i dati di input in un formato compatibile con il modello di valutazione
    - **Input**:
      - `data`: dizionario contenente i dati di input
    - **Output**:
      - oggetto di tipo `DataFrame` contenente i dati trasformati

#### 4.1.1.2.4.2 TestStatusTrackerImpl




- **Attributi**
  - `in_progress`: booleano che rappresenta se il test è in corso o meno
  - `total_questions`: intero che rappresenta il numero totale di domande del test
  - `questions_completed`: intero che rappresenta il numero della domanda completate
  - `id_risultato`: intero che rappresenta l'id del risultato del test
  - `error`: Exception che rappresenta l'eventuale eccezione sollevata durante l'esecuzione del test
- **Metodi**
  - `start_test(total_questions: int): None`
    - **Descrizione**:
      - inizializza il tracciamento dello stato del test
    - **Input**:
      - `total_questions`: intero che rappresenta il numero totale di domande del test
    - **Output**:
      - nessuno
  - `update_progress()`: None
    - **Descrizione**:
      - aggiorna lo stato del test in corso facendo avanzare il numero delle domande completate
    - **Input**:
      - nessuno
    - **Output**:
      - nessuno
  - `set_id_risultato(id_risultato: int): None`
    - **Descrizione**:
      - setta l'id del risultato del test
    - **Input**:
      - `id_risultato`: intero che rappresenta l'id del risultato del test
    - **Output**:
      - nessuno
  - `finish_test(): None`
    - **Descrizione**:
      - termina il tracciamento dello stato del test impostando in_progress a False
    - **Input**:
      - nessuno
    - **Output**:
      - nessuno

- `set_error(error: Exception): None`
  - **Descrizione**:
    - setta l'eventuale eccezione sollevata durante l'esecuzione del test
  - **Input**:
    - `error`: eccezione sollevata durante l'esecuzione del test
  - **Output**:
    - nessuno
- `get_status(): dict`
  - **Descrizione**:
    - ritorna lo stato del test in corso
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `dict` che rappresenta lo stato del test in corso

#### 4.1.1.2.4.3 Scorer

- **Attributi**
  - `bertScorer`: oggetto di tipo `BertScorer` che si occupa di calcolare lo score Bert
  - `crossEncoder`: oggetto di tipo `CrossEncoder` che si occupa di calcolare lo score CrossEncoder
  - `rougeScorer`: oggetto di tipo `RougeScorer` che si occupa di calcolare lo score Rouge
- **Metodi**
  - `clexical_score(ipotesi: str, riferimento: str): (float,float,float)`
    - **Descrizione**:
      - calcola i punteggi Bleu, Ter e Chrf a partire da un'ipotesi e un riferimento
    - **Input**:
      - `ipotesi`: stringa che rappresenta l'ipotesi
      - `riferimento`: stringa che rappresenta il riferimento
    - **Output**:
      - oggetto di tipo `tuple` contenente i punteggi Bleu, Ter e Chrf
  - `ROUGE_score(ipotesi: str, riferimento: str): float`
    - **Descrizione**:
      - calcola il punteggio Rouge a partire da un'ipotesi e un riferimento
    - **Input**:
      - `ipotesi`: stringa che rappresenta l'ipotesi
      - `riferimento`: stringa che rappresenta il riferimento
    - **Output**:
      - valore dello score Rouge in float
  - `BERT_score(ipotesi: str, riferimento: str): float`
    - **Descrizione**:
      - calcola il punteggio Bert a partire da un'ipotesi e un riferimento
    - **Input**:
      - `ipotesi`: stringa che rappresenta l'ipotesi
      - `riferimento`: stringa che rappresenta il riferimento
    - **Output**:
      - valore dello score Bert in float
  - `score(ipotesi: str, riferimento: str): dict[str, float]`
    - **Descrizione**:
      - calcola i punteggi di valutazione a partire da un'ipotesi e un riferimento

- **Input**:
  - `ipotesi`: stringa che rappresenta l'ipotesi
  - `riferimento`: stringa che rappresenta il riferimento
- **Output**:
  - oggetto di tipo `dict` contenente i le metriche utilizzate e i relativi punteggi di valutazione

#### 4.1.1.3 Input Adapters (REST Controllers)

##### 4.1.1.3.1 *Elemento Domanda$_G$*

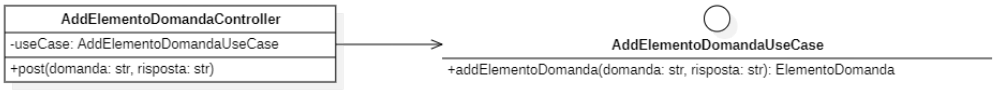
Le dipendenze dei REST controllers di *elemento domanda$_G$* sono:

- **ElementoDomandaUseCase**: rappresentano i casi d'uso implementati dai

servizi

- **Containers**: rappresentano le classi che gestiscono le dependency injection
- **Inject** e **Provide**: rappresentano le funzioni per l'iniezione delle dipendenze
- **BadRequest** e **UnsupportedMediaType**: rappresentano alcune eccezioni catturate

###### 4.1.1.3.1.1 AddController




###### 4.1.1.3.1.1.1 AddElementoDomandaController

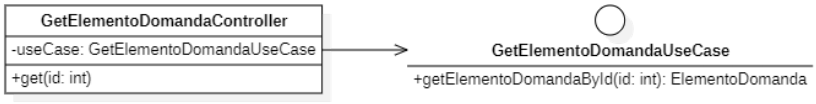
- **Attributi**
  - `useCase`: caso d'uso per aggiungere un *elemento domanda$_G$*, iniettato tramite dependency injection
- **Metodi**
  - `post(): Response`
    - **Descrizione**:
      - aggiunge un *elemento domanda$_G$* a partire da domanda e risposta e lo ritorna all'interno della risposta
    - **Input**:
      - `domanda` e `risposta` tramite body json
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno l'*elemento domanda$_G$* appena creato
      - messaggio di errore in caso di:
        - problemi di validazione della domanda o della risposta
        - errore interno del server
        - mancanza di domanda o risposta nel body json

###### 4.1.1.3.1.1.2 AddElementoDomandaUseCase

- **Metodi**
  - `addElementoDomanda(domanda: str, risposta: str): ElementoDomanda`
    - **Descrizione**:
      - Aggiunge un *elemento domanda$_G$* a partire da domanda e risposta e lo ritorna
    - **Input**:
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta

- **Output**:
  - oggetto di tipo `ElementoDomanda`

#### 4.1.1.3.1.2 GetController



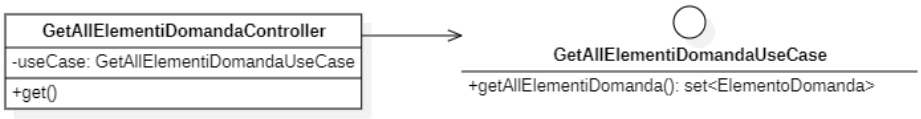

##### 4.1.1.3.1.2.1 GetElementoDomandaController

- **Attributi**
  - `useCase`: caso d'uso per ottenere un *elemento domanda*$_G$, iniettato tramite dependency injection
- **Metodi**
  - `get(id: int): Response`
    - **Descrizione**:
      - ritorna una risposta contenente un *elemento domanda*$_G$, a partire dal suo id
    - **Input**:
      - `id` tramite url param
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno l'*elemento domanda*$_G$ appena ottenuto
      - messaggio di errore in caso di:
        - problemi di validazione dell'id
        - errore interno del server

##### 4.1.1.3.1.2.2 GetElementoDomandaUseCase

- **Metodi**
  - `getElementoDomandaById(id: int): ElementoDomanda`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - **Output**:
      - oggetto di tipo `ElementoDomanda`

#### 4.1.1.3.1.3 GetAllController




##### 4.1.1.3.1.3.1 GetAllElementiDomandaController

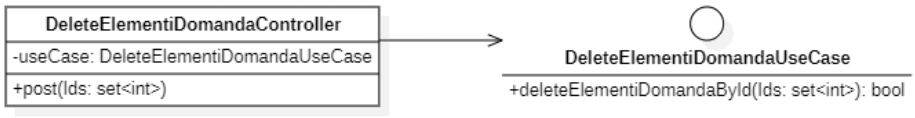
- **Attributi**
  - `useCase`: caso d'uso per ottenere tutti gli elementi domanda, iniettato tramite dependency injection
- **Metodi**
  - `get(): Response`

  - **Descrizione**:
    - ritorna una risposta contenente un set di tutti gli elementi domanda presenti
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il set di elementi domanda appena ottenuti
    - messaggio di errore in caso di:
      - errore interno del server

#### 4.1.1.3.1.3.2 GetAllElementiDomandaUseCase

- **Metodi**
  - `getAllElementiDomanda(): Set[ElementoDomanda]`
    - **Descrizione**:
      - ritorna un set di tutti gli elementi domanda presenti
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `ElementoDomanda`

### 4.1.1.3.1.4 DeleteController




#### 4.1.1.3.1.4.1 DeleteElementiDomandaController

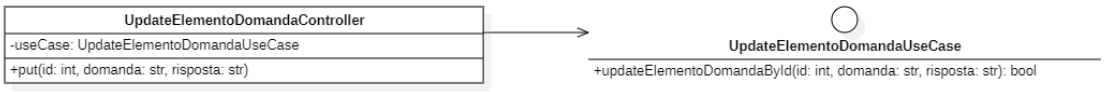
- **Attributi**
  - `useCase`: caso d'uso per eliminare gli elementi domanda, iniettato tramite dependency injection
- **Metodi**
  - `delete(): Response`
    - **Descrizione**:
      - elimina uno o più elementi domanda a partire da un set di id
    - **Input**:
      - `ids` tramite body json
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il risultato dell'operazione di eliminazione
      - messaggio di errore in caso di:
        - problemi di validazione degli id
        - errore interno del server
        - mancanza di ids nel body json

#### 4.1.1.3.1.4.2 DeleteElementiDomandaUseCase

- **Metodi**
  - `deleteElementiDomanda(ids: Set[int]): bool`
    - **Descrizione**:
      - elimina uno o più elementi domanda a partire da un set di id

- **Input**:
  - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
- **Output**:
  - booleano che rappresenta il risultato dell'operazione di eliminazione

#### 4.1.1.3.1.5 UpdateController




##### 4.1.1.3.1.5.1 UpdateElementoDomandaController

- **Attributi**
  - `useCase`: caso d'uso per aggiornare un *elemento domanda*$_G$, iniettato tramite dependency injection
- **Metodi**
  - `put(id: int): Response`
    - **Descrizione**:
      - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id` tramite url param
      - `domanda` e `risposta` tramite body json
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il risultato dell'operazione di aggiornamento
      - messaggio di errore in caso di:
        - problemi di validazione dell'id, della domanda o della risposta
        - errore interno del server
        - mancanza di domanda o risposta nel body json

##### 4.1.1.3.1.5.2 UpdateElementoDomandaUseCase

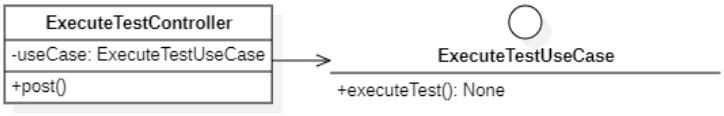
- **Metodi**
  - `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
    - **Descrizione**:
      - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di aggiornamento

#### 4.1.1.3.2 Esecuzione Test

Le dipendenze dei REST controllers di esecuzione test sono:

- **ExecuteTestUseCase**: rappresentano i casi d'uso per l'esecuzione del test implementati dai servizi
- **GetTestStatusUseCase**: rappresenta il caso d'uso per ottenere lo stato del test
- **Containers**: rappresentano le classi che gestiscono le dependency injection
- **Inject** e **Provide**: rappresentano le funzioni per l'iniezione delle dipendenze

#### 4.1.1.3.2.1 ExecuteController



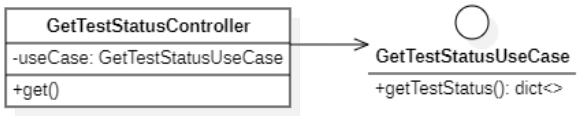

##### 4.1.1.3.2.1.1 ExecuteTestController

- **Attributi**
  - `useCase`: caso d'uso per eseguire il test, iniettato tramite dependency injection
  - `status_tracker`: caso d'uso per monitorare lo stato di avanzamento del test, iniettato tramite dependency injection
- **Metodi**
  - `post(): Response`
    - **Descrizione**:
      - avvia il test a partire da tutti gli elementi domanda presenti nel sistema e ne monitora lo stato durante la sua esecuzione
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno un messaggio di corretto avvio del test
      - messaggio di errore in caso di:
        - errore di comunicazione con l'$LLM_G$
        - errore interno del server

##### 4.1.1.3.2.1.2 ExecuteTestUseCase

- **Metodi**
  - `executeTest(): None`
    - **Descrizione**:
      - esegue il test a partire da tutti gli elementi domanda presenti nel sistema, utilizzando l'$LLM_G$ per ottenere le risposte da confrontare. Si occupa anche di valutare le risposte ottenute e di salvare il risultato del test
    - **Input**:
      - nessuno
    - **Output**:
      - nessuno

#### 4.1.1.3.2.2 StatusController




##### 4.1.1.3.2.2.1 GetTestStatusController

- **Attributi**
  - `useCase`: caso d'uso per ottenere lo stato del test, iniettato tramite dependency injection
- **Metodi**

- `get(): Response`
  - **Descrizione**:
    - ritorna lo stato del test in corso
  - **Input**:
    - nessuno
  - **Output**:
    - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno lo stato del test in corso
    - messaggio di errore in caso di:
      - errore interno del server

#### 4.1.1.3.2.2.2 GetTestStatusUseCase

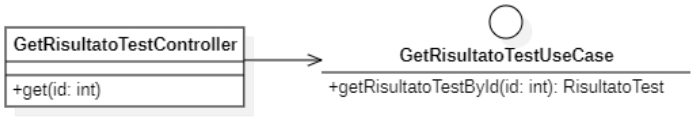
- **Metodi**
  - `getTestStatus(): dict`
    - **Descrizione**:
      - ritorna lo stato del test in corso
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `dict` che rappresenta lo stato del test in corso

#### 4.1.1.3.3 Risultato Test

Le dipendenze dei REST controllers di risultato test sono:

- **GetRisultatoTestUseCase**: rappresenta il caso d'uso per ottenere il risultato del test
- **GetAllRisultatiTestUseCase**: rappresenta il caso d'uso per ottenere tutti i risultati del test
- **GetRisultatoSingolaDomandaUseCase**: rappresenta il caso d'uso per ottenere il risultato di una singola domanda
- **Containers**: rappresentano le classi che gestiscono le dependency injection
- **Inject** e **Provide**: rappresentano le funzioni per l'iniezione delle dipendenze

#### 4.1.1.3.3.1 Get Controllers




##### 4.1.1.3.3.1.1 GetRisultatoTestController

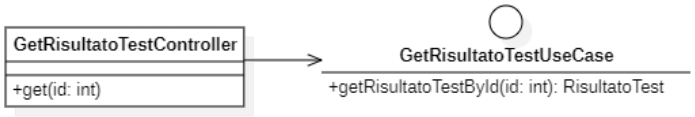
- **Attributi**
  - `useCase`: caso d'uso per ottenere il risultato del test, iniettato tramite dependency injection
- **Metodi**
  - `get(id: int): Response`
    - **Descrizione**:
      - ritorna una risposta contenente il risultato del test a partire dal suo id
    - **Input**:
      - `id` tramite url param
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il risultato del test appena ottenuto
      - messaggio di errore in caso di:

        - problemi di validazione dell'id
        - errore interno del server

#### 4.1.1.3.3.1.2 GetRisultatoTestUseCase

- **Metodi**
  - `getRisultatoTest(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna il risultato del test con l'id specificato
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato del test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato del test appena ottenuto

#### 4.1.1.3.3.2 GetAllRisultatoTest




#### 4.1.1.3.3.2.1 GetAllRisultatiTestController

- **Attributi**
  - `useCase`: caso d'uso per ottenere tutti i risultati del test, iniettato tramite dependency injection
- **Metodi**
  - `get(): Response`
    - **Descrizione**:
      - ritorna una risposta contenente un set di tutti i risultati del test
    - **Input**:
      - nessuno
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il set di risultati del test appena ottenuti
      - messaggio di errore in caso di:
        - errore interno del server

#### 4.1.1.3.3.2.2 GetAllRisultatiTestUseCase

- **Metodi**
  - `getAllRisultatiTest(): set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati del test
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `RisultatoTest` che rappresentano i risultati del test appena ottenuti

#### 4.1.1.3.3.3 GetRisultatoSingolaDomanda


##### 4.1.1.3.3.3.1 GetRisultatoSingolaDomandaController

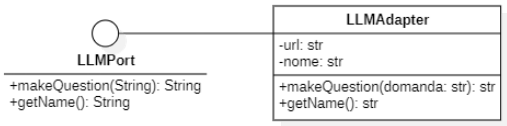
- **Attributi**
  - `useCase`: caso d'uso per ottenere il risultato di una singola domanda, iniettato tramite dependency injection
- **Metodi**
  - `get(id: int): Response`
    - **Descrizione**:
      - ritorna una risposta contenente il risultato di una singola domanda a partire dal suo id
    - **Input**:
      - `id` tramite url param
    - **Output**:
      - oggetto di tipo `Response` che rappresenta la risposta HTTP con all'interno il risultato della singola domanda appena ottenuto
      - messaggio di errore in caso di:
        - problemi di validazione dell'id
        - errore interno del server

##### 4.1.1.3.3.3.2 GetRisultatoSingolaDomandaUseCase

- **Metodi**
  - `getRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna il risultato di una specifica domanda di un test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato della domanda di uno specifico test
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato della domanda appena ottenuto

### 4.1.1.4 Output Adapters

#### 4.1.1.4.1 $LLM_G$




Le dipendenze dell'Adapter di $LLM_G$ sono:

- **LLMPort**: rappresenta la porta implementata dall'Adapter stesso

##### 4.1.1.4.1.1 LLMAdapter

- **Attributi**

- `url`: stringa che rappresenta l'url per comunicare la domanda all'$LLM_G$ ed ottenere la risposta
- `nome`: stringa che rappresenta il nome/modello dell'$LLM_G$ utilizzato

- **Metodi**
  - `makeQuestion(domanda: str): str`
    - **Descrizione**:
      - invia la domanda all'$LLM_G$ e ne ritorna la risposta
    - **Input**:
      - `domanda`: stringa che rappresenta il testo della domanda
    - **Output**:
      - stringa che rappresenta la risposta fornita dall'$LLM_G$
    - **Eccezioni**:
      - `Exception`: eccezione sollevata nel caso in cui ci siano problemi di comunicazione con l'$LLM_G$
  - `getName(): str`
    - **Descrizione**:
      - ritorna il nome dell'$LLM_G$
    - **Input**:
      - nessuno
    - **Output**:
      - stringa che rappresenta il nome dell'$LLM_G$

#### 4.1.1.4.1.2 LLMPort

- **Metodi**
  - `makeQuestion(domanda: str): str`
    - **Descrizione**:
      - invia la domanda all'$LLM_G$ e ne ritorna la risposta
    - **Input**:
      - `domanda`: stringa che rappresenta il testo della domanda
    - **Output**:
      - stringa che rappresenta la risposta fornita dall'$LLM_G$
  - `getName(): str`
    - **Descrizione**:
      - ritorna il nome dell'$LLM_G$
    - **Input**:
      - nessuno
    - **Output**:
      - stringa che rappresenta il nome dell'$LLM_G$

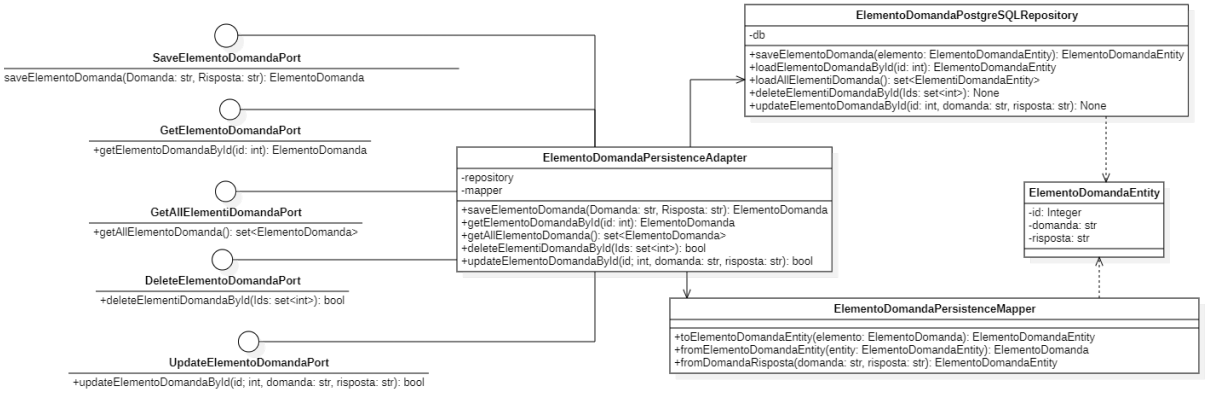
### 4.1.1.4.2 Persistence (PostreSQL)

#### 4.1.1.4.2.1 *Elemento Domanda$_G$*

Le dipendenze della sezione persistence di *elemento domanda$_G$* sono:

- **ElementoDomanda**: rappresenta il dominio di *elemento domanda$_G$*
- **ElementoDomandaPorts**: rappresentano le porte implementate dell'Adapter stesso
- **SQLAlchemyError** e **NoResultFound**: rappresentano alcune eccezioni lanciate da SQLAlchemy e catturate dall'Adapter

- **db**: rappresenta l'istanza del database utilizzato dalle entità del database e dalla *repository*$_G$ per comunicare con il database stesso




#### 4.1.1.4.2.1.1 ElementoDomandaPersistenceAdapter

- **Attributi**
  - `repository`$_G$: *repository*$_G$ utilizzata per comunicare con il database
  - `mapper`: mapper utilizzato per mappare le entità del dominio di business con le entità del database
- **Metodi**
  - `saveElementoDomanda(elementoDomanda: ElementoDomanda): ElementoDomanda`
    - **Descrizione**:
      - salva un *elemento domanda*$_G$ nel database
    - **Input**:
      - `elementoDomanda`: oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ da salvare
    - **Output**:
      - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena salvato nel database
  - `getElementoDomandaById(id: int): ElementoDomanda`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - **Output**:
      - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena ottenuto dal database
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui non venga trovato alcun *elemento domanda*$_G$ con l'id specificato
  - `getAllElementiDomanda(): Set[ElementoDomanda]`
    - **Descrizione**:
      - ritorna un set di tutti gli elementi domanda presenti nel database
    - **Input**:
      - nessuno
    - **Output**:

- set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appena ottenuti dal database

▸ `deleteElementiDomanda(ids: Set[int]): bool`
- **Descrizione**:
  - elimina uno o più elementi domanda a partire da un set di id
- **Input**:
  - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
- **Output**:
  - booleano che rappresenta il risultato dell'operazione di eliminazione nel database

▸ `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
- **Descrizione**:
  - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
- **Input**:
  - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
  - `domanda`: stringa che rappresenta il testo della domanda
  - `risposta`: stringa che rappresenta il testo della risposta
- **Output**:
  - booleano che rappresenta il risultato dell'operazione di aggiornamento nel database
- **Eccezioni**:
  - `ValueError`: eccezione sollevata nel caso in cui non venga trovato alcun *elemento domanda*$_G$ con l'id specificato

#### 4.1.1.4.2.1.2 SaveElementoDomandaPort

- **Metodi**

▸ `saveElementoDomanda(elementoDomanda: ElementoDomanda): ElementoDomanda`
- **Descrizione**:
  - salva un *elemento domanda*$_G$ nel database
- **Input**:
  - `elementoDomanda`: oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ da salvare
- **Output**:
  - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena salvato nel database

#### 4.1.1.4.2.1.3 GetElementoDomandaPort

- **Metodi**

▸ `getElementoDomandaById(id: int): ElementoDomanda`
- **Descrizione**:
  - ritorna un *elemento domanda*$_G$ a partire dal suo id
- **Input**:
  - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
- **Output**:
  - oggetto di tipo `ElementoDomanda` che rappresenta l'*elemento domanda*$_G$ appena ottenuto dal database

#### 4.1.1.4.2.1.4 GetAllElementiDomandaPort

- **Metodi**

- `getAllElementiDomanda(): Set[ElementoDomanda]`
  - **Descrizione**:
    - ritorna un set di tutti gli elementi domanda presenti nel database
  - **Input**:
    - nessuno
  - **Output**:
    - set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appena ottenuti dal database

#### 4.1.1.4.2.1.5 DeleteElementiDomandaPort

- **Metodi**
  - `deleteElementiDomanda(ids: Set[int]): bool`
    - **Descrizione**:
      - elimina uno o più elementi domanda a partire da un set di id
    - **Input**:
      - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di eliminazione nel database

#### 4.1.1.4.2.1.6 UpdateElementoDomandaPort

- **Metodi**
  - `updateElementoDomanda(id: int, domanda: str, risposta: str): bool`
    - **Descrizione**:
      - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
      - `domanda`: stringa che rappresenta il testo della domanda
      - `risposta`: stringa che rappresenta il testo della risposta
    - **Output**:
      - booleano che rappresenta il risultato dell'operazione di aggiornamento nel database

#### 4.1.1.4.2.1.7 ElementoDomandaEntity

- **Attributi**
  - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
  - `domanda`: stringa che rappresenta il testo della domanda
  - `risposta`: stringa che rappresenta il testo della risposta

#### 4.1.1.4.2.1.8 ElementoDomandaPersistenceMapper

- **Metodi**
  - `fromElementoDomandaEntity(entity: ElementoDomandaEntity): ElementoDomanda`
    - **Descrizione**:
      - mappa un oggetto di tipo `ElementoDomandaEntity` in un oggetto di tipo `ElementoDomanda`
    - **Input**:
      - `entity`: oggetto di tipo `ElementoDomandaEntity` da mappare
    - **Output**:
      - oggetto di tipo `ElementoDomanda`

- `toElementoDomandaEntity(elementoDomanda: ElementoDomanda): ElementoDomandaEntity`
  - **Descrizione**:
    - mappa un oggetto di tipo `ElementoDomanda` in un oggetto di tipo `ElementoDomandaEntity`
  - **Input**:
    - `elementoDomanda`: oggetto di tipo `ElementoDomanda` da mappare
  - **Output**:
    - oggetto di tipo `ElementoDomandaEntity`
- `fromDomandaRisposta(domanda: str, risposta: str): ElementoDomandaEntity`
  - **Descrizione**:
    - mappa una domanda e una risposta in un oggetto di tipo `ElementoDomandaEntity`
  - **Input**:
    - `domanda`: stringa che rappresenta il testo della domanda
    - `risposta`: stringa che rappresenta il testo della risposta
  - **Output**:
    - oggetto di tipo `ElementoDomandaEntity`

#### 4.1.1.4.2.1.9 ElementoDomandaPostgreSQLRepository

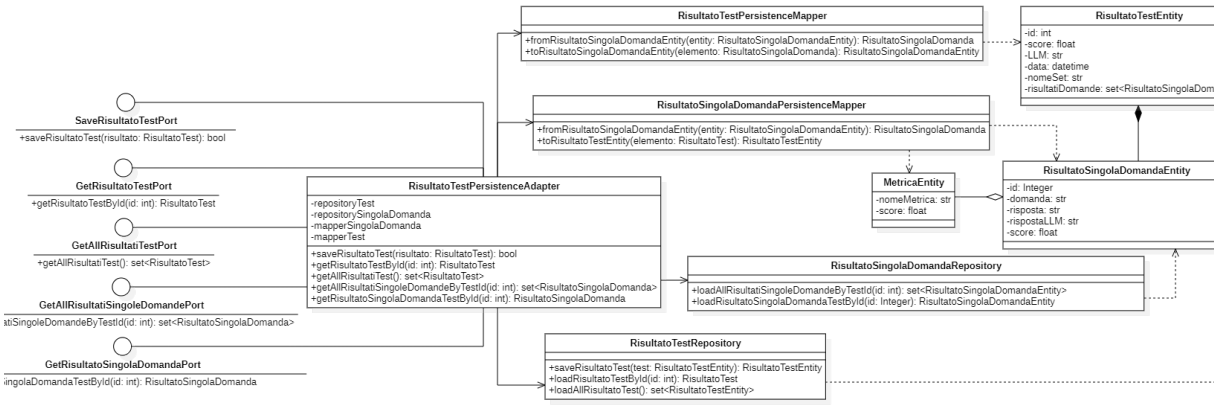
- **Attributi**
  - `db`: istanza del database utilizzata per comunicare con il database stesso
- **Metodi**
  - `saveElementoDomanda(elementoEntity: ElementoDomandaEntity): ElementoDomandaEntity`
    - **Descrizione**:
      - salva un *elemento domanda*$_G$ nel database
    - **Input**:
      - `elementoEntity`: oggetto di tipo `ElementoDomandaEntity` che rappresenta l'*elemento domanda*$_G$ da salvare
    - **Output**:
      - oggetto di tipo `ElementoDomandaEntity` che rappresenta l'*elemento domanda*$_G$ appena salvato nel database
    - **Eccezioni**:
      - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database
  - `loadElementoDomandaById(id: int): ElementoDomandaEntity`
    - **Descrizione**:
      - ritorna un *elemento domanda*$_G$ a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - **Output**:
      - oggetto di tipo `ElementoDomandaEntity` che rappresenta l'*elemento domanda*$_G$ appena ottenuto dal database
    - **Eccezioni**:
      - `NoResultFound`: eccezione sollevata nel caso in cui non venga trovato alcun *elemento domanda*$_G$ con l'id specificato
      - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database

- ▸ `loadAllElementiDomanda(): Set[ElementoDomandaEntity]`
  - **Descrizione**:
    - ritorna un set di tutti gli elementi domanda presenti nel database
  - **Input**:
    - nessuno
  - **Output**:
    - set di oggetti di tipo `ElementoDomandaEntity` che rappresentano gli elementi domanda appena ottenuti dal database
  - **Eccezioni**:
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database
- ▸ `deleteElementiDomanda(id: Set[int]): None`
  - **Descrizione**:
    - elimina uno o più elementi domanda a partire da un set di id
  - **Input**:
    - `ids`: set di interi che rappresentano gli identificativi univoci degli elementi domanda da eliminare
  - **Output**:
    - booleano che rappresenta il risultato dell'operazione di eliminazione nel database
  - **Eccezioni**:
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database
- ▸ `updateElementoDomanda(id: int, domanda: str, risposta: str): None`
  - **Descrizione**:
    - aggiorna domanda e risposta di un *elemento domanda*$_G$ a partire dal suo id
  - **Input**:
    - `id`: intero che rappresenta l'identificativo univoco dell'*elemento domanda*$_G$
    - `domanda`: stringa che rappresenta il testo della domanda
    - `risposta`: stringa che rappresenta il testo della risposta
  - **Output**:
    - booleano che rappresenta il risultato dell'operazione di aggiornamento nel database
  - **Eccezioni**:
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database
    - `NoResultFound`: eccezione sollevata nel caso in cui non venga trovato alcun *elemento domanda*$_G$ con l'id specificato

#### 4.1.1.4.2.2 Risultato Test

Le dipendenze della sezione persistence di risultato test sono:

- **RisultatoTest**: rappresenta il dominio di risultato test
- **RisultatoSingolaDomanda**: rappresenta il dominio di risultato del singolo *elemento domanda*$_G$
- **RisultatoTestPorts**: rappresentano le porte implementate dell'Adapter stesso
- **SQLAlchemyError** e **NoResultFound**: rappresentano alcune eccezioni lanciate da SQLAlchemy e catturate dall'Adapter
- **db**: rappresenta l'istanza del database utilizzato dalle entità del database e dalla *repository*$_G$ per comunicare con il database stesso




#### 4.1.1.4.2.2.1 RisultatoTestPersistenceAdapter

- **Attributi**
  - `repositoryTest` : *repository*$_G$ utilizzata per comunicare con il database contenente i risultati del test
  - `repositorySingolaDomanda` : *repository*$_G$ utilizzata per comunicare con il database contenente i risultati del singolo *elemento domanda*$_G$
  - `mapperTest` : mapper utilizzato per mappare le entità del dominio di business con le entità del database risultato test
  - `mapperSingolaDomanda` : mapper utilizzato per mappare le entità del dominio di business con le entità del database singola domanda
- **Metodi**
  - `saveRisultatoTest(risultatoTest: RisultatoTest): RisultatoTest`
    - **Descrizione**:
      - salva un risultato test nel database
    - **Input**:
      - `risultatoTest`: oggetto di tipo `RisultatoTest` che rappresenta il risultato test da salvare
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato test appena salvato nel database
  - `getRisultatoTestById(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna un risultato test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato test appena ottenuto dal database
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui non venga trovato alcun risultato test con l'id specificato
  - `getAllRisultatiTest(): Set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati test presenti nel database
    - **Input**:

        - nessuno
      - **Output**:
        - set di oggetti di tipo `RisultatoTest` che rappresentano i risultati test appena ottenuti dal database

  - `getRisultatoSingolaDomandaById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna un risultato singola domanda a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato singola domanda
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato singola domanda appena ottenuto dal database
    - **Eccezioni**:
      - `ValueError`: eccezione sollevata nel caso in cui non venga trovato alcun risultato singola domanda con l'id specificato

#### 4.1.1.4.2.2.2 SaveRisultatoTestPort

- **Metodi**
  - `saveRisultatoTest(risultatoTest: RisultatoTest): RisultatoTest`
    - **Descrizione**:
      - salva un risultato test nel database
    - **Input**:
      - `risultatoTest`: oggetto di tipo `RisultatoTest` che rappresenta il risultato test da salvare
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato test appena salvato nel database

#### 4.1.1.4.2.2.3 GetRisultatoTestPort

- **Metodi**
  - `getRisultatoTestById(id: int): RisultatoTest`
    - **Descrizione**:
      - ritorna un risultato test a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato test
    - **Output**:
      - oggetto di tipo `RisultatoTest` che rappresenta il risultato test appena ottenuto dal database

#### 4.1.1.4.2.2.4 GetAllRisultatiTestPort

- **Metodi**
  - `getAllRisultatiTest(): Set[RisultatoTest]`
    - **Descrizione**:
      - ritorna un set di tutti i risultati test presenti nel database
    - **Input**:
      - nessuno
    - **Output**:
      - set di oggetti di tipo `RisultatoTest` che rappresentano i risultati test appena ottenuti dal database

##### 4.1.1.4.2.2.5 GetRisultatoSingolaDomandaPort

- **Metodi**
  - `getRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomanda`
    - **Descrizione**:
      - ritorna un risultato singola domanda a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato singola domanda
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda` che rappresenta il risultato singola domanda appena ottenuto dal database

##### 4.1.1.4.2.2.6 RisultatoTestEntity

- **Attributi**
  - `id`: intero che rappresenta l'identificativo univoco del risultato test
  - `score`: float che rappresenta il punteggio del test
  - $LLM_G$ : stringa che rappresenta il nome dell'$LLM_G$ utilizzato per eseguire il test
  - `data`: stringa che rappresenta la data in cui è stato eseguito il test
  - `nomeSet` : stringa che rappresenta il nome del set di domande utilizzato per eseguire il test
  - `risultatiDomande`: set di oggetti di tipo `RisultatoSingolaDomandaEntity` che rappresentano i risultati delle domande del test

##### 4.1.1.4.2.2.7 RisultatoSingolaDomandaEntity

- **Attributi**
  - `id`: intero che rappresenta l'identificativo univoco del risultato singola domanda
  - `domanda`: stringa che rappresenta il testo della domanda
  - `risposta`: stringa che rappresenta il testo della risposta attesa
  - `rispostaLLM`: stringa che rappresenta il testo della risposta ottenuta dall'$LLM_G$
  - `score`: float che rappresenta il punteggio della domanda
  - `risultatoTestId` : intero che rappresenta l'identificativo univoco del risultato test a cui appartiene il risultato singola domanda
  - `risultatiMetriche`: set di oggetti di tipo `RisultatoSingolaDomandaMetricheEntity` che rappresentano i risultati delle metriche del risultato singola domanda

##### 4.1.1.4.2.2.8 RisultatoMetricaEntity

- **Attributi**
  - `nomeMetrica`: stringa che rappresenta il nome della metrica
  - `score`: float che rappresenta il valore della metrica
  - `risultatoSingolaDomandaId` : intero che rappresenta l'identificativo univoco del risultato singola domanda a cui appartiene il risultato metrica

##### 4.1.1.4.2.2.9 RisultatoTestPersistenceMapper

- **Metodi**
  - `fromRisultatoTestEntity(entity: RisultatoTestEntity): RisultatoTest`
    - **Descrizione**:
      - mappa un oggetto di tipo `RisultatoTestEntity` in un oggetto di tipo `RisultatoTest`
    - **Input**:
      - `entity`: oggetto di tipo `RisultatoTestEntity` da mappare
    - **Output**:
      - oggetto di tipo `RisultatoTest`

- `toRisultatoTestEntity(risultatoTest: RisultatoTest): RisultatoTestEntity`
  - **Descrizione**:
    - mappa un oggetto di tipo `RisultatoTest` in un oggetto di tipo `RisultatoTestEntity`
  - **Input**:
    - `risultatoTest`: oggetto di tipo `RisultatoTest` da mappare
  - **Output**:
    - oggetto di tipo `RisultatoTestEntity`

#### 4.1.1.4.2.2.10 RisultatoSingolaDomandaPersistenceMapper

- **Metodi**
  - `fromRisultatoSingolaDomandaEntity(entity: RisultatoSingolaDomandaEntity): RisultatoSingolaDomanda`
    - **Descrizione**:
      - mappa un oggetto di tipo `RisultatoSingolaDomandaEntity` in un oggetto di tipo `RisultatoSingolaDomanda`
    - **Input**:
      - `entity`: oggetto di tipo `RisultatoSingolaDomandaEntity` da mappare
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomanda`
  - `toRisultatoSingolaDomandaEntity(risultatoSingolaDomanda: RisultatoSingolaDomanda): RisultatoSingolaDomandaEntity`
    - **Descrizione**:
      - mappa un oggetto di tipo `RisultatoSingolaDomanda` in un oggetto di tipo `RisultatoSingolaDomandaEntity`
    - **Input**:
      - `risultatoSingolaDomanda`: oggetto di tipo `RisultatoSingolaDomanda` da mappare
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomandaEntity`

#### 4.1.1.4.2.2.11 RisultatoTestPostgreSQLRepository

- **Attributi**
  - `db`: istanza del database utilizzata per comunicare con il database stesso
- **Metodi**
  - `saveRisultatoTest(risultatoEntity: RisultatoTestEntity): RisultatoTestEntity`
    - **Descrizione**:
      - salva un risultato test nel database
    - **Input**:
      - `risultatoEntity`: oggetto di tipo `RisultatoTestEntity` che rappresenta il risultato test da salvare
    - **Output**:
      - oggetto di tipo `RisultatoTestEntity` che rappresenta il risultato test appena salvato nel database
    - **Eccezioni**:
      - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database
  - `loadRisultatoTestById(id: int): RisultatoTestEntity`

  - **Descrizione**:
    - ritorna un risultato test a partire dal suo id
  - **Input**:
    - `id`: intero che rappresenta l'identificativo univoco del risultato test
  - **Output**:
    - oggetto di tipo `RisultatoTestEntity` che rappresenta il risultato test appena ottenuto dal database
  - **Eccezioni**:
    - `NoResultFound`: eccezione sollevata nel caso in cui non venga trovato alcun risultato test con l'id specificato
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database

- `deleteRisultatoTest(id: int): None`
  - **Descrizione**:
    - elimina un risultato test a partire dal suo id
  - **Input**:
    - `id`: intero che rappresenta l'identificativo univoco del risultato test da eliminare
  - **Output**:
    - booleano che rappresenta il risultato dell'operazione di eliminazione nel database
  - **Eccezioni**:
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database

- `loadAllRisultatiTest(): Set[RisultatoTestEntity]`
  - **Descrizione**:
    - ritorna un set di tutti i risultati test presenti nel database
  - **Input**:
    - nessuno
  - **Output**:
    - set di oggetti di tipo `RisultatoTestEntity` che rappresentano i risultati test appena ottenuti dal database
  - **Eccezioni**:
    - `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database

#### 4.1.1.4.2.2.12 RisultatoSingolaDomandaPostgreSQLRepository

- **Attributi**
  - `db`: istanza del database utilizzata per comunicare con il database stesso
- **Metodi**
  - `loadRisultatoSingolaDomandaTestById(id: int): RisultatoSingolaDomandaEntity`
    - **Descrizione**:
      - ritorna un risultato singola domanda a partire dal suo id
    - **Input**:
      - `id`: intero che rappresenta l'identificativo univoco del risultato singola domanda
    - **Output**:
      - oggetto di tipo `RisultatoSingolaDomandaEntity` che rappresenta il risultato singola domanda appena ottenuto dal database
    - **Eccezioni**:

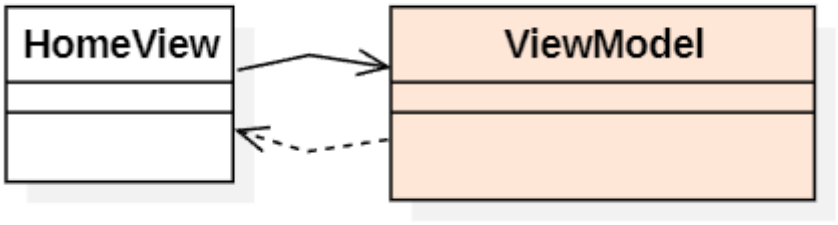
- `NoResultFound`: eccezione sollevata nel caso in cui non venga trovato alcun risultato singola domanda con l'id specificato
- `SQLAlchemyError`: eccezione sollevata nel caso in cui ci siano problemi durante le operazione svolte sul database

#### 4.1.2 Frontend

Il frontend di questa applicazione rende possibile l'esecuzione e la visualizzazione dei risultati, in modod grafic, attravero una pagina web costruita attraverso la tecnologia *Single Page application*$_G$. Per il frontend viene utilizzato Vue JS che utilizza il Pattern MVVM (ModelViewViewModel),

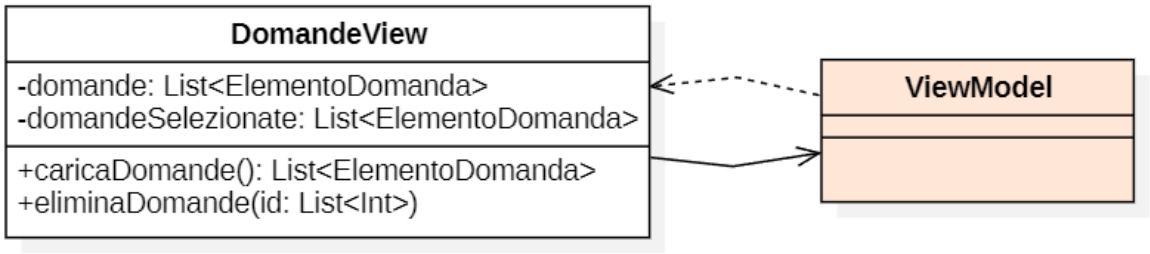
##### 4.1.2.1 HomeView

Vista principale dell'applicazione. Funziona come punto informativo e descrittivo dell'applicazione. Non ha particolari metodi o funzioni in quanto è solo di presentazione.




##### 4.1.2.2 DomandeView

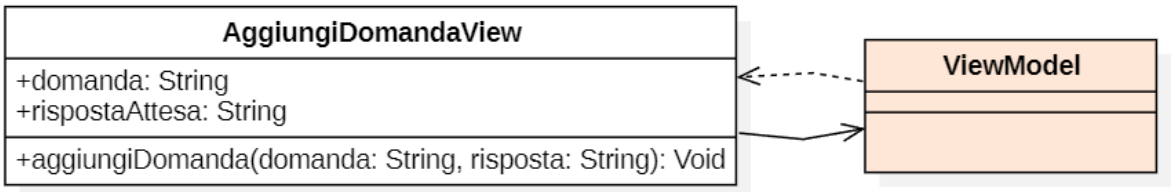
Gestisce e visualizza la lista delle domande esistenti.




- **Attributi**:
  - `domande: List<ElementoDomanda>` : Lista elementi domanda da visualizzare
  - `domandeSelezionate: List<ElementoDomanda>`: Lista elementi domanda selezionati per l'eliminazione
- **Metodi**
  - `caricaDomande(): List<ElementoDomanda>` : Void: Carica le domande dal backend
  - `eliminaDomande(in id:List<Int>)` : Elimina le domande selezionate

##### 4.1.2.3 AggiungiDomandaView

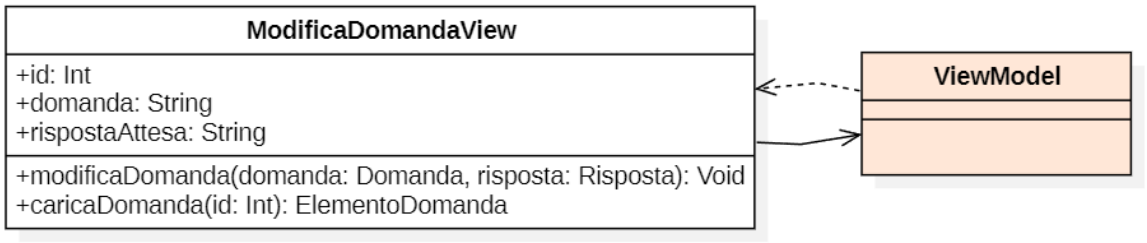
Vista responsabile dell'inserimento di una nuova domanda. Fornisce un modulo per l'inserimento della domanda e della risposta attesa, e invia i dati al backend.




- **Attributi**
  - `domanda: String`: Contenuto testuale della domanda da inserire.
  - `rispostaAttesa: String`: Contenuto testuale della risposta prevista.
- **Metodi**
  - `aggiungiDomanda(domanda:String, in risposta:String): Void`: Valida che entrambi i campi siano compilati, invia i dati tramite una chiamata `POST` al backend, e reindirizza l'utente alla vista delle domande in caso di successo.

#### 4.1.2.4 ModificaDomandaView

Vista per la modifica di una domanda esistente. Carica automaticamente i dati della domanda da modificare e consente l'aggiornamento tramite form.



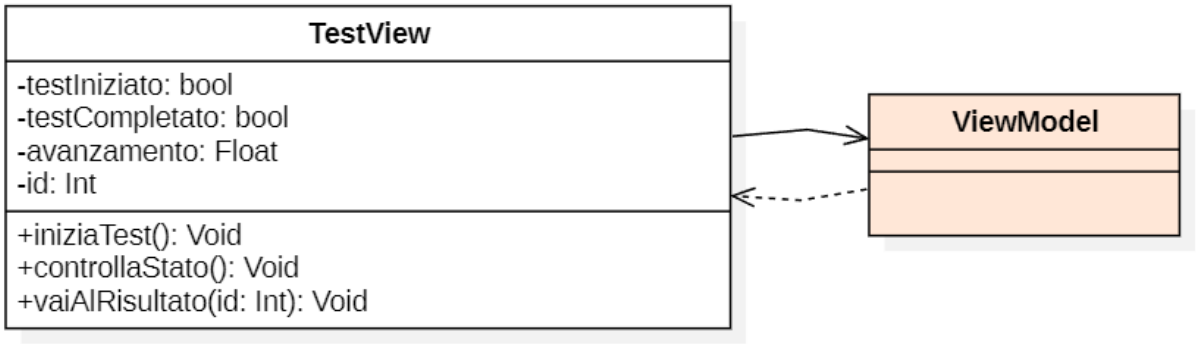

- **Attributi**
  - `id: Int`: Identificativo univoco della domanda, ottenuto dai parametri della route.
  - `domanda: String`: Testo della domanda da modificare.
  - `rispostaAttesa: String`: Testo della risposta prevista da modificare.
- **Metodi**
  - `caricaDomanda(in id:Int): ElementoDomanda`:

  Recupera la domanda da modificare tramite richiesta al backend.
  - `modificaDomanda(in domanda:Domanda, in risposta:Risposta): Void`:

  Invia la modifica della domanda al backend.

#### 4.1.2.5 TestView

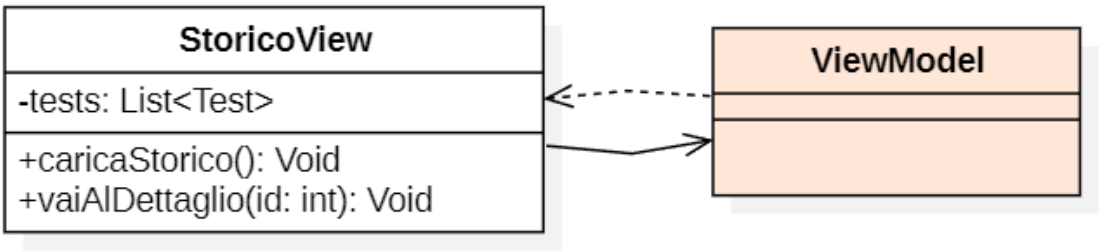
Gestisce l'interfaccia per eseguire un test. Avvia il test, controlla lo stato in tempo reale tramite polling e gestisce la visualizzazione del progresso o del risultato finale.




- **Attributi**
  - `id: Int`: Identificativo del risultato del test completato.
  - `testIniziato: bool`: Indica se il test è attualmente in corso.
  - `testCompletato: bool`: Indica se il test è stato completato.
  - `avanzamento: Float`: Percentuale di avanzamento del test.
- **Metodi**
  - `iniziaTest(): Void`: Invia la richiesta di avvio del test al backend
  - `controllaStato(): Void`: Recupera lo stato corrente del test dal backend e aggiorna l'interfaccia in base alla risposta.
  - `vaiAlRisultato(in id:Int): Void`: Naviga alla vista dei risultati del test appena completato.

#### 4.1.2.6 StoricoView

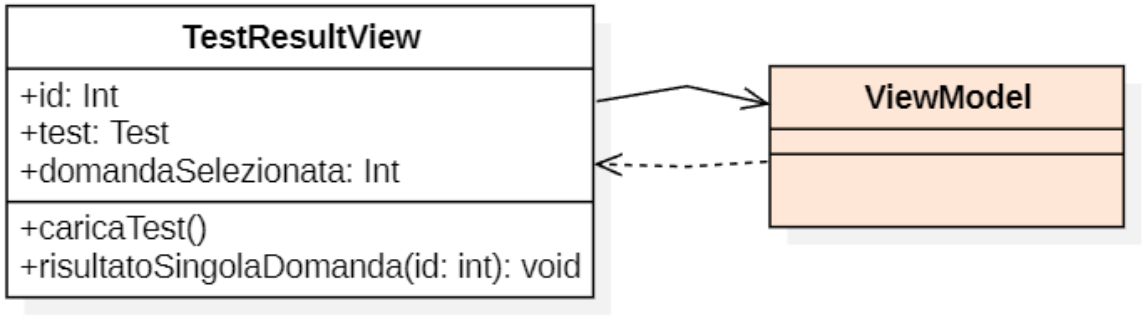
Componente che mostra lo storico dei test eseguiti dall'utente.




- **Attributi**
  - `tests: List<Object>`: Lista dei test eseguiti recuperati dal backend.
- **Metodi**
  - `caricaStorico(): Void`: Recupera la lista dei test dal backend e li ordina per data decrescente.
  - `vaiAlDettaglio(in id:int): Void`: Naviga alla vista di dettaglio del risultato del test selezionato.

#### 4.1.2.7 TestResultView

Componente che visualizza i dettagli di un test, incluse tutte le domande con i rispettivi punteggi. Permette di visualizzare il risultato dettagliato di ogni singola domanda.




- **Attributi**
  - `id: Int`: Identificativo del test, ottenuto dai parametri del router.
  - `test: Test`: Oggetto che rappresenta il test da visualizzare, completo di dettagli e risultati.
  - `domandaSelezionata: Int`: Oggetto che contiene i dati della domanda selezionata per la visualizzazione singola.
- **Metodi**
  - `caricaTest(): Void`: Recupera i dati completi del test dal backend usando l'id corrente.
  - `risultatoSingolaDomanda(in id:int): void`: Recupera dal backend il dettaglio del risultato di una domanda.

#### 4.1.2.8 ViewModel

Il *ViewModel* è una componente concettuale offerta da Vue.js, non da implementare nel nostro sistema. Agisce come un intermediario reattivo tra la logica dell'interfaccia utente (*View*) e i dati del modello (*Model*). Vue si occupa automaticamente di aggiornare la vista ogni volta che i dati cambiano e viceversa, grazie al binding bidirezionale.

### 4.1.3 Tracciamento requisiti

Qui di seguito verrà riportato in una tabella il tracciamento dei requisiti funzionali sulle varie classi del back-end e del front-end.

| Codice | Descrizione | Sezioni |
|---|---|---|
| **RF-1.1** | L'utente deve poter visualizzare gli elementi domanda. | Back-end: 4.1.1.2.1.3, 4.1.1.3.1.3, 4.1.1.4.2.1<br><br>Front-end: 4.1.2.2 |
| **RF-1.1.1** | L'utente deve poter visualizzare un singolo *elemento domanda*$_G$. | Back-end: 4.1.1.2.1.2, 4.1.1.3.1.2, 4.1.1.4.2.1<br><br>Front-end: 4.1.2.2 |
| **RF-1.1.1.1** | L'utente deve poter visualizzare il testo di una domanda. | |
| **RF-1.1.1.2** | L'utente deve poter visualizzare il testo della risposta attesa associata ad una domanda. | |
| **RF-1.2** | L'utente deve poter modificare una domanda. | Back-end: 4.1.1.2.1.5, |
| **RF-1.3** | L'utente deve poter modificare una risposta attesa. | |

| | | |
|---|---|---|
| | | 4.1.1.3.1.5,<br>4.1.1.4.2.1<br><br>Front-end:<br>4.1.2.4 |
| **RF-1.4** | L'utente deve poter eliminare un *elemento domanda*$_G$. | Back-end:<br>4.1.1.2.1.4,<br>4.1.1.3.1.4,<br>4.1.1.4.2.1<br><br>Front-end:<br>4.1.2.2 |
| **RF-1.5** | L'utente deve poter aggiungere un *elemento domanda*$_G$. | Back-end:<br>4.1.1.2.1.1,<br>4.1.1.3.1.1,<br>4.1.1.4.2.1<br><br>Front-end:<br>4.1.2.3 |
| **RF-1.5.1** | L'utente deve poter aggiungere manualmente un *elemento domanda*$_G$. | |
| **RF-1.5.1.1** | L'utente deve poter aggiungere manualmente una domanda. | |
| **RF-1.5.1.2** | L'utente deve poter aggiungere manualmente una risposta attesa. | |
| **RF-1.12** | L'utente deve poter eseguire un test utilizzando gli elementi domanda presenti nel sistema. | Back-end:<br>4.1.1.2.2.1,<br>4.1.1.3.2.1<br><br>Front-end:<br>4.1.2.5 |
| **RF-1.13** | L'utente deve poter visualizzare il risultato del test. | Back-end:<br>4.1.1.2.3.1,<br>4.1.1.3.3.1.1,<br>4.1.1.4.2.2<br><br>Front-end:<br>4.1.2.7 |
| **RF-1.13.1** | L'utente deve poter visualizzare nel risultato la valutazione generale del test. | |
| **RF-1.13.2** | L'utente deve poter visualizzare nel risultato la valutazione di ogni singola domanda. | |
| **RF-1.13.3** | L'utente deve poter visualizzare nel risultato la lista delle domande eseguite nel test ordinate in modo crescente rispetto alla singola valutazione. | |
| **RF-1.13.4** | L'utente deve poter visualizzare nel risultato il nome del set su cui è stato eseguito il test. | |
| **RF-1.13.5** | L'utente deve poter visualizzare nel risultato la data di esecuzione del test. | |
| **RF-1.13.6** | L'utente deve poter visualizzare nel risultato l'*LLM*$_G$ utilizzato nel test. | |
| **RF-1.14** | L'utente deve poter visualizzare il risultato di un test per singolo *elemento domanda*$_G$. | Back-end:<br>4.1.1.2.3.7,<br>4.1.1.3.3.1.5,<br>4.1.1.4.2.2 |
| **RF-1.14.1** | L'utente deve visualizzare l'*elemento domanda*$_G$ nella visualizzazione del risultato per singolo *elemento domanda*$_G$. | |
| **RF-1.14.2** | L'utente deve visualizzare la risposta data dall'*LLM*$_G$ nella visualizzazione del risultato per singolo *elemento domanda*$_G$. | |

| | | |
|---|---|---|
| **RF-1.14.3** | L'utente deve visualizzare il nome delle metriche utilizzate nella visualizzazione del risultato per singolo *elemento domanda*$_G$. | Front-end: 4.1.2.7 |
| **RF-1.14.4** | L'utente deve visualizzare la valutazione delle metriche utilizzate nella visualizzazione del risultato per singolo *elemento domanda*$_G$. | |
| **RF-1.17** | L'utente deve poter visualizzare lo storico dei test eseguiti, ordinati in ordine decrescente rispetto all'ordine di esecuzione. | Back-end: 4.1.1.2.3.4, 4.1.1.3.3.1.3, 4.1.1.4.2.2<br><br>Front-end: 4.1.2.6 |
| **RF-1.17.1** | L'utente deve poter visualizzare all'interno dello storico, un elemento riguardante l'esecuzione di un test. | |
| **RF-1.17.1.1** | L'utente deve poter visualizzare all'interno dello storico, un elemento riguardante l'esecuzione di un test contenente la data di esecuzione del test. | |
| **RF-1.17.1.2** | L'utente deve poter visualizzare all'interno dello storico, un elemento riguardante l'esecuzione di un test contenente lo score generale del test. | |
| **RF-1.17.1.3** | L'utente deve poter visualizzare all'interno dello storico, un elemento riguardante l'esecuzione di un test contenente l'$LLM_G$ utilizzato nel test. | |
| **RF-1.17.1.4** | L'utente deve poter visualizzare all'interno dello storico, un elemento riguardante l'esecuzione di un test contenente il set di domande su cui è stato eseguito il test. | |

## 4.2 Database

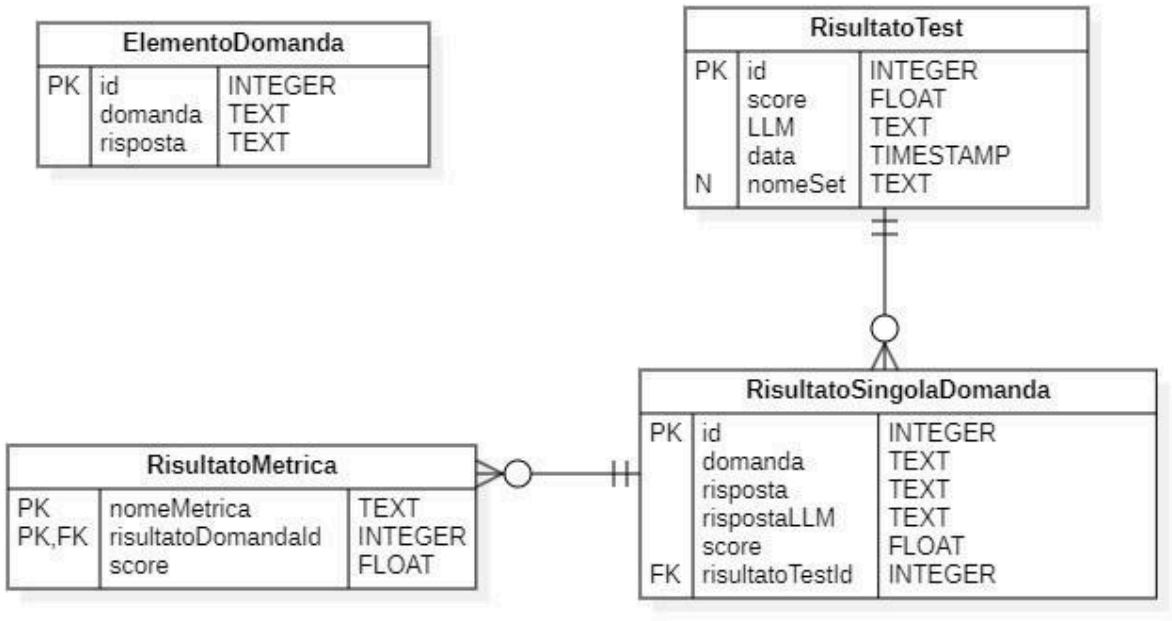



Figura 9: Architettura del database

### 4.2.1 Entità

Il database è composto da 5 entità:

- **ElementoDomanda**: rappresenta una singola domanda con annessa risposta. È composto da:
  - id: identificativo univoco dell'*elemento domanda*$_G$
  - domanda: l' oggetto domanda
  - risposta: l' oggetto risposta attesa
- **Domanda** : rappresenta la domanda. È composto da:
  - testo : il testo della domanda
- **Risposta** : rappresenta la risposta. È composto da:
  - testo : il testo della risposta
- **RisultatoTest**: rappresenta il risultato di un test di valutazione svolto su delle domande. È composto da:
  - id: identificativo univoco del risultato
  - score: punteggio ottenuto dal test
  - *LLM*$_G$: nome dell'*LLM*$_G$ a cui vengono poste le domande
  - data: data in cui è stato svolto il test
  - nomeSet: nome del set di domande utilizzato per il test. Questo campo può essere nullo se il test viene svolto su tutte le domande disponibili
- **RisultatoSingolaDomanda**: rappresenta il risultato ottenuto da una singola domanda all'interno del risultato di un test . È composto da:
  - id: identificativo univoco del risultato
  - idRisultatoTest: identificativo del risultato del test a cui appartiene
  - domanda: testo della domanda testata
  - risposta: testo della risposta testata
  - rispostaLLM: testo della risposta fornita dall'*LLM*$_G$
  - score: punteggio ottenuto dalla domanda
  - metriche : un dizionario di coppie «nome metrica» e «score parziale ottenuto» corrispondente

#### 4.2.2 Query

Le operazioni principali che vengono eseguite sul database sono:

- **Inserimento**:
  - di un *elemento domanda*$_G$
  - di un risultato di un test, con conseguente caricamento di risultati delle singole domande e dei relativi risultati delle singole metriche.
- **Ottenimento**:
  - di un *elemento domanda*$_G$ o di tutti gli elementi domanda presenti.
  - di un risultato di un test o di tutti i risultati di test presenti. In entrambi i casi, con conseguente ottenimento di risultati delle singole domande e dei relativi risultati delle singole metriche.
  - di un risultato di una singola domanda, con conseguente ottenimento dei risultati delle singole metriche.
- **Eliminazione**:
  - di uno o più elementi domanda
- **Aggiornamento**:
  - di domanda o risposta di un determinato *elemento domanda*$_G$

## 4.3 Architettura

### 4.3.1 Architettura di Deployment

L'architettura di deployment utilizzata dall'applicativo sia lato client che lato server è di tipo **monolitico**. Questo tipo di architettura porta diversi vantaggi:

- Semplicità di sviluppo
- Facilità di deployment
- Facilità di testing

### 4.3.2 Architettura Logica

#### 4.3.2.1 Backend

L'architettura logica della componente backend dell'applicativo è di tipo **esagonale**, il quale permette di **isolare** la logica di business e renderla indipendente da eventuali servizi esterni, con i quali andrà a **comunicare** tramite l'uso di **interfacce** ben definite. Questo permette di isolare la business logic del sistema, limitando la possibilità di avere dipendenze non necessarie.

La parte centrale dell'esagono è rappresentata quindi dalla **logica di business**, la quale contiene il dominio del programma.

Le porte, punto focale di comunicazione con l'esterno, sono rappresentate dalle **interfacce**. Ne distinguiamo di due tipi:

- Le **Inbound Ports** (o *Use Case*$_G$) sono quelle utilizzate da attori esterni per comunicare con il sistema. Queste interfacce sono implementate dal sistema stesso tramite un sistema di API. Esse definiscono quindi i casi d'uso e le operazioni implementate dal sistema.
- Le **Outbound Ports** sono invece quelle utilizzate dal sistema stesso per comunicare con attori esterni.

Gli **Adapters** sono invece le classi che vanno ad implementare in maniera concreta le porte definite dalle interfacce. Queste classi fanno quindi da ponte tramite la logica interna e i servizi esterni. Anche qui ne distinguiamo di due tipi:

- Gli **Inbound Adapters** sono quelli che implementano le interfacce Inbound, andando a definire quindi le operazioni che il sistema esegue in base ai casi d'uso.
- Gli **Outbound Adapters** sono quelli che implementano le interfacce Outbound, andando a definire quindi le operazioni che il sistema esegue per comunicare con servizi esterni.

#### 4.3.2.2 Frontend

Per quanto riguarda il frontend è stata invece adottata un'architettura di tipo **MVVM** (Model-View-ViewModel), implementata in modo implicito dal framework **Vue.js**. Questa struttura consente un approccio dichiarativo e reattivo, favorendo una netta separazione tra la logica di business e l'interfaccia utente. Nel dettaglio, il pattern MVVM si articola nei seguenti elementi:

- Model: rappresenta i dati dell'applicazione. In Vue.js, ciò corrisponde allo stato locale dei componenti, eventualmente esteso attraverso strumenti come Vuex o Pinia per la gestione dello stato globale.
- View: è l'interfaccia grafica dell'applicazione, costituita dai template HTML. Grazie al data binding reattivo offerto da Vue.js, la vista si aggiorna automaticamente al variare dei dati sottostanti.
- ViewModel: funge da ponte tra il Model e la View, gestendo la logica di interazione e l'aggiornamento dello stato. In Vue.js, il componente stesso, nonché il motore di Vue.js che

svolge il ruolo di ViewModel, che definisce proprietà, metodi, computed properties e gestori degli eventi.

Questo tipo di architettura, oltre a facilitare lo sviluppo di *Single Page Application*$_G$ (*SPA*$_G$), consente un'elevata modularità del codice. Tale approccio favorisce il riuso dei componenti, la separazione delle responsabilità e una migliore testabilità dell'interfaccia, rendendo l'applicazione più manutenibile ed estensibile nel tempo.

## 4.4 Design Patterns Utilizzati

### 4.4.1 Singleton

Il pattern **Singleton** viene utilizzato per garantire che una classe abbia una sola istanza e fornire un punto di accesso globale a tale istanza. Ciò è utile in casi in cui sia hanno classi diverse che utilizzano un'istanza di un'altra classe in maniera condivisa. Questo pattern è stato utilizzato per gestire la creazione di un'unica istanza di connessione al database.

### 4.4.2 Strategy

Il pattern **Strategy** viene principalmente utilizzato per definire un'interafccia comune che permette l'utilizzo di algoritmi differenti, senza dover modificare le dipendenze delle classi di business. Questo pattern è stato utilizzato per definire le diverse strategie e algoritmi che permettono la valutazione delle domande.

### 4.4.3 Factory Method

Il pattern **Factory Method** viene utilizzato per definire un'interfaccia per la creazione di un oggetto, lasciando alle sottoclassi la responsabilità di implementare la logica di creazione dell'oggetto concreto. Questo pattern è stato utilizzato per la creazione e gestione di dipendenze tramite dependency injection.

### 4.4.4 Decorator

Il pattern **Decorator** viene utilizzato per aggiungere funzionalità a un oggetto dinamicamente, incapsulandolo in un nuovo oggetto che contiene queste funzionalità. Questo pattern è stato utilizzato per esempio per l'«iniezione» delle dipendenze all'interno dei controllers.

### 4.4.5 Dependency injection

Il pattern **Dependency Injection** viene utilizzato per fornire alle classi le loro dipendenze dall'esterno, piuttosto che crearle direttamente al loro interno. Questo approccio migliora la modularità e facilita il testing, permettendo di sostituire facilmente le dipendenze con implementazioni alternative o mock durante i test.
Nel nostro progetto, la dependency injection è stata utilizzata per gestire la configurazione e l'inizializzazione dei servizi principali, come i *repository*$_G$, i servizi di business e i controllers per la comunicazione con servizi esterni.

### 4.4.6 MVVM

Il pattern **MVVM** (Model-View-ViewModel) permette di separare la logica di presentazione dalla logica di business e dall'interfaccia utente. Questo pattern è stato utilizzato per strutturare l'applicazione frontend, consentendo una chiara separazione delle responsabilità tra i componenti e facilitando la manutenzione e l'estensibilità del codice.