Rod2Cod

Manuale Sviluppatore

Il documento ha lo scopo di fornire una **guida** volta ad aiutare i membri del gruppo e gli sviluppatori che dovranno mantenere ed estendere il prodotto. Di seguito sono riportati gli strumenti e le tecnologie utilizzate nel progetto, accompagnati da loro descrizione e metodologie di utilizzo.

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

Informazioni

Versione	2.0
Uso	Esterno
Data	27/04/2025
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

Manuale Sviluppatore

Versione	Data	Descrizione	Autore	Verificatore	Validatore
2.0	2025-04-26				Annalisa Egidi
1.4	2025-04-10	Sezione *LLM*$_G$ Evaluator - Aggiunta sezione allenamento random forest	Alberto Maggion	Filippo Bellon	
1.3	2025-04-10	Sezione Backend-Flask - Aggiunta configurazione di sistema	Filippo Bellon	Annalisa Egidi	
1.2	2025-04-09	Sezioni Backend-Flask e Problemi Noti - Aggiunto Test e sezione Problemi Noti	Filippo Bellon	Alberto Maggion	
1.1	2025-04-07	Sezione Backend-Flask - Aggiunte possibili estensioni	Filippo Bellon	Alberto Maggion	
1.0.0	2025-02-24				Annalisa Egidi
0.3.0	2025-02-13	Verifica documento		Michele Nesler	
0.2.1	2025-01-20	Agginta Sezione Bootstrap, Axios, Backend - Flask, *LLM*$_G$ Evaluator - Python,	Luca Calzetta		
0.2.0	2024-12-19	Verifica documento		Filippo Bellon	
0.1.2	2024-12-13	Aggiunta sezione Docker, React, Postgres *SQL*$_G$	Luca Calzetta		
0.1.1	2024-11-23	Aggiunta sezione Introduzione	Filippo Bellon		
0.1.0	2024-11-22	Verifica documento		Alberto Maggion	
0.0.3	2024-11-22	Struttura del documento	Filippo Bellon		
0.0.2	2024-11-20	*Github*$_G$ Actions	Filippo Bellon		
0.0.1	2024-11-15	Prima stesura	Filippo Bellon Michele Nesler		

Indice

1 Introduzione

1.1 Glossario

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

2 *Gh-Pages*$_G$ e *Jekyll*$_G$

Gh-Pages$_G$ è un servizio offerto da *github*$_G$ che permette di hostare un **sito statico** direttamente da *Github*$_G$. ***Jekyll***$_G$ è un framework consigliato da *Github*$_G$ che permette di sviluppare molto velocemente un sito dinamico con un template preesistente per poi eventualmente modificarlo secondo le proprie esigenze.

2.1 Rilasciare i documenti

Per il rilascio dei documenti, data la presenza di Actions di *Github*$_G$ che vanno a recuperare anche il materiale sul ramo «master» in fase di checkout, non è necessario caricarli nel branch «*gh-pages*$_G$» dedicato alle *Github*$_G$ Pages, ma è sufficiente caricarli nel branch «master»:

1. spostarsi nel branch **master**
2. caricare i **documenti** dentro le apposite **cartelle**

2.1.1 Aggiunta cartella documenti

1. aprire il file «index.md»
2. andare ad aggiungere nel punto desiderato all'interno del file una sezione come quella in foto

```
### Interni
<!--  così vado a prendermi i file pdf in Verbali/Interni -->

{% for pdf in site.static_files %}
{% if pdf.path contains "Verbali/Interni" and pdf.extname == ".pdf" %}

    {% assign data = pdf.name | split: "Verbale" | last | split: "." | first | split: "-" %}

    {% assign giorno =  data[2] %}

    {% assign mese =  data[1] %}

    {% assign anno =  data[0] %}

- [ Verbale {{ giorno }}/{{ mese }}/{{ anno }} ]({{ pdf.path | relative_url }}){:target="_blank"}
{% endif %}
{% endfor %}
```

3. andare a modificare la sezione subito successiva a «contains» (in questo caso rappresentata da «Verbali/Interni») con le cartelle nella quale si trovano i file da mostrare
4. andare a modificare il codice Liquid secondo le esigenze
   Esempio: La sezione `{% assign data = pdf.name | split: "Verbale" | last | split: "." | first | split: "-" %}` scomposta va a:
   - prendere **pdf.name** e **dividerlo** seguendo un marcatore (| split: "Verbale" | con «Verbale» come marcatore)
   - prendere l'**ultima sezione** risultante (| last |) e **dividerla** seguendo un marcatore (| split: "." | con «.» come marcatore)
   - prendere la **prima sezione** risultante (| first |) e **dividerla** seguendo un marcatore (| split: "-" | con «-» come marcatore)

   Per riferimenti più approfonditi: https://shopify.dev/docs/api/liquid.

2.2 *Github*$_G$ Pages

Ci sono 2 modi in cui *Github*$_G$ gestisce il rilascio delle *Github*$_G$ Pages e si possono trovare sotto la sezione Settings >> Pages assieme al link del sito:

1. da un **branch** e **cartella** (ogni volta che la cartella nel branch viene modificata il sito viene aggiornato)
2. attraverso le ***Github*$_G$ Actions** (l'aggiornamento del sito viene gestito attraverso le actions)

2.3 *Jekyll*$_G$

Il framework ***Jekyll***$_G$ consente di applicare template («themes») preesistenti o di propria creazione a dei file di markdown (.md o .markdown) che definiscono i contenuti delle pagine.
I template possono anche essere applicati ad altri template per unirli ed utilizzarli assieme. La documentazione si può trovare all'indirizzo: https://jekyllrb.com/docs/.

2.3.1 Info generali

I template sono definiti in formato **HTML** e **Liquid**. Il secondo è un linguaggio che viene automaticamente compilato da *Jekyll*$_G$ e permette di utilizzare costrutti propri della programmazione simili a markdown che poi andranno automaticamente convertiti e inseriti all'interno del file HTML.
La documentazione la si può trovare a questo indirizzo: https://shopify.dev/docs/api/liquid.

2.3.2 File Markdown

Ogni pagina Markdown ha un header(«**front matter**») che inizia e finisce con «—» (tre trattini consecutivi) in cui, tramite **variabili**, si vanno ad assegnare alla pagina dei parametri come il **layout HTML** da utilizzare ed il **titolo**, ed eventualmente, anche valori ad altre variabili create all'interno del file HTML.
Al di sotto del tag «—» di fine invece, andrà messo tutto ciò che si vorrà inserire all'interno del tag «**{{ content }}**» presente nel file layout HTML.

2.3.3 Cartelle

- **_site**: vi sono presenti i file che genera *jekyll*$_G$ dopo la compilazione, generalmente non ci si mette mano
- **_layouts**: vi sono i templates/layouts
- **_assets**: vi sono i file a cui si può accedere tramite le keyword liquid site.assets che identificano un array con tutti i file presenti che non hanno estensione .md o .html
- **_drafts**: vi sono le pagine bozze, non vengono visualizzate, servono solo per lo sviluppo
- **_post**: vi sono i post di un blog oppure le pagine più in generale che vengono create nel tempo in modo iterativo
- **_includes**: contiene file HTML o Markdown riutilizzabili in più pagine, in particolare utilizzato per file header o footer aggiuntivi
- **_data**: contiene file dati in formato YAML, JSON o CSV, utilizzabili per popolare contenuti strutturati
- **_sass**: conserva file SASS/SCSS per la gestione degli stili, utili per creare un CSS modulare

2.3.4 File di configurazione

- **_config.yml**: file di configurazione principale di *Jekyll*$_G$, dove vengono definiti **parametri globali** del sito (come **titolo**, **URL**, permalinks, e impostazioni per i **plugin**).
- **Gemfile**: specifica le gems (pacchetti) di *Ruby*$_G$ necessarie al progetto *Jekyll*$_G$, permettendo la gestione delle **dipendenze** tramite Bundler, come *Jekyll*$_G$ stesso e altri plugin.

2.3.5 Installare *Jekyll*$_G$ e Bundler e creare un nuovo sito

Seguire le istruzioni riportate qui di seguito: https://jekyllrb.com/docs/#instructions

2.3.5.1 Lavorare su un sito già esistente

1. eseguire `git clone indirizzo_repo`
2. spostarsi nella cartella della *repo*$_G$
3. eseguire `bundle install`
4. fare le modifiche desiderate
5. eseguire `bundle exec jekyll`$_G$ `serve`

2.3.6 Applicare un template esistente

1. scegliere un template da (https://jekyllrb.com/resources/) oppure cercare il template nel sito https://rubygems.org/ cercando *jekyll*$_G$-theme
2. nel Gemfile:
    1. sostituire `gem \"minima\", \"~> 2.5\"` con quella indicata (generalmente) nel readme del template scelto
    2. eseguire `bundle install`
    3. andare a fare le modifiche desiderate
    4. eseguire `bundle exec jekyll`$_G$ `serve`
3. nel file _config.yml:
    - eliminare o eventualmente sostituire `remote_theme: pages-themes/slate\@v0.2.0` con il codice indicato

Fare attenzione che i templates nella cartella layouts abbiano lo stesso nome oppure modificarli in un secondo momento e aggiungere quelli mancanti, altrimenti tutte le pagine che li usano verranno nascoste dal sito.

2.3.7 Hostare il sito su *github*$_G$ pages

Attualmente l'hosting del sito è gestito automaticamente attraverso ***Github***$_G$ **Actions** che vanno ad automatizzare sia il **recupero** dei file **pdf** da inserire nel sito, sia il **caricamento** del **sito** su *Github*$_G$ Pages dal branch *gh-pages*$_G$ della *repository*$_G$.

2.3.8 Comandi utili

1. `bundle exec jekyll`$_G$ `serve` : **compila** il codice e va a creare un sito statico all'interno dell'**ambiente locale**
2. `bundle install` : **installa** i pacchetti necessari, utilizzato quando vi sono cambiamenti nelle configurazioni (templates, file config, Gemfile,...)

3 *Github*$_G$ Actions

Github$_G$ Actions è una piattaforma proposta da *Github*$_G$ per l'utilizzo di tecnologie quali **Continuous Integration** (CI) e **Continuous Delivery** (CD). Funziona tramite l'esecuzione uno o vari **workflows** che andranno ad avviarsi solo al verificarsi di determinate situazioni e/o **eventi**.

Il tutto funziona tramite l'utilizzo di **file .yml** che permettono di operare tramite il processo di «***Infrastructure as code***$_G$».

3.1 Composizione files *Jekyll*$_G$

Qui di seguito viene proposta una scomposizione dei files utilizzati per la creazione e la successiva impostazione di una pagina web su ***Github***$_G$ **Pages** tramite il framework ***Jekyll***$_G$, per andare ad analizzare la sua struttura in caso di eventuali modifiche.
I due files (**jekyll_gh-pages.yml** e **jekyll_gh-pages.yml**) sono indentici per la maggior parte del codice e si distinguono solo per il branch sul quale vanno a verificare l'evento. Di seguito la scomposizione:

- `name: Deploy Jekyll`$_G$ `site to Pages`

  Qui andiamo a dare un **nome** alla Action.

- ```
  on:
    push:
      branches:
        - master
      paths:
        - 'Verbali/**'
        - 'Candidatura/**'
  ```

  per il ramo **master**, e

  ```
  on:
    push:
      branches:
        - gh-pages
  ```
  $_G$

  per il ramo ***gh-pages***$_G$.

  Questa è la parte che gestisce l'**evento** che attiverà la Action.
  Qui stiamo ad indicare che la Action verrà attivata all'avvenire di un processo di **push** sul ramo ***gh-pages***$_G$, o sul ramo **master** solo però se si vanno a modificare **specifiche cartelle/files**.

- `workflow_dispatch:`

  Permettiamo al workflow di poter essere avviato anche **manualmente**.

- ```
  permissions:
    contents: read
    pages: write
    id-token: write
  ```

  Questo permette di impostare i **permessi** per il *Github*$_G$ **Token** che ci servirà per fare il deploy della pagina web in *Github*$_G$ Pages.

- ```
  concurrency:
    group: "pages"
    cancel-in-progress: false
  ```
  ```

Questa parte consente solo **un'esecuzione** alla volta, in particolare sul ramo *gh-pages*$_G$, e **previene** la possibilità di **arrestare** un'esecuzione in corso.

- `jobs:`

  Questo indica tutti i **processi** che la Action prevede vengano eseguiti.

- ```
  build:
    runs-on: ubuntu-latest

    steps:
  ```

  Il primo processo che verrà avviato sarà quello di **build**, ovvero costruzione della pagina, o meglio di un **artifact** della pagina. Specifichiamo che vogliamo lavorare su una macchina **linux**, e **indichiamo** i vari **steps/sottoprocessi**:

- ```
  - name: Checkout
      uses: actions/checkout@v4
      with:
        fetch-depth: 0
  ```

  Il primo step del lavoro di **build**, è quello di andare a recuperare e rendere disponibili alla Action il materiale su cui andare a lavorare. In questo caso, andiamo quindi a impostare un'operazione di **Checkout** sul ramo *gh-pages*$_G$(sottinteso in quanto la Action è situata là). Per fare ciò, utilizziamo un'altra **Action** già presente su *Github*$_G$, e indichiamo tra i parametri il fatto che vogliamo che si vadano a **recuperare tutti i files** presenti su quel ramo.

- ```
  - name: Checkout pdf branch
      run: |
          git fetch origin master
          git checkout origin/master -- Verbali Candidatura
  ```

  Stessa cosa andiamo a fare per il ramo **master**, in quanto contiene i nostri files riguardanti la **documentazione**. Qui però, tramite un processo diverso, andiamo a recuperare solo **determinate cartelle**.

- ```
  - name: Setup Ruby_G
    uses: ruby_G/setup-ruby_G@8575951200e472d5f2d95c625da0c7bec8217c42 # v1.161.0
    with:
      ruby_G-version: '3.3' # Not needed with a .ruby_G-version file
      bundler-cache: true # runs 'bundle install' and caches installed gems
  automatically
      cache-version: 0 # Increment this number if you need to re-download cached
  gems
  ```

  Questa parte definisce la **preparazione** dell'**ambiente** di creazione dell'**artifact** della pagina web. Qui andiamo ad **installare *Ruby*$_G$** ed i pacchetti da noi utilizzati, in particolare ***Jekyll*$_G$**.

- ```
  - name: Setup Pages
      id: pages
      uses: actions/configure-pages@v5
  ```

  Qui andiamo a **prepare** l'**ambiente** per le ***Github*$_G$ Pages** ed a **estrarre** vari **metadati** dalla pagina.

- ```
  - name: Build with Jekyll_G
      run: bundle exec jekyll_G build --baseurl
  "${{ steps.pages.outputs.base_path }}"
  ```

```
    env:
      JEKYLL_ENV: development
```

Con questo step, usiamo ***Jekyll***$_G$ con i nostri files, per andare a **costruire** l'**artifact** della pagina web all'interno della cartella *./_site*. La variabile d'ambiente definita all'interno, ci permette di utilizzare **plugins** per metadati, in particolare quelli di *Github*$_G$.

- ```
  - name: Upload artifact
    uses: actions/upload-pages-artifact@v3
  ```

  Andiamo infine a **caricare** l'**artifact** del sito così creato, all'interno della nostra ***repository***$_G$, in particolare nella cartella *./_site*.

  Con quest'ultima parte termina il processo di build.

- ```
  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    needs: build
    steps:
  ```

  Iniziamo così il secondo processo, quello di **deploy**, che ci permetterà di **impostare** così l'artifact precedentemente caricato, come **pagina web** tramite *Github*$_G$ Pages.
  Specifichiamo nuovamente che vogliamo lavorare su una macchina **linux**. Andiamo poi a definire **variabili d'ambiente** utili agli **steps/sottoprocessi**, e imponiamo l'**esecuzione** di questo processo **solo** in caso il precedente, ossia quello di **build**, sia terminato con **successo**.
  Indichiamo quindi i vari **steps/sottoprocessi**:

- ```
  - name: Deploy to GitHub Pages
    id: deployment
    uses: actions/deploy-pages@v4
  ```

  Andiamo a definire la **Action** già presente in *Github*$_G$ che utilizzeremo per il **deploy** della **pagina web** su *Github*$_G$ Pages.

4 Docker

Docker è una piattaforma di containerizzazione che consente di creare, distribuire e gestire applicazioni in ambienti isolati chiamati container. Ogni container include tutto ciò che serve per eseguire l'applicazione, come il codice, il runtime, le librerie e le configurazioni necessarie. Questo approccio garantisce che l'applicazione si comporti in modo identico su qualsiasi macchina o ambiente di lavoro.

4.1 Vantaggi principali

- **Portabilità**: I container possono essere eseguiti su qualsiasi sistema che supporti Docker, indipendentemente dall'ambiente sottostante.
- **Isolamento**: Ogni container opera in modo isolato, evitando conflitti tra le dipendenze delle applicazioni.
- **Efficienza**: I container utilizzano meno risorse rispetto alle macchine virtuali poiché condividono lo stesso kernel del sistema operativo.
- **Scalabilità**: Docker consente di scalare facilmente applicazioni distribuite, garantendo un uso efficiente delle risorse.
  ```

4.2 Metodi di Installazione di Docker

4.2.1 Installazione su Linux

1. Aggiorna il sistema:

   ```
   sudo apt update && sudo apt upgrade -y
   ```

2. Installa Docker:

   ```
   sudo apt install docker.io
   ```

3. Installa Docker Compose:

   ```
   sudo apt install docker-compose
   ```

4. Verifica l'installazione:

   ```
   docker --version
   docker-compose --version
   ```

4.2.2 Installazione su macOS

1. Scarica Docker Desktop:
   - Visita il sito ufficiale: [Docker Desktop](https://www.docker.com/products/docker-desktop).
2. Installa Docker:
   - Apri il file `.dmg` scaricato e segui le istruzioni a schermo.
3. Avvia Docker Desktop:
   - Trova Docker nelle Applicazioni e avvialo.
4. Verifica l'installazione:

   ```
   docker --version
   ```

4.2.3 Installazione su Windows

1. Scarica Docker Desktop:
   - Vai sul sito di Docker Desktop (https://www.docker.com/products/docker-desktop) e scarica la versione per Windows.
2. Installa Docker:
   - Esegui il file `.exe` scaricato e segui il processo guidato di installazione.
3. Abilita *WSL*$_G$ 2 (*Windows Subsystem for Linux*$_G$):
   - Durante l'installazione, assicurati di abilitare *WSL*$_G$ 2, necessario per eseguire Docker su Windows.
4. Verifica l'installazione eseguendo sul terminale il seguente comando:

   ```
   docker --version
   ```

5 Frontend - Vue JS Framework

Vue.js è un framework JavaScript progressivo sviluppato per creare interfacce utente dinamiche e interattive. È particolarmente adatto allo sviluppo di applicazioni web a pagina singola (*SPA*$_G$).

5.1 Vantaggi principali

- **Component-Based Architecture**: Vue utilizza componenti modulari e riutilizzabili per costruire interfacce utente.
- ***Virtual DOM***$_G$: Aggiorna solo le parti necessarie del *DOM*$_G$ reale, migliorando le prestazioni.

- **Ecosistema**: Offre un vasto ecosistema di librerie e strumenti per semplificare lo sviluppo.

5.2 Installazione e Avvio del Frontend

Requisiti:

- Assicurati che **Docker** sia installato e configurato correttamente (vedi punto 4.2).
- Installa **Node.js** e *npm*$_G$ seguendo questo link: https://nodejs.org/en/download/package-manager

5.3 Utilizzo con Docker

5.3.1 Avvio dell'applicazione

1. Posizionati nella directory `docker-vue`.
2. Esegui:

```
docker-compose up --build -d
```

3. L'applicazione sarà disponibile all'indirizzo: http://localhost:5173

5.3.2 Esecuzione dei Test

1. Avvia i test unitari:

```
docker exec -it frontend-vue npm_G run test
```

2. Genera il report di coverage:

```
docker exec -it frontend-vue npm_G run test:coverage
```

5.3.3 Arresto dell'applicazione

- Ferma i container:

```
docker-compose down
```

5.4 Utilizzo con Node.js

5.4.1 Avvio dell'applicazione

1. Posizionati nella directory `vue_app`.
2. Esegui:

```
npm_G run dev
```

3. L'app sarà disponibile all'indirizzo: http://localhost:5173

5.4.2 Esecuzione dei Test

1. Esegui i test unitari:

```
npm_G run test
```

2. Genera il report di coverage:

```
npm_G run test:coverage
```

5.4.3 Arresto dell'applicazione

- Interrompi il processo:

```
CTRL + C
```

5.4.4 Risoluzione dei Problemi

- Se l'applicazione non si avvia:
  - Controlla che tutte le *dipendenze* siano installate correttamente.

    - Verifica la *versione* di Node.js (consigliata: LTS_G).
- Se ci sono errori durante la fase di build:
    - Controlla la *configurazione* del file `docker-compose.yml` o verifica eventuali moduli mancanti in Node.js.
- Se ci sono errori di rete:
    - Assicurati che nessun altro servizio stia utilizzando la *porta* `5173`.

5.5 Bootstrap

Bootstrap è una libreria CSS progettata per semplificare lo sviluppo di interfacce utente responsive e moderne. Offre un'ampia gamma di componenti predefiniti, stili CSS e JavaScript per creare layout uniformi e compatibili con diversi dispositivi.

La versione più recente di Bootstrap si basa su Flexbox e CSS Grid, offrendo una maggiore flessibilità nel design dei layout.

Abbiamo deciso di integrarla nello sviluppo della nostra applicazione Vue per accelerare lo sviluppo front-end senza la necessità di scrivere CSS complessi da zero. Utilizzando Node come gestore dei pacchetti, l'installazione avviene tramite npm_G.

5.6 Axios

Axios è una libreria JavaScript per fare richieste HTTP, basata su Promises. È utilizzata per comunicare con API, recuperare dati da server remoti e inviare informazioni.

Axios supporta tutte le principali operazioni HTTP, come GET, POST, PUT, DELETE, e fornisce funzionalità avanzate come intercettori, configurazioni globali e gestione automatica delle intestazioni.

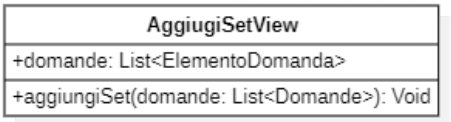
Abbiamo scelto questa libreria in quanto offre molta più flessibilità per la comunicazione con le API rispetto al semplice `fetch` fornito da JavaScript base.

5.7 Possibili Estensioni

Le estensioni che sono state pensate e progettate per il frontend riguardano le viste che gestiscono le funzionalità associate ai Set.

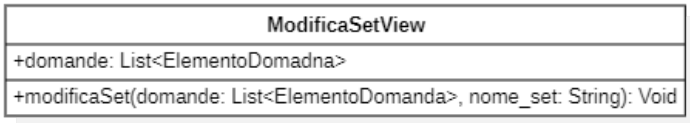
5.7.1 Vista Per l'Aggiunta di un Set

Questa vista si occuperà tramite model e viewmodel di Vue di effettuare l'aggiunta di un Set.



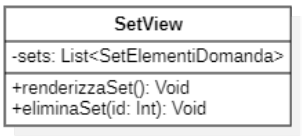
5.7.2 Vista Per la modifica di un Set

Questa vista si occuperà tramite model e viewmodel di Vue di effettuare la modifica di un Set.



5.7.3 Vista Per la visualizzazione di tutti i Set

Questa vista si occuperà di visualizzare il nome di tutti i Set presenti nel sistema.



6 Database *SQL*$_G$ - Postgres

PostgreSQL è un sistema di gestione di database relazionale avanzato e open source. Viene spesso utilizzato per applicazioni che richiedono robustezza, scalabilità e conformità agli standard *SQL*$_G$. ***pgAdmin***$_G$ è uno strumento grafico per la gestione dei database PostgreSQL.

6.1 Requisiti

- Docker e Docker Compose installati sul sistema (vedi punto 4.2).
- Permessi di amministratore per configurare directory locali e impostare volumi.

6.2 Avvio del Sistema

1. Posizionati nella directory del progetto contenente il file `docker-compose.yml`.
2. Esegui il comando per avviare i container in background:

   ```
   docker compose up -d
   ```
3. Il database PostgreSQL esporrà delle API alla porta `5432`.
4. *pgAdmin*$_G$ sarà accessibile all'indirizzo: http://localhost:5050.

6.3 Configurazione di *pgAdmin*$_G$

1. Accedi a *pgAdmin*$_G$ tramite il browser (http://localhost:5050).
2. Usa le credenziali predefinite configurate nel file `docker-compose.yml` (email `admin@admin.com` e password `admin`).
3. Una volta entrato dovrebbe esserci già il database impostato e visibile. In caso contrario configura un nuovo server per connetterti al database PostgreSQL:
   - Nome PostgreSQL
   - Host `postgres`
   - Porta `5432`
   - Utente quello configurato (`admin`)
   - Password quella configurata nel file `docker-compose.yml`.

6.4 Gestione del Database

- Per accedere alla shell interattiva del database:

  ```
  docker exec -it <nome_container_postgres> psql -U <utente>
  ```

6.5 Arresto del Sistema

- Per arrestare e rimuovere i container:

  ```
  docker-compose down
  ```

6.6 Note Finali

- Assicurati che le directory `db_data` e `pgadmin_data` siano correttamente configurate per garantire la persistenza dei dati.

- Se riscontri problemi, controlla i log dei container con:

```
docker logs <nome_container>
```

7 Backend - Flask

Flask è un framework leggero e super flessibile per Python, utile per creare applicazioni web e API. È molto usato per progetti piccoli e medi, ma con le giuste estensioni può gestire anche cose più grandi. Le sue API sono perfette per creare sistemi RESTful e far comunicare il backend con frontend o altri servizi. Motivo per cui abbiamo scelto di utilizzarlo nel progetto.

7.1 Requisiti

- Docker e Docker Compose installati sul sistema (vedi punto 4.2).
- Permessi di amministratore per configurare directory locali e impostare volumi.

7.2 Configurazione del Sistema

Alcune parti del sistema richiedono una determinata configurazione, modificale tramite il file `config.ini` presente nella cartella del progetto. All'interno di questo file si possono configurare le seguenti variabili:

- Sezione llm_G
  - `url`: url per l'invio delle richieste al LLM_G.
  - `nome`: nome del modello LLM_G da utilizzare.
- Sezione `evaluator`
  - `model`: percorso del file .joblib contenente il modello di valutazione. Questo percorso è relativo a partire dalla cartella del progetto.
- Sezione `database`
  - `uri`: URI per la connessione al database Postgres.

7.3 Avvio del Sistema

1. Posizionati nella directory del progetto contenente il file docker-compose.yml.
2. Esegui il comando per avviare i container in background:

```
docker compose up -d
```

3. L'API Flask sarà disponibile all'indirizzo: http://localhost:5000.

7.4 Configurazione dell'API

1. Una volta avviato il sistema, l'API Flask sarà accessibile tramite richieste HTTP all'indirizzo: http://localhost:5000.
2. Verifica che l'endpoint di default risponda correttamente:

```
curl http://localhost:5000
```

3. Utilizzeremo questo endpoint anche nel progetto, per cui tutte le richieste verranno effettuate da axios (vedi punto 5.6) verso questo endpoint, utilizzando poi le varie route, per sfruttare servizi diversi.

7.5 Gestione del Progetto

- Eseguire comandi all'interno del container Flask:

```
docker exec -it <nome_container_flask> /bin/sh
```

- Controllare i log delle richieste in tempo reale:

```
docker logs -f <nome_container_flask>
```

7.6 Arresto del Sistema

Per arrestare e rimuovere i container:

```
docker-compose down
```

7.7 Test

Per il test dei componenti di backend, viene fornito ed utilizzato lo strumento **pytest** e di seguito viene presentata una scaletta di esempio per avviarne l'esecuzione e testarne quindi le classi.

1. Prima di tutto creare un **ambiente virtuale** e installare le **dipendenze** necessarie(tra queste sarà presente anche pytest):

   ```
   python3 -m venv .venv
   source .venv/bin/activate
   pip_G install -r requirements.txt
   ```

2. Aggiungere quindi alla **variabile d'ambiente** `PYTHONPATH` il percorso della cartella contenente le sottocartelle **src** e **test**:

   ```
   export PYTHONPATH=$PYTHONPATH:/path/alla/cartella/padre
   ```

3. Posizionarsi quindi all'interno della cartella padre contenente **src** e **test**

4. **Avviare** il test con il seguente comando:

   ```
   pytest -v --cov-report=term-missing --cov=./src -n 2 ./test
   ```

   - Il flag `-v` permette di visualizzare i **dettagli** dei test eseguiti.
   - Il flag `--cov-report=term-missing` permette di visualizzare la **copertura** dei test eseguiti a terminale.
   - Il flag `--cov=./src` permette di specificare la **cartella** da **testare**.
   - Il flag `-n 2` permette di eseguire i test in **parallelo** su 2 core.
   - Il flag `./test` permette di specificare la **cartella** contenente i **test** da eseguire.

5. In caso di terminazione dei test senza errori, apparira a terminale una tabella indicante la copertura dei test effettuati.

7.7.0.1 Riferimenti

Per eventuali **approndimentimenti** riguardanti l'utilizzo di pytest, è possibile consultare le seguenti documentazioni:

- **Pytest** (https://docs.pytest.org/en/latest/)
- **Pytest-cov** (https://pytest-cov.readthedocs.io/en/latest/)
- **Pytest-xdist** (https://pytest-xdist.readthedocs.io/en/stable/).

7.8 Note Finali

- Per testare endpoint più complessi, utilizza strumenti come Postman o cURL.
- In caso di problemi:
  - Controlla i log del container Flask(vedi punto 7.4)
  - Assicurati che le porte configurate nel file docker-compose.yml siano disponibili.

7.9 Possibili Estensioni

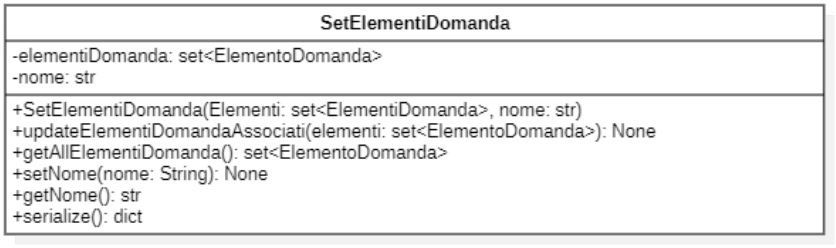
Qui di seguito sono elencate alcune possibili estensioni per la sezione backend del progetto.

7.9.1 Set di Elementi Domanda

Come descritto all'interno della Specifica Tecnica, il dominio delle domande presenti nel sistema viene rappresentato dall'oggetto **ElementoDomanda**, composto da **Domanda** e **Risposta**. Al

momento dell'esecuzione del test però, si vanno a ricavare tutti gli Elementi Domanda presenti nel sistema, senza la possibilità di lavorare solo su alcuni di questi. La soluzione potrebbe essere quella di implementare un'entità **SetElementiDomanda**, in modo da poter raggruppare alcuni Elementi Domanda in base ad un certo criterio, ed andare quindi ad eseguire il test solo su questi.

7.9.1.1 Entità Set Elementi Domanda



7.9.1.1.1 Attributi :

- elementiDomanda : set di oggetti di tipo `ElementoDomanda` che rappresentano gli elementi domanda appartenenti al set.
- nome : nome identificativo del set.

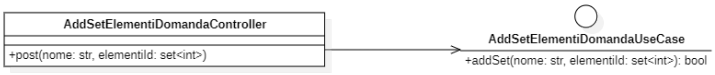
7.9.1.1.2 Metodi :

- updateElementiDomandaAssociati : viene passato come parametro il set degli elementi domanda che si vuole siano associati al set. Tutti gli elementi domanda precedentemente associati vengono dissociati e sostituiti con i nuovi.
- getAllElementiDomanda : restituisce il set di elementi domanda associati al set.
- setNome : una stringa contenente il nuovo nome viene passata come parametro e il nome del set viene sostituito con essa.
- getNome : restituisce la stringa contenente il nome del set.
- getSize : restituisce il numero di elementi domanda associati al set.

7.9.1.2 Controllers

I **Controllers** si occuperanno di fornire tramite **API**, degli endpoint per gestire le richieste HTTP. Per portare a termine la richiesta faranno quindi uso di uno o più **UseCase** definiti precedentemente, ed implementati nei **Services** corrispondenti. Di seguito vengono quindi elencate le possibili strutture dei vari Controllers.

Aggiunta singola



Questo controller si occuperà di esporre un endpoint per l'aggiunta di un Set di Elementi Domanda, prendendo in input, il nome che si vuole assegnare al Set, e gli id degli Elementi Domanda che si vogliono aggiungere a quest'ultimo. Si andrà quindi poi a creare il nuovo Set tramite lo **UseCase** e il **Service** corrispondente.

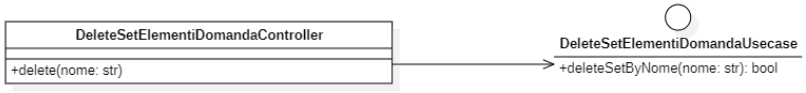
Ottenimento singolo

Questo controller si occuperà di esporre un endpoint per l'ottenimento di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole ottenere. Si andrà quindi a recuperare il Set tramite lo **UseCase** e il **Service** corrispondente.

Ottenimento collettivo

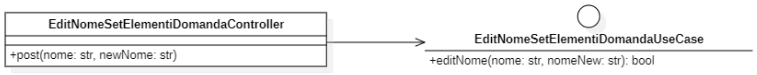
Questo controller si occuperà di esporre un endpoint per l'ottenimento di tutti i Set di Elementi Domanda presenti nel sistema. Si andranno quindi a recuperare i vari Set tramite lo **UseCase** e il **Service** corrispondente.

Eliminazione singola



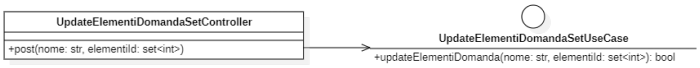
Questo controller si occuperà di esporre un endpoint per l'eliminazione di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole eliminare. Si andrà quindi a rimuovere il Set tramite lo **UseCase** e il **Service** corrispondente.

Modifica nome



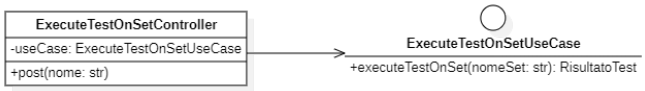
Questo controller si occuperà di esporre un endpoint per la modifica del nome di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole modificare e il nuovo nome che si vuole assegnare. Si andrà quindi a modificare il nome del Set tramite lo **UseCase** e il **Service** corrispondente.

Aggiornamento Elementi Domanda associati



Questo controller si occuperà di esporre un endpoint per l'aggiornamento degli Elementi Domanda associati ad un Set, prendendo in input, il nome del Set che si vuole modificare e gli id degli Elementi Domanda che si vogliono sostituire a quelli attualmente assegnati. Si andrà quindi a modificare gli Elementi Domanda associati al Set tramite lo **UseCase** e il **Service** corrispondente.

Esecuzione Test

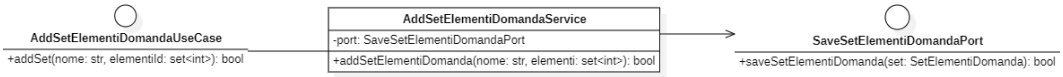


Questo controller si occuperà di esporre un endpoint per l'esecuzione del test su un Set di Elementi Domanda, prendendo in input, il nome del Set su cui si vuole eseguire il test. Si andrà quindi a recuperare il Set tramite lo **UseCase** e il **Service** corrispondente, e si andrà a generare il test.

7.9.1.3 Services

I **Services** si occuperanno di implementare gli **UseCase** e realizzare quindi la logica di business. Per portare a termine la richiesta faranno quindi uso di **Porte**, le quali veranno implementate nel o negli **Adapters** corrispondenti.

Aggiunta singola

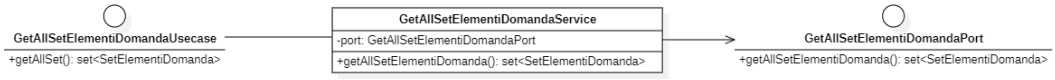


Questo **Service** si occuperà di eseguire la logica per l'aggiunta di un Set di Elementi Domanda, prendendo in input, il nome che si vuole assegnare al Set, e gli id degli Elementi Domanda che si vogliono aggiungere a quest'ultimo. Si andrà quindi a creare il nuovo Set tramite la **Porta** e l'**Adapter** corrispondente.

Ottenimento singolo

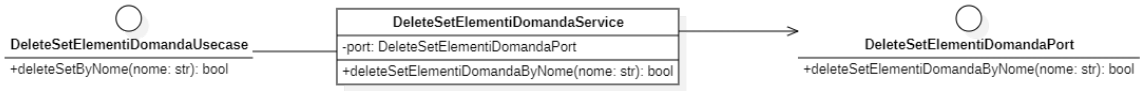
Questo **Service** si occuperà di eseguire la logica per l'ottenimento di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole ottenere. Si andrà quindi a recuperare il Set tramite la **Porta** e l'**Adapter** corrispondente.

Ottenimento collettivo



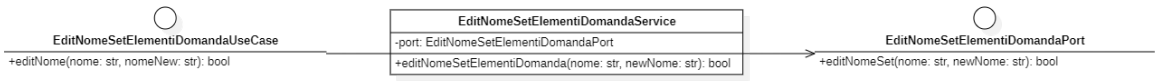
Questo **Service** si occuperà di eseguire la logica per l'ottenimento di tutti i Set di Elementi Domanda presenti nel sistema. Si andranno quindi a recuperare i vari Set tramite la **Porta** e l'**Adapter** corrispondente.

Eliminazione singola



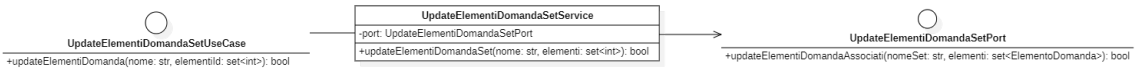
Questo **Service** si occuperà di eseguire la logica per l'eliminazione di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole eliminare. Si andrà quindi a rimuovere il Set tramite la **Porta** e l'**Adapter** corrispondente.

Modifica nome



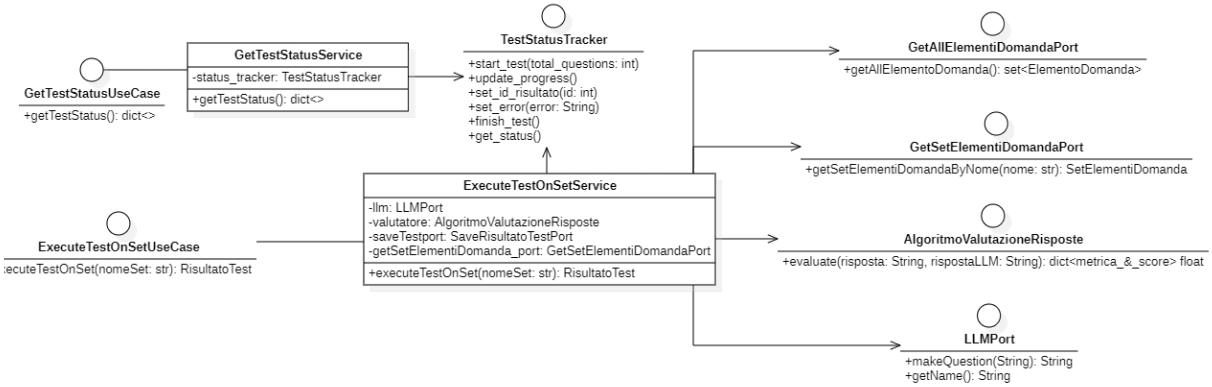
Questo **Service** si occuperà di eseguire la logica per la modifica del nome di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole modificare e il nuovo nome che si vuole assegnare. Si andrà quindi a modificare il nome del Set tramite la **Porta** e l'**Adapter** corrispondente.

Aggiornamento Elementi Domanda associati



Questo **Service** si occuperà di eseguire la logica per l'aggiornamento degli Elementi Domanda associati ad un Set, prendendo in input, il nome del Set che si vuole modificare e gli id degli Elementi Domanda che si vogliono sostituire a quelli attualmente assegnati. Si andrà quindi a modificare gli Elementi Domanda associati al Set tramite la **Porta** e l'**Adapter** corrispondente.

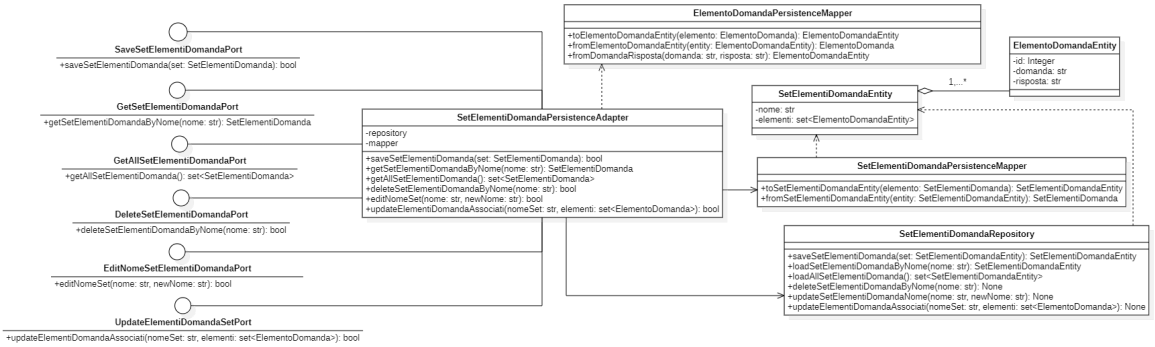
Esecuzione Test



Questo **Service** si occuperà di eseguire la logica per l'esecuzione del test su un Set di Elementi Domanda, prendendo in input, il nome del Set su cui si vuole eseguire il test. Si andrà quindi a recuperare il Set e le relative domande tramite la **Porta** corrispondente. Successivamente si potrà eseguire il test tramite le apposite **Porte** fornite e precedentemente. Infine, per l'operazione di salvataggio si farà uso della **Porta** e dell'**Adapter** corrispondente già presenti.

7.9.1.4 Adapter

L'**Adapter** si occuperà di implementare le **Porte** definite precedentemente, e di esporre quindi i metodi per la comunicazione con il database. Per portare a termine la richiesta farà quindi uso di uno o più ***Repository***$_G$, **Mappers** ed **Entity**.



L'**Adapter** andrà quindi ad implementare le **Porte** precedentemente definite, ed a utilizzare la rispettiva ***Repository***$_G$ per andare a comunicare con il database. Per portare a termine la richiesta farà quindi uso di uno o più **Mappers** ed **Entity**. Le loro operazioni riguarderanno la possibilità di:

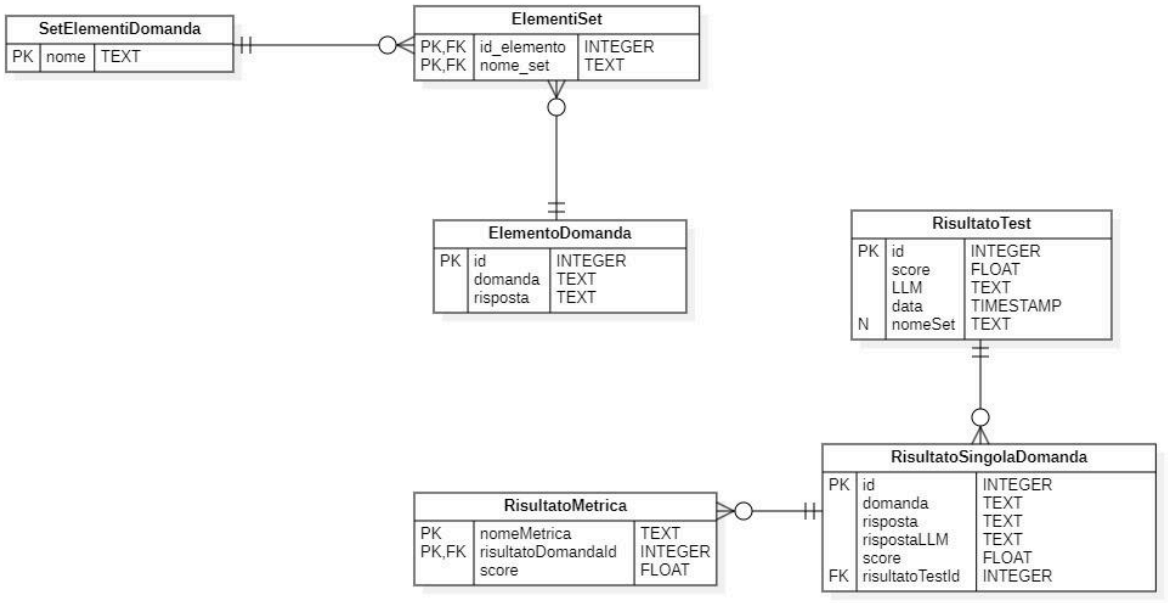
- **Creare** un nuovo Set di Elementi Domanda, prendendo in input, il nome che si vuole assegnare al Set, e gli id degli Elementi Domanda che si vogliono aggiungere a quest'ultimo.

- **Recuperare** un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole ottenere.
- **Recuperare** tutti i Set di Elementi Domanda presenti nel sistema.
- **Rimuovere** un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole eliminare.
- **Modificare** il nome di un Set di Elementi Domanda, prendendo in input, il nome del Set che si vuole modificare e il nuovo nome che si vuole assegnare.
- **Aggiornare** gli Elementi Domanda associati ad un Set, prendendo in input, il nome del Set che si vuole modificare e gli id degli Elementi Domanda che si vogliono sostituire a quelli attualmente assegnati.

Il **Mapper** sarà quindi sempre presente e si occuperà di «tradurre» le **Entity** del database in elementi del dominio quali **SetElementiDomanda** ed **Elementi Domanda**, e viceversa.

7.9.2 Database

La struttura del database subirà perciò alcuni cambiamenti, simili a quelli elencati in seguito.



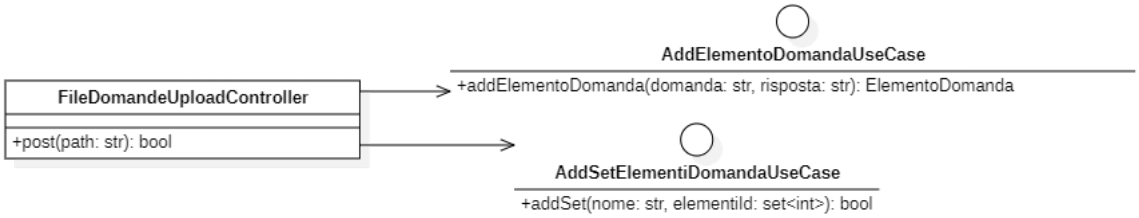
7.9.3 Caricamento da File

Attualmente gli unici modi disponibili per **inserire** un ***Elemento Domanda***$_G$ o il **Risultato** di un **Test** sono, nel primo caso attraverso l'**inserimento manuale** di domanda e risposta e l'invio di queste ultime alle API. Nel secondo caso invece il Risultato del Test viene **generato automaticamente** dal sistema. Una possibile soluzione potrebbe essere quella di permettere quindi l'inserimento sia di uno o più Elementi Domanda eventualmente raggruppati all'interno di un Set, che di un Risultato di Test, tramite il **caricamento** di un **file**.

7.9.3.1 Controllers

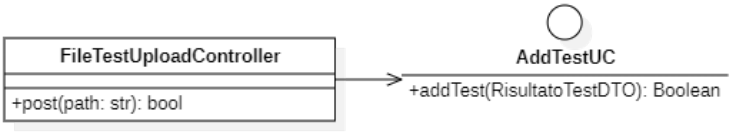
I **Controllers** si occuperanno di fornire tramite **API**, degli endpoint per gestire le richieste HTTP. Per portare a termine la richiesta faranno quindi uso di uno o più **UseCase** definiti precedentemente, ed implementati nei **Services** corrispondenti. Di seguito vengono quindi elencate le possibili strutture dei vari Controllers.

Caricamento Elementi Domanda o Set



Questo controller si occuperà di esporre un endpoint per il caricamento di uno o più Elementi Domanda, o di un Set di Elementi Domanda, prendendo in input, il file che contiene gli Elementi Domanda o il Set. Si andrà quindi ad utilizzare gli **UseCase** e i **Service** corrispondenti già presenti, per andare a creare prima gli Elementi Domanda e poi l'eventuale Set.

Caricamento Risultato Test

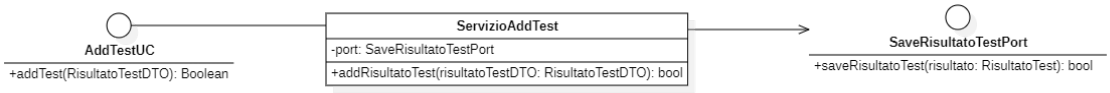


Questo controller si occuperà di esporre un endpoint per il caricamento del Risultato di un Test, prendendo in input, il file che contiene il Risultato. Successivamente andrà quindi a fornire, tramite specifici DTO intermedi, i dati necessari allo **UseCase** e al **Service** corrispondente, per andare quindi ad inserire il nuovo Risultato.

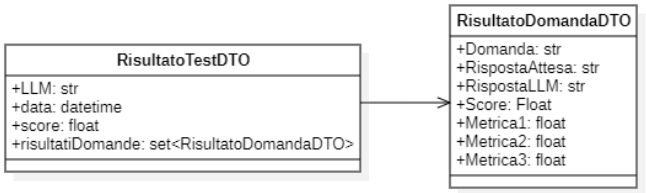
7.9.3.2 Services

I **Services** si occuperanno di implementare gli **UseCase** e realizzare quindi la logica di business. Per portare a termine la richiesta faranno quindi uso di **Porte**, le quali verranno implementate nel o negli **Adapters** corrispondenti. Di seguito vengono quindi elencate le possibili strutture dei vari Services.

Caricamento Risultato Test



Questo **Service** si occuperà di eseguire la logica per il caricamento del Risultato di un Test, prendendo in input, il **DTO** riguardante il Risultato da caricare. Si andrà quindi ad aggiungere quest'ultimo tramite la **Porta** e l'**Adapter** corrispondente già presente.



8 *LLM$_G$* Evaulator - Python

***LLM$_G$* Evaluator** è un sistema progettato per valutare frasi utilizzando diversi modelli *LLM$_G$*, tra cui BERT e RoBERTa. Integra inoltre numerose metriche di valutazione, come BLEU, TER, CHRF, ROUGE, BERTscore e CrossEncoder. Per implementare tutte queste tecnologie in Python, è stato utilizzato il gestore di pacchetti *pip$_G$*, che consente di installare facilmente le librerie necessarie. Nel progetto, l'intero sistema è stato strutturato all'interno della cartella src/application/evaluation, all'interno della cartella Artificial_QI. Questo file include tutte le componenti e le funzioni necessarie per garantire il corretto funzionamento del sistema.

8.1 Allenamento Random Forest

Il nostro sistema utilizza un metodo chiamato Random Forest per calcolare i punteggi, sfruttando alberi decisionali per determinare un punteggio di somiglianza tra le due frasi in ingresso. Questo approccio permette di ottenere risultati suufficientemente accurati rispetto alla realtà. Le Random Forest richiedono un processo di allenamento per funzionare correttamente. Sebbene il nostro sistema sia già stato addestrato su un dataset di 5000 domande, l'allenamento è un'operazione opzionale. Qualora fosse necessario ricalibrare i pesi, viene fornito tutto il necessario all'interno della cartella `trainModel`. Si consiglia di utilizzare cuda con la corrispondente versione di pytorch per velocizzare il processo di allenamento. Il file `requirements.txt` contiene le librerie aggiuntive che è necessario installare per scaricare un dataset ed elaborarlo. Il primo passo per l'allenamento consiste nello scaricare il dataset, il modello preimpostato è `mteb/stsb_multi_mt`(solo la parte in italiano). Dopo aver scaricato e preparato il modello si puo procedere con l'allenamento. Il file `train.py` contiene il codice necessario per farlo.
Nello specifico il dataset viene scaricato, elaborato e salvato nella cartella assets come `ds1`. Il trainer carica poi il dataset elaborato e procede con l'allenamento. Il modello viene salvato nella cartella assets come `newmodel.joblib`.

8.1.0.1 Passi da eseguire per l'allenamento (default)

1. Installare le dipendenze aggiuntive:

   ```
   pip_G install -r trainModel/requirements.txt
   ```
2. Scaricare e preparare il dataset:

   ```
   python trainModel/dsLoader.py
   ```
3. Eseguire lo script di allenamento:

   ```
   python trainModel/train.py
   ```

8.2 Sistema di valutazione

Come descritto in precedenza, il sistema è interamente implementato nel backend, all'interno di un file denominato AlgoritmoValutazioneRisposteImpl.py. Questo file rappresenta il cuore del sistema di valutazione e richiede un modello che accetti in input i valori delle metriche e in output restituisca lo score.

8.2.1 Installazione

Per l'installazione non serve altro che eseguire un comando di Python per installare tutte le librerie specificate nel file `requirements.txt`, un file che contiene la lista di librerie necessarie per il corretto funzionamento:

```
pip_G install -r requirements.txt
```

Grazie a questo comando tutte le librerie necessarie verranno installate.

8.2.2 Utilizzo

L'utilizzo del sistema è stato integrato all'interno delle API tramite una route dedicata, sviluppata utilizzando Flask. Il loro utilizzo è descitto nel documento di Specifica Tecnica nel capitodo dedicato alle API.

9 Problemi Noti

Qui di seguito vengono descritti i principali problemi noti presenti all'interno del sistema.

9.1 Backend Flask

Nella prima operazione di esecuzione di un test dopo il deployment dell'applicativo, il sistema potrebbe sembrare in stallo, non restituire un risultato o ritornare un errore.
Ciò è dovuto al fatto che il sistema non ha ancora caricato i modelli LLM_G utilizzati dall'Algoritmo di Valutazione.
Il caricamento dei modelli richiede un tempo variabile a seconda della potenza del server, della velocità di scaricamento dei modelli stessi e dalle loro grandezze.

Una volta che viene eseguito il test per la prima volta ed il modello viene caricato, il sistema non dovrebbe più presentare questo problema.