Specifica Tecnica

Gruppo	Alt+F4
Data	12 Giugno 2025
Versione	v1.1

Registro modifiche

Versione	Data	Autore/i	Verificatore/i	Descrizione
v1.1	4 Giugno 2025	Guirong Lan	Marko Peric	Modifica sezione Architettura
v1.0	4 Giugno 2025		Enrico Bianchi	Release
v0.29	5 Maggio 2025	Guirong Lan	Marko Peric	Stesura sezioni Componenti
v0.28	26 Aprile 2025	Marko Peric	Guirong Lan	Stesura sezione Progettazione di dettaglio del frontend riguardante i test
v0.27	24 Aprile 2025	Enrico Bianchi	Guirong Lan	Stesura progettazione in dettaglio dei componenti riguardanti i dataset
v0.26	23 Aprile 2025	Francesco Savio, Pedro Leoni	Eghosa Matteo Igbinedion Osamwonyi	Terminata stesura Progettazione alto livello back-end
v0.25	15 Aprile 2025	Francesco Savio, Marko Peric, Eghosa Matteo Igbinedion Osamwonyi, Pedro Leoni	Enrico Bianchi	Terminata stesura Progettazione alto livello front-end
v0.24	11 Aprile 2025	Pedro Leoni	Guirong Lan	Stesura Progettazione database
v0.23	9 Aprile 2025	Enrico Bianchi	Guirong Lan	Stesura Funzioni riguardanti il dataset nella Progettazione ad alto livello del back-end
v0.22	8 Aprile 2025	Marko Peric	Guirong Lan	Stesura sezione Funzioni Test

v0.21	8 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Francesco Savio	Stesura sezione get_LLM_datas, get_all_LLMs, delete_LLM, write_LLM
v0.20	6 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Guirong Lan	Stesura sezioni LLMListPage, LLMPage
v0.19	5 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Marko Peric	Stesura sezione StandardPage
v0.18	4 Aprile 2025	Guirong Lan, Pedro Leoni	Marko Peric	Aggiunta sezione Progettazione ad alto livello del front-end
v0.17	3 Aprile 2025	Guirong Lan	Francesco Savio	Stesura sezione Architettura del Back-end
v0.17	3 Aprile 2025	Guirong Lan	Francesco Savio	Stesura sezione Architettura del Back-end
v0.16	3 Aprile 2025	Enrico Bianchi	Marko Peric	Stesura sezione Architettura del Front-end
v0.15	3 Aprile 2025	Pedro Leoni	Marko Peric	Sezione Introduzione all'architettura
v0.14	2 Aprile 2025	Francesco Savio	Guirong Lan	Sezioni TypeScript, PostgreSQL, Hugging Face
v0.13	1 Aprile 2025	Pedro Leoni	Francesco Savio	Sezione Docker
v0.12	1 Aprile 2025	Pedro Leoni	Francesco Savio	Sezione Chart.js
v0.11	1 Aprile 2025	Enrico Bianchi	Francesco Savio	Sezione Jest
v0.10	1 Aprile 2025	Pedro Leoni	Francesco Savio	Sezione Angular

v0.9	1 Aprile 2025	Guirong Lan	Francesco Savio	Sezione Dependency Injector
v0.8	1 Aprile 2025	Guirong Lan	Francesco Savio	Sezione Black
v0.7	1 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Francesco Savio	Sezione Pytorch
v0.6	1 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Francesco Savio	Sezione Pydentic
v0.5	1 Aprile 2025	Enrico Bianchi	Francesco Savio	Sezione Pytest
v0.4	1 Aprile 2025	Eghosa Matteo Igbinedion Osamwonyi	Francesco Savio	Sezione SQLAlchemy
v0.3	1 Aprile 2025	Enrico Bianchi	Francesco Savio	Sezione Flask
v0.2	1 Aprile 2025	Guirong Lan	Francesco Savio	Sezione Python
v0.1	27 Marzo 2025	Francesco Savio	Guirong Lan	Sezioni Introduzione

Indice

1 Introduzione

Il seguente documento definisce la specifica tecnica del progetto Artificial QI, un software progettato per valutare l'affidabilità delle risposte fornite da un LLM_G rispetto a un dataset di domande. L'obiettivo di questa specifica è fornire una definizione chiara e dettagliata delle scelte fatte dal team di sviluppo riguardo l'architettura del sistema, i design pattern utilizzati, le tecnologie adottate e le scelte progettuali effettuate. Questo documento andrà a indicare:

- Le architetture e i design pattern scelti sia lato front-end che back-end.
- Le tecnologie utilizzate per la realizzazione del software.
- La progettazione ad alto livello dei componenti sia lato front-end che back-end.
- La progettazione di dettaglio dei componenti sia lato front-end che back-end.

1.1 Riferimenti

1.1.1 Normativi

- Norme di progetto
- Capitolato https://www.math.unipd.it/~tullio/IS-1/2024/Progetto/C1.pdf
- Regolamento del progetto: https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/PD1.pdf

1.1.2 Informativi

- Analisi dei Requisiti
- Glossario
- Pattern architetturali https://www.math.unipd.it/~rcardin/swea/2022/Software%20Architecture%20Patterns.pdf
- Architettura esagonale https://github.com/rcardin/hexagonal
- Progettazione software https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T06.pdf
- Analisi statica https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T10.pdf

- Analisi dinamica https://www.math.unipd.it/~tullio/IS-1/2024/Dispense/T11.pdf
- Pattern creazionali https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Creazionali.pdf
- Pattern strutturali https://www.math.unipd.it/~rcardin/swea/2022/Design%20Pattern%20Strutturali.pdf
- Pattern comportamentali https://drive.google.com/file/d/1cpi6rORMxFtC91nI6_sPrG1Xn-28z8eI/view
- Verbali interni
  - 2024_03_24
- Verbali esterni

2 Tecnologie utilizzate

All'interno della sezione vengono indicate tutte le tecnologie utilizzate dal gruppo per lo sviluppo del progetto. La validità delle tecnologie adottate è stata provata durante lo sviluppo del $Proof\ of\ Concept_G$.

2.1 Backend

2.1.1 Python

$Python_G$ è un linguaggio di programmazione ad alto livello ed è considerato il riferimento principale per l'intelligenza artificiale e il machine learning.

Motivazioni:

1. **LLM**: è il linguaggio più utilizzato per interagire con modelli di linguaggio di grandi dimensioni.
2. **Librerie**: dispone di una vasta gamma di librerie e moduli di terze parti che semplificano lo sviluppo di applicazioni.
3. **Orientato agli oggetti**: supporta la programmazione orientata agli oggetti, consentendo di organizzare il codice in modo modulare e aderente ai principi $SOLID_G$.

4. **Dinamicamente tipizzato**: offre flessibilità e rapidità di sviluppo, permettendo di assegnare variabili senza dichiararne esplicitamente il tipo.

5. **Gestione eccezioni**: permette di intercettare e gestire errori in fase di esecuzione, migliorando la robustezza del codice.

6. **Comunità**: una vasta comunità di sviluppatori garantisce supporto continuo, aggiornamenti e soluzioni ai problemi più comuni.

2.1.2 Flask

Flask è un micro-framework per la creazione di API e applicazioni web in Python.

Motivazioni:

1. **Leggerezza e semplicità**: Flask è un framework minimalista che fornisce solo gli strumenti essenziali per lo sviluppo web, lasciando agli sviluppatori la libertà di scegliere librerie e strumenti aggiuntivi;

2. **Flessibilità**: Non impone una struttura rigida, permettendo di organizzare il codice in modo personalizzato e adattandolo alle necessità del progetto;

3. **Modularità**: Flask permette di organizzare il codice in Blueprints, facilitando la suddivisione di un'app in moduli più gestibili e scalabili;

4. **Ecosistema esteso**: Grazie alla sua popolarità, esiste un vasto ecosistema di estensioni disponibili per aggiungere funzionalità come autenticazione, gestione del database e gestione delle sessioni;

5. **Facilità di apprendimento**: La curva di apprendimento è molto bassa, rendendolo ideale per sviluppatori alle prime armi con lo sviluppo web in Python;

6. **Compatibilità**: Può essere facilmente integrato con ORM come SQLAlchemy;

7. **Testabilità**: Flask include strumenti per il testing delle API, semplificando il testing del codice;

8. **Controllo completo**: A differenza di framework più strutturati come Django, Flask offre agli sviluppatori un controllo totale su routing, gestione delle richieste e configurazione del progetto.

2.1.3 SQLAlchemy

SQLAlchemy è una libreria open source che fornisce un toolkit SQL e un Object Relational Mapper (ORM) per Python, facilitando la gestione e l'interazione con i database in maniera flessibile e performante. La sua architettura consente di operare sia a livello di query SQL dirette sia tramite l'astrazione offerta dall'ORM, rendendola adatta a diverse esigenze applicative.

Motivazioni:

1. **Flessibilità:** SQLAlchemy permette di alternare facilmente tra l'uso diretto del linguaggio SQL e l'approccio ORM, offrendo agli sviluppatori la libertà di scegliere il livello di astrazione più adatto al progetto.
2. **Compatibilità:** Supporta una vasta gamma di database relazionali, come PostgreSQL, MySQL, SQLite e altri, garantendo interoperabilità e facilità di migrazione.
3. **Prestazioni:** Grazie a funzionalità avanzate come il lazy loading e il caching, SQLAlchemy ottimizza le operazioni di accesso e manipolazione dei dati, migliorando le prestazioni complessive dell'applicazione.
4. **Integrazione:** Si integra agevolmente con altri framework e librerie Python, facilitando lo sviluppo di applicazioni web e di sistemi complessi.
5. **Comunità attiva:** Una vasta comunità di sviluppatori e una documentazione dettagliata assicurano supporto continuo e aggiornamenti costanti, rendendo SQLAlchemy una scelta affidabile anche per progetti di lunga durata.

2.1.4 Pytest

Pytest è un framework di testing per Python progettato per essere semplice, potente e scalabile, ideale per il testing di unità e integrazione.

Motivazioni:

1. **Semplicità e sintassi intuitiva**: Pytest permette di scrivere test con una sintassi chiara e concisa, senza la necessità di classi o metodi speciali;
2. **Riconoscimento automatico dei test**: Pytest rileva automaticamente i file e le funzioni di test, senza bisogno di registrarle manualmente (ad esempio, un test viene riconosciuto semplicemente se inizia con test_);

3. **Fixture e Dependency Injection**: Pytest permette di definire fixture per la configurazione e il teardown degli ambienti di test, rendendo i test più puliti e modulari;

4. **Parametrizzazione dei test**: Pytest consente di eseguire lo stesso test con input diversi, evitando duplicazione di codice;

5. **Report dettagliati**: Pytest fornisce messaggi di errore dettagliati, fornendo tracebacks e assert introspection, e supporta strumenti di debugging come $pytest\ –trace$;

6. **Ampio ecosistema di plugin**: Esistono numerosi plugin ufficiali e di terze parti che estendono le sue funzionalità per soddisfare esigenze specifiche (ad esempio, plugin di supporto per il mocking);

7. **Integrazione con strumenti per CI e calcolo del coverage**: Pytest si integra perfettamente con strumenti di Continuous Integration (CI) (come GitHub Actions), per automatizzare l'esecuzione dei test in fase di build. Inoltre, grazie alla sua compatibilità con il plugin $pytest\text{-}cov$ (per il calcolo della copertura), è possibile monitorare e analizzare automaticamente la copertura del codice durante i processi di CI.

2.1.5 Pydentic

Pydentic è una libreria Python dedicata alla validazione dei dati. Permette di definire modelli di dati sfruttando le annotazioni di tipo, assicurando che i dati in ingresso rispettino i formati e i tipi attesi. Questa tecnologia risulta particolarmente utile per la creazione di API e applicazioni robuste, semplificando il controllo e la gestione della correttezza dei dati.

Motivazioni:

1. **Validazione automatica:** Pydentic consente una validazione rigorosa e automatica dei dati in ingresso, assicurando che ogni campo rispetti il formato e il tipo previsto.

2. **Compatibilità con type hints:** L'utilizzo nativo delle annotazioni di tipo migliora la leggibilità del codice e facilita l'individuazione di errori in fase di sviluppo.

3. **Integrazione fluida:** Si integra facilmente con altri framework e librerie, come Flask, rendendo la gestione dei dati nelle API semplice e diretta.

4. **Prestazioni ottimizzate:** Nonostante offra funzionalità avanzate, Pydentic è progettato per garantire elevate prestazioni, contribuendo a mantenere l'efficienza complessiva dell'applicazione.

5. **Facilità d'uso e documentazione:** La sua interfaccia intuitiva e la documentazione completa permettono una rapida adozione e una facile manutenzione nel tempo.

2.1.6 Pytorch

Pytorch è una libreria open source per il machine learning e il deep learning, sviluppata per offrire flessibilità e prestazioni elevate nello sviluppo di modelli neurali.

Motivazioni:

1. **Prestazioni:** Grazie all'integrazione con CUDA, Pytorch sfrutta appieno le potenzialità delle GPU, accelerando l'addestramento e l'inferenza dei modelli su dataset di grandi dimensioni.

2. **Orientato alla ricerca:** La sua natura dinamica e l'interfaccia intuitiva lo rendono ideale per la sperimentazione e l'innovazione, permettendo agli sviluppatori di iterare rapidamente sulle proprie idee.

3. **Ampio ecosistema:** Pytorch si integra facilmente con altre librerie scientifiche e di machine learning, offrendo un ambiente completo per la realizzazione di soluzioni avanzate.

4. **Comunità attiva:** Una vasta comunità di ricercatori e sviluppatori contribuisce continuamente con nuovi modelli, strumenti e documentazione, favorendo lo sviluppo collaborativo e l'aggiornamento costante.

5. **Modernità:** Con aggiornamenti regolari e il supporto per le ultime tecnologie, Pytorch si adatta rapidamente alle evoluzioni del campo dell'intelligenza artificiale, rimanendo all'avanguardia.

2.1.7 Black

$Black_G$ è un formatter di codice $Python_G$, progettato per garantire uno stile di codifica uniforme e migliorare la leggibilità del codice.

Motivazioni:

1. **Uniformità e Manutenibilità**: viene applicato uno stile unico a tutto il codice, indipendentemente da chi lo scrive. Questo rende il codice più leggibile e manutenibile, senza differenze di formattazione tra i file.

2. **Standardizzazione nei Team**: vengono eliminate le differenze tra le scritture di diversi programmatori, fornendo uno standard nello sviluppo.

3. **Efficacia**: non è necessario perdere tempo a discutere su spaziature o virgolette, rendendo il processo di sviluppo più efficiente.

2.1.8 Dependency Injector

$Dependency\ Injector_G$ è una libreria $Python_G$ che implementa il pattern $Dependency\ Injection_G$, gestendo centralmente le dipendenze tra le classi.

Motivazioni:

1. **Gestione flessibile delle dipendenze**: supporta una varietà di provider (Factory, Singleton, ecc.), che permette di costruire e configurare le dipendenze in modo versatile e adattabile alle esigenze specifiche del progetto.

2. **Test e ambienti dinamici**: la possibilità di sovrascrivere i provider al volo è particolarmente utile per i test, consentendo di sostituire facilmente le dipendenze con versioni fittizie in ambienti di sviluppo o test.

3. **Integrazione con framework popolari**: Supporta l'iniezione automatica delle dipendenze in funzioni e metodi, facilitando l'integrazione con framework web. Questo approccio migliora la modularità e semplifica la gestione delle dipendenze anche in applicazioni complesse.

2.2 Front-end

2.2.1 Angular

Angular è un framework open source scritto in Typescript per lo sviluppo di Single-Page Application SPA.

Motivazioni

1. **Basato su componenti**: impone una suddivisione modulare dell'applicazione in componenti, ciascuno composto da tre parti principali: logica (classe TypeScript), interfaccia (HTML) e stile (CSS);

2. **Dependency Injection**: offre un'implementazione integrata del pattern Dependency Injection, permettendo di iniettare servizi nei componenti demandando al framework l'onere della loro costruzione;

3. **Reattività**: grazie ai meccanismi di event binding e property binding, consente di reagire alle interazioni dell'utente e aggiornare dinamicamente l'interfaccia in base allo stato dell'applicazione;

4. **Pacchetti**: l'ecosistema di Angular include numerosi pacchetti che forniscono componenti e direttive aggiuntive per estendere le funzionalità dell'applicazione. Questi pacchetti possono essere facilmente gestiti tramite la Angular CLI, un tool a riga di comando che semplifica la gestione dei progetti Angular;

5. **Routing**: Angular fornisce nativamente un servizio di routing che semplifica la gestione della navigazione nella SPA;

6. **TypeScript**: superset di JavaScript che introduce sintassi per supportare funzionalità aggiuntive quali: controllo dei tipi a tempo di compilazione, Interfacce e tipi generici. Il codice TypeScript viene poi compilato in JavaScript per essere eseguito nel browser client.

2.2.2 Jest

Jest è un framework di testing per JavaScript e TypeScript, progettato per essere semplice, veloce e configurabile. Offre una suite completa di strumenti per il testing automatizzato, rendendo il processo di scrittura dei test semplice, veloce ed efficiente.

Motivazioni:

1. **Semplicità**: Jest funziona con una configurazione iniziale minima per la maggior parte dei progetti, semplificando l'integrazione nei workflow di sviluppo;

2. **Velocità e parallelizzazione**: Jest esegue i test in parallelo per impostazione predefinita, migliorando significativamente le prestazioni e riducendo i tempi di esecuzione dei test;

3. **Mocking e Spying integrati**: Include strumenti per creare mock di funzioni, moduli e timer, semplificando il testing unitario;

4. **Copertura del codice**: Fornisce strumenti integrati per misurare la copertura del codice ($-coverage$), evidenziando le parti non testate.

5. **Feedback chiaro e dettagliato**: Quando un test fallisce, Jest fornisce messaggi di errore estremamente chiari e informativi, aiutando rapidamente gli sviluppatori a identificare il problema. Inoltre, la modalità "watch" permette di eseguire i test automaticamente quando vengono modificati i file, supportando una modalità di sviluppo continuo e iterativo;

6. **Estendibilità e plugin**: Oltre alle sue funzionalità integrate, Jest è altamente estensibile grazie alla vasta gamma di plugin disponibili, questi strumenti permettono di ampliare le funzionalità di Jest per adattarsi meglio a specifiche esigenze di testing, senza alterare il codice sorgente;

7. **Integrazione con CI**: Jest è perfettamente integrato con piattaforme di integrazione e distribuzione continua (come GitHub Actions), Questo rende l'automazione del processo di testing estremamente semplice e fluida, contribuendo a mantenere un flusso di lavoro di sviluppo efficiente e automatizzato.

2.2.3 Chart.js

Chart.js è una libreria JavaScript che permette di integrare diverse tipologie di grafici (grafici a torta, grafici a barre, grafici di dispersione) all'interno di una pagina web tramite l'utilizzo dei HTML5 canvans.

Motivazioni:

1. **Popolarità**: è la libreria di grafici più utilizzata;

2. **Integrazione**: è compatibile con Angular per cui esiste un apposito pacchetto che fornisce delle direttive per ogni tipologia di grafico.

2.2.4 TypeScript

TypeScript è un superset di JavaScript sviluppato da Microsoft che aggiunge il supporto per la tipizzazione statica. Offre funzionalità avanzate come interfacce, classi, moduli e decoratori, rendendo il codice più sicuro, leggibile e scalabile.

Motivazioni:

1. **Tipizzazione statica**: Grazie alla definizione chiara dei tipi TypeScript aiuta a prevenire errori rilevati solo a runtime, migliorando la sicurezza del codice e riducendo i bug infine il codice è più leggibile e facile da mantenere.

2. **Supporto per il codice**: TypeScript include funzionalità avanzate di JavaScript, offrendo allo stesso tempo compatibilità con vecchie versioni di JS.

2.3 Altre tecnologie

2.3.1 Docker

Docker è un software open source che consente lo sviluppo e la distribuzione di applicazioni all'interno di container, ambienti isolati che condividono il kernel del sistema operativo.

Motivazioni:

1. **Isolamento**: ogni applicazione viene eseguita all'interno di un container dedicato, che rappresenta un ambiente di esecuzione isolato. Questo evita problemi di incompatibilità tra le dipendenze delle diverse applicazioni, comprese quelle in esecuzione sulla macchina target. Inoltre, eventuali malfunzionamenti rimangono confinati all'interno del container, senza compromettere l'intero sistema;

2. **Portabilità**: l'ambiente di sviluppo è identico a quello di produzione, questo rende la transizione semplice e senza errori inaspettati. Infatti basta che sulla macchina target sia installato Docker;

3. **Semplificazione dello sviluppo**: consente di automatizzare la configurazione e l'esecuzione dell'ambiente di runtime attraverso un insieme di comandi strutturato. Ciò riduce la probabilità di errori e semplifica l'avvio del sistema a un solo comando;

4. **Scalabilità**: i container sono più leggeri rispetto alle macchine virtuali, questo permette di aggiungere e rimuovere container in modo rapido ed efficiente.

2.3.2 PostgreSQL

PostgreSQL è un database relazionale open source avanzato, noto per la sua affidabilità, scalabilità e conformità agli standard SQL.

Motivazioni:

1. **Open Source**: PostgreSQL è completamente open source, senza costi di licenza, con una comunità attiva che lo mantiene e aggiorna continuamente.

2. **Affidabilità e integrità dei dati**: Supporta transazioni ACID (Atomicità, Consistenza, Isolamento, Durabilità), garantendo la sicurezza e la coerenza dei dati.

2.3.3 Hugging Face

Hugging Face è una piattaforma leader nell'intelligenza artificiale che fornisce strumenti open-source per il Natural Language Processing (NLP), la visione artificiale e altri campi dell'AI. Offre una vasta libreria di modelli pre-addestrati, dataset ottimizzati e API facili da usare, supportando framework come PyTorch.

Motivazioni:

1. **Comunità attiva e collaborativa**: Una vasta community di sviluppatori e ricercatori contribuisce con modelli, dataset e miglioramenti costanti.
2. **Compatibilità con diverse piattaforme** : Supporta framework come TensorFlow, PyTorch e JAX, rendendo l'adozione flessibile per diversi ambienti di sviluppo.
3. **Facilità d'uso**: Le API e le librerie semplificano l'integrazione di modelli avanzati con poche righe di codice.
4. **Ampia libreria di modelli pre-addestrati**: Offre migliaia di modelli di intelligenza artificiale per NLP, visione artificiale e altre applicazioni, riducendo i tempi di sviluppo.
5. **Strumenti di valutazione e benchmarking**: Fornisce metriche integrate per confrontare le prestazioni dei modelli e scegliere le migliori soluzioni.

3 Architettura

3.1 Introduzione

Il sistema ArtificialQI è basato sull'architettura client-server. In questa architettura il carico di lavoro viene suddiviso tra due componenti:

1. **client**: interfaccia con cui gli utenti interagiscono con il sistema per richiedere risorse e/o servizi;
2. **server**: applicazione che riceve ed elabora le richieste eseguite dal client e fornisce le risposte corrette.

Il gruppo ha quindi deciso di sviluppare due componenti separati:

1. **front-end**: componente client del sistema, è un interfaccia grafica sviluppata utilizzando il framework Angular;

2. **back-end**: componente server del sistema che utilizza Python come linguaggio e Flask come framework, implementa la logica di business e la logica di persistenza.

La comunicazione tra front-end e back-end avviene tramite una API. L'API verrà implementata seguendo i principi indicati nello stile architetturale REST (Representational State Transfer), utilizzando il protocollo di comunicazione HTTP e il formato di rappresentazione delle risorse JSON. Una caratteristica delle REST API è quella di essere stateless, il che significa che la componente server non deve memorizzare le interazioni passate per poter rispondere a una richiesta del client. Questo permette di ridurre il carico sul server e migliora la scalabilità. La persistenza delle risorse viene implementata utilizzando un database relazionale gestito utilizzando PostgreSQL come DBMS.

3.1.1 Integrazione dei componenti

Il gruppo ha deciso di utilizzare un container Docker per ognuno dei due componenti sopra citati. Per gestire e permettere la comunicazione tra i due container menzionati, il gruppo ha deciso di utilizzare Docker Compose.

3.2 Front-end

$Angular_G$ adotta il pattern architetturale Model-View-ViewModel (MVVM), un pattern architetturale che separa le responsabilità dell'interfaccia utente (UI) dalla $business\ logic$ di un'applicazione. In $Angular_G$, l'MVVM è implementato attraverso i componenti, che fungono da ViewModel e gestiscono il legame tra il Model (dati e logica dell'applicazione) e la View (interfaccia utente). Grazie al data binding bidirezionale, $Angular_G$ sincronizza automaticamente i cambiamenti tra il Model e la View, semplificando l'interazione tra i livelli dell'applicazione. Il pattern architetturale MVVM è formato dai seguenti tre layer:

- **Model**: Si riferisce ai dati dell'applicazione e alla $business\ logic$, come la validazione dei dati o l'elaborazione delle operazioni. Il Model non è a conoscenza dei livelli View o ViewModel, il che lo rende indipendente dalla presentazione. In un'architettura MVVM, il Model rappresenta generalmente i dati che l'applicazione elabora e visualizza. In $Angular_G$, il Model è generalmente costituito da classi TypeScript che definiscono la struttura e il comportamento dei dati dell'applicazione. Queste classi possono includere proprietà, metodi e regole di validazione;

- **View**: È responsabile della struttura, del layout e dell'aspetto dell'interfaccia utente. La View riceve le interazioni dell'utente e le passa al ViewModel tramite data binding. Non contiene la $business\ logic$, ma si collega al ViewModel, che fornisce i dati e le operazioni necessarie per visualizzare e interagire con le

informazioni. La View in $Angular_G$ è rappresentata dai template di $Angular_G$, che sono file HTML arricchiti con direttive e binding, questi template definiscono come i dati vengono presentati all'utente e come vengono catturati gli input dell'utente. Quando i dati del Model cambiano a seguito dell'elaborazione della *business logic*, la View si aggiorna automaticamente grazie al data binding bidirezionale con il ViewModel;

- **ViewModel**: Il ViewModel funge da livello intermedio tra la View e il Model. Il ViewModel rappresenta lo stato dei dati presenti nel Model e gestisce la comunicazione bidirezionale tra il Model e la View attraverso la tecnica del data binding. Inoltre, il ViewModel può contenere logica legata all'elaborazione e alla validazione dei dati, consentendo operazioni più complesse, come il filtraggio, l'ordinamento o il raggruppamento dei dati, senza dover intervenire sul livello della View o del Model. In $Angular_G$, il ViewModel è spesso rappresentato dai Component e dalle relative classi TypeScript. In $Angular_G$ i Component contengono l'*application logic*, espongono proprietà e metodi utilizzati nei template per il data binding e comunicano con il Model attraverso i servizi (indicati con il decoratore *@Injectable*).

I concetti fondamentali implementati da $Angular_G$ sono:

- **Data Binding**: $Angular_G$ utilizza un avanzato sistema di data binding che consente una sincronizzazione automatica tra il Model (dati dell'applicazione) e la View (interfaccia utente). Il data binding può avvenire in una o due direzioni, con il two-way data binding i dati rimangono sincronizzati tra Model e View, garantendo un aggiornamento dinamico dell'interfaccia utente;

- **Modularità**: $Angular_G$ è basato su un'architettura modulare, in cui un'applicazione è suddivisa in moduli (*NgModules*). Ogni modulo può contenere componenti, servizi e altre funzionalità, permettendo una suddivisione chiara e organizzata del codice, facilitando lo sviluppo e il riutilizzo delle componenti;

- **Component-based architecture**: $Angular_G$ segue un'architettura basata sui componenti, in cui l'interfaccia utente è costruita attraverso componenti riutilizzabili e indipendenti. Ogni componente gestisce la propria logica e il proprio template, migliorando la manutenibilità e la scalabilità dell'applicazione;

- **Directive e template engine**: $Angular_G$ estende l'HTML con directive (direttive) personalizzate che permettono di manipolare il DOM in modo dinamico. Esistono directive strutturali per il controllo del flusso della UI e directive attributo per la modifica degli elementi HTML;

- **RxJS e programmazione reattiva**: $Angular_G$ utilizza *RxJS* (Reactive Extensions for $Javascript_G$) per gestire lo stato e la comunicazione tra componenti

in modo asincrono e reattivo. Questo permette di gestire flussi di dati come eventi utente, richieste HTTP e aggiornamenti dello stato attraverso Observable e operatori reattivi.

Il codice sviluppato per il front-end verrà organizzato nel seguente modo:

```
ArtificialQI/
├── src/
│   └── app/
│       ├── core/
│       │   ├── services/
│       │   ├── models/
│       │   └── interceptors/
│       ├── shared/
│       │   ├── components/
│       │   ├── directives/
│       │   └── pipes/
│       ├── features/
│       │   └── feature1/
│       │       ├── components/
│       │       └── pages/
│       └── layout/
├── assets/
└── environments/
```

Di seguito è riportata una breve descrizione delle cartelle principali:

- **src**: contiene il codice sorgente;
- **assets**: contiene i file statici, come immagini o font;
- **environments**: contiene file di configurazione per ambienti diversi (sviluppo e produzione).

La cartella *src* contiene il codice sorgente dell'applicazione all'interno della sottocartella *app*, *app* è suddivisa in:

- **core**: contiene modelli e *business logic* all'interno delle sottocartelle:
  - **models**: interfacce e classi che rappresentano i dati e le entità principali dell'applicazione;
  - **services**: gestiscono la *business logic* e le chiamate API;
  - **interceptors**: modificano le chiamate HTTP (ad esempio per aggiungere autenticazione o logging) e intercettano errori API per gestirli in modo centralizzato.

- **shared**: contiene moduli, componenti e pipe riutilizzabili, all'interno delle sottocartelle:
  - **components**: componenti condivise (es. pulsanti);
  - **directives**: direttive personalizzate;
  - **pipes**: pipe personalizzate per formattazione dati.
- **features**: moduli feature-base, ogni feature ha una cartella dedicata organizzata in:
  - **components**: contiene i componenti specifici della feature, che possono essere combinati per formare una pagina;
  - **pages**: contiene le pagine principali della feature, ogni pagina funge da container component del ViewModel nel pattern MVVM.
- **layout**: contiene componenti per il layout dell'app (es. navbar, sidebar, footer).

3.3 Back-end

Il team ha adottato l'architettura esagonale come modello di riferimento per la progettazione del back-end. Questo paradigma architetturale trae origine dall'architettura a livelli ma ne supera i limiti, in particolare la dipendenza rigida tra i livelli. La rappresentazione esagonale è motivata da diversi fattori. Da un lato, richiama la struttura modulare di un alveare, dove ogni cella si connette ad altre in modo scalabile e componibile. Dall'altro, la simmetria dell'esagono facilita la separazione delle responsabilità, suddividendo chiaramente i confini tra la logica di business e le infrastrutture esterne.

3.3.1 Principali avversità dell'architettura esagonale

Nonostante i numerosi vantaggi, l'architettura esagonale presenta alcune complessità:

- **Elevata complessità progettuale**: richiede un'implementazione rigorosa e una conoscenza avanzata dei pattern architetturali per evitare violazioni dei principi di separazione.
- **Difficoltà nel debugging**: la netta separazione tra il core e le dipendenze esterne può introdurre problematiche di integrazione, rendendo più complessa la diagnostica dei malfunzionamenti.

3.3.2 Struttura dell'architettura esagonale

L'architettura si articola in tre componenti principali:

- **Core**
  - Costituisce il nucleo applicativo, contenente la logica di business.
  - È completamente indipendente dalle interfacce utente, dai database e da qualsiasi servizio esterno.
  - Definisce le porte necessarie per interagire con il mondo esterno.
- **Port**
  - Le porte costituiscono punti di comunicazione tra il core e le infrastrutture esterne. Esistono due tipi principali di porte:
    - Inbound ports: definiscono le interfacce attraverso cui il core riceve input da componenti esterni (ad esempio, controller o API).
    - Outbound ports: specificano le interfacce per l'interazione con servizi esterni, come database o API di terze parti.
- **Adapter**
  - Implementano le porte, fungendo da strato di traduzione tra il core e le infrastrutture esterne.
  - Consentono di isolare il dominio applicativo dai dettagli implementativi, garantendo disaccoppiamento e testabilità.

3.3.3 Vantaggi dell'architettura esagonale

- **Separazione delle responsabilità**: la logica di business è confinata nel core, mentre le porte definiscono le interfacce per l'interazione con le dipendenze esterne. Gli adapter implementano queste interfacce, garantendo un elevato grado di modularità e disaccoppiamento tra i componenti.
- **Flessibilità**: grazie all'isolamento del core, l'architettura supporta l'aggiunta, la modifica o la rimozione di funzionalità senza impattare il sistema nel suo complesso, migliorando l'adattabilità a nuove esigenze.
- **Scalabilità**: la possibilità di integrare nuovi adapter consente al sistema di supportare differenti tecnologie senza alterare la logica di business. Il modello esagonale permette di scalare sia orizzontalmente, distribuendo i carichi, sia verticalmente, aggiungendo nuove funzionalità in modo incrementale.

- **Testabilità**: l'isolamento del core semplifica il collaudo dell'applicazione, favorendo l'uso di mock e stub per simulare il comportamento delle dipendenze esterne e garantire test affidabili e indipendenti.
- **Manutenibilità**: ogni modifica, sia nella logica di business che nelle integrazioni con servizi esterni, resta confinata al rispettivo modulo senza impattare il resto dell'architettura, rendendo il sistema più robusto ed evolvibile nel tempo.

Il gruppo ha scelto di adottare un'architettura monolitica basata sul modello esagonale per garantire una netta separazione tra logica di business e presentazione. Questo approccio consente di mantenere un core applicativo ben strutturato e indipendente dalle tecnologie esterne, migliorando la modularità e la manutenibilità del sistema. Grazie all'uso degli adapter, l'architettura esagonale facilita l'integrazione con servizi esterni, come database, modelli LLM e API, permettendo al core di interagire con diverse infrastrutture senza dipendere direttamente dai dettagli implementativi.

L'organizzazione del back-end è strutturata in conformità ai principi dell'architettura esagonale, con l'obiettivo di facilitare la traduzione della progettazione in codice e garantire una chiara separazione delle responsabilità.

Il codice sviluppato per il front-end verrà organizzato nel seguente modo:

```
ArtificialQI/
├── core/
├── port/
│   ├── incoming/
│   └── outcoming/
├── adapter/
│   ├── incoming/
│   └── outcoming/
│       └── sql-alchemy/
├── components/
├── models/
├── routes/
└── tools/
```

3.4 Deplopyment

3.4.1 Nota preliminare

Assicurarsi che **Docker Desktop sia correttamente avviato e in esecuzione** prima di eseguire qualsiasi comando. Nel caso in cui, durante l'esecuzione dei comandi in un

terminale Windows, venga restituito un errore simile a:

```
SecurityError:  (:)  [], PSSecurityException
```

è possibile risolvere temporaneamente il problema eseguendo il seguente comando PowerShell:

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Per effettuare il deployment del **frontend** e del **backend** sviluppati, è necessario avere installato e in esecuzione l'applicazione **Docker Desktop**.

3.4.2 Fase 1: Build delle immagini Docker

Aprire una console qualsiasi e digitare il seguente comando:

docker compose build

Questo comando genererà tre immagini Docker denominate:

- `angularclient` (frontend sviluppato in Angular)
- `flaskapi` (backend sviluppato in Flask)
- `postgres` (database PostgreSQL)

3.4.3 Fase 2: Avvio del contenitore

Per avviare l'intera applicazione, digitare:

docker compose up

Questo comando creerà e avvierà un ambiente composto dai tre servizi sopra elencati.

3.4.4 Fase 3: Accesso all'applicazione

Una volta che il sistema è correttamente avviato, l'utente potrà accedere all'interfaccia dell'applicazione tramite il seguente indirizzo nel browser:

http://localhost:4200

3.4.5 Immagini Docker

- **PostgreSQL DB**
  - Immagine: `postgres:15.1-alpine`;
  - Ambiente: local, sviluppo;
  - Porte:
    - 5432:5432: database PostgreSQL.
  - Volume:
    - `pgdata` per la persistenza dei dati;
    - `./db/init.sql` per l'inizializzazione automatica del database.
- **Flask API**
  - Immagine: `flaskapi:latest`, basata su `python:3.12.10-slim`;
  - Ambiente: local, sviluppo;
  - Porte:
    - 5000:5000: interfaccia REST API Flask.
  - Volume:
    - `./back-end/ArtificialQI` montato in `/usr/src/ArtificialQI`.
  - Comando di avvio:
    - `flask run --debug --host 0.0.0.0`
  - Dipendenze:
    - `postgres` (connessione al database tramite variabile `DB_URL`).
- **Angular Client**
  - Immagine: `angularclient:latest`, basata su `node:22.15.0-alpine`;
  - Ambiente: local, sviluppo;
  - Porte:
    - 4200:4200: interfaccia utente Angular.
  - Volume:
    - `./front-end/ArtificialQI/src` montato in `/usr/src/ArtificialQI/src`.
  - Comando di avvio:
    - `ng serve --poll 2000 --host 0.0.0.0`
  - Dipendenze:
    - `flaskapi` (consumo dei servizi esposti dall'API).

4 Progettazione ad alto livello

4.1 Front-end

4.1.1 StandardPage

4.1.1.1 Descrizione

Il componente StandardPage definisce la struttura di layout comune utilizzata da ogni pagina con le seguenti funzionalità:

- Creazione di un messaggio di sistema;
- Avvio di un caricamento grafico.

4.1.1.2 SottoComponenti

StandardPage è composto dai seguenti sotto-componenti:

- **MenuComponent**: permette la navigazione tra le varie pagine;
- **FooterComponent**: contiene informazioni riguardanti i contatti e il copyright;
- **LoadingComponent**: si occupa dell'avvio del caricamento grafico;
- **MessageComponent**: gestisce l'apparizione di messaggi di sistema;
- **ConfirmComponent**: gestisce la conferma di operazioni critiche.

4.1.2 HomePage

4.1.2.1 Descrizione

HomePage è il componente che guida l'utente nelle altre pagine web.

4.1.2.2 SottoComponenti

HomePage è composto dai seguenti sotto-componenti:

- **ContentComponent**: componente che descrive il progetto e le funzionalità della web app.

4.1.3 DatasetListPage

4.1.3.1 Descrizione

DatasetListPage è il componete che mostra le informazioni riguardanti tutti i dataset salvati. Presenta le seguenti funzionalità:

- Caricamento di un dataset salvato;
- Rinominazione di un dataset salvato;
- Eliminazione di un dataset salvato;
- Creazione di una copia di un dataset salvato;
- Ricerca di un dataset salvato;
- Creazione di un dataset temporaneo;
- Creazione dataset da file JSON;
- Scorrimento della lista dei dataset salvati;
- Visualizzazione delle informazioni di un dataset.

4.1.3.2 SottoComponenti

DatasetListPage è composto dai seguenti sotto-componenti:

- **SearchBar**: barra per la ricerca dei dataset;
- **DatasetListView**: lista scorrevole dei dataset salvati. Contiene il sottocomponente: **DatasetElement**;
- **DatasetNameDialog**: form che permette di rinominare un dataset salvato;
- **JsonFileUpload**: form che permette di creare un dataset a partire da un file JSON.

4.1.3.3 Tracciamento dei requisiti

ID	**Componente**
RFO-7.0	DatasetListPage

RFO-7.01, RFO-7.03	DatasetListView
RFO-8.01, RRFO-8.02, RFO-8.03, RFO-8.04, RFO-8.05, RFO-8.07	DatasetElement
RFO-8.09, RFF-13.03, RFF-13.4	MessageComponent
RFO-7.04	SearchBar
RFO-6.01, RFO-6.02, RFO-6.03	DatasetNameDialog
RFO-8.08, RFO-8.06	ConfirmComponent
RFF-13.01, RFF-13.02	JsonFileUpload

4.2 Back-end

4.2.1 get_all

4.2.1.1 Descrizione

La funzione `get_all_datasets` permette di recuperare tutti i dataset salvati nel sistema che contengono la query nel proprio nome.

4.2.1.2 Dettagli dell'endpoint

- **HTTP method**: GET;
- **Endpoint**: `/datasets`;
- **Response Model**: `DatasetsListDTO`;
- **Dependency Injection**:
  - **dataset_service (DatasetUseCase)**: la dipendenza viene risolta tramite *Dependency Injector*. Questo permette di ottenere l'istanza del `DatasetService` dal `Container`, che si occupa di tutte le configurazioni e inizializzazioni necessarie per il servizio.

4.2.1.3 Implementazione

- Effettua una chiamata al servizio di gestione dei dataset per recuperare l'elenco completo dei dataset salvati;

- Restituisce lo stato e un oggetto `DatasetListDTO` nel caso in cui l'operazione sia andata a buon fine.

4.2.2 create

4.2.2.1 Descrizione

La funzione `create` permette di creare un nuovo dataset nel sistema.

4.2.2.2 Dettagli dell'endpoint

- **HTTP method**: POST;
- **Endpoint**: `/datasets`;
- **Response Model**: `DatasetDTO`;
- **Dependency Injection**:
  - **dataset_service (DatasetUseCase)**: la dipendenza viene risolta tramite *Dependency Injector*. Questo permette di ottenere l'istanza del `DatasetService` dal `Container`, che si occupa di tutte le configurazioni e inizializzazioni necessarie per il servizio.

4.2.2.3 Parametri

- **name**: nome da assegnare al dataset;

4.2.2.4 Implementazione

- Effettua una chiamata al servizio di gestione dei dataset per creare un nuovo dataset;
- Restituisce lo stato e un oggetto `DatasetDTO` nel caso in cui l'operazione sia andata a buon fine.

4.2.3 update

4.2.3.1 Descrizione

La funzione `update` permette di salvare le modifiche apportate a un dataset esistente o appena creato.

4.2.3.2 Dettagli dell'endpoint

- **HTTP method**: PUT;
- **Endpoint**: `/datasets/<dataset_id>`;
- **Response Model**: `DatasetDTO`;
- **Dependency Injection**:
  - **dataset_service (DatasetUseCase)**: la dipendenza viene risolta tramite *Dependency Injector*. Questo permette di ottenere l'istanza del `DatasetService` dal `Container`, che si occupa di tutte le configurazioni e inizializzazioni necessarie per il servizio;

4.2.3.3 Parametri

- **dataset (DatasetDTO)**: metadati del dataset.

4.2.3.4 Implementazione

- Effettua una chiamata al servizio di gestione dei dataset per aggiornare il dataset;
- Restituisce lo stato e un oggetto `DatasetDTO` nel caso in cui l'operazione sia andata a buon fine.

4.2.4 delete

4.2.4.1 Descrizione

La funzione `delete` permette di eliminare un dataset.

4.2.4.2 Dettagli dell'endpoint

- **HTTP method**: DELETE;
- **Endpoint**: `/datasets/<dataset_id>`;
- **Dependency Injection**:
  - **dataset_service (DatasetUseCase)**: la dipendenza viene risolta tramite *Dependency Injector*. Questo permette di ottenere l'istanza del `DatasetService` dal `Container`, che si occupa di tutte le configurazioni e inizializzazioni necessarie per il servizio;

4.2.4.3 Parametri

- **dataset_id (UUID)**: ID univoco del dataset da eliminare.

4.2.4.4 Implementazione

- Effettua una chiamata al servizio di gestione dei dataset per eliminare il dataset specificato;
- Restituisce l'esisto dell'operazione.

4.2.4.5 Tracciamento dei requisiti

ID	**Funzione**
RFO-16.05, RFO-16.06, RFO-16.07	`get_all`
RFO-17.04, RFO-17.05	`create`
RFO-16.01, RFO-16.02, RFO-16.04, RFO-17.03, RFO-17.04	`update`
RFO-16.03, RFF-19.01, RFF-19.02	`delete`

5 Progettazione di dettaglio

5.1 Front-end

5.2 Componenti

5.2.1 StandardPage

Modulo

- StandardModule, modulo funzionale dell'applicazione che gestisce i componenti, servizi e le risorse legate al layout e alle funzionalità generali delle pagine.

Elementi chiave

- **Menu**: insieme di link disponibili che permette all'utente di navigare tra le diverse pagine della webapp, realizzata dal sottocomponente **MenuComponent**.
- **Footer**: consente di visualizzare altre informazioni rilevanti al sito, realizzata dal sottocomponente **FooterComponent**.

- **LoadingAnimation**: gestisce l'operazione di caricamento di un operazione o pagina, realizzata dal sottocomponente **LoadingComponent**.
- **Message**: visualizzazione di un messaggio tramite finestra, realizzata dal sottocomponente **MessageComponent**.
- **ConfirmButton**: finestra che permette di confermare o meno le operazioni critiche, realizzata dal sottocomponente **ConfirmButton**.

5.2.2 MenuComponent

Componente padre

- StandardPage.

Elementi chiave

- **LogoIcon**: icona che rappresenta il logo del gruppo.
- **HomeTextLink**: testo che rappresenta il link alla home page dell'applicazione;
- **LLMTextLink**: testo che rappresenta il link alla pagina di gestione degli LLM;
- **DatasetTextLink**: testo che rappresenta il link alla pagina di gestione dei dataset;
- **TestTextLink**: testo che rappresenta il link alla pagina di gestione dei test;

5.2.3 FooterComponent

Componente padre

- StandardPage.

Elementi chiave

- **GroupContactText**: sezione testuale che contiene le informazioni di contatto del gruppo;
- **CopyrightText**: sezione testuale contenente le informazioni relative al copyright, in conformità con la normativa italiana ed europea

5.2.4 LoadingComponent

Componente padre

- StandardPage.

Elementi chiave

- **LoadingAnimatedIcon**: icona che rappresenta il caricamento della pagina o dell'operazione;

5.2.5 MessageComponent

Componente padre

- StandardPage.

Elementi chiave

- **MessageText**: sezione testuale che contiene il messaggio da visualizzare all'utente.

5.2.6 ConfirmComponent

Componente padre

- StandardPage.

Elementi chiave

- **ConfirmButton**: pulsante che permette di confermare l'operazione critica;
- **CancelButton**: pulsante che permette di annullare l'operazione critica;
- **OperationResultMessage**: messaggio che indica il risultato dell'operazione, realizzato dal sottocomponente **MessageComponent**;

5.2.7 HomePage

Componente padre

- StandardPage.

Modulo

- HomeModule, modulo funzionale dell'applicazione mostra le informazioni riguardo all'applicazione.

Elementi chiave

- **ContextContainer**: sezione che contiene i componenti principali della pagina, realizzata dal sottocomponente **ContentComponent**;

5.2.8 ContentComponent

Componente padre

- HomePage.

Elementi chiave

- **ContentText**: sezione testuale contenente informazioni che riguardano lo scopo e il funzionamento della webapp;

5.2.9 DatasetListPage

Componente padre

- StandardPage.

Modulo

- DatasetModule, modulo funzionale dell'applicazione che raccoglie e organizza tutti i componenti, servizi e risorse legati alla creazione, visualizzazione e gestione di dataset.

Elementi chiave

- **SearchDataset**: barra per la ricerca dei dataset, realizzata tramite sottocomponente **SearchBar**;
- **DatasetListContainer**: lista scorrevole dei dataset salvati, realizzata tramite sottocomponente **DatasetListView**;
- **CreateDatasetButton**: pulsante che permette la creazione di un dataset temporaneo;
- **CreateDatasetFromJSONButton**: pulsante che permette di richiedere la creazione di un dataset a partire da un file JSON, la creazione viene poi realizzata tramite **JsonFileUpload**;
- **StatusMessage**: popup di conferma di eliminazione di un dataset o di sovrascrittura del dataset caricato (se modificato), realizzato tramite sottocomponente **ConfirmComponent**;
- **ErrorMessage**: messaggio di errore visualizzato in seguito a un errore nell'ottenimento o nel salvataggio dei dati, realizzato tramite sottocomponente **MessageComponent**.

Servizio

- **DatasetService**: servizio iniettato tramite '@Injectable', fornisce le funzionalità principali per la gestione della lista di dataset.

5.2.10 DatasetListView

Componente padre

- DatasetListPage.

Elementi chiave

- **StatusMessage**: paragrafo contenente il messaggio che viene visualizzato dall'utente nel caso non ci siano dataset salvati nel sistema;
- **DatasetList**: lista scorrevole dei dataset salvati nel sistema, permette la creazione di un sottocomponente **DatasetElement** per ogni dataset salvato (tramite la direttiva `*ngFor`).

Input

- data sets: lista contenente tutti i dataset salvati e/o ricercati nel sistema.

Output

- rename Dataset: evento emesso quando riceve da **DatasetElement** la richiesta di rinomina di un dataset;
- copy Dataset: evento emesso quando riceve da **DatasetElement** la richiesta di copia di un dataset;
- delete Dataset: evento emesso quando riceve da **DatasetElement** la richiesta di eliminazione di un dataset;
- load Dataset: evento emesso quando riceve da **DatasetElement** la richiesta di caricare un dataset.

5.2.11 DatasetElement

Componente padre

- DatasetListView.

Elementi chiave

- **NameText**: sezione testuale contenente il nome del dataset;
- **DatasetDate**: sezione testuale contenente la data dell'ultima modifica del dataset;
- **RenameButton**: pulsante per richiedere la rinominazione del dataset, questa viene poi realizzata tramite il sottocomponente **DatasetNameDialog**;

- **CopyButton**: pulsante per richiedere la copia del dataset;
- **DeleteButton**: pulsante per richiedere l'eliminazione del dataset, questa viene poi confermata o annullata tramite il sottocomponente **ConfirmComponent**;
- **LoadButton**: pulsante per richiedere il caricamento del dataset, se il dataset attualmente caricato presenta modifiche viene richiesta la conferma o l'annullamento della sovrascrittura tramite il sottocomponente **ConfirmComponent**.

Input

- dataset: dataset associato al componente.

Output

- rename: evento emesso quando viene premuto il pulsante per richiedere la rinominazione del dataset;
- copy: evento emesso quando viene premuto il pulsante per richiedere la copia del dataset;
- delete: evento emesso quando viene premuto il pulsante per richiedere l'eliminazione del dataset;
- load: evento emesso quando viene premuto il pulsante per richiedere il caricamento del dataset.

5.2.12 DatasetNameDialog

Componente padre

- DatasetListPage;
- DatasetContentPage.

Elementi chiave

- **NewNameLabel**: etichetta associata al campo di inserimento del nuovo nome;
- **NewNameInput**: campo di inserimento del nuovo nome da assegnare al dataset;
- **ConfirmRenameButton**: pulsante che permette di confermare la richiesta di assegnazione o modifica del nome del dataset;
- **CancelRenameButton**: pulsante che permette di annullare la richiesta di assegnazione o modifica del nome del dataset.

Input

- currentName: gli viene assegnato il nome del dataset nel caso non sia temporaneo.

Output

- confirmRename: evento emesso quando viene premuto il pulsante per confermare l'assegnazione o modifica del nome del dataset;

5.3 Progettazione di dettaglio del Backend

5.4 Progettazione database

Il diagramma E-R (entità-relazione) mostrato in Figura 1 espone lo schema concettuale del database.

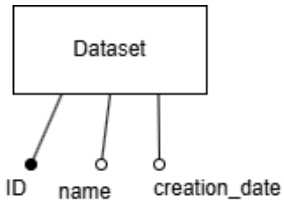

Figura 1: Schema concettuale database.

5.4.1 Dataset

- ID: valore numerico che identifica univocamente un dataset;
- $Nome$: stringa che identifica univocamente il dataset;
- $DataCreazione$: data di creazione del dataset;

5.5 Diagramma delle classi

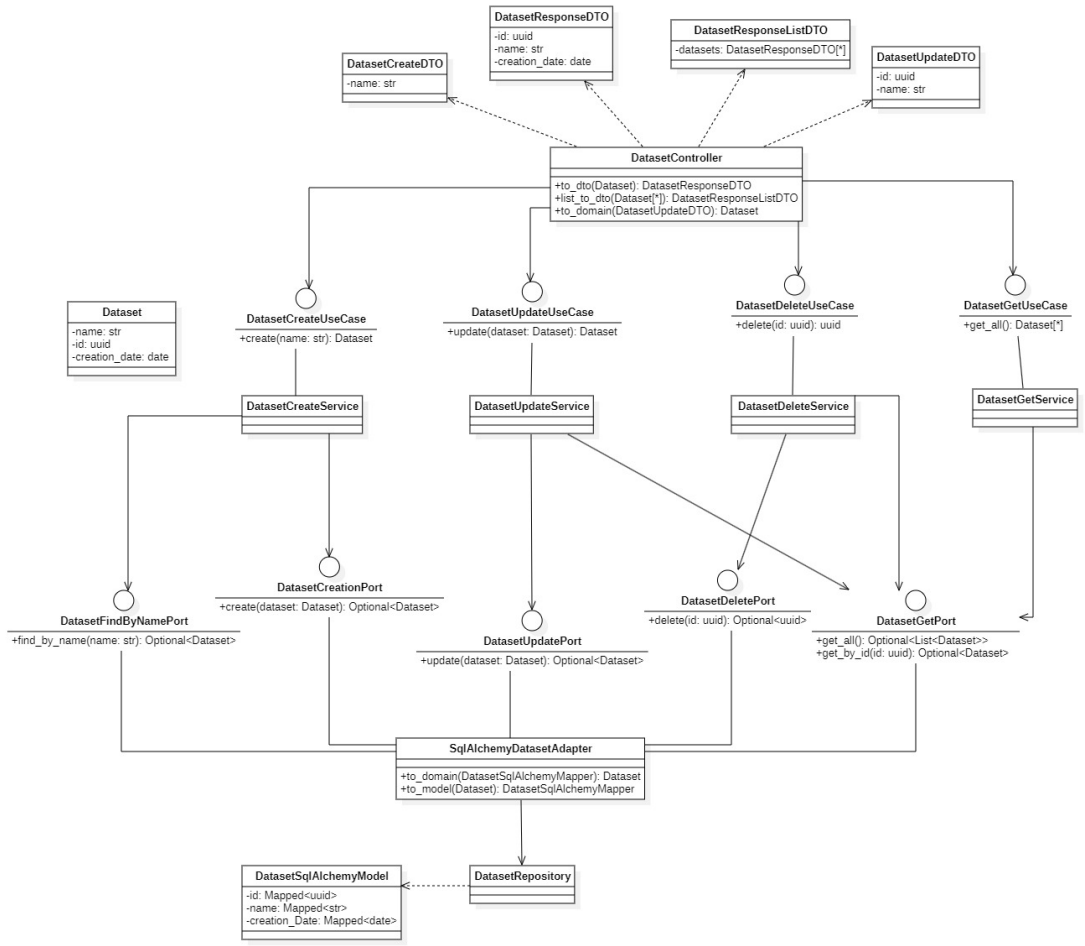

Figura 2: Diagramma delle classi legate alla gestione dei dataset

Il seguente diagramma rappresenta l'architettura software per la gestione dei $Dataset$. Ogni componente ha un ruolo ben definito per garantire separazione delle responsabilità, scalabilità e manutenibilità.

5.6 Descrizione delle classi

La seguente sezione descrive le classi utilizzate per la gestione dei $Dataset$ nel $back\text{-}end$. Vengono descritti gli attributi e i metodi appartenenti a ciascuna classe, le interfacce che sono state implementate e le dipendenze tra le classi del sistema. Le descrizioni sono suddivise in due sezioni:

- **incoming**: classi che permettono ai livelli esterni di invocare operazioni di business, vengono fornite delle interfacce chiare per l'implementazione dei casi d'uso definiti.
- **outcoming**: classi che permettono l'interazione con servizi esterni, vengono fornite delle interfacce che permettono di realizzare operazioni tecniche mantenendo il dominio indipendente da tacnologie specifiche.

Le seguenti sezioni contengono la descrizione di *ports* e *adapters* implementati.

5.7 Incoming

5.7.1 Ports

5.7.1.1 DatasetUseCase

- **Descrizione**: definisce il contratto per i casi d'uso relativi alla gestione dei dataset;
- **Operazioni**:
  - Creazione di un nuovo dataset;
  - Eliminazione di un dataset;
  - Aggiornamento dei metadati di un dataset;
  - Recupero della lista di dataset salvati nel sistema.
- **Implementazione**: i dettagli dell'implementazione sono riportati nella classe concreta.

5.7.2 Adapters

5.7.2.1 DatasetService

- **Descrizione**: è responsabile della gestione dei dataset;
- **Interfaccia implementata**: DatasetUseCase;
- **Attributi**: `_dataset_repo`: istanza della classe `DatasetRepository` utilizzata per gestire i dataset nel sistema;
- **Metodi**:
  - `__init__(self, dataset_repo: DatasetRepository)`: costruttore della classe;

  - `+ create(self, name: str)`: crea un nuovo dataset con il nome fornito. Se è già presente un dataset con lo stesso nome del sistema viene alzata un'eccezione di tipo `DuplicateNameDatasetError(name)`, se ci sono problemi con la creazione del dataset da parte di `_dataset_repo` viene alzata un'eccezione di tipo `PersistenceError`, se l'operazione ha buon fine viene restituito il dataset creato.
  - `+ delete(self, id: UUID)`: elimina il dataset con l'id fornito. Se `_dataset_repo` non riesce a ottenere il dataset con l'id fornito viene alzata un'eccezione di tipo `DatasetNonExsistentError(id)`, se ci sono problemi con l'eliminazione del dataset da parte di `_dataset_repo` viene alzata un'eccezione di tipo `PersistenceError`, se l'operazione ha buon fine viene restituito l'id del dataset eliminato.
  - `+ update(self, dataset: Dataset)`: viene rinominato il dataset con i dati ricevuti da `dataset`. Se `_dataset_repo` non riesce a ottenere il dataset tramite id viene alzata un'eccezione di tipo `DatasetNonExsistentError(id)`, se ci sono problemi con l'aggiornamento del dataset da parte di `_dataset_repo` viene alzata un'eccezione di tipo `PersistenceError`, se l'operazione ha buon fine viene restituito il dataset aggiornato.
  - `+ get_all(self)`: si ottengono tutti i dataset salvati nel sistema. Se ci sono problemi con l'ottenimento dei dati da parte di `_dataset_repo` viene alzata un'eccezione di tipo `PersistenceError`, se l'operazione ha buon fine viene restituita una lista contenente tutti i dataset salvati.

- **Dipendenze**: DatasetRepository;

5.8 Outcoming

5.8.1 Ports

5.8.1.1 DatasetRepository

- **Descrizione**: definisce un contratto per la gestione dei dataset all'interno del sistema;
- **Operazioni**:
  - Creazione di un nuovo dataset;
  - Eliminazione di un dataset;
  - Aggiornamento dei metadati di un dataset;
  - Recupero della lista di dataset salvati nel sistema;

  - Recupero di un dataset tramite il suo id;
  - Recupero di un dataset tramite il suo nome.

- **Implementazione**: i dettagli dell'implementazione sono riportati nella classe concreta.

5.8.2 Adapters

5.8.2.1 SqlAlchemyDatasetAdapter

- **Descrizione**: interagisce con un database utilizzando $SqlAlchemy$ per la gestione dei dataset;
- **Interfaccia implementata**: DatasetRepository;
- **Attributi**: `_session`: istanza della classe `Session`, creata tramite la factory `session_factory`, utilizzata per eseguire operazioni di lettura e scrittura sul database attraverso SQLAlchemy;
- **Metodi**:
  - `__init__(self, session_factory: Callable[[], Session])`: costruttore della classe;
  - `+ create(self, dataset: Dataset)`: salva il dataset fornito all'interno del database. A partire dal `dataset`, tramite `DatasetSqlAlchemyMapper` viene convertito l'oggetto di dominio `Dataset` in un oggetto ORM `DatasetSqlAlchemyModel`, utilizzato in seguito per il salvataggio del dataset nel database. Se l'operazione ha buon fine viene restituito il dataset creato, altrimenti viene restituito `None`.
  - `+ delete(self, id: UUID)`: elimina il dataset con l'id fornito. Tramite `DatasetSqlAlchemyModel` viene eliminato nel database il dataset che ha come id l'id ottenuto. Se l'operazione ha buon fine viene restituito l'id del dataset eliminato, altrimenti viene restituito `None`.
  - `+ update(self, dataset: Dataset)`: viene rinominato nel database il dataset. Tramite `DatasetSqlAlchemyModel` viene rinominato il dataset con il nuovo nome ottenuto da `dataset`. Se l'operazione ha buon fine viene restituito il dataset rinominato, altrimenti viene restituito `None`.
  - `+ get_all(self)`: si ottengono tutti i dataset salvati nel database. Tramite `DatasetSqlAlchemyMapper` e `DatasetSqlAlchemyModel` si ottiene la lista completa di tutti i dataset salvati nel database. Se l'operazione ha buon fine viene restituita la lista dei dataset, altrimenti viene restituito `None`.

  - `+ find_by_name(self, name: str)`: si ottiene il dataset salvato con il nome fornito. Tramite `DatasetSqlAlchemyMapper` e `DatasetSqlAlchemyModel` si ottiene il dataset salvato nel database che ha come campo nome `name`. Se l'operazione ha buon fine viene restituito il dataset ottenuto, altrimento viene restituito `None`.
  - `+ get_by_id(self, id: UUID)`: si ottiene il dataset salvato con l'id fornito. Tramite `DatasetSqlAlchemyMapper` e `DatasetSqlAlchemyModel` si ottiene il dataset salvato nel database che ha come campo id l'`id` ricevuto. Se l'operazione ha buon fine viene restituito il dataset ottenuto, altrimento viene restituito `None`.

- **Dipendenze**: DatasetRepository;

5.9 Design Pattern

Nella seguente sezione vengono descritti i *design pattern* utilizzati, indicando le motivazioni per cui sono stati utilizzati e in quali parti di codice.

5.9.1 Dependency Injection

- **Descrizione**: La Dependency Injection (DI) è un *design pattern* che facilita la gestione delle dipendenze tra componenti di un'applicazione, rendendole più flessibili, testabili e manutenibili. È una tecnica per fornire oggetti (dipendenze) a una classe dall'esterno, piuttosto che crearli al suo interno, l'oggetto riceve (viene "iniettato" con) le dipendenze di cui ha bisogno. L'obiettivo è quello di separare la creazione delle dipendenze dal loro utilizzo, rispettando il principio Inversion of Control (IoC): non è più l'oggetto a controllare come ottenere le sue dipendenze, ma qualcun altro gliele fornisce.

- **Utilizzo**:

  - **backend**: nel backend è stato utilizzato il $framework_G$ *Dependency Injector*. Questo $framework_G$ fornisce un *container* esplicito delle dipendenze, che può essere configurato per creare, gestire e risolvere oggetti e relazioni tra oggetti in maniera centralizzata. È stato definito un contenitore delle dipendenze, una classe centrale che si occupa di dichiarare e configurare tutti i servizi e componenti di cui l'applicazione ha bisogno. All'interno di questo contenitore sono stati specificati diversi provider, ognuno con un ruolo preciso: ad esempio, uno per leggere i parametri di configurazione esterni, uno per istanziare l'engine di connessione al database in modalità singleton (quindi condiviso), e altri per la

creazione del gestore delle sessioni e dei servizi dell'applicazione. Nelle *route* di *Flask* realizzate, grazie al decoratore `@Inject`, viene fornita automaticamente la dipendenza `DatasetUseCase` (interfaccia implementata da `DatasetService`), `Provide[AppContainer.dataset\_service]` fornisce l'oggetto `dataset\_service` che è stato registrato nel contenitore. In questo modo, non viene istanziato manualmente `DatasetUseCase` ma lo si riceve dal contenitore già configurato.

- **frontend**: nel frontend è stata utilizzata la gestione della dependency injection nativa del $framework_G$ *Angular*. Il `DatasetService` definito in *Angular* è stato annotato con `@Injectable` e definito come *Singleton* tramite `provideIn: root`, in questo modo può essere iniettato automaticamente nei componenti o in altri servizi. Questo ha permesso una gestione centralizzata della logica di business lato client e un'integrazione efficace con l'$API\ REST_G$ del backend.

5.9.2 Port e Adapter

- **Descrizione**: *Port e Adapter*, noto anche come *Hexagonal Architecture*, è un design pattern architetturale che realizza la separazione tra il *core* e le interfacce di input e otuput dell'applicazione. Questo pattern permette la separazione delle responsabilità mantenendo il *core* isolato, agevolando la manutenzione del sistema e la fase di testing;

- **Utilizzo**: vi è la separazione in:

  - **incoming**: la porta `DatasetUseCase` rappresenta le operazioni (gli *use case*) che l'applicazione espone al mondo esterno, consiste in un interfaccia che fornisce i metodi necessari per gestire i dati in ingresso ed è implementata dall'adapter `DatasetService`.
  - **outcoming**: la porta `DatasetRepository` rappresenta i punti attraverso cui la logica applicativa delega operazioni a sistemi esterni, viene realizzata tramite un'interfaccia ed è implementata dall'adapter `SqlAlchemyDatasetAdapter`. In questo modo l'applicazione rimane indipendente dai dettagli tecnici esterni, rendendo il sistema più testabile e facilmente manutenibile.

5.9.3 DTO (Data Transfer Object)

Qui vengono descritti i DTO utilizzati per il trasferimento dei dati tra gli strati dell'applicazione. I DTO sono dei contenitori semplici di dati non contenenti logica di business, progettati per aggregare dati e facilitare la comunicazione tra i vari livelli dell'applicazione.

- **DatasetCreateDto**:
  DTO utilizzato per rappresentare i dati necessari alla creazione di un nuovo dataset nel sistema.
  - `name: str`, nome del dataset da creare.
- **DatasetUpdateDto**:
  DTO utilizzato per rinominare un dataset esistente nel sistema.
  - `id: UUID`, identificatore univoco del dataset da aggiornare;
  - `name: str`, nuovo nome da assegnare al dataset.
- **DatasetResponseDto**:
  DTO utilizzato per rappresentare un singolo dataset restituito dal sistema.
  - `id: UUID`, identificatore univoco del dataset;
  - `name: str`, nome del dataset;
  - `creation_date: date`, data di creazione del dataset.
- **DatasetResponseListDto**:
  DTO utilizzato per rappresentare una lista di dataset presenti nel sistema.
  - `datasets: DatasetResponseDto[*]`, elenco di oggetti `DatasetResponseDto` che rappresentano i dataset disponibili.