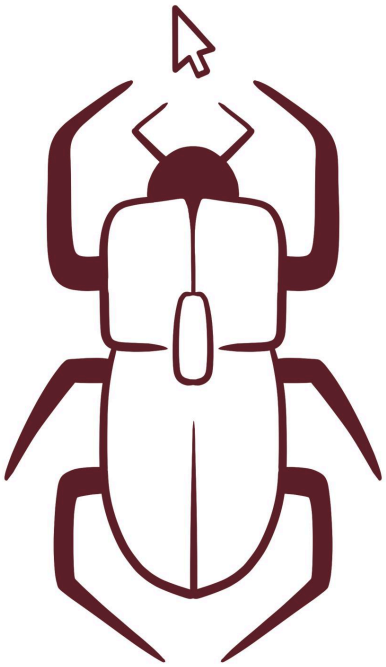

Specifica Tecnica

v1.0.0

Registro delle Modifiche

Data	Versione	Descrizione	Redattore	Verificatore
2026/04/22	1.0.0	Revisione per PB		Suar Alberto
2026/04/22	0.22.0	Aggiunti diagrammi complessivi di Account Microservice	Zago Alice	Suar Alberto
2026/04/22	0.21.0	Aggiunti diagrammi frontend e pattern repository	Suar Alberto	Basso Kevin
2026/04/22	0.20.0	Aggiunti diagrammi complessivi di Analysis Microservice	Basso Kevin	Suar Alberto
2026/04/22	0.19.0	Revisione Introduzione, Introduzione a Architettura di Deployment, Introduzione a Architettura Logica	Suar Alberto	Sandu Antonio
2026/04/22	0.18.0	Aggiunta la sezione di mappatura dei requisiti	Sandu Antonio	Suar Alberto
2026/04/22	0.17.0	Aggiunta sezione design patterns	Basso Kevin	Sgreva Andrea
2026/04/21	0.16.0	Completati i componenti della sezione presentation per Analysis Microservice	Sandu Antonio	Sgreva Andrea
2026/04/21	0.15.0	Completati i componenti della	Sgreva Andrea	Basso Kevin

Data	Versione	Descrizione	Redattore	Verificatore
		sezione infrastructure per Analysis Microservice		
2026/04/20	0.14.0	Completati i componenti delle sezioni domain e application per Analysis Microservice	Sgreva Andrea	Sandu Antonio
2026/04/19	0.13.0	Miglioramento parte frontend in seguito alle integrazioni con i microservizi di analisi e di autentificazione	Martinello Riccardo	Suar Alberto
2026/04/16	0.12.0	Aggiunta sezione Design Patterns per Account Microservice	Zago Alice	Suar Alberto
2026/04/13	0.11.0	Aggiunta architettura del Frontend e aggiornamento frameworks in sezione Tecnologie	Martinello Riccardo	Suar Alberto
2026/04/12	0.10.0	Aggiunti tutti i componenti mancanti per l'entità DocumentationReport di Analysis Microservice	Sgreva Andrea	Suar Alberto
2026/04/12	0.9.0	Aggiunta introduzione Account Microservice e	Zago Alice	Sgreva Andrea

Data	Versione	Descrizione	Redattore	Verificatore
		aggiornamento di alcuni componenti		
2026/04/09	0.8.0	Aggiunti tutti i componenti di Account Microservice	Zago Alice	Suar Alberto
2026/04/08	0.7.0	Aggiunti tutti i componenti di Analysis Microservice	Suar Alberto	Zago Alice
2026/04/06	0.6.0	Aggiunta sezione scelta tool per l'analisi della sicurezza	Sandu Antonio	Suar Alberto
2026/03/31	0.5.0	Stesura dei VO dell'Account Microservice	Zago Alice	Suar Alberto
2026/03/31	0.4.0	Completata Introduzione e aggiunti primi Command	Suar Alberto	Zago Alice
2026/03/30	0.3.0	Stesura delle tecnologie, dei VO e delle Entity dell'Analysis Microservice	Suar Alberto	Basso Kevin
2026/03/07	0.2.0	Prima strutturazione della sezione tecnologie	Berengan Riccardo	Suar Alberto
2026/03/04	0.1.0	Prima stesura del documento	Suar Alberto	Berengan Riccardo

Indice

Indice immagini

Indice tabelle

1 Introduzione

1.1 Scopo del Prodotto

Il presente documento descrive la Specifica Tecnica [G] relativa al progetto [G] Code Guardian [G], commissionato dall'azienda Var Group [G] e realizzato dal gruppo di studenti Skarab Group [G] nell'ambito del corso di Ingegneria del Software [G] presso l'Università degli Studi di Padova.

Il progetto ha come obiettivo la realizzazione di un sistema per l'automazione dei processi di audit [G] e remediation [G] delle vulnerabilità del software. L'architettura si basa sul paradigma degli agenti [G] software intelligenti, operanti in modo asincrono su repository [G] di codice sorgente.

La piattaforma supporta attività di analisi statica [G] del codice e individuazione delle principali criticità di sicurezza, fornendo suggerimenti di correzione automatizzati. Tale meccanismo è governato da modelli di linguaggio di grandi dimensioni (LLM [G]), integrati nel workflow per formulare e validare le modifiche senza compromettere l'integrità logica del software analizzato. La conformità del sistema è strettamente vincolata ai requisiti concordati e formalizzati nel documento di **Analisi dei Requisiti**.

1.1.1 Perimetro e Vincoli Operativi

Lo sviluppo del Minimum Viable Product [G] (MVP) di Code Guardian è soggetto a specifici vincoli operativi per garantire la compatibilità dell'interfaccia web sui principali ambienti operativi e di navigazione. Nello specifico, l'applicativo deve supportare i seguenti sistemi:

- **Sistemi Operativi:** Windows 10/11, macOS 14+, distribuzioni Linux (Ubuntu 22.04+);
- **Browser Web:** Google Chrome 120+, Mozilla Firefox 120+, Apple Safari 17+.

1.2 Finalità del Documento

Il presente documento traccia il passaggio dalla fase di analisi dei requisiti [G] alla progettazione architetturale e di dettaglio, definendo le scelte implementative necessarie alla realizzazione di Code Guardian. Costituisce il riferimento tecnico primario per il team di sviluppo e per gli stakeholder [G], perseguendo i seguenti obiettivi:

- definire l'architettura logica [G] del sistema, descrivendo la scomposizione in layer basata sul pattern esagonale [G] per garantire un elevato disaccoppiamento tra il dominio applicativo e le infrastrutture esterne;
- illustrare l'architettura di deployment [G], specificando i nodi di calcolo, la topologia di rete e le strategie di containerizzazione necessarie all'erogazione del servizio;
- formalizzare i design pattern [G] adottati (creazionali, strutturali e comportamentali), motivandone l'impiego per assicurare manutenibilità, modularità e testabilità del codice;
- definire le interfacce e i contratti di comunicazione, garantendo l'estensibilità del sistema verso l'integrazione di nuovi provider LLM o l'analisi di ulteriori linguaggi di programmazione;
- fornire una rappresentazione grafica mediante diagrammi UML [G], facilitando la comprensione delle dipendenze e del flusso dei dati;
- garantire la tracciabilità bidirezionale, dimostrando che ogni specifica individuata nell'**Analisi dei Requisiti** trovi riscontro in un'adeguata componente tecnica.

1.3 Destinatari del Documento

I principali destinatari di questo documento sono:

- i **Progettisti** [G] e i **Programmatori** [G] del gruppo Skarab Group, che lo utilizzeranno come linea guida prescrittiva per l'implementazione e l'integrazione del software;
- i **Verificatori** [G] del gruppo Skarab Group, per comprendere le dipendenze architetturali e pianificare adeguate strategie di test (unitari e di integrazione);

- l'azienda proponente **Var Group**, per valutare la coerenza delle scelte tecnologiche e progettuali rispetto alle richieste del capitolato d'appalto;
- i committenti del progetto, **Prof. Tullio Vardanega** e **Prof. Riccardo Cardin**, per la valutazione accademica inerente alla correttezza formale e architetturale del sistema progettato.

1.4 Struttura del Documento

Il documento è organizzato nei seguenti capitoli principali:

- **1. Introduzione:** definisce lo scopo, i destinatari, la struttura del documento e i riferimenti normativi e informativi utilizzati.
- **2. Tecnologie Adottate:** specifica i linguaggi di programmazione, i framework e gli strumenti di supporto scelti per la realizzazione dei diversi layer applicativi.
- **3. Architettura di Deployment:** illustra la distribuzione fisica e logica del sistema, includendo la descrizione dell'architettura esagonale adottata e la mappatura sui nodi di calcolo.
- **4. Architettura Logica:** approfondisce la progettazione di dettaglio tramite diagrammi delle classi, l'applicazione dei design pattern e la spiegazione funzionale dei singoli componenti del sistema.
- **5. Mappatura dei Requisiti:** documenta lo stato di soddisfacimento dei requisiti definiti in fase di analisi.

1.5 Glossario

Al fine di prevenire ambiguità interpretative, è stato redatto un glossario che definisce in modo univoco la terminologia tecnica, gli acronimi e i concetti di dominio utilizzati all'interno della documentazione.

Nel testo, **ogni termine evidenziato tramite la lettera G come apice** (reso graficamente tramite apposita formattazione), rimanda alla voce corrispondente del Glossario pubblicato sul sito ufficiale del gruppo, consentendo al lettore di accedere direttamente alla definizione associata.

La versione più recente del Glossario è disponibile al seguente link:
Link al Glossario (v2.0.0).

1.6 Riferimenti

1.6.1 Riferimenti Normativi

I seguenti documenti hanno valore vincolante per la redazione della Specifica Tecnica:

- **Capitolato C2**: Piattaforma ad agenti per l'audit e la remediation dei repository software.
  https://www.math.unipd.it/~tullio/IS-1/2025/Progetto/C2.pdf
  (ultimo accesso: **22/04/2026**)
- **Analisi dei Requisiti**: insieme dei requisiti e dei casi d'uso [G] coperti nel Minimum Viable Product.
  https://skarabgroup.github.io/DocumentazioneProgetto/PB/AdR.pdf
  (versione: **v2.0.0**)
- **Norme di Progetto**: regole, convenzioni e standard di qualità adottati dal gruppo.
  https://skarabgroup.github.io/DocumentazioneProgetto/PB/NdP.pdf
  (versione: **v2.0.0**)

1.6.2 Riferimenti Informativi

- **Standard IEEE/ISO/IEC 42010-2022**: International Standard for Software, systems and enterprise – Architecture description

https://ieeexplore.ieee.org/document/9938446
(ultimo accesso: **22/04/2026**)

- **Dispense del Corso di Ingegneria del Software**:
  Progettazione Software
  (ultimo accesso: **22/04/2026**)
  Dependency Injection
  (ultimo accesso: **22/04/2026**)
  Dependency Management in Object-Oriented Programming
  (ultimo accesso: **22/04/2026**)
  Diagrammi delle Classi
  (ultimo accesso: **22/04/2026**)
  Diagrammi delle Attività
  (ultimo accesso: **22/04/2026**)
  Pattern Architetturali
  (ultimo accesso: **22/04/2026**)
  Model-View Patterns
  (ultimo accesso: **22/04/2026**)
  Design Pattern Creazionali
  (ultimo accesso: **22/04/2026**)
  Design Pattern Strutturali
  (ultimo accesso: **22/04/2026**)
  Design Pattern Comportamentali
  (ultimo accesso: **22/04/2026**)

2 Tecnologie

2.1 Linguaggi di Programmazione

Tecnologia	Versione	Motivazioni
TypeScript	5.9.3	TypeScript [G] è il linguaggio principale adottato per lo sviluppo dei microservizi backend [G] e del frontend [G]. La tipizzazione statica forte consente di rilevare errori a tempo di compilazione anziché a runtime, riducendo il rischio di regressioni e rendendo i contratti tra componenti espliciti e verificabili staticamente. Questa caratteristica è particolarmente rilevante in un'architettura a microservizi dove le interfacce tra moduli devono essere chiare e stabili nel tempo. La compatibilità con l'intero ecosistema Node.js garantisce accesso a un vasto insieme di librerie mature e mantenute attivamente.
Python	3.12.3	Python [G] è il linguaggio adottato per la componente agentica del sistema, ospitata all'interno del microservizio di analisi. La scelta è motivata dalla maturità e dall'ampiezza del suo ecosistema nel dominio dell'intelligenza artificiale: Python dispone delle librerie più aggiornate per l'integrazione con modelli di linguaggio di grandi dimensioni (LLM) e per la costruzione di sistemi agentici complessi e pipeline di machine learning.

2.2 Framework e Librerie

Tecnologia	Versione	Motivazioni
NestJS	11.0.16	NestJS [G] è il framework adottato per i microservizi TypeScript. La scelta è motivata da molteplici fattori rispetto ad alternative più minimali come Express. NestJS è un framework completo che impone una struttura modulare ben definita. Ogni modulo incapsula un dominio funzionale coeso, favorendo la separazione delle responsabilità. Il supporto nativo alla Dependency Injection, basato su decoratori, consente di dichiarare le dipendenze in modo esplicito, meccanismo che si sposa perfettamente con l'architettura Ports & Adapters adottata: le porte vengono definite come interfacce e gli adapter come implementazioni concrete, iniettate dal container.
AWS Strands	Managed	AWS [G] Strands è il framework adottato per la definizione e l'orchestrazione degli agenti software. Fornisce le primitive necessarie per integrare i modelli LLM all'interno di flussi agentici strutturati, gestendo il ciclo di vita degli agenti, la comunicazione e la composizione dei tool a loro disposizione per l'analisi e la remediation.
React	19.0.0	React [G] è una libreria JavaScript per la costruzione di interfacce utente basata su componenti dichiarativi e aggiornamenti reattivi tramite

Tecnologia	Versione	Motivazioni
		Virtual DOM. Costituisce la base architetturale di tutte le interfacce, viste e comportamenti dinamici dell'applicazione lato client.
Vite	6.0.0	Build tool e dev server ad altissime prestazioni per progetti moderni. Offre Hot Module Replacement (HMR) fulmineo e un sistema avanzato di proxy per il forwarding delle richieste HTTP verso i microservizi durante lo sviluppo e testing locale.
React Router	7.0.0	Libreria standardizzata per il routing client-side in Single Page Application. Gestisce la navigazione tra viste pubbliche e protette applicando il pattern *nested routes*, consentendo cambi di URL [G] e renderizzazione dinamica dei contenuti senza ricaricamento del browser.

2.3 Servizi Esterni (Cloud e Infrastruttura)

Tecnologia	Versione	Motivazioni
MongoDB Atlas	Managed	MongoDB [G] Atlas è il sistema di persistenza adottato per il microservizio di analisi. La motivazione principale risiede nella natura eterogenea dei report prodotti: i risultati di audit variano pesantemente per struttura a seconda del tipo di analisi e dei tool coinvolti, rendendo inadeguato uno schema relazionale rigido. Un document store permette di persistere queste entità senza complesse migrazioni di schema. La versione gestita Atlas elimina l'onere amministrativo di provisioning e backup.
Amazon RDS	Managed	Amazon RDS è il sistema di persistenza adottato per il microservizio dedicato alla gestione delle credenziali e degli account. A differenza dei risultati eterogenei dell'analisi, le entità di gestione utenti [G] possiedono una struttura relazionale rigorosa, con vincoli di integrità forti. RDS garantisce consistenza transazionale (ACID) e integrità referenziale ottimali. Essendo un servizio completamente gestito, automatizza i backup, le patch e la disponibilità Multi-AZ.
Amazon ECS	Managed	Amazon Elastic Container Service [G] (ECS) è il servizio di orchestrazione dei container adottato per ospitare il microservizio di analisi. Garantisce un elevato livello di flessibilità e controllo nella gestione dei carichi di lavoro dockerizzati. L'adozione di ECS consente un'allocazione granulare delle risorse di calcolo e un'integrazione fluida e sicura con il resto dei servizi infrastrutturali dell'ecosistema AWS, garantendo un'efficiente scalabilità orizzontale in base alla mole di analisi richieste.
Amazon S3	Managed	Amazon Simple Storage Service [G] (S3) è utilizzato come storage a oggetti altamente scalabile in molteplici contesti dell'applicativo. Per

Tecnologia	Versione	Motivazioni
		il motore di analisi, S3 opera come “Data Staging Layer” sicuro per ospitare il codice sorgente scaricato prima della fase di ispezione, garantendo così l’immutabilità del dataset in elaborazione e disaccoppiando l’acquisizione logica dal calcolo. Inoltre, S3 è utilizzato come origin storage per servire gli asset e le build statiche del frontend applicativo web.
Amazon CloudFront	Managed	Amazon CloudFront funge da Content Delivery Network (CDN) primaria del sistema, occupandosi della distribuzione dell’interfaccia utente web (la Single Page Application in React) con la minima latenza e la massima larghezza di banda. L’integrazione nativa con S3 assicura tempi di caricamento istantanei a livello globale e fornisce un layer fondamentale di sicurezza esterna bloccando attacchi DDoS e applicando la terminazione TLS per le connessioni cifrate HTTPS.
Amazon CloudWatch	Managed	Amazon CloudWatch funge da cruscotto centralizzato per il monitoraggio operativo (osservabilità) e la gestione log dell’intera infrastruttura cloud. Raccogliendo automaticamente stream di dati e metriche di utilizzo da ECS, RDS e dagli applicativi, permette al team di avere visibilità in tempo reale sullo stato di salute del backend. Questa integrazione consente di instaurare alert per l’individuazione e risoluzione celere di anomalie prestazionali durante l’audit.
Amazon App Runner	Managed	AWS App Runner è utilizzato per l’hosting del microservizio di gestione delle credenziali. Offre un livello di astrazione superiore rispetto ai classici orchestratori: il deployment avviene direttamente da un’immagine container senza necessità di configurare bilanciatori di carico o task definition complesse, rendendolo ideale per servizi con endpoint HTTP classici e requisiti operativi diretti.

2.4 Librerie e Strumenti Frontend

Tecnologia	Versione	Descrizione
Tailwind CSS	4.0.0	Framework CSS utility-first. Permette di definire stili direttamente come classi semantiche inline nell’HTML, eliminando la necessità di file CSS separati e garantendo consistenza visiva attraverso un design system altamente scalabile.
shadcn/ui + Radix UI	—	Set di componenti UI modulari, headless e accessibili. Fornisce primitive interattive come modali, tab, bottoni, barra di progressione ed elementi form integrati coerentemente nello stile visivo del progetto.

Tecnologia	Versione	Descrizione
Axios	1.7.9	Client HTTP basato su Promise per chiamate REST asincrone. È stato integrato attraverso interceptor personalizzati per la gestione dell'autenticazione Bearer e il rinfresco proattivo dei token alla ricezione di codici 401 Unauthorized.
Zod	3.24.2	Libreria robusta per la dichiarazione e validazione tipizzata di schemi dati. Previene invii di richieste non valide intercettando gli input utente errati in modo formale direttamente lato client, restituendo feedback mirato.
Recharts	2.15.0	Libreria performante per la visualizzazione dei dati analitici sotto forma di grafici interattivi SVG su DOM React. Sfruttata diffusamente nei cruscotti per tracciare temporalmente la qualità e sicurezza dei repository scansionati.
Sonner	1.7.0	Libreria dedicata al sistema unificato di notifiche (*toast*) a comparsa. Veicola i micro-feedback transazionali per gli utenti (successo analisi, salvataggi, disconnessioni di rete) senza risultare invasivo nell'esperienza d'uso.

2.5 Strumenti di analisi agentica

2.5.1 Strumenti per l'Agente di Sicurezza

Tecnologia	Versione	Motivazioni
Semgrep CE	Gestita via pip	Semgrep CE (Community Edition) è uno strumento di Static Application Security Testing [G] (SAST) open source integrato via Python pip. Trova vulnerabilità e pattern pericolosi nel codice sorgente ed è in grado di rilevare il livello di aderenza di un repository alla conformità OWASP [G] Top 10 tramite regole strutturate. Si rivela rapido nell'analisi, copre linguaggi multipli e permette l'estrazione di report in formati parsabili (JSON) necessari all'integrazione con gli agenti.
Trivy	Latest (sh script)	Trivy è uno scanner universale, installato tramite script di shell, adottato per rilevare segreti hardcoded [G] (credenziali DB, chiavi AWS, token) inavvertitamente committati nel repository. Sfrutta meccanismi di detection espandibili basati su un database costantemente aggiornato e su espressioni regolari. Come per gli altri strumenti della suite, il risultato restituito come JSON è indispensabile al motore d'analisi per strutturare una remediation appropriata all'esposizione.

Tecnologia	Versione	Motivazioni
Syft	Latest (sh script)	Syft è uno strumento specializzato che analizza nativamente i file di configurazione dei repository per generare una Software Bill of Materials [G] (SBOM). Adottare un approccio SBOM-based svincola l'ispezione architetturale dalla dipendenza da specifici manifest o lockfile: Syft distilla un elenco formale di ogni dipendenza e lo standardizza per i successivi controlli di vulnerabilità.
Grype	Latest (sh script)	Grype è uno scanner di vulnerabilità di terze parti, disegnato per agire in sinergia formale con Syft. Riceve l'output SBOM da quest'ultimo ed esamina le librerie confrontandole con i database di vulnerabilità noti (CVE [G]). Il design combinato Syft+Grype massimizza l'efficienza rispetto agli analizzatori monolitici, fornendo riscontri isolati unicamente sulle dipendenze introdotte nel progetto.

2.5.2 Strumenti per l'Agente di Documentazione

Tecnologia	Versione	Motivazioni
Spectral CLI	Gestita via npm	Spectral (`@stoplight/spectral-cli`) è uno strumento avanzato di linting e validazione adottato specificamente per le interfacce API e file di specifica. Introdotto nel loop degli audit documentali per assicurare la consistenza e la robustezza del design dell'API, analizza le specifiche applicando regole stringenti (*ruleset*) che bloccano pattern insicuri o mancanze nella documentazione tecnica.
Repomix	Gestita via npm	Repomix è la tecnologia fondamentale adottata per preparare ed aggregare il materiale sorgente in formato "*AI-friendly*". Prima che gli agenti IA scansionino i sorgenti integrali, Repomix aggrega e compatta molteplici file di progetto limitando le astrazioni superflue. Questo strumento è imperativo per ridurre radicalmente il consumo di token e mantenere l'input focalizzato entro la *context window* concessa all'LLM incaricato di orchestrare l'aggiornamento documentale.

2.5.3 Strumenti per l'Agente di Codice

Tecnologia	Versione	Motivazioni
PMD	7.0.0	PMD è un analizzatore statico multi-linguaggio scaricato in versione standalone ed eseguito tramite Java Runtime Environment (`openjdk-21-jre-headless`). Identifica difetti di programmazione comuni, come variabili non utilizzate, blocchi catch vuoti, e complessità ciclomatica elevata. Viene integrato per fornire all'agente un set di ispezioni statiche di base su linguaggi supportati, garantendo audit strutturali al codice sorgente.

Tecnologia	Versione	Motivazioni
Biome	Gestita via npm	Biome (`@biomejs/biome`) è uno strumento ad alte prestazioni che combina funzionalità di formattazione e linting per l'ecosistema web (JavaScript, TypeScript, JSON, ecc.). È stato integrato nell'agente di codice per eseguire check rapidissimi sulla sintassi e sulla stilistica del codice prima e dopo le potenziali modifiche dell'LLM.
ESLint	Gestita via npm	ESLint è lo strumento standard di analisi statica per identificare pattern problematici nel codice JavaScript/TypeScript. L'agente di codice se ne serve per validare la correttezza semantica e far rispettare le regole di qualità del software prima di proporre e approvare una correzione definitiva.
Jest	Gestita via npm	Jest è un framework di testing completo. Viene fornito all'agente di codice per abilitare la validazione automatica delle modifiche: l'agente può invocare l'esecuzione delle test suite esistenti nel repository analizzato per verificare che le proprie *remediation* non abbiano introdotto regressioni comportamentali.
TypeScript CLI	Gestita via npm	Il compilatore TypeScript (`typescript`) è incluso per permettere all'agente di eseguire la validazione dei tipi (*type-checking*) sul codice sorgente. Garantisce che le correzioni proposte per basi di codice TS rispettino rigorosamente i vincoli di tipizzazione definiti nel progetto analizzato.
pnpm e Yarn	Gestite via npm	Gestori di pacchetti alternativi a npm installati a livello globale. La loro presenza all'interno dell'ambiente di esecuzione dell'agente assicura che il sistema sia in grado di interpretare, risolvere le dipendenze ed eseguire script indipendentemente dal package manager adottato nativamente dal repository target.

2.6 Strumenti di Testing

Tecnologia	Versione	Motivazioni
Jest	Gestita via npm	Indipendentemente dall'utilizzo interno all'agente di codice, Jest è il framework ufficiale adottato dal team di sviluppo per la stesura e l'esecuzione dei test di unità e di integrazione sui microservizi in NestJS e sul frontend. Costituisce il core delle *quality gates* all'interno delle pipeline di Continuous Integration (CI/CD): la corretta esecuzione della test suite e il rispetto della configurazione dei plugin (`jestPlugin.configs.recommended.rules`) sono vincolanti per la validazione della build e per il successivo deployment in produzione del sistema Code Guardian.

3 Architettura

3.1 Architettura di Deployment

Il sistema Code Guardian è stato progettato per operare interamente in ambiente cloud, adottando un approccio containerizzato basato sul paradigma a microservizi. La topologia di deployment sfrutta nativamente i servizi gestiti di Amazon Web Services (AWS) per garantire scalabilità orizzontale, alta disponibilità e una netta separazione delle responsabilità (Separation of Concerns).

3.1.1 Adozione del Paradigma a Microservizi

La scomposizione del backend nelle due unità computazionali indipendenti (Microservizio Account e Microservizio Analysis) deriva da una rigorosa valutazione dei requisiti non funzionali e delle criticità intrinseche al dominio applicativo. Questa consapevolezza si riflette in molteplici vantaggi sistemici:

- **Scalabilità Asimmetrica:** Il dominio della gestione utenti (Account) è caratterizzato da un traffico leggero, costante e prevedibile. Al contrario, il dominio dell'analisi (Analysis) è soggetto a picchi di carico improvvisi, dovuti al download di repository massivi e all'orchestrazione degli agenti. Disaccoppiare i servizi permette ad App Runner di scalare orizzontalmente solo il modulo sotto stress, ottimizzando i costi operativi (FinOps).
- **Isolamento dei Guasti (Fault Isolation):** La netta separazione degli ambienti di esecuzione su App Runner confina le eventuali criticità al singolo dominio applicativo. Qualora il Microservizio Analysis subisca un'anomalia grave (ad esempio, un *memory leak* causato dall'estrazione di un pacchetto ZIP malformato o un timeout verso le API di GitHub), l'errore non si propaga al resto del sistema. Il Microservizio Account permane integro e pienamente operativo, garantendo che gli utenti possano continuare ad autenticarsi, consultare il proprio profilo e interagire con le funzionalità di storico della piattaforma senza percepire un disservizio generalizzato.
- **Allineamento al Domain-Driven Design (DDD):** I due microservizi rappresentano dei *Bounded Context* rigorosi. Ognuno possiede il proprio database esclusivo (RDS per Account, MongoDB per Analysis), prevenendo la creazione di anti-pattern come il database condiviso. Questo isolamento garantisce che i contratti dei dati possano evolvere indipendentemente senza causare regressioni a cascata.

3.1.2 Topologia dell'Infrastruttura

3.1.2.1 Frontend e Content Delivery (S3 + CloudFront)

L'interfaccia utente è implementata come una Single Page Application (SPA) sviluppata in React e buildata tramite Vite. Essendo un'applicazione client-side, il processo di build genera esclusivamente artefatti statici (HTML, CSS, JS minificati e asset).

Il deployment del frontend si basa su due servizi AWS orchestrati in sinergia:

- **Amazon S3 (Simple Storage Service):** funge da origin storage immutabile per gli artefatti della build. La scelta di S3 garantisce durabilità e un costo di archiviazione trascurabile, eliminando la necessità di avere un web server computazionale dedicato in esecuzione per servire il client.
- **Amazon CloudFront:** funge da Content Delivery Network (CDN) globale e rappresenta l'unico *entrypoint* pubblico dell'intero sistema. CloudFront garantisce una latenza minima memorizzando in cache gli asset nelle *Edge Location*, agisce da reverse proxy unificato annullando le problematiche CORS (instradando `/api/*` ai backend) e gestisce centralmente la terminazione TLS (HTTPS).

3.1.2.2 Microservizi Core (ECR + App Runner)

Il deployment dei due microservizi in NestJS è gestito attraverso la seguente catena:

- **Amazon ECR (Elastic Container Registry):** registro privato per le immagini Docker. Le pipeline di Continuous Integration effettuano il push delle immagini compilate su ECR, garantendo il versionamento semantico e un repository unico per gli ambienti di test e produzione.
- **AWS App Runner:** servizio di orchestrazione fully-managed scelto per l'esecuzione. I Controller NestJS implementati nel progetto espongono un'architettura puramente RESTful e *stateless*. App Runner astrae la complessità del bilanciamento del carico, istanziando dinamicamente i container necessari. I Controller fungono da **Primary Adapters** (Ports & Adapters): intercettano l'HTTP, validano i payload e traducono la richiesta in comandi puri per il Domain Core.

3.1.2.3 Infrastruttura Agentica e di Analisi Asincrona (Amazon ECS + S3)

Il cuore computazionale di Code Guardian è delegato a un'infrastruttura separata basata su **Amazon ECS (Elastic Container Service)**, garantendo che le operazioni intensive non degradino le performance delle API.

Gli agenti seguono un modello a container effimeri (*ephemeral tasks*):

- **Isolamento e Sandbox:** Gli agenti (Sicurezza, Documentazione, Codice) sono racchiusi in immagini Docker Python distinte. Quando l'Analysis Service riceve una richiesta, avvia un Task ECS dedicato su un cluster serverless. Ogni task opera in una sandbox di memoria e rete isolata, prevenendo contaminazioni tra i repository di clienti diversi.
- **Data Staging Layer (Amazon S3):** Per l'elaborazione, si adotta il pattern "*Claim Check*". Il Microservizio Analysis carica il repository target su un bucket S3 adibito a staging. Il Task ECS riceve solo l'URI: l'agente scarica i sorgenti nel proprio volume effimero, esegue i tool (es. Semgrep, PMD), compatta il contesto con Repomix e interroga i provider LLM tramite AWS Strands.
- **Persistenza e Ritorno:** Al termine del flusso, l'agente Python **non** appesantisce il database documentale inserendo file massivi. Al contrario, salva i log dettagliati, i report di audit e i diff delle remediation direttamente sotto forma di artefatti strutturati in **Amazon S3**. Per rilevare la fine delle operazioni non si utilizzano meccanismi di push, bensì una logica di *polling* governata dal Microservizio Analysis: un **Secondary Adapter** dedicato rimane in ascolto interrogando periodicamente lo stato del task infrastrutturale. Una volta rilevato il completamento dell'esecuzione e la disponibilità degli artefatti su S3, l'adapter informa il dominio che provvede ad aggiornare lo stato dell'analisi su MongoDB, mentre il container ECS effimero viene liberato.

3.1.2.4 Configurazioni Infrastrutturali Esterne (VPC e IAM)

Per completezza, è imperativo citare elementi di sicurezza essenziali che, pur non essendo componenti di "codice" o microservizi esplicitamente rappresentati nel ciclo di deployment applicativo, costituiscono l'infrastruttura di base (spesso definita tramite approcci *Infrastructure as Code*):

- **Virtual Private Cloud (VPC):** Le configurazioni di rete, incluse le subnet private e i Security Group, isolano i database (RDS e Atlas) dalla rete Internet pubblica. Si tratta di un'impostazione a livello di account AWS che prescinde dal codice dell'applicativo.
- **Identity Access Management (IAM):** La gestione dei permessi segue il principio del *Least Privilege*. I ruoli IAM (ad esempio, il `TaskExecutionRole` che permette all'agente su ECS di scrivere esclusivamente sul bucket S3 di destinazione) sono astrazioni gestionali di AWS e non blocchi di software in esecuzione.

3.1.2.5 Flusso Operativo e di Integrazione

Il diagramma architetturale traccia la sequenza operativa end-to-end di un'attività di audit, riassumibile nelle seguenti fasi:

1. **Interazione Utente:** L'utente accede alla piattaforma scaricando la SPA React via CloudFront.
2. **API Requests:** Le operazioni di business vengono instradate ai microservizi (`/api/auth`, `/api/analysis`).
3. **Acquisizione Metadati:** L'Analysis Service interroga le API di GitHub per recuperare lo stato del repository.
4. **Staging del Codice:** L'Analysis Service scarica il sorgente e lo deposita su S3, separando i dati grezzi dalla logica computazionale.
5. **Trigger Task:** Viene invocata l'API di ECS per istanziare un container agentico usa-e-getta.
6. **Pull Repo Zip:** Il Task ECS avvia l'agente Python, che recupera il pacchetto sorgente da S3.
7. **Prompt e Validazione:** L'agente genera il contesto e interroga il modello LLM.
8. **Persistenza Artefatti:** L'agente termina le operazioni caricando i report finali e le remediation direttamente su **Amazon S3** ed emettendo l'evento di conclusione.

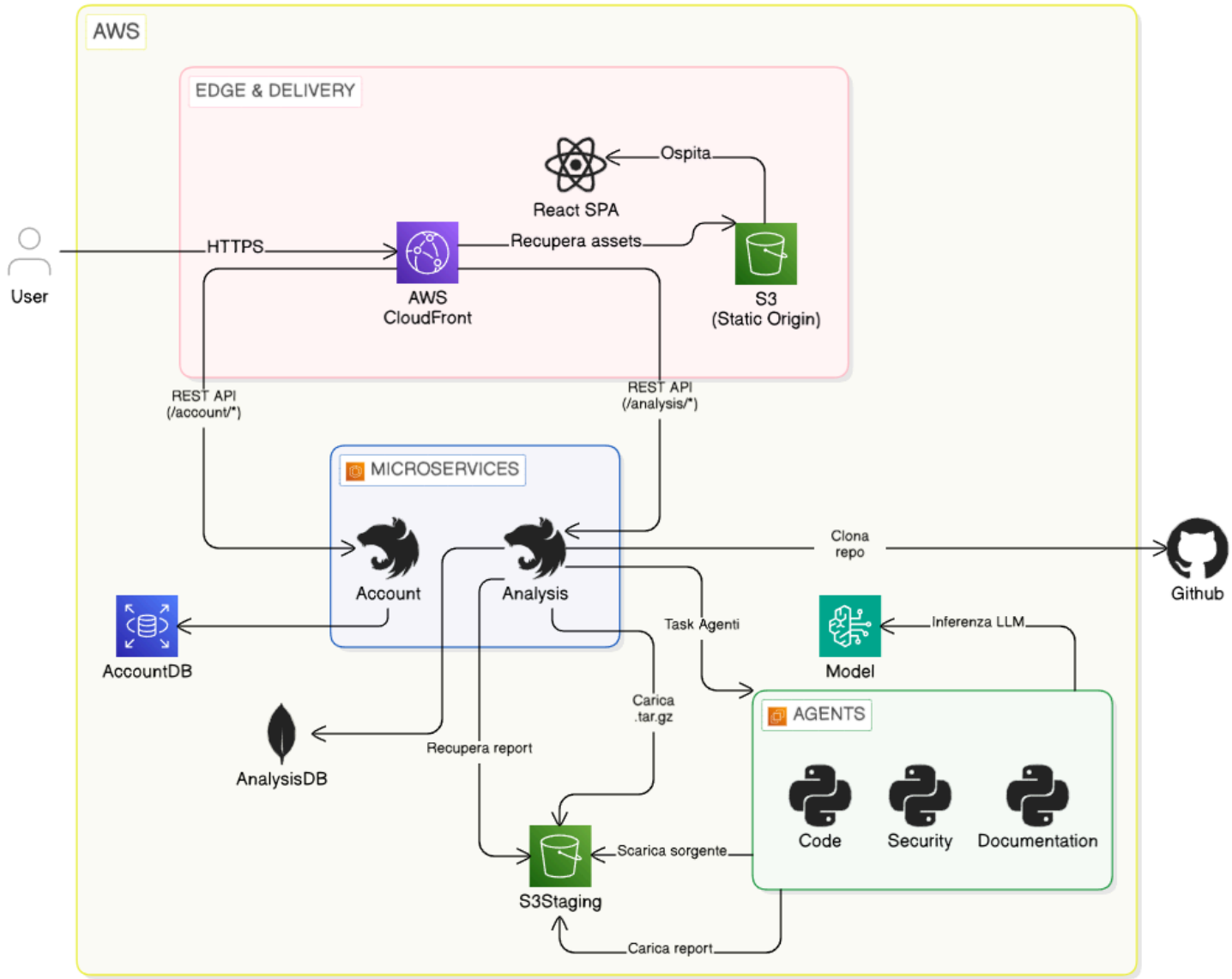

Figure 1: Diagramma UML dell'Architettura di Deployment

3.2 Architettura Logica

3.2.1 Microservizio Analysis

Il Microservizio **Analysis**, sviluppato in TypeScript tramite il framework NestJS, è responsabile della gestione e coordinamento del flusso di analisi all'interno del sistema Code Guardian. Esso incapsula la logica di audit e validazione delle vulnerabilità, fungendo da punto di aggregazione tra input esterni, logica di dominio e servizi infrastrutturali.

Al fine di gestire la complessità derivante dall'integrazione con servizi cloud eterogenei e garantire elevata manutenibilità ed evolvibilità nel tempo, l'architettura logica è stata progettata seguendo il pattern **Ports and Adapters** (Architettura Esagonale).

3.2.1.1 Motivazioni Architetturali

L'adozione di tale pattern è guidata da esigenze specifiche del dominio applicativo e da considerazioni ingegneristiche:

- **Agnosticismo Tecnologico (Framework Independence):** Le regole di business relative all'analisi del codice e alla validazione delle vulnerabilità sono completamente indipendenti da NestJS, dai driver di persistenza (es. MongoDB) e dagli SDK cloud (es. AWS). Ciò consente di sostituire o aggiornare le tecnologie infrastrutturali senza impattare il dominio.
- **Testabilità Isolata (Shift-Left Testing):** La definizione esplicita delle porte consente di testare i casi d'uso in isolamento, sostituendo gli adapter reali con implementazioni fittizie (Mock e Stub). Questo approccio abilita test unitari rapidi e deterministici nelle pipeline CI/CD, eliminando la dipendenza da risorse esterne.
- **Inversione delle Dipendenze (Dependency Inversion):** In conformità ai principi SOLID, il flusso delle dipendenze è orientato verso l'interno: il dominio definisce le astrazioni (porte), mentre gli adapter le implementano. Questo disaccoppia completamente la logica di business dai dettagli tecnici.
- **Evolvibilità del Dominio:** Il sistema Code Guardian è progettato per integrare nel tempo nuovi strumenti di analisi (statici e dinamici) e nuovi provider infrastrutturali. L'architettura esagonale consente di introdurre nuovi adapter senza modificare i casi d'uso esistenti, preservando la stabilità del core applicativo.
- **Isolamento delle Integrazioni Esterne:** Le interazioni con sistemi esterni (storage, servizi di analisi, orchestrazione cloud) sono confinate negli adapter, riducendo l'impatto di cambiamenti o fault esterni sul dominio.
- **Trade-off Architetturali:** L'adozione del pattern introduce un overhead strutturale dovuto alla presenza di interfacce, adapter e livelli di astrazione aggiuntivi. Tuttavia, tale complessità è giustificata dalla necessità di garantire scalabilità, manutenibilità ed evoluzione controllata del sistema nel lungo periodo.

3.2.1.2 Scomposizione dei Livelli

L'architettura è organizzata in tre aree concentriche, separate da confini ben definiti:

3.2.1.2.1 Domain Core (Logica di Business)

Rappresenta il cuore dell'architettura e non dipende da alcun framework o libreria esterna.

Contiene:

- **Entità e Modelli:** Strutture dati pure che rappresentano i concetti del dominio e ne incapsulano le invarianti.

- **Casi d'Uso (Use Cases):** Servizi applicativi che implementano la logica operativa. Coordinano il flusso di analisi trasformando input in output, senza effetti collaterali diretti verso l'esterno.
- **Ports (Interfacce):** Contratti che definiscono le modalità di interazione:
  - **Primary Ports:** espongono le operazioni disponibili agli adapter in ingresso
  - **Secondary Ports:** definiscono i servizi richiesti dal dominio (persistenza, storage, orchestrazione)

3.2.1.2.2 Primary Adapters (Driving Adapters)

Costituiscono i punti di ingresso del sistema e hanno il compito di tradurre le richieste esterne in invocazioni ai casi d'uso.

Responsabilità principali:

- Ricezione input (HTTP, eventi, messaggi)
- Validazione formale dei dati (DTO)
- Invocazione dei casi d'uso tramite le Primary Ports
- Formattazione della risposta verso il chiamante

Questo livello dipende dal framework (NestJS), ma il dominio ne rimane completamente isolato.

3.2.1.2.3 Secondary Adapters (Driven Adapters)

Implementano concretamente i servizi richiesti dal dominio attraverso le Secondary Ports.

Comprendono:

- **Adapter di Persistenza:** Gestiscono la traduzione tra entità di dominio e modelli di database.
- **Adapter di Integrazione:** Incapsulano la comunicazione con API esterne e strumenti di analisi.
- **Adapter di Storage:** Gestiscono il trasferimento e la gestione dei file.
- **Adapter di Orchestrazione:** Traducono le richieste del dominio in operazioni su infrastrutture asincrone o sistemi distribuiti.

Questi componenti rappresentano l'unico punto in cui vengono utilizzate librerie specifiche o SDK esterni.

3.2.1.3 Flussi completi di Esecuzione

Il diagramma seguente illustra un flusso operativo completo, evidenziando l'interazione tra i livelli dell'architettura esagonale a partire da un singolo controller HTTP fino alla conclusione della richiesta. Questa sezione ha l'obiettivo di fornire una panoramica ad alto livello del percorso di esecuzione, evidenziando i passaggi chiave e le interazioni tra i componenti, senza entrare nei dettagli di implementazione specifici, interazioni con oggetti di dominio o contratti tra la parti, in quanto questo livello di dettaglio sará visibile nelle sezioni successive.

3.2.1.3.1 Analysis

Il flusso che segue rappresenta la sequenza operativa di un'analisi completa, partendo dalla ricezione della richiesta HTTP fino alla conclusione dell'audit e alla restituzione dei risultati all'utente. Nota: la risposta viene ritornata all'utente immediatamente dopo la fase di staging su S3, mentre l'esecuzione dell'agente e la generazione dei report avvengono in modo asincrono.

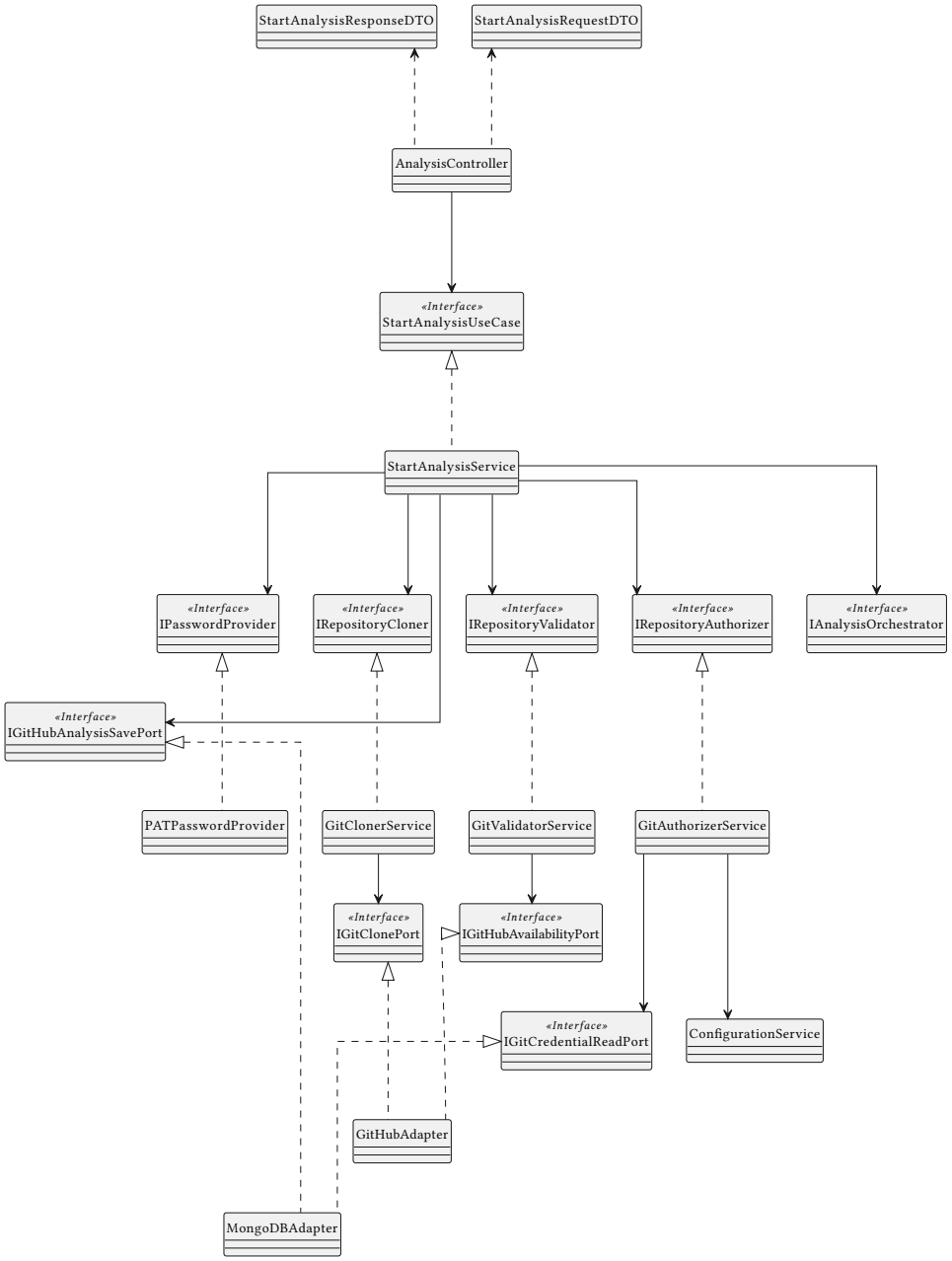

Figure 2: Controller Diagram — AnalysisControllerReachableClasses

Il diagramma nel diagramma sopra non é presente il flusso di orchestrazione agentica in quanto, come giá esposto, l'esecuzione dell'agente avviene in modo asincrono e non blocca la risposta HTTP. Il controller si limita a orchestrare le operazioni sincrone (interazione con GitHub, staging su S3) e a delegare l'orchestrazione degli agenti all'OrchestratorService, senza attendere il completamento di quest'ultima per rispondere all'utente. Per questo motivo, il flusso di orchestrazione agentica é rappresentato in un diagramma a parte.

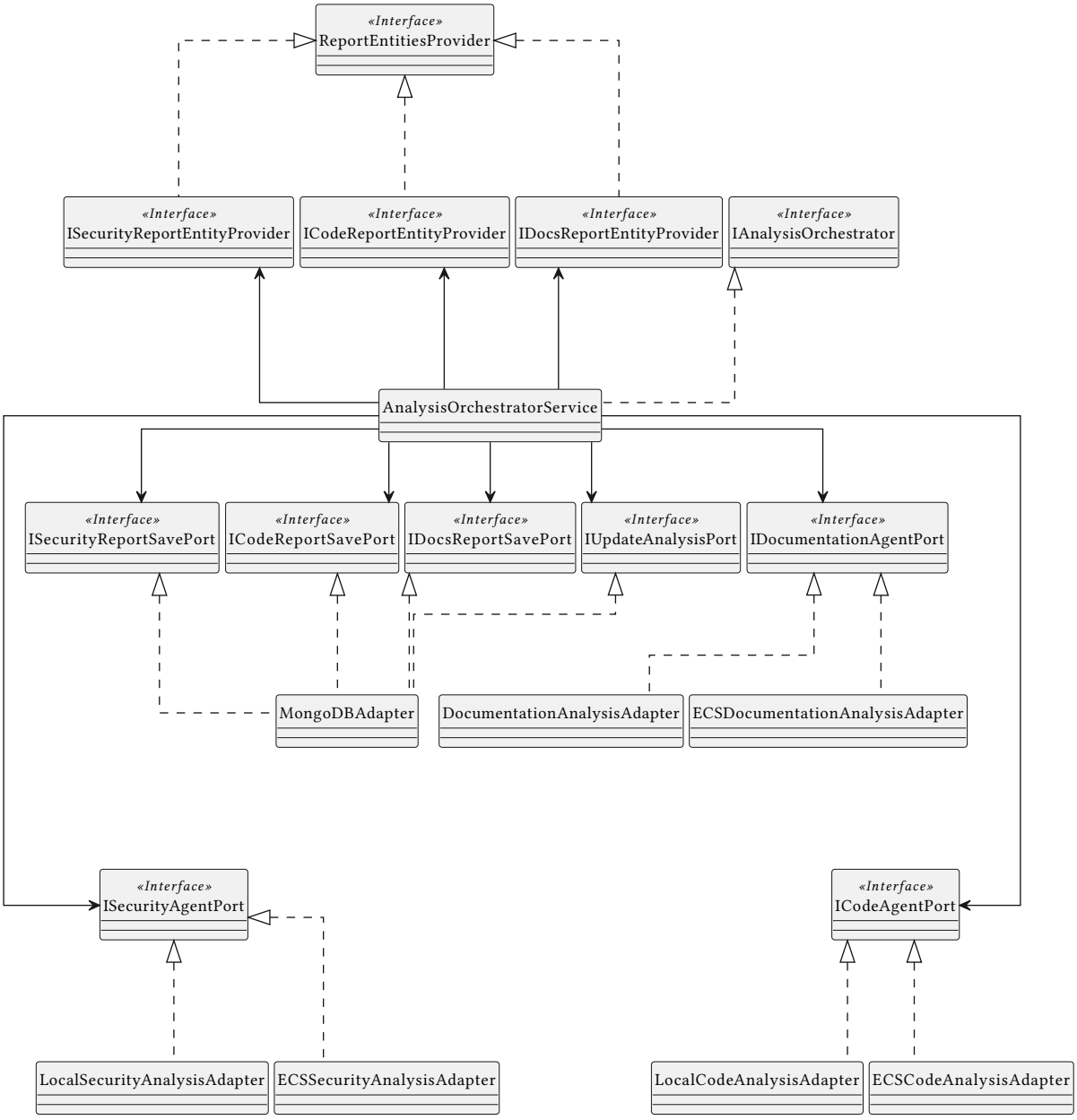

Figure 3: Controller Diagram — OrchestratorReachableClasses

3.2.1.3.2 PAT

Il controller dei PAT (Personal Access Token) gestisce ogni operazione su di essi, il salvataggio in uno nuovo, la modifica e l'eliminazione.

Figure 4: Controller Diagram — PatControllerReachableClasses

3.2.1.3.3 Repositories

Il controller dei repository gestisce ogni operazione sul database delle collections e delle analisi, in particolare permette di:

- Creare una nuova collection
- Richiedere i metadati di tutte le analisi di un utente indipendentemente dalla repo, branch o commit analizzati
- Richiedere tutte le collection di un dato user
- Richiedere i metadati di una collection a partire dall'url della repo
- Richiedere il dettaglio di una analisi a partire dal suo ID
- Cancellare una collezione
- Richiedere i dettagli di tutte le analisi di una collection

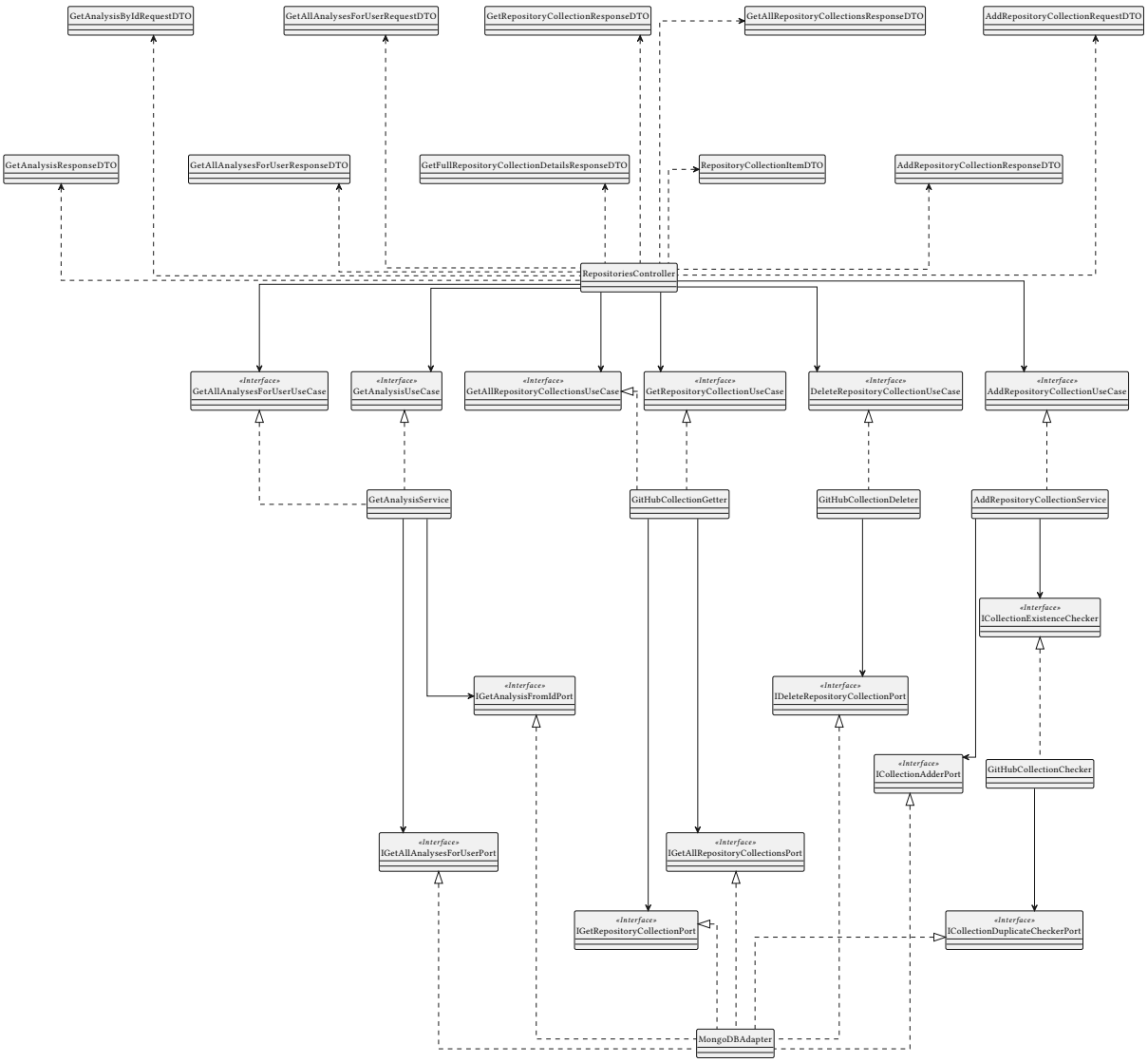

Figure 5: Controller Diagram — RepositoryControllerReachableClasses

3.2.1.4 Domain

Il Dominio rappresenta il nucleo centrale dell'architettura esagonale, dove risiedono esclusivamente la logica di business e le regole vitali del progetto. Questa sezione è progettata per essere totalmente agnostica rispetto alla tecnologia: non possiede alcuna conoscenza di database, protocolli di comunicazione (HTTP/REST) o framework esterni.

L'obiettivo del Domain Core è modellare la realtà del problema attraverso un linguaggio comune (*Ubiquitous Language* [Q]), garantendo che ogni operazione sia coerente con le aspettative del business.

- **Isolamento Tecnologico:** Il dominio non importa librerie esterne di infrastruttura. Questo garantisce che la logica rimanga testabile in isolamento e protetta dall'obsolescenza dei framework.
- **Integrità e Validazione:** È responsabilità del dominio impedire la creazione di oggetti inconsistenti. Ogni componente (Value Object o Entity) è un "garante" della propria validità.
- **Espressione delle Regole:** Non è un semplice deposito di dati, ma un insieme di componenti attivi che governano i processi (es. il ciclo di vita di un'analisi).

3.2.1.4.1 Value Object

I Value Object rappresentano concetti del dominio definiti esclusivamente dai loro attributi. Sono progettati per essere **immutabili**: una volta istanziati, il loro stato non può subire variazioni, garantendo la thread-safety e la stabilità dei riferimenti durante l'intero ciclo di vita della richiesta. L'uguaglianza tra due Value Object è determinata dal valore delle proprietà incapsulate e non dall'identità dell'istanza in memoria.

3.2.1.4.1.1 AIInterpretation

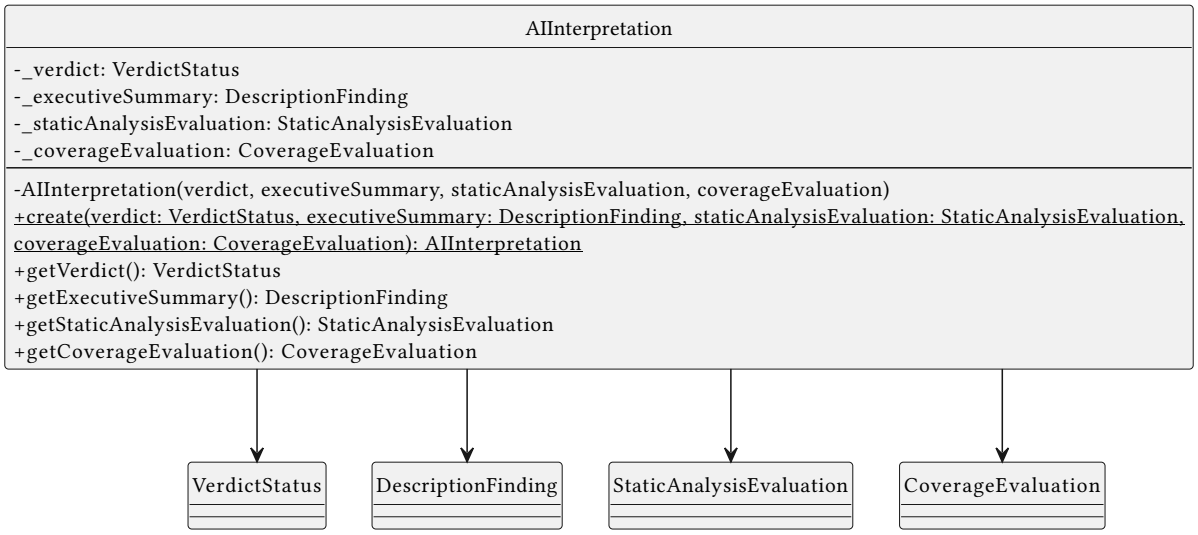

Figure 6: Class Diagram — AIInterpretation

`AIInterpretation` è il Value Object che rappresenta la valutazione complessiva prodotta dall'agente AI a partire dai risultati dell'analisi statica e della copertura. Aggrega il verdetto finale, il sommario esecutivo e le due valutazioni di dettaglio.

- **Verdetto Vincolato:** Il campo `_verdict` è tipizzato sull'enumerazione `VerdictStatus`, garantendo che il giudizio dell'agente sia sempre espresso con un valore riconosciuto dal dominio.
- **Composizione Duale:** Aggrega `StaticAnalysisEvaluation` e `CoverageEvaluation`, offrendo una visione unificata dei due assi di qualità analizzati.

3.2.1.4.1.2 AnalysisId

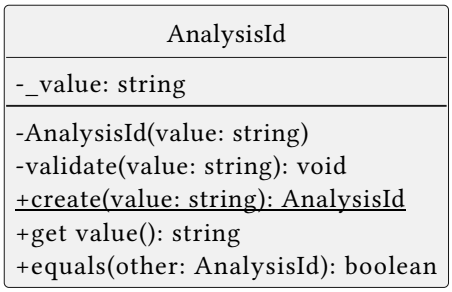

Figure 7: Class Diagram — AnalysisId

L'entità `AnalysisId` è il Value Object che rappresenta l'identità univoca di un'analisi nel sistema. Incapsula un UUID v7 validato, garantendo che ogni analisi possa essere identificata in modo non ambiguo e cronologicamente ordinabile.

- **Immutabilità e Validazione:** Il costruttore privato e il metodo statico `create()` impediscono la creazione di istanze con identificatori malformati, garantendo che solo UUID v7 validi possano essere usati come chiave identitaria.
- **Uguaglianza Strutturale:** Il metodo `equals()` implementa la semantica dei Value Object: due `AnalysisId` sono uguali se e solo se il loro valore stringa è identico.
- **Contratto verso il Dominio:** Viene utilizzato da `GitHubAnalysis` come identificatore primario e da `IRepositoryCloner` per contestualizzare le operazioni di clonazione.

3.2.1.4.1.3 APIViolation

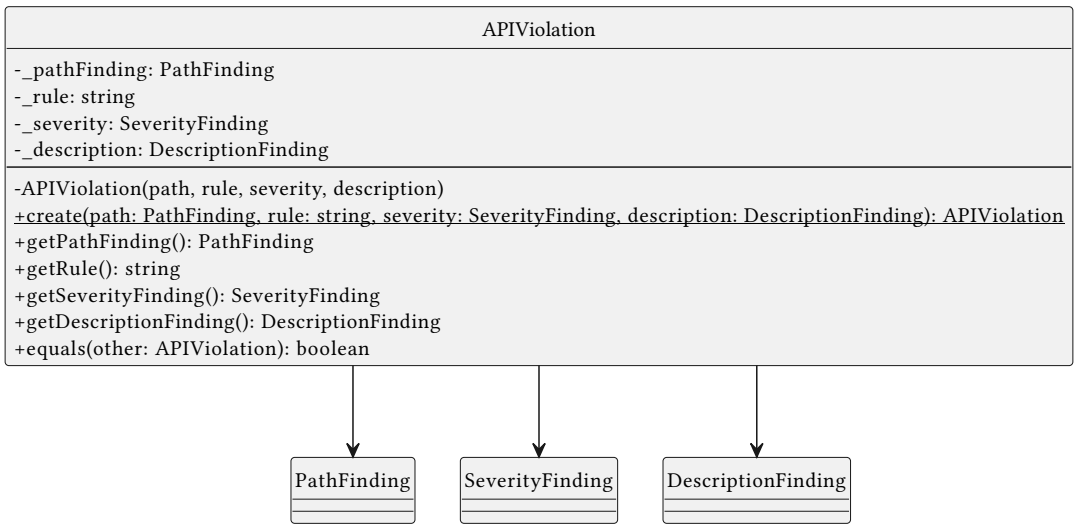

Figure 8: Class Diagram — APIViolation

`APIViolation` è il Value Object che rappresenta una violazione rilevata nel contratto API del repository analizzato. Aggrega la localizzazione del problema, la regola violata, la severità e una descrizione testuale.

- **Regola Esplicita:** Il campo `_rule` identifica la specifica norma del contratto API non rispettata, consentendo classificazioni aggregate per tipologia di violazione.
- **Composizione:** Aggrega `PathFinding`, `SeverityFinding` e `DescriptionFinding`, contestualizzando ogni violazione con la sua localizzazione, criticità e dettaglio testuale.

3.2.1.4.1.4 BranchName

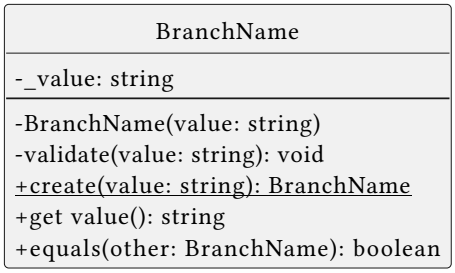

Figure 9: Class Diagram — BranchName

Il Value Object `BranchName` incapsula e valida un nome di branch Git, applicando le regole sintattiche del protocollo Git a livello di dominio. Rappresenta un concetto esplicito del dominio, evitando l'uso di stringhe primitive non validate.

- **Validazione di Dominio:** Tramite una regex derivata dalle specifiche Git (`git-check-ref-format`), impedisce la creazione di branch name che iniziano o terminano con `/`, contengono `..` o caratteri speciali proibiti (`~`, `^`, `:`, `?`, `*`, `[`).
- **Uso come Parametro Contestuale:** Viene aggregato da `GitHubAnalysis` per fissare la versione della sorgente analizzata.

3.2.1.4.1.5 CodeAgentMetadata

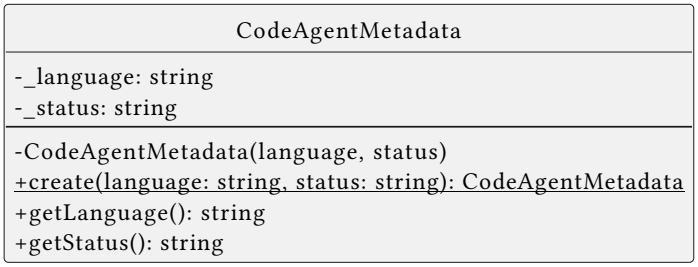

Figure 10: Class Diagram — CodeAgentMetadata

`CodeAgentMetadata` è il Value Object che rappresenta i metadati contestuali prodotti dall'agente di analisi del codice, descrivendo il linguaggio analizzato e lo stato dell'esecuzione.

- **Contesto dell'Esecuzione:** Il campo `_status` garantisce che ogni report sia accompagnato da un'indicazione esplicita sull'esito dell'esecuzione dell'agente, distinguendo analisi completate da quelle parziali o fallite.
- **Linguaggio Dichiarato:** Il campo `_language` associa il report al linguaggio di programmazione analizzato, contestualizzando i risultati per i layer superiori che ne fanno uso.

3.2.1.4.1.6 CommitHash

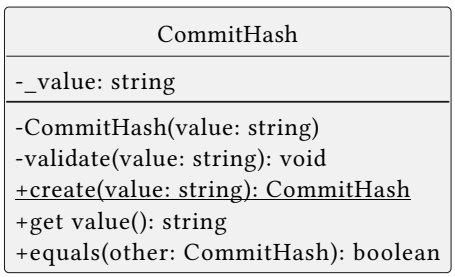

Figure 11: Class Diagram — CommitHash

`CommitHash` è il Value Object che rappresenta un hash SHA-1 di un commit Git (40 caratteri esadecimali). Garantisce che ogni riferimento a un commit nel sistema sia sintatticamente corretto e immutabile.

- **Validazione Strutturale:** La regex `/^[0-9a-fA-F]{40}$/` assicura che solo hash commit validi possano essere istanziati, rendendo impossibile la propagazione di hash corrotti nel dominio.
- **Riproducibilità:** In combinazione con `BranchName` e `RepoURL`, fissa lo stato esatto del repository, rendendo ogni analisi un'operazione deterministica.

3.2.1.4.1.7 ConfigDependency

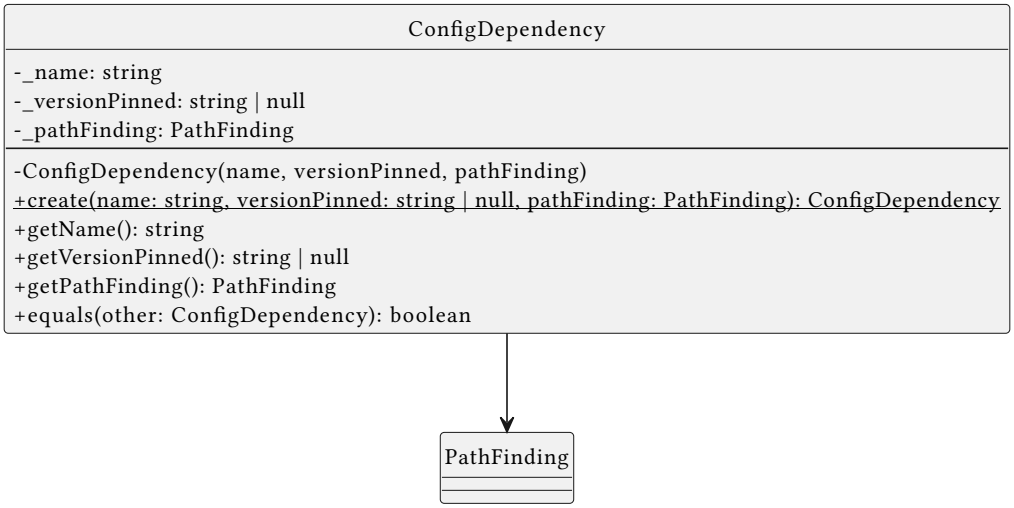

Figure 12: Class Diagram — ConfigDependency

`ConfigDependency` è il Value Object che rappresenta una dipendenza rilevata in un file di configurazione del progetto (es. `package.json`, `requirements.txt`).

- **Localizzazione della Fonte:** Il campo `_pathFinding` di tipo `PathFinding` identifica il file di configurazione specifico in cui la dipendenza è stata rilevata, permettendo di risalire alla fonte senza ambiguità.
- **Versione Pinned Opzionale:** Il campo `_versionPinned` è nullable, gestendo i casi in cui una dipendenza sia dichiarata senza vincolo di versione nel file di configurazione.

3.2.1.4.1.8 CoverageEvaluation

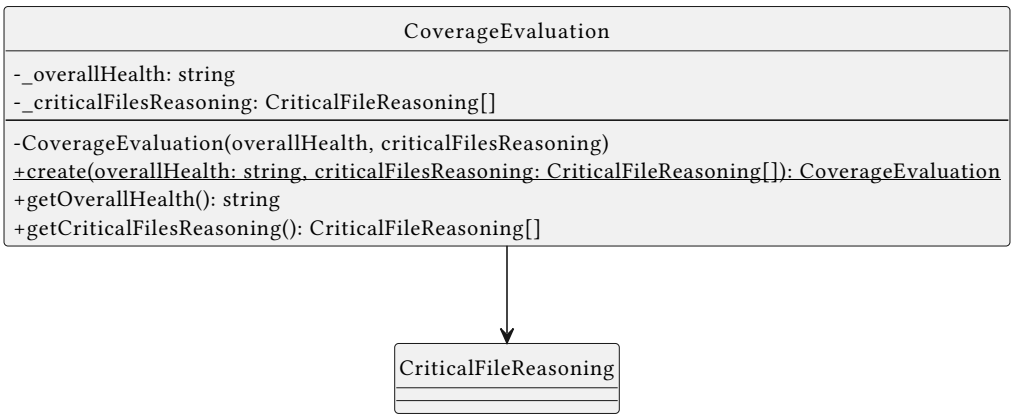

Figure 13: Class Diagram — CoverageEvaluation

`CoverageEvaluation` è il Value Object che aggrega i risultati della valutazione dell'analisi di code coverage, combinando un giudizio sintetico sulla salute complessiva con il dettaglio ragionato per i file critici.

- **Salute Aggregata:** Il campo `_overallHealth` fornisce una valutazione sintetica dell'intera copertura, permettendo ai layer superiori di ottenere un giudizio immediato senza dover ispezionare i singoli file.
- **Dettaglio per File:** La collezione di `CriticalFileReasoning` raccoglie il ragionamento dettagliato per ciascun file critico, fornendo localizzazione delle lacune e spiegazione contestuale in un'unica struttura coesa.

3.2.1.4.1.9 CoveragePercentage

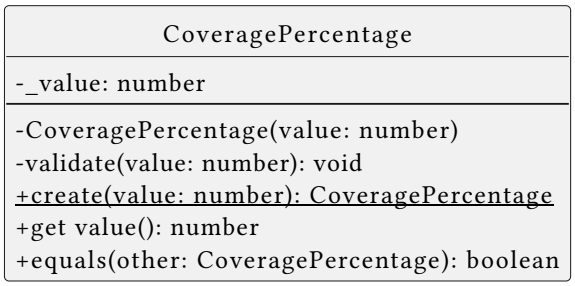

Figure 14: Class Diagram — CoveragePercentage

`CoveragePercentage` è il Value Object che rappresenta una percentuale di copertura del codice come numero decimale nell'intervallo `[0, 1]`.

- **Invariante Numerica:** La validazione garantisce che il valore sia un numero finito compreso tra 0 e 1 inclusivi, prevenendo valori impossibili come percentuali superiori al 100%.
- **Primitivo di Copertura:** Viene riutilizzato come building block da `CriticalFileReasoning` per rappresentare la percentuale di copertura per linee dei file critici.

3.2.1.4.1.10 CriticalFileReasoning

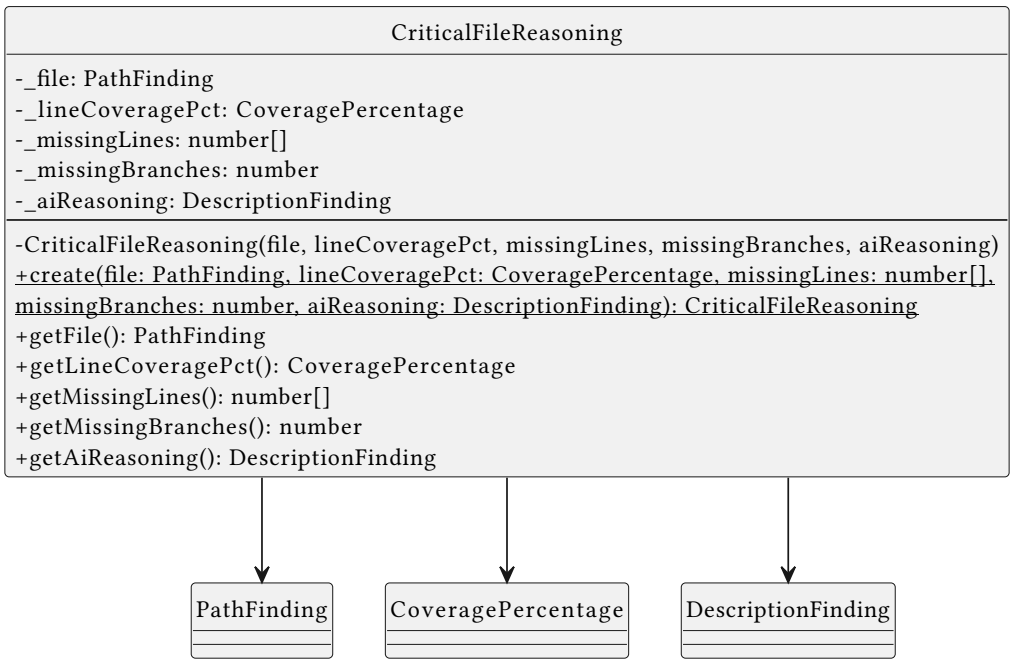

Figure 15: Class Diagram — CriticalFileReasoning

`CriticalFileReasoning` è il Value Object che rappresenta il ragionamento dettagliato su un singolo file critico per la copertura, associando i dati quantitativi delle lacune alla spiegazione contestuale del problema.

- **Dati Quantitativi:** I campi `_missingLines` e `_missingBranches` localizzano con precisione le lacune di copertura, permettendo di intervenire direttamente sulle righe e i branch non coperti.
- **Spiegazione Contestuale:** Il campo `_aiReasoning` di tipo `DescriptionFinding` arricchisce i dati numerici con una descrizione del problema, trasformando il dato grezzo in un'indicazione azionabile.

3.2.1.4.1.11 DependencyAudit

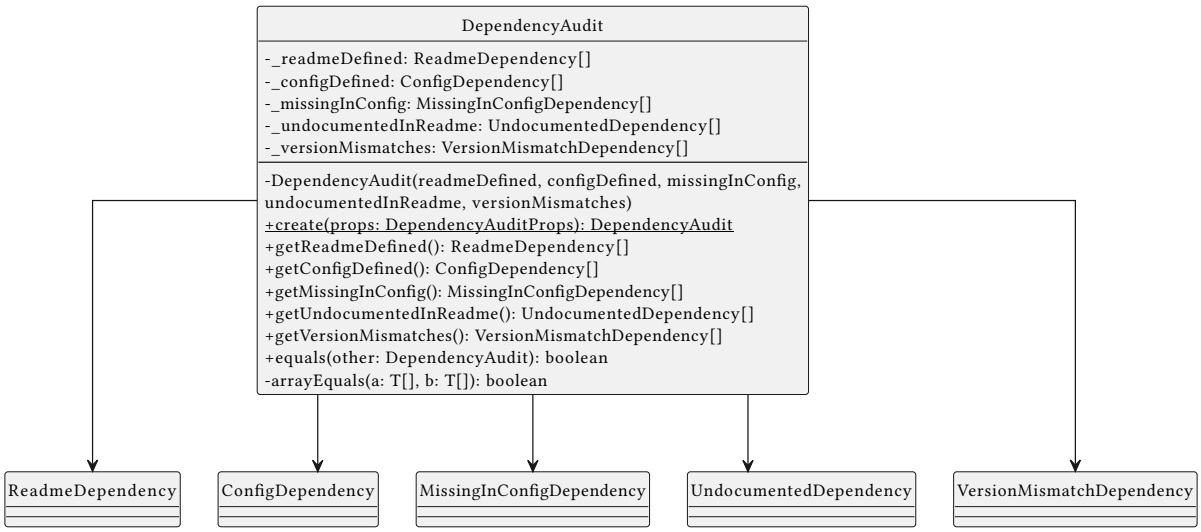

Figure 16: Class Diagram — DependencyAudit

`DependencyAudit` è il Value Object che aggrega l'intero risultato dell'analisi delle dipendenze, confrontando quanto dichiarato nel README con quanto configurato nei file di progetto.

- **Visione Completa:** Raccoglie in un unico oggetto le dipendenze documentate (`ReadmeDependency`), quelle configurate (`ConfigDependency`), quelle mancanti (`MissingInConfigDependency`), quelle non documentate (`UndocumentedDependency`) e i disallineamenti di versione (`VersionMismatchDependency`).
- **Uguaglianza Ordinata:** Il confronto tra collezioni è indipendente dall'ordine di inserimento, garantendo che due audit con le stesse dipendenze siano sempre considerati equivalenti.

3.2.1.4.1.12 DependencyFinding

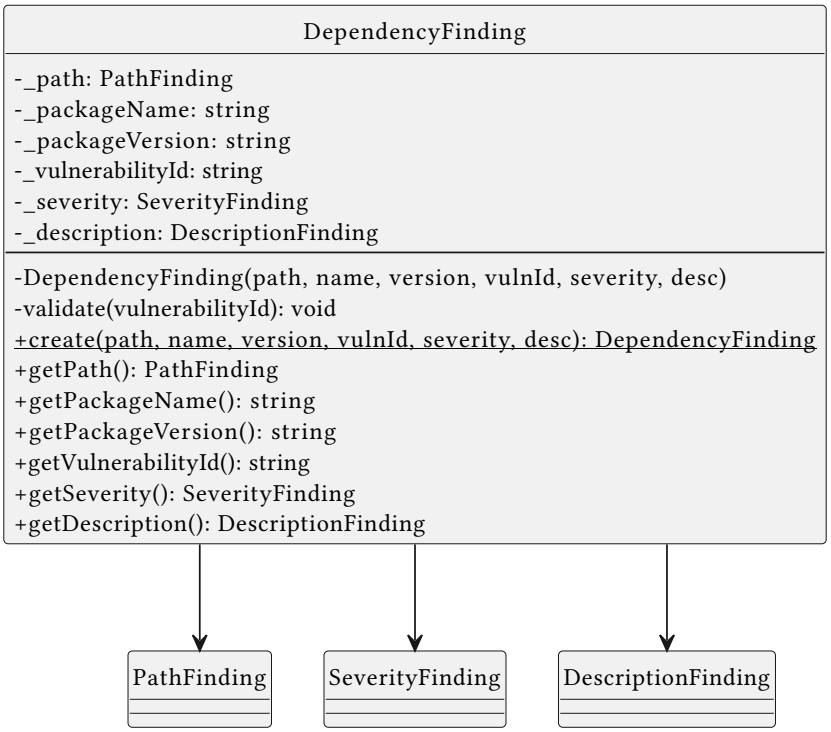

Figure 17: Class Diagram — DependencyFinding

`DependencyFinding` è il Value Object che rappresenta una vulnerabilità nota rilevata in una dipendenza del progetto (es. CVE in un pacchetto npm o pip).

- **Tracciabilità CVE:** Il campo `vulnerabilityId` permette di correlare il finding con i database pubblici di vulnerabilità (es. CVE-2021-44228).
- **Contesto Completo:** Aggrega nome e versione del pacchetto, la severità (`SeverityFinding`) e una descrizione (`DescriptionFinding`).

3.2.1.4.1.13 DescriptionFinding

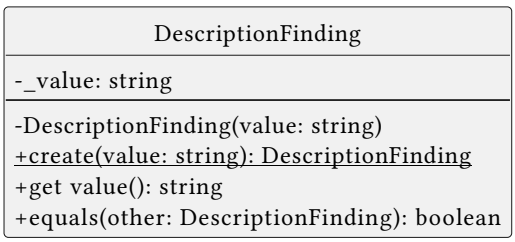

Figure 18: Class Diagram — DescriptionFinding

`DescriptionFinding` è il Value Object primitivo che rappresenta la descrizione testuale di un finding di analisi. Garantisce che la descrizione non sia mai vuota o composta da soli spazi.

- **Riuso Composizionale:** Viene riutilizzato come componente da Value Object più complessi quali `ErrorFinding` e `DependencyFinding`.

3.2.1.4.1.14 DocsDiscrepancy

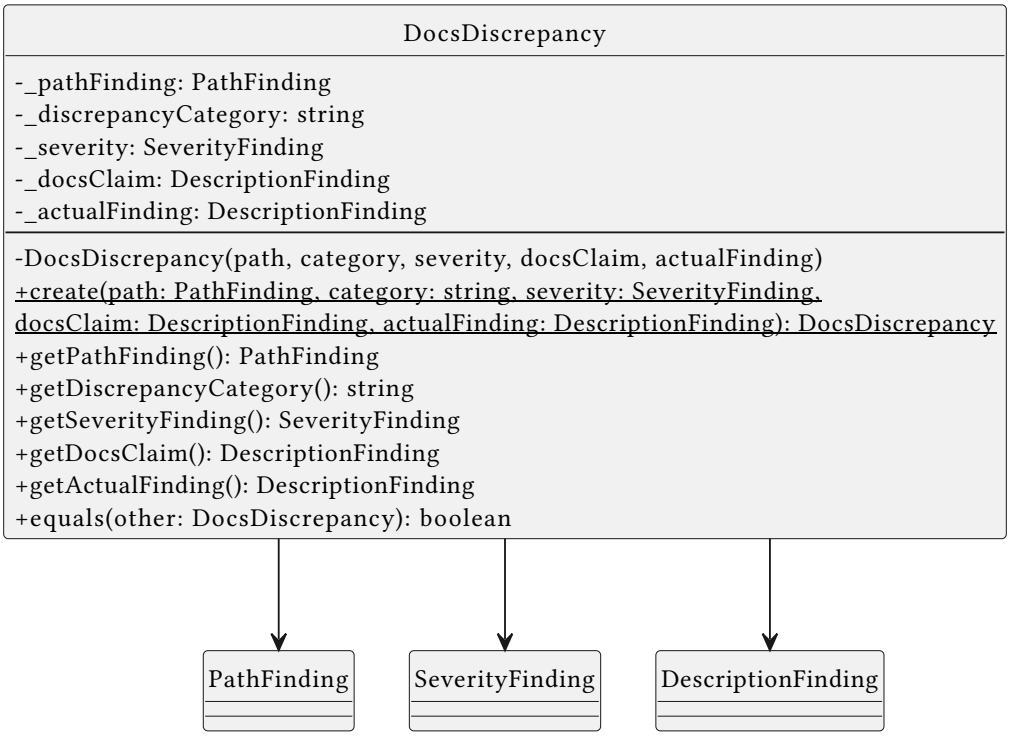

Figure 19: Class Diagram — DocsDiscrepancy

`DocsDiscrepancy` è il Value Object che rappresenta una discrepanza tra quanto dichiarato nella documentazione e quanto effettivamente rilevato nel codice sorgente.

- **Confronto Duale:** I campi `_docsClaim` e `_actualFinding` catturano esplicitamente entrambi i lati della divergenza, rendendo il problema autoesplicativo senza necessità di ricorrere al codice sorgente.
- **Categorizzazione:** Il campo `_discrepancyCategory` raggruppa le discrepanze per tipologia, abilitando analisi aggregate e prioritizzazione degli interventi correttivi.

3.2.1.4.1.15 ErrorFinding

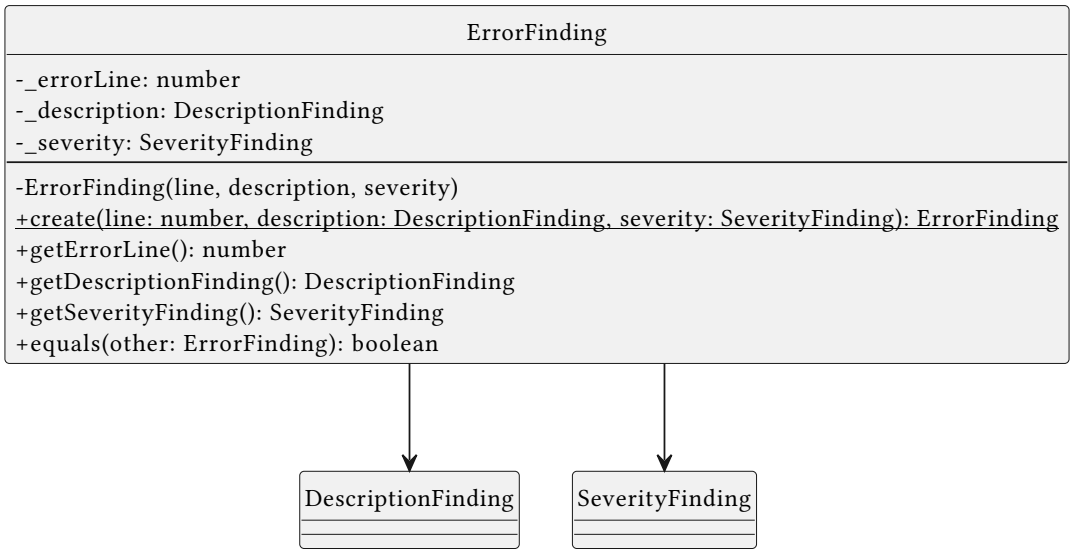

Figure 20: Class Diagram — ErrorFinding

`ErrorFinding` è il Value Object composito che rappresenta un singolo errore rilevato durante l'analisi, aggregando la riga di codice incriminata, una descrizione e un livello di severità.

- **Composizione di Primitivi:** Aggrega `DescriptionFinding` e `SeverityFinding`, arricchendoli con il numero di riga (intero positivo).
- **Blocco Costruttivo:** Viene utilizzato come componente da `OWASPFinding` e `SecretFinding`.

3.2.1.4.1.16 IssueLocation

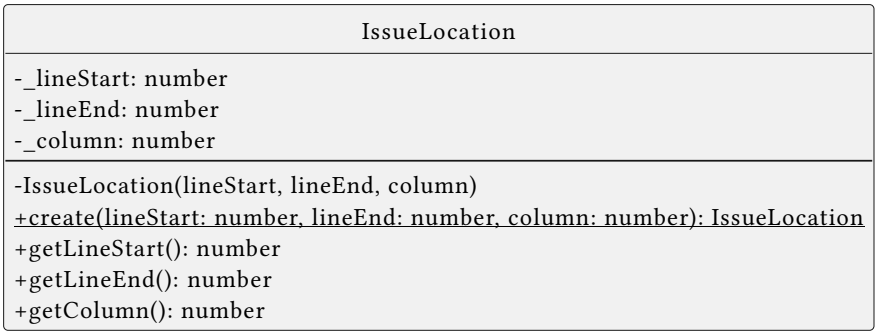

Figure 21: Class Diagram — IssueLocation

`IssueLocation` è il Value Object che rappresenta la posizione precisa di un problema all'interno di un file sorgente, identificando l'intervallo di righe e la colonna coinvolti.

- **Posizione Intervallare:** I campi `_lineStart` e `_lineEnd` modellano problemi che si estendono su più righe, garantendo tramite validazione che `lineStart` non superi mai `lineEnd` e che entrambi siano positivi.
- **Localizzazione Completa:** Combinato con `PathFinding` in `KeyIssueReasoning`, consente di identificare un problema con la precisione necessaria per navigarvi direttamente in un editor.

3.2.1.4.1.17 KeyIssueReasoning

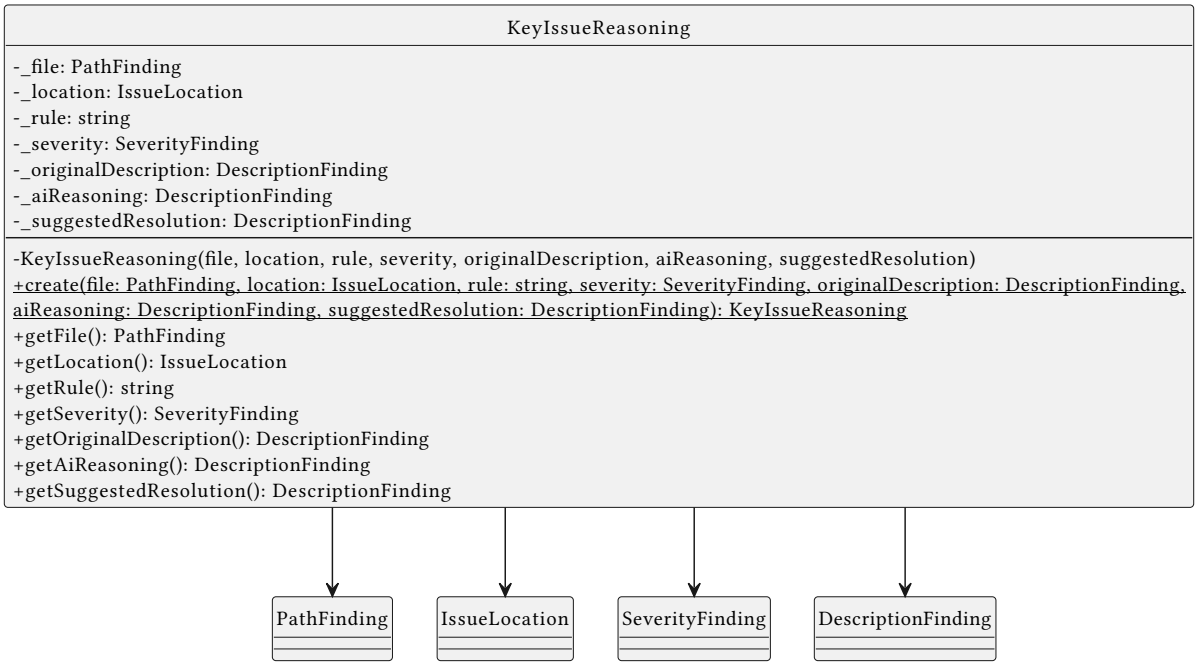

Figure 22: Class Diagram — KeyIssueReasoning

`KeyIssueReasoning` è il Value Object che rappresenta il ragionamento dettagliato su un singolo problema rilevato dall'analisi statica, arricchendo il dato grezzo con interpretazione e risoluzione suggerita.

- **Triplice Descrizione:** I campi `_originalDescription`, `_aiReasoning` e `_suggestedResolution`, tutti di tipo `DescriptionFinding`, separano esplicitamente il problema rilevato dallo strumento, la sua interpretazione e la strategia di risoluzione proposta.
- **Localizzazione Precisa:** Aggrega `PathFinding` e `IssueLocation`, permettendo di navigare direttamente al punto esatto del codice senza ambiguità.

3.2.1.4.1.18 MissingFile

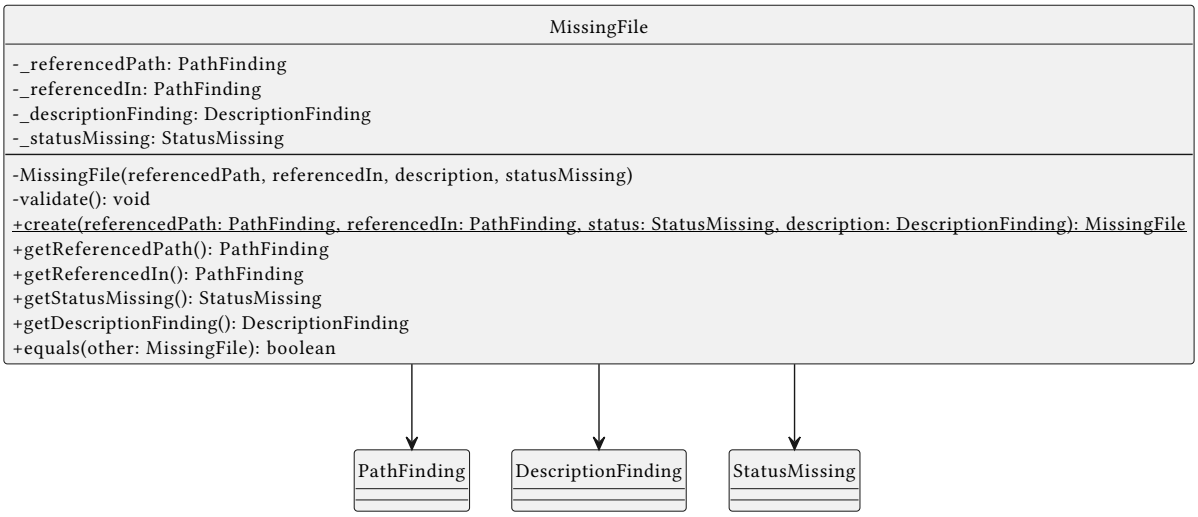

Figure 23: Class Diagram — MissingFile

`MissingFile` è il Value Object che rappresenta un file referenziato nella documentazione ma assente nel repository analizzato.

- **Doppia Localizzazione:** I campi `_referencedPath` e `_referencedIn` permettono di risalire non solo al file mancante, ma anche al documento che ne dichiara l'esistenza, rendendo il finding immediatamente azionabile.
- **Stato Classificato:** Il campo `_statusMissing` di tipo `StatusMissing` distingue semanticamente le diverse cause di assenza, abilitando politiche di gestione differenziate.

3.2.1.4.1.19 MissingInConfigDependency

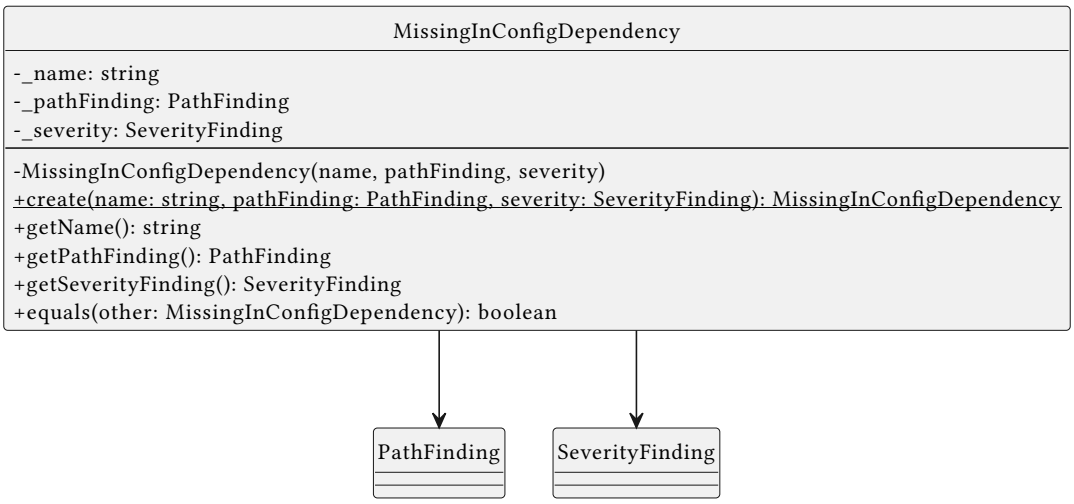

Figure 24: Class Diagram — MissingInConfigDependency

`MissingInConfigDependency` è il Value Object che rappresenta una dipendenza documentata nel README ma assente nei file di configurazione del progetto.

- **Severità Associata:** Il campo `_severity` di tipo `SeverityFinding` permette di graduare il rischio della mancanza, distinguendo dipendenze critiche da quelle accessorie.
- **Riferimento al Contesto:** Il campo `_pathFinding` di tipo `PathFinding` indica il file di configurazione in cui la dipendenza avrebbe dovuto essere presente, contestualizzando il problema per chi deve risolverlo.

3.2.1.4.1.20 OWASPFinding

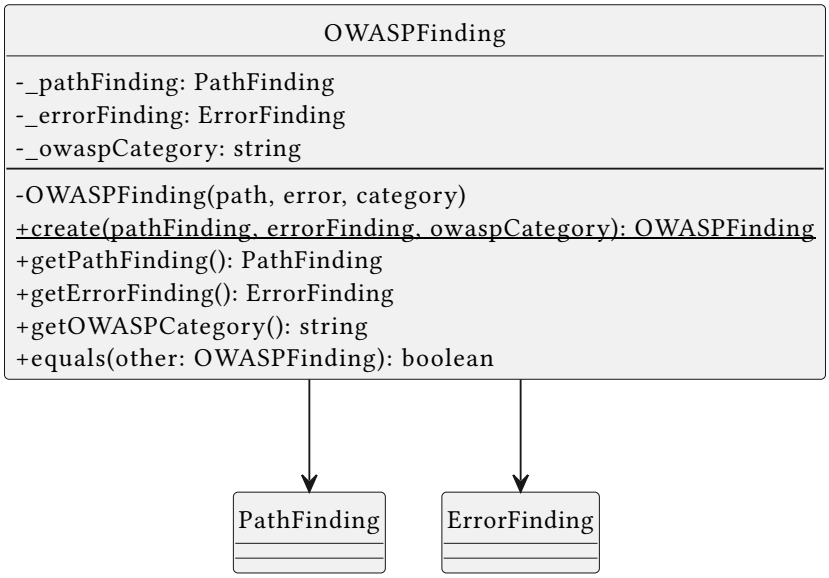

Figure 25: Class Diagram — OWASPFinding

`OWASPFinding` è il Value Object che rappresenta una vulnerabilità di sicurezza classificata secondo la tassonomia OWASP. Aggrega localizzazione, dettaglio dell'errore e categoria OWASP.

- **Classificazione Standard:** Il campo `owaspCategory` permette di associare ogni vulnerabilità a una categoria riconosciuta (es. `A01:2021-Broken Access Control`).

3.2.1.4.1.21 PATPassword

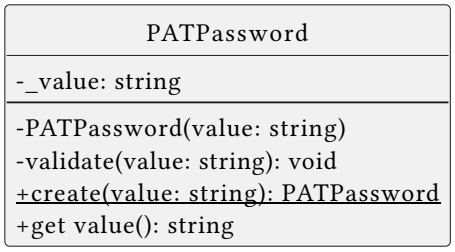

Figure 26: Class Diagram — PATPassword

`PATPassword` è il Value Object che rappresenta l'hash SHA-256 di una password usata per proteggere un Personal Access Token. Non contiene mai la password in chiaro.

- **Sicurezza per Design:** Accetta esclusivamente stringhe nel formato SHA-256 (64 caratteri esadecimali), impedendo la circolazione di password in chiaro nel dominio.

3.2.1.4.1.22 PathFinding

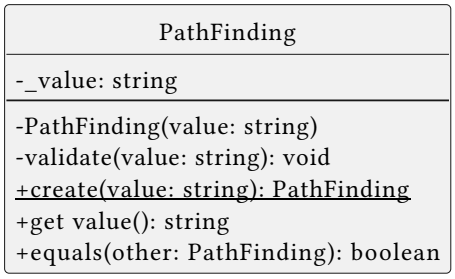

Figure 27: Class Diagram — PathFinding

`PathFinding` è il Value Object che rappresenta un percorso relativo a un file nel repository analizzato. Applica regole di sicurezza per impedire path traversal e percorsi assoluti.

- **Sicurezza Strutturale:** Vieta percorsi assoluti, separatori Windows (\) e sequenze `..`, garantendo che i finding riferiscano sempre a file interni al repository clonato.

3.2.1.4.1.23 PersonalAccessToken

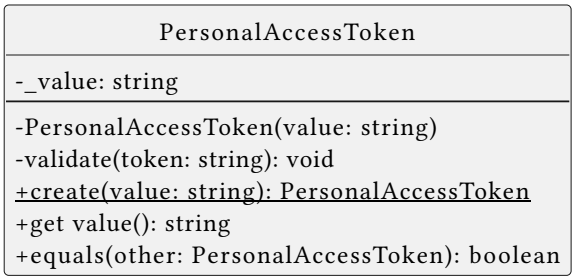

Figure 28: Class Diagram — PersonalAccessToken

`PersonalAccessToken` è il Value Object che incapsula un GitHub PAT, validandone il formato secondo i pattern ufficiali GitHub (`ghp_*` o `github_pat_*`).

- **Validazione del Formato:** La regex garantisce che solo token con formato ufficiale possano essere usati, prevenendo la registrazione di token malformati.

3.2.1.4.1.24 ReadmeDependency

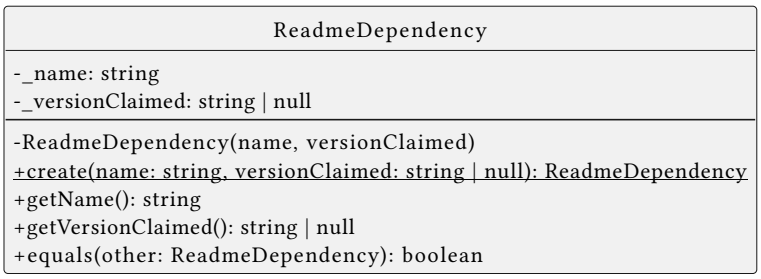

Figure 29: Class Diagram — ReadmeDependency

`ReadmeDependency` è il Value Object che rappresenta una dipendenza dichiarata nel file README del repository.

- **Versione Opzionale:** Il campo `_versionClaimed` è nullable, riflettendo la realtà documentale in cui un README può citare una dipendenza senza specificarne la versione esatta.
- **Fonte Documentale:** Viene aggregato da `DependencyAudit` come rappresentazione della dipendenza dal punto di vista della documentazione, da confrontare con quanto dichiarato nei file di configurazione.

3.2.1.4.1.25 ReportId

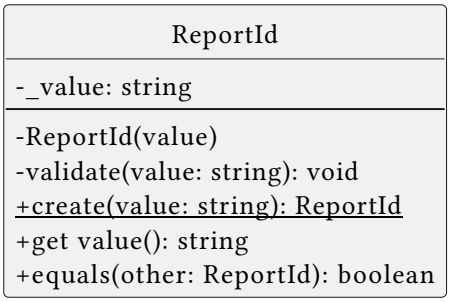

Figure 30: Class Diagram — ReportId

`ReportId` è il Value Object che rappresenta l'identità univoca di un report nel sistema. Incapsula un UUID v7 validato, garantendo che ogni report possa essere identificato in modo non ambiguo e cronologicamente ordinabile.

- **Uguaglianza Strutturale:** Il metodo `equals()` implementa la semantica dei Value Object: due `ReportId` sono uguali se e solo se il loro valore stringa è identico.

3.2.1.4.1.26 RepoURL

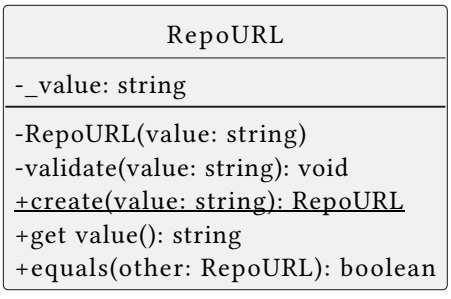

Figure 31: Class Diagram — RepoURL

`RepoURL` è il Value Object che rappresenta l'URL di un repository GitHub. La validazione garantisce la conformità al formato `https://github.com/<owner>/<repo>`.

- **Contesto Esplicito:** Vincola il sistema a operare esclusivamente su repository GitHub, rendendo esplicito il bounded context del modulo.
- **Sicurezza:** Il requisito HTTPS previene l'iniezione di URL arbitrari.

3.2.1.4.1.27 SecretFinding

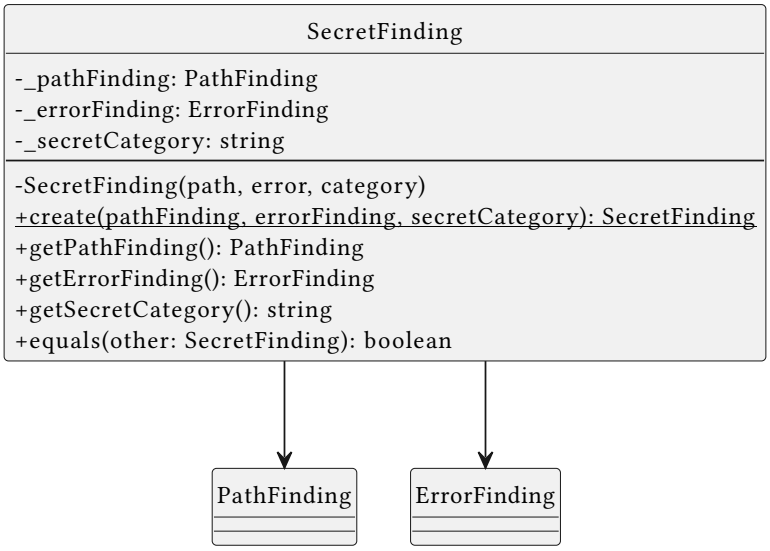

Figure 32: Class Diagram — SecretFinding

`SecretFinding` rappresenta la rilevazione di un segreto esposto (API key, token, password) all'interno del codice sorgente analizzato.

- **Categorizzazione:** Il campo `secretCategory` (es. `AWS_ACCESS_KEY`) permette al sistema di classificare il tipo di segreto trovato.
- **Criticità:** Indica un rischio immediato di compromissione delle risorse accedute dal segreto rilevato.

3.2.1.4.1.28 SeverityFinding

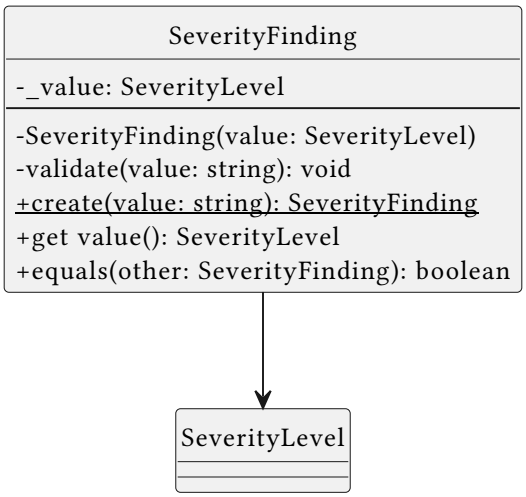

Figure 33: Class Diagram — SeverityFinding

`SeverityFinding` è il Value Object che rappresenta il livello di severità di un finding, vincolato all'enumerazione `SeverityLevel` (`LOW`, `MEDIUM`, `HIGH`, `CRITICAL`).

- **Coerenza del Dominio:** Normalizza la stringa in input e la valida, garantendo che nessun livello arbitrario possa essere introdotto.

3.2.1.4.1.29 StaticAnalysisEvaluation

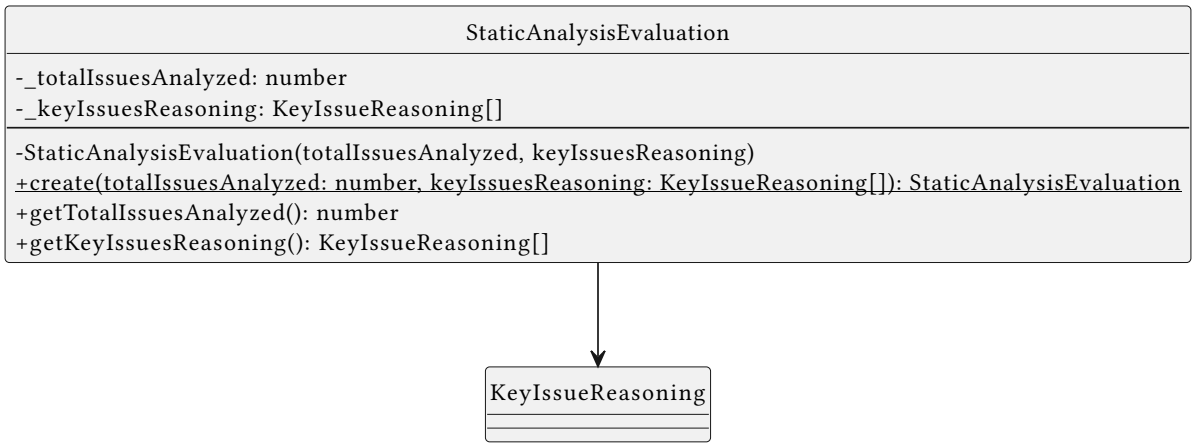

Figure 34: Class Diagram — StaticAnalysisEvaluation

`StaticAnalysisEvaluation` è il Value Object che aggrega i risultati della valutazione dell'analisi statica, combinando il conteggio totale dei problemi rilevati con la collezione dei ragionamenti dettagliati per ciascun problema.

- **Conteggio Totale:** Il campo `_totalIssuesAnalyzed` preserva il numero complessivo di problemi analizzati, permettendo di avere una visione quantitativa immediata dell'entità dei problemi rilevati.
- **Ragionamenti Aggregati:** La collezione di `KeyIssueReasoning` raccoglie il dettaglio analitico per ciascun problema, fornendo localizzazione, severità e risoluzione suggerita in un'unica struttura coesa.

3.2.1.4.1.30 ToolError

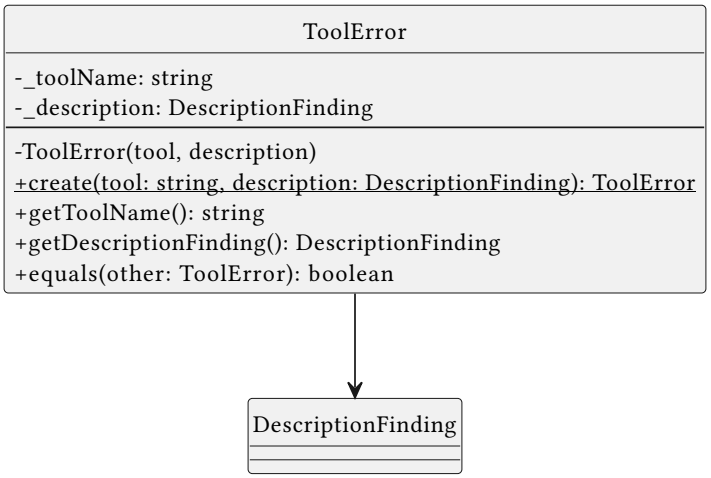

Figure 35: Class Diagram — ToolError

`ToolError` è il Value Object che rappresenta un errore verificatosi durante l'esecuzione di uno strumento di analisi, associando il nome dello strumento alla descrizione del problema riscontrato.

- **Identificazione della Fonte:** Il campo `_toolName` permette di risalire immediatamente allo strumento che ha generato l'errore.

- **Uguaglianza per Contenuto:** Il metodo `equals()` confronta sia il nome dello strumento che la descrizione, garantendo che due errori identici prodotti dallo stesso tool siano riconosciuti come equivalenti.

3.2.1.4.1.31 UndocumentedDependency

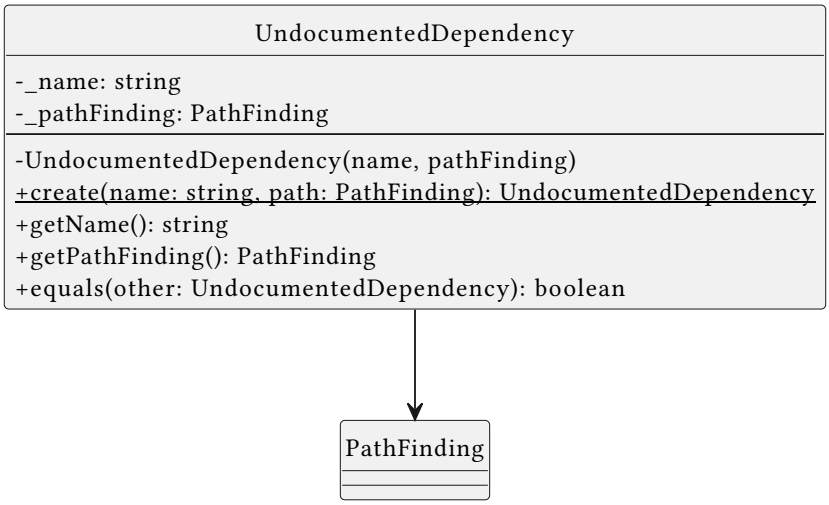

Figure 36: Class Diagram — UndocumentedDependency

`UndocumentedDependency` è il Value Object che rappresenta una dipendenza presente nei file di configurazione del progetto ma non menzionata nel README.

- **Gap di Documentazione:** La sua presenza segnala una libreria introdotta senza aggiornamento del README, riducendo la comprensibilità del progetto per i nuovi contributori.
- **Localizzazione Precisa:** Il campo `_pathFinding` di tipo `PathFinding` indica il file di configurazione in cui la dipendenza è stata rilevata, facilitando l'intervento correttivo sulla documentazione.

3.2.1.4.1.32 UserId

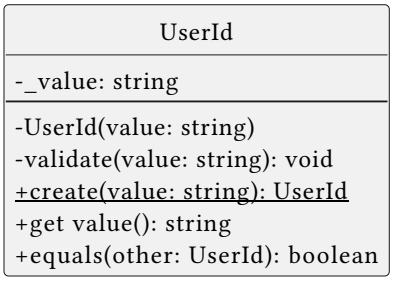

Figure 37: Class Diagram — UserId

`UserId` è il Value Object che rappresenta l'identità dell'utente che ha richiesto l'analisi. Incapsula un UUID standard validato.

- **Tracciabilità:** Viene aggregato da `GitHubAnalysis` per associare ogni analisi al richiedente, abilitando audit trail e politiche di accesso.
- **Separazione:** Permette di evolvere il modello identitario senza impattare la logica di analisi del dominio.

3.2.1.4.1.33 VersionMismatchDependency

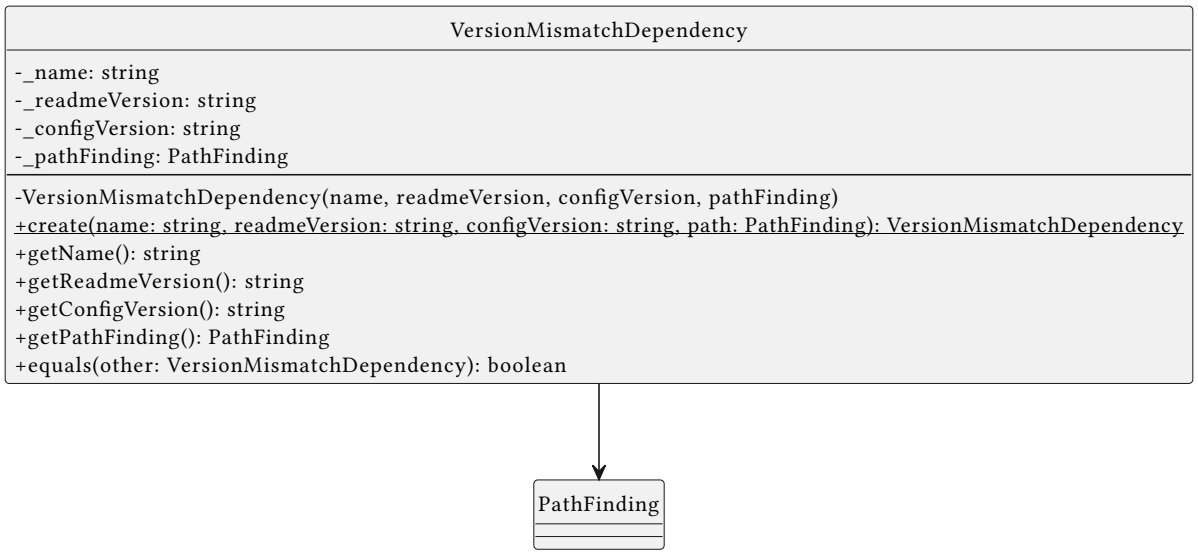

Figure 38: Class Diagram — VersionMismatchDependency

`VersionMismatchDependency` è il Value Object che rappresenta una dipendenza la cui versione dichiarata nel README differisce da quella specificata nel file di configurazione.

- **Confronto Esplicito:** I campi `_readmeVersion` e `_configVersion` preservano entrambe le versioni rilevate, permettendo al report di mostrare la discrepanza in modo diretto senza perdita di informazione.
- **Tracciabilità della Fonte:** Il campo `_pathFinding` di tipo `PathFinding` indica il file di configurazione in cui la versione discordante è stata rilevata, guidando l'intervento correttivo verso il file specifico da aggiornare.

3.2.1.4.2 Enums

3.2.1.4.2.1 AnalysisStatus

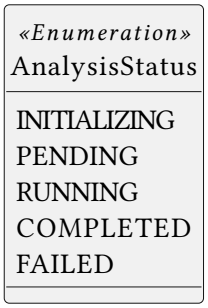

Figure 39: Class Diagram — AnalysisStatus

`AnalysisStatus` è l'enumerazione che definisce gli stati del ciclo di vita di un'analisi: `PENDING`, `IN_PROGRESS`, `COMPLETED`, `FAILED`.

- **Macchina a Stati:** Definisce le transizioni valide gestite da `GitHubAnalysis`: `PENDING → IN_PROGRESS` (via `inProgress()`), `IN_PROGRESS → COMPLETED` (via `complete()`), `IN_PROGRESS → FAILED` (via `failed()`). Ogni transizione non valida genera un errore esplicito.
- **Osservabilità:** Permette ai servizi applicativi e all'infrastruttura di monitorare e persistere lo stato di avanzamento dell'analisi in modo type-safe.

3.2.1.4.2.2 SeverityLevel

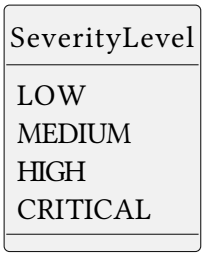

Figure 40: Class Diagram — SeverityLevel

`SeverityLevel` è l'enumerazione che definisce i livelli di criticità dei finding: `LOW`, `MEDIUM`, `HIGH`, `CRITICAL`.

- **Vocabolario Condiviso:** Definisce il vocabolario ufficiale del dominio per la classificazione della severità, usato da `SeverityFinding` come insieme di valori validi.
- **Standardizzazione:** Evita l'uso di stringhe libere, garantendo che tutti i componenti del sistema parlino lo stesso linguaggio per la prioritizzazione dei problemi rilevati.

3.2.1.4.2.3 StatusMissing

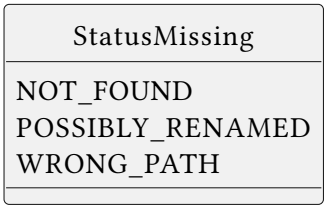

Figure 41: Class Diagram — StatusMissing

`StatusMissing` è l'enumerazione che classifica la causa di assenza di un file referenziato nella documentazione: `NOT_FOUND`, `POSSIBLY_RENAMED`, `WRONG_PATH`.

- **Vocabolario Diagnostico:** Definisce il vocabolario ufficiale del dominio per distinguere le diverse cause di assenza, usato da `MissingFile` per caratterizzare semanticamente ogni file mancante.
- **Azionabilità:** I tre valori guidano interventi distinti: `NOT_FOUND` suggerisce un file mai creato, `POSSIBLY_RENAMED` un refactoring non riflesso nella documentazione, `WRONG_PATH` un errore di percorso nel riferimento.

3.2.1.4.2.4 VerdictStatus

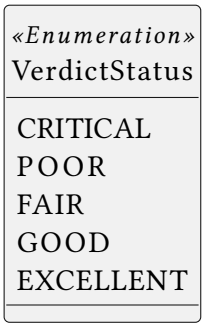

Figure 42: Class Diagram — VerdictStatus

`VerdictStatus` è l'enumerazione che definisce i livelli di qualità complessiva restituiti dall'analisi del codice: `CRITICAL`, `POOR`, `FAIR`, `GOOD`, `EXCELLENT`.

- **Scala Ordinata:** I cinque valori formano una scala progressiva dalla qualità peggiore alla migliore, usata da `AIInterpretation` per esprimere il giudizio sintetico sull'intero repository analizzato.
- **Standardizzazione:** Traduce il verdetto testuale presente nel JSON dell'agente in un valore type-safe del dominio, impedendo la propagazione di verdetti arbitrari nei layer applicativi.

3.2.1.4.3 Entity

A differenza dei Value Object, le Entity sono definite dalla loro **identità** persistente nel tempo e non solo dai loro attributi. Un'Entity mantiene la propria individualità anche se i suoi dati interni subiscono variazioni. Esse incapsulano lo stato e il comportamento del business, garantendo che le transizioni di stato avvengano nel rispetto delle regole del dominio.

- **Identità Univoca:** Ogni Entity è associata a un identificatore immutabile che ne permette la distinzione univoca all'interno del sistema.
- **Ciclo di Vita e Stato:** Le Entity possiedono un ciclo di vita (creazione, modifica, archiviazione) e gestiscono attivamente le proprie mutazioni interne attraverso metodi espliciti.
- **Integrità Comportamentale:** Non si limitano a esporre dati (getter/setter), ma offrono metodi che rappresentano azioni di business, assicurando che l'oggetto passi solo attraverso stati validi e coerenti.

3.2.1.4.3.1 GitHubAnalysis

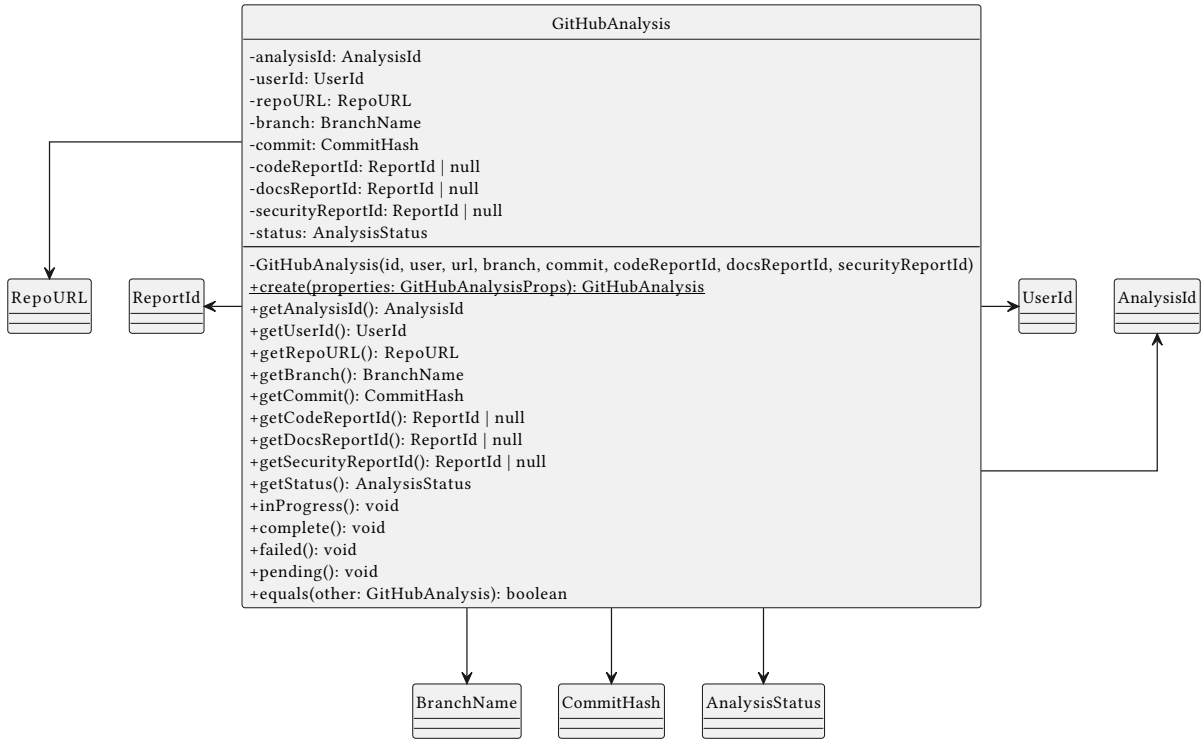

Figure 43: Class Diagram — GitHubAnalysis

`GitHubAnalysis` è l'entità radice dell'aggregato che rappresenta un'analisi di un repository GitHub. Incapsula l'identità, l'utente richiedente, il contesto del repository, i riferimenti ai report prodotti dalle analisi e la macchina a stati del ciclo di vita.

- **Identità Basata sull'Analisi:** Il metodo `equals()` confronta due istanze esclusivamente tramite `AnalysisId`, esprimendo la semantica fondamentale delle entity: due `GitHubAnalysis` sono la stessa analisi se condividono l'identità, indipendentemente dai dati aggregati.

- **Radice dell'Aggregato:** Coordina e protegge la consistenza dei Value Object `AnalysisId`, `UserId`, `RepoURL`, `BranchName`, `CommitHash` e `ReportId`.
- **Riferimenti ai Report Opzionali:** I campi `codeReportId`, `docsReportId` e `securityReportId` sono nullable, modellando il fatto che un'analisi può coinvolgere solo un sottoinsieme dei tre tipi di report disponibili in base a quanto richiesto.
- **Macchina a Stati Protetta:** I metodi `inProgress()`, `complete()` e `failed()` implementano le transizioni valide dell'`AnalysisStatus`, rifiutando transizioni non consentite con errori espliciti.

3.2.1.4.3.2 CodeAgentReport

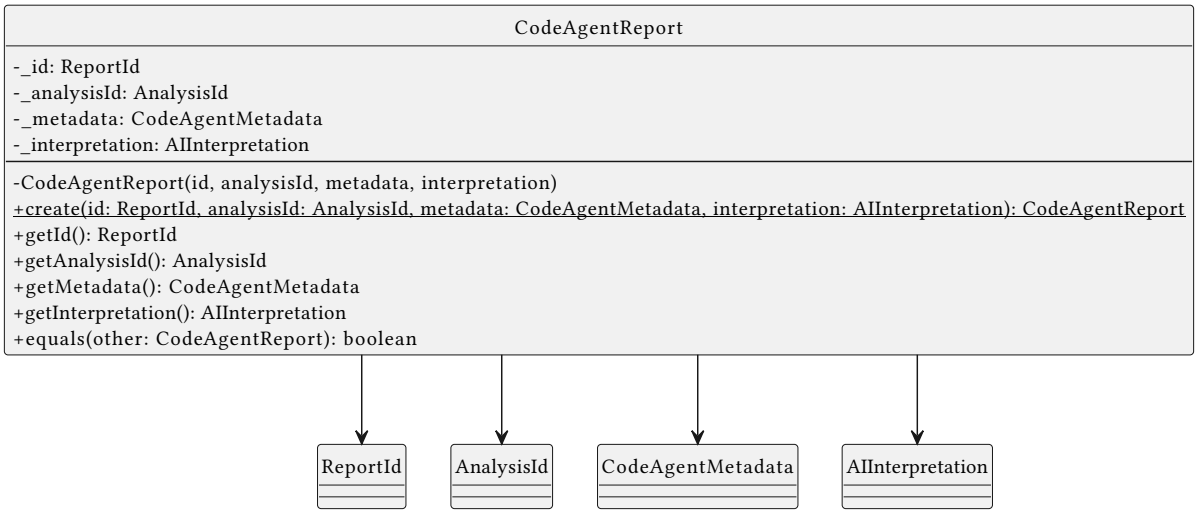

Figure 44: Class Diagram — CodeAgentReport

`CodeAgentReport` è l'entità che rappresenta il report prodotto dall'analisi del codice per una specifica analisi. Aggrega i metadati dell'esecuzione e l'interpretazione complessiva dei risultati, collegandosi all'analisi di appartenenza tramite identificatori.

- **Identità Basata sul Report:** Il metodo `equals()` confronta due istanze esclusivamente tramite `ReportId`, esprimendo la semantica fondamentale delle entity: due `CodeAgentReport` sono lo stesso report se condividono l'identità, indipendentemente dai dati aggregati.
- **Contenuto del Report:** Aggrega `CodeAgentMetadata` per i metadati contestuali dell'esecuzione e `AIInterpretation` per il verdetto complessivo e le valutazioni di dettaglio sull'analisi statica e sulla copertura.

3.2.1.4.3.3 DocumentationReport

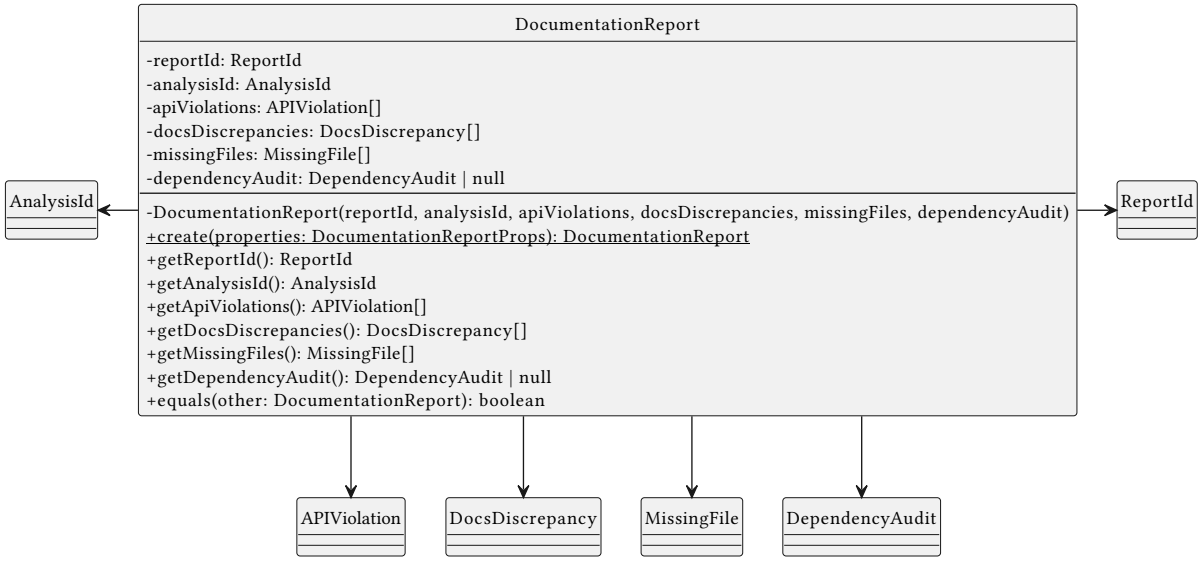

Figure 45: Class Diagram — DocumentationReport

`DocumentationReport` è l'entità che rappresenta il report prodotto dall'analisi della documentazione per una specifica analisi. Aggrega le violazioni API, le discrepanze documentali, i file mancanti e l'audit delle dipendenze, collegandosi all'analisi di appartenenza tramite identificatori.

- **Identità Basata sul Report:** Il metodo `equals()` confronta due istanze esclusivamente tramite `ReportId`, esprimendo la semantica fondamentale delle entity: due `DocumentationReport` sono lo stesso report se condividono l'identità, indipendentemente dai dati aggregati.
- **Risultati Opzionali:** Le collezioni `apiViolations` (`APIViolation`), `docsDiscrepancies` (`DocsDiscrepancy`) e `missingFiles` (`MissingFile`) vengono inizializzate a array vuoto se non fornite, mentre `dependencyAudit` (`DependencyAudit`) è nullable, modellando il fatto che ciascuna categoria di risultati può essere assente in base all'esito dell'analisi.

3.2.1.4.3.4 SecurityReport

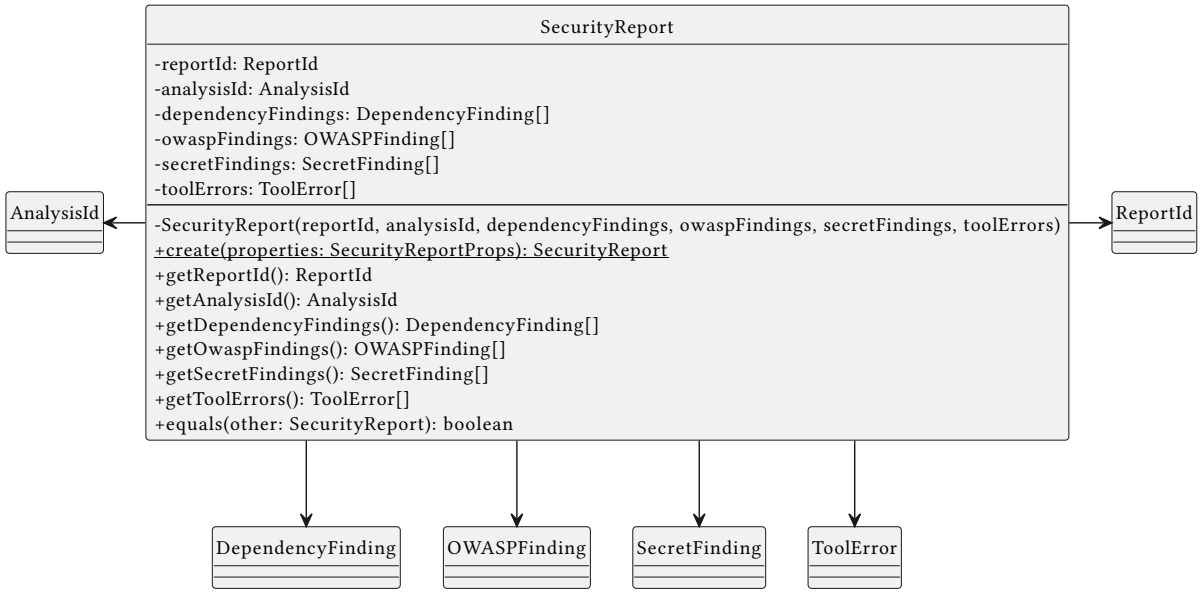

Figure 46: Class Diagram — SecurityReport

`SecurityReport` è l'entità che rappresenta il report prodotto dall'analisi di sicurezza per una specifica analisi. Aggrega i finding sulle dipendenze vulnerabili, le violazioni OWASP, i segreti esposti e gli eventuali errori degli strumenti di analisi, collegandosi all'analisi di appartenenza tramite identificatori.

- **Identità Basata sul Report:** Il metodo `equals()` confronta due istanze esclusivamente tramite `ReportId`, esprimendo la semantica fondamentale delle entity: due `SecurityReport` sono lo stesso report se condividono l'identità, indipendentemente dai dati aggregati.
- **Tre Assi di Sicurezza:** Le collezioni `dependencyFindings` (`DependencyFinding`), `owaspFindings` (`OWASPFinding`) e `secretFindings` (`SecretFinding`) modellano tre categorie distinte di problemi di sicurezza (standard OWASP, dipendenze vulnerabili e segreti), ciascuna con la propria semantica e struttura dati.
- **Tracciabilità degli Errori:** La collezione `toolErrors` di tipo `ToolError` preserva i fallimenti degli strumenti di analisi all'interno del report stesso, rendendo visibile anche un'esecuzione parziale senza perdere i risultati già raccolti.

3.2.1.4.4 Service

3.2.1.4.4.1 IPasswordProvider

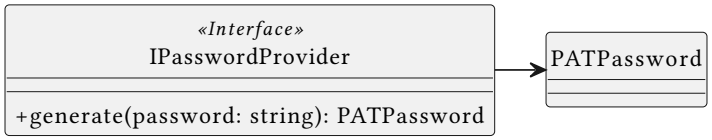

Figure 47: Class Diagram — IPasswordProvider

`IPasswordProvider` è l'interfaccia del Domain Service responsabile della trasformazione di una password in chiaro in un `PATPassword` (hash SHA-256 validato).

- **Separazione della Responsabilità:** Isola la logica di hashing dall'applicazione, permettendo di sostituire l'algoritmo di hashing senza modificare i servizi applicativi che dipendono da questo contratto.
- **Porta del Dominio:** Definisce un contratto che viene implementato dal Domain Service concreto `PATPasswordProvider`, seguendo il pattern Ports & Adapters anche all'interno del dominio.

3.2.1.4.4.2 ICodeReportEntityProvider

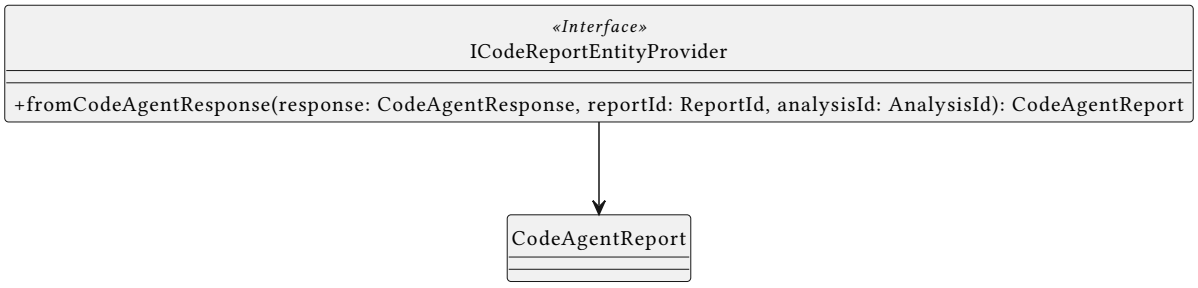

Figure 48: Class Diagram — ICodeReportEntityProvider

`ICodeReportEntityProvider` è l'interfaccia del Domain Service responsabile della costruzione di un'entità `CodeAgentReport` a partire dalla risposta grezza dell'agente di analisi del codice.

- **Porta del Dominio:** Definisce il contratto che separa il dominio dal layer applicativo, impedendo che la logica di mapping del JSON dell'agente penetri nelle entità di dominio.

- **Contestualizzazione:** Il metodo riceve `ReportId` e `AnalysisId` come parametri espliciti, garantendo che ogni entità prodotta sia immediatamente contestualizzata nel sistema senza ambiguità.

3.2.1.4.4.3 IDocsReportEntityProvider

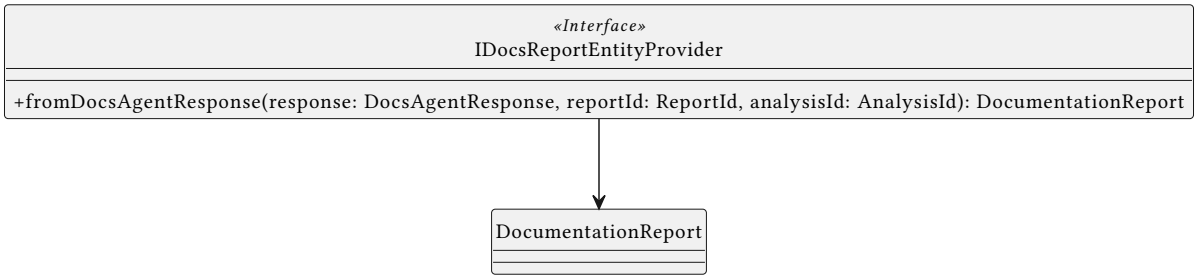

Figure 49: Class Diagram — IDocsReportEntityProvider

`IDocsReportEntityProvider` è l'interfaccia del Domain Service responsabile della costruzione di un'entità `DocumentationReport` a partire dalla risposta grezza dell'agente di analisi della documentazione.

- **Porta del Dominio:** Definisce il contratto che separa il dominio dal layer applicativo, impedendo che la logica di mapping del JSON dell'agente penetri nelle entità di dominio.
- **Contestualizzazione:** Il metodo riceve `ReportId` e `AnalysisId` come parametri espliciti, garantendo che ogni entità prodotta sia immediatamente contestualizzata nel sistema senza ambiguità.

3.2.1.4.4.4 PATPasswordProvider

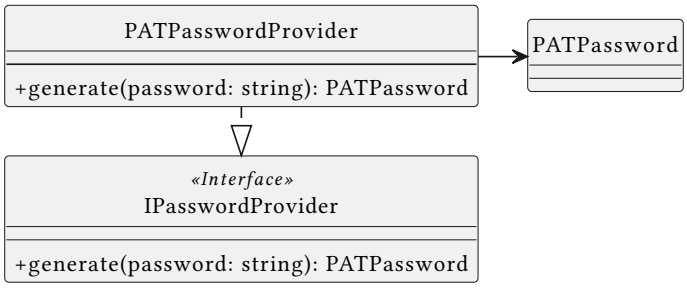

Figure 50: Class Diagram — PATPasswordProvider

`PATPasswordProvider` è l'implementazione concreta del Domain Service `IPasswordProvider`. Valida i requisiti di complessità della password (maiuscola, numero, carattere speciale, lunghezza minima) e produce un `PATPassword` tramite hash SHA-256.

- **Politica di Sicurezza:** Applica una politica di complessità esplicita prima dell'hashing, garantendo che solo password sufficientemente robuste possano essere usate per proteggere i PAT.
- **Immutabilità del Risultato:** Il risultato è sempre un `PATPassword` immutabile, che può circolare nel dominio senza rischio di esposizione della password originale.

3.2.1.4.4.5 ISecurityReportEntityProvider

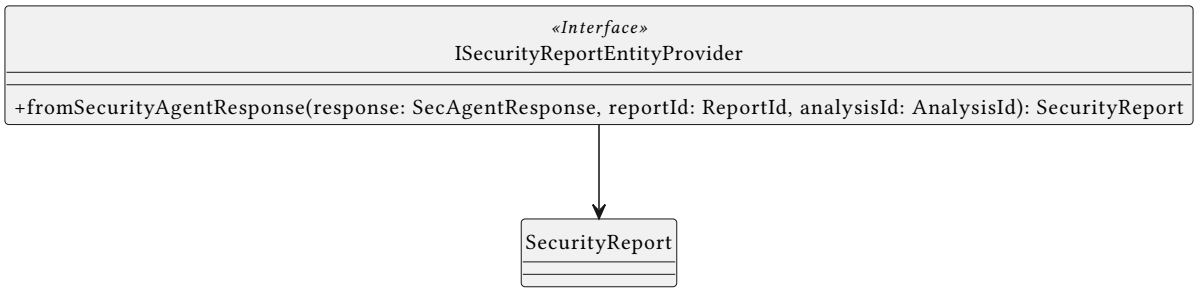

Figure 51: Class Diagram — ISecurityReportEntityProvider

`ISecurityReportEntityProvider` è l'interfaccia del Domain Service responsabile della costruzione di un'entità `SecurityReport` a partire dalla risposta grezza dell'agente di analisi di sicurezza.

- **Porta del Dominio:** Definisce il contratto che separa il dominio dal layer applicativo, impedendo che la logica di mapping del JSON dell'agente penetri nelle entità di dominio.
- **Contestualizzazione:** Il metodo riceve `ReportId` e `AnalysisId` come parametri espliciti, garantendo che ogni entità prodotta sia immediatamente contestualizzata nel sistema senza ambiguità.

3.2.1.4.4.6 ReportEntitiesProvider

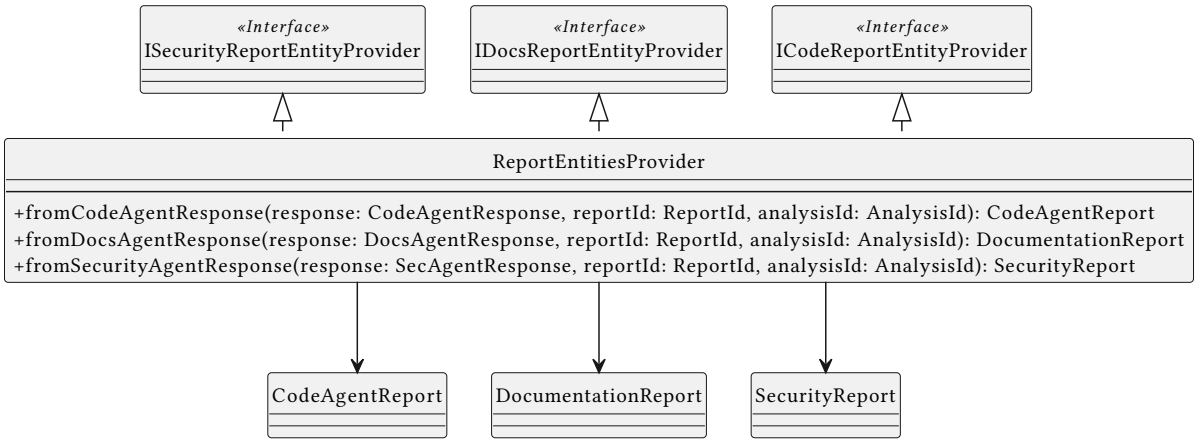

Figure 52: Class Diagram — ReportEntitiesProvider

`ReportEntitiesProvider` è il Domain Service concreto che implementa le tre interfacce `ICodeReportEntityProvider`, `IDocsReportEntityProvider` e `ISecurityReportEntityProvider`, centralizzando in un'unica classe tutta la logica di mapping dalle risposte grezze degli agenti alle entità di dominio.

- **Normalizzazione degli Alias:** Gestisce internamente le tabelle di alias (`STATUS_MISSING_ALIASES`, `SEVERITY_ALIASES`, `VERDICT_ALIASES`) che traducono i valori testuali non standardizzati prodotti dagli agenti nei valori type-safe delle enumerazioni di dominio, proteggendo le entità da input arbitrari.
- **Mapping Strutturato:** Per ciascun tipo di report, decompone la risposta JSON in Value Object atomici — costruendo ad esempio `KeyIssueReasoning`, `CriticalFileReasoning` e `AIInterpretation` per il report del codice — prima di assemblarli nell'entità finale.
- **Implementazione Tripla:** Una singola classe concreta implementa tre contratti distinti, ma viene esposta nel container NestJS tramite token separati per ciascuna interfaccia. Questa scelta riduce la duplicazione infrastrutturale preservando basso accoppiamento, aderenza al principio di

segregazione delle interfacce e sostituibilità indipendente dei port nei test e nelle evoluzioni future.

3.2.1.5 Application

Lo strato application è un livello architetturale che funge da intermediario tra il dominio del business e il mondo esterno (presentation, API, interfacce)

3.2.1.5.1 Command

Lo strato Application contiene i Command Object che rappresentano le richieste di azioni da parte dell'utente o di sistemi esterni. Questi oggetti sono progettati per essere semplici contenitori di dati (DTO) che trasportano le informazioni necessarie per eseguire un use case specifico, senza contenere logica di business o dipendenze verso il dominio.

3.2.1.5.1.1 AddRepositoryCollectionCommand

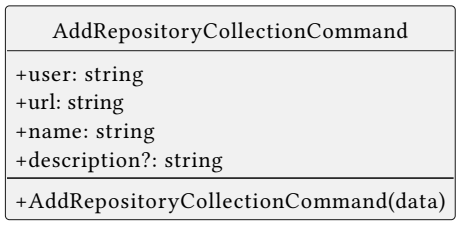

Figure 53: Class Diagram — AddRepositoryCollectionCommand

`AddRepositoryCollectionCommand` è il Command Object per la creazione di una nuova collezione di repository, trasportando l'identità dell'utente, l'URL del repository, il nome della collezione e una descrizione opzionale.

- **Oggetto di Trasferimento Validato:** I decorator `class-validator` garantiscono che i campi obbligatori siano presenti e non vuoti prima che il comando raggiunga il servizio applicativo.
- **Neutralità verso il Dominio:** Lavora con tipi primitivi (`string`), delegando la costruzione dei Value Object al servizio applicativo, rispettando la separazione tra layer applicativo e dominio.

3.2.1.5.1.2 DeletePatCommand

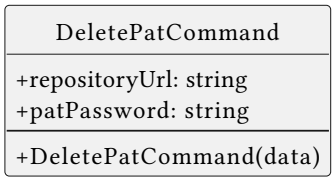

Figure 54: Class Diagram — DeletePatCommand

`DeletePatCommand` è il Command Object per l'eliminazione di un PAT esistente, richiedendo URL del repository e password di protezione come meccanismo di autorizzazione all'eliminazione.

- **Autorizzazione Implicita:** La richiesta della `patPassword` garantisce che solo chi conosce la password possa eliminare le credenziali, implementando un controllo di accesso a livello applicativo.

3.2.1.5.1.3 DeleteRepositoryCollectionCommand

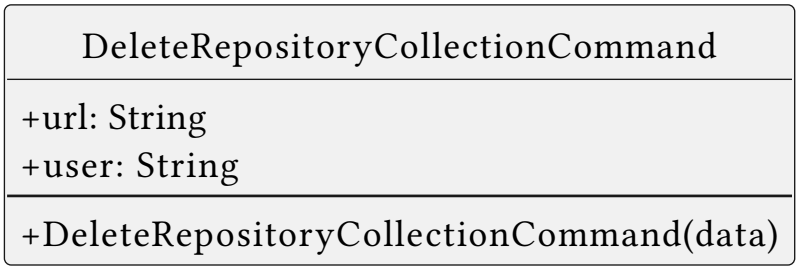

Figure 55: Class Diagram — DeleteRepositoryCollectionCommand

`DeleteRepositoryCollectionCommand` è il Command Object per l'eliminazione di una collezione di repository, identificando la collezione tramite URL e l'utente richiedente.

3.2.1.5.1.4 GetAllAnalysesForUserCommand

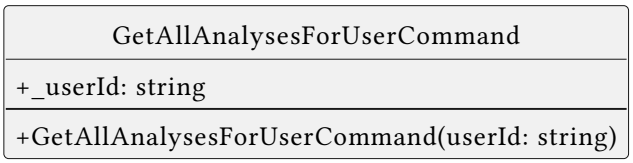

Figure 56: Class Diagram — GetAllAnalysesForUserCommand

`GetAllAnalysesForUserCommand` è il Command Object per il recupero di tutte le analisi associate a un utente, trasportando esclusivamente l'identificatore dell'utente richiedente.

- **Oggetto di Trasferimento Validato:** Il decorator `@IsNotEmpty()` garantisce che l'identificatore utente sia sempre presente prima che il comando raggiunga il servizio applicativo.

3.2.1.5.1.5 GetAllRepositoryCollectionsCommand

Figure 57: Class Diagram — GetAllRepositoryCollectionsCommand

`GetAllRepositoryCollectionsCommand` è il Command Object per il recupero di tutte le collezioni di repository associate a un utente, trasportando esclusivamente l'identificatore dell'utente richiedente.

- **Oggetto di Trasferimento Validato:** Il decorator `@IsNotEmpty()` garantisce che l'identificatore utente sia sempre presente prima che il comando raggiunga il servizio applicativo.

3.2.1.5.1.6 GetAnalysisFromIdCommand

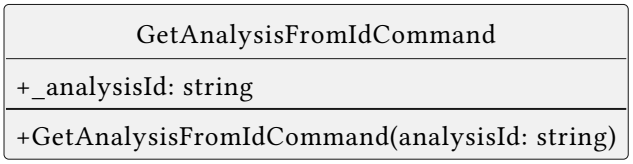

Figure 58: Class Diagram — GetAnalysisFromIdCommand

`GetAnalysisFromIdCommand` è il Command Object per il recupero di una singola analisi tramite il suo identificatore, trasportando esclusivamente l'`analysisId` come stringa primitiva.

- **Oggetto di Trasferimento Validato:** Il decorator `@IsNotEmpty()` garantisce che l'identificatore sia sempre presente prima che il comando raggiunga il servizio applicativo.

3.2.1.5.1.7 GetRepositoryCollectionCommand

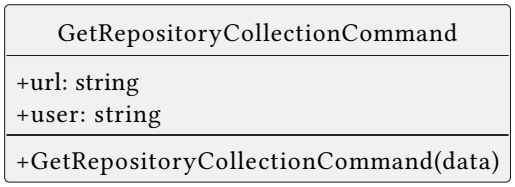

Figure 59: Class Diagram — GetRepositoryCollectionCommand

`GetRepositoryCollectionCommand` è il Command Object per il recupero di una specifica collezione di repository, identificandola tramite URL e utente richiedente.

- **Oggetto di Trasferimento Validato:** I decorator `class-validator` garantiscono che entrambi i campi siano presenti e non vuoti prima che il comando raggiunga il servizio applicativo.
- **Neutralità verso il Dominio:** Lavora con tipi primitivi (`string`), delegando la costruzione dei Value Object al servizio applicativo, rispettando la separazione tra layer applicativo e dominio.

3.2.1.5.1.8 NewPatCommand

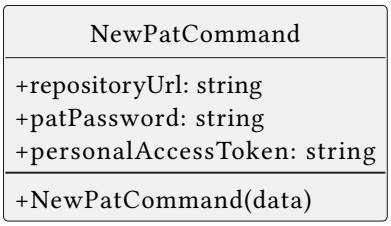

Figure 60: Class Diagram — NewPatCommand

`NewPatCommand` è il Command Object per la registrazione di un nuovo Personal Access Token, trasportando l'URL del repository, la password di protezione e il token PAT in chiaro.

- **Dati Sensibili in Transito:** Il PAT è presente in chiaro solo nel Command, che viene processato e dismesso immediatamente dopo l'esecuzione del use case, minimizzando il tempo di esposizione del segreto.

3.2.1.5.1.9 StartAnalysisCommand

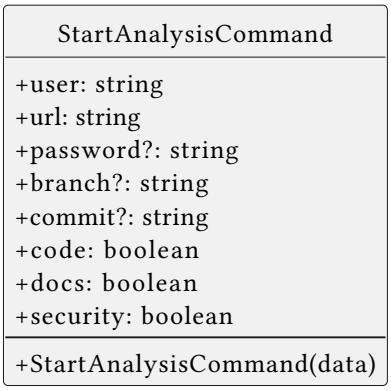

Figure 61: Class Diagram — StartAnalysisCommand

`StartAnalysisCommand` è il Command Object che trasporta i dati di input per avviare una nuova analisi: l'identità dell'utente, l'URL del repository, la password opzionale per repository privati, e i flag per i tre tipi di analisi (codice, documentazione, sicurezza).

- **Oggetto di Trasferimento Validato:** I decorator `class-validator` garantiscono che i campi obbligatori siano presenti e correttamente formattati prima che il comando raggiunga il servizio applicativo.
- **Neutralità verso il Dominio:** Lavora con tipi primitivi (`string`, `boolean`), delegando la costruzione dei Value Object al servizio `StartAnalysisService`, rispettando la separazione tra layer applicativo e dominio.

3.2.1.5.1.10 UpdatePatCommand

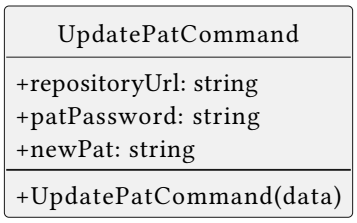

Figure 62: Class Diagram — UpdatePatCommand

`UpdatePatCommand` è il Command Object per l'aggiornamento di un PAT esistente, richiedendo URL, password corrente e il nuovo PAT da sostituire.

- **Sostituzione Sicura:** La verifica della password corrente (`patPassword`) prima della sostituzione impedisce aggiornamenti non autorizzati, garantendo che solo il proprietario delle credenziali possa modificarle.

3.2.1.5.2 Use Cases

I **Use Case** rappresentano i contratti applicativi che espongono le capacità del sistema verso il layer di presentazione. Ogni use case definisce un'operazione atomica del dominio applicativo (es. avvio analisi, gestione PAT, recupero collezioni) tramite un'interfaccia stabile, così che i controller dipendano da astrazioni e non da implementazioni concrete.

Nel microservizio di Analisi, i use case seguono un modello uniforme:
- ricevono un **Command Object** in input, che trasporta i parametri della richiesta;
- delegano l'orchestrazione ai servizi applicativi, che coordinano Domain Service e Port;

- restituiscono un **Result Object** tipizzato, che esplicita successo o fallimento senza esporre dettagli infrastrutturali.

Questa struttura consente disaccoppiamento, testabilità e sostituibilità delle implementazioni, mantenendo il confine tra Application Layer e Presentation Layer chiaro e verificabile.

3.2.1.5.2.1 AddRepositoryCollectionUseCase

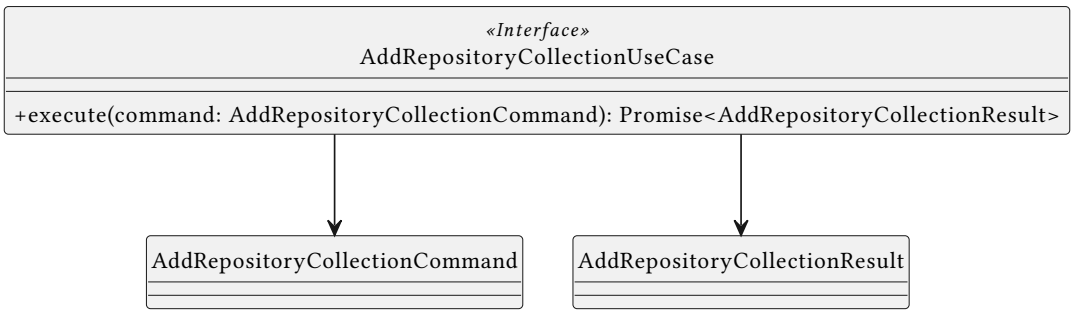

Figure 63: Class Diagram — AddRepositoryCollectionUseCase

`AddRepositoryCollectionUseCase` è l'interfaccia del use case per la creazione di una nuova collezione di repository, implementata dal servizio applicativo corrispondente.

3.2.1.5.2.2 DeletePatUseCase

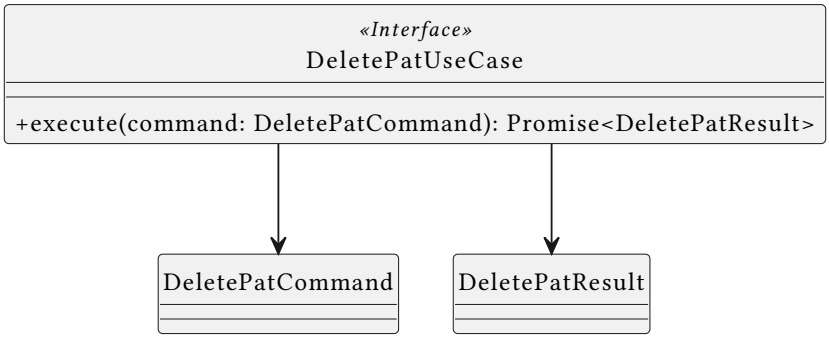

Figure 64: Class Diagram — DeletePatUseCase

`DeletePatUseCase` è l'interfaccia del use case per l'eliminazione di un PAT, implementata da `DeletePatService`.

3.2.1.5.2.3 DeleteRepositoryCollectionUseCase

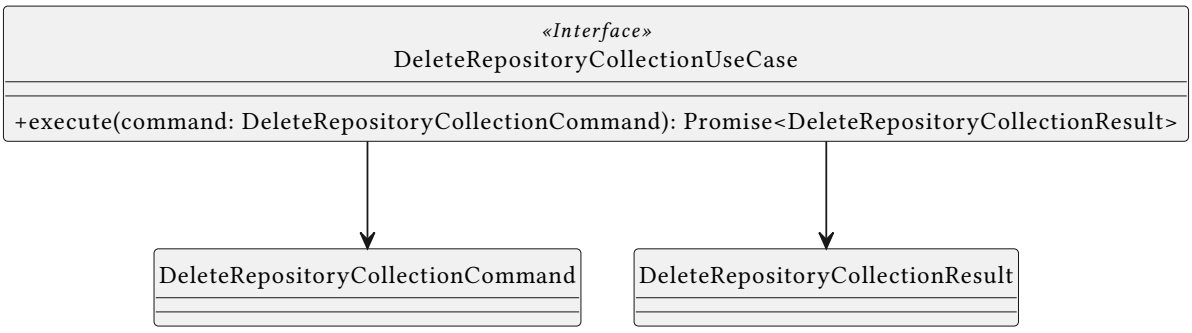

Figure 65: Class Diagram — DeleteRepositoryCollectionUseCase

`DeleteRepositoryCollectionUseCase` è l'interfaccia del use case per l'eliminazione di una collezione di repository, implementata da `GitHubCollectionDeleter`.

3.2.1.5.2.4 GetAllAnalysesForUserUseCase

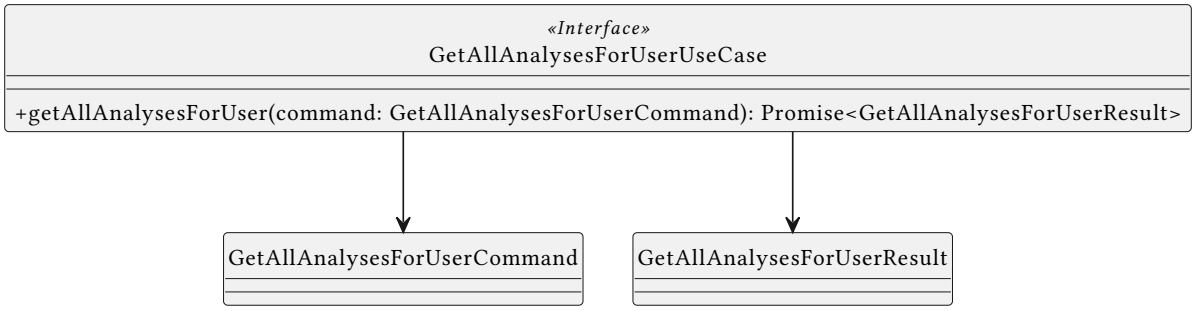

Figure 66: Class Diagram — GetAllAnalysesForUserUseCase

`GetAllAnalysesForUserUseCase` è l'interfaccia del use case per il recupero di tutte le analisi associate a un utente, implementata da `GetAnalysisService`.

3.2.1.5.2.5 GetAllRepositoryCollectionsUseCase

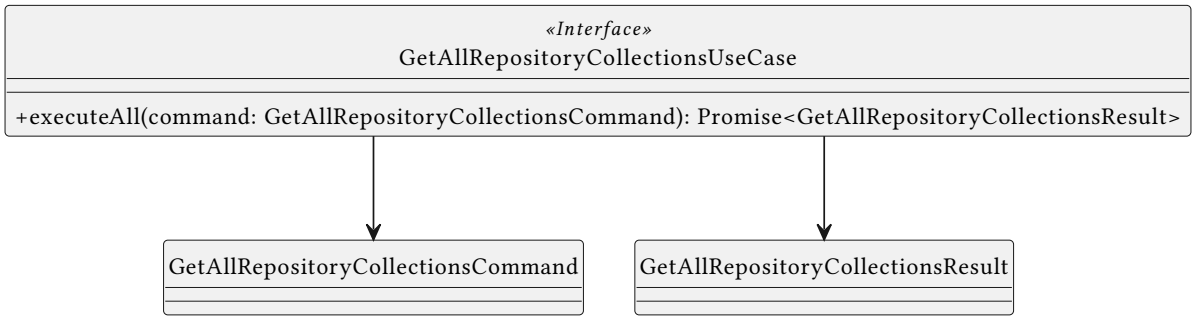

Figure 67: Class Diagram — GetAllRepositoryCollectionsUseCase

`GetAllRepositoryCollectionsUseCase` è l'interfaccia del use case per il recupero di tutte le collezioni di repository di un utente, implementata da `GitHubCollectionGetter`.

3.2.1.5.2.6 GetAnalysisUseCase

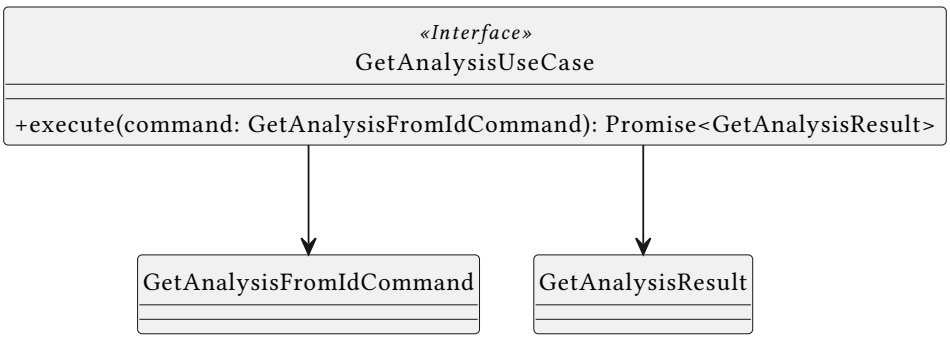

Figure 68: Class Diagram — GetAnalysisUseCase

`GetAnalysisUseCase` è l'interfaccia del use case per il recupero di una singola analisi tramite il suo identificatore, implementata da `GetAnalysisService`.

3.2.1.5.2.7 GetRepositoryCollectionUseCase

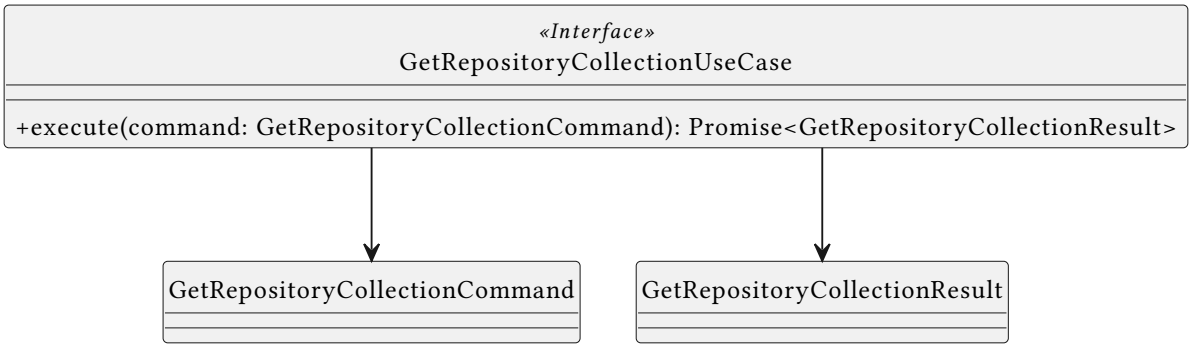

Figure 69: Class Diagram — GetRepositoryCollectionUseCase

`GetRepositoryCollectionUseCase` è l'interfaccia del use case per il recupero di una specifica collezione di repository tramite URL e utente, implementata da `GitHubCollectionGetter`.

3.2.1.5.2.8 NewPatUseCase

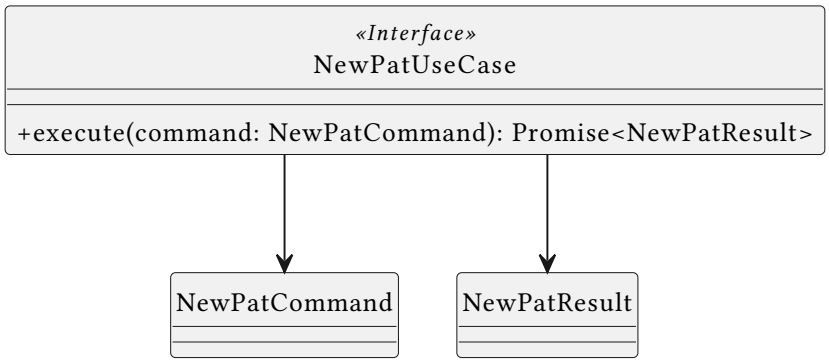

Figure 70: Class Diagram — NewPatUseCase

`NewPatUseCase` è l'interfaccia del use case per la registrazione di un nuovo PAT, implementata da `NewPatService`.

3.2.1.5.2.9 StartAnalysisUseCase

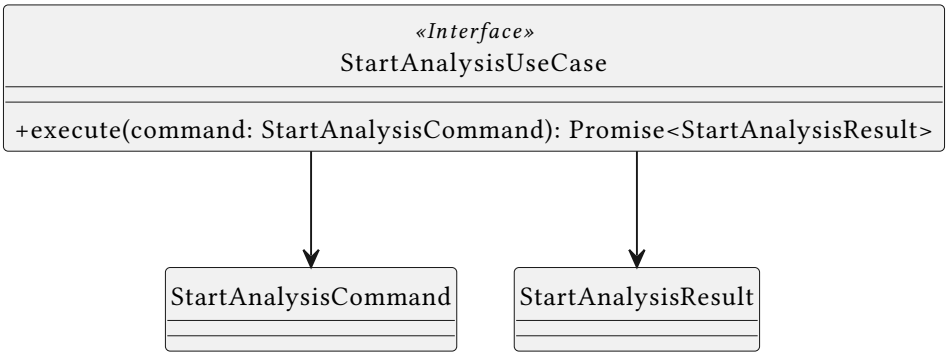

Figure 71: Class Diagram — StartAnalysisUseCase

`StartAnalysisUseCase` è l'interfaccia dello use case principale del sistema, ed é implementata da `StartAnalysisService`.

3.2.1.5.2.10 UpdatePatUseCase

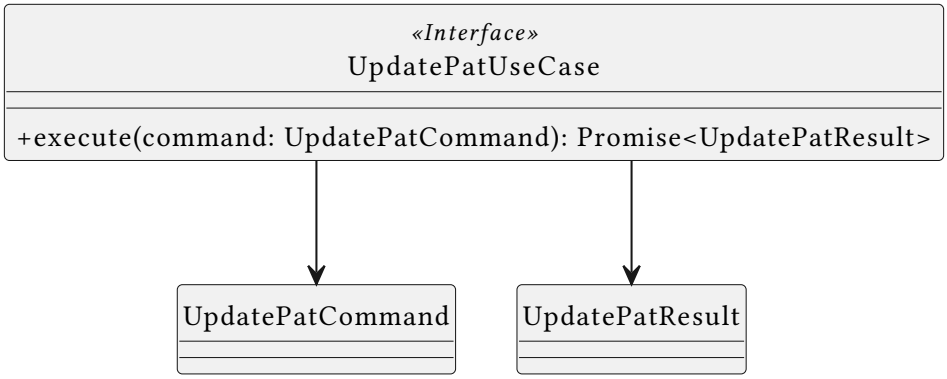

Figure 72: Class Diagram — UpdatePatUseCase

`UpdatePatUseCase` è l'interfaccia del use case per l'aggiornamento di un PAT, implementata da `UpdatePatService`.

3.2.1.5.3 Results

I result sono i contratti di risposta dello use case verso il layer di presentazione. Incapsulano sia i dati di output in caso di successo che le informazioni sull'eventuale fallimento, permettendo al controller di gestire in modo strutturato e type-safe le diverse casistiche di esito. Per questo il messaggio é inteso come di errore e non é presente nel caso di successo, mentre tutti i campi di output sono valorizzati solo in caso di successo (o impostati a null/array vuoto in caso di fallimento) evitando che il controller debba gestire dati parziali o inconsistenti in caso di fallimento.

3.2.1.5.3.1 AddRepositoryCollectionResult

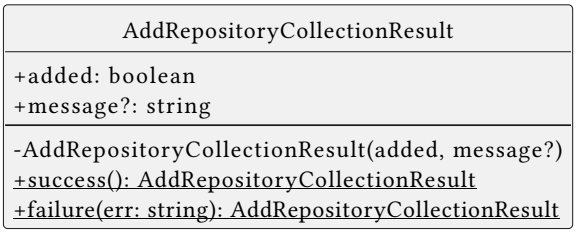

Figure 73: Class Diagram — AddRepositoryCollectionResult

`AddRepositoryCollectionResult` è il Result Object per l'esito della creazione di una nuova collezione di repository, con i metodi `success()` e `failure(err)` permette la sua creazione a stati di successo o fallimento.

3.2.1.5.3.2 DeletePatResult

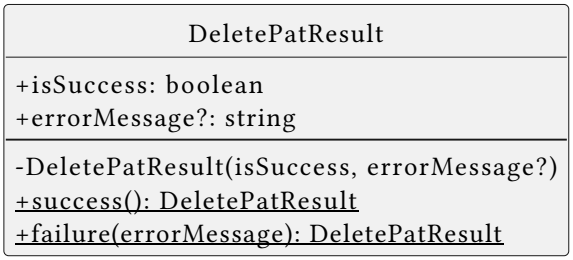

Figure 74: Class Diagram — DeletePatResult

`DeletePatResult` è il Result Object per l'esito dell'eliminazione di un PAT

3.2.1.5.3.3 DeleteRepositoryCollectionResult

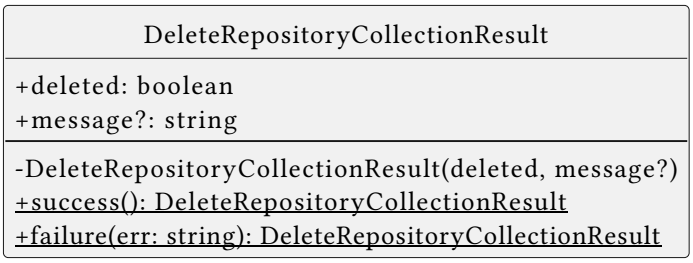

Figure 75: Class Diagram — DeleteRepositoryCollectionResult

`DeleteRepositoryCollectionResult` è il Result Object per l'esito dell'eliminazione di una collezione di repository, con factory method `success()` e `failure(err)`.

3.2.1.5.3.4 GetAllAnalysesForUserResult

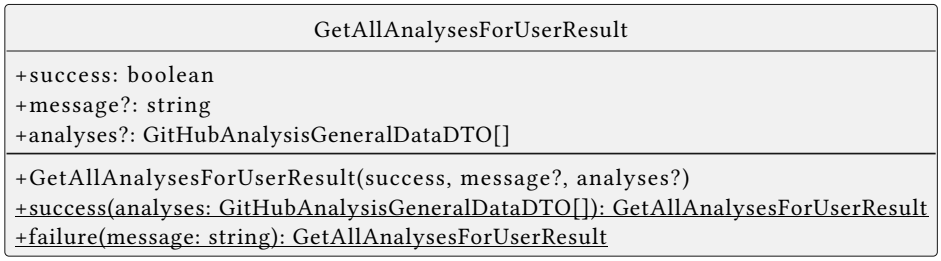

Figure 76: Class Diagram — GetAllAnalysesForUserResult

`GetAllAnalysesForUserResult` è il Result Object per l'esito del recupero di tutte le analisi associate a un utente, trasportando in caso di successo la collezione di `GitHubAnalysisGeneralDataDTO`.

3.2.1.5.3.5 GetAllRepositoryCollectionsResult

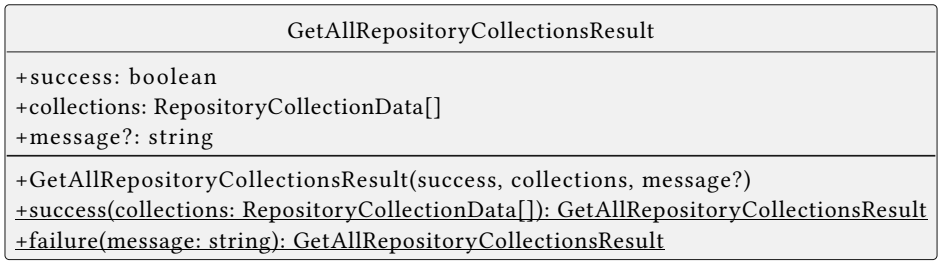

Figure 77: Class Diagram — GetAllRepositoryCollectionsResult

`GetAllRepositoryCollectionsResult` è il Result Object per l'esito del recupero di tutte le collezioni di repository di un utente, trasportando in caso di successo la collezione di `RepositoryCollectionData`.

- **Fallimento con Array Vuoto:** Il factory method `failure()` inizializza `collections` a array vuoto, garantendo che il layer di presentazione non riceva mai `undefined` al posto di una collezione iterabile.

3.2.1.5.3.6 GetAnalysisResult

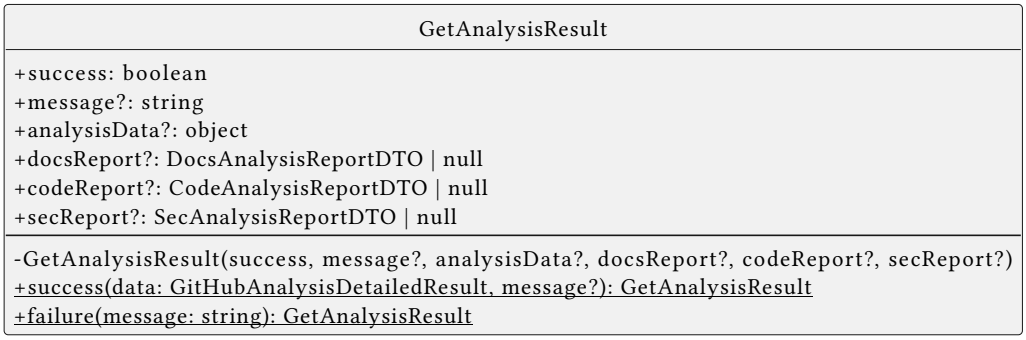

Figure 78: Class Diagram — GetAnalysisResult

`GetAnalysisResult` è il Result Object per l'esito del recupero di una singola analisi, aggregando i metadati identificativi dell'analisi e i tre report opzionali (codice, documentazione, sicurezza).

- **Report Opzionali:** I campi `docsReport`, `codeReport` e `secReport` sono nullable, riflettendo il fatto che un'analisi può coinvolgere solo un sottoinsieme dei tre tipi di report in base a quanto richiesto.

3.2.1.5.3.7 GetRepositoryCollectionResult

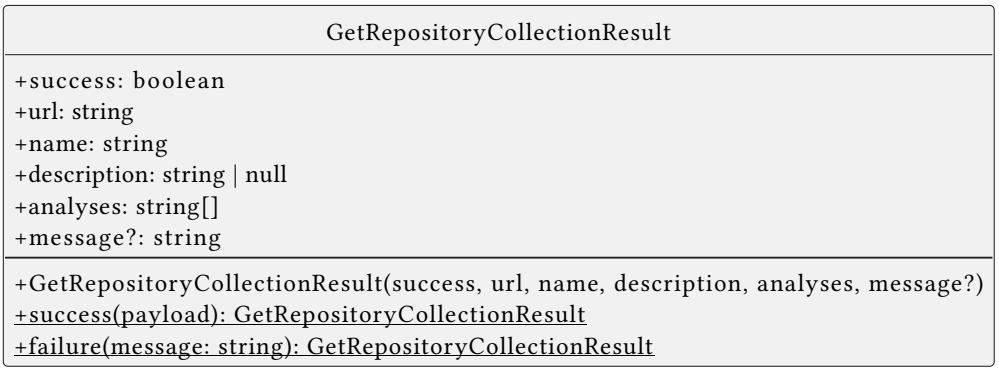

Figure 79: Class Diagram — GetRepositoryCollectionResult

`GetRepositoryCollectionResult` è il Result Object per l'esito del recupero di una specifica collezione di repository, trasportando URL, nome, descrizione opzionale e lista degli identificatori delle analisi associate.

- **Fallimento Strutturato:** Il factory method `failure()` inizializza i campi obbligatori con valori vuoti, garantendo che il result sia sempre deserializzabile dal layer di presentazione indipendentemente dall'esito.

3.2.1.5.3.8 NewPatResult

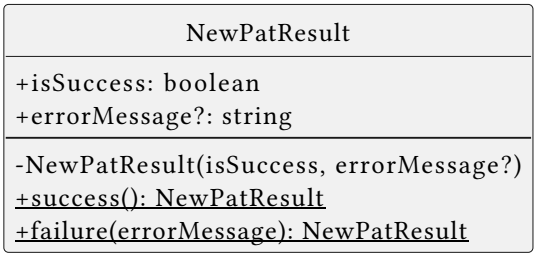

Figure 80: Class Diagram — NewPatResult

`NewPatResult` è il Result Object per l'esito della registrazione di un PAT, con factory method `success()` e `failure(errorMessage)`.

- **Semplicità Intenzionale:** Il risultato è binario (successo/fallimento), riflettendo la natura atomica dell'operazione di salvataggio delle credenziali.

3.2.1.5.3.9 StartAnalysisResult

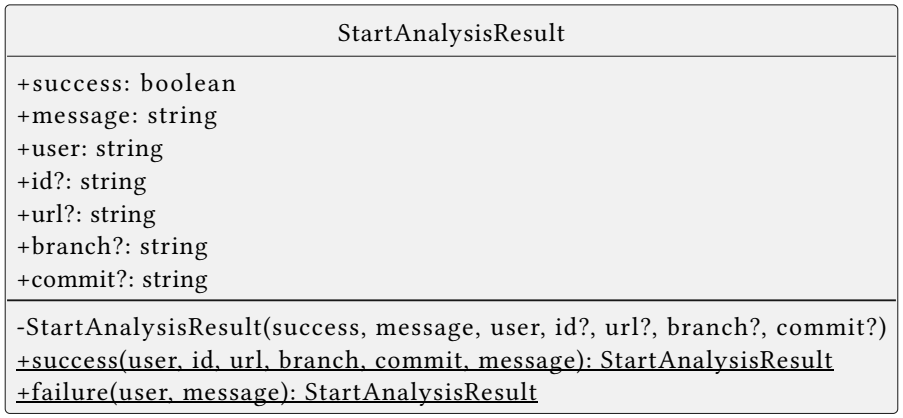

Figure 81: Class Diagram — StartAnalysisResult

`StartAnalysisResult` è il Result Object che trasporta l'esito dell'avvio di un'analisi verso il layer di presentazione, includendo in caso di successo tutti i metadati identificativi dell'analisi creata.

- **Pattern Result:** I factory method `success()` e `failure()` rendono esplicito l'esito dell'operazione, evitando eccezioni non gestite come meccanismo di controllo del flusso.
- **Risposta Completa:** Il successo restituisce `user`, `id`, `url`, `branch` e `commit`, fornendo al client tutte le informazioni necessarie per tracciare e monitorare l'analisi avviata.

3.2.1.5.3.10 UpdatePatResult

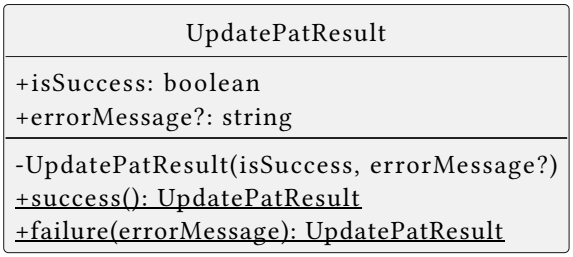

Figure 82: Class Diagram — UpdatePatResult

`UpdatePatResult` è il Result Object per l'esito dell'aggiornamento di un PAT.

3.2.1.5.4 Service

Gli application service sono le implementazioni concrete dei use case, orchestrano la logica di business e coordinano l'interazione con i Domain Service, i Repository e le altre dipendenze necessarie per realizzare i requisiti del dominio. Ogni servizio applicativo implementa una o piú use cases, permettendo al layer di presentazione di dipendere solo da contratti astratti e facilitando la sostituzione o evoluzione delle implementazioni senza impattare il controller.

3.2.1.5.4.1 Interfacce degli Helper Services

Queste servono a disaccoppiare la logica specifica di alcune operazioni (es. orchestrazione degli agenti, verifica duplicati, autorizzazione, validazione, clonazione) dal servizio applicativo principale, permettendo di sostituire le strategie implementative senza modificare i servizi che le utilizzano.

3.2.1.5.4.1.1 IAnalysisOrchestrator

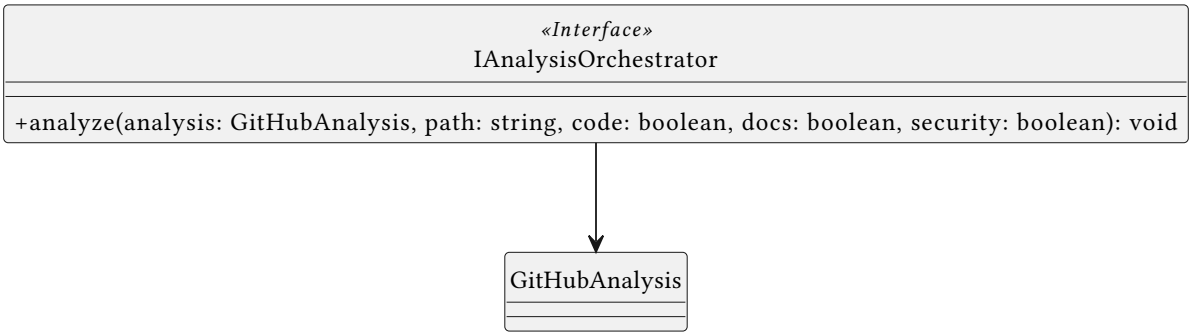

Figure 83: Class Diagram — IAnalysisOrchestrator

`IAnalysisOrchestrator` è l'interfaccia del servizio applicativo che coordina l'esecuzione degli agenti di analisi (codice, documentazione, sicurezza) su un repository clonato.

- **Orchestrazione Asincrona:** Il metodo `analyze()` è `void` (fire-and-forget), permettendo al chiamante di avviare l'analisi e rispondere immediatamente al client senza attendere il completamento dei processi AI.
- **Punto di Estensione:** L'interfaccia permette di sostituire la strategia di orchestrazione (es. sequenziale vs. parallela, locale vs. distribuita) senza modificare l'utilizzatore `StartAnalysisService`.

3.2.1.5.4.1.2 ICollectionExistenceChecker

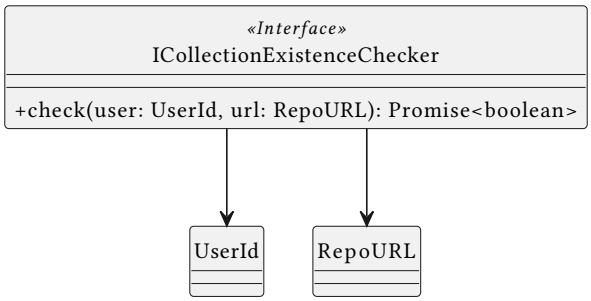

Figure 84: Class Diagram — ICollectionExistenceChecker

`ICollectionExistenceChecker` è l'interfaccia del servizio che verifica l'esistenza di una collezione di repository per un dato utente, restituendo un booleano che indica la presenza di una collection corrispondente a user e url.

- **Punto di Estensione:** Disaccoppia la logica di verifica duplicati dall'implementazione concreta `GitHubCollectionChecker`, permettendo di sostituire la strategia di controllo senza modificare i servizi applicativi che la utilizzano.

3.2.1.5.4.1.3 IRepositoryAuthorizer

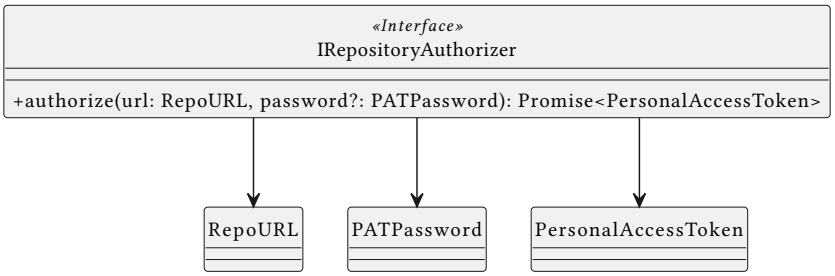

Figure 85: Class Diagram — IRepositoryAuthorizer

`IRepositoryAuthorizer` è l'interfaccia del servizio che recupera un `PersonalAccessToken` valido per un dato repository, gestendo la distinzione tra repository pubblici e privati.

- **Strategia di Autorizzazione:** Nasconde la logica di selezione tra `PublicAuthorizationStrategy` (token di sistema utilizzato per il cloning tramite https per repositories pubbliche) e `PrivateAuthorizationStrategy` (token utente presente nel database per repositories private), esponendo un'unica API uniforme al use case.

3.2.1.5.4.1.4 IRepositoryCloner

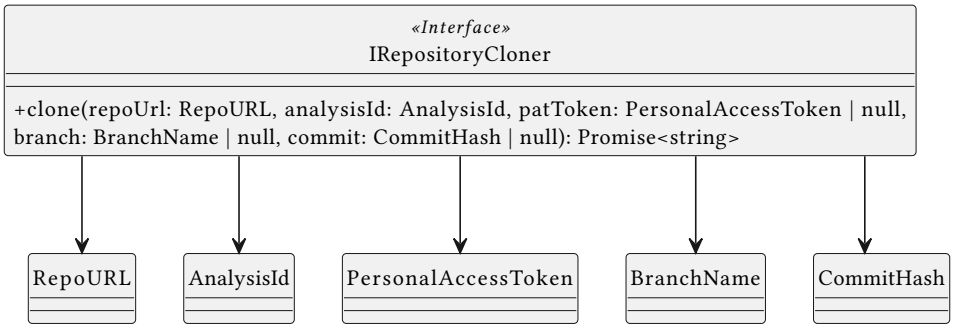

Figure 86: Class Diagram — IRepositoryCloner

`IRepositoryCloner` è l'interfaccia del servizio che esegue la clonazione fisica del repository su filesystem locale, restituendo il percorso della cartella clonata.

- **Isolamento dell'Infrastruttura:** Nasconde i dettagli operativi della clonazione Git (comandi git, gestione autenticazione, path di destinazione) al servizio utilizzatore, che opera solo sul percorso risultante.

3.2.1.5.4.1.5 IRepositoryValidator

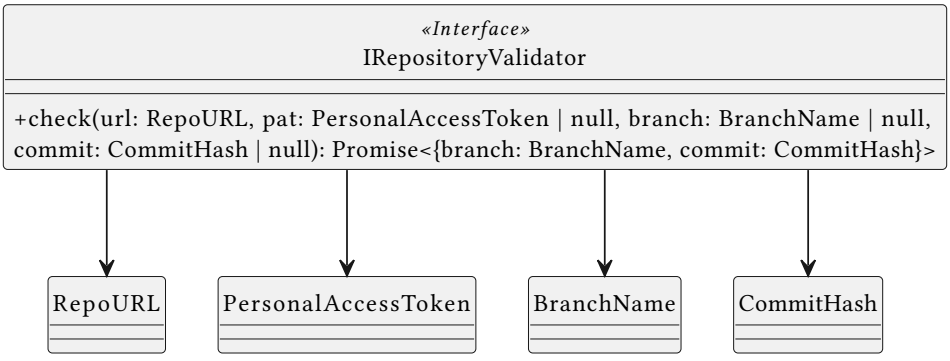

Figure 87: Class Diagram — IRepositoryValidator

`IRepositoryValidator` è l'interfaccia del servizio che verifica la raggiungibilità e la validità di un repository GitHub, risolvendo branch e commit a valori concreti in modo da evitare failure successive durante la clonazione o l'analisi.

- **Risoluzione dei Parametri:** Il metodo `check()` restituisce sempre `{ branch: BranchName; commit: CommitHash }` valori risolti, garantendo che l'analisi parta sempre da riferimenti esatti e non ambigui e lancia un exception esplicita in caso di problemi di raggiungibilità o validità del repository, che può essere gestita dal servizio chiamante per restituire un errore strutturato al client.

3.2.1.5.4.2 Helper Services

Questi sono application services che non implementano uno use case diretto, ma forniscono funzionalità specifiche (es. validazione, autorizzazione, controllo duplicati, orchestrazione) che vengono utilizzate dai servizi dei use case per realizzare la logica di business richiesta. Questi helper services permettono di isolare e sostituire facilmente strategie specifiche senza modificare i servizi applicativi principali.

3.2.1.5.4.2.1 AnalysisOrchestratorService

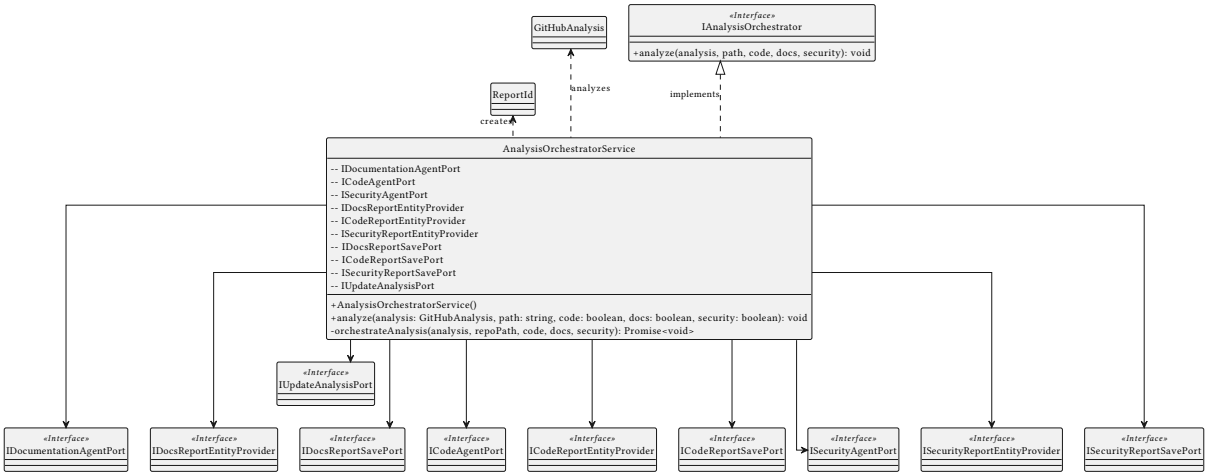

Figure 88: Class Diagram — AnalysisOrchestratorService

`AnalysisOrchestratorService` implementa `IAnalysisOrchestrator`, avviando in modo asincrono (fire-and-forget) l'orchestrazione degli agenti AI sul repository clonato. In seguito valida i risultati e li trasforma in entità di dominio prima di salvarli nel sistema di persistenza e associarli all'analisi.

I passaggi chiave dell'orchestrazione sono:

- Gli agenti vengono eseguiti in parallelo tramite `Promise.all()`, massimizzando l'efficienza temporale dell'analisi complessiva. Ogni agente é gestito da un adapter specifico e la sua implementazione é nascosta dietro un'interfaccia(`ICodeAgentPort`, `IDocsAnalysisAgent`, `ISecurityAgent`), permettendo di sostituire o modificare gli agenti senza impattare l'orchestratore.
- Le risposte degli agenti (in formato JSON grezzo) vengono trasformate in entità di dominio tramite un service che implementa `IDocsReportEntityProvider`, `ICodeReportEntityProvider` e `ISecurityReportEntityProvider`, permettendo di validare e contestualizzare i dati prima di inserirli nel dominio.
- I report validati vengono salvati nel sistema di persistenza tramite `ISecurityReportSavePort`, `ICodeReportSavePort` e `IDocsReportSavePort`, che si occupano di gestire la persistenza dei report e le eventuali relazioni con l'entità dell'analisi.
- I report validati vengono associati all'analisi tramite `IUpdateAnalysisPort`, che si occupa di aggiornare l'entità nel database.

3.2.1.5.4.2.2 GitAuthorizerService

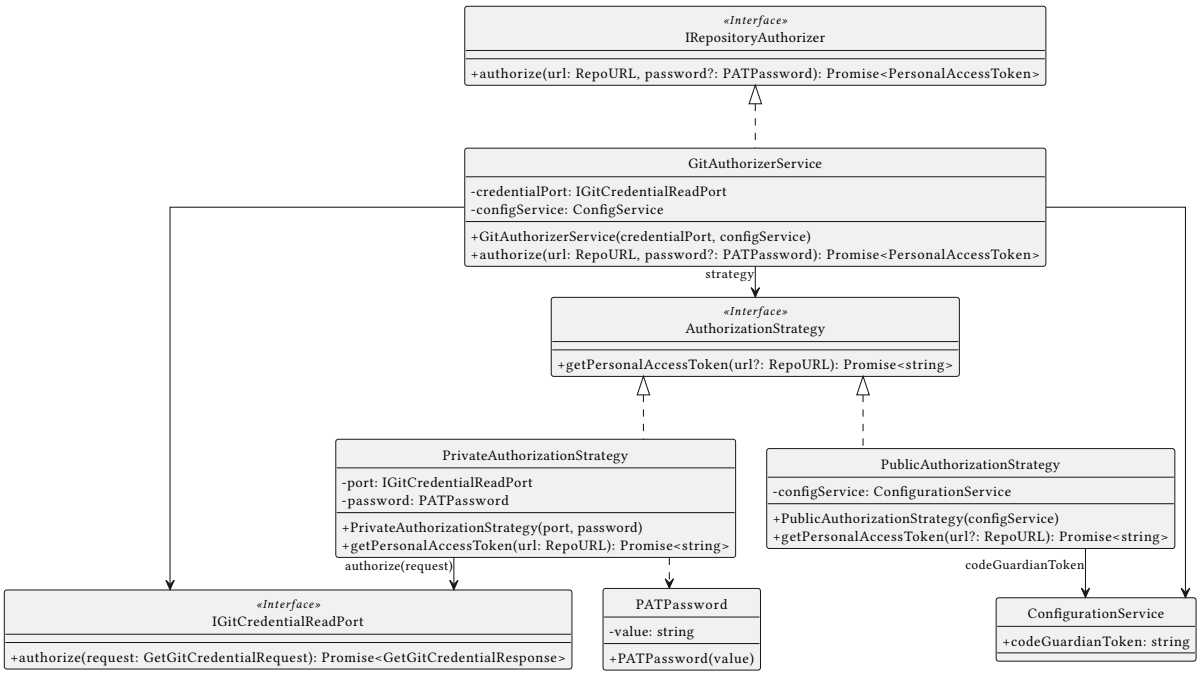

Figure 89: Class Diagram — GitAuthorizerService

`GitAuthorizerService` implementa `IRepositoryAuthorizer` utilizzando un pattern `Strategy Pattern` per gestire in modo trasparente l'autorizzazione sia per repository pubblici che privati.

- **Strategia Pubblica vs Privata:** Se il repository è pubblico, utilizza `PublicAuthorizationStrategy` che fornisce un token di sistema per l'accesso in sola lettura. Se il repository è privato, utilizza `PrivateAuthorizationStrategy` che recupera il token utente dal database tramite `IGitCredentialReadPort` e lo restituisce per l'autenticazione.
- **Isolamento della Logica di Autorizzazione:** Il servizio nasconde completamente i dettagli dell'autorizzazione al servizio chiamante, che riceve semplicemente un `PersonalAccessToken` valido indipendentemente dalla natura del repository, semplificando la logica del use case e permettendo di modificare le strategie di autorizzazione senza impattare i servizi applicativi.

3.2.1.5.4.2.3 GitClonerService

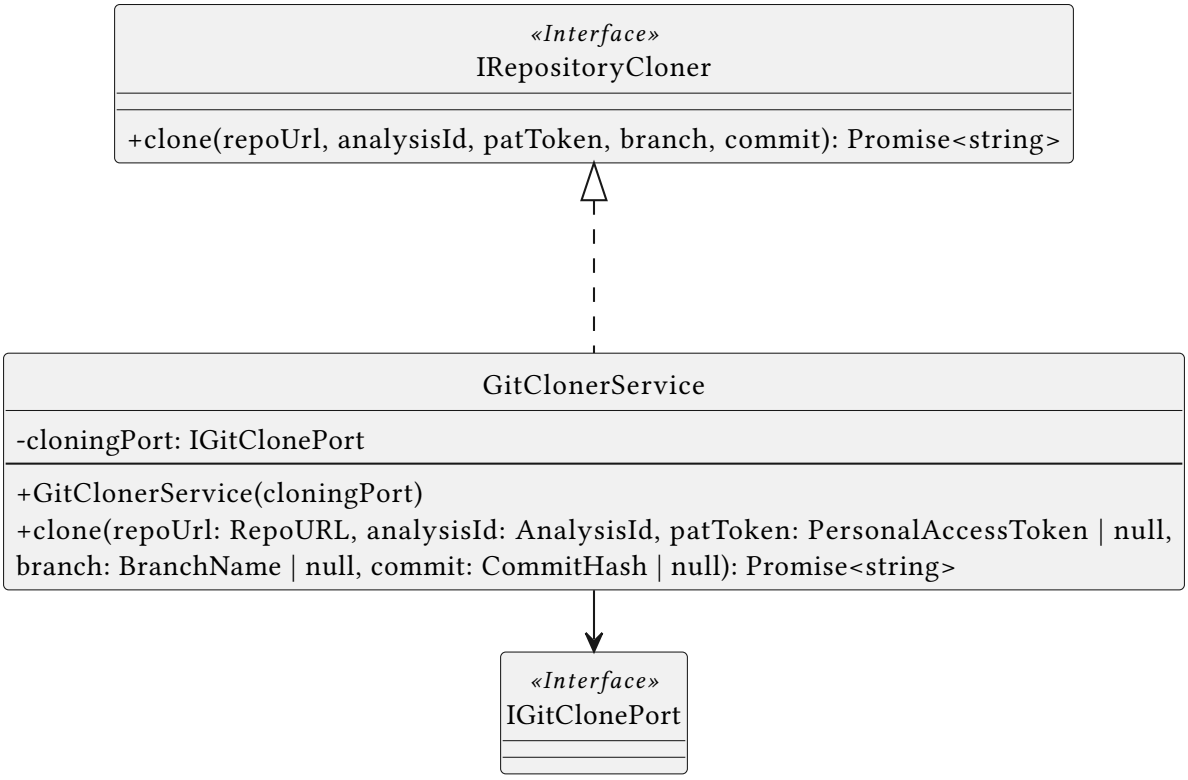

Figure 90: Class Diagram — GitClonerService

`GitClonerService` implementa `IRepositoryCloner` delegando l'operazione di clonazione al port `IGitClonePort`, costruendo il `CloneRepoRequest` e gestendo l'esito.

- **Adattamento del Contratto:** Traduce i parametri del servizio applicativo (Value Objects) nel DTO di richiesta per l'infrastruttura, e solleva un'eccezione esplicita in caso di fallimento della clonazione.

3.2.1.5.4.2.4 GitHubCollectionChecker

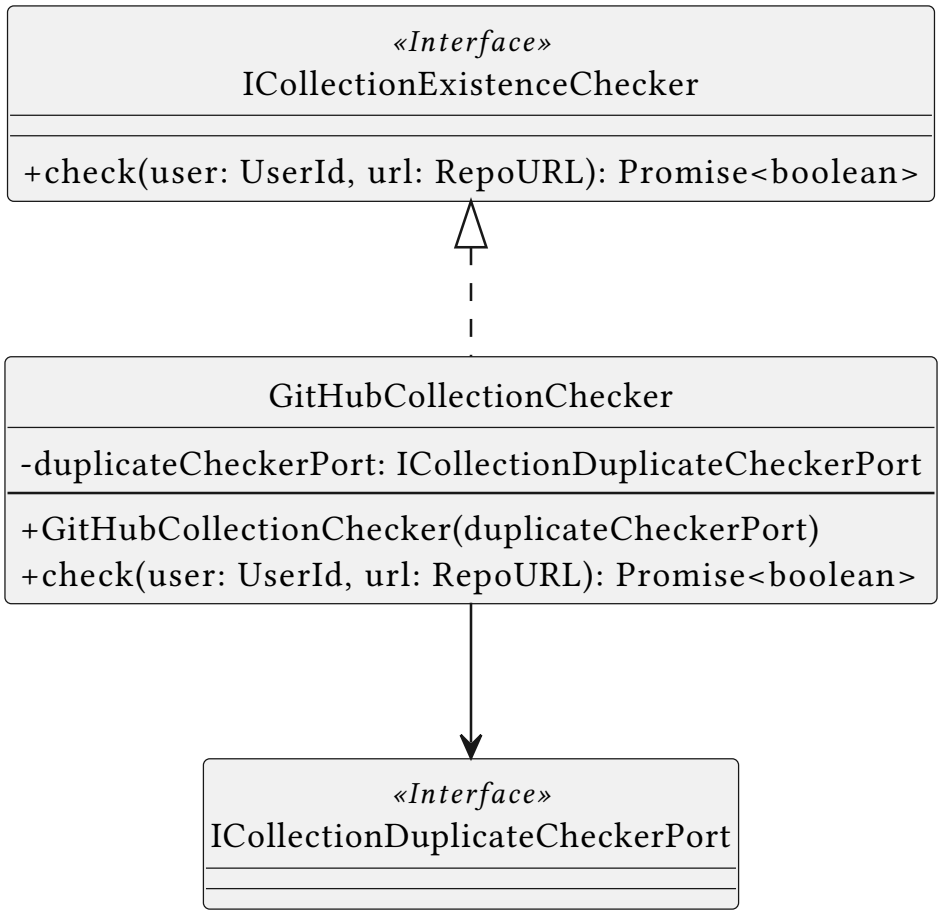

Figure 91: Class Diagram — GitHubCollectionChecker

`GitHubCollectionChecker` implementa `ICollectionExistenceChecker`, delegando la verifica di duplicati al port `ICollectionDuplicateCheckerPort` e restituendo il booleano estratto dalla risposta.

- **Adattamento del Contratto:** Traduce i Value Object `UserId` e `RepoURL` nel DTO di richiesta per l'infrastruttura, isolando il layer applicativo dai dettagli della persistenza.

3.2.1.5.4.2.5 GitValidatorService

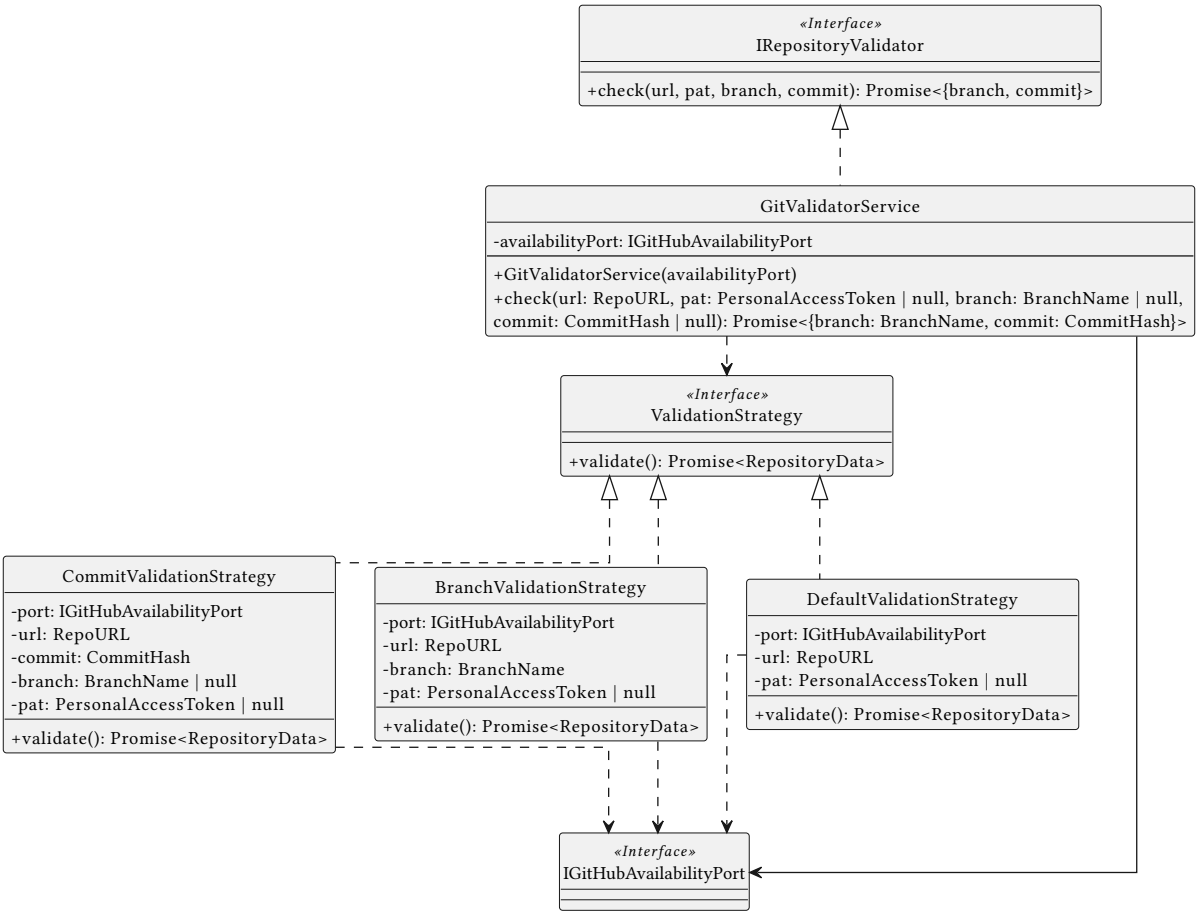

Figure 92: Class Diagram — GitValidatorService

`GitValidatorService` implementa `IRepositoryValidator` selezionando la strategia di validazione appropriata tramite uno `Strategy Pattern` in base alla natura del repository (pubblico vs privato) e delegando la validazione al port `IGitHubAvailabilityPort`.

Di default usa solo l' URL e prende il branch di defaul all'ultimo commit, se é richiesto un branch specifico o un commit specifico, verifica che esistano e siano raggiungibili, restituendo i valori risolti per URL, branch e commit in caso di successo o lanciando un'eccezione esplicita in caso di problemi di raggiungibilità o validità del repository.

3.2.1.5.4.3 Application Services

Questa sezione include i servizi che implementano direttamente i use case, orchestrando la logica di business e coordinando le dipendenze necessarie per realizzare i requisiti del dominio. Ogni servizio applicativo implementa uno o più use case, permettendo la modifica della logica applicativa senza impattare il layer di presentazione, che dipende solo dalle interfacce dei use case.

3.2.1.5.4.3.1 AddRepositoryCollectionService

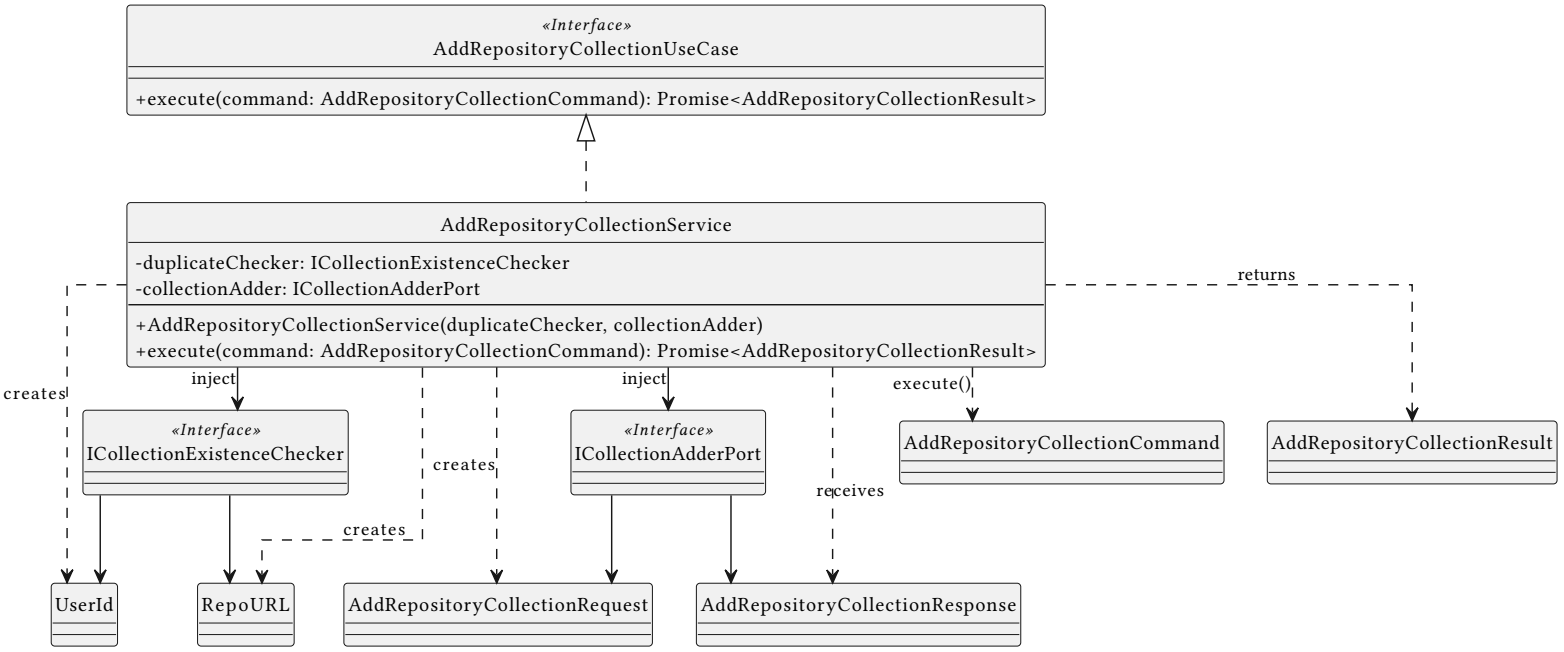

Figure 93: Class Diagram — AddRepositoryCollectionService

`AddRepositoryCollectionService` implementa `AddRepositoryCollectionUseCase`, orchestrando il controllo dei duplicati e la persistenza della nuova collezione.

- **Guardia Anti-Duplicato:** Prima di procedere alla creazione, interroga `ICollectionExistenceChecker` per verificare che non esista già una collezione per l'URL fornito, restituendo un fallimento esplicito in caso positivo.

3.2.1.5.4.3.2 DeletePatService

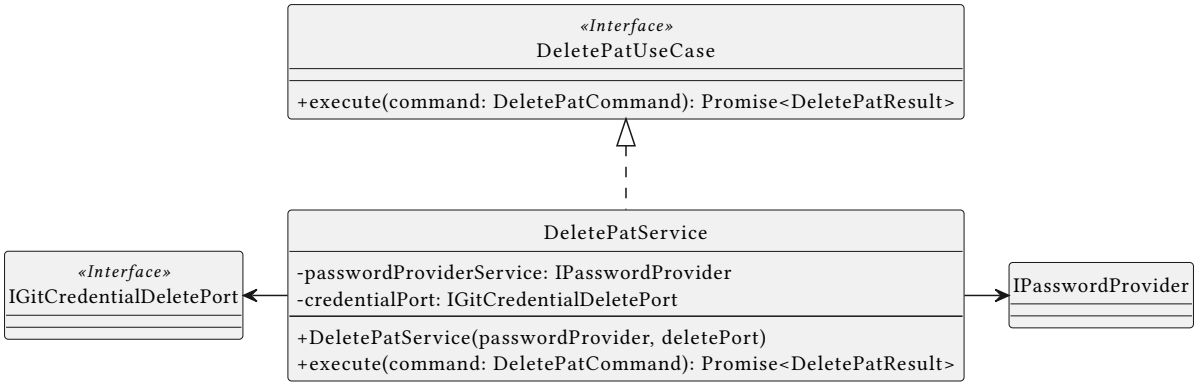

Figure 94: Class Diagram — DeletePatService

`DeletePatService` implementa `DeletePatUseCase`, costruendo la richiesta di eliminazione con URL e hash della password validando quest'ultima tramite il Domain Service `IPasswordProvider`, e delegando a `IGitCredentialDeletePort` cl'effettiva eliminazione dal sistema di persistenza.

3.2.1.5.4.3.3 GetAnalysisService

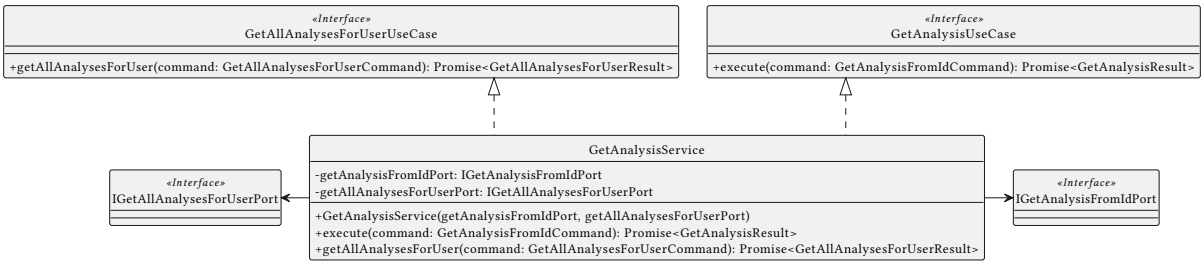

Figure 95: Class Diagram — GetAnalysisService

`GetAnalysisService` implementa simultaneamente `GetAnalysisUseCase` e `GetAllAnalysesForUserUseCase`, centralizzando in un'unica classe i due use case di lettura delle analisi.

- **Implementazione Doppia:** Riunisce il recupero di una singola analisi per ID e il recupero di tutte le analisi per utente, evitando la proliferazione di classi per operazioni di lettura strettamente correlate. Per ciascuna richiesta delega la query alla port di riferimento specifica per quell'operazione (`IGetAnalysisFromIdPort` e `IGetAllAnalysesForUserPort`) e traduce i risultati nel result applicativo corrispondente.

3.2.1.5.4.3.4 GitHubCollectionDeleter

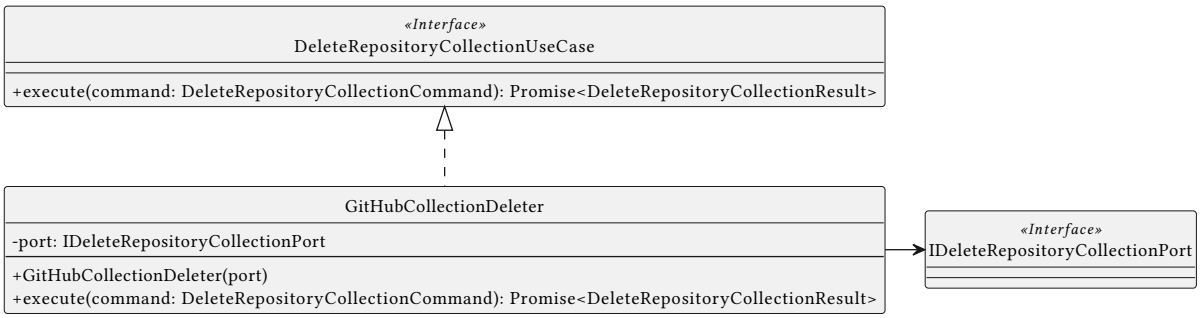

Figure 96: Class Diagram — GitHubCollectionDeleter

`GitHubCollectionDeleter` implementa `DeleteRepositoryCollectionUseCase`, delegando l'eliminazione della collezione al port `IDeleteRepositoryCollectionPort` e traducendo la risposta nel result applicativo e gestendo eventuali errori di recupero con risultati strutturati.

3.2.1.5.4.3.5 GitHubCollectionGetter

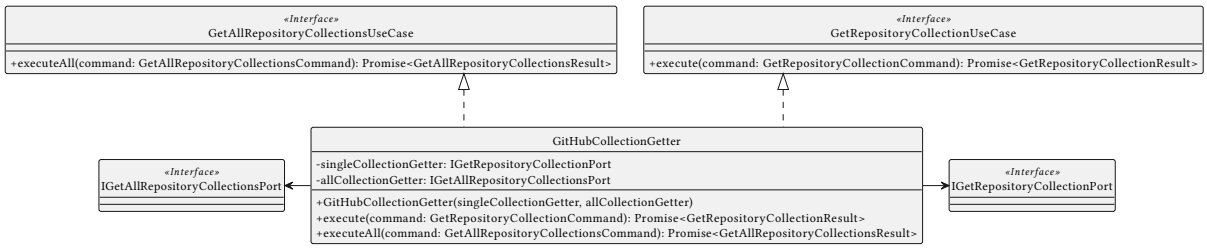

Figure 97: Class Diagram — GitHubCollectionGetter

`GitHubCollectionGetter` implementa simultaneamente `GetRepositoryCollectionUseCase` e `GetAllRepositoryCollectionsUseCase`, centralizzando in un'unica classe i due use case di lettura delle collezioni delegando le query alle rispettive port di riferimento (`IGetRepositoryCollectionPort` e `IGetAllRepositoryCollectionsPort`) e traducendo i risultati nei result applicativi corrispondenti.

- **Implementazione Doppia:** Riunisce il recupero di una singola collezione per URL e utente e il recupero di tutte le collezioni di un utente, evitando la proliferazione di classi per operazioni di lettura strettamente correlate.

3.2.1.5.4.3.6 NewPatService

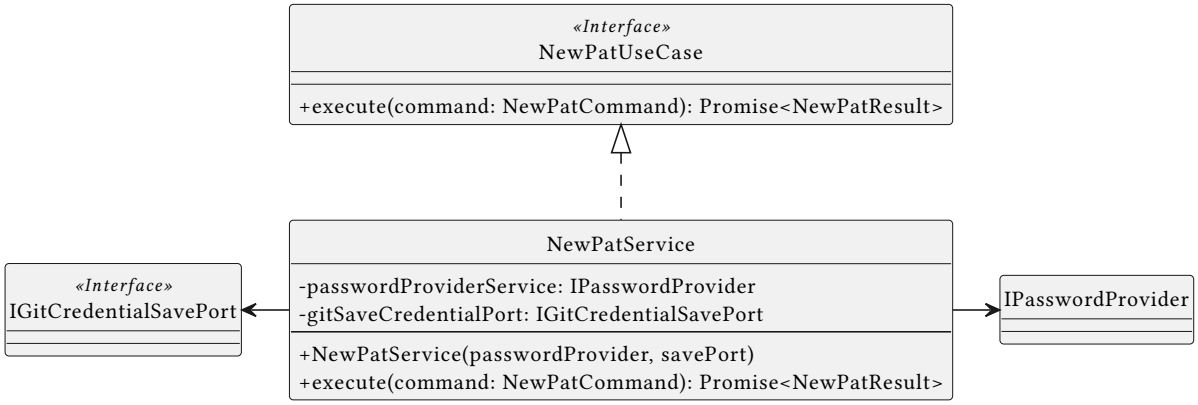

Figure 98: Class Diagram — NewPatService

`NewPatService` implementa `NewPatUseCase`, orchestrando la validazione del PAT, delegando l'hashing della password al domain service `IPasswordProvider` e il salvataggio alla porta `IGitCredentialSavePort`.

3.2.1.5.4.3.7 StartAnalysisService

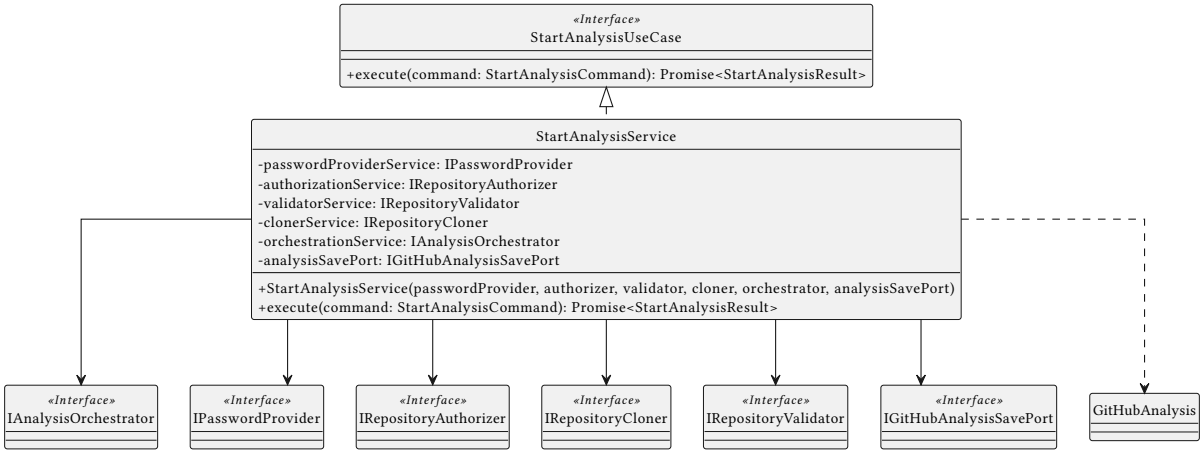

Figure 99: Class Diagram — StartAnalysisService

`StartAnalysisService` è l'Application Service principale: implementa `StartAnalysisUseCase` orchestrando l'intero flusso di avvio analisi — validazione, autorizzazione, clonazione, persistenza e dispatching agli agenti.

Svolge i seguenti passaggi per permettere a `AnalysisOrchestratorService` di eseguire l'analisi in modo asincrono dopo aver risposto al client:

1. Genera l'hash della password ricevuta (se presente) tramite `IPasswordProvider` per l'autorizzazione alla clonazione.
2. Recupera un PAT valido per il repository tramite `IRepositoryAuthorizer`, che gestisce la distinzione tra repository pubblici e privati.
3. Valida la raggiungibilità e la validità del repository tramite `IRepositoryValidator`, risolvendo branch e commit a valori concreti.
4. Clona il repository tramite `IRepositoryCloner`.

5. Crea una nuova entità `GitHubAnalysis` con i metadati identificativi e la persistenza tramite `IGitHubAnalysisSavePort`.
6. Avvia l'orchestrazione degli agenti in modo asincrono tramite `IAnalysisOrchestrator`, e i riferimenti all'analisi appena creata per l'associazione dei risultati e l'Id per recuperare la repository clonata.
7. Restituisce al client un `StartAnalysisResult` con i metadati dell'analisi avviata, permettendo al client di tracciare l'analisi in corso.

3.2.1.5.4.3.8 UpdatePatService

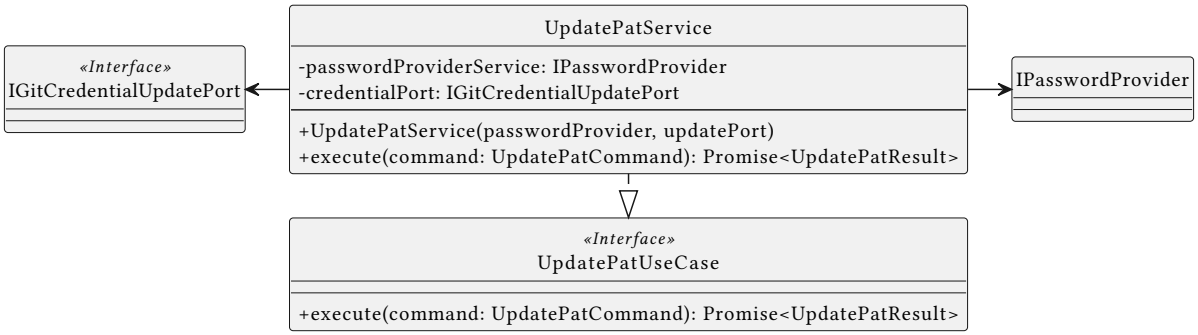

Figure 100: Class Diagram — UpdatePatService

`UpdatePatService` implementa `UpdatePatUseCase`, validando il nuovo PAT e la password corrente tramite il domain service `IPasswordProvider`, e delegando l'aggiornamento a `IGitCredentialUpdatePort`.

3.2.1.5.5 Port

Le porte sono le interfacce che definiscono i contratti di comunicazione tra il dominio applicativo e le dipendenze esterne (infrastruttura, agenti, persistenza). Ogni porta rappresenta un punto di estensione che permette di sostituire o modificare l'implementazione concreta senza impattare la logica applicativa, facilitando testabilità, manutenibilità e evoluzione del sistema. Ogni porta è progettata per essere il più possibile specifica e orientata al caso d'uso, evitando di esporre operazioni generiche o non necessarie che potrebbero portare a dipendenze indesiderate o a un accoppiamento eccessivo tra layer.

3.2.1.5.5.1 ICodeAgentPort

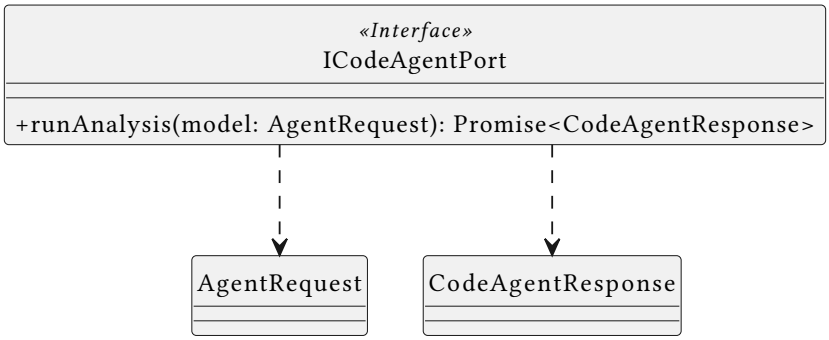

Figure 101: Class Diagram — ICodeAgentPort

`ICodeAgentPort` è la Porta che definisce il contratto per l'invocazione dell'agente di analisi del codice, accettando un `AgentRequest` contenente un `AnalysisId` e restituendo una `CodeAgentResponse` con i risultati dell'analisi tramite dei DTO.

3.2.1.5.5.2 ICollectionAdderPort

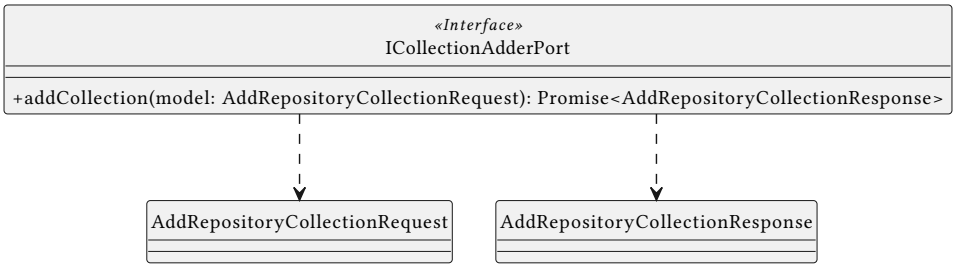

Figure 102: Class Diagram — ICollectionAdderPort

`ICollectionAdderPort` è la Porta che gestisce la persistenza di una nuova collezione di repository. Accetta un `AddRepositoryCollectionRequest` e restituisce una `AddRepositoryCollectionResponse` che indica l'esito dell'operazione.

3.2.1.5.5.3 ICodeReportSavePort

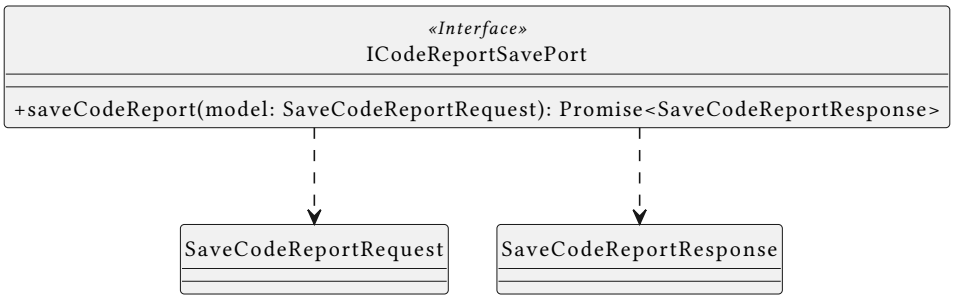

Figure 103: Class Diagram — ICodeReportSavePort

`ICodeReportSavePort` è la Porta per la persistenza di un `CodeAgentReport` prodotto dall'analisi del codice.

3.2.1.5.5.4 IDocumentationAgentPort

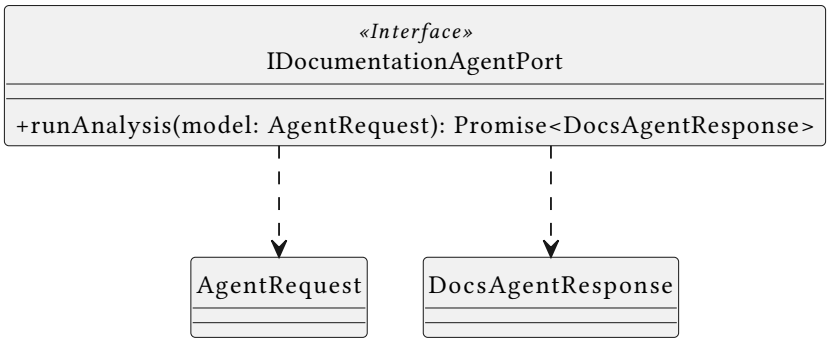

Figure 104: Class Diagram — IDocumentationAgentPort

`IDocumentationAgentPort` è la Porta che definisce il contratto per l'invocazione dell'agente di analisi della documentazione, accettando un `AgentRequest` contenente un `AnalysisId` e restituendo una `DocsAgentResponse` contenente i risultati dell'analisi tramite dei DTO.

3.2.1.5.5.5 ICollectionDuplicateCheckerPort

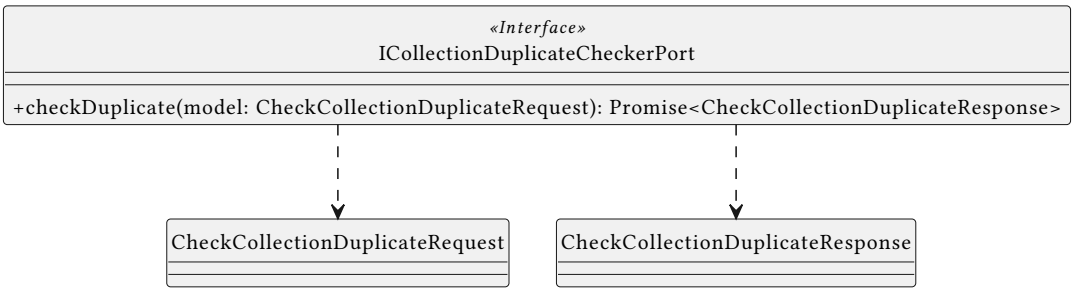

Figure 105: Class Diagram — ICollectionDuplicateCheckerPort

`ICollectionDuplicateCheckerPort` è la porta per la verifica dell'esistenza di una collezione di repository nella persistenza, utilizzato da `GitHubCollectionChecker` per implementare il controllo duplicati.

3.2.1.5.5.6 IDeleteRepositoryCollectionPort

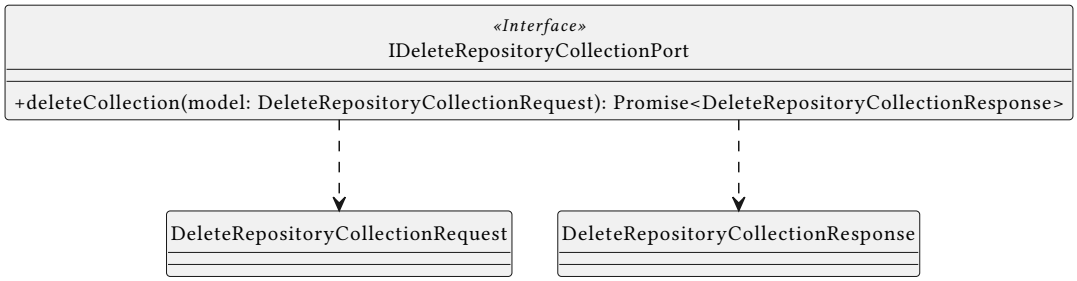

Figure 106: Class Diagram — IDeleteRepositoryCollectionPort

`IDeleteRepositoryCollectionPort` è la porta per l'eliminazione di una collezione di repository dalla persistenza, accettando un `DeleteRepositoryCollectionRequest` e restituendo un `DeleteRepositoryCollectionResponse` che indica l'esito dell'operazione.

3.2.1.5.5.7 IDocsReportSavePort

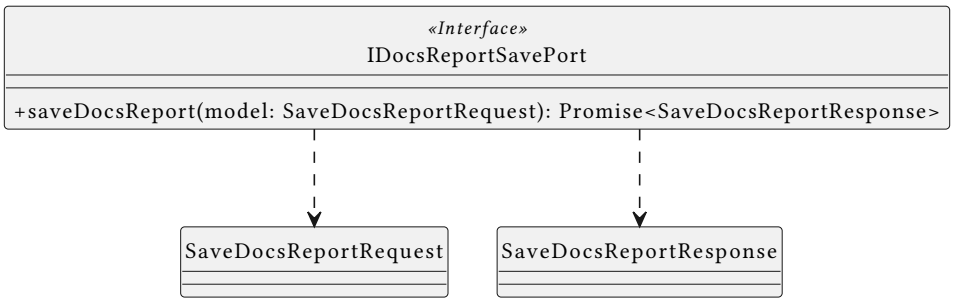

Figure 107: Class Diagram — IDocsReportSavePort

`IDocsReportSavePort` è la porta per la persistenza di un `DocumentationReport` prodotto dall'analisi della documentazione. Accenttando un `DocumentationReport` come input, permette di disaccoppiare la logica di persistenza dei report dal formato specifico dei dati restituiti dagli agenti, facilitando l'evoluzione indipendente di entrambi.

3.2.1.5.5.8 IGetAllAnalysesForUserPort

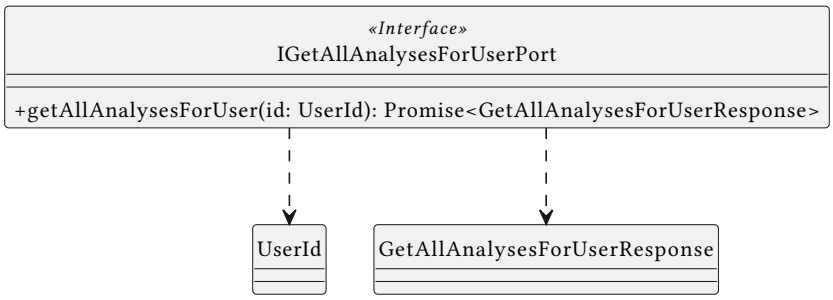

Figure 108: Class Diagram — IGetAllAnalysesForUserPort

`IGetAllAnalysesForUserPort` è la porta per il recupero di tutte le analisi associate a un utente dalla persistenza. Accetta un semplice value object `UserId` e restituisce una collezione di GitHubAnalysisGeneralDataDTO che aggregano i metadati identificativi di ciascuna analisi senza includere i report dettagliati.

3.2.1.5.5.9 IGetAllRepositoryCollectionsPort

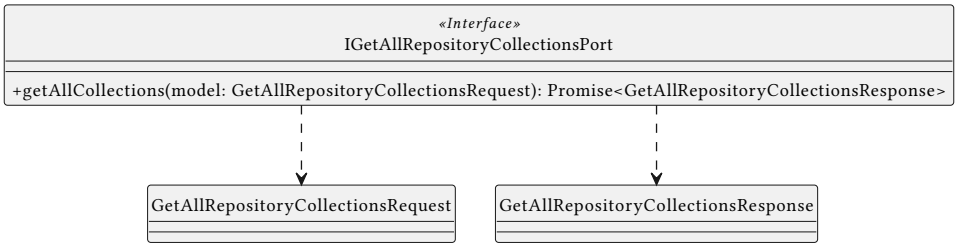

Figure 109: Class Diagram — IGetAllRepositoryCollectionsPort

`IGetAllRepositoryCollectionsPort` è la porta per il recupero di tutte le collezioni di repository di un utente dalla persistenza.

3.2.1.5.5.10 IGetAnalysisFromIdPort

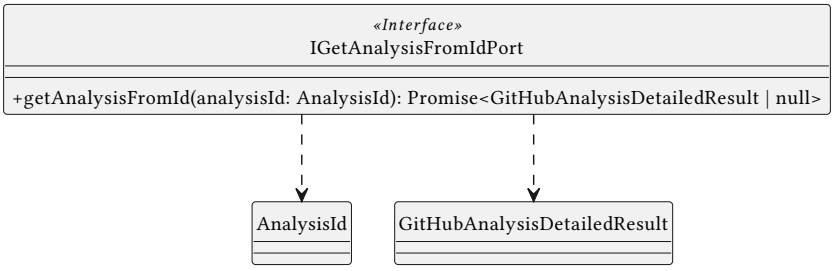

Figure 110: Class Diagram — IGetAnalysisFromIdPort

`IGetAnalysisFromIdPort` è la porta per il recupero di una singola analisi per identificatore dalla persistenza, restituendo `null` se non trovata.

3.2.1.5.5.11 IGetRepositoryCollectionPort

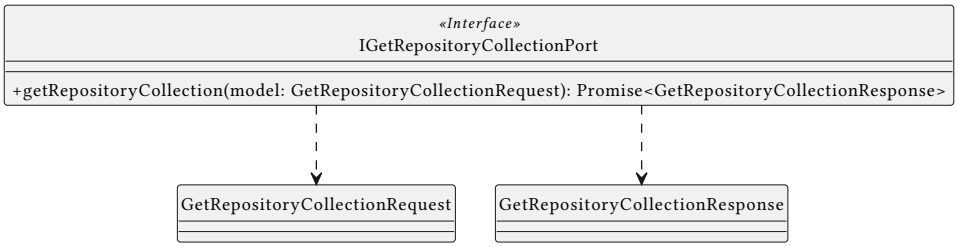

Figure 111: Class Diagram — IGetRepositoryCollectionPort

`IGetRepositoryCollectionPort` è la porta per il recupero di una specifica collezione di repository dalla persistenza tramite URL e utente.

3.2.1.5.5.12 IGitHubAnalysisSavePort

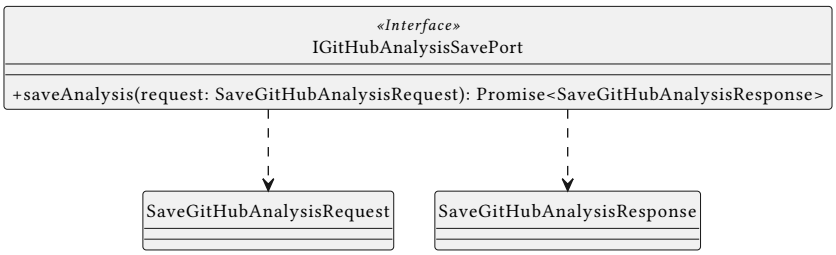

Figure 112: Class Diagram — IGitHubAnalysisSavePort

`IGitHubAnalysisSavePort` è la porta per la persistenza di una nuova entità `GitHubAnalysis` al momento dell'avvio dell'analisi.

3.2.1.5.5.13 IGitClonePort

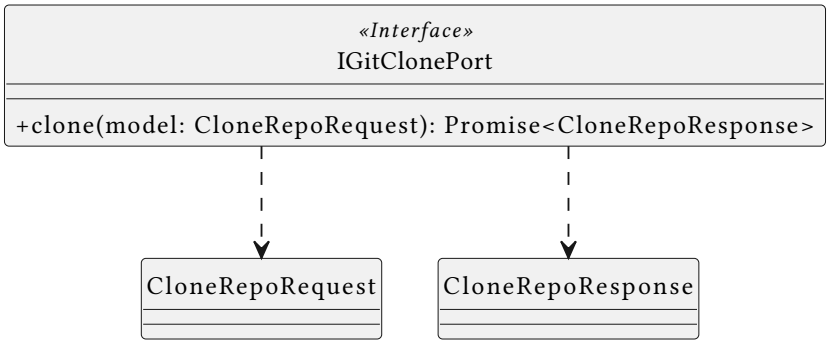

Figure 113: Class Diagram — IGitClonePort

`IGitClonePort` è la porta per l'operazione di clonazione Git, accettando un `CloneRepoRequest` e restituendo un `CloneRepoResponse`.

3.2.1.5.5.14 IGitCredentialDeletePort

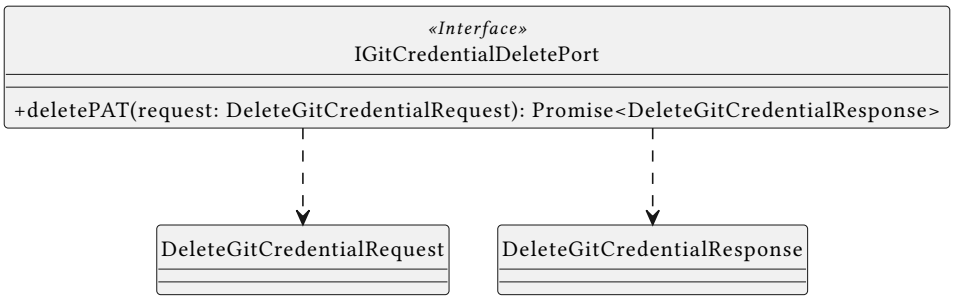

Figure 114: Class Diagram — IGitCredentialDeletePort

`IGitCredentialDeletePort` è la porta per l'eliminazione di credenziali Git.

3.2.1.5.5.15 IGitCredentialReadPort

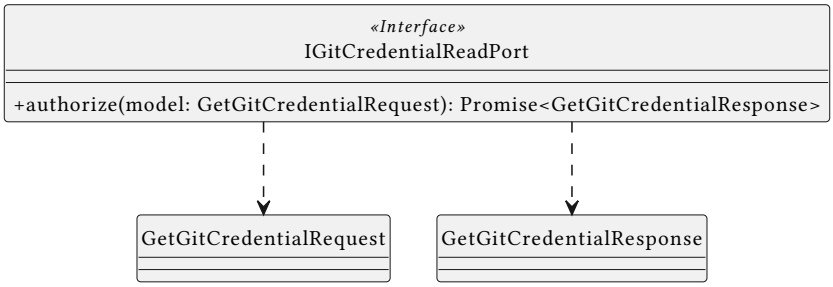

Figure 115: Class Diagram — IGitCredentialReadPort

`IGitCredentialReadPort` è la porta per la lettura/autorizzazione delle credenziali Git dal repository di persistenza.

3.2.1.5.5.16 IGitCredentialSavePort

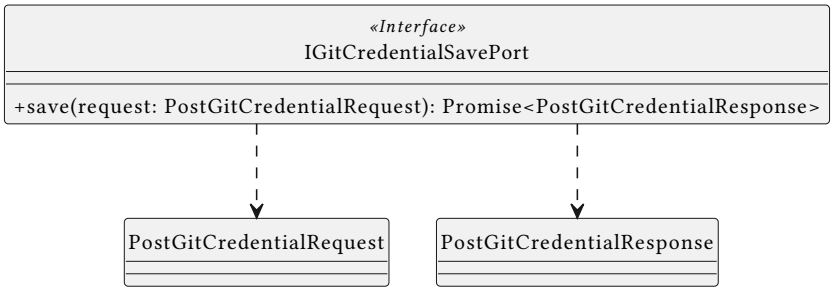

Figure 116: Class Diagram — IGitCredentialSavePort

`IGitCredentialSavePort` è la porta per il salvataggio di nuove credenziali Git.

3.2.1.5.5.17 IGitCredentialUpdatePort

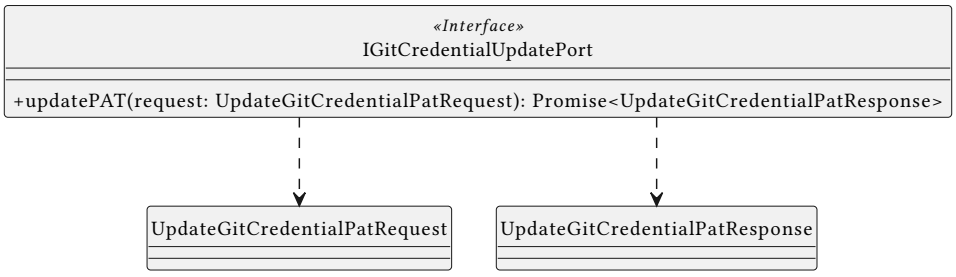

Figure 117: Class Diagram — IGitCredentialUpdatePort

`IGitCredentialUpdatePort` è la porta per l'aggiornamento del PAT di credenziali esistenti.

3.2.1.5.5.18 IGitHubAvailabilityPort

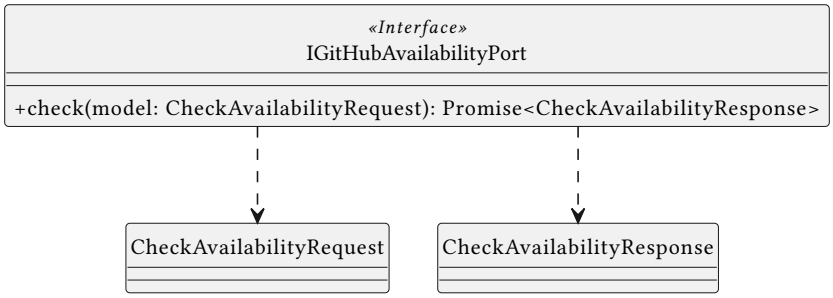

Figure 118: Class Diagram — IGitHubAvailabilityPort

`IGitHubAvailabilityPort` è la ports che definisce il contratto per verificare la raggiungibilità e i metadati di un repository GitHub, accettando un `CheckAvailabilityRequest` e restituendo un `CheckAvailabilityResponse`.

3.2.1.5.5.19 ISecurityAgentPort

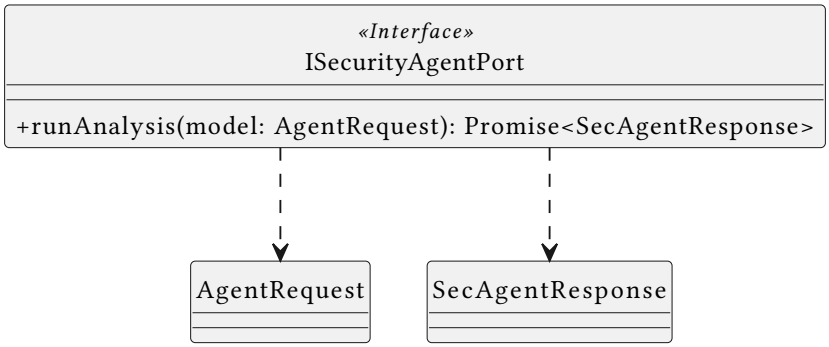

Figure 119: Class Diagram — ISecurityAgentPort

`ISecurityAgentPort` è la ports che definisce il contratto per l'invocazione dell'agente di analisi di sicurezza, accettando un `AgentRequest` e restituendo una `SecAgentResponse`.

3.2.1.5.5.20 ISecurityReportSavePort

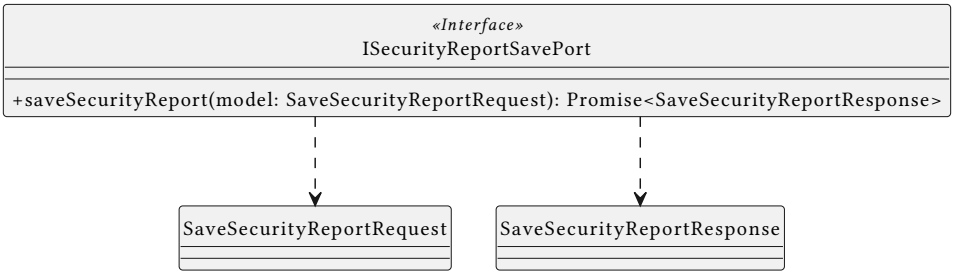

Figure 120: Class Diagram — ISecurityReportSavePort

`ISecurityReportSavePort` è la porta per la persistenza di un `SecurityReport` prodotto dall'analisi di sicurezza.

3.2.1.5.5.21 IUpdateAnalysisPort

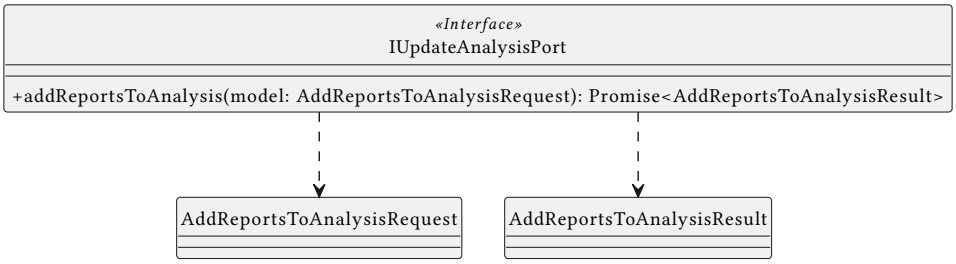

Figure 121: Class Diagram — IUpdateAnalysisPort

`IUpdateAnalysisPort` è la porta per l'aggiornamento di un'analisi esistente con i riferimenti ai report prodotti al termine dell'orchestrazione degli agenti.

3.2.1.5.6 Request

I contratti di richiesta sono i DTO che trasportano i dati necessari dalle classi del dominio applicativo (application service, domain service) verso le implementazioni concrete delle porte, hanno un formato specifico richiesto da ciascuna porta. Ogni request è progettata per essere il più possibile specifica e orientata al caso d'uso, evitando di esporre dati non necessari o di permettere l'invio di informazioni incomplete o incoerenti.

3.2.1.5.6.1 CheckAvailabilityRequest

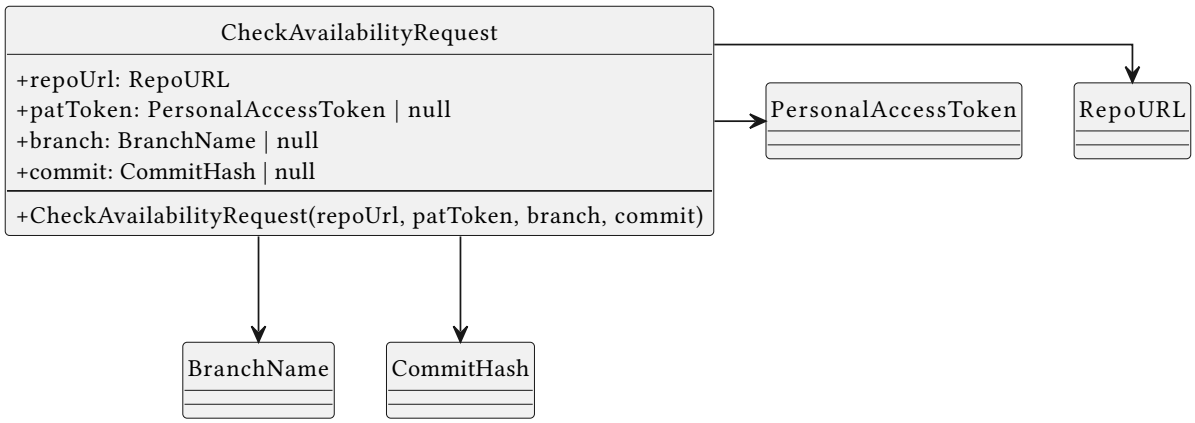

Figure 122: Class Diagram — CheckAvailabilityRequest

`CheckAvailabilityRequest` è il DTO di richiesta per la verifica della raggiungibilità di un repository, aggregando URL, token PAT opzionale, branch e commit opzionali.

- **Contratto del Port:** Definisce l'insieme minimo di informazioni che l'applicazione deve fornire all'infrastruttura per eseguire la verifica, disaccoppiando il dominio dai dettagli dell'API GitHub.

3.2.1.5.6.2 CloneRepoRequest

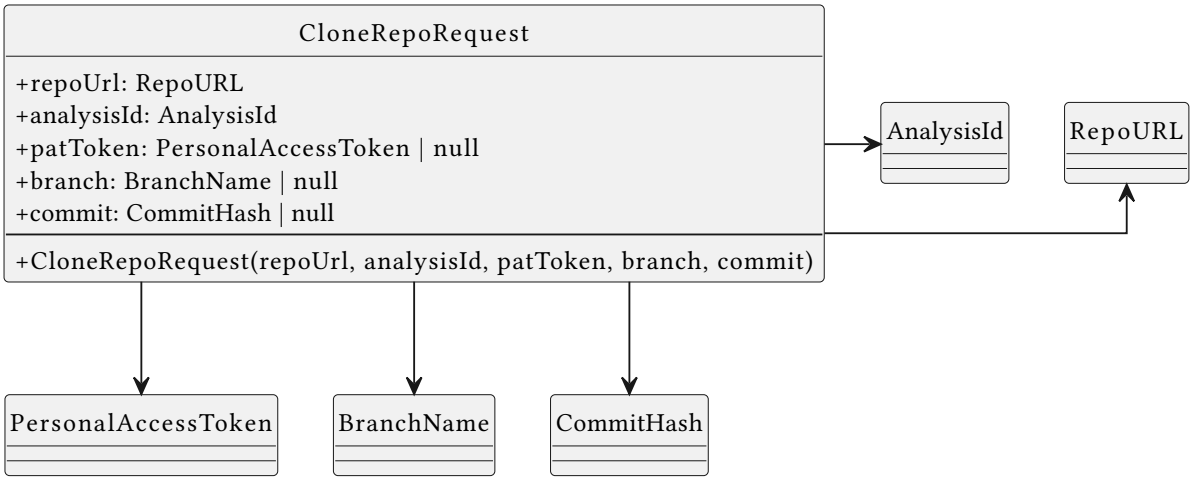

Figure 123: Class Diagram — CloneRepoRequest

`CloneRepoRequest` è il DTO di richiesta per la clonazione di un repository, trasportando tutti i parametri necessari all'operazione Git: URL, ID analisi (per nominare la cartella), PAT, branch e commit.

3.2.1.5.6.3 GetGitCredentialRequest

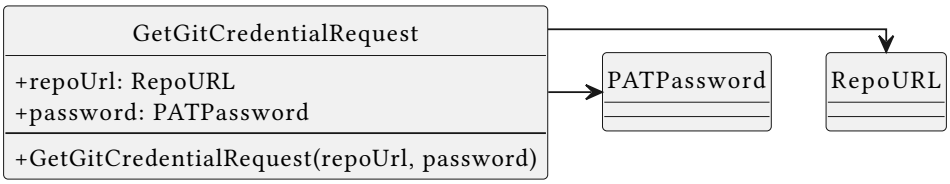

Figure 124: Class Diagram — GetGitCredentialRequest

`GetGitCredentialRequest` trasporta URL e hash della password per la lettura delle credenziali dal database.

3.2.1.5.6.4 PostGitCredentialRequest

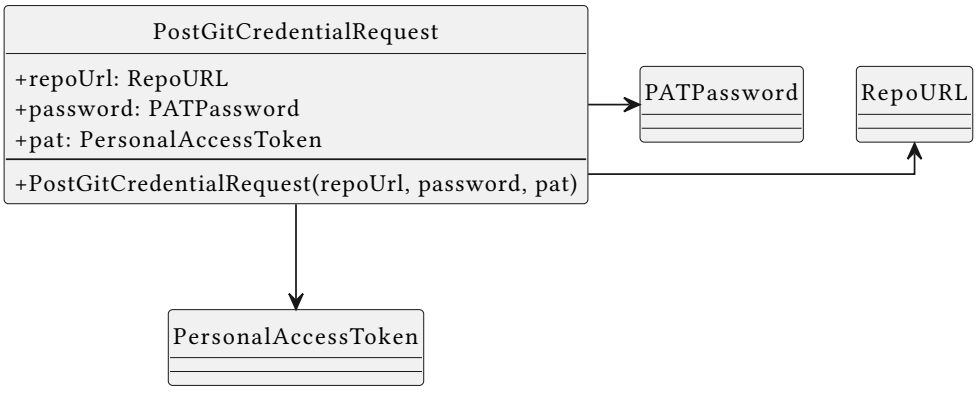

Figure 125: Class Diagram — PostGitCredentialRequest

`PostGitCredentialRequest` trasporta URL, hash della password e PAT per il salvataggio di nuove credenziali.

3.2.1.5.6.5 DeleteGitCredentialRequest

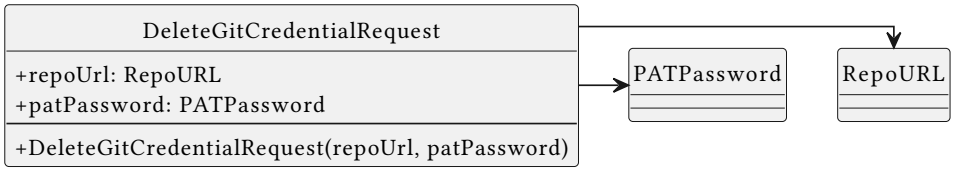

Figure 126: Class Diagram — DeleteGitCredentialRequest

`DeleteGitCredentialRequest` trasporta URL e hash della password per l'eliminazione delle credenziali.

3.2.1.5.6.6 UpdateGitCredentialPatRequest

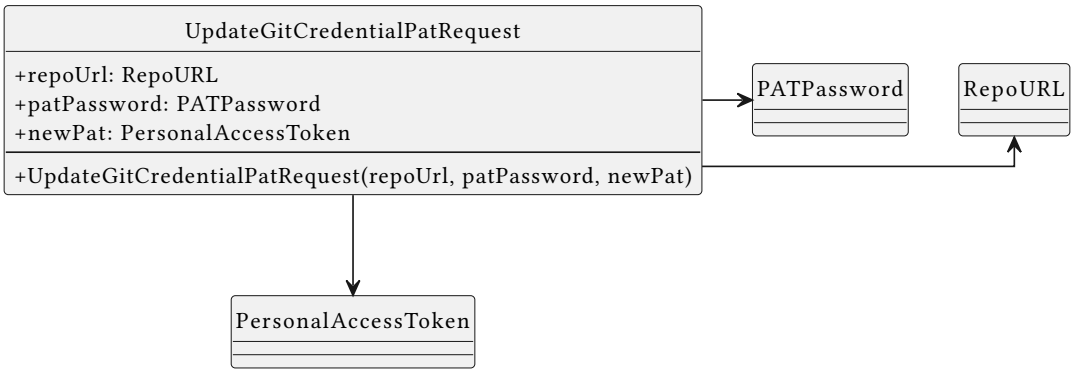

Figure 127: Class Diagram — UpdateGitCredentialPatRequest

`UpdateGitCredentialPatRequest` trasporta URL, hash della password e nuovo PAT per l'aggiornamento delle credenziali.

3.2.1.5.6.7 AddReportsToAnalysisRequest

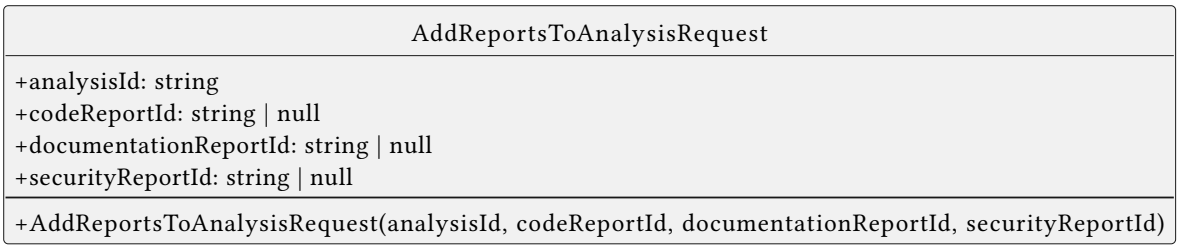

Figure 128: Class Diagram — AddReportsToAnalysisRequest

`AddReportsToAnalysisRequest` trasporta l'identificatore dell'analisi e i tre riferimenti opzionali ai report da associare, utilizzato da `IUpdateAnalysisPort` al termine dell'orchestrazione.

3.2.1.5.6.8 AddRepositoryCollectionRequest

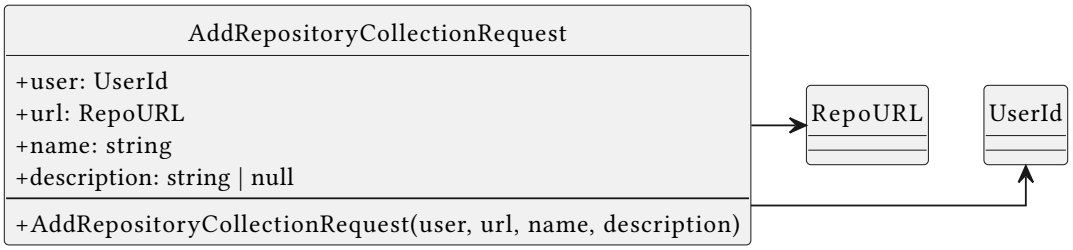

Figure 129: Class Diagram — AddRepositoryCollectionRequest

`AddRepositoryCollectionRequest` trasporta i dati necessari alla creazione di una nuova collezione: utente, URL del repository, nome e descrizione opzionale, utilizzato da `ICollectionAdderPort`.

3.2.1.5.6.9 AgentRequest

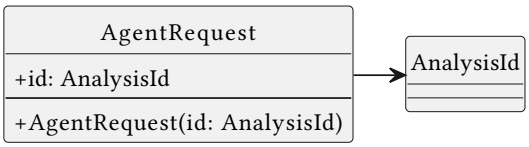

Figure 130: Class Diagram — AgentRequest

`AgentRequest` è il DTO di richiesta condiviso dai tre port degli agenti, trasportando esclusivamente l'`AnalysisId` che identifica l'analisi da eseguire.

- **Contratto Uniforme:** La struttura minimale e condivisa tra `ICodeAgentPort`, `IDocumentationAgentPort` e `ISecurityAgentPort` permette di invocare qualsiasi agente con la stessa interfaccia.

3.2.1.5.6.10 CheckCollectionDuplicateRequest

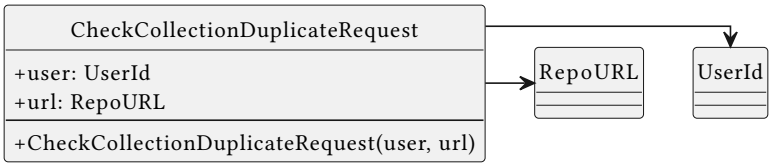

Figure 131: Class Diagram — CheckCollectionDuplicateRequest

`CheckCollectionDuplicateRequest` trasporta utente e URL del repository per la verifica di duplicati, utilizzato da `ICollectionDuplicateCheckerPort`.

3.2.1.5.6.11 DeleteRepositoryCollectionRequest

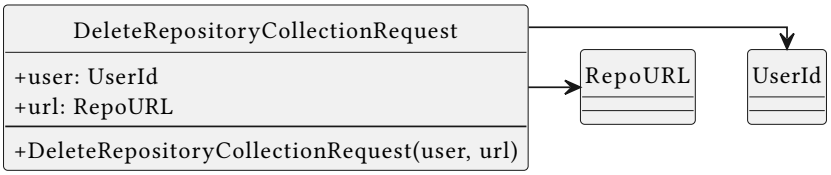

Figure 132: Class Diagram — DeleteRepositoryCollectionRequest

`DeleteRepositoryCollectionRequest` trasporta utente e URL del repository per identificare la collezione da eliminare, utilizzato da `IDeleteRepositoryCollectionPort`.

3.2.1.5.6.12 GetAllRepositoryCollectionsRequest

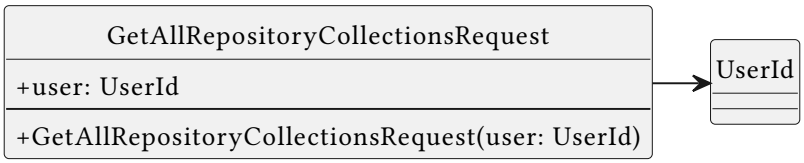

Figure 133: Class Diagram — GetAllRepositoryCollectionsRequest

`GetAllRepositoryCollectionsRequest` trasporta esclusivamente l'identificatore utente per il recupero di tutte le sue collezioni, utilizzato da `IGetAllRepositoryCollectionsPort`.

3.2.1.5.6.13 GetRepositoryCollectionRequest

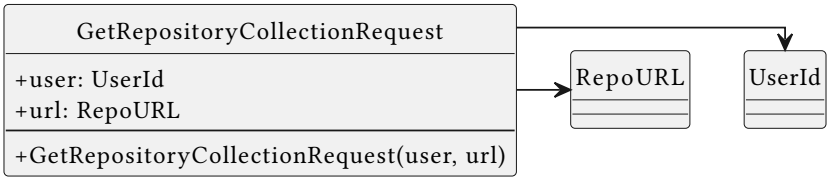

Figure 134: Class Diagram — GetRepositoryCollectionRequest

`GetRepositoryCollectionRequest` trasporta utente e URL del repository per il recupero di una specifica collezione, utilizzato da `IGetRepositoryCollectionPort`.

3.2.1.5.6.14 SaveCodeReportRequest

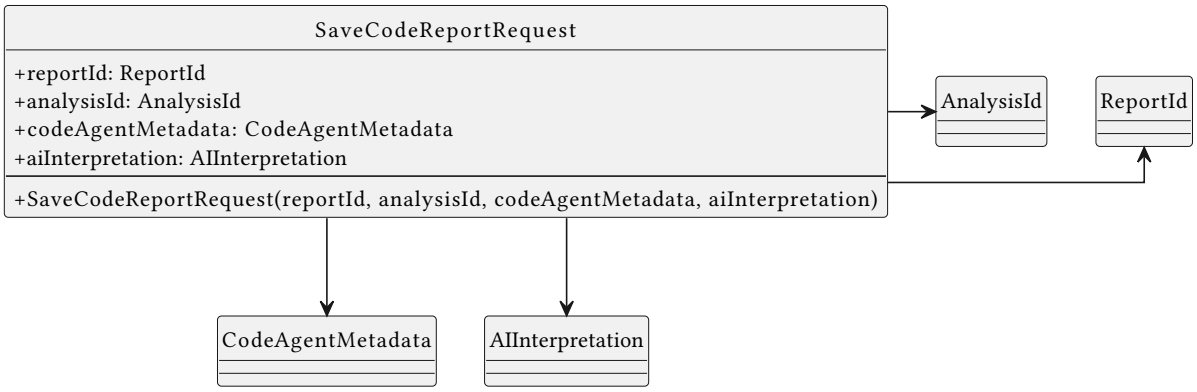

Figure 135: Class Diagram — SaveCodeReportRequest

`SaveCodeReportRequest` trasporta i dati dell'entità `CodeAgentReport` verso la persistenza, aggregando identificatori e i Value Object `CodeAgentMetadata` e `AIInterpretation`.

3.2.1.5.6.15 SaveDocsReportRequest

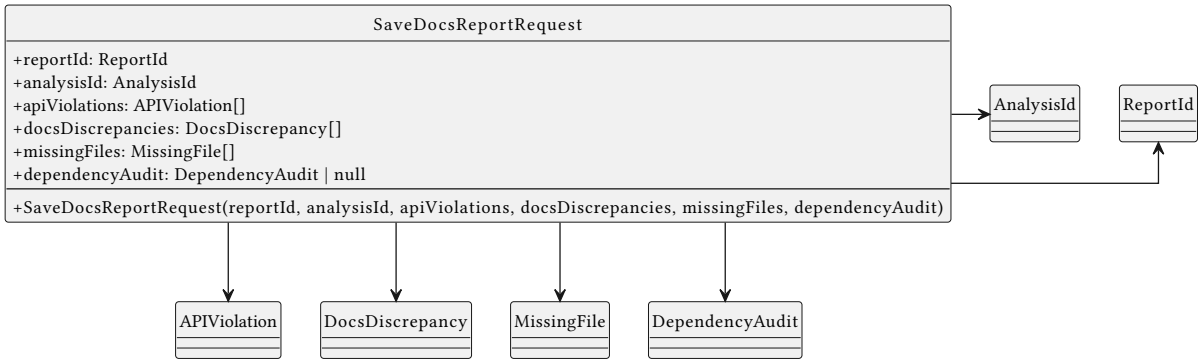

Figure 136: Class Diagram — SaveDocsReportRequest

`SaveDocsReportRequest` trasporta i dati dell'entità `DocumentationReport` verso la persistenza, aggregando identificatori e le collezioni di Value Object prodotti dall'analisi della documentazione.

3.2.1.5.6.16 SaveGitHubAnalysisRequest

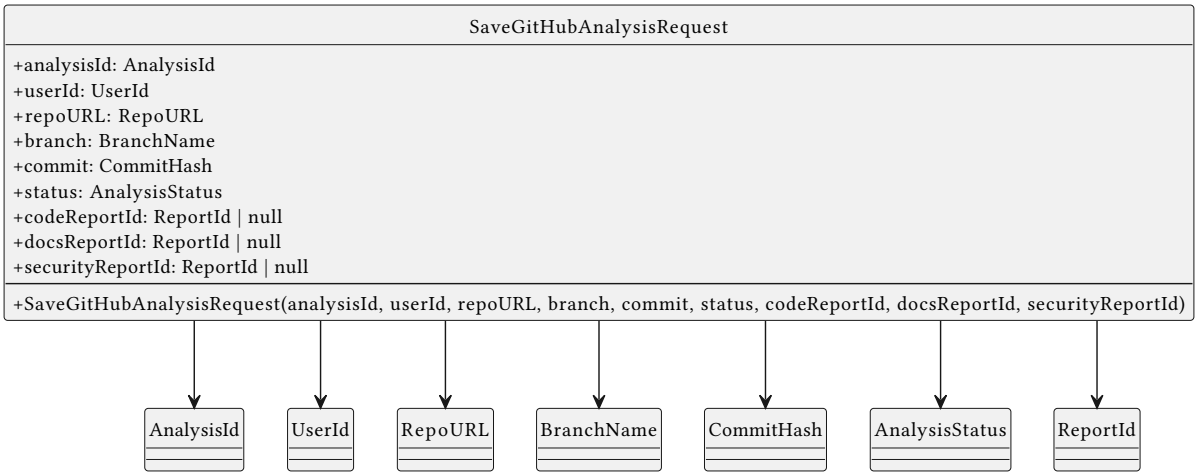

Figure 137: Class Diagram — SaveGitHubAnalysisRequest

`SaveGitHubAnalysisRequest` trasporta i dati dell'entità `GitHubAnalysis` verso la persistenza al momento della sua creazione, includendo tutti i Value Object identificativi e i riferimenti opzionali ai tre report.

3.2.1.5.6.17 SaveSecurityReportRequest

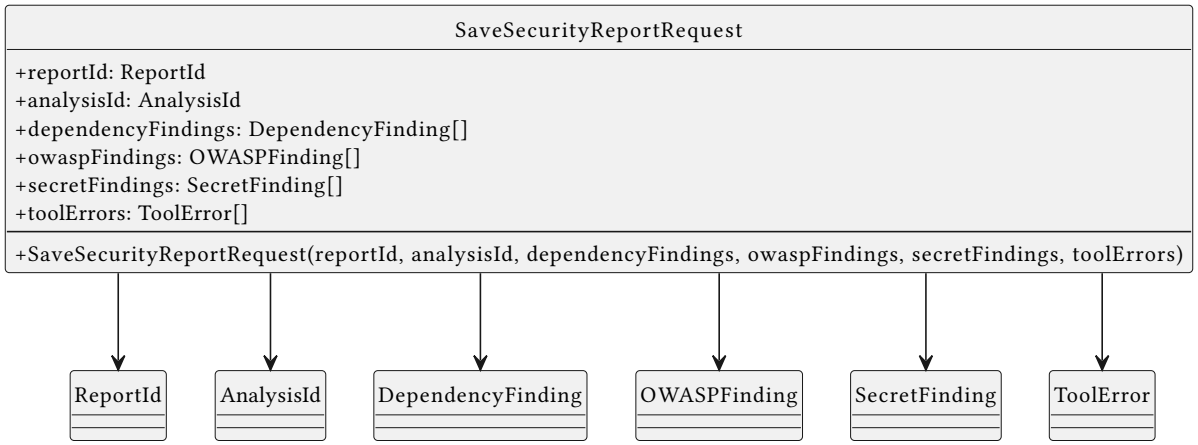

Figure 138: Class Diagram — SaveSecurityReportRequest

`SaveSecurityReportRequest` trasporta i dati dell'entità `SecurityReport` verso la persistenza, aggregando identificatori e le collezioni di finding suddivisi per categoria di sicurezza.

3.2.1.5.7 Response

I contratti di risposta sono i DTO che trasportano i dati restituiti dalle implementazioni concrete delle porte verso le classi del dominio applicativo (application service, domain service), hanno un formato specifico restituito da ciascuna porta. Ogni response è progettata per essere il più possibile specifica e orientata al caso d'uso, evitando di esporre dati non necessari o di permettere l'invio di informazioni incomplete o incoerenti.

3.2.1.5.7.1 CheckAvailabilityResponse

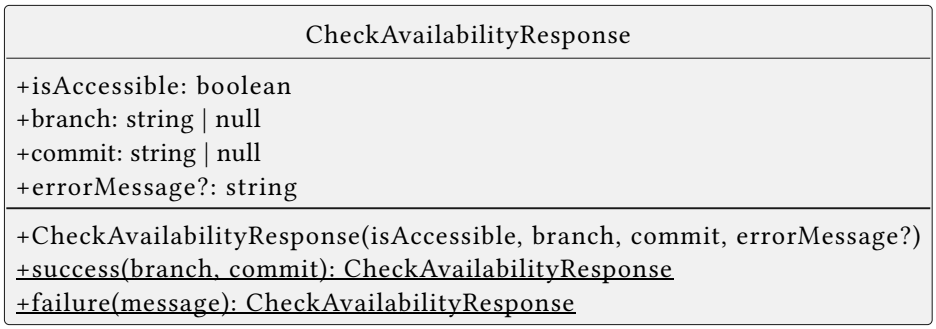

Figure 139: Class Diagram — CheckAvailabilityResponse

`CheckAvailabilityResponse` trasporta il risultato della verifica di raggiungibilità: flag di accessibilità, branch e commit risolti, eventuale messaggio di errore.

3.2.1.5.7.2 CloneRepoResponse

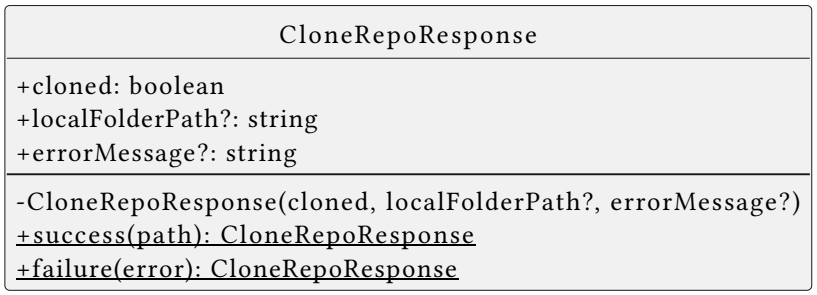

Figure 140: Class Diagram — CloneRepoResponse

`CloneRepoResponse` trasporta l'esito della clonazione: flag di successo, percorso locale della cartella clonata, eventuale messaggio di errore.

3.2.1.5.7.3 GetGitCredentialResponse

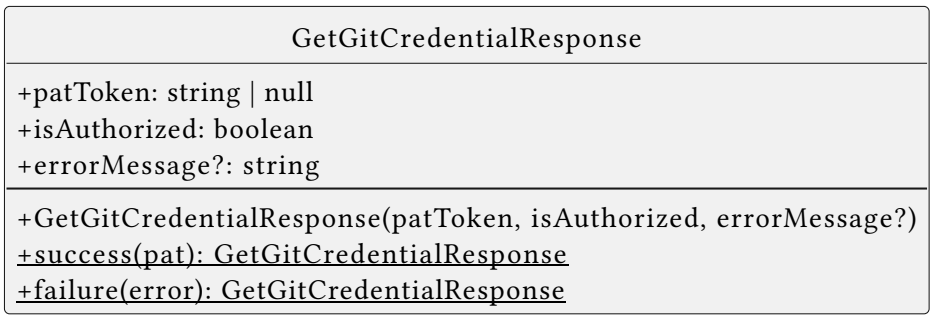

Figure 141: Class Diagram — GetGitCredentialResponse

`GetGitCredentialResponse` trasporta il PAT recuperato (o null in caso di errore) e il flag di autorizzazione.

3.2.1.5.7.4 PostGitCredentialResponse

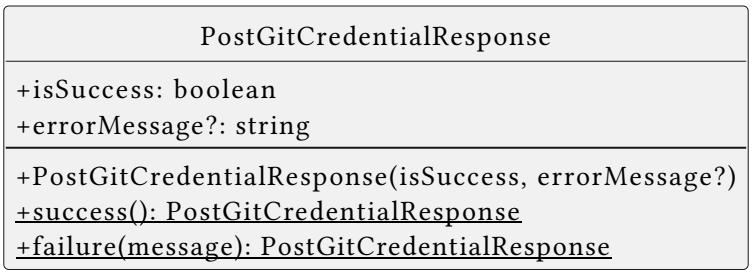

Figure 142: Class Diagram — PostGitCredentialResponse

`PostGitCredentialResponse` indica l'esito del salvataggio delle credenziali.

3.2.1.5.7.5 DeleteGitCredentialResponse

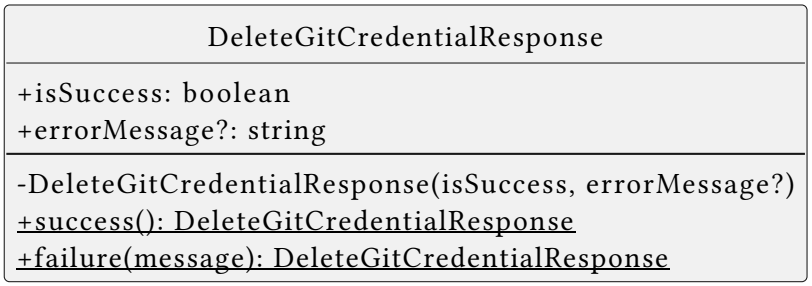

Figure 143: Class Diagram — DeleteGitCredentialResponse

`DeleteGitCredentialResponse` indica l'esito dell'eliminazione delle credenziali.

3.2.1.5.7.6 UpdateGitCredentialPatResponse

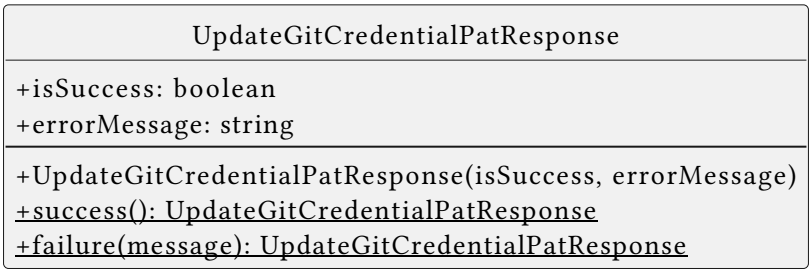

Figure 144: Class Diagram — UpdateGitCredentialPatResponse

`UpdateGitCredentialPatResponse` indica l'esito dell'aggiornamento del PAT.

3.2.1.5.7.7 AddReportsToAnalysisResult

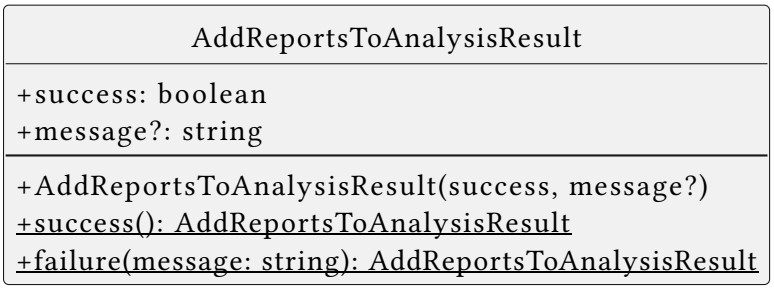

Figure 145: Class Diagram — AddReportsToAnalysisResult

`AddReportsToAnalysisResult` indica l'esito dell'aggiornamento dell'analisi con i riferimenti ai report prodotti dall'orchestrazione.

3.2.1.5.7.8 AddRepositoryCollectionResponse

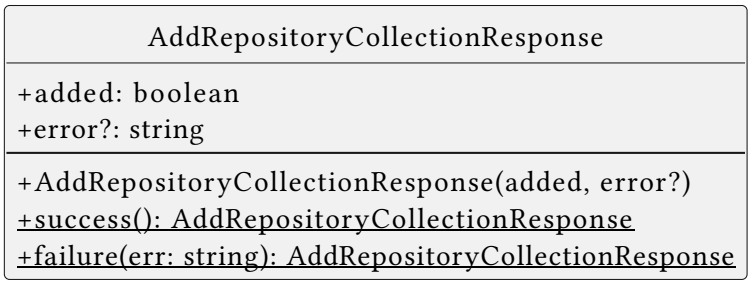

Figure 146: Class Diagram — AddRepositoryCollectionResponse

`AddRepositoryCollectionResponse` indica l'esito della persistenza di una nuova collezione di repository.

3.2.1.5.7.9 CheckCollectionDuplicateResponse

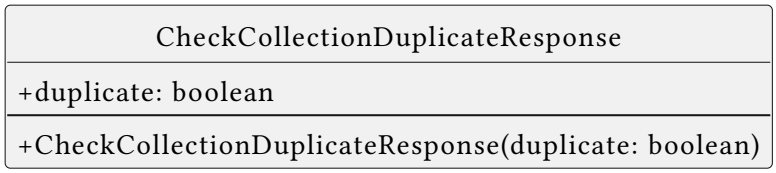

Figure 147: Class Diagram — CheckCollectionDuplicateResponse

`CheckCollectionDuplicateResponse` trasporta esclusivamente il flag `duplicate`, indicando se esiste già una collezione per l'URL e l'utente forniti.

3.2.1.5.7.10 DeleteRepositoryCollectionResponse

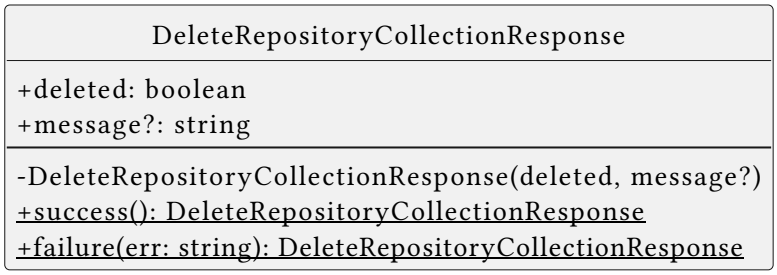

Figure 148: Class Diagram — DeleteRepositoryCollectionResponse

`DeleteRepositoryCollectionResponse` indica l'esito dell'eliminazione di una collezione di repository dalla persistenza.

3.2.1.5.7.11 GetAllAnalysesForUserResponse

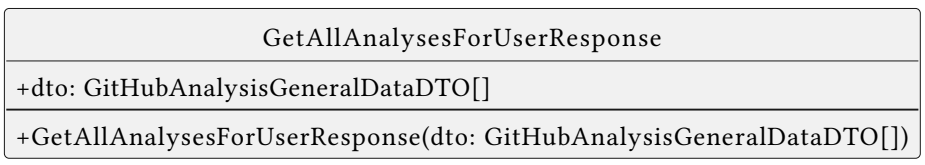

Figure 149: Class Diagram — GetAllAnalysesForUserResponse

`GetAllAnalysesForUserResponse` trasporta la collezione di `GitHubAnalysisGeneralDataDTO` restituita dalla persistenza per un dato utente.

3.2.1.5.7.12 GetAllRepositoryCollectionsResponse

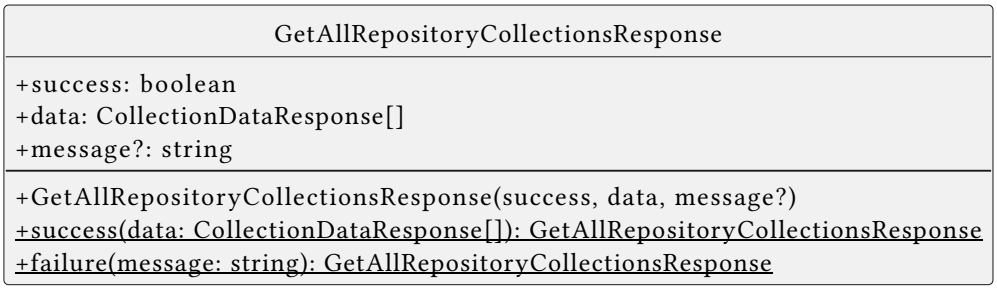

Figure 150: Class Diagram — GetAllRepositoryCollectionsResponse

`GetAllRepositoryCollectionsResponse` trasporta la collezione di `CollectionDataResponse` restituita dalla persistenza per un dato utente, con factory method `success()` e `failure()`.

3.2.1.5.7.13 GetRepositoryCollectionResponse

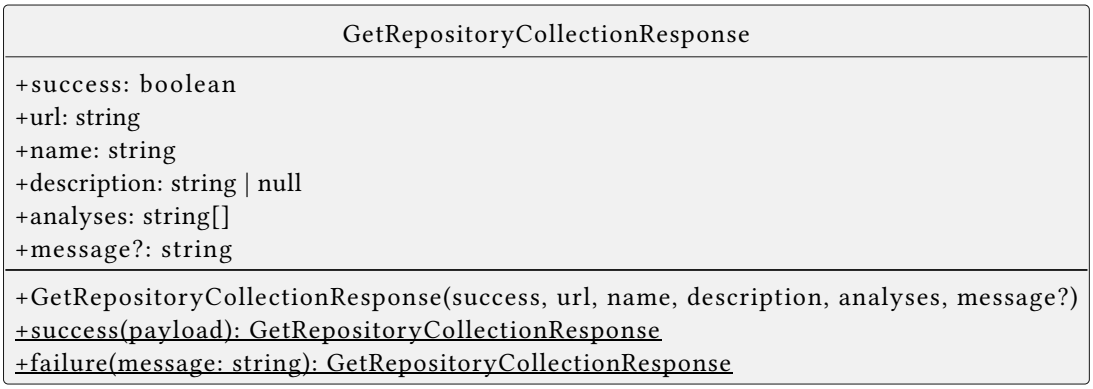

Figure 151: Class Diagram — GetRepositoryCollectionResponse

`GetRepositoryCollectionResponse` trasporta i dati di una specifica collezione di repository — URL, nome, descrizione e lista degli identificatori delle analisi associate — con factory method `success()` e `failure()`.

3.2.1.5.7.14 SaveCodeReportResponse

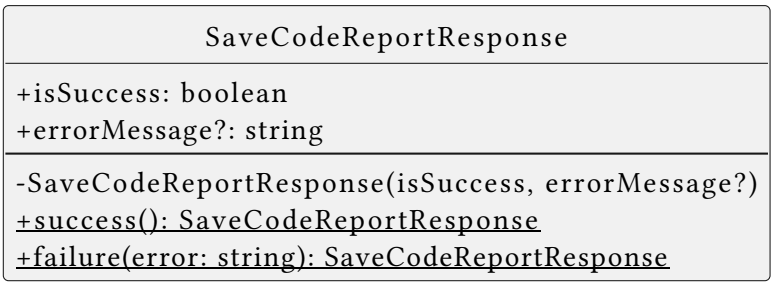

Figure 152: Class Diagram — SaveCodeReportResponse

`SaveCodeReportResponse` indica l'esito della persistenza di un report di analisi del codice.

3.2.1.5.7.15 SaveDocsReportResponse

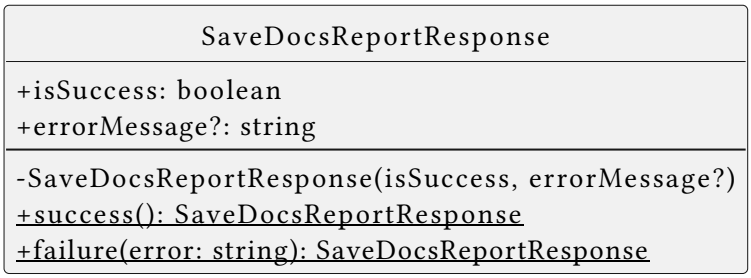

Figure 153: Class Diagram — SaveDocsReportResponse

`SaveDocsReportResponse` indica l'esito della persistenza di un report di analisi della documentazione.

3.2.1.5.7.16 SaveGitHubAnalysisResponse

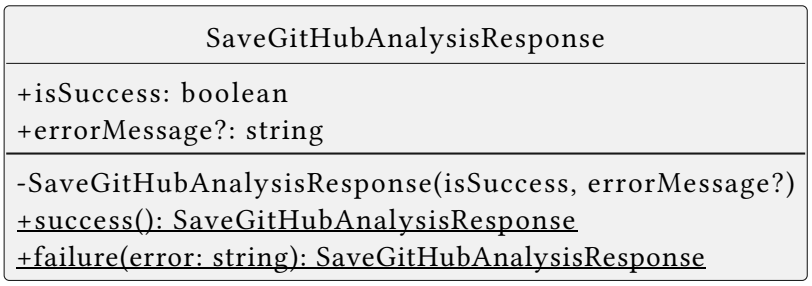

Figure 154: Class Diagram — SaveGitHubAnalysisResponse

`SaveGitHubAnalysisResponse` indica l'esito della persistenza di una nuova analisi GitHub.

3.2.1.5.7.17 SaveSecurityReportResponse

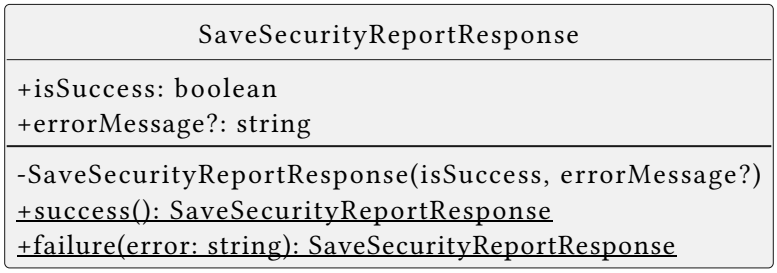

Figure 155: Class Diagram — SaveSecurityReportResponse

`SaveSecurityReportResponse` indica l'esito della persistenza di un report di analisi di sicurezza.

3.2.1.5.7.18 GitHubAnalysisDetailedResult

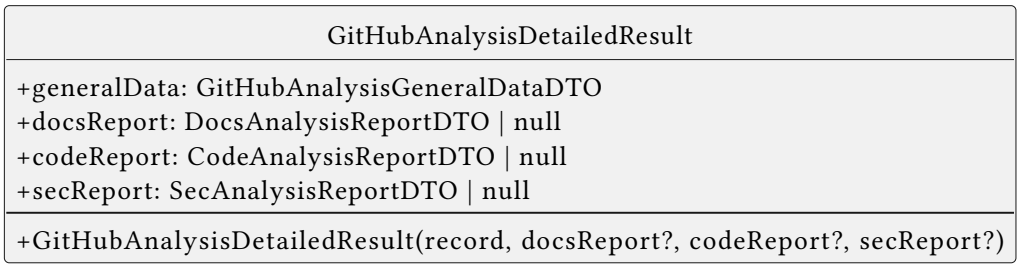

Figure 156: Class Diagram — GitHubAnalysisDetailedResult

`GitHubAnalysisDetailedResult` aggrega i dati generali di un'analisi con i tre report opzionali, utilizzato da `IGetAnalysisFromIdPort` per restituire al layer applicativo una visione completa dell'analisi recuperata.

- **Report Opzionali:** I campi `docsReport`, `codeReport` e `secReport` sono nullable, riflettendo il fatto che un'analisi può coinvolgere solo un sottoinsieme dei tre tipi di report in base a quanto richiesto.

3.2.1.6 Infrastructure

3.2.1.6.1 Adapters

Questa sezione descrive i Driven Adapter, i componenti concreti del livello infrastrutturale incaricati di implementare i contratti (Port) definiti nel livello Application. Nel rigoroso rispetto dell'Architettura Esagonale, gli adapter agiscono come strato di confine e di traduzione tra il nucleo applicativo e l'infrastruttura esterna, isolando la logica di business da qualsiasi dettaglio tecnologico. Essi incapsulano tutta la complessità necessaria per interagire con il database (MongoDB), le API esterne (GitHub), i processi di sistema (esecuzioni Docker locali) e l'infrastruttura Cloud (AWS ECS e S3). Grazie a questo isolamento, la logica di business e l'orchestrazione dei flussi rimangono puramente agnostiche e protette dai dettagli di I/O, garantendo un'altissima testabilità e flessibilità architetturale.

3.2.1.6.1.1 GitHubAdapter

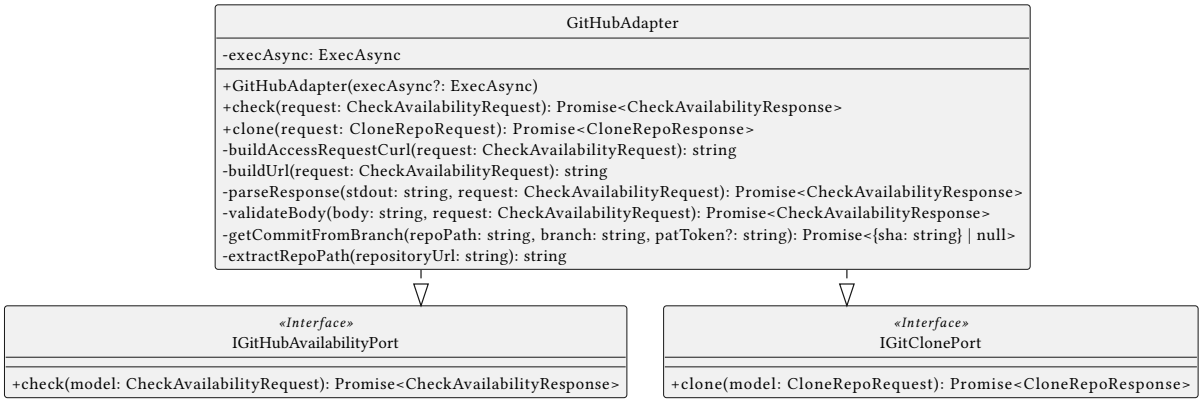

Figure 157: Class Diagram — GitHubAdapter

`GitHubAdapter` è il Driven Adapter responsabile dell'interazione con l'ecosistema GitHub. Implementando i port `IGitHubAvailabilityPort` e `IGitClonePort`, funge da ponte traduttore: prende le richieste del dominio applicativo e le trasforma nei comandi tecnici necessari per comunicare con l'esterno, come chiamate di rete (`curl`) e comandi shell nativi (`git`).

- **Integrazione Lightweight tramite Shell:** Invece di dipendere da SDK esterni pesanti, l'adapter utilizza direttamente comandi shell di sistema. Il costruttore accetta una funzione `execAsync`

iniettabile (di default basata su `child_process.exec`), permettendo un mocking completo durante i test unitari senza dover effettuare reali chiamate di rete.
- **Risoluzione Dinamica e Validazione (`check`):** Il metodo `check` non si limita a verificare i permessi. Interrogando l'API REST di GitHub tramite `curl`, estrae e analizza lo status code HTTP. Se il repository è accessibile, processa il payload JSON per risolvere dinamicamente l'esatto hash SHA del commit (sia che l'utente abbia richiesto un branch specifico, un commit esatto, o si sia affidato al branch di default). Questo garantisce che le fasi successive dell'analisi siano assolutamente deterministiche. Nel caso in cui venga richiesto il branch di default, il metodo esegue una seconda chiamata HTTP tramite `getCommitFromBranch` per risolvere il commit SHA esatto, poiché la risposta iniziale sull'endpoint `/repos/{owner}/{repo}` non lo espone direttamente.
- **Clonazione Ottimizzata (`clone`):** La logica di clonazione applica strategie diverse per minimizzare l'uso di banda e disco. Se viene richiesto un branch specifico o il branch di default, esegue una clonazione "shallow" (`--depth 1`), scaricando solo l'ultima versione dei file ignorando lo storico dei commit passati; se è richiesto un commit storico specifico, esegue una clonazione standard seguita da un `checkout` mirato.
- **Gestione Sicura dell'Autenticazione:** L'adapter inietta in modo sicuro i `PersonalAccessToken` passandoli come header `Bearer` per le API o incorporandoli dinamicamente nell'URL HTTPS durante la clonazione. Inoltre, implementa un meccanismo di fallback a livello di sistema qualora l'utente non fornisca credenziali proprie.
- **Resilienza e Cleanup:** Per prevenire il rapido esaurimento dello spazio su disco del server (disk leak), l'adapter isola ogni clonazione in una cartella temporanea univoca in `/tmp/` basata sull'ID dell'analisi. Garantisce inoltre, tramite un blocco `catch`, che le directory temporanee vengano rimosse forzatamente in caso di fallimento del clone.

3.2.1.6.1.2 LocalCodeAnalysisAdapter

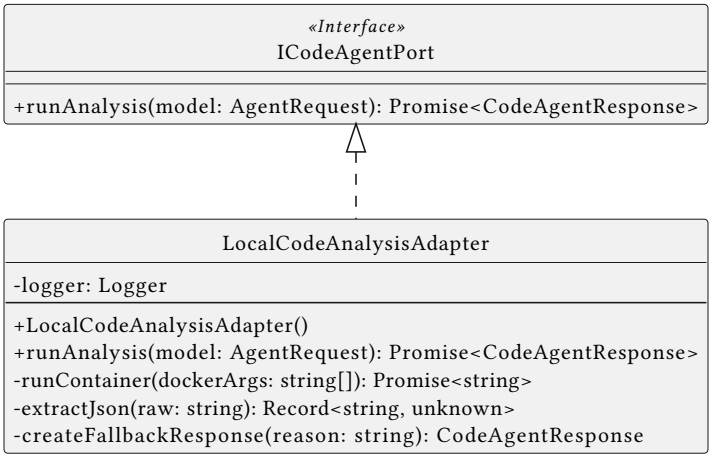

Figure 158: Class Diagram — LocalCodeAnalysisAdapter

`LocalCodeAnalysisAdapter` è il Driven Adapter che implementa il port `ICodeAgentPort`. È responsabile dell'orchestrazione locale dell'agente di analisi del codice, incapsulando l'esecuzione del container Docker e il recupero sicuro dei risultati.

- **Orchestrazione Docker Nativa:** Il metodo `runContainer()` utilizza il modulo `child_process.spawn` di Node.js per avviare il container `strands-code-analyzer`. Si occupa di montare dinamicamente i volumi condivisi (`analysis_tmp_data`) e di iniettare in modo sicuro le variabili d'ambiente necessarie (verificando la presenza del file `.env` tramite `fs.existsSync`) senza esporle nel codice.

- **Parsing Resiliente a Tolleranza d'Errore:** Lo `stdout` di un container Docker include spesso log di boot o warning estranei al risultato. Per questo, il metodo `extractJson()` esegue due passaggi: prima scansiona l'output alla ricerca di un token esplicito di errore (`{"status": "error"`); se non lo trova, applica un sofisticato algoritmo iterativo di bilanciamento delle parentesi per isolare il blocco JSON valido contenente il nodo root `analysis_report`. In aggiunta, il metodo `runContainer()` implementa una logica di tolleranza sull'exit code: se il container termina con un codice diverso da zero ma ha comunque prodotto output su `stdout`, il risultato viene comunque promosso invece di essere scartato, permettendo il recupero di report parziali da container che crashano dopo aver completato la scrittura.
- **Arricchimento del Payload:** Prima di restituire il risultato tramite il metodo `runAnalysis()`, l'adapter funge da strato di traduzione. Intercetta il JSON grezzo emesso dall'agente e vi inietta dinamicamente i metadati operativi cruciali (come l'identificativo del `repository` e lo `status` dell'operazione), garantendo che il DTO finale rispetti rigorosamente le aspettative del livello Application.
- **Risoluzione dei fallimenti (Fallback):** In caso di crash improvviso del container, fallimento del Docker o corruzione dell'output testuale, l'eccezione non viene propagata. Il blocco `catch` invoca `createFallbackResponse()`, che istanzia e restituisce una risposta strutturata, type-safe e con verdetto `Critical`, incapsulando il motivo del fallimento. Questo isolamento garantisce che l'orchestratore globale non si blocchi per colpa di un singolo agente.

3.2.1.6.1.3 DocumentationAnalysisAdapter

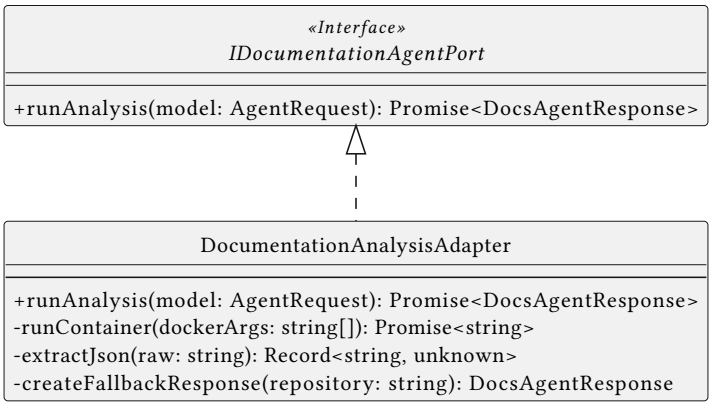

Figure 159: Class Diagram — DocumentationAnalysisAdapter

`DocumentationAnalysisAdapter` è il Driven Adapter che implementa il port `IDocumentationAgentPort`. Gestisce l'orchestrazione locale dell'agente incaricato di valutare la qualità, le discrepanze e i file mancanti della documentazione del repository.

- **Esecuzione Isolata via Docker:** Il metodo `runContainer()` utilizza il modulo `child_process.spawn` per avviare il container `strands-documentation-analyzer`. Si occupa di montare dinamicamente il volume condiviso (`analysis_tmp_data`) per l'accesso al codice e di iniettare il file di configurazione ambientale `.env`.
- **Overriding Dinamico dell'Entrypoint:** A differenza degli altri adapter, sovrascrive dinamicamente l'entrypoint di default del container (`--entrypoint sh`) per lanciare esplicitamente lo script Python dell'agente. In questa fase, applica un quoting rigoroso al path del repository (`"${repoPathInContainer}"`) per prevenire bug legati al word-splitting della shell (ad esempio se il nome della repo contiene spazi).
- **Parsing Resiliente a Tolleranza d'Errore:** Consapevole che lo `stdout` Docker non è mai un JSON "puro", l'adapter impiega il metodo custom `extractJson()`. Dapprima verifica l'eventuale presenza di un token esplicito di errore; in sua assenza, utilizza un algoritmo iterativo basato sul

conteggio delle parentesi per scansionare l'output, scartare il rumore di boot e isolare il blocco JSON valido contenente l'oggetto `analysis_report`.
- **Arricchimento del Payload:** Prima di restituire l'esito tramite `runAnalysis()`, l'adapter inietta nel JSON grezzo i metadati operativi mancanti (come l'ID del `repository` e lo `status` di successo), allineando strutturalmente l'output alle aspettative del livello Application.
- **Risoluzione dei fallimenti (Fallback):** Se l'agente Python va in crash o genera un output incomprensibile, il blocco `catch` invoca `createFallbackResponse()`. Questo metodo inietta uno stato di errore controllato (`status: 'error'`) e restituisce una risposta strutturata contenente array vuoti per tutte le categorie (violazioni, audit, file mancanti). Ciò permette alla pipeline generale di concludersi senza corrompere o bloccare l'esecuzione degli altri agenti di analisi paralleli.

3.2.1.6.1.4 LocalSecurityAnalysisAdapter

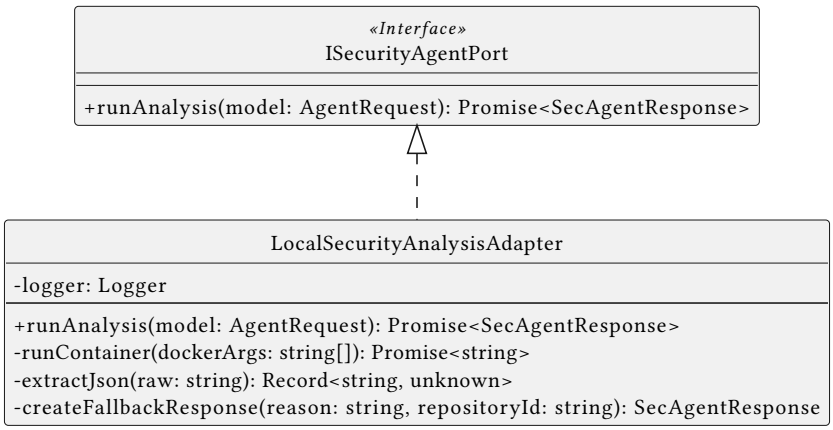

Figure 160: Class Diagram — LocalSecurityAnalysisAdapter

`LocalSecurityAnalysisAdapter` è il Driven Adapter che implementa il port `ISecurityAgentPort`. Gestisce l'orchestrazione locale dell'agente dedicato alla scansione delle vulnerabilità, incapsulando l'esecuzione dell'immagine Docker e la complessa gestione dei risultati aggregati dei vari tool.

- **Orchestrazione Docker Sicura:** Il metodo `runContainer()` utilizza `child_process.spawn` per avviare in isolamento il container `strands-security-analyzer`. Inietta dinamicamente il file `.env` di configurazione (verificandone preventivamente l'esistenza tramite `fs.existsSync`) e mappa il volume condiviso (`analysis_tmp_data`) in cui risiede il codice clonato, garantendo che l'agente abbia accesso esclusivo al contesto necessario.
- **Parsing Resiliente a Tolleranza d'Errore:** Poiché lo `stdout` del container viene spesso inquinato dai log di avvio dei tool sottostanti, il metodo `extractJson()` adotta una strategia a due fasi: dapprima scansiona l'output alla ricerca di un token esplicito di errore (`{"status": "error"`); in sua assenza, utilizza un algoritmo custom di bilanciamento delle parentesi per scansionare il flusso testuale, scartare il rumore e isolare esclusivamente il payload JSON valido associato alla chiave `analysis_report`.
- **Arricchimento del Contesto:** Prima di istanziare la risposta finale, l'adapter agisce da strato di traduzione arricchendo il JSON grezzo: inietta l'identificativo del `repository` e impone lo stato `success` nei `metadata`, allineando l'output grezzo dell'agente alle aspettative strutturali del livello Application.
- **Risoluzione dei fallimenti (Fallback):** In caso di fallimento infrastrutturale (es. crash del container o errore di Docker), l'adapter applica un pattern di graceful degradation tramite `createFallbackResponse()`. Invece di far fallire l'orchestratore, restituisce un DTO strutturato con stato `FAILED` e incapsula esplicitamente il motivo del crash all'interno dell'array `errors`

associandolo al tool fittizio `'agent'`, garantendo la tracciabilità del problema direttamente nel report di sicurezza finale.

3.2.1.6.1.5 MongoDBAdapter

Figure 161: Class Diagram — MongoDBAdapter

`MongoDBAdapter` è il Driven Adapter centralizzato che implementa l'intero livello di persistenza del sistema su MongoDB. Funge da ponte tra i contratti definiti nel livello Application e il database fisico, incapsulando la logica di accesso, traduzione e aggregazione attraverso la libreria Mongoose. Inietta nel costruttore i sei modelli definiti nel dominio e implementa sedici port distinti, suddividendo il suo operato su diverse aree funzionali:

- **Gestione Sicura delle Credenziali:** Tramite i metodi `authorize()`, `save()`, `updatePAT()` e `deletePAT()`, gestisce il ciclo di vita dei token di accesso mappandoli sullo schema `GitCredential`. Oltre alle operazioni CRUD, isola gli errori infrastrutturali intercettando il codice `11000` di MongoDB per tradurlo in un fallimento di "credenziali duplicate" gestibile dal dominio.
- **Tracciamento del Ciclo di Vita dell'Analisi:** Il metodo `saveAnalysis()` inizializza il documento `GitHubAnalysisRecord` all'avvio del processo. Successivamente, `addReportsToAnalysis()` agisce da aggregatore: riceve gli identificativi dei report generati dagli

agenti e aggiorna atomicamente il record principale, associando le chiavi esterne e spostandone lo status a `COMPLETED`.
- **Archiviazione Multi-Report:** Espone tre metodi dedicati (`saveCodeReport()`, `saveDocsReport()` e `saveSecurityReport()`) per riversare le complesse alberature dei Value Object di dominio all'interno dei documenti di database. Più nello specifico:
  - Traduce le metriche di copertura, le issue strutturali e i verdetti dell'AI nello schema `CodeReportModel`.
  - Mappa l'intero albero delle discrepanze testuali, i file mancanti e l'audit delle dipendenze all'interno dello schema `DocumentationReportModel`.
  - Scompone logicamente le vulnerabilità rilevate, separandole per tool di origine (Trivy, Semgrep, Grype) e per categoria, strutturandole all'interno del `SecurityReportModel`.
- **Aggregazione Dinamica in Lettura:** Il metodo `getAnalysisFromId()` orchestra query complesse al posto di una semplice `find`. Recupera il record base e, tramite interrogazioni condizionali sui modelli Mongoose, "pesca" i tre report separati (se presenti), assemblandoli al volo nel DTO `GitHubAnalysisDetailedResult` richiesto dal frontend. Il metodo `getAllAnalysesForUser()` fornisce invece viste generalizzate leggere.
- **Gestione Dinamica delle Collezioni:** Attraverso metodi come `addCollection()`, `deleteCollection()` e `getRepositoryCollection()`, l'adapter gestisce le viste aggregate per utente basate sullo schema `GitHubCollection`. Nell'orchestrare le letture, non duplica i dati storici ma interroga dinamicamente la collezione `github_analyses` filtrando per `url` e `userId` (ordinando per `createdAt`), garantendo che la collezione restituisca uno storico sempre aggiornato.

3.2.1.6.1.6 S3Adapter

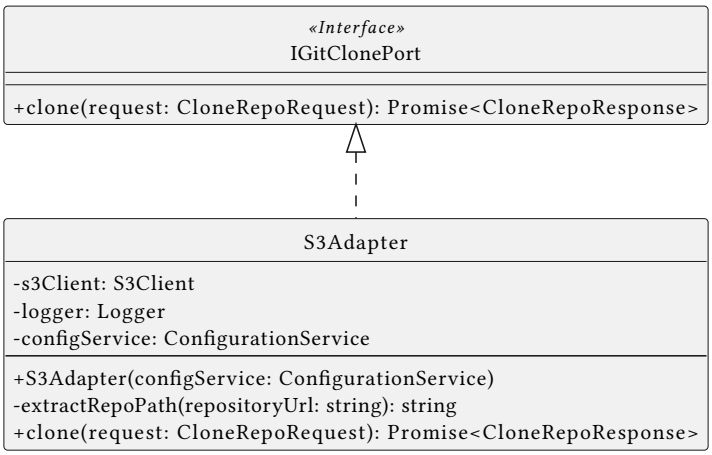

Figure 162: Class Diagram — S3Adapter

`S3Adapter` è il Driven Adapter cloud-native che implementa `IGitClonePort`. Sostituisce la clonazione locale preparando il codice per un'architettura distribuita.

- **Clonazione Dinamica e Autenticazione:** Clona il repository localmente adattando la strategia alla richiesta (`--depth 1` per branch/default, o checkout mirati per commit storici). Gestisce l'autenticazione iniettando il PAT dell'utente o applicando dinamicamente il token di sistema di fallback (`CODE_GUARDIAN_TOKEN`).
- **Compressione e Upload S3:** Una volta clonato il codice, utilizza la libreria `tar` per comprimere l'intera cartella in un archivio `.tar.gz`. Successivamente, lo carica su un bucket AWS S3 tramite `PutObjectCommand`. Questo file diventa il volume di partenza "congelato" per i container di analisi.
- **Gestione Sicura del Ciclo di Vita:** Utilizza un blocco `try/catch` per garantire la pulizia assoluta del file system locale dell'orchestratore. Sia in caso di successo che di eccezione, rimuove

forzatamente sia la directory clonata (`rm -rf`) che l'archivio generato (`fs.unlinkSync`), prevenendo ogni leak di spazio su disco.

3.2.1.6.1.7 ECSCodeAnalysisAdapter

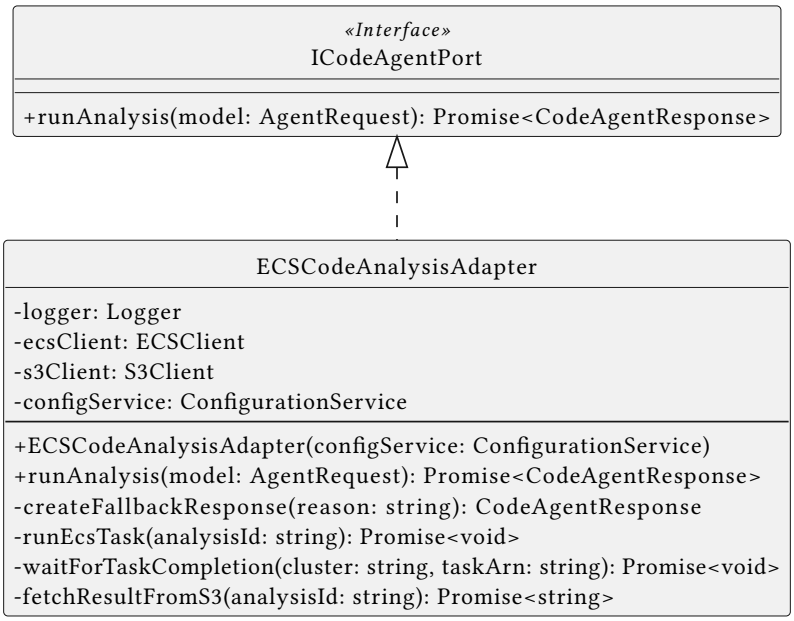

Figure 163: Class Diagram — ECSCodeAnalysisAdapter

`ECSCodeAnalysisAdapter` è il Driven Adapter che implementa `ICodeAgentPort`, delegando l'esecuzione dell'agente di analisi del codice all'infrastruttura serverless AWS ECS (Fargate).

- **Orchestrazione Serverless:** Il metodo `runEcsTask()` avvia un task isolato tramite `RunTaskCommand`, configurando esplicitamente la rete VPC (subnet, security group). Inietta nel container le variabili d'ambiente fondamentali (`ANALYSIS_ID` e `S3_BUCKET_NAME`) necessarie all'agente per scaricare il codice e caricare il risultato.
- **Monitoraggio Attivo (Polling):** Dato che ECS è asincrono, l'adapter implementa `waitForTaskCompletion()`. Questo loop utilizza `DescribeTasksCommand` interrogando AWS ogni 10 secondi fino al raggiungimento dello stato `STOPPED`. Verifica rigorosamente l'exit code del container: un'uscita diversa da zero solleva immediatamente un'eccezione infrastrutturale.
- **Recupero e Arricchimento:** Tramite `GetObjectCommand` scarica da S3 il file di reportistica prodotto (`code_report.json`). Agendo da strato di traduzione, l'adapter inietta nel JSON i metadati applicativi (`repository` ID e `status: 'success'`) prima di passare il controllo al livello Application.
- **Risoluzione dei fallimenti (Fallback):** In caso di timeout, fallimento di AWS o exit code anomalo, il blocco `catch` invoca `createFallbackResponse()`. Invece di far crollare l'applicazione, restituisce un DTO type-safe con verdetto `Critical`, incapsulando il motivo esatto del fallimento infrastrutturale.

3.2.1.6.1.8 ECSDocumentationAnalysisAdapter

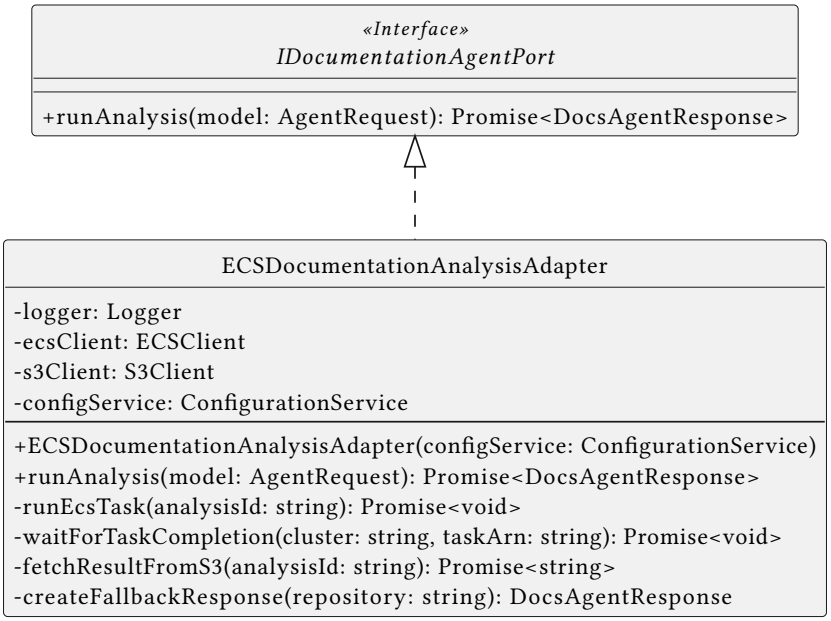

Figure 164: Class Diagram — ECSDocumentationAnalysisAdapter

`ECSDocumentationAnalysisAdapter` è il Driven Adapter che implementa `IDocumentationAgentPort` eseguendo l'agente di documentazione su AWS ECS (Fargate).

- **Esecuzione Distribuita:** Il metodo `runEcsTask()` utilizza `RunTaskCommand` per avviare il task basato sulla definizione `ecsTaskDefinitionDocs`. Passa il contesto operativo iniettando `ANALYSIS_ID` e `S3_BUCKET_NAME` come variabili d'ambiente.
- **Gestione dell'Attesa (Polling):** L'adapter delega a `waitForTaskCompletion()` l'attesa asincrona. Tramite chiamate ripetute a `DescribeTasksCommand`, interroga lo stato del container Fargate e controlla rigorosamente la proprietà `exitCode` per validare l'integrità dell'esecuzione remota.
- **Integrazione S3 e Payload:** Tramite `fetchResultFromS3()`, scarica dal bucket il file prodotto dall'agente (`docs_report.json`). Valida la radice strutturale `analysis_report` e vi inietta i metadati applicativi (`repository` e `status: 'success'`) prima di completare la risoluzione.
- **Graceful Degradation:** Se il task fallisce o restituisce un exit code anomalo, il catch block invoca `createFallbackResponse()`. L'eccezione viene trasformata in una risposta controllata con array vuoti (per violazioni, audit e file mancanti) e stato `'error'`, evitando di far fallire l'intera pipeline globale.

3.2.1.6.1.9 ECSSecurityAnalysisAdapter

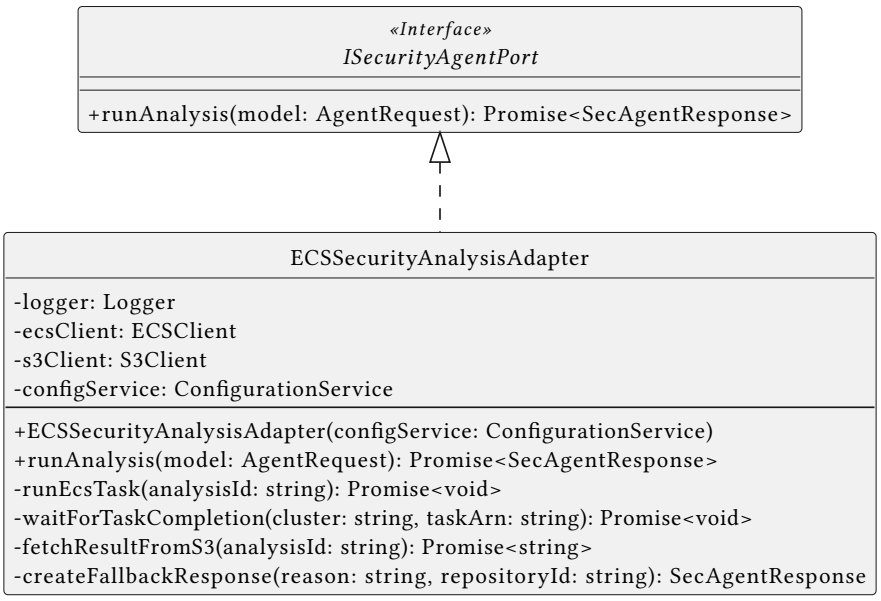

Figure 165: Class Diagram — ECSSecurityAnalysisAdapter

`ECSSecurityAnalysisAdapter` è il Driven Adapter che implementa `ISecurityAgentPort` eseguendo la suite di sicurezza su AWS ECS (Fargate).

- **Scalabilità e Isolamento:** Il metodo `runEcsTask()` lancia il container (`ecsTaskDefinitionSecurity`) applicando le regole di rete VPC (subnet e security group) necessarie per garantire un ambiente cloud isolato durante la scansione delle vulnerabilità.
- **Monitoraggio dell'Esecuzione:** Il flusso viene bloccato in attesa sicura dal metodo `waitForTaskCompletion()`. Questo ciclo verifica che il task ECS raggiunga lo stato `STOPPED` senza errori sistemici, sollevando eccezioni in caso di `exitCode` diverso da zero.
- **Estrazione Dati Cloud-Native:** Al termine dell'esecuzione, il metodo `fetchResultFromS3()` preleva dal bucket l'artefatto JSON (`security_report.json`). L'adapter lo parsa e lo arricchisce dinamicamente con i metadati necessari a soddisfare il contratto del livello Application.
- **Tracciabilità degli Errori:** Il pattern di fallback, gestito da `createFallbackResponse()`, è particolarmente curato. Se l'esecuzione su ECS fallisce, non si limita a impostare lo stato a `FAILED`: inserisce l'eccezione infrastrutturale nell'array `errors` associandola a un tool fittizio (`tool: 'agent'`), per garantire totale trasparenza sul motivo del blocco al front-end.

3.2.1.6.2 Schema

Questa sezione descrive gli Schema Mongoose, ovvero i modelli di dati fisici utilizzati dal livello di persistenza per interfacciarsi con il database MongoDB. Nel rispetto dell'Architettura Esagonale, gli Schema fungono da proiezione persistente delle Entità e dei Value Object definiti nel Domain Layer. Essi incapsulano esclusivamente dettagli infrastrutturali — come i vincoli di unicità, l'indicizzazione per l'ottimizzazione delle query e la gestione dei tipi nativi del database (es. `ObjectId` e `timestamps`) — mantenendo il dominio puro e completamente agnostico rispetto alla tecnologia di memorizzazione.

3.2.1.6.2.1 GitCredential

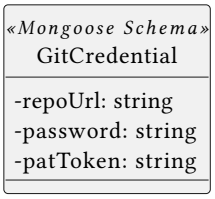

Figure 166: Class Diagram — GitCredential

`GitCredential` è lo schema Mongoose che definisce la struttura del documento MongoDB per le credenziali Git: URL del repository (chiave univoca), hash della password e PAT.

- **Persistenza delle Credenziali:** Rappresenta la proiezione di persistenza dei dati gestiti dai Value Object `RepoURL`, `PATPassword` e `PersonalAccessToken`, adattandoli al formato MongoDB.
- **Ricerca Ottimizzata:** Il campo `repoUrl` è marcato come `unique` e indicizzato (`index: true`), garantendo l'unicità delle credenziali per repository e ricerche fulminee durante l'autorizzazione.

3.2.1.6.2.2 GitHubAnalysisRecord

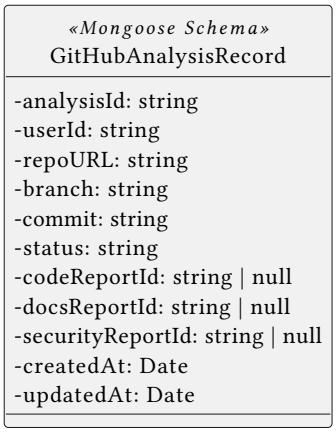

Figure 167: Class Diagram — GitHubAnalysisRecord

`GitHubAnalysisRecord` è lo schema Mongoose che definisce la persistenza dell'entità `GitHubAnalysis`, memorizzando i metadati dell'analisi e i riferimenti ai vari report generati.

- **Proiezione dell'Entità:** Mappa gli attributi gestiti dai Value Object `AnalysisId`, `UserId`, `RepoURL`, `BranchName`, `CommitHash` e l'enumerazione `AnalysisStatus` in tipi primitivi persistibili nel database.
- **Tracciamento dei Report:** Mantiene i riferimenti opzionali (di tipo stringa, derivati dal Value Object `ReportId`) ai documenti separati che contengono i payload massivi generati dagli agenti.
- **Gestione Temporale:** Utilizza l'opzione `timestamps: true` di Mongoose per gestire automaticamente i campi `createdAt` e `updatedAt`.

3.2.1.6.2.3 GitHubCollection

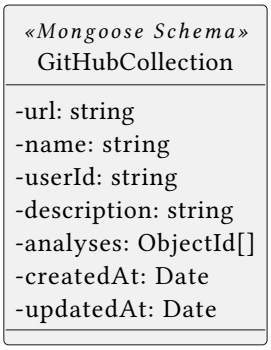

Figure 168: Class Diagram — GitHubCollection

`GitHubCollection` è lo schema Mongoose che raggruppa le analisi ripetute su uno stesso repository per un dato utente, creando una vista "storica" o di progetto.

- **Relazioni MongoDB:** Il campo `analyses` utilizza `ObjectId` per referenziare multipli documenti della collezione `github_analyses` (ossia analisi derivanti dall'entità `GitHubAnalysis`), modellando una relazione uno-a-molti. Tuttavia, per garantire uno storico sempre aggiornato e inclusivo anche delle analisi precedenti alla creazione della collezione, il `MongoDBAdapter` non utilizza questo campo in lettura: le analisi vengono recuperate dinamicamente tramite query diretta sulla collezione `github_analyses`, filtrando per `url` e `userId`. Il campo rimane presente per compatibilità strutturale del documento.
- **Indice Composto:** Definisce un indice composto e univoco su `{ url: 1, userId: 1 }` per garantire che un utente non possa creare più collezioni per lo stesso repository, ottimizzando contemporaneamente le query di lookup basate in origine su `RepoURL` e `UserId`.

3.2.1.6.2.4 CodeReportModel

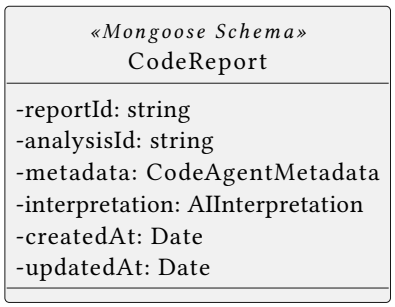

Figure 169: Class Diagram — CodeReportModel

`CodeReportModel` è lo schema Mongoose che archivia i risultati dettagliati prodotti dall'agente di analisi del codice, fungendo da proiezione persistente per l'entità `CodeAgentReport`.

- **Integrità Strutturale:** Utilizza regex per validare che `reportId` e `analysisId` (rappresentazioni testuali di `ReportId` e `AnalysisId`) siano formattati correttamente come UUID v7.
- **Sub-documenti Strutturati:** Fa un uso estensivo di classi Schema interne per mappare fedelmente l'alberatura complessa prodotta dai Value Object `CodeAgentMetadata` e `AIInterpretation`.
- **Indicizzazione Strategica:** Crea indici specifici su `interpretation.verdict` (direttamente correlato all'enumerazione `VerdictStatus`) e `metadata.language` per permettere aggregazioni e filtri rapidi a livello di database.

3.2.1.6.2.5 DocumentationReportModel

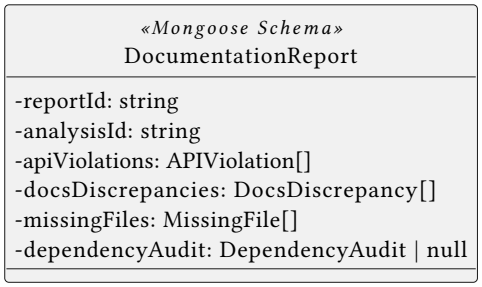

Figure 170: Class Diagram — DocumentationReportModel

`DocumentationReportModel` è lo schema Mongoose dedicato al salvataggio massivo dei risultati emessi dall'agente di analisi della documentazione, fungendo da proiezione persistente per l'entità `DocumentationReport`.

- **Mappatura delle Discrepanze:** Salva direttamente gli array di oggetti complessi derivati dai Value Object `APIViolation`, `DocsDiscrepancy`, `MissingFile` e `DependencyAudit`.
- **Integrità Relazionale:** Come gli altri report, vincola i campi legati a `ReportId` e `AnalysisId` ad essere univoci.
- **Ottimizzazione delle Ricerche:** Implementa indici manuali sui campi di severità annidati (correlati all'enumerazione `SeverityLevel`), fondamentali per estrarre rapidamente le metriche senza caricare interi documenti in memoria.

3.2.1.6.2.6 SecurityReportModel

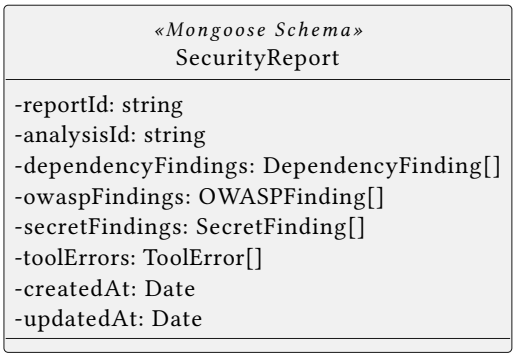

Figure 171: Class Diagram — SecurityReportModel

`SecurityReportModel` è lo schema Mongoose progettato per immagazzinare in modo strutturato le vulnerabilità riscontrate, fungendo da proiezione persistente per l'entità `SecurityReport`.

- **Categorizzazione Multi-Tool:** Separa logicamente i risultati in array di sub-documenti tipizzati che riflettono esattamente le collezioni di Value Object dell'entità: `DependencyFinding`, `OWASPFinding`, `SecretFinding` e gli errori `ToolError`.
- **Indicizzazione Profonda:** Include indici composti e specifici sulle proprietà annidate (come i livelli di severità legati a `SeverityFinding` o le categorie OWASP) per supportare query ad alte prestazioni necessarie per i cruscotti di sicurezza.

3.2.1.7 Presentation

3.2.1.7.1 Helpers

Questa sezione descrive i componenti ausiliari del livello di presentazione, responsabili di fornire funzionalità trasversali riutilizzabili dai controller. In particolare, raggruppa i meccanismi di

autenticazione e autorizzazione basati su JWT, isolando la logica di validazione dei token dal codice applicativo dei controller e garantendo che ogni endpoint protetto possa verificare l'identità del chiamante in modo uniforme e disaccoppiato.

3.2.1.7.1.1 JwtHelper

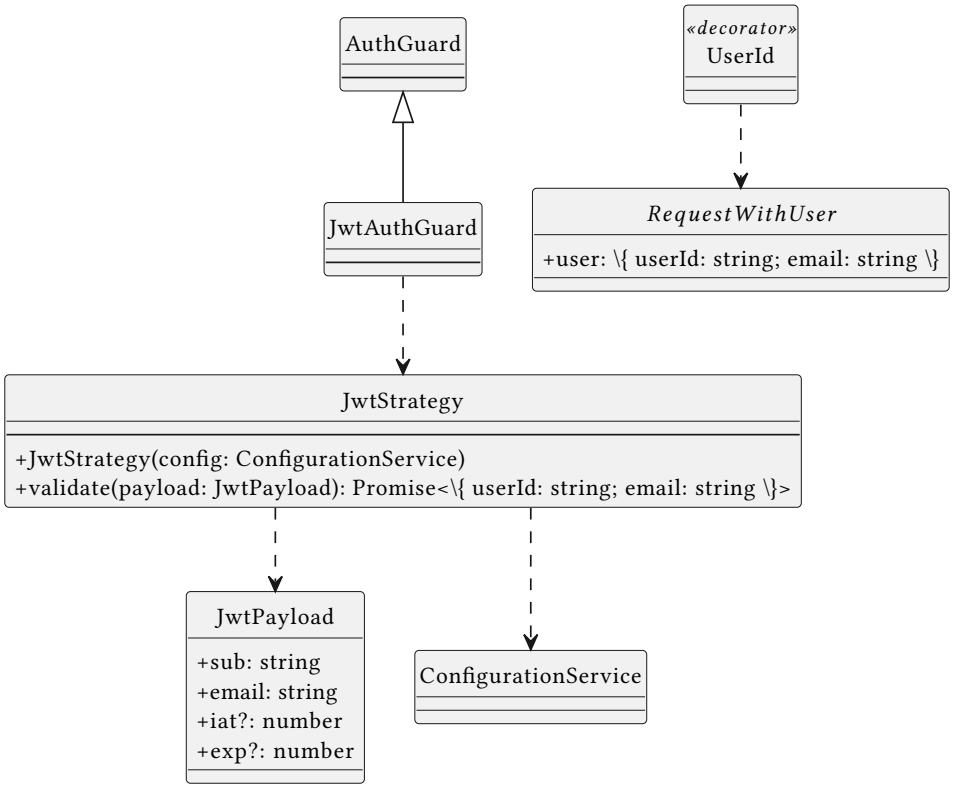

Figure 172: Class Diagram — JwtHelper

`JwtHelper` raggruppa i componenti responsabili dell'autenticazione basata su JWT, integrando il meccanismo di validazione dei token con il framework applicativo. Include la strategia di validazione (`JwtStrategy`), il meccanismo di protezione degli endpoint (`JwtAuthGuard`) e un decoratore per l'estrazione dell'identità utente (`UserId`).

- **Separazione tra Validazione e Accesso:** La `JwtStrategy` è responsabile della validazione del token e della costruzione del contesto utente, mentre `JwtAuthGuard` si occupa di applicare tale validazione agli endpoint protetti.
- **Integrazione con il Framework di Autenticazione:** Il guard estende il meccanismo standard (`AuthGuard`), permettendo di riutilizzare l'infrastruttura di autenticazione senza introdurre logica applicativa nel controller.
- **Accesso Tipizzato all'Utente:** Il decoratore `UserId` consente di accedere in modo tipizzato alle informazioni dell'utente estratte dal token, evitando la propagazione diretta del modello HTTP nei livelli superiori.

3.2.1.7.2 Controllers

Questa sezione descrive i Controller, i componenti del livello di presentazione incaricati di esporre gli endpoint HTTP e di tradurre le richieste in ingresso nei comandi applicativi corrispondenti. Nel rispetto dell'Architettura Esagonale, i controller non contengono logica di business: si limitano a trasformare i DTO di trasporto in comandi, delegare l'esecuzione ai rispettivi Use Case e restituire al client i DTO di risposta appropriati.

3.2.1.7.2.1 AnalysisController

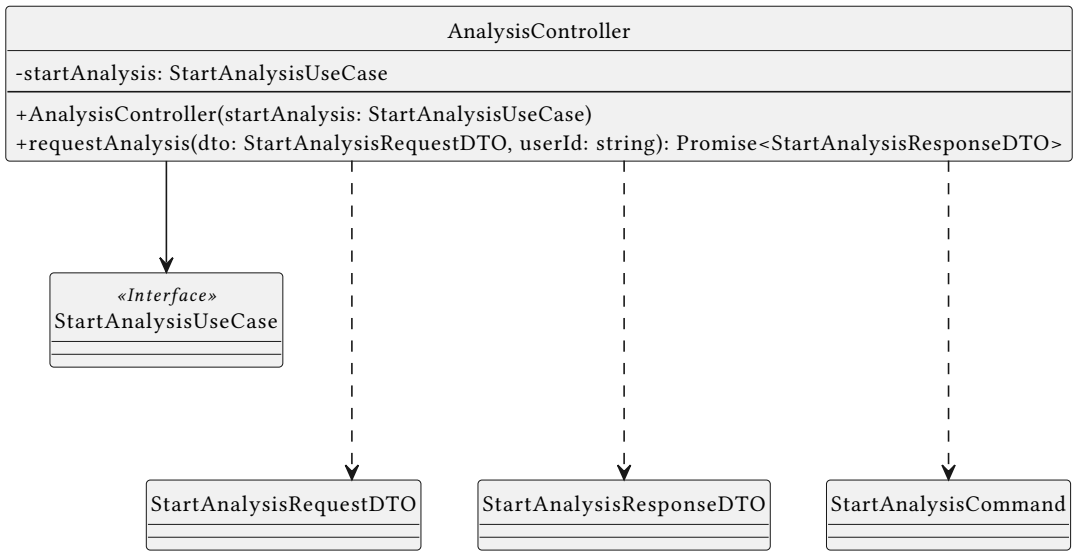

Figure 173: Class Diagram — AnalysisController

`AnalysisController` espone l'endpoint HTTP per richiedere l'avvio di una nuova analisi su un repository. Inietta `StartAnalysisUseCase`, al quale delega completamente l'esecuzione del flusso applicativo, ricevendo un `StartAnalysisRequestDTO` e restituendo un `StartAnalysisResponseDTO`.

- **Separazione tra API e Dominio:** Il controller traduce il `StartAnalysisRequestDTO` in un `StartAnalysisCommand`, mantenendo separati il modello di trasporto HTTP e quello applicativo.
- **Delegazione del Flusso:** La logica di orchestrazione è interamente demandata al `StartAnalysisUseCase`, mantenendo il controller come semplice punto di ingresso e uscita del sistema.

3.2.1.7.2.2 PatController

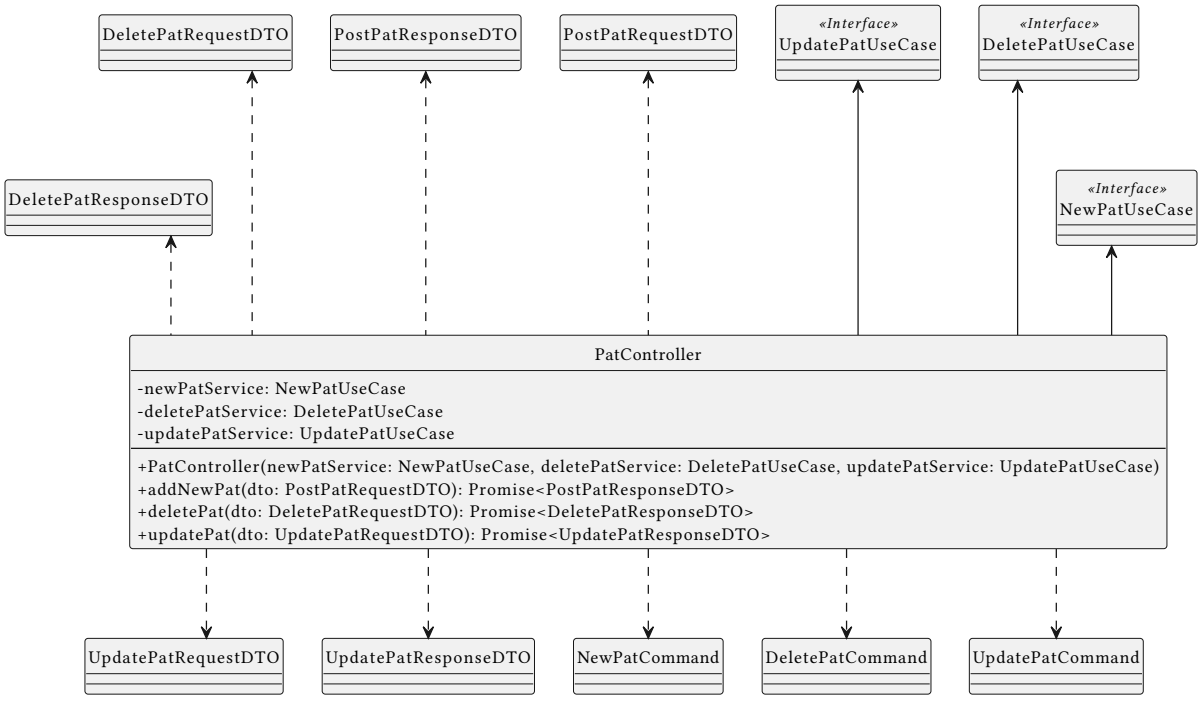

Figure 174: Class Diagram — PatController

`PatController` espone gli endpoint HTTP per la gestione dei Personal Access Token. Inietta tre use case distinti (`NewPatUseCase`, `DeletePatUseCase`, `UpdatePatUseCase`), ciascuno responsabile di una specifica operazione, e restituisce i rispettivi DTO di risposta.

- **Segregazione dei Casi d'Uso:** Ogni operazione (creazione, aggiornamento, eliminazione) è delegata a un use case dedicato, garantendo isolamento dei flussi applicativi e coerenza con il principio di singola responsabilità.
- **Uniformità del Flusso Applicativo:** Tutti gli endpoint seguono lo stesso schema: trasformazione del DTO di input in command e delega al caso d'uso, favorendo consistenza e manutenibilità.

3.2.1.7.2.3 RepositoriesController

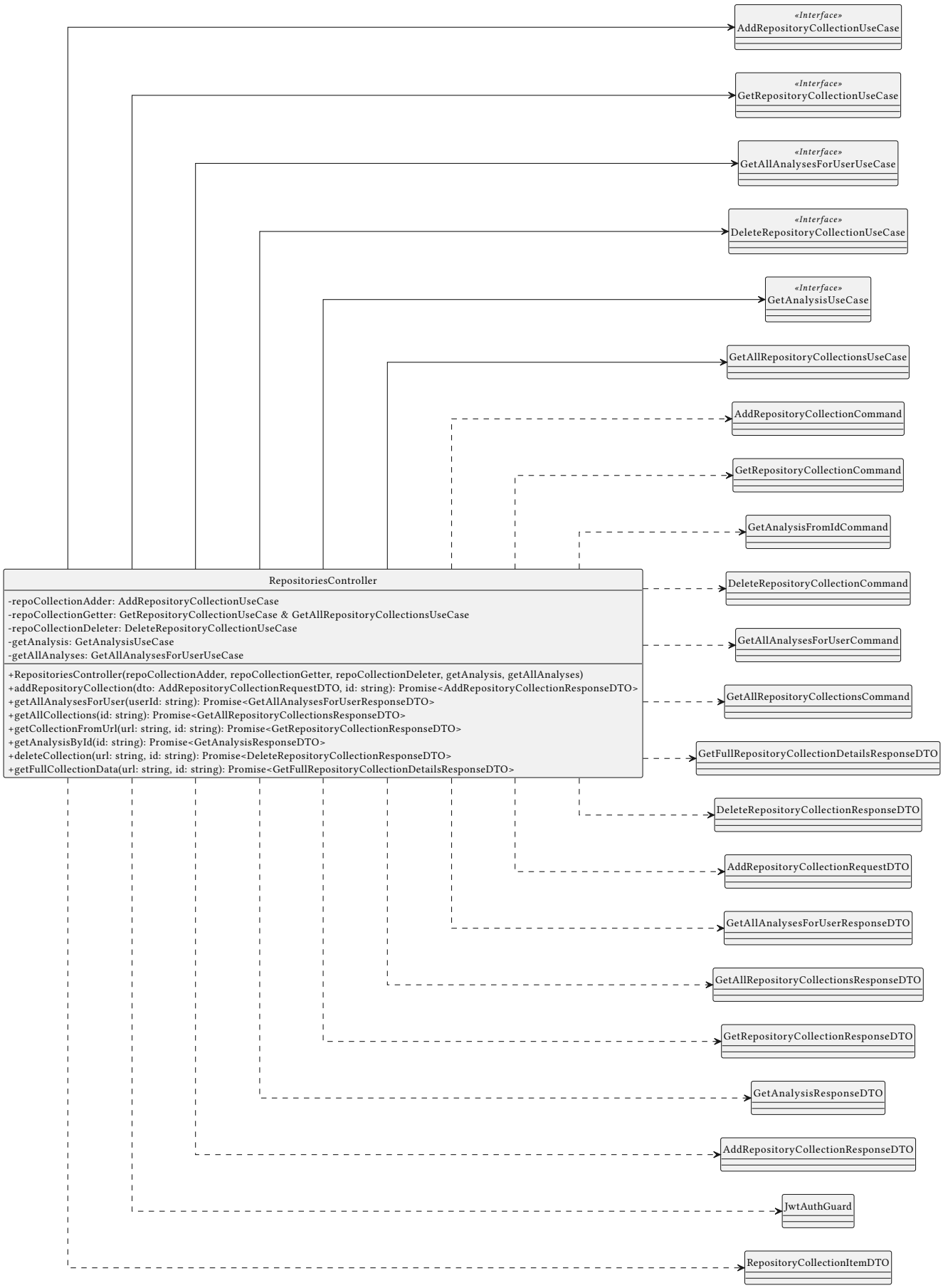

Figure 175: Class Diagram — RepositoriesController

`RepositoriesController` espone gli endpoint HTTP per la gestione delle collezioni di repository e delle analisi associate. Inietta diversi use case per coprire operazioni di creazione, recupero e cancellazione, restituendo DTO di risposta specifici per ciascun endpoint.

- **Composizione dei Casi d'Uso:** Il controller coordina più use case distinti per gestire scenari complessi, mantenendo comunque separata la logica applicativa nei rispettivi componenti.
- **Aggregazione dei Dati in Lettura:** Alcuni endpoint combinano risultati provenienti da più casi d'uso per restituire viste più ricche, centralizzando l'aggregazione a livello di controller senza introdurre logica di business. In particolare, l'endpoint `getFullCollectionData` recupera prima gli identificativi delle analisi dalla collezione, quindi esegue il recupero dei dettagli di ciascuna analisi in parallelo tramite `Promise.all`, ottimizzando i tempi di risposta in presenza di collezioni con molte analisi associate.

3.2.1.7.3 Presentation DTOs - Requests

Questa sezione descrive i DTO di richiesta del livello di presentazione, ovvero i contratti che definiscono la struttura dei dati in ingresso per ciascun endpoint HTTP. Essi fungono da strato di traduzione tra il formato atteso dal client e il modello applicativo interno, garantendo che i controller ricevano dati strutturati e tipizzati prima di costruire i comandi da inviare ai Use Case.

3.2.1.7.3.1 AddRepositoryCollectionRequestDTO

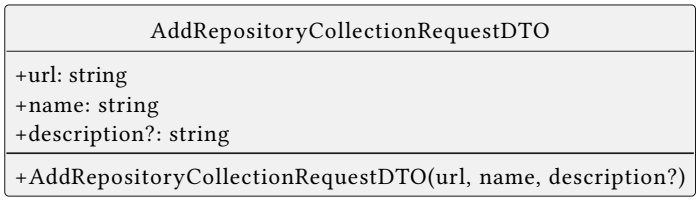

Figure 176: Class Diagram — AddRepositoryCollectionRequestDTO

`AddRepositoryCollectionRequestDTO` definisce il contratto del body della richiesta HTTP per la creazione di una nuova collezione di repository. I campi `url` e `name` sono obbligatori, mentre `description` è opzionale, riflettendo la possibilità di fornire metadati aggiuntivi senza renderli necessari al completamento dell'operazione.

3.2.1.7.3.2 DeletePatRequestDTO

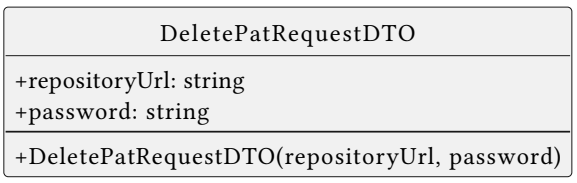

Figure 177: Class Diagram — DeletePatRequestDTO

`DeletePatRequestDTO` definisce il contratto del body della richiesta HTTP per la rimozione di un Personal Access Token. I campi `repositoryUrl` e `password` sono obbligatori, garantendo che il sistema disponga delle informazioni necessarie per identificare il repository e autorizzare l'operazione.

3.2.1.7.3.3 PostPatRequestDTO

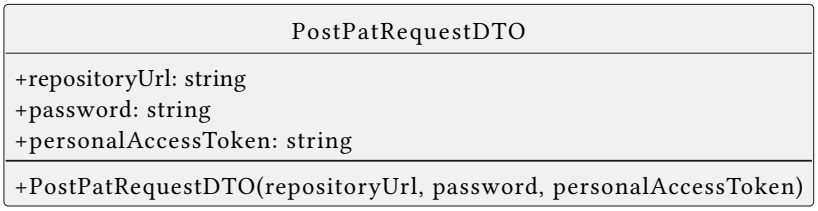

Figure 178: Class Diagram — PostPatRequestDTO

`PostPatRequestDTO` definisce il contratto del body della richiesta HTTP per la creazione di un Personal Access Token. I campi `repositoryUrl`, `password` e `personalAccessToken` sono obbligatori, garantendo che il sistema disponga delle informazioni necessarie per associare e validare il token rispetto al repository indicato.

3.2.1.7.3.4 StartAnalysisRequestDTO

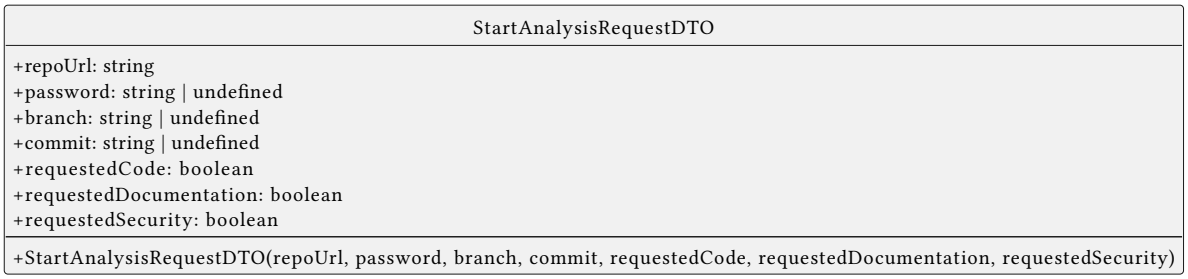

Figure 179: Class Diagram — StartAnalysisRequestDTO

`StartAnalysisRequestDTO` definisce il contratto del body della richiesta HTTP per l'avvio di una nuova analisi. Il campo `repoUrl` è obbligatorio, mentre `password`, `branch` e `commit` sono opzionali, permettendo di specificare credenziali e contesto di analisi solo quando necessario. I flag booleani (`requestedCode`, `requestedDocumentation`, `requestedSecurity`) consentono al client di configurare il tipo di analisi richiesta.

- **Configurabilità dell'Analisi:** La presenza di flag espliciti permette al client di selezionare in modo granulare le componenti dell'analisi, evitando la necessità di endpoint distinti per ogni variante.

3.2.1.7.3.5 UpdatePatRequestDTO

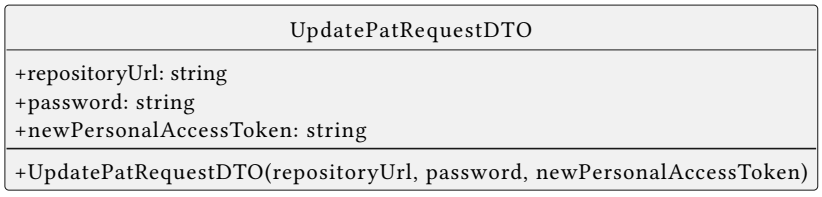

Figure 180: Class Diagram — UpdatePatRequestDTO

`UpdatePatRequestDTO` definisce il contratto del body della richiesta HTTP per l'aggiornamento di un Personal Access Token. I campi `repositoryUrl`, `password` e `newPersonalAccessToken` sono obbligatori, assicurando che il sistema possa identificare il contesto corretto e sostituire in modo sicuro il token esistente.

3.2.1.7.4 Presentation DTOs - Response

Questa sezione descrive i DTO di risposta del livello di presentazione, ovvero i contratti che definiscono la struttura dei dati restituiti al client per ciascun endpoint HTTP. Essi fungono da strato di traduzione tra i risultati prodotti dal livello applicativo e il formato esposto verso l'esterno, garantendo il disaccoppiamento tra i modelli interni e la rappresentazione pubblica dell'API.

3.2.1.7.4.1 AddRepositoryCollectionResponseDTO

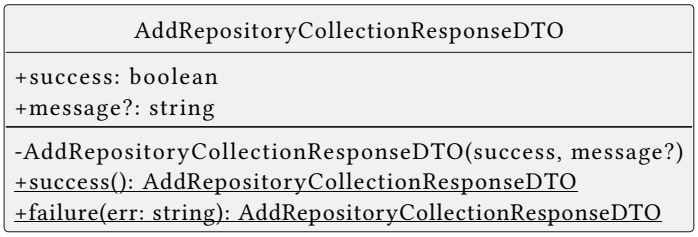

Figure 181: Class Diagram — AddRepositoryCollectionResponseDTO

`AddRepositoryCollectionResponseDTO` definisce il contratto della risposta HTTP per l'operazione di creazione di una collezione di repository. Il campo booleano `success` indica l'esito dell'operazione, mentre `message` fornisce un eventuale dettaglio descrittivo in caso di errore.

- **Factory Method per la Creazione:** L'utilizzo di metodi statici (`success`, `failure`) centralizza la costruzione delle risposte, garantendo coerenza nella rappresentazione degli esiti.

3.2.1.7.4.2 DeletePatResponseDTO

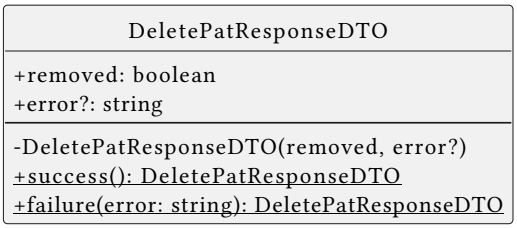

Figure 182: Class Diagram — DeletePatResponseDTO

`DeletePatResponseDTO` definisce la risposta HTTP per l'operazione di rimozione di un Personal Access Token. Il campo `removed` rappresenta l'esito dell'operazione, mentre `error` consente di trasportare un messaggio descrittivo in caso di fallimento.

- **Esplicitazione dell'Esito:** L'utilizzo di un campo booleano dedicato consente al client di distinguere chiaramente tra successo e fallimento senza dipendere esclusivamente dal codice HTTP.

3.2.1.7.4.3 DeleteRepositoryCollectionResponseDTO

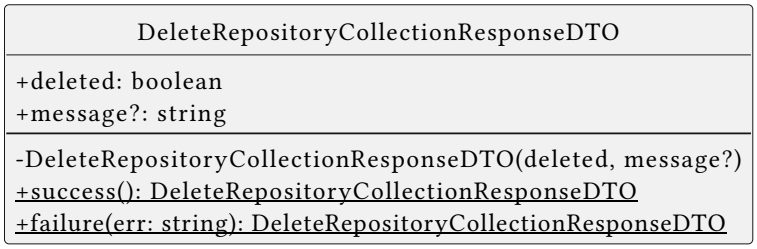

Figure 183: Class Diagram — DeleteRepositoryCollectionResponseDTO

`DeleteRepositoryCollectionResponseDTO` definisce la risposta HTTP per l'operazione di eliminazione di una collezione di repository. Il campo `deleted` indica se l'operazione è stata completata con successo, mentre `message` fornisce eventuali dettagli aggiuntivi.

- **Contratto Semplice e Tipizzato:** La struttura minimale del DTO riflette la natura dell'operazione, fornendo al client un'informazione chiara e immediata sull'esito.

3.2.1.7.4.4 GetAllAnalysesForUserResponseDTO

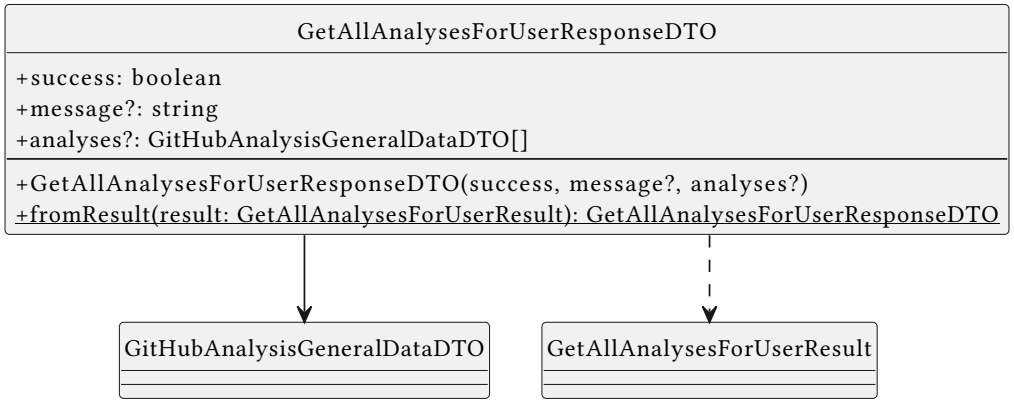

Figure 184: Class Diagram — GetAllAnalysesForUserResponseDTO

`GetAllAnalysesForUserResponseDTO` definisce il contratto della risposta HTTP per il recupero delle analisi associate a un utente. Oltre ai campi `success` e `message`, espone la collezione `analyses`, che corrisponde a una rappresentazione sintetica dei dati di analisi tramite `GitHubAnalysisGeneralDataDTO`.

- **Aggregazione di Dati:** Il DTO raccoglie una lista di elementi, permettendo al client di ottenere una visione complessiva delle analisi con una singola risposta.
- **Separazione tra Result e Response:** Il metodo `fromResult` consente di trasformare l'oggetto applicativo (`GetAllAnalysesForUserResult`) nel formato esposto verso l'esterno, mantenendo disaccoppiati i livelli applicativo e di presentazione.

3.2.1.7.4.5 GetAllRepositoryCollectionsResponseDTO

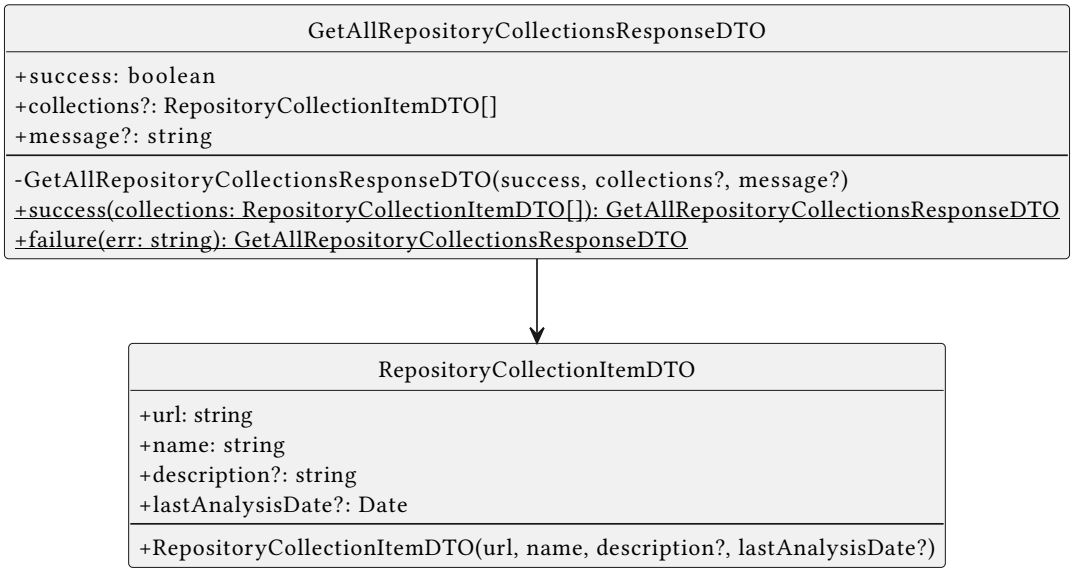

Figure 185: Class Diagram — GetAllRepositoryCollectionsResponseDTO

`GetAllRepositoryCollectionsResponseDTO` definisce la risposta HTTP per il recupero delle collezioni di repository associate all'utente. Il campo `collections` contiene una lista di `RepositoryCollectionItemDTO`, che rappresentano una proiezione sintetica delle informazioni rilevanti per ciascun repository.

- **Aggregazione di Elementi:** Il DTO espone una collezione tipizzata, consentendo al client di ottenere una vista compatta delle risorse disponibili.
- **Separazione della Proiezione:** L'utilizzo di `RepositoryCollectionItemDTO` evita l'esposizione diretta di modelli interni, mantenendo il disaccoppiamento tra livelli.

3.2.1.7.4.6 GetAnalysisResponseDTO

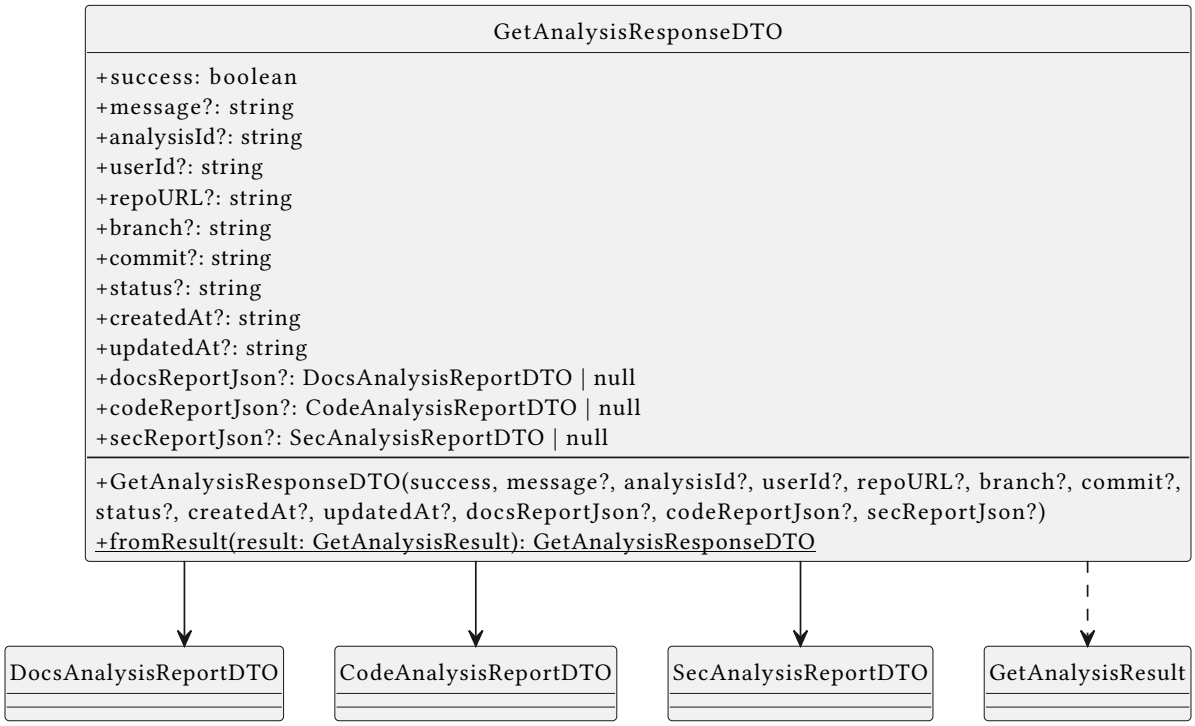

Figure 186: Class Diagram — GetAnalysisResponseDTO

`GetAnalysisResponseDTO` definisce la risposta HTTP per il recupero dettagliato di una singola analisi. Oltre ai metadati principali (identificativi, repository, stato e timestamp), include i report opzionali (`docsReportJson`, `codeReportJson`, `secReportJson`) che rappresentano i risultati delle diverse componenti di analisi.

- **Composizione di Dati Complessi:** Il DTO aggrega più sotto-strutture (`DocsAnalysisReportDTO`, `CodeAnalysisReportDTO`, `SecAnalysisReportDTO`), permettendo al client di ottenere una vista completa dell'analisi.
- **Gestione di Dati Opzionali:** La presenza di campi opzionali consente di rappresentare analisi parziali o in corso.
- **Separazione tra Result e Response:** Il metodo `fromResult` realizza la trasformazione dal livello applicativo (`GetAnalysisResult`) al formato esposto verso l'esterno.

3.2.1.7.4.7 GetFullRepositoryCollectionDetailsResponseDTO

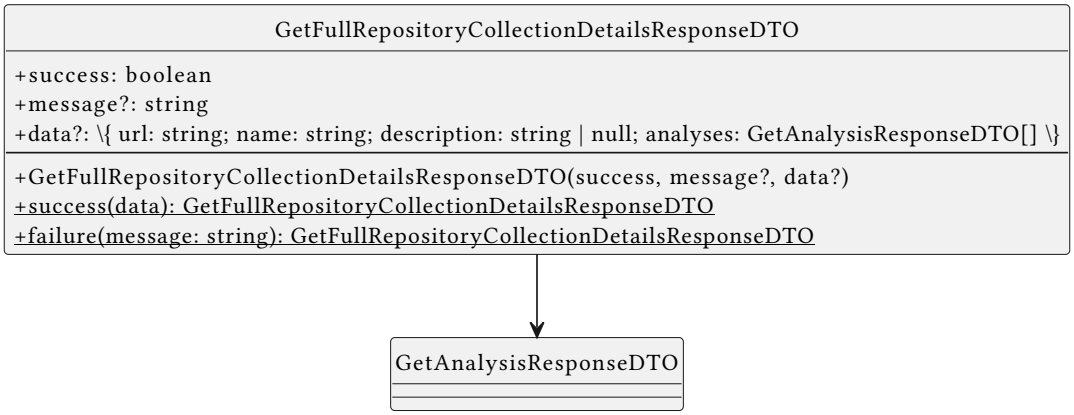

Figure 187: Class Diagram — GetFullRepositoryCollectionDetailsResponseDTO

`GetFullRepositoryCollectionDetailsResponseDTO` definisce la risposta HTTP per il recupero completo dei dettagli di una collezione di repository. Il campo `data` aggrega le informazioni della collezione insieme alla lista delle analisi associate, rappresentate tramite `GetAnalysisResponseDTO`.

- **Aggregazione Gerarchica:** Il DTO combina informazioni di alto livello della collezione con una lista dettagliata di analisi, fornendo una vista completa in un'unica risposta.
- **Composizione di DTO:** L'utilizzo di `GetAnalysisResponseDTO` consente di riutilizzare una rappresentazione già definita, mantenendo coerenza tra endpoint.

3.2.1.7.4.8 GetRepositoryCollectionResponseDTO

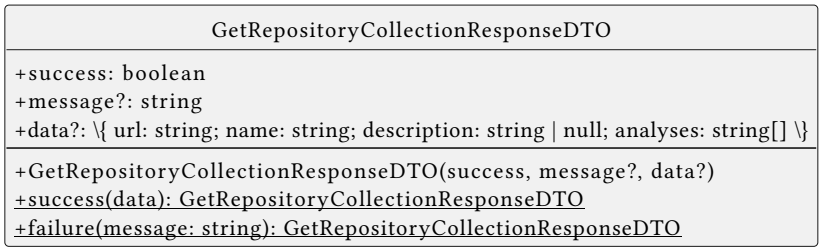

Figure 188: Class Diagram — GetRepositoryCollectionResponseDTO

`GetRepositoryCollectionResponseDTO` definisce la risposta HTTP per il recupero sintetico di una collezione di repository. Il campo `data` include le informazioni principali della collezione e una lista di identificativi (`analyses`) delle analisi associate.

- **Vista Sintetica:** A differenza della versione completa, il DTO espone solo gli identificativi delle analisi, riducendo il payload e migliorando le performance.
- **Differenziazione dei Livelli di Dettaglio:** La presenza di DTO distinti per vista completa e sintetica consente al client di scegliere il livello di dettaglio più appropriato.

3.2.1.7.4.9 PostPatResponseDTO

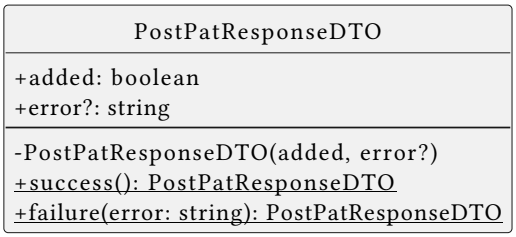

Figure 189: Class Diagram — PostPatResponseDTO

`PostPatResponseDTO` definisce la risposta HTTP per l'operazione di creazione di un Personal Access Token. Il campo `added` indica l'esito dell'operazione, mentre `error` consente di trasportare un messaggio descrittivo in caso di fallimento.

- **Factory Method per la Creazione:** I metodi statici (`success`, `failure`) garantiscono una costruzione coerente delle risposte, evitando stati non validi.

3.2.1.7.4.10 StartAnalysisResponseDTO

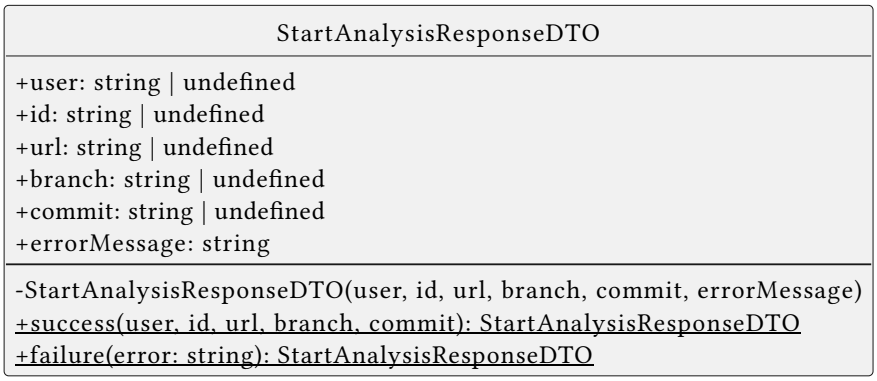

Figure 190: Class Diagram — StartAnalysisResponseDTO

`StartAnalysisResponseDTO` definisce la risposta HTTP per l'avvio di una nuova analisi. Il DTO include i principali metadati dell'analisi avviata (`user`, `id`, `url`, `branch`, `commit`) e un campo `errorMessage` che rappresenta il risultato dell'operazione.

- **Trasporto di Informazioni Operative:** In caso di successo, il DTO restituisce i dati identificativi dell'analisi appena avviata.
- **Messaggio Sempre Presente:** Il campo `errorMessage` viene popolato con la stringa fissa `Analysis Started Successfully` in caso di successo, o con il messaggio di errore specifico in caso di fallimento, fornendo un feedback uniforme al client indipendentemente dall'esito.
- **Costruzione Controllata:** I metodi statici assicurano che i dati siano valorizzati solo nei casi appropriati, mantenendo coerenza tra successo e fallimento.

3.2.1.7.4.11 UpdatePatResponseDTO

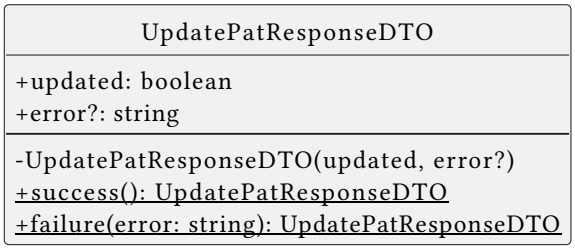

Figure 191: Class Diagram — UpdatePatResponseDTO

`UpdatePatResponseDTO` definisce la risposta HTTP per l'aggiornamento di un Personal Access Token. Il campo `updated` indica l'esito dell'operazione, mentre `error` fornisce eventuali dettagli in caso di errore.

- **Contratto Semplice:** La struttura minimale riflette la natura dell'operazione, permettendo al client di interpretare facilmente il risultato.
- **Coerenza dei Pattern:** L'utilizzo di factory method mantiene allineato il comportamento con gli altri DTO di risposta.

3.2.2 Microservizio Account

L'Account Microservice rappresenta il modulo centrale per la gestione del ciclo di vita delle identità all'interno di *CodeGuardian*. Progettato seguendo i principi dell'**Architettura Esagonale**, il servizio isola rigorosamente i processi core — quali la gestione delle utenze, l'autenticazione basata su JWT e la sicurezza delle credenziali — dalle tecnologie di persistenza (PostgreSQL) e di cifratura (Bcrypt). Grazie a una netta separazione tra porte e adattatori, il microservizio garantisce l'integrità del dominio utente e la flessibilità nell'evoluzione dei criteri di sicurezza, fungendo da garante per l'accesso protetto a tutte le funzionalità della piattaforma.

3.2.2.1 Flussi completi di Esecuzione

Il diagramma seguente illustra un flusso operativo completo, evidenziando l'interazione tra i livelli dell'architettura esagonale a partire da un singolo controller HTTP fino alla conclusione della richiesta. Questa sezione ha l'obiettivo di fornire una panoramica ad alto livello del percorso di esecuzione, evidenziando i passaggi chiave e le interazioni tra i componenti, senza entrare nei dettagli di implementazione specifici, interazioni con oggetti di dominio o contratti tra la parti, in quanto questo livello di dettaglio sará visibile nelle sezioni successive.

3.2.2.1.1 Registration

Il flusso di registrazione gestisce la creazione di un nuovo account utente, verificando l'unicità dell'email, eseguendo l'hashing della password e generando i token di sessione iniziali.

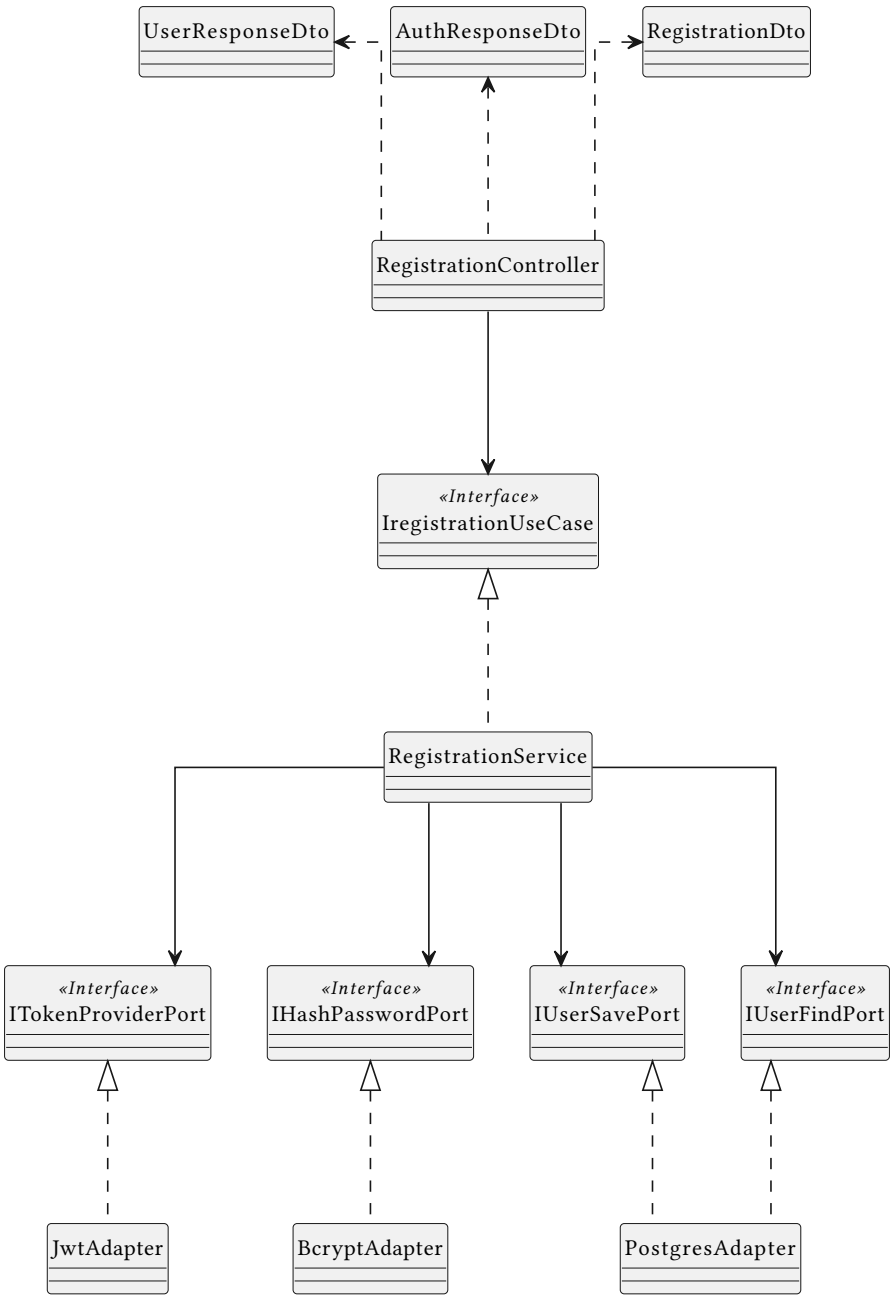

Figure 192: Controller Diagram — RegistrationControllerReachableClasses

3.2.2.1.2 Login

Il flusso di login autentica un utente esistente confrontando le credenziali fornite con quelle memorizzate e generando una nuova sessione di lavoro.

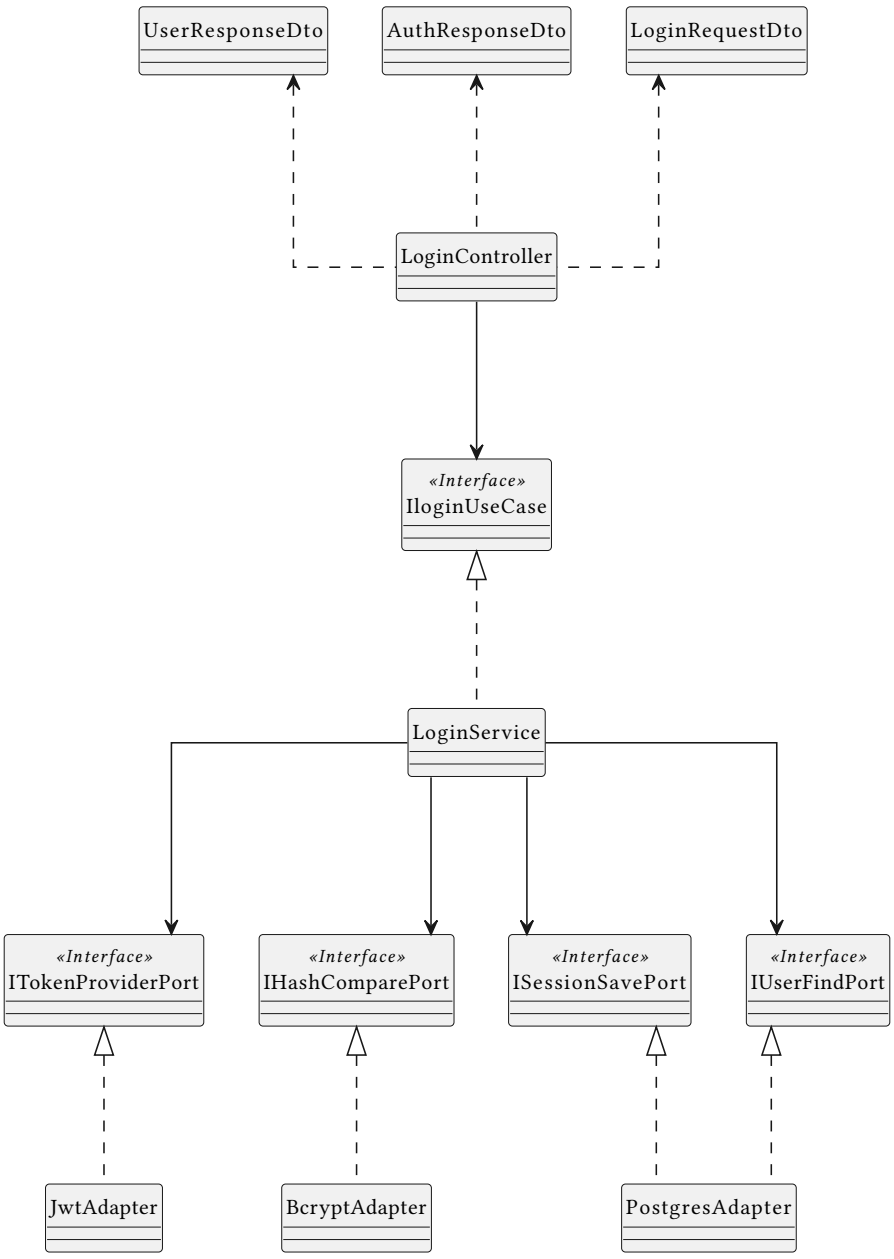

Figure 193: Controller Diagram — LoginControllerReachableClasses

3.2.2.1.3 Logout

Il flusso di logout invalida la sessione corrente dell'utente rimuovendo il refresh token memorizzato nel sistema di persistenza.

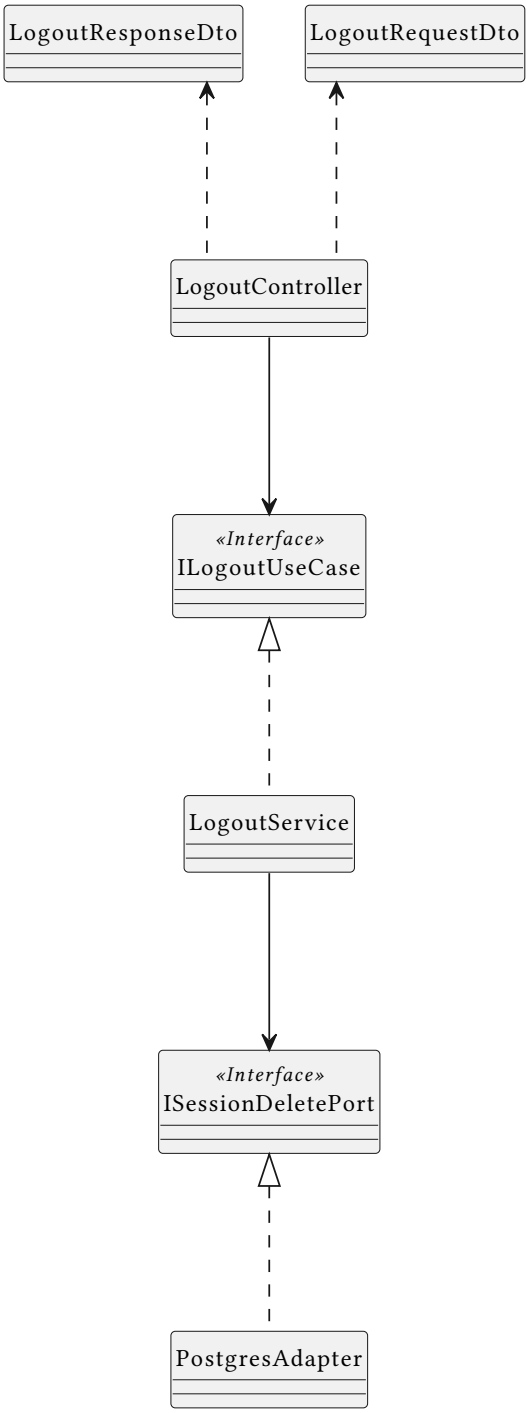

Figure 194: Controller Diagram — LogoutControllerReachableClasses

3.2.2.1.4 Update

Il flusso di aggiornamento permette a un utente autenticato di modificare le proprie credenziali (password), garantendo che l'identità sia verificata tramite il token JWT.

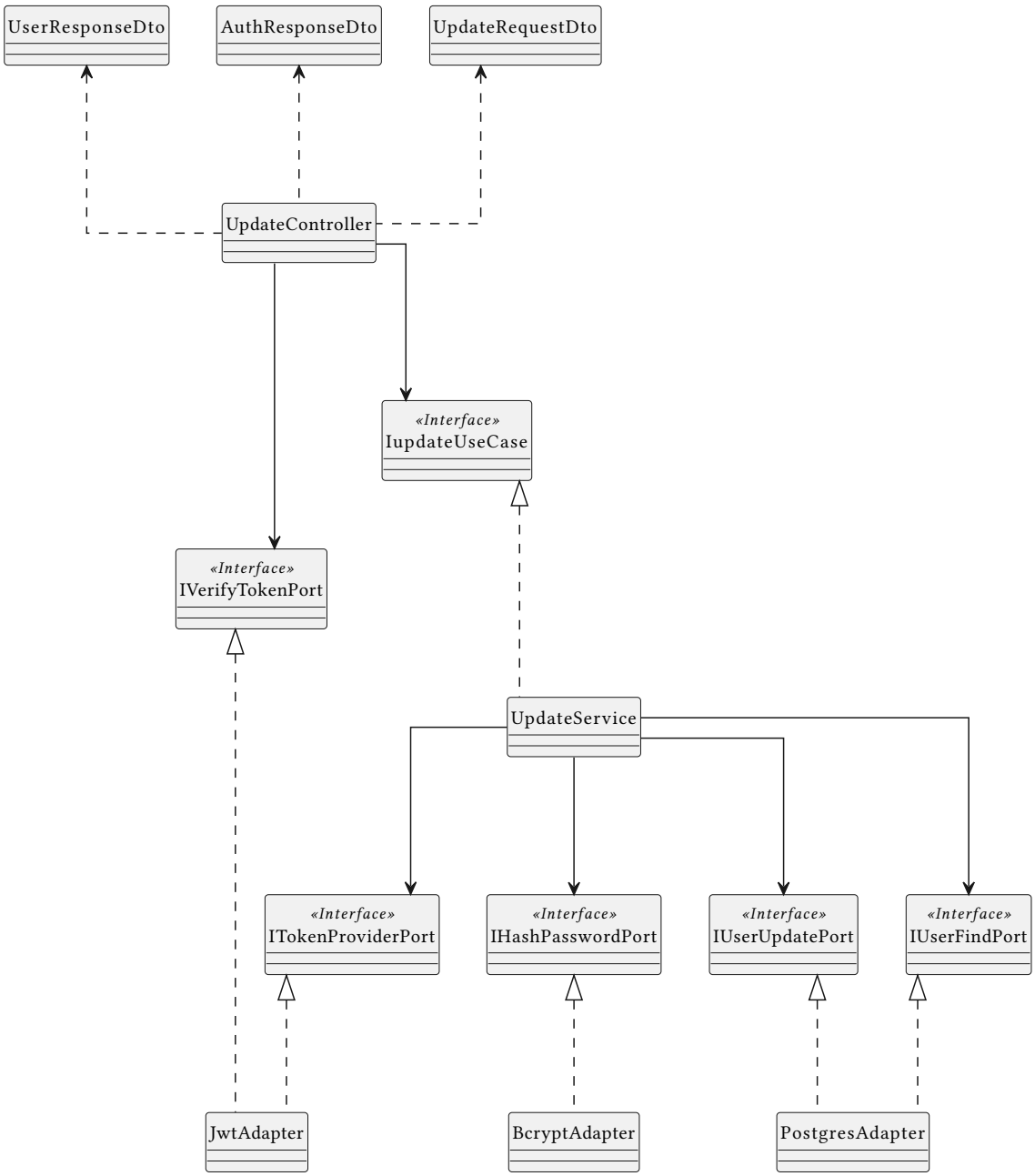

Figure 195: Controller Diagram — UpdateControllerReachableClasses

3.2.2.1.5 Delete

Il flusso di cancellazione permette a un utente autenticato di rimuovere definitivamente il proprio account e tutti i dati associati dal sistema.

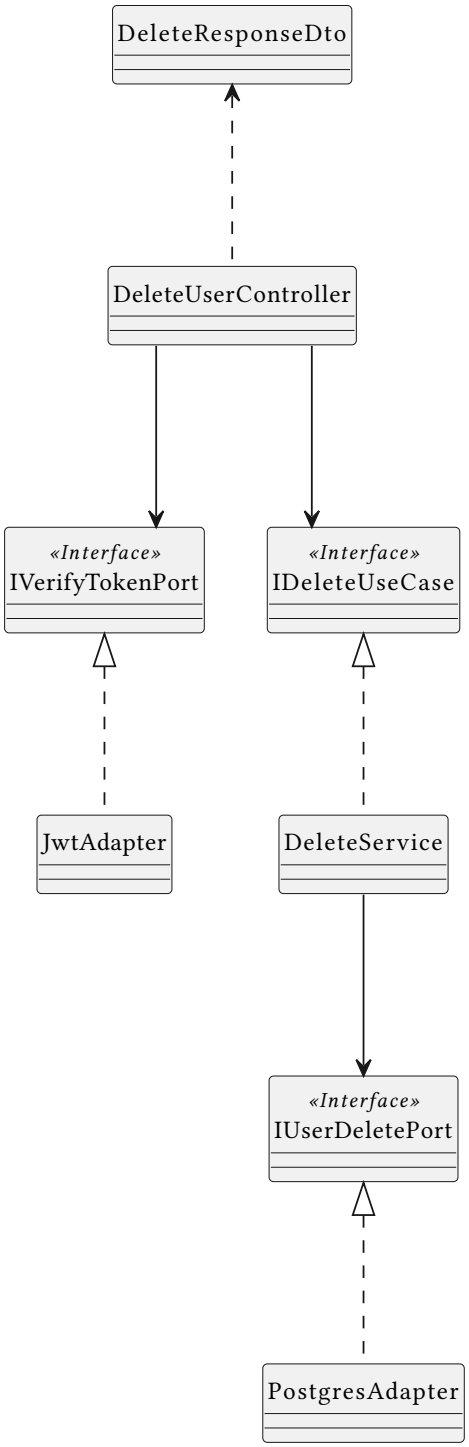

Figure 196: Controller Diagram — DeleteControllerReachableClasses

3.2.2.2 Domain

Il Dominio rappresenta il nucleo centrale dell'architettura esagonale, dove risiedono esclusivamente la logica di business e le regole vitali del progetto. Questa sezione è progettata per essere totalmente agnostica rispetto alla tecnologia: non possiede alcuna conoscenza di database, protocolli di comunicazione (HTTP/REST) o framework esterni.

L'obiettivo del Domain Core è modellare la realtà del problema attraverso un linguaggio comune (*Ubiquitous Language*), garantendo che ogni operazione sia coerente con le aspettative del business.

- **Isolamento Tecnologico:** Il dominio non importa librerie esterne di infrastruttura. Questo garantisce che la logica rimanga testabile in isolamento e protetta dall'obsolescenza dei framework.
- **Integrità e Validazione:** È responsabilità del dominio impedire la creazione di oggetti inconsistenti. Ogni componente (Value Object o Entity) è un "garante" della propria validità.
- **Espressione delle Regole:** Non è un semplice deposito di dati, ma un insieme di componenti attivi che governano i processi (es. il ciclo di vita di un'analisi).

3.2.2.2.1 Value Object

I Value Object rappresentano concetti del dominio definiti esclusivamente dai loro attributi. Sono progettati per essere **immutabili**: una volta istanziati, il loro stato non può subire variazioni, garantendo la thread-safety e la stabilità dei riferimenti durante l'intero ciclo di vita della richiesta. L'uguaglianza tra due Value Object è determinata dal valore delle proprietà incapsulate e non dall'identità dell'istanza in memoria.

3.2.2.2.1.1 UserId

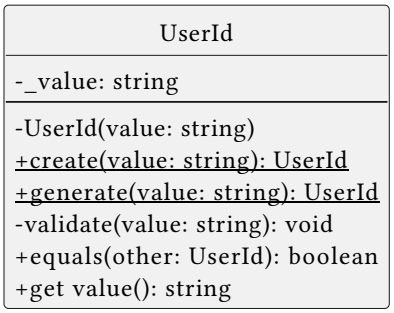

Figure 197: Class Diagram — UserIdAccount

L'identificativo `UserId` costituisce l'atomo di identità dell'utente all'interno del dominio. Esso rappresenta univocamente un registrante nei sistemi di persistenza.

- **Invariante di Formato:** La sua validazione garantisce che l'identificativo sia un UUID versione 7 (v7) formattato correttamente, prevenendo l'introduzione di chiavi primarie invalide o attacchi tramite stringhe malformate.
- **Astrazione della Persistenza:** Disaccoppia la logica di business dall'implementazione fisica della chiave primaria, assicurando che lo strato di dominio comunichi tramite un tipo forte e non attraverso primitive volatili come le stringhe.
- **Prevenzione del Type Mismatch:** Impedisce l'interscambiabilità accidentale con altri identificativi testuali, prevenendo bug che la normale tipizzazione a stringa non riuscirebbe a intercettare.
- **Comparazione Deterministica:** Semplifica e rende sicura l'uguaglianza tra identificatori tramite un metodo centralizzato, garantendo una risoluzione coerente quando gli utenti vengono ricercati o confrontati.

3.2.2.2.1.2 Email

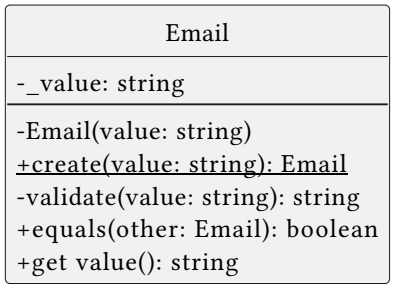

Figure 198: Class Diagram — Email

L'oggetto `Email` incapsula l'indirizzo di posta elettronica dell'utente, fungendo da identificativo principale per le procedure di autenticazione e recupero credenziali.

- **Validazione Formale Sicura:** Assicura che ogni stringa in ingresso sia conforme allo standard degli indirizzi email tramite pattern matching, prevenendo errori o comportamenti inattesi durante l'invio di notifiche o il login.
- **Normalizzazione del Dato:** Gestisce internamente la pulizia della stringa (conversione in minuscolo e rimozione delle spaziature esterne), garantendo un processo di autenticazione indifferente al maiuscolo/minuscolo e riducendo duplicazioni anomale a database.
- **Invariante di Dominio:** Assicurando che non esistano oggetti `Email` nulli o formattati erroneamente, solleva i servizi applicativi e gli adattatori di persistenza dal dover validare ripetutamente il dato, centralizzando la logica di consistenza.

3.2.2.2.1.3 Password

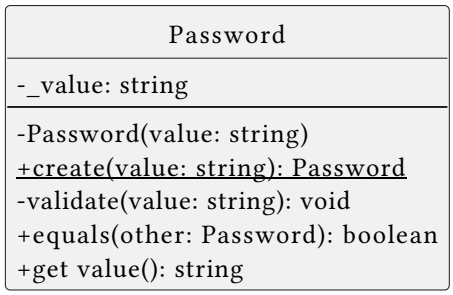

Figure 199: Class Diagram — Password

L'oggetto `Password` rappresenta una password in chiaro nel momento del suo inserimento. Il dominio garantisce che questa istanza sia temporanea e serva esclusivamente per le fasi di controllo qualitativo e crittografico.

- **Enforcement della Complessità:** Verifica rigorosamente le regole di sicurezza e gli standard industriali (minimo 8 caratteri, presenza di maiuscole, minuscole, numeri e caratteri speciali), rigettando password deboli ancor prima che raggiungano gli strati inferiori.
- **Garanzia di Sicurezza Proattiva:** Centralizza la business rule relativa alla robustezza della password. Un cambiamento alle politiche di sicurezza avverrà unicamente in questo contesto, propagandosi automaticamente in ogni punto del sistema.
- **Limitazione dell'Esposizione:** Essendo un oggetto effimero, il suo scopo principale è transitare in modo controllato verso i servizi di crittografia (per la generazione dell'hash) o di comparazione, impedendone l'accidentale salvataggio in chiaro.

3.2.2.2.1.4 PasswordHash

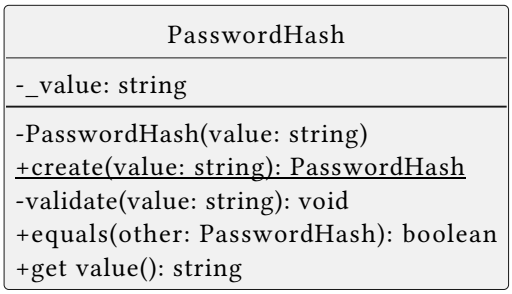

Figure 200: Class Diagram — PasswordHash

L'oggetto `PasswordHash` rappresenta la credenziale cifrata salvata in isolamento e persistita nel sistema. L'infrastruttura di dominio non possiede le chiavi in formati leggibili ma esclusivamente la loro traduzione crittografica sicura.

- **Invariante Crittografica:** Assicura attraverso la validazione che la stringa instanziata sia effettivamente un hash compatibile con lo standard `bcrypt` (identificato dal prefisso `$2a$` o `$2b$`), precludendo il salvataggio o l'utilizzo di testi in chiaro nel posto di un hash.
- **Scudo per la Persistenza:** Costituisce l'unica rappresentazione della password ammessa nel modello persistente, certificando allo strato di database che il dato fornito è già stato processato e validato da un servizio crittografico.
- **Confronto Cifrato Sicuro:** Identifica esplicitamente il dominio di competenza per le collisioni e agevola la comunicazione con i servizi di hashing durante le procedure di login per il ricalcolo e confronto dell'hash reale.

3.2.2.2.2 Entity

A differenza dei Value Object, le Entity sono definite dalla loro **identità** persistente nel tempo e non solo dai loro attributi. Un'Entity mantiene la propria individualità anche se i suoi dati interni subiscono variazioni. Esse incapsulano lo stato e il comportamento del business, garantendo che le transizioni di stato avvengano nel rispetto delle regole del dominio.

- **Identità Univoca:** Ogni Entity è associata a un identificatore immutabile che ne permette la distinzione univoca all'interno del sistema.
- **Ciclo di Vita e Stato:** Le Entity possiedono un ciclo di vita (creazione, modifica, archiviazione) e gestiscono attivamente le proprie mutazioni interne attraverso metodi espliciti.
- **Integrità Comportamentale:** Non si limitano a esporre dati (getter/setter), ma offrono metodi che rappresentano azioni di business, assicurando che l'oggetto passi solo attraverso stati validi e coerenti.

3.2.2.2.2.1 User

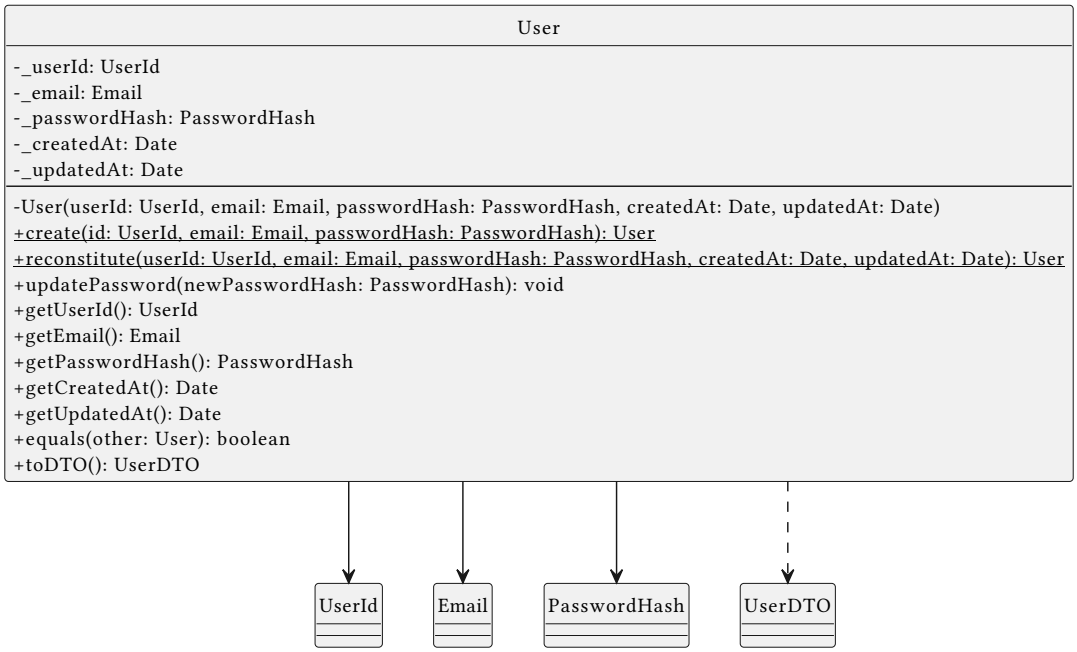

Figure 201: Class Diagram — User

L'entità `User` costituisce l'entità radice del dominio di autenticazione. Essa incapsula l'identità dell'utente (`UserId`), le credenziali di accesso nella loro forma protetta (`Email`, `PasswordHash`) e i metadati temporali di ciclo di vita (`createdAt`, `updatedAt`).

- **Incapsulamento del Ciclo di Vita:** I metodi `create` e `reconstitute` impongono percorsi di costruzione distinti e semanticamente precisi, facilitando la tracciabilità delle operazioni di dominio.
- **Mutazione Controllata:** Il metodo `updatePassword` è l'unico punto di modifica dello stato interno dell'entità, garantendo che ogni cambio di credenziali passi attraverso la logica di dominio e non avvenga mediante accesso diretto ai campi.
- **Proiezione verso il Layer Applicativo:** Il metodo `toDTO` genera una rappresentazione dell'entità esente da dettagli implementativi, permettendo il trasferimento sicuro dei dati verso i layer superiori senza esporre i segreti del dominio.

3.2.2.3 Application

3.2.2.3.1 Commands

3.2.2.3.1.1 DeleteCommand

Figure 202: Class Diagram — DeleteCommand

`DeleteCommand` incapsula il parametro necessario all'eliminazione di un account utente. Trasporta l'identificativo dell'utente da cancellare (`userToDelete`) come stringa non nulla, garantendo che il caso d'uso di cancellazione riceva un riferimento esplicito e non ambiguo al soggetto dell'operazione.

3.2.2.3.1.2 LoginCommand

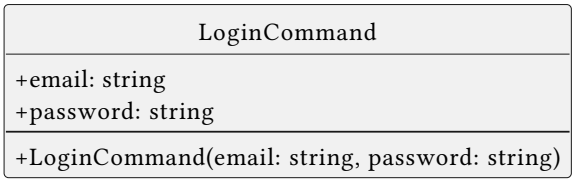

Figure 203: Class Diagram — LoginCommand

`LoginCommand` trasporta le credenziali di autenticazione dell'utente.

- **Accoppiamento Esplicito delle Credenziali:** Il costruttore impone la co-presenza di `email` e `password`, impedendo l'invio di un comando di autenticazione parziale che potrebbe generare comportamenti indefiniti nel servizio sottostante.

3.2.2.3.1.3 LogoutCommand

Figure 204: Class Diagram — LogoutCommand

`LogoutCommand` incapsula il parametro necessario all'invalidazione di una sessione attiva. Il `refreshToken` rappresenta il token di sessione da revocare, identificando univocamente la sessione dell'utente da terminare senza richiederne l'identità diretta.

3.2.2.3.1.4 RegistrationUserCommand

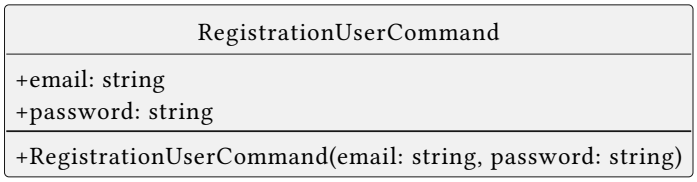

Figure 205: Class Diagram — RegistrationUserCommand

`RegistrationUserCommand` trasporta i dati necessari alla creazione di un nuovo account. Il costruttore esplicito garantisce che email e password siano sempre fornite contestualmente, rendendo impossibile avviare il flusso di registrazione in assenza di uno dei due parametri fondamentali.

3.2.2.3.1.5 UpdateUserCommand

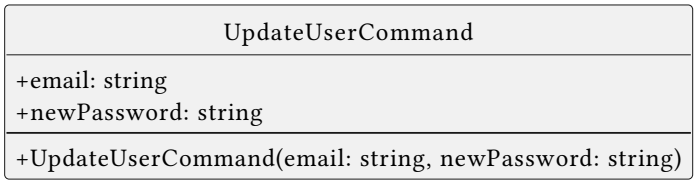

Figure 206: Class Diagram — UpdateUserCommand

`UpdateUserCommand` incapsula i parametri per l'aggiornamento delle credenziali di un utente esistente. Trasporta l'email come identificativo del soggetto e la `newPassword`, separando semanticamente l'aggiornamento dalla creazione.

- **Identificazione Implicita del Soggetto:** L'email funge sia da identificativo per recuperare l'utente dal repository, sia da dato invariante dell'account, riflettendo la decisione di design per cui l'email non è modificabile in questa versione del sistema.

3.2.2.3.2 Application DTOs

3.2.2.3.2.1 AuthResultDto

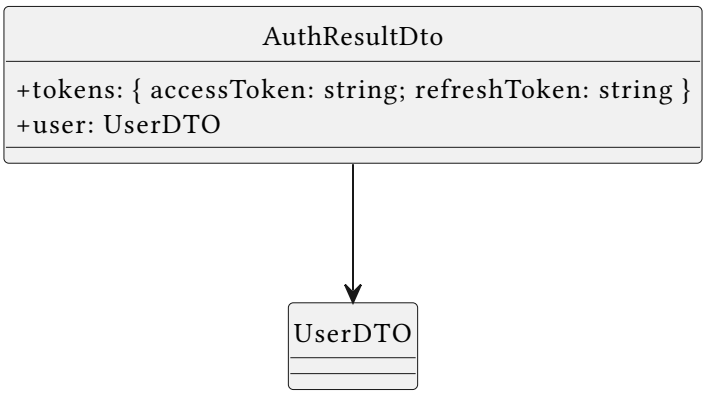

Figure 207: Class Diagram — AuthResultDto

`AuthResultDto` rappresenta il risultato di un'operazione di autenticazione riuscita. Aggrega i token di sessione (`accessToken` e `refreshToken`) con la proiezione dell'utente autenticato (`UserDTO`).

- **Completezza del Contratto di Autenticazione:** La co-presenza di token e dati utente evita una doppia richiesta al backend (una per i token e una per il profilo), ottimizzando il flusso di autenticazione.

3.2.2.3.2.2 JwtPayload

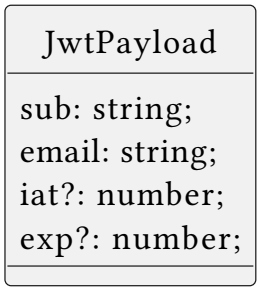

Figure 208: Class Diagram — JwtPayload

`JwtPayload` definisce la struttura del payload del token JWT. Incapsula il `sub` (subject, ovvero l'`userId`), l'`email`, e i campi temporali standard `iat` (issued at) e `exp` (expiration).

- **Contratto Condiviso tra Porte:** Sia `ITokenProviderPort` che `IVerifyTokenPort` dipendono da questo tipo, garantendo che la struttura del payload sia coerente tra il momento della firma e quello della verifica, senza duplicazioni di definizione.
- **Campi Temporali Opzionali:** `iat` ed `exp` sono marcati come opzionali (`?`) poiché possono essere aggiunti dalla libreria JWT durante la firma e non devono essere necessariamente presenti nel payload in input alla generazione.

3.2.2.3.2.3 UserDTO

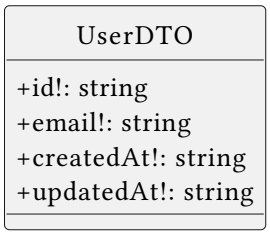

Figure 209: Class Diagram — UserDTO

`UserDTO` è la proiezione del dominio utente destinata alla comunicazione interna tra i livelli applicativo e di presentazione. Espone i soli campi necessari alle operazioni di lettura (`id`, `email`, `createdAt`, `updatedAt`), occultando i dettagli sensibili come `PasswordHash`.

3.2.2.3.3 Exceptions

3.2.2.3.3.1 InvalidCredentialsException

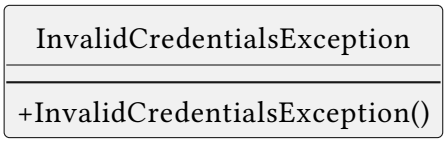

Figure 210: Class Diagram — InvalidCredentialsException

`InvalidCredentialsException` è l'eccezione sollevata dal `LoginService` quando la combinazione email/password fornita non corrisponde a nessun account valido nel sistema. Il costruttore senza parametri formalizza un errore di business che non richiede dettagli aggiuntivi: l'unica informazione rilevante è che le credenziali sono invalide.

3.2.2.3.4 Ports

3.2.2.3.4.1 IHashComparePort

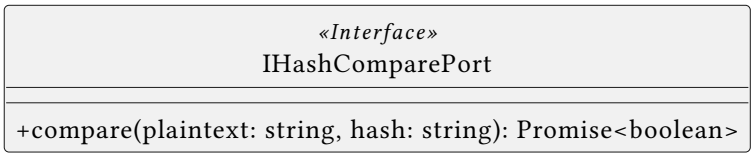

Figure 211: Class Diagram — IHashComparePort

`IHashComparePort` definisce il contratto per la verifica di una password in chiaro rispetto a un hash crittografico. Il metodo `compare(plaintext, hash)` restituisce una `Promise<boolean>`, astraendo l'algoritmo di hashing effettivamente utilizzato dall'interno del Core applicativo.

- **Inversione della Dipendenza:** Il `LoginService` dipende da questa interfaccia e non da una specifica implementazione, rendendo possibile la sostituzione dell'algoritmo di hashing senza modificare la logica di business.
- **Testabilità:** In fase di test unitario, questa porta può essere sostituita da un'implementazione mock che restituisce `true` o `false` in modo deterministico, isolando il `LoginService` dall'overhead computazionale del vero algoritmo crittografico.

3.2.2.3.4.2 IHashPasswordPort

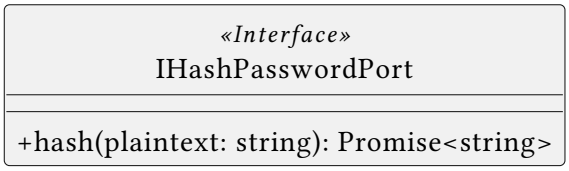

Figure 212: Class Diagram — IHashPasswordPort

`IHashPasswordPort` definisce il contratto per la trasformazione di una password in chiaro nel suo hash crittografico. Il metodo `hash(plaintext)` restituisce una `Promise<string>`, rendendo il Core indipendente dall'algoritmo di hashing e dalla sua configurazione.

- **Separazione delle Responsabilità:** La porta di hashing è distinta da quella di confronto (`IHashComparePort`), poiché si tratta di operazioni semanticamente diverse usate in contesti diversi (registrazione vs login), con diversi use case come dipendenti.
- **Sicurezza by Design:** Incapsulare l'hashing in una porta formalizza la regola che nessuna password deve mai essere salvata in chiaro nel sistema, rendendo questo vincolo di sicurezza esplicito nell'architettura.

3.2.2.3.4.3 ISessionDeletePort

Figure 213: Class Diagram — ISessionDeletePort

`ISessionDeletePort` definisce il contratto per la revoca di una sessione attiva. Il metodo `deleteSession(refreshToken)` richiede il token di sessione come identificativo, delegando all'adattatore la ricerca e l'eliminazione del record corrispondente nel layer di persistenza.

- **Semantica del Logout:** L'uso del `refreshToken` come parametro riflette la decisione architetturale di trattare la sessione come un'entità identificata dal token e non dall'utente, permettendo il logout selettivo in scenari multi-sessione.
- **Isolamento dalla Persistenza:** Il Core non ha conoscenza del meccanismo di archiviazione delle sessioni; tale dettaglio è completamente nascosto dall'adattatore che implementa questa porta.

3.2.2.3.4.4 ISessionSavePort

Figure 214: Class Diagram — ISessionSavePort

`ISessionSavePort` definisce il contratto per la creazione e il salvataggio di una nuova sessione. Il metodo `saveSession(userId, refreshToken, expiresAt)` riceve i tre parametri essenziali che caratterizzano una sessione: il soggetto, il token di accesso rinnovabile e la scadenza.

- **Completezza del Contratto di Sessione:** I tre parametri riflettono i requisiti minimi per una gestione sicura delle sessioni: l'`expiresAt` permette la scadenza automatica e il `refreshToken` identifica univocamente la sessione per operazioni future.

- **Disaccoppiamento dallo Storage:** Il Core delega completamente all'adattatore la scelta di dove e come persistere la sessione.

3.2.2.3.4.5 ITokenProviderPort

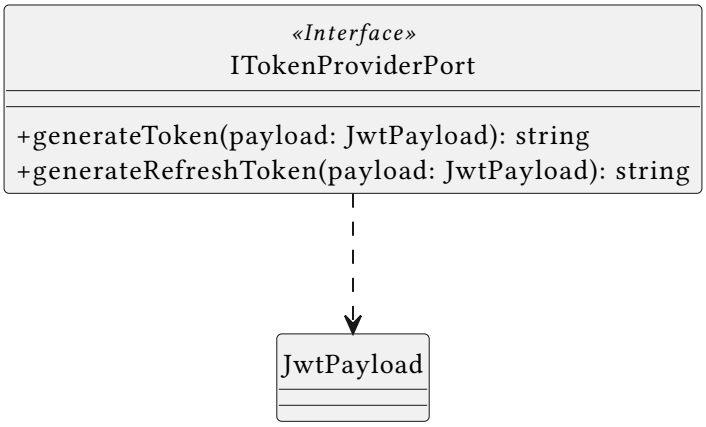

Figure 215: Class Diagram — ITokenProviderPort

`ITokenProviderPort` definisce il contratto per la generazione di token di autenticazione. Il metodo `generateToken` produce il token di accesso a breve scadenza, mentre `generateRefreshToken` produce il token di rinnovo a lunga scadenza, entrambi a partire da un `JwtPayload`.

- **Separazione dei Tipi di Token:** L'esistenza di due metodi distinti riflette la differente semantica e configurazione dei due tipi di token, rendendo esplicita nell'architettura la distinzione tra access token e refresh token.
- **Indipendenza dalla Libreria JWT:** Il Core non importa direttamente librerie come `jsonwebtoken`; la generazione è delegata all'adattatore `JwtAdapter`, che può essere sostituito con qualsiasi altra implementazione compatibile con il contratto.

3.2.2.3.4.6 IUserDeletePort

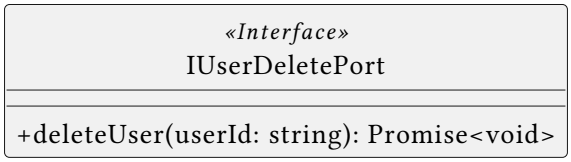

Figure 216: Class Diagram — IUserDeletePort

`IUserDeletePort` definisce il contratto per la rimozione permanente di un utente dal sistema. Il metodo `deleteUser(userId)` utilizza l'identificativo come unico parametro, separando semanticamente l'eliminazione dell'utente dalla cancellazione delle sue sessioni (gestita da `ISessionDeletePort`).

- **Granularità delle Operazioni:** La separazione tra la porta di cancellazione dell'utente e quella della sessione consente al `DeleteService` di orchestrare la rimozione in più fasi, o di implementare soft delete senza modificare i contratti.
- **Atomicità Delegata:** La gestione dell'atomicità dell'operazione è responsabilità dell'adattatore o dello strato di infrastruttura, non del Core.

3.2.2.3.4.7 IUserFindPort

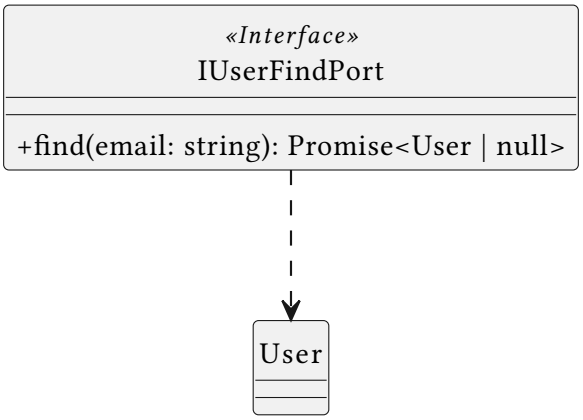

Figure 217: Class Diagram — IUserFindPort

`IUserFindPort` definisce il contratto per il recupero di un'entità `User` dalla persistenza tramite email. La firma `find(email): Promise<User | null>` comunica esplicitamente che l'utente potrebbe non esistere, obbligando i chiamanti a gestire il caso di assenza senza ricorrere a eccezioni per il controllo di flusso ordinario.

- **Return Type Esplicito del "Not Found":** Il tipo di ritorno `User | null` è preferito alla propagazione di un'eccezione per l'assenza dell'utente, distinguendo a livello di tipo tra un errore operativo e un esito atteso ma negativo della ricerca.

3.2.2.3.4.8 IUserSavePort

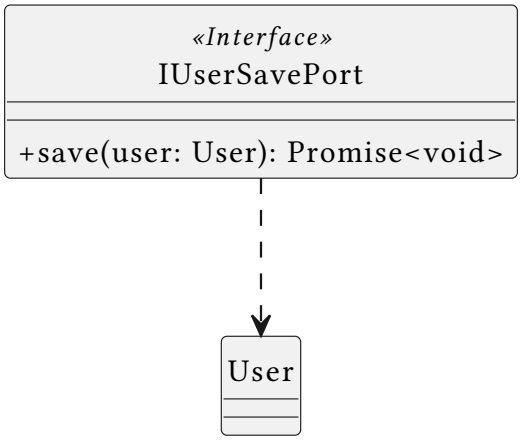

Figure 218: Class Diagram — IUserSavePort

`IUserSavePort` definisce il contratto per la persistenza di una nuova entità `User`. Il metodo `save(user)` riceve l'intera entità di dominio, delegando all'adattatore la traduzione nel formato specifico del database (es. record SQL).

- **Accoppiamento all'Entità di Dominio:** A differenza delle porte che accettano primitive, questa porta riceve un oggetto `User` completo, garantendo che solo entità coerenti e già validate dalla logica di dominio possano essere persistite.
- **Separazione da Update:** L'esistenza di porte distinte per `save` e `update` (tramite `IUserUpdatePort`) permette all'adattatore di distinguere tra un'operazione `INSERT` e un `UPDATE` a livello di database, ottimizzando le query sottostanti.

3.2.2.3.4.9 IUserUpdatePort

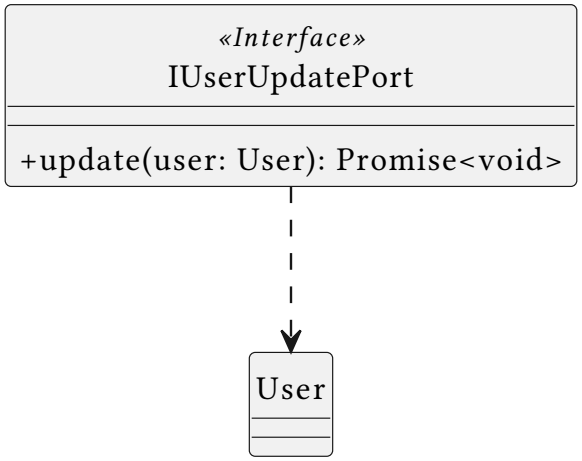

Figure 219: Class Diagram — IUserUpdatePort

`IUserUpdatePort` definisce il contratto per la modifica di un'entità `User` già esistente nel sistema. Il metodo `update(user)` riceve l'entità aggiornata, lasciando all'adattatore la responsabilità di determinare quali campi modificare e come gestire la transazione.

- **Gestione dell'`updatedAt` Delegata:** Sebbene l'entità `User` gestisca il campo `updatedAt`, la porta permette all'adattatore di aggiornarlo a livello di database, garantendo la coerenza temporale della persistenza.

3.2.2.3.4.10 IVerifyTokenPort

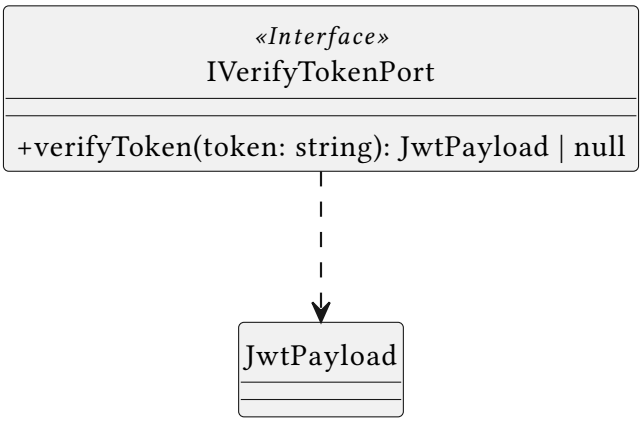

Figure 220: Class Diagram — IVerifyTokenPort

`IVerifyTokenPort` definisce il contratto per la verifica e il parsing di un token JWT. Il metodo `verifyToken(token)` restituisce il `JwtPayload` estratto se il token è valido, o `null` se la verifica fallisce (token scaduto, firma non valida, ecc.), evitando l'uso di eccezioni per scenari di token non validi.

- **Return Type Null-Safe:** Il tipo di ritorno `JwtPayload | null` comunica esplicitamente che un token non valido è un esito atteso, semplificando la gestione nel controller che usa questa porta.
- **Co-Localizzazione con `ITokenProviderPort`:** Il fatto che `JwtAdapter` implementi sia la generazione che la verifica dei token, ma che le due capacità siano esposte come porte distinte, permette di iniettare solo la capacità necessaria nei diversi use case, rispettando il principio di minimo privilegio.

3.2.2.3.5 Services

3.2.2.3.5.1 DeleteService

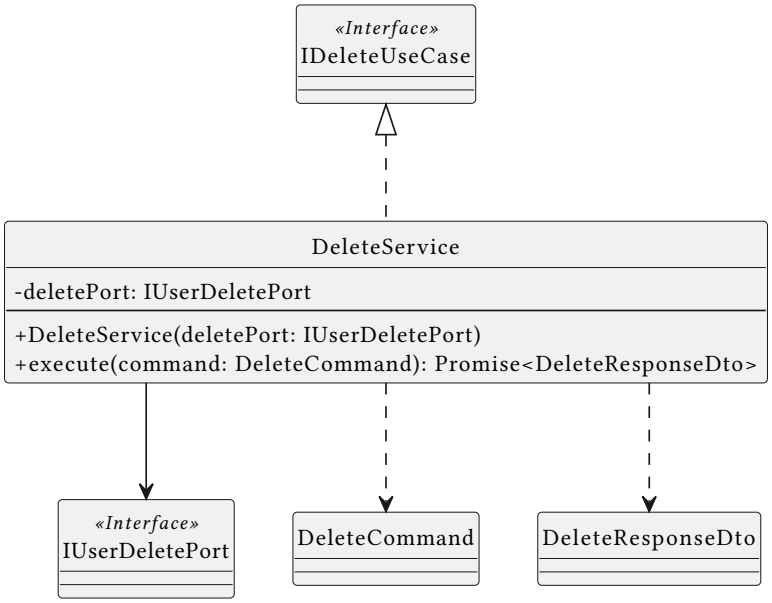

Figure 221: Class Diagram — DeleteService

`DeleteService` implementa il caso d'uso di eliminazione dell'account. Inietta `IUserDeletePort` tramite costruttore e coordina la cancellazione dell'utente. Implementa `IDeleteUseCase` e restituisce un `DeleteResponseDto` per confermare l'esito dell'operazione.

- **Orchestrazione Minima:** La logica del servizio è deliberatamente semplice: recupera il comando, delega la cancellazione alla porta e costruisce la risposta. La complessità transazionale (es. eliminare prima le sessioni) può essere gestita a livello di adattatore o aggiungendo dipendenze da `ISessionDeletePort` in evoluzioni future.
- **Implementazione del Contratto:** Implementando `IDeleteUseCase`, il servizio garantisce che il controller dipenda dall'interfaccia e non dalla classe concreta, mantenendo l'invertibilità della dipendenza.

3.2.2.3.5.2 LoginService

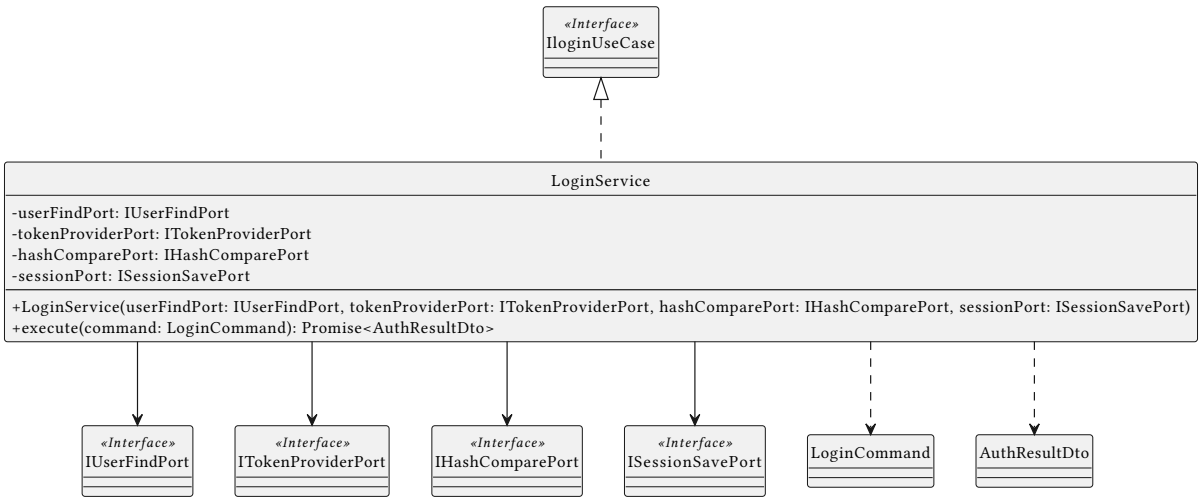

Figure 222: Class Diagram — LoginService

`LoginService` implementa il caso d'uso di autenticazione. Coordina quattro porte: recupera l'utente tramite `IUserFindPort`, verifica la password con `IHashComparePort`, genera i token con `ITokenProviderPort` e persiste la sessione con `ISessionSavePort`. In caso di credenziali invalide solleva `InvalidCredentialsException`.

- **Orchestrazione Multi-Porta:** L'elevato numero di dipendenze riflette la complessità intrinseca del flusso di autenticazione, che richiede la cooperazione di più capacità infrastrutturali.
- **Short-Circuit in Caso di Errore:** Il servizio interrompe il flusso non appena le credenziali risultano invalide, sollevando `InvalidCredentialsException` prima di procedere alla generazione dei token, minimizzando le operazioni eseguite a fronte di un tentativo fallito.

3.2.2.3.5.3 LogoutService

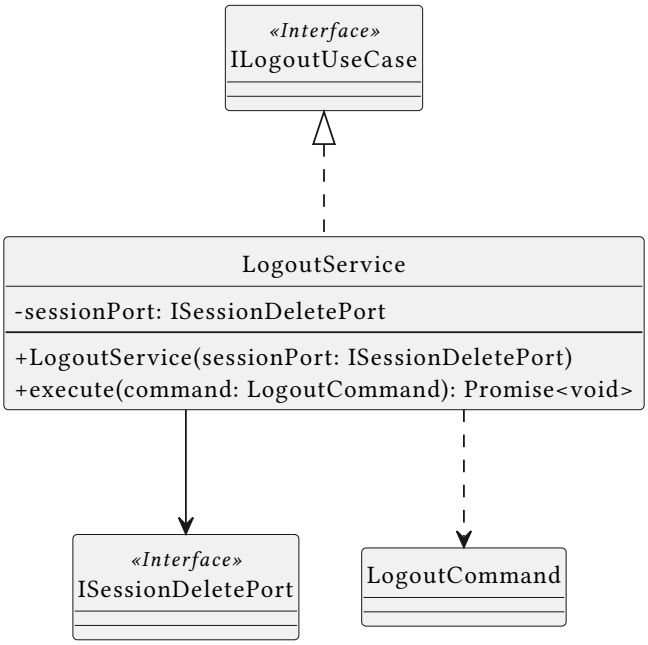

Figure 223: Class Diagram — LogoutService

`LogoutService` implementa il caso d'uso di chiusura della sessione. Inietta `ISessionDeletePort` e invoca `deleteSession` con il `refreshToken` estratto dal `LogoutCommand`. L'operazione non restituisce dati applicativi rilevanti (ritorno `void`).

- **Semplicità Intenzionale:** La singola dipendenza del servizio riflette la natura atomica dell'operazione di logout, che si riduce alla revoca di un token senza effetti collaterali sul profilo utente.
- **Idempotenza Implicita:** L'eliminazione di un token già revocato o inesistente è gestita a livello di adattatore.

3.2.2.3.5.4 RegistrationService

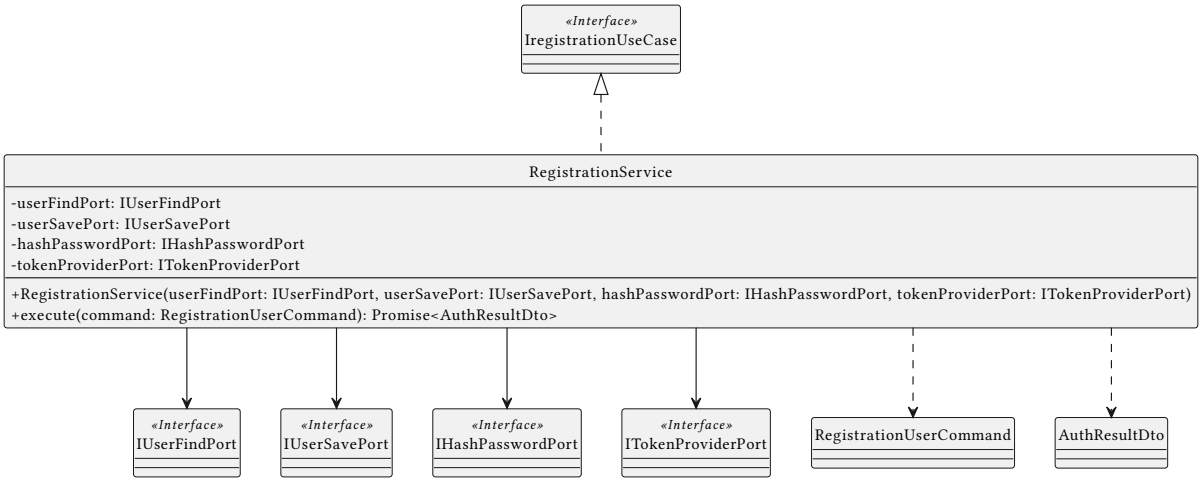

Figure 224: Class Diagram — RegistrationService

`RegistrationService` implementa il caso d'uso di creazione di un nuovo account. Coordina `IUserFindPort` (verifica unicità dell'email), `IHashPasswordPort` (hashing della password), `IUserSavePort` (persistenza dell'utente) e `ITokenProviderPort` (generazione dei token post-registrazione), restituendo un `AuthResultDto` completo.

- **Verifica di Unicità Pre-Creazione:** Il servizio verifica che l'email non sia già registrata prima di procedere con hashing e salvataggio, proteggendo l'invariante di unicità dell'account a livello applicativo prima ancora che il database possa sollevare un constraint error.
- **Registrazione e Login Unificati:** La restituzione di un `AuthResultDto` completo (con token) al termine della registrazione riflette la scelta di UX di autenticare automaticamente l'utente al termine del processo di registrazione, eliminando un secondo round-trip di login.

3.2.2.3.5.5 UpdateService

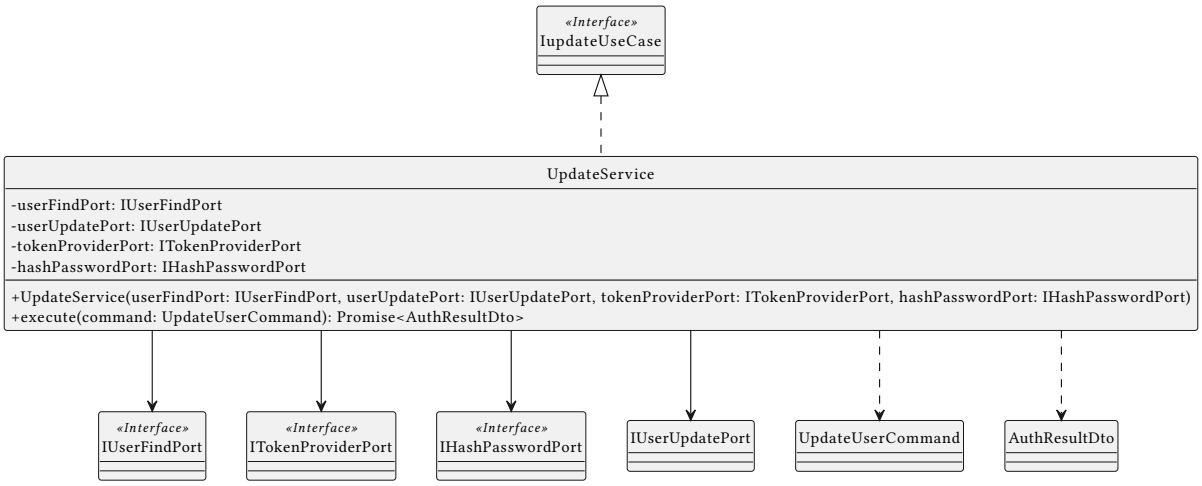

Figure 225: Class Diagram — UpdateService

`UpdateService` implementa il caso d'uso di aggiornamento delle credenziali. Coordina `IUserFindPort` (recupero dell'utente esistente), `IHashPasswordPort` (hashing della nuova password), `IUserUpdatePort` (persistenza della modifica) e `ITokenProviderPort` (generazione di nuovi token post-aggiornamento), restituendo un `AuthResultDto` aggiornato.

- **Rinnovo dei Token Post-Update:** La restituzione di nuovi token dopo l'aggiornamento della password riflette la pratica di sicurezza di invalidare le sessioni precedenti dopo un cambio di credenziali, forzando la ri-autenticazione su tutti i dispositivi.
- **Recupero dell'Entità Prima della Modifica:** Il servizio recupera l'utente tramite email prima di applicare la modifica, garantendo che l'aggiornamento avvenga su un'entità già esistente e coerente con lo stato attuale del dominio.

3.2.2.3.6 Use Cases

3.2.2.3.6.1 IDeleteUseCase

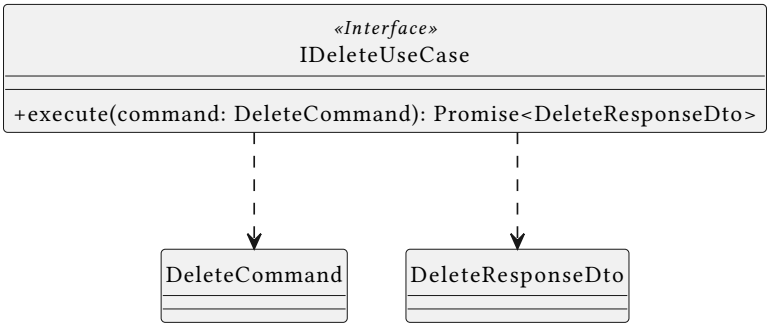

Figure 226: Class Diagram — IDeleteUseCase

`IDeleteUseCase` definisce il contratto del caso d'uso di cancellazione account. Il metodo `execute(command: DeleteCommand): Promise<DeleteResponseDto>` standardizza la firma del flusso di eliminazione, permettendo al `DeleteUserController` di invocare l'operazione senza conoscere l'implementazione concreta del servizio.

3.2.2.3.6.2 IloginUseCase

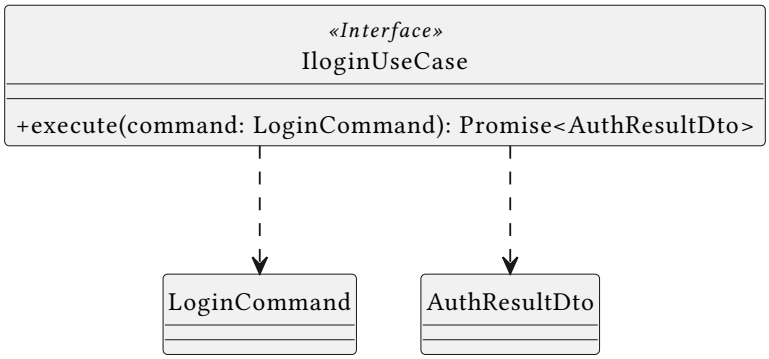

Figure 227: Class Diagram — IloginUseCase

`IloginUseCase` definisce il contratto del caso d'uso di autenticazione. Il metodo `execute(command: LoginCommand): Promise<AuthResultDto>` standardizza la firma del flusso di login, disaccoppiando il `LoginController` dall'implementazione concreta del `LoginService`.

3.2.2.3.6.3 ILogoutUseCase

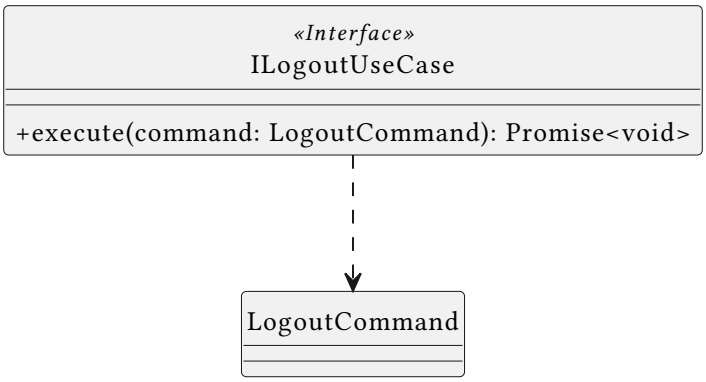

Figure 228: Class Diagram — ILogoutUseCase

`ILogoutUseCase` definisce il contratto del caso d'uso di logout. Il metodo `execute(command: LogoutCommand): Promise<void>` standardizza la firma del flusso di chiusura sessione, rendendo il `LogoutController` indipendente dalla concreta implementazione del `LogoutService`.

3.2.2.3.6.4 IregistrationUseCase

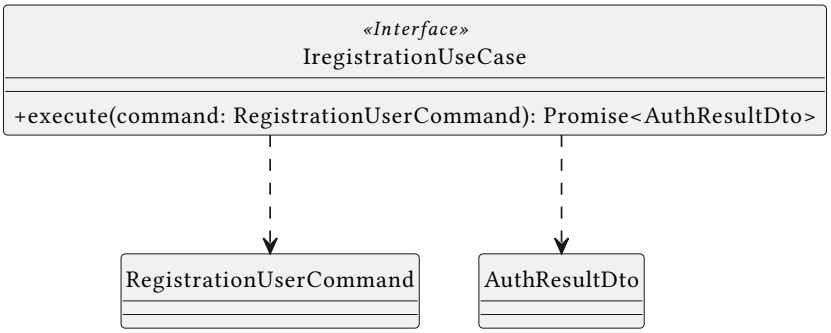

Figure 229: Class Diagram — IregistrationUseCase

`IregistrationUseCase` definisce il contratto del caso d'uso di registrazione. Il metodo `execute(command: RegistrationUserCommand): Promise<AuthResultDto>` standardizza la firma del flusso di creazione account, disaccoppiando il `RegistrationController` dall'implementazione concreta del `RegistrationService`.

3.2.2.3.6.5 IupdateUseCase

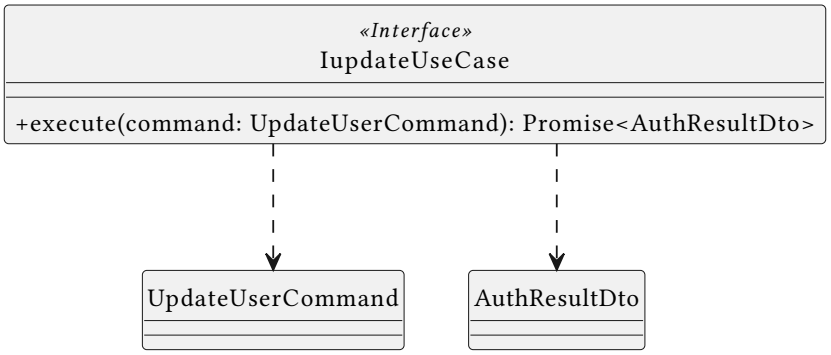

Figure 230: Class Diagram — IupdateUseCase

`IupdateUseCase` definisce il contratto del caso d'uso di aggiornamento credenziali. Il metodo `execute(command: UpdateUserCommand): Promise<AuthResultDto>` standardizza la firma del

flusso di modifica password, rendendo l'`UpdateController` indipendente dall'implementazione concreta dell'`UpdateService`.

3.2.2.4 Infrastructure

3.2.2.4.1 Adapters

3.2.2.4.1.1 BcryptAdapter

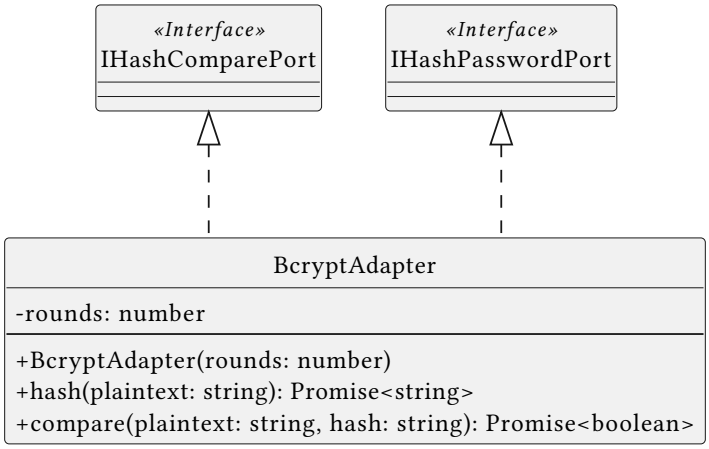

Figure 231: Class Diagram — BcryptAdapter

`BcryptAdapter` è l'adattatore Driven che implementa sia `IHashPasswordPort` sia `IHashComparePort`, fornendo le operazioni crittografiche di hashing e verifica delle password tramite l'algoritmo bcrypt. Il campo `rounds` configura il fattore di costo dell'algoritmo, bilanciando sicurezza e performance.

- **Implementazione Doppia Porta:** Il fatto che un singolo adattatore implementi due porte distinte è una scelta pragmatica: bcrypt è l'algoritmo comune a entrambe le operazioni, e separare le implementazioni non apporterebbe vantaggi architetturali. Le porte rimangono comunque distinte, consentendo di iniettarne solo una nei servizi che ne necessitano.
- **Configurabilità del Fattore di Costo:** Il campo `rounds` permette di calibrare il fattore di costo di bcrypt in base all'ambiente (es. più basso nei test per ridurre la latenza, più alto in produzione per aumentare la resistenza agli attacchi brute-force).
- **Operazioni Asincrone:** I metodi `hash` e `compare` restituiscono `Promise`, riflettendo la natura computazionalmente intensa di bcrypt e la necessità di non bloccare il thread dell'event loop di Node.js durante l'esecuzione.

3.2.2.4.1.2 JwtAdapter

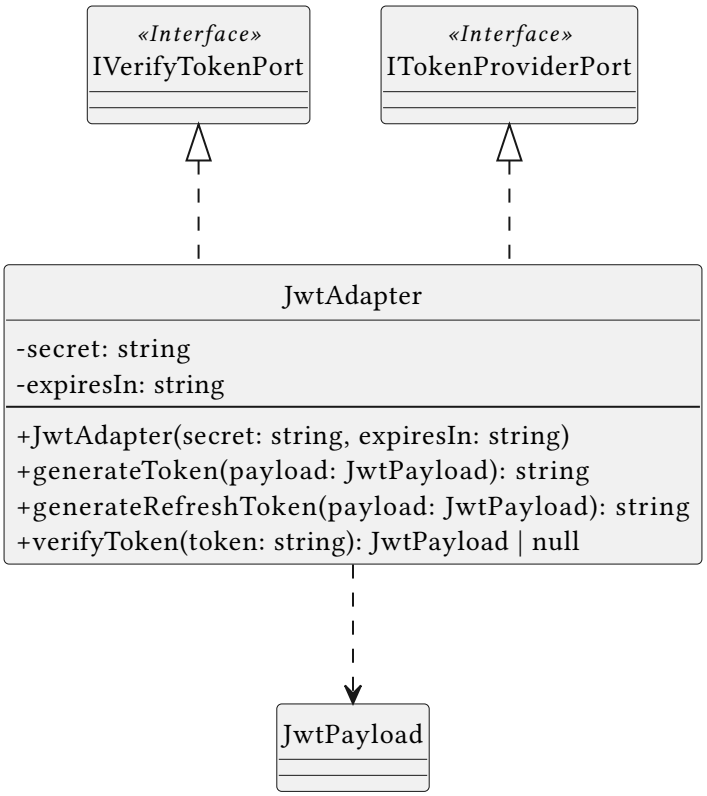

Figure 232: Class Diagram — JwtAdapter

`JwtAdapter` è l'adattatore Driven che implementa sia `ITokenProviderPort` sia `IVerifyTokenPort`, gestendo la generazione e la verifica dei token JWT tramite la configurazione di `secret` ed `expiresIn`. Dipende dal tipo `JwtPayload` per garantire la coerenza strutturale del payload.

- **Implementazione Doppia Porta:** Analogamente a `BcryptAdapter`, l'implementazione di due porte in un singolo adattatore è motivata dalla coerenza: la stessa chiave segreta e la stessa configurazione sono necessarie sia per firmare che per verificare i token.
- **Configurabilità Centralizzata:** I campi `secret` ed `expiresIn` centralizzano la configurazione dei token, rendendo semplice la sostituzione dei valori tramite variabili d'ambiente senza modificare la logica del servizio.
- **Gestione del Fallimento di Verifica:** Il metodo `verifyToken` restituisce `null` in caso di token non valido (anziché propagare un'eccezione), trasferendo la responsabilità di gestire l'assenza di un payload valido al chiamante in modo esplicito e sicuro.

3.2.2.4.1.3 PostgresAdapter

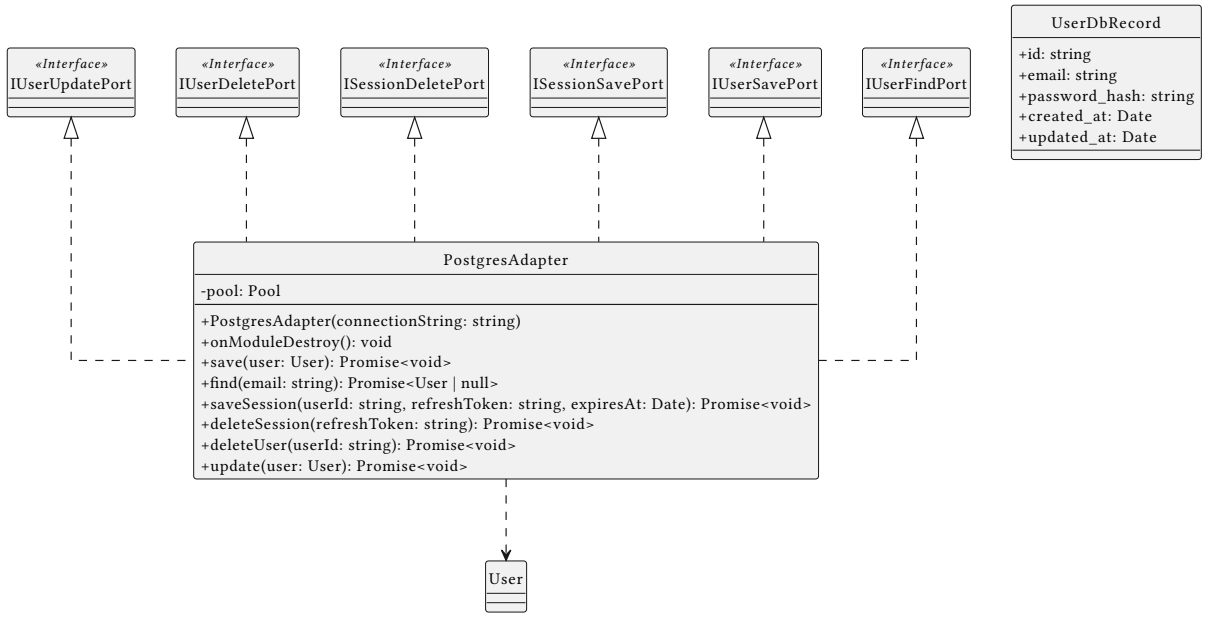

Figure 233: Class Diagram — PostgresAdapter

`PostgresAdapter` è l'adattatore Driven principale del microservizio Account. Implementa sei porte: `IUserFindPort`, `IUserSavePort`, `IUserUpdatePort`, `IUserDeletePort`, `ISessionSavePort` e `ISessionDeletePort`, centralizzando tutta la comunicazione con il database PostgreSQL tramite un `Pool` di connessioni. L'interfaccia interna `UserDbRecord` definisce la forma del record nel database.

- **Aggregazione delle Porte di Persistenza:** La scelta di implementare tutte le porte di accesso ai dati in un unico adattatore riflette la coerenza della sorgente dati sottostante: operazioni su utenti e sessioni condividono la stessa connessione al database, semplificando la gestione delle transazioni e della coerenza.
- **Gestione del Pool di Connessioni:** L'uso di un `Pool` anziché di connessioni singole garantisce performance e resilienza in scenari concorrenti, delegando al pool la gestione del ciclo di vita delle connessioni.
- **`onModuleDestroy` per la Pulizia:** L'implementazione del lifecycle hook `onModuleDestroy` garantisce che il pool di connessioni venga chiuso correttamente allo spegnimento del modulo, prevenendo resource leak in ambienti di deployment containerizzati.
- **`UserDbRecord` come Contratto di Mapping:** L'interfaccia interna `UserDbRecord` definisce la forma esatta del record nel database, separando la struttura di persistenza dall'entità di dominio `User` e centralizzando la logica di mapping in un unico punto.

3.2.2.5 Presentation

3.2.2.5.1 Controllers

3.2.2.5.1.1 DeleteUserController

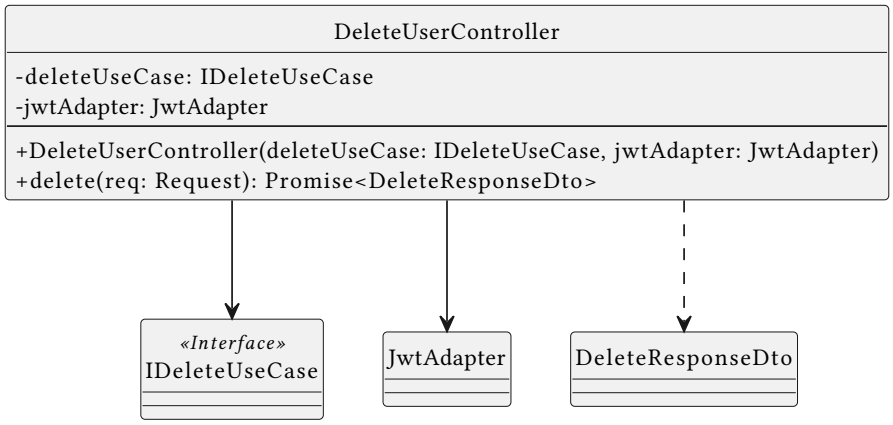

Figure 234: Class Diagram — DeleteUserController

`DeleteUserController` espone l'endpoint HTTP per la cancellazione dell'account. Inietta `IDeleteUseCase` per l'esecuzione del flusso di business e `JwtAdapter` per l'estrazione dell'identità dell'utente dal token JWT presente nella richiesta, restituendo un `DeleteResponseDto`.

- **Estrazione dell'Identità dal Token:** La dipendenza da `JwtAdapter` nel controller riflette la necessità di identificare il soggetto della cancellazione dal token di autenticazione incluso nella richiesta, senza richiedere all'utente di fornire esplicitamente il proprio ID nel body.
- **Delegazione al Use Case:** Il controller non contiene logica di business; si limita a costruire il `DeleteCommand` con le informazioni estratte dalla richiesta e a passarlo al caso d'uso, rispettando il principio di singola responsabilità.

3.2.2.5.1.2 LoginController

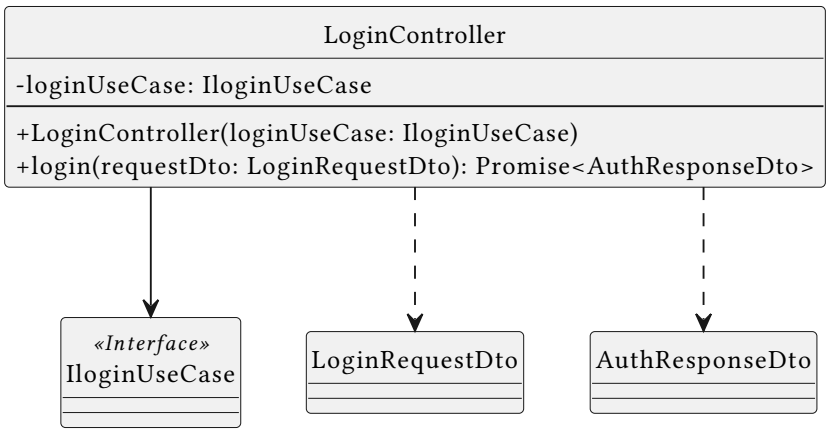

Figure 235: Class Diagram — LoginController

`LoginController` espone l'endpoint HTTP di autenticazione. Inietta `IloginUseCase`, costruisce un `LoginCommand` dal `LoginRequestDto` ricevuto nel body della richiesta e restituisce un `AuthResponseDto` in caso di successo.

- **Dipendenza dall'Interfaccia:** La dipendenza da `IloginUseCase` anziché da `LoginService` garantisce che il controller possa essere testato con un mock dell'interfaccia senza dover istanziare l'intera catena di dipendenze del servizio.

3.2.2.5.1.3 LogoutController

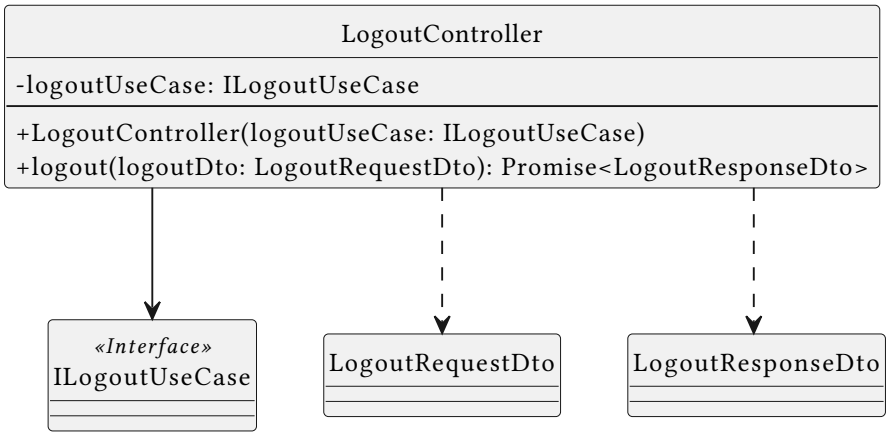

Figure 236: Class Diagram — LogoutController

`LogoutController` espone l'endpoint HTTP di chiusura sessione. Inietta `ILogoutUseCase`, costruisce un `LogoutCommand` dal `LogoutRequestDto` e invoca il caso d'uso, restituendo un `LogoutResponseDto`.

- **Dipendenza dall'Interfaccia:** La dipendenza da `ILogoutUseCase` anziché da `LogoutService` garantisce che il controller possa essere testato con un mock dell'interfaccia senza dover istanziare l'intera catena di dipendenze del servizio.

3.2.2.5.1.4 RegistrationController

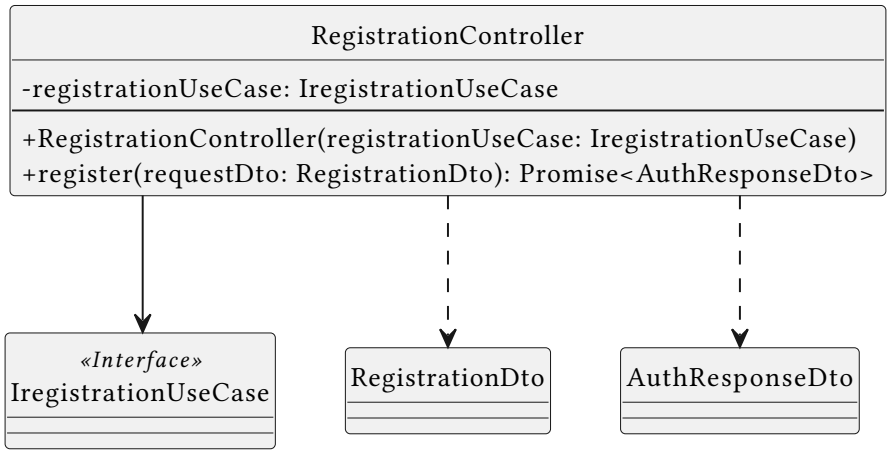

Figure 237: Class Diagram — RegistrationController

`RegistrationController` espone l'endpoint HTTP di creazione account. Inietta `IregistrationUseCase`, costruisce un `RegistrationUserCommand` dal `RegistrationDto` ricevuto nel body e restituisce un `AuthResponseDto` completo di token e profilo utente.

- **Registrazione e Autenticazione Contestuale:** La restituzione di un `AuthResponseDto` (contenente i token) al termine della registrazione riflette la scelta UX di autenticare l'utente immediatamente dopo la creazione dell'account.

3.2.2.5.1.5 UpdateController

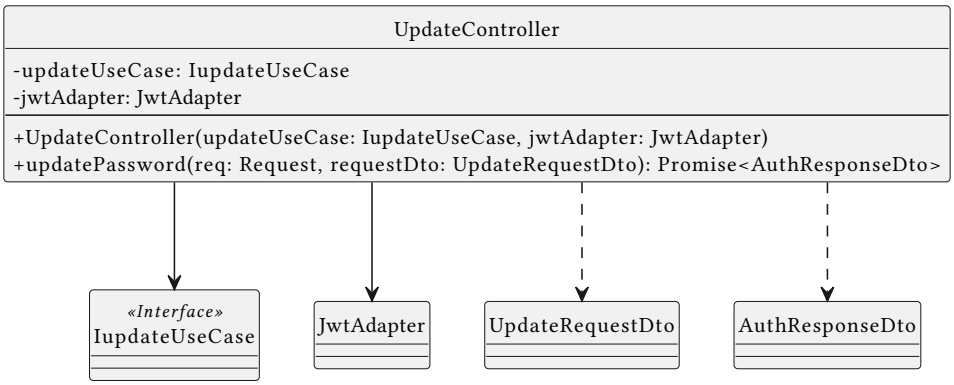

Figure 238: Class Diagram — UpdateController

`UpdateController` espone l'endpoint HTTP di aggiornamento credenziali. Inietta sia `IupdateUseCase` per l'esecuzione del caso d'uso, sia `JwtAdapter` per estrarre l'email dell'utente autenticato dal token JWT nella richiesta, costruendo l'`UpdateUserCommand` con i dati combinati di richiesta e identità.

- **Identità dall'Autenticazione:** La dipendenza da `JwtAdapter` permette di estrarre l'email dell'utente dal token di sessione, evitando che il client debba includere la propria identità nel body della richiesta e proteggendo da attacchi di impersonation.
- **Costruzione del Comando Arricchito:** Il controller combina le informazioni del `UpdateRequestDto` (nuova password) con quelle estratte dal token (email), producendo un `UpdateUserCommand` completo prima di delegare al use case.

3.2.2.5.2 Presentation DTOs

3.2.2.5.2.1 LoginRequestDto

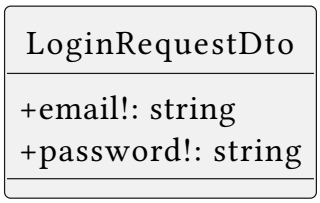

Figure 239: Class Diagram — LoginRequestDto

`LoginRequestDto` definisce il contratto del body della richiesta HTTP di login. I campi `email` e `password` sono entrambi obbligatori, garantendo che il framework di validazione (es. `class-validator`) rifiuti le richieste incomplete prima che raggiungano il controller.

3.2.2.5.2.2 LogoutRequestDto

Figure 240: Class Diagram — LogoutRequestDto

`LogoutRequestDto` definisce il contratto del body della richiesta HTTP di logout. Il campo `refreshToken` trasporta il token di sessione da revocare, che il controller utilizzerà per costruire il `LogoutCommand`.

3.2.2.5.2.3 RegistrationDto

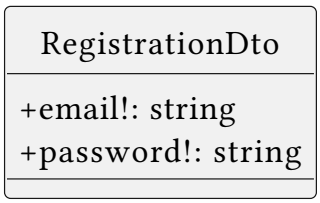

Figure 241: Class Diagram — RegistrationDto

`RegistrationDto` definisce il contratto del body della richiesta HTTP di registrazione. I campi `email` e `password` sono obbligatori, rispecchiando i requisiti minimi necessari alla creazione di un nuovo account nel sistema.

3.2.2.5.2.4 UpdateRequestDto

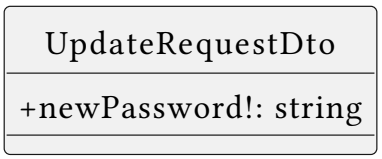

Figure 242: Class Diagram — UpdateRequestDto

`UpdateRequestDto` definisce il contratto del body della richiesta HTTP di aggiornamento credenziali. Il solo campo obbligatorio `newPassword` riflette la scelta di non richiedere al client di includere la propria identità nel body.

3.2.2.5.2.5 AuthResponseDto e UserResponseDto

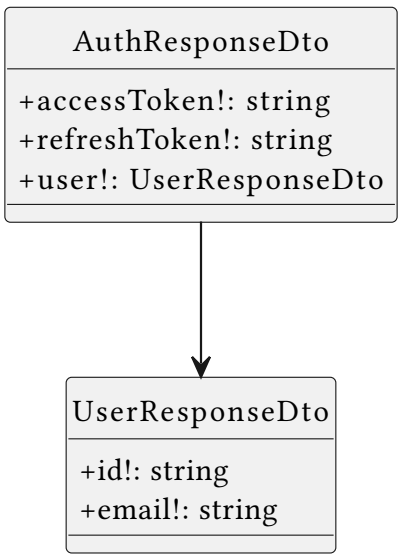

Figure 243: Class Diagram — AuthResponseDto

`AuthResponseDto` definisce il contratto della risposta HTTP per le operazioni di autenticazione. Aggrega i token di sessione (`accessToken`, `refreshToken`) con la proiezione ridotta dell'utente tramite `UserResponseDto`, che espone solo `id` ed `email`.

- **Proiezione Minima dell'Utente:** `UserResponseDto` espone meno campi di `UserDTO` (omette `createdAt` e `updatedAt`), riflettendo la necessità del client di disporre dell'identità dell'utente autenticato senza sovraccaricare la risposta con metadati non essenziali al flusso di autenticazione.
- **Contratto Stabile verso il Client:** La forma di `AuthResponseDto` costituisce il contratto pubblico del microservizio per le operazioni di autenticazione.

3.2.2.5.2.6 DeleteResponseDto

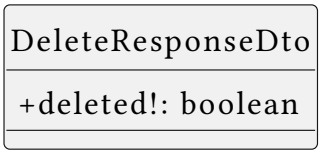

Figure 244: Class Diagram — DeleteResponseDto

`DeleteResponseDto` definisce la risposta HTTP per l'operazione di cancellazione account. Il campo booleano `deleted` fornisce una conferma esplicita e tipizzata dell'esito dell'operazione, permettendo al client di distinguere tra un'eliminazione avvenuta con successo e un esito negativo senza dover interpretare esclusivamente il codice HTTP.

3.2.2.5.2.7 LogoutResponseDto

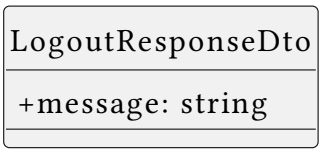

Figure 245: Class Diagram — LogoutResponseDto

`LogoutResponseDto` definisce la risposta HTTP per l'operazione di logout. Il campo `message` trasporta un messaggio testuale di conferma, fornendo al client un feedback descrittivo dell'esito dell'operazione di chiusura sessione.

3.2.2.5.3 Filters

3.2.2.5.3.1 AllExceptionsFilter

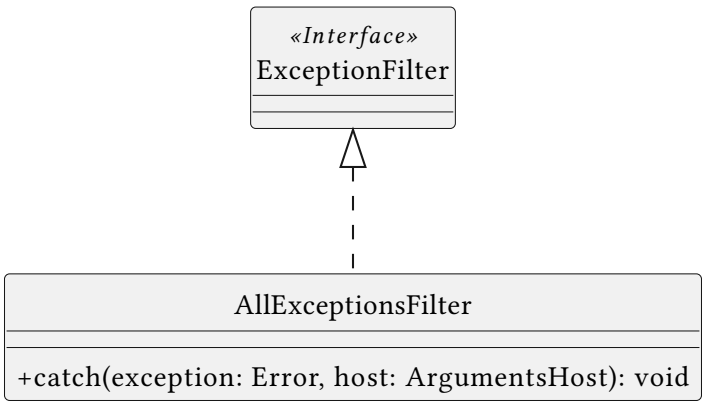

Figure 246: Class Diagram — AllExceptionsFilter

`AllExceptionsFilter` è il filtro globale delle eccezioni del microservizio Account. Implementa l'interfaccia `ExceptionFilter` di NestJS e intercetta tutte le eccezioni non gestite che emergono dalla catena di elaborazione delle richieste, traducendole in risposte HTTP strutturate e coerenti.

- **Centralizzazione della Gestione degli Errori:** Concentrare la traduzione delle eccezioni in un unico filtro garantisce uniformità nel formato delle risposte di errore verso i client, evitando che dettagli tecnici interni vengano esposti accidentalmente.
- **Mapping Eccezioni - HTTP:** Il filtro implementa la logica di mapping tra le eccezioni di dominio (es. `InvalidCredentialsException`) e i codici di stato HTTP appropriati (es. `401 Unauthorized`), centralizzando questa trasformazione e rimuovendo la necessità di gestirla nei singoli controller.

3.2.3 Frontend Application

Il frontend di Code Guardian è una **Single-Page Application** (SPA) sviluppata in TypeScript con React 19, strutturata seguendo una rigorosa implementazione del pattern architetturale **Model-View-ViewModel** (MVVM). L'obiettivo di questa architettura è massimizzare il disaccoppiamento tra la logica di business, lo strato di comunicazione di rete e la presentazione visiva. Le responsabilità sono distribuite in quattro strati orizzontali con un flusso di dipendenze strettamente unidirezionale, che fluisce sempre dalla View verso il Model, garantendo un'elevata testabilità dei singoli moduli.

3.2.3.1 Pattern architetturale: MVVM

Il pattern MVVM (Model-View-ViewModel) adottato separa le responsabilità in tre aree principali, limitando i side-effect e favorendo la riusabilità del codice.

3.2.3.1.0.1 Model

Rappresenta i dati di dominio, i contratti di rete e la logica di accesso remoto. Comprende i moduli API (`AuthApi`, `UsersApi`, `RepositoriesApi`, `AnalysisApi`) e i tipi TypeScript condivisi. Questo strato è completamente agnostico rispetto a React: non utilizza hook, non conosce il ciclo di vita dei componenti ed è testabile tramite framework Node.js standard.

3.2.3.1.0.2 ViewModel

Agisce da collante tra il Model e la View. Mantiene lo stato applicativo osservabile e incapsula la logica di business e le interazioni asincrone. È implementato tramite React Context (`AuthContext`) per lo stato globale e custom hook (`useAuth`, `useAnalysisPolling`) per gli stati locali e transazionali. Il ViewModel intercetta le azioni della View, invoca i metodi del Model e aggiorna reattivamente lo stato, innescando il re-rendering della UI.

3.2.3.1.0.3 View

Costituisce l'interfaccia utente puramente dichiarativa, composta da pagine e componenti React. La View si limita a leggere lo stato reattivo esposto dal ViewModel e a delegare ad esso le azioni utente (es. click, submit). Essendo priva di logica di business complessa o chiamate HTTP dirette, risulta facilmente testabile tramite snapshot e test di accessibilità.

La dipendenza è rigorosamente unidirezionale: View → ViewModel → Model. Nessuno strato dipende dallo strato superiore, e il Model ignora l'esistenza degli altri due.

3.2.3.2 Strati dell'architettura

3.2.3.2.1 Model — API Layer

Lo strato Model è composto da cinque moduli specializzati, costruiti attorno a un'istanza Axios fortemente tipizzata e centralizzata denominata `Gateway`:

3.2.3.2.1.1 Gateway

Istanza Axios singleton che gestisce dinamicamente il `baseURL` e centralizza le logiche trasversali (Cross-Cutting Concerns). L'interceptor in **request** analizza il path (es. `/account` o `/analysis`) per instradare la chiamata al microservizio corretto leggendo le variabili d'ambiente (`VITE_ACCOUNT_URL` o `VITE_ANALYSIS_URL`) e inietta l'header `Authorization` con il token Bearer. L'interceptor in **response** esegue due operazioni critiche:

1. **Data Normalization:** Adatta le risposte grezze del backend ai formati attesi dalla UI (es. converte la copertura test da proporzione decimale a percentuale intera, e mappa le costanti del database in stringhe standardizzate per il frontend).
2. **Refresh Token Queue:** Implementa una logica avanzata di gestione dell'errore 401 (Unauthorized). In caso di token scaduto, il Gateway sospende temporaneamente tutte le richieste

HTTP in uscita, mettendole in una coda di attesa, e lancia una singola chiamata di `/refresh`. In caso di successo, aggiorna le credenziali, applica il nuovo token e svuota la coda rieseguendo le chiamate sospese in modo trasparente per l'utente. Se il refresh fallisce (o se la richiesta originale era proprio il refresh), il Gateway purga lo storage locale e forza un hard redirect alla schermata di login.

3.2.3.2.1.2 AuthApi

Espone le chiamate di autenticazione (`login`, `register`, `refresh`, `logout`) verso il microservizio Account. Implementa la deserializzazione sicura delle risposte, restituendo le tuple di token (`accessToken`, `refreshToken`) e i metadati dell'oggetto `user`.

3.2.3.2.1.3 UsersApi

Gestisce le operazioni critiche legate all'identità, incluse la mutazione della password e la procedura irreversibile di cancellazione dell'account.

3.2.3.2.1.4 PatApi

Incapsula le operazioni CRUD relative ai Personal Access Token (PAT) necessari per l'analisi di repository GitHub privati, associandoli crittograficamente ai repository specifici.

3.2.3.2.1.5 RepositoriesApi

Gestisce l'entità Repository. Include la paginazione server-side, la ricerca full-text, il recupero dello storico e la generazione della classifica globale per score. Il metodo `startAnalysis` agisce da proxy unificato: formatta il payload configurazionale (branch, aree selezionate, flag IA) e innesca la catena di orchestrazione sul backend.

3.2.3.2.1.6 AnalysisApi

Dedicato esclusivamente al ciclo di vita del report. Include il recupero dei dati di audit e l'esposizione di endpoint formattati per l'esportazione documentale (blob PDF) o strutturata (JSON raw).

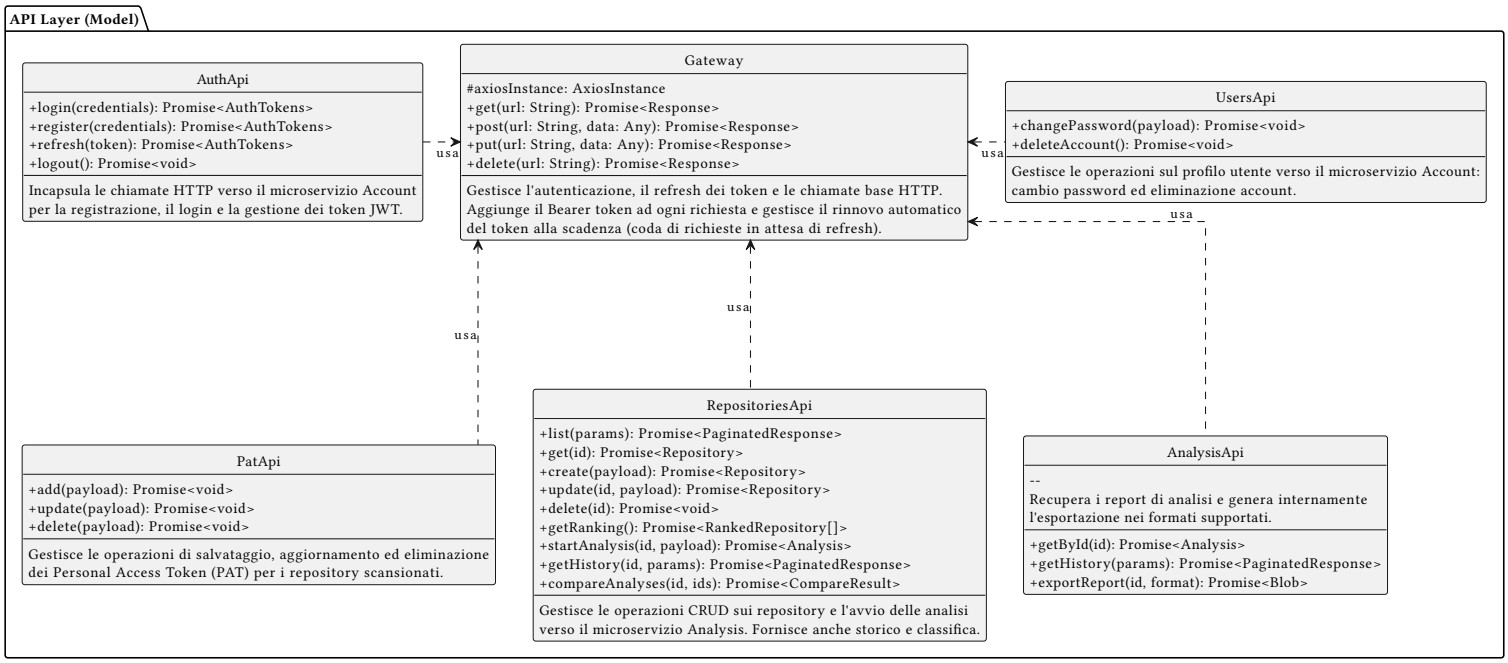

Figure 247: Class Diagram — api_layer

3.2.3.2.2 Model — Tipi di dominio

Il modulo `@/types` costituisce la singola fonte di verità (Single Source of Truth) per i contratti dati dell'intera applicazione. Questo garantisce un rigoroso controllo a tempo di compilazione.

3.2.3.2.2.1 Entità

Interfacce TypeScript dettagliate: `User`, `Repository`, `Analysis` (stato del job), `AnalysisReport` (aggregato), `Issue` (vulnerabilità generica), `RankedRepository`, `CodeAgentStaticIssue` e `AIInterpretation`. Queste interfacce mappano esplicitamente i DTO restituiti da NestJS.

3.2.3.2.2.2 Enum e Type Union

Sfruttando le potenzialità di TypeScript, vengono definiti tipi unione stretti per evitare stringhe magiche: `AnalysisStatus` (`not-analyzed` | `pending` | `in-progress` | `completed` | `failed`), `IssueSeverity` (`critical` | `high` | `medium` | `low` | `info`), e `AnalysisArea` (`code` | `security` | `documentation`).

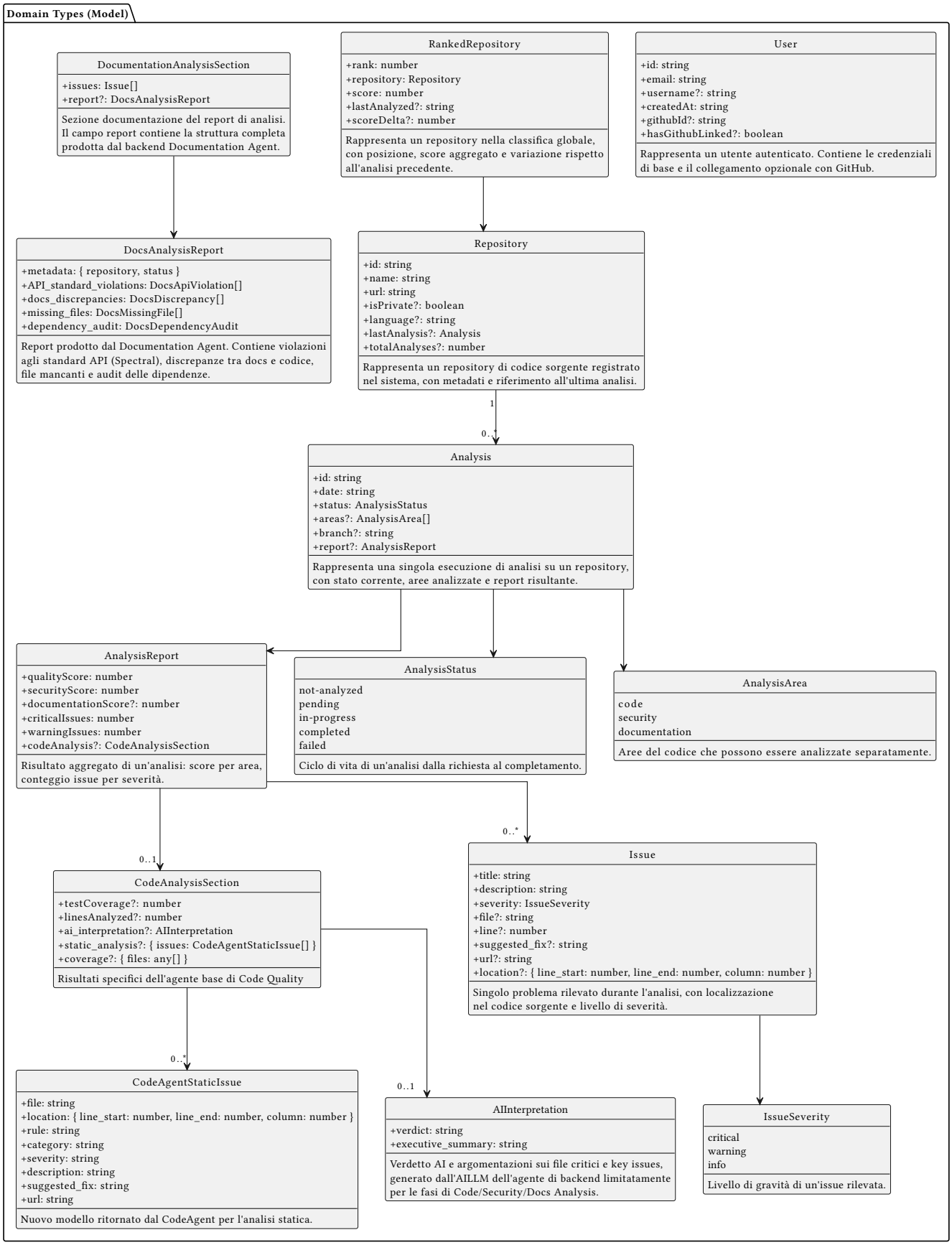

Figure 248: Class Diagram — types

3.2.3.2.3 ViewModel — Context Layer

Il React Context funge da provider globale per lo stato che deve sopravvivere alla navigazione tra le route:

3.2.3.2.3.1 AuthProvider

Gestisce la macchina a stati dell'utente autenticato (`user`, `isAuthenticated`, `isLoading`). Per ottimizzare il First Contentful Paint (FCP) ed evitare waterfall di rete, al mount il provider non chiama il backend. Effettua invece un'operazione di **JWT Decoding** in locale: estrae il payload in base64 del token salvato, ne esegue il parsing, verifica la firma temporale (claim `exp` contro `Date.now()`) e, se valido, idrata istantaneamente lo stato utente. Espone tramite Context API le funzioni mutazionali `login`, `register`, `logout` e `refreshUser`.

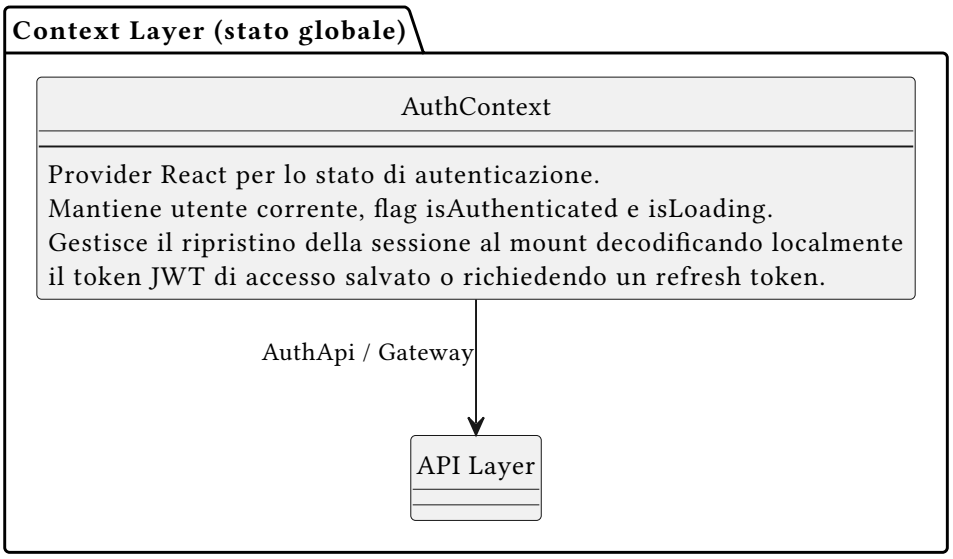

Figure 249: Class Diagram — contexts

3.2.3.2.4 ViewModel — Hooks Layer

I custom hook applicano il principio di Single Responsibility isolando le logiche transazionali dai componenti visivi:

3.2.3.2.4.1 useAuth

Hook di utilità che astrae l'accesso ad `AuthContext`. Implementa controlli di sicurezza e Type Narrowing: se invocato all'interno di un componente protetto, garantisce al compilatore TypeScript che la proprietà `user` non sia mai `null`, eliminando la necessità di controlli opzionali superflui nella View.

3.2.3.2.4.2 useAnalysisPolling

Hook complesso incaricato di gestire il ciclo di vita del polling HTTP. Implementa internamente `useRef` per tracciare l'ID dell'intervallo di polling e `useEffect` con funzione di cleanup per garantire che le chiamate vengano interrotte al dismount del componente, prevenendo memory leak e aggiornamenti di stato su componenti non montati. Gestisce la deduplicazione delle chiamate e invoca proattivamente i callback `onStarted`, `onProgress`, `onCompleted` e `onFailed` per informare la UI.

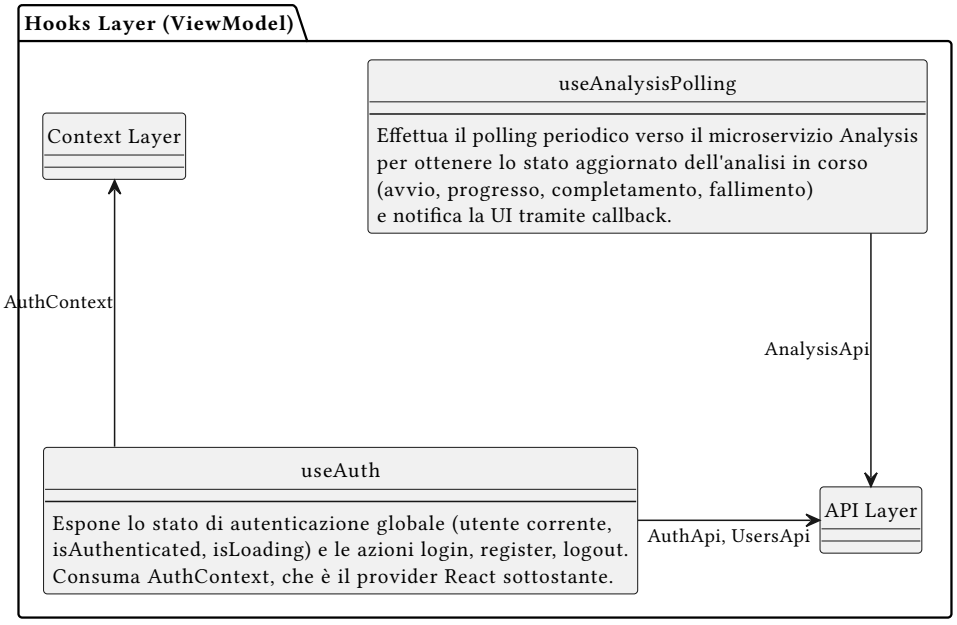

Figure 250: Class Diagram — hooks_logic

3.2.3.2.5 View — Componenti

La UI è sviluppata seguendo l'approccio **Atomic Design** e fa largo uso della libreria Tailwind CSS per lo styling utility-first.

Componenti di dominio:

3.2.3.2.5.1 AppLayout

Higher-Order Component visivo che avvolge le route protette. Esegue il rendering condizionale: se `isAuthenticated` è falso, blocca il render dell'albero sottostante e invoca un redirect dichiarativo (`<Navigate />`) verso `/login`.

3.2.3.2.5.2 Sidebar

Gestisce la navigazione principale. Implementa indicatori visivi di stato attivo per la route corrente e incapsula la logica di logout.

3.2.3.2.5.3 ScoreCard

Componente analitico altamente ottimizzato (tramite `React.memo` per prevenire re-render inutili). Renderizza un gauge SVG semicircolare calcolato matematicamente. L'interfaccia cromatica reagisce dinamicamente allo score: verde (≥ 75, sicuro), giallo (≥ 50, warning), rosso (< 50, critico).

3.2.3.2.5.4 AnalysisStatusBadge

Badge semantico che mappa i cinque stati dell'analisi in icone e classi colore specifiche, garantendo coerenza visiva in tutta l'app.

3.2.3.2.5.5 AddRepositoryModal e AnalysisOptionsModal

Dialog modali complessi che gestiscono lo stato dei form interni. Integrano librerie come `react-hook-form` e `zod` per la validazione sincrona e accessibile degli input (es. verifica regex dell'URL GitHub). Il modale di opzioni mappa dinamicamente il payload di invio in base al contesto del repository selezionato.

Primitive UI: Implementate tramite Radix UI e `shadcn/ui`, garantiscono la totale aderenza agli standard di accessibilità **WAI-ARIA** (navigazione da tastiera, screen reader support, focus trap nei

modali). Includono componenti come `Button` (con varianti polimorfiche), `Card`, `Dialog`, `Progress`, `Skeleton` (per i caricamenti progressivi) e `Separator`.

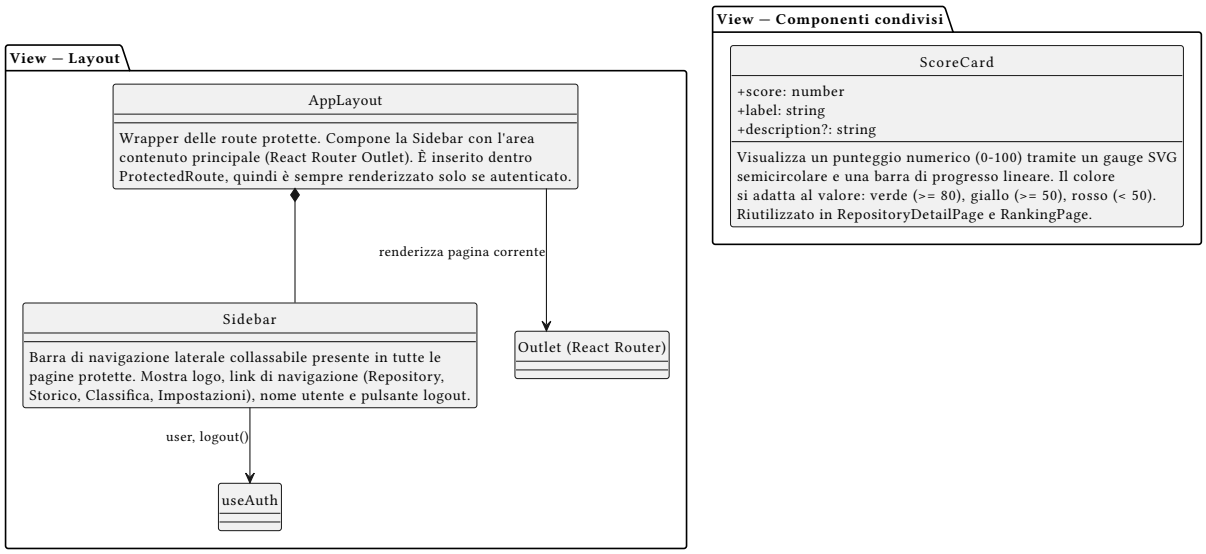

Figure 251: Class Diagram — components_1

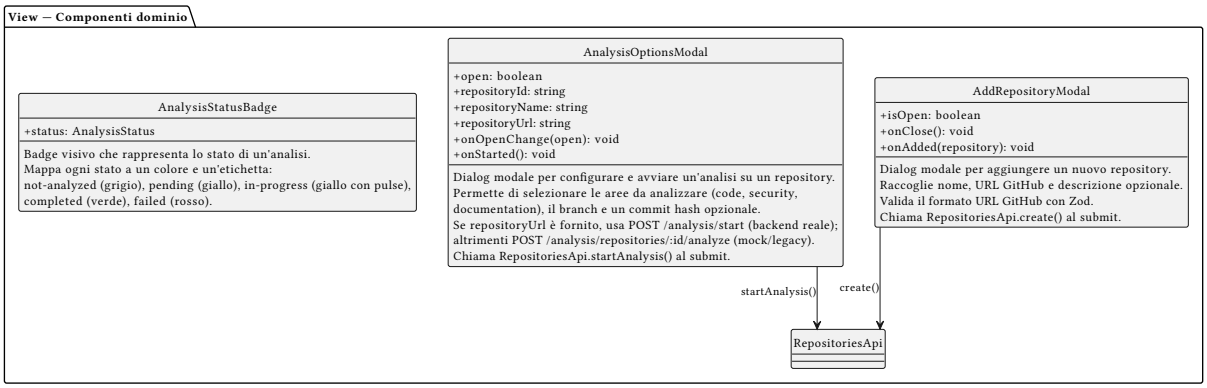

Figure 252: Class Diagram — components_2

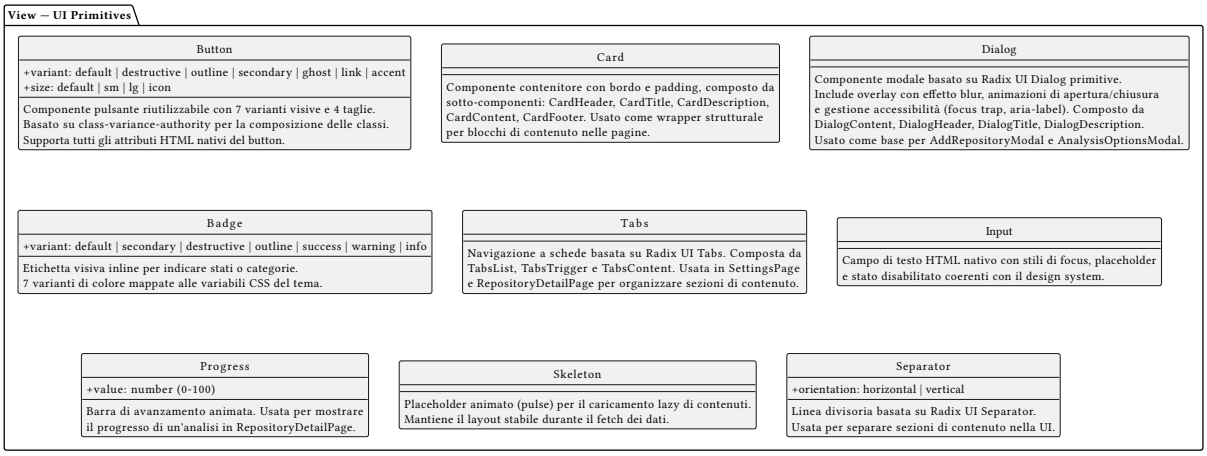

Figure 253: Class Diagram — components_3

3.2.3.2.6 View — Pagine

Il layer più alto della View mappa direttamente l'albero di routing.

Pagine pubbliche:

3.2.3.2.6.1 LandingPage e NotFoundPage

Pagine statiche orientate alla presentazione e al fallback visivo per errori di navigazione 404.

3.2.3.2.6.2 LoginPage e RegisterPage

Gestiscono i form di autenticazione. Delegano la validazione strutturale dei campi a schema Zod e passano i dati validati ai metodi esposti da `useAuth`, occupandosi unicamente di renderizzare eventuali messaggi di errore restituiti dal backend.

Pagine protette:

3.2.3.2.6.3 RepositoriesPage

Punto di ingresso operativo. Implementa strategie di debouncing per la barra di ricerca, ottimizzando le chiamate API, e gestisce lo stato di paginazione server-side dei repository monitorati.

3.2.3.2.6.4 RepositoryDetailPage

La pagina più complessa della SPA. Sfrutta il pattern a tab per sezionare l'enorme mole di dati del report (Code Quality, Security, Documentation, History). Integra `useAnalysisPolling`: durante un'analisi in corso, la pagina disabilita i pulsanti di azione, mostra una progress bar skeleton e resta in attesa dell'evento `onCompleted` per eseguire il fetch silenzioso del nuovo report e aggiornare i grafici.

3.2.3.2.6.5 HistoryPage e RankingPage

Cruscotti analitici. La `RankingPage` in particolare computa dinamicamente il delta prestazionale rispetto alla scansione precedente (`scoreDelta`), renderizzando icone di tendenza semantiche (`TrendingUp` in verde, `TrendingDown` in rosso) per fornire immediata contezza dell'evoluzione qualitativa del codice.

3.2.3.2.6.6 SettingsPage

Area di gestione sicura per mutazioni sensibili, incluse le configurazioni dei Personal Access Token e le zone di pericolo (Danger Zone) per la revoca dell'account, protette da modali di doppia conferma.

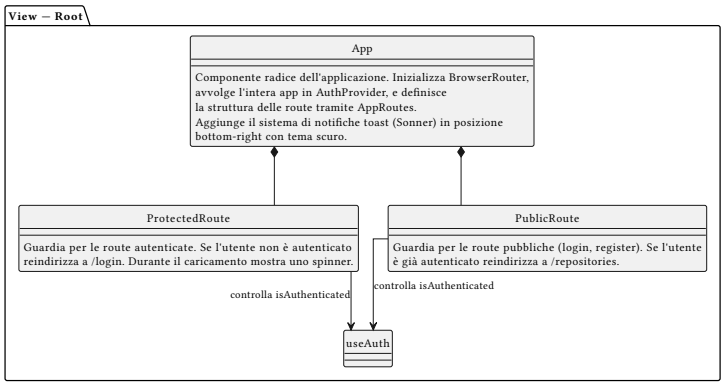

Figure 254: Class Diagram — app

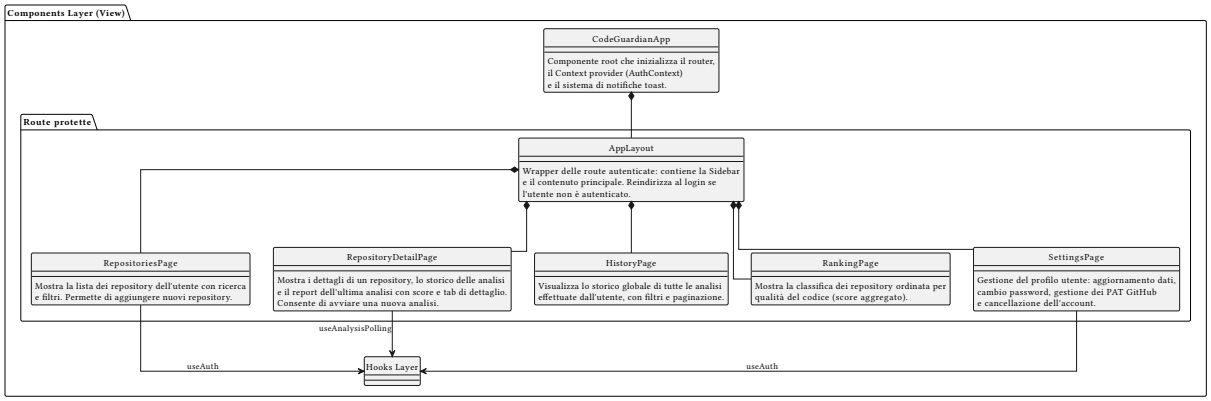

Figure 255: Class Diagram — components_view_private

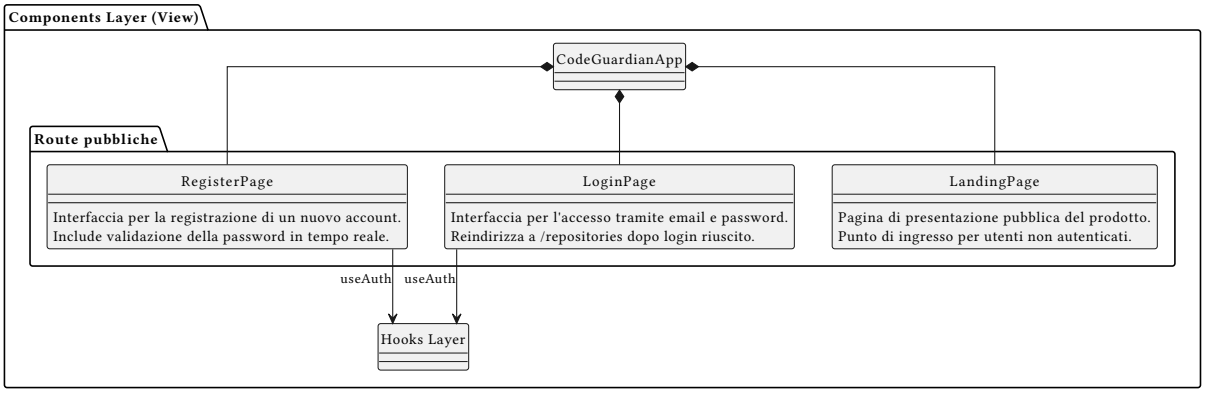

Figure 256: Class Diagram — component_view_public

3.2.3.3 Flusso di autenticazione

Il sistema di routing gerarchico è demandato a React Router v7. `App.tsx` definisce un router basato su browser history con due rami principali: le **Route pubbliche** e le **Route protette** (incapsulate da `AppLayout`).

L'intero flusso vitale dell'autenticazione è gestito proattivamente:

1. **Inizializzazione (Mount):** `AuthProvider` estrae il JWT. Se valido (decodifica payload locale superata), la sessione viene idratata.
2. **Scadenza Silenziosa:** Se l'utente chiude il tab e torna dopo giorni con token scaduto, l'inizializzazione locale fallisce istantaneamente e il router devia l'utente alla `/login` senza spreco di traffico di rete.
3. **Scadenza in Sessione:** Se il token scade mentre l'utente sta navigando (scatenando un 401 Unauthorized da una chiamata API), subentra l'interceptor del `Gateway`. Le chiamate in volo vengono congelate, l'invisibile richiesta di `/refresh` viene processata e, in caso positivo, le chiamate originarie ripartono in modo impercettibile per l'utente, garantendo un'esperienza fluida e ininterrotta (Seamless User Experience).

3.2.3.4 Aggiornamenti in tempo reale (Polling Strategico)

Dato che l'analisi agentica dei repository richiede un'elaborazione intensiva, asincrona e di lunga durata (ordine dei minuti dipendente dalla context window dell'LLM e dalle API di GitHub), l'architettura necessita di un sistema per fornire feedback all'utente.

Invece di adottare soluzioni complesse a livello infrastrutturale (come WebSockets o Server-Sent Events) che avrebbero gravato eccessivamente sull'API Gateway AWS, il frontend sfrutta un meccanismo di **Smart HTTP Polling** governato dall'hook `useAnalysisPolling`.

La logica implementativa è rigorosa:

- **Innesco:** All'avvio di un'analisi o all'apertura di un repository in stato `pending`/`in-progress`, l'hook innesca un timer (`setInterval`).
- **Ciclo Vitale:** Ogni n secondi, viene effettuata una richiesta HEAD o GET leggera per verificare il campo `status` del job.
- **Gestione Effetti:** A ogni cambio di stato, l'hook emette eventi specifici per aggiornare la UI progressivamente.
- **Cleanup e Ottimizzazione:** Non appena il backend segnala uno stato terminale (`completed` o `failed`), o nel momento in cui l'utente naviga verso un'altra pagina provocando l'unmount del componente, la funzione di cleanup di `useEffect` distrugge il timer. Questo meccanismo previene in modo assoluto memory leak, chiamate fantasma in background e race conditions nel re-rendering dello stato.

4 Design Patterns Applicati

4.1 Creazionali

4.1.1 Singleton

Dato l'utilizzo di nest per entrambi i microservizi, non è necessario implementare pattern singleton a livello di codice, in quanto il framework gestisce l'istanza dei servizi e degli adattatori come singleton per default. Ovvero un provider dichiarato in un modulo viene istanziato una sola volta e condiviso tra tutti i componenti che lo iniettano, garantendo implicitamente il comportamento singleton senza dover implementare manualmente il pattern. Questo permette di mantenere il codice pulito e focalizzato sulla logica di business, delegando al framework la gestione del ciclo di vita delle istanze.

4.1.2 Strutturali

4.1.3 Ports and Adapters

Il pattern adapter è presente in entrambi i microservizi data l'architettura logica applicata.

4.1.3.1 Problema risolto

Evita il forte accoppiamento logico tra il nucleo applicativo (Domain e Application) e i layer esterni come database, interfacce utente e servizi di terze parti, isolando la logica di business e rendendola indipendente dalle tecnologie di contorno.

4.1.3.2 Implementazione

- Nel microservizio Credenziali, gli adapters permettono di astrarre completamente la logica di business in merito ai dettagli sulle operazioni di memorizzazione dei dati e alle query sql, mantenendo nascosta la specifica tecnologia di database relazionale utilizzata (PostgreSQL). Al contempo soddisfano molteplici porte del core applicativo, garantendo un disaccoppiamento così netto da permettere, qualora si rivelasse necessario, di sostituire agilmente il database con una tecnologia differente.
- Nel microservizio Analysis, gli adapters permettono di astrarre completamente la logica di business in merito ai dettagli sulle operazioni di memorizzazione dei dati, gestione API esterne come github e AWS. Al contempo soddisfano molteplici porte del core applicativo, garantendo un disaccoppiamento così netto da permettere, qualora si rivelasse necessario, di sostituire agilmente un database o un servizio esterno con una tecnologia differente.

Inoltre, in entrambi i microservizi ogni porta espone un solo metodo dell'adapter aderendo al principio di segregazione delle interfacce, evitando di esporre metodi non necessari e mantenendo un contratto chiaro e specifico tra il core applicativo e le implementazioni infrastrutturali.

4.1.4 Facade

4.1.4.1 Problema risolto

Fornisce un'interfaccia semplificata e unificata a un insieme di interfacce in un sottosistema, nascondendo la complessità delle interazioni tra i componenti sottostanti e facilitando l'uso del sistema da parte dei client.

4.1.4.2 Implementazione

Nel microservizio di analisi, `StartAnalysisService` funge da Facade, orchestrando un flusso complesso che coinvolge più adapter (GitHubAdapter, S3Adapter, MongoDBAdapter) e `AnalysisOrchestratorService` per eseguire un'analisi completa. Fornisce un'interfaccia

semplificata che nasconde la complessità sottostante, permettendo al controller di avviare un'analisi con una singola chiamata.

4.2 Comportamentali

4.2.1 Orchestrator

4.2.1.1 Problema risolto

Coordina l'esecuzione di un processo complesso che coinvolge più componenti o servizi, definendo l'ordine delle operazioni e gestendo le dipendenze tra di esse, senza che i componenti coinvolti debbano conoscere l'intero flusso o le responsabilità degli altri.

4.2.1.2 Implementazione

Nel microservizio di analisi, `AnalysisOrchestratorService` funge da Orchestrator, coordinando l'intero processo di analisi del codice. Gestisce l'ordine delle operazioni, come la chiamata selettiva degli adapter per gli agenti, la memorizzazione dei risultati ottenuti e la gestione degli errori, senza che i singoli adapter o servizi coinvolti debbano conoscere l'intero flusso o le responsabilità degli altri componenti.

4.2.2 Command

Il pattern Command è ampiamente utilizzato in entrambi i microservizi per incapsulare tutte le informazioni necessarie a eseguire un'azione o un'operazione specifica, permettendo di disaccoppiare il mittente dell'azione dalla logica che la esegue. L'utilizzo di questo patter è guidato dalla scelta di architettura logica esagonale.

4.2.2.1 Problema risolto

Semplifica le firme dei metodi nei casi d'uso, evitando il passaggio di liste di argomenti lunghe e fragili alle modifiche.

4.2.2.2 Implementazione

Invece di passare molteplici parametri sparsi ai metodi dei servizi, ogni Use Case accetta come unico parametro un oggetto istanza di un Command specifico, che raggruppa logicamente e tipizza tutti i parametri necessari per svolgere l'operazione. Facendo una prima validazione dei campi con dei decoratori(`@IsString`,`@IsNotEmpty`...), questo evita che i dati in ingresso siano incompleti o malformati, e permette di bloccare richieste con body non validi prima di essere processate.

4.2.3 State

4.2.3.1 Problema risolto

Permette di gestire in modo chiaro e organizzato i diversi stati di un processo o entità, definendo transizioni ben definite tra di essi e facilitando la manutenzione del codice.

4.2.3.2 Implementazione

Nel microservizio di analisi, il pattern State è applicato alla gestione dello stato dell'analisi del codice. L'entità `GitHubAnalysis` ha un campo `status` che rappresenta lo stato attuale dell'analisi (es. `pending`, `in-progress`, `completed`, `failed`). Le transizioni di stato sono gestite internamente all'entitá, evitando un passaggio non valido da uno stato all'altro, come tra `failed` e `completed` o tra `pending` e `completed`.

4.2.4 Strategy

4.2.4.1 Problema risolto
Permette di variare il comportamento di validazione e autorizzazione del repository senza introdurre logica condizionale complessa nei servizi applicativi.

4.2.4.2 Implementazione
Nel microservizio di Analisi il pattern è applicato in due punti: `GitValidatorService`, che seleziona dinamicamente la strategia tra validazione per commit, branch o default, e `GitAuthorizerService`, che sceglie tra autorizzazione privata (token utente da persistenza) e pubblica (token di sistema da configurazione). In questo modo il servizio chiamante dipende da un contratto unico, mentre l'algoritmo concreto viene scelto a runtime in base al contesto della richiesta.

4.2.5 Dependency Injection
Sfruttando nativamente le capacità del framework NestJS, l'**Iniezione delle Dipendenze (DI)** rappresenta uno dei pattern tecnici principali alla base del progetto software.

4.2.5.1 Problema risolto
La Dependency Injection risolve il problema dell'accoppiamento rigido tra una classe e le sue dipendenze concrete. Senza DI, ogni componente crea direttamente i servizi che usa, rendendo il codice più fragile ai cambiamenti e difficile da testare.

Con DI:
- le dipendenze sono fornite dall'esterno (container IoC);
- il codice dipende da interfacce/contratti, non da classi concrete;
- modularità, riuso e testabilità (mock/stub) migliorano in modo significativo.

4.2.5.2 Implementazione
Attraverso i costruttori di classe, i vari Controllers e i Services ricevono all'avvio del sistema le loro rispettive dipendenze sotto forma ridotta di interfacce/componenti di istanziazione validati. Un container `Inversion of Control` (IoC) organizzato in un module di NestJs di supporto si prende in totale carico l'apposita istanziazione ed assegnazione dei componenti.

4.2.6 Repository

4.2.6.1 Problema risolto
Isola la logica di accesso ai dati dal livello di business, nascondendo i dettagli legati al database (query, connessioni, ORM/ODM). Questo permette al dominio applicativo di trattare la persistenza come una semplice collezione di oggetti in memoria, garantendo testabilità (tramite mock) e la possibilità di cambiare tecnologia di storage senza impattare la logica di core.

4.2.6.2 Implementazione
Nel microservizio Analysis, il dominio definisce il contratto attraverso porte specifiche, le quali stabiliscono le firme dei metodi per recuperare e salvare le entità di dominio come i job di analisi. L'implementazione concreta è delegata agli adapter infrastrutturali, come `MongoDBAdapter`, che traducono queste chiamate in comandi nativi per Mongoose/MongoDB. Questo approccio garantisce che i casi d'uso orchestrino i dati in modo totalmente agnostico rispetto alla natura documentale del database sottostante. Analogamente, nel microservizio Account, il pattern astrae le operazioni sul database relazionale gestito tramite l'apposito adapter `PostgresAdapter`.

4.2.7 Data Transfer Object (DTO)

4.2.7.1 Problema risolto

Il pattern DTO permette di trasferire dati tra i diversi layer del microservizio e verso i client esterni senza esporre direttamente le entità di dominio interno che contengono una logica di core che non deve essere esposta.

4.2.7.2 Implementazione

Il pattern **DTO** viene impiegato sistematicamente in entrambi i microservizi sia a livello di presentazione (Request e Result DTOs) che a livello applicativo per trasportare dati sotto forma di tipi primitivi. Tramite i DTO, i dati in transito assumono una forma asettica e consona per le sole esigenze di comunicazione.

5 Mappatura dei Requisiti di Sistema

5.1 Stato Attuale dei Requisiti Funzionali

ID	Descrizione	Stato
FRObl	Il Sistema deve consentire all'Utente non registrato l'accesso alla sezione di creazione account.	SODDISFATTO
FRObl2	Il Sistema deve predisporre un comando di conferma per l'invio del modulo di registrazione.	SODDISFATTO
FRObl3	Il Sistema deve eseguire la validazione completa dei campi obbligatori (presenza, formato, conformità ai vincoli e univocità) al momento dell'invio del modulo di registrazione.	SODDISFATTO
FRObl4	Il Sistema deve permettere la finalizzazione della registrazione solo a seguito della validazione positiva di tutti i campi obbligatori.	SODDISFATTO
FRObl5	Il Sistema deve creare e memorizzare un record account che includa almeno username, email, hash della password e salt associato a seguito di registrazione completata con esito positivo.	SODDISFATTO
FRObl6	Il Sistema deve memorizzare le chiavi di accesso esclusivamente in forma cifrata tramite un algoritmo di hashing sicuro e generare un salt univoco per ciascun account.	SODDISFATTO
FRObl7	Il Sistema deve garantire l'atomicità della procedura di registrazione: in caso di fallimento della persistenza, nessun record parziale deve essere mantenuto nel database.	SODDISFATTO
FRObl8	Il Sistema deve visualizzare un messaggio di conferma esplicito a seguito della creazione corretta dell'account CodeGuardian.	SODDISFATTO
FRObl9	Il Sistema deve rilevare il tentativo di invio del modulo di registrazione in presenza di campi obbligatori vuoti.	SODDISFATTO
FRObl10	Il Sistema deve inibire la registrazione e notificare l'utente indicando specificamente quali dati obbligatori non sono stati inseriti.	SODDISFATTO

ID	Descrizione	Stato
FRObi11	Il Sistema deve consentire l'immissione di un username alfanumerico con lunghezza compresa tra 4 e 20 caratteri.	SODDISFATTO
FROp12	Il Sistema deve verificare l'univocità dello username rispetto agli account esistenti nel database.	NON SODDISFATTO
FROp13	Il Sistema deve imporre vincoli di unicità lato persistenza su username per prevenire registrazioni duplicate anche in presenza di richieste concorrenti.	NON SODDISFATTO
FROp14	In caso di violazione del vincolo di unicità in fase di persistenza, il Sistema deve annullare la registrazione e notificare l'utente con il messaggio previsto per username già in uso.	NON SODDISFATTO
FRObi15	Il Sistema deve inibire l'avanzamento della procedura e mostrare un messaggio di errore qualora lo username inserito non rispetti i vincoli sintattici previsti.	SODDISFATTO
FRDe16	Il Sistema deve consentire l'immissione di un indirizzo email conforme allo standard RFC 5322.	NON SODDISFATTO
FRObi17	Il Sistema deve rifiutare indirizzi email contenenti spazi o privi del carattere "@".	SODDISFATTO
FRObi18	Il Sistema deve verificare l'univocità dell'indirizzo email rispetto agli account esistenti nel database.	SODDISFATTO
FRObi19	Il Sistema deve imporre vincoli di unicità lato persistenza su indirizzo email per prevenire registrazioni duplicate.	SODDISFATTO
FRObi20	Il Sistema deve inibire l'avanzamento della procedura e mostrare un messaggio di errore qualora l'indirizzo email inserito non sia conforme ai requisiti sintattici.	SODDISFATTO
FRObi21	Il Sistema deve accettare una password solo se di lunghezza pari o superiore ad 8 caratteri.	SODDISFATTO
FRObi22	Il Sistema deve accettare una password solo se include almeno una lettera maiuscola, una lettera minuscola, una cifra e un carattere speciale.	SODDISFATTO
FRObi23	Il Sistema deve rifiutare password che coincidono con lo username o che contengono lo username come sottostringa.	SODDISFATTO

ID	Descrizione	Stato
FRObb24	Il Sistema deve inibire l'avanzamento della procedura e mostrare un messaggio che specifichi i requisiti di sicurezza non soddisfatti.	SODDISFATTO
FRObb25	Il Sistema deve consentire all'Utente non autenticato l'accesso alla sezione di autenticazione (Login).	SODDISFATTO
FRObb26	Il Sistema deve predisporre un comando di conferma per finalizzare la procedura di accesso.	SODDISFATTO
FRObb27	Il Sistema deve eseguire la validazione completa delle credenziali (presenza, formato e corrispondenza) al momento dell'invio del modulo.	SODDISFATTO
FRObb28	Il Sistema deve garantire l'accesso alle funzionalità riservate esclusivamente a seguito di una corretta validazione delle credenziali.	SODDISFATTO
FRObb29	Il Sistema deve reindirizzare l'Utente verso la dashboard principale a seguito di autenticazione avvenuta con successo.	SODDISFATTO
FRObb30	Il Sistema deve utilizzare protocolli di comunicazione sicuri (HTTPS) per il trasferimento delle credenziali durante il login.	SODDISFATTO
FRDee31	Il Sistema deve utilizzare lo username fornito per recuperare dalla persistenza il record account associato.	NON SODDISFATTO
FRObb32	Il Sistema deve verificare la password inserita confrontando l'hash calcolato con l'hash memorizzato tramite la medesima funzione di hashing e salt.	SODDISFATTO
FRObb33	Il Sistema deve implementare meccanismi di rate limiting o lockout temporaneo a seguito di ripetuti tentativi di autenticazione falliti.	SODDISFATTO
FRObb34	Il Sistema deve visualizzare un indicatore di caricamento (spinner) durante la validazione delle credenziali per prevenire invii multipli.	SODDISFATTO
FRObb35	Il Sistema deve rilevare campi incompleti nel login e inibire l'accesso notificando l'utente tramite avviso specifico.	SODDISFATTO
FRObb36	Il Sistema deve inibire l'avanzamento della procedura e mostrare un messaggio di errore qualora le credenziali	SODDISFATTO

ID	Descrizione	Stato
	non risultino valide o non corrispondano a nessun account registrato.	
FROp37	Il Sistema deve consentire all'Utente Autorizzato l'accesso alla sezione dedicata al collegamento del profilo GitHub.	NON SODDISFATTO
FROp38	Il Sistema deve impedire l'avvio della procedura di collegamento qualora un profilo GitHub risulti già associato all'account CodeGuardian dell'utente.	NON SODDISFATTO
FROp39	Il Sistema deve utilizzare un parametro di stato (state) per prevenire attacchi di tipo Cross-Site Request Forgery (CSRF) durante il flusso OAuth2.	NON SODDISFATTO
FROp40	Il Sistema deve memorizzare i token di accesso ottenuti da GitHub esclusivamente in forma cifrata tramite algoritmi di crittografia forte (es. AES-256).	NON SODDISFATTO
FROp41	Il Sistema deve evitare la persistenza di token o associazioni qualora la procedura di collegamento non termini con esito positivo.	NON SODDISFATTO
FROp42	Il Sistema deve mostrare un avviso informativo obbligatorio prima di procedere al reindirizzamento verso il dominio esterno GitHub.	NON SODDISFATTO
FROp43	Il Sistema deve consentire all'utente di annullare il reindirizzamento, ripristinando lo stato della sezione integrazioni senza alcuna modifica.	NON SODDISFATTO
FROp44	Il Sistema deve gestire i timeout nelle chiamate verso le API di GitHub durante lo scambio del token, notificando l'utente del fallimento temporaneo.	NON SODDISFATTO
FROp45	Il Sistema deve inibire il collegamento qualora il profilo GitHub risulti già associato a un altro account CodeGuardian.	NON SODDISFATTO
FROp46	Il Sistema deve mostrare un messaggio di errore specifico qualora l'utente neghi il consenso alla condivisione dei dati su GitHub.	NON SODDISFATTO
FROb47	Il Sistema deve consentire l'immissione dell'URL del repository GitHub nel modulo di richiesta analisi.	SODDISFATTO

ID	Descrizione	Stato
FRObi48	Il Sistema deve validare che l'URL del repository GitHub inserito utilizzi il protocollo "https://" e punti al dominio "github.com".	SODDISFATTO
FRObi49	Il Sistema deve verificare la dimensione del repository tramite API GitHub e inibire l'analisi qualora questa superi i limiti tecnici prestabiliti.	SODDISFATTO
FRObi50	Il Sistema deve disabilitare il comando di conferma dell'invio a seguito della pressione dell'utente per prevenire richieste duplicate.	SODDISFATTO
FROpi51	Il Sistema deve consegnare la notifica di fine analisi attraverso i canali scelti dall'utente (es. email o notifiche app).	NON SODDISFATTO
FRDei52	Il Sistema deve mostrare i dettagli dell'analisi (nome progetto e ora) direttamente nell'avviso ricevuto dall'utente.	SODDISFATTO
FRObi53	Il Sistema deve inviare un avviso immediato se un'analisi si interrompe per un errore imprevisto, spiegandone brevemente il motivo.	SODDISFATTO
FRObi54	Il Sistema deve restituire immediatamente il report esistente, senza avviare una nuova elaborazione, informando l'utente qualora i dati remoti risultino già aggiornati.	SODDISFATTO
FROpi55	Il Sistema deve accodare la richiesta di analisi al processo già in corso per il medesimo repository, informando l'utente dell'avvenuta presa in carico.	NON SODDISFATTO
FRObi56	Il Sistema deve inibire la richiesta di analisi qualora non venga selezionata almeno un'area di interesse.	SODDISFATTO
FRObi57	Il Sistema deve ordinare l'elenco dei repository analizzati in ordine decrescente rispetto alla data dell'ultima analisi disponibile.	SODDISFATTO
FRObi58	Il Sistema deve garantire che l'utente possa consultare i risultati nella propria area personale anche se la notifica via email non viene recapitata.	SODDISFATTO
FRObi59	Il Sistema deve contrassegnare l'analisi come "Fallita" nella lista dei progetti dell'utente se il processo non può essere completato.	SODDISFATTO

ID	Descrizione	Stato
FROb60	Il Sistema deve rendere visibili le cause del fallimento all'interno della dashboard, indipendentemente dall'invio o dalla ricezione dell'avviso di errore.	SODDISFATTO
FROb61	Il Sistema deve inibire la visualizzazione della lista e mostrare un'informativa specifica qualora non risultino repository analizzati.	SODDISFATTO
FROb62	Il Sistema deve inibire il rendering della lista e mostrare una notifica di errore qualora i servizi di persistenza non siano raggiungibili.	SODDISFATTO
FROb63	Il Sistema deve fornire un comando di aggiornamento (Refresh) per consentire un nuovo tentativo di caricamento in caso di errore tecnico.	SODDISFATTO
FROb64	Il Sistema deve consentire la selezione di un repository dalla lista per il recupero del report di dettaglio associato.	SODDISFATTO
FROb65	Il Sistema deve validare lato server che il report richiesto appartenga al repository associato all'account dell'Utente Autorizzato prima del rendering.	SODDISFATTO
FROb66	Il Sistema deve inibire il rendering e mostrare un errore di autorizzazione qualora l'utente tenti di accedere a un report di un repository non associato al proprio profilo.	SODDISFATTO
FROb67	Il Sistema deve gestire i timeout nel recupero dei dati analitici dalla persistenza, notificando l'utente in caso di indisponibilità temporanea del report.	SODDISFATTO
FROb68	Il Sistema deve permettere la selezione o deselezione dinamica delle aree analitiche (Codice, Sicurezza, Documentazione) tramite interfaccia utente.	SODDISFATTO
FROb69	Il Sistema deve aggiornare dinamicamente il contenuto a video in base ai filtri applicati senza richiedere il ricaricamento dell'intera pagina.	SODDISFATTO
FROb70	Il Sistema deve inibire la visualizzazione delle aree analitiche e mostrare un avviso informativo qualora non risulti selezionata alcuna area nei filtri.	SODDISFATTO
FROb71	Il Sistema deve esporre i metadati identificativi del report recuperati in fase di caricamento.	SODDISFATTO

ID	Descrizione	Stato
FROb72	Il Sistema deve esporre il timestamp (data e ora ISO 8601) relativo alla generazione del report.	SODDISFATTO
FROb73	Il Sistema deve visualizzare l'identificativo SHA del commit GitHub analizzato, fornendo un link diretto al commit sulla piattaforma esterna.	SODDISFATTO
FROb74	Il Sistema deve esporre lo username o l'identificativo dell'account che ha originato la scansione.	SODDISFATTO
FROb75	Il Sistema deve presentare le metriche tecniche aggregate (es. punteggi di qualità, numero bug, vulnerabilità) per ogni sezione attiva sulla stessa schermata.	SODDISFATTO
FROb76	Il Sistema deve caricare e visualizzare la lista delle azioni correttive (remediation), esponendo per ogni elemento un titolo identificativo, il livello di criticità e una breve descrizione dell'intervento consigliato.	SODDISFATTO
FROb77	Il Sistema deve consentire l'espansione dei dettagli di ogni singola remediation per la visualizzazione della proposta di risoluzione tecnica.	SODDISFATTO
FROb78	Il Sistema deve visualizzare un messaggio di conferma esito positivo (badge "Clean" o simile) qualora il motore di analisi non rilevi criticità nella sezione.	SODDISFATTO
FROb79	Il Sistema deve consentire la selezione di un intervallo temporale tramite input di data (inizio e fine) per l'estrazione dei report storici dal database.	SODDISFATTO
FROb80	Il Sistema deve predisporre un comando di conferma per l'invio della richiesta di confronto dei dati.	SODDISFATTO
FROb81	Il Sistema deve inibire il caricamento dei dati e visualizzare un avviso specifico qualora i campi relativi alle date non risultino popolati.	SODDISFATTO
FROb82	Il Sistema deve impedire l'invio della richiesta qualora la data di inizio sia cronologicamente successiva alla data di fine, segnalando l'errore di coerenza.	SODDISFATTO
FROb83	Il Sistema deve limitare l'ampiezza dell'intervallo temporale a un massimo di 12 mesi solari, inibendo la richiesta e notificando l'utente in caso di superamento.	SODDISFATTO

ID	Descrizione	Stato
FRObi84	Il Sistema deve gestire l'assenza di dati nel periodo selezionato visualizzando un'informativa di "Nessun report trovato" senza interrompere la sessione utente.	SODDISFATTO
FRDe85	Il Sistema deve generare rappresentazioni grafiche dinamiche (es. grafici a linee o istogrammi) per illustrare l'evoluzione temporale delle metriche analitiche.	SODDISFATTO
FRDe86	Il Sistema deve abilitare tooltips informativi al passaggio del cursore (hover) sui punti dati dei grafici per mostrare i valori esatti e l'hash del commit associato.	SODDISFATTO
FRObi87	Il Sistema deve presentare una tabella comparativa che elenchi i report selezionati in ordine cronologico crescente.	SODDISFATTO
FRObi88	Il Sistema deve calcolare e visualizzare gli indicatori di variazione (trend incrementali o decrementali) tra ogni analisi e quella immediatamente precedente.	SODDISFATTO
FRObi89	Il Sistema deve garantire l'allineamento dei dati tra la vista grafica e la vista tabellare, effettuando una singola operazione di fetch atomica per l'intero intervallo.	SODDISFATTO
FRObi90	Il Sistema deve gestire eventuali errori di rendering dei grafici (es. mancanza di librerie client-side) mostrando in alternativa i dati grezzi in formato tabellare.	SODDISFATTO
FRObi91	Il Sistema deve caricare e visualizzare i dati relativi alla sezione "Codice" esclusivamente se l'area risulta attiva nei filtri di visualizzazione del report.	SODDISFATTO
FRObi92	Il Sistema deve esporre i risultati dell'analisi statica (bug, code smell) indicando per ogni rilievo la gravità e la posizione nel file sorgente.	SODDISFATTO
FRObi93	Il Sistema deve esporre la percentuale di copertura dei test (Code Coverage) e il rapporto tra test superati e falliti rispetto al totale eseguito.	SODDISFATTO
FRObi94	Il Sistema deve presentare la lista delle remediation specifiche per il codice, esponendo per ogni elemento un titolo identificativo, il file associato, la riga di codice	SODDISFATTO

ID	Descrizione	Stato
	interessata e il livello di severità, permettendo inoltre la navigazione verso il dettaglio della singola azione.	
FROb95	Il Sistema deve visualizzare un'informativa di "Codice Conforme" qualora non siano rilevati bug o violazioni degli standard qualitativi.	SODDISFATTO
FROb96	Il Sistema deve caricare i dati della sezione "Sicurezza" in modo asincrono rispetto alle altre sezioni per ottimizzare i tempi di risposta.	SODDISFATTO
FROb97	Il Sistema deve esporre l'elenco delle dipendenze vulnerabili indicando il codice CVE, il grado di severità (CVSS) e la versione sicura consigliata.	SODDISFATTO
FROb98	Il Sistema deve mappare i rilievi di sicurezza rispetto alle categorie della Top 10 OWASP per facilitare la valutazione della conformità.	SODDISFATTO
FROb99	Il Sistema deve presentare la lista delle remediation di sicurezza, esponendo per ogni elemento un titolo identificativo, la libreria o dipendenza vulnerabile associata, il livello di severità e l'azione correttiva consigliata, ordinandole prioritariamente in base alla criticità.	SODDISFATTO
FROb100	Il Sistema deve visualizzare un'informativa di "Repository Sicuro" qualora non siano rilevate vulnerabilità note nelle dipendenze o nel codice.	SODDISFATTO
FROb101	Il Sistema deve caricare e visualizzare i dati della sezione "Documentazione" analizzando la presenza e la sintassi dei file Markdown e testuali.	SODDISFATTO
FROb102	Il Sistema deve segnalare gli errori sintattici e i link interrotti individuati all'interno della documentazione del repository.	SODDISFATTO
FROb103	Il Sistema deve calcolare e mostrare un indice di completezza documentale basato sulla copertura delle interfacce pubbliche descritte.	SODDISFATTO
FROb104	Il Sistema deve esporre nell'elenco delle remediation documentali il nome del file interessato, il livello di severità e i suggerimenti testuali per l'integrazione delle parti di documentazione mancanti o incomplete.	SODDISFATTO

ID	Descrizione	Stato
FROb105	Il Sistema deve visualizzare un'informativa di "Documentazione Completa" qualora non siano rilevati errori o mancanze informative.	SODDISFATTO
FROb106	Il Sistema deve calcolare un punteggio di qualità globale (0-100) per ogni repository analizzato, basandosi sulle metriche pesate di codice, sicurezza e documentazione.	SODDISFATTO
FROb107	Il Sistema deve generare una graduatoria dinamica dei repository associati all'account dell'Utente Autorizzato, ordinata per punteggio di qualità decrescente.	SODDISFATTO
FROb108	Il Sistema deve esporre, per ogni riga del ranking: posizione in classifica, nome del repository, punteggio globale e un indicatore di trend rispetto al mese precedente.	SODDISFATTO
FROb109	Il Sistema deve inibire il rendering del ranking e visualizzare un'informativa specifica qualora non risultino analisi completate per l'account utente.	SODDISFATTO
FROp110	Il Sistema deve consentire la rimozione dell'integrazione GitHub esclusivamente previa conferma esplicita dell'Utente Avanzato.	NON SODDISFATTO
FROp111	Il Sistema deve inviare una richiesta di revoca del token OAuth alle API di GitHub al momento della conferma della disconnessione.	NON SODDISFATTO
FROp112	Il Sistema deve eliminare definitivamente dal database i token (access e refresh) e l'ID utente GitHub associato all'account CodeGuardian.	NON SODDISFATTO
FROp113	Il Sistema deve gestire eventuali errori di comunicazione con GitHub durante la revoca, procedendo comunque alla cancellazione locale dei dati sensibili.	NON SODDISFATTO
FROb114	Il Sistema deve rendere disponibile il file generato tramite un link di download	SODDISFATTO
FROb115	Il Sistema deve consentire l'esportazione dei report nei formati PDF (per consultazione) e JSON (per interoperabilità dati).	SODDISFATTO

ID	Descrizione	Stato
FROb116	Il Sistema deve inibire l'invio della richiesta di generazione file qualora l'utente non selezioni formalmente uno dei formati previsti.	SODDISFATTO
FROb117	Il Sistema deve generare il documento includendo i metadati del report (timestamp, commit hash) e i risultati delle sezioni effettivamente analizzate.	SODDISFATTO
FROb118	Il Sistema deve gestire il processo di generazione del file asincronamente per evitare il blocco dell'interfaccia utente durante il parsing di report voluminosi.	SODDISFATTO
FROb119	Il Sistema deve consentire all'Utente Autorizzato l'accesso alla sezione dedicata alla modifica della chiave di accesso.	SODDISFATTO
FROb120	Il Sistema deve richiedere l'immissione della password attualmente in uso e validarne la corrispondenza con l'hash memorizzato prima di procedere alla variazione.	SODDISFATTO
FROb121	Il Sistema deve inibire la procedura e mostrare un errore specifico qualora la password corrente non venga inserita o risulti errata.	SODDISFATTO
FROb122	Il Sistema deve validare che la nuova password rispetti i vincoli di complessità stabiliti per la registrazione iniziale.	SODDISFATTO
FROb123	Il Sistema deve confrontare l'hash della nuova password con quello attuale e impedire la modifica qualora i valori coincidano.	SODDISFATTO
FROb124	Il Sistema deve aggiornare la password nella persistenza esclusivamente tramite un nuovo processo di hashing sicuro e generazione di un nuovo salt univoco.	SODDISFATTO
FROb125	Il Sistema deve inviare una notifica email automatica all'indirizzo associato al profilo a seguito dell'avvenuta modifica delle credenziali.	SODDISFATTO
FROb126	Il Sistema deve invalidare tutte le sessioni attive dell'utente (ad eccezione di quella corrente) a seguito del cambio password avvenuto con successo.	SODDISFATTO
FROb127	Il Sistema deve consentire la visualizzazione dei dettagli tecnici di una specifica remediation selezionata dall'utente.	SODDISFATTO

ID	Descrizione	Stato
FROb128	Il Sistema deve esporre per ogni remediation: descrizione del difetto, snippet di codice interessato (se applicabile), grado di severità e proposta di risoluzione.	SODDISFATTO
FROb129	Il Sistema deve includere riferimenti o link a documentazione esterna (es. CWE, OWASP) qualora la remediation riguardi una vulnerabilità di sicurezza nota.	SODDISFATTO
FROb130	L'Orchestratore deve gestire il ciclo di vita della verifica accessibilità tramite chiamate asincrone verso le API REST di GitHub.	SODDISFATTO
FROb131	L'Orchestratore deve implementare un meccanismo di "Exponential Backoff" per gestire i tentativi di riconnessione in caso di errori di rete temporanei verso GitHub.	SODDISFATTO
FROb132	L'Orchestratore deve validare la raggiungibilità dell'endpoint API di GitHub inviando una richiesta di "Heartbeat" prima di tentare il fetch del repository.	SODDISFATTO
FROb133	L'Orchestratore deve prima tentare l'accesso al repository senza intestazioni di autorizzazione per verificare se la risorsa è di dominio pubblico.	SODDISFATTO
FROb134	In caso di errore HTTP 404 o 403 sulla risorsa pubblica, l'Orchestratore deve tentare una seconda richiesta iniettando nel modulo di autorizzazione il token OAuth 2.0 dell'utente.	SODDISFATTO
FROb135	L'Orchestratore deve verificare che il token fornito disponga degli "scopes" (permessi) minimi di lettura (repo o public_repo) necessari per il clonaggio.	SODDISFATTO
FROb136	In caso di fallimento definitivo (es. token scaduto o repository eliminato), l'Orchestratore deve inviare un segnale di interruzione al modulo di notifica e aggiornare lo stato dell'audit in "FAILED_ACCESS".	SODDISFATTO
FRDe137	Il Sistema deve consentire l'applicazione automatica delle modifiche al repository tramite l'integrazione GitHub a seguito dell'accettazione della remediation.	NON SODDISFATTO

ID	Descrizione	Stato
FRDe138	Il Sistema deve eseguire una validazione di integrità sulla proposta correttiva prima dell'invio del commit verso il repository esterno.	NON SODDISFATTO
FRDe139	Il Sistema deve aggiornare lo stato della remediation in "Applied" o "Dismissed" nel database di persistenza a seguito dell'azione dell'utente.	NON SODDISFATTO
FRDe140	Il Sistema deve notificare l'utente in caso di fallimento del processo di scrittura (commit) sul repository remoto durante l'accettazione.	NON SODDISFATTO
FRDe141	Il Sistema deve consentire all'Utente Autorizzato la definizione di un nome univoco per la raccolta di report all'interno del proprio account.	SODDISFATTO
FRDe142	Il Sistema deve validare la sintassi dell'URL GitHub fornito, assicurando l'uso del protocollo HTTPS e la corretta struttura del path user/repo.	SODDISFATTO
FRDe143	Il Sistema deve interrogare le API di GitHub per confermare l'esistenza e la raggiungibilità del repository indicato prima di finalizzare la raccolta.	SODDISFATTO
FRDe144	Il Sistema deve gestire i casi di inaccessibilità del repository (es. repository privato senza permessi) notificando l'utente tramite avviso specifico.	SODDISFATTO
FRDe145	Il Sistema deve impedire la creazione di raccolte duplicate che puntano al medesimo repository per lo stesso utente.	SODDISFATTO
FRDe146	Il Sistema deve memorizzare la descrizione facoltativa della raccolta supportando la codifica UTF-8 per caratteri speciali e simboli.	SODDISFATTO
FRDe147	L'Orchestratore deve parallelizzare le richieste di analisi verso i diversi strumenti per ottimizzare il tempo complessivo di esecuzione dell'audit.	SODDISFATTO
FRDe148	L'Orchestratore deve includere nella richiesta verso gli strumenti esterni i parametri di configurazione definiti dall'utente durante la fase di richiesta.	SODDISFATTO
FRDe149	L'Orchestratore deve trasmettere in modo sicuro (tramite secret manager) le credenziali o i token di accesso al servizio AWS incaricato della clonazione.	SODDISFATTO

ID	Descrizione	Stato
FROb150	L'Orchestratore deve monitorare il completamento della clonazione e gestire eventuali timeout o errori di spazio disco insufficiente sul volume di destinazione.	SODDISFATTO
FROb151	In caso di errore durante la clonazione, l'Orchestratore deve inibire l'invio delle richieste agli strumenti di analisi e liberare immediatamente le risorse allocate.	SODDISFATTO
FROb152	L'Orchestratore deve inoltrare la codebase o i file specifici agli strumenti di analisi esterna (Codice, Sicurezza, Documentazione) tramite protocolli di trasferimento sicuri.	SODDISFATTO
FROb153	Il Sistema deve registrare lo stato dell'analisi nel sistema di persistenza impostandolo a "PENDING" a seguito dell'inizializzazione corretta di tutti i servizi esterni.	SODDISFATTO
FROb154	Il Sistema deve associare univocamente l'ID dell'analisi al repository oggetto dell'audit e all'identificativo dell'utente richiedente.	SODDISFATTO
FROb155	Il Sistema deve persistere i metadati di avvio, inclusi l'hash del commit analizzato e il timestamp di sistema.	SODDISFATTO
FROb156	In caso di errore critico durante la scrittura dello stato (UC22.0.1), l'Orchestratore deve tentare una procedura di "Rollback" informando gli strumenti esterni di annullare l'analisi.	SODDISFATTO
FROb157	Il Sistema deve registrare nei log di audit ogni fallimento di persistenza dello stato, includendo lo stack trace dell'errore per finalità diagnostiche.	SODDISFATTO
FROb158	L'Orchestratore deve verificare regolarmente se gli strumenti esterni hanno terminato l'analisi del repository.	SODDISFATTO
FROb159	L'Orchestratore deve scaricare i risultati delle analisi non appena questi vengono messi a disposizione dagli strumenti esterni.	SODDISFATTO
FROb160	L'Orchestratore deve controllare che i file ricevuti siano completi e leggibili prima di utilizzarli.	SODDISFATTO

ID	Descrizione	Stato
FROb161	Il Sistema deve poter proseguire con la creazione del report anche se uno degli strumenti fallisce, utilizzando solo i dati recuperati con successo.	SODDISFATTO
FROb162	L'Orchestratore deve impostare un tempo massimo di attesa per le analisi, oltre il quale smette di aspettare lo strumento ritardatario.	SODDISFATTO
FROb163	Il Sistema deve segnalare all'interno del database se il report finale contiene solo dati parziali a causa di un problema tecnico.	SODDISFATTO
FROb164	Il Sistema deve unificare i dati provenienti dai diversi strumenti (codice, sicurezza, documentazione) in un unico documento di sintesi.	SODDISFATTO
FROb165	Il Sistema deve convertire i diversi formati dei dati ricevuti dagli strumenti esterni in un modello standard comune.	SODDISFATTO
FROb166	Il Sistema deve verificare che il report finale contenga tutte le informazioni essenziali (risultati, data, versione del codice) prima di procedere al salvataggio.	SODDISFATTO
FROb167	Il Sistema deve calcolare i punteggi di riepilogo generali basandosi sui singoli risultati ottenuti nelle varie aree analizzate.	SODDISFATTO
FROb168	Il Sistema deve archiviare il report in modo permanente, collegandolo correttamente al repository dell'utente.	SODDISFATTO
FROb169	Il Sistema deve modificare lo stato dell'analisi in "Completato" solo dopo aver confermato che il salvataggio dei dati è andato a buon fine.	SODDISFATTO
FROb170	Il Sistema deve informare l'utente con un messaggio di errore se un problema tecnico impedisce il salvataggio definitivo del report.	SODDISFATTO
FROb171	Il Sistema deve tenere traccia internamente dei motivi del fallimento del salvataggio per permettere controlli tecnici successivi.	SODDISFATTO
FROb172	In caso di errore nel salvataggio, il Sistema deve tentare di mantenere una copia temporanea del report per evitare la perdita totale dei dati elaborati.	SODDISFATTO

ID	Descrizione	Stato
FROb173	Il Sistema deve generare automaticamente un avviso per l'utente non appena il report di analisi è pronto e salvato correttamente.	SODDISFATTO
FROb174	La notifica inviata deve contenere un link o un pulsante che permetta all'utente di accedere direttamente alla visualizzazione del report.	SODDISFATTO
FROb175	Il Sistema deve includere nella notifica informazioni di base per identificare l'analisi, come il nome del repository e la data di esecuzione.	SODDISFATTO
FROb176	Il Sistema deve garantire che l'invio della notifica non interferisca con lo stato dell'analisi: se la notifica fallisce, il report deve comunque rimanere disponibile.	SODDISFATTO
FROb177	In caso di errore nell'invio del messaggio (es. email non raggiungibile), il Sistema deve segnare l'anomalia nei registri interni per permettere verifiche tecniche.	SODDISFATTO
FROb178	Il Sistema deve tentare nuovamente l'invio della notifica per un numero limitato di volte in caso di problemi temporanei di rete.	SODDISFATTO
FROb179	Il Sistema deve esporre per ogni elemento selezionato della lista: nome del repository, URL di riferimento e data dell'ultima analisi.	SODDISFATTO
FROb180	Il Sistema deve richiedere l'inserimento della password attuale come verifica di identità obbligatoria prima di avviare la cancellazione dell'account.	SODDISFATTO
FROb181	Il Sistema deve mostrare un avviso di irreversibilità prima della cancellazione definitiva del profilo, consentendo l'annullamento dell'operazione.	SODDISFATTO
FROb182	A seguito della cancellazione del profilo, il Sistema deve rimuovere i dati personali e le associazioni OAuth, invalidando ogni credenziale di accesso precedente.	SODDISFATTO
FRDe183	Il Sistema deve trasformare il codice provvisorio fornito da GitHub in una chiave di accesso permanente per poter leggere i repository.	NON SODDISFATTO
FRDe184	Il Sistema deve proteggere la chiave di accesso di GitHub nascondendola tramite cifratura prima di salvarla nei propri archivi.	NON SODDISFATTO

ID	Descrizione	Stato
FRDe185	Il Sistema deve collegare la chiave di GitHub in modo esclusivo al profilo dell'utente che ha autorizzato l'operazione.	NON SODDISFATTO
FRDe186	Il Sistema deve annullare il collegamento e chiedere all'utente di rifare la procedura se la chiave provvisoria risulta scaduta o non valida.	NON SODDISFATTO
FROb187	Il Sistema deve consentire all'Utente Autorizzato la visualizzazione del dettaglio di una singola remediation relativa all'analisi del codice.	SODDISFATTO
FROb188	Il Sistema deve includere nel dettaglio della remediation del codice il titolo, la descrizione, la tipologia di criticità e il livello di severità.	SODDISFATTO
FROb189	Il Sistema deve consentire all'Utente Autorizzato la visualizzazione del dettaglio di una singola remediation relativa all'analisi della sicurezza.	SODDISFATTO
FROb190	Il Sistema deve includere nel dettaglio della remediation di sicurezza il titolo, la descrizione, la tipologia di vulnerabilità e il livello di severità.	SODDISFATTO
FROb191	Il Sistema deve consentire all'Utente Autorizzato la visualizzazione del dettaglio di una singola remediation relativa all'analisi della documentazione.	SODDISFATTO
FROb192	Il Sistema deve includere nel dettaglio della remediation documentale il titolo, la descrizione e la tipologia di rilievo documentale.	SODDISFATTO
FRDe193	Il Sistema deve consentire all'Utente Autorizzato di accettare una remediation relativa all'analisi del codice.	NON SODDISFATTO
FRDe194	Il Sistema deve applicare automaticamente alla codebase le modifiche previste dalla remediation del codice accettata.	NON SODDISFATTO
FRDe195	Il Sistema deve aggiornare lo stato della remediation del codice come "eseguita" nella dashboard a seguito dell'applicazione riuscita.	NON SODDISFATTO
FRDe196	Il Sistema deve gestire errori durante l'applicazione della remediation del codice notificando il fallimento all'utente e mantenendo invariata la codebase.	NON SODDISFATTO

ID	Descrizione	Stato
FRDe197	Il Sistema deve consentire all'Utente Autorizzato di rifiutare una remediation relativa all'analisi del codice.	NON SODDISFATTO
FRDe198	Il Sistema deve aggiornare lo stato della remediation del codice come "rifiutata" nella dashboard senza apportare modifiche al repository.	NON SODDISFATTO
FRDe199	Il Sistema deve consentire all'Utente Autorizzato di accettare una remediation relativa all'analisi della sicurezza.	NON SODDISFATTO
FRDe200	Il Sistema deve applicare le patch o le configurazioni di sicurezza previste dalla remediation di sicurezza accettata.	NON SODDISFATTO
FRDe201	Il Sistema deve aggiornare lo stato della remediation di sicurezza come "eseguita" nella dashboard a seguito dell'applicazione riuscita.	NON SODDISFATTO
FRDe202	Il Sistema deve gestire errori durante l'applicazione della remediation di sicurezza notificando il fallimento all'utente.	NON SODDISFATTO
FRDe203	Il Sistema deve consentire all'Utente Autorizzato di rifiutare una remediation relativa all'analisi della sicurezza.	NON SODDISFATTO
FRDe204	Il Sistema deve aggiornare lo stato della remediation di sicurezza come "rifiutata" nella dashboard senza modificare il repository.	NON SODDISFATTO
FRDe205	Il Sistema deve consentire all'Utente Autorizzato di accettare una remediation relativa all'analisi della documentazione.	NON SODDISFATTO
FRDe206	Il Sistema deve applicare automaticamente ai file documentali le modifiche previste dalla remediation documentale accettata.	NON SODDISFATTO
FRDe207	Il Sistema deve aggiornare lo stato della remediation documentale come "eseguita" nella dashboard a seguito dell'applicazione riuscita.	NON SODDISFATTO
FRDe208	Il Sistema deve gestire errori durante l'applicazione della remediation documentale notificando il fallimento all'utente.	NON SODDISFATTO

ID	Descrizione	Stato
FRDe209	Il Sistema deve consentire all'Utente Autorizzato la visualizzazione del dettaglio di una singola remediation relativa all'analisi della documentazione.	NON SODDISFATTO
FRDe210	Il Sistema deve consentire all'Utente Autorizzato di rifiutare una remediation relativa all'analisi della documentazione.	NON SODDISFATTO
FRDe211	Il Sistema deve aggiornare lo stato della remediation documentale come "rifiutata" nella dashboard a seguito del rifiuto confermato dall'utente.	NON SODDISFATTO
FRDe212	Il Sistema deve garantire che il rifiuto di una remediation documentale non comporti alcuna modifica ai file sorgente o di documentazione del repository.	NON SODDISFATTO
FRDe213	Il Sistema deve rimuovere la remediation rifiutata dalla lista delle azioni pendenti dell'area "Documentazione" o marcarla visivamente come scartata.	NON SODDISFATTO
FRDe214	Il Sistema deve mostrare all'Utente Autorizzato una conferma visiva dell'avvenuto rifiuto della proposta correttiva.	NON SODDISFATTO
FROb215	Il Sistema deve consentire all'Utente Avanzato di richiedere un'analisi per un repository GitHub privato a condizione che l'integrazione GitHub sia attiva.	SODDISFATTO
FROb216	Il Sistema deve validare la presenza di un'integrazione GitHub valida prima di accettare la richiesta di analisi per una risorsa privata.	SODDISFATTO
FROb217	Il Sistema deve inibire la richiesta di analisi privata se l'utente non seleziona almeno un'area di interesse (Codice, Sicurezza, Documentazione).	SODDISFATTO
FROb218	Il Sistema deve consentire all'Utente Avanzato di inserire l'URL di un repository privato di sua proprietà nel proprio catalogo personale.	SODDISFATTO
FROb219	Il Sistema deve impedire l'inserimento di un URL repository già presente nel catalogo personale dell'Utente Avanzato, notificando la duplicazione.	SODDISFATTO
FROb220	Il Sistema deve ordinare l'elenco dei repository privati registrati in ordine decrescente rispetto a data di inserimento.	SODDISFATTO

ID	Descrizione	Stato
FROb221	Il Sistema deve visualizzare un'informativa specifica che suggerisce l'inserimento della prima risorsa qualora il catalogo privato risulti vuoto.	SODDISFATTO
FROb222	Il Sistema deve esporre all'Utente Avanzato la lista dei repository privati registrati, includendo per ciascuno il nome e l'URL della risorsa.	SODDISFATTO
FROb223	Il Sistema deve consentire all'Utente Avanzato di avviare la procedura di rimozione di un repository dal proprio catalogo privato.	SODDISFATTO
FROb224	Il Sistema deve consentire la rimozione di un repository dal catalogo privato previa conferma esplicita dell'utente.	SODDISFATTO
FROb225	In caso di annullamento della procedura di rimozione, il Sistema deve garantire l'integrità del catalogo mantenendo la risorsa selezionata.	SODDISFATTO
FROb226	Il Sistema deve mostrare all'Utente Avanzato l'elenco dei profili autorizzati alla consultazione dei report per un repository privato selezionato in ordine alfabetico.	SODDISFATTO
FROb227	Il Sistema deve informare l'utente proprietario qualora l'accesso ai report di un repository privato sia limitato esclusivamente al suo profilo.	SODDISFATTO
FROb228	Il Sistema deve esporre all'Utente Avanzato le informazioni identificative di ogni profilo autorizzato presente nella lista, includendo lo username e/o l'indirizzo email associato.	SODDISFATTO
FROb229	Il Sistema deve deve consentire l'aggiunta di un nuovo profilo autorizzato alla consultazione dei report per un repository privato.	SODDISFATTO
FROb230	Il Sistema deve consentire l'aggiunta di un utente autorizzato tramite l'inserimento dello username o dell'indirizzo email del profilo destinatario.	SODDISFATTO
FROb231	Il Sistema deve inibire la procedura e mostrare un messaggio di errore qualora l'identificativo inserito per l'autorizzazione non rispetti il formato previsto.	SODDISFATTO

ID	Descrizione	Stato
FROb232	Il Sistema deve validare che l'identificativo inserito corrisponda a un profilo effettivamente registrato nella piattaforma.	SODDISFATTO
FROb233	Il Sistema deve impedire l'autorizzazione multipla del medesimo profilo per lo stesso repository privato.	SODDISFATTO
FROb234	Il Sistema deve mostrare un avviso di obbligatorietà e inibire l'aggiunta qualora il campo identificativo risulti vuoto al momento della conferma.	SODDISFATTO
FROb235	Il Sistema deve consentire all'Utente Avanzato di selezionare un utente dalla lista e avviare la procedura di revoca dei suoi permessi.	SODDISFATTO
FROb236	Il Sistema deve consentire la revoca dei permessi di consultazione per un utente precedentemente autorizzato a seguito di conferma del proprietario.	SODDISFATTO
FROb237	Il Sistema deve consentire la rimozione di una raccolta di report senza che questo comporti l'eliminazione dei singoli report di analisi in essa contenuti.	SODDISFATTO
FROb238	Il Sistema deve richiedere una conferma esplicita prima di procedere con l'eliminazione definitiva di una raccolta dal profilo.	SODDISFATTO
FROb239	Il Sistema deve consentire di annullare la procedura di rimozione della raccolta, mantenendola inalterata nel sistema.	SODDISFATTO

5.2 Stato Attuale dei Requisiti di Qualità

ID	Descrizione	Stato
QROb1	L'architettura deve garantire un'alta coesione e un basso accoppiamento tra l'orchestratore NestJS e gli agenti Python, verificabile tramite revisione dei diagrammi UML2.5.	SODDISFATTO
QROb2	Il sistema deve garantire tempi di risposta della dashboard web ottimizzati, minimizzando il carico computazionale lato client durante il rendering dei report di audit.	SODDISFATTO
QROb3	Ogni componente software deve essere testabile isolatamente; la logica di business deve essere separata dalle interfacce di comunicazione (API/Database).	SODDISFATTO
QROb4	È necessario rispettare rigorosamente le metriche di qualità del codice (complessità ciclomatica, duplicazione) definite nelle **Norme di Progetto**.	SODDISFATTO
QROb5	Il team deve svolgere un'attività di analisi preliminare includendo Design Thinking, User Story Mapping, Business Requirements e Diagrammi UML degli Use Case	SODDISFATTO
QROb6	Deve essere fornita documentazione tecnica tramite standard OpenAPI 3.0 (Swagger) per le API e documentazione del codice sorgente tramite TypeDoc	SODDISFATTO
QROb7	Deve essere fornito un Manuale Utente come parte integrante della fornitura finale	SODDISFATTO
QROb8	Al termine del progetto deve essere consegnato un MVP funzionante accompagnato da una Demo Live e dallo Schema Design relativo alla base dati	SODDISFATTO
QROb9	Il codice prodotto deve raggiungere una copertura minima del 70% tramite test di unità automatizzati misurati con Jest	SODDISFATTO
QROb10	Il codice sorgente deve essere versionato utilizzando Git (v2.40+) seguendo la branching strategy definita nelle NdP	SODDISFATTO

ID	Descrizione	Stato
QRObl1	L’analisi di sicurezza deve essere conforme agli standard OWASP Top 10 (v2021 o successivi)	SODDISFATTO

5.3 Stato Attuale dei Requisiti di Vincolo

ID	Descrizione	Stato
VRObi	L'applicativo deve essere strutturato in moduli indipendenti, garantendo che l'aggiunta di un nuovo agente di analisi avvenga senza richiedere modifiche al codice sorgente degli agenti già esistenti	SODDISFATTO
VRObi	Deve essere fornito un sistema di Bug Reporting strutturato su GitHub Issues per tracciare e gestire le anomalie tramite apposite label	SODDISFATTO
VRObi	Il Back-end e l'Orchestratore devono essere sviluppati utilizzando il framework NestJS v10+	SODDISFATTO
VRObi	L'interfaccia Front-end deve essere sviluppata utilizzando la libreria React v18.3+	SODDISFATTO
VRObi	Gli agenti di analisi devono essere sviluppati utilizzando il linguaggio Python v3.12+	SODDISFATTO
VRObi	L'architettura deve essere ospitata su infrastruttura cloud AWS, utilizzando esclusivamente gli account IAM forniti dall'azienda proponente	SODDISFATTO
VRObi	Devono essere utilizzate GitHub Actions per implementare pipeline di Continuous Integration e Continuous Deployment (CI/CD)	SODDISFATTO
VRObi	L'interfaccia web deve essere compatibile con Windows 10/11	SODDISFATTO
VRObi	L'interfaccia web deve essere compatibile con macOS 14+	SODDISFATTO
VRObi	L'interfaccia web deve essere compatibile con distribuzioni Linux (Ubuntu 22.04+)	SODDISFATTO
VRObi	L'interfaccia web deve essere compatibile su browser Chrome 120+	SODDISFATTO
VRObi	L'interfaccia web deve essere compatibile su browser Firefox 120+	SODDISFATTO

ID	Descrizione	Stato
VROb13	L'interfaccia web deve essere compatibile su browser Safari 17+	SODDISFATTO

5.4 Tabella Riassuntiva

TIPO	COMPLETATI	TOTALI	PERCENTUALE
FROb	185	185	100%
FRDe	3	35	8,57%
FROp	0	19	0%
QROb	11	11	100%
VROb	13	13	100%

Sono dunque stati soddisfatti tutti i requisiti obbligatori previsti, solo una piccola parte di quelli desiderabili ma nessun opzionale.