Relazione progetto d'esame: EduQuest

Mattia Gorla (Matr. 914259)
Sofia Fonte (Matr. 909420)
Nicolò Manca (Matr. 909468)
Giovanni Deiana (Matr. 913146)

15 Febbraio 2026

Prefazione

Il presente documento illustra le fasi di analisi, progettazione e sviluppo del sistema **EduQuest**.
Seguendo i principi metodologici appresi durante il corso di *Analisi e Progettazione del Software*, lo sviluppo è stato condotto con un approccio iterativo ispirato al **Processo Unificato (UP)** e supportato dalla pratica del **pair programming**, volta a ottimizzare la produzione del codice e la qualità degli artefatti.
Nello specifico, il documento descrive le prime due iterazioni del ciclo di vita del software. La prima è stata focalizzata sulla definizione dei requisiti e sulla creazione di un nucleo architetturale stabile, producendo una versione embrionale del sistema testabile tramite *Postman* ma non ancora fruibile dall'utente finale. La seconda iterazione ha invece riguardato l'implementazione dell'interfaccia grafica e la gestione della persistenza tramite database **MySQL**, nonché l'introduzione di funzionalità aggiuntive e di meccaniche di *gamification*, portando alla realizzazione di un'applicazione web completa e funzionante.
L'enfasi del progetto è posta sull'analisi e progettazione orientata agli oggetti, con l'obiettivo di creare un sistema affidabile, mantenibile e conforme alle *best practices* dello sviluppo moderno. Tramite l'utilizzo del framework **Spring Boot**, **EduQuest** si propone come una piattaforma di *gamification* scolastica che permette ai docenti di somministrare questionari interattivi, stimolando l'apprendimento attraverso una sana competizione tra gli studenti.
All'interno del documento viene dedicato ampio spazio alla modellazione tramite diagrammi **UML**, alle scelte architetturali basate sul pattern **MVC** e alle attività di testing. L'obiettivo primario non è stato la semplice fornitura di un software funzionante, ma l'applicazione rigorosa dei concetti di *Ingegneria del Software* necessari a garantire la solidità e la correttezza dell'intera architettura.

Indice

Prefazione **1**

1 Requisiti **4**
1.1 Introduzione . . . 4
1.2 Regole di dominio . . . 5
1.2.1 Glossario . . . 5
1.3 Requisiti del sistema . . . 6
1.3.1 Requisiti funzionali . . . 6
1.3.2 Requisiti non funzionali . . . 8
1.4 Modello dei Casi d'Uso . . . 8
1.4.1 Diagramma UML dei Casi d'Uso . . . 9
1.4.2 Descrizione degli Attori . . . 9

2 Iterazione 1 - Core Architetturale e Logica di Business **10**
2.1 Obiettivi dell'iterazione 1 . . . 10
2.2 Modello di Dominio . . . 11
2.3 Casi d'uso . . . 12
2.3.1 UC1: **CreaAccountStudente** . . . 12
2.3.2 UC2: **ModificaAccount** . . . 13
2.3.3 UC3: **CreaQuestionario** . . . 14
2.3.4 UC4: **ModificaQuestionario** . . . 15
2.4 Diagrammi di Sequenza di Sistema (SSD) e Contratti . . . 17
2.4.1 UC1: CreaAccountStudente . . . 17
2.4.1.1 Diagrammma di sequenza di sistema . . . 17
2.4.1.2 Contratti delle operazioni . . . 17
2.4.2 UC2: ModificaAccount . . . 19
2.4.2.1 Diagrammma di sequenza di sistema . . . 19
2.4.3 UC3: CreaQuestionario . . . 20
2.4.3.1 Diagrammma di sequenza di sistema . . . 20
2.4.3.2 Contratti delle operazioni . . . 20
2.4.4 UC4: ModificaQuestionario . . . 22
2.4.4.1 Diagramma di sequenza di sistema . . . 22
2.5 Progettazione ed implementazione . . . 23
2.5.1 Architettura del sistema . . . 23
2.5.2 Tecnologie e Strumenti . . . 24
2.5.3 Diagrammi di Sequenza . . . 25
2.5.4 Diagramma delle Classi di Progetto . . . 26
2.5.5 Note sull'implementazione . . . 29
2.6 Verifica e testing . . . 29
2.6.1 Validazione tramite Postman . . . 30

3 Iterazione 2 - Compitini, esercitazioni e gamification **31**
3.1 Obiettivi dell'iterazione 2 . . . 31
3.2 Modello di dominio . . . 31
3.3 Realizzazione dei Casi d'Uso e artefatti correlati . . . 32
3.4 Progettazione ed implementazione . . . 33
3.4.1 Integrazione del front-end con Thymeleaf . . . 33
3.5 Macchina a stati e diagramma di attività . . . 34
3.5.1 Diagramma della Macchina a Stati: Compilazione . . . 34
3.5.2 Diagramma di attività: ricerca e filtraggio Questionari . . . 35
3.6 Diagramma delle Classi di Progetto . . . 36
3.7 Note sull'implementazione . . . 39
3.8 Soddisfacimento dei requisiti non funzionali . . . 41
3.9 Verifica e testing . . . 41
3.9.1 Estensione del testing unitario . . . 41
3.9.2 Stress Test: Modulo Registrazione Utenti . . . 42
3.9.2.1 Configurazione del test . . . 42
3.9.2.2 Analisi dei risultati . . . 42

4 Analisi Statica **43**
4.1 Analisi con SonarQube . . . 43
4.2 Qualità Metrica e Analisi Strutturale con Understand . . . 44
4.2.1 Complessità e Distribuzione del Codice . . . 44
4.2.2 Architettura . . . 44
4.2.3 Debito Tecnico . . . 45
4.2.4 Sintesi Valutativa . . . 46

5 Conclusioni **47**
5.1 Organizzazione del lavoro e metodologia . . . 47
5.2 Il ruolo dell'AI nel progetto . . . 47
5.3 Sviluppi futuri . . . 48
5.4 Considerazioni finali . . . 48

Capitolo 1

Requisiti

1.1 Introduzione

Negli ultimi anni, il settore dell'istruzione ha conosciuto una progressiva integrazione delle tecnologie digitali come supporto ai processi di insegnamento e apprendimento. In particolare, l'adozione di piattaforme software dedicate alla didattica ha permesso di affiancare ai metodi tradizionali nuovi strumenti interattivi, capaci di favorire una maggiore partecipazione degli studenti e di rendere più efficace la trasmissione dei contenuti. Tuttavia, uno degli aspetti più critici della didattica, soprattutto in contesti scolastici, rimane il mantenimento di un elevato livello di coinvolgimento e motivazione da parte degli studenti.
In questo contesto, la *gamification* si è affermata come un approccio promettente, in grado di applicare dinamiche tipiche del gioco — quali competizione, punteggi e obiettivi — a contesti non ludici, con l'obiettivo di stimolare l'interesse e migliorare l'esperienza di apprendimento. L'utilizzo consapevole di tali meccaniche in ambito educativo consente di trasformare attività valutative tradizionali, come questionari e test, in esperienze più coinvolgenti, favorendo la partecipazione attiva degli studenti e supportando i docenti nel monitoraggio del progresso formativo.
Il problema affrontato dal presente progetto consiste pertanto nella progettazione e realizzazione di un sistema software in grado di supportare l'attività didattica attraverso la somministrazione di questionari interattivi, integrando al contempo meccaniche di *gamification* finalizzate ad aumentare il coinvolgimento degli studenti e a favorire una partecipazione più attiva.

Il presente documento illustra l'intero ciclo di vita del software EduQuest, dall'analisi del dominio fino alle fasi di verifica e testing. La trattazione segue fedelmente l'approccio iterativo adottato, articolandosi in due iterazioni principali per evidenziare l'evoluzione incrementale del sistema e delle scelte progettuali.
La sezione iniziale è dedicata alla definizione dei requisiti e all'analisi del dominio: attraverso la redazione di un apposito glossario e del modello concettuale, vengono delineati i confini del sistema e il linguaggio comune agli attori coinvolti. Successivamente, vengono formalizzati i requisiti funzionali e non funzionali, supportati dalla modellazione dei casi d'uso per descrivere le interazioni previste tra gli utenti e la piattaforma.

1.2 Regole di dominio

Il dominio applicativo di EduQuest si colloca nell'ambito della didattica digitale, con un focus specifico sulla valutazione formativa e sul potenziamento della motivazione dello studente attraverso la *gamification*.
Come precedentemente introdotto, il problema centrale del dominio scolastico moderno non risiede solo nella trasmissione dell'informazione, ma nel mantenimento di un elevato livello di attenzione e partecipazione. In questo scenario, il sistema si propone di modellare un ambiente in cui l'attività di verifica (il questionario) perde la sua connotazione puramente valutativa per trasformarsi in un'esperienza interattiva.
L'analisi del dominio mira a mappare i concetti dell'insegnamento in elementi software concreti, distinguendo due aree principali:

- **La parte didattica**: riguarda la creazione, gestione e compilazione dei quiz;
- **La parte di gioco**: comprende i punteggi, i feedback (nel caso di *esercitazioni*), i punti bonus e le classifiche, necessari per rendere lo studio più stimolante e interattivo.

Definire con precisione questi aspetti è essenziale per creare un sistema che non sia solo un contenitore di domande, ma una piattaforma completa capace di seguire e valorizzare i progressi dello studente.

1.2.1 Glossario

Al fine di garantire una comprensione univoca dei requisiti e delle entità del sistema, viene di seguito riportato il glossario dei termini principali. Questi termini verranno utilizzati in modo consistente in tutta la documentazione tecnica e nei diagrammi UML.

Termine	Descrizione
Utente Non Loggato	Attore che rappresenta un utente non registrato al sistema EduQuest.
Docente	Attore responsabile della creazione, modifica e gestione dei contenuti didattici (Questionari e Domande).
Studente	Attore che fruisce dei Questionari, accumula punteggi e visualizza il proprio posizionamento nelle classifiche.
Questionario	L'entità centrale del sistema; consiste in un insieme organizzato di domande su un determinato argomento o modulo didattico creato da un Docente.
Domanda	Singola unità informativa di un Questionario, composta da un testo (quesito), da un insieme di opzioni di risposta, di cui una o più corrette e da dei punti (assegnati nel caso di selezione della risposta corretta). Nella corrente implementazione di EduQuest vi sono 3 tipi possibili di Domanda: vero/falso, a risposta multipla con una sola risposta corretta e a risposta multipla con multiple risposte corrette.
Risposta	Istanza legata ad una specifica domanda, di cui rappresenta appunto una possibile risposta. È formata da un campo di testo e un valore booleano che indica se sia la/una risposta corretta per la domanda associata
Compilazione	Istanza che rappresenta lo svolgimento (o meglio compilazione) di un Questionario da parte di uno Studente. Registra le risposte fornite e il punteggio ottenuto.

Termine	Descrizione
Compitino	Istanza che rappresenta un Questionario con una finestra temporale e limitati tentativi di compilazione
Compitino scaduto	Compitino la cui finestra temporale non è più valida e che non può più essere compilato.
Esercitazione	Istanza che rappresenta un Questionario alla cui fine vengono mostrati l'elenco delle domande con la risposta data dallo Studente, per ogni domanda un Feedback e delle note generali lasciate dal Docente creatore dell'Esercitazione.
Feedback	Testo di commento associato ad una specifica Domanda di una Esercitazione, definito dal Docente creatore.
Punteggio	Valore numerico calcolato al termine di una compilazione, equivalente alla somma dei punti delle domande a cui è stata data la risposta corretta.
EduPoints	Valore numerico che rappresenta punti bonus ottenibili dalle varie dinamiche di gamification.

Tabella 1.1: Glossario del sistema EduQuest

1.3 Requisiti del sistema

1.3.1 Requisiti funzionali

I requisiti funzionali descrivono i servizi che il sistema EduQuest deve offrire agli utenti. Di seguito sono elencati i requisiti identificati, suddivisi per aree di responsabilità del sistema.

Codice	Descrizione del Requisito
Accesso e Gestione Profilo	
RF1	Il sistema deve permettere all'utente non loggato di creare un account, distinguendo tra il ruolo di Studente e Docente.
RF2	Il sistema deve permettere l'autenticazione tramite Login e la disconnessione (LogOut).
RF3	Ogni utente autenticato deve poter modificare i propri dati o eliminare il proprio profilo.
Creazione Questionari (Docente)	
RF4	Il sistema deve permettere al Docente di creare, modificare e cancellare Questionari interattivi impostando livelli di difficoltà.
RF5	Il sistema deve permettere al Docente di inserire, editare o rimuovere Domande all'interno di un Questionario.
RF6	Il sistema deve permettere al Docente di assegnare un punteggio specifico a ogni singola domanda.

Codice	Descrizione del Requisito
RF7	Il sistema deve supportare tre tipologie di Domande: a risposta multipla singola, vero/falso, e risposta multipla con più risposte corrette.
RF8	Il Docente deve poter inserire, editare o rimuovere Risposte all'interno di una Domanda di un Questionario.
RF9	Il Docente deve poter impostare e modificare la/e risposte corrette per una Domanda di un Questionario.
RF10	Il Docente deve poter definire la modalità del questionario: "Compitino" (con finestre temporali e tentativi limitati) o "Esercitazione" (con feedback generale e ad ogni Domanda).
Compilazioni	
RF11	Lo studente deve poter ricercare i Questionari disponibili per la compilazione.
RF12	Il sistema deve permettere l'avvio di una Compilazione per un determinato quiz.
RF13	Il sistema deve gestire la sottomissione delle risposte, salvandole in modo incrementale.
RF14	Al termine di una Compilazione, il sistema deve calcolare il punteggio ottenuto, aggiornare correttamente la media dei punteggi di uno studente e salvare questi due dati.
RF15	Al termine di una Compilazione, il sistema deve mostrare una schermata di riepilogo con il punteggio ottenuto.
RF16	Al termine della Compilazione di una "Esercitazione", il sistema deve fornire un feedback (generale e per singola domanda).
Ricerca	
RF17	Il sistema deve filtrare i risultati della ricerca per uno Studente escludendo i Compitini scaduti o i Compitini per cui lo Studente ha già effettuato il numero massimo di tentativi di compilazione.
RF18	Il sistema deve mostrare ad un Docente tutti i questionari da lui creati.
RF19	Il sistema deve filtrare i risultati di una ricerca per una keyword specificata.
Dashboards	
RF20	Il sistema deve permettere al Docente di consultare una Dashboard con varie statistiche sui Questionari, tra le quali il Questionario con il maggior numero di Compilazioni.
RF21	Il sistema deve mostrare nella Dashboard di un Docente i Compitini non scaduti da lui creati e il numero di volte che è stato compilato.
Gamification	

Codice	Descrizione del Requisito
RF22	Il sistema deve calcolare, al termine della compilazione, gli eventuali EduPoints da assegnare (basandosi sulla percentuale di correttezza della compilazione).
RF23	Il sistema deve aggiornare (se necessario) correttamente gli EduPoints di uno Studente dopo ogni chiusura di Compilazione.
RF24	Il sistema deve permettere al Docente di assegnare degli EduPoints ad un Compitino.
RF25	Il sistema deve verificare, tra i Compitini scaduti, quali non hanno ancora assegnato gli EduPoints e assegnarli ai tre Studenti che hanno ottenuto i punteggi più alti nelle Compilazioni del Compitino. Per ogni Studente viene considerata esclusivamente la Compilazione con il punteggio più alto.
RF26	Il sistema deve generare due classifiche, basate rispettivamente sulla media dei punteggi degli Studenti e sugli EduPoints degli Studenti, entrambe ordinate in modo decrescente rispetto ai punteggi o agli EduPoints.

Tabella 1.2: Requisiti funzionali del sistema EduQuest

1.3.2 Requisiti non funzionali

I requisiti non funzionali descrivono proprietà e vincoli del sistema EduQuest. Di seguito sono elencati i requisiti non funzionali identificati:

Codice	Descrizione del Requisito
RNF1	**Usabilità**: l'interfaccia deve essere intuitiva e semplice, garantendo un'esperienza d'uso fluida sia per studenti che per docenti.
RNF2	**Performance**: le pagine del sistema devono essere caricate in un tempo medio inferiore ai 2 secondi.
RNF3	**Sicurezza**: Gestione sicura degli account e protezione dei dati personali in conformità ai principi del GDPR.
RNF4	**Scalabilità**: Il sistema deve supportare sessioni con classi numerose (centinaia di utenti simultanei) ed eventi sincroni.
RNF5	**Affidabilità**: disponibilità del servizio >99% e presenza di meccanismi di salvataggio per evitare la perdita di dati durante i quiz.

Tabella 1.3: Requisiti non funzionali del sistema EduQuest

1.4 Modello dei Casi d'Uso

Il Modello dei Casi d'Uso descrive le funzionalità del sistema dal punto di vista degli attori esterni. Esso permette di chiarire come il software risponde alle necessità degli attori esterni, definendo i confini del sistema e le interazioni principali.

1.4.1 Diagramma UML dei Casi d'Uso

Il seguente diagramma UML dei Casi d'Uso offre una panoramica delle funzionalità di EduQuest. Gli attori sono stati identificati in base al loro ruolo e ai permessi di accesso: l'**Utente Non Loggato** ha accesso alle funzioni di registrazione, mentre lo **Studente** e il **Docente** condividono le funzioni di gestione profilo, differenziandosi nella gestione e fruizione dei questionari.

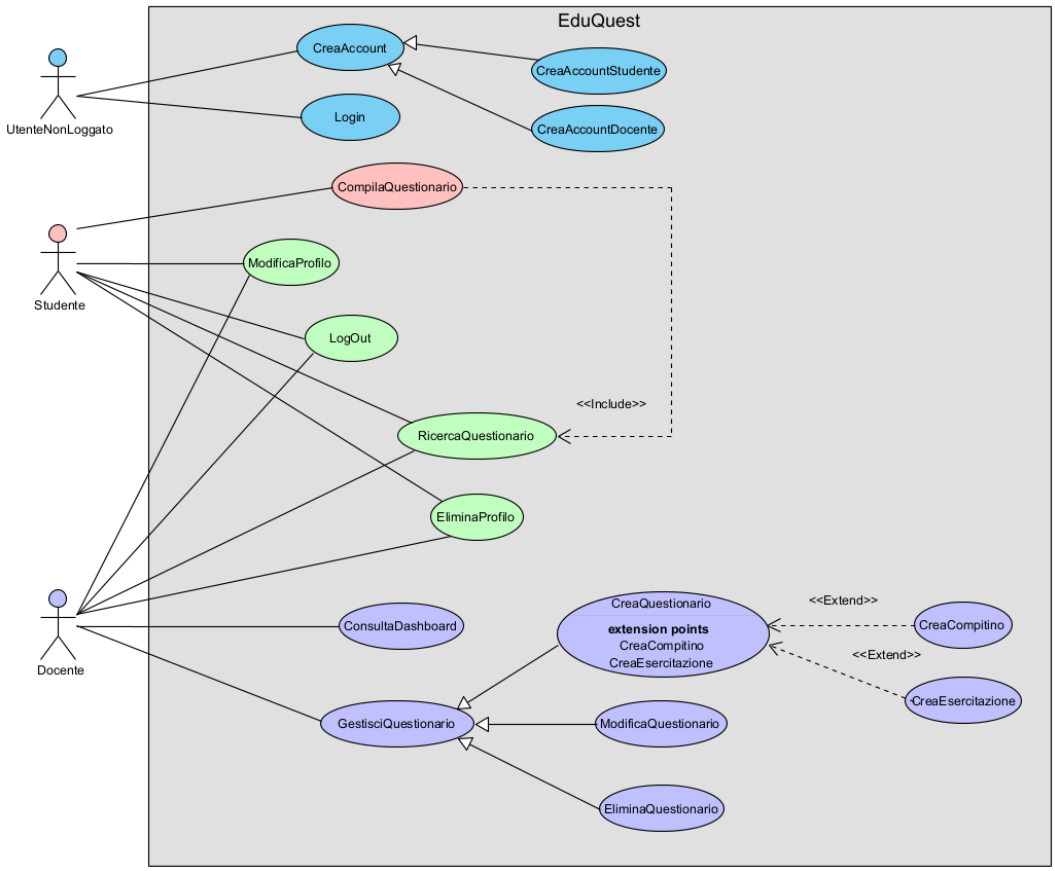

Figura 1.1: Diagramma dei Casi d'Uso - Sistema EduQuest

1.4.2 Descrizione degli Attori

Di seguito vengono specificati i ruoli e le responsabilità degli attori coinvolti:

- **Utente Non Loggato**: utente che non ha ancora effettuato l'accesso. Può solo registrarsi (scegliendo il ruolo tra Studente e Docente) o autenticarsi.
- **Studente**: utente che cerca e completa i questionari. Il suo scopo è ottenere punteggi alti nelle compilazioni per scalare le classifiche e accumulare EduPoints.
- **Docente**: utente responsabile dei contenuti. Crea e gestisce i questionari e monitora l'andamento degli studenti tramite una dashboard dedicata.

Capitolo 2

Iterazione 1 - Core Architetturale e Logica di Business

2.1 Obiettivi dell'iterazione 1

La prima iterazione del progetto si concentra sulla creazione di un nucleo stabile del sistema. L'obiettivo primario non è fornire un'interfaccia completa per l'utente finale, ma implementare le logiche di business fondamentali, testare la solidità dell'architettura scelta e stabilire una prima forma di persistenza dei dati.
Nello specifico, gli obiettivi prefissati per questa fase sono:

- **Definizione dell'architettura**: configurazione dell'ambiente di sviluppo basato su Spring Boot e impostazione del pattern MVC (Model-View-Controller).
- **Implementazione del dominio base**: sviluppo delle entità principali relative ai profili e alla gestione dei questionari, delle domande e delle risposte.
- **Progettazione della persistenza iniziale**: configurazione della connessione a un database **MySQL** tramite l'integrazione del driver **mysql-connector-java**. La gestione dei dati è affidata a query SQL dirette, permettendo un controllo granulare sulle operazioni di inserimento e recupero.
- **Sviluppo dei Servizi REST**: creazione dei controller necessari per permettere la creazione di un questionario e la sua compilazione di base.
- **Validazione tramite testing**: verifica delle funzionalità implementate attraverso test di integrazione eseguiti con *Postman*, assicurando che lo scambio di dati in formato JSON avvenga correttamente tra client e server.
- **Gestione della logica di valutazione**: dell'algoritmo iniziale per il calcolo del punteggio basato sul numero di risposte risposte.

Al termine di questa iterazione, il sistema, purché privo di interfaccia grafica, disporrà di un'infrastruttura solida, in grado di comunicare con MySQL senza l'ausilio di ORM (Object-Relational Mapping) complessi, garantendo prestazioni elevate e una gestione trasparente delle transazioni.

2.2 Modello di Dominio

In questa prima iterazione, il Modello di Dominio si focalizza sulle entità indispensabili per il funzionamento del nucleo centrale di EduQuest. L'obiettivo è definire la struttura dei dati che verrà poi tradotta in tabelle MySQL e gestita tramite logica JDBC.

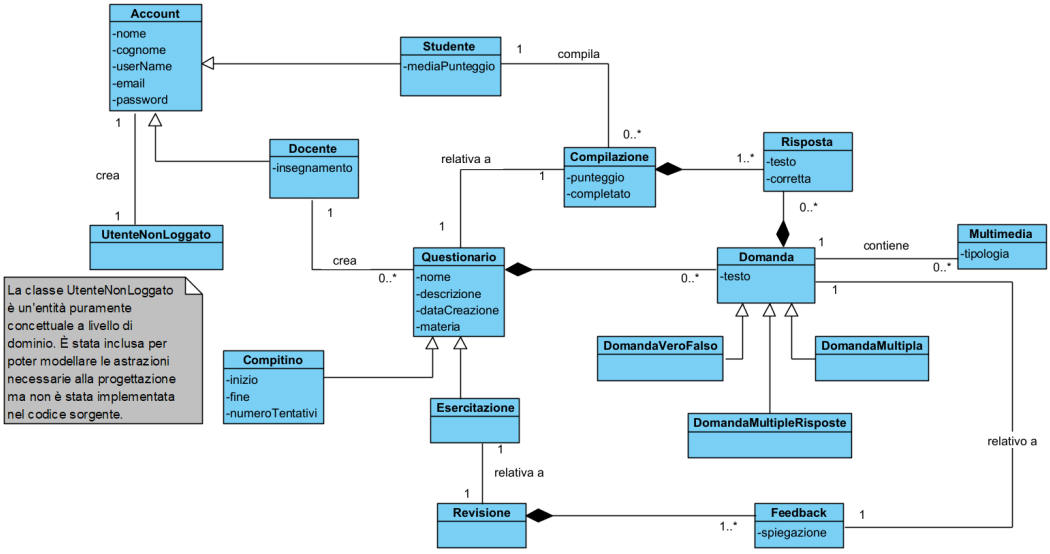

Figura 2.1: Modello di Dominio - Iterazione 1

Le entità principali e le loro interazioni in questa fase sono:

- **Gerarchia degli Account**: il sistema prevede una classe base **Account** (nome, cognome, email, etc.) da cui ereditano **Studente** (caratterizzato dalla *media punteggio*) e **Docente** (caratterizzato dall'*insegnamento*). È presente inoltre l'entità **UtenteNonLoggato**, utilizzata esclusivamente a fini concettuali per modellare la fase di creazione dell'Account.
- **Struttura del Questionario**: l'entità **Questionario** è creata dal Docente e aggrega un insieme di **Domande**. La classe **Domanda** è specializzata in tre tipologie (**VeroFalso, Multipla, MultipleRisposte**) per supportare i requisiti di sistema.
- **Logica di compilazione**: lo Studente compila un Questionario generando una **Compilazione**, che tiene traccia delle risposte date, del punteggio e dello stato di completamento.

Nota sulla modellazione iterativa: il diagramma include alcune classi concettuali che verranno approfondite nella seconda iterazione, quali **Compitino, Esercitazione e Feedback**. Queste entità sono state incluse del diagramma per delineare da subito il perimetro del sistema ma saranno debitamente raffinate e modellate nella successiva iterazione.

2.3 Casi d'uso

In questa sezione vengono analizzati alcuni casi d'uso di interesse per l'iterazione 1, descritti in formato dettagliato. Di seguito vengono riportate le descrizione in formato dettagliato per i casi d'uso **UC1 (CreaAccountStudente)**, **UC2 (ModificaAccount)**, **UC3 (CreaQuestionario)** e **UC4 (ModificaQuestionario)**.

2.3.1 UC1: CreaAccountStudente

Nome del caso d'Uso: CreaAccountStudente	
Portata:	Sistema EduQuest
Livello:	Obiettivo UtenteNonLoggato
Attore Primario:	UtenteNonLoggato
Parti interessate e interessi:	L'UtenteNonLoggato vuole creare un account Studente.
Pre-condizioni:	L'UtenteNonLoggato non deve possedere già un account a suo nome.
Post-condizioni:	L'UtenteNonLoggato ha creato un account Studente.
Scenario Principale di Successo:	1. Il caso d'uso inizia quando un UtenteNonLoggato entra nel sito; 2. Il sistema chiede se desidera registrarsi o accedere (effettuare il LogIn); 3. L'UtenteNonLoggato seleziona "Registra Studente"; 4. Il sistema chiede i dati dell'UtenteNonLoggato (nome, cognome, username, e-mail, password); 5. L'UtenteNonLoggato inserisce i dati richiesti; 6. Il sistema notifica che il proprio account Studente è stato creato con successo; 7. L'utente viene reindirizzato alla pagina iniziale.
Estensioni:	5a. Durante la registrazione vengono inseriti uno o più dati non validi 6a. Il sistema chiede nuovamente l'inserimento dei dati non validi 7a. L'UtenteNonLoggato inserisce i dati corretti Si torna al passo 6.
Extension Points:	-
Requisiti Speciali:	-
Frequenza di ripetizione:	La creazione di un Account viene effettuata una sola volta per UtenteNonLoggato.

Tabella 2.2: Descrizione in formato dettagliato UC1 - EduQuest

2.3.2 UC2: ModificaAccount

Nome del caso d'Uso: ModificaAccount	
Portata:	Sistema EduQuest
Livello:	Obiettivo Studente
Attore Primario:	Studente
Parti interessate e interessi:	Uno Studente vuole modificare le informazioni relative al proprio account.
Pre-condizioni:	Uno Studente deve avere un account, quindi essere registrato al sistema EduQuest
Post-condizioni:	Lo Studente ha modificato le informazioni relative al proprio account.
Scenario Principale di Successo:	1. Il caso d'uso inizia quando uno Studente entra nell'area privata del proprio profilo e seleziona "Modifica Account"; 2. Il sistema chiede quali informazioni si desidera modificare (tra nome, cognome, email, password); 3. Lo Studente seleziona le informazioni che vuole modificare; 4. Il sistema chiede di inserire le nuove informazioni; 5. Lo Studente inserisce le nuove informazioni; 6. Il sistema chiede la conferma o annullamento dell'inserimento delle nuove informazioni; 7. Lo Studente conferma l'inserimento; 8. Il sistema notifica che la modifica è avvenuta con successo.
Estensioni:	7a. Lo Studente annulla la modifica delle proprie informazioni 8a. Il sistema reindirizza lo Studente all'area privata del proprio profilo
Extension Points:	-
Requisiti Speciali:	-
Frequenza di ripetizione:	Potrebbe essere quasi ininterrotta.

Tabella 2.4: Descrizione in formato dettagliato UC2 - EduQuest

Nota: Il caso d'uso UC2 è stato descritto considerando lo Studente come attore primario, ma lo stesso processo si applica anche nel caso in cui l'attore primario sia un Docente.

2.3.3 UC3: CreaQuestionario

Nome del caso d'Uso: CreaQuestionario	
Portata:	Sistema EduQuest
Livello:	Obiettivo Docente
Attore Primario:	Docente
Parti interessate e interessi:	Un Docente vuole creare un Questionario
Pre-condizioni:	-
Post-condizioni:	Il Docente ha creato un questionario con successo.
Scenario Principale di Successo:	1. Il caso d'uso inizia quando il Docente accede alla sezione "Crea Questionario" del sistema; 2. Il sistema chiede il nome del questionario; 3. Il Docente inserisce il nome del questionario; 4. Il Docente seleziona di voler inserire una nuova domanda; 5. Il sistema chiede il testo della domanda; 6. Il Docente inserisce il testo della domanda; 7. Il sistema chiede il numero di risposte per la domanda; 8. Il Docente inserisce il numero di risposte; 9. Il sistema chiede il testo della risposta; 10. Il Docente inserisce il testo della risposta; I passi 9 e 10 si ripetono un numero di volte pari al numero di risposte inserito dal docente per la domanda. 11. Il sistema chiede qual è la risposta corretta alla domanda; 12. Il Docente seleziona la risposta corretta alla domanda; Si ripetono i passi dal 4 al 12 finché il docente non conferma di aver finito la creazione del questionario. 13. Il Docente conferma la creazione del questionario; 14. Il sistema notifica che il questionario è stato creato con successo.
Estensioni:	-
Extension Points:	-
Requisiti Speciali:	-
Frequenza di ripetizione:	Potrebbe essere ininterrotta

Tabella 2.6: Descrizione in formato dettagliato UC3 - EduQuest

2.3.4 UC4: ModificaQuestionario

Nome del caso d'uso: ModificaQuestionario	
Portata:	Sistema EduQuest
Livello:	Obiettivo Docente
Attore Primario:	Docente
Parti interessate e interessi:	Il Docente vuole modificare un Questionario già esistente.
Pre-condizioni:	• Deve esistere un'istanza di Questionario; • Per la modifica di domande, deve esistere almeno un'istanza di Ddomanda per l'istanza di Questionario scelta; • Per la modifica di una risposta, deve esistere almeno un'istanza di Risposta per l'istanza di Domanda scelta.
Post-condizioni:	Il Docente ha modificato con successo il Questionario.
Scenario principale di successo:	1. Il caso d'uso inizia quando il Docente accede alla sezione "Modifica Questionario" del sistema; 2. Il sistema chiede quale operazione si desidera eseguire; 3. Il Docente seleziona "Modifica nome"; 4. Il sistema chiede il nome da assegnare al questionario; 5. Il Docente inserisce il nome; 6. Il Docente conferma la modifica; 7. Il sistema notifica il docente che il questionario è stato modificato con successo.
Estensioni:	3a. Il Docente seleziona "Modifica domande"; 4a. Il sistema chiede quale domanda si desidera modificare; 5a. Il Docente seleziona la domanda da modificare; 6a. Il sistema chiede quale operazione di modifica si vuole effettuare; 7a1. Il Docente seleziona "Modifica testo domanda"; 8a1. Il sistema chiede il testo della domanda; 9a1. Il Docente inserisce il testo; 10a1. Il sistema notifica che la domanda è stata modificata con successo. 7a2. Il Docente seleziona "Modifica risposte"; 8a2. Il sistema chiede quale risposta si vuole modificare; 9a2. Il Docente seleziona la risposta da modificare; 10a2. Il sistema chiede il testo della risposta; 11a2. Il Docente inserisce il testo; 12a1. Il sistema notifica che la risposta è stata modificata con successo 7a3. Il Docente seleziona "Aggiungi risposta"; 8a3. Il sistema chiede il testo della risposta; 9a3. Il Docente inserisce il testo; 10a3. Il sistema notifica che la risposta 'e stata aggiunta con successo.

Nome del caso d'uso: ModificaQuestionario	
	7a4. Il Docente seleziona "Modifica risposta corretta"; 8a4. Il sistema chiede la risposta corretta corretta; 9a4. Il Docente seleziona la risposta corretta; 10a4. Il sistema notifica che la risposta corretta è stata modificata con successo. I passi dal 3a si ripetono finché il Docente non conferma di aver finito la modifica delle domande. Si torna al passo 6. 3b. Il Docente seleziona "Aggiungi domande"; 4b. Il Docente seleziona di voler inserire una nuova domanda; 5b. Il sistema chiede il testo della domanda; 6b. Il Docente inserisce il testo della domanda; 7b. Il sistema chiede il numero di risposte per la domanda; 8b. Il Docente inserisce il numero di risposte; 9b. Il sistema chiede il testo della risposta; 10b. Il Docente inserisce il testo della risposta; I passi 9b e 10b si ripetono un numero di volte pari al numero di risposte inserito dal docente per la domanda. 11b. Il sistema chiede qual è la risposta corretta alla domanda; 12b. Il Docente seleziona la risposta corretta alla domanda; Si ripetono i passi dal 4b al 12b finché il Docente non conferma di aver finito la modifica del questionario. Si torna al passo 6.
Extension Points:	-
Requisiti Speciali:	-
Frequenza di ripetizione:	Potrebbe essere ininterrotta

Tabella 2.7: Descrizione in formato dettagliato UC4 - EduQuest

2.4 Diagrammi di Sequenza di Sistema (SSD) e Contratti

In questa sezione viene formalizzato il comportamento del sistema in risposta agli eventi generati dagli attori. Coerentemente con l'approccio iterativo, la modellazione è stata limitata alle operazioni ritenute essenziali e derivate direttamente dai casi d'uso descritti in precedenza.
I **Diagrammi di Sequenza di Sistema** (SSD) illustrano, per ogni scenario, la sequenza di messaggi in ingresso al sistema, trattato come un'unica entità (black-box). Ai messaggi più significativi sono associati i relativi **Contratti delle operazioni**, che definiscono le variazioni di stato del sistema in termini di post-condizioni (creazione di oggetti, modifica di attributi o formazione di collegamenti).

2.4.1 UC1: CreaAccountStudente

2.4.1.1 Diagrammma di sequenza di sistema

Il diagramma di sequenza di sistema per il caso d'uso UC1 è il seguente:

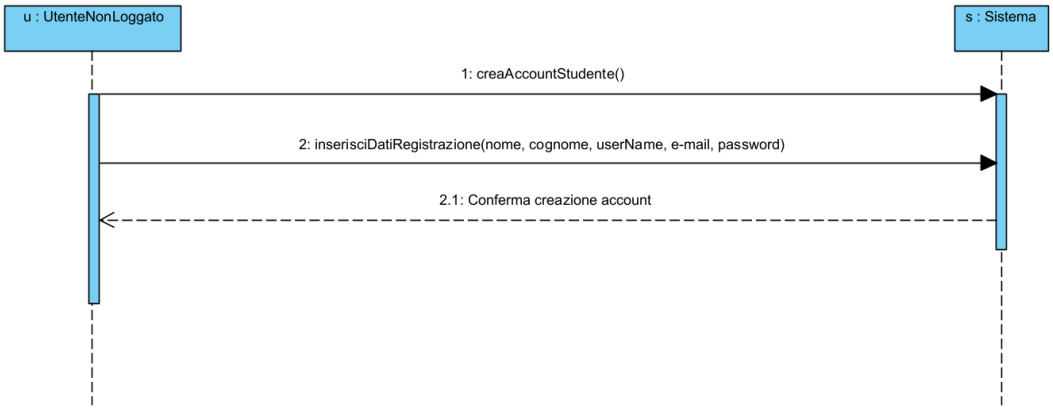

Figura 2.2: SSD - CreaAccountStudente

2.4.1.2 Contratti delle operazioni

InserisciDatiRegistrazione L'operazione di sistema *inserisciDatiRegistrazione* viene invocata quando l'utente inserisce i propri dati. Essa ha il compito di popolare gli attributi della nuova istanza di Account e di stabilire i collegamenti necessari tra l'entità generica e il profilo specifico che si sta creando.

Contratto: inserisciDatiRegistrazione	
Operazione:	`inserisciDatiRegistrazione(nome, cognome, userName, e-mail, password)`
Riferimenti:	UC1: creaAccount
Pre-condizioni:	È in corso la creazione di un account Studente **s** da parte di un UtenteNonLoggato **u**.

Postcondizioni:	• È stata creata un'istanza **a** di Account (creazione di oggetto); • **a**.nome è stato inizializzato a *nome* (modifica di attributo); • **a**.cognome è stato inizializzato a *cognome* (modifica di attributo); • **a**.userName è stato inizializzato a *username*(modifica di attributo); • **a**.e-mail è stato inizializzato a *e-mail* (modifica di attributo); • **a**.password è stato inizializzato a *password* (modifica di attributo); • È stata creata un'istanza **s** di Studente (creazione di oggetto); • L'istanza **a** di Account è stata associata all'istanza **s** di Studente (formazione di collegamento). • **s**.mediaPunteggio è stata inizializzata a 0.0 (modifica di attributo);

Tabella 2.8: Contratto dell'operazione inserisciDatiRegistrazione

CreaAccountStudente L'operazione di sistema *creaAccountStudente* formalizza la creazione di un nuovo profilo Studente all'interno del sistema EduQuest. Oltre alla creazione dell'istanza, l'operazione garantisce l'integrità dei dati inizializzando i parametri di gioco, come la media del punteggio, al valore predefinito di partenza.

Contratto: creaAccountStudente	
Operazione:	creaAccountStudente
Riferimenti caso d'uso:	creaAccount
Precondizioni:	È in corso la creazione di un Account **a** da parte di un UtenteNonLoggato **u**.
Postcondizioni:	• È stata creata un'istanza **a** di Account(creazione di oggetto); • **a**.ID è stato inizializzato univoco (modifica di attributo); • Gli altri attributi di **a** sono stati inizializzati (modifica di attributo); • L'istanza **a** di Account è stata associata all'istanza **u** di UtenteNonLoggato (formazione di collegamento).

Tabella 2.9: Contratto dell'operazione creaAccountStudente

2.4.2 UC2: ModificaAccount

2.4.2.1 Diagrammma di sequenza di sistema

Il diagramma di sequenza di sistema per il caso d'uso UC2 è il seguente:

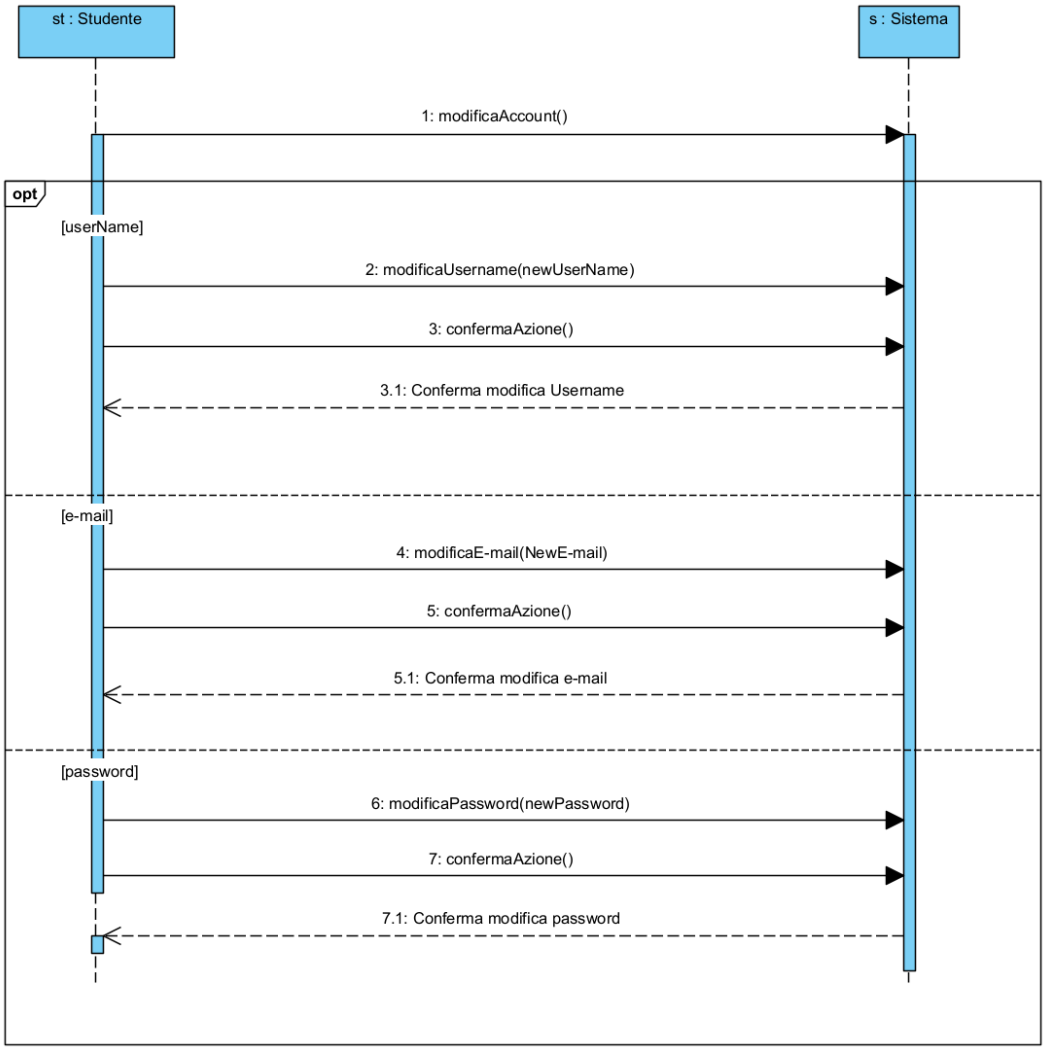

Figura 2.3: SSD - ModificaAccount

2.4.3 UC3: CreaQuestionario

2.4.3.1 Diagrammma di sequenza di sistema

Il diagramma di sequenza di sistema per il caso d'uso UC3 è il seguente:

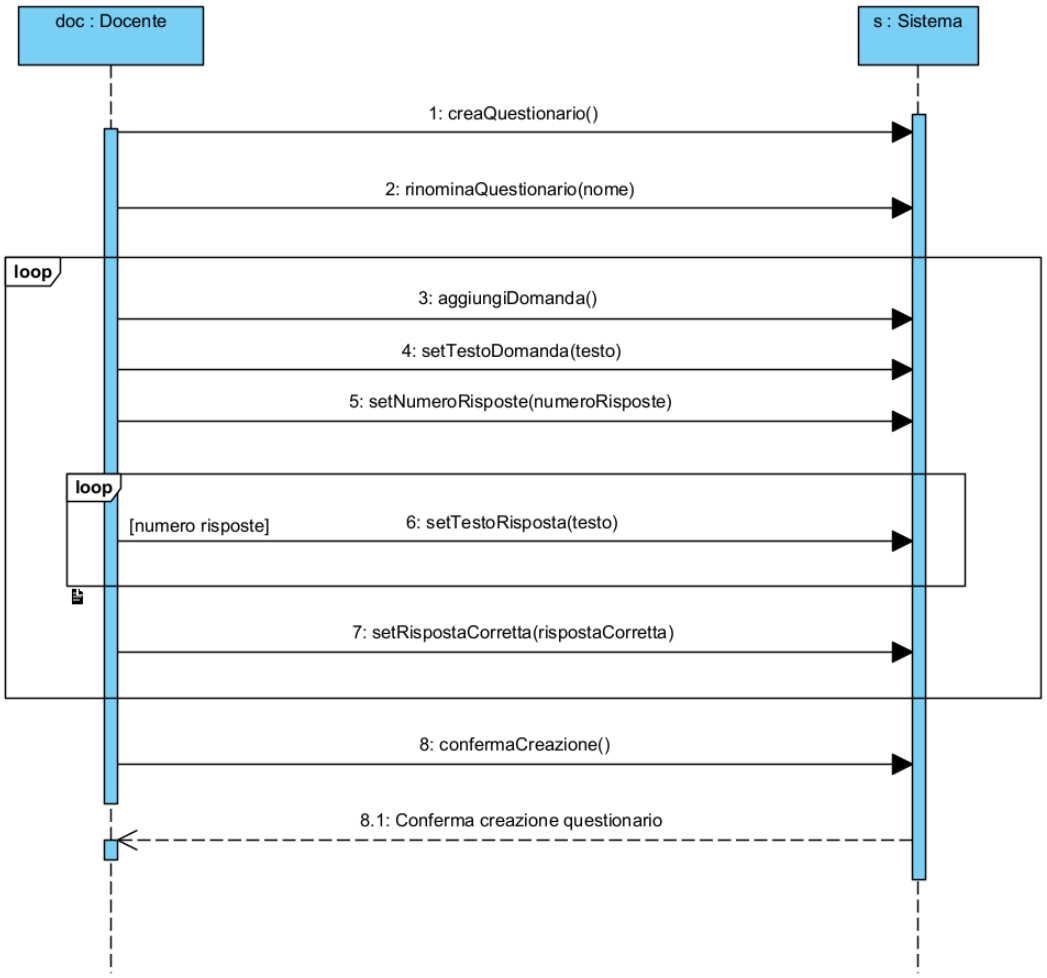

Figura 2.4: SSD - CreaQuestionario

2.4.3.2 Contratti delle operazioni

Rinomina Questionario L'operazione di sistema *rinominaQuestionario* consente al docente di modificare il titolo di un questionario esistente o in fase di creazione, aggiornando lo stato dell'oggetto nel sistema.

Contratto: rinominaQuestionario	
Operazione:	rinominaQuestionario(nome: String)
Riferimenti caso d'uso:	CreaQuestionario, ModificaQuestionario

Precondizioni:	È in corso la creazione/modifica di un Questionario **q** da parte di un Docente **doc**.
Postcondizioni:	• **q**.nome è stato impostato a *nome* (modifica di attributo)

Tabella 2.10: Contratto dell'operazione rinominaQuestionario

Crea Questionario L'operazione di sistema *creaQuestionario* formalizza l'istanziazione di un nuovo oggetto Questionario nel sistema, garantendo l'assegnazione di un identificativo univoco e la corretta datazione del contenuto.

Contratto: creaQuestionario	
Operazione:	creaQuestionario()
Riferimenti caso d'uso:	CreaQuestionario
Precondizioni:	È in corso la creazione di un Questionario **q** da parte di un Docente **doc**.
Postcondizioni:	• È stata creata un'istanza **q** di Questionario (creazione di oggetto); • **q**.ID è stato inizializzato univoco (modifica di attributo); • **q**.dataCreazione è stato inizializzato alla data corrente (modifica di attributo); • Gli altri attributi di **q** sono stati inizializzati (modifica di attributo); • L'istanza **q** di Questionario è stata associata all'istanza **doc** di Docente (formazione di collegamento).

Tabella 2.11: Contratto dell'operazione creaQuestionario

2.4.4 UC4: ModificaQuestionario

2.4.4.1 Diagramma di sequenza di sistema

Il diagramma di sequenza di sistema per il caso d'uso UC4 è il seguente:

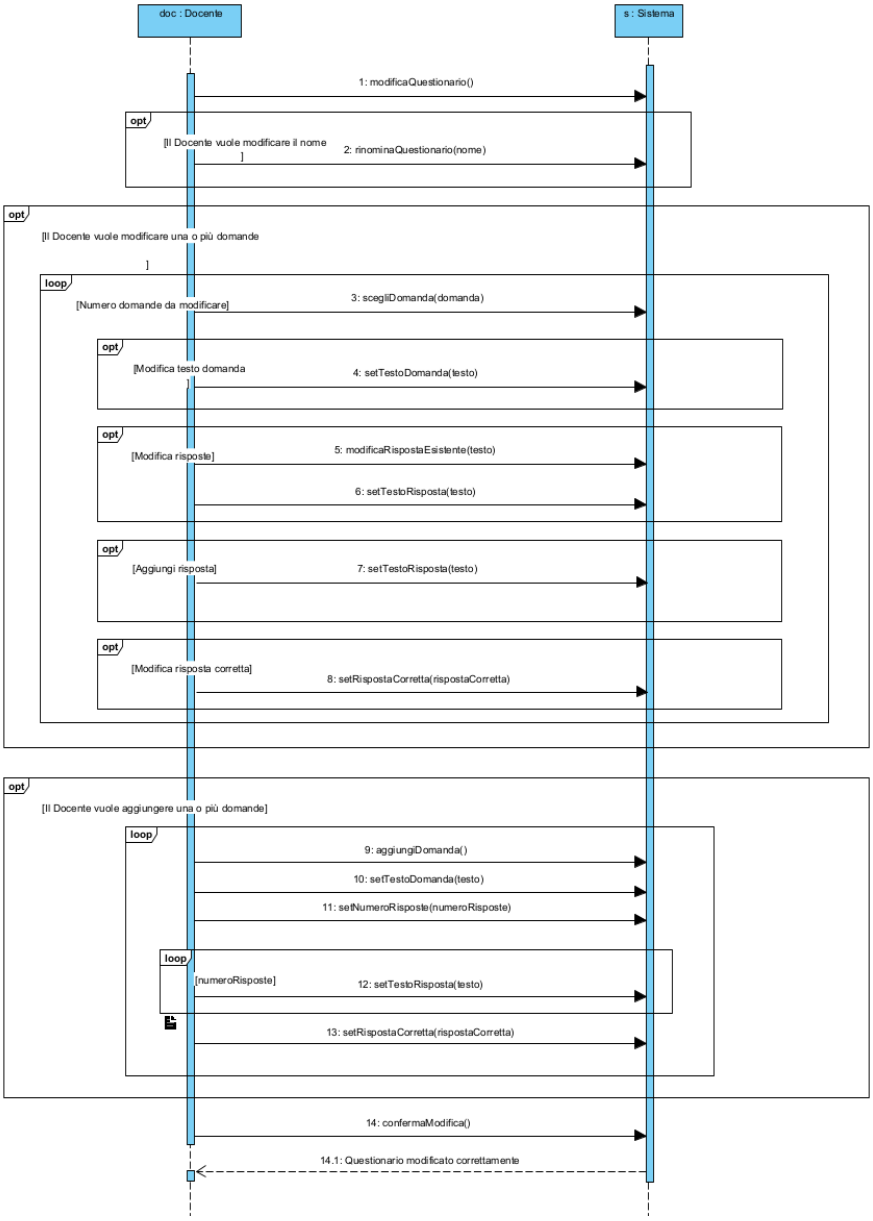

Figura 2.5: SSD - ModificaQuestionario

2.5 Progettazione ed implementazione

In questa fase, i requisiti analizzati nelle sezioni precedenti vengono tradotti in una struttura software concreta e funzionante. L'obiettivo primario del processo di progettazione è stato quello di garantire la **manutenibilità** e l'**estensibilità** del sistema EduQuest, adottando un approccio modulare basato sulla netta separazione delle responsabilità (Separation of concerns).
Per raggiungere tale scopo, l'architettura è stata strutturata seguendo il **pattern MVC** (Model-View-Controller), che permette di isolare la logica di business dalla gestione della persistenza e dall'interfaccia grafica. L'implementazione si avvale del framework **Spring Boot**, sfruttando l'ecosistema dei suoi *starter* per la gestione delle dipendenze e la configurazione del web server embedded.

2.5.1 Architettura del sistema

Il sistema è stato sviluppato utilizzando il framework **Spring Boot**, scelto per la capacità di semplificare la la gestione del ciclo di vita dei componenti tramite la *Dependency Injection*. L'architettura segue il pattern architetturale **MVC (Model-View-Controller)**, organizzato in una struttura a livelli (Layered Architecture) per garantire la separazione delle responsabilità e l'indipendenza tra la logica applicativa e la persistenza dei dati.
Il seguente **diagramma dei package UML** illustra l'architettura logica del sistema EduQuest, evidenziando la suddivisione del software in package e la gerarchia dei moduli software.

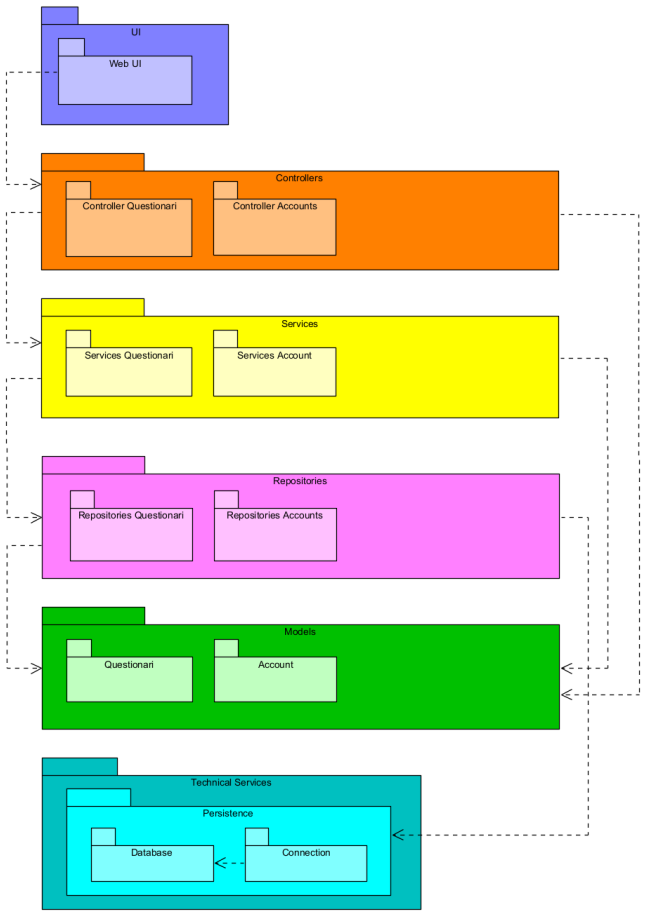

Figura 2.6: Diagramma dei package (Iterazione 1)

Nota: il package *UI* presente nel diagramma dei package sarà modellato nella seconda iterazione.

Il flusso dei dati attraversa i seguenti layer distinti:

- **Controller Layer (REST API)**: rappresenta l'interfaccia di comunicazione verso l'esterno (per questa prima iterazione). I controller gestiscono le richieste HTTP, espongono gli endpoint API, validano l'input e delegano l'esecuzione della logica di business al Service layer (strato sottostante). La comunciazione avviene tramite formato standard **JSON**, garantendo l'interoperabilità con potenziali client.
- **Service Layer (Logica di business)**: costituisce il nucleo operativo del sistema dove risiede l'intera **logica di business**. Questo strato è responsabile del coordinamento delle operazioni (come il calcolo dei punteggi) ed è mantenuto indipendente dalla persistenza dei dati.
- **Repository Layer**: questo strato si occupa esclusivamente dell'accesso ai dati. È stato implementato tramite il pattern **DAO** (Data Access Object). Utilizza query SQL esplicite per l'interazione con il database, garantendo un controllo granulare sulle prestazioni e sulla struttura delle interrogazioni.
- **Model Layer (POJO)**: definisce le entità del dominio di EduQuest. Si tratta di classi "puramente Java" (Plain Old Java Objects) che *modellano* i dati (Account, Studente, Questionario, ecc.), ovvero mappano direttamente le tabelle del database. Questi oggetti vengono utilizzati trasversalmente da tutti i layer come veicolo di informazione tra i vari strati.
- **DataBase MySQL**: il driver *my-sql-connectory-java* permette la connessione JDBC tra l'applicazione e il DBMS relazionale, incaricato di assicurare l'integrità referenziale e la persistenza a lungo termine.

L'adozione di questa struttura *Controller* → *Service* → *Repository* → *Database* permette di ottenere un sistema altamente disaccoppiato, facilitando la manutenzione del codice e permettendo l'evoluzione indipendente dei singoli moduli.

2.5.2 Tecnologie e Strumenti

Lo sviluppo della prima iterazione di EduQuest si è basato su uno stack tecnologico orientato alla robustezza e alla verificabilità del codice.

- **Linguaggio di Programmazione**: Java 21 (LTS), scelto per le prestazioni, le moderne feature del linguaggio e familiarità con tutti i componenti del gruppo.
- **Framework Backend**: Spring Boot 4.0.1, utilizzato come orchestratore per la Dependency Injection e per la gestione del ciclo di vita dell'applicazione.
- **Build Automation**: Maven, per la gestione centralizzata delle dipendenze e del ciclo di build.

Persistenza dei dati La gestione dei dati è affidata a MySQL 8.0. A differenza degli approcci standard basati su ORM (JPA), si è scelto di operare una persistenza di basso livello per garantire il massimo controllo sulle interrogazioni:

- **Driver JDBC**:*mysql-connector-java* (v. 8.0.29).
- **Strategia**: implementazione del pattern DAO per l'esecuzione di query SQL native, garantendo il disaccoppiamento tra il modello relazionale e le entità di business.

Qualità del Codice e Testing Un punto cardine dell'implementazione è stato il monitoraggio della qualità:

- **SonarQube**: utilizzato tramite il *sonar-maven-plugin* per l'analisi statica del codice, l'individuazione di code smells e il monitoraggio del debito tecnico.
- **JaCoCo (Java Code Coverage)**: integrato per misurare la copertura dei test. Il plugin è configurato per generare report dettagliati durante la fase di test, assicurando che la logica di business sia adeguatamente verificata.
- **Testing**: Spring Boot Starter Test, utilizzato per la validazione dei componenti e degli endpoint.

2.5.3 Diagrammi di Sequenza

I Diagrammi di Sequenza illustrano la vista dinamica del sistema, descrivendo l'interazione temporale tra gli oggetti software necessaria per realizzare le operazioni definite nei Contratti di Sistema. Questi schemi dettagliano il flusso dei messaggi che attraversa i diversi layer architetturali: dalla ricezione della richiesta HTTP da parte del **Controller** all'esecuzione della logica di business all'interno del **Service**. Ogni diagramma garantisce che le post-condizioni stabilite in fase di analisi siano soddisfatte attraverso una corretta collaborazione tra le classi di progetto.
Di seguito vengono analizzati i diagrammi relativi alle operazioni dell'Iterazione 1: creaAccountStudente e creaQuestionario.

SD - creaAccountStudente Il diagramma illustra il processo di registrazione di un nuovo profilo. La richiesta, ricevuta dallo **StudenteController**, tramite un payload di dati, viene inoltrata all'**AccountService** per la gestione della logica di business. In questa fase, si evidenzia l'impiego del pattern **Factory** (**AccountFactory**) per l'istanziazione dell'oggetto **Studente**. Questa scelta progettuale garantisce un disaccoppiamento tra la logica di controllo e la creazione delle entità, rispettando i principi di alta coesione e basso accoppiamento.

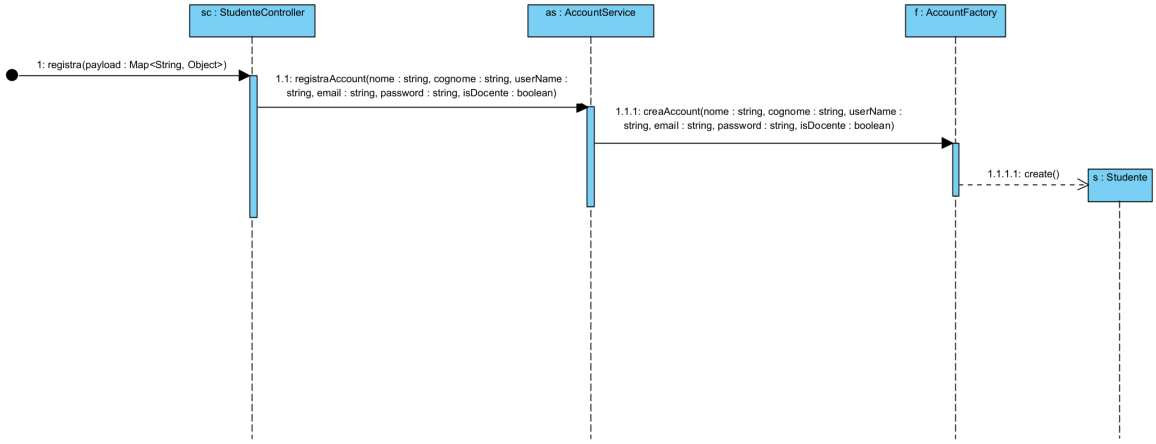

Figura 2.7: Diagramma di sequenza dell'operazione creaAccountStudente

SD - creaQuestionario Questo diagramma descrive la dinamica software pe la creazione di un nuovo Questionario. Il flusso ha inizio nel **QuestionarioController**, che riceve i parametri identificativi del docente, e prosegue nel **QuestionarioService**, che coordina la creazione dell'istanza di **Questionario**. Il passaggio cruciale è l'operazione di *associaDocente* che soddisfa la post-condizione di formazione del collegamento definita nel relativo contratto.

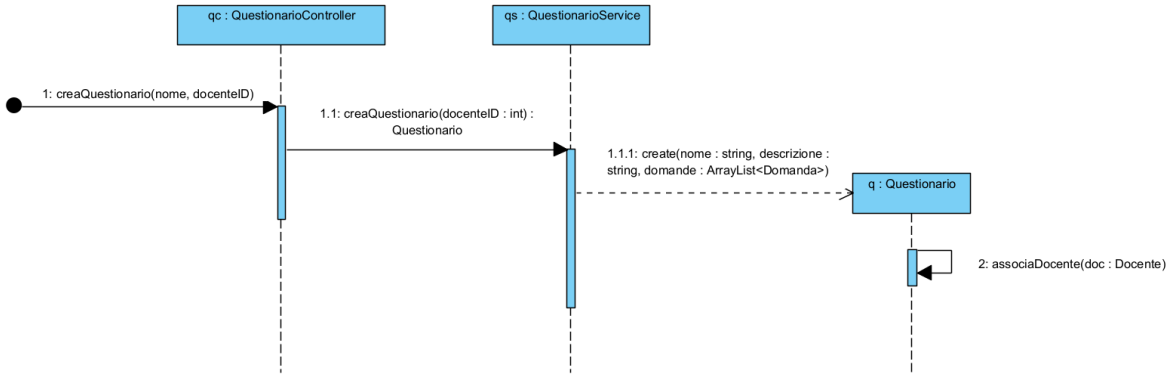

Figura 2.8: Diagramma di sequenza dell'operazione creaQuestionario

2.5.4 Diagramma delle Classi di Progetto

In questa sezione viene presentato il **Diagramma delle Classi di Progetto**, che costituisce la vista statica definitiva del software. Rispetto al modello di dominio, questo schema include i dettagli implementativi emersi durante la progettazione dinamica (SD), specificando le firme dei metodi, i tipi di ritorno e le relazioni di dipendenza tra le classi. Il diagramma evidenzia chiaramente la separazione tra le *Entities* (oggetti di business), i *Repositories* (interfacce di persistenza) e i *Services* (orchestratori della logica applicativa). Per garantire la massima leggibilità del diagramma, esso è stato diviso in 5 parti.

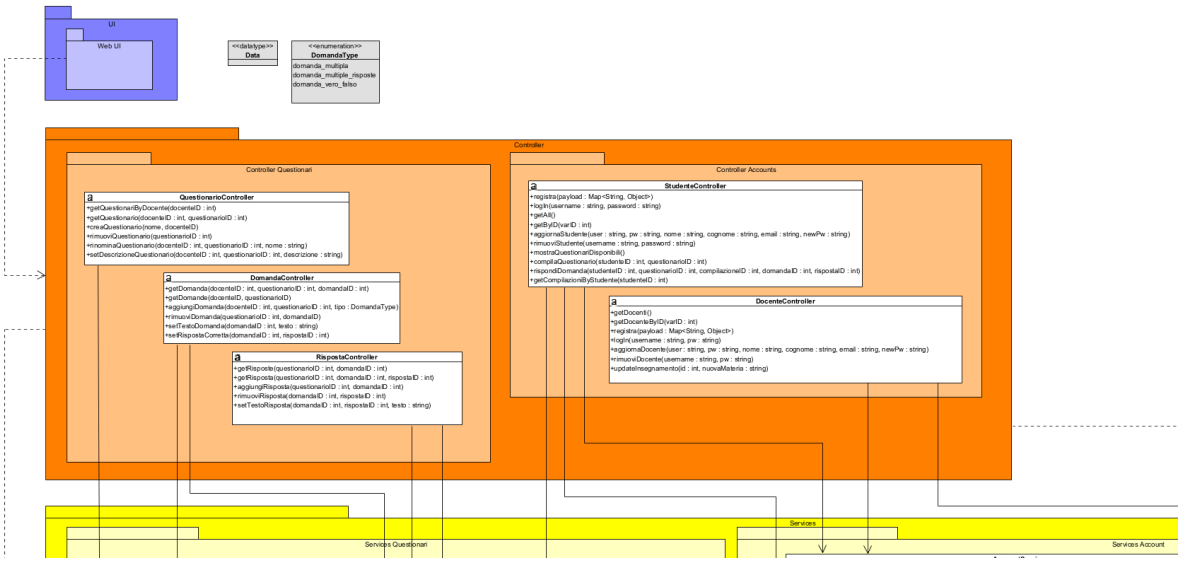

Figura 2.9: Diagramma delle Classi di progetto, pt.1

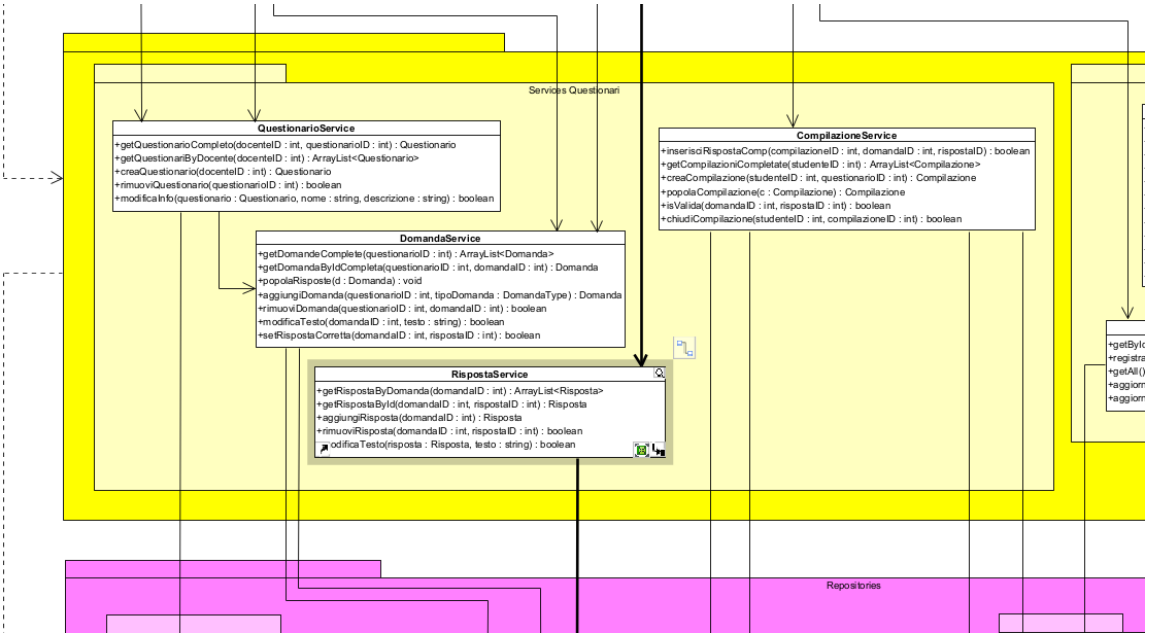

Figura 2.10: Diagramma delle Classi di progetto, pt.2

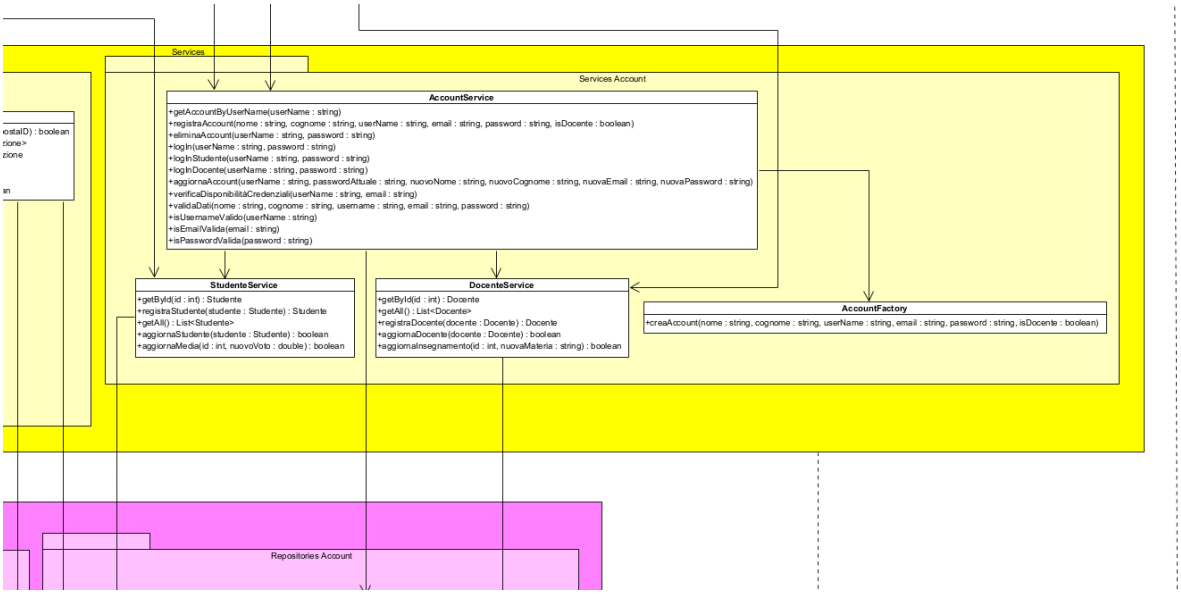

Figura 2.11: Diagramma delle Classi di progetto, pt.3

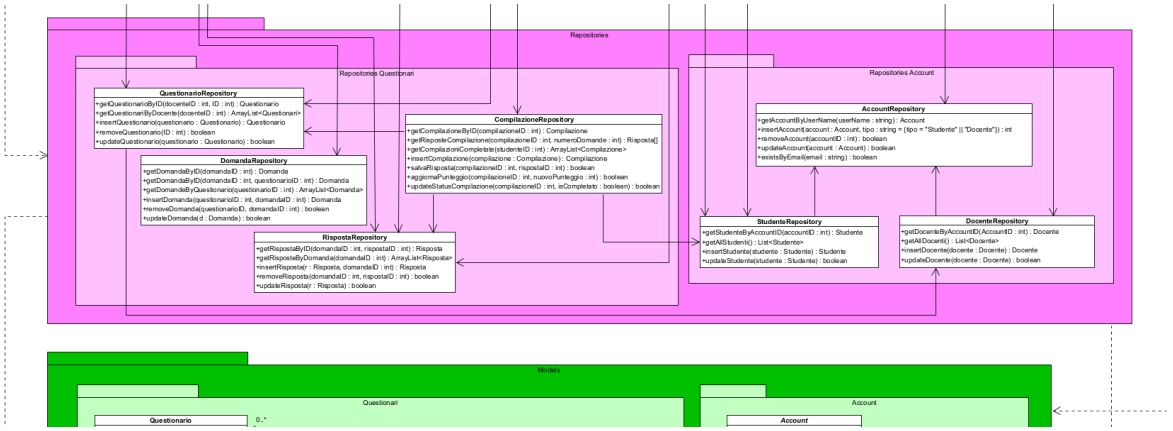

Figura 2.12: Diagramma delle Classi di progetto, pt.4

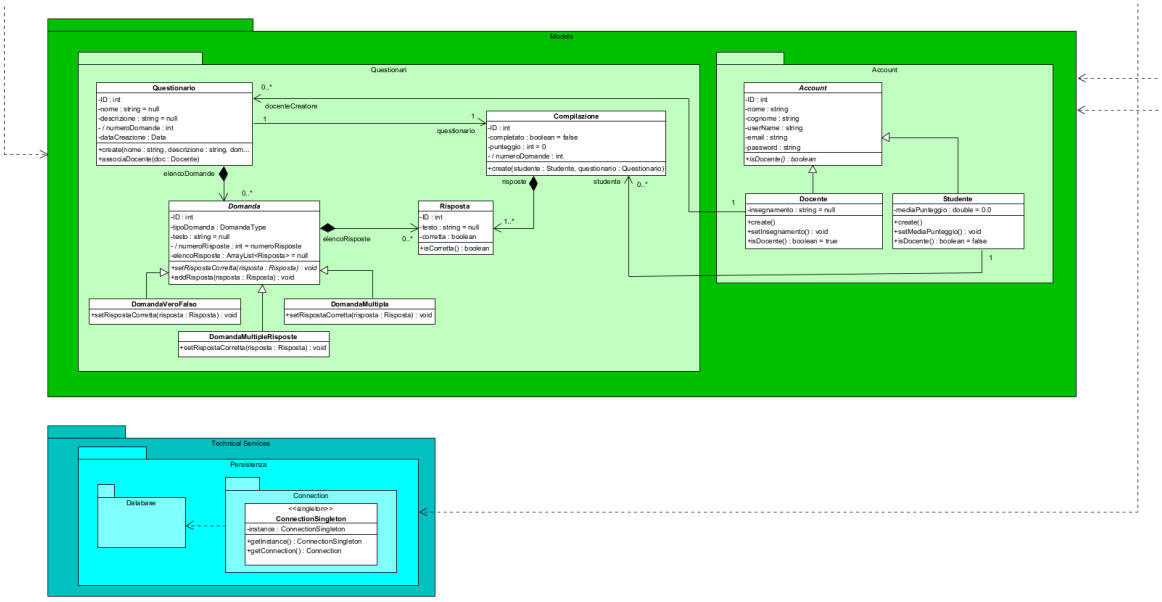

Figura 2.13: Diagramma delle Classi di progetto, pt.5

Nota sulla rappresentazione delle classi software: nel diagramma sono stati omessi i metodi `getter` e `setter` per tutte le classi software. Tale scelta è motivata dalla volontà di non appesantire inutilmente la visualizzazione grafica con operazioni puramente accessorie e prive di logica di business specifica. Si assume convenzionalmente che ogni attributo privato sia corredato dai relativi metodi di get e set, i quali verranno implementati direttamente in fase di implementazione.

2.5.5 Note sull'implementazione

In questa sezione vengono approfondite alcune scelte tecniche e implementative adottate per garantire l'efficienza e la manutenibilità del sistema EduQuest:

- **Pattern Factory**: per la gestione dei diversi profili utente è stata delegata alla classe i dettagli implementativi emersi durante la progettazione dinamica. L'uso di questo pattern permette di centralizzare la logica di istanziazione, disaccoppiando la creazione degli oggetti `Studente` e `Docente` dalla logica di controllo dei Controller.
- **Gestione della Persistenza (pattern Singleton)**: l'accesso al database MySQL è stato gestto attraverso il pattern **Singleton** implementato nella classe `ConnectionSingleton`. Tale scelta assicura l'esistenza di un'unica istanza della connessione, ottimizzando l'impiego delle risorse di sistema e prevenendo potenziali conflitti di accesso al database.
- **Flessibilità dei Payload**: nei Controller è stata adottata la struttura `Map<String, Object>` per la ricezione dei dati in formato JSON. Tale approccio conferisce flessibilità nello scambio di dati tra front-end e back-end, consentendo una validazione granulare prima della mappatura nelle entità di dominio.
- **Disaccoppiamento tramite Service Layer**: l'intera la logica di business è isolata nel **Service Layer**. Questo garantisce che i Controller mantengano esclusivamente responsabilità di "routing" e validazione dell'input, facilitando il riuso del codice e l'implementazione di test unitari per la logica di business.
- **Strutture Dati**: la gestione delle associazioni uno-a-molti (come tra `Questionario` e `Domanda`) è stata implementata tramite la classe `ArrayList`. Questa scelta garantisce una manipolazione efficiente e ordinata delle sequenze di oggetti durante il ciclo di vita dell'applicazione.
- **Identità e Persistenza**: Per agevolare il mapping tra le classi Java e le tabelle del database relazionale, è stato introdotto in ogni entità un attributo `int id`. Tale identificativo viene gestito tramite la clausola **AUTO_INCREMENT** lato MySQL, garantendo l'unicità della chiave primaria senza gravare sulla logica di business. Questo permette al sistema di recuperare in modo univoco le istanze persistite e di gestire correttamente le relazioni (foreign keys) tra gli oggetti durante le operazioni di interrogazione.

L'adozione di questi pattern e scelte progettuali riflette un approccio orientato agli oggetti che privilegia il basso accoppiamento e l'alta coesione tra i componenti del sistema.

2.6 Verifica e testing

La fase di testing è stata condotta parallelamente allo sviluppo per garantire la correttezza delle funzionalità implementate e la robustezza del sistema. Per questa prima iterazione sono stati scritti **test unitari** mirati per le classi dei Models e Services:

- Per i **Models**, i test hanno verificato la corretta gestione dello stato degli oggetti e l'integrità dei dati nei costruttori.
- Per i **Services**, è stato utilzizato il framework **Mockito** per isolare la logica di business dalle dipendenze esterne (come i Repository). L'uso dei Mock ha permesso di simulare il comportamento del database, verificando che le operazioni rispondessero correttamente agli input in modo deterministico.

L'attività di testing è stata monitorata tramite **JaCoCo**, i cui report sulla code coverage sono stati integrati in **SonarQube** per un'analisi statica approfondita volta a minimizzare il debito tecnico.

2.6.1 Validazione tramite Postman

La validazione degli endpoint REST è stata effettuata tramite **Postman**. Questi test sono stati configurati come **test di integrazione 'Black-Box'**. Il sistema viene approcciato dall'esterno tramite le sue interfacce REST, verificando che la collaborazione tra i diversi layer applicativi e il database produca lo stato finale desiderato, senza intervenire direttamente sulla logica interna del codice.
Attraverso l'invio di request JSON, è stato verificato il corretto flusso dei dati dal Controller fino alla persistenza finale. Nello specifico, sono stati testati i principali casi d'uso (come la registrazione di uno studente e la creazione di un questionario), accertando che i codici di risposta HTTP fossero coerenti con le operazioni eseguite e che i dati inviati venissero correttamente processati dal sistema.
Di seguito vengono illustrati due esempi di verifica degli endpoint (esposti su `localhost:8080`) relativi a due delle operazioni di sistema modellate:

- **Verifica Endpoint *creaAccount***: è stata testata la rotta `POST /api/account/studente` inviando un payload JSON contenente i dati di registrazione. Il sistema ha risposto correttamente lo stato **201 Created**, confermando l'avvenuta creazione dell'account e l'inizializzazione della media punteggio tramite il layer *Service*.
- **Verifica Endpoint *creaQuestionario***: è stata testata l'operazione di creazione questionario inviando i parametri necessari (nome e ID del docente). Il test ha confermato che il sistema associa correttamente il nuovo oggetto al **Docente** di riferimento, come previsto dal diagramma di sequenza.

Capitolo 3

Iterazione 2 - Compitini, esercitazioni e gamification

3.1 Obiettivi dell'iterazione 2

In questa seconda fase del ciclo di sviluppo, il progetto EduQuest evolve da un insieme di servizi di back-end a una piattaforma web completa. L'attenzione si sposta sull'esperienza utente, sulla diversificazione delle modalità di valutazione e sull'introduzione di meccanismi di gamification.
Gli obiettivi prefissati per questa iterazione mirano a consolidare l'architettura esistente e a estenderne le funzionalità core:

- **Sviluppo dell'interfaccia utente (front-end)**: implemntazione di una UI dinamica e responsvia il motore di templating **Thymeleaf**, integrato con **HTML5** e **CSS3**, per permettere l'interazione diretta degli utenti con il sistema.
- **Materia e livello di difficoltà dei Questionari**: evoluzione dell'entità Questionario tramite l'introduzione della `materia` di riferimento e del `livello di difficoltà`. Questo permette una gestione dei contenuti più granulare e una migliore organizzazione del materiale didattico.
- **Diversificazione delle tipologie di Questionari**: introduzione delle modalità **Compitino** ed **Esercitazione**, caratterizzate da diverse logiche di valutazione e vincoli temporali, per coprire sia scenari di verifica formale che di studio autonomo.
- **Implementazione delle dashboard**: creazione di pannelli di controllo personalizzati per Studenti e Docenti, con conseguente implementazione delle funzioni necessarie a popolarli. La Dashboard deve fungere da hub centrale per la visualizzazione delle statistiche, lo storico dei questionari e la gestione dei contenuti.
- **Introduzione di dinamiche di gamification**: integrazione del sistema degli EduPoints, una metrica di punteggio bonus volta a incentivare l'impegno dello studente attraverso il gioco, migliorando il coinvolgimento e la continuità d'uso.
- **Raffinamento della persistenza**: estensione del database MySQL e dei relativi DAO per supportare le nuove entità.

3.2 Modello di dominio

Il modello di dominio viene ampliato per riflettere le nuove logiche di business introdotte. Le modifiche principali riguardano la categorizzazione dei contenuti e l'introduzione di attributi legati alla gamification e alle diverse tipologie di questionari.

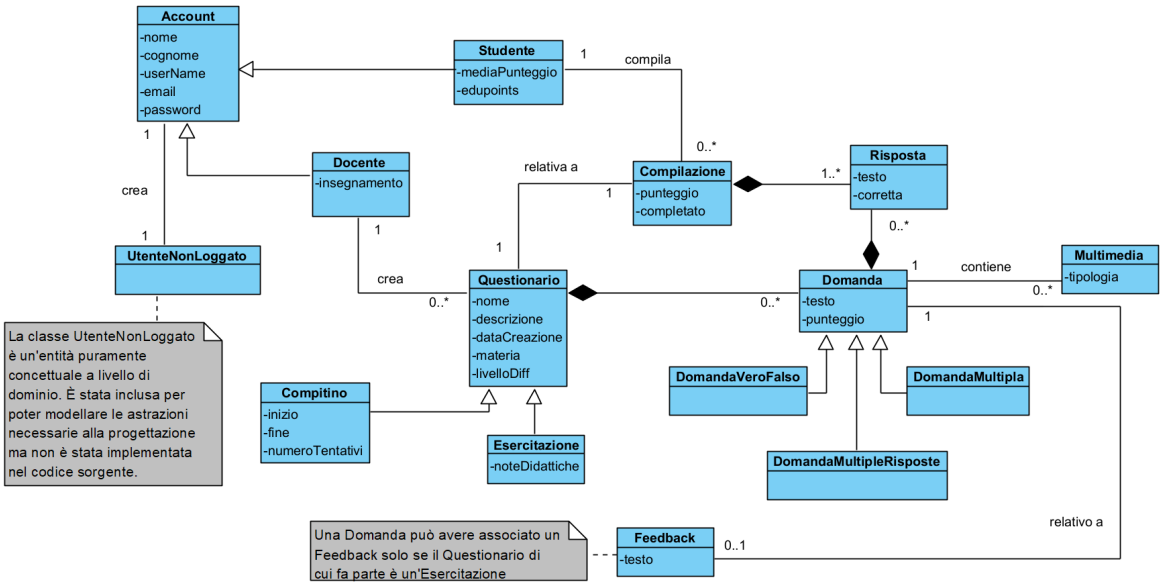

Figura 3.1: Modello di Dominio (Iterazione 2)

Principali integrazioni

- **Questionario**: la classe è stata arricchita con gli attributi `materia` (es. Matematica, Storia) e `livelloDiff`, che rappresenta il livello di difficoltà (Facile, Medio, Difficile). Queste proprietà permettono una classificazione del materiale didattico.
- **Studente**: la classe è stata arricchita con l'attributo `eduPoints`. Come descritto nel Glossario 1.1, questo valore rappresenta il punteggio cumulativo ottenuto attraverso le dinamiche di gioco, distinguendosi dal punteggio tecnico del singolo questionario.
- **Domanda**: la classe è stata arricchita con l'attributo `punteggio`. I punteggi delle domande decisi dal Docente e vengono sommati al punteggio di una Compilazione nel caso di risposta corretta.
- **Specializzazione dei Questionari**: viene modellata la distinzione tra **Compitino** ed **Esercitazione**. Sebbene condividano la struttura base del Questionario, esse differiscono per i vincoli di tempo o le modalità di feedback.

Classi eliminate nella seconda iterazione

- **Revisione**: rimossa per evitare ridondanza informativa rispetto all'entità Compilazione.
- **Multimedia**: eliminata dal perimetro del progetto a causa dei vincoli temporali, dando priorità alle funzionalità core.

3.3 Realizzazione dei Casi d'Uso e artefatti correlati

Questa seconda iterazione non ha comportato l'identificazione di nuovi Casi d'Uso, poiché il perimetro funzionale del sistema era già stato delineato nella fase iniziale. In linea con la metodologia UP, non si è ritenuto necessario produrre un nuovo diagramma dei Casi d'Uso, nuovi SSD o Contratti di Sistema. Lo sforzo analitico è stato invece concentrato verso la modellazione del comportamento dinamico delle nuove funzionalità e sul raffinamento di quelle già implementate.

3.4 Progettazione ed implementazione

In questa seconda fase, il lavoro si è spostato dalla costruzione delle fondamenta del sistema alla realizzazione dell'esperienza utente vera e propria. Se la prima iterazione era servita a definire "cosa" il sistema dovesse gestire a livello di dati, questa fase si è concentrata su "come" l'utente interagisce con la piattaforma.

Architettura del sistema L'architettura logica definita nella prima iterazione si è dimostrata solida e versatile, rimanendo invariata anche a fronte dell'introduzione di nuove funzionalità.

3.4.1 Integrazione del front-end con Thymeleaf

Per dare vita all'interfaccia di EduQuest, lo stack tecnologico è stato arricchito con l'introduzione di **Thymeleaf**, integrato tramite l'apposito *starter* di SpringBoot. Questo motore di templating ha permesso di trasformare i servizi di back-end in pagine web interattive in modo semplice e integrato, permettendo una netta separazione tra logica di business e presentazione:

- **Creazione delle dashboard**: sono stati realizzati i pannelli di controllo per Studenti e Docenti, traducendo i dati grezzi provenienti dal database in elenchi e riepiloghi intuitivi.
- **Interattività e dinamismo**: grazie a Thymeleaf, le pagine HTML non sono più statiche, ma si adattano all'utente autenticato, mostrando, ad esempio, lo storico delle compilazione dello specifico Studente e i suoi EduPoints personali.
- **Utilizzo dei fragment**: è stata sfruttata la funzionalità dei `fragment` per standardizzare componenti comuni (come la barra di navigazione). Ciò ha permesso di ridurre la ridondanza del codice HTML e di semplificare la manutenzione dell'interfaccia.
- **Passaggio dati tramite l'interfaccia Model**: l'utlizzo dell'interfaccia `org.springframework.ui.Model` ha permesso di veicolare i dati dai Controller alla vista in modo efficiente. In questo modo, si sono stati *simulati* dei DTO (Data Transfer Object) dinamici per facilitare il trasferimento delle informazioni tra lato client e lato server.
- **HTTP Sessions**: per garantire la persistenza dell'autenticazione durante la navigazione, sono state implementate le **HTTP Sessions**. Questo meccanismo permette una gestione del login sicura e leggera, mantenendo i dati del profilo utente accessibili tra le diverse richieste del client.

3.5 Macchina a stati e diagramma di attività

Per comprendere meglio il comportamento dinamico del sistema, riportiamo di seguito un diagramma UML della **Macchina a Stati**, focalizzati sul ciclo di vita della Compilazione di un Questionario, e un **diagramma di Attvità**, che illustra il flusso logico delle operazioni per effettuare la ricerca di un Questionario.

3.5.1 Diagramma della Macchina a Stati: Compilazione

Il diagramma seguente illustra i vari stati in cui può trovarsi l'oggetto "Compilazione" durante il suo ciclo di vita, evidenziando le transizioni scatenate dalle azioni dello studente e le attività interne associate a ogni fase.

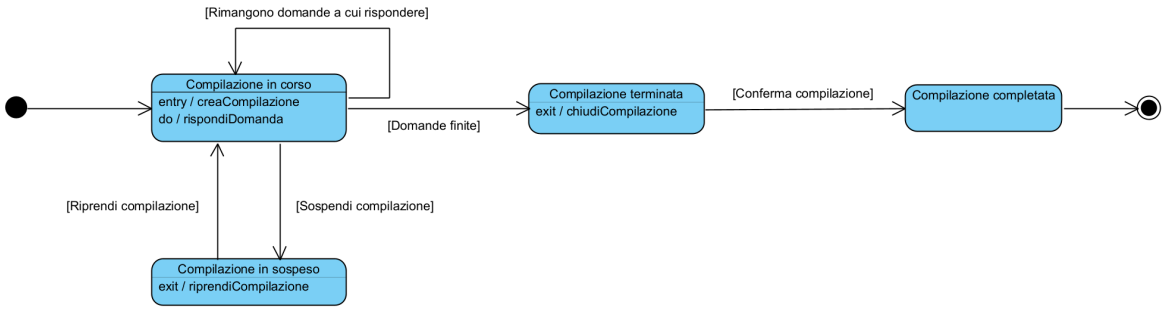

Figura 3.2: Diagramma Macchina a Stati creaCompilazione

Descrizione degli stati e delle transizioni

- **Compilazione in corso**: lo stato iniziale attivato dall'azione `creaCompilazione`. All'interno di questo stato, lo studente esegue l'attività `rispondiDomanda`. Il sistema rimane in questo stato finché la condizione di guardia `[Rimangono domanda a cui rispodnere è vera]`.
- **Compilazione in sospeso**: attraverso l'evento `[Sospendi compilazione]`, lo Studente può mettere in pausa la compilazione. L'uscita da questo stato stato (`exit`) invoca il metodo `riprendiCompilazione` per riportare lo Studente al punto in cui la Compilazione era stat interrotta.
- `Compilazione terminata`: quando la condizione `[Domande finite]` si avvera, il sistema transita verso al chiusura formale della sessione tramite l'azione `chiudiCompilazione`, che garantisce l'assegnazione degli eventuali EduPoints e la persistenza sul database.
- **Compilazione completata**: l'ultimo stato logico raggiunto dopo la `[Conferma compilazione]`, che precede la terminazione del ciclo di vita dell'oggetto.

3.5.2 Diagramma di attività: ricerca e filtraggio Questionari

Mentre la macchina a stati descrive il ciclo di vita di un oggetto, il diagramma di attività si concentra sul flusso logico delle operazioni. Qui abbiamo modellato il processo di ricerca, che permette allo studente di individuare il materiale didattico più adatto alle sue esigenze.

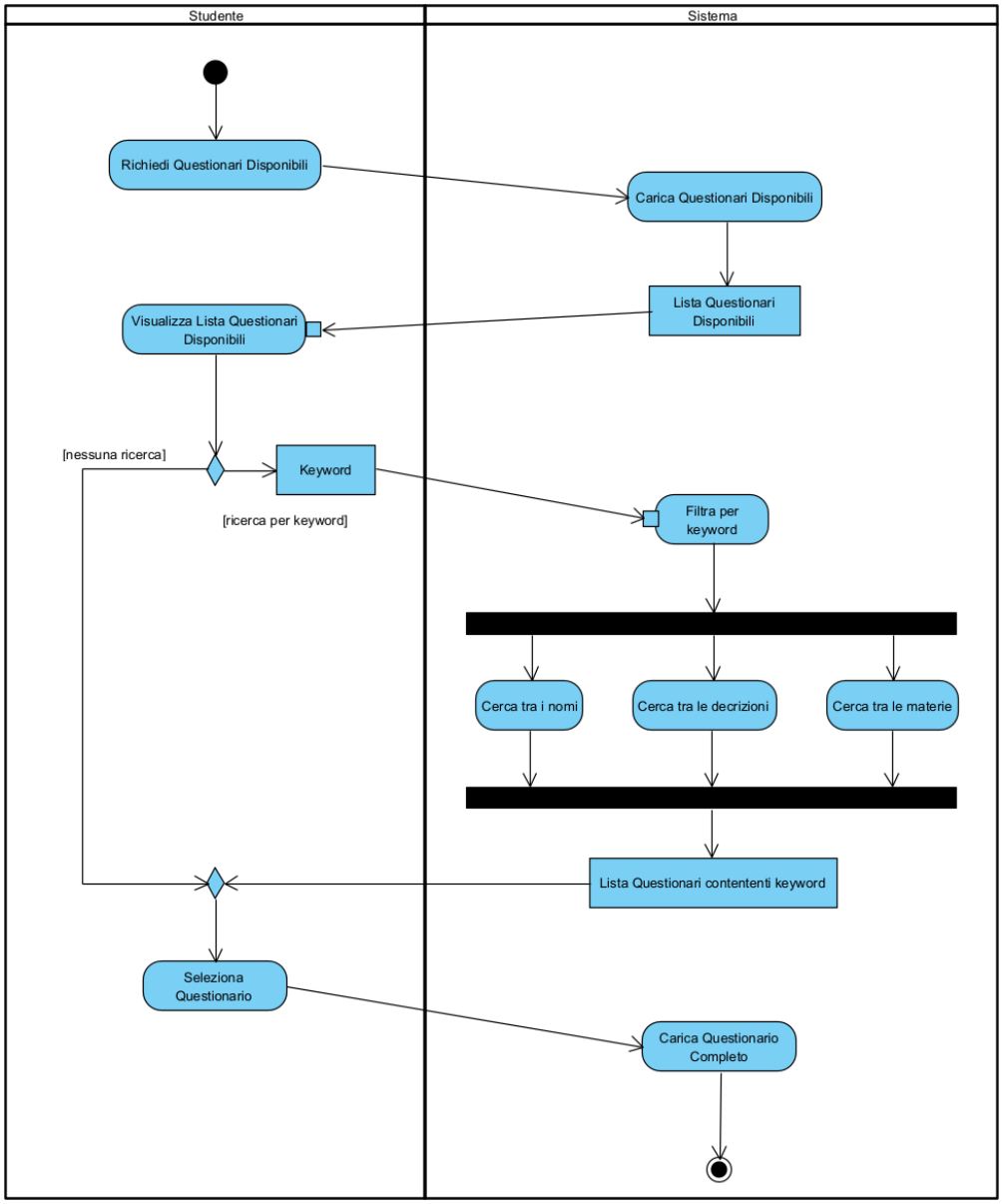

Figura 3.3: Diagramma di Attività ricercaQuestionario

3.6 Diagramma delle Classi di Progetto

Il Diagramma delle Classi di Progetto è stato aggiornato per riflettere l'evoluzione funzionale del sistema. In questa fase, l'attenzione si è focalizzata sull'integrazione delle nuove logiche di categorizzazione e gamification, ottimizzando al contempo la struttura esistente per rispondere alle esigenze emerse durante lo sviluppo del front-end.

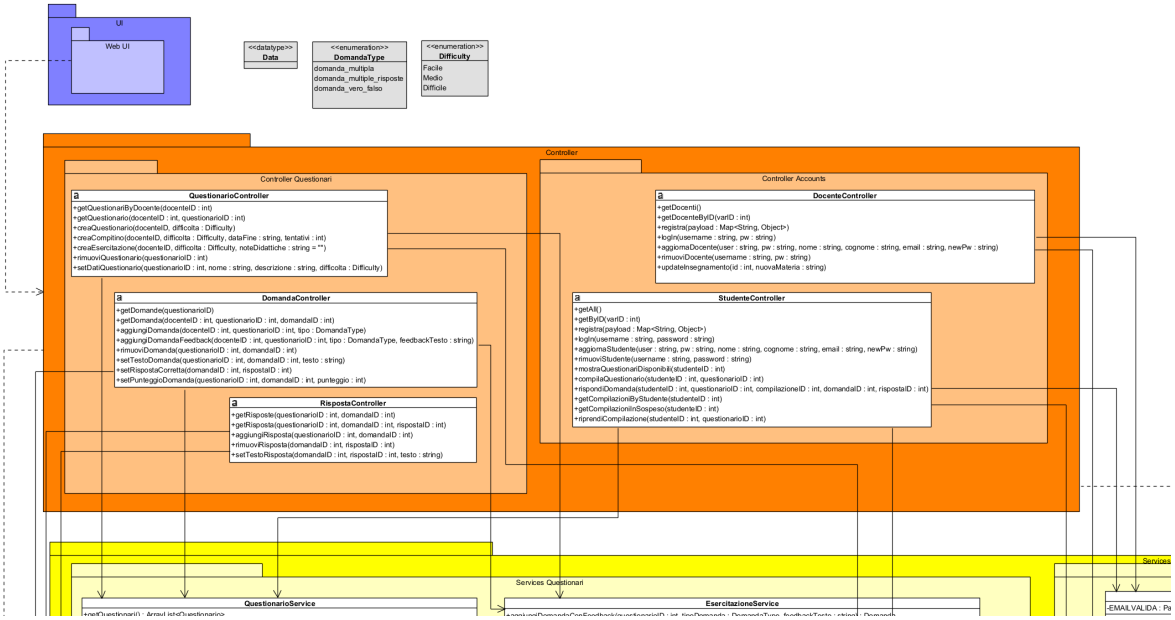

Figura 3.4: Diagramma delle Classi di progetto IT2, pt.1

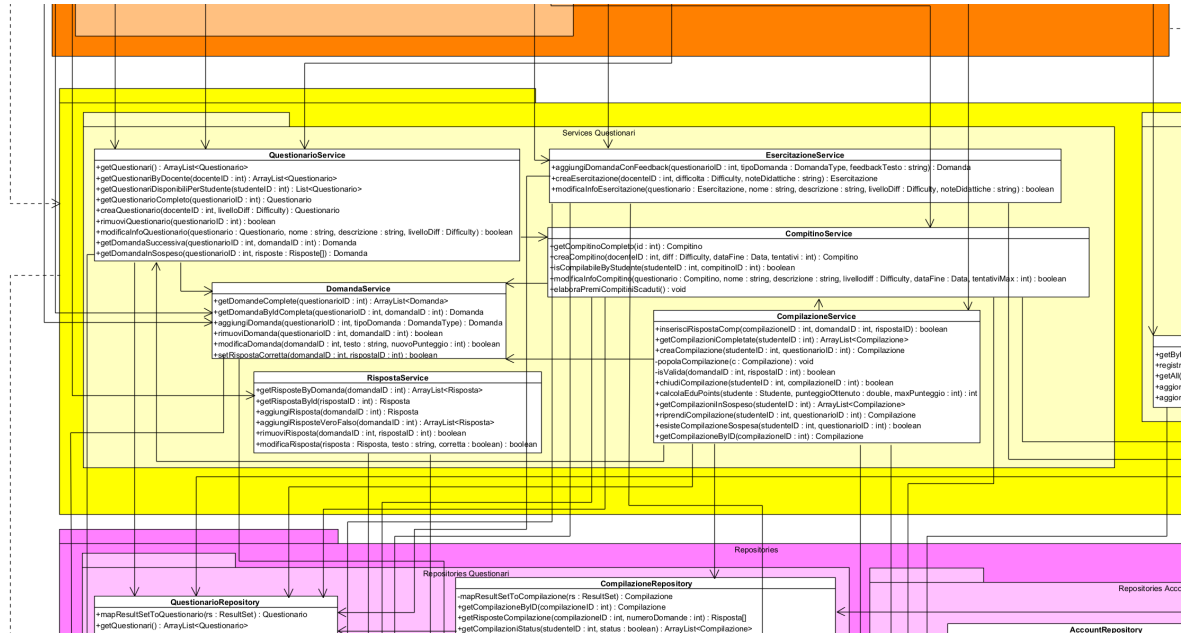

Figura 3.5: Diagramma delle Classi di progetto IT2, pt.2

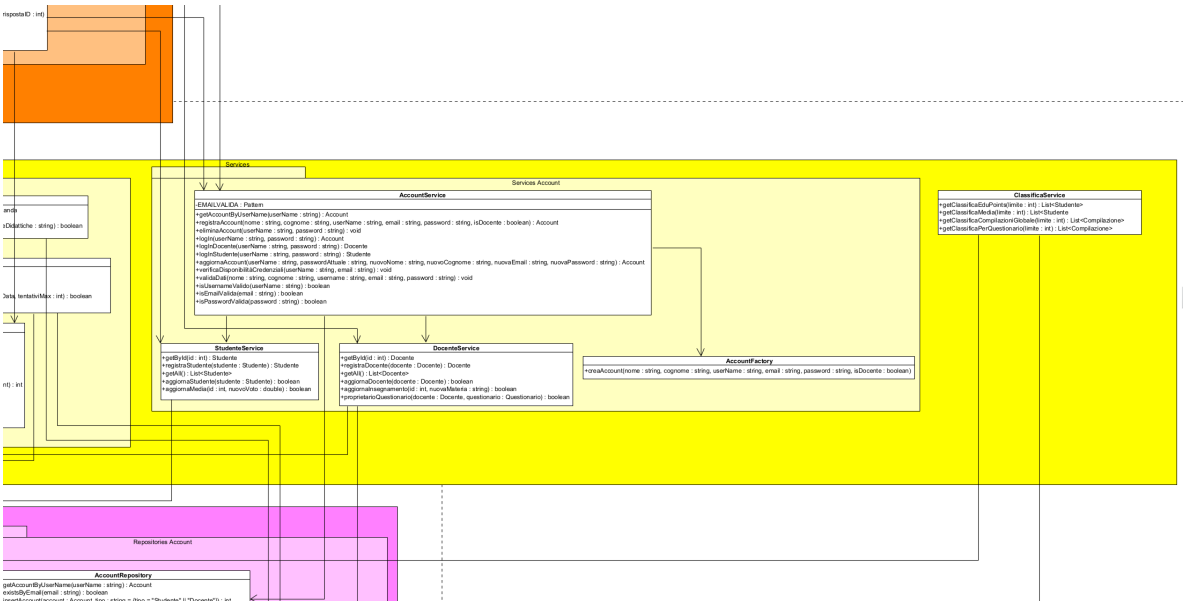

Figura 3.6: Diagramma delle Classi di progetto IT2, pt.3

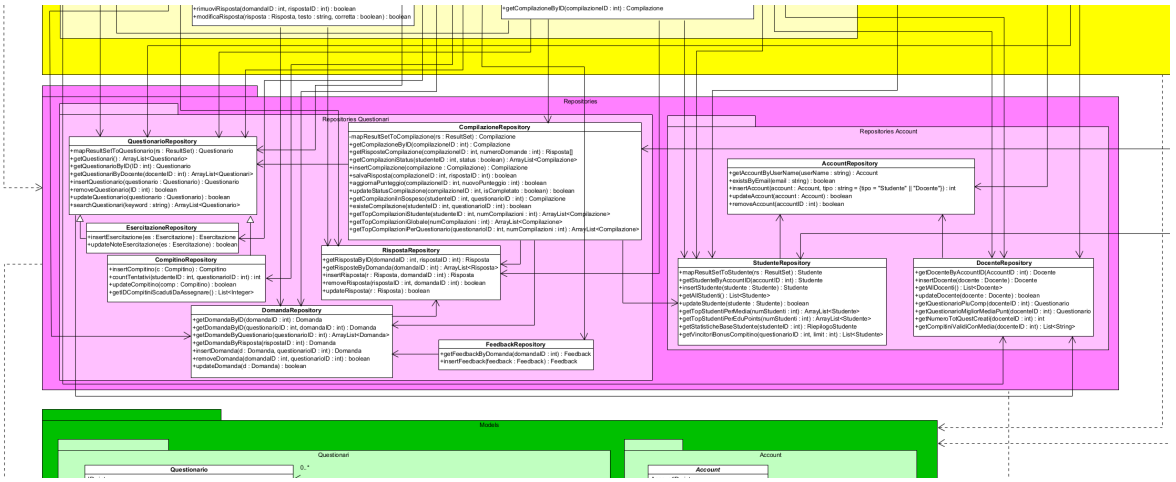

Figura 3.7: Diagramma delle Classi di progetto IT2, pt.4

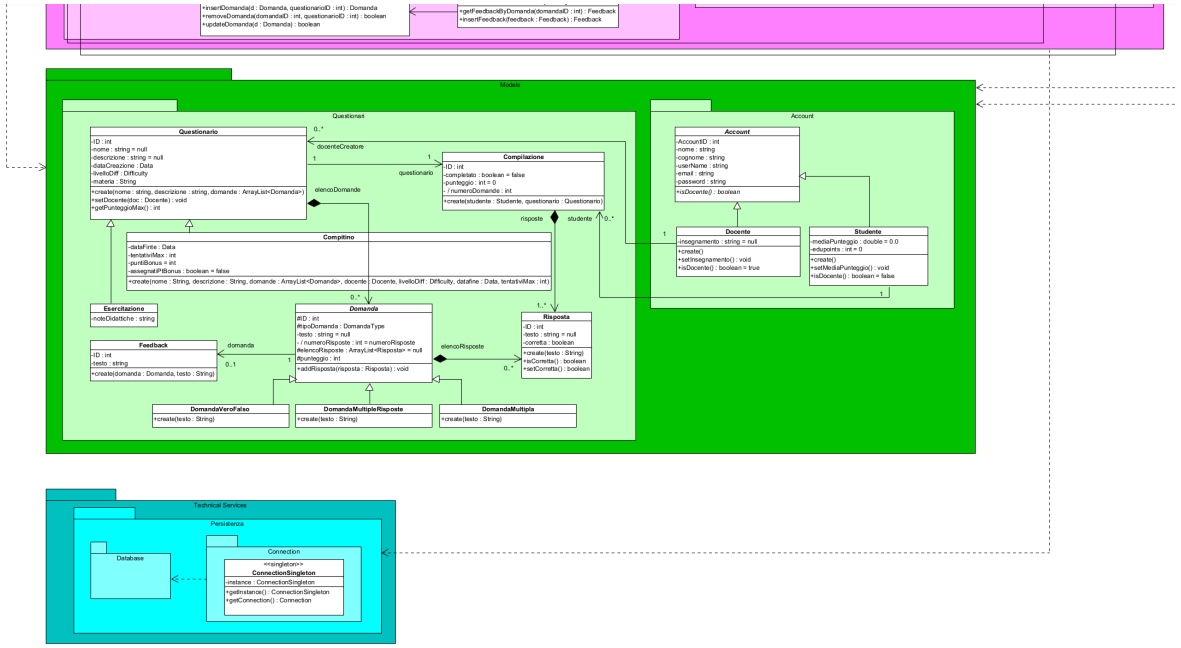

Figura 3.8: Diagramma delle Classi di progetto IT2, pt.5

Note sull'evoluzione del diagramma:

- **Integrazione nuove funzionalità**: come indicato anche nel 2.1 sono stati aggiunti gli attributi `materia` e `livelloDiff` alla classe `Questionario`, l'attributo `punteggio` alla classe `Domanda` e l'attributo `eduPoints` alla classe `Studente`. I relativi Service, Repositories e Models sono stati estesi con metodi specifici per la gestione di questi dati (ad esempio, il calcolo dinamico dei punti bonus al termine di una sessione).
- **Ottimizzazione dei metodi**: alcuni metodi preesistenti sono stati modificati per migliorare l'efficienza dello scambio di dati (ad esempio l'eliminazione degli `numeroDomande` e `numeroRisposte`). Altri metodi e classi sono stati modificati per permettere compatibilità con i nuovi Controller Web.
- **Esclusione dei Web Controller**: si è scelto deliberatamente di non includere nel diagramma le classi relative ai **Controller Web** (responsabili esclusivamente del routing delle viste Thymeleaf).Tale scelta è motivata dalla volontà di mantenere il diagramma focalizzato sulla logica di business e sulla persistenza, evitando di appesantire il modello con componenti puramente legati all'interfaccia utente.

Nota sull'applicazione dei principi SOLID Durante lo sviluppo della seconda iterazione, abbiamo cercato di seguire il più possibile i **principi SOLID** per evitare che il codice diventasse troppo rigido o difficile da modificare in futuro. In particolare, ci siamo concentrati su alcuni aspetti chiave:

- **Single Responsibility Principle (SRP)**: abbiamo lavorato per far sì che ogni classe avesse una sola responsabilità. Ad esempio, abbiamo separato nettamente le classi del `Modello` (POJO) dai `Service` (che gestiscono la logica di business) e dai `Repository` (che si occupano della comunicazione con il database).
- **Open/Closed Principle (OCP)**: l'architettura è stata pensata per essere "aperta alle estensioni". Un esempio chiaro è la gerarchia dei `Service`: se in futuro volessimo aggiungere un nuovo tipo di prova (oltre a `Compitini` ed `Esercitazioni`), potremmo farlo estendendo `QuestionarioService` senza dover rimettere mano al codice già scritto e testato.
- **Liskov Substitution Principle (LSP)**: grazie all'uso corretto dell'ereditarietà, il nostro sistema può trattare un `Compitino` o un'`Esercitazione` semplicemente come un `Questionario` ovunque non sia richiesta la logica specifica, garantendo che le sottoclassi rispettino sempre il comportamento della classe padre.
- **Interface Segregation & Dependency Inversion**: sfruttando le potenzialità di Spring Boot, abbiamo utilizzato l'iniezione delle dipendenze (*Dependency Injection*). In questo modo, le classi di alto livello non dipendono direttamente dalle implementazioni di basso livello, rendendo il sistema molto più flessibile e facile da testare con i Mock.

3.7 Note sull'implementazione

In questa sezione vengono approfondite le logiche di business e le scelte implementative che hanno guidato lo sviluppo delle nuove funzionalità di EduQuest.

Vincoli di dominio e gestione materie Per garantire la coerenza dei dati didattici, la creazione dei questionari è stata strettamente vincolata al profilo del Docente:

- **Vincolo di insegnamento**: un Docente può creare questionari solo dopo aver impostato la propria materia di riferimento nel profilo (attributo `insegnamento`).

- **Propagazione degli aggiornamenti**: la `materia` di un Questionario non è modificabile singolarmente, poiché la eredita direttamente dall'`insegnamento` del Docente creatore. In caso di aggiornamento dell'`insegnamento` nel profilo del Docente, il sistema riflette automaticamente il cambiamento su tutti i questionari da lui prodotti, garantendo l'integrità del catalogo.

Logica delle Dashboard Le dashboard sono state progettate per fungere da centri di controllo personalizzati, separando la gestione dei dati propri da quella dei dati globali della piattaforma:

- **Dashboard Docente**: focalizzata sulla gestione del materiale didattico e sul monitoraggio delle proprie attività. Permette di visualizzare le proprie informazioni personali, i questionari creati (con i relativi comandi per l'editing) e i comandi per la creazione di nuovi questionario (`Questionari`, `Esercitazioni` o `Compitini`).
- **Dashboard Studente**: orientata al monitoraggio del percorso personale. Qui lo studente può visualizzare le proprie statistiche (numero totale di compilazioni effettuate, media voti globale e saldo EduPoints) e accedere alla funzionalità di ricerca per individuare nuovi `Questionari`, `Esercitazioni` o `Compitini` da compilare.

Dinamiche di Gamification: assegnazione EduPoints e classifiche Il sistema degli **EduPoints** è gestito attraverso due logiche distinte, una immediata e una differita:

1. **Assegnazione immediata**: al termine di ogni compilazione, il sistema normalizza il voto in centesimi e assegna eventuali EduPoints.
   - 15 punti per un punteggio pieno (100%)
   - 10 punti per risultati superiori al 75%
   - 5 punti per risultati superiori al 50%
2. **Assegnazione differita**: è stata implementata una funzione automatica che si attiva all'accesso della dashboard da parte di uno Studente. Il sistema verifica i compitini scaduti i cui bonus non sono ancora stati erogati (tramite un flag booleano); per ognuno di essi, identifica i 3 studenti con la miglior prestazione (considerando solo la Compilazione il voto più alto per ogni Studente) e assegna loro i punti extra previsti dal Compitino (decidi dal Docente).

Per favorire la trasparenza e la competizione, è stata introdotta una vista separata per le **Classifiche globali**, accessibile dalla barra degli strumenti sia da studenti che da docenti. Questa sezione aggrega i dati a livello globale mostrando:

- La classifica degli studenti per le medie punteggio;
- La classifica degli studenti per EduPoints;
- Il miglior punteggio di Compilazione ottenuto su tutti i questionari;
- Il miglior punteggio di Compilazione su un questionario specifico.

Gerarchia dei Service A livello di architettura del codice, è stata applicata l'ereditarietà per massimizzare il riuso del codice e il polimorfismo:

- Le classi `CompitinoService` ed `EsercitazioneService` sono modellate come figlie della classe `QuestionarioService`.

Questa struttura permette di ereditare i metodi comuni di gestione e persistenza, lasciando alle classi figlie l'implementazione delle logiche specifiche (come la gestione delle scadenze per i compitini o il feedback per le esercitazioni).

3.8 Soddisfacimento dei requisiti non funzionali

Durante lo sviluppo della seconda iterazione, ci siamo assicurati che il sistema non rispetti solo i requisiti funzionali, ma anche quelli qualitativi definiti all'inizio del progetto:

- **Usabilità e Performance**: grazie all'integrazione di Thymeleaf e all'uso di CSS moderno, abbiamo creato un'interfaccia pulita e leggera. I tempi di risposta sono minimi e il caricamento delle dashboard avviene quasi istantaneamente, rispettando ampiamente il limite dei 2 secondi.
- **Sicurezza e Privacy**: pur lavorando con un database locale e senza un server pubblico, abbiamo implementato i principi base della sicurezza e del **GDPR**. Abbiamo garantito che un utente non possa accedere ai dati personali degli altri e abbiamo previsto una funzione di rimozione completa dal sistema su richiesta, assicurando il "diritto all'oblio".
- **Affidabilità e salvataggio incrementale:**: per evitare che un problema di rete o una chiusura accidentale del browser facciano perdere i progressi allo studente, abbiamo implementato un **meccanismo di salvataggio incrementale**. Ogni risposta viene salvata nel database man mano che viene data, permettendo di riprendere la compilazione esattamente da dove era stata interrotta (come mostrato nella macchina a stati 3.2).
- **Scalabilità**: questo requisito è stato **verificato a livello teorico**. Non potendo testare il sistema con centinaia di persone reali contemporaneamente, abbiamo simulato questo scenario tramite uno stress test con **Apache JMeter**. I risultati (descritti nella seguente sezione di Verifica Test) confermano che l'architettura è in grado di reggere carichi di lavoro elevati.

3.9 Verifica e testing

In questa seconda fase, l'attività di verifica si è estesa per coprire l'intera pipeline dei dati, dal database alla logica di business. Oltre all'integrazione di ulteriori test unitari, è stata effettuata un'analisi delle prestazioni tramite uno stress test con **Apache JMeter** (descritto nel dettaglio successivamente) per valutare la *scalabilità teorica* del sistema a fronte di un numero elevato di richieste simultanee.

3.9.1 Estensione del testing unitario

L'infrastruttura di test è stata potenziata per includere le nuove funzionalità di gamification e tipologie di questionari:

- **Model e Service**: sono stati implementati nuovi test unitari per le classi di dominio aggiornate (es. calcolo degli EduPoints e gestione delle materie) e per le nuove classi aggiunte nel corso della seconda iterazione.
- **Repository**: in questa iterazione sono stati introdotti i test per i Repository. Per mantenere i test snelli e indipendenti dall'infrastruttura, si è scelto di **simulare l'accesso al database tramite Mock**. Questo approccio ha permesso di verificare che le chiamate ai metodi di persistenza restituissero i dati attesi senza dover configurare un database reale o in-memory (come H2), velocizzando l'esecuzione della suite di test.
- **Validazione del Front-end**: la correttezza dei Controller Web e delle viste Thymeleaf è stata verificata tramite sessioni di **User Acceptance Testing** (UAT). Attraverso prove manuali dirette sulla UI, è stata accertata la corretta visualizzazione delle dashboard e la gestione dei flussi di navigazione, confermando che il sistema risponde ai requisiti funzionali dal punto di vista dell'utente finale.

3.9.2 Stress Test: Modulo Registrazione Utenti

Per verificare la robustezza e la scalabilità del sistema EduQuest, è stato condotto uno stress test focalizzato sulla procedura di registrazione dei nuovi account. Questa operazione è stata scelta in quanto rappresenta una delle attività più critiche, coinvolgendo scritture multiple sul database e la gestione di vincoli di integrità.

3.9.2.1 Configurazione del test

Il test è stato eseguito utilizzando **Apache JMeter**, simulando un carico di lavoro intensivo con le seguenti specifiche:

- **Target**: Endpoint `POST /register`
- **Payload**: dati dinamici (nome, cognome, username, email) caricati tramite file CSV per simulare utenti reali ed evitare conflitti di unicità.
- **Carico**: 100 richieste sequenziali/simultanee inviate in un arco di tempo ridotto (Ramp-up).

3.9.2.2 Analisi dei risultati

I dati raccolti durante la simulazione sono riassunti nella tabella seguente, ottenuta tramite l'ascoltatore *Summary Report* di JMeter:

Label	Samples	Average (ms)	Min (ms)	Max (ms)	Std. Dev.	Error %	Throughput
HTTP Request	100	1527	7	50626	8616.10	0.00%	2.0/min
TOTAL	**100**	**1527**	**7**	**50626**	**8616.10**	**0.00%**	**2.0/min**

Tabella 3.1: Risultati dello Stress Test - Modulo Registrazione

Il dato più significativo è l'assenza totale di errori (**Error 0.00%**). Il tempo di risposta medio si attesta sui **1527 ms**, un valore che indica una gestione efficiente delle transazioni per singolo utente. Nonostante il picco massimo registrato (probabilmente dovuto alla latenza di cold-start o alla gestione dei primi vincoli di database), il server ha processato correttamente l'intero carico. La stabilità del sistema (Resilience) è dimostrata dalla capacità di mantenere l'integrità dei dati senza generare eccezioni di timeout o rifiuti di connessione.

Capitolo 4

Analisi Statica

Per garantire che il codice prodotto fosse non solo funzionante, ma anche manutenibile e privo di vulnerabilità, abbiamo sottoposto l'intero progetto a processi di analisi statica. Questo ci ha permesso di individuare "code smells", potenziali bug e criticità strutturali senza dover eseguire l'applicazione.

4.1 Analisi con SonarQube

Abbiamo utilizzato SonarQube per ottenere una panoramica oggettiva sulla qualità del codice di EduQuest. Di seguito si riporta il risultato dell'ultima analisi effettuata sul codice (tag per la consegna **v1.0**):

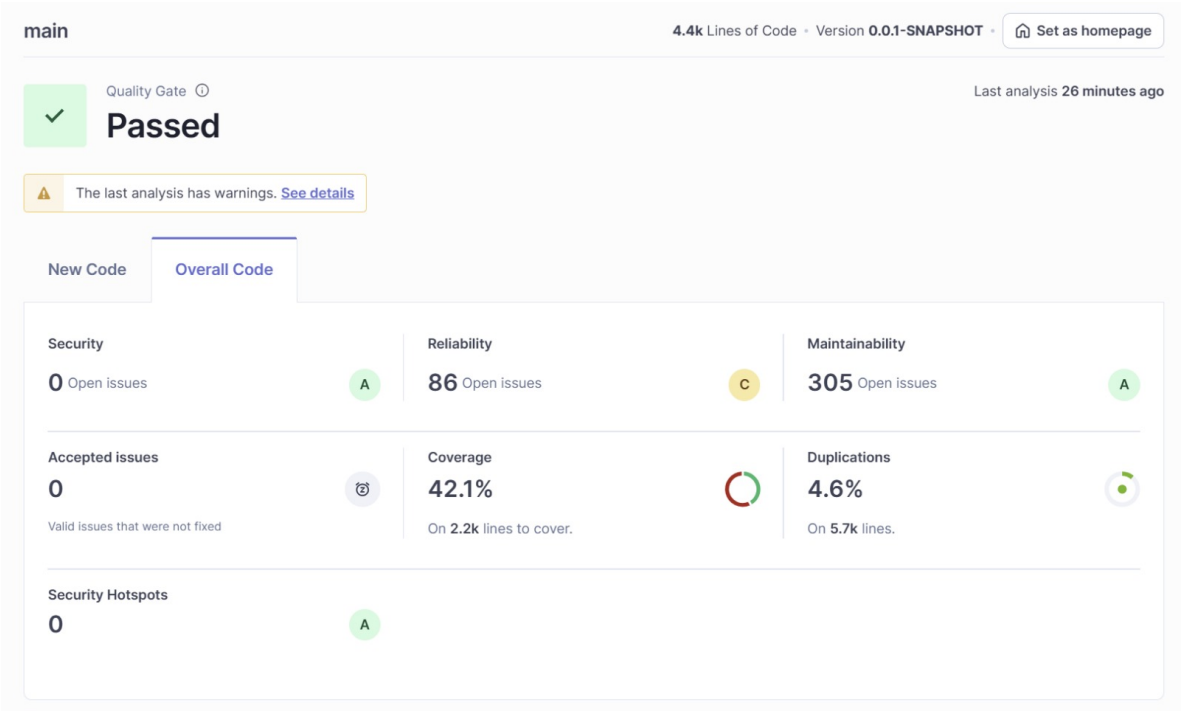

Figura 4.1: Risultato analisi finale SonarQube

Come si può osservare dallo screenshot dell'analisi finale effettuata sul ramo main:

- **Quality Gate Passed**: il progetto ha superato con successo i criteri di qualità impostati, confermando la solidità generale dell'implementazione.
- **Security e Maintainability**: abbiamo ottenuto il massimo voto (Rating A) in entrambe le categorie. Non sono state rilevate vulnerabilità di sicurezza ("0 Open issues") e il debito tecnico è stato mantenuto sotto controllo nonostante i tempi ristretti.
- **Reliability**: il rating ottenuto è C con 86 open issues. Si tratta per lo più di segnalazioni minori legate alla gestione delle eccezioni o all'uso della *Dependency Injection*.
- **Coverage**: la copertura dei test si attesta al 42.1%. ebbene possa sembrare un valore migliorabile, abbiamo dato priorità ai test sulla logica di business (Service) e sui Repository tramite Mock, lasciando la verifica del Front-end allo UAT manuale.
- **Duplicazioni**: il tasso di codice duplicato è molto basso (4.6%), a conferma di un buon uso dei principi di astrazione e della gerarchia delle classi.

4.2 Qualità Metrica e Analisi Strutturale con Understand

L'analisi statica, condotta con *SciTools Understand* su una base di codice di 12.684 righe e 123 file, ha evidenziato una struttura complessivamente robusta. La **Parse Accuracy del 100%** conferma l'affidabilità delle metriche estratte.

4.2.1 Complessità e Distribuzione del Codice

Il progetto mantiene un profilo di complessità eccellente (**Rating Overall A**). L'analisi di dettaglio mostra:

- **Controllo della Complessità Ciclomatica:** Il metodo più complesso, `aggiornaAccount`, registra un valore di **9**, mantenendosi sotto la soglia di rischio (solitamente 10-15). Questo garantisce un'elevata testabilità e manutenibilità del nostro codice.
- **Identificazione degli Hotspot:** La *Metrics Treemap* (Fig. 4.2) evidenzia una disparità volumetrica. La classe `AccountService.java` appare significativamente più grande delle altre (il "Largest File" con oltre 650 righe), pur mantenendo una densità di complessità (colore blu) non allarmante. Tuttavia, questa concentrazione suggerisce che la classe potrebbe agire come *God Class*, accentrando logiche di validazione e business che potrebbero essere ulteriormente decomposte.

4.2.2 Architettura

La struttura delle directory conferma l'adozione di un'architettura a livelli (*Layered Architecture*) conforme al pattern MVC applicato.

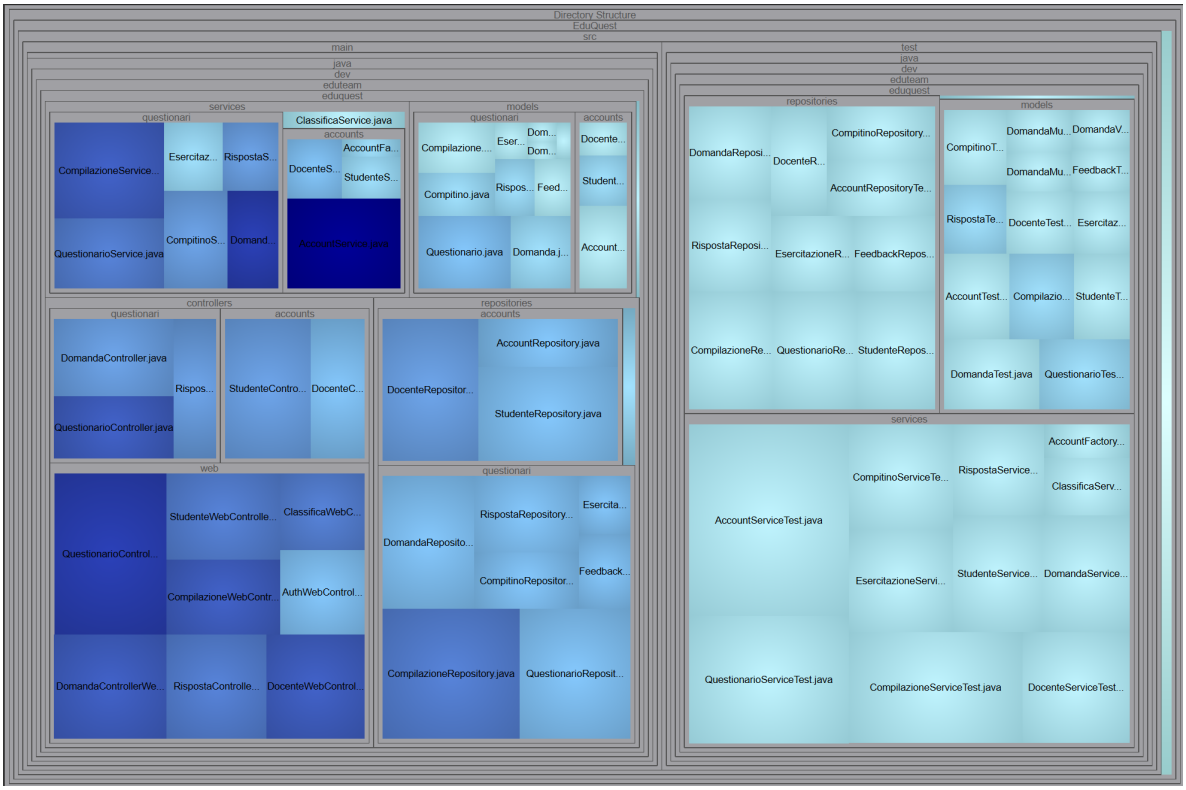

Figura 4.2: Metrics Treemap (Dimensione: Lines of Code, Colore: Max Cyclomatic)

4.2.3 Debito Tecnico

Un punto di attenzione emerge dal **rapporto Commenti/Codice**, che si attesta al **4.6%** (371 righe di commento su 8.002 di codice). Sebbene il codice risulti "pulito" (basso debito tecnico), tale percentuale è inferiore agli standard industriali (10-15%), suggerendo un possibile futuro refactoring per aumentare la comprensione del codice. Sebbene il grafo delle dipendenze mostri un accoppiamento fisiologico nel "core" dell'applicazione, la separazione logica tra *Controller*, *Service* e *Repository* è rispettata.

Classe	Linee di Codice (LOC)	Complessità Max	Rating
AccountService.java	∼660	9	A
AccountServiceTest.java	∼480	4	A
CompilazioneService.java	∼320	6	A
DomandaRepository.java	∼250	3	A
QuestionarioController.java	∼220	7	A

Tabella 4.1: Analisi dimensionale e qualitativa delle classi critiche (Dati: SciTools Understand).

4.2.4 Sintesi Valutativa

In conclusione, **EduQuest** presenta una qualità del codice elevata, priva di violazioni critiche o debito tecnico bloccante. La gestione della complessità è ottimale anche nei componenti più voluminosi. Gli interventi futuri dovrebbero focalizzarsi esclusivamente sul *refactoring* di `AccountService` per distribuire meglio le responsabilità e sull'incremento della copertura documentale.

Capitolo 5

Conclusioni

5.1 Organizzazione del lavoro e metodologia

Nonostante i vincoli temporali dettati dalla sessione d'esami, il gruppo ha adottato un approccio iterativo ed evolutivo ispirato al **Processo Unificato** (UP). La gestione del progetto è stata strutturata per massimizzare la collaborazione e garantire la coerenza del prodotto finale attraverso le seguenti pratiche:

- **Divisione in task**: come tracciato nel diagramma di Gantt (che alleghiamo alla documentazione), il lavoro di progetto è stato suddiviso in task. Ogni task è stato assegnato a una coppia di componenti del gruppo. In tal modo è stata anche applicata la pratica agile del **Pair Programming**, che ha permesso di confrontarci sul codice e limitare gli errori dettati da incompresioni tra i componenti del gruppo.
- **Gestione del versionamento**: lo sviluppo sul repository GitHub è stato tracciato utilizzando una strategia basata su branch separati per ogni task. Questo ci ha permesso di lavorare in parallelo senza interferenze, mantenendo stabile il ramo principale (main). Negli ultimi giorni dello sviluppo, il lavoro finale di refactoring e fix di alcuni bug è stato effettuato direttamente sul main poiché non è stato considerato necessario creare un branch apposito.
- **Technical Review e merge**: i merge dei branch sono stati effettuati esclusivamente durante meeting di Technical Review. In queste occasioni, il gruppo ha revisionato insieme il codice e risolto eventuali conflitti, procedendo all'integrazione solo dopo una validazione condivisa.
- **Divisione dei Ruoli**: per ottimizzare i tempi, a partire dalla seconda iterazione Manca e Deiana si sono focalizzati sullo sviluppo del front-end e sull'integrazione con Thymeleaf, mentre Fonte e Gorla hanno curato le rimanenti logiche di back-end, l'analisi statica e la stesura della documentazione tecnica.

5.2 Il ruolo dell'AI nel progetto

In linea con un approccio moderno allo sviluppo software, abbiamo deciso di integrare l'uso dell'Intelligenza Artificiale come strumento di supporto in alcune fasi del progetto. L'IA non è stata utilizzata per sostituire il nostro lavoro di analisi o di programmazione, ma è servita come un "assistente" per velocizzare alcune operazioni ripetitive o per superare piccoli ostacoli tecnici:

- **Popolamento del database**: abbiamo sfruttato l'IA per generare dataset per le tabelle del nostro database. Questo ci ha permesso di testare il nostro sistema con una mole (per quanto limitata) di dati realistica, che sarebbe stata tediosa da scrivere manualmente.

- **Supporto al problem solving**: in alcuni momenti di "blocco" durante l'implementazione di metodi particolarmente complessi (specialmente nelle query più articolate), abbiamo consultato l'IA per esplorare diverse soluzioni logiche o per capire meglio come correggere determinati errori di sintassi.
- **Revisione della documentazione**: l'IA ci ha fornito un supporto utile anche nella fase finale di stesura di questa relazione, aiutandoci a revisionare alcune parti di testo per assicurarci che l'esposizione fosse chiara, fluida e grammaticalmente corretta.
- **Consapevolezza e limitazioni**: la scelta di farci aiutare dall'IA è arrivata anche a seguito del task proposto durante il corso sull'analisi di codice prodotto dall'AI. Grazie a tale task, siamo diventati consapevoli dei limiti di questi strumenti nell'ambito dell'Ingegneria del Software. Sappiamo che l'IA può generare soluzioni non ottimali ed è per questo che ogni suo suggerimento è stato analizzato, verificato e confermato prima di essere integrato nel progetto.

5.3 Sviluppi futuri

Il gruppo ha scelto deliberatamente di implementare un set limitato di requisiti per garantire un software stabile, completo e privo di bug, evitando la frammentazione dovuta a un eccessivo numero di funzionalità non rifinite. Tuttavia, l'architettura di EduQuest è stata pensata per essere estensibile, lasciando spazio a evoluzioni future:

- **Personalizzazione del profilo**: implementazione di uno "Store" virtuale dove gli studenti possono spendere gli EduPoints accumulati per sbloccare elementi estetici, come badge esclusivi, avatar personalizzati o temi grafici per la propria dashboard.
- **Classi**: introduzione di classi per ogni Docente del sistema EduQuest. Gli studenti possono liberamente iscriversi alla classe di un Docente o esservi invitati. In tal modo, i questionari creati da uno specifico Docente sarebbero compilabili solo dai componenti della sua classe.

5.4 Considerazioni finali

Sviluppare EduQuest ci ha permesso di applicare tutto quello che abbiamo appreso durante il corso di Ingegneria del software e nel corso del nostro percorso di laurea. Come primo vero approccio ad un progetto concreto, passare dalla teoria alla pratica non è stato immediato. Ci siamo scontrati con alcuni dei problemi reali della programmazione, come la gestione dei dati e i dubbi sull'interfaccia utente, che ci hanno fatto capire quanto sia fondamentale una buona progettazione prima ancora di scrivere il codice.

È stato particolarmente stimolante il lavoro di ricerca che abbiamo dovuto fare per implementare le varie parti del sistema. Non ci siamo limitati a scrivere righe di codice, ma abbiamo cercato di farlo nel modo giusto, applicando le competenze apprese. Durante lo sviluppo abbiamo integrato diversi pattern architetturali e design pattern (che abbiamo citato nei vari capitoli), cercando di rendere il software il più possibile scalabile e facile da mantenere.

In conclusione, questa esperienza è stata utilissima per imparare a lavorare davvero in team e a gestire le responsabilità all'interno di esso. Ci teniamo a ringraziare la prof.ssa Arcelli e il prof. Riganelli per l'opportunità di poter finalmente "sporcarci le mani" e applicare le competenze acquisite in un progetto concreto, vedendo una web app prendere vita da zero.