Corso di Laurea
in
Ingegneria Informatica

Progettazione e Sviluppo di Sistemi Software
Elaborato d'esame

prof. Anna Rita Fasolino

M63000989	Fabio d'Andrea
M63000986	Guido Di Chiara
M63001040	Antimo Iannucci
M63001026	Dario Daniele

Università degli Studi di Napoli "Federico II"

Scuola Politecnica e delle Scienze di Base

Anno Accademico 2019/2020

Primo Semestre

Indice

Capitolo 1

Processo di Sviluppo

Si è scelto di utilizzare un processo di sviluppo agile, seguendo le indicazioni fornite dal framework Scrum.

1.1 Scrum

Come previsto dalla Scrum Guide [1], lo scrum team dedicato al progetto è composto dai seguenti *ruoli*:

- **Product owner**: figura responsabile di massimizzare il valore del prodotto da realizzare ed unico responsabile del product backlog.
- **Scrum master**: figura responsabile di promuovere e sostenere Scrum, aiutando il team a comprendere appieno i principi su cui si fonda. Ha il compito di rimuovere gli impedimenti riscontrati dal team di sviluppo, gestendo le interazioni tra coloro al di fuori del team e il team stesso.
- **Development team**: insieme di professionisti il cui compito è quello di consegnare incrementi del prodotto potenzialmente rilasciabili. La dimensione del team di sviluppo può variare dai 3 ai 9 elementi.

Sebbene consapevoli della separazione esistente trai diversi ruoli, a scopo didattico ciascun membro del team ha ricoperto all'occorrenza ognuno di essi.

Durante il processo di sviluppo si è cercato di rispettare quanto più fedelmente gli *eventi* previsti da Scrum:

- **Sprint** (2 settimane): evento container contenente gli eventi descritti in seguito. All'interno di uno sprint non si distinguono attività di analisi, progettazione e testing, risultando difatti interconnesse.
- **Sprint planning** (4 ore): evento in cui si pianifica il lavoro da svolgere durante uno sprint. In particolare viene definito *cosa* deve essere realizzato e *come* deve essere svolto il lavoro. Durante tale evento viene definito lo **sprint goal**, ovvero l'obiettivo da raggiungere al termine dello sprint.

- **Daily Scrum** (15 min): evento quotidiano in cui il team di sviluppo pianifica il lavoro da svolgere nelle successive 24 ore.
- **Sprint Review** (2 ore): evento in cui viene revisionato l'incremento rilasciato al termine di uno sprint. Lo scrum team illustra agli stakeholders il lavoro completato, al fine di ricevere feedback sull'incremento prodotto e su cosa realizzare nell'incremento successivo.
- **Sprint Retrospective** (1 ora e mezza): evento in cui lo scrum team analizza se stesso al fine di migliorare il proprio rendimento nello sprint successivo.

Nello specifico sono stati realizzati due sprint della durata di due settimane ciascuno.

Durante lo sviluppo sono stati prodotti gli *artefatti* previsti da Scrum:

- **Product Backlog**: elenco ordinato di tutto ciò che è necessario al prodotto. È l'unica fonte a cui bisogna fare riferimento in caso di cambiamenti. È bene osservare che il product backlog non è mai completo, ma evolve contemporaneamente all'ambiente.
- **Sprint Backlog**: insieme degli elementi del product backlog selezionati per lo sprint corrente, a cui si aggiungono tutte le informazione necessarie per consegnare l'incremento del prodotto e realizzare lo sprint goal.
- **Incremento**: somma di tutti gli elementi del product backlog completati durante lo sprint corrente e di tutti quelli realizzati negli sprint precedenti. È bene osservare che un incremento rilasciato al termine di uno sprint deve essere sempre potenzialmente rilasciabile.

Oltre agli artefatti richiesti da Scrum, per scopi didattici sono stati prodotti diversi diagrammi UML, cercando di mantenere comunque un approccio agile. Per questo motivo è stato prodotto solo lo stretto necessario per una piena comprensione del sistema. La realizzazione dei diagrammi è stata integrata all'interno del product backlog stesso. In altre parole, la stesura di un diagramma è stata vista come un elemento del backlog.

Un altro importante concetto di Scrum da definire è stato la cosiddetta **Definition of Done**, ovvero il criterio di valutazione secondo cui un item del backlog può ritenersi effettivamente "fatto". Si è scelto come definition of done la verifica che la funzionalità implementata rispetti le specifiche richieste.

1.2 Pratiche Agili

Nel contesto di Scrum sono state applicate diverse pratiche agili tra cui vale la pena menzionare:

- *comunicazione stretta* tra membri del team;
- *sviluppo iterativo*;
- *pair programming*;

- *refactoring* del codice;
- *semplicità* di progettazione, modellazione, documentazione e codice;
- *time-boxing*, inteso come suddivisione del progetto in intervalli temporali ben definiti;

1.3 Strumenti Software di Supporto

1.3.1 Jira

Come supporto all'intero processo di sviluppo è stato adottato la piattaforma **Jira Software**. In particolare tra le funzionalità offerte sono state utilizzate le seguenti:

- definizione del product backlog;
- definizione di un *epic*;
- definizione di una *storia utente*;
- definizione di un *task*;
- assegnazione di *priorità* ad una storia utente;
- assegnazione di *story points* ad una storia utente;
- assegnazione di *sottotask* ad un task o ad una storia utente;
- assegnazione di una storia utente ad un epic;
- avvio di uno sprint

Per accedere a tutte le informazioni di progetto contenute su Jira è possibile utilizzare il seguente link:

`https://bit.ly/3k0YHhx`

Le credenziali di accesso al progetto sono:

E-mail	**Password**
progettopsssunina@gmail.com	progetto2020

1.3.2 GitHub

Per supportare la collaborazione tra i diversi membri del team è stato utilizzato il *version control system* **GitHub**. In questo modo è stato possibile lavorare in parallelo sul codice senza avere particolari conflitti. Al seguente link è possibile accedere al repository del progetto:

`https://github.com/elaboratopsss/registroelettronico`

Capitolo 2

Avvio del Progetto

Si vuole realizzare un sistema software per gestire un registro elettronico di un istituto scolastico secondario di primo o secondo grado.

2.1 Parti Interessate

Il registro elettronico è organizzato in modo tale da essere utilizzato da 3 attori: **Segretario**, **Docente** e **Studente**.

Il segretario deve poter gestire l'istituto nel suo complesso, a partire dalla definizione delle entità in esso coinvolte (materie, classi, docenti, studenti, etc).

Docenti e studenti devono poter usufruire di tutti gli strumenti digitali di supporto alla didattica tradizionale.

2.2 Obiettivi Utente

2.2.1 Segretario

Il sistema deve consentire al segretario della scuola di gestire (creare, aggiornare o eliminare) le materie, le classi, i docenti e gli insegnamenti. Il segretario gestisce inoltre l'iscrizione di uno studente. Infine, il segretario deve poter gestire l'orario di una classe e le comunicazioni generali dell'istituto.

2.2.2 Docente

Il sistema deve fornire al docente, oltre ad un'area personale dove poter visualizzare i propri dati, tutte le funzionalità di gestione di un insegnamento in una classe.

In particolare, un docente deve poter visualizzare l'orario delle lezioni che deve sostenere ed annotare eventuali assenze, ritardi e note disciplinari di uno studente.

Un docente deve poter registrare i voti di verifiche orali e scritte e tenere sotto controllo l'andamento didattico di uno studente, visualizzando la sua scheda personale. Inoltre, al termine del periodo scolastico, deve poter registrare la valutazione finale di ogni studente.

Il sistema deve permettere al docente di gestire un registro di classe digitale, consentendo di inserire le attività svolte durante la lezione e tenere traccia del contenuto delle

lezioni passate. Al termine di una lezione il docente deve poter registrare un assegno e fornire del materiale didattico di supporto ai suoi studenti.

Infine, il docente deve poter visualizzare le comunicazioni di istituto.

2.2.3 Studente

Il sistema deve fornire allo studente, oltre ad una area personale dove poter visualizzare i propri dati, tutte le funzionalità per poter monitorare la didattica.

Nel dettaglio lo studente deve poter visualizzare i propri ritardi, assenze e note disciplinari. Inoltre lo studente può visualizzare l'orario della propria classe, gli assegni ed il materiale didattico fornito dai docenti e consultare il registro delle attività giornaliere della classe.

Il sistema deve consentire allo studente di monitorare il proprio andamento didattico visualizzando i voti ricevuti e le valutazioni finali raccolte nella propria pagella.

Lo studente deve essere notificato nel momento in cui si verificano eventi che lo riguardano, come la registrazione di un voto, di un'assenza o di una nota o, più in generale, la pubblicazione di una comunicazione di istituto.

Infine, lo studente deve poter consultare le comunicazioni di istituto.

Capitolo 3

Specifica dei Requisiti

In linea con la metodologia agile adottata, si è scelto di documentare i requisiti del sistema attraverso il meccanismo delle **storie utente**.

3.1 Storie utente

Le storie utente sono state riportate all'interno del product backlog stesso. Ogni storia utente, in quanto elemento del backlog, è caratterizzata da:

- *titolo*;
- *descrizione*: breve frase che descrive una funzionalità del prodotto dal punto di vista di un attore specifico, scritta usando il seguente formato:

  Come [attore]

  Io voglio [funzionalità]

  In modo da [motivo]

- *priorità*: indica il grado di importanza della storia utente all'interno del sistema. Storie utente con maggiore priorità rappresentano funzionalità da implementare prima nel processo di sviluppo.
- *story points*: unità di misura per esprimere una stima dello sforzo richiesto per implementare completamente la storia utente.
  Per l'assegnazione degli story points è stato seguito un processo ben definito. Per prima cosa è stato definito un insieme discreto di valori numerici, selezionando i primi nove numeri della sequenza di Fibonacci: 1, 2, 3, 5, 8, 13, 21, 34 e 55. Si è preferita questa scelta per via del più ampio margine tra i valori della sequenza rispetto ad una sequenza lineare, permettendo di fatto una più facile valutazione. Successivamente, seguendo il cosiddetto *approccio triangolare*, sono state individuate tre storie utente all'interno del backlog associate rispettivamente al minimo (1), medio (8) e massimo (55) valore di priorità. Alle restanti storie utente è stato associato un valore confrontandole con le tre definite inizialmente.
- *epic link* (opzionale): epic a cui la storia utente appartiene;

- *assegnatario* (opzionale), membro del team incaricato di implementare la storia utente;
- *etichette* (opzionali): parole chiave associate alla storia utente;

Di seguito si riportano le storie utente realizzate per la specifica dei requisiti, organizzate negli sprint in cui sono state implementate.

3.1.1 Sprint 1

Si è scelto di implementare in tale sprint la preliminare funzionalità di autenticazione, nonché le storie utente che permettono al segretario di creare ed aggiornare le principali entità del sistema, quali studente, docente, materia e classe.

Sprint Goal: implementare le funzionalità principali del sottosistema che permette al segretario di inizializzare l'applicazione.

Storia Utente	
Titolo	Autenticazione utente
Descrizione	*Come* segretario/docente/studente *Io voglio* autenticarmi *In modo da* poter accedere alle funzionalità del sistema
Priorità	Massima
Story Points	13
Epic link	Autenticazione
Etichette	Requisito Segretario Docente Studente

Tabella 3.1: Storia Utente: Autenticazione utente

Da tale storia utente ne derivano i requisiti non funzionali riportati nel Paragrafo 3.3.3.

Storia Utente	
Titolo	Aggiungi materia
Descrizione	*Come* segretario *Io voglio* inserire una nuova materia *In modo da* mantenere una lista di materie aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Materie
Etichette	Requisito Segretario

Tabella 3.2: Storia Utente: Aggiungi materia

Storia Utente	
Titolo	Aggiorna materia
Descrizione	*Come* segretario *Io voglio* aggiornare i dati di una materia disciplinare *In modo da* mantenere una lista di materie aggiornate
Priorità	Alta
Story Points	3
Epic link	Gestione Materie
Etichette	Requisito Segretario

Tabella 3.3: Storia Utente: Aggiorna materia

Storia Utente	
Titolo	Cerca materia per nome
Descrizione	*Come* segretario *Io voglio* cercare una materia disciplinare specificandone il nome *In modo da* poterne visualizzare i dettagli
Priorità	Alta
Story Points	3
Epic link	Gestione Materie
Etichette	Requisito Segretario

Tabella 3.4: Storia Utente: Cerca materia per nome

Storia Utente	
Titolo	Aggiungi studente
Descrizione	*Come* segretario *Io voglio* inserire un nuovo studente *In modo da* tenere una lista di studenti aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Studenti
Etichette	Requisito Segretario

Tabella 3.5: Storia Utente: Aggiungi studente

Storia Utente	
Titolo	Aggiorna studente
Descrizione	*Come* segretario *Io voglio* aggiornare i dati di uno studente *In modo da* tenere una lista di studenti aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Studenti
Etichette	Requisito Segretario

Tabella 3.6: Storia Utente: Aggiorna studente

Storia Utente	
Titolo	Cerca studente per nome
Descrizione	*Come* segretario *Io voglio* cercare uno studente specificandone il nome *In modo da* tenerne visualizzare i dettagli
Priorità	Alta
Story Points	3
Epic link	Gestione Studenti
Etichette	Requisito Segretario

Tabella 3.7: Storia Utente: Cerca studente per nome

Storia Utente	
Titolo	Aggiungi docente
Descrizione	*Come* segretario *Io voglio* inserire un nuovo docente *In modo da* tenere una lista aggiornata di docenti
Priorità	Alta
Story Points	3
Epic link	Gestione Docenti
Etichette	Requisito Segretario

Tabella 3.8: Storia Utente: Aggiungi docente

Storia Utente	
Titolo	Aggiorna docente
Descrizione	*Come* segretario *Io voglio* aggiornare i dati di un docente *In modo da* tenere una lista aggiornata di docenti
Priorità	Alta
Story Points	3
Epic link	Gestione Docenti
Etichette	Requisito Segretario

Tabella 3.9: Storia Utente: Aggiorna docente

Storia Utente	
Titolo	Cerca docente per nome
Descrizione	*Come* segretario *Io voglio* cercare un docente specificandone il nome *In modo da* poterne visualizzare i dettagli
Priorità	Alta
Story Points	3
Epic link	Gestione Docenti
Etichette	Requisito Segretario

Tabella 3.10: Storia Utente: Cerca docente per nome

Storia Utente	
Titolo	Aggiungi classe
Descrizione	*Come* segretario *Io voglio* inserire una nuova classe *In modo da* tenere una lista di classi aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.11: Storia Utente: Aggiungi classe

Storia Utente	
Titolo	Aggiorna classe
Descrizione	*Come* segretario *Io voglio* aggiornare i dati di una classe *In modo da* tenere una lista di classi aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.12: Storia Utente: Aggiorna classe

Storia Utente	
Titolo	Cerca classe per anno e sezione
Descrizione	*Come* segretario *Io voglio* cercare una classe specificandone l'anno e la sezione *In modo da* poterne visualizzare i dettagli
Priorità	Alta
Story Points	3
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.13: Storia Utente: Cerca classe per anno e sezione

3.1.2 Sprint 2

Si è scelto in tale sprint di completare le funzionalità di gestione di studenti, docenti, materie e classi, realizzando le storie utente di rimozione di questi ultimi. Inoltre, sono state sviluppate alcune storie utente relative alla creazione di un insegnamento ed alla visualizzazione di informazioni legate a classi e materie.

Sprint Goal: estendere le funzionalità del sottosistema che permette al segretario di inizializzare l'applicazione.

Storia Utente	
Titolo	Rimuovi materia
Descrizione	*Come* segretario *Io voglio* rimuovere una materia disciplinare *In modo da* mantenere una lista di materie aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Materie
Etichette	Requisito Segretario

Tabella 3.14: Storia Utente: Rimuovi materia

Storia Utente	
Titolo	Rimuovi studente
Descrizione	*Come* segretario *Io voglio* rimuovere un nuovo studente *In modo da* tenere una lista di studenti aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Studenti
Etichette	Requisito Segretario

Tabella 3.15: Storia Utente: Rimuovi studente

Storia Utente	
Titolo	Rimuovi docente
Descrizione	*Come* segretario *Io voglio* rimuovere un docente *In modo da* tenere una lista aggiornata di docenti
Priorità	Alta
Story Points	3
Epic link	Gestione Docenti
Etichette	Requisito Segretario

Tabella 3.16: Storia Utente: Rimuovi docente

Storia Utente	
Titolo	Rimuovi classe
Descrizione	*Come* segretario *Io voglio* rimuove una nuova classe *In modo da* tenere una lista di classi aggiornata
Priorità	Alta
Story Points	3
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.17: Storia Utente: Rimuovi classe

Storia Utente	
Titolo	Aggiungi insegnamento
Descrizione	*Come* segretario *Io voglio* aggiungere un insegnamento ad una classe *In modo da* tenere traccia dei docenti che insegnano nella classe
Priorità	Alta
Story Points	3
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.18: Storia Utente: Aggiungi insegnamento

Storia Utente	
Titolo	Visualizzare docenti materia
Descrizione	*Come* segretario *Io voglio* visualizzare i docenti che possono insegnare una materia *In modo da* poter assegnare loro un insegnamento
Priorità	Media
Story Points	1
Epic link	Gestione Materie
Etichette	Requisito Segretario

Tabella 3.19: Storia Utente: Visualizzare docenti materia

Storia Utente	
Titolo	Visualizza studenti classe
Descrizione	*Come* segretario *Io voglio* visualizzare gli studenti appartenenti ad una classe *In modo da* garantirne la coerenza
Priorità	Media
Story Points	1
Epic link	Gestione Classi
Etichette	Segretario

Tabella 3.20: Storia Utente: Visualizza studenti classe

Storia Utente	
Titolo	Visualizza insegnamenti classe
Descrizione	*Come* segretario *Io voglio* visualizzare i docenti e le materie da loro insegnare in una classe *In modo da* garantirne la coerenza
Priorità	Media
Story Points	1
Epic link	Gestione Classi
Etichette	Requisito Segretario

Tabella 3.21: Storia Utente: Visualizza insegnamenti classe

3.2 Epic

Un **epic** contiene diverse storie utente legate semanticamente tra loro. È quindi più generico rispetto ad una storia utente ed è descritto attraverso un livello di dettaglio più alto. Tipicamente le storie utente di un epic sono implementate nel corso di più sprint. Ogni epic riporta i seguenti campi:

- *titolo*;
- *riepilogo*;
- *storie utente*;

Di seguito si riportano gli epic realizzati.

Epic	
Titolo	Autenticazione
Riepilogo	Autenticazione dei diversi utenti del sistema
Storie Utente	Autenticazione utente Recupero credenziali

Tabella 3.22: Epic: Autenticazione

Epic	
Titolo	Gestione materie
Riepilogo	Gestione delle materie dell'istituto da parte del segretario
Storie Utente	Aggiungi materia Rimuovi materia Aggiorna materia Cerca materia per nome Visualizza docenti materia

Tabella 3.23: Epic: Gestione materie

Epic	
Titolo	Gestione studenti
Riepilogo	Gestione degli studenti dell'istituto da parte del segretario
Storie Utente	Aggiungi studente Rimuovi studente Aggiorna studente Cerca studente per nome

Tabella 3.24: Epic: Gestione studenti

Epic	
Titolo	Gestione classi
Riepilogo	Gestione delle classi dell'istituto da parte del segretario
Storie Utente	Aggiungi classe Rimuovi classe Aggiorna classe Cerca classe per anno e sezione Aggiungi insegnamento Rimuovi insegnamento

Tabella 3.25: Epic: Gestione classi

Epic	
Titolo	Gestione docenti
Riepilogo	Gestione dei docenti dell'istituto da parte del segretario
Storie Utente	Aggiungi docente Rimuovi docente Aggiorna docente Cerca docente per nome

Tabella 3.26: Epic: Gestione docenti

Le storie utente e gli epic realizzati possono essere reperiti all'interno del product backlog presente su Jira [1].

3.3 Altri Requisiti

Di seguito sono riportati ulteriori requisiti dell'applicazione, organizzati seguendo la classificazione presente in [2].

3.3.1 Requisiti del Prodotto

Requisiti di Usabilità

Il sistema deve essere intuitivo e facile da comprendere, in modo da poter essere utilizzato dai diversi utenti senza la necessità di corsi di formazione.

Requisiti di Affidabilità

Il sistema deve essere sempre disponibile per tutti gli utenti. Il sistema deve poter essere acceduto in concorrenza da diversi utenti.

Requisiti di Portabilità

Il sistema deve essere accessibile tramite un qualsiasi browser, su di un qualsiasi dispositivo connesso ad Internet (PC, smartphone, tablet).

3.3.2 Requisiti Organizzativi

Requisiti di Consegna

Deve essere consegnata una demo del progetto entro un mese.

Requisiti di Implementazione

Il sistema deve essere implementato in linguaggio Java e deve essere sviluppato come applicazione web.

[1] Per il link e le credenziali di accesso fare riferimento al Paragrafo 1.3.1.

3.3.3 Requisiti Esterni

Requisiti di Riservatezza

Ciascun attore del sistema può accedere solo alle aree del sistema ad esso riservate. Docenti e studenti non possono accedere ai dati personali di altri utenti. Un docente può accedere a tutte e sole le informazioni relative alle classi in cui insegna. Uno studente può accedere esclusivamente alla propria carriera scolastica. Tale requisito è richiesto dalla storia utente in Tabella 3.1.

Requisiti di Sicurezza

Le password degli utenti devono essere cifrate. Tale requisito è richiesto dalla storia utente in Tabella 3.1.

Capitolo 4

Analisi dei Requisiti

In questa sezione sono riportati i diagrammi di analisi realizzati. Si vuole ancora sottolineare come si è scelto di realizzare solo i diagrammi strettamente necessari alla comprensione del sistema e delle sue funzionalità più complesse. Inoltre, restando fedeli all'approccio agile, tali diagrammi sono stati rifiniti durante l'intero ciclo di sviluppo, evolvendo insieme alla comprensione stessa del sistema da parte del team.

Alcune delle notazioni utilizzate nei diagrammi stilati all'interno della documentazione sono tratte da [3].

4.1 System Context Diagram

Il primo diagramma realizzato in fase di analisi è il *System Context Diagram*, un diagramma di analisi che mostra esplicitamente i confini tra il sistema, trattato come una black box, e l'ambiente esterno [4].

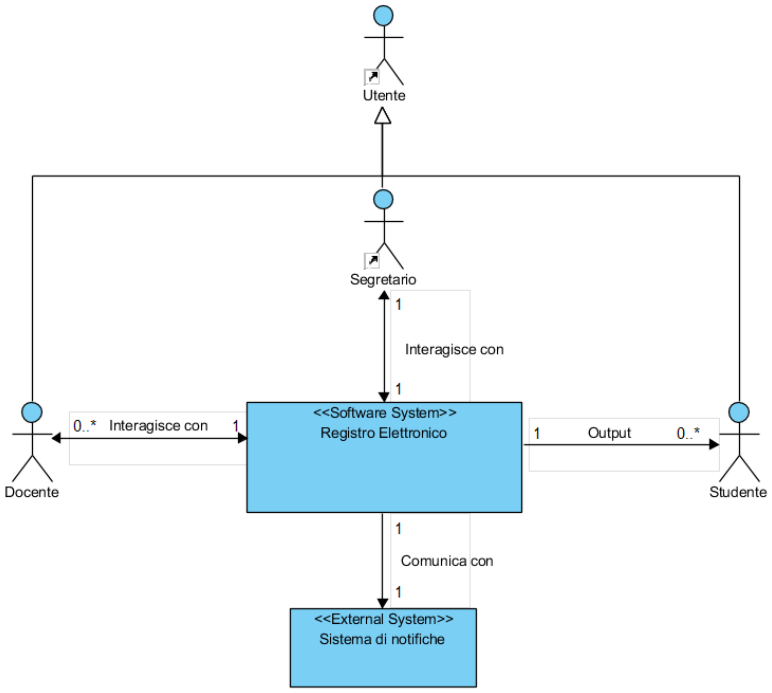

Figura 4.1: System Context Diagram

4.2 Class Diagram

Uno dei diagrammi più importanti nell'analisi dei requisiti è sicuramente il **System Domain Model**, un *Class Diagram* di alto livello in cui sono riportate le principali entità del sistema, le loro responsabilità e le relazioni esistenti tra di esse.

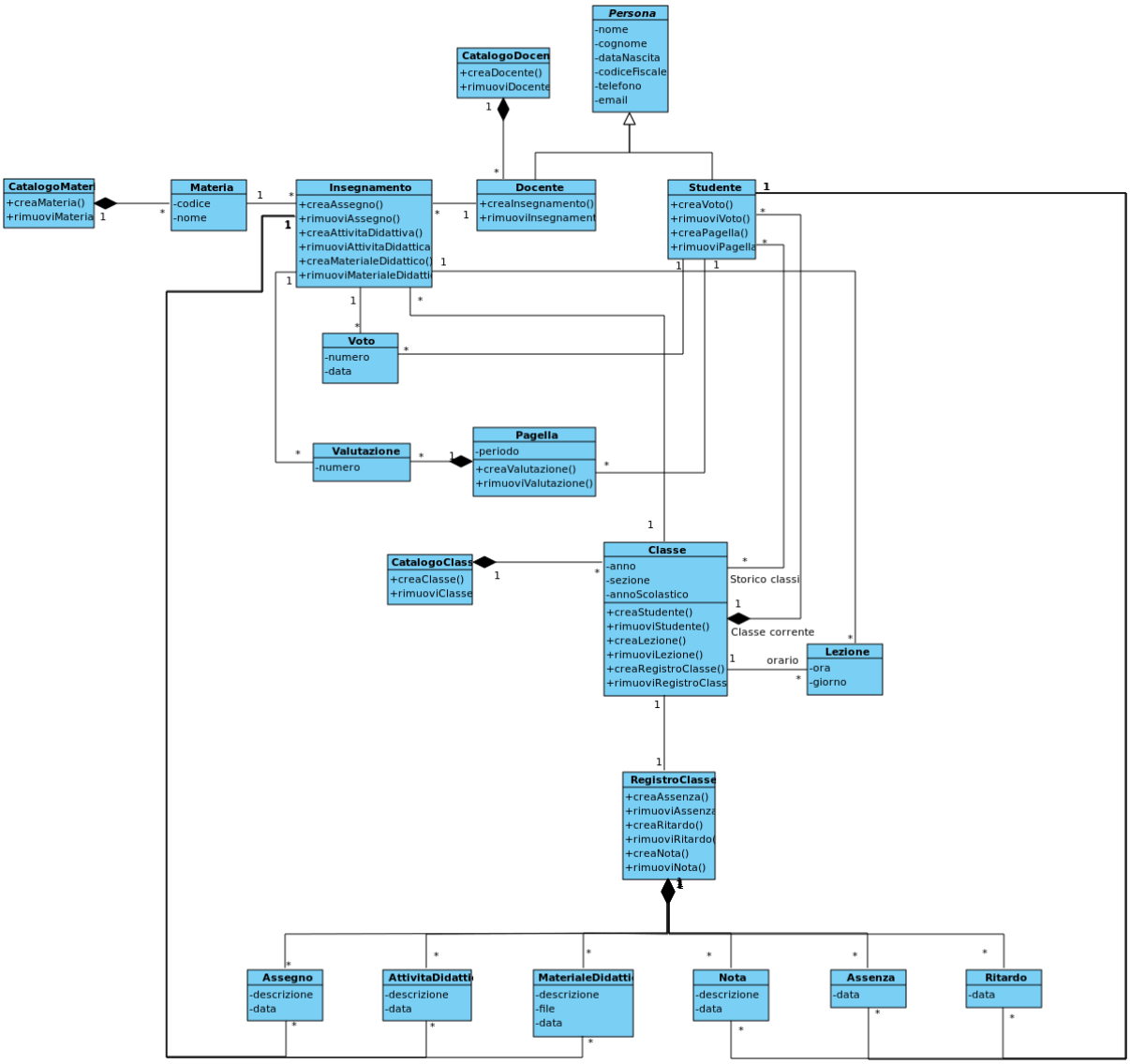

Figura 4.2: Class Diagram: System Domain Model

4.3 Sequence Diagram

Al fine di comprendere le funzionalità di base del sistema sono stati realizzati diversi *Sequence Diagram* di alto livello. In particolare si è scelto di realizzare i diagrammi relativi alla creazione, l'aggiornamento e l'eliminazione di un docente.

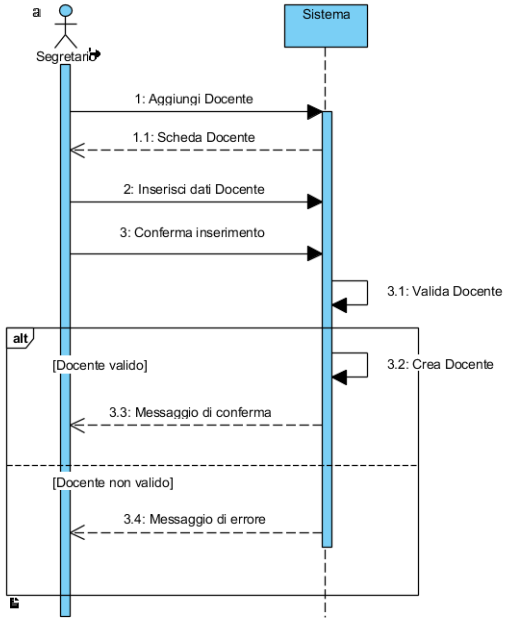

Figura 4.3: Sequence Diagram: Aggiungi Docente

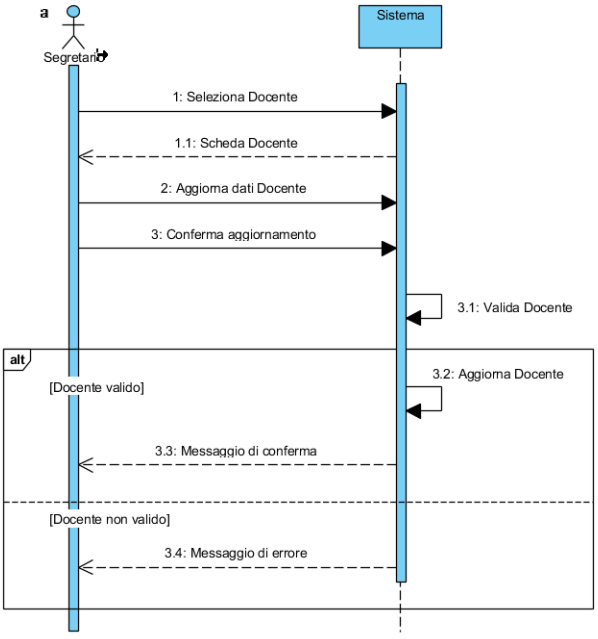

Figura 4.4: Sequence Diagram: Aggiorna Docente

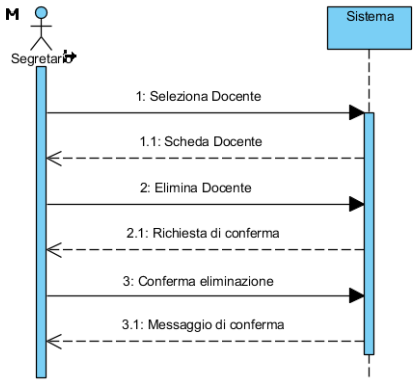

Figura 4.5: Sequence Diagram: Rimuovi Docente

Creazione, aggiornamento ed eliminazione delle altre entità del sistema prevedono interazioni analoghe tra l'utente e il sistema. Per questo motivo, mantenendo uno spirito agile, si è scelto di evitare la stesura dei relativi diagrammi.

Sono stati invece realizzati i diagrammi necessari alla comprensione delle funzionalità più complesse del sistema. Nello specifico, si riporta la creazione di un insegnamento nel contesto di una classe.

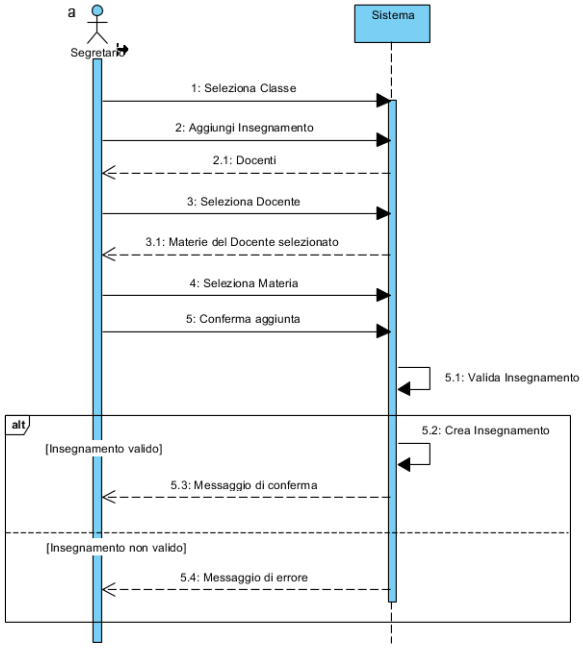

Figura 4.6: Sequence Diagram: Aggiungi Insegnamento

Capitolo 5

Progettazione

Nella sezione corrente è riportato tutto ciò che è inerente alla progettazione del sistema, dai diagrammi di dettaglio alle scelte progettuali. In generale, si è scelto di dividere la progettazione in due fasi distinte:

- **Responsability Driven Design (RDD)**: modello di progettazione legato alla programmazione object-oriented, in cui l'attenzione è posta sulle responsabilità dei diversi oggetti e sulle informazioni che essi condividono.
- **Framework Oriented Design (FOD)**: modello di progettazione orientato alle caratteristiche dei framework scelti.

5.1 Progettazione dell'architettura

Seguendo un approccio *architecture-centric*, la prima attività svolta dopo la fase di analisi dei requisiti è stata la progettazione dell'architettura del sistema. Considerando le caratteristiche del sistema da sviluppare, si è scelto di adottare uno stile architetturale **Client-Server**.

5.1.1 Component Diagram

Si è scelto di proporre una prima vista architetturale mediante il seguente *Component Diagram*.

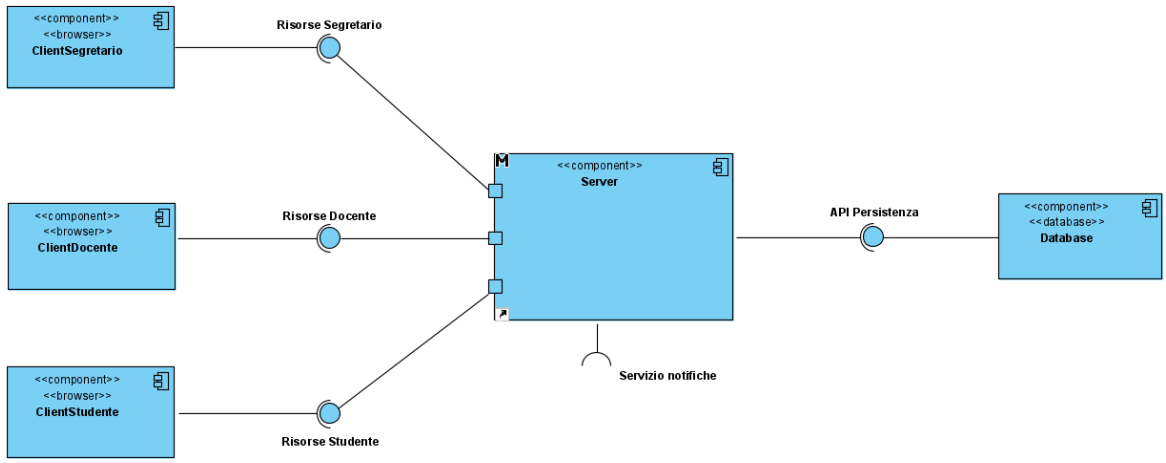

Figura 5.1: Component Diagram: Architettura del Sistema

Come evidenziato dal diagramma, i componenti del sistema sono:

- **Client**: web browser in grado di inoltrare richieste HTTP al server. I diversi client possono essere visti come *thin-client*, in quanto la logica applicativa è concentrata quasi interamente sul server.
- **Server**: web server che espone risorse ai client. Il server richiede un'interfaccia fornita dal sistema di notifiche esterno.
- **Database**: database relazionale che permette al server di gestire la persistenza dei dati.

5.2 Responsability Driven Design (RDD)

Dopo aver definito l'architettura del sistema, si è passati alla progettazione delle sue entità, in termini di classi e oggetti.

5.2.1 Class Diagram

Il System Domain Model è stato raffinato aggiungendo ulteriori classi di progetto, ottenedo un *Class Diagram* di dettaglio definito **System Model RDD**.

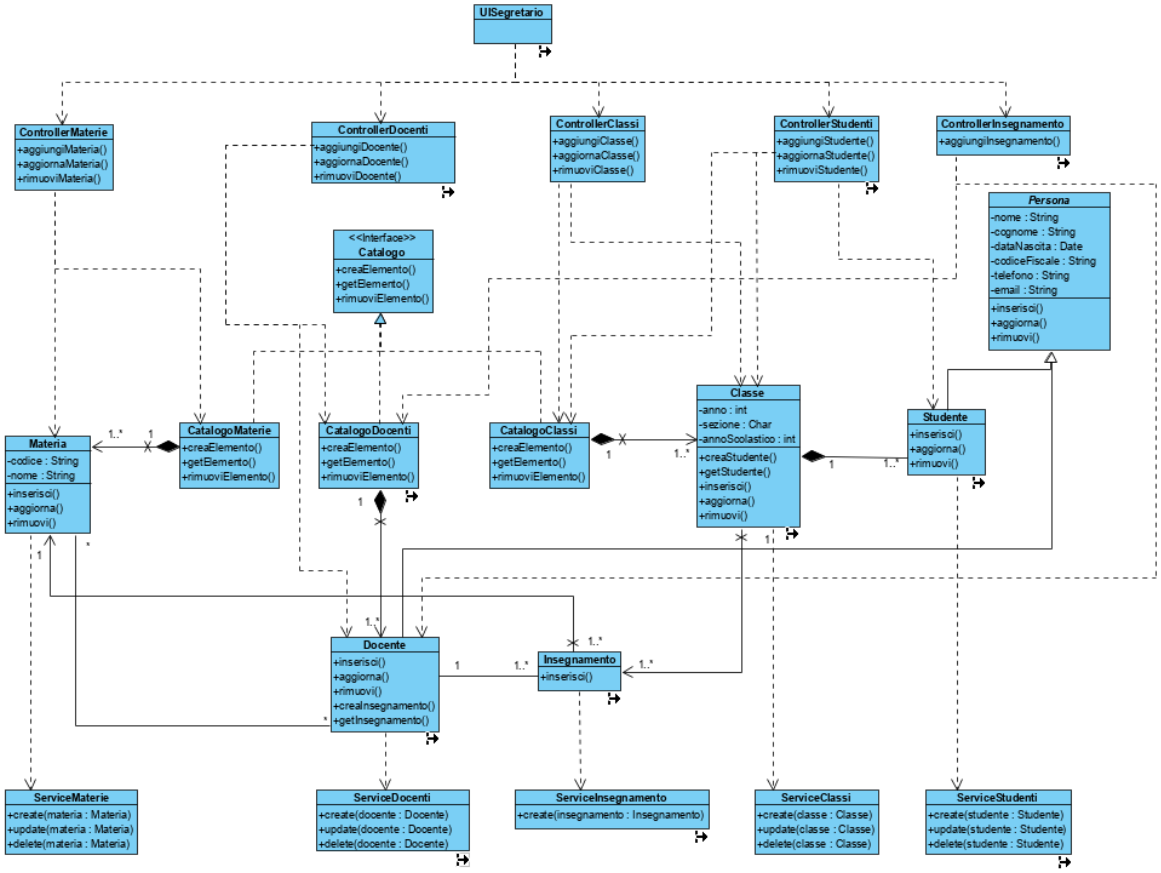

Figura 5.2: Class Diagram: System Model - RDD

Nella realizzazione di tale diagramma sono stati seguiti diversi pattern GRASP, tra cui:

- **Information Expert**: impone di assegnare una responsabilità alla classe che possiede la maggior parte delle informazioni necessarie per poterla soddisfare. Ad esempio, nel diagramma in Figura 5.2, `Classe` è information expert della classe `Studente`, in quanto possiede una lista di insegnamenti.
- **Creator**: impone di assegnare ad una classe `B` la responsabilità di creare istanze di un'altra classe `A` se una o più delle seguenti affermazioni sono valide:
  - un'istanza di `B` contiene istanze di `A`;
  - un'istanza di `B` utilizza istanze di `A`;
  - un'istanza di `B` possiede le informazioni necessarie ad inizializzare istanze di `A`;

  Ad esempio, nel diagramma in Figura 5.2, la classe `Docente` è creator della classe `Insegnamento`, in quanto possiede una lista di insegnamenti.

- **Controller**: impone di assegnare la responsabilità di gestire gli eventi del sistema ad una classe non appartenente all'interfaccia utente. In altre parole, un controller non è altro che la prima classe posta oltre la UI che riceve le richieste dell'utente e coordina le operazioni del sistema. Nel fare ciò, il controller dovrebbe delegare ad altre classi di dominio il lavoro da svolgere per soddisfare le richieste dell'utente. In generale è possibile seguire due possibili approcci per l'applicazione di tale pattern:
  - *Façade Controller*: si prevede un'unica classe controller che rappresenta l'intero sistema;
  - *Session Controller*: si prevedono diverse classi controller, ognuna dei quali rappresenta il sistema nello scenario di uno o più casi d'uso.

  Come riportato nel diagramma in Figura 5.2, si è scelto di utilizzare il secondo approccio, prevedendo un controller per più storie utente correlate tra loro.
- **Pure Fabrication**: consente di definire classi che non rappresentano un concetto nel dominio del problema, in modo da garantire un basso accoppiamento ed un'elevata coesione. Tali classi sono indicate tipicamente come *service*. Nel diagramma in Figura 5.2 alle classi service è stata affidata la logica di accesso al database.
- **Low Coupling**: impone di assegnare le responsabilità alle classi in modo da garantire un basso accoppiamento, dove per accoppiamento si intende quanto una classe dipende da un'altra.
- **High Coesion**: impone di assegnare le responsabilità alle classi in modo da garantire un'alta coesione, dove per coesione si intende quanto le responsabilità di una classe sono fortemente correlate tra loro.
- **Polymorphism**: indica di attribuire la responsabilità di definire delle variazioni dei comportamenti basati sul tipo, ai tipi stessi per i quali avviene tale variazione. Come illustrato nel diagramma in Figura 5.2, il polimorfismo è stato ottenuto definendo un'interfaccia `ICatalogo`, implementata da tutte le classi catalogo.

5.2.2 Sequence Diagram

Definito il *Class Diagram* di progetto sono stati prodotti diversi *Sequence Diagram*, al fine di mostrare le interazioni tra gli oggetti che permettono di realizzare le principali funzionalità del sistema.

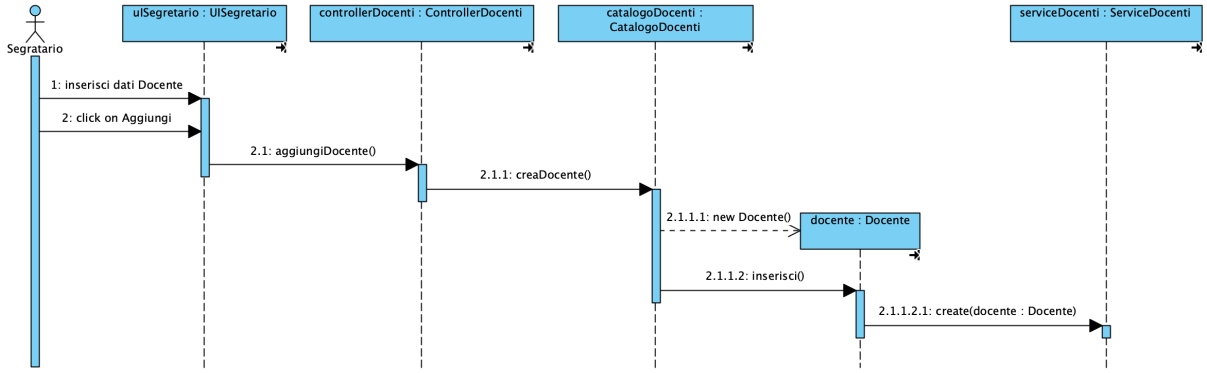

Figura 5.3: Sequence Diagram: Aggiungi Docente - RDD

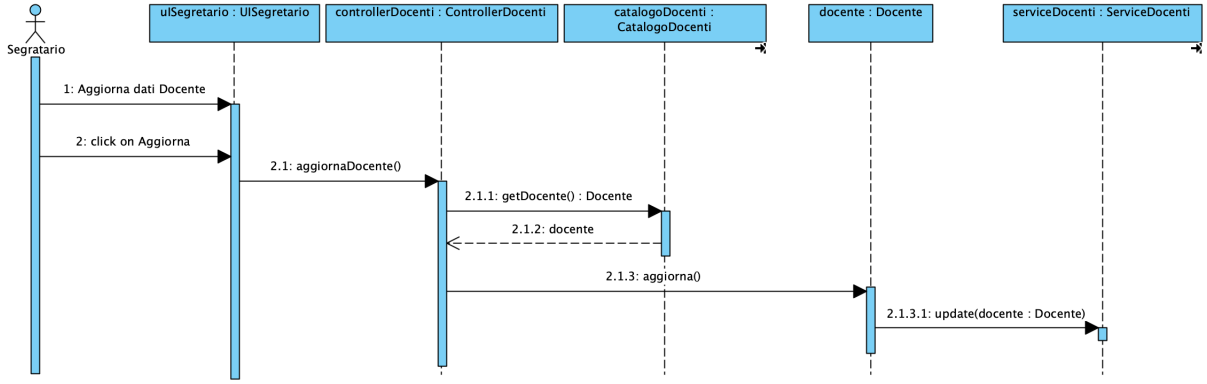

Figura 5.4: Sequence Diagram: Aggiorna Docente - RDD

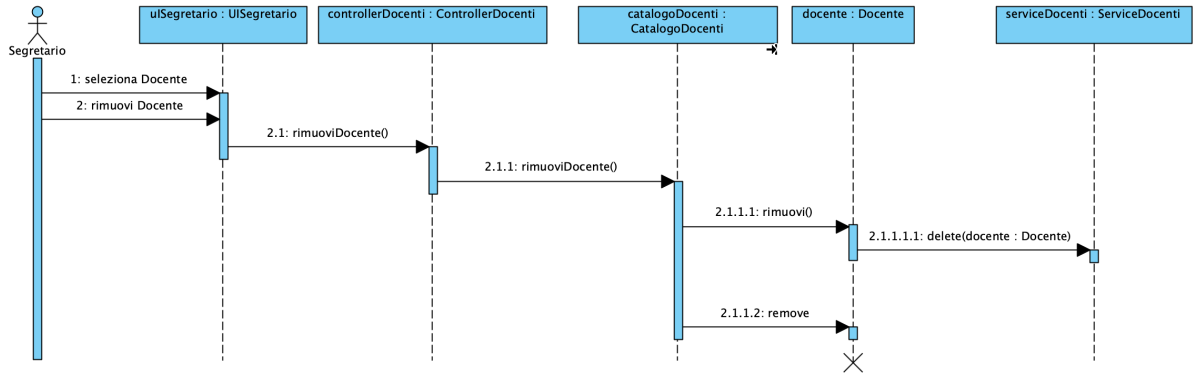

Figura 5.5: Sequence Diagram: Rimuovi Docente - RDD

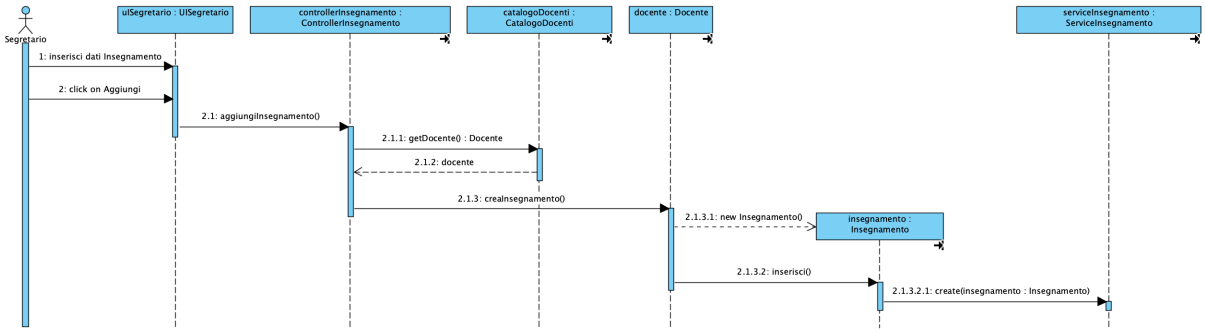

Figura 5.6: Sequence Diagram: Aggiungi Insegnamento - RDD

5.3 Scelte progettuali

Dopo una prima fase di progettazione pura, condotta seguendo l'approccio RDD, sono state compiute diverse scelte volte a colmare il gap esistente tra progettazione ed implementazione. Dovendo sviluppare un'applicazione web, gli aspetti da definire sono stati essenzialmente tre: frontend, backend e database.

5.3.1 Frontend

Per il frontend sono stati valutati i principali framework per lo sviluppo di *one-page application*. Tra questi si è scelto di utilizzare **Vaadin**, dal momento che permette di sviluppare un'interfaccia grafica web in Java, direttamente lato server.

5.3.2 Backend

È stata effettuata un'analisi relativa allo stato dell'arte dello sviluppo di applicazioni web. Tale analisi ha portato a scegliere per lo sviluppo del backend uno dei framework attualmente più usati in ambito enterprise, ovvero **Spring**.

5.3.3 Database

Si è scelto di utilizzare il database relazionale open source **MySQL**.

5.4 Frameworks utilizzati

5.4.1 Spring Framework

Spring Framework è un framework per lo sviluppo di moderne applicazioni enterprise *Java-based*, nato come alternativa leggera a Java EE. Fornisce un supporto infrastrutturale all'applicazione, sollevando gli sviluppatori da tale onere e permettendo loro di concentrarsi sulla logica di business.[5]

Un aspetto chiave del framework è sicuramente la sua **modularità**. Tra i principali moduli riportati in Figura 5.7, quelli utilizzati sono:

- **Core**: fornisce le funzionalità principali del framework, tra cui vale la pena menzionare la *Dependency Injection*, la quale permette di risolvere il problema delle dipendenze tra oggetti. Gli oggetti implementati rappresentano entità del mondo reale e in quanto tali non sono indipendenti ma dipendono da altri oggetti. In generale, per soddisfare le dipendenze tra gli oggetti è possibile seguire due approcci:
  - ciascun oggetto soddisfa le proprie dipendenze;
  - ciascun oggetto dichiara le proprie dipendenze e qualcun altro le soddisfa;

  Il secondo approccio è ciò che viene definito dependency injection. Spring Core offre quindi un *Dependency Injection Container* che si fa carico di creare e gestire gli oggetti e le loro dipendenze.
- **Web**: offre diverse funzionalità per lo sviluppo di applicazioni web.
  - **Spring Web MVC**: framework web costruito al di sopra della *Servlet API* offerta da Java. Fornisce un API di livello più alto, più semplice da usare. In generale, un *Servlet* è un oggetto Java, definito nel contesto di un *web server*, che riceve una richiesta e genera una risposta sulla base della richiesta ricevuta. Spring MVC è progettato secondo il pattern *Front Controller*, il quale prevede un servlet centrale, detto *Dispatcher Servlet*, che fornisce un unico punto di accesso all'applicazione. Si fa carico della ricezione di tutte le richieste, inoltrandole ai Servlet a cui sono indirizzate.

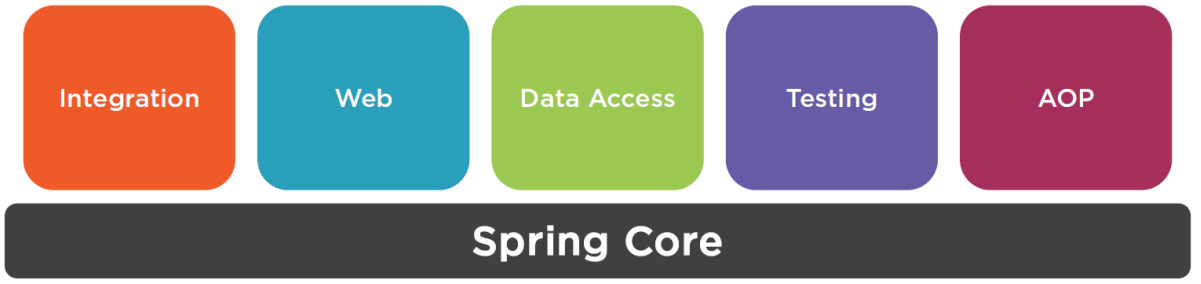

Figura 5.7: Moduli di Spring Framework

Per utilizzare le diverse funzionalità offerte da Spring Framework è sufficiente definire alcune configurazioni. A tal proposito è possibile seguire due approcci distinti: un primo approccio prevede di effettuare la configurazione tramite file XML, un secondo approccio si basa invece sul meccanismo delle annotazioni.

In conclusione, si riportano i punti di forza del framework:

- **Solidità**: è un framework sicuro e testato la cui solidità è garantita da 18 anni di sviluppo ed utilizzo; il primo rilascio risale infatti al 2002.

- **Completezza**: nato come alternativa a Java EE, negli ultimi anni Spring è cresciuto enormemente componendosi di progetti dedicati alla risoluzione di tutta una serie di problematiche tipiche del mondo enterprise.
- **Modularità**: la struttura modulare del framework, che ad oggi si compone di oltre venti sotto-progetti, ne costituisce un altro punto di forza.
- **Flessibilità**: lascia ampia libertà allo sviluppatore riguardo il percorso da seguire per arrivare al risultato desiderato, specificando pochi vincoli da rispettare.
- **Produttività**: da un lato mette a disposizione utilità per qualunque esigenza, dall'altro fornisce integrazioni con la maggior parte delle librerie e dei framework più diffusi in ambito enterprise.
- **Testabilità**: permettendo di eliminare dal codice dipendenze dirette tra le classi, consente una facile testabilità dei singoli componenti dell'applicazione.

Al di sopra di Spring Framework sono stati sviluppati diversi progetti, riportati in Figura 5.8, ognuno dei quali è orientato a fornire specifiche funzionalità. Tra i diversi progetti si riportano solo quelli utilizzati nello sviluppo dell'applicazione.

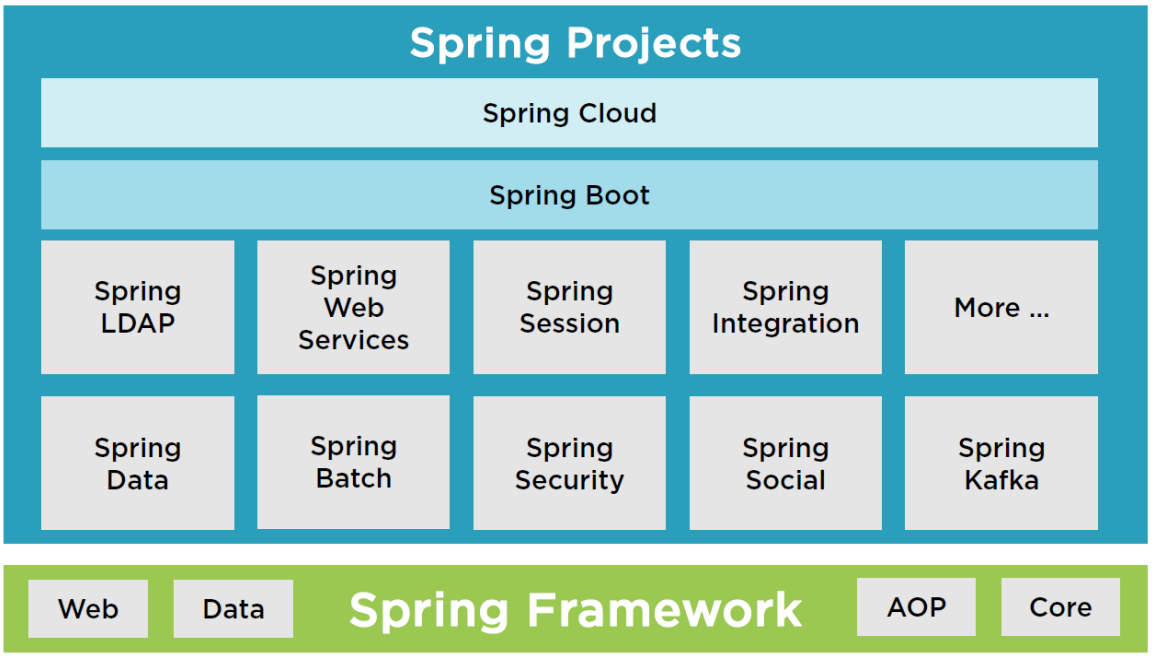

Figura 5.8: Architettura di Spring

5.4.2 Spring Data

Spring Data è un progetto nato per estendere il modulo Data Access di Spring Framework. Fornisce un modello di programmazione intuitivo in grado di facilitare l'utilizzo

delle tecnologie di accesso ai dati, supportando diversi tipi di database: relazionali, non relazionali e cloud-based.

Tra le diverse funzionalità offerte dal framework si hanno:

- risoluzione dell'*object-relational impedance mismatch*;
- derivazione dinamica delle query sulla base del nome dei metodi di classi repository;
- possibilità di integrare query esplicite sul database;
- implementazione legata ad API (e.g. JPA e JDBC) o database specifici (e.g. MySQL, Oracle e MongoDB);

Tra i diversi moduli di Spring Data si è scelto di utilizzare **Spring Data JPA**, il quale permette di implementare con il minimo sforzo il layer di accesso ai dati. È sufficiente definire l'interfaccia dei repository, specificando la firma dei metodi richiesti, e Spring fornisce automaticamente l'implementazione di tali metodi.

5.4.3 Spring Security

Spring Security è un progetto di Spring Framework che consente di gestire tutti gli aspetti relativi alla sicurezza di un'applicazione web Spring-based.

Le diverse funzionalità offerte dal framework includono:

- Supporto ad autenticazione e autorizzazione, entrambi aspetti chiave della sicurezza di un'applicazione.
  - L'**autenticazione** consiste nel verificare se l'utente è davvero chi dice di essere. Tipicamente viene effettuata verificandone credenziali come username e password.
  - L'**autorizzazione** consiste invece nel verificare a quali risorse l'utente autenticato può avere accesso. Tipicamente viene effettuata associando a ciascun utente un ruolo appropriato.
- Protezione per le tipiche vulnerabilità a cui è esposta un'applicazione web, tra cui *session fixation*, *clickjacking* e *cross site request forgery*.
- Integrazione con la Servlet API e Spring Web. Spring Security è infatti integrato con il Servlet Container, attraverso un Servlet Filter posto a monte del Dispatcher Servlet, il cui compito è quello di intercettare e filtrare le richieste dirette all'applicazione. Attraverso una vera e propria catena di filtri sono implementate tutte le funzionalità descritte nei punti precedenti.
- Cifratura delle password degli utenti.

Communication Diagram

Di seguito si riporta un *Communication Diagram* che descrive l'interazione tra i principali oggetti coinvolti nei processi di autenticazione e autorizzazione.

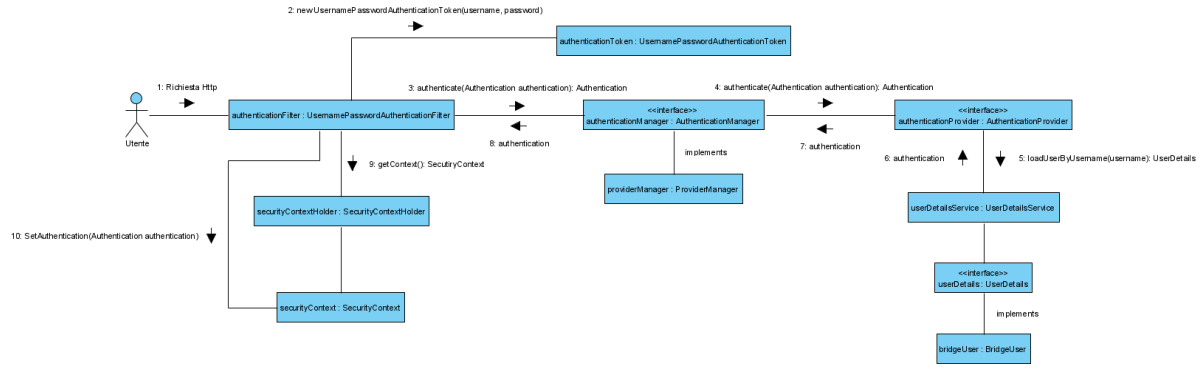

Figura 5.9: Communication Diagram: Autenticazione e Autorizzazione Utente

5.4.4 Spring Boot

Spring Boot è uno dei più recenti progetti Spring sviluppati, nato con l'obiettivo di semplificare lo sviluppo di applicazioni Spring-based, la cui configurazione può risultare complessa. Di fatto fornisce una visione categorica (*opinionated*) di Spring Framework ed è proprio questo ciò che riduce la complessità. In altre parole, Spring Boot definisce una configurazione di base per l'utilizzo di Spring Framework e di tutte le librerie di terze parti incluse in un progetto, semplificandone notevolemente l'avvio. In termini pratici, l'utilizzo di Spring Boot si riflette in un minor numero di annotazioni da specificare all'interno del codice.

Di seguito sono elencate alcune delle caratteristiche principali di Spring Boot:

- Configurazione automatica di Spring Framework e di librerie di terze parti, quando possibile.
- Nessuna necessità di file XML di configurazione.
- Possibilità di includere un web server/container embedded, come Tomcat o Jetty. Di conseguenza non è necessario il deployment di file WAR (*Web Application Archive*) all'interno di un container esterno all'applicazione.
- Offre diversi *starter dependencies*, ovvero configurazioni automatiche di librerie e dipendenze da poter includere nell'applicazione (e.g. spring-boot-starter-data-jpa, spring-boot-starter-web).

5.4.5 Vaadin

Vaadin Framework è un framework di sviluppo di applicazioni Web Java progettato per semplificare la creazione e la manutenzione di interfacce utente. Vaadin supporta due diversi modelli di programmazione: *server-side* e *client-side*. [6]

Il modello di programmazione **server-side** è il più potente e consente al programmatore di astrarre i dettagli implementativi di una tipica applicazione web e di programmare le interfacce utente in modo molto simile alla programmazione di un'applicazione desktop con i tradizionali toolkit Java, come AWT, Swing o SWT.

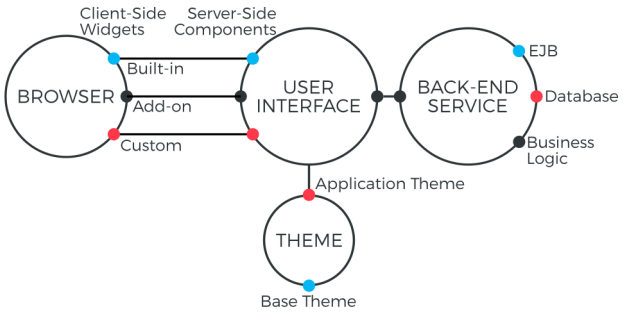

Figura 5.10: Architettura di Vaadin

La Figura 5.10 illustra l'architettura di base delle applicazioni web realizzate con Vaadin, costituita dal *server-side framework* e da un *client-side engine*.

L'engine viene eseguito nel browser come codice JavaScript, ed effettua il rendering dell'interfaccia utente. La logica dell'interfaccia utente di un'applicazione viene eseguita come servlet Java in un application server Java. Poiché il client-side engine viene eseguito come JavaScript nel browser, non sono necessari plug-in del browser per l'utilizzo delle applicazioni realizzate con Vaadin. Il framework si affida al supporto di *Google Web Toolkit* (GWT) per un'ampia gamma di browser, in modo che lo sviluppatore non debba preoccuparsi di tale aspetto.

Dal momento che HTML, JavaScript e altre tecnologie browser sono essenzialmente invisibili alla logica dell'applicazione, è possibile vedere il browser web solo come una piattaforma *thin client*. Un thin client renderizza l'interfaccia utente e comunica gli eventi che si verificano su di essa al server. La logica di controllo dell'interfaccia utente viene eseguita su un web server basato su Java, insieme alla logica di business.

Dietro il modello di sviluppo basato su server, Vaadin utilizza tecniche AJAX (*Asynchronous JavaScript and XML*) che consentono di creare *Rich Internet Applications* (RIA), applicazioni web reattive e interattive come quelle desktop.

Oltre allo sviluppo di applicazioni Java server-side, è possibile sviluppare seguendo l'approccio **client-side**, creando nuovi widget in Java e persino applicazioni lato client pure, che vengono eseguite esclusivamente nel browser. Il framework lato client include GWT, che fornisce un compilatore da Java a JavaScript che viene eseguito nel browser, nonché un framework di interfaccia utente completo. Con questo approccio, Vaadin è puro Java su entrambi i lati server e client.

5.5 Framework Oriented Design (FOD)

Per tenere traccia del gap esistente tra progettazione RDD e progettazione FOD è stata realizzata la **Matrice di Tracciabilità RDD-FOD** in Tabella 5.1. Dal momento che

risulta complesso ottenere un mapping uno ad uno fra le classi di progetto RDD riportate in Figura 5.2 e le classi di progetto FOD, si è tenuto traccia della corrispondenza fra classi di progetto RDD e package FOD. I package contengono infatti classi che realizzano il comportamento specificato nelle classi RDD.

	`ui.segretario`	`backend.services`	`backend.models`	`backend.repositories`
`UISegretario`	✘			
Classi Controller		✘		
Classi di Dominio			✘	
Classi Service				✘

Tabella 5.1: Matrice di tracciabilità RDD-FOD

5.5.1 Component Diagram

La scelta del framework di sviluppo per il frontend ha chiaramente influenzato anche la progettazione di client e server.

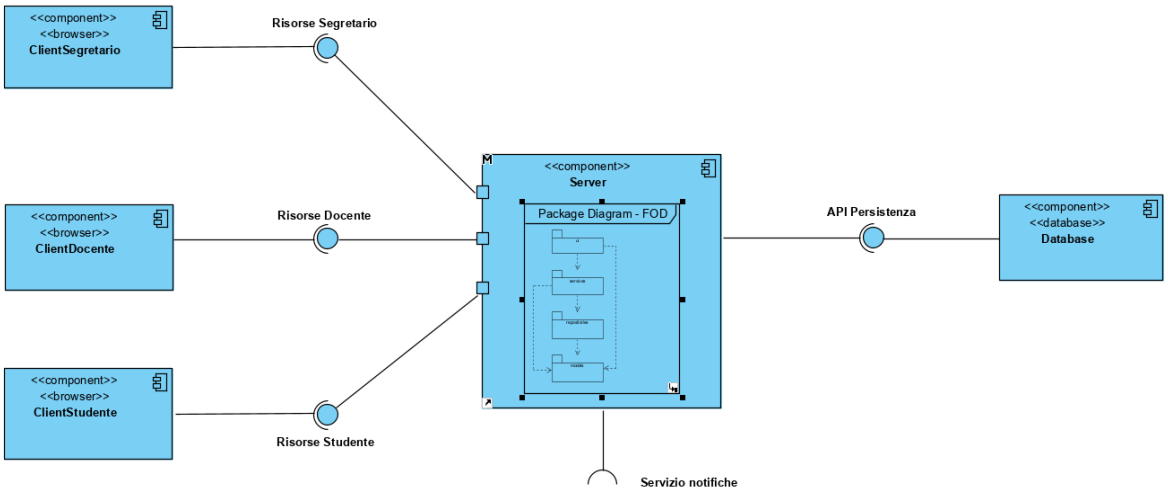

Figura 5.11: Component Diagram: Architettura del Sistema

Come evidenziato dal diagramma in Figura 5.11, il server espone tre insiemi distinti di risorse, ognuno dei quali può essere consumato da un tipo di utente diverso (segretario, docente e studente). Ciascuna risorsa è rappresentata da una URL e permette al client di eseguire determinate operazioni.

Di seguito si riportano nel dettaglio le risorse esposte dal server al client del segretario.

- *segretario/materie*: permette al segretario di gestire (aggiungere, aggiornare ed eliminare) le materie del sistema;
- *segretario/classe*: permette al segretario di gestire le classi del sistema e i relativi insegnamenti;
- *segretario/studenti*: permette al segretario di gestire gli studenti del sistema;

- *segretario/docenti*: permette al segretario di gestire i docenti del sistema;

Comunicazione Client-Server

Come specificato in precedenza, nella Sezione 5.4.5, Vaadin utilizza la tecnologia AJAX nella comunicazione tra client e server.

Il client non possiede alcuna logica applicativa, ma trasmette al server gli eventi che si verificano sulla UI. Il messaggio contiene quindi solo informazioni di basso livello sull'interazione dell'utente: tutta la logica applicativa è presente lato server.

L'interazione tra client e server avviene a mezzo di XHR (*XML HTTP Request*). Di conseguenza il client è in grado di effettuare solo richieste HTTP di tipo GET e POST. Il payload dei pacchetti inoltrati dal client contiene informazioni in formato JSON che permettono di realizzare una RPC (*Remote Procedure Call*). La trasmissione di tali informazioni richiede necessariamente operazioni di marshalling dei dati coinvolti.

Il server gestisce le richieste inoltrate dai client interpretando gli eventi verificatisi sulla UI e modificando lo stato degli oggetti del sistema. Il server, inoltre, invia i cambiamenti di stato degli elementi dell'interfaccia utente al client, mentre il client engine ne aggiorna la visualizzazione.

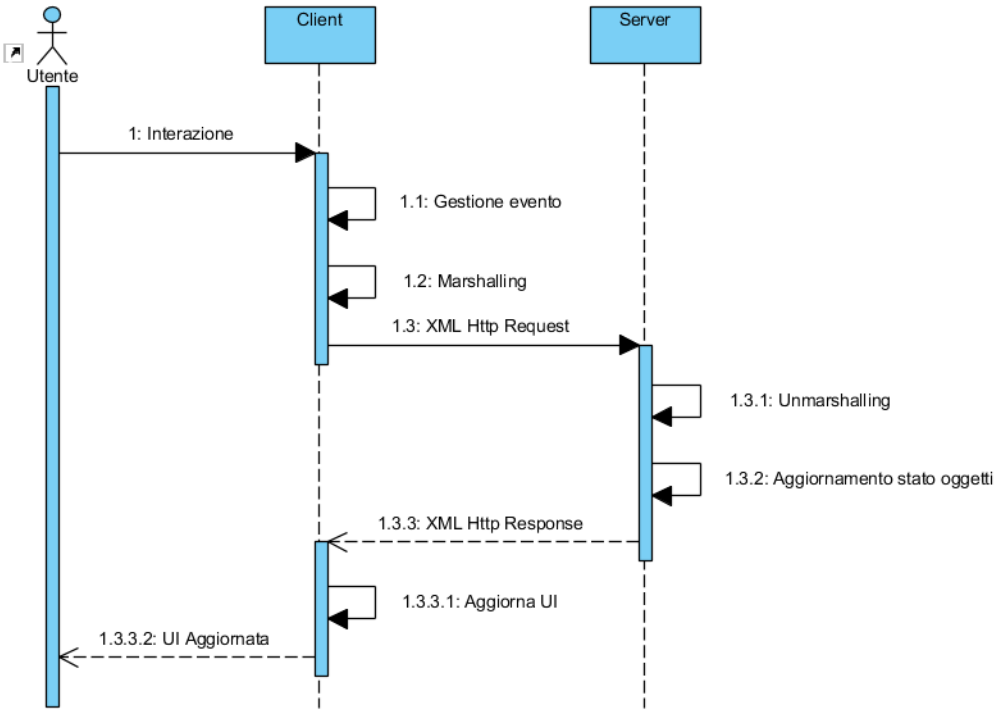

Figura 5.12: Sequence Diagram: Comunicazione Client-Server

5.5.2 Package Diagram

Per definire l'architettura interna del server è stato realizzato un *Package Diagram* in cui si evidenzia la struttura del componente.

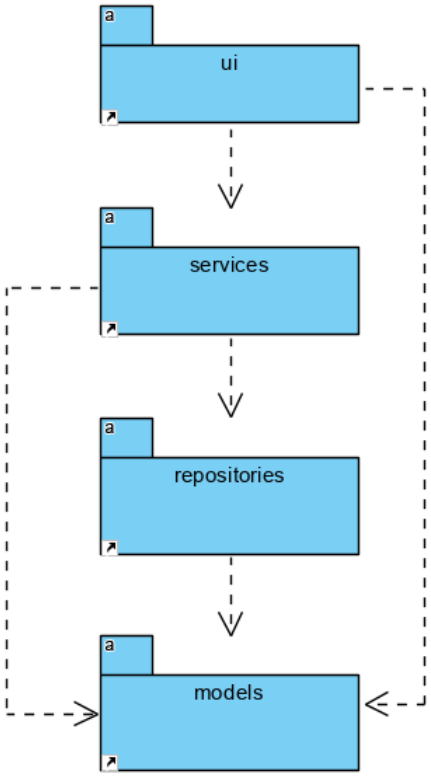

Figura 5.13: Package Diagram: Architettura Server

Come evidenziato dal diagramma in Figura 5.13, il server è organizzato in un'**architettura a livelli**:

- **UI layer**: contiene le classi che compongono l'interfaccia utente;
- **Service layer**: contiene le classi che implementano la logica di business dell'applicazione;
- **Repository layer**: contiene le interfacce che forniscono la logica di accesso al database;
- **Model layer**: contiene le classi di dominio del sistema;

Evidentemente l'architettura non risulta di tipo *strict-layered*, in quanto i package `ui` e `services` hanno una dipendenza diretta con il package `models`. Tuttavia, è rispettato il vincolo di un'architettura *layered* in quanto ciascun livello richiede i servizi esposti esclusivamente da livelli inferiori.

La dipendenza tra `ui` e `models` è dovuta al fatto che alcune classi dell'interfaccia utente richiedono le classi di dominio per effettuare meccanismi di binding e validazione dei dati.

La dipendenza tra `services` e `models` è invece legata al fatto che la realizzazione della logica di business richiede l'utilizzo di alcune classi di dominio.

5.5.3 Class Diagram

Ottenuta una buona conoscenza dei framework utilizzati è stato realizzato il **System Model**, un *Class Diagram* di dettaglio in cui sono riportate le classi del sistema implementate. Nella stesura di tale diagramma si è cercato di ridurre al minimo il gap tra progettazione ed implementazione.

Data la notevole dimensione del diagramma, per conservare la qualità dell'immagine e per garantirne la leggibilità, esso è stato allegato separatamente nella stessa cartella della documentazione. È possibile visualizzarlo utilizzando il seguente collegamento ipertestuale:

System Model

5.5.4 Design Pattern

Dove possibile durante la progettazione, sono stati applicati diversi design pattern, al fine di migliorare la qualità del software prodotto.

Façade

Il design pattern *Façade* permette, attraverso un'interfaccia unificata, l'accesso a sottosistemi che espongono interfacce complesse e diverse tra loro. In questo modo si rende il sottosistema più semplice da utilizzare.

Come riportato nel diagramma in Figura 5.14, è stato realizzato un `ServiceFacade` che espone un'interfaccia per la gestione (creazione, aggiornamento, eliminazione) di Materie, Classi, Insegnamenti, Docenti e Studenti. Nel dettaglio il façade espone i seguenti servizi:

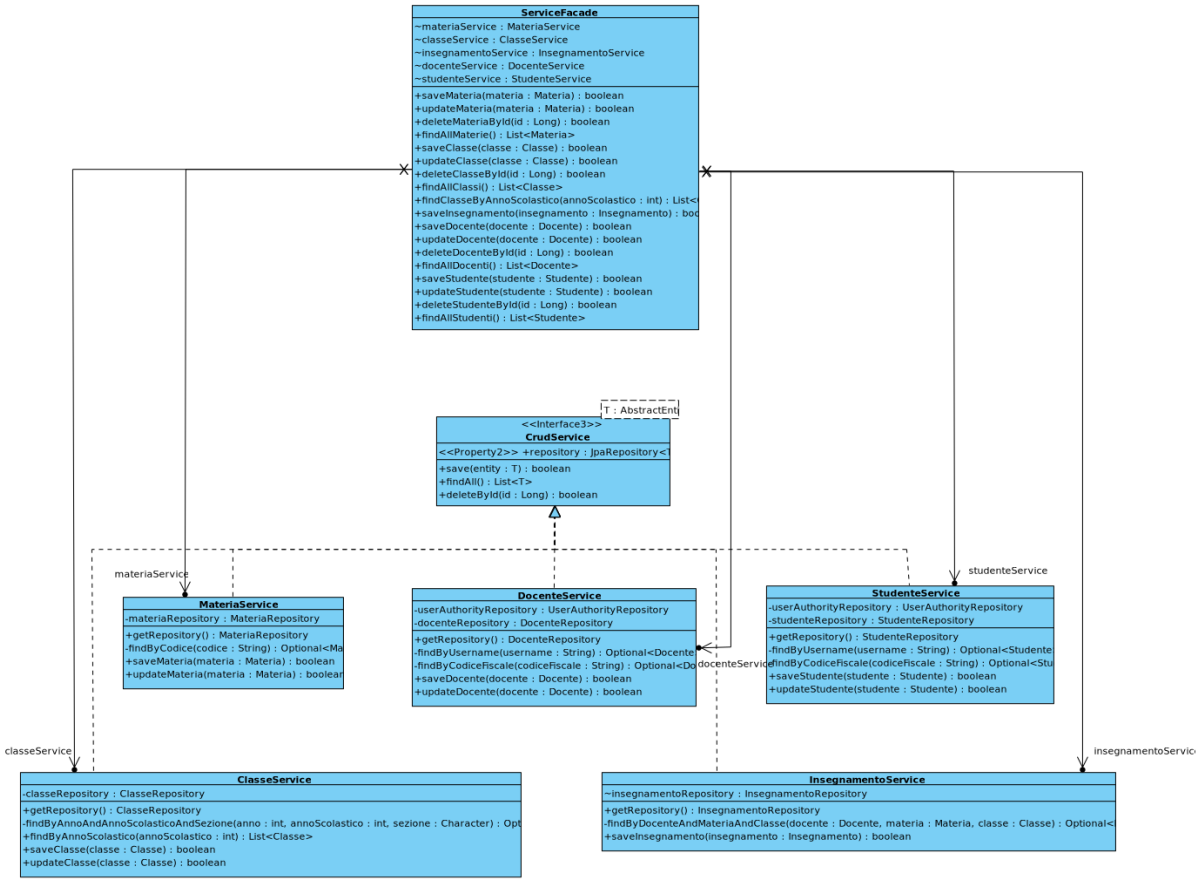

Figura 5.14: Class Diagram: Façade

Abstract Factory

L'*Abstract Factory* fornisce un'interfaccia per creare famiglie di oggetti connessi o dipendenti tra loro, in modo che non ci sia necessità da parte dei client di specificare i nomi delle classi concrete all'interno del proprio codice. In questo modo si permette che un sistema sia indipendente dall'implementazione degli oggetti concreti e che il client, attraverso l'interfaccia, utilizzi diverse famiglie di prodotti.

Si è scelto di utilizzare tale pattern per la realizzazione dell'interfaccia utente, al fine di disaccoppiare le classi view dai componenti che utilizzano.

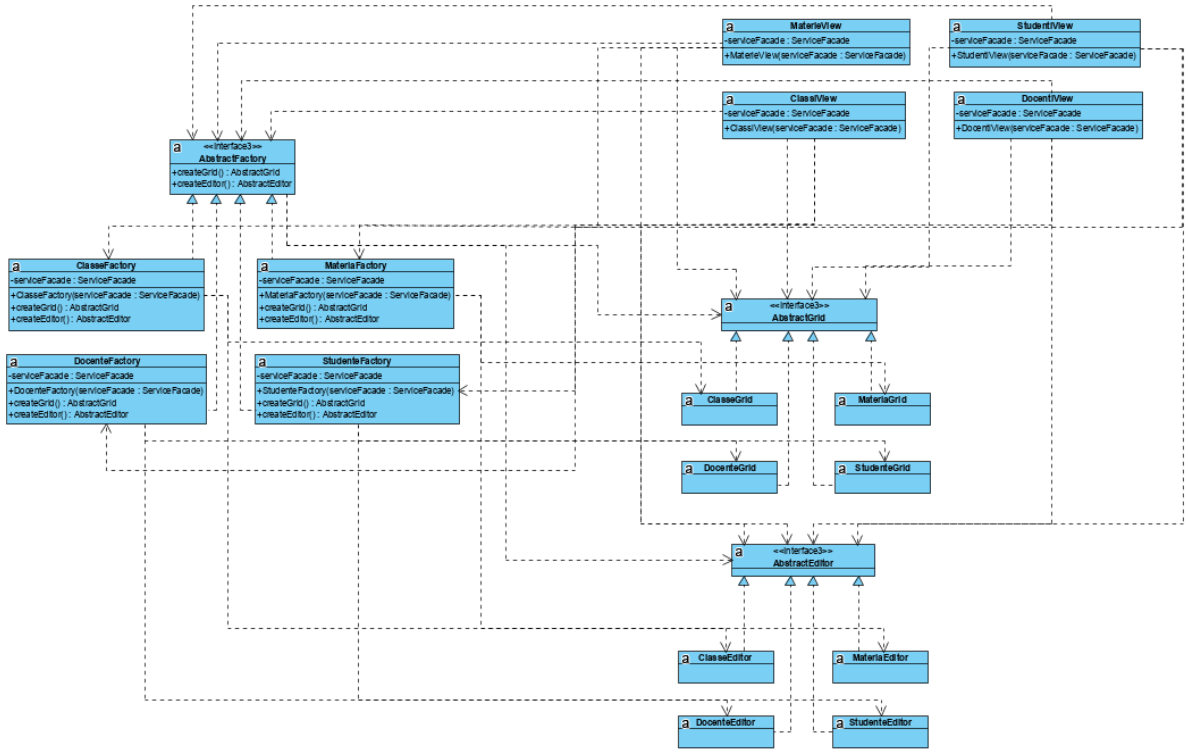

Figura 5.15: Class Diagram: Abstract Factory

Factory Method

Il *Factory Method* fornisce un'interfaccia per la creazione di un oggetto, lasciando alle sottoclassi la decisione del tipo di oggetto da istanziare. L'applicazione del pattern consente quindi di eliminare le dipendenze dai tipi concreti utilizzati.

Tale pattern è stato utilizzato nel contesto dell'*Abstract Factory* per fornire i metodi di creazione dei componenti concreti.

5.5.5 Sequence Diagram

Definito il *Class Diagram* di progetto sono stati prodotti diversi *Sequence Diagram*, al fine di mostrare le interazioni tra gli oggetti che permettono di realizzare le principali funzionalità del sistema.

Nel dettaglio, dato che molte interazioni sono comuni a tutte le funzionalità, si riportano, a titolo di esempio, solo i diagrammi relativi a creazione, aggiornamento ed eliminazione di un docente ed alla creazione di un insegnamento.

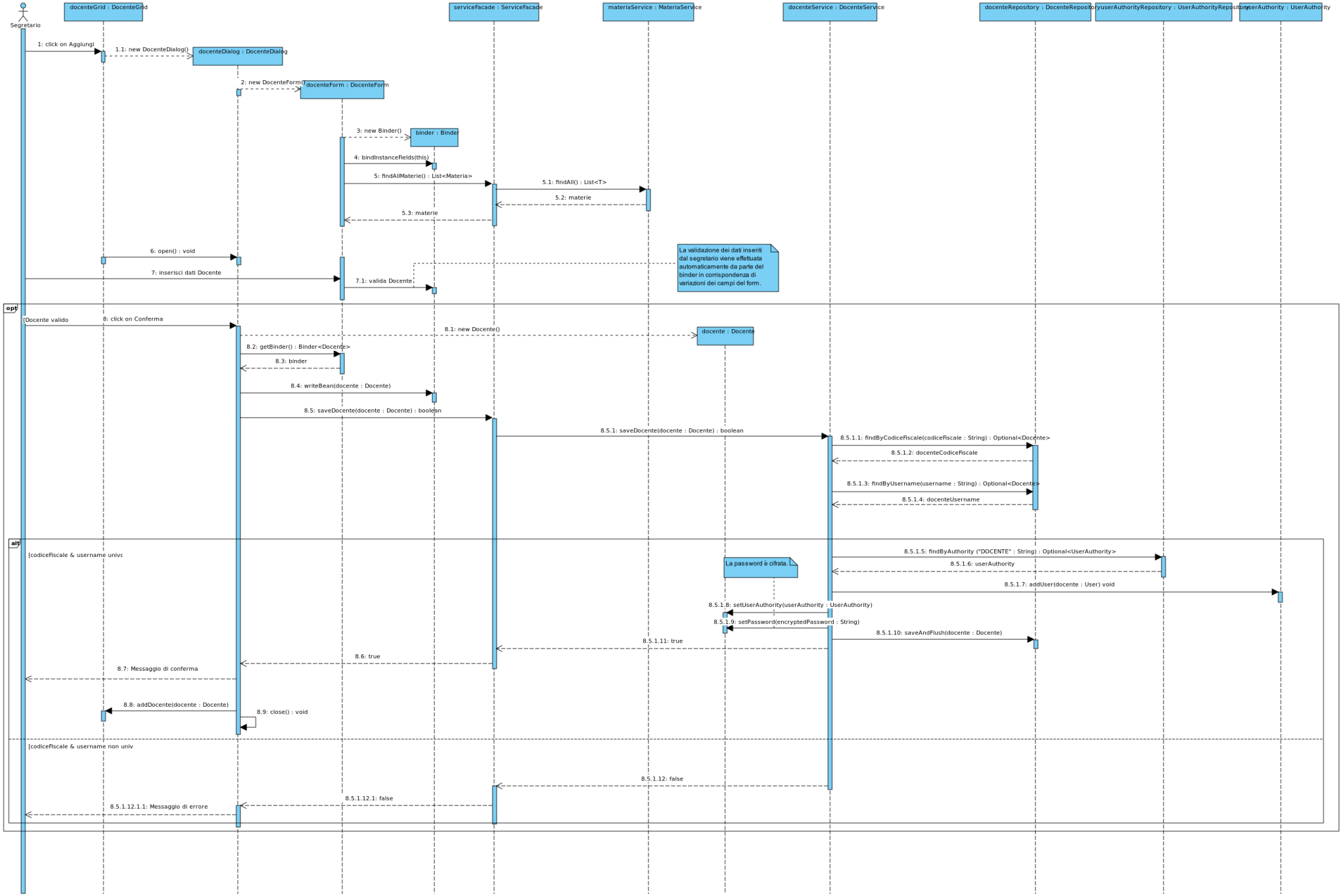

Figura 5.16: Sequence Diagram: Aggiungi Docente - FOD

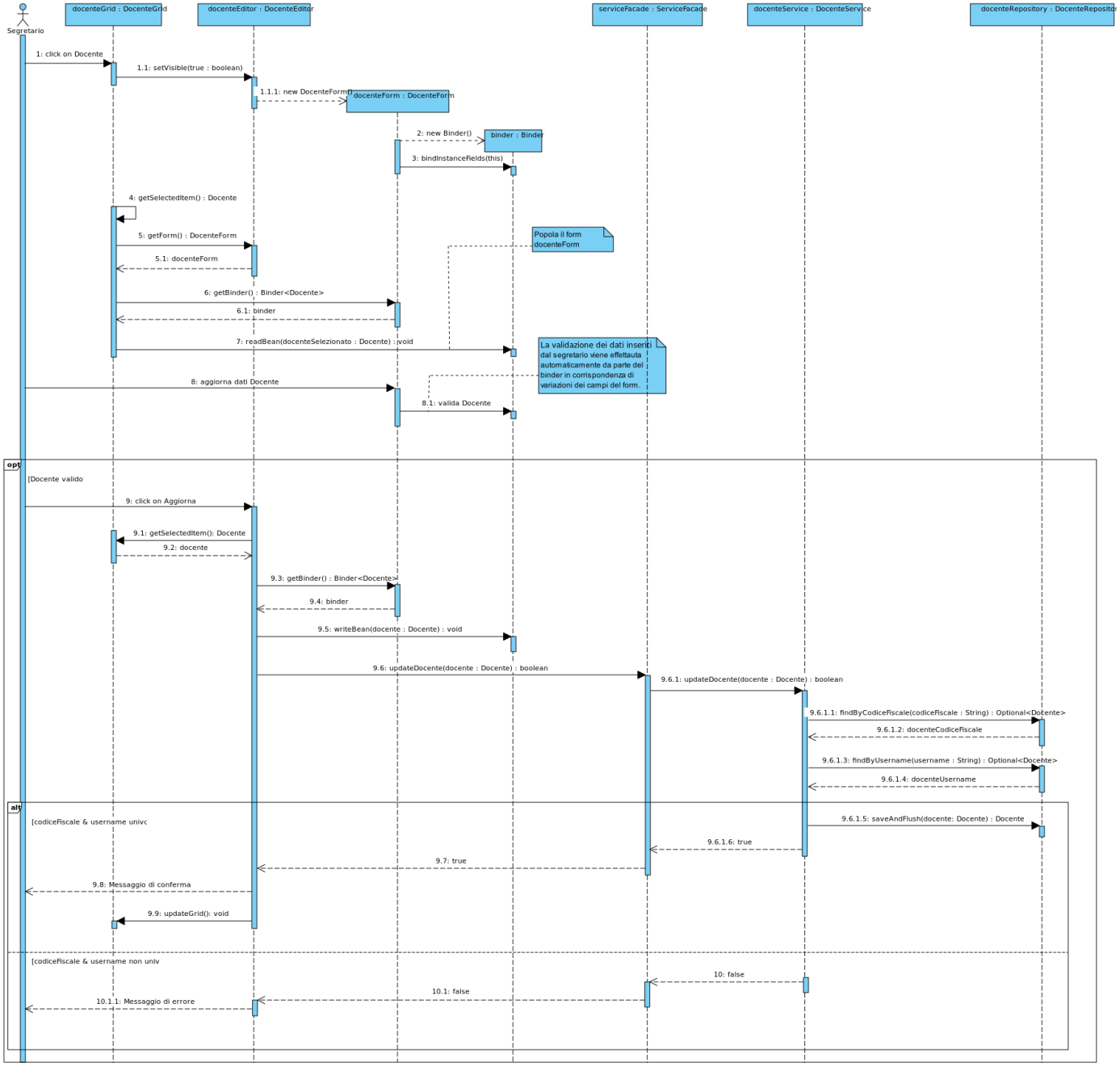

Figura 5.17: Sequence Diagram: Aggiorna Docente - FOD

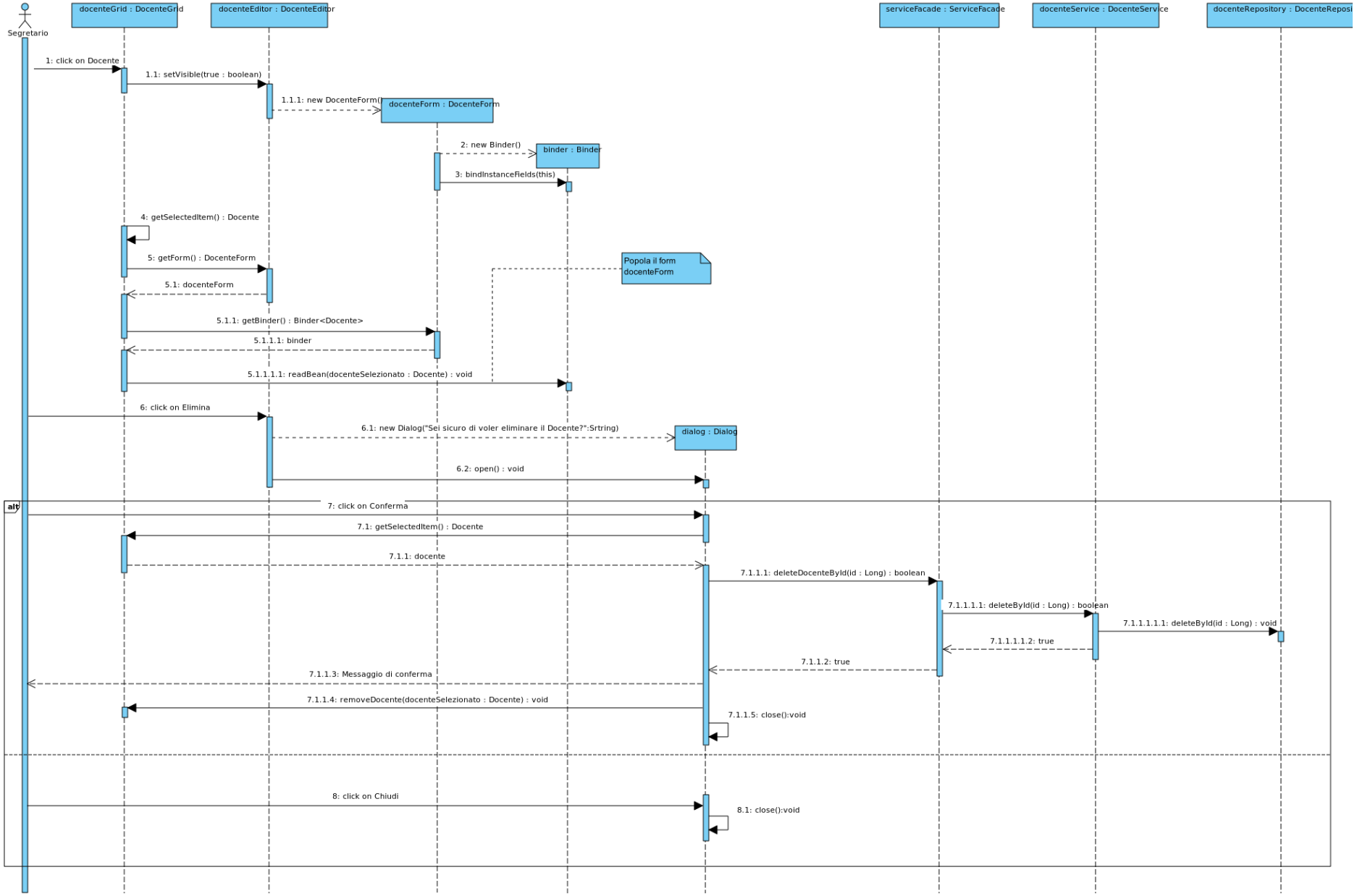

Figura 5.18: Sequence Diagram: Rimuovi Docente - FOD

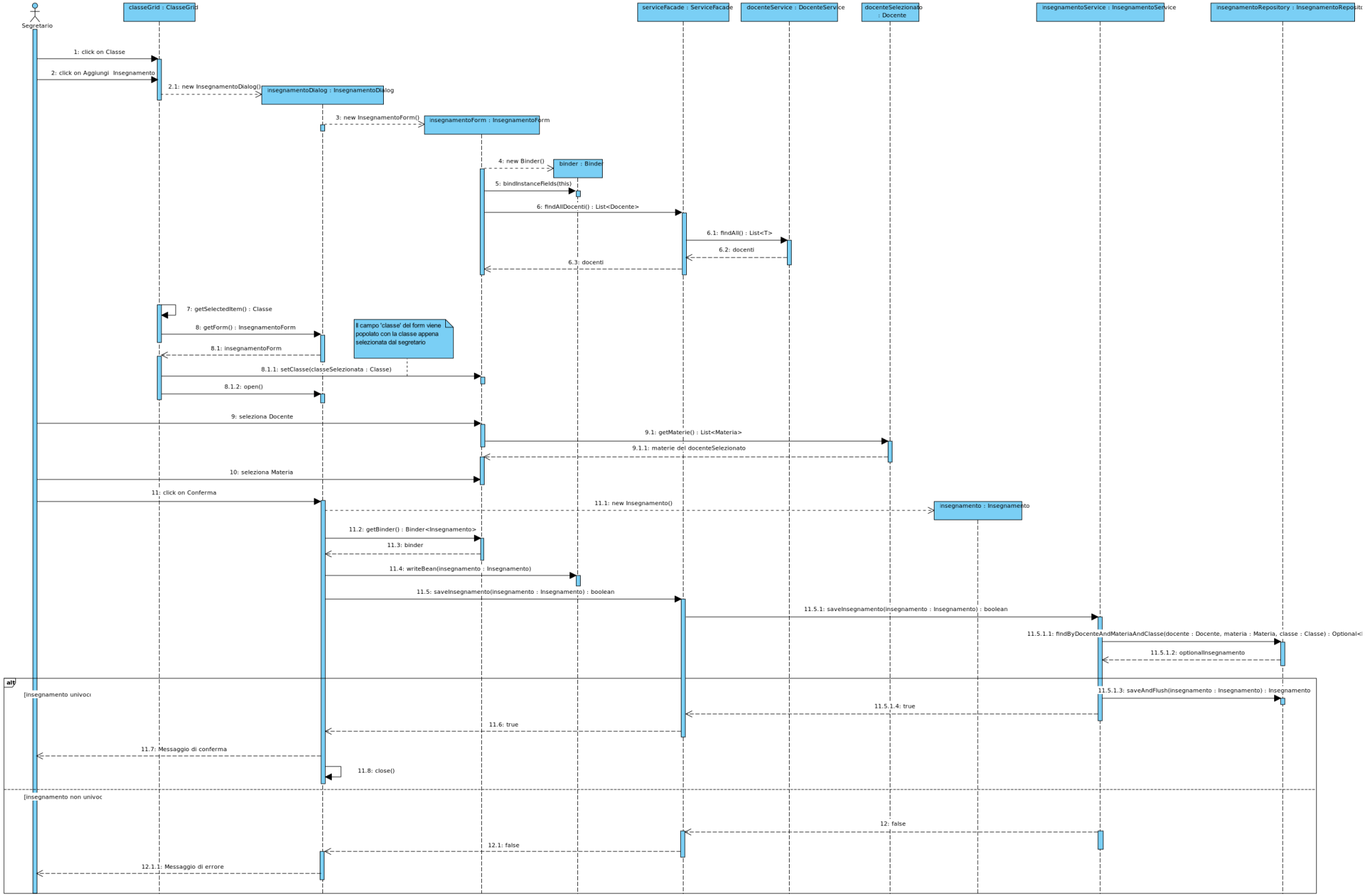

Figura 5.19: Sequence Diagram: Aggiungi Insegnamento - FOD

Infine, si propone un diagramma volto a descrivere i meccanismi coinvolti nella creazione delle principali classi dell'interfaccia utente, ovvero griglia ed editor, e del loro popolamento. Ancora una volta il diagramma realizzato è relativo ai docenti, ma può essere generalizzato per le altre entità del sistema.

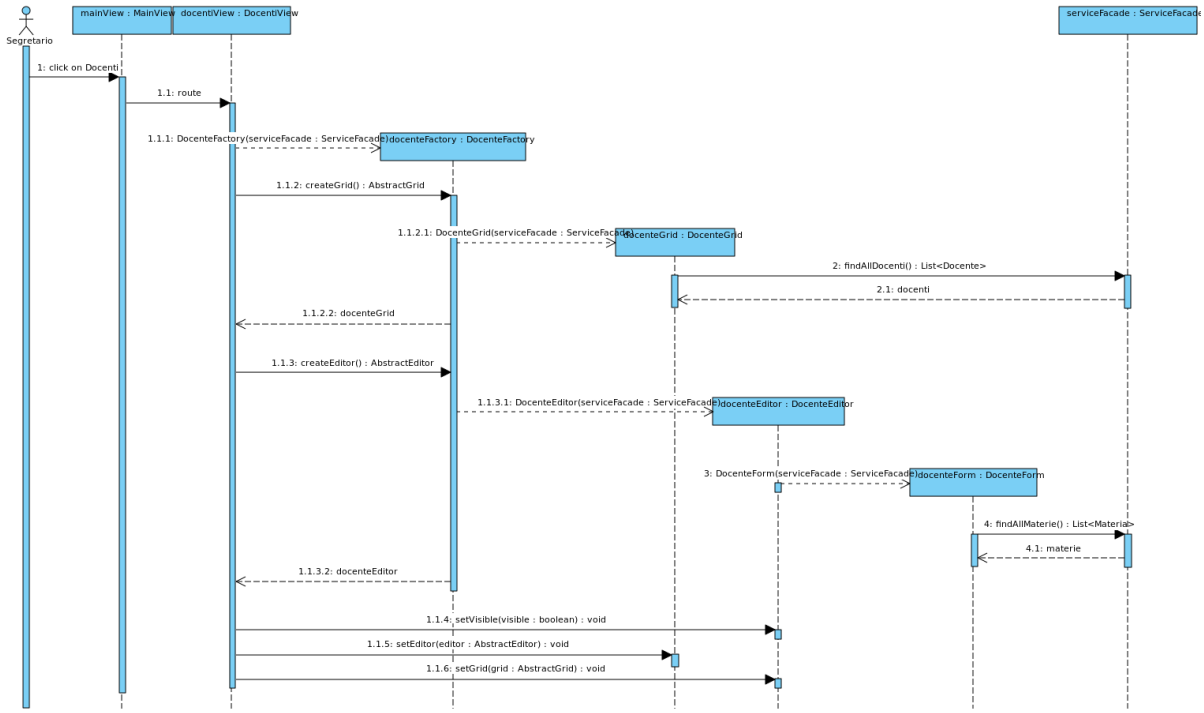

Figura 5.20: Sequence Diagram: Creazione griglia ed editor

5.5.6 Statechart Diagram

Al fine di chiarire ulteriormente il processo di validazione di un'entità del sistema si è scelto di realizzare uno *Statechart Diagram*. Sebbene il diagramma riportato faccia riferimento allo stato di un docente, esso può essere generalizzato per qualsiasi entità del sistema (studente, classe, materia o insegnamento).

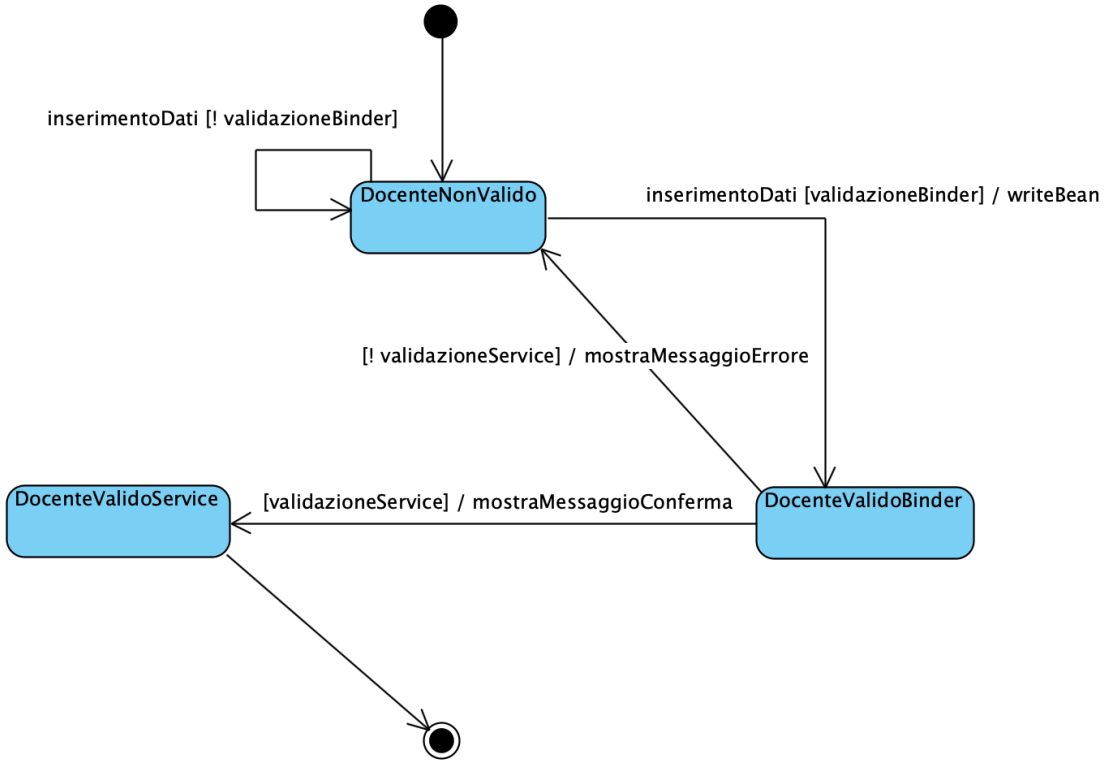

Figura 5.21: Statechart Diagram: Validazione Docente

5.5.7 Activity Diagram

Si è scelto di realizzare un *Activity Diagram* per rappresentare il processo di autenticazione di un utente.

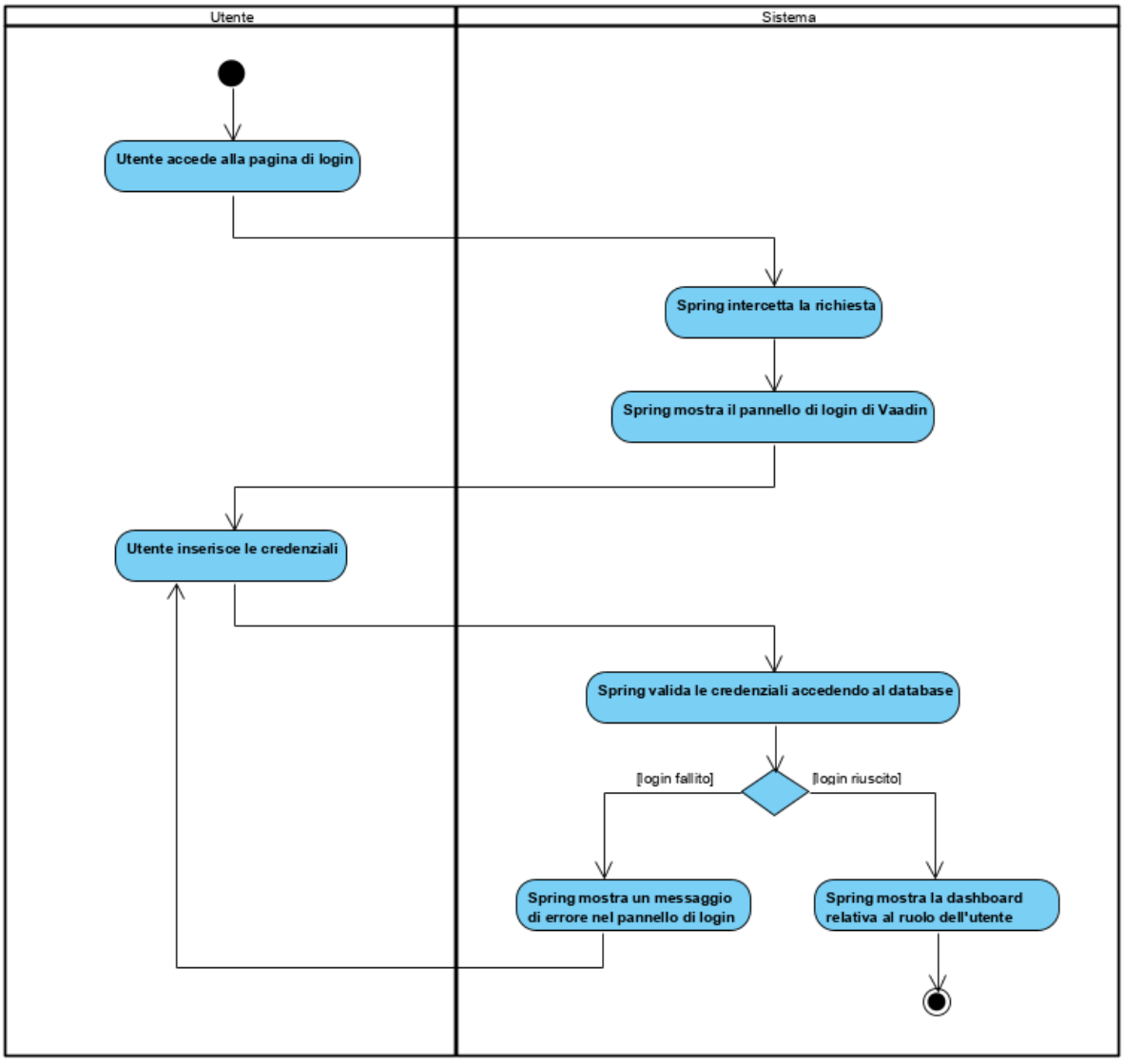

Figura 5.22: Activity Diagram: Autenticazione Utente

Capitolo 6

Implementazione

In questa sezione sono riportati tutti gli aspetti relativi all'implementazione del sistema.

6.1 Integrated Development Environment (IDE)

Per supportare l'intera fase di implementazione si è fatto uso dell'ambiente di sviluppo **IntelliJ**, realizzato da JetBrains. Tale IDE offre tra le sue funzionalità:

- *code completion* e *code navigation*;
- supporto al *testing* e al *debugging*;
- integrazione con Spring e Git;
- supporto al *building automation* tramite Maven;

6.2 Building

Per realizzare il building del progetto è stato utilizzato **Maven**, uno strumento di building automation sviluppato da Apache, usato principalmente per progetti Java.

Maven permette di gestire non solo aspetti relativi al building del progetto, ma anche alle sue dipendenze. Per utilizzarlo è necessario scrivere un file XML, chiamato tipicamente POM (*Project Object Model*) in cui è descritto il progetto, le sue dipendenze con moduli esterni ed altre indicazioni relative al processo di building. Sulla base di quanto riportato nel file POM, Maven è in grado di scaricare automaticamente le librerie necessarie, prelevandole da appositi repository Maven, memorizzandoli in una cache locale.

Di seguito si riportano le dipendenze specificate nel file `pom.xml` relative alle librerie dei framework utilizzati, descritti nel Paragrafo 5.4.

Spring Boot

pom.xml

```
...
<dependencies>
    ...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    ...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    ...
</dependencies>
...
```

Project Lombok

Project Lombok è una libreria Java utile per ridurre la quantità di codice *boilerplate* in un progetto, ovvero codice ripetuto molte volte. Lombok fornisce i suoi servizi tramite annotazioni che possono essere aggiunte alla classe Java per la quale sono desiderati metodi comuni, come getter, setter, costruttori, etc. La maggior parte delle annotazioni sono autodescrittive nei loro nomi, ad esempio: `@Getter` , `@Setter`, `@EqualsAndHashCode`, `@ToString` e `@NoArgsConstructor`.

pom.xml

```
...
<dependencies>
    ...
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    ...
</dependencies>
...
```

Spring Data

pom.xml

```
...
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    ...
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
    ...
</dependencies>
...
```

Spring Security

pom.xml

```
...
<dependencies>
    ...
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
    </dependency>
    ...
</dependencies>
...
```

Vaadin

pom.xml

```
...
<dependencies>
    ...
    <dependency>
        <groupId>com.vaadin</groupId>
        <artifactId>vaadin-spring-boot-starter</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    ...
    <dependency>
        <groupId>org.vaadin.gatanaso</groupId>
        <artifactId>multiselect-combo-box-flow</artifactId>
        <version>LATEST</version> <!-- use appropriate version -->
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.vaadin</groupId>
            <artifactId>vaadin-bom</artifactId>
            <version>\${vaadin.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<repositories>
    <repository>
        <id>vaadin-addons</id>
        <url>http://maven.vaadin.com/vaadin-addons</url>
    </repository>
</repositories>
...
```

6.3 Server-side

Nella sezione corrente si riportano i principali aspetti implementativi relativi al lato server.

Per evidenziare le dipendenze tra i diversi package del sistema, realizzati ed importati, è stato stilato un *Package Diagram* di dettaglio.

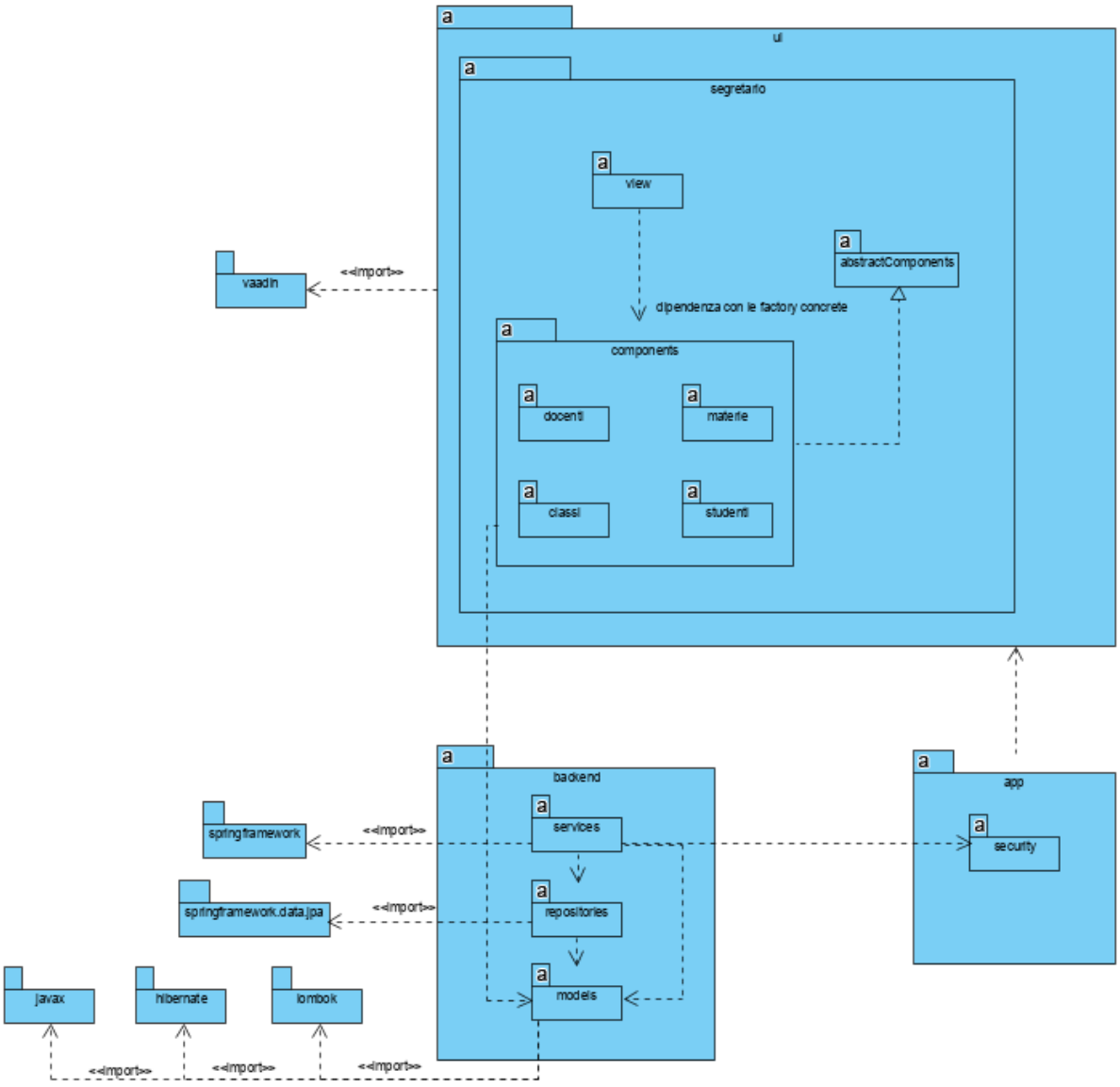

Figura 6.1: Package Diagram: Architettura Server

6.3.1 Frontend

Per la realizzazione della UI dell'applicazione si è scelto di utilizzare il modello di programmazione server-side previsto da Vaadin. I componenti fondamentali del framework utilizzati nel progetto sono:

- **Componenti/widget dell'interfaccia utente**: l'interfaccia utente di un'applicazione Vaadin è costituita da componenti che vengono creati e disposti dall'applicazione. Ogni componente lato server ha una controparte lato client, un widget, mediante il quale viene visualizzato nel browser e con il quale l'utente interagisce.
- **Client-side engine**: il motore lato client di Vaadin gestisce il rendering dell'interfaccia utente nel browser web utilizzando vari widget lato client.

- **Data binding**: oltre al modello di interfaccia utente, Vaadin fornisce un'API per associare i dati presentati nei componenti di interfaccia, come *text field*, *check box* e *selection component*, con oggetti delle classi di business. Utilizzando il data binding, i componenti dell'interfaccia utente possono aggiornare direttamente i dati dell'applicazione, spesso senza la necessità di alcun codice di controllo.

6.3.2 Backend

Per lo sviluppo del Backend sono state utilizzate le tecnologie fornite da Spring, in particolare service e repository.

Le classi service sono particolari componenti offerti dal framework, definite attraverso l'annotation `@Service`. In quanto componenti, Spring gestisce il ciclo di vita degli oggetti di tali classi. Tuttavia, le classi marcate con tale annotazione si distinguono dai tradizionali `@Component` in quanto gestiscono la logica di business del sistema.

Le classi repository sono anch'esse dei particolari component, annotate con `@Repository`, il cui compito è quello di gestire la logica di accesso al database. Nel dettaglio, esse forniscono operazioni CRUD per tutte le entità del database stesso.

Entrambe le suddette annotazioni permettono di identificare un *bean*, ovvero un oggetto gestito interamente dal container Spring. Di default, tali bean sono definiti con lo scope *singleton*: ciò significa che il container crea una sola istanza di quel bean e tutte le richieste indirizzate all'id ad esso associato verranno inoltrate a tale istanza.

In aggiunta, per definire le varie entità del database, è sufficiente annotare con `@Entity` tutte le classi di cui si voglia memorizzare le istanze in maniera persistente.

6.4 Database

Per poter accedere al database MySQL attraverso le interfacce fornite da Spring Data JPA, sono state specificate in un apposito file di configurazione `application.properties` le seguenti linee di codice.

application.properties

```
...
spring.datasource.url=jdbc:mysql://localhost/registro?useSSL=false
                        &serverTimezone=UTC&useLegacyDatetimeCode=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.main.lazy-initialization=true
```

6.4.1 Diagramma Relazionale

Di seguito si riporta il *Diagramma Relazionale* realizzato per descrivere la struttura del database.

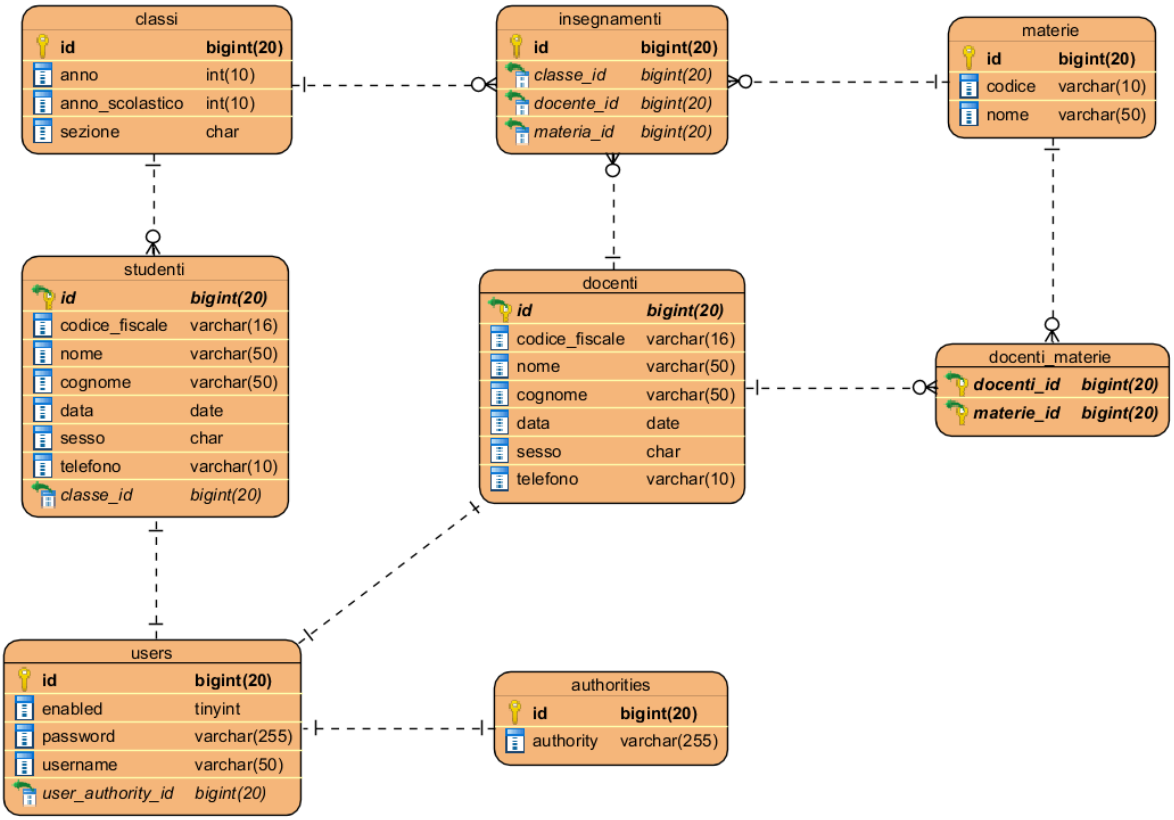

Figura 6.2: Diagramma Relazionale

6.5 Deployment

6.5.1 AWS Elastic Beanstalk

AWS Elastic Beanstalk è un servizio di distribuzione di applicazioni e servizi Web sviluppati in Java, .NET, Node.js, Python etc. su server comuni come Apache, Nginx, Passenger e IIS. Caricando il proprio codice, Elastic Beanstalk gestisce automaticamente il deployment, dal provisioning di capacità e autoscaling al monitoraggio della salute dell'applicazione. Al contempo, l'utente mantiene il completo controllo sulle risorse AWS su cui si basa la sua applicazione e può accedere in qualsiasi momento alle risorse implicate.

Panoramica di funzionamento

Per usare Elastic Beanstalk è necessario creare un'applicazione, caricare una versione di essa sotto forma di un bundle di origine, ad esempio un file in formato *.jar* o *.war* Java, e fornire alcune informazioni di configurazione. Elastic Beanstalk avvia automaticamente

un ambiente per poi creare e configurare le risorse AWS necessarie per eseguire il codice. Una volta avviato l'environment, esso può essere gestito e aggiornato con il caricamento di nuove versioni dell'applicazione. Lo schema seguente illustra il flusso di lavoro di Elastic Beanstalk.

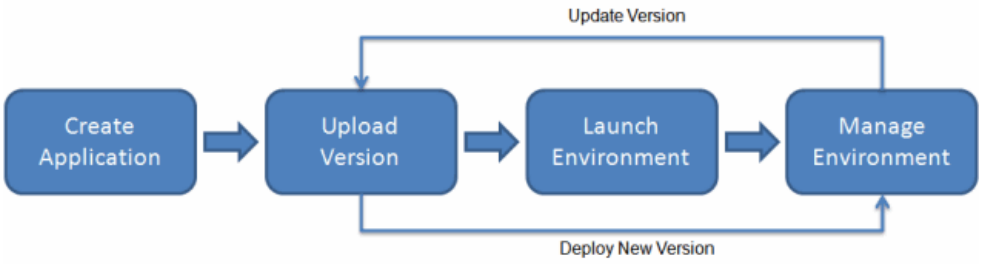

Figura 6.3: Funzionamento di AWS Beanstalk

Dopo aver creato e distribuito l'applicazione, le informazioni relative ad essa, inclusi parametri, eventi e stato dell'ambiente, saranno disponibili tramite la console Elastic Beanstalk, le API o le interfacce a riga di comando.

6.5.2 Distribuzione dell'applicazione

Di seguito sono riportati i passi fondamentali che sono stati eseguiti per il deployment dell'applicazione in rete [7].

1. **Preparazione alla produzione in IntelliJ**

   Prima di poter distribuire l'applicazione, è necessario renderla pronta alla produzione. Quando si crea una versione di produzione di un'app Vaadin, si verifica quanto segue: tutte le risorse front-end vengono raggruppate e minimizzate per accelerare il tempo di caricamento dell'app. Vaadin viene eseguito in modalità di produzione per nascondere il debug e altre informazioni dal browser. Nel dettaglio, per creare l'app con la Vaadin Production Mode abilitata è sufficiente aggiungere la seguente linea di codice nel file di configurazione `application.properties` del progetto in IntelliJ:

   application.properties

   ```
   ...
   vaadin.productionMode = true
   ...
   ```

   In secondo luogo, è necessario dotare il progetto di un file aggiuntivo di configurazione, denominato `application-prod.properties`. In questo modo, quando la modalità di produzione risulta abilitata, viene utilizzata un'apposita configurazione del database e dei porti di produzione.

application-prod.properties

```
server.port=5000
spring.datasource.url=jdbc:mysql://\${RDS_HOSTNAME}:\${RDS_PORT}/\${RDS_DB_NAME}
spring.datasource.username=\${RDS_USERNAME}
spring.datasource.password=\${RDS_PASSWORD}
spring.jpa.hibernate.ddl-auto=create
```

Da tali righe di codice ne consegue che:

- Elastic Beanstalk mappa la porta interna 5000 sulla porta esterna 80 per esporre l'applicazione in rete.
- Sarà Elastic Beanstalk a fornire le informazioni sul database alle variabili di ambiente, in modo tale che non sia necessario archiviarle in chiaro nel file delle proprietà.

2. **Creazione di una build di produzione dell'app Vaadin**

   Usando il Production Profile di Maven si può creare una build ottimizzata in formato .jar pronta per la produzione.

   ```
   mvn clean package -Pproduction -DskipTests
   ```

3. **Configurazione di Amazon Elastic Beanstalk**

   Sono a questo punto riportati gli step fondamentali per la configurazione dell'ambiente di deployment in Amazon Elastic Beastalk.

   (a) Creazione di un account AWS;

   (b) Creazione di una nuova applicazione nella console di Amazon Elastic Beanstalk, impostando in questo modo i vari campi richiesti:
   - Nome applicazione: RegistroElettronico
   - Livello ambiente: ambiente Web Server
   - Piattaforma selezionata: Amazon Corretto 11

   (c) Aggiunta di una proprietà all'ambiente in modo tale che le impostazioni per la produzione siano correttamente caricate:

Name	**Value**
SPRING_PROFILES_ACTIVE	prod

   (d) Nella sezione Database si è impostato MySQL come engine e sono stati impostati degli opportuni valori di username e password per accedere ad esso.

A questo punto sono stati impostati opportunamente tutti i requisiti necessari alla corretta inizializzazione dell'applicazione, potendo procedere di conseguenza alla creazione dell'ambiente. Una volta ultimata la fase di creazione, Elastic Beanstalk avrà creato le seguenti risorse AWS:

- **Istanza EC2 (Elastic Compute Cloud)**: una macchina virtuale di Amazon EC2 configurata per eseguire applicazioni Web sulla piattaforma selezionata. La maggior parte delle piattaforme utilizza Apache o Nginx come proxy inverso che elabora il traffico su cui viene eseguita l'applicazione web, inoltra le richieste all'app, fornisce asset statici e genera log degli accessi e di errore. Nel nostro caso si è fatto uso di Nginx come server proxy sulla piattaforma Tomcat.
- **Istanze del gruppo di sicurezza**: un gruppo di sicurezza di Amazon EC2 configurato per abilitare il traffico in entrata sulla porta 80. Questa risorsa consente al traffico HTTP proveniente dal sistema di bilanciamento del carico di raggiungere l'istanza EC2 in esecuzione sull'app web. Per impostazione predefinita, il traffico non è consentito su altre porte.
- **Bucket Amazon S3 (Simple Storage Service)**: una destinazione di storage per i codici di origine, i log, i file di configurazione e altri elementi creati durante l'utilizzo di Elastic Beanstalk.
- **Allarmi Amazon CloudWatch**: due allarmi CloudWatch che monitorano il carico sulle istanze dell'ambiente in uso e che vengono attivati se esso è troppo elevato o insufficiente. Quando viene attivato un allarme, il gruppo Auto Scaling aumenta o diminuisce di conseguenza.
- **Stack AWS CloudFormation**: Elastic Beanstalk utilizza AWS CloudFormation per avviare le risorse nell'ambiente in uso e propagare le modifiche di configurazione. Le risorse sono definite in un modello, visualizzabile nella console AWS CloudFormation.
- **Nome dominio**: un nome di dominio che rimanda alla propria applicazione Web in forma di *sottodominio.regione.elasticbeanstalk.com*.
- **Amazon RDS SQL Database**: database relazionale creato all'atto della costruzione dell'environment, il cui ciclo di vita risulta essere strettamente legato a quello dell'ambiente stesso.

4. **Distribuzione dell'applicazione**

   Nella console Dashboard è possibile caricare e distribuire in rete l'applicativo precedentemente esportato nel formato *.jar*.

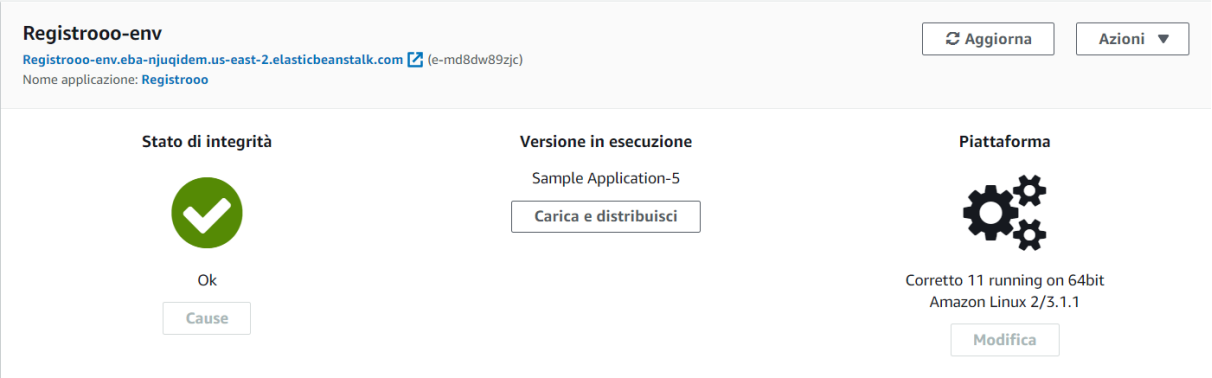

Figura 6.4: Dashboard di Elastic Beanstalk

Completata la procedura, lo stato di integrità dell'ambiente dovrebbe indicare *Ok* (segno di spunta verde). L'applicazione è in esecuzione e quindi accessibile sul web tramite il collegamento nella parte superiore della dashboard. Si riporta di seguito l'URL dell'applicazione realizzata.

`http://registrooo-env.eba-njuqidem.us-east-2.elasticbeanstalk.com/`

6.5.3 Deployment Diagram

Per illustrare come l'applicazione è stata distribuita sui servizi AWS precedentemente descritti è stato realizzato il seguente *Deployment Diagram*.

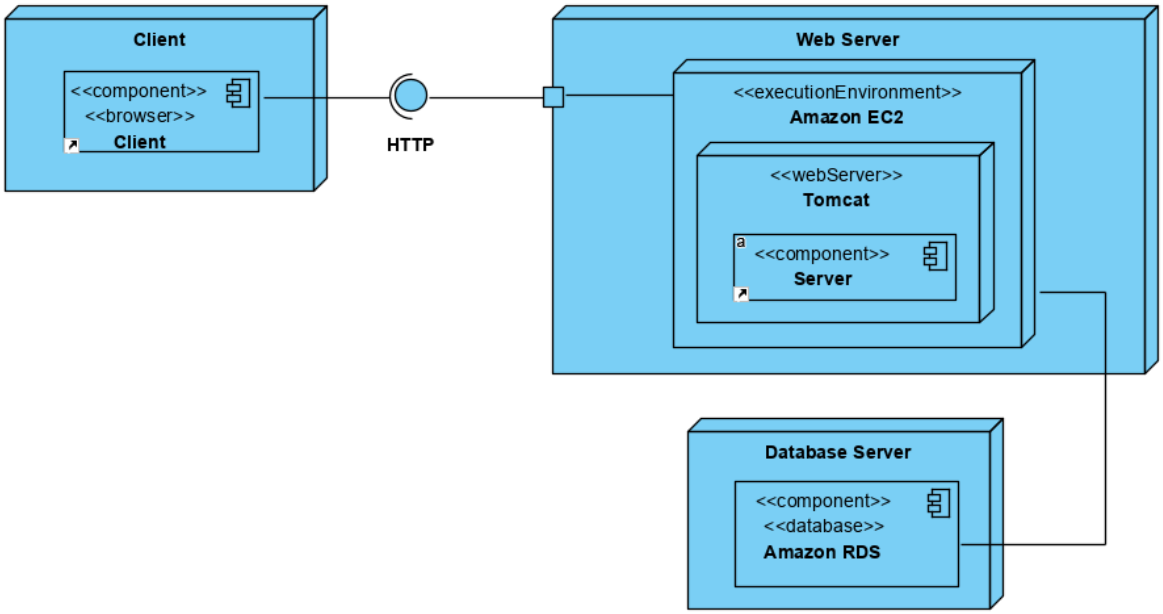

Figura 6.5: Deployment Diagram

Bibliografia

[1] *Scrum Guide*. URL: https://www.scrumguides.org/scrum-guide.html.

[2] *Software Engineering*. Pearson, 2007.

[3] *UML 2 Unified Process*. Mc-Graw Hill, 2005.

[4] *Software Modelling Design*. Cambridge, 2010.

[5] *Spring Framework*. URL: https://docs.spring.io/spring/docs/.

[6] *Documentazione Vaadin*. URL: https://vaadin.com/docs.

[7] *Guida al Deployment*. URL: https://vaadin.com/learn/tutorials/modern-web-apps-with-spring-boot-and-vaadin/deploy-spring-boot-on-aws-elastic-beanstalk.