

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA “TULLIO LEVI-CIVITA”

CORSO DI LAUREA IN INFORMATICA



Sviluppo API REST di un'applicazione web per la pianificazione delle risorse aziendali

Tesi di laurea

Relatore

Prof. Paolo Baldan

Laureando

Marco Brigo

ANNO ACCADEMICO 2022-2023

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage, della durata di 320 ore, dal laureando Marco Brigo presso l'azienda Omicron Consulting Srl di Padova nel periodo che va dal 19 Giugno 2023 al 24 Agosto 2023.

Il progetto consisteva nello sviluppo di un software di richieste di pianificazioni delle risorse aziendali, verso determinati incarichi e secondo determinati parametri. L'obiettivo dello stage era inserirmi all'interno del progetto, più precisamente nel lato back-end. Il mio lavoro si concentrava sulla progettazione e l'implementazione di un'API_g REST e sulla creazione e la gestione di nuove tabelle nel database aziendale, utilizzate per la fruizione delle funzionalità da me implementate.

L'attività si è svolta partendo da uno studio preliminare delle tecnologie da utilizzare tramite esercitazioni e materiale fornito, passando per una progettazione dell'architettura del progetto e del database.

“Life is really simple, but we insist on making it complicated”

— Confucius

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Nome Cognome, relatore della mia tesi, per l'aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio.

Ho desiderio di ringraziare poi i miei amici per tutti i bellissimi anni passati insieme e le mille avventure vissute.

Padova, Mese AAAA

Nome Cognome

Indice

1	Introduzione	1
1.1	L'azienda ospitante	1
1.2	Il progetto	2
1.2.1	Presentazione	2
1.2.2	Motivazione del progetto	2
1.2.3	Il mio ruolo nel progetto	2
1.3	Organizzazione del testo	3
1.4	Convenzioni tipografiche nel documento	3
2	Analisi dei Requisiti	4
2.1	Descrizione generale	4
2.2	Semplificazioni adottate nei casi d'uso	4
2.3	Attori	5
2.4	Casi d'uso	6
2.4.1	Scenario Anagrafiche	6
2.4.2	Scenario Richieste	13
2.4.3	Scenario Pianificazioni	20
2.4.4	Scenario Milestone	27
2.5	Tracciamento dei requisiti	30
3	Background tecnologico	37
3.1	Tecnologie	37
3.2	Strumenti	39
3.2.1	Strumenti di sviluppo	39
3.2.2	Strumenti di supporto	40
4	Progettazione	42
4.1	Architettura REST	42
4.2	Spring MVC	43
4.2.1	Model	43
4.2.2	View	44
4.2.3	Controller	44
4.3	Pattern Client-Server	45
4.4	Design Pattern	45
4.5	Configurazione iniziale del progetto	47
4.5.1	Spring Initializr	47
4.5.2	Contenuto del pacchetto	48
4.5.2.1	Pom.xml	48

4.5.2.2	Application.properties	50
5	Modello dati	51
5.1	Configurazione di Docker	51
5.2	Progettazione del database	52
5.3	Gestione dei log nel sistema	58
5.3.1	Tabella PianificazioniAudit	59
6	Sviluppo API REST	61
6.1	Convenzioni di denominazione REST	61
6.2	Verbi standard HTTP	61
6.3	Endpoint sviluppati	62
6.4	Swagger	67
7	Codifica	70
7.1	Packages Common e config	70
7.1.1	Common	70
7.1.2	Config	71
7.2	Package Entities	71
7.2.1	Esempio di un'entità	72
7.2.2	Mappatura delle relazioni	73
7.2.3	Subpackage dtoentities	74
7.3	Package dto	74
7.3.1	Common	75
7.3.2	Requests e Body	76
7.3.3	Responses	77
7.4	Repository-Service-Controller	77
7.4.1	Controller	78
7.4.1.1	Configurazione di un controller	79
7.4.1.2	Generazione automatica dello Swagger	79
7.4.2	Repository	80
7.4.3	Service	81
7.4.3.1	Esempio con utilizzo di metodi dei Repository	81
7.4.3.2	Esempio di creazione query con filtri	82
7.4.3.3	Esempio di creazione file Excel	83
8	Verifica e Validazione	85
8.1	Processo di verifica	85
8.1.1	Analisi statica	85
8.1.2	Analisi dinamica	85
8.1.2.1	Debugging	85
8.1.2.2	Postman	86
8.2	Testing	86
8.2.1	Esempio di un test effettuato	87
8.3	Validazione	88
9	Deploy dell'applicazione	90
9.1	Deploy	90
9.1.1	Dockerfile	90
9.1.2	Costruzione dell'immagine e avvio dell'applicazione	91

10 Conclusioni	92
10.1 Consuntivo finale	92
10.2 Principali problematiche e soluzioni	93
10.3 Analisi critica del prodotto	93
10.3.1 Utilizzo del prodotto	93
10.3.2 Valutazione strumenti utilizzati	94
10.3.3 Completamento dei Requisiti	94
10.3.4 Possibili estensioni	94
10.4 Conoscenze acquisite	94
10.5 Valutazione personale	95
Glossario	96
Bibliografia	99

Elenco delle figure

1.1	Logo Omicron	1
2.1	Attori	5
2.2	Casi d'Uso del scenario Anagrafiche	6
2.3	Caso d'Uso 1 espanso	6
2.4	Caso d'Uso 3 espanso	8
2.5	Caso d'Uso 3.1 espanso	8
2.6	Caso d'Uso 4 espanso	9
2.7	Caso d'Uso 4.1 espanso	9
2.8	Caso d'Uso 5 espanso	10
2.9	Caso d'Uso 5.1 espanso	11
2.10	Caso d'Uso 6 espanso	11
2.11	Caso d'Uso 6.1 espanso	12
2.12	Casi d'Uso del scenario Richieste	13
2.13	Caso d'Uso 12 espanso	15
2.14	Caso d'Uso 12.1 espanso	16
2.15	Caso d'Uso 13 espanso	17
2.16	Caso d'Uso 13.1 espanso	17
2.17	Casi d'Uso del scenario Pianificazioni	20
2.18	Caso d'Uso 25 espanso	23
2.19	Caso d'Uso 25.1 espanso	24
2.20	Caso d'Uso 26 espanso	26
2.21	Caso d'Uso 26.1 espanso	26
2.22	Casi d'Uso del scenario Milestone	27
2.23	Casi d'Uso 30 espanso	28
2.24	Casi d'Uso 30.1 espanso	29
4.1	Schema Model-View-Controller	43
4.2	Client-Server in una REST API	45
4.3	Interfaccia Spring Initializr	47
4.4	Snippet pom.xml generato	49
4.5	Foto application.properties configurato e utilizzato	50
5.1	Schema Database	52
5.2	Codice Trigger: Inizializzazione variabili	59
5.3	Codice Trigger: Inserimento dei valori nella tabella di log	60
6.1	Inizio del documento di definizione dell'API	67
6.2	Example value del body richiesto dall'endpoint	68

6.3	Composizione oggetto response ritornato dall'endpoint	68
6.4	Errori possibili in un endpoint	69
7.1	Package common e config	70
7.2	Package entità	71
7.3	Esempio di mappatura di un'entità	72
7.4	Esempio di mappatura delle relazioni della tabella RichiestaTestata . .	73
7.5	Esempio di classe DTO di Pianificazione	74
7.6	Subpackage common	75
7.7	Subpackage requests e body	76
7.8	Subpackage responses e dtoresponses	77
7.9	Schema di collegamento tra i componenti Controller, Service e Repository	77
7.10	Controller sviluppati	78
7.11	Firma di un metodo nell'interfaccia di un Controller	79
7.12	Esempio di controllo di validazione	79
7.13	Annotazioni utili a generare la documentazione per l'API	80
7.14	Repository sviluppati	80
7.15	Esempio di una Repository	81
7.16	Service sviluppati	81
7.17	Metodo deleteMilestone() nel Service	82
7.18	Code snippet della creazione della query basata su filtri inseriti	82
7.19	Query dinamica	83
7.20	Code snippet semplificato della creazione di un file Excel	83
7.21	Response fornita dal Controller contenente il file Excel da scaricare . .	84
8.1	Interfaccia Postman	86
8.2	Configurazione ambiente di testing	87
8.3	Code snippet test di creazione di una nuova Milestone	88
9.1	Configurazione Dockerfile	90

Elenco delle tabelle

2.1	Lista dei Requisiti	36
5.1	Tabella sched_progetto	54
5.2	Tabella sched_richiesta_testata	54
5.3	Tabella sched_richiesta_testata_tipo_skill	55
5.4	Tabella sched_richiesta_dettaglio	56
5.5	Tabella sched_pianificazione	57

5.6	Tabella sched_milestone	58
6.1	Verbi Standard HTTP utilizzati	62
6.2	Endpoint Anagrafiche sviluppati	63
6.3	Endpoint Milestones sviluppati	64
6.4	Endpoint Pianificazioni sviluppati	65
6.5	Endpoint Richieste sviluppati	66

Capitolo 1

Introduzione

1.1 L'azienda ospitante



Figura 1.1: Logo Omicron

L'azienda Omicron Consulting è specializzata nello sviluppo di software gestionali e di revisione di processi aziendali. Essa è presente nel mercato ICT_g dal 1980, spiccando su vari settori, in cui hanno effettuato importanti implementazioni in area ICT come: Manufacturing, Automotive, Aerospace, Logistics e altre aree. Con particolare riferimento al settore Manufacturing si sono specializzati nello sviluppo di progetti di trasformazione ERP_g, stringendo alleanze strategiche con realtà ICT nazionali ed internazionali.

Offrono servizi di gestione e supporto di sistemi ERP, sviluppo di progetti di Business Intelligence_g e personalizzazione di sistemi software.

Per completare il pacchetto dei servizi offerti, Omicron ha un'esperienza di alto livello negli ambiti Banking, Finance and Insurance, lavorando con le principali istituzioni bancarie e assicurative italiane.

Omicron oltre alle offerte che dedica ai clienti, si occupa anche di garantire un'alta formazione delle proprie risorse, investendo su progetti di ricerca e sviluppo.

1.2 Il progetto

1.2.1 Presentazione

L'applicativo permette la creazione di richieste di pianificazioni di risorse aziendali per svolgere un determinato incarico. In questo progetto per risorse aziendali si fa sempre riferimento alle risorse umane, quindi al personale o alla forza lavoro dell'azienda.

Il Project Manager può inoltrare richieste per l'allocazione di risorse aziendali per svolgere una task rapida o un incarico relativo ad un progetto più ampio, chiedendo disponibilità di figure professionali con determinate skill richieste ed ulteriori parametri come l'effort e la seniority.

In una fase successiva, il Program Manager prende visione delle richieste effettuate dal Project Manager e si incarica di creare una Pianificazione per ciascuna figura professionale richiesta. Questa fase prevede l'assegnazione delle risorse adeguate a ciascuna richiesta, specificando parametri quali la durata, l'attività da svolgere e molti altri aspetti rilevanti per garantire una gestione efficiente delle risorse umane aziendali.

1.2.2 Motivazione del progetto

L'approccio tradizionale adottato per la gestione delle pianificazioni delle risorse e la loro disponibilità all'interno dell'azienda era basato su fogli Excel compilati e costantemente rivisti dai Program Manager. Le nuove richieste di pianificazioni provenienti dai Project Manager venivano comunicate ai Program Manager attraverso diversi canali, tra cui posta elettronica, telefono e chat. Questo metodo di comunicazione rendeva spesso arduo il compito di tenere traccia di tutte le richieste e di coordinare efficacemente le risorse umane.

Il progetto nasce dunque dall'esigenza di semplificare e modernizzare la gestione delle richieste e delle pianificazioni delle risorse aziendali. L'obiettivo principale è garantire una visione più rapida, accurata e centralizzata della disponibilità delle risorse, riducendo al minimo i potenziali conflitti e le sovrapposizioni nelle pianificazioni.

1.2.3 Il mio ruolo nel progetto

L'obiettivo principale di questa attività di stage era quello di inserirmi all'interno del progetto aziendale, concentrandomi specificamente sullo sviluppo del lato back-end. Il mio compito principale consisteva nell'implementare un'API REST e le relative tabelle del database su cui l'API si basava. Queste tabelle dovevano essere integrate con il database aziendale esistente, permettendo il recupero dei dati necessari per il funzionamento delle varie funzionalità.

Il mio lavoro consisteva nel sviluppare le funzionalità che consentivano operazioni di Creazione, Lettura, Aggiornamento e Cancellazione (CRUD_g) relative a richieste di figure professionali, pianificazioni e milestone da associare alle pianificazioni, oltre a operazioni di sola lettura sulle anagrafiche dell'azienda.

Prima di immergermi nell'implementazione, ho dedicato tempo a studiare le tecnologie e gli strumenti necessari per il progetto completando esercitazioni mirate per acquisire familiarità con le nuove tecnologie e approcci di sviluppo.

Durante la fase di sviluppo, mi sono basato su due documenti forniti: un documento mock_g dell'API e un'Analisi dei Requisiti che descriveva il prodotto nella sua totalità.

1.3 Organizzazione del testo

Questa sezione è dedicata alla spiegazione della struttura del documento, per dare indicazioni su come è organizzato il testo.

Capitolo 1: Introduzione, introduce il progetto, il mio ruolo all'interno del progetto e il profilo aziendale;

Capitolo 2: Analisi dei Requisiti, descrive i casi d'uso individuati ed i relativi requisiti;

Capitolo 3: Background Tecnologico, descrive le tecnologie studiate e gli strumenti utilizzati per sviluppare il prodotto;

Capitolo 4: Progettazione, descrive il periodo di progettazione dell'architettura su cui poggia il sistema e i concetti fondamentali;

Capitolo 5: Modello dati, parla della progettazione del database e delle tabelle utilizzate spiegandone l'utilizzo;

Capitolo 6: Sviluppo API REST, tratta dell'API sviluppata e della sua implementazione;

Capitolo 7: Codifica, descrive la codifica dell'API REST, l'organizzazione del codice, i moduli sviluppati e le parti più rilevanti;

Capitolo 8: Verifica e Validazione, descrive le fasi di Verifica e Validazione del prodotto;

Capitolo 9: Deploy dell'applicazione, descrive la fase di Deploy dell'applicazione;

Capitolo 10: Conclusioni, tratta delle considerazioni personali sull'attività svolta;

1.4 Convenzioni tipografiche nel documento

Durante la stesura del testo sono state adottate le seguenti convenzioni:

- la sezione del glossario contiene i termini ritenuti ambigui o non di uso comune, che necessitano quindi di una loro definizione. Il suo scopo è quello di fornire una comprensione comune del linguaggio utilizzato e di evitare confusione o interpretazioni errate;
- per la prima occorrenza di un termine inserito nel glossario viene utilizzata la seguente nomenclatura: termine_g.

Capitolo 2

Analisi dei Requisiti

Il seguente capitolo di Analisi dei Requisiti rappresenta una dettagliata e approfondita esplorazione delle necessità e delle aspettative che guidano la creazione e lo sviluppo del progetto in questione. Questa analisi è stata condotta al fine di definire chiaramente gli obiettivi e le funzionalità del prodotto, fornendo una base solida per la progettazione e l'implementazione del software.

2.1 Descrizione generale

Ogni caso d'uso è stato schematizzato secondo i seguenti punti:

- **attore coinvolto:** in cui si specifica l'attore;
- **descrizione:** offre una spiegazione più dettagliata del caso d'uso;
- **precondizioni:** rappresenta la condizione che deve essere soddisfatta e verificata affinché il caso d'uso possa essere eseguito con successo;
- **postcondizioni:** rappresenta lo stato dell'attore in seguito all'esecuzione con successo del caso d'uso;
- **estensioni:** in cui si specificano le eventuali estensioni;
- **inclusioni:** in cui si specificano le eventuali inclusioni.

Vengono inserite anche delle immagini dell'UML_g per fornire una spiegazione visiva che può aiutare maggiormente la comprensione.

2.2 Semplificazioni adottate nei casi d'uso

All'interno dei casi d'uso è possibile leggere l'abbreviazione "vis.". Il seguente termine è utilizzato per abbreviare la parola "Visualizzazione".

Per agevolare la lettura delle immagini dei casi d'uso non è stato inserito il database come attore esterno collegamento tra gli scenari principali e il database. È dato per scontato quindi, che ogni informazione venga recuperata dal database.

2.3 Attori

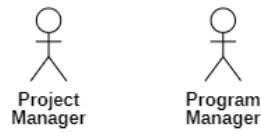


Figura 2.1: Attori

Gli attori che possiamo trovare all'interno dei casi d'uso rappresentano due risorse aziendali:

Project Manager

Conosciuto come il "Responsabile di Progetto", si occupa dell'avvio, pianificazione, esecuzione e controllo di un singolo progetto, seguendo tecniche di gestione del progetto. Le sue principali responsabilità sono:

- assicurarsi che i progetti siano allineati con gli obiettivi aziendali;
- coordinare le risorse umane, assicurandosi che vengano utilizzate in modo efficiente;
- stabilisce milestone, scadenze e obiettivi, monitorando lo stato di avanzamento dei progetti;
- comunica con stakeholder_g, team di progetto e altre parti interessate.

Program Manager

È un ruolo di gestione all'interno di un'organizzazione, ed è responsabile della pianificazione complessiva e del controllo di più progetti che compongono il suo programma. Collabora strettamente col Project Manager ed i loro compiti spesso si sovrappongono ma differiscono di portata, in quanto il Program Manager supervisiona gruppi di progetti gestiti singolarmente dai Project Manager.

2.4 Casi d'uso

2.4.1 Scenario Anagrafiche

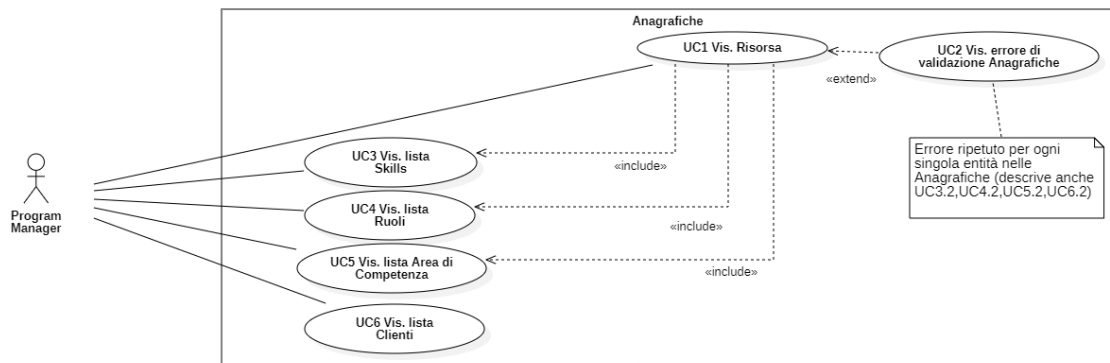


Figura 2.2: Casi d'Uso del scenario Anagrafiche

UC1 - Vis. Risorsa

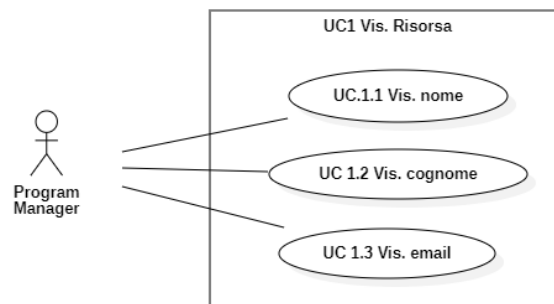


Figura 2.3: Caso d'Uso 1 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare una Risorsa;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Risorsa selezionata è visualizzabile dal Program Manager;
- **Estensioni:** UC2;
- **Inclusioni:** UC3, UC4, UC5.

UC1.1 - Vis. nome

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare il nome di una Risorsa;
- **Precondizioni:** la Risorsa è visualizzabile dal Program Manager;

- **Postcondizioni:** il Program Manager può visualizzare il nome della Risorsa selezionata;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC1.2 - Vis. cognome

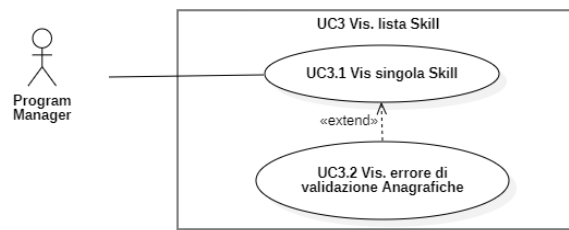
- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare il cognome di una Risorsa;
- **Precondizioni:** la Risorsa è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare il cognome della Risorsa selezionata;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC1.3 - Vis. email

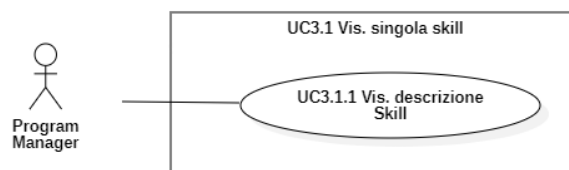
- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare l'email di una Risorsa;
- **Precondizioni:** la Risorsa è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare l'email della Risorsa selezionata;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC2 - Vis. errore di validazione Anagrafiche

- **Attore:** Program Manager;
- **Descrizione:** questo caso d'uso descrive anche UC3.2,UC4.2,UC5.2,UC6.2. Viene visualizzato un messaggio di errore in caso vengano eseguite funzionalità con dati non validi. Esso rappresenta errori di validazione nella richiesta fornita dall'utilizzatore;
- **Precondizioni:** il Program Manager sta effettuando operazioni con dati non validi;
- **Postcondizioni:** l'esecuzione della funzionalità è interrotta e viene visualizzato il messaggio di errore;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC3 - Vis. lista Skill**Figura 2.4:** Caso d'Uso 3 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la lista delle Skill;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la lista delle Skill è visualizzabile dal Program Manager;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC3.1 - Vis. singola Skill**Figura 2.5:** Caso d'Uso 3.1 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare una Skill;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Skill selezionata è visualizzabile dal Program Manager;
- **Estensioni:** UC3.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC3.1.1 - Vis. descrizione Skill

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la descrizione di una Skill;

- **Precondizioni:** la Skill è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare la descrizione della Skill selezionata;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC4 - Vis. lista Ruoli

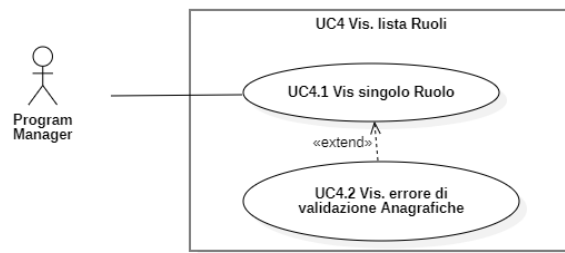


Figura 2.6: Caso d'Uso 4 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la lista dei Ruoli;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la lista dei Ruoli è visualizzabile dal Program Manager;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC4.1 - Vis. singolo Ruolo

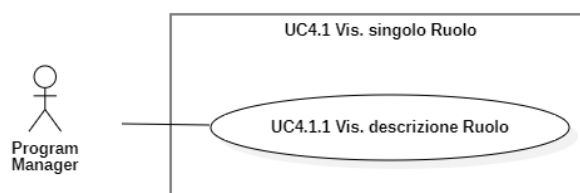


Figura 2.7: Caso d'Uso 4.1 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare un Ruolo;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** il Ruolo selezionato è visualizzabile dal Program Manager;

- **Estensioni:** UC4.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC4.1.1 - Vis. descrizione Ruolo

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la descrizione di un Ruolo;
- **Precondizioni:** il Ruolo è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare la descrizione del Ruolo selezionato;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC5 - Vis. lista Area di Competenza

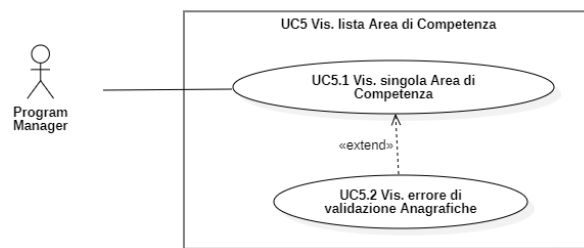
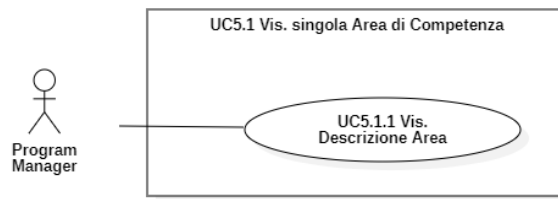


Figura 2.8: Caso d'Uso 5 espanso

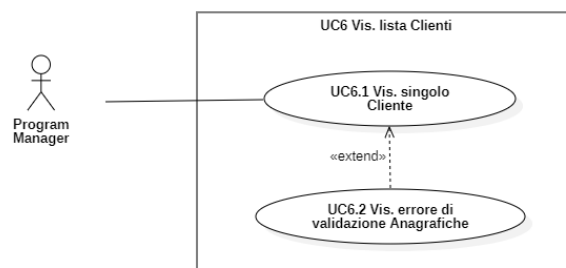
- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la lista delle Aree di Competenza;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la lista delle Aree di Competenza è visualizzabile dal Program Manager;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC5.1 - Vis. singola Area di Competenza**Figura 2.9:** Caso d'Uso 5.1 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare un'Area di Competenza;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** l'Area di Competenza selezionata è visualizzabile dal Program Manager;
- **Estensioni:** UC5.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC5.1.1 - Vis. descrizione Area

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la descrizione di un'Area di Competenza;
- **Precondizioni:** l'Area è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare la descrizione dell'Area di Competenza selezionata;
- **Estensioni:** non ci sono estensioni;
- **Inclusioni:** non ci sono inclusioni.

UC6 - Vis. lista Clienti**Figura 2.10:** Caso d'Uso 6 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la lista dei Clienti;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la lista dei Clienti è visualizzabile dal Program Manager;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC6.1 - Vis. singolo Cliente

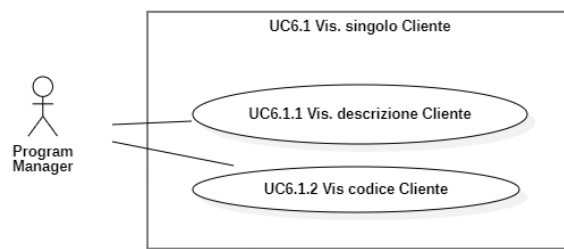


Figura 2.11: Caso d'Uso 6.1 espanso

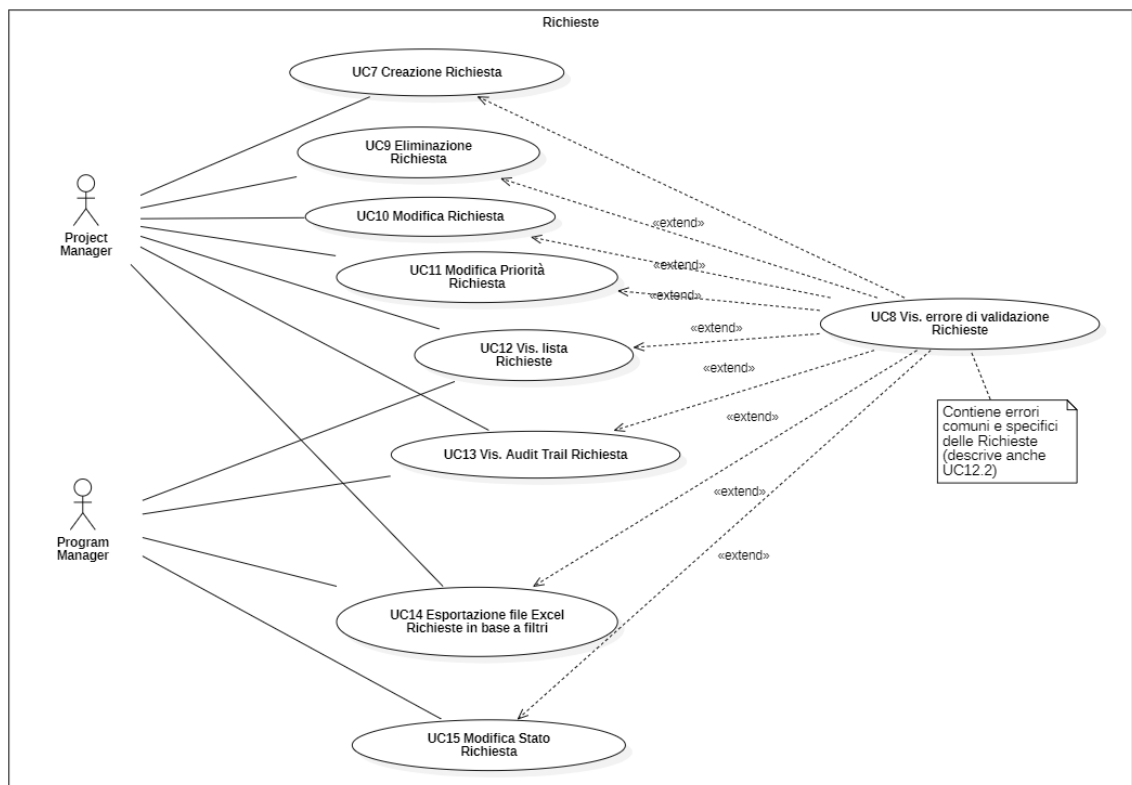
- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare un Cliente;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** il Cliente selezionato è visualizzabile dal Program Manager;
- **Estensioni:** UC6.2;
- **Inclusioni:** non ci sono inclusioni.

UC6.1.1 - Vis. descrizione Cliente

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la descrizione di Cliente;
- **Precondizioni:** il Cliente è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare la descrizione del Cliente selezionato;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC6.1.2 - Vis. codice Cliente

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare il codice del Cliente;
- **Precondizioni:** il Cliente è visualizzabile dal Program Manager;
- **Postcondizioni:** il Program Manager può visualizzare il codice del Cliente selezionato;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

2.4.2 Scenario Richieste**Figura 2.12:** Casi d'Uso del scenario Richieste**UC7 - Creazione Richiesta**

- **Attore:** Project Manager;
- **Descrizione:** il Project Manager può creare una nuova Richiesta;
- **Precondizioni:** il richiedente è un Project Manager;

- **Postcondizioni:** la Richiesta è stata creata dal Project Manager con successo;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC8 - Vis. errore di validazione Richieste

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** questo caso d'uso descrive anche UC12.2. Viene visualizzato un messaggio di errore in caso vengano eseguite funzionalità con dati non validi. Esso rappresenta i seguenti errori comuni all'interno delle Richieste: dati non validi, filtri non valorizzati, entità associate non valide, risultati nulli o non valorizzati;
- **Precondizioni:** il Program Manger o il Project Manager stanno effettuando operazioni con dati non validi;
- **Postcondizioni:** l'esecuzione della funzionalità è interrotta e viene visualizzato il messaggio di errore;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC9 - Eliminazione Richiesta

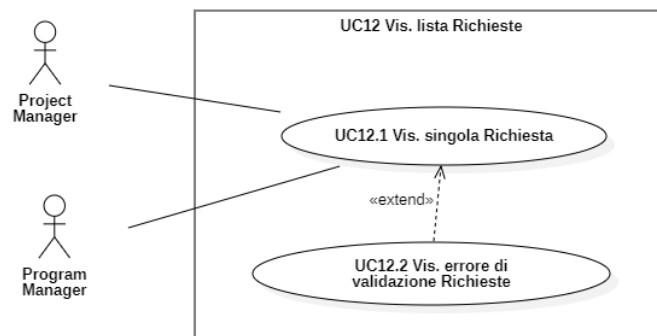
- **Attore:** Project Manager;
- **Descrizione:** il Project Manager può eliminare una Richiesta esistente;
- **Precondizioni:** il richiedente è un Project Manager;
- **Postcondizioni:** la Richiesta è stata eliminata dal Project Manager con successo;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC10 - Modifica Richiesta

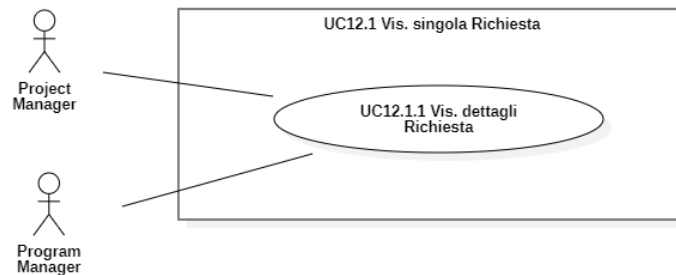
- **Attore:** Project Manager;
- **Descrizione:** il Project Manager può modificare una Richiesta esistente nella sua totalità sovrascrivendola;
- **Precondizioni:** il richiedente è un Project Manager;
- **Postcondizioni:** la Richiesta selezionata è stata modificata con successo;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC11 - Modifica Priorità Richiesta

- **Attore:** Project Manager;
- **Descrizione:** il Project Manager può modificare l'attributo Priorità di una Richiesta esistente inserendo un valore tra: Alta, Media o Bassa;
- **Precondizioni:** il richiedente è un Project Manager;
- **Postcondizioni:** la Richiesta è stata modificata con successo solo nel campo Priorità dal Project Manager;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC12 - Vis. lista Richieste**Figura 2.13:** Caso d'Uso 12 espanso

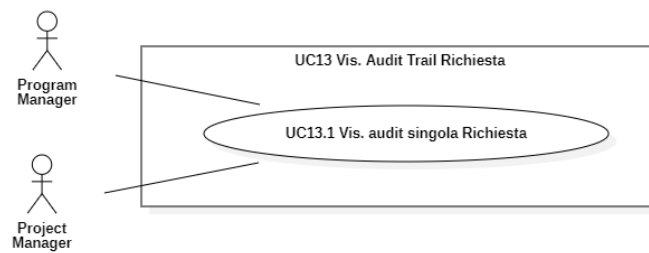
- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare una lista di Richieste dopo aver inserito filtri e/o una parola nella ricerca rapida e aver selezionato se i filtri applicati devono essere congiunti o disgiunti;
- **Precondizioni:** il richiedente è un Project Manager o un Program Manager;
- **Postcondizioni:** la lista delle Richieste è visualizzabile dal Project Manager e dal Program Manager;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC12.1 - Vis. singola Richiesta**Figura 2.14:** Caso d'Uso 12.1 espanso

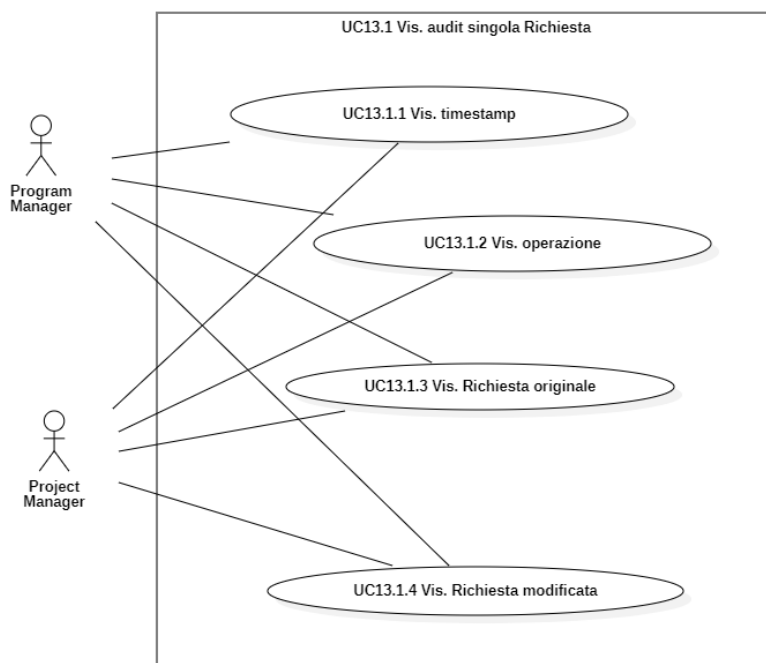
- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare la Richiesta selezionata;
- **Precondizioni:** la lista delle Richieste è visualizzabile;
- **Postcondizioni:** la Richiesta selezionata è visualizzabile dal Program Manager e dal Project Manager;
- **Estensioni:** UC12.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC12.1.1 - Vis. dettagli Richiesta

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare la Richiesta selezionata;
- **Precondizioni:** la Richiesta singola è visualizzabile;
- **Postcondizioni:** il Project Manager e il Program Manager possono visualizzare i campi di una Richiesta selezionata;
- **Estensioni:** il caso d'uso non ha esclusioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13 - Vis. audit_g trail Richiesta**Figura 2.15:** Caso d'Uso 13 espanso

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare l'audit trail di una Richiesta selezionata.;
- **Precondizioni:** il richiedente è un Project Manager o un Program Manager;
- **Postcondizioni:** la traccia di audit della Richiesta selezionata è visualizzabile dal Project Manager e dal Program Manager;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13.1 - Vis. audit singola Richiesta**Figura 2.16:** Caso d'Uso 13.1 espanso

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare l'audit di una singola Richiesta;
- **Precondizioni:** l'audit trail è visualizzabile;
- **Postcondizioni:** l'audit di una singola Richiesta è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13.1.1 - Vis. timestamp_g

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare il timestamp di una singola audit di una Richiesta;
- **Precondizioni:** la singola audit di una Richiesta è visualizzabile;
- **Postcondizioni:** il timestamp della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13.1.2 - Vis. operazione

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare l'operazione di una singola audit di una Richiesta;
- **Precondizioni:** la singola audit di una Richiesta è visualizzabile;
- **Postcondizioni:** l'operazione della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13.1.3 - Vis. Richiesta originale

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare la singola audit di una Richiesta prima che l'operazione venga eseguita;
- **Precondizioni:** la singola audit di una Richiesta è visualizzabile;
- **Postcondizioni:** la Richiesta originale della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC13.1.4 - Vis. Richiesta modificata

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare la singola audit di una Richiesta dopo che l'operazione è stata eseguita;
- **Precondizioni:** la singola audit di una Richiesta è visualizzabile;
- **Postcondizioni:** la Richiesta modificata della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC14 - Esportazione file Excel Richieste in base a filtri

- **Attore:** Project Manager e Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono esportare in un file Excel scaricabile un report delle Richieste in base a dei filtri inseriti;
- **Precondizioni:** il richiedente è un Project Manager o un Program Manager;
- **Postcondizioni:** il report Excel è stato generato correttamente ed è possibile scaricarlo;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC15 - Modifica Stato Richiesta

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può cambiare lo stato di una Richiesta esistente impostandolo in uno dei seguenti valori: Evasa, Non Evasa, Aperta, In corso, Chiusa;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** solo lo stato della Richiesta è stato modificato con successo;
- **Estensioni:** UC8;
- **Inclusioni:** il caso d'uso non ha inclusioni.

2.4.3 Scenario Pianificazioni

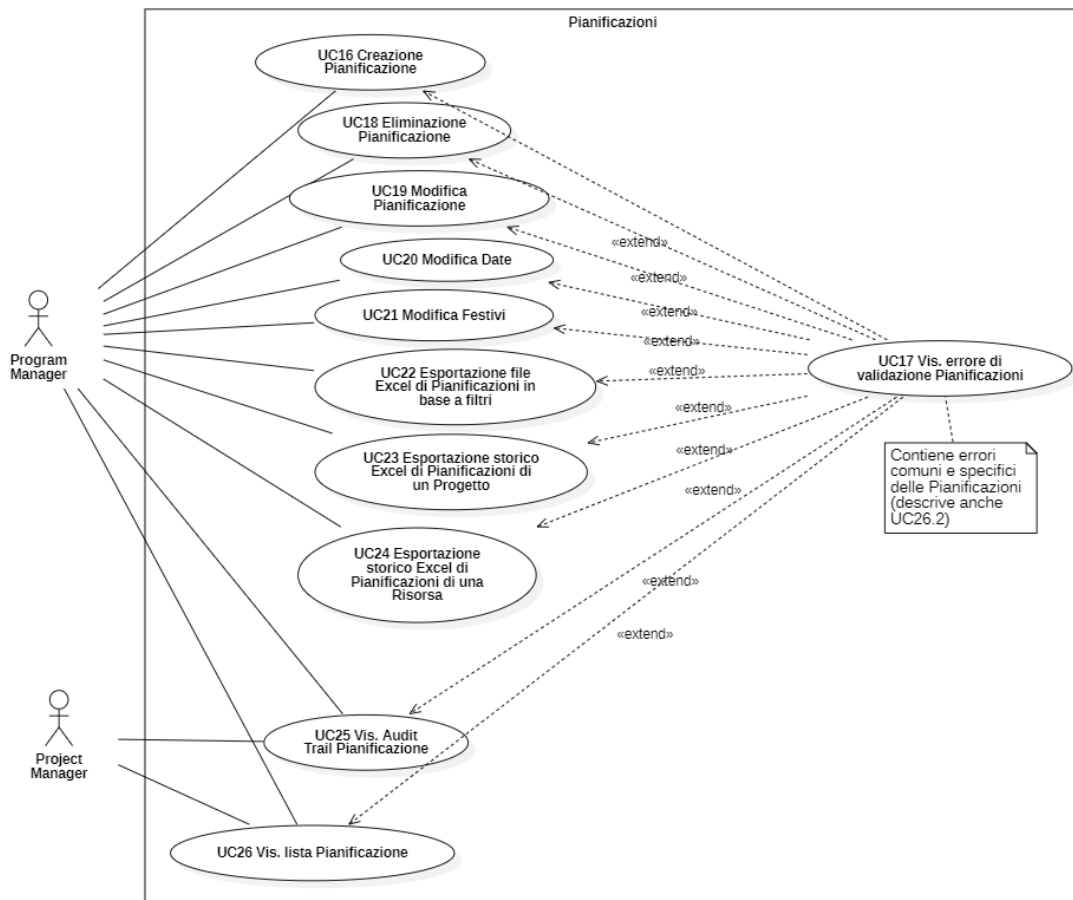


Figura 2.17: Casi d'Uso del scenario Pianificazioni

UC16 - Creazione Pianificazione

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può creare una Pianificazione per una figura professionale richiesta;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Pianificazione è stata creata dal Program Manager con successo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC17 - Vis. errore di validazione Pianificazioni

- **Attore:** Project Manager e Program Manager;

- **Descrizione:** questo caso d'uso descrive anche UC26.2. Viene visualizzato un messaggio di errore in caso vengano eseguite funzionalità con dati non validi. Esso rappresenta i seguenti errori comuni all'interno delle Pianificazioni: dati non validi, filtri non valorizzati, entità associate non valide, risultati nulli o non valorizzati;
- **Precondizioni:** il Program Manager o il Project Manager stanno effettuando operazioni con dati non validi;
- **Postcondizioni:** l'esecuzione della funzionalità è interrotta e viene visualizzato il messaggio di errore;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC18 - Eliminazione Pianificazione

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può eliminare una Pianificazione esistente;
- **Precondizioni:** il richiedente è il Program Manager;
- **Postcondizioni:** la Pianificazione è stata eliminata dal Program Manager con successo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC19 - Modifica Pianificazione

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può modificare una Pianificazione esistente nella sua totalità sovrascrivendola;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Pianificazione selezionata è stata modificata con successo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC20 - Modifica Date

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può modificare le date di inizio e/o di fine di una Pianificazione;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Pianificazione è stata modificata con successo solo nel campo Data inizio e/o Data fine dal Program Manager;

- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC21 - Modifica Festivi

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può modificare il campo Festivi di una Pianificazione. Se posto a vero il lavoratore potrà lavorare anche nei giorni festivi;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Pianificazione è stata modificata con successo solo nel campo Festivi dal Program Manager;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC22 - Esportazione file Excel di Pianificazioni in base a filtri

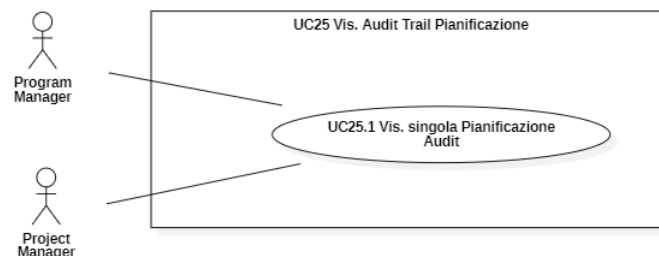
- **Attore:** Program Manager;
- **Descrizione:** : il Program Manager può esportare in un file Excel scaricabile un report delle Pianificazioni in base a dei filtri inseriti;
- **Precondizioni:** : il richiedente è un Program Manager;
- **Postcondizioni:** il report Excel è stato generato correttamente ed è possibile scaricarlo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC23 - Esportazione storico Excel di Pianificazioni di un Progetto

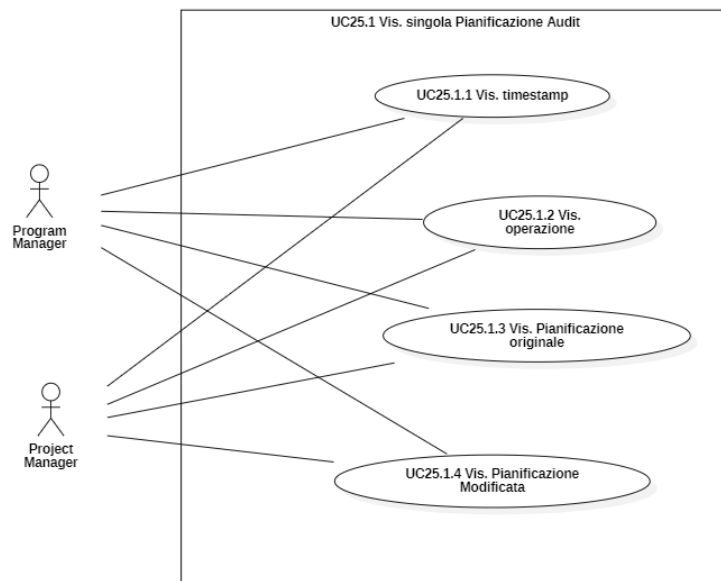
- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può esportare in un file Excel scaricabile uno storico di Pianificazioni associate ad un Progetto contenente tutte le risorse allocate e l'effettivo impiego di queste nelle attività;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** il report Excel è stato generato correttamente ed è possibile scaricarlo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC24 - Esportazione storico Excel di Pianificazioni di una Risorsa

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può esportare in un file Excel scaricabile uno storico di Pianificazioni in cui ha lavorato una determinata Risorsa, visualizzando data di fine e di inizio e le relative mansioni;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** il report Excel è stato generato correttamente ed è possibile scaricarlo;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25 - Vis. Audit Trail Pianificazione**Figura 2.18:** Caso d'Uso 25 espanso

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare l'audit trail di una Richiesta selezionata. Il Project Manager può vedere solo le tracce di audit delle Pianificazioni associate alle sue Richieste, mentre il Program Manager può vederle tutte;
- **Precondizioni:** il richiedente è un Program Manager o un Project Manager;
- **Postcondizioni:** la traccia di audit della Pianificazione selezionata è visualizzabile dal Project Manager o dal Program Manager;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25.1 - Vis. singola Pianificazione Audit**Figura 2.19:** Caso d'Uso 25.1 espanso

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager può vedere l'audit di una singola Pianificazione. Il Project Manager può vedere solo le Pianificazioni associate alle sue Richieste, mentre il Program Manager può vederle tutte;
- **Precondizioni:** l'audit trail è visualizzabile;
- **Postcondizioni:** l'audit di una singola Pianificazione è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25.1.1 - Vis. timestamp

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare il timestamp di una singola audit di una Pianificazione;
- **Precondizioni:** la singola audit di una Pianificazione è visualizzabile;
- **Postcondizioni:** il timestamp della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25.1.2 - Vis. operazione

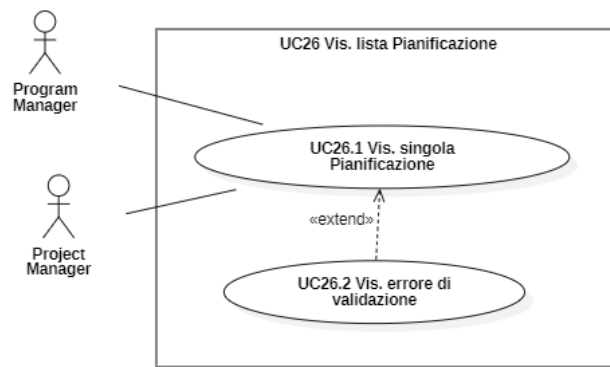
- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager e il Program Manager possono visualizzare l'operazione di una singola audit di una Pianificazione;
- **Precondizioni:** la singola audit di una Pianificazione è visualizzabile;
- **Postcondizioni:** l'operazione della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25.1.3 - Vis. Pianificazione originale

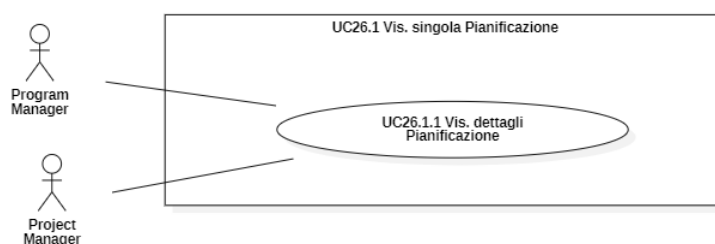
- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare la singola audit di una Pianificazione prima che l'operazione venga eseguita;
- **Precondizioni:** la singola audit di una Pianificazione è visualizzabile;
- **Postcondizioni:** la Richiesta originale della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC25.1.4 - Vis. Pianificazione modificata

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare la singola audit di una Pianificazione dopo che l'operazione è stata eseguita;
- **Precondizioni:** la singola audit di una Pianificazione è visualizzabile;
- **Postcondizioni:** la Pianificazione modificata della singola audit è visualizzabile;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC26 - Vis. lista Pianificazione**Figura 2.20:** Caso d'Uso 26 espanso

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare una lista di Richieste dopo aver inserito filtri e/o una parola nella ricerca rapida e aver selezionato se i filtri applicati devono essere congiunti o disgiunti. Il Project Manager può vedere solo le Pianificazioni associate alle sue Richieste, mentre il Program Manager può vederle tutte;
- **Precondizioni:** il richiedente è un Project Manager o un Program Manager;
- **Postcondizioni:** la lista delle Richieste è visualizzabile dal Project Manager o dal Program Manager;
- **Estensioni:** UC17;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC26.1 - Vis. singola Pianificazione**Figura 2.21:** Caso d'Uso 26.1 espanso

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare la Pianificazione selezionata. Il Project Manager può vedere solo le Pianificazioni associate alle sue Richieste, mentre il Program Manager può vederle tutte;

- **Precondizioni:** la lista delle Pianificazioni è visualizzabile;
- **Postcondizioni:** la Pianificazione selezionata è visualizzabile dal Program Manager o dal Project Manager;
- **Estensioni:** UC26.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC26.1.1 - Vis. dettagli Pianificazione

- **Attore:** Project Manager o Program Manager;
- **Descrizione:** il Project Manager o il Program Manager possono visualizzare la Pianificazione selezionata;
- **Precondizioni:** la Pianificazione singola è visualizzabile;
- **Postcondizioni:** il Project Manager e il Program Manager possono visualizzare i campi di una Pianificazione selezionata;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

2.4.4 Scenario Milestone

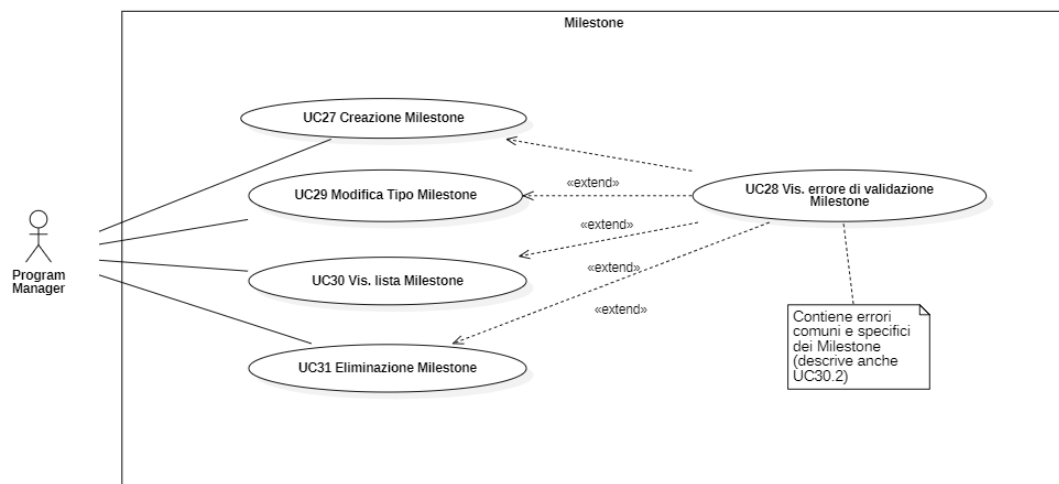


Figura 2.22: Casi d'Uso del scenario Milestone

UC27 - Creazione Milestone

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può creare una nuova Milestone da associare ad una Pianificazione;
- **Precondizioni:** il richiedente è un Program Manager;

- **Postcondizioni:** la Milestone è stata creata dal Program Manager con successo;
- **Estensioni:** UC28;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC28 - Vis. errore di validazione Milestone

- **Attore:** Program Manager;
- **Descrizione:** questo caso d'uso descrive anche UC30.2. Viene visualizzato un messaggio di errore in caso vengano eseguite funzionalità con dati non validi. Esso rappresenta i seguenti errori comuni all'interno delle Milestone: dati non validi, filtri non valorizzati, entità associate non valide, risultati nulli o non valorizzati;
- **Precondizioni:** il Program Manager sta effettuando operazioni con dati non validi;
- **Postcondizioni:** l'esecuzione della funzionalità è interrotta e viene visualizzato il messaggio di errore;
- **Estensioni:** il caso d'uso non ha estensioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC29 - Modifica Tipo Milestone

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può modificare il Tipo di una Milestone in due possibili valori: Alert o Reminder;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Milestone è stata modificata con successo solo nel campo Tipo dal Program Manager;
- **Estensioni:** UC28;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC30 - Vis. lista Milestone

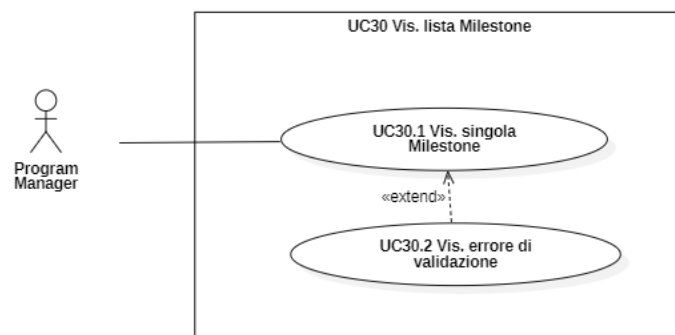


Figura 2.23: Casi d'Uso 30 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare una lista di Milestone dopo aver inserito filtri e/o una parola nella ricerca rapida e aver selezionato se i filtri applicati devono essere congiunti o disgiunti;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la lista delle Milestone è visualizzabile dal Program Manager;
- **Estensioni:** UC28;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC30.1 - Vis. singola Milestone

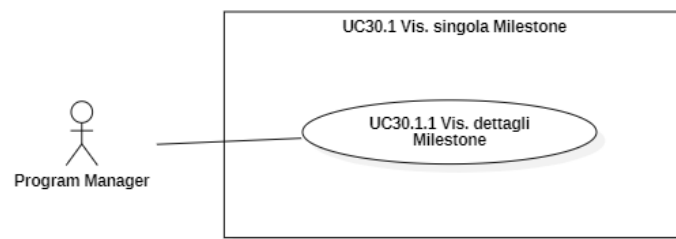


Figura 2.24: Casi d'Uso 30.1 espanso

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la Milestone selezionata;
- **Precondizioni:** la lista delle Milestone è visualizzabile;
- **Postcondizioni:** la Milestone selezionata è visualizzabile dal Program Manager;
- **Estensioni:** UC30.2;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC30.1.1 - Vis. dettagli Milestone

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può visualizzare la Milestone selezionata;
- **Precondizioni:** la Milestone singola è visualizzabile;
- **Postcondizioni:** il Project Manager può visualizzare i campi di una Milestone selezionata;
- **Estensioni:** il caso d'uso non ha esclusioni;
- **Inclusioni:** il caso d'uso non ha inclusioni.

UC31 - Eliminazione Milestone

- **Attore:** Program Manager;
- **Descrizione:** il Program Manager può eliminare una Milestone esistente;
- **Precondizioni:** il richiedente è un Program Manager;
- **Postcondizioni:** la Milestone è stata eliminata dal Program Manager con successo;
- **Estensioni:** UC28;
- **Inclusioni:** il caso d'uso non ha inclusioni.

2.5 Tracciamento dei requisiti

Di seguito elenco i requisiti funzionali estrapolati dallo studio degli use case. Per poterli elencare e distinguere è stata utilizzata il seguente codice identificativo:

RF[OB/DE/OP]-[Numero progressivo]

All'interno del codice possiamo osservare RF che significa *Requisito funzionale*, mentre le altre abbreviazioni indicano l'importanza del requisito:

- *OB*, obbligatorio;
- *DE*, desiderabile;
- *OP*, opzionale.

La tabella elenca i requisiti per Codice, come descritto, Descrizione del requisito, e la Fonte che può essere una decisione interna o un use case.

Codice	Descrizione	Fonte
RFOP-01	Deve essere predisposto un accesso controllato al sistema, che consenta solo a utente autorizzati di effettuare determinate operazioni	Decisione interna
RFOB-02	Il sistema permette la visualizzazione dei dettagli di una Risorsa selezionata	UC1
RFOB-03	Il sistema permette la visualizzazione del nome della Risorsa	UC1.1
RFOB-04	Il sistema permette la visualizzazione del cognome della Risorsa	UC1.2

RFOB-05	Il sistema permette la visualizzazione dell'email della Risorsa	UC1.3
RFOB-06	Il sistema controlla se si sta eseguendo operazioni nell'ambito Anagrafiche con dati non validi	UC2
RFOB-07	Il sistema controlla se il dato fornito non ha portato ad alcun risultato	UC2
RFOB-08	Il sistema permette la visualizzazione della lista di Skills	UC3
RFOB-09	Il sistema permette la visualizzazione di una singola Skill dalla lista	UC3.1
RFOB-10	Il sistema permette la visualizzazione diretta di una singola Skill	UC3.1
RFOB-11	Il sistema verifica che la richiesta alla singola Skill sia valida	UC3.2
RFOB-12	Il sistema permette la visualizzazione della descrizione della Skill	UC3.1.1
RFOB-13	Il sistema permette la visualizzazione della lista di Ruoli	UC4
RFOB-14	Il sistema permette la visualizzazione di uno singolo Ruolo dalla lista	UC4.1
RFOB-15	Il sistema permette la visualizzazione diretta di un singolo Ruolo	UC4.1
RFOB-16	Il sistema verifica che la richiesta al singolo Ruolo sia valida	UC4.2
RFOB-17	Il sistema permette la visualizzazione della descrizione del Ruolo	UC4.1.1
RFOB-18	Il sistema permette la visualizzazione della lista di Area di Competenza	UC5

RFOB-19	Il sistema permette la visualizzazione di una singola Area di Competenza dalla lista	UC5.1
RFOB-20	Il sistema permette la visualizzazione diretta di una singola Area di Competenza	UC5.1
RFOB-21	Il sistema verifica che la richiesta alla singola Area di Competenza sia valida	UC5.2
RFOB-22	Il sistema permette la visualizzazione della descrizione dell'Area di Competenza	UC5.1.1
RFOB-23	Il sistema permette la visualizzazione della lista di Clienti	UC6
RFOB-24	Il sistema permette la visualizzazione di uno singolo Cliente dalla lista	UC6.1
RFOB-25	Il sistema permette la visualizzazione diretta di un singolo Cliente	UC6.1
RFOB-26	Il sistema verifica che la richiesta al singolo Cliente sia valida	UC6.2
RFOB-27	Il sistema permette la visualizzazione della descrizione del Cliente	UC6.1.1
RFOB-28	Il sistema permette la visualizzazione del codice del Cliente	UC6.1.2
RFOB-29	Il sistema permette la creazione di una nuova Richiesta di figure professionali	UC7
RFOB-30	Il sistema controlla che il Richiedente esista	UC8
RFOB-31	Il sistema controlla che le Figure richieste e le Skill richieste siano coerenti	UC8
RFOB-32	Il sistema controlla che il Cliente associato sia corretto	UC8
RFOB-33	Il sistema controlla che il Progetto associato sia corretto	UC8

RFOB-34	Il sistema controlla che una Richiesta che si vuole eliminare non sia associata ad una Pianificazione	UC8
RFOB-35	Il sistema controlla che i nuovi valori per la modifica di un singolo campo siano corretti	UC8,UC17,UC28
RFOB-36	Il sistema controlla che i filtri inseriti siano valorizzati	UC8,UC17,UC28
RFOB-37	Il sistema controlla che l'entità su cui si va ad operare esista	UC8,UC17,UC28
RFOB-38	Il sistema permette l'eliminazione di una Richiesta	UC9
RFOB-39	Il sistema permette la modifica totale di una Richiesta esistente sovrascrivendola	UC10
RFOB-40	Il sistema permette la modifica del solo campo Priorità di una Richiesta esistente	UC11
RFOB-41	Il sistema permette la visualizzazione della lista di Richieste risultanti in seguito ad una richiesta formata da: filtri forniti dall'utente, parola da inserire nella ricerca rapida e se i filtri devono essere congiunti o disgiunti	UC12
RFOB-42	Il sistema permette la visualizzazione di una singola Richiesta dalla lista	UC12.1
RFOB-43	Il sistema permette la visualizzazione diretta di una singola Richiesta	UC12.1
RFOB-44	Il sistema verifica che la richiesta alla singola Richiesta sia valida	UC12.2
RFOB-45	Il sistema permette la visualizzazione dei dettagli della Richiesta selezionata	UC12.1.1
RFDE-46	Il sistema permette la visualizzazione della traccia di audit di una Richiesta	UC13
RFDE-47	Il sistema permette la visualizzazione di un audit di una Richiesta della traccia	UC13.1

RFDE-48	Il sistema permette la visualizzazione del timestamp dell'operazione in un singolo audit di un audit trail	UC13.1.1
RF4DE-9	Il sistema permette la visualizzazione dell'operazione in un singolo audit di un audit trail	UC13.1.2
RFDE-50	Il sistema permette la visualizzazione della Richiesta originale in un singolo audit di un audit trail	UC13.1.3
RFDE-51	Il sistema permette la visualizzazione della Richiesta modificata in un singolo audit di un audit trail	UC13.1.4
RFDE-52	Il sistema permette di esportare un file Excel contenente le Richieste risultanti dai filtri inseriti dall'utente nella richiesta	UC14
RFOB-53	Il sistema permette la modifica dell'attributo Stato di una Richiesta	UC15
RFOB-54	Il sistema permette la creazione di una nuova Pianificazione	UC16
RFOB-55	Il sistema controlla che il Progetto associato sia corretto	UC17
RFOB-56	Il sistema controlla che la Risorsa non sia occupata in quel Ruolo	UC17
RFOB-57	Il sistema controlla che la Risorsa possa svolgere il Ruolo richiesto	UC17
RFOB-58	Il sistema permette l'eliminazione di una Pianificazione	UC18
RFOB-59	Il sistema permette la modifica totale di una Pianificazione sovrascrivendola	UC19
RFOB-60	Il sistema permette la modifica delle date di una Pianificazione	UC20
RFOB-61	Il sistema permette la modifica del campo Festivi di una Pianificazione	UC21

RFDE-62	Il sistema permette di esportare un file Excel contenente le Pianificazioni risultanti dai filtri inseriti dall'utente nella richiesta	UC22
RFOP-63	Il sistema permette di esportare un file Excel contenente lo storico delle Pianificazioni relative ad un Progetto, mostrando le Risorse allocate e l'effettivo impiego di queste nelle attività	UC23
RFOP-64	Il sistema permette di esportare un file Excel contenente lo storico delle Pianificazioni di una singola Risorsa, visualizzando data fine e data inizio e le relative mansioni	UC24
RFDE-65	Il sistema permette la visualizzazione della traccia di audit di una Pianificazione	UC25
RFDE-66	Il sistema permette la visualizzazione di un audit di una Pianificazione della traccia	UC25.1
RFDE-67	Il sistema permette la visualizzazione del timestamp dell'operazione in un singolo audit di un audit trail	UC25.1.1
RFDE-68	Il sistema permette la visualizzazione dell'operazione in un singolo audit di un audit trail	UC25.1.2
RFDE-69	Il sistema permette la visualizzazione della Pianificazione originale in un singolo audit di un audit trail	UC25.1.3
RFDE-70	Il sistema permette la visualizzazione della Pianificazione modificata in un singolo audit di un audit trail	UC25.1.4
RFOB-71	Il sistema permette la visualizzazione della lista di Pianificazioni risultanti in seguito ad una richiesta formata da: filtri forniti dall'utente, parola da inserire nella ricerca rapida e se i filtri devono essere congiunti o disgiunti	UC26
RFOB-72	Il sistema permette la visualizzazione di una singola Pianificazione dalla lista	UC26.1
RFOB-73	Il sistema permette la visualizzazione diretta di una singola Pianificazione	UC26.1

RFOB-74	Il sistema verifica che la richiesta alla singola Pianificazione sia valida	UC26.2
RFOB-75	Il sistema permette la visualizzazione dei dettagli della Pianificazione selezionata	UC26.1.1
RFOB-76	Il sistema permette di creare una nuova Milestone da associare ad una Pianificazione	UC27
RFOB-77	Il sistema controlla che il Progetto, il Commerciale associati alla Milestone siano corretti	UC28
RFOB-78	Il sistema controlla che la Milestone che si vuole eliminare non sia associata ad una Pianificazione	UC28
RFOB-79	Il sistema permette la modifica totale di una Milestone sovrascrivendola	UC29
RFOB-80	Il sistema permette la visualizzazione della lista di Milestone risultanti in seguito ad una richiesta formata da: filtri forniti dall'utente, parola da inserire nella ricerca rapida e se i filtri devono essere congiunti o disgiunti	UC30
RFOB-81	Il sistema permette la visualizzazione di una singola Milestone dalla lista	UC30.1
RFOB-82	Il sistema permette la visualizzazione diretta di una singola Milestone	UC30.1
RFOB-83	Il sistema verifica che la richiesta alla singola Pianificazione sia valida	UC30.2
RFOB-84	Il sistema permette la visualizzazione dei dettagli della Pianificazione selezionata	UC30.1.1
RFOB-85	Il sistema permette l'eliminazione di una Milestone	UC31

Tabella 2.1: Lista dei Requisiti

Capitolo 3

Background tecnologico

Questo capitolo tratta delle tecnologie e gli strumenti di sviluppo e di supporto utilizzati per la realizzazione del prodotto.

L'apprendimento delle seguenti tecnologie e strumenti è stato affrontato nella prima parte del tirocinio. In questo periodo di formazione il tutor mi ha fornito materiale ed esercitazioni per poter comprendere al meglio il contesto tecnologico.

3.1 Tecnologie

Java

Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e fortemente tipizzato, sviluppato originariamente da Sun Microsystems.

La sua popolarità deriva dalla sua portabilità, dalla vasta comunità di sviluppatori e dalle numerose risorse disponibili per l'apprendimento e lo sviluppo.

All'interno del mio progetto è stato utilizzato per lo sviluppo del lato back-end del prodotto.

SQL

SQL, acronimo di Structured Query Language, è un linguaggio di programmazione utilizzato per gestire e manipolare dati in un database relazionale. SQL fornisce una serie di comandi standardizzati che consentono agli sviluppatori e agli amministratori di database di eseguire operazioni come l'interrogazione dei dati, l'aggiornamento dei dati, l'inserimento di nuovi dati e la creazione e gestione degli schemi dei database.

Questo noto linguaggio è stato utilizzato nel progetto per i seguenti motivi:

- creare le tabelle o trigger utili alla fruizione dei servizi del progetto;
- eseguire query per inserire, recuperare, eliminare o modificare dati in base alle richieste.

Microsoft SQL Server

SQL Server è un DBMS (Database Management System) relazionale sviluppato da Microsoft. È una piattaforma dati che si utilizza per creare e gestire database, principalmente in ambito aziendale.

Spring

Spring è un framework di sviluppo di applicazioni Java che offre un'ampia gamma di strumenti e librerie per semplificare la creazione di applicazioni aziendali.

Spring fornisce anche moduli specifici per la gestione dei dati, lo sviluppo web e la sicurezza, rendendolo uno degli strumenti più utilizzati per lo sviluppo Java.

A questo framework sono associati tanti altri progetti, che hanno nomi composti come Spring Boot, Spring Data e molti altri.

All'interno del progetto Spring è stato utilizzato per sviluppare l'API REST.

Spring Boot

Spring Boot è un modulo del framework di sviluppo Java Spring che semplifica la creazione, la configurazione e l'avvio di applicazioni Java. Fornisce un ambiente pronto all'uso per sviluppare rapidamente applicazioni Spring, eliminando gran parte della complessità associata alla configurazione. Spring Boot utilizza convenzioni intelligenti e configurazioni predefinite per accelerare lo sviluppo, consentendo agli sviluppatori di concentrarsi sulle funzionalità dell'applicazione anziché sulla configurazione di base.

Spring Data

Spring Data è un modulo del framework Spring che fornisce un'astrazione per semplificare l'accesso e la gestione dei dati nelle applicazioni Java. Esso offre un'interfaccia unificata per interagire con una varietà di fonti di dati, tra cui database relazionali, database NoSQL e altri servizi di memorizzazione dati.

Questo modulo è stato utile nel progetto per interfacciare l'applicazione con il database.

JSON

JSON, acronimo di JavaScript Object Notation, è un formato leggero di scambio di dati utilizzato comunemente per rappresentare oggetti e strutture di dati. JSON è ampiamente utilizzato per rappresentare dati in applicazioni web, servizi API, scambio di dati tra client e server, configurazioni di applicazioni e molto altro.

All'interno del progetto permette lo scambio di dati tra il lato front-end ed il lato back-end.

3.2 Strumenti

3.2.1 Strumenti di sviluppo

Gli strumenti di sviluppo sono utilizzati direttamente per creare, implementare e testare le funzionalità dell'applicazione. Essi contribuiscono alla realizzazione delle funzionalità dell'applicazione stessa.

IntelliJ IDEA

IntelliJ IDEA è un potente ambiente di sviluppo integrato (IDE) sviluppato da JetBrains, progettato principalmente per la programmazione in linguaggi come Java, Kotlin, Scala e altri. È noto per la sua ricca serie di funzionalità progettate per migliorare l'efficienza degli sviluppatori e semplificare il processo di sviluppo del software. Il seguente IDE è stato utilizzato per la scrittura del codice in Java.

Maven

Maven è uno strumento di gestione delle build e delle dipendenze che fornisce un sistema di automazione per la compilazione, il packaging delle applicazioni e il download delle dipendenze_g.

All'interno di questo progetto Maven è stato utilizzato con Spring Boot per semplificare la gestione delle dipendenze e delle versioni.

DBeaver

DBeaver è un'applicazione di amministrazione di database universale e strumento client SQL. È utilizzato per connettersi, esplorare, gestire e interrogare diversi tipi di database.

All'interno del progetto è stato utilizzato per interagire col database.

Hibernate

Hibernate è un framework di mapping oggetto-relazionale (ORM_g). L'obiettivo principale di Hibernate è semplificare la gestione e l'accesso ai dati in un database relazionale utilizzando oggetti Java anziché scrivere query SQL manualmente.

Postman

Postman è un'applicazione di sviluppo di API (Application Programming Interface) che consente agli sviluppatori di creare, testare, documentare e monitorare le API. Nel corso del progetto è stato uno strumento altamente utilizzato sia come API testing tool.

JUnit

JUnit è un framework di testing per Java utilizzato per la scrittura e l'esecuzione di test d'unità. Questo framework fornisce un ambiente di testing in cui è possibile definire e strutturare test.

Questo framework offre funzionalità come annotazioni e vari metodi che semplificano il processo di scrittura di questi test, automatizzando la verifica che il codice soddisfi i requisiti e produca risultati attesi.

Mockito

Mockito è un framework di testing per Java che si concentra sulla creazione di oggetti simulati (mock) per testare unità di codice. Gli oggetti mock imitano il comportamento di oggetti reali, consentendo ai test di focalizzarsi su parti specifiche del codice, permettendo ai test di non dover interagire con database o sistemi esterni. Permette inoltre di verificare interazioni con i mock e molto altro, al fine di agevolare il processo di testing.

3.2.2 Strumenti di supporto

Gli strumenti di supporto sono utilizzati per attività che sostengono lo sviluppo del progetto. Essi contribuiscono a migliorare la gestione, la qualità e l'efficienza del processo di sviluppo.

Microsoft Teams

Microsoft Teams è una piattaforma di comunicazione e collaborazione aziendale sviluppata da Microsoft. Offre strumenti per la chat, le videoconferenze, la condivisione di documenti e la gestione dei progetti, consentendo ai team di lavorare insieme in modo efficace sia in ufficio che a distanza.

Questa piattaforma è stata utilizzata nel corso del progetto per poter comunicare con il tutor anche quando non era in ufficio o con altri colleghi per determinate situazioni lavorative.

Notion

Notion è un'applicazione di gestione delle informazioni, utilizzata per prendere appunti, creare elenchi di attività, scrivere documenti e molto altro grazie alla sua interfaccia flessibile personalizzabile dagli utenti.

Questa applicazione è stata utilizzata nel corso della mia attività di Stage per prendere appunti o tenere traccia delle tasks che dovevo svolgere.

Microsoft Excel

Microsoft Excel è un'applicazione software di fogli di calcolo sviluppata da Microsoft. È utilizzata per creare, organizzare e analizzare dati in forma di tavole e grafici, offrendo inoltre molte funzionalità per eseguire calcoli.

Questa applicazione è stata utilizzata nel progetto allo scopo di velocizzare la creazione

di dati fittizi per popolare le tabelle del database per poter testare quanto prodotto.

Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft. È progettato per fornire un ambiente di sviluppo leggero, flessibile e personalizzabile per programmatori e sviluppatori.

È stato utilizzato nel progetto scaricando varie estensioni per visualizzare l'API e l'UML del database.

Docker

Docker è una piattaforma di "containerizzazione" che consente di creare, distribuire e gestire container virtualizzati. Permette di far funzionare le applicazioni in altri ambienti con facilità.

Nel progetto è stato utilizzato per creare un server locale con un'immagine del database MSSQL.

Swagger

Swagger è un insieme di strumenti e specifiche che consentono la documentazione, la progettazione e il test di API. Permette una migliore comunicazione dei propri endpoint dell'API semplificandone la lettura e la comprensione.

Git

Git è un sistema di controllo versione distribuito utilizzato nello sviluppo software. Consente di tenere traccia delle modifiche apportate al codice sorgente e di semplificare la collaborazione tra sviluppatori in progetti di programmazione.

GitLab

GitLab è una piattaforma di sviluppo software basata su web che offre una serie di strumenti per la gestione del ciclo di vita dello sviluppo delle applicazioni. Le principali funzionalità sono: la gestione dei repository Git, la collaborazione tra i membri del team e il monitoraggio delle issue.

All'interno del progetto è stata utilizzata come repository per tenere salvato il progresso del progetto. Nella mia attività di stage ho lavorato su un branch a me dedicatomi.

Git Extensions

Git Extensions è un'interfaccia grafica per il sistema di controllo versione distribuito Git. Questo software fornisce una modalità visuale per interagire con i repository Git.

Capitolo 4

Progettazione

Successivamente all'apprendimento del background tecnologico, sono state improntate le basi dell'architettura del progetto, definendo lo schema del database su cui si poggia l'API e i concetti base del progetto.

Nei seguenti paragrafi verrà trattata l'architettura e le configurazioni utilizzate che funzionano da base per l'implementazione di quanto sviluppato.

4.1 Architettura REST

L'architettura REST, acronimo di “Representational State Transfer”, è un approccio di progettazione per la creazione di servizi web che si basa sui principi dell'HTTP_g (Hypertext Transfer Protocol).

Le principali caratteristiche di un'architettura REST includono:

- **Sistema client-server**, dove il client è chi fa le richieste e il server fornisce le risposte;
- **Sistema layered**, perchè possono essere composte da più livelli di servizi indipendenti;
- **Stateless**, significa che il server non contiene client state_g, quindi ogni richiesta ha abbastanza informazione per il server per processarla;
- **Cacheable**, significa che l'architettura può memorizzare le risposte dei server e riutilizzarle;
- **Resource-based**: questo approccio è considerato tale, dato che si concentra sull'identificazione e la gestione delle risorse all'interno di un sistema. Le risorse vengono identificate da degli URI_g ed esse possono essere create, aggiornate, richieste o eliminate (operazioni CRUD) attraverso operazioni HTTP standard (POST, PUT, GET, DELETE);
- **Manipolazione delle risorse**, poichè le risorse sono diverse dalla loro rappresentazione logica, utilizzando formati come JSON o XML.

Questo stile architetturale è utilizzato soprattutto per la realizzazione di API poichè l'adozione dei principi standard di HTTP mantiene un'interfaccia uniforme e l'approccio

resource-based ne semplifica la progettazione, dato che le risorse vengono identificate da URI.

4.2 Spring MVC

Nello sviluppo di una API REST tramite Spring Boot è comune l'utilizzo del pattern architetturale Model-View-Controller(MVC).

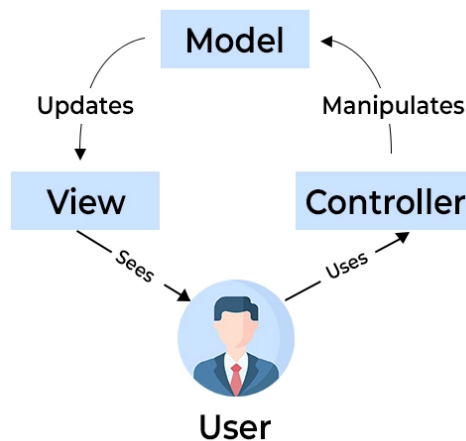


Figura 4.1: Schema Model-View-Controller

4.2.1 Model

Il Modello o Model in inglese, rappresenta i dati e i metodi per accedere ai dati dell'applicazione. Se esso viene aggiornato in seguito ad azioni o eventi, notificherà la View e il Controller del cambiamento.

Nel contesto dello sviluppo di un'API REST, il Model contiene dati che vengono elaborati e restituiti dall'API.

All'interno del progetto il Model è formato dai seguenti elementi.

Entità

Le entità sono risorse rappresentate con classi Java segnate a class level_g con l'annotazione_g `@Entity`. Questa annotazione viene utilizzata per identificare una classe che è mappata su una tabella nel database. All'interno di queste classi, vengono dichiarati gli attributi corrispondenti alla tabella di riferimento utilizzando annotazioni JPA appropriate, e vengono gestite le relazioni tra le tabelle attraverso altre annotazioni.

Repository

Le repository implementate nel progetto gestiscono l'accesso ai dati e definiscono metodi per eseguire operazioni di base sui dati, come l'inserimento, la modifica, la cancellazione e la ricerca. Tutto questo è stato possibile all'interno del progetto estendendo interfacce JPA, che consentono di eseguire operazioni in una base di dati senza scrivere codice SQL.

DTO

Data Transfer Object (DTO), oggetti utilizzati per modellare le rappresentazioni dei dati, utilizzati per definire sia i dati inviati dal client nelle richieste che quelli che inviati dal server al cliente nelle risposte.

4.2.2 View

La Vista o View in inglese, è responsabile di mostrare i dati provenienti dal Model e dell'interazione con l'utente. Essa cattura gli input dell'utente e delega al Controller l'elaborazione.

Dato che il progetto si concentrava sul lato back-end, precisamente sullo sviluppo dell'API REST, non ho quindi sviluppato una vera e propria View, poichè l'output è in formato JSON. In questo contesto l'output prodotto contenente i dati presentati al client in formato JSON, potrebbe essere considerata come "View".

4.2.3 Controller

Il Controller gestisce il flusso dell'applicazione. Esso riceve i comandi dell'utente attraverso la View e reagisce eseguendo operazioni conseguenti. Queste sue operazioni possono o meno coinvolgere il Model, ma portano generalmente sempre ad un cambiamento di stato della View.

All'interno del meccanismo di Spring MVC il Controller gestisce le richieste HTTP in arrivo, selezionando il Controller adeguato. Sono quindi responsabili di ricevere le richieste, elaborarle e restituire le risposte corrispondenti.

All'interno del progetto il Controller è formato dai seguenti elementi.

Controller

I Controller contengono i metodi a cui vengono associati i percorsi delle richieste HTTP. Esso interpreta i parametri che possono provenire dall'URL_g o dal corpo della richiesta. Internamente a questi metodi viene richiamato il Service appropriato per eseguire operazioni di business logic al risultato finale da restituire.

Per garantire che i dati inseriti dagli utenti o provenienti da richieste siano conformi alle aspettative e non causino errori vengono inseriti dei controlli di validazioni all'interno dei metodi del Controller.

Service

I Service eseguono operazioni complesse ed elaborazioni di dati, gestendo la business logic dell'applicazione. Essi interagiscono con i Repository per recuperare dati dal database e vengono richiamati all'interno dei metodi del Controller. Contengono ulteriori controlli di validazione per verificare la correttezza dei dati utilizzati.

Exception Handler

Per centralizzare la gestione delle eccezioni è stato creato un handler che garantisce uniformità alle eccezioni che l'applicazione può produrre.

4.3 Pattern Client-Server

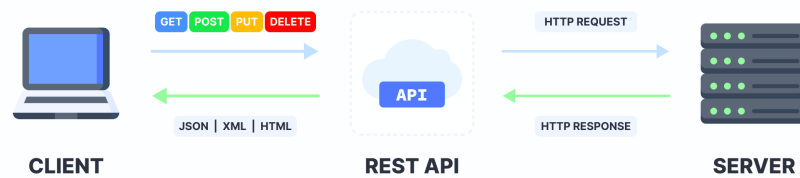


Figura 4.2: Client-Server in una REST API

È un pattern architetturale composto da due componenti: client e server. Il client rappresenta il richiedente del servizio, mentre il server fornisce i contenuti e i servizi ai client, restando in attesa delle loro richieste.

Il pattern client-server si adatta perfettamente ad una API REST e le sue componenti sono identificate in questo modo.

Client

In questo contesto il client è rappresentato da applicazioni o servizi che comunicano con l'API REST, inviando le richieste HTTP al server per ottenere quanto richiesto. Il client invia richieste utilizzando verbi standard HTTP per accedere alle risorse e alle funzionalità che l'API offre.

Server

Il server invece è colui che gestisce le richieste HTTP ricevute e genera le risposte in formato JSON. Esso quindi elabora le richieste ed interagisce col database per recuperare le informazioni.

4.4 Design Pattern

Nel prodotto sviluppato possiamo notare i seguenti Design Pattern. Ad eccezione dell'ultimo Design Pattern adottato, in riferimento all'elenco sottostante, i restanti Design Pattern sono già implementati da Spring e Spring Boot.

Repository pattern

Repository è un pattern ideato per dividere le operazioni di business logic da quelle di persistenza dei dati. Questo pattern fornisce astrazione ai dati nascondendo i dettagli di accesso, facilita il testing, supporta diverse sorgenti di dati e mantiene un codice riutilizzabile in quanto non sarà necessario modificare ampiamente il codice in caso di modifiche alla sorgente dati.

In Spring Boot vengono implementate interfacce annotandole con l'annotazione *@Repository* ed estendendole con interfacce JPA che permettono l'utilizzo di metodi che forniscono query basilari o la possibilità di creare metodi custom per query personalizzate.

Dependency Injection

Dependency Injection è un pattern che inietta una dipendenza in una classe senza sapere l'implementazione effettiva. Questo può avvenire attraverso constructor injection, setter injection o method injection. Gli obiettivi di questo pattern sono: eliminare il forte accoppiamento tra le classi, rendere il codice più manutenibile e più facile da testare. In Spring Boot risulta evidente tramite l'utilizzo di annotazioni come *@Autowired*, iniettando direttamente le dipendenze necessarie, promuovendo il concetto successivo di Inversion of Control;

Inversion of Control

Inversion of Control (IoC) è un design pattern nato sul concetto di invertire il controllo del flusso del sistema nella gestione delle dipendenze e del controllo interno di un'applicazione. Normalmente è l'applicazione che controlla le dipendenze o il flusso di esecuzione, ma questa responsabilità, utilizzando questo pattern, viene trasferita ad un framework. Il framework instanzia oggetti, inietta le dipendenze e coordinerà il flusso di esecuzione.

Spring Boot è costruito proprio su questo concetto chiave. Esso è correlato a IoC per i seguenti motivi:

- Component scan, esegue automaticamente uno scan delle classi individuando quelle contrassegnate con annotazioni come *@Service*, *@Repository*, *@Controller* e altre;
- Semplifica la gestione delle dipendenze, fornendo degli insiemi di dipendenze utili sotto il nome di "starters", velocizzando l'inizializzazione delle dipendenze senza doverle configurare manualmente;
- Dependency Injection;
- Application Context, agisce come contenitore per i bean dell'applicazione. Esso gestisce i loro cicli di vita e inietta le dipendenze nei punti appropriati.

Front Controller

Il Front Controller è un design pattern che permette di centralizzare la gestione delle richieste all'interno di un'applicazione. In Spring MVC è implementato dal framework per gestire le richieste HTTP verso il Controller adeguato.

Data Transfer Object

Data Transfer Object (DTO) è un pattern utilizzato per gestire il trasferimento dei dati tra client e server. Gli obiettivi principali del pattern sono:

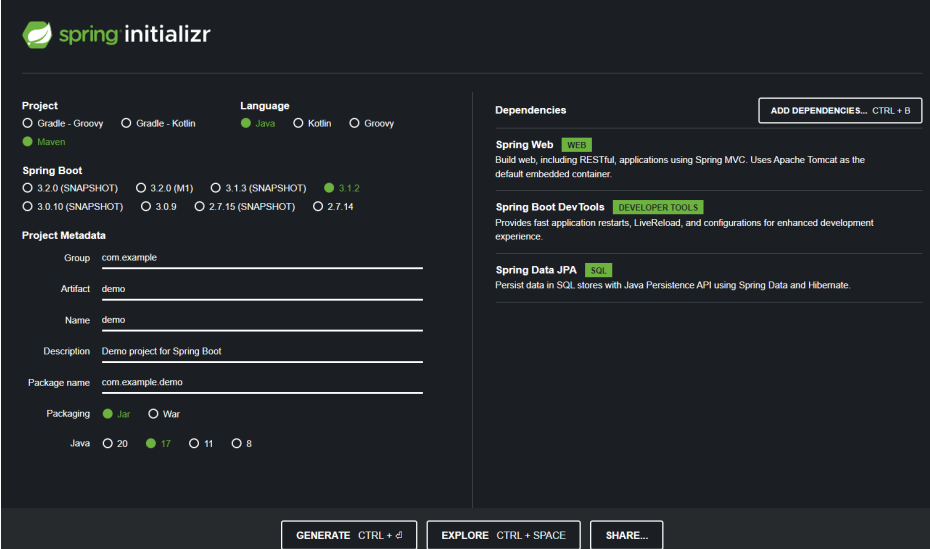
- Sicurezza, in quanto puoi controllare cosa si invia;
- Flessibilità, puoi adattare i DTO in base alle esigenze dell'API;

- Separano la rappresentazione interna da quella esterna dei dati.

Nel contesto di una API REST, i DTO vengono utilizzati sia nella Request che nella Response. Vengono utilizzati in entrambi i punti perchè quando il client invia dei dati magari non necessita degli oggetti completi ma solo di alcune informazioni. Invece quando il server restituisce dei dati il client se li può aspettare sotto una determinata struttura.

4.5 Configurazione iniziale del progetto

4.5.1 Spring Initializr



The screenshot shows the Spring Initializr web interface. It is divided into several sections:
1. **Project**: Radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven (selected).
2. **Language**: Radio buttons for Java (selected), Kotlin, and Groovy.
3. **Spring Boot**: Radio buttons for various versions, with 3.1.2 (selected) highlighted in green.
4. **Project Metadata**: Text input fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo).
5. **Packaging**: Radio buttons for Jar (selected) and War.
6. **Java**: Radio buttons for versions 20, 17 (selected), 11, and 8.
7. **Dependencies**: A section on the right with a button 'ADD DEPENDENCIES... CTRL + B'. It lists 'Spring Web' (WEB), 'Spring Boot Dev Tools' (DEVELOPER TOOLS), and 'Spring Data JPA' (SQL).
8. **Bottom Bar**: Three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

Figura 4.3: Interfaccia Spring Initializr

Spring Initializr è uno strumento online fornito dalla community di Spring Framework, che consente di creare rapidamente un progetto Spring Boot personalizzato, con le dipendenze e le configurazioni preselezionate dall'utente. Questo strumento semplifica notevolmente il processo di inizializzazione di un progetto Spring Boot, permettendo agli sviluppatori di risparmiare tempo e concentrarsi sulla scrittura del codice.

L'utente può selezionare il tipo di progetto di cui ha bisogno, come ad esempio un progetto Maven o Gradle, e specificare il linguaggio e la versione di Spring Boot desiderati. Inoltre, può anche inserire i metadati del progetto, come il nome del progetto e il nome dei packages.

Una volta selezionate le opzioni desiderate, l'utente può scegliere le dipendenze per il progetto. Le dipendenze sono librerie di terze parti che forniscono funzionalità aggiuntive al progetto.

Dopo aver selezionato le dipendenze, l'utente può scaricare il progetto Spring Boot personalizzato in formato ZIP.

4.5.2 Contenuto del pacchetto

Il file ZIP scaricato precedentemente contiene tutti i file necessari per iniziare a lavorare sul progetto. All'interno di questo pacchetto troviamo i seguenti elementi rilevanti:

- file di configurazione *application.properties* e file di build *pom.xml* con le dipendenze selezionate;
- classe principale, rappresenta il punto di ingresso dell'applicazione ed è annotata con *@SpringBootApplication*. Il metodo *main()* al suo interno avvierà l'applicazione Spring;
- una struttura base dei package.

4.5.2.1 Pom.xml

Di seguito riporto un frammento di codice del file di build *pom.xml* che si ottiene creando un progetto con le dipendenze sopra selezionate. La configurazione di Maven avviene proprio tramite questo file.

All'interno di questo file si possono trovare i seguenti tag:

- *<dependencies>*, indica una lista di dipendenze;
- *<build>*, contiene impostazioni di costruzione e compilazione;
- *<plugins>*, contiene plugin di Maven;
- *<properties>*, contiene proprietà definite dall'utente;
- *<groupId>*, organizzazione che ha creato il progetto;
- *<artifactId>*, nome unico del progetto;
- *<version>*, versione del progetto.

Nel file si possono notare le seguenti dipendenze:

- *spring-boot-starter-web*, per utilizzare il framework Spring MVC per la creazione di applicazioni Web;
- *spring-boot-starter-jpa*, per la persistenza dei dati;
- *spring-boot-devtools*, offre tools per migliorare il processo di sviluppo;
- *spring-boot-starter-test*, per includere librerie di testing.

```
<properties>
  <java.version>17</java.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Figura 4.4: Snippet pom.xml generato

Rispetto al file utilizzato nel progetto mancano le seguenti dipendenze:

- *spring-boot-starter-jdbc*, per avviare Spring con il supporto JDBC nel progetto;
- *mssql-jdbc*, contenente il driver JDBC di Microsoft SQL Server;
- *springdoc-openapi-starter-webmvc-ui*, dipendenza per l'integrazione di Springdoc OpenAPI con Spring Boot per generare la documentazione dell'API utilizzando l'interfaccia utente WebMvc UI;
- *poi* e *poi-ooxml*, due dipendenze relative ad Apache POI che offre funzionalità per poter lavorare con documenti Microsoft Office;
- *jxls-jexcel*, dipendenza che include la libreria jXLS per creare e manipolare documenti Excel;
- *fastexcel* e *fastexcel-reader*, altre dipendenze inerenti a file Excel che offrono funzionalità per lettura, scrittura e modifica di questi file.

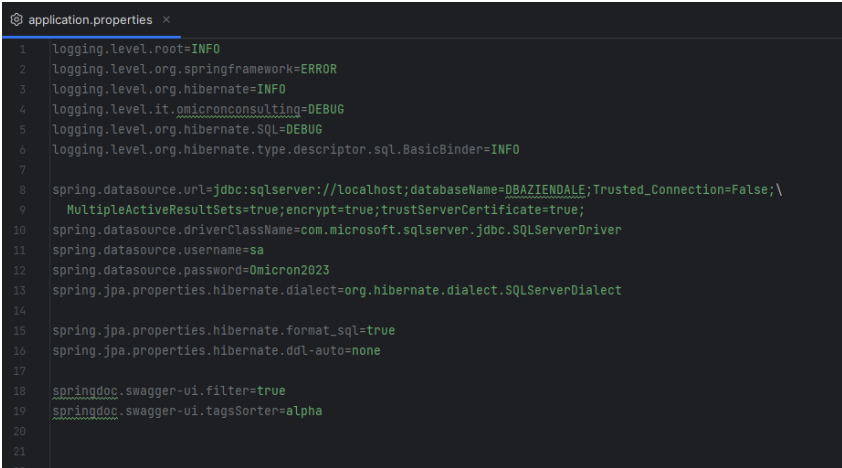
Per il testing sono state aggiunte:

- *junit*, framework usato in Java per scrivere test unitari;

- *hamcrest-library*, libreria utile a scrivere asserzioni più espressive e comprensibili nei test unitari;
- *h2*, si tratta della dipendenza per la libreria H2 Database Engine, che è un database SQL. Utilizzato con scope "test" per crearne un'istanza temporanea e utilizzarlo nei test.

4.5.2.2 Application.properties

Il secondo file qui sotto rappresentato è il file configurazione *application.properties*. Di seguito riporto il file che ho utilizzato nel progetto:

A screenshot of a code editor showing the contents of the application.properties file. The file is titled 'application.properties' in the tab. The code is as follows:

```
1 logging.level.root=INFO
2 logging.level.org.springframework=ERROR
3 logging.level.org.hibernate=INFO
4 logging.level.it.omiconconsulting=DEBUG
5 logging.level.org.hibernate.SQL=DEBUG
6 logging.level.org.hibernate.type.descriptor.sql.BasicBinder=INFO
7
8 spring.datasource.url=jdbc:sqlserver://localhost;databaseName=DBAZIENDALE;Trusted_Connection=False;\
9   MultipleActiveResultSets=true;encrypt=true;trustServerCertificate=true;
10 spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
11 spring.datasource.username=sa
12 spring.datasource.password=Omicron2023
13 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDialect
14
15 spring.jpa.properties.hibernate.format_sql=true
16 spring.jpa.properties.hibernate.ddl-auto=none
17
18 springdoc.swagger-ui.filter=true
19 springdoc.swagger-ui.tagsSorter=alpha
20
21
```

Figura 4.5: Foto application.properties configurato e utilizzato

Possiamo notare come il file utilizzi un formato di configurazione basato su chiavi e valori. Le chiavi rappresentano le diverse proprietà di configurazione dell'applicazione, mentre i valori rappresentano le impostazioni specifiche.

Nel file possiamo notare le seguenti chiavi:

- *logging.level*, servono per configurare il livello di dettaglio dei log per diverse classi o package all'interno dell'applicazione;
- *spring.datasource*, servono per collegarsi ad un database, in questo caso locale, inserendo username e password e driver di MSSQL;
- *spring.jpa.properties.hibernate*, servono per configurare le impostazioni di Hibernate tra cui la validazione schema-entità, il dialetto_g del server SQL e il suo livello di log;
- *springdoc.swagger-ui*, libreria che fornisce integrazione tra Spring Boot e Swagger UI.

Capitolo 5

Modello dati

Questo capitolo descrive il database utilizzato all'interno del progetto. Verrà descritto inizialmente la configurazione di Docker, passando per la struttura del database e la gestione dei log nel database.

5.1 Configurazione di Docker

I container Docker offrono un'isolamento completo dell'ambiente, che aiuta a evitare conflitti di dipendenze e interferenze con altre applicazioni o servizi che potrebbero essere presenti sul sistema. Per questo motivo si è deciso di utilizzare l'approccio di sandboxing che offre Docker, poichè l'utilizzo di container consente di isolare le applicazioni e i servizi in ambienti virtualizzati, condividendo il kernel del sistema operativo host ma separando le loro risorse e i loro processi.

Dato che le tabelle di mia creazione dovevano integrarsi con il database aziendale, tramite il tool Docker Compose e un file di configurazione *docker-compose.yaml*, è stato possibile avviare un nuovo container contenente un'immagine di Microsoft SQL Server, che forniva un backup del database aziendale con dati e tabelle.

5.2 Progettazione del database

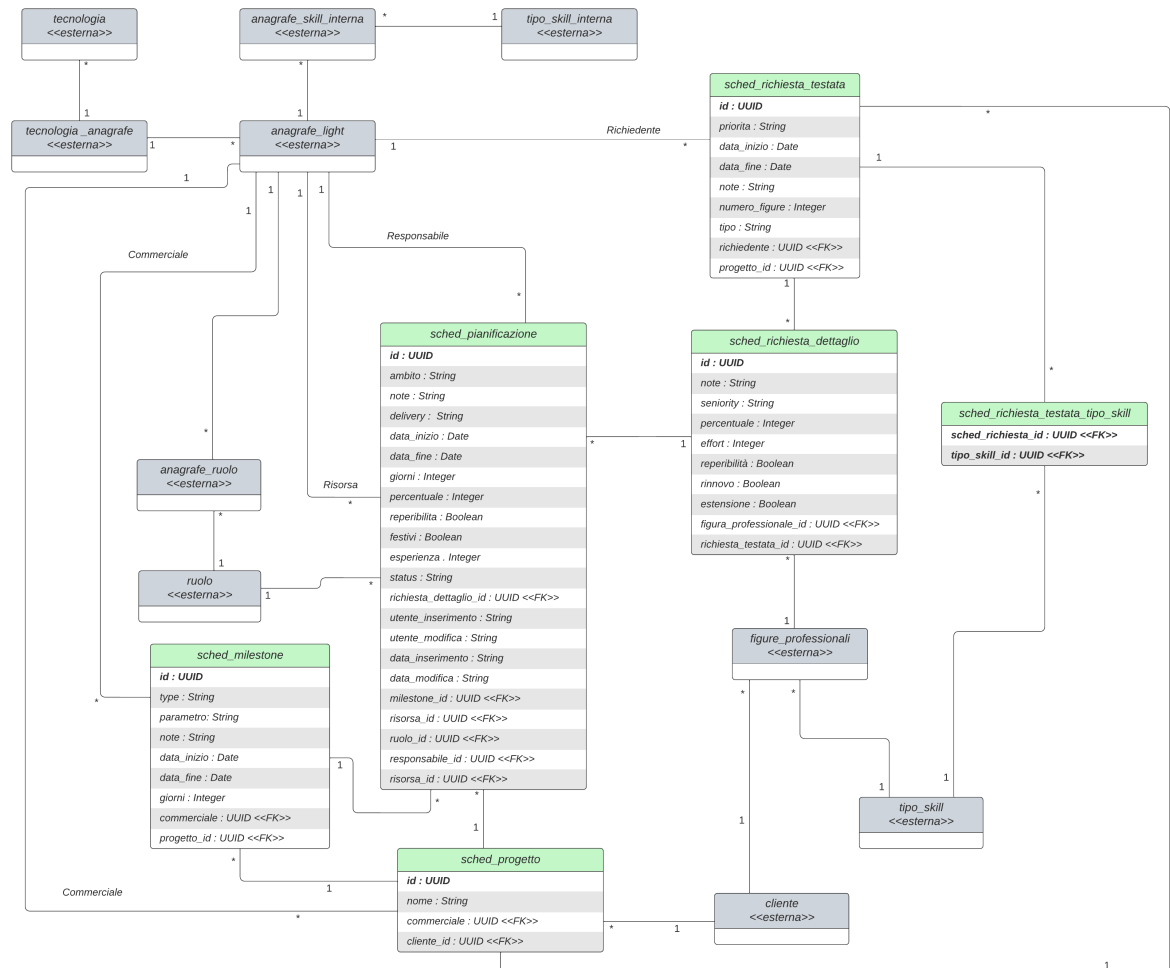


Figura 5.1: Schema Database

Nel seguente disegno si pu  notare il database su cui poggia l'API.

  stata inserita una nomenclatura «esterna» per indicare le tabelle provenienti dal database aziendale. Per ogni tabella creata da me   stato deciso di utilizzare il prefisso "sched_".

  stato utilizzato un UUID_g come chiave primaria per ogni tabella.

Di seguito elenco l'utilizzo delle tabelle create da me e quelle del database aziendale, non trattando per  gli attributi all'interno di quest'ultime:

tecnologia «esterna»

Tabella del database aziendale utilizzata per mostrare le skill che una Risorsa possiede.

tecnologia_anagrafe «esterna»

Tabella di join del database aziendale tra la tabella contenente le anagrafiche aziendali e la tabella contenente le skill possedute da una Risorsa.

tipo_skill_interna «esterna»

Tabella del database aziendale utilizzata per fornire le Aree di Competenza di ogni Risorsa.

anagrafe_skill_interna «esterna»

Tabella di join del database aziendale tra la tabella *tipo_skill_interna* e la tabella contenente le anagrafiche aziendali.

anagrafe_light «esterna»

Tabella del database aziendale contenente i dati anagrafici di una Risorsa. All'interno del Database viene utilizzata questa tabella per prendere le informazioni in merito a: Richiedente della Richiesta, Responsabile della Pianificazione, Risorsa associata alla Pianificazione e Commerciale.

ruolo «esterna»

Tabella del database aziendale contenente i ruoli possibili associati ad una Risorsa. Rappresenta il Ruolo che impiegherà nella Pianificazione.

anagrafe_ruolo «esterna»

Tabella di join del database aziendale tra *anagrafe_light* e *ruolo*.

cliente «esterna»

Tabella del database aziendale contenente i dati dei Clienti associabili ad ai Progetti.

sched_progetto

Tabella contenente i Progetti a cui una Richiesta o una Pianificazione può essere associata.

Attributo	Descrizione
Id	UUID univoco «PK»
Nome	Nome del progetto
Commerciale	UUID del Commerciale associato «FK»

Cliente_Id	UUID del Cliente associato «FK»
------------	---------------------------------

Tabella 5.1: Tabella sched_progetto**sched_richiesta_testata**

Tabella creata per gestire le Richieste create. Quando un Project Manager effettua una Richiesta creerà una nuova riga in questa tabella e nella tabella *sched_richiesta_dettaglio* per ogni Figura Professionale richiesta.

Attributo	Descrizione
Id	UUID univoco «PK»
Priorità	Attributo che rappresenta la priorità (Alta,Media,Bassa)
Data_inizio	Specifica la data di inizio per l'incarico richiesto
Data_fine	Specifica la data di fine per l'incarico richiesto
Note	Note inseribili dal Richiedente
Numero_figure	Specifica il numero di figure richieste
Tipo	Specifica il tipo di attività richiesta
Richiedente	UUID del Richiedente della Richiesta «FK»
Progetto_Id	UUID del Progetto associato «FK»

Tabella 5.2: Tabella sched_richiesta_testata**sched_richiesta_testata_tipo_skill**

Tabella di join creata tra *sched_richiesta_testata* e *tipo_skill*.

È stata creata per rappresentare la relazioni multi-a-molti tra le due tabelle.

Attributo	Descrizione
sched_richiesta_id	Id della Richiesta associata «PK» «FK»
tipo_skill_id	Id della Skill richiesta associata «PK» «FK»

Tabella 5.3: Tabella sched_richiesta_testata_tipo_skill**tipo_skill «esterna»**

Tabella del database aziendale contenente skill conosciute dalle Figure Professionali.

sched_richiesta_dettaglio

Tabella creata per contenere i dettagli di ogni Figura richiesta. Ogni Figura richiesta corrisponde ad una nuova riga nella tabella.

È risultato obbligatorio utilizzare questo approccio per gestire la varietà di parametri assegnati per ogni figura richiesta dal Richiedente.

Attributo	Descrizione
Id	UUID univoco «PK»
Note	Note inseribili per Figura
Seniority	Tipo di Seniority per ogni Figura (Junior, Middle, Senior)
Percentuale	Percentuale di impegno previsto per l'incarico
Effort	Quantità di impegno
Reperibilità	Booleano per indicare la reperibilità di una Figura
Rinnovo	Booleano per indicare il rinnovo dell'impegno della risorsa oltre la data indicata
Estensione	Booleano per che si riferisce all'estensione del lavoro sull'attività corrente

Figura_professionale_id	UUID della Figura Professionale associata «FK»
Richiesta_testata_id	UUID della Richiesta principale a cui è associata «FK»

Tabella 5.4: Tabella sched_richiesta_dettaglio**figure_professionali «esterna»**

Tabella del database aziendale che contiene le Figure Professionali associabili ad una determinata skill richiesta.

sched_pianificazione

Tabella creata per contenere le Pianificazioni. Quando il Program Manager prende in carico una Richiesta posta dal Project Manager, per ogni Figura richiesta, ne crea una Pianificazione associando una Risorsa libera adeguata.

Attributo	Descrizione
Id	UUID univoco «PK»
Ambito	Ambito di assegnazione della Pianificazione
Note	Note inseribili per la Pianificazione
Delivery	Valore che può essere Delivery Core o Delivery Digital
Data_inizio	Data di inizio della Pianificazione
Data_fine	Data di fine della Pianificazione
Giorni	Giorni di durata della Pianificazione
Percentuale	Indica la percentuale di impegno effettivo
Reperibilità	Booleano per indicare la reperibilità effettiva di una Figura

Festivi	Booleano per indicare l'opzione delle festività per una Figura
Esperienza	Intero rappresentante gli anni di esperienza
Status	Indica lo status della Pianificazione
Utente_inserimento	Utente che ha inserito la Pianificazione
Utente_modifica	Ultimo utente che ha modificato la Pianificazione
Data_inserimento	Data di inserimento della Pianificazione
Data_modifica	Ultima data di modifica della Pianificazione
Richiesta_dettaglio_id	UUID della Richiesta dettaglio associato «FK»
Milestone_id	UUID della Milestone associata «FK»
Progetto_id	UUID del Progetto associato «FK»
Ruolo_id	UUID del Ruolo pianificato «FK»
Responsabile_id	UUID del Responsabile associato «FK»
Risorsa_id	UUID della Risorsa pianificata «FK»

Tabella 5.5: Tabella sched_pianificazione**sched_milestone**

Tabella contenente i milestone associabili alla tabella delle Pianificazioni.

Una Milestone non può essere associata ad una Pianificazione se quest'ultima non è associata ad un Progetto.

Attributo	Descrizione
Id	UUID univoco «PK»
Type	Indica il tipo di Milestone
Parametro	Specifica un'opzione personalizzata della Milestone
Note	Note inseribili per la Milestone
Data_inizio	Data di inizio della Milestone
Data_fine	Data di fine della Milestone
Giorni	Numero dei giorni di durata della Milestone
Commerciale	UUID del Commerciale associato «FK»
Progetto_id	UUID del Progetto associato «FK»

Tabella 5.6: Tabella sched_milestone

5.3 Gestione dei log nel sistema

Una tabella di log in un database è una struttura dati che registra le attività che si verificano su una tabella del database. I log sono utili per le seguenti finalità:

- Sicurezza, possono essere utilizzati per monitorare le attività e gli accessi nel database;
- Risoluzione dei problemi, possono aiutare gli amministratori del database a capire la radice del problema, facilitando la risoluzione di bug;
- Recupero dati, ritornano utili in caso di perdita di dati o di necessità a riprendere dati cancellati o modificati;

Per soddisfare il requisito [RFDE-65](#) è stata creata una tabella di log per le Pianificazioni. Per limiti di tempo non è stata creata e gestita anche l'altra tabella di log delle Richieste come scritto nel requisito [RFDE-46](#).

5.3.1 Tabella PianificazioniAudit

La tabella di log per le Pianificazioni contiene tutti gli attributi di Pianificazioni, ma duplicati. Si distinguono in attributi "old", facendo riferimento ai valori della Pianificazione prima dell'operazione e "new", facendo riferimento ai valori della Pianificazione dopo l'operazione. La chiave primaria della tabella è una chiave composta da: Id della Pianificazione e timestamp dell'attività. Si possono notare anche gli attributi "Utente", per tenere traccia di chi ha effettuato la modifica e l'attributo "Operazione", per indicare se si tratta di un'operazione di cancellazione, inserimento o modifica.

Il popolamento di questa tabella avviene tramite un trigger_g che si avvia non appena c'è un inserimento, un update o una delete di una riga nella tabella delle Pianificazioni.

```
CREATE TRIGGER sched_pianificazione_audit_insert_update_delete
ON sched_pianificazione
FOR INSERT,UPDATE,DELETE

AS DECLARE @operation VARCHAR(20),@PianUtenteIns_ins VARCHAR(36), @PianUtenteMod_ins VARCHAR(36),
@PianUtenteIns_del VARCHAR(36),@PianUtenteMod_del VARCHAR(36),@PianDataIns_ins DATETIME,
@PianDataMod_ins DATETIME,@PianDataIns_del DATETIME,@PianDataMod_del DATETIME,
@PianId_ins VARCHAR(36),@PianId_del VARCHAR(36),@PianAmbito_ins VARCHAR(20),
@PianAmbito_del VARCHAR(20),@PianNote_ins VARCHAR(255),@PianNote_del VARCHAR(255),
@PianDelivery_ins VARCHAR(20),@PianDelivery_del VARCHAR(20),@PianDataInizio_ins DATE,
@PianDataInizio_del DATE,@PianDataFine_ins DATE,@PianDataFine_del DATE,
@PianGiorni_ins INT,@PianGiorni_del INT,@PianPercentuale_ins INT,
@PianPercentuale_del INT,@PianReperibilita_ins BIT,@PianReperibilita_del BIT,
@PianFestivi_ins BIT,@PianFestivi_del BIT,@PianEsperienza_ins INT,
@PianEsperienza_del INT,@PianStatus_ins VARCHAR(20),@PianStatus_del VARCHAR(20),
@PianRichiestDett_ins VARCHAR(36),@PianRichiestDett_del VARCHAR(36),
@PianMilestone_ins VARCHAR(36),@PianMilestone_del VARCHAR(36),
@PianProgetto_ins VARCHAR(36),@PianProgetto_del VARCHAR(36),
@PianRisorsa_ins VARCHAR(36),@PianRisorsa_del VARCHAR(36),
@PianResponsabile_ins VARCHAR(36),@PianResponsabile_del VARCHAR(36),
@PianRuolo_ins VARCHAR(36),@PianRuolo_del VARCHAR(36)

SET @operation = 'INSERIMENTO' -- Di base Inserimento

SELECT @PianId_ins = ins.Id,@PianAmbito_ins = ins.ambito,@PianNote_ins = ins.note,@PianDelivery_ins = ins.delivery,
@PianDataInizio_ins = ins.data_inizio,@PianDataFine_ins = ins.data_fine,@PianGiorni_ins = ins.giorni,@PianPercentuale_ins = ins.percentuale,
@PianReperibilita_ins = ins.reperibilita,@PianFestivi_ins = ins.festivi,@PianEsperienza_ins = ins.esperienza,@PianStatus_ins = ins.status,
@PianRichiestDett_ins = ins.richiesta_dettaglio_id,@PianMilestone_ins = ins.milestone_id,@PianProgetto_ins = ins.progetto_id,
@PianRisorsa_ins = ins.risorsa_id,@PianResponsabile_ins = ins.responsabile_id,@PianRuolo_ins = ins.ruolo_id,
@PianUtenteIns_ins = ins.utente_inserimento, @PianUtenteMod_ins = ins.utente_modifica, @PianDataIns_ins = ins.data_inserimento,
@PianDataMod_ins = ins.data_modifica FROM INSERTED ins;

SELECT @PianId_del = del.Id,@PianAmbito_del = del.ambito,@PianNote_del = del.note,@PianDelivery_del = del.delivery,
@PianDataInizio_del = del.data_inizio,@PianDataFine_del = del.data_fine,@PianGiorni_del = del.giorni,@PianPercentuale_del = del.percentuale,
@PianReperibilita_del = del.reperibilita,@PianFestivi_del = del.festivi,@PianEsperienza_del = del.esperienza,
@PianRichiestDett_del = del.richiesta_dettaglio_id,@PianMilestone_del = del.milestone_id,
@PianProgetto_del = del.progetto_id,@PianRisorsa_del = del.risorsa_id,@PianResponsabile_del = del.responsabile_id,@PianRuolo_del = del.ruolo_id,
@PianUtenteIns_del = del.utente_inserimento, @PianUtenteMod_del = del.utente_modifica, @PianDataIns_del = del.data_inserimento,
@PianDataMod_del = del.data_modifica FROM DELETED del;

IF EXISTS(SELECT TOP 1 1 FROM DELETED) AND NOT EXISTS(SELECT TOP 1 1 FROM INSERTED)
SET @operation = 'CANCELLAZIONE'

IF EXISTS(SELECT TOP 1 1 FROM DELETED) AND EXISTS(SELECT TOP 1 1 FROM INSERTED)
SET @operation = 'MODIFICA'
```

Figura 5.2: Codice Trigger: Inizializzazione variabili

```

BEGIN
INSERT INTO sched_pianificazione_audit
(id,
[timestamp],
operazione,
old_utente_inserimento,
new_utente_inserimento,
old_utente_modifica,
new_utente_modifica,
old_data_modifica,
new_data_modifica,
old_data_inserimento,
new_data_inserimento,
new_ambito,
old_ambito,
new_note,
old_note,
new_delivery,
old_delivery,
new_datainizio,
old_datainizio,
new_datafine,
old_datafine,
new_giorni,
old_giorni,
new_percentuale,
old_percentuale,
new_reperibilita,
old_reperibilita,
new_festivi,
old_festivi,
new_esperienza,
old_esperienza,
new_status,
old_status,
new_richiesta_dettaglio_id,
old_richiesta_dettaglio_id,
VALUES(
coalesce(@PianId_ins,@PianId_del),
GETDATE(),
@operazione,
@PianUtenteIns_del,
@PianUtenteIns_ins,
@PianUtenteMod_del,
@PianUtenteMod_ins,
@PianDataMod_del,
@PianDataMod_ins,
@PianDataIns_del,
@PianDataIns_ins,
@PianAmbito_del,
@PianAmbito_ins,
@PianNote_del,
@PianNote_ins,
@PianDelivery_del,
@PianDelivery_ins,
@PianDataInizio_del,
@PianDataInizio_ins,
@PianDataFine_del,
@PianDataFine_ins,
@PianGiorni_del,
@PianGiorni_ins,
@PianPercentuale_del,
@PianPercentuale_ins,
@PianReperibilita_del,
@PianReperibilita_ins,
@PianFestivi_del,
@PianFestivi_ins,
@PianEsperienza_del,
@PianEsperienza_ins,
@PianStatus_del,
@PianStatus_ins,
@PianRichiestDett_del,
@PianRichiestDett_ins,
@PianMilestone_del,
@PianMilestone_ins,
@PianProgetto_del,
@PianProgetto_ins,
@PianRisorsa_del,
@PianRisorsa_ins,
@PianResponsabile_del,
@PianResponsabile_ins,
@PianRuolo_del);
END

```

Figura 5.3: Codice Trigger: Inserimento dei valori nella tabella di log

Dopo aver definito le variabili, identificato l'operazione effettuata tramite un controllo, vengono eseguiti gli inserimenti nei campi appositi all'interno della tabella.

Capitolo 6

Sviluppo API REST

Questo capitolo fornisce una spiegazione dettagliata dell'API che è stata sviluppata. Comprende informazioni sulle convenzioni di denominazione degli endpoint, i verbi HTTP utilizzati, una lista completa di tutti gli endpoint sviluppati e una descrizione del file Swagger che ha sostituito il mock iniziale dell'API.

6.1 Convenzioni di denominazione REST

Buona prassi nella creazione di un'API REST è il rispetto di convenzioni per la nomenclatura degli endpoint. Sebbene non esista un'unica convenzione obbligatoria, esistono delle best practices per garantire una facile lettura e comprensibilità per agevolare gli sviluppatori che adoperano l'API.

Le seguenti convenzioni sono state rispettate nello sviluppo dell'API:

- pluralizzare le risorse, per distinguere se si fa riferimento ad una lista di una determinata risorsa o ad una singola risorsa;
- usare lettere minuscole;
- non usare estensioni dei file;
- in caso di nomi composti utilizzare il "-";
- non usare underscore;
- non utilizzare abbreviazioni o slang;
- non utilizzare verbi, poichè l'azione dovrebbe essere indicata dal metodo HTTP utilizzato.

6.2 Verbi standard HTTP

I verbi standard forniti da HTTP utilizzati applicati all'API REST sono i seguenti:

Verbo	Descrizione
POST	Utilizzato per inviare dati al server al fine di creare una nuova risorsa ed aggiungerla all'insieme corrente
GET	Utilizzato per richiedere dati al server in merito ad una o più risorse senza modificarle
DELETE	Utilizzato per eliminare una risorsa specifica dal server
PUT	Utilizzato per aggiornare totalmente una risorsa sovrascrivendola
PATCH	Utilizzato per effettuare aggiornamenti parziali a una risorsa esistente

Tabella 6.1: Verbi Standard HTTP utilizzati

6.3 Endpoint sviluppati

In questa sezione si trovano le descrizioni di tutti gli endpoint_g implementati, suddivisi in base all'ambito di interesse. Sarà fornito anche il verbo HTTP e il percorso necessario per effettuare ciascuna richiesta.

Anagrafiche

Questi endpoint vengono utilizzati dal Program Manager per effettuare operazioni di lettura sulle anagrafiche aziendali.

Verbo	Path	Descrizione
GET	/area-competenza/{id}	Permette la lettura di un'Area di Competenza dato l'ID
GET	/area-competenza/{id}	Permette la lettura di tutte le Aree di Competenza

GET	/clienti	Permette la lettura da DB di tutti i Clienti ottenendo come risposta la lista di tutti i Clienti
GET	/clienti/{id}	Permette la lettura da DB di un Cliente dato l'Id ottenendo come risposta il Cliente corrispondente
GET	/risorse/{id}	Permette la lettura da DB di un Risorsa dato l'Id ottenendo come risposta la Risorsa corrispondente
GET	/ruoli	Permette la lettura da DB di tutti i Ruoli ottenendo come risposta la lista di tutti i Ruoli
GET	/ruoli/{id}	Permette la lettura da DB di un Ruolo dato l'Id ottenendo come risposta il Ruolo corrispondente
GET	/skill	Permette la lettura da DB di tutte le Skill ottenendo come risposta la lista di tutte le Skill
GET	/skill/{id}	Permette la lettura da DB di una Skill dato l'Id ottenendo come risposta la Skill corrispondente

Tabella 6.2: Endpoint Anagrafiche sviluppati

Milestones

Questi endpoint consentono la creazione, lettura, modifica ed eliminazione di Milestones. Vengono utilizzati principalmente dal Program Manager.

Verbo	Path	Descrizione
POST	/milestones	Permette l'inserimento di una nuova Milestone, inserendo il Progetto associato, il Commerciale, la Pianificazione e altri dettagli.
POST	/milestones/list	Rappresenta una POST di ricerca. Restituisce una lista di Milestone in base ai filtri inseriti dall'utente, alla parola chiave inserita nella quicksearch (q) e a un booleano che determina se applicare tutti i filtri in modo congiuntivo (AND) o disgiuntivo (OR).
PATCH	/milestones/{id}/type	Permette la modifica del campo Type di una Milestone su DB.
GET	/milestones/{id}	Permette la lettura da DB di una Milestone dato l'Id ottenendo come risposta la Milestone corrispondente.
DELETE	/milestones/{id}	Permette l'eliminazione da DB di una Milestone dato l'Id ottenendo come risposta la Milestone eliminata.

Tabella 6.3: Endpoint Milestones sviluppati

Pianificazioni

Questi endpoint consentono la creazione, lettura, modifica ed eliminazione di Pianificazioni. Vengono utilizzati principalmente dal Program Manager.

Verbo	Path	Descrizione
GET	/pianificazioni/{id}	Permette la lettura da DB di una Pianificazione dato l'Id ottenendo come risposta la Pianificazione corrispondente.

PUT	/pianificazioni/{id}	Permette la modifica parziale o totale di campi di una Pianificazione su DB sovrascrivendola.
DELETE	/pianificazioni/{id}	Permette l'eliminazione da DB di una Pianificazione dato l'Id ottenendo come risposta la Pianificazione eliminata.
POST	/pianificazioni	Permette l'inserimento di una nuova Pianificazione ed eventuali entità associate come: risorsa, ruolo, milestone, progetto, responsabile e figura.
POST	/pianificazioni/xlsx	Permette l'esportazione di report Excel di Pianificazioni in base a filtri inseriti dall'utente.
POST	/pianificazioni/list	Rappresenta una POST di ricerca. Restituisce una lista di Pianificazioni in base ai filtri inseriti dall'utente, alla parola chiave inserita nella quick-search (q) e a un booleano che determina se applicare tutti i filtri in modo congiuntivo (AND) o disgiuntivo (OR).
PATCH	/pianificazioni/{id}/times	Permette la modifica dei campi Data Inizio e Data Fine di una Pianificazione su DB.
PATCH	/pianificazioni/{id}/festivi	Permette la modifica del campo Festivi di una Pianificazione su DB.
GET	/pianificazioni/{id}/audit	Permette la lettura delle modifiche effettuate su una Pianificazione dalla tabella di log.

Tabella 6.4: Endpoint Pianificazioni sviluppati

Richieste

Questi endpoint consentono la creazione, lettura, modifica ed eliminazione di Richieste di Pianificazioni. Vengono utilizzati principalmente dal Project Manager.

Verbo	Path	Descrizione
GET	/richieste/{id}	Permette la lettura da DB di una Richiesta dato l'Id ottenendo come risposta la Richiesta corrispondente.
PUT	/richieste/{id}	Permette la modifica parziale o totale di campi di una Richiesta su DB sovrascrivendola.
DELETE	/richieste/{id}	Permette l'eliminazione da DB di una Richiesta dato l'Id ottenendo come risposta la Richiesta eliminata.
POST	/richieste	Permette l'inserimento di una nuova Richiesta, inserendo le Figure Professionali e le Skill richieste e altri dettagli.
POST	/richieste/list	Rappresenta una POST di ricerca. Restituisce una lista di Richieste in base ai filtri inseriti dall'utente, alla parola chiave inserita nella quicksearch (q) e a un booleano che determina se applicare tutti i filtri in modo congiuntivo (AND) o disgiuntivo (OR).
DELETE	/richieste/{id}/stato	Permette la modifica del campo Stato di una Richiesta su DB.
DELETE	/richieste/{id}/priorita	Permette la modifica del campo Priorità di una Richiesta su DB.

Tabella 6.5: Endpoint Richieste sviluppati

6.4 Swagger

Swagger è un framework utilizzato per la documentazione, progettazione e gestione di API REST. Per garantire ordine sul mio lavoro e una comunicazione più trasparente e chiara con chi lavora nel lato Front-End è stato creato questo documento utilizzando lo standard OpenAPI.

Questo disegno dell'API ha sostituito poi il mock iniziale dell'API, considerando questo nuovo documento come unico per la consultazione dell'API finale.

API

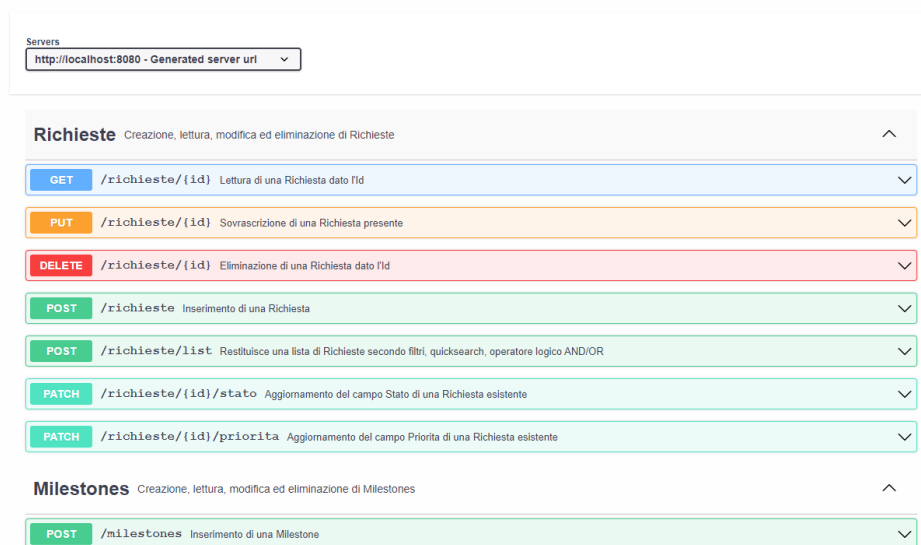


Figura 6.1: Inizio del documento di definizione dell'API

Il documento rappresenta un disegno completo dell'implementazione dell'API REST. Contiene tutti gli endpoint sviluppati, come sono strutturati al loro interno e arricchito di descrizioni per endpoint, risposte, richieste ed errori.

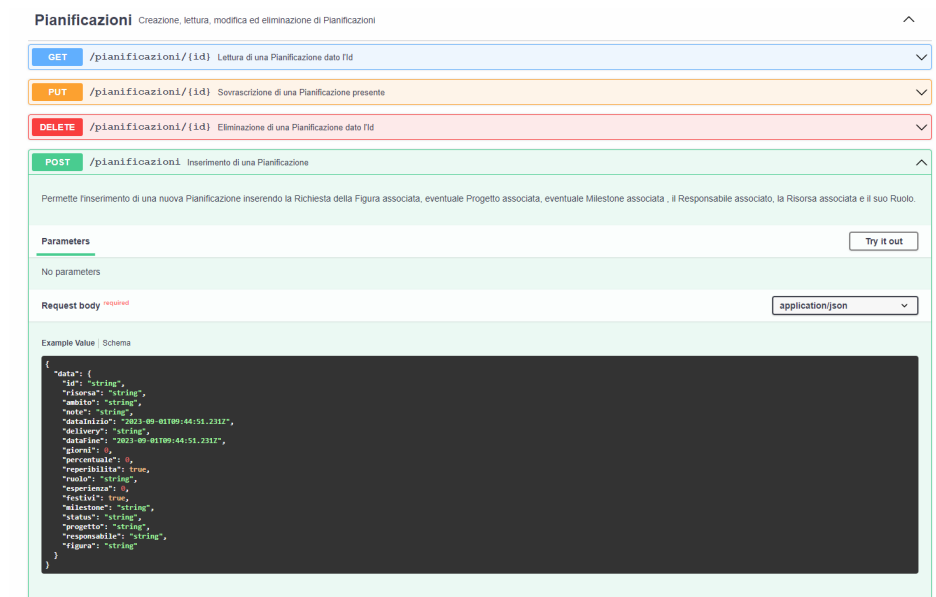


Figura 6.2: Example value del body richiesto dall'endpoint

In questa immagine possiamo notare un "Example value" del body richiesto da parte dell'endpoint POST /pianificazioni.

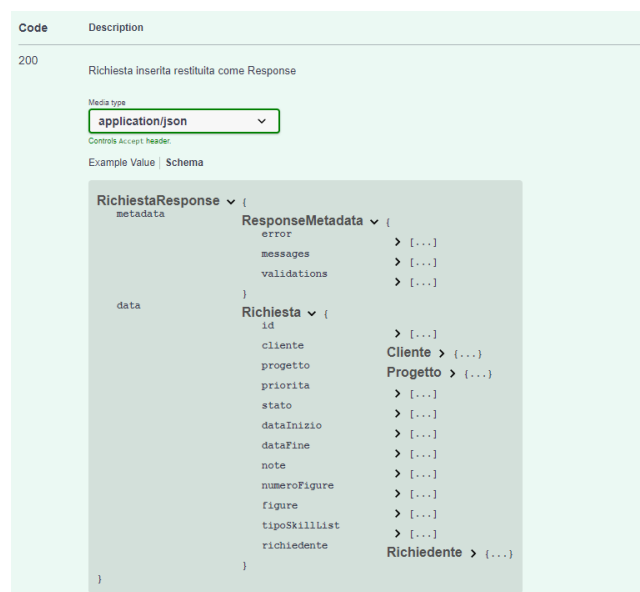


Figura 6.3: Composizione oggetto response ritornato dall'endpoint

Inoltre, è possibile esaminare la struttura di ciascun oggetto, sia nelle richieste che nelle risposte. Questo è particolarmente vantaggioso per gli sviluppatori Front-End, poiché consente loro di interagire in modo accurato con il Back-End. In questo modo, il

Back-End indica al Front-End cosa e come ci si aspetta di ricevere per quanto riguarda i dati nelle richieste, mentre il Front-End viene informato su come e cosa riceverà dal Back-End nelle risposte.

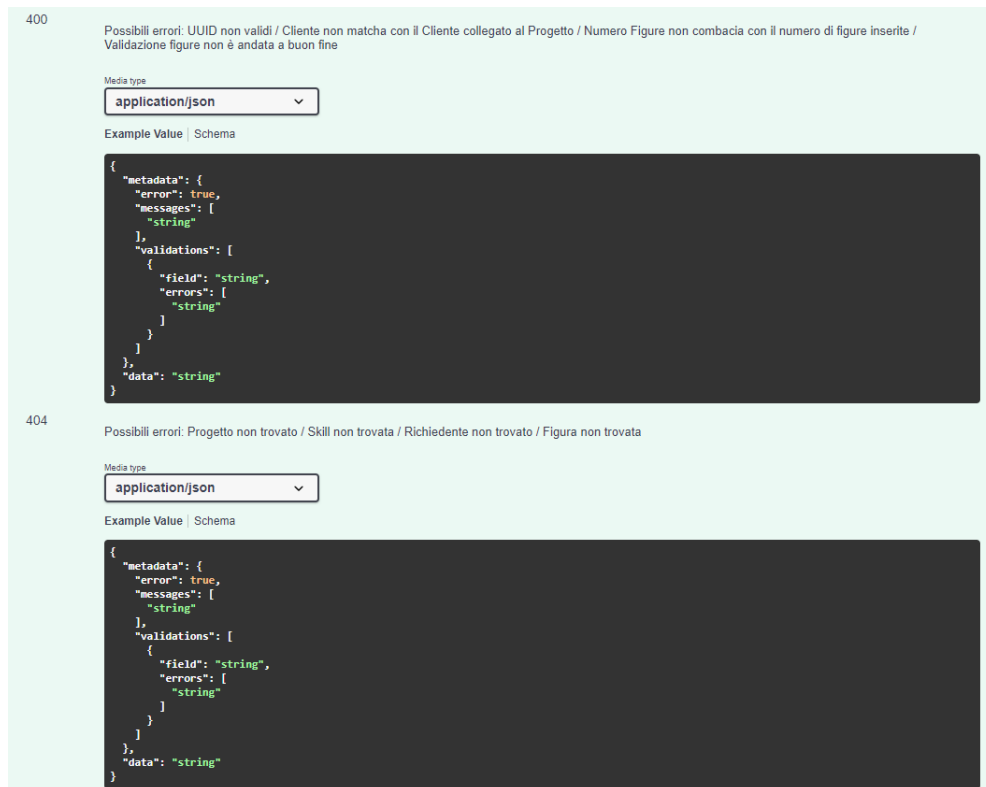


Figura 6.4: Errori possibili in un endpoint

Per garantire una comprensione chiara dei possibili errori che potrebbero verificarsi, vengono identificate e descritte varie situazioni, ciascuna associata a un codice di errore specifico, al fine di fornire una struttura generale per codice di errore.

Capitolo 7

Codifica

In questo capitolo viene illustrato come sono stati organizzati i package_g e le classi al loro interno, la spiegazione di frammenti di codice rilevanti ed infine una sezione per la creazione dello Swagger in maniera automatica.

7.1 Packages Common e config

I package contengono i seguenti file .java:

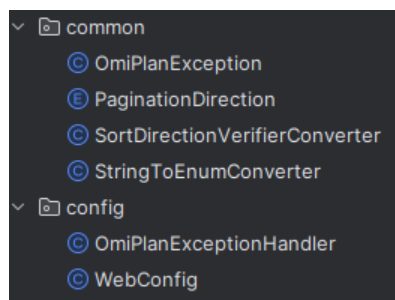


Figura 7.1: Package common e config

7.1.1 Common

- **OmiPlanException**, eccezione personalizzata utilizzata per gestire le *RuntimeException*. Classe formata da un `HttpStatus` per mostrare lo stato di risposta HTTP e dal messaggio fornito al lancio dell'eccezione;
- **PaginationDirection**, classe enumerativa per gestire la direzione della paginazione_g (ASC o DESC);
- **SortDirectionVerifierConverter**, classe che implementa l'interfaccia *Converter<S,T>* (componente Java utilizzato quando si lavora con strutture dati o oggetti che devono essere trasformati o adattati in tipi diversi) per verificare che la direzione inserita sia ASC o DESC e gestire la *RunTimeException* in caso non sia una variabile enum;

- **StringToEnumConverter**, classe che implementa l'interfaccia *Converter<S,T>* per convertire la direzione Stringa inserita nell'enum della direzione di pagina.

7.1.2 Config

- **OmiPlanExceptionHandler**, handler con il compito di gestire le eccezioni runtime, segnando quelle non gestite come "Unexpected error". Questa classe è annotata con *@ControllerAdvice*, poiché definisce una classe che gestisce in maniera centralizzata le eccezioni. Contiene due metodi annotati con *@ExceptionHandler* che gestiscono rispettivamente le *RunTimeException* e le altre eccezioni *Exception*;
- **WebConfig**, classe annotata con l'annotazione *@Configuration* indicando che è una classe di configurazione e che contiene definizioni di bean o altre configurazioni necessarie per l'applicazione. Essa infatti estende *WebMvcConfigurer* che consente di modificare le configurazioni predefinite di Spring MVC, in questo caso aggiungendo i converter sopra citati.

7.2 Package Entities

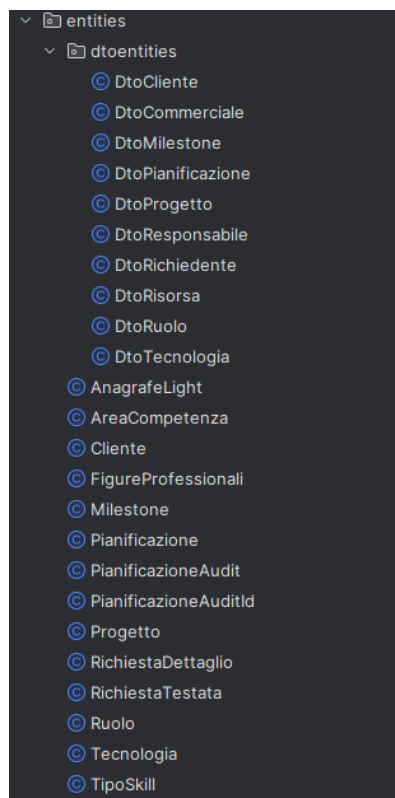


Figura 7.2: Package entità

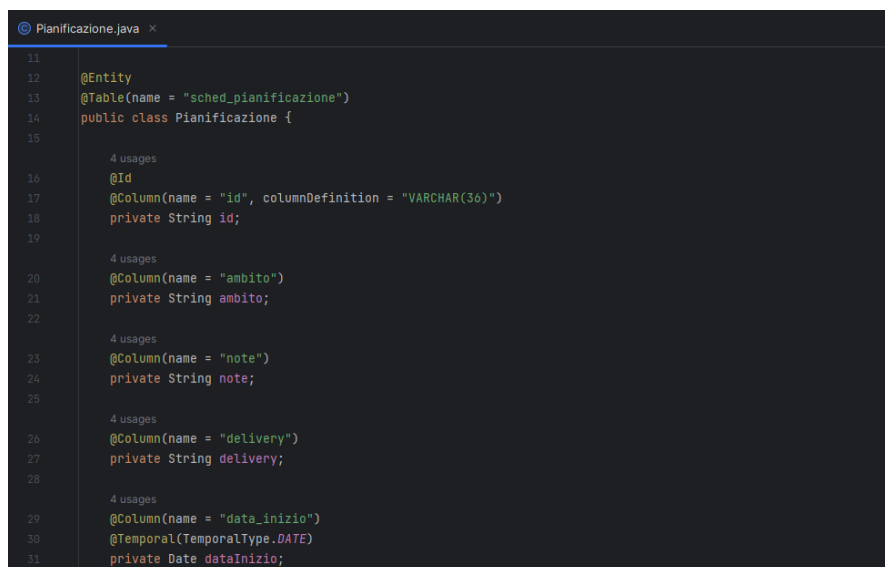
All'interno del package Entities troviamo le entità. Per entità si intendono tutte quelle classi Java che definiscono i modelli di dati dell'applicazione.

Queste classi vengono annotate con annotazioni JPA utili a stabilire come la classe venga associata a una tabella presente nel database relazionale. Ogni istanza di un'entità rappresenta una riga nella tabella relativa.

Di tutte le tabelle da me create è stata mappata ogni relazione tra tabelle e campo, mentre per ogni tabella del database aziendale sono stati mappati tutti i campi e le relazioni ad altre tabelle che potevano tornarmi utili.

In ogni entità troviamo setters e getters, costruttori con e senza argomenti.

7.2.1 Esempio di un'entità



```
11
12 @Entity
13 @Table(name = "sched_pianificazione")
14 public class Pianificazione {
15
16     4 usages
17     @Id
18     @Column(name = "id", columnDefinition = "VARCHAR(36)")
19     private String id;
20
21     4 usages
22     @Column(name = "ambito")
23     private String ambito;
24
25     4 usages
26     @Column(name = "note")
27     private String note;
28
29     4 usages
30     @Column(name = "delivery")
31     private String delivery;
32
33     4 usages
34     @Column(name = "data_inizio")
35     @Temporal(TemporalType.DATE)
36     private Date dataInizio;
```

Figura 7.3: Esempio di mappatura di un'entità

Nell'immagine qui sopra possiamo notare un frammento di codice dell'entità Pianificazione in cui sono state utilizzate le seguenti annotazioni:

- *@Entity*, per mappare le classi Java che rappresentano una tabella in un database, si inserisce questa notazione specificandola a class level.
- *@Table*, nella maggior parte dei casi il nome di una tabella nel database e il nome dell'entità non sono gli stessi. Per questo motivo è stata utilizzata la seguente annotazione per specificare il nome della tabella;
- *@Column*, utilizzata per mappare una campo di una classe a una colonna di una tabella nel database. Questa annotazione ha dei parametri, come ad esempio "name", che serve a specificare il nome della colonna a cui è associato il campo;
- *@Id*, per identificare un campo all'interno di una classe che mappa la chiave primaria di una tabella del database viene utilizzata questa annotazione;
- *@Temporal*, utilizzata per specificare se un campo di tipo Date dovrebbe essere mappato come *TemporalType.DATE*, per una data senza orario, *TemporalTy-*

pe.TIME per un orario senza data e infine *TemporalType.TIMESTAMP*, utilizzato nella tabella di log di Pianificazione per mappare un campo con data e orario.

7.2.2 Mappatura delle relazioni

```

4 usages
@ManyToOne
@JoinColumn(name = "progetto_id")
private Progetto progetto;

4 usages
@OneToMany(mappedBy = "richiestaTestata", orphanRemoval = true)
private List<RichiestaDettaglio> dettaglioList;

3 usages
@ManyToOne
@JoinColumn(name = "richiedente_id")
private AnagrafeLight richiedente;

3 usages
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name = "sched_richiesta_testata_tipo_skill",
    joinColumns = @JoinColumn(name = "sched_richiesta_testata_id"),
    inverseJoinColumns = @JoinColumn(name = "tipo_skill_id"))
private List<TipoSkill> skills;

```

Figura 7.4: Esempio di mappatura delle relazioni della tabella *RichiestaTestata*

Per mantenere le relazioni tra le tabelle sono state utilizzate annotazioni che si possono osservare nel frammento di codice qui sopra raffigurante una porzione della classe Java che mappa la tabella *RichiestaTestata*.

Nel caso in cui sia necessario che entrambi i lati di una relazione debbano accedere all'informazione, viene dichiarata una relazione bidirezionale. In questo contesto, è fondamentale gestire correttamente la sincronizzazione tra le due entità.

- **uno-a-molti**, relazione in cui il campo associato sarà una lista di oggetti della classe opposta (in relazione all'esempio, una *RichiestaTestata* è associata a più *RichiesteDettaglio*), mentre nella relazione molti-a-uno sarà un oggetto singolo della classe opposta annotato con *@JoinColumn* con "name" uguale a quello del campo nella tabella del database (in relazione all'esempio, una *RichiestaTestata* ha un solo *Richiedente*).

Nell'esempio è stato utilizzato il parametro *mappedBy* per mappare la relazione opposta e *orphanRemoval* per poter implementare la cancellazione di una *RichiestaTestata* garantendo l'eliminazione di tutte le *RichiesteDettaglio* associate al momento della rimozione.

- **uno-a-uno**, relazione in cui un record della tabella è associato ad un solo record dell'altra tabella. Anche in questo caso se si vuole mappare da entrambi le parti la relazione bisogna utilizzare lo stesso principio dichiarato nella relazione precedente, solo che entrambe le parti avranno come campo un oggetto della classe opposta.

Non sono state identificate relazioni uno-a-uno nel database.

- **multi-a-molti**, relazione in cui viene mappata la tabella di join presente nel database tra le due tabelle, utilizzando il codice che si vede in esempio. È stata identificata solo una relazione multi-a-molti tra `RichiestaTestata` e `TipoSkill`. Per gestire operazioni di rimozione tra le due tabelle è stato utilizzato un metodo annotato con `@PreRemove` nella classe `TipoSkill`, che entra in azione quando una `RichiestaTestata` viene eliminata, assicurando che la disconnessione avvenga in modo appropriato.

7.2.3 Subpackage dtoentities

All'interno del package "dtoentities" troviamo tutte quelle classi DTO delle entità di cui servivano soltanto determinate informazioni da restituire all'utente, evitando così ridondanza o informazioni superflue. Queste classi non contengono annotazioni, ma sono semplicemente delle classi contenenti i campi semplificati delle entità. A loro volta possono contenere altri DTO di altre entità in base alle relazioni che hanno.

```
public class DtoPianificazione {
    5 usages
    private String id;

    5 usages
    private String ambito;

    5 usages
    private String note;

    5 usages
    private String delivery;

    5 usages
    private DtoRisorsa risorsa;

    5 usages
    private DtoResponsabile responsabile;

    5 usages
    private Date dataInizio;

    5 usages
    private Date dataFine;

    5 usages
    private Integer giorni;

    5 usages
    private Integer percentuale;

    public DtoPianificazione(Pianificazione p){
        this.id=p.getId();
        this.ambito=p.getAmbito();
        this.note=p.getNote();
        this.delivery=p.getDelivery();
        this.dataInizio =p.getDataInizio();
        this.dataFine =p.getDataFine();
        this.giorni=p.getGiorni();
        this.percentuale=p.getPercentuale();
        this.reperibilita=p.isReperibilita();

        if(p.getRuolo() != null){
            this.ruolo=new DtoRuolo(p.getRuolo());
        }

        this.festivi=p.isFestivi();
        this.esperienza=p.getEsperienza();
        this.status=p.getStatus();

        this.risorsa =new DtoRisorsa(p.getRisorsa());
        this.responsabile =new DtoResponsabile(p.getResponsabile());

        this.dataInserimento = p.getDataInserimento();
        this.dataModifica = p.getDataModifica();

        this.utenteInserimento = p.getUteInserimento();
        this.utenteModifica = p.getUteModifica();
    }
}
```

Figura 7.5: Esempio di classe DTO di Pianificazione

Ogni entità DTO possiede tre costruttori: costruttore senza argomenti e con argomenti e infine un costruttore che velocizzava la conversione da entità ad entità DTO (come mostrato nell'immagine qui sopra).

7.3 Package dto

Per la composizione delle richieste che il client invia o delle risposte che il server manda, sono state create vari tipi di requests e responses.

Ogni oggetto di risposta è composto da un oggetto "data" e un oggetto "metadata", mentre gli oggetti di richiesta possono variare. Se la richiesta è costruita per un endpoint che restituisce una lista di risultati, possiamo avere due casistiche:

- corpo della richiesta (body) formato da un oggetto "data", contenente informazioni utili alla risposta, e "metadata" che contiene i dati di paginazione;
- nessun corpo, ma soltanto i dati di paginazione passati come parametri di query_g.

Se invece la richiesta è costruita per un endpoint che restituisce un singolo oggetto nell'oggetto data, non viene utilizzato alcun body o parametri di query, ma solo una path variable.

Questa suddivisione "data" e "metadata" è un modello comune nella progettazione API per separare i dati effettivi o dati che forniscono informazioni per ottenere un tipo di risposta, dalle informazioni aggiuntive che descrivono il risultato o aggiungono caratteristiche alla richiesta.

Questo package è formato da vari subpackage.

7.3.1 Common

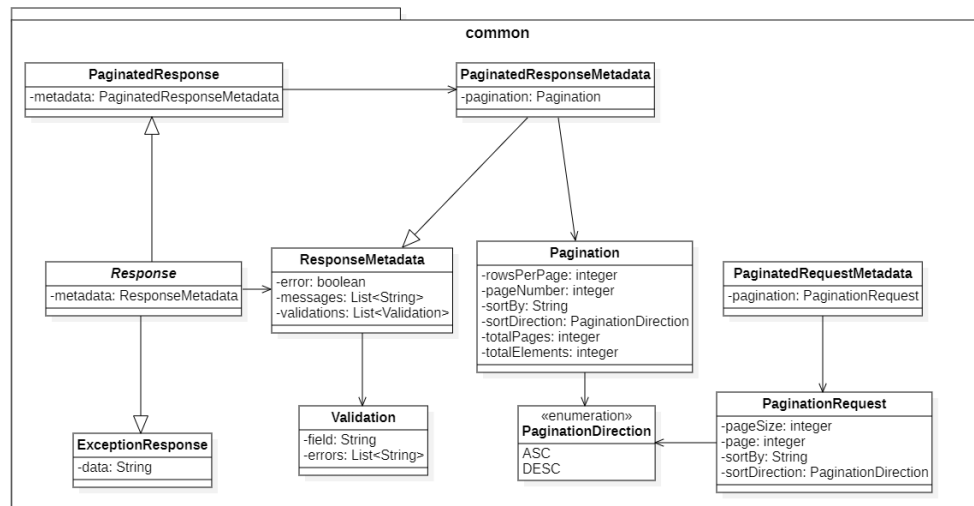


Figura 7.6: Subpackage common

Questo subpackage contiene la response di base, response paginata, la response utilizzata nell'eccezione personalizzata, una request per la paginazione e i vari metadata utilizzati nelle responses.

Ogni **Metadata** contiene un boolean che indica se si è andati in errore (true) o meno (false), il messaggio di errore e una lista di **Validation**, contenente informazioni di validazione dei dati.

Ogni **Response** verrà poi estesa dalle classi che costituiscono le risposte ritornate per entità nel subpackage Responses.

È stata inoltre creata una classe **Pagination** per avere un controllo personalizzato sulla Paginazione.

7.3.2 Requests e Body

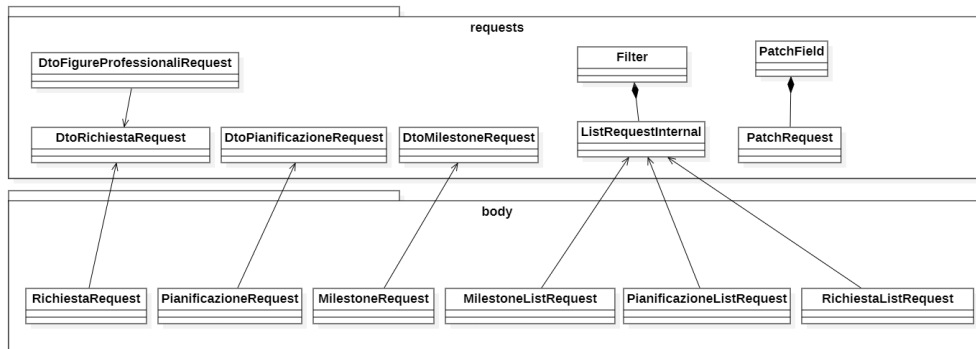


Figura 7.7: Subpackage requests e body

Questi due subpackages sono correlati tra di loro dato che ogni body ha come attributo una request presa dal subpackage requests. Esistono due tipi di body:

- ogni **Request** singola è utilizzata per richiedere una specifica risorsa e contiene una request DTO formata dai campi utili che l'utente deve inserire;
- ogni **ListRequest** è utilizzata per richiedere una o più risorse ed è formata da un oggetto data di tipo **ListRequestInternal**, che contiene tre parametri:
 - filters, lista di oggetti **Filter** formati da campo-valore;
 - una stringa q (quicksearch) utile nella query personalizzata per cercare in determinati campi quello che l'utente inserisce;
 - andOperator per impostare i filtri in And o in Or nella query.

All'interno del subpackage requests troviamo anche **PatchRequest**, contenente un unico campo corrispondente ad una lista di **PatchField**. Questa richiesta viene utilizzata nella richieste PATCH, in cui è possibile inserire uno o più valori in base a cosa si va a modificare.

7.3.3 Responses

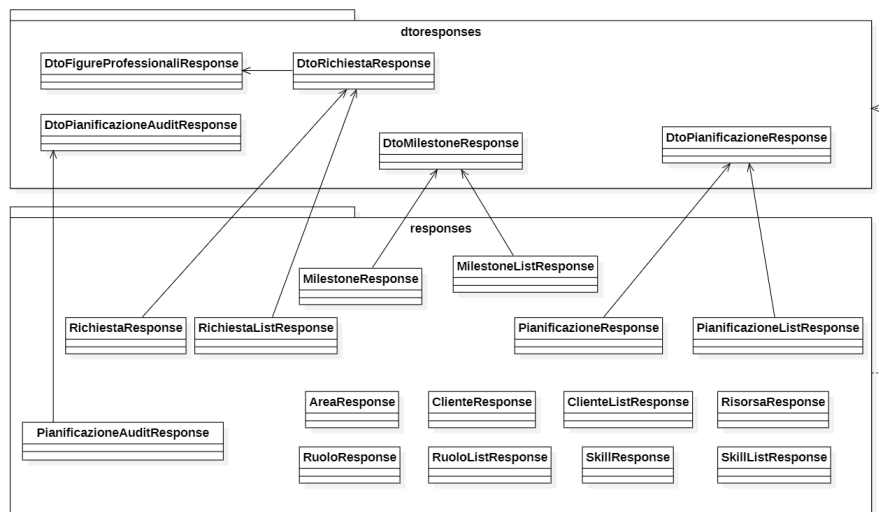


Figura 7.8: Subpackage responses e dtoresponses

Il package responses contiene tutte le response per entità. Ogni response estende la classe astratta **Response** se è una risposta singola, altrimenti estende la classe **PaginatedResponse** se è una lista di risposte. Questo permette all'utente di visualizzare la risposta, dopo aver interagito con un'endpoint, formata da "data" e "metadata". Ogni response contiene un solo attributo che corrisponde all'oggetto "data" e può essere:

- una response DTO del subpackage dtoresponses;
- un'entità DTO del subpackage delle entità;
- in caso di una **ListResponse** l'attributo sarà una lista di response.

7.4 Repository-Service-Controller

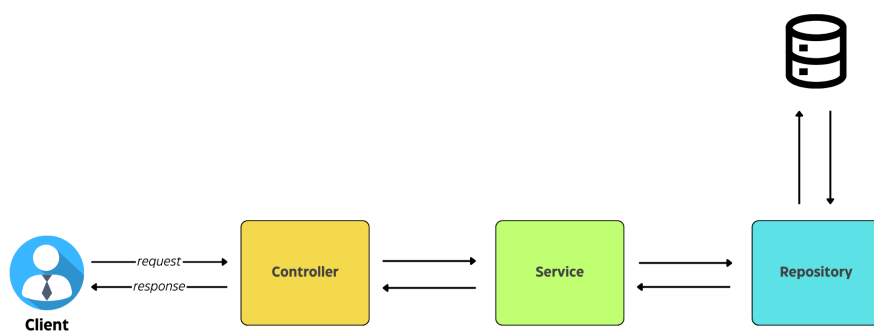


Figura 7.9: Schema di collegamento tra i componenti Controller, Service e Repository

In un'applicazione Spring Boot che implementa una REST API l'interazione tra questi tre componenti è fondamentale per gestire le richieste HTTP, elaborare la business logic e interagire con il database.

Come descritto nell'immagine l'interazione tra i componenti segue questo flusso:

1. il client invia una richiesta HTTP e arriva al Controller tramite URL;
2. il controller estrae i dati della richiesta, effettua controlli di validazione e chiama il metodo del service appropriato;
3. il metodo del service invocato conterrà la business logic per eseguire la richiesta e ulteriori controlli di validazione;
4. le repository coinvolte ottengono i dati richiesti dal database che vengono restituiti al service che li rielaborerà, restituendo a sua volta il risultato al controller;
5. il controller crea una risposta appropriata in formato JSON e la restituisce al client.

7.4.1 Controller

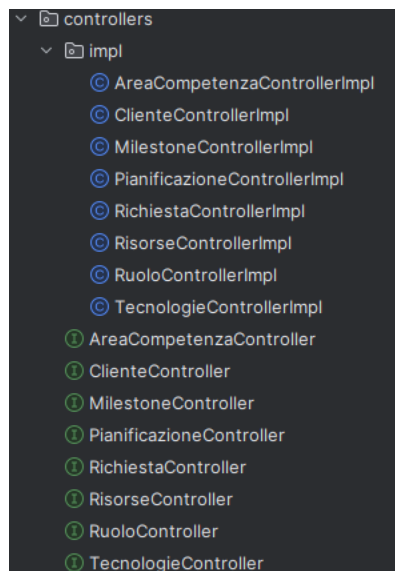


Figura 7.10: Controller sviluppati

Il Controller è punto di instradamento per le richieste HTTP. Riceve le richieste dal client, estrae i dati dai parametri della richiesta o dal corpo (body), esegue appropriati controlli di validazione ed infine inoltra le operazioni al Service. I controlli di validazione comprendono: validazione degli ID, validazione del formato delle date inserite, validazione che un dato obbligatorio non sia nullo e controllo che i filtri inseriti siano non nulli e valorizzati.

Ogni Controller è formato dalla sua interfaccia, contenente la firma dei metodi associati agli endpoint, e dalla sua implementazione. Ognuno di essi è annotato con `@RestController`.

7.4.1.1 Configurazione di un controller

```
@PostMapping("/milestones/list")
MilestoneListResponse getMoreMilestone(@RequestBody MilestoneListRequest request);
```

Figura 7.11: Firma di un metodo nell'interfaccia di un Controller

All'interno dell'interfaccia del Controller troviamo tutti i metodi associati agli endpoint. Per identificare a quale endpoint appartiene vengono utilizzate annotazioni come: `@PostMapping`, `@GetMapping`, `@DeleteMapping`, `@PutMapping` e `@PatchMapping`. Tramite l'annotazione `@RequestBody` Spring deserializza automaticamente il JSON in un tipo Java, in questo caso `MilestoneListRequest`. Come libreria standard utilizzata per mappare i campi JSON inseriti nella request dall'utente in un oggetto Java, viene utilizzata la libreria standard Jackson.

```
if(dtoMilestoneRequest.getProgetto() != null) {
    try {
        UUID.fromString(dtoMilestoneRequest.getProgetto());
    } catch (IllegalArgumentException e) {
        throw new OmiPlanException(HttpStatus.BAD_REQUEST,
            String.format("Sintassi errata del Progetto Id: %s",
                dtoMilestoneRequest.getProgetto()));
    }
}

return schedMilestoneService.postMilestone(milestoneRequest);
```

Figura 7.12: Esempio di controllo di validazione

Nelle classi `ControllerImpl` troviamo l'implementazione dei metodi dell'interfaccia, controlli di validazione che possono lanciare eccezioni e il metodo del Service utilizzato. In questo code snippet vediamo il controllo di validazione su l'Id del Progetto associato ad una Milestone. Se passerà il controllo, allora verrà inoltrata la richiesta al Service.

7.4.1.2 Generazione automatica dello Swagger

Per creare la documentazione dell'API è stata utilizzata la dipendenza `springdoc-openapi-starter-webmvc-ui` citata nella sotto-sezione 4.7.2.1 nel capitolo di Progettazione. Grazie a questa dipendenza è stato possibile generare automaticamente la documentazione dell'API basata sulle annotazioni presenti nel codice sorgente. SpringDoc API utilizza le annotazioni di Spring per identificare i Controller, i metodi delle API, i dati in input e in output. Per visualizzare questo documento generato è possibile accederci tramite un endpoint specifico: https://localhost:porta_scelta/swagger-ui.html.

```

@PostMapping("/milestones/list")
@Operation(summary = "Restituisce una lista di Milestone secondo filtri, quicksearch, booleano AND/OR",
description = "Restituisce una lista di Milestone in base a: filtri inseriti dall'utente, parola chiave" +
" nella quicksearch, booleano in cui si decide se applicare tutti i filtri (AND) o che ne valga almeno uno (OR)"
)
@ApiResponses(value={
    @ApiResponse(responseCode = "200",
        description = "Lista delle Milestone restituite come Response",
        content = {
            @Content(mediaType = "application/json", schema =
                @Schema(implementation = MilestoneListResponse.class)) }
        ),
    @ApiResponse(responseCode = "400",
        description = "Possibili errori: UUID non validi / Valore del filtro nullo o vuoto / Numero di elementi elevato",
        content = {
            @Content(mediaType = "application/json", schema =
                @Schema(implementation = ExceptionResponse.class)) }
        ),
    @ApiResponse(responseCode = "500",
        description = "Internal Server Error",
        content = {
            @Content(mediaType = "application/json", schema =
                @Schema(implementation = ExceptionResponse.class)) }
        )
})
MilestoneListResponse getMoreMilestone(@RequestBody MilestoneListRequest request);

```

Figura 7.13: Annotazioni utili a generare la documentazione per l'API

La dipendenza importata fornisce anche annotazioni per arricchire il documento. In questa immagine notiamo le seguenti annotazioni:

- *@Operation*, specifica un sommario dell'endpoint che l'utente può vedere prima di visualizzare nella sua totalità l'endpoint e un campo descrizione per fornire una descrizione più approfondita sul funzionamento;
- *@ApiResponses*, formata da tanti *@ApiResponse* che specificano il codice di errore, la descrizione di quando questo errore esce e lo schema che avrà quando si presenterà.

Per agevolare la lettura dell'API è stata anche utilizzata la notazione *@Schema* per fornire un nome appropriato a campi o classi.

7.4.2 Repository

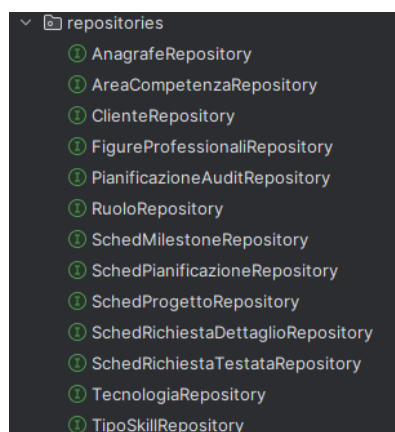


Figura 7.14: Repository sviluppati

Un Repository in Spring Boot fornisce un modo per interagire con una fonte di dati, in questo caso un database. Permette di eseguire operazioni CRUD tramite Spring Data JPA che genera automaticamente query SQL in base ai metodi che l'interfaccia del Repository contiene.

Le Repository da me create estendono, per poter eseguire le query, l'interfaccia `JpaRepository` o `PagingAndSortingRepository` per risposte paginate.

Ogni Repository deve essere annotato con `@Repository`.

```
@Repository
public interface SchedPianificazioneRepository extends JpaRepository<Pianificazione,String> {

    2 usages
    Optional<Pianificazione> findByRisorsaAndRuolo(AnagrafeLight risorsa,Ruolo ruolo);

}
```

Figura 7.15: Esempio di una Repository

È possibile creare delle query personalizzate SQL tramite parole chiave di Spring Data JPA. In questo caso nell'immagine la query corrisponderà alla ricerca della Pianificazione che ha la stessa Risorsa e lo stesso Ruolo.

7.4.3 Service

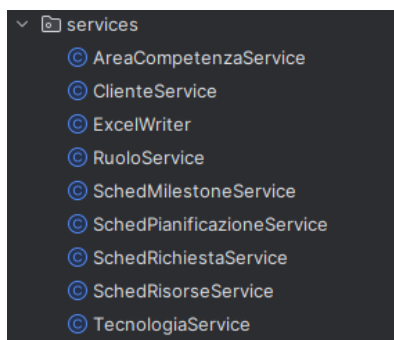


Figura 7.16: Service sviluppati

Il Service è un componente che contiene elaborazioni di dati, operazioni complesse, chiamate a metodi nei Repository per accedere al database e restituisce i risultati al Controller.

Ogni Service è annotato `@Service`. Può contenere fino a tre casistiche diverse di logiche implementative che sono elencate nelle prossime sotto-sezioni.

7.4.3.1 Esempio con utilizzo di metodi dei Repository

Questa casistica è la più frequente poiché permette di effettuare le semplici operazioni CRUD che gli endpoint solitamente richiedono. In questi casi vengono utilizzati metodi dei Repository come: `save()` per creare una nuova istanza di un'entità nel database, `deleteById()` per cancellare un'istanza da database e `findById()` per ricercare un'istanza precisa di un'entità nel database.

```

public MilestoneResponse deleteMilestone(String id) {

    Optional<Milestone> milestoneOptional = milestoneRepository.findById(id);
    if (milestoneOptional.isEmpty()) {
        throw new OmiPlanException(HttpStatus.NOT_FOUND, String.format("L'id %s inserito non è nel DB!", id));
    }

    if (!milestoneOptional.get().getPianificazioneList().isEmpty()) {
        throw new OmiPlanException(HttpStatus.CONFLICT, String.format("La milestone %s è collegata ad una Pianificazione",
            milestoneOptional.get().getId()));
    }

    milestoneRepository.deleteById(id);

    ResponseMetadata responseMetadata = new ResponseMetadata();

    return new MilestoneResponse(responseMetadata, new DtoMilestoneResponse(milestoneOptional.get()));
}

```

Figura 7.17: Metodo deleteMilestone() nel Service

In questo code snippet possiamo osservare come funziona il metodo `deleteMilestone()` all'interno del `MilestoneService`. Viene eseguito un controllo tramite `findById()` per verificare se l'ID è presente nel database e in caso negativo viene lanciata un'eccezione che impedisce l'eliminazione della Milestone. Questa eccezione comporta la restituzione di un codice di errore 404 per informare l'utente che la Milestone con l'ID specificato non esiste nel database e quindi non può essere cancellata.

Il controllo successivo verifica che la Milestone non sia associata ad una Pianificazione ed in caso di esito positivo lancia un'eccezione che contiene un codice di errore 409. Se tutti questi controlli non interrompono l'esecuzione, allora avverrà la cancellazione nel database tramite il metodo `deleteById()` della Milestone.

Viene infine generata la response da ritornare al Controller formata da un oggetto "metadata" vuoto, dato che non ci sono stati errori, e una nuova `MilestoneResponse`, contenente la Milestone rimossa che comporrà l'oggetto "data".

7.4.3.2 Esempio di creazione query con filtri

Un'altra logica rilevante utilizzata nei Service è quella che serve a generare una query dinamica in base ai filtri inseriti dall'utente.

Dato che la query con Spring Data JPA sarebbe stata troppo lunga e poco leggibile, è stato deciso di crearla tramite le interfacce JPA `CriteriaBuilder` e `CriteriaQuery`, che permettono di creare query dinamiche in modo programmatico.

```

CriteriaBuilder builder = entityManager.getCriteriaBuilder();
CriteriaQuery<Milestone> criteriaQuery = builder.createQuery(Milestone.class);
Root<Milestone> root = criteriaQuery.from(Milestone.class);
criteriaQuery.select(root).distinct(true);

Join<Milestone, Progetto> milestoneProgetto = root.join( attributeNamed: "progetto", JoinType.LEFT);

List<Predicate> finalPredicateBase = new ArrayList<>();
criteriaQuery.where(finalPredicateBase.toArray(new Predicate[]{}));

TypedQuery<Milestone> typedQuery = entityManager.createQuery(criteriaQuery);
typedQuery.setFirstResult(pagination.getPage()*pagination.getPageSize());
typedQuery.setMaxResults(pagination.getPageSize());
List<Milestone> result = typedQuery.getResultList();

```

Figura 7.18: Code snippet della creazione della query basata su filtri inseriti

Nel seguente estratto di codice ho ridotto la complessità mantenendo solo le parti essenziali che illustrano il funzionamento.

Nel primo blocco di istruzioni inizializziamo *CriteriaBuilder*, una nuova *CriteriaQuery* e selezioniamo tutte le *Milestone* uniche.

Nelle successive istruzioni andiamo a creare tutte le operazioni di *Join*<> tra le entità, inseriamo dei predicati (condizioni logiche, utili quando la query è dinamica) nella clausola *where()* della query e infine generiamo la query simulando la paginazione, recuperando la lista di risultati in una lista di *Milestone*.

Questa logica veniva utilizzata per costruire la query dinamica relativa agli endpoint che finiscono con */list*, per Richiesta, Pianificazioni e Milestone.

```
SELECT DISTINCT *
FROM sched_pianificazione as sp
INNER JOIN sched_richiesta_dettaglio srd ON sp.richiesta_dettaglio_id = srd.id
-- Ulteriori Join
WHERE 1=1
AND (filtro1 = 'valore' OR filtro2 = 'valore' OR filtro3 = 'valore' ...)
AND q = '';
```

Figura 7.19: Query dinamica

7.4.3.3 Esempio di creazione file Excel

```
Row header = sheet.createRow( rownum: 0);
Row rowValue = sheet.createRow( rownum: 1);

Cell headerCell = header.createCell( 0);
Cell cell = rowValue.createCell( 0);

CellStyle cellStyle = workbook.createCellStyle();

cell.setCellValue("valore");
cell.setCellStyle(cellStyle);

ByteArrayOutputStream fileOutput = new ByteArrayOutputStream();
try{
    workbook.write(fileOutput);
    workbook.close();
}catch(IOException e){
    throw new OmiPlanException(HttpStatus.INTERNAL_SERVER_ERROR,e.getMessage());
}
```

Figura 7.20: Code snippet semplificato della creazione di un file Excel

L'ultima logica rilevante è quella utilizzata per la creazione di un file Excel. In questo frammento di codice è stato semplificato il funzionamento per mostrare i punti cardine. La sintassi utilizzata dalla libreria Apache POI è molto semplice e intuitiva. Nelle

prime righe andiamo a creare una nuova riga `Row` per l'header e una che conterrà valori. Per ogni riga bisognerà creare poi una cella `Cell`. Successivamente è possibile applicare uno stile alla cella `CellStyle` e il suo valore. Dopo che il file Excel è stato compilato, il contenuto verrà scritto su un array di byte di tipo `ByteArrayOutputStream`.

```
return ResponseEntity
    .ok()
    .header(HttpHeaders.CONTENT_DISPOSITION, ..headerValues: "attachment; filename=estrazioni.xlsx")
    .contentType(MediaType.parseMediaType("application/vnd.ms-excel"))
    .body(fileToBeDownloaded);
```

Figura 7.21: Response fornita dal Controller contenente il file Excel da scaricare

Questo array verrà restituito al Controller che specificherà nella risposta HTTP fornita:

1. l'intestazione "Content-Disposition" nel formato "attachment", che indica al browser che il contenuto dovrebbe essere trattato come un allegato scaricabile;
2. imposta il tipo di contenuto del file, in questo caso Excel con estensione .xls;
3. specifica nel body della risposta l'array ritornato dal Service.

Il client vedrà la finestra di dialogo di download nel browser e potrà scaricare il file.

Capitolo 8

Verifica e Validazione

Questo capitolo descrive come è stata attuata la verifica e la validazione all'interno del progetto. Grazie al processo di verifica garantiamo che ogni attività dei processi svolti non introduca errori nel prodotto e che soddisfi i requisiti. Mentre con il processo di validazione viene determinato in maniera oggettiva che il prodotto sia conforme ai requisiti richiesti.

8.1 Processo di verifica

Durante tutto il periodo di tirocinio, ad ogni avanzamento di funzionalità e quindi cambio di requisito da soddisfare, con il tutor Antonio Fasolato è stato verificato che il codice sviluppato fino a quel momento fosse conforme con quanto aspettato e producesse output veritieri. Questo era aiutato inoltre dai controlli di validazione effettuati nei vari Controller e ulteriori controlli inseriti nei Service per garantire l'utilizzo di valori corretti.

Svolgendo le seguenti analisi regolarmente è stato possibile identificare e correggere errori evitando la loro propagazione nel corso della Codifica.

8.1.1 Analisi statica

È stata condotta un'analisi statica del codice prodotto mediante un processo di inspection, che consisteva in una revisione mirata e concentrata sui più comuni e probabili errori, prima di passare a quelli più specifici in base alla funzionalità implementata. Questa analisi non richiedeva l'esecuzione effettiva del software.

8.1.2 Analisi dinamica

L'analisi dinamica implica l'esecuzione di test sul prodotto in esecuzione. In caso di errore, veniva eseguita una verifica aggiuntiva dopo che avevo corretto l'errore riscontrato.

8.1.2.1 Debugging

Per identificare al meglio un errore a run-time veniva usata la tecnica di debugging: consiste nell'individuazione e la correzione di errori che provocano anomalie rilevate

durante l'esecuzione del programma. Questa tecnica è possibile grazie allo strumento di debugging che offre IntelliJ che permette di inserire dei breakpoint, cioè dei punti di interruzione nel codice, che non appena raggiunti durante l'esecuzione, sospende l'esecuzione ed è possibile ispezionare lo stato delle variabili, delle strutture dati e lo stack trace delle chiamate.

8.1.2.2 Postman

Per testare l'API veniva utilizzato il software Postman. Esso permetteva di osservare la response dell'endpoint in analisi per verificare che fosse conforme alle attese.

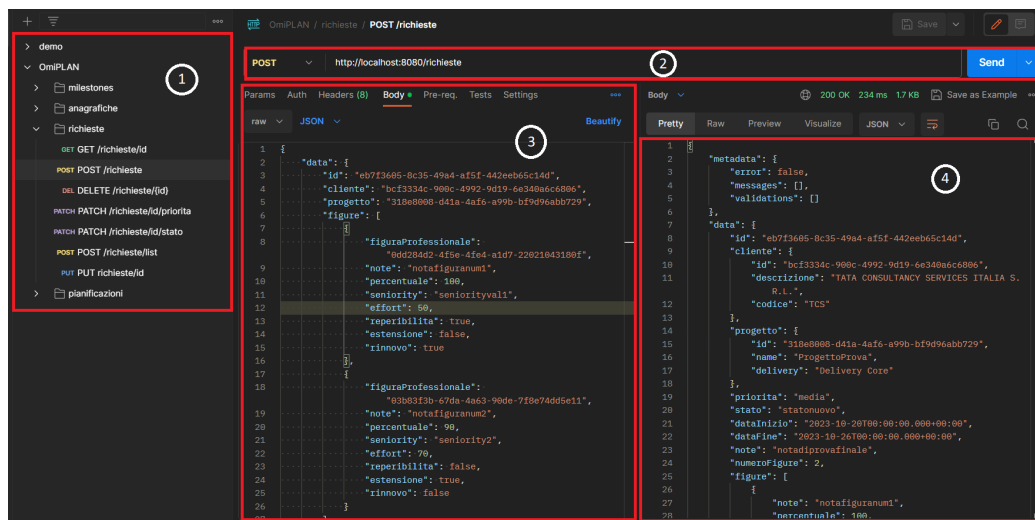


Figura 8.1: Interfaccia Postman

Per testare un endpoint si andava a configurare Postman nelle seguenti sezioni:

1. sezione in cui è possibile creare una nuova richiesta all'interno di una Collezione;
2. sezione in cui è possibile inserire l'URL che indica l'indirizzo del server e il percorso dell'endpoint a cui si desidera accedere, si seleziona il verbo HTTP e si aggiungono eventuali parametri di query. I parametri di query sono parte dell'URL che consente all'utente di filtrare, paginare o personalizzare i risultati;
3. sezione in cui è possibile inserire il body della request. Sopra al body si possono notare varie opzioni, di cui è stata utilizzata principalmente la sezione "Params", che permette di scrivere i parametri di query in modo più strutturato;
4. sezione in cui si potrà visualizzare la risposta in seguito al click sul pulsante "Send". La risposta sarà conforme alle specifiche selezionate.

8.2 Testing

Per garantire la sicurezza e l'affidabilità, negli ultimi giorni di tirocinio, sono stati introdotti i test di unità.

Questo processo è iniziato con lo studio dei framework JUnit e Mockito.

I test di unità verificano il funzionamento corretto di singole unità di codice, come metodi o classi. Questi test vengono sviluppati isolandosi dalle dipendenze esterne permettendo di verificare che il codice funzioni correttamente e che produca risultati attesi.

È stato ritenuto ragionevole testare ciò che io avevo prodotto come metodi o classi. Componenti introdotti da librerie o framework non hanno necessitato di test, dando per assodato il loro funzionamento.

Dato il poco tempo rimanente, per mettere in pratica quanto appreso, si è deciso di limitare lo svolgimento dei test su un *@Service* semplice, `MilestoneService`, andando a testare il corretto funzionamento dei suoi metodi.

8.2.1 Esempio di un test effettuato

```
@ExtendWith(MockitoExtension.class)
public class MilestoneServiceTests {

    14 usages
    @InjectMocks
    SchedMilestoneService milestoneService;

    11 usages
    @Mock
    SchedMilestoneRepository milestoneRepository;

    1 usage
    @Mock
    SchedProgettoRepository progettoRepository;

    1 usage
    @Mock
    SchedPianificazioneRepository pianificazioneRepository;

    @BeforeEach
    public void init() { MockitoAnnotations.openMocks( testClass: this); }
```

Figura 8.2: Configurazione ambiente di testing

Per poter eseguire dei test su oggetti simulati (mock) è stata utilizzata l'annotazione *@Mock* che ha permesso di creare i componenti mock da iniettare all'interno del Service sotto test, tramite l'annotazione *@InjectMocks*. Tramite il metodo `init()`, annotato con *@BeforeEach*, inizializziamo i componenti che vogliamo mockare permettendo l'utilizzo delle annotazioni che Mockito fornisce.

Qui sotto riporto un code snippet dei controlli eseguiti nel test relativo al metodo di creazione di una nuova Milestone. Per garantire il corretto funzionamento della creazione della Milestone, è stata testata ogni parte delicata che avrebbe compromesso il funzionamento del metodo, controllando l'eccezione sollevata, al momento dell'errore, assicurando che fosse quella corretta.

```
//Forcing the exception "Milestone già esistente"
when(milestoneRepository.findById(milestoneAlreadyPresent)).thenReturn(Optional.of(new Milestone()));
try {
    response = milestoneService.postMilestone(milestoneRequest);
    fail("The method should have raised an OmiPlanException");
} catch (OmiPlanException e) {
    // We expect the exception
    if(!e.getMessage().equalsIgnoreCase(String.format("L'id %s esiste già nel DB", milestoneAlreadyPresent))) {
        fail(String.format("Wrong exception raised (%s)", e.getMessage()));
    }
    // Correct exception raised
}

//Reset the Milestone Id to a new one
request.setId(UUID.randomUUID().toString());

when(progettoRepository.findById(projectId)).thenReturn(Optional.of(progetto));

when(pianificazioneRepository.findById(pianificazione1)).thenReturn(Optional.of(pianificazione));

when(milestoneRepository.save(any(Milestone.class))).thenReturn(new Milestone());

response = milestoneService.postMilestone(milestoneRequest);

ArgumentCaptor<Milestone> requestArgumentCaptor = ArgumentCaptor.forClass(Milestone.class);
verify(milestoneRepository, times(wantedNumberOfInvocations: 1)).save(requestArgumentCaptor.capture());

Milestone capturedRequest = requestArgumentCaptor.getValue();

Assertions.assertThat(response).isNotNull();
assertEquals(capturedRequest.getId(), request.getId());
```

Figura 8.3: Code snippet test di creazione di una nuova Milestone

In questo frammento di codice possiamo osservare come è stato forzato il lancio dell'eccezione quando si va a creare una Milestone con Id già esistente, controllando che sia stata proprio quella l'eccezione lanciata dal metodo.

Mockito offre strumenti, come il metodo `when()`, che consentono di effettuare stubbing dei metodi, simulando il loro comportamento. Questo permette di definire come tali metodi dovrebbero rispondere quando vengono invocati, restituendo risultati specifici anziché eseguire il codice reale.

Un ulteriore metodo fornito da Mockito è `verify()`, utilizzato per controllare che il metodo `save()` della repository della Milestone venga chiamato una sola volta. Con `ArgumentCaptor<>` (che permette di raccogliere i parametri passati in una funzione) andiamo a "catturare" la Milestone salvata col metodo `save()`.

Per accertare che questa parte di test abbia avuto successo è stato verificato che la response ricevuta dal metodo non fosse vuota e, dato che il DTO restituito differisce per alcune caratteristiche dall'istanza completa di una Milestone, non è stato possibile un confronto intero tra i due oggetti, ed è stato ritenuto esaustivo garantire l'uguaglianza dei due Id. Queste verifiche sono state effettuate tramite metodi di asserzione forniti da JUnit. Con l'ausilio di queste metodologie è stato possibile simulare e controllare la corretta creazione di una Milestone.

8.3 Validazione

Nell'ultimo giorno di tirocinio si è tenuto un incontro conclusivo insieme al tutor Antonio Fasolato e al Senior Technical Leader Giovanni Incammicia, a conferma definitiva della conformità alle specifiche richieste del prodotto finale. Durante questa

riunione, è stata effettuata una revisione dell'Analisi dei Requisiti fornita all'inizio del progetto, con particolare attenzione ai requisiti di mia competenza, al fine di verificare il loro soddisfacimento. Tuttavia, per scrivere il codice dell'API, è stato utilizzato principalmente un mock dell'API e l'Analisi dei Requisiti fornita è stata utilizzata per aiutare la comprensione di alcuni endpoint. Questo approccio ha comportato alcune discrepanze tra le funzionalità implementate e i requisiti specificati. Rispetto all'Analisi dei Requisiti mancavano delle informazioni che il Front-end necessitava o alcune funzionalità che necessitavano implementazioni particolari o di collegamenti a servizi esterni di cui non c'era tempo a sufficienza per poter studiare. Nonostante queste mancanze, analizzando le funzionalità implementate e le risposte fornite, è stato ritenuto sufficiente per il soddisfacimento dei requisiti di cui gli endpoint sviluppati facevano parte.

A causa delle problematiche da me sopra menzionate, non è stato possibile condurre una fase di Collaudo effettiva per valutare l'interazione tra il Front-end e il Back-end.

Capitolo 9

Deploy dell'applicazione

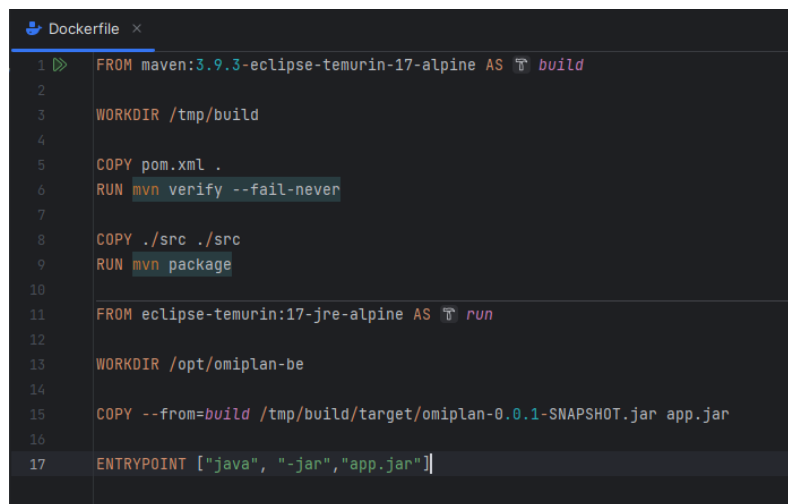
Al termine dell'attività di stage, insieme al tutor, ho potuto osservare come avviene il deploy di una Spring Boot app su Docker.

Per deployment del software si intende la distribuzione o avviamento, con relativa installazione e configurazione, di un'applicazione. È considerabile come fase finale del ciclo di vita di un software, fase che conclude lo sviluppo e il collaudo, dando inizio alla manutenzione.

9.1 Deploy

9.1.1 Dockerfile

Per cominciare è stato creato un **Dockerfile** nella root del progetto. Questo file definirà come Docker dovrebbe costruire l'immagine.

A screenshot of a code editor showing a Dockerfile. The file is titled 'Dockerfile' with a close button. It contains 17 lines of code, numbered on the left. The code defines two stages: 'build' and 'run'. The 'build' stage starts with 'FROM maven:3.9.3-eclipse-temurin-17-alpine AS build', sets 'WORKDIR /tmp/build', copies 'pom.xml', runs 'mvn verify --fail-never', copies source files, and runs 'mvn package'. The 'run' stage starts with 'FROM eclipse-temurin:17-jre-alpine AS run', sets 'WORKDIR /opt/omiplan-be', copies the built jar file from the build stage, and sets the 'ENTRYPOINT' to run the jar file.

```
1 FROM maven:3.9.3-eclipse-temurin-17-alpine AS build
2
3 WORKDIR /tmp/build
4
5 COPY pom.xml .
6 RUN mvn verify --fail-never
7
8 COPY ./src ./src
9 RUN mvn package
10
11 FROM eclipse-temurin:17-jre-alpine AS run
12
13 WORKDIR /opt/omiplan-be
14
15 COPY --from=build /tmp/build/target/omiplan-0.0.1-SNAPSHOT.jar app.jar
16
17 ENTRYPOINT ["java", "-jar", "app.jar"]
```

Figura 9.1: Configurazione Dockerfile

All'interno del file, possiamo notare due stage separati, quello di build e quello di run (esecuzione).

Stage Build

Nello stage di "build" viene indicato che si sta costruendo un'immagine di Docker basata su Maven 3.9.3 e Java 17. Vengono poi eseguite operazioni relative alla compilazione dell'applicazione, come la copia del codice sorgente, la verifica delle dipendenze tramite il tool di build Maven e la creazione del pacchetto `.jar`.

Stage Run

Lo stage di "run" ha come obiettivo l'esecuzione dell'applicazione. In questo stage viene utilizzata nuovamente la stessa immagine di Java e vengono copiati i risultati della prima fase. L'istruzione finale `ENTRYPOINT` definisce il comando che verrà eseguito quando il container basato su questa immagine verrà avviato.

9.1.2 Costruzione dell'immagine e avvio dell'applicazione

In seguito alla creazione del `Dockerfile` si apre il terminale nella directory in cui si trova questo file.

Seguono poi i seguenti passi:

1. viene eseguito il comando `docker build -t nome_immagine` per creare l'immagine Docker con le specifiche selezionate;
2. una volta costruita l'immagine si può avviare il container tramite il comando `docker run -ti nome_immagine`;
3. per accedere all'applicazione è possibile utilizzare un browser o un'applicazione da riga di comando come `cURL`, digitando il seguente indirizzo per interagire con l'applicazione: `http://localhost:porta_scelta`.

Capitolo 10

Conclusioni

Il mio percorso è iniziato con lo studio dei concetti di base e delle tecnologie che avrei utilizzato, sotto la guida del tutor Antonio Fasolato, prima di passare alle esercitazioni sulle tecnologie.

Tra le esercitazioni iniziali, quella su SpringBoot è risultata essere la più complessa. Essendo concetti nuovi che andavo ad affrontare, come l'utilizzo delle annotazioni di SpringBoot, non è stato semplice all'inizio comprendere il funzionamento del framework e questo ha portato ad un maggior investimento di tempo rispetto alle altre esercitazioni. È stato comunque un investimento proficuo poichè ha permesso di formarmi al meglio sulla tecnologia principale e alla base di quello che andavo a sviluppare.

Per poter comprendere al meglio ciò che andavo a creare, è stato svolto un incontro iniziale con il Senior Technical Leader Giovanni Incammicia, che mi ha illustrato le funzionalità del progetto dal lato Front-End. Successivamente è stato quindi deciso di definire le basi del mio prodotto, iniziando dal database e dallo scheletro delle risposte e delle richieste utilizzate dall'API REST.

Come linea guida è stato utilizzato un disegno iniziale dell'API e un'Analisi dei Requisiti fornita che comprendeva anche il lato Front-End.

Da qui è cominciato il mio lavoro sullo sviluppo delle nuove tabelle del database su cui poggiava l'API, integrandole con le tabelle aziendali fino allo sviluppo delle risposte e delle richieste che l'API utilizzava per ogni funzionalità.

Le ultime attività effettuate sono state il testing e la validazione di quanto sviluppato. Tuttavia, lo sviluppo delle funzionalità ed eventuali problematiche implementative, hanno ridotto il tempo disponibile per il testing. Di conseguenza, sebbene sia stato affrontato in modo adeguato a livello teorico, non è stato possibile implementarlo pienamente, come evidenziato nella sezione di [Testing](#).

10.1 Consuntivo finale

L'attività di tirocinio è durata 320 ore, come preventivato nel piano di lavoro, iniziando il 19 Giugno e finendo il 24 Agosto. La ripartizione delle ore mostrata nel piano di lavoro, anche se indicativa, è stata abbastanza rispettata.

Inizialmente, le esercitazioni sono state spalmate in modo adeguato nelle prime settimane. Successivamente, il tempo dedicato alla progettazione è stato utilizzato completamente, mentre il tempo riservato al testing non è stato sufficiente, a causa delle problematiche riscontrate nello sviluppo dell'API dovute all'adozione di tecnologie poco conosciute, che hanno causato rallentamenti.

10.2 Principali problematiche e soluzioni

Problematiche

1. Per cominciare lo sviluppo del progetto è stata presa come base un'Analisi dei Requisiti del progetto completo insieme ad una bozza dell'API che dovevo andare a sviluppare. È stato dunque difficile riuscire a far concordare quello che l'API suggeriva di implementare con il requisito completo che l'Analisi dei Requisiti forniva ed arrivare al codice da sviluppare;
2. Le funzionalità che ho implementato richiedevano l'accesso a dati recuperati da tabelle da me create. Tuttavia queste tabelle necessitavano di dati da altre tabelle già esistenti nel database aziendale. La problematica era quella di identificare quali tabelle del database aziendale dovevo collegare alle tabelle di mia creazione;
3. Nel periodo finale di tirocinio mi sono interfacciato con chi lavorava nel lato Front-End per poter cominciare ad integrare il mio lavoro ed avere una visione più completa di quanto implementato. Per poter fare questo bisognava quindi avere una documentazione dell'API chiara e concordata tra le parti.

Soluzioni

1. Per lo sviluppo è stato preso in considerazione il mock iniziale dell'API , utilizzando come consulto per un chiarimento sulla funzionalità fornita dall'endpoint, l'Analisi dei Requisiti, come spiegato nella sezione di [Validazione](#). A fine del tirocinio è stata poi creato un nuovo documento dell'API, come descritto nella sezione dello [Swagger](#) nel capitolo di Sviluppo dell'API REST, considerandolo come documento unico per la consultazione dell'API sviluppata;
2. Per avere una comprensione migliore del database aziendale mi è stato fornito un backup del database di testing aziendale con dati fittizi in cui c'era la possibilità di testare come volevo. Grazie al tutor e agli altri collaboratori ho avuto una visione più completa di come funzionasse il database aziendale e a quali tabelle dovevo fare riferimento;
3. Per poter comunicare correttamente con gli altri sviluppatori è stato creato il file Swagger. Tuttavia, come spiegato nel capitolo di [Validazione](#), a fine progetto, anche se l'API era chiara e documentata, ci sono state delle discrepanze tra le funzionalità implementate che non hanno permesso la fase di Collaudo.

10.3 Analisi critica del prodotto

10.3.1 Utilizzo del prodotto

Il prodotto sviluppato verrà integrato con la parte di Front-End in seguito ad un refactoring del codice e all'aggiunta delle funzionalità mancanti da chi di dovere. Le tabelle da me create verranno integrate al database aziendale per permettere la fruizione dei servizi dell'API.

In seguito all'integrazione delle funzionalità il prodotto sarà disponibile all'utilizzo da parte dei Project Manager e dei Program Manager per semplificare la creazione di richieste e di pianificazioni di risorse aziendali.

10.3.2 Valutazione strumenti utilizzati

Spring

L'utilizzo delle annotazioni per fornire un significato preciso, introducendo quindi il design pattern Inversion of Control, e l'iniezione di alcuni componenti all'interno di altri, mi ha messo in difficoltà all'inizio del progetto, rallentando la fase di sviluppo. Un elemento che mi ha colpito dell'adozione di Spring è stato come andasse a recuperare le configurazioni di ogni componente e, una volta capito al meglio la funzionalità del framework, quanto fosse facile aggiungerne.

Spring Data

Mi ha particolarmente colpito l'uso delle interfacce offerte da Spring Data JPA, come ad esempio *JpaRepository*, e la possibilità di effettuare operazioni CRUD personalizzate nel database mediante parole chiave. Questo mi ha fatto comprendere quanto semplificasse l'accesso al database.

Swagger

Strumento utilizzato per rappresentare lo sviluppo della mia API. Sono rimasto soddisfatto della completezza dello strumento e della chiarezza della sua interfaccia grafica. All'inizio del progetto, questa chiarezza mi ha aiutato a comprendere al meglio la composizione delle risposte e delle richieste di ogni endpoint del mock dell'API fornito.

10.3.3 Completamento dei Requisiti

In totale è stato sviluppato il circa il 90% dei requisiti estrapolati dall'analisi di cui tutti quelli obbligatori e alcuni desiderabili. I restanti requisiti non soddisfatti corrispondono al [RFOP-01](#), [RFDE-46](#) e i conseguenti, [RFDE-52](#), [RFOP-63](#) e [RFOP-64](#). Questi requisiti non sono stati sviluppati per limiti di tempo. Sono stati comunque trattati con il tutor e discusso su una possibile loro implementazione.

10.3.4 Possibili estensioni

Il mio prodotto non è completo e sicuramente mancano alcune funzionalità che lo porterebbero ad essere un prodotto migliore.

Le possibili estensioni possono essere:

- un refactoring del codice;
- implementazione di servizi esterni per il recupero di informazioni sulle risorse aziendali;
- calcolo delle date di fine o dei giorni in base alla disponibilità delle risorse, interfacciandosi con ulteriori software di gestione delle risorse umane o delle attività.

10.4 Conoscenze acquisite

Le conoscenze acquisite in questi anni universitari hanno agevolato di molto la comprensione delle nuove tecnologie che sono andato a studiare. Ritengo di aver acquisito,

in seguito a questa esperienza, conoscenze soddisfacenti verso le tecnologie elencate nel capitolo di [Background Tecnologico](#), con particolare rilevanza sottolineo: SpringBoot, Spring Data, Postman e Swagger.

10.5 Valutazione personale

Al termine di questa attività, mi ritengo soddisfatto e felice di aver intrapreso un'esperienza di questo tipo. Questa opportunità mi ha aiutato a comprendere come funziona un'azienda nel campo dell'informatica, ma soprattutto mi ha messo alla prova sulle mie conoscenze, contribuendo a consolidarle.

Anche se sono contento di ciò che ho sviluppato e imparato, mi rammarico di non essere riuscito ad osservare direttamente l'integrazione tra ciò che ho implementato io e il Front-End.

Ringrazio il mio tutor, Antonio Fasolato, per la sua disponibilità, pazienza e dedizione nel farmi comprendere concetti a me sconosciuti, incoraggiandomi ad affrontare tutti i problemi.

Ringrazio inoltre tutte le persone che ho conosciuto all'interno dell'azienda, apprezzando molto l'accoglienza ricevuta e l'atmosfera che si respirava in azienda.

Considero questa opportunità un'esperienza ottima per il concludere il mio percorso di laurea e ringrazio l'Università e l'azienda Omicron Consulting della possibilità.

Glossario

Annotazione

È un tipo di metadato che viene utilizzato per fornire informazioni aggiuntive su classi, metodi o campi all'interno di un'applicazione Spring.

API

Acronimo di Application Programming Interface, è un insieme di regole e protocolli che permettono a diversi componenti di comunicare tra loro, definendo il loro modo di interagire.

Audit

Per traccia di audit si intende la registrazione dettagliata di attività o eventi in un database. Una traccia di audit contiene molteplici audit, cioè delle registrazioni di attività singole che possono contenere: data e ora, utente, operazioni e dati cambiati.

Business Intelligence

Ci si riferisce a un insieme di processi aziendali o di tecnologie per raccogliere dati ed analizzare informazioni strategiche.

Business Logic

In ambito dello sviluppo software per logica di business (business logic) si intende la logica che rende operativa un'applicazione. Ci si riferisce ad operazioni complesse ed elaborazioni sui dati.

CRUD

Acronimo che sta per operazioni di base effettuate sui dati in un'applicazione o nel contesto informatico. Le operazioni a cui si fa riferimento sono: Create, Read, Update, Delete.

Class level

Si utilizza questo termine per indicare che un elemento è condiviso da tutte le istanze di quella class e che opera allo stesso livello della classe.

Dialecto del server SQL

In informatica, un "dialetto SQL" si riferisce ad una specifica implementazione del linguaggio SQL.

Dipendenza

Pacchetti necessari alla costruzione del progetto.

Endpoint

È un luogo digitale esposto tramite l'API dal quale l'API riceve le richieste e invia le risposte. Ogni endpoint è un URL che fornisce la posizione di una risorsa sul server.

Framework

È una struttura dove un software può essere progettato e realizzato, facilitandone lo sviluppo al programmatore. Sono progettati per affrontare problemi comuni e offrire una base solida.

HTTP

Acronimo di Hypertext Transfer Protocol, è un protocollo usato come sistema di trasmissione di informazioni sul web.

ICT

Per ICT (tecnologie dell'informazione e della comunicazione, in italiano) si intende tutte le attività e le tecniche di elaborazione automatica, trasmissione e di archiviazione delle informazioni tramite computer ed elaboratori elettronici.

Mapping

Per mapping si intende il processo di associazione (mappatura) dei dati tra due insiemi di dati.

Mock

Con il termine mock o mockare o mockato, si intende la simulazione del comportamento di un componente. Termine utilizzato nel testing software poichè consente di isolare unità di codice e verificare che il software funzioni correttamente.

ORM - Object relational mapping

È una tecnica di programmazione che facilita l'integrazione di sistemi software basati sulla programmazione orientata agli oggetti con sistemi di gestione di basi di dati relazionali.

Paginazione

È una tecnica utilizzata quando siamo di fronte ad grande insieme di dati e permette di presentarlo all'utente in pagine di elementi del risultato. I parametri della paginazione possono essere: numero di pagina, quanti elementi per pagina, direzione del risultato e ordinamento per termine.

Package

Permettono di organizzare classi Java in gruppi logici in base al contesto. Essi possono contenere ulteriori package, i subpackage.

Parametri di query

Sono dati o valori inseriti in una richiesta HTTP e vengono utilizzati per filtrare, ordinare o personalizzare i risultati. Nelle richieste HTTP i parametri di query vengono aggiunti dopo il "?" e separati da un "&". Un esempio può essere l'URL: `https://localhost:porta/riciesta/cerca?parametro1=prova¶metro2=test`.

Sandboxing

Termine utilizzato nella sicurezza informatica per far eseguire applicazioni in un ambiente controllato, fornendo un set di risorse limitate.

Stakeholder

Rappresentano i portatori di interesse, un insieme di persone coinvolte in decisioni importanti nel ciclo di vita di un software e che hanno influenza sul prodotto o sul processo.

Timestamp

Si intende il formato standard di timestamp composto da data e ora con precisione al secondo.

Trigger

Sono delle procedure associate ad una tabella di un database che si attivano automaticamente dopo un determinato evento su quella tabella. I possibili eventi sono operazioni di INSERT, UPDATE e DELETE.

UML

Acronimo di Unified Modeling Language, creato per realizzare un linguaggio di modellazione visivo comune.

URI

Acronimo di Uniform Resource Identifier e consiste in una sequenza di caratteri che identifica una risorsa univocamente. Un esempio di URI sono gli URL.

URL

Acronimo di Uniform Resource Locator che identifica univocamente una risorsa in rete.

UUID

Acronimo di Universally unique identifier che permette di garantire l'univocità di una chiave. Esso è composto da 16 byte (128 bit). Nella sua forma canonica è rappresentato da 32 caratteri esadecimali, visualizzati in cinque gruppi separati da trattini, nella forma 8-4-4-4-12.

Bibliografia

Siti web consultati

Business Intelligence. https://it.wikipedia.org/wiki/Business_intelligence
Java. [https://it.wikipedia.org/wiki/Java_\(linguaggio_di_programmazione\)](https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione))
Spring. <https://docs.spring.io/spring-framework/reference/overview.html>
Spring Boot. <https://spring.io/projects/spring-boot>
Spring Data. <https://spring.io/projects/spring-data>
SQL. https://it.wikipedia.org/wiki/Structured_Query_Language
JSON. <https://www.json.org/json-it.html>
IntelliJ IDEA. https://it.wikipedia.org/wiki/IntelliJ_IDEA
Maven. <https://www.baeldung.com/maven>
DBeaver. <https://en.wikipedia.org/wiki/DBeaver>
Microsoft SQL Server. https://it.wikipedia.org/wiki/Microsoft_SQL_Server
Hibernate. <https://it.wikipedia.org/wiki/Hibernate>
ORM. https://it.wikipedia.org/wiki/Object-relational_mapping
Microsoft Teams. https://it.wikipedia.org/wiki/Microsoft_Teams
Notion. [https://en.wikipedia.org/wiki/Notion_\(productivity_software\)](https://en.wikipedia.org/wiki/Notion_(productivity_software))
Microsoft Excel. https://it.wikipedia.org/wiki/Microsoft_Excel
Visual Studio Code. https://it.wikipedia.org/wiki/Visual_Studio_Code
Docker. <https://it.wikipedia.org/wiki/Docker>
Docker Multi-stage. <https://docs.docker.com/build/building/multi-stage/>
Swagger. <https://swagger.io/docs/specification/about/>
Git. <https://git-scm.com/docs/git>
GitLab. <https://it.wikipedia.org/wiki/GitLab>
GitExtensions. <http://gitextensions.github.io/>
Front Controller. https://it.wikipedia.org/wiki/Front_Controller_pattern
Entità. <https://www.baeldung.com/jpa-entities>
Program Manager. <https://www.adecco.it/il-lavoro-che-cambia/program-manager>
Project Manager. <https://www.adecco.it/il-lavoro-che-cambia/project-manager>
Mockito. <https://site.mockito.org/>
JUnit. <https://it.wikipedia.org/wiki/JUnit>
Deployment. <https://it.wikipedia.org/wiki/Deployment>
Sistema Client-Server. https://it.wikipedia.org/wiki/Sistema_client/server

Architettura REST. <https://www.restapitutorial.com/>
Repository pattern. [https://java-design-patterns.com/patterns/repository/
#intent](https://java-design-patterns.com/patterns/repository/#intent)
IoC. <https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>
DTO Pattern. <https://www.javatpoint.com/dto-java>
Naming conventions. <https://restfulapi.net/resource-naming/>
API. https://it.wikipedia.org/wiki/Application_programming_interface
ICT. <https://www.javatpoint.com/spring-boot-annotations>
Sandbox. [https://it.wikipedia.org/wiki/Sandbox_\(sicurezza_informatica\)](https://it.wikipedia.org/wiki/Sandbox_(sicurezza_informatica))
UUID. https://it.wikipedia.org/wiki/Universally_unique_identifier