

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Documentazione Task T-1 Gruppo G11

Team:

Luigi Scotto Rosato

Leonardo Sole

Davide Vitale

M63001488

M63001487

M63001469

Docente:

Prof. Anna Rita Fasolino

Anno Accademico:

2022/2023

Indice

Capitolo 1: Introduzione	5
Capitolo 2: Specifiche di progetto	6
2.1 Utenti del sistema	6
2.2 Glossario dei termini	6
2.3 Requisiti Funzionali	7
2.4 Storie utente	8
2.5 Criteri di Accettazione	8
2.6 Requisiti Non Funzionali	10
2.7 Visione ad alto livello del sistema	11
Capitolo 3: Processo di Sviluppo	16
3.1 Strumenti Software di Supporto	16
3.2 Analisi delle Tecnologie Software	17
Capitolo 4: Fase di analisi	17
4.1 Use Case Diagram	18
4.2 Sequence Diagram	18
4.2.1 Sequence Diagram: RicercaClassi	18
4.2.2 Sequence Diagram: elencaClassi	19
4.2.3 Sequence Diagram: ordinaClassi	20
4.2.4 Sequence Diagram: filtraClassi	21
4.2.5 Sequence Diagram: DownloadClasse	22
4.2.6 Sequence Diagram: ModificaClasse	23
4.2.7 Sequence Diagram: UploadClasse	24
4.2.8 Sequence Diagram: eliminaClasse	25
4.2.9 Sequence Diagram: elencaReport	25
4.2.10 Sequence Diagram: nuovoLike	25
4.2.11 Sequence Diagram: nuovoReport	26
4.3 Activity Diagram	26
4.3.1 Activity Diagram: RicercaClassi	27
4.3.2 Activity Diagram: elencaClassi	28
4.3.3 Activity Diagram: ordinaClassi	29
4.3.4 Activity Diagram: filtraClassi	30
4.3.5 Activity Diagram: DownloadClasse	31
4.3.6 Activity Diagram: ModificaClasse	32
4.3.7 Activity Diagram: UploadClasse	33
4.4 System Domain Model	34
Capitolo 5: Fase di Progettazione	35
5.1 Pattern Architetturale MVC	35
5.1.1 Spring MVC	35
5.2 Component Diagram	36
5.3 Package Diagram	38
5.4 Application Class Diagram	40
5.5 Deployment	41

5.6 Integrabilità	42
5.6.1 API Locali	42
5.6.2 Strutture Dati	43
5.6.2.1 ClassUT	43
5.6.2.2 Admin	43
5.6.2.3 Interaction	44
5.6.2.4 Operation	44
5.6.3 Services	45
5.6.3.1 Login admin	45
5.6.3.3 Registrazione admin	45
5.6.3.4 Modifica Classe	45
5.6.3.5 Download Classe	46
5.6.3.6 Ricerca Classe	46
5.6.3.7 Eliminazione Classe	46
5.6.3.8 Upload Classe	47
5.6.3.9 Elenca Classi	47
5.6.3.10 Filtra Classe	47
5.6.3.11 Ordinamento Classi	47
5.6.3.12 Nuovo Report	48
5.6.3.13 Nuovo Like	48
5.6.3.14 Upload Interaction	48
5.6.3.15 Elenca Report	48
5.6.3.16 Elenca Interaction	49
5.6.4 Utils	49
5.6.4.1 getLikes	49
5.6.4.2 findReport	49
5.6.4.3 findByText	49
5.6.4.4 findAdminByUsername	50
5.6.4.5 searchAndFilter	50
5.6.4.6 filterByCategory	50
5.6.4.7 orderByDate	50
5.6.4.10 searchAndFilter	51
5.6.4.11 downloadClassFile	51
5.6.4.12 saveClassFile	52
5.6.4.13 deleteDirectory	52
5.6.5 Rest API	52
5.6.5.1 Elenca classi “/home” (GET)	53
5.6.5.2 Ricerca Classe per nome “/home/{text}” (GET)	53
5.6.5.3 Ordina classi per nome “/orderbyname” (GET)	54
5.6.5.4 Ordina classi per data “/orderbydate” (GET)	54
5.6.5.5 Filtra classi per categoria “/Cfilterby/{category}” (GET)	55
5.6.5.6 Filtra classi per categoria “/Cfilterby/{category}” (GET)	55
5.6.5.7 Filtra classi per categoria “/Dfilterby/{difficulty}” (GET)	56
5.6.5.8 Filtra classi per categoria “/downloadFile/{name}” (GET)	56

5.6.5.9	Upload della descrittore classe+ file.java "/uploadFile (POST)	57
5.6.5.10	Modifica classe "/update/{name} (POST)	58
5.6.5.11	Nuovo like ad una classe "/newLike/{name} (POST)	58
5.6.5.12	Nuovo report ad una classe "/newReport/{name} (POST)	58
5.6.5.12	Registrazione Admin "/registraAdmin/ (POST)	59
Capitolo 6: Implementazione		60
6.1	Introduzione	60
6.2	Sequence Diagram di Dettaglio	60
6.2.1	Sequence Diagram: RicercaClassi	60
6.2.2	Sequence Diagram: elencaClassi	61
6.2.3	Sequence Diagram: ordinaClassi	61
	Order by date	61
	Order by name	62
6.2.4	Sequence Diagram: filtraClassi	63
	Filter by difficulty	63
6.2.5	Sequence Diagram: DownloadClasse	64
6.2.6	Sequence Diagram: ModificaClasse	64
6.2.7	Sequence Diagram: UploadClasse	65
Capitolo 7: Testing		66
7.1	Test Suite per le API REST	66
7.1.1	Piano di test per la funzionalità: RicercaClassi	67
7.1.2	Piano di test per la funzionalità: filtraClassi	69
7.1.3	Piano di test per la funzionalità: DownloadClasse	70
7.1.4	Piano di test per la funzionalità: ModificaClasse	72
7.1.5	Piano di test per la funzionalità: UploadClasse	74
7.2	Testing con Postman	76
7.2.1	Testing RicercaClassi	76
7.2.2	Testing filtraClassi	77
7.2.3	Testing DownloadClasse	78
7.2.4	Testing ModificaClasse	79
7.2.5	Testing UploadClasse	82
Capitolo 8: Installazione		83
8.1	Installazione software in locale	83
8.1.1	Generazione del file jar	84
8.1.2	Esecuzione con Docker Desktop	84
8.1.3	Esecuzione con parametri di Default	85
8.1.4	Definizione degli indici di ricerca nel database	86

Capitolo 1: Introduzione

Il compito assegnato al Team è quello di sviluppare un catalogo online di classi di programmazione da testare, accessibile sia agli studenti che agli amministratori. Gli utenti avranno la possibilità di cercare, visualizzare, filtrare e scaricare classi di programmazione.

Gli studenti potranno inoltre interagire con le classi del sistema attraverso un like o un report.

Gli amministratori invece avranno il compito di inserire, modificare e cancellare classi all'interno del catalogo e visualizzare i report inseriti dagli studenti.

L'applicazione fornirà agli utenti un'interfaccia web intuitiva per navigare e cercare tra le varie classi di programmazione disponibili. Saranno in grado di utilizzare filtri per restringere la ricerca in base a diversi criteri come la data di creazione della classe, il livello di difficoltà o la categoria di interesse.

Inoltre, gli utenti potranno visualizzare i dettagli di ogni classe, come la descrizione, la data di creazione, il numero di like e le categorie di appartenenza. Saranno in grado di scaricare il file associati a ogni classe.

Gli amministratori avranno accesso a un'interfaccia web dedicata per gestire il catalogo. Potranno aggiungere nuove classi, modificarne i dettagli o eliminarle se necessario. Questa funzionalità consentirà agli amministratori di mantenere il catalogo aggiornato e rilevante per gli utenti.

Gli amministratori avranno un'interfaccia web per il login e una per la registrazione di un nuovo Admin.

L'obiettivo principale dell'applicativo sarà fornire una piattaforma user-friendly e funzionale per studenti di diversi livelli di esperienza in programmazione, consentendo loro di accedere a una vasta gamma di classi di programmazione e supportando sia il loro apprendimento che il loro progresso nel campo della programmazione.

Capitolo 2: Specifiche di progetto

2.1 Utenti del sistema

- **Studenti:** Gli studenti sono gli utenti principali del sistema, possono accedere al catalogo di classi di programmazione, cercare, visualizzare, filtrare e scaricare le classi disponibili; possono inoltre interagire con le classi attraverso un like o un report.
- **Amministratori:** Gli amministratori sono responsabili della gestione del catalogo di classi di programmazione. Hanno accesso privilegiato per inserire nuove classi, modificarne i dettagli o eliminarle se necessario. Possono accedere a un'interfaccia dedicata per gestire le operazioni amministrative. Hanno il compito di mantenere il catalogo aggiornato, garantendo che le informazioni sulle classi siano accurate e pertinenti. Possono monitorare l'utilizzo del sistema visualizzando le interazioni degli studenti con le classi.

2.2 Glossario dei termini

Termine	Alias	Descrizione	Formato
Amministratore	Admin, Utente	Un utente con autorizzazioni speciali per gestire il software	Costituito da stringhe riguardanti l'username e la password
Studente	Utente, Fruitore del servizio	Studente che utilizza l'applicazione per accedere alle funzionalità disponibili, senza poter effettuare operazioni di amministrazione	Costituito da stringhe riguardanti i dati anagrafici, l'id, l'username e la password
Class List	Elenco delle classi	Lista di tutte le classi testabili disponibili nell'applicazione	Lista di classi
Interaction	Interazione, Like , Report	Le interazioni tra studente e classe possono essere un like o un report in cui si possono effettuare segnalazioni sulla classe in questione	Azione
Operation	Operazione	Indica le operazioni dell'amministrazione sulla classe , possono essere :creazione,modifica,eliminazione	Azione
Scaricare	Download	Trasferire un file o un insieme di file da un server remoto al proprio dispositivo	Azione
Inserire	Aggiungere	Aggiungere una nuova classe al sistema	Azione
Cancellare	Rimuovere	Rimuovere una classe esistente dal sistema	Azione

Criteri di ordinamento	Ordinamento	Opzioni per ordinare le classi in base a diversi attributi	Azione
Etichette	Tag, Categorie	Le tag o le categorie assegnate alle classi per facilitare la ricerca e il filtraggio	Attributo della classe
Filtrare	Ridurre	Ridurre l'elenco di classi visualizzate in base alle etichette selezionate dall'utente	Azione
Modificare	Modifiche	Apportare modifiche a una classe esistente nel sistema	Azione
Interfaccia utente	UI	Il layout e l'aspetto visivo dell'applicazione che l'utente vede e con cui interagisce	Elemento grafico
Ricerca	Cercare	Cercare una classe specifica nel sistema	Azione
Barra di ricerca	Campo di ricerca	Area dell'interfaccia utente in cui l'utente può inserire testo per cercare un elemento specifico all'interno dell'applicazione	Elemento grafico dell'interfaccia utente
Difficoltà	Complessità	Misura della complessità del codice di una classe, valutata in base al numero di righe di codice, al numero di dipendenze e ad altre metriche	Numero, Valore numerico

2.3 Requisiti Funzionali

I requisiti funzionali previsti per la web app sono i seguenti:

- L'applicazione deve mostrare un elenco di tutte le classi.
- L'applicazione deve consentire all'utente di scaricare il codice di una classe selezionata.
- L'applicazione deve consentire all'amministratore di inserire una nuova classe con nome, descrizione e codice.
- L'applicazione deve consentire all'amministratore di cancellare una classe esistente selezionata.
- L'applicazione deve consentire all'utente di cercare una classe per nome, attraverso un campo di ricerca.
- L'applicazione deve consentire all'utente di scegliere e cambiare i criteri di ordinamento delle classi visualizzate.
- L'applicazione deve consentire all'utente di selezionare una o più etichette per filtrare le classi visualizzate.
- L'applicazione deve consentire all'amministratore di modificare una classe esistente, inclusi nome, descrizione e codice.
- L'applicazione deve consentire all'amministratore di registrarsi inserendo le proprie informazioni personali (nome, cognome, username e password).

- L'applicazione deve consentire all'utente di effettuare il login al sistema inserendo il proprio username e password.
- L'applicazione deve consentire allo studente di inserire un nuovo like ad una classe.
- L'applicazione deve consentire allo studente di inserire un nuovo report ad una classe.
- L'applicazione deve consentire a studenti e amministratori di visualizzare il numero di like per ogni classe disponibile.
- L'applicazione deve consentire all'amministratore di visualizzare l'elenco dei report effettuati dagli studenti sulle classi

2.4 Storie utente

STORIA	STIMA TEMPORALE	PRIORITÀ
Come utente voglio consultare le classi disponibili	1	1
Come utente voglio poter scaricare il codice di una classe	3	2
Come amministratore voglio inserire una nuova classe	3	3
Come amministratore voglio cancellare una classe esistente	2	4
Come utente voglio ricercare una classe per nome	2	5
Come utente voglio scegliere e cambiare i criteri di ordinamento delle classi.	2	6
Come utente voglio scegliere una o più etichette per filtrare le classi	3	7
Come amministratore voglio modificare una classe esistente	3	8
Come amministratore voglio potermi registrare nel sistema	3	9
Come amministratore voglio potermi autenticare nel sistema	3	10
Come studente voglio inserire un like ad una classe	2	11
Come studente voglio inserire un report ad una	2	12

classe		
Come amministratore voglio visualizzare i report effettuati dagli studenti sulle classi	1	13
TOTALE	30	

***Parametri di ordinamento:** Complessità, data di ultima modifica, per livelli.

***Parametri di filtraggio:** Livello, etichetta, complessità.

2.5 Criteri di Accettazione

I criteri di accettazione sono specifiche dettagliate scritte utilizzando il modello GIVEN-WHEN-THEN-AND, che definiscono il comportamento atteso del sistema. Qui di seguito segue l'elenco completo dei criteri di accettazione basati su questo modello per le storie utente fornite:

1. Come utente voglio consultare le classi disponibili

GIVEN: La web application è accessibile

WHEN: L'utente accede alla pagina delle classi disponibili

THEN: L'utente visualizza l'elenco delle classi disponibili con il nome, il livello di difficoltà e un breve riassunto

AND: L'utente può cliccare su una classe per visualizzarne dettagli, come la descrizione completa, gli esempi di utilizzo e il codice sorgente.

2. Come utente voglio poter scaricare il codice di una classe

GIVEN: L'utente ha accesso alla pagina della classe selezionata.

WHEN: L'utente clicca sul pulsante di download del codice sorgente.

THEN: Il codice sorgente della classe selezionata viene scaricato sul dispositivo dell'utente.

3. Come amministratore voglio inserire una nuova classe

GIVEN: L'amministratore ha accesso alla sezione di gestione delle classi.

WHEN: L'amministratore inserisce i dettagli della nuova classe e clicca sul pulsante di conferma.

THEN: La nuova classe viene aggiunta all'elenco delle classi disponibili.

AND: Gli utenti possono visualizzare e scaricare il codice sorgente della nuova classe.

4. Come amministratore voglio cancellare una classe esistente

GIVEN: L'amministratore ha accesso alla sezione di gestione delle classi.

WHEN: L'amministratore seleziona la classe da cancellare e clicca sul pulsante di conferma.

THEN: La classe selezionata viene cancellata dall'elenco delle classi disponibili.

AND: Il codice sorgente della classe non è più disponibile per il download.

5. Come utente voglio ricercare una classe per nome

GIVEN: La web application è accessibile.

WHEN: L'utente inserisce il nome della classe nella barra di ricerca e clicca sul pulsante di ricerca.

THEN: L'utente visualizza l'elenco delle classi che corrispondono al nome cercato.

AND: L'utente può cliccare su una classe per visualizzarne dettagli, come la descrizione completa e il codice sorgente.

6. Come utente voglio scegliere e cambiare i criteri di ordinamento delle classi

GIVEN: L'utente ha accesso alla pagina delle classi disponibili.

WHEN: L'utente seleziona una o più opzioni di ordinamento dall'elenco a tendina.

THEN: L'elenco delle classi disponibili viene ordinato in base ai criteri selezionati.

AND: L'utente può cambiare i criteri di ordinamento in qualsiasi momento.

7. Come utente voglio scegliere una o più etichette per filtrare le classi

GIVEN: L'utente ha accesso alla pagina delle classi disponibili.

WHEN: L'utente seleziona una o più etichette dall'elenco a tendina.

THEN: L'elenco delle classi disponibili viene filtrato in base alle etichette selezionate.

AND: L'utente può cambiare le etichette di filtro in qualsiasi momento.

8. Come amministratore voglio modificare una classe esistente

GIVEN: L'amministratore ha accesso alla sezione di gestione delle classi.

WHEN: L'amministratore seleziona la classe da modificare e apre la pagina di modifica.

THEN: L'amministratore può modificare i dettagli della classe, come il nome, la descrizione, il codice sorgente e difficoltà

AND: L'amministratore clicca sul pulsante di conferma per salvare le modifiche.

AND: La classe modificata viene aggiornata nell'elenco delle classi disponibili e il codice sorgente aggiornato è disponibile per il download.

9. Come amministratore voglio potermi registrare nel sistema

GIVEN: L'amministratore non ha ancora un account nel sistema.

WHEN: L'amministratore sceglie l'opzione di registrazione e inserisce le sue informazioni personali.

THEN: Il sistema verifica che l'email fornita dall'amministratore non sia già presente nel database degli utenti.

THEN: Il sistema verifica che la password inserita sia sicura e rispetti i requisiti minimi di lunghezza e complessità.

THEN: Il sistema salva le informazioni dell'amministratore nel database degli utenti e crea un account amministratore.

AND: Il sistema visualizza un messaggio di conferma che comunica all'amministratore che la registrazione è stata completata con successo.

AND: L'amministratore può accedere al sistema utilizzando le sue credenziali di accesso.

10. Come amministratore voglio potermi autenticare nel sistema

GIVEN: L'amministratore ha già un account nel sistema.

WHEN: L'amministratore inserisce le sue credenziali di accesso (email e password) nella pagina di login.

THEN: Il sistema verifica che l'email inserita corrisponda ad un account amministratore valido nel database degli utenti.

THEN: Il sistema verifica che la password inserita sia corretta e coincida con quella associata all'account amministratore.

AND: Il sistema reindirizza l'amministratore alla dashboard del sistema.

11. Come studente voglio inserire un like ad una classe

GIVEN: Lo studente ha effettuato l'accesso al sistema.

WHEN: Lo studente seleziona l'opzione "Like" associata ad una classe di programmazione.

THEN: Il sistema aumenta il contatore dei like associato alla classe di programmazione selezionata nel database.

AND: Il sistema visualizza un messaggio di conferma che comunica allo studente che il like è stato registrato con successo.

12. Come studente voglio inserire un report ad una classe

GIVEN: Lo studente ha effettuato l'accesso al sistema.

WHEN: Lo studente seleziona l'opzione "Report" associata ad una classe di programmazione.

THEN: Il sistema reindirizza lo studente ad una pagina dedicata alla compilazione del report.

THEN: Lo studente inserisce il proprio report e seleziona l'opzione "Invia".

AND: Il sistema salva il report nel database associato alla classe di programmazione selezionata.

AND: Il sistema visualizza un messaggio di conferma che comunica allo studente che il report è stato registrato con successo.

13. Come amministratore voglio visualizzare i report effettuati dagli studenti sulle classi

GIVEN: L'amministratore ha effettuato l'accesso al sistema.

WHEN: L'amministratore seleziona l'opzione "Visualizza report" associata ad una classe di programmazione.

THEN: Il sistema recupera tutti i report associati alla classe di programmazione selezionata dal database.

AND: Il sistema visualizza i report in una lista ordinata per data di invio.

AND: L'amministratore può visualizzare il contenuto di ogni report selezionando l'opzione "Visualizza dettagli".

2.6 Requisiti Non Funzionali

- **Usabilità:**
 - L'applicazione deve essere intuitiva e facile da usare, con un'interfaccia utente intuitiva e ben progettata.
 - Gli utenti devono poter navigare nel catalogo e utilizzare le funzionalità in modo semplice ed efficace.
- **Affidabilità:**
 - L'applicazione deve essere disponibile in modo continuo, garantendo una disponibilità del servizio pari almeno al 99,9% del tempo.
 - Gli utenti devono poter accedere al sistema senza interruzioni significative o tempi di inattività prolungati.
- **Scalabilità:**
 - L'applicazione deve essere scalabile in modo da supportare un aumento del numero di utenti e del traffico dati.
- **Sicurezza:**
 - L'applicazione deve essere sicura, proteggendo i dati degli utenti e delle classi.
- **Prestazioni:**
 - L'applicazione deve essere performante, con tempi di risposta rapidi e senza ritardi significativi.
 - Le richieste degli utenti, come la ricerca o il download di classi di programmazione, devono essere elaborate in modo efficiente e veloce.
- **Compatibilità:**
 - L'applicazione deve essere compatibile con diversi browser web e dispositivi, garantendo una buona esperienza utente su tutte le piattaforme.
- **Manutenibilità:**
 - L'applicazione deve essere facilmente manutenibile e aggiornabile, con un codice ben strutturato e documentato.
 - I futuri sviluppatori o amministratori devono essere in grado di comprendere e modificare il codice senza difficoltà, garantendo la scalabilità e la gestione continua dell'applicazione.

2.7 Visione ad alto livello del sistema

In figura (2.1) si presenta una vista semplice e chiara dall'esterno del sistema software, mostrando le User Interface con le quali le entità esterne interagiscono e i servizi esterni che vengono utilizzati. Nel sistema è previsto che tramite il browser si possa accedere alle seguenti interfacce utente:

- **Home_adminUI:** Questa interfaccia è dedicata agli amministratori del sistema, mostra un elenco delle classi disponibili. Possono cancellare e modificare le classi e fornisce un accesso ai report inviati dagli utenti.
- **HomeUI:** Questa interfaccia è dedicata agli studenti del sistema, mostra un elenco delle classi disponibili in una visualizzazione adatta all'utente. Hanno la possibilità di mettere "Mi piace" ad una classe e inviare un report.
- **ReportsUI:** Questa interfaccia è dedicata agli amministratori del sistema, mostra un elenco dei report inviati dagli utenti riguardanti le classi. Gli amministratori possono visualizzare i dettagli di ogni report, come la classe segnalata e il messaggio associato.
- **Report_classeUI:** Questa interfaccia è dedicata agli studenti e consente di inviare un report riguardante una classe specifica, gli utenti possono inserire un messaggio che descrive il problema o l'errore riscontrato nella classe.
- **Upload_classeUI:** Questa interfaccia è dedicata agli amministratori e consente di inserire una nuova classe e fare l'upload del file .java relativo.
- **Modifica_classeUI:** Questa interfaccia è dedicata agli amministratori e consente di modificare i dati di una classe esistente.
- **Admin_loginUI:** Questa interfaccia è dedicata agli amministratori e consente di autenticarsi nel sistema.
- **Modifica_classeUI:** Questa interfaccia è dedicata agli amministratori e consente di registrarsi nel sistema.

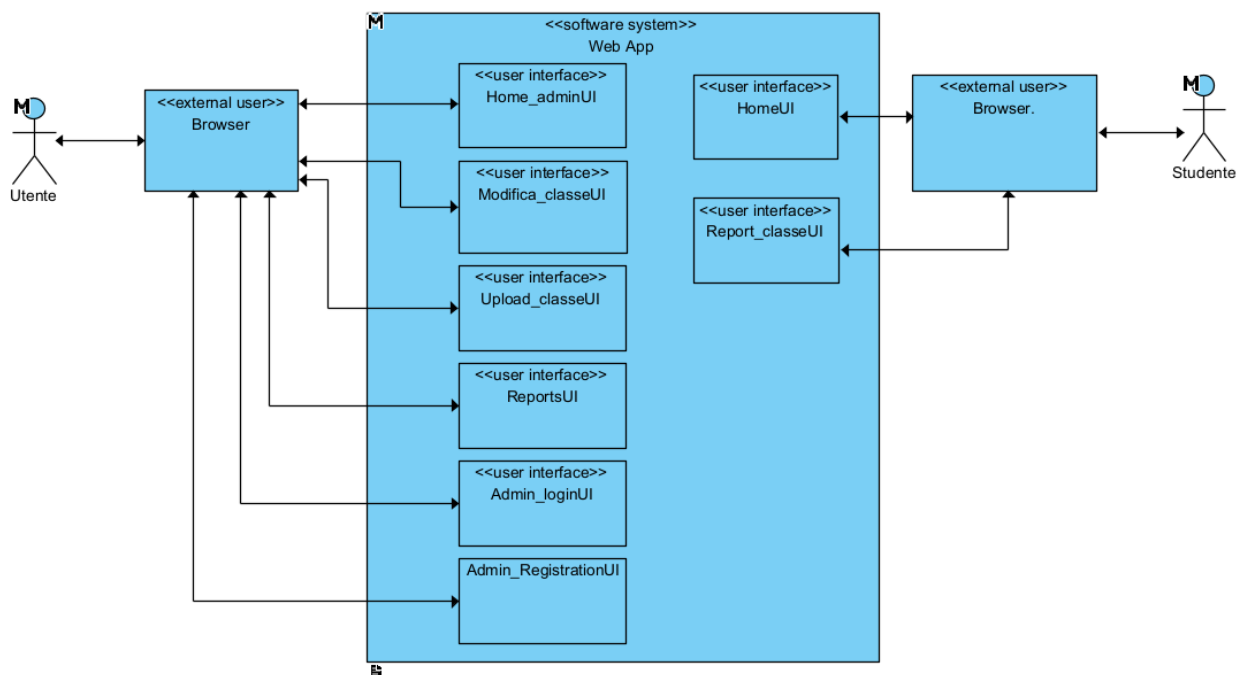


Figura 2.1: Context Diagram

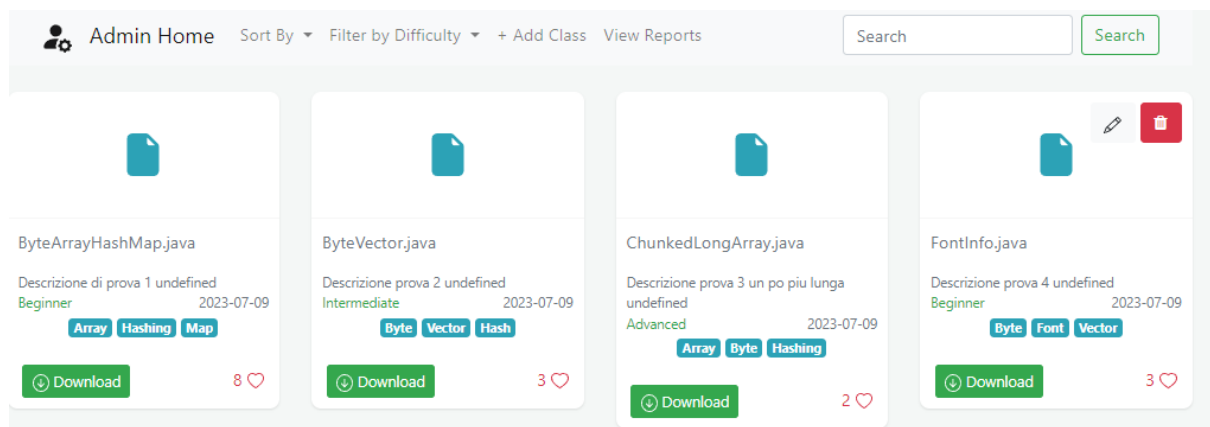


Figura 2.2: Home_adminUI

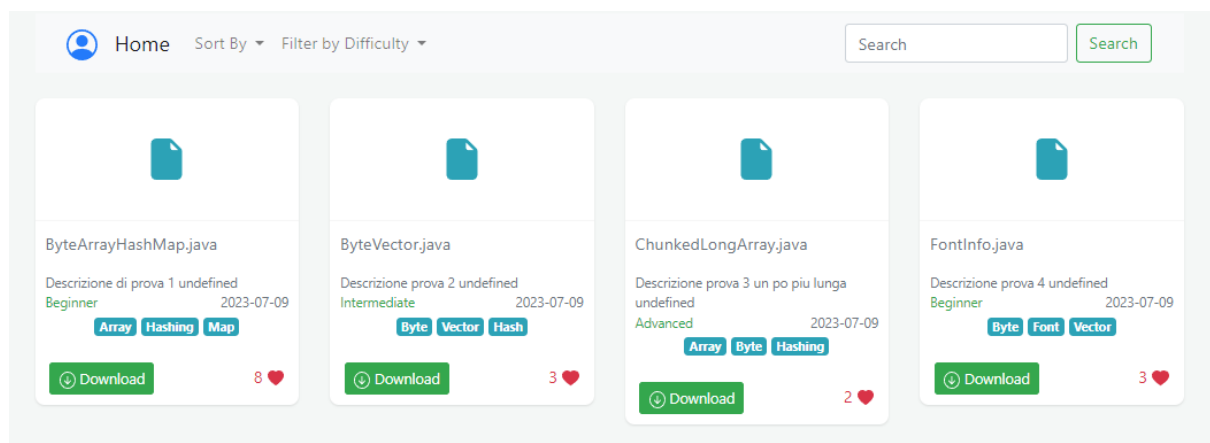


Figura 2.3: HomeUI

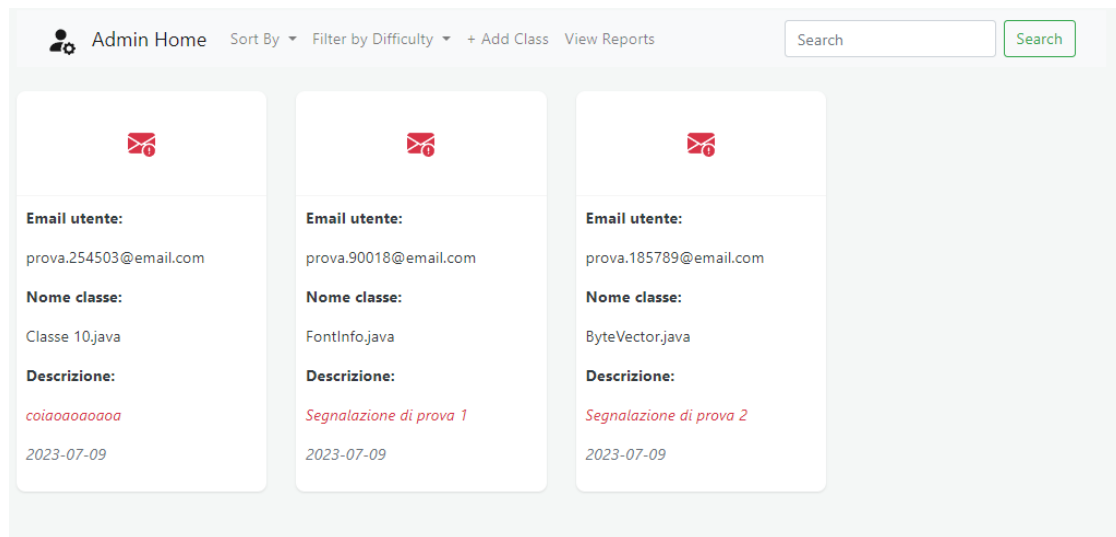


Figura 2.4: ReportsUI

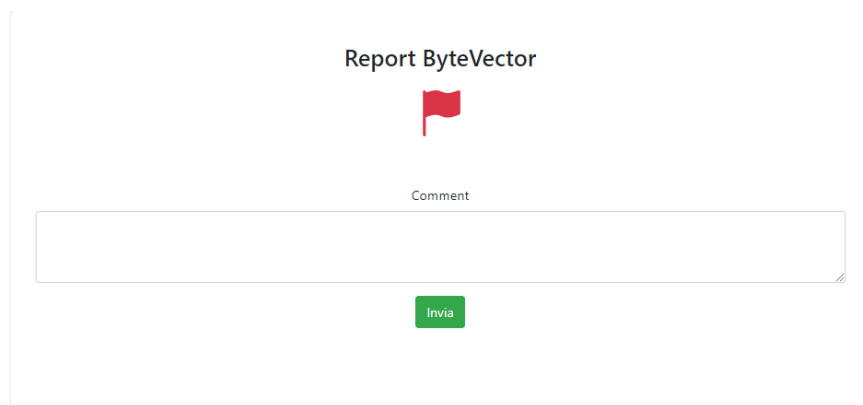



Figura 2.5: Report_classeUI

Class upload



Class name

Date

Difficulty Beginner

Select the difficulty level of this item

Description

Category 1

Category 2


Category 3

Upload your file (java) : Scegli file Nessun file selezionato

Upload

Figura 2.6: Upload_classeUI

Modifica Classe: ByteArrayHashMap



Class name

Date

Difficulty Beginner

Select the difficulty level of this item

Description

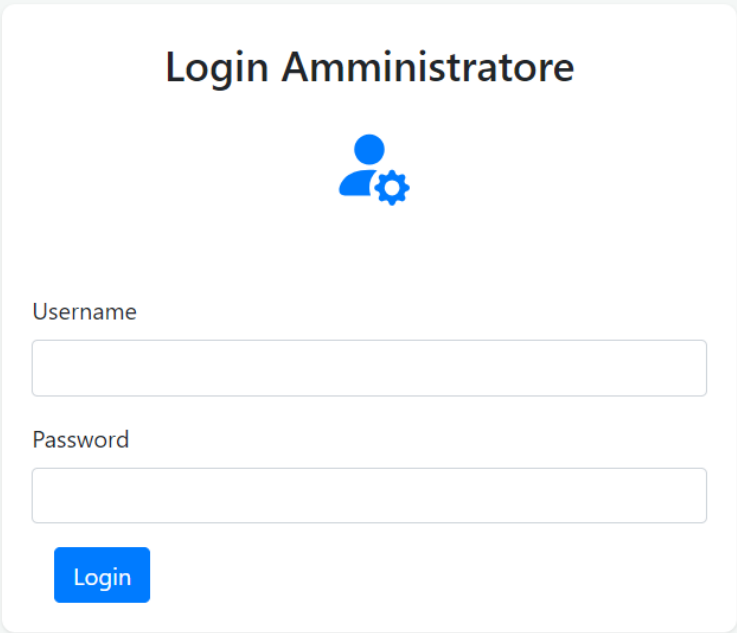
Category 1

Category 2

Category 3


Upload

Figura 2.7: Modifica_classeUI



The image shows a login interface for an administrator. It features a title 'Login Amministratore' at the top, followed by a blue icon of a person with a gear. Below this are two input fields: 'Username' and 'Password'. At the bottom is a blue 'Login' button.

Login Amministratore

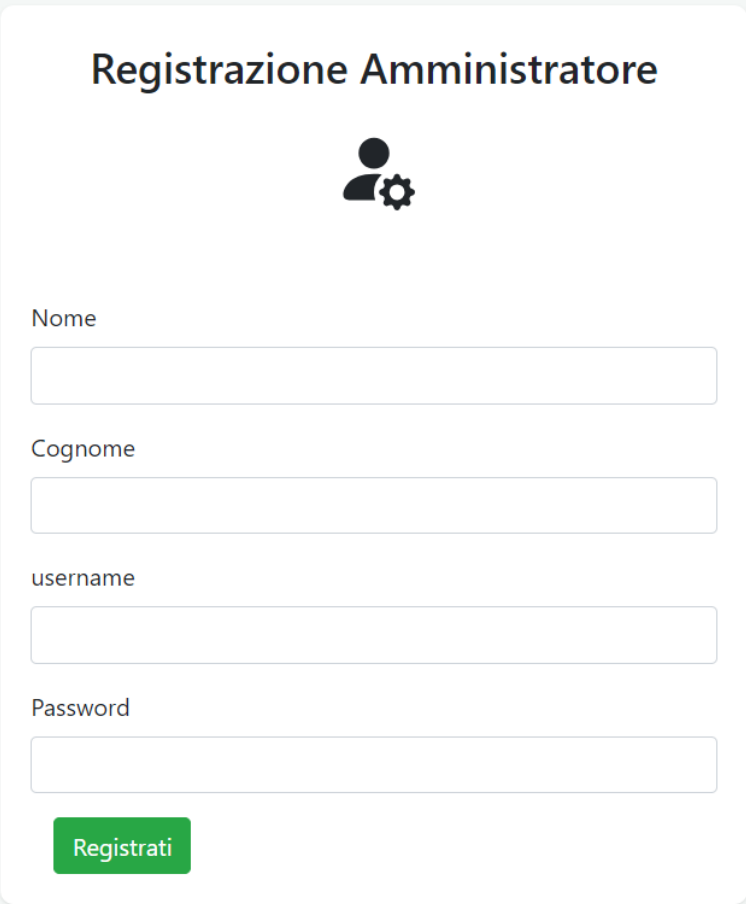


Username

Password


Login

Figura 2.8: Admin_loginUI



The image shows a registration interface for an administrator. It features a title 'Registrazione Amministratore' at the top, followed by a black icon of a person with a gear. Below this are four input fields: 'Nome', 'Cognome', 'username', and 'Password'. At the bottom is a green 'Registrati' button.

Registrazione Amministratore



Nome

Cognome

username

Password

Registrati

Figura 2.8: Registration_adminUI

Capitolo 3: Processo di Sviluppo

3.1 Strumenti Software di Supporto

Per il progetto di sviluppo del software, sono stati utilizzati i seguenti strumenti software di supporto:

- **Strumenti per la comunicazione del team:** si è utilizzato Discord e Microsoft Teams per la comunicazione del team. Questi strumenti hanno permesso di comunicare in tempo reale, scambiare informazioni e risolvere problemi in modo collaborativo. È stata utilizzata anche la funzione di messaggistica istantanea e la videoconferenza per facilitare la collaborazione a distanza.
- **Strumenti per la condivisione di file e documenti:** si è utilizzato Google Drive per la condivisione di file e documenti. Questo strumento ha permesso di accedere ai file in modo rapido e semplice da qualsiasi dispositivo e di lavorare contemporaneamente su di essi in tempo reale. Sono state utilizzate le funzioni di condivisione e commento per facilitare la collaborazione tra i membri del team.
- **Strumenti per la modellazione dei diagrammi:** si è utilizzato diagrams.net e Visual Paradigm per la modellazione dei diagrammi. Questi strumenti hanno permesso di creare diagrammi UML e di altri tipi, di visualizzare la struttura del software e di identificare i requisiti del sistema. Sono state utilizzate anche le funzioni di condivisione e collaborazione per lavorare contemporaneamente sui diagrammi.
- **Strumenti per l'implementazione del software:** si è utilizzato Eclipse e Visual Studio Code per l'implementazione del software. Questi strumenti hanno permesso di scrivere e testare il codice, gestire le dipendenze e compilare l'applicazione. È stata utilizzata anche la funzione di debugging per identificare e risolvere i problemi di codice.
Per il deployment dell'applicazione, è stato utilizzato Docker, una piattaforma di containerizzazione che ha semplificato la distribuzione dell'applicazione su server e dispositivi remoti.
- **Strumenti di sviluppo e gestione del progetto:** Git è stato utilizzato come sistema di controllo versione per il progetto. GitHub è stato utilizzato come piattaforma di hosting del progetto, fornendo uno spazio di archiviazione per il codice e uno strumento per la collaborazione tra i membri del team. JIRA è stato utilizzato come strumento di gestione del progetto, fornendo una piattaforma per la pianificazione, il monitoraggio e la gestione delle attività.

In sintesi, sono stati utilizzati una combinazione di strumenti software di supporto per facilitare la comunicazione del team, la condivisione di file e documenti, la modellazione dei diagrammi e l'implementazione del software. Questi strumenti hanno permesso di lavorare in modo collaborativo e di gestire il progetto in modo efficiente e produttivo.

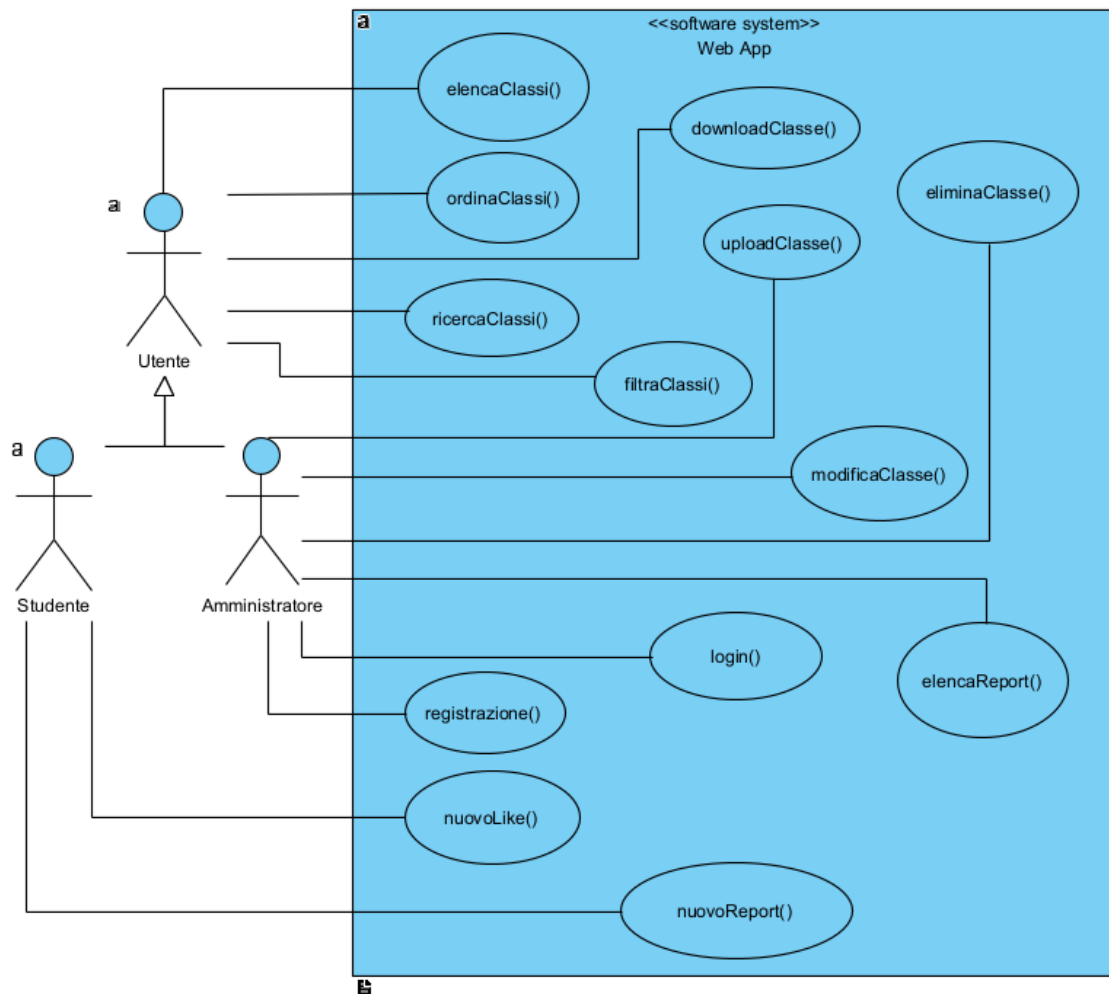
3.2 Analisi delle Tecnologie Software

- **Linguaggio di programmazione per il backend:** Java è stato scelto come linguaggio di programmazione per lo sviluppo del backend del software.
- **Framework per il backend:** Spring è stato scelto come framework per Java per lo sviluppo del backend del software. Spring è un framework flessibile e modulare che fornisce un'ampia gamma di funzionalità per lo sviluppo di applicazioni web. Grazie alla sua architettura modulare, Spring consente di selezionare solo i moduli necessari per l'applicazione e di integrarli facilmente con altri framework e librerie.
- **Database:** MongoDB è stato scelto come database per l'archiviazione dei dati del software. MongoDB è un database NoSQL orientato ai documenti, noto per la sua scalabilità, la flessibilità del modello dei dati e la facilità di utilizzo. Grazie alla sua architettura distribuita, MongoDB è una scelta comune per lo sviluppo di applicazioni web scalabili e ad alte prestazioni.
- **Linguaggi di markup, stili e scripting per il front end:** HTML, CSS e JavaScript sono stati scelti come linguaggi di markup, stili e scripting per lo sviluppo del front end del software. HTML è il linguaggio di markup standard per la creazione di pagine web, mentre CSS è il linguaggio di stile utilizzato per la formattazione del contenuto HTML. JavaScript è il linguaggio di scripting utilizzato per interagire con gli elementi HTML e per aggiungere funzionalità dinamiche alle pagine web.
- **Framework per il front end:** Bootstrap è stato scelto come framework per il front end del software. Bootstrap è un framework CSS e JavaScript che fornisce un set di strumenti per lo sviluppo di interfacce utente reattive e mobile-friendly. Grazie alla sua architettura modulare e alla sua flessibilità, Bootstrap consente di creare rapidamente interfacce utente accattivanti e funzionali.

Capitolo 4: Fase di analisi

Il presente capitolo si concentra sulla documentazione di analisi prodotta dal team al fine di chiarire le funzionalità e i servizi all'interno del sistema. Verranno presentati i diagrammi realizzati utilizzando il tool Visual Paradigm, che adotta la notazione UML (Unified Modeling Language).

4.1 Use Case Diagram



4.2 Sequence Diagram

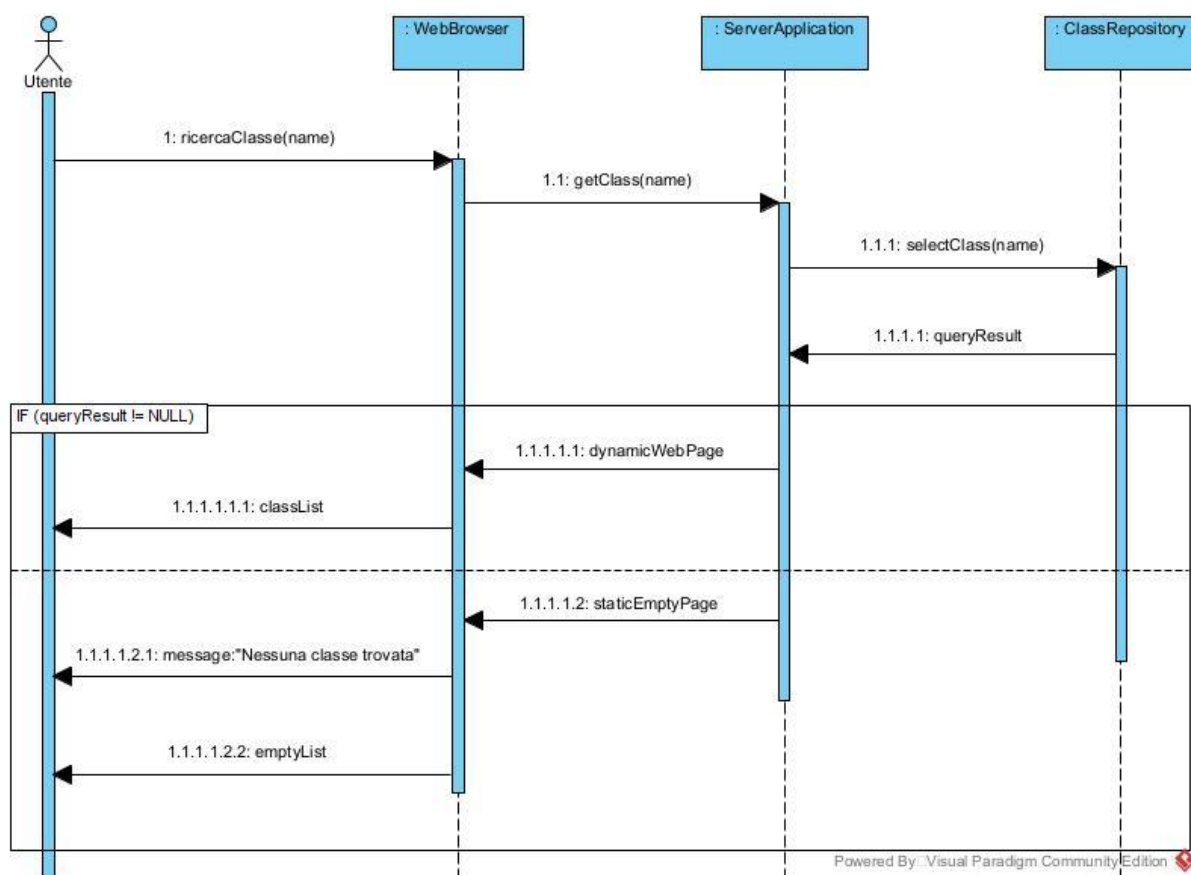
Si riportano i sequence diagram applicati in fase di analisi utilizzati per descrivere il comportamento del sistema. Tali diagrammi si concentrano sulla descrizione del comportamento del sistema, mostrando le interazioni tra attori e sistema nel tempo.

4.2.1 Sequence Diagram: RicercaClassi

- **Nome del caso d'uso:** Ricerca delle classi
- **Portata:** File Manager Man vs automated Testing Tools challenges
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente
- **Parti interessate:**
 - Utente (vuole ricercare una classe per nome all'interno del sistema)
- **Pre-condizioni:** L'utente è autenticato ed identificato
- **Garanzia di Successo:** Le classi presenti sono mostrate con successo e sono selezionabili
- **Scenario Principale di successo (Flusso Base)**
 1. L'utente chiede di ricercare una classe
 2. Il sistema richiede di inserire il nome della classe
 3. L'utente inserisce il nome della classe e dà la conferma
 4. Il sistema mostra la/le classe/i corrispondenti

Flussi alternativi:

- 4.a: Non è presente nessuna classe corrispondente al nome ricercato
 1. Il sistema mostra un elenco vuoto



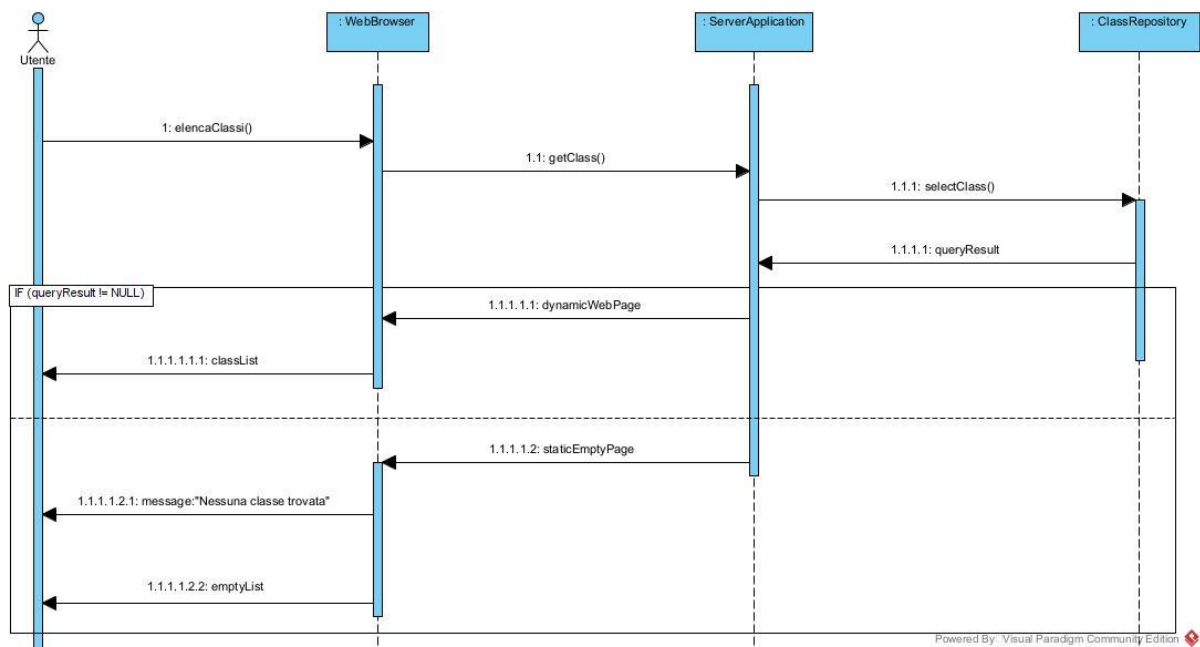
4.2.2 Sequence Diagram: elencaClassi

- **Nome del caso d'uso:** Elenco delle classi
- **Portata:** File Manager Man vs automated Testing Tools challenges

- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente
- **Parti interessate:**
 - Utente (vuole visualizzare tutte le classi all'interno del sistema)
- **Pre-condizioni:** L'utente è autenticato ed identificato
- **Garanzia di Successo:** Le classi presenti sono mostrate con successo e sono selezionabili
- **Scenario Principale di successo (Flusso Base)**
 1. L'utente chiede di visualizzare le classi
 2. Il sistema mostra la/le classe/i corrispondenti

Flussi alternativi:

- 2.a: Non è presente nessuna classe nel sistema
 1. Il sistema mostra un elenco vuoto



4.2.3 Sequence Diagram: ordinaClassi

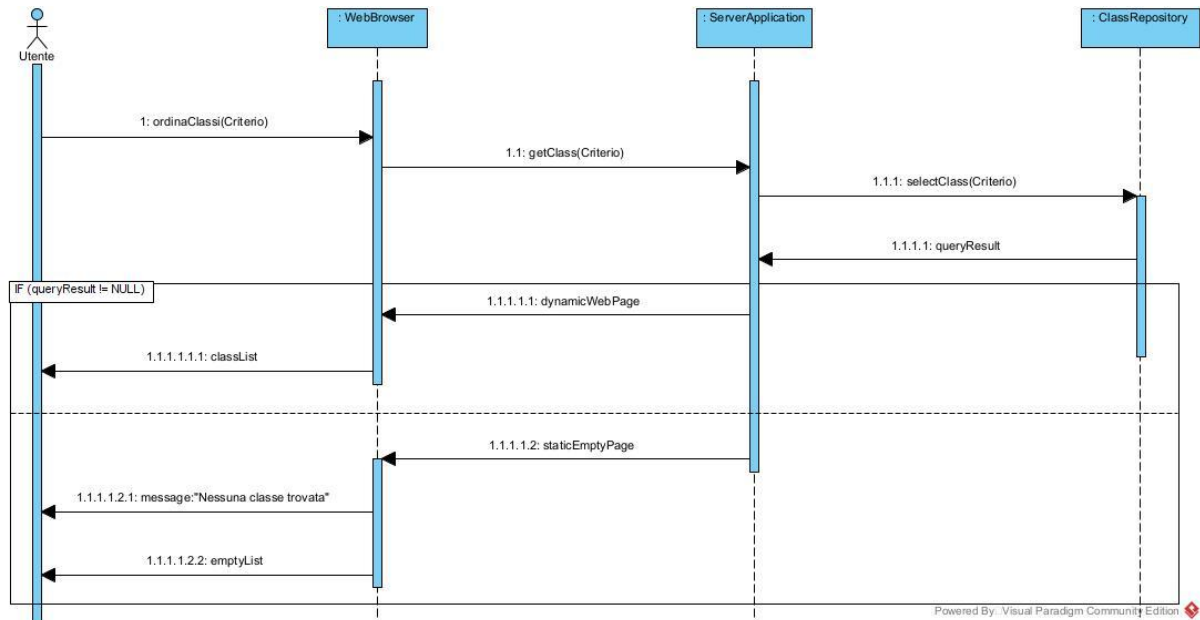
- **Nome del caso d'uso:** Ordinamento delle classi
- **Portata:** File Manager Man vs automated Testing Tools challenges
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente
- **Parti interessate:**
 - Utente (vuole ordinare le classi all'interno del sistema in base ad un criterio scelto)
- **Pre-condizioni:** L'utente è autenticato ed identificato
- **Garanzia di Successo:** Le classi presenti sono mostrate con successo e sono selezionabili
- **Scenario Principale di successo (Flusso Base)**
 1. L'utente chiede di ordinare le classi del sistema
 2. Il sistema richiede di inserire il criterio di ordinamento delle classi
 3. L'utente inserisce il criterio di ordinamento delle classi e dà la conferma

4. Il sistema mostra la/le classe/i corrispondenti

Flussi alternativi:

2.a: Non è presente nessuna classe nel sistema

1. Il sistema mostra un elenco vuoto



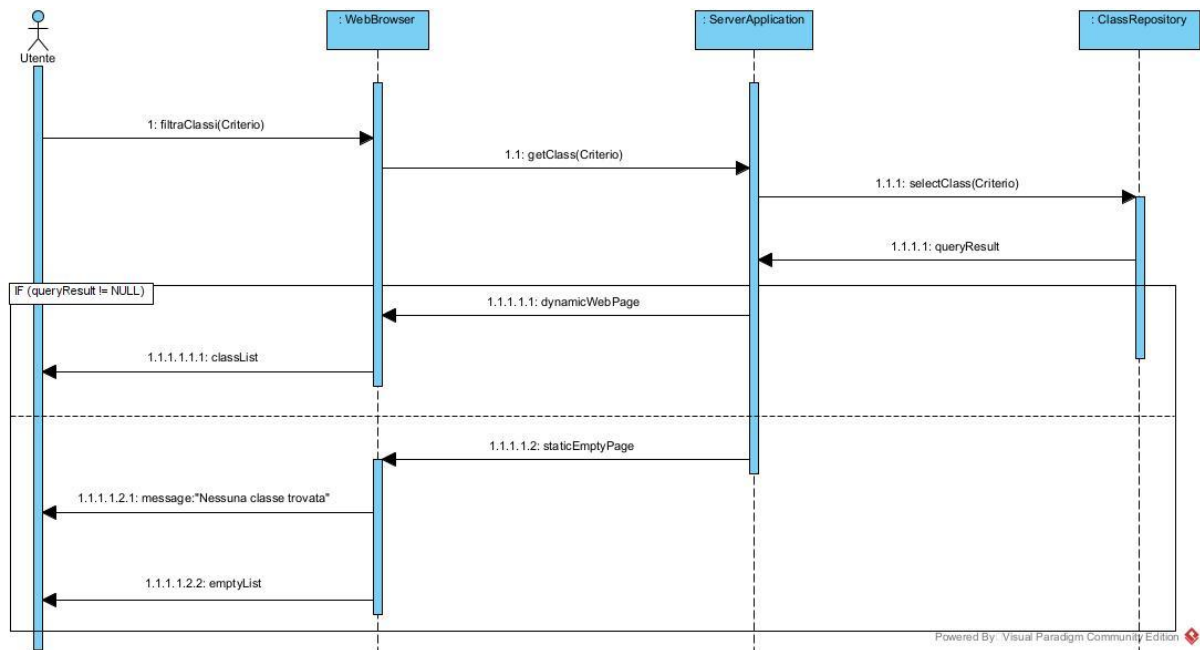
4.2.4 Sequence Diagram: filtraClassi

- **Nome del caso d'uso:** Filtraggio delle classi
- **Portata:** File Manager Man vs automated Testing Tools challenges
- **Livello:** Obiettivo Utente
- **Attore Primario:** Utente
- **Parti interessate:**
 - Utente (vuole filtrare le classi all'interno del sistema in base ad un criterio scelto)
- **Pre-condizioni:** L'utente è autenticato ed identificato
- **Garanzia di Successo:** Le classi presenti sono mostrate con successo e sono selezionabili
- **Scenario Principale di successo (Flusso Base)**
 1. L'utente chiede di filtrare le classi del sistema
 2. Il sistema richiede di inserire il criterio di filtraggio delle classi
 3. L'utente inserisce il criterio di filtraggio delle classi e da la conferma
 4. Il sistema mostra la/le classe/i corrispondenti

Flussi alternativi:

3.a: Non è presente nessuna classe corrispondente al nome ricercato

1. Il sistema mostra un elenco vuoto

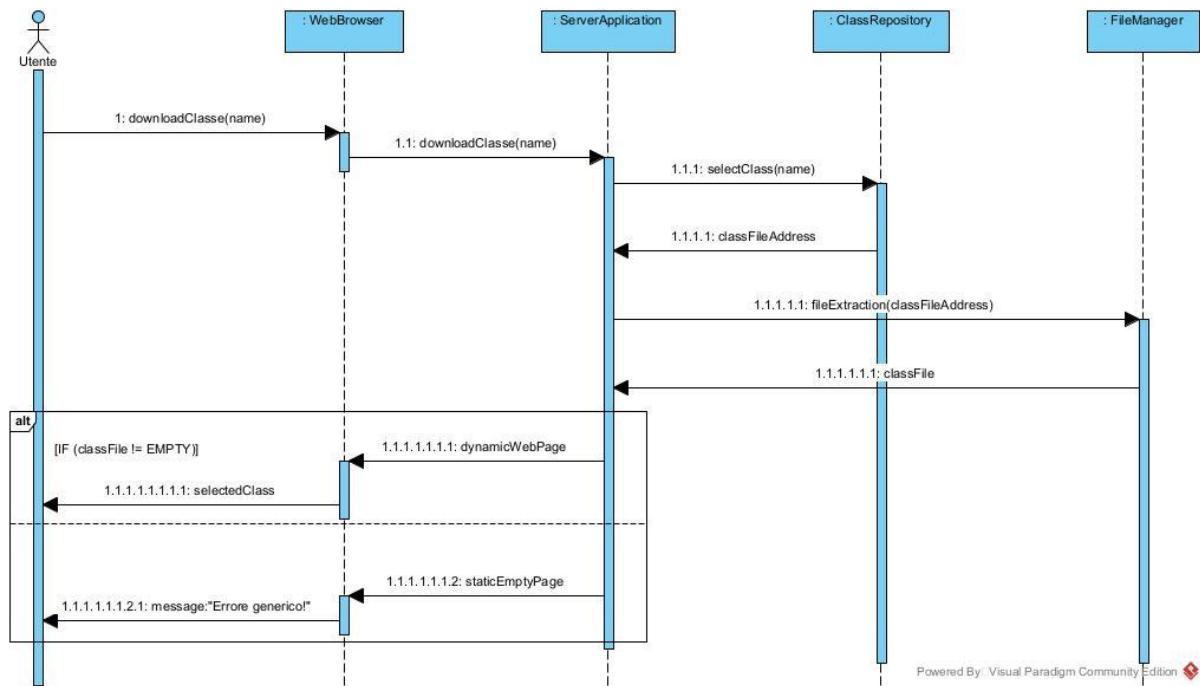


4.2.5 Sequence Diagram: DownloadClasse

- **Nome del Caso d'uso** : Download di una classe
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Utente
- **Attore Primario**: Utente
- **Parti interessate**:
 - Utente (vuole scaricare il codice di una classe)
- **Pre-condizioni**: L'utente è autenticato ed identificato
- **Garanzia di Successo**: Il file viene scaricato con successo , il download è registrato.
- **Scenario Principale di successo (Flusso Base)**
 1. L'utente seleziona la classe da scaricare
 2. L'utente conferma di voler effettuare il download
 3. Il sistema registra il download
 4. L'utente riceve il file
 5. L'utente continua la sua navigazione

Flussi alternativi:

- 3.a: Il sistema segnala un errore nel download.
 1. Il sistema non registra il download.
 2. Il sistema chiede all'utente di riprovare

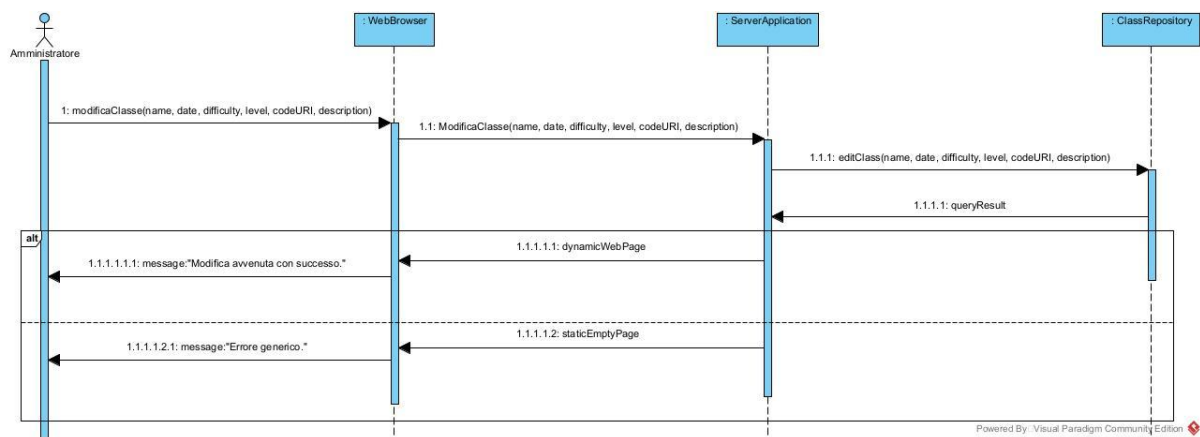


4.2.6 Sequence Diagram: ModificaClasse

- **Nome del Caso d'uso** : Modifica di una classe
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Amministratore
- **Attore Primario**: Amministratore
- **Parti interessate**:
 - Amministratore (vuole modificare le informazioni di una classe)
- **Pre-condizioni**: L'amministratore è autenticato ed identificato
- **Garanzia di Successo**: La classe viene modificata con successo.
- **Scenario Principale di successo (Flusso Base)**
 1. L'amministratore seleziona la classe da modificare
 2. L'amministratore inserisce le informazioni da modificare
 3. Il sistema controlla la validità dei campi
 4. Il sistema registra le modifiche
 5. L'amministratore continua la sua navigazione

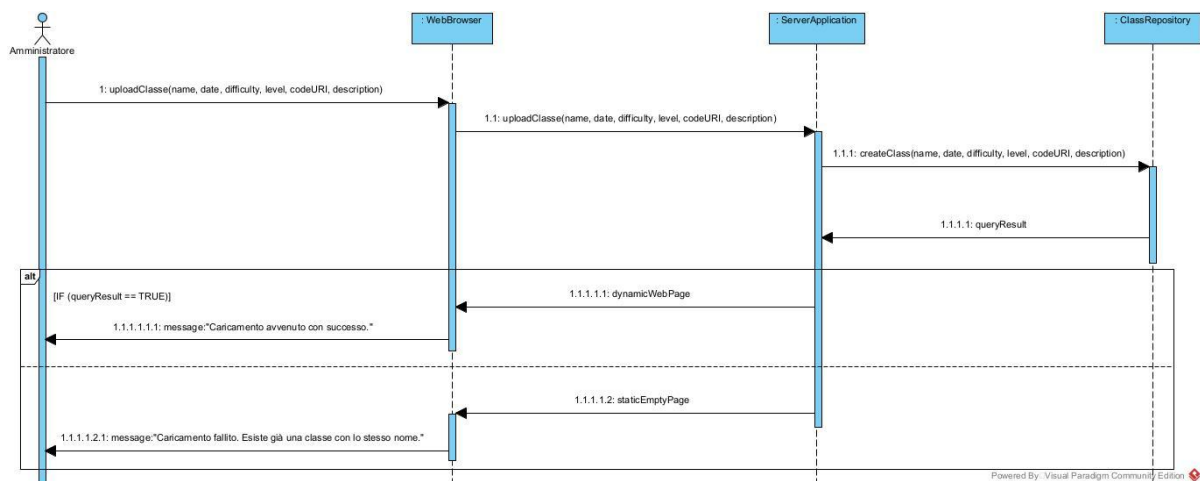
Flussi alternativi:

- 3.a: Il sistema segnala un errore nell'inserimento dei campi.
- b. Il sistema non registra la modifica..
- c. Il sistema chiede all'amministratore di riprovare



4.2.7 Sequence Diagram: UploadClasse

- **Nome del Caso d'uso** : Upload di una classe
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Amministratore
- **Attore Primario**: Amministratore
- **Parti interessate**:
 - Amministratore (vuole caricare una nuova classe nel sistema)
- **Pre-condizioni**: L'amministratore è autenticato ed identificato
- **Garanzia di Successo**: La classe viene caricata con successo.
- **Scenario Principale di successo (Flusso Base)**
 1. L'amministratore chiede di caricare una classe
 2. Il sistema richiede di inserire il nome, la data, la difficoltà, la descrizione, le categorie e il file della classe
 3. L'amministratore inserisce le informazioni richieste e il file della classe e da la conferma
 4. Il sistema mostra la/le classe/i corrispondenti
- **Flussi alternativi**:
 - 3.a: Il sistema segnala un errore nell'inserimento dei campi.
 - b. Il sistema non registra l'inserimento
 - c. Il sistema chiede all'amministratore di riprovare



4.2.8 Sequence Diagram: eliminaClasse

- **Nome del Caso d'uso** : Eliminazione di una classe
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Amministratore
- **Attore Primario**: Amministratore
- **Parti interessate**:
 - Amministratore (vuole eliminare una classe nel sistema)
- **Pre-condizioni**: L'amministratore è autenticato ed identificato
- **Garanzia di Successo**: La classe viene eliminata con successo.
- **Scenario Principale di successo (Flusso Base)**
 1. L'amministratore chiede di eliminare una classe
 2. Il sistema richiede la conferma di eliminazione della classe
 3. L'amministratore conferma di voler effettuare l'eliminazione
 4. Il sistema elimina la classe corrispondente

Flussi alternativi:

- 4.a: Il sistema segnala un errore generico nell'eliminazione di una classe.
- b. Il sistema non registra l'eliminazione

4.2.9 Sequence Diagram: elencaReport

- **Nome del caso d'uso**: Elenco dei report
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Amministratore
- **Attore Primario**: Amministratore
- **Parti interessate**:
 - Amministratore (vuole visualizzare tutti i report all'interno del sistema)
- **Pre-condizioni**: L'amministratore è autenticato ed identificato
- **Garanzia di Successo**: I report presenti sono mostrati con successo e sono selezionabili
- **Scenario Principale di successo (Flusso Base)**
 1. L'amministratore chiede di visualizzare i report
 2. Il sistema mostra i report disponibili

Flussi alternativi:

- 2.a: Non è presente nessun report nel sistema
- b. Il sistema mostra un elenco vuoto

4.2.10 Sequence Diagram: nuovoLike

- **Nome del Caso d'uso** : Inserimento di un like ad una classe
- **Portata**: File Manager Man vs automated Testing Tools challenges
- **Livello**: Obiettivo Studente
- **Attore Primario**: Studente
- **Parti interessate**:
 - Studente (vuole esprimere un gradimento per una classe nel sistema)
- **Pre-condizioni**: Lo studente è autenticato ed identificato
- **Garanzia di Successo**: Il "Mi piace" viene salvato e incrementato per la classe scelta con successo.

- **Scenario Principale di successo (Flusso Base)**

1. Lo studente chiede di inserire il “Mi piace” ad una classe
2. Il sistema richiede la conferma di inserimento “Mi piace” per la classe
3. Lo studente conferma di voler effettuare l’inserimento
4. Il sistema incrementa il numero di “Mi piace” per la classe corrispondente

Flussi alternativi:

- 4.a: Il sistema segnala un errore generico nell’inserimento di un “Mi piace”.
- b. Il sistema non registra l’inserimento

4.2.11 Sequence Diagram: nuovoReport

- **Nome del Caso d’uso** : Nuovo Report

- **Portata:** File Manager Man vs automated Testing Tools challenges

- **Livello:** Obiettivo Studente

- **Attore Primario:** Studente

- **Parti interessate:**

-Studente (vuole inserire un nuovo report nel sistema)

- **Pre-condizioni:** Lo studente autenticato ed identificato

- **Garanzia di Successo:** Il report viene caricato con successo.

- **Scenario Principale di successo (Flusso Base)**

1. Lo studente chiede di caricare un nuovo report
2. Il sistema richiede di inserire la motivazione del report
3. Lo studente inserisce la motivazione e dà la conferma
4. Lo studente continua la propria navigazione

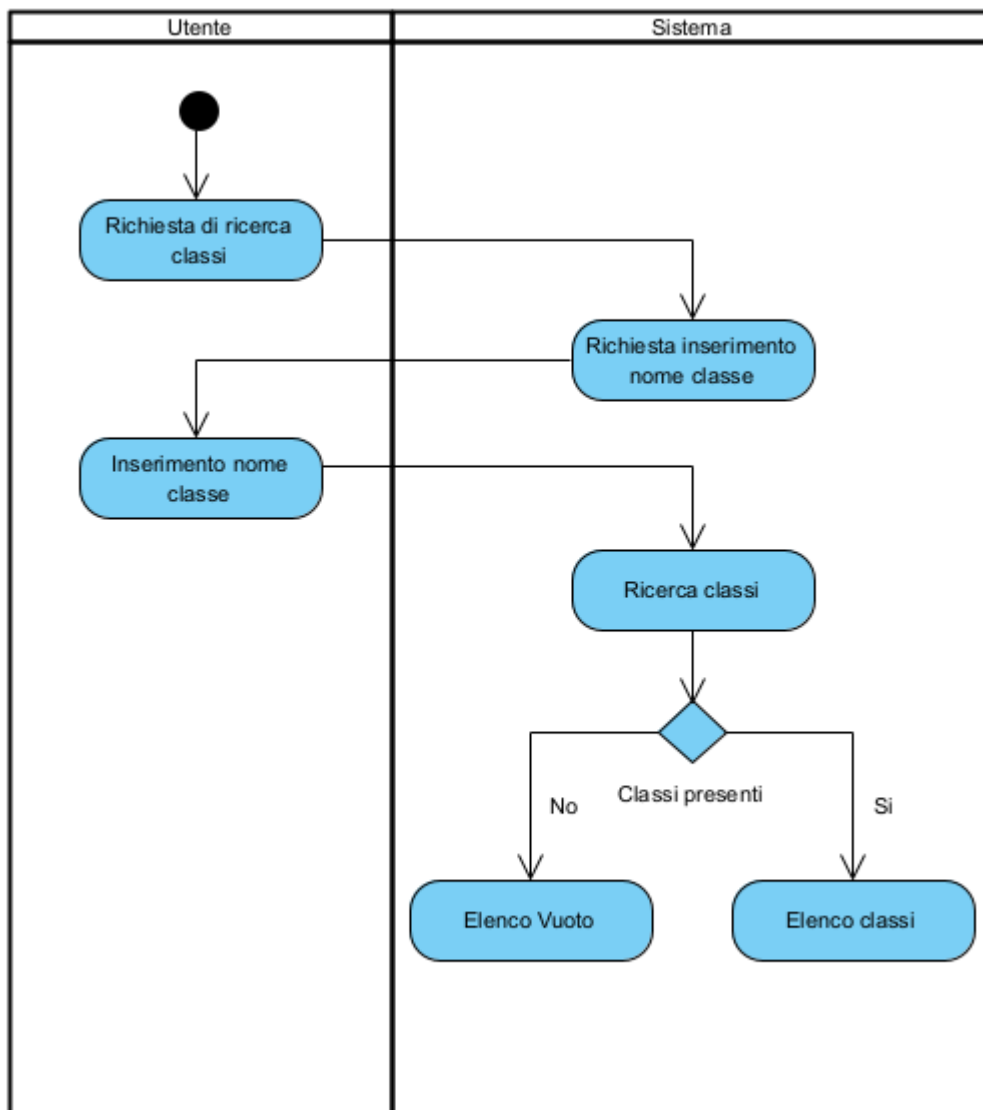
Flussi alternativi:

- 3.a: Il sistema segnala un errore generico nell’inserimento.
- b. Il sistema non registra il report
- c. Il sistema chiede allo studente di riprovare

4.3 Activity Diagram

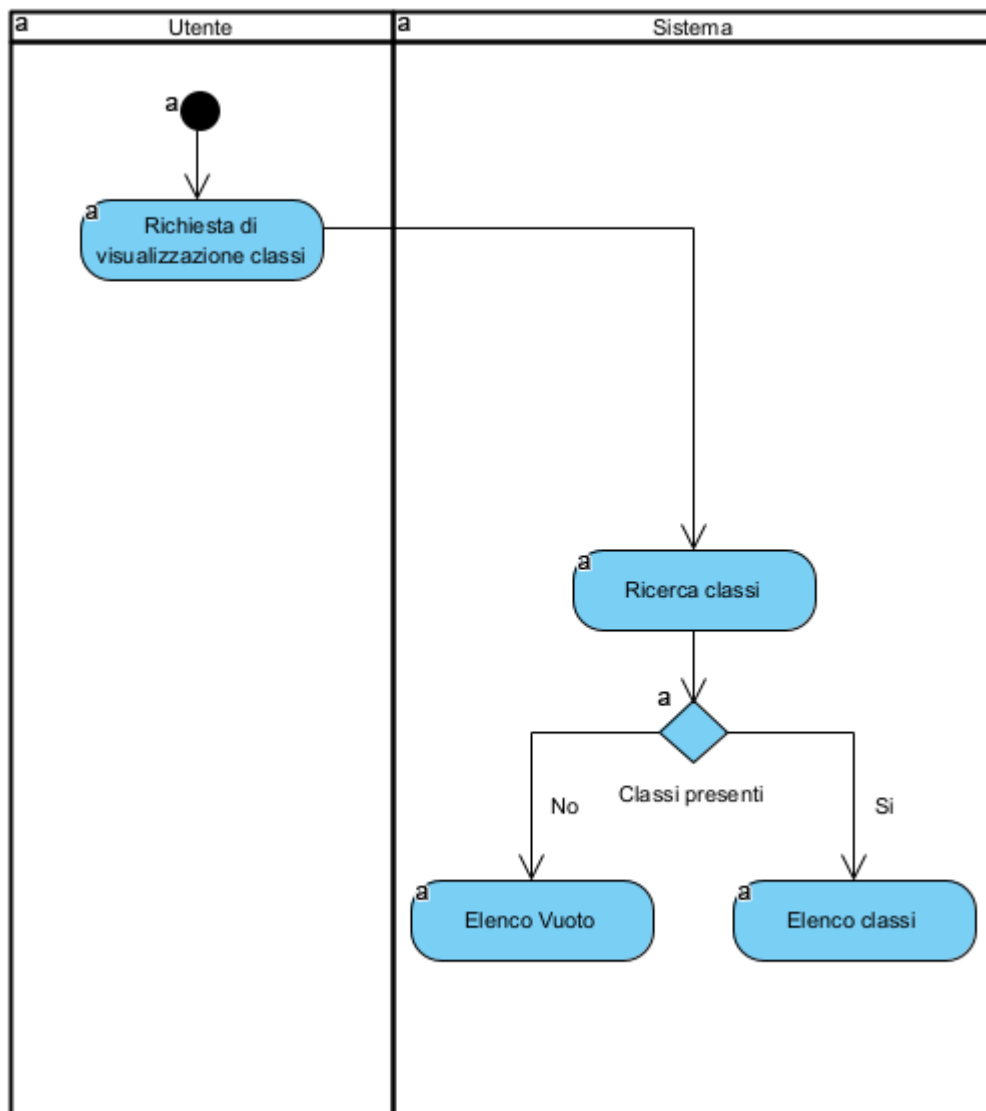
Per offrire una panoramica del flusso di attività di ciascun caso d'uso e dell'interazione tra l'attore e il sistema, sono presentati di seguito i seguenti diagrammi di attività:

4.3.1 Activity Diagram: RicercaClassi



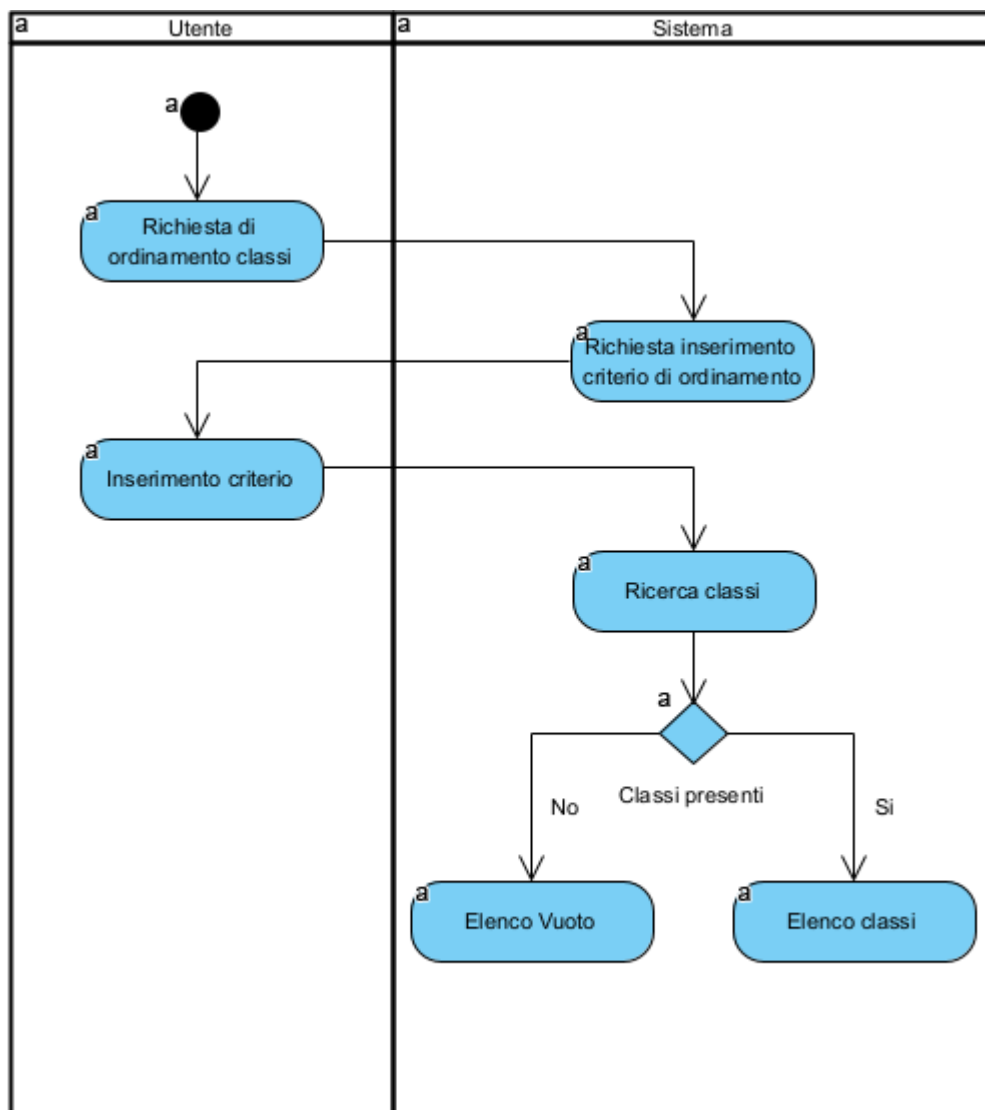
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Ricerca delle classi". Sono state definite due partizioni: la prima rappresenta l'utente che vuole effettuare la ricerca di una classe, la seconda il sistema che richiede l'inserimento del nome della classe da ricercare. Se la ricerca non riscontra risultati nel database viene mostrato un elenco vuoto altrimenti un elenco pieno.

4.3.2 Activity Diagram: elencaClassi



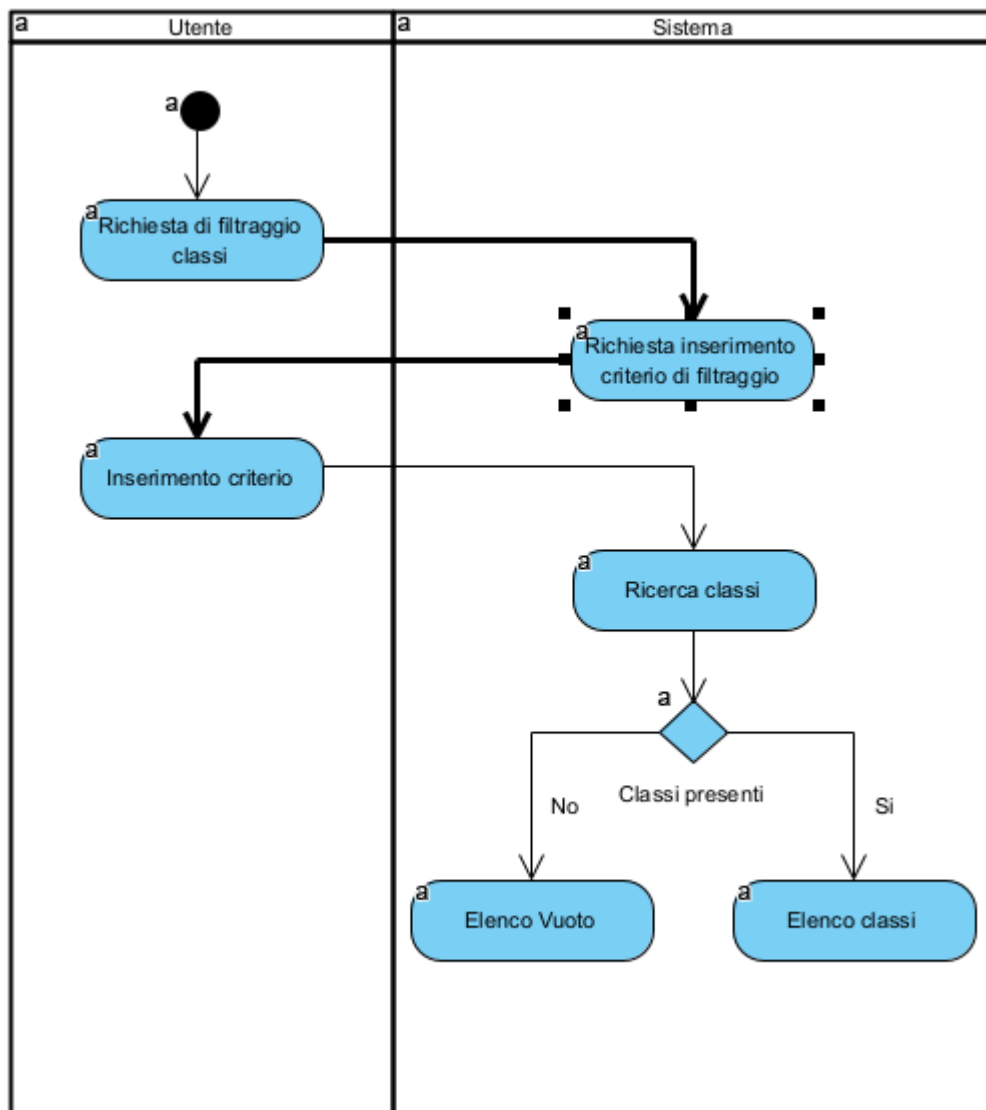
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Elenca classi". Sono state definite due partizioni: la prima rappresenta l'utente che vuole visualizzare le classi presenti nel sistema, la seconda il sistema che effettua la ricerca. Se la ricerca non riscontra risultati nel database viene mostrato un elenco vuoto altrimenti un elenco pieno.

4.3.3 Activity Diagram: ordinaClassi



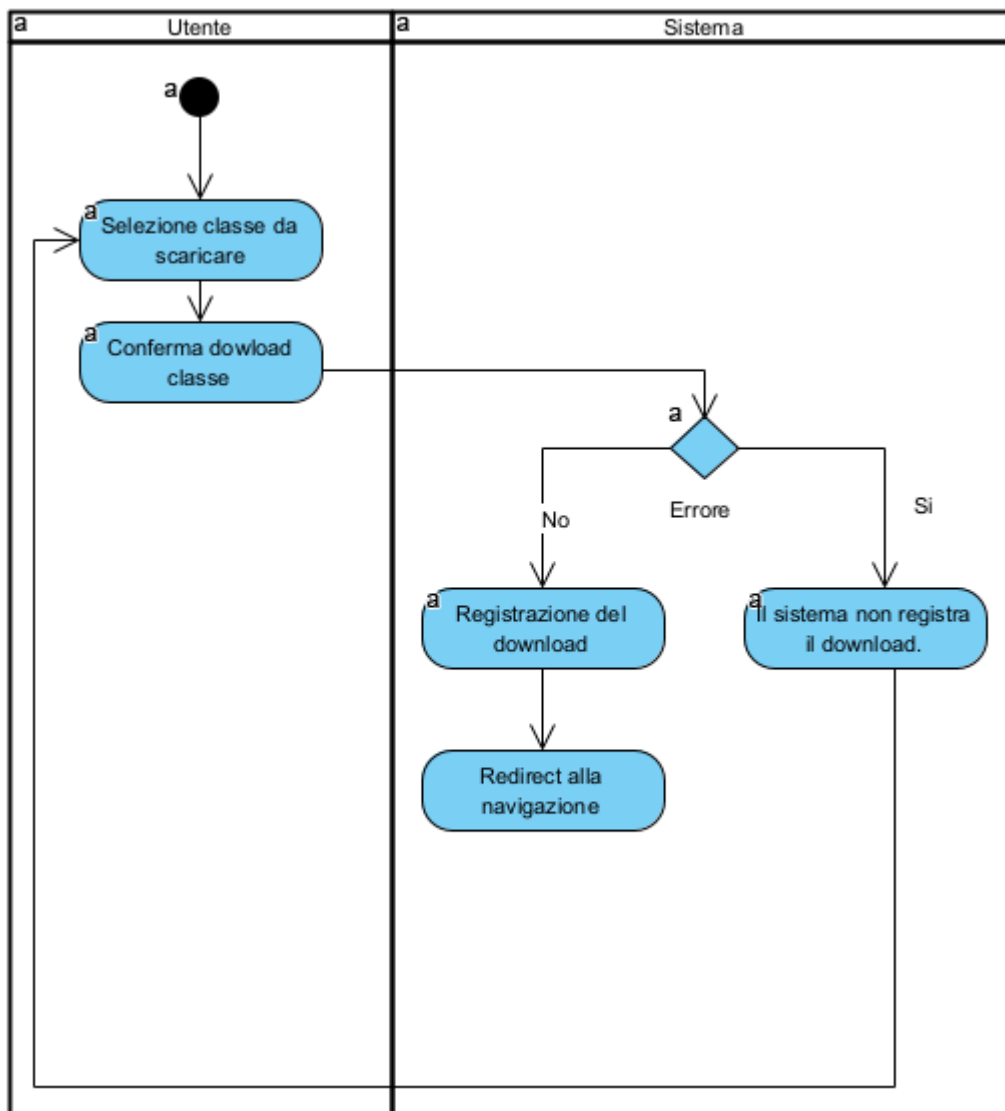
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Ordinamento delle classi". Sono state definite due partizioni: la prima rappresenta l'utente che vuole ordinare le classi da visualizzare, la seconda il sistema che richiede l'inserimento del criterio di ordinamento della classe da applicare. Se la ricerca non riscontra risultati nel database viene mostrato un elenco vuoto altrimenti un elenco pieno.

4.3.4 Activity Diagram: filtraClassi



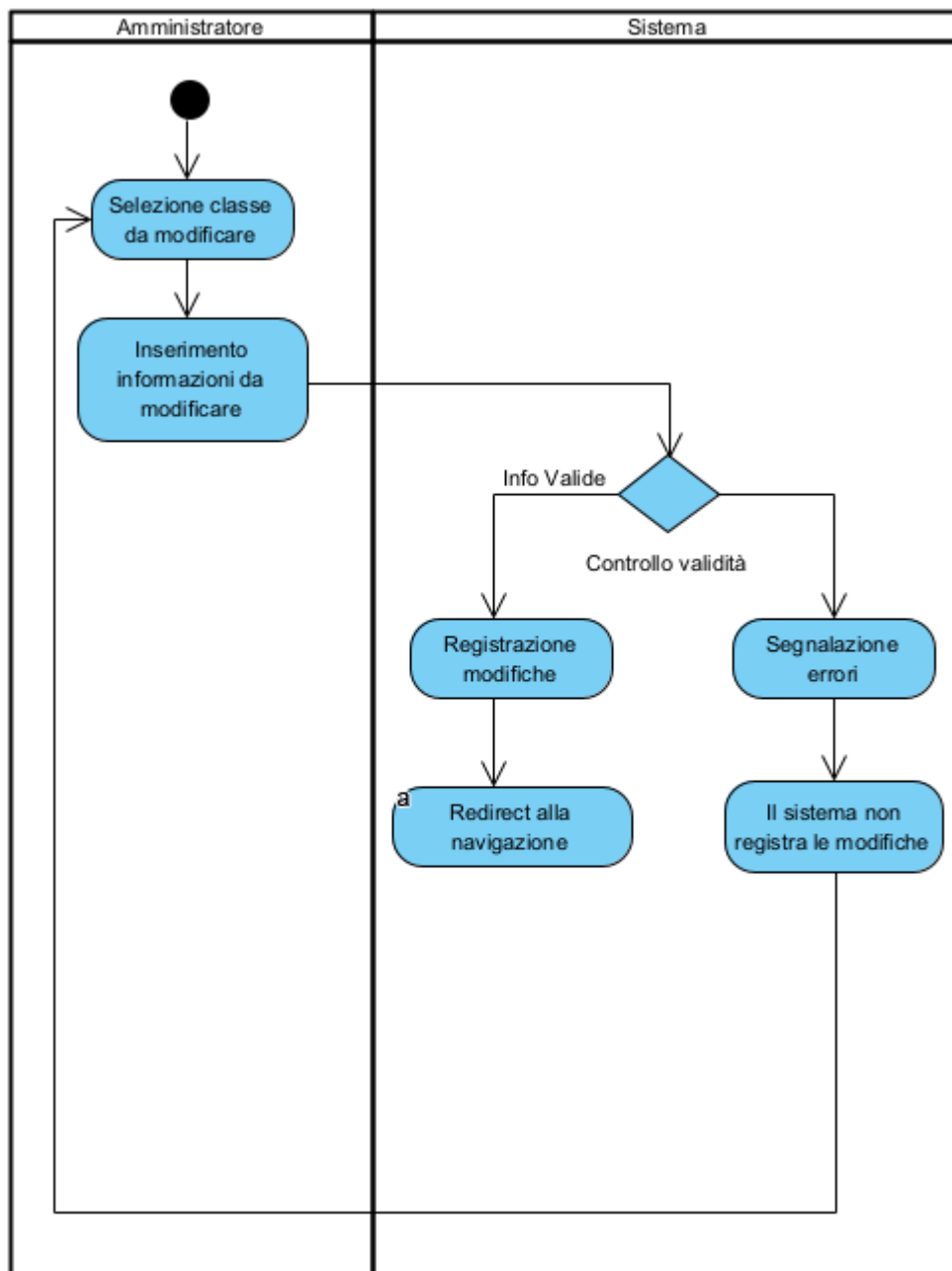
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Filtraggio delle classi". Sono state definite due partizioni: la prima rappresenta l'utente che vuole effettuare il filtraggio delle classi, la seconda il sistema che richiede l'inserimento del criterio di filtraggio della classe da applicare. Se la ricerca non riscontra risultati nel database viene mostrato un elenco vuoto altrimenti un elenco pieno.

4.3.5 Activity Diagram: DownloadClasse



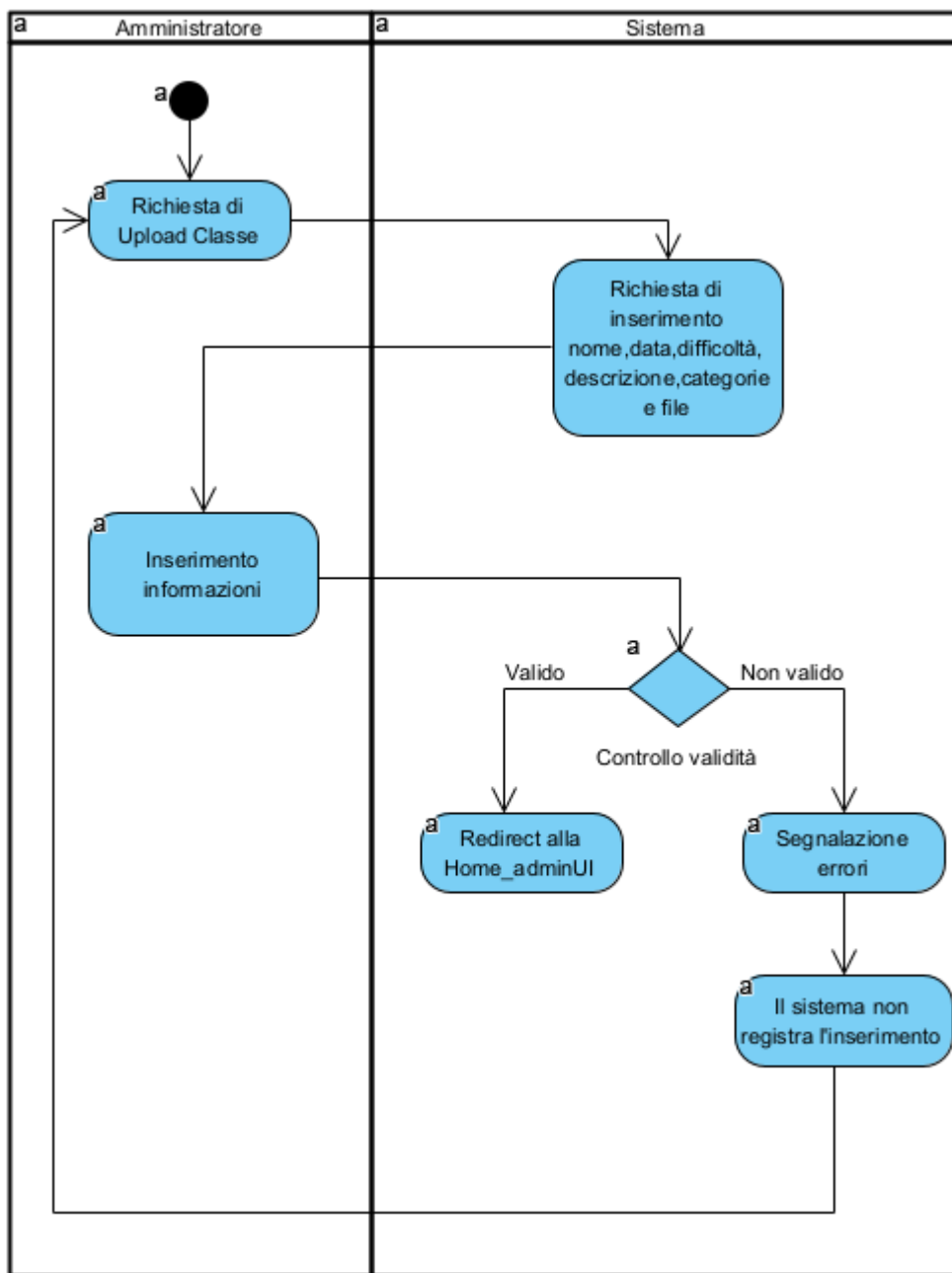
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Download di una classe". Sono state definite due partizioni: la prima rappresenta l'utente che vuole effettuare il download, la seconda il sistema che controlla se il download va a buon fine senza errori. Se dovessero esserci errori il sistema chiede all'utente di riprovare.

4.3.6 Activity Diagram: ModificaClasse



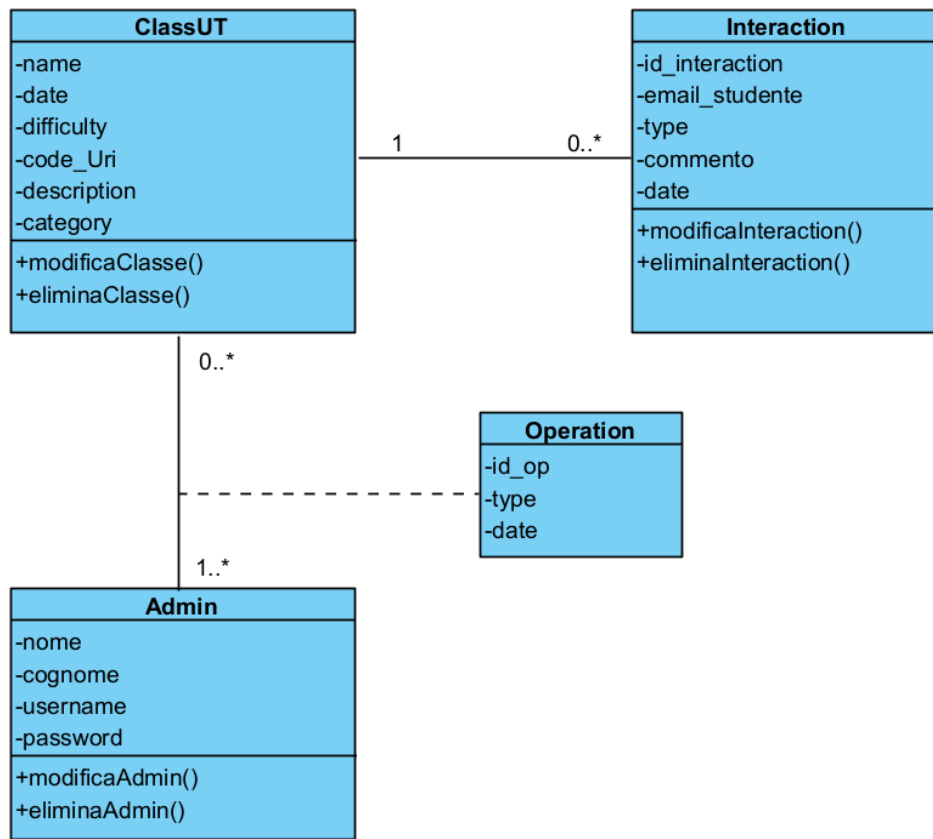
Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Modifica di una classe". Sono state definite due partizioni: la prima rappresenta l'amministratore che vuole effettuare la modifica, la seconda il sistema che controlla se l'inserimento delle informazioni sono corrette. Se dovessero esserci errori il sistema chiede all'amministratore di riprovare.

4.3.7 Activity Diagram: UploadClasse



Il diagramma di attività rappresenta il flusso delle azioni coinvolte nel caso d'uso "Upload di una classe". Sono state definite due partizioni: la prima rappresenta l'amministratore che vuole effettuare l'upload, la seconda il sistema che controlla se l'inserimento delle informazioni sono corrette. Se dovessero esserci errori il sistema chiede all'amministratore di riprovare.

4.4 System Domain Model



Il System Domain Model (SDM) è un modello di dominio che rappresenta gli oggetti, i concetti e le interazioni all'interno del dominio del sistema, fornendo una visione ad alto livello del sistema software. Il SDM descrive i concetti del sistema e le relazioni tra di essi, senza specificare ancora come questi saranno implementati come classi e oggetti nel sistema software.

Nel sistema in questione, il System Domain Model (SDM) rappresenta le entità principali del sistema attraverso due classi: **ClassUT**, **Interaction**, **Operation** e **Admin**.

La classe **ClassUT** modella le classi testabili disponibili sul sistema e contiene attributi che descrivono le loro proprietà. Tra gli attributi, ci sono il nome della classe, la data di creazione, l'indirizzo del file associato, la descrizione e la categoria. Questi attributi consentono di identificare e descrivere le classi disponibili nel sistema.

La classe **ClassUT** contiene anche le responsabilità che definiscono le azioni che possono essere eseguite su di essa. Tra le responsabilità sono presenti la modifica e l'eliminazione di un'istanza di una classe.

La classe **Interaction** modella le interazioni tra studenti e classi e contiene attributi che descrivono le sue proprietà. Tra gli attributi, c'è l'attributo `type`, che si riferisce al tipo di interazione che può essere `like` o `report`. Questo attributo consente di identificare il tipo di interazione che un utente del sistema ha con una classe specifica.

La classe **Admin** modella gli amministratori del sistema e contiene attributi che descrivono le loro proprietà. Tra gli attributi, ci sono il nome, il cognome, lo username e la password. Questi attributi consentono di identificare e autenticare gli amministratori del sistema.

La classe Admin ha le responsabilità di gestire e monitorare il sistema. Tra le responsabilità ci sono la creazione, l'aggiornamento e la cancellazione di classi testabili.

Admin e ClassUT sono collegati dalla classe associativa **Operation** che tiene traccia di tutte le operazioni di inserimento, modifica e cancellazione che un admin può fare sulle classi del sistema.

Capitolo 5: Fase di Progettazione

La fase di progettazione del sistema è stata caratterizzata dalla scelta delle strategie e delle soluzioni architetturali da adottare per soddisfare gli obiettivi e i requisiti del software. Durante questa fase, sono stati utilizzati i diagrammi di analisi per definire la struttura dei componenti e i flussi di dati all'interno del sistema.

Inoltre, si è redatta una documentazione di progettazione dettagliata in grado di fornire una chiara visione degli obiettivi del sistema, delle scelte architetturali adottate e delle interazioni tra le varie parti del software.

5.1 Pattern Architetturale MVC

Il pattern architetturale MVC (Model-View-Controller) è un approccio strutturale per progettare e sviluppare applicazioni software. Questo pattern separa logicamente le responsabilità all'interno di un'applicazione in tre componenti principali: il Modello (Model), la Vista (View) e il Controllore (Controller). Ognuno di questi componenti svolge un ruolo specifico e collabora insieme per creare un'architettura ben strutturata e scalabile.

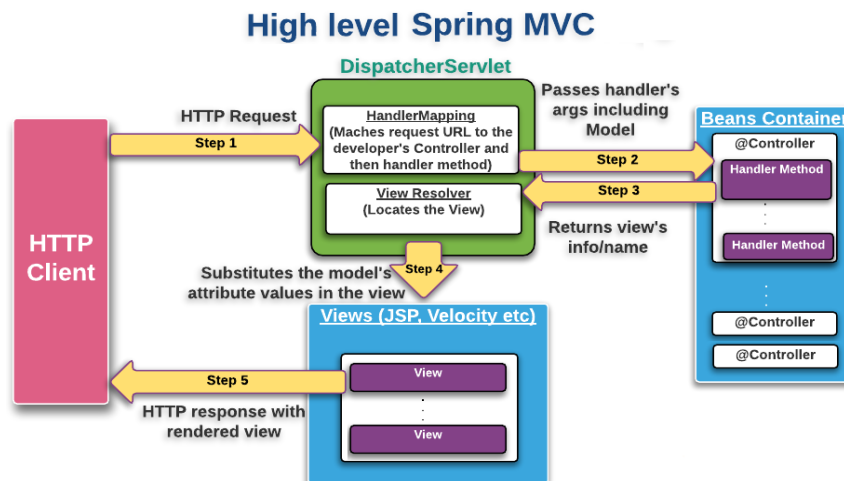
5.1.1 Spring MVC

Spring MVC è un framework basato su Spring Framework che offre un'implementazione del pattern architetturale MVC per lo sviluppo di applicazioni web Java. Spring MVC semplifica la creazione di applicazioni web scalabili, modulari e mantenibili, fornendo una struttura solida per organizzare il codice e separare le responsabilità.

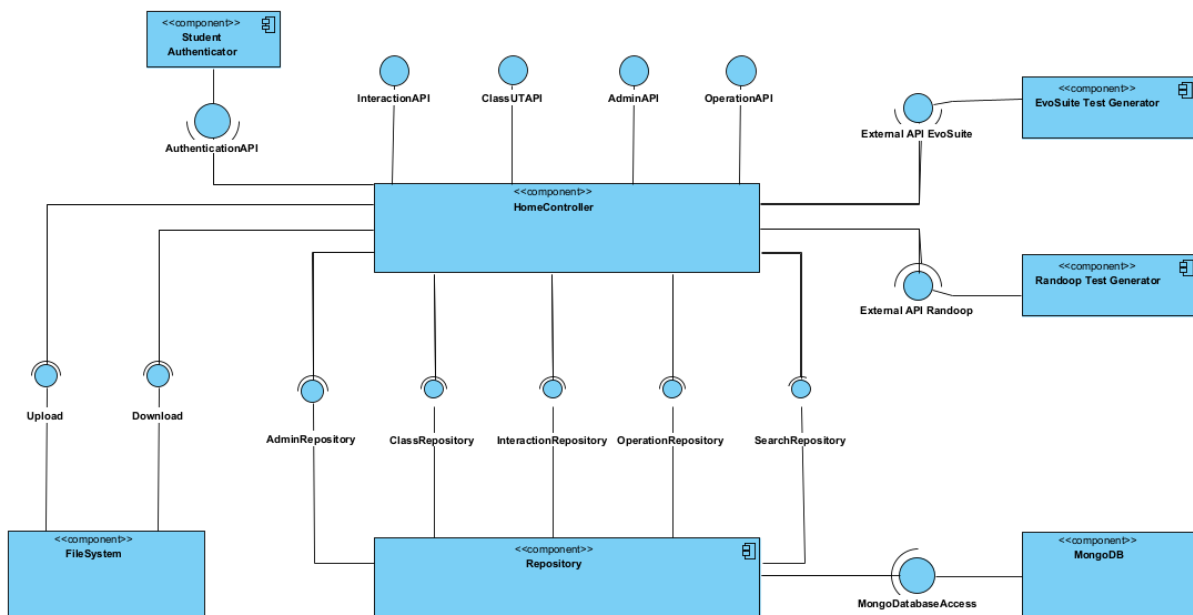
I vantaggi principali di Spring MVC sono:

- **Separazione delle responsabilità:** Spring MVC adotta il pattern architetturale MVC, che consente una chiara separazione delle responsabilità tra il Modello, la Vista e il Controllore. Questo favorisce la modularità e la manutenibilità del codice, facilitando la comprensione e l'evoluzione dell'applicazione nel tempo.
- **Configurazione dichiarativa:** Spring MVC utilizza un approccio basato su annotazioni o file di configurazione XML per definire il comportamento dell'applicazione. Questo consente di gestire in modo dichiarativo le interazioni tra i componenti, riducendo la complessità e la quantità di codice da scrivere.

- **Scalabilità e performance:** Spring MVC è progettato per fornire prestazioni elevate e scalabilità. Offre meccanismi per la memorizzazione nella cache, la gestione delle richieste concorrenti e l'ottimizzazione delle risorse, consentendo di costruire applicazioni web efficienti e scalabili.
- **Gestione delle richieste e delle risposte:** fornisce un sistema di gestione delle richieste e delle risposte che consente di mappare richieste HTTP a metodi specifici del controller utilizzando annotazioni come `@RequestMapping`.



5.2 Component Diagram



Il Component Diagram è un utile strumento per descrivere l'architettura di un sistema software e le interazioni tra i suoi componenti. Analizzando il Component Diagram di un sistema, è possibile avere una visione ad alto livello della struttura del sistema e

delle relazioni tra i suoi componenti. In particolare, i componenti principali del sistema e le loro interfacce, nonché le dipendenze tra di essi, possono essere esplorati e valutati.

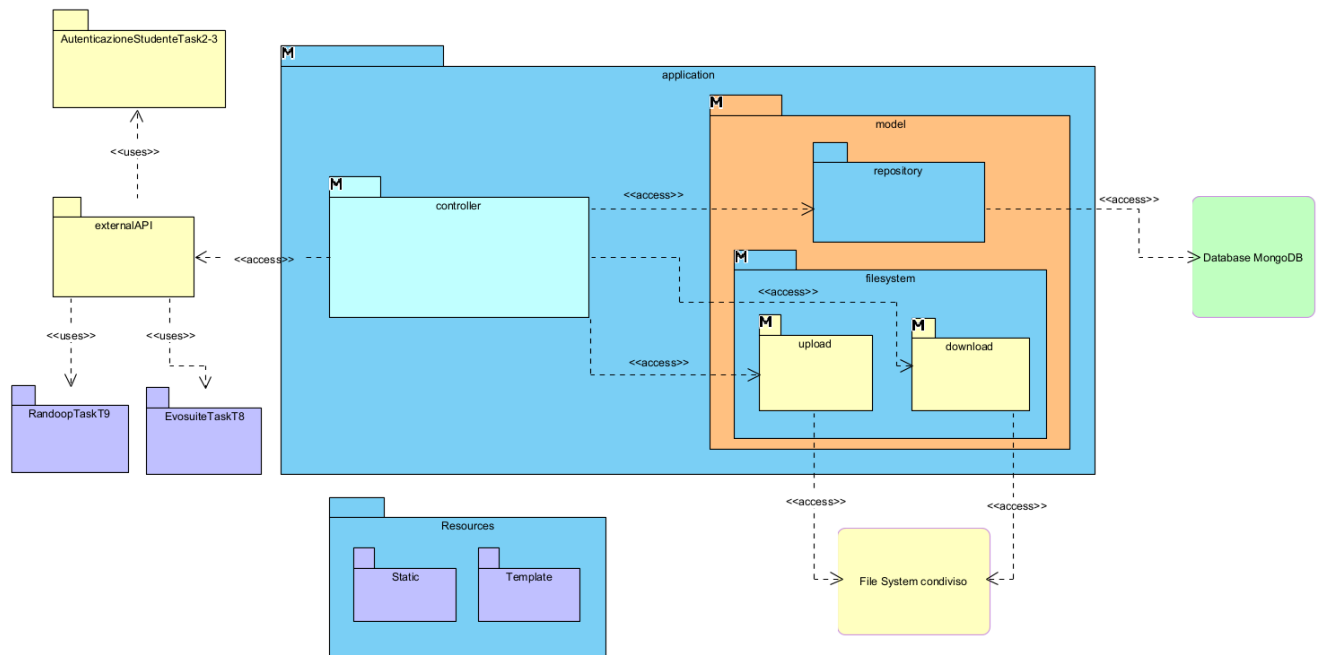
Esso presenta i seguenti componenti:

- **HomeController** è il componente centrale del sistema, responsabile di fornire API per le operazioni di inserimento, cancellazione, modifica e visualizzazione delle Classi UT, nonché delle interazioni tra studenti e classi, dei dati degli amministratori e delle operazioni che questi ultimi possono effettuare sulle classi. Questo componente si interfaccia con la MongoRepository attraverso le interfacce fornite dal componente Repository e con il Filesystem tramite le interfacce di download e upload. HomeController si interfaccia con altri componenti esterni, come RandoopTest Generator, EvosuiteTestGenerator e Student Authenticator, attraverso delle interfacce esterne.
- **Repository** è responsabile di interfacciarsi con il MongoDB e di fornire una serie di interfacce per le operazioni CRUD e le operazioni di ricerca e filtraggio delle collection presenti sul Database. Il MongoDB, a sua volta, è il database fisico su cui vengono memorizzati i metadati relativi alle classi, le informazioni degli amministratori e le interazioni e le operazioni che coinvolgono le classi.
- **FileSystem** è responsabile della gestione della directory condivisa su cui vengono memorizzati fisicamente i file .java relativi alle classi under test. Esso espone le interfacce per il download e l'upload sulla repository.
- **Lo Student Authenticator** viene utilizzato per richiedere i parametri dello studente attualmente loggato nella sessione attraverso l'interfaccia AuthenticatorAPI.
- **EvosuiteTestGenerator** viene utilizzato per la generazione automatica dei Test Evosuite una volta caricata una nuova classe da testare.
- **RandoopTestGenerator** viene utilizzato per la generazione automatica dei Test Randoop una volta caricata una nuova classe da testare.

5.3 Package Diagram

Il package diagram è uno strumento utilizzato per mostrare l'organizzazione e le dipendenze tra i package in un sistema o un'applicazione software. In particolare,

fornisce una panoramica del raggruppamento dei diversi package nel sistema e delle loro interazioni.



Model: In un'applicazione Spring MVC, il package model contiene le classi che rappresentano i dati dell'applicazione e forniscono i metodi per accedere e modificare questi dati. Le classi del package model sono utilizzate dal Controller per interagire con il Model e dal View per visualizzare i dati.

Il package model contiene dunque le classi del dominio (ClassUT, Admin, Operation, Interaction).

Il package model contiene inoltre altri package per l'interazione con il Database e il Filesystem

- Repository è il package per l'interazione con MongoDB contiene le classi che gestiscono l'accesso ai dati nel database MongoDB utilizzando l'interfaccia MongoRepository di Spring.
- FileSystem contiene i package Download e Upload per scaricare e caricare file sulla repository condivisa, un package per l'interazione con la MongoRepository e un package per l'interazione con il File System condiviso (che include a sua volta packages che forniscono funzionalità per l'upload ed il download dei file .java).

View: In un'applicazione basata sul pattern architetturale MVC, la View rappresenta la componente responsabile della presentazione dell'interfaccia utente e della visualizzazione dei dati provenienti dal Model. La View ha il compito di mostrare le informazioni in un formato comprensibile e interattivo per l'utente. Essa si occupa di rendere visibili gli elementi dell'interfaccia utente, come pulsanti, caselle di testo, form, consentendo all'utente di interagire con essi.

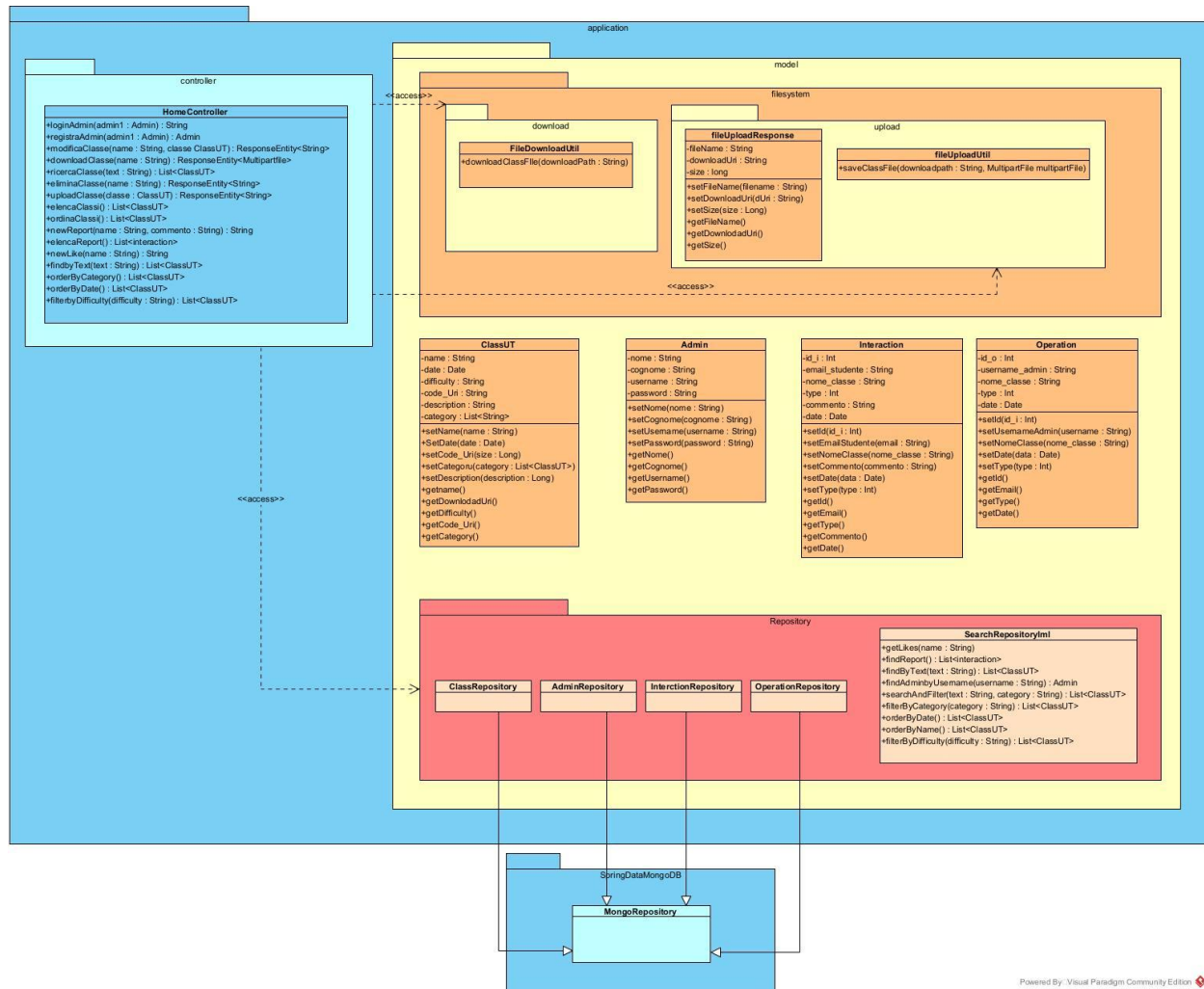
Le View sono contenute nel package "Resources", in particolar modo sono presenti 7 interfacce Web implementate con file .html che sono rivolte agli utenti amministratori e agli studenti.

Controller:In un'applicazione Spring MVC, il Controller è il componente che si occupa di gestire le richieste HTTP inviate dal client e di coordinare le operazioni tra il Model e la View.

Il Controller riceve le richieste HTTP dal client e le elabora, utilizzando i dati presenti nel Model per generare la risposta da inviare al client. In pratica, il Controller riceve i dati dal client, li elabora e li passa al Model per la memorizzazione e la gestione dei dati. Successivamente, il Controller utilizza i dati dal Model per generare la risposta da inviare al client, utilizzando la View per la presentazione dei dati.

externalAPI:Serve a utilizzare le API per l'interfacciamento con il software per la generazione dei Test Randoop ed Evosuite delle classi appena caricate e per richiedere informazioni sull'utente appena loggato al servizio di Autenticazione dello studente

5.4 Application Class Diagram



Model:

Nel Package Model sono presenti le Classi ClassUT , Admin , Interaction ed Operation.

- La classe ClassUT contiene gli attributi :
name,data,difficulty,code_Uri,description e category e i metodi get e set per gli attributi.
- La classe Admin contiene gli attributi : nome,cognome,username,password e i metodi get e set per gli attributi.
- La classe Interaction contiene gli attributi
:id_interaction,email_studente,nome_classe,commento,type,date e i metodi get e set per gli attributi.
- La classe Operation contiene gli attributi:id_operation,username_admin,nome_classe,type,date e i metodi get e set per gli attributi

FileSystem:

Nel Package FileSystem sono presenti i package download e upload

- Download contiene la classe FileDownloadUtil che fornisce il metodo “downloadClassFile” per recuperare un file contenuto nel filesystem condiviso
- Upload contiene la classe fileUploadResponse che rappresenta il descrittore del file inserito (attributi: fileName, codeUri, size) e fileUploadUtil che fornisce il metodo saveClassFile per inserire il file nel filesystem condiviso.

Repository:

Nel package Repository ci sono le classi per l'interazione con il MongoDB.

Le classi ClassRepository, AdminRepository, InteractionRepository e ControlRepository estendono la classe MongoRepository, che fornisce una serie di metodi per garantire le operazioni CRUD con il Database.

La classe SearchRepository contiene i metodi per effettuare delle query più complesse, che comprendono delle pipe contenenti operazioni di ricerca, ordinamento e filtraggio dei documenti presenti nelle collection del MongoDB.

5.5 Deployment

La fase di deployment, o distribuzione, è un passaggio essenziale nel ciclo di vita di un'applicazione, in cui l'applicazione viene resa disponibile agli utenti finali attraverso un ambiente di produzione online. Questo processo implica una serie di attività complesse, tra cui la preparazione dell'ambiente di hosting, la configurazione dei server, l'installazione dei software necessari e la gestione delle dipendenze. Un deployment ben eseguito è fondamentale per garantire l'affidabilità, la scalabilità e le prestazioni ottimali dell'applicazione, assicurando un'esperienza utente positiva.

Il diagramma di deployment specifica come l'applicazione viene distribuita e allocata all'interno dei componenti hardware e software. In questo caso specifico, l'applicazione sviluppata viene eseguita su un computer che funge da server, su cui è installato il sistema operativo Windows 11. All'interno del sistema operativo viene eseguita l'applicazione Docker Desktop, che si occupa della gestione delle istanze di tipo Docker. Docker Desktop crea un container in cui vengono istanziati due Docker: uno per l'applicazione e uno per il Database.

Per consentire l'esecuzione dell'applicazione nel Docker, viene creato un ambiente di esecuzione basato su JVM17. I due Docker vengono connessi tramite una rete privata creata appositamente da Docker Desktop. In particolare, il Database viene connesso all'applicazione tramite il protocollo TCP alla porta 27017. Inoltre, viene esposto all'esterno del Docker la possibilità di connettersi all'applicazione tramite un qualsiasi browser, da qualsiasi sistema operativo, utilizzando il protocollo HTTP alla porta 8080.

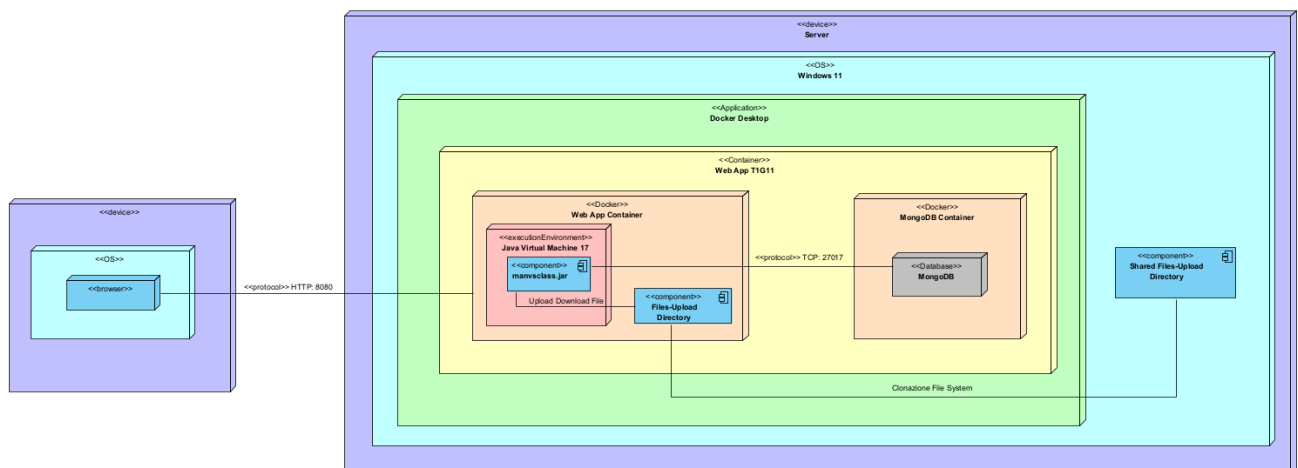
Nel contesto del Web App Container, viene creata una cartella dedicata per il salvataggio dei file. Questa cartella viene clonata nell'ambiente Windows 11 al fine di creare un volume condiviso che può essere accessibile anche da altri task, garantendo l'accesso ai file delle classi anche quando il container è spento.

Questa soluzione offre diversi vantaggi. Prima di tutto, consente la persistenza dei file delle classi anche dopo lo spegnimento del container. In questo modo, i file rimangono accessibili e non vengono persi o eliminati durante la chiusura o la riavvio del container.

Inoltre, il volume condiviso facilita la condivisione dei file tra diversi task o componenti del sistema. Ad esempio, se l'applicazione web ha bisogno di accedere ai file delle classi anche durante l'esecuzione di altri task o servizi, può farlo attraverso il volume condiviso senza dover dipendere esclusivamente dal funzionamento del container.

In sintesi, l'utilizzo di una cartella di salvataggio dei file clonata nell'ambiente Windows 11 e la creazione di un volume condiviso offrono la possibilità di mantenere i file delle classi accessibili anche quando il container è spento, garantendo la persistenza dei dati e facilitando la condivisione dei file tra diverse parti del sistema.

Questo setup di deployment garantisce che l'applicazione sia eseguita in modo isolato all'interno del Docker e che i componenti necessari siano connessi correttamente per consentire la comunicazione e l'accesso all'applicazione tramite il protocollo HTTP.



5.6 Integrabilità

5.6.1 API Locali

Si riportano le API locali che sono utilizzate da altri moduli nello stesso ambiente durante lo sviluppo. Le API offrono i seguenti servizi:

- Login admin
- Registrazione admin
- Modifica Classe
- Download Classe
- Ricerca Classe
- Eliminazione Classe
- Upload Classe
- Elenca Classi
- Filtra Classi
- Ordinamento Classi
- Nuovo Report
- Nuovo Like
- Upload Interaction
- Elenca Report
- Elenca Interaction

5.6.2 Strutture Dati

5.6.2.1 ClassUT

E' la struttura dati utilizzata per memorizzare le informazioni sulle classi ed è strutturata nel seguente modo:

```
//Nome Classe
private String name;
//Data di inserimento
private String date;
//Difficoltà / Complessità
private String difficulty;
//Path nel file system
private String code Uri;
//Breve descrizione della classe
private String description;
//Categorie di appartenenza della classe
private List<String> category;
```

5.6.2.2 Admin

E' la struttura dati utilizzata per memorizzare le informazioni personali e di accesso per gli amministratori ed è strutturata nel seguente modo:

```

//Informazioni personali
private String nome;
private String cognome;

//Informazioni profilo amministratore
private String username; //Username fase di registrazione
private String password; //Password di accesso al profilo

```

5.6.2.3 Interaction

E' la struttura dati utilizzata per memorizzare le informazioni sulle interazioni dell'utente con le classi, come i "Mi piace" o i report, ed è strutturata nel seguente modo:

```

//Id interazione
private int id_i;
//Email utente
private String email;
//Nome della classe
private String name;
//Id utente
private long id;
//Tipo di interazione:
//0 -> Report
//1 -> Like
private int type;
//Commento nel caso del report, null altrimenti
private String commento;
//Data di inserimento
private String date;

```

5.6.2.4 Operation

E' la struttura dati utilizzata per memorizzare le informazioni sulle operazioni dell'amministratore con le classi, come le eliminazioni o le modifiche, ed è strutturata nel seguente modo:

```

//Id Operazione
private int id_op;
//Username dell'admin che ha effettuato l'operazione
private String username_admin;
//Nome della classe
private String nome_classe;
//Tipo dell'operazione
private int type;
//Data dell'operazione
private String date;

```

5.6.3 Services

Di seguito sono riportati i metodi esposti dai Servizi in base al loro package di appartenenza

5.6.3.1 Login admin

```
public String loginAdmin(Admin admin1)
```

Metodo che permette di autenticare l'admin al fine di fornirgli l'accesso al sistema. Come parametro di ingresso deve essere passato un oggetto di tipo Admin con i campi username e password popolati. L'oggetto può essere popolato da un apposito form.

Input:

- Admin1 oggetto di tipo Admin che deve contenere i campi username e password popolati, al fine di autenticare l'amministratore.

Return: ritorna il messaggio "Ok" se l'autenticazione va a buon fine, "utente non loggato" altrimenti.

5.6.3.3 Registrazione admin

```
public Admin registraAdmin(Admin admin1)
```

Metodo che permette di salvare l'admin sulla base dati. Come parametro di ingresso deve essere passato un oggetto di tipo Admin con tutti i campi popolati. L'oggetto può essere popolato da un apposito form.

Input:

- Admin1 oggetto di tipo Admin che deve contenere i campi pieni, al fine di salvare l'amministratore sulla base dati.

Return: ritorna l'entità salvata nel database dopo l'operazione di salvataggio.

5.6.3.4 Modifica Classe

```
public ResponseEntity<String> modificaClasse( String name, ClassUT newContent)
```

Metodo che permette di modificare una classe già esistente nel sistema. Come parametri di ingresso devono essere passati il nome della classe da modificare e un oggetto di tipo ClassUT con tutti i nuovi campi popolati. L'oggetto può essere popolato da un apposito form.

Input:

- Name oggetto di tipo String che contiene il nome della classe
- newContent oggetto di tipo ClassUT che deve contenere tutti i campi popolati al fine di sostituire i valori vecchi della classe già esistente

Return: ritorna un oggetto di tipo ResponseEntity con il messaggio "Aggiornamento eseguito correttamente" e lo stato HTTP "OK" se non si verificano errori nella modifica.

Oppure il messaggio "Nessuna classe trovata o nessuna modifica effettuata" e lo stato NOT FOUND nel caso in cui ci dovessero essere errori.

5.6.3.5 Download Classe

```
public ResponseEntity<?> downloadClasse(("name") String name) throws Exception
```

Metodo che permette il download di un file contenente il codice sorgente di una classe dal sistema. Come parametro di ingresso deve essere passato il nome della classe da scaricare. La funzione cerca l'oggetto di tipo ClassUT corrispondente nel database utilizzando il parametro name e scarica il file contenente il codice della classe tramite il metodo FileDownloadUtil.downloadClassFile(). Il file scaricato viene poi restituito all'utente nel corpo della risposta HTTP.

Input:

- Name oggetto di tipo String che contiene il nome della classe

Return:

Restituisce un oggetto ResponseEntity contenente il file scaricato, utilizzando l'URI dell'oggetto ClassUT trovato.

5.6.3.6 Ricerca Classe

```
public List<ClassUT> ricercaClasse( String text)
```

Metodo che permette di cercare una classe all'interno del sistema. Come parametro di ingresso deve essere passato il nome della classe da ricercare. Il metodo usufruisce della funzione findByText() che si occupa di creare una lista di oggetti di tipo ClassUT corrispondenti alla ricerca del testo all'interno del campo nome della classe.

Input:

- Text oggetto di tipo String che contiene il nome della classe

Return: Restituisce una lista di oggetti di tipo ClassUT che corrispondono alla ricerca effettuata.

5.6.3.7 Eliminazione Classe

```
public ClassUT eliminaClasse( String name)
```

Metodo che permette di eliminare una classe all'interno del sistema. Come parametro di ingresso deve essere passato il nome della classe da eliminare.

Input:

- Name oggetto di tipo String che contiene il nome della classe

Return:

Restituisce l'oggetto ClassUT che è stato rimosso dalla collezione. In caso di mancata corrispondenza della query con alcun documento della collezione, il valore di ritorno sarà null.

5.6.3.8 Upload Classe

```
public ClassUT UploadClasse( ClassUT classe)
```

Metodo che permette di caricare una classe all'interno del sistema. Come parametro di ingresso deve essere passato un oggetto di tipo ClassUT con tutti i campi popolati. L'oggetto può essere popolato da un apposito form.

Input:

- Classe oggetto di tipo ClassUT che deve contenere tutti i campi popolati al fine di caricare la nuova classe nel sistema

Return: ritorna l'entità salvata nel database dopo l'operazione di salvataggio.

5.6.3.9 Elenca Classi

```
public List<ClassUT> elencaClassi()
```

Metodo che permette di visualizzare tutte le classi all'interno del sistema.

Return: Restituisce una lista di oggetti di tipo ClassUT che sono presenti all'interno del sistema

5.6.3.10 Filtra Classe

```
public List<ClassUT> filtraClassiD(String difficulty)
```

Metodo che permette di filtrare le classi all'interno del sistema. Come parametro di ingresso deve essere passato un oggetto di tipo String contenente la difficoltà secondo cui filtrare.

Input:

- Difficulty oggetto di tipo String che contiene la categoria secondo cui filtrare

Return: Restituisce una lista di oggetti di tipo ClassUT che corrispondono alla ricerca effettuata.

5.6.3.11 Ordinamento Classi

```
public List<ClassUT> ordinaClassi()
```

Metodo che permette di ordinare le classi all'interno del sistema per data. Si distingue dall'ordinamento per nomi che possiede un nome diverso:

```
public List<ClassUT> ordinaClassiNomi()
```

Le funzioni utilizzano i metodi orderByDate e orderByNome rispettivamente.

Return: Restituisce una lista di oggetti di tipo ClassUT che corrispondono alla ricerca effettuata.

5.6.3.12 Nuovo Report

```
public String newReport( String name, String commento )
```

Metodo che permette di caricare un nuovo report all'interno del sistema. Come parametro di ingresso deve essere passato un oggetto di tipo String contenente il nome della classe di cui fare il report e un oggetto di tipo String contenente una breve descrizione del report effettuato. Gli oggetti possono essere popolati con un apposito form.

Input:

- Name oggetto di tipo String che contiene il nome della classe da segnalare.
- Commento oggetto di tipo String che contiene una breve descrizione della segnalazione effettuata.

Return: ritorna il messaggio "Nuova interazione di tipo report inserita"

5.6.3.13 Nuovo Like

```
public String newLike( String name)
```

Metodo che permette di caricare un nuovo like all'interno del sistema. Come parametro di ingresso deve essere passato un oggetto di tipo String contenente il nome della classe a cui applicare il like

Input:

- Name oggetto di tipo String che contiene il nome della classe a cui applicare il like

Return: ritorna il messaggio "Nuova interazione di tipo like inserita"

5.6.3.14 Upload Interaction

```
public interaction UploadInteraction( interaction interazione)
```

Metodo che permette di caricare una nuova interazione all'interno del sistema. Come parametro di ingresso deve essere passato un oggetto di tipo interaction contenente tutti i campi popolati.

Input:

- Interazione oggetto di tipo Interaction che contiene tutti i campi popolati al fine di inserire una nuova interazione nel sistema

Return: ritorna l'entità salvata nel database dopo l'operazione di salvataggio.

5.6.3.15 Elenca Report

```
public List<interaction> elencaReport()
```

Metodo che permette di visualizzare tutti i report all'interno del sistema.

Return: Restituisce una lista di oggetti di tipo Interaction con il campo type=0 che sono presenti all'interno del sistema

5.6.3.16 Elenca Interaction

```
public List<interaction> elencaInt()
```

Metodo che permette di visualizzare tutti i report all'interno del sistema.

Return: Restituisce una lista di oggetti di tipo Interaction che sono presenti all'interno del sistema

5.6.4 Utils

Di seguito vengono riportate le API dei servizi di Utilità.

5.6.4.1 getLikes

```
public long getLikes(String name)
```

Restituisce il numero di "Likes" (valore intero) per un'interazione nella collezione "interaction" di MongoDB.

Input:

name (String): il nome dell'interazione di cui si vuole conoscere il numero di "Likes".

Return:

count (long): il numero di "Likes" dell'interazione.

5.6.4.2 findReport

```
public List<interaction> findReport()
```

Restituisce una lista di oggetti di tipo "interaction" e type=0 dalla collezione "interaction" di MongoDB, ovvero i report presenti.

Input:

Nessun parametro di input.

Return:

posts (List<interaction>): una lista di oggetti di tipo "interaction".

5.6.4.3 findByText

```
public List<ClassUT> findByText(String text)
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, filtrati in base al nome della classe.

Input:

text (String): il testo da cercare all'interno del nome della classe.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.4 findAdminByUsername

```
public Admin findAdminByUsername(String username)
```

Restituisce un oggetto di tipo "Admin" dalla collezione "Admin" di MongoDB, filtrato in base allo username.

Input:

username (String): lo username dell'amministratore da cercare.

Return:

admin (Admin): un oggetto di tipo "Admin".

5.6.4.5 searchAndFilter

```
public List<ClassUT> searchAndFilter(String text, String category)
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, filtrati in base al testo da cercare all'interno del nome e alla categoria della classe.

Input:

text (String): il testo da cercare all'interno del nome e della descrizione della classe.

category (String): la categoria della classe da cercare.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.6 filterByCategory

```
public List<ClassUT> filterByCategory(String category)
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, filtrati in base alla categoria della classe.

Input:

category (String): la categoria della classe da cercare.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.7 orderByDate

```
public List<ClassUT> orderByDate()
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, ordinati in base alla data di creazione della classe.

Input:

Nessun parametro di input.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.8 orderByName

```
public List<ClassUT> orderByName()
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, ordinati in base al nome della classe.

Input:

Nessun parametro di input.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.9 filterByDifficulty

```
public List<ClassUT> filterByDifficulty(String difficulty)
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, filtrati in base alla difficoltà della classe.

Input:

difficulty (String): la difficoltà della classe da cercare.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.10 searchAndFilter

```
public List<ClassUT> searchAndFilter(String text, String difficulty)
```

Restituisce una lista di oggetti di tipo "ClassUT" dalla collezione "ClassUT" di MongoDB, filtrati in base al testo da cercare all'interno del nome e alla difficoltà della classe.

Input:

text (String): il testo da cercare all'interno del nome e della descrizione della classe.

difficulty (String): la difficoltà della classe da cercare.

Return:

posts (List<ClassUT>): una lista di oggetti di tipo "ClassUT".

5.6.4.11 downloadClassFile

```
public static ResponseEntity<Resource> downloadClassFile(String downloadpath) throws Exception
```

Restituisce un'entità di risposta HTTP contenente il file Java situato nel percorso specificato.

Input:

downloadpath (String): il percorso del file Java da scaricare.

Return:

ResponseEntity<Resource>: un'entità di risposta HTTP contenente il file Java.

Exception:

Exception se si verifica un errore durante la creazione dell'entità di risposta HTTP o durante l'accesso al file Java.

5.6.4.12 saveClassFile

```
public static void saveClassFile(String fileName,String cname,MultipartFile multipartFile) throws IOException
```

Salva un file all'interno di una specifica directory "Files-Upload/nomeclasse". Se la directory non esiste, viene creata. Il file viene salvato con il nome specificato.

Input:

fileName (String): il nome del file da salvare.

cname (String): il nome della classe a cui appartiene il file.

multipartFile (MultipartFile): il file da salvare.

Return:

Nessun valore di ritorno.

Exception:

IOException: se si verifica un errore durante il salvataggio del file.

5.6.4.13 deleteDirectory

```
public static void deleteDirectory(File directory) throws IOException
```

Elimina una directory e tutti i file al suo interno.

Input:

directory (File): la directory da eliminare.

Return:

Nessun valore di ritorno.

Exception:

IOException: se si verifica un errore durante l'eliminazione della directory o dei file al suo interno.

5.6.5 Rest API

La REST API (Representational State Transfer) è un'architettura software che definisce un insieme di principi per la creazione di servizi web. Questi servizi consentono ai sistemi di comunicare e scambiare dati in modo efficiente grazie all'utilizzo di interfacce comuni.

Le API REST sono stateless, il che significa che ogni richiesta del client deve contenere tutte le informazioni necessarie per elaborare quella richiesta. Utilizzano i metodi HTTP, come GET, POST, PUT e DELETE, per definire le operazioni che

possono essere eseguite sui dati. Ogni risorsa in un'API REST è identificata da un URI univoco. I dati vengono rappresentati in formati come JSON o XML.

Nel progetto, sono state implementate le REST API relative a ogni caso d'uso. Queste API consentono di accedere ai dati e di eseguire operazioni come la creazione, l'aggiornamento e la cancellazione di informazioni specifiche.

5.6.5.1 **Elenca classi** “/home” **(GET)**

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce l'elenco di tutte le classi disponibili.

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.2 **Ricerca Classe per nome** “/home/{text}” **(GET)**

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce una classe con un nome.

{text} rappresenta il parametro di ricerca per nome

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito una classe o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.3 Ordina classi per nome “/orderbyname” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce un elenco di classi ordinate per nome.

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.4 Ordina classi per data “/orderbydate” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce un elenco di classi ordinate per data.

Restituisce Oggetti Json del seguente tipo:


```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.5 Filtra classi per categoria “/Cfilterby/{category}” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce un elenco di classi filtrate per categoria.

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.6 Filtra classi per categoria “/Cfilterby/{category}” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce un elenco di classi filtrate per categoria.

{category} rappresenta il parametro di filtraggio della categoria

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.7 Filtra classi per categoria “/Dfilterby/{difficulty}” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce un elenco di classi filtrate per categoria.

{difficulty} rappresenta il parametro di filtraggio per difficoltà

Restituisce Oggetti Json del seguente tipo:

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

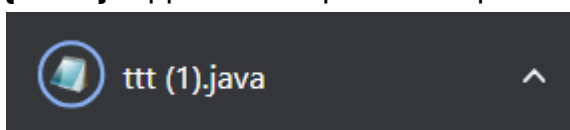
Tipi di Risposta :

- **200 OK:** viene restituito un elenco di classi o un elenco vuoto [].
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.8 Filtra classi per categoria “/downloadFile/{name}” (GET)

Questa API permette di inoltrare una richiesta HTTP di tipo GET che restituisce il file java corrispondente alla classe richiesta.

{name} rappresenta il parametro per scaricare il file :



Tipi di Risposta :

- **200 OK:** viene restituito il contenuto del file .java .
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.9 Upload della descrittore classe+ file.java “/uploadFile (POST)

Questa API permette di inoltrare una richiesta HTTP di tipo POST che inserisce il descrittore della classe e il file .java corrispondente.

Parametri richiesti:

File in formato .java

model contenente la stringa JSON corrispondente alla classe da inserire

	Key	Value	Content type
<input checked="" type="checkbox"/>	file	FTPFile.java ×	Auto
<input checked="" type="checkbox"/>	model	{ "name": "rafft2", "date": "2022-05-13", "difficulty": "Intermediate", "code Uri": "/code/123", "description": "This is a test class", "category": ["Java", "Programming", "Testing"] }	Auto
	Key		Auto

Tipi di Risposta :

- **200 OK:** viene restituito un messaggio con il path di salvataggio del file e la dimensione del file appena caricato.

Body	Cookies	Headers (5)	Test Results
<div> <div>Pretty</div> <div>Raw</div> <div>Preview</div> <div>Visualize</div> <div>JSON</div> <div>↺</div> </div> <pre> 1 { 2 "fileName": "FTPFile.java", 3 "downloadUri": "/downloadFile", 4 "size": 15051 5 }</pre>			

- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.10 Modifica classe

"/update/{name}"

(POST)

Questa API permette di inoltrare una richiesta HTTP di tipo POST che aggiorna i campi della classe specificata dal parametro di ricerca name.

Parametri richiesti:

{name} parametro di ricerca della classe

body contenente la stringa JSON del report con i parametri modificati

```
{
  "name": "sdds",
  "date": "2023-07-10",
  "difficulty": "Beginner",
  "code_uri": "Files-Upload/sdds/ciao.java",
  "description": "sddsrr",
  "category": [
    "sdsdyyyy",
    "sdsd",
    "sdd"
  ]
},
```

Tipi di Risposta :

- **200 OK:** viene restituito un messaggio che conferma il successo dell'operazione di modifica.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.11 Nuovo like ad una classe

"/newLike/{name}"

(POST)

Questa API permette di inoltrare una richiesta HTTP di tipo POST che aggiorna i campi della classe specificata dal parametro di ricerca name.

Parametri richiesti:

{name} parametro di ricerca della classe

Tipi di Risposta :

- **200 OK:** viene restituito un messaggio che conferma il successo dell'operazione di inserimento del nuovo like.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.12 Nuovo report ad una classe

"/newReport/{name}"

(POST)

Questa API permette di inoltrare una richiesta HTTP di tipo POST che aggiorna i campi della classe specificata dal parametro di ricerca name.

Parametri richiesti:

{name} parametro di ricerca della classe

stringa JSON con i parametri del report da inserire

```
{
  "id_i": 0,
  "email": "prova.449660@email.com",
  "name": "ttt",
  "id": 449660,
  "type": 0,
  "commento": "sdsd",
  "date": "2023-07-11"
}
```

Tipi di Risposta :

- **200 OK:** viene restituito un messaggio che conferma il successo dell'operazione di inserimento del nuovo report.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

5.6.5.12 Registrazione Admin "/registraAdmin/ (POST)

Questa API permette di inoltrare una richiesta HTTP di tipo POST che inserisce un nuovo Admin nel sistema

Parametri richiesti:

stringa JSON con i parametri dell'admin da inserire

```
{
  "nome": "Mario",
  "cognome": "Rossi",
  "username": "mario.rossi",
  "password": "password123"
}
```

Tipi di Risposta :

- **200 OK:** viene restituito un messaggio che conferma il successo dell'operazione di inserimento del nuovo admin.
- **400 Bad Request:** Il payload di input è incompleto o non valido.
- **500 Internal Server Error:** Si è verificato un errore interno al server.

Capitolo 6: Implementazione

6.1 Introduzione

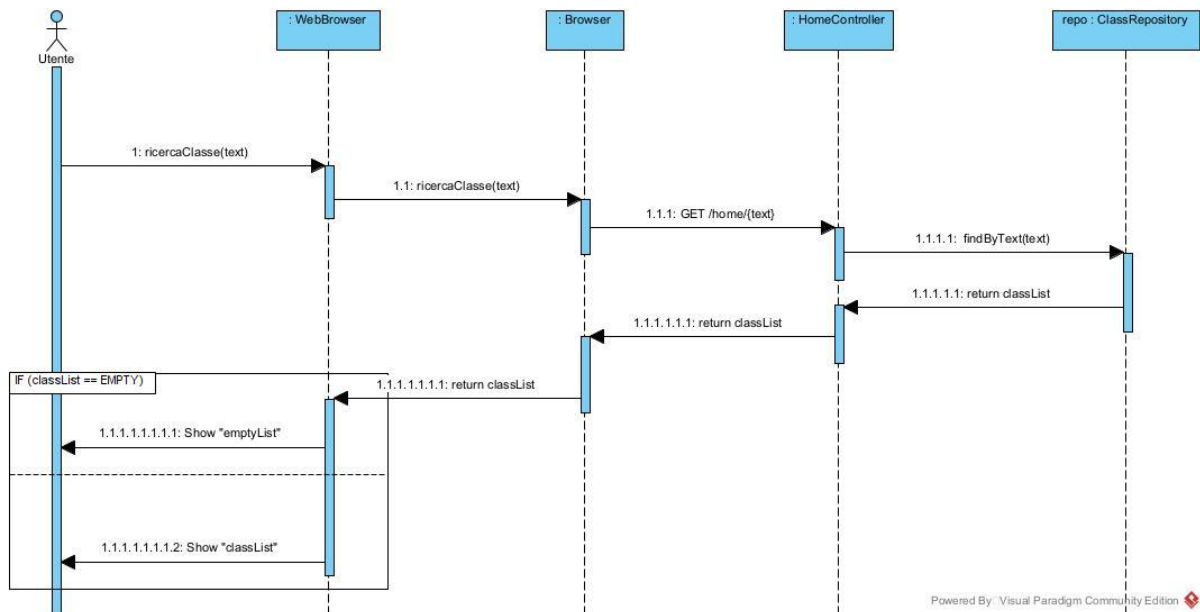
In questo capitolo, viene presentata l'implementazione del nostro progetto introducendo i relativi diagrammi di sequenza di dettaglio.

6.2 Sequence Diagram di Dettaglio

I Sequence Diagram di dettaglio sono un tipo di diagramma dinamico che forniscono una visione dettagliata dell'interazione tra gli oggetti all'interno di un sistema software. Questi diagrammi mostrano le chiamate di metodo tra gli oggetti nel corso del tempo e illustrano l'ordine e il flusso delle operazioni. Nei prossimi paragrafi verranno presentati i Sequence Diagram di dettaglio per illustrare l'interazione tra gli oggetti nel nostro sistema.

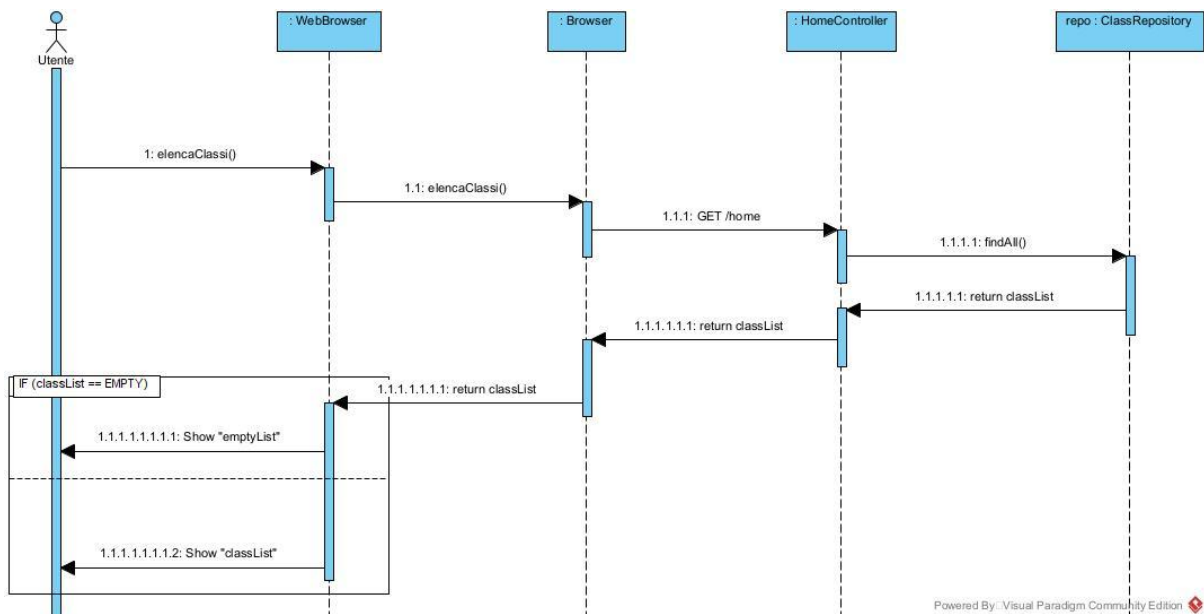
6.2.1 Sequence Diagram: RicercaClassi

L'utente fa una richiesta GET al percorso /home/{text} attraverso il web browser. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo ricercaClasse() del servizio. Il servizio chiama il metodo findByText() del ClassRepository per trovare le classi corrispondenti al testo passato nella richiesta. Se il ClassRepository trova delle classi, le restituisce al servizio, che a sua volta restituisce le classi al HomeController, che restituisce le classi al web browser. Se invece il ClassRepository non trova delle classi, restituisce una lista vuota al servizio, che a sua volta restituisce la lista vuota al HomeController, che restituisce la lista vuota al web browser.



6.2.2 Sequence Diagram: elencaClassi

L'utente fa una richiesta GET al percorso /home attraverso il web browser. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo elencaClassi() del servizio. Il servizio chiama il metodo findAll() del ClassRepository per trovare tutte le classi. Infine, il ClassRepository restituisce la lista di tutte le classi al servizio, che a sua volta restituisce la lista di tutte le classi al HomeController, che restituisce la lista di tutte le classi al web browser.

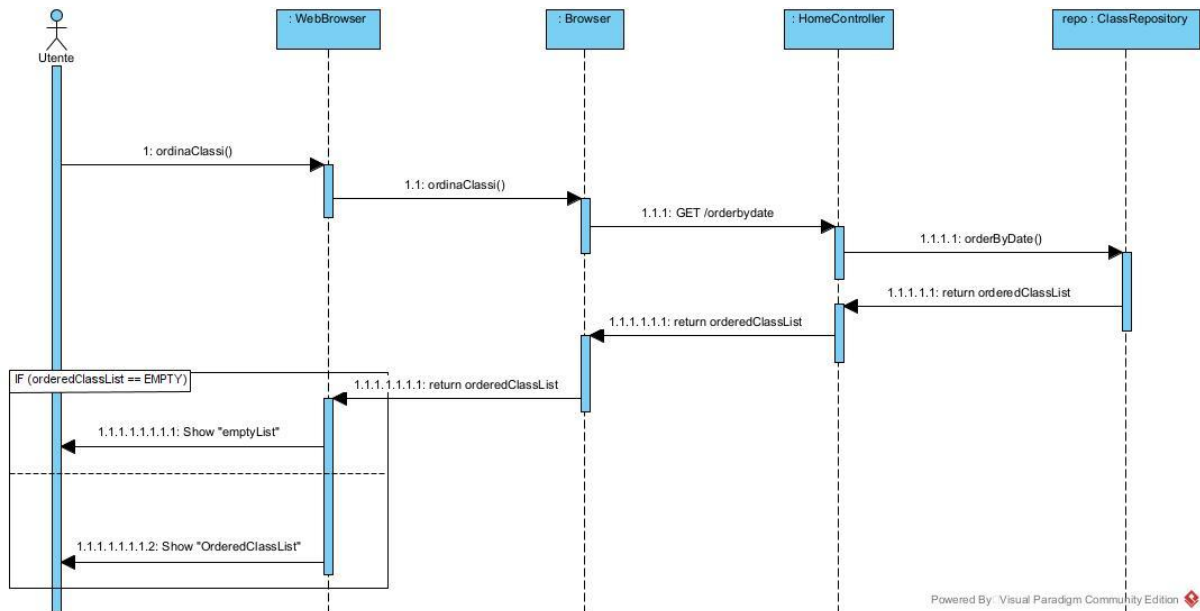


6.2.3 Sequence Diagram: ordinaClassi

Order by date

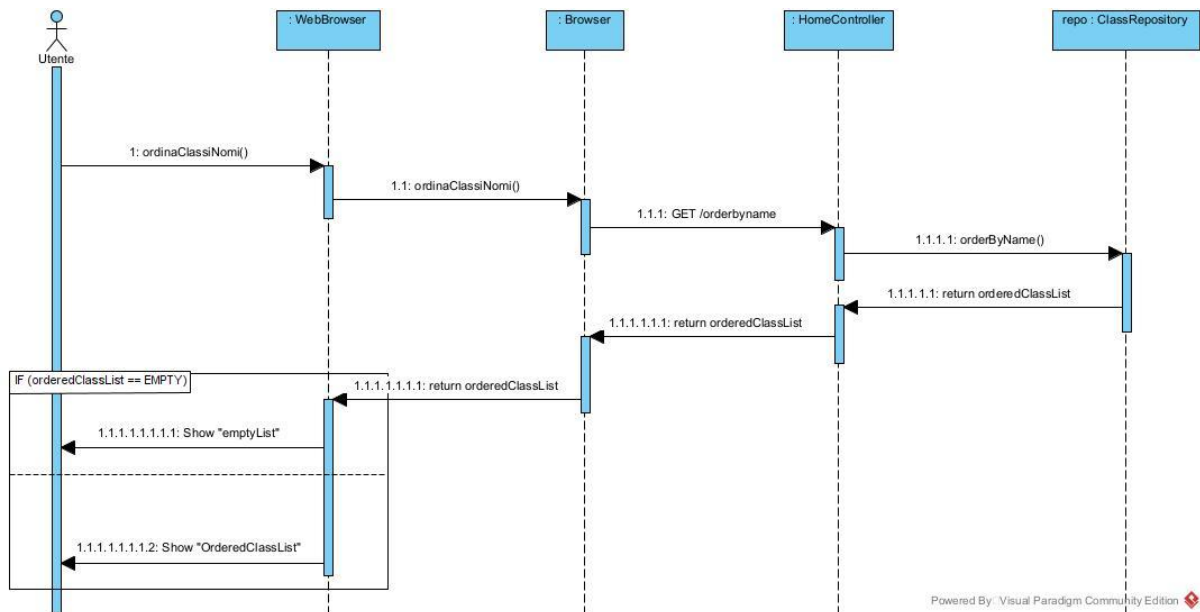
L'utente fa una richiesta GET al percorso /orderbydate attraverso il web browser. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo

ordinaClassi() del servizio. Il servizio chiama il metodo orderByDate() del repository srepo per ordinare la lista di oggetti ClassUT per data. Infine, il repository restituisce la lista di oggetti ClassUT ordinati per data al servizio, che a sua volta restituisce la lista di oggetti ClassUT ordinati per data al HomeController, che restituisce la lista di oggetti ClassUT ordinati per data al web browser.



Order by name

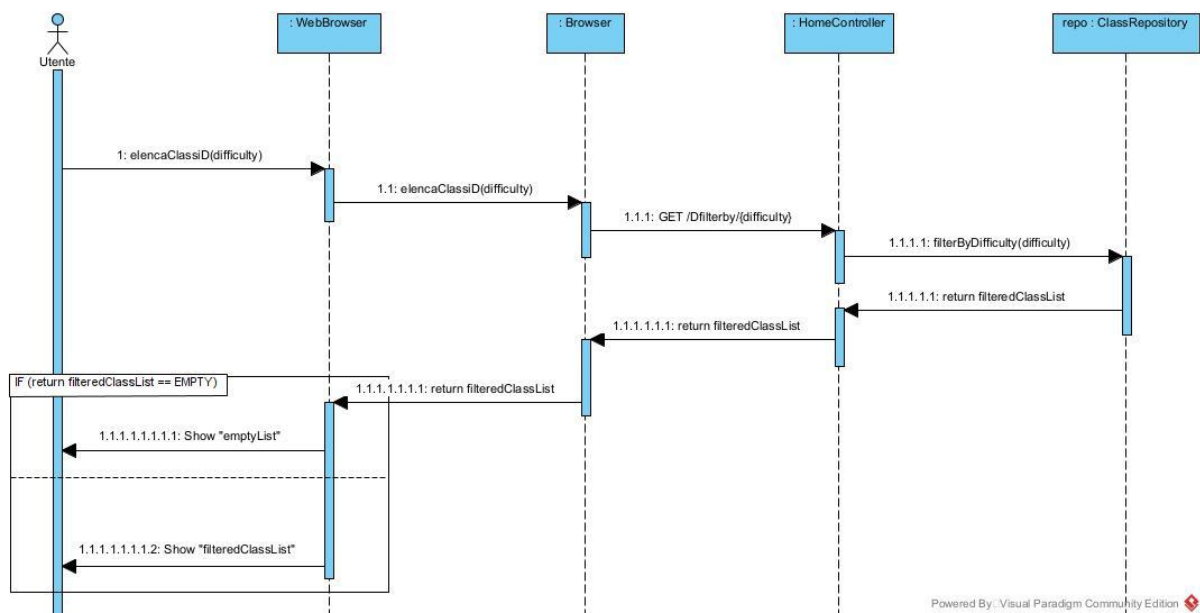
L'utente fa una richiesta GET al percorso /orderbyname attraverso il web browser. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo ordinaClassiNomi() del servizio. Il servizio chiama il metodo orderByName() del repository srepo per ordinare la lista di oggetti ClassUT per nome. Infine, il repository restituisce la lista di oggetti ClassUT ordinati per nome al servizio, che a sua volta restituisce la lista di oggetti ClassUT ordinati per nome al HomeController, che restituisce la lista di oggetti ClassUT ordinati per nome al web browser.



6.2.4 Sequence Diagram: filtraClassi

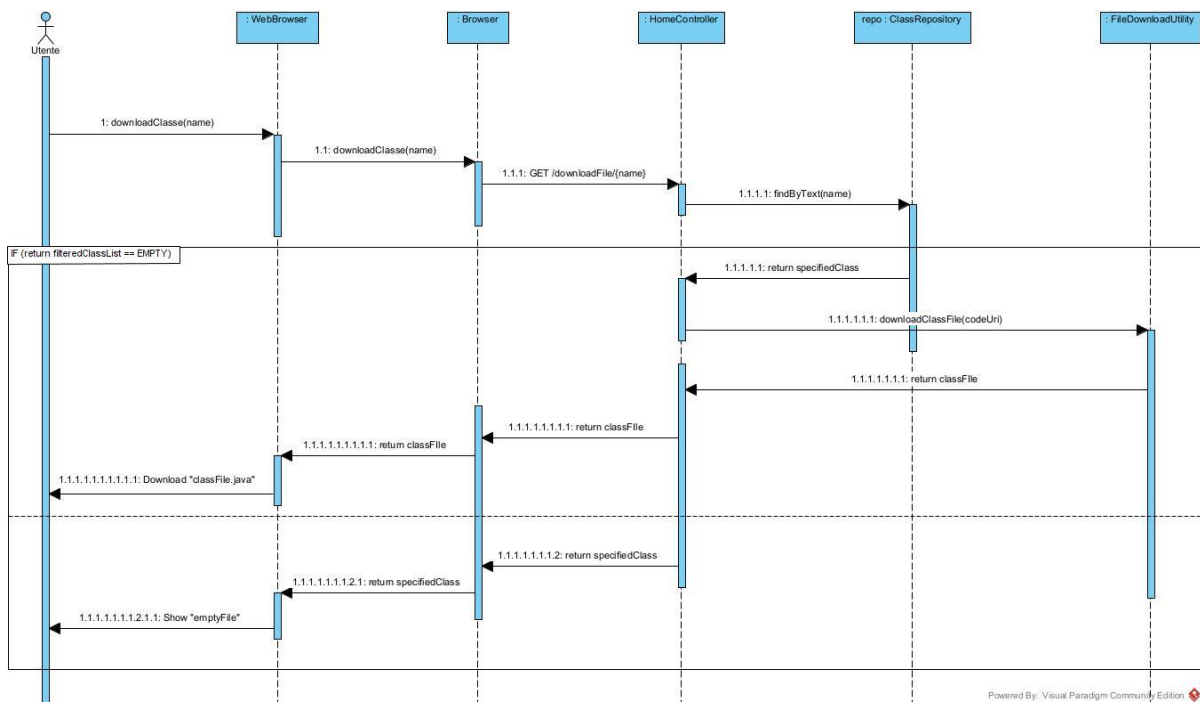
Filter by difficulty

L'utente fa una richiesta GET al percorso /Dfilterby/{difficulty} attraverso il web browser, passando la difficoltà da filtrare. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo `elencaClassiD()` del servizio, passando la difficoltà. Il servizio chiama il metodo `filterByDifficulty()` del repository `srepo` per filtrare la lista di oggetti `ClassUT` per difficoltà. Infine, il repository restituisce la lista di oggetti `ClassUT` filtrati per difficoltà al servizio, che a sua volta restituisce la lista di oggetti `ClassUT` filtrati per difficoltà al HomeController, che restituisce la lista di oggetti `ClassUT` filtrati per difficoltà al web browser.



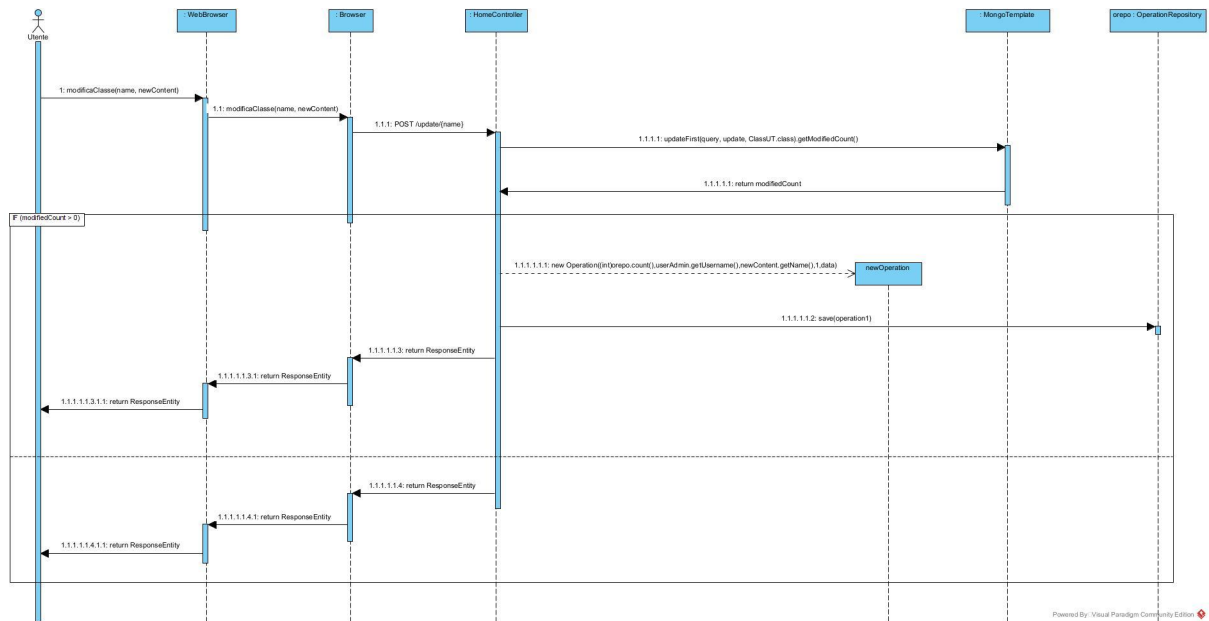
6.2.5 Sequence Diagram: DownloadClasse

L'utente fa una richiesta GET al percorso /downloadFile/{name} attraverso il web browser, passando il nome del file della classe da scaricare. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo downloadClasse() del servizio, passando il nome del file della classe da scaricare. Il servizio chiama il metodo findByText() del repository srepo per trovare la classe con il nome specificato. Successivamente, il servizio chiama il metodo downloadClassFile() della classe FileDownloadUtil, passando il codice URI della classe. Infine, il metodo downloadClassFile() restituisce una risposta al servizio, che a sua volta restituisce la risposta al HomeController, che restituisce la risposta al web browser.



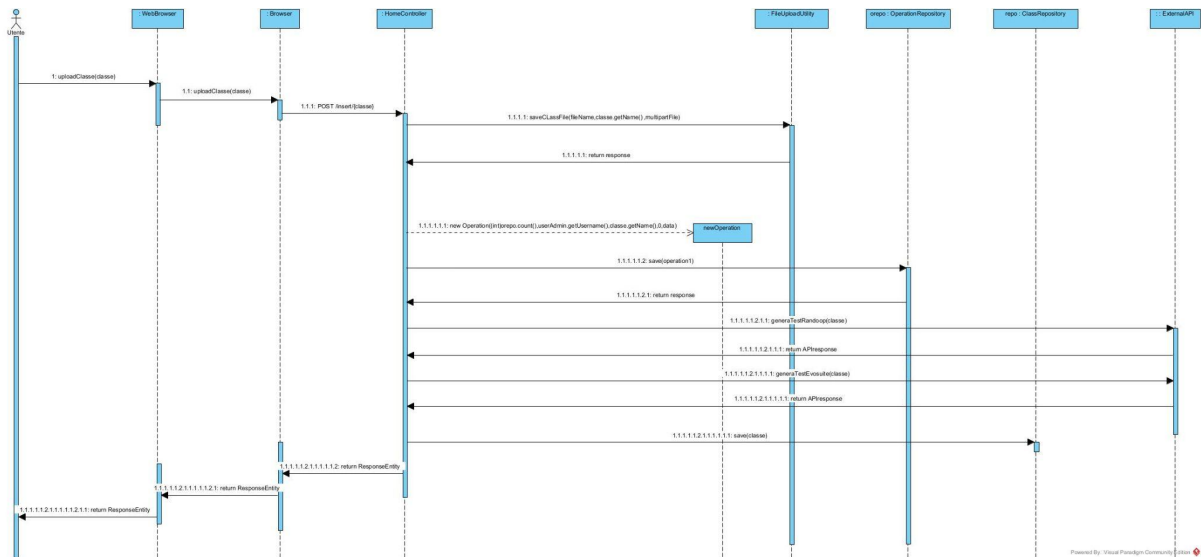
6.2.6 Sequence Diagram: ModificaClasse

L'utente fa una richiesta POST al percorso /update/{name} attraverso il web browser, passando il nome della classe da modificare e il nuovo contenuto della classe. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo modificaClasse() del servizio, passando il nome della classe da modificare e il nuovo contenuto della classe. Il servizio crea una query per trovare la classe da modificare nel database, e crea un oggetto Update con i nuovi valori. Successivamente, il servizio chiama il metodo updateFirst() del MongoTemplate con la query e l'oggetto Update per modificare la classe nel database. Se la modifica ha avuto successo, il servizio crea un oggetto Operation per registrare l'operazione di modifica nel repository OperationRepository. Infine, il servizio restituisce una risposta al HomeController, che a sua volta restituisce la risposta al web browser. La risposta contiene un messaggio che indica se l'operazione di modifica è stata eseguita correttamente o se non è stata trovata alcuna classe da modificare.



6.2.7 Sequence Diagram: UploadClasse

L'utente fa una richiesta POST al percorso /insert attraverso il web browser, passando un oggetto ClassUT. Il web browser invia la richiesta al HomeController, che a sua volta chiama il metodo UploadClasse() del servizio, passando l'oggetto ClassUT. Il servizio crea un oggetto Operation per registrare l'operazione di inserimento nel repository OperationRepository. Successivamente, il servizio salva la classe UT nel database attraverso il repository ClassUTRepository. Infine, il servizio restituisce una risposta al HomeController, che a sua volta restituisce la risposta al web browser. La risposta contiene la classe UT inserita nel database.



Capitolo 7: Testing

Il testing è una fase cruciale nel processo di sviluppo software, poiché ha l'obiettivo di individuare difetti e verificare se il sistema soddisfa i suoi requisiti. Durante tutto il ciclo di sviluppo, è stata data grande importanza alla continuità del testing al fine di garantire la qualità e l'affidabilità dell'applicazione.

Il processo di testing è stato progettato per identificare difetti, errori o comportamenti indesiderati nel sistema. È stato applicato un approccio sistematico, in cui sono state definite diverse strategie di test per coprire tutti gli aspetti critici dell'applicazione, compresi i requisiti funzionali e non funzionali.

7.1 Test Suite per le API REST

La test suite delle API REST è un elemento fondamentale per garantire la qualità e l'affidabilità delle nostre applicazioni. Attraverso un insieme di test automatizzati, la test suite verifica il corretto funzionamento delle API, consentendo di identificare eventuali bug, errori o comportamenti indesiderati. Questo strumento ci permette di effettuare test approfonditi su diverse funzionalità delle API, compresi i metodi HTTP, i parametri di input, le risposte attese e gli scenari di errore. Grazie alla test suite, possiamo eseguire i test in modo coerente e ripetibile, garantendo la robustezza delle nostre API e fornendo una solida base per il loro sviluppo e manutenzione. Di seguito vengono riportati i piani di test per le 5 API scelte:

7.1.1 Piano di test per la funzionalità: RicercaClassi

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese	Output Ottenuti	Post-condizioni Ottenute	Esito
1	Nessuna corrispondenza	Il database "manvsclass" contiene una collezione "ClassUT" senza documenti che soddisfano il criterio di ricerca	"test"	Lista vuota	Nessuna modifica al database	Lista vuota	Nessuna modifica al database	PASS
2	Corrispondenza esatta	Il database "manvsclass" contiene una collezione "ClassUT" con un documento che ha il campo "name" uguale a "test"	"test"	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	PASS
3	Corrispondenza parziale	Il database "manvsclass" contiene una collezione "ClassUT" con due documenti, uno con il campo "name" con i primi caratteri a "test" e l'altro con il campo "name" i primi caratteri a "test"	"test"	Lista con due elementi che corrispondono ai documenti	Nessuna modifica al database	Lista con due elementi che corrispondono ai documenti	Nessuna modifica al database	PASS

4	Test case con maiuscole/minuscole	Il database "manvsclass" contiene una collezione "ClassUT" con un documento che ha il campo "name" uguale a "Test"	"test"	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	PASS
5	Test case con caratteri speciali	Il database "manvsclass" contiene una collezione "ClassUT" con un documento che ha il campo "name" uguale a "test@"	"test@"	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	PASS

I test case sopra coprono scenari comuni come la mancanza di corrispondenze, corrispondenze esatte, corrispondenze parziali, sensibilità alle maiuscole/minuscole e la gestione di caratteri speciali nella stringa di ricerca.

7.1.2 Piano di test per la funzionalità: filtraClassi

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese	Output Ottenuti	Post-condizioni Ottenute	Esito
1	Nessuna corrispondenza	Il database "manvsclass" contiene una collezione "ClassUT" senza documenti che hanno il campo "difficulty" uguale al valore di input	"Beginner"	Lista vuota	Nessuna modifica al database	Lista vuota	Nessuna modifica al database	PASS
2	Corrispondenza esatta	Il database "manvsclass" contiene una collezione "ClassUT" con almeno un documento che ha il campo "difficulty" uguale a "Intermediate"	"Intermediate"	Lista con un elemento che corrisponde al documento	Nessuna modifica al database	Lista con almeno un elemento che corrisponde al documento	Nessuna modifica al database	PASS
3	Test case con maiuscole/minuscole	Il database "manvsclass" contiene una collezione "ClassUT" con almeno un documento che ha il campo "difficulty" uguale a "Advanced"	"AdvancEd"	Lista vuota data la non validità dell'input	Nessuna modifica al database	Lista vuota	Nessuna modifica al database	PASS

4	Test case con valori non validi	Il database "manvsclass" contiene una collezione "ClassUT" con un documento che ha il campo "difficulty" uguale a "Advanced"	"invalid"	Lista vuota	Nessuna modifica al database	Lista vuota	Nessuna modifica al database	PASS
---	---------------------------------	--	-----------	-------------	------------------------------	-------------	------------------------------	------

7.1.3 Piano di test per la funzionalità: DownloadClasse

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese	Output Ottenuti	Post-condizioni Ottenute	Esito
1	File esistente	la classe test è presente nel database e il file associato è presente	"test"	Risposta HTTP con il file allegato	Nessuna modifica ai file o allo stato del sistema	Risposta HTTP con il file allegato	Nessuna modifica ai file o allo stato del sistema	PASS
2	File non esistente	Il percorso relativo alla classe test2 non corrisponde a un file esistente	"test2"	"status": 500, "error": "Internal Server Error"	Nessuna modifica ai file o allo stato del sistema	"status": 500, "error": "Internal Server Error"	Nessuna modifica ai file o allo stato del sistema	PASS

3	File non leggibile	Il percorso relativo alla classe test2 non corrisponde a un file esistente	"test2"	"status": 500, "error": "Internal Server Error"	Nessuna modifica ai file o allo stato del sistema	"status": 500, "error": "Internal Server Error"	Nessuna modifica ai file o allo stato del sistema	PASS
4	Download di un file non Java	Il percorso downloadpath punta a un file che non è un file Java	"test5"	Risposta HTTP con il file allegato e i corretti header e tipo di contenuto	Nessuna modifica ai file o allo stato del sistema	Risposta HTTP con il file allegato non Java	Nessuna modifica ai file o allo stato del sistema	PASS

7.1.4 Piano di test per la funzionalità: ModificaClasse

Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese	Output Ottenuti	Post-condizioni Ottenute	Esito
1	Classe esistente	Nel database è presente una classe con il nome specificato	Nome valido e oggetto ClassUT con nuovi valori	Risposta HTTP con lo stato 200 OK e il messaggio "Aggiornamento eseguito correttamente"	Modifica dei campi della classe nel database	Risposta HTTP con lo stato 200 OK e il messaggio "Aggiornamento eseguito correttamente"	Modifica dei campi della classe nel database	PASS
2	Classe non trovata	Nel database non è presente una classe con il nome specificato	Nome valido e oggetto ClassUT con nuovi valori	Risposta HTTP con lo stato 404 NOT FOUND e il messaggio "Nessuna classe trovata o nessuna modifica effettuata"	Nessuna modifica dei campi della classe nel database	Risposta HTTP con lo stato 404 NOT FOUND e il messaggio "Nessuna classe trovata o nessuna modifica effettuata"	Nessuna modifica dei campi della classe nel database	PASS
3	Modifica parziale	Nel database è presente una classe con il nome specificato	Nome valido e oggetto ClassUT con alcuni campi nuovi e alcuni campi non modificati	Risposta HTTP con lo stato 200 OK e il messaggio "Aggiornamento eseguito correttamente"	Modifica dei campi specificati della classe nel database	Risposta HTTP con lo stato 200 OK e il messaggio "Aggiornamento eseguito correttamente"	Modifica dei campi specificati della classe nel database	PASS

4	Nome non valido	Nel database è presente una classe con il nome specificato	Nome non valido (vuoto o formato non valido) e oggetto ClassUT con nuovi valori	Risposta HTTP con lo stato 404 NOT FOUND e il messaggio di errore HTTP	Nessuna modifica dei campi della classe nel database	Risposta HTTP con lo stato 404 NOT FOUND e il messaggio di errore HTTP	Nessuna modifica dei campi della classe nel database	PASS
---	-----------------	--	--	--	--	--	--	------

7.1.5 Piano di test per la funzionalità: UploadClasse

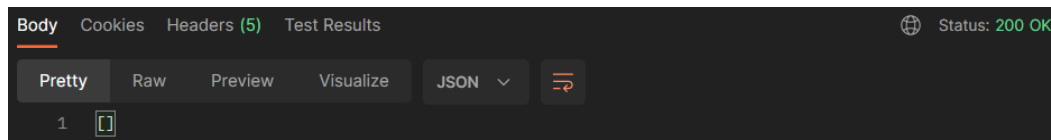
Test Case ID	Descrizione	Pre-condizioni	Input	Output Attesi	Post-condizioni Attese	Output Ottenuti	Post-condizioni Ottenute	Esito
1	Upload corretto	Nessuna classe esistente con lo stesso nome	Oggetto ClassUT valido file formato valido	Oggetto ClassUT salvato correttamente nel database	Inserimento della nuova classe nel database e del file nel filesystem condiviso	Oggetto ClassUT salvato correttamente nel database	Inserimento della nuova classe nel database e del file nel filesystem condiviso	PASS
2	Nome duplicato	Classe con lo stesso nome già presente nel database	Oggetto ClassUT con nome duplicato	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	PASS
3	Input non valido	File mancante nel form di inserimento	Oggetto ClassUT con campi mancanti o dati non validi	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	PASS
4	Campi mancanti	Alcuni campi obbligatori dell'oggetto ClassUT non sono presenti	Oggetto ClassUT con campi mancanti	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	"status": 500, "error": "Internal Server Error",	Nessuna modifica al database	PASS

7.2 Testing con Postman

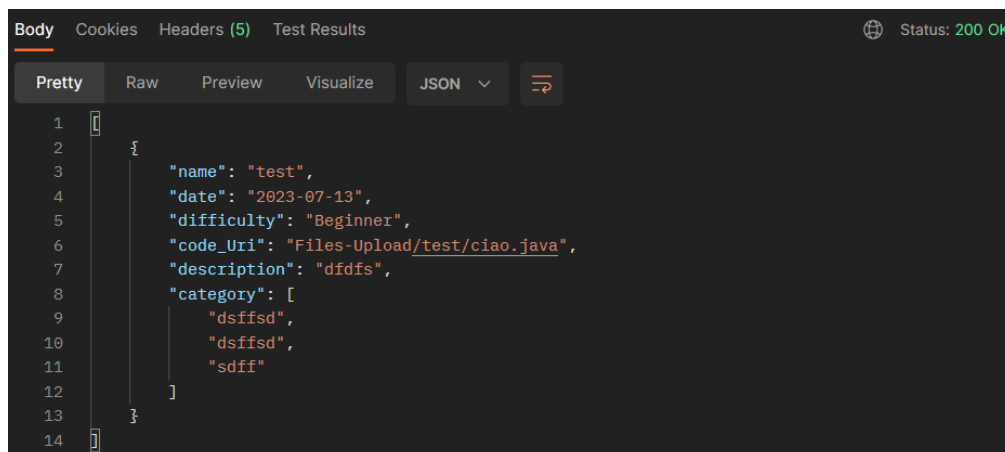
I test dell'API sono stati resi automatizzati mediante l'utilizzo di Post-man, ossia un'applicazione che consente di inviare richieste HTTP e visualizzare le risposte in modo interattivo. In particolare, consente di verificare se le risposte sono conformi alle aspettative, semplificando il processo di controllo delle qualità dell'API.

7.2.1 Testing RicercaClassi

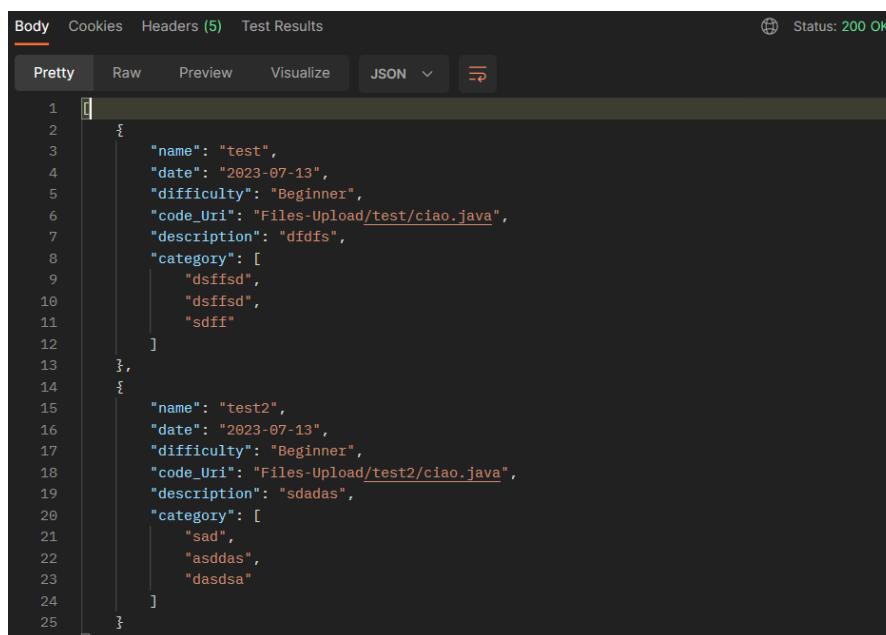
Test Case 1:



Test Case 2:



Test Case 3:



Test Case 4:

```
1 {
2   {
3     "name": "test",
4     "date": "2023-07-13",
5     "difficulty": "Beginner",
6     "code_uri": "Files-Upload/test/ciao.java",
7     "description": "sdasda",
8     "category": [
9       "sadds",
10      "sdad",
11      "sadsdadsa"
12    ]
13  }
14 }
```

Test Case 5:

```
1 {
2   {
3     "name": "@test2",
4     "date": "2023-07-13",
5     "difficulty": "Beginner",
6     "code_uri": "Files-Upload/@test2/ciao.java",
7     "description": "wsacasd",
8     "category": [
9       "ssad",
10      "sadd",
11      "asdsda"
12    ]
13  }
14 }
```

7.2.2 Testing filtraClassi

Test Case 1:

GET <http://localhost:8080/Dfilterby/Beginner> Send

Params Auth Headers (8) Body Pre-req Tests Settings Cookies

Query Params

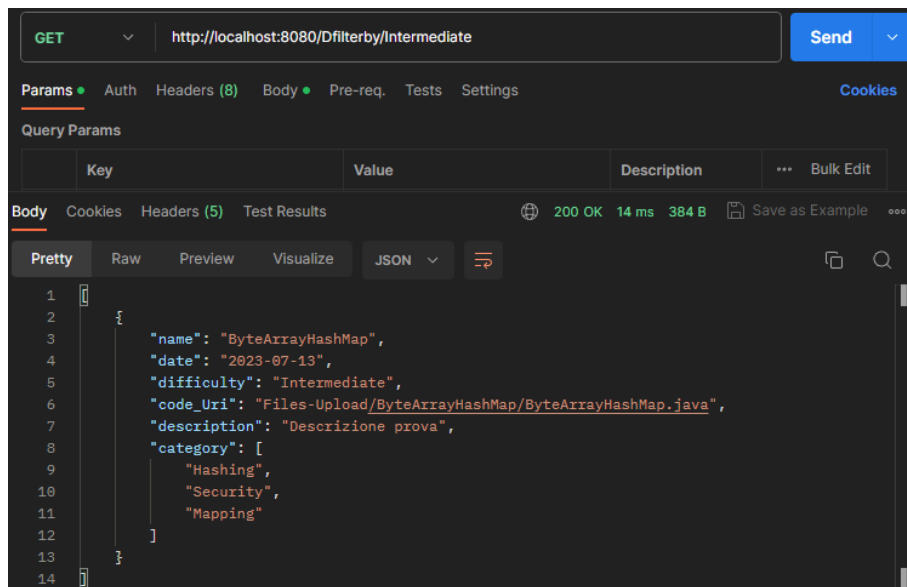
Key	Value	Description	...	Bulk Edit
-----	-------	-------------	-----	-----------

Body Cookies Headers (5) Test Results 200 OK 37 ms 166 B Save as Example

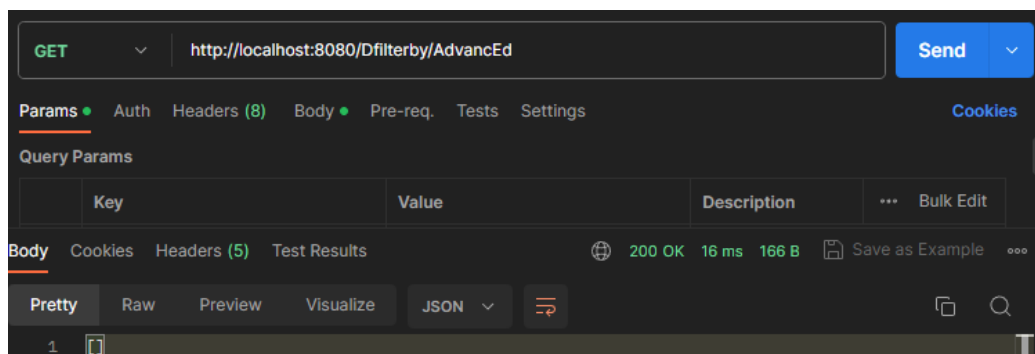
Pretty Raw Preview Visualize JSON

```
1 { }
```

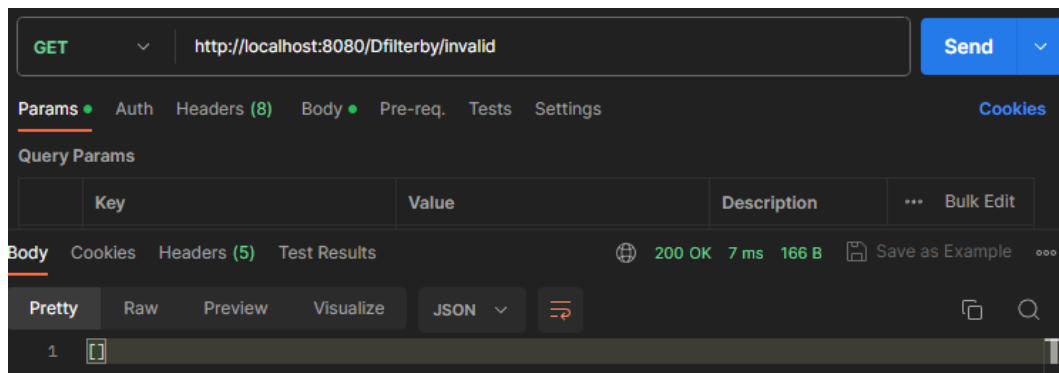
Test Case 2:



Test Case 3:



Test Case 4:



7.2.3 Testing DownloadClasse

Test Case 1:

The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (7), and Test Results. The Body tab is selected, and the response is displayed in 'Pretty' format. The status is '200 OK'. The response content is Java code for a class implementing a parser.

```
1
2 import java.io.Serializable;
3 import java.util.StringTokenizer;
4 import java.util.Vector;
5
6 /**
7  * This class implements a parser to read properties that have
8  * a hierarchy(i.e. tree) structure. Conceptually it's similar to
9  * the XML DOM/SAX parser but of course is much simpler and
10 * uses dot as the separator of levels instead of back-slash.<br>
11 * It provides interfaces to both build a parser tree and traverse
12 * the tree.
```

Test Case 2:

The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (4), and Test Results. The Body tab is selected, and the response is displayed in 'JSON' format. The status is '500 Internal Server Error'. The response content is a JSON object.

```
1 {
2   "timestamp": "2023-07-13T10:44:27.422+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "path": "/downloadFile/testttttt"
6 }
```

Test Case 3:

The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (4), and Test Results. The Body tab is selected, and the response is displayed in 'JSON' format. The status is '500 Internal Server Error'. The response content is a JSON object.

```
1 {
2   "timestamp": "2023-07-13T10:44:27.422+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "path": "/downloadFile/testttttt"
6 }
```

Test Case 4:

The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (7), and Test Results. The Body tab is selected, and the response is displayed in 'Text' format. The status is '200 OK'. The response content is a file listing.

```
1 HELP.md
2 target/
3 !.mvn/wrapper/maven-wrapper.jar
4 !***/src/main/**/target/
5 !***/src/test/**/target/
6
7 ### STS ###
8 .apt_generated
9 .classpath
```

7.2.4 Testing ModificaClasse

Test Case 1:

-Valori Prima della modifica:

```
[
  {
    "name": "ByteArrayHashMap",
    "date": "2023-07-13",
    "difficulty": "Intermediate",
    "code_uri": "Files-Upload/ByteArrayHashMap/ByteArrayHashMap.java",
    "description": "Descrizione prova",
    "category": [
      "Hashing",
      "Security",
      "Mapping"
    ]
  }
]
```

-Richiesta di modifica con risultato:

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8080/update/ByteArrayHashMap
- Body:** JSON


```
1 {
2   "name": "ByteArrayHashMap",
3   "date": "2023-07-13",
4   "difficulty": "Intermediate",
5   "description": "Descrizione cambiata",
6   "category": [
7     "Hashing",
8     "Byte",
9     "Mapping"
10  ]
11 }
```
- Status:** 200 OK, 11 ms, 201 B
- Response:** 1 Aggiornamento eseguito correttamente.

-Post Condizione, valori aggiornati:

```
[
  {
    "name": "ByteArrayHashMap",
    "date": "2023-07-13",
    "difficulty": "Intermediate",
    "code_uri": "Files-Upload/ByteArrayHashMap/ByteArrayHashMap.java",
    "description": "Descrizione cambiata",
    "category": [
      "Hashing",
      "Byte",
      "Mapping"
    ]
  }
]
```

Test Case 2:

The screenshot shows a REST client interface with the following details:

- Status:** 404 Not Found, 16 ms, 224 B
- Response:** 1 Nessuna classe trovata o nessuna modifica effettuata.

Test Case 3:

-Valori Prima della modifica:

```
[
  {
    "name": "ByteArrayHashMap",
    "date": "2023-07-13",
    "difficulty": "Intermediate",
    "code_uri": "Files-Upload/ByteArrayHashMap/ByteArrayHashMap.java",
    "description": "Descrizione cambiata",
    "category": [
      "Hashing",
      "Byte",
      "Mapping"
    ]
  }
]
```

-Richiesta di modifica con risultato:

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/update/ByteArrayHashMap`. The request body is a JSON object. The response is a 200 OK status with a 7 ms response time and 201 B of data. The response body contains the message "Aggiornamento eseguito correttamente."

```
POST http://localhost:8080/update/ByteArrayHashMap
Content-Type: application/json

{
  "name": "ByteArrayHashMap",
  "date": "2023-07-13",
  "difficulty": "Intermediate",
  "code_uri": "Files-Upload/ByteArrayHashMap/ByteArrayHashMap.java",
  "description": "Descrizione cambiata parzialmente, il resto è uguale",
  "category": [
    "Hashing",
    "Byte",
    "Mapping"
  ]
}
```

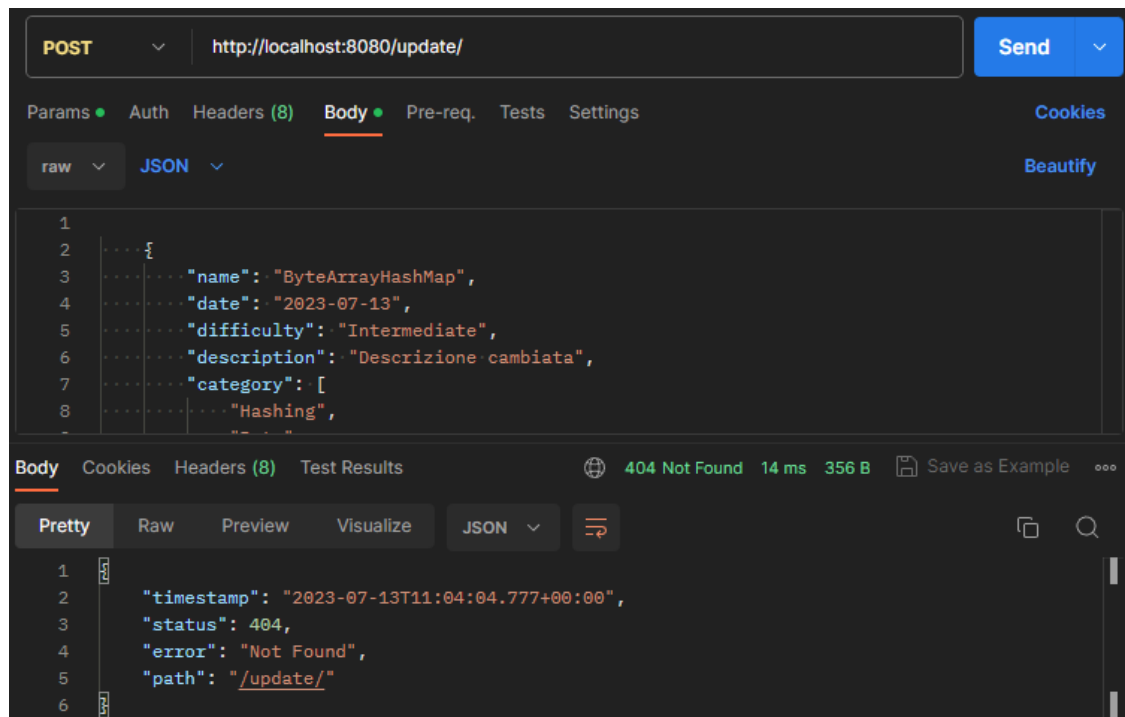
Body: Cookies Headers (5) Test Results 200 OK 7 ms 201 B Save as Example

1 Aggiornamento eseguito correttamente.

-Post Condizione, valori aggiornati:

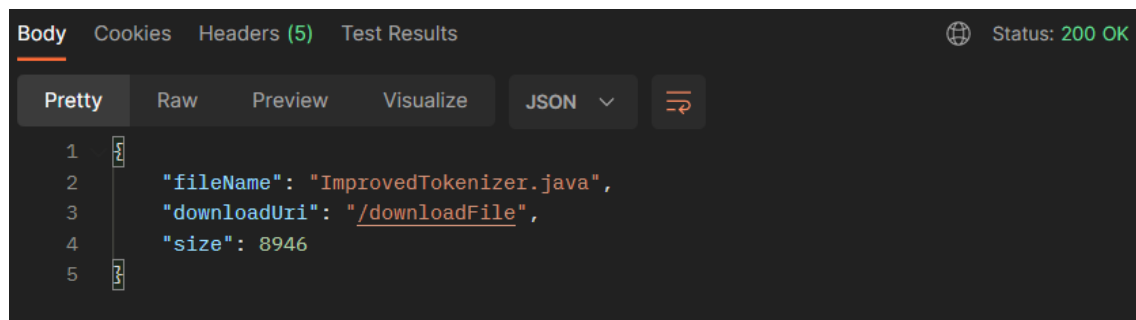
```
{
  "name": "ByteArrayHashMap",
  "date": "2023-07-13",
  "difficulty": "Intermediate",
  "code_uri": "Files-Upload/ByteArrayHashMap/ByteArrayHashMap.java",
  "description": "Descrizione cambiata parzialmente, il resto è uguale",
  "category": [
    "Hashing",
    "Byte",
    "Mapping"
  ]
}
```

Test Case 4:

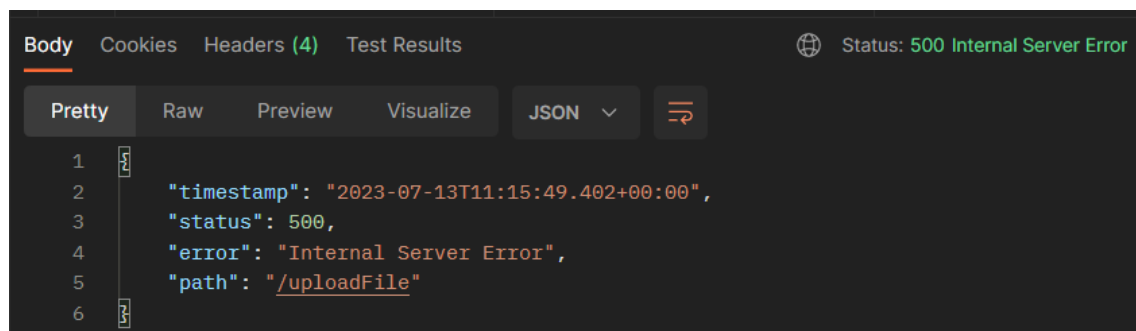


7.2.5 Testing UploadClasse

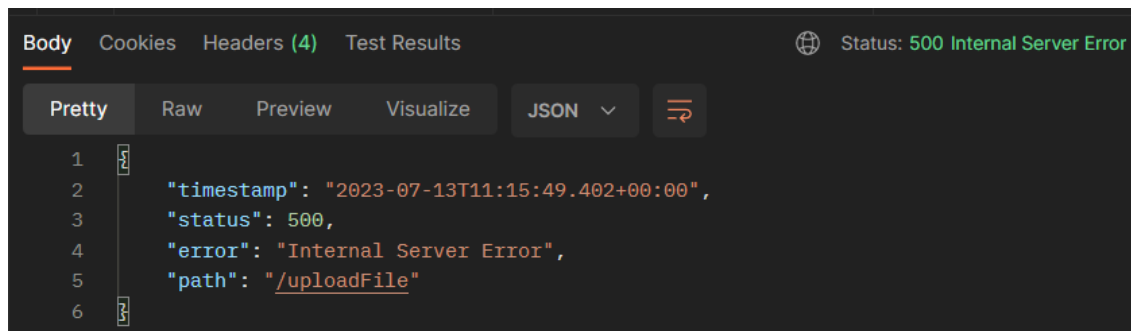
Test Case 1:



Test Case 2:

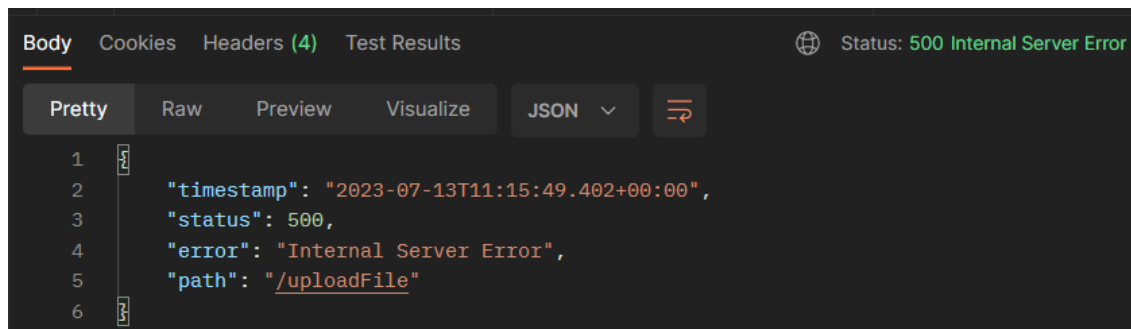


Test Case 3:



```
Body Cookies Headers (4) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "timestamp": "2023-07-13T11:15:49.402+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "path": "/uploadFile"
6 }
```

Test Case 4:



```
Body Cookies Headers (4) Test Results
Pretty Raw Preview Visualize JSON
1 {
2   "timestamp": "2023-07-13T11:15:49.402+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "path": "/uploadFile"
6 }
```

Capitolo 8: Installazione

Il presente capitolo fornisce un'analisi approfondita sull'importanza dell'installazione del software e illustra il processo dettagliato per installare correttamente e eseguire un'applicazione web. Saranno esplorati i passaggi necessari per assicurarsi che l'installazione venga effettuata con successo e che l'applicazione web possa essere eseguita senza problemi. Saranno fornite istruzioni chiare e dettagliate per guidare gli utenti attraverso il processo di installazione, compresi i requisiti di sistema, la configurazione delle dipendenze e le procedure di avvio dell'applicazione web. L'obiettivo è fornire una guida completa per garantire una corretta installazione e un'esperienza di esecuzione senza intoppi dell'applicazione web.

8.1 Installazione software in locale

Per l'installazione in locale del software, l'applicazione viene fornita come file JAR chiamato "manvsc.jar". È importante assicurarsi che il file di configurazione "application.properties" sia presente nella stessa cartella del file JAR per garantire un corretto funzionamento. Attraverso il file di configurazione, è possibile configurare diversi parametri, inclusi quelli relativi al database e alla connessione ad esso.

Per garantire la compatibilità, il sistema su cui viene installato il software deve avere installata una versione di Java 17 o successiva e un database documentale MongoDB. Assicurarsi che queste dipendenze siano soddisfatte prima di procedere con l'installazione.

È importante notare che eventuali modifiche apportate al progetto originale richiederanno la rigenerazione del file JAR, quindi sarà necessario generare un nuovo file JAR con le modifiche applicate prima di eseguire l'installazione o l'esecuzione dell'applicazione.

8.1.1 Generazione del file jar

Utilizzando l'IDE "Eclipse" è stata utilizzata la seguente procedura per la generazione del file Jar:

1. Importare il progetto all'interno dell'IDE. A tale scopo: File->Open Projects from File System-> Selezionare come Import source la cartella del progetto-> Spuntare la casella "Search for nested projects" -> Finish.
2. Una volta che il progetto viene caricato e buildato e le dipendenze risolte bisogna cliccare su File->Export->Java->Jar File-> Selezionare l'intera cartella "manvsclass" da esportare->Finish.

NB: Se sono presenti dei jar file precedenti vanno prima eliminati altrimenti si genererà un errore nella generazione del file jar.

8.1.2 Esecuzione con Docker Desktop

Docker Desktop è un'applicazione che consente di creare e eseguire oggetti in un ambiente virtuale chiamati "Docker". In questo contesto, gli oggetti Docker si riferiscono all'applicazione web e al database MongoDB. Per crearli, è necessario definire i parametri di configurazione in un file apposito chiamato "docker-compose.yml". Questi parametri definiscono le dipendenze necessarie per l'esecuzione dell'applicazione.

Inoltre, è richiesto un altro file chiamato "Dockerfile" per specificare i parametri del container che si desidera generare. Questi parametri includono le porte che devono essere esposte all'esterno, il file eseguibile, l'ambiente in cui deve essere eseguito il file "manvsclass.jar" e l'entry point, che specifica quale file deve essere eseguito come file principale all'interno del container.

Il file "docker-compose.yml" contiene due sezioni: la prima riguarda la configurazione dell'applicazione principale e la seconda riguarda la configurazione del database. Per installare l'applicazione utilizzando Docker, è necessario creare i file "docker-compose.yml" e "Dockerfile" con le corrette configurazioni.

```
# Versione del Compose
version: '3.12.12'

# Servizi
services:
  # Servizio controller
  controller:
    build: .
    restart: always
    ports:
      - 8080:8080
    depends_on:
      - mongo_db
    volumes:
      - ./FilesUpload:/Files-Upload
  # Servizio MongoDB
  mongo_db:
    image: "mongo:6.0.6"
    restart: always
    ports:
      - 27017:27017
    volumes:
      - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
```

Docker-Compose

```
FROM maven AS build
WORKDIR /app
COPY pom.xml /app
RUN mvn dependency:resolve
COPY . /app
RUN mvn clean
RUN mvn package -DskipTests

FROM eclipse-temurin:20-jdk-alpine
COPY --from=build /app/target/*.jar manvsclass.jar
ENV PORT 8080
EXPOSE 8080
CMD ["java", "-jar", "manvsclass.jar"]
```

DockerFile

8.1.3 Esecuzione con parametri di Default

In questo caso l'applicazione viene fornita già con il docker-compose e DockerFile pronti all'esecuzione. Per poter creare il container e i suoi Docker bisogna eseguire nella cartella principale il comando da power shell Windows:

```
docker-compose build
docker-compose up
```

Con privilegi di amministratore

8.1.4 Definizione degli indici di ricerca nel database

Ecco una descrizione dei passaggi da effettuare per l'utilizzo dei comandi nel terminale del container MongoDB:

1. Utilizzare dal terminale del container MongoDB il comando "mongosh":
Questo comando consente di avviare l'interfaccia interattiva della shell di MongoDB all'interno del terminale del container. Una volta avviata la shell, sarà possibile eseguire comandi e interrogazioni sul database.
2. Utilizzare il comando "use manvsclass" per selezionare il database "manvsclass": Questo comando consente di selezionare il database "manvsclass" come database di lavoro corrente. Tutti i comandi successivi verranno eseguiti all'interno di questo database.
3. Utilizzare i comandi :

```
db.createCollection("ClassUT");
```

```
db.createCollection("interaction");
```

```
db.createCollection("Admin");
```

```
db.createCollection("Operation");
```

Per creare le collezioni: Questi comandi creano le collezioni nel database corrente. In particolare, vengono create le collezioni "ClassUT", "interaction", "Admin" e "Operation".

4. Utilizzare i comandi:

```
db.ClassUT.createIndex({ difficulty: 1 }) db.Interaction.createIndex({ name:  
"text", type: 1 }) db.interaction.createIndex({ name: "text" })  
db.Admin.createIndex({username: 1})
```

Per creare gli indici: Questi comandi creano gli indici sulle collezioni specificate. Ad esempio, "db.ClassUT.createIndex({ difficulty: 1 })" crea un indice sulla collezione "ClassUT" con il campo "difficulty" ordinato in modo ascendente.

Una volta generati questi indici di ricerca, le pipeline funzioneranno correttamente e sarà quindi possibile utilizzare il software in modo corretto.