

UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E
DELLE TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

Corso di Software Architecture Design

Documentazione A13-R6-B15

DOCENTE

Prof.ssa Anna Rita Fasolino

STUDENTI

Vittorio Guerra – M63001850
Fabrizio Imparato – M63001758
Tommaso Rivieccio – M63001845

ANNO ACCADEMICO 2024-2025

CONTATTI

Vittorio Guerra – vit.guerra@studenti.unina.it
Fabrizio Imparato – fabri.imparato@studenti.unina.it
Tommaso Rivieccio – t.rivieccio@studenti.unina.it

Il seguente documento fa riferimento al progetto presentato e sviluppato
nella seguente repository di GitHub: <https://github.com/vittorioguerra/A13-R6-B15>

Repository di partenza: <https://github.com/lor3nzopoly/A13>

Repository master: <https://github.com/Testing-Game-SAD-2023/A13>

Indice

1.	Punto di partenza e obiettivi del progetto	6
1.1.	Versione di partenza.....	6
1.2.	Nuovi requisiti assegnati	10
1.3.	Processo di sviluppo.....	10
1.4.	Strumenti utilizzati.....	14
2.	Analisi dei Requisiti	15
2.1.	Diagramma dei casi d'uso.....	15
2.2.	Diagramma di sequenza	18
3.	Analisi d'impatto	19
4.	Progettazione della soluzione per realizzare i requisiti richiesti	21
5.	Implementazione.....	25
5.1.	Modifiche ai file di installazione e configurazione.....	25
5.2.	Modifiche ai file e ai package	26
5.2.1.	Model	26
5.2.2.	Responses.....	27
5.2.3.	Service.....	27
5.2.4.	Controller	32
5.2.5.	Pagine HTML	35
5.2.6.	UI Gateway	36
5.3.	Aggiunta di API	37
6.	Architettura finale	39
7.	Testing	43
7.1.	Testing GUI.....	43
7.2.	Testing API	47
7.2.1.	GET/classes.....	47
7.2.1.1.	Test 1	47
7.2.1.2.	Test 2	48
7.2.1.3.	Test 3	49
7.2.2.	GET/classes/{className}	50
7.2.2.1.	Test 1	50
7.2.2.2.	Test 2	51
7.2.2.3.	Test 3	52
7.2.3.	GET/classes/{className}/robots	53
7.2.3.1.	Test 1	53
7.2.3.2.	Test 2	54
7.2.3.3.	Test 3	55
7.2.3.4.	Test 4	56

7.2.4.	GET/classes/{className}/{robotName}	57
7.2.4.1.	Test 1	57
7.2.4.2.	Test 2	58
7.2.4.3.	Test 3	59
7.2.4.4.	Test 4	60
7.2.5.	POST/classes/{className}	61
7.2.5.1.	Test 1	61
7.2.5.2.	Test 2	62
7.2.5.3.	Test 3	63
7.2.6.	POST/classes/{className}/{robotName}	64
7.2.6.1.	Test 1	64
7.2.6.2.	Test 2	65
7.2.6.3.	Test 3	66
7.2.6.4.	Test 4	67
7.2.6.5.	Test 5	68
7.2.7.	DELETE/classes/{className}	69
7.2.7.1.	Test 1	69
7.2.7.2.	Test 2	70
7.2.7.3.	Test 3	71
7.2.8.	DELETE/classes/{className}/{robotName}	72
7.2.8.1.	Test 1	72
7.2.8.2.	Test 2	73
7.2.8.3.	Test 3	74
7.2.8.4.	Test 4	75
7.2.9.	POST/fileSystem	76
7.2.9.1.	Test 1	76
7.2.9.2.	Test 2	77
7.2.9.3.	Test 3	78
7.2.9.4.	Test 4	79
7.2.10.	DELETE/fileSystem	80
7.2.10.1.	Test 1	80
7.2.10.2.	Test 2	81
7.2.10.3.	Test 3	82
7.2.11.	POST/lock	83
7.2.11.1.	Test 1	83
7.2.11.2.	Test 2	84
7.2.11.3.	Test 3	85

7.2.12.	POST/unlock.....	86
7.2.12.1.	Test 1	86
7.2.12.2.	Test 2.....	87
7.2.12.3.	Test 3.....	88
7.3.	Testing sulla concorrenza.....	89
8.	Sviluppi futuri.....	91

1. Punto di partenza e obiettivi del progetto

Il lavoro si colloca all'interno del progetto ERASMUS denominato **ENACTEST** (**E**uropean **i****Nnovative **A**llian**C**e for **T**ESTing **e**duca**T**ion), sviluppato in parte dagli studenti del corso di Software Architecture Design dell'Università degli Studi di Napoli Federico II. Questo progetto si propone un obiettivo ambizioso: valorizzare il ruolo del testing, una disciplina spesso trascurata e poco approfondita nei corsi universitari, attraverso l'adozione della *gamification*. Tale strategia innovativa consiste nell'integrare elementi tipici dei giochi in contesti non ludici.**

L'applicazione di questo approccio ha portato alla progettazione e sviluppo del gioco interattivo "*Man vs Automated Testing Tools Challenges*". In questa sfida, gli studenti, denominati players, competono ideando test con il framework JUnit contro robot come Randoop o EvoSuite, in grado di generare automaticamente test. La vittoria spetta al partecipante che riesce a raggiungere un determinato obiettivo di copertura.

1.1. Versione di partenza

L'applicazione ENACTEST adotta un pattern architettonico MVC (Model-View-Controller) basato su uno stile Client-Server. Inoltre, l'architettura è stata progettata utilizzando microservizi.

Il progetto ha avuto origine dalla versione A13. Di seguito sono presentati alcuni diagrammi che illustrano lo stato iniziale dell'applicazione da cui partirà il nostro intervento.

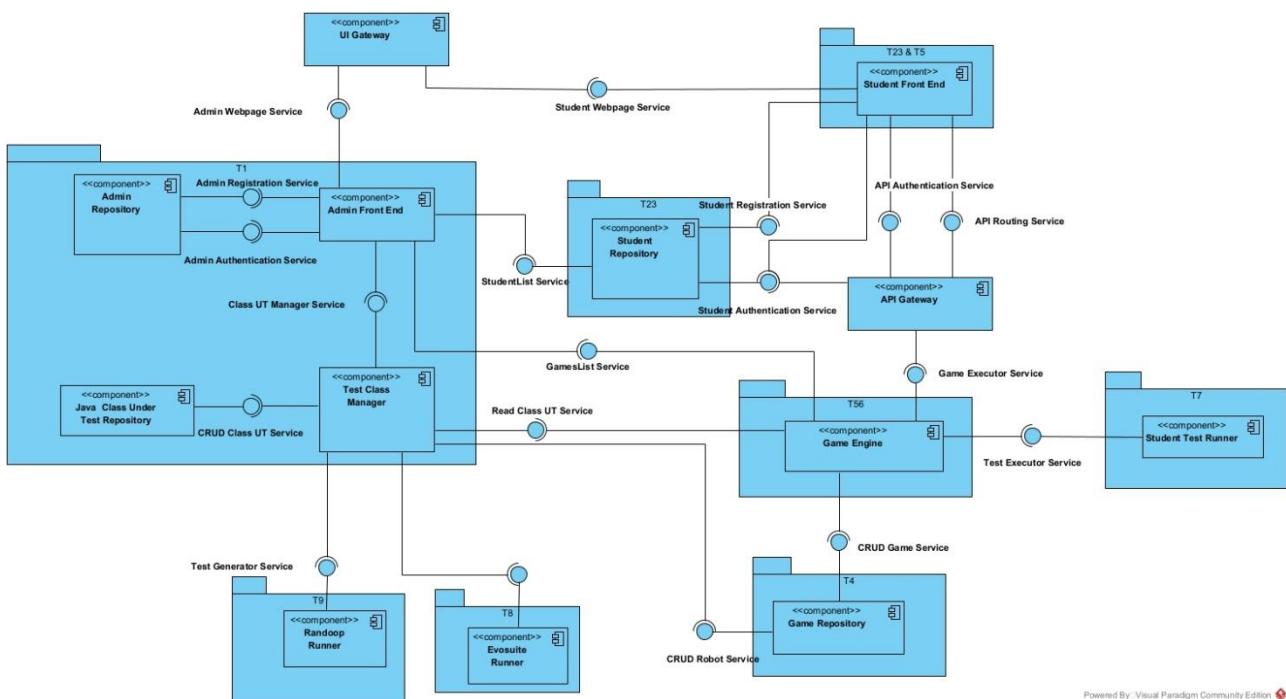


Figura 1: Composite Diagram della versione iniziale

Powered By: Visual Paradigm Community Edition

Si riporta anche il deployment diagram al fine di mostrare i componenti che compongono il sistema:

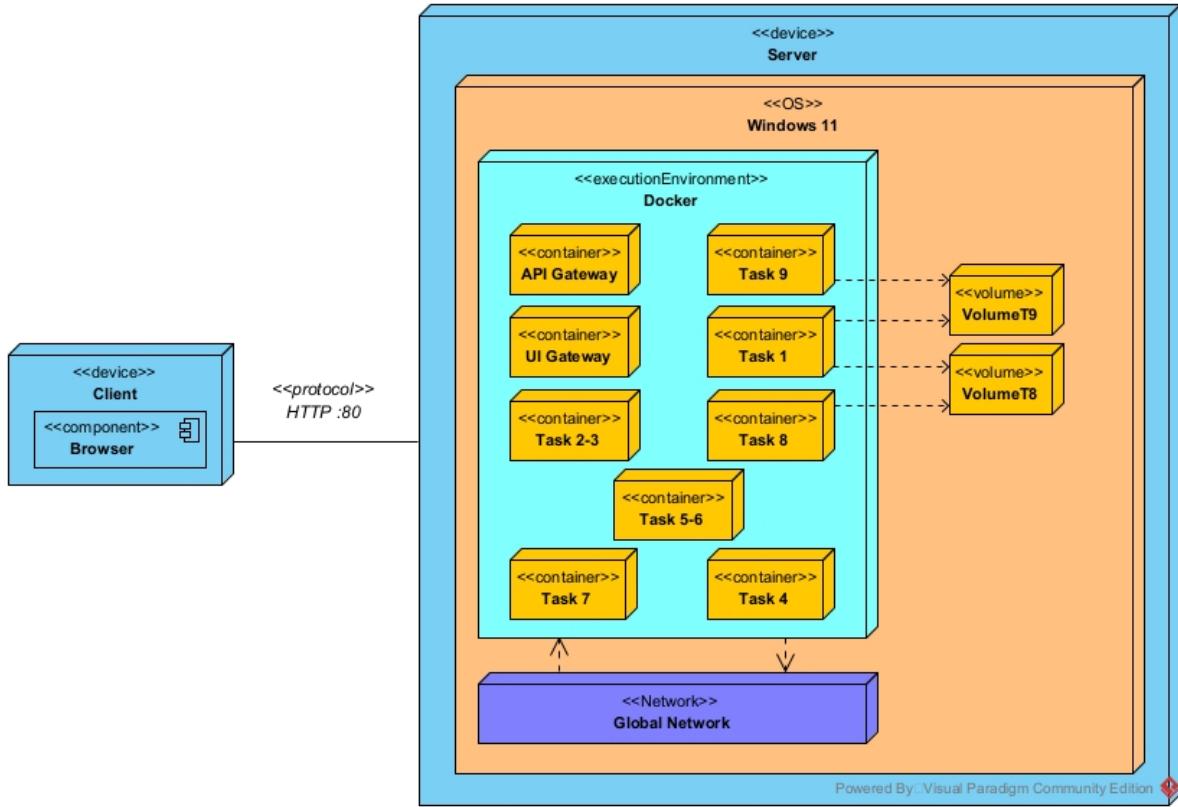


Figura 2: Deployment Diagram della versione iniziale

Come si può notare, l'architettura è composta da nove container Docker, ciascuno con un ruolo specifico. Di seguito una breve descrizione del compito di ognuno:

- **T1:** consente agli amministratori di registrarsi e autenticarsi tramite login. Inoltre, offre una dashboard che permette di gestire classi e test, includendo funzionalità per aggiungerli, caricarli, scaricarli ed eliminarli. È inoltre possibile visualizzare l'elenco dei giocatori registrati.
- **T23:** permette ai players di registrarsi e autenticarsi tramite un login dedicato, garantendo loro l'accesso a un'area riservata.
- **T4:** permette al game engine di creare, aggiornare, recuperare ed eliminare una partita. Inoltre, fornisce anche una logica di gestione dei round e dei turni associati a questi ultimi.
- **T56:** permette ai players correttamente autenticati di accedere all'area riservata per la gestione dei parametri di gioco, dove è possibile scegliere classe, robot e difficoltà. Il servizio mette a disposizione anche un editor di testo dove è possibile scrivere la propria classe di test. Successivamente il player ha la possibilità di compilare il codice e di inviarlo per andare a valutare la copertura per poi confrontarla con quella del robot scelto.
- **T7:** permette di compilare ed eseguire i casi di test prodotti dal player in partita.
- **T8:** permette di eseguire il robot EvoSuite su di una data classe Java e restituire in output le informazioni relative l'esito dei test prodotti dal robot.

- **T9:** permette di eseguire il robot Randoop su di una data classe Java e restituire in output le informazioni relative l'esito dei test prodotti dal robot.

Per integrare tutti questi servizi viene utilizzato il Gateway pattern, il cui scopo è quello di fornire un unico entry point per tutte le richieste. L'implementazione di questo pattern comporta la creazione di due componenti principali:

- **UI Gateway:** punto di accesso al sistema, gestisce le richieste in entrata per poi smistarle ai vari container. Se la richiesta inizia con /api viene reindirizzata all'API Gateway.
- **API Gateway:** inoltra le richieste provenienti dall'UI Gateway verso i vari container gestendo autenticazione e autorizzazione attraverso i token.

Sono presenti anche due volumi:

- **VolumeT8:** volume condiviso tra i container T1 e T8, memorizza le classi e i test di EvoSuite.
- **VolumeT9:** volume condiviso tra i container T1 e T9, memorizza le classi e i test di Randoop.

Per motivi che verranno chiariti nei capitoli successivi, riportiamo anche alcuni diagrammi specifici per il container T1 al fine di comprenderne il comportamento.

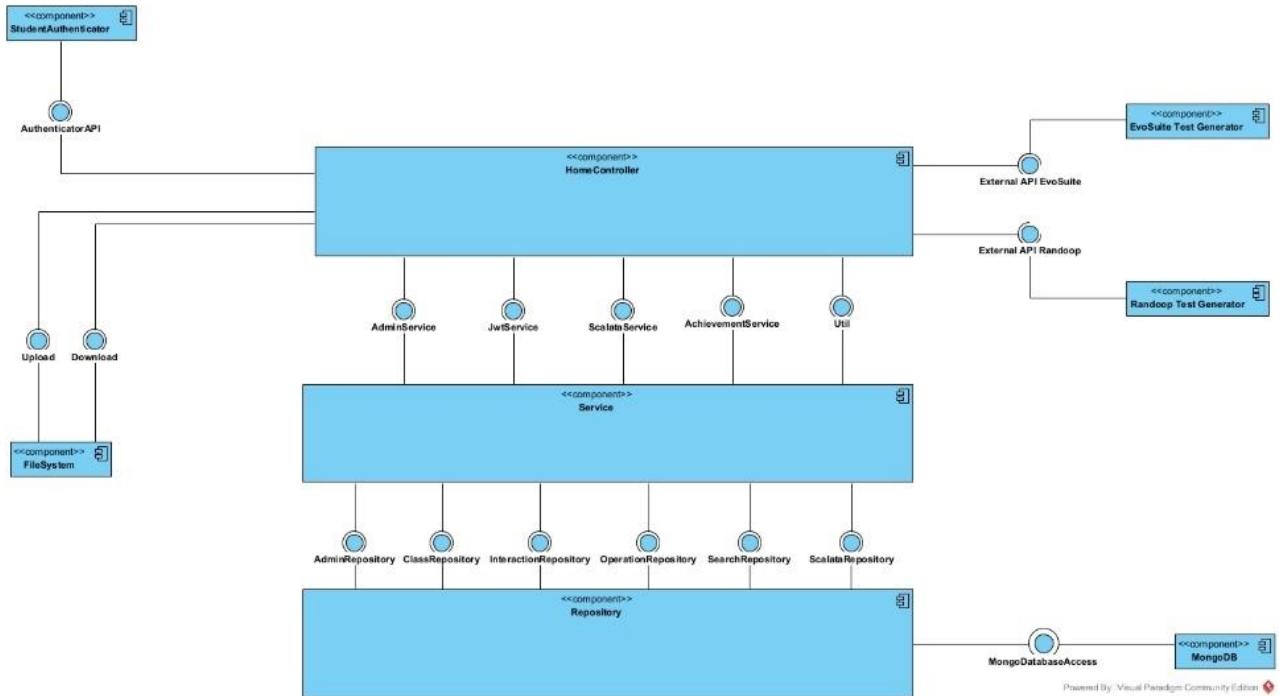


Figura 3: Component Diagram iniziale del container T1

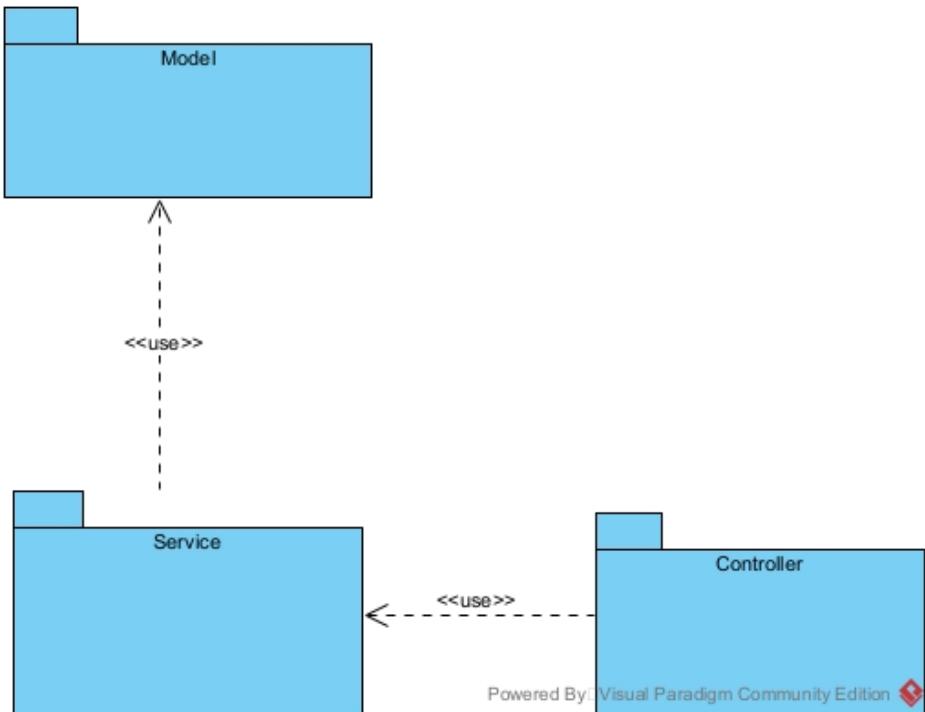


Figura 4: Package Diagram iniziale del container T1

Come strumento di persistenza dei dati per T1 si è utilizzato MongoDB. Proprio per la sua natura non relazionale, si è scelto di adottare un Document JSON like per mostrare lo stato attuale del DB e delle collection che ne fanno parte:

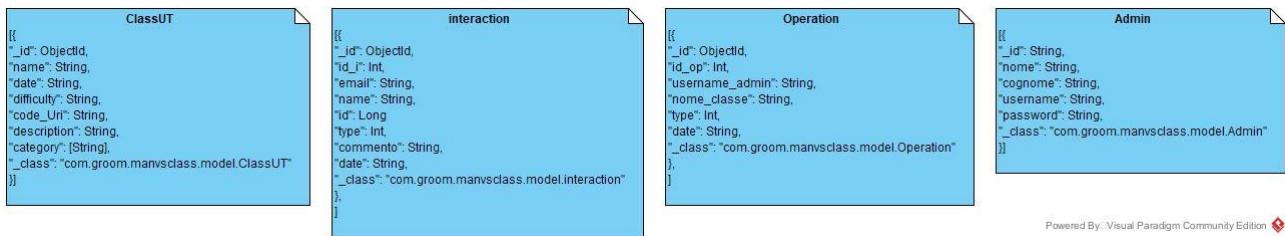


Figura 5: Collection iniziali presenti del database MongoDB

1.2. Nuovi requisiti assegnati

Il requisito assegnato è il seguente:

ID	Descrizione
R6	Studiare in T1 le modalità di gestione dei test dei Robot per offrire all'Amministratore la possibilità di caricare anche classi di test generati da altri robot (oltre a Randoop ed Evosuite) o con altre tecniche (AI, Manuale, etc..).

1.3. Processo di sviluppo

Il gruppo ha scelto di seguire un approccio Agile, basato sul framework SCRUM. Questo metodo si è rivelato particolarmente adatto per gestire la complessità del progetto, suddividendo le attività in fasi più piccole e facilmente monitorabili.

Il gruppo si è affidato a Trello, uno strumento collaborativo che ha facilitato la comunicazione e il coordinamento tra i membri, specialmente durante le fasi di pianificazione e revisione.

All'inizio di ogni iterazione, è stato creato un backlog contenente gli obiettivi principali da raggiungere. Durante lo svolgimento delle attività, il backlog è stato aggiornato dinamicamente, suddividendo le attività in tre categorie: quelle pianificate, quelle in corso e quelle completate. Questo approccio ha consentito al gruppo di avere sempre una visione chiara dello stato del progetto e delle priorità.

In tutto abbiamo previsto tre sprint dalla durata di una settimana ognuno. Ad ogni attività è stato assegnato un effort utilizzando la sequenza di Fibonacci:

- 0 (Nessun lavoro richiesto)
- 1 (Molto piccolo, banale)
- 2 (Piccolo, semplice)
- 3 (Lavoro richiesto moderato, un po' di complessità)
- 5 (Più grande, più complesso, ma ancora gestibile)
- 8 (Complesso, potrebbe richiedere più risorse)
- 13 (Molto complesso o incerto)
- 21 (Estremamente complesso, eventualmente da suddividere in attività più piccole)

Di seguito sono riportati come sono state organizzate le attività nei vari sprint e come queste ultime sono state ripartite tra il team di sviluppo.

Sprint #1

The screenshot shows the SAD - Task R6 application interface. On the left, the "Product Backlog" section contains several items with their respective story points and team members assigned (VG, TR, FG). In the center, the "Current Sprint Backlog" section lists three items: 1. Create the shared volume between containers (Story point 1, assigned to VG), 2. Modify the DB structure to support generalization (Story point 2, assigned to FG), and 3. Modify the "AddClassAndTest" page to support multiple robots (Story point 5, assigned to TR). To the right, three completed sprints are shown: Sprint #1 Complete (1 item), Sprint #2 Complete (1 item), and Sprint #3 Complete (1 item). Each completed sprint has a "Addi una scheda" button.

Sprint #2

The screenshot shows the SAD - Task R6 application interface. The "Product Backlog" section contains two items: 1. Create API for interacting with the shared file system (Story point 8, assigned to TR and FG) and 2. Clean code and reorganize packages linearly (Story point 3, assigned to VG). The "Current Sprint Backlog" section lists three items: 1. Add logic for managing configuration files (Story point 2, assigned to VG), 2. Modify delete logic for classes and tests (Story point 8, assigned to TR), and 3. Modify save logic for classes and tests (Story point 5, assigned to FG). To the right, three completed sprints are shown: Sprint #1 Complete (1 item), Sprint #2 Complete (1 item), and Sprint #3 Complete (1 item). Each completed sprint has a "Addi una scheda" button.

Sprint #3

The screenshot shows a digital task board with four columns: Product Backlog, Current Sprint Backlog, Sprint #1 Complete, Sprint #2 Complete, and Sprint #3 Complete.

- Product Backlog:** Contains a button "+ Aggiungi una scheda".
- Current Sprint Backlog:** Contains two items:
 - Item 8: "Creare API per l'interazione con il file system condiviso." Status: VG (Green)
 - Item 3: "Pulizia del codice e riorganizzazione dei package in modo lineare." Status: VG (Green)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #1 Complete:** Contains three items:
 - Item 1: "Creare il volume condiviso tra i container." Status: VG (Green)
 - Item 2: "Modificare la struttura del DB per permettere la generalizzazione dei Robot." Status: F (Yellow)
 - Item 5: "Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #2 Complete:** Contains two items:
 - Item 2: "Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili." Status: VG (Green)
 - Item 8: "Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #3 Complete:** Contains two items:
 - Item 3: "Pulizia del codice e riorganizzazione dei package in modo lineare." Status: VG (Green)
 - Item 8: "Creare API per l'interazione con il file system condiviso." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.

Situazione finale

The screenshot shows the same digital task board after the completion of Sprint #3.

- Product Backlog:** Contains a button "+ Aggiungi una scheda".
- Current Sprint Backlog:** Contains two items:
 - Item 8: "Creare API per l'interazione con il file system condiviso." Status: VG (Green)
 - Item 3: "Pulizia del codice e riorganizzazione dei package in modo lineare." Status: VG (Green)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #1 Complete:** Contains three items:
 - Item 1: "Creare il volume condiviso tra i container." Status: VG (Green)
 - Item 2: "Modificare la struttura del DB per permettere la generalizzazione dei Robot." Status: F (Yellow)
 - Item 5: "Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #2 Complete:** Contains two items:
 - Item 2: "Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili." Status: VG (Green)
 - Item 8: "Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.
- Sprint #3 Complete:** Contains two items:
 - Item 3: "Pulizia del codice e riorganizzazione dei package in modo lineare." Status: VG (Green)
 - Item 8: "Creare API per l'interazione con il file system condiviso." Status: TR (Red)A button "+ Aggiungi una scheda" is located at the bottom.

Successivamente sono state effettuate alcune operazioni di Adjustment, per le quali è stato pianificato e realizzato un ulteriore Sprint dedicato alla gestione dei nuovi task.

Sprint #4 Adjustment

The screenshot shows a digital task board interface with the following sections:

- Product Backlog:** Contains a button "+ Aggiungi una scheda".
- Current Sprint Backlog:** Contains 8 tasks:
 - 1: Aggiungere gestione concorrenza e rollback per il File system. Status: F (blue)
 - 2: Test regressione API e GUI. Status: VG (orange)
 - 2: Test concorrenza con Ngrok. Status: VG (orange)
 - 8: Creare nuove API per la gestione attraverso un path generico del File system condiviso. Status: TR (red)
- Sprint #4 Complete:** Contains 3 tasks:
 - 1: Creare il volume condiviso tra i container. Status: VG (orange)
 - 2: Modificare la struttura del DB per permettere la generalizzazione dei Robot. Status: F (blue)
 - 5: Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. Status: TR (red)
- Sprint #1 Complete:** Contains 3 tasks:
 - 1: Creare il volume condiviso tra i container. Status: VG (orange)
 - 2: Modificare la struttura del DB per permettere la generalizzazione dei Robot. Status: F (blue)
 - 5: Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. Status: TR (red)
- Sprint #2 Complete:** Contains 3 tasks:
 - 2: Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili. Status: VG (orange)
 - 8: Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. Status: TR (red)
 - 8: Modificare la logica di salvataggio delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. Status: F (blue)

Situazione finale

The screenshot shows the same digital task board interface, but with the following changes:

- Product Backlog:** Contains a button "+ Aggiungi una scheda".
- Current Sprint Backlog:** Contains 8 tasks:
 - 1: Aggiungere gestione concorrenza e rollback per il File system. Status: F (blue)
 - 2: Test regressione API e GUI. Status: VG (orange)
 - 2: Test concorrenza con Ngrok. Status: VG (orange)
 - 8: Creare nuove API per la gestione attraverso un path generico del File system condiviso. Status: TR (red)
- Sprint #4 Complete:** Contains 3 tasks:
 - 1: Creare il volume condiviso tra i container. Status: VG (orange)
 - 2: Modificare la struttura del DB per permettere la generalizzazione dei Robot. Status: F (blue)
 - 5: Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. Status: TR (red)
- Sprint #1 Complete:** Contains 3 tasks:
 - 1: Creare il volume condiviso tra i container. Status: VG (orange)
 - 2: Modificare la struttura del DB per permettere la generalizzazione dei Robot. Status: F (blue)
 - 5: Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. Status: TR (red)
- Sprint #2 Complete:** Contains 3 tasks:
 - 2: Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili. Status: VG (orange)
 - 8: Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. Status: TR (red)
 - 8: Modificare la logica di salvataggio delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. Status: F (blue)

1.4. Strumenti utilizzati

Durante lo sviluppo del progetto sono stati utilizzati diversi strumenti per ottimizzare la collaborazione, la gestione del codice e la modellazione delle funzionalità:

- **Microsoft Teams:** Impiegato per il coordinamento delle attività e il monitoraggio costante dei progressi. È risultato utile anche per lo scambio immediato di file e messaggi, facilitando un efficace coordinamento tra i membri del team.
- **GitHub:** Selezionato come repository centrale per la gestione del codice, ha permesso di mantenere un flusso di sviluppo incrementale. Il controllo di versione ha garantito un codice sempre aggiornato e condiviso tra tutti i partecipanti al progetto.
- **Visual Paradigm:** Utilizzato nelle fasi di progettazione e analisi dei requisiti per la creazione dei diagrammi UML inclusi nella documentazione. Questo strumento ha consentito di rappresentare graficamente l'architettura e i flussi del sistema, facilitando la comprensione della struttura del progetto.
- **Docker:** Adottato come strumento per la containerizzazione, grazie alla sua efficienza nella creazione di ambienti isolati. Le principali caratteristiche apprezzate sono la leggerezza, la flessibilità e la capacità di creare ambienti standardizzati, garantendo l'esecuzione coerente del software su diverse piattaforme. Inoltre, il supporto per molteplici linguaggi e stack tecnologici ne ha favorito l'utilizzo in contesti di sviluppo differenti.
- **Visual Studio Code:** Utilizzato come editor di codice per scrittura e test. L'integrazione nativa con GitHub ha semplificato la gestione dei branch direttamente all'interno dell'ambiente di sviluppo.
- **Postman:** Utilizzato per facilitare il testing delle API sviluppate durante il progetto. Ha permesso al team di inviare richieste HTTP direttamente dall'IDE, per verificare le risposte e testare l'interazione tra i vari servizi del sistema senza dover lasciare l'ambiente di sviluppo.
- **Ngrok:** Utilizzato per esporre localmente i servizi in sviluppo a Internet in modo rapido e sicuro. Grazie a questo strumento, è stato possibile creare tunnel HTTPS temporanei, semplificando il testing e l'integrazione di servizi esterni. Ngrok si è dimostrato particolarmente utile per condividere endpoint locali con il team o per testare le API su dispositivi remoti, garantendo un flusso di lavoro flessibile e immediato.

2. Analisi dei Requisiti

Per supportare questa analisi, verranno presentati diversi diagrammi. Ciascun diagramma illustrerà in dettaglio le aree coinvolte e le interazioni tra i vari elementi del sistema interessati dagli interventi.

Dopo una prima analisi siamo riusciti ad individuare i requisiti funzionali e non funzionali.

ID	Descrizione
RF01	Il sistema deve permettere all'Amministratore di caricare classi di test generate da robot esterni (oltre a Randoop ed Evosuite), assicurandosi che rispettino un formato standard configurabile.
RF02	Il sistema deve gestire una lista di robot disponibili configurabile dall'Amministratore, garantendo uno standard di nomenclatura uniforme per la loro identificazione.
RF03	Il sistema deve centralizzare i dati relativi ai robot e alle classi di test caricate, garantendo un accesso strutturato e uniforme da parte di T1 e altri componenti autorizzati.
RF04	Il container T1 deve mettere a disposizione un'API REST per consentire l'interazione con i dati centralizzati, fornendo operazioni CRUD sui robot e le classi di test.
RNF01	Il sistema deve rispettare i principi di separazione delle responsabilità tra i componenti, garantendo che ogni interazione avvenga attraverso i container o servizi previsti nell'architettura.

2.1. Diagramma dei casi d'uso

Il Diagramma dei casi d'uso è uno strumento essenziale per rappresentare le interazioni tra gli utenti e il sistema, delineando le principali funzionalità e le relazioni tra di esse. Questo tipo di diagramma aiuta a comprendere come gli utenti finali interagiscono con il sistema e quali sono le operazioni disponibili.

Qui di seguito si riporta il diagramma dei casi d'uso completo dove viene evidenziato in verde il punto in cui si è effettuata la modifica.

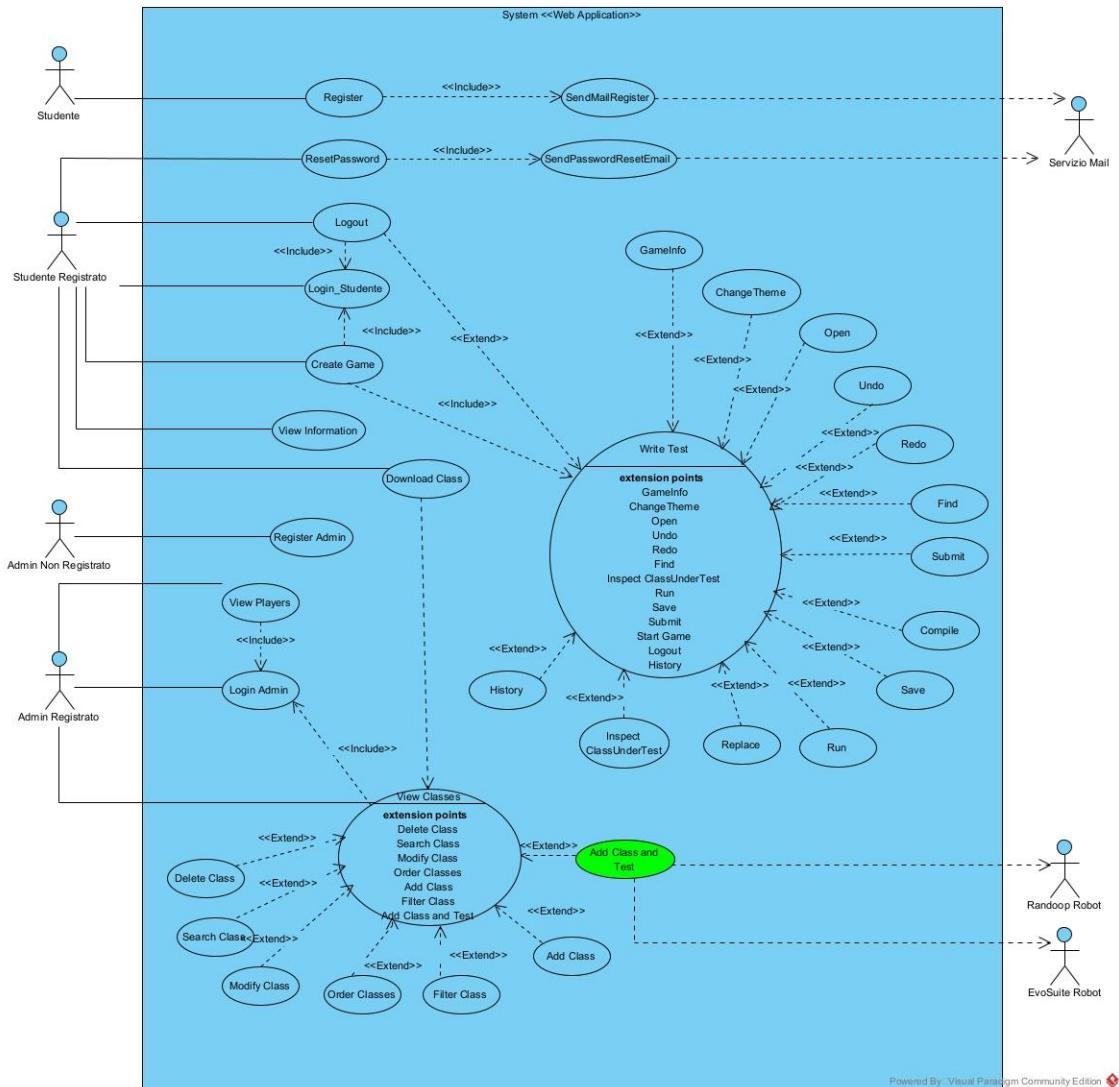


Figura 6: Diagramma dei casi d'uso completo

Qui di seguito si riporta il diagramma dei casi d'uso più dettagliato soltanto del container T1 (sempre in verde si evidenzia la modifica effettuata):

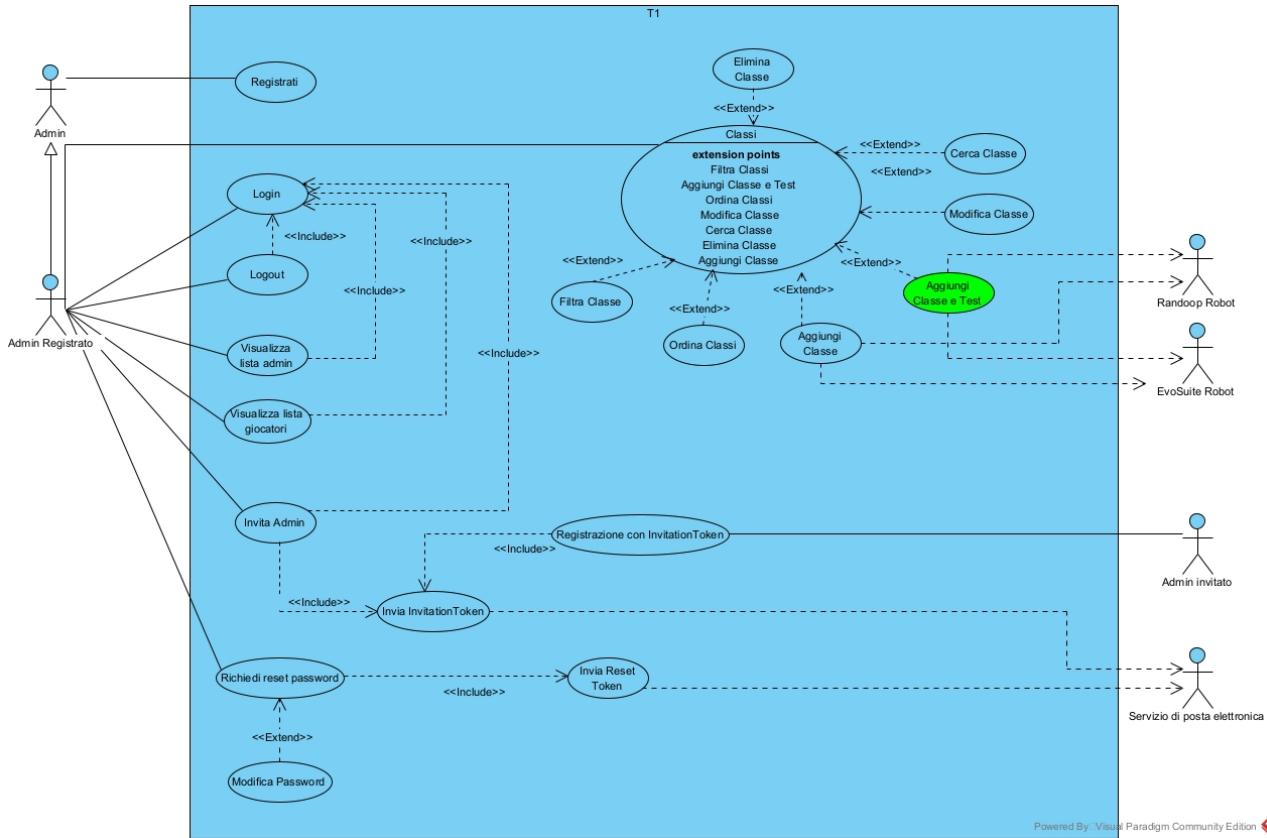


Figura 7: Diagramma dei casi d'uso dettagliato del container T1

2.2. Diagramma di sequenza

Questa sezione presenta il diagramma di sequenza relativo alla funzionalità "Add Class and Test". Il diagramma si è dimostrato particolarmente utile per analizzare il codice esistente, identificare i componenti su cui intervenire e comprendere le modalità di interazione tra di essi.

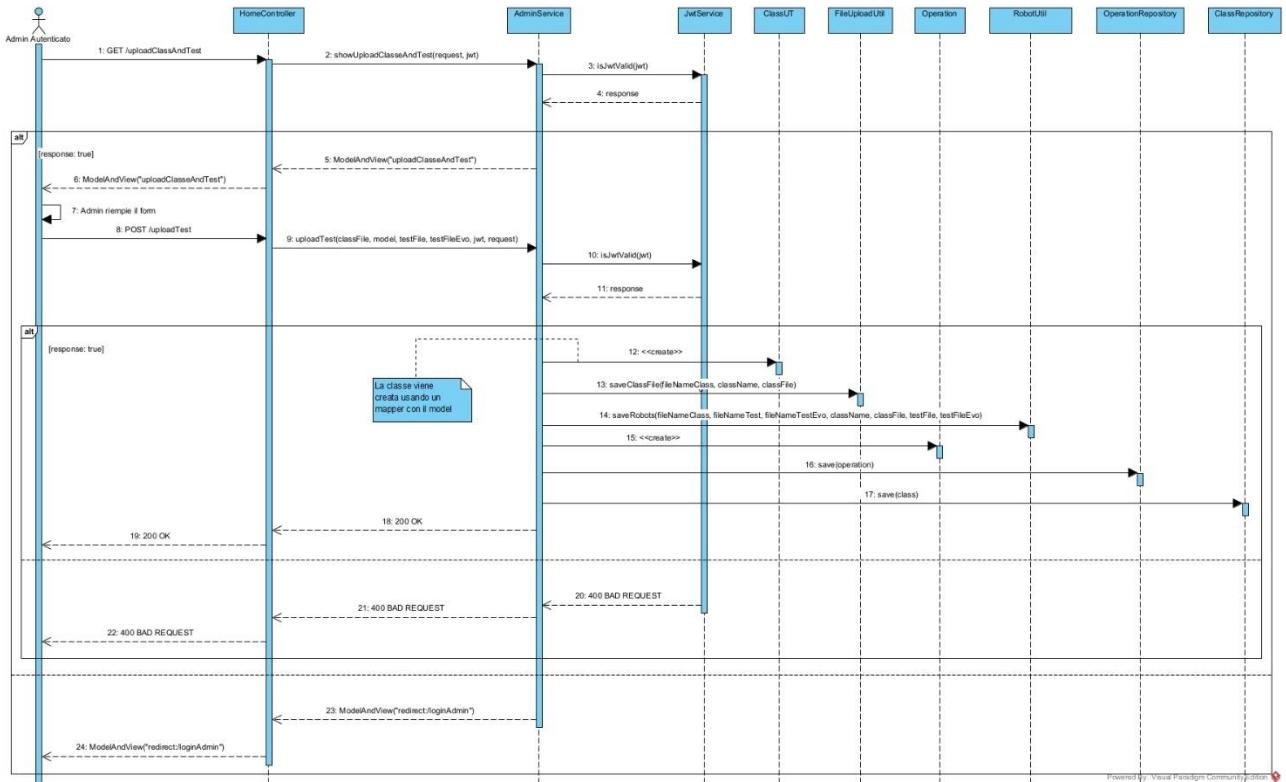


Figura 8: Sequence Diagram iniziale della funzionalità “Add Class and Test”

3. Analisi d'impatto

La web application presenta un problema significativo legato alla sua natura rigida e troppo specifica. Attualmente è progettata per funzionare esclusivamente con due robot, Randoop ed EvoSuite, limitando la possibilità di gestire e caricare test generati da altri robot. Questa restrizione riduce notevolmente la flessibilità e la scalabilità del sistema, risultando incompatibile con i principi di un'architettura a microservizi.

Considerando che i robot costituiscono il nucleo centrale dell'applicazione, è necessario implementare una gestione più tipizzata e generalizzata per semplificare l'integrazione di nuovi robot in futuro. Tuttavia, questa modifica avrà un impatto significativo su tutti i container del sistema.

A causa degli alti costi di un intervento complessivo, è stato deciso di focalizzarsi sul container T1, apportando modifiche mirate per preparare il sistema alla futura gestione tipizzata dei robot. Questo approccio consente di preservare le logiche attuali ed evitare interventi invasivi che potrebbero compromettere l'operatività del sistema.

Un'ulteriore criticità riguarda l'organizzazione del file system. Al momento, i test vengono salvati su due volumi distinti, uno per ciascun robot, mentre i file delle classi sono distribuiti su tre posizioni differenti:

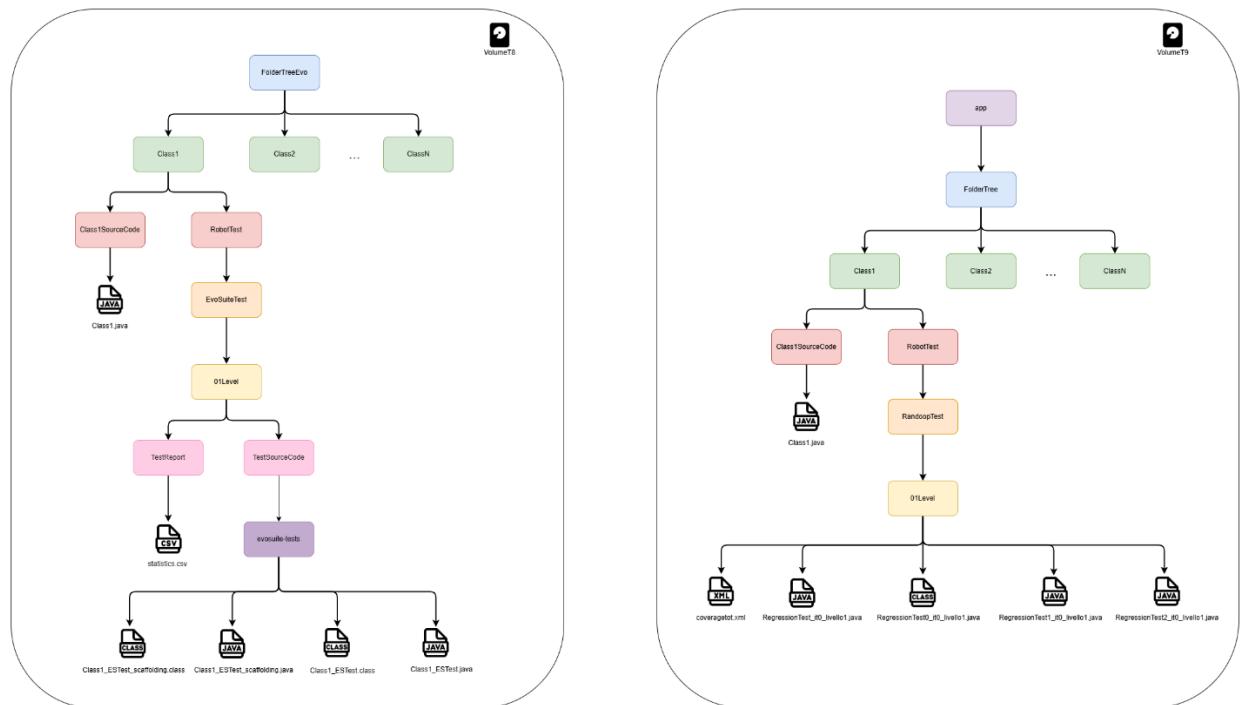


Figura 9: File system presente in VolumeT8 (sinistra) e VolumeT9 (destra)

Questa struttura introduce ridondanza, aumentando la complessità e l'inefficienza nella gestione e nel recupero dei dati. Proprio per questa gestione in alcune situazioni come la compilazione dei test, questi ultimi vengono passati nel body delle richieste HTTP, appesantendo il traffico.

Nel complesso, queste problematiche evidenziano limiti strutturali che compromettono l'efficienza, la scalabilità e la manutenibilità della web application nell'attuale architettura a microservizi.

4. Progettazione della soluzione per realizzare i requisiti richiesti

Per la gestione tipizzata dei robot, il sistema prevede un file di configurazione che permette all'amministratore di specificare i robot disponibili a livello globale. La prima modifica sarà nella pagina di "uploadClasseAndTest", che dovrà consentire la selezione dei test disponibili in base alla configurazione definita dall'amministratore. Di seguito è riportato il mockup della pagina:

The mockup shows a form titled "Test upload". It includes fields for "Class name", "Date" (with a date picker), "Difficulty" (set to "Beginner"), "Description", and three categories ("Category 1", "Category 2", "Category 3"). There are two file upload fields: one for "Upload your class (.java)" and another for "Upload your test (.zip)". A dropdown menu for "Robot" is present, with "Select a robot" as the current selection. At the bottom, there are "Upload class and tests" and "Go Back" buttons.

Figura 10: Mockup della pagina “uploadClasseAndTest”

Per quanto riguarda la distribuzione dei dati, si prevede di implementare un volume condiviso e accessibile a tutti i container. Questo volume sarà organizzato secondo una struttura gerarchica ben definita e conterrà inizialmente due tipologie di file: le classi e i test.

Anche se il volume è condiviso tra tutti i container, la sua gestione sarà centralizzata e affidata esclusivamente al container T1, che, almeno in questa fase, sarà l'unico autorizzato a scrivere nel file system. Gli altri container avranno accesso al volume in modalità di sola lettura e non saranno a conoscenza della sua struttura interna.

L'interazione avverrà tramite una nuova API fornita dal container T1, che permetterà agli altri container di accedere alle risorse del file system in maniera controllata.

L'obiettivo principale è evitare di fornire direttamente il file al container richiedente. Al contrario, T1 restituirà il percorso del file, consentendo al container di leggerlo autonomamente o di copiarlo temporaneamente nel proprio file system per le elaborazioni necessarie.

Di seguito è riportata la struttura del file system prevista:

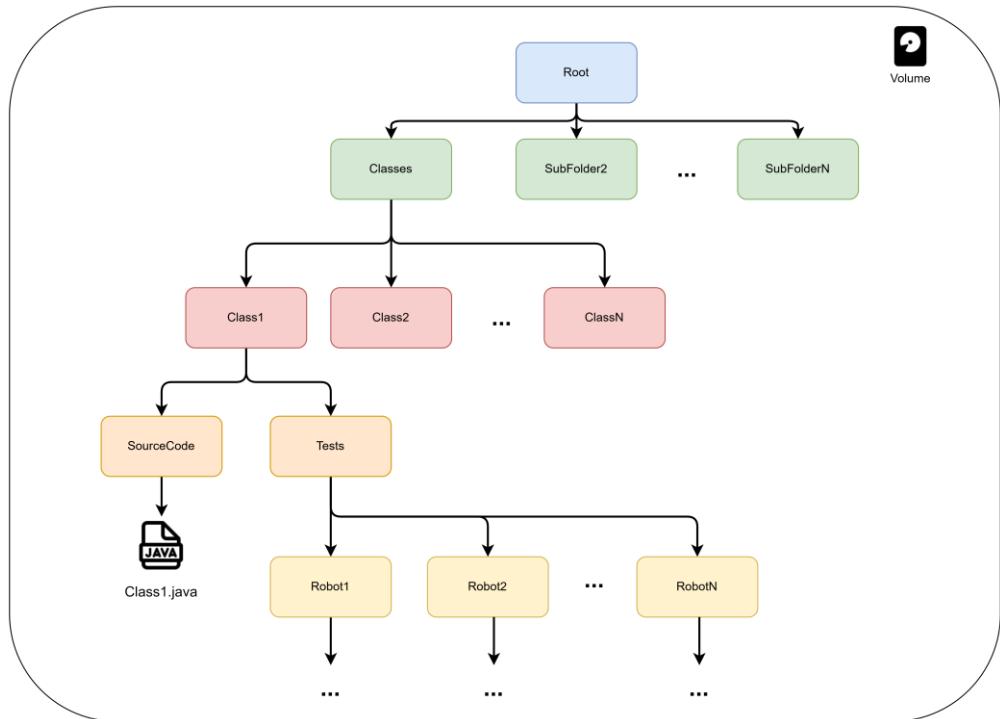


Figura 11: Nuovo file system presente nel volume condiviso

I file che il nuovo file system gestirà saranno:

- File delle classi: file con estensione .java.
- File dei test: file compressi con estensione .zip.
-

In questa prima fase, il nostro compito si limiterà alla gestione e allo scompattamento dei file, senza alcuna elaborazione del loro contenuto. In futuro, sarà necessario definire uno standard per i file relativi a ciascun robot.

L'intervento previsto sarà poco invasivo, in quanto introdurremo nuovi componenti nel container T1, mantenendo intatte le logiche legacy già in uso.

Per supportare lo sviluppo dell'API, modificheremo la struttura della collection **ClassUT**, aggiungendo un nuovo campo: una lista di **Robot**, ognuno dei quali sarà caratterizzato da un nome e dal percorso in cui risiede.

Le nuove API esporranno una serie di endpoint per le operazioni CRUD. Tuttavia, a differenza delle tradizionali API CRUD che operano esclusivamente su un database, queste opereranno sia sul database che sul file system condiviso.

A seguito di queste modifiche, verrà pianificata un'operazione di pulizia del codice, accompagnata da una riorganizzazione dei package, con l'obiettivo di ottenere una struttura più lineare e comprensibile, facilitando così la futura manutenzione e scalabilità del sistema.

Anche se come già anticipato il carico di modifiche è elevato e ci concentreremo solo su T1, di seguito riportiamo il flusso operativo che ci aspettiamo una volta che tutti i container si adatteranno alla gestione tipizzata e centralizzata dei dati:

Il flusso operativo che abbiamo proposto stabilisce che ogni container che desidera interagire con il file system condiviso debba necessariamente passare attraverso T1. Per esempio, nel caso in cui T7 necessiti di un file per la compilazione, il processo prevede che T7 utilizzi le nuove API sviluppate per comunicare con T1 e ottenere il percorso del file richiesto.

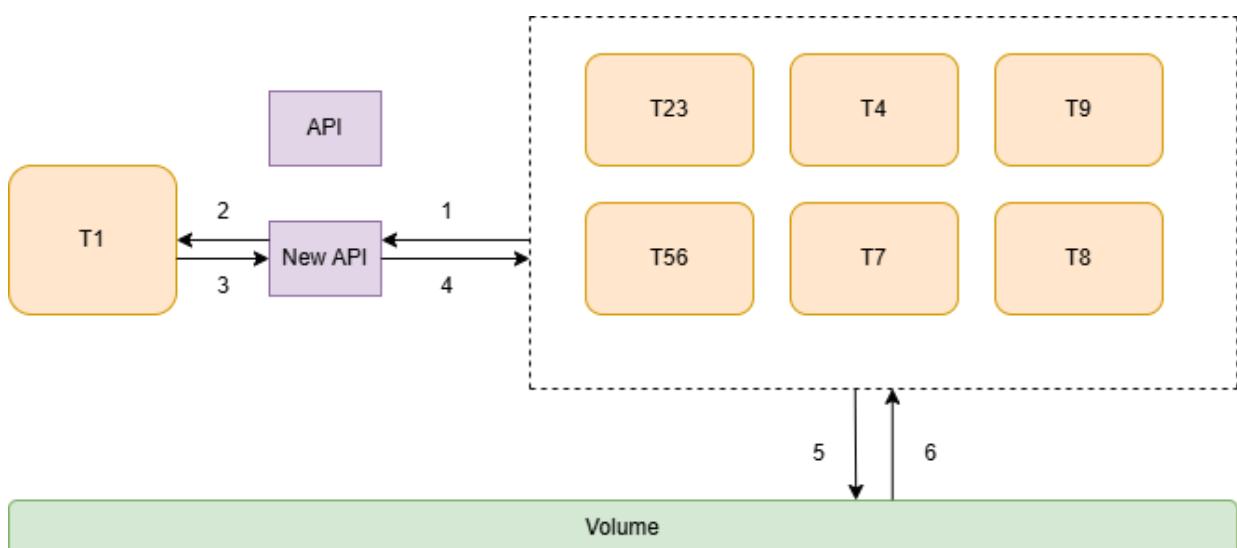


Figura 12: Flusso operativo previsto lettura

Una volta ottenuto il percorso, T7 può accedere autonomamente al file, ma solo dopo aver verificato che la risorsa sia stata bloccata per la lettura, impedendo ad altri di interagire con essa. T7 potrà quindi leggere il file direttamente o creare una copia temporanea all'interno del proprio progetto per la compilazione.

Al termine dell'operazione, T7 deve notificare T1 per sbloccare il file, rendendolo nuovamente accessibile agli altri container in modo sicuro. Nel caso in cui T7 si dimentichi di notificare T1, verrà aggiunto un time-out per garantire la continuità dell'applicazione.

In uno scenario diverso, quando un container deve eseguire operazioni di delete, add o update, deve semplicemente interagire con T1 tramite le nuove API, fornendo il path relativo all'operazione che intende eseguire. In questo caso, non sarà necessario sincronizzarsi con T1, poiché sarà T1 stesso a gestire internamente l'operazione. Di seguito viene descritto il flusso operativo per tale scenario:

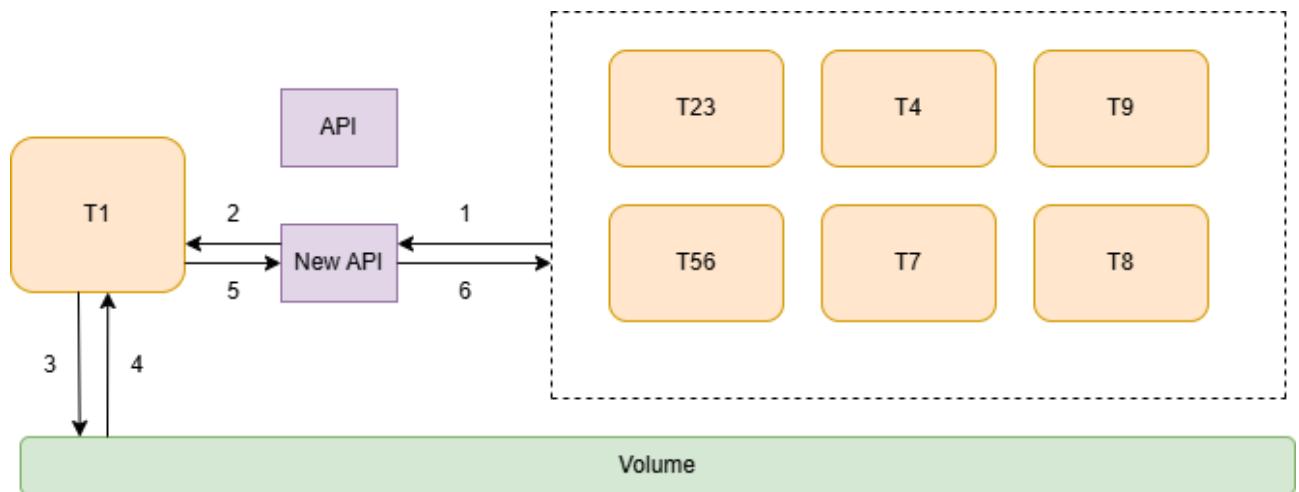


Figura 13: Flusso operativo previsto per add, update e delete

5. Implementazione

5.1. Modifiche ai file di installazione e configurazione

Per la creazione del volume condiviso, è stato necessario modificare i file di installazione (*installer.bat* per Windows, *installermac.sh* per Mac e *installer-linux-server.sh* per Linux) per includere la gestione del nuovo volume:

```
# Creazione del volume Docker 'Volume'
docker volume create Volume || echo "Errore nella creazione del volume"
```

Inoltre, si è resa necessaria la modifica del file *docker-compose.yml* di ogni container per abilitare l'uso del volume condiviso. Di seguito viene riportato, come esempio, il file *docker-compose.yml* del container T1. In questo caso, oltre a dichiarare il volume condiviso **Volume**, è stata dichiarata anche la cartella **RobotConfig**, una cartella bind mount, cioè una cartella dell'host mappata nel container. Quest'ultima è utilizzata per contenere il file di configurazione *robots.txt*, che include la lista di tutti i robot presenti.

```
version: '3.12.12'

services:
  controller:
    build: .
    restart: always
    expose:
      - 8080
    # ports:
    #   - 8080:8080
    depends_on:
      - mongo_db
    volumes:
      - ./RobotConfig:/RobotConfig
      - VolumeT9:/VolumeT9
      - VolumeT8:/VolumeT8
      - Volume:/Volume
    networks:
      - global-network

  mongo_db:
    image: "mongo:6.0.6"
    restart: always
    ports:
      - 27017:27017
    volumes:
      - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
    networks:
      - global-network
networks:
  global-network:
    external: true

volumes:
  VolumeT9:
    external: true
```

```
VolumeT8:  
  external: true  
Volume:  
  external: true
```

5.2. Modifiche ai file e ai package

Data la creazione del nuovo FileSystem sono state apportate modifiche al file `\T1-G11\applicazione\manvsclass\src\main\resources\`:

- `application.properties`, modificato per aggiungere proprietà globali che ne rappresentassero la struttura:
 - `filesystem.rootPath = /Volume/Root/`
 - `filesystem.classesPath = ${filesystem.rootPath}Classes/`
 - `filesystem.sourceFolder = SourceCode/`
 - `filesystem.testsFolder = Tests/`
 - `config.pathRobot = /RobotConfig/robots.txt`
- `\templates\uploadClasseAndTest.html`, è stato rimosso il codice JavaScript e spostato in `\static\t1\js\addClassAndTest.js`;
- `\static\t1\js\class.js`, è stato modificato per aggiungere i tag dei robot nella pagina delle classi.

Sono state apportate delle modifiche a come era organizzato il codice in `\T1-G11\applicazione\manvsclass\src\main\java\com\groom\manvsclass`.

Adesso i package sono:

- configuration;
- controller;
- model;
- repository;
- responses;
- service;
- util.

In particolare, sono state effettuate delle modifiche ai package: model, responses, service e controller.

5.2.1. Model

Al package model è stata aggiunta la classe `Robot.java`. Questo nuovo model è stato introdotto in quanto era necessaria una rappresentazione dei vari tipi di Robot.

La nuova classe è stata usata successivamente per aggiungere un attributo alla classe e due metodi `ClassUT.java`. Il nuovo attributo è una lista di Robot.

5.2.2. Responses

Il package responses contiene varie classi di risposta utilizzate dall'applicazione per inviare dati diversi al Client in base alla richiesta effettuata. In particolare, le classi implementate sono:

- *Response.java*, classe “base” che contiene come unica variabile un generico messaggio;
- *FileResponse.java*, classe che estende Response e viene usata per l'invio di dati in seguito all'inserimento di una ClasseUT;
- *ApiResponse.java*, classe che estende Response e viene usata per l'invio di dati in seguito a richieste API;
- *FileUploadResponse.java*, classe già presente precedentemente nel sistema e preservata per evitare di creare problemi con il codice legacy che già la usa.

5.2.3. Service

Nel package service sono stati introdotti tre nuovi service:

- *ClassService.java*, che fa da “gestore” delle logiche legate alle classi (quali inserimento nel database o nel FileSystem);
- *FileSystemService.java*, che si occupa solo delle logiche del file system e viene infatti anche usato da ClassService.
- *ApiService.java*, che contiene la logica di gestione delle nuove API.

I metodi creati in ClassService sono:

- **saveAll()**, che si occupa di effettuare il salvataggio sia di classi che di test inserite in front-end;

```
public ResponseEntity<FileResponse> saveAll(MultipartFile classFile,
String modelJSON, Map<String, MultipartFile> tests, String jwt,
HttpServletRequest request)
```

- **deleteAll()**, che si occupa di eliminare un'intera classe dal sistema.

```
public ResponseEntity<String> deleteAll(String className, String jwt)
```

Per ora si occupa di salvataggio e cancellazione di classi e test dall'applicazione, ma in futuro dovrebbe occuparsi di tutto ciò che riguarda la gestione delle classi. C'era necessità di questa nuova classe perché AdminService da solo si occupava di molte logiche non necessariamente connesse tra loro e diventava difficile andare alla ricerca delle porzioni di codice da controllare. Di per sé la struttura dell'AdminService non è cambiata, a patto di qualche aggiunta di *if-statements* per rendere il codice robusto all'inserimento di solo Randoop, Evosuite o nessuno dei due dato che in precedenza la presenza di entrambi i test era necessaria. La creazione e cancellazione di classi

dell'AdminService sono utilizzate ancora dallo stesso ClassService per mantenere la compatibilità con il resto dell'applicazione, dato che, se il codice del T1 venisse radicalmente cambiato, potrebbero esserci ripercussioni fatali su tutto il resto del sistema. ClassService, inoltre, si appoggia per l'inserimento delle classi nel nuovo FileSystem a FileSystemService.

I metodi creati in FileSystemService sono:

- **saveClass()**, che si occupa di salvare il file della classe nel FileSystem, creando anche le cartelle necessarie;

```
public Path saveClass(String className, MultipartFile classFile)
```

- **saveTest()**, che si occupa di salvare i file di test nel FileSystem, creando anche le cartelle necessarie;

```
public Path saveTest(String className, String robotName, MultipartFile testFile)
```

- **deleteAll()**, che si occupa di eliminare un'intera classe (compresi i test) dal sistema;

```
public Path deleteAll(String className)
```

- **deleteClass()**, che si occupa di eliminare solo la cartella del source-code di una classe;

```
public Path deleteClass(String className)
```

- **deleteTest()**, che si occupa di eliminare uno specifico test di una data classe;

```
public Path deleteTest(String className, String robotName)
```

- **rawCreateFolder()**, che si occupa di creare una cartella e tutte le cartelle precedenti specificate dal percorso;

```
private Path rawCreateFolder(Path path)
```

- **createFolder()**, che potenzia il metodo **rawCreateFolder()** aggiungendoci meccanismi di sincronizzazione e rollback. Ne esistono due versioni: una prende in ingresso un oggetto String, un'altra un oggetto Path;

```
public Path createFolder(String path)
public Path createFolder(Path path)
```

- **rawSaveFile()**, che si occupa di salvare un file in un percorso specificato del FileSystem;

```
private Path rawSaveFile(MultipartFile file, String path)
```

- **saveFile()**, che potenzia il metodo **rawSaveFile()** con meccanismi di sincronizzazione e rollback. Ne esistono due versioni: una prende in ingresso un oggetto String, un'altra un oggetto Path;

```
public Path saveFile(MultipartFile file, String path)
public Path saveFile(MultipartFile file, Path path)
```

- **unzip()**, che si occupa di effettuare l'unzip di un file “.zip” all'interno di un percorso specificato;

```
public Path unzip(MultipartFile zipFile, Path unzipPath)
```

- **deleteDirectory()**, che si occupa di eliminare un file o una cartella (compreso il contenuto) di un percorso specificato. Viene usata anche per eseguire il rollback in seguito ad una creazione non andata a buon fine;

```
public Path deleteDirectory(Path directory)
```

- **existsPath()**, che si occupa di verificare l'esistenza di un determinato percorso;

```
public boolean existsPath(Path path)
```

- **validatePath()**, che si occupa di verificare che il Path in ingresso sia un percorso valido;

```
public boolean validatePath(Path path)
```

Il metodo **unzip()** era già esistente all'interno di *RobotUtil.java*, ma presentava dei problemi. Prevedeva che dapprima venisse caricato il file “.zip” nel sistema, poi rintracciato in base al percorso, e poi si poteva fare l'unzip. Invece il nuovo metodo utilizza il MultipartFile consegnato nel body della richiesta HTTP, ed esegue l'unzip direttamente nel percorso specificato.

Inoltre, si è passati dall'utilizzo della classe *java.io.File*, ad un approccio migliorato con le classi *java.nio.file.Files*, *java.nio.file.Path* e *java.nio.file.Paths*.

Per ora FileSystemService è utilizzato solo da ClassService e ApiService, ma in futuro potrebbero servire anche ad altri componenti nel caso di espansione del FileSystem.

Tutti i metodi che “**raw**” sono metodi “**di base**”, cioè che non prevedono rollback e sincronizzazione. Sono stati usati per dividere la logica del comportamento di base dalle altre.

I metodi di appoggio, sempre interni a FileSystemService, usati per la gestione della concorrenza e il rollback sono:

- **deleteRollback()**, esegue il rollback in seguito ad una operazione di delete non andata a buon fine;

```
private void deleteRollback(Path path, byte[] file)
```

- **readLock()**, acquisisce il lock per operazioni di lettura (per esempio necessità di copiare un file);

```
private void readLock(Path path)
```

- **readUnlock()**, rilascia il lock per operazioni di lettura;

```
private void readUnlock(Path path)
```

- **writeLock()**, acquisisce il lock per operazioni di scrittura (per esempio cancellare o sovrascrivere un file);

```
private void writeLock(Path path)
```

- **writeUnlock()**, rilascia il lock per operazioni di scrittura;

```
private void writeUnlock(Path path)
```

- **wait()**, i thread che vogliono operare su un certo path si mettono in attesa di ricevere una notifica prima di continuare le loro operazioni;

```
private boolean wait(Path path)
```

- **notify()**, notifica un thread in attesa su un certo path;

```
private void notify(Path path)
```

I metodi creati in ApiService sono:

- **getClasses()**, che si occupa di restituire tutte le classi presenti nel sistema;

```
public ResponseEntity<ApiResponse> getClasses(String jwt)
```

- **getClass()**, che si occupa di restituire il path della classe all'interno del file system condiviso;

```
public ResponseEntity<ApiResponse> getClass(String className, String jwt)
```

- **getRobots()**, che si occupa di restituire la lista di robot associati alla classe specificata;

```
public ResponseEntity<ApiResponse> getRobots
```

- **getRobot()**, che si occupa di restituire il path del robot specifico associato alla classe;

```
public ResponseEntity<ApiResponse> getRobot(String className, String
robotName, String jwt)
```

- **setClass()**, che si occupa di eseguire l'UPDATE del file contenente la classe;

```
public ResponseEntity<ApiResponse> setClass(String className, MultipartFile
classFile, String jwt) throws IOException
```

- **setRobot()**, che si occupa di eseguire l'INSERT o l'UPDATE del robot e del suo file;

```
public ResponseEntity<ApiResponse> setRobot(String className, MultipartFile
robotFile, String jwt, String robotName) throws IOException
```

- **deleteClass()**, che si occupa di eliminare l'intera classe specificata con tutti i suoi test;

```
public ResponseEntity<ApiResponse> deleteClass(String className, String jwt)
throws IOException
```

- **deleteRobot()**, che si occupa di eliminare un solo robot associato alla classe.

```
public ResponseEntity<ApiResponse> deleteRobot(String className, String jwt,
String robotName) throws IOException
```

- **setFileSystem()**, che si occupa di aggiungere o aggiornare un file al path specificato se questo esiste ed è valido.

```
public ResponseEntity<ApiResponse> setFileSystem(String pathRequest,
MultipartFile file, String jwt) throws IOException
```

- **deleteFileSystem()**, che si occupa di eliminare un file o una directory al path specificato se questo esiste ed è valido.

```
public ResponseEntity<ApiResponse> deleteFileSystem(String pathRequest,
String jwt) throws IOException
```

- **lock()**, che si occupa di effettuare l'operazione di lock sul path specificato se questo esiste ed è valido.

```
public ResponseEntity<ApiResponse> lock(String pathRequest, String jwt)
throws InterruptedException
```

- **unlock()**, che si occupa di effettuare l'operazione di unlock sul path specificato se questo esiste ed è valido.

```
public ResponseEntity<ApiResponse> unlock(String pathRequest, String jwt)
throws InterruptedException
```

- **runThread()**, che si rappresenta il comportamento del Thread che viene creato per effettuare le operazioni di lock e unlock sul path.

```
private void runThread(ReentrantLock lock, Condition condition, AtomicBoolean
isReady, Path path) throws InterruptedException
```

Sono poi state fatte delle modifiche anche a *AdminService.java*, che si occupava di molte delle logiche del T1, quali upload di classi e tutto ciò che riguarda l'admin.

Le modifiche fatte all'AdminService sono:

- **getRobots()**, che si occupa di prelevare la lista di Robot supportati da un file di configurazione “*robots.txt*”;

```
public ResponseEntity<List<String>> getRobots(HttpServletRequest request,
String jwt)
```

- *if-statements* nelle routes `/delete/{name}` e `/uploadTest` per rendere il codice legacy compatibile con le nuove funzionalità.

5.2.4. Controller

Il package controller adesso contiene:

- *HomeController.java*, che si occupa delle route “interne” al T1;
- *ApiController.java*, che si occupa delle route “esterne” al T1, cioè di fornire un'interfaccia per gli altri microservizi dell'applicazione.

HomeController è pressoché identico a prima. Le modifiche riguardano il cambio dell'uso di AdminService con ClassService per alcune route, tra cui:

- Route `/delete/{name}`, che si occupa dell'eliminazione di una classe e di tutti i suoi test dal sistema;

```
return classService.deleteAll(name, jwt)
```

- Route **/uploadTest**, che si occupa del salvataggio di una classe e tutti i suoi test nel sistema.

```
return classService.saveAll(classFile, modelJSON, testFiles, jwt, request)
```

È stata aggiunta anche una nuova route:

- Route **/listofrobots**, che si occupa di fornire la lista di Robots supportati dall'applicazione.

```
public ResponseEntity<List<String>> getRobots(HttpServletRequest request,
@CookieValue(name = "jwt", required = false) String jwt)
```

ApiController ha il compito di gestire tutte le richieste per interagire con il file system condiviso. Mette a disposizione una serie di metodi per gestire le seguenti route:

- **getClasses()** che gestisce la route GET/**classes**;

```
@GetMapping("classes")
public ResponseEntity<ApiReponse> getClasses(@CookieValue(name = "jwt",
required = false) String jwt)
```

- **getClass()** che gestisce la route GET/**classes/{className}**;

```
@GetMapping("classes/{className}")
public ResponseEntity<ApiReponse> getClass(@PathVariable(value = "className")
String className, @CookieValue(name = "jwt", required = false) String jwt)
```

- **getRobots()** che gestisce la route GET/**classes/{className}/robots**;

```
@GetMapping("classes/{className}/robots")
public ResponseEntity<ApiReponse> getRobots(@PathVariable(value =
"className") String className, @CookieValue(name = "jwt", required = false)
String jwt)
```

- **getRobot()** che gestisce la route GET/**classes/{className}/{robotName}**;

```
@GetMapping("classes/{className}/{robotName}")
public ResponseEntity<ApiReponse> getRobot(@PathVariable(value = "className")
String className,@PathVariable(value = "robotName") String robotName,
@CookieValue(name = "jwt", required = false) String jwt)
```

- **setClass()** che gestisce la route POST/classes/{className};

```
@PostMapping("classes/{className}")
public ResponseEntity<ApiReponse> setClass(@PathVariable(value = "className")
String className, @RequestParam(name = "classFile", required = false)
MultipartFile classFile, @CookieValue(name = "jwt", required = false) String
jwt) throws IOException
```

- **setRobot()** che gestisce la route POST/classes/{className}/{robotName};

```
@PostMapping("classes/{className}/{robotName}")
public ResponseEntity<ApiReponse> setRobot(@PathVariable(value = "className")
String className, @PathVariable(value = "robotName") String robotName,
@RequestParam(name = "robotFile", required = false) MultipartFile robotFile,
@CookieValue(name = "jwt", required = false) String jwt) throws IOException
```

- **deleteClass()** che gestisce la route DELETE/classes/{className};

```
@DeleteMapping("classes/{className}")
public ResponseEntity<ApiReponse> deleteClass(@PathVariable(value =
"className") String className,@CookieValue(name = "jwt", required = false)
String jwt) throws IOException
```

- **deleteRobot()** che gestisce la route DELETE/classes/{className}/{robotName}.

```
@DeleteMapping("classes/{className}/{robotName}")
public ResponseEntity<ApiReponse> deleteRobot(@PathVariable(value =
"className") String className, @PathVariable(value = "robotName") String
robotName,@CookieValue(name = "jwt", required = false) String jwt) throws
IOException
```

- **setFileSystem()**, che si occupa di gestire la route POST/fileSystem.

```
@PostMapping("fileSystem")
public ResponseEntity<ApiResponse> setFileSystem(@RequestParam(name = "path")
String pathRequest, @RequestParam(name = "file") MultipartFile file,
@CookieValue(name = "jwt", required = false) String jwt) throws IOException
```

- **deleteFileSystem()**, che si occupa di gestire la route DELETE/fileSystem.

```
@DeleteMapping("fileSystem")
public ResponseEntity<ApiResponse> deleteFileSystem(@RequestParam(name =
"path") String pathRequest, @CookieValue(name = "jwt", required = false)
String jwt) throws IOException
```

- **lock()**, che si occupa di gestire la route POST/lock.

```
@PostMapping("/lock")
public ResponseEntity<ApiResponse> lock(@RequestParam(name = "path") String
pathRequest, @CookieValue(name = "jwt", required = false) String jwt) throws
InterruptedException
```

- **unlock()**, che si occupa di gestire la route POST/**unlock**.

```
@PostMapping("/unlock")
public ResponseEntity<ApiResponse> unlock(@RequestParam(name = "path") String
pathRequest, @CookieValue(name = "jwt", required = false) String jwt) throws
InterruptedException
```

5.2.5. Pagine HTML

Sono state apportate modifiche alla pagina di inserimento di classi e test. Ora è possibile inserire un numero arbitrario di test, con la restrizione che ciascun test sia associato a un solo Robot supportato. Nella pagina è stato introdotto un tasto "+" che consente di aggiungere dinamicamente un nuovo menu di inserimento per un test. Inoltre, un menu a tendina permette di selezionare il Robot corrispondente al test che si sta inserendo.

The screenshot shows a user interface for adding classes and tests. At the top, there's a field to upload a Java class (.java) with a 'Scegli il file' button and the file name 'Calcolatrice.java'. Below this, a 'Robot' dropdown menu is set to 'Randoop'. A note says 'Select the robot you wish to upload the test'. Underneath, another file upload field for tests (.zip) has a 'Scegli il file' button and the file name 'RandoopTest.zip'. The next section is also labeled 'Robot' and features a dropdown menu with 'ChatGPT' selected. This dropdown also includes options 'Select a robot', 'ChatGPT', 'Gemini', and 'EvoSuite'. At the bottom, there's a 'Upload your test (.zip):' field with a 'Scegli il file' button and the file name 'EvoSuiteTest.zip'. A green button with a '+' icon is available for adding more tests. Finally, there are two buttons at the bottom: 'Upload class and tests' (green) and 'Go Back'.

Figura 13: Pagina di Add Class and Test modificata

La lista dei robot è dinamica e viene prelevata dal file di configurazione *robots.txt*, gestito tramite JavaScript e situato all'interno della cartella *RobotConfig* nel container T1. Il menu a tendina è anch'esso dinamico e iterativo: quando un robot viene selezionato, esso viene rimosso dalla lista, garantendo che ogni test sia associato a un robot unico.

Ad essere stata leggermente modificata è stata anche la pagina che visualizza le classi. Adesso quando si espande la classe, compariranno anche dei tag che indicano i robot esistenti per quella classe.

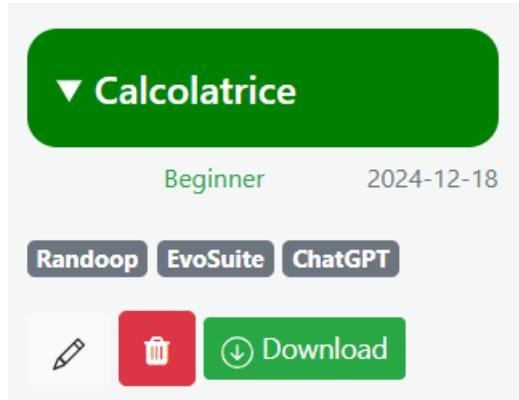


Figura 14: Aggiunta dei tag dei robot associati ad una classe

In questo caso la classe Calcolatrice contiene test per i robot: Randoop, Evosuite e ChatGPT.

5.2.6. UI Gateway

Come descritto nei capitoli precedenti, l'integrazione dei vari servizi avviene tramite l'adozione del Gateway pattern, il cui obiettivo è fornire un unico punto di accesso per tutte le richieste. Questo pattern prevede la realizzazione di due componenti principali, tra cui l'UI Gateway, che funge da unico entry point al sistema.

L'UI Gateway è implementato utilizzando **Nginx**, configurato come reverse proxy. Un *reverse proxy* è un server intermedio che si posiziona tra i client e i server backend: il suo compito è ricevere le richieste dai client, inoltrarle ai server appropriati e restituire le risposte ai client.

La configurazione di Nginx viene gestita tramite un file dedicato, dove sono definite tutte le rotte. A seguito del nostro intervento, abbiamo aggiunto al file di configurazione alcune nuove rotte nel blocco dedicato a T1. Queste rotte inoltrano le richieste al backend configurato tramite la direttiva proxy_pass `http://manvsclass-controller-1:8080`. Le rotte aggiunte sono:

- `/classes`: per la gestione della nuova API;
- `/listofrobots`: per la gestione del file di configurazione dei robot;
- `/fileSystem`: per la gestione delle API generiche per il fileSystem;
- `/lock`: per la gestione dell'operazione di lock su un path;
- `/unlock`: per la gestione dell'operazione di unlock su un path.

5.3. Aggiunta di API

Come già anticipato la nuova API messa a disposizione da T1 permette agli altri container di interagire con il file system condiviso. L'entità centrale da gestire sarà la "classe", che per coerenza nel design verrà rappresentata nella nostra nomenclatura come "classes". Di seguito una rappresentazione delle route messe a disposizione dalla nuova interfaccia:

<i>/classes</i>			
	Body	Descrizione	Risposta
GET		Ritorna l'elenco delle classi presenti nel sistema.	200: Classes found 401: Error, token not valid 404: Error, classes not found
<i>/classes/{className}</i>			
	Body	Descrizione	Risposta
GET		Ritorna il percorso all'interno del file system dove si trova la classe specificata.	200: Class found 401: Error, token not valid 404: Error, class not found
POST	“classFile”: file.java	Aggiorna il file.java per una classe.	200: Class setted 401: Error, token not valid 404: Error, class not found
DELETE		Elimina una classe con tutti i robot associati.	200: Class deleted 401: Error, token not valid 404: Error, class not found
<i>/classes/{className}/{robotName}</i>			
	Body	Descrizione	Risposta
GET		Ritorna il percorso all'interno del file system dove si trova lo specifico robot associato alla classe.	200: Robot found 401: Error, token not valid 404: Error, robot not found 404: Error, class not found
POST	“robotFile”: file.zip	Aggiunge/Aggiorna i file per uno specifico robot, solo se questo è presente nel file di configurazione dei robot (robots.txt).	200: Robot setted 401: Error, token not valid 400: Error, robot not available 404: Error, class not found
DELETE		Ritorna il percorso all'interno del file system dove si trova lo specifico robot associato alla classe.	200: Robot found 401: Error, token not valid 404: Error, robot not found 404: Error, class not found
<i>/classes/{className}/robots</i>			
	Body	Descrizione	Risposta
GET		Ritorna la lista di robot associata alla classe specificata.	200: Robots found 401: Error, token not valid 404: Error, robots not found 404: Error, class not found
<i>/fileSystem</i>			
	Body	Descrizione	Risposta
POST	“path”: “/....” “file”: file	Aggiunge o Aggiorna un file al path specificato se questo è valido ed esiste.	200: Path element setted 401: Error, token not valid 404: Error, path not valid 404: Error, file not present
DELETE	“path”: “/....”	Elimina un file o una directory al path specificato se questo è valido ed esiste.	200: Path element deleted 401: Error, token not valid 404: Error, path not valid

<i>/lock</i>			
	Body	Descrizione	Risposta
POST	“path”: “/....”	Effettua il lock per il path specificato se questo esiste ed è valido.	200: Path locked 401: Error, token not valid
<i>/unlock</i>			
	Body	Descrizione	Risposta
POST	“path”: “/....”	Effettua l’operazione di unlock per il path specificato se questo esiste ed è valido.	200: Path unlocked 401: Error, token not valid

6. Architettura finale

Di seguito è presentato il diagramma di deployment finale, che evidenzia i cambiamenti apportati all'architettura dove in giallo si evidenziano i moduli aggiunti e in verde quelli modificati:

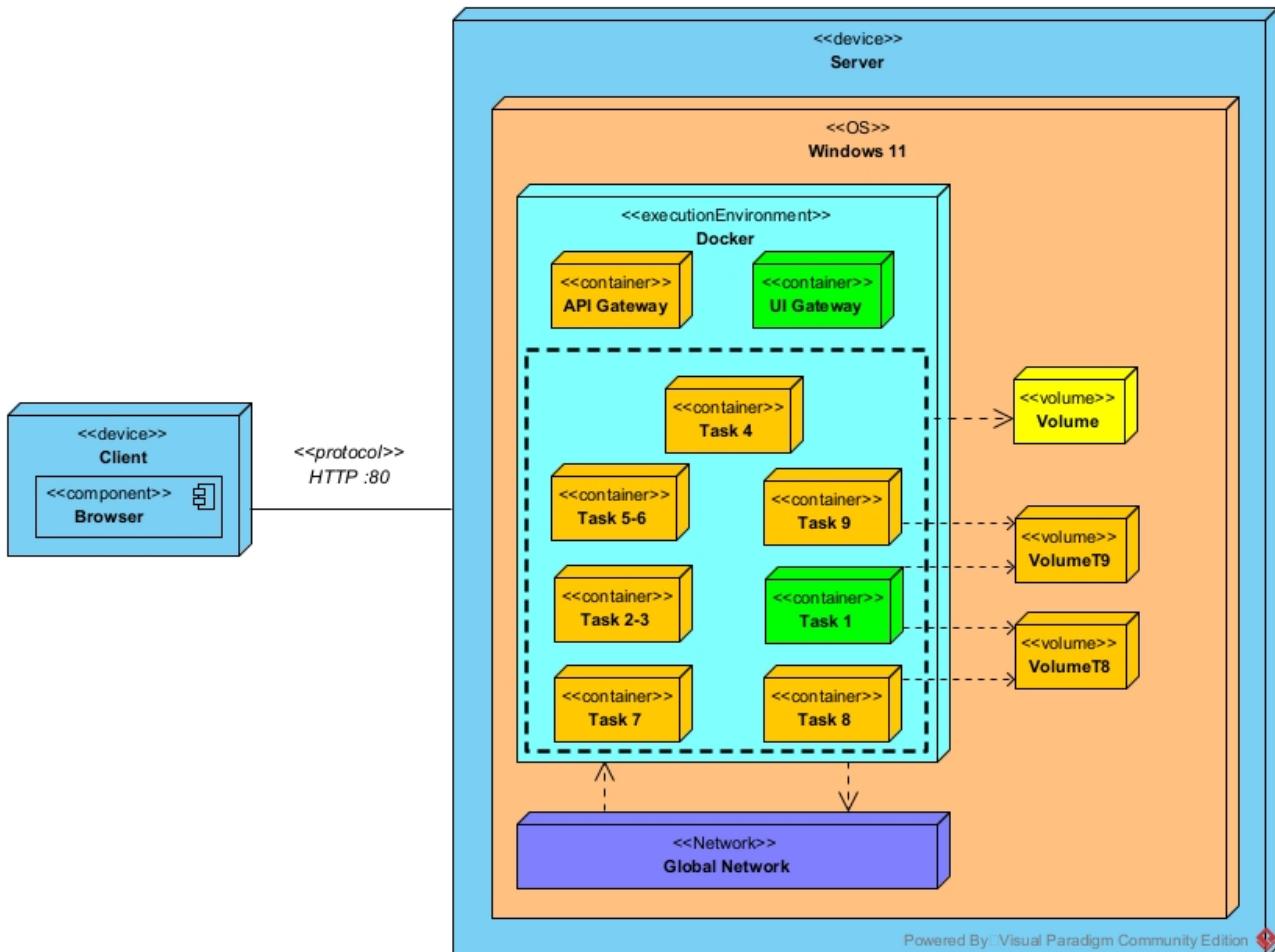


Figura 15: Deployment Diagram finale

Come si può notare, ora tutti e nove i container condividono il volume **Volume**. La linea tratteggiata non rappresenta nessuna divisione interna all'ambiente di esecuzione, ma è semplicemente un modo grafico per illustrare che i container condividono lo stesso volume.

Si riportano anche le collection di MongoDB aggiornate:

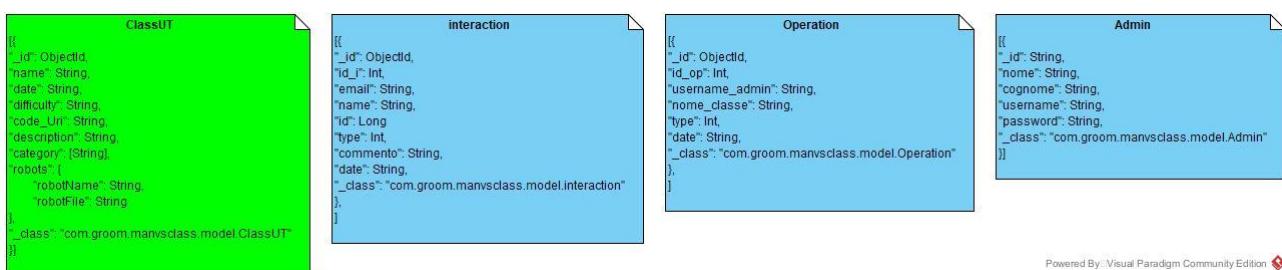


Figura 16: Collection di MongoDB aggiornate

Di seguito viene riportato il nuovo Sequence Diagram della funzionalità di "Add Class And Test" a seguito delle recenti modifiche:

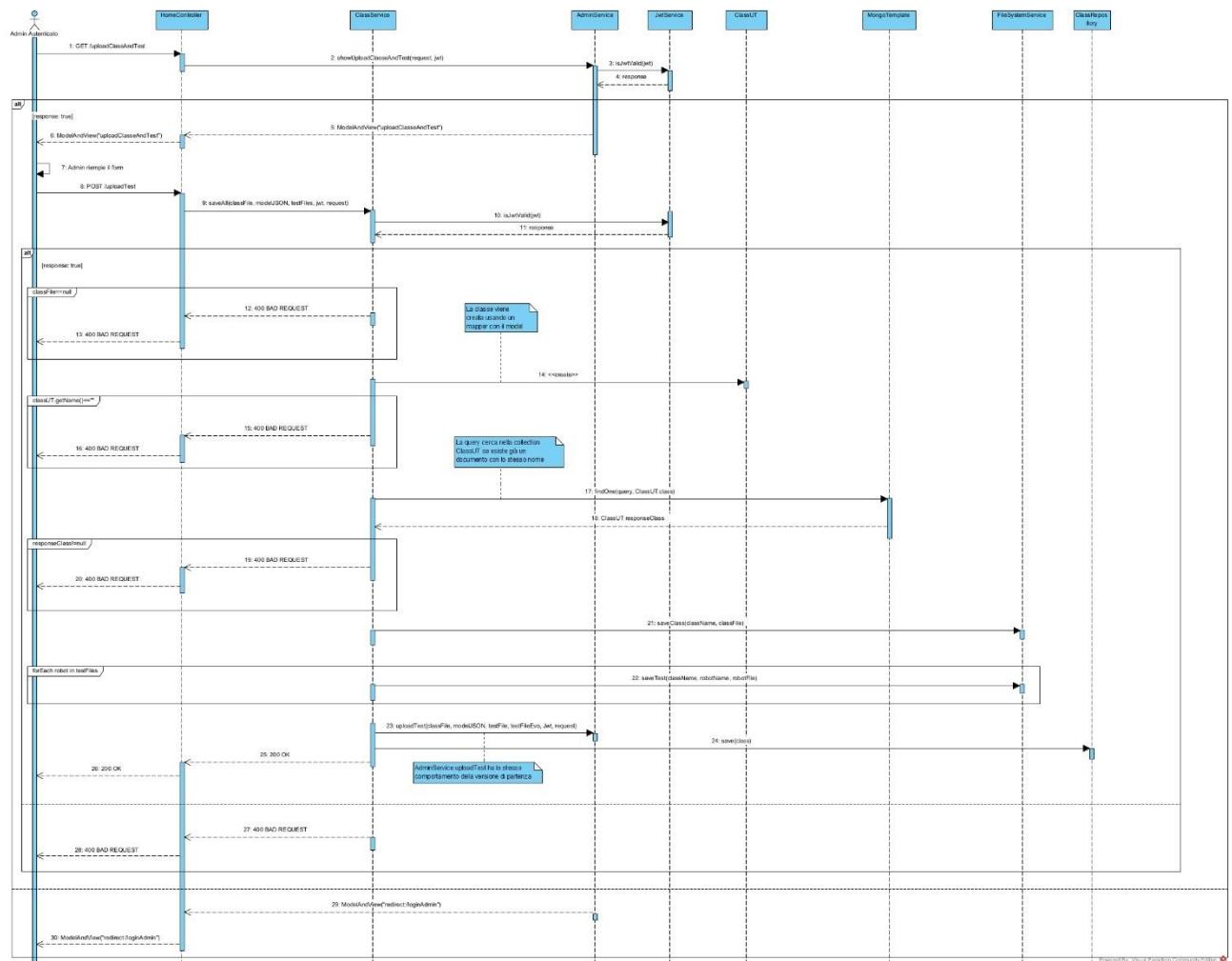


Figura 17: Sequence Diagram finale della funzionalità Add Class and Test

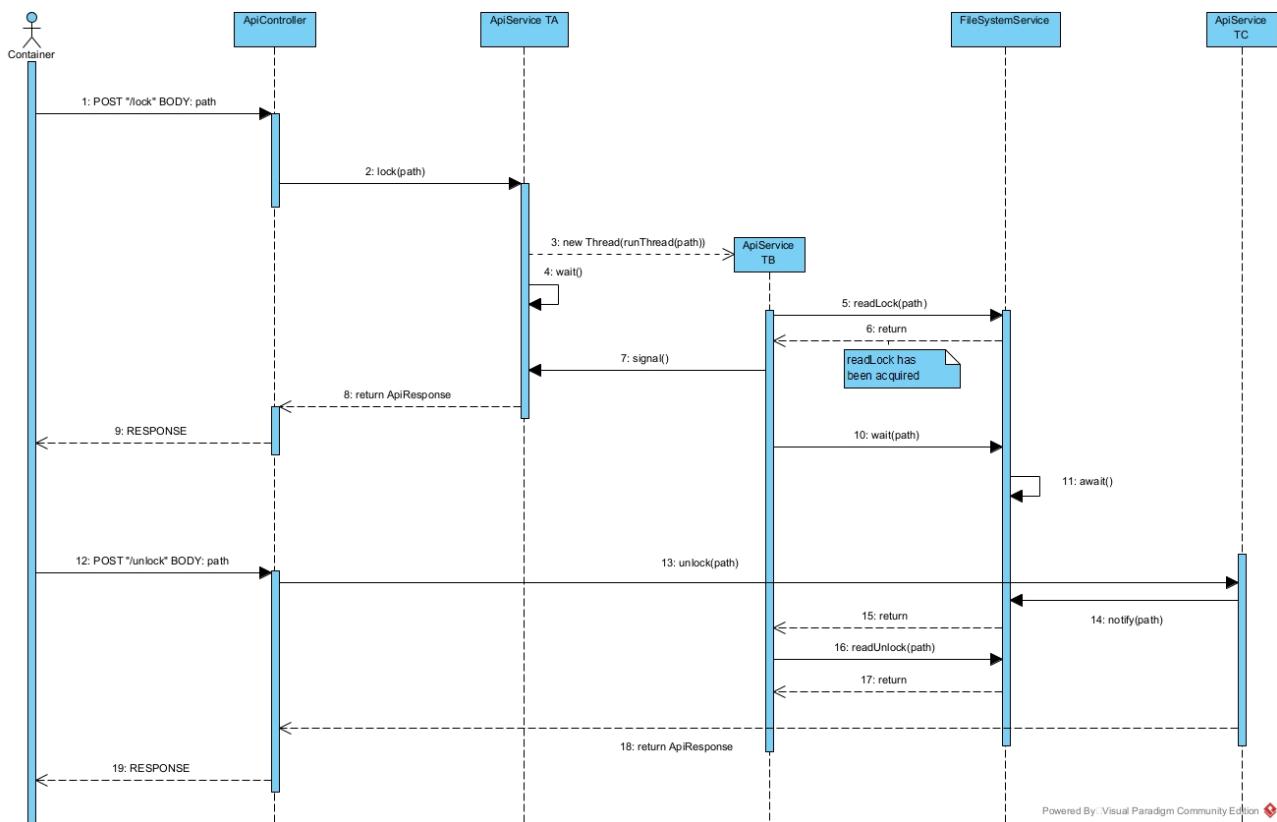


Figura 18: Sequence Diagram della concorrenza della lettura del file system

Il sequence diagram nella figura 18 illustra come funziona la lettura di un file dal file system. Un container qualsiasi invia una richiesta di POST con il path a cui vuole ottenere i “diritti di lettura” nel body. Questo è necessario per evitare che, durante la lettura, i file vengano modificati o eliminati, causando errori.

Una volta ricevuta la richiesta, l’**ApiController** avvierà un thread (**TB**) che si occuperà del blocco e dello sblocco del path, usufruendo di specifici metodi di **FileSystemService**. Una volta bloccato il path per la lettura, verrà restituita una risposta di **OK** al container richiedente.

Quando il richiedente ha terminato le sue operazioni di lettura, è consigliato inviare una nuova POST di sblocco per liberare il path. L’ **ApiService** utilizzerà nuovamente le funzioni di **FileSystemService** per svegliare il thread **TB** e consentirgli di sbloccare il path. Successivamente, viene inviata una risposta di "azione eseguita correttamente" al container.

Lo sblocco da parte di **TB**, tuttavia, avviene anche in assenza della seconda POST, dopo un intervallo temporale predefinito (da noi impostato a 30 secondi) per evitare che, in caso di dimenticanze, la risorsa rimanga bloccata indefinitivamente. Inoltre, il blocco non viene applicato se il path passato nella POST è invalido.

A conclusione si riportano il Component Diagram di T1 e il Package Class finale con tutte le modifiche eseguite (dove sempre in giallo si evidenziano i moduli aggiuntati e in verde i modificati):

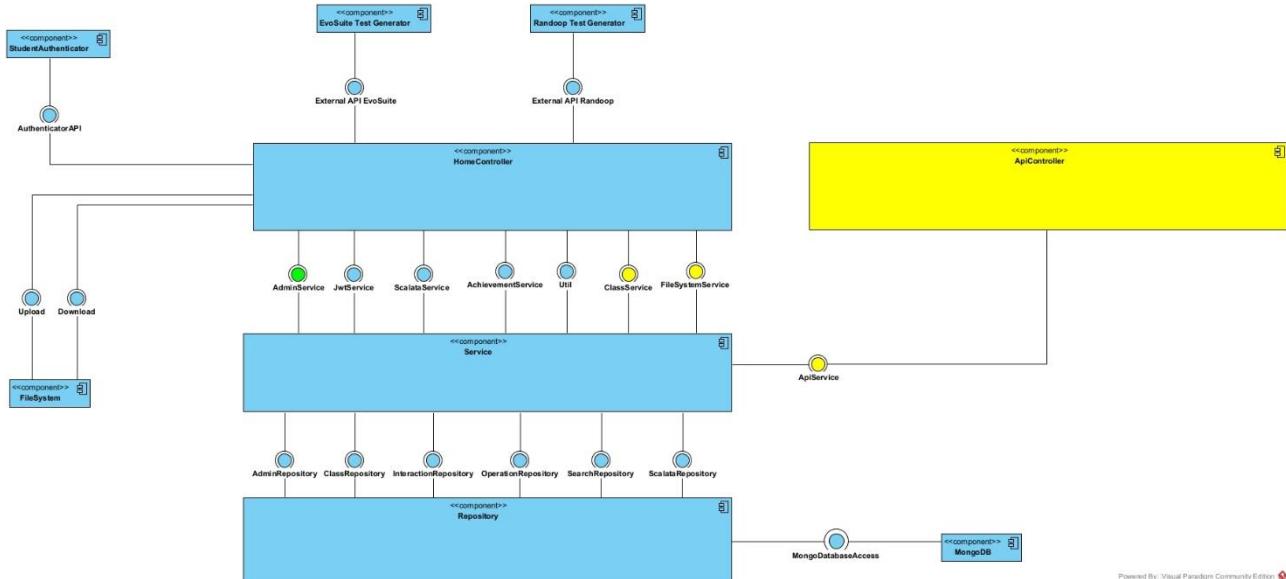


Figura 19: Component Diagram di T1

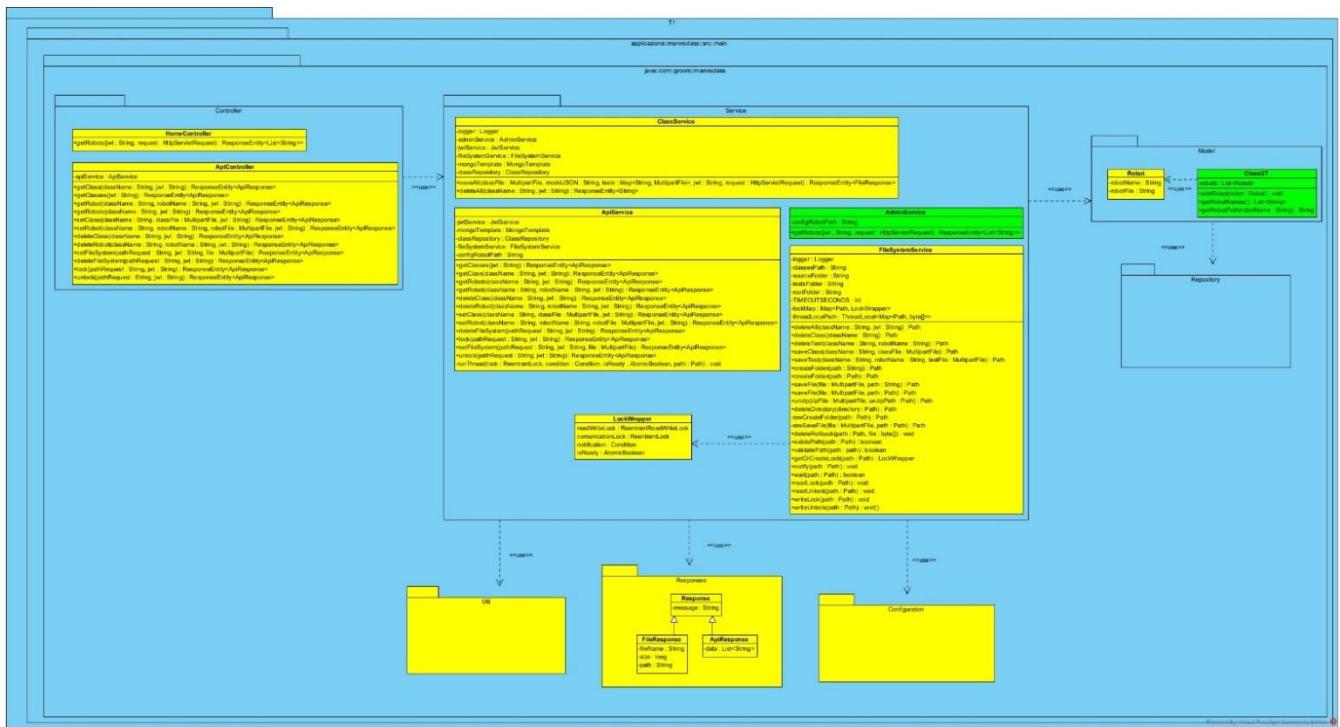


Figura 20: Package Class completo

7. Testing

7.1. Testing GUI

Per verificare il corretto funzionamento dell'interfaccia grafica, è stata svolta una sessione di testing manuale, documentata attraverso una test suite strutturata. Ogni caso di test è descritto in modo dettagliato prendendo in considerazione i seguenti elementi chiave:

- **Descrizione:** Una breve spiegazione del caso di test, che definisce chiaramente l'obiettivo e il contesto della verifica.
- **Precondizione:** Le condizioni iniziali del sistema, inclusi eventuali requisiti o configurazioni specifiche necessarie per avviare il test.
- **Input:** La sequenza esatta di azioni o comandi eseguiti dall'utente sull'interfaccia grafica. Questo può includere l'inserimento di dati, l'interazione con pulsanti, menu o altri elementi UI.
- **Output atteso:** Il risultato previsto in risposta agli input forniti, sia in termini di comportamento visivo dell'interfaccia sia di output funzionale del sistema.
- **Post condizione attesa:** Lo stato finale atteso del sistema dopo l'esecuzione del test, comprensivo di eventuali modifiche o effetti persistenti sull'applicazione.
- **Output ottenuto:** Il risultato effettivo osservato durante l'esecuzione del test, confrontato direttamente con l'output atteso. Viene riportato in modo chiaro e oggettivo.
- **Esito:** La valutazione finale del caso di test, che determina se l'esito è positivo (PASS) o negativo (FAIL). L'esito è stabilito in base alla corrispondenza tra output atteso e output ottenuto, tenendo conto di eventuali discrepanze.

Di seguito si riporta la test suite:

ID	Descrizione	Precondizione	Input	Output atteso	Post condizione attesa	Output ottenuto	Esito
TC1	Caricare una classe senza nome. Verificare che i test non vengano caricati se non si dà il nome alla classe che si sta caricando.	L'Admin ha effettuato il login e si trova nella pagina Add Class and Test.	Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “Errore, richiesto il nome della classe”.	Ritorno sulla pagina Add Class and Test per risolvere l'errore.	Visualizzazione di un popup con scritto “Errore, richiesto il nome della classe”.	PASS
TC2	Caricare una classe senza file.java Verificare che i test non vengano caricati se non si carica il file.java associato.	L'Admin ha effettuato il login e si trova nella pagina Add Class and Test.	Scrivere il nome della classe. Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “Errore, richiesto il file.java della classe”.	Ritorno sulla pagina Add Class and Test per risolvere l'errore.	Visualizzazione di un popup con scritto “Errore, richiesto il file.java della classe”.	PASS
TC3	Caricare una classe già esistente. Verificare che la classe non venga caricata se è già presente nel sistema.	L'Admin si trova nella pagina Add Class and Test. Nel Sistema esiste già una classe denominata “Calcolatrice”.	Scrivere il nome della classe: “Calcolatrice”. Aggiungere il file.java. Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “Errore, classe già presente”.	Ritorno sulla pagina Add Class and Test per risolvere l'errore.	Visualizzazione di un popup con scritto “Errore, classe già presente”.	PASS
TC4	Caricare una classe senza Randoop ed EvoSuite. Verificare che la classe venga caricata anche se non si inseriscono i test associati ai due robot.	L'Admin ha effettuato il login e si trova nella pagina Add Class and Test.	Scrivere il nome della classe. Aggiungere il file.java. Aggiungere il test di ChatGPT. Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	Ritorno alla pagina di visualizzazione di tutte le classi caricate. Nella schermata è presente la classe con riferimento a ChatGPT. Nel volume vengono caricate classi e test, mentre in T8 e T9 non viene salvato nulla.	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	PASS
TC5	Caricare una classe con Randoop o EvoSuite. Verificare che la classe venga caricata anche se si carica solo uno dei due robot.	L'Admin ha effettuato il login e si trova nella pagina Add Class and Test.	Scrivere il nome della classe. Aggiungere file generati da EvoSuite. Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	Ritorno alla pagina di visualizzazione di tutte le classi caricate. Nella schermata è presente la classe con riferimento a EvoSuite. In T8 e nel volume condiviso vengono caricate la classe	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	PASS

					e il test.		
TC6	Caricare una classe senza test. Verificare che la classe venga caricata anche se non si caricano i test associati.	L'Admin ha effettuato il login e si trova nella pagina Add Class and Test.	Scrivere il nome della classe. Aggiungere il file.java della classe. Cliccare sul tasto “Upload class and tests”.	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	Ritorno alla pagina di visualizzazione di tutte le classi caricate. Nella schermata è presente la classe senza il riferimento a nessun robot.	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	PASS

localhost dice
Errore, richiesto il nome della classe.

Category 1

Category 2

Category 3

Upload your class (.java) Caricamento in corso...

Robot

Randoop

Select the robot you wish to upload the test

Upload your test (.zip) RandoopTest.zip

TC1

localhost dice
Errore, richiesto il file.java della classe.

Category 1

Category 2

Category 3

Upload your class (.java) Caricamento in corso...

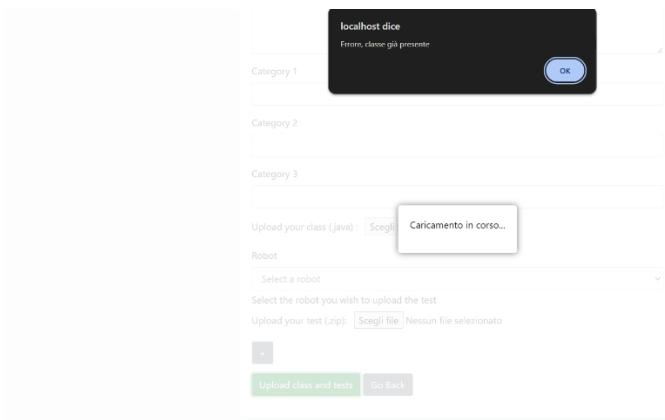
Robot

Randoop

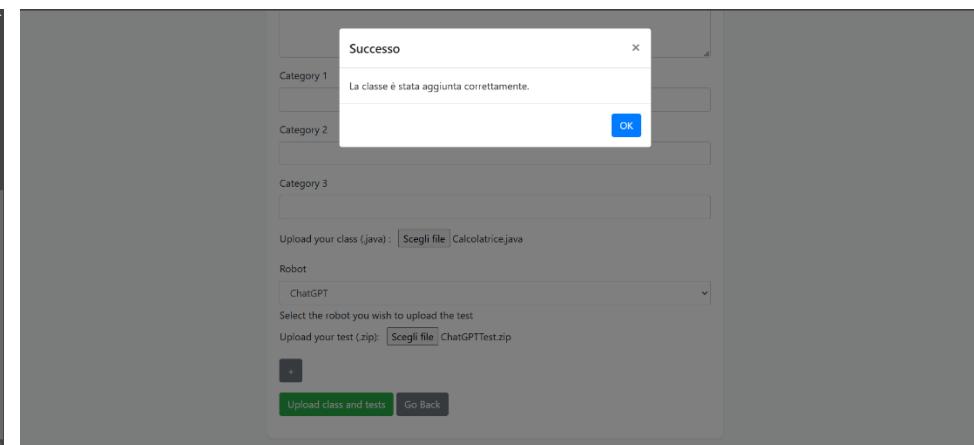
Select the robot you wish to upload the test

Upload your test (.zip) RandoopTest.zip

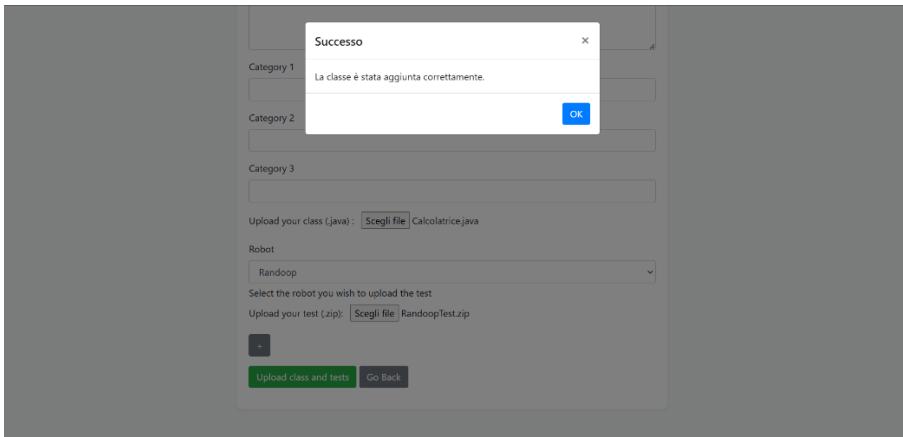
TC2



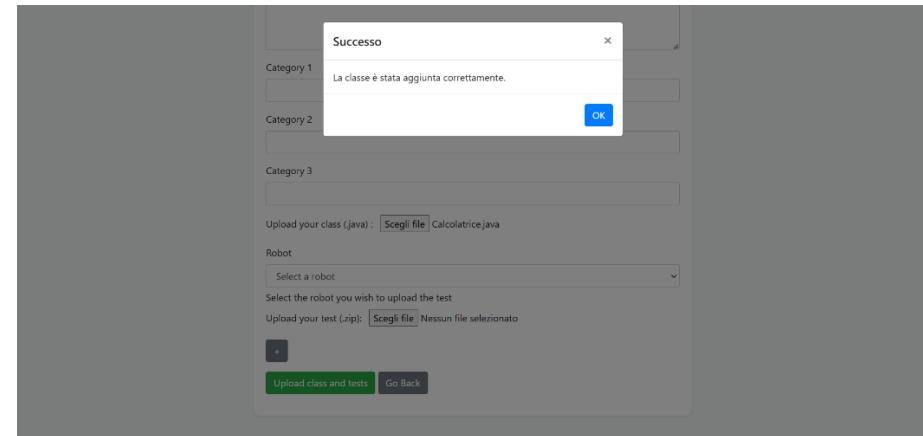
TC3



TC4



TC5



TC6

7.2. Testing API

Per testare correttamente il funzionamento dell'API è stato usato Postman facendo riferimento alla test suite riportata di seguito.

7.2.1. GET/classes

Descrizione: restituisce la lista di classi presenti nel sistema

Metodo: GET

7.2.1.1. Test 1

Body:

Precondizione: nel sistema c'è la classe Calcolatrice e la classe FontInfo

Postcondizione: la chiamata restituisce Calcolatrice e FontInfo

Risposta attesa: 200

```
GET http://localhost/classes

Params Authorization Headers (7) Body Scripts • Settings
Pre-request
1 pm.test("Status code è 200",function(){
2 |   pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Classi presenti",function(){
6 |   var data = pm.response.json();
7 |   pm.expect(data.data).to.eql(["Calcolatrice","FontInfo"]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) | ⏪
Pretty Raw Preview Visualize JSON ▾
1 {
2   "message": "Classes found",
3   "data": [
4     "Calcolatrice",
5     "FontInfo"
6   ]
7 }
```

```
GET http://localhost/classes

Params Authorization Headers (7) Body Scripts • Settings
Pre-request
1 pm.test("Status code è 200",function(){
2 |   pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Classi presenti",function(){
6 |   var data = pm.response.json();
7 |   pm.expect(data.data).to.eql(["Calcolatrice","FontInfo"]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) | ⏪
Filter Results ▾

PASSED Status code è 200
PASSED Output Atteso: Classi presenti
```

7.2.1.2. Test 2

Body:

Precondizione: nel sistema non c'è alcuna classe

Postcondizione: la chiamata restituisce un errore data l'assenza di classi

Risposta attesa: 404

The screenshot shows the Postman interface with two separate test runs for a GET request to `http://localhost/classes`.

Test 1 (Top):

- Method: GET
- URL: `http://localhost/classes`
- Script (Post-response):

```
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classi non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9
```

- Body (Pretty):

```
1 {
2     "message": "Error, classes not found",
3     "data": []
4 }
```

Test 2 (Bottom):

- Method: GET
- URL: `http://localhost/classes`
- Script (Post-response):

```
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classi non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9
```

- Body (Pretty):

```
1
2
3
4
```

Both tests passed, indicated by green **PASSED** status messages:

- Status code è 404
- Output Atteso: Classi non presenti

7.2.1.3. Test 3

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

The screenshot shows the Postman interface with a test configuration for a 401 error response. The URL is set to `http://localhost/classes`. The "Scripts" tab contains a "Post-response" script:

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

The "Body" tab displays the expected JSON response:

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```

The screenshot shows the Postman interface displaying the test results for the 401 error response. The URL is again `http://localhost/classes`. The "Test Results" tab shows one result:

PASSED Status code è 401

7.2.2. GET/classes/{className}

Descrizione: restituisce il path della classe specificata

Metodo: GET

7.2.2.1. Test 1

Body:

Precondizione: nel sistema non è presente nessuna classe

Postcondizione: la chiamata restituisce un errore data l'assenza di classi

Risposta attesa: 404

The screenshot shows the Postman interface for a GET request to `http://localhost/classes/Calcolatrice`. The 'Scripts' tab is selected, containing the following code:

```
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classe non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9
```

The 'Body' tab is selected, showing the expected JSON response:

```
1 {
2     "message": "Error, class not found",
3     "data": []
4 }
```

The screenshot shows the Postman interface for the same GET request. The 'Test Results (2/2)' tab is selected, displaying two successful tests:

- PASSED Status code è 404
- PASSED Output Atteso: Classe non presenti

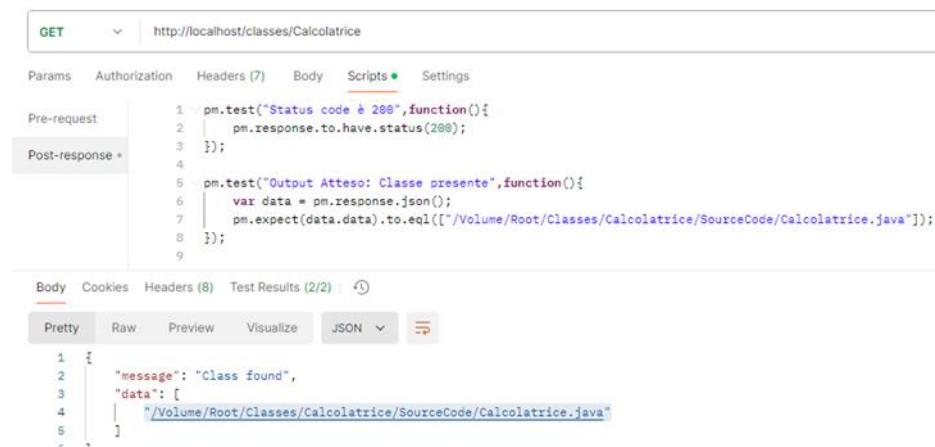
7.2.2.2. Test 2

Body:

Precondizione: nel sistema è presente la classe calcolatrice

Postcondizione: la chiamata restituisce il path della classe all'interno del file system

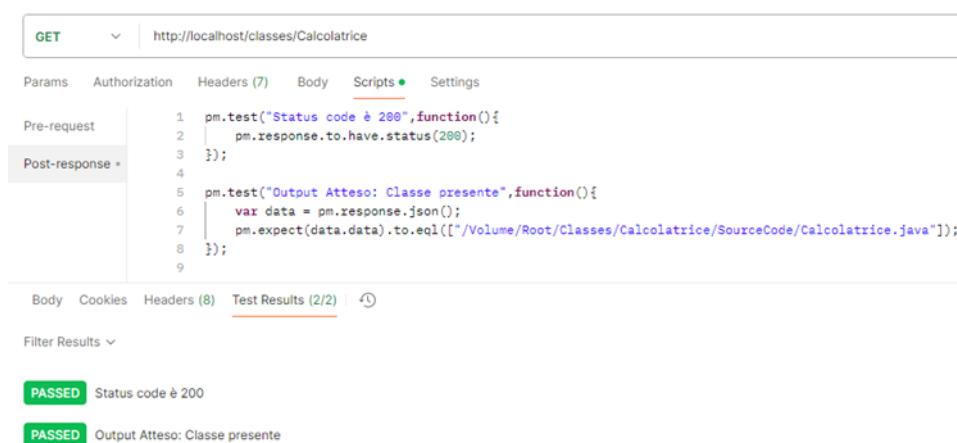
Risposta attesa: 200



```
GET http://localhost/classes/Calcolatrice

Params Authorization Headers (7) Body Scripts * Settings
Pre-request
1 pm.test("Status code è 200",function(){
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Classe presente",function(){
6   var data = pm.response.json();
7   pm.expect(data.data).to.eql(["/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java"]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) ⚡
Pretty Raw Preview Visualize JSON ↗
1 {
2   "message": "Class found",
3   "data": [
4     "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java"
5   ]
6 }
```



```
GET http://localhost/classes/Calcolatrice

Params Authorization Headers (7) Body Scripts * Settings
Pre-request
1 pm.test("Status code è 200",function(){
2   pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Classe presente",function(){
6   var data = pm.response.json();
7   pm.expect(data.data).to.eql(["/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java"]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) ⚡
Filter Results ▾
PASSED Status code è 200
PASSED Output Atteso: Classe presente
```

7.2.2.3. Test 3

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

GET http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

Pre-request Post-response

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```

GET http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

Pre-request Post-response

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 401

7.2.3. GET/classes/{className}/robots

Descrizione: restituisce la lista di robot associati alla classe specificata

Metodo: GET

7.2.3.1. Test 1

Body:

Precondizione: nel sistema non è presente nessuna classe

Postcondizione: la chiamata restituisce un errore data l'assenza della classe specificata

Risposta attesa: 404

The screenshot shows two Postman test results sections. The top section is for the URL `http://localhost/classes/Calcolatrice/robots`. It has a status of `PASSED` and a message: "Status code è 404". The bottom section is for the same URL, also showing a `PASSED` status and a message: "Output Atteso: Classe presente". Both sections include a script tab with the following code:

```
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classe presente",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9
```

7.2.3.2. Test 2

Body:

Precondizione: nel sistema è presente la classe calcolatrice senza alcun robot

Postcondizione: la chiamata restituisce un errore data l'assenza di robot associati alla classe

Risposta attesa: 404

```
GET http://localhost/classes/Calcolatrice/robots

Params Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Robot non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) | ⏱
Pretty Raw Preview Visualize JSON 🔍

1 {
2     "message": "Robots not found",
3     "data": []
4 }
```

```
GET http://localhost/classes/Calcolatrice/robots

Params Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Robot non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9

Body Cookies Headers (8) Test Results (2/2) | ⏱
Filter Results ▾
```

PASSED Status code è 404

PASSED Output Atteso: Classe presente

7.2.3.3. Test 3

Body:

Precondizione: nel sistema è presente la classe calcolatrice con i test di ChatGPT e Randoop

Postcondizione: la chiamata restituisce la lista di robot associati alla classe specificata

Risposta attesa: 200

GET <http://localhost/classes/Calcolatrice/robots>

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 200",function(){
2 | pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Robot presenti",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql(["Randoop","ChatGPT"]);
8 });
9
```

Body Cookies Headers (8) Test Results (2/2)

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Robots found",
3   "data": [
4     "Randoop",
5     "ChatGPT"
6   ]
7 }
```

GET <http://localhost/classes/Calcolatrice/robots>

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 200",function(){
2 | pm.response.to.have.status(200);
3 });
4
5 pm.test("Output Atteso: Robot presenti",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql(["Randoop","ChatGPT"]);
8 });
9
```

Body Cookies Headers (8) Test Results (2/2)

Filter Results ▾

PASSED Status code è 200

PASSED Output Atteso: Robot presenti

7.2.3.4. Test 4

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

GET | http://localhost/classes/Calcolatrice/robots

Params | Authorization | Headers (9) | Body | Scripts | Settings

Pre-request | Post-response

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

Body | Cookies | Headers (8) | Test Results (1/1) | ⏱

Pretty | Raw | Preview | Visualize | JSON | ⚡

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```

GET | http://localhost/classes/Calcolatrice/robots

Params | Authorization | Headers (9) | Body | Scripts | Settings

Pre-request | Post-response

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

Body | Cookies | Headers (8) | Test Results (1/1) | ⏱

Filter Results ▾

PASSED Status code è 401

7.2.4. GET/classes/{className}/{robotName}

Descrizione: restituisce il path del robot associato alla classe specificata

Metodo: GET

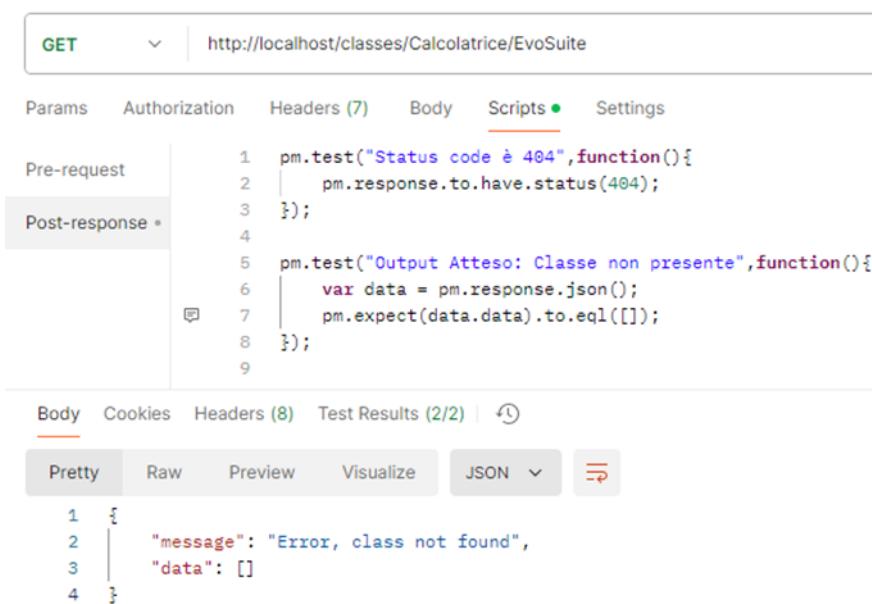
7.2.4.1. Test 1

Body:

Precondizione: nel sistema non è presente nessuna classe

Postcondizione: la chiamata restituisce un errore data la mancanza della classe specificata

Risposta attesa: 404



```
GET http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classe non presente",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql([]);
8 });
9

Post-response
Body Cookies Headers (8) Test Results (2/2) ⏱

Pretty Raw Preview Visualize JSON ↻

1 {
2   "message": "Error, class not found",
3   "data": []
4 }
```



```
GET http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5 pm.test("Output Atteso: Classe non presente",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql([]);
8 });
9

Post-response
Body Cookies Headers (8) Test Results (2/2) ⏱

Filter Results ▾

PASSED Status code è 404
PASSED Output Atteso: Classe non presente
```

7.2.4.2. Test 2

Body:

Precondizione: nel sistema è presente la classe calcolatrice senza EvoSuite

Postcondizione: la chiamata restituisce un errore data la mancanza della di EvoSuite associato alla classe specificata

Risposta attesa: 404

GET http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts ● Settings

Pre-request

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4 
```

Post-response

```
5 pm.test("Output Atteso: Robot non presente",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql([]);
8 });
9 
```

Body Cookies Headers (8) Test Results (2/2) ⚡

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Error, robot not found",
3   "data": []
4 }
```

GET http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts ● Settings

Pre-request

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4 
```

Post-response

```
5 pm.test("Output Atteso: Robot non presente",function(){
6 | var data = pm.response.json();
7 | pm.expect(data.data).to.eql([]);
8 });
9 
```

Body Cookies Headers (8) Test Results (2/2) ⚡

Filter Results ▾

PASSED Status code è 404

PASSED Output Atteso: Robot non presente

7.2.4.3. Test 3

Body:

Precondizione: nel sistema è presente la classe calcolatrice con i test di EvoSuite

Postcondizione: la chiamata restituisce il path dove si trova il test generato da EvoSuite associato alla classe specificata

Risposta attesa: 200

PASSED Status code è 200

PASSED Output Atteso: Robot presente

7.2.4.4. Test 4

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

The screenshot shows a Postman test configuration for a GET request to `http://localhost/classes/Calcolatrice/EvoSuite`. The 'Scripts' tab is selected. In the 'Pre-request' section, there is no code. In the 'Post-response' section, the following JavaScript code is present:

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

Below the scripts, the 'Body' tab is selected, showing a JSON response:

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```

The screenshot shows the test results for the same POST request. The 'Scripts' tab is selected. In the 'Post-response' section, the same JavaScript code is present:

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

The 'Test Results' tab is selected, showing one successful result:

PASSED Status code è 401

7.2.5. POST/classes/{className}

Descrizione: esegue l'update della classe da testare

Metodo: POST

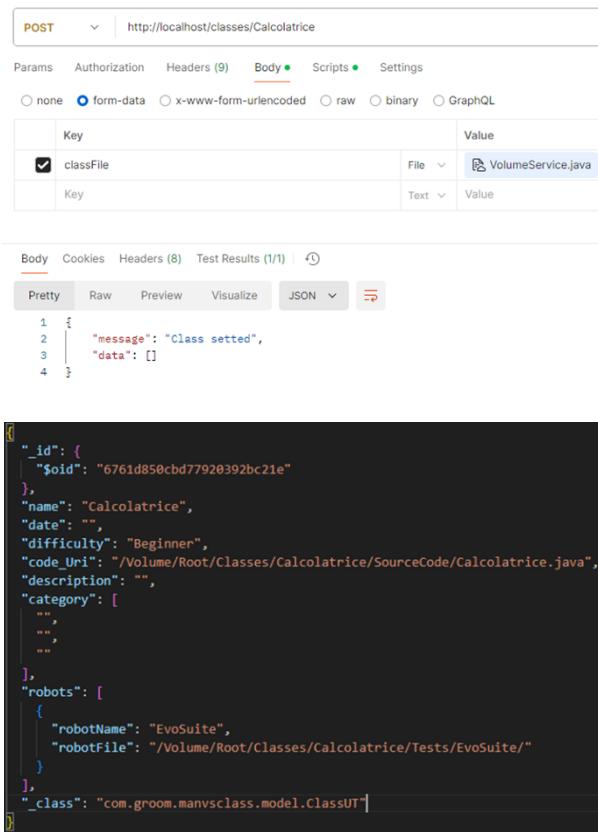
7.2.5.1. Test 1

Body: "classFile": VolumeService.java

Precondizione: nel sistema è presente la classe Calcolatrice con associato il file Calcolatrice.java

Postcondizione: la chiamata restituisce esito positivo e sia nel DB, che nel filesystem viene aggiornato il file.

Risposta attesa: 200



POST http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

Body (1)

Key Value

classFile File VolumeService.java

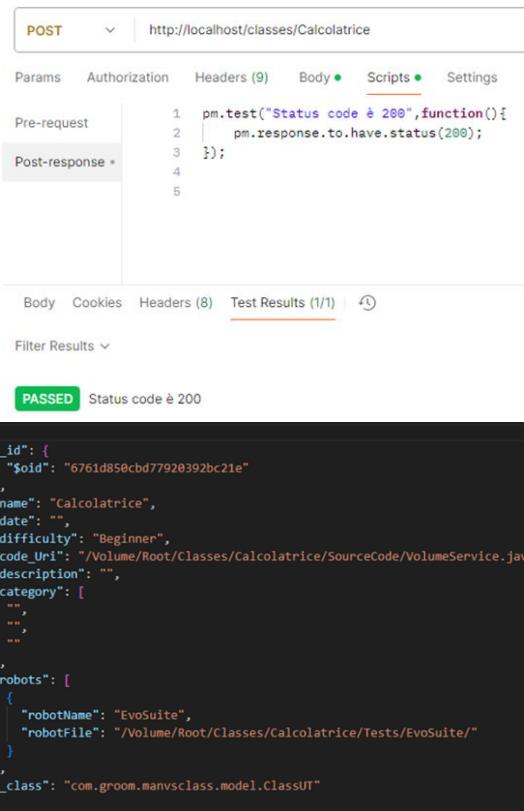
Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Class setted",
3   "data": []
4 }
```

Body (1)

```
{"_id": {"$oid": "6761d850cbd77920392bc21e"}, "name": "Calcolatrice", "date": "", "difficulty": "Beginner", "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java", "description": "", "category": [ "", "", "" ], "robots": [ { "robotName": "EvoSuite", "robotFile": "/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/" } ], "_class": "com.groom.mansclass.model.ClassUT"}
```



POST http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

Pre-request

```
1 pm.test("Status code è 200",function(){
2   pm.response.to.have.status(200);
3 });
4
5
```

Post-response

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 200

```
{"_id": {"$oid": "6761d850cbd77920392bc21e"}, "name": "Calcolatrice", "date": "", "difficulty": "Beginner", "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/VolumeService.java", "description": "", "category": [ "", "", "" ], "robots": [ { "robotName": "EvoSuite", "robotFile": "/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/" } ], "_class": "com.groom.mansclass.model.ClassUT"}
```

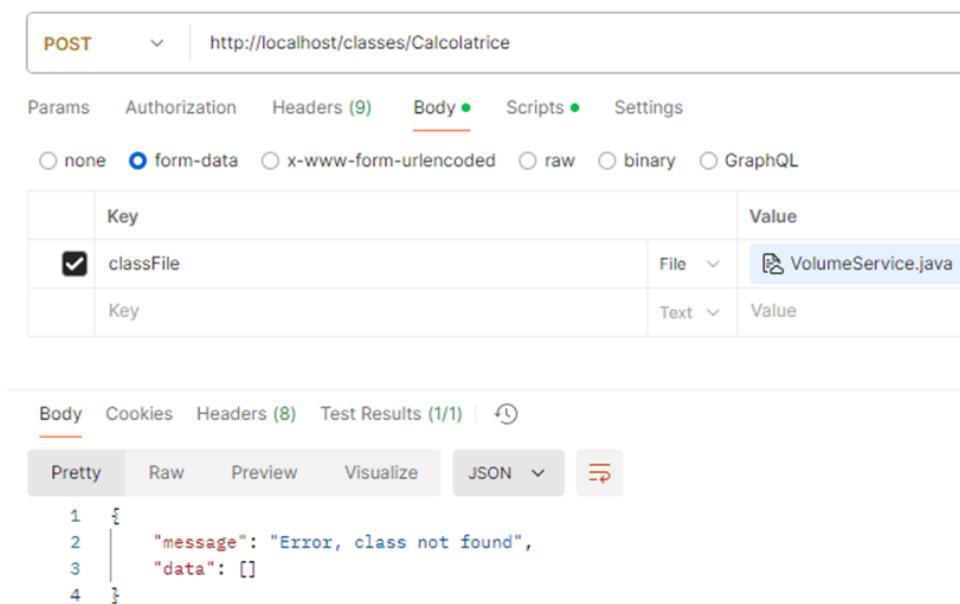
7.2.5.2. Test 2

Body: "classFile": VolumeService.java

Precondizione: nel sistema non è presente la classe calcolatrice

Postcondizione: la chiamata restituisce un errore data l'assenza della classe Calcolatrice

Risposta attesa: 404



POST http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

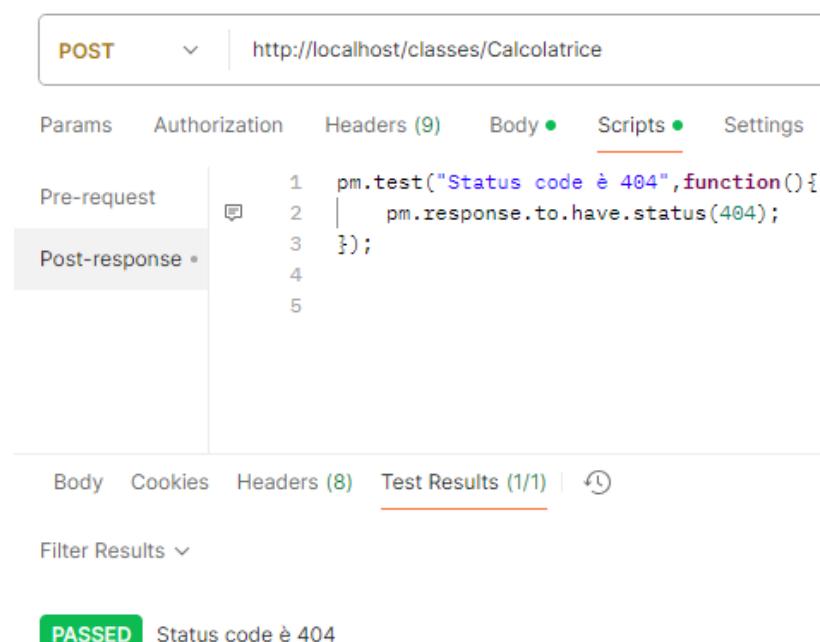
Body (form-data)

Key	Value
classFile	File VolumeService.java
Key	Text Value

Body Cookies Headers (8) Test Results (1/1) ↻

Pretty Raw Preview Visualize JSON ✖

```
1 {  
2   "message": "Error, class not found",  
3   "data": []  
4 }
```



POST http://localhost/classes/Calcolatrice

Params Authorization Headers (9) Body Scripts Settings

Pre-request Post-response

```
1 pm.test("Status code è 404",function(){  
2   | pm.response.to.have.status(404);  
3 });  
4  
5
```

Body Cookies Headers (8) Test Results (1/1) ↻

Filter Results ▼

PASSED Status code è 404

7.2.5.3. Test 3

Body: "classFile": VolumeService.java

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

The screenshot shows two identical Postman test results for a POST request to `http://localhost/classes/Calcolatrice`. Both tests have the following parameters:

- Method: POST
- URL: `http://localhost/classes/Calcolatrice`
- Headers (9): (This section is listed but appears empty in the screenshot)
- Body (form-data):
 - Key: classFile Value: VolumeService.java
 - Key: (empty) Value: (empty)
- Test Results (1/1):
 - Body:

```
1 {  
2   "message": "Error, token not valid",  
3   "data": []  
4 }
```
 - Cookies: (empty)
 - Headers (8): (empty)
 - Test Results (1/1): (highlighted in red)

At the bottom, there is a green button labeled "PASSED" with the text "Status code è 401".

7.2.6. POST/classes/{className}/{robotName}

Descrizione: esegue l'UPDATE o l'INSERT di un test

Metodo: POST

7.2.6.1. Test 1

Body: "robotFile": ChatGPTTest.zip

Precondizione: nel sistema è presente la classe Calcolatrice senza il robot ChatGPT

Postcondizione: la chiamata restituisce esito positivo e sia nel DB, che nel filesystem viene aggiunto il test generato da ChatGPT per la classe Calcolatrice

Risposta attesa: 200

The image shows two side-by-side Postman request panels. Both requests are for the URL `http://localhost/classes/Calcolatrice/ChatGPT` using the POST method.

Request 1 (Left):

- Body:** A file named `ChatGPTTest.zip` is selected for the `robotFile` key.
- Response Body (Pretty JSON):**

```
1 {  
2   "_id": {  
3     "$oid": "676209578aa6ce4abe551279"  
4   },  
5   "name": "calcolatrice",  
6   "date": "",  
7   "difficulty": "Beginner",  
8   "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java",  
9   "description": "",  
10  "category": [  
11    "",  
12    "",  
13    "",  
14  ],  
15  "robots": [],  
16  "_class": "com.groom.manvsclass.model.ClassUT"  
17}
```

Request 2 (Right):

- Body:** A file named `ChatGPTTest.zip` is selected for the `robotFile` key.
- Response Body (Pretty JSON):**

```
1 {  
2   "_id": {  
3     "$oid": "676209578aa6ce4abe551279"  
4   },  
5   "name": "calcolatrice",  
6   "date": "",  
7   "difficulty": "Beginner",  
8   "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java",  
9   "description": "",  
10  "category": [  
11    "",  
12    "",  
13    "",  
14  ],  
15  "robots": [  
16    {  
17      "robotName": "ChatGPT",  
18      "robotFile": "/Volume/Root/Classes/Calcolatrice/Tests/ChatGPT/"  
19    }  
20  ],  
21  "_class": "com.groom.manvsclass.model.ClassUT"  
22}
```

7.2.6.2. Test 2

Body: "robotFile": EvoSuiteTest.zip

Precondizione: nel sistema è presente la classe Calcolatrice con già il robot ChatGPT

Postcondizione: la chiamata restituisce esito positivo e sia nel DB, che nel filesystem viene aggiornato il test generato da ChatGPT per la classe Calcolatrice

Risposta attesa: 200

The image shows two side-by-side Postman API requests. Both requests are to the URL `http://localhost/classes/Calcolatrice/ChatGPT` via POST method.

Request 1 (Left): The body is set to "form-data" and contains a file named "EvoSuiteTest.zip" under the key "robotFile". The response shows a JSON object with a single entry: "message": "Robot setted", "data": [].

```
1 {
2   "message": "Robot setted",
3   "data": []
4 }
```

Request 2 (Right): The body is also set to "form-data" and contains a file named "EvoSuiteTest.zip" under the key "robotFile". The response is marked as "PASSED" with the message "Status code è 200".

Both requests show a successful status code of 200 in their respective "Test Results" sections.

7.2.6.3. Test 3

Body: “robotFile”: VolumeService.zip

Precondizione: nel sistema è presente la classe Calcolatrice e nel file di configurazione non esiste il robot Pippo.

Postcondizione: la chiamata restituisce un errore in quanto il robot non è disponibile

Risposta attesa: 400

The screenshot shows the Postman interface with two separate requests side-by-side.

Request 1 (Left): Targeted at `http://localhost/classes/Calcolatrice/Pippo`. It's a POST request with the following script in the "Pre-request" section:

```
1 pm.test("Status code è 400",function(){
2 | pm.response.to.have.status(400);
3 });
4
5
```

Request 2 (Right): Targeted at `http://localhost/classes/Calcolatrice/Pippo`. It's also a POST request with the same "Pre-request" script:

```
1 pm.test("Status code è 400",function(){
2 | pm.response.to.have.status(400);
3 });
4
5
```

Both requests have "Test Results (1/1)" status indicators below them. The results for Request 1 show a JSON response body:

```
1 {
2   "message": "Robot not available.",
3   "data": []
4 }
```

The results for Request 2 show a green "PASSED" status with the message "Status code è 400".

7.2.6.4. Test 4

Body: "robotFile": VolumeService.zip

Precondizione: nel sistema non è presente la classe Calcolatrice

Postcondizione: la chiamata restituisce un errore causato dall'assenza della classe Calcolatrice

Risposta attesa: 404

The screenshot shows the Postman interface with a POST request to `http://localhost/classes/Calcolatrice/EvoSuite`. The request includes a Pre-request script and a Post-response script. The response body is shown in JSON format.

Pre-request Script:

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5
```

Post-response Script:

```
1 {
2   "message": "Error, class not found",
3   "data": []
4 }
```

The screenshot shows the Postman interface with a POST request to `http://localhost/classes/Calcolatrice/EvoSuite`. The request includes a Pre-request script and a Post-response script. The response body is shown in JSON format. The status bar at the bottom indicates the test passed.

Test Results:

PASSED Status code è 404

7.2.6.5. Test 5

Body: "robotFile": ChatGPTTest.zip

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

POST http://localhost/classes/Calcolatrice/ChatGPT

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value
<input checked="" type="checkbox"/> classFile	File <input type="file"/> ChatGPTTest.zip
Key	Text Value

Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1
2     "message": "Error, token not valid",
3     "data": []
4
```

POST http://localhost/classes/Calcolatrice/ChatGPT

Params Authorization Headers (9) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value
<input checked="" type="checkbox"/> classFile	File <input type="file"/> ChatGPTTest.zip
Key	Text Value

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 401

7.2.7. DELETE/classes/{className}

Descrizione: elimina l'intera classe con tutti i test associati

Metodo: **DELETE**

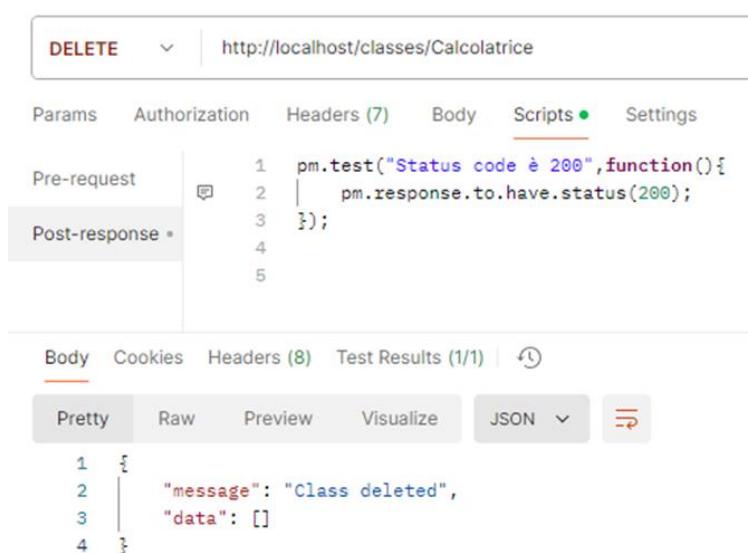
7.2.7.1. Test 1

Body:

Precondizione: nel sistema è presente la classe Calcolatrice

Postcondizione: la chiamata restituisce esito positivo e sia nel DB, che nel filesystem viene eliminata la classe

Risposta attesa: 200



DELETE <http://localhost/classes/Calcolatrice>

Params Authorization Headers (7) Body Scripts • Settings

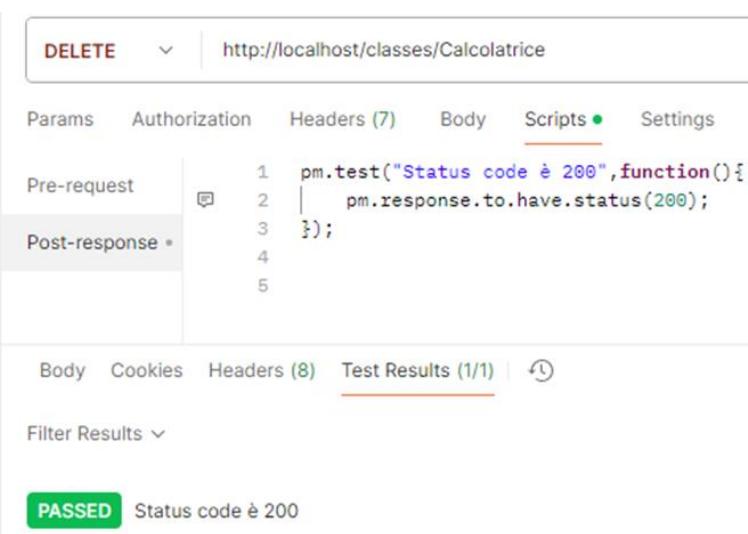
Pre-request Post-response

```
1 pm.test("Status code è 200",function(){
2 | pm.response.to.have.status(200);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Class deleted",
3   "data": []
4 }
```



DELETE <http://localhost/classes/Calcolatrice>

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 200",function(){
2 | pm.response.to.have.status(200);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 200

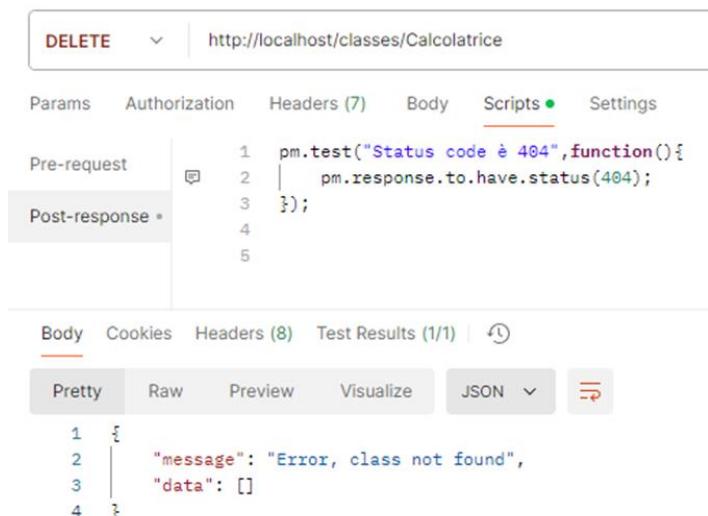
7.2.7.2. Test 2

Body:

Precondizione: nel sistema non è presente la classe calcolatrice

Postcondizione: la chiamata restituisce un errore data l'assenza della classe Calcolatrice

Risposta attesa: 404



DELETE <http://localhost/classes/Calcolatrice>

Params Authorization Headers (7) Body Scripts • Settings

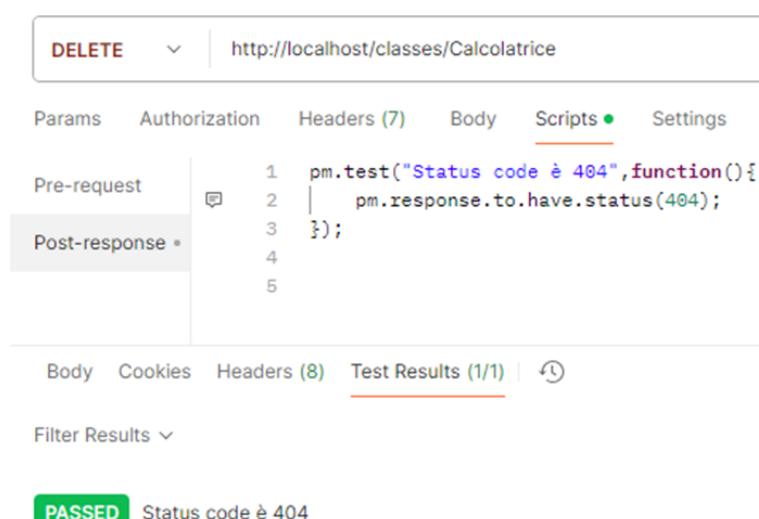
Pre-request Post-response

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1) ⏱

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Error, class not found",
3   "data": []
4 }
```



DELETE <http://localhost/classes/Calcolatrice>

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1) ⏱

Filter Results ▾

PASSED Status code è 404

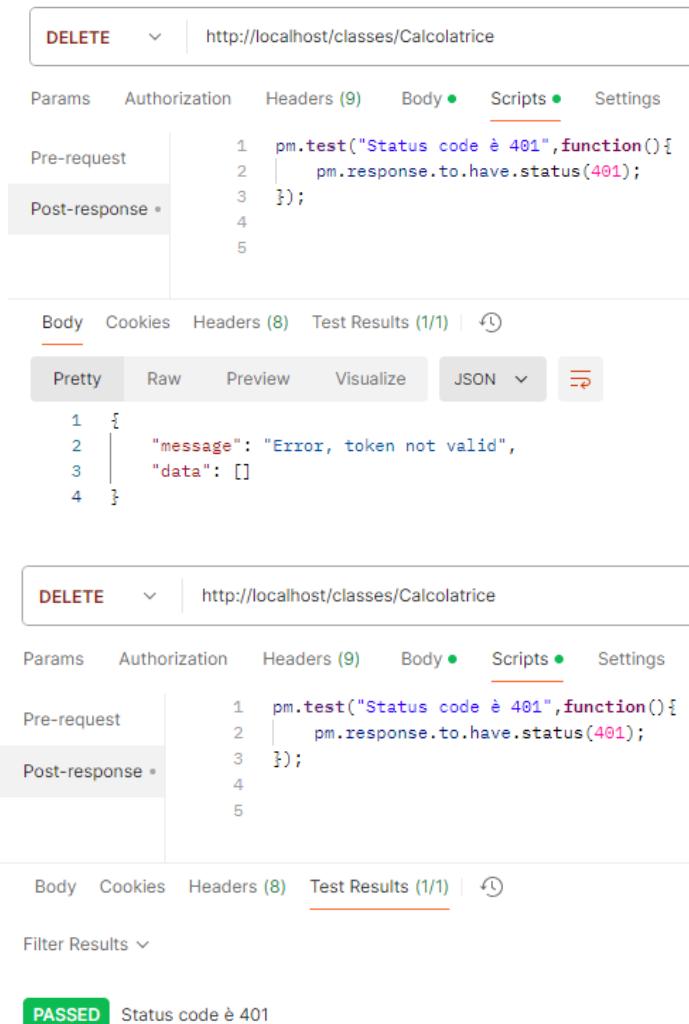
7.2.7.3. Test 3

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401



The screenshot shows the Postman interface with a test result for a DELETE request to `http://localhost/classes/Calcolatrice`. The test script in the 'Post-response' section is:

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```

The 'Body' tab shows the response body in JSON format:

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```

The 'Test Results' tab indicates 1/1 test passed. A green 'PASSED' button at the bottom left shows the status code 401.

7.2.8. DELETE/classes/{className}/{robotName}

Descrizione: elimina un solo test specificato associato alla classe

Metodo: **DELETE**

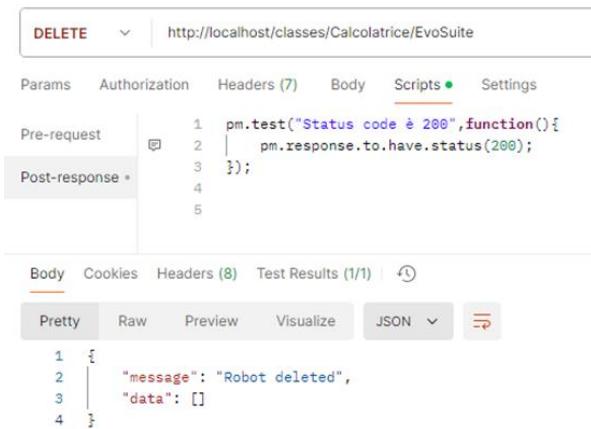
7.2.8.1. Test 1

Body:

Precondizione: nel sistema è presente la classe Calcolatrice con il test EvoSuite

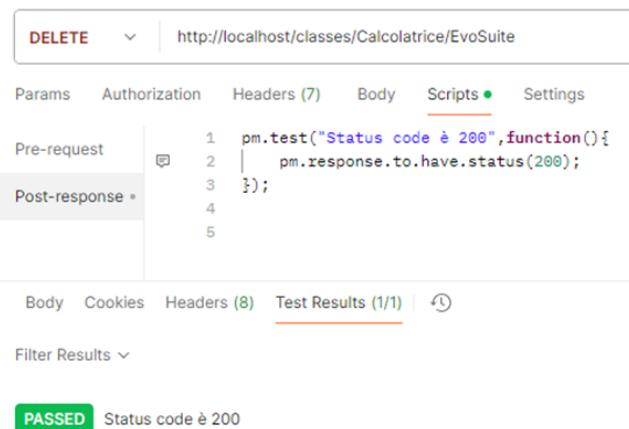
Postcondizione: la chiamata restituisce esito positivo e sia nel DB, che nel filesystem viene eliminato il test EvoSuite associato alla classe

Risposta attesa: 200



```
DELETE http://localhost/classes/Calcolatrice/EvoSuite
{
  "message": "Robot deleted",
  "data": []
}
```

```
[{"_id": {"$oid": "6762087d8aa6ce4abe551276"}, "name": "Calcolatrice", "date": "", "difficulty": "Beginner", "code_Uri": "/Volume/Root/classes/Calcolatrice/SourceCode/Calcolatrice.java", "description": "", "category": [ "", "", "" ], "robots": [ {"robotName": "EvoSuite", "robotFile": "/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/" } ], "_class": "com.groom.mansclass.model.ClassUT"}]
```



```
PASSED Status code è 200
```

```
[{"_id": {"$oid": "6762087d8aa6ce4abe551276"}, "name": "Calcolatrice", "date": "", "difficulty": "Beginner", "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java", "description": "", "category": [ "", "", "" ], "robots": [], "_class": "com.groom.mansclass.model.ClassUT"}]
```

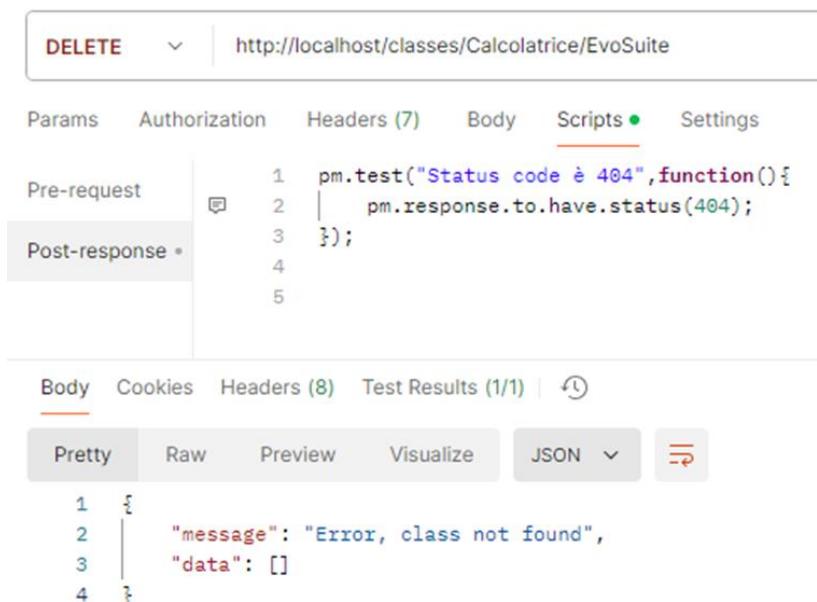
7.2.8.2. Test 2

Body:

Precondizione: nel sistema non è presente la classe calcolatrice

Postcondizione: la chiamata restituisce un errore data l'assenza della classe Calcolatrice

Risposta attesa: 404



DELETE http://localhost/classes/Calcolatrice/EvoSuite

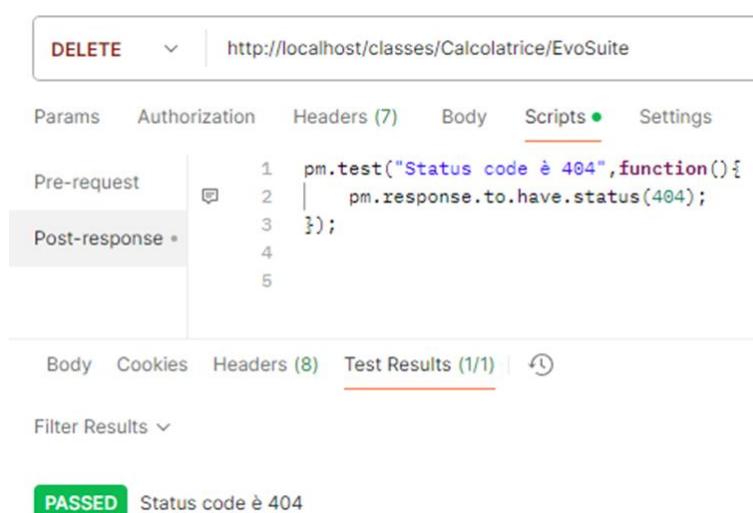
Params Authorization Headers (7) Body Scripts • Settings

Pre-request 1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 |});
4
5

Post-response Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Error, class not found",  
3   "data": []  
4 }
```



DELETE http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts • Settings

Pre-request 1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 |});
4
5

Post-response Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 404

7.2.8.3. Test 3

Body:

Precondizione: nel sistema è presente la classe calcolatrice, ma questa non ha il test EvoSuite

Postcondizione: la chiamata restituisce un errore data l'assenza del test generato da EvoSuite per la classe Calcolatrice

Risposta attesa: 404

DELETE http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Error, robot not found",
3   "data": []
4 }
```

DELETE http://localhost/classes/Calcolatrice/EvoSuite

Params Authorization Headers (7) Body Scripts • Settings

Pre-request Post-response

```
1 pm.test("Status code è 404",function(){
2 | pm.response.to.have.status(404);
3 });
4
5
```

Body Cookies Headers (8) Test Results (1/1)

Filter Results ▾

PASSED Status code è 404

7.2.8.4. Test 4

Body:

Precondizione: l'utente non ha effettuato l'accesso

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT

Risposta attesa: 401

The screenshot shows two identical Postman test runs for the URL `http://localhost/classes/Calcolatrice/Calcolatrice`. Both are set to DELETE method.

Test 1 (Top):

- Script (Post-response):**

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```
- Body (Pretty):**

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```
- Test Results (1/1):** Failed (401)

Test 2 (Bottom):

- Script (Post-response):**

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3 });
4
5
```
- Body (Pretty):**

```
1 {
2   "message": "Error, token not valid",
3   "data": []
4 }
```
- Test Results (1/1):** Failed (401)

Overall Status: PASSED Status code è 401

7.2.9. POST/fileSystem

Descrizione: aggiunge o aggiorna il file al path specificato

Metodo: POST

7.2.9.1. Test 1

Body: path: “/Volume/Root/ChatGPTTest.java” file: VolumeService.java

Precondizione: nel filesystem condiviso è presente il path /Volume/Root/TestChatGPT.java

Postcondizione: la chiamata restituisce esito positivo ed aggiorna il file al path specificato

Risposta attesa: 200

The screenshot shows a file system interface with the following structure:

- Root
 - > 02Level
 - > 03Level
 - ChatGPTTest.zip
 - > Classes
 - TestChatGPT.java

Below this, there are two Postman requests:

Request 1 (Left):

- Method: POST
- URL: http://localhost/fileSystem
- Body (form-data):
 - path: /Volume/Root/TestChatGPT.java
 - file: VolumeService.java

Request 2 (Right):

- Method: POST
- URL: http://localhost/fileSystem
- Body (form-data):
 - path: /Volume/Root/TestChatGPT.java
 - file: VolumeService.java

The results for both requests show a successful response:

Test Results (1/1):

```
1 {  
2   "message": "Path element setted",  
3   "data": []  
4 }
```

File System Result (Bottom):

- Root
 - > 02Level
 - > 03Level
 - ChatGPTTest.zip
 - > Classes
 - VolumeService.java

7.2.9.2. Test 2

Body: path: “/Vlume/Root/VolumeService.java” file: VolumeService.java

Precondizione: nel filesystem condiviso è presente il path /Vlume/Root/VolumeService.java

Postcondizione: la chiamata restituisce un errore in quanto il path non è valido (non inizia con Volume/Root) e non esiste.

Risposta attesa: 400

The screenshot shows the Postman interface with two requests. The top request is a POST to `http://localhost/fileSystem` with a body containing a JSON object with keys `path` and `file`. The response body is a JSON object with a message "Error, path not valid". The bottom request is also a POST to `http://localhost/fileSystem`, with a script in the Pre-request tab that checks if the status code is 400. The test results show one failure where the status code was 400.

POST http://localhost/fileSystem

Params Authorization Headers (8) Body Scripts Settings

Body (form-data)

Key	Value
path	/Vlume/Root/VolumeService.java
file	VolumeService.java

Body Cookies Headers (8) Test Results (1/1)

{ } JSON Preview Visualize

```
1 {  
2   "message": "Error, path not valid",  
3   "data": []  
4 }
```

POST http://localhost/fileSystem

Params Authorization Headers (8) Body Scripts Settings

Pre-request

```
1 pm.test("Status code è 400",function(){  
2   pm.response.to.have.status(400);  
3 });
```

Post-response

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 400

7.2.9.3. Test 3

Body: path: “/Volume/Root/Classes/Cal” file: VolumeService.java

Precondizione: nel filesystem condiviso è presente il path /Volume/Root/Classes/Cal”

Postcondizione: la chiamata restituisce esito positivo dato che il path esiste ed è valido.

Il file viene copiato nella cartella.

Risposta attesa: 200

The screenshot shows a file system interface with the following structure:

- Root
 - 02Level
 - 03Level
 - ChatGPTTest.zip
 - Classes
 - Cal
 - VolumeService.java

A POST request is being prepared to upload 'VolumeService.java' to the 'Cal' folder. The request details are as follows:

- Method: POST
- URL: http://localhost/fileSystem
- Body tab is selected.
- Params, Authorization, Headers (8), Scripts, and Settings tabs are visible but not selected.
- Body content:
 - Key: path, Value: /Volume/Root/Classes/Cal
 - Key: file, Value: VolumeService.java

The screenshot shows a POST request in Postman with the following details:

- Method: POST
- URL: http://localhost/fileSystem
- Body tab is selected.
- Body content (JSON):

```
1 {  
2   "message": "Path element setted",  
3   "data": []  
4 }
```
- Headers (8) tab is selected.
- Test Results (1/1) tab is selected, showing a green 'PASSED' status.

The screenshot shows a POST request in Postman with the following details:

- Method: POST
- URL: http://localhost/fileSystem
- Body tab is selected.
- Body content (JSON):

```
1 pm.test("Status code è 200",function(){  
2   pm.response.to.have.status(200);  
3 });
```
- Headers (8) tab is selected.
- Test Results (1/1) tab is selected, showing a green 'PASSED' status.

The screenshot shows a file system interface with the following structure:

- Root
 - 02Level
 - 03Level
 - ChatGPTTest.zip
 - Classes
 - Cal
 - SourceCode
 - VolumeService.java
 - VolumeService.java

The 'Cal' folder under 'Classes' now contains 'SourceCode' and two 'VolumeService.java' files, indicating the file was successfully copied.

7.2.9.4. Test 4

Body: path: “/Volume/Root/Classes/Cal” file: VolumeService.java

Precondizione: il chiamante non possiede il JWT

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT.

Risposta attesa: 401

The screenshot shows the Postman interface with a failed test result. The test script in the 'Post-response' tab is as follows:

```
1 pm.test("Status code è 401",function(){
2     pm.response.to.have.status(401);
3 });
```

The 'Test Results' section shows 1/1 failure, indicating the test passed. The status message is "Status code è 401".

7.2.10. DELETE/fileSystem

Descrizione: elimina il file o la directory al path specificato

Metodo: POST

7.2.10.1. Test 1

Body: path: "/Volume/Root/ VolumeService.java"

Precondizione: nel filesystem condiviso è presente il path /Volume/Root/ VolumeService.java

Postcondizione: la chiamata restituisce esito positivo ed elimina il file al path specificato

Risposta attesa: 200

The screenshot shows the Postman interface with a DELETE request to `http://localhost/fileSystem`. The request body is set to `path: /Volume/Root/ VolumeService.java`. The response body is displayed as JSON:

```
1 {  
2   "message": "Path element deleted",  
3   "data": []  
4 }
```

The left sidebar shows a file system structure with a root folder containing 02Level, 03Level, ChatGPTTest.zip, and Classes. Inside Classes, there is a file named VolumeService.java. After the delete operation, the Classes folder is empty.

7.2.10.2. Test 2

Body: path: “/Volume/Rot/ChatGPTTest.zip”

Precondizione: nel filesystem condiviso non è presente il path /Volume/Rot/ChatGPTTest.zip

Postcondizione: la chiamata restituisce errore in quanto il path non esiste.

Risposta attesa: 404

The screenshot shows a Postman interface with a DELETE request to `http://localhost/fileSystem`. The **Body** tab is selected, showing a form-data body with a key `path` set to `/Volume/Rot/ChatGPTTest.zip`. A file named `VolumeService.java` is attached to the `file` field. The test results show 1/1 passed with status code 404.

The screenshot shows a Postman interface with a DELETE request to `http://localhost/fileSystem`. The **Scripts** tab is selected, displaying a failing test script:

```
1 pm.test("Status code è 404",function(){
2     pm.response.to.have.status(404);
3 });
```

The test results show 1/1 failed with status code 404.

7.2.10.3. Test 3

Body: path: “/Volume/Root/Classes/Cal”

Precondizione: il chiamante non possiede il JWT

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT.

Risposta attesa: 401

The screenshot shows the Postman interface with a DELETE request to `http://localhost/fileSystem`. The Body tab is selected, showing a JSON structure with two fields: `path` (set to `/Volume/Root/Classes/Cal`) and `Key` (set to `Value`). The Scripts tab contains a pre-request script with the following code:

```
1 pm.test("Status code è 401",function(){
2 | pm.response.to.have.status(401);
3});
```

The Test Results tab shows a single result: **PASSED** Status code è 401.

7.2.11. POST/lock

Descrizione: effettua l'operazione di lock sul path specificato

Metodo: POST

7.2.11.1. Test 1

Body: path: "/Volume/Root/ChatGPTTest.zip"

Precondizione: nel filesystem condiviso è presente il path /Volume/Root/ChatGPTTest.zip e nessun altro ha bloccato la risorsa per la scrittura.

Postcondizione: la chiamata restituisce esito positivo e il percorso viene bloccato per la lettura.

Risposta attesa: 200

The screenshot shows the Postman interface with a successful POST request to `http://localhost/lock`. The request body is set to `form-data` with a key `path` containing the value `/Volume/Root/ChatGPTTest.zip`. The response body is displayed as JSON:

```
1 {  
2   "message": "Path locked",  
3   "data": []  
4 }
```

In the bottom section, a script is defined in the `Post-response` tab:

```
1 pm.test("Status code è 200",function(){  
2   pm.response.to.have.status(200);  
3 });
```

The results show a single test result: **PASSED** Status code è 200.

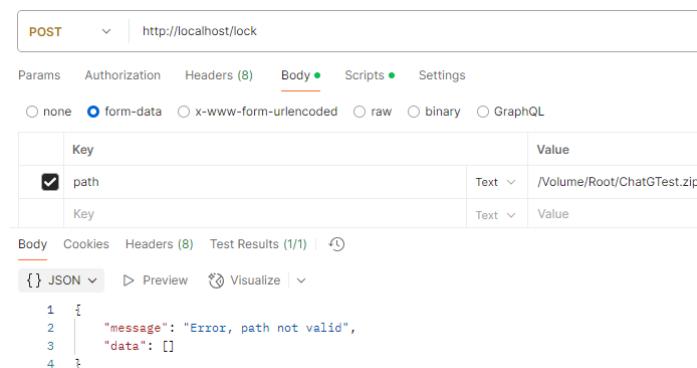
7.2.11.2. Test 2

Body: path: "/Volume/Root/ChatGTest.zip"

Precondizione: nel filesystem condiviso non è presente il path /Volume/Root/ChatGTest.zip e nessun altro ha bloccato la risorsa per la scrittura.

Postcondizione: la chiamata restituisce errore in quanto il path fornito non è esiste.

Risposta attesa: 400



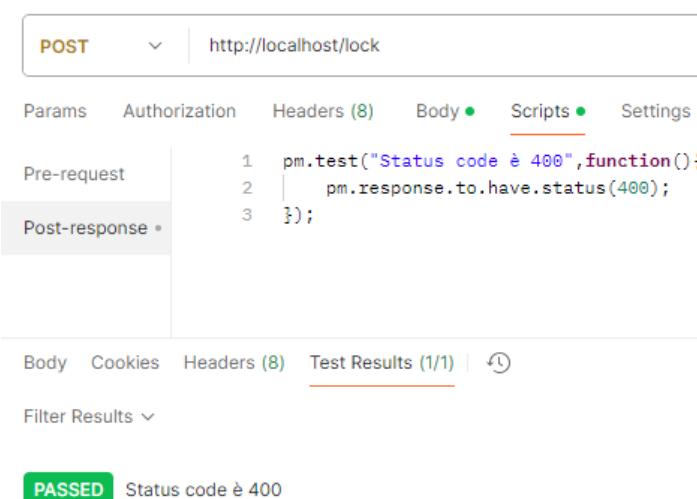
POST http://localhost/lock

Params Authorization Headers (8) Body Scripts Settings

Body (1/1)

{} JSON ▶ Preview ⚡ Visualize

```
1 {  
2   "message": "Error, path not valid",  
3   "data": []  
4 }
```



POST http://localhost/lock

Params Authorization Headers (8) Body Scripts Settings

Pre-request

```
1 pm.test("Status code è 400",function(){  
2   pm.response.to.have.status(400);  
3 });
```

Post-response

Body Cookies Headers (8) Test Results (1/1) ⚡

Filter Results

PASSED Status code è 400

7.2.11.3. Test 3

Body: path: "/Volume/Root/Classes/Cal"

Precondizione: il chiamante non possiede il JWT

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT.

Risposta attesa: 401

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost/lock
- Body:** form-data (selected)
 - path: /Volume/Root/Classes/Cal
 - Key: Value
- Test Results (1/1):**

```
1 {  
2   "message": "Error, token not valid",  
3   "data": []  
4 }
```
- Script:** pm.test("Status code è 401", function(){
 pm.response.to.have.status(401);
});
- Status:** PASSED Status code è 401

7.2.12. POST/unlock

Descrizione: effettua l'operazione di unlock sul path specificato

Metodo: POST

7.2.12.1. Test 1

Body: path: "/Volume/Root/ChatGPTTest.zip"

Precondizione: nel filesystem condiviso è presente il path /Volume/Root/ChatGPTTest.zip e qualcuno ha già bloccato la risorsa in lettura.

Postcondizione: la chiamata restituisce esito positivo e il percorso viene sbloccato per la lettura.

Risposta attesa: 200

The screenshot shows the Postman interface with two requests. The top request is a POST to `http://localhost/unlock` with a JSON body containing a message key with the value "Path unlocked". The bottom request is also a POST to `http://localhost/unlock`, but it includes a test script in the 'Post-response' tab that checks if the status code is 200. The test result is PASSED with the message "Status code è 200".

```
1 {  
2   "message": "Path unlocked",  
3   "data": []  
4 }
```

```
1 pm.test("Status code è 200",function(){  
2   | pm.response.to.have.status(200);  
3 });
```

PASSED Status code è 200

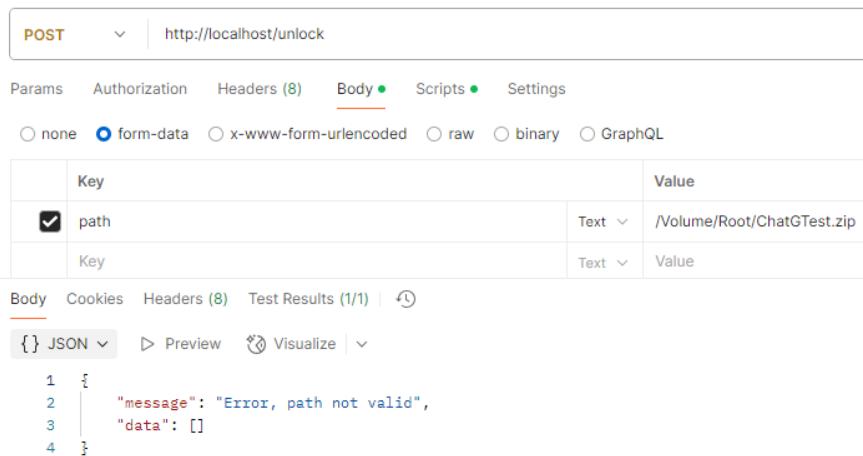
7.2.12.2. Test 2

Body: path: "/Volume/Root/ChatGTest.zip"

Precondizione: nel filesystem condiviso non è presente il path /Volume/Root/ChatGTest.zip e nessun altro ha bloccato la risorsa per la scrittura.

Postcondizione: la chiamata restituisce errore in quanto il path fornito non è esiste.

Risposta attesa: 400



POST http://localhost/unlock

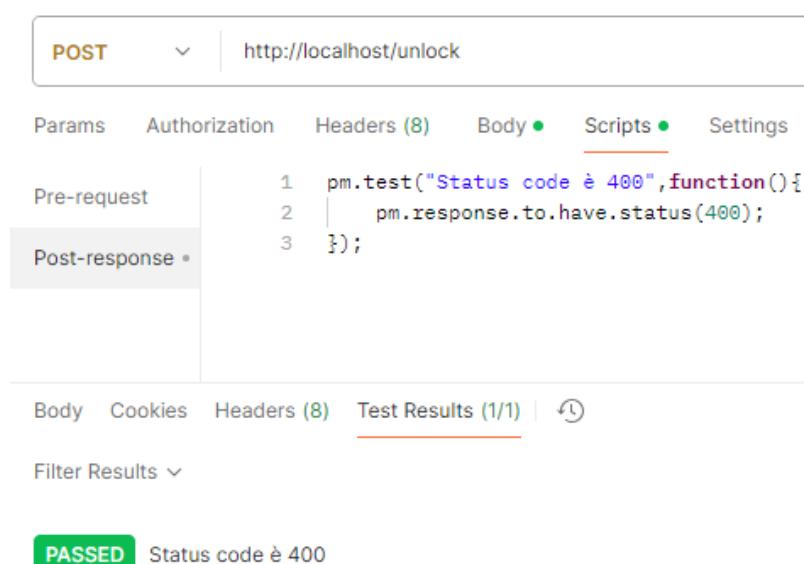
Params Authorization Headers (8) Body Scripts Settings

Body (1/1)

Key	Value
path	/Volume/Root/ChatGTest.zip
Key	Value

{ } JSON Preview Visualize

```
1 {  
2   "message": "Error, path not valid",  
3   "data": []  
4 }
```



POST http://localhost/unlock

Params Authorization Headers (8) Body Scripts Settings

Pre-request

```
1 pm.test("Status code è 400",function(){  
2   pm.response.to.have.status(400);  
3 });
```

Post-response

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 400

7.2.12.3. Test 3

Body: path: "/Volume/Root/Classes/Cal"

Precondizione: il chiamante non possiede il JWT

Postcondizione: la chiamata restituisce un errore data l'assenza di validità del token JWT.

Risposta Attesa: 401

The screenshot shows the Postman interface with a POST request to `http://localhost/unlock`. The 'Body' tab is selected, showing a form-data structure with a key 'path' and a value '/Volume/Root/Classes/Cal'. Other options like 'x-www-form-urlencoded', 'raw', and 'binary' are available but not selected.

The screenshot shows the Postman interface after the request has been made. The 'Body' tab is selected, displaying the JSON response:

```
1 {  
2   "message": "Error, token not valid",  
3   "data": []  
4 }
```

The screenshot shows the Postman interface with a POST request to `http://localhost/unlock`. The 'Body' tab is selected. In the 'Post-response' tab, there is a script block containing:

```
1 pm.test("Status code è 401",function(){  
2   pm.response.to.have.status(401);  
3 });
```

 This script is used to assert that the status code is 401.

Body Cookies Headers (8) Test Results (1/1)

Filter Results

PASSED Status code è 401

7.3. Testing sulla concorrenza

Per i test di concorrenza abbiamo usato il software Ngrok per rendere accessibile un'istanza del gioco in rete in modo da potere accedere contemporaneamente al gioco.

ID	Descrizione	Precondizione	Input	Output atteso	Post condizione attesa	Output ottenuto	Esito
TC1	Caricare due classi con lo stesso nome in contemporanea dalla GUI Verificare che solo una classe venga caricata	Gli Admin hanno effettuato il login e si trovano nella pagina Add Class and Test.	Scrivere il nome della classe: "Calcolatrice". Aggiungere il file.java. Cliccare sul tasto "Upload class and tests". Questo per entrambi gli Admin.	Visualizzazione di un popup con scritto "La classe è stata aggiunta correttamente" per un Admin. Visualizzazione di un popup con scritto "Errore, classe già presente" per l'altro.	Solo una delle classi viene caricata nel sistema.	Visualizzazione di un popup con scritto "La classe è stata aggiunta correttamente" per un Admin. Visualizzazione di un popup con scritto "Errore, classe già presente" per l'altro.	PASS
TC2	Eliminare la stessa classe dalla GUI Verificare che la classe venga eliminata correttamente.	Gli Admin hanno effettuato il login e si trovano nella pagina di visualizzazione delle classi	Click sul tasto di delete della classe. Questo per entrambi gli admin.	Refresh della pagina per l'Admin che ha cancellato correttamente e pagina bloccata per l'altro.	Eliminazione della classe.	Refresh della pagina per l'Admin che ha cancellato correttamente e pagina bloccata per l'altro.	PASS
TC3	Blocco per la lettura dal FileSystem Verificare che non si possa modificare un file bloccato per la lettura.	Il path passato deve esistere, deve essere valido e non deve essere già bloccato per una scrittura.	Richiesta HTTP di lock della risorsa. Richiesta HTTP di delete prima dello sblocco. Richiesta HTTP di unlock della risorsa.	L'operazione di delete si deve arrestare fino allo sblocco della risorsa.	La richiesta di delete rimane bloccata.	La richiesta di delete rimane bloccata e esegue solo dopo lo sblocco della risorsa.	PASS
TC4	Blocco per la lettura del FileSystem con path errato Verificare che se il path in ingresso è sbagliato, allora il lock non avviene.	Il path passato deve essere non valido.	Richiesta HTTP di lock della risorsa con un path inesistente o invalido.	Risposta di errore da inviare al richiedente.	Nessun lock acquisito.	Nessun lock acquisito e risposta di errore inviata.	PASS

TC5	Sblocco della lettura in seguito al TIMEOUT	Il lock è stato acquisito precedentemente. Verificare che il meccanismo di TIMEOUT per lo sblocco della lettura funzioni correttamente	Nessuno	Lock rilasciato per la risorsa dopo il TIMEOUT.	Lock rilasciato per la risorsa dopo il TIMEOUT.	Lock rilasciato per la risorsa dopo il TIMEOUT.	PASS
-----	--	---	---------	--	---	---	-------------

8. Sviluppi futuri

Durante il lavoro sull'applicazione, il gruppo ha individuato alcuni miglioramenti da apportare:

- **Refactoring per una migliore divisione delle responsabilità:** ad esempio, creare Controller specifici in base all'ambito delle richieste da gestire. Durante lo sviluppo è stato introdotto *ApiController* (dedicato esclusivamente alle API) anziché inserirle in *HomeController*, che attualmente si occupa di tutte le richieste al T1. Lo stesso principio può essere applicato anche ad alcuni Service, come *AdminService*;
- **Rintracciare e aggiornare l'utilizzo delle vecchie API:** identificare nei microservizi esistenti dove vengono utilizzate le vecchie API e sostituirle con le nuove, completando così la migrazione all'uso del nuovo file system;
- **Valutare la sostituzione del database:** potrebbe essere opportuno migrare da *MongoDB* a un database relazionale (es. *MySQL*).