

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 – M63000000
Fabrizio Imparato – M63001758
Tommaso Rivieccio – M63000000

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 sviluppo
nella seguente repo di GitHub: [A13-R6-B15](#)

Indice

1.	Punto di partenza e obiettivi del progetto	5
1.1.	Versione di partenza.....	5
1.2.	Nuovi requisiti assegnati	9
1.3.	Processo di sviluppo.....	9
1.4.	Strumenti utilizzati	11
2.	Analisi dei Requisiti	13
2.1.	Diagramma dei casi d'uso.....	13
2.2.	Diagramma di sequenza	16
3.	Analisi d'impatto	17
4.	Progettazione della soluzione per realizzare i requisiti richiesti	18
5.	Implementazione.....	21
5.1.	Modifiche ai file di installazione e configurazione.....	21
5.2.	Modifiche ai file e ai package	22
5.2.1.	Model	23
5.2.2.	Responses.....	23
5.2.3.	Service.....	23
5.2.4.	Controller	27
5.2.5.	Modifiche alle pagine.....	29
5.2.6.	UI Gateway	30
5.3.	Aggiunta di API.....	31
6.	Architettura finale	32
7.	Testing	35
7.1.	Testing GUI.....	35
7.2.	Testing API	39
7.2.1.	GET /classes.....	39
7.2.1.1.	Test 1	39
7.2.1.2.	Test 2	40
7.2.1.3.	Test 3	41
7.2.2.	GET/classes/{className}	42
7.2.2.1.	Test 1	42
7.2.2.2.	Test 2	43
7.2.2.3.	Test 3	44
7.2.3.	GET/classes/{className}/robots	45
7.2.3.1.	Test 1	45
7.2.3.2.	Test 2	46
7.2.3.3.	Test 3	47

7.2.3.4. Test 4	48
7.2.4. GET/classes/{className}/{robotName}	49
7.2.4.1. Test 1	49
7.2.4.2. Test 2	50
7.2.4.3. Test 3	51
7.2.4.4. Test 4	52
7.2.5. POST/classes/{className}	53
7.2.5.1. Test 1	53
7.2.5.2. Test 2	54
7.2.5.3. Test 3	55
7.2.6. POST/classes/{className}/{robotName}	56
7.2.6.1. Test 1	56
7.2.6.2. Test 2	57
7.2.6.3. Test 3	58
7.2.6.4. Test 4	59
7.2.6.5. Test 5	60
7.2.7. DELETE/classes/{className}	61
7.2.7.1. Test 1	61
7.2.7.2. Test 2	62
7.2.7.3. Test 3	63
7.2.8. DELETE/classes/{className}/{robotName}	64
7.2.8.1. Test 1	64
7.2.8.2. Test 2	65
7.2.8.3. Test 3	66
7.2.8.4. Test 4	67
8. Sviluppi futuri	68

1. Punto di partenza e obiettivi del progetto

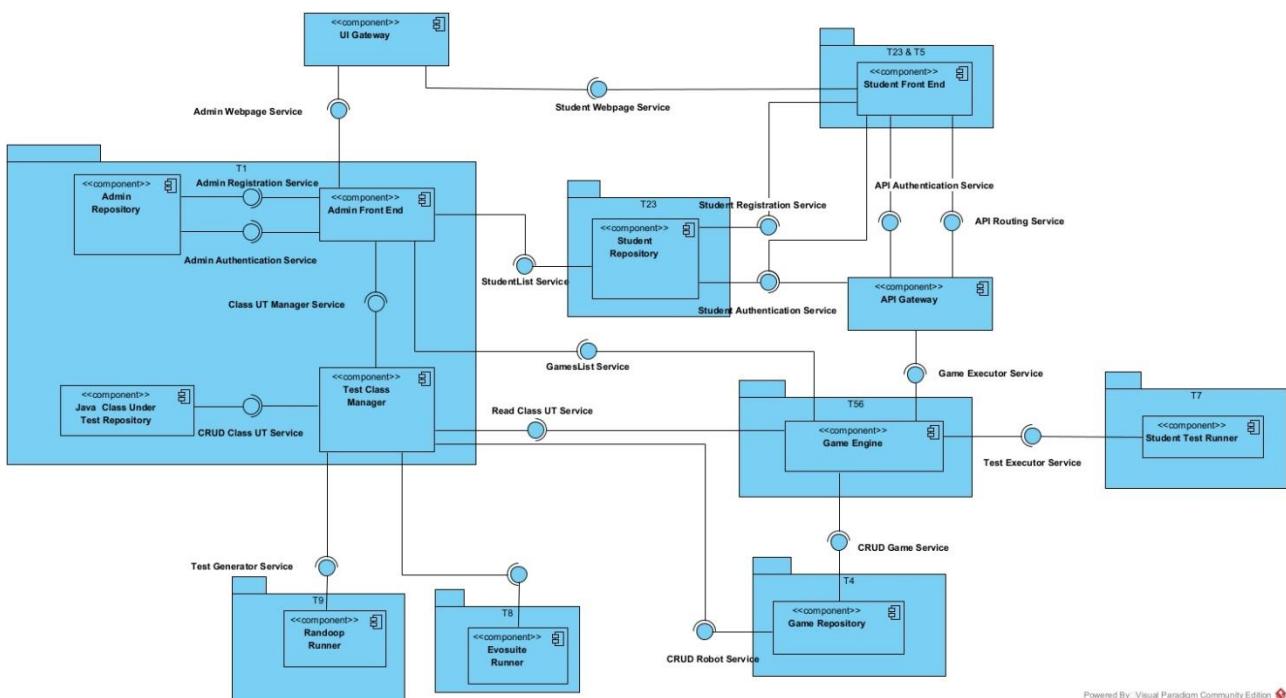
Il lavoro si colloca all'interno del progetto ERASMUS denominato **ENACTEST** (**E**uropean **i**nno**v**ative **A**llian**C**e for **T**ESTing **e**du**T**ion), sviluppato in parte dagli studenti del corso di Software Architecture Design dell'Università degli Studi di Napoli Federico II. Questo progetto 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 ([qui](#) per raggiungere la repository) adotta un pattern architettonicale 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.



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

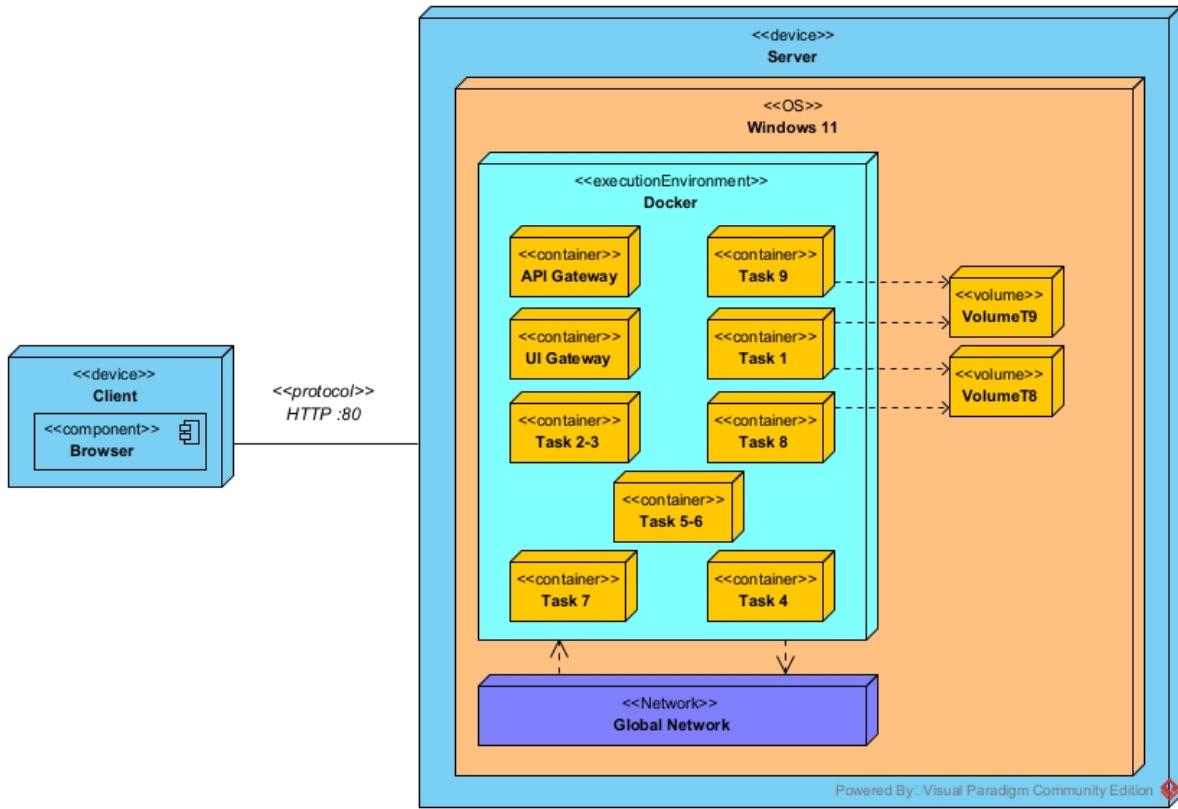


Figura 2: Deployment Diagram della versione iniziale

Come si può notare, l'architettura è composta da nove container contenuti in 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.

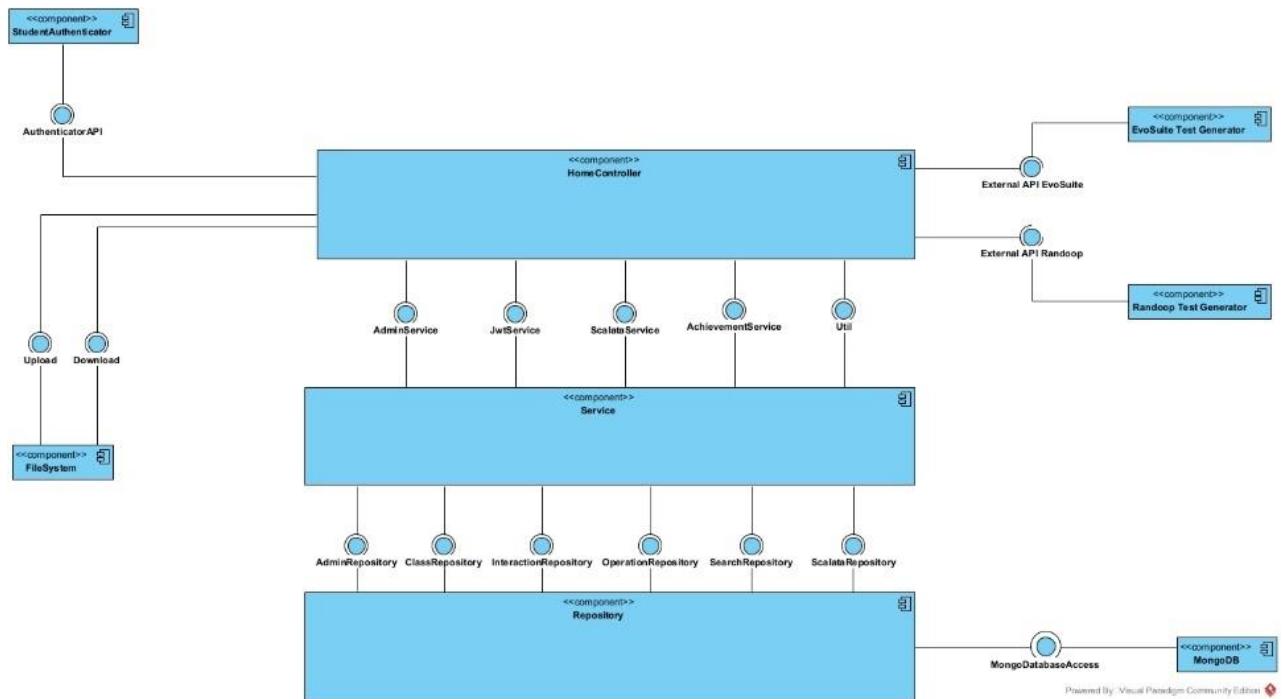


Figura 3: Component Diagram iniziale del container T1

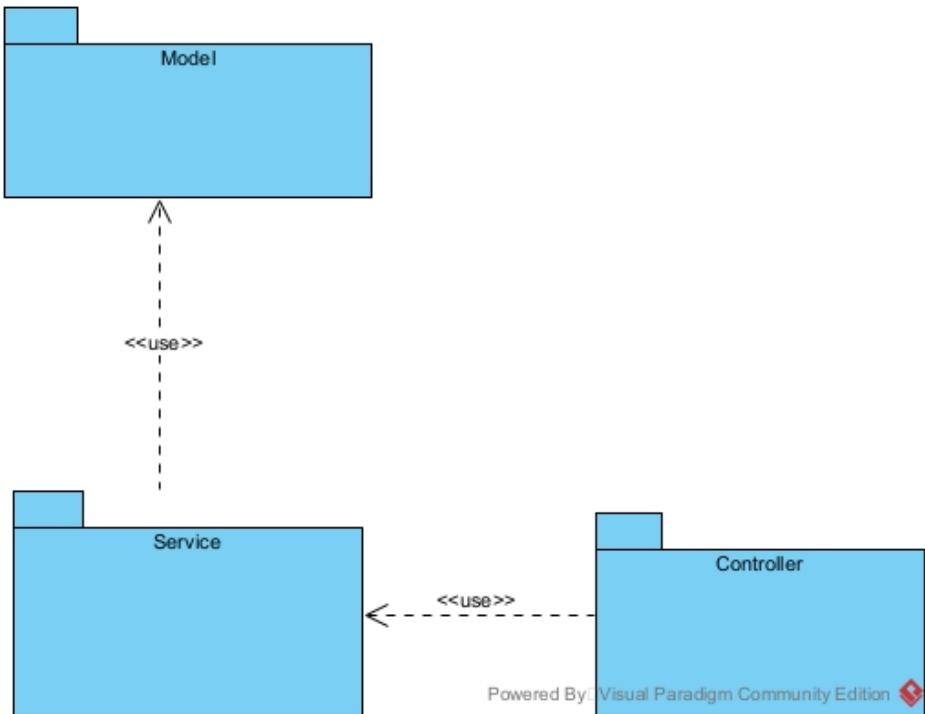


Figura 4: Package Diagram iniziale del container T1

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:

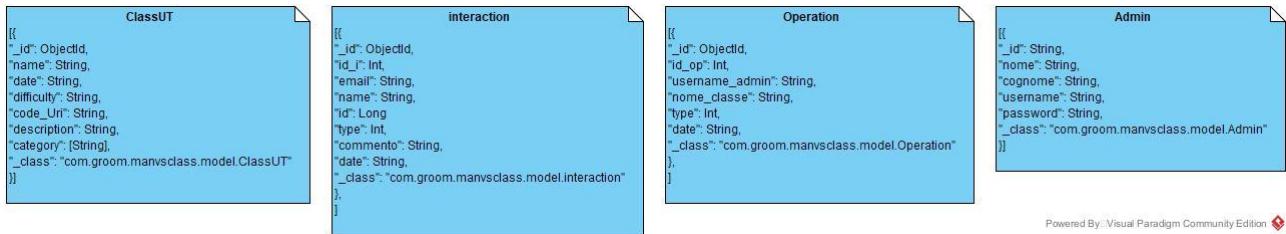


Figura 5: Collection iniziali presenti del database MongoDB

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** contains 10 items with various priority levels (2, 8, 5, 8, 8, 3) and assigned team members (VG, TR, FG). In the center, the **Current Sprint Backlog** lists three tasks: 1. Create the shared volume between containers (Priority 1, assigned to VG), 2. Modify the DB structure to support generalization (Priority 2, assigned to FG), and 3. Modify the "AddClassAndTest" page (Priority 5, assigned to TR). To the right, three completed sprints are shown: **Sprint #1 Complete** (1 task), **Sprint #2 Complete** (1 task), and **Sprint #3 Complete** (1 task). Each completed sprint has a "Add Scheda" button.

Sprint #2

The screenshot shows the SAD - Task R6 application interface. The **Product Backlog** now contains 7 items. The **Current Sprint Backlog** lists three tasks: 1. Create API for file system interaction (Priority 2, assigned to TR and FG), 2. Modify delete logic (Priority 8, assigned to TR), and 3. Modify save logic (Priority 8, assigned to FG). The completed sprints remain the same as in Sprint #1: **Sprint #1 Complete**, **Sprint #2 Complete**, and **Sprint #3 Complete**. Each completed sprint has a "Add Scheda" button.

Sprint #3

Product Backlog	Current Sprint Backlog	Sprint #1 Complete	Sprint #2 Complete	Sprint #3 Complete
+ Aggiungi una scheda	8 Creare API per l'interazione con il file system condiviso. 3 Pulizia del codice e riorganizzazione dei package in modo lineare. + Aggiungi una scheda	1 Creare il volume condiviso tra i container. 2 Modificare la struttura del DB per permettere la generalizzazione dei Robot. 5 Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. + Aggiungi una scheda	2 Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili. 8 Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. 8 Modificare la logica di salvataggio delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. + Aggiungi una scheda	3 Pulizia del codice e riorganizzazione dei package in modo lineare. 8 Creare API per l'interazione con il file system condiviso. + Aggiungi una scheda

Situazione finale

Product Backlog	Current Sprint Backlog	Sprint #1 Complete	Sprint #2 Complete	Sprint #3 Complete
+ Aggiungi una scheda	+ Aggiungi una scheda	1 Creare il volume condiviso tra i container. 2 Modificare la struttura del DB per permettere la generalizzazione dei Robot. 5 Modificare la pagina di "AddClassAndTest" per permettere la scelta di più robot. + Aggiungi una scheda	2 Aggiunta della logica di gestione di un file.txt per la configurazione dei robot disponibili. 8 Modificare la logica di delete delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. 8 Modificare la logica di salvataggio delle classi e dei test per aggiungere la nuova logica e preservare la vecchia. + Aggiungi una scheda	3 Pulizia del codice e riorganizzazione dei package in modo lineare. 8 Creare API per l'interazione con il file system condiviso. + Aggiungi una scheda

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.

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.

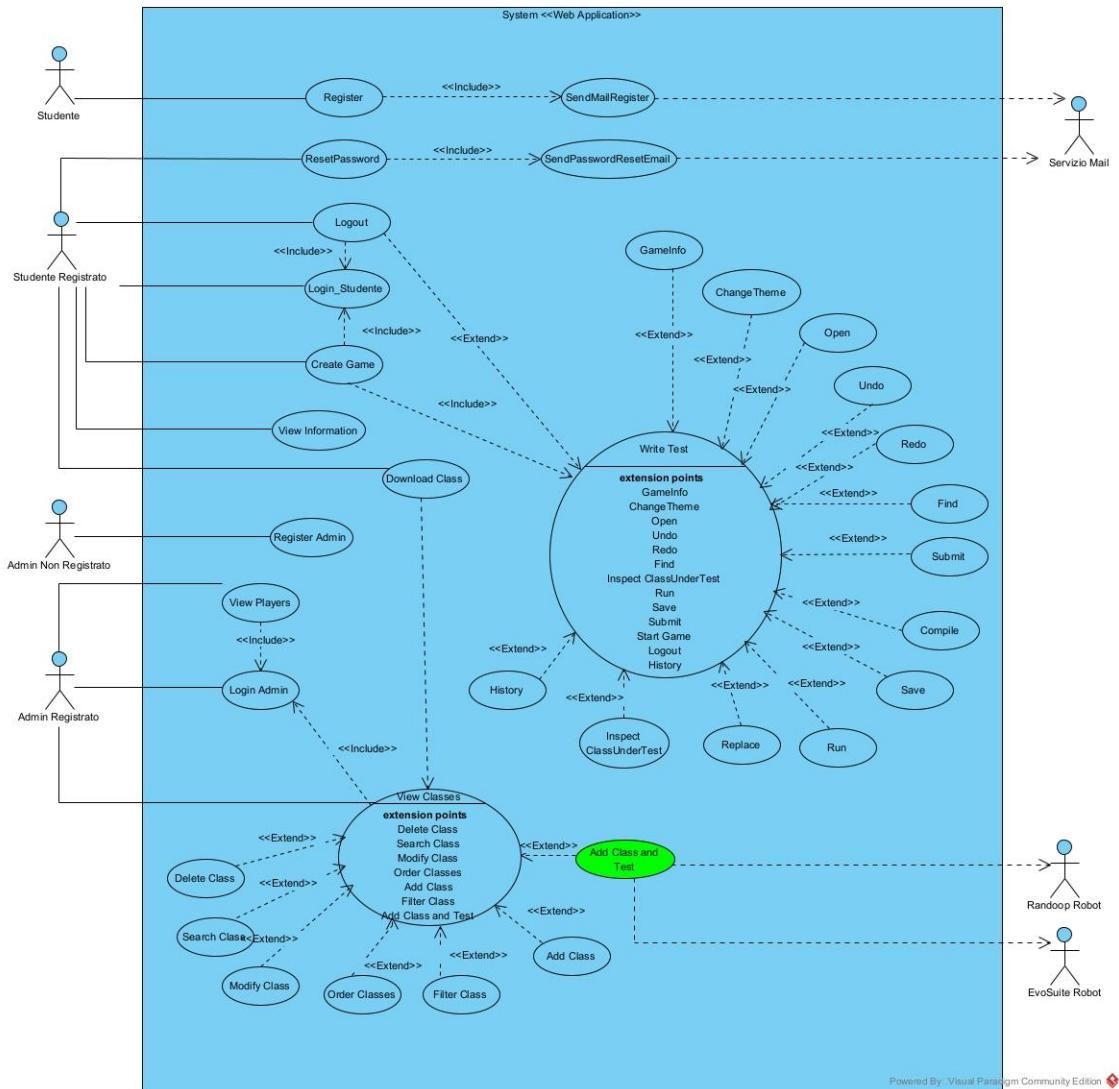


Figura 6: Diagramma dei casi d'uso completo

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):

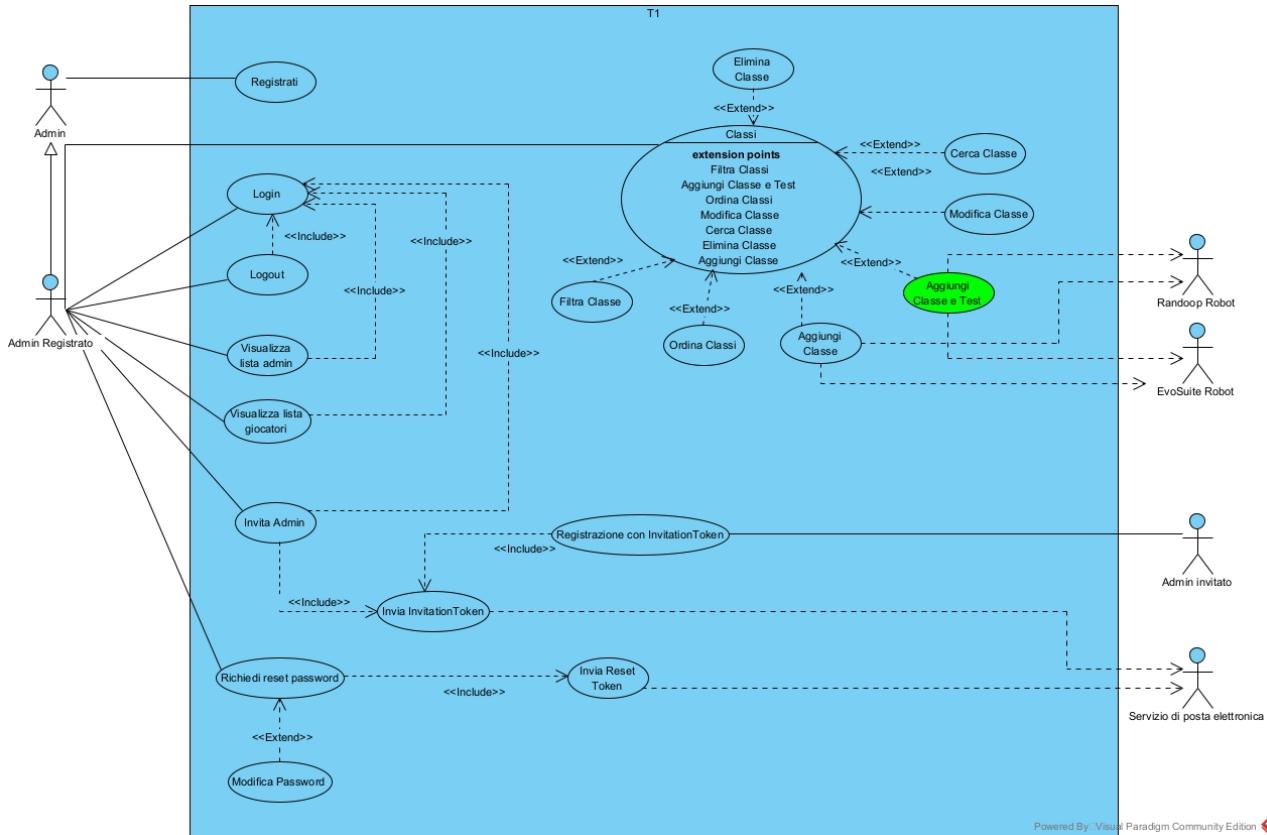


Figura 7: Diagramma dei casi d'uso dettagliato del container T1

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.

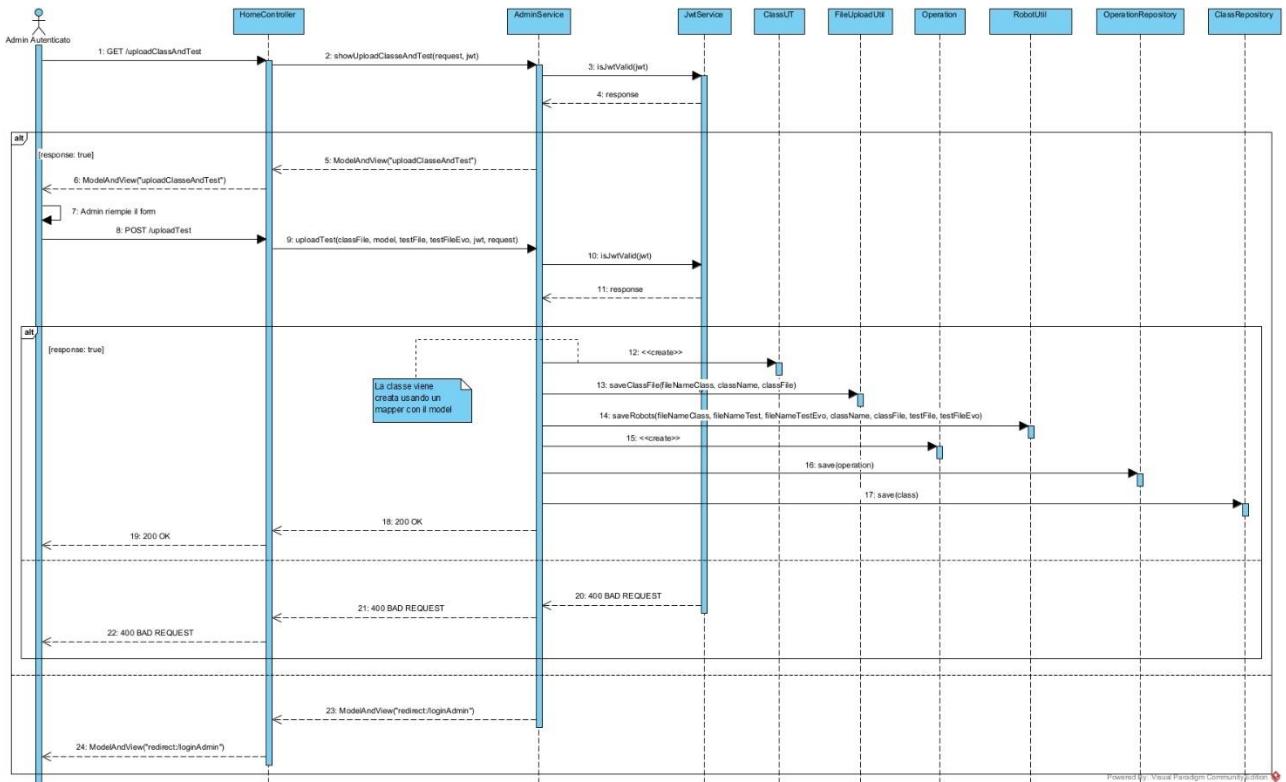


Figura 8: Sequence Diagram iniziale della funzionalità “Add Class and Test”

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:

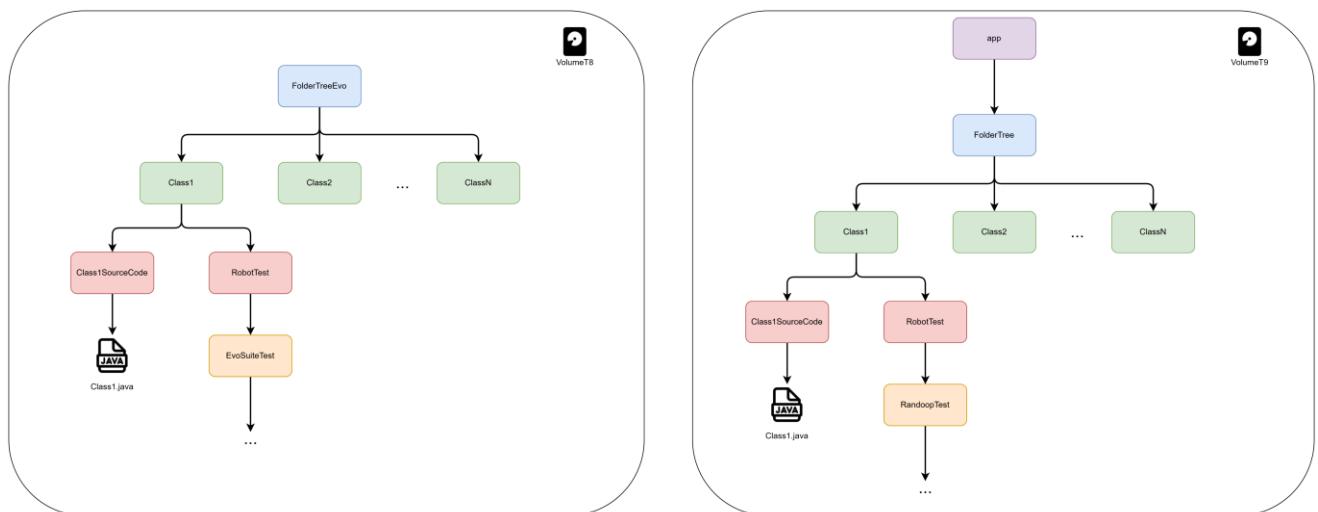


Figura 9: File system presente in VolumeT8 (destra) e VolumeT9 (sinistra)

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 input field and a calendar icon), 'Difficulty' (set to 'Beginner'), 'Description' (a large text area), and three categories ('Category 1', 'Category 2', 'Category 3') each with a text input field. Below these are sections for 'Robot' (with a dropdown menu set to 'Select a robot') and 'Upload your test (.zip)' (with a file selection input field). At the bottom are two buttons: 'Upload class and tests' (green) and 'Go Back'.

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:

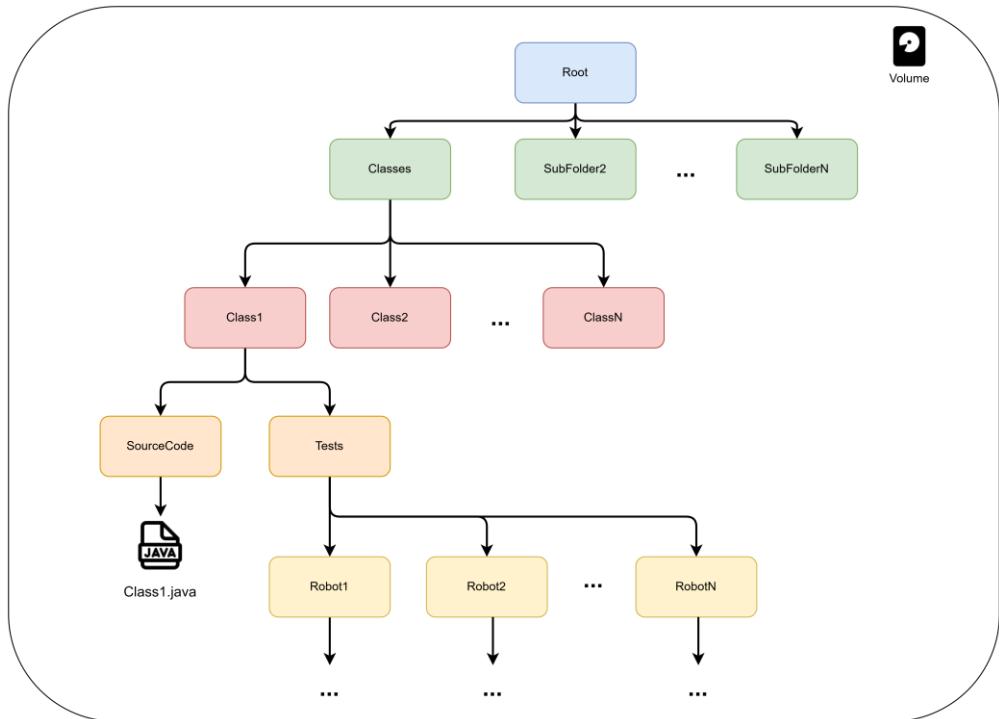


Figura 11: Nuovo file system presente nel volume condiviso

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:

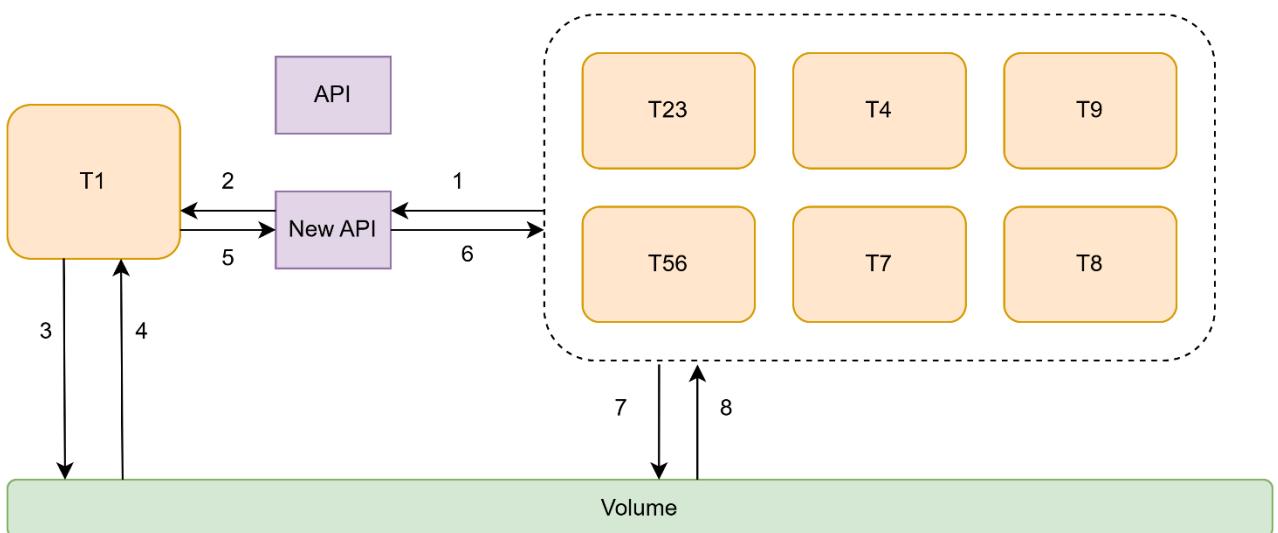


Figura 12: Flusso operativo previsto

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 *ClassUT.java*. Questo 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 FileSystem e viene infatti anche usato da ClassService.
- *ApiService*, 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)
```

- **createFolder()**, che si occupa di creare una cartella e tutte le cartelle precedenti specificate dal percorso. Ne esistono due versioni una che prende in ingresso un oggetto String ed un’altra un oggetto Path;

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

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

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

- **deleteDirectory()**, che si occupa di eliminare un file o una cartella (compreso il contenuto) di un percorso specificato;

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

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, ma in futuro potrebbero servire anche ad altri componenti nel caso di espansione del FileSystem.

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
```

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-statments* 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
```

5.2.5. Modifiche alle pagine

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 the 'Add Class and Tests' page. At the top, there is a file upload input labeled 'Upload your class (.java)' with the value 'Calcolatrice.java'. Below it is a dropdown menu labeled 'Robot' containing 'Randoop'. A placeholder text 'Select the robot you wish to upload the test' is visible. Another file upload input labeled 'Upload your test (.zip)' with the value 'RandoopTest.zip' is shown. The bottom section features a dropdown menu labeled 'Robot' with 'ChatGPT' selected. A sub-menu titled 'Select a robot' lists 'ChatGPT', 'Gemini', and 'EvoSuite'. Below this is another file upload input labeled 'Upload your test (.zip)' with the value 'EvoSuiteTest.zip'. At the bottom left is a button with a plus sign '+', and at the bottom right are two buttons: 'Upload class and tests' (green background) 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.

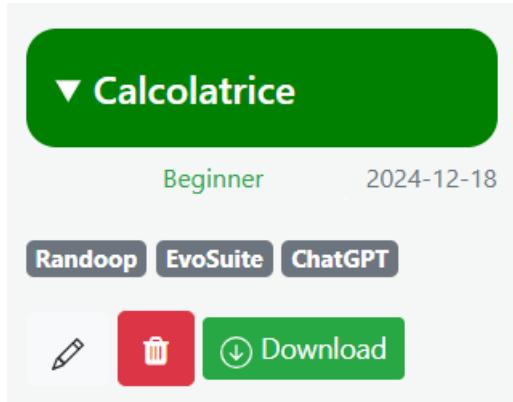


Figura 14: Aggiunta dei tag dei robot associati ad una classe

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.

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

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:

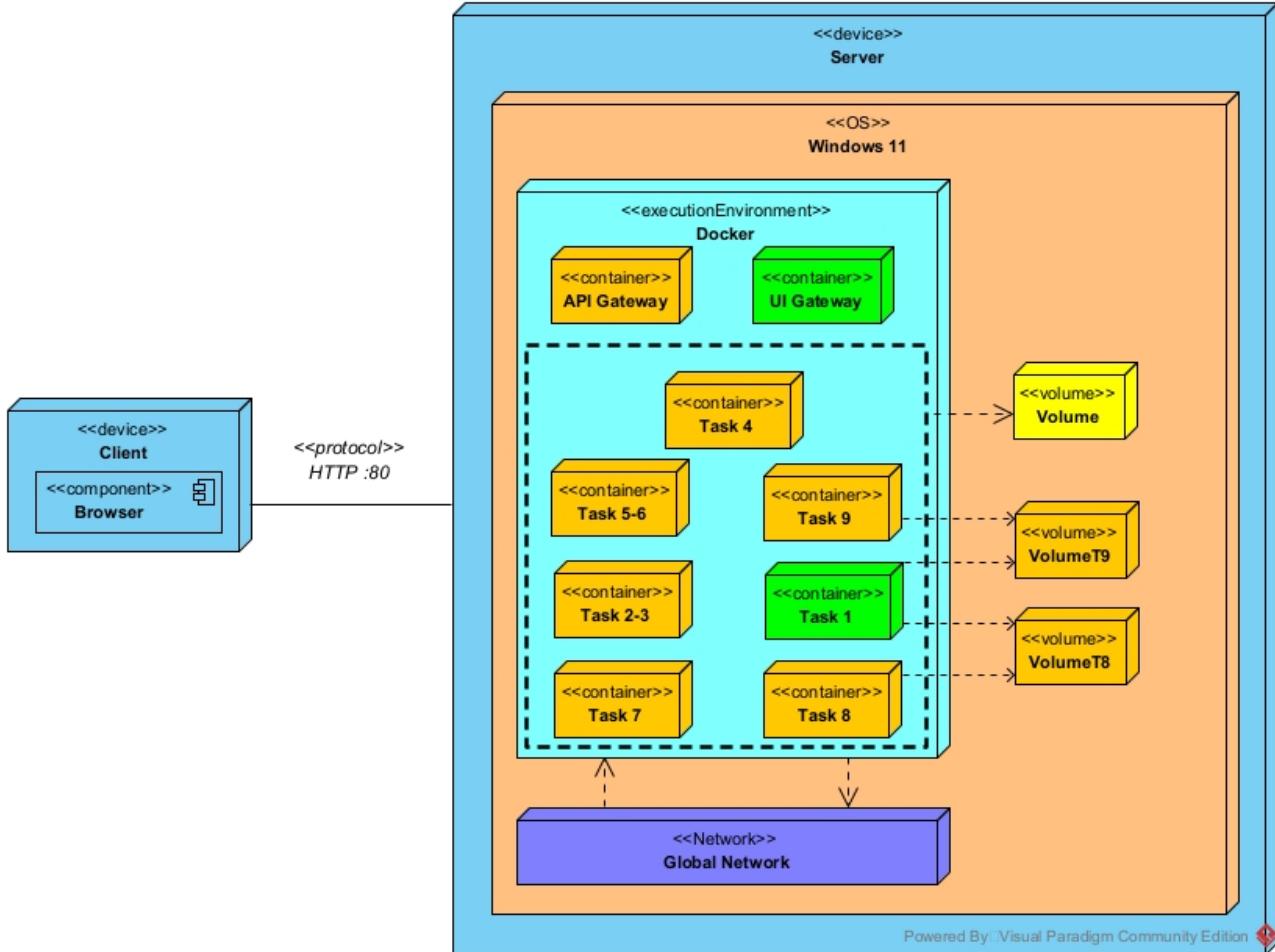


Figura 15: Deployment Diagram finale

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:

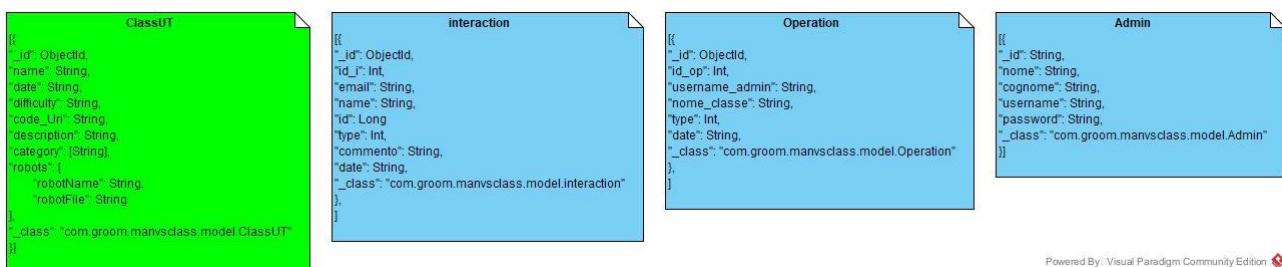


Figura 16: Collection di MongoDB aggiornate

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

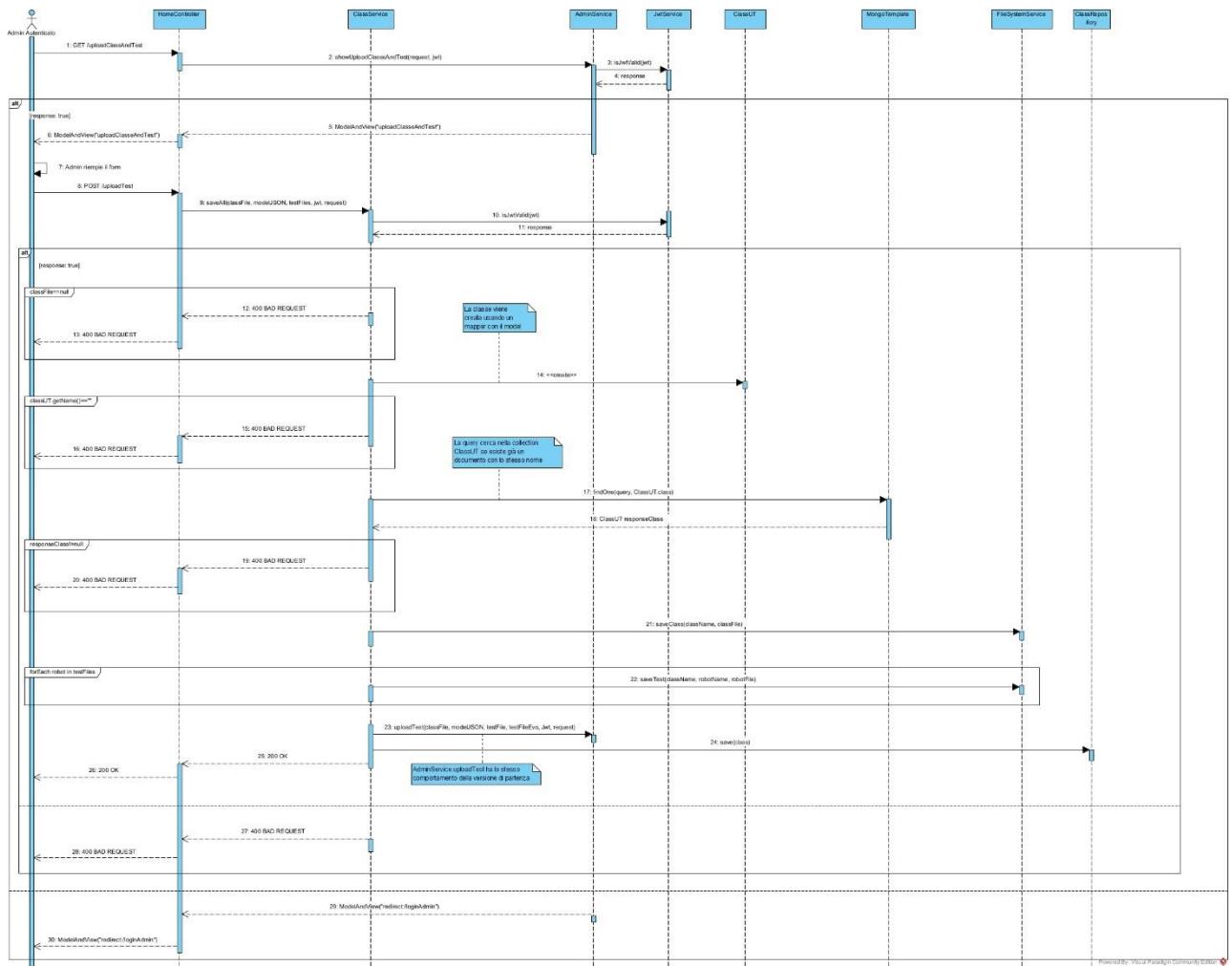


Figura 17: Sequence Diagram finale della funzionalità Add Class and Test

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 aggiunti e in verde i modificati):

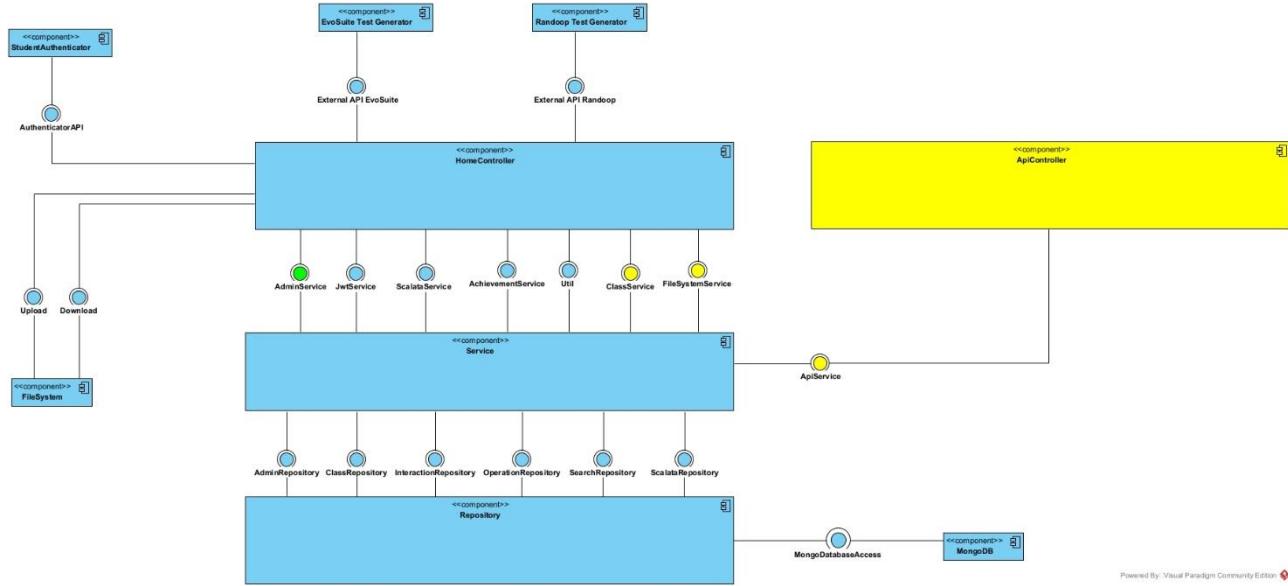


Figura 18: Component Diagram di T1

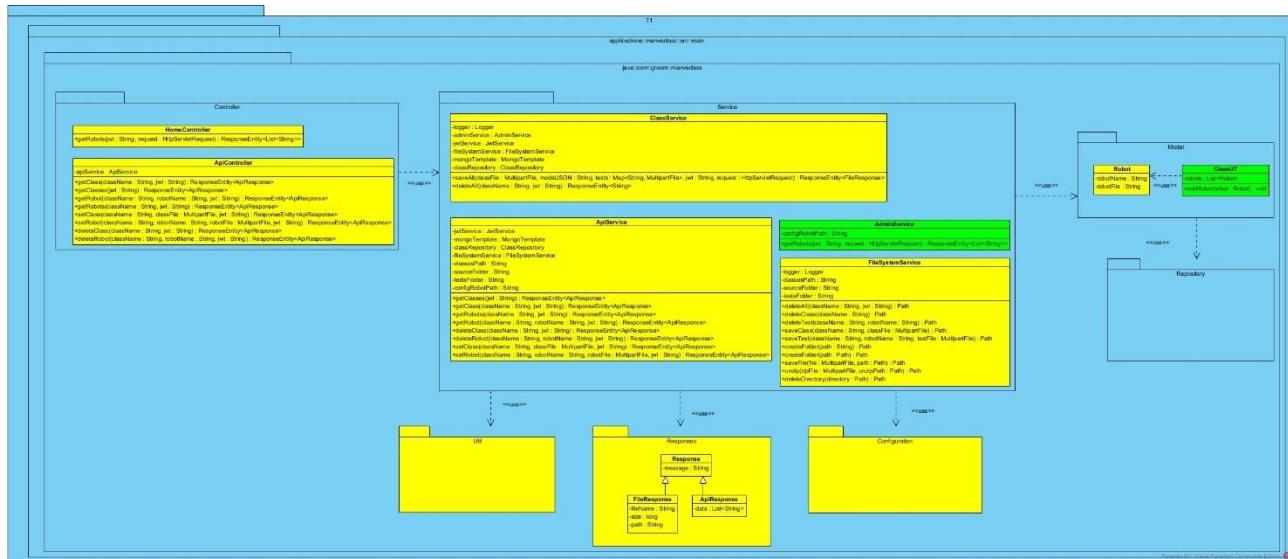


Figura 19: Package Class completo

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	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	Visualizzazione di un popup con scritto “La classe è stata aggiunta correttamente”.	PASS

	due robot.				vengono caricate la classe 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...

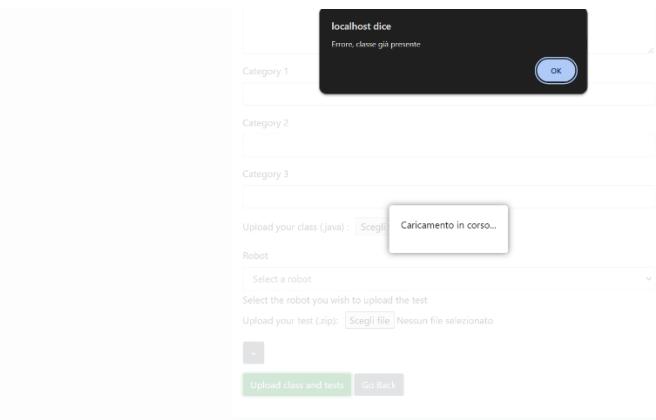
Robot

Randoop

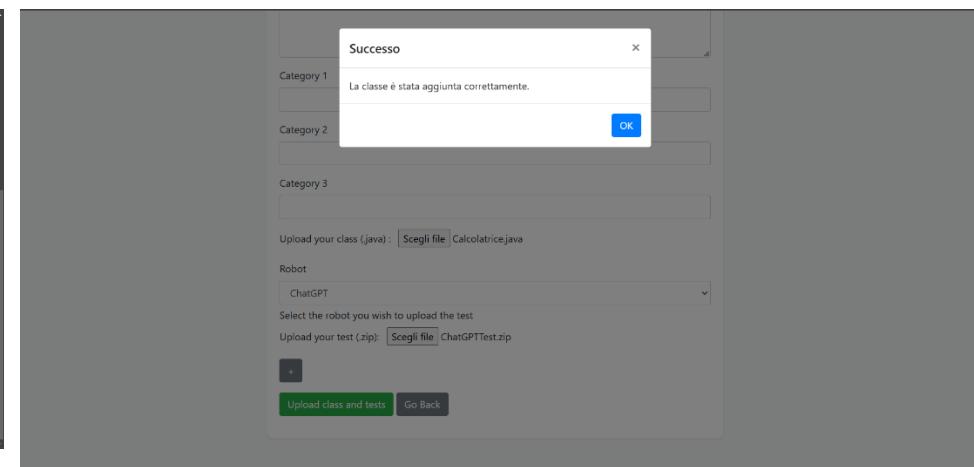
Select the robot you wish to upload the test

Upload your test (.zip) RandoopTest.zip

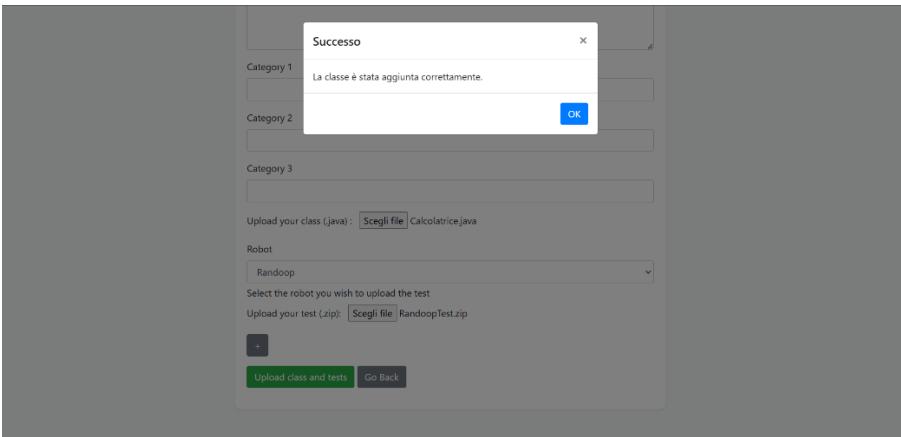
TC2



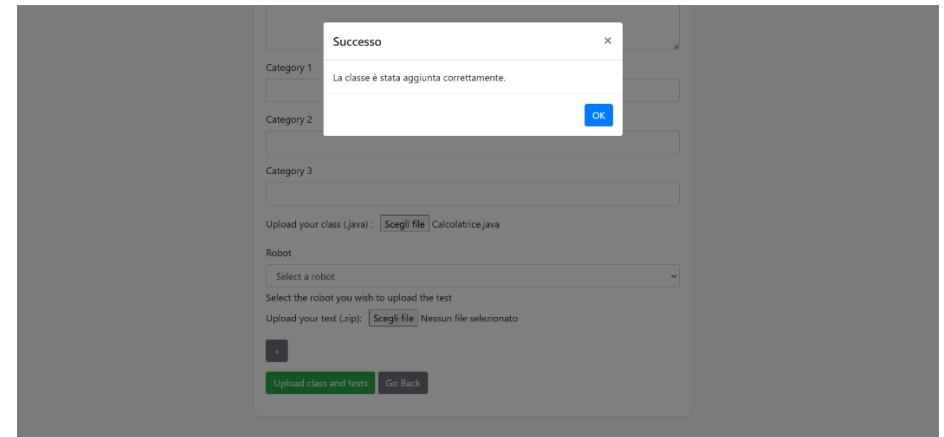
TC3



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

Post condizione: la chiamata restituisce Calcolatrice e FontInfo

Risposta Attesa: 200

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

pm.test("Output Atteso: Classi presenti", function(){
  var data = pm.response.json();
  pm.expect(data.data).to.eql(["Calcolatrice","FontInfo"]);
});
```

The screenshot shows the Postman interface for a GET request to http://localhost/classes. The 'Post-response' tab is selected, containing a JavaScript test script. The script first checks if the status code is 200, then extracts the JSON response body and compares its 'data' array against the expected values ['Calcolatrice', 'FontInfo']. The 'Test Results' section shows 2/2 tests passed.

The screenshot shows the Postman interface for the same test. The 'Test Results' section indicates 2/2 tests passed. The results are summarized as follows:

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

7.2.1.2. Test 2

Body:

Precondizione: nel sistema non c'è alcuna classe

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

Risposta Attesa: 404

The screenshot shows the Postman interface with a test script for a 404 error response. The URL is set to `http://localhost/classes`. The test script consists of two parts: Pre-request and Post-response. The Post-response part contains 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: Classi non presenti",function(){
6     var data = pm.response.json();
7     pm.expect(data.data).to.eql([]);
8 });
9
```

The Body tab shows the JSON response:

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

At the bottom, the results show two green "PASSED" status messages:

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

7.2.1.3. Test 3

Body:

Precondizione: l'utente non ha effettuato l'accesso

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

Risposta Attesa: 401

The screenshot shows the Postman interface for a test configuration. The URL is set to `http://localhost/classes`. The "Scripts" tab is selected, showing a pre-request script and a post-response script. The post-response script contains the following code:

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

The "Body" tab is selected, showing a JSON response with the following structure:

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

The screenshot shows the Postman interface displaying the test results. The URL is again `http://localhost/classes`. The "Test Results" tab is selected, showing one successful result. The result details are as follows:

PASSSED 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

Post condizione: 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. The 'Pre-request' script contains: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 });
9The 'Post-response' script is empty. Below the scripts, the 'Pretty' tab is selected, showing the JSON response:1 {
2 | "message": "Error, class not found",
3 | "data": []
4 }

The screenshot shows the Postman interface with the same setup. The 'Test Results (2/2)' tab is selected, showing 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

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

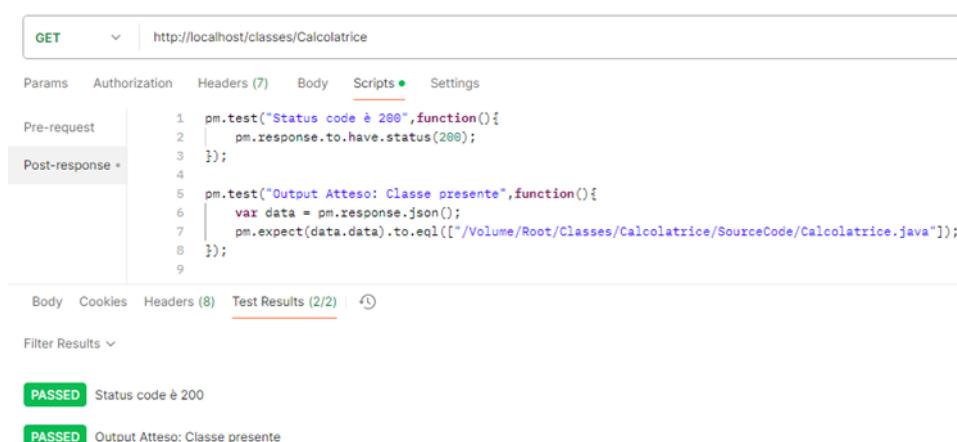
Risposta Attesa: 200



```
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
```

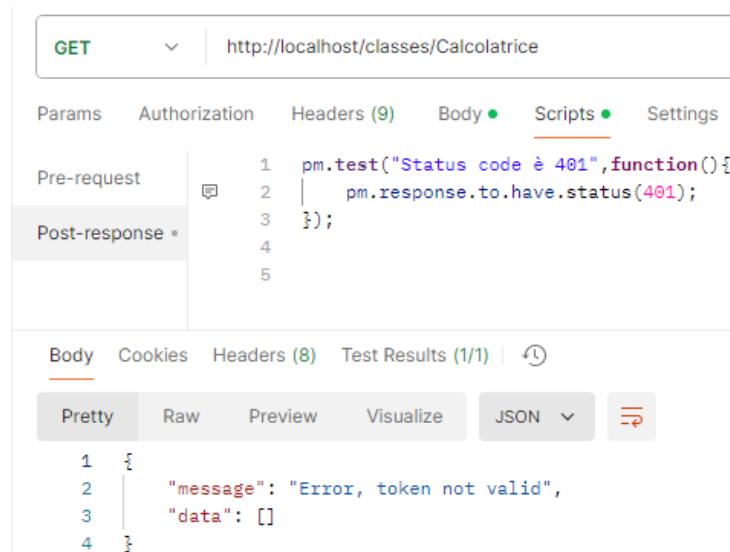
7.2.2.3. Test 3

Body:

Precondizione: l'utente non ha effettuato l'accesso

Post condizione: 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

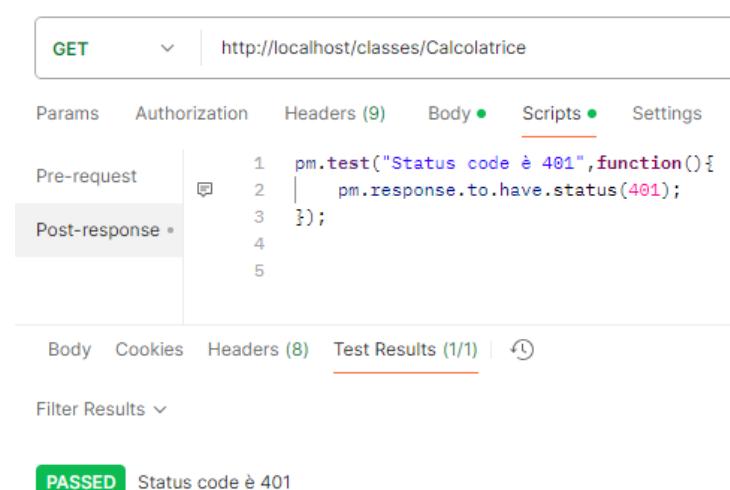
Pre-request | Post-response |

```
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

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

Risposta Attesa: 404

The screenshot shows the Postman test runner interface. A test has failed with the following details:

Test Result: Failed (1 of 1 tests failed)

Assertion: Status code è 404

Test Script:

```
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
```

Test Results: 2/2

Response Body:

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

Test 2 (Passed):

Test Result: Passed (1 of 1 tests passed)

Assertion: Output Atteso: Classe presente

Test Script:

```
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
```

Test Results: 2/2

Filter Results: ▾

7.2.3.2. Test 2

Body:

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

Post condizione: 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
```

Post-response

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
```

Post-response

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

Post condizione: 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

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

Risposta Attesa: 401

The screenshot shows the Postman interface for a test configuration. The request method is GET, and the URL is `http://localhost/classes/Calcolatrice/robots`. The 'Scripts' tab is selected under the 'Settings' section. The 'Post-response' script contains the following code:

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

The 'Body' tab is selected, showing a JSON response with the following structure:

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

Below the body, the 'Test Results' section shows a single result: **PASSED** Status code è 401.

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

Descrizione: restituisce il path del robot associato alla classe specificata

Metodo: GET

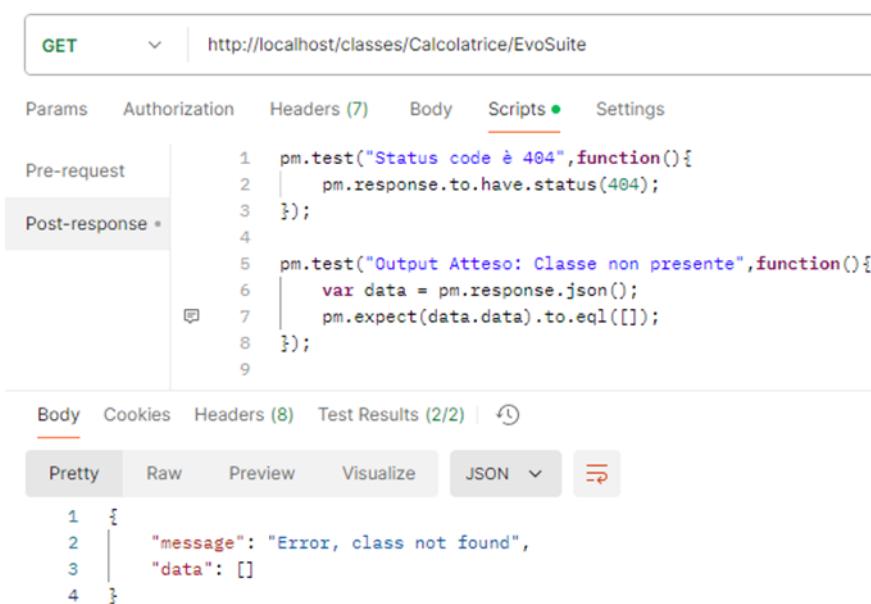
7.2.4.1. Test 1

Body:

Precondizione: nel sistema non è presente nessuna classe

Post condizione: 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

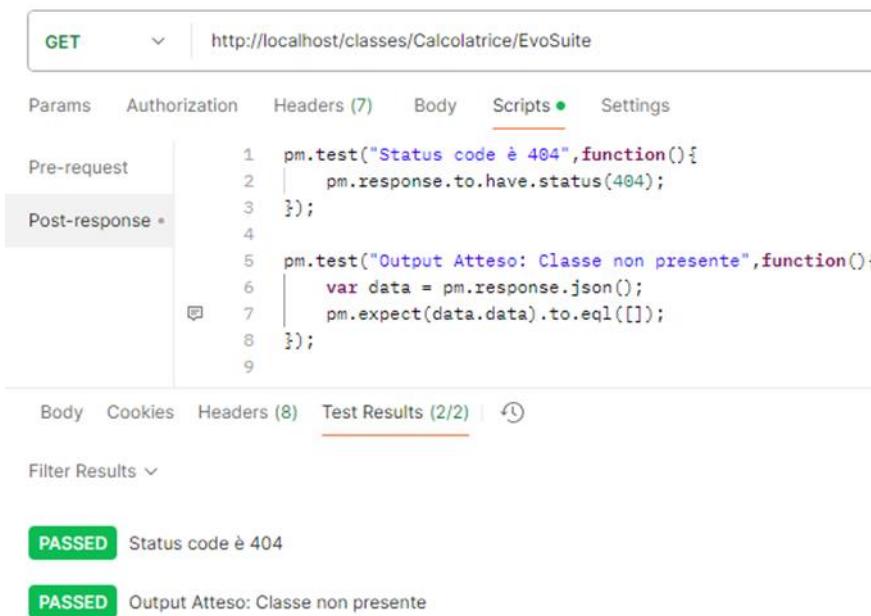
Pre-request Post-response

```
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
```

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 Post-response

```
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
```

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

Post condizione: 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
5 pm.test("Output Atteso: Robot 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, 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
5 pm.test("Output Atteso: Robot 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: Robot non presente

7.2.4.3. Test 3

Body:

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

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

Risposta Attesa: 200

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

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 presente",function(){
6 |   var data = pm.response.json();
7 |   pm.expect(data.data).to.eql(["/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/"]);
8 });
9
```

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

Pretty Raw Preview Visualize JSON

```
1
2   "message": "Robot found",
3   "data": [
4     "/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/"
5   ]
```

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

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 presente",function(){
6 |   var data = pm.response.json();
7 |   pm.expect(data.data).to.eql(["/Volume/Root/Classes/Calcolatrice/Tests/EvoSuite/"]);
8 });
9
```

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

Filter Results ▾

PASSED Status code è 200

PASSED Output Atteso: Robot presente

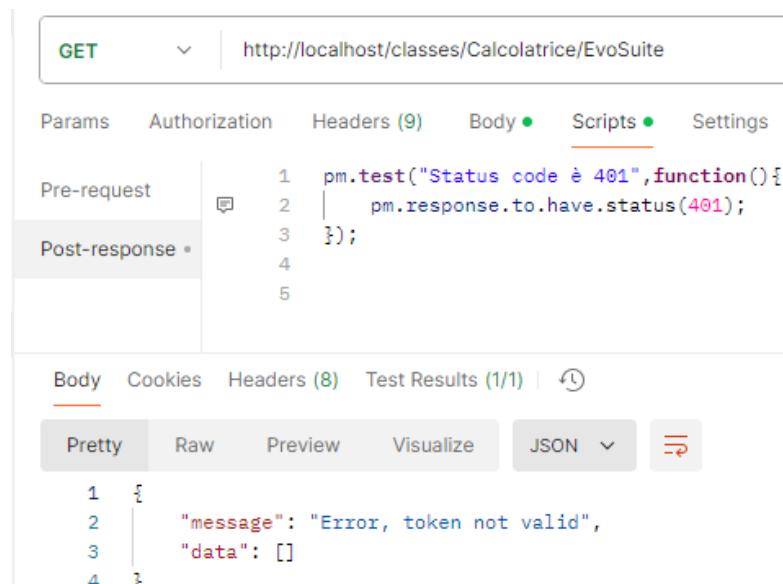
7.2.4.4. Test 4

Body:

Precondizione: l'utente non ha effettuato l'accesso

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

Risposta Attesa: 401



GET

Params Authorization Headers (9) Body Scripts Settings

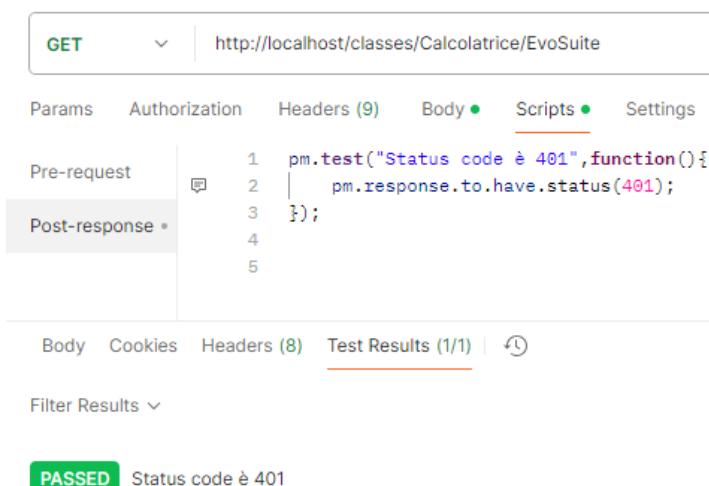
Pre-request Post-response

```
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

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

7.2.5. POST/classes/{className}

Descrizione: esegue l'UPDATE della classe da testare

Metodo: POST

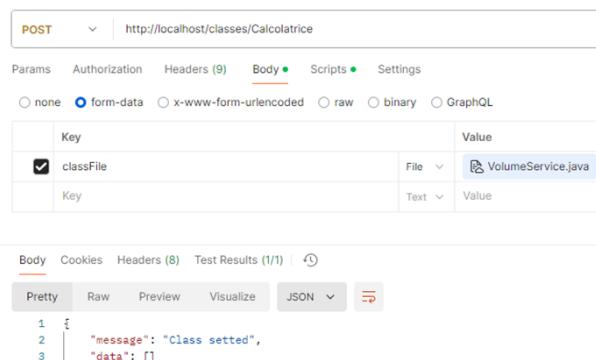
7.2.5.1. Test 1

Body: "classFile": VolumeService.java

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

Post condizione: 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

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

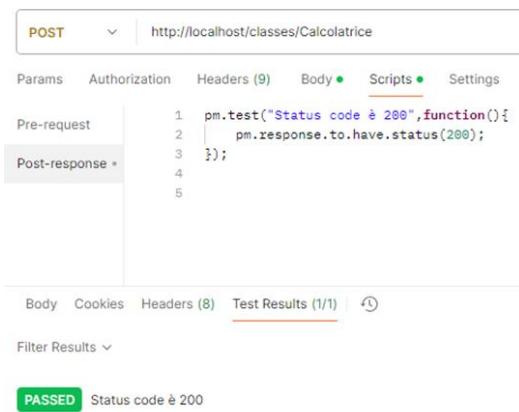
Key	Value
classFile	VolumeService.java

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

Pretty Raw Preview Visualize JSON

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

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



POST http://localhost/classes/Calcolatrice

Params Authorization Headers (9) 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)

PASSED Status code è 200

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

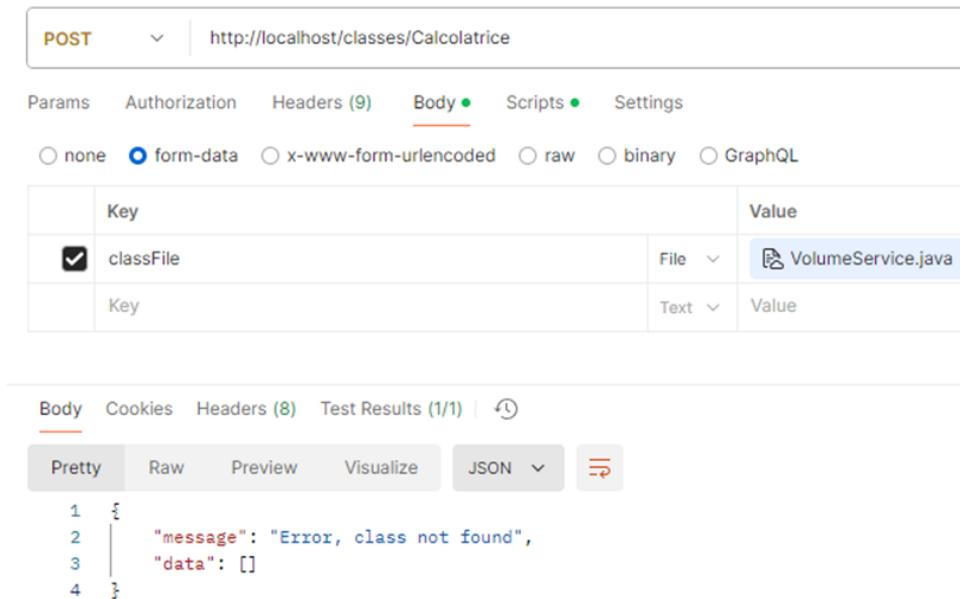
7.2.5.2. Test 2

Body: "classFile": VolumeService.java

Precondizione: nel sistema non è presente la classe calcolatrice

Post condizione: 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

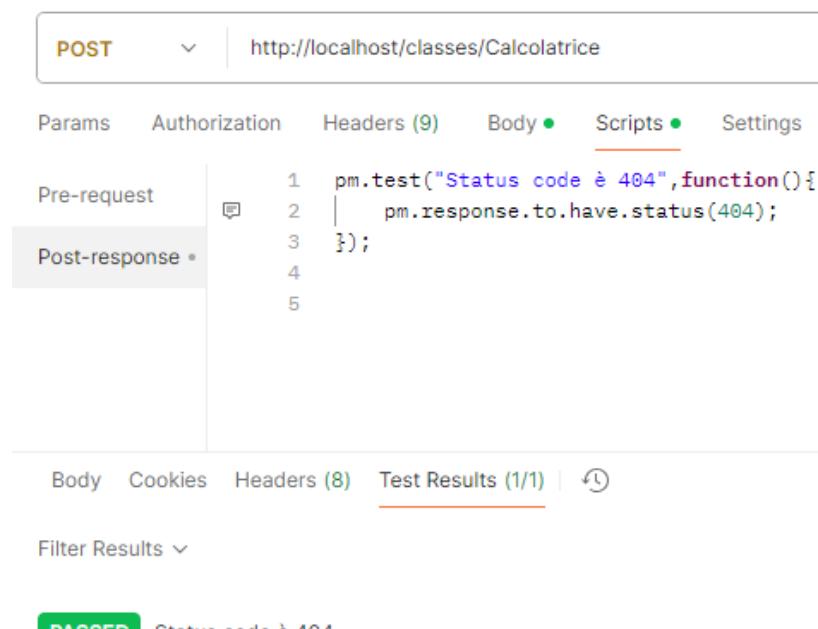
none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value
<input checked="" type="checkbox"/> classFile	File <input type="button" value="▼"/> <input type="button" value="VolumeService.java"/>
Key	Text <input type="button" value="▼"/> 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

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

Risposta Attesa: 401

The screenshot shows two separate test results from the Postman interface, both for a POST request to `http://localhost/classes/Calcolatrice`.

Test Result 1 (Top):

- Method: POST
- URL: `http://localhost/classes/Calcolatrice`
- Body Type: form-data
- Body Content:

```
1 {  
2   "message": "Error, token not valid",  
3   "data": []  
4 }
```
- Status: Failed (indicated by a red circle icon)

Test Result 2 (Bottom):

- Method: POST
- URL: `http://localhost/classes/Calcolatrice`
- Body Type: form-data
- Body Content:

```
1 {  
2   "message": "Error, token not valid",  
3   "data": []  
4 }
```
- Status: Passed (indicated by a green circle icon)

Both tests have a status code of 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

Post condizione: 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 screenshot displays two side-by-side API requests in a tool like Postman or similar.

Left Request (Initial State):

```
{
  "_id": {
    "$oid": "676209578aa6ce4abe551279"
  },
  "name": "Calcolatrice",
  "date": "",
  "difficulty": "Beginner",
  "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java",
  "description": "",
  "category": [
    "",
    "",
    ""
  ],
  "robots": [],
  "_class": "com.groom.mansclass.model.ClassUT"
}
```

Right Request (Updated State):

```
[
  {
    "_id": {
      "$oid": "676209578aa6ce4abe551279"
    },
    "name": "Calcolatrice",
    "date": "",
    "difficulty": "Beginner",
    "code_Uri": "/Volume/Root/Classes/Calcolatrice/SourceCode/Calcolatrice.java",
    "description": "",
    "category": [
      "",
      "",
      ""
    ],
    "robots": [
      {
        "robotName": "ChatGPT",
        "robotFile": "/Volume/Root/Classes/Calcolatrice/Tests/ChatGPT/"
      }
    ],
    "_class": "com.groom.mansclass.model.ClassUT"
  }
]
```

Request Details:

- Method:** POST
- URL:** http://localhost/classes/Calcolatrice/ChatGPT
- Body:** ChatGPTTest.zip

Request Headers: (9)
Params, Authorization, Headers (9), Body (highlighted), Scripts, Settings

Body Options: none, form-data (selected), x-www-form-urlencoded, raw, binary, GraphQL

Body Fields:

Key	Value
robotFile	File (ChatGPTTest.zip)
Key	Text (Value)

Response Headers: (8)
Body, Cookies, Headers (8), Test Results (1/1)

Test Results: PASSED Status code è 200

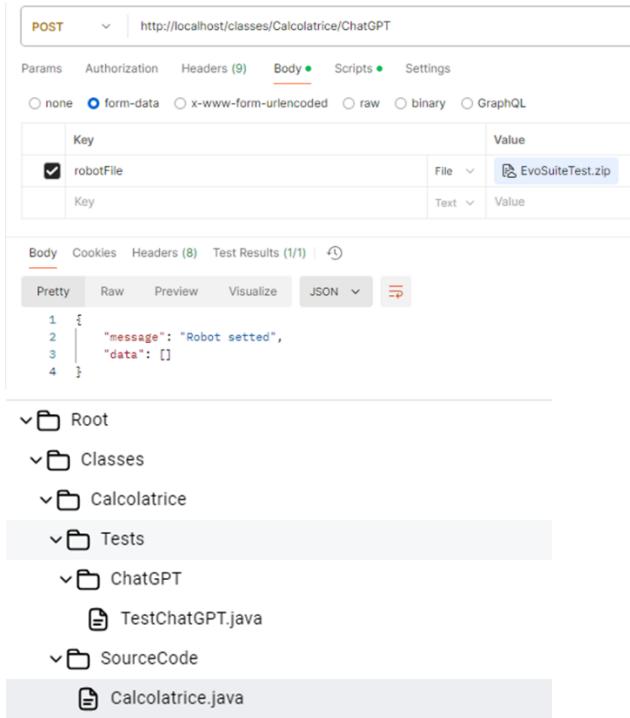
7.2.6.2. Test 2

Body: "robotFile": EvoSuiteTest.zip

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

Post condizione: 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



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
robotFile	File <input type="file" value="EvoSuiteTest.zip"/>
Key	Text Value

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

Pretty Raw Preview Visualize JSON

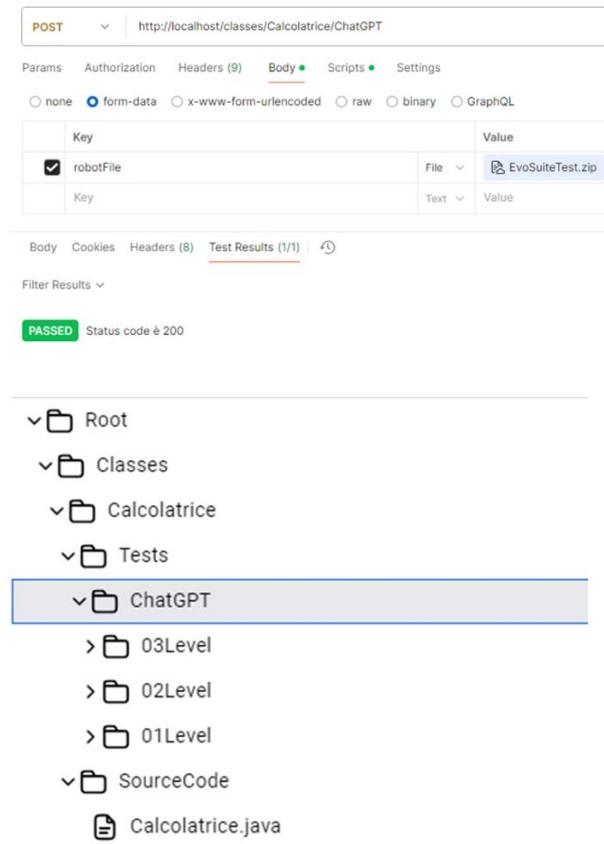
```
1 {
2     "message": "Robot setted",
3     "data": []
4 }
```

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

Filter Results ▾

PASSED Status code è 200

Root
Classes
Calcolatrice
Tests
ChatGPT
TestChatGPT.java
SourceCode
Calcolatrice.java



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
robotFile	File <input type="file" value="EvoSuiteTest.zip"/>
Key	Text Value

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

Filter Results ▾

PASSED Status code è 200

Root
Classes
Calcolatrice
Tests
ChatGPT
03Level
02Level
01Level
SourceCode
Calcolatrice.java

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.

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

Risposta Attesa: 400

```
T1-G11 > applicazione > manvsclass > RobotConfig > robots.txt
1 Randoop
2 EvoSuite
3 ChatGPT
4 Gemini
```

POST http://localhost/classes/Calcolatrice/Pippo

Params Authorization Headers (9) Body Scripts Settings

Pre-request

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

Post-response *

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

Pretty Raw Preview Visualize JSON

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

POST http://localhost/classes/Calcolatrice/Pippo

Params Authorization Headers (9) Body Scripts Settings

Pre-request

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

Post-response *

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

Filter Results

PASSED Status code è 400

7.2.6.4. Test 4

Body: "robotFile": VolumeService.zip

Precondizione: nel sistema non è presente la classe Calcolatrice

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

Risposta Attesa: 404

The screenshot shows the Postman interface with two separate test cases for a POST request to `http://localhost/classes/Calcolatrice/EvoSuite`.

Test Case 1 (Top):

- Method:** POST
- URL:** `http://localhost/classes/Calcolatrice/EvoSuite`
- Headers:** (9)
- Body:** (Pretty) JSON response:

```
1 {  
2   "message": "Error, class not found",  
3   "data": []  
4 }
```
- Scripts:** (1)

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

Test Case 2 (Bottom):

- Method:** POST
- URL:** `http://localhost/classes/Calcolatrice/EvoSuite`
- Headers:** (9)
- Body:** (Pretty) JSON response:

```
1 {  
2   "message": "Error, class not found",  
3   "data": []  
4 }
```
- Scripts:** (1)

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

Test Results:

- Test Results (1/1):** PASSED Status code è 404

7.2.6.5. Test 5

Body: "robotFile": ChatGPTTest.zip

Precondizione: l'utente non ha effettuato l'accesso

Post condizione: 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 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 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

Post condizione: 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

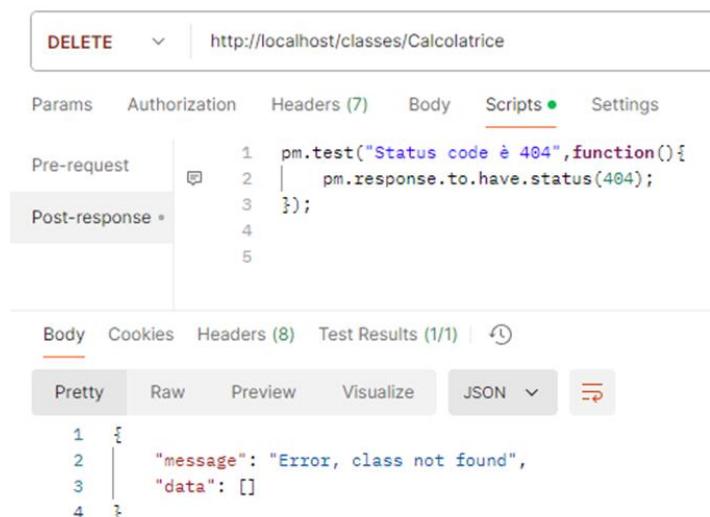
7.2.7.2. Test 2

Body:

Precondizione: nel sistema non è presente la classe calcolatrice

Post condizione: 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

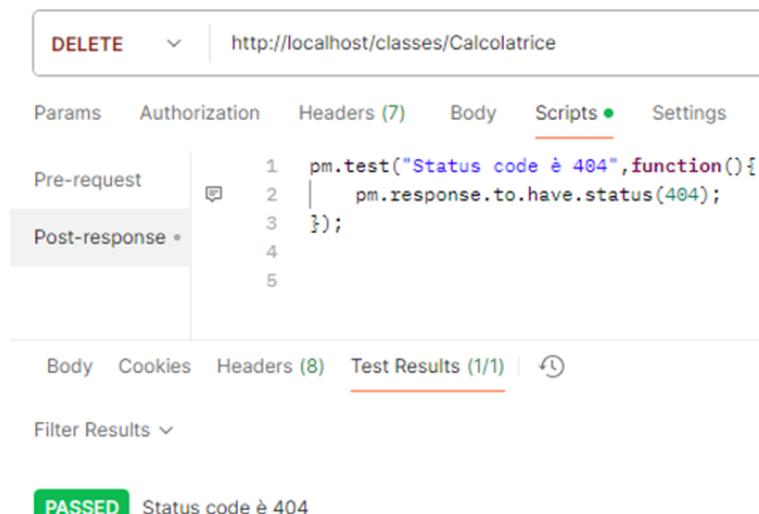
Pre-request Post-response

```
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

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

Risposta Attesa: 401

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

Test Case 1 (Top):

- Method: **DELETE**
- URL: `http://localhost/classes/Calcolatrice`
- Pre-request Script (Post-response selected):

```
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 Case 2 (Bottom):

- Method: **DELETE**
- URL: `http://localhost/classes/Calcolatrice`
- Pre-request Script (Post-response selected):

```
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 }
```

At the bottom of the interface, there is a green **PASSED** button with the text "Status code è 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

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

Risposta Attesa: 200

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

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

The screenshot shows the Postman interface displaying the test results for the DELETE request. The status is 'PASSED' with the message 'Status code è 200'. The response body is:

```
PASSED Status code è 200
{
  "message": "Robot deleted",
  "data": []
}
```

The screenshot shows a MongoDB document in the Compass interface. The document has the following structure:

```
{
  "_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"
}
```

The screenshot shows the same MongoDB document in the Compass interface, but it has been modified to show an empty array for the 'category' field. The 'robots' field also contains one less entry than before.

```
{
  "_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

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

Risposta Attesa: 404

The screenshot shows the Postman interface with a DELETE request to `http://localhost/classes/Calcolatrice/EvoSuite`. The request includes a `pm.test` script in the Post-response tab:

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

The response body is displayed in Pretty format:

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

The Test Results tab shows a single result: **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

Post condizione: 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 | 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, robot 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

7.2.8.4. Test 4

Body:

Precondizione: l'utente non ha effettuato l'accesso

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

Risposta Attesa: 401

DELETE

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 }
```

DELETE

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

8. Sviluppi futuri

Durante il lavoro sull'applicazione, il team 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 *FileSystem*;
- **Valutare la sostituzione del database:** potrebbe essere opportuno migrare da *MongoDB* a un database relazionale (es. *MySQL*).