



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Elaborato Software Architecture Design

Documentazione Integrazione Man vs Automated Testing Tools challenges

Anno Accademico 2023/2024

Professoressa
Anna Rita Fasolino

Studenti
Marco Di Fiandra matr. M63001444
Carlo Portosalvo matr. M63001538
Mariano Barone matr. M63001419
Gaetano Saviano matr. M63001502

Contents

1	Introduzione	1
1.1	Progetto	1
1.1.1	Cluster di task assegnato	2
1.2	Metodologia applicata	3
1.3	Workflow Diagram	4
2	Analisi dei Requisiti	5
2.1	User Stories	5
2.2	Diagramma dei casi d'uso	6
2.3	Scenari	8
2.3.1	Login e Logout	8
2.3.2	Add Class	10
2.3.3	Create Game	10
2.3.4	Write Code	11
2.3.5	inspectClassUnderTest	12
2.3.6	Compile	13
2.3.7	Run	13
2.3.8	Run with JaCoCo	14
2.4	Interrelazioni tra i Task	15

2.5	Obiettivi prefissati e pre analisi dei task	16
3	Progettazione	18
3.1	Architettura a microservizi	18
3.1.1	Nota su Microservizi non integrabili	19
3.1.2	Gateway Pattern	20
3.1.3	UIGateway	20
3.1.4	APIGateway	21
3.2	Component Diagram	26
3.3	Composite Structure Diagram	27
3.4	Tecnologie usate	28
3.4.1	Perchè Spring Cloud ?	30
4	Error Repo	32
4.1	Documentazione Task	32
4.2	Tecnologie usate per lo sviluppo	36
4.3	Test dell'API	36
4.4	Deployment	39
4.4.1	Deployment Diagram	39
4.4.2	Possibili sviluppi ed utilità del microservizio . .	40
5	Modifiche apportate	42
5.1	Modifiche Task 2-3	42
5.2	Modifiche Task 1	44
5.3	Modifiche Task 6	44
5.4	Modifiche Task 8	45
5.5	Modifiche task 5	47

6	Testing	48
6.0.1	Tool	49
6.1	Casi di Test	51
6.1.1	Login Test	52
6.1.2	Editor Test:	57
6.2	Evosuite:	66
6.2.1	Risultati	68
7	Deployment	70
7.1	Deployment Diagram	71
8	Installazione	74
9	Conclusioni e future implementazioni	76
9.1	Spunti e possibili implementazioni	78

Chapter 1

Introduzione

Di seguito è presentato il processo di integrazione del progetto "Man vs Automated Testing Tools challenges" ideato nel contesto di EN-ACTEST

1.1 Progetto

Si vuole realizzare una applicazione software che permetta di stimolare gli studenti a cimentarsi nel Test case design per mezzo di un gioco dove questi ultimi devono competere contro strumenti di generazione automatica del test. Gli strumenti automatici scelti a tale scopo sono EvoSuite e Randoop.

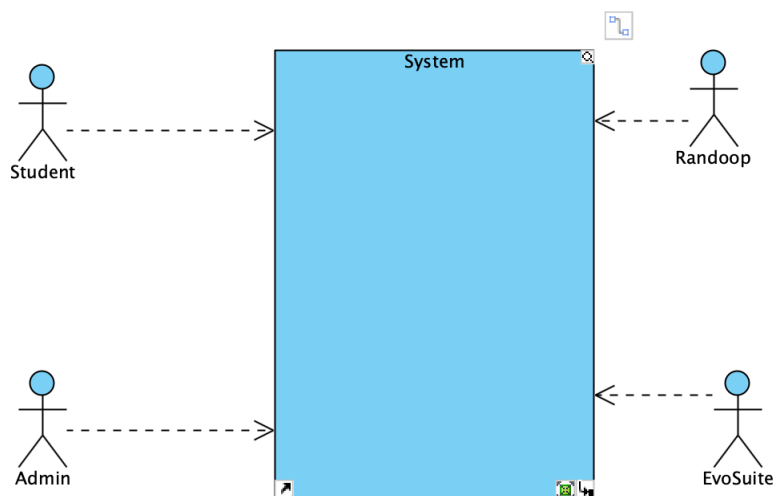


Figure 1.1: Context Diagram

1.1.1 Cluster di task assegnato

Il task assegnatoci è l'integrazione del sistema, ovvero attraverso un set di servizi sviluppati rendere funzionante il sistema.

Il set di servizi assegnatoci è riportato nella figura 1.2

Cluster	T1	T2-3	T4	T5	T6	T7	T8	T9
1	G11	G1	G18	G2	G12	G31	G21	G19

Figure 1.2: Task set

Come si può vedere, l'applicazione è stata immaginata come una serie di task che dovranno comunicare tra di loro secondo il paradigma a microservizi per produrre il software finale.

1.2 Metodologia applicata

Al fine di produrre l'elaborato si è deciso di ricondursi ad un approccio **AGILE** basato sul paradigma **SCRUM**, il quale è un framework per la gestione iterativa e incrementale del progetto.

Lo sviluppo si è articolato su varie iterazioni, come si può notare della figura 1.3 Alla fine di tre iterazioni, il gruppo è riuscito a raggiungere l'obiettivo finale completo di documentazione.

	A ITERATION 1	A DONE1	A ITERATION 2	A DONE2	A ITERATION 3	A DONE3
1	Stakeholder Meeting	Stakeholder Meeting	Early Architecture Documentation	Early Architecture Documentation	Full Architecture Documentation	Full Architecture Documentation
2	High Level Problem Analysis	High Level Problem Analysis	Task Debugging and Remapping	Task Debugging and Remapping	Integration Test and Demo	Integration Test and Demo
3	Case Study	Case Study	Ulgateway and API development	Ulgateway and API development	Final Review	Final Review
4	Technical Meeting with Other Teams	Technical Meeting with Other Teams	Integration Tasks	Integration Tasks	Release Product	Release Product
5	Task Interface Study	Task Interface Study	Full Architecture Documentation			
6	Task Debugging and Remapping					
7	Early Documentation					

Figure 1.3: Iteration Plan

La nostra metodologia di lavoro si è basata su *sprint* settimanale : all'inizio di ognuno di essi, si programmavano gli obiettivi dei singoli componenti del gruppo e le attività per raggiungerli.

La collaborazione all'interno del team è stata possibile da tecnologie come **git**, fondamentale per la condivisione e la modifica del codice, la funzione **Live Share** dell'IDE Visual Studio Code, permettendoci di scrivere codice attenzionato da tutto il team, nei passi fondamen-

tali per il raggiungimento dell'obiettivo del team. Per i meeting con gli stakeholder e inter-team sono è stata utilizzata la piattaforma **Teams**. Infine per il testing sono stati utilizzati **Selenium**, ambiente finalizzato all'automatizzazione delle operazioni sul browser e **Postman**, un'applicazione utilizzata per semplificare il processo di test e sviluppo delle API offrendo un'interfaccia utente intuitiva che consente agli sviluppatori di inviare richieste HTTP a un server o a un'applicazione web, visualizzare le risposte ricevute e analizzare i dati restituiti.

1.3 Workflow Diagram

Il contesto di gioco delineato in questa situazione è quello illustrato nella 1.4, con un unico partecipante, un unico turno e un unico round. Per descriverlo, abbiamo utilizzato un diagramma di flusso di lavoro, che rappresenta sequenze logiche di attività che, quando combinate, descrivono i processi aziendali. Da questo diagramma è semplice individuare i microservizi che possono essere sviluppati.

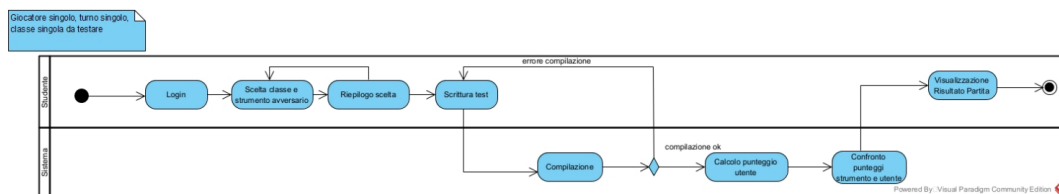


Figure 1.4: Workflow Diagram

Chapter 2

Analisi dei Requisiti

Per dettagli specifici sui requisiti dei singoli microservizi, si rimanda alla documentazione individuale relativa agli altri compiti, dove è stata condotta un'analisi dettagliata.

2.1 User Stories

Sono stati identificati ulteriori requisiti, di cui il più significativo riguarda la generazione dei test del robot, da effettuare contestualmente all'inserimento della classe nella repository da parte dell'amministratore. La ragione di questa scelta è il fatto che questa operazione richiede tempo e pertanto è preferibile separarla dalla partita: i risultati devono essere già disponibili per non ostacolare l'esperienza di gioco degli studenti e devono essere calcolati in anticipo. Questi nuovi requisiti sono descritti nelle seguenti user stories.

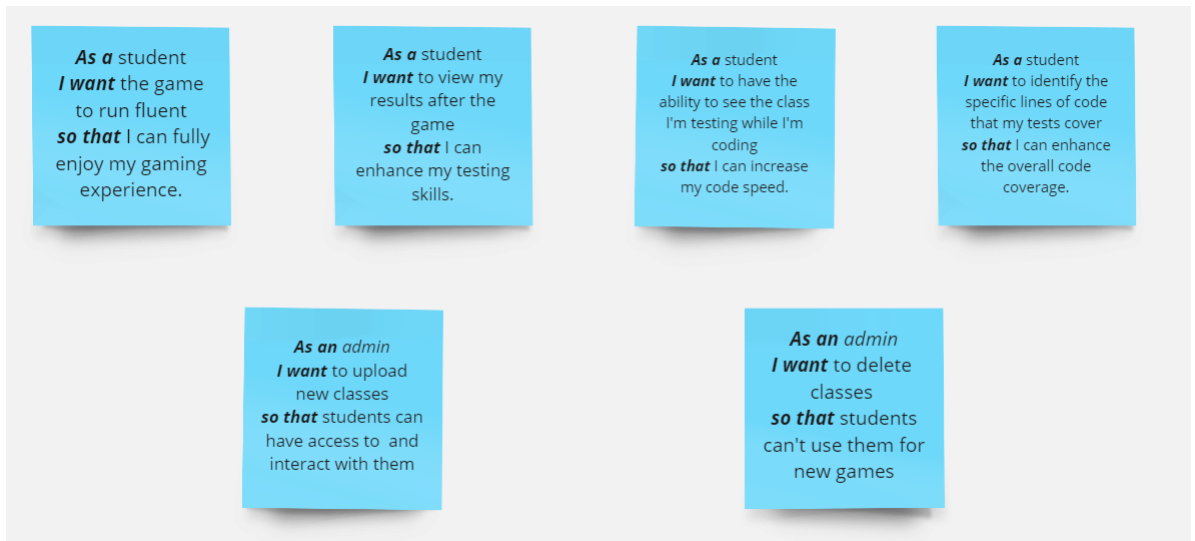


Figure 2.1: User Stories importanti individuate

2.2 Diagramma dei casi d'uso

Si riporta di seguito un diagramma dei casi d'uso generale che rappresenti tutte le funzionalità disponibili in questa integrazione. I casi d'uso realizzati al fine dell'integrazione sono contrassegnati in verde nel diagramma. Si è cercato di mantenere coerenza nei nomi dei casi d'uso rispetto a quelli dei singoli task. Tuttavia, per i casi d'uso relativi all'amministratore, è stato necessario tradurre i nomi in inglese al fine di mantenere coerenza linguistica. È importante osservare che sia "View classes" che "Create Game" includono la fase di accesso preliminare, la quale è richiesta. Inoltre, sono disponibili una serie di casi d'uso di estensione, opzionali per entrambi i tipi di utente. È da notare che "Run" rappresenta effettivamente la sottomissione dell'attempt da

parte dello studente, ma per coerenza con il task 6, è stato mantenuto con questo nome e non è stato rinominato in "Submit".

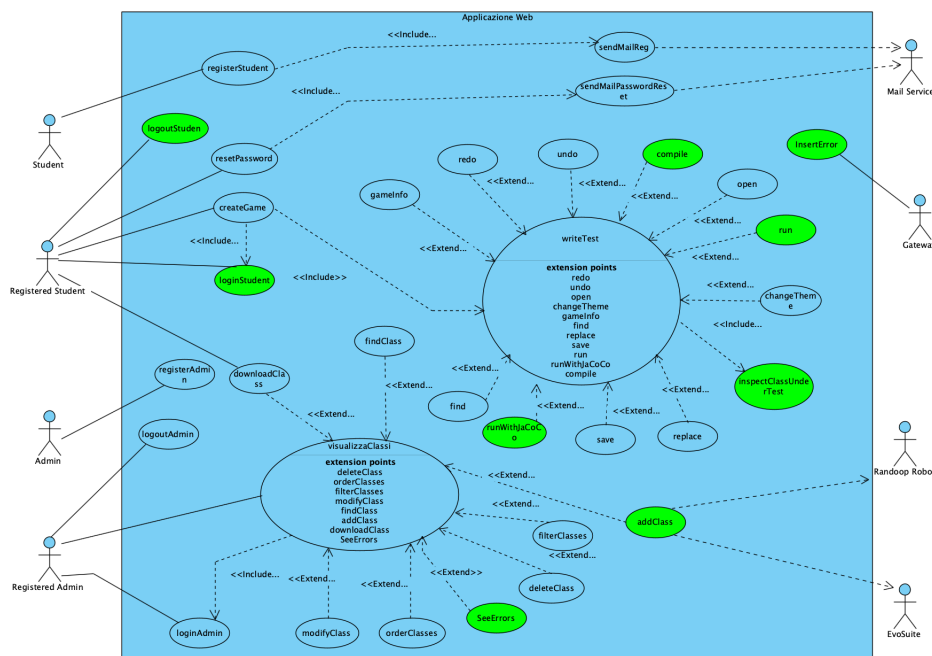


Figure 2.2: Diagramma dei casi d'uso

Attori

Il "*Registered Student*" è l'utente che ha la possibilità di partecipare a una partita contro il robot, scrivendo il proprio test. Il "*Registered Admin*" è l'utente che gestisce le classi con le quali è possibile giocare all'interno del sistema. Nel caso in cui questi due attori non siano registrati, vengono semplicemente chiamati "*Student*" e "*Admin*".

Il "*Randoop Robot*" e "*EvoSuite*" sono due attori secondari che interagiscono con il caso d'uso di aggiunta di una nuova classe da parte dell'Admin; contestualmente, i robot generano i loro test. Il "*Servizio*

Mail" è anch'esso un attore secondario che partecipa nella fase di registrazione e nel recupero della password in collaborazione con lo studente

2.3 Scenari

Qui verranno descritti gli scenari dei casi d'uso evidenziati in verde, in cui sono state apportate modifiche significative. Per quanto riguarda gli altri casi d'uso, è possibile trovare dettagli nelle documentazioni relative al Task 1 per i casi d'uso dell'Admin, nel Task 2-3 per i casi d'uso di autenticazione e nel Task 6 per le funzioni di estensione semplici del caso d'uso "Write Code". Le modifiche più rilevanti sono state apportate nel caso d'uso "Add class", dove è stata presa la decisione di eseguire localmente e contemporaneamente all'inserimento della classe da parte dell'Admin nella repository la generazione automatica dei test da parte dei robot Randoop ed EvoSuite. Pertanto, lo scenario è differente rispetto a quello originale.

2.3.1 Login e Logout

Lo studente, prima di poter effettuare la scelta delle classi, deve effettuare l'autetificazione tramite indirizzo email e password.

Caso d'uso	Login
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato può effettuare l'autenticazione
Pre-condizioni	Lo studente deve essere registrato
Sequenza di eventi principale	1) Lo studente registrato inserisce i dati per l'autenticazione negli appositi riquadri 2) Lo studente clicca sul pulsante "Accedi"
Post-Condizione	Lo studente registrato visualizza la schermata di inizio partita
Casi d'uso correlati	Null
Sequenza di eventi alternativi	1) Lo studente registrato ha inserito uno dei dati per l'autenticazione in maniera errata. Viene mostrato un messaggio d'errore 2) Lo studente registrato può reimpostare la password cliccando sul pulsante "reimposta password".

Figure 2.3: Login

Dopo aver utilizzato l'applicazione, lo studente può disconnettere il proprio account effettuando il logout.

Caso d'uso	Logout
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato può effettuare il logout
Pre-condizioni	Lo studente deve essere autenticato
Sequenza di eventi principale	1) Lo studente registrato clicca il pulsante "Logout".
Post-Condizione	Lo studente registrato visualizza la schermata di autenticazione
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Null

Figure 2.4: Logout

2.3.2 Add Class

L'Admin inserisce nuove classi nella repository che verranno successivamente utilizzate dagli studenti per effettuare il game.

Caso d'uso	Add Class
Attore primario	Registered Admin
Attore secondario	Robot Randoop
Descrizione	L'admin aggiunge una nuova classe da testare all'interno della repository
Pre-condizioni	L'admin si è autenticato tramite login
Sequenza di eventi principale	<ol style="list-style-type: none"> 1) L'admin decide di creare una nuova classe. Clicca sul pulsante add class 2) Il sistema mostra i campi d'informazione da riempire 3) L'admin riempie i campi d'informazione con nome, data, difficoltà della classe, una descrizione e tre categorie se si vuole 4) Clicca sul pulsante per fare l'upload della classe 5) Il sistema salva la classe 6) Il Robot Randoop genera i test relativi a quella classe 7) Il sistema calcola la copertura e si salvano i risultati 8) Il sistema restituisce tutte le classi restituite
Post-Condizione	La nuova classe inserita è disponibile per lo studente durante la creazione della partita. Sia il Robot Randoop e il livello di difficoltà possono essere selezionati durante la creazione della partita.
Casi d'uso correlati	Null
Sequenza di eventi alternativi	

Figure 2.5: Add Class

2.3.3 Create Game

Uno studente, che ha effettuato il login, può creare un game scegliendo il robot da sfidare, il livello di difficoltà e i test da effettuare.

Caso d'uso	Create Game
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato sceglie la classe da testare e i robot con il livello da affrontare
Pre-condizioni	Lo studente deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1) Lo studente registrato effettua l'autenticazione 2) Il sistema mostra una lista di classi da poter scegliere. 3) Lo studente registrato sceglie la classe. 4) Il sistema mostra i robot e i livelli disponibili. 5) Lo studente sceglie il robot e il livello e clicca il tasto "Submit" 6) Il sistema mostra una pagina di riepilogo 7) Lo studente registrato conferma la sua scelta 8) Il sistema salva le informazioni e mostra allo studente l'editor
Post-Condizione	Lo studente registrato si trova nella schermata di editor dove può scrivere il suo test
Casi d'uso correlati	Null
Sequenza di eventi alternativi	<p>Lo studente registrato clicca il tasto "Back".</p> <p>Lo studente registrato viene portato alla schermata di selezione delle classi.</p>

Figure 2.6: Create Game

2.3.4 Write Code

Dopo che lo studente ha creato la partita, sarà in grado di scrivere il test e, se lo desidera, attivare alcune funzionalità sull'editor.

Caso d'uso	Write Code
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato può scrivere il suo test.
Pre-condizioni	Lo studente registrato ha avviato la partita.
Sequenza di eventi principale	1) Lo studente registrato scrive il codice Java nell'editor.
Post-Condizione	Null
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Lo studente registrato carica da locale la classe di test

Figure 2.7: Write Code

2.3.5 inspectClassUnderTest

Dopo aver avviato l'editor, è possibile caricare all'interno di quest'ultimo la classe scelta dallo studente registrato per il testing.

Caso d'uso	inspectClassUnderTest
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato può visualizzare una classe
Pre-condizioni	Lo studente registrato deve aver scelto la classe dalla Repository
Sequenza di eventi principale	1) La Class Under Test è caricata nell'editor di testo
Post-Condizione	Lo studente registrato vede la classe caricata nell'editor
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Null

Figure 2.8: inspectClassUnderTest

2.3.6 Compile

Dopo aver premuto il riquadro "Compile", il risultato della compilazione deve essere mostrato a video in un apposito spazio.

Caso d'uso	Compile
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato effettua la compilazione
Pre-condizioni	La classe di test deve essere implementata
Sequenza di eventi principale	1) Lo studente registrato clicca sul tasto relativo alla compilazione
Post-Condizione	Lo studente registrato vede l'esito della compilazione
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Null

Figure 2.9: Compile

2.3.7 Run

Si desidera che, al clic del pulsante "Run", l'utente invii il proprio tentativo al robot e che i risultati della partita vengano mostrati successivamente.

Caso d'uso	Run
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato esegue il test
Pre-condizioni	Lo studente registrato deve aver effettuata la compilazione
Sequenza di eventi principale	1) Lo studente registrato clicca sul tasto relativo all'esecuzione.
Post-Condizione	Lo studente registrato vede l'esito dell'esecuzione
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Null

Figure 2.10: Run

2.3.8 Run with JaCoCo

Dopo aver premuto il riquadro "Run with JaCoCo" lo studente registrato vede la copertura della classe di test compilata. Nello specifico, vengono evidenziate le linee di codice scritte per mostrare in maniera efficiente la copertura del test.

Caso d'uso	runWithJaCoCo
Attore primario	Registered Student
Attore secondario	Null
Descrizione	Lo studente registrato può lanciare il test con JaCoCo
Pre-condizioni	Lo studente registrato deve aver compilato la classe di test
Sequenza di eventi principale	1) Lo studente registrato clicca sul tasto.
Post-Condizione	Lo studente registrato vede la copertura del test
Casi d'uso correlati	Null
Sequenza di eventi alternativi	Null

Figure 2.11: Run With JaCoCo

2.4 Interrelazioni tra i Task

Per eseguire con successo l'integrazione dei diversi Task, è stato imperativo tracciare e evidenziare le dipendenze esistenti tra di essi. Questo ha richiesto una particolare attenzione alle operazioni o casi d'uso che richiedono l'impiego di servizi forniti da altri Task. Al fine di fornire una panoramica esaustiva, si è optato per la rappresentazione grafica delle dipendenze tra le diverse operazioni. In questa rappresentazione, per ciascuna operazione (Operations), abbiamo chiarito:

- Da quali Task dipende (Collaborators).
- Quali operazioni dei Task sorgente utilizza.
- Gli endpoint delle REST API che consentono l'esecuzione di tali operazioni.

Segue un diagramma che illustra le interrelazioni tra i Task:

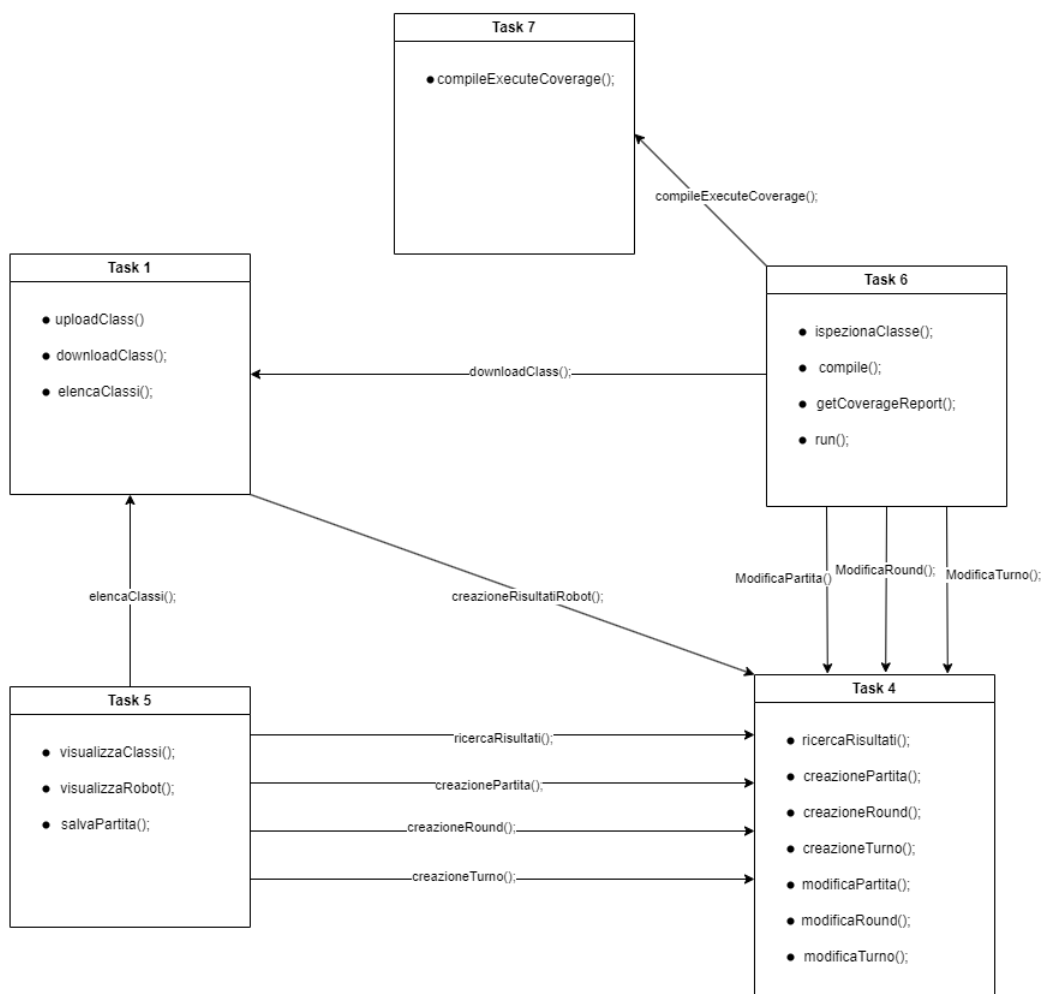


Figure 2.12: Diagramma delle dipendenze

E' possibile visionare ogni funzione inserita nel diagramma delle dipendenze nelle documentazioni dei relativi task.

2.5 Obiettivi prefissati e pre analisi dei task

Durante l'analisi del problema sono stati analizzati i task.

Inoltre sono state analizzate anche proposte di integrazioni già presen-

tate.

Tale analisi ci ha permesso di capire quali task sono integrabili e nel caso di integrabilità su quali punti agire per migliorare la qualità del software utilizzando anche tecnologie più recenti messe a disposizione da Spring Boot.

I task in analisi preliminare sono stati ri-testati secondo i loro documenti di test e nel caso di alcuni task sono stati integrati i dockerfile per il deploy di questi ultimi.

Quindi in prima analisi si è provata la funzionalità e l'interoperatività dei task.

Da queste analisi sono stati tratti degli obiettivi che sono:

- Autenticazione tramite token
- Utilizzo di tecnologie nuove per i gateway
- Aumento delle prestazioni diminuendo eventuali overhead
- Sicurezza dei microservizi
- Migliorare il monitoraggio
- Uniformare il più possibile

Chapter 3

Progettazione

3.1 Architettura a microservizi

Nell'ambito della nostra web application, abbiamo adottato un'architettura basata su microservizi, come illustrato nel diagramma riportato nella figura 3.1. In questo sistema, il cliente può accedere all'applicazione esclusivamente attraverso due gateway via internet. Un aspetto importante da sottolineare è che i microservizi non sono direttamente accessibili dall'esterno della loro rete locale. Questa configurazione offre un ulteriore strato di sicurezza, garantendo che l'accesso ai microservizi avvenga solo attraverso i gateway designati, contribuendo così a proteggere il sistema da potenziali minacce esterne.

3.1.1 Nota su Microservizi non integrabili

Durante la fase di Progettazione si è notato che alcuni task non fossero integrabili. Di seguito sono elencati i task con le relative motivazioni per cui non è possibile effettuare l'integrazione :

- **T4:** non è previsto il front-end per la gestione della sessione di gioco dell'utente (scelta/creazione di un Game, inserimento di round/turni): in tal senso non vi è soluzione di continuità tra la fase di autenticazione gestita dal T2-3 e la scelta della classe UnderTest e del Robot del task T5.
- **T7:** il microservizio prevedeva una funzionalità corretta, generando però errori nel momento dell'interfacciamento con l'Editor del Task T6. Per ragioni di tempo, non siamo stati in grado di risolvere la problematica di interfacciamento. Pertanto, se ne rimanda la risoluzione alle prossime implementazioni.
- **T9:** anche tale microservizio prevedeva una funzionalità corretta ma al momento dell'integrazione sono sorti problemi nella generazione dei casi di test al caricamento della Class Under Test da parte dell'admin, che sempre per questioni di tempo non si è riusciti a risolvere.

3.1.2 Gateway Pattern

Al centro del nostro obiettivo di sviluppare un'applicazione sicura e altamente scalabile c'è l'adozione del Gateway Pattern. Questo approccio si basa sull'introduzione di componenti specializzati che costituiscono gli unici punti di accesso ai nostri microservizi. Questa scelta strategica ci consente di implementare meccanismi avanzati di autenticazione, autorizzazione, routing, bilanciamento del carico e composizione delle richieste API in modo estremamente versatile ed efficiente. In pratica, il Gateway Pattern agisce come un guardiano e un intermediario tra il cliente e i nostri microservizi, offrendo un'interfaccia di accesso centralizzata e potenziando la sicurezza e le prestazioni complessive dell'applicazione. Questo approccio costituisce un elemento fondamentale nella creazione di un ambiente di sviluppo robusto e scalabile per soddisfare le nostre esigenze di sicurezza e crescita futura.

3.1.3 UIGateway

Questa componente è stata implementata utilizzando SpringBoot, creando effettivamente un server proxy inverso. La sua unica responsabilità è quella di gestire il reindirizzamento delle richieste provenienti dal frontend (UI). Questa configurazione ha consentito di rendere l'applicazione accessibile tramite la porta 80, che è la porta predefinita per le connessioni HTTP, garantendo così una certa uniformità.

3.1.4 APIGateway

Questa componente è stata sviluppata utilizzando Spring Boot insieme a Spring Cloud. Anche in questo caso, agisce come un server proxy inverso ma con responsabilità aggiuntive. Oltre a gestire il reindirizzamento delle richieste, si occupa anche dell'autenticazione e dell'autorizzazione delle richieste relative alle REST API. Grazie a questa configurazione, tutte le API sono rese disponibili sotto il percorso URL `"/api/"`, garantendo così una maggiore uniformità nell'accesso alle risorse.

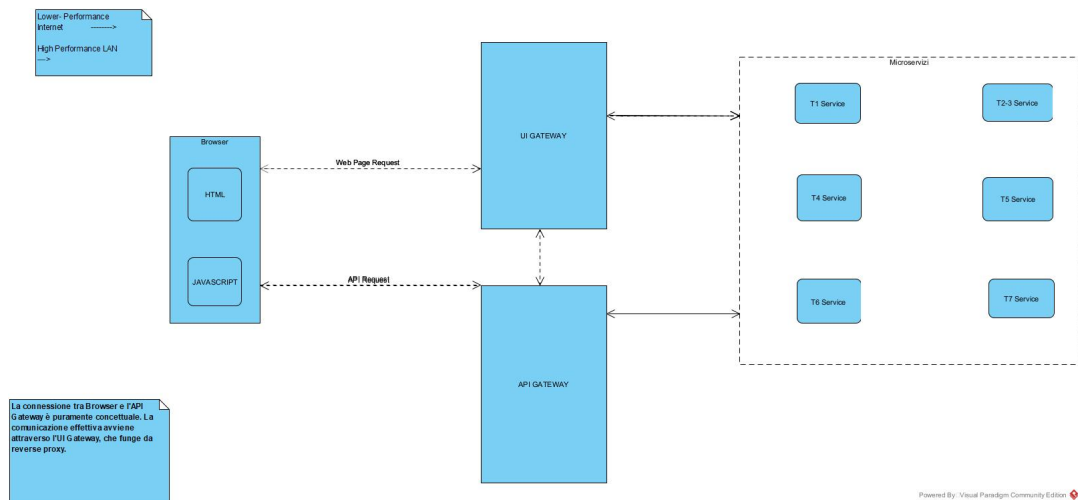


Figure 3.1: Architettura Generale del sistema

Di seguito si riportano dei sequence riguardanti la principale interazione tra l'utente i gateway e i microservizi. Dopodichè nei successivi diagrammi tale interazione sarà omessa.

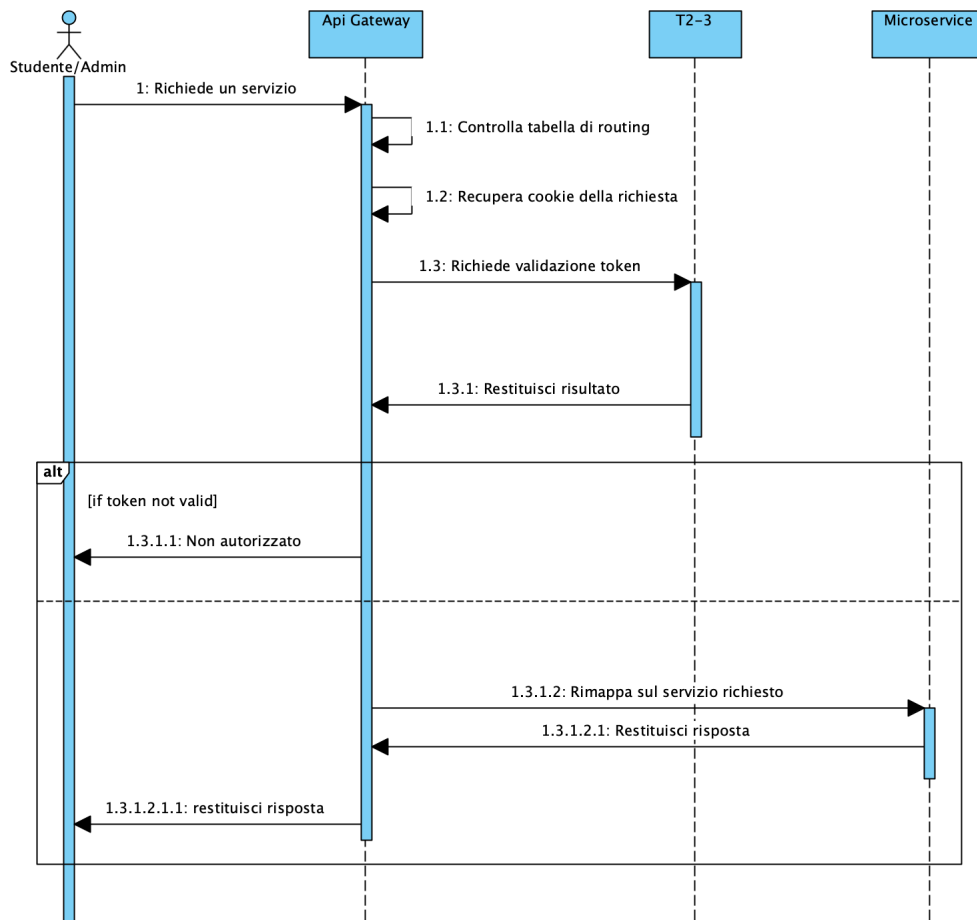


Figure 3.2: Interazione con ApiGateway

Come si può vedere l’api-gateway si occupa del re-routing, difatti rappresenta il punto di accesso del sistema che poi reindirizzerà ai servizi richiesti.

Si occupa anche dell’autenticazione dell’utente tramite token, ovvero mi sincero che l’utente non solo sia registrato ma abbia fatto anche accesso.

Come si può infatti vedere dalla figura 3.2 uno studente (che supponiamo in questo caso abbia fatto il login) richiede un servizio all’api gateway, quest’ultimo controlla il suo token che è stato generato in

fase di login, se tale token è valido allora può fare uso del servizio, altrimenti viene restituito un codice di accesso non autorizzato.

L'autenticazione è garantita dal fatto che il token è criptato e dunque una decrittazione consistente con la generazione è possibile solo usando la stessa chiave di cifratura che è posseduta da T2-3.

Ovviamente avere solo un Api-Gateway può portare ad un **SPOF** (Single Point Of Failure) per cui si può pensare a replicare la struttura gateway.

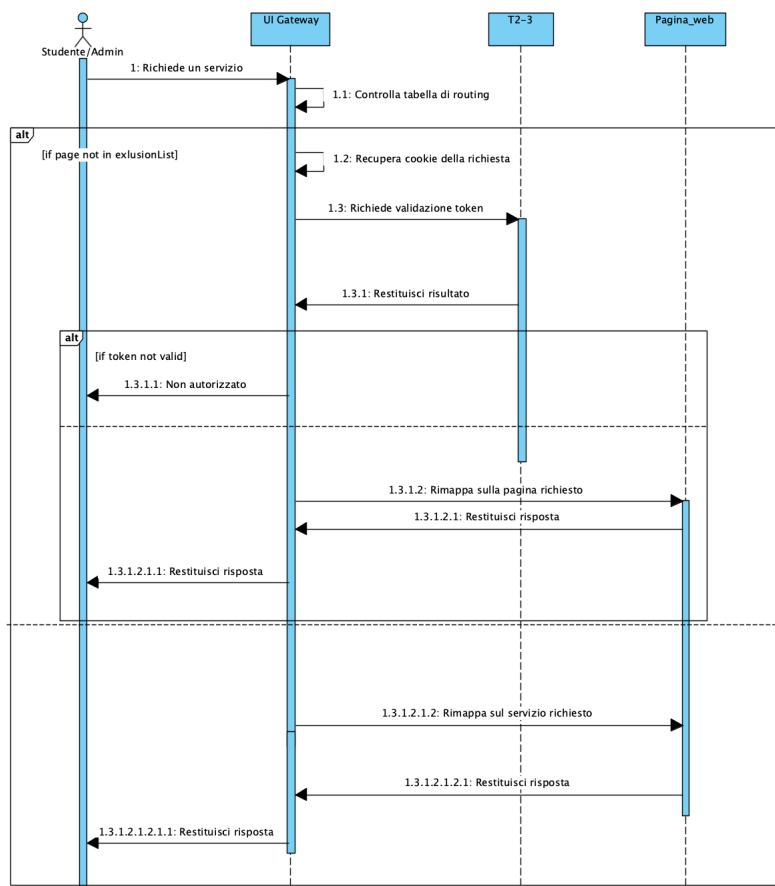


Figure 3.3: Interazione con UIGateway

Come si può vedere dal precedente diagramma l'UI-gateway svolge un lavoro simile all'api-gateway ma con le interfacce web.

Si è deciso di aggiungere una lista di esclusione pagine poichè non tutte le pagine prevedevano l'accesso dell'utente nell'applicazione (ad esempio login).

Volendo descrivere l'interazione nella figura 3.3 abbiamo che uno studente/admin (che stavolta non deve essere per forza autenticato) chiede un servizio. L'UI gateway a differenza dell'api-gateway ha una lista di pagine escluse dal controllo del token (poichè sono pagine entry point, come il login o la registrazione) se la pagina non è presente in tale lista, significa che tale pagina prevede autenticazione e dunque devo controllare il token, il quale flusso è del tutto uguale all'api gateway. In precedenza sono stati riportati come sono stati pensati le nuove entità del progetto per effettuare l'integrazione, di seguito però riportiamo quella che è l'implementazione effettiva dei gateway e di come avviene l'interazione con essi.

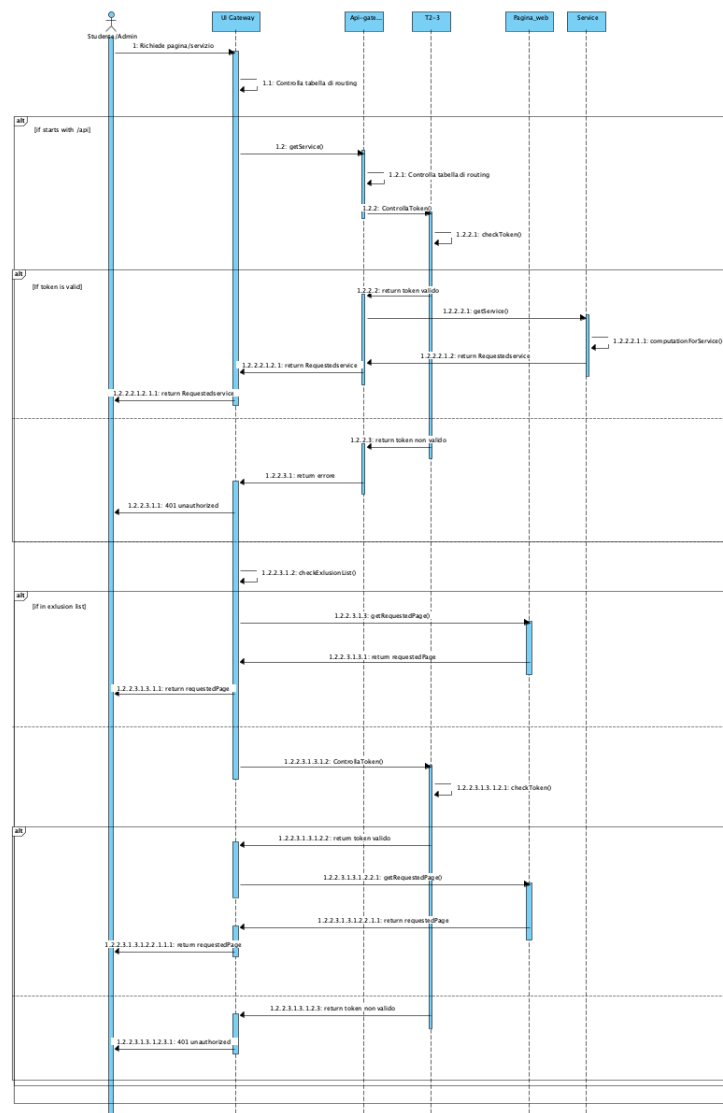


Figure 3.4: Interazione completa

Come si può vedere l'entry point vero e proprio è rappresentato dall'UI-gateway che nel caso la richiesta dell'utente sia di un servizio API, viene re-indirizzata all'api gateway con lo stesso procedimento visto in precedente, e per non effettuare un doppio controllo, le richieste di servizio non sono affette nell'UI-gateway a controllo del token, tale controllo verrà effettuato dall'api-gateway.

3.2 Component Diagram

Il diagramma dei componenti è uno strumento che ci aiuta a comprendere, durante l'esecuzione, quali entità compongono il sistema e come interagiscono tra di loro per svolgere le funzioni richieste.

È importante notare che i componenti responsabili delle funzioni relative agli studenti sono isolati da quelli dedicati all'amministrazione. Questa separazione è dovuta a un diverso approccio all'autenticazione. Per quanto riguarda l'accesso degli studenti, è stato implementato un ulteriore livello di sicurezza grazie all'utilizzo dei token JWT forniti dal componente Student Repository. Inoltre, è stato possibile garantire un controllo di sicurezza aggiuntivo grazie all'API Gateway. Tuttavia, è importante sottolineare che quest'ultimo non offre la stessa protezione per le API relative all'amministrazione.

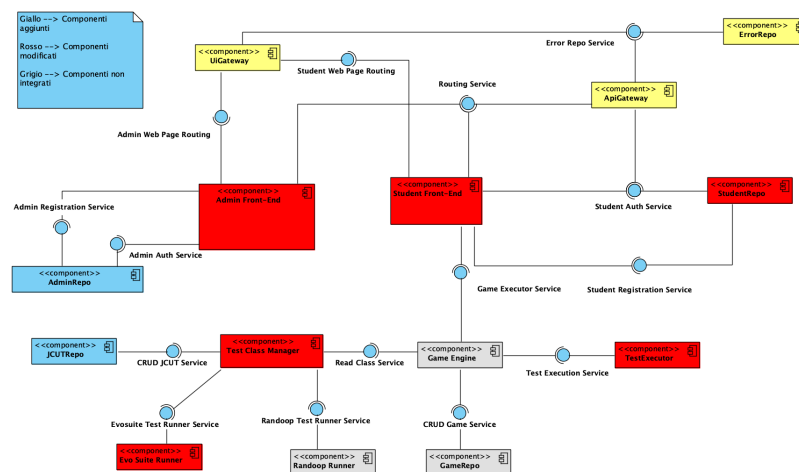


Figure 3.5: Component Diagram

3.3 Composite Structure Diagram

Nel diagramma seguente, è stata rappresentata la mappatura di quali task sono responsabili dello sviluppo di ciascun componente visibile durante l'esecuzione del sistema. La notazione utilizzata, coerente con il Component Diagram, prevede l'uso di un pacchetto per ciascun componente, specificando quali task sono responsabili del suo sviluppo e contribuiscono al suo funzionamento.

Mentre alcuni componenti sono completamente realizzati da task individuali, ce ne sono altri, come lo Student Front End e il Game Engine, che coinvolgono due task distinti.

Il componente dello Student Front End è stato realizzato dal task 2-3 per quanto riguarda l'aspetto legato all'accesso al gioco, che include le schermate di login, registrazione e recupero password. Per la parte che riguarda il gioco vero e proprio, il front end è stato sviluppato nel task 5.

Il Game Engine, invece, è stato realizzato in due parti: la parte relativa al salvataggio iniziale della partita è stata sviluppata dal task 5, mentre le altre funzionalità, come l'esecuzione e la compilazione, sono state gestite dal task 6.

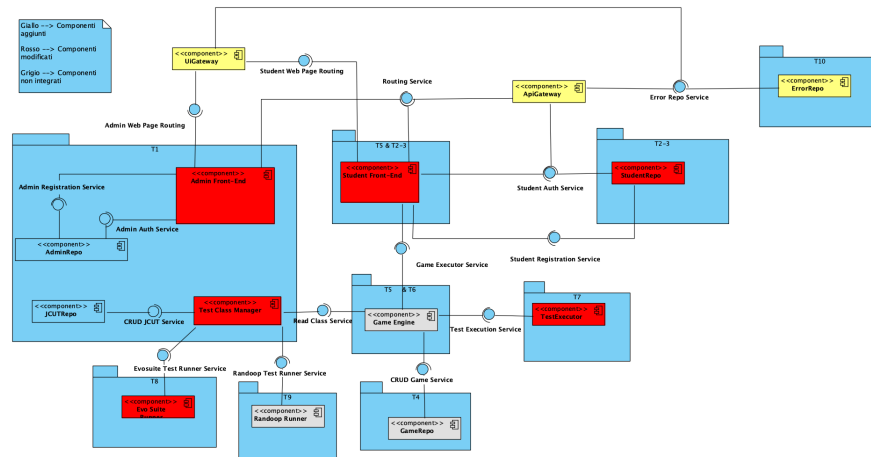


Figure 3.6: Composite Structure Diagram

3.4 Tecnologie usate

Durante lo sviluppo del software di integrazione sono stati usate le seguenti tecnologie:

- **Spring Cloud:** è un framework open-source sviluppato per semplificare lo sviluppo di applicazioni basate su microservizi nel contesto del framework Spring di Java.

In particolare permette di creare velocemente alcuni dei pattern più diffusi nei sistemi distribuiti.

- **Maven:** è uno strumento di gestione delle dipendenze e di automazione della compilazione nel mondo dello sviluppo software Java.

E' ampiamente utilizzato nella gestione dei progetti Java per

semplificare il processo di compilazione, test, distribuzione e gestione delle librerie di dipendenze.

- **Docker:** Docker è una piattaforma open-source per la creazione, la distribuzione e l'esecuzione di applicazioni in contenitori.

I contenitori sono ambienti leggeri e autonomi che includono tutto il necessario per eseguire un'applicazione, tra cui il codice, le librerie e le dipendenze.

- **Postman:** Postman è un'applicazione utilizzata dagli sviluppatori per semplificare il processo di sviluppo, testing e documentazione delle API.

È ampiamente utilizzato nella gestione di richieste HTTP, nella validazione delle risposte delle API e nell'automazione di test e flussi di lavoro legati alle API.

- **Miro:** è una piattaforma di collaborazione online che offre un ambiente virtuale per la creazione di lavagne digitali condivise. Questa piattaforma consente a gruppi di persone di lavorare insieme in tempo reale, disegnando, scrivendo, aggiungendo note e collegando idee su una lavagna virtuale.

- **ChatGPT:** chatbot basato su intelligenza artificiale e apprendimento automatico sviluppato da OpenAI. E' stato particolarmente utile come assistenza alla stesura del codice tramite il suo database di conoscenza

3.4.1 Perchè Spring Cloud ?

In questo capitolo andremo più in profondo nella scelta di usare Spring Cloud rispetto ad un altro servizio di api-gateway come Netflix-Zuul. In generale sono entrambi servizi che permettono di implementare il pattern api-gateway e quindi alcune funzionalità sono comuni però ci sono alcune differenze importanti.

Netflix-Zuul è un Api-Gateway open source sviluppato da Netflix per gestire il routing ed il filtering delle richieste tra Netflix e i suoi microservizi interni.

Spring Cloud anche è un servizio di Api-Gateway open source sviluppato dal team di Spring. E' costruito al di sopra di Spring WebFlux ed usa la libreria Reactor per il reactive programming. Permette di avere un modo semplice e flessibile per instradare le richieste supportando diversi algoritmi di routing.

Le principali differenze trovate tra Netflix Zuul e Spring Cloud sono:

- **Architettura:** Netflix Zuul è progettato su un Servlet API e usa l'I/O bloccante, limitando la scalabilità mentre Spring Cloud è progettato su Spring WebFlux e usa l'I/O non bloccante.
- **Filtri:** Spring Cloud prevede un modo più flessibile ed estensibile per definire filtri attraverso l'interfaccia WebFilter
- **Integrazione con SpringBoot:** Essendo Spring Cloud facente già parte nativamente nell'ecosistema Spring è con esso ben in-

tegrabile

Per approfondire il discorso sulle differenze si consigliano questi due articoli:

[How is Spring Cloud Gateway different from Zuul?](#)

[Differences between Netflix zuul and Spring cloud gateway](#)

Chapter 4

Error Repo

Durante la progettazione dell'integrazione del sistema si è ritenuto opportuno aggiungere un nuovo servizio che permettesse di tenere traccia di eventuali errori che possono avvenire quando vengono effettuate le richieste ai microservizi.

Tale servizio è una semplice repository consultabile da un amministratore dove sono riportati eventuali malfunzionamenti che sono stati poi catturati dall'Api-gateway.

Questa scelta è stata fatta al fine di migliorare la qualità dell'applicativo e permettere un monitoraggio da parte dell'amministratore.

4.1 Documentazione Task

Di seguito riportiamo la modellizzazione della repository tramite un class diagram:

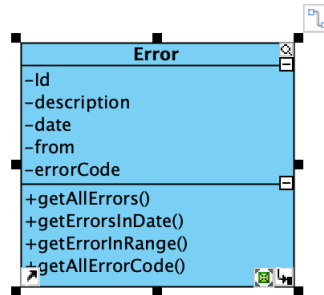


Figure 4.1: Error Repo Model

Come si può notare il servizio fa uso di una sola tabella nella quale verranno salvati i dettagli dell'errore per permettere una diagnostica e una eventuale analisi mirata verso la richiesta che ha generato il codice di errore.

L'errore può essere identificato grazie al meccanismo dei post filtri dell'api gateway che permette di analizzare la risposta restituita dal microservizio.

Al momento questo microservizio implementa le interazioni base di

- Inserimento dell'errore
- Visualizzazione dell'errore

Però in futuro non si escludono sviluppi riguardanti questo microservizio di monitoraggio.

Di seguito riportiamo anche alcuni sequence diagram esplicativi del caso d'uso:

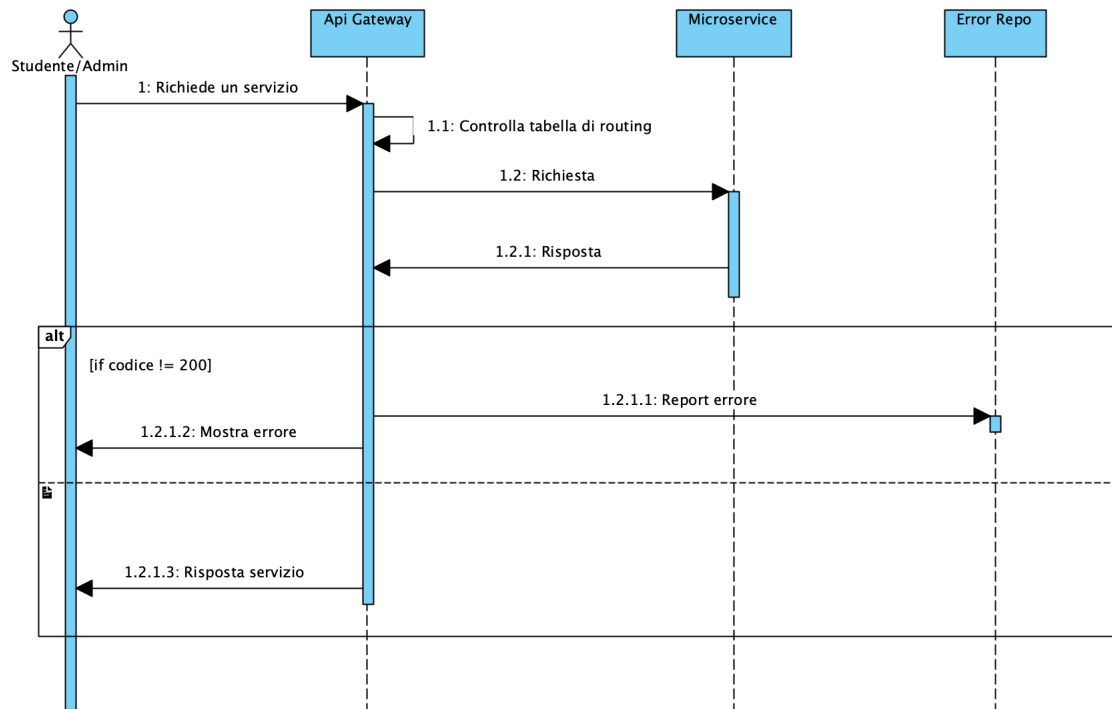


Figure 4.2: Sequence errore

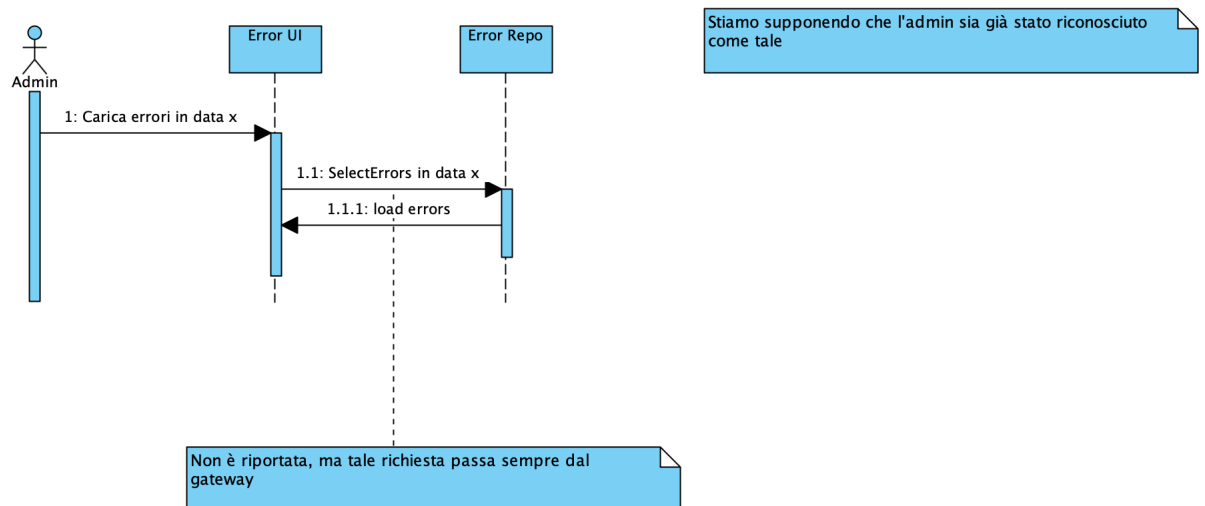


Figure 4.3: Sequence visualizzazione admin

Nel sequence 4.2 viene riportato come agisce il servizio.

Difatti l'api-gateway rappresenta per l'applicazione il punto di ingresso, per cui tutte le richieste passano per quest'ultimo servizio.

Una delle funzionalità dell'api-gateway sono i filtri che possono essere programmati.

In particolare i filtri in questo ambito sono componenti o funzionalità che consentono di manipolare, verificare, migliorare o modificare le richieste e le risposte che passano attraverso l'API Gateway.

Esistono 3 categorie principali di filtri:

- **Pre-filtri:** sono eseguiti prima che la richiesta raggiunga il servizio di destinazione. Questi filtri vengono utilizzati per manipolare o validare la richiesta in ingresso, eseguire autenticazione, autorizzazione, o altre operazioni necessarie prima che la richiesta venga inoltrata ai servizi di back-end.
- **Filtri di Routing:** sono utilizzati per determinare a quale servizio di back-end inoltrare una richiesta, basandosi su criteri come l'URL, gli header, i parametri, ecc. Questi filtri sono tipicamente eseguiti dopo i pre-filtri e sono responsabili dell'instradamento corretto delle richieste ai servizi appropriati.
- **Post-filtri:** sono eseguiti dopo che la risposta dai servizi di back-end è stata ricevuta, ma prima che venga restituita al client.

Difatti il meccanismo descritto si basa sulla programmazione di un

post-filtro che permette di analizzare la risposta del servizio.

4.2 Tecnologie usate per lo sviluppo

Di seguito riportiamo delle tecnologie aggiuntive usate per sviluppare il suddetto task:

- **Hibernate:** Hibernate è un framework open-source per la gestione di dati e persistenza in applicazioni Java. È una parte essenziale dell'ecosistema Java Enterprise Edition (Java EE) e Java Standard Edition.
E' utilizzato principalmente per semplificare l'interazione con i database relazionali in applicazioni Java.
- **Thymeleaf:** Thymeleaf è un motore di template per la creazione di pagine web dinamiche in applicazioni Java.
- **MySQL:** è un sistema di gestione di database relazionali open source ampiamente utilizzato in tutto il mondo.
È uno dei database relazionali più popolari ed è utilizzato in una vasta gamma di applicazioni.

4.3 Test dell'API

Per testare il nuovo microservizio abbiamo usato **Postman**, in particolare è stata creata una collezione con tutti i casi di test previsti.

I casi di test provati sono i seguenti:

1. Visualizza tutti gli errori
2. Visualizza gli errori in una determinata data
3. Visualizza gli errori in base allo status code
4. Carica un errore
5. Visualizza tutti gli errori in un range di tempo
6. Visualizza errori tramite l'id

Riportiamo alcuni screen di postman per evidenziare le risposte che vengono restituite, gli altri test effettuati vengono poi riportate nella tabella 4.6.

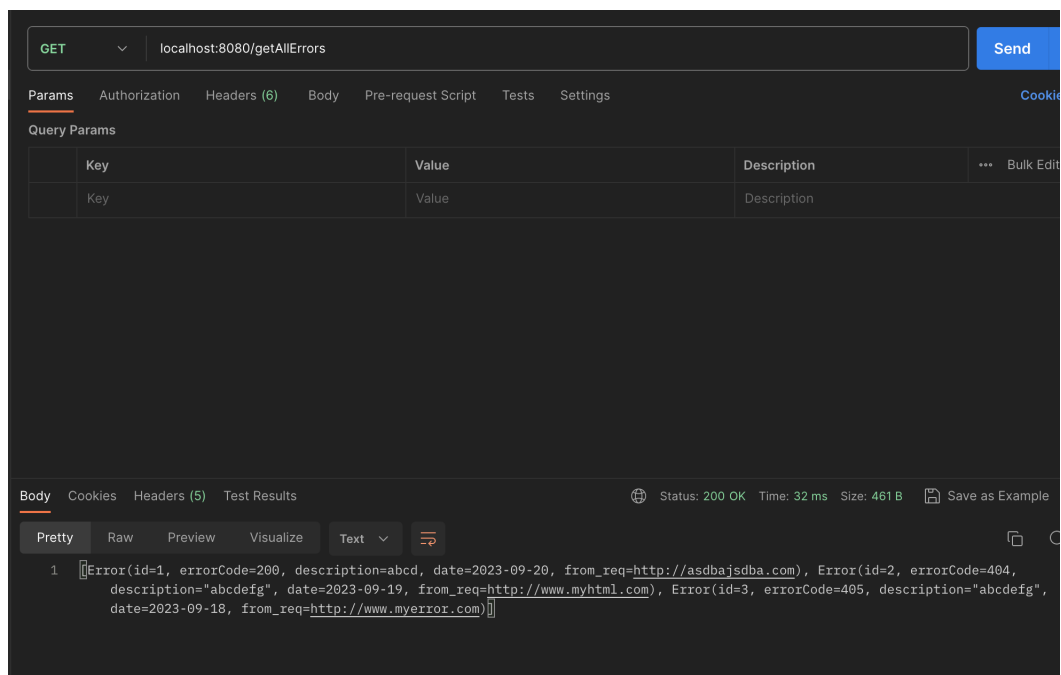


Figure 4.4: Testing Visualizza tutti gli errori

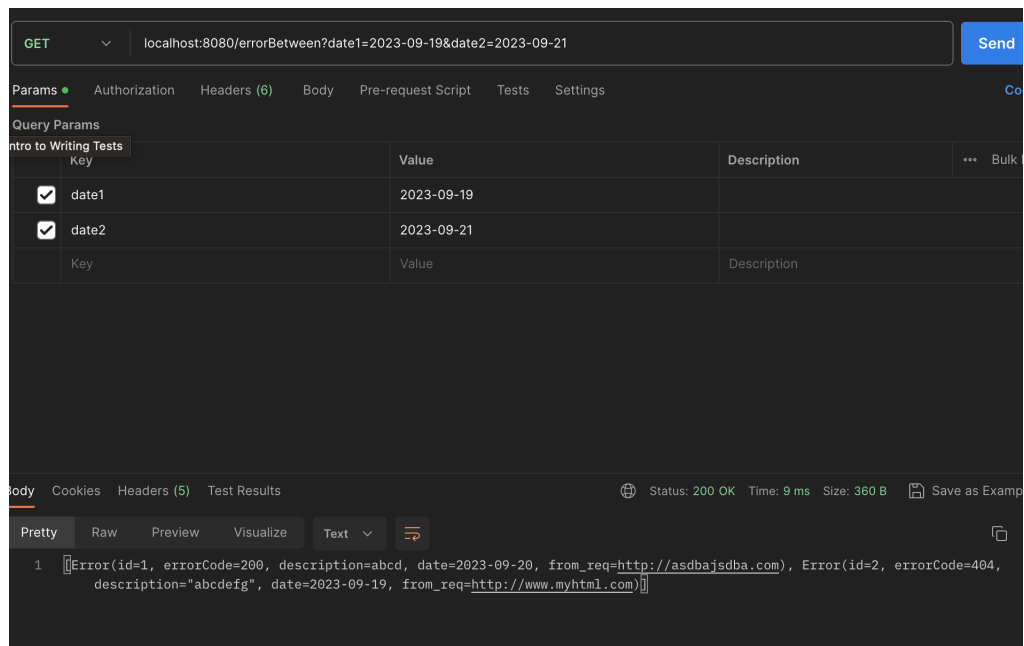


Figure 4.5: Testing Visualizza tutti gli errori in un range di tempo

Di seguito sono riportati altri test effettuati sul microservizio con il corrispettivo risultato ed esito.

Test name	Output	Esito
Invalid date in findBydate	Text 'pippo' could not be parsed at index 0	PASS
Find By Id	Optional[Error(id=1, errorCode=200, description=abcd, date=2023-09-20, from_req=http://asdbajsdba.com)]	PASS
Insert Error	Errore riportato	PASS
Insert Error without description	Description is empty	PASS
Inser Error without From request	From Req is empty	PASS

Figure 4.6: Test table

4.4 Deployment

Come spiegheremo nel prossimo capitolo, il deployment è una fase importante del processo di sviluppo di un software nella quale quest'ultimo viene distribuito e reso disponibile all'utente finale.

Di seguito riporteremo la procedura che bisogna seguire per il deploy del servizio **Error Repo**:

1. Aprire Docker desktop
2. Aprire il terminale da amministratore e porsi nella cartella del progetto
3. Digitare il comando "docker-compose up"
4. Per effettuare le richieste aprire un browser (o eventualmente usando postman) ed inserire il giusto percorso (ad esempio *localhost:8080/getAll*)

4.4.1 Deployment Diagram

Anche questo diagramma, come spiegheremo di nuovo più avanti, è molto importante.

Difatti questo diagramma ci presenta una visione ad alto livello che illustra la distribuzione fisica dei componenti del sistema software su hardware o ambienti di esecuzione senza dettagliare il funzionamento dei singoli componenti.

Di seguito si riporta il singolo Deployment Diagram per il task sviluppato:

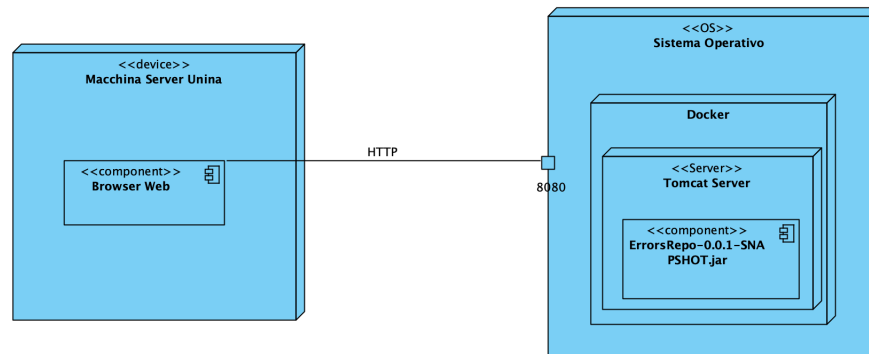


Figure 4.7: Deplyment Diagram

4.4.2 Possibili sviluppi ed utilità del microservizio

Tale microservizio è stato ideato per la necessità di monitoraggio dell'applicazione web, in particolare di errori che possono restituire i servizi e le pagine web.

Ad oggi sono implementate le due funzionalità di base ed una semplice interfaccia di interazione che permette ad un utente admin di visualizzare gli errori effettuando una ricerca per data,range oppure la visualizzazione di tutti gli errori.

Possibili sviluppi di tale microservizio possono essere:

- Aggiunta dell'operazione di deletion
- Aggiunta di una operazione di filtraggio degli errori sul microservizio

- Categorizzazione degli errori (più grave, meno grave)
- statistiche sugli errori

Chapter 5

Modifiche apportate

Durante la progettazione dell'integrazione e durante la stesura del codice sono state apportate delle modifiche ai microservizi di base, in questo capitolo riportiamo le principali modifiche apportate.

5.1 Modifiche Task 2-3

Il task 2-3 si occupava dell'autenticazione, registrazione degli studenti. Dopo il login allo studente veniva assegnato un token.

Per migliorare la sicurezza dell'applicativo si è pensato di usare il meccanismo dei token per l'autenticazione attraverso il gateway.

La modifica apportata a tale task è relativa alla verifica del token, il quale all'atto del login verrà salvato in un cookie.

Di seguito riportiamo un sequence che analizza la casistica di verifica del token richiesta dall'api-gateway.

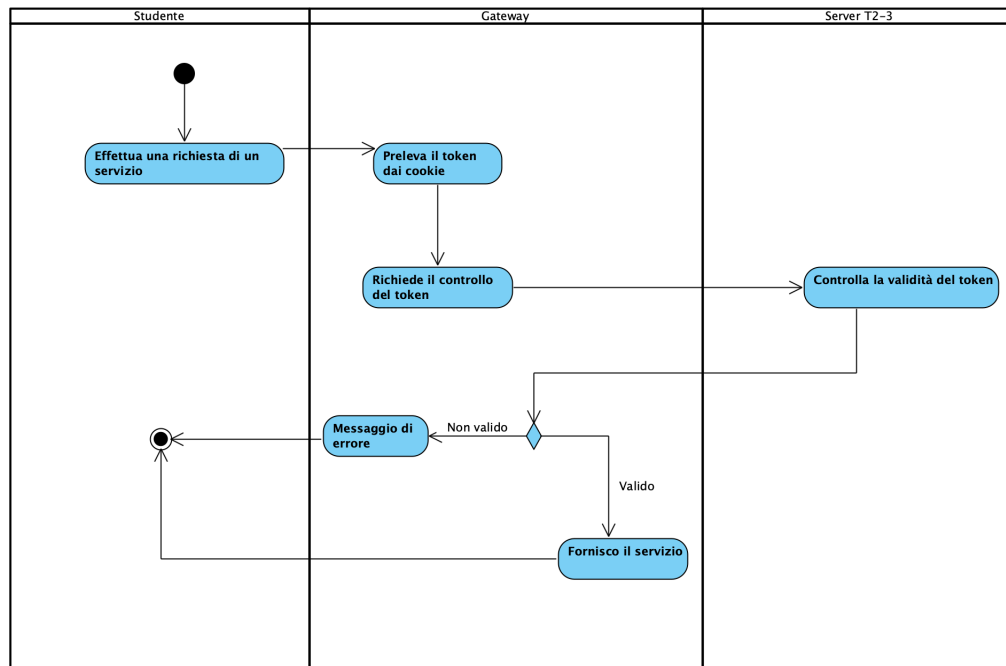


Figure 5.1: Activity per token Authentication

Tale Activity spiega l'interazione dettagliata che intercorre tra i sistemi in esame. In particolare quando lo studente fa una richiesta, che in questo caso supponiamo che non sia tra le richieste escluse da controllo del token, il gateway fa una richiesta di verifica al server T2-3 che si occuperà di validare il token che lo studente ha salvato nei cookie all'atto del login.

Tale token è criptato e la chiave di crittazione è presente nel server T2-3.

Nel caso la verifica vada a buon fine significa che lo studente è uno studente autorizzato e può usufruire del servizio, altrimenti significa che è un utente non autorizzato (con un token d'accesso non valido) e

dunque va bloccato.

Come ultima modifica è stato implementato il reindirizzamento sia a valle della registrazione (avvenuta con successo) che a valle del login.

5.2 Modifiche Task 1

In tale task sono state apportate non molte modifiche, la maggior parte delle quali a funzionalità front-end. La modifica maggiore che è stata apportata è avvenuta nella schermata di UploadFile nella quale ora il nome del file da caricare non è editabile ma tale nome viene inserito automaticamente in base al nome del file caricato.

5.3 Modifiche Task 6

In tale task sono state fatte alcune modifiche significative che elencheremo di seguito:

- Nel controller è stata aggiunta una funzione di **generazione .java** che permette di salvare su un volume (strutturando con un opportuno folder tree) il codice di test scritto dall'utente che poi sarà prelevato da Evosuite per trarne le statistiche
- E' stato usato come metodo di generazione delle statistiche di test il robot Evosuite, tale caso viene descritto tramite il se-

quence 5.3

- Aggiunta del confronto tra statistiche utente e statistiche robot

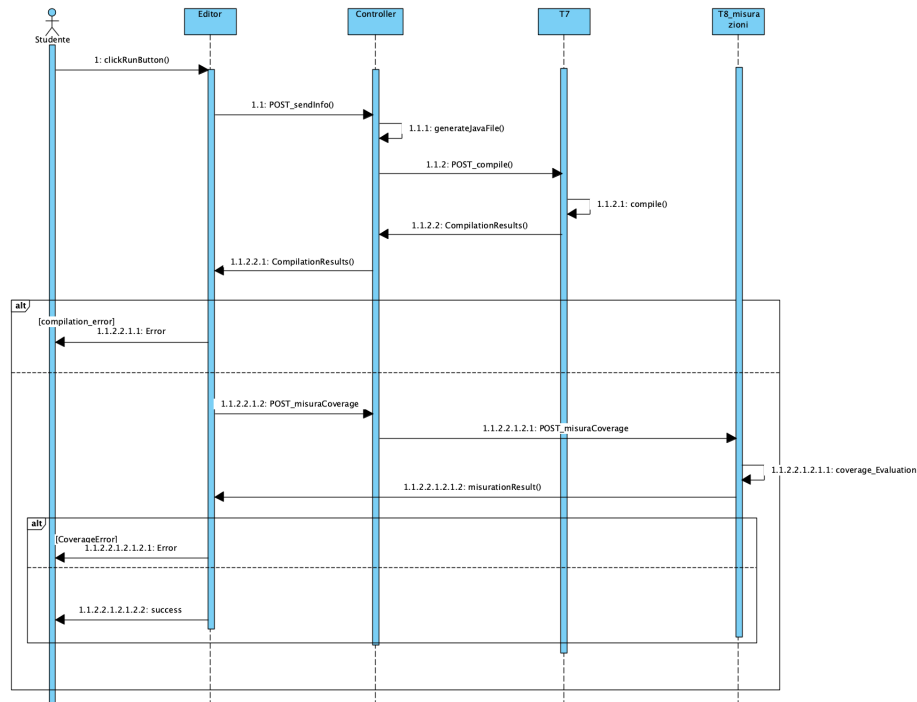


Figure 5.2: Sequence di statistiche tramite Evosuite

Come si può vedere dal digramma 5.3 dopo aver fatto la misurazione online con evosuite possiamo confrontare le statistiche che il robot ha prodotto in fase di generazione con quelle che ha prodotto in base ai test dell'utente ed in base ad un confronto possiamo decretare il vincitore.

5.4 Modifiche Task 8

Il Task 8 riguarda la generazione della Test Suite per le Classi Under Test e del file `Statistics.csv` contenente la relativa misura di

Coverage. Le modifiche apportate al task sono molteplici, di seguito sono elencate quelle più significative :

- È stato modificato il path di salvataggio dei file prodotti da EvoSuite nello script bash di generazione e misurazione: questi vengono salvati in un volume condiviso secondo la folder tree specificata per il progetto.
- Per la parte di generazione è stato aggiunto un server Node per l'integrazione con il task T1 (Dashboard Admin) : la generazione avviene quindi online, su richiesta dell'utente Admin come mostrato nel sequence 5.3. Come si può vedere ora quando l'admin carica un file (.java) in maniera automatica viene mandata una richiesta a T8 in modo che generi i casi di test secondo iol robot EvoSuite
- È stato dockerizzato l'intero task affinché sia coerente con l'architettura a microservizi della piattaforma.

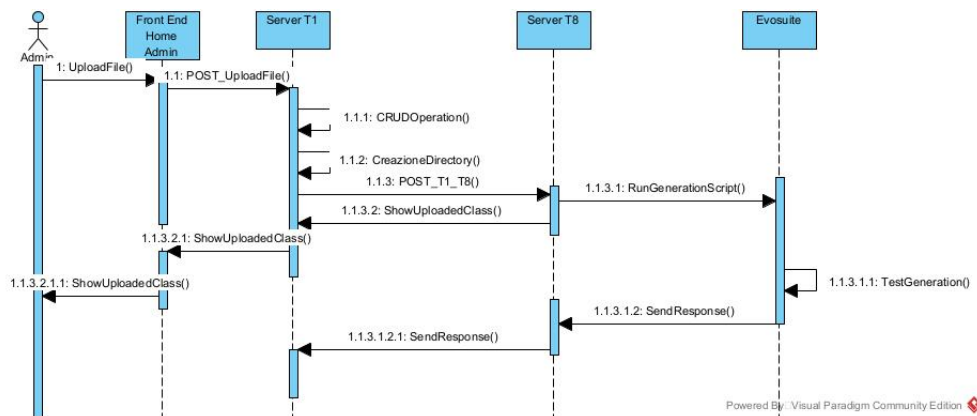


Figure 5.3: Admin - Sequence dell'esecuzione

5.5 Modifiche task 5

La principale modifica fatta al task 5 è la possibilità, per lo studente, di visualizzare le classi di cui sono stati generati i livelli con il robot. Tale interazione è descritta dal sequence 5.4

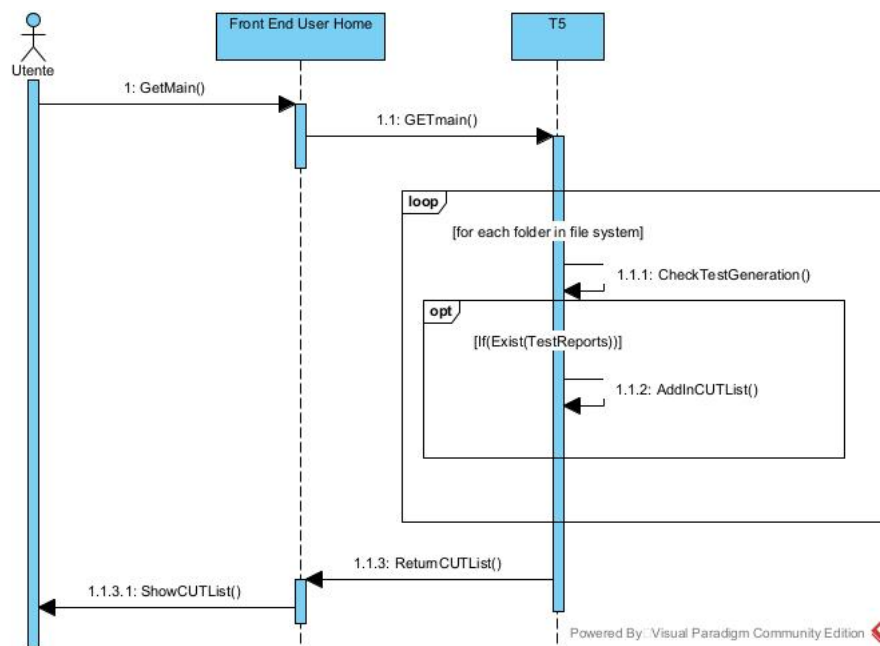


Figure 5.4: Utente - Show CUT Sequence

Come si può notare quando l'utente (che in tale caso sarà uno studente) accederà al main (operazione effettuata dopo un login effettuato con successo), viene inviata una richiesta al task 5 per recuperare tutte le classi giocabili.

Tale richiesta attiverà il processo di richiesta delle classi giocabili che saranno solo quelle che sono provviste dei livelli generati da EvoSuite.

Chapter 6

Testing

Dopo aver effettuato l'Integration Test sui singoli moduli ed aver effettuato le opportune modifiche come indicato nel Chapter 5, abbiamo provveduto ad effettuare il "Test End-to-End(E2E), noto come "Test di Sistema".

Questo approccio di testing è progettato per valutare l'intero flusso operativo dell'applicazione, garantendo che tutti i suoi componenti collaborino efficacemente nelle condizioni reali. Il software è sottoposto a una rigorosa verifica, simulando un contesto realistico che comprende l'interfaccia utente, i servizi di backend, i database e la comunicazione di rete, il tutto dalla prospettiva dell'utente finale. L'obiettivo principale qui è convalidare il comportamento complessivo dell'applicazione, che include aspetti come funzionalità, affidabilità, prestazioni e sicurezza.

Il fine ultimo del test E2E è individuare eventuali difetti o problemi

che potrebbero emergere quando le diverse parti dell'applicazione interagiscono tra loro. In genere, questo tipo di test viene eseguito dopo il test di integrazione, il quale si concentra sulla verifica dei singoli moduli, ma prima del test di accettazione dell'utente, il quale garantisce che l'applicazione soddisfi pienamente i requisiti specifici dell'utente.

6.0.1 Tool

Per l'effettuazione di questo tipo di test è stato utilizzato Selenium, uno strumento per l'automatizzazione dei test per l'applicazione web, JUnit e Google Chrome, il browser di Google usato per collegarli all'applicazione web.

Selenium

Selenium è un tool di testing automatico. Questo tool permette di automatizzare il processo di testing che a differenza di altri tool di testing non automatico, non necessita di dover inserire manualmente ogni input e registrare, sempre manualmente, ogni output per poi compararli. Il testing non automatico risulterebbe un task tedioso, spendendo inutilmente ore di lavoro direttamente proporzionali alla lunghezza e alla complessità dell'applicazione sviluppata. In particolare, la scelta presa dal gruppo è ricaduta su Selenium per vari ragioni: a differenza delle altre controparti (quali HP QTP, IBM RFT, Test Complete), Selenium è un tool di testing Open Source. E' inoltre disponibile su molteplici

sistemi operativi (Windows, Linux, Mac) e supporta una vasta quantità linguaggi di programmazione (Java, C/sharp, Ruby, Python, Perl e PHP).

JUnit

JUnit è un framework di test unitari per il linguaggio di programmazione Java. È ampiamente utilizzato dagli sviluppatori Java per scrivere e eseguire test automatici sul proprio codice. I test unitari sono piccoli frammenti di codice progettati per verificare il comportamento di una singola unità di codice, come una classe o un metodo, in modo da garantire che funzioni correttamente. Le principali caratteristiche di JUnit includono:

- **Suite di test:** JUnit consente di organizzare i test in suite, in modo che sia possibile eseguire un gruppo di test correlati in una sola volta.
- **Annotation:** JUnit fa ampio uso delle annotazioni Java per identificare i metodi di test. Ad esempio, è possibile annotare un metodo con `@Test` per indicare che è un metodo di test.
- **Assertions:** JUnit fornisce una serie di metodi di asserzione che consentono di verificare se un risultato atteso è uguale al risultato effettivo del test. Ad esempio, `assertEquals` viene utilizzato per confrontare due valori per l'uguaglianza.

- **Before e After:** È possibile utilizzare le annotazioni `@Before` e `@After` per specificare metodi di inizializzazione e pulizia che vengono eseguiti prima e dopo l'esecuzione di ciascun test. Questo è utile per impostare lo stato iniziale e ripristinare lo stato dopo i test.
- **Runners:** JUnit offre una varietà di "runners" che consentono di eseguire test in modi diversi, come eseguire test in parallelo o eseguire test basati su parametri.

JUnit ha contribuito in modo significativo a migliorare la qualità del software Java, poiché permette di scrivere test automatici che possono essere eseguiti rapidamente e facilmente ogni volta che vengono apportate modifiche al codice. Questo aiuta a individuare e correggere bug in modo tempestivo e a garantire che le nuove funzionalità non introducano errori nel software esistente.

6.1 Casi di Test

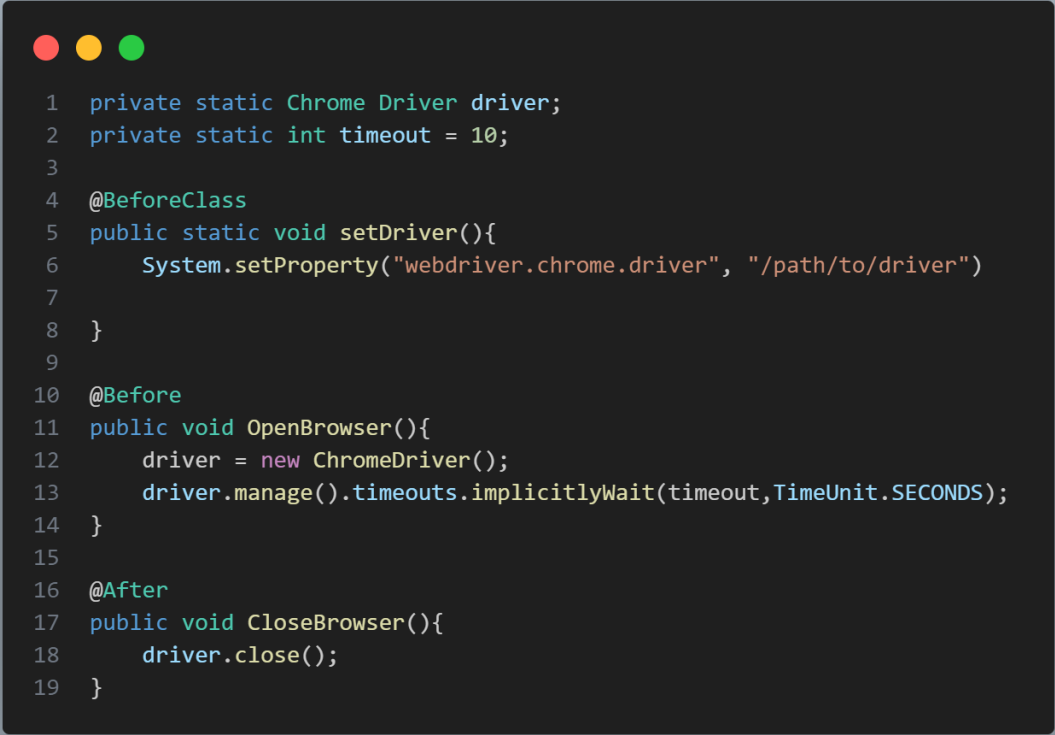
Sono elencati di seguito i diversi test condotti, i quali hanno coinvolto i flussi dell'applicazione relativi al login (Login Test), all'utilizzo dell'editor (Test dell'Editor) e all'utilizzo di EvoSuite (Test di EvoSuite). In tutti questi contesti, è possibile identificare due fasi cruciali: una concernente la configurazione dell'ambiente e l'altra dedicata ai test effettivi.

6.1.1 Login Test

Configurazione:

La fase di configurazione nel contesto dei test automatizzati con Selenium riguarda la preparazione preliminare necessaria prima di iniziare effettivamente i test. Queste operazioni iniziali includono:

- **Configurare il driver di Selenium:** Questo significa stabilire le impostazioni e le preferenze per il driver di Selenium in modo che sia in grado di comunicare con il browser. Il driver è un componente essenziale che consente a Selenium di controllare e automatizzare il browser web. È importante configurare il driver correttamente per garantire che funzioni in modo efficace.
- **Aprire il browser all'avvio di un test:** Prima di iniziare un test, è necessario aprire il browser web che verrà utilizzato per eseguire il test. Questo passaggio preparatorio assicura che il browser sia pronto per eseguire le azioni e le operazioni previste nel test.
- **Chiudere il browser al termine di un test:** Una volta che il test è stato completato con successo o anche in caso di errori, è importante chiudere il browser in modo pulito e corretto. Questo aiuta a liberare le risorse del sistema e a garantire che il browser sia pronto per l'esecuzione di altri test successivi.



```
1 private static Chrome Driver driver;
2 private static int timeout = 10;
3
4 @BeforeClass
5 public static void setDriver(){
6     System.setProperty("webdriver.chrome.driver", "/path/to/driver")
7 }
8
9
10 @Before
11 public void OpenBrowser(){
12     driver = new ChromeDriver();
13     driver.manage().timeouts.implicitlyWait(timeout, TimeUnit.SECONDS);
14 }
15
16 @After
17 public void CloseBrowser(){
18     driver.close();
19 }
```

Figure 6.1: Setup - Login

In sintesi, la fase di configurazione riguarda la preparazione iniziale delle condizioni necessarie per l'esecuzione dei test automatizzati con Selenium. Ciò include l'impostazione del driver di Selenium per la comunicazione col browser, l'apertura del browser prima di iniziare un test e la sua chiusura al termine del test. Questi passaggi preparatori sono fondamentali per garantire che i test si eseguano in modo affidabile e che il browser sia utilizzato in modo efficiente durante il processo di automazione dei test.

Autenticazione corretta:

Questo test si concentra sulla verifica dell'accesso corretto all'applicazione utilizzando le credenziali valide di uno studente già registrato. Il processo viene eseguito dal robot seguendo le seguenti operazioni:

- **Connessione alla pagina di accesso (/login):** Inizialmente, il robot si connette alla pagina di accesso dell'applicazione. Questa è la prima tappa dell'intero processo di accesso.
- **Inserimento delle credenziali (email e password):** Dopo essere arrivato alla pagina di accesso, il robot inserisce le credenziali necessarie per accedere all'applicazione. Queste credenziali generalmente includono l'indirizzo email e la password associate all'account dello studente.
- **Click sul pulsante di accesso:** Una volta inserite le credenziali, il robot fa clic sul pulsante di accesso. Questo simula l'azione dell'utente che cerca di accedere al sistema.
- **Verifica del redirect alla prima pagina dell'editor (/main):** Dopo aver cliccato sul pulsante di accesso, il robot verifica che il sistema abbia eseguito correttamente il processo di autenticazione. In particolare, controlla se l'utente viene reindirizzato alla prima pagina dell'editor dell'applicazione, che potrebbe essere denominata "/main". Questa verifica è fondamentale per

garantire che l'accesso sia avvenuto con successo e che l'utente sia ora nella sezione appropriata dell'applicazione.

Questo test automatizzato è progettato per confermare che l'accesso all'applicazione utilizzando credenziali valide funzioni senza problemi, eseguendo una serie di azioni chiave, tra cui l'inserimento delle credenziali, il clic sul pulsante di accesso e la verifica del reindirizzamento alla pagina principale dell'editor.

A screenshot of a code editor with a dark background and light-colored text. The code is a Java test method named `validCredential()`. It uses Selenium WebDriver to interact with a web application. The code includes comments in Italian. The test performs the following steps: 1. Get the login page. 2. Find the email input field and send the email address. 3. Find the password input field and send the password. 4. Click the submit button. 5. Wait for the page to change to the main page. 6. Assert that the current URL is the main page. If a timeout occurs, the test fails.

```
1 @Test
2 public void validCredential(){
3     driver.get("http://localhost/login");
4     driver.findElement(By.id("email")).sendKeys("forzanapoli1926@mail.com");
5     driver.findElement(By.id("password")).sendKeys("0simhen9");
6     driver.findElement(By.cssSelector("input[type-submit]")).click();
7
8     WebDriverWait wait = new WebDriverWait(driver, timeout);
9     String urlPaginaDiRedirezione = "http://localhost/main";
10    try{
11        wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
12    }
13    catch(TimeoutException e){
14        Assert.fail();
15    }
16    Assert.assertEquals("Test fallito! Il login non è avvenuto correttamente.", driver.getCurrentUrl(), urlPaginaDiRedirezione);
17 }
```

Figure 6.2: Login - Autenticazione Corretta

Autenticazione incorretta:

Il test si concentra sulla verifica che l'accesso sia negato quando vengono utilizzate credenziali non valide. Il robot esegue le seguenti operazioni per effettuare questo test:

- **Connessione alla pagina di accesso (/login):** Inizialmente, il robot si collega alla pagina di accesso dell'applicazione. Questa

è la prima fase del processo di prova.

- **Inserimento delle credenziali (email e password):** Dopo essere arrivato alla pagina di accesso, il robot inserisce deliberatamente delle credenziali non valide, come un indirizzo email o una password errati. Questo simula un tentativo di accesso non autorizzato.
- **Clic sul pulsante di accesso:** Una volta inserite le credenziali non valide, il robot fa clic sul pulsante di accesso. Questo emula l'azione di un utente che cerca di accedere al sistema con credenziali errate.
- **Verifica del mancato redirect alla prima pagina dell'editor (/main):** Dopo aver cliccato sul pulsante di accesso, il robot verifica che il sistema abbia negato correttamente l'accesso. In particolare, controlla se l'utente non viene reindirizzato alla prima pagina dell'editor dell'applicazione (ad esempio, "/main"). Questa verifica è cruciale per confermare che il sistema riconosce credenziali non valide e impedisce l'accesso non autorizzato.

Il test automatizzato è progettato per confermare che l'accesso all'applicazione venga negato quando vengono utilizzate credenziali non valide. Questo viene fatto attraverso l'inserimento di credenziali errate, il clic sul pulsante di accesso e la verifica che il sistema non permetta l'accesso, come evidenziato dal mancato reindirizzamento alla pagina principale

dell'editor.

A screenshot of a code editor with a dark background and light-colored text. The code is a Java test method named `InvalidCredential()`. It uses Selenium WebDriver to navigate to `http://localhost/login`, find the email and password input fields, enter the email `paolino1990@mail.com` and password `Juventus1`, and click the submit button. It then waits for the page to contain the text `Incorrect password`. If a `TimeoutException` occurs, it fails the test. Finally, it asserts that the page contains the text `Incorrect password`.

```
1 @Test
2 public void InvalidCredential(){
3     driver.get("http://localhost/login");
4     driver.findElement(By.id("email")).sendKeys("paolino1990@mail.com");
5     driver.findElement(By.id("password")).sendKeys("Juventus1");
6     driver.findElement(By.cssSelector("input[type-submit]")).click();
7
8     WebDriverWait wait = new WebDriverWait(driver,timeout);
9     try{
10         wait.until(ExpectedConditions.textToBe(by.tagName("body"),"Incorrect password"));
11     }catch(TimeoutException e){
12         Assert.fail();
13     }
14     Assert.assertEquals("Test fallito!Il login non è avvenuto correttamente.", driver.findElement(By.tagName("body")), "incorrect password");
15 }
16 }
```

Figure 6.3: Login - Autenticazione Incorretta

6.1.2 Editor Test:

Configurazione:

La fase di configurazione è un passaggio iniziale cruciale nel processo di esecuzione dei test automatizzati e coinvolge la definizione di una serie di operazioni preliminari necessarie prima che i test possano iniziare. Queste operazioni specifiche includono:

- **Impostazione del driver di Selenium:** Questa operazione implica la configurazione delle impostazioni e delle preferenze necessarie per il driver di Selenium, che è l'elemento chiave per comunicare con il browser. Assicurarsi che il driver sia impostato correttamente è fondamentale per garantire che Selenium possa interagire con il browser in modo efficace.

- **Impostazione del percorso dei download:** Se i test richiedono il download di file o risorse dal browser, è importante configurare il percorso di download in modo che i file possano essere gestiti e verificati correttamente durante l'esecuzione dei test.
- **Apertura del browser e autenticazione dell'utente all'avvio di un test:** Prima di eseguire un test, il robot automatizzato deve aprire il browser web e autenticare un utente, se necessario. Questo può implicare l'inserimento di credenziali di accesso o l'esecuzione di altre operazioni per assicurarsi che l'applicazione sia pronta per l'esecuzione dei test.
- **Chiusura del browser al termine di un test:** Alla fine di ciascun test, è importante chiudere il browser in modo pulito e corretto. Questo aiuta a liberare le risorse del sistema e a garantire che il browser sia pronto per l'esecuzione di test successivi. La chiusura del browser è un passo critico per mantenere un ambiente di test coerente e prevenire eventuali interferenze tra i test.

La fase di configurazione riguarda la preparazione iniziale delle condizioni necessarie per eseguire i test automatizzati con Selenium. Ciò include la configurazione del driver di Selenium, la gestione del percorso dei download, l'apertura e l'autenticazione del browser all'inizio di ciascun test e la chiusura del browser al termine di ogni test. Queste

operazioni preliminari sono fondamentali per garantire che i test si svolgano con successo e in modo affidabile.

```
1 private static ChromeDriver driver;
2
3 private static int timeout = 60;
4
5 @BeforeClass
6 public static void setDriver() {
7     System.setProperty("webdriver.chrome.driver", "/path/to/driver")
8 }
9
10
11
12 @Before
13 public void opensBrowser(){
14     Chromeoptions options = new Chromeoptions();
15     options.setCapability(CapabilityType.UNEXPECTED_ALERT_BEHAVIOUR, UnexpectedAlertBehaviour.ACCEPT);
16
17
18     HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
19     chromePrefs.put("profile.default_content settings. popups", 0);
20     chromePrefs.put("download.default_directory", "/path/to/download");
21     options.setExperimentalOption("prefs", chromePrefs);
22
23     driver = new ChromeDriver(options);
24     driver.manage().timeouts().implicitWait(timeout, TimeUnit.SECONDS);
25
26     driver.get("http: //localhost/login");
27     driver.findElement(By.id("email")).sendKeys("forzanapoli1926@mail.com");
28     driver.findElement(By.id("password")).sendKeys("Osimhen9");
29     driver.findElement(By.cssSelector("input [type=submit]")).click();
30
31     WebDriverwait wait = new WebDriversait(driver, timeout);
32
33     String urlPaginadiRedirezione = "http://localhost/main"
34     try {
35         wait.until(ExpectedConditions.urlToBe(urlPaginadiRedirezione));
36     } catch(Timeoutexception e) {
37         Assert. fail();
38     }
39 }
40
41
42 @After
43 public void closesrowser(){
44     driver. close();
45 }
46 }
```

Figure 6.4: Setup - Editor

Selezione Classi e Robot:

Questo test è progettato per verificare che la scelta di una classe e di un robot all'interno di un'applicazione o di un sistema avvenga correttamente. Il robot automatizzato esegue le seguenti operazioni per effettuare questa verifica:

- **Selezione della classe e del robot:** Inizialmente, il robot naviga all'interno dell'applicazione o del sistema e seleziona sia una classe che un robot specifici. Questa azione simula l'interazione dell'utente con l'interfaccia per la scelta di una classe e di un robot tra le opzioni disponibili.
- **Clic sul pulsante di conferma:** Dopo aver selezionato la classe e il robot desiderati, il robot fa clic sul pulsante di conferma. Questa azione rappresenta l'azione di conferma della scelta fatta dall'utente.
- **Verifica del redirect alla pagina di conferma (/report):** Una volta confermata la scelta, il robot verifica che il sistema abbia correttamente eseguito il redirect alla pagina di conferma, che potrebbe essere denominata "/report". Questa verifica è fondamentale per confermare che la scelta della classe e del robot sia stata registrata correttamente dal sistema e che l'utente sia ora sulla pagina di conferma.

Il test automatizzato serve a confermare che la selezione di una classe

e di un robot all'interno dell'applicazione avvenga senza errori. Ciò viene fatto attraverso l'interazione con l'interfaccia dell'applicazione, la scelta degli elementi desiderati, il clic sul pulsante di conferma e la verifica del reindirizzamento alla pagina di conferma. Questo tipo di test è utile per assicurarsi che la funzionalità di scelta e conferma all'interno dell'applicazione sia operativa e che l'utente venga portato correttamente alla pagina successiva dopo la selezione.



```
1 @Test
2 public void selection() {
3     String urlPaginaDiRedirezione = "http://localhost/report";
4
5     movetoReport(urlPaginaDiRedirezione);
6     Assert.assertEquals("Test fallito!La selezione non è avvenuta correttamente", driver.getCurrentUrl(), urlPaginaDiRedirezione)
7 }
```

Figure 6.5: Editor - Selection Class

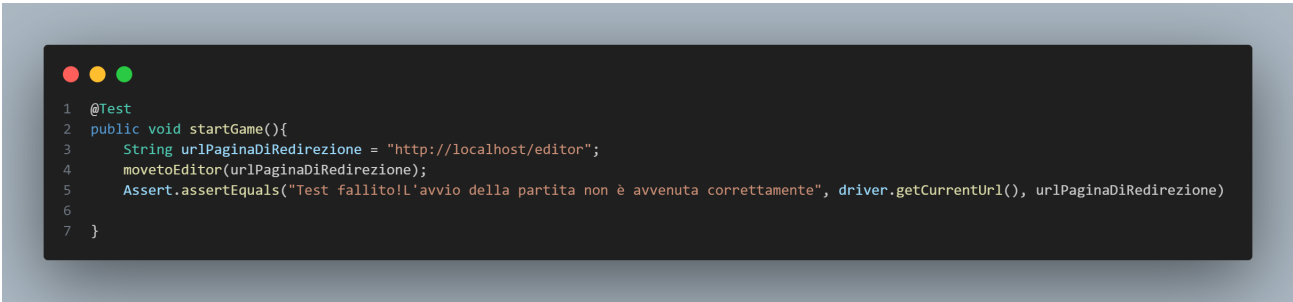
Avvio Partita:

Questo test è progettato per verificare che l'avvio di una partita all'interno di un'applicazione o di un sistema avvenga correttamente. Il robot automatizzato esegue le seguenti operazioni per effettuare questa verifica:

- **Clic sul pulsante di conferma delle scelte effettuate:** Dopo che l'utente ha effettuato tutte le scelte necessarie per iniziare una partita, il robot fa clic sul pulsante di conferma. Questo rappresenta l'azione di conferma delle scelte fatte dall'utente, indicando che è pronto per iniziare la partita.

- **Verifica del redirect all'editor (/editor):** Dopo aver confermato le scelte, il robot verifica che il sistema abbia correttamente eseguito il redirect alla pagina dell'editor, che potrebbe essere denominata "/editor". Questa verifica è essenziale per confermare che l'avvio della partita sia stato inizializzato correttamente e che l'utente sia ora sulla pagina dell'editor pronta per iniziare la partita.

Il test automatizzato è finalizzato a confermare che l'avvio di una partita all'interno dell'applicazione o del sistema sia senza errori. Ciò viene fatto attraverso l'interazione con l'interfaccia dell'applicazione, il clic sul pulsante di conferma delle scelte e la verifica del reindirizzamento alla pagina dell'editor. Questo tipo di test è utile per garantire che la funzionalità di avvio della partita funzioni correttamente e che l'utente sia portato alla pagina dell'editor in modo appropriato.

A screenshot of a code editor with a dark background and light-colored text. The code is a Java test method named `startGame()`. It includes a `@Test` annotation, a `String` variable `urlPaginaDiRedirezione` set to `"http://localhost/editor"`, a call to `moveToEditor(urlPaginaDiRedirezione)`, and an `Assert.assertEquals` statement to verify the current URL. The code is numbered from 1 to 7 on the left side.

```
1 @Test
2 public void startGame(){
3     String urlPaginaDiRedirezione = "http://localhost/editor";
4     moveToEditor(urlPaginaDiRedirezione);
5     Assert.assertEquals("Test fallito! L'avvio della partita non è avvenuta correttamente", driver.getCurrentUrl(), urlPaginaDiRedirezione)
6
7 }
```

Figure 6.6: Editor - Avvio Partita

Compilazione Classe Utente:

Questo test è progettato per verificare che la compilazione di una classe di test scritta dall'utente avvenga correttamente. Il robot automatizzato esegue le seguenti operazioni per effettuare questa verifica:

- **Attesa del caricamento della classe da testare:** Inizialmente, il robot attende che la classe di test scritta dall'utente sia caricata nell'ambiente o nell'editor dell'applicazione. Questo assicura che la classe sia pronta per essere compilata.
- **Click sul pulsante di compilazione:** Una volta che la classe di test è pronta, il robot fa clic sul pulsante di compilazione. Questa azione simula l'azione di un utente che richiede all'applicazione di compilare la classe di test.
- **Verifica della visibilità del risultato della compilazione:** Dopo aver avviato la compilazione, il robot verifica che il risultato della compilazione sia visibile nell'interfaccia dell'applicazione. Questa verifica è fondamentale per assicurarsi che la compilazione abbia avuto successo e che il risultato sia disponibile per l'utente.

Il test automatizzato è finalizzato a confermare che la compilazione di una classe di test scritta dall'utente avvenga senza errori. Ciò viene fatto attraverso l'interazione con l'interfaccia dell'applicazione, l'attesa del caricamento della classe, il clic sul pulsante di compilazione e la

verifica della visibilità del risultato della compilazione. Questo tipo di test è utile per garantire che la funzionalità di compilazione delle classi di test sia operativa e che i risultati siano correttamente visualizzati per l'utente.

```
1  @Test
2  public void compile() {
3      String urlPaginaDiRedirezione = "http://localhost/editor" ;
4      moveToEditor(urlPaginaDiRedirezione);
5
6      WebDriverwait wait = new WebDriverwait(driver, timeout);
7
8      try {
9          wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
10 textarea + div > * div.CodeMirror-code > * "), 1));
11      } catch(TimeoutException e) {
12          Assert.fail();
13      }
14
15      driver.findElement(By.id("compileButton")).click();
16
17      try {
18          wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#console-
19 textarea + div > * div.Codemirror-code > *"), 1));
20      } catch(TimeoutException e) {
21          Assert.fail();
22      }
23  }
```

Figure 6.7: Editor - Compilazione Classe Utente

Submit della partita:

Questo test è progettato per verificare che il tentativo dell'utente venga consegnato e che la partita venga processata correttamente. Il robot automatizzato esegue le seguenti operazioni per effettuare questa ver-

ifica:

- **Attesa del caricamento della classe da testare:** Inizialmente, il robot attende che la classe di test scritta dall'utente sia caricata nell'ambiente o nell'editor dell'applicazione. Questo assicura che la classe sia pronta per essere utilizzata per il tentativo di gioco.
- **Click sul pulsante di submit (consegna):** Dopo che la classe di test è pronta, il robot fa clic sul pulsante di submit (consegna). Questa azione simula l'azione dell'utente che invia il proprio tentativo o la propria soluzione per la partita.
- **Verifica della visibilità dell'esito della partita:** Dopo aver consegnato il tentativo, il robot verifica che l'esito della partita sia visibile nell'interfaccia dell'applicazione. Questa verifica è fondamentale per assicurarsi che il tentativo dell'utente sia stato processato correttamente e che l'esito della partita sia reso disponibile all'utente.

Il test automatizzato serve a confermare che il tentativo dell'utente venga consegnato con successo e che la partita venga processata in modo accurato. Ciò viene fatto attraverso l'interazione con l'interfaccia dell'applicazione, l'attesa del caricamento della classe, il clic sul pulsante di consegna e la verifica della visibilità dell'esito della partita. Questo tipo di test è utile per garantire che la funzionalità di con-

segna dei tentativi e di elaborazione dei risultati della partita funzioni correttamente per l'utente.

A screenshot of a code editor with a dark background and light-colored text. The code is a Java test method annotated with `@Test`. It uses Selenium WebDriver to navigate to a URL, wait for a sidebar text area to appear, click a 'runButton', wait for a console text area to appear, and finally click the button again. The code is numbered from 1 to 24 on the left side.

```
1  @Test
2  public void run() {
3      String urlPaginaDiRedirezione = "http://localhost/editor";
4      moveToEditor(urlPaginaDiRedirezione);
5
6      WebDriverwait wait = new WebDriverwait(driver, timeout);
7
8      try {
9          wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#sidebar-
10         textarea + div > * div.CodeMirror-code > *"), 1)
11      } catch(TimeoutException e) {
12          Assert.fail();
13      }
14
15      driver.findElement(By.id("runButton")).click();
16
17      try {
18          wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.cssSelector("#console-
19         textarea2 + div > * div.CodeMirror-code > *"), 1))
20      } catch(TimeoutException e) {
21          Assert.fail();
22      }
23  }
24
```

Figure 6.8: Editor - Submit

6.2 Evosuite:

Questo test è progettato per valutare il comportamento del sistema durante la generazione dei test suite utilizzando Evosuite. In particolare, il test mira a verificare come il sistema gestisce i casi in cui l'amministratore fornisce un nome di classe diverso da quello effettivo attraverso il modulo di generazione dei test. L'obiettivo principale è

confermare che il sistema è in grado di rilevare questa discrepanza.

- **Procedura del Test:**

Inizializza il driver del browser (Chrome) utilizzando Selenium. Naviga alla pagina web del sistema contenente il modulo di generazione dei test suite.

- **Esecuzione del Test:** Individua il campo di input per il nome della classe nell'amministratore del modulo. Inserisci deliberatamente un nome di classe diverso da quello effettivo nel campo di input. Trova il pulsante per avviare la generazione dei test suite e fai clic su di esso.

- **Verifica:** Verifica se è stato visualizzato un messaggio di errore o avviso sulla pagina. Assicurarsi che il messaggio presenti un avviso appropriato che informa l'utente sulla discrepanza tra il nome fornito e il nome effettivo della classe.

- **Esito del Test:** Se il messaggio di errore o avviso è stato visualizzato in modo corretto, il test è considerato riuscito. Se il messaggio non è stato visualizzato o è stato visualizzato in modo errato, il test è considerato non riuscito.

```
1 public class EvosuiteTest {
2     private WebDriver driver;
3
4     @Before
5     public void setUp() {
6         // Imposta il percorso del driver di Chrome
7         System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
8
9         // Avvia il browser Chrome
10        driver = new ChromeDriver();
11    }
12
13    @Test
14    public void testAdminClassNameMismatch() {
15        // Naviga alla pagina web del tuo sistema
16        driver.get("http://localhost:90/home_admin");
17
18        // Trova il campo di input per il nome della classe nell'admin form
19        WebElement adminClassNameInput = driver.findElement(By.id("adminClassName"));
20
21        // Inserisci un nome di classe diverso da quello effettivo
22        adminClassNameInput.sendKeys("NomeClasseDiverso");
23
24        // Trova il pulsante per avviare la generazione dei test suite
25        WebElement generateButton = driver.findElement(By.id("generateButton"));
26
27        // Clicca sul pulsante per generare i test suite
28        generateButton.click();
29
30        // Verifica che ci sia un messaggio di errore o un avviso sulla pagina
31        WebElement errorMessage = driver.findElement(By.id("errorMessage"));
32
33        assert errorMessage.isDisplayed() : "Il test ha fallito: Errore non rilevato.";
34    }
35
36    @After
37    public void CloseBrowser() {
38        // Chiudi il browser
39        driver.quit();
40    }
41 }
```

Figure 6.9: Evosuite - Check Nome

6.2.1 Risultati

In tale sottosezione si presentano i risultati dei casi di test sopracitati:

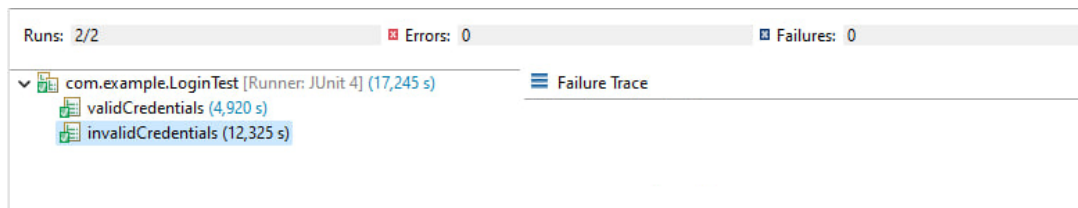


Figure 6.10: Test Login

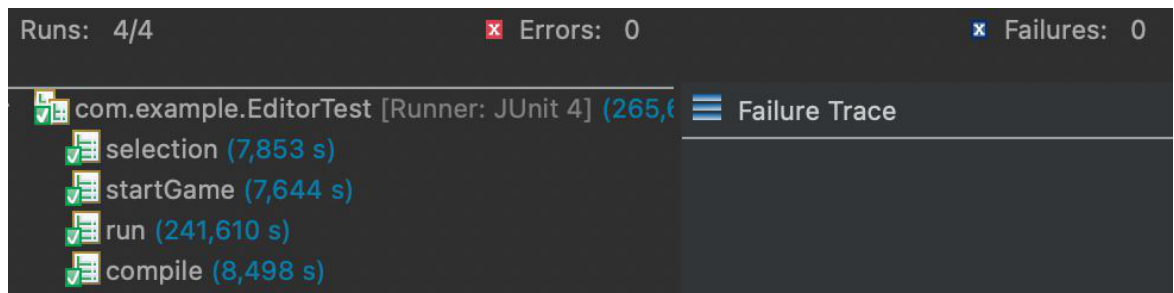


Figure 6.11: Test Editor

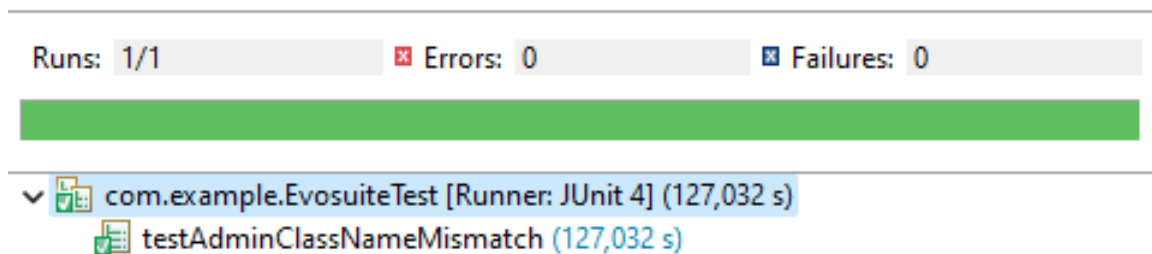


Figure 6.12: Test Evosuite

Chapter 7

Deployment

La fase di Deployment rappresenta l'ultimo passo cruciale nel ciclo di vita di un progetto. In questa fase, il sistema sviluppato viene preparato per essere distribuito e reso disponibile per l'uso effettivo da parte degli utenti finali. Questa fase comprende una serie di attività e processi chiave, tra cui:

- **Preparazione dell'Ambiente:** Prima del deployment, è fondamentale preparare l'ambiente in cui il sistema verrà installato. Ciò può includere l'installazione e la configurazione di server, database, reti e tutti gli altri componenti necessari.
- **Pacchettizzazione dell'Applicazione:** Il software o l'applicazione viene confezionato in un formato appropriato per la distribuzione, come container Docker, pacchetti RPM/DEB per sistemi Linux o file di installazione per sistemi Windows.

- **Test di Deployment:** Prima del rilascio effettivo, è importante eseguire test di deployment per garantire che il sistema possa essere installato e avviato correttamente nell'ambiente di produzione. Questi test possono rilevare problemi di configurazione o dipendenze mancanti.
- **Deployment in Produzione:** Una volta superati i test con successo, il software viene distribuito nell'ambiente di produzione. Questo può essere fatto in modo graduale o tramite un rilascio completo, a seconda delle esigenze e delle strategie di deployment.

La fase di Deployment è critica per garantire che il sistema sia pronto per essere utilizzato in modo efficace e sicuro dagli utenti finali. Un deployment ben pianificato e gestito può contribuire al successo del progetto, assicurando che il software soddisfi le aspettative degli utenti e funzioni senza intoppi nell'ambiente di produzione.

7.1 Deployment Diagram

Un Deployment Diagram, è diagramma utilizzato per rappresentare la disposizione fisica dei componenti software e hardware in un sistema distribuito. Questo tipo di diagramma mostra come i diversi elementi del sistema, come server, dispositivi, nodi di rete e software, sono posizionati fisicamente e come comunicano tra loro attraverso connessioni

di rete o canali di comunicazione.

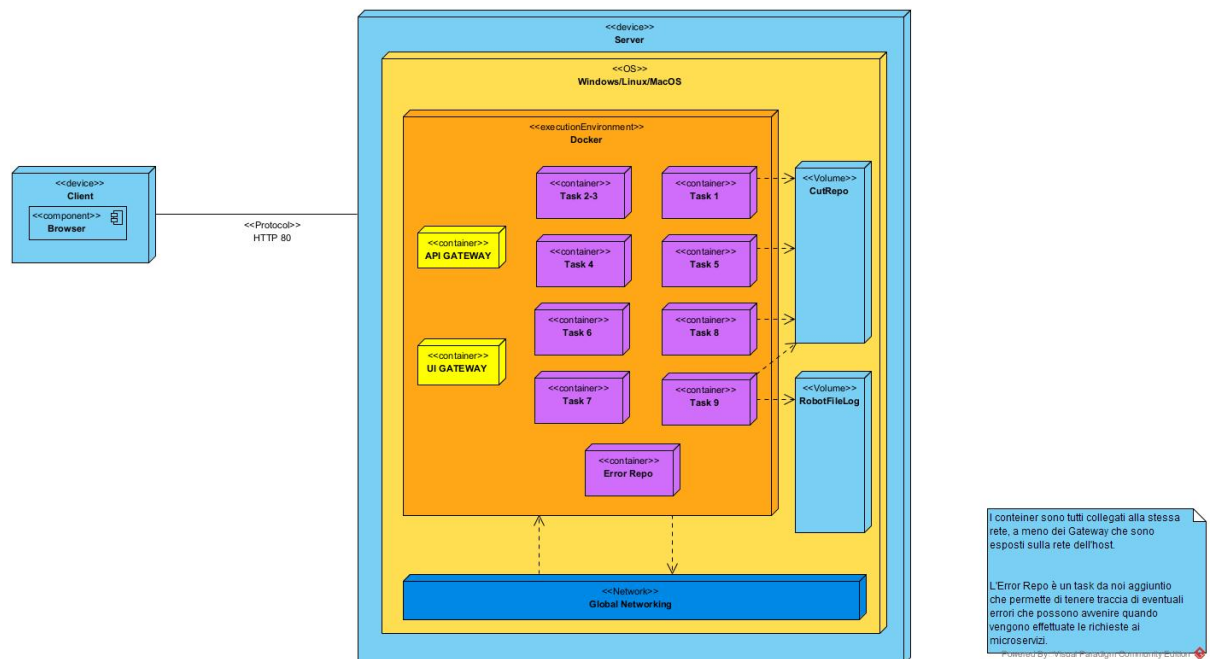


Figure 7.1: Deployment Diagram

Nel nostro diagramma, si possono notare due elementi in particolare:

- **Global Network:** I container sono tutti collegati alla stessa rete, a meno dei Gateway che sono esposti sulla rete dell'host. Ciò permette di isolare i container dalle rete dell'host e al tempo stesso permettere la comunicazione tra di essi. I Gateway devono invece gestire le richieste.
- **Volumi:** Il Volume CutRepo è condiviso da T1, T5 e T8 in quanto è necessario condividere tra di loro la stessa porzione di File System.

Il volume RobotFileLog invece è stato creato per avere un resoconto d'esecuzione dei robot.

Per completezza di informazione si è deciso di riportare una tabella di compatibilità delle immagine legata alle architetture.

Image	Os/architecture
T1 - OpenJDK-21-rc-slim	linux/amd64
T2_3 - OpenJDK	linux/amd64
T5 - OpenJD-17-jdk;	linux/amd64
T6 - openjdk.22-ea-17-slim	linux/amd64
T8 - OpenJDK-8-jdk	linux/amd64

Figure 7.2: Tabella di Compatibilità

I dati riportati all'interno della tabella sono presi da Docker Hub dove sono riportati anche le vulnerabilità dei singoli task.

Chapter 8

Installazione

In questo capitolo, viene descritta la procedura di installazione dell'applicazione completa, che è stata semplificata e resa automatica grazie all'uso di Docker. Per rendere il processo di installazione agevole, si è ricorsi alla piattaforma DockerHub: dopo aver effettuato la *push* delle immagini dei singoli microservizi sulla repository con visibilità pubblica, è stato creato un *docker-compose* file attraverso il quale è possibile installare la piattaforma a microservizi sull'host personale.

Si riporta di seguito la procedura d'installazione:

1. operazione di *clone* della cartella *DockerHubInstall* nella repository del progetto (vi saranno i file *docker-compose*, *initMongo.js* e i due installer bash e batch);
2. verifica d'installazione di python3 e del pacchetto *socket* per il corretto avvio dell'installer;

3. dopo essersi posizionati con il terminale nella directory appena citata, basterà eseguire il file *.py*. Per quanto riguarda windows, sarà necessario avviare il cmd come amministratore, dirigersi nella cartella dove è situato il file *.py* e *.yaml* e inserire il comando:
`python installer.py`.

L'installer andrà a modificare il *docker-compose* file presente (impostando come variabile d'ambiente per ogni microservizio l'ip dell'host sul quale partirà la piattaforma), avvierà l'operazione di *pull* di tutte le immagini dalla repository *Docker Hub*, configurerà i volumi e la *custom network* necessarie per il corretto funzionamento della piattaforma e inizializzerà il database mongo presente tramite il relativo *initializer*. La piattaforma sarà accessibile all'indirizzo ip mostrato sulla console all'avvio dello script.

Chapter 9

Conclusioni e future implementazioni

Tutti gli obiettivi proposti sono stati raggiunti con successo. L'utilizzo di **Spring Cloud** ha giocato un ruolo cruciale nell'ottenere un miglioramento significativo in diversi attributi di qualità del nostro sistema.

- **Flessibilità tramite il Routing dei Gateway:** L'implementazione del meccanismo di **routing attraverso i gateway** ha notevolmente migliorato la **flessibilità** del sistema. Questa architettura consente di gestire dinamicamente il flusso del traffico tra i microservizi, consentendo una maggiore agilità nelle operazioni di routing.
- **Performance con Load Balancing:** L'utilizzo di un **Load Balancer** ha portato a un significativo aumento delle **perfor-**

mance del sistema. Il bilanciamento del carico distribuisce in modo uniforme le richieste tra i microservizi, migliorando la velocità di risposta complessiva del sistema.

- **Autenticazione Gestita dai Gateway:** L'implementazione di un sistema di **autenticazione gestito dai gateway** ha contribuito a ridurre l'overhead dei singoli microservizi. Ciò ha portato a un aumento della **modificabilità** del sistema. Inoltre, l'uso delle configurazioni **application.properties** ha reso la configurazione del gateway altamente parametrica, migliorando ulteriormente l'adattabilità alle esigenze specifiche del sistema.
- **Miglioramento della Monitorabilità con Error Repo:**
L'introduzione del nuovo componente **Error Repo** ha notevolmente migliorato la **monitorabilità** del sistema. Questo componente consente di evidenziare e registrare le failure nei microservizi, semplificando la diagnosi e il debugging delle problematiche.
- **Miglioramento della Portabilità tramite Docker:** L'adozione del sistema Docker ha apportato notevoli miglioramenti alla **portabilità** del nostro sistema. Ha permesso una collaborazione agevole su ambienti omogenei, garantendo un isolamento efficace. Questo isolamento può essere facilmente trasportato anche in fase di deployment, contribuendo così a migliorare l'attributo di

deployability del sistema.

- **Miglioramento della modificabilità:** attraverso l'uso di una architettura a microservizi è notevolmente migliorata la modificabilità del sistema permettendo una più agevole implementazione di nuove funzionalità

9.1 Spunti e possibili implementazioni

Le possibili implementazioni che abbiamo analizzato possano essere di estrema utilità e completezza al progetto possono essere le seguenti :

- **Generazione di livelli multipli per i Robot con l'uso di Kubernetes:** Attualmente, il sistema dei robot è progettato per sfruttare delle directories nel file system statiche: la generazione concorrente di più test suites crea conflitto. Per far fronte alla problematica, si potrebbe optare per due soluzioni:
 - **Modifica dell'environment di generazione:** attraverso la modifica dello script bash di generazione, si potrebbe utilizzare una struttura di cartelle generata a runtime per ogni richiesta di generazione (evitando conflitti sugli artefatti intermedi).
- Per tale eventualità, essendo il task CPU-bound, si consiglia di effettuare un dependability test del server che ospita la

piattaforma per accertarsi che il singolo container gestisca correttamente il task.

- **Ricorso a tool di orchestrazione:** la tecnologia degli orchestratori (kubernetes, docker swarm) consentirebbe di associare un trigger ad una richiesta di generazione, istanziando nuovi container a runtime. Questa soluzione non prevede la modifica al task originale e non genererebbe sovraccarichi al singolo container.
- **Autenticazione con token per gli Admin:** Per migliorare la sicurezza del sistema, sarebbe consono implementare un sistema di autenticazione basato su token per gli Admin. Ciò garantirebbe l'accesso alla funzionalità di amministrazione ai soli utenti autorizzati.
- **Rendering dei risultati di gioco:** Per migliorare l'esperienza degli utenti, è possibile studiare un modulo grafico per la gestione dei risultati dei giochi. Questo modulo fornirà statistiche dettagliate, classifiche e altre informazioni pertinenti per i giocatori.
- **Miglioramento dell'Availability tramite la redundancy:** Per garantire una disponibilità elevata del sistema, è possibile una strategia di ridondanza. Questa strategia includerà duplicazione di server, sistemi e risorse critiche per garantire la conti-

nuità delle operazioni in quanto il sistema da noi implementato presenta un Single Point of Failure.

- **Controlli sintattici dell’editor:** Per migliorare l’esperienza di gioco utente e la fruibilità della piattaforma si potrebbe implementare il rilevamento degli errori di sintassi, delle incoerenze nel codice o nei testi tramite analisi statica del codice, analisi grammaticale o altri metodi di analisi dei contenuti. Aggiungere la capacità di fornire suggerimenti e correzioni automatiche quando vengono rilevati errori. Questo potrebbe richiedere la creazione di un sistema di completamento automatico, correzione ortografica o grammaticale. Inoltre si potrebbe implementare un sistema di highlighting per le righe di codice coperte dai test.
- **Inibire l’indirizzamento non autorizzato ai servizi:** Poiché il codice è stato sviluppato come beta le pagine sono esposte e direttamente raggiungibili, secondo l’architettura sviluppata questo non deve essere possibile poiché tutte le richieste devono essere indirizzate al gateway che effettua l’autenticazione tramite token. Per cui si suggerisce in fase di distribuzione del software di provvedere meccanismi di inibizione di indirizzamento ai servizi e alle pagine non autorizzato attraverso l’uso dei seguenti comandi:

- Per le pagine html bisognerebbe usare il seguente codice

javascript *document.referrer* che permette di identificare chi ha indirizzato tale pagina che secondo l'architettura deve essere il gateway e nessun altro.

- Per i servizi si dovrebbe prevedere una cosa simile tramite *request.getHeader("referrer");* che permette di indicare chi ha indirizzato tale servizio e quindi controllare che questo sia stato fatto tramite il gateway