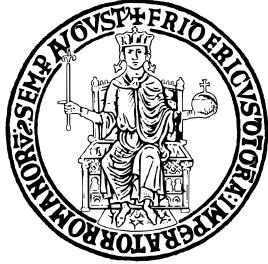


UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE
DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

**DOCUMENTAZIONE T10-G37
SOFTWARE ARCHITECTURE DESIGN
2022/23**

Prof.ssa

Anna Rita FASOLINO

Candidati

Alberto Aimone M63/1508

Valerio Di Domenico M63/1465

Genny Fedele M63/1422

Anno Accademico 2022–2023

Contents

1	Introduzione	1
1.1	Caso di studio	1
1.1.1	Task assegnato	2
1.1.2	Metodologie adottate	2
1.2	Planning	3
1.2.1	Diario	8
1.3	Workflow diagram	14
2	Analisi dei requisiti	15
2.1	User Stories	15
2.1.1	Storie Utente non complete	16
2.1.2	Criteri di accettazione	17
2.2	Modello dei casi d'uso	23
2.2.1	Attori	24
2.3	Scenari	24
2.3.1	Logout	24
2.3.2	Create game	25
2.3.3	Write code	25
2.3.4	Inspect ClassUnderTest	26
2.3.5	Run UserTest	26
2.3.6	Run with JaCoCo	27
2.3.7	Compile	27
2.3.8	Start Game	28
2.3.9	Add Class	29
2.4	Dipendenze tra i Task	30
3	Progettazione	31
3.1	L' architettura a microservizi scelta	31
3.1.1	Gateway Pattern	31
3.1.2	UI Gateway	31
3.1.3	API Gateway	31
3.2	Component Diagram	33
3.2.1	UI Gateway	33

3.2.2	API Gateway	33
3.2.3	Student Front End	34
3.2.4	Student Repository	34
3.2.5	Admin Front End	34
3.2.6	Admin Repository	34
3.2.7	Game Engine	34
3.2.8	Student Test Runner	34
3.2.9	Test Class Manager	35
3.2.10	Java Class Under Test Repository	35
3.2.11	Randoop Runner	35
3.2.12	EvoSuite Runner	35
3.2.13	Game Repository	35
3.3	Composite Structure Diagram	36
3.3.1	Modifiche Apportate	37
3.4	Gateway Sequence Diagrams	46
3.4.1	UI Gateway	46
3.4.2	API Gateway	47
3.5	Sequence Diagrams	48
3.5.1	Inspect ClassUnderTest	48
3.5.2	Create Game	49
3.5.3	Run	51
3.5.4	Run UserTest	52
3.5.5	Run with JaCoCo	53
3.5.6	Compile	54
3.5.7	Add class	55
3.6	Activity Diagrams	56
3.6.1	Add Class	56
4	Deployment	57
5	Testing	59
5.1	Tool Utilizzati	59
5.2	Casi di Test	59
5.2.1	Login Test	60
5.2.2	Editor Test	62
5.3	Risultati	69
6	Installazione	71
6.1	Passo 1	71
6.2	Passo 2	72
6.3	Passo 3	72

7 Guida alle future integrazioni	73
7.1 Integrazione Container	73
7.2 Integrazione con UI Gateway	74
7.3 Integrazione con API Gateway	74
7.4 Integrazione Installer	74

-1-

Introduzione

Il testo qui presentato documenta l'attività di integrazione dei microservizi che compongono il gioco educativo "Man vs Automated Testing Tools challenges" ideato nel contesto del progetto ENACTEST.

1.1 Caso di studio

Il caso di studio prevede l'implementazione di un primo scenario di gioco: un giocatore può effettuare una partita contro un singolo avversario robotico testando una singola classe. In particolare, all'atto della creazione della partita, il giocatore sceglie quale tool sfidare. In fase preliminare, gli strumenti automatici scelti sono EvoSuite e Randoop.

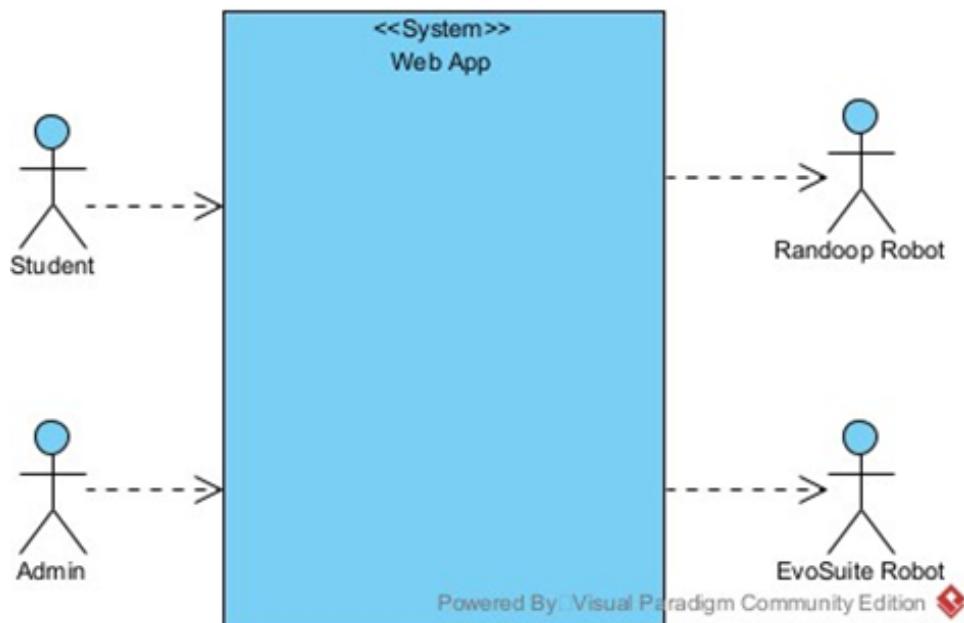


Figure 1.1: Diagramma di contesto ad alto livello

Nota sulla documentazione

Nel tentativo di offrire una panoramica completa del sistema software, ci si è sforzati di incorporare all'interno di questa documentazione elementi significativi provenienti da documenti prodotti da altri team. Tuttavia, per una comprensione più approfondita, si consiglia di consultare innanzitutto la documentazione generata dai singoli Task. Questo permetterà di comprendere le modifiche che abbiamo apportato per far funzionare tali task, nonché le variazioni nel loro funzionamento in questo contesto.

1.1.1 Task assegnato

L'architettura della Web Application è stata suddivisa in 8 componenti, da T1 a T9. Il compito assegnato a T10 è quello di integrare il task T8 a partire dall'integrazione effettuata dal task T10-G40 al fine di ottenere la versione finale funzionante dell'applicazione web. È importante notare che i vari task sono stati sviluppati da gruppi diversi in più occasioni. I dettagli dei task specifici da integrare sono riportati nella tabella (figura 1.2).

Cluster	T1	T2-3	T4	T5	T6	T7	T8	T9	Task di Integrazione
1	G11	G1	G18	G2	G12	G31	G21	G19	T10

Figure 1.2: Tabella task da integrare

1.1.2 Metodologie adottate

Per lo sviluppo del nostro software abbiamo adottato un **approccio agile** basato sul paradigma **SCRUM** ovvero un Framework agile per la gestione iterativa ed incrementale del ciclo di sviluppo del software. La nostra metodologia di lavoro si è basata su *sprint*, durante i quali i componenti del gruppo si sono concentrati su un insieme di funzionalità specifiche. Ogni sprint è stato preceduto da una pianificazione in cui sono stati definiti gli obiettivi dell'iterazione e le attività necessarie per raggiungerli.

Si è fatto intensamente uso della pratica del **pair programming**, estremamente idonea al team dato il limitato numero di persone componenti. Questa pratica ci ha consentito di migliorare la qualità del codice e di ridurre il numero di errori, in quanto ogni riga di codice è stata esaminata da almeno due persone. Ciò è stato possibile grazie a strumenti quali **git**, fondamentale per la gestione e l'integrazione delle modifiche individuali e la sua integrazione con Visual Studio Code che ci ha permesso di scrivere codice contemporaneamente e di poter revisionare istantaneamente l'un l'altro. Per il testing è stato utilizzato **Selenium**, ecosistema per automatizzare il browser, mentre **Nginx** è stato utilizzato come reverse proxy per rendere più fluida la navigazione web, facendo in modo da rendere tutta l'applicazione web disponibile sulla porta 80.

Tramite cinque iterazioni siamo giunti al prodotto finale completo di documentazione. Di seguito è visualizzato il prodotto della pianificazione delle iterazioni (fig 1.4).

1.2 Planning

Nel contesto del framework **SCRUM**, la pianificazione gioca un ruolo cruciale nell'organizzazione e nell'esecuzione dei progetti. Questo processo iterativo consente ai team di sviluppo di definire obiettivi chiave e determinare il percorso per raggiungere i risultati desiderati, garantendo che le storie utente siano chiare, il lavoro sia distribuito in modo efficace e gli obiettivi siano realistici e raggiungibili.

Durante questa fase, viene stabilito un **piano temporale** che determina quale lavoro deve essere completato in ciascuna iterazione (nota come **sprint**) e quali obiettivi vanno raggiunti entro un periodo prestabilito.

Abbiamo notato che la **pianificazione settimanale** poteva spesso essere problematica a causa degli impegni universitari e personali dei membri del gruppo. Alcuni di noi avevano orari universitari intensi o altri impegni al di fuori del contesto accademico, che rendevano difficile rispettare uno sprint basato rigidamente su settimane di 7 giorni consecutivi. In queste situazioni, l'**agilità** e l'**adattabilità** emergono come fondamentali.

Per massimizzare la **flessibilità** e consentire a ciascun membro del team di partecipare attivamente senza compromettere gli obblighi accademici o personali, abbiamo optato per una pianificazione di sprint basati su 7 incontri anziché su 7 giorni adiacenti. Questo ci ha consentito di adattare il nostro programma di sviluppo alle nostre esigenze individuali, riflettendo il nostro impegno per un ambiente di lavoro collaborativo e inclusivo, in cui ciascun membro può contribuire al massimo delle proprie capacità. Siamo convinti che è anche grazie a questa flessibilità che ci è stato possibile raggiungere i nostri obiettivi di progetto in modo efficace e armonioso.

Nel corso del progetto, abbiamo notato che la pianificazione basata esclusivamente sulle storie utente aveva le sue limitazioni. Mentre le storie utente erano un punto di partenza essenziale per definire gli obiettivi del progetto, fornendo una visione ad alto livello delle funzionalità richieste, man mano che scendevamo nel dettaglio, sprint dopo sprint, sono emersi nuovi aspetti tecnici e requisiti specifici che non erano chiari all'inizio e che richiedevano maggiore adattabilità.

Questo ci ha portato a una profonda comprensione di quanto fosse necessario adattare la nostra pianificazione, riconsiderando il nostro approccio che necessitava una maggiore **adattabilità**. Questa esperienza ci ha infatti insegnato l'importanza di essere reattivi alle problematiche che emergevano e di adattare la strategia in base all'evoluzione delle conoscenze.

Ogni sprint ci ha permesso di acquisire una conoscenza più dettagliata del progetto, di identificare gli errori e di colmare le lacune nelle fasi successive. Alcune funzionalità richieste si sono rivelate più complesse da implementare di quanto inizialmente previsto,

mentre altre richiedevano risorse aggiuntive o avevano requisiti tecnici specifici che non erano stati identificati all'inizio.

Nel **planning** di ogni sprint, stabilivamo le priorità e i compiti chiave, mentre alla fine di ogni sprint, una **sessione di revisione** ci permetteva di esaminare il lavoro svolto evidenziando in **verde** gli obiettivi raggiunti completamente ed in **giallo** quelli raggiunti in maniera solo parziale e di creare una base che sarebbe stata utilizzata nella successiva pianificazione per affrontare le sfide nel prossimo sprint.

Abbiamo introdotto una pratica aggiuntiva nel processo di planning: la colonna "**Problemi Emergenti**", che ci ha permesso di affrontare in modo più agile le sfide impreviste. Questo elemento distintivo è diventato una parte fondamentale della nostra metodologia di lavoro poiché il nostro impegno non era concentrato solo sullo sviluppo, ma anche sulla correzione di problemi esistenti.

La pianificazione ci ha aiutato a rimanere allineati sugli obiettivi, a garantire il controllo su tutte le attività in corso e a mantenere un **approccio agile** nell'affrontare le sfide del progetto, il che dimostra che l'essenza dell'Agilità non riguarda solo la capacità di seguire un piano, ma anche la capacità di adattarsi in modo efficace alle mutevoli esigenze non solo del progetto, ma anche del team di sviluppo.

Planning	Obiettivi raggiunti	Problemi emergenti
1° Sprint		
Acquisizione materiale	Acquisizione materiale	T8 non eseguito correttamente in locale
Lettura documentazioni e comprensione generale del sistema	Lettura documentazioni e comprensione generale del sistema	Problemi con Docker e Maven
Installazione software necessari	Installazione software	
Esecuzione T10 allo stato attuale	Esecuzione T10 allo stato attuale	
Avviare T8 in locale		
+	+	+
2° Sprint		
Approfondimento documentazioni nelle parti critiche	Approfondimento documentazioni nelle parti critiche	Problemi Java JDK
Confronto con gruppi dei task precedenti per avere migliore chiarezza	Confronto con gruppi dei task precedenti per avere migliore chiarezza	Problemi Maven
Risolvere problemi Docker e conoscere meglio il software	Risoluzione problemi Docker e migliore conoscenza del software	T8 ancora non funzionante
Risolvere problemi Maven e conoscere meglio lo strumento	Risoluzione di alcuni problemi di Maven e comprensione di alcune funzionalità dello strumento	
Risolvere problemi T8	Risoluzione parziale dei problemi in T8	
+	+	+

Planning |

Obiettivi raggiunti |

Problemi emergenti |

3° Sprint |

Risolvere problemi Maven di dipendenze

Risoluzione problemi Java JDK

Concludere task T8

Integrare T8 con gli altri task

+

Risolti problemi di dipendenze Maven

Risolti problemi Java JDK

Task T8 completo e funzionante in locale

T8 Integrato con tutti i task

+

Calcolo della coverage ancora a carico di JaCoCo, si vuole Evosuite

+

4° Sprint |

Rimpiazzare funzionalità JaCoCo con Evosuite

Automatizzare installazione iniziale su Docker

Effettuare testing del sistema

Iniziare a formalizzare la documentazione

+

JaCoCo rimpiazzato con Evosuite

Installazione iniziale su Docker automatizzata

Formalizzazione della documentazione iniziata

Problemi nel testing

+



Figure 1.3: Iteration planning

1.2.1 Diario

Nel contesto del **framework Scrum**, la **pianificazione** gioca un ruolo cruciale nell'organizzazione e nell'esecuzione dei progetti. Questo processo iterativo consente ai team di sviluppo di definire obiettivi chiave e determinare il percorso per raggiungere i risultati desiderati, garantendo che le storie utente siano chiare, il lavoro sia distribuito in modo efficace e gli obiettivi siano realistici e raggiungibili.

Per gestire in modo accurato i progressi e le attività del nostro progetto di sviluppo software, abbiamo adottato un sistema di registrazione dettagliato. Abbiamo mantenuto un **diario** delle sessioni di lavoro dedicate al progetto, documentando con precisione **orari**, **attività svolte** e **risultati ottenuti** in ciascuna sessione. Questo diario ha rappresentato un punto di riferimento cruciale per il nostro team, consentendoci di riprendere il lavoro in modo **efficiente**, senza dover affrontare il dispendio di tempo e sforzi necessari per ricordare ogni dettaglio di sessioni precedenti. Inoltre, abbiamo utilizzato questa pratica per monitorare i nostri **progressi** complessivi e identificare eventuali sfide ricorrenti, contribuendo così a migliorare il nostro processo di sviluppo.

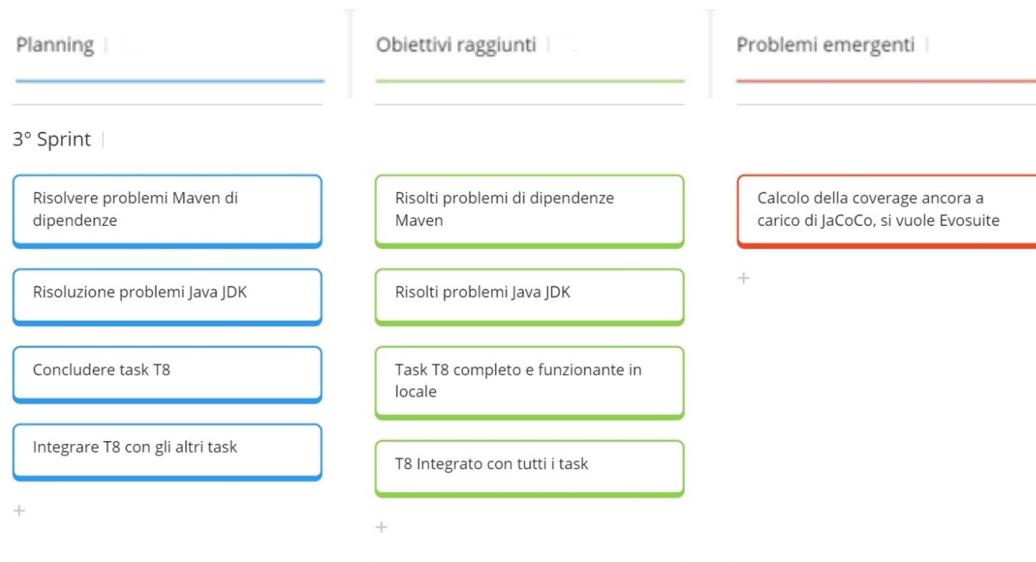
È stato riconosciuto che l'utilizzo di un diario abbia migliorato l'**efficienza** e la **chiarezza** del proprio lavoro. Come si è capito, ciò che conta nei processi di sviluppo Agili, è adottare le pratiche che meglio soddisfano le esigenze specifiche del team e del progetto in corso.



Giorno	Ora	Attività	Problematiche
P 14/09	15:30 - 16:00	Ricevimento, comunicazione task al gruppo	
15/09	10:30 - 12:30 17:30 - 19:10	Download repository, download Docker. Download WSL, Installazione Docker, installazione T10 installer.bat	Problemi installazione Docker. Problemi WSL
18/09	9:50 - 13:00 15:00 - 18:00	Installazione WSL, modifica variabili di sistema, impostazione ambiente Ubuntu. Installazione GitHub Desktop.	
19/09	10:30 - 13:15	Installazione Node.js, installazione T8	T8 non correttamente funzionante
20/09	13:00 - 13:30 16:45 - 18:15	Prova T10 aggiunta classi. Confronto con gruppi precedenti, capito meccanismo aggiunta e rimozione classi	Problema cartella FolderTree in VolumeT9
21/09	11:10 - 13:30 17:00 - 18:00	Lettura documentazione di tutti i task. Studio diagrammi, contatto con G40	
R 22/09	9:30 - 12:50 15:30 - 18:20	Inizio ispezione codice task T8. Comprensione di possibili errori di compatibilità e del flusso di esecuzione	

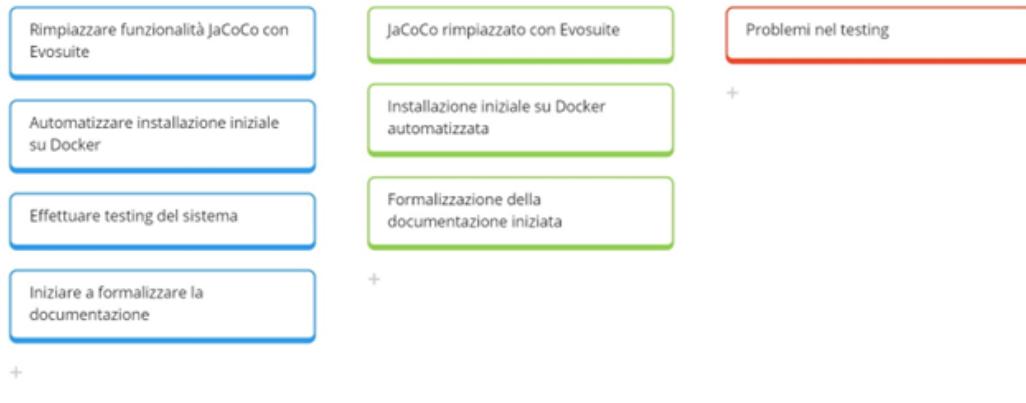


Giorno	Ora	Attività	Problematiche
P 25/09	12:30 - 14:30 15:30 - 17:00	Approfondimento funzionamento Docker. Studio documentazione e flusso di esecuzione conseguente all'aggiunta di una class in RobotUtil	
28/09	12:00 - 13:30 16:30 - 18:00	Correzioni T8: non accesso porta 80, dockerfile, dockercompose, entrypoint. Modifica installer.bat con aggiunta T8. Rimozione package per utilizzo di entrambi i robot.	Errore nella creazione del VolumeT8
30/09	9:10 - 12:15 15:40 - 18:50	Modifica task finale file misurazione_livelli.sh, robot_generazione.sh, RobotUtil.java	
02/10	9:45 - 14:30 15:30 - 18:10	Aggiunta funzione robotgenerate e creazione metodo estrai coverage da csv. Installazione Maven, installazione Java JDK. Aggiunta dipendenze csv pom	Creazione indesiderata cartella RobotTest. I target non si aggiornano correttamente. Errore nei path di VolumeT8, non trovati i file .sh
03/10	12:30 - 13:00 16:00 - 18:00	Aggiunta variabili di ambiente Maven e Java JDK, uso mvn clean install. Modifica path .sh in RobotUtil.	Incompatibilità tra alpine e ubuntu per i Robot, modifica flag inefficaci.
04/10	16:00 - 19:00	Modifica dockerfile per java (alpine, ubuntu) Modifica path java in robot.sh T9 Modifica dockerfile bash T1 path java	Problemi Maven per installazione.sh T8
R 05/10	13:30 - 14:30 16:30 - 17:30	Risolto problema Maven con modifica in installazione.sh con permessi e installazione esplicita di Maven.	Problema Randoop, report.xml non generato.



Giorno	Ora	Attività	Problematiche
P 06/10	13:00 - 13:30 15:00 - 17:30	Installazione dos2unix per conversione Unix file .sh di T8. Modifica misurazione_livelli.sh aggiungendo il path corretto di generazione statistics.csv e aggiornamento RobotUtil coerentemente la modifica	
09/10	15:30 - 18:40	Modifica T5 GuiController per aggiunta livelli Evosuite a quelli di Randoop in maniera grafica	Livelli Evosuite non visibili
11/10	10:30 - 12:30 16:30 - 18:00	Modifica T5: GuiController, main.html, editor.html, report.html	Visualizzazione non corretta report.html (undefined, undefined)
12/10	10:30 - 13:30 16:15 - 18:00	Correzione report.html. Modifica invio informazioni robot e livello in T7. Modifica app.java	
13/10	12:30 - 13:00 15:30 - 17:00	Ricevimento: segnalazione integrazione finale e dettagli documentazione. Creazione entrypoint.sh in T7 e modifica Dockerfile per installazione Evosuite per rimpiazzare JaCoCo	Problemi gitattributes, flusso di esecuzione T5-T6-T7 poco chiaro
14/10	10:20 - 12:00 14:30 - 18:30	Modifica gitattributes, studio flusso di esecuzione T5-T6-T7. Modifica dockerfile T7 variabili ambiente Java. Installazione Java 8 mantenendo Java17 e definizione variabili di ambiente	File tools.jar non trovato. Problemi di incompatibilità con Java17. Errori compilazioni test Evosuite
R 16/10	9:30 - 13:00 15:10 - 18:00	Modificato App.java T7 modifica passaggio parametri classe Evosuite	Problemi package, non trovate classi nel package e metodi associati. Coverage 0%

4° Sprint |



Giorno	Ora	Attività	Problematiche
P 17/10	12:30 - 13:45 15:00 - 17:00	Risoluzioni problemi visibilità classi e metodi. Pom aggiornato con dipendenze Junit4 anzichè Junit5 non compatibile con Evosuite 1.0.6	entrypoint.sh non eseguito correttamente
18/10	10:10 - 12:00 15:30 - 18:30	Risoluzione problemi comandi maven a entrypoint.sh. Inizio formalizzazione documentazione. Modifica per passaggio parametri corretti a eseguibile. Inizio modifica passaggio coverage	Coverage non passato correttamente
19/10	8:30 - 13:30	Modifiche MyController T6 per corretto passaggio coverage da xml a plain_text. Aggiunta alert. Download software per documentazione Miro e Visual Paradigm.	
20/10	15:30 - 17:30	Scelta stile documentazione. Visione di modifiche da effettuare nei diagrammi presenti in T10.	
21/10	9:00 - 11:30	Automatizzazione installazione entrypoint.sh, installazione.sh all'atto della chiamata installer.bat	
23/10	8:30 - 12:30 15:30 - 17:30	Documentazione: diagrammi e formalizzazione storie utente. Inizio fase di testing	
R 24/10	12:30 - 13:30 14:30 - 17:00	Documentazione: criteri di accettazione, planning. Testing	Problemi nel testing



Giorno	Ora	Attività	Problematiche
P 25/10	15:00 - 18:00	Modifica EditorTest.java e LoginTest.java e pom relativo, rimuovendo dipendenza non utilizzata, e sono stati effettuati i test.	
26/10	8:30 - 13:30 16:10 - 17:45	Documentazione: formalizzazione diario, descrizione formale modifiche T8. Aggiustamenti software: aggiunta bottone LogOut in tutte le schermate utente	
27/10	8:45 - 10:30 15:00 - 17:30	Documentazione: diagrammi Ricevimento: aggiornamento progetto. Ripristino funzionalità highlight JaCoCo.	Errore goal JaCoCo
28/10	10:00 - 12:00 15:00 - 18:50	Risoluzione problemi goal minimum coverage JaCoCo Aggiunta nuove misure di coverage in codice.sh T7 Modifica funzione estrazione da file csv T7 Modifica passaggio parametri T7	Errore nel passaggio dei parametri
30/10	9:30 - 10:30 15:00 - 18:20	Modifica passaggio parametri T6, T5 Risoluzione passaggio delle statistiche Aggiunta funzione highlight a run Modifica diagrammi: sequence, composite, use case	
01/11	10:30 - 13:10 17:00 - 19:10	Risoluzione problemi pom T9 Aggiunta casi test mail errata Ricevimento d'urgenza: comunicazione invio Repository Inizio Formalizzazione readme	Terminare entro domattina la formalizzazione del readme, la creazione dei video e ripulire ciò che rimane da rivedere del codice.
02/11	8:30 - 13:30	Completamento formalizzazione readme Creazione video Pulizia codice Invio Repository completa	
03/11	15:30 - 18:30	Formattazione Latex della documentazione Ricevimento: concordanza esame Screenshot del codice di particolare rilevanza. Screenshot diagrammi	
R 04/11	10:30 - 13:30 15:50 - 17:30	Terminazione della documentazione Revisione generale	

1.3 Workflow diagram

Lo scenario di gioco che è qui sviluppato è quello descritto in figura 1.4, con singolo giocatore, singolo turno, singolo round. Per descriverlo è stato usato un workflow diagram, sequenze logiche di attività che, insieme, modellano processi aziendali. Da questo schema è facile individuare i microservizi sviluppabili.

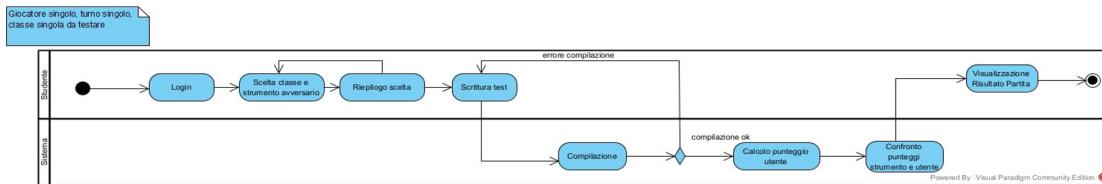


Figure 1.4: Primo scenario di gioco

-2-

Analisi dei requisiti

Per i singoli requisiti si rimanda alla documentazione degli altri task dove vi è un'analisi molto approfondita.

2.1 User Stories

Le **storie utente**, o user stories, rappresentano in modo semplice un **requisito** di valore per un utente del sistema, catturando i **ruoli**, le **attività** e gli **obiettivi** dell'utente, fornendo una guida per il processo di sviluppo agile. Sono **piccoli incrementi di valore** che descrivono piccoli pezzi di funzionalità desiderata dal punto di vista dell'**attore** coinvolto, espresse nella breve forma "**As a** (ruolo dell'utente), **I want** (azione, funzionalità), **so that** (valore, beneficio, desiderio)".

Le user stories hanno lo scopo di identificare e documentare dettagliatamente le esigenze e le aspettative degli utenti finali, offrendo così una guida chiara per la progettazione e lo sviluppo delle funzionalità, consentendo inoltre di classificarle in base all'importanza per l'utente, sostenendo così il processo di gestione del progetto, soprattutto all'interno del framework SCRUM.

Per garantire la loro efficacia seguono il principio **INVEST** (Indipendenti, Negoziali, di Valore, Stimabili, di dimensioni ridotte, Testabili). I **criteri di accettazione** ne confermano la correttezza: seguendo il concetto delle "3C" (Card, Conversation, Confirmation) consentono di scrivere le storie in modo chiaro, discutere i dettagli necessari e definire i criteri di accettazione per il loro completamento; queste tre componenti sono fondamentali per assicurare che le storie utente siano ben comprese, facilmente gestibili e testabili all'interno del processo di sviluppo per garantire un adeguato sviluppo e test delle funzionalità.

2.1.1 Storie Utente non completate

Nel nostro percorso di sviluppo, abbiamo ereditato alcune storie utente da documentazioni precedenti che purtroppo non sono state portate a compimento. Il non completamento di queste storie è stato causato dal fatto che chi ha lavorato al progetto prima di noi non ha implementato alcune funzionalità critiche, impedendo così che le storie fossero portate a termine come inizialmente previsto.

Queste storie erano state originariamente concepite per soddisfare determinati requisiti, ma la mancanza di alcune funzionalità chiave le ha rese impossibili da completare nell'ambito delle condizioni esistenti. In questa fase del progetto, è essenziale riconsiderare queste storie e affrontare i problemi che hanno impedito il loro completamento, al fine di soddisfare i requisiti utente e migliorare il prodotto complessivo. È quindi essenziale comprendere i motivi per cui queste storie non sono state concluse con successo in passato, al fine di garantire una corretta implementazione.

Inizialmente, identificare le storie utente è stata una sfida, specialmente quando non si conoscono profondamente le parti del programma che presentano malfunzionamenti. Inoltre, prevedere l'onerosità di queste storie è stato complicato in assenza di dettagli esaustivi sull'implementazione e delle aree in cui si verificano gli errori. È stato particolarmente difficile per noi, agli inizi del percorso, stimare il tempo necessario per sviluppare una specifica funzionalità del programma. Solitamente, queste valutazioni richiedono esperienza, e coloro che hanno affrontato simili compiti più volte possono fornire stime più accurate.

Il nostro approccio ha richiesto quindi, soprattutto durante il primo sprint, uno studio approfondito del sistema, finalizzato a individuare le funzionalità mancanti che necessitavano di implementazione. Questo processo ci ha permesso di delineare in modo più preciso le storie utente da affrontare.

Così come per il planning, anche qui abbiamo adottato il tool **Miro** per la creazione delle user stories, una decisione motivata dal desiderio di mantenere un approccio uniforme e coeso con il gruppo G40 che ci ha preceduto nello sviluppo del progetto. L'uso di Miro ci permette di mantenere uno stile e una struttura coerente nelle storie utente, contribuendo così a una transizione fluida e a una comprensione comune tra i team di sviluppo, migliorando complessivamente la chiarezza e l'efficienza del processo di sviluppo.

Di seguito vengono presentate alcune delle storie utente precedentemente documentate. Queste sono state ulteriormente affinate e adattate alla struttura '**As a, I want, so that**' al fine di migliorare la chiarezza e la semplicità, rispecchiando al meglio le linee guida delle user story.

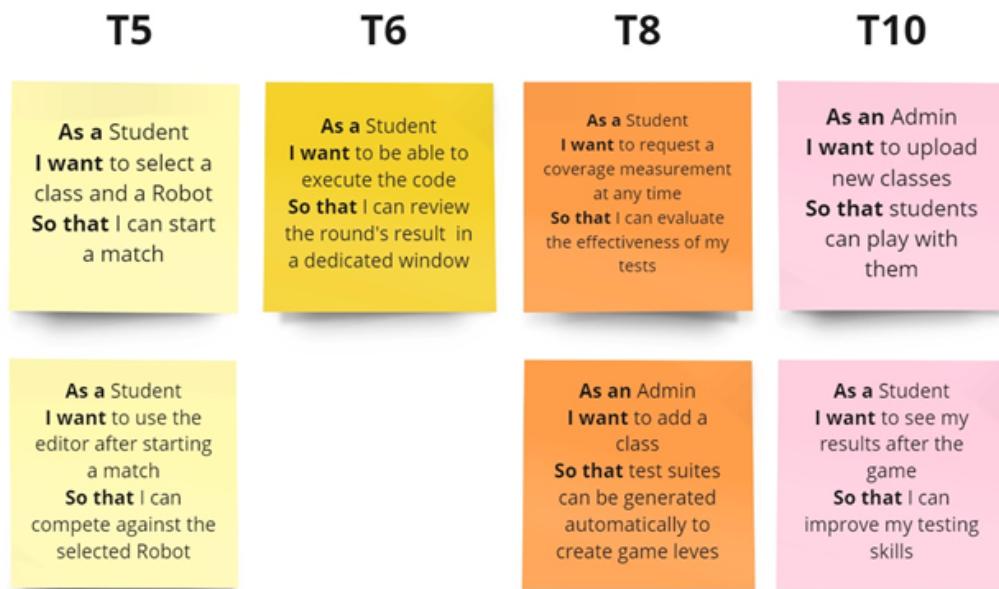


Figure 2.1: User Stories non completate

Le principali motivazioni per le quali queste storie utente non sono state portate a compimento erano dovute a malfunzionamenti nel task T8 e, a cascata, mancanti considerazioni di altri task per l'integrazione di quest'ultimo. Il risultato di questa mancanza era l'impossibilità per l'admin di aggiungere livelli con tipologia di robot Evosuite al momento di aggiunta di una nuova classe e, di conseguenza, l'impossibilità per l'utente di selezionare il robot Evosuite come avversario di un match.

Inoltre, in assenza del funzionamento di Evosuite, è stata delegata temporaneamente la misurazione della copertura del codice allo strumento JaCoCo. Tuttavia, con il successivo ripristino del corretto funzionamento di Evosuite, è stato possibile affidare la misurazione di copertura ad EvoSuite mentre JaCoCo ha mantenuto la sua funzionalità di evidenziare le linee di codice coperte.

2.1.2 Criteri di accettazione

I criteri di accettazione, noti come "**acceptance criteria**", sono dettagli specifici scritti dal punto di vista del testing per le storie utente nella metodologia SCRUM. Questi criteri sono creati dall'**Agile Team** e rappresentano condizioni chiare e specifiche che devono essere soddisfatte affinché una storia utente possa essere considerata completata con successo. I criteri di accettazione sono fondamentali nel processo di sviluppo agile, poiché definiscono in modo preciso ciò che il team deve realizzare per soddisfare i requisiti dell'utente.

In SCRUM, i criteri di accettazione vengono spesso scritti utilizzando il modello "**Given-**

When-Then" o una struttura simile, che stabilisce le condizioni iniziali (**Given**), l'azione o evento (**When**), l'outcome desiderato (**Then**) e ulteriori condizioni o aspetti specifici (**And**). Questi criteri aiutano il team di sviluppo a comprendere chiaramente cosa si aspetta dall'implementazione della storia e forniscono una base solida per il processo di testing. In breve, i criteri di accettazione assicurano che il lavoro sia allineato con le aspettative dell'utente e contribuiscono a garantire la qualità e la completezza delle storie durante lo sviluppo.

L'aggiunta dei criteri di accettazione alle storie utente è stata fatta per garantire una maggiore chiarezza, completezza e una migliore comprensione delle esigenze degli utenti all'interno del team di sviluppo. Nel corso dello sviluppo del nostro progetto, ci siamo resi conto che alcune delle documentazioni precedenti non includevano in modo esplicito i criteri di accettazione per le storie utente.

Si è voluto introdurre questa pratica per avere una maggiore chiarezza nella definizione delle funzionalità, consentendo di stabilire in modo chiaro cosa si aspetta l'utente finale da una determinata funzionalità o requisito. Inoltre, ha consentito una miglior comunicazione all'interno del team fornendo un linguaggio comune tra i membri del team di sviluppo, facilitando la discussione e l'implementazione delle funzionalità.

Al contempo i criteri di accettazione forniscono punti di riferimento concreti per valutare il successo o il fallimento di una funzionalità, aiutando a tracciare lo stato di avanzamento del progetto in modo più accurato.

Ogni criterio stabilisce le condizioni che devono essere soddisfatte per ritenerе una storia utente completata con successo, aiutando a evitare fraintendimenti e ambiguità nella comprensione delle esigenze degli utenti.

GIVEN: The web application is accessible, and the student has logged in
WHEN: The student accesses the page of available classes
THEN: The student views the list of available classes with their respective robot levels
AND: The student can choose the class-robot-difficulty combination for the match

As a Student
I want to select a class and a Robot
So that I can start a match

GIVEN: The student has viewed the list of available classes
WHEN: The student selects a class for the match
THEN: The selected class is visually highlighted or confirmed
AND: The student can now select the level and robot to challenge

GIVEN: The student has selected a class for the match
WHEN: The student selects a robot and a level related to the chosen class
THEN: The selected class is visually highlighted or confirmed
AND: The student is ready to start the match

As a Student
I want to use the editor after starting a match
So that I can compete against the selected Robot

GIVEN: The student has access to the page of the selected class
WHEN: The student starts the match
THEN: The student views the editor
AND: The student views the class for the match
AND: The student can write and edit code in the editor
AND: The student can use all the editor's features, including code execution and testing

As a Student
I want to be able to execute the code
So that I can review the round's result in a dedicated window

GIVEN: The student has started a match and written error-free code
WHEN: The student clicks the execution button
THEN: The student views the round results in a dedicated window
AND: The student views a log of the correct compilation and code execution

GIVEN: The student has executed code with errors
WHEN: The round is completed
THEN: The dedicated window highlights errors and provides suggestions for correction.
AND: The student can view a detailed log of errors and exceptions generated during execution

As a Student
I want to request a coverage measurement at any time
So that I can evaluate the effectiveness of my tests

GIVEN: The student has executed code with errors
WHEN: The round is completed
THEN: The dedicated window highlights errors and provides suggestions for correction.
AND: The student can view a detailed log of errors and exceptions generated during execution

GIVEN: The admin has logged in
WHEN: The admin selects the option to add a new class
THEN: The system allows the admin to enter details of the new class, such as name, description, and additional information
AND: Automatically, the system generates test suites based on the new class added and uses them to create new game levels

As an Admin
I want to add a class
So that test suites can be generated automatically to create game levels

GIVEN: The admin has logged in
WHEN: The admin selects the option to add a new class
THEN: The admin can enter the details of the new class, such as name, description, and class file upload.
AND: The system verifies that the class has been successfully added by displaying it among the available classes.

GIVEN: The admin has provided all the necessary information for adding a new class
WHEN: The admin confirms the addition of a class
THEN: Test suites based on the new class added are automatically generated
AND: The test suites are used to create new game levels ready for student use

GIVEN: The admin has logged into the system
WHEN: The admin selects the option to add a new test class
THEN: The system allows the admin to enter class details, including a description, and upload the class file.
AND: The admin has the option to complete the entry and make the class available to students

As an Admin
I want to upload new classes
So that students can play with them

GIVEN: The admin is on the class addition page
WHEN: The admin uploads the class file
THEN: The system verifies the format of the class file and confirms its validity
AND: The admin sees the class correctly added among the available classes

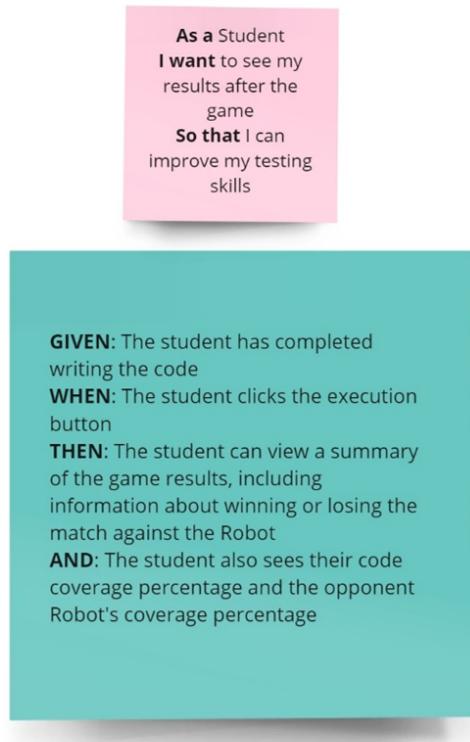


Figure 2.2: Criteri di accettazione

2.2 Modello dei casi d'uso

Si è deciso di realizzare un diagramma dei casi d'uso generale di tutte le funzionalità disponibili in questa integrazione. Sono evidenziati in **rosso** i casi d'uso qui prodotti e/o resi funzionanti.

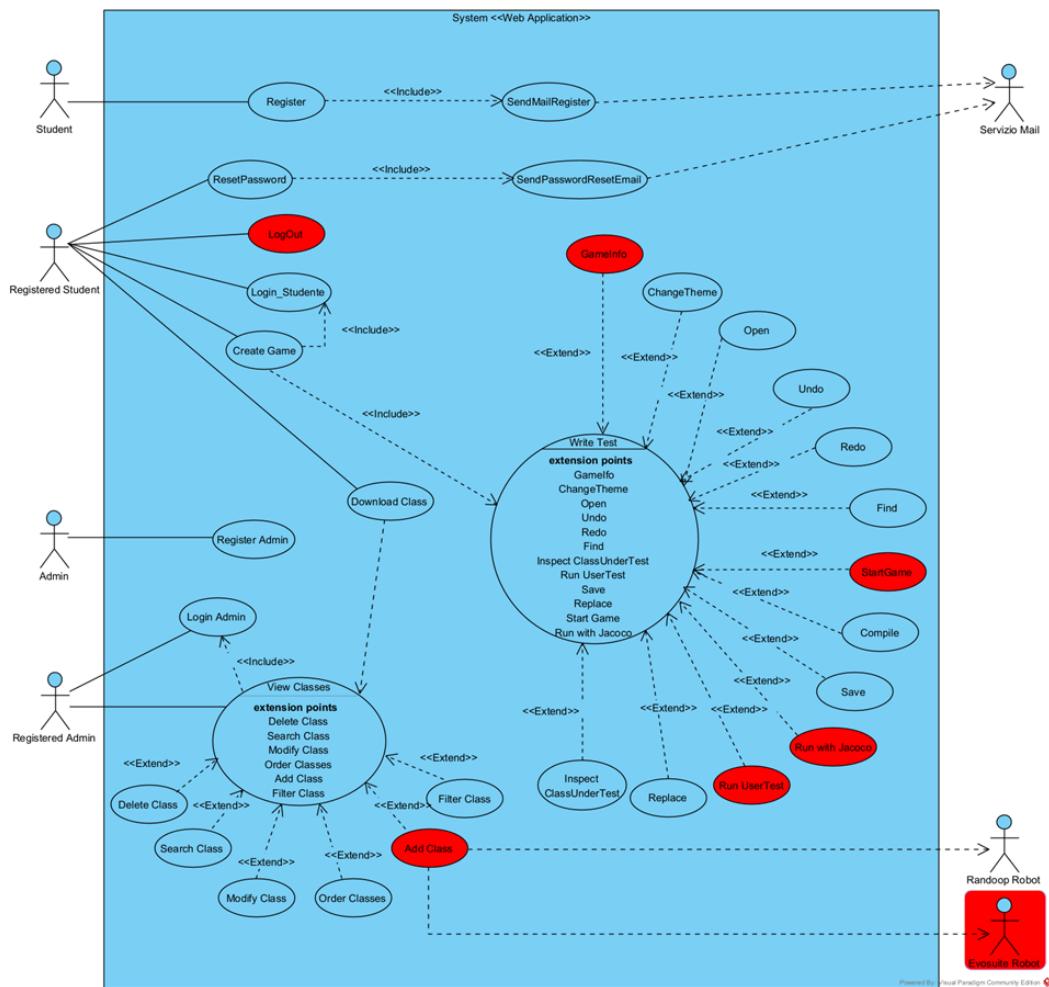


Figure 2.3: Diagramma dei casi d'uso

Si è cercato per quanto possibile di mantenere coerenti i nomi dei casi d'uso con quelli dei singoli task, anche se è stato necessario per i casi d'uso relativi all'amministratore, tradurre i nomi in inglese per mantenere coerenza linguistica. **"View classes"** e **"Create Game"** includono il login che è necessario fare preliminarmente. Sono poi disponibili una gamma di casi d'uso d'estensione, opzionali per i due tipi di utente.

"Start Game" consiste nella sottomissione del tentativo da parte dello studente mentre **"Run with Jacoco"** consiste nell'evidenziare le linee di codice coperte dal test e **"Run UserTest"** consiste nel mostrare a video la copertura del test utente valutata con EvoSuite.

2.2.1 Attori

Il **Registered Student** è colui che può giocare la partita contro il robot scrivendo il suo test.

Il **Registered Admin** è colui che gestisce le classi con cui è possibile giocare all'interno del sistema. Se questi due attori non sono registrati sono rinominati Student e Admin.

Randoop Robot è un attore secondario che interagisce con il caso d'uso di aggiunta di una nuova classe da parte dell'Admin: contestualmente il robot produrrà i suoi test.

EvoSuite Robot è un attore secondario che interagisce con il caso d'uso di aggiunta di una nuova classe da parte dell'Admin: contestualmente il robot produrrà i suoi test.

Servizio Mail è ancora un attore secondario che interagisce nella fase di registrazione e recupero password con lo studente.

2.3 Scenari

Verranno qui descritti gli scenari dei casi d'uso evidenziati in **rosso**, in cui sono presenti modifiche. Per i restanti, si rimanda alle documentazioni del Task 1 per i casi d'uso dell'Admin; al Task 2-3 per i casi d'uso di autenticazione; al Task 6 per le funzioni di estensione semplici del caso d'uso "**Write Code**". Le modifiche più importanti sono in "Add class" dove si è deciso di eseguire, localmente e contestualmente all'inserimento della classe da parte dell'admin nel repository, il robot EvoSuite di generazione automatica dei test in aggiunta al robot Randoop già presente.

2.3.1 Logout

Lo studente può effettuare il logout dopo aver effettuato il login. Lo studente può cliccare sull'apposito pulsante di logout per essere de-autenticato e reindirizzato alla pagina di login.

Caso d'uso	Logout
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Permette ad uno studente di effettuare il logout
Pre-Condizioni	Lo studente deve essere loggato
Sequenza di eventi principale	1)Lo studente clicca su "Logout"
Post-Condizioni	Lo studente visualizza la schermata di login
Casi d'uso correlati	-
Sequenza di eventi alternativi	-

Figure 2.4: Logout

2.3.2 Create game

Lo studente può iniziare una partita dopo aver effettuato il login. Si vuole sia possibile per l'utente scegliere quale robot sfidare e con quale livello, su quale classe scrivere i test e poi salvare queste informazioni opportunamente. Dopo la creazione della partita, si avvia il caso d'uso **"Write Code"**.

Caso d'uso	Create Game
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente sceglie la classe da testare e il robot con il <u>livello da affrontare</u>
Pre-Condizioni	Lo studente ha effettuato il login
Sequenza di eventi principale	1)Il sistema mostra una lista di classi 2)Lo studente sceglie la classe 3)Il sistema mostra i robot ed i livelli disponibili 4)Lo studente sceglie il robot con il relativo livello e clicca "Submit"
Post-Condizioni	Lo studente visualizza la schermata di report
Casi d'uso correlati	-
Sequenza di eventi alternativi	-

Figure 2.5: Create game

2.3.3 Write code

Dopo la creazione della partita da parte dello studente, sarà possibile scrivere effettivamente il test e attivare opzionalmente alcune funzionalità sull'editor.

Caso d'uso	Write Code
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente scrive all'interno dell'editor di testo il codice java del test in formato Junit4
Pre-Condizioni	Lo studente ha creato la partita
Sequenza di eventi principale	Lo studente scrive il test in formato Junit4 all'interno della sezione opportuna
Post-Condizioni	-
Casi d'uso correlati	-
Sequenza di eventi alternativi	Lo studente possiede già una classe di test e vuole caricarla

Figure 2.6: Write code

2.3.4 Inspect ClassUnderTest

All'avvio dell'editor ("Write Code") si vuole si carichi in un apposito riquadro la classe che lo studente ha scelto di testare.

Caso d'uso	Inspect Class Under Test
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente possiede già una classe di test in formato Junit4 e vuole caricarla nella sezione
Pre-Condizioni	Scelta della ClassUnderTest dal repository
Sequenza di eventi principale	1)Lo studente clicca sull'icona per effettuare il caricamento 2)Lo studente sceglie il file del test in formato Junit4
Post-Condizioni	La ClassUnderTest viene visualizzata nella finestra di gioco
Casi d'uso correlati	-
Sequenza di eventi alternativi	Errore nell'upload del file

Figure 2.7: Inspect ClassUnderTest

2.3.5 Run UserTest

Si vuole che, alla pressione del tasto "**Run userTest**", l'utente possa visualizzare il punteggio del suo test.

Caso d'uso	Run UserTest
Attore primario	Studente registrato
Attore secondario	Robot Evosuite
Descrizione	Lo studente dopo aver scritto un test vuole valutare la copertura raggiunta senza far partire la partita.
Pre-Condizioni	Test Case scritto
Sequenza di eventi principale	1)Lo studente clicca sull'icona di EvoSuite per iniziare la valutazione
Post-Condizioni	Un alert mostra a video la copertura del test
Casi d'uso correlati	-
Sequenza di eventi alternativi	Il codice fornito presenta errori di compilazione. Viene mostrato un alert di errore.

Figure 2.8: Run UserTest

2.3.6 Run with JaCoCo

Si vuole che, alla pressione del tasto “**Run with JaCoCo**”, l’utente possa visualizzare le linee di codice coperte dal suo test utilizzando JaCoCo.

Caso d’uso	Run with JaCoCo
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente dopo aver scritto un test vuole valutare la copertura raggiunta graficamente.
Pre-Condizioni	Test Case scritto
Sequenza di eventi principale	1) Lo studente clicca sull’icona di JaCoCo per iniziare la valutazione
Post-Condizioni	Colorazione delle linee di codice coperte dalla classe di test tramite JaCoCo
Casi d’uso correlati	-
Sequenza di eventi alternativi	1) Il codice fornito presenta errori di compilazione. Viene mostrato un alert di errore. 2) Il test case scritto non copre nessuna linea di codice

Figure 2.9: Run with JaCoCo

2.3.7 Compile

Si vuole che, alla pressione del tasto “**Compile**”, sia mostrato l’output della compilazione a video, in una apposita sezione.

Caso d’uso	Compile
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente dopo aver scritto un test vuole compilare per valutare eventuali errori.
Pre-Condizioni	Test Case scritto
Sequenza di eventi principale	1) Lo studente clicca sull’icona di Compile per avviare
Post-Condizioni	Visualizzazione dell’esito della compilazione
Casi d’uso correlati	-
Sequenza di eventi alternativi	Il codice fornito presenta errori di compilazione

Figure 2.10: Compile

2.3.8 Start Game

Si vuole che, alla pressione del tasto di "Start Game", l'utente sottometta il suo tentativo contro il robot e vengano visualizzati i risultati della partita.

Caso d'uso	Start Game
Attore primario	Studente registrato
Attore secondario	-
Descrizione	Lo studente dopo aver scritto un test vuole concludere la partita
Pre-Condizioni	Test Case scritto
Sequenza di eventi principale	1)Lo studente clicca sull'icona di Start Game per avviare la partita
Post-Condizioni	1)Visualizzazione dell'output 2)Alert che mostra il vincitore 3)Colorazione delle linee di codice coperte dalla classe di test tramite JaCoCo 4)Visualizzazione punteggio del robot, dell'utente e di tutte le altre metriche
Casi d'uso correlati	-
Sequenza di eventi alternativi	1)Il codice fornito presenta errori di compilazione. Viene mostrato un alert di errore.

Figure 2.11: Start Game

2.3.9 Add Class

Si vuole che l'admin possa inserire una classe nel repository e che contestualmente a ciò, vengano prodotti i test e calcolate le coperture di Randoop ed Evosuite.

Caso d'uso Add Class	
Attore primario	Admin
Attore secondario	Robot Randoop e Robot Evosuite
Descrizione	L'admin aggiunge una classe al repository
Pre-Condizioni	L'admin ha effettuato il login
Sequenza di eventi principale	1)L'admin clicca su "Add class" dalla propria interfaccia 2)Il sistema mostra un form da compilare 3)L'admin inserisce il nome, la data, la difficoltà, optionalmente una descrizione e tre categorie. 4)L'admin fa l'upload della classe 5)Il sistema salva la classe 6)Il Robot Randoop genera i test relativi a quella classe divisi in livelli 7) Il Robot EvoSuite genera i test relativi a quella classe divisi in livelli 8) I risultati ottenuti vengono salvati opportunamente 9)Il sistema mostra le classi inserite
Post-Condizioni	La classe inserita è disponibile allo studente durante la creazione della partita. Il punteggio è disponibile per l'elaborazione del vincitore alla conclusione della partita
Casi d'uso correlati	-
Sequenza di eventi alternativi	1) Errore durante l'upload della classe

Figure 2.12: Add Class

2.4 Dipendenze tra i Task

Per effettuare l'integrazione dei vari Task è stato necessario tracciare e mettere in evidenza quelle che sono le dipendenze tra i vari Task, ponendo quindi l'attenzione sulle operazioni (casi d'uso) che richiedono l'utilizzo di servizi offerti da altri. Per fornire, quindi, una panoramica abbastanza completa, si è deciso di rappresentare, in forma tabellare, le dipendenze tra le varie operazioni mettendo in evidenza, per ogni operazione (Operations):

- i task da cui dipende (Collaborators)
- le operazioni dei task da cui dipende che utilizza
- le REST API endpoint che permettono di effettuare

Service	Operation	Collaborators
Task T1	uploadClass ()	Task 4: creazione RisultatiRobot () /robots POST
Task 5	Visualizza Classi () VisualizzaRobot () SalvaPartita ()	Task 1: elencaClassi (): /home GET Task 4: ricercaRisultatiRobot (): /robots GET Task 4: creazionePartita (): /games POST creazioneRound () /rounds POST creazioneTurno () /turns POST
Task 6	ReceiveClassUnderTest () compile () getReport () /getJaCoCoReport	Task 1: downloadClasse () /downloadFile GET Task 7: compileExecuteCoverage (): /compile-and-codecoverage POST Task 7: compileExecuteCoverage(): /compile-and-codecoverage POST
	Run ()	Task 7: compileExecuteCoverage (): /compile-and-codecoverage POST Task 4 ParametriPartita () /robots GET Task 4: ModificaPartita (): /games PUT modificaRound (): /rounds PUT ModificaTurno (): /turns PUT

Figure 2.13: Tabella delle dipendenze

Si rimanda alla documentazione dei singoli task per la documentazione delle API presenti nella tabella.

—3—

Progettazione

3.1 L' architettura a microservizi scelta

L'architettura a microservizi della web application realizzata si sviluppa come rappresentato in figura 3.1. Il client può accedere all'applicazione esclusivamente interfacciandosi con i due gateway tramite internet. I microservizi non sono quindi esposti all'esterno della loro rete locale: ciò aggiunge un ulteriore livello di sicurezza.

3.1.1 Gateway Pattern

Alla base c'è il tentativo di realizzare un'applicazione sicura e scalabile: ciò è reso possibile grazie all'utilizzo del **Gateway Pattern**. Esso consiste nell'introdurre dei componenti che costituiscono gli unici punti di accesso ai microservizi: ciò permette di implementare dei meccanismi di autenticazione, autorizzazione, routing, load-balancing e API composition (una sorta di facade) delle richieste molto più versatili ed efficienti.

3.1.2 UI Gateway

Tale componente è stato realizzato utilizzando *Nginx*, realizzando di fatto un **reverse proxy**: la sola responsabilità di questo gateway è quella di effettuare il routing delle richieste relative al frontend (UI). Ciò ha permesso di rendere disponibile l'intera applicazione alla porta 80 (porta predefinita per le connessioni HTTP), ottenendo così una certa uniformità.

3.1.3 API Gateway

Tale componente è stato realizzato utilizzando **Spring Boot** insieme a **Netflix Zuul**, realizzando, anche in questo caso, un reverse proxy ma con più responsabilità: si occupa di effettuare il routing, di gestire autenticazione e autorizzazione delle richieste relative alle REST API. Ciò ha permesso di rendere disponibili tutte le API al di sotto del percorso URL ”/api/”, ottenendo così anche qui una certa uniformità.

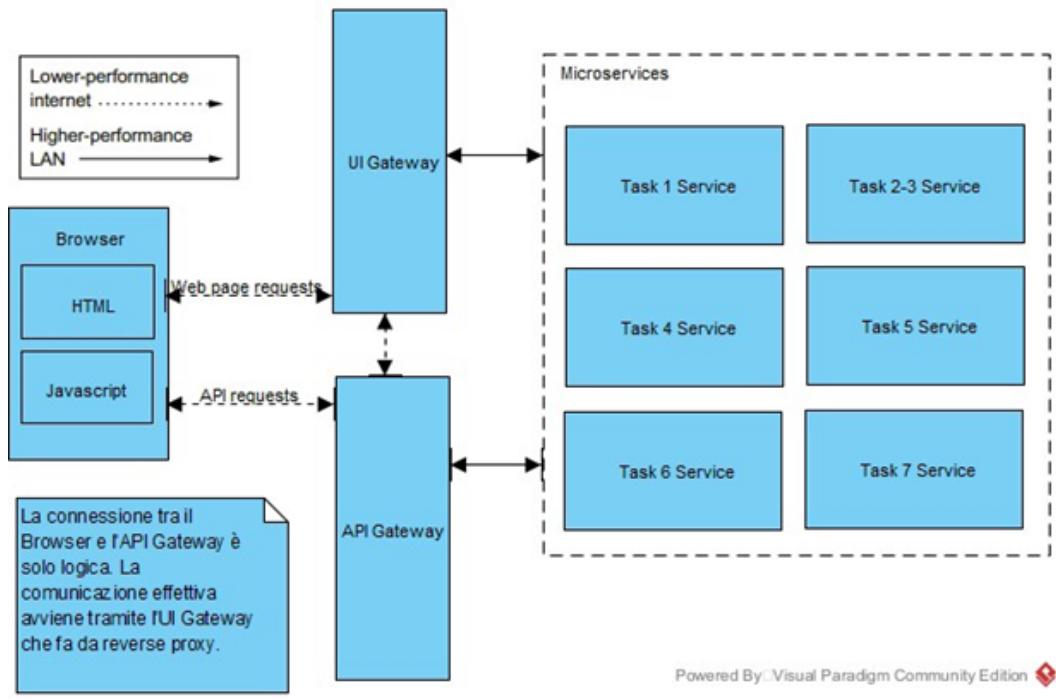


Figure 3.1: Architettura generale sistema

3.2 Component Diagram

Il **Component diagram** ci aiuta a capire a tempo di esecuzione quali entità ci sono nel sistema e come interagiscono tra di loro per adempiere alle funzioni richieste.

Si noti che i componenti che svolgono funzioni relative allo studente sono disaccoppiate da quelli dell'amministratore: il motivo sta nel differente approccio all'autenticazione.

Per quanto riguarda l'accesso dello Studente, grazie al servizio di autenticazione offerto dal componente **Student Repository** tramite **token JWT**, è stato possibile implementare un controllo di sicurezza ulteriore grazie all'uso di dell'API Gateway, che però non protegge le API lato Admin.

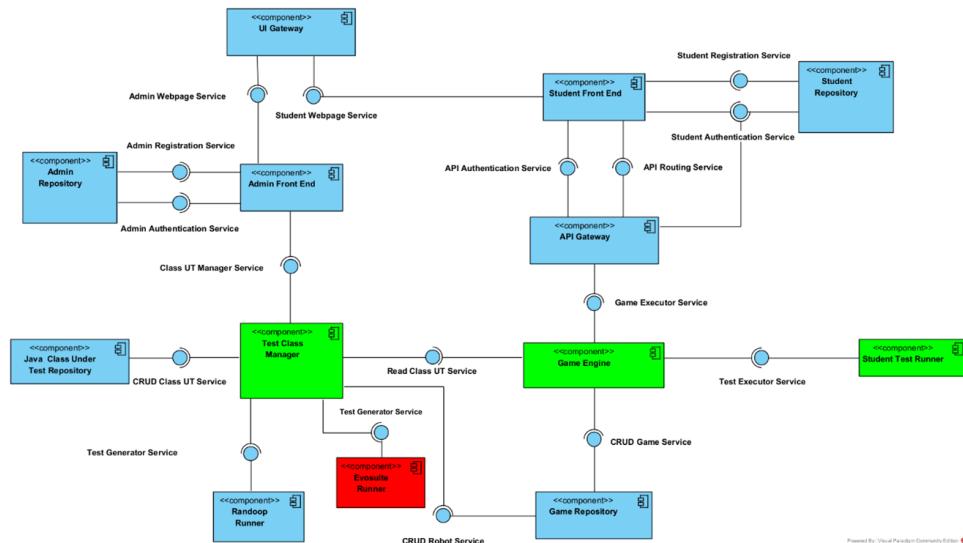


Figure 3.2: Component diagram

3.2.1 UI Gateway

Gestisce le richieste alla User Interface implementando il meccanismo di routing.

Usa i servizi dello Student e dell'Admin Front End per fornire le corrette pagine web al client che le richiede.

Realizzato con **Nginx**.

3.2.2 API Gateway

Gestisce le richieste alle API implementando meccanismi di routing, autenticazione e autorizzazione. Introduce quindi un livello di sicurezza che si basa sul controllo di validità del token JWT memorizzato in uno specifico cookie.

Usa i servizi offerti dal **Game Engine** per adempiere alle richieste di cui se ne è verificata la sicurezza. Realizzato con **Spring Boot** e **Netflix Zuul**.

3.2.3 Student Front End

Fornisce le **User Interfaces** navigabili dall'utente, dal login alle schermate di gioco. Realizzato con **Spring Boot** e **Code Mirror**.

3.2.4 Student Repository

Si occupa della gestione dell'autenticazione dell'utente Studente: mantiene in un database i dati relativi agli studenti registrati e gestisce l'autenticazione tramite un token JWT memorizzato in un cookie.

Realizzato con **Spring Boot** e **MySQL**.

3.2.5 Admin Front End

Fornisce le User Interfaces di gestione delle classi e di accesso per l'utente Admin. Realizzato con **Spring Boot**.

3.2.6 Admin Repository

Si occupa della gestione dell'autenticazione dell'utente Admin: mantiene in un database gli amministratori registrati.

Realizzato con **Spring Boot** e **MongoDB**.

3.2.7 Game Engine

Si occupa della gestione della partita.

Usa i servizi offerti dallo Student Test Runner e dal Game Repository per un corretto svolgimento della partita: ottiene i risultati della compilazione e della coverage del test dello studente, i risultati dei robot e salva le informazioni della partita in corso. Usa, inoltre, i servizi del Test Class Manager al fine di ottenere la classe che l'utente ha scelto di testare.

Realizzato con **Spring Boot**.

3.2.8 Student Test Runner

Si occupa della compilazione e del calcolo della coverage del codice di test sviluppato dall'utente studente che riceve dal Game Engine. Realizzato con **Maven**, **JaCoCo**, **Spring Boot**.

3.2.9 Test Class Manager

Si occupa della gestione delle classi giocabili inserite dall'Admin. Usa i servizi del componente **Java Class Under Test Repository** per mantenere i dati delle classi.

Usa il servizio di compilazione e calcolo della coverage dei due robot, ogni volta che viene inserita una classe nel sistema.

Usa il servizio del Game Repository per salvare e rendere disponibili al Game Engine i risultati prodotti dai robot.

Realizzato con **Spring Boot**.

3.2.10 Java Class Under Test Repository

Si occupa del salvataggio delle classi inserite dall'Admin: salva i file della classi sul filesystem e le informazioni su di essa in un database.

Realizzato con **MongoDB** e **Spring Boot**.

3.2.11 Randoop Runner

Genera test a partire da una classe con il robot Randoop e li salva localmente, per poi calcolarne la copertura con il tool Emma. La generazione dei test avviene secondo più livelli in modo dinamico, se si generano fissate delle impostazioni, più test con diversa copertura, li si raggruppa in livelli.

Realizzato con **Randoop** e **Emma**.

3.2.12 EvoSuite Runner

Genera test a partire da una classe con il robot EvoSuite e li salva localmente insieme al report in formato csv. La generazione dei test avviene secondo più livelli il cui numero per simmetria è pari a quello generato da Randoop.

Realizzato con **EvoSuite**.

3.2.13 Game Repository

Mantiene lo storico delle partite, le classi giocabili, i risultati dei robot per livello e tutte le informazioni necessarie al corretto svolgimento della partita.

Realizzato con **Chi (GoLang)** e **PostgreSQL**.

3.3 Composite Structure Diagram

Nel seguente diagramma, si è riportato la mappatura di chi, tra i vari task, ha sviluppato quale componente visibile a tempo di esecuzione. La notazione scelta, per mantenere coerenza con Component Diagram è di inserire un package per ogni componente che specifichi quali task ne sono responsabili e ne contribuiscono al funzionamento.

Mentre alcuni componenti sono completamente realizzati da singoli task, ce ne sono altri come lo **Student Front End** e il **Game Engine** che sono distribuiti su due task.

Lo Student Front End è realizzato dal task 2-3 per quanto riguarda la parte dell'accesso al gioco, dunque le schermate di login, registrazione, recupero password. Per la parte che riguarda il gioco vero e proprio, il front end è stato realizzato nel task 5.

Il Game Engine, invece, è realizzato nella parte di salvataggio iniziale della partita, dal task 5, e per il resto delle funzionalità dal task 6 (run, compile etc.).

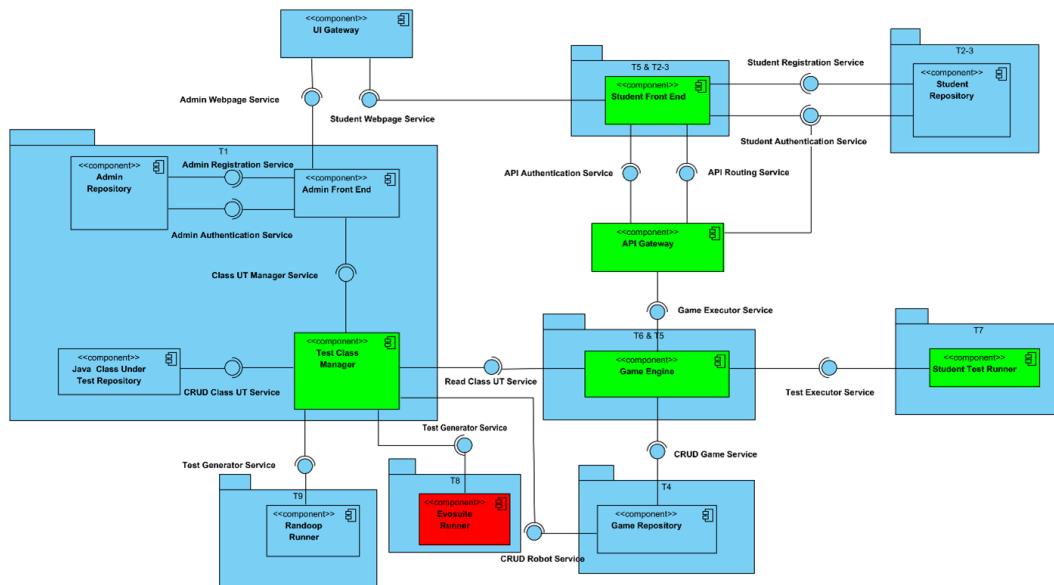


Figure 3.3: Composite Structure Diagram

3.3.1 Modifiche Apportate

Nei seguenti paragrafi saranno riportate tutte le modifiche effettuate ai singoli Task per rendere l'intero sistema funzionante e permetterne così l'integrazione.

Per ogni task viene fornita la struttura ad albero dei file modificati di maggior rilievo.

Task 1

```
\---T1-G11\applicazione\mansclass
    +---docker-compose.yml
    +---Dockerfile
    +---pom.xml
    \---src\main\java\com\groom\mansclass\model\filesystem
        \---RobotUtil.java
```

- È stata aggiunta la generazione dei test e il calcolo della copertura da parte del robot EvoSuite contestualmente all'inserimento della classe da parte dell'amministratore. Per fare ciò si è reso necessario utilizzare il volume condiviso **VolumeT8** con il Task 8.

```
Path directory_EvoSuite = Paths.get("/VolumeT8/app/FolderTree/" + cname + "/" + cname + "SourceCode");
try {
    // Verifica se la directory esiste già
    if (!Files.exists(directory_EvoSuite)) {
        // Crea la directory
        Files.createDirectories(directory_EvoSuite);
        System.out.println(x:"La directory è stata creata con successo.");
    } else {
        System.out.println(x:"La directory esiste già.");
    }
} catch (Exception e) {
    System.out.println("Errore durante la creazione della directory: " + e.getMessage());
}
try (InputStream inputStream = multipartFile.getInputStream()) {
    Path filePath = directory_EvoSuite.resolve(fileName);
    System.out.println(filePath.toString());
    Files.copy(inputStream, filePath, StandardCopyOption.REPLACE_EXISTING);
}
ProcessBuilder processBuilderEvoSuite = new ProcessBuilder();
// Per simmetria, il numero di livelli generato da EvoSuite è uguale a quello di Randoop
// Tale numero è ottenuto considerando il numero di file prodotti
int l=resultsDir.listFiles().length;
String livelli=Integer.toString(l);
processBuilderEvoSuite.command(...command:"bash", "/VolumeT8/app/Prototipo2.0/robot_generazione.sh",
    |   |   cname, cname, "/VolumeT8/app/FolderTree/" + cname + "/" + cname + "SourceCode" , livelli);
processBuilderEvoSuite.directory(new File(pathname:"/VolumeT8/app/"));

Process processEvoSuite = processBuilderEvoSuite.start();

BufferedReader readerEvoSuite = new BufferedReader(new InputStreamReader(processEvoSuite.getInputStream()));
String lineEvoSuite;
while ((lineEvoSuite = readerEvoSuite.readLine()) != null)
    |   System.out.println(lineEvoSuite);
```

```

try {
    int exitCode = processEvoSuite.waitFor();

    System.out.println("ERRORE CODE: " + exitCode);
} catch (InterruptedException e) {
    System.out.println(e);
    e.printStackTrace();
}
File resultDirEvoSuite = new File("/VolumeT8/app/FolderTree/" + cname + "/RobotTest/EvoSuiteTest");
File resultsEvoSuite [] = resultDirEvoSuite.listFiles();
for(File result : resultsEvoSuite) {
    System.out.println(result.getAbsolutePath());
    int score = LineCoverageCSV(result.getAbsolutePath() + "/TestReport/statistics.csv");
    System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
    int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
    System.out.println("La copertura del livello " + String.valueOf(livello) + " e': " + String.valueOf(score));

    HttpClient httpClient = HttpClientBuilder.create().build();
    HttpPost httpPost = new HttpPost(uri:"http://t4-g18-app-1:9000/robots");
    JSONArray arr = new JSONArray();
    JSONObject rob = new JSONObject();

    rob.put(name:"scores", String.valueOf(score));
    rob.put(name:"type", value:"evoSuite");
    rob.put(name:"difficulty", String.valueOf(livello));
    rob.put(name:"testClassId", cname);
    arr.put(rob);

    JSONObject obj = new JSONObject();
    obj.put(name:"robots", arr);
    StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
    httpPost.setEntity(jsonEntity);
    HttpResponse response = httpClient.execute(httpPost);
}

```

Figure 3.4: RobotUtil.java - EvoSuite

- Il file *RobotUtil.java* è stato modificato con l'aggiunta della funzione **LineCoverageCSV**. Questa funzione consente di estrarre il parametro di copertura per linee dal file CSV generato da EvoSuite.

```

public static int LineCoverageCSV(String path) {
    try {
        // Creare un oggetto CSVReader
        CSVReader reader = new CSVReaderBuilder(new FileReader(path)).withSkipLines(skipLines:1).build();
        // Leggere le righe del file CSV
        String[] nextLine;
        while ((nextLine = reader.readNext()) != null) {
            // Verificare se la riga ha il formato desiderato
            if (nextLine[1].equals(anObject:"LINE")) {

                // Estrai e converti il valore in double, moltiplica per 100 e converte in intero
                double coverageDouble = Double.parseDouble(nextLine[2]) * 100;
                return (int) coverageDouble;
            }
        }
        // Chiudere il reader
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return -1;
}

```

Figure 3.5: RoboUtil.java - LineCoverageCsv

Task 5

```
\---T5-G2\t5
    +---docker-compose.yml
    +---Dockerfile
    +---pom.xml
    \---src
        \---main
            +---java\com\g2\t5
            |   \---GuiController.java
            \---resources
                +---static
                |   \---t5
                |       +---Icone
                |       \---js
                |           \---main.js
            \---template
                +---main.html
                \---editor.html
```

- Il file *main.html* è stato aggiornato con l'aggiunta di un bottone di **logout** e la relativa funzione JavaScript.

```
<div class="col-3">
<button id="logoutButton" type="button" style="margin-bottom:-50px" class="btn custom-button" th:onclick="logout()>Logout</button>
</div>
```

Figure 3.6: main.html - logoutButton

```
<!-- Funzione JavaScript per il logout -->
<script type="text/javascript">
    function logout() {
        // Effettua il reindirizzamento alla pagina di logout
        window.location.href = "/logout";
    }
</script>
```

Figure 3.7: main.html - Funzione JavaScript per logout

-
- Inoltre, la costruzione della lista di robot è stata modificata in quanto era fortemente dipendente da Randoop.

```
<span th:each="entry: ${hashMap2}" th:id="${'levels-' + entry.key}" class="spa levels">
    <button th:each="value: ${entry.value}" th:id="${entry.key + '-' + value}" type="button" class="lista-item my-button"
        th:text="${value}" th:attr="onclick=|Handlebuttonrobot('${value}', this)|"></button>
</span>
```

Figure 3.8: main.html – lista robot

- È stato necessario apportare modifiche all’handle del pulsante robot nel file *main.js* poiché il campo ’value’ ora contiene l’intera stringa (ad esempio ’Randoop LVL 1’ anziché solo ’1’).

```
function Handlebuttonrobot(id, button) {
    $(document).ready(function () {
        console.log('contenuto id' + id);
        var split=id.split(" ");
        robot = split[0];
        difficulty = split[2];
        console.log('Hai cliccato sul bottone del robot con id: ' + robot);
        // Se il bottone precedentemente selezionato è diverso da null
        // allora rimuoviamo la classe highlighted
        if (bottonePrecedente2 != null) {
            bottonePrecedente2.classList.remove("highlighted");
        }
        if (button.classList.contains("highlighted")) {
            button.classList.remove("highlighted");
        } else {
            button.classList.add("highlighted");
        }
        bottonePrecedente2 = button;
    });
}
```

Figure 3.9: main.js – Handlebuttonrobot

- La funzione **getLevels**, presente nel file *GuiController.java*, è stata modificata in modo da costruire correttamente la lista dei livelli. È importante notare che il numero di livelli di *EvoSuite* ora dipende dalla scelta effettuata in precedenza per il numero di livelli fissato ed è uguale a quello di *Randoop*.

```

public List<String> getLevels(String className) {
    List<String> result = new ArrayList<String>();
    int i;
    for(i = 1; i < 11; i++) {
        try {
            restTemplate.getForEntity("http://t4-g18-app-1:3000/robots?testClassId=" +
                className + "&type=randoop&difficulty="+String.valueOf(i), responseType: Object.class);
        } catch (Exception e) {
            break;
        }
        String lvl= "randoop lvl " + i;
        result.add(lvl);
    }
    System.out.println("indice corrente : " + i );
    for(int j=(1);j<i;j++){
        String lvlEvo= "evoSuite lvl " + j;
        result.add(lvlEvo);
        | System.out.println("indice corrente : " + j );
    }
    return result;
}

```

Figure 3.10: *GuiController.java* – *getLevels*

- Il file **editor.html** è stato aggiornato ed in particolare sono stati aggiunti i pulsanti per le nuove funzionalità di *logout* e *Run UserTest*. Sono stati implementati anche dei controlli sui pulsanti di *Run UserTest* e *HighlightLine* affinché non fossero più utilizzabili al termine della partita.

Inoltre a causa dell'incompatibilità di *Junit5* con la versione 1.0.6 di *EvoSuite* è stato necessario modificare la struttura di default del test case nel formato *Junit4*. In questo modo il test case da completare da parte dell'utente risulta essere misurabile con *EvoSuite*.

Infine si mostra come venga costruito diversamente l'output mostrato nella console *ConsoleArea2*, all'interno della quale vengono visualizzate tutte le metriche misurabili tramite *EvoSuite*.

```

consoleArea2.setValue(`Esito Risultati (percentuale di linee coperte)
Il tuo punteggio: ${response.score.toString()}% LOC
Il punteggio del robot: ${response.robotScore.toString()}% LOC
Le restanti statistiche sono :
Branch:           ${response.statistic[2]} % LOC
WeakMutation:     ${response.statistic[3]} % LOC
Exception:        ${response.statistic[4]} % LOC
Method:           ${response.statistic[5]} % LOC
Output:           ${response.statistic[6]} % LOC
CBranch:          ${response.statistic[7]} % LOC
MethodNoException: ${response.statistic[8]} % LOC`)
```

Figure 3.11: editor.html – output metriche EvoSuite

Task 6

```

\---T6-G12
    +---docker-compose.yml
    +---Dockerfile
\---T6
    +---pom.xml
    \---src\main\java\com\exemple\T6
        \---MyController.java
```

- Nel file **MyController.java** è stata modifica l' API */run* in quanto l'oggetto Json inviato dal task 7 nel campo *coverage* adesso contiene una stringa con tutte le metriche di *EvoSuite*, che devono essere separate in un array per una corretta gestione, oltre al report xml ottenuto con il tool *JaCoCo*, che risulta essere necessario per evidenziare le linee di codice coperte.

```

HttpEntity entity = response.getEntity();
String responseBody = EntityUtils.toString(entity);
JSONObject responseObj = new JSONObject(responseBody);
String statistic = responseObj.getString(name:"coverage");
String [] coverage=statistic.split(regex:" ");
System.out.println("Score Utente : "+ coverage[1]);
String outCompile = responseObj.getString(name:"outCompile");
String xml_string = responseObj.getString(name:"xml");
// PRESA DELLO SCORE UTENTE
Integer userScore= Integer.parseInt(coverage[1]);
```

Figure 3.12: MyController.java - score utente

- È stata aggiunta una API */GetReport* che si occupa della calcolare la copertura del test inserito dallo studente tramite *Evosuite*, mostrando a video tramite un alert la copertura per linee ottenuta.

Task 7

```
\---T7-G31\RemoteCCC
    +---docker-compose.yml
    +---Dockerfile
    +---pom.xml
    +---entrypoint.sh
    +---ClientProject
        |   \---codice.sh
    \---src\main\java\RemoteCCC
        +---Config.java
        \---App
            \---App.java
```

- È stata inserita, nel file **App.java**, una funzione per la lettura del file .csv prodotto da *EvoSuite*, all'interno della quale tutte le metriche calcolate vengono inserite in un vettore di stringhe.

```
public static String LineCoverageCSV(String path) {
try {
    // Creare un oggetto CSVReader
    CSVReader reader = new CSVReaderBuilder(new FileReader(path)).withSkipLines(skipLines:1).build();
    String statistic="";
    // Leggere le righe del file CSV
    String[] nextLine;
    while ((nextLine = reader.readNext()) != null) {
        double coverageDouble = Double.parseDouble(nextLine[2]) * 100;
        int coverage= (int) coverageDouble;
        statistic= statistic + " " + String.valueOf(coverage);
    }
    // Chiudere il reader
    reader.close();
    System.out.println("Output CSV : " + statistic);
    return statistic;
} catch (IOException e) {
    e.printStackTrace();
}
// Restituire un valore predefinito se il valore non è stato trovato
return "Errore csvCoverage";
}
```

Figure 3.13: App.java - LineCoverageCSV

- È stata modificata l'API */compile-and-codecoverage* in quanto adesso la risposta costruita contiene tutti i valori delle metriche di *EvoSuite* (Si noti che il campo CoverageFolder è stato necessariamente aggiunto all'oggetto Config.java), insieme al report xml di JaCoCo. In particolare nella funzione *compileExecuteCoverageWithMaven* viene eseguito il file “codice.sh” il quale contiene tutti i comandi per effettuare le differenti *MeasureCoverage* di *EvoSuite*.

```
if(compileExecuteCoverageWithMaven(output_maven,request)){
    //Salvo il path in cui verrà inserito il file csv
    Path path = Paths.get(Config.getCoverageFolder());
    //Estraggo il valore della copertura per linee tramite la funzione LineCoverageCSV
    String coverage=LineCoverageCSV(Config.getCoverageFolder());
    //Estraggo il file xml prodotto da jacoco
    String retXmlJacoco = readFileToString(Config.getxmlFolder());
    try {
        Files.delete(path);
        System.out.println("Il file è stato eliminato con successo.");
    } catch (IOException e) {
        System.err.println("Impossibile eliminare il file: " + e.getMessage());
    }
    //Costruisco la risposta con i valori ottenuti
    response.setError(error:false);
    response.setoutCompile(output_maven[0]);
    response.setCoverage(coverage);
    response.setXml(retXmlJacoco);
}
```

Figure 3.14: App.java - compileExecuteCoverageWithMaven

- È stata modificata la API relativa a *JaCoCo*, rinominata *highlightline* che non effettuerà più la misurazione di coverage (ora delegata ad Evosuite) ma manterrà la funzionalità di evidenziazione delle righe coperte.
- È stato aggiunto l'attributo xml a *ResponseDTO*, con relativi metodi Get e Set, in modo da poter correttamente inviare le informazioni fornite da *JaCoCo*.

```
private static class ResponseDTO{

    private Boolean error;
    private String outCompile;
    private String coverage;
    private String xml;
```

Figure 3.15: App.java - ResponseDTO

Task 8

```
\---T8-G21\Progetto_SAD_GRUPPO21_TASK8\Progetto_def\opt_livelli\Prototipo2.0
    +---docker-compose.yml
    +---Dockerfile
    +---docker-entrypoint.sh
    +---installazione.sh
    +---pom.xml
    \---Prototipo2.0
        +---misurazione_livelli.sh
        +---robot_generazione.sh
        \---generazione_livelli.java
```

- Il Task T8 risultava non funzionante nella sua versione locale a causa di errori legati alla struttura delle cartelle presenti. È stata quindi modificata tale struttura ed inoltre è stata eliminata la dipendenza dai package per avere un formato unificato e compatibile con *Randoop*. Infine è stato creato il file docker necessario per creazione del *container* e del *volume* T8 condiviso con il task T1.

3.4 Gateway Sequence Diagrams

Ogni richiesta da parte del client viene filtrata dai componenti UI Gateway e API Gateway: si ometterà negli altri sequence diagram questa interazione ai fini di una maggiore leggibilità e sarà apposta una apposita nota per notificarlo.

3.4.1 UI Gateway

Ogni volta che un client richiede l'accesso a un pagina web, si interfaccia con l'*UI Gateway* che si occupa di effettuarne il routing. L'interazione è descritta nel seguente diagramma di sequenza. Il client richiede una pagina web e l'UI gateway, dopo aver interrogato la propria tabella di routing, effettua la richiesta al microservizio richiesto per conto del client e gli inoltra la risposta.

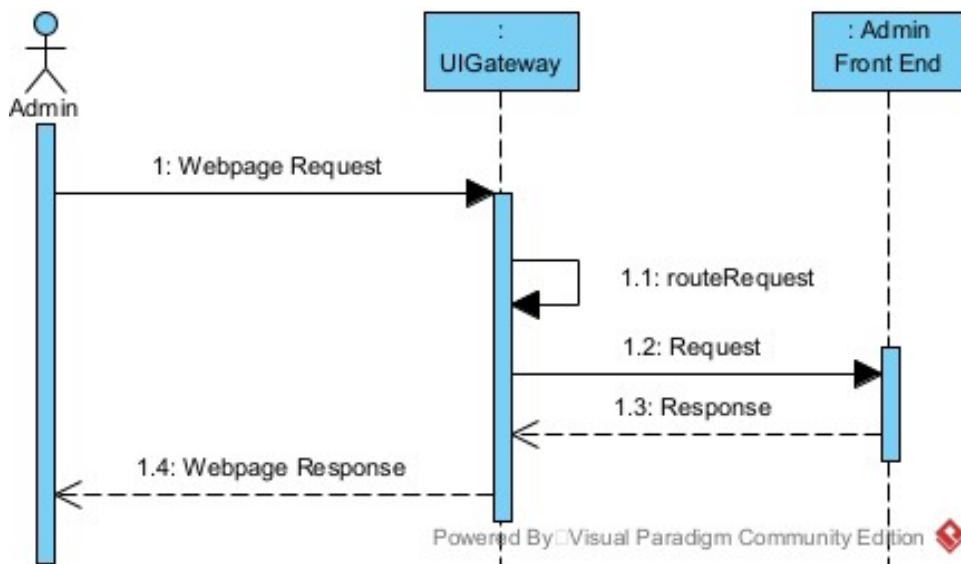


Figure 3.16: UI Gateway Sequence Diagram (Admin)

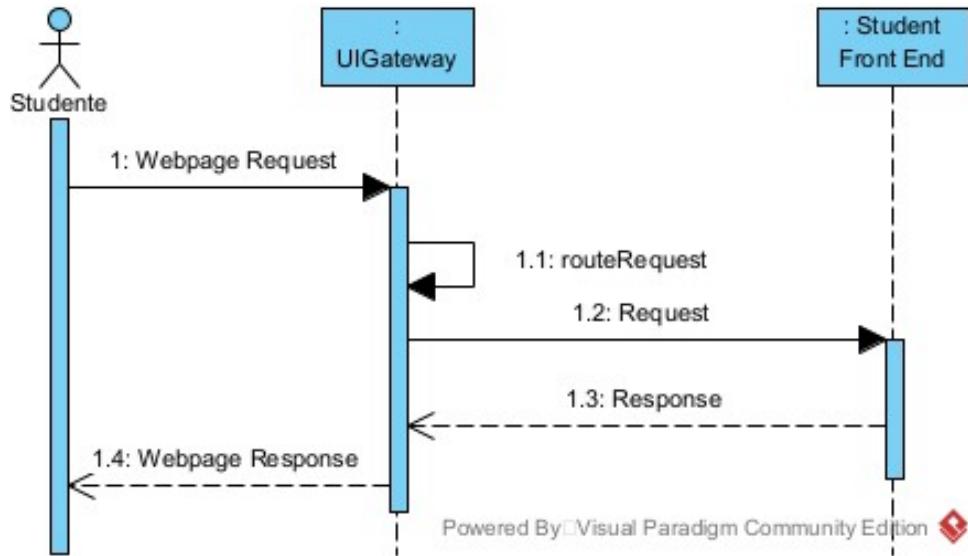


Figure 3.17: UI Gateway Sequence Diagram (Student)

3.4.2 API Gateway

Ogni volta che un client richiede il servizio di un API, si interfaccia con l'API Gateway che si occupa di effettuarne il routing, di verificarne l'autenticazione e di prepararne l'autorizzazione. L'interazione è descritta nel seguente diagramma di sequenza. Quando il client richiede l'utilizzo di un API, l'API Gateway effettua le seguenti operazioni:

1. estrae il token *JWT* contenuto in un cookie della richiesta
2. verifica che il token *JWT* sia valido, utilizzando il servizio offerto dallo Student Repository (nel caso in cui non lo sia, restituisce un errore *"Not Authorized"*)
3. estrae i claim contenuti all'interno del token *JWT* (ovvero l'*ID* dello studente) e li aggiunge come *Header* di autorizzazione della richiesta
4. interroga la propria tabella di routing
5. effettua la richiesta al microservizio richiesto per conto del client e gli inoltra la risposta

Come si può notare, in realtà questo componente non implementa completamente un meccanismo di autorizzazione, ma la avvia: tale responsabilità, infatti, è comunque lasciata al singolo microservizio, permettendo così di dare un maggiore controllo e versatilità ad ognuno di essi.

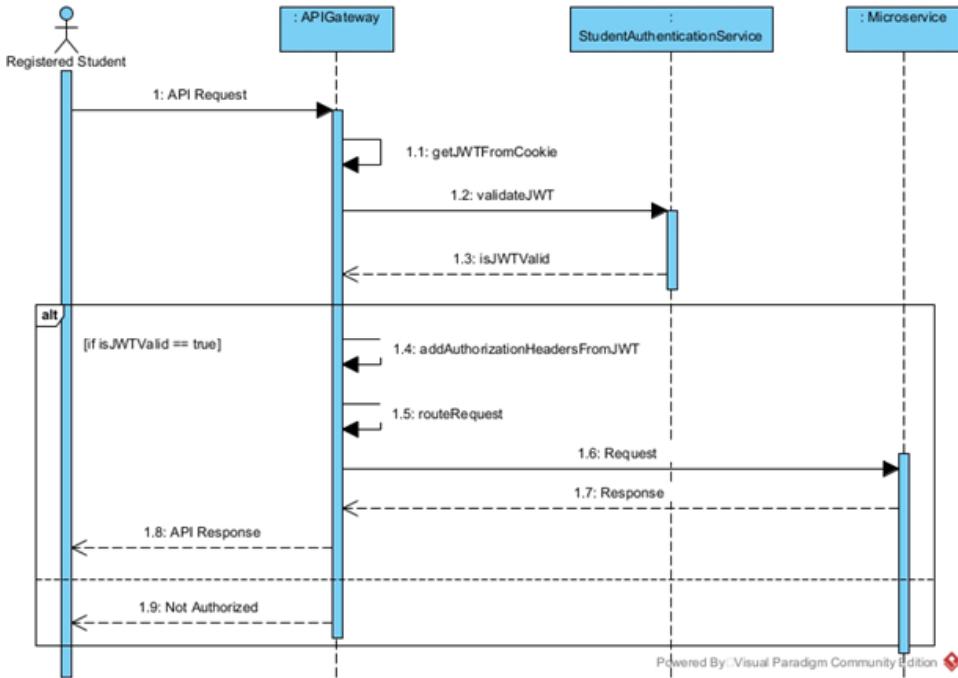


Figure 3.18: API Gateway Sequence Diagram

3.5 Sequence Diagrams

Al fine di avere una panoramica completa del funzionamento dell'applicazione web desiderata funzionante, vengono riportati i principali sequence diagrams, tenendo conto che sono state apportate alcune modifiche alle funzioni implementate dai task e aggiunte delle altre.

3.5.1 Inspect ClassUnderTest

Questo caso d'uso si attiva dopo la scelta della classe, del robot e del livello da sfidare da parte dello studente, all'atto della pressione del tasto *submit*. Permette il caricamento in una apposita schermata del codice della classe che lo studente ha scelto di testare. Vengono effettuate le seguenti operazioni:

1. Viene effettuata una richiesta **GET** all'API */receiveClassUnderTest* (realizzata dal Task 6) presente nel Game Engine Controller e indicando nei parametri il nome della classe. Si omette il passaggio per l'API Gateway descritto precedentemente.

2. Il Game Engine Controller, a sua volta, effettua una richiesta GET all'API /download-File (realizzata dal Task 1) presente nella Class Repository Controller e indicando il nome della classe all'interno dell'URL, per riceverne il codice.
3. Se tutto va a buon fine il codice viene ritornato come stringa e poi visualizzato nell'apposito riquadro, altrimenti si ritorna all'attore studente l'errore HTTP che ha impedito il successo dell'operazione.

I nomi usati per le entità sono stati modificati da quelli reali per rendere più comprensibile l'operazione.

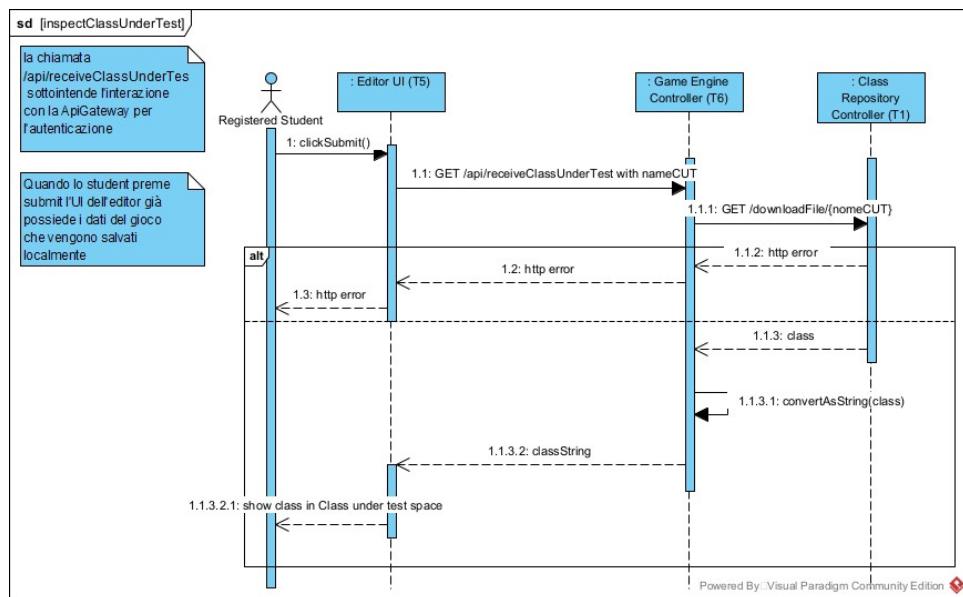


Figure 3.19: Inspect ClassUnderTest Sequence Diagram

3.5.2 Create Game

Questo caso d'uso permette la creazione della partita, a partire dalla selezione della classe da testare e dell'avversario da affrontare. Per UI dell'editor si considera tutto il front end sviluppato all'interno del task 5. Vengono effettuate le seguenti operazioni:

1. L'Editor UI mostra la schermata di selezione della classe all'utente studente, egli la sceglie e gli verrà poi mostrata, in base alla scelta, una lista di robot disponibili classificati per livello.
2. L'Editor UI chiede conferma della selezione complessiva. Lo studente clicca poi *Submit* e viene svolto il caso d'uso incluso *InspectClassUnderTest* che restituisce il codice della classe scelta da mostrare poi successivamente a video.

-
3. Viene poi effettuata una richiesta **POST** all'API */save-data* (realizzata dal Task 5) e presente nel GUI Controller. Qui avviene anche una fase di controllo sull'autorizzazione all'uso dell'API, ovvero se l'utente autenticato che ha passato i controlli dell'API gateway, sta facendo una richiesta legittima di salvataggio della propria partita (funzione `checkApiAuthorization`). Se ciò fallisce si restituisce all'utente l'errore.
 4. Viene dunque creato un oggetto Game con le informazioni sulla partita che viene poi utilizzato dalla funzione presente in Game Data Writer, che si occupa del salvataggio vero e proprio.
 5. Game Data Writer legge tutti i campi necessari dall'oggetto Game ricevuto
 6. Game Data Writer effettua poi una serie di richieste **POST** all'API */games*, */rounds* e */turns* verso il Game Repository (realizzato dal Task 4) per creare una partita, un round ed un turno.
 7. Infine, le informazioni relative alla partita appena creata vengono restituite all'Editor UI per essere memorizzate e viene visualizzato l'Editor che permette allo studente di giocare la partita.

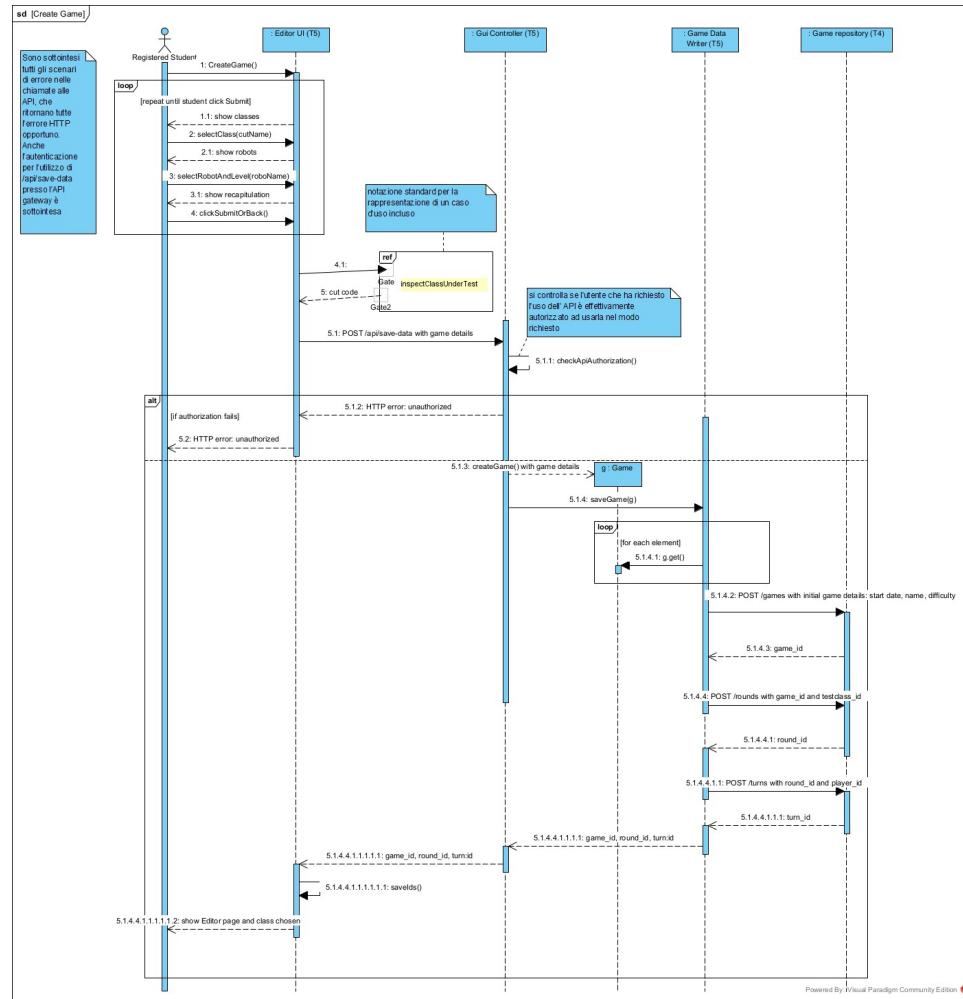


Figure 3.20: Create Game Sequence Diagram

3.5.3 Run

Questo caso d'uso permette la conclusione e il salvataggio della partita, nonché il calcolo del **vincitore**. Si attiva all'atto di pressione del tasto *Run* da parte dello studente mentre si trova nell'editor. Vengono effettuate le seguenti operazioni:

1. L'**Editor UI** effettua una richiesta **POST** all'API */run* (realizzata dal Task 6) presente nel **Game Engine Controller**, che si occupa di svolgere tutte le azioni necessarie, inserendo all'interno del body le informazioni sulla partita effettuata e il codice sottomesso.
2. Il Game Engine Controller effettua a sua volta una richiesta **POST** all'API */compile-and-codecoverage* (realizzata dal Task 7) che ha il compito di compilare il codice dell'utente e calcolarne la **coverage**, restituendo quest'ultima assieme all'output di compilazione e al report **csv**.

3. Il Game Engine Controller effettua una richiesta **GET** all'API `/robots` verso il **Game Repository** (realizzato dal Task 4), ottenendo così il **punteggio** del robot scelto dall'utente.
4. Il Game Engine Controller effettua poi una serie di richieste **PUT** alle API `/turns`, `/rounds` e `/games` verso il Game Repository (realizzato dal Task 4) per aggiornare le informazioni sulla partita, sul turno e sul round (indicando nel caso del turno anche chi è il vincitore).
5. Vengono mostrati i **risultati** della partita, tutte le metriche di **coverage** di **EvoSuite** e le linee di codice coperte dal test.

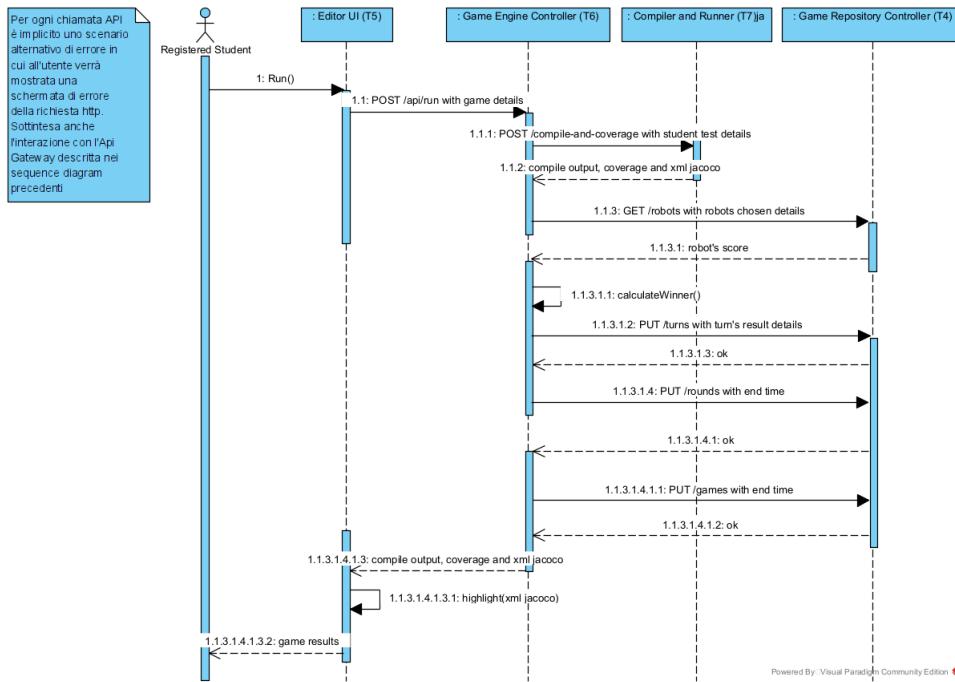


Figure 3.21: Run Sequence Diagram

3.5.4 Run UserTest

Questo caso d'uso si occupa del calcolo della copertura del codice scritto dall'utente studente tramite **EvoSuite**. Si attiva alla pressione del tasto *RunUserTest* presente nell'**Editor UI**. Vengono effettuate le seguenti operazioni:

1. L'Editor UI si rivolge tramite una richiesta **POST** all'API `/getReport` (realizzata dal Task 6), presente nel **Game Engine Controller**, fornendo nel body il codice dell'utente.

2. A sua volta il Game Engine Controller si rivolge, tramite una richiesta **POST** all'API */compile-and- codecoverage* (realizzata dal Task 7) che ha il compito di compilare il codice dell'utente e calcolarne la coverage.
3. In caso di errore, questo viene restituito fino all'utente. Altrimenti, al Game Engine Controller viene ritornata la **coverage** che viene trasmessa all' **Editor UI** e visualizzata dall'utente sottoforma di **alert**. In questo modo l'utente può ricevere un feedback sull'andamento della partita e può migliorarsi senza sottomettere il tentativo.

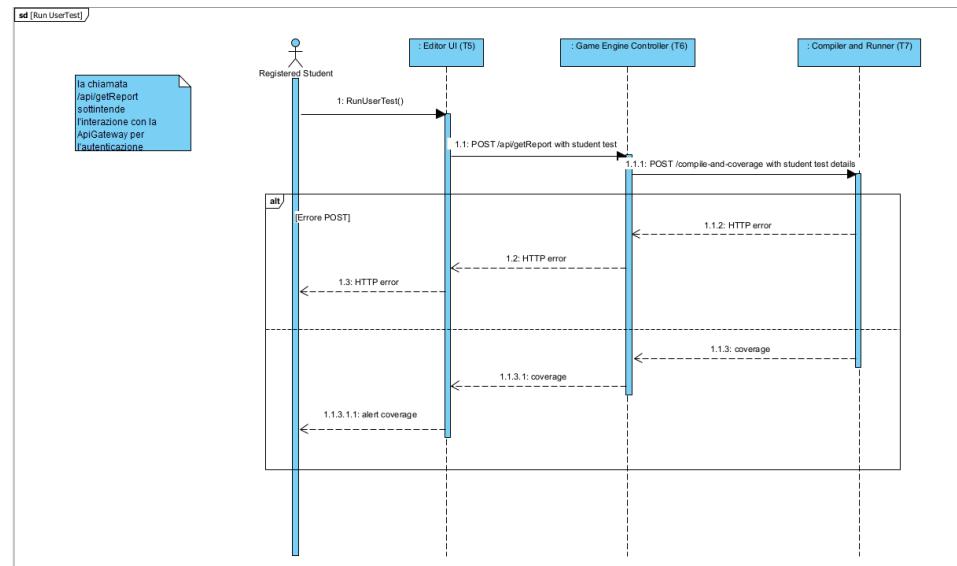


Figure 3.22: Run UserTest Sequence Diagram

3.5.5 Run with JaCoCo

Questo caso d'uso si occupa di **evidenziare** la copertura del codice scritto dall'utente studente. Si attiva alla pressione del tasto *JaCoCo* presente nell'**Editor UI**. Vengono effettuate le seguenti operazioni:

1. L'**Editor UI** si rivolge tramite una richiesta **POST** all'API */getJaCoCoReport* (realizzata dal Task 6), presente nel Game Engine Controller, fornendo nel body il codice dell'utente.
2. A sua volta il **Game Engine Controller** si rivolge, tramite una richiesta **POST** all'API */compile-and- codecoverage* (realizzata dal Task 7) che ha il compito di compilare il codice dell'utente e calcolarne la coverage.
3. In caso di errore, questo viene restituito fino all'utente. Altrimenti, al Game Engine Controller vengono ritornate le coverage e l'output di compilazione: solo la

copertura viene trasmessa all' Editor UI che **evidenzierà** le linee di codice della classe sotto test nel seguente modo: se la linea è coperta, parzialmente coperta e non coperta, si usano rispettivamente i colori **verde**, **giallo** e **rosso**.

In questo modo l'utente può ricevere un feedback sull'andamento della partita e può migliorarsi senza sottomettere il tentativo.

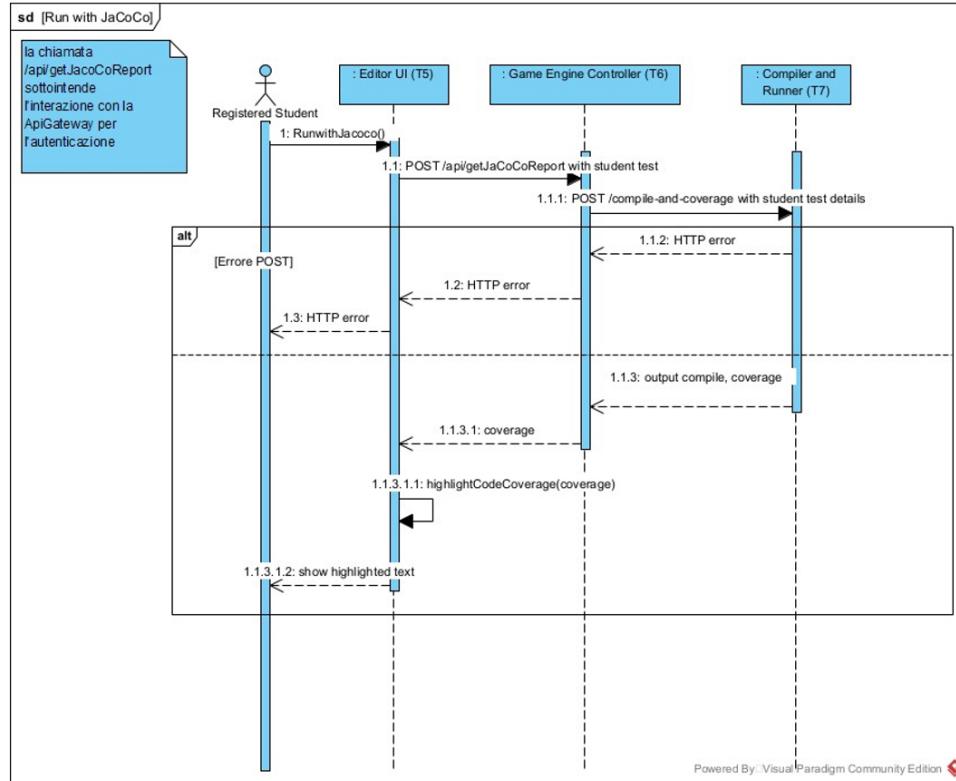


Figure 3.23: Run with JaCoCo Sequence Diagram

3.5.6 Compile

Questo caso d'uso permette di **compilare** il tentativo dello studente e di visualizzare il risultato della compilazione in un apposito **riquadro**. Si attiva alla pressione del tasto *Compile* presente nell'**Editor UI**. Vengono effettuate le seguenti operazioni:

1. L'Editor UI si rivolge tramite una richiesta **POST** all'API */sendInfo* (realizzata dal **Task 6**), presente nel **Game Engine Controller**, fornendo nel body il codice dell'utente.
2. A sua volta il Game Engine Controller si rivolge, tramite una richiesta **POST** all'API */compile-and- get-coverage* (realizzata dal **Task 7**) che ha il compito di compilare il codice dell'utente e calcolarne la coverage.

3. In caso di errore, questo viene restituito fino all'utente. Altrimenti, al Game Engine Controller vengono ritornate le coverage e l'output di compilazione: solo l'**output** di **compilazione** viene trasmesso all'**Editor UI** che lo mostrerà a video.

In questo modo l'utente può ricevere un feedback sull'andamento della partita e vedere se effettivamente il codice prodotto compila e, se non lo fa, quali sono gli errori mostrati dal compilatore.

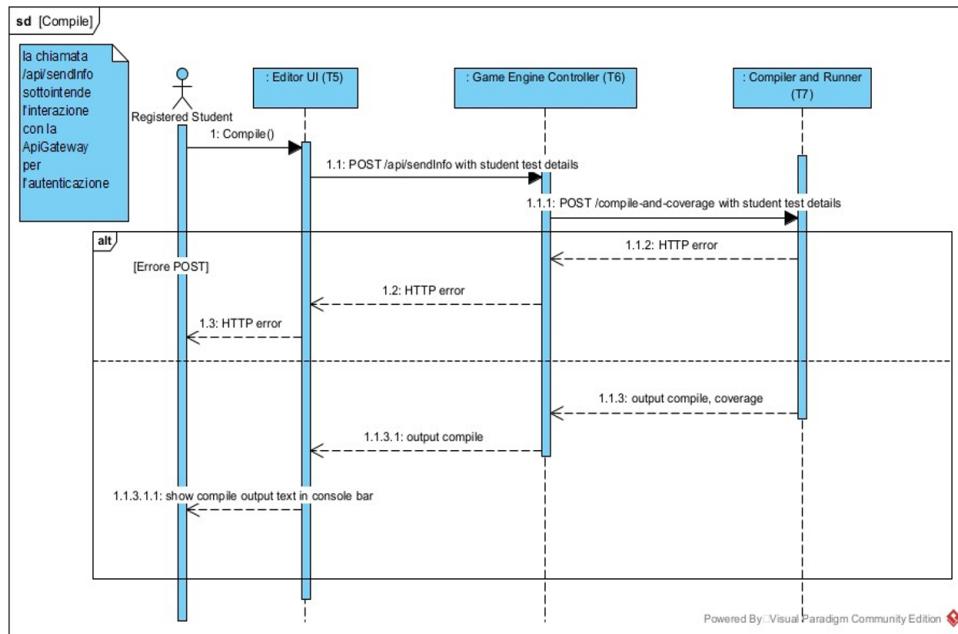


Figure 3.24: Compile Sequence Diagram

3.5.7 Add class

Questo caso d'uso permette all'amministratore di **inserire** una nuova classe per il gioco. Si attiva alla pressione del testo *Add Class* presente sull'**Admin UI**. Vengono effettuate le seguenti operazioni:

La funzione nuova è la *generateAndSaveRobots()* della classe **RobotUtil**: ha come parametro il file della classe e permette ai robot *Randoop* ed *Evosuite* di generare i test. In questa funzione avviene anche il salvataggio del punteggio, per ogni livello generato, delle coperture del robot calcolato con il tool *Emma* per Randoop mentre è Evosuite stesso a effettuare le proprie misurazioni di copertura. Per far ciò, viene effettuata una richiesta di tipo **POST** all'API */robots* verso il **Game Repository**, che ne permette la memorizzazione.

Si noti che, per come è stato implementato il caso d'uso dal task 1, il file **sorgente** della classe si troverà sul **File System** (nella cartella *Files-Upload*), mentre i **metadati** relativi (nome, percorso in cui è memorizzato, data, categoria etc.) saranno sul *database*.

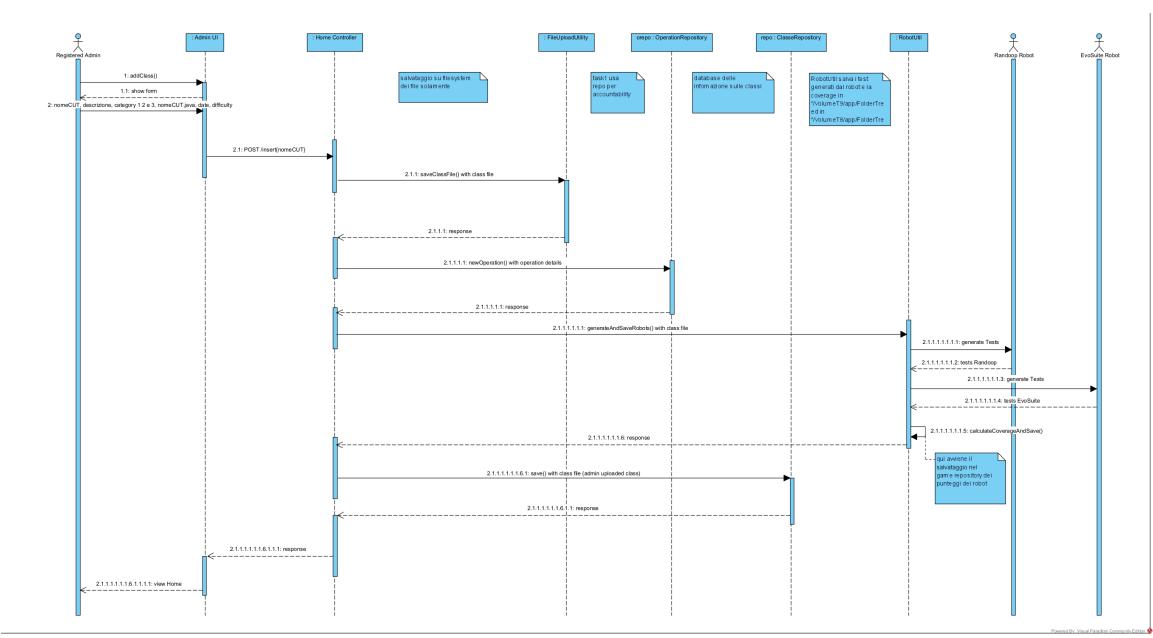


Figure 3.25: Add Class Sequence Diagram

3.6 Activity Diagrams

Si riporta di seguito l'unico activity diagram modificato rispetto alle precedenti documentazioni.

3.6.1 Add Class

Con questo diagramma si rende più evidente la scelta di **integrazione** fatta per la **generazione** dei **test** con i robot *Randoop* ed *Evosuite*.

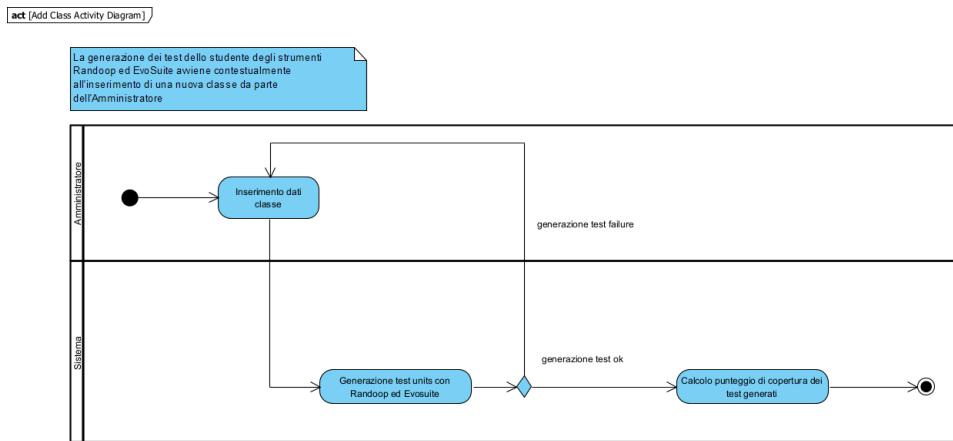


Figure 3.26: Add Class Activity Diagram

-4-

Deployment

Nel seguente diagramma si mette in evidenza la **disposizione fisica** dei **componenti** di un sistema software e la loro **interazione** all'interno di un ambiente di esecuzione. Il server che ospita l'applicativo utilizza come sistema operativo **Windows 11** e, più precisamente, il software è in esecuzione nell'ambiente **Docker**, distribuito sui vari container (quelli evidenziati in **giallo** sono stati **modificati** e mentre quelli in **rosso** sono stati **aggiunti** da noi). Si può notare la presenza di altri tre elementi specifici: il volume "VolumeT8", il volume "VolumeT9" e la rete "Global Network".

- Il *VolumeT9* è condiviso dai container dei Task 1 e 9, ciò risulta necessario in quanto entrambi devono poter accedere alla stessa porzione del File System: quella relativa alle classi e alla copertura generate da **Randoop** ed **Emma**.
- Il *VolumeT8* è analogamente condiviso dai container dei Task 1 e 8, la porzione del **File System** in comune è quella relativa alle classi e alla copertura generate da **Evosuite**.
- La *rete* è **condivisa** dai container di tutti i Task, ciò permette di **isolare** i container dalla rete dell'host e al tempo stesso permettere la **comunicazione** tra di essi (si ottiene così una sorta di rete virtuale): l'unico container che è esposto verso l'esterno è il *Gateway* che si occupa di gestire le richieste.

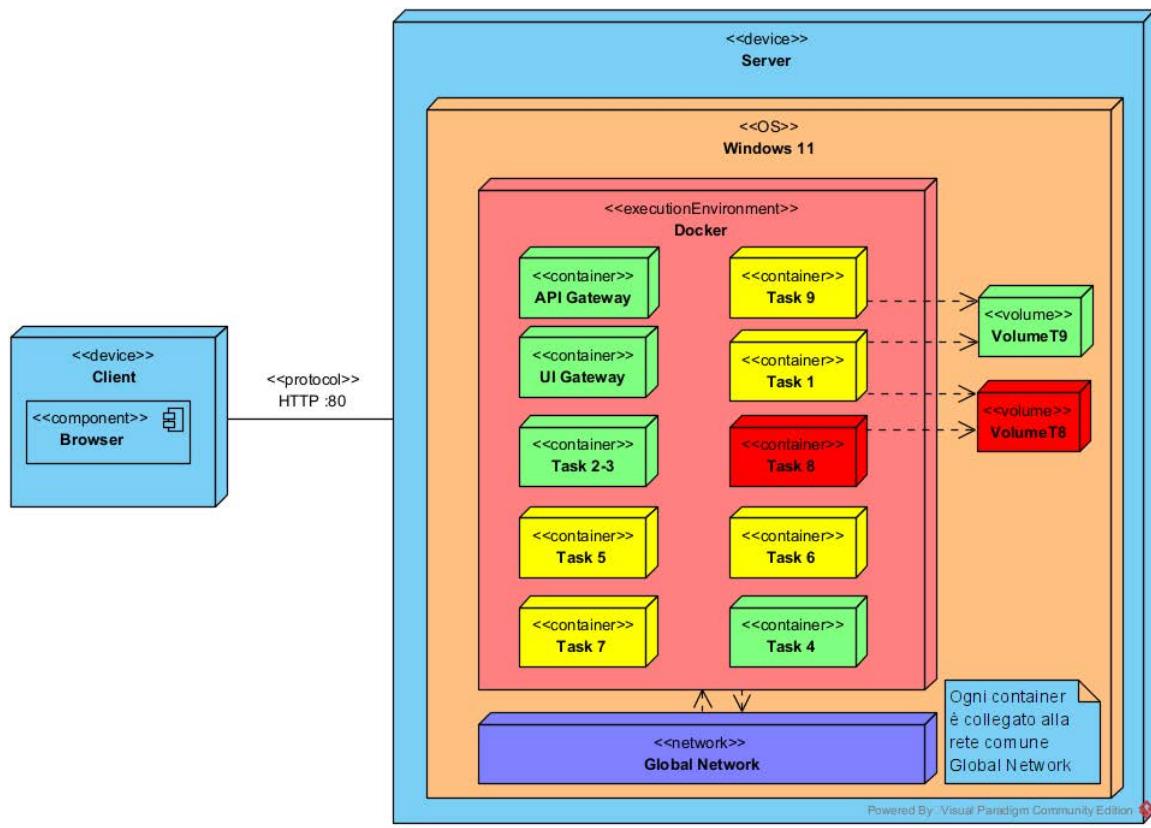


Figure 4.1: Diagramma di Deployment

—5—

Testing

Il nostro obiettivo è l'**integrazione** di tutti i componenti per ottenere un'applicazione funzionante. A tale scopo, è stato condotto il "**Test End-to-End (E2E)**", noto anche come "**Test di Sistema**".

Si tratta di un metodo di testing che valuta l'**intero** flusso dell'applicazione, garantendo che **tutti** i componenti, combinati tra loro, operino come previsto in scenari reali. Infatti, il software viene testato dal punto di vista dell'**utente finale**, simulando uno scenario realistico, inclusa l'**interfaccia utente**, i servizi di **backend**, i **database** e la **comunicazione di rete**. Tra gli scopi c'è quello di convalidare il comportamento **complessivo** dell'applicazione, includendo **funzionalità, affidabilità, prestazioni e sicurezza**.

L'obiettivo finale del test E2E è individuare **difetti o problemi** che potrebbero sorgere quando le diverse parti dell'applicazione interagiscono tra loro. Solitamente, il test E2E viene eseguito dopo il test di integrazione, che verifica i singoli moduli, e prima del test di accettazione dell'utente, che assicura che l'applicazione soddisfi i requisiti dell'utente.

5.1 Tool Utilizzati

Per effettuare questo tipo di test è stato utilizzato **Selenium**, uno strumento per il testing **automatizzato** per applicazioni web, insieme a **JUnit**, il noto framework di testing, e **Google Chrome**, il browser utilizzato per collegarsi all'applicazione.

5.2 Casi di Test

Di seguito sono riportati i vari test effettuati. I flussi dell'applicazione testati sono quelli relativi all'accesso (**Login Test**) e all'utilizzo dell'editor (**Editor Test**). In entrambi si possono individuare due fasi importanti: una relativa alla **configurazione** dell'ambiente e un'altra relativa ai **test** veri e propri.

5.2.1 Login Test

Setup

La fase di **configurazione** consiste nel definire quali operazioni devono essere effettuare in via preliminare, prima che i test comincino. Entrando più nello specifico, tali operazioni riguardano:

- l'impostazione del **driver** di **Selenium** che gli permette di utilizzare il browser
- l'**apertura** del browser all'avvio di un test
- la **chiusura** del browser al termine di un test

```
private static ChromeDriver driver;
private static int timeout = 60;

@BeforeClass
public static void setDriver() {
    System.setProperty(key:"webdriver.chrome.driver",
        value:"C:\\\\Users\\\\didom\\\\Downloads\\\\chromedriver-win64\\\\chromedriver-win64\\\\chromedriver.exe");
}

@Before
public void openBrowser(){
    driver = new ChromeDriver();
    driver.manage().timeouts().implicitlyWait(timeout, TimeUnit.SECONDS);
}

@After
public void closeBrowser(){
    driver.close();
}
```

Figure 5.1: Login Test - Setup

Credenziali valide

Tale test verifica che l'**accesso** avvenga correttamente utilizzando credenziali valide (ovvero di uno studente già **registrato**). Per fare ciò, il robot effettua le seguenti operazioni:

1. si collega alla pagina di accesso (*/login*)
2. inserisce le credenziali (email e password)
3. clicca il pulsante di *accesso*
4. verifica che sia stato effettuato il redirect alla prima pagina dell'editor (*/main*)

```

@Test
public void validCredentials(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("test@gmail.com");
    driver.findElement(By.id("password")).sendKeys("Testpass0");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    String urlPaginaDiRedirezione = "http://localhost/main";
    try {
        wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    Assert.assertEquals("Test fallito! Il login non è avvenuto correttamente.",
        driver.getCurrentUrl(), urlPaginaDiRedirezione);
}

```

Figure 5.2: Login Test - Credenziali valide

Password errata

Tale test verifica che l'accesso venga negato utilizzando una password **non** valida. Per fare ciò, il robot effettua le seguenti operazioni:

1. si collega alla pagina di accesso (*/login*)
2. inserisce le credenziali (email e password)
3. clicca il pulsante di *accesso*
4. verifica che venga visualizzato a video l'errore *Incorrect password*

```

@Test
public void invalidCredentials(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("test@gmail.com");
    driver.findElement(By.id("password")).sendKeys("password");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.textToBe(By.tagName("body"), "Incorrect password"));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    Assert.assertEquals("Test fallito! Il login è avvenuto correttamente.",
        driver.findElement(By.tagName("body")).getText(), "Incorrect password");
}

```

Figure 5.3: Login Test – Password errata

E-mail errata

Tale test verifica che l'accesso venga negato utilizzando una e-mail **non** valida. Per fare ciò, il robot effettua le seguenti operazioni:

1. si collega alla pagina di accesso (*/login*)
2. inserisce le credenziali (email e password)
3. clicca il pulsante di *accesso*
4. verifica che venga visualizzato a video l'errore *E-mail not found*

```
@Test
public void incorrectEmail(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("incorrecttest@gmail.com");
    driver.findElement(By.id("password")).sendKeys("password");
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.textToBe(By.tagName("body"), "Email not found"));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    Assert.assertEquals("Test fallito! Il login è avvenuto correttamente.",
        driver.findElement(By.tagName("body")).getText(), "Email not found");
}
```

Figure 5.4: Login Test – Email errata

5.2.2 Editor Test

Setup

La fase di **configurazione** consiste nel definire quali operazioni devono essere effettuare in via preliminare, prima che i test comincino. Entrando più nello specifico, tali operazioni riguardano:

- l'impostazione del **driver** di **Selenium** che gli permette di utilizzare il browser
- l'impostazione del **percorso** dei download
- l'**apertura** del browser e l'autenticazione dell'utente all'avvio di un test
- la **chiusura** del browser al termine di un test

```

private static ChromeDriver driver;
private static int timeout = 60;

@BeforeClass
public static void setDriver() {
    System.setProperty(key:"webdriver.chrome.driver",
                       value:"C:\\Users\\didom\\Downloads\\chromedriver-win64\\chromedriver.exe");
}

@Before
public void openBrowser() {
    ChromeOptions options = new ChromeOptions();
    options.setCapability(CapabilityType.UNEXPECTED_ALERT_BEHAVIOUR, UnexpectedAlertBehaviour.ACCEPT);
    HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
    chromePrefs.put(key:"profile.default_content_settings.popups", value:0);
    chromePrefs.put(key:"download.default_directory", value:"C:\\Users\\didom\\Downloads");
    options.setExperimentalOption("prefs", chromePrefs);

    driver = new ChromeDriver(options);
    driver.manage().timeouts().implicitlyWait(timeout, TimeUnit.SECONDS);

    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("test@gmail.com");
    driver.findElement(By.id("password")).sendKeys("Testpass0");

    // Ora esegui l'istruzione desiderata
    driver.findElement(By.cssSelector("input[type=submit]")).click();

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    String urlPaginaDiRedirezione = "http://localhost/main";
    try {
        wait.until(ExpectedConditions.urlToBe(urlPaginaDiRedirezione));
    } catch (TimeoutException e) {
        Assert.fail();
    }
}

@After
public void closeBrowser() {
    driver.close();
}

```

Figure 5.5: Editor Test - Setup

Download classe

Tale test verifica che il download di una classe avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. clicca la prima classe disponibile
2. clicca il pulsante di *download*
3. verifica che il file scaricato sia presente all'interno dei download

```

@Test
public void download() throws InterruptedException {
    driver.findElement(By.id("0")).click();
    Thread.sleep(millis:1000);
    driver.findElement(By.id("downloadButton")).click();

    File f = new File(pathname:"C:\\Users\\didom\\Downloads\\class.java");

    Thread.sleep(millis:5000);

    Assert.assertTrue(f.exists());
}

```

Figure 5.6: Editor Test - Download classe

Scelta classi e robot

Tale test verifica che la scelta di una classe e di un robot avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. seleziona la classe e il robot
2. clicca il pulsante di *conferma*
3. verifica che sia stato effettuato il redirect alla pagina di conferma (*/report*)

```
@Test
public void selection() {
    String urlPaginaDiRedirezione = "http://localhost/report";

    moveToReport(urlPaginaDiRedirezione);

    Assert.assertEquals("Test fallito! La selezione non è avvenuta correttamente.",
        driver.getCurrentUrl(),
        urlPaginaDiRedirezione);
}
```

Figure 5.7: Editor Test - Riepilogo scelta

Avvio partita

Tale test verifica che l'avvio di una partita avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. clicca il pulsante di *conferma* delle scelte effettuate
2. verifica che sia stato effettuato il redirect all'editor (*/editor*)

```
@Test
public void startGame() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    Assert.assertEquals("Test fallito! L'avvio della partita non è avvenuto correttamente.",
        driver.getCurrentUrl(),
        urlPaginaDiRedirezione);
}
```

Figure 5.8: Editor Test - Avvio partita

Compilazione classe utente

Tale test verifica che la **compilazione** della classe di test scritta dall'utente avvenga correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. clicca il pulsante di *compilazione*
3. verifica che sia visibile il risultato della compilazione

```
@Test
public void compile() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("compileButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#console-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail();
    }
}
```

Figure 5.9: Editor Test - Compilazione classe utente

Copertura classe utente con Jacoco

Tale test verifica che la copertura della classe di test scritta dall'utente avvenga correttamente e che vengano **sottolineate** correttamente le linee di codice coperte dall'utente. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. clicca il pulsante *jacocoButton*
3. verifica che sia visibile il risultato della copertura

```

@test
public void highlightJacoco() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan
            (By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch(TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("jacocoButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan
            (By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > * .uncovered-line"), 0));
    } catch(TimeoutException e) {
        Assert.fail();
    }
}

```

Figure 5.10: Editor Test - Copertura classe utente

Submit della partita

Tale test verifica che il tentativo dell’utente venga consegnato e che la partita venga processata correttamente. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. clicca il pulsante di *StartGame*
3. verifica che sia visibile l’esito della partita

```

@test
public void run() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("runButton")).click();

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#console-textarea2 + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail();
    }
}

```

Figure 5.11: Editor Test - Submit della partita

Logout main

Tale test verifica che il logout dell'utente venga effettuato correttamente dalla pagina **main** e che si venga reindirizzati alla pagina di login. Per fare ciò, il robot effettua le seguenti operazioni:

1. clicca il pulsante di *logout*
2. verifica che sia stato effettuato il redirect alla pagina di login (*/login*)

```
@Test
public void logout() {

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    driver.findElement(By.id("logoutButton")).click();

    try {
        wait.until(ExpectedConditions.urlToBe("http://localhost/login"));
    } catch (TimeoutException e) {
        Assert.fail();
    }
}
```

Figure 5.12: Logout test

Logout editor

Tale test verifica che il logout dell’utente venga effettuato correttamente dalla pagina di **editor** e che si venga reindirizzati alla pagina di login. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. clicca il pulsante di *logout*
3. verifica che sia stato effettuato il redirect alla pagina di login (*/login*)

```
@Test
public void logoutEditor() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail();
    }

    driver.findElement(By.id("logoutButton")).click();

    try {
        wait.until(ExpectedConditions.urlToBe("http://localhost/login"));
    } catch (TimeoutException e) {
        Assert.fail();
    }
}
```

Figure 5.13: Editor Test- Logout test

RunUserTest

Tale test verifica che la **copertura** della classe di test scritta dall’utente avvenga correttamente tramite **EvoSuite**. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. clicca il pulsante di *EvoSuite*
3. verifica che sia visibile il risultato della copertura

```
@Test
public void runUserTest() {
    String urlPaginaDiRedirezione = "http://localhost/editor";
    moveToEditor(urlPaginaDiRedirezione);

    WebDriverWait wait = new WebDriverWait(driver, timeout);

    try {
        wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(
            By.cssSelector("#sidebar-textarea + div > * div.CodeMirror-code > *"), 1));
    } catch (TimeoutException e) {
        Assert.fail("Element not found within the specified timeout.");
    }

    driver.findElement(By.id("coverageButton")).click();

    wait.until(ExpectedConditions.alertIsPresent());

    Alert alert = driver.switchTo().alert();

    String alertText = alert.getText();
    System.out.println(alertText);
    alert.accept();

    if (alertText.equals(anObject:"Risultato copertura Test utente : 0 %")) {
        System.out.println(x:"Esito positivo: Copertura test utente è al 0%.");
    } else {
        Assert.fail("Esito negativo: Copertura test utente non è al 0%.");
    }
}
```

Figure 5.14: Editor Test- runUserTest

5.3 Risultati

Di seguito si riportano i risultati dei test con l'**output** di **Maven**.

```
Results:

Tests run: 12, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time:  01:37 min
[INFO] Finished at: 2023-10-30T17:21:20+01:00
[INFO] -----
```

Figure 5.15: Risultati “mvn test”

–6–

Installazione

In questo capitolo è riportata la procedura di installazione dell'intera applicazione. Essa avviene in maniera completamente **automatica** tramite l'avvio del file batch "*installer.bat*" che funge da installer. Tale procedura è stata semplificata e resa possibile grazie all'utilizzo di **Docker**: la logica su cui si basa prevede di accedere alle cartelle dei singoli Task in cui sono presenti i file di configurazione "*docker-compose.yml*" ed effettuare così l'installazione di un Task alla volta.

Nota: la procedura dettagliata con eventuali errori durante l'installazione è indicata nel file *readme* del progetto.

6.1 Passo 1

Si deve avviare lo script "*installer.bat*". Saranno effettuate le seguenti operazioni:

1. creazione della rete "global-network" comune a tutti i container
2. creazione del volume "VolumeT9" comune ai Task 1 e 9
3. creazione del volume "VolumeT8" comune ai Task 1 e 8
4. installazione di ogni singolo container
5. esecuzione dei file di installazione nei container del task T8 e T7
6. avvio dei container

6.2 Passo 2

Si deve configurare il container "manvsclass-mongo db-1" così come descritto anche nella documentazione del Task 1. Per fare ciò bisogna fare le seguenti operazioni:

1. posizionarsi all'interno del terminale del container
2. digitare il comando "mongosh"
3. digitare i seguenti comandi:

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

6.3 Passo 3

L'intera applicazione è adesso pienamente **configurata** e raggiungibile sulla porta :80, raggiungibile cioè all'indirizzo "<http://localhost>".

-7-

Guida alle future integrazioni

Di seguito sono indicate delle procedure per effettuare l'**integrazione** di ulteriori futuri Task, mantenendo l'architettura generale da noi proposta.

7.1 Integrazione Container

Per procedere con l'integrazione del **container** all'interno della stessa **rete condivisa**, si deve utilizzare il seguente file di configurazione "*docker-compose.yml*":

```
version: '2'
services:
  #sostituire "app" con il nome del container app:
    build: ./app
    expose:
      # sostituire "8000" con la porta utilizzata
      - 8000
    networks:
      - global-network
  networks:
    global-network:
      external: true
```

7.2 Integrazione con UI Gateway

Per procedere con l'integrazione di nuovo **Front End**, si deve modificare il file di configurazione *"/ui gateway/default.conf"* utilizzato da Nginx andando ad aggiungere un nuovo blocco "location":

```
# sostituire "webpage" con l'endpoint utilizzato che serve il frontend
location ~ ^/(webpage) {
    include /etc/nginx/includes/proxy.conf;
    # sostituire "app" con il nome del container e "8000" con la porta
    # utilizzata proxy_pass http://app:8000;
}
```

7.3 Integrazione con API Gateway

Per procedere con l'integrazione di nuovo **endpoint REST API**, si deve modificare il file di configurazione *"/api gateway/src/resources/application.yml"* utilizzato da Netflix Zuul andando ad aggiungere un nuovo blocco "route":

```
routes:
  #sostituire "app" con il nome del servizio che si vuole dare a questo endpoint app-service:
  sensitiveHeaders:
    ##sostituire "endpoint" con il nome dell'endpoint
    path: /api/endpoint/**
    url: http://app:8000/endpoint
```

7.4 Integrazione Installer

Per procedere con l'integrazione all'interno dell'**installer**, si deve modificare lo script batch di installazione *"installer.bat"* andando a modificare la lista delle cartelle in cui sono presenti i file di configurazione *"docker-compose.yml"*:

```
REM Definizione dei percorsi delle directory da visitare
set list= "./T1-G11/applicazione/manvsclass" "./T23-G1" "./T4-G18" "./T5-G2/t5"
"./T6-G12/T6" "./T7-G31/RemoteCCC" "./T8-G21\Progetto_SAD_GRUPPO21_TASK8\Progetto_def\opt_livelli\Prototipo2.0"
"./T9-G19\Progetto-SAD-G19-master" "./api_gateway" "./ui_gateway"
```