



UNIVERSITÀ<sup>9</sup> DEGLI STUDI DI  
NAPOLI FEDERICO II

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

Elaborato finale in **Software Architecture Design**

# *Integrazione Man vs Automated Testing Tools challenges*

Anno Accademico 2022-2023

Candidati  
**Chiara Capocello**  
matr. M63001451

**Luigi Cecere**  
matr. M63001413

**Michelangelo Formato**  
matr. M63001519

**Giovanni Gentile**  
matr. M63001450

# Indice

<b>Introduzione</b>	<b>1</b>
<b>1 Descrizione del progetto</b>	<b>2</b>
1.1 Introduzione al progetto . . . . .	2
1.1.1 Workflow Diagram . . . . .	6
1.2 Metodologia adottata . . . . .	8
<b>2 Analisi dei requisiti</b>	<b>10</b>
2.1 User Stories . . . . .	10
2.2 Use Case Diagram . . . . .	12
2.3 Scenari . . . . .	13
2.3.1 Create Game . . . . .	13
2.3.2 Add Class . . . . .	16
2.3.3 Run . . . . .	19
2.3.4 Submit . . . . .	21
2.3.5 Logout . . . . .	23
2.3.6 Storico . . . . .	25
2.3.7 Miglioramenti . . . . .	27
<b>3 Progettazione</b>	<b>28</b>

3.1	Architettura a microservizi . . . . .	28
3.2	Component Diagram . . . . .	29
3.2.1	Test Class Manager . . . . .	30
3.2.2	Student Front End . . . . .	31
3.2.3	EvoSuite Runner . . . . .	31
3.2.4	Game Engine . . . . .	32
3.2.5	Game Repository . . . . .	32
3.2.6	Randoop Runner . . . . .	32
3.3	Composite Structure Diagram . . . . .	33
<b>4</b>	<b>Scrum e Iterazioni</b>	<b>35</b>
4.1	Introduzione a Scrum . . . . .	35
4.2	User Stories totali . . . . .	36
4.2.1	Criteri di accetazione . . . . .	39
4.3	Prima iterazione . . . . .	43
4.3.1	Iteration Planning iterazione 1 . . . . .	43
4.3.2	Primo Sprint . . . . .	44
4.3.3	Secondo Sprint . . . . .	46
4.3.4	Terzo Sprint . . . . .	47
4.3.5	Quarto Sprint . . . . .	48
4.4	Seconda iterazione . . . . .	49
4.4.1	Primo Sprint . . . . .	50
4.4.2	Secondo Sprint . . . . .	51
4.4.3	Terzo Sprint . . . . .	52
4.4.4	Quarto Sprint . . . . .	54
4.4.5	Sprint EXTRA . . . . .	56

<b>5 Modifiche effettuate</b>	<b>57</b>
5.1 Iterazione 1 - Sprint 1 . . . . .	57
5.2 Iterazione 1 - Sprint 2 . . . . .	58
5.2.1 Eliminazione dipendenze Task8 . . . . .	58
5.2.2 Creazione container Task 8 . . . . .	60
5.3 Iterazione 1 - Sprint 3 . . . . .	63
5.3.1 Adattamento Task 8 . . . . .	63
5.3.2 Modifiche task 1 . . . . .	67
5.3.3 Omologazione del numero di livelli generati .	70
5.4 Iterazione 1 - Sprint 4 . . . . .	72
5.4.1 Modifiche task 5 - Backend . . . . .	72
5.4.2 Modifiche task 5 - Frontend . . . . .	76
5.5 Iterazione 2 - Sprint 1 . . . . .	80
5.5.1 Modifica server Task 8 . . . . .	81
5.5.2 Modifica script di generazione statistiche Task 8	83
5.5.3 Modifica Docker file e Docker compose Task 8	86
5.6 Iterazione 2 - Sprint 2 . . . . .	88
5.6.1 Creazione e gestione richiesta http POST bot-	
tone RUN . . . . .	88
5.6.2 Adattamento dipendenza del task 7 da JUnit4	89
5.6.3 Adattamento bottoni Run e Submit . . . . .	90
5.6.4 Sequence Diagram Run . . . . .	91
5.7 Iterazione 2 - Sprint 3 . . . . .	92
5.7.1 Creazione e gestione richiesta http POST bot-	
tone Submit . . . . .	92
5.7.2 Creazione "Logout" in interfaccia di Editor .	93

5.7.3	Creazione "Logout" in interfaccia di scelta della classe . . . . .	95
5.7.4	Sequence Diagram Logout . . . . .	95
5.8	Iterazione 2 - Sprint 4 . . . . .	96
5.8.1	Modifica logica di fine partita . . . . .	96
5.8.2	Caso di sconfitta - creazione nuovo turno . . . . .	97
5.8.3	Creazione e gestione "Storico" . . . . .	100
5.8.4	Salvataggio del valore di copertura . . . . .	103
5.8.5	Sequence Diagram Storico . . . . .	104
5.8.6	Ulteriori metriche di copertura di Evosuite . . . . .	105
5.8.7	Incremento Portabilità . . . . .	107
5.9	Iterazione 2 - Sprint EXTRA . . . . .	107
5.9.1	Visualizzazione del test nello storico per ogni turno . . . . .	107
5.10	SVILUPPI FUTURI . . . . .	110
5.10.1	Integrazione nuovi task . . . . .	112
<b>6</b>	<b>Deployment</b>	<b>115</b>
<b>7</b>	<b>Testing</b>	<b>118</b>
7.1	Testing delle API . . . . .	118
7.1.1	Richiesta http POST task 8 per statistiche . . . . .	118
7.1.2	Richiesta http GET task 8 per test . . . . .	120
7.2	Testing End to End . . . . .	121
7.2.1	Casi di Test . . . . .	122
7.2.2	Login test . . . . .	122
7.2.3	Editor test . . . . .	125

7.2.4	Risultati . . . . .	134
<b>8</b>	<b>Guida all'installazione e all'utilizzo</b>	<b>136</b>
8.1	Installazione . . . . .	136
8.1.1	Passo 1 . . . . .	136
8.1.2	Passo 2 . . . . .	137
8.1.3	Passo 3 . . . . .	138
8.2	Utilizzo . . . . .	140
8.2.1	Registrazione Admin . . . . .	140
8.2.2	Upload della classe da testare . . . . .	142
8.2.3	Registrazione Utente . . . . .	147
8.2.4	Guida all'interfaccia di editor . . . . .	149
<b>9</b>	<b>Glossario</b>	<b>156</b>
9.1	Robot . . . . .	156
9.2	Turno . . . . .	156
9.3	Giocatore . . . . .	157
9.4	Docker . . . . .	157
9.5	Trello . . . . .	157
9.6	Postman . . . . .	159
9.7	Miro . . . . .	159

# Introduzione

Il progetto European iNnovative AllianCe fot TESTing, **ENACTEST**, vuole valorizzare l'importanza del testing in quanto lo sviluppo e poi il lancio in web delle Application spesso presenta delle falle non ancora individuate ma che si sarebbero dovute fixare durante la fase di Testing del suddetto progetto.

Il Testing però risulta ancora una disciplina poco curata e valorizzata.

L'Università di Napoli Federico II, insieme a piccole imprese, vuole coltivare l'uso della disciplina del Testing attraverso un **Educational Game** dove i Player sfidano un software di generazione automatica di test nella copertura dei casi di test.

Questo dovrebbe incentivarne l'apprendimento rendendolo più accattivante.

# Capitolo 1

# Descrizione del

# progetto

**Man vs Automated Testing Tools challenges** è un educational game in cui gli studenti competono contro strumenti in grado di generare automaticamente casi di test JUnit (ad esempio Randooop o EvoSuite), nel raggiungimento di un determinato obiettivo di copertura (ad esempio copertura LOC).

## 1.1 Introduzione al progetto

In questa documentazione è descritto come è stato portato avanti il lavoro di integrazione della Web App iniziato dal T10-G40 che si è occupato dell'integrazione dei seguenti task:

Task	T1	T2-3	T4	T5	T6	T7	T9
Gruppo	G11	G1	G18	G2	G12	G31	G19

Nella pagina web GitHub <https://github.com/Testing-Game-SAD-2023> si può scaricare il lavoro di ogni singolo gruppo.

Ogni gruppo aveva il compito di implementare dei requisiti funzionali accorpati in task.

Per analisi più dettagliate dei singoli task si raccomanda di consultare la documentazione messa a disposizione da ogni gruppo.

Il compito portato avanti in questo progetto è quello di integrare l'ultimo task non ancora presente, ovvero T8-G21, che fornisce “**EvoSuite**”, un tool per:

- **La generazione automatica di test** per classi java da testare.

Nel contesto di questo progetto, l'obiettivo è di potersi confrontare con più robot (vedi 9.1) contro cui gareggiare, per valutare la copertura del proprio test.

- **Il calcolo della copertura** delle classi di test sfruttando diversi criteri di misurazione.

Sono state aggiunte inoltre **nuove funzionalità** alla applicazione, spiegate nel dettaglio nei prossimi capitoli, quali:

- la possibilità di poter effettuare un **Logout** da parte dell'utente

- La possibilità di effettuare più **turni in una sola partita**
- La possibilità di visualizzare **uno storico** dei turni nella partita corrente

Di seguito è riportata la modifica del diagramma di contesto ad alto livello (fig. 1.1): all'atto della creazione della partita, il giocatore prima dell'integrazione poteva scegliere solo robot generati da **Randoop**.

In seguito al lavoro illustrato in questa documentazione, il giocatore può scegliere quale tool sfidare tra Randoop e **Evosuite**.

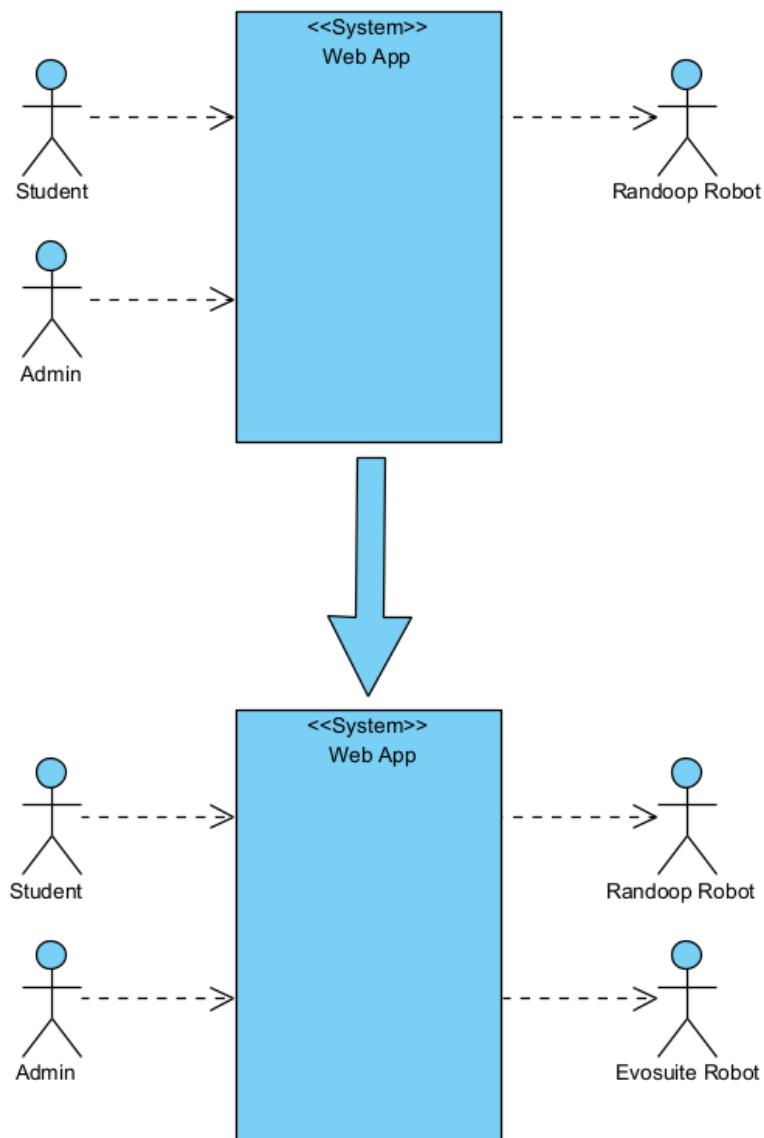


Figura 1.1: Diagramma di contesto ad alto livello

### 1.1.1 Workflow Diagram

Allo scopo di descrivere il contesto di gioco, di seguito (fig.1.2) è riportato un workflow diagram di alto livello, in cui il giocatore dapprima effettua il **Login**, poi effettua la **scelta della classe** da testare e il tool avversario contro cui gareggiare e, dopo aver confermato la scelta, **scrive effettivamente il test** della classe nella schermata di editor.

Viene successivamente effettuata la **compilazione** del test, poi un controllo dell'esito della stessa e **calcolato il punteggio utente**.

In seguito all'integrazione di Evosuite, il punteggio finale è calcolato sia da Jacoco che da Evosuite.

Successivamente il sistema fornisce il **confronto dei punteggi** misurati sia per l'utente che per il Robot in quel turno, e poi **li mostra a video**.

Il lavoro svolto in questo progetto oltre all'integrazione già descritta, è stato quello di aggiungere la possibilità di giocare più turni nella stessa partita.

Pertanto il worflow riportato descrive anche l'opportunità di **riscrivere il test** e tentare di battere il Robot in caso di sconfitta, nella stessa partita, sulla medesima classe scelta.

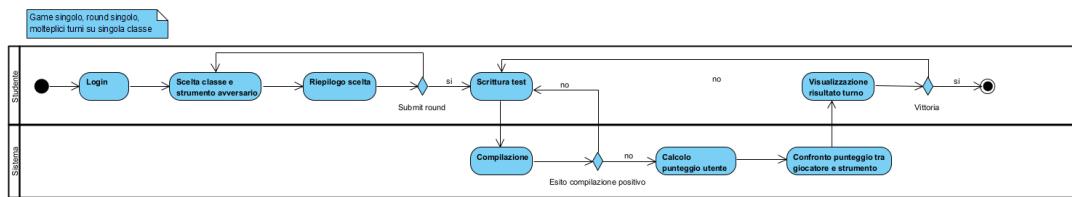


Figura 1.2: Workflow diagram

## 1.2 Metodologia adottata

Il carico di lavoro è stato suddiviso fra tutti i membri del gruppo nel corso del tempo seguendo un approccio **Agile** basato sul paradigma **SCRUM**, un Framework per la gestione iterativa ed incrementale del ciclo di sviluppo di un software.

Per la realizzazione del task assegnato, sono state effettuate **due iterazioni**, ognuna formata da **quattro sprint** della durata di circa una settimana ciascuno, per una durata totale di circa 2 mesi.

Ogni iterazione è stata organizzata al seguito di un'**iteration planning** in cui è stato deciso quali delle User Stories implementare in quella iterazione.

Sono stati inoltre utilizzati tool come **Trello** e **Miro** per tenere traccia delle attività da svolgere nello sprint corrente, quelle già eseguite, quelle future e quali sono state le problematiche riscontrate nel corso del tempo.

La maggior parte del lavoro svolto è stato eseguito in sede Universitaria, intervallandolo con videochiamate e possibilità di condivisione dello schermo per dare un contributo personale alle scelte di progetto e alla stesura del codice.

Una spiegazione nel dettaglio delle attività svolte nelle iterazioni seguendo il paradigma Agile è fornita nel Capitolo 4.4.5.

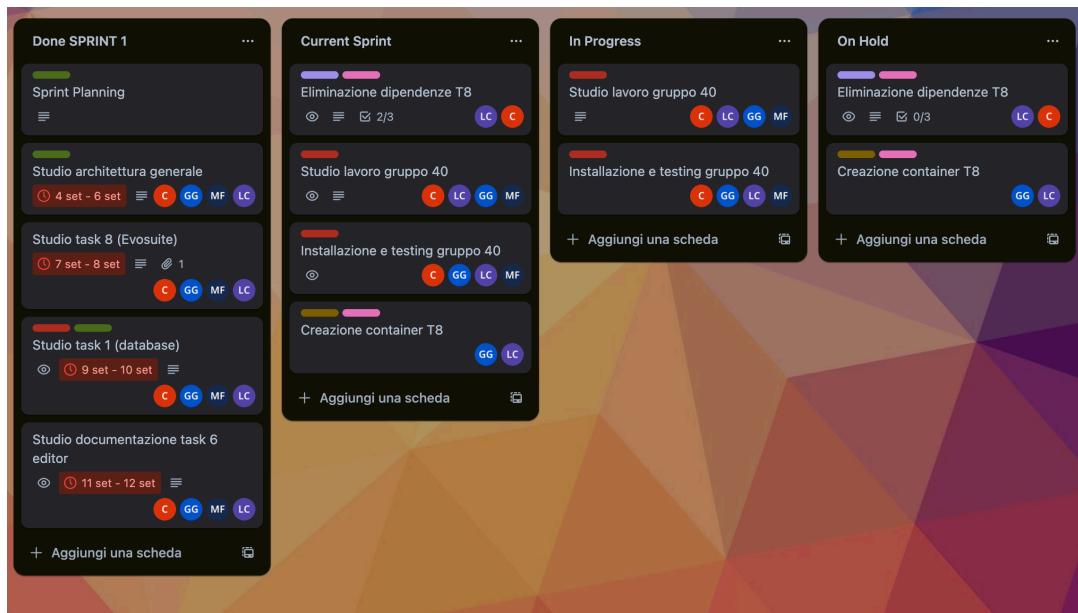


Figura 1.3: Trello

La figura mostra soltanto un esempio dell'utilizzo di Trello nel secondo sprint della prima iterazione, con la suddivisione degli sprint in compiti da eseguire, i compiti che si stanno svolgendo ( "*in Progress*" ) e quelli da svolgere nei giorni futuri ( "*On Hold*" ).

# Capitolo 2

## Analisi dei requisiti

L'Analisi dei requisiti riportata in questo capitolo riguarda esclusivamente il lavoro di integrazione e l'inserimento di nuove funzionalità apportate dal gruppo, ma si rimanda ai singoli task una documentazione più dettagliata e approfondita degli altri progetti.

### 2.1 User Stories

Con il susseguirsi delle iterazioni si è arrivati all'ideazione di ben 7 User Stories (figura 2.1) per stabilire gli obiettivi finali che si vogliono raggiungere.

Ogni storia rappresenta un **requisito di sistema** e seguendo il paradigma SCRUM la struttura con la quale vengono descritte è:

- **Come:** descrive il tipo di utente coinvolto

- **Vorrei:** rappresenta il requisito/funzionalità di sistema
- **In modo tale:** indica l'obiettivo da raggiungere

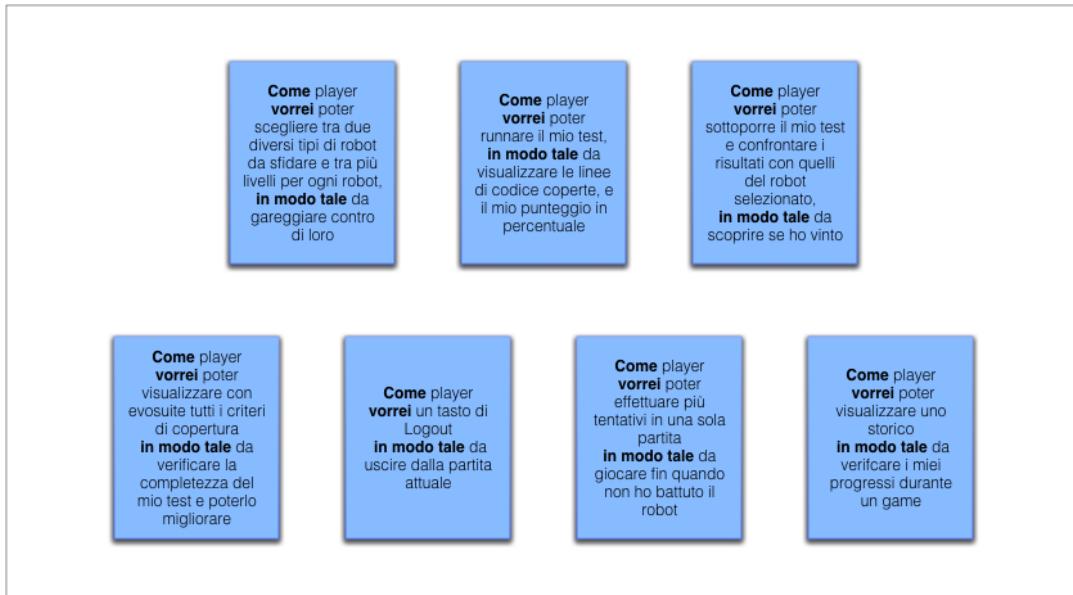


Figura 2.1: User Stories

Tutte le stories utente sono state portate a termine seguendo il processo iterativo descritto nel Capitolo 4.  
La lista completa delle User Stories implementate è riportata in figura.

## 2.2 Use Case Diagram

In accordo con gli obiettivi finali da raggiungere, il diagramma dei casi d'uso complessivo di tutte le funzionalità, a valle delle modifiche effettuate risulta essere quello che segue:

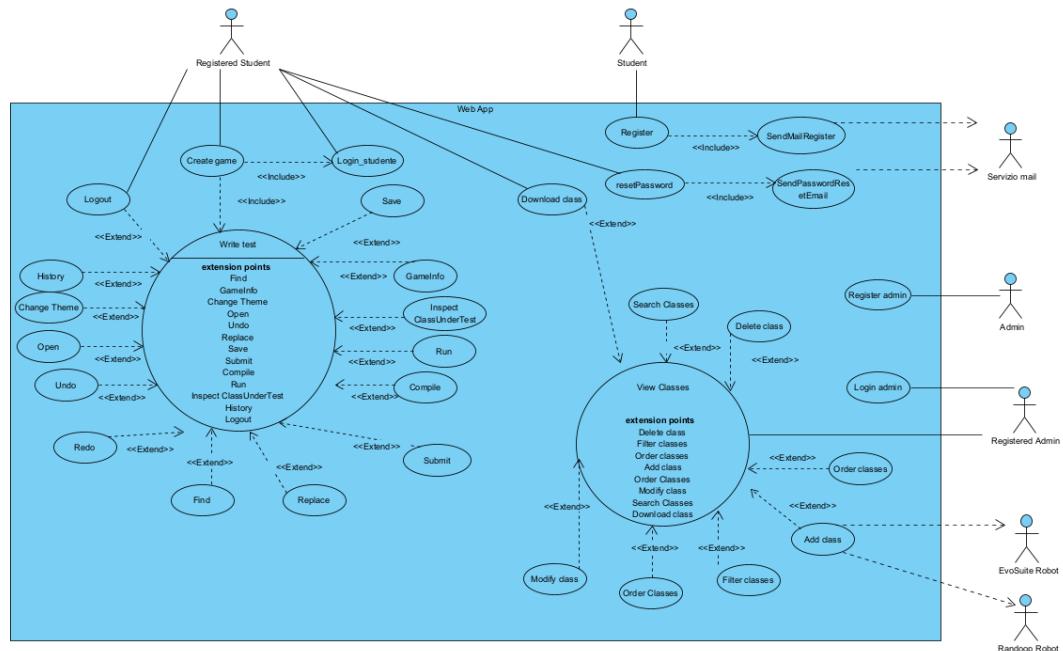


Figura 2.2: Use case diagram

- **”EvoSuite Robot”** è un attore secondario che interagisce con il caso d'uso di aggiunta di una nuova classe da parte dell’Admin.
- Il caso d'uso **“Logout”** permette la disconnessione dello Studente Registrato dal Game in corso.

- **History** (o Storico) è un caso d'uso che permette la visualizzazione in console dello storico dei turni da parte dello studente registrato.

## 2.3 Scenari

In questa sezione si evidenziano le modifiche che subiscono i casi d'uso in seguito all'integrazione del task 8, in quanto un nuovo attore secondario è stato aggiunto alla Web App.

Inoltre, sono stati aggiunti nuovi casi d'uso finalizzati ad ampliare l'esperienza di gioco del giocatore 9.3 .

### 2.3.1 Create Game

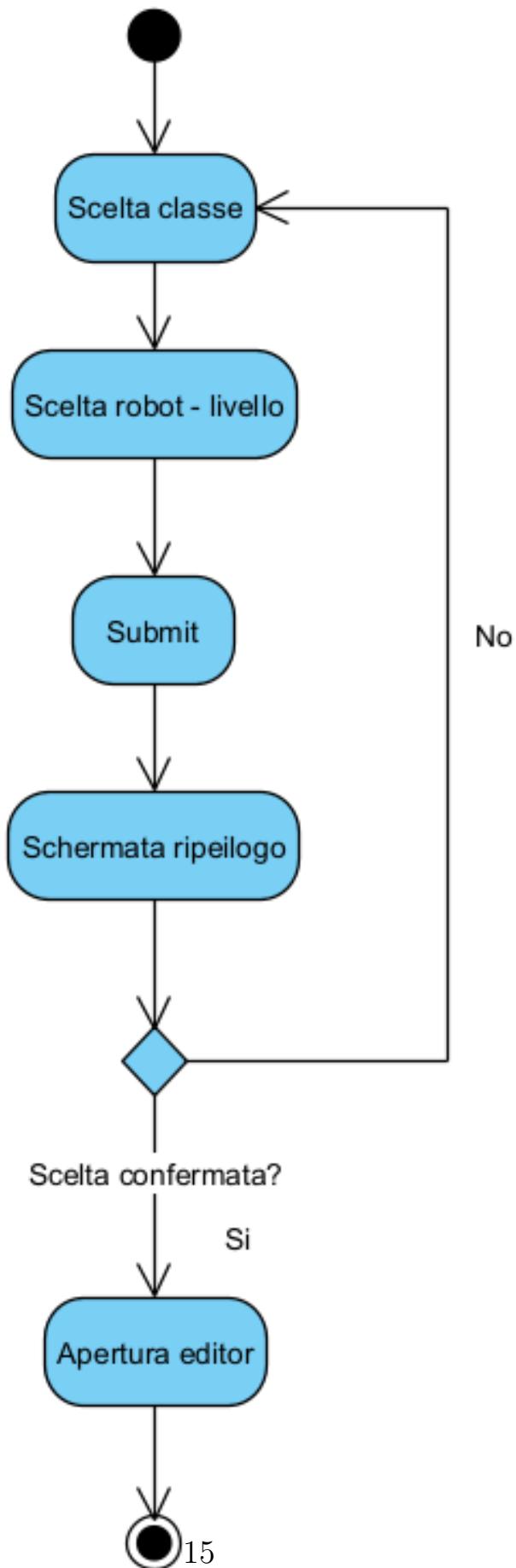
La descrizione del caso d'uso **Create Game** è invariata ma a seguito dell'integrazione del Task 8 il sistema mostra due tipologie di robot tra cui scegliere e i rispettivi livelli di difficoltà.

CASO D'USO	CREATE GAME
<b>ATTORE PRIMARIO</b>	Registered Student
<b>DESCRIZIONE</b>	Il giocatore sceglie una classe da testare e il robot con il livello da affrontare
<b>PRE-CONDIZIONI</b>	Il giocatore ha effettuato il login
<b>SEQUENZA DI EVENTI</b>	<ol style="list-style-type: none"> <li>1. Il caso d'uso inizia quando il giocatore ha effettuato il login.</li> <li>2. Il sistema mostra una lista di classi.</li> <li>3. Il giocatore sceglie la classe.</li> <li>4. Il sistema mostra i robot e i livelli disponibili.</li> <li>5. Il giocatore sceglie il robot e il livello e clicca “Submit”.</li> <li>6. Il sistema mostra un riepilogo allo studente.</li> <li>7. Il giocatore conferma.</li> <li>8. Il sistema salva le informazioni e mostra al giocatore l'editor.</li> </ol>
<b>PRINCIPALE</b>	
<b>POST-CONDIZIONE</b>	Il giocatore può scrivere il suo test.
<b>SEQUENZA DI EVENTI</b>	Il giocatore non clicca “Submit” ma “Back”, il sistema lo riporta alla schermata di selezione classi.
<b>ALTERNATIVI</b>	

Figura 2.3: Create Game Use Case

### Activity Diagram Create Game

Di seguito l'activity Diagram sull'insieme dei passi che compie il giocatore a partire dalla scelta della classe.



In questo flusso di esecuzione si integra al caso d'uso la parte relativa al robot **EvoSuite**: Al momento della selezione del robot, l'utente può scegliere tra due diverse tipologie di robot contro cui gareggiare.

Il resto delle azioni che l'utente può compiere resta invariato.

### 2.3.2 Add Class

Per quanto riguarda il caso d'uso **Add Class**, una volta aggiunta una nuova classe anche **Evosuite** genera i test relativi alla classe e calcola la copertura raggiunta.

CASO D'USO	ADD CLASS
<b>ATTORE PRIMARIO</b>	Registered Admin
<b>ATTORE SECONDARIO</b>	Robot <i>Randoop</i> - Robot <i>EvoSuite</i>
<b>DESCRIZIONE</b>	L'admin aggiunge una classe al repository delle class under test
<b>PRE-CONDIZIONI</b>	L'admin ha effettuato il login
<b>SEQUENZA DI EVENTI PRINCIPALE</b>	<ol style="list-style-type: none"> <li>1 Il caso d'uso inizia quando l'amministratore clicca sul pulsante “Add class”.</li> <li>2 Il sistema mostra un form da compilare.</li> <li>3 L'admin inserisce il nome, la data, la difficoltà della classe, optionalmente una descrizione e tre categorie e fa l'upload del file della classe.</li> <li>4 Il sistema salva la classe.</li> <li>5 <i>Randoop</i> Robot genera i test relative a quella classe divisi in livelli.</li> <li>6 Il sistema calcola la copertura raggiunta e salva i risultati relativi a <i>Randoop</i>.</li> <li>7 <i>EvoSuite</i> Robot genera i test relative a quella classe divisi in livelli.</li> <li>8 Il sistema calcola la copertura raggiunta e salva i risultati relativi a <i>EvoSuite</i>.</li> <li>9 Il sistema mostra le classi inserite.</li> </ol>
<b>POST-CONDIZIONE</b>	La classe inserita è disponibile allo studente durante la creazione della partita. <i>Randoop</i> Robot e <i>EvoSuite</i> Robot sono selezionabili dallo studente durante la creazione della partita. Il punteggio è disponibile per l'elaborazione del vincitore alla conclusione della partita.

Figura 2.5: Add class Use Case

## Activity Diagram Add Class

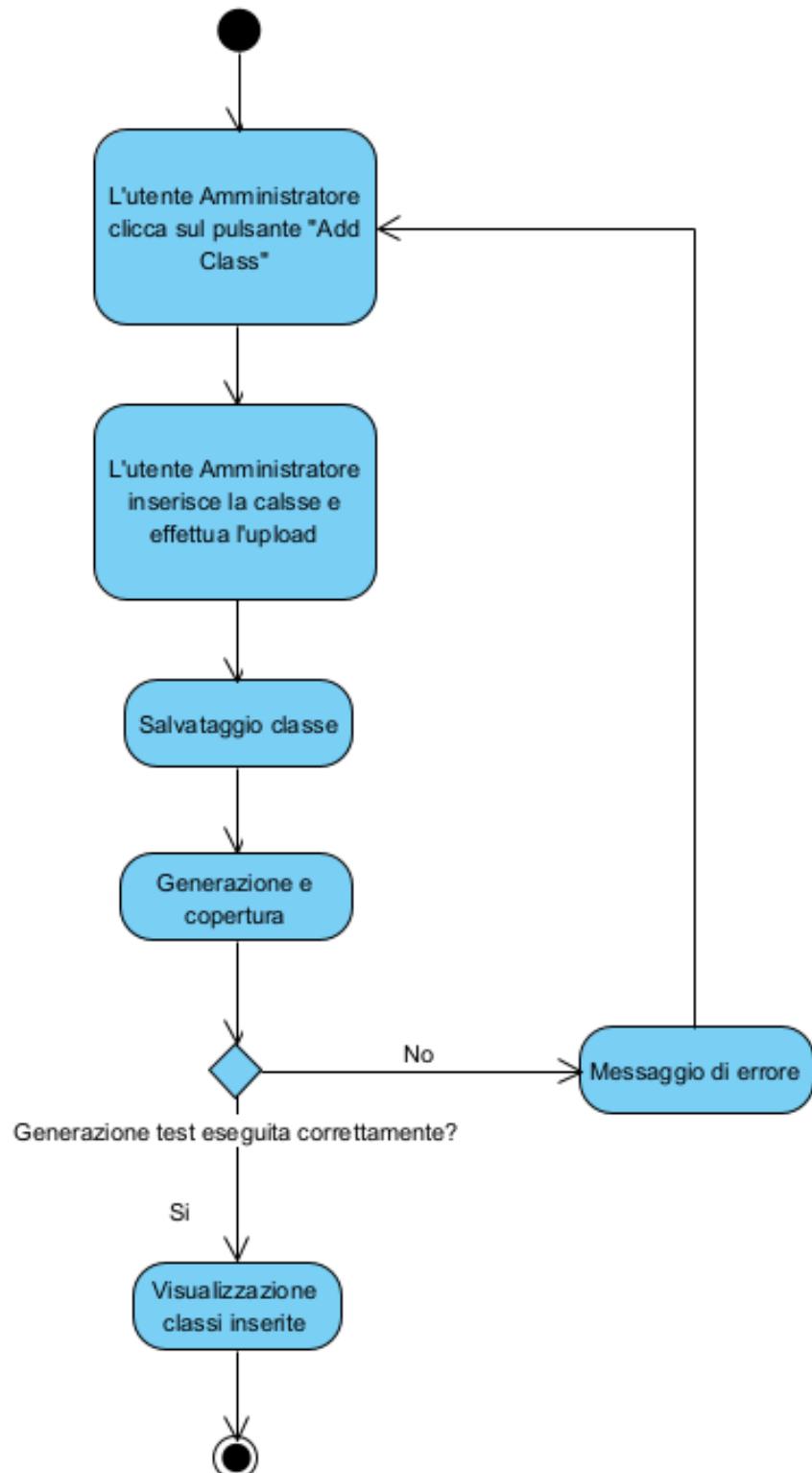


Figura 2.6: Activity Diagram Add class

### 2.3.3 Run

CASO D'USO	RUN
<b>ATTORE PRIMARIO</b>	Registered <u>Student</u>
<b>DESCRIZIONE</b>	Il giocatore esegue i test da lui scritti.
<b>PRE-CONDIZIONI</b>	Classe di testo compilata
<b>SEQUENZA DI EVENTI</b>	Il giocatore clicca sul bottone “ <u>Run</u> ”
<b>PRINCIPALE</b>	
<b>POST-CONDIZIONE</b>	Viene mostrato sulla console la percentuale di codice coperto da parte dello studente sulla classe UT e vengono colorate le linee di codice col colore appropriato in relazione alla copertura effettuata (verde/giallo/rosso).

Figura 2.7: Run Use Case

Il tasto **RUN** attuale è stato sostituito al vecchio ”**Run With JaCoCo**”; nella versione precedente era eseguito **JaCoCo** per la copertura delle linee di codice e le colorava in relazione a quanto erano coperte (rosso/giallo/verde).

Nella nuova versione si integra il tool **EvoSuite** il quale fornisce la percentuale di copertura totale del test sulla classe.

## Activity Diagram Run

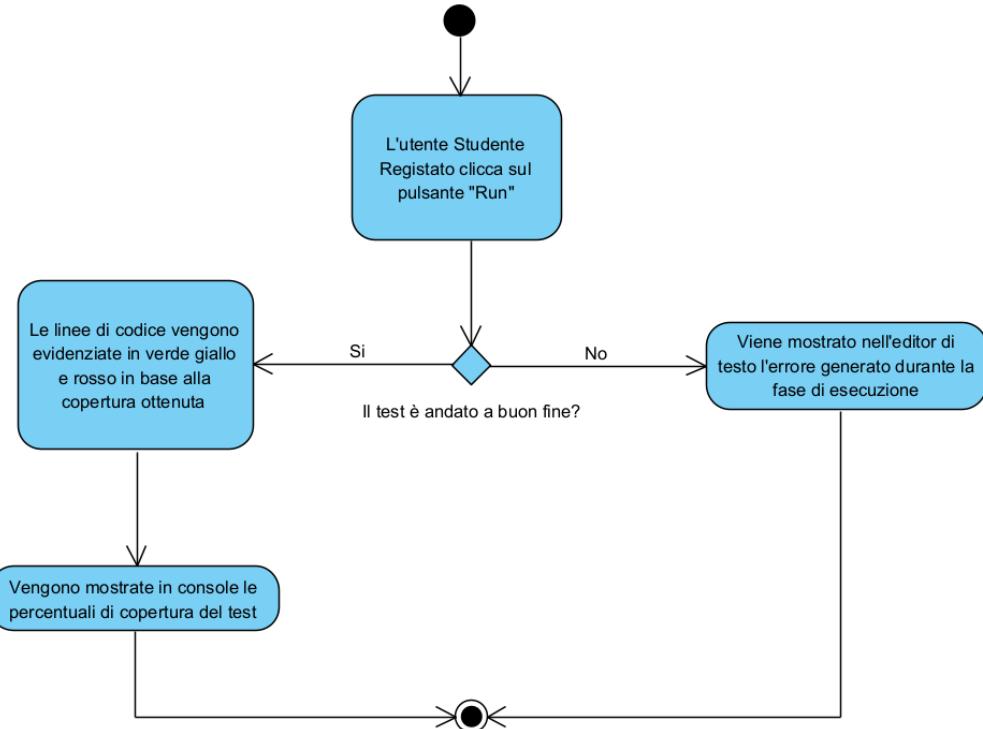


Figura 2.8: Activity Diagram Run

A seguito del lavoro di integrazione è possibile, dopo la visualizzazione delle linee colorate, visualizzare in **Console** le percentuali di copertura del test calcolate attraverso Evosuite.

## 2.3.4 Submit

Il caso d'uso Run in questo progetto è diventato **Submit**.

In precedenza, alla pressione del pulsante erano eseguiti i test scritti dell'utente ed in console era mostrata la percentuale di copertura del codice.

Dopo le modifiche, Submit sottopone il test sulla classe scritto dall'utente al calcolo della copertura, prende i risultati ottenuti e li confronta con quelli del robot mostrando in Console l'esito di compilazione e in "Confronto risultati" se è stata raggiunta la vittoria o meno.

CASO D'USO	SUBMIT
<b>ATTORE PRIMARIO</b>	Registered <u>Student</u>
<b>DESCRIZIONE</b>	Si eseguono i test e vengono confrontati i risultati con il robot scelto.
<b>PRE-CONDIZIONI</b>	Classe di test correttamente compilata.
<b>SEQUENZA DI EVENTI</b>	Il giocatore preme sul bottone " <b>Submit</b> "
<b>PRINCIPALE</b>	
<b>POST-CONDIZIONE</b>	Visualizzazione dei risultati ottenuti da parte dello studente all'interno della console e confronto con quelli del robot.

Figura 2.9: Submit Use Case

## Activity Diagram Submit

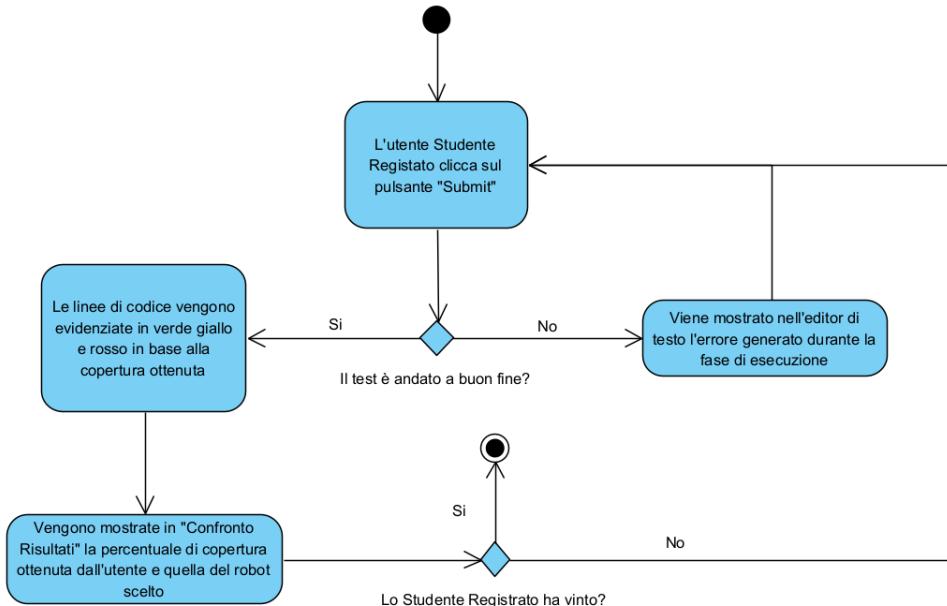


Figura 2.10: Activity Diagram Submit

Precedentemente il giocatore poteva effettuare un solo Submit per Game. Successivamente alle modifiche invece **l'utente può continuare a sottoporre nuovi test per la classe scelta** fin quando non effettua un Logout o fin quando la percentuale di copertura del proprio test non è superiore a quella del robot scelto.

In tal caso l'utente vince e non potrà più sottoporre test per quella classe.



Figura 2.11: Icona di Submit

## 2.3.5 Logout

E' stata inserita come funzionalità aggiuntiva la possibilità di effettuare un **Logout** in due schermate diverse:

- L'interfaccia di Editor
- L'interfaccia di selezione della classe e del robot da battere

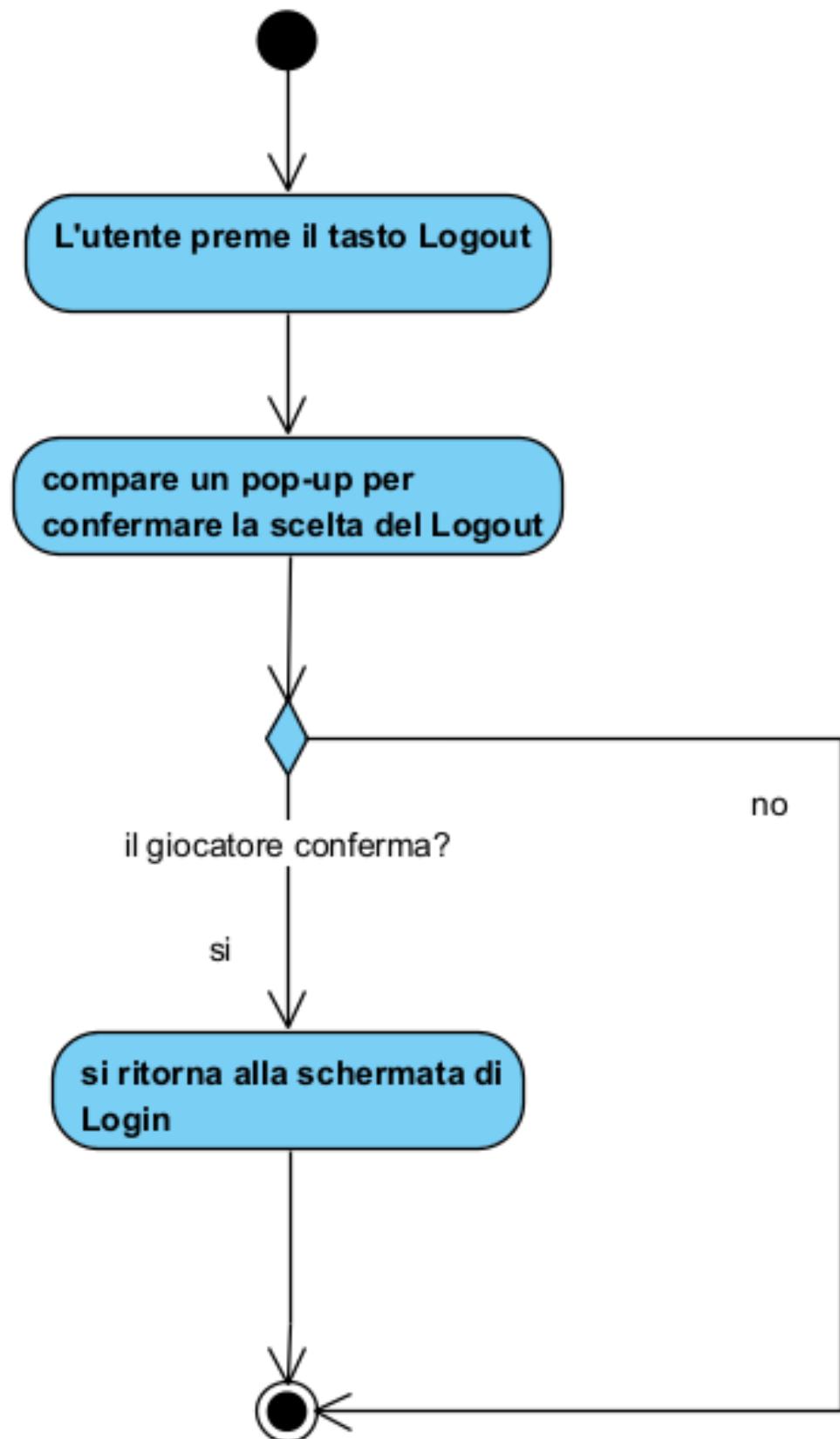
CASO D'USO	LOGOUT
<b>ATTORE PRIMARIO</b>	Registered <u>Student</u>
<b>DESCRIZIONE</b>	Il giocatore vuole disconnettersi e si ritorna all'interfaccia di login.
<b>PRE-CONDIZIONI</b>	Il giocatore ha effettuato il login.
<b>SEQUENZA DI EVENTI</b>	Il giocatore preme sul bottone “Logout”
<b>PRINCIPALE</b>	
<b>POST-CONDIZIONE</b>	I <u>coockies</u> di login vengono ripristinati per non conservare credenziali dell'utente.

Figura 2.12: Logout Icona



Figura 2.13: Logout Use Case

## Activity Diagram Logout



In entrambi i casi una volta premuto il tasto di **Logout**, compare un pop-up di conferma della scelta e nel caso di risposta affermativa si viene reindirizzati alla schermata di Login.

### 2.3.6 Storico

A seguito del lavoro di integrazione, è stato inserito il pulsante di **Storico** che permette di visualizzare in **Console** i tentativi di gioco nella singola partita sulla singola classe da testare.

In particolare per ogni tentativo 9.2 è visualizzato il punteggio da battere, il numero del tentativo, l'esito (Vittoria o Sconfitta), la percentuale di copertura delle linee di codice e infine il codice di test scritto.

CASO D'USO	HISTORY
<b>ATTORE PRIMARIO</b>	Registered Student
<b>DESCRIZIONE</b>	Visualizzazione in console dello storico dei codici di test scritti dall'utente coi relativi esiti di vittoria.
<b>PRE-CONDIZIONI</b>	Il giocatore si trova nell'editor di testo.
<b>SEQUENZA DI EVENTI</b>	Il giocatore preme sul bottone “History”
<b>PRINCIPALE</b>	
<b>POST-CONDIZIONE</b>	In console viene visualizzato lo storico dei turni submitati con il relativo esito e codice di test.
<b>SEQUENZA DI EVENTI</b>	Se il giocatore non ha ancora effettuato nessun submit dei test della classe allora in console viene mostrato un messaggio di avvertimento.
<b>ALTERNATIVI</b>	

Figura 2.15: History Use Case



Figura 2.16: Storico Icona

### Activity Diagram Storico

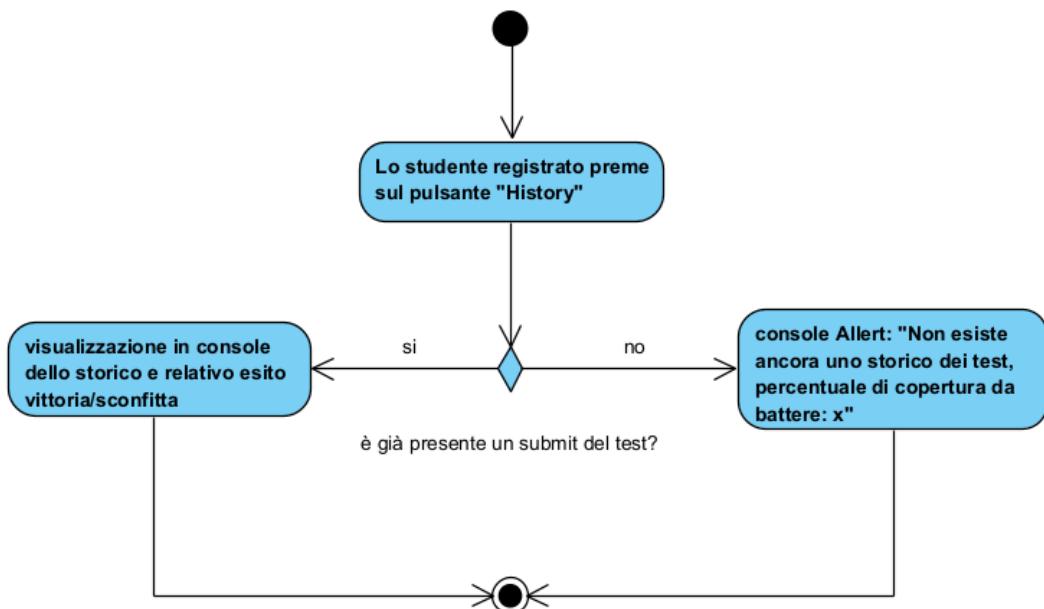


Figura 2.17: Activity Diagram Storico

Una volta che lo studente registrato clicca sul pulsante di Storico, se non è mai stato effettuato il Submit della classe, verrà mostrato un **Alert** in cui si avverte il giocatore che non esiste ancora uno storico dei test. Una volta che lo studente registrato clicca sul pulsante di Storico, se non è mai stato effettuato il Submit della classe, verrà mostrato un **Alert** in cui si avverte il giocatore che non esiste ancora uno storico dei test.

Verrà però mostrata la percentuale di copertura da superare per ottenere una vittoria.

### 2.3.7 Miglioramenti

Nella versione precedente, gli ‘ex’ pulsanti “Run” e “J” dell’interfaccia di Editor servivano rispettivamente per eseguire i test e confrontare i risultati con quelli del robot scelto e per eseguire la copertura delle linee di codice usando JaCoCo.

Si è deciso di invertirne il funzionamento per renderlo di più facile intuizione.

Quindi il tasto che si chiamava “Run with JaCoCo” ora si chiama “Run” e si occupa della copertura delle linee di codice usando i tool sia di JaCoCo che di EvoSuite. Il tasto “Run” è invece stato rinominato “Submit” e chiama il compilatore per il codice scritto dall’utente, calcola la copertura ottenuta sul proprio test e confronta i risultati con quelli ottenuti dal robot avversario.

# Capitolo 3

# Progettazione

## 3.1 Architettura a microservizi

L’architettura a microservizi dell’applicazione resta la stessa. Tuttavia è stato aggiunto il microservizio fornito dal Task 8.

Il client può accedere ai servizi forniti dall’applicazione tramite l’interfaccia data dai due gateway mediante la connessione Internet. L’**UI Gateway** ha il compito di effettuare il routing delle richieste http relative al frontend (UI).

L’**API Gateway** si occupa del routing e della gestione dell’autenticazione e autorizzazione per le richieste relative alle API del sistema.

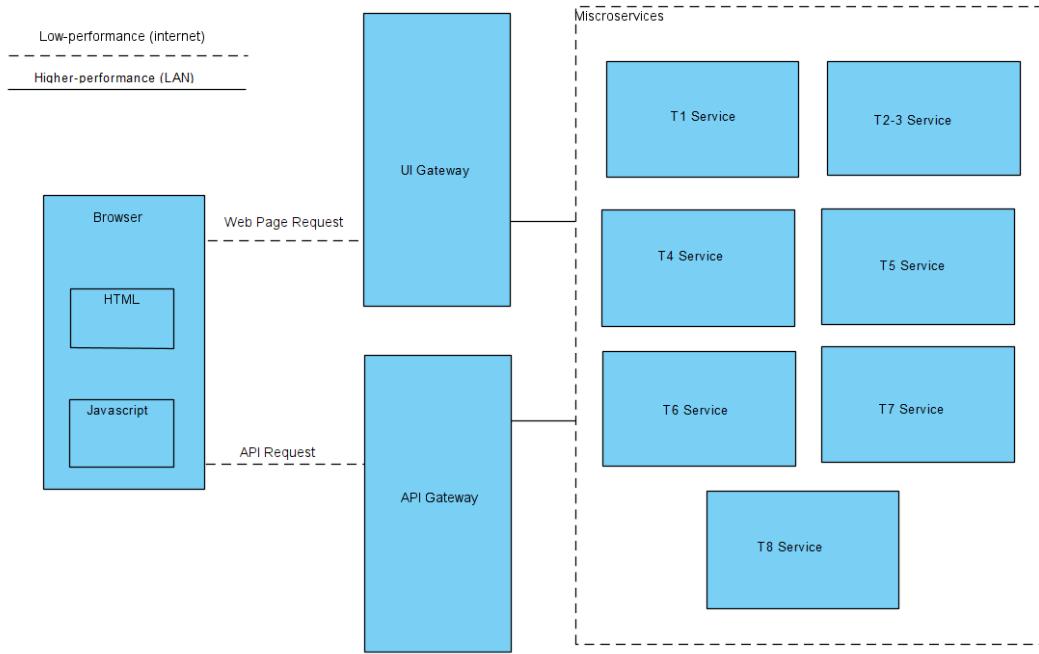


Figura 3.1: Architettura a microservizi

## 3.2 Component Diagram

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

A seguito del lavoro di integrazione è stata aggiunto un nuovo componente, **EvoSuite Runner** che:

- Genera test data una classe con il robot EvoSuite e ne calcola la copertura.

Questo è rappresentato da una relazione di tipo **usage** del componente EvoSuite Runner da parte del **Test Class Manager**

- Calcola la copertura del test scritto dal giocatore.

**EvoSuite Runner** è usato perciò da **Student Front End** all’atto di richiesta HTTP POST da parte dello studente, quando vuole compilare ed eseguire casi di test sulla classe scelta.

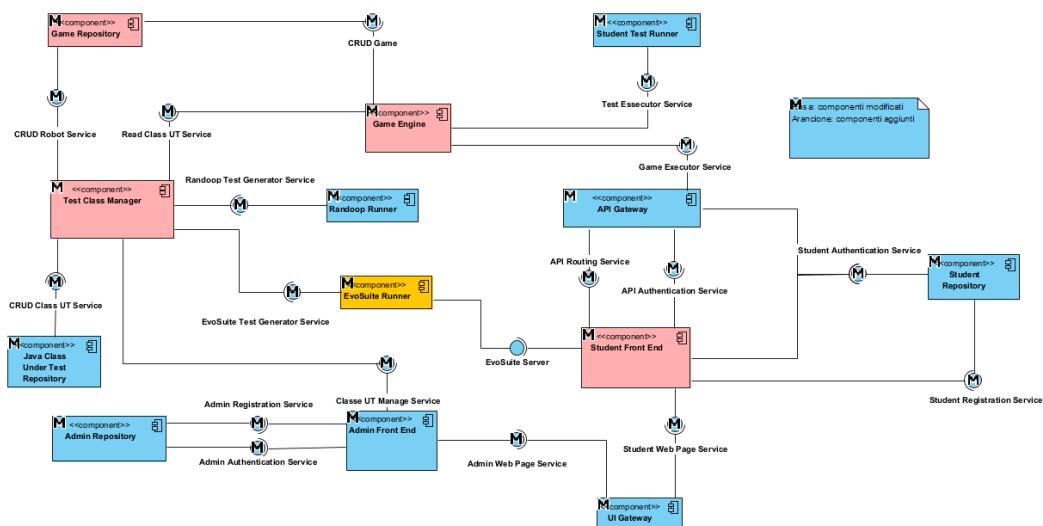


Figura 3.2: Component Diagram

**Legenda** utilizzata in figura per i componenti:

- Blu: componenti non modificati
- Arancione: componenti aggiunti
- Rosa: componenti modificati

### 3.2.1 Test Class Manager

Il **Test Class Manager** si occupa della gestione delle classi giocabili che sono inserite dall’admin.

Utilizza i servizi di copertura del codice offerti oltre che da Randoop, anche da EvoSuite.

Ad ogni classe inserita sono generati tutti i livelli possibili dei robot, prima attraverso Randoop e poi attraverso EvoSuite.

Pertanto questo componente ha subito delle modifiche in questo lavoro di integrazione.

### 3.2.2 Student Front End

Fornisce un’interfaccia all’utente “Registered Student”, a partire dalla registrazione fino alle schermate di gioco che comprendono la scelta del robot contro il quale giocare e l’interfaccia di editor.

Sono state inserite nuove funzionalità pertanto è stato modificato il componente (rosa).

Inoltre, è stata creata una nuova relazione di tipo usage verso EvoSuite Runner perché si vuole permettere al giocatore la compilazione con EvoSuite al fine di ottenere la percentuale di copertura del test.

### 3.2.3 EvoSuite Runner

Come detto in precedenza, è stato creato il component **Evosuite Runner**.

Esso genera i robot EvoSuite e restituisce le statistiche di copertura del codice. Comunica con il componente Test Class Manager per generare la copertura tramite Evosuite, all’atto del caricamento della

classe da parte dell'Admin.

E' usato dallo Student Front End per calcolare la copertura del codice del test scritto dallo studente.

### 3.2.4 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. Il Game Engine è stato modificato perché è stata introdotta una **nuova logica di gioco**: ogni Game offre la possibilità di molteplici turni fin quando non viene superata la percentuale di copertura del test scritto dal robot.

### 3.2.5 Game Repository

Ogni Submit viene salvato in un repository interno al T4, dedicato al salvataggio delle partite.

Ciò permette di avere uno storico dei risultati delle coperture dei test, ottenute nei diversi turni.

### 3.2.6 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.

E' stato modificato il componente per renderlo compatibile con la generazione dei Robot effettuata da Evosuite.

### 3.3 Composite Structure Diagram

Per mantenere uniformità con il diagramma dei componenti, nel Composite Structure Diagram è stata adottata una notazione che prevede l'utilizzo di un package per ciascuna componente, evidenziando chiaramente quali task sono responsabili per la loro realizzazione e contribuiscono al corretto funzionamento del sistema.

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.).

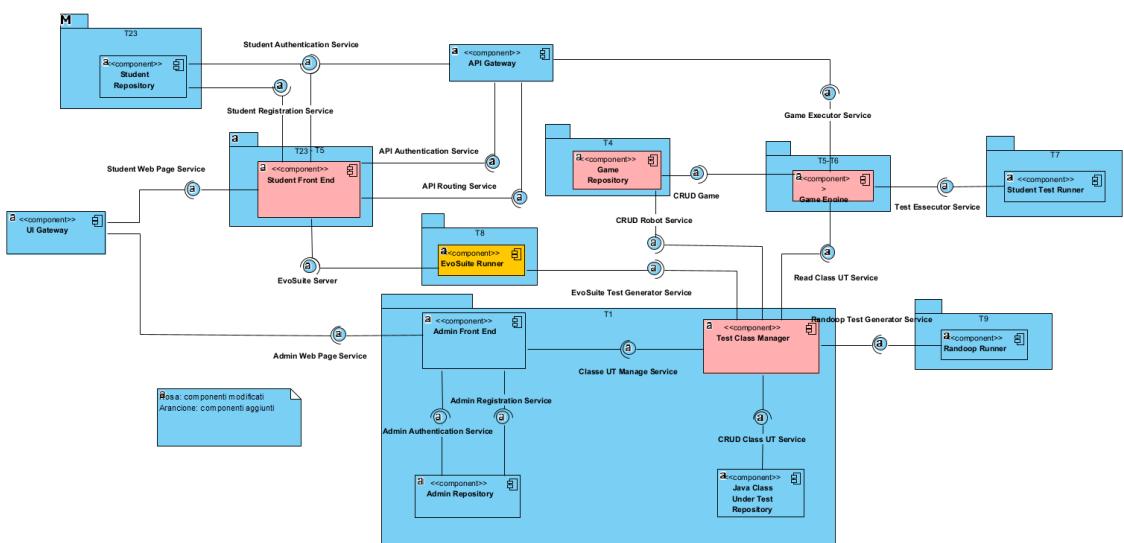


Figura 3.3: Composite Diagram

# Capitolo 4

## Scrum e Iterazioni

Scrum è un framework Agile che si concentra sulla gestione dei progetti e lo sviluppo del software. È basato su un set di principi e valori definiti nel "Manifesto Agile" e offre una struttura per aiutare i team a lavorare in modo collaborativo.

### 4.1 Introduzione a Scrum

Nel contesto del progetto, è stato adottato il framework Scrum come metodologia di gestione e sviluppo del task di integrazione e di nuove funzionalità.

In particolare il progetto è stato sviluppato in due **iterazioni**, ognuna delle quali è formata da 4 **Sprint** più uno EXTRA per la durata complessiva di circa 2 mesi di lavoro.

L'inizio di ogni iterazione ha previsto un'**Iteration Planning** volto

dapprima a creare le User Stories, e poi a comprendere quali di esse implementare nell’iterazione corrente.

In ogni iterazione sono stati svolti 4 **Sprint Planning** all’inizio di ogni nuovo sprint, in cui si sono suddivise le stories da implementare in **task** più piccoli assegnati ai membri del team.

Si è tenuta traccia dei task da implementare, quelli già implementati e quelli futuri tramite piattaforme quali Trello (9.5) e Miro (9.7).

## 4.2 User Stories totali

Le storie utente descrivono parti di funzionalità desiderate dal punto di vista dell’attore coinvolto, che riescono a catturare i ruoli, le attività e gli obiettivi dell’utente, fornendo una guida per lo sviluppo del processo agile.

Delle user stories ottimali seguono il paradigma **INVEST** ovvero:

- **Indipendenti:** quando le user stories non sono indipendenti tra di loro possono produrre stime non corrette, vincoli di implementazione e potenziali rigidità nella pianificazione. Nel caso in cui non sia possibile scrivere user stories indipendenti è meglio definire un ordine di implementazione;
- **Negoziabili:** le storie sono brevi descrizioni delle funzionalità desiderate, catturano l’essenza di una richiesta e non le speci-

fiche di dettaglio. I dettagli vengono quindi negoziati perché emergono dai dialoghi tra il cliente, il product owner e il team;

- **Di valore:** in un mondo utopistico tutte le storie sono di valore per l'utente finale, nella realtà è buon uso interrogarsi sul vero beneficiario della user story. L'importante è evitare di perdere tempo (e soldi) in storie di valore per nessuno;
- **Stimabili:** dato che SCRUM prevede iterazioni autoconclusive, la stima di una storia e del tempo necessario per produrla è un'attività fondamentale del team. Nel caso in cui gli sviluppatori non conoscano bene il dominio applicativo o non abbiano una conoscenza totale della tecnologia da utilizzare si può iniziare a programmare un'attività di analisi da parte del team (una *spike*) per arrivare a una stima dell'ordine di grandezza delle attività. Nel caso in cui le stories risultassero poco chiare o troppo grandi è compito del Product Owner farsi carico della segmentazione di esse e dettagliare maggiormente i criteri di accettazione;
- **Della giusta dimensione:** le storie non devono essere né troppo grandi né troppo piccole. In questi casi sarà opportuno granulare le storie troppo grandi o combinare più storie piccole;
- **Testabili:** Al termine di ogni *sprint* vengono rilasciate delle funzionalità che sono *done*. Per questo motivo nelle user sto-

ries non possono mancare i criteri di accettazione misurabili attraverso test.

Le User Stories seguono il concetto delle **3C**, ovvero *Card*, *Conversation*, *Confirmation*:

- **Card (Carta)**: La user story è rappresentata da una "carta" che fornisce una breve descrizione del comportamento desiderato del sistema.
- **Conversation (Conversazione)**: La conversazione si riferisce alle discussioni dettagliate che avvengono tra gli stakeholder e i membri del team per chiarire e approfondire la comprensione della user story.
- **Confirmation (Conferma)**: La conferma definisce i criteri di accettazione che indicano quando la user story è completata correttamente.

Questi criteri sono fondamentali per assicurare che le storie utente vengano gestite facilmente, comprese e quindi testate all'interno del processo di sviluppo.

La lista delle User Stories finali ideate e implementate in questo progetto è mostrata in figura 2.1 riportata nel Capitolo 2.

### 4.2.1 Criteri di accettazione

In *SCRUM* i criteri vengono scritti usando il modello "**Given-When-Then**", ovvero una struttura che definisce le condizioni iniziali, l'azione o l'evento scatenante, il risultato desiderato e, nel caso, ulteriori condizioni (**AND**). Ciò assicura che il lavoro del team sia allineato alle aspettative dell'utente.

Questo metodo di lavoro è stato necessario per avere una maggiore chiarezza nella definizione delle funzionalità, consentendo di stabilire cosa si aspetta l'utente da una determinata funzionalità o requisito. Inoltre, tali criteri forniscono metriche per valutare il successo o il fallimento di una funzionalità aiutando a tracciare lo stato di avanzamento del progetto.

In conclusione, ogni criterio fornisce delle condizioni che devono essere soddisfatte per ritenere una storia utente completata con successo, aiutando a evitare fraintendimenti e ambiguità.

Di seguito sono riportati i criteri di accettazione per ogni storia creata.

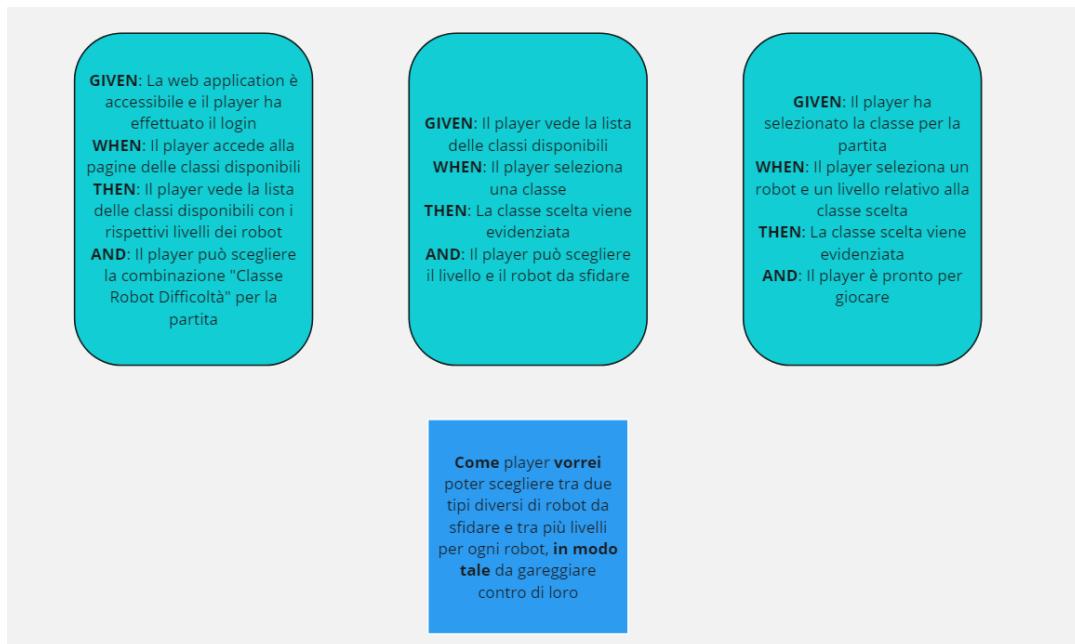


Figura 4.1: Criterio di accettazione 1

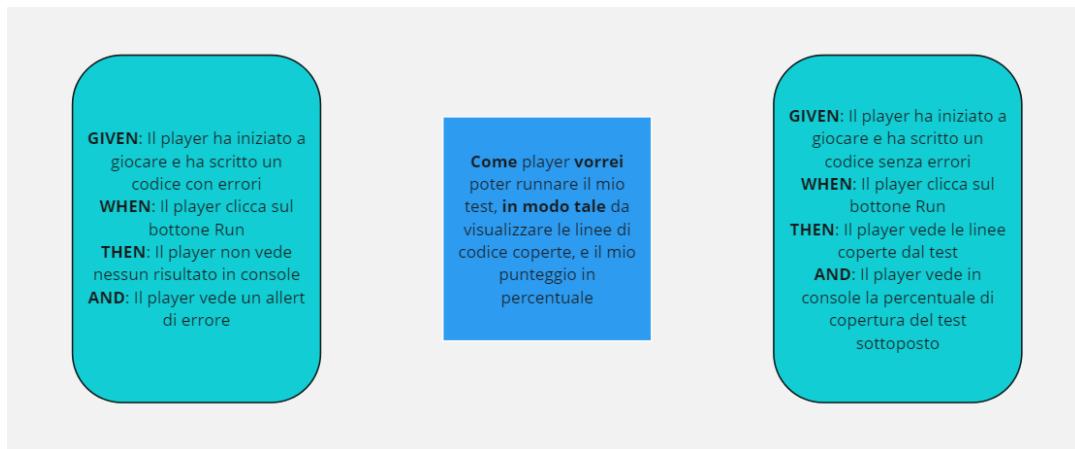


Figura 4.2: Criterio di accettazione 2

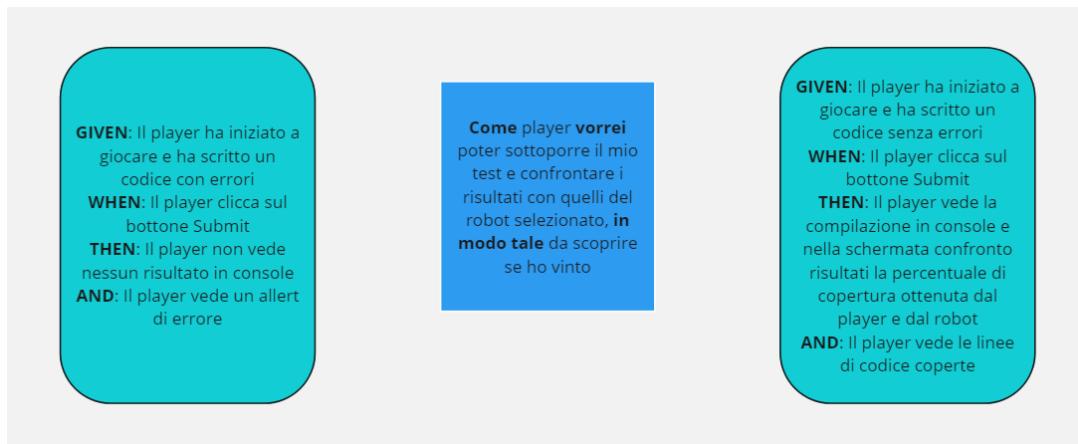


Figura 4.3: Criterio di accettazione 3

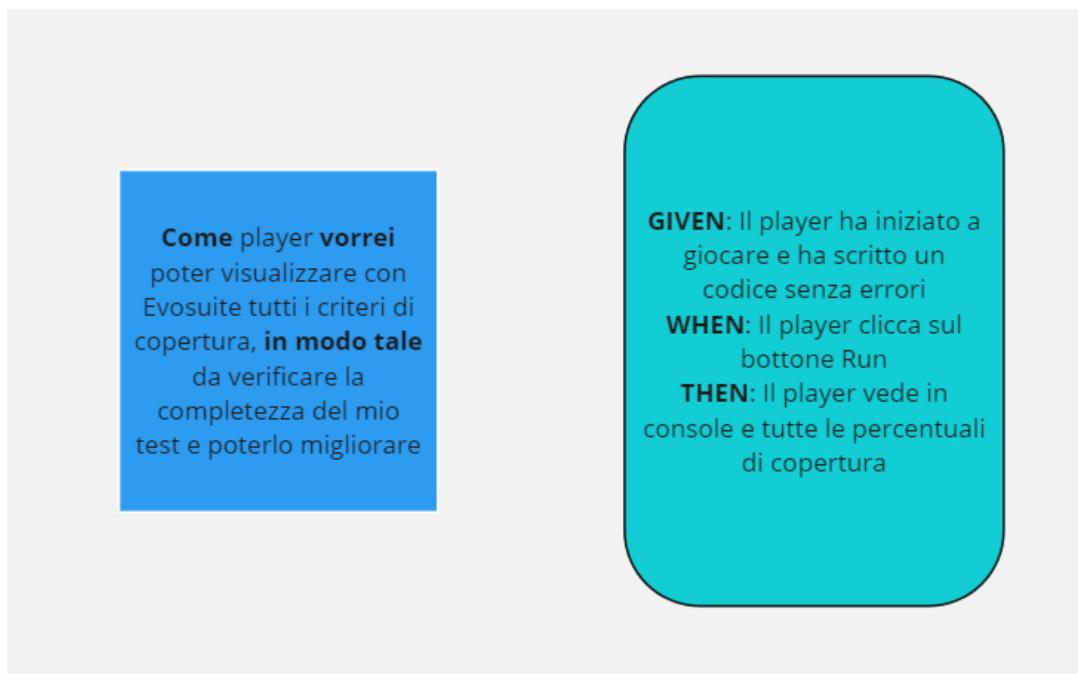


Figura 4.4: Criterio di accettazione 4

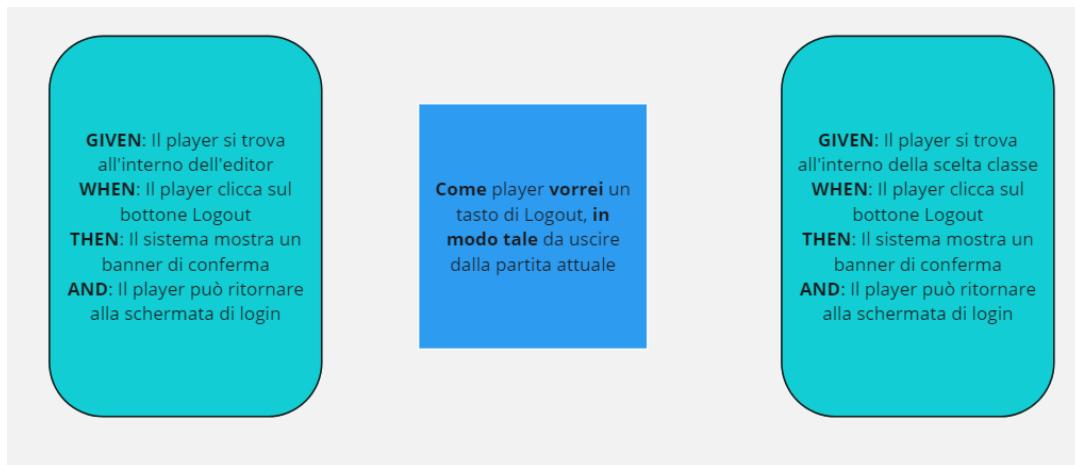


Figura 4.5: Criterio di accettazione 5

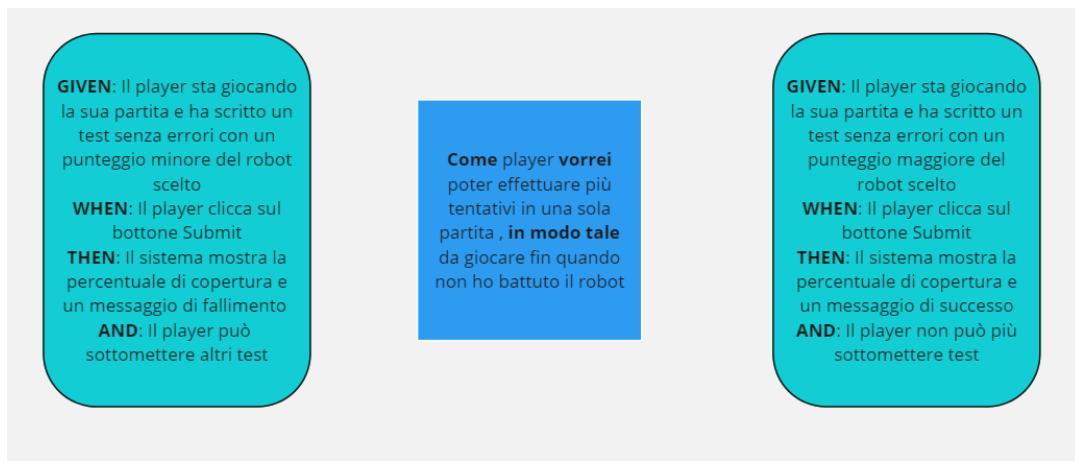


Figura 4.6: Criterio di accettazione 6



Figura 4.7: Criterio di accettazione 7

## 4.3 Prima iterazione

Per l'implementazione di questa applicazione, la prima iterazione è ufficialmente iniziata con una dettagliata riunione di iteration planning, durante la quale sono stati definiti gli obiettivi e le priorità per questa fase iniziale di sviluppo.

### 4.3.1 Iteration Planning iterazione 1

L'iteration Planning ha portato all'individuazione di un **backlog** in cui sono evidenziate le User stories da implementare nella prima iterazione.

#### User Stories

Nella figura di seguito sono mostrate le storie create nel primo Iteration Planning.



Figura 4.8: Stories Iterazione 1

N.B: Alla fine della prima iterazione, la storia effettivamente implementata è stata solo la prima; l'implementazione delle restanti tre è stata rimandata alla seconda iterazione.

### 4.3.2 Primo Sprint

Successivamente all'ideazione delle User stories e assegnatene le priorità, si è deciso quali di esse si sarebbero implementate nello sprint corrente.

In realtà il primo Sprint è stato dedicato allo studio delle documentazioni dei progetti e dei singoli task.

La figura 4.9 mostra infatti l'utilizzo del Tool **Miro** per lo Sprint Planning in cui è stata fatta la suddivisione in singoli Task e per tenere traccia degli **obiettivi raggiunti** o dei **problemi emersi** in ogni sprint.

N.B: Le problematiche emerse in ogni Sprint sono state esplicitate nel capitolo 5, ed è stato riportato nel dettaglio anche il modo in cui sono state superate e risolte.

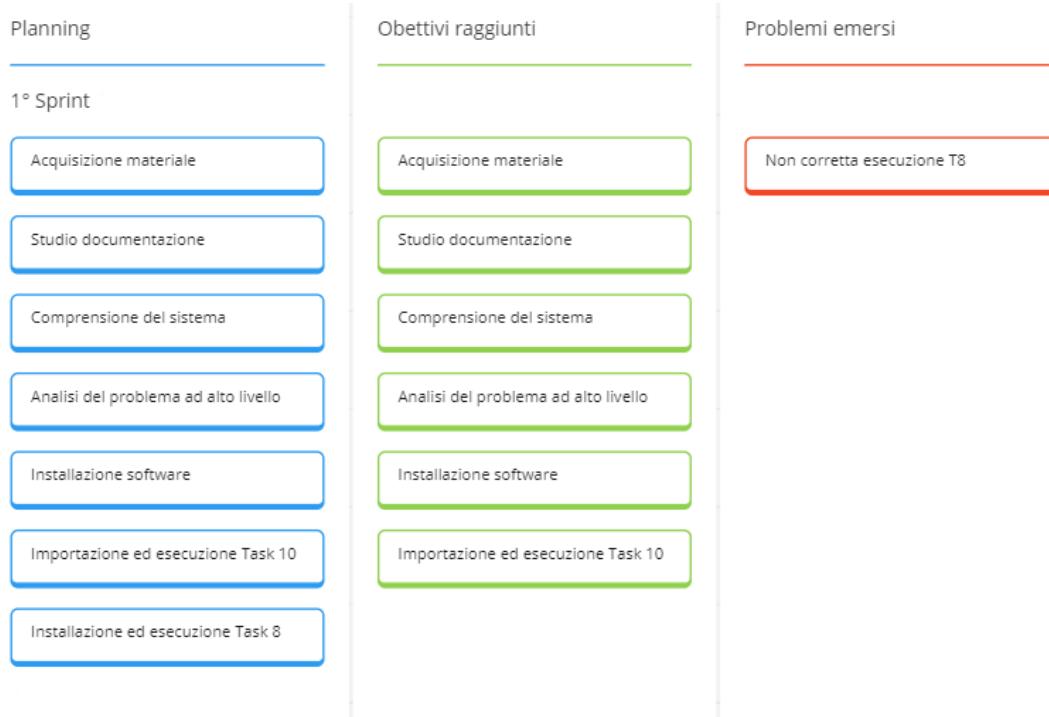


Figura 4.9: Iterazione 1 Sprint 1

Come si può osservare in figura, nel primo sprint ci si è concentrati sul:

- L'acquisizione del materiale
- Lo studio della documentazione
- La comprensione dell'intera Web App
- L'analisi del problema ad alto livello
- L'intallazione del software
- L'importazione ed esecuzione del Task 10 cioè quello sull'integrazione

### 4.3.3 Secondo Sprint

Successivamente ad uno sprint planning, si è arrivati all'assegnazione dei task in figura per il secondo sprint.

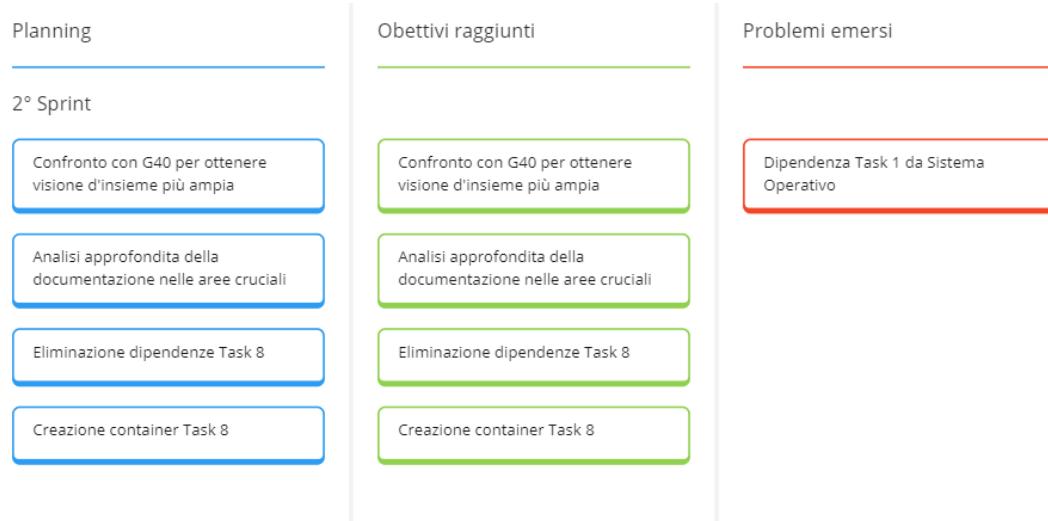


Figura 4.10: Iterazione 1 Sprint 2

E' stato pertanto:

- studiato il lavoro di integrazione del gruppo 40
- analizzato la documentazione
- eliminate le dipendenze del task 8 (si veda il paragrafo 5.2.1)
- creato il container del task 8 (5.2.2)

Le problematiche riscontrate in questo sprint hanno riguardato il lavoro di integrazione del task 8 per renderlo compatibile con il Task 1.

Tutte le modifiche al Task 1 effettuate per rendere possibile questa integrazione sono state svolte nello Sprint successivo.

#### 4.3.4 Terzo Sprint

Nel terzo Sprint è stato effettivamente adattato il Task 8 agli altri Task.

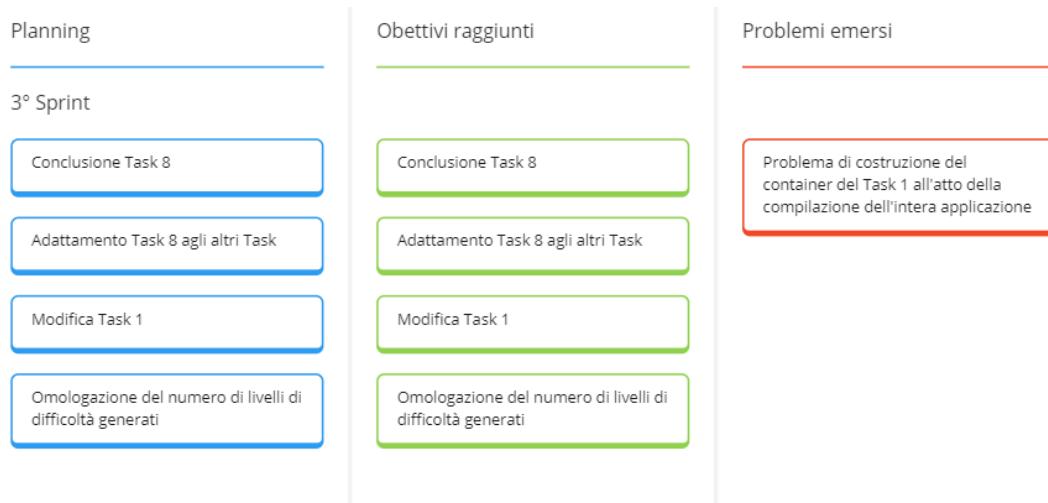


Figura 4.11: Iterazione 1 Sprint 3

In particolare questo è stato fatto:

- adattando il Task 8 al task 1 e agli altri task (5.3.1)
- modificando il Task 1 (5.3.2) per aggiungere la generazione dei test con Evosuite

Nell'eseguire queste modifiche sono state riscontrate tre problematiche principali: Una sul salvataggio dei file di statistiche e robot generati, un'altra sulla generazione del numero dei livelli di difficoltà e l'ultima sulla compilazione del task 1. Le prime due sono state risolte nello sprint corrente (si rimanda al capitolo 5 per approfondimenti) mentre la terza nello sprint successivo.

### 4.3.5 Quarto Sprint



Figura 4.12: Iterazione 1 Sprint 4

Nel quarto sprint si è provveduto alla risoluzione del problema di compilazione del Task 1 ed inoltre alle:

- modifiche del Task 5 per il Backend
- modifiche del Task 5 per il Frontend

Questo per poter visualizzare i robot generati da Evosuite.

Alla fine del Quarto Sprint è stata dunque portata a termine la user story: **”Come Player vorrei** poter scegliere tra due diversi tipi di robot da sfidare e tra più livelli per ogni robot, **in modo tale** da gareggiare contro di loro”

## 4.4 Seconda iterazione

La seconda iterazione, come di consueto è iniziata con un'Iteration Planning.

### Iteration Planning - User Stories

Si possono osservare in figura 4.13 le stories che sono state create in questo Planning, e implementate nella Seconda Iterazione.



Figura 4.13: Stories Iterazione 2

N.B: la disparità nel numero di stories e di requisiti del sistema portate a termine tra le due iterazioni risiede esclusivamente nel fatto che si deve prendere in considerazione il delay iniziale per lo studio

dell'intera architettura, e per tutte le problematiche riscontrate nel primo approccio ad essa.

#### 4.4.1 Primo Sprint

Il primo Sprint della Seconda Iterazione inizia con lo scopo di implementare la seconda user story.

Di conseguenza il goal da raggiungere era quello di **stampare in console la percentuale di copertura fornita da Evosuite**.

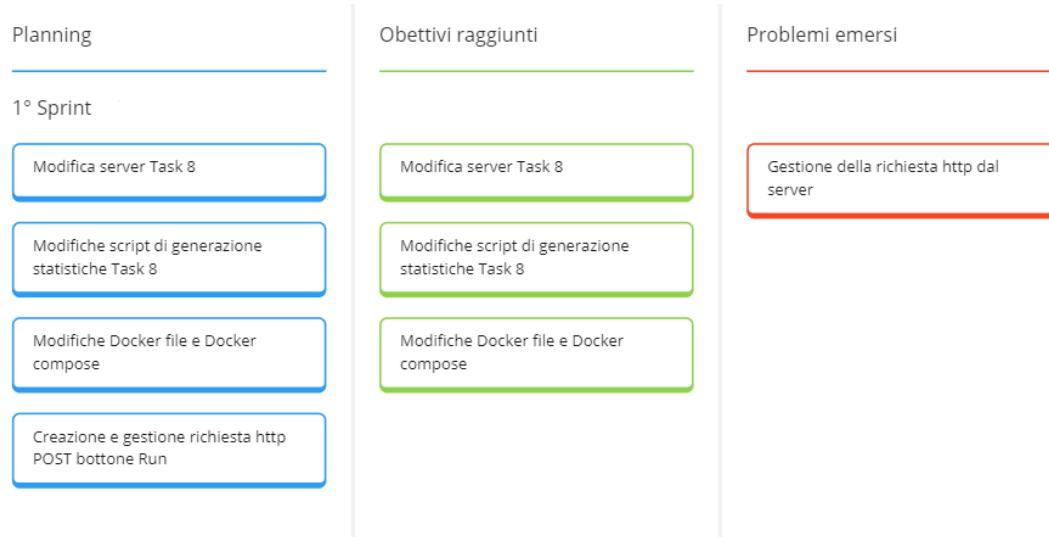


Figura 4.14: Iterazione 2 Sprint 1

I task eseguiti in questo sprint sono dunque quelli di:

- modifica del server del Task 8 (5.5.1)
- modifica dello script per la generazione delle statistiche (5.5.2)
- modifica dei Docker file e Docker compose del Task 8 (5.5.3)

A causa della problematica riscontrata sulla gestione delle richieste http nel server, il goal non è stato raggiunto in questo sprint ma in quello successivo.

#### 4.4.2 Secondo Sprint



Figura 4.15: Iterazione 2 Sprint 2

Nel secondo sprint della seconda iterazione i task portati a termine sono stati:

- La gestione della richiesta http dal server
- La creazione e gestione della richiesta http POST del bottone di RUN (5.6.1)
- L'adattamento delle dipendenze del Task 7 dalla versione di JUnit4 (5.6.2)
- L'adattamento dei buttoni di Run e Submit (5.6.3)

Con l'implementazione dei primi due task si è portato a termine il goal sulla visualizzazione in Console della percentuale di copertura fornita da Evosuite alla pressione del tasto di Run.

Con i secondi due invece si può ritenere conclusa anche la prima User story scritta nell'iteration backlog:

**"Come Player, vorrei poter runnare il mio test, in modo tale da visualizzare le linee di codice coperte, e il mio punteggio in percentuale."**

#### 4.4.3 Terzo Sprint

Nello sprint planning avvenuto all'inizio del terzo sprint, si è deciso di **completare il lavoro di integrazione e introdurre una nuova funzionalità**: la possibilità di introdurre un Logout nelle due schermate di scelta della classe e di editor.



Figura 4.16: Iterazione 2 Sprint 3

Nel terzo sprint sono stati quindi portati a termine questi 3 Task:

- Creazione e gestione richiesta http POST bottone Submit (5.7.1)
- Creazione "Logout" in interfaccia di Editor (5.7.2)
- Creazione "Logout" in interfaccia di scelta della classe da testare e robot da battere (5.7.3)

Con l'implementazione del primo task è stato portato a termine anche il requisito equivalente alla user story:

**"Come Player vorrei** poter sottoporre il mio test e confrontare i risultati con quelli del robot selezionato, **in modo tale** da scoprire se ho vinto".

Si è pertanto concluso anche il vero e proprio lavoro di integrazione. Il secondo e il terzo Task invece riguardano la logica del logout che, una volta implementata, ha permesso di concludere anche la user story:

**"Come Player vorrei** un tasto di Logout **in modo tale** da uscire dalla partita attuale"

#### 4.4.4 Quarto Sprint



Figura 4.17: Iterazione 2 Sprint 4

Nel Quarto Sprint, successivamente ad uno **Sprint Planning**, si è deciso di implementare anche nuove funzionalità:

1. La funzionalità di **Storico**, ovvero la possibilità di permettere all'utente di giocare più turni in una sola partita
2. La visualizzazione di informazioni aggiuntive sulla copertura del codice fornite da Evosuite

Pertanto i task completati sono:

- la modifica della logica di fine partita (5.8.1)

- la creazione e gestione dello storico (5.8.3)
- il salvataggio del valore di copertura in locale al Task 4 (5.8.4)
- la visualizzazione delle informazioni aggiuntive di copertura (5.8.6)
- L'incremento della portabilità del sistema (5.8.7)
- pulizia del codice
- testing finale
- revisione della documentazione

Con il primo task si è implementata la user story: **”Come Player vorrei poter effettuare più tentativi in una sola partita, in modo tale da giocare fin quando non ho battuto il robot”.**

Con il secondo e il terzo task è stata invece ultimata la user story: **”Come Player vorrei poter visualizzare uno storico, in modo tale da verificare i miei progressi durante un game”.**

Con il quarto task invece è stata implementata la user story: **”Come Player vorrei poter visualizzare con Evosuite tutti i criteri di copertura, in modo tale da verificare la completezza del mio test e poterlo migliorare”.**

#### 4.4.5 Sprint EXTRA

Infine a causa del mancato accesso al canale di github, si è avuto il tempo di realizzare un’ulteriore funzionalità in quello che è stato chiamato lo ”Sprint EXTRA”.

E’ stata pertanto data la possibilità di **visualizzare il proprio test nello Storico** per ogni tentativo di gioco contro il robot.

E’ possibile approfondire l’implementazione di questo task nel paragrafo 5.9.1.

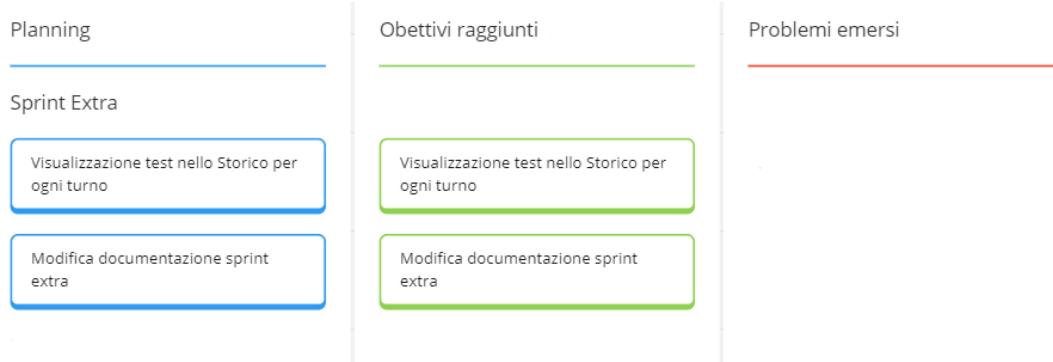


Figura 4.18: Iterazione 2 Sprint EXTRA

# **Capitolo 5**

## **Modifiche effettuate**

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

### **5.1 Iterazione 1 - Sprint 1**

Come già descritto nel Capitolo 4, il primo Sprint della prima iterazione si è basato esclusivamente sullo studio dell'architettura generale del progetto e della documentazione dei singoli task.

Questa trattativa pertanto si concentrerà sulla descrizione delle modifiche effettuate a partire dal secondo Sprint.

## 5.2 Iterazione 1 - Sprint 2

Una volta studiato il **TASK 8** attraverso la documentazione, l'installazione e il testing, è stato possibile iniziare a modificare il codice in quanto è stato notato fin da subito una presunta problematica per l'integrazione. Essa verrà descritta approfonditamente nella sezione seguente.

### 5.2.1 Eliminazione dipendenze Task8

L'implementazione del task 8 presupponeva delle **dipendenze dai package**.

Esso infatti aveva fornito delle classi di esempio "calcolatrice.java" e "calendario.java", contenute in un package.

Poichè il progetto Man Vs Automated Testing Tools challenges riguarda esclusivamente **test di unità**, la dipendenza da package era un possibile problema se si voleva andare a testare classi che non appartenevano a nessun package. Di conseguenza la prima modifica effettuata, come si evince dalla figura 5.1 è stata quella di modificare il file pom.xml utilizzato per la compilazione, eliminando le dipendenze dal package (NOME CLASSE al posto di NOME PACKAGE)

```

50 cat >>pom.txt << EOF
51 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
52   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
53   <modelVersion>4.0.0</modelVersion>
54
55   <groupId>$NOME_CLASSE</groupId>
56   <artifactId>$NOME_CLASSE</artifactId>
57   <version>1.0-SNAPSHOT</version>
58   <packaging>jar</packaging>
59   <name>$NOME_CLASSE</name>
60   <url>http://maven.apache.org</url>
61
62   <properties>
63     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
64   </properties>
65
66   <dependencies>
67     <dependency>
68       <groupId>junit</groupId>
69       <artifactId>junit</artifactId>
70       <version>4.13.1</version>
71       <scope>test</scope>
72     </dependency>
73   </dependencies>
74 </project>
75
76
77 EOF

```

Figura 5.1: Modifica file pom.xml

Successivamente è stato modificato anche il file **"robot\_generazione.sh"** del Task 8, presente nel path

/T11-G41-main/T8-G21/Progetto\_SAD\_GRUPPO21\_TASK8  
 /Progetto\_def/opt\_livelli  
 /Prototipo2.0/Prototipo2.0/robot\_generazione.sh apportando la stessa eliminazione delle classi dai package (fig. 5.2 )

```

67 echo "Test 1"
68 $EVOSUITE -class $NOME_CLASSE -projectCP target/classes
69
70 echo "facciamo partire il test"
71 mvn dependency:copy-dependencies
72 export CLASSPATH=target/classes:evosuite-standalone-runtime-1.0.6.jar:evosuite-tests:target/dependency/junit-4.13.1.jar:target/dependency
73 javac evosuite-tests/$NOME_CLASSE/*.java
74 java org.junit.runner.JUnitCore ${NOME_CLASSE}_ESTest
75 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=${NOME_CLASSE}_ESTest -projectCP $PERCORSO/target/classes/:evosuite-tests
76 mv evosuite-report IO
77 mv evosuite-tests IO
78 sleep 1
79
80
81
82
83
84
85
86
87
88

```

Figura 5.2: Modifica dipendenze file robot generazione.sh

N.B: Lo screen in figura è soltanto esemplificativo, in quanto le reali modifiche nel file sono su molte più righe di codice non riportate in figura, poichè il file "robot generazione.h" ha il compito di generare 13 livelli di robot, ognuno con una propria percentuale di copertura d linee di codice calcolata per quel test sulla classe. Le modifiche pertanto sono state svolte per ogni livello.

Successivamente è stato modificato anche il file **Misurazione livelli.sh**, lo script che si occupa di calcolare la percentuale di copertura con Evosuite dei test generati da Evosuite stesso, eliminando anche lì le dipendenze dai package (fig. 5.3 ).

```

24 echo "il file $filename è stato creato con successo."
25 ((cont++))
26 export CLASSPATH=target/classes:evosuite-standalone-runtime-1.0.6.jar:$var/evosuite-tests:target/dependency/junit-4.13.1.jar:target/dependency/hamcrest-core-1.3.jar
27 javac $var/evosuite-tests/$NOME_CLASSE/*.java
28
29 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=LINE
30 echo "coverage 1 finita" >> "$filename"
31 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=BRANCH
32 echo "coverage 2 finita" >> "$filename"
33 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=EXCEPTION
34 echo "coverage 3 finita" >> "$filename"
35 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=WEAKMUTATION
36 echo "coverage 4 finita" >> "$filename"
37 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=OUTPUT
38 echo "coverage 5 finita" >> "$filename"
39 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=METHOD
40 echo "coverage 6 finita" >> "$filename"
41 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=METHODNOEXCEPTION
42 echo "coverage 7 finita" >> "$filename"
43 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=$NOME_CLASSE_ESTest -projectCP $PERCORSO/target/classes/:$var/evosuite-tests -Dcriterion=CBRANCH
44 echo "coverage 8 finita" >> "$filename"

```

Figura 5.3: Eliminazione dipendenze file Misurazione Livelli.sh

## 5.2.2 Creazione container Task 8

Al fine di integrare il Task 8 nel progetto complessivo, è stata necessaria la creazione di un container nel Docker Desktop chiamato **T8\_generazione** che al principio non esisteva.

Inoltre avendo studiato la documentazione del gruppo 40, omologando le scelte di progetto, è stata aggiunta la generazione dei test e il

calcolo della copertura da parte dei robot Evosuite contestualmente all'inserimento della classe da parte dell'amministratore.

Per fare ciò si è resa necessaria la creazione del volume condiviso **"VolumeT8"** tra il Task 1 e il Task 8.

Di seguito è riportato (fig.5.4) lo script del **Docker file** in cui è descritto il sistema operativo utilizzato e il comando di copia del contenuto della cartella corrente nella cartella **"app"**.

Inoltre quando il container è avviato, verrà automaticamente eseguito lo script contenuto in **docker-entrypoint.sh**, che copierà il contenuto della cartella app in Volume T8.

```
FROM ubuntu:latest

COPY . /app

WORKDIR /app

RUN bash installazione.sh

ENTRYPOINT [ "bash", "docker-entrypoint.sh" ]
```

Figura 5.4: Creazione docker file T8

Nel file **Docker compose** (fig. 5.5 ) invece sono state eseguite queste azioni:

- Agganciato il container t8\_generazione alla rete global network

- Creato il volume T8
- Settati i volumi e la rete come external

```
version: '2'

services:
  t8_generazione:
    build: ./t8_generazione
    networks:
      - global-network
    volumes:
      - VolumeT8:/VolumeT8

networks:
  global-network:
    external: true

volumes:
  VolumeT8:
    external: true
```

Figura 5.5: Creazione docker compose

## 5.3 Iterazione 1 - Sprint 3

Successivamente alla creazione del Volume T8, è stato testato il corretto funzionamento delle modifiche effettuate. E' stato perciò runnato lo script di generazione dei robot contestualmente al Task 1.

### 5.3.1 Adattamento Task 8

Pertanto l'iter seguito è stato quello di:

- Caricare dall'interfaccia dell'amministratore una classe da testare
- L'applicazione ha prodotto i robot e i valori di copertura relativi a Randoop
- Runnare nella console del container 1 manualmente lo script "generazione\_robot.sh" del task 8 sulla classe salvata in T1.

Nel compiere l'ultimo passo, si è resa necessaria:

- La modifica del **Docker file del task 1**: all'interno del container del task 1 si è eliminata la dipendenza da Alpine Linux inserendo quella da Ubuntu per poter eseguire file '.sh' del task 8 (fig. 5.6 linea 1)
- L'installazione in ubuntu delle stesse librerie precedentemente installate in alpine (fig. 5.6 linee 7-8)

- La modifica successiva del file **”robot.sh” del TASK 9** in cui è stato cambiato il path per gli eseguibili di java e javac, in cui vi era la dipendenza dal sistema operativo Alpine.
- La modifica del file **Docker compose del task 1** come in figura 5.7 aggiungendo il Volume condiviso T8.

```
1  FROM ubuntu:latest
2  # FROM openjdk:8-alpine as java8
3
4  # FROM openjdk:17-alpine
5  # COPY --from=java8 /usr/lib/jvm/java-1.8-openjdk /usr/lib/jvm/java-1.8-openjdk
6
7  RUN apt-get update && apt-get install -y openjdk-8-jdk bash
8  RUN apt-get install -y openssl libncurses5 libstdc++6
9
10 # RUN apk update && apk add bash
11 # RUN apk add --no-cache openssl ncurses-libs libstdc++
12
13 COPY target/manvsclass-0.0.1-SNAPSHOT.jar /app/manvsclass.jar
14 COPY installazione.sh /app
15 COPY evosuite-1.0.6.jar /app
16 COPY evosuite-standalone-runtime-1.0.6.jar /app
17
18 WORKDIR /app
19
20 RUN bash installazione.sh
21
22 #WORKDIR /app
23
24 EXPOSE 8080
25 CMD ["java","-jar","manvsclass.jar"]
```

Figura 5.6: Modifica Docker File Task1

N.B: Si noti che in figura 5.6 è presente il run dello script ”installazione.sh” che permette l’installazione di Evosuite.

Esso è stato inserito manualmente nei file del task 1.

La scelta di progetto di eseguire questo run direttamente all’avvio del container T1 è stata dettata dall’impossibilità di runnarlo direttamente dal volume T8 in fase di installazione dei container.

```
1  version: '3.12.12'
2
3  services:
4    controller:
5      build: .
6      restart: always
7      expose:
8        - 8080
9      # ports:
10     #   - 8080:8080
11     depends_on:
12       - mongo_db
13     volumes:
14       - ./FilesUpload:/Files-Upload
15       - VolumeT9:/VolumeT9
16       - VolumeT8:/VolumeT8
17     networks:
18       - global-network
19
20   mongo_db:
21     image: "mongo:6.0.6"
22     restart: always
23     ports:
24       - 27017:27017
25     volumes:
26       - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
27     networks:
28       - global-network
29
30   networks:
31     global-network:
32       external: true
33
34   volumes:
35     VolumeT9:
36       external: true
37     VolumeT8:
38       external: true
```

Figura 5.7: Modifica Docker Compose Task1

Successivamente alle fasi preliminari di modifica del task 1, si è voluto verificare la corretta generazione del robot Evosuite e la copertura sulla classe da testare salvata in T1.

Si è pertanto riscontrato un **problema nel salvataggio del file di statistiche e di robot generati.**

Esso era causato da un salvataggio fatto tramite un path statico, perciò si è dovuto modificare il percorso nel file "misurazione livelli.sh" rendendolo dinamico (fig 5.8), utilizzando la variabile **salvataggio**.

```
1 #!/bin/bash
2 cont=0
3 PERCORSO="$(cd "$(dirname "$0")" && pwd)"
4 NOME_CLASSE=$(head -n 1 "output.txt")
5 NOME_PACKAGE=$(tail -n 1 "output.txt")
6 echo "nome classe :$NOME_CLASSE"
7 echo "nome package :$NOME_PACKAGE"
8 export EVOSUITE="java -jar $(pwd)/evosuite-1.0.6.jar"
9 filename=$PERCORSO/invio.txt
10
11 SALVATAGGIO="$(realpath -s "$(dirname "$0")"/..)" #aggiunto path della cartella 1 livello sopra
12 #####
13 #####
```

Figura 5.8: Modifica path misurazione livelli.sh

Inoltre sono state create dinamicamente anche le cartelle in cui salvare il file di statistiche e di generazione dei robot (fig. 5.9) e successivamente, è stato indicato il percorso corretto in cui salvare i file appena citati.

```
46 mkdir -p $SALVATAGGIO/FolderTreeEvo/$NOME_CLASSE/RobotTest/EvoSuiteTest/0${cont}Level/TestReport #aggiunto
47 mkdir -p $SALVATAGGIO/FolderTreeEvo/$NOME_CLASSE/RobotTest/EvoSuiteTest/0${cont}Level/TestSourceCode #aggiunto
48 mv evosuite-report/statistics.csv $SALVATAGGIO/FolderTreeEvo/$NOME_CLASSE/RobotTest/EvoSuiteTest/0${cont}Level/TestReport #modificato
49 mv I${var}/evosuite-tests/$NOME_CLASS $SALVATAGGIO/FolderTreeEvo/$NOME_CLASSE/RobotTest/EvoSuiteTest/0${cont}Level/TestSourceCode #modificato
50
51
52
```

Figura 5.9: Modifica path dinamico

### 5.3.2 Modifiche task 1

Avendo risolto i problemi in locale, si è successivamente modificata la funzione del backend del **TASK 1** che eseguiva automaticamente (alla chiamata dell’API per il salvataggio della classe caricata dall’amministratore) lo script di generazione dei test solo per Randoop.

E’ stato quindi modificato il file **RobotUtil.java**, in particolare la funzione *generateAndSaveRobots()* allo scopo di generare i test anche con Evosuite, come segue:

- Creazione directory in cui salvare la classe da testare
- Salvataggio della classe nella cartella appena creata
- Creazione e avvio del processo per il run del comando specificato in figura 5.10.

- Lettura del file ”**statistics.csv**” generato dal run di Evosuite (fig. 5.11 linea 221)
- Funzione **ad hoc** chiamata **LineCoverage()** che preleva dal file statistics e converte in intero solo il valore di copertura delle linee di codice (fig. 5.11 linee 223-224)
- Salvataggio per ogni livello di robot generato, dei valori di copertura nel database del task 4 tramite **richiesta http POST** (fig. 5.11 linee 228 a 248)

```
ProcessBuilder processBuilderE = new ProcessBuilder();
processBuilderE.command("bash", "robot_generazione.sh", cname, "\"\"", "/VolumeT9/app/FolderTree/" + cname + "/" + cname + "SourceCode", String.valueOf(liv));
processBuilderE.directory(new File("/VolumeT8/Prototipo2.0/"));
Process processE = processBuilderE.start();
```

Figura 5.10: Modifica path dinamico

```
217
218
219
220 ▼
221     File resultsDirE = new File("/VolumeT8/FolderTreeEvo/" + cname + "/RobotTest/EvoSuiteTest");
222
223     File resultsE [] = resultsDirE.listFiles();
224     for(File result : resultsE) {
225         int score = LineCoverageE(result.getAbsolutePath() + "/TestReport/statistics.csv");
226
227         System.out.println(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
228         int livello = Integer.parseInt(result.toString().substring(result.toString().length() - 7, result.toString().length() - 5));
229
230         System.out.println("La copertura del livello " + String.valueOf(livello) + " è: " + String.valueOf(score));
231
232         HttpClient httpClient = HttpClientBuilder.create().build();
233         HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/robots");
234
235         JSONArray arr = new JSONArray();
236
237         JSONObject rob = new JSONObject();
238         rob.put("scores", String.valueOf(score));
239         rob.put("type", "evoSuite");
240         rob.put("difficulty", String.valueOf(livello));
241         rob.put("testClassId", cname);
242
243         arr.put(rob);
244
245         JSONObject obj = new JSONObject();
246         obj.put("robots", arr);
247
248         StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);
249
250         httpPost.setEntity(jsonEntity);
251
252         HttpResponse response = httpClient.execute(httpPost);
```

Figura 5.11: Modifica file robotUtil.java T1

N.B: La funzione lineCoverageE() in figura 5.12 implementata in questo progetto, prende il valore (float) di copertura dal file statistics.csv, lo trasforma in valore percentuale e successivamente lo converte in intero (fig. 5.12 ).

In particolare, la prima riga del file statistics contiene le informazioni relative al contenuto delle colonne del file, pertanto è necessario andare a leggere la seconda riga.

Per poter prelevare l'elemento di interesse, bisogna splittare gli elementi separati da virgole attraverso la funzione **split()**.

A questo punto il valore di copertura delle linee di codice è quello contenuto nella terza colonna (elements[2]).

```
52     public static int LineCoverageE(String path) {
53         float elemento = 0.0f;
54
55         try (BufferedReader br = new BufferedReader(new FileReader(path))) {
56             // Leggi la prima riga (la riga 1 è la seconda riga nel conteggio base 1)
57             String firstLine = br.readLine();
58
59             // Leggi la seconda riga
60             String secondLine = br.readLine();
61
62             if (secondLine != null) {
63                 // Dividi la seconda riga in elementi separati da virgole
64                 String[] elements = secondLine.split(",");
65
66                 // Prendo il valore di copertura per linea
67                 elemento = Float.parseFloat(elements[2]);
68             }
69         } catch (IOException e) {
70             e.printStackTrace();
71         }
72
73         return Math.round(elemento * 100);
74     }
```

Figura 5.12: Funzione LineCoverageE

### 5.3.3 Omologazione del numero di livelli generati

Per quanto riguarda la **generazione del numero di livelli di difficoltà** dei test, è stato necessario omologare il numero di livelli generati da Evosuite con il numero generato da Randoop, già implementato.

La problematica risiedeva nel fatto che il Task 9, impiegato per la generazione dei test tramite randoop, generava il numero di livelli tramite una propria metrica.

Il task 8 invece dava la possibilità di settare il livello all'atto del lancio del comando di generazione dei test. Per attuare tale integrazione, è stato salvato in una variabile **"liv"** (file RobotUtil.java) il numero di livelli scelti dallo script di Randoop come mostrati in figura 5.13, e inserito successivamente nel comando di Evosuite (fig. 5.10).

```
File results [] = resultsDir.listFiles();
for(File result : results) {
    int score = LineCoverage(result.getAbsolutePath() + "/coveragegetot.xml");

    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) + " è: " + String.valueOf(score));

    HttpClient httpClient = HttpClientBuilder.create().build();
    HttpPost httpPost = new HttpPost("http://t4-g18-app-1:3000/robots");

    JSONArray arr = new JSONArray();

    JSONObject rob = new JSONObject();
    rob.put("scores", String.valueOf(score));
    rob.put("type", "randoop");
    rob.put("difficulty", String.valueOf(livello));
    rob.put("testClassId", cname);

    arr.put(rob);

    JSONObject obj = new JSONObject();
    obj.put("robots", arr);

    StringEntity jsonEntity = new StringEntity(obj.toString(), ContentType.APPLICATION_JSON);

    httpPost.setEntity(jsonEntity);
    HttpResponse response = httpClient.execute(httpPost);

    if(livello > liv)
        liv = livello;
}
```

Figura 5.13: Numero livelli file RobotUtil.java T1

## Problematica di compilazione

Una volta effettuate le modifiche e integrato il Task 8, è stato provato a runnare l'intera applicazione attraverso l'esecuzione dell'**installer.bat** che permette la creazione e l'avvio dei Docker.

Si è quindi provato a testare la funzionalità appena aggiunta andando a caricare nell'interfaccia dell'amministratore la classe da testare, per verificare anche la creazione dei robot di Evosuite.

Questo però non succedeva e dopo numerosi tentativi, si è arrivati al motivo che faceva sorgere questa problematica:

il task 1 deve essere prima buildato in locale attraverso il comando **”mvn clean install”**, questo perchè la compilazione non è effettuata automaticamente all'atto della costruzione del container, a

differenza di come viene fatto nel task 8.

Per i futuri sviluppatori, il consiglio dato è quello di verificare quali sono i task che effettuano una compilazione automatica e, se non dovesse essere così, dopo aver effettuato una modifica, compilare sempre manualmente.

## 5.4 Iterazione 1 - Sprint 4

Una volta risolta la problematica di compilazione, ci si è resi conto che i test sono stati generati correttamente da Evosuite all'atto del caricamento della classe da testare.

Nel Quarto Sprint, conseguentemente a uno Sprint Planning, si è effettuata la modifica del task 5 che si occupa della scelta delle classi da testare da parte dell'utente, e dei livelli di robot contro cui gareggiare.

### 5.4.1 Modifiche task 5 - Backend

Si è partiti con il modificare il file di **Backend ”GuiController.java”** del task 5 come segue:

- E' stato calcolato attraverso il ciclo for in figura 5.14 il numero di livelli totali generati per Evosuite, da aggiungere a quelli già calcolati per Randoop, tramite la richiesta http GET al Task4 per ricevere le informazioni dei livelli generati

- E' stato modificato l'**HashMap** che come valore non ha più una lista di Stringhe ma una lista di "**MyData**". Questo tipo di dato è stato creato ad hoc e spiegato nel paragrafo successivo.
- Per ogni classe, è stato creato un algoritmo di adattamento per ottenere il livello corretto dei robot generati, includendo anche quelli di Evosuite. Per una maggiore comprensione si rimanda allo studio del codice e la lettura dei commenti in figura 5.15
- E' stata riempita successivamente la lista struttura con i valori di "levels" e "evo"
- Sono stati assegnati agli HashMap i valori calcolati (linee 151 e 152)
- Sono passati gli Hashmap al FrontEnd (linee 156-160)

```
86     for(int j = i; j-i+1 < i; j++){  
87         try { // aggiunto  
88             restTemplate.getEntity("http://t4-g18-app-1:3000/robots?testClassId=" + className + "&type=evosuite&difficulty=" + String.valueOf(j-i+1), Object.class);  
89         } catch (Exception e) {  
90             break;  
91         }  
92         result.add(String.valueOf(j));  
93     }  
94     return result;  
95 }  
96  
97 }
```

Figura 5.14: Ciclo For Calcolo numero livelli

```
121     Map<Integer, String> hashMap = new HashMap<>();
122     Map<Integer, List<MyData>> robotList = new HashMap<>();
123     //Map<Integer, List<String>> evosuiteLevel = new HashMap<>();
124
125     for (int i = 0; i < classes.size(); i++) {
126         String valore = classes.get(i).getName();
127
128         //la funzione getLevels() restituisce il numero totale di Robot generati
129         List<String> levels = getLevels(valore);
130         System.out.println(levels);
131
132         List<String> evo = new ArrayList<>(); //aggiunto
133         for(int j = 0; j<levels.size(); j++){ //aggiunto
134             if(j>=levels.size()/2) //A partire dalla seconda metà dei livelli,
135                 //si aggiunge alla lista di stringhe "Evo" i livelli di Evosuite partendo da 1
136                 evo.add(j,levels.get(j-(levels.size()/2)));
137             else{
138                 evo.add(j,levels.get(j+(levels.size()/2)));
139             }
140         }
141         System.out.println(evo);
142
143         List<MyData> struttura = new ArrayList<>();
144
145         for(int j = 0; j<levels.size(); j++){
146             MyData strutt = new MyData(levels.get(j),evo.get(j));
147             struttura.add(j,strutt);
148         }
149         for(int j = 0; j<struttura.size(); j++)
150             System.out.println(struttura.get(j).getList1());
151         hashMap.put(i, valore);
152         robotList.put(i, struttura);
153         //evosuiteLevel.put(i, evo);
154     }
155
156     model.addAttribute("hashMap", hashMap);
157
158     // hashMap2 = com.g2.Interfaces.t8.RobotList();
159
160     model.addAttribute("hashMap2", robotList);
```

Figura 5.15: Hashmap Mydata

## Classe MyData

La classe MyData presenta due attributi di tipo stringa che hanno lo scopo di salvare i valori dei livelli relativi a Randoop, passato come primo parametro di ingresso nel costruttore, e Evosuite passato come secondo parametro.

```
5  public class MyData {
6      public String list1;
7      public String list2;
8
9      public MyData(String list1, String list2) {
10         this.list1 = list1;
11         this.list2 = list2;
12     }
13
14     public String getList1() {
15         return list1;
16     }
17
18     public void setList1(String list1) {
19         this.list1 = list1;
20     }
21
22     public String getList2() {
23         return list2;
24     }
25
26     public void setList2(String list2) {
27         this.list2 = list2;
28     }
29 }
```

Figura 5.16: Classe MyData

## 5.4.2 Modifiche task 5 - Frontend

Una volta modificato il Backend al fine di creare la logica per scegliere correttamente tra i due tipi di robot, si è dovuto riadattare anche il frontend del task 5 che permette all’utente la scelta vera e propria del robot da battere, nella schermata apposita (fig. 8.8).

- **Modifica file main.html**

E’ stato modificato il file **main.html** del task 5 come segue:

```

46  <div class="col-md-5 custom-column">
47    <!-- Contenuto della seconda colonna -->
48    <div class="v5_2111">
49      <div class="v5_2116">
50        <span class="v5_2117">CHOOSE A ROBOT TO FIGHT</span>
51        <div class="lista">
52          <!-- Creo una lista composta di elementi lista-item che stampano
53          il valore degli elementi contenuti nella lista java "classi" -->
54          <span th:each="entry: ${hashMap2}" th:id="${'levels-' + entry.key}" class="spa levels">
55            <!-- button th:each="value: ${entry.value}" th:id="${entry.key + '-' + value}" type="button"
56            | class ="lista-item my-button" th:text="${'randoop - LVL ' + value}" th:attr="|Handlebuttonrobot('${value}', this)"></button> modificato con quello sotto-
57            <button th:each="value, iterStat: ${entry.value}" th:id="${entry.key + '-' + value}" type="button"
58            | class="lista-item my-button" th:text="${iterStat.index < entry.value.size() / 2 ? 'randoop' : 'evosuite'} +
59            | ' - LVL ' + ${iterStat.index < entry.value.size() / 2 ? value.list1 : value.list2}" th:attr="|Handlebuttonrobot('${value.list1}', this, ${iterStat.index < entry.value.size() / 2 ? 'randoop' : 'evosuite'})', ${entry.value.size()} |></button>
60          </span>
61        </div>
62      </div>
63    </div>
64  </div>
65 </div>
66 </div>
67

```

Figura 5.17: Main.html task 5

Per raffigurare i robot relativi alla classe selezionata, si cicla sulla dimensione del HashMap2 che corrisponde al numero totale dei livelli generati per quella classe.

La prima metà dell’HashMap contiene i livelli di Randoop, la seconda quelli di Evosuite.

Per mostrare il livello giusto, si preleva il valore corretto salvato nella classe MyData.

- Modifica file main.js

```
57     function Handlebuttonrobot(id, button, rob, size) { //modificato
58         $(document).ready(function () {
59             robot = rob;
60             if(robot == "evoSuite"){ // aggiunto
61                 difficulty = parseInt(id)-parseInt(size)/2;
62                 difficulty = difficulty.toString();
63             }
64             else{ // aggiunto
65                 difficulty = id;
66             }
67             //difficulty = id;
68             console.log('Hai cliccato sul bottone del robot con id: ' + robot);
69
70             // Se il bottone precedentemente selezionato è diverso da null
71             // allora rimuoviamo la classe highlighted
72             if (bottonePrecedente2 != null) {
73                 bottonePrecedente2.classList.remove("highlighted");
74             }
75
76             if (button.classList.contains("highlighted")) {
77                 button.classList.remove("highlighted");
78             } else {
79                 button.classList.add("highlighted");
80             }
81             bottonePrecedente2 = button;
82
83         });
84     }
```

Figura 5.18: Modifica file main.js

Nel file main.js, la funzione **”Handlebuttonrobot()”** è stata modificata aggiungendo 2 ulteriori parametri: rob e size. Il primo è utile a passare dinamicamente il nome del robot selezionato, in quanto precedentemente era assegnato staticamente **”randoop”**.

Il secondo è, invece, utile al calcolo del livello del robot selezionato nel caso si fosse scelto EvoSuite.

Questo perché il livello è assegnato a partire dall'id del hashmap.

Poiché i robot di Evosuite sono situati nella seconda metà della struttura, per poter calcolare il livello effettivo del robot scelto è necessario sottrarre all'id la metà della size del hashmap (linea 61).

Settare queste variabili di robot e difficulty correttamente, insieme alla classe selezionata, è utile al fine di poter costruire il "localStorage" alla pressione del bottone di Submit nell'interfaccia, in modo tale da poter rendere disponibili queste informazioni alle schermate html successive.

- **Modifica file editor.html**

A questo punto si è potuto testare il corretto funzionamento delle modifiche effettuate, andando a simulare un "run" nell'interfaccia di editor.

Questo portava all'output dei risultati soltanto per Randoop. La causa era che non era predisposta l'interfaccia alla selezione di più tipologie di Robot.

E' stato perciò modificato il file **editor.html** utilizzando la variabile "localStorage" già descritta per richiamare il robot opportuno dinamicamente.

## Package Diagram Task 5

Di seguito il Package Diagram che descrive il pattern MVC usato nel Task 5:

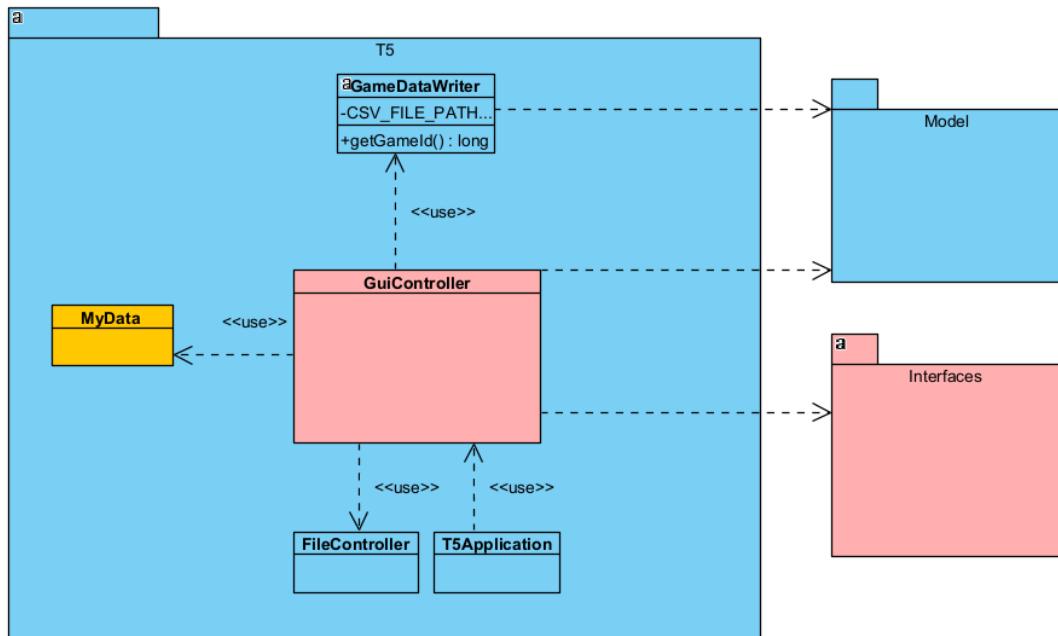


Figura 5.19: Package Diagram task 5 Class myData

Si noti che è stata aggiunta la classe MyData come descritto nel paragrafo precedente.

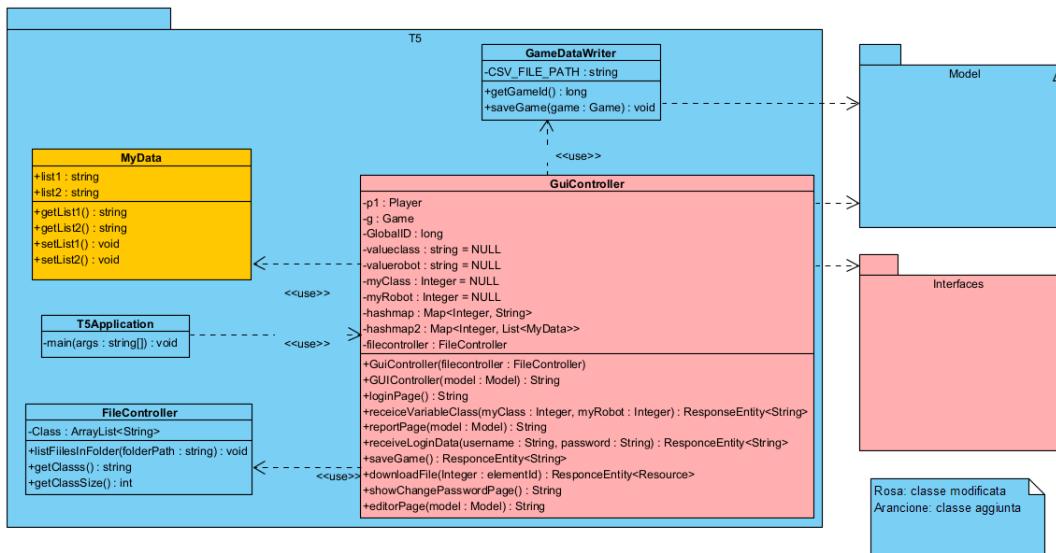


Figura 5.20: Package Diagram basso livello

## 5.5 Iterazione 2 - Sprint 1

Con l'ultima modifica della prima iterazione, è stata finalmente **completata l'implementazione della prima user stories** decritta nel Capitolo 4.

A valle di un secondo Iteration Planning, è iniziata la seconda iterazione con lo scopo di implementare la seconda delle user stories.

**Goal1: Stampare sulla console la percentuale di copertura fornita da Evosuite**

Il **goal** in questa sezione è quello di modificare la funzionalità del tasto **”Run with Jacoco”** che colorava le linee di codice coperte dal test utilizzando Jacoco (si legga le sezione 2.3.7), aggiungendo la possibilità di ricevere in output il punteggio (espresso in percentuale)

delle linee di codice coperte calcolato con **Evosuite**.

Il tasto "Run with Jacoco" sarà denominato "**Run**".

### 5.5.1 Modifica server Task 8

Le modifiche descritte di seguito sono state effettuate nel file **prova\_esecuzione\_parametri4.js** del task 8 e si possono osservare in figura 5.21.

```

8     let body = '';
9     req.on('data', chunk => {
10       body += chunk.toString(); // convert Buffer to string
11     });
12     req.on('end', () => {
13       const codiceJava = body; // Il codice Java inviato nel corpo della richiesta POST
14
15       //Rimuove '/api/' dalla stringa dell'URL e sostituisce '+' con spazi
16       var paramString = req.url.slice(5).replace(/\+/g, ' ');
17       var parametri = paramString.split(' ');//Converte la stringa in un array di stringhe
18       var params = parametri[0];//salva il primo path (contenente il percorso fino alla classe .java)
19       var filePath = params.substring(0, params.lastIndexOf('/'));//salva il path fino al package
20       var prova = params.substring(params.lastIndexOf('/') + 1 );
21       var filename = prova.substring(0, prova.lastIndexOf("."));//salva il nome della classe.java
22       var directoryName = filePath.substring(filePath.lastIndexOf('/') + 1); //salva il nome del package
23
24       //Elimina la prima stringa dall'array, così da poterla aggiornare con i parametri sopra descritti
25       parametri.shift();
26       parametri = "/" + filePath + " " + directoryName + " " + filename + " " + parametri[0] + " " + parametri[1];
27       console.log (parametri);
28
29       //aggiunto codice ricezione classe di test
30       //percorso salvataggi classe di test
31       const percorsoFile = '/app/Serv/src/test/java/Tests/Test'+ filename + '.java';
32
33       fs.writeFile(percorsoFile, codiceJava, err => { //scrittura del test ricevuto
34         if (err) {
35           console.error(err);
36           res.status(500).send('Si è verificato un errore durante il salvataggio del file');
37         } else {
38           const command = `sh robot_misurazione_utente.sh ${parametri}`;
39           exec(command,
40             function (error, stdout, stderr) {
41               if (error !== null) {
42                 console.log(error);
43               } else {
44                 console.log('stdout: ' + stdout); console.log('stderr: ' + stderr);
45                 //aggiunto
46                 const csvContent = fs.readFileSync('/app/statistics.csv', 'utf8');
47                 console.log(csvContent);
48
49                 // Imposta gli header
50                 res.setHeader('Content-Type', 'text/csv');
51                 res.setHeader(
52                   'Content-Disposition',
53                   'attachment; filename="${filename}.csv"';
54                 );
55
56                 // Invia il contenuto del file come corpo della risposta
57                 res.end(csvContent);
58               }
59             }
60           );
61         }
62       });
63     });
64   });
65 }
66
67 module.exports = router;

```

Figura 5.21: Modifica file prova esecuzione parametri4.js

Come si può osservare è stato effettuato:

- Modifica richiesta http che prima era una "get" e ora una "post", che riceve anche il test scritto dall'utente da cui calcolare la copertura (linea 8-13)
- Salvataggio del test in locale al server

- Esecuzione dello script **”robot\_misurazione\_utente.sh”**  
che calcola la copertura di linee di codice sulla classe da testare
- Generazione del file **”statistics.csv”** con le statistiche di copertura
- Invio di quest’ultimo in risposta al richiedente (editor.html)

### **5.5.2 Modifica script di generazione statistiche**

#### **Task 8**

E’ stato modificato il file **robot\_misurazione\_utente.sh** (fig. 5.22) del task 8 come segue:

- Sono state eliminate le dipendenze dai package (problematica già descritta nella sezione precedente)
- E’ stata eliminato il comando di creazione del file ”pom.xml” e creato il file manualmente nella stessa cartella

```

33 # cat >>pom.txt << EOF
34 # <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
35 #   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
36 #   <modelVersion>4.0.0</modelVersion>
37 #   <groupId>$NOME_PACCHETTO</groupId>
38 #   <artifactId>$NOME_CLASSE</artifactId>
39 #   <version>1.0-SNAPSHOT</version>
40 #   <packaging>jar</packaging>
41 #   <name>$NOME_CLASSE</name>
42 #   <url>http://maven.apache.org</url>
43 #   <properties>
44 #     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
45 #   </properties>
46 #   <dependencies>
47 #     <dependency>
48 #       <groupId>junit</groupId>
49 #       <artifactId>junit</artifactId>
50 #       <version>4.13.1</version>
51 #       <scope>test</scope>
52 #     </dependency>
53 #   </dependencies>
54 # </project>
55 # EOF
56 # mv pom.txt pom.xml
57
58 export EVOSUITE="java -jar $(pwd)/evosuite-1.0.6.jar"
59
60 mvn clean install
61
62 echo "facciamo partire i test"
63
64 mvn dependency:copy-dependencies
65
66 export CLASSPATH=target/classes:evosuite-standalone-runtime-1.0.6.jar:target/test-classes:target/dependency/junit-4.13.1.jar:target/dependency/hamcrest-core-1.3.jar
67 #javac target/test-classes/$NOME_PACKAGE/*.java
68
69 #java org.junit.runner.JUnitCore $NOME_PACKAGE.$(NAME_CLASSE)_test
70
71 sleep 2
72
73 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test$(NAME_CLASSE) -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=LINE
74
75 cd $PERCORSO
76
77 mv -f evosuite-report/statistics.csv $PERCORSO_CSV
78
79 wget http://localhost:3081/api/
80
81 rm -r $PERCORSO/test/*
82 #rm -r $PERCORSO/src/main/java/*
83 #rm -r $PERCORSO/target
84 rm -r $PERCORSO/evosuite-report
85 rm $PERCORSO/index.html

```

Figura 5.22: Modifica file robot\_misurazione\_utente.sh

Il file **pom.xml** creato manualmente, è stato reso generico permettendo la compilazione di qualsiasi test.  
E' stato inoltre aggiunta la dipendenza da **JUnit 4.13.2** compatibile con la scrittura dei test, e da **Java 1.8** versione compatibile per Evosuite.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3   |   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   |   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5
6   <modelVersion>4.0.0</modelVersion>
7   <groupId>Tests</groupId>
8   <artifactId>code-coverage</artifactId>
9   <packaging>jar</packaging>
10  <version>1.0-SNAPSHOT</version>
11
12  <properties>
13      <!-- https://maven.apache.org/general.html#encoding-warning -->
14      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15      <maven.compiler.source>1.8</maven.compiler.source>
16      <maven.compiler.target>1.8</maven.compiler.target>
17  </properties>
18
19  <dependencies>
20      <dependency>
21          <groupId>org.xmlunit</groupId>
22          <artifactId>xmlunit-core</artifactId>
23          <version>2.9.0</version>
24          <scope>test</scope>
25      </dependency>
26      <!-- junit 4, unit test -->
27      <dependency>
28          <groupId>junit</groupId>
29          <artifactId>junit</artifactId>
30          <version>4.13.2</version>
31          <scope>test</scope>
32      </dependency>
33
34  </dependencies>
35  <build>
36      <finalName>maven-code-coverage</finalName>
37      <plugins>
38          <plugin>
39              <groupId>org.apache.maven.plugins</groupId>
40              <artifactId>maven-surefire-plugin</artifactId>
41              <version>3.0.0-M1</version>
42          </plugin>
43      </plugins>
44  </build>
45
46</project>
```

Figura 5.23: Modifica file pom.xml server T8

### 5.5.3 Modifica Docker file e Docker compose Task 8

Si noti che è stato modificato il **Docker file** già mostrato in figura 5.4 arrivando alla versione definitiva.

In questa versione, all'avvio del container del Task 8, viene runnato lo script **prova\_esecuzione\_parametri.js** per generare le statistiche di Evosuite:

```
1  FROM ubuntu:latest
2
3  COPY . /app
4
5  WORKDIR /app
6
7  RUN bash installazione.sh
8
9  RUN bash ./docker-entrypoint.sh
10
11 WORKDIR /app/Serv
12
13 EXPOSE 3080
14 ENTRYPOINT [ "node", "prova_esecuzione_parametri4.js" ]
```

Figura 5.24: Docker file versione definitiva T8

E' stato inoltre riportato alla versione definitiva anche il file **Docker compose** configurando il porto **3080** in cui il server è posto in ascolto:

```
1  version: '2'
2
3  services:
4    t8_generazione:
5      build: ./t8_generazione
6      ports:
7        - 3080:3080
8      networks:
9        - global-network
10     volumes:
11       - VolumeT8:/VolumeT8
12
13   networks:
14     global-network:
15       external: true
16
17   volumes:
18     VolumeT8:
19       external: true
```

Figura 5.25: Docker compose versione definitiva T8

## 5.6 Iterazione 2 - Sprint 2

### 5.6.1 Creazione e gestione richiesta http POST bottone RUN

Configurato correttamente il server per la ricezione delle richieste http POST, è necessaria la configurazione del client **editor.html** del task 6 nell'invio delle richieste http al server per ottenere le statistiche generate da Evosuite, come segue:

```
1022 |         var classe = 'VolumeT8/FolderTreeEvo/' + localStorage.getItem("classe")
1023 |         + '/' + localStorage.getItem("classe") + 'SourceCode/' + localStorage.getItem("classe") + '.java';
1024 |
1025 |         var test = '/VolumeT8/Serv/src/test/java/Tests/Test' + localStorage.getItem("classe") + '.java'; //editor.getValue();
1026 |
1027 |         // Definisci il percorso dell'API
1028 |         var apiUrlBase = 'http://localhost:3080/api/';
1029 |
1030 |         // Concatena il percorso della classe al percorso dell'API
1031 |         var url = apiUrlBase + classe + '+' + test + '+app';
1032 |
1033 |         const javaCode = editor.getValue(); // codice della classe di test
1034 |
1035 |         fetch(url, {
1036 |             method: 'POST',
1037 |             headers: { 'Content-Type': 'text/plain' },
1038 |             body: javaCode
1039 |         }) // resa funzione una POST
1040 |         .then(function(response) {
1041 |             if (!response.ok) {
1042 |                 throw new Error('Errore nella richiesta HTTP: ' + response.status);
1043 |                 console.log('Errore nella richiesta HTTP: ', response.status);
1044 |             }
1045 |
1046 |             return response.blob();
1047 |         })

```

Figura 5.26: Modifica file Editor.html Task6

Si noti che la richiesta http è stata generata secondo il formato previsto e predisposto dal gruppo G21 che ha implementato il task 8.

## Problematiche riscontrate

All'atto del testing delle funzionalità aggiunte, sono state riscontrate 3 problematiche principali:

1. Il server del task 8 era stato configurato per ricevere solo le richieste http locali. E' stato pertanto modificato il file 5.21 aggiungendo la possibilità di ricevere richieste http da qualsiasi sorgente utilizzando il codice:

```
res.setHeader('Access-Control-Allow-Origin', '*')
```

2. Esisteva un'incompatibilità di versioni tra il test scritto in **JU-nit5**, Evosuite compatibile solo con **JUnit4**, e Jacoco che runnava i test scritti in **JUnit5**. La soluzione finale pertanto prevede:

- JUnit versione 4
- Jacoco e Evosuite compatibili con Junit4

### 5.6.2 Adattamento dipendenza del task 7 da JUnit4

E' stata effettuata la **modifica del task 7** che richiama Jacoco nel file **pom.xml** per eliminare la dipendenza da Junit5 e aggiungere quella da JUnit4.

3. Problema di sincronismo tra client (editor.html del task 6) e server: il client manda la richiesta http per ottenere le statistiche e attraverso una post passa il test scritto in JUnit al server del task 8.

Il server non aspettava di ricevere il file dal client ma tentava subito di fare operazioni su di esso.

La soluzione è stata quella di effettuare la sincronizzazione attraverso una verifica dell'errore sulla funzione fs.writeFile() e l'inserimento del codice nel ramo else. (figura 5.21 da linea 37).

### 5.6.3 Adattamento buttoni Run e Submit

**Specifiche:** Modificare l'interfaccia di editor e renderla coerente con l'integrazione

Come descritto nella sezione sui miglioramenti, il tasto **Run with Jacoco**, deve diventare solo **Run**, e al click deve far apparire la copertura del proprio codice nella schermata “Console”.

Il tasto che ora si chiama **Run**, è invece quello legato al game vero e proprio, ovvero permette di gareggiare con il robot scelto.

Esso verrà chiamato tasto di **Submit**.

## 5.6.4 Sequence Diagram Run

E' stata portata a termine la funzionalità di RUN.

Il sequence Diagram di seguito schematizza le interazioni tra le classi dei Task coinvolti.

### Sequence Diagram Run

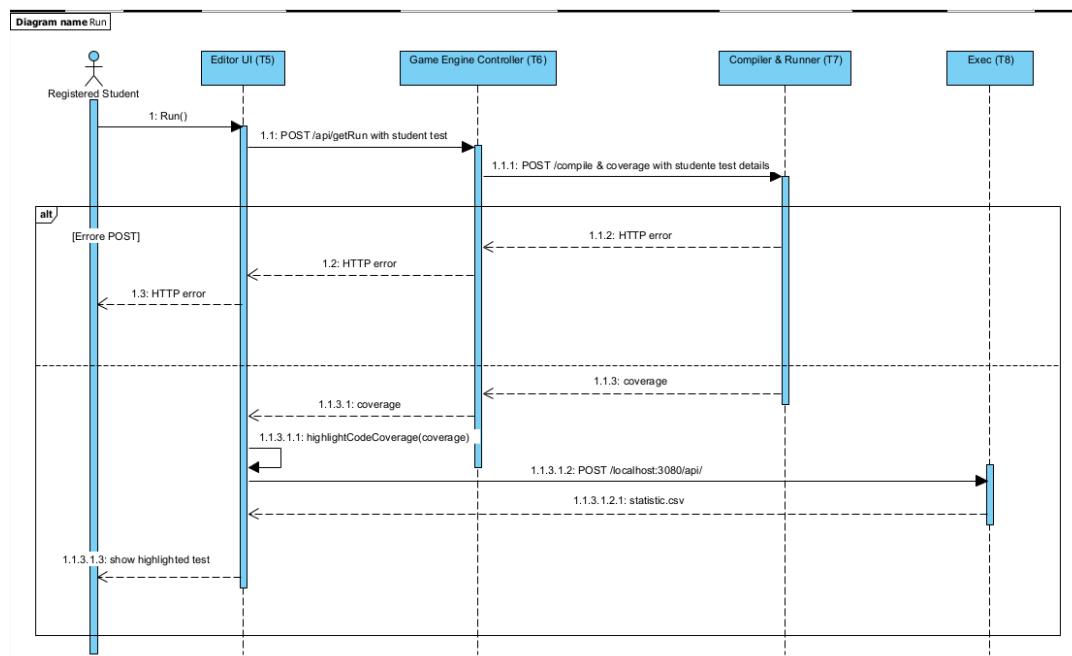


Figura 5.27: Sequence Diagram Run

Rispetto all'implementazione precedentemente che prevedeva solo l'uso di JaCoCo, adesso i dati della classe sono mandati tramite richiesta http POST all'EXEC T8 per compilare la classe scritta dall'utente con EvoSuite e ricevere i risultati della copertura raggiunta.

## 5.7 Iterazione 2 - Sprint 3

### 5.7.1 Creazione e gestione richiesta http POST bottone Submit

Le modifiche effettuate sono riportate di seguito:

1. Aggiunta richiesta http POST all'API fornita dal task 8 nel file "editor.html" legata al tasto di submit per ottenere la copertura del codice calcolata con Evosuite
2. Modifica dell'output schermata **Confronto Risultati**:  
E' stato modificato il file "editor.html" del task 6 creando una variabile "risp", utilizzata per salvare la risposta alla richiesta http verso il task 7, per poi ottenere il concatenamento del messaggio in modo da visualizzare sia il punteggio ottenuto attraverso Evosuite, che quello attraverso Jacoco.
3. Modifica del valore percentuale del risultato in modo da omologare gli output di Evosuite e Jacoco
4. Modifica delle precedenti icone di "Run" e "Run with Jacoco".

Con le modifiche descritte in questi paragrafi, una volta testate opportunamente e verificato il corretto comportamento, si è potuto considerare completato il **lavoro di integrazione**.

## 5.7.2 Creazione "Logout" in interfaccia di Editor

Come descritto nel Capitolo 4, si è deciso di aggiungere alla Web App una nuova funzionalità.

**Goal:** Permettere all'utente la possibilità di effettuare un Logout

- Modifica file **editor.html** per l'inserimento del bottone di logout. È stato aggiunto nella *toolbar* il bottone di logout uniformandolo agli altri bottoni già presenti (Css) con identificativo **“logout-button”** (fig. 5.28)

```
155     .logout-button{ /* aggiunto */
156         background-image: url("./t5/Icone/logout.png");
157         margin-left: 3%;
158     }
```

Figura 5.28: Inserimento logout-button

- Collegamento con l'icona di logout nell'interfaccia (`<button id="logout" class="logout-button" title="Logout"></button>`)
- Creazione in Html del bottone con id “logout”, collegamento con immagine button del css e aggiunta della funzione **handler** relativa al click del button di logout in figura 5.29

```

623     //funzione handler del tasto di logout
624     var logout = document.getElementById("logout");
625     logout.addEventListener("click", function(){
626         if(confirm("Sei sicuro di voler effettuare il logout?")){
627             fetch('http://localhost/logout', {
628                 method: 'GET',
629             })
630             .then(response => {
631                 if (!response.ok) {
632                     throw new Error('Richiesta logout non andata a buon fine');
633                 }
634                 else{
635                     console.log("stai per essere reindirizzato alla pagina di login");
636                     window.location.href = "/login";
637                 }
638             })
639             .catch(error) => {
640                 console.error('Error:', error);
641             });
642         }
643     });
644 
```

Figura 5.29: Funzione Handler Logout

- Richiesta http POST sulla porta 80 fatta al server implementato dal **task 2-3** situato nel file “**Controller.java**”, per permettere l’eliminazione dei cookies che mantenevano all’interno dell’applicazione i dati di quella sessione (fig. 5.30).

Vi è quindi un reindirizzamento alla schermata di Login.

```

223     // Logout
224     @GetMapping("/logout")
225     public ModelAndView logout(HttpServletRequest response) {
226         Cookie jwtTokenCookie = new Cookie("jwt", null);
227         jwtTokenCookie.setMaxAge(0);
228         response.addCookie(jwtTokenCookie);
229
230         return new ModelAndView("redirect:http://localhost/login");
231     } 
```

Figura 5.30: Modifica file controller.java

### 5.7.3 Creazione "Logout" in interfaccia di scelta della classe

E' stato inserito il tasto di Logout anche nella schermata di scelta della classe da testare e dei robot da battere, con queste modifiche:

E' stato inserito il bottone di Logout nel file **main.html** del task 5.

La sua pressione va a richiamare la funzione javascript "**redirectToLogin()**" presente nel file **main.js** la quale effettua la richiesta http precedentemente illustrata.

```
17 <div class="mt-2 row">
18   <p class="user_position_main" style="color: black; font-family: Arial, Helvetica, sans-serif; font-weight: bold;">Username: <span id="usernameField"></span></p>
19   <button id="logoutButton" type="button" class="logout-button" th:onclick="redirectToLogin()">Logout</button>
20 
```

Figura 5.31: Modifica main.html per logout

Una volta effettuata quest'ultima modifica, è stata implementata correttamente la funzione di Logout.

### 5.7.4 Sequence Diagram Logout

Di seguito il Sequence Diagram che mostra lo scambio di chiamate tra le classi dopo la pressione da parte dell'utente sul pulsante di Logout.

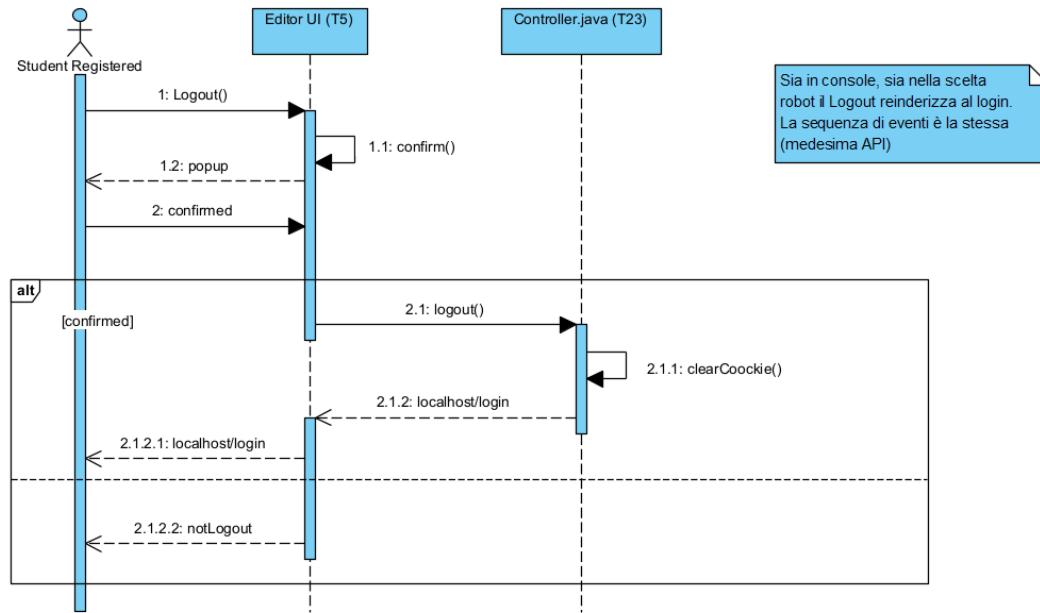


Figura 5.32: Sequence Diagram Logout

## 5.8 Iterazione 2 - Sprint 4

**Storia:** Permettere all’utente di giocare più turni in una sola partita

**Specifiche:** Alla pressione del tasto Submit, la partita non deve terminare individuando un solo punteggio e stabilendo se si ha vinto o si ha perso, ma deve essere possibile rigiocare il turno ed effettuare altri tentativi.

### 5.8.1 Modifica logica di fine partita

**Modifiche effettuate nel file editor.html.**

Nella richiesta http al task 7 del tasto Submit, in base all’esito del

turno, si è:

1. Incrementata una variabile ”**turno**” che tiene traccia dei turni effettivamente giocati fino a quel momento
2. In caso di vittoria viene resettato il gameId all’interno del ”**localStorage**”.

Viene effettuata infatti una verifica al gameId ogni volta che si clicca sul tasto di Submit.

Se il **gameId** è **null** significa che la partita è stata vinta e non è possibile giocare un ulteriore turno. (fig 5.34)

### 5.8.2 Caso di sconfitta - creazione nuovo turno

In caso di sconfitta viene effettuata una richiesta http verso il task 5 il quale effettua delle richieste POST verso il task 4 al fine di creare un nuovo turno. In risposta alla richiesta vengono restituiti i nuovi valori per gameId, roundId e turnId. (fig 5.33).

Le richieste http possono essere osservate nel dettaglio nel Sequence Diagram 5.35.

In una prima implementazione il valore delle variabili gameId, roundId e turnId veniva incrementato localmente e si effettuavano le richieste POST manualmente verso il task 4.

Tali richieste non erano effettuate nel modo corretto e quindi si è risolto richiamando la richiesta http ”*localhost/api/sava-data*” per la creazione del nuovo turno di gioco.

```

794     alert(response.win == true ? "Hai vinto!" : "Hai perso!");
795     //localStorage.clear();
796     if(response.win == true){
797         turno++; // incremento il numero di turno giocati fino ad ora
798         //localStorage.clear();
799         localStorage.setItem("gameId",null); // setto gameId null invece che tutto
800         //console.log(localStorage.getItem("gameId"));
801     }
802     else{
803         //localStorage.setItem("gameId",(parseInt(localStorage.getItem("gameId"))+1).toString());
804         //localStorage.setItem("roundId",(parseInt(localStorage.getItem("roundId"))+1).toString());
805         //localStorage.setItem("turnId",(parseInt(localStorage.getItem("turnId"))+1).toString());
806         //console.log(localStorage.getItem("turnId"));
807
808         $.ajax({
809             url:'http://localhost/api/save-data',
810             data: {
811                 playerId: parseJwt(getCookie("jwt")).userId,
812                 classe: localStorage.getItem("classe"),
813                 robot: localStorage.getItem("robot"),
814                 difficulty: localStorage.getItem("difficulty")
815             },
816             type:'POST',
817             success: function (response) {
818                 // Gestisci la risposta del server qui
819                 localStorage.setItem("gameId", response.game_id);
820                 localStorage.setItem("turnId", response.turn_id);
821                 localStorage.setItem("roundId", response.round_id);
822                 console.log(localStorage.getItem("turnId"));
823                 turno++; // incremento il numero di turno giocati fino ad ora
824                 //window.location.href = "/editor";
825             },
826             dataType: "json",
827             error: function (error) {
828                 console.error('Errore nell invio dei dati');
829                 alert("Dati non inviati con successo");
830                 // Gestisci l'errore qui
831             }
832         });
833     }
834 }
```

Figura 5.33: Modifica editor.html per turni

```

757     $(document).ready(function() {
758         if(localStorage.getItem("gameId") == "null") { //controllo game invece che turn
759             alert("Impossibile effettuare un nuovo tentativo: Hai già vinto!");
760         } else {
761             var risp; // variabile per salvare response
762             var formData = new FormData();
763             formData.append("testingClassName", "Test" + localStorage.getItem("classe") + ".java");
764             formData.append("testingClassCode", editor.getValue());
765             formData.append("underTestClassName", localStorage.getItem("classe")+".java");
766             formData.append("underTestClassCode", sidebarEditor.getValue());
767
768             formData.append("turnId",localStorage.getItem("turnId"));
769             formData.append("roundId",localStorage.getItem("roundId"));
770             formData.append("gameId",localStorage.getItem("gameId"));
771             formData.append("testClassId", localStorage.getItem("classe"));
772             formData.append("difficulty", localStorage.getItem("difficulty"));
773             formData.append("type", localStorage.getItem("robot")); // modificato
```

Figura 5.34: Verifica vittoria

## Sequence Diagram Submit

Di seguito sono descritte le interazioni temporali tra gli oggetti all'interno del sistema dopo le modifiche effettuate, alla pressione da parte dell'utente del pulsante di Submit.

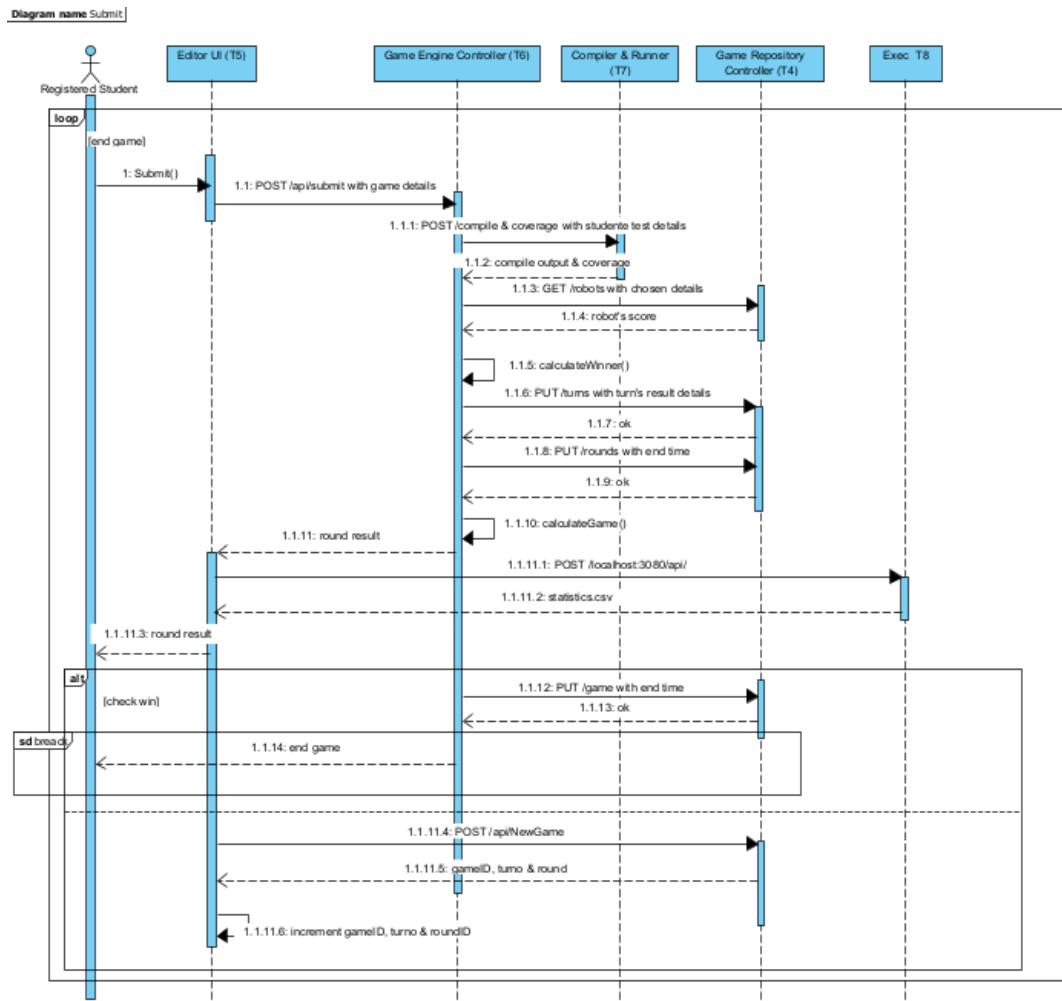


Figura 5.35: Sequence Diagram Submit

Come è possibile osservare dal Sequence Diagram, una volta ottenuti i risultati per il turno giocato, viene verificata la vittoria o la sconfitta del giocatore.

Nel caso di vittoria il Game si può considerare concluso ("break"), altrimenti viene creato un nuovo turno in cui poter giocare.

### 5.8.3 Creazione e gestione "Storico"

**Goal:** Possibilità di rivedere i tentativi di gioco in uno STORICO

**Specifiche:** Ogni volta che è premuto il tasto di Storico, si effettua una richiesta http al database (T4) ciclando sulla variabile turno, che restituisce le informazioni da inserire nella schermata di editor:

- risultato da battere
- percentuale di LOC ottenuta in ogni turno
- esito della partita

Come è stata realizzata l'implementazione effettiva del goal da raggiungere è mostrata di seguito:

1. E' stato modificato il file editor.html per l'inserimento del bottone di Storico. E' stato aggiunto nella *toolbar* il bottone di storico uniformandolo agli altri bottoni già presenti (Css) con identificativo "**storico-button**" (fig. 5.36)

```
159 |     .storico-button{ /* aggiunto */
160 |       background-image: url("./t5/Icone/storico.png");
161 |       margin-left: 3%;
162 |   }
```

Figura 5.36: Modifica editor.html per turni

2. Collegamento con l'icona di storico nell'interfaccia

```
(<button id="storico" class="storico-button"
title="Storico"></button>)
```

3. In Html creazione del bottone con id “storico”, collegamento con immagine button del css e aggiunta della funzione **handler** relativa al click del button di storico in figura 5.37

```

645     //funzione handler del tasto di storico
646     var storico = document.getElementById("storico");
647     storico.addEventListener("click", function(){
648       if(turno == 0){
649         async function valRob() {
650           try {
651             let response = await fetch('http://localhost:3000/robots?testClassId='+localStorage.getItem("classe")+
652               '&type=' +localStorage.getItem("robot") + '&difficulty=' +localStorage.getItem("difficulty"),
653               { method: 'GET' });
654             if (!response.ok) {
655               throw new Error('Richiesta risultati robot non andata a buon fine');
656             }
657             let data = await response.json();
658             perc_robot = data.scores;
659           } catch (error) {
660             console.error('Error:', error);
661           }
662           alert("Non esiste ancora uno storico dei test\nPercentuale di copertura da battere: "+perc_robot);
663         }
664         valRob();
665       }
666     }else{
667       var dastampare = "Percentuale di copertura da battere: " + perc_robot + "\n";
668
669       async function fetchTurns(turno) {
670         for(var i=1; i<=turno; i++){
671           if(localStorage.getItem("gameId") == "null"){
672             var val = parseInt(localStorage.getItem("turnId"))-turno+i;
673           }else{
674             var val = parseInt(localStorage.getItem("turnId"))-turno+(i-1);
675           }
676
677           try {
678             let response = await fetch('http://localhost:3000/turns/' +val.toString(), { method: 'GET' });
679             if (!response.ok) {
680               throw new Error('Richiesta risultati turno non andata a buon fine');
681             }
682             let data = await response.json();
683             if(data.isWinner)
684               dastampare += "Tentativo " + i.toString() + ": Vittoria\n" + "Percentuale di copertura ottenuta: " + data.scores + "\n";
685             else
686               dastampare += "Tentativo " + i.toString() + ": Sconfitta\n" + "Percentuale di copertura ottenuta: " + data.scores + "\n";
687             console.log(data.scores);
688           } catch (error) {
689             console.error('Error:', error);
690           }
691         }
692       }
693       dastampare +=
694       consoleArea.setValue(dastampare);
695       console.log(dastampare);
696     }
697     fetchTurns(turno);
698   }
699 }
700 );

```

Figura 5.37: Verifica vittoria

N.B: Come si evince dal codice in figura, alla pressione del pulsante di Submit, viene verificato se è stato già effettuato almeno un tentativo di gioco.

Se il turno è il primo, in output verrà fornito semplicemente il valore di copertura di LOC da superare.

## 5.8.4 Salvataggio del valore di copertura

Si riscontrata l'esigenza di salvare i valori di copertura, in modo tale da avere un messaggio di Alert alla **pressione del tasto Storico se ancora non è stato effettuato un Submit**.

In questa sezione si è fatto in modo di **salvare in locale al Task 4** i valori di copertura ottenuti dal test sulla classe scelta.

Per non effettuare ogni volta la richiesta http al task 4 per ottenere le informazioni sul robot selezionato, si è scelto quindi di utilizzare una variabile **perc\_robot** (inizializzata a 0) allo scopo di salvare la percentuale del robot da battere, per poi poter usufruire dei valori salvati.

Il salvataggio in questa variabile viene effettuato in due possibili occasioni:

- Alla pressione del tasto Submit viene salvare il valore di copertura nella variabile `perc_robot=response.robotScore.toString`
- Alla pressione del tasto storico non avendo ancora iniziato la partita:  
viene fatta una richiesta al task 4 (linea 647) per avere il valore corretto del robot.

## 5.8.5 Sequence Diagram Storico

Il Sequence Diagram mostra il meccanismo appena descritto che parte alla pressione del tasto di Storico da parte dell'utente.

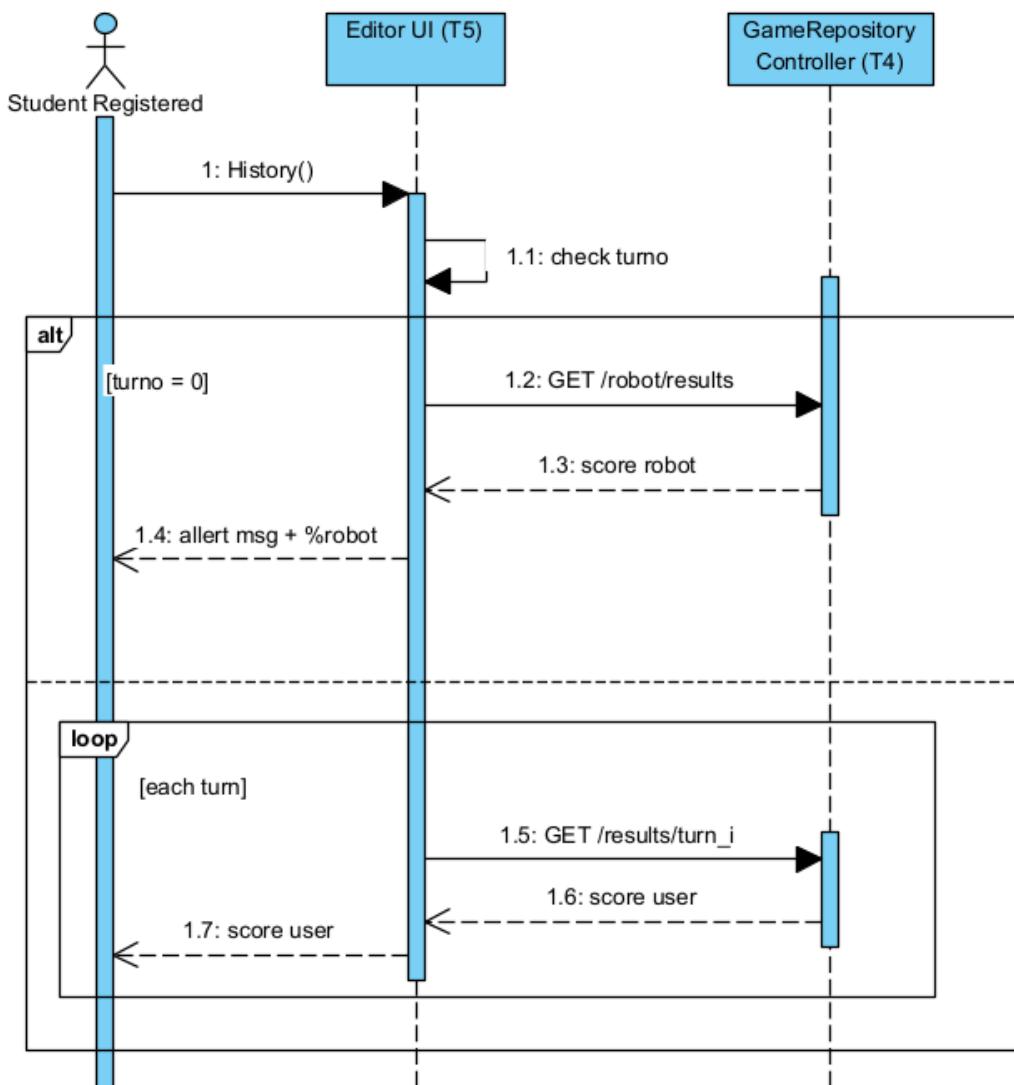


Figura 5.38: Sequence Diagram Storico

## 5.8.6 Ulteriori metriche di copertura di EvoSuite

**Goal:** Possibilità di visualizzare le informazioni aggiuntive calcolate con EvoSuite.

**Specifico:** Aggiungere ulteriori coperture di EvoSuite (oltre alla copertura per linee di codice) e visualizzarle nella schermata di editor.

Per raggiungere l'obiettivo previsto, è stato:

- Modificato ulteriormente lo script del server **robot\_generazione\_utente.sh** del Task 8 per il run di evo-suite per il calcolo di ulteriori criteri di copertura

```
79 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=LINE
80
81 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=BRANCH
82
83 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=EXCEPTION
84
85 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=WEAKMUTATION
86
87 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=OUTPUT
88
89 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=METHOD
90
91 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=METHODNOEXCEPTION
92
93 $EVOSUITE -measureCoverage -class $NOME_CLASSE -Djunit=Test${NOME_CLASSE} -projectCP $PERCORSO/target/classes:$PERCORSO/target/test-classes -Dcriterion=CBRANCH
94
```

Figura 5.39: Aggiunta metriche di copertura

- Modificato il client **editor.html** per prelevare dal file statistics.csv le righe contenenti tutte le informazioni aggiuntive di copertura.

Per ottenerle, si è dovuta attuare la stessa procedura di "splittaggio" già trattata in questa documentazione per accedere al valore di copertura corretto (fig. 5.40).

```

1049         .then(function (blob) {//aggiunta da noi
1050             var reader = new FileReader();
1051
1052             reader.onload = function () {
1053                 var csvContent = reader.result;
1054
1055                 console.log(csvContent);
1056
1057                 // Dividi il contenuto CSV in righe separate
1058                 var lines = csvContent.split('\n');
1059
1060                 if (lines.length > 1) {
1061                     // Dividi la seconda riga in elementi separati da virgole
1062                     var secondRowElements = lines[1].split(',');
1063                     var terzoRowElements = lines[2].split(',');
1064                     var quartoRowElements = lines[3].split(',');
1065                     var quintoRowElements = lines[4].split(',');
1066                     var sestoRowElements = lines[5].split(',');
1067                     var settimoRowElements = lines[6].split(',');
1068                     var ottavoRowElements = lines[7].split(',');
1069                     var nonoRowElements = lines[8].split(',');
1070
1071                     if (secondRowElements.length > 2) {
1072                         // Estrai il terzo elemento (indice 2) della seconda riga
1073                         var terzoElemento = parseInt(secondRowElements[2]*100);
1074                         var terzoElemento1 = parseInt(terzoRowElements[2]*100);
1075                         var terzoElemento2 = parseInt(quartoRowElements[2]*100);
1076                         var terzoElemento3 = parseInt(quintoRowElements[2]*100);
1077                         var terzoElemento4 = parseInt(sestoRowElements[2]*100);
1078                         var terzoElemento5 = parseInt(settimoRowElements[2]*100);
1079                         var terzoElemento6 = parseInt(ottavoRowElements[2]*100);
1080                         var terzoElemento7 = parseInt(nonoRowElements[2]*100);
1081
1082                         console.log('Terzo elemento della seconda riga:', terzoElemento);
1083                         consoleArea.setValue(`Esito Risultati (percentuale di linee coperte)
1084 Il tuo punteggio: ${terzoElemento}% LOC
1085 Informazioni aggiuntive di copertura:
1086 Il tuo punteggio EvoSuite: ${terzoElemento1}% Branch
1087 Il tuo punteggio EvoSuite: ${terzoElemento2}% Exception
1088 Il tuo punteggio EvoSuite: ${terzoElemento3}% WeakMutation
1089 Il tuo punteggio EvoSuite: ${terzoElemento4}% Output
1090 Il tuo punteggio EvoSuite: ${terzoElemento5}% Method
1091 Il tuo punteggio EvoSuite: ${terzoElemento6}% MethodNoException
1092 Il tuo punteggio EvoSuite: ${terzoElemento7}% CBranch`);
1093
1094
1095
1096
1097
1098             reader.readAsText(blob);
1099         }
1100     .catch(function (error) {
1101         console.log('Si è verificato un errore:', error);
1102     });
1103

```

Figura 5.40: Modifica client editor.html per altri criteri

## 5.8.7 Incremento Portabilità

**Goal:** Incrementare la portabilità del sistema permettendo di installare l'applicazione su **Linux e MacOS**.

Per aumentare la portabilità del sistema è stato creato, in aggiunta al file di installazione "installer.bat", anche un file di installazione compatibile con il sistema operativo macOS e linux denominato **installermac.sh**, presente nella directory */T11-G41-main*.

I passi da compiere per una adeguata installazione sono illustrati nel capitolo 8.

## 5.9 Iterazione 2 - Sprint EXTRA

### 5.9.1 Visualizzazione del test nello storico per ogni turno

In questa sezione si è completata la funzionalità di "Storico" introdotta.

In particolare, si vuole dare la possibilità di poter visualizzare, oltre ai risultati ottenuti, anche i test che l'utente ha sottoposto al sistema durante i turni giocati.

Quello che è stato fatto è:

1. Salvare il test in locale nel VolumeT8.

In particolare, si sfrutta il fatto che il test viene già salvato nel Container **t8\_generazione-1** nella cartella **Tests** raggiungibile al path:

/app/Serv/src/test/java/Tests.

Infatti all'atto del "RUN" e del "SUBMIT", quando è effettuata la richiesta http POST verso il server del task 8 viene salvato il test ai fini di ottenere la copertura del codice di EvoSuite.

Quindi, il lavoro aggiuntivo effettuato è stato quello di andare a prendere il test salvato e copiarlo nel VolumeT8 attraverso questo comando scritto nel file robot\_misurazione\_utente.sh:

```
cp -r -f $PERCORSO/src/test/java  
/Tests/Test$NOME_CLASSE.java  
$PERCORSO_TEST
```

N.B: Il \$PERCORSO\_TEST è passato direttamente nella richiesta http POST che il client (ovvero editor.html del task 6) effettua per salvare il test in locale e ottenere le statistiche ed ha questa struttura:

"/VolumeT8/FolderTreeEvo/Tests/numero\_turno"

dove numero\_turno distingue univocamente il test da salvare.

Di seguito la figura 5.41 mostra il salvataggio di ogni tentativo di gioco in una directory del docker (Volume T8).

✓	FolderTreeEvo	VOLUME	1 day ago	drwxr-xr-x
>	Calcolatrice	VOLUME	1 day ago	drwxr-xr-x
✓	Tests	VOLUME	1 hour ago	drwxr-xr-x
>	10	VOLUME	1 hour ago	drwxr-xr-x
>	2	VOLUME	1 day ago	drwxr-xr-x
>	3	VOLUME	1 day ago	drwxr-xr-x
>	4	VOLUME	1 day ago	drwxr-xr-x
>	5	VOLUME	1 day ago	drwxr-xr-x
✓	8	VOLUME	2 hours ago	drwxr-xr-x
	TestVCardBean.java	VOLUME	1.4 kB	-rw-r--r--
>	9	VOLUME	2 hours ago	drwxr-xr-x
>	VCardBean	VOLUME	1 day ago	drwxr-xr-x

Figura 5.41: Salvataggio in Volume T8 dei test in un turno

## 2. Creare una nuova API per il server del task 8

”prova\_esecuzione\_parametri4.js”, in modo tale che possa gestire nuove richieste http del tipo

”localhost:3080/tests/numero\_turno/nomeTest” come in figura 5.42

Così facendo quando al server arriva una richiesta che inizia con ”tests” (diversa da quella già implementata che inizia con ”api”), è consapevole che deve semplicemente andare a reperire il test presente al path

”/VolumeT8/FolderTreeEvo/Tests/numero\_turno” e fornirlo in risposta alla richiesta.

```

73     else if (req.url.startsWith('/tests/')) {
74         res.setHeader('Access-Control-Allow-Origin', '*');
75
76         const path = req.url.split('/').slice(2); // Rimuovi il primo elemento vuoto e "/tests/"
77
78         if (path.length >= 2) {
79             const numero = path[0];
80             const nome = path[1];
81             console.log(`Valore di "numero": ${numero}, Valore di "nome": ${nome}`);
82
83             const testPath = '/VolumeT8/FolderTreeEvo/Tests/' + numero + '/' + nome + '.java';
84             fs.readFile(testPath, (err, data) => {
85                 if (err) {
86                     res.statusCode = 500;
87                     res.end('Errore nel leggere il file Java');
88                 } else {
89                     res.setHeader('Content-Disposition', 'attachment; filename=yourfile.java');
90                     res.setHeader('Content-Type', 'text/plain');
91                     res.end(data);
92                 }
93             });
94         } else {
95             console.log('Percorso della richiesta non contiene numero e nome');
96         }
97     }

```

Figura 5.42: Modifica server nuova API

3. Aggiungere nuove richieste GET che il client "editor.html" deve inviare alla pressione del bottone di "Storico" al server del task 8 (fig. 5.43).

In questo modo si può ricevere il codice del test sottoposto ad ogni turno giocato e stamparlo in console.

```

693     try{
694         let response = await fetch('http://localhost:3000/tests/' + val.toString() + '/Test' + localStorage.getItem("classe"), { method: 'GET' });
695         if (!response.ok) {
696             throw new Error('Richiesta risultati turno non andata a buon fine');
697         }
698         let data = await response.text();
699         testi += "Codice di test sottoposto al tentativo " + i.toString() + ":" + "\n" + data + "\n\n";
700     } catch (error){
701         console.error('Error:', error);
702     }
703 }
704
705 }
706 consoleArea.setValue(dastampare+testi);
707 console.log(dastampare);

```

Figura 5.43: richiesta get in client editor.html

## 5.10 SVILUPPI FUTURI

Al fine di migliorare l'applicazione si possono effettuare diverse modifiche come aggiungere nuove funzionalità o migliorarne delle ca-

ratteristiche. In particolare l'applicazione necessita di migliorie che riguardano:

- **La sicurezza**, poiché le pagine web mostrano in chiaro i sorgenti javascript che vengono richiamati. Inoltre l'applicazione dà la possibilità di iniettare codice malevolo nel codice di test che si sottopone al sistema senza effettuare alcun controllo.
- **La scalabilità**, poiché attualmente l'applicazione permette di far giocare un solo utente alla volta.  
Si riscontrano problematiche se più utenti provano a giocare contemporaneamente.
- **La robustezza**, poiché l'applicazione con livelli bassi di connettività dà diversi problemi nel sottoporre nuovi test al sistema.

Attualmente, il requisito più grande da soddisfare consiste nell'estendere l'applicazione alla rete globale. Infatti, l'utente ha la possibilità di giocare solo collegandosi all'applicazione tramite la rete locale. Tuttavia, è essenziale ampliare questa funzionalità per consentire anche ad altri utenti di accedere da altri nodi della rete. Inoltre attualmente la funzione di Storico permette di visualizzare i test scritti soltanto nella partita corrente. Se l'utente effettua il Logout non ha la possibilità di visualizzare i tentativi di gioco effettuati in una partita precedente.

Per ampliare questa funzionalità è necessario modificare l'intero Task 4 in quanto non è presente una distinzione tra Game, Round e Turno (sono trattati allo stesso modo).

### 5.10.1 Integrazione nuovi task

Per l'integrazione di nuovi task si possono seguire questi passi per mantenere l'architettura invariata:

- **Integrazione Container:** per procedere con l'integrazione del container all'interno della stessa rete condivisa, si deve utilizzare un file di configurazione "docker-compose.yml" che abbia questa struttura:

```
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
```

Figura 5.44: Integrazione Container

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

```
...
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
...
...
```

Figura 5.45: Integrazione con UI Gateway

- **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” andando ad aggiungere un nuovo blocco ”route”:

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

Figura 5.46: Integrazione con API Gateway

- **Integrazione Installer:** per integrare nei due file di installazione ”installer.bat” e ”installermac.sh” un nuovo container da costruire in docker, bisogna andare ad inserire nelle rispettive list il path che conduce al ”docker-compose.yml” come segue:

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

Figura 5.47: Integrazione Installer.bat

```
...
list=( "T8-G21/Progetto_SAD_GRUPPO21_TASK8/Progetto_def/opt_livelli/Prototipo2.0"
"T1-G11/applicazione/manvsclass" "T23-G1" "T4-G18" "T5-G2/t5"
"T6-G12/T6" "T7-G31/RemoteCCC" "T9-G19/Progetto-SAD-G19-master"
"api_gateway" "ui_gateway")
...
```

Figura 5.48: Integrazione Installermac.sh

# Capitolo 6

## Deployment

Nel diagramma sottostante, viene messa in evidenza la disposizione fisica dei componenti di un sistema software e la loro interazione all'interno di un ambiente di esecuzione.

L'applicativo è ospitato su un server che utilizza il sistema operativo Windows 11. In particolare, il software è in esecuzione in ambiente Docker ed è distribuito su vari container.

Da evidenziare i colori dei vari container:

- **Arancione:** Aggiunti;
- **Rosa:** Modificati;
- **Blu:** Inalterati.

Nel diagramma sono presenti degli elementi a cui prestare attenzione:

- 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 una rete virtuale): l'unico container che è esposto verso l'esterno è il *Gateway* che si occupa di gestire le richieste;
- Il *Volume T8* è condiviso dai container dei Task 1 e 8, la parte del *File System* in comune è quella relativa alle classi e alla copertura generata da *Evosuite*;
- Il *Volume T9* è condiviso dai container dei Task 1 e 9, questo perchè entrambi i Task devono accedere alla stessa porzione di *File System*: quella relativa alle classi e alla copertura generata da *Randoop* e *Emma*.

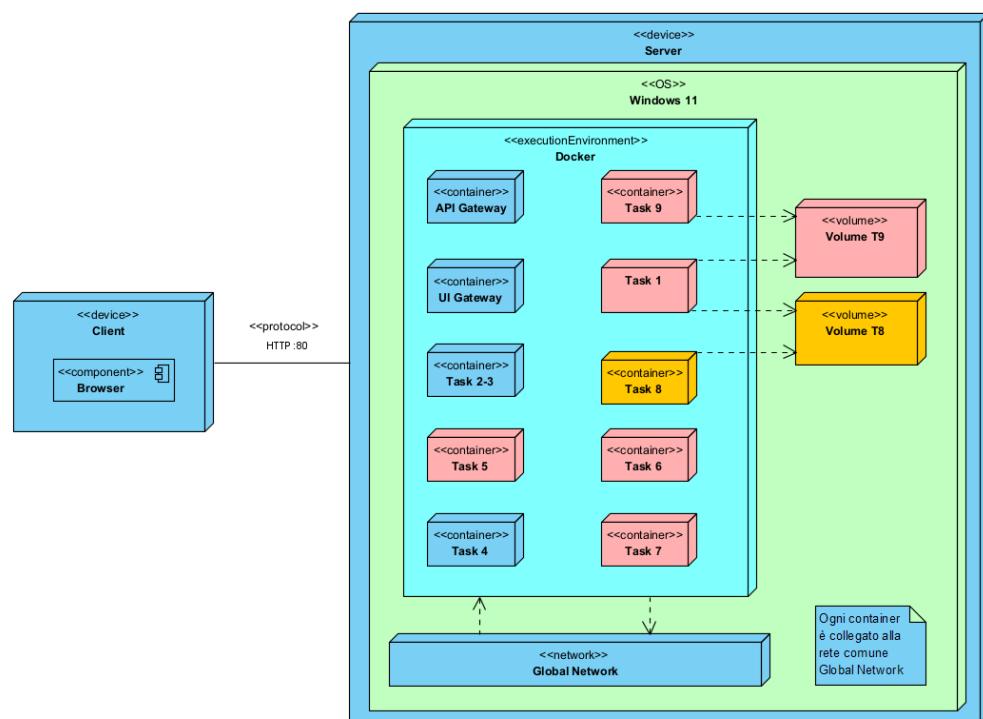


Figura 6.1: Deployment

# Capitolo 7

## Testing

### 7.1 Testing delle API

#### 7.1.1 Richiesta http POST task 8 per statistiche

In questo paragrafo viene testata l'API fornita dal task 8 ma integrata in questo progetto, la quale deve:

- Poder ricevere correttamente il test scritto dall'utente e salvarlo in locale in un file java
- Inviare il file "statistics.csv" in risposta alla richiesta http

Attraverso l'utilizzo di **Postman** 9.6 è stato quindi possibile testare le funzionalità che l'API deve fornire.

A tal fine si sono seguiti i seguenti passi:

1. **Primo passo:** Composizione dell'url della richiesta http POST

**2. Secondo passo:** Configurazione degli **headers** inserendo una nuova intestazione con la chiave 'Content-Type' ed il valore 'text/plain' per specificare che il body contiene dati di tipo testuale come in figura 7.1

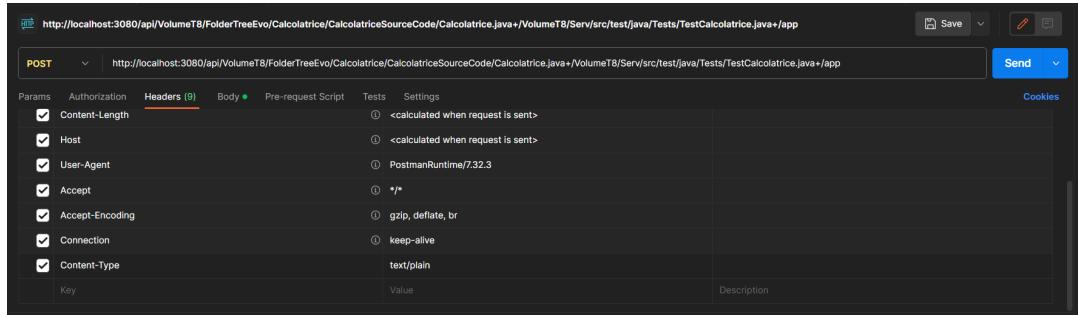


Figura 7.1: Headers Postman

**3. Terzo passo:** inserimento di un body di tipo raw testuale nel quale si è andato ad inserire il test vuoto per la classe "Calcolatrice" scritto in Junit 4

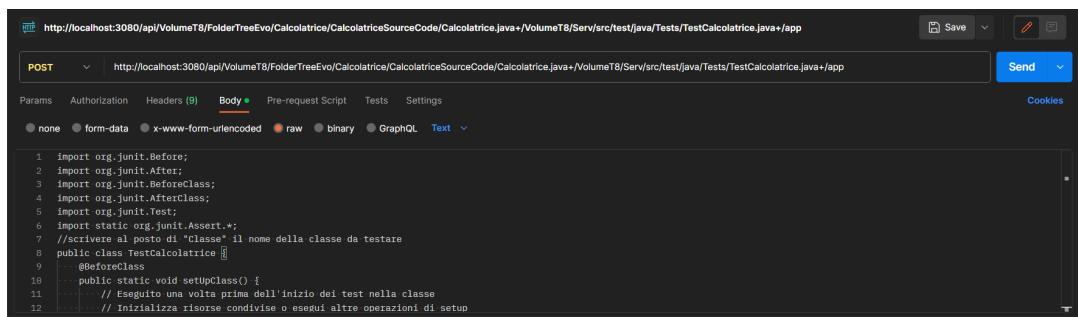


Figura 7.2: Body API Postman

A valle di questi passi il **risultato della richiesta** è stato il seguente:

The screenshot shows a Postman test result for a POST request to `http://localhost:3080/api/VolumeT8/FolderTreeEvo/Calcolatrice/CalcolatriceSourceCode/Calcolatrice.java+VolumeT8/Serv/src/test/java/Tests/TestCalcolatrice.java+app`. The response status is **200 OK**, time is **1m 57.24 s**, size is **544 B**. The response body contains a CSV file named `statistics.csv` with the following content:

```

1 TARGET_CLASS,criterion,Coverage,Total_Goals,Covered_Goals
2 Calcolatrice.LINE,0.0,4,0
3 Calcolatrice.BRANCH,0.0,3,0
4 Calcolatrice.EXCEPTION,1.0,0,0
5 Calcolatrice.WEAKMUTATION,0.0,20,0
6 Calcolatrice.OUTPUT,0.0,3,0
7 Calcolatrice.METHOD,0.0,2,0
8 Calcolatrice.METHODNOEXCEPTION,0.0,2,0
9 Calcolatrice.CBRANCH,0.0,3,0
10

```

Figura 7.3: Risultato test API per ottenere statistiche

Come è possibile notare dalla figura la richiesta è andata a buon fine, avendo restituito codice **"200 OK"** e ricevendo il file "statistics.csv" in risposta correttamente.

### 7.1.2 Richiesta http GET task 8 per test

E' stata testata un'ulteriore API, quella creata nello sprint EXTRA. In particolare è testata la richiesta http GET al task 8 che il client editor.html effettua per ottenere i test salvati per ogni turno giocato. A differenza dell'API della sezione precedente, stavolta non si è utilizzata nessuna configurazione particolare per gli headers e per il body.

Come si evince in figura 7.4 il test ha dato esito positivo e l'API è stata implementata correttamente.

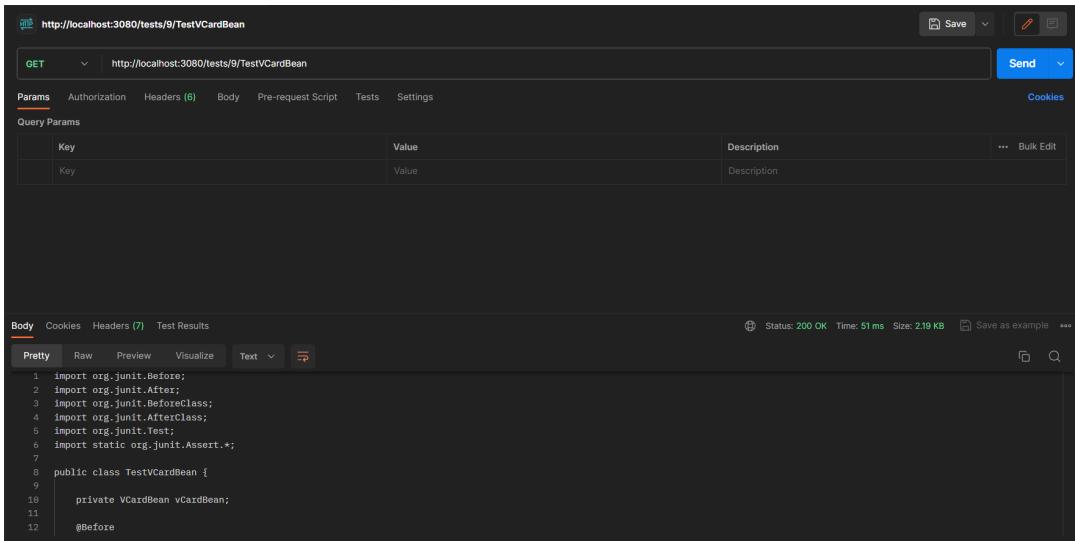


Figura 7.4: Risultato test API per ottenere i test

## 7.2 Testing End to End

L’obiettivo principale è l’integrazione completa di tutti i componenti per garantire il corretto funzionamento dell’applicazione. A questo scopo, è stato condotto il Test End-to-End (E2E), noto anche come Test di Sistema. Questo approccio di testing valuta l’intero flusso dell’applicazione, simulando scenari realistici per verificare che tutti i componenti, tra cui interfaccia utente, servizi di backend, database e comunicazioni di rete, operino come previsto.

Il test E2E mira a convalidare il comportamento complessivo dell’applicazione, inclusi aspetti come funzionalità, affidabilità, prestazioni e sicurezza. Il suo obiettivo finale è individuare eventuali difetti o problemi che potrebbero emergere durante l’interazione tra le diver-

se parti dell'applicazione.

Per condurre il test E2E, sono stati utilizzati strumenti come **Selenium** per il testing automatizzato delle applicazioni web, **JUnit** come framework di testing e **Google Chrome** come browser per l'accesso all'applicazione. I casi di test comprendono flussi legati all'accesso (Login Test) e all'utilizzo dell'editor (Editor Test), ognuno con fasi di configurazione dell'ambiente e di esecuzione dei test veri e propri.

### 7.2.1 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.

### 7.2.2 Login test

#### Setup

Le operazioni coinvolte nella fase di configurazione sono:

- 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 = 10;

@BeforeClass
public static void setDriver() {
    System.setProperty(key:"webdriver.chrome.driver", value:"C:\\\\Users\\\\luix1\\\\Downloads\\\\chromedriver-win64\\\\chromedriver.exe");
}

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

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

Figura 7.5: 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("prova@gmail.com");
    driver.findElement(By.id("password")).sendKeys("Prova123");
    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);
}
```

Figura 7.6: 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 invalidpassword(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("prova@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");
}
```

Figura 7.7: 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 invalidemail(){
    driver.get("http://localhost/login");
    driver.findElement(By.id("email")).sendKeys("sbaglio@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");
}
```

Figura 7.8: Login test Email errata

### 7.2.3 Editor test

#### Setup

Le operazioni effettuate durante la fase di configurazione 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

```

@BeforeClass
public static void setDriver() {
    System.setProperty(key:"webdriver.chrome.driver", value:"c:\\\\Users\\\\luix1\\\\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\\\\luix1\\\\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("prova@gmail.com");
    driver.findElement(By.id("password")).sendKeys("Prova123");
    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();
}

```

Figura 7.9: 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();

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

    File f = new File(pathname:"C:\\\\Users\\\\luix1\\\\Downloads\\\\class.java");

    Thread.sleep(millis:5000);

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

Figura 7.10: 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 nella funzione **”moveToReport()”**:

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);
}
```

Figura 7.11: Editor test Scelta classi e robot

## 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. clicca su pulsate di conferma per confermare il logout
3. verifica che sia stato effettuato il redirect alla pagina di login (/login)

```
@Test
public void logout_main() {
    WebDriverWait wait = new WebDriverWait(driver, timeout);

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

    Alert alert = driver.switchTo().alert();
    alert.accept(); // clicco su conferma per effettuare il logout

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

Figura 7.12: Editor test Logout main

## 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);
}
```

Figura 7.13: 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. inserisce il test da compilare
3. clicca il pulsante di compilazione
4. 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();
    }

    // Trova l'elemento CodeMirror
    WebElement codeMirrorElement = driver.findElement(By.cssSelector("#editor + div"));

    // Inserisci il tuo testo nell'editor CodeMirror
    String code = "import org.junit.Before;\n" +
    "import org.junit.After;\n" +
    "import org.junit.BeforeClass;\n" +
    "import org.junit.AfterClass;\n" +
    "import org.junit.Test;\n" +
    "import static org.junit.Assert.*;\n" +
    "\n" +
    "//scrivere al posto di \"TestClasse\" il nome della classe da testare\n" +
    "public class TestVCardBean {\n" +
    "    @BeforeClass\n" +
    "    public static void setUpClass() {\n" +
    "        // Eseguito una volta prima dell'inizio dei test nella classe\n" +
    "        // Inizializza risorse condivise o esegui altre operazioni di setup\n" +
    "    }\n" +
    "    (...)";
    ((JavascriptExecutor) driver).executeScript("arguments[0].CodeMirror.setValue(arguments[1]);", codeMirrorElement, code);

    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();
    }
}
```

Figura 7.14: Editor test Compilazione classe utente

## Run classe utente

Tale test verifica che la copertura della classe di test scritta dall'utente avvenga correttamente e che appaia correttamente in Console la percentuale di copertura. Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. inserisce il test da runnare
3. clicca il pulsante di Run
4. verifica che sia visibile il risultato della copertura in Console

```

@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();
    }

    // Trova l'elemento CodeMirror
    WebElement codeMirrorElement = driver.findElement(By.cssSelector("#editor + div"));

    // Inserisci il tuo testo nell'editor CodeMirror
    String code = "import org.junit.Before;\n" +
    "import org.junit.After;\n" +
    "import org.junit.BeforeClass;\n" +
    "import org.junit.AfterClass;\n" +
    "import org.junit.Test;\n" +
    "import static org.junit.Assert.*;\n" +
    "\n" +
    "//scrivere al posto di \"TestClasse\" il nome della classe da testare\n" +
    "public class TestVCardBean {\n" +
    "    @BeforeClass\n" +
    "    public static void setUpClass() {\n" +
    "        // Eseguito una volta prima dell'inizio dei test nella classe\n" +
    "        // Inizializza risorse condivise o esegui altre operazioni di setup\n" +
    "    }\n" +
    "    \n";
    ((JavascriptExecutor) driver).executeScript("arguments[0].CodeMirror.setValue(arguments[1]);", codeMirrorElement, code);

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

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

```

Figura 7.15: Editor test Run 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. inserisce il test da sottoporre al sistema
3. clicca il pulsante di Submit
4. verifica che sia visibile l'esito della partita in Confronto risultati

```
@Test
public void submit() {
    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();
    }

    WebElement element = driver.findElement(By.cssSelector("span.cm-def"));

    // Scorrere l'elemento in vista
    JavascriptExecutor executor = (JavascriptExecutor) driver;
    executor.executeScript("arguments[0].scrollIntoView(true);", element);

    // Trova l'elemento CodeMirror
    WebElement codeMirrorElement = driver.findElement(By.cssSelector("#editor + div"));

    // Inserisci il tuo testo nell'editor CodeMirror
    String code = "import org.junit.Before;\n" +
    "import org.junit.After;\n" +
    "import org.junit.BeforeClass;\n" +
    "import org.junit.AfterClass;\n" +
    "import org.junit.Test;\n" +
    "import static org.junit.Assert.*;\n" +
    "\n" +
    "//scrivere al posto di \"TestClasse\" il nome della classe da testare\n" +
    "public class TestCardBean {\n" +
    "...";
    ((JavascriptExecutor) driver).executeScript("arguments[0].CodeMirror.setValue(arguments[1]);", codeMirrorElement, code);

    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();
    }
}
```

Figura 7.16: Editor test Submit della partita

## Storico

Tale test verifica che il tentativo di ricevere lo storico dei turni nella partita sia correttamente processato.

Per fare ciò, il robot effettua le seguenti operazioni:

1. attende che la classe da testare sia caricata
2. inserisce il test da sottoporre al sistema
3. clicca il pulsante di Submit
4. verifica che sia visibile l'esito della partita in Confronto risultati

5. clicca sul pulsante di Storico
6. verifica che sia presente il risultato in Console

```
@Test
public void storico() {
    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();
    }

    // Trova l'elemento CodeMirror
    WebElement codeMirrorElement = driver.findElement(By.cssSelector("#editor + div"));

    // Inserisci il tuo testo nell'editor CodeMirror
    String code = "import org.junit.Before;\n" +
        "import org.junit.After;\n" +
        "import org.junit.BeforeClass;\n" +
        "import org.junit.AfterClass;\n" +
        "import org.junit.Test;\n" +
        "import static org.junit.Assert.*;\n" +
        "...";
    ((JavascriptExecutor) driver).executeScript("arguments[0].CodeMirror.setValue(arguments[1]);", codeMirrorElement, code);

    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();
    }

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

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

Figura 7.17: Editor test Storico

## 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. clicca sul pulsante di conferma del logout
4. verifica che sia stato effettuato il redirect alla pagina di login  
(/login)

```
@Test
public void logout_editor() {
    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("logout")).click();

    Alert alert = driver.switchTo().alert();
    alert.accept(); // clicco su conferma per effettuare il logout

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

Figura 7.18: Editor test Logout editor

## 7.2.4 Risultati

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

### Risultati per Editor Test

```
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 213.2 s --- in com.example.EditorTest
[INFO] Running com.example.LoginTest
```

Figura 7.19: Risultato per Editor Test

### Risultati per Login Test

```
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 6.671 s --- in com.example.LoginTest
[INFO]
```

Figura 7.20: Risultato per Login Test

## Risultati totali

```
[INFO] Results:  
[INFO]  
[INFO] Tests run: 12, Failures: 0, Errors: 0, Skipped: 0  
[INFO]  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO]  
[INFO] -----  
[INFO] Total time: 03:45 min  
[INFO] Finished at: 2023-11-09T18:34:29+01:00  
[INFO] -----
```

Figura 7.21: Risultato finale

# **Capitolo 8**

# **Guida all'installazione e all'utilizzo**

In questo capitolo è riportata la sequenza esatta di passi da seguire, da zero, per effettuare l'installazione del progetto "Integrazione Man vs Automated Testing Tools challenges".

Successivamente all'installazione, è inoltre descritta la procedura da seguire per avviare l'applicazione e utilizzarla correttamente.

## **8.1 Installazione**

### **8.1.1 Passo 1**

Scaricare Docker Desktop (9.4) per il proprio sistema operativo.

N.B. Si potrebbe riscontrare all'avvio di Docker, l'errore

*"Docker desktop - unexpected wsl error"*. In tal caso eseguire nel terminale il comando **wsl - -shutdown** e riavviare la macchina. Se neanche questo dovesse funzionare, si dovrebbe installare WSL (**wsl - -install** sul terminale) o aggiornarlo all'ultima versione.

### 8.1.2 Passo 2

Per Windows - avviare lo script file installer.bat.

Per MacOS - eseguire nella cartella dove è presente il file "installermac.sh" il comando **chmod +x installermac.sh** per renderlo eseguibile, e poi **./installermac.sh** per eseguirlo.

Tale installazione porterà alla:

1. Creazione della rete "global-network" comune a tutti i container
2. Creazione del volume "VolumeT9" comune ai Task 1 e 9 e del "VolumeT8" comune ai Task 1 e 8.
3. Creazione dei singoli container nel Docker desktop.

Alla fine dell'installazione, bisognerà avviare tutti i container che non si presentano attivi, **tranne ui\_gateway**, il quale dovrà essere avviato solo dopo l'attivazione di tutti gli altri.

N.B: il container relativo al Task 9 ("Progetto-SAD-G19-master") si sosponderà autonomamente dopo l'avvio. Esso viene utilizzato solo per "popolare" il volume "VolumeT9" condiviso con il Task 1.

### 8.1.3 Passo 3

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 in Docker Desktop all'interno del terminale del container come in figura 8.1.3
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)
```

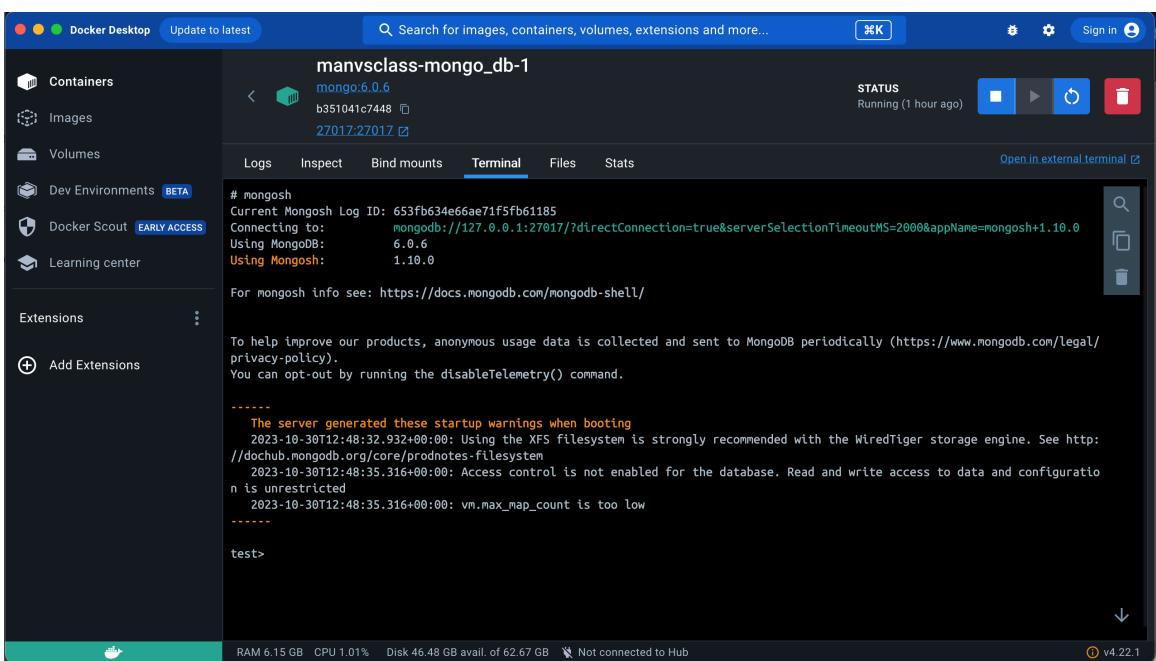


Figura 8.1: Terminale manvsclass-mongo

L'intera applicazione è adesso pienamente configurata e raggiungibile sulla porta :80.

## 8.2 Utilizzo

Una volta effettuata l'installazione e avviati i container, è possibile effettuare la registrazione nella schermata di Login. Prima di poter giocare però, è necessario inserire una classe di test da testare.

Questo upload può farlo solo l'utente amministratore, che deve pertanto registrarsi e poter caricare la classe di test.

Di seguito è illustrato come è possibile fare quanto detto.

### 8.2.1 Registrazione Admin

L'amministratore, se è il primo accesso alla applicazione, si deve registrare nella schermata di registrazione ottenuta inserendo nella barra degli indirizzi questo url:

**`http://localhost/registraAdmin.`**

Apparirà pagina web in figura 8.2 in cui è possibile inserire il proprio nome, cognome, username e password.

## Registrazione Amministratore



Nome

Not

## Cognome

Yet

username

NotYetSad

## Password

• • • • •

[Registrati](#)

Figura 8.2: Registrazione Admin

## 8.2.2 Upload della classe da testare

In seguito alla registrazione, si dovrà caricare la classe da testare. In particolare si deve:

1. Inserire il nome della classe, che **deve essere lo stesso del nome del file .java da caricare**
2. La data di upload
3. La difficoltà della classe da testare
4. La descrizione
5. La categoria indicativa del tipo di classe da testare

Nell'esempio in figura 8.3 si è scelto di caricare una classe di test semplice chiamata Calcolatrice.

Questo progetto presenta due tipi di file d'esempio contenenti due classi da testare:

- **Calcolatrice.java**, una classe piuttosto semplice da testare
- **VCardBean.java**, una classe più difficile

Entrambe si possono trovare nella cartella "Tests" seguendo il path */T11-G41-main/ClassiUT/Tests/**ClassName.java***

## Class upload



**Class name**  
Calcolatrice

**Date**  
30/10/2023

**Difficulty** Beginner 

Select the difficulty level of this item

**Description**  
Classe di test semplice da testare

**Category 1**  
Math

**Category 2**

**Category 3**

Upload your file (.java) :   Calcolatrice.java



Figura 8.3: Upload della classe

E' possibile verificare la corretta esecuzione dell'uploading del file nella sezione "Logs" del container "mansclass-controller-1" come in figura 8.4.

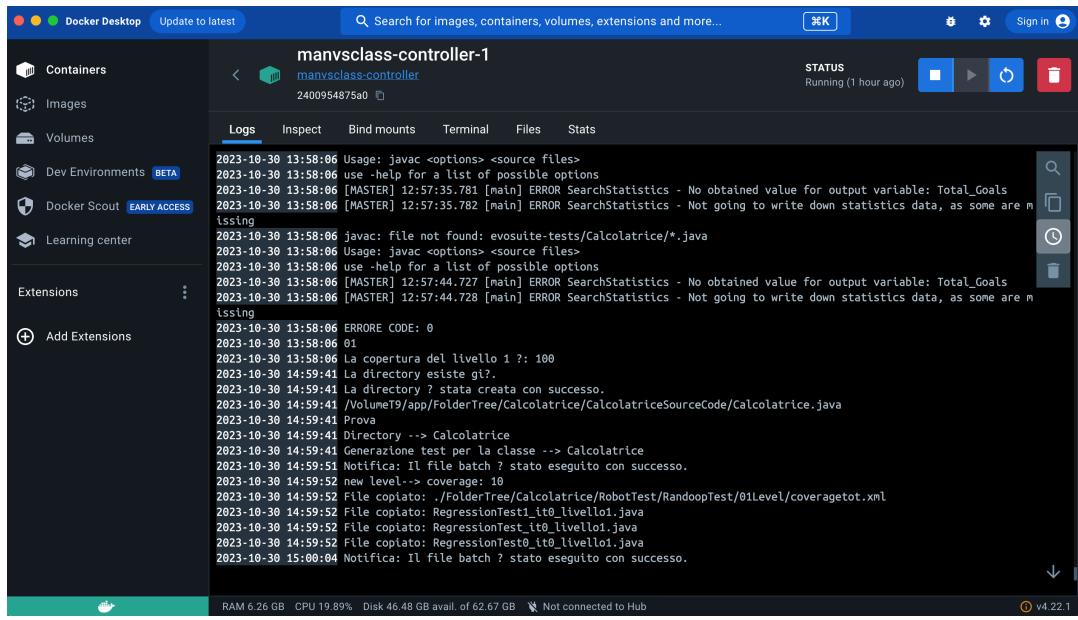


Figura 8.4: Log dell'upload

Al termine dell'upload, se effettuato con successo, si può visualizzare l'effettivo caricamento della classe.

La schermata in figura 8.5 conterrà l'insieme di tutte le classi da testare caricate dall'amministratore.

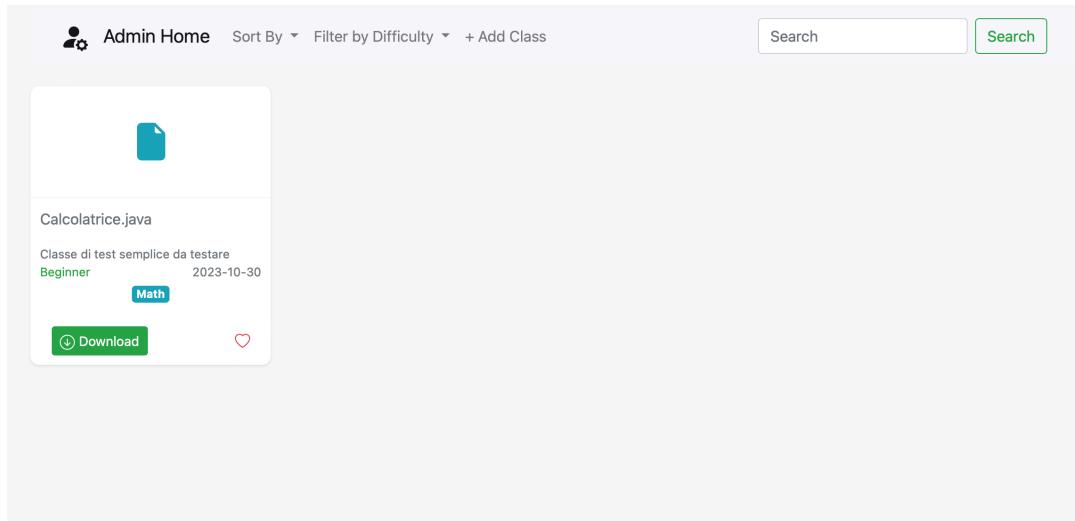


Figura 8.5: Classi caricate

Successivamente, se l'amministratore vuole inserire una nuova classe di test, deve effettuare il Login con le credenziali registrate, immettendo nella barra degli indirizzi l'url **http://localhost/loginAdmin**. Apparirà la schermata in fig.8.6 in cui è possibile inserire i propri dati.

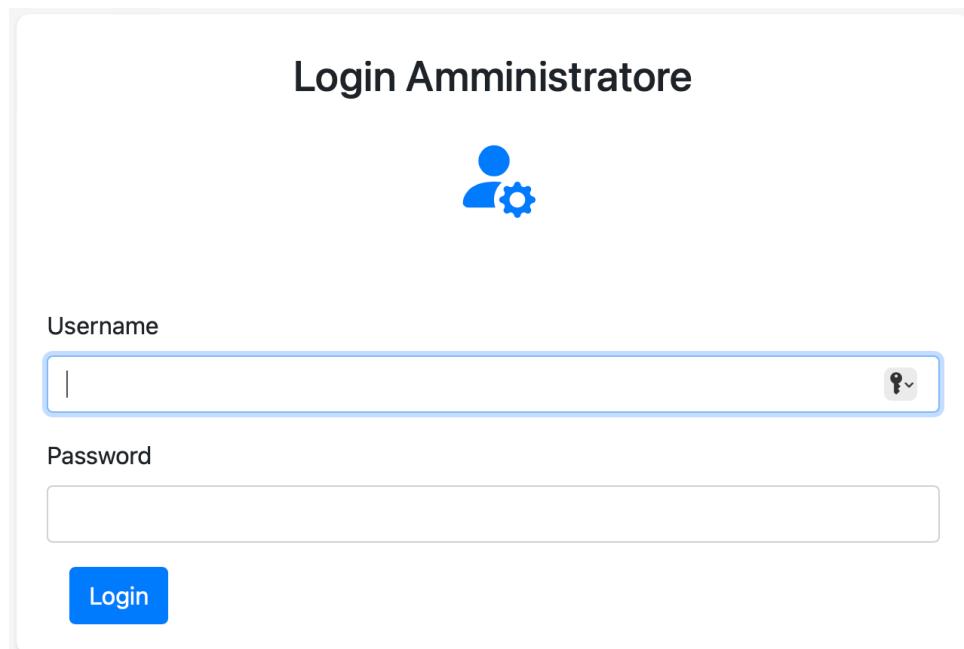


Figura 8.6: Login Admin

Una volta inserite correttamente le classi da testare, l'utente può finalmente scegliere la classe e iniziare a gareggiare contro il robot, questo previa registrazione o login (se è già registrato).

### 8.2.3 Registrazione Utente

Se l'utente è al primo utilizzo dell' applicazione, deve digitare l'url **http://localhost/login** per accedere alla schermata di login, e cliccare sulla dicitura "Non sei ancora registrato? Registrati."

A questo punto è possibile effettuare la registrazione inserendo, come in fig.8.7, il proprio nome, cognome, email, e password.

N.B: La password deve contenere almeno un carattere speciale, una lettera maiuscola, una minuscola, e deve contenere un minimo di 8 caratteri.

# REGISTRAZIONE

Not

Yet

notyetsad@gmail.com

•••••••

•••••••

MSc

Non sono un robot   
reCAPTCHA  
Privacy - Termini

Invia

[Sei già registrato? Accedi.](#)

Figura 8.7: Registrazione utente

Una volta registrato e loggato correttamente, l’utente può scegliere, tra le classi di test caricate dall’amministratore, quale sottoporre al sistema (fig.8.8) e finalmente, iniziare a giocare per provare a battere il robot, e raggiungere un livello di copertura del test maggiore!

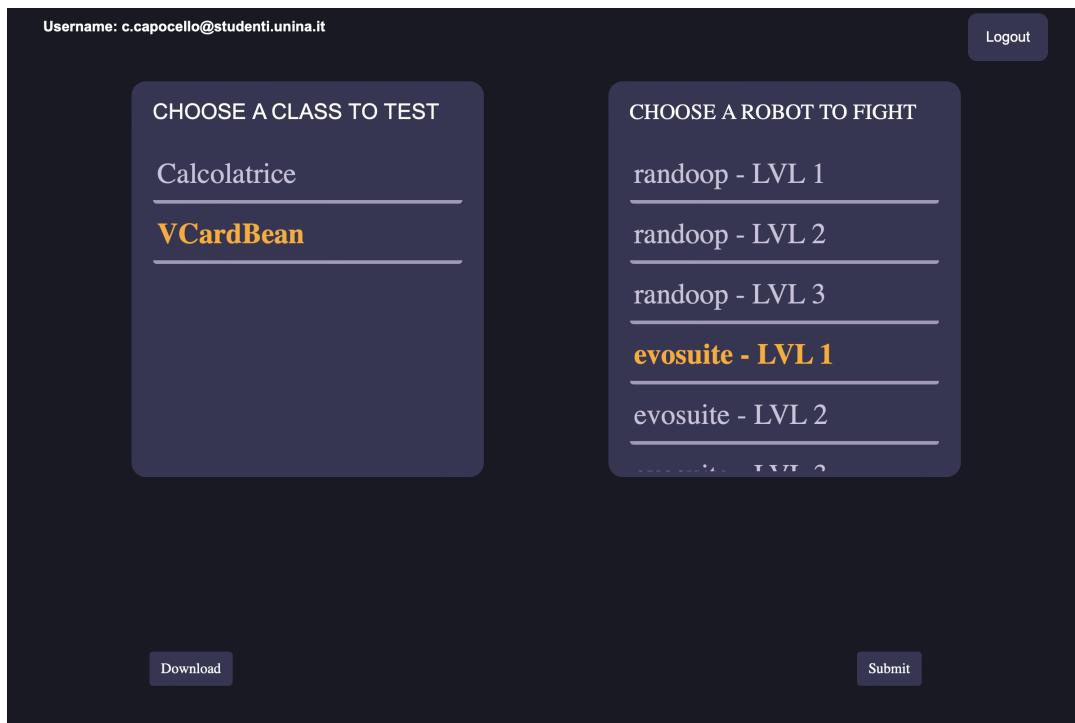


Figura 8.8: Scelta classi e robot da battere

#### 8.2.4 Guida all’interfaccia di editor

Successivamente alla scelta della classe, del livello e del robot contro cui gareggiare, è possibile accedere alla schermata di editor (fig.8.9), in cui come azione preliminare è necessario cambiare il nome della classe **"TestClasse"** con il nome della classe scelta (nell’esempio **"TestVCardBean"**).

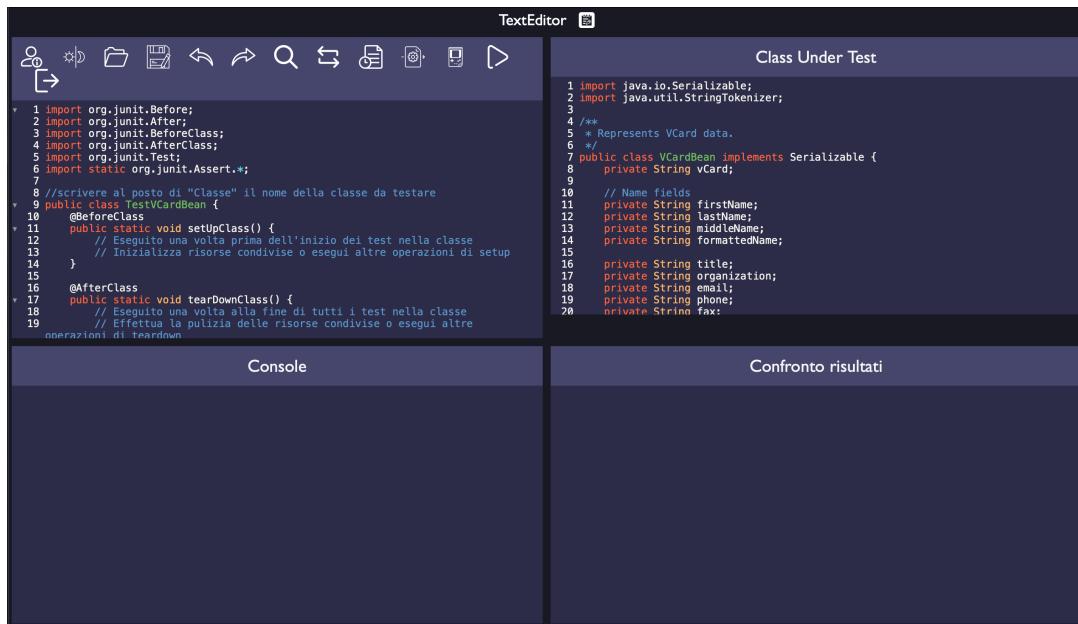


Figura 8.9: Cambio nome classe

E' possibile inoltre cliccare sul tasto **Storico** evidenziato in figura 8.10 con il cerchio rosso, per avere contezza della percentuale di copertura da battere per vincere il round in cui si sta giocando.

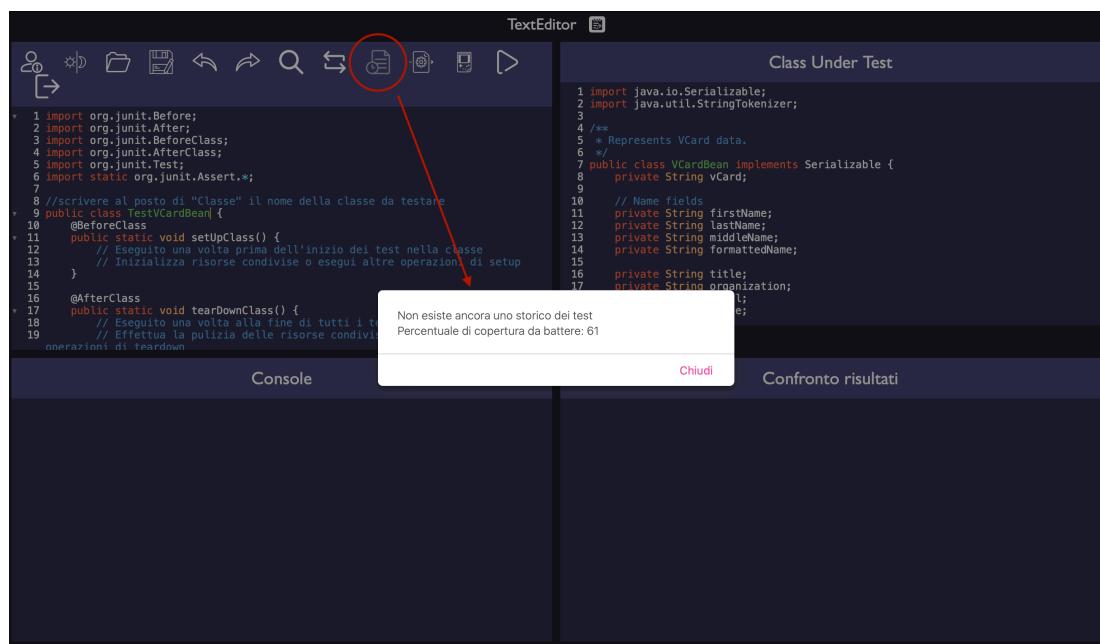


Figura 8.10: Storico primo click

N.B: Il tasto **Storico** permette di ottenere un report dei punteggi del giocatore per ogni turno giocato in una partita e visualizzarlo in Console.

Poichè non è stata sottoposta ancora nessuna classe di test, lo storico dei test risulterà inizialmente vuoto.

## Tasto Run

L'icona cerchiata in figura 8.11 rappresenta il tasto **Run**, che permette di visualizzare il punteggio del giocatore ottenuto dal test scritto per la classe caricata. In particolare in Console verrà visualizzata la percentuale di copertura delle linee di codice (**LOC**), e le informazioni aggiuntive di copertura delle eccezioni, dei Branch e di altri criteri di copertura ottenuti con *EvoSuite*.

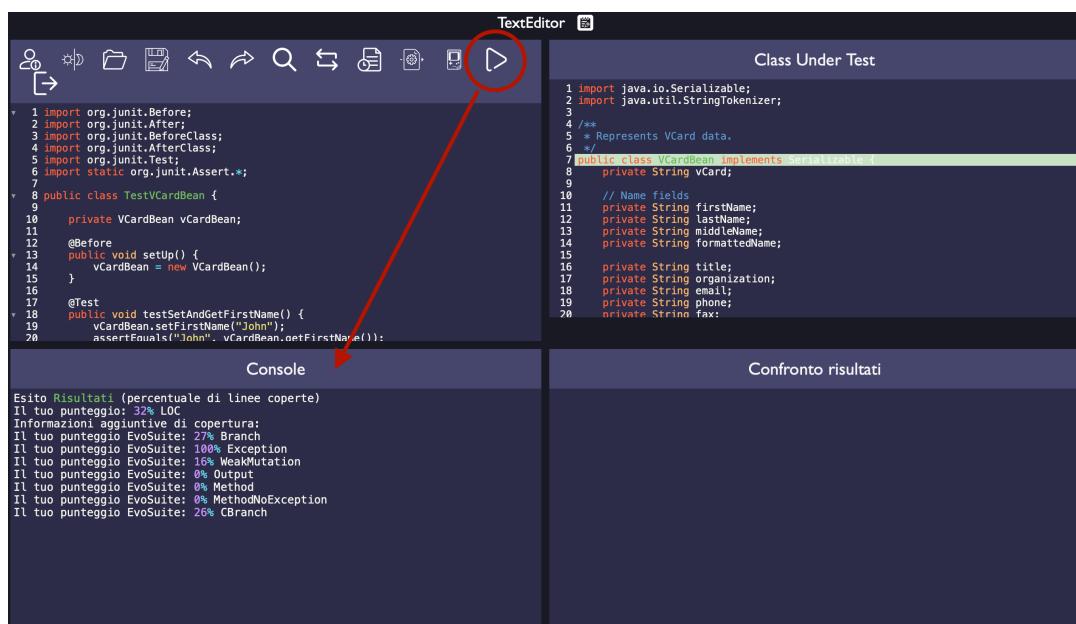


Figura 8.11: Tasto Run

## Tasto Submit

Il tasto **Submit** permette di gareggiare effettivamente contro il robot selezionato e capire se il giocatore ha vinto o perso il turno.

Il criterio scelto per stabilire vittoria o sconfitta è la percentuale di linee di codice coperte, visualizzabile in **Confronto Risultati** come mostrato in figura 8.12.

The screenshot shows the Eclipse IDE interface with several open panes:

- TextEditor**: Shows Java code for a test class `TestVCardBean` and a class `VCardBean`.
- Console**: Displays build logs for a project named "ClientProject" using "code-coverage".
- Class Under Test**: Shows the source code for the `VCardBean` class.
- Confronto risultati**: Shows coverage results comparing EvoSuite (32% LOC), Jacoco (32% LOC), and the robot (61% LOC).

Figura 8.12: Tasto Submit

## Tasto Storico

Il tasto **Storico**, come già detto, permette di visualizzare in Console i tentativi del giocatore nel battere il robot nella partita in corso.

In particolare si può verificare per ogni tentativo:

- La percentuale di copertura da battere

- La percentuale di copertura ottenuta per il test sottoposto
- Il codice di ogni test sottoposto al sistema che ha fornito quella percentuale

Nella figura 8.13 d'esempio, è stato sottoposto un test che ha ottenuto il 0% di copertura di LOC, e successivamente un altro che ha ottenuto il 32%, che non è comunque riuscito a battere il test generato da EvoSuite Lv1.

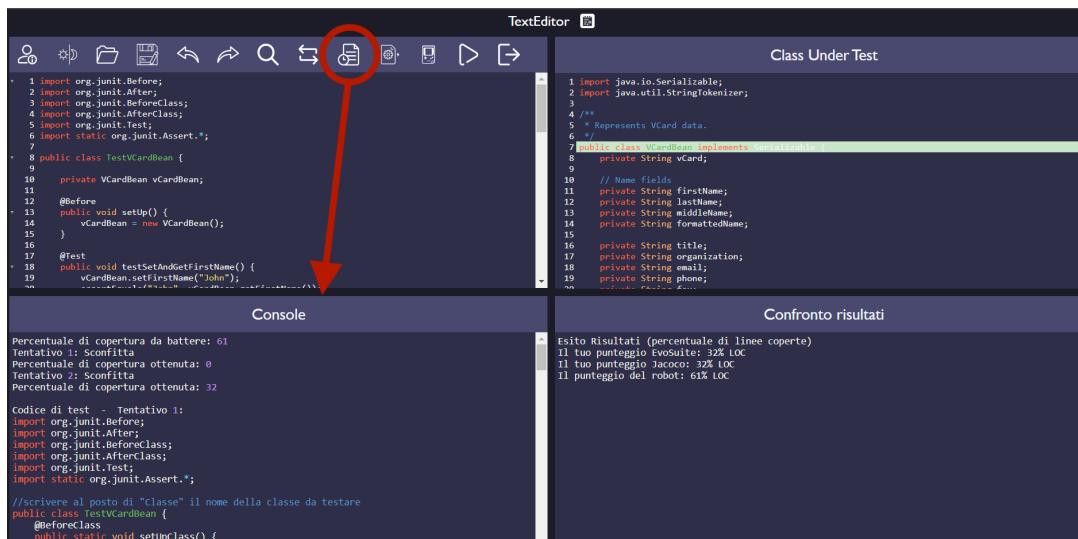


Figura 8.13: Tasto Storico

Con la sottomissione di un test migliorato, è stato possibile ottenere una copertura di LOC del 77%, che ha permesso di battere il robot.

Nello storico pertanto comparirà la Vittoria.

```

1 import org.junit.Before;
2 import org.junit.Test;
3 import static org.junit.Assert.*;
4 
5 public class TestVCardBean {
6     private VCardBean vCardBean;
7 
8     @Before
9     public void setup() {
10         vCardBean = new VCardBean();
11     }
12 
13     @Test
14     public void testSetAndGetFirstName() {
15         vCardBean.setFirstName("John");
16         assertEquals("John", vCardBean.getFirstName());
17     }
18 }

```

Percentuale di copertura da battere: 61  
Tentativo 1: Sconfitta  
Percentuale di copertura ottenuta: 0  
Tentativo 2: Vittoria  
Percentuale di copertura ottenuta: 32  
Tentativo 3: Vittoria  
Percentuale di copertura ottenuta: 77

Codice di test Tentativo 1:  
import org.junit.Before;  
import org.junit.Test;  
import org.junit.BeforeClass;  
import org.junit.AfterClass;  
import org.junit.Test;  
import static org.junit.Assert.\*;

//scrivere al posto di "Classe" il nome della classe da testare  
public class TestVCardBean {

Esito Risultati (percentuale di linee coperte)  
Il tuo punteggio EsoSuite: 77% LOC  
Il tuo punteggio Jacoco: 77% LOC  
Il punteggio del robot: 61% LOC

Figura 8.14: Vittoria

Se ora si volesse effettuare un ulteriore tentativo di gioco sottomettendo un nuovo test, questo non sarà possibile e comparirà l'avviso **"Impossibile effettuare un nuovo tentativo: Hai già vinto!"**.

## Tasto Logout

Per terminare la partita è possibile infine effettuare un **Logout** premendo la apposita icona (cerchiata in rosso in figura 8.15). Comparirà un pop-up di conferma che se premuto, rimanderà alla pagina di Login.

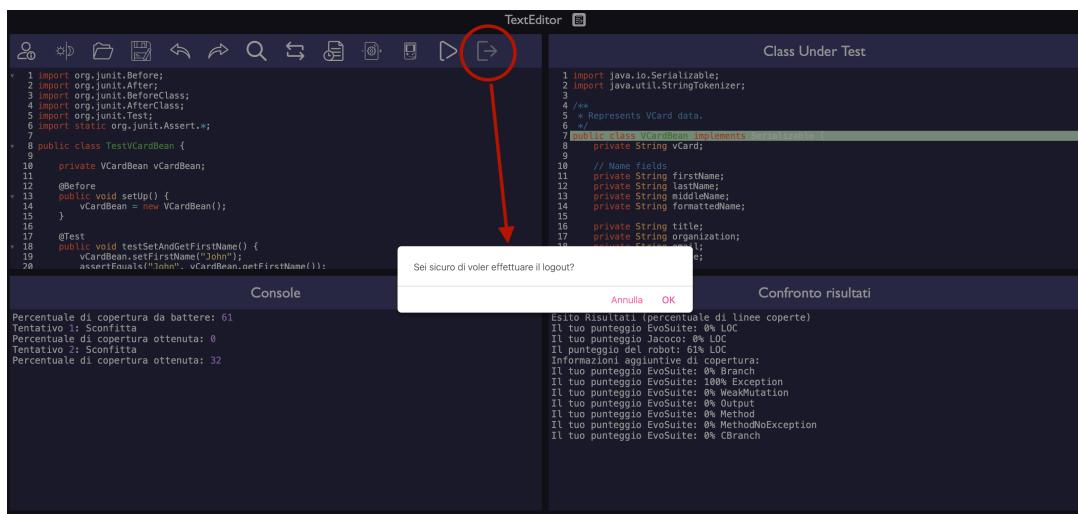


Figura 8.15: Tasto Logout

# Capitolo 9

## Glossario

### 9.1 Robot

Per Robot in questo contesto si intende il tool di generazione automatica di test quale Randoop o Evosuite, contro cui dovrà "gareggiare" lo studente di Software Testing che utilizza questa Applicazione.

### 9.2 Turno

Ogni volta che un giocatore inizia una nuova partita (Game) dovrà scegliere il robot e la classe da testare (round). Ogni tentativo di Submit sarà chiamato **turno** o tentativo.

## 9.3 Giocatore

L'attore “Registered Student” in questa documentazione è chiamato “giocatore” oppure ”Player”.

## 9.4 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità. Docker raccoglie il software in unità standardizzate chiamate **container** che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime. Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.

L'utilizzo di Docker in questa applicazione è stato necessario per l'integrazione e la comunicazione di task implementati con diverse tecnologie.

## 9.5 Trello

Trello è una piattaforma di gestione delle attività e dei progetti basata su schede e bacheche virtuali. Permette ai gruppi di gestire visivamente qualsiasi tipo di progetto, flusso di lavoro o monitoraggio dei task.

Sono sufficienti BACHECHE, LISTE e SCHEDE per avere una visione chiara di chi fa cosa e cosa bisogna fare.

BACHECHE: indipendentemente dal progetto o dalla procedura, le bacheche mantengono i task organizzati e fanno procedere il lavoro.

La schermata è fatta in modo tale da poter tenere sott'occhio sia i task completati che quelli da portare a termine.

LISTE: Le liste servono a organizzare e categorizzare le schede all'interno di un progetto o di un flusso di lavoro. Le schede vengono posizionate all'interno delle liste in base al loro stato o alla fase del processo in cui si trovano

SCHEDE: le schede rappresentano i task e contengono tutte le informazioni necessarie per completare il lavoro. Mentre si procede nel lavoro, è possibile spostare le schede da una lista ad un'altra per rendere visibile il loro stato. È possibile, inoltre, assegnare membri, aggiungere date di scadenza, lasciare commenti e altro.

Trello è noto per la sua interfaccia user-friendly e la sua flessibilità, che consente agli utenti di personalizzare il sistema in base alle proprie esigenze. È utilizzato in una varietà di contesti, come la gestione dei progetti, la pianificazione delle attività personali, il tracciamento delle attività di sviluppo software, la gestione delle operazioni aziendali e molto altro.

## 9.6 Postman

Postman è un'applicazione utilizzata nel campo dello sviluppo software per semplificare il processo di sviluppo, test e documentazione delle API (Interfacce di Programmazione delle Applicazioni). Le API consentono a diversi software e servizi di comunicare tra loro, consentendo lo scambio di dati e funzionalità.

Postman è ampiamente utilizzato da sviluppatori, team di sviluppo, tester di delle API per semplificare e migliorare il processo di sviluppo, test e gestione delle stesse. È disponibile come applicazione desktop e offre anche una versione web, rendendo facile l'accesso da qualsiasi luogo.

## 9.7 Miro

Miro è una piattaforma di collaborazione online che offre una lavagna virtuale e strumenti di collaborazione per il lavoro di squadra. Si tratta di uno strumento utilizzato per la creazione e la condivisione di diagrammi, mappe concettuali, schede di lavoro, wireframe, e altri tipi di contenuti visivi. La piattaforma è progettata per facilitare la comunicazione e la collaborazione tra membri di un team, anche se si trovano in posizioni geografiche diverse.