



UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

Scuola Politecnica e delle Scienze di base

Corso di Laurea Magistrale in Ingegneria Informatica

Documentazione di sviluppo SAD:

Analisi e progettazione

Task T5 - Gruppo 2

Corso di Software Architecture Design

Prof.ssa Anna Rita Fasolino

A.A. 2022-2023

Alessandro Acerola M63001398

Giorgio Casolaro M63001404

Michele De Fazio M63001440

Lorenzo Esposito M63001417

Armando Zevola M63001514

Indice

1	Introduzione	5
1.1	Traccia	5
1.2	Metodologie di sviluppo.....	5
1.2.1	Daily Scrum	5
1.2.2	Sprint Backlog.....	5
1.2.3	Strumenti utilizzati.....	6
2	Analisi del dominio.....	8
2.1	Analisi preliminare	8
2.2	Specifiche dei requisiti	9
2.2.1	Storie utente	9
2.2.2	Requisiti funzionali	9
2.2.3	Requisiti non funzionali	10
2.2.4	Scenari di funzionamento.....	10
2.2.5	Definizione dei casi d'uso.....	12
2.3	Diagrammi di sequenza.....	17
3	Documentazione di progetto	19
3.1	Introduzione al contesto	19
3.1.1	Diagramma di contesto	19
3.2	Tecnologie e ambiente di sviluppo	20
3.2.1	Spring Boot	20
3.2.2	Thymeleaf	20
3.2.3	Bootstrap	20
3.2.4	Maven.....	21
3.2.5	Estensioni – Spring Boot Tools e Spring Boot Dashboard	21
3.3	Diagramma di sequenza MVC	21
3.4	Diagramma dei deployment	23

3.5	Diagramma dei package	24
3.6	Diagramma delle classi	25
3.7	Diagramma dei componenti	26
3.8	Considerazioni sull'implementazione	31
4	Guida all'installazione e all'utilizzo del software.....	32
4.1	Docker e WSL	32
4.2	Istruzioni per l'installazione	33
4.3	Linee guida per l'utilizzo	35
4.4	Diagramma di Installation View.....	38
5	Testing.....	40
5.1	Selenium.....	40
5.2	Configurazione Selenium con Spring Boot.....	41

Abstract

Il seguente documento serve da riferimento per mostrare gli sviluppi del progetto assegnato al gruppo 2, con task T5, dall'analisi delle storie utente fino alla compilazione del codice, passando per la progettazione dei casi d'uso e degli scenari. Verrà inoltre mostrata, passo per passo, l'interfaccia grafica ed eventuali suoi aggiornamenti, il tutto nelle differenti iterazioni che porteranno al progetto finale. L'utilizzo di una metodologia AGILE permetterà lo sviluppo in maniera progressiva del progetto nel suo complesso ed è per questo che saranno necessarie diverse iterazioni per il risultato definitivo.

1 Introduzione

1.1 Traccia

Il giocatore (dopo essersi autenticato) avvia una nuova partita del Primo Scenario, l'applicazione gli mostra un elenco di classi da testare ed un elenco di Robot disponibili, il giocatore sceglie la classe ed il Robot contro cui confrontarsi. A questo punto il sistema crea la partita con tutte le scelte fatte, le associa un IdPartita, e la salva. Successivamente l'applicazione avvia l'ambiente di editing in cui visualizza la classe da testare e gli offre una finestra in cui può scrivere la classe di test.

1.2 Metodologie di sviluppo

Il primo passo per la realizzazione del task assegnato è stato quello di introdurre la metodologia **AGILE** basata su Daily SCRUM e Sprint Backlog. La riunione iniziale del team ha avuto come obiettivo proprio l'identificazione dello Sprint Backlog, analizzando preliminarmente la traccia assegnata per gestire eventuali scadenze.

1.2.1 Daily Scrum

La necessità primaria è stata quella di comprendere e analizzare il **dominio** del problema da affrontare con le difficoltà da superare nel processo di realizzazione del task. Al fine di portare a termine questo processo di analisi, sono state organizzate varie **riunioni** con cadenza di 1-2 giorni, per dare una prima forma di organizzazione al team.

Gli incontri si sono tenuti in doppia modalità: in presenza e in smartworking. In particolare, si è preferito riservare le riunioni in presenza per discutere delle problematiche principali, e quelle a distanza per fare il punto della situazione e rilevare lo stato dell'arte.

1.2.2 Sprint Backlog

Le riunioni organizzative sono state utili per dare una forma agli **Sprint Backlog**, realizzare una prima suddivisione dei task tra i membri del team e prefissare le **scadenze** per il completamento dei vari obiettivi, basandosi sulle date delle review prefissate dai product owner.

Difatti, le deadline sono state impostate ogni due settimane nel periodo di marzo-giugno e, al termine di questo, con l'avvicinarsi della data di presentazione del progetto, è stato ritenuto necessario prevederne una ogni settimana, per **intensificare il lavoro e aumentare la produttività** per il raffinamento degli artefatti.

1.2.3 Strumenti utilizzati

Microsoft Teams

L'applicazione Microsoft Teams ha permesso al team sia di **comunicare** nelle fasi di sviluppo degli artefatti tramite **videochiamate** di gruppo, sia di utilizzare la chat del canale appositamente creato per lo scambio rapido di file e messaggi.

Notion

Per la realizzazione dello Sprint Backlog è stato utilizzato il software Notion, risultato utile al fine di **calendarizzare riunioni**, segnalare aggiornamenti sui **progressi** riguardanti lo sviluppo del task e mettere a disposizione una repository condivisa per l'**archiviazione** di tutte le tipologie di file inerenti al progetto. In Figura 1.1 viene mostrata la sezione **Avanzamenti** nella quale si tiene traccia dello stato dei vari obiettivi raggiunti o ancora da raggiungere

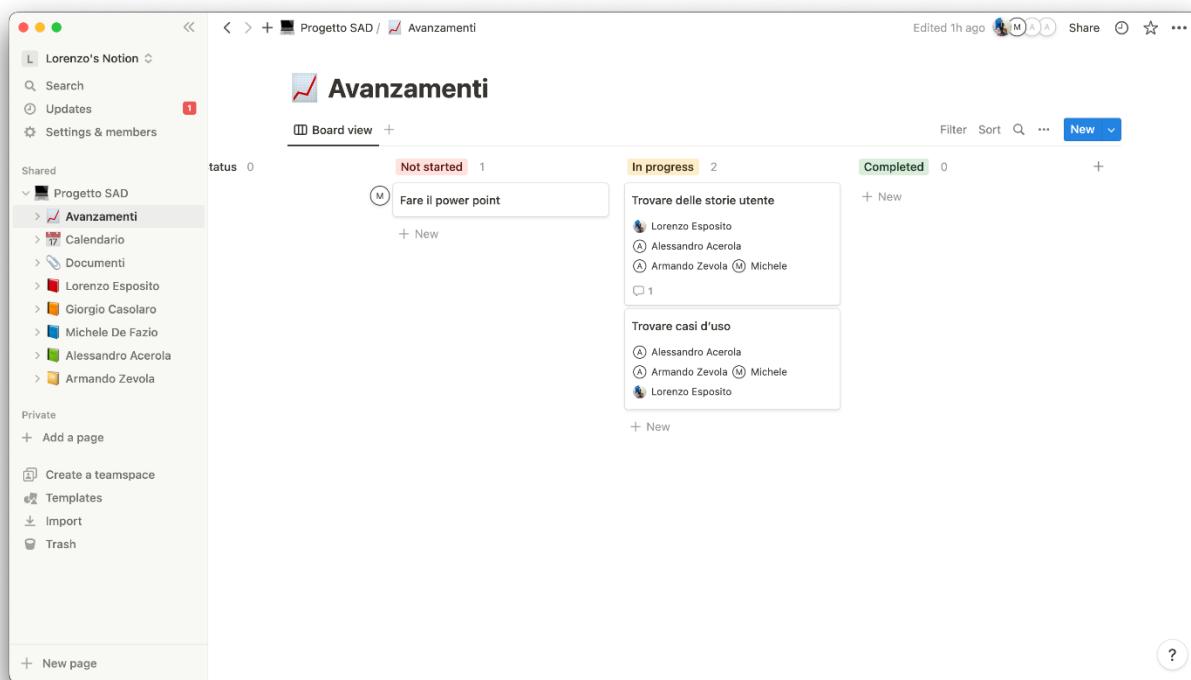


Figura 1.1 – Schermata di Notion

GitHub

Per poter condividere in maniera efficiente il codice e realizzare uno **sviluppo incrementale** suddiviso in più step è stato impiegato il servizio di hosting GitHub, il quale ha permesso al team di eseguire un lavoro di gruppo sul codice e di ottimizzare l'impiego della tecnica di **pair programming**.

Visual Paradigm

Il tool di modellazione Visual Paradigm è stato d'aiuto al team nella fase di **progettazione** e di definizione dei **requisiti** e **casi d'uso**, agevolando la creazione dei vari **diagrammi UML**, riportati nelle apposite sezioni all'interno del documento.

Figma

L'applicazione Figma è stata impiegata per realizzare l'**identità grafica** dell'applicativo: dall'ideazione dei **mock-up** alla costruzione di una paletta di colori, è stato utile nel corso delle varie iterazioni al fine di fornire un riferimento grafico sul quale sviluppare il codice delle GUI. Inoltre, l'impiego di un plugin integrato ha permesso al team di accelerare lo sviluppo, autogenerando parte del codice CSS necessario alla realizzazione del front-end dell'applicativo.

Visual Studio Code

Da menzionare l'editor Visual Studio Code, utilizzato da tutto il team per **scrivere e compilare il codice** prodotto, grazie soprattutto all'introduzione di diverse estensioni per ottenere il risultato richiesto; in primis, il framework **Spring Boot** è stato facilmente integrato grazie all'estensione “*Spring Boot Dashboard*”, utile al fine di realizzare i requisiti definiti nelle fasi iniziali. Inoltre, l'integrazione con l'estensione di GitHub ha permesso di coordinare i vari branch di sviluppo direttamente dall'ambiente di editing del codice.

The screenshot shows the Visual Studio Code interface with the "Spring Boot Dashboard" extension active. The left sidebar displays the project structure with a "SPRING BOOT DASHBOARD" folder containing "APPS" (with "t5" selected), "BEANS" (with "t5" selected), and "ENDPOINT MAPPINGS" (listing endpoints like /editor [GET], /login [GET], /main [GET], /report [GET], and /sendVariable [POST]). The main editor area shows the code for T5Application.java:

```
t5 > src > main > java > com > g2 > t5 > T5Application.java
1 package com.g2.t5;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class T5Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(primarySource:T5Application.class, args);
11     }
12 }
13 }
```

The bottom status bar indicates "Riga 1, colonna 1 Dimensione tabulazione: 4 UTF-8 LF (Java)".

Figura 1.2 – Spring Boot Dashboard in VS Code

2 Analisi del dominio

2.1 Analisi preliminare

L'analisi preliminare del dominio ha permesso di individuare i **requisiti funzionali e non funzionali**, essenziali al fine di specializzare il task assegnato nella realizzazione dei vari **componenti** dell'architettura, a loro volta suddivisi in vari blocchi. Nel caso specifico, è stato fondamentale considerare l'**architettura** di base fornita dal product owner per l'identificazione degli attori e della posizione del task assegnato all'interno di essa. In Figura 2.1 viene riportato lo schema basato sui componenti dell'architettura.

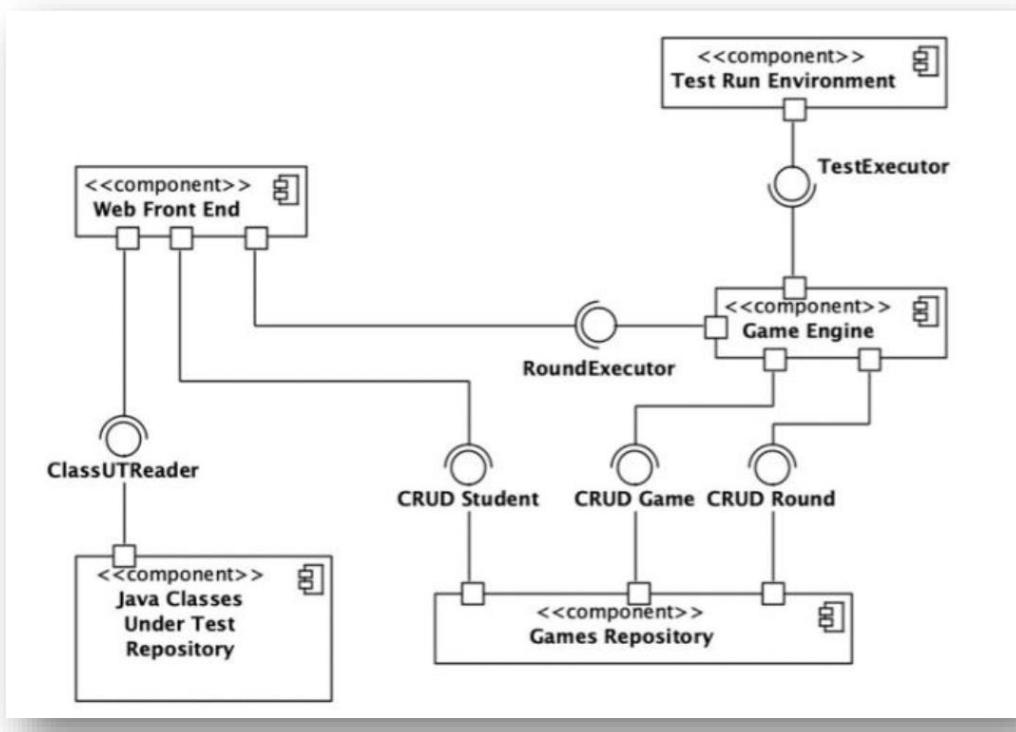


Figura 2.1 – Architettura di base fornita dal product owner

Grazie allo schema appena visto, è stata individuata la posizione del task tra i componenti “*Web Front End*” e “*Game Engine*”, immaginando due attori principali: l'**utente** e l'**amministratore** del sistema.

Attore	Descrizione	Sinonimi
Studente	Colui che gioca la partita	Utente, Giocatore
Amministratore	Colui che gestisce il repository e i robot del gioco	Gestore del sistema

2.2 Specifica dei requisiti

Per arrivare alla specifica dei requisiti si è partito dalle **user stories**, seguendo il pattern “as (actor) i want to (action) so that (change of state)”; in questo modo è possibile individuare una funzionalità del sistema dal punto di vista dell’attore. Successivamente, sono stati esplosi i requisiti nel dettaglio, suddividendoli in requisiti funzionali e non, per poi, infine, ricavare i casi d’uso.

2.2.1 Storie utente

In Figura 2.2 sono mostrate le storie utente scritte immaginando un product backlog con post-it su lavagna.



Figura 2.2 – Storie utente in product backlog

2.2.2 Requisiti funzionali

I requisiti funzionali sono fondamentali per la **costruzione dei casi d’uso**. A partire dalle storie utente e dalla traccia, sono stati estratti i seguenti requisiti:

- R1) Lo studente deve poter creare una partita.
- R2) Lo studente deve poter accedere nel sistema di gioco mediante login.
- R3) Lo studente deve poter selezionare una classe con cui giocare dopo essersi autenticato.
- R4) Lo studente deve poter selezionare un robot contro cui competere dopo essersi autenticato.
- R5) Lo studente ha la possibilità di salvare la partita, la quale verrà identificata mediante l’ausilio di un ID.

- R6) Lo studente per poter avviare una partita deve aver selezionato un robot e una classe.
- R7) Lo studente deve poter avviare l'editor una volta creata una nuova partita.
- R8) L'utente se non ricorda la password deve poter recuperare le proprie credenziali.

Nota: Si definisce come **Utente** l'utilizzatore il quale non ha effettuato il login, operazione necessaria al fine di essere riconosciuto come **Studente**. Inoltre, è importante sottolineare che i requisiti R2 ed R8 sono frutto della collaborazione con il task T2-3, poiché per costruire delle GUI per il login è stato strettamente necessario impiegare almeno i requisiti di base per la costruzione di una user interface, che fornisce all'utente almeno le funzioni di base per poter impiegare quanto sviluppato dai gruppi responsabili del suddetto task.

2.2.3 Requisiti non funzionali

I requisiti non funzionali permettono di **definire** quelle **caratteristiche** del sistema che non si traducono in un caso d'uso ma che ne influenzano le proprietà. Di seguito vengono riportati i requisiti individuati:

1. Il sistema deve garantire la consistenza dei dati e che gli identificativi siano univoci.
2. Il sistema deve garantire un buon grado di usabilità per renderlo facilmente fruibile dall'utente.
3. Il sistema deve garantire un buon grado di sicurezza nella gestione delle credenziali dell'utente.

2.2.4 Scenari di funzionamento

Durante la stesura degli scenari di funzionamento, sono stati individuati, a partire dai requisiti funzionali, due **scenari di base**: il primo, riguardante la **selezione** (da parte dello studente) **dell'avversario e della classe** con cui giocare, ed il secondo riguardante il **login** dell'utente per l'identificazione come studente.

2.2.4.1 Scenario uno – selezione

- Portata: Applicazione gioco sul testing
- Livello: Obiettivo utente
- Attore principale: Studente
- Parti interessate:
 1. Studente – vuole scegliere la classe da testare, vuole scegliere il robot con cui combattere (il suo avversario)
 2. Archivio Partite – vuole la consistenza dell'istanza partita e che la partita sia identificata
- Precondizioni: Lo studente è autenticato e identificato

- Garanzie di successo: La partita viene salvata
- **Scenario principale di successo (flusso base):**
 1. Lo studente, dopo essersi identificato, visualizza la collezione di classi testabili e robot disponibili.
 2. Il sistema richiede l'elenco delle classi e dei robot e li mostra allo studente
 3. Lo studente seleziona la classe e il robot che gli interessa testare
 4. Il sistema mostra un riepilogo delle scelte
 5. Lo studente conferma la sua scelta
 6. Il sistema salva la partita e la identifica; infine, apre l'editor Java per avviare il gioco
- **Flussi alternativi**

1.e Lo studente non conferma

1. Il sistema riporta lo studente all'elenco delle classi e dei robot
2. Lo studente seleziona la classe e il robot che gli interessa testare
3. Il sistema mostra un riepilogo delle scelte
4. Lo studente conferma la sua scelta
5. Il sistema salva la partita e la identifica; infine, apre l'editor Java per avviare il gioco

2.e Lo studente conferma la sua scelta senza aver selezionato nessuna classe e/o robot

1. Il sistema richiede allo studente di effettuare una selezione valida
2. Lo studente procede come da flusso base

2.2.4.2 Scenario due – login

- Portata: Applicazione gioco sul testing
- Livello: Obiettivo utente
- Attore principale: Studente
- Parti interessate:
 1. Studente – vuole autenticarsi tramite il portale
- Precondizioni: Lo studente non è autenticato
- Garanzie di successo: L'utente ha effettuato l'accesso
- **Scenario principale di successo (flusso base):**
 1. L'utente inserisce l'username e la password.
 2. Il sistema verifica la validità delle credenziali e permette l'accesso.
 3. L'utente risulta autenticato e riconosciuto come studente.
- **Flussi alternativi**

1.f Username e password non sono validi

1. L'utente inserisce username e password.
2. Il sistema verifica la validità delle credenziali e nega l'accesso.
3. L'utente visualizza l'errore e riesegue il flusso base.

2.2.5 Definizione dei casi d'uso

Di seguito viene riportata la rappresentazione tabellare e i rispettivi grafici in UML dei due casi d'uso, implementati a partire dai requisiti funzionali e dagli scenari con i flussi base e alternativi.

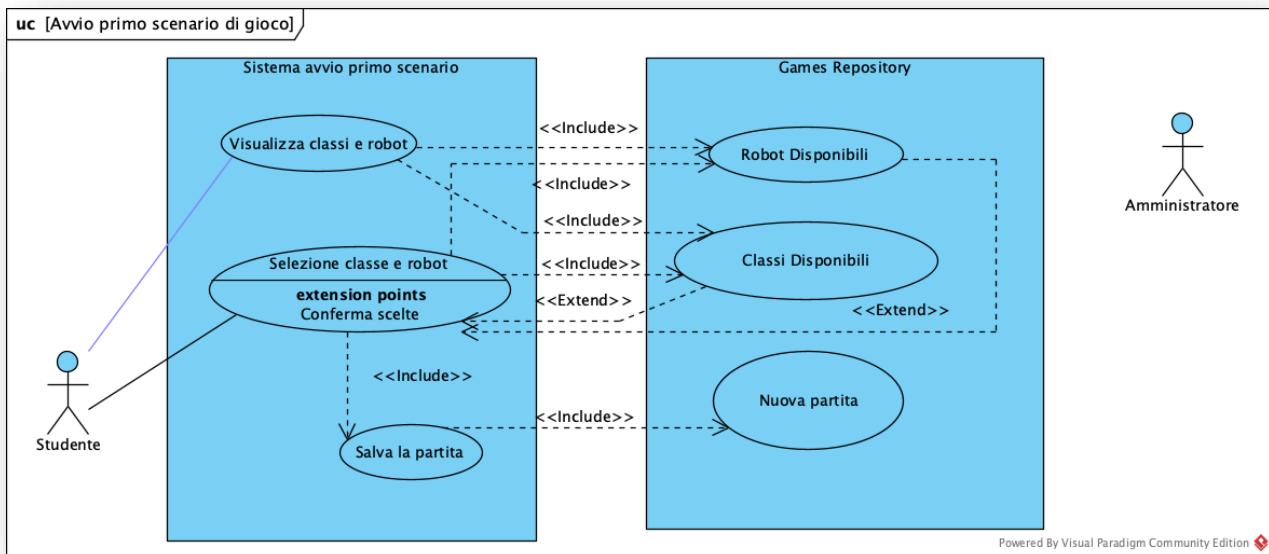


Figura 2.3 – Caso d'uso

Nel caso d'uso “Avvio primo scenario di gioco”, sono presenti due attori:

- Lo **studente**, nonché l'attore principale.
- L'**amministratore**, un attore fuori scena che beneficia in maniera indiretta delle azioni dello studente.

All'interno del primo sistema, come è possibile osservare dalla Figura 2.3, è possibile avviare due azioni: la **visualizzazione** delle classi e dei robot e la **selezione** di una classe e di un robot. Per quanto concerne la visualizzazione, è possibile interfacciarsi, mediante degli **<<include>>**, con il Games Repository, che fornirà le classi e i robot disponibili. Similmente a quanto visto precedentemente per la selezione delle classi, oltre agli **<<include>>** appena citati si trovano degli **<<extend>>** che, nel caso di conferma delle scelte, permettono di salvare una nuova partita con l'assegnazione di un ID. Di seguito viene riportata nel dettaglio la specifica del caso d'uso con la rappresentazione tabellare:

- 1) Visualizza classi e robot:

Attore Primario	Studente
-----------------	----------

Descrizione	Lo studente visualizza le classi e i robot disponibili.
Pre-condizione	Il giocatore deve essere autenticato.
Flusso Principale	1) Il giocatore richiede l'elenco delle classi e dei robot.
Post Condizione	Il sistema restituisce gli elenchi.

2) Robot disponibili:

Attore Primario	Studente
Descrizione	Lo studente, mediante <<include>>, richiede la lista dei robot disponibile al Games Repository.
Pre-condizioni	L'utente deve essere loggato e deve aver richiesto l'elenco dei robot.
Flusso Principale	1) Il giocatore richiede l'elenco dei robot. 2) Il giocatore visualizza l'elenco dei robot.
Post-Condizioni	Il sistema fornisce l'elenco dei robot presenti nel file system.

3) Classi disponibili:

Attore Primario	Studente
Descrizione	Lo studente, mediante <<include>>, richiede la lista delle classi disponibili al Games Repository.
Pre-condizioni	L'utente deve essere loggato e deve aver richiesto l'elenco delle classi.
Flusso Principale	1) Il giocatore richiede l'elenco delle classi. 2) Il giocatore visualizza l'elenco delle classi.
Post-Condizioni	Il sistema fornisce l'elenco delle classi presenti nel filesystem.

4) Selezione classe e robot:

Attore Primario	Studente

Descrizione	Lo studente, mediante <<include>>, richiede la lista delle classi e dei robot disponibili al Games Repository ed effettua una scelta da confermare mediante l'impiego dell'<<extend>>.
Pre-condizioni	L'utente deve essere loggato e deve aver richiesto l'elenco delle classi.
Flusso Principale	<ol style="list-style-type: none"> 1) Il giocatore richiede l'elenco delle classi e dei robot. 2) Il giocatore visualizza l'elenco delle classi e dei robot. 3) Il giocatore seleziona la classe e il robot con cui giocare.
Post-Condizioni	Il sistema fornisce un report della selezione effettuata e conferma la scelta effettuata, creando una nuova partita.

5) Salva partita:

Attore Primario	Studente
Descrizione	Lo studente, dopo l'<<include>> dall'azione seleziona classe e robot, crea una nuova partita.
Pre-condizioni	L'utente deve aver selezionato la classe e il robot con cui giocare.
Flusso Principale	<ol style="list-style-type: none"> 1) Il giocatore crea una nuova partita. 2) Il giocatore visualizza l'editor.
Post-Condizioni	La partita viene creata e si avvia l'editor.

6) Nuova partita

Attore Primario	Studente
Descrizione	Lo studente, dopo l'<<include>>, salva la nuova partita nel Game Repository.

Pre-condizioni	L'utente, dopo aver salvato la partita, crea una nuova istanza partita.
Flusso Principale	1) Il giocatore crea una nuova partita. 2) Il giocatore visualizza l'editor.
Post-Condizioni	La partita viene salvata e identificata mediante un ID.

Nel caso d'uso login sono presenti **tre attori**:

- **Utente**: l'attore che non ha effettuato il login.
- **Studente**: l'attore principale, dopo l'autenticazione.
- **Amministratore**: un attore fuori scena, che beneficia in maniera indiretta delle azioni dello studente.

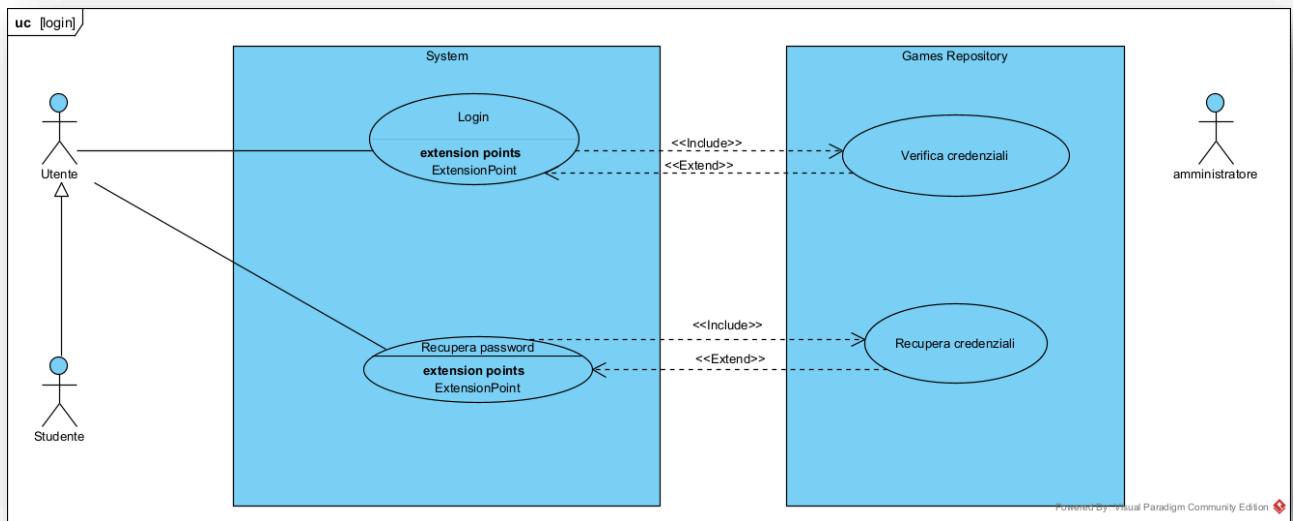


Figura 2.4 – Caso d'uso login

In questo diagramma vengono individuati due blocchi principali, che rappresentano il **System** e il **Games Repository**: l'utente, nel primo caso, proverà ad **effettuare il login** e, mediante il collegamento di **<<include>>**, si rappresenta il collegamento con il Games Repository, grazie al quale è possibile recuperare le credenziali con l'impiego dell'**<<extend>>**; se il login va a buon fine, l'utente verrà riconosciuto come studente e quindi sarà autenticato. Un'altra funzionalità messa a disposizione dell'utente è il **recupero delle credenziali**, grazie al quale l'utilizzatore può richiedere il recupero della password, rappresentato attraverso l'**<<include>>** che permette di accedere al Games Repository e cambiare la password; in caso di esito positivo, mediante l'**<<extend>>** rappresentato

l'utente potrà ottenere così la nuova password con cui interfacciarsi all'applicativo. Di seguito viene riportata nel dettaglio la specifica del caso d'uso con la rappresentazione tabellare:

1) Login:

Attore Primario	Utente
Descrizione	L'utente inserisce le credenziali e richiede l'accesso al sistema.
Pre-condizioni	Nessuna.
Flusso principale	1) L'utente inserisce le credenziali.
Post-Condizioni	Il sistema riconosce come studente l'utente che ha effettuato il login.

2) Verifica credenziali

Attore Primario	Utente
Descrizione	L'utente attende la verifica delle credenziali e in caso di esito positivo risulterà autenticato in quanto studente.
Pre-condizioni	Lo studente deve aver inserito le credenziali nel sistema.
Flusso principale	1)L'utente inserisce le credenziali. 2)L'utente richiede l'accesso al sistema.
Post-Condizioni	Il sistema riconosce le credenziali inserite.

3) Recupera password:

Attore Primario	Utente
Descrizione	L'utente recupera la password per poter accedere.
Pre-condizioni	L'utente non deve essere autenticato per poter recuperare le credenziali.
Flusso principale	1) L'utente richiede delle nuove credenziali.
Post-Condizioni	Il sistema accoglie la richiesta di recupero.

4) Recupera credenziali:

Attore Primario	Utente
Descrizione	L'utente recupera le credenziali.
Pre-condizioni	L'utente non deve essere autenticato per poter recuperare le credenziali.
Flusso principale	1)L'utente richiede delle nuove credenziali.
Post-Condizioni	Il sistema restituisce all'utente delle nuove credenziali.

2.3 Diagrammi di sequenza

I *diagrammi di sequenza* sono uno **strumento complementare** rispetto ai casi d'uso, utili a mostrare i flussi d'esecuzione rispetto agli oggetti interessati dai requisiti funzionali. In Figura 2.5 viene mostrato il diagramma di sequenza che descrive l'interazione dell'attore Studente con l'architettura.

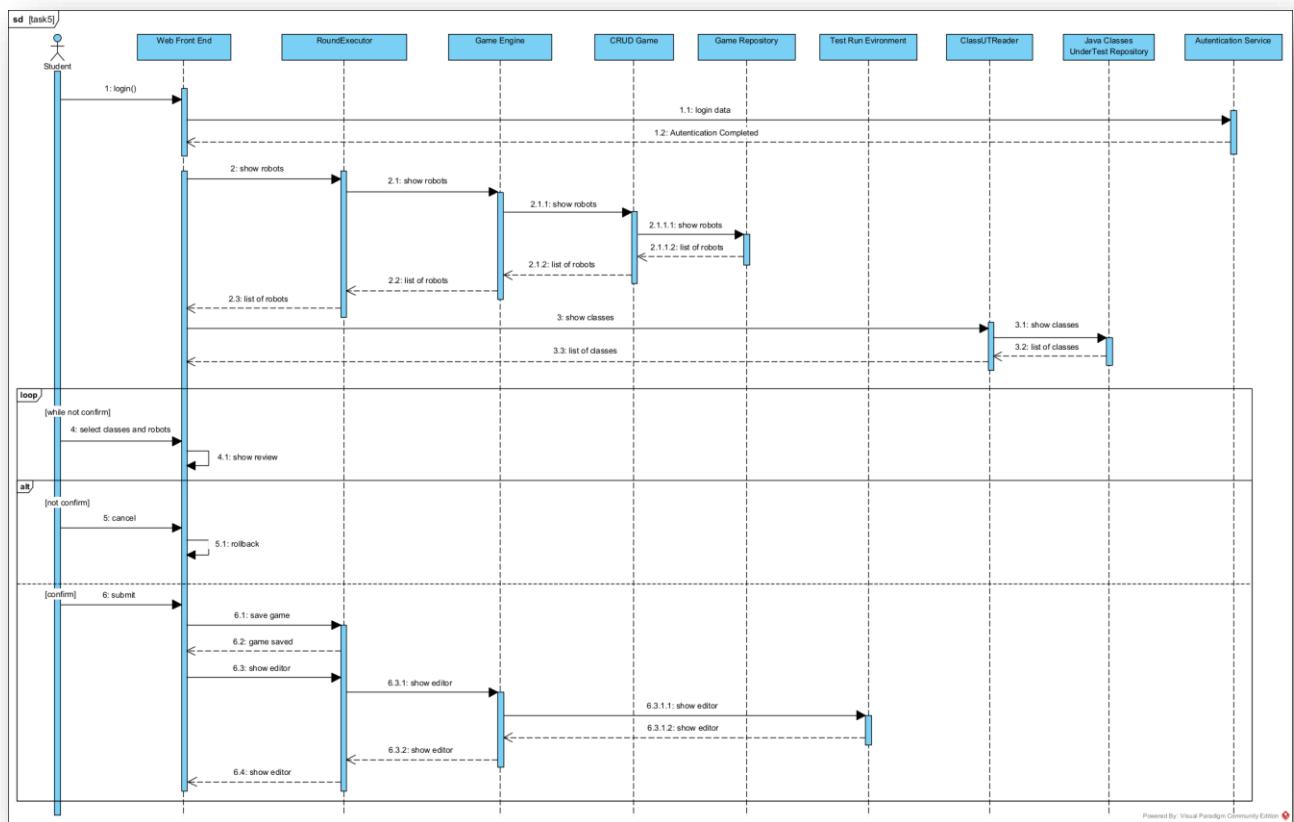


Figura 2.5 – Diagramma di sequenza

Nel diagramma sono state prese come lifeline di riferimento le componenti della Figura 2.1 fornite dal product owner. L'idea di base è che l'attore si **interfacci** principalmente con il **Web Front End**, incluso come componente del T5. Lo scopo del diagramma è quello di permettere al team di individuare i servizi correlati alle specifiche funzionali, precedentemente esaminate con i casi d'uso. Dall'analisi è emerso che il task T5 impiega diversi componenti dell'architettura di base e che per eseguire il login e salvare la partita è strettamente necessario interfacciarsi con il Game Engine e il Games Repository. Di seguito viene riportata una descrizione dei vari scambi di messaggi mostrati nel diagramma.

L'attore Studente effettua, in primis, la procedura di login, comunicando tramite il Web Front End per sfruttare il servizio di autenticazione (implementato nel task T2-3). Terminata la procedura, vengono mostrate le liste dei robot e delle classi sulla Web Application. Visualizzate le liste, lo studente può effettuare la selezione della classe da testare e del robot su cui effettuare il test. In seguito, viene mostrato un riepilogo delle scelte e viene offerta la possibilità di confermare le opzioni, salvare i dati relativi alla partita e avviare l'apposito editor. Nel caso in cui non si intenda procedere con l'avvio dell'editor, è possibile annullare le operazioni e tornare alla pagina di selezione delle classi e dei robot tramite rollback.

3 Documentazione di progetto

3.1 Introduzione al contesto

Nel capitolo precedente è stato effettuata un'analisi del dominio del problema esclusivamente dal punto di vista dei requisiti, funzionali e non, e dei possibili scenari di funzionamento dell'applicativo. In questa sezione, invece, si sposta il focus sullo sviluppo di **diagrammi** per la descrizione dell'**architettura** sotto vari aspetti, che sono risultati fondamentali per le implementazioni dei codici di front-end e back-end.

3.1.1 Diagramma di contesto

Il punto di partenza è definire un *diagramma di contesto* (Figura 3.1), attraverso il quale si vuole fornire una descrizione del quadro generale relativo all'esecuzione dell'applicativo, includendo gli attori interessati e tutti gli elementi che permettono la realizzazione del servizio implementato.

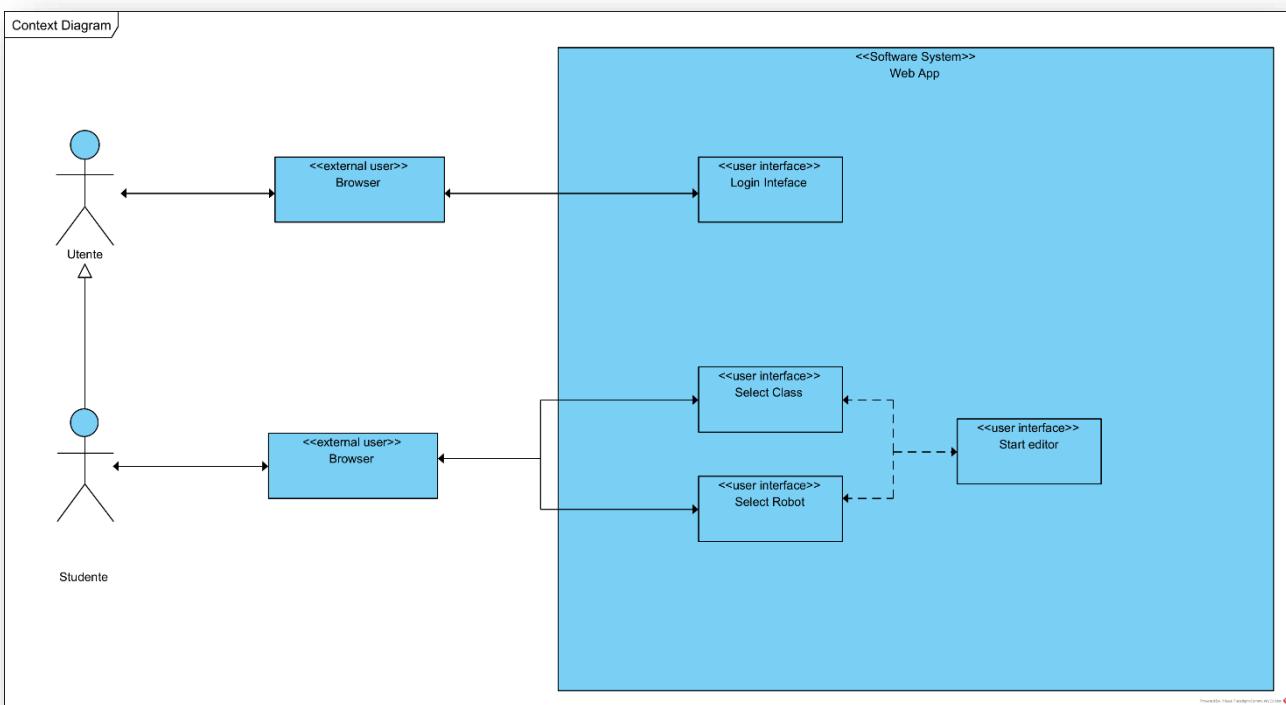


Figura 3.1 – Diagramma di contesto

Nella fattispecie, vengono individuati due attori: Utente e Studente. Il secondo è una specializzazione del primo. Entrambi gli attori in questione usufruiscono del browser per potersi servire dell'interfaccia utente messa a disposizione dal sistema software e sfruttare le funzionalità offerte.

In particolare, l'attore Utente può interfacciarsi solo con la pagina di login, in quanto solo gli utenti registrati e autenticati correttamente posso accedere al gioco.

Questo passa alla specializzazione Studente una volta effettuata l'autenticazione, e, a questo punto, potrà utilizzare la user interface per la **selezione** delle classi e dei robot e il successivo **avvio dell'editor** della partita con le scelte compiute.

3.2 *Tecnologie e ambiente di sviluppo*

Come menzionato nella sezione 1.2.3, è stato impiegato l'editor Visual Studio Code per la scrittura e la compilazione del codice. Pur nascendo come editor di codice leggero, l'intero team se n'è servito utilizzandolo come ambiente di sviluppo, in quanto è considerabile un IDE adeguato grazie alla possibilità di aggiungere delle estensioni per l'introduzione di framework e altre tecnologie.

3.2.1 *Spring Boot*

Spring Boot è un framework open-source basato su Java che semplifica lo sviluppo di applicazioni Java. Fornisce un'infrastruttura pronta all'uso per gestire molte delle complessità associate allo sviluppo di applicazioni, consentendo agli sviluppatori di concentrarsi sulla logica aziendale principale. Spring Boot include funzionalità per la gestione delle dipendenze, la configurazione automatica, l'implementazione di server Web integrati e la facilità di sviluppo e test. È ampiamente utilizzato per la creazione rapida di applicazioni Java scalabili, robuste e facilmente gestibili.

3.2.2 *Thymeleaf*

Thymeleaf è un motore di template Java open-source che consente di creare facilmente e dinamicamente pagine HTML. È progettato per essere integrato con applicazioni web basate su Java, come ad esempio quelle sviluppate con Spring Boot. Thymeleaf supporta la creazione di template HTML con l'aggiunta di espressioni e attributi speciali che permettono di rendere dinamici i contenuti delle pagine. È in grado di collegare i dati del modello Java con gli elementi HTML, facilitando la generazione di pagine web dinamiche e interattive. Thymeleaf offre inoltre funzionalità avanzate come il supporto internazionale, la gestione dei layout e la manipolazione dei form. È una scelta popolare per la creazione di interfacce utente in applicazioni web Java.

3.2.3 *Bootstrap*

Bootstrap è un framework front-end open-source che semplifica la progettazione e lo sviluppo di siti web responsivi e intuitivi. Fornisce una collezione di componenti predefiniti, come pulsanti, moduli, barre di navigazione e griglie di layout, che possono essere utilizzati per costruire rapidamente interfacce utente moderne e ben progettate. Bootstrap utilizza HTML, CSS e JavaScript per fornire un set di strumenti e stili coerenti che rendono più facile la creazione di siti web responsive che si adattano automaticamente a diversi dispositivi e dimensioni dello schermo. È altamente personalizzabile e offre anche una vasta gamma di temi e plugin aggiuntivi per estendere le

funzionalità di base. Bootstrap è ampiamente utilizzato nel web development per risparmiare tempo e sforzi nella creazione di interfacce utente di alta qualità.

3.2.4 Maven

Maven è uno strumento di gestione di progetti Java che semplifica la gestione delle dipendenze e il processo di compilazione. La sua integrazione nel progetto è utile in quanto Maven permette di specificare le librerie necessarie nel file di configurazione “**pom.xml**” e si occupa di scaricarle automaticamente. Inoltre, Maven promuove una struttura di progetto standardizzata, semplificando la collaborazione nel team. Grazie al suo ciclo di vita predefinito, è possibile compilare, testare e imballare l'applicazione facilmente. Maven supporta anche plugin che estendono le funzionalità di base, come il controllo della qualità del codice o la generazione di documentazione automatica.

3.2.5 Estensioni – Spring Boot Tools e Spring Boot Dashboard

Spring Boot Tools e Spring Boot Dashboard offrono strumenti e funzionalità utili per semplificare e migliorare lo sviluppo di applicazioni Spring Boot all'interno di Visual Studio Code, consentendo agli sviluppatori di lavorare in modo più efficiente e produttivo.

In particolare, Spring Boot Tools supporta funzionalità come il completamento del codice, l'analisi statica, il debug e la gestione delle dipendenze specifiche di Spring Boot. Consente anche di eseguire le applicazioni Spring Boot direttamente da Visual Studio Code, facilitando il processo di sviluppo e test.

Con Spring Boot Dashboard, d'altra parte, è possibile visualizzare e gestire le applicazioni Spring Boot in esecuzione sul proprio ambiente locale o su server remoti. Fornisce informazioni dettagliate sullo stato delle applicazioni, consente di avviare, arrestare e riavviare le applicazioni, e offre anche la visualizzazione dei log dell'applicazione e delle metriche di monitoraggio delle prestazioni.

3.3 Diagramma di sequenza MVC

Il *diagramma di sequenza MVC* è una rappresentazione visuale dell'interazione tra i componenti principali dell'applicativo basata sul **pattern architetturale MVC**. Questo diagramma illustra la sequenza degli eventi e delle azioni che si verificano tra il modello, la vista e il controller nell'applicazione, e come l'attore Studente interagisce con essi. È utile mostrare le differenze con il diagramma di sequenza mostrato nel Paragrafo 2.3, dove l'attore Studente si interfaccia principalmente con le componenti menzionate nei requisiti funzionali, mentre qui viene rappresentato il flusso di controllo del sistema nel suo complesso e come l'utente può eseguire le azioni indicate nella descrizione del task assegnato. In Figura 3.2 è riportato il diagramma.

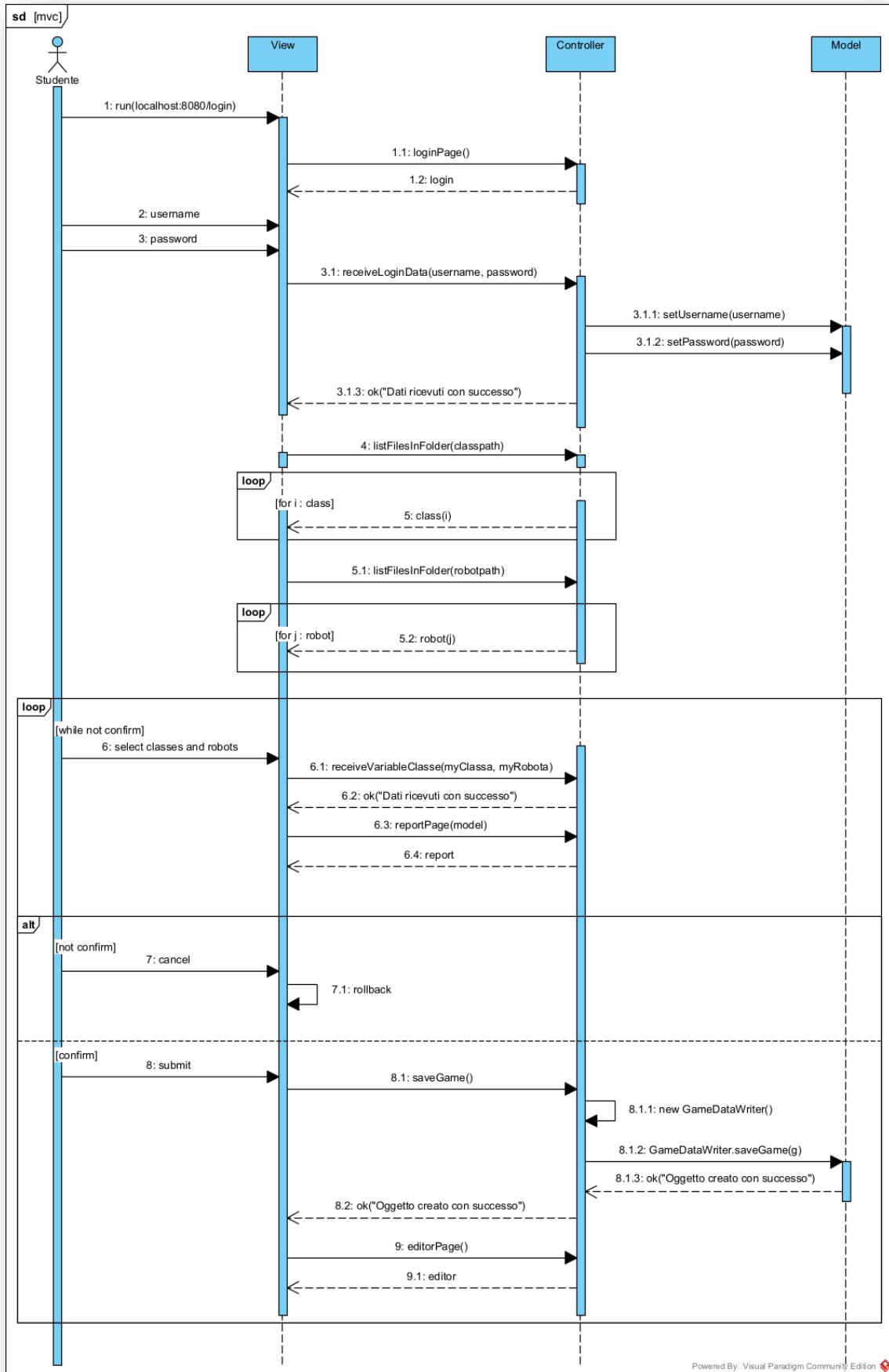


Figura 3.2 – Diagramma di sequenza MVC

3.4 Diagramma dei deployment

Per la descrizione della distribuzione dei componenti del sistema software su hardware e risorse di rete è stato introdotto un *diagramma di deployment*, che fornisce una visione ad alto livello dell'**architettura fisica** del sistema, mostrando le connessioni e le interazioni tra i componenti software e l'infrastruttura sottostante.

In Figura 3.3 è riportato il diagramma.

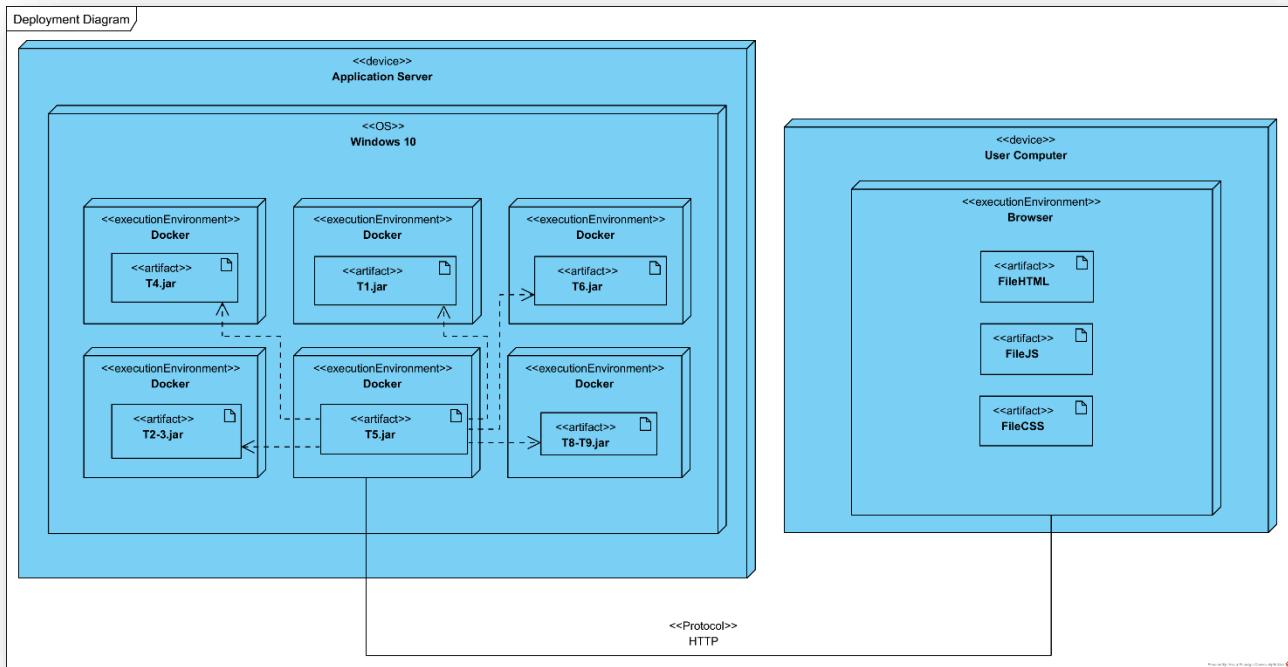


Figura 3.3 – Diagramma di deployment

Come si può evincere dal diagramma, vengono individuati due device: *User Computer* (client) e *Application Server* (server).

Dal lato client, l'utente si serve di un qualsiasi browser come ambiente di esecuzione, e interagisce con l'applicativo sfruttando un'interfaccia implementata a partire da artefatti di tipo HTML, JavaScript e CSS.

Dal lato server, l'applicazione impiega **Docker** come ambiente per l'esecuzione, in modo tale da evitare problemi di integrabilità. Lo scopo è quello di creare un container con un ambiente dedicato per ogni artefatto. Inoltre, dal lato server è necessario realizzare delle interconnessioni tra i vari container per permettere agli artefatti di comunicare tra di loro.

3.5 Diagramma dei package

Il *diagramma dei package* rappresenta uno strumento fondamentale per comprendere la **struttura organizzativa** di un sistema software basato sul modello MVC. Esso offre una visualizzazione delle relazioni tra i componenti **Model**, **View** e **Controller**, fornendo una panoramica esaustiva dell'architettura dell'applicazione e delle dipendenze tra i diversi componenti alla base di quest'ultima.

In Figura 3.4 è riportato il diagramma dei package.

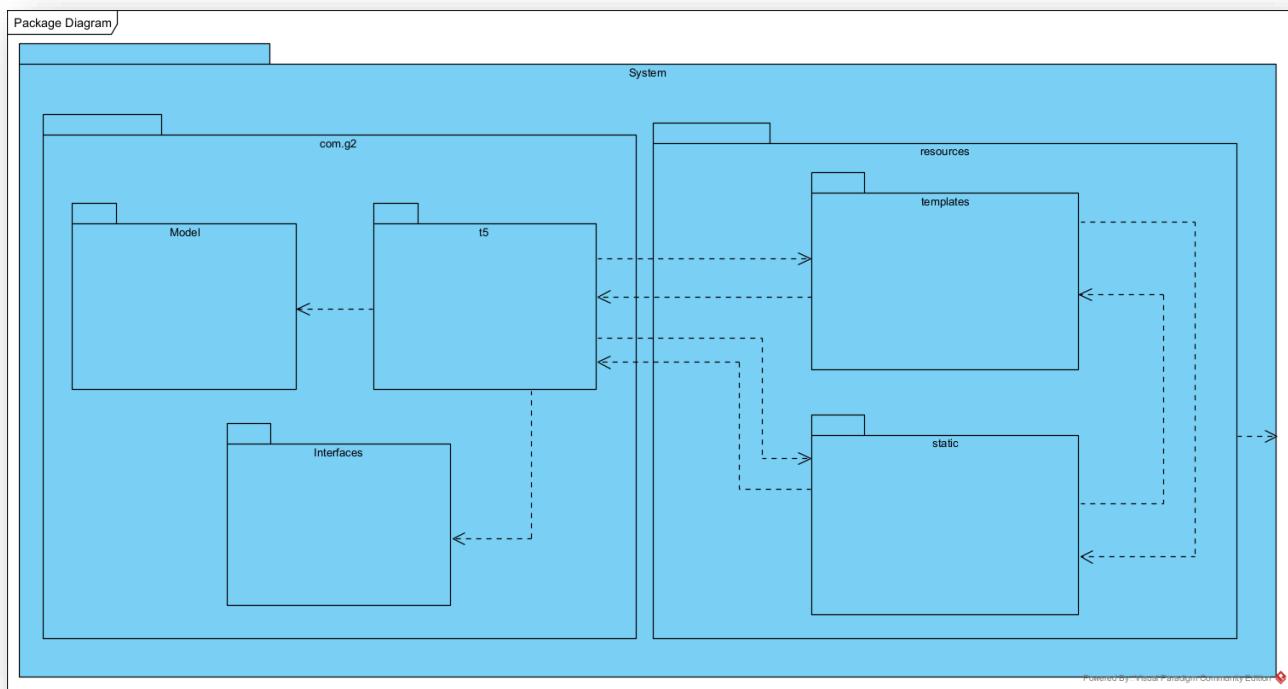


Figura 3.4 – Diagramma dei package

Dal diagramma è facile evincere che il sistema viene rappresentato mediante un package **System**, il quale include i due repository principali: **com.g2** e **resources**.

In *com.g2* viene implementata la componente di back-end. Esso contiene il package **Interfaces** per l'integrazione con altri task, il package **Model** per la gestione delle classi **Game** e **Player**, e il package **t5** che rappresenta la sezione di controllo.

Il package *resources*, invece, contiene le componenti alla base della View, che vengono visualizzati a livello client. Quanto viene illustrato chiarisce meglio la composizione interna dell'architettura software esaminata precedentemente nel diagramma di deployment. All'interno di *resources* vengono individuati i package **static** e **templates**. Il primo contiene cartelle di file relativi alla definizione del layout e delle funzionalità dinamiche della pagina web. Il secondo include la parte inerente ai codici HTML.

3.6 Diagramma delle classi

Il *diagramma delle classi* offre una **visualizzazione chiara e strutturata** delle **classi** del sistema e delle **relazioni** tra di loro. Riveste un ruolo fondamentale nel definire la struttura e le interazioni del software, facilitando la comprensione e la comunicazione tra gli sviluppatori e gli stakeholder coinvolti. Esso fornisce una descrizione dettagliata delle classi, mettendo in luce le loro funzionalità principali. Questa descrizione facilita l'implementazione e la manutenzione del sistema software, garantendo una base solida per il successo del progetto. In Figura 3.5 viene riportato il diagramma delle classi realizzato.

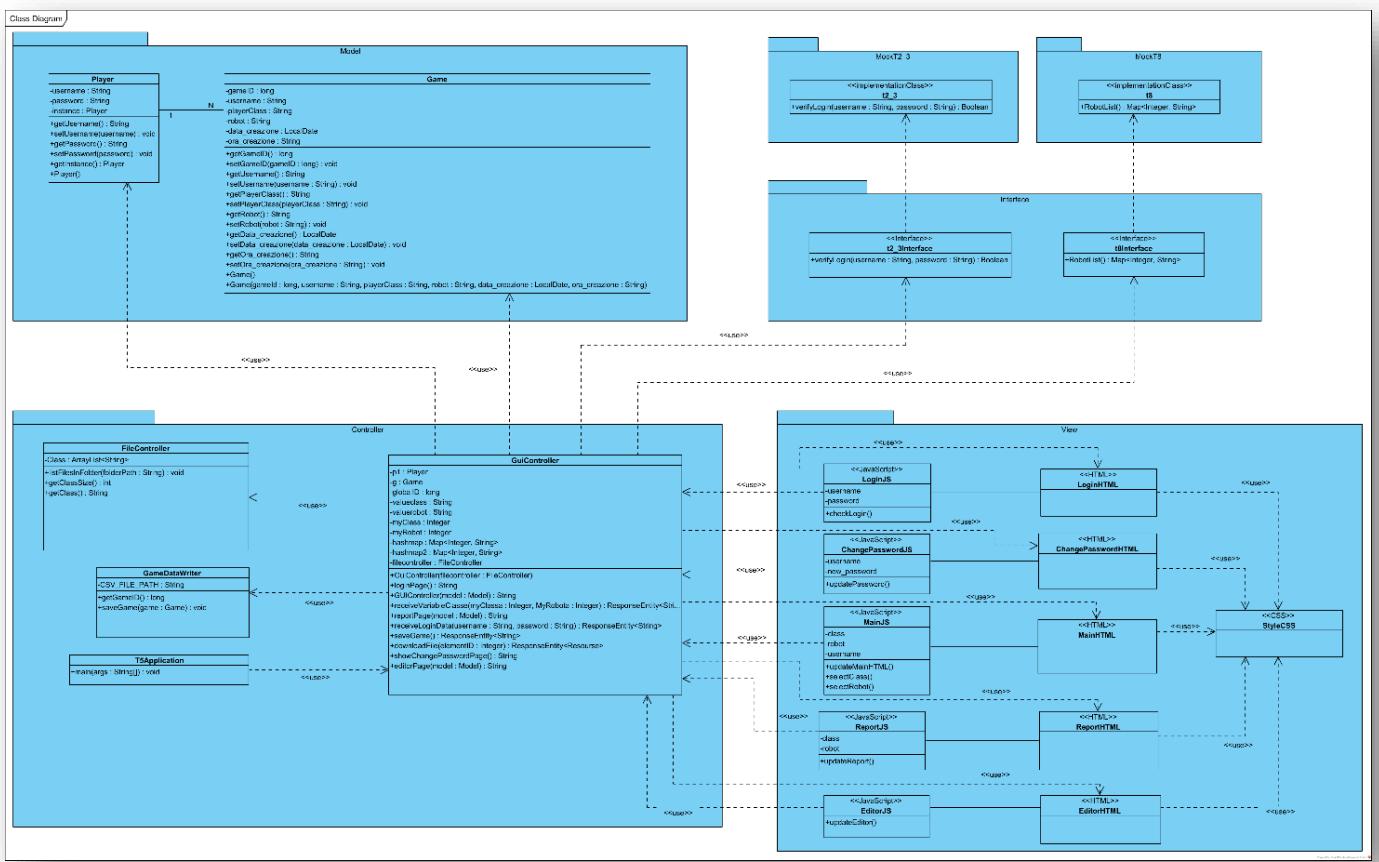


Figura 3.5 – Diagramma delle classi

Nel diagramma sono raffigurati **quattro package**, di cui tre rappresentano le componenti **MVC** e uno le interfacce. La classe **T5application** rappresenta il main dell'applicazione. Mediante questa classe e la funzione di partenza si lancia il **GUIController**, che permette di gestire le varie pagine dell'applicazione con l'ausilio di diverse funzioni. Per interfacciarsi con i vari file per la gestione dei dati e per creare gli oggetti Giocatore e Partita, il GUIController impiega le relazioni `<<use>>`. Tra le classi fondamentali per il funzionamento dell'applicativo possiamo trovare **Game** e **Player**, le quali appartengono al package Model. In particolare, la classe Player è stata pensata per essere un singleton e per essere in una relazione 1:N con la classe Game. Questo garantisce di mantenere una

singola istanza del giocatore, la quale, però, può essere associata a più partite. Il package **Interface** ingloba al suo interno le classi di interfaccia e le corrispondenti classi che le implementano, con lo scopo di simulare la presenza di altri task nel funzionamento dell'applicazione. Infine, il package **View** include le classi JS, HMTL e CSS relative all'implementazione del front-end.

3.7 Diagramma dei componenti

Il *diagramma dei componenti* è uno strumento essenziale per la progettazione del software, poiché fornisce una visione ad alto livello della struttura del sistema. Rappresentando i componenti principali e le loro relazioni, aiuta a comprendere l'**architettura complessiva**. Questo diagramma permette di identificare punti critici e migliorare l'integrazione dei vari componenti. Utilizzato in combinazione con altri diagrammi (package e classi), offre una rappresentazione completa del sistema. Dettagli implementativi, come interfacce e dipendenze, possono essere aggiunti per ottenere ulteriori informazioni utili per l'integrazione. In Figura 3.6 viene rappresentato il diagramma dei componenti realizzato.

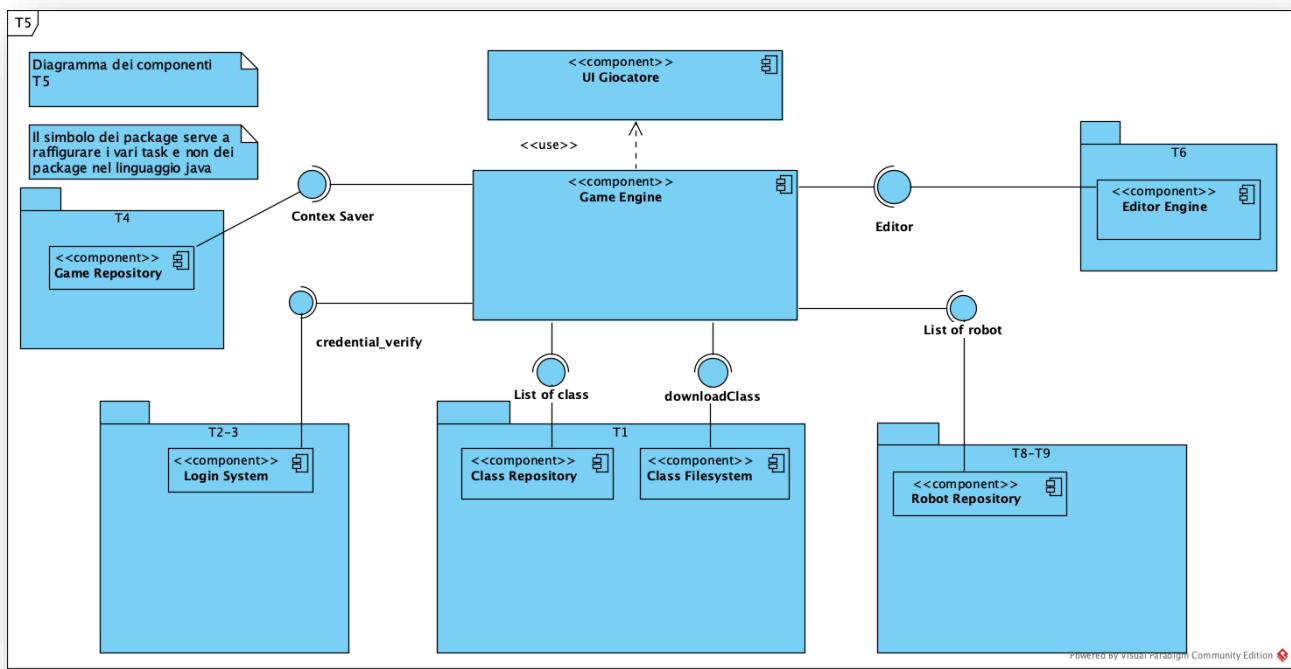


Figura 3.6 – Diagramma dei componenti

Come è possibile notare dal diagramma, il **Game Engine** è il <<component>> di base, il quale permette la gestione della logica dell'applicazione e la raccolta dei dati necessari per il corretto funzionamento dai componenti implementati dagli altri task. L'integrazione avviene mediante l'ausilio di interfacce, e l'implementazione di quest'ultime è stata concordata con gli altri team per creare un software consistente. E' stato scelto di soffermarsi sulla realizzazione dell'interfaccia

Editor in collaborazione con il task T6 del gruppo G12. Di seguito vengono riportate le interfacce con le funzioni e i parametri previsti al fine di realizzare un'integrazione completa con gli altri task:

- **Task T4:** Il task T4 si occupa di salvare la partita e di gestire la logica di salvataggio e di consistenza dei dati inerenti ad essa. In Figura 3.7 e 3.8 vengono riportate le funzioni ed i parametri previsti dall'interfaccia *Context Saver*, il cui scopo è quello di effettuare il salvataggio della partita.

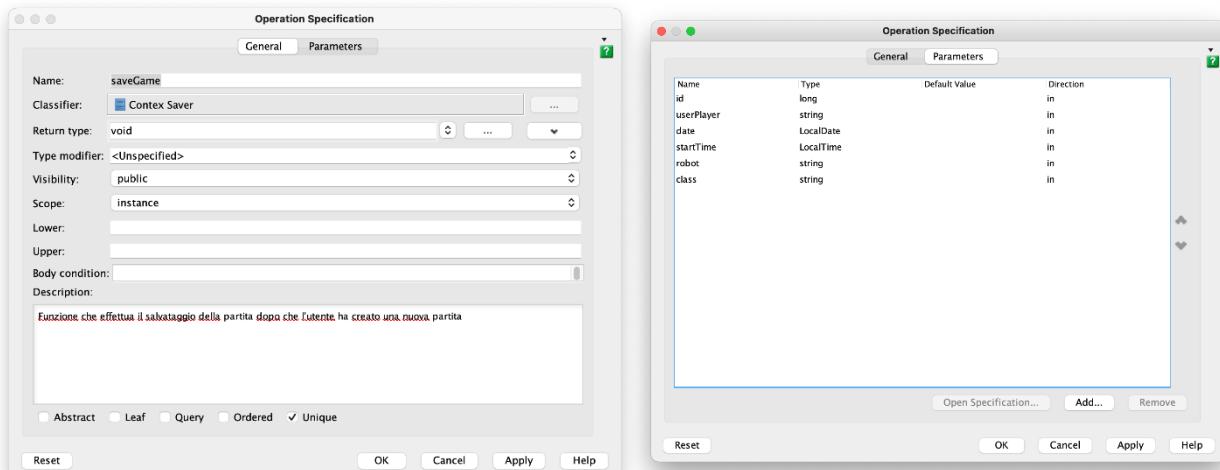


Figura 3.7-3.8 – Funzione di salvataggio della partita e parametri

- **Task T2-3:** Il task T2-3 si occupa della gestione del login e permette al task T5 di esporre la GUI inerente all'accesso e al recupero delle credenziali. Inoltre, permette di realizzare il recupero dello user ID, il quale deve essere associato al salvataggio della partita e all'editor per l'identificazione del giocatore che scrive il test della classe. In Figura 3.9, 3.10, 3.11 e 3.12 vengono riportate le funzioni e i parametri dell'interfaccia *credential_verify*.

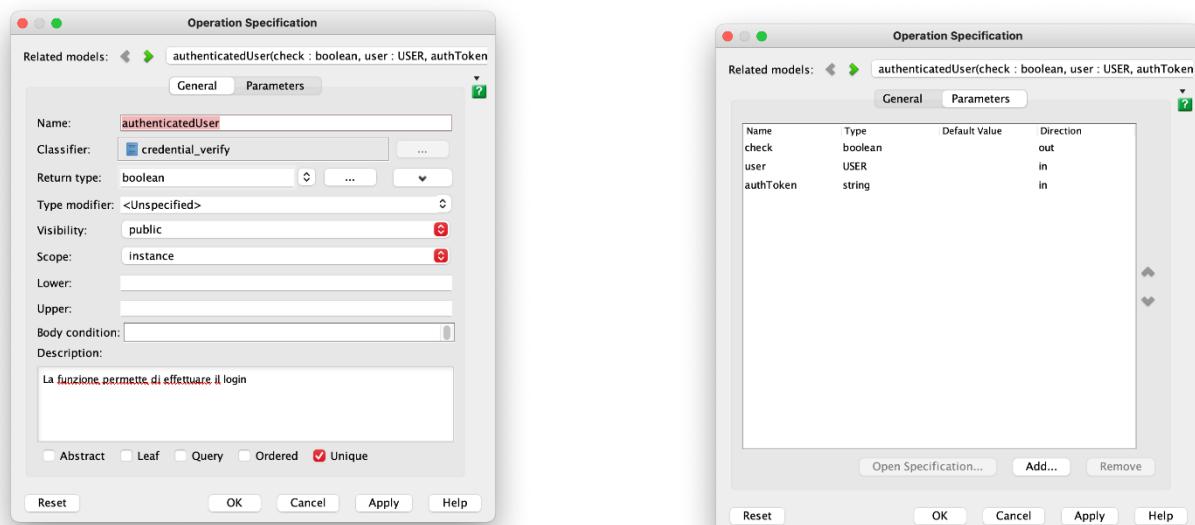


Figura 3.9-3.10 – Funzione di autenticazione e parametri

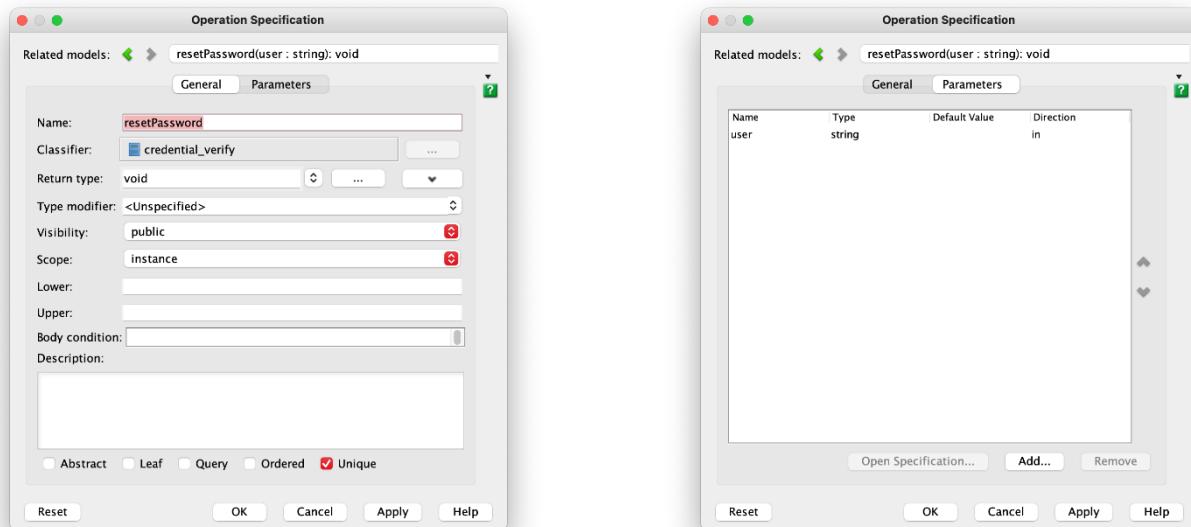


Figura 3.11-3.12 – Funzione di reset della password e parametri

- **Task T1:** Il task T1 si occupa di gestire le classi nel gioco, fornendo un elenco di tutte quelle che sono a disposizione del giocatore. Inoltre, si individuano due interfacce differenti: una per la gestione del download della classe e una per la gestione dell’elenco delle classi. In Figura 3.13, 3.14, 3.15 e 3.16 vengono riportate le funzioni e i parametri dell’interfaccia *List of class* e *downloadClass*.

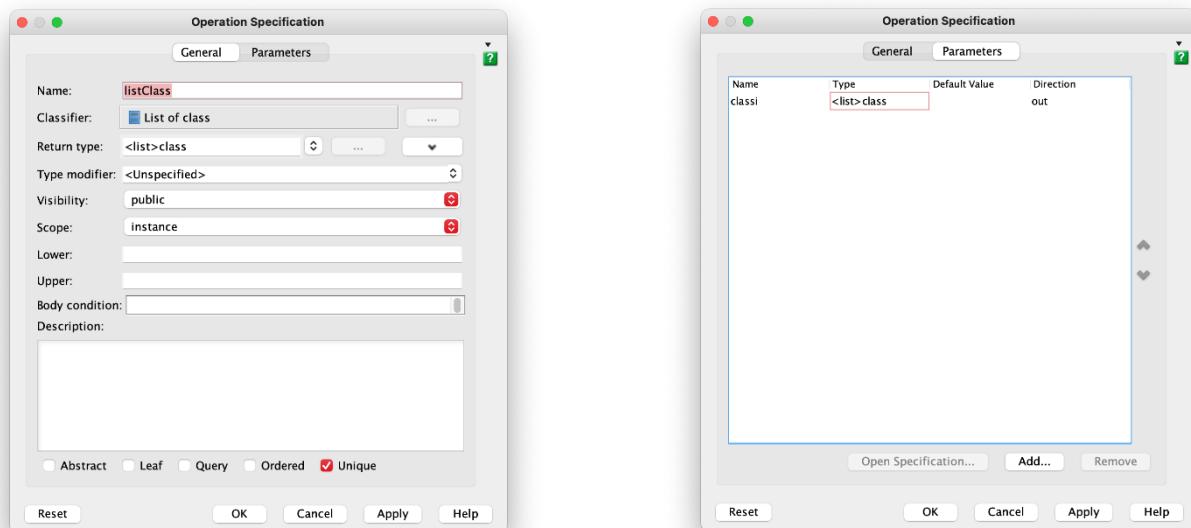


Figura 3.13-3.14 – Funzione di visualizzazione della lista delle classi e parametri

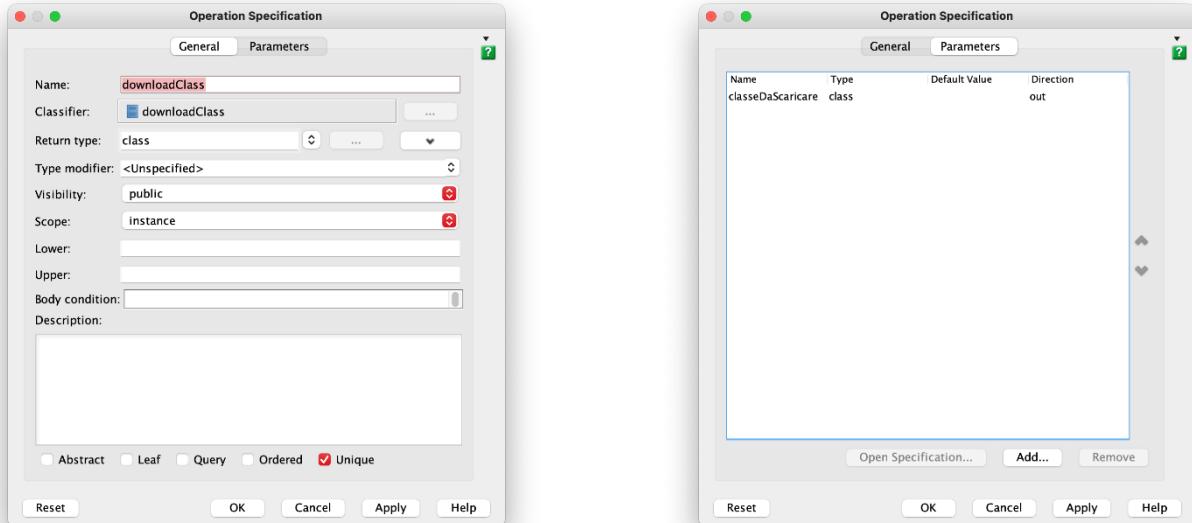


Figura 3.15-3.16 – Funzione di download delle classi e parametri

- Task T8-9:** Il task T8-9 si occupa della gestione e della creazione dei robot che fungono da avversari per il giocatore nella realizzazione dei test su una determinata classe. Lo scopo dell’interfacciamento con questo task è quello di ottenere una lista di avversari selezionabili, i quali, similmente a quanto avviene con le classi, vengono scelti dal giocatore all’avvio della partita. In Figura 3.17 e 3.18 vengono riportate le funzioni e i parametri relativi all’interfaccia *List of Robot*.

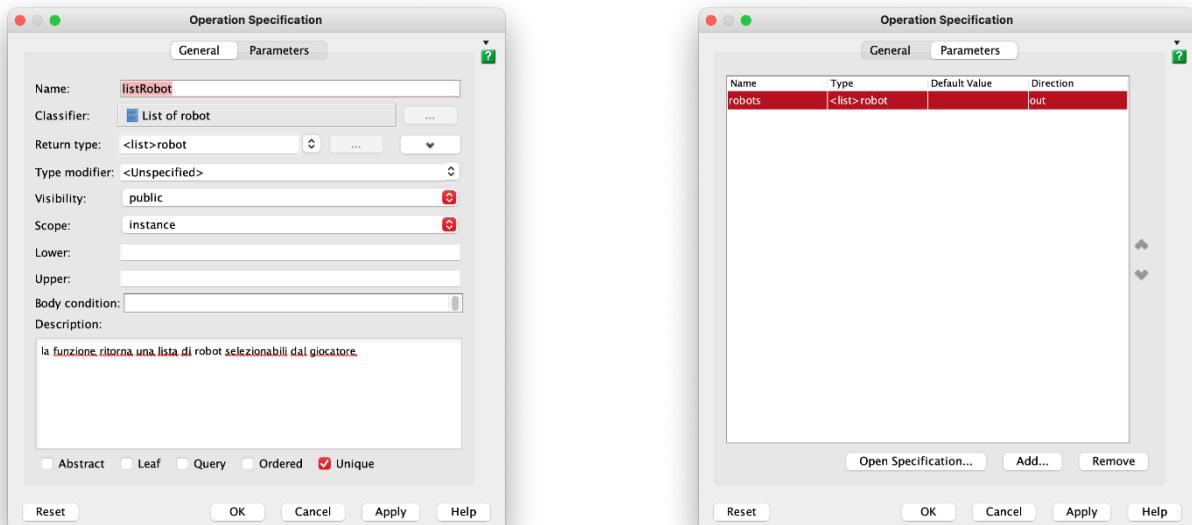


Figura 3.17-3.18 – Funzione di visualizzazione della lista dei robot e parametri

- Task T6:** Il task T6 si occupa di creare l’editor e l’ambiente in cui inserire nuove classi nel file system e di modificare le classi già esistenti. Questo task riceve dal T5 i dati relativi alla partita appena creata. Lo scopo del T5 è, quindi, quello di creare la partita e di passare tutte

le informazioni relative alle selezioni effettuate dall'utente al task T6, per mostrare a schermo quanto precedentemente selezionato. In Figura 3.19 e 3.20 vengono riportate le funzioni e i parametri relativi all'interfaccia *Editor*.

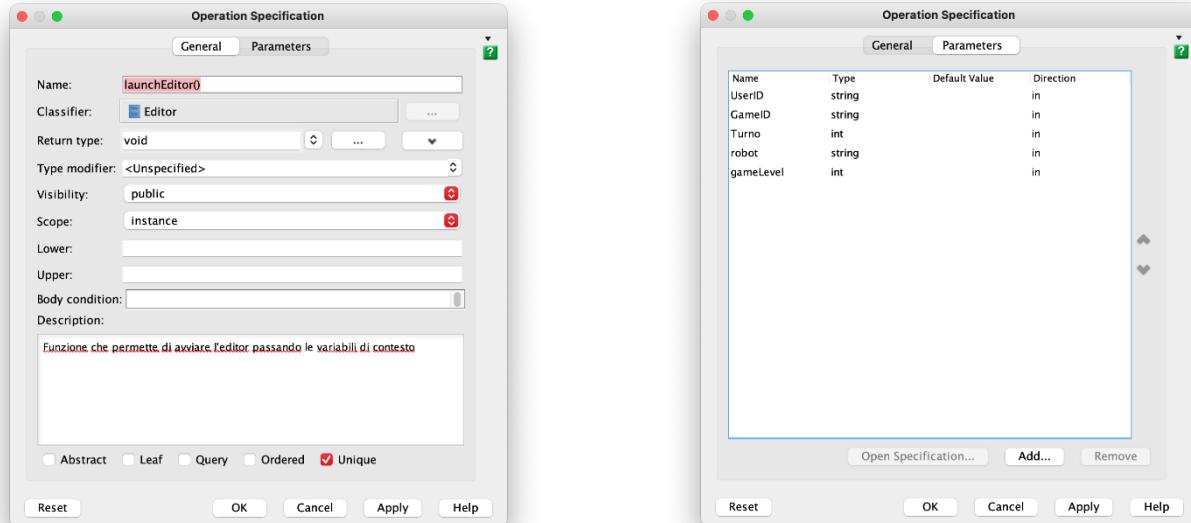


Figura 3.19-3.20 – Funzione di lancio dell'editor e parametri

3.8 Considerazioni sull'implementazione

L'architettura dell'applicativo sviluppato si basa sul pattern strutturale **Model-View-Controller** (MVC). Questa scelta è stata fatta considerando le caratteristiche vantaggiose di questo pattern per strutturare la web app mediante l'impiego di Spring Boot.

Il pattern MVC separa le responsabilità dell'applicazione in tre componenti principali. Il **Modello** gestisce i dati e la logica dell'applicazione, la **Vista** offre un'interfaccia utente attraverso la quale è possibile visualizzare i dati e raccogliere gli input, il **Controller** coordina le interazioni utente, gestisce la logica di controllo e aggiorna il Modello e la Vista di conseguenza.

Inoltre, il pattern MVC favorisce la scalabilità dell'applicazione ed è ampiamente supportato dai principali framework e tecnologie per lo sviluppo web, offrendo un ecosistema di strumenti e risorse che semplificano lo sviluppo dell'applicazione.

Si è preferito utilizzare tale pattern a discapito di un *pattern BCE* in quanto quest'ultimo riduce la flessibilità nella gestione delle interfacce utente e presenta una complessità maggiore nella manutenzione, dato che eventuali modifiche o estensioni del software richiedono una comprensione approfondita delle interazioni tra *Boundaries*, *Controllers* ed *Entities*. Inoltre, il concetto di Entities è scarsamente compatibile con la gestione dei dati realizzata nel T5 e la necessità di realizzare una UI hanno decretato la necessità di impiegare il pattern MVC.

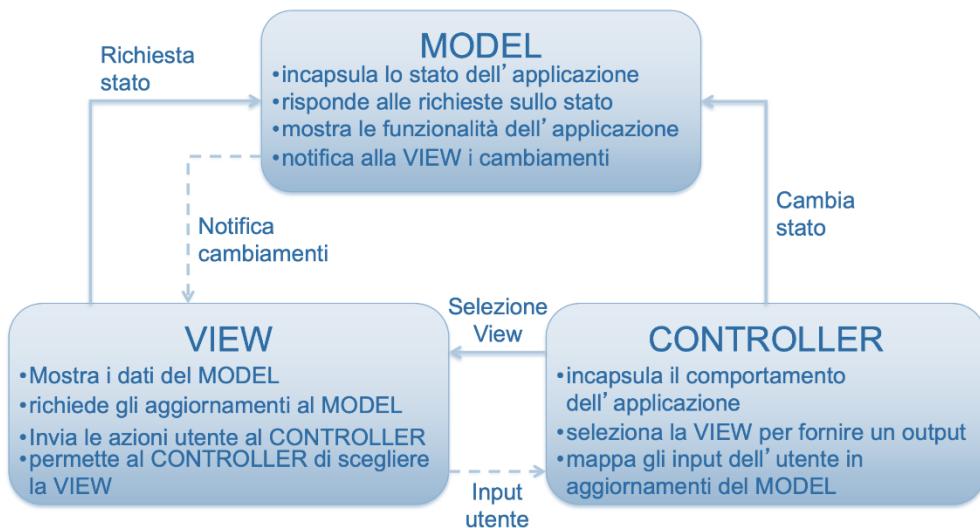


Figura 3.21 – Pattern MVC

4 Guida all'installazione e all'utilizzo del software

Si desidera fornire una documentazione esaustiva e dettagliata sul processo di installazione e sull'utilizzo del software proposto, con tutte le istruzioni necessarie affinché l'installazione dell'applicativo sul sistema ospitante vada a buon fine. In aggiunta, si offrirà una guida attraverso tutte le funzionalità principali, facilitandone un corretto utilizzo.

4.1 Docker e WSL

Docker è una piattaforma di **containerizzazione** che consente la creazione, la distribuzione e la gestione efficiente e isolata di applicazioni. Ciò assicura la massima portabilità dell'applicazione software, indipendentemente dall'ambiente di esecuzione sottostante.

Per effettuare l'esecuzione di Docker in ambiente Windows è necessario l'impiego di **WSL** (Windows Subsystem for Linux), che permette di eseguire un ambiente Linux all'interno di Windows, consentendo agli utenti di utilizzare strumenti e applicazioni Linux direttamente sul proprio sistema operativo Windows.

L'utilizzo di Docker permette di creare immagini e container che operano in ambienti isolati, consentendo l'esecuzione di servizi separati e la comunicazione tra di essi attraverso il mapping dei porti specifici. Questo approccio favorisce la modularità, la scalabilità e semplifica la distribuzione delle applicazioni, ottimizzando l'efficienza e la gestione delle risorse.

È stato possibile includere un file system locale all'interno del container, utilizzando percorsi assoluti per accedervi in combinazione con l'artefatto prodotto. Da questa configurazione è stata creata un'istanza di macchina virtuale, favorendo l'utilizzo delle risorse del file system senza dovervi rinunciare.

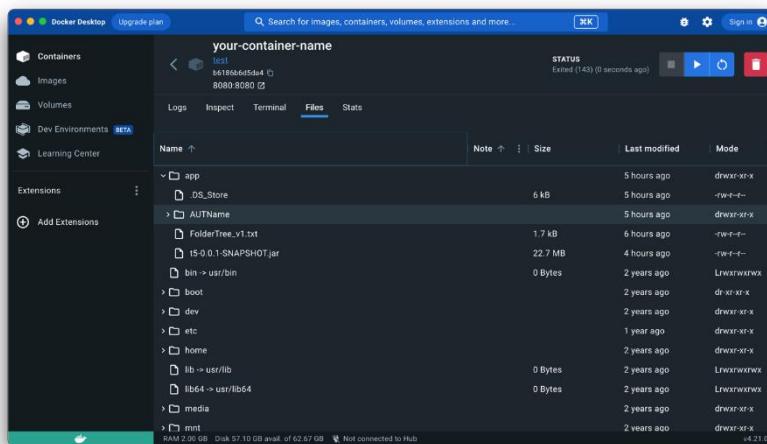


Figura 4.1 – Integrazione del file system in Docker

4.2 Istruzioni per l'installazione

Per installare l'applicazione, è necessario seguire i seguenti passaggi:

1. Clonare il repository da GitHub al seguente link: <https://github.com/Testing-Game-SAD-2023/T5-G2.git>
2. Aprire il terminale e navigare fino alla directory in cui è stata scaricata la cartella.
3. Eseguire i seguenti comandi:

```
cd t5
```

```
mvn package
```

```

G2_Progetto_SAD
TERMINALE
PROBLEMI 1 OUTPUT CONSOLO DI DEBUG TERMINALE COMMENTI
HELP.md mvnw.cmd pom.xml src target
Scanning for projects...
Building t5 0.0.1-SNAPSHOT < com.g2:t5 >
from pom.xml
[ jar ]
[INFO] -- resources:3.3.1:resources (default-resources) @ t5 --
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 34 resources from src/main/resources to target/classes
[INFO] -- compiler:3.11.0:compile (default-compile) @ t5 --
[INFO] Nothing to compile - all classes are up to date
[INFO] -- resources:3.3.1:testResources (default-testResources) @ t5 --
[INFO] skip non existing resourceDirectory /Users/lorenzoesposito/Desktop/G2_Progetto_SAD/t5/src/test/resources
[INFO] -- compiler:3.11.0:testCompile (default-testCompile) @ t5 --
[INFO] Nothing to compile - all classes are up to date
[INFO] -- surefire:3.0.0:test (default-test) @ t5 --
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running com.g2.t5.T5ApplicationTests
15:50:09,341 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not detect default configuration classes for test class [com.g2.t5.T5ApplicationTests]: T5ApplicationTests does not declare any static, non-private, non-final, nested classes annotated with @Configuration
15:50:09,441 [main] INFO org.springframework.boot.test.context.SpringBootTestBootstrapper -- Found @SpringBootConfiguration com.g2.t5.T5Application fo
r test class com.g2.t5.T5ApplicationTests
:: Spring Boot ::
(v3.1.1)
2023-07-01T15:50:09.772+02:00 INFO 1788 --- [ main] com.g2.t5.T5ApplicationTests : Starting T5ApplicationTests using Java 20.0.1 with
PID 1788 (started by lorenzoesposito in /Users/lorenzoesposito/Desktop/G2_Progetto_SAD/t5)
2023-07-01T15:50:09.774+02:00 INFO 1788 --- [ main] com.g2.t5.T5ApplicationTests : No active profile set, falling back to 1 default pr
ofile: "default"
2023-07-01T15:50:11.218+02:00 INFO 1788 --- [ main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 1 endpoint(s) beneath base path '/actuator'
Riga 5, colonna 33 Spazi: 4 UTF-8 LF Dockerfile ⌂ ⌂

```

Figura 4.2 – Build Maven

Dopo aver seguito i passaggi descritti, si ottiene come output un **file** artefatto con estensione “.jar”.

Questo file “.jar” sarà utilizzato per creare l'**immagine** Docker utilizzando il **Dockerfile**. Il Dockerfile dell'applicazione si trova nella cartella "**target**" insieme al file “.jar” generato da Maven. Di seguito sono riportati i passaggi per l'installazione dell'immagine Docker e la struttura del file Docker utilizzato.

```
cd target
```

```
docker build -t test -f test.dockerfile .
```

```
docker run -d -p 8080:8080 --name test_container-name test
```



The image shows a screenshot of a code editor with a dark theme. On the left, there's a vertical toolbar with icons for file operations (New, Open, Save, Find, Replace, etc.) and a notifications badge showing '39'. The main area displays a Dockerfile with syntax highlighting. The Dockerfile defines a Java application container with the following commands:

```
t5 > target > test.dockerfile > ...
1 # Usa un'immagine di base con Java installato
2 FROM openjdk:17-jdk
3
4 # Copia il tuo JAR nel container
5 COPY t5-0.0.1-SNAPSHOT.jar /app/
6
7 # Copia il repository nel container
8 COPY FolderTree /app/
9
10 # Imposta la directory di lavoro all'interno del container
11 WORKDIR /app
12
13 # Comando di avvio dell'applicazione
14 CMD ["java", "-jar", "t5-0.0.1-SNAPSHOT.jar"]
15
```

Figura 4.3 – Struttura del Docker File

Il Dockerfile mostrato nella *Figura 4.3* utilizza la versione 17 di JDK. Alle righe 5 e 8, vengono copiati i file presenti nella cartella "target" all'interno del container Docker, che sarà creato dalla Docker image. È importante notare che nella macchina virtuale non sarà presente solo il file jar, ma anche la repository FolderTree, che viene utilizzata come file system locale. Infine, l'applicazione viene avviata automaticamente all'interno del Docker utilizzando il file .jar tramite il comando **CMD**.

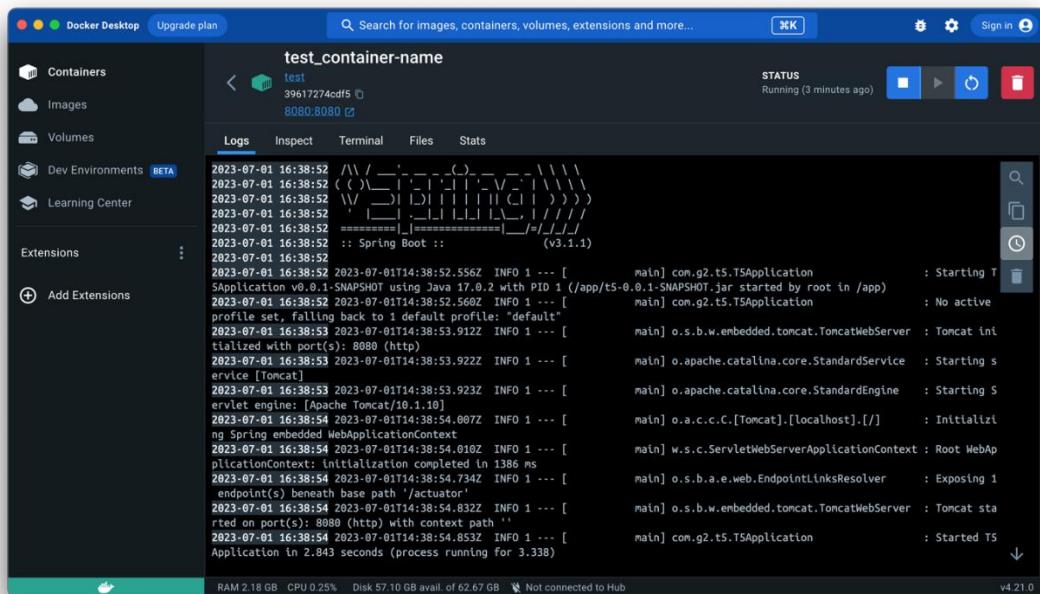


Figura 4.4 – Esecuzione server su Docker

Nella Figura 4.4, viene mostrato l'output dei log che non sono visibili all'utente, ma che confermano l'avvio corretto del server per il funzionamento dell'applicativo. Come si può notare da

questi output, il file jar è in esecuzione con il PID 1 della macchina Docker e sta correttamente eseguendo sulla porta 8080, come previsto durante l'installazione mediante l'apposito comando.

4.3 Linee guida per l'utilizzo

Il software sviluppato, come illustrato precedentemente, utilizza un modello Client-Server, in cui il server viene avviato su Docker a seguito dell'installazione descritta in precedenza. Il client, che rappresenta la componente con cui il giocatore interagisce, può essere avviato tramite qualsiasi browser. Per accedere alle pagine dell'applicazione web, è necessario utilizzare l'**indirizzo IP** della macchina server e conoscere la **porta** su cui è mappato il container Docker che ospita il servizio. Nella *Figura 4.5* è mostrato un esempio di accesso all'applicazione web con utilizzo dell'indirizzo IP del server. Un'interessante caratteristica dell'applicativo è la possibilità di accedere alla rete locale in cui è mappato il server da qualsiasi dispositivo presente nella rete stessa. Ciò consente di creare un paradigma Client-Server in cui è possibile separare l'esecuzione del codice Java per la logica del back-end dall'esecuzione del codice di front-end HTML.

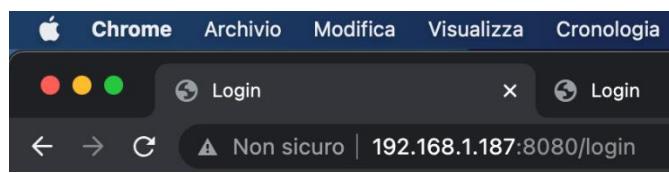


Figura 4.5 – Accesso pagina Login

Una volta avviata l'applicazione, verrà visualizzata la **schermata di login**, nella quale l'utente avrà diverse opzioni a disposizione: è possibile, infatti, inserire le proprie credenziali e accedere all'applicazione, oppure, nel caso in cui le credenziali fossero state dimenticate, si fornisce la possibilità di **ripristinare la password**. La *Figura 4.6* mostra la schermata di login, mentre la *Figura 4.7* mostra la schermata di recupero delle credenziali.

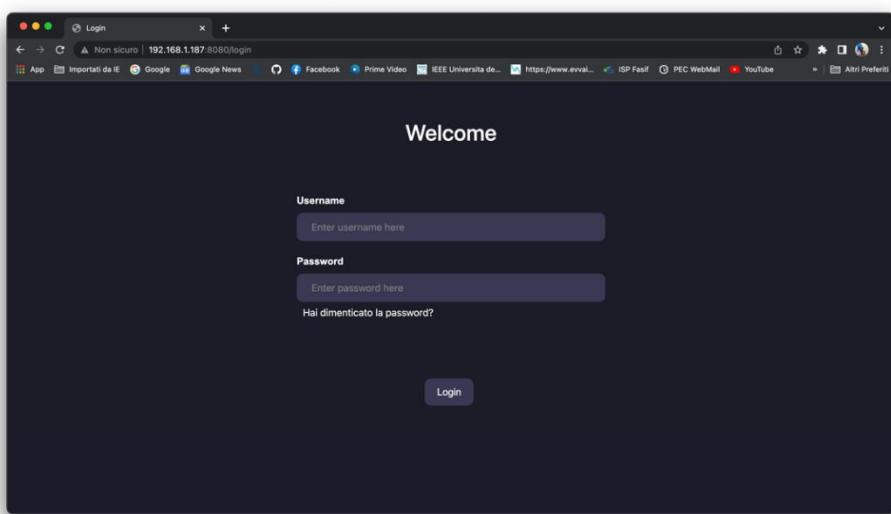


Figura 4.6 – Schermata Login

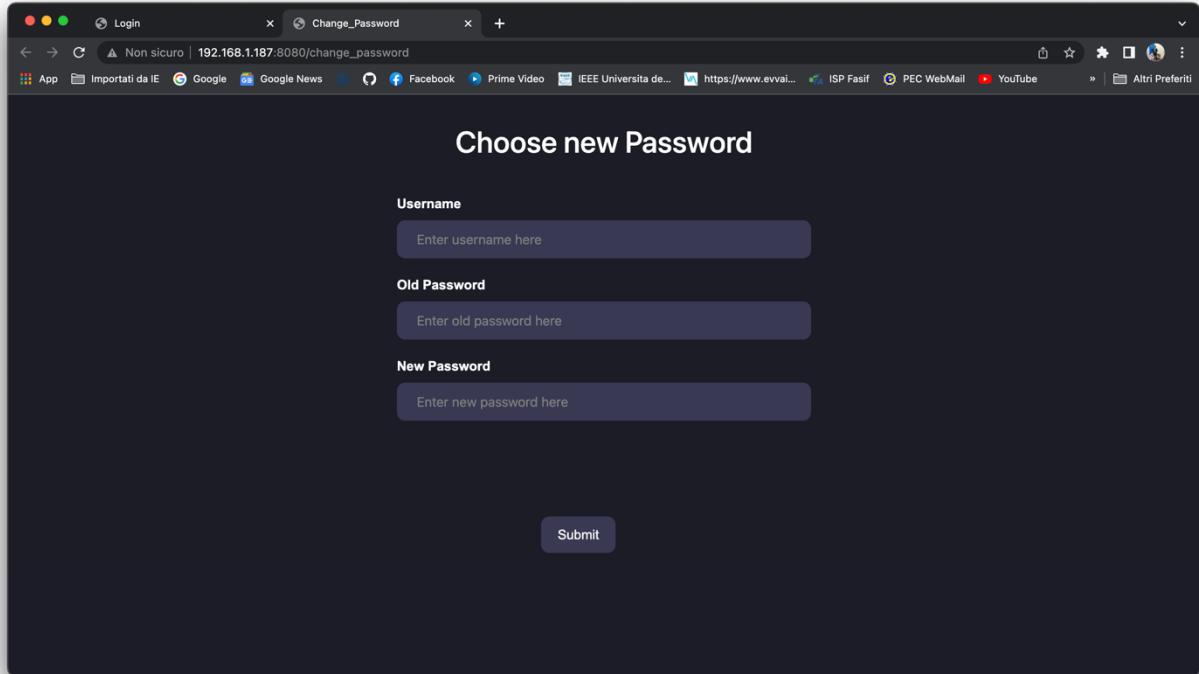


Figura 4.7 – Schermata recupero credenziali

Dopo aver effettuato l'accesso, l'utente ha la possibilità di **selezionare la classe e l'avversario** con cui desidera giocare. Nella schermata di selezione sono presenti anche due pulsanti. Il pulsante "**Download**" consente di scaricare la classe selezionata, mentre il pulsante "**Submit**" permette di confermare la scelta effettuata e passare alla pagina successiva. La *Figura 4.8* mostra quanto appena descritto.

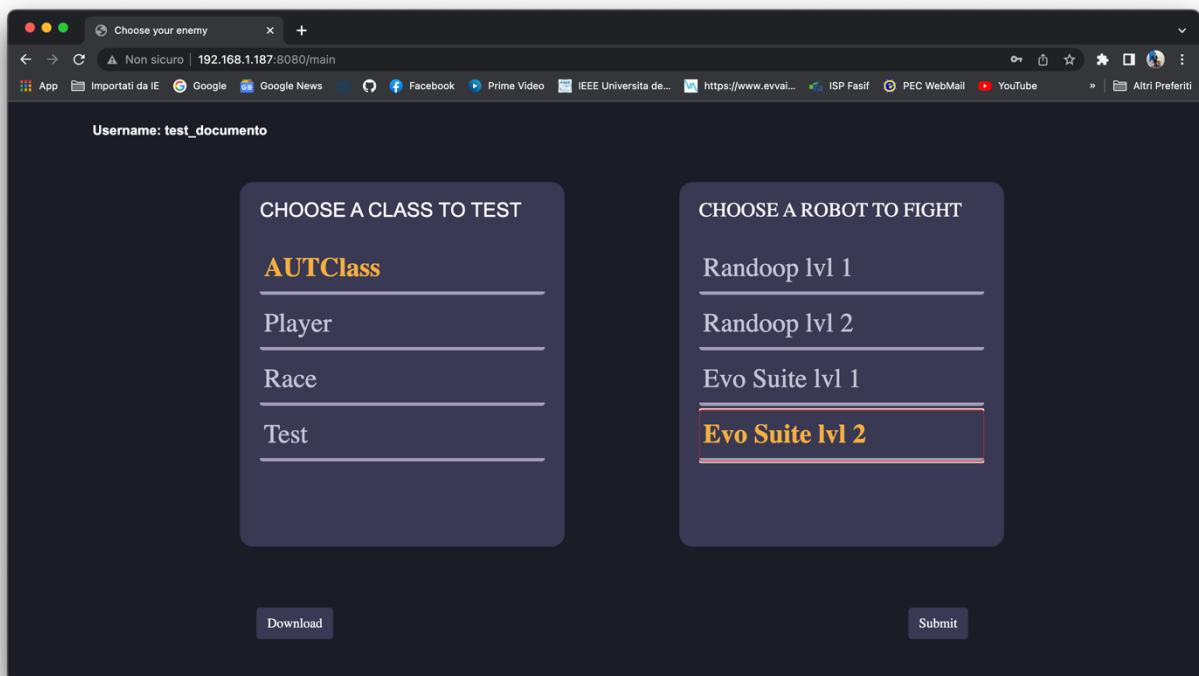


Figura 4.8 – Schermata pagina classi e robot

Una volta confermata la selezione da parte dell'utente, il server lo reindirizza a una pagina di **report**. In questa pagina, gli verrà data la possibilità di **confermare** la sua scelta o di effettuare un **rollback**, tornando alla schermata di selezione delle classi. Nel caso in cui si opti per la conferma, la partita viene salvata e l'editor viene avviato. La *Figura 4.9* rappresenta la schermata di report.

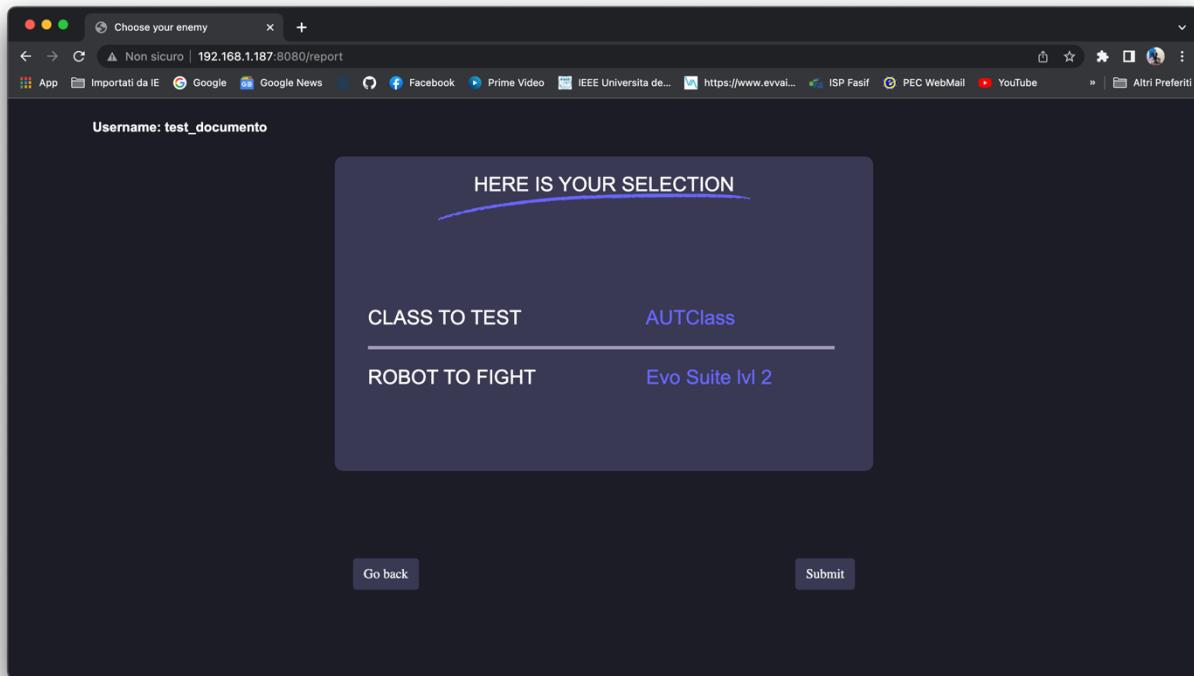


Figura 4.9 – Schermata report

Una volta avviato l'**editor** implementato dal T6-G12, l'utente può usufruire di diverse funzionalità. Queste includono la **visualizzazione del GameID** assegnato alla partita appena creata e un **riepilogo** delle impostazioni precedentemente selezionate. Inoltre, viene fornita la possibilità di scrivere nuove classi di test da inserire nel file system e di scrivere i test correlati alle classi da testare. La *Figura 4.10* e la *Figura 4.11* mostrano quanto appena descritto.

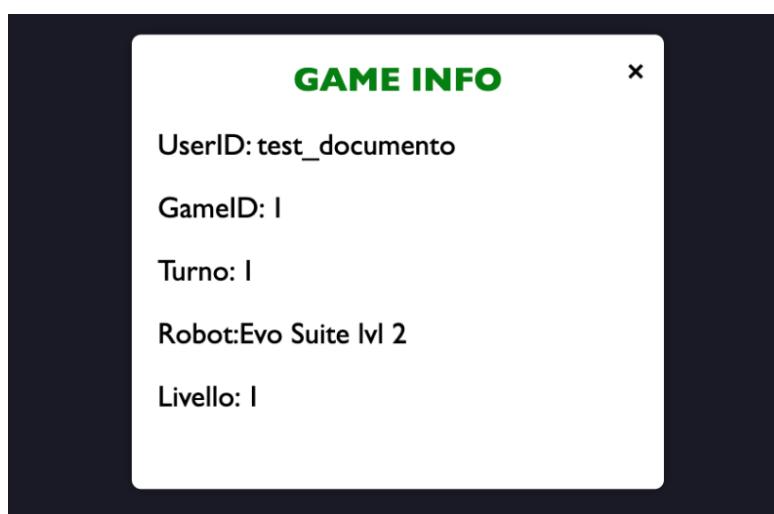


Figura 4.10 – Informazioni sulla partita in corso

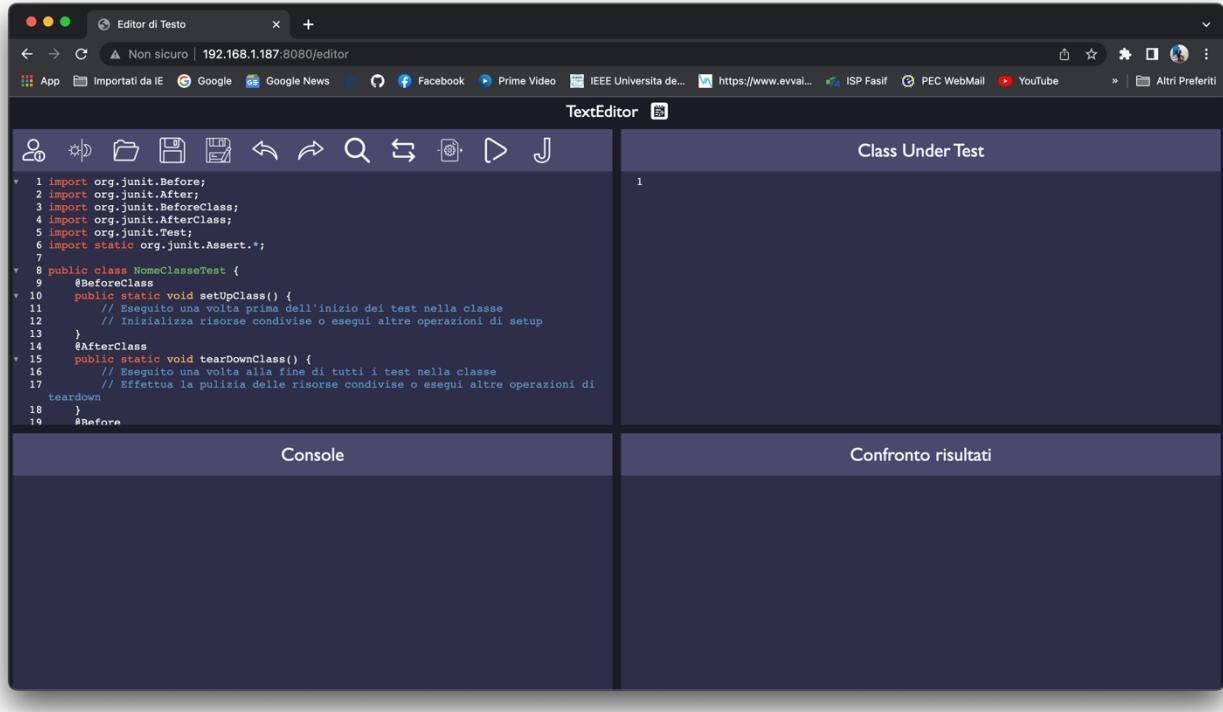


Figura 4.11 – Schermata editor

4.4 Diagramma di Installation View

Gli installation view diagram in UML sono un tipo di diagramma che descrive l'architettura di un sistema distribuito e ne illustra la disposizione fisica dei componenti hardware e software.

Questi diagrammi mostrano come i componenti del sistema sono distribuiti su nodi fisici, come server, dispositivi di rete o macchine virtuali. Sono utili per comprendere come il sistema viene installato e distribuito all'interno di un ambiente fisico.

Nel caso specifico, all'interno dell'installation view viene effettuata una sorta di decompressione del file “.jar”, al fine di visualizzarne il contenuto. Inoltre, il diagramma fornisce una migliore comprensione dell'integrazione che è stata realizzata con l'editor per visualizzare i framework e i file che sono stati integrati a livello di visualizzazione, in relazione al task implementato.

In Figura 4.12 viene riportato l'installation view diagram appositamente realizzato.

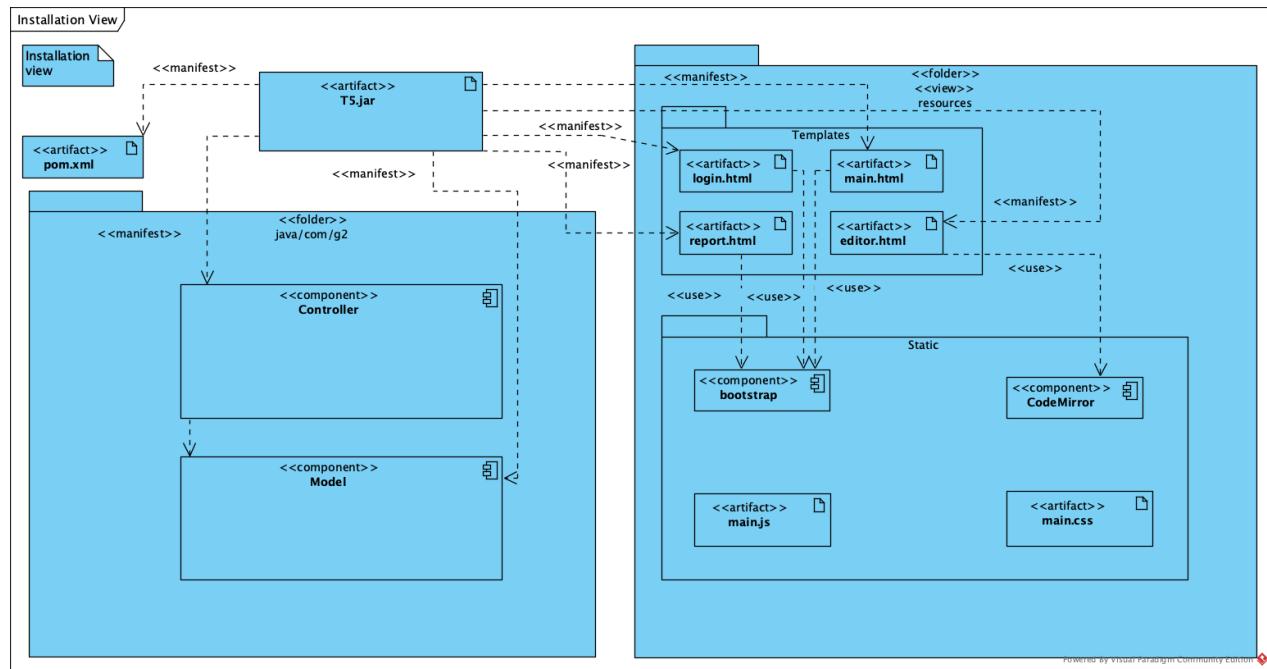


Figura 4.12 – Diagramma di Installation View

5 Testing

Nel contesto del testing della web app implementata è stato adottato l'uso di **Selenium**, grazie al quale è possibile eseguire test accurati e affidabili per la verifica del corretto funzionamento delle diverse funzionalità del software.

5.1 Selenium

Selenium è un tool ampiamente utilizzato per l'**automazione dei test funzionali** su applicazioni web, consentendo di automatizzare le interazioni con il browser per eseguire test ripetibili, migliorare la qualità del software e accelerare i processi di sviluppo. Selenium indica in realtà una **suite**, composta da diversi strumenti:

- **Selenium IDE** (prima conosciuto come Selenium Recorder) è un ambiente di sviluppo integrato completo per i test, che consente di creare, registrare e fare il debug dei test. Viene fornito come estensione Chrome e Firefox.
- **Selenium Builder** è un'estensione che traduce il comportamento utente in comandi per strutturare i test Selenium e renderne più veloce la creazione.
- **Selenium WebDriver**, utilizzato per il test della Web App, è l'API principale di Selenium, che consente di interagire con il browser in modo programmatico. WebDriver offre metodi per trovare elementi nella pagina, eseguire azioni come click, inserimento di testo, navigazione tra le pagine e gestione dei cookie.
- **Selenium Grid** potenzia WebDriver, consentendo di eseguire contemporaneamente test su diverse macchine, riducendo il tempo necessario per l'esecuzione su più browser e sistemi operativi.

Selenium fornisce un linguaggio di dominio (DSL), in quanto, per automatizzare l'interazione con il browser, c'è bisogno delle features che fornisce un DSL, come ad esempio l'astrazione. In questo modo è possibile scrivere codice con Selenium indipendentemente dal browser settato nel WebDriver. Inoltre, un DSL risulta utile per la scrittura di test su alcuni tra i maggiori linguaggi di programmazione, tra cui Java, C # e Python. Si tratta di uno strumento multipiattaforma, che può essere eseguito su Windows, Linux e Mac.

5.2 Configurazione Selenium con Spring Boot

Il primo step per configurare Selenium con Spring Boot è quello di **aggiungere delle dependency** nel file “pom.xml”.

```

<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>

    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-chrome-driver</artifactId>

        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>io.github.bonigarcia</groupId>
        <artifactId>webdrivermanager</artifactId>
        <version>4.4.3</version>
    </dependency>

```

Figura 5.1 – Dependency nel file “pom.xml” per l'utilizzo di Selenium

Le prime due sono dependency strettamente relative a Selenium, mentre la terza serve per gestire l'utilizzo di uno specifico driver, che cambia a seconda del browser che si vuole testare. Questo passaggio è fondamentale per l'automatizzazione dei test.

E' necessario, quindi, scaricare il WebDriver del browser d'interesse. Nel caso specifico, è stato utilizzato Google Chrome. Un dettaglio importante è quello di effettuare il download della versione del driver associata all'attuale versione del browser. I test di Selenium sono stati eseguiti utilizzando JUnit come framework. Un esempio è riportato di seguito:

```

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.DEFINED_PORT)
class T5ApplicationTests {

    private WebDriver driver;

    @BeforeEach // il metodo setup deve essere eseguito prima di ogni test
    public void setUp() {
        // Imposta il percorso del driver di Chrome
        System.setProperty("webdriver.chrome.driver", "src/test/java/drivers/chromedriver.exe");

        // Crea un'istanza del WebDriver per Chrome
        ChromeOptions options = new ChromeOptions();
        driver = new ChromeDriver(options);
    }

    @AfterEach // il metodo tearDown deve essere eseguito dopo ogni test
    public void tearDown() {
        // Chiudi il driver e il browser
        driver.quit();
    }
}

```

```

@Test
public void testLogin() {
    // Naviga alla pagina di login
    driver.get("http://localhost:8080/login");

    // Trova il campo di input dell'username e immetti l'username
    WebElement usernameInput = driver.findElement(By.id("username"));
    usernameInput.sendKeys("Test");
    try { // sleep messe solo per una cosa prettamente grafica
        Thread.sleep(1000);
    } catch(InterruptedException e) {
        e.printStackTrace();
    }
    // Trova il campo di input della password e immetti la password
    WebElement passwordInput = driver.findElement(By.id("password"));
    passwordInput.sendKeys("test");

    try { // sleep messe solo per una cosa meramente grafica
        Thread.sleep(1000);
    } catch(InterruptedException e) {
        e.printStackTrace();
    }

    // Trova il pulsante di login e premilo
    WebElement loginButton = driver.findElement(By.className("login-button"));
    loginButton.click();

    // Bisogna disattivare gli alert dalla pagina di login quando vengono inviati i dati
    // attendi che la pagina successiva si carichi completamente
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
    wait.until(ExpectedConditions.titleContains("Choose your enemy"));

    String expectedNextPage = "Choose your enemy";
    String actualNextPage = driver.getTitle();
    System.out.println(actualNextPage);

    assertTrue(actualNextPage.equals(expectedNextPage));
}

```

Il test introdotto crea un'istanza di Chrome in cui viene caricata la schermata di login, vengono immessi username e password e viene premuto il tasto di login. Tramite `wait.until()`, si attende il caricamento della pagina main, e dopodiché si fa corrispondere la stringa `ExpectedNextPage` al titolo della pagina che ci si aspetta venga caricata e la stringa `actualNextPage` al `driver.getTitle()`, così da ottenere il titolo della pagina attuale.

Successivamente, viene effettuata una chiamata al metodo `assertTrue()`. Se `actualNextPage` è uguale a `expectedNextPage`, l'asserzione `assertTrue` risulterà essere vera e il test avrà esito positivo. Nel caso in cui le pagine siano diverse, l'asserzione sarà falsa e il test avrà esito negativo.

Quando si è pronti per eseguire il test, è necessario inserire il comando "**mvn test**" nella console. Dopodiché, il test sarà avviato e, se l'esito è positivo, verrà restituito il seguente risultato:

```
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 15.951 s - in com.g2.t5.  
[INFO]  
[INFO] Results:  
[INFO]  
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0  
[INFO]  
[INFO]  
[INFO] BUILD SUCCESS  
[INFO]  
[INFO] Total time: 25.198 s  
[INFO] Finished at: 2023-07-13T02:37:19+02:00  
[INFO]
```

Se si cambiasse la variabile *expectedNextPage* con un titolo diverso, il test avrebbe esito negativo e l'output sarebbe il seguente:

```
[ERROR] Tests , Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 15.9 s <<< FAILURE! - in com.g2.t5.  
[ERROR] com.g2.t5.T5ApplicationTests.testLogin Time elapsed: 8.778 s <<< FAILURE!  
org.opentest4j.AssertionFailedError: expected: <true> but was: <false>  
    at com.g2.t5.T5ApplicationTests.testLogin(T5ApplicationTests.java:89)  
  
[INFO]  
[INFO] Results:  
[INFO]  
[ERROR] Failures:  
[ERROR]   T5ApplicationTests.testLogin:89 expected: <true> but was: <false>  
[INFO]  
[ERROR] Tests run: 1, Failures: 1, Errors: 0, Skipped: 0  
[INFO]  
[INFO]  
[INFO] BUILD FAILURE  
[INFO]  
[INFO] Total time: 25.226 s  
[INFO] Finished at: 2023-07-13T02:40:28+02:00
```