

UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II



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

Documentazione del progetto di Software Architecture Design

TASK 5 – G14

Anno Accademico 2022-2023

Membri del gruppo:

Pitacoro Lorenzo M63001499

Russo Antonio M63001518

Russo Lavinia M63001479

Togna Martina M63001476

Toscano Stefano M63001537

Indice

TASK 5 – G14	1
1. Introduzione	4
1.1. Descrizione del task	4
1.2. Tecnologie utilizzate	4
1.2.1. Spring	4
1.2.2. Maven	4
1.2.3. Thymeleaf	5
1.2.4. Tomcat	5
1.2.5. Jira Cloud, Microsoft Teams e GitHub	5
1.2.6. Visual Paradigm	6
1.2.7. Docker	6
1.3. Scrum	7
2. Documentazione di Analisi	8
2.1. Glossario dei termini	8
2.2. Revisione dei Requisiti	9
2.2.1. Requisiti funzionali:	9
2.2.2. Requisiti non funzionali:	9
2.3. Storia Utente	10
2.4. System Domain Model	10
2.5. Diagramma di Sequenza	11
3. Documentazione di Progettazione	13
3.1. Diagramma delle Classi	13
3.2. Diagramma dei Package	14
3.3. Diagramma di Sequenza	16
3.4. Diagramma dei Componenti	18
3.5. Diagramma delle Interfacce	19
4. Descrizione dell'Implementazione	20
4.1. Package	20
4.1.1. com.testingGameT5.demo	20
4.1.2. com.testingGameT5.controller	20
4.1.3. com.testingGameT5.model	22
4.2. Resources	23
4.3. Ulteriori file di interesse	24
5. Test	25
5.1. Testing API tramite Postman	29
5.1.1. API che ritorna la Pagina di Start	29

5.1.2.	API che ritorna la Pagina di Login	29
5.1.3.	API che ritorna la Pagina di Avvio Editor.....	30
5.1.4.	API di Inserimento delle Credenziali	31
5.1.5.	API di Scelta delle Classi da Testare.....	32
5.1.6.	API di Creazione della Partita	33
6.	Diagramma di Deployment e Guida all'Istallazione	34
6.1.	Diagramma di Deployment	36

1. Introduzione

1.1. Descrizione del task

Requisiti sull'avvio del Primo Scenario di Gioco

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. Tecnologie utilizzate

Nello sviluppo di questo progetto, sono state utilizzate le tecnologie illustrate di seguito per semplificare lo sviluppo e facilitare l'organizzazione del lavoro.

1.2.1. Spring

Per lo sviluppo della nostra applicazione Web in Java abbiamo utilizzato il framework Spring. In particolare, abbiamo usufruito del modulo “Spring MVC”, basato sul pattern architetturale Model-View-Controller. Un pattern architetturale è un insieme di decisioni di progettazione architetturale che sono applicabili ad un problema di progettazione ricorrente. Il pattern MVC, nello specifico, disaccoppia il Model e la View tramite un protocollo di sottoscrizione-notifica, dove il Controller si occuperà di mappare gli input della View in azioni eseguite dal Model, implementando in questo modo la logica di controllo dell'applicazione. In particolare, Spring MVC prevede l'aggiunta del Dispatcher Servlet che si avvale di due componenti: l'Handler mapping, il quale si occupa di inoltrare la richiesta utente al controller più adatto per la sua gestione; il View Resolver si occupa di effettuare il rendering della vista richiesta dal controller. Infine, abbiamo configurato l'ambiente di sviluppo tramite il tool Spring Boot e l'ausilio dell'IDE di sviluppo Eclipse e SPRING TOOLS 4.

1.2.2. Maven

Maven è uno strumento di *build automation* che abbiamo utilizzato per semplificare la gestione delle dipendenze della nostra applicazione e per semplificare la fase di build. Infatti, ogni volta che si è ritenuto necessario, abbiamo modificato le dipendenze tramite la configurazione e l'aggiornamento del file “pom.xml” presente nel nostro progetto.

1.2.3. Thymeleaf

Una delle tecnologie che Spring Boot ci offre è Thymeleaf, un motore di template HTML, XML, CSS, JavaScript. Per poter utilizzare Thymeleaf all'interno dell'applicazione è stata inserita una dipendenza all'interno del file "pom.xml".

1.2.4. Tomcat

Tomcat è un server web, integrato in un'applicazione Spring Boot, utilizzato come Dispatcher delle richieste http ricevute dal cliente. Esso, infatti, riceverà le richieste dai client e le inoltrerà al controller corretto; appena la risposta del Controller sarà disponibile, Tomcat si occuperà di inviarla al client.

1.2.5. Jira Cloud, Microsoft Teams e GitHub

Per lo sviluppo della nostra applicazione abbiamo utilizzato strumenti per la condivisione e organizzazione del lavoro come Microsoft Teams e GitHub, e il software di project management Jira Cloud. Tali tools sono stati fondamentali per semplificare il coordinamento e pianificazione delle attività, per la condivisione delle risorse e per garantire la pratica agile "Collective Ownership". Infatti, avendo condiviso tutto il materiale tramite questi strumenti, tutti i membri del gruppo conoscono a pieno il progetto e ogni membro ha accesso al codice del programma.

Per quanto riguarda Jira, abbiamo pianificato tutte le attività da svolgere durante le iterazioni (Sprint in Scrum).

Si riportano di seguito le quattro iterazioni in Jira (quattro perché abbiamo considerato la stesura finale del progetto come un'ultima iterazione).

Sprint completato	T5S-5 Stesura glossario dei termini
	T5S-6 Stesura user story e suddivisione del task in sottotask
	T5S-7 Creazione diagramma delle classi
	T5S-8 Creazione diagramma di sequenza
	T5S-9 Analisi delle tecnologie da utilizzare per future iterazioni
	T5S-10 Stesura di una relazione e creazione di una presentazione del lavoro svolto
Sprint completato	T5S-11 Studio SpringMVC
	T5S-12 Prototipo implementazione
	T5S-13 Raffinamento diagramma di sequenza secondo il pattern SpringMVC
	T5S-14 Configurazione ambiente di lavoro
Sprint completato	T5S-15 Raffinamento Sequence Diagram
	T5S-16 Finalizzazione interfacce con altri task
	T5S-17 Implementazione delle funzionalità richieste
	T5S-18 Risoluzione di bug introdotti nella precedente iterazione
Sprint completato	T5S-19 Risoluzione di bug introdotti nella precedente iterazione
	T5S-20 Sviluppo di classi mock per il testing del sistema
	T5S-21 Testing delle interfacce grafiche
	T5S-22 Raffinamento finale della documentazione

Descriviamo brevemente quanto fatto durante le iterazioni:

1. Nella **prima iterazione** (04/04/2023 – 18/04/2023) ci siamo prefissati di chiarire i requisiti del task, fornire una versione preliminare dell'architettura e identificare i modi in cui il task in oggetto possa interfacciarsi con altri task per la richiesta di servizi. Come si può vedere dalla foto precedente, i task da T5S-5 a T5S-10 rappresentano le attività svolte in questa iterazione.
2. Come obiettivi della **seconda iterazione** (24/04/2023 – 08/05/2023) abbiamo approfondito il diagramma di sequenza con il pattern Spring MVC e realizzato un'implementazione parziale del task con lo scopo di illustrarne il funzionamento. Le attività relative alla seconda iterazione corrispondono ai T5S-11/T5S-14 di Jira.
3. Per la **terza iterazione** (23/05/2023 – 06/06/2023), invece, abbiamo continuato l'implementazione del task, ci siamo interfacciati con gli altri gruppi al fine di rendere l'implementazione compatibile con gli altri task in vista della realizzazione del progetto complessivo. Le attività svolte in questa iterazione sono riportate in Jira a partire dal task T5S-15 fino al task T5S-18.
4. La **quarta iterazione** (29/06/2023 – 13/07/2023) rappresenta l'iterazione conclusiva, in cui abbiamo effettuato le ultime modifiche necessarie per portare a termine il progetto, completato la documentazione e raffinato le interazioni con gli altri Task, testate tramite classi Mock. I task T5S-20/T5S-22 di Jira corrispondono alle attività ultimate durante quest'ultima iterazione.

Infine, abbiamo utilizzato GitHub per la condivisione del codice sviluppato e soprattutto per la collaborazione con gli altri gruppi. Infatti, non è stato sempre possibile concordare “faccia a faccia” le interfacce con i gruppi, abbiamo quindi usato GitHub per esaminare il codice degli altri Task e selezionare le interfacce di cui avevamo bisogno, come ad esempio nel caso del task 4 con il gruppo G04 per l'interfaccia di “CreaPartita”.

1.2.6. Visual Paradigm

Visual Paradigm è uno strumento di modellazione e progettazione dei diagrammi UML, che abbiamo utilizzato per mostrare il nostro sistema attraverso viste diverse, sia statiche che dinamiche, a diversi livelli di dettaglio.

1.2.7. Docker

Docker è una piattaforma utilizzata per facilitare la distribuzione e l'esecuzione di applicazioni all'interno di un container. Per approfondire l'uso di Docker nello sviluppo della nostra applicazione si rimanda alla sezione dell'installazione.

1.3. Scrum

Per gestire il lavoro in gruppo in modo organizzato e sperimentare alcune delle pratiche più comuni dello sviluppo software, si è deciso di servirsi del framework Scrum e di alcune pratiche di sviluppo Agile, come le storie utente e il team programming.

Per quanto riguarda Scrum, esso può essere definito come un framework agile per gestire il ciclo di sviluppo del software in maniera iterativa; dal momento che non si tratta di una tecnica o di un processo di sviluppo software, esso fornisce delle linee guida generali senza vincolare eccessivamente i team adibiti allo sviluppo delle applicazioni. Ciò è un elemento fondamentale della filosofia di Scrum, che pone i team stessi al centro del processo di sviluppo e non gli stakeholders o i project manager. Scrum prevede tre fasi:

- Nella fase iniziale si stabiliscono gli obiettivi del progetto e l'architettura del software;
- Nei cicli di sprint il software viene effettivamente sviluppato per incrementi successivi;
- Nella fase di chiusura il progetto e la relativa documentazione sono ultimati e si esamina ciò che si è appreso a valle del lavoro svolto, sia in termini di successi che di fallimenti e problematiche incontrate nello sviluppo.

In Scrum, la gestione di ciò che bisogna sviluppare viene effettuata tramite due elementi fondamentali: il Product Backlog e lo Sprint Backlog. Il primo viene redatto ed aggiornato dal Product Owner mentre il secondo è gestito dal team di sviluppo e contiene gli Item prelevati dal Product Backlog che verranno sviluppati nello Sprint corrente. Ogni Sprint di un progetto ha una durata fissa (solitamente pari a due o tre settimane) ed è regolato da diversi eventi, come le riunioni giornaliere (Daily Scrum, un evento time-boxed dalla durata molto breve).

Nel caso di questo progetto, le fasi di sviluppo seguite dalle Iteration Review hanno avuto una durata pari a due settimane e hanno compreso lo sviluppo di parti dell'applicazione prodotta, con annessa documentazione; in particolare, inizialmente ci si è concentrati sul comprendere in modo chiaro il task tramite l'utilizzo di storie utente e diagrammi di analisi. Successivamente, questi sono stati approfonditi, portando alla produzione dei diagrammi di progettazione; parallelamente alla produzione della documentazione, in accordo con le metodologie di sviluppo agile, sono stati sviluppati diversi prototipi dell'applicazione da produrre, in modo da avere un ulteriore mezzo per riscontrare se la valutazione iniziale dei requisiti fosse adatta a quanto richiesto e se fosse necessario effettuare delle modifiche. Dopo le Iteration Review tenutesi durante il corso, che hanno costituito la conclusione dei primi tre Sprint, una parte fondamentale dello sviluppo di questo progetto è stata la Retrospectiva dei vari Sprint, ossia una riunione tenuta dal team in cui si è discusso di cosa mantenere e cosa migliorare per quanto riguarda il processo di lavoro per i successivi Sprint.

Per quanto riguarda altre pratiche adottate che non sono prettamente legate al framework Scrum, è stata data grande importanza alle storie utente come strumento di definizione e specifica dei requisiti, poi ulteriormente formalizzati tramite la suddivisione in requisiti funzionali e non funzionali, e alla tecnica di team programming: la maggior parte del codice prodotto è infatti frutto del lavoro collettivo di tutto il team e questo ha portato a tempi di sviluppo contenuti senza inficiare sulla qualità del software prodotto.

2. Documentazione di Analisi

2.1. Glossario dei termini

Il Glossario dei termini consente di individuare i principali oggetti del dominio applicativo. Specificandone il significato e i sinonimi, è possibile chiarire le classi concettuali a cui si farà riferimento successivamente nel presente documento.

Termine	Descrizione	Sinonimi
Giocatore	Un generico utente che dopo aver effettuato il login avvia una nuova partita.	Studente
Partita	Una sessione di gioco avviata e configurata da un giocatore che è caratterizzata da un identificativo.	
Primo Scenario	Una modalità di gioco caratterizzata da singolo turno, singolo avversario e singola classe da testare.	
Classe	Una classe Java che può essere scelta dal giocatore per una determinata partita.	Classe da testare
Robot	L'avversario artificiale contro cui confrontarsi.	
Ambiente di editing	Un editor di testo che consente la visualizzazione della classe da testare e fornisce la possibilità di scrivere il codice per il testing.	

A questa prima analisi, è possibile aggiungere ulteriori termini, in vista delle future possibili estensioni di tale applicativo.

Termine	Descrizione	Sinonimi
Secondo Scenario	Una modalità di gioco caratterizzata da multipli turni (numero prefissato), singolo avversario e singola classe da testare.	
Turno	Sessione interna ad una partita opportunamente configurata dall'utente.	Round
Terzo Scenario	Una modalità di gioco caratterizzata da multipli turni (numero prefissato), multipli giocatori e singola classe da testare.	
Giocatore Guest	Un utente che viene invitato a partecipare ad una partita configurata da un altro giocatore (Host).	
Giocatore Host	Un giocatore che avvia una partita contro un robot o un altro giocatore, dopo aver effettuato il login.	

Giocatore Guest e Giocatore Host vengono introdotti nel Terzo Scenario per discriminare chi avvia la partita da chi vi partecipa soltanto; pertanto, Giocatore Host coincide con il Giocatore nei primi due scenari.

In questo documento, si porterà a compimento soltanto il Primo Scenario, ma si predispone comunque il sistema al fine di renderlo adattabile a tali ulteriori estensioni (evolubile).

2.2.Revisione dei Requisiti

Dalla specifica informale si sono ricavati innanzitutto i requisiti funzionali, ossia le funzionalità che il sistema software che verrà sviluppato dovrà offrire. Sono poi stati individuati i requisiti non funzionali che individuano caratteristiche che dovranno avere determinati oggetti del dominio e requisiti sull'accessibilità delle diverse funzionalità.

2.2.1. Requisiti funzionali:

- 1. Il sistema deve consentire al giocatore di autenticarsi.*
- 2. Il sistema deve consentire al giocatore che si è precedentemente autenticato di avviare una nuova partita.*
- 3. Il sistema deve consentire al giocatore di configurare la partita del Primo Scenario, ossia scegliere la classe da testare e il robot da sfidare.*
- 4. Il sistema crea la partita sulla base delle scelte fatte dal giocatore, le associa un identificativo e ne effettua il salvataggio.*
- 5. Il sistema deve avviare l'ambiente di editing in cui viene visualizzata la classe da testare e la finestra in cui il giocatore può scrivere la classe di test.*

2.2.2. Requisiti non funzionali:

- 1. La configurazione della partita deve essere consentita solo al giocatore che si è autenticato.*
- 2. Un giocatore autenticato è caratterizzato da un identificativo.*
- 3. Ciascuna partita deve essere caratterizzata da un identificativo e un numero di round.*

2.3. Storia Utente

Per individuare gli aspetti di interesse all'utente finale è stata inoltre scritta la seguente storia utente.

Come giocatore voglio poter avviare una partita

Il giocatore richiede di avviare una partita.

Il sistema mostra al giocatore la finestra per effettuare il login, finché quest'operazione non va a buon fine non deve essere possibile usufruire delle altre funzionalità di questo task.

Al giocatore autenticato, il sistema mostra una lista di classi da testare e un elenco di robot disponibili.

Il giocatore sceglie la classe da testare e il robot avversario.

Il sistema crea la partita opportunamente configurata, le associa un Id e la salva.

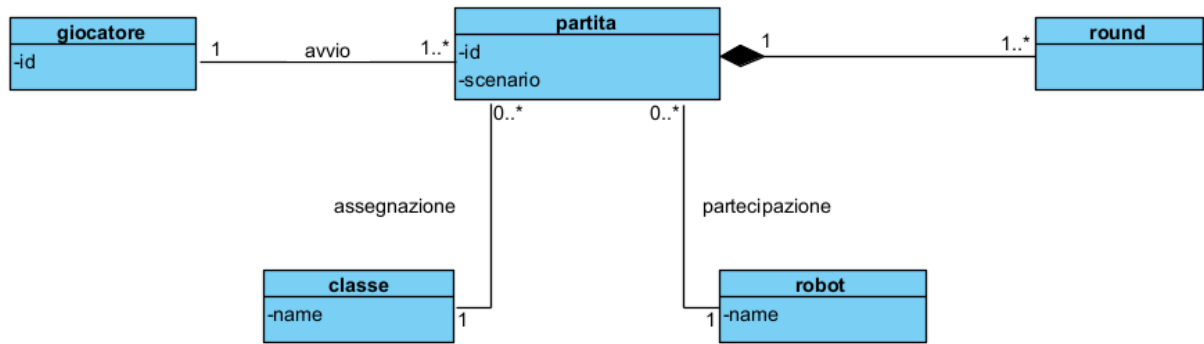
Il sistema avvia l'ambiente di editing in cui verrà mostrata la classe da testare e una finestra in cui il giocatore potrà inserire il suo codice.

2.4. System Domain Model

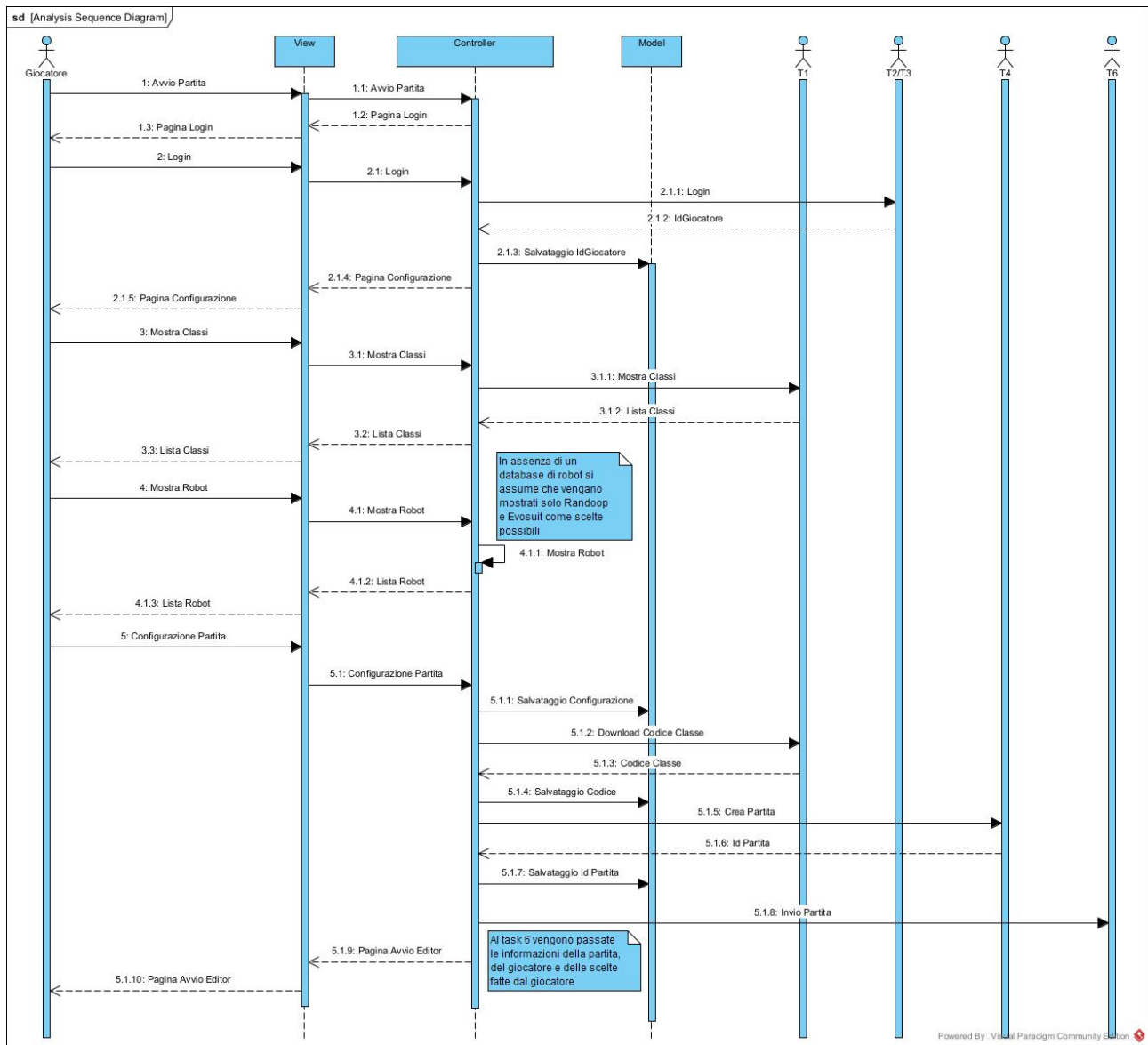
Il System Domain Model è un diagramma che consente di individuare le classi e gli attributi concettuali rilevanti del Dominio.

Il giocatore, caratterizzato da un identificativo assegnatogli a seguito dell'operazione di login, può avviare una o più partite. Nel Primo Scenario, ciascuna partita è associata ad un solo giocatore; negli scenari successivi, si assume la presenza di più giocatori, sebbene vi sia sempre un solo giocatore che avvia la partita.

La partita è caratterizzata da un identificativo e dallo scenario ed è costituita da uno o più round, il Primo Scenario è caratterizzato da un solo round. Alla partita è associata una classe e un robot che rappresentano la configurazione scelta dal giocatore.



2.5. Diagramma di Sequenza



Il presente task deve offrire la possibilità al giocatore di avviare una partita. Il giocatore interagisce tramite la view e questa prima azione ha solo l'effetto di avviare il task e mostrare una nuova view.

Per poter configurare e creare effettivamente la partita il giocatore deve essersi precedentemente autenticato, per tale motivo la prima schermata mostrata è quella per effettuare il login. Il giocatore inserisce le sue credenziali, che verranno inoltrate dal controller e gestite dai task T2 e T3, adibiti a tale servizio. Quando l'operazione va a buon fine, viene restituito l'identificativo del giocatore, che dovrà essere salvato nel model, al fine di conservare tale valore per le operazioni future.

Dopo aver effettuato il login, il giocatore vede una nuova vista che gli consente di configurare la partita. L'utente richiede prima di mostrare l'elenco delle classi, che viene costruito perché il controller richiama una funzionalità del task T1, che si occupa della gestione del database di classi, e successivamente l'elenco di robot. Date le due liste, il giocatore effettua le sue scelte che verranno salvate nel model. Il controller richiede al task T1 il codice della classe scelta e lo salva. Successivamente, tutti i dati salvati nel model vengono inviati al task T4, che si occupa del salvataggio della partita; a seguito di quest'operazione viene ritornato l'identificativo, che sarà poi salvato nel model.

Infine, tutti i dati presenti nel model, ossia l'identificativo del giocatore e della partita e le scelte dell'utente, sono inviati dal controller al task T6, che si occupa dell'editor. Il task si conclude proprio con tale operazione di avvio del task T6.

3. Documentazione di Progettazione

3.1. Diagramma delle Classi

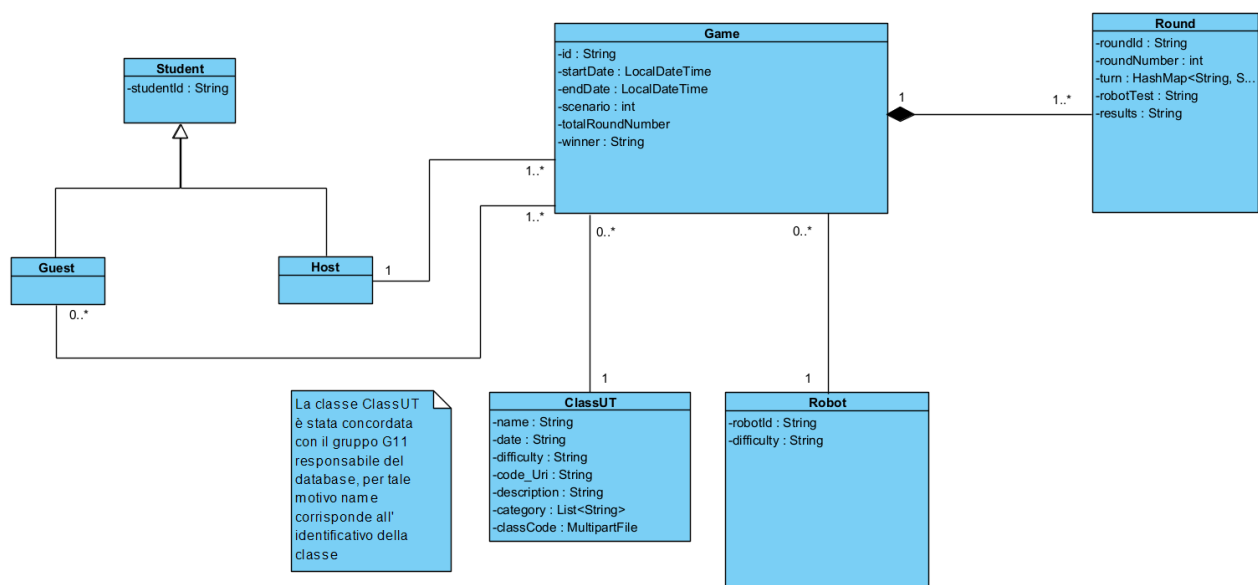
Dal raffinamento del diagramma delle classi di dominio, si è ricavato il seguente diagramma delle classi raffinato.

Per rendere compatibile il sistema in vista della futura integrazione, le seguenti classi sono state rese compatibili con i task che si occupano della persistenza di tali entità. In particolare, la classe ClassUT è stata concordata con il gruppo G11-T1, responsabile della gestione del database di classi da testare. Le altre classi, invece, sono state adeguate alle classi previste dal gruppo G4-T4, responsabili del salvataggio nel database della partita.

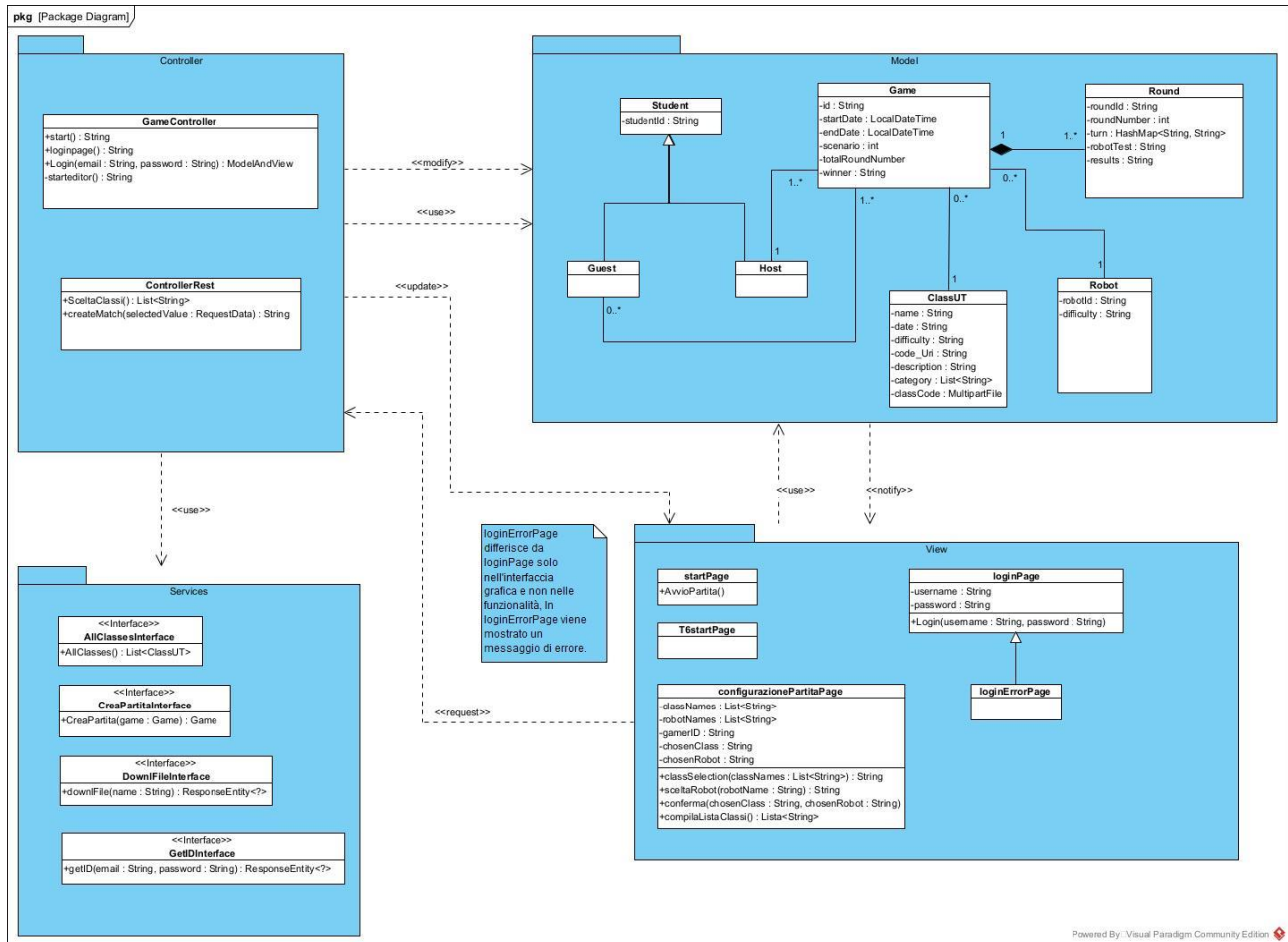
Per predisporre il sistema ai futuri scenari di gioco, si è distinto, mediante una gerarchia, il giocatore che avvia la partita da quello che viene invitato a parteciparvi, come previsto nel Terzo Scenario. Mentre ogni partita deve necessariamente avere un giocatore che l'avvia, dunque un Host, potrebbe o meno avere un Guest.

La classe ClassUT rappresenta la classe da testare, viene identificata da un nome e tra i diversi attributi presenta anche il codice, di cui bisognerà fare il download usando una funzionalità offerta dal task T1.

La classe Game è costituita di un certo numero di Round, dall'identificativo della partita, dallo scenario e da altri attributi di interesse per gli altri task. Alla classe Game, oltre allo studente, sono associati il robot contro cui sfidarsi e la classe da testare, che rappresentano la configurazione della partita.



3.2. Diagramma dei Package



Il diagramma dei Package è una vista statica del sistema che consente di individuare le relazioni che sussistono tra i diversi moduli. Il singolo package rappresenta un modulo caratterizzato da un insieme di responsabilità coeso e definisce un namespace unico.

Il Package Model è caratterizzato dall'insieme dei moduli software che incapsulano lo stato dell'applicazione. Tale Package contiene le classi e le relazioni precedentemente mostrate nel diagramma delle Classi.

Nel Package View sono presenti le diverse viste del sistema: qui le entità sono rappresentate come classi che consentono di individuare i parametri su cui lavorano le pagine e le operazioni che permettono di svolgere.

1. La pagina che si presenta all'avvio del task, ossia startPage, presenta un bottone che consente l'avvio della partita. Quest'operazione reindirizza l'utente alla pagina di login.

2. La pagina di login, ossia `loginPage`, presenta i campi username e password che vengono processati tramite l'operazione di Login; se quest'operazione va a buon fine ritorna l'identificativo del giocatore e si va avanti, altrimenti si resta nella pagina di errore (`loginErrorPage`) finché non si completa correttamente l'operazione di autenticazione. Tale pagina differisce dalla precedente soltanto nell'interfaccia grafica in quanto mostra un messaggio di errore, ma presenta le medesime funzionalità.
3. All'utente autenticato viene mostrata la pagina che consente la configurazione della partita. Tale view presenta i seguenti parametri: la lista di classi da testare e la lista di robot, tra cui l'utente può scegliere; le scelte utente (classe e robot) e l'identificativo del giocatore, che viene passato dalla pagina precedente.
4. La pagina di avvio dell'editor, invece, rappresenta la pagina finale del nostro task, non presenta alcuna funzione ad essa associata in quanto dovrà poi essere collegata al task che si occupa dell'editor, per tale motivo infatti il bottone non è cliccabile e il task termina.

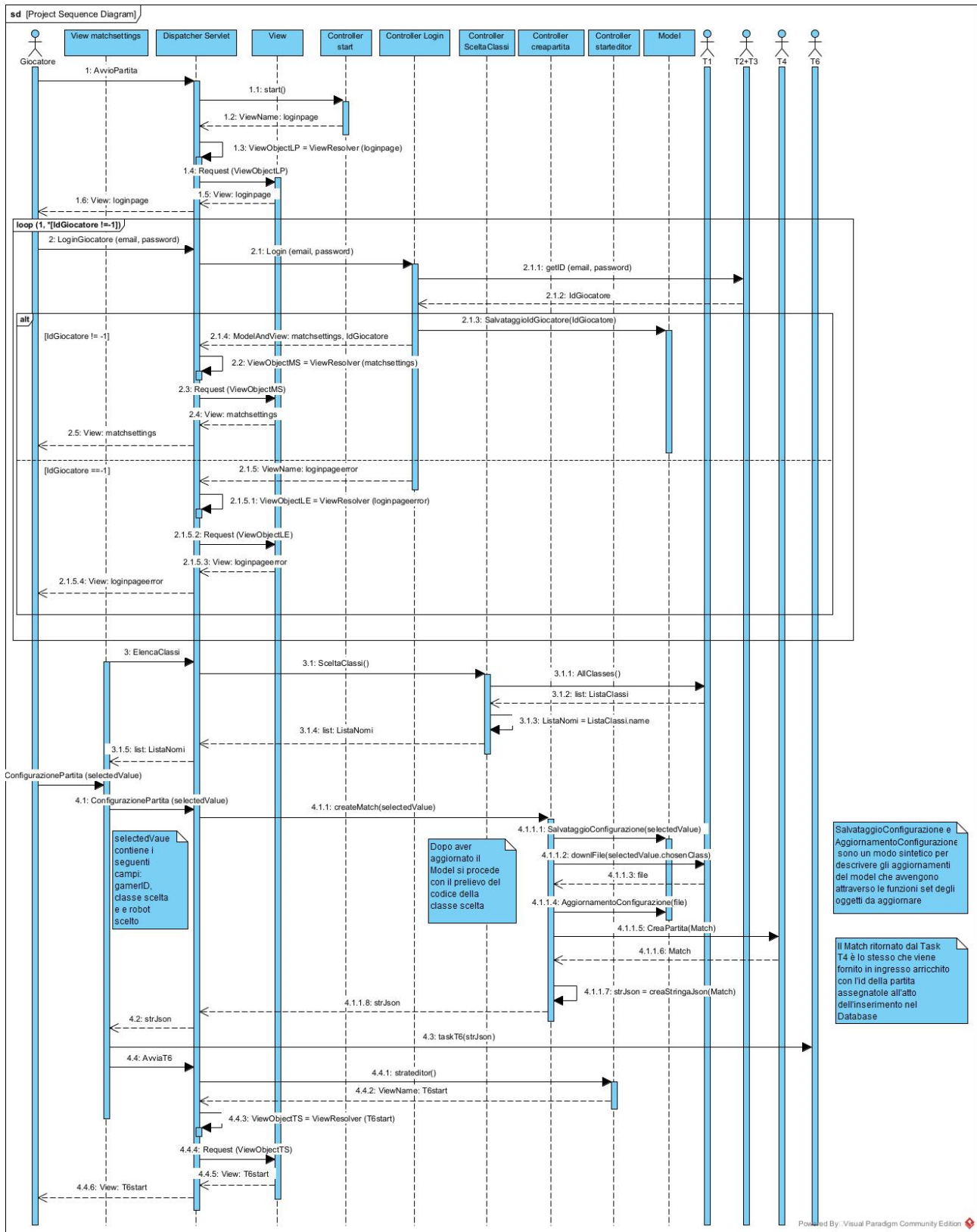
Nel Package Controller sono presenti le funzioni richiamate tramite URL, in particolare è il componente Dispatcher che, data una richiesta, la inoltra al controller adatto a servirla. Nel package si vede la distinzione tra due moduli software:

- GameController contiene le API che ritornano viste con l'aggiunta di eventuali parametri (Model and View). La funzione `start ()` infatti ritorna la vista iniziale, quella di `loginpage ()` riporta alla pagina in cui è possibile inserire le credenziali e quella di `starteditor ()` restituisce la pagina finale con il bottone non cliccabile. La funzione `Login (String email, String password)`, invece, ritorna un Model and View, infatti restituisce la vista per la configurazione dei parametri e l'identificativo del giocatore, oppure la pagina di errore se l'operazione di autenticazione non dovesse essere andata a buon fine.
- RestController contiene le API che non restituiscono viste: la funzione `SceltaClassi` restituisce la lista dei nomi delle classi da testare che servono per popolare il menù a tendina mostrato all'utente; mentre la funzione `createMatch` si occupa della creazione effettiva della partita sulla base delle scelte prese dall'utente.

Il Package Services rappresenta i moduli esterni che sono stati utilizzati e dunque la parte relativa ai servizi offerti dagli altri task, per tale motivo sono state considerate le interfacce, che verranno approfondite nel corrispondente diagramma.

Le relazioni che intercorrono tra i diversi package consentono di comprendere meglio il funzionamento dell'applicazione che risponde al pattern architetturale scelto. Infatti, il *Controller* usa e modifica il *Model*, andando ad effettuare operazione sui dati in esso contenuti. Inoltre, il *Controller* si occupa della selezione e dell'aggiornamento della *View* e usa i servizi offerti dai componenti esterni. La *View* inoltra gli input dell'utente al *Controller*, infatti lo stereotipo <<request>> indica l'operazione di reindirizzamento delle richieste utente. Infine, la *View* richiede e usa i dati del *Model*, che a sua volta le notifica eventuali cambiamenti di stato dei suoi dati.

3.3. Diagramma di Sequenza



Rispetto a quanto fatto in fase di analisi, si è adeguato tale diagramma al framework Spring MVC per implementare il pattern model-view-controller. Questo comporta l'introduzione del Dispatcher Servlet, un elemento che intercetta le richieste utente provenienti dalle viste e le inoltra al controller adibito alla loro gestione, con l'aiuto dell'Handler Mapping, componente che è stato considerato interno al Dispatcher. Inoltre, il Dispatcher, con l'aiuto del View Resolver, si occupa di selezionare la vista scelta dal controller.

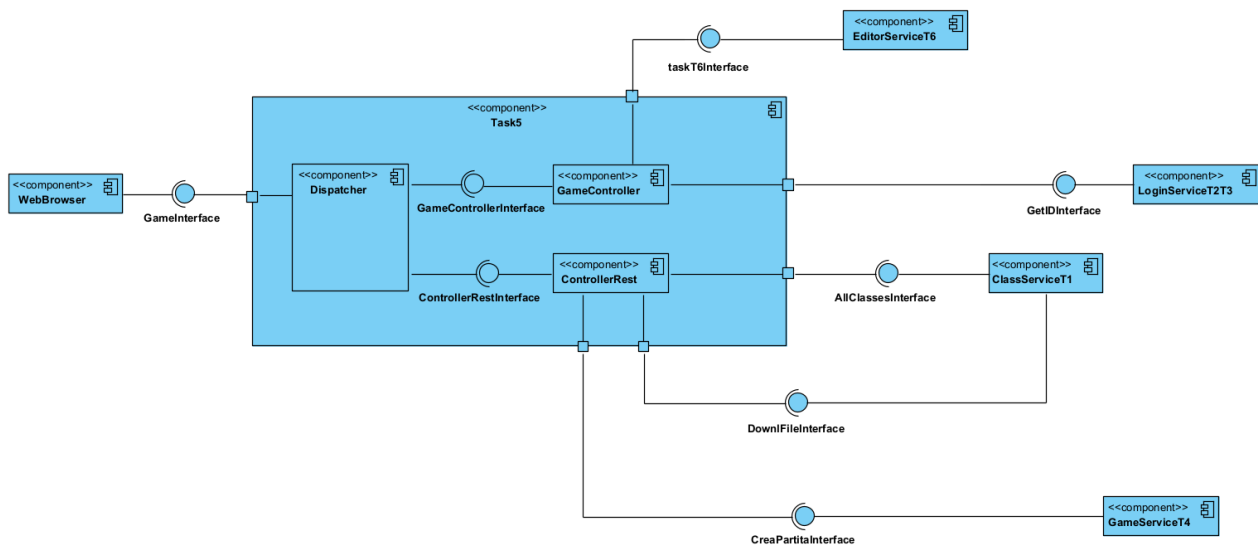
Sono stati aggiunti i controller necessari per la gestione delle azioni atomiche del task, quali l'avvio della partita, il login, la configurazione della partita, la sua creazione e l'avvio dell'editor.

In questo diagramma, si è fatta una distinzione tra "View" e "View matchsettings", in quanto adesso la "View" rappresenta una collezione di pagine html opportunamente selezionate da controller e Dispatcher. Le interazioni dell'utente con le specifiche viste sono omesse, eccetto quelle con "View matchsettings", dato che oltre a reindirizzare gli input dell'utente, effettua una serie di operazioni che richiedono una modellazione specifica e che sono approfondite nel seguito.

A seguito dell'avvio della partita viene mostrata la schermata per effettuare il login. La funzione Login che prende email e password come input viene eseguita almeno una volta e finché l'operazione non va a buon fine, come espresso tramite il blocco Loop. Se le credenziali non sono corrette, la funzione ritorna -1 e viene mostrata nuovamente la schermata di login con un messaggio di errore, altrimenti restituisce l'identificativo del giocatore e viene mostrata la schermata per la configurazione della partita.

All'attivazione della View matchsettings viene richiamata la funzione per popolare la lista di classi, tra cui il giocatore può scegliere. L'utente effettua le sue scelte che vengono inviate come *selectedValue*, una struttura contenente come campi l'identificativo del giocatore autenticato, la classe scelta e il robot selezionato. Questi valori sono salvati nel model, successivamente viene scaricato il codice della classe scelta, usando un servizio offerto dal task 1. Il model aggiornato con tale file viene inviato al task 4 per la creazione della partita. La partita, arricchita dell'identificativo a seguito della funzione fornita dal task che gestisce il database di gioco, viene salvata nel model. I dati in esso contenuti vengono trasformati in formato Json e inviati al task 6 attivato dalla vista matchsettings, che successivamente attiva il controller starteditor, che restituisce la schermata finale, in cui è possibile richiamare l'ambiente di editing.

3.4. Diagramma dei Componenti



Il Diagramma dei Componenti è una rappresentazione statica del sistema che consente di individuare i componenti principali, distinguendo quelli interni al confine del sistema da quelli esterni e le interfacce tramite cui essi comunicano.

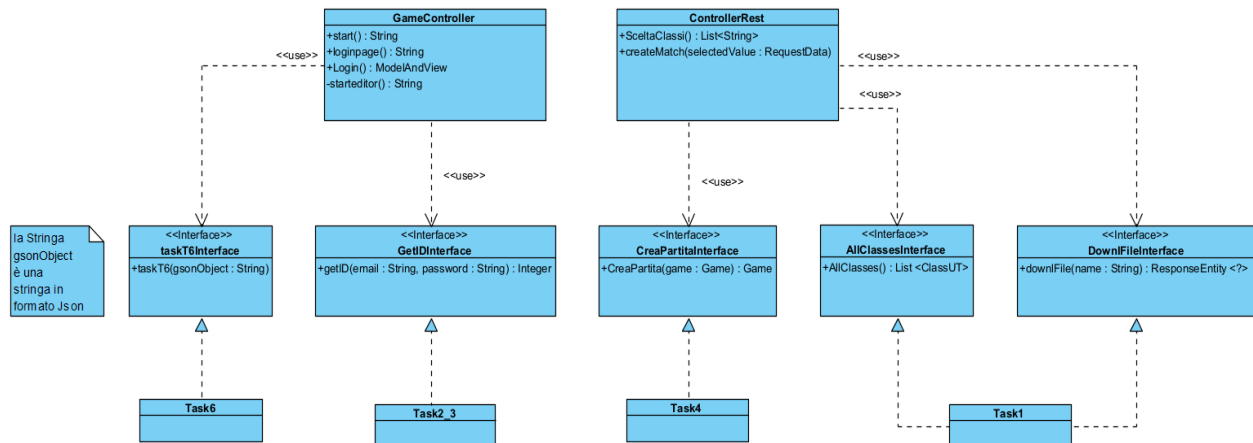
All'interno del nostro sistema è presente il componente Dispatcher che si occupa di ricevere le richieste provenienti dal WebBrowser, visualizzare i Controller disponibili e selezionare tra questi quello più adatto a servire quanto chiesto dall'utente.

I Controller si interfacciano con i diversi componenti esterni rappresentati dai task che ci offrono i diversi servizi. In particolare, il componente ClassServiceT1 espone due interfacce: una corrispondente al servizio che offre un'intera lista di classi e un'altra relativa al download della classe scelta dall'utente, che avviene a valle della configurazione della partita. Il componente LoginServiceT2T3 offre l'interfaccia per il servizio di autenticazione, mentre GameServiceT4 quella per il salvataggio della partita nel database. Infine, il component EditorServiceT6 offre l'interfaccia per l'avvio dell'editor con i dati di configurazione scelti dal giocatore.

3.5. Diagramma delle Interfacce

Nel diagramma delle Interfacce vengono specificate le interfacce necessarie per la comunicazione tra i diversi componenti, come già mostrato nel diagramma dei Componenti.

Tali interfacce sono state concordate con i task responsabili delle diverse funzionalità.



Il GameController usa le interfacce GetID e taskT6:

- GetID è stata concordata con il gruppo G01 che si occupa del task T2 e T3, responsabile dell'operazione di login. Tale funzione prende in ingresso email e password e ritorna un intero: infatti se l'operazione va a buon fine restituisce l'identificativo, altrimenti restituisce -1 in caso di errore.
- taskT6 è stata concordata con il gruppo G08 che si occupa del task T6, responsabile dell'avvio dell'editor di gioco. Tale funzione prende in ingresso la stringa Json contenente tutti i dati della partita necessari a tale task, quali identificativo del giocatore, nome della classe da testare, codice di questa, nome del robot da sfidare e identificativo della partita. Al termine di questa funzione termina il nostro task.

Il ControllerRest usa le interfacce CreaPartita, AllClasses e downlFile:

- CreaPartita è stata concordata con il gruppo G04 che si occupa del task T4, responsabile del salvataggio della partita nel database. Tale funzione prende in ingresso la partita da noi costruita avente IDGiocatore, nome della classe e del robot e codice della classe settati, e restituisce una partita avente gli stessi attributi con l'aggiunta del suo identificativo, legato all'inserimento nel database.
- AllClasses è stata concordata con il gruppo G11 che si occupa della gestione del database di classi, task T1. Questa funzione restituisce una lista di classi.
- downlFile è stata concordata sempre con il gruppo G11, e, dato in ingresso il nome della classe, ne restituisce il codice.

4. Descrizione dell'Implementazione

Il codice realizzato per sviluppare l'applicazione documentata fino a questo punto è stato scritto in linguaggio JAVA con l'utilizzo dell'IDE Eclipse con l'ausilio di Spring Tool Suite 4.

4.1.Package

Le diverse classi Java sono state divise in cinque package:

- `com.testingGameT5.demo` (contiene il main dell'applicazione);
- `com.testingGameT5.demo.controller` (contiene le classi che incapsulano i metodi che fungono da Controller);
- `com.testingGameT5.demo.model` (contiene le classi utilizzate come Model);
- `com.testingGameT5.demo.services` (contiene le classi interfaccia che dovranno essere implementate dai task che forniscono i servizi di cui si necessita);
- `com.testingGameT5.demo.mock` (contiene le classi mock che implementano le interfacce al fine di testare l'applicativo utilizzato non disponendo ancora dei servizi effettivi).

4.1.1. `com.testingGameT5.demo`

All'interno di tale package è presente il file `DemoApplication.java` contenente il main dell'applicazione che predispone il server a ricevere le richieste.

4.1.2. `com.testingGameT5.controller`

In questo package sono presenti due file: `GameController` e `ControllerRest`.

Il primo file contiene i Controller che sono in grado di fornire come valore di ritorno il nome della nuova vista da selezionare e mostrare all'utente con al più un elemento Model con cui arricchire tale vista con informazioni attuali provenienti dal back-end. Il secondo file contiene tutti quei metodi Controller che possono essere richiamati attraverso URL, ma che non ritornano alcuna vista. Tale file offre quindi delle API REST che possono essere utilizzate per usufruire di alcuni servizi del back-end.

Il Dispatcher Servlet interno a Spring dovrà essere in grado di identificare i diversi controller. Per il primo file è necessaria l'annotazione `@Controller`, per il secondo, invece, `@RestController`. In entrambi i casi, per indicare che un singolo metodo deve essere utilizzato in risposta ad una richiesta, viene utilizzata l'annotazione `@RequestMapping` oppure `@GetMapping` (seguiti dal path URL necessario a richiamare il metodo Controller richiesto). Se il metodo deve ricevere dei parametri allora è utilizzata l'annotazione `@PostMapping` e, per i singoli parametri, `@RequestParam`.

Di seguito, viene mostrato un esempio delle precedenti annotazioni:

```

@Controller
public class GameController {

    /* Funzione che restituisce la pagina iniziale dell'applicazione */
    @RequestMapping("/")
    public String start() {

        return "start";
    }

    @PostMapping("/checklogin")
    public ModelAndView Login(@RequestParam("email") String email, @RequestParam("password") String password) {

```

- GameController

- start: restituisce la vista iniziale quando si inserisce il rispettivo path nella barra di ricerca.
- loginpage: restituisce la vista di login a seguito della pressione del pulsante “Avvio Partita” presente nella pagina iniziale.
- Login: riceve in ingresso e-mail e password attivandosi a seguito dell’inserimento delle credenziali e della pressione del pulsante “Login” nella pagina di inserimento delle credenziali. Se le credenziali sono corrette viene restituita la vista di configurazione della partita con l’id del giocatore. Se le credenziali sono errate, viene restituita una pagina di errore che consente di reinserire le credenziali riattivando questo stesso metodo. Per verificare le credenziali si usufruisce del servizio offerto dal task 2-3.
- starteditor: restituisce la vista in cui è possibile richiamare l’editor a seguito della pressione del tasto “Conferma” nella pagina di configurazione della partita dopo aver inserito tutte le scelte necessarie.

- ControllerRest

- SceltaClassi: viene invocata quando viene selezionata la pagina di configurazione della partita e si occupa di popolare il menu a tendina di tale pagina con i nomi delle classi disponibili nel sistema. Tale funzione sfrutta il servizio offerto dal task 1 per recuperare una lista di classi che viene poi convertita in una lista di nomi da ritornare.
- createMatch: viene invocata in seguito alla pressione del tasto “Conferma” nella pagina di configurazione della partita. In ingresso riceve la struttura dati RequestData che contiene l’id del giocatore che vuole iniziare una nuova partita e le scelte effettuate (classe da testare e robot contro cui giocare). Viene istanziato un oggetto “Game” per inserire al suo interno le informazioni raccolte. A tali informazioni va aggiunto il codice della classe scelta che viene recuperato sottoforma di MultipartFile grazie all’ulteriore servizio offerto dal task 1. Tale servizio restituisce una ResponseEntity il cui contenuto viene utilizzato per generare il file. L’oggetto “Game” viene quindi fornito al task 4 per l’inserimento nel Database. Il servizio ritorna il medesimo oggetto, arricchendo le informazioni in esso contenute con l’id generato per tale partita dal Database all’atto dell’inserimento. Come ultima operazione, viene creata una stringa in formato Json contenente le informazioni sulla partita creata necessarie al task responsabile dell’editor per il suo corretto funzionamento. Viene istanziato un oggetto di tipo MatchAdapter che contiene solo le informazioni richieste dal task 6 sulla partita

nel formato concordato (il codice della classe prima contenuto nel MultipartFile viene convertito in una stringa). Tale metodo restituisce la stringa Json creata.

4.1.3. com.testingGameT5.model

Questo package contiene tutte le classi utilizzate come Model all'interno del sistema. Queste sono state realizzate in accordo con i diversi diagrammi sviluppati sia in fase di Analisi che di Progettazione. Pertanto, di seguito verranno riportate solo le aggiunte che sono state apportate in fase di Codifica, aggiunte che peraltro, non introducono logica addizionale, ma sono necessarie solo al corretto interfacciamento con i servizi utilizzati e per rendere il codice conforme con alcune scelte legate all'implementazione effettiva.

```
public class MatchAdapter {  
  
    private String IdPartita;  
    private String IdGiocatore;  
    private String IdClasse;  
    private String CodiceClasse;  
    private String IdRobot;  
    private int NumeroRound;
```

La classe MatchAdapter, come già accennato in precedenza, è stata introdotta per adattare la classe "Game" all'interfaccia concordata con il task 6.

Tale classe presenta, quindi, i seguenti attributi:

- ID della partita creata;
- ID del giocatore che ha creato la partita;
- ID (NOME) della classe da testare;
- Codice della classe da testare;
- ID (NOME) del robot contro cui giocare;
- Numero di round della partita (campo introdotto in vista di futuri sviluppi dato che al momento è previsto un unico round per partita).

```
public class RequestData {  
  
    private String value1;  
    private String value2;  
    private String value3;
```

La classe RequestData contiene i valori necessari all'inizializzazione della partita, scelti dall'utente interagendo con la pagina di configurazione.

Sono presenti i seguenti campi nell'ordine riportato:

- ID del giocatore che ha configurato la partita;
- ID (NOME) della classe scelta;
- ID (NOME) del robot scelto contro cui giocare.

4.2.Resources

Per realizzare ciò che viene visualizzato dal giocatore nel front-end, sono state realizzate risorse html, css e JavaScript e delle immagini utilizzate per abbellire l'interfaccia utente.

I file html sono i TEMPLATE utilizzati per realizzare le diverse viste del sistema. Tali TEMPLATE sono arricchiti, quando necessario, con i valori del model. I file css sono utilizzati per realizzare la grafica delle rispettive pagine html. Infine, i JavaScript, anch'essi legati a determinate pagine html, svolgono specifiche funzioni necessarie per la corretta visualizzazione degli elementi e, quando necessario, si occupano di sfruttare alcune funzionalità del back-end richiamando i Controller REST precedentemente documentati.

All'interno della directory del progetto sono presenti due file JavaScript.

taskT6.js è solamente una sorta di “file mock” che sostituisce il file non ancora disponibile fornito dal task 6. La funzione presente al suo interno dovrebbe occuparsi di ricevere in ingresso la stringa in formato Json contenente le informazioni della partita ed utilizzate da tale task per creare l'ambiente di editing. La funzione mock realizzata ed esportata dal file si limita a stampare sulla console del browser la stringa ricevuta per verificare la correttezza dei valori forniti.

matchsettings.js è legato alla pagina di configurazione della partita e contiene le diverse funzioni che si attivano in risposta ai diversi eventi scatenati dall'utente che interagisce con la pagina.

Il file contiene le seguenti variabili:

- l'ID del giocatore che è già presente all'interno della pagina;
- il menu a tendina dedicato alle classi;
- il menu a tendina dedicato ai robot;
- la classe scelta;
- il robot scelto;
- il bottone di “Conferma”.

Nel momento in cui viene scelta una classe nel menu a tendina delle classi, la scelta attuale viene memorizzata nella variabile “classe scelta”, inizializzata a “null”.

Nel momento in cui viene scelto un robot nel menu a tendina dei robot, la scelta attuale viene memorizzata nella variabile “robot scelto”, inizializzata a “null”.

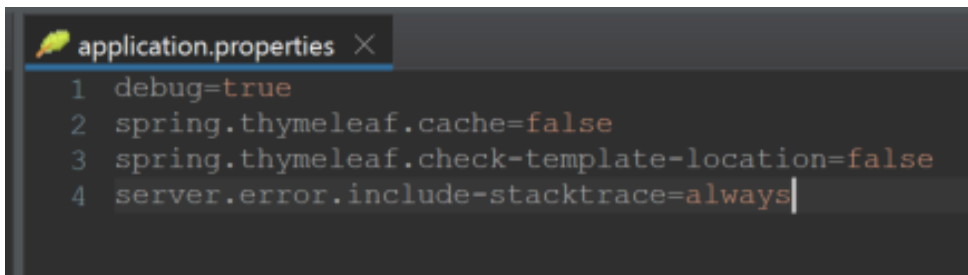
Nel momento in cui la pagina viene caricata, viene automaticamente invocata la funzione che si occupa di popolare il menu a tendina delle classi richiamando la funzione del Controller REST “SceltaClassi”.

Nel momento in cui viene premuto il pulsante “Conferma” si verifica una sola delle due possibili sequenze di operazioni.

- 1) Se anche solo una delle due scelte dovesse ancora essere al valore “null” viene riportato un messaggio sulla console del browser in cui si invita l’utente ad effettuare entrambe le scelte prima di premere il pulsante di “Conferma” e non è possibile procedere alla schermata successiva.
- 2) Se entrambe le scelte sono state effettuate viene richiamata la funzione del Controller REST “createMatch” fornendo in ingresso id del giocatore, classe e robot scelti. Ricevuta in risposta la stringa Json contenente le informazioni da fornire al task 6, questa viene passata attraverso la funzione presente nell’altro file JavaScript precedentemente importata. Dopo aver effettuato queste operazioni, si viene rediretti alla pagina di avvio dell’editor (viene automaticamente invocato il Controller “starteditor”).

4.3.Ulteriori file di interesse

- **pom.xml**: tale file contiene le dipendenze necessarie al corretto funzionamento dell’applicativo e i plug-in utilizzati in fase di build.
- **Application.properties**: tale file è utilizzato per configurare Spring Boot (nello specifico, oltre alle informazioni di default già presenti, sono state aggiunte delle properties necessarie per individuare i template sviluppati).



```
application.properties X
1 debug=true
2 spring.thymeleaf.cache=false
3 spring.thymeleaf.check-template-location=false
4 server.error.include-stacktrace=always
```


5. Test

Sono stati eseguiti test specifici sulle interfacce grafiche dell'applicazione web al fine di verificarne il corretto funzionamento. Nel nostro caso disponiamo di 4 interfacce grafiche. La prima è una semplice home page della web application con un pulsante per avviare la partita. Successivamente, c'è l'interfaccia di login dell'utente, dove una volta effettuato l'accesso, viene presentata l'interfaccia per la scelta della classe da testare e del robot da fronteggiare. Infine, c'è la pagina di avvio dell'editor che richiama il task 6. In particolare, il focus di questo specifico test è verificare che tutte le azioni compiute dall'utente, interagendo con le interfacce grafiche, vengano gestite correttamente. Inoltre, ci si vuole assicurare che le risposte a tali azioni corrispondano alle aspettative dell'utente. Sono stati condotti test in base ai seguenti aspetti:

- Validazione dei bottoni;
- Corretta navigazione tra le pagine;
- Funzionamento dei menu a tendina;
- Verifica della corretta visualizzazione di messaggi di errore o avvisi, quando necessario.

Test Case ID	Descrizione	Pre-condizioni	Input	Output attesi	Post-condizioni attese	Output ottenuti	Post-condizioni ottenute	Esito (PASS-FAIL)
1	Il bottone della home page "Avvio partita" permette di accedere alla pagina di inserimento delle credenziali	-	Click sul bottone	Reindirizzamento verso la pagina di login per l'inserimento delle credenziali	-	La pagina di login viene visualizzata con successo	-	PASS
2	L'utente inserisce e-mail e password corretti, entra nel sistema e viene riportato nella pagina di configurazione della partita	L'utente deve aver avviato una nuova partita	E-mail: "antoniorusso@gmail.com"; Password:"1234"	L'utente deve entrare nel sistema ed essere reindirizzato verso la pagina per la configurazione della partita	-	L'utente accede al sistema e la pagina di configurazione della partita viene correttamente visualizzata	-	PASS

3	L'utente non inserisce le sue credenziali	L'utente deve aver avviato una nuova partita	E-mail:""; Password:"";	La pagina non deve reindirizzare l'utente verso la pagina successiva ma mostrare un opportuno messaggio di errore che chiede di inserire le credenziali ("compila questo campo")	L'utente non deve accedere al sistema	Viene correttamente visualizzato un messaggio di errore che chiede di inserire le proprie credenziali ("compila questo campo")	L'utente correttamente non accede al sistema	PASS
4	L'utente inserisce solo l'email	L'utente deve aver avviato una nuova partita	E-mail:"antoniorusso@gmail.com"; Password:"";	La pagina non deve reindirizzare l'utente verso la pagina successiva ma mostrare un opportuno messaggio di errore che chiede di inserire la password ("compila questo campo")	L'utente non deve accedere al sistema	Viene correttamente visualizzato un messaggio di errore che chiede di inserire la propria password ("compila questo campo")	L'utente correttamente non accede al sistema	PASS
5	L'utente inserisce solo la password	L'utente deve aver avviato una nuova partita	E-mail:""; Password:"12";	La pagina non deve reindirizzare l'utente verso la pagina successiva ma mostrare un opportuno messaggio di errore che chiede di inserire l'email ("compila questo campo")	L'utente non deve accedere al sistema	Viene correttamente visualizzato un messaggio di errore che chiede di inserire la propria email ("compila questo campo")	L'utente correttamente non accede al sistema	PASS

6	L'utente inserisce credenziali errate	L'utente deve aver avviato una nuova partita	E-mail: "ciao"; Password: "1234";	La pagina deve mostrare un messaggio di errore di credenziali non valide	L'utente deve poter reinserire le sue credenziali	La pagina mostra correttamente il messaggio di errore "credenziali non valide"	L'utente è in grado di inserire nuovamente le sue credenziali	PASS
7	L'utente seleziona la classe da testare e il robot da fronteggiare tramite due menu a tendina e salva le sue scelte cliccando sul pulsante di "Conferma"	L'utente deve essere autenticato	Click dei bottoni che aprono i menu a tendina; Classe: "Classe 4"; Robot: "Randoop" Click sul pulsante "Conferma"	In seguito al click sui pulsanti predisposti alla scelta della classe e del robot devono aprirsi i relativi menu a tendina, consentire la scelta della classe e del robot ed infine permettere il click sul pulsante di "Conferma" che reindirizza alla schermata di avvio editor	-	Viene correttamente consentita la scelta della classe da testare ed il robot da fronteggiare. Infine, alla pressione del pulsante "Conferma", l'utente viene reindirizzato correttamente alla pagina di avvio editor	-	PASS
8	In seguito alla pressione del pulsante per la visualizzazione della lista delle classi da testare, viene visualizzata tale lista con una lunghezza variabile in base al numero di	L'utente deve essere autenticato	Click del pulsante che apre il menu a tendina delle classi da scegliere	In base alla lunghezza della lista delle classi, viene mostrato un menu a tendina che dinamicamente si adatta a tale lunghezza	-	Viene visualizzata correttamente la lista con tutte le classi al suo interno	-	PASS

	classi prelevate dal task 1							
9	L'utente non effettua alcuna scelta per configurare la partita e clicca sul pulsante di "Conferma"	L'utente deve essere autenticato	Click del pulsante "Conferma"	L'utente deve restare bloccato nella pagina attuale fin quando non sceglie la classe da testare ed il robot da fronteggiare	-	L'utente erroneamente viene reindirizzato nella pagina successiva nonostante e non abbia effettuato le scelte	-	FAIL
10	L'utente sceglie solo la classe da testare ma non il robot per configurare la partita e clicca sul pulsante di "Conferma"	L'utente deve essere autenticato	Click del bottone che apre il menu a tendina delle classi; Classe: "Classe 4"; Click del pulsante "Conferma"	L'utente deve restare bloccato nella pagina attuale fin quando non sceglie la classe da testare ed il robot da fronteggiare	-	L'utente erroneamente viene reindirizzato nella pagina successiva nonostante e non abbia effettuato la scelta del robot	-	FAIL
11	L'utente sceglie solo il robot da fronteggiare ma non la classe per configurare la partita e clicca sul pulsante di "Conferma"	L'utente deve essere autenticato	Click del bottone che apre il menu a tendina dei robot; Robot: "Randoop" Click del pulsante "Conferma"	L'utente deve restare bloccato nella pagina attuale fin quando non sceglie la classe da testare ed il robot da fronteggiare	-	L'utente erroneamente viene reindirizzato nella pagina successiva nonostante e non abbia effettuato la scelta della classe	-	FAIL

Gli ultimi tre casi di test hanno fornito esito negativo evidenziando un errore nel sistema. Di conseguenza, la soluzione si è ottenuta inserendo il riferimento alla pagina di avvio dell'editor solo quando l'utente ha precedentemente selezionato la classe da testare e il robot con cui interagire. Nello specifico, per risolvere tale problematica si è tolto il riferimento alla pagina di avvio dell'editor da "matchsettings.html" e lo si è inserito in "matchsettings.js" dopo la verifica di tale condizione.

5.1. Testing API tramite Postman

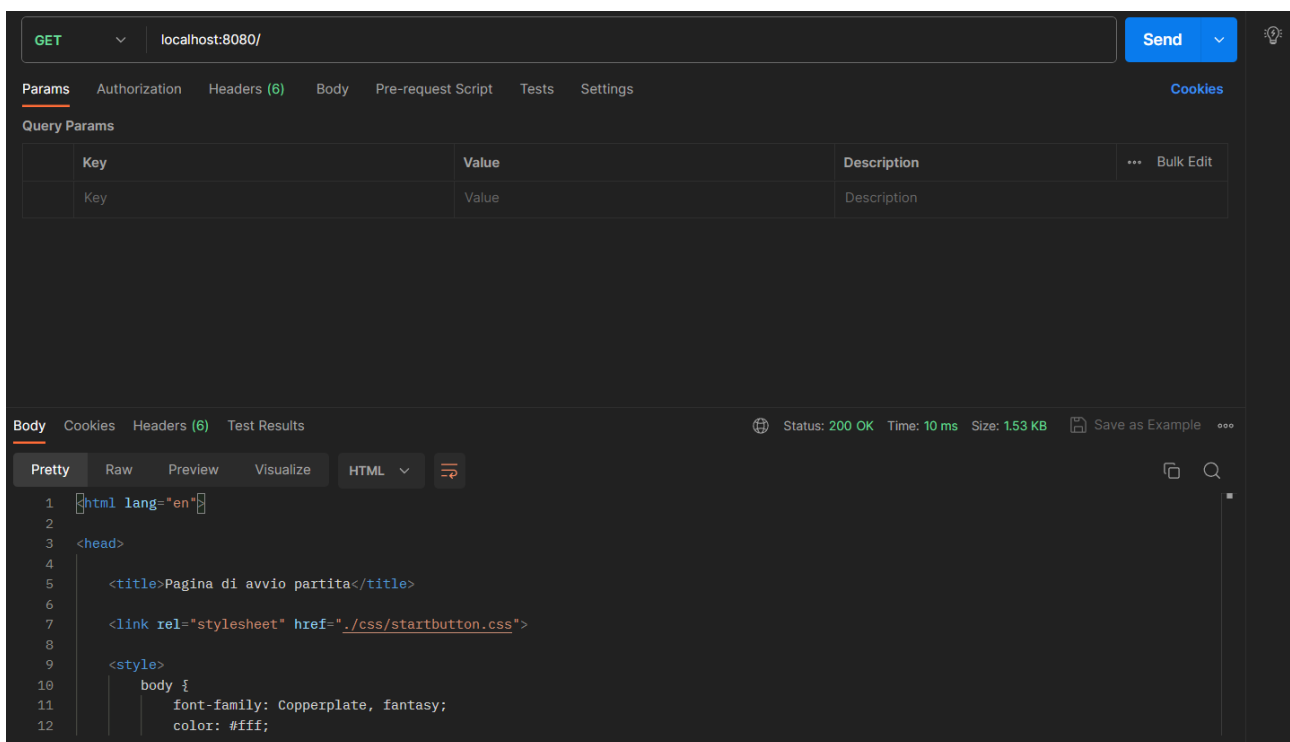
Le API, che sta per Interfacce di Programmazione delle Applicazioni (Application Programming Interface), sono regole e protocolli che consentono a diverse applicazioni software di comunicare e interagire tra di loro. Nella nostra web application, abbiamo implementato diverse API all'interno dei due controller: "GameController" e "ControllerRest". Il primo controller gestisce le API che restituiscono le viste, ovvero le pagine HTML, mentre il secondo controller contiene le API REST.

Per effettuare i test delle API implementate è stato utilizzato lo strumento *Postman*, che consente di farlo in maniera rapida ed efficiente.

Di seguito sono riportate le descrizioni delle API testate mediante *Postman*.

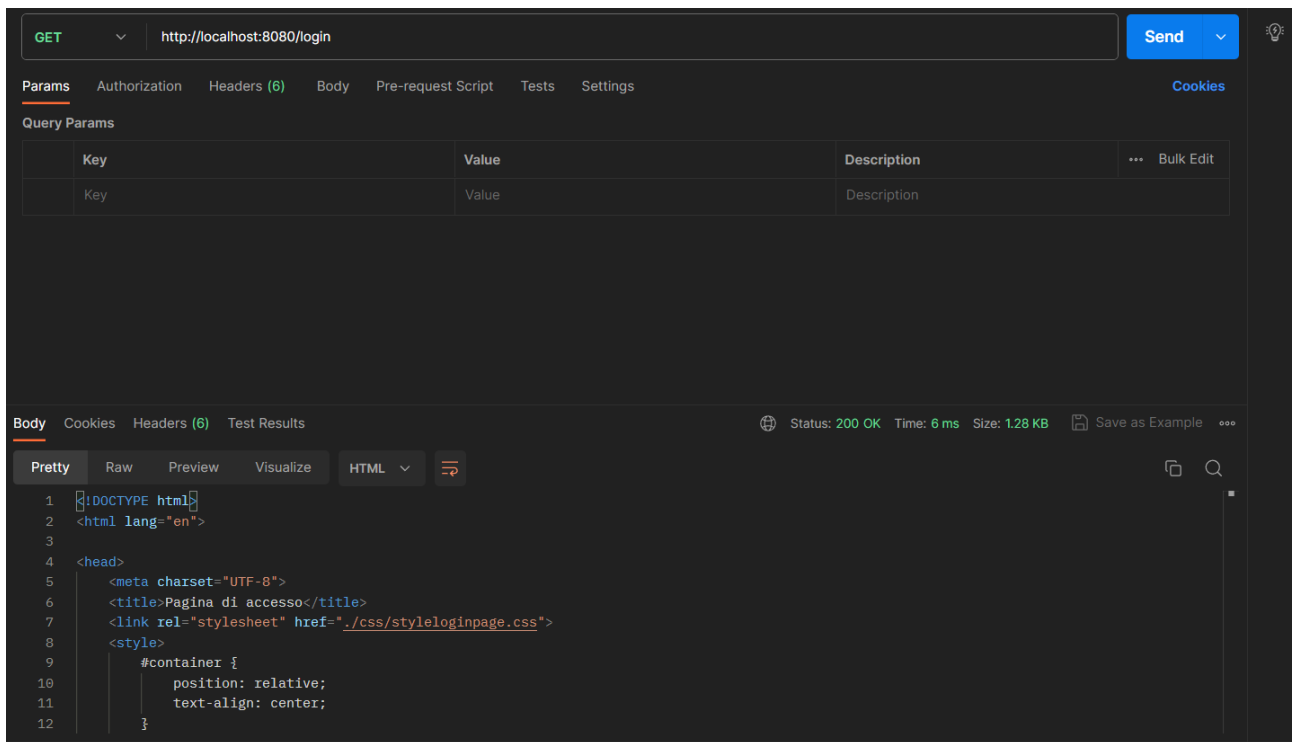
5.1.1. API che ritorna la Pagina di Start

Tale API, presente all'URL "localhost:8080/", è costituita semplicemente dal ritorno della pagina iniziale dell'applicazione. Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta l'HTML della pagina principale.



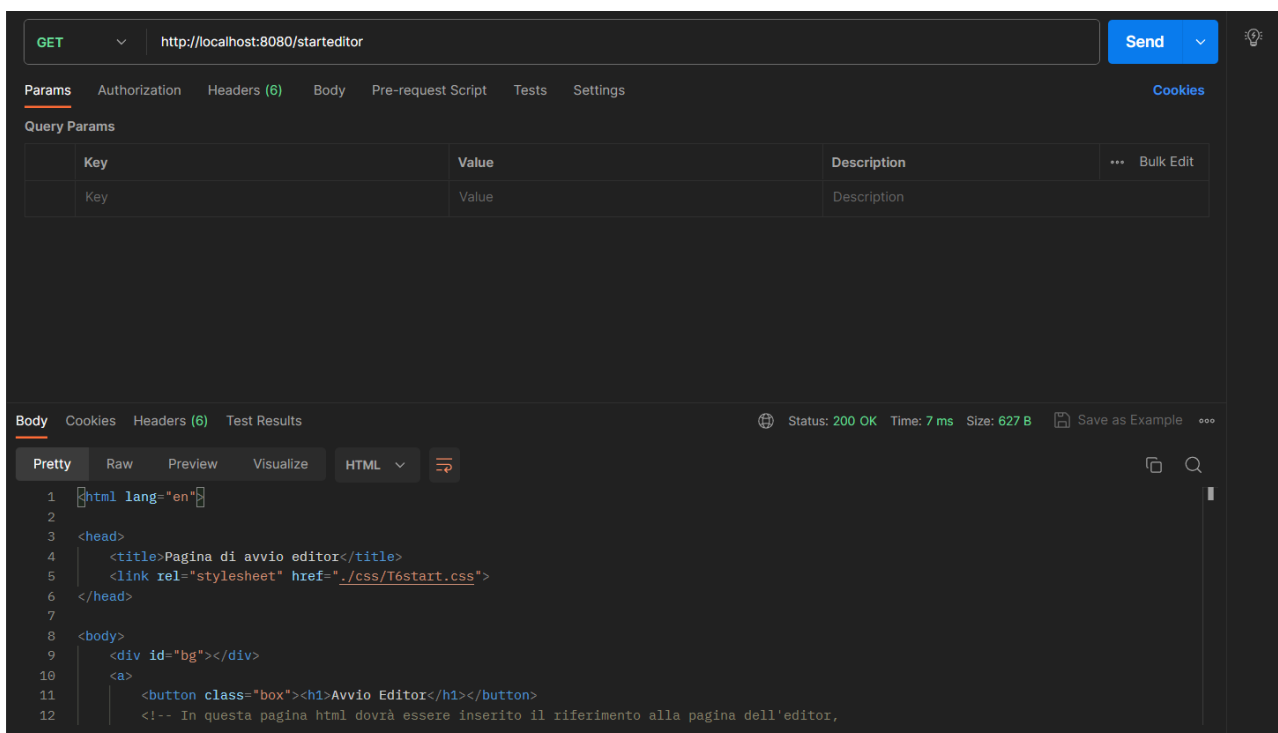
5.1.2. API che ritorna la Pagina di Login

Tale API, presente all'URL "localhost:8080/login", è costituita dal ritorno della pagina di login dell'utente. Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta l'HTML della pagina attesa.



5.1.3. API che ritorna la Pagina di Avvio Editor

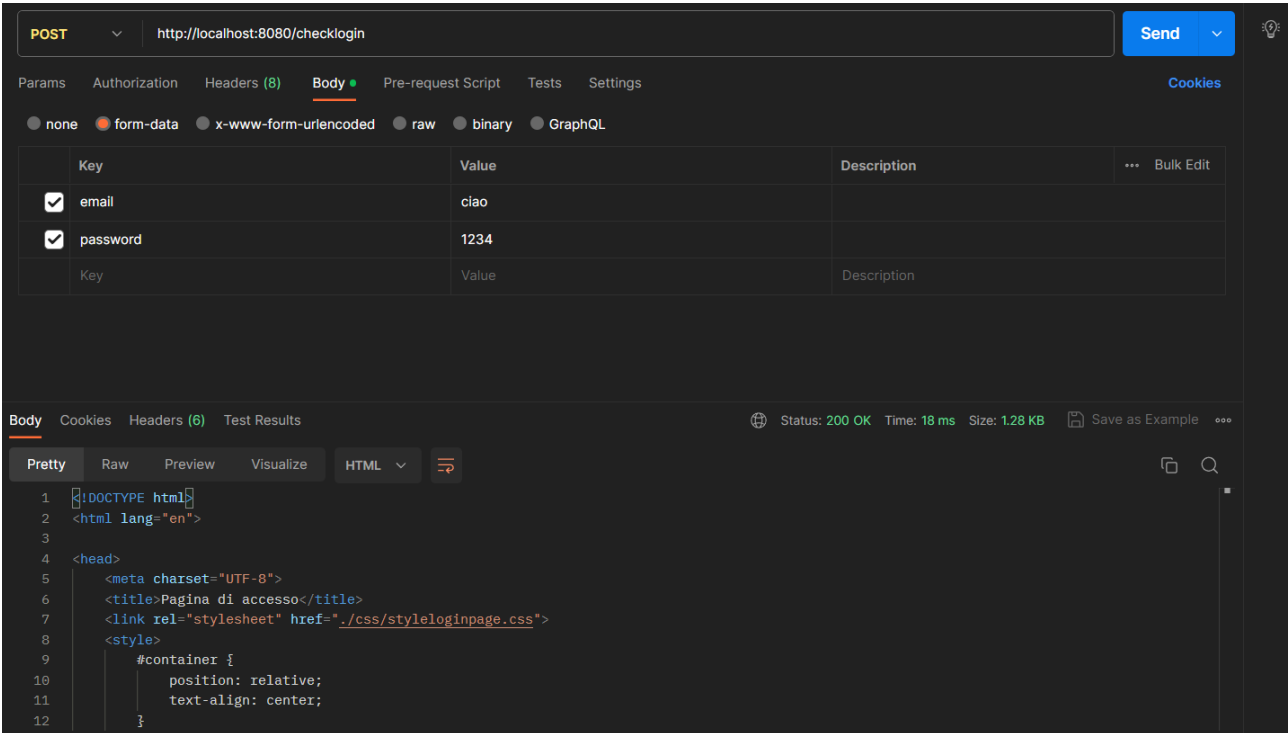
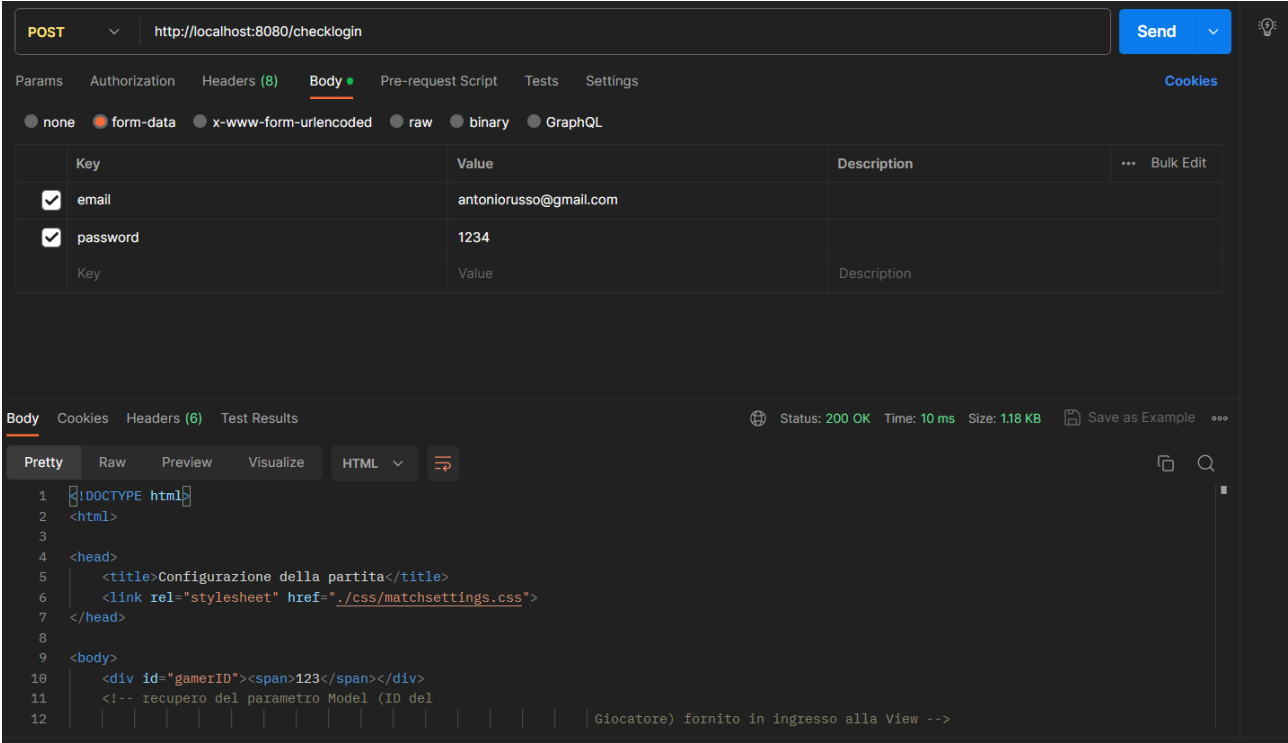
Tale API, presente all'URL "localhost:8080/starteditor", è costituita dal ritorno della pagina di avvio dell'editor di gioco. Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta l'HTML della pagina attesa.



5.1.4. API di Inserimento delle Credenziali

Tale API, presente all'URL "localhost:8080/checklogin", rappresenta la funzionalità di inserimento credenziali da parte dell'utente che richiama la funzione del task 2-3. In particolare, accetta come parametri di ingresso email e password del giocatore.

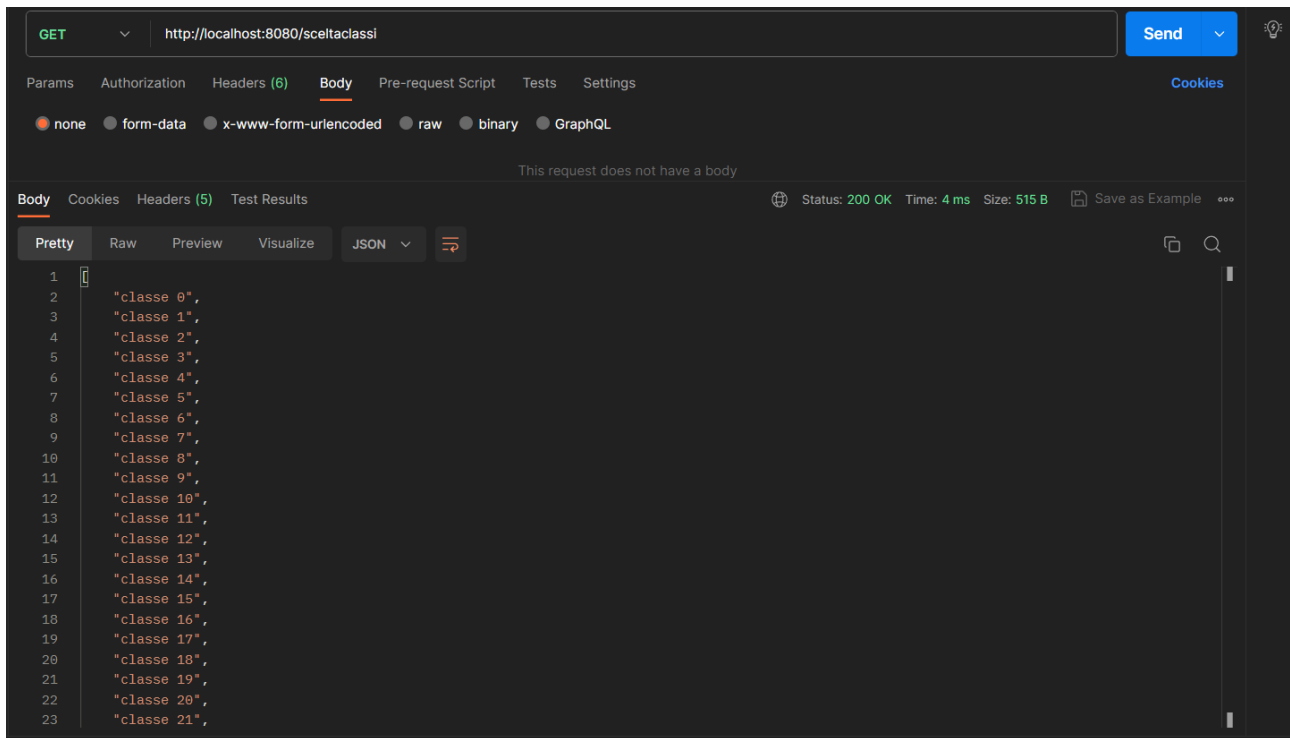
Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta l'HTML della pagina attesa, ovvero consente all'utente di entrare nel sistema. Se invece l'utente inserisce credenziali errate, viene restituita la stessa pagina di login con un messaggio di errore.



5.1.5. API di Scelta delle Classi da Testare

Tale API, presente all'URL "localhost:8080/sceltaclasse", rappresenta la funzionalità che richiama il servizio offerto dal task 1 di prelievo dal database delle classi da testare e la trasforma nella corrispondente lista di nomi delle classi disponibili.

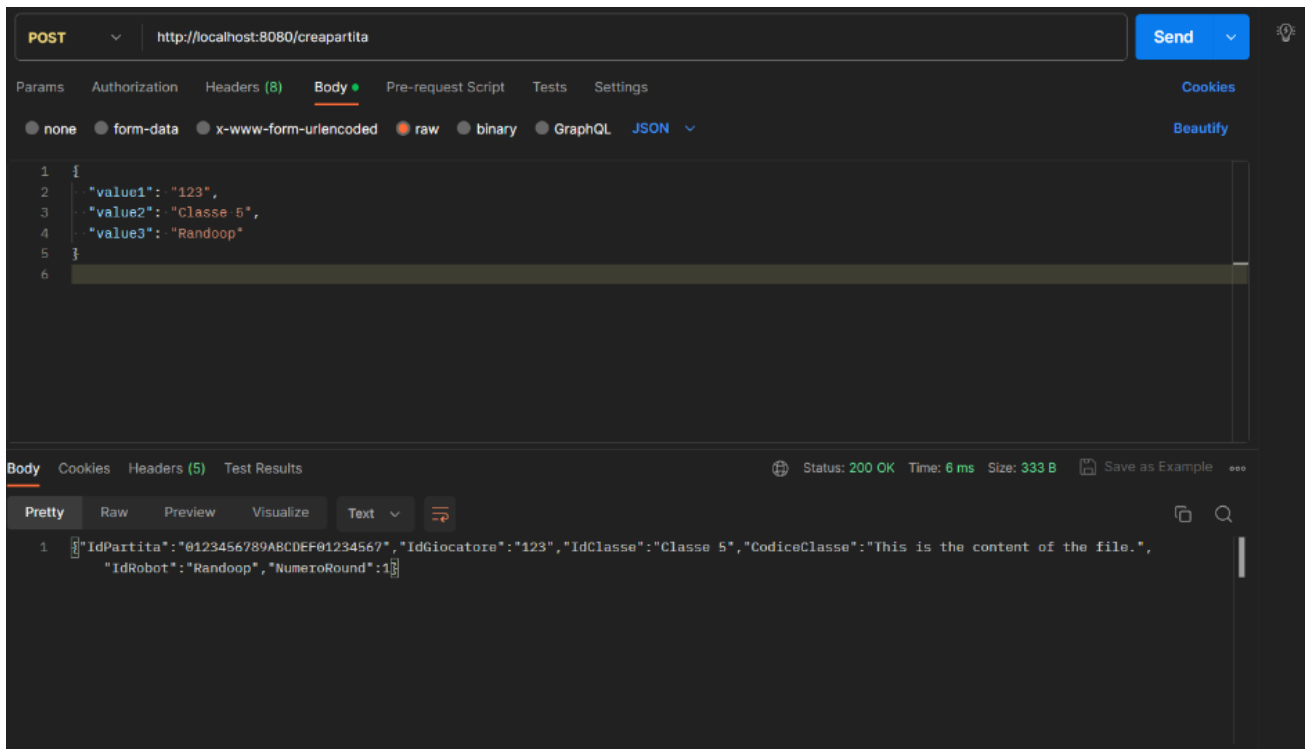
Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta, in formato JSON, la lista di nomi delle classi disponibili.



5.1.6. API di Creazione della Partita

Tale API, presente all'URL "localhost:8080/creapartita", rappresenta la funzionalità che si occupa di creare la partita e restituisce la stringa Json con i dati ad essa associati. Questa funzione riceve una struttura dati che contiene i campi relativi all'ID del giocatore, alla classe scelta e al robot da sfidare.

Di seguito si verifica come effettivamente viene restituito lo status 200 OK, e correttamente *Postman* restituisce come risposta, in formato JSON, la partita creata.



6. Diagramma di Deployment e Guida all'Istallazione

Dopo aver approfondito gli aspetti relativi all'implementazione e al testing del sistema sviluppato, si esamina la procedura di installazione del sistema all'interno di un container gestito tramite Docker, un'applicazione che consente la creazione e la gestione di ambienti in cui installare il software per agevolare lo sviluppo di un sistema complesso da parte di più team. La configurazione delle operazioni che verranno effettuate tramite Docker è stata realizzata tramite il file "Dockerfile". Esso consiste di tre comandi: il primo, "FROM openjdk:17" serve a specificare l'architettura che consente il corretto funzionamento dell'applicazione (che, in questo caso, è la versione 17 del Java Development Kit); il secondo comando, "ADD target/demo-0.0.1-SNAPSHOT.jar app.jar" consente di inserire nel Container il file indicato dal parametro "target/demo-0.0.1-SNAPSHOT.jar" e gli fornisce l'alias "app.jar", utilizzato nel comando successivo; il terzo e ultimo comando, "ENTRYPOINT ["java", "-jar", "/app.jar"]", consente di definire il comando shell tramite cui l'applicazione inserita all'interno del Container viene eseguita.

Di seguito verrà illustrata la procedura di installazione da seguire per poter utilizzare il task sviluppato. Affinché essa possa essere correttamente portata a termine, è necessario scaricare i file presenti nella repository "Testing-Game-SAD-2023/T5-G14" (accessibile al link <https://github.com/Testing-Game-SAD-2023/T5-G14>) ed inserirli in una cartella che, per fini dimostrativi, verrà indicata come "demo_directory" nella procedura; inoltre, è richiesto che sia installato Docker e sia attivo l'Engine sul dispositivo. La procedura di installazione segue questi passaggi:

1. Dopo il download dei file indicati sopra, estrarre il file "target.rar" nella directory corrente mantenendo il nome "target" per la cartella estratta. La cartella "target" contiene già il file .jar ottenuto a valle dell'operazione di build del sistema; esso sarà utilizzato nei passi successivi, in quanto fondamentale per la creazione del container Docker. La directory del progetto dovrebbe apparire come mostrato nella figura seguente (la cartella "demo" corrisponde a "demo_directory"):

Nome	Ultima modifica	Tipo	Dimensione
.mvn	09/07/2023 23:33	Cartella di file	
frontend	09/07/2023 23:33	Cartella di file	
src	09/07/2023 23:33	Cartella di file	
target	09/07/2023 23:33	Cartella di file	
.classpath	09/07/2023 16:31	File CLASSPATH	2 KB
.factorypath	09/07/2023 16:31	File FACTORYPATH	34 KB
.gitignore	09/07/2023 23:48	txtfile	1 KB
.project	09/07/2023 16:31	File PROJECT	1 KB
Dockerfile	09/07/2023 16:31	File	1 KB
HELP.md	09/07/2023 16:31	File MD	3 KB
mvnw	09/07/2023 16:31	File	11 KB
mvnw.cmd	09/07/2023 16:31	Script di comandi Windows	7 KB
pom.xml	09/07/2023 16:31	xmlfile	4 KB
target.rar	09/07/2023 23:36	Archivio WinRAR	56.069 KB

2. Aprire il terminale all'interno della cartella "demo_directory" ed immettere il comando `<<docker build -t "nome_immagine" .>>`, sostituendo "nome_immagine" con il nome che si desidera assegnare all'immagine in Docker;

3. Attraverso il comando `<<docker images>>` è possibile visualizzare l'elenco delle immagini create con i rispettivi identificativi; tra le immagini presenti comparirà anche quella appena creata con nome "nome_immagine" nella prima colonna e un identificativo esadecimale che Docker indica come "IMAGE_ID". Eseguire il comando `<<docker images>>` perché servirà conoscere l'identificativo dell'immagine per il passo successivo;
4. Per creare il container partendo dall'immagine desiderata, eseguire il comando `<<docker run -p"porto_Scelto":8080 "primi_tre_caratteri_IMAGE_ID">>`, modificando "porto_scelto" con il numero di porto su cui si vuole far eseguire l'applicazione e "primi_tre_caratteri_IMAGE_ID" con i primi tre caratteri dell'identificativo citato al punto 3.
5. A questo punto, l'applicazione interna al container sarà in ascolto sul porto scelto nel comando indicato al punto precedente. Per verificare la sua funzionalità sarà sufficiente collegarsi al porto desiderato (`http://localhost:"porto_scelto"`); la prima volta, per interrompere l'esecuzione dell'applicazione nel container sarà sufficiente interrompere il processo avviato dal terminale con la combinazione CTRL+C;
6. Per visualizzare l'elenco dei container creati con i rispettivi identificativi e un'indicazione del loro stato attuale, utilizzare il comando `<<docker ps -a>>`;
7. Per avviare il Container (e quindi l'applicazione) in background, eseguire il comando `<<docker start "id_container">>`, dove "id_container" indica l'identificativo del container. Se l'operazione ha esito positivo, l'id del Container avviato verrà mostrato nel terminale;
8. Per interrompere l'esecuzione del Container (e quindi dell'applicazione), eseguire il comando `<<docker stop "id_container">>`, dove "id_container" è analogo a quanto detto nel punto precedente. Se l'operazione ha esito positivo, l'id del Container arrestato verrà mostrato nel terminale.

Caratteristiche macchina target:

- Windows 11;
- WSL con Linux;
- Docker;
- 16GB di RAM.

6.1. Diagramma di Deployment

Per concludere questa sezione, si riporta di seguito l'install view del task sviluppato, realizzato per avere una vista chiara rispetto al deployment e all'installazione. In esso è possibile vedere i vari componenti delle differenti categorie istanziati tramite il file "demo-0.0.1-SNAPSHOT.jar", che viene prodotto a seguito dell'operazione di build del sistema, effettuata tramite Maven che provvede anche alla gestione automatica delle dipendenze tramite il file "pom.xml".

