



**UNIVERSITÀ⁹ DEGLI STUDI DI
NAPOLI FEDERICO II**

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

Corso di **Software Architecture Design**

***Documentazione A13 - Versione
Migliorata***

Repository: github.com/mar9ia9/A13_versione_migliorata

Anno Accademico 2024/2025

Prof.ssa

Anna Rita Fasolino

Candidato

Monti Maria matr. M63001468

Santoro Emanuele matr. M63001549

Mondillo Angelica matr. M63001482

Indice

1 Progettazione	1
1.1 Descrizione del progetto	1
1.2 Punti chiave	2
1.3 Stato iniziale e Requisiti	3
1.3.1 Struttura	3
1.4 Architettura e analisi	7
1.4.1 Diagramma dei componenti	7
1.4.2 Diagramma dei casi d'uso	9
1.4.3 Composite diagram	11
1.4.4 Sequence diagram	12
1.4.5 Sequence diagram MVC	13
1.4.6 Workflow diagram	15
1.4.7 Diagramma delle classi	16
1.4.8 Deployment diagram	18
2 Processo di sviluppo	20
2.1 Approccio	20
2.1.1 Processo AGILE e SCRUM del Gruppo di Lavoro . . .	20
2.1.2 Fasi del Processo di Lavoro	21
2.2 Strumenti utilizzati	22
2.2.1 Discord	22

2.2.2	Microsoft Teams	23
2.2.3	Visual Studio Code	23
2.2.4	Docker	24
2.2.5	Strumento di ispezione browser	24
2.2.6	Docker	25
2.2.7	Visual Paradigm	26
2.2.8	GitHub	26
2.3	Descrizione requisiti di progetto	27
2.3.1	Requisiti non funzionali	27
2.3.2	Requisiti funzionali	28
3	Implementazione requisiti R2, R3, R4, R5	30
3.1	Implementazione R2 - Pulsante di salvataggio editor	30
3.1.1	Codice	31
3.2	Implementazione R3 - Combinazione colori GUI	34
3.2.1	Codice	37
3.3	Implementazione R4 - Nome classe in caricamento	40
3.3.1	Codice	41
3.3.2	Testing funzionalità	45
3.4	Implementazione R5 - Refresh della pagina	46
3.4.1	Codice	48
4	Implementazione requisito R1 - Pagina Profilo Utente	54
4.1	Implementazione Task 5	56
4.1.1	/main.html	56
4.1.2	/main.js	58
4.1.3	/profile.html, /profile.js & /profile.css	59
4.1.4	/ProfileController.java	68
4.2	Implementazione Task 2-3	70

4.2.1	/controller.java	70
4.3	Implementazione UI Gateway	72
4.3.1	/default.conf	72
5	Requisito R6 - Campagna testing di concorrenza	73
5.1	Configurazioni iniziali	74
5.2	Testing	75
5.2.1	Tabella dei Test case	76
6	Guida all'installazione e all'utilizzo	80
6.1	Installazione	80
6.1.1	Docker e applicazione	80
6.1.2	Ngrok	81
6.2	Utilizzo	82
6.2.1	Admin	82
6.2.2	User	83
6.2.3	Sviluppatore Backend	83
6.2.4	Modifica il codice	84
6.2.5	Tips & Tricks	84

Chapter 1

Progettazione

Il capitolo seguente offre una descrizione approfondita degli obiettivi, delle dinamiche e della struttura del progetto in questione. Prima di avviare l'implementazione dei requisiti, è stato indispensabile svolgere un'analisi dettagliata delle richieste e delle implicazioni che ciascun requisito avrebbe comportato. Questo metodo è stato scelto per ridurre le modifiche unicamente a quelle strettamente necessarie.

1.1 Descrizione del progetto

Il progetto **ENACTEST** (European iNnovative AllianCe for TESTing) si propone di sottolineare l'importanza del testing, un aspetto spesso trascurato a causa della mancanza di competenze adeguate da parte degli sviluppatori rispetto alle aspettative aziendali. Da questa esigenza nasce il gioco educativo "*Man vs Automated Testing Tools Challenges*", che utilizza la gamification per incentivare gli studenti a progettare casi di test, mettendoli in competizione con strumenti automatizzati come Randoop o EvoSuite.

Il gioco si basa su una sfida in cui gli studenti competono contro questi strumenti di testing automatizzato, capaci di generare automaticamente casi

di test JUnit, con l'obiettivo di raggiungere un certo livello di copertura, ad esempio la copertura del codice. Ogni gruppo di studenti è incaricato di sviluppare specifici requisiti funzionali, suddivisi in vari task. Per ciascun requisito è disponibile una documentazione dedicata¹.

1.2 Punti chiave

Considerando la complessità del progetto, la sua natura stratificata, il coinvolgimento di numerosi sviluppatori e l'evoluzione costante, è stato fondamentale fare affidamento sulla piattaforma GitHub. Questo ha permesso di:

- facilitare la collaborazione tra i diversi team di sviluppo, consentendo di visualizzare in tempo reale le modifiche al codice e gestire efficacemente eventuali conflitti;
- lavorare serenamente sulla propria parte assegnata

Il lavoro descritto in questa documentazione prende avvio dal progetto **A13-2024** e si è sviluppato appositamente per integrare i seguenti requisiti:

ID	Descrizione
R1(issue #15)	Si prevede la creazione di una pagina "Profilo utente" che mostrerà nome, cognome, username, punti, partite giocate e tempo totale di gioco dell'utente.
R2 (issue #10)	Il pulsante di salvataggio nell'editor è stato aggiornato per salvare correttamente in locale lo stato attuale della classe di test, includendo tutte le modifiche apportate dal giocatore, anziché il template iniziale.
R3 (issue #9)	La combinazione di colori nell'editor, per entrambi i temi disponibili, è stata modificata per migliorare la visibilità del testo e la leggibilità complessiva, con un contrasto ottimizzato tra sfondo e caratteri
R4	È stata aggiornata la sezione di caricamento della classe nell'interfaccia admin. Ora, il nome della classe viene assegnato automaticamente in base al nome del file caricato, eliminando la necessità per l'admin di inserirlo manualmente
R5 (issue #6 e #11)	Il sistema è stato aggiornato per garantire la coerenza dell'applicazione durante il refresh (F5) della pagina di gioco, evitando la perdita di dati e comportamenti anomali.
R6 (issue #20)	È stato progettato e condotto un set di test di concorrenza per verificare l'uso simultaneo dell'applicazione da parte di più giocatori, cercando di replicare errori legati all'accesso al filesystem sul Volume T8 condiviso.

¹<https://github.com/Testing-Game-SAD-2023/A13>

1.3 Stato iniziale e Requisiti

Si descrive lo stato iniziale del progetto per offrire una chiara comprensione della sua struttura originale, fondamentale per l'evoluzione architetturale. Questa spiegazione permette di contestualizzare le trasformazioni successive, fornendo una visione completa delle decisioni e modifiche apportate durante lo sviluppo.

1.3.1 Struttura

La struttura architetturale del nostro progetto si basa sui microservizi, scelta dettata da diverse esigenze:

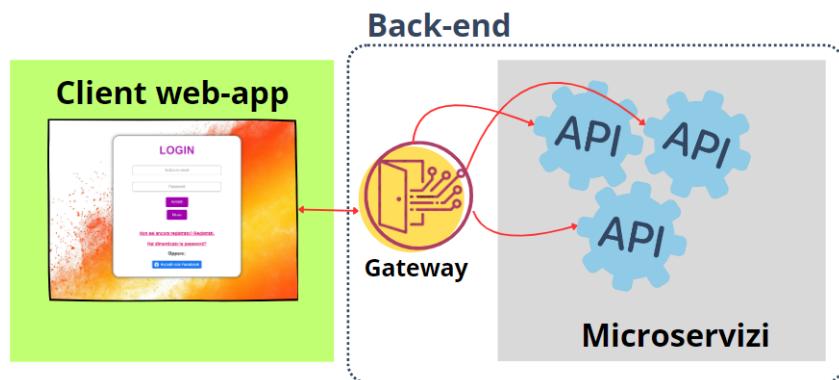
- Assicurare una maggiore agilità ai vari team di sviluppo, permettendo loro di lavorare in completa autonomia. Ogni team può concentrarsi su progettazione, sviluppo, testing e distribuzione di un singolo servizio senza la necessità di interagire o collaborare con altri team che gestiscono servizi indipendenti.
- Migliorare la scalabilità del sistema, poiché è possibile scalare le istanze dei singoli servizi invece di dover scalare l'intera applicazione.
- Facilitare l'integrazione e l'adozione di nuove tecnologie all'interno del sistema.

Andiamo a vedere, più nel dettaglio, i servizi che popolano la nostra applicazione:

ID	Descrizione sintetica	Descrizione dettagliata
T1	Servizio di gestione delle classi da testare e visualizzazione della classifica dei <i>players</i>	<p>Il servizio permette agli <i>amministratori</i> di (a) registrarsi inserendo: nome, cognome, username e password, (b) effettuare la procedura di login.</p> <p>Il servizio permette agli <i>amministratori</i>, interagendo con un opportuno cruscotto, di: (a) caricare, modificare, ordinare, filtrare, ricercare, scaricare ed eliminare classi di programmazione da testare all'interno del catalogo, (b) visualizzare una classifica rudimentale dei <i>players</i>.</p> <p>Il servizio permette ai <i>players</i> correttamente autenticati ed in procinto di giocare, di: visualizzare tutte le classi precedentemente caricate dagli <i>amministratori</i>.</p>
T23	Servizio di autenticazione e registrazione dei <i>players</i>	<p>Il servizio permette ai <i>players</i> di: (a) registrarsi inserendo: nome, cognome, e-mail, password e specificando il corso di studi ('BSc', 'MSc', 'ALTRO'), (b) autenticarsi inserendo le proprie credenziali.</p> <p>Il servizio permette ai <i>players</i> correttamente registrati di: impostare una nuova password nel caso in cui l'avessero dimenticata. Il servizio permette ai <i>players</i> correttamente registrati ed autenticarsi di: (a) accedere all'area riservata di selezione dei parametri di gioco, (b) effettuare il logout</p>
T4	Servizio di repository dei dati di gioco	<p>Il servizio permette al <i>game engine</i> di: (a) creare una partita memorizzando una serie di informazioni, (b) aggiornarla per recuperare la data e l'ora di inizio e fine, (c) eliminarla, (d) generare, contestualmente la partita appena creata, i round, (e) aggiorna li, (f) creare, nell'ambito dei round, diversi turni associati a ciascun partecipante della partita, (g) aggiornarli (h) recuperare, durante ciascun round, i punteggi prodotti dai robot relativi la classe di test in gioco.</p>
T5	Front-end avvio partita	<p>Il servizio permette ai <i>players</i> correttamente autenticati di: (a) accedere all'area riservata di selezione dei parametri di gioco dove poter visualizzare le classi ed i robot disponibili precedentemente caricati dagli amministratori; (b) avviare una partita accedendo all'arena di gioco vero e proprio; (c) accedere alla pagina 'Profilo utente', dove è possibile visualizzare il nome, cognome, username, punti, partite giocate e il tempo totale di gioco dell'utente.</p>
T6	Front-end per giocare una partita	<p>Il servizio mette a disposizione un <i>editor di test case</i> in grado di: (a) fornire una finestra di editing testuale Java all'interno della quale il player correttamente autenticato avrà l'opportunità di scrivere codice ed eseguire le classiche operazioni di editing, richiedere la compilazione, visualizzare i risultati di copertura, (b) confrontare i risultati prodotti dal giocatore con quelli ottenuti dal robot e decretare un vincitore</p>
T7	Servizio di compilazione ed esecuzione	<p>Il servizio permette di compilare ed eseguire i casi di test prodotti dal <i>player</i> in partita.</p>
T8	Servizi robot EvoSuite	<p>Il servizio permette di: (a) eseguire il robot EvoSuite su di una data classe Java e (b) restituire in output le informazioni relative l'esito dei test prodotti dal robot (informazioni di copertura, decisioni, ...).</p>
T9	Servizio robot Randoop	<p>Il servizio permette di: (a) eseguire il robot Randoop su di una data classe Java e (b) restituire in output le informazioni relative l'esito dei test prodotti dal robot (informazioni di copertura, decisioni, ...).</p>

Per integrare tali servizi eterogenei, è stata scelta l'adozione del Gateway

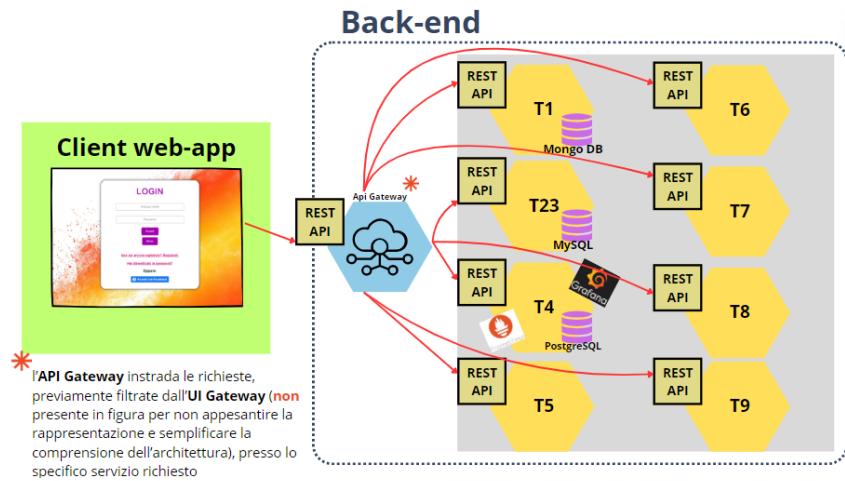
Pattern, il cui obiettivo principale è offrire, sia agli utenti finali che al sistema stesso, un’interfaccia unificata e semplificata per accedere a tutti i servizi disponibili. Questo approccio consente di nascondere, verso l’esterno, i dettagli implementativi e la complessità interna degli elementi sottostanti. In tal modo, viene fornito un unico punto d’accesso che permette di comunicare, ricevere risposte e ottenere risultati.



Come possiamo osservare in figura, il pattern si compone dei seguenti tre elementi fondamentali:

1. **Gateway**: ”elemento di unione” il cui ruolo è quello di: (a) fornire una interfaccia unificata per poter accedere ai servizi messi a disposizione dal sistema che si intende incapsulare, (b) gestire le richieste provenienti dal **Client** ed infine (c) eseguire tutte le operazioni necessarie a restituire i risultati al **Client**.
2. **Client**: entità che interagisce sfruttando i servizi messi a disposizione dal **Gateway** rimanendo sempre inconsapevole della complessità, della struttura e dei dettagli implementativi del sistema sottostante.
3. **Sistema sottostante**: un sistema oppure una collezione di servizi eterogenei che si intende isolare agli ”occhi” del **Client** mediante incapsulazione tramite il **Gateway**.

L’**UI Gateway** gestisce il routing delle richieste HTTP indirizzate al front-end dell’applicazione. L’**API Gateway** si occupa del routing delle richieste verso le API del sistema, gestendo anche l’autenticazione e l’autorizzazione.



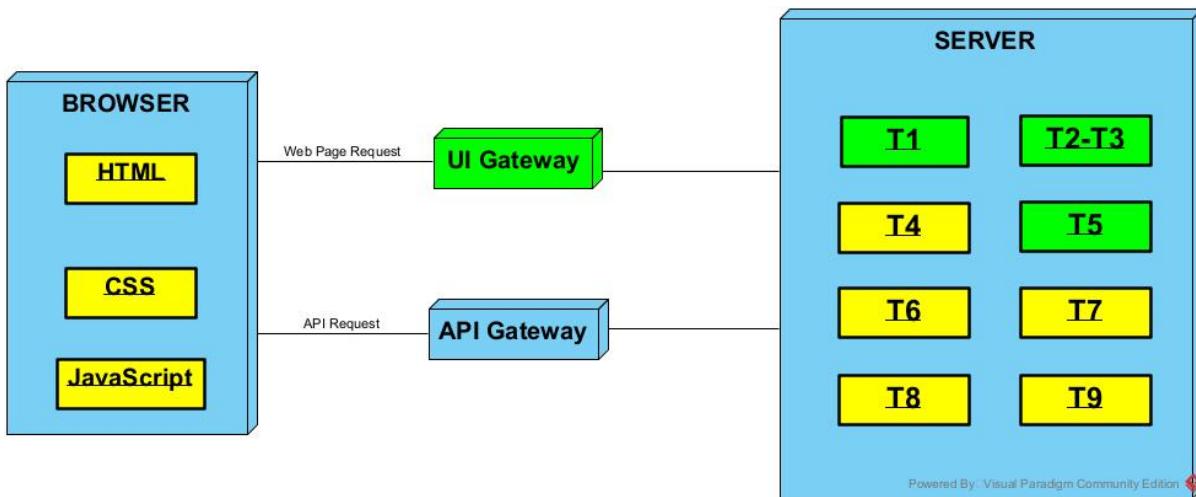
Per evitare di perdgersi nella complessità del back-end, è possibile consultare l'elenco completo degli endpoint API nella documentazione del gruppo A13. Questi endpoint sono essenzialmente "luoghi digitali" dove avviene lo scambio di informazioni attraverso l'API. In termini più tecnici, gli endpoint sono URI (Uniform Resource Identifiers) che possono essere raggiunti dalle applicazioni per inviare o ricevere dati.

Nella documentazione del gruppo A13 sono riportati tutti gli endpoints disponibili, ma in questo contesto viene mostrato solo l'endpoint aggiunto nel task T5, ovvero /profile, che è stato recentemente introdotto a seguito di una modifica.

/profile					
	description	operationID	parameters	RequestBody	response
GET	Restituisce la pagina del profilo utente se il token JWT associato al player è valido	showProfilePage			<p>200: Accesso alla pagina del profilo (JWT valido).</p> <p>302: Reindirizzamento alla pagina di login del player (token JWT non valido).</p>

Figure 1.1: Descrizione dell'API endpoints, aggiunto, relativo al servizio T5.

La documentazione A13 fornisce una rappresentazione dell'architettura a microservizi della web application, evidenziando in **verde** i task modificati per completare i requisiti.



1.4 Architettura e analisi

Si prosegue con una descrizione complessiva dell’architettura, illustrando dove sono stati effettuati gli interventi. A tal fine, verranno presentati un diagramma dei casi d’uso e un diagramma dei componenti, che evidenzieranno le aree coinvolte.

1.4.1 Diagramma dei componenti

Il **Diagramma dei Componenti** è uno strumento prezioso perché:

- Offre una panoramica dell’architettura, mostrando come i vari componenti sono organizzati e interconnessi, facilitando la comprensione del progetto per gli stakeholder.
- Permette di evidenziare e comprendere le dipendenze tra i componenti.
- Aiuta a identificare punti critici e potenziali problemi di progettazione.

Di seguito è riportato il diagramma dei componenti dell’architettura attuale, con i componenti da noi modificati evidenziati in rosso e in giallo quelli usati

per accedere o recuperare dati necessari per la nuova funzionalità del profilo utente.

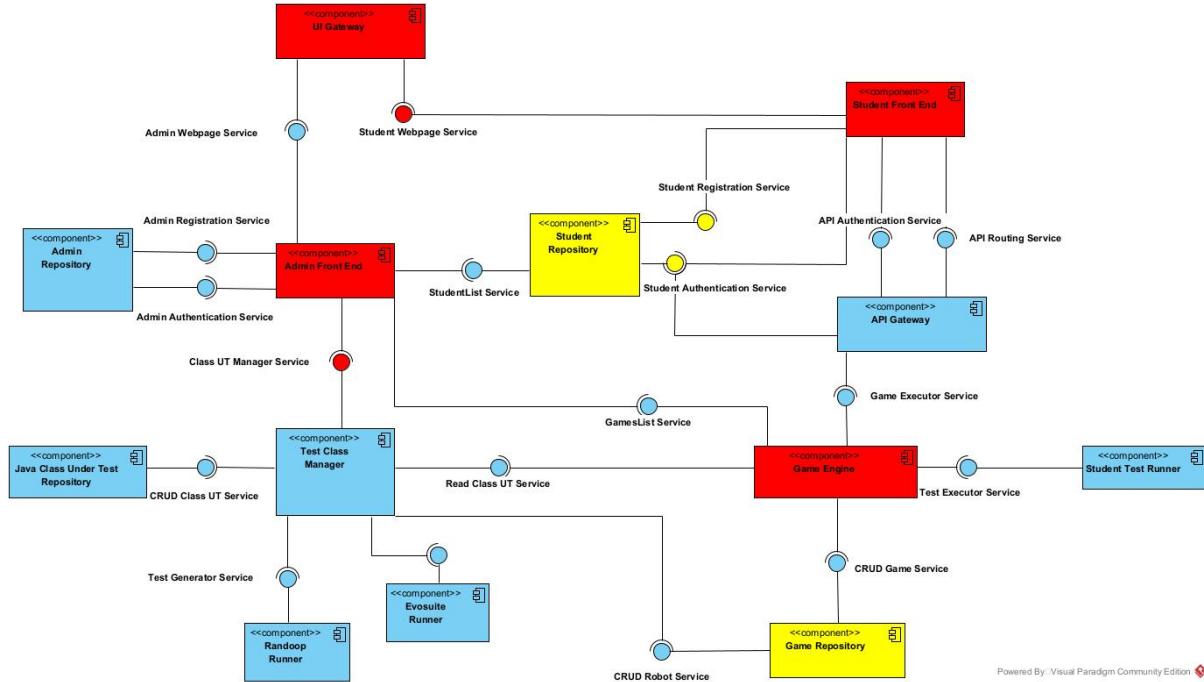


Figure 1.2: Component Diagram

Student Front End

L'aggiunta della pagina *Profilo Utente* influenza il componente *Student Front End* dell'applicazione. Questa modifica permette agli studenti di accedere ad una nuova pagina, gestita dal nuovo controller. L'aggiunta del Profilo Utente comporta le modifiche anche di altri componenti:

- **UI Gateway:** Aggiunta della rotta */profile* nel gateway per instradare le richieste verso il controller del profilo.
- **Student Repository:** Utilizzato per recuperare e modificare i dati utente.
- **Game Repository:** Utilizzato per ottenere dati di gioco, come punteggi e numero delle partite giocate, necessari per arricchire il profilo utente.

Admin Front End

È stato modificato per consentire agli amministratori di caricare una classe senza inserire manualmente il nome, che viene ora acquisito automaticamente dal file caricato.

Game Engine

Il Game Engine è il componente principale che gestisce la logica del gioco, è stato modificato per gestire correttamente il flusso dei turni di gioco.

1.4.2 Diagramma dei casi d'uso

Il **Diagramma dei Casi d'Uso** è uno strumento utile per:

- Rappresentare le funzionalità di un sistema dal punto di vista degli utenti (attori) e delle loro interazioni con il sistema.
- Fornire una visione chiara dei requisiti funzionali, aiutando a identificare i diversi scenari di utilizzo.
- Facilitare la comunicazione tra gli sviluppatori e gli stakeholder, offrendo una descrizione visiva di cosa il sistema dovrebbe fare.

In un diagramma dei casi d'uso, gli attori sono collegati ai casi d'uso che rappresentano le azioni o i servizi che possono eseguire all'interno del sistema. Si evidenziano in rosso i casi d'uso modificati e in verde il caso d'uso aggiunto.

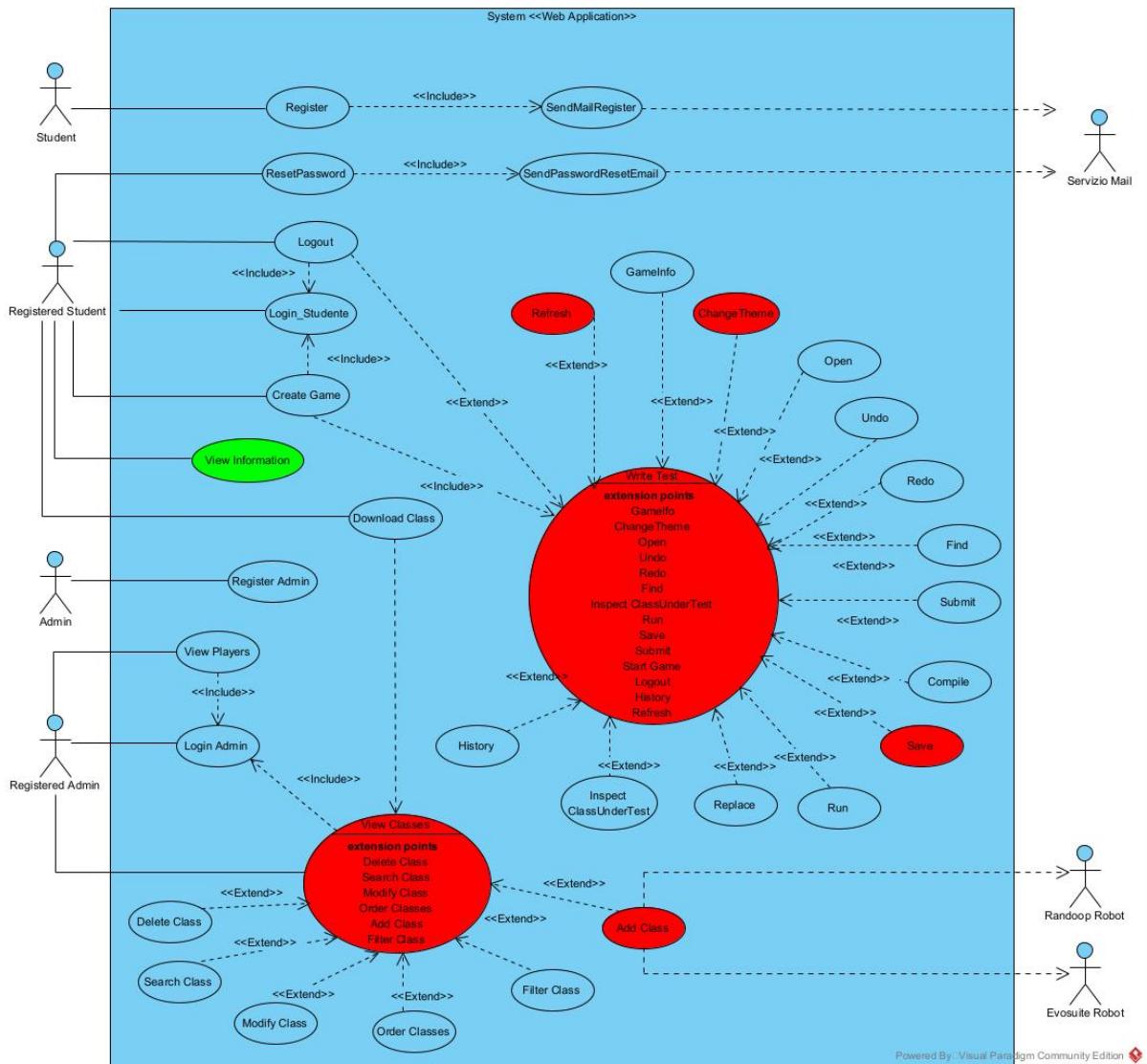


Figure 1.3: Use Case Diagram

View Information

È stato introdotto un nuovo caso d'uso: **View Information**. Questo caso d'uso riguarda l'aggiunta di una nuova pagina utente che consente di visualizzare le informazioni per ogni giocatore.

Add Class

È stato inoltre modificato il caso d'uso **Add Class**, permettendo agli amministratori di caricare una classe senza dover inserire manualmente il nome.

Save

Il caso d'uso **Save** è stato aggiornato per consentire il salvataggio anche delle modifiche apportate.

1.4.3 Composite diagram

Il **Composite Structure Diagram** è uno strumento utile per:

- Rappresentare l'interna struttura di un sistema, mostrando come le parti o i componenti di una classe o oggetto interagiscono tra loro.
- Evidenziare le relazioni e le interazioni tra i diversi elementi interni di un sistema complesso, inclusi oggetti, parti e connettori.
- Facilitare la comprensione dettagliata di come le varie parti di una classe o componente collaborano per fornire una specifica funzionalità.

Il diagramma è necessario per visualizzare rapidamente quali sono i task responsabili dello sviluppo e gestione dei singoli componenti, permettendo una visione chiara della struttura interna del sistema. Si evidenziano in rosso e in verde i componenti e i package modificati e in giallo quelli usati per accedere o recuperare dati necessari alla nuova funzionalità aggiunta.

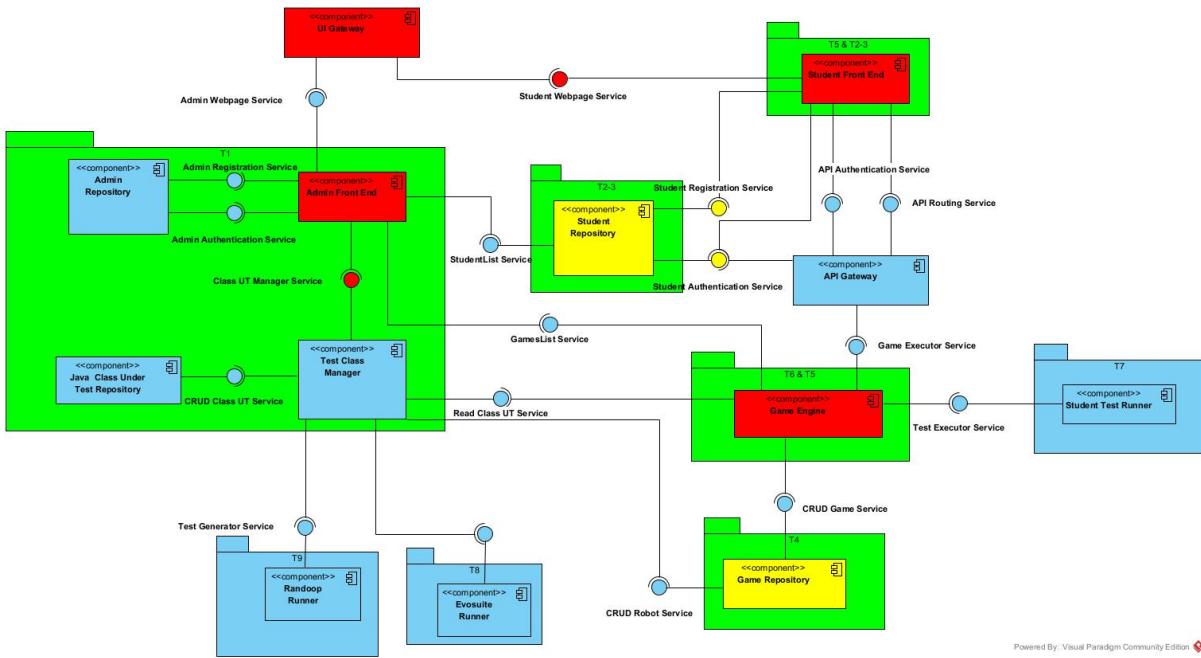


Figure 1.4: Composite Diagram

1.4.4 Sequence diagram

Questo diagramma di sequenza descrive l'interazione tra lo Studente e i diversi componenti del sistema durante l'esecuzione di varie operazioni. A questo diagramma si aggiunge la possibilità opzionale per lo Studente di accedere a una pagina chiamata **Profilo utente** dopo aver effettuato il login. Per supportare questa nuova funzionalità, è stato introdotto un componente aggiuntivo nel sistema, **Student Repository**, che gestisce il recupero delle informazioni personali dello Studente. La pagina "*Profilo utente*" consente allo Studente di visualizzare:

- Informazioni personali provenienti dallo *Student Repository* (task T2-3), come nome, cognome ed email.
- Informazioni sulle partite provenienti dal *Game Repository* (task T4), come punteggi, partite giocate e tempo di gioco.

Dopo aver completato il login, il sistema offre allo Studente la possibilità di decidere se accedere a questa pagina profilo utente oppure proseguire

direttamente con altre operazioni. Se lo Studente sceglie di visualizzare il profilo, il Web Front End recupera i dati personali dal Student Repository e i dati di gioco dal Game Repository, per poi visualizzare la pagina con tutte le informazioni per poi continuare con il flusso principale. Se invece lo Studente sceglie di non accedere alla pagina profilo, il flusso continua come nel diagramma originale.

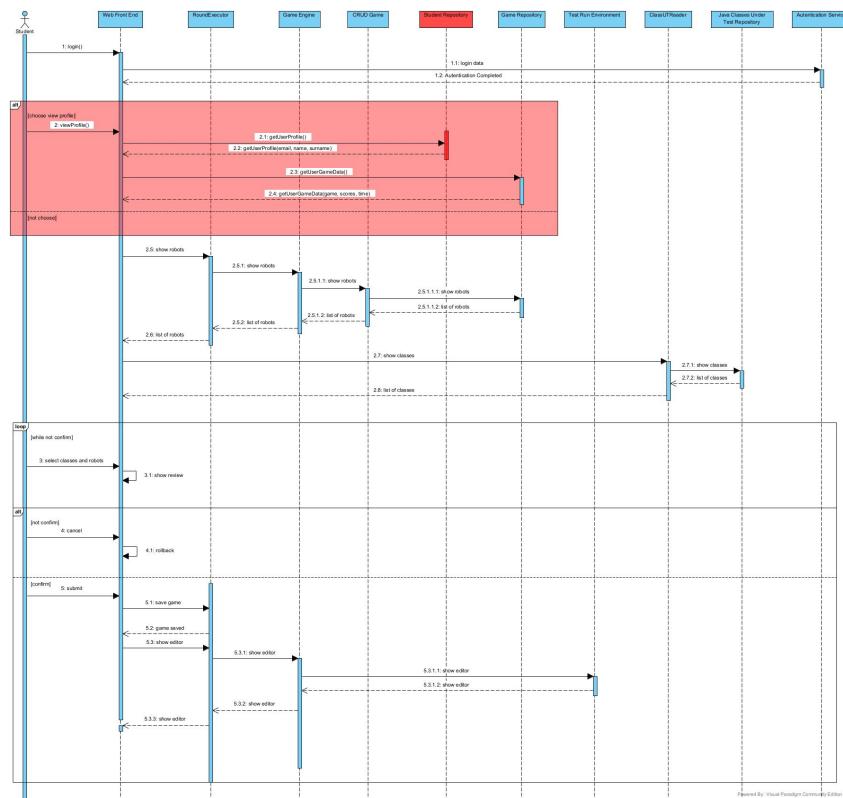


Figure 1.5: Sequence Diagram

1.4.5 Sequence diagram MVC

Nell'architettura **Model-View-Controller**, il diagramma di sequenza mostra come l'utente interagisce con l'interfaccia, come il Controller elabora le richieste e come il Model gestisce i dati, per poi restituire i risultati alla View. Viene utilizzato per comprendere e documentare il comportamento dinamico di un sistema, visualizzando il flusso di azioni e risposte nel tempo. In questo diagramma di sequenza aggiornato, viene mostrato il flusso di inter-

azioni principali che si verificano tra lo Studente, la View, il Controller e il Model. La nuova aggiunta riguarda l'opzione di accedere alla pagina del **profilo utente** subito dopo il login, che è stata rappresentata utilizzando un frammento **alt**, cioè una biforcazione che offre due percorsi alternativi:

- Visualizzare il Profilo Utente: Se l'utente sceglie di visualizzare il proprio profilo, viene eseguita una serie di azioni:
 - La View chiama il metodo *viewProfile()* nel Controller.
 - Il Controller interagisce con il Model per ottenere i dati utente, chiamando *getUserInfo()*.
 - Il Model restituisce le informazioni dell'utente al Controller.
 - La View riceve i dati dal Controller e visualizza le informazioni del profilo all'utente.
- Non Visualizzare il Profilo Utente: Se l'utente decide di non visualizzare il profilo, si salta questo passaggio e si continua direttamente con il flusso principale.

Indipendentemente dalla scelta fatta dall'utente, il flusso principale riprende con tutte le altre operazioni previste dal sistema. L'introduzione di questa opzione nel diagramma rende chiaro che l'accesso al profilo è un'operazione opzionale, non obbligatoria.

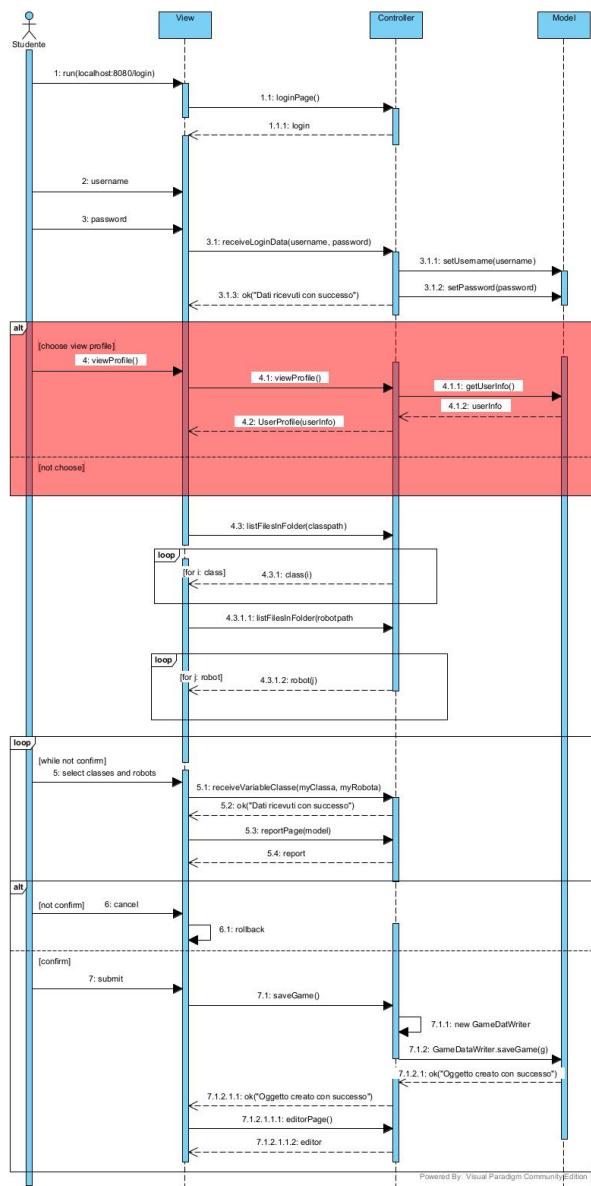


Figure 1.6: Sequence Diagram MVC

1.4.6 Workflow diagram

Il **Workflow Diagram** è uno strumento utile per:

- Rappresentare visualmente il flusso di lavoro o processo, mostrando le sequenze di attività, i partecipanti e le decisioni prese lungo il percorso.
- Aiutare a comprendere come le attività si susseguono e come si scambiano informazioni tra i diversi passaggi, migliorando la chiarezza e l'efficienza del processo.

- Facilitare l'identificazione di colli di bottiglia, inefficienze o punti critici nel flusso di lavoro.

Nel diagramma, le attività o i passaggi sono collegati da frecce che rappresentano il flusso delle operazioni.

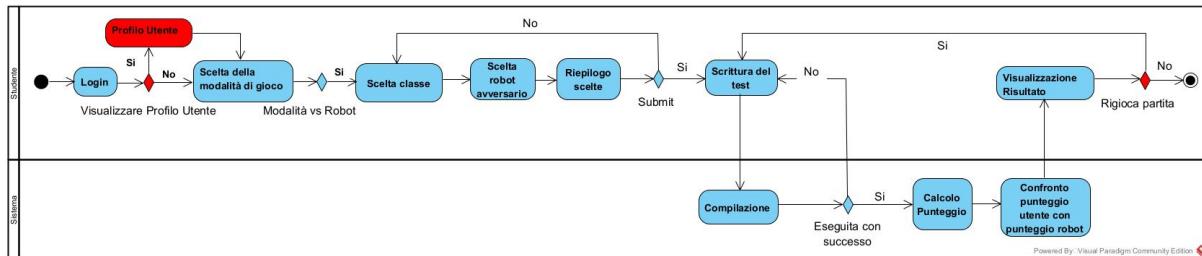


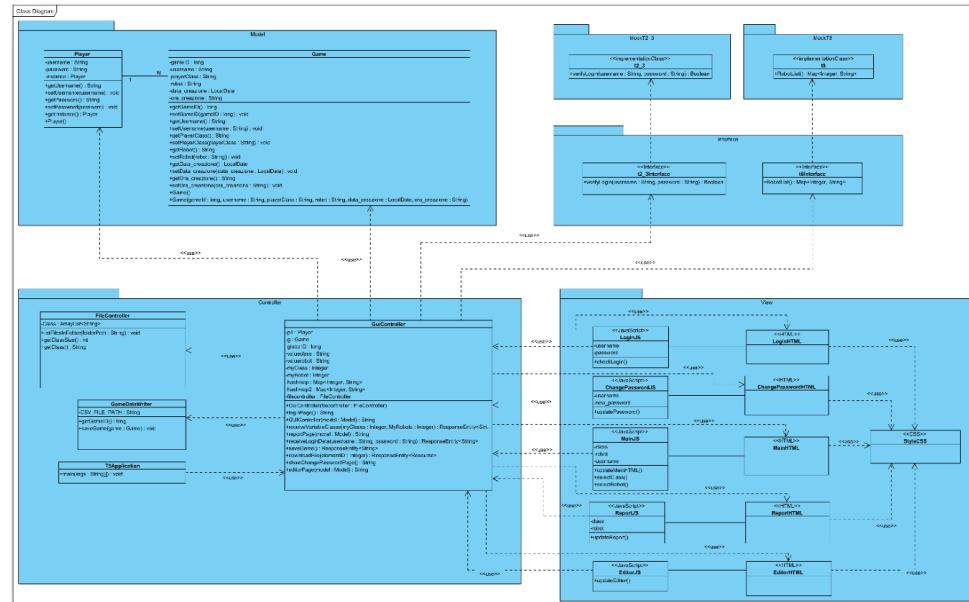
Figure 1.7: Workflow Diagram

Profilo Utente

Nel diagramma è stata aggiunta una nuova attività, relativa alla funzionalità *Profilo Utente*. Ora è possibile scegliere se visualizzare o meno il profilo utente, che contiene informazioni specifiche sul giocatore.

1.4.7 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. Fornisce una descrizione dettagliata delle classi, mettendo in luce le loro funzionalità principali. In figura è riportato il diagramma delle classi della documentazione A13.



Di seguito è riportato il diagramma delle classi, aggiornato con l'aggiunta del **ProfileController** all'interno del package **controller**. Questa aggiunta è stata necessaria per gestire il profilo dell'utente all'interno dell'applicazione. Il ProfileController permette di visualizzare e gestire la pagina del profilo, verificando l'autenticazione dell'utente tramite token JWT. Oltre all'aggiunta del controller, sono stati apportati altri cambiamenti nel package **view**:

- **profile.html**: La pagina HTML definisce la struttura della pagina del profilo.
 - **profile.js**: Il file JavaScript gestisce le interazioni dinamiche e la logica lato client per la pagina del profilo.
 - **profile.css**: Il file CSS contiene lo stile della pagina, assicurando una corretta visualizzazione.

Queste modifiche integrano la gestione del profilo utente all'interno dell'architettura del sistema, migliorando l'interazione tra l'utente e l'applicazione.

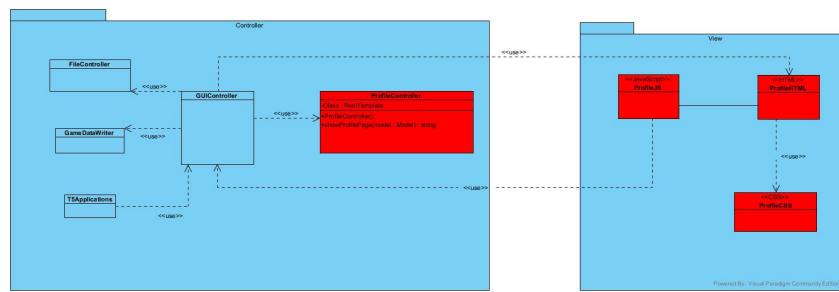


Figure 1.8: Aggiunta gestione Profilo Utente

1.4.8 Deployment diagram

Il Deployment Diagram è uno strumento utile per:

- Rappresentare la distribuzione fisica di un sistema, mostrando come i componenti software vengono eseguiti su nodi hardware (server, dispositivi, ecc.).
- Offrire una visione chiara dell’infrastruttura tecnica, evidenziando dove sono installati i vari elementi del sistema e come comunicano tra di loro.
- Facilitare l’individuazione di potenziali problemi legati alla distribuzione, come prestazioni, affidabilità o dipendenze tra le risorse hardware e software.

Nel diagramma, i nodi hardware sono rappresentati come contenitori che ospitano componenti software, e le connessioni tra i nodi indicano le comunicazioni o interazioni tra le parti del sistema distribuito.

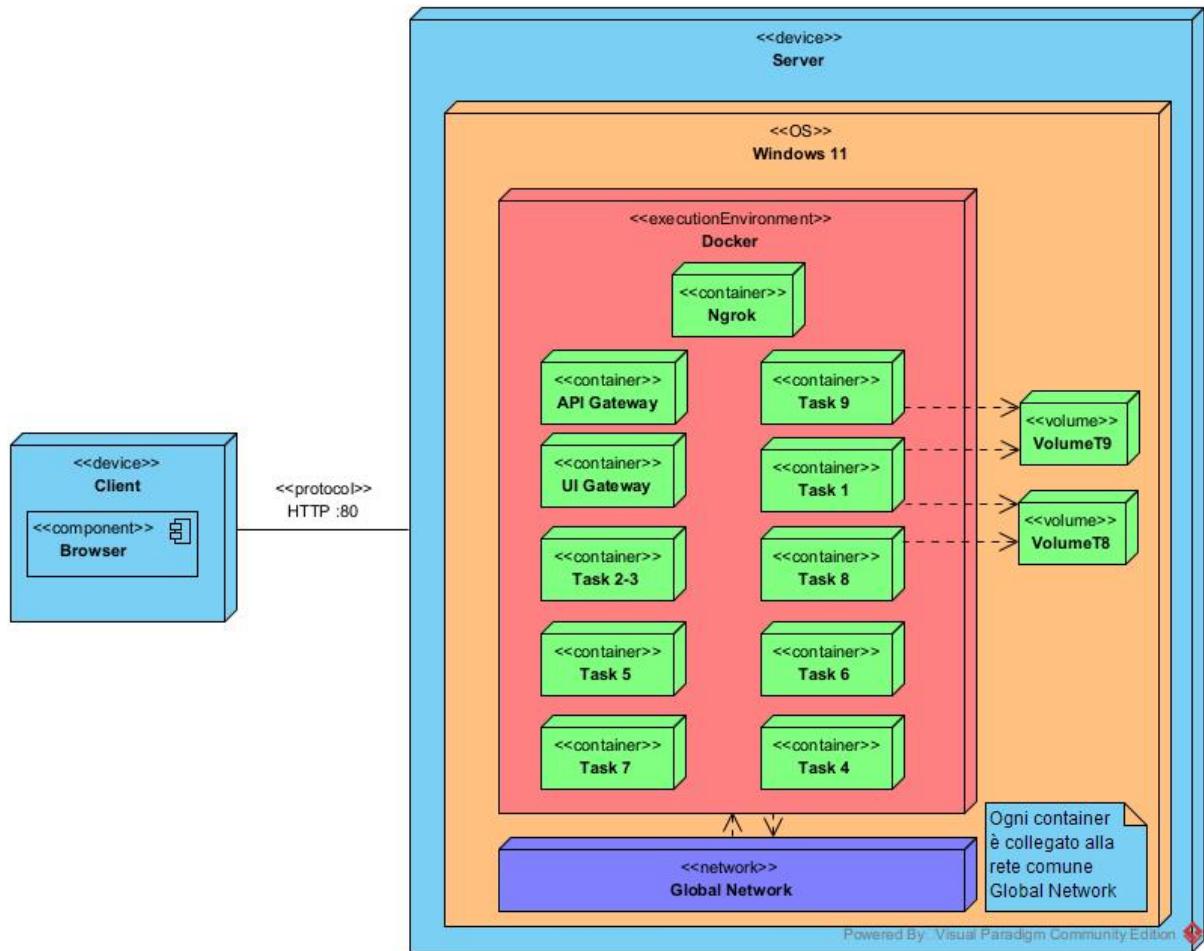


Figure 1.9: Deployment Diagram

Non essendoci state modifiche, è stato riportato il diagramma della prima soluzione proposta nella versione A10-2024, che prevede l'uso del servizio di "tunneling" offerto da Ngrok. La scelta di includere solo la prima delle due soluzioni proposte è dovuta al fatto che soltanto quella con il servizio di "tunneling" è stata effettivamente testata e utilizzata, risultando così la base del nostro lavoro. Per ulteriori dettagli, si rimanda alla documentazione della versione A10-2024 dell'applicazione.

Chapter 2

Processo di sviluppo

In questo capitolo vengono introdotte le issue e le implementazione a noi assegnate con lo scopo di migliorare l'esperienza generale e ampliare le funzionalità della web-app. Questi interventi mirano sia ad aggiungere nuove feature, che a risolvere errori di moduli già esistenti. Viene, inizialmente, descritto l'approccio avuto nei confronti del progetto, elencando dapprima gli strumenti utilizzati, per poi passare alla descrizione e comprensione dei singoli requisiti da implementare.

2.1 Approccio

2.1.1 Processo AGILE e SCRUM del Gruppo di Lavoro

Abbiamo adottato la metodologia **AGILE**¹, utilizzando il framework **SCRUM**² per gestire lo sviluppo del software. Questa metodologia ci ha permesso di essere flessibili, concentrandoci sul codice e sugli obiettivi piuttosto che sulla documentazione e sul design, adattandoci rapidamente ai cambiamenti dei

¹AGILE è una metodologia di sviluppo software basata su iterazioni brevi e flessibili, che permette di adattarsi rapidamente ai cambiamenti dei requisiti, concentrandosi su consegne rapide e funzionali.

²SCRUM è un framework AGILE che organizza il lavoro in cicli chiamati sprint (di solito da 1 a 4 settimane), durante i quali il team lavora su specifici obiettivi e si coordina attraverso incontri giornalieri.

requisiti.

La gestione del lavoro si è basata su una pianificazione dettagliata delle attività da svolgere per ogni sprint. Invece dei classici incontri di 15 minuti, il nostro team ha partecipato a call di gruppo di diverse ore ogni giorno, durante le quali ci siamo sincronizzati, discusso eventuali problemi e pianificato le attività successive. Queste call sono state fondamentali per la collaborazione e la risoluzione rapida delle problematiche. Durante le chiamate, il team lavorava insieme in tempo reale, sia su issue differenti che sullo stesso issue, a seconda delle necessità del momento.

2.1.2 Fasi del Processo di Lavoro

Le attività sono state organizzate e suddivise in quattro fasi:

- Fase 1 - *Analisi della documentazione A13*: Il primo passo è stato l'analisi della documentazione prodotta dal gruppo A13. Ciò ci ha permesso di comprendere meglio i requisiti e identificare le eventuali criticità legate al loro sviluppo.
- Fase 2 - *Implementazione dei Requisiti in Parallello*: Abbiamo deciso di implementare due requisiti in parallelo, suddividendo il lavoro tra i membri del team. In questa fase ci siamo concentrati sui requisiti di più facile implementazione, che richiedessero una minore forza lavoro, lasciando i requisiti più importanti, come la campagna di testing e la creazione del profilo utente per la fase successiva.
- Fase 3 - *Sviluppo del Profilo Utente e Campagna di Test*: In questa fase, il team ha lavorato insieme per sviluppare il profilo utente e la campagna di testing di concorrenza. Grazie alle lunghe videochiamate quotidiane, è stato possibile individuare i task dove intervenire, sud-

dividendo il carico di lavoro in parti uguali per sviluppare il nuovo codice nella maniera più efficiente.

- Fase 4 - *Test Locale e Caricamento su GitHub*: Una volta completate le modifiche, sono state testate in ambiente locale per assicurarsi che funzionassero correttamente. Dopo la verifica, il codice è stato caricato sul repository **GitHub**, rendendolo disponibile per ulteriori revisioni e aggiornamenti.

2.2 Strumenti utilizzati

Di seguito i tool utilizzati durante il processo di lavoro.

2.2.1 Discord

Il team ha utilizzato Discord per comunicare da remoto, sfruttandolo sia per effettuare videochiamate dettagliate in cui si discutevano i passaggi da seguire, sia per scambiarsi rapidamente messaggi, file e aggiornamenti.

Per ulteriori informazioni: <https://www.discord.com>



Figure 2.1: Logo di Discord

2.2.2 Microsoft Teams

Il team ha utilizzato Microsoft Teams per comunicare con la professoressa referente del progetto e discutere i passaggi da seguire, aggiornandosi di volta in volta sullo stato di avanzamento.

Per ulteriori informazioni: <https://www.microsoft.com/it-it/microsoft-teams/group-chat-software>



Figure 2.2: Logo di Microsoft Teams

2.2.3 Visual Studio Code

Visual Studio Code è stato utilizzato per la scrittura e compilazione del codice, essendo uno strumento già familiare e molto versatile. Grazie alla possibilità di aggiungere numerose estensioni, risulta leggero e rapido nelle prestazioni. Tra i suoi principali vantaggi, si trovano:

- il supporto per molteplici linguaggi di programmazione, offrendo un'ampia gamma di opzioni (in questo caso, soprattutto JavaScript, HTML e Python).
- l'integrazione nativa con GitHub, che consente di eseguire operazioni di push direttamente dall'editor in maniera semplice e veloce.

Per ulteriori informazioni: <https://code.visualstudio.com>



Figure 2.3: Logo di VSCode

2.2.4 Docker

Docker è stato impiegato come strumento per la containerizzazione, essendo già conosciuto per la sua efficienza nel gestire ambienti isolati. Grazie alla sua leggerezza e flessibilità, è ampiamente adottato nello sviluppo e distribuzione di applicazioni. Tra i suoi principali vantaggi si evidenziano:

- la possibilità di creare ambienti standardizzati, garantendo che il software funzioni in modo coerente indipendentemente dalla piattaforma;
- il supporto per molteplici linguaggi e stack tecnologici, rendendolo adattabile a diverse esigenze di sviluppo.

Inoltre, tramite estensione è perfettamente integrato in Visual Studio.

Per ulteriori informazioni: <https://www.docker.com>



Figure 2.4: Logo di Docker

2.2.5 Strumento di ispezione browser

Lo strumento di ispezione del codice del browser è stato utilizzato come piattaforma per sviluppare e testare codice HTML, CSS e JavaScript in tempo reale. Questo strumento integrato nei principali browser consente di modificare, visualizzare e testare direttamente le modifiche sul sito web. Tra i suoi vantaggi principali:

- la possibilità di vedere immediatamente i risultati delle modifiche al codice, offrendo un feedback rapido per il debugging e l'ottimizzazione;
- l'accesso diretto alla struttura del DOM e agli stili applicati, facilitando l'individuazione e la correzione di errori;

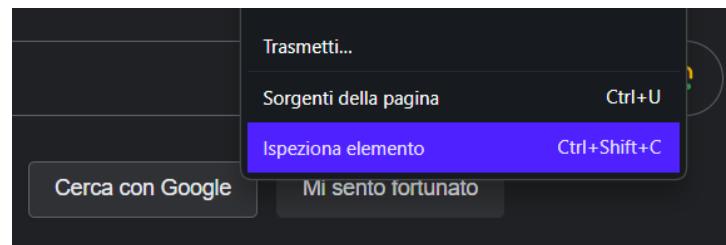


Figure 2.5: Ispeziona pagina web browser

2.2.6 Docker

Ngrok è stato utilizzato come strumento per esporre localmente un server in modo sicuro su Internet, facilitando lo sviluppo e il testing di applicazioni web. Questo servizio crea tunnel sicuri, permettendo di accedere a server locali tramite un URL pubblico, senza necessità di configurare il port forwarding. I suoi principali vantaggi includono:

- la creazione di tunnel HTTPS sicuri per rendere accessibili applicazioni in fase di sviluppo da remoto;
- l'analisi in tempo reale del traffico di rete, utile per il debugging;
- la semplicità di configurazione e l'integrazione con vari stack tecnologici.

Per ulteriori informazioni: <https://ngrok.com>



Figure 2.6: Logo di Ngrok

2.2.7 Visual Paradigm

Visual Paradigm è uno strumento software completo per la progettazione e la modellazione di sistemi software, utilizzato principalmente per creare diagrammi UML (Unified Modeling Language) e per supportare metodologie di sviluppo agile, come Scrum. È ideale per l'analisi e la progettazione di sistemi complessi, offrendo una gamma di funzionalità avanzate, tra cui:

- la creazione di diagrammi di classe, sequenza, attività e molti altri, utili per visualizzare l'architettura del software;
- strumenti di gestione dei requisiti e tracciamento, per assicurare che le specifiche del progetto siano seguite;
- la reverse engineering, semplificando il passaggio dalla progettazione allo sviluppo.

Per ulteriori informazioni: <https://www.visual-paradigm.com>



Figure 2.7: Logo di Visual Paradigm

2.2.8 GitHub

GitHub è stata utilizzata come piattaforma per la gestione e il controllo di versione del progetto, poiché offre un ambiente centralizzato e collaborativo per lo sviluppo del software. Si tratta di un servizio basato su Git, un sistema di controllo di versione distribuito, che consente di monitorare le modifiche al codice sorgente e facilitare la collaborazione tra più sviluppatori.

Abbiamo scelto GitHub per diversi motivi::

- permette di tenere traccia delle modifiche al codice, garantendo la possibilità di tornare a versioni precedenti se necessario, migliorando così l'affidabilità del processo di sviluppo;
- gestione di contributi da più sviluppatori attraverso strumenti come le pull request, che facilitano la revisione del codice e l'integrazione controllata dei cambiamenti;
- facile integrazione con sistemi di automatizzazione della fase di testing e distribuzione del software.

Per ulteriori informazioni: <https://github.com>



Figure 2.8: Logo di GitHub

2.3 Descrizione requisiti di progetto

Il documento ha l'obiettivo di fornire una visione chiara dei requisiti funzionali e non funzionali del sistema, assicurando una comprensione comune tra sviluppatori, progettisti e stakeholder sulle funzionalità e aspettative del progetto.

2.3.1 Requisiti non funzionali

Ecco i requisiti non funzionali che il sistema deve soddisfare per garantire un funzionamento efficace e di qualità:

- **Disponibilità:** il sistema deve essere sempre accessibile e pronto all'uso.

- **Affidabilità:** il sistema deve funzionare in modo stabile e coerente nel tempo, riducendo al minimo guasti e interruzioni, e soddisfacendo i requisiti.
- **Usabilità:** il sistema deve essere facile da usare, comprensibile e intuitivo.
- **Manutenibilità:** il sistema deve poter essere modificato, corretto e aggiornato facilmente, senza impattare negativamente le funzionalità esistenti.
- **Interoperabilità:** il sistema deve essere in grado di scambiare dati e comunicare efficacemente con altri sistemi.

2.3.2 Requisiti funzionali

I requisiti funzionali definiscono le funzionalità e i comportamenti specifici che un sistema deve offrire per soddisfare le esigenze dell'utente.

R1 - Profilo Utente

Si prevede la creazione di una pagina "Profilo utente" che mostrerà nome, cognome, username, punti, partite giocate e tempo totale di gioco dell'utente.

R2 - Pulsante di salvataggio editor

Il pulsante di salvataggio nell'editor deve consentire il salvataggio locale dello stato attuale della classe di test, includendo tutte le modifiche apportate dal giocatore, invece del template iniziale.

R3 - Combinazione colori GUI

La combinazione di colori nell'editor, per entrambi i temi disponibili, deve essere ottimizzata per migliorare la visibilità del testo e la leggibilità comp-

lessiva, garantendo un contrasto adeguato tra sfondo e caratteri.

R4 - Nome classe in caricamento

La sezione di caricamento della classe nell’interfaccia admin deve assegnare automaticamente il nome della classe in base al nome del file caricato, eliminando la necessità di un inserimento manuale da parte dell’admin.

R5 - Refresh della pagina

Il sistema deve garantire la coerenza dell’applicazione durante il refresh (F5) della pagina di gioco, prevenendo la perdita di dati e comportamenti anomali.

R6 - Campagna di test di concorrenza

Il sistema deve essere sottoposto a un set di test di concorrenza per verificare l’uso simultaneo da parte di più giocatori, replicando eventuali errori legati all’accesso al filesystem sul Volume T8 condiviso.

Chapter 3

Implementazione requisiti

R2, R3, R4, R5

In questo capitolo, ci concentreremo sull'implementazione tecnica e funzionale dei requisiti R2, R3, R4 e R5, che rappresentano la prima parte del lavoro su cui ci siamo focalizzati.

3.1 Implementazione R2 - Pulsante di salvataggio editor

L'obiettivo di questo requisito è quello di riparare il pulsante di salvataggio della classe di test nell'editor, garantendo che il file salvato localmente contenga lo stato aggiornato della classe, comprensivo di tutte le modifiche apportate dal giocatore. In precedenza, il sistema salvava in locale solo il template iniziale della classe di test, ignorando le modifiche successive. Grazie a questa correzione, il pulsante di salvataggio ora funziona correttamente, assicurando che il giocatore possa salvare localmente il proprio lavoro esattamente nello stato in cui si trova, evitando la perdita di progressi. A seguito dello studio della documentazione A13 e dei file sorgente, siamo giunti alla

conclusione di dover intervenire sul seguente file all'interno del task T5:

- **commonEditor.js** - Questo file configura e gestisce l'editor di codice tramite la libreria CodeMirror¹. È fondamentale per la gestione dell'editor e della sua interfaccia, offrendo funzionalità avanzate come la formattazione del codice, l'evidenziazione della sintassi e caratteristiche tipiche degli editor moderni, come la chiusura automatica delle parentesi e l'indentazione intelligente.

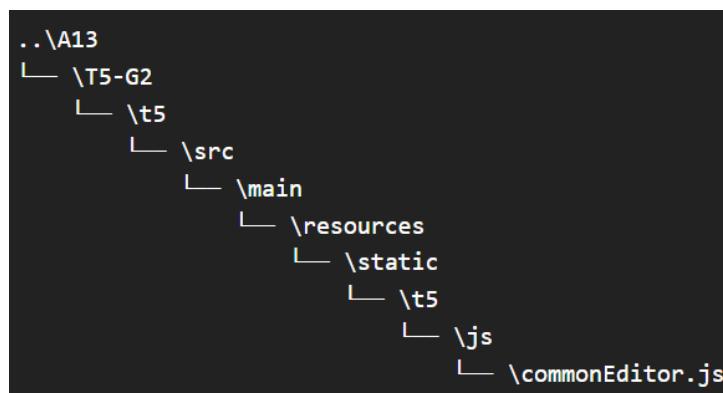


Figure 3.1: Directory di /commonEditor.js

3.1.1 Codice

/commonEditor.js

Il problema principale risiedeva nel metodo di salvataggio utilizzato precedentemente per raccogliere il contenuto del codice da salvare. Il sistema tentava di ottenere il codice tramite l'elemento DOM collegato a un `<textarea>`, che però non veniva aggiornato con le modifiche apportate tramite CodeMirror. CodeMirror, una volta inizializzato su un `<textarea>`, non modifica più il valore di questo elemento HTML. Invece, gestisce internamente il contenuto attraverso la sua API.

Nella precedente implementazione, la funzione di salvataggio era la seguente:

¹CodeMirror è una libreria JavaScript potente e versatile progettata per creare editor di codice all'interno di applicazioni web

```
function saveCode() {
    var code = document.getElementById("editor").value;
    var fileName = prompt("Inserisci il nome del file:", "MyFile.java");
    if (fileName != null && fileName != "") {
        var blob = new Blob([code], { type: "text/plain;charset=utf-8" });
        saveAs(blob, fileName);
    }
}
function saveAs(blob, fileName) {
    var link = document.createElement("a");
    link.download = fileName;
    link.href = URL.createObjectURL(blob);
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
}
```

Figure 3.2: Funzione di salvataggio prima

In questo approccio, il sistema tentava di ottenere il valore dell'editor accedendo al contenuto del <textarea> con l'ID "editor" tramite:

```
var code = document.getElementById("editor").value;
```

Figure 3.3: Approccio della funzione di salvataggio prima

Tuttavia, poiché CodeMirror gestisce il contenuto internamente, il valore del <textarea> non veniva aggiornato, risultando nel salvataggio del template iniziale della classe di test. In altre parole, le modifiche dell'utente erano solo visivamente presenti nell'editor, ma non erano incluse nel file salvato. Per risolvere il problema, la funzione di salvataggio è stata aggiornata per interagire direttamente con l'API di CodeMirror. Invece di ottenere il contenuto dall'elemento DOM <textarea>, la nuova implementazione utilizza il metodo `getValue()` di CodeMirror per ottenere il contenuto corrente effettivamente modificato e visibile nell'editor.

Il nuovo codice della funzione è il seguente:

```

function saveCodeLocally() {
    // Ottieni il contenuto del codice dall'editor
    var codeContent = editor.getValue();

    // Nome di base del file
    var defaultFileName = "myFile.java";

    // Chiedi all'utente il nome del file, con un valore predefinito
    var fileName = prompt("Inserisci il nome del file (con estensione):", defaultFileName);

    // Se l'utente non ha fornito un nome, esci dalla funzione
    if (!fileName) {
        alert("Nome del file non valido!");
        return;
    }

    // Crea un blob di testo con il contenuto del codice
    var blob = new Blob([codeContent], { type: 'text/plain' });

    // Crea un link temporaneo per scaricare il file
    var link = document.createElement('a');
    link.href = window.URL.createObjectURL(blob);
    link.download = fileName; // Usa il nome fornito dall'utente o quello di default
    link.click();

    // Opzionale: cancella l'oggetto dopo il download
    window.URL.revokeObjectURL(link.href);
}

```

Figure 3.4: Funzione di salvataggio dopo

In sintesi, le differenze principali tra la vecchia e la nuova implementazione possono essere descritte come segue:

Accesso al contenuto dell'editor:

- **Prima:** Il contenuto veniva recuperato dall'elemento <textarea>originale, che però non veniva sincronizzato con le modifiche apportate tramite CodeMirror. Questo portava al salvataggio del solo template iniziale.
- **Dopo:** Il contenuto è ora recuperato direttamente dall'editor tramite il metodo `editor.getValue()`, che garantisce l'accesso al codice aggiornato, inclusivo delle modifiche effettuate dall'utente.

Sincronizzazione del contenuto:

- **Prima:** Il codice visualizzato nell'editor non era effettivamente collegato al valore del <textarea>, quindi il salvataggio risultava errato.

- **Dopo:** Utilizzando `editor.getValue()`, il contenuto riflette sempre le modifiche attuali e visibili nell'editor, risolvendo il problema.

Ulteriori Modifiche Oltre al miglioramento nell'accesso al contenuto dell'editor, sono state introdotte altre ottimizzazioni. **Prima**, il prompt per il nome del file verificava semplicemente che il nome non fosse vuoto, senza fornire un feedback visivo in caso di errore. **Ora**, il prompt include un controllo esplicito che avvisa l'utente se il nome è invalido, garantendo una maggiore robustezza nella gestione degli errori. Inoltre, **prima**, il processo di download funzionava correttamente, ma non liberava l'URL temporaneo generato. **Ora**, grazie all'uso di `window.URL.revokeObjectURL()`, le risorse vengono liberate immediatamente dopo il download, migliorando l'efficienza nella gestione della memoria.

3.2 Implementazione R3 - Combinazione colori GUI

L'obiettivo di questo requisito è quello di ottimizzare la combinazione di colori nell'editor per entrambi i temi disponibili, al fine di migliorare la visibilità del testo e la leggibilità complessiva. Per soddisfare questa richiesta, è stato necessario garantire un contrasto adeguato tra lo sfondo e i caratteri.

Per soddisfare le richieste del requisito R2, è stato essenziale esaminare e valutare le combinazioni cromatiche attuali e apportare modifiche mirate per ottimizzare la leggibilità. A tal fine, abbiamo analizzato entrambi i temi disponibili nell'editor, concentrandoci in particolare su come migliorare il contrasto tra i vari elementi del testo e lo sfondo.

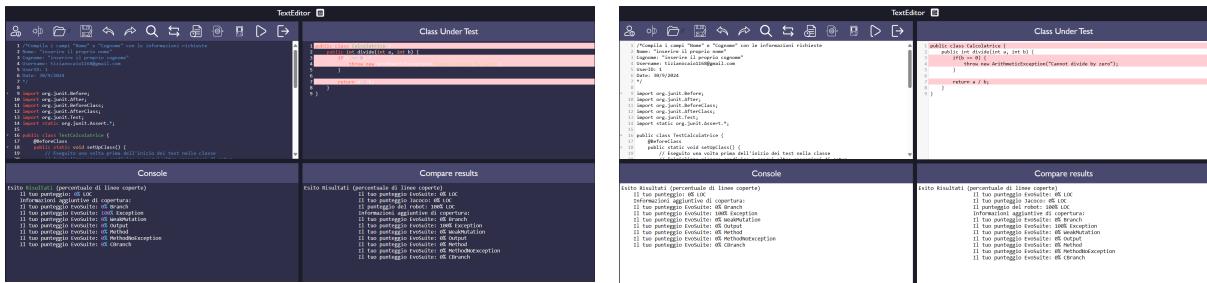


Figure 3.5: Temi prima delle modifiche

Sono stati quindi apportati i seguenti cambiamenti:

- Revisione delle palette di colori per entrambi i temi, assicurandoci che i rapporti di contrasto rispettassero le linee guida per l'accessibilità.
- Modifica dei colori del testo, link e altri elementi testuali per garantire una maggiore visibilità.
- Aggiornamento del tema chiaro, introducendo colori diversi per diverse parti del codice, migliorando la distinzione tra gli elementi del linguaggio e aumentando la leggibilità.

Per soddisfare le richieste del requisito R2 è stato necessario analizzare le pagine HTML relative all'editor e, tramite lo strumento di *Ispezione del codice* del browser, siamo riusciti a risalire ai file su cui intervenire per modificare i contrasti, tutti compresi nel task **T5**. E' stato necessario modificare le seguenti pagine.



Figure 3.6: Directory dei file

/codemirror/theme/lucario.css

Implementa un tema personalizzato per l'editor, basato su CodeMirror. Il tema definisce una combinazione di colori scuri per migliorare la leggibilità del testo e del codice. Vengono utilizzati colori specifici per diversi elementi del codice, come:

- Commenti: blu chiaro (#5c98cd)
- Stringhe: giallo (#ffe600)
- Numeri e variabili: blu brillante (#00d5ff)
- Parole chiave: rosso (#b80000)
- Operatori: azzurro (#66D9EF)

Il tema usa uno sfondo scuro (#141429) per ridurre l'affaticamento visivo durante sessioni di utilizzo prolungate.

/css/editor.css

Il file contiene lo stile per l'editor, parte dell'interfaccia web. Ecco cosa fa:

- Struttura della pagina: Definisce lo stile per container verticali e orizzontali, impostando lo sfondo e le dimensioni delle sezioni principali.
- Barra dell'editor: Include una barra con pulsanti per varie azioni (es. "salva", "compila", "esegui") con icone personalizzate.
- CodeMirror: Configura elementi specifici dell'editor di codice come i marker di piegatura e lo sfondo per la selezione di testo.
- Console: Gestisce lo stile della console di output con font monospazio e sfondo scuro.

In generale, il file definisce vari elementi visivi per migliorare l'interfaccia utente e la navigazione nell'editor.

/codemirror/lib/codemirror.css

Il file è utilizzato per configurare l'aspetto di CodeMirror. Definisce varie proprietà visive per l'editor, come altezza, font, colori per i diversi elementi del codice (commenti, stringhe, variabili, numeri, ecc.), e lo stile del cursore e del selezionatore di righe. Include anche stili per gestire gli spazi bianchi, il layout delle linee di codice e l'aspetto delle scrollbar. Inoltre, imposta lo **schema di colori** personalizzato per il **tema chiaro** e fornisce alcune impostazioni per la visualizzazione di testi selezionati e cursori in modalità di scrittura sovrascrittura.

3.2.1 Codice

Come prima cosa siamo intervenuti sul codice del file *editor.css* per andare a modificare la tonalità di colore della funzione di copertura dell'editor che evidenzia alcune righe del codice quando attivata.

```
296     .uncovered-line {
297         background-color: #7f918a;
298     }
```

Figure 3.7: Tonalità copertura in /editor.css

Inizialmente, la tonalità proposta rendeva poco leggibile alcune parti di codice. Abbiamo deciso, quindi, di adottare il colore `#7f918a`, una tonalità di grigio-verde, che ben si sposta con lo sfondo e con le modifiche al codice che vedremo a breve.

<pre>1 public class Calcolatrice { 2 public int divide(int a, int b) { 3 if(b == 0) { 4 throw new ArithmeticException("Cannot divide by zero"); 5 } 6 return a / b; 7 } 8 } 9 }</pre>	<pre>1 public class Calcolatrice { 2 public int divide(int a, int b) { 3 if(b == 0) { 4 throw new ArithmeticException("Cannot divide by zero"); 5 } 6 return a / b; 7 } 8 } 9 }</pre>
---	---

Prima

Dopo

Siamo, poi, passati al file *codemirror.css* per introdurre con il tema chiaro colori differenti per diverse parti di codice, come tag, stringhe, numeri, ecc

```

13 /* LIGHT THEME FONT EDITED*/
14 .CodeMirror {
15     /* Set height, width, borders, and global font properties here */
16     font-family: monospace;
17     height: 300px;
18     direction: ltr;
19
20     span.cm-comment { color: #123f67; } /*colore base 5c98cd*/
21     span.cm-string, span.cm-string-2 { color: #ffe600; } /*colore base #E6DB74*/
22     span.cm-number { color: #009aff; } /*colore base ca94ff*/
23     span.cm-variable { color: #009aff; } /*colore base #f8f8f2*/
24     span.cm-variable-2 { color: #009aff; }
25     span.cm-def { color: #37ff00; } /*colore base 72C05D*/
26     span.cm-operator { color: #009aff; }
27     span.cm-keyword { color: #ff0404; } /*colore base ff6541*/
28     span.cm-atom { color: #bd93f9; }
29     span.cm-meta { color: #000000; }
30     span.cm-tag { color: #bb2200; }
31     span.cm-attribute { color: #66D9EF; }
32     span.cm-qualifier { color: #72C05D; }
33     span.cm-property { color: #f8f8f2; }
34     span.cm-builtin { color: #72C05D; }
35     span.cm-variable-3, span.cm-type { color: #ff8400; }
36 }

```

Figure 3.8: /codemirror.css

Abbiamo aggiunto questa porzione di codice per introdurre il tema personalizzato di colori, facendo vari test per verificare quale fosse la combinazione migliore, muovendoci tramite codifica esadecimale per modificare il colore di ogni scritta e aiutandoci anche con i tool offerti da VSCode per muoversi nella tavolozza.

La scelta è ricaduta su colori molto accesi per spiccare con la tonalità della copertura precedentemente modificata.

<pre> 1 /*Compila i campi "Nome" e "Cognome" con le informazioni richieste 2 Nome: "inserire il proprio nome" 3 Cognome: "inserire il proprio cognome" 4 Username: tizianocaiol168@gmail.com 5 UserID: 1 6 Date: 30/9/2024 7 */ 8 9 import org.junit.Before; 10 import org.junit.After; 11 import org.junit.BeforeClass; 12 import org.junit.AfterClass; 13 import org.junit.Test; 14 import static org.junit.Assert.*; 15 16 public class TestCalcolatrice { 17 @BeforeClass 18 public static void setUpClass() { 19 // Eseguito una volta prima dell'inizio dei test nella classe </pre>	<pre> 1 /*Compila i campi "Nome" e "Cognome" con le informazioni richieste 2 Nome: "inserire il proprio nome" 3 Cognome: "inserire il proprio cognome" 4 Username: tizianocaiol168@gmail.com 5 UserID: 1 6 Date: 30/9/2024 7 */ 8 9 import org.junit.Before; 10 import org.junit.After; 11 import org.junit.BeforeClass; 12 import org.junit.AfterClass; 13 import org.junit.Test; 14 import static org.junit.Assert.*; 15 16 public class TestCalcolatrice { 17 @BeforeClass 18 public static void setUpClass() { 19 // Eseguito una volta prima dell'inizio dei test nella classe </pre>
--	--

Prima

Dopo

Infine, abbiamo modificato il file *lucario.css* in due punti.

Prima siamo intervenuti sulla porzione di codice sottostante per modificare il colore del background durante il tema scuro per una migliore visibilità e contrasto; la scelta è ricaduta sul colore #141429, una tonalità molto scura

di blu/grigio.

```

8 .cm-s-lucario.CodeMirror, .cm-s-lucario .CodeMirror-gutters {
9     background-color: #141429 !important; /*colore base 2e2e48*/
10    color: #f8f8f2 !important;
11    border: none;
12 }

```

Figure 3.9: Colore sfondo /lucario.css

Poi, abbiamo modificato singolarmente le tonalità dei tag, delle stringhe e delle variabili nel codice per migliorare il contrasto interno e con lo sfondo.

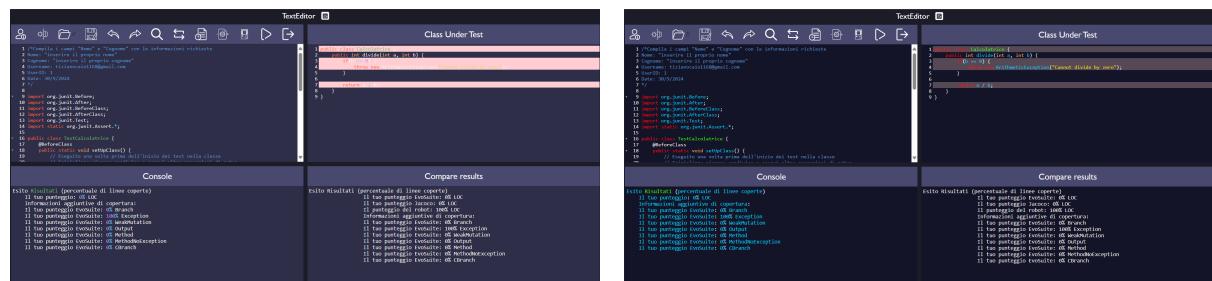
```

13 .cm-s-lucario .CodeMirror-gutters { color: #2b3e50; }
14 .cm-s-lucario .CodeMirror-cursor { border-left: solid thin #E6C845; }
15 .cm-s-lucario .CodeMirror-linenumber { color: #F8F8F2; }
16 .cm-s-lucario .CodeMirror-selected { background: #A243443; }
17 .cm-s-lucario .CodeMirror-line:selection, .cm-s-lucario .CodeMirror-line > span::selection, .cm-s-lucario .CodeMirror-line > span > span::selection { background: #243443; }
18 .cm-s-lucario .CodeMirror-line::-moz-selection, .cm-s-lucario .CodeMirror-line > span::-moz-selection, .cm-s-lucario .CodeMirror-line > span > span::-moz-selection { background: #243443; }
19 .cm-s-lucario span.cm-comment { color: #5c98d; }
20 .cm-s-lucario span.cm-string, .cm-s-lucario span.cm-string-2 { color: #fe600; } /*colore base #E6D874*/
21 .cm-s-lucario span.cm-number { color: #00d5ff; } /*colore base ca94ff*/
22 .cm-s-lucario span.cm-variable { color: #00d5ff; } /*colore base #f8f8f2*/
23 .cm-s-lucario span.cm-variable-2 { color: #00d5ff; }
24 .cm-s-lucario span.cm-def { color: #37ffff; } /*colore base 72C050*/
25 .cm-s-lucario span.cm-keyword { color: #609EF; } /*colore base #ff6541*/
26 .cm-s-lucario span.cm-atom { color: #B93F9; } /*colore base #F8F8F2; */
27 .cm-s-lucario span.cm-meta { color: #F8F8F2; }
28 .cm-s-lucario span.cm-tat { color: #F65541; }
29 .cm-s-lucario span.cm-attribute { color: #6609EF; }
30 .cm-s-lucario span.cm-qualifier { color: #72C050; }
31 .cm-s-lucario span.cm-property { color: #F8F8F2; }
32 .cm-s-lucario span.cm-builtin { color: #72C050; }
33 .cm-s-lucario span.cm-variable-3, .cm-s-lucario span.cm-type { color: #ffb86c; }
34 .
35 .cm-s-lucario .CodeMirror-activeLineBackground { background: #243443; }
36 .cm-s-lucario .CodeMirror-matchingBracket { text-decoration: underline; color: white !important; }
37

```

Figure 3.10: Colore codice /lucario.css

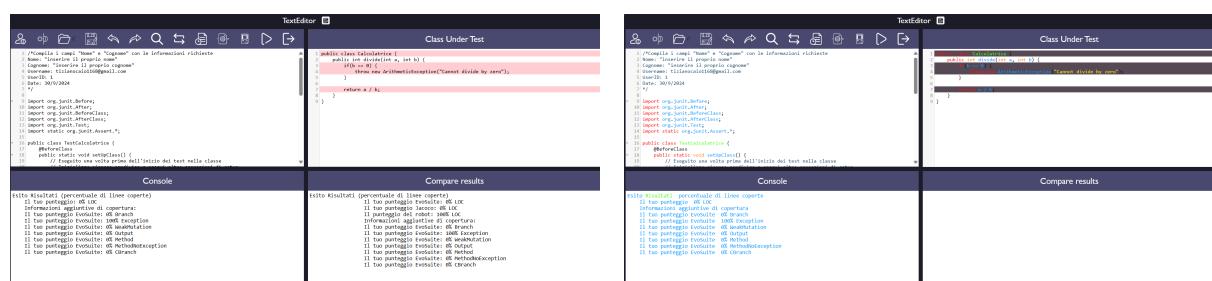
Il risultato finale complessivo è il seguente:



Prima

Dopo

Figure 3.11: Tema scuro



Prima

Dopo

Figure 3.12: Tema chiaro

3.3 Implementazione R4 - Nome classe in caricamento

Durante la fase di caricamento della classe da parte di un *admin* è presente un form di testo chiamato "Class Name" che consente di modificare il nome della classe inserita. Il problema sorge quando il nome inserito differisce da quello assegnato al file classe **.java** che si sta caricando, rendendo di fatto inutile la possibilità di poter inserire un nome che si preferisce (ex. Se il nome del file classe è *Calcolatrice.java* il campo "Class Name" dovrà avere per forza come valore stringa *Calcolatrice*).

Il requisito R4 richiede che la sezione di caricamento della classe nell'interfaccia *admin* deve assegnare automaticamente il nome della classe in base al nome del file caricato. L'obiettivo è eliminare la necessità di inserimento manuale del nome della classe da parte dell'*admin*, semplificando e velocizzando il processo di caricamento. Il sistema dovrà estrarre il nome del file senza l'estensione e utilizzarlo come nome predefinito per la classe.

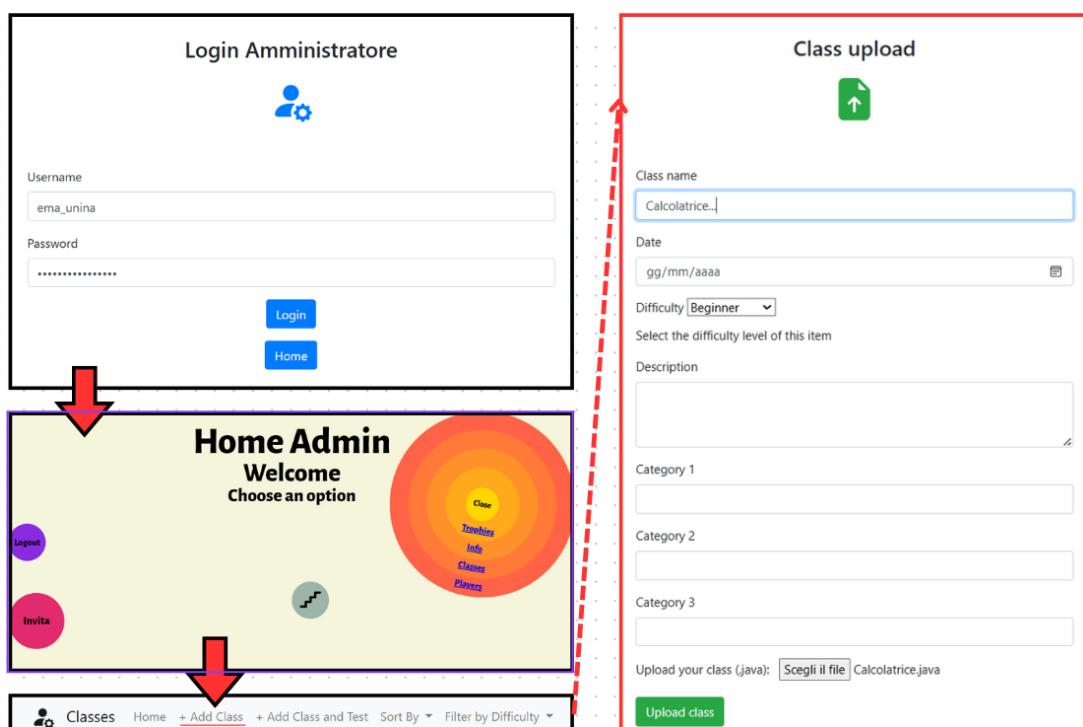


Figure 3.13: Pagina di caricamento classe prima delle modifiche

A seguito dello studio della documentazione A13 e dei file sorgente, siamo giunti alla conclusioni di dover intervenire su due file:

- **uploadClasse.html** - Implementa una pagina web che consente agli amministratori di caricare un file di classe (in formato .java) tramite un form, con la possibilità di inserire dettagli aggiuntivi come nome della classe, data, livello di difficoltà, categorie e descrizione.
- **HomeController.java** - Gestisce le richieste HTTP e interagisce con un database MongoDB per fornire risposte alle richieste del client.

```
..\\A13
└── \T1-G11
    └── \applicazione
        └── \manvsclass
            └── \src
                └── \main
                    ├── \resources
                    │   └── \templates
                    │       └── \uploadClasse.html
                    └── \java
                        └── \com
                            └── \groom
                                └── \manvsclass
                                    └── \controller
                                        └── HomeController.java
```

Figure 3.14: Directory dei file

3.3.1 Codice

/uploadClasse.html

Nel codice *html* relativo al form di caricamento della classe, abbiamo aggiunto alla riga che costituisce il campo dove inserire il nome della classe (che, come sudetto, è un'opzione inutile) la specifica *readonly* in modo da rendere il campo di sola lettura e non modificabile.

```
123 <div class="card-body">
124   <form id="uploadForm">
125     <div class="mb-3">
126       <label for="className" class="form-label">Class name</label>
127       <input type="text" class="form-control" id="className" aria-describedby="classNameHelp" readonly>
128     </div>
```

Figure 3.15: Specifica read-only

Abbiamo, poi, aggiunto nella sezione di *script* del codice *html* una funzione che aggiorna automaticamente il nome della classe basandosi sul nome del file *.java* caricato.

La funzione **updateClassName()** aggiorna automaticamente il campo "Class name" quando un file Java viene selezionato dall'utente. Essa prende il file caricato dall'elemento *input* (dove si sceglie il file), estraе il nome del file senza l'estensione (cioè rimuovendo *.java*), e imposta questo nome come valore del campo di testo "Class name". In questo modo, l'utente non deve inserire manualmente il nome della classe, riducendo il rischio di errori.

```
189 <script>
190 // Aggiorna automaticamente il nome della classe in base al nome del file
191 function updateClassName() {
192     const fileInput = document.getElementById('fileInput');
193     const file = fileInput.files[0];
194     if (file) {
195         const fileName = file.name.split('.')[0]; //Rimuove l'estensione
196         document.getElementById('className').value = fileName;
197     }
198 }
```

Figure 3.16: Funzione updateClassName()

Inoltre, abbiamo aggiornato la versione di **jQuery**² da utilizzare, in quanto inizialmente veniva utilizzata una versione datata. Ora è stata introdotta una versione più recente di jQuery, inclusa tramite il CDN³ <https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js> più moderna è meglio ottimizzata, più sicura, e che garantisce compatibilità con le funzionalità web più recenti.

```
185 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
186 <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>
187 <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.0/dist/js/bootstrap.bundle.min.js"></script>
```

Figure 3.17: Nuova versione di jQuery

²jQuery è una popolare libreria JavaScript che semplifica la scrittura di codice per manipolare il DOM (Document Object Model) di una pagina web, gestire eventi, eseguire animazioni e inviare richieste AJAX.

³Un CDN (Content Delivery Network) è una rete distribuita di server che ospita e distribuisce contenuti.

Nota bene che abbiamo lasciato anche la versione vecchia di jQuery da importare in quanto è necessaria per altri punti del codice.

/HomeController.java

Il file riguarda la gestione del caricamento e del download di file, che è tipico in un progetto più ampio di web application, gestendo le richieste HTTP del client.

Il pezzo di codice di nostro interesse è l'endpoint⁴ *POST /uploadFile* che permette agli utenti di caricare un file tramite una richiesta HTTP POST. Il file caricato viene elaborato e salvato, insieme ad altri metadati, come il nome del file, il modello associato e la dimensione. Il JWT (JSON Web Token) viene utilizzato per autenticare la richiesta e verificare se l'utente ha i permessi per eseguire l'operazione. Se l'autenticazione è valida, il file viene salvato nel filesystem (oppure in un database, a seconda delle modifiche). Una risposta JSON viene inviata al front-end, contenente informazioni sul file caricato, come la dimensione e il percorso per il download.

```

448     @PostMapping("/uploadFile")
449     @ResponseBody
450     public ResponseEntity<FileUploadResponse> uploadFile(
451         @RequestParam("file") MultipartFile classefile,
452         @RequestParam("model") String model,
453         @CookieValue(name = "jwt", required = false) String jwt,
454         HttpServletRequest request) throws IOException {
455
456         if (!isJwtValid(jwt)) {
457             System.out.println("Token valido (uploadFile)");
458             //Legge i metadati della classe della parte "model" del body HTTP e li salva in un oggetto ClassUT
459             ObjectMapper mapper = new ObjectMapper();
460             ClassUT classe = mapper.readValue(model, valueType=ClassUT.class);
461
462             //Salva il nome del file caricato
463             String fileName = StringUtil.cleanPath(classfile.getOriginalFilename());
464             long size = classfile.getSize();
465             //Imposta il nome della classe al nome del file (senza estensione)
466             String classNameFromfile = fileName.substring(0, fileName.lastIndexOf('.'));
467             classe.setNome(classNameFromfile); //Forza il nome della classe a essere uguale al nome del file
468             //Salva la classe nel filesystem condiviso
469             FileUploadUtil.saveClassFile(fileName, classe.getName(), classe);
470             //Genera un nome univoco per il file da salvare nel database
471             RobotUtil.generateAndSaveRobots(fileName, classe.getName(), classe);
472             //Prepara la risposta per il Front-end
473             FileUploadResponse response = new FileUploadResponse();
474             response.setFileName(fileName);
475             response.setSize(size);
476             response.setDownloadUri("./downloadFile");
477             //Setta data di caricamento e percorso di download
478             LocalDate today = LocalDate.now();
479             classe.setUrl("files-Upload/" + classe.getName() + "/" + fileName);
480             classe.setDate(today.toString());
481             DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
482             String data = today.format(formatter);
483             response.setOperazione(new Operazione(int) repto.count(), userAdmin.getUsername(), classe.getName(), type, data);
484             //Salva l'operazione fatta nel database
485             operazioneRepository.save(response);
486             System.out.println("Operazione completa con successo (uploadFile)");
487             return new ResponseEntity<(Response, HttpStatus.OK);
488         } else {
489             System.out.println("Token non valido (uploadfile)");
490             FileUploadResponse response = new FileUploadResponse();
491             response.setErrorMassage(errorMessage("Errore, il token non è valido"));
492             return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(response);
493         }
494     }

```

Figure 3.18: Codice uploadFile()

⁴Un endpoint rappresenta un punto di comunicazione a cui un client può inviare una richiesta per interagire con un servizio o un sistema.

La modifica principale è relativa al fatto che nel codice originale veniva usato il nome della classe estratto dall'oggetto *ClassUT* presente nel body HTTP (*classe.getName()*), non modificando il nome della classe. Ora, invece, viene impostato il nome della classe forzandolo a essere uguale al nome del file caricato (senza estensione).

```
468 | | | | String classNameFromFile = fileName.substring(beginIndex:0, fileName.lastIndexOf(ch:'.'));
469 | | | | classe.setName(classNameFromFile);
```

Figure 3.19: Codice

Inoltre, è stata apportata una modifica anche alla gestione della data in modo da migliorare l'affidabilità, la consistenza, la leggibilità e la QoL⁵ generale. La variabile *today* veniva usata per ottenere la data corrente, ma non era definita esplicitamente nel codice.

classe.setDate(today.toString());

Ora variabile *today* è correttamente definita come *LocalDate*:

LocalDate today = LocalDate.now();

Inoltre, veniva usato direttamente il metodo *today.toString()* per impostare la data. Siamo intervenuti per utilizzare un *DateTimeFormatter*⁶ con il formato "YYYY-MM-DD".

L'uso di *DateTimeFormatter* garantisce che la data sia salvata in un formato coerente, standardizzato e comunemente utilizzato nei database e nelle applicazioni. Questa modifica rende il codice più robusto e compatibile con eventuali specifiche del sistema.

```
489 | | | | DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd");
490 | | | | String data = today.format(formatter);
491 | | | | Operation operation1 = new Operation((int) orepo.count(), userAdmin.getUsername(), classe.getName(), 0, data);
```

Figure 3.20: Codice

⁵La QoL, acronimo di Quality of Life, è un concetto che si applica in ambito informatico, spesso riferito all'esperienza dell'utente (User Experience, UX) e alla soddisfazione degli utenti nei confronti di un software, di un sistema o di un servizio.

⁶DateTimeFormatter è una classe in Java, che fa parte delle API di data e ora, utilizzata per formattare e analizzare oggetti temporali.

In questo modo, poi, mentre prima creava l'oggetto *Operation* senza alcuna modifica al formato della data, ora la data formattata viene usata per creare l'oggetto *Operation*, assicurando che segua il formato "YYYY-MM-DD".

3.3.2 Testing funzionalità

Abbiamo verificato il corretto funzionamento dell'implementazione; ora caricata la classe il campo "Class Name" del form verrà aggiornato automaticamente e non sarà modificabile.

The screenshot shows a 'Class upload' interface. The 'Class name' field contains 'Calcolatrice' with a red border around it. The 'Date' field shows 'gg/mm/aaaa'. The 'Difficulty' dropdown is set to 'Beginner'. The 'Description' field is empty. There are three 'Category' fields: 'Category 1', 'Category 2', and 'Category 3', each with an empty input field. At the bottom, there's a file upload section labeled 'Upload your class (.java):' with a 'Scegli il file' button and a selected file 'Calcolatrice.java'. A green 'Upload class' button is at the bottom right.

Figure 3.21: Caricamento classe con modifiche apportate

3.4 Implementazione R5 - Refresh della pagina

L’obiettivo di questo requisito è garantire che, in caso di aggiornamento della pagina durante una sessione di gioco nell’editor, il sistema permetta al giocatore di iniziare una nuova partita senza generare errori. In precedenza, il refresh (F5) della pagina causava la perdita del codice di test e della classe sotto test; il codice della classe di test veniva ripristinato alla versione iniziale, mentre il codice della classe sotto test veniva completamente perso. Inoltre, un successivo tentativo di compilazione generava errori nel sistema, come la creazione di un sotto-albero con nome del Play e numero del Game null in VolumeT8. Anche un tentativo di calcolo della copertura causava un crash nell’applicazione prototipo20.

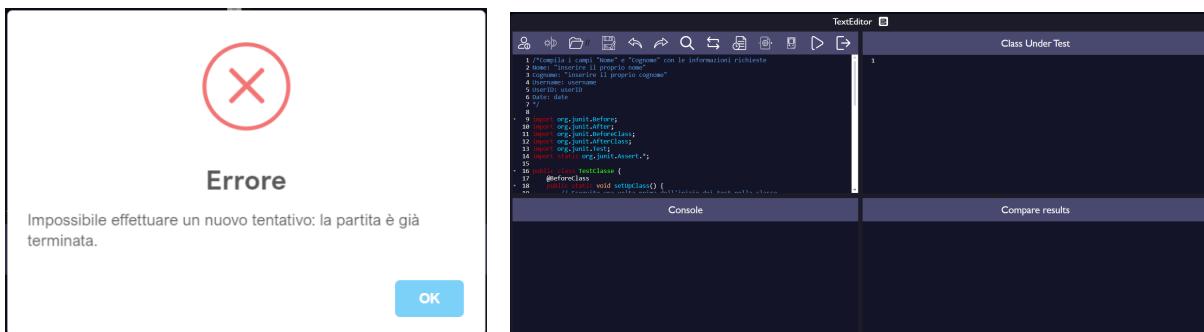


Figure 3.22: Errore restituito e di salvataggio prima delle modifiche

```
2024-05-13 20:08:02 (prova_esecuzione_parametri4.js) Creation of the directory where the test will be saved  
/VolumeT8/FolderTreeEvo/HSLColor/StudentLogin/Player14/Game16/Round16/Turn1/TestReport  
2024-05-13 20:08:03 node:internal/fs/utils:351  
2024-05-13 20:08:03 throw err;  
2024-05-13 20:08:03 ^  
2024-05-13 20:08:03  
2024-05-13 20:08:03 Error: no such file or directory, open  
'/VolumeT8/FolderTreeEvo/HSLColor/StudentLogin/Player11/Game11/Round11/Turn9/TestReport/GameData.csv'  
2024-05-13 20:08:03 at Object.openSync (node:fs:596:3)  
2024-05-13 20:08:03 at Object.readFileSync (node:fs:464:35)  
2024-05-13 20:08:03 at /app/Serv/prova_esecuzione_parametri4.js:180:55  
2024-05-13 20:08:03 at ChildProcess.exithandler (node:child_process:414:7)  
2024-05-13 20:08:03 at ChildProcess.emit (node:events:517:28)  
2024-05-13 20:08:03 at maybeClose (node:internal/child_process:1098:16)  
2024-05-13 20:08:03 at ChildProcess._handle.onexit (node:internal/child_process:303:5) {  
2024-05-13 20:08:03 errno: -2,  
2024-05-13 20:08:03 syscall: 'open',  
2024-05-13 20:08:03 code: 'ENOENT',  
2024-05-13 20:08:03 path:  
'/VolumeT8/FolderTreeEvo/HSLColor/StudentLogin/Player11/Game11/Round11/Turn9/TestReport/GameData.csv'  
2024-05-13 20:08:03 }
```

Figure 3.23: Errore copertura

A seguito dello studio della documentazione A13 e dei file sorgente, siamo giunti alla conclusione di dover intervenire sui seguenti file, tutti all'interno del task **T5**:

- **commonEditor.js** - Configura e gestisce un editor di codice utilizzando la libreria CodeMirror. Si concentra sulla gestione dell'editor e sulla sua visualizzazione, fornendo funzionalità di formattazione del codice, evidenziazione della sintassi e altre caratteristiche dell'editor di testo.
- **editor.js** - Gestisce diverse funzionalità legate all'inizio e al monitoraggio di una partita. Contiene logica per interagire con un server tramite chiamate API, raccogliendo informazioni sull'utente e sulle sue scelte di gioco, come il robot selezionato e la difficoltà.
- **main.js** - Gestisce l'interfaccia utente e le interazioni con il sistema di gioco, tra cui la selezione della modalità di gioco, la gestione del login e logout, e il flusso di dati tra il client e il server.

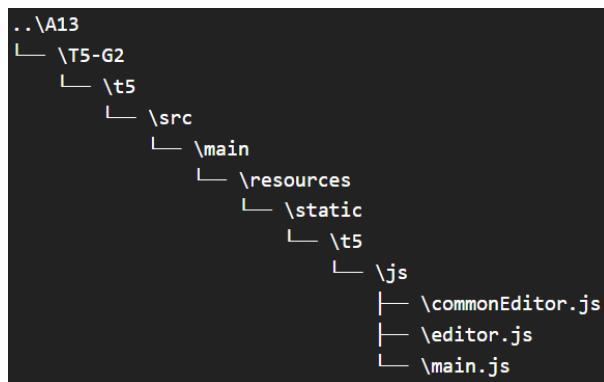


Figure 3.24: Directory file

3.4.1 Codice

/commonEditor.js

Nel file *commonEditor.js*, abbiamo introdotto delle funzionalità per risolvere il problema della perdita di dati quando la pagina viene ricaricata. Prima di questa modifica, un refresh della pagina di gioco comportava una serie di problemi e malfunzionamenti.

È stato, quindi, necessario intervenire sui file del task **T5**, in modo da poter ripristinare correttamente la pagina di gioco. Abbiamo aggiunto due funzioni principali, **saveEditorContent()** e **loadEditorContent()**, che utilizzano il `localStorage`⁷ del browser per memorizzare automaticamente il contenuto sia dell'editor principale che della classe sotto test.

- La funzione `saveEditorContent()` salva il testo attualmente presente nell'editor principale e nella sidebar all'interno del `localStorage`, utilizzando le chiavi `"editorContent"` e `"underTestContent"`. In questo modo, ogni modifica viene registrata automaticamente.

⁷Il LocalStorage è una funzionalità del browser che permette di memorizzare dati a livello locale sul dispositivo dell'utente sotto forma di coppie chiave-valore. I dati memorizzati persistono anche dopo che il browser è stato chiuso.

```

448     function saveEditorContent() {
449         var editorContent = editor.getValue();
450         var underTestContent = sidebarEditor.getValue();
451         localStorage.setItem("editorContent", editorContent);
452         localStorage.setItem("underTestContent", underTestContent);
453     }

```

Figure 3.25: funzione saveEditorContent()

- La funzione *loadEditorContent()* viene eseguita quando la pagina viene ricaricata e ripristina i contenuti precedentemente salvati negli editor. Se nel localStorage sono presenti dati, questi vengono reinseriti nei rispettivi editor, permettendo all'utente di riprendere esattamente da dove aveva lasciato.

```

440     function loadEditorContent() {
441         var savedEditorContent = localStorage.getItem("editorContent");
442         var savedUnderTestContent = localStorage.getItem("underTestContent");
443
444         if (savedEditorContent !== null) {
445             editor.setValue(savedEditorContent);
446         }
447         if (savedUnderTestContent !== null) {
448             sidebarEditor.setValue(savedUnderTestContent);
449         }
450     }

```

Figure 3.26: funzione loadEditorContent()

Abbiamo associato un evento *onChange* ad entrambi gli editor. Questo assicura che ogni modifica fatta al codice venga automaticamente salvata nel localStorage, prevenendo la perdita di dati.

```

468     editor.on("change", function () {
469         saveEditorContent();
470     });
471
472     sidebarEditor.on("change", function () {
473         saveEditorContent();
474     });

```

Figure 3.27: Evento onChange

/editor.js

Inizialmente, nel file **editor.js** era presente la funzione *window.onbeforeunload*, la quale aveva il compito di resettare alcuni valori nel localStorage, impo-

standoli a null. Lo scopo di questa funzione era quello di pulire i dati salvati nel localStorage, rimuovendo le informazioni relative alla sessione attuale come l'ID del gioco, l'ID del turno, la classe sotto test, il robot selezionato e il livello di difficoltà. Per provare a risolvere il problema del *refresh*, in un primo momento, abbiamo rimosso questa funzione.

```
776 window.onbeforeunload = function () {  
777   if (localStorage.getItem("modalita") != "Scalata") {  
778     localStorage.setItem("gameId", null);  
779     localStorage.setItem("turnId", null);  
780     localStorage.setItem("classe", null);  
781     localStorage.setItem("robot", null);  
782     localStorage.setItem("difficulty", null);  
783   }  
784};
```

Figure 3.28: Funzione window.onbeforeunload

In questo modo, i dati relativi alla sessione, vengono mantenuti e non cancellati automaticamente, permettendo il corretto ripristino delle informazioni. Inoltre, è possibile giocare una nuova partita anche dopo il refresh, cosa che prima non accadeva a causa dell'assegnazione del valore null a gameId. Tuttavia, ci siamo resi conto che quando provavamo a giocare una nuova partita o ad effettuare un calcolo della copertura continuavamo ad avere lo stesso errore, poiché le variabili non venivano aggiornate correttamente. Al refresh della pagina, il sistema continuava a utilizzare i valori di gameId, turnId e altre variabili presenti nel localStorage. Questo comportava un'errata creazione del percorso di salvataggio dei file, causando errori ENOENT (file o directory non trovati). Per gestire le diverse partite giocate dopo aver effettuato il refresh abbiamo deciso di intervenire sull'organizzazione dei turni. Dunque, nell'**editor.js**, è stata aggiunta una gestione tramite *localStorage* per la variabile *orderTurno*. Questa modifica assicura che, in caso di refresh della pagina, il sistema riconosca la nuova partita come un nuovo turno. Ad esempio, se il giocatore si trova su *Player1/Game1/Round1/Turn1*, dopo il refresh la nuova partita sarà considerata come *Player1/Game1/Round1/Turn2*.

```
23 |   orderTurno = localStorage.getItem("orderTurno");
```

Figure 3.29: LocalStorage

Nel flusso della gestione dei dati dei turni di gioco, effettuiamo l'incremento di *orderTurno* che avviene quando un turno di gioco viene completato. Il codice verifica lo stato di *orderTurno* e lo incrementa prima di procedere ulteriormente. Dopo aver incrementato il valore di *orderTurno*, il nuovo valore viene immediatamente salvato nel *localStorage*. Questa modifica è stata introdotta per tenere traccia del numero di turni effettuati durante la partita. Grazie a questa nuova organizzazione dei diversi turni il sistema non genera percorsi sbagliati, risolvendo l'errore ENOENT e altri problemi di accesso ai file.

```
265 |   orderTurno++;
266 |   localStorage.setItem("orderTurno", orderTurno);
267 |
```

Inoltre, abbiamo modificato la funzione **ajax**, definita nell'*editor.js*, che ha il compito di ricevere il codice sorgente di una classe sotto test dall'endpoint */api/receiveClassUnderTest*.

```
51 | $ajax({
52 |   url: "/api/receiveClassUnderTest",
53 |   type: "GET",
54 |   data: {
55 |     idUtente: idUtente,
56 |     idPartita: idPartita,
57 |     idTurno: idTurno,
58 |     nomeCUT: nomeCUT,
59 |     robotScelto: robotScelto,
60 |     difficolta: difficolta
61 |   },
62 |   dataType: "text",
63 |   success: function (response) {
64 |     console.log(nameCUT+`SourceCode \n\n\n`+JSON.parse(response).class);
65 |     // Mantieni il contenuto dell'editor prima di sovrascriverlo
66 |     var currentEditorContent = editor.getValue();
67 |     // Imposta il contenuto della classe sotto test solo nel sidebarEditor
68 |     sidebarEditor.setValue(JSON.parse(response).class);
69 |     // Formata date to dd/mm/yyyy
70 |     var formattedDate = currentDate.getDate() + "/" + (currentDate.getMonth() + 1) + "/" + currentDate.getFullYear();
71 |     console.log(`formattedDate: ${formattedDate}`);
72 |     // Ottieni il contenuto dell'editor principale (dove si trova TestClasse)
73 |     var textAreaContent = currentEditorContent;
74 |     // Sostituisci "testClasse" con il nome della classe sotto test
75 |     newContent = textAreaContent.replace("testClasse", testClassName);
76 |     // Sostituisci "username" con il nome utente del giocatore
77 |     newContent = newContent.replace("username", parseJwt(getCookie("jwt")).sub);
78 |     // Sostituisci "userID" con l'ID del giocatore
79 |     newContent = newContent.replace("userID", parseJwt(getCookie("jwt")).userId);
80 |     // Sostituisci "date" con la data corrente
81 |     newContent = newContent.replace("date", formattedDate);
82 |     console.log(`newContent \n\n${newContent}`);
83 |     // Mantieni le modifiche esistenti e imposta il nuovo contenuto solo con i cambiamenti necessari
84 |     editor.setValue(newContent);
85 |     alert(`Classe: ${nameCUT} .java ricevuta con successo`);
86 |   },
87 |   error: function () {
88 |     console.log(`Errore durante la ricezione del file ${nameCUT}.java`);
89 |   }
90 | },
```

Figure 3.30: Chiamata ajax

In particolare:

- Imposta il codice della classe ricevuta nell'editor laterale (sidebarEditor), lasciando intatto l'editor principale.
- Recupera il contenuto corrente dell'editor principale e applica una serie di sostituzioni personalizzate, come il nome della classe sotto test, il nome utente, l'ID del giocatore e la data corrente.
- Dopo le sostituzioni, aggiorna l'editor principale con il nuovo contenuto elaborato, mantenendo eventuali modifiche preesistenti.

/main.js

Nella funzione *redirectToPageeditor()* del file main.js, è stato modificato il valore di inizializzazione della variabile orderTurno per la modalità "Sfida". Precedentemente, orderTurno veniva impostato a 1, mentre ora viene impostato a 0, assicurando che il sistema riconosca correttamente l'inizio di un nuovo turno.

```

255   },
256   type: 'POST',
257   traditional: true,
258   success: function (response) {
259     localStorage.setItem("gameId", response.game_id);
260     localStorage.setItem("turnId", response.turn_id);
261     localStorage.setItem("roundId", response.round_id);
262     localStorage.setItem("orderTurno", "0");
263     window.location.href = "/editor";
264   },

```

Figure 3.31: Modifica orderTurno

Sempre nella funzione *redirectToPageeditor()*, vengono aggiunti e utilizzati due nuovi comandi:

```

262   localStorage.removeItem("editorContent");
263   localStorage.removeItem("underTestContent");

```

Questi comandi, vengono eseguiti dopo che i dati di gioco sono stati inviati con successo al server. Questo significa che prima di iniziare una nuova

partita, il contenuto precedente viene eliminato.

Le due istruzioni servono a ripulire lo stato del gioco e dell'editor, evitando che i vecchi dati rimangano memorizzati quando non sono più necessari, viene, quindi, effettuata una pulizia tra una partita e l'altra:

- "*editorContent*" e "*underTestContent*" contengono rispettivamente il codice dei test e la classe sotto test (CUT) che l'utente stava utilizzando nella partita precedente. Rimuoverli dal localStorage serve a resettare l'editor prima di una nuova partita, assicurando che i vecchi dati non interferiscano con il nuovo gioco.

Chapter 4

Implementazione requisito

R1 - Pagina Profilo Utente

L’obiettivo di questo requisito è la creazione di una nuova pagina dedicata al profilo utente, accessibile dalla home page ed univoca per ogni giocatore. Il profilo includerà diverse informazioni essenziali, tra cui: *nome, cognome, email/username, punti accumulati, partite giocate e tempo totale trascorso nel gioco*. Grazie a questa implementazione, ogni giocatore dispone ora di un’area personale dove può visualizzare e gestire i propri dati, contribuendo a migliorare l’esperienza di gioco e garantendo un accesso facile e veloce alle informazioni rilevanti.

Per soddisfare le richieste del requisito R1, è stato necessario analizzare la struttura delle pagine HTML e modificare file già esistenti. Le modifiche effettuate riguardano principalmente il task **T5**, che ha comportato l’aggiunta di due nuove pagine HTML e CSS per il profilo, più ulteriori interventi ai file necessari per garantire la corretta integrazione. Inoltre, per il recupero delle informazioni, ci siamo serviti dei task **T2-3** per recuperare i dati utente e del task **T4** per ottenere le informazioni relative al punteggio, al numero

di partite e al tempo di gioco. Infine, tramite **UIGateway**, è stata inserita una nuova rotta¹ per gestire il profilo.

Dove siamo intervenuti nel task T5:

- main.html – Aggiunta del pulsante "**Profile**" per accedere alla nuova pagina del profilo.
- main.js – Aggiunta della funzione `redirectToProfile()` per il reindirizzamento alla pagina del profilo.
- Creazione di una nuova pagina ***profile.html***.
- Creazione di una nuova pagina ***profile.js***.
- Creazione di un nuovo controller dedicato al profilo utente: ***ProfileController.java***.
- Creazione del file ***profile.css*** per la gestione dello stile e dell’interfaccia della pagina profilo.

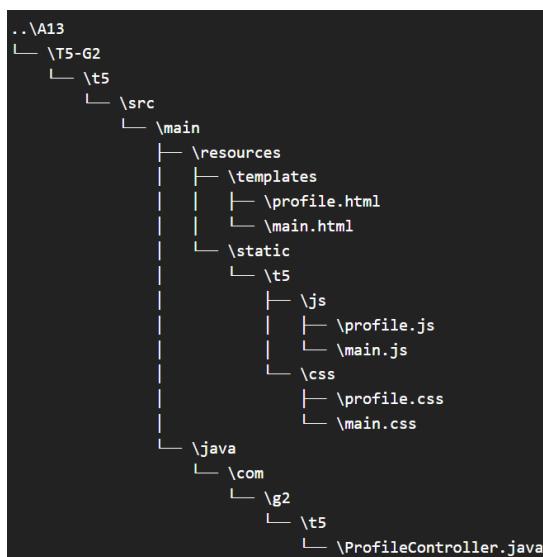


Figure 4.1: Directory file T5

¹In HTML, la "rotta" (o "percorso", in inglese route o path) si riferisce al percorso di un file o una risorsa a cui il browser deve accedere. Può trattarsi di un file HTML, CSS, JavaScript o di qualsiasi altra risorsa.

Nei task T2-3:

- controller.java - Aggiunta di parametri per poter prendere le informazioni necessarie.

```
..\\A13
└── \T23-61
    └── \src
        └── \main
            └── \java
                └── \com
                    └── \example
                        └── \db_setup
                            └── \Controller.java
```

Figure 4.2: Directory file T2-3

Modifica apportata nell'UI Gateway:

- default.conf – Aggiunta della voce "*/profile*" sotto UIGateway per la gestione della nuova rottta.

```
..\\A13
└── \ui_gateway
    └── \default.conf
```

Figure 4.3: Directory file UI Gateway

4.1 Implementazione Task 5

In questa sezione descriveremo le implementazioni relative al codice presente nel task T5.

4.1.1 /main.html

Come prima cosa, abbiamo introdotto nella pagina "home" del lato *user* un nuovo **button "Profile"** che serve ad indirizzare l'utente sulla nuova pagina Profilo Utente che abbiamo creato.

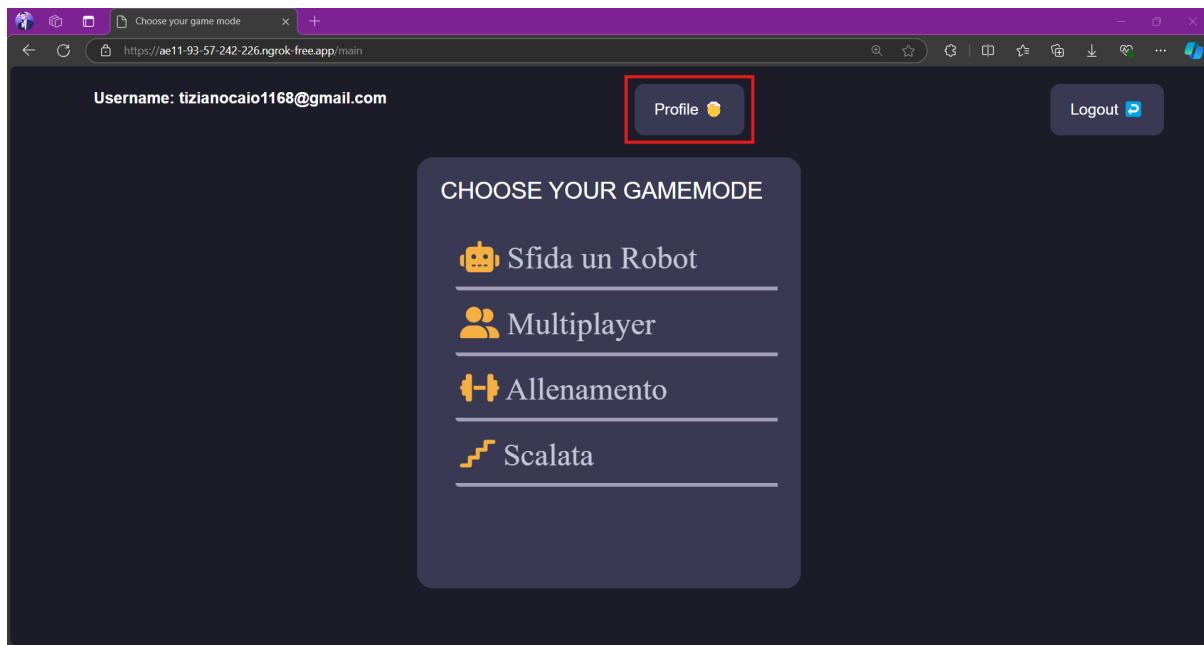


Figure 4.4: Pagina principale lato User

Abbiamo modificato il *body HTML* inserendo un pulsante tipo *button*:

```
30 | | | <button id="profileButton" type="button" class="profile-button" onclick="redirectToProfile()">Profile 🌟</button>
```

Principali caratteristiche:

- Classe (class="profile-button"): Applica uno stile specifico al pulsante tramite la classe *CSS profile-button*. La definizione di questa classe si trova nel foglio di stile *main.css* associato al progetto.
- Attributo onclick (onclick="redirectToProfile()"): Specifica che al click sul pulsante viene eseguita una funzione JavaScript chiamata *redirectToProfile()*, che reindirizza l'utente alla pagina del profilo.

Modifica di /main.css

La classe "*profile-button*" di cui prima, per applicare lo stile al pulsante, è stata inserita nel file .CSS della main page (*main.css*) come segue.

```
660 .profile-button {  
661   background-color: #rgba(57,57,83,1);  
662   color: #fff;  
663   border: #rgba(57,57,83,1);  
664   padding: 10px 20px;  
665   font-size: 16px;  
666   cursor: pointer;  
667   border-radius: 10px;  
668   font-family: Arial, Helvetica, sans-serif;  
669   position: relative;  
670   top: 10px;  
671   right: 60px;  
672 }  
673  
674 .profile-button:focus {  
675   outline:none; /*Rimuove il colore di sfondo dei bordi quando il tasto è attivo*/  
676 }  
677 .profile-button:hover {  
678   background-color: #white; /*Cambia il colore di sfondo al passaggio del cursore*/  
679   color: #black;  
680 }
```

Figure 4.5: Stile del pulsante Profile

Quando il pulsante è attivo, non mostra alcun contorno visibile (*outline: none*), migliorando l'aspetto pulito durante la navigazione tramite tastiera. Quando l'utente passa con il mouse sopra il pulsante (*:hover*), il colore di sfondo cambia in bianco e il testo diventa nero, offrendo un effetto visivo chiaro che segnala l'interazione. Questo effetto rende il pulsante più user-friendly.

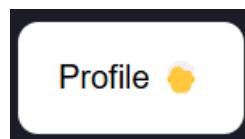


Figure 4.6: Pulsante al passaggio del cursore

4.1.2 /main.js

Questo file ha il compito di gestire diverse funzionalità interattive dell'interfaccia utente nella web application, utilizzando JavaScript per collegare la logica di interazione con il backend e aggiornare il comportamento del front-end in risposta alle azioni dell'utente.

Siamo intervenuti dichiarando una nuova funzione *redirectToProfile()*, una

semplice funzione JavaScript che si occupa di reindirizzare l’utente alla pagina del profilo, utilizzando l’URL “/profile”. Utilizza il metodo `window.location.href`² per cambiare la pagina corrente dell’utente.

```
347  function redirectToProfile() {  
348    |   window.location.href = "/profile";  
349  }
```

Figure 4.7: Codice redirectToProfile()

4.1.3 /profile.html, /profile.js & /profile.css

In questo paragrafo descriveremo la creazione dei file principali relativi alla pagine profilo utente: quella della struttura HTML, JavaScript e quella dello stile CSS.

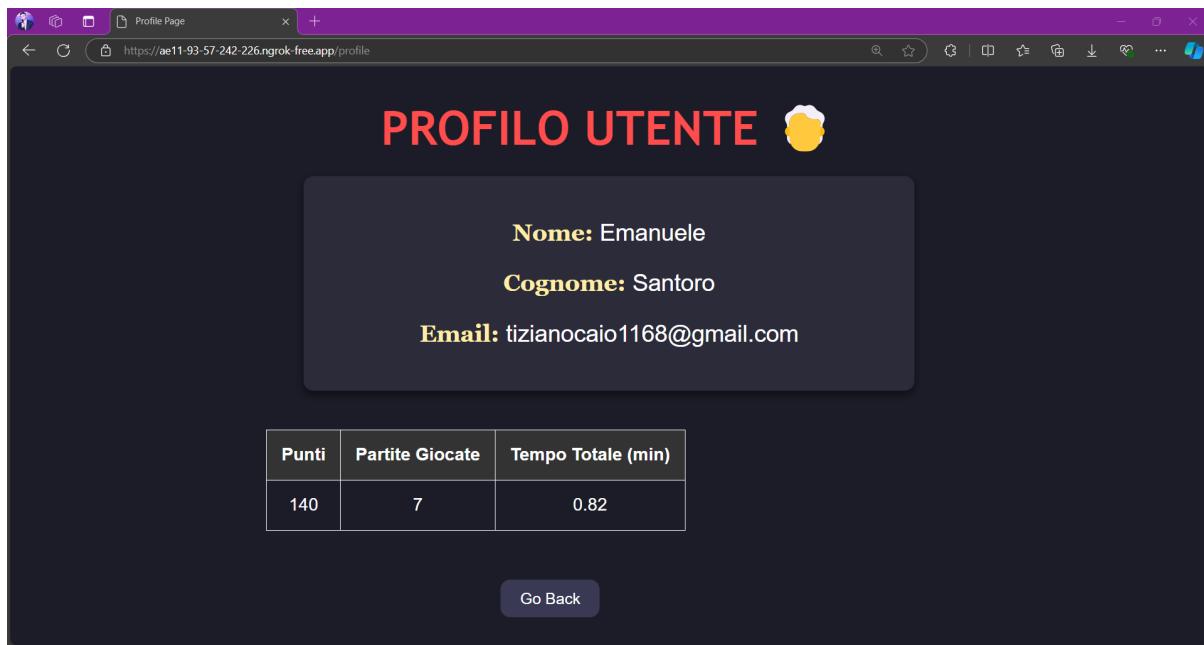
/profile.html & /profile.js

La pagina `profile.html` è progettata per visualizzare il Profilo Utente e fornisce un’interfaccia dove l’utente può vedere i propri dati personali e statistiche relative al gioco.

Di seguito entreremo nel dettaglio solo delle sue funzionalità principali, per il codice completo fare riferimento al file di progetto.

Questa pagina consente all’utente di visualizzare informazioni come il proprio nome, cognome, email (o username) e statistiche personali riguardanti le partite giocate nel sistema. Funziona come una dashboard dove l’utente può accedere rapidamente ai dati del proprio profilo.

²E’ una proprietà di JavaScript che può essere usata sia per ottenere l’URL della pagina corrente sia per impostare un nuovo URL, provocando un reindirizzamento della pagina.



Struttura della pagina:

1. Intestazione (*head*) - Include collegamenti ai fogli di stile CSS (per il design della pagina), icone FontAwesome, SweetAlert e la libreria JQuery (per le richieste AJAX³).

```

4  <head>
5      <link href="./t5/css/profile.css" rel="stylesheet" />
6
7      <!-- FontAwesome icons -->
8      <script src="https://kit.fontawesome.com/d963f915e6.js" crossorigin="anonymous"></script>
9      <!-- Sweet Alert -->
10     <script src="https://unpkg.com/sweetalert/dist/sweetalert.min.js"></script>
11     <!-- JQuery -->
12     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
13
14     <title>Pagina Profilo</title>
15 </head>
```

Figure 4.8: Intestazione

2. Corpo della pagina (*body*) - La pagina mostra il nome, cognome ed email dell'utente in formato compatto, recuperati dal token JWT salvato nel cookie dell'utente. Questi dati sono visualizzati automaticamente quando l'utente accede al proprio profilo.

³Le richieste AJAX (Asynchronous JavaScript and XML) sono un metodo utilizzato in JavaScript per inviare e ricevere dati dal server in modo asincrono, senza dover ricaricare completamente una pagina web, in modo dinamico

```

19 |     <div class="container">
20 |
21 |         <span class="title">PROFILO UTENTE </span>
22 |         <!-- Qui verranno mostrati nome, cognome e username in formato compatto -->
23 |         <div class="user-info">
24 |             <p><span class="bold">Nome:</span> <span id="nome"></span></p>
25 |             <p><span class="bold">Cognome:</span> <span id="cognome"></span></p>
26 |             <p><span class="bold">Email:</span> <span id="email"></span></p> <!-- Email aggiunta qui -->
27 |         </div>

```

Figure 4.9: Box dei dati

3. Tabella Dati Utente - Una tabella mostra ulteriori dettagli sulle statistiche del gioco, come il punteggio, il numero di partite giocate e il tempo totale di gioco, quando i dati sono disponibili. Se l'utente non ha giocato alcuna partita, viene visualizzato un messaggio che informa l'utente con la frase "Nessuna partita giocata".

```

29 |     <table id="userDetailsTable" style="display:none;">
30 |         <thead>
31 |             <tr>
32 |                 <th>Punti</th>
33 |                 <th>Partite Giocate</th>
34 |                 <th>Tempo Totale (min)</th>
35 |             </tr>
36 |         </thead>
37 |         <tbody>
38 |             <!-- I dati saranno inseriti qui dinamicamente -->
39 |         </tbody>
40 |     </table>
41 |     <!-- Aggiunta per il messaggio nessuna partita -->
42 |     <p id="noGamesMessage" style="display:none;">Nessuna partita giocata.</p>
43 |
44 | 
```

Figure 4.10: Tabella delle statistiche

Per quanto riguarda le funzionalità principali tramite JavaScript:

- Gestione del token JWT - La pagina utilizza funzioni JavaScript per decodificare il token JWT (JSON Web Token) dell'utente dal cookie, che contiene informazioni personali come nome, cognome e email.

```

61     function getJwtToken() {
62         const cookieString = document.cookie;
63         const cookies = cookieString.split(';');
64         for (let cookie of cookies) {
65             const [name, value] = cookie.split('=');
66             if (name.trim() === 'jwt') {
67                 return value;
68             }
69         }
70         return null;
71     }

```

Figure 4.11: Tabella delle statistiche

Cerca tra i cookie del browser quello chiamato jwt e restituisce il suo valore, che contiene il token JWT dell’utente loggato. Viene chiamata all’inizio del caricamento della pagina per recuperare il token cheterrà le informazioni sull’utente.

```

50     function parseJwt(token) {
51         var base64Url = token.split('.')[1];
52         var base64 = base64Url.replace(/-/g, '+').replace(/\_/g, '/');
53         var jsonPayload = decodeURIComponent(atob(base64).split('').map(function (c) {
54             return '%' + ('00' + c.charCodeAt(0).toString(16)).slice(-2);
55         }).join(''));
56
57         return JSON.parse(jsonPayload);
58     }

```

Figure 4.12: Tabella delle statistiche

Decodifica la parte del payload⁴ del token JWT (che è in formato base64⁵) e lo trasforma in un oggetto JavaScript. Questa funzione permette di estrarre i dati dell’utente memorizzati nel token.

- Popolamento dinamico delle informazioni utente - Dopo aver decodificato il token JWT, lo script estrae le informazioni dell’utente e le inserisce dinamicamente nella pagina HTML.

⁴Quando si parla del payload di un token (come nel caso dei JWT), ci si riferisce alla parte del token che contiene le informazioni o i dati veri e propri che il token vuole trasportare.

⁵Il Base64 è un sistema di codifica che viene utilizzato per rappresentare dati binari (come immagini, file o testi) in formato testuale, utilizzando solo caratteri ASCII.

```

83     document.addEventListener("DOMContentLoaded", function () {
84         const jwtToken = getJwtToken();
85         if (!jwtToken) {
86             console.error('JWT token non trovato.');
87             return;
88         }
89
90         //Decodifica il token JWT per ottenere i dati utente
91         const userData = parseJwt(jwtToken);
92
93         //Estrae nome, cognome ed email dal token JWT
94         const nome = userData.nome;
95         const cognome = userData.cognome;
96         const email = userData.sub;
97         const userId = userData.userId;
98
99         //Mostra il nome, cognome ed email dell'utente nel profilo
100        document.getElementById("nome").textContent = nome;
101        document.getElementById("cognome").textContent = cognome;
102        document.getElementById("email").textContent = email;

```

Figure 4.13: Tabella delle statistiche

Quando la pagina viene caricata (*DOMContentLoaded*⁶), il token JWT viene recuperato e decodificato. Successivamente, il nome, cognome ed email dell’utente vengono inseriti nei rispettivi elementi HTML (con ID nome, cognome, e email).

- Recupero dei dati di gioco - Lo script effettua una richiesta asincrona *fetch()* al server per ottenere le statistiche di gioco dell’utente e visualizzarle nella pagina.

⁶DOMContentLoaded è un evento in JavaScript che viene attivato quando il documento HTML è stato completamente caricato e analizzato, ma senza attendere che tutte le risorse esterne (come immagini, fogli di stile o frame) siano state completamente caricate.

```

105    fetch('/games')
106      .then(response => response.json())
107      .then(gameData => {
108        const accountStats = {};
109
110        gameData.data.forEach(entry => {
111          const timeDifference = calculateTimeDifference(entry.startedAt, entry.closedAt);
112
113          entry.players.forEach(player => {
114            const accountId = player.accountId;
115
116            //Aggiunge o aggiorna i dati relativi al conteggio delle partite e al tempo totale
117            if (!accountStats[accountId]) {
118              accountStats[accountId] = { count: 1, totalTime: timeDifference, score: entry.score || 0 };
119            } else {
120              accountStats[accountId].count++;
121              accountStats[accountId].totalTime += timeDifference;
122              accountStats[accountId].score += entry.score || 0;
123            }
124          });
125        });
126
127        //Mostra i dati dell'utente loggato
128        const userStats = accountStats[userId];
129        if (userStats) {
130          document.getElementById("userDetailsTable").style.display = "block";
131
132          const tableBody = document.querySelector("#userDetailsTable tbody");
133          const row = document.createElement("tr");
134          row.innerHTML =
135            `
136              <td>${userStats.score}</td>
137              <td>${userStats.count}</td>
138              <td>${userStats.totalTime.toFixed(2)}</td>
139            `;
140          tableBody.appendChild(row);
141        } else {
142          //Mostra il messaggio "Nessuna partita giocata" se non ci sono dati
143          document.getElementById("userDetailsTable").style.display = "block";
144          document.getElementById("noGamesMessage").style.display = "block";
145        }
146      })
147      .catch(error => {
148        console.error('Errore durante la richiesta dei dati:', error);
149      });

```

Figure 4.14: Tabella delle statistiche

Per identificare l'utente specifico, lo script JavaScript decodifica il token JWT salvato nel cookie del browser, estraendo l'accountId, che viene poi utilizzato per filtrare le partite appartenenti all'utente. L'endpoint supporta anche filtri per intervalli di date e la paginazione, ma nel nostro caso utilizziamo l'accountId per ottenere solo le partite giocate dall'utente corrente. Una volta ricevuti i dati, lo script elabora le informazioni per ogni partita, mostrando nella tabella del profilo il numero di partite giocate, il punteggio totale accumulato e altre metriche rilevanti, come il tempo totale di gioco. Se l'utente non ha ancora giocato partite, viene mostrato il messaggio "Nessuna partita giocata".

Sapendo che il Task **T4** è responsabile della gestione e archiviazione dei

dati di gioco, abbiamo riconosciuto la necessità di utilizzare il *Game Repository* per recuperare le informazioni necessarie. Grazie alla documentazione generata con Swagger⁷, che il team del Task T4 ha utilizzato per definire in modo chiaro gli endpoint e le operazioni disponibili, abbiamo individuato l'endpoint /games che fornisce i dati delle partite giocate dall'utente autenticato.

The screenshot shows the Swagger UI interface. On the left, there's a sidebar with 'Info', 'Tags', and 'Servers' sections, and a search bar. Below these are lists of HTTP methods and their corresponding URLs: PUT /games/{id}, DELETE /games/{id}, POST /games, and GET /games. Under 'GET /games', there are sections for 'rounds', 'turns', and 'turns/{id}/files'. The main area displays the JSON schema for the GET /games operation. The schema includes tags: 'games' and 'summary: Retrieve games by date interval'. It also includes a detailed description: 'Retrieve games in the system within an interval and paginated. Default interval is current day.' Parameters are defined as follows:

- query**: startDate (string, required: false, default: 2022-01-30)
- query**: endDate (string, required: false, default: 2023-12-31)
- query**: page (integer, required: false, default: 1)
- query**: pageSize (integer, required: false, default: 100)
- query**: accountId (string, required: false)

At the bottom of the schema section, it says 'Last Saved: 3:35:05 pm - Oct 6, 2022'. On the right, there's a 'Try it out!' button and a note: 'Attiva Windows Passa a Impostazioni per attivare Windows'.

Figure 4.15: Swagger

- Calcolo del tempo di gioco - Lo script include una funzione per calcolare il tempo trascorso in ogni partita, utilizzando le date di inizio e fine di ciascuna sessione di gioco.

```

74     function calculateTimeDifference(startedAt, closedAt) {
75         if (!closedAt) {
76             return 0;
77         }
78         const startTimestamp = new Date(startedAt).getTime();
79         const closedTimestamp = new Date(closedAt).getTime();
80         return (closedTimestamp - startTimestamp) / 60000; //Differenza in minuti
81     }

```

Figure 4.16: Tabella delle statistiche

Calcola la differenza tra la data di inizio (*startedAt*) e la data di fine (*closedAt*) di una partita, restituendo il risultato in minuti.

- Pulsante "Go Back" - Alla fine della pagina, c'è un pulsante che con-

⁷Swagger è uno strumento che facilita la progettazione, documentazione e test delle API.

sente all'utente di tornare alla pagina principale dell'applicazione cliccandoci.

```
157 | <button type="button" class="go-back-button" onclick="redirectToPagemain(); saveLoginData()">Go Back</button>
```

Figure 4.17: Tabella delle statistiche

Il pulsante esegue una funzione JavaScript che reindirizza l'utente a "/main". La funzione è *redirectToPagemain()*.

```
151 |     function redirectToPagemain() {
152 |       window.location.href = "/main";
153 |     }
```

Figure 4.18: Tabella delle statistiche

/profile.css

Il file contiene vari stili che gestiscono l'aspetto visivo della pagina web Profilo Utente.

Ecco un focus dettagliato sugli elementi chiave: il box informazioni utente e la tabella delle statistiche.

Box informazioni utente:

- Il box utilizza un colore grigio scuro, che contrasta con lo sfondo del corpo principale (più scuro), rendendo il box ben visibile ma non troppo invasivo.
- L'interno del box ha un *padding*, che crea uno spazio sufficiente tra il contenuto (testo o elementi) e i bordi, migliorando la leggibilità.

```

13 body {
14     background-color: #1c1c28;
15     color: #ffffff;
16     font-family: Arial, sans-serif;
17     text-align: center; /*Testo centrato per tutti gli elementi*/
18     padding: 20px; /*Aggiunto padding*/
19 }
20
21 .user-info {
22     background-color: #2b2b3a; /*Colore più chiaro rispetto allo sfondo del body*/
23     padding: 20px;
24     border-radius: 10px; /*Angoli arrotondati per il box*/
25     width: 50%;
26     margin: 20px auto;
27     box-shadow: 0 4px 8px rgba(0, 0, 0, 0.5);
28     font-size: 24px;
29 }
30
31 /*Stile per rendere il testo in grassetto*/
32 .bold {
33     font-weight: bold;
34     color: #ffeda5;
35     font-family: 'Georgia', serif;
36 }

```

Figure 4.19: Tabella delle statistiche

Tabella delle statistiche:

- La tabella utilizza la proprietà *border-collapse: collapse*, che rimuove gli spazi tra i bordi delle celle, rendendo la tabella più compatta e ordinata.
- Ogni cella ha un *padding*, il che aumenta la spaziatura interna e rende i contenuti della tabella più leggibili.

```

39 table {
40     margin: 40px auto;
41     border-collapse: collapse;
42     width: 60%;
43 }
44
45 table th, table td {
46     border: 1px solid #ffffff;
47     padding: 15px; /*Aumentata la spaziatura per una lettura migliore*/
48     font-size: 18px;
49     text-align: center;
50 }
51
52 table th {
53     background-color: #333333;
54 }

```

Figure 4.20: Tabella delle statistiche

Questi elementi, combinati, creano un'interfaccia informativa che è visivamente accattivante, ben organizzata e facile da navigare. Il box e la tabella sono progettati per garantire che le informazioni siano presentate in modo chiaro e leggibile.

4.1.4 /ProfileController.java

Il file è il controller dedicato basato su Spring Framework e si occupa di gestire la visualizzazione della pagina del profilo utente.

Inizialmente abbiamo tutte le importazioni necessarie. Poi, abbiamo la classe annotata con `@Controller`, che indica a Spring che questa classe gestisce richieste HTTP e invia risposte sotto forma di pagine web.

```
12  @Controller
13  public class ProfileController {
```

Qui viene utilizzato `RestTemplate`, iniettato attraverso il costruttore, per effettuare chiamate a servizi esterni (ad esempio per la validazione del token JWT).

```
15  private RestTemplate restTemplate;
16
17  @Autowired
18  public ProfileController(RestTemplate restTemplate) {
19      this.restTemplate = restTemplate;
20  }
```

Abbiamo, poi, il metodo `GetMapping`, un'annotazione di Spring che mappa le richieste HTTP di tipo `GET` a un determinato metodo. In questo caso, mappa le richieste GET all'`URL /profile`. Ogni volta che un utente accede alla URL `"/profile"`, Spring instrada quella richiesta a questo metodo specifico.

```
22  @GetMapping("/profile")
23  public String showProfilePage(Model model, @CookieValue(name = "jwt", required = false) String jwt) {
```

Tramite **@Cookie Value** estraiamo il valore di un cookie dalla richiesta HTTP. Il parametro *required = false* indica che il cookie può essere facoltativo. Se non è presente, il valore di *jwt* sarà "null".

Con *System.out.println("GET /profile, accesso alla pagina del profilo");* stampiamo nel terminale o log della console del server ogni volta che viene effettuata una richiesta GET.

MultiValueMap è una struttura dati che contiene una chiave associata a più valori. In questo caso, serve per raccogliere i dati da inviare all'endpoint esterno.

```
28 |     MultiValueMap<String, String> formData = new LinkedMultiValueMap<>();  
29 |     formData.add(key:"jwt", jwt);
```

Viene creata la mappa *formData* dove il token JWT è aggiunto come coppia chiave-valore con *jwt* come chiave e il valore del cookie JWT come valore. Questa mappa viene utilizzata per inviare dati al servizio di validazione del token

Per la chiamata al servizio esterno per la validazione del Token JWT utilizziamo **RestTemplate**, che effettua una chiamata HTTP POST all'endpoint di validazione.

```
32     Boolean isAuthenticated = restTemplate.postForObject(url:"http://t23-g1-app-1:8080/validateToken", formData, responseType:Boolean.class);
```

Il metodo *postForObject()* invia la richiesta e attende una risposta di tipo booleano che indica se l'utente è autenticato (true) o meno (false).

Dopo la chiamata al servizio esterno, il valore di *isAuthenticated* viene controllato:

- Se il risultato è "null" o "false", l'utente viene reindirizzato alla pagina di login ("redirect:/login").

```
35      if (isAuthenticated == null || !isAuthenticated) {  
36          return "redirect:/login";  
37      }
```

- Se il token JWT è valido, il metodo restituisce la stringa "*profile*". In Spring, restituire una stringa come questa significa che il framework cercherà una pagina HTML chiamata "profile".

```
40      return "profile";
```

Riassunto del Flusso:

1. Un utente visita l'URL */profile*.
2. Il token JWT viene recuperato dal cookie (se esiste).
3. Il token viene inviato a un servizio esterno per la validazione.
4. Se il token è valido, l'utente viene mostrato la pagina del profilo.
5. Se il token non è valido, l'utente viene reindirizzato alla pagina di login.

4.2 Implementazione Task 2-3

In questa sezione descriveremo le implementazioni relative al codice presente nei task T2-3.

4.2.1 /controller.java

Il file, come *ProfileController.java*, gestisce varie richieste HTTP come l'autenticazione e la gestione dell'utente. Ecco una lista delle funzionalità principali:

- Registrazione utente.
- Login e autenticazione.
- Reset della password
- Visualizzazione della lista degli studen

In particolare noi siamo intervenuti sulla funzione *generateToken()* che si occupa di creare un token JWT per un utente specifico. Prende in input un *oggetto User* e utilizza la libreria *Jwts* per costruire un token firmato. Il token contiene vari claim⁸, come l'email dell'utente (che diventa il subject del token) e altre informazioni personalizzate (come ID, nome, cognome, o ruolo). Il token ha una durata di un'ora e viene firmato con un algoritmo HMAC-SHA256⁹ utilizzando una chiave segreta ("mySecretKey").

```

422 |     public static String generateToken(User user) {
423 |         Instant now = Instant.now();
424 |         Instant expiration = now.plus(1, ChronoUnit.HOURS);
425 |
426 |         String token = Jwts.builder()
427 |             .setSubject(user.getEmail())
428 |             .setIssuedAt(Date.from(now))
429 |             .setExpiration(Date.from(expiration))
430 |             .claim("userId", user.getId())
431 |             .claim("nome", user.getName())
432 |             .claim("cognome", user.getSurName())
433 |             .signWith(SignatureAlgorithm.HS256, "mySecretKey") // Chiave segreta per firmare il token
434 |             .compact();
435 |
436 |         return token;
437     }

```

Figure 4.21: Metodo generateToken() modificato

Nella prima versione, il token conteneva i seguenti claim: *userId* e *role*. Con le nostre modifiche abbiamo aggiunto: *nome* e *cognome*.

⁸Un claim è una dichiarazione contenuta all'interno di un token JWT (JSON Web Token) che trasporta informazioni sull'utente o sul contesto dell'autenticazione. Sono utilizzati per trasmettere dati rilevanti tra il client e il server.

⁹L'algoritmo HMAC-SHA256 è un meccanismo di autenticazione dei messaggi (HMAC, Hash-based Message Authentication Code) che utilizza l'algoritmo di hash SHA-256 per garantire l'integrità e l'autenticità di un messaggio o di un dato.

4.3 Implementazione UI Gateway

In questa sezione descriveremo le implementazioni relative al codice presente nella componente UI Gateway. È una componente cruciale in un'architettura a microservizi, poiché fornisce un punto di accesso centralizzato per l'utente e un sistema di gestione delle richieste che indirizza verso i microservizi o le API appropriati.

4.3.1 /default.conf

Questo è un file di configurazione *NGINX*, il cui ruolo principale è quello di gestire l'instradamento delle richieste HTTP verso le componenti backend. La UI Gateway si posiziona tra l'utente finale (frontend) e i vari servizi backend.

La principale funzione del file è agire da **proxy inverso**, inoltrando le richieste dell'utente verso i microservizi backend appropriati in base agli URL specifici. Ad esempio, quando un utente accede a pagine come "/login" o "/register", NGINX inoltra la richiesta al backend appropriato (come *t23-g1-app-1:8080* o *mansclass-controller-1:8080*).

Siamo intervenuti sulla configurazione del file NGINX per gestire il traffico verso i nuovi servizi introdotti nella nostra applicazione web. Abbiamo aggiunto l'instradamento dell'endpoint "/profile" al servizio del task T5.

```
48 |     location ~ ^/(gamemode_scalata|main|gamemode|editor|editorAllenamento|report|t5|profile) {  
49 |         include /etc/nginx/includes/proxy.conf;  
50 |         proxy_pass http://t5-app-1:8080;  
51 |     }
```

Figure 4.22: Modifica endpoint

Questa integrazione con il servizio esistente migliora l'efficienza del sistema e garantisce una maggiore coerenza.

Chapter 5

Requisito R6 - Campagna testing di concorrenza

Il Testing è una fase cruciale nello sviluppo software in quanto permette di identificare eventuali problemi che possono verificarsi.

L’obiettivo di questo requisito è sottoporre il sistema a un set test di concorrenza per verificare il comportamento dell’applicazione quando viene utilizzata simultaneamente da più giocatori. L’idea è replicare eventuali errori legati all’accesso concorrente al filesystem condiviso sul Volume *T8* durante *la fase di gioco*.

Questo processo richiede la progettazione di scenari di test in cui più giocatori eseguono sfide contro i Robot, verificando la corretta gestione delle risorse condivise.

Il Testing è stato condotto manualmente da più user; questo requisito è una componente essenziale per identificare problemi di affidabilità e robustezza del sistema.

Per ogni scenario di test vengono definite **pre-condizioni, sequenze di**

eventi, output attesi, post-condizioni attese, con un'attenta analisi dei log restituiti. Gli scenari includono test sia in sequenza che in concorrenza.

Per l'esecuzione dei test di concorrenza, si utilizza sempre Ngrok, che consente di esporre in modo sicuro l'applicazione locale a internet, simulando così l'accesso simultaneo da parte di più giocatori reali. Grazie a Ngrok, è possibile creare più sessioni parallele da diverse origini, riproducendo scenari in cui utenti distribuiti geograficamente possono interagire con il sistema nello stesso momento.

Questo consente di osservare in tempo reale il comportamento del sistema sotto stress e di identificare eventuali colli di bottiglia o problematiche legate all'accesso concorrente al filesystem condiviso sul Volume T8.

5.1 Configurazioni iniziali

Descrizione delle condizioni iniziali del sistema in cui è stato eseguito il testing.

- 10 utenti loggati di cui: **9** come *User* e **1** come *Admin*.
- Due classi caricate dall'admin: *Calcolatrice.java* e *VCardBean.java*.

Ogni test varia in base alla classe e al robot utilizzato dai vari user. Ogni utente caricherà il codice per la propria classe di test (noi abbiamo utilizzato quelli presenti nella directory /ClassiUT/Tests) prima di ogni partita. L'admin visualizzerà la pagine *Players* dalla propria pagina principale, che contiene informazioni dinamiche sulle statistiche degli utenti autenticati in quel momento.

L'applicazione è in esecuzione locale sulla macchina dell'admin, e gli utenti vi accederanno da macchine diverse tramite il dominio statico condiviso di

Ngrok.

Gli user sono tutte persone differenti reclutate per l'occorrenza tra amici e colleghi.

5.2 Testing

La tabella è una struttura tipica utilizzata nella documentazione di test case per verificare i requisiti di un sistema. Di seguito la descrizione dei parametri di testing presenti nelle colonne:

- **Test Case ID** - Questo campo contiene l'identificatore univoco del test case.
- **Descrizione** - Questo campo contiene una breve descrizione del test case. È generalmente un riepilogo chiaro e conciso del comportamento che si vuole validare.
- **Classi di Equivalenza Coperte** - Questo campo elenca le classi di equivalenza che vengono coperte dal test. Le classi di equivalenza sono insiemi di input considerati equivalenti ai fini del test, e permettono di ridurre il numero di test necessari, coprendo più situazioni con un singolo test case.
- **Elemento di Sistema Provider** - Indica l'origine dei dati o delle risorse necessarie per eseguire il test, come un servizio di terze parti, un sistema esterno o un database interno. In questo contesto abbiamo sempre usato Ngrok.
- **Pre-condizioni** - Descrive lo stato o i requisiti necessari prima di eseguire il test. Le pre-condizioni possono includere stato del sistema, autenticazione dell'utente, configurazioni specifiche, o dati necessari.
- **Input** - Specifica i dati o file di input utilizzati per il test. Può includere dati di input, comandi o azioni che l'utente o il sistema devono compiere per eseguire il test.
- **Output Attesi** - Definisce i risultati attesi in seguito all'esecuzione del test. Si tratta dei comportamenti o degli stati che il sistema deve mostrare se il test ha esito positivo, basato sui requisiti. Può includere messaggi di successo, aggiornamenti dello stato, output di dati, ecc.

- **Post-condizioni Attese** - Descrive lo stato in cui il sistema dovrebbe trovarsi dopo l'esecuzione del test. Elenca le condizioni che devono essere soddisfatte alla fine del test.
- **PASS/FAIL** - Questa colonna registra l'esito del test case. "PASS" indica che il sistema ha soddisfatto tutte le aspettative definite nell'output atteso e nelle post-condizioni. "FAIL" indica che il sistema non ha soddisfatto uno o più criteri attesi, e il test ha riscontrato un errore o un'anomalia.

5.2.1 Tabella dei Test case

Test Case ID	Descrizione	Classi di equivalenza	Elemento di Sistema provider	Pre-condizioni	Input	Output attesi	Post-condizioni attese	PASS/FAIL
1	9 user contemporanei e 1 admin. Codice di test vincente per tutti i giocatori, stessa classe, stesso robot,	Vittoria contro i robot per tutti i giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, hanno scelto la classe Calcolatrice e Randoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java } Utente5:{Codice: TestCalcolatrice.java } Utente6:{Codice: TestCalcolatrice.java } Utente7:{Codice: TestCalcolatrice.java } Utente8:{Codice: TestCalcolatrice.java } Utente9:{Codice: TestCalcolatrice.java } Admin:{visualizzazione Players}	Tutti vincitori	Messaggio di vittoria, con esito dei risultati per tutti i giocatori	PASS
2	9 user contemporanei e 1 admin. Codice di test perdente per tutti i giocatori, stessa classe, stesso robot,	Sconfitta contro i robot per tutti i giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, hanno scelto la classe Calcolatrice e Randoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice_1.java } Utente2:{Codice: TestCalcolatrice_1.java } Utente3:{Codice: TestCalcolatrice_1.java } Utente4:{Codice: TestCalcolatrice_1.java } Utente5:{Codice: TestCalcolatrice_1.java } Utente6:{Codice: TestCalcolatrice_1.java } Utente7:{Codice: TestCalcolatrice_1.java } Utente8:{Codice: TestCalcolatrice_1.java } Utente9:{Codice: TestCalcolatrice_1.java } Admin:{visualizzazione Players}	Tutti perdenti	Messaggio di sconfitta per tutti i giocatori	PASS

CHAPTER 5. REQUISITO R6 - CAMPAGNA TESTING DI CONCORRENZA

3	9 user contemporanei e 1 admin. Codice di test vincente per 4 giocatori e perdente per 5 giocatori, stessa classe, ma robot misti.	Vittoria di solo 4 giocatori contro i robot.	Ngrok	Gli utenti sono regolarmente loggati nel sistema, hanno scelto la classe Calcolatrice e, 4 utenti Radoop e 5 utenti EvoSuite, come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestCalcolatrice.java }; Utente6:{Codice: TestCalcolatrice.java }; Utente7:{Codice: TestCalcolatrice.java }; Utente8:{Codice: TestCalcolatrice.java }; Utente9:{Codice: TestCalcolatrice.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3, Utente4, vincitori. Utente5, Utente6, Utente7, Utente8, Utente9.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4. Messaggio di sconfitta per Utente5, Utente6, Utente7, Utente8, Utente9.	PASS
4	9 user contemporanei e 1 admin. Codice di test vincente per tutti i giocatori, classi miste, stesso robot.	Vittoria contro i robot per tutti i giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 4 hanno scelto la classe Calcolatrice, 5 hanno scelto VCardBean e tutti Radoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestVCardBean2.java }; Utente6:{Codice: TestVCardBean2.java }; Utente7:{Codice: TestVCardBean2.java }; Utente8:{Codice: TestVCardBean2.java }; Utente9:{Codice: TestVCardBean2.java }; Admin:{visualizzazione Players}	Tutti vincitori	Messaggio di vittoria, con esito dei risultati per tutti i giocatori	PASS
5	9 user contemporanei e 1 admin. Codice di test vincente per 5 giocatori e perdente per 4 giocatori, classi miste, stesso robot.	Vittoria contro i robot di soli 5 giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 5 hanno scelto la classe Calcolatrice, 4 VCardBean e tutti Radoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestVCardBean.java }; Utente6:{Codice: TestVCardBean.java }; Utente7:{Codice: TestVCardBean.java }; Utente8:{Codice: TestVCardBean.java }; Utente9:{Codice: TestVCardBean.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3, Utente4 e Utente5 vincitori. Utente6, Utente7, Utente8, Utente9 perdenti.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4, Utente5. Messaggio di sconfitta per Utente6, Utente7, Utente8, Utente9.	PASS
6	9 user contemporanei e 1 admin. Codice di test vincente per 4 giocatori e perdente per 5 giocatori, classi miste, robot misti,	Vittoria contro i robot di soli 4 giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 4 hanno scelto la classe VCardBean e Radoop come robot da sfidare. E 5 hanno scelto la classe Calcolatrice e EvoSuite come robot. Un amministratore è regolarmente loggato al sistema.	Utente1: Codice: TestVCardBean3.java ; Utente2:{Codice: TestVCardBean3.java }; Utente3:{Codice: TestVCardBean3.java }; Utente4:{Codice: TestVCardBean3.java }; Utente5:{Codice: TestCalcolatrice.java }; Utente6:{Codice: TestCalcolatrice.java }; Utente7:{Codice: TestCalcolatrice.java }; Utente8:{Codice: TestCalcolatrice.java }; Utente9:{Codice: TestCalcolatrice.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3 e Utente4 vincitori. Utente5, Utente6, Utente7, Utente8, Utente9 perdenti.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4. Messaggio di sconfitta per Utente5, Utente6, Utente7, Utente8, Utente9.	PASS

CHAPTER 5. REQUISITO R6 - CAMPAGNA TESTING DI CONCORRENZA

3	9 user contemporanei e 1 admin. Codice di test vincente per 4 giocatori e perdente per 5 giocatori, stessa classe, ma robot misti.	Vittoria di solo 4 giocatori contro i robot.	Ngrok	Gli utenti sono regolarmente loggati nel sistema, hanno scelto la classe Calcolatrice e, 4 utenti Radoop e 5 utenti EvoSuite, come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestCalcolatrice.java }; Utente6:{Codice: TestCalcolatrice.java }; Utente7:{Codice: TestCalcolatrice.java }; Utente8:{Codice: TestCalcolatrice.java }; Utente9:{Codice: TestCalcolatrice.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3, Utente4, Utente5, Utente6, Utente7, Utente8, Utente9.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4. Messaggio di sconfitta per Utente5, Utente6, Utente7, Utente8, Utente9.	PASS
4	9 user contemporanei e 1 admin. Codice di test vincente per tutti i giocatori, classi miste, stesso robot.	Vittoria contro i robot per tutti i giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 4 hanno scelto la classe Calcolatrice, 5 hanno scelto VCardBean e tutti Radoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestVCardBean2.java }; Utente6:{Codice: TestVCardBean2.java }; Utente7:{Codice: TestVCardBean2.java }; Utente8:{Codice: TestVCardBean2.java }; Utente9:{Codice: TestVCardBean2.java }; Admin:{visualizzazione Players}	Tutti vincitori	Messaggio di vittoria, con esito dei risultati per tutti i giocatori	PASS
5	9 user contemporanei e 1 admin. Codice di test vincente per 5 giocatori e perdente per 4 giocatori, classi miste, stesso robot.	Vittoria contro i robot di soli 5 giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 5 hanno scelto la classe Calcolatrice, 4 VCardBean e tutti Radoop come robot da sfidare. Un amministratore è regolarmente loggato al sistema.	Utente1:{Codice: TestCalcolatrice.java }; Utente2:{Codice: TestCalcolatrice.java }; Utente3:{Codice: TestCalcolatrice.java }; Utente4:{Codice: TestCalcolatrice.java }; Utente5:{Codice: TestVCardBean.java }; Utente6:{Codice: TestVCardBean.java }; Utente7:{Codice: TestVCardBean.java }; Utente8:{Codice: TestVCardBean.java }; Utente9:{Codice: TestVCardBean.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3, Utente4 e Utente5 vincitori. Utente6, Utente7, Utente8, Utente9 perdenti.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4, Utente5. Messaggio di sconfitta per Utente6, Utente7, Utente8, Utente9.	PASS
6	9 user contemporanei e 1 admin. Codice di test vincente per 4 giocatori e perdente per 5 giocatori, classi miste, robot misti,	Vittoria contro i robot di soli 4 giocatori	Ngrok	Gli utenti sono regolarmente loggati nel sistema, 4 hanno scelto la classe VCardBean e Radoop come robot da sfidare. E 5 hanno scelto la classe Calcolatrice e EvoSuite come robot. Un amministratore è regolarmente loggato al sistema.	Utente1: Codice: TestVCardBean3.java ; Utente2:{Codice: TestVCardBean3.java }; Utente3:{Codice: TestVCardBean3.java }; Utente4:{Codice: TestVCardBean3.java }; Utente5:{Codice: TestCalcolatrice.java }; Utente6:{Codice: TestCalcolatrice.java }; Utente7:{Codice: TestCalcolatrice.java }; Utente8:{Codice: TestCalcolatrice.java }; Utente9:{Codice: TestCalcolatrice.java }; Admin:{visualizzazione Players}	Utente1, Utente2, Utente3 e Utente4 vincitori. Utente5, Utente6, Utente7, Utente8, Utente9 perdenti.	Messaggio di vittoria con esito dei risultati per Utente1, Utente2, Utente3, Utente4. Messaggio di sconfitta per Utente5, Utente6, Utente7, Utente8, Utente9.	PASS

In conclusione, i test case condotti sono stati fondamentali per validare il comportamento del sistema in uno scenario di utilizzo concorrente, in linea con l'obiettivo principale del test: verificare la gestione delle risorse condivise, in particolare il filesystem sul Volume T8, durante l'accesso simultaneo da parte di più giocatori.

I risultati del test hanno evidenziato che il sistema è stato in grado di gestire correttamente la concorrenza, mantenendo l'integrità delle risorse e garantendo una gestione efficiente delle richieste multiple. Abbiamo osservato che la capacità del sistema di sincronizzare gli accessi ai dati condivisi ha dimostrato un buon livello di **robustezza** e **affidabilità**.

In definitiva, questo test ha raggiunto il suo scopo di valutare la stabilità del sistema sotto carico, evidenziando i suoi punti di forza.

Chapter 6

Guida all'installazione e all'utilizzo

Di seguito è riportata la procedura di installazione e utilizzo dei tool necessari e dell'applicazione, in ambiente Windows.

6.1 Installazione

6.1.1 Docker e applicazione

Passaggi da seguire:

1. Scaricare Docker Desktop dal sito ufficiale ed eseguire l'installazione.
2. Una volta scaricata la cartella del progetto da GitHub, eseguire lo script “installer.bat” su Windows. Al termine si avrà:
 - la creazione di una rete global comune a tutti i container;
 - la creazione e installazione dei singoli container nell'applicazione Docker Desktop.
 - avvio dei container.

Nota bene:

- L'**UI-Gateway** deve essere l'ultimo container da avviare!
- Il container **progetto-sad-g19-master** è normale che risulti non in esecuzione!

uninstaller.bat

Nel caso non sia la prima installazione, è necessario effettuare prima la disin-stallazione utilizzando “*uninstaller.bat*” mentre si ha in esecuzione Docker: in questo modo si elimina qualunque file presente su Docker.

6.1.2 Ngrok

Seguire i passi:

1. Registrarsi sul sito ufficiale.
2. Accedere alla dashboard scegliendo come agente Docker o Windows.
3. Mentre Docker è in funzione, eseguire i comandi nel *Prompt comandi* di Windows. Si consiglia l'uso di un *dominio statico*. L'utilizzo di un dominio statico è consigliato per evitare la creazione di un nuovo container ad ogni avvio.

```
docker pull ngrok/ngrok  
docker run -net=host -it -e NGROK_AUTH_TOKEN=*(Inserire  
token)* ngrok/ngrok:latest http 80
```

Dopo l'esecuzione dei comandi, verrà installato un nuovo container all'interno di Docker. Nel prompt comparirà una schermata che specifica il dominio al quale accedere.

Nel caso il container sia già installato, è possibile accedere direttamente (una volta avviati i container) alla web application tramite il dominio presente nella sezione laterale "Endpoints" sul sito di Ngrok.

6.2 Utilizzo

Per l'utilizzo, è necessario avviare tutti i container ad eccezione di ui-gateway, da avviare per ultimo.

6.2.1 Admin

Registrazione admin

L'amministratore, se è il primo accesso all'applicazione, si deve registrare nella schermata apposita. Dalla pagine *Home*:

1. Admin
2. Registrati
3. Compila il form nel modo giusto

Come nome utente utilizza la forma *USER_unina*.

Caricamento classe di test

Prima di iniziare a giocare è necessario inserire una classe di test da sottoporre a verifica e ciò può essere fatto solo dall'amministratore. Una volta eseguito il login, seguire i seguenti passi:

1. Classes
2. + Add Class
3. Compilare il modulo con le giuste informazioni
4. Cliccare su *Scegli il file* e scegliere la classe *.java* da caricare

6.2.2 User

Registrazione utente

L'utente, se è il primo accesso all'applicazione, si deve registrare nella schermata apposita. Dalla pagina *Home*:

1. User
2. Registrati
3. Compila il form nel modo giusto

Giocare una partita

Dalla schermata principale dell'utente seguire i passi:

1. Sfida un robot
2. Scegli una classe e un robot
3. Clicca su *Submit* nelle due schermate successive
4. Clicca su *Gioca* nell'editor per avviare una partita
5. Visualizza il risultato e naviga nella barra per le altre opzioni

6.2.3 Sviluppatore Backend

Se sei uni sviluppatore che vuole dare il proprio contributo all'applicazione, questa guida ti illustra i passaggi principali da seguire per la modifica del codice.

6.2.4 Modifica il codice

Apri la directory del progetto in Visual Studio Code e modifica i file necessario nei rispettivi task.

Aggiorna e visualizza il codice modificato

Una volta terminate le modifiche, per visualizzare segui i passi:

1. Recarsi nell'applicazione Docker Desktop;
2. Aprire la sezione *containers*;
3. Effettuare la *delete* dei containers modificati;
4. Aprire la sezione *images*;
5. Effettuare la *delete* le images relative ai container modificati;
6. Recarsi sull'IDE utilizzato (VS Code nel nostro caso)
7. Aprire il terminale integrato nella cartella relativa al task modificato;
8. Eseguire nel terminale comando "*mvn clean package*";
9. Nelle cartelle dei container modificati, cerca il file "docker-compose.yml", tasto destro e clicca sull'opzione *Compose Up* (necessaria l'estensione Docker installata nell'IDE)

Atteso il riavvio dei container, collegarsi al dominio Ngrok per visualizzare le modifiche.

6.2.5 Tips & Tricks

Qui alcuni suggerimenti per utenti e sviluppatori futuri.

Cache del browser

Disattivare la cache del browser permette di assicurarsi che ogni modifica al codice sorgente o alle risorse dell'applicazione venga visualizzata immediatamente. La cache, infatti, può causare problemi di compatibilità, e disabilitarla rende più semplice il processo di debugging. In questo modo, gli sviluppatori possono individuare e correggere eventuali bug più rapidamente, senza doversi preoccupare della persistenza dei file nella cache che potrebbe nascondere il problema.

Errore Bad Gateway

A volte, anche se tutti i container sono stati avviati correttamente, la pagina web potrebbe restituire un errore del tipo *502 Bad Gateway*. Questo problema potrebbe essere dovuto al tentativo di accedere all'applicazione web troppo rapidamente, subito dopo il riavvio dei container. In tali situazioni, la piattaforma potrebbe avere bisogno di più tempo per completare l'inizializzazione e stabilire correttamente le connessioni tra i vari componenti. Spesso, attendere qualche istante prima di provare di nuovo può risolvere il problema. Tuttavia, se l'errore persiste nonostante l'attesa, è consigliabile riavviare o avviare il container associato al componente **UI Gateway**.

Avvio manuale dei container "app-1" per T4 e T23

A volte, dopo l'avvio dell'applicazione, i container app-1 dei task **T4** e **T23** potrebbero risultare in stato "Exited". In tal caso, è necessario avviarli manualmente.

<input type="checkbox"/>	 t23-g1	Running (1/2)	N/A	2 minutes ago	<input type="checkbox"/>			
<input type="checkbox"/>	 app-1 eb3badfa3d1b	t23-g1-app Exited (143)	N/A	2 minutes ago				
<input type="checkbox"/>	 db-1 58268ee0fc6f	mysql:latest Running	3306:3306	N/A	2 minutes ago	<input type="checkbox"/>		

CHAPTER 6. GUIDA ALL'INSTALLAZIONE E ALL'UTILIZZO

<input type="checkbox"/>	t4-g18	Running (3/4)	N/A	1 minute ago	<input type="checkbox"/>		
<input type="checkbox"/>	app-1 081df8afa4f7 ⚡	Exited (1) 3000:3000	N/A	1 minute ago			
<input type="checkbox"/>	prometheus-1 a2cb1c1841ec ⚡	Running 3330:9090 ⚡	N/A	1 minute ago	<input type="checkbox"/>		
<input type="checkbox"/>	grafana-1 18e7af36ac66 ⚡	Running 3300:3000 ⚡	N/A	1 minute ago	<input type="checkbox"/>		
<input type="checkbox"/>	db-1 097f3563f6d5 ⚡	Running 5432:5432 ⚡	N/A	1 minute ago	<input type="checkbox"/>		