

Università degli Studi di Napoli Federico II
Corso di Laurea Magistrale in Ingegneria Informatica
Corso di Software Architecture Design
Prof. A.R. Fasolino - A.A. 2022 - 23

Task 5

Requisiti sull'avvio del Primo Scenario di Gioco

- Angelo Cristiano M63001424
angelo.cristiano@studenti.unina.it
- Stefano Bruno M63001430
stefano.bruno2@studenti.unina.it
- Domenico Palladino M63001466
domen.palladino@studenti.unina.it

Indice

Capitolo 1: Avvio del progetto	3
Strumenti utilizzati dal team di sviluppo	4
Processo di sviluppo	5
Unified process	6
SCRUM	7
Capitolo 2: Specifica dei requisiti	8
Modellazione dei casi d'uso	9
Analisi dei casi d'uso: Diagrammi UML	13
System Domain Model	23
Capitolo 3: Architettura e progettazione	24
Client-Server Pattern	24
Pattern MVC	25
Vista Componenti e Connettori	29
Diagrammi di sequenza di progetto	31
Documentazione API	35
Capitolo 4: Stima dei costi	39
Capitolo 5: Deployment e tecnologie	44
Tecnologie utilizzate	44
Manuale di Installazione	50

Capitolo 1: Avvio del progetto

Si vuole realizzare una applicazione per consentire a studenti di Software Testing di fare addestramento su Task di Unit Testing.

- I task consistono nello sviluppo di casi di test di unità per Classi Java che devono raggiungere uno specifico obiettivo. I possibili obiettivi sono la copertura delle LOC, dei Branch, delle Decisioni, delle Eccezioni, di WeakMutation, etc. I casi di test sono da implementare in JUnit4.
- Per motivare e stimolare gli studenti, si vuole realizzare un approccio basato su Gamification, nel quale gli studenti potranno competere contro diversi tipi di strumenti di generazione automatica di casi di test (anche detti Robot), quali ad esempio Randoop[1] o Evosuite[2].

Task 5:

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

Parti interessate:

Gli utilizzatori del servizio si dividono in anzitutto in due tipologie principali di utenti: quelli registrati al servizio e quelli non registrati. Una volta registrati gli utenti possono inviare e ricevere messaggi.

Utenti:

Giocatore non registrato

Gli utenti non registrati non hanno accesso ai servizi del gioco. Al momento dell'accesso all'applicazione sarà data loro la possibilità di effettuare la registrazione. Al momento della registrazione, l'utente specifica un'e-mail ed una password che sarà necessaria per accedere in modo sicuro all'applicazione. Effettuata la registrazione potrà effettuare il login al servizio.

Giocatore registrato

Gli utenti registrati hanno accesso alle funzionalità del sistema.

A seguito di una fase di login questi saranno reindirizzati alla schermata iniziale dell'applicazione, dalla quale potranno visualizzare la schermata per la configurazione di una nuova partita.

Attore	Obiettivo
Giocatore non registrato	Effettuare la registrazione
Giocatore registrato	Effettuare il login
Giocatore autenticato	Avviare una nuova partita

Strumenti utilizzati dal team di sviluppo

Microsoft Teams

Microsoft Teams è un'applicazione di comunicazione e collaborazione che consente agli utenti di chattare, fare chiamate audio e video, condividere file e collaborare su documenti in tempo reale. È particolarmente utile per il lavoro remoto e il telelavoro, poiché consente agli utenti di connettersi e comunicare con i colleghi ovunque si trovino. Inoltre, Teams integra diverse app e servizi di Microsoft, come Office 365, OneDrive e SharePoint, rendendo più semplice l'accesso e la condivisione di file e dati.

Github

GitHub è una piattaforma di hosting per il controllo delle versioni dei progetti software. Consente agli utenti di collaborare su progetti di sviluppo software, tenere traccia delle modifiche apportate ai file e gestire il flusso di lavoro di sviluppo da un'unica interfaccia utente. Gli utenti possono creare repository, caricare il codice sorgente, gestire problemi e richieste di pull, eseguire test e rilasciare il software. Inoltre, GitHub supporta una vasta gamma di integrazioni e strumenti di automazione che rendono il lavoro degli sviluppatori più efficiente.

Trello

Trello è un'applicazione di gestione dei progetti che consente agli utenti di organizzare le attività in schede, liste e bacheche. Ogni scheda rappresenta una specifica attività del progetto, mentre le liste rappresentano le fasi del lavoro. Le bacheche sono invece i contenitori delle schede e delle liste e possono essere utilizzate per organizzare e visualizzare i progetti. Trello è altamente personalizzabile, consentendo agli utenti di creare etichette, impostare scadenze, assegnare compiti e commentare le schede.

Visual Paradigm

Visual Paradigm è un software di modellazione UML che consente agli utenti di creare diagrammi di flusso, diagrammi di classe, diagrammi di sequenza e molti altri. È particolarmente utile per gli sviluppatori software e gli analisti di business, poiché consente loro di visualizzare e comprendere meglio i processi aziendali e il flusso di lavoro. Visual Paradigm è altamente personalizzabile e fornisce una vasta gamma di funzionalità avanzate, come la generazione automatica di codice, l'integrazione con altri strumenti di sviluppo e la collaborazione in tempo reale.

Processo di sviluppo

Un Processo Software è un insieme strutturato di attività necessarie per lo sviluppo di un sistema software (specifica, progettazione, sviluppo, validazione, evoluzione dopo il rilascio...).

Al fine di poter realizzare il progetto, è stato necessario utilizzare strumenti e pratiche ad hoc per organizzare al meglio il lavoro. Per questo motivo si mostrano in questo capitolo tutte le tecnologie ed i tool utilizzati durante lo sviluppo del progetto e le pratiche agili adoperate per migliorare lo sviluppo dell'applicazione. In particolare, si è deciso di adottare un processo di sviluppo di tipo Agile (e quindi non plan-based/guidato dai piani) per cui la pianificazione è incrementale e risulta più semplice modificare il processo in modo tale da riflettere e adattarsi alle mutevoli esigenze del cliente. A supporto di questa metodologia, durante il processo di progettazione, sono stati realizzati dei prototipi ad-hoc allo scopo di realizzare al meglio i requisiti e per prendere familiarità con le tecnologie utilizzate.

Unified process

UP è un framework di processo di sviluppo software iterativo ed incrementale. Infatti, è una metodologia che prevede lo sviluppo del software come un'attività guidata dalla definizione dei requisiti funzionali, espressi attraverso i casi d'uso. L'analisi dei casi d'uso, di conseguenza, permette di definire le caratteristiche dell'architettura software che li realizza in modo integrato. Esso è basato sull'ampliamento e sul raffinamento di un sistema attraverso diverse iterazioni, con feedback e adattamenti ciclici. Il sistema è sviluppato in maniera incrementale col passare del tempo, iterazione per iterazione, e infatti questo approccio è anche conosciuto come sviluppo di software iterativo ed incrementale. Le iterazioni sono divise su quattro fasi ciascuna delle quali consiste in una o più iterazioni

• **Inception:** la prima e la fase più breve nel progetto. È utilizzata per preparare la base del progetto, che include: stabilire lo *scope* del progetto, definire i *vincoli*, creare la tabella *AttoriObiettivi*, delineare i *requisiti* chiave e le possibili soluzioni architetture insieme ai *compromessi* di progettazione. Una durata eccessivamente prolungata della fase di inception potrebbe essere sintomo di una mancata chiarezza, da parte degli stakeholders, della visione e degli obiettivi del progetto. Senza obiettivi e visione chiari, il progetto molto probabilmente è destinato a fallire. In questo scenario è meglio prendere una pausa all'inizio del progetto per raffinare visione e obiettivi. In caso contrario, ciò potrebbe portare a ritardi di organizzazione non benevoli per le fasi successive.

• **Elaborazione:** durante questa fase, il team deve elencare la maggior parte dei requisiti di sistema (per esempio, nella forma di use case), eseguire una analisi dei rischi identificati e definire un piano di risk management per ridurre o eliminarne l'impatto sulla schedule finale e sul prodotto, stabilire la progettazione e l'architettura (utilizzando class diagram di base, package diagram o deployment diagram), creare un piano (calendario, stime dei costi, ecc.) per la fase successiva (costruzione).

• **Costruzione:** la fase più lunga e più ampia di UP. Durante questa fase, la progettazione del sistema viene finalizzata e perfezionata e il sistema viene costruito utilizzando le basi create durante la fase di elaborazione. La fase di costruzione è suddivisa in più iterazioni, ognuna delle quali deve portare a un rilascio eseguibile del sistema. L'iterazione finale della fase di costruzione permette di ottenere il sistema completo, che deve essere distribuito durante

la fase di transizione.

• **Transizione:** fase finale del progetto che consegna il nuovo sistema agli utenti finali. La fase di transizione comprende anche la migrazione dei dati dai sistemi legacy e la formazione degli utenti. Chiaramente questo modello di sviluppo non è restrittivo né sulla realizzazione di queste fasi, né sul numero di iterazioni da compiere in ogni fase. UP è quindi aperto all'uso di pratiche agili e prevede l'uso di **VCS** (Version Control System) per mantenere sempre una versione funzionante del software. Il ciclo di sviluppo dell'applicazione sfrutta le idee di UP, in quanto la prima fase, che corrisponde a quella di Inception in UP, ha permesso di porre le basi del progetto, con la scrittura del Glossario e di una prima scrittura dei casi d'uso in formato breve. Da lì in poi lo sviluppo è sempre avvenuto in maniera incrementale e attraverso iterazioni, ma utilizzando un mix di tecniche riprese dal metodo di sviluppo software SCRUM .

SCRUM

Scrum è un framework agile per la gestione del ciclo di sviluppo del software, iterativo ed incrementale, concepito per la gestione dei progetti e dei prodotti software. Non è una tecnica o un processo, ma un framework all'interno del quale è possibile usare più processi e tecniche. Per quanto riguarda il progetto, Scrum rappresenta il framework da cui sono stati attinti più processi e tecniche. In primis, dopo la fase di Inception, sono stati sviluppati 2 cicli di Sprint, per cui in ogni ciclo viene realizzato un incremento del sistema.

Lo Sprint è un periodo di tempo limitato durante il quale viene creato un incremento di prodotto utilizzabile e potenzialmente rilasciabile. Gli Sprint hanno durata costante durante il progetto, nel nostro caso equivale a 14 giorni. Inoltre, alla chiusura di uno Sprint, si avvia immediatamente il successivo.

Un'altra tecnica di Scrum utilizzata nel ciclo di sviluppo è il Product Backlog, un elenco di voci per portare a termine il prodotto, ordinato per priorità, che viene aggiornato man mano che il lavoro avanza.

Quindi, all'inizio di ciascuno Sprint, il Team seleziona dal Backlog un insieme di voci da sviluppare in quella iterazione (lo Sprint Goal) e crea lo Sprint Backlog, ovvero i compiti da svolgere per arrivare all'obiettivo dello Sprint. Ciò è stato semplificato dall'utilizzo di una Task Board, offerta da Trello per tenere traccia del lavoro svolto. Inoltre, per avere una stima dell'effort per ogni attività, è stata utilizzata la feature di inserimento delle etichette di Trello.

Infine, tra uno Sprint ed il successivo sono state organizzate delle Sprint Retrospective.

Una SprintRetrospective è un meeting, in cui il team analizza le proprie attività e crea un piano di miglioramenti da attuare per il prossimo Sprint. In questo contesto, si cerca di aumentare la qualità del prodotto migliorando i processi di lavoro. In particolare, è stato discusso:

- Cosa è andato bene durante lo Sprint
- Cosa potrebbe essere migliorato
- Cosa ci si impegna a migliorare nel prossimo Sprint

Capitolo 2: Specifica dei requisiti

Requisiti:

1. L'utente deve poter avviare una partita.
2. L'utente deve poter visualizzare l'elenco delle classi da testare.
3. L'utente deve poter visualizzare l'elenco di robot disponibili.
4. L'utente deve poter selezionare la classe da testare.
5. L'utente deve poter selezionare il robot.
6. Il sistema deve associare alla partita creata un id.
7. Il sistema deve poter salvare la configurazione della partita.
8. Il sistema deve poter avviare un ambiente di editing.
9. Il giocatore deve poter interagire con un ambiente di editing per scrivere il test.
10. L'utente deve poter selezionare il numero di turni della partita
11. L'utente deve poter selezionare la modalità della partita.
12. L'utente deve poter invitare un altro utente
13. L'utente deve poter accettare o rifiutare un invito.

Glossario:

- **Giocatore autenticato:** l'utente che ha già effettuato l'autenticazione.
- **Autenticazione:** il processo di verifica dell'identità dell'utente tramite l'inserimento di credenziali di accesso valide.
- **Robot:** software per effettuare testing automatico, avversario del giocatore.
- **Partita:** una sessione di gioco in cui il giocatore può mettere alla prova le sue abilità di testing contro un Robot.
- **IdPartita:** un identificatore univoco associato a una partita.
- **Ambiente di editing:** interfaccia utente che permette al giocatore di poter visualizzare la classe e di scrivere la classe di test.
- **Classe di test:** una classe che contiene una serie di metodi di test per verificare la corretta implementazione di una classe.

Modellazione dei casi d'uso

Attori Primari:

- Giocatore autenticato

Scenari utente:

- Avvia nuova partita
- Avvia configurazione
- Scelta Modalità e turni
- Scelta classe
- Invita Amici
- Scelta robot
- Accetta/Declina invito

1) Avvia Configurazione

Attore Primario	Giocatore autenticato
Descrizione	Il giocatore autenticato avvia una nuova partita del primo scenario
Pre-condizioni	Il giocatore si è autenticato correttamente.
Flusso Principale	1. Il giocatore seleziona il riquadro conferma selezione
Post-condizione	Il sistema mostra la seconda schermata di configurazione

2) Scelta Modalità e turni

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore configura la partita e conferma le selezioni
Pre-condizioni	Il giocatore si è autenticato correttamente
Flusso principale	<ol style="list-style-type: none">1. Il giocatore visualizza i pulsanti relativi alla configurazione2. Il giocatore seleziona le sue preferenze3. Il giocatore preme sul pulsante "Avvia Partita"
Post-Condizioni	Il sistema salva le scelte effettuate e mostra la seconda schermata di configurazione

3) Scelta Classe

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore può consultare la lista delle classi da poter testare e effettuare una scelta.
Pre-condizioni	Il giocatore ha terminato la prima configurazione
Flusso principale	<ol style="list-style-type: none">1. Il giocatore visualizza l'elenco delle classi da poter testare.2. Il giocatore seleziona la classe da testare.
Post-Condizioni	Il sistema salva la classe scelta.

4) Scelta Robot

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore può consultare la lista dei robot con cui confrontarsi e effettuare una scelta.
Pre-condizioni	Il giocatore ha terminato la seconda configurazione.
Flusso principale	<ol style="list-style-type: none">1. Il giocatore visualizza l'elenco dei robot.2. Il giocatore seleziona un robot.
Post-Condizioni	Il sistema salva la scelta effettuata.

5) Invita Amici

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore può invitare un altro giocatore
Pre-condizioni	Il giocatore ha selezionato la modalità multigiocatore ed ha confermato le selezioni
Flusso principale	1. Il giocatore visualizza il form per invitare un giocatore 2. Il giocatore inserisce l'email dell'amico da invitare 3. Il giocatore preme il bottone invita
Post-Condizioni	Il sistema ha registrato correttamente l'invito

6) Avvia nuova partita

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore avvia una nuova partita
Pre-condizioni	Il giocatore deve aver selezionato la classe da testare e il robot contro cui confrontarsi
Flusso principale	1. Il giocatore visualizza il bottone avvia nuova partita 2. Il giocatore preme il bottone
Post-Condizioni	Il sistema salva tutte le scelte, avvia l'ambiente di editing e associa alla partita un id

7) Accetta/Declina invito

Attore primario	Giocatore Autenticato
Descrizione	Il giocatore accetta l'invito
Pre-condizioni	Il giocatore deve essersi autenticato correttamente
Flusso principale	1. Il giocatore visualizza la lista degli inviti ricevuti 2. (if invito accettato) Il giocatore preme il bottone di accetta invito 3.(else invito rifiutato) Il giocatore preme il bottone di rifiuta invito
Post-Condizioni	(ramo if) Il giocatore si unisce alla partita a cui è stato invitato e il sistema avvia una nuova partita con le configurazioni relative . (ramo else) Il sistema cancella l'invito.

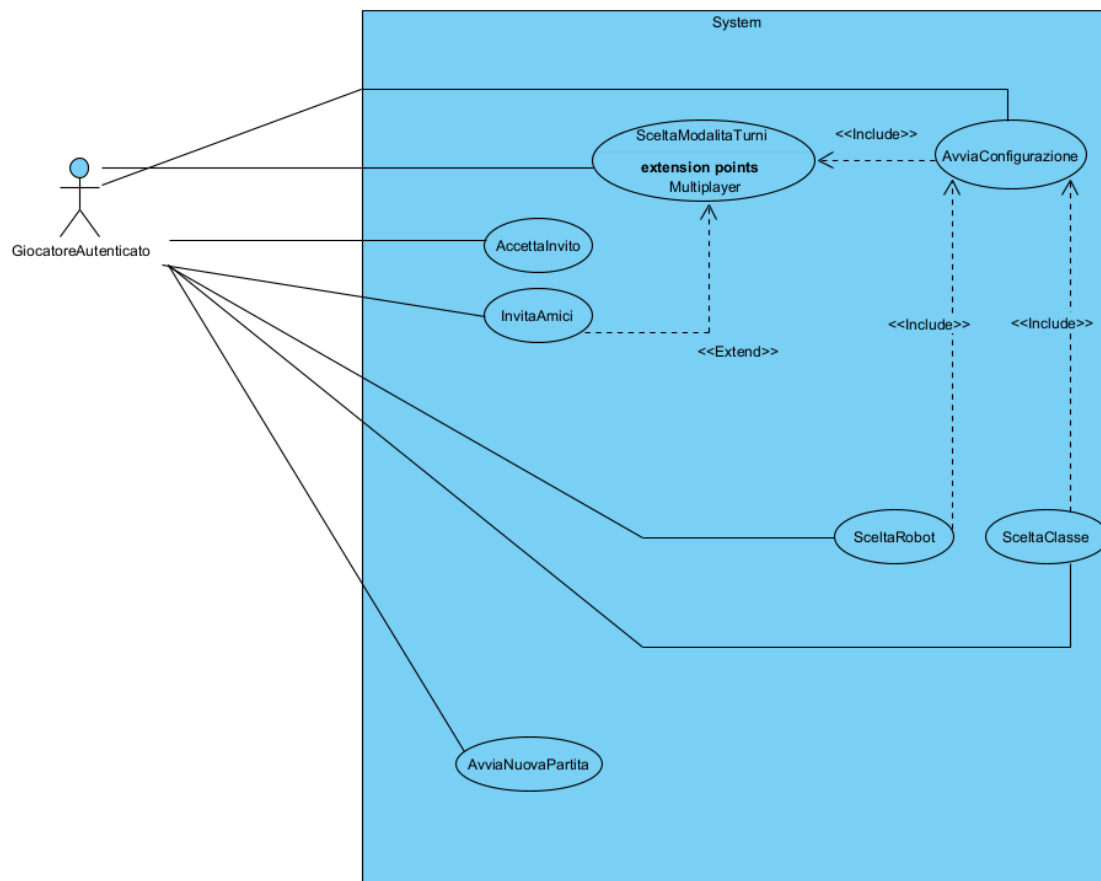
Product Backlog dell'ultima iterazione

#ID	Item	Categoria	Status	Effort
1	<i>Come Giocatore, voglio poter avviare la configurazione</i>	User Story	Completed	2
2	<i>Come Giocatore, voglio poter scegliere la classe da testare</i>	User Story	Completed	5
3	<i>Come Giocatore, voglio poter scegliere il robot da sfidare</i>	User Story	Completed	1
4	<i>Come Giocatore, voglio potermi registrare</i>	User Story	Completed	8
5	Creare prototipo interfaccia utente	Discovery	Completed	2
6	<i>Come Giocatore, voglio poter scegliere la modalità di gioco: Singolo Giocatore-Multigiocatore</i>	User Story	Completed	1
7	Realizzazione GUI in React	Discovery	Completed	8
8	Come Giocatore, voglio poter invitare altri giocatori	Epic	Completed	13
9	Come Giocatore, voglio poter scegliere il numero di turni	User Story	Completed	1
10	Sviluppo del backend e della logica di gestione dei dati	Discovery	Completed	21
11	Come Giocatore, voglio poter autenticarmi	User Story	Completed	13
12	Come Giocatore, voglio poter resettare la mia password	User Story	Completed	5
13	Deployment dell'app su docker	Discovery	Completed	5
14	Come Giocatore, voglio poter vedere il mio storico inviti ricevuti	User Story	Completed	5
15	Come Giocatore, voglio poter accettare o declinare un invito	User Story	Completed	21

Analisi dei casi d'uso: Diagrammi UML

Diagramma dei casi d'uso

Al fine di sintetizzare graficamente i casi d'uso definiti in fase di specifica dei requisiti, è stato realizzato un diagramma dei casi d'uso. Esso mette in luce le funzionalità offerte dal sistema, permettendo al contempo di visualizzare i diversi tipi di attori del sistema e come essi interagiscono con il sistema stesso. Ciascun caso d'uso del diagramma racchiude una serie di scenari con cui l'attore interagisce per il raggiungimento di uno specifico obiettivo.



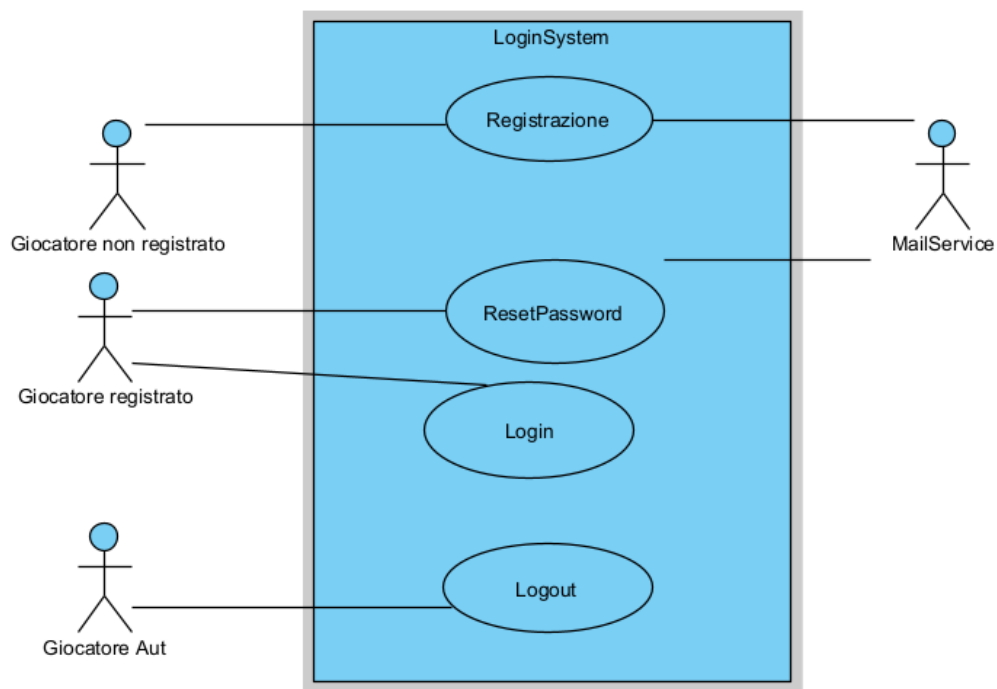


Diagramma di contesto

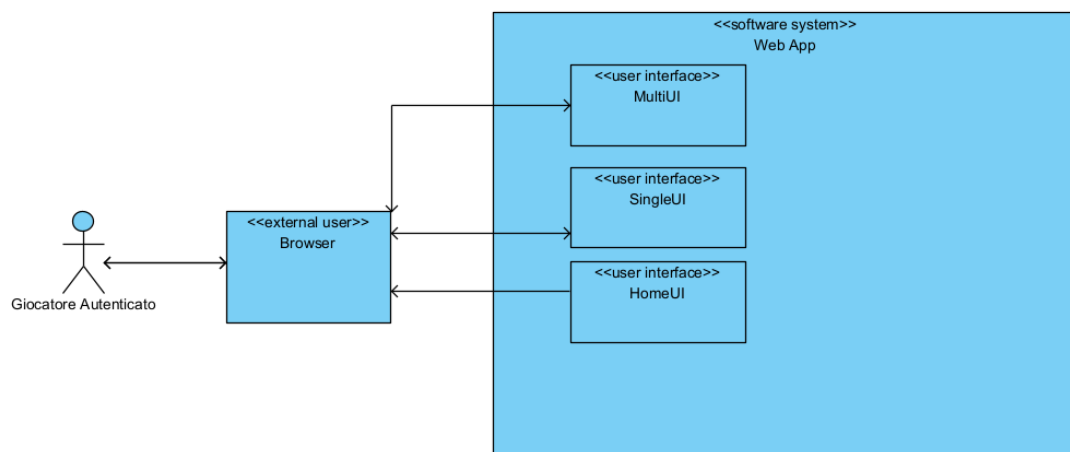
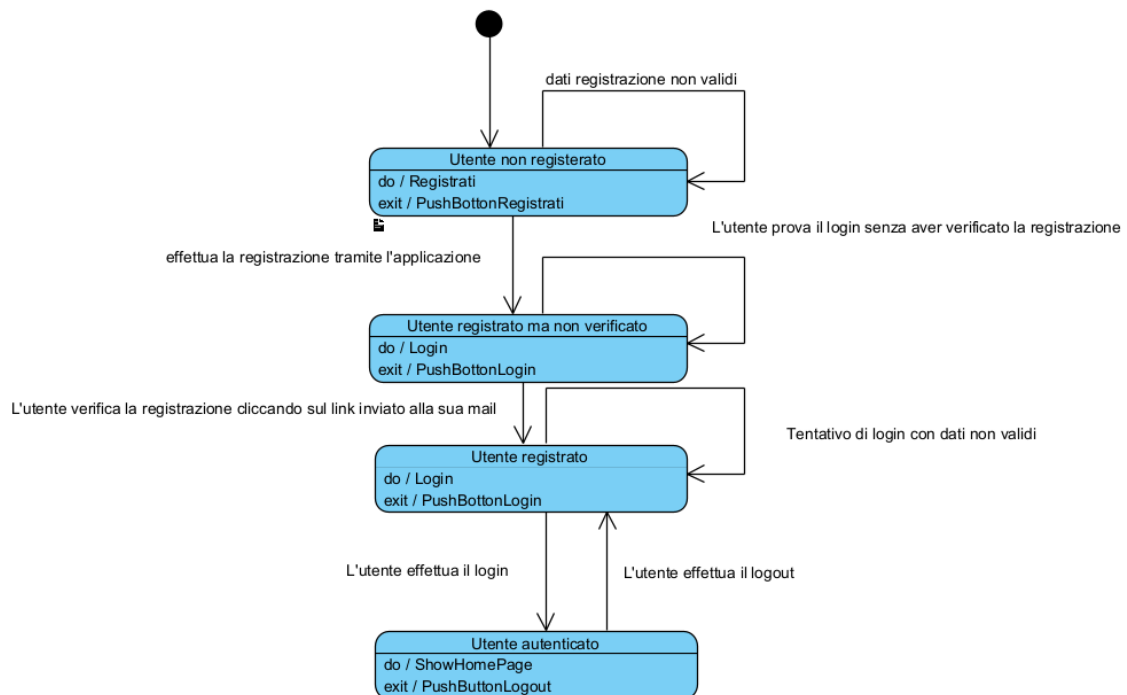


Diagramma degli stati



E' stato progettato uno SMD per modellare i diversi stati dell'utente nel momento in cui si registra e accede all'app.

Tale diagramma, chiaramente di alto livello, permette di capire tutti i vari passaggi che ci sono da quando l'utente apre l'applicazione nel Client la prima volta fino a quando è autenticato all'interno dell'app. Vengono mostrati quindi tutti gli stati, ovvero le condizioni in cui si può trovare l'utente in un determinato istante, e contemporaneamente tutti gli eventi che causano una transizione da uno stato all'altro.

Inizialmente, l'utente non è registrato e finchè i dati non sono validi, ovvero non vengono rispettati i requisiti richiesti per ognuno dei campi della registrazione, rimane in tale stato. L'exit activity di tale stato è, quindi, il push sul bottone "Registrati".

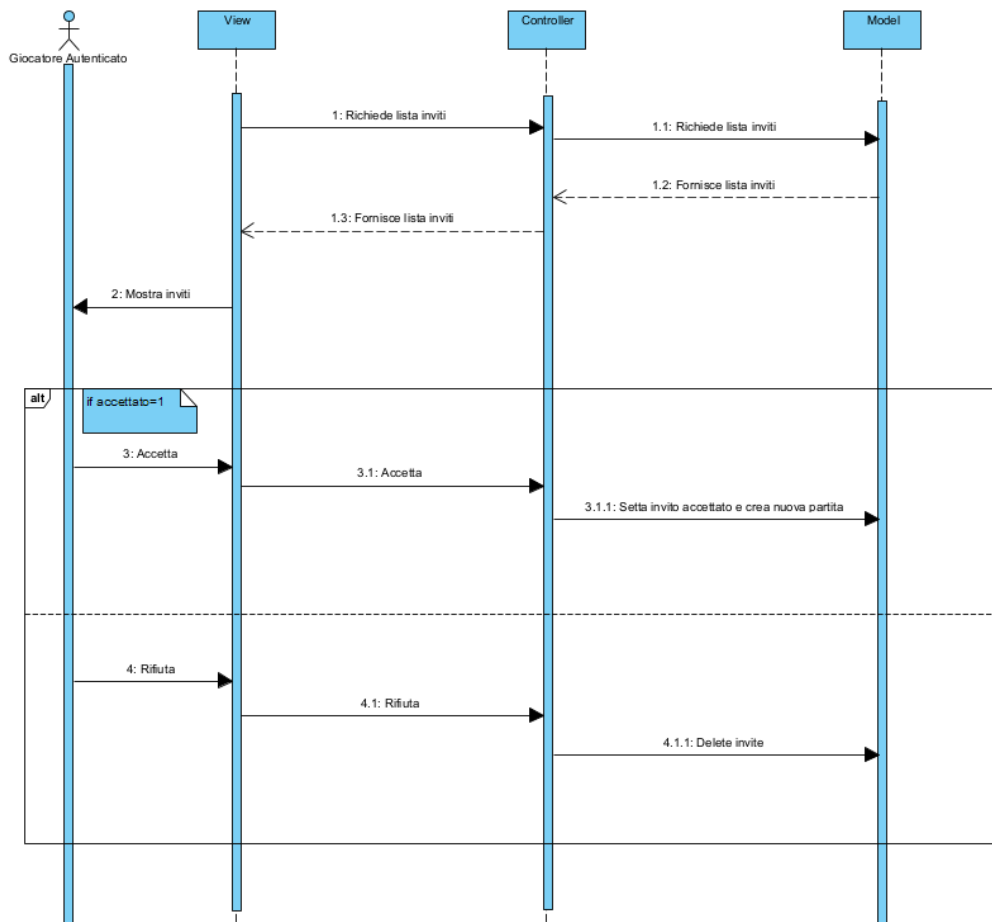
Dopodichè, l'utente effettua la registrazione tramite l'applicazione, ma non è ancora verificato. Transiterà nello stato di utente registrato solo quando verificherà la registrazione cliccando sul link inviato alla mail.

Da questo stato, che ha come exit activity il click sul bottone "Login", si passerà allo stato di utente autenticato solo dopo aver effettuato correttamente il login e quindi aver inserito i dati validi.

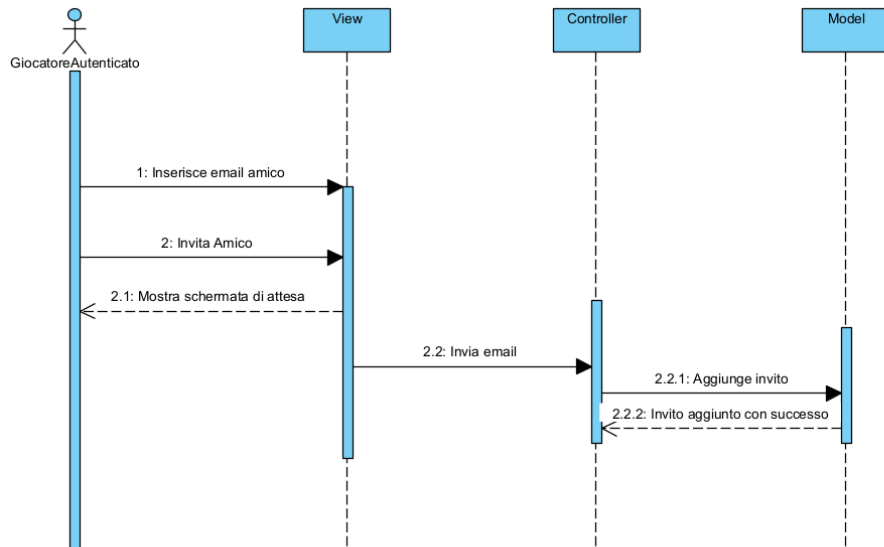
Nello stato di utente autenticato, avremo come do activity quella di

visualizzare la propria HomePage, come exitActiviyt abbiamo sempre il push sul bottone "Logout", che farà ritornare l'utente nello stato di utente registrato.

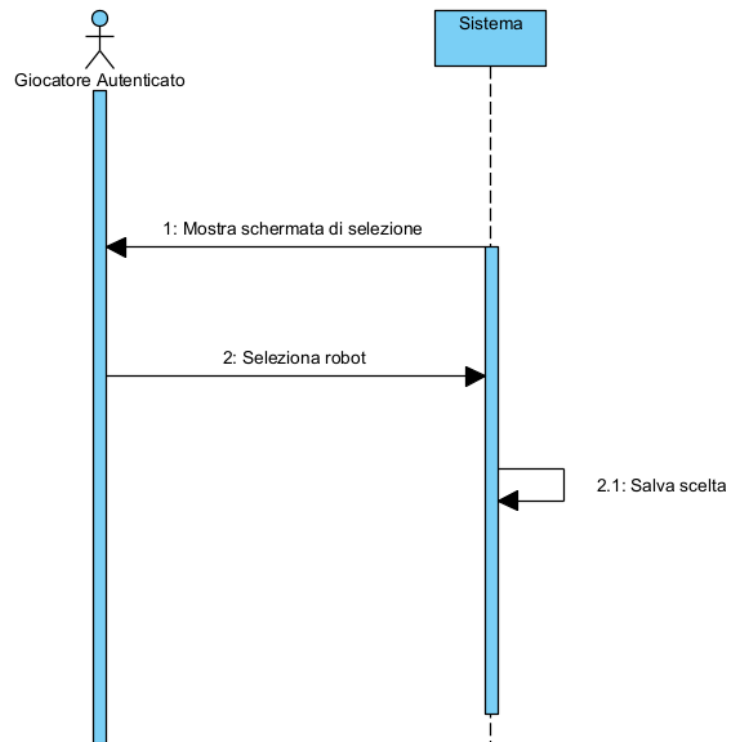
Diagrammi di sequenza di analisi



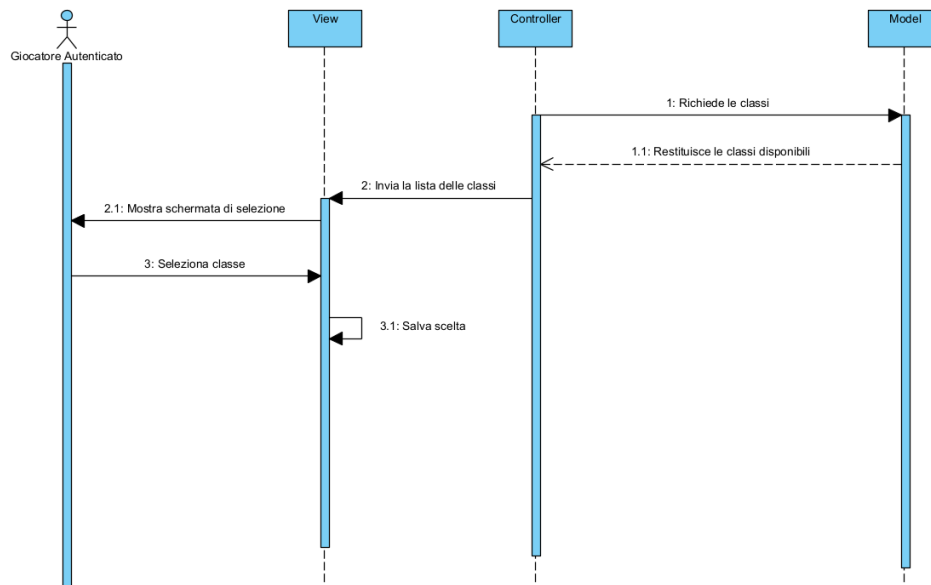
Accetta/Declina Invito



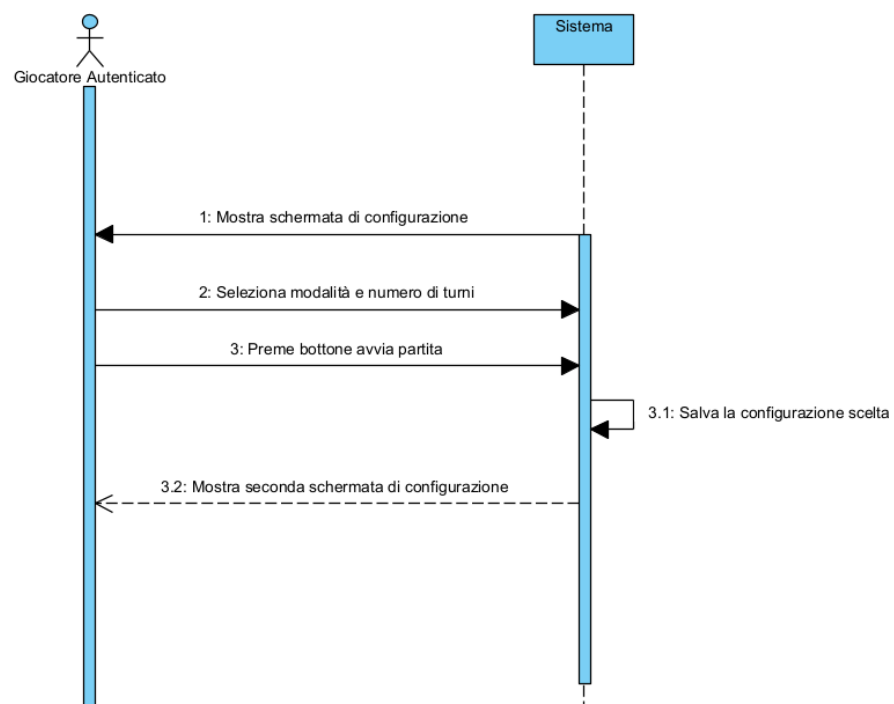
Invita Amico



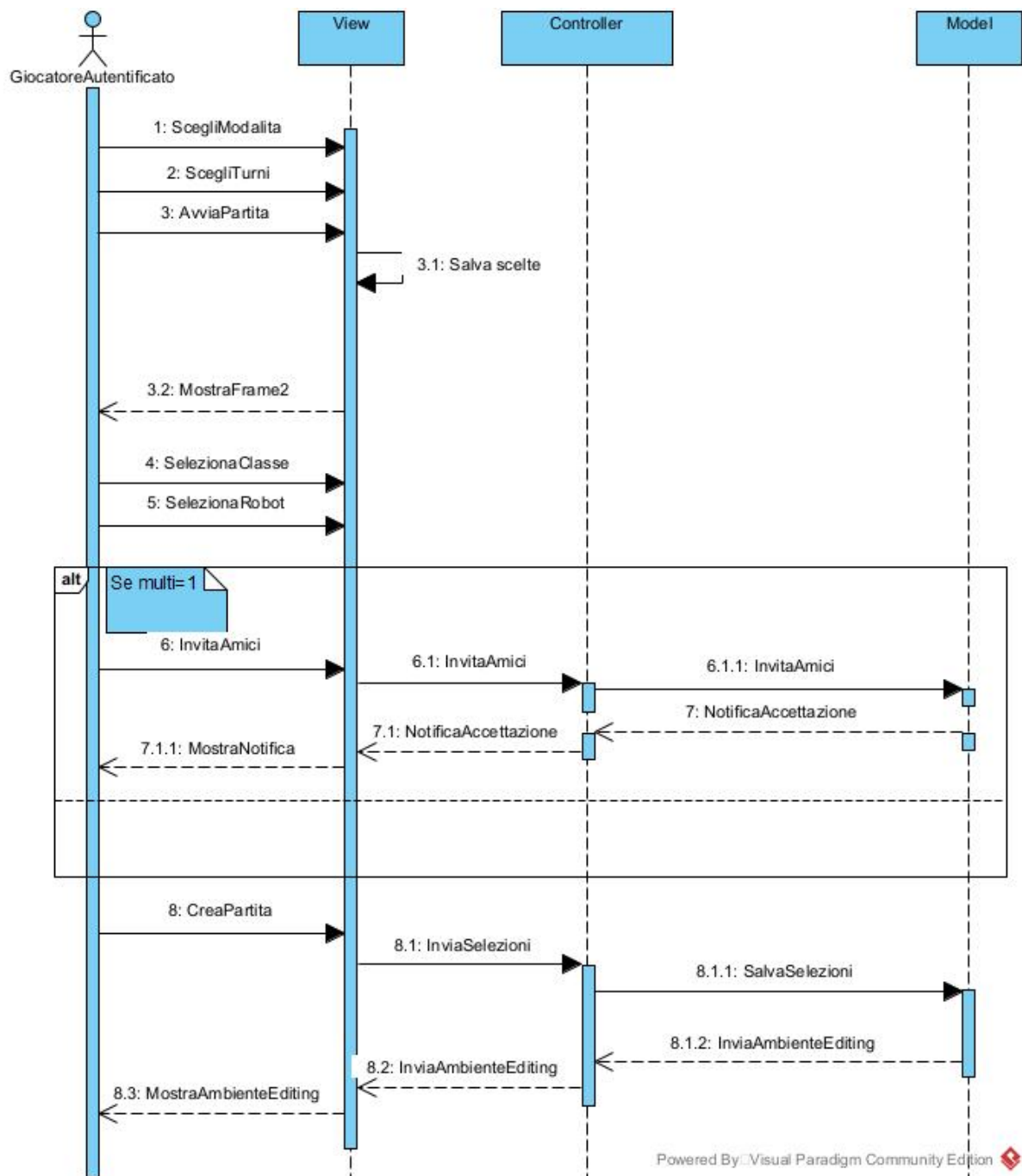
Seleziona Robot



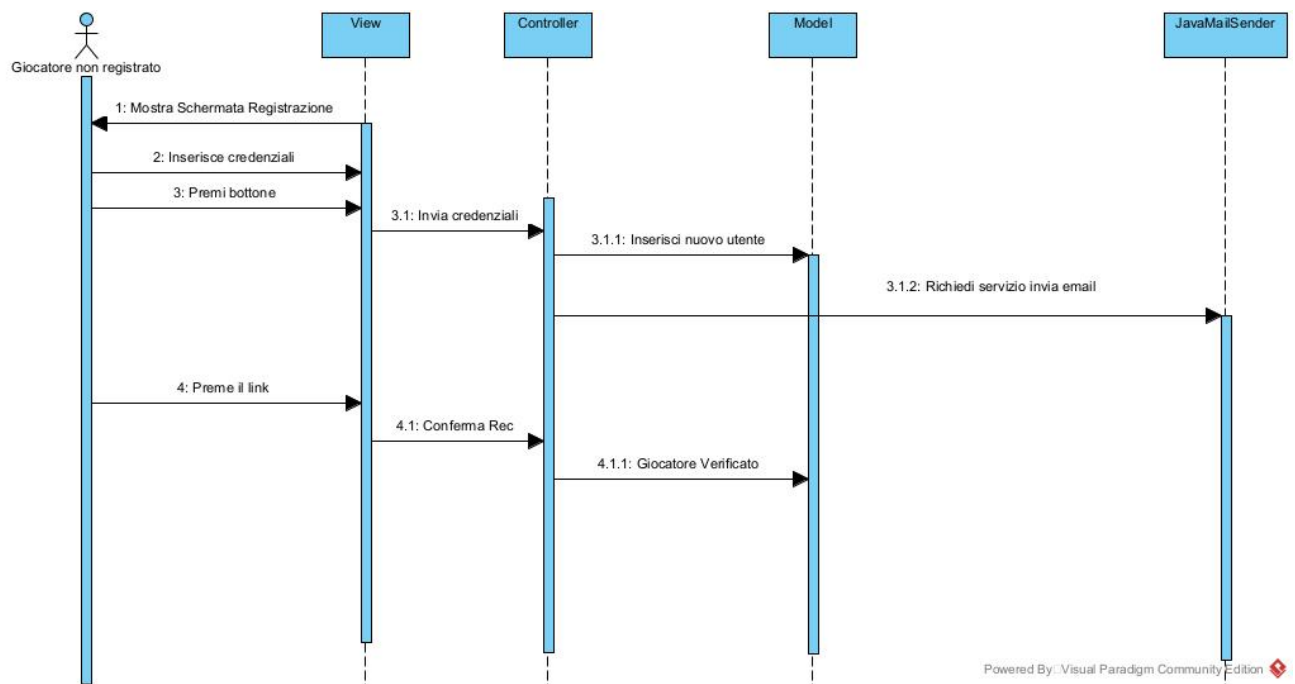
Seleziona Classe



Seleziona Modalità e Turni

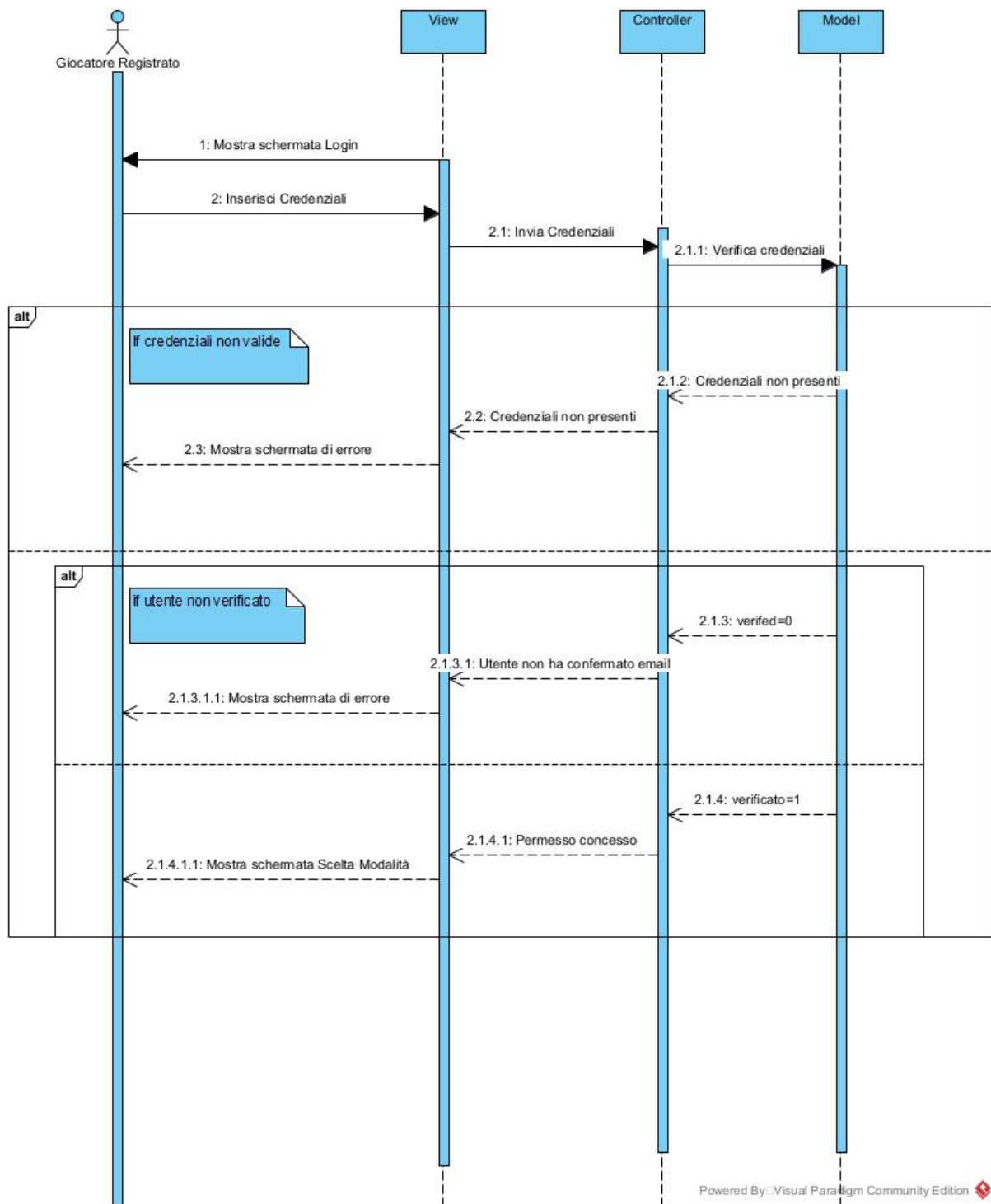


AvviaPartita



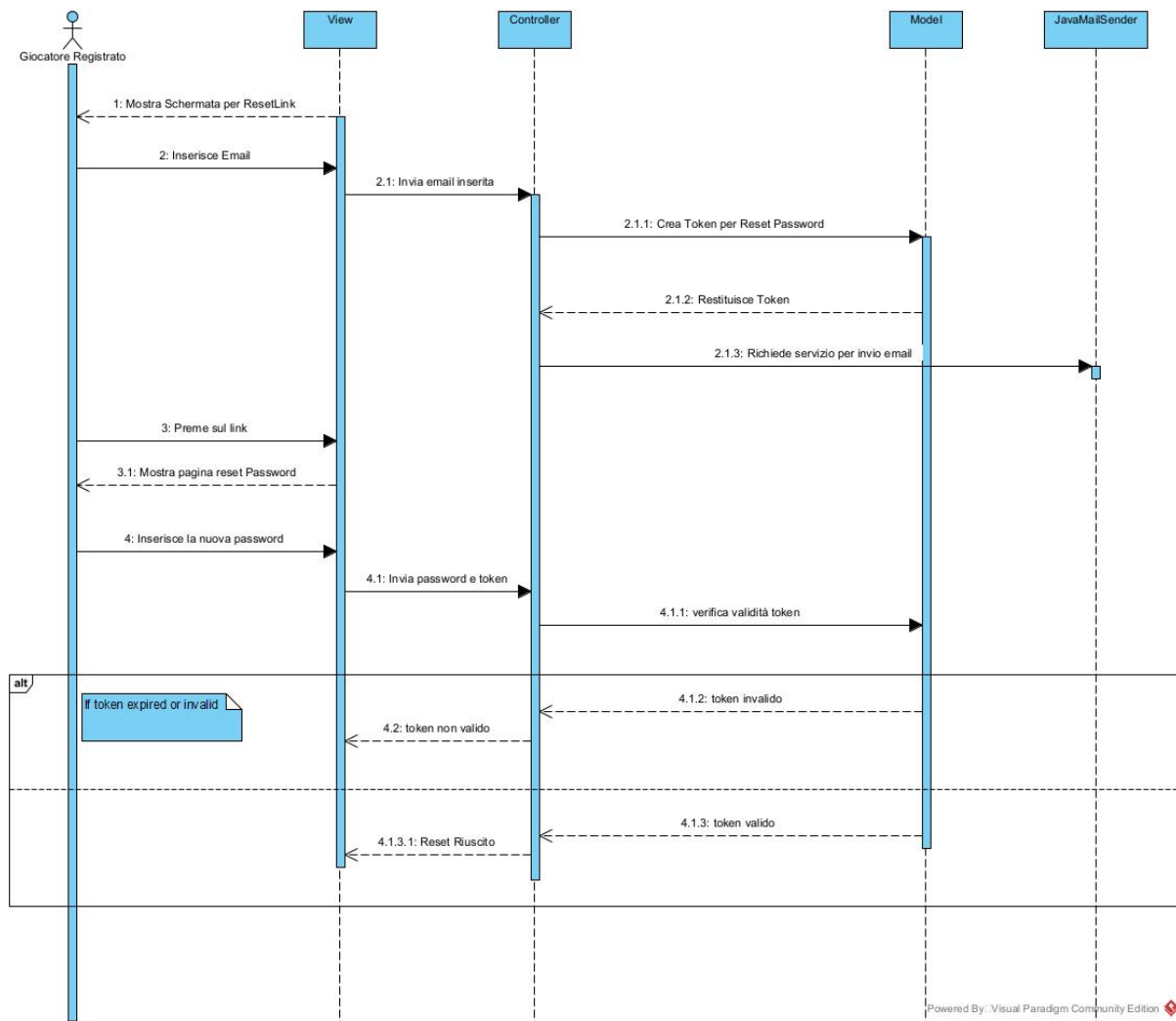
Powered By: Visual Paradigm Community Edition

Registrazione



Powered By: Visual Paradigm Community Edition

Login

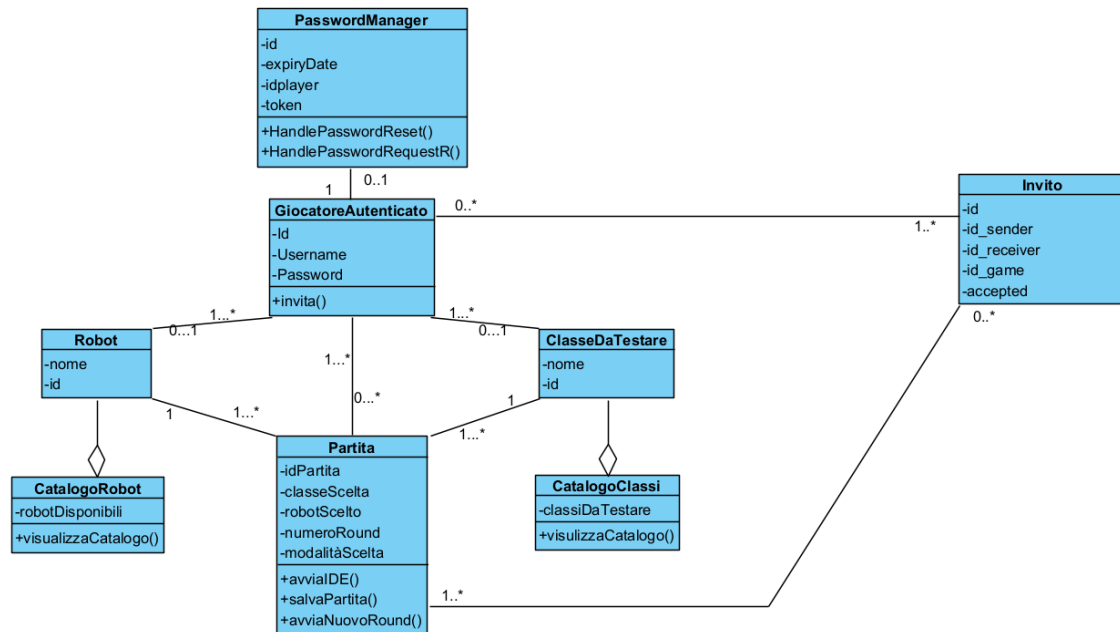


ResetPassword

System Domain Model

Durante la fase di analisi del sistema è stato sviluppato anche il diagramma di dominio del sistema software. In particolare, un modello di dominio è un **modello concettuale** che astrae e rappresenta gli aspetti (concetti, elementi o entità) principali pertinenti al dominio stesso (in tal caso il sistema software che si vuole realizzare), descrivendo le relazioni tra essi, oltre che i loro attributi e ruoli. Si è utilizzato dunque un UML Class Diagram per realizzare il modello di dominio (essendo il sistema object oriented, questo tipo di notazione si presta più che bene) di alto livello del sistema software. Sono state modellate dunque le seguenti entità principali componenti il dominio del sistema software:

- **Giocatore Autenticato**: si tratta dell'utilizzatore del gioco. Esso è univocamente identificato da un *Id*, ed accede al servizio utilizzando la coppia *email-password* determinata in fase di registrazione.
- **Partita**: viene istanziata dal giocatore autenticato, è identificata univocamente da un id e presenta una serie di attributi che rappresentano la configurazione scelta dal giocatore autenticato.
- **Invito**: viene istanziato dal giocatore autenticato, è identificato univocamente da un id e presenta l'id del giocatore mittente, giocatore destinatario, id partita.
- **Robot**: è il generatore automatico di casi di testi, contro cui confrontarsi.
- **Classe**: rappresenta la classe da testare che verrà scelta dal giocatore autenticato.
- **Password Manager**: rappresenta la classe che gestisce la richiesta di reset password.



Capitolo 3: Architettura e progettazione

La fase successiva riguarda l'architettura del software e la sua progettazione. Risulta dunque fondamentale, a questo punto, avviare l'individuazione di un'architettura complessiva che evidenzi i componenti che costituiscono il sistema con i loro collegamenti e interazioni. Diversi dei requisiti presentati nel capitolo precedente sono dunque stati raffinati e dettagliati; tuttavia, il processo di sviluppo impiegato non consente di avere una visione definitiva dell'architettura data la possibile variabilità dei requisiti stessi e delle possibili funzionalità richieste.

Client-Server Pattern

L'applicazione è stata realizzata seguendo il pattern ClientServer. Secondo quest'ultimo, infatti, il Server è in grado di fornire ai vari Client collegati diverse funzionalità anche avvalendosi di servizi esterni, tramite un'interazione di tipo richiesta-risposta. Tale scelta architetturale è giustificata da numerosi vantaggi,

- Connessione tra Client e Server stabilita dinamicamente.
- Basso accoppiamento tra Server e i Client.
- Il numero di Client può scalare facilmente, anche se ciò dipende dalle capacità del Server, il quale può scalare a sua volta.
- Client e Server possono evolvere indipendentemente.
- L'interazione con l'utente è circoscritta al solo Client.

Ovviamente, si è consapevoli dei limiti che tale scelta comporta soprattutto in termini di possibili ritardi dei messaggi dovuti alla congestione/degradazione della rete.

Per la strutturazione dell'applicazione si è scelto come pattern di riferimento l'MVC (Model - View - Controller). Tale pattern si adatta perfettamente alle nostre esigenze con la possibilità di separare in maniera netta i componenti di presentazione dei dati da quelli che gestiscono i dati stessi.



I componenti principali di tale pattern sono:

Model: contiene le classi le cui istanze rappresentano i dati da manipolare e visualizzare sulle varie page dell'applicazione. Questo non deve essere dipendente né dal livello View né da quello Controller, pur esponendo ai due le funzionalità per l'accesso e l'aggiornamento dei dati stessi.

Inoltre, esso ha la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller per permettere alle View di presentare dati sempre aggiornati;

View: gestisce la logica di presentazione (UI) dei dati contenuti nel Model. Essa interagisce con la strategia push coi dati del Model, implementando il pattern Observer;

Controller: contiene gli oggetti che controllano e gestiscono l'interazione sia con il livello View che con il Model. Esso realizza la corrispondenza tra input utente e le azioni eseguite dal Model, selezionando anche le schermate della View da presentare.

Servizi: contiene le classi di utilità come quella per la crittografia o per l'interfacciamento col database. Queste classi vengono utilizzate a supporto delle funzionalità del Controller.

Model:

Il package model contiene le seguenti classi:

- Game: classe che consente di creare, e gestire i dati relativi alle partite.
- Player: classe che consente di gestire e passare le informazioni dello studente tra le classi dell'applicazione.
- Invitation: classe che consente di creare e gestire i dati relativi agli inviti.
- ResetToken: classe che consente di resettare la password.

View:

In questo caso, il package view contiene tutte le varie viste dell'applicazione.

- *Modalità.js*: pagina per selezionare modalità di gioco e numero di turni.
- *Single.js*: pagina per selezionare classe da testare e robot da sfidare in modalità giocatore singolo
- *Multi.js*: pagina per selezionare la classe da testare, robot da sfidare e player da invitare
- *ResetPass.js*: pagina per resettare la password.
- *PasswordResetlink.js*: pagina per inviare l'email di reset.

Controller:

Il package controller contiene tutte le classi che intercettano le richieste di tipo GET e POST, richiamando i servizi per soddisfare le varie richieste e restituire la view interessata. I controller sono i seguenti:

- **PlayerController**: Questo controller viene utilizzato per recuperare le informazioni del player quali idplayer ed emailplayer.
- **GameController**: Questo controller ci permette di creare e salvare una nuova partita
- **InvitationController**: Questo controller ci permette di recuperare gli inviti ricevuti, inviare un invito a un altro player e accettare o declinare un invito.
- **ResetTokenController**: Questo controller ci permettere di inviare richiesta reset password e di conseguenza resettarla

Vantaggi del MVC:

La scelta di tale Pattern Architeturale per il Client è giustificata dai seguenti vantaggi:

- Indipendenza dei vari componenti, che permette la suddivisione del lavoro.
- Esiste la possibilità di scrivere viste e controllori diversi utilizzando lo stesso modello di accesso ai dati e, quindi, riutilizzando parte del codice già scritto precedentemente.
- Supporto a nuovi tipi di Client con la semplice riscrittura di alcune View e alcuni Controller.
- Utilizzo di un modello rigido e di regole standard nella stesura del progetto, cosa che facilita un eventuale lavoro di manutenzione e agevola la comprensione da parte di altri programmatori.
- La possibilità di avere un controllore separato dal resto dell'applicazione, rende la progettazione più semplice e permette di concentrare gli sforzi sulla logica del funzionamento.
- Interattività, fortemente richiesta dalla nostra applicazione.

Diagramma dei Packages

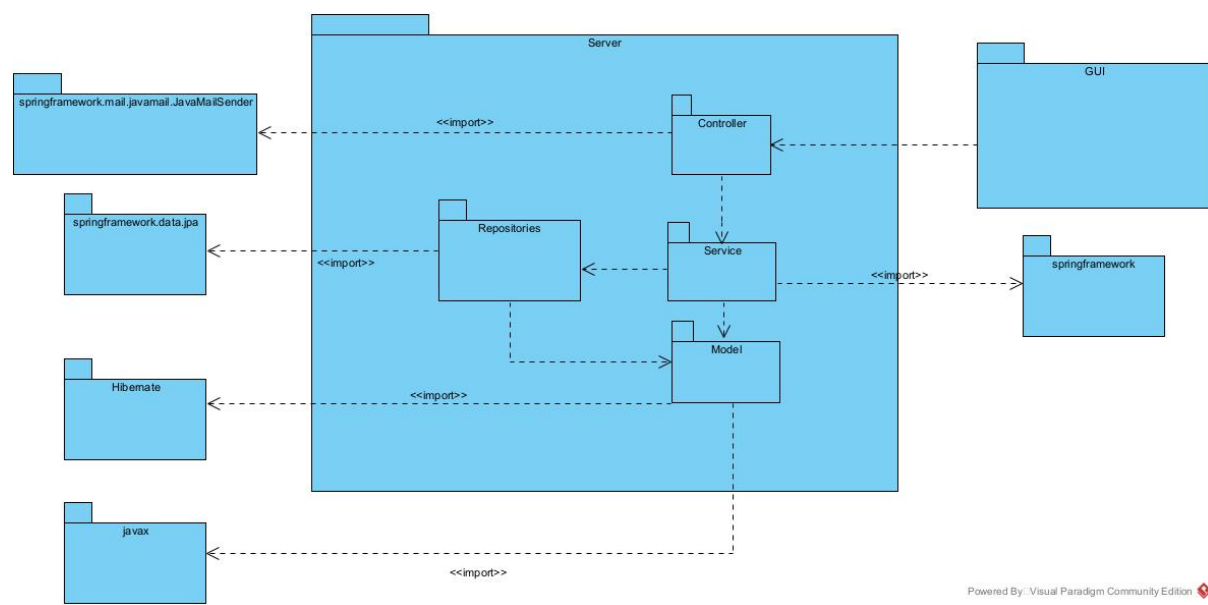
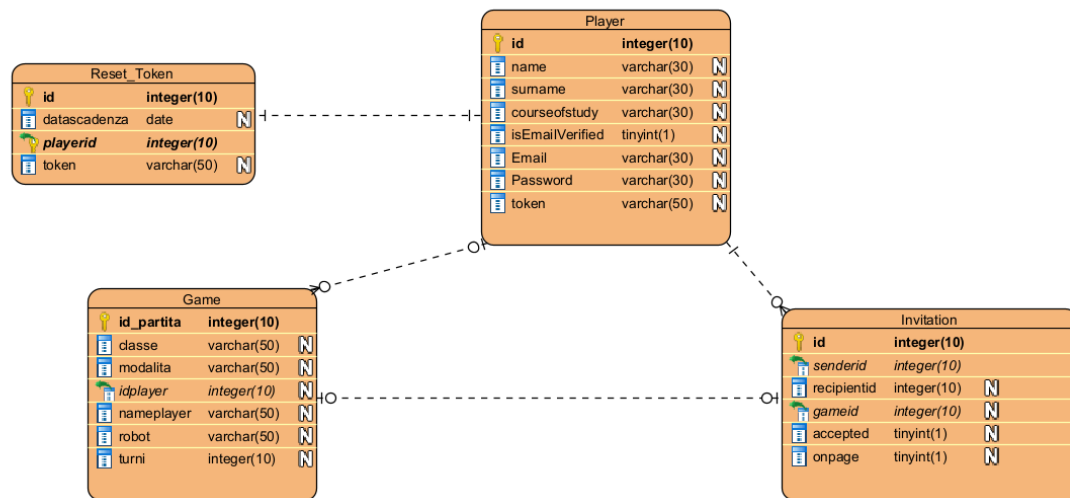


Diagramma ER

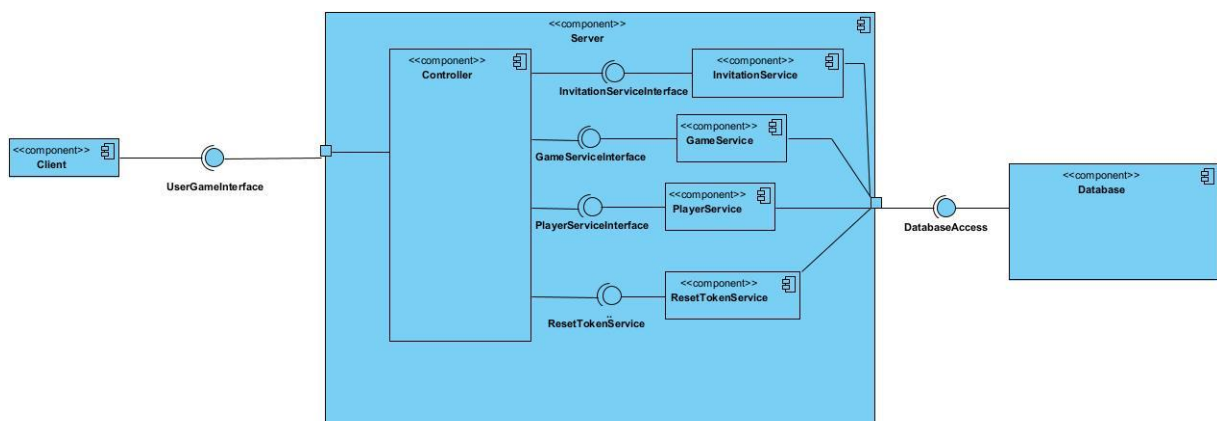


Vista Componenti e Connettori

Come primo passo nella definizione dell'architettura complessiva è opportuno valutare la strutturazione

del sistema nei vari componenti e le modalità di interazione per la realizzazione delle funzionalità richieste, tenendo sempre in conto anche i requisiti non funzionali specificati. A tal scopo può essere utile rappresentare il sistema tramite una vista *Componenti e Connettori*.

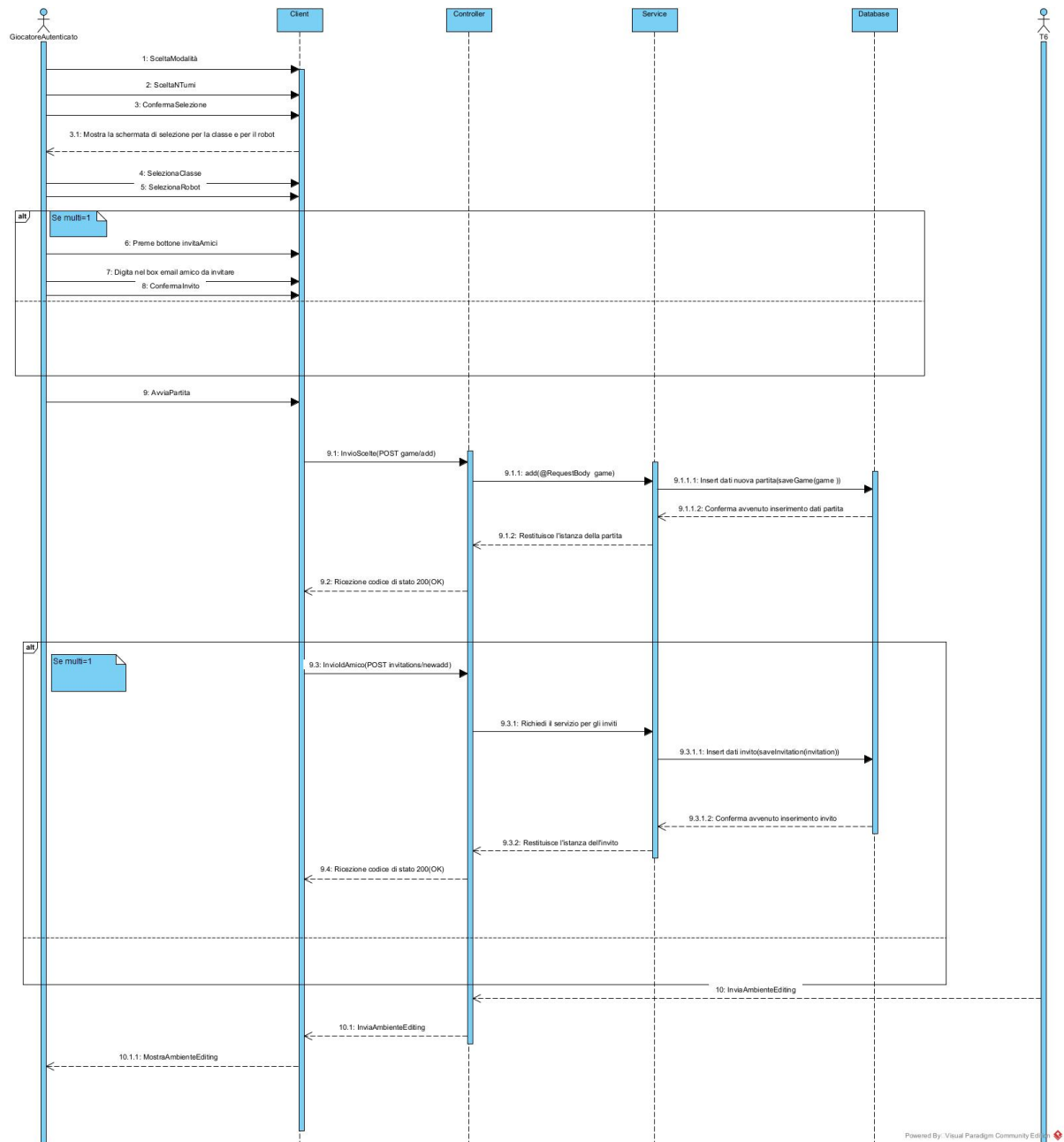
Diagramma dei componenti



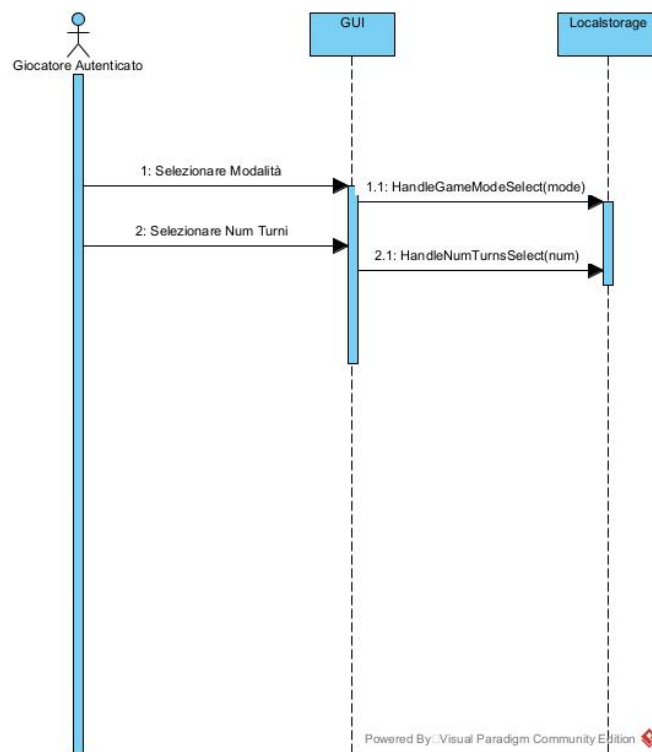
Nel caso specifico, i **componenti** rappresentati sono:

- **Client**: utilizzatore dell'applicazione
- **PlayerService**: Componente che ha il compito di gestire richiedendo e memorizzando le informazioni relative a tutti i player del gioco.
- **GameService**: Componente che svolge la funzione di gestire e memorizzare i dati relativi alla partita.
- **InvitationService**: Componente che svolge la funzione di gestire e memorizzare i dati relativi agli inviti.
- **ResetTokenService**: Componente che permette di resettare la password mediante la gestione di un token.
- **Database**: Componente che rappresenta un sistema di gestione dati.

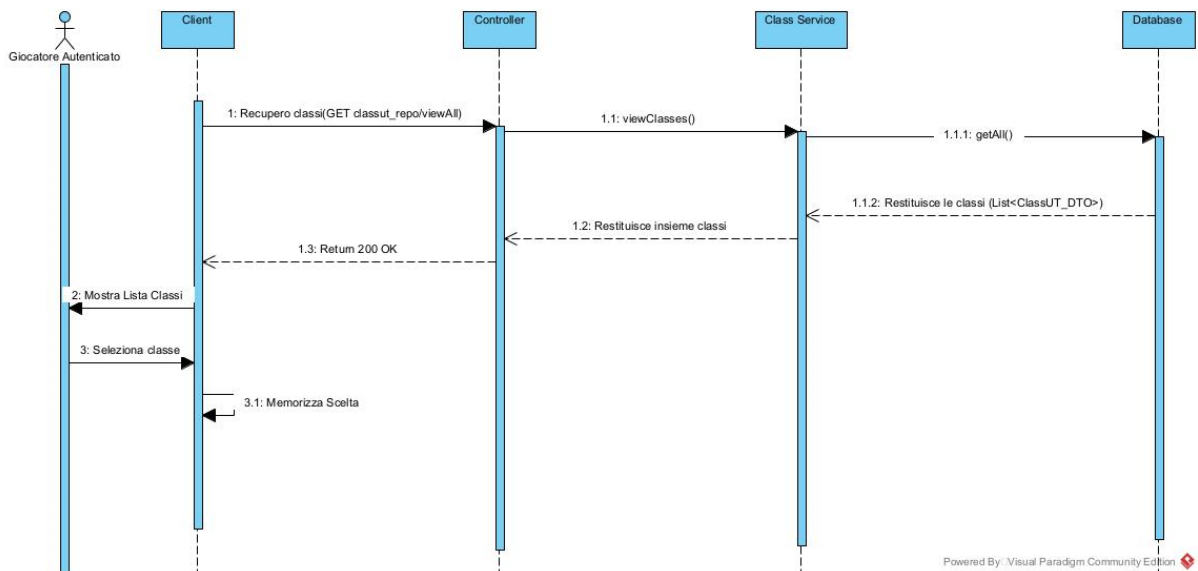
Diagrammi di sequenza di progetto



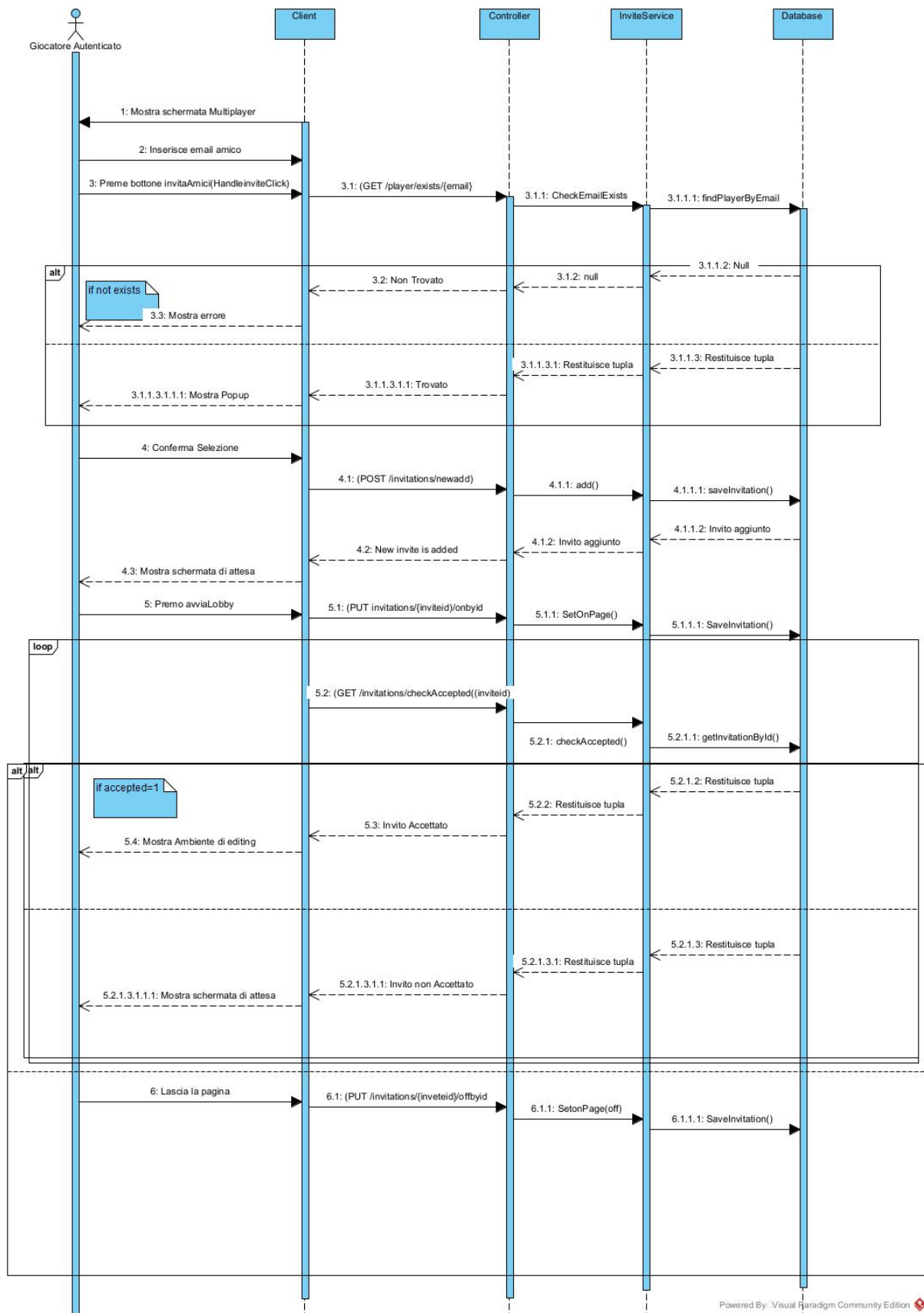
AvviaPartita (progettazione)



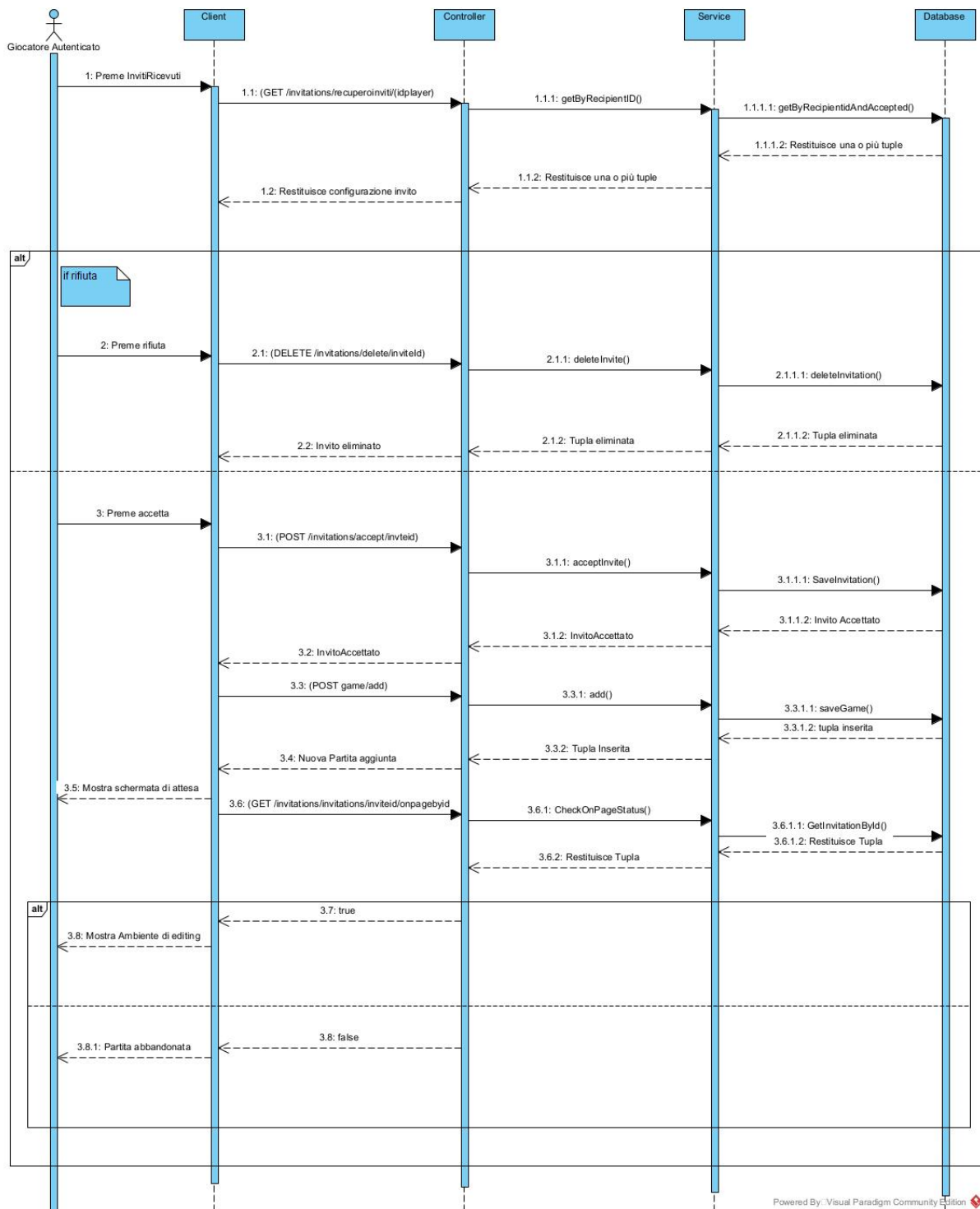
SceltaModalitaTurni (progettazione)



SceltaClasse (progettazione)



InvitaAmici (progettazione)



AccettaDeclinaInvito (progettazione)

Documentazione API

Game Controller

Method	Url	Request body	Response body
POST	/game/add	{ "classe":<classe> "robot":<robot> "turni":<turni> "id":<id> }	{ "ok":<result> }
GET	/game/{id}		{ "id_partita":<id_partita> "classe":<classe> "robot":<robot> "turni":<turni> "modalita":<modalita> "id":<id> }

Player Controller

Method	Url	Request body	Response body
POST	/player/add	{ "email":<email> "password":<password> "name":<name> "surname":<surname> "courseofstudy":<courseofstudy> }	{ "ok":<result> }
POST	/player/login/	{ "email":<email> "password":<password> }	{ "ok":<result> }
GET	/player/confirm?email={email}&token={token}		{ "ok":<result> }
POST	/player/exists/{email}		{ "message": "Il player è presente" }

Invitation Controller

Method	URL	Request Body	Response
POST	/invitations/newadd	{ "senderId": 1, "recipientId": 2, "gameId":123, "accepted": false }	{ "id": 1, "message": "New invite is added" }
GET	/invitations/getAll		{ "senderId": 1, "recipientId": 2, "gameId":1, "accepted": false }
GET	/invitations/recuperoinviti/{recipientid}		{ "id": 1, "senderid": 1, "recipientid": 2, "game_id":1, "accepted": false }
PUT	/invitations/accept/{inviteId}		{ "message": "Invite accepted and removed from database" }
DELETE	/invitations/delete/{inviteid}		{ "message": "Invite deleted successfully" }
GET	/invitations/checkAccepted/{inviteId}		{ "message": "Invitation with ID 1 has been accepted" }
GET	/invitations/{senderId}/recent		["recipient1@example.com", "recipient2@example.com", "recipient3@example.com"]
PUT	/invitations/{invitationId}/onbyid		{ "message": "onpage has been set for invitation with ID 1" }
PUT	/invitations/{invitationId}/offbyid		{ "message": "onpage has been set for invitation with ID 1" }
GET	/invitations/invitations/{invitationId}/onpagebyid		{ "invitationId": 1, "onpage": true }

- **POST /invitations/newadd** - Questa API viene utilizzata per aggiungere un nuovo invito al database. Il corpo della richiesta deve contenere un oggetto Invito. La risposta restituirà una mappa di stringhe e oggetti con l'ID del nuovo invito e un messaggio che indica se l'invito è stato aggiunto con successo o meno.
- **GET /invitations/getAll** - Questa API viene utilizzata per recuperare tutti gli inviti esistenti dal database. La risposta restituirà un oggetto lista di Inviti contenente tutti gli inviti.
- **GET /invitations/recuperoinviti/{recipientid}** - Questa API viene utilizzata per recuperare tutti gli inviti per un destinatario specifico che non li ha ancora accettati. La risposta restituirà una lista di mappe di stringhe e oggetti contenente l'ID del mittente, l'ID del destinatario, l'ID del gioco e lo stato di accettazione di ciascun invito.
- **PUT /invitations/accept/{inviteId}** - Questa API viene utilizzata per accettare un invito e rimuoverlo dal database. L'URL della richiesta deve contenere l'ID dell'invito da accettare. La risposta restituirà una mappa di stringhe e oggetti con un messaggio che indica se l'invito è stato accettato e rimosso con successo dal database o meno.
- **DELETE /invitations/delete/{inviteId}** - Questa API viene utilizzata per eliminare un invito dal database. L'URL della richiesta deve contenere l'ID dell'invito da eliminare. La risposta restituirà una mappa di stringhe e oggetti con un messaggio che indica se l'invito è stato eliminato con successo o meno.
- **GET /invitations/checkAccepted/{inviteId}** - Questa API viene utilizzata per verificare se un invito è stato accettato o meno. L'URL della richiesta deve contenere l'ID dell'invito da verificare. La risposta restituirà una mappa di stringhe e oggetti con lo stato di accettazione dell'invito.
- **GET /invitations/{senderId}/recent** - Questa API viene utilizzata per recuperare gli ultimi inviti inviati da un mittente specifico. L'URL della richiesta deve contenere l'ID del mittente. La risposta restituirà una lista di stringhe contenenti l'ID degli inviti recenti.
- **PUT /invitations/{invitationId}/onbyid** - Questa API viene utilizzata per impostare lo stato "true" di un invito specifico. L'URL della richiesta deve contenere l'ID dell'invito e il parametro "onpage". La risposta restituirà

una mappa di stringhe e oggetti con un messaggio che indica se lo stato è stato impostato con successo o meno.

- **PUT /invitations/{invitationId}/offbyid** - Questa API viene utilizzata per impostare lo stato "false" di un invito specifico. L'URL della richiesta deve contenere l'ID dell'invito e il parametro "onpage". La risposta restituirà una mappa di stringhe e oggetti con un messaggio che indica se lo stato è stato impostato con successo o meno.
- **GET /invitations/invitations/{invitationId}/onpagebyid** - Questa API viene utilizzata per verificare lo stato "on" di un invito specifico. L'URL della richiesta deve contenere l'ID dell'invito. La risposta restituirà una mappa di stringhe e oggetti con lo stato "on" dell'invito.

RESET CONTROLLER

Method	URL	Request body	Response body
POST	/preset/request	{ "email": "johndoe@example.com" }	{ "ok":<result> }
POST	/preset/reset	{ "token": "abc123", "password": "newpassword" }	{ "ok":<result> }

Capitolo 4: Stima dei costi

Per stimare i costi e i tempi necessari per completare un progetto di sviluppo software, è possibile utilizzare la tecnica degli Use Case Points (UCP). Questo metodo si basa sui requisiti del sistema definiti tramite casi d'uso e si applica quando viene impiegato UML per la progettazione e sviluppo del software. L'UCP consente di stimare la dimensione del software necessaria per il progetto, includendo fattori tecnici ed ambientali. Utilizzando questo metodo, è possibile calcolare l'effort necessario per completare il progetto. La dimensione del software (UCP) viene calcolata in base agli elementi dei casi d'uso, utilizzando un fattore di correzione per considerare le diverse variabili. In questo modo, l'UCP rappresenta uno strumento utile per la stima dei costi e dei tempi associati ad un progetto di sviluppo software basato su UML.

Unadjusted Use Case Weight (UUCW)

I valori di riferimento assegnati a ciascun caso d'uso, detti anche USE CASE CLASSIFICATION, sono:

- Simple(1-3 transiction) Weight 5.
- Average(4-7 transiction) Weight 10.
- Complex(8 or more transiction) Weight 15.

USE CASE	COMPLESSITA'
GIOCATORE AUTENTICATO – Avvia Configurazione	Simple
GIOCATORE AUTENTICATO – Scelta Modalita e Turni	Average
GIOCATORE AUTENTICATO – Scelta Classe	Simple
GIOCATORE AUTENTICATO – Scelta Robot	Simple
GIOCATORE AUTENTICATO – Invita Amici	Simple
GIOCATORE AUTENTICATO – Avvia Nuova Partita	Simple
GIOCATORE AUTENTICATO – Accetta/Declina Invito	Complex

Valutazione dell'UUCW:

$$UCCW = (TotalN.ofSimpleUC * 5) + (TotalN.AverageUC * 10) + (TotalN.ComplexUC * 15)$$

Complessità	Peso	N.UC	N.UC*Peso
Simple	5	5	25
Average	10	1	10
Complex	15	1	15
TOTALE			50

$$UUCW = 50$$

Unadjusted Actor Weight (UAW)

Valutazione dell'associazione attori del sistema e complessità di realizzazione.

I valori di riferimento:

- Simple: Sistema esterno che deve interagire con il sistema utilizzando un'API ben definita - 1.
- Average: Sistema esterno che deve interagire con il sistema utilizzando protocolli di comunicazione standard (es. TCP/IP, FTP, HTTP, database) - 2.
- Complex: Attore umano che utilizza un'interfaccia applicativa GUI - 3.

Valutazione dell'UAW:

$$UAW = (TotalN.ofSimpleActors * 1) + (TotalN.AverageActors * 2) + (TotalN.ComplexActors * 3)$$

ACTORS	COMPLESSITÀ
GIOCATORE AUTENTICATO	Complex

Complessità	Peso	N. Actors	N. Actors*Peso
Simple	1	0	0
Average	2	0	0
Complex	3	1	3
Totale			3

$$UAW = 3$$

Technical Complexity Factor (TCF)

Il TCF è uno dei fattori applicati alla dimensione stimata del software al fine di tenere conto delle considerazioni tecniche del sistema. Si determina assegnando un punteggio compreso tra 0 (il fattore è irrilevante) e 5 (il fattore è essenziale) a ciascuno dei 13 fattori tecnici elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

I valori di riferimento:

- T1 - Sistema distribuito, 2.0.
- T2 - Tempo di risposta/obiettivi prestazionali, 1.0.
- T3 - Efficienza dell'utente finale, 1.0.
- T4 - Complessità di elaborazione interna, 1.0.
- T5 - Riutilizzabilità del codice, 1.0.
- T6 - Facile da installare, 0.5.
- T7 - Facile da usare, 0.5.
- T8 - Portabilità su altre piattaforme, 2.0.
- T9 - Manutenzione del sistema, 1.0.
- T10 - Elaborazione simultanea/parallela, 1.0.
- T11 - Funzioni di sicurezza, 1.0.
- T12 - Accesso per terze parti, 1.0.
- T13 - Formazione dell'utente finale, 1.0.

Per il sistema software in esame si usano i seguenti valori di fattori:

T1 (3), T2 (4), T3 (5), T4 (2), T5 (5), T6 (4), T7 (5), T8 (4), T9 (3), T10 (3), T11 (2), T12 (5), T13 (1).

Il totale di tutti i valori calcolati e il fattore tecnico (TF), calcolato come segue:

$$TF = 3 * 2 + 4 * 1 + 5 * 1 + 2 * 1 + 5 * 1 + 4 * 0.5 + 5 * 0.5 + 4 * 2 + 3 * 1 + 3 * 1 + 2 * 1 + 5 * 1 + 1 * 1 = 48.5$$

Il TF viene quindi utilizzato per calcolare il TCF con la seguente formula:

$$TCF = 0.6 + \left(\frac{TF}{100}\right) = 0.6 + \left(\frac{48.5}{100}\right) = 0.6 + 0.485 = 1.085$$

Environmental Complexity Factor (ECF)

L'ECF è un fattore applicato alla dimensione stimata del software che tiene conto delle considerazioni ambientali del sistema. Viene determinato assegnando un punteggio compreso tra 0 (nessuna esperienza) e 5 (esperto) a ciascuno degli 8 fattori ambientali elencati nella tabella seguente. Questo punteggio viene quindi moltiplicato per il valore ponderato definito per ciascun fattore.

Valori di riferimento:

- E1 - Familiarità con il processo di sviluppo utilizzato, 1.5.
- E2 - Esperienza applicativa, 0.5.
- E3 - Esperienza di team orientata agli oggetti, 1.0.
- E4 - Capacità di lead analyst, 0.5.
- E5 - Motivazione del team, 1.0.
- E6 - Stabilità dei requisiti, 2.0.
- E7 - Personale part-time, -1.0.
- E8 - Linguaggio di programmazione difficile, -1.0.

Per il sistema software in esame si usano i fattori:

E1 (2) + E2 (2) + E3 (2) + E4 (3) + E5 (5) + E6 (4) + E7 (0) + E8 (2).

Il totale di tutti i valori calcolati e il fattore ambiente (EF), calcolato come segue:

$$EF = 2 * 1.5 + 2 * 0.5 + 2 * 1 + 3 * 0.5 + 5 * 1 + 4 * 2 + 0 * (-1) + 2 * (-1) = 18.5$$

L'EF viene quindi utilizzato per calcolare l'ECF con la seguente formula:

$$ECF = 1.4 + (-0.03 * 18.5) = 0.845$$

Total Use Case Points (UCP)

Infine, una volta determinate le dimensioni del progetto Unadjusted (UUCW e UAW), e calcolati il fattore tecnico (TCF) e il fattore ambientale (ECF) si può calcolare l'UCP.

L'UCP viene calcolato in base alla seguente formula:

$$\begin{aligned} \mathbf{UCP} &= (UUCW + UAW) * TCF * ECF = (50 + 3) * 1.085 * 0.845 \\ &= 48.59 \end{aligned}$$

Da quest'ultimo parametro è possibile valutare il numero di ore di lavoro totali considerando che un singolo UC ` venga sviluppato in 8 ore mediamente:

$$\mathbf{Th} = UCP * 8 = 48.59 * 8 = 388,73h$$

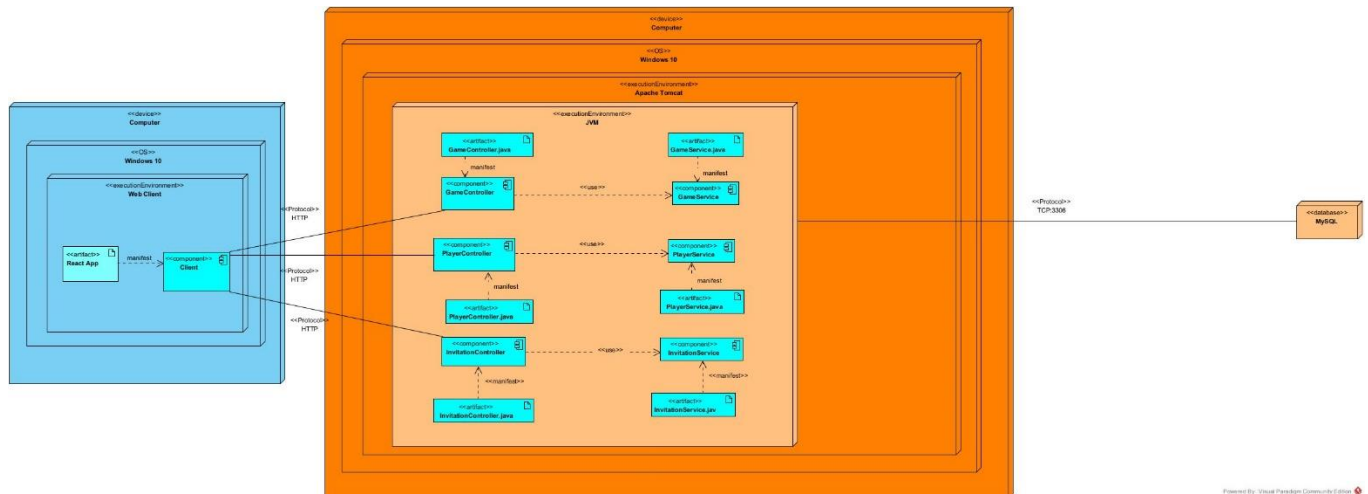
In definitiva, se si suppone che un membro del progetto lavori per circa 60 ore a settimana (Wh), ed il team sia composto da 3 membri, essi lavoreranno per un totale di 180 ore a settimana (TWh).

Th è il numero di ore di lavoro totali necessarie a terminare il progetto e TWh rappresenta il numero di ore di lavoro a settimana del gruppo di lavoro (team), dunque facendone il rapporto si ottiene il numero di settimane necessarie per terminare il lavoro:

$$\frac{Th}{TWh} = \frac{388.73}{180} = 2.16 \text{ settimane}$$

Capitolo 5: Deployment e tecnologie

Deployment Diagram (in locale)



Tecnologie utilizzate

React Framework

React è un framework open source molto popolare per lo sviluppo del frontend delle applicazioni web, grazie alla sua capacità di semplificare la creazione di interfacce utente reattive e altamente performanti utilizzando componenti riutilizzabili. La sua architettura basata su una virtual DOM permette di gestire in modo efficiente gli aggiornamenti della pagina e ottenere prestazioni elevate. Grazie alla sua flessibilità e alla vasta gamma di librerie e strumenti disponibili, React è una scelta comune per il frontend delle applicazioni web moderne e scalabili.


Schermata di registrazione

Sign up

Course of study
Seleziona un corso di studi

☐ I agree all statements in [Terms of service](#)

Register



Schermata di Login

Login

Indirizzo email

Password

[Registrati](#)

[Login](#)

[Password dimenticata?](#)

Schermata della scelta della modalità e del numero di turni

Configura nuova partita

cerdigufu@gufum.com

Modalità di gioco

Singolo giocatore

Multigiocatore

Seleziona numero di turni

1 Turno

Avvia gioco

Profilo

Inviti ricevuti

Esci

Schermata per la scelta della classe, del robot e per invitare altri giocatori

Sceita configurazione cerdigufyu@gufum.com

Classi disponibili

Cerca classe

Classe 1

☒ Classe 2

Classe 3

vobahom232@bodeem.com **Invita**

Amici Suggesti

vobahom232@bodeem.com

Robot disponibili

☒ Randoop

☐ Evosuite

Conferma selezione

← →

Schermata inviti ricevuti

Configura nuova partita vobahom232@bodeem.com

Modalità di gioco

Singolo giocatore

Multigiocatore

Inviti ricevuti

Emails

- Email: cerdigufyu@gufum.com - ID invito: 11 **Accetta** **Rifiuta**

Chiudi

Java Spring

Spring è una libreria Java open source che semplifica lo sviluppo di applicazioni web e microservizi. Fornisce un'infrastruttura preconfigurata, riducendo il tempo di sviluppo e aumentando la produttività. Inoltre, Spring Boot si integra facilmente con altri framework di Spring, come Spring Data e Spring Security. Infine, supporta l'implementazione di applicazioni attraverso l'uso di container.

Spring Data JPA è una libreria di framework Spring che semplifica l'accesso ai dati attraverso l'Object Relational Mapping (ORM) in applicazioni basate su Java. In pratica, Spring Data JPA permette di definire le entità del database come classi Java, e di mapparle automaticamente alle tabelle del database utilizzando le annotazioni JPA. In questo modo, è possibile interagire con il database utilizzando metodi Java standard, senza dover scrivere query SQL manualmente.

Spring Java Mail Sender è una libreria di Spring Framework che semplifica l'invio di email in applicazioni Java. Utilizzando Java Mail API, Spring Java Mail Sender fornisce una serie di classi e metodi che permettono di configurare e inviare email in modo semplice e intuitivo. In particolare, Spring Java Mail Sender supporta l'autenticazione SMTP, la gestione degli allegati, l'HTML e il testo normale, la cifratura SSL e TLS e molto altro. Inoltre, Spring Java Mail Sender si integra facilmente con altri componenti di Spring Framework, come Spring Boot e Spring Security, semplificando ulteriormente lo sviluppo delle applicazioni che richiedono l'invio di e-mail.

Docker

Docker è una tecnologia open source di containerizzazione che consente di creare, distribuire e gestire applicazioni in modo consistente e portabile su diversi ambienti di esecuzione. In pratica, Docker consente di creare dei container isolati che contengono tutte le dipendenze e le librerie necessarie per l'esecuzione dell'applicazione, insieme al proprio codice. In questo modo, un'applicazione Docker può essere eseguita su qualsiasi sistema che supporti Docker, a prescindere dalle librerie o dalle configurazioni specifiche del sistema operativo sottostante.

ClearDB

ClearDB è un servizio di database basato su MySQL offerto da Oracle. Essenzialmente, ClearDB offre un'infrastruttura di database gestita basata su cloud, che consente agli sviluppatori di creare, gestire e scalare facilmente database MySQL su piattaforme cloud come Heroku, Azure e AWS. ClearDB offre diverse opzioni di piano, a seconda delle esigenze dell'utente, come il numero di connessioni simultanee e la dimensione del database. Inoltre, ClearDB offre backup automatizzati e ripristino del database, monitoraggio delle prestazioni e sicurezza avanzata, garantendo l'affidabilità e la disponibilità dei dati.

Apache Maven

Maven è uno strumento open-source di gestione delle dipendenze per le applicazioni Java, che semplifica la definizione e la gestione delle librerie esterne e delle versioni dei pacchetti. Inoltre, offre un'automatizzazione del processo di compilazione dell'applicazione e della gestione delle sue dipendenze.

Manuale di Installazione

L'installazione e la configurazione sono estremamente semplici.

Requisiti preliminari:

1. Installare Docker
2. Installare WSL
3. Avviare Docker

`GIT CLONE HTTPS://GITHUB.COM/TESTING-GAME-SAD-2023/T5-G5.GIT`

Aprire terminale e spostarsi nel path del repo con cd aggiungendo il percorso della cartella, eseguire in cmd:

`DOCKER BUILD -T FRONT .`

`DOCKER-COMPOSE UP`

Successivamente aprire un altro terminale ritornare nel path del repo ed eseguire:

`CD BACKEND`

`DOCKER BUILD -T BACK .`

`DOCKER-COMPOSE UP`

Nel caso spring desse errore stoppare con ctrl+c e rieseguire docker-compose up

Dockerfile – Relativo al back end

```
1 FROM maven:3.8.3-jdk-11-slim AS build
2
3 WORKDIR /app
4 # Copia il file JSON delle credenziali nell'immagine Docker
5
6
7
8
9 COPY pom.xml .
10
11 RUN mvn dependency:go-offline
12
13 COPY src ./src
14
15 RUN mvn package -DskipTests
16
17 FROM openjdk:11-jdk-slim
18
19 WORKDIR /app
20
21 COPY --from=build /app/target/*.jar app.jar
22
23 ENTRYPOINT ["java", "-jar", "app.jar"]
```

docker-compose.yml relativo al back end

```
version: '3.1'

services:
  db:
    image: mysql
    command: --default-authentication-plugin=mysql_native_password
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 'rootpassword'

    ports:
      - "3306:3306"
    expose:
      - "3306"
    volumes:
      - my-db:/var/lib/mysql
  adminer:
    image: adminer
    restart: always
    ports:
      - "9000:8080"

  springboot-app:

    build: .
    ports:
      - 8080:8080
    depends_on:
      - db

volumes:
  my-db:
```

Dockerfile – Relativo al front end

```
FROM node:alpine as development

# Imposta la directory di lavoro all'interno del container
WORKDIR /app

# Copia il file package.json nella directory di lavoro
COPY package*.json ./

# Esegue il comando npm install per installare le dipendenze
RUN npm install

# Copia il contenuto della directory build nell'immagine Docker
COPY ./build ./build

# Espone la porta 3000 del container
EXPOSE 3000

# Avvia l'applicazione React con il comando npm start
CMD ["npm", "start"]
```

docker-compose.yml Relativo al front end

```
1  version: "3.8"
2
3  services:
4    app:
5      container_name: app-dev
6      image: app-dev
7      build:
8        context: .
9        target: development
10     volumes:
11       - ./src:/app/src
12       - ./public:/app/public
13     ports:
14       - 3000:3000
15
16
```

Informazioni Aggiuntive

1. Il frontend è in esecuzione su `http://localhost:3000`
2. Su `http://localhost:9000` è presente un adminer per gestione database, per accedervi:

UTENTE: root

PASSWORD: rootpassword

3. I dati vengono inseriti sul database mysql

Sviluppi futuri: Multiplayer

Il seguente progetto permette al momento a due player di partecipare alla stessa partita. È previsto in futuro la possibilità di estendere il numero di giocatori e migliorare la relativa logica di multiplayer.