Documentazione progetto

Jose Raul Luizaga Yujra e Rizzi Emanuele

Matricole n. 1046611, e 1046324

Università degli Studi di Bergamo

Laurea Magistrale in Ingegneria Informatica

Corso di Informatica III

Modulo di Progettazione e Algoritmi (6 CFU)

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

Il nostro progetto nasce dalla necessità di gestire le attività all' interno di un circolo scacchistico, dove talvolta risulta difficile gestire il tutto a livello amministrativo. Le principali attività all'interno di un circolo scacchistico possono essere raggruppate come segue:

- Organizzazione di tornei all'interno del circolo;
- Organizzazione delle attività didattiche per gli iscritti;
- Organizzazione di eventi speciali;
- Archiviazione dei tornei;
- Gestione delle attività giornaliere;
- Associarsi;
- Tesseramento;

In particolare, il nostro caso di studio andrà a concentrarsi nell'interazione tra socio del circolo(utente) e amministratore. I problemi che sono stati riscontrati,durante l'analisi del sistema in esame, sono:

- Lato utente:
  - Avere la possibilità di preiscriversi al torneo;
  - Avere la possibilità di disiscriversi dal torneo in tempo;
  - Vedere in tempo reale la classifica dei punteggi di tutti i partecipanti al torneo di nostro interesse;
- Lato amministratore:
  - Gestire l'iscrizione al circolo scacchistico.
  - Avere la possibilità di creare un torneo;

- Avere anche la possibilità di annullare un torneo;

Il nostro scopo principale è quello di semplificare e risolvere tutte queste problematiche, attraverso l'utilizzo di una semplice applicazione, che gestisce i dati per entrambi gli utenti. Un altro obiettivo è quello di creare un'applicazione avente un interfaccia intuitiva e godibile a livello di grafica e di design.

2 Iterazione 0

2.1 Requirements engineering

In questa fase abbiamo svolto l'analisi dei requisiti partendo dalla metodologia di analisi dei casi d'uso, il quale permette di studiare i possibili scenari all'interno del problema. Il team ha svolto questa fase attraverso una sessione di brainstorming. Da ricordare che, per definizione di processo agile, le Use Case Stories descritte sono state utilizzate come input per le fasi successive dove sono state analizzate e raffinate con cura.

2.2 Panoramica del sistema

In questo progetto verrà implementata l'applicazione chiamata ***ChessClub*** che si occuperà della gestione delle attività organizzate all' interno di un circolo scacchistico. Le attività possono essere suddivise (anche per ordine di priorità) nel seguente modo:

1. Organizzazione e gestione dei tornei
2. Predisposizione delle attività didattiche
3. Gestione disponibilità delle sedi
4. Archiviazione dei tornei
5. Gestione Soci
6. Gestione procedure di tesseramento

Entrando nello specifico di queste attività...

2.2.1 Organizzazione Tornei

- torneo ufficiale
- torneo non ufficiale

2.2.2 Predisposizione attività didattiche

- Lezioni per principianti
- Lezioni per bambini
- Lezioni intermedie
- Lezioni private e/o di gruppo da GM (Gran Maestri di scacchi)
- Seminari da parte di veterani

2.2.3 Sedi disponibili

- Orario di apertura
- Posizione
- Via
- ecc...

2.2.4 Archiviazione tornei

- Alla fine di ogni torneo ufficiale (opzionale per i non ufficiali), tenere traccia delle partite tramite i fogli delle mosse in notazione scacchistica utilizzati durante il torneo. Seguendo il regolamento della FIDE(Federazione Internazionale Degli Scacchi).

2.2.5 Associarsi

Associarsi al circolo ha un prezzo in base al tipo di ruolo che si vuole ricoprire all'interno del circolo:

- Socio ordinario
- Socio sostenitore

- Socio giovane

2.2.6 Tesseramento

Possibile soltanto se si è diventati soci ed anche in questo caso ha un costo che varia in base all'età e al tipo di competizione a cui l'utente vuole partecipare:

- Tessera Agonistica
- Tessera Ordinaria
- Tessera Ordinaria ridotta
- Tessera Junior

2.2.7 Altre possibili idee implementative

- Attività soci tra cui discussione attività, bilanci ecc...
- Quiz giornalieri ()
- Partita del giorno e/o del torneo

2.3 Vincoli

- Sistema realizzato come installazione unica, per funzionare all'interno di un circolo scacchistico che può avere più sedi. Sistema pensato per essere utilizzato soprattutto tra i soci di un circolo scacchistico, ma questo non implica che altre persone al di fuori del circolo non possano usufruire del sistema.
- Operazioni svolte dai soci del circolo e amministratori, quindi ho due attori principali

2.4 Requisiti Funzionali

La numerazione dei requisiti è stata numerata in base alle priorità implementative..

2.4.1 Requisiti Funzionali lato Utente

1. Il sistema deve permettere l'iscrizione al sistema
2. Il sistema deve permettere la cancellazione dell'iscrizione al sistema
3. Il sistema deve permettere il log-in
4. Il sistema deve permettere il log-out
5. Il sistema deve permettere preiscrizione ad un torneo
6. Il sistema deve permettere la cancellazione di una preiscrizione ad un torneo
7. Il sistema deve permettere la visualizzazione delle informazioni (chi siamo,dove siamo,numero telefonico, ecc...)
8. Il sistema deve permettere l'iscrizione alle attività didattiche
9. Il sistema deve permettere la cancellazione dell'iscrizione alla attività didattica
10. Il sistema deve permettere l'associarsi al circolo
11. Il sistema deve permettere il tesseramento alla Federazione Scacchistica Italiana (FSI)
12. Il sistema deve permettere la ricezione delle notifiche

2.4.2 Requisiti Funzionali lato Amministratore

1. Il sistema deve permettere il log-in
2. Il sistema deve permettere il log-out
3. Il sistema deve permettere la creazione di un torneo
4. Il sistema deve permettere l'annullamento di un torneo
5. il sistema deve permettere l'inserimento dei partecipanti ad un torneo

6. Il sistema deve permettere l'inserimento dei punteggi dei turni di un torneo
7. Il sistema deve permettere la modifica delle informazioni riguardanti un torneo
8. Il sistema genera il nuovo turno del torneo in base ai punteggi inseriti dall'amministratore.
9. Il sistema deve permettere l'inserimento in archivio delle partite giocate ai tornei
10. Il sistema deve permettere la consultazione dell'archivio
11. Il sistema deve permettere l'invio di avvisi di qualunque tipo
12. Il sistema deve permettere la creazione di una sessione di attività didattiche
13. Il sistema deve permettere l'annullamento di una sessione di attività didattiche
14. Il sistema deve permettere la visualizzazione dei soci
15. Il sistema deve permettere la visualizzazione dei tesserati

2.5 Casi d'uso: LATO UTENTE

ID Caso d'uso	Titolo
UC1	Registrazione utente
UC2	Cancellazione Registrazione
UC3	Log-In
UC4	Log-out
UC5	Preiscrizione torneo
UC6	Cancellazione Preiscrizione
UC7	Visualizzazione Informazioni
UC8	Iscrizione attività didattiche
UC9	Cancellazione Iscrizione attività didattiche
UC10	Associarsi al circolo
UC11	Tesseramento

Tabella 1: Casi d'uso utente

2.5.1 UC1: Iscrizione al sistema

UC1	Iscrizione al sistema
1	L'utente entra nella sezione iscrizione
2	Inserisce le proprie credenziali
3	Conferma l'iscrizione

Tabella 2: Caso d'uso **UC1**

2.5.2 UC2: Cancella iscrizione

UC2	Cancellazione iscrizione al sistema
1	L'utente ISCRITTO entra nella sezione account
2	Clicca su elimina account
3	Conferma l'eliminazione dell'account

Tabella 3: Caso d'uso **UC2**

2.5.3 UC3: Log-In

UC3	Log-In
1	L'utente va nella pagina di Log-in
2	L'utente inserisce le proprie credenziali
3	L'utente ISCRITTO schiaccia log-in

Tabella 4: Caso d'uso **UC3**

2.5.4 UC4: Log-Out

UC4	Log-Out
1	L'utente Iscritto e Loggato schiaccia su Log-Out

Tabella 5: Caso d'uso **UC4**

2.5.5 UC5: Iscrizione torneo

UC5	Pre-Iscrizione torneo
1	L'utente entra nella sezione tornei
2	Selezione il torneo disponibile a cui vuole partecipare
3	Conferma pre-iscrizione al torneo

Tabella 6: Caso d'uso **UC5**

2.5.6 UC6: Cancellazione iscrizione torneo

UC6	Eliminazione pre-iscrizione al torneo
1	L'utente entra nella sezione tornei
2	Selezione il torneo disponibile a cui vuole cancellare la partecipazione
3	Conferma cancellazione pre-iscrizione al torneo

Tabella 7: Caso d'uso **UC6**

2.5.7 UC7: Visualizzazione informazioni

UC7	Visualizzazione delle informazioni
1	L'utente entra nella sezione informazioni
2	L'utente osserva le informazioni

Tabella 8: Caso d'uso **UC7**

2.5.8 UC8: Iscrizione attività didattica

UC8	Iscrizione attività didattiche
1	L'utente entra nella sezione attività didattiche
2	Scelta delle tipologia di attività da svolgere
3	Conferma l'iscrizione all'attività

Tabella 9: Caso d'uso **UC8**

2.5.9 UC9: Cancellazione iscrizione attività didattica

UC9	Eliminazione iscrizione attività didattiche
1	L'utente entra nella sezione attività didattiche
2	Scelta annullamento partecipazione all'attività
3	Conferma la cancellazione partecipazione dell'attività

Tabella 10: Caso d'uso **UC9**

2.5.10 UC10: Associazione al circolo

UC10	Associazione al circolo
1	L'utente ISCRITTO entra nella sezione soci
2	Scelta tipologia di associazione
3	conferma invio richiesta di associazione

Tabella 11: Caso d'uso **UC10**

2.5.11 UC11: Tesseramento

UC11	Tesseramento
1	L'utente ISCRITTO e ASSOCIATO entra nella sezione tessere
2	Scelta tipologia di tesseramento
3	conferma inizio richiesta di tesseramento

Tabella 12: Caso d'uso **UC11**

2.6 Vista casi d'uso: Utente

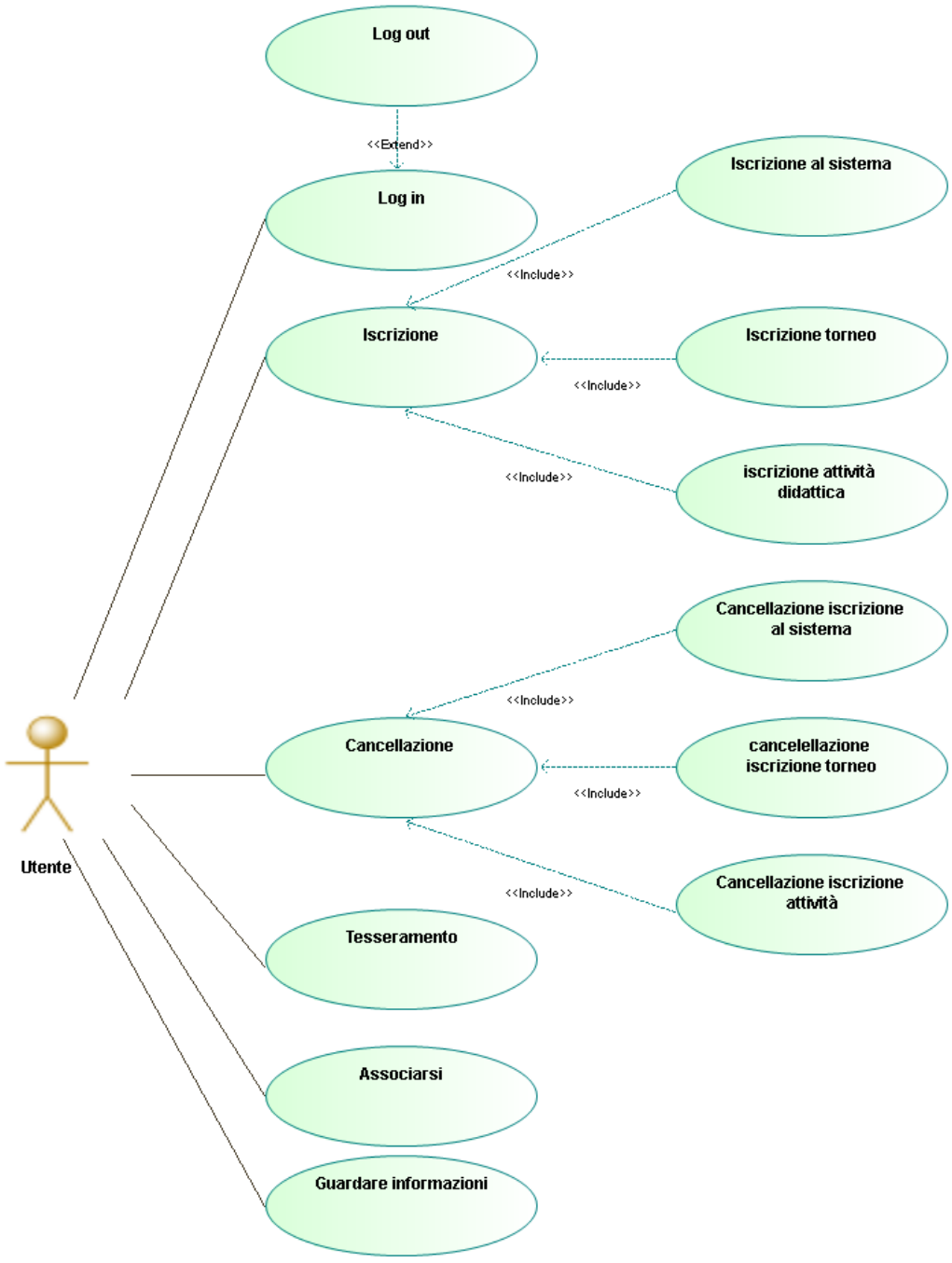

Figura 1: Casi d'uso Utente

2.7 Casi d'uso: LATO AMMINISTRATORE

ID Caso d'uso	Titolo
UC12	Log-In
UC13	Log-Out
UC14	Creazione torneo
UC15	Cancellazione torneo
UC16	Modifica torneo
UC17	Inserimento giocatori al torneo
UC18	Inserimento punteggio torneo
UC19	Generazione turni torneo
UC20	Archiviazione partite torneo
UC21	Creazione attività didattiche
UC22	Cancellazione attività didattiche
UC23	Invio avvisi agli utenti
UC24	Visualizzazione soci
UC25	Visualizzazione tesserati
UC26	Visualizzazione lista tornei

Tabella 13: Casi d'uso amministratore

2.7.1 UC12: Log-in

UC12	Log-in
1	L'utente ISCRITTO schiaccia log-in
2	inserimento delle credenziali

Tabella 14: Caso d'uso **UC12**

2.7.2 UC13: Log-out

UC13	Log-out
1	L'utente ISCRITTO e LOGGATO schiaccia log-out

Tabella 15: Caso d'uso **UC13**

2.7.3 UC14: Creazione torneo

UC14	Creazione torneo
1	Amministratore entra nella sezione tornei
2	Amministratore clicca nella sezione crea
3	Amministratore inserisce data,luogo, ecc..
4	Amministratore inserisce tipologia di torneo
5	conferma creazione torneo

Tabella 16: Caso d'uso **UC14**

2.7.4 UC15: Cancellazione torneo

UC15	Annullamento torneo
1	Amministratore entra nella sezione tornei
2	Amministratore clicca nella sezione elimina
3	Amministratore seleziona il torneo da eliminare
4	Amministratore conferma eliminazione torneo

Tabella 17: Caso d'uso **UC15**

2.7.5 UC16: Modifica torneo

UC16	Modifica torneo
1	Amministratore entra nella sezione tornei
2	Amministratore clicca nella sezione modifica
3	Amministratore seleziona il/i parametro/i da modificare
4	Amministratore conferma la modifica del torneo

Tabella 18: Caso d'uso **UC16**

2.7.6 UC17: Inserimento giocatori nel torneo

UC17	Inserimento giocatori al torneo
1	Amministratore entra nella sezione tornei
2	Amministratore clicca nella sezione inserisci giocatore
3	Amministratore sceglie il giocatore da inserire
4	Amministratore conferma inserimento giocatore

Tabella 19: Caso d'uso **UC17**

2.7.7 UC18: Inserimento punteggi torneo

UC18	Inserimento punteggi torneo
1	Amministratore entra nella sezione torneo
2	Amministratore clicca sul torneo in cui è presente la partita che vuole modificare
3	Amministratore seleziona la partita da modificare
4	Amministratore inserisce il risultato della partita
5	Amministratore conferma l'inserimento del risultato

Tabella 20: Caso d'uso **UC18**

2.7.8 UC19: Generazione turni torneo

UC19	Generazione turni torneo
1	Amministratore entra nella sezione torneo
2	Amministratore clicca sul torneo in cui vuole generare il turno
3	Amministratore clicca sul bottone cambia turno

Tabella 21: Caso d'uso **UC19**

2.7.9 UC20: Archiviazione partite torneo

UC20	Archiviazione partite torneo
1	Amministratore entra nella sezione archivio
2	Amministratore clicca nella sezione inserisci partita
3	Amministratore inserisce manualmente le mosse riempendo alcuni campi
4	Amministratore conferma le mosse inserite

Tabella 22: Caso d'uso **UC20**

2.7.10 UC21: Creazione attività didattiche

UC21	Creazione attività didattiche
1	Amministratore entra nelle sezione Attività
2	Amministratore clicca nella sezione crea attività
3	Amministratore inserisce data,luogo, ecc..
4	Amministratore inserisce tipologia di attività
5	Amministratore conferma creazione attività

Tabella 23: Caso d'uso **UC21**

2.7.11 UC22: Cancellazione attività didattiche

UC22	Cancellazione attività didattiche
1	Amministratore entra nelle sezione Attività
2	Amministratore clicca nella sezione distruggi attività
3	Amministratore selezione la attività da distruggere
4	conferma distruzione della attività

Tabella 24: Caso d'uso **UC22**

2.7.12 UC23: Invio avvisi

UC23	Invio avvisi
1	Amministratore entra nella sezione avvisi
2	Amministratore scrive un messaggio
3	Amministratore sceglie i destinatari
4	Amministratore programma l'invio del messaggio
5	Amministratore conferma l'invio dell'avviso

Tabella 25: Caso d'uso **UC23**

2.7.13 UC24: Visualizzazione soci

UC24	Visualizzazione soci
1	L'amministratore entra nella sezione soci
2	Amministratore visualizza la lista dei soci

Tabella 26: Caso d'uso **UC24**

2.7.14 UC25: Visualizzazione tesserati

UC25	Visualizzazione soci
1	L'amministratore entra nella sezione tesserati
2	Amministratore visualizza la lista dei tesserati

Tabella 27: Caso d'uso **UC25**

2.7.15 UC26: Visualizzazione lista tornei

UC26	Visualizzazione tornei
1	L'amministratore entra nella sezione tornei
2	Amministratore visualizza la lista dei tornei

Tabella 28: Caso d'uso **UC26**

2.8 Vista casi d'uso: Amministratore

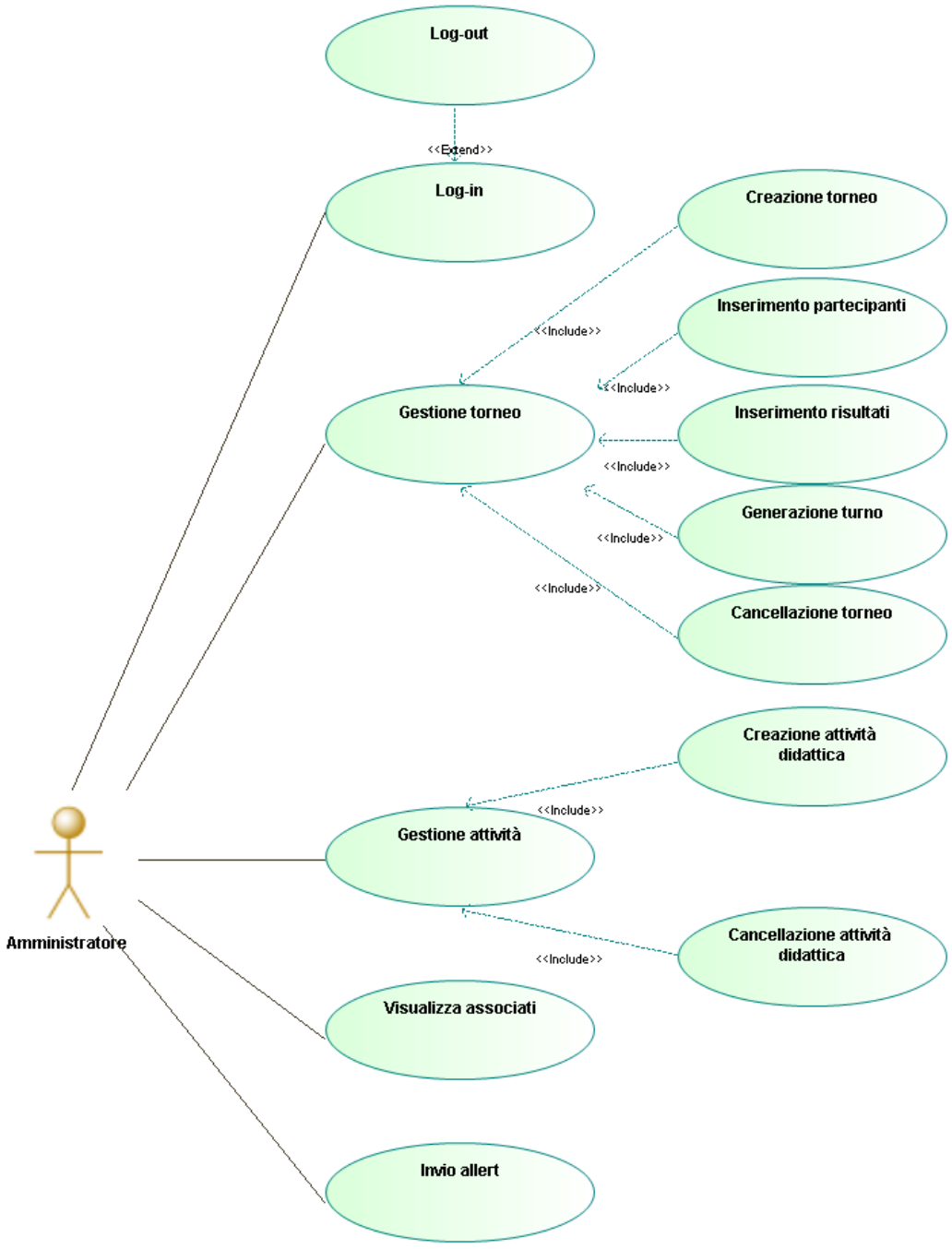

Figura 2: Casi d'uso Amministratore

2.9 Vista generale dei casi d'uso

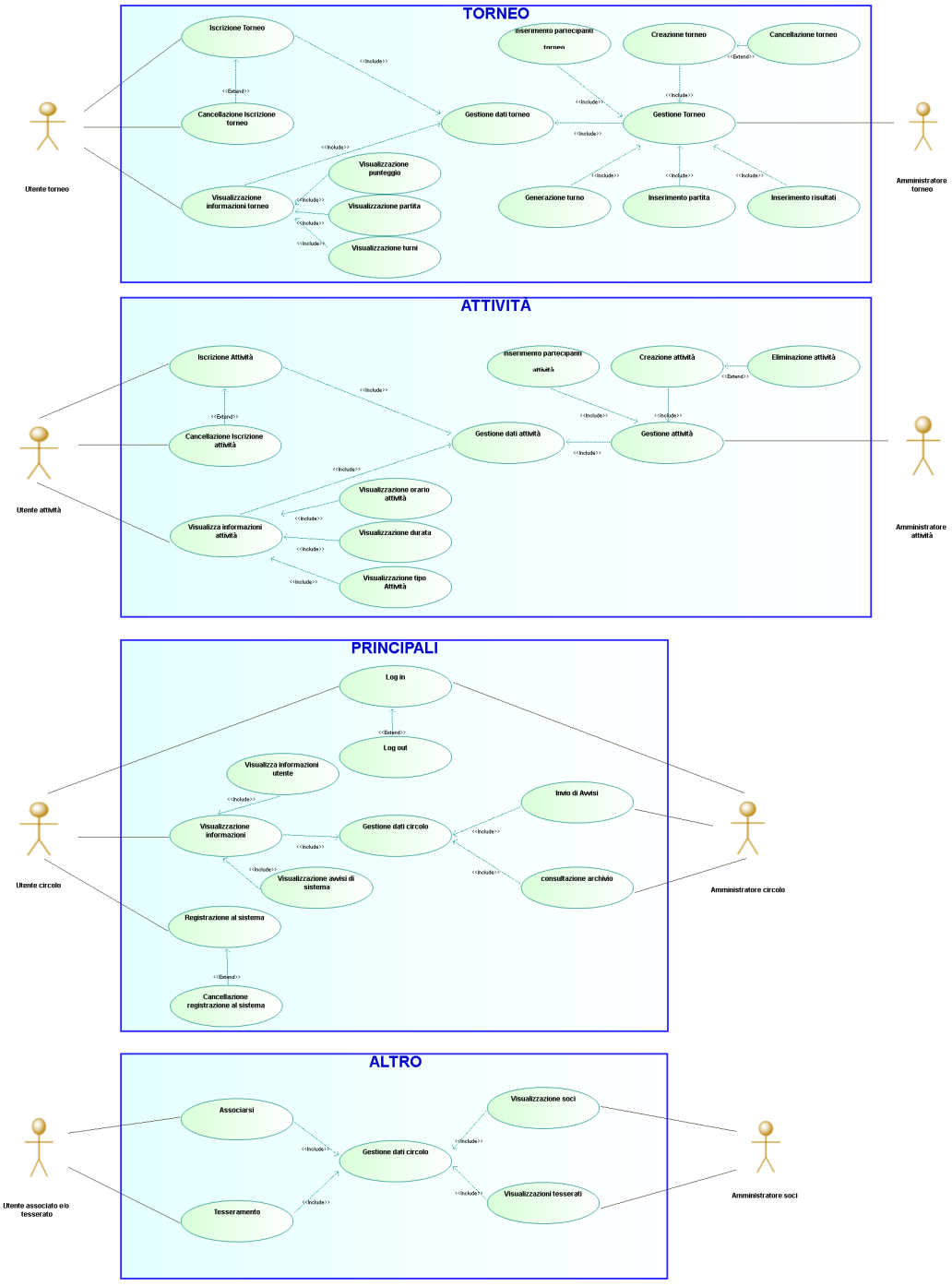

Figura 3: Casi d'uso sistema

2.10 Requisiti Non Funzionali

Il progetto verrà sviluppato tenendo in considerazione alcuni requisiti non funzionali, quali la ***manutenibilità***, ***l'efficienza***, ***l'usabilità*** e ***la riusabilità***.

2.10.1 Manutenibilità

Il requisito di manutenibilità verrà rispettato mediante l'implementazione di tutte le risorse necessarie e persistenti, ovvero: dati del giocatore e dell'amministratore,iscrizioni ai tornei,iscrizioni alle attività del circolo e le informazioni riguardanti il circolo. In questo modo, anche se in futuro si dovesse aggiornare l'applicazione con degli ingenti cambiamenti, sarebbe facile sia per l'amministratore che per il giocatore ritrovarle all'interno dell'applicazione.

2.10.2 Efficienza

Il requisito dell'efficienza è anche l'obiettivo primario del progetto, ovvero la creazione di un algoritmo che permetta durante lo svolgimento di un torneo di generare i turni in base al punteggio corrente dei partecipanti.

2.10.3 Usabilità

L'usabilità è garantita dalla decisione di sviluppare l'applicazione in Android, di facile utilizzo per tutti gli utenti, siano essi giocatori e amministratori, di qualsiasi livello di esperienza informatica. Per ottenere ciò, sarà di vitale importanza la creazione di un'interfaccia pratica ed intuitiva. In base alla topologia, è possibile notare che il database verrà ospitato da un hosting online e quindi sarà sempre possibile, mediante qualsiasi dispositivo connesso alla rete Internet, accedere a tutte le funzionalità dell'applicazione.

2.10.4 Riusabilità

La riusabilità è garantita dal fatto che utilizzando una architettura *N-Tier*, ogni layer può essere rimpiazzato con uno nuovo, senza alcun problema.

2.11 Topologia del Sistema

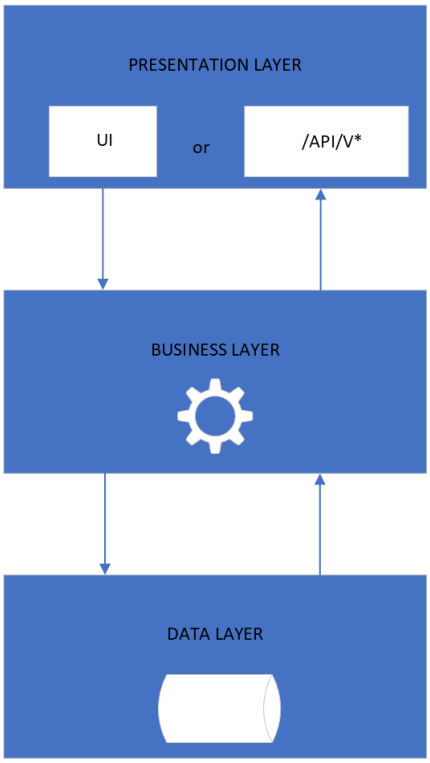

Figura 4: Topologia sistema

Abbiamo pensato ad un'architettura dove fosse possibile riutilizzare componenti in modo semplice(**riusabilità**)

2.11.1 Architettura N-Tier

Ci permette di separare il codice in *layers*.

2.11.2 Presentation layer

Solitamente può essere il mio *user interface* in react o angular(oppure altri framework), ma nel nostro caso è una *Restful API*.Questo *layer* è il principale *entry point* per la nostra applicazione. Il *layer presentation* prende la richiesta che proviene dal *client* (questa è la sua funzione principale). Successivamente la richiesta passa al *Business layer*.

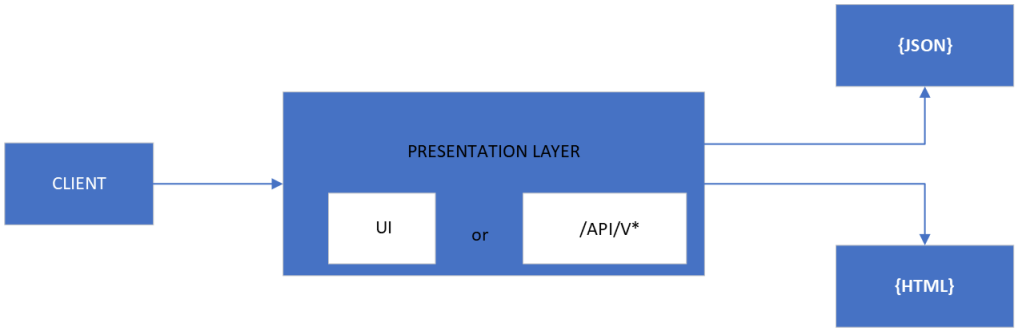

Figura 5: Presentation Layer

Nella figura 5 possiamo osservare il seguente processo:

1. *Client* manda una *request* all'applicazione
2. La *request* può essere un *API REST*
3. Produzione di qualcosa come ad esempio file *JSON* o *HTML*

Quindi ricapitolando si prende la richiesta dai *client*, solitamente l'applicazione espone una serie di *end-point*, quindi utilizzo la semantica tipica del *HTTP*: *POST*, *GET*, *PUT*, *DELETE*.

2.11.3 Business layer

Layer che si occupa principalmente della *business logic* (logica che rende operativa un'applicazione), quindi l'algoritmica che gestisce lo scambio di informazioni tra *UI* attraverso il *Presentation layer* con le elaborazioni intermedie sui dati estratti ed eventualmente una sorgente dati.

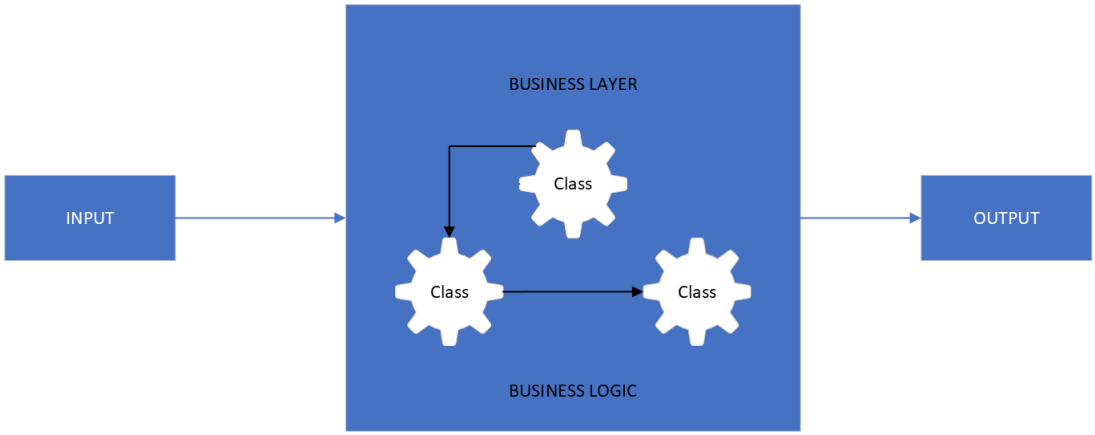

Figura 6: Business Layer

Nella figura 6 l'input può essere per esempio un *DTO*, ovvero un oggetto che porta dati tra processi

2.11.4 Data layer

Layer che si occupa principalmente del ruolo di interagire con qualsiasi *Data Source*

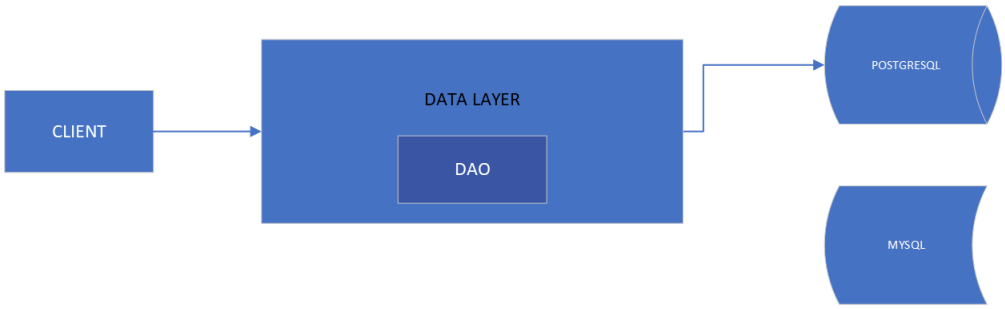

Figura 7: Data Layer

Nella figura 7:

- *Client*: sono classi che vogliono i dati
- *Data Layer*: interfaccia dove definisco il contratto e nel nostro caso abbiamo una implementazione che è *Postgresql*

- *DAO*: Data Access Object

Nel nostro caso abbiamo pensato di utilizzare *Postgresql*, ma questo non esclude la possibilità di utilizzare altri tipi di database come *Mysql*.

2.12 Design Pattern MVP

L'intero sistema ed i suoi componenti verrano sviluppati utilizzando il design pattern *Model View Presenter*, il quale è una derivazione del Model View Controller (*MVC*). Come si può intuire dal nome di questo design è strutturato in 3 layer:

1. **Model**: è lo strato di Data Access per la gestione dei dati. Può essere visto come una interfaccia che è responsabile di accedere alle API connesse con un database locale oppure una cache.

2. **View**: lavora con il *Presenter* per mostrare i dati e notificare le azioni dell'utente, solitamente non ha nessuna logica applicativa, ma solo visuale. Nel caso di applicazioni multi piattaforma (Web, Mobile, Desktop), possiamo avere una singola interfaccia per la *View* e multiple implementazioni.

3. **Presenter**: lavora come intermediario tra la *View* e il *Model*. Prende i dati dal *Model*, li elabora e li restituisce alla *View*

2.13 Toolchain e tecnologie utilizzate

2.13.1 Modello Agile

L'utilizzo della metodologia agile è stata di fondamentale importanza per lo sviluppo del software. Infatti, con questo metodo siamo riusciti a gestire tempo e risorse che con altre metodologie non saremmo riusciti a controllare. I quattro valori agili importanti sono:

- Persone e interazioni sono più importanti del processo e dei tools utilizzati.

- Un software funzionante è più importante di una chiara documentazione.

- La collaborazione del cliente va aldilà del contratto stipulato.
- La capacità di rispondere a cambiamenti è una caratteristica fondamentale. Può succedere spesso che vengano imposte delle modifiche al prodotto: bisogna adottare un processo in grado di apportare continui cambiamenti a ciò che stiamo realizzando.

2.13.2 Toolchain

Per la realizzazione del software verranno utilizzati i seguenti strumenti:

Tool/Tecnologia	**Utilizzo**
Java 18.0.1	Linguaggio di programmazione del back-end
Eclipse	IDE per sviluppo back-end
Android Studio	IDE per sviluppo front-end
Spring Boot	Java Framework per la realizzazione del back-end
Git e GitHub	Software e piattaforma per gestire i vari update dell'applicazione e condivisione di codice e documentazione
Google Drive	Cloud Storage per condivisione di documenti e immagini
Modelio,Visio e diagram.net	Tool per realizzazione di diagrammi
Postman	Piattaforma per test dinamico e documentazione delle API
Postgresql	DataBase
H2	DataBase per il testing
Overleaf	Software per la stesura e formattazione della documentazione in linguaggio LaTeX

Tabella 29: Toolchain e tecnologie utilizzate

3 Iterazione 1

3.1 Introduzione

Nella prima iterazione si è scelto di implementare i seguenti casi d'uso:

- Gestione pagina di login:
  - **UC1**: Registrazione
  - **UC3/UC12**: Log-in
  - **UC4/UC13**: Log-out
  - **UCx**: Password Dimenticata
- Gestione attività torneo:
  - **UC14**: Creazione Torneo
  - **UC26**: Visualizzazione lista Tornei
  - **UC15**: Cancellazione Torneo
  - **UC16**: Modifica Torneo

  Da notare che nella lista precedente è presente il caso d'uso **UCx**, il motivo della presenza di questo nuovo caso è dovuto alla riunione, svoltasi in data *17 novembre 2022*. In questa riunione, il gruppo ha individuato la necessità di introdurre questo nuovo caso d'uso. Tale caso d'uso è stato rilevato in base alla divisione dei compiti tra i due componenti del gruppo. Infatti il gruppo ha deciso di procedere in parallelo con il *backend* ed il *frontend*, di conseguenza durante il design del front-end è uscita questa problematica ed è stata risolta in questo modo.

3.2 UC1: Registrazione

Breve descrizione: l'utente che ha scaricato l'applicazione, quindi è interessato ad iscriversi, visualizzerà la pagina principale ed entrerà nella sezione di registrazione.

Precondizione:

- Aver scaricato l'applicazione

Credenziali necessarie per la registrazione:

- Nome
- Cognome
- Email
- Password
- Conferma Password

Attori coinvolti :

- Utente
- Sistema

Risposta del sistema:

- Se tutti i campi sono stati inseriti correttamente:
  - *successo*: registrazione effettuata, invio della mail di avvenuta registrazione;
  - *fallimento*: risposta tramite mail, dovuto all'inserimento di una mail già esistente;
- Se alcuni campi non sono stati riempiti,oppure se i campi *password* e *conferma password* non coincidono, verrà mostrato un messaggio di errore nell'applicazione.

PostCondizione: l'utente, dopo la registrazione, viene reindirizzato nella pagina principale di *login*, dove dovrà inserire le credenziali utilizzate durante l'operazione di registrazione.
Procedimento :

1. Click sulla sezione di registrazione
2. Inserimento di *nome, cognome, mail, password, conferma password*
3. Conferma registrazione
4. Reindirizzamento alla pagina di *login*
5. Visualizzazione mail

3.3 UC3/UC12: Log-in

Breve descrizione: l'utente che ha scaricato l'applicazione ed ha effettuato l'operazione di *registrazione*, ora può svolgere l'operazione di *log-in* per accedere ai servizi dell'applicazione come: *iscrizione al torneo*, *cancellazione dell'iscrizione al torneo* , *visualizzazione del profilo* , *visualizzazione del torneo in corso* ed effettuare il *log-out*.
Pre-condizione:

- Aver scaricato l'applicazione
- Aver effettuato con successo l'operazione di registrazione

Credenziali necessarie per l'operazione di log-in:

- Email
- Password

Attori coinvolti:

- Utente
- Sistema

Risposta del sistema:

- *successo*:

  - Se il log-in è eseguito da amministratore, si viene reindirizzati nella *Administrator-page*
  - Se il log-in è eseguito da utente del circolo, si viene reindirizzati nella *User-page*

- *fallimento*: viene mostrato un messaggio di log-in fallito

PostCondizione: l'utente/amministratore, dopo il log-in, viene reindirizzato nella *User-page* o nella *Administrator-page*, dove avrà accesso alle funzionalità dell'applicazione per gestire le attività all'interno del circolo
Procedimento :

1. Click sulla sezione *log-in*
2. Inserimento *email* e *password*
3. Conferma *log-in*
4. Reindirizzamento alla *User-page*/*Administrator-page*
5. Visualizzazione servizi applicazione

3.4 UC4/UC13: Log-out

Breve descrizione: l'utente che ha effettuato il *log-in* per uscire dalla applicazione deve effettuare il *log-out*.
PreCondizione:

- Aver scaricato l'applicazione
- Aver effettuato il log-in, quindi deve anche essere registrato

Attori coinvolti:

- Utente

- Sistema

Risposta del sistema:
Se si è effettuato il *log-out*, il sistema reindirizzerà l'utente alla pagina di *log-in*.
PostCondizione: L'utente, dopo il *log-out*, sarà reindirizzato alla pagina di *log-in*.
Procedimento:

1. Click sulla sezione *log-out*
2. Utente reindirizzato nella pagina di *log-in*
3. Utente visualizza la pagina di *log-in*

3.5 UCx: Password Dimenticata

Breve descrizione: l'utente che non si ricorda più la *password* che ha inserito durante l'operazione di *registrazione*, entra nella sezione *password dimenticata*. All'interno di questa sezione, sarà richiesta l'e-mail di registrazione. L'e-mail serve per inviare un codice che poi l'utente dovrà incollare nell'applicazione per rimpiazzare la *password* vecchia con una nuova che verrà scelta in quel momento.
PreCondizione:

- Aver scaricato l'applicazione
- Essere registrato

Attori coinvolti:

- Utente
- Sistema

Risposta del sistema:

- Se l'e-mail inserita, esiste all'interno del circolo, allora il sistema invia una e-mail con un codice all'utente

- Il sistema reindirizza l'utente ad una nuova pagina, dove dovrà inserire il codice
- Se il codice incollato è uguale a quello inviato dal sistema, allora il sistema reindirizza ad una nuova pagina
- La nuova pagina ha dei campi da riempire che sono *Nuova Password* e *Conferma nuova Password*

PostCondizione: Dopo aver eseguito tutti i passi con successo, il sistema reindirizza *l'utente* alla pagina di *log-in*, dove il nuovo utente potrà testare la nuova *password*.
Procedimento:

1. Click sulla sezione *password dimenticata*
2. Inserimento e-mail
3. Visualizzazione posta elettronica
4. Copia codice ricevuto tramite mail dal sistema
5. Incolla codice nell'applicazione
6. Inserimento nuova password
7. Conferma nuova password
8. Prova nuove credenziali

3.6 UC14: Creazione Torneo

Breve descrizione: L'amministratore entra nella sezione *Crea torneo* per effettuare l'operazione di creazione di un nuovo torneo. Per la creazione di un torneo sono necessarie una serie dati: *nome torneo, data , luogo, turni torneo.*
Precondizione:

- Aver scaricato l'applicazione

- Aver effettuato il log-in

Attori coinvolti:

- Amministratore
- Sistema

Risposta del sistema:

- *Successo:*
  - Se tutti i campi sono stati riempiti correttamente ed il nome del torneo non coincide con il nome di un torneo già esistente, allora avviene la creazione
- *Fallimento:*
  - Se il torneo inserito ha un nome identico ad un torneo già esistente, allora questo non viene creato
  - Se non sono stati riempiti tutti i campi, allora il torneo non viene creato

PostCondizione:

- Se la creazione è avvenuta con successo si viene reindirizzati nella *Administrator-page*
- Se la creazione è fallita, allora l'amministratore visualizzerà un avviso di fallimento della creazione del torneo

Procedimento:

- Amministratore esegue il log-in
- Amministratore visualizza *Administrator-page*
- Amministratore fa Click sulla sezione *crea torneo*

- Amministratore inserisce
  - *Nome torneo*
  - *Luogo torneo*
  - *Data torneo*
  - *Turni torneo*
- Se la creazione è avvenuta con successo, l'amministratore è reindirizzato nella *Administrator-page*
- Altrimenti visualizza l'avviso di fallimento creazione torneo

3.7 UC26: Visualizzazione lista tornei

Breve descrizione:
L'amministratore entra nella sezione *Visualizza tornei* per effettuare l'operazione di visualizzazione dei tornei. Per la visualizzazione dei tornei si è utilizzato una lista con tutti i tornei creati dall'amministratore. Quindi l'amministratore potrà visualizzare.
PreCondizione:

- Aver scaricato l'applicazione
- Aver effettuato il log-in
- Avere dei tornei già presenti nel sistema da poter visualizzare

Attori coinvolti:

- Amministratore
- Sistema

Risposta del sistema:

- *Successo:*

  - Tornei visualizzati con successo

- *Fallimento:*

  - Non sono presenti tornei che si possono visualizzare

PostCondizione:

- Se la visualizzazione è avvenuta con successo, si rimane nella stessa pagina
- Nel caso in cui non sono presenti tornei, si rimane nella stessa pagina

Procedimento:

- Amministratore esegue il log-in
- Amministratore visualizza *Administrator-page*
- Amministratore fa Click sulla sezione *visualizza tornei*
- Amministratore visualizza la lista dei tornei

3.8 UC15: Cancellazione Torneo

Breve descrizione:
L'amministratore entra nella sezione *Cancella torneo* per effettuare l'operazione di eliminazione torneo. Per l'eliminazione di un torneo verrà mostrata una lista con tutti i tornei creati dall'amministratore. Quindi l'amministratore dovrà scegliere quale torneo desidera eliminare.
Precondizione:

- Aver scaricato l'applicazione
- Aver effettuato il log-in
- Avere dei tornei già presenti nel sistema da poter eventualmente cancellare

Attori coinvolti:

- Amministratore
- Sistema

Risposta del sistema:

- *Successo:*
  - Torneo eliminato con successo
- *Fallimento:*
  - Non sono presenti tornei che si possano eliminare, quindi tale operazione non verrà eseguita
  - Il torneo è in corso e non è ancora finito, quindi l'operazione di cancellazione del torneo non potrà essere eseguita

PostCondizione:

- Se la cancellazione è avvenuta con successo, la lista tornei verrà aggiornata
- Nel caso in cui il torneo non è stato eliminato con successo, l'amministratore visualizzerà un messaggio che lo avviserà del fallimento di tale operazione

Procedimento:

- Amministratore esegue il log-in
- Amministratore visualizza *Administrator-page*
- Amministratore fa Click sulla sezione *elimina torneo*
- Amministratore visualizza la lista dei tornei
- Amministratore clicca sul torneo che desidera eliminare

- Amministratore conferma l'eliminazione
- Se la eliminazione è avvenuta con successo, l'amministratore sarà reindirizzato nella lista tornei
- Altrimenti visualizzerà l'avviso di fallimento eliminazione torneo

3.9 UC16: Modifica Torneo

Breve descrizione:
L'amministratore entra nella sezione *Modifica torneo* per effettuare l'operazione di modifica torneo. Per la modifica di un torneo verrà mostrata una lista con tutti i tornei creati dall'amministratore. Quindi l'amministratore dovrà scegliere quale torneo desidera modificare.
Precondizione:

- Aver scaricato l'applicazione
- Aver effettuato il log-in
- Avere dei tornei già presenti nel sistema da poter eventualmente modificare

Attori coinvolti:

- Amministratore
- Sistema

Risposta del sistema:

- *Successo:*
  - Torneo modificato con successo
- *Fallimento:*

  - Non sono presenti tornei che si possano modificare, quindi tale operazione non potrà essere eseguita

PostCondizione:

- Se la modifica è avvenuta con successo, la lista tornei verrà aggiornata
- Nel caso in cui il torneo non è stato modificato con successo, l'amministratore visualizzerà un messaggio che lo avviserà del fallimento di tale operazione

Procedimento:

- Amministratore esegui il log-in
- Amministratore visualizza *Administrator-page*
- Amministratore fa Click sulla sezione *modifica torneo*
- Amministratore visualizza la lista dei tornei
- Amministratore clicca sul torneo che desidera modificare
- Amministratore modifica i parametri del torneo che desidera modificare, che sono:
  - *Nome torneo*
  - *Luogo torneo*
  - *Data torneo*
  - *Turni torneo*
- Se la modifica è avvenuta con successo, l'amministratore sarà reindirizzato nella lista tornei
- Altrimenti visualizzerà l'avviso di fallimento modifica torneo

3.10 Entity-Relationship Diagram

Qui si presentano le entità e le loro relazioni.

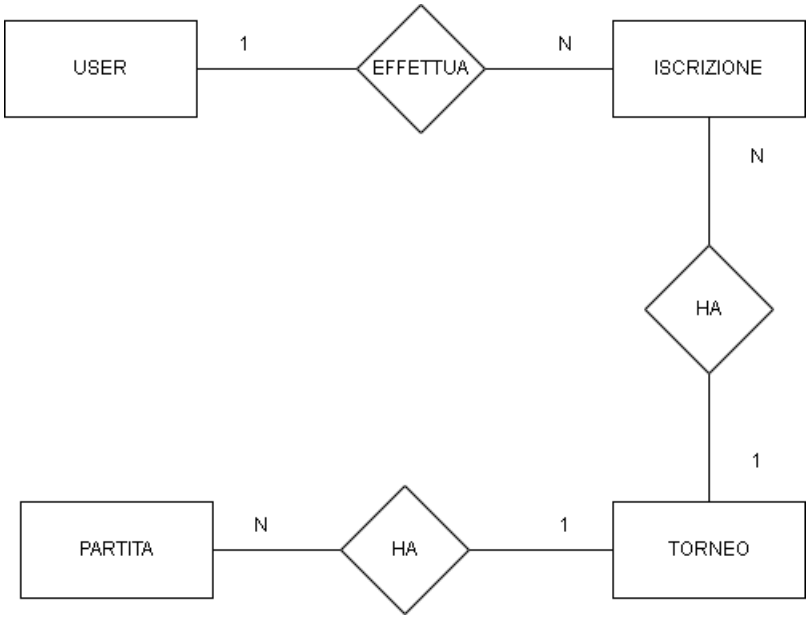

Figura 8: Entity-Relationship Diagram

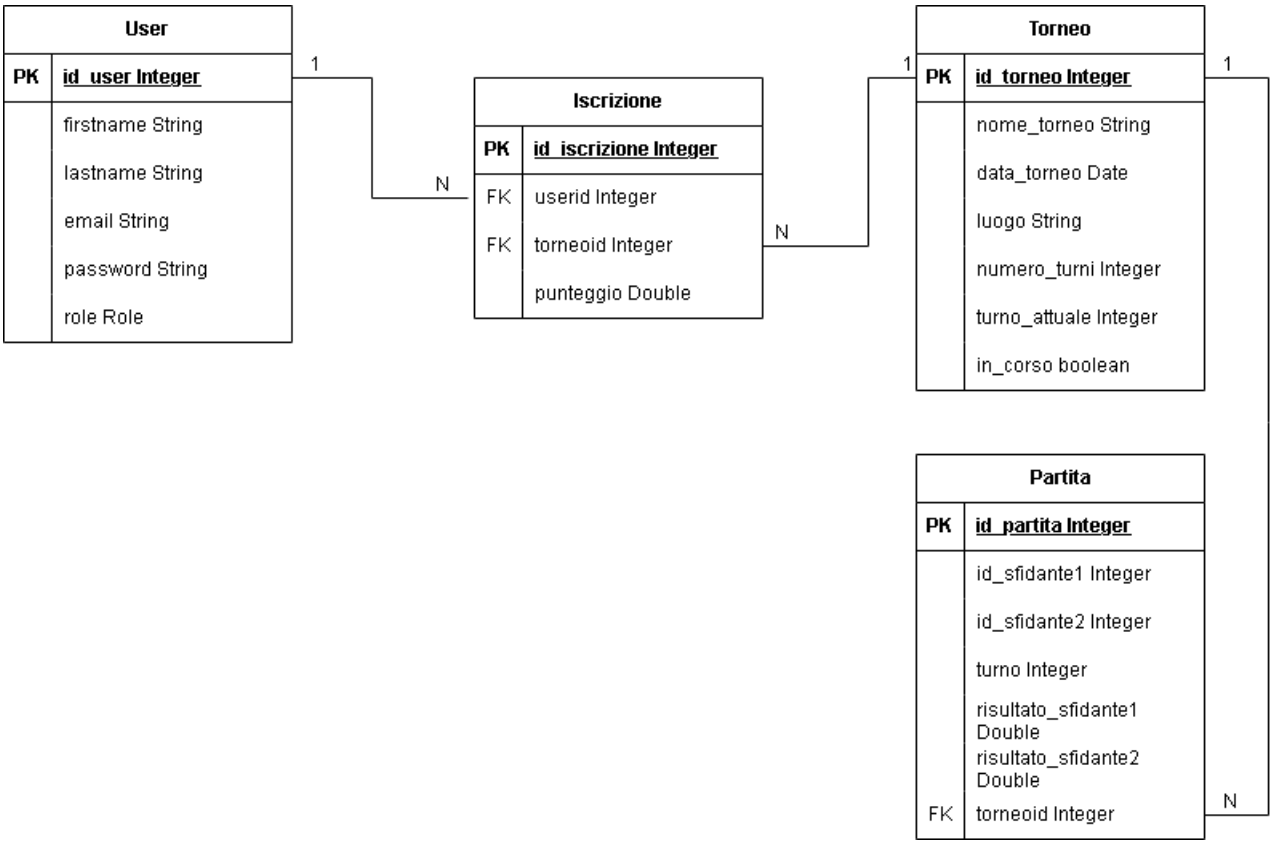

Figura 9: Entity-Relationship Diagram

3.11 UML Component Diagram

Qui è rappresentato il *Component Diagram* dell'applicazione all'iterazione 1. Siamo partiti rappresentando il sistema come unico macroelemento ad alto livello, che espone le interfacce tramite tecnologia *API REST*.

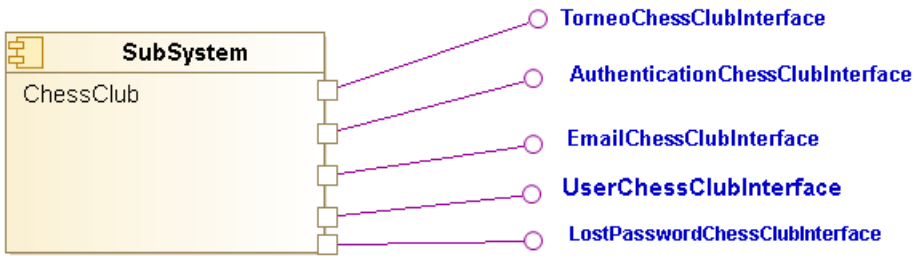

Figura 10: ClassDiagram1

I casi d'uso che abbiamo discusso nella iterazione1 sono state mappate in metodi resi disponibili dalle seguenti interfacce, nello specifico vediamo:

- *TorneoChessClubInterface*:
  - UC14: Creazione torneo
  - UC26: Visualizzazione lista tornei
  - UC15: Cancellazione torneo
  - UC16: Modifica torneo
- *AuthenticationChessClubInterface*:
  - UC1: Registrazione
  - UC3/UC12: Log-in
  - UC4/UC13: Log-out
  - UCx: Password dimenticata
- *EmailChessClubInterface*: Interfaccia utilizzata per l'invio della e-mail ed è utilizzata principalmente dall'interfaccia *AuthenticationChessClubInterface*.
- *UserChessClubInterface*: Interfaccia utilizzata per la creazione degli utenti dell'applicazione.
- *LostPasswordChessClubInterface*: Interfaccia utilizzata per effettuare il cambio password nel caso in cui l'utente se la sia dimenticata.

3.12 UML Class Diagram per interfacce lato server

Ora mettiamo in evidenza le interfacce del sistema con le relative funzioni e dati di input e di output.

3.12.1 AuthenticationChessClubInterface

Tramite questa interfaccia definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di:

- registrazione
- autenticazione
- permesso cambio password

Figura 11: AuthenticationChessClubInterface

3.12.2 TorneoChessClubInterface

Tramite questa interfaccia definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di gestione dei tornei.

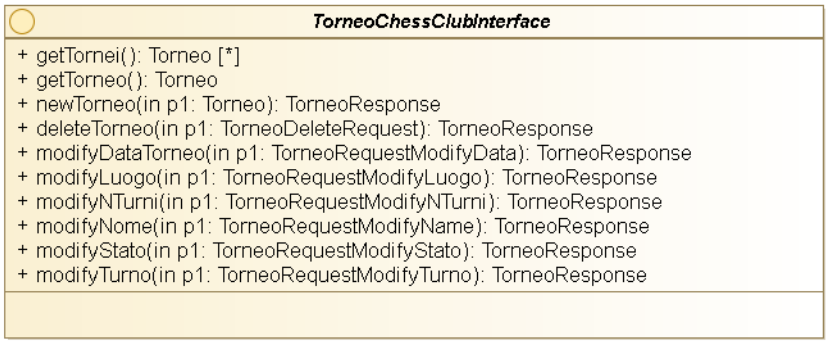

Figura 12: TorneoChessClubInterface

3.12.3 EmailChessClubInterface

Tramite questa interfaccia definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di invio delle email.

Figura 13: EmailChessClubInterface

3.12.4 UserChessClubInterface

Tramite questa interfaccia definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di gestione degli user.

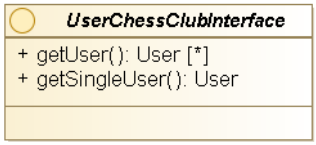

Figura 14: UserChessClubInterface

3.12.5 LostPasswordChessClubInterface

Tramite questa interfaccia definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di recupero password.

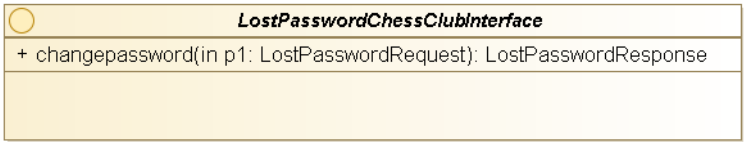

Figura 15: LostPasswordChessClubInterface

3.13 UML Class Diagram per tipi di dato lato server

Qui sono presenti i tipi di dato necessari per lo sviluppo dell'applicazione. Per la progettazione dei tipi e delle interfacce sono state seguite le euristiche di design.

3.13.1 Classi per Authentication

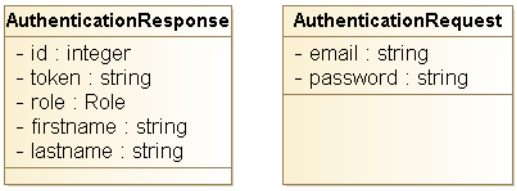

Figura 16: StrutturaDatiAuthentication

3.13.2 Classi per User

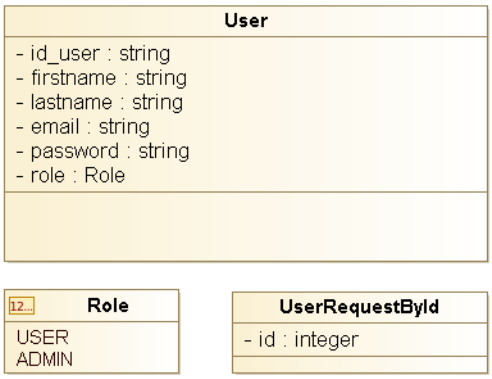

Figura 17: StrutturaDatiUser

3.13.3 Classi per Torneo

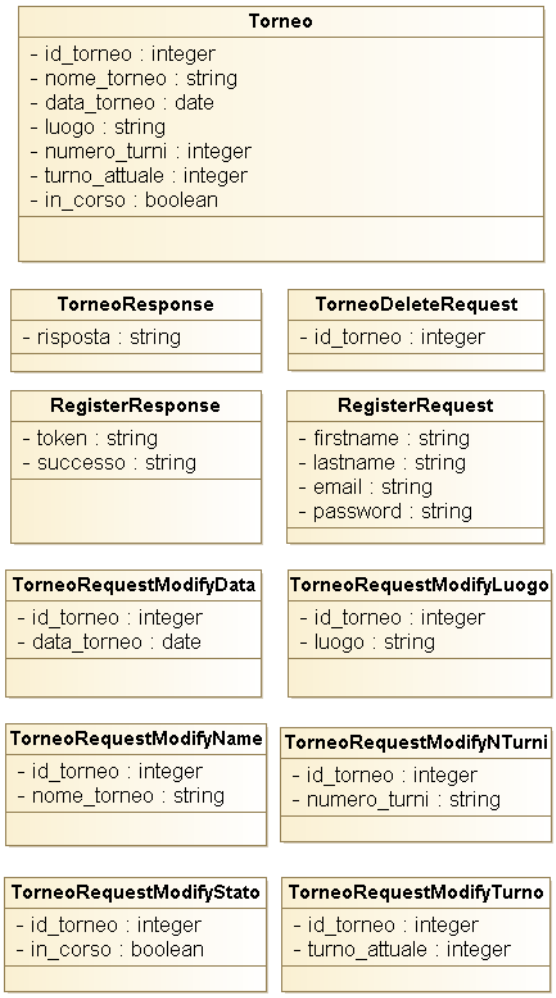

Figura 18: StrutturaDatiTorneo

3.13.4 Classi per Email

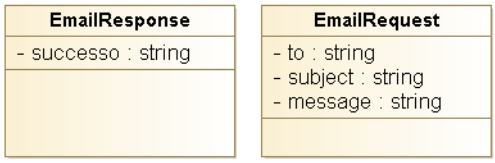

Figura 19: StrutturaDatiEmail

3.13.5 Classi per LostPassword

Figura 20: StrutturaDatiLostPassword

3.14 UML Class Diagram per interfacce lato client

Ora mettiamo in evidenza le interfacce lato client con le relative funzioni e dati di input e di output

3.14.1 AuthenticationAPICall

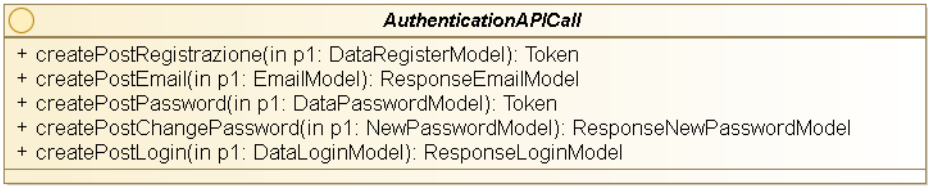

Figura 21: AuthenticationAPICall

3.14.2 AdministratorAPICall

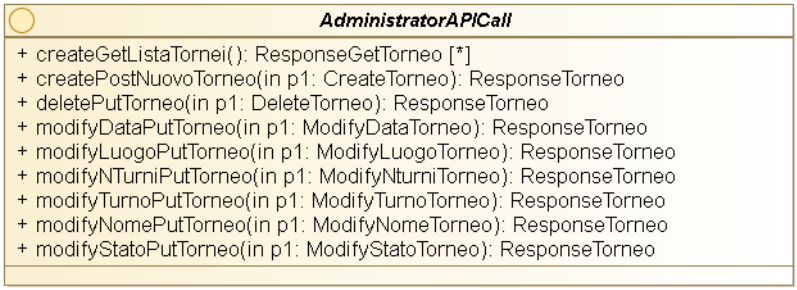

Figura 22: AdministratorAPICall

3.15 UML Class Diagram per tipi di dato lato client

3.15.1 Classi per AuthenticationAPICall

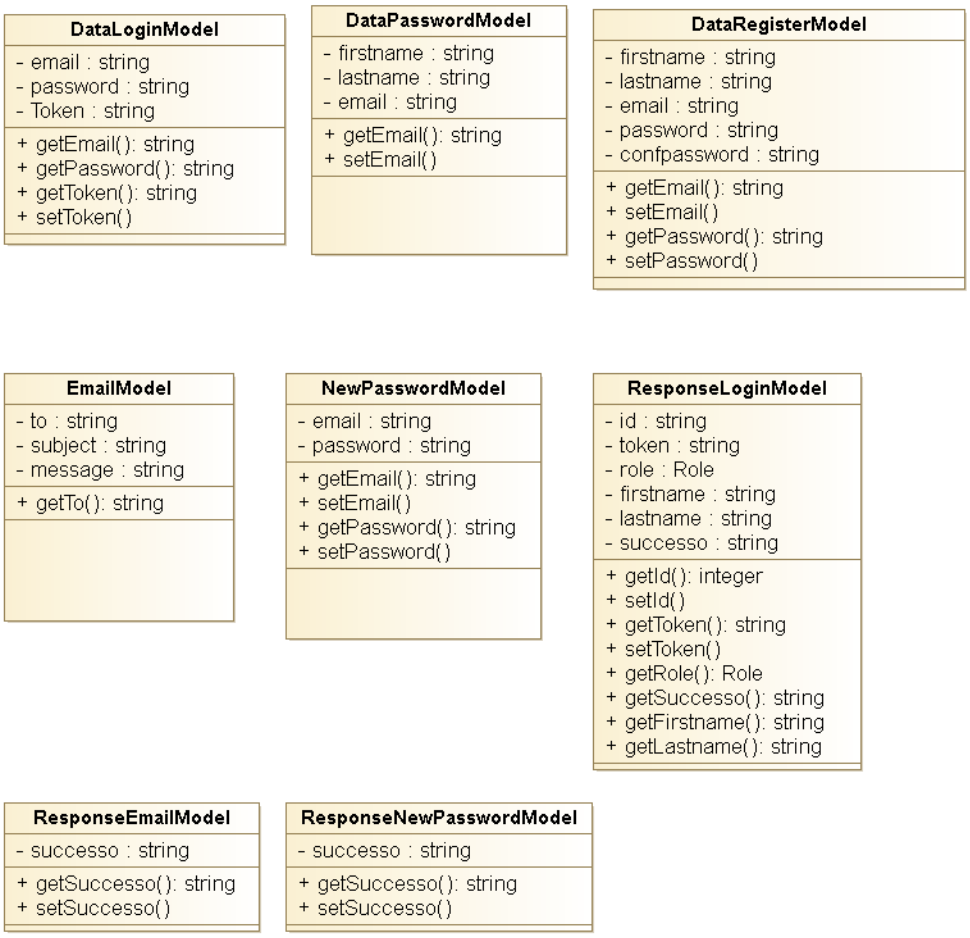

Figura 23: Classi per AuthenticationAPICall

3.15.2 Classi per AdministratorAPICall

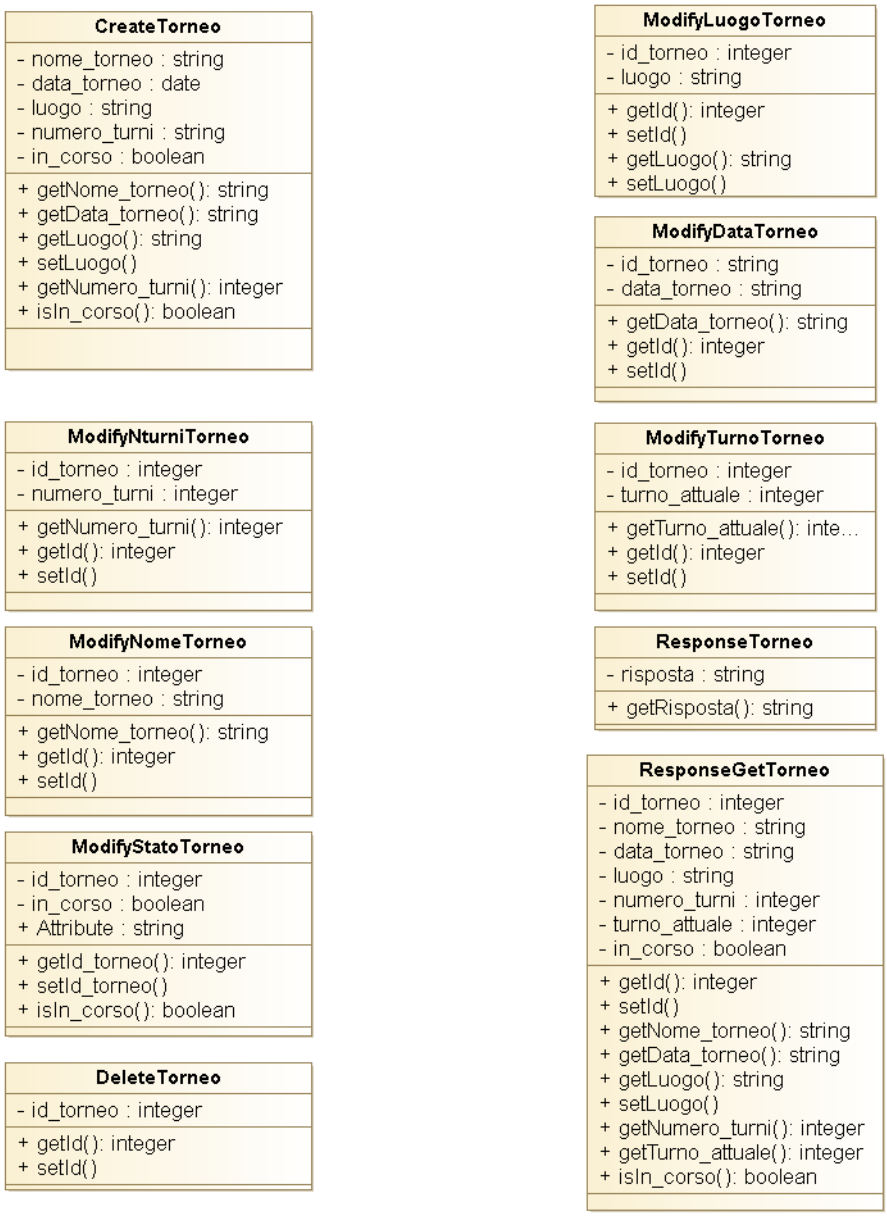

Figura 24: Classi per AdministratorAPICall

3.15.3 Classe Token

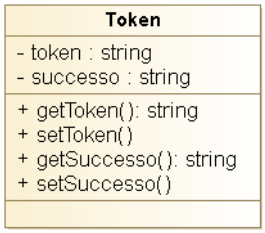

Figura 25: Classe Token

3.15.4 Vista generale lato Client e Server

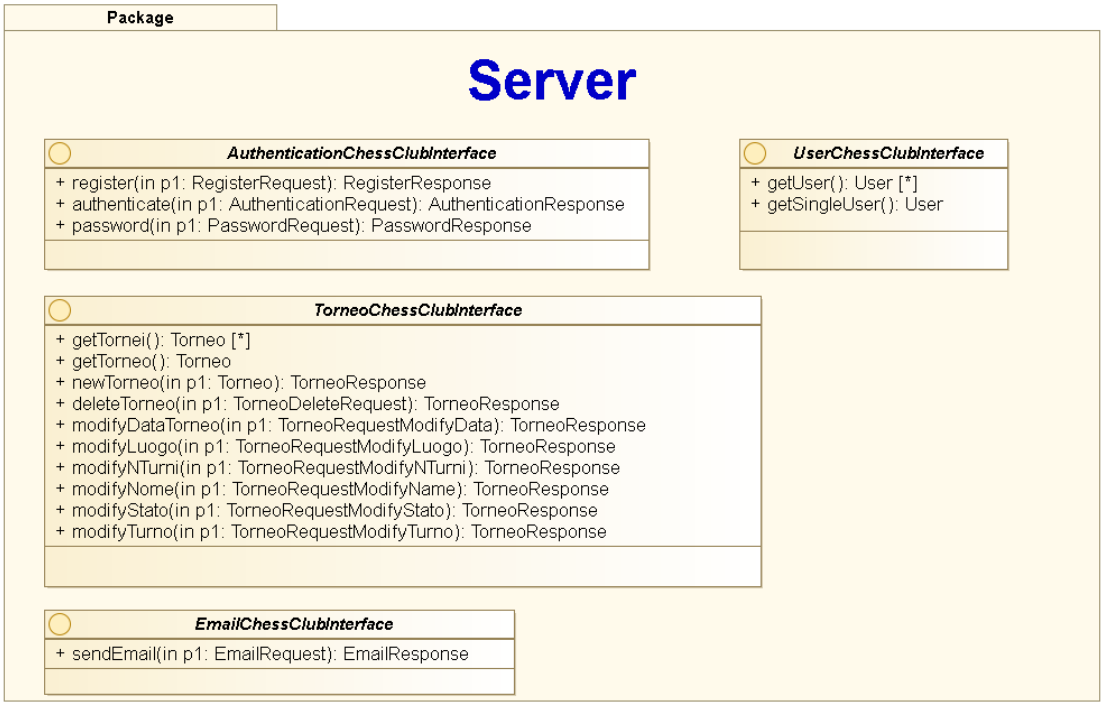

Figura 26: UML Diagram Class lato Server

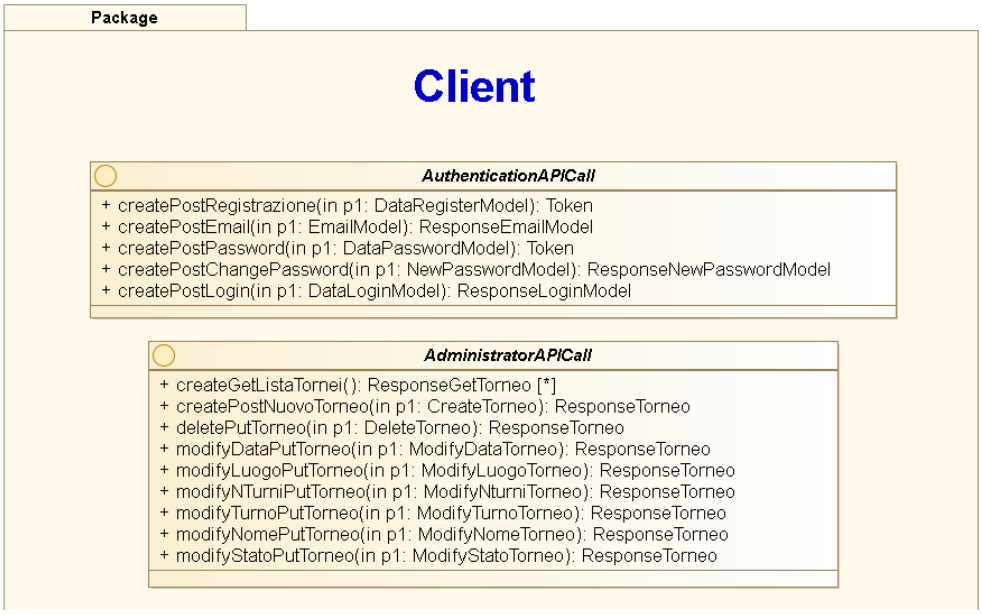

Figura 27: UML Diagram Class lato client

3.16 UML Deployment Diagram

Mediante l'utilizzo di un Deployment Diagram UML è possibile mostrare la rappresentazione hardware e software del sistema. Utilizziamo le seguenti componenti:

- *«boundary»*: componenti lato front-end con cui gli attori si interfacciano direttamente.
- *«control»*: componenti lato back-end che gestiscono la logica del programma ed espongono le API al front-end, richiedendone a loro volta al database.
- *«data»*: componenti lato back-end che interagiscono con il database.

Nella seguente figura vengono mostrati i componenti contenuti nei seguenti nodi:

- Cellulare amministratore, nodo in cui l'amministratore potrà gestire gli utenti,tornei e partite del circolo.
- Cellulare utente, nodo in cui un soggetto potrà registrarsi, iscriversi alle partite e gestire il suo profilo.

- Web Server, espone le API richiesta lato front-end
- DataBase, funge da storage dei dati.

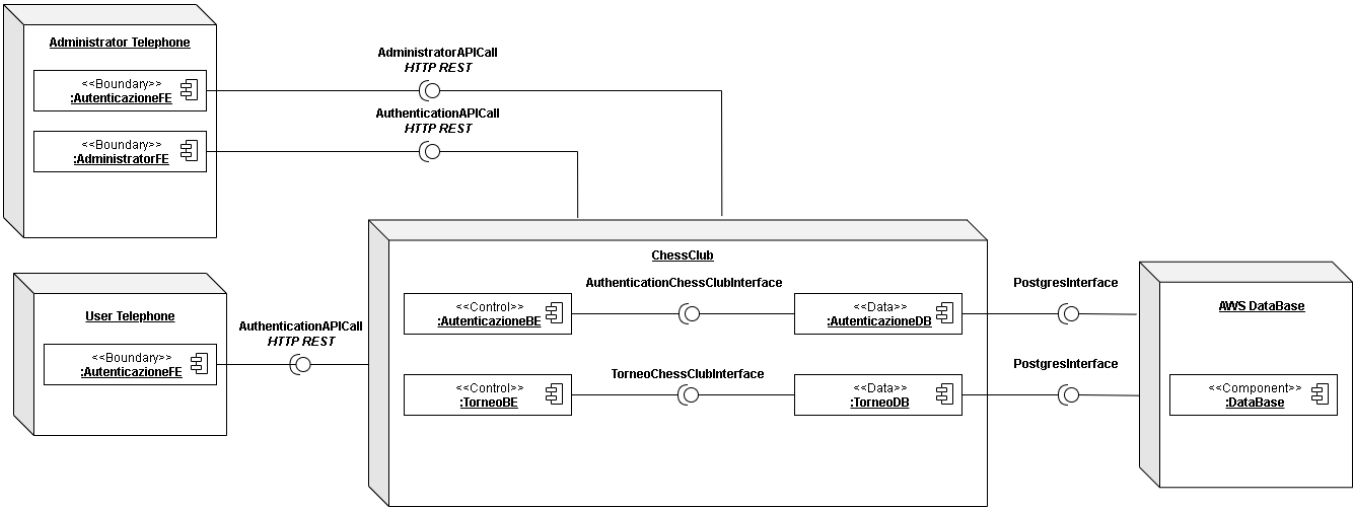

Figura 28: UML Deployment Diagram

3.17 Testing

3.17.1 Analisi statica: Unit test

Per effettuare l'analisi statica, di alcune funzioni presenti nel back-end, è stato utilizzato il framework JUnit Test. Quest'ultimo attraverso un'estensione è stato implementato all'interno dell' IDE di sviluppo *Eclipse*.
Per ogni *Service* utilizzato durante l'implementazione dell'iterazione1 sono stati effettuati dei test.

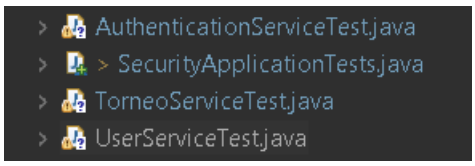

Figura 29: File di test

3.17.2 Test UserService

Figura 30: Run test User

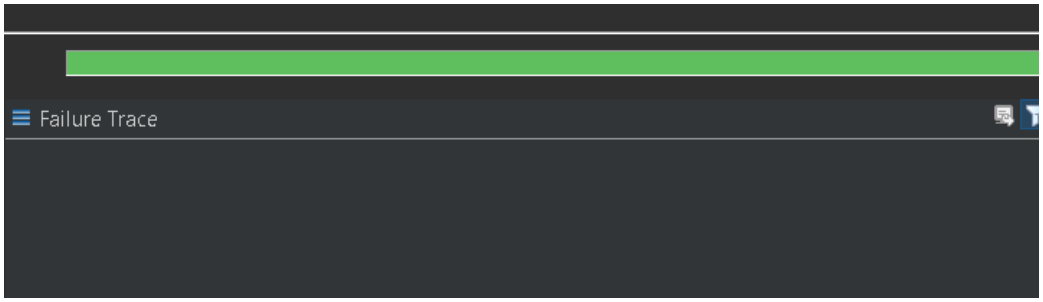

Figura 31: Failure trace User

3.17.3 Test TorneoService

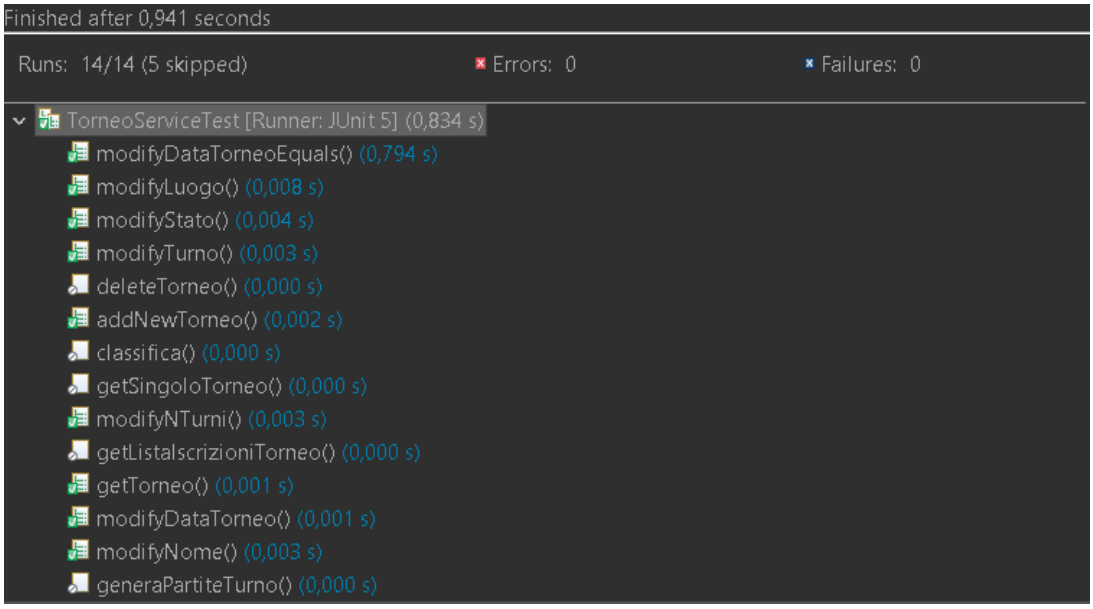

Figura 32: Run test Torneo

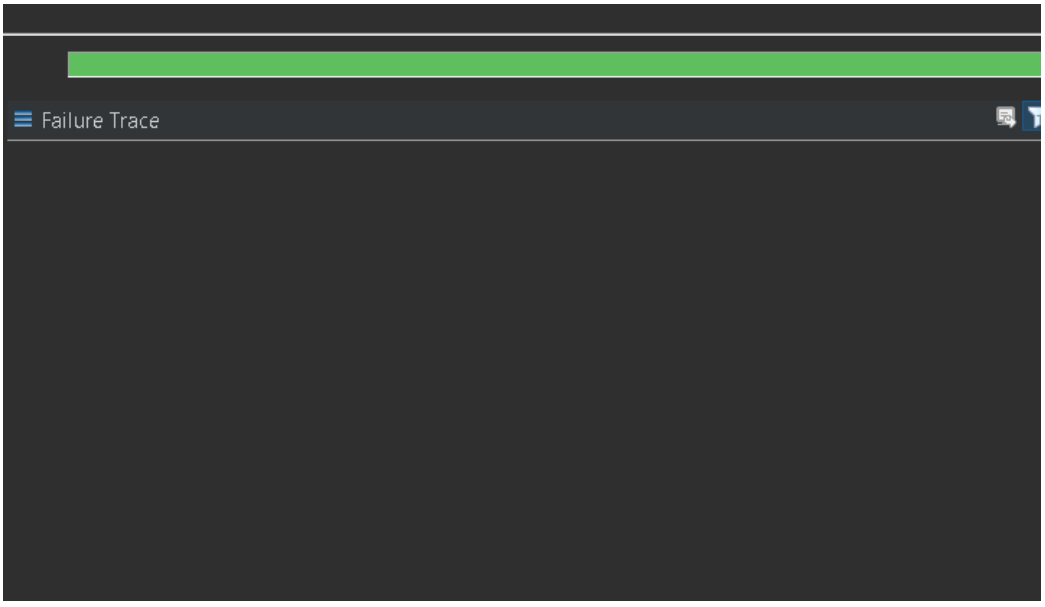

Figura 33: Failure trace Torneo

3.17.4 Test AuthenticationService

Figura 34: Run test Authentication

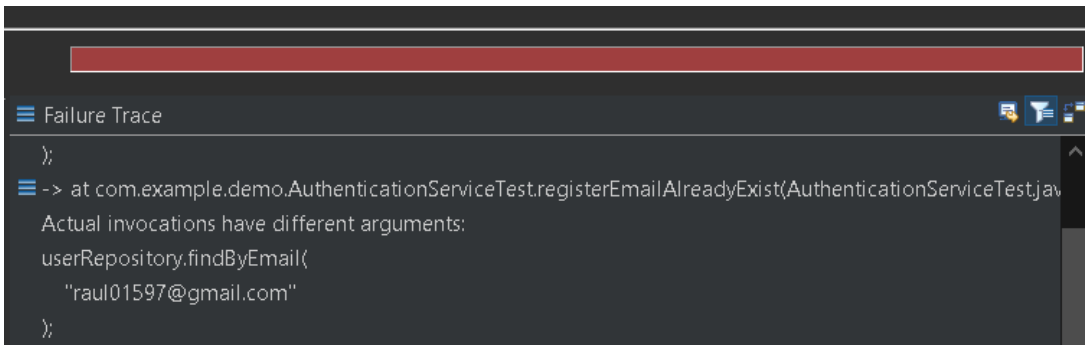

Figura 35: Failure trace Authentication

Come possiamo vedere in questo il test è fallito, ma il fatto che fallisce vuol dire che il test è stato effettuato correttamente.

Figura 36: register

Figura 37: registerEmailAlreadyExist

3.17.5 Analisi dinamica

Per effettuare l'analisi dinamica delle API è stata utilizzato Postman. Quest'ultimo è una piattaforma per creare, progettare, iterare e testare le proprie API.Postman ha confermato la correttezza delle API testate. Di seguito verranno presentati singolarmente i risultati dei test sulle API congiuntamente alla loro documentazione.

Authenticate

- **Registrazione**

  Input Request:

  – firstname

  – lastname

  – email

  – password

  Output Response:

  – token

  – successo

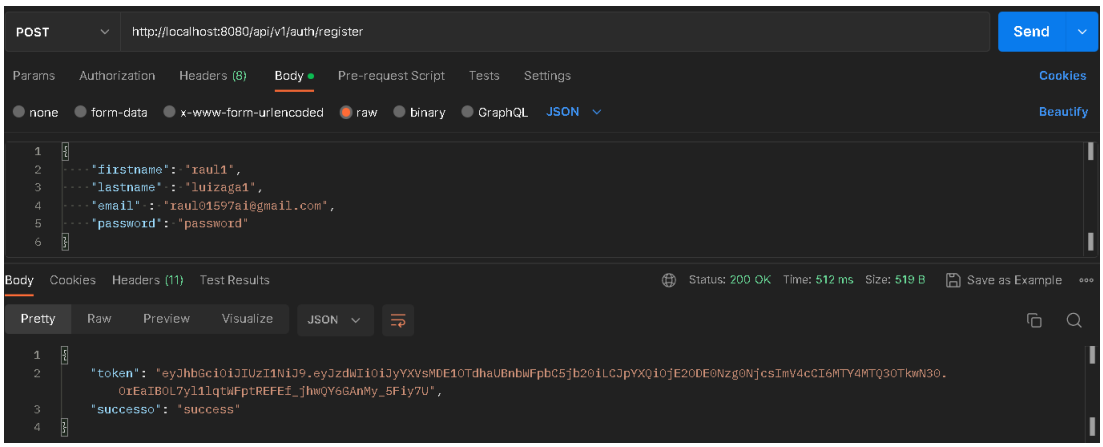

Figura 38: Registrazione

- **Autenticazione**

  Input Request:

  – email

  – password

  Output Response:

  – id_user

  – token

  – role

  – firstname

  – lastname

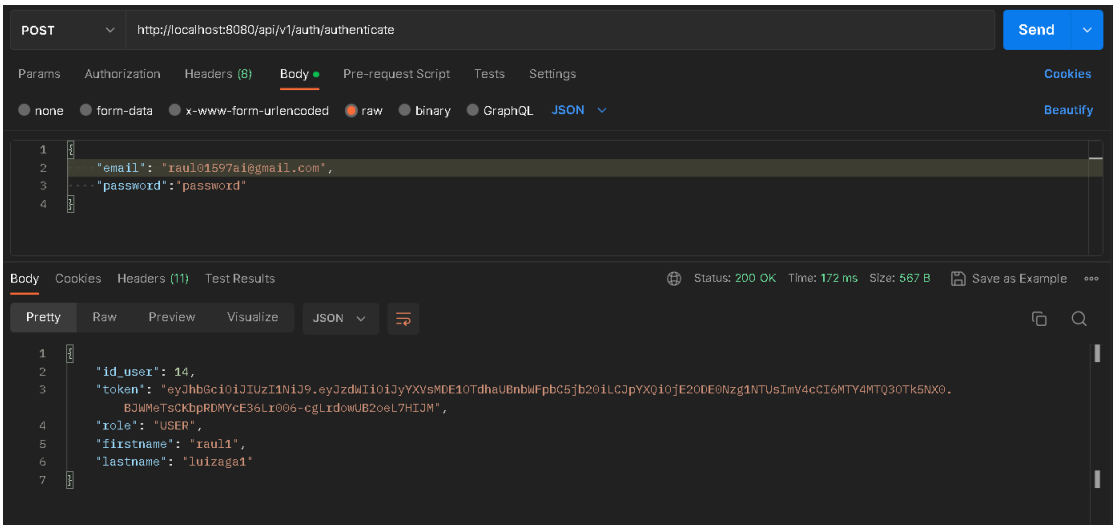

Figura 39: Autenticazione

- **Richiesta cambio password**

  Input Request:

  – firstname

- lastname
- password

Output Response:

- token
- success

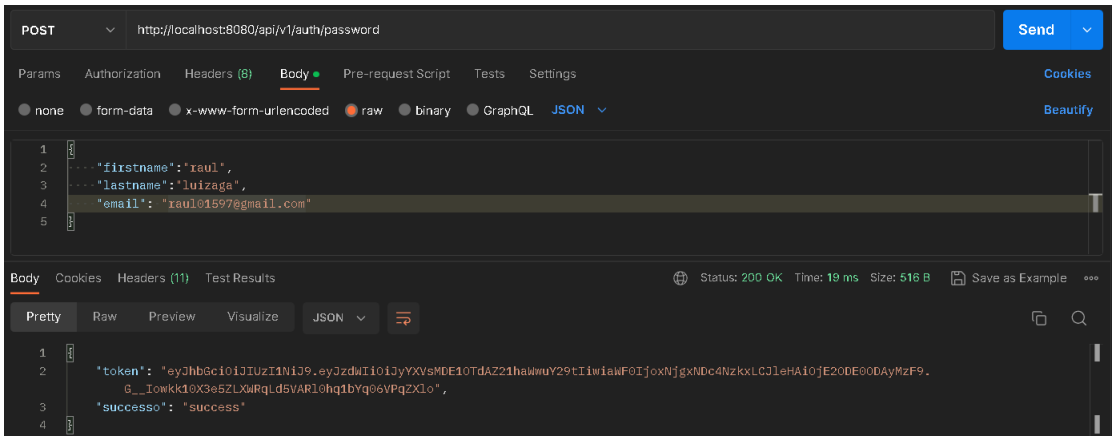

Figura 40: Password

- **Cambio password**

  Input Request:

  - email
  - password

  Output Response:

  - success

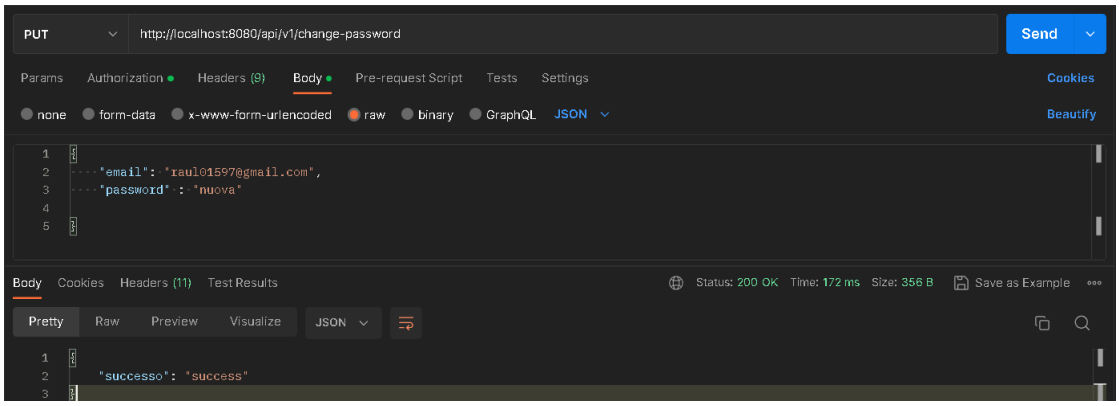

Figura 41: Change Password

- **Autenticazione con password vecchia**

  Input Request:

  - email
  - password

  Output Response:

  - errore

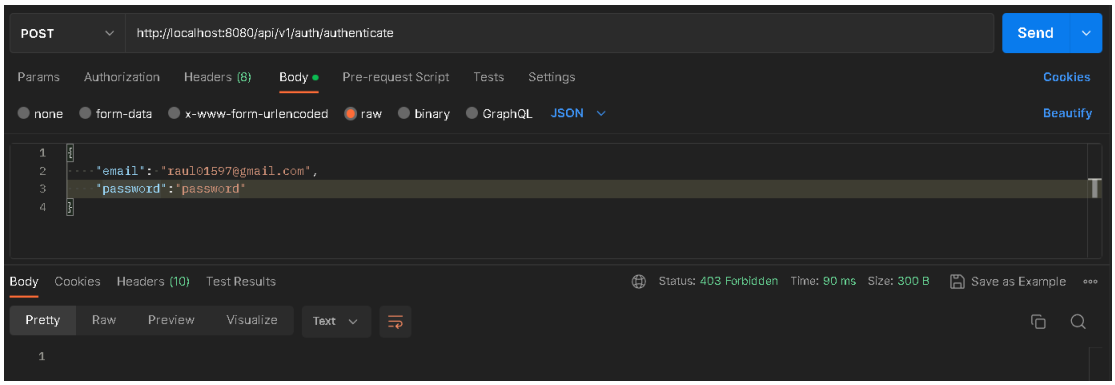

Figura 42: Password cambiata

- **Autenticazione con password aggiornata**

  Input Request:

- email
- password

Output Response:

- id_user
- token
- role
- firstname
- lastname

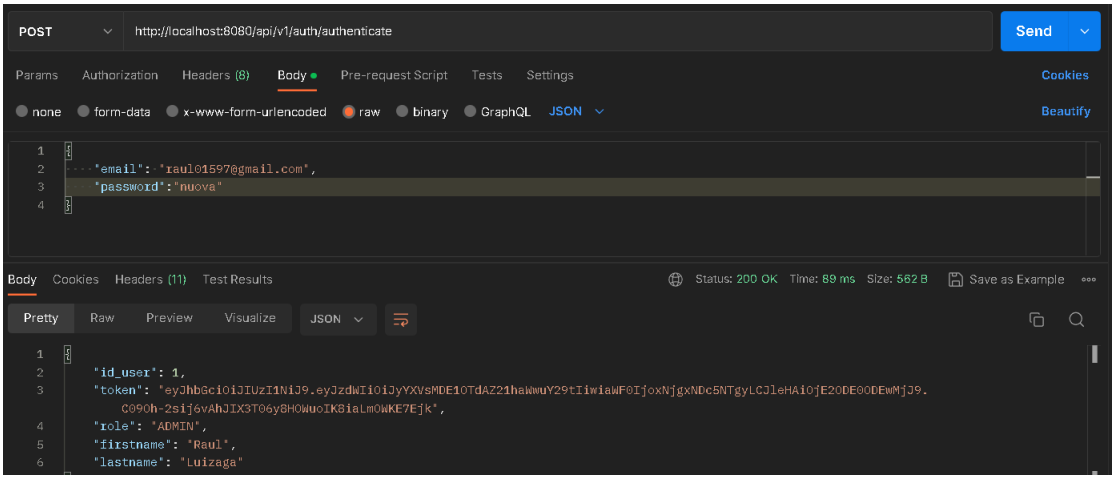

Figura 43: Password cambiata

Da notare che *Cambio password* non è dentro l'entry point di *Authenticate*, di conseguenza per come è stato implementato il sistema di sicurezza del sistema si dovrà inserire un *token*

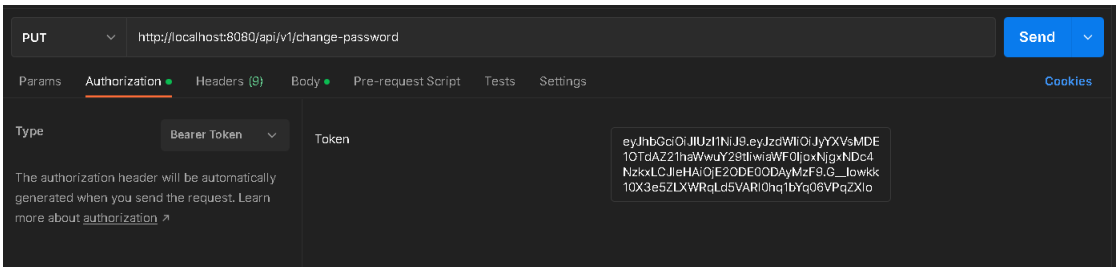

Figura 44: Bearer Token

Torneo

- **Nuovo Torneo**

  Input Request:

  - nome_torneo
  - data_torneo
  - luogo
  - numero_turni

  Output Response:

  - risposta

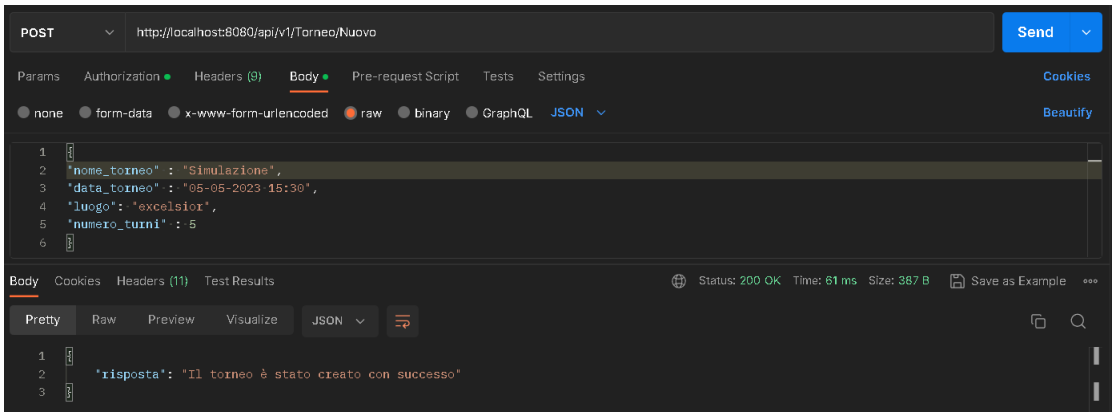

Figura 45: Nuovo Torneo

- **Get Tornei**

  Input Request:

  – nulla

  Output Response:

  – Lista dei tornei presenti

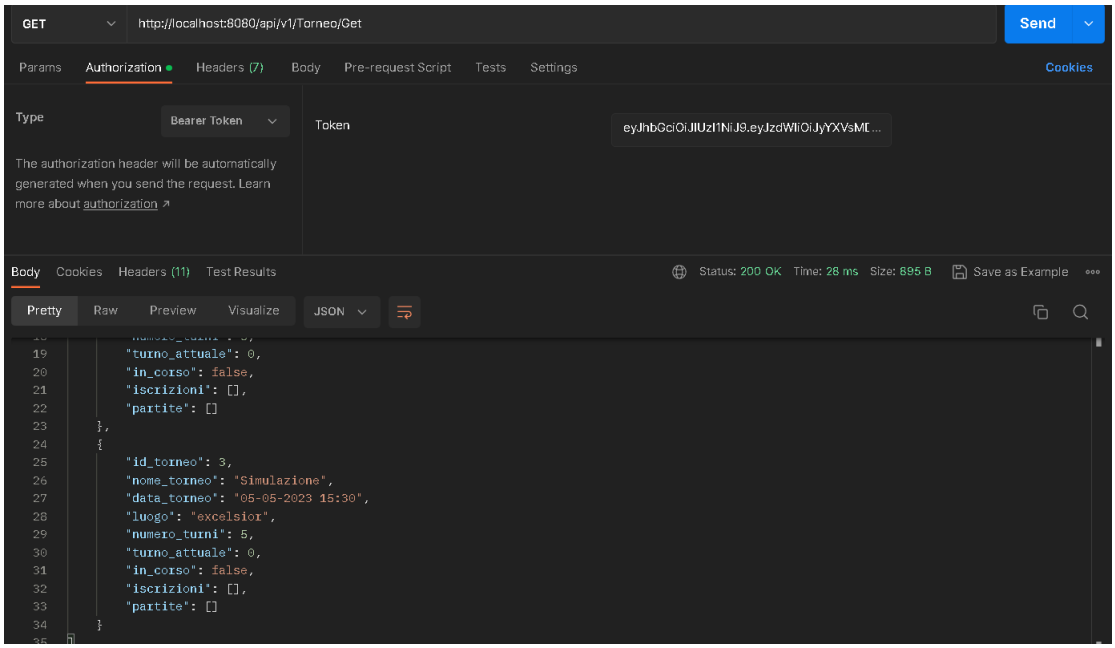

Figura 46: Get Torneo

- **Modifica Data**

  Input Request:

  – id_torneo

  – data_torneo

  Output Response:

  – risposta

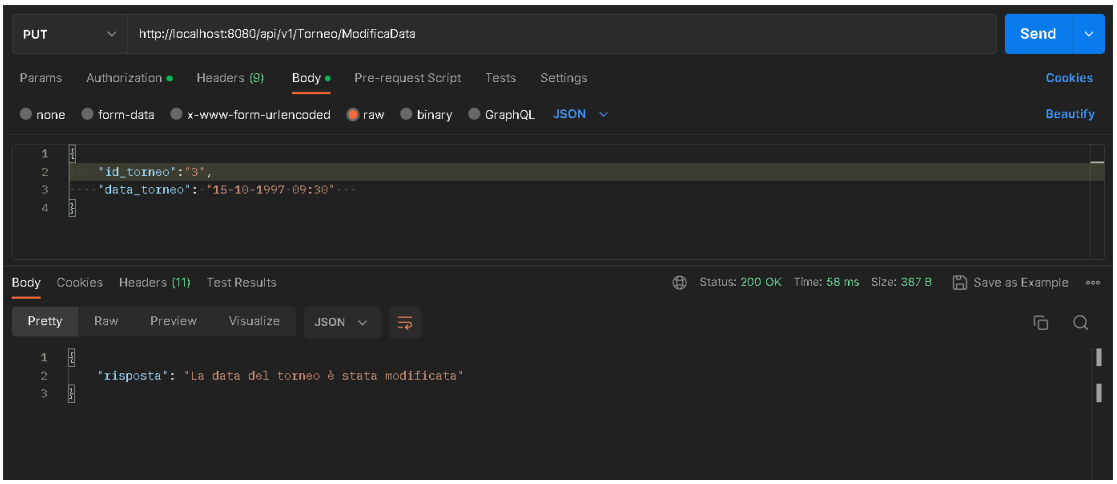

Figura 47: Modifica Data Torneo

- **Get Tornei Dopo la modifica della data precedente**

  Input Request:

  – nulla

  Output Response:

  – Lista dei tornei presenti

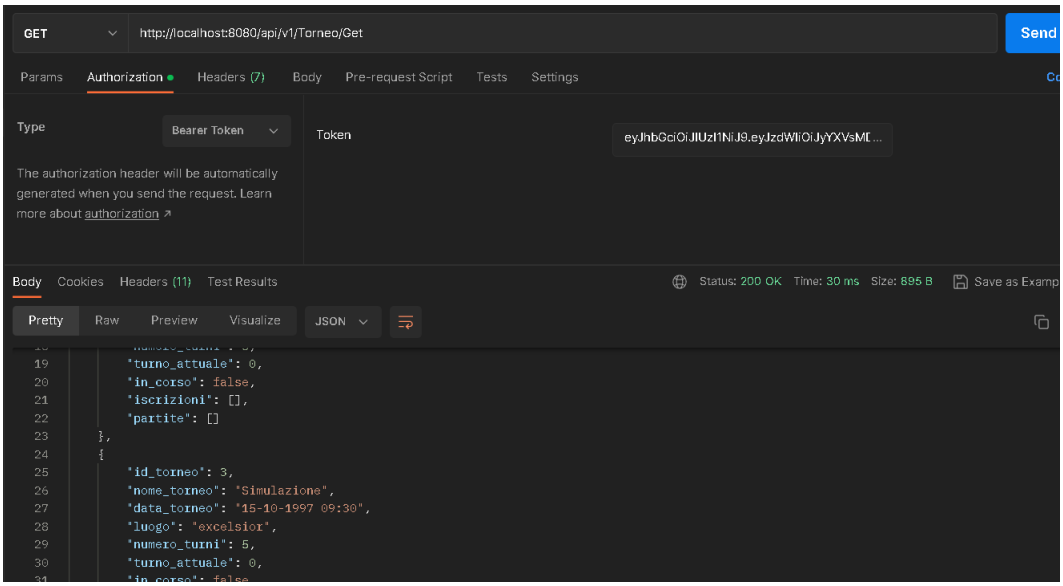

Figura 48: Get Torneo dopo la modifica della data

User

- *Get User*

  Input Request:

  - nulla

  Output Response:

  - Lista degli user

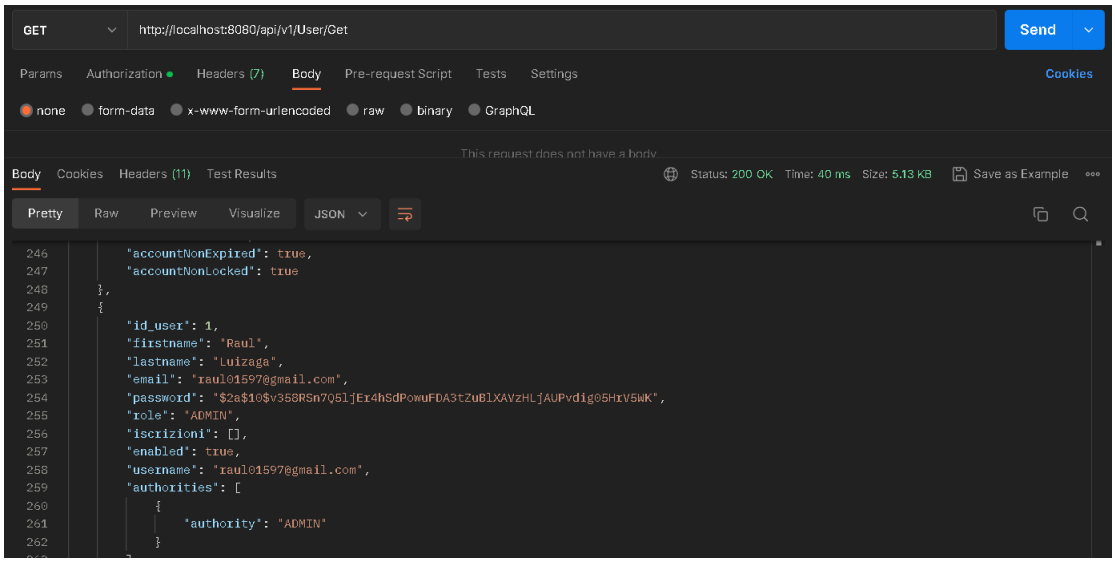

Figura 49: Get User

- *Get User By Id*

  Input Request:

  - id

  Output Response:

  - User

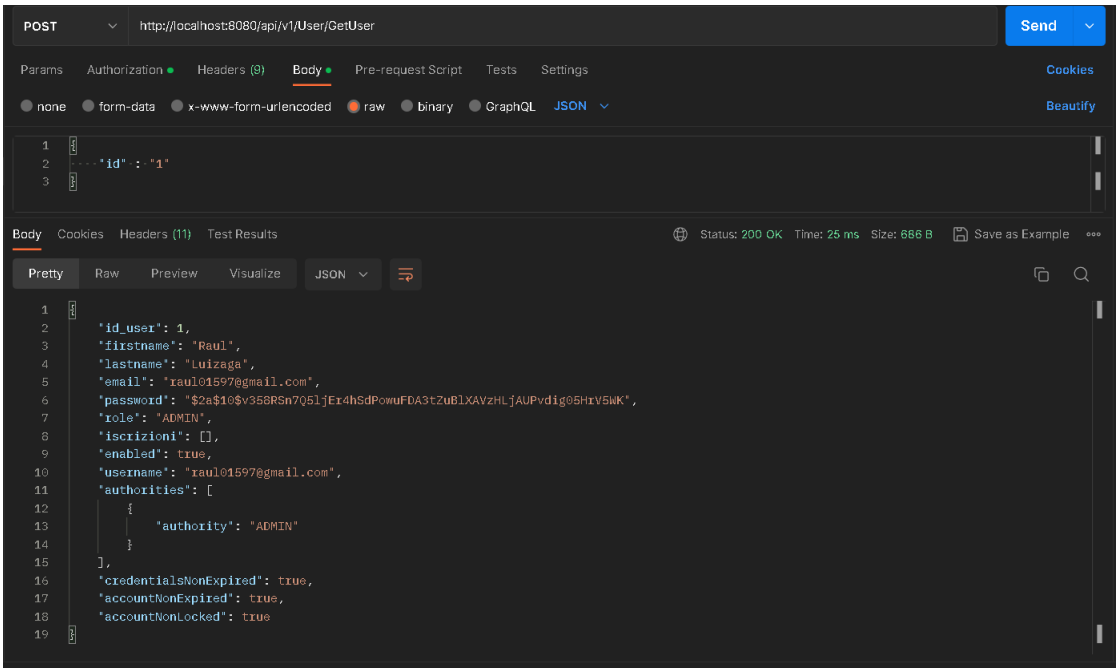

Figura 50: Get User By Id

4 Security

4.1 Introduzione

Come si è potuto notare dall'iterazione1, abbiamo utilizzato i *token* all'interno del sistema. Questi servono per effettuare dei controlli di autenticazione. Qui in questa sezione spieghiamo brevemente come sono stati utilizzati all'interno del nostro sistema.

4.2 Funzionamento JWT Authentication

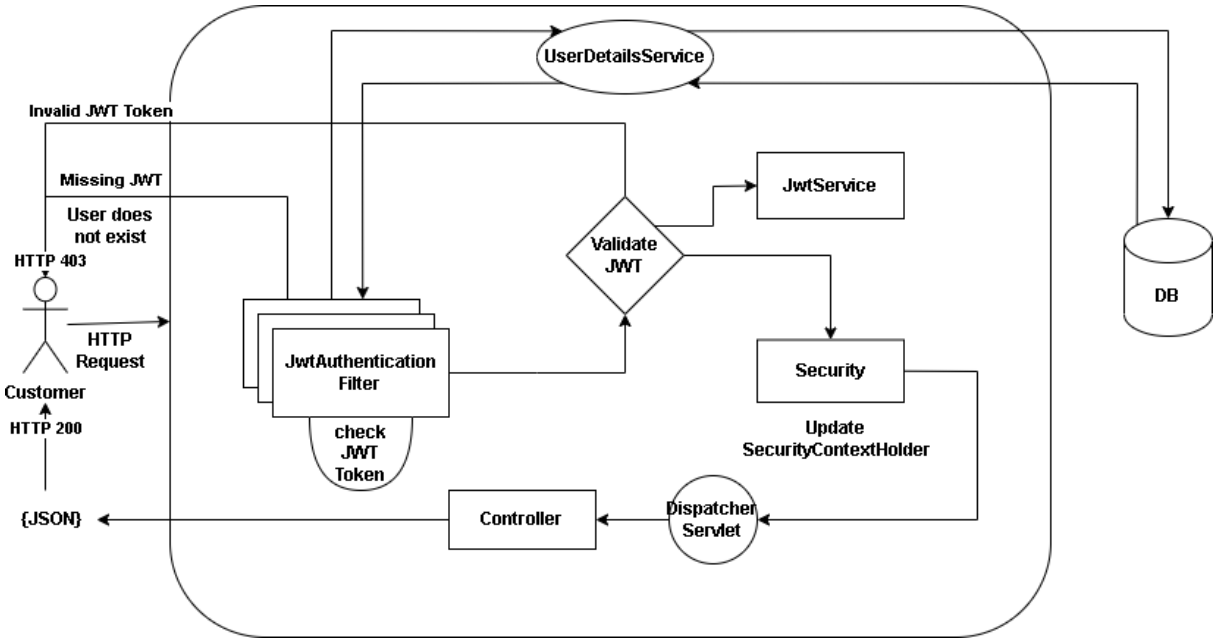

Figura 51: JWT Authentication

Passi di esecuzione per l'autenticazione:

1. Un *Customer* invia una request al nostro sistema che sta funzionando con *SpringBoot* e *Tomcat*

2. La richiesta passa per prima nel *JwtAuthenticationFilter*

3. Il filtro fa un check interno in cui controlla se la request ha un token:

- Se non ha il *token* si invia una response *HTTP 403 Missing JWT*
- Se si ha il *token* si passa al prossimo step

4. Si fa una chiamata con *UserDetailsService* al database cercando una corrispondenza con l'username, nel nostro caso tramite la mail inserita durante la registrazione, questo vuol dire che si è settata la mail come claim. L'username da utilizzare per la ricerca sarà dentro il *token* che estraggo nel *JwtAuthencationFilter*

   - Username non trovato nel database, quindi mando una risposta *HTTP 403* come response
   - Username trovato, quindi posso iniziare il processo di validazione. Si svolge questo processo perché per ogni *user* vado a generare uno specifico *token*

5. La validazione avviene in quanto si cerca di chiamare il *JwtService*. Il processo di validazione ci porta a due distinti scenari:

   - *token* non valido perché scaduto o non appartiene a questo user, quindi invio una response *HTTP 403* invalid token
   - Aggiornamento del *SecurityContextHolder*, quindi setto l'user come autenticato

6. Una volta che l'user è autenticato si manda la request al *dispatcher serlet*

7. La *servlet* poi manderà la request allo specifico controllore per eseguire la richiesta, quindi si avrà la risposta *HTTP 200*

4.3 Spring Security Architecture

Prima di entrare nello specifico nei vari componenti della **Spring Security Architecture** si illustra nella figura 52 il funzionamento generale del processo di validazione.

4.4 Visione generale del processo di validazione

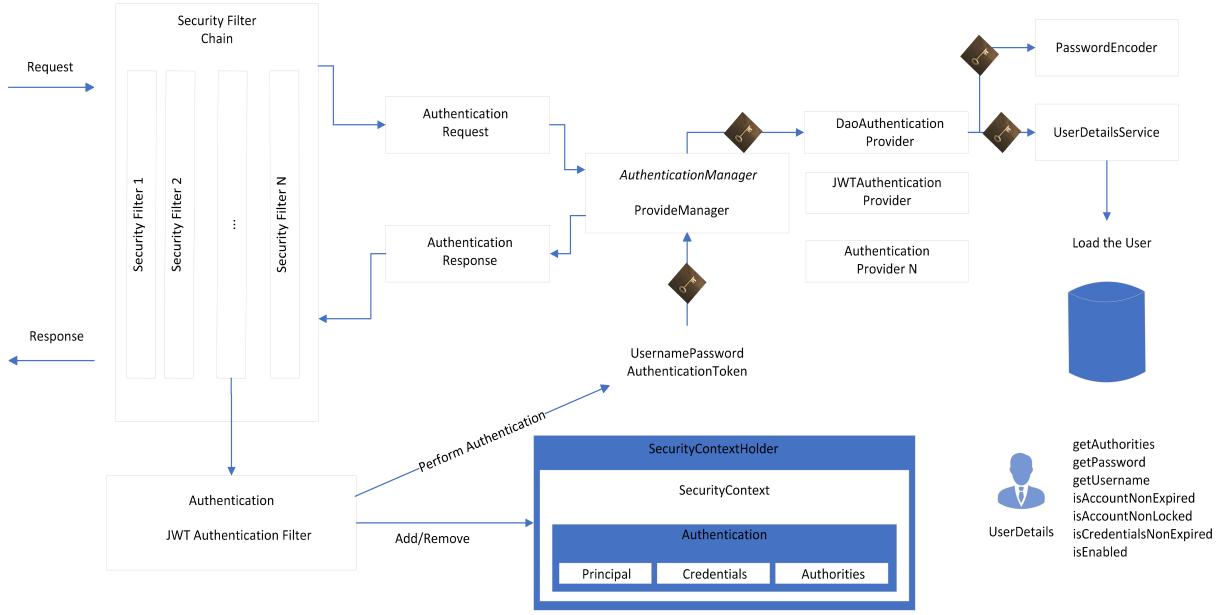

Figura 52: Spring Security Architecture

4.5 Percorso di validazione

1. L'user effettua una *request*
2. La *request* prima di raggiungere la *servlet* passa attraverso una serie di filtri
3. Nel caso di autenticazione non andata a buon fine avremo un errore del tipo *401* o *403*.
4. Nel caso di autenticazione positiva allora avremo in uscita una *Authenticazion request*.
5. *AuthenticationManager* si occupa di gestire la richiesta di autenticazione che hanno superato la catena di filtri.
6. Esso prende la *Authentication Request* ed esegue l'autenticazione in base al *provider*, in questo caso *DAO Authentication Provider*. *DAO Authentication Provider* è una

autenticazione basata su *Username* e *Password*, il quale ha bisogno di un *Password Encoder* per cercare nel *DB* e fare *match*.

7. Successivamente carica l'*user* dal *DataBase*.

   - Se l'autenticazione fallisce l'*user* viene eliminato dal *SecurityContextHolder*
   - Se l'autenticazione è andata a buon fine la inseriamo nella *SecurityContextHolder*.

4.5.1 Security Filter Chain

Rappresenta un serie di filtri che vengono eseguiti in modo sequenziali in base alla configurazione definita dal progettista. Si occupa di effettuare una serie di controlli delle richieste inviate da parte dell'utente.

4.5.2 JWT Authentication Filter

Filtro che intercetta la *request* e controlla che il *token* è stato inviato al server. Nel caso di esito negativo rifiuta la *request*. Nel caso di esito positivo il filtro crea un *Username* ed una *Password* che servono per far funzionare *AuthenticationManager*

4.5.3 UsernamePassword e AuthenticationToken

Oggetto creato da *JWT Authenticaiton Filter*, il quale sarà utilizzato per far eseguire l'autenticazione dal *AuthenticationManager*

4.5.4 Authentication Manager and Provider Manager

AuthenticationManager è un *API* che definisce come i filtri di sicurezza utilizzati da *Spring* vengono eseguiti per l'autenticazione. *ProviderManager* è l'implementazione più usata di *AuthenticationManager*, infatti *ProviderManager* è un interfaccia, mentre *ProviderManager* è la classe che implementa tale interfaccia.

4.5.5 DAO Authentication Provider

L'autenticazione avviene facendo dei controlli su *username* e *password*. DAO = Data Access Object.

4.5.6 User Details Service

Componente che carica l'*user* dal *database*

4.5.7 UserDetails

Intefaccia che è implementata dall'*user* per utilizzare i seguenti metodi:

- getAuthorities: gestione ruoli e permessi
- getPassword: password che non viene salvata in chiaro e sarà mappato con il *password encoder*, il quale si occupa della crittografia
- getUsername: identificativo user
- isAccountNonExpired: booleano
- isCredentialNonExpired: booleano
- isEnable: booleano

4.5.8 Security Context Holder

Oggetto che contiene i dettagli delle autenticazioni correnti degli *user*. Se l'autenticazione fallisce, allora si rimuove l'user dal *context holder* o l'autenticazione. Se l'autenticazione ha successo, allora lo aggiungiamo al *context holder*

4.6 Funzionamento Validazione

Quando un client invia una richiesta http al backend-system, la prima cosa che viene eseguita dall'applicazione Spring è il filtro. Viene creato ogni volta... *JWT AUTHENTICA-*

TION FILTER è il nome del filtro che ha funzione di validazione / controllo di tutto ciò che riguarda i *JWT* token.
La prima cosa che succede è un check interno al filtro,viene chiesto se siamo in possesso del *JWT* token.

- In caso di mancanza del token la risposta al client sarà di tipo *HTTP 403*
- In caso di possesso del token viene avviato la procedura di *Validation Process*

4.7 Validation Process

Il filtro prima esegue una chiamata al suo interno utilizzando *User Detail Service* per cercare di recuperare le informazioni sull'utente dal database *POSTGRESQL*. Il recupero delle informazioni dell'utente avviene tramite la mail inserita durante la registrazione che imposteremo come *Claim* (attestato / richiesta) o un oggetto token che estrarremo all'interno del *JWT Authentication Filter* (Prima chiamata). Una volta che l'user è stato recuperato avremo una risposta dal database , la risposta è una delle seguenti:

- User esistente
- User non esistente

La risposta del database verrà inviata al *JWT Authentication Filter*, di conseguenza il filtro farà i suoi controlli:

- Se l'user esiste: inizio *Validation Process*
- Se l'user non esiste : *Error 403*

Si inizia il *Validation Process* perché il *JWT* token è stato generato per uno specifico User, così vogliamo validare questo token in base all'user.

4.8 JWT Token

Un *JWT Token* è anche detto *JSON WEB TOKEN* e nello specifico definisce un modo compatto e autonomo per la trasmissione sicura di informazioni tra le parti di interesse come oggetto *JSON*. Queste informazioni possono essere verificate e sono attendibili perché sono firmate digitalmente. I *JWT* possono essere firmati utilizzando un segreto (con l'algoritmo HMAC) o una coppia di chiavi pubblica/privata utilizzando RSA o ECDSA. Sebbene i *JWT* possano essere crittografati per garantire anche la segretezza tra le parti, ci concentreremo sui *token* firmati. I *token* firmati possono verificare l'integrità delle attestazioni contenute al suo interno, mentre i *token* crittografati nascondono tali attestazioni da altre parti. Quando i *token* vengono firmati utilizzando coppie di chiavi pubblica/privata, la firma certifica anche che solo la parte in possesso della chiave privata è quella che l'ha firmata.

4.8.1 Scenari di utilizzo dei JSON WEB TOKENS

- **Authorization**:

  Questo è lo scenario più comune per l'utilizzo di *JWT*. Una volta che l'utente ha effettuato l'accesso, ogni richiesta successiva includerà il *JWT*, consentendo all'utente di accedere a percorsi, servizi e risorse consentiti con quel *token*. Single Sign-On è una funzionalità che utilizza ampiamente JWT al giorno d'oggi, a causa del suo piccolo sovraccarico e della sua capacità di essere facilmente utilizzata in diversi domini.

- **Information Exchange** :

  I *token Web JSON* sono un buon modo per trasmettere in modo sicuro le informazioni tra le parti. Poiché i *JWT* possono essere firmati, ad esempio utilizzando coppie di chiavi pubblica/privata, puoi essere certo che i mittenti siano chi dicono di essere. Inoltre, poiché la firma viene calcolata utilizzando *l'intestazione* ,il *payload*, puoi anche verificare che il contenuto non sia stato manomesso.

4.8.2 Struttura JSON Web Token

Nella sua forma compatta, i *Web Token JSON* sono costituiti da tre parti separate da punti (.), che sono:

- **Header**
- **Payload**
- **Signature**

Tipicamente un *JWT* si presenta come nel seguente modo:

xxxxx.yyyyy.zzzzz

Ora vediamo separatamente le varie parti...

4.8.3 Header

L'intestazione in genere consiste di due parti: il tipo di *token*, che è *JWT*, e l'algoritmo di firma utilizzato, come *HMAC SHA256* o *RSA*.
Un piccolo esempio di *Header*

```
1 {
2     "alg": "HS256"
3     "typ" : "JWT"
4 }
```

Listing 1: Header JWT

Quindi, questo JSON è codificato Base64Url per formare la prima parte del JWT.

4.8.4 Payload

La seconda parte del *token* è il *payload*, che contiene le attestazioni. Le attestazioni sono dichiarazioni su un'entità (in genere, l'utente) e dati aggiuntivi. Esistono tre tipi di claims: registrati, pubblici e privati.

- **Registered claims**:

  Si tratta di un insieme di attestazioni predefinite che non sono obbligatorie ma consigliate, per fornire un insieme di attestazioni utili e interoperabili. Alcuni di essi sono: iss (emittente), exp (tempo di scadenza), sub (oggetto), aud (pubblico) e altri.

- **Public claims**: Questi possono essere definiti a piacimento da coloro che utilizzano i *JWT*. Tuttavia, per evitare collisioni, dovrebbero essere definiti nel *registro dei token Web JSON IANA* o essere definiti come un *URI* che contiene uno spazio dei nomi, resistente alle collisioni.

- **Private claims**: Si tratta di attestazioni personalizzate, create per condividere informazioni tra le parti che concordano di utilizzarle e non sono attestazioni registrate o pubbliche.

Un esempio di *payload* potrebbe essere il seguente....

```
1 {
2     "sub": "1234567890",
3     "name" : "John Doe",
4     "admin" : true
5 }
```

Listing 2: Header JWT

Il *payload* viene quindi codificato *Base64Url* per formare la seconda parte del *JSON Web Token*.

4.8.5 Signature

Per creare la parte della firma devi prendere l'intestazione codificata, il *payload* codificato, un segreto, l'algoritmo specificato nell'intestazione e firmarlo.
Ad esempio, se si desidera utilizzare l'algoritmo HMAC SHA256, la firma verrà creata nel seguente modo:

```
1 HMACSHA256(
2   base64UrlEncode(header) + "." +
3   base64UrlEncode(payload),
4   secret)
5
```

Listing 3: Header JWT

La firma viene utilizzata per verificare che il messaggio non sia stato modificato lungo il percorso e, nel caso di *token* firmati con una chiave privata, può anche verificare che il mittente del *JWT* sia chi dice di essere.

4.8.6 Mettendo tutto insieme

L'output è costituito da tre stringhe *Base64-URL*, separate da punti che possono essere facilmente passate in ambienti *HTML* e *HTTP*, pur essendo più compatte rispetto agli standard basati su *XML* come *SAML*. Di seguito viene mostrato un *JWT* con l'intestazione e il *payload* precedenti codificati ed è firmato con un segreto.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Figura 53: JWTencoded

Se vuoi giocare con JWT e mettere in pratica questi concetti, puoi utilizzare jwt.io Debugger per decodificare, verificare e generare JWT.

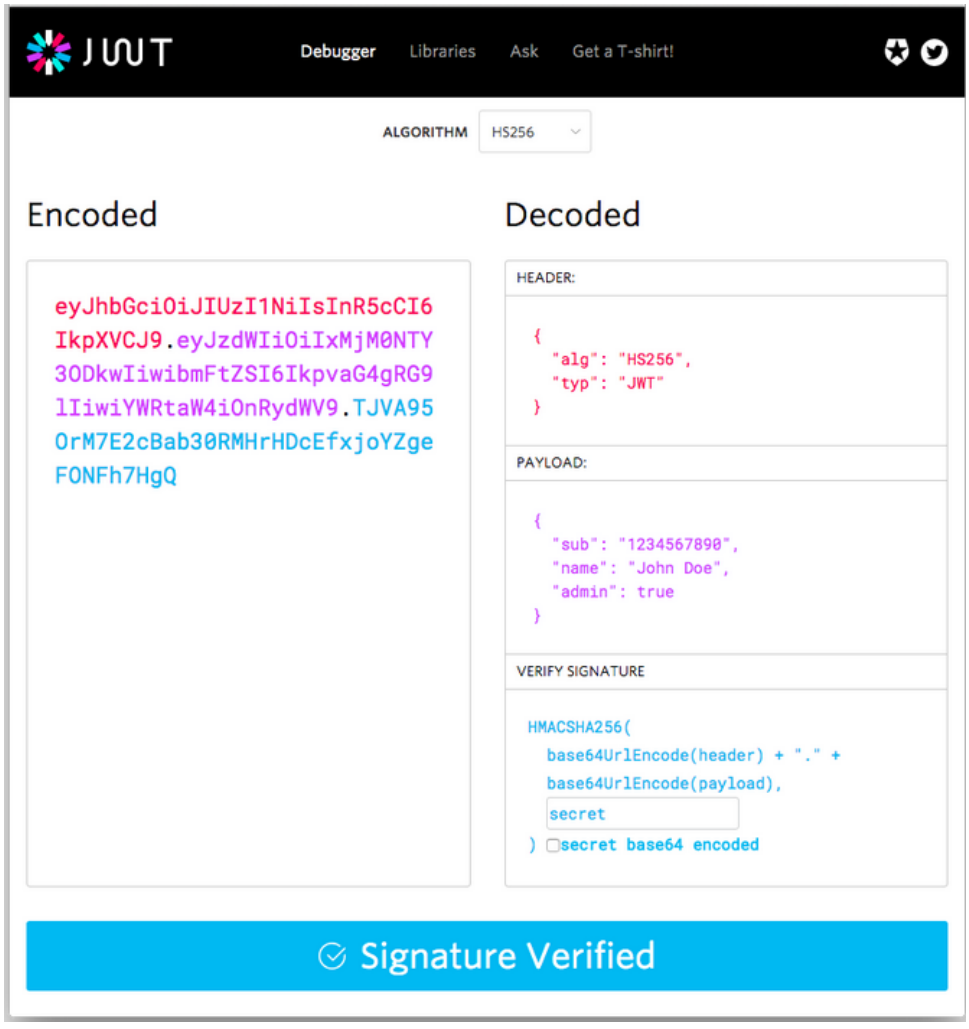

Figura 54: JWTGeneral

4.9 Basic Authentication

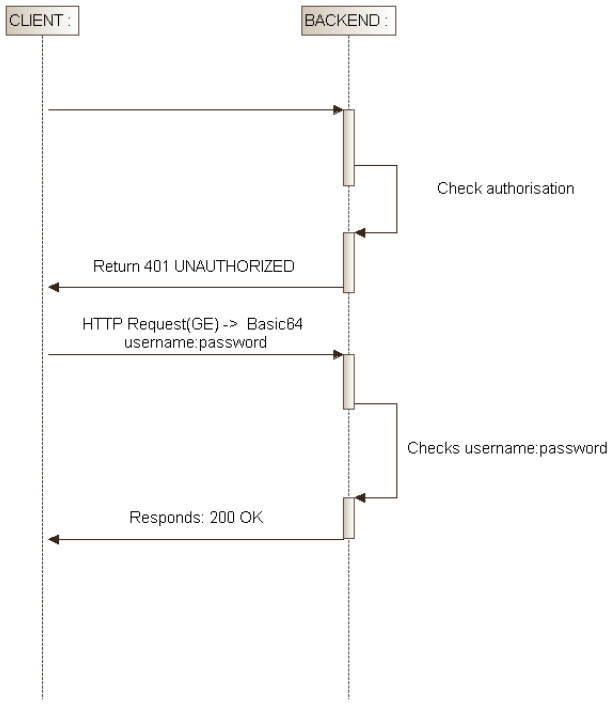

Figura 55: Sequence Diagram Basic Authentication

Sul lato sinistro, abbiamo il *client* che può essere un browser, un'applicazione web o un applicazione mobile che vuole comunicare con il nostro *Backend*. Quindi il *client*, la prima volta invierà una richiesta *HTTP* senza alcuna autorizzazione. Il *backend* controllerà l'autorizzazione, quindi in questo caso in cui non è stata fornita l'autorizzazione,per esempio, come *utente e password* risponderà con un messaggio *401 UNAUTHORIZED*. La prossima volta o la seconda volta, il *client* invierà una richiesta *HTTP* con un *username e password* nel formato *Basic64*. Il *Backend*, in questo caso, effettuerà ancora un controllo ma con la presenza di un autorizzazione. In questo caso, viene fatto un controllo più specifico sull'*username e password* inviati dall'*utente*, in quanto bisogna controllare che tale coppia faccia parte del nostro sistema di *backend* o del sistema di autenticazione. Quindi se la risposta è positiva si manda al *client* una risposta del tipo *200 OK*.

4.10 JWT Validate Process

JWT Validate Process è un processo/meccanismo il quale cerca di chiamare il *JWT Service*. *JWT Service* prende i parametri:

- User
- Token e/o Token String e/o JWT Token

Dopo l'esecuzione di *Validate Process* ho 2 scenari possibili:

1. Token non valido:
   - Token scaduto
   - Token trovato appartenente ad un altro User

   Quindi viene mandata una risposta alla chiamata http,*Invalid JWT Token*.

2. Aggiornamento del *Security Context Holder*, dove si setta l'User come connesso. Questo perché quando estraggo *User Detail Information* dal database, noi siamo in grado di settare il *Security Context Holder*. In poche parole, diciamo a Spring o al resto della catena di filtri che questo User è autenticato. Aggiorniamo *Authentication Manager* ogni volta che controllo che l'User è autenticato per la richiesta e la risposta è *Successful*; una volta che il *Context Holder* sarà aggiornato, questo sarà automaticamente in grado di evadere la richiesta e sarà mandato al *Dispatcher Servlet* e da qui sarà mandato direttamente al *Controller* dove svolgeremo le esecuzioni di cui abbiamo bisogno, quindi chiamaremo il *Service*, andremo nel database e mandaremo indietro la risposta alla http request. In questo caso la risposta sarò un http 200.

User Detail Service controlla l'esistenza nel database, appoggiandosi al *JWT Service* per estrarre l'username che nel nostro caso sarà la mail inserita durante la registrazione.

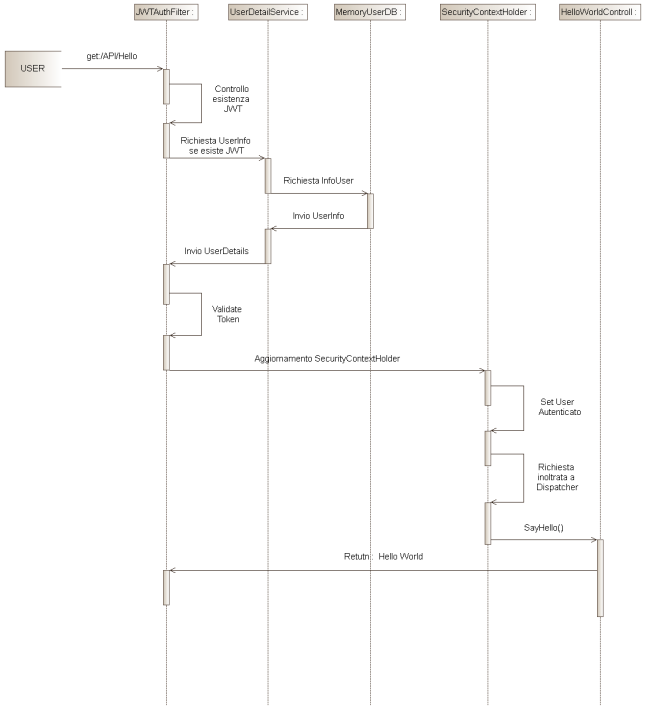

Figura 56: Interaction Sequence Diagram

5 Iterazione 2

5.1 Introduzione

Nella seconda iterazione si è scelto di implementare i casi d'uso riguardanti:

- Gestione torneo
  - **UC5**: Preiscrizione Torneo
  - **UC6**: Cancellazione Preiscrizione
  - **UC18**: Inserimento punteggio torneo
  - **UC19**: Generazione turni torneo
  - **UCY**: Generazione classifica

UCY è un caso d'uso che è stato pensato dopo una riunione con il team di sviluppo, prima di iniziare con l'iterazione 2. Inoltre nei casi d'uso **UC19** e **UCY** è stato utilizzato l'algoritmo di ordinamento *MergeSort*.

5.2 UC5: Preiscrizione Torneo

Breve descrizione: l'utente che ha scaricato l'applicazione,si è registrato ed ha effettuato il log-in, visualizzerà la sezione iscrizione tornei ed effettuerà l'iscrizione per il torneo di suo interesse.
Precondizione:

- Utente:
  - Aver scaricato l'applicazione
  - Aver effettuato la registrazione
  - Aver effettuato il log-in
- Amministratore

  - Aver creato un torneo

Attori coinvolti :

- Utente
- Amministratore
- Sistema

Risposta del sistema:

- *successo*: torneo d'interesse disponibile, quindi iscrizione avvenuta. Il sistema rimanderà l'utente alla lista delle iscrizioni
- *fallimento*: torneo d'interesse non disponibile, quindi iscrizione non avvenuta.

PostCondizione:

- successo: Il sistema rimanderà l'utente alla lista delle iscrizioni
- fallimento: Il sistema non farà niente in quanto l'utente non può iscriversi e di conseguenza quest'ultimo dovrà iscriversi ad un altro torneo

Procedimento :

1. Log-in
2. Click sulla sezione iscrizione tornei
3. Visualizzazione tornei disponibili
4. Click sul torneo di interesse
5. Conferma iscrizione
6. Reindirizzamento alla pagina lista tornei

5.3 UC6: Cancellazione Preiscrizione Torneo

Breve descrizione: l'utente che ha scaricato l'applicazione,si è registrato, ha effettuato il log-in ed è registrato ad almeno un torneo, visualizzerà la sezione *miei tornei* ed effettuerà la cancellazione dell'iscrizione per il torneo d'interesse
Precondizione:

- Utente:
  - Aver scaricato l'applicazione
  - Aver effettuato la registrazione
  - Aver effettuato il log-in
  - Aver già effettuato l'iscrizione ad un torneo
- Amministratore
  - Aver creato un torneo

Attori coinvolti :

- Utente
- Amministratore
- Sistema

Risposta del sistema:

- *successo*: torneo d'interesse disponibile, cancellazione iscrizione avvenuta. Il sistema rimanderà l'utente alla lista delle iscrizioni
- *fallimento*: torneo d'interesse non disponibile, quindi cancellazione iscrizione non avvenuta.

PostCondizione:

- successo: Il sistema rimanderà l'utente alla lista dei tornei a cui è iscritto
- fallimento: Il sistema non farà niente in quanto l'utente non può cancellare l'iscrizione ad un torneo in cui non è iscritto

Procedimento :

1. Log-in
2. Click sulla sezione iscrizione tornei
3. Visualizzazione *miei tornei*
4. Click sul torneo d'interesse
5. Conferma cancellazione iscrizione
6. Reindirizzamento alla pagina *miei tornei*

5.4 UC19: Generazioni Turni Torneo

Breve descrizione: l'amministratore visualizzerà la sezione *tornei in corso*, selezionerà il torneo a cui è interessato e selezionerà il tasto *generazione nuovo turno*.
Precondizione:

- Amministratore:
  - Aver effettuato il log-in
  - Aver creato un torneo
  - Aver avviato il torneo

Attori coinvolti :

- Amministratore
- Sistema

Risposta del sistema:

- *successo*: turno per il torneo selezionato generato. Il sistema farà visualizzare il *turno attuale* incrementato di una unità
- *fallimento*:turno per il torneo selezionato non generato. Quindi il sistema farà visualizzare un messaggio di errore nella generazione del turno

PostCondizione:

- successo: Il sistema incrementerà il parametro *turno attuale* di 1, ma comunque si rimarrà nella stessa pagina
- fallimento: Il sistema farà visualizzare a schermo un messaggio di errore

Procedimento :

1. Log-in
2. Click sulla sezione iscrizione tornei
3. Visualizzazione *miei tornei*
4. Click sul torneo d'interesse
5. Click sulla voce *genera turno*
6. Visualizzazione parametro *turno attuale* incrementato di 1

5.5 UC18: Inserimento Punteggio Torneo

Breve descrizione: l'amministratore visualizzerà la sezione *turni*, selezionerà il torneo a cui è interessato e selezionerà la partita d'interesse a cui vuole aggiornare il punteggio.
Precondizione:

- Amministratore

  - Aver creato un torneo
  - Aver avviato il torneo
  - Aver generato il turno

Attori coinvolti :

- Amministratore
- Sistema

Risposta del sistema:

- *successo*:
  - torneo d'interesse disponibile
  - torneo d'interesse avviato
  - turno generato d'interesse generato

  Se queste condizioni valgono,allora l'inserimento del punteggio può essere effettuato. Il punteggio inserito deve essere uno dei seguenti:

  - Vittoria Bianco
    - Punteggio Bianco: 1.0
    - Punteggio Nero: 0.0
  - Vittoria Nero
    - Punteggio Bianco: 0.0
    - Punteggio Nero: 1.0
  - Pareggio
    - Punteggio Bianco: 0.5
    - Punteggio Nero: 0.5

- *fallimento*:

  Se una delle seguenti condizioni non vale, quindi fallisce.

  - torneo d'interesse disponibile
  - torneo d'interesse avviato
  - turno d'interesse generato

  Se queste condizioni valgono,allora l'inserimento del punteggio può essere effettuato.

  Il punteggio inserito **non** deve essere uno dei seguenti:

  - Vittoria Bianco
    - Punteggio Bianco: 1.0
    - Punteggio Nero: 0.0
  - Vittoria Nero
    - Punteggio Bianco: 0.0
    - Punteggio Nero: 1.0
  - Pareggio
    - Punteggio Bianco: 0.5
    - Punteggio Nero: 0.5

PostCondizione:

- successo: Il sistema rimanderà l'utente alla lista delle partite del torneo
- fallimento:
  - Se una condizione per l'inserimento dei punteggi non è rispettata,allora il sistema non risponde in quanto non è possibile eseguire tale operazione
  - Nel caso di inserimento di un punteggio errato e non coerente, allora il sistema invierà un messaggio di errore ed inviterà a reinserire il punteggio.

Procedimento:

1. Log-in
2. Click sulla sezione punteggi
3. Visualizzazione tornei in corso
4. Click sul torneo d'interesse
5. Visualizzazione partite del turno corrente
6. Click sulla partita in cui inserire il punteggio
7. Inserimento punteggio
8. Reindirizzamento alla lista delle partite per il turno corrente

5.6 UCY: Generazione classifica

Breve descrizione: l'amministratore visualizzerà la sezione *tornei in corso*, selezionerà il torneo a cui è interessato e selezionerà il tasto *classifica*.
Precondizione:

- Amministratore:
  - Aver effettuato il log-in
  - Aver creato un torneo
  - Aver avviato il torneo

Attori coinvolti :

- Amministratore
- Sistema

Risposta del sistema:

- *successo*: classifica per il torneo selezionato generato. Il sistema farà la lista dei partecipanti con i punti del turno i-esimo
- *fallimento*:classifica per il torneo selezionato non generato. Quindi il sistema farà visualizzare un messaggio di errore nella pagina di generazione della classifica

PostCondizione:

- successo: Il sistema farà visualizzare la lista dei partecipanti con i punti del turno corrente. Si rimarrà nella stessa pagina
- fallimento: Il sistema non farà visualizzare a schermo la classifica del torneo

Procedimento :

1. Log-in
2. Click sulla sezione tornei
3. Visualizzazione *tornei*
4. Click sul torneo d'interesse
5. Click sulla voce *classifica*
6. Visualizzazione lista partecipanti con i punteggi ordinati

5.7 UML Component Diagram

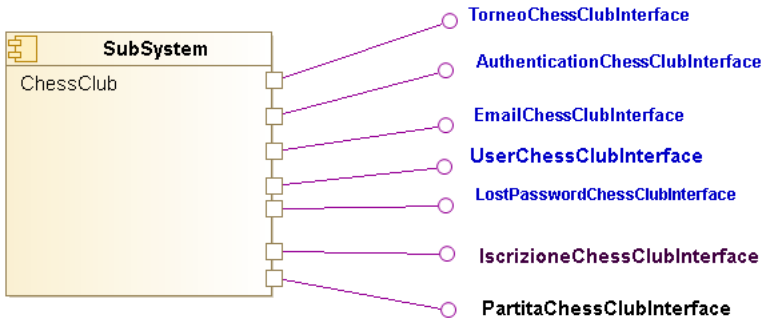

Figura 57: ClassDiagram2

I casi d'uso che abbiamo discusso nella iterazione2 sono state mappate in metodi resi disponibili dalle seguenti interfacce, nello specifico vediamo delle nuove interfacce:

- *IscrizioneChessClubInterface*
  - UC5: Preiscrizione Torneo
  - UC6: Cancellazione Preiscrizione Torneo
- *PartitaChessClubInterface*
- UC18: Inserimento Punteggio Torneo

Mentre nell'interfaccia *TorneoChessClubInterface* abbiamo aggiunto dei metodi per i casi d'uso:

- UC19: Generazione Turni
- UCY: Generazione Classifica

5.8 UML Class Diagram per interfacce lato server

Ora mettiamo in evidenza le interfacce del sistema con le relative funzioni e dati di input e di output.

5.8.1 TorneoChessClubInterface

Rispetto all'iterazione 1, l'interfaccia possiede dei nuovi metodi.

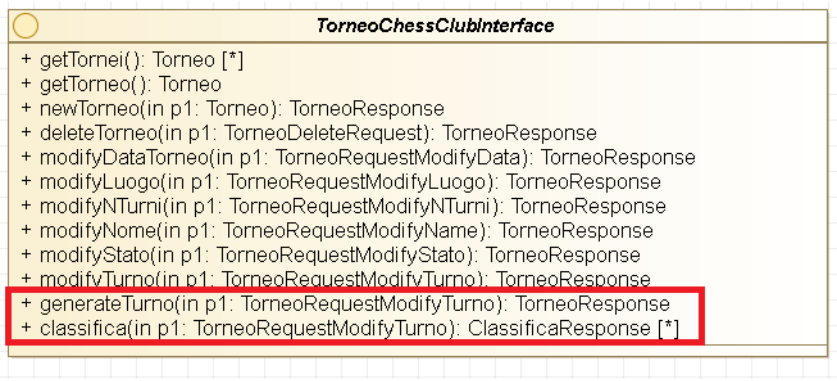

Figura 58: TorneoClubChessInterfaceModificato

5.8.2 PartitaChessClubInterface

Tramite questa interfaccia, definiamo il prototipo di funzioni che dovranno essere implementate per permettere di gestire le partite,visualizzare la lista delle partite che si sono svolte e i loro risultati.

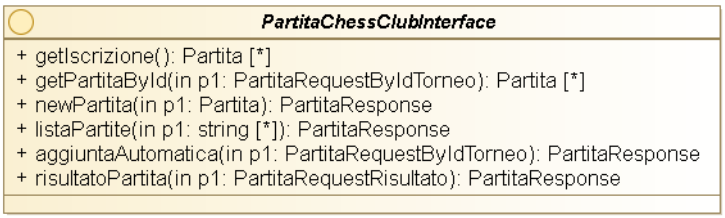

Figura 59: PartitaChessClubInterface

5.8.3 IscrizioneChessClubInterface

Tramite questa interfaccia, definiamo il prototipo di funzioni che dovranno essere implementate per permettere le operazioni di iscrizione.Inoltre,sarà possibile usare le iscrizioni per gestire i tornei.

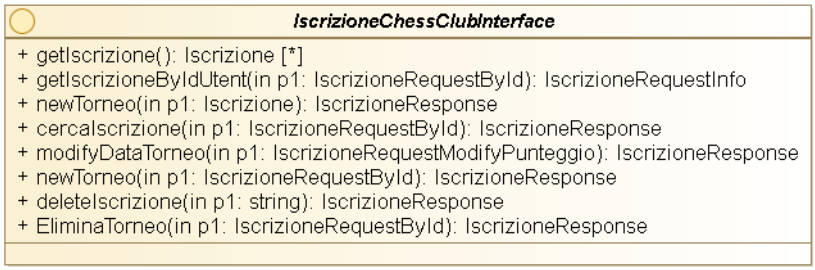

Figura 60: IscrizioneChessClubInterface

5.9 UML Class Diagram per tipi di dato lato server

Qui sono presenti i tipi di dato necessari per lo sviluppo dell'applicazione. Per la progettazione dei tipi e delle interfacce sono state seguite le euristiche di design. Qui, di seguito troviamo i tipi di dato che sono stati implementati nell'iterazione 2.

5.9.1 Classi per Partita

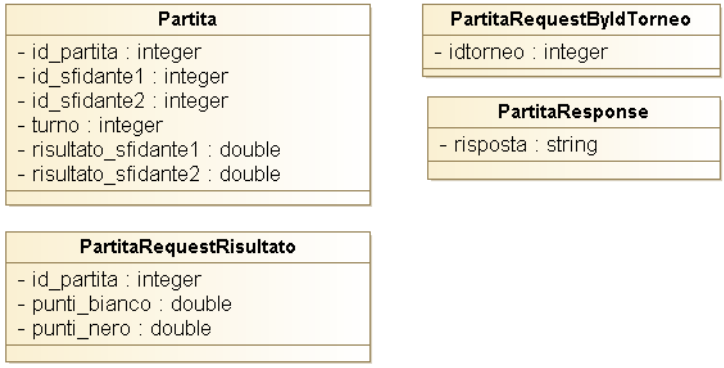

Figura 61: StrutturaDatiPartita

5.9.2 Classi per Iscrizione

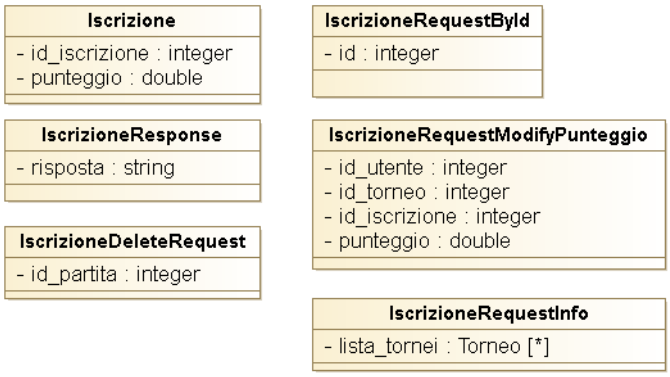

Figura 62: StrutturaDatiIscrizione

5.10 UML Class Diagram per interfacce lato client

Ora mettiamo in evidenza le interfacce lato client con le relative funzioni e dati di input e di output

5.10.1 AdministratorAPICall

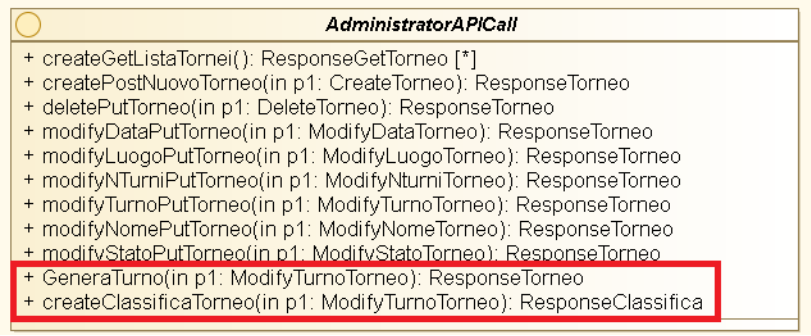

Figura 63: AdministratorAPICall aggiornata

5.10.2 AdministratorPartitaAPICall

Figura 64: AdministratorAPICall

5.10.3 UserIscrizioneAPICall

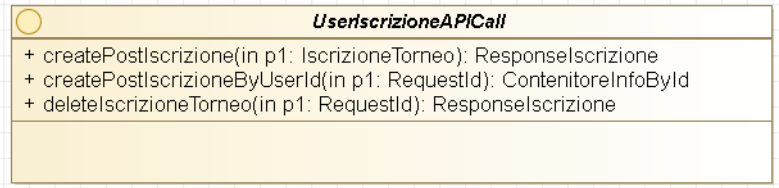

Figura 65: UserIscrizioneAPICall

5.10.4 UserPartitaAPICall

Figura 66: UserPartitaAPICall

5.11 UML Class Diagram per tipi di dato lato client

5.11.1 Classi per AdministratorAPICall aggiornata

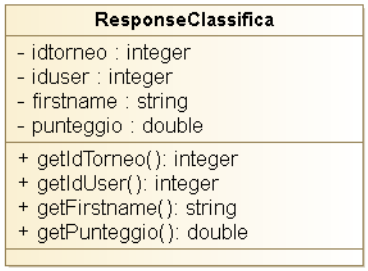

Figura 67: StrutturaDatiTorneoit2

5.11.2 Classi per AdministratorPartitaAPICall e UserPartitaAPICall

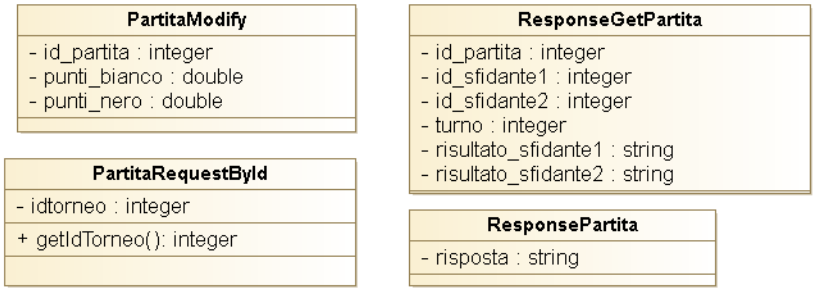

Figura 68: StrutturaDatiTorneoit2

5.11.3 Classi per UserIscrizioneAPICall

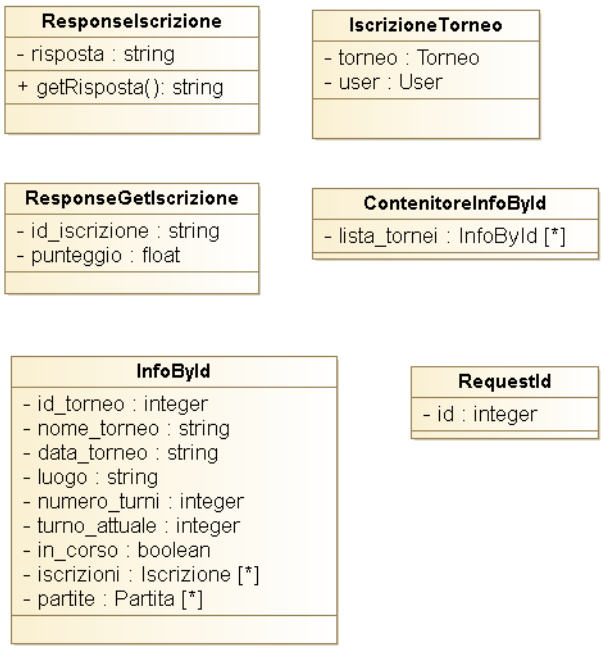

Figura 69: StruttureDatiIscrizione

5.11.4 Vista generale lato Client e Server

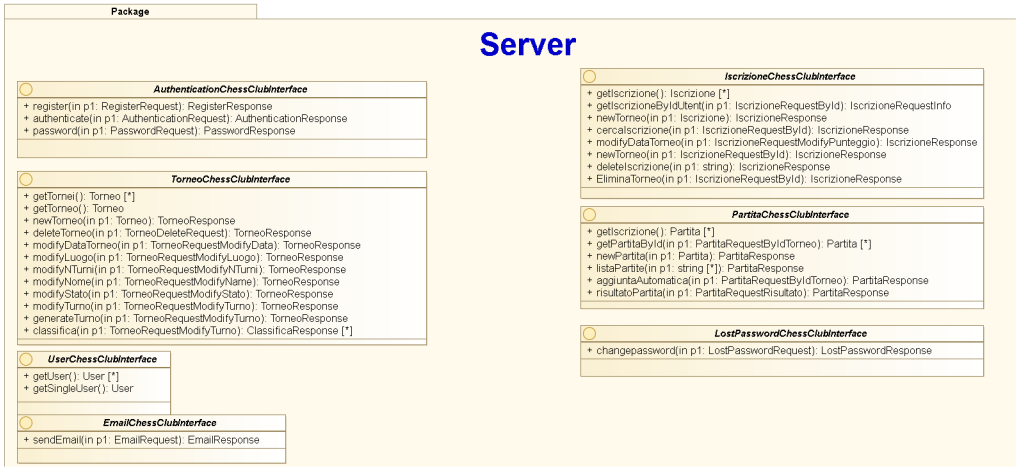

Figura 70: UML Diagram Class lato Server

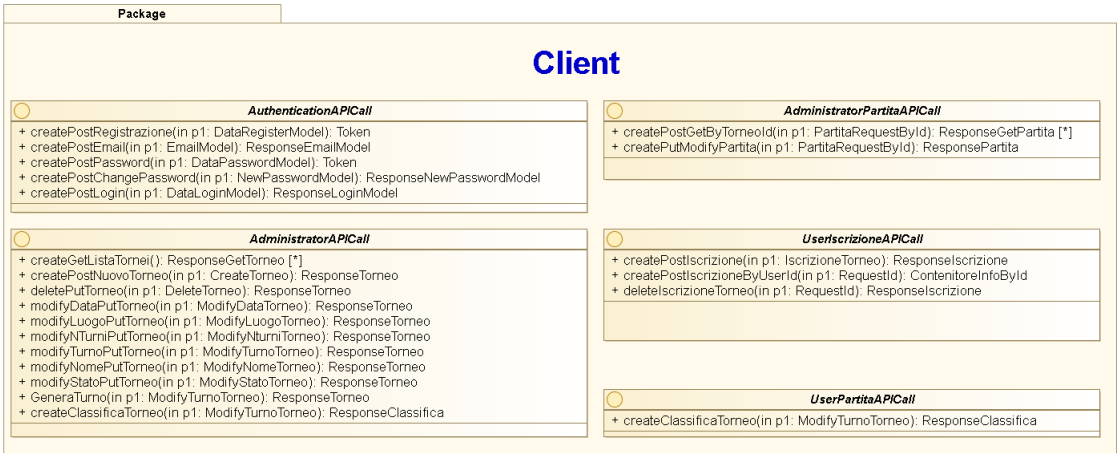

Figura 71: UML Diagram Class lato Client

5.12 UML Deployment Diagram

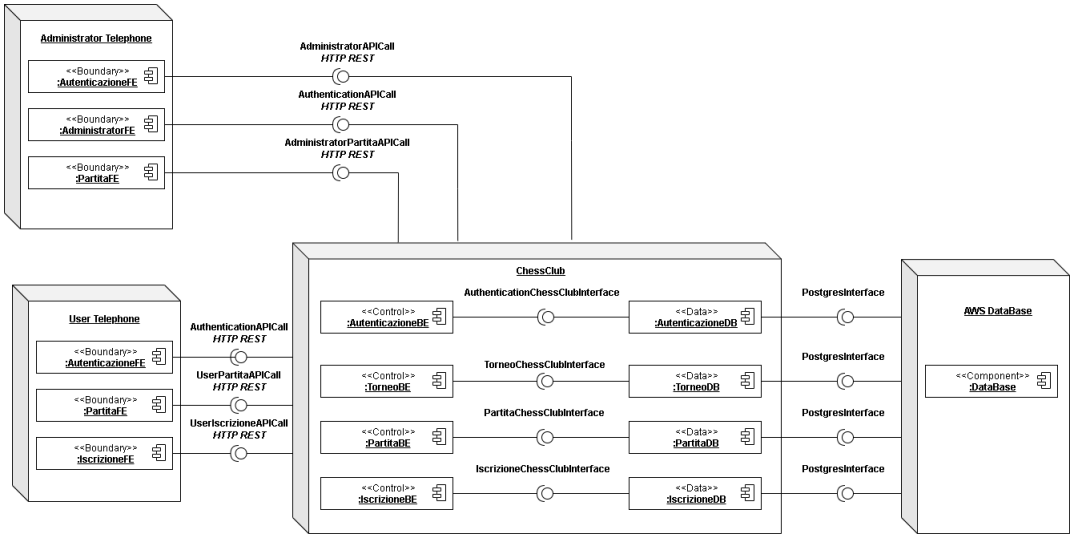

Figura 72: UML Deployment Diagram

5.13 Testing

5.13.1 Analisi statica: Unit test

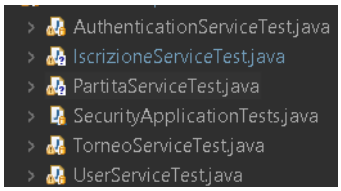

Figura 73: File di test

5.13.2 Test IscrizioneService

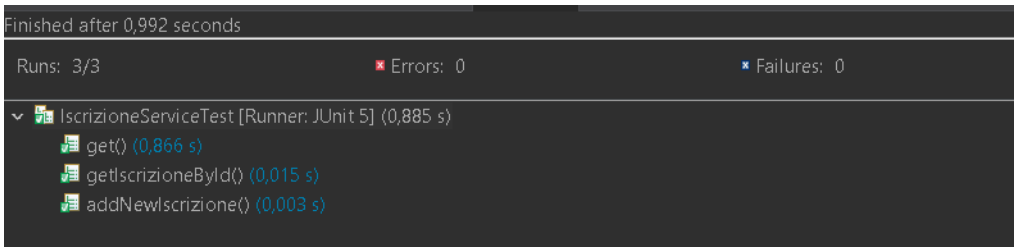

Figura 74: Run test Iscrizione

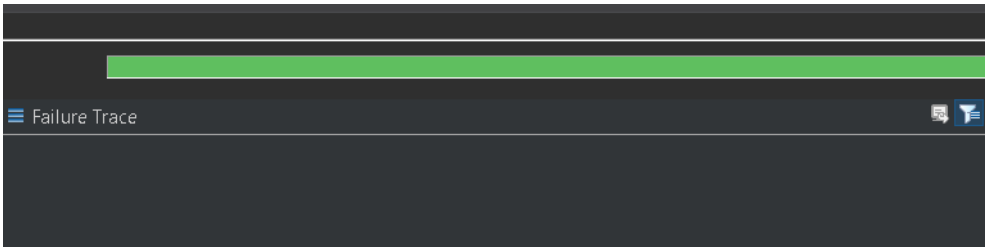

Figura 75: Failure trace Iscrizione

5.13.3 Test PartitaService

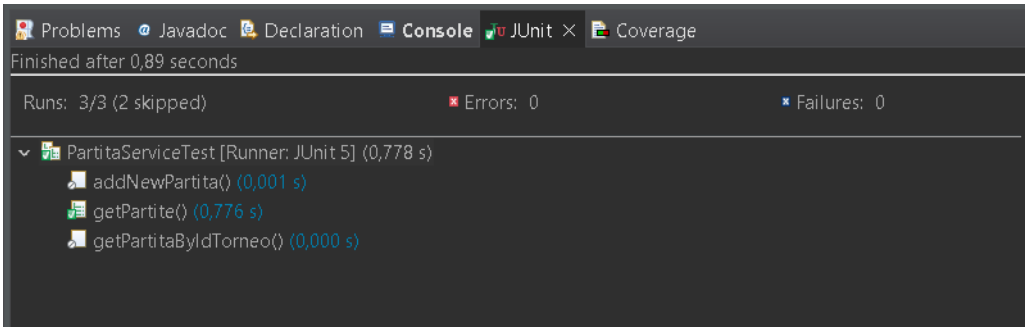

Figura 76: Run test Partita

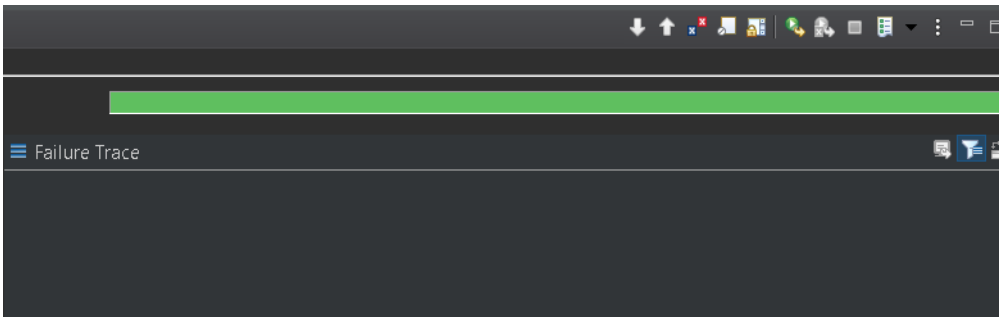

Figura 77: Failure trace Partita

5.13.4 Analisi dinamica

Iscrizione

- **RiempiTorneo**

  Input Request:

  – id

  Output Response:

  – risposta

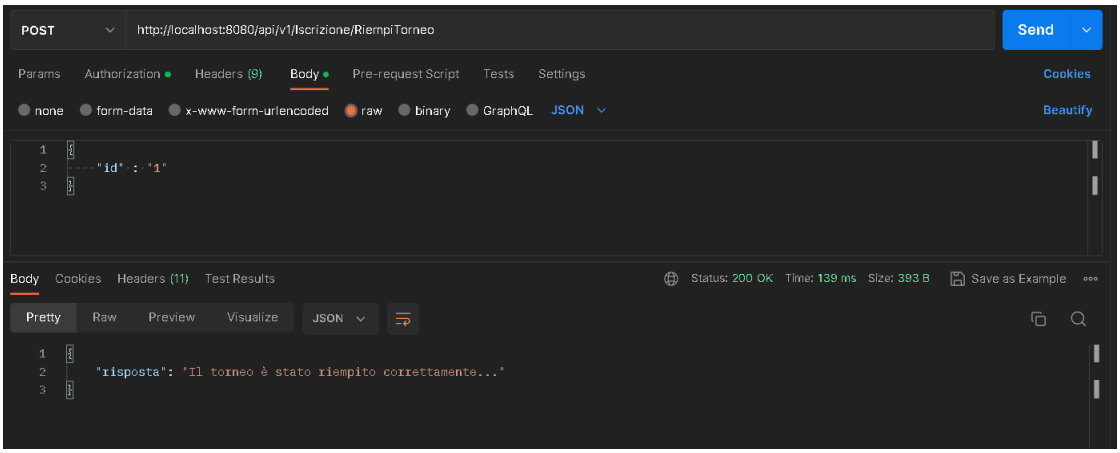

Figura 78: RiempiTorneo

Torneo

- **ModificaStato**

  Input Request:

  – id_torneo

  – in_corso

  Output Response:

  – risposta

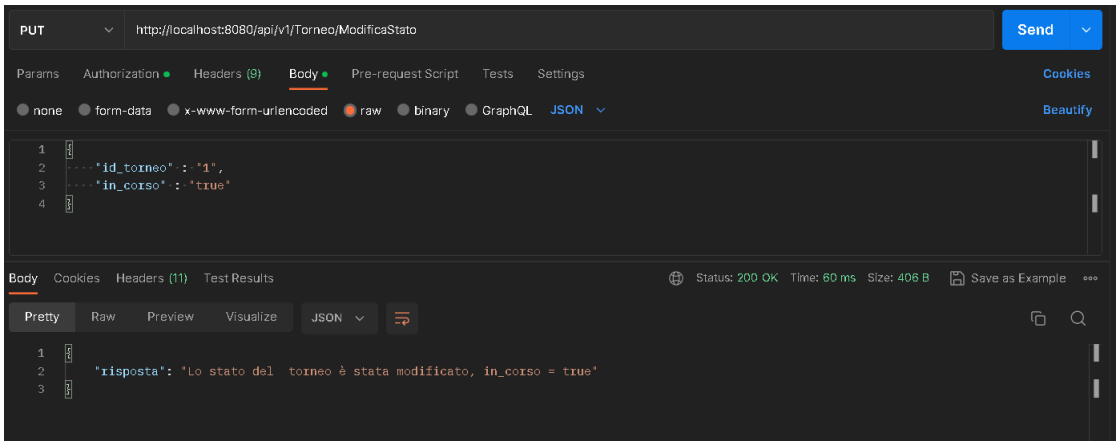

Figura 79: ModificaStato

- **Get**

  Input Request:

  – nessun input

  Output Response:

  – id_torneo

  – nome_torneo

  – data_torneo

  – luogo

  – numero_turni

  – turno_attuale

  – in_corso

  – iscrizioni

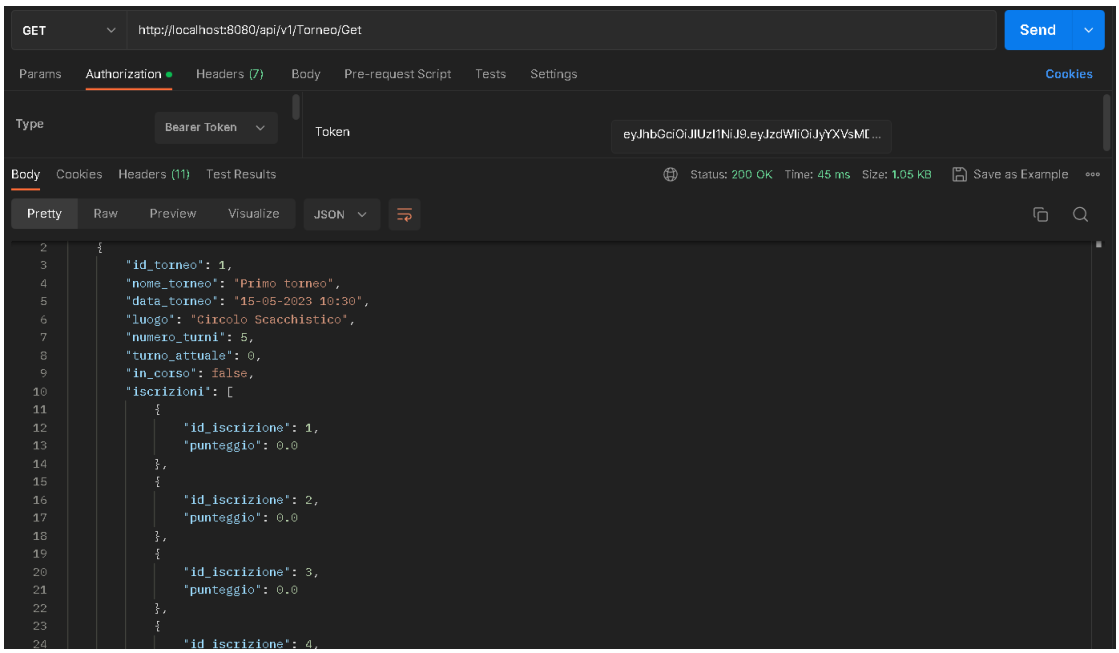

Figura 80: get

- **GeneraTurno**

  Input Request:

  – nessun input

  Output Response:

  – risposta

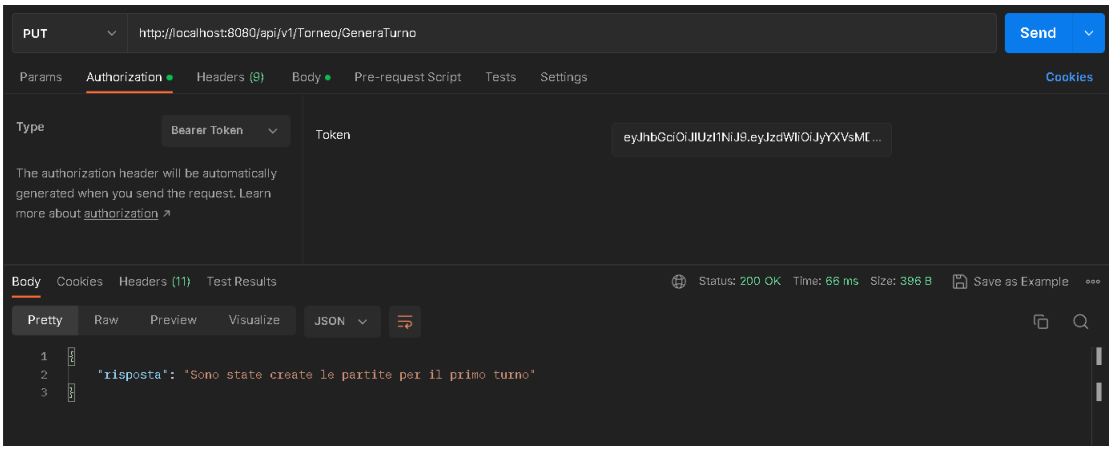

Figura 81: GeneraTurno

Partita

- *ModifyRisultato*

  Input Request:

  – id_partita

  – punti_bianco

  – punti_nero

  Output Response:

  – risposta

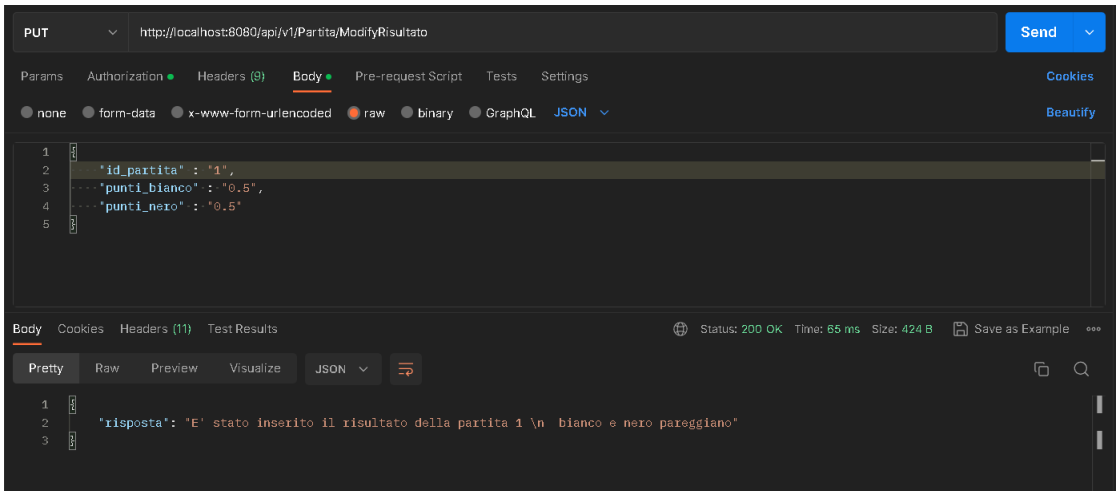

Figura 82: ModifyRisultato 1

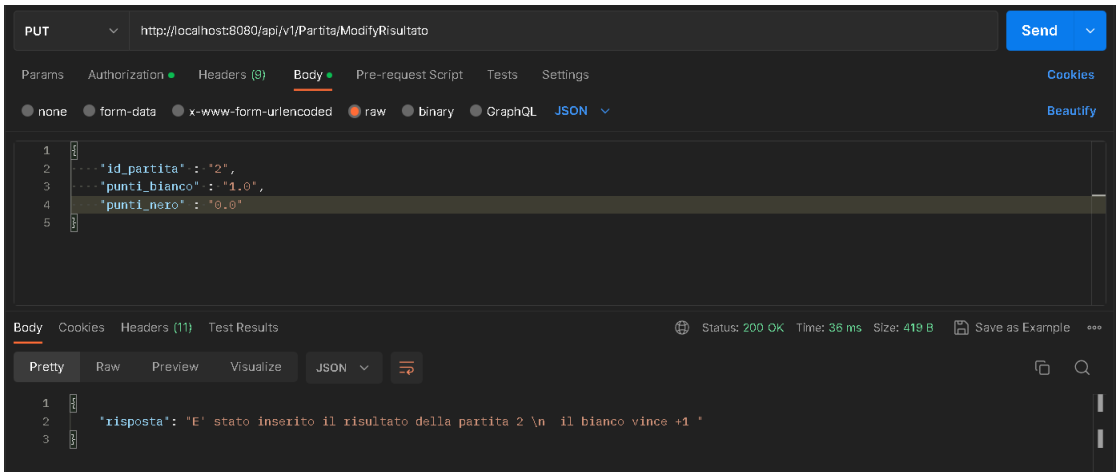

Figura 83: ModifyRisultato 2

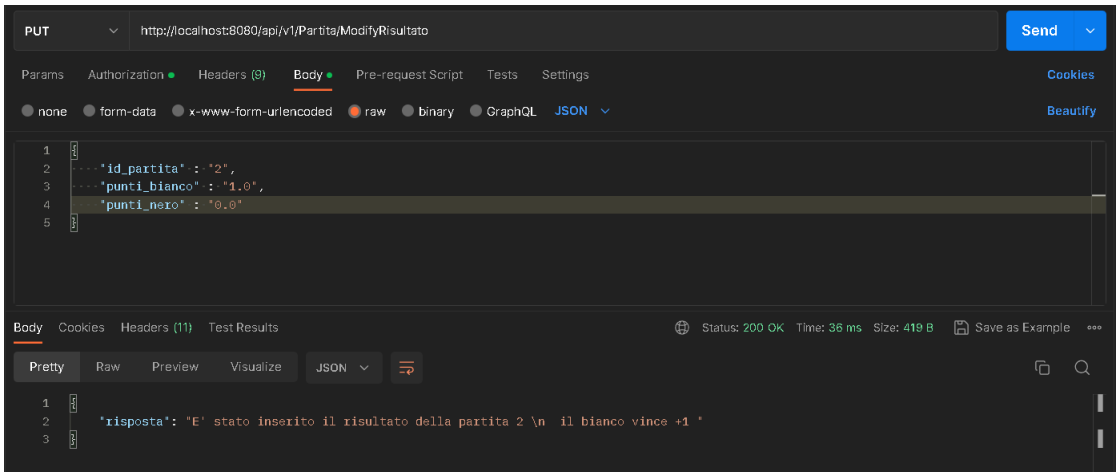

Figura 84: ModifyRisultato 3

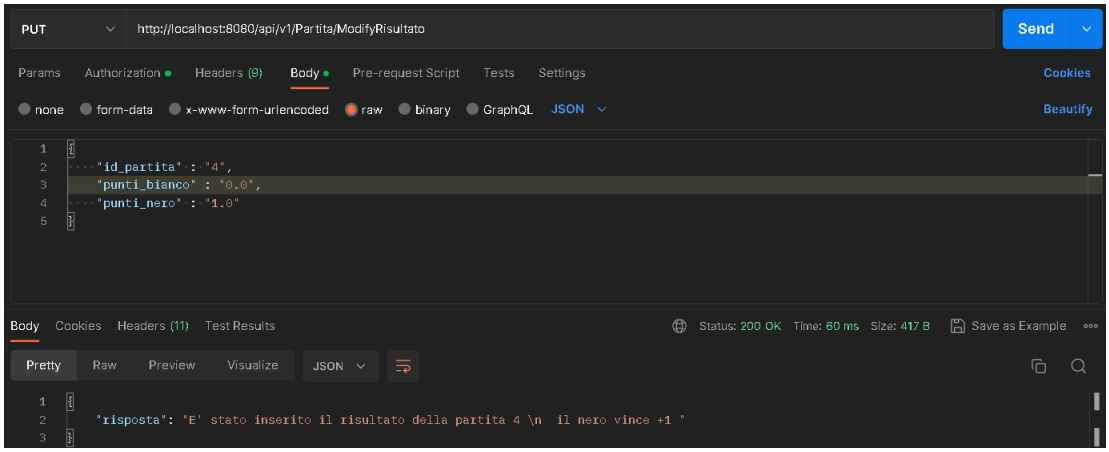

Figura 85: ModifyRisultato 4

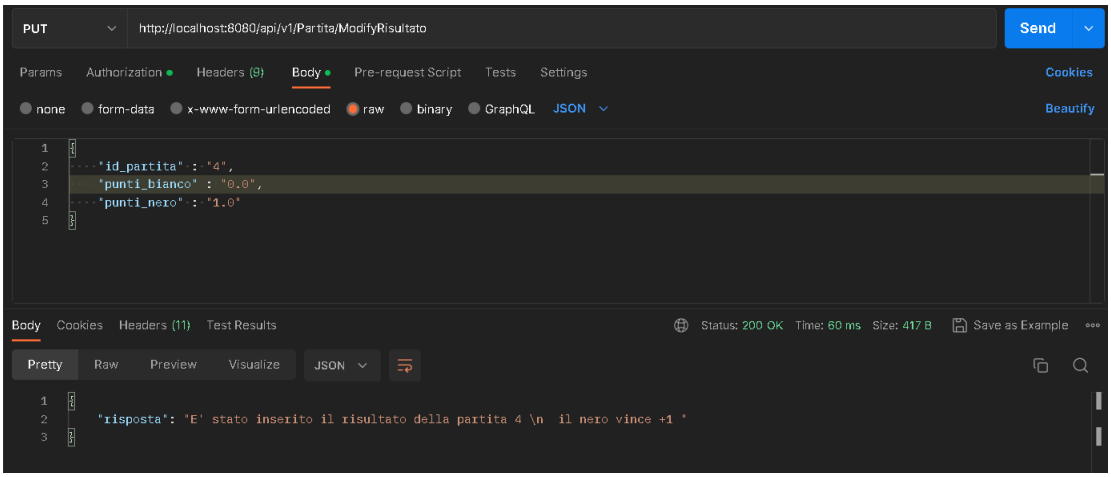

Figura 86: ModifyRisultato 5

Torneo

- *Classifica Input Request*:
  - id_torneo
  - turno_attuale

  Output Response:
  - lista di:
    - idtorneo
    - iduser
    - firstname
    - punteggio

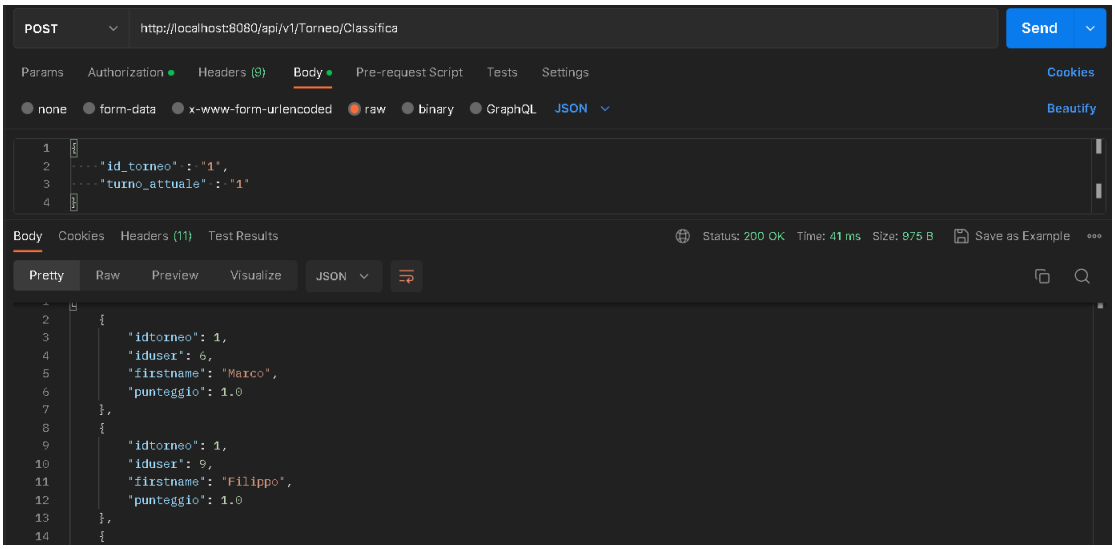

Figura 87: Classifica

Figura 88: Classifica

```
    "punteggio": 0.5
  },
  {
    "idtorneo": 1,
    "iduser": 12,
    "firstname": "Giulio",
    "punteggio": 0.5
  },
  {
    "idtorneo": 1,
    "iduser": 8,
    "firstname": "Lorenzo",
    "punteggio": 0.0
  },
  {
    "idtorneo": 1,
    "iduser": 10,
    "firstname": "Youssef",
    "punteggio": 0.0
  },
  {
    "idtorneo": 1,
    "iduser": 13,
    "firstname": "Michele",
    "punteggio": 0.0
  }
]
```


Figura 89: Classifica

Iscrizione

- *IscrizioneTorneo*

  Input Request:

  - user
  - torneo

  Output Response:

  - risposta

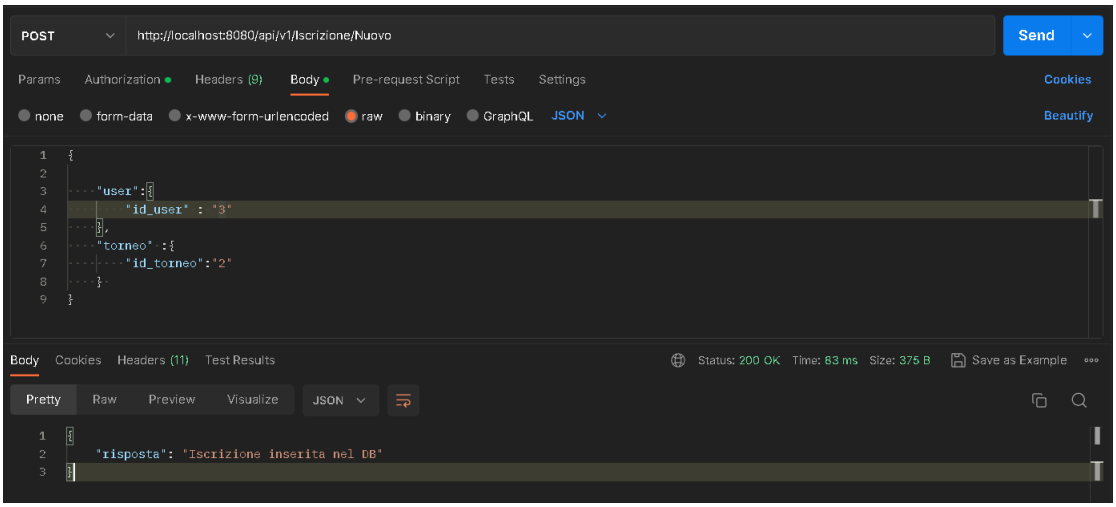

Figura 90: Classifica

Torneo

- *GetByTorneoId*

  Input Request:

  - id_torneo

  Output Response:

  - id_torneo
  - nome_torneo
  - data_torneo
  - luogo
  - numero_turni
  - turno_attuale
  - in_corso
  - iscrizioni
  - partite

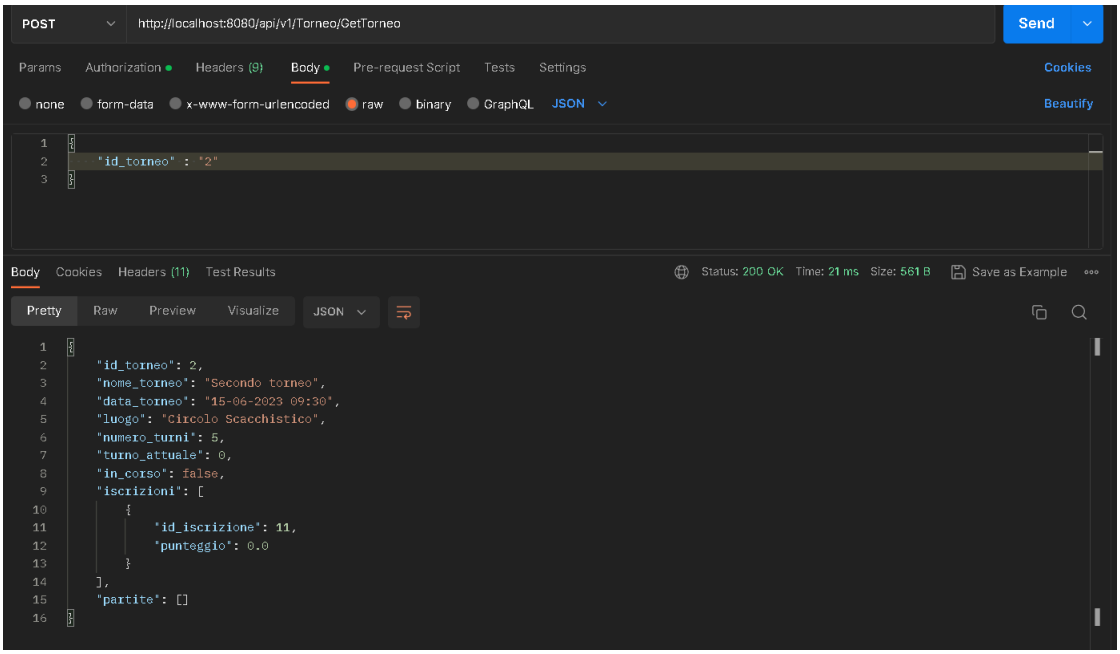

Figura 91: get

5.14 Algoritmo

L'algoritmo di ordinamento *MergeSort* è stato usato nei seguenti servizi:

- *Generazione turno*
- *Classifica*

5.14.1 Generazione turno

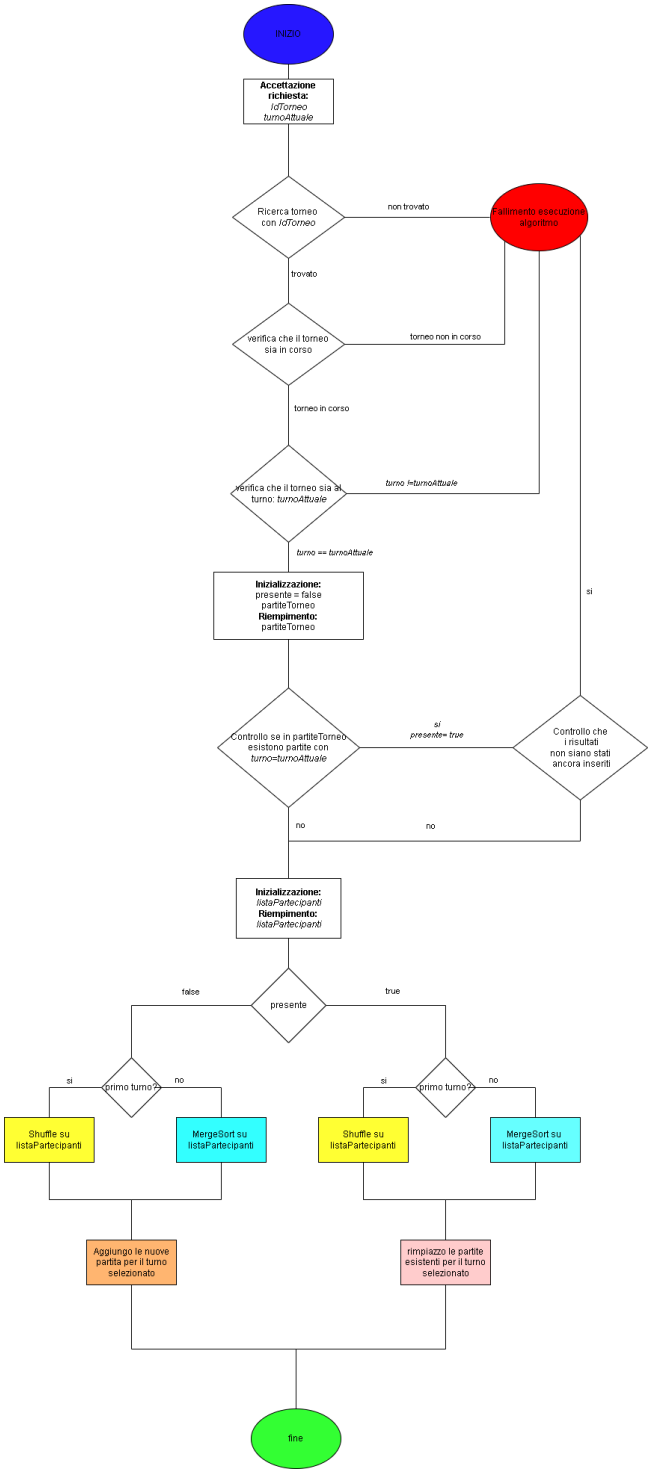

Figura 92: FlowChart di Generazione turno

- Controllo se il torneo esiste
- Controllo che la generazione del turno sia all'interno del range dei turni del torneo
- Verifica che la generazione del turno avvenga se non sono stati già inseriti dei risultati, variabile booleana utilizzata per verificare questo punto

```
public TorneoResponse generaPartiteTurno(TorneoRequestModifyTurno richiesta){

    Torneo torneo =  torneoRepository.findById(richiesta.getId_torneo()).orElseThrow(()-> new RuntimeException("Torneo con questo id non esiste nel

    // Devo controllare che per questa richiesta di generazione partite del turno non ci sia già stata, anzi devo sovrascriverle
    // nel caso

    if(torneo.isIn_corso() == false) {

        return TorneoResponse.builder()
                .risposta("Il torneo non è in corso, non puoi generare turni")
                .build();

    }

    if(torneo.getNumero_turni() < richiesta.getTurno_attuale() || richiesta.getTurno_attuale() <= 0 ) {

        return TorneoResponse.builder()
                .risposta("Il torneo inizia i turni da 1 a " + torneo.getNumero_turni() + "...")
                .build();
    }

    // Da verificare ....

    if(torneo.getTurno_attuale() != richiesta.getTurno_attuale()){

        return TorneoResponse.builder()
                .risposta("Il turno del torneo non combacia con il turno richiesto...")
                .build();

    }
```

Figura 93: Algoritmo di genera partite

- Controllo il turno per il quale si vuole generare il turno
  - se è il primo turno: utilizzo *Collections.shuffle()*
  - se non è il primo turno: utilizzo *MergeSort()*

```
List<Partita> partite_torneo = new ArrayList<>();

List<Partita> lista_replace = new ArrayList<>();

partite_torneo = torneo.getPartite();

boolean presente = false;

for(int i =0 ; i<partite_torneo.size(); i++){

    if(partite_torneo.get(i).getTurno().equals(richiesta.getTurno_attuale())) {

        presente = true;
        System.out.println("sono già state generate partite per questo turno");


        if(partite_torneo.get(i).getRisultato_sfidante1() == null || partite_torneo.get(i).getRisultato_sfidante2()== null) {
            System.out.println("non faccio nulla");
        }
        else {

            return TorneoResponse.builder()
            .risposta("Impossibile cambiare turno perché alcune partite sono già terminate")
            .build();
        }

        }

}
```

O(n)

Figura 94: Algorimo di genera partite

- – se *presente* == *true* Creazione delle partite per il turno e successivo salvataggio
  – se *presente* == *false* Rigenerazione del turno perché non sono ancora stati inseriti dei risultati

```
if(presente == false) {

    // La mia lista è stata ordinata in base al punteggio... Ora posso generare il nuovo turno... con le nuove partite

    if(torneo.getTurno_attuale() == 1) {

        Collections.shuffle(partecipanti); // ho un ordinamento randomico
    }


    else {

        MergeSort lista_merge = new MergeSort(partecipanti);
        lista_merge.sortGivenArray();
    }

    for(int i = 0; i<partecipanti.size();i=i+2){

        // Qui prendo i e i+1 poi al ciclo successivo prendere i+2 ed i+2+1

        Partita partita = new Partita();

        partita.setId_sfidante1(partecipanti.get(i).getUser().getId_user());
        partita.setId_sfidante2(partecipanti.get(i+1).getUser().getId_user());
        partita.setTorneo(partecipanti.get(i).getTorneo());
        partita.setTurno(richiesta.getTurno_attuale());
        partita.setTorneo(torneo);

        torneo.AddPartite(partita);
        partite_torneo.add(partita);

        // così torneo tiene traccia che sono state aggiunte queste partite

        partitaRepository.save(partita);

        }

    return TorneoResponse.builder()
            .risposta("Sono state create le partite per il primo turno")
            .build();

}
```

O(n)

O(n log n)

O(n)

Figura 95: Algoritmo di genera partite

Figura 96: Algoritmo di genera partite

5.14.2 Classifica

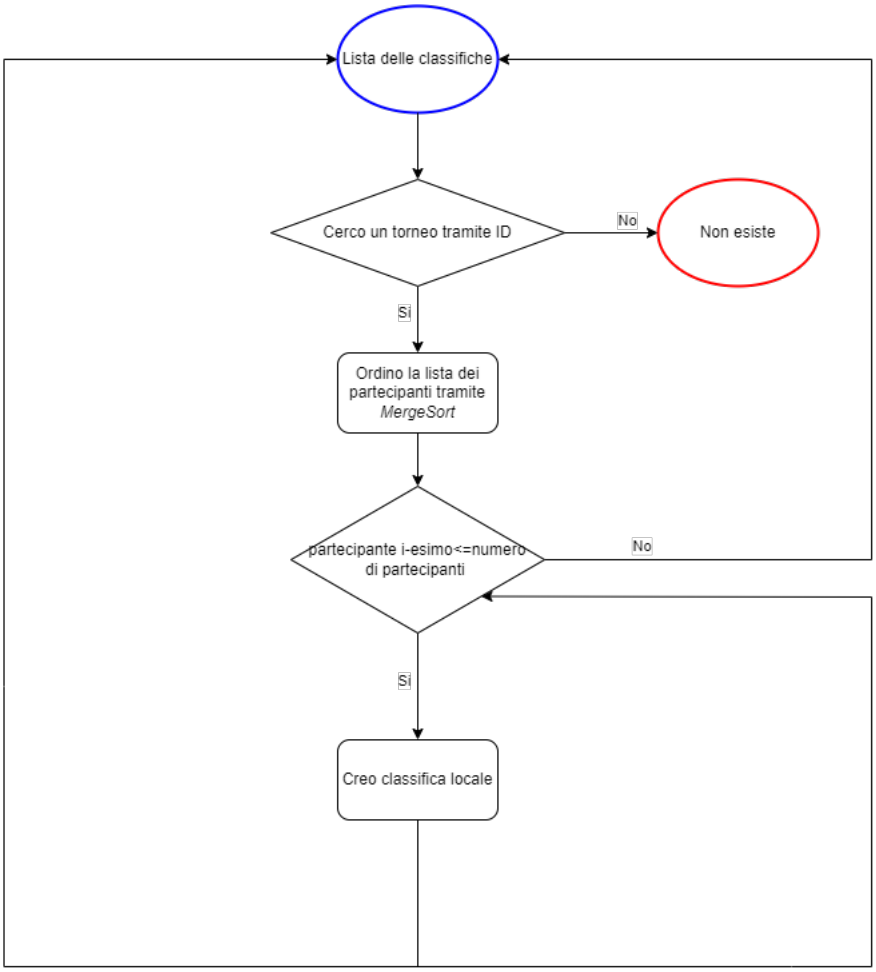

Figura 97: FlowChart di Classifica

- Creazione lista classifica
- Applicazione algoritmo di *MergeSort* sui partecipanti al torneo
- ciclo for sui partecipanti ed inserimento in lista classifica
- return lista classifica

Figura 98: Algoritmo Classifica