Iterazione 1

1.0 Introduzione

La prima iterazione del progetto *TeamManager* ha l'obiettivo di analizzare e implementare le funzionalità di base dell'applicazione, ponendo le basi per una gestione digitale efficace di una squadra di calcio. In questa fase si è deciso di concentrarsi sull'implementazione dei **casi d'uso principali** legati alla gestione operativa quotidiana da parte dell'allenatore e dei giocatori.
Nello specifico, questa iterazione prevede la progettazione e realizzazione degli scenari principali di successo per i seguenti casi d'uso:

- **UC1: Gestione della rosa** per aggiungere, modificare o rimuovere giocatori dalla squadra;
- **UC2: Gestione eventi** per creare e aggiornare allenamenti e partite amichevoli nel calendario della squadra;
- **UC3: Gestione disponibilità** per consentire ai giocatori di comunicare la propria presenza o assenza agli eventi;
- **UC7: Gestisci Giocatore** consente di creare e mantenere un archivio generale dei giocatori, inserendone le informazioni anagrafiche e sportive. I giocatori inseriti in questo archivio potranno successivamente essere aggiunti alla rosa tramite UC1.

Per semplicità, **in questa iterazione non verrà implementato il sistema di notifiche automatiche**, inizialmente previsto in fase di ideazione, in quanto richiederebbe un sistema di gestione degli eventi e degli utenti in background che si è scelto di non fare.

Inoltre, in questa iterazione non è prevista l'integrazione di un sistema di login o autenticazione complesso basato su framework esterni, poiché non necessario in questa fase iniziale del progetto. Tuttavia, è stata realizzata una **gestione simulata della sessione** tramite la classe Sessione e la classe Utente, per distinguere correttamente i **Giocatori_Rosa** dall'**Allenatore**, permettendo di tracciare l'utente attualmente attivo durante l'interazione con il sistema.
L'obiettivo di questa iterazione è quindi fornire una **prima versione funzionante** dell'applicazione, incentrata sulla gestione dei dati principali e sulla corretta interazione tra le entità (giocatori, rosa,

eventi, disponibilità), costituendo le **fondamenta per lo sviluppo incrementale** delle funzionalità nelle iterazioni successive.

2.0 Fase di Analisi

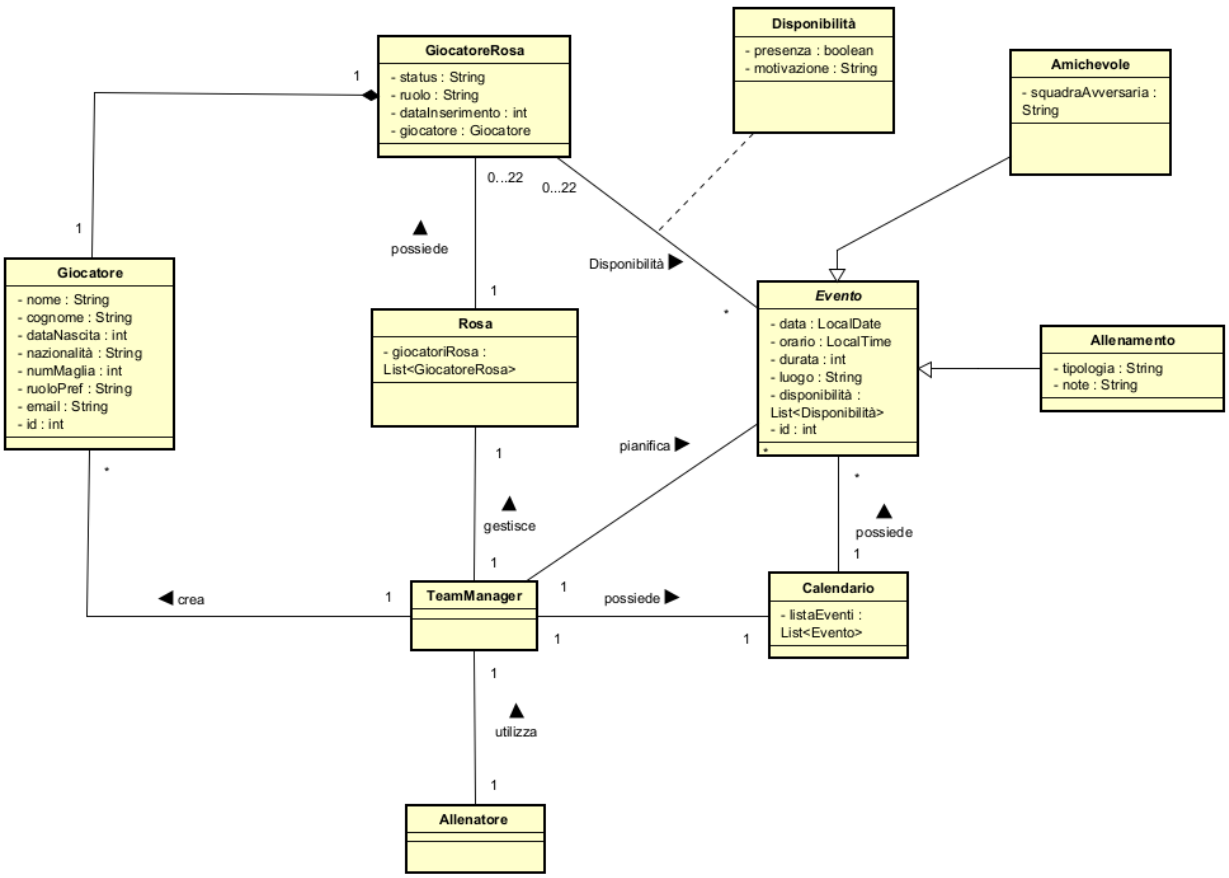
In questa fase vengono individuate le principali **classi concettuali** coinvolte nei casi d'uso previsti per la prima iterazione. Le classi concettuali rappresentano gli elementi significativi del dominio del problema e saranno successivamente trasformate in componenti software all'interno del modello di dominio.

Di seguito sono elencate le classi concettuali emerse dall'analisi dei casi d'uso **UC1, UC2, UC3 e UC7**, accompagnate da una breve descrizione:

Classe	Descrizione
Giocatore	Rappresenta un individuo iscrivibile alla squadra. Contiene informazioni anagrafiche (nome, cognome, e-mail), tecniche (ruolo).
Giocatore_rosa	Rappresenta il giocatore pronto ad essere inserito all'interno della rosa a cui vengono associati parametri esclusivi.
Rosa	Contiene i giocatori attualmente attivi nella squadra. È gestita dall'allenatore ed è derivata dall'archivio generale.
Evento	Rappresenta una sessione di squadra (allenamento o partita amichevole), con informazioni su data, orario, luogo e obiettivo.
Disponibilità	Associa un giocatore a un evento specifico, indicandone la disponibilità (presente/assente) e una possibile motivazione.
Allenatore	Figura responsabile della gestione della rosa, della creazione degli eventi e della consultazione delle disponibilità.

Classe	Descrizione
Calendario	Contenitore logico degli eventi previsti per la squadra. Permette di organizzare le attività nel tempo.
TeamManager	Rappresenta il sistema TeamManager

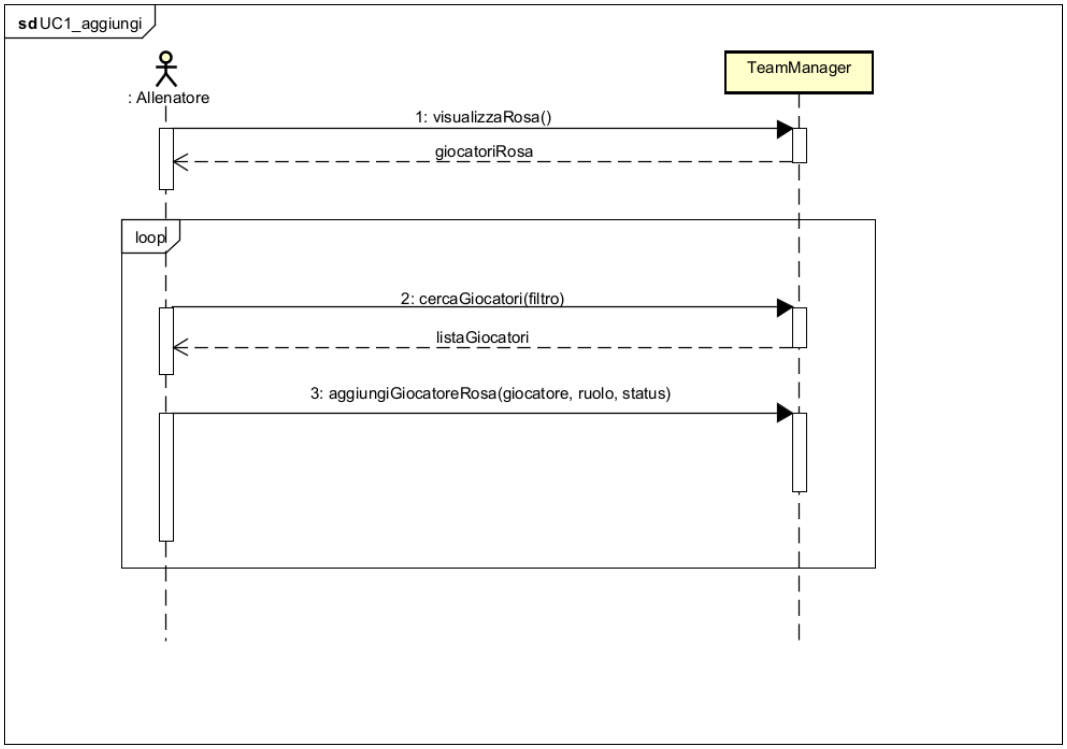
Da cui tenendo conto dei vari attributi e associazioni di ogni entità, si è generato il seguente modello di dominio:

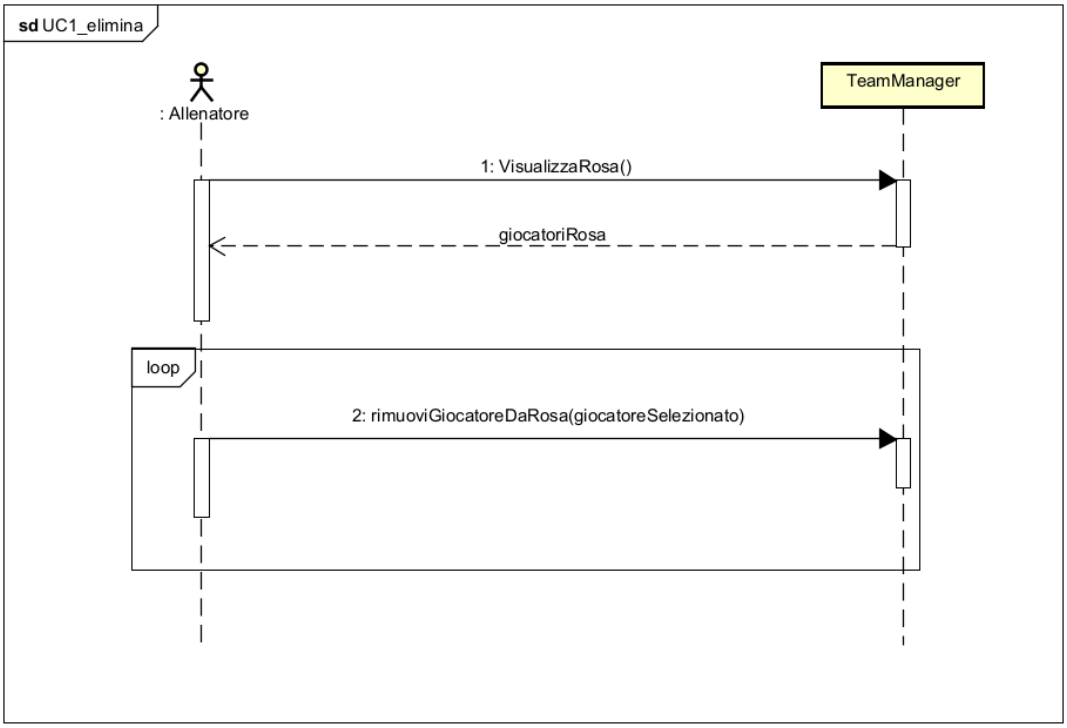


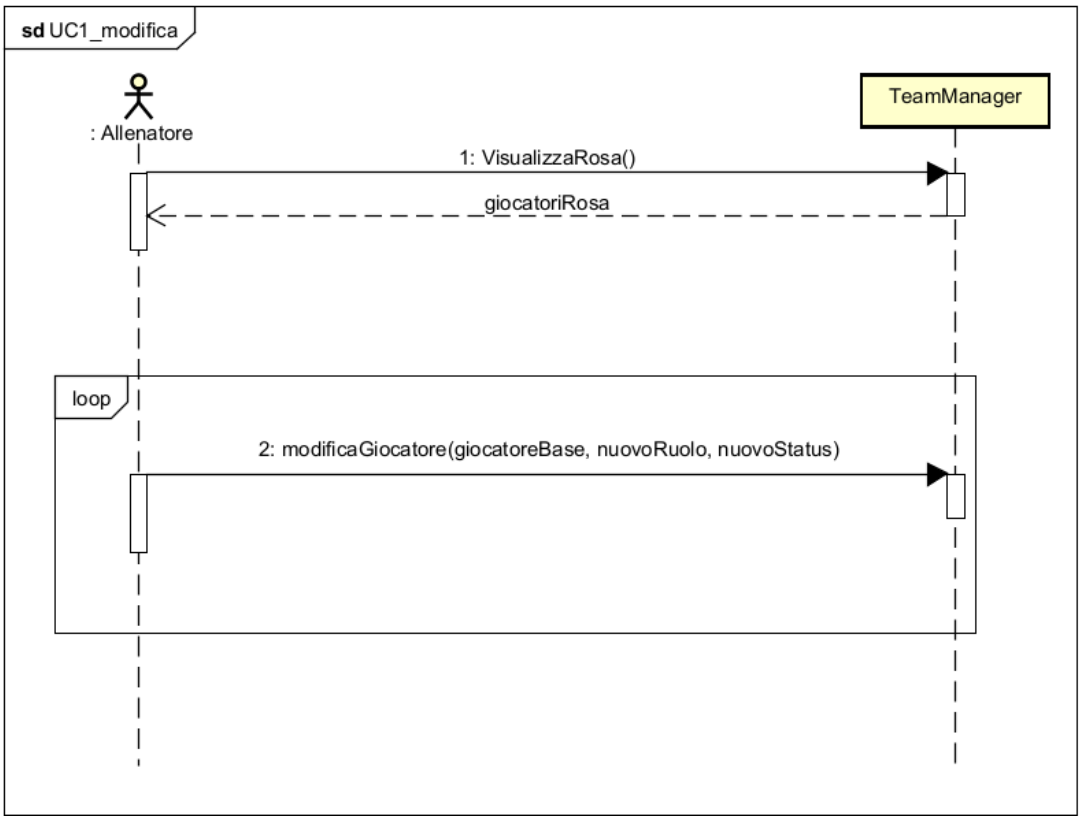
2.1 Diagrammi di Sequenza di Sistema e Contratti delle Operazioni

UC1 – Gestisci Rosa

I seguenti Diagrammi di Sequenza di Sistema illustrano le interazioni I/O dell'utente con il sistema TeamManager relative al Caso d'Uso 1 con i vari scenari di successo alternativi:







Contratti delle operazioni UC1

I contratti delle operazioni descrivono i principali metodi identificati nei SSD di UC1:

CONTRATTO 1 UC1: Gestisci Rosa

- **Operazione:** visualizzaRosa()
- **Riferimento:** UC1: Gestisci Rosa
- **Pre-condizione:**
    1. L'allenatore ha effettuato l'accesso e abbia selezionato l'opzione Rosa dal menu principale.
    2. Il file contenente i dati della rosa è accessibile e leggibile dal sistema.
    3. I dati della rosa sono stati precedentemente salvati in modo coerente.
- **Post-condizione:**
    1. Viene restituita una lista di Giocatori_Rosa;
    2. L'utente (allenatore) visualizza correttamente a schermo l'elenco dei giocatori presenti nella rosa attiva, con i relativi attributi.
    3. Nessuna modifica viene apportata ai dati.

CONTRATTO 2 UC1: Gestisci Rosa

- **Operazione:** cercaGiocatori(filtro)
- **Riferimento:** UC1: Gestisci Rosa
- **Pre-condizione:**
    1. L'archivio dei giocatori è accessibile e caricato in memoria.
    2. L'allenatore ha accesso alla barra di ricerca.

Post-condizione:

1. Viene restituito un sottoinsieme dell'archivio che corrisponde ai criteri di ricerca forniti.
2. Se nessun giocatore corrisponde ai criteri, viene restituita una lista vuota.

CONTRATTO 3 UC1: Gestisci Rosa

Operazione: aggiungiGiocatoreRosa(giocatore, ruolo, status)

Riferimento: UC1: Gestisci Rosa

Pre-condizione:

1. Il giocatore esiste nell'archivio dei giocatori.
2. Il giocatore non è già presente nella rosa (per evitare duplicati).

- **Post-condizione:**
    1. Viene creata un'istanza di GiocatoreRosa.
    2. L'istanza creata viene inserita nella giocatoriRosa in Rosa.

CONTRATTO 4 UC1: Gestisci Rosa

- **Operazione:** rimuoviGiocatoreDaRosa(giocatoreSelezionato)
- **Riferimento:** UC1: Gestisci Rosa
- **Pre-condizione:**
    1. Il giocatore è effettivamente presente nella rosa.
    2. Il file o la struttura dati contenente la rosa è accessibile.
- **Post-condizione:**
    1. Il giocatore viene rimosso dalla rosa.

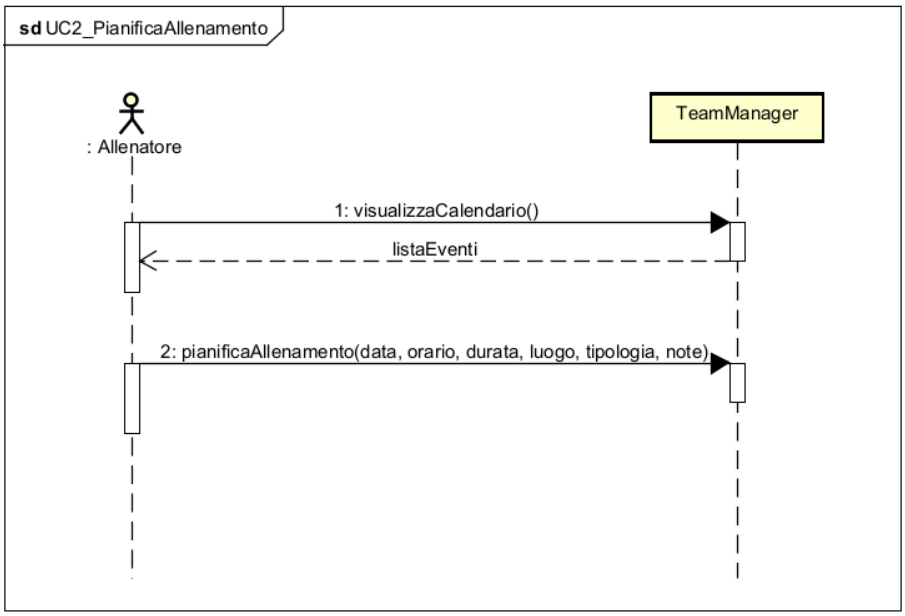
CONTRATTO 5 UC1: Gestisci Rosa

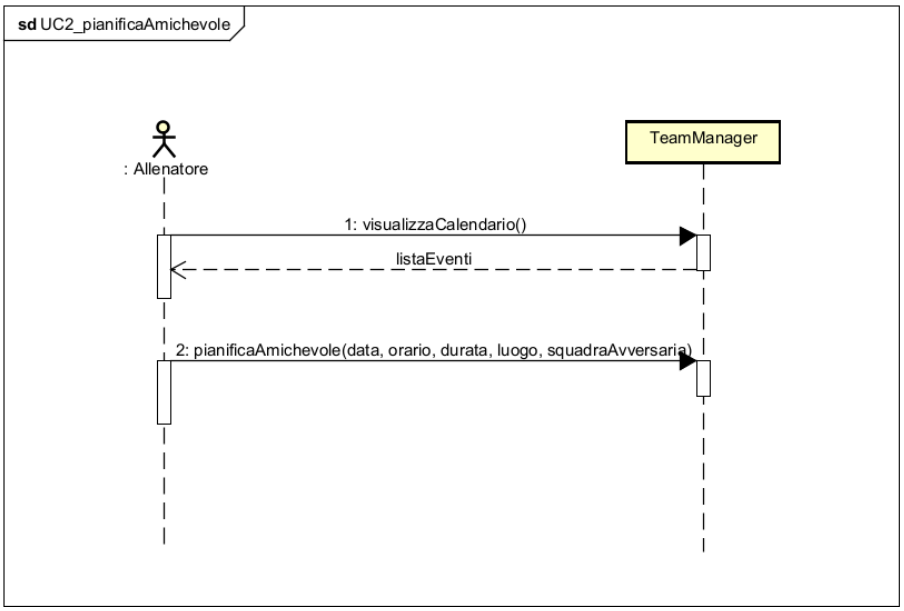
- **Operazione:** modificaGiocatore(giocatoreBase, nuovoRuolo, nuovoStatus)
- **Riferimento:** UC1: Gestisci Rosa
- **Pre-condizione:**
    1. Il giocatore da modificare è già presente nella rosa.
    2. I valori ruolo e status sono validi.

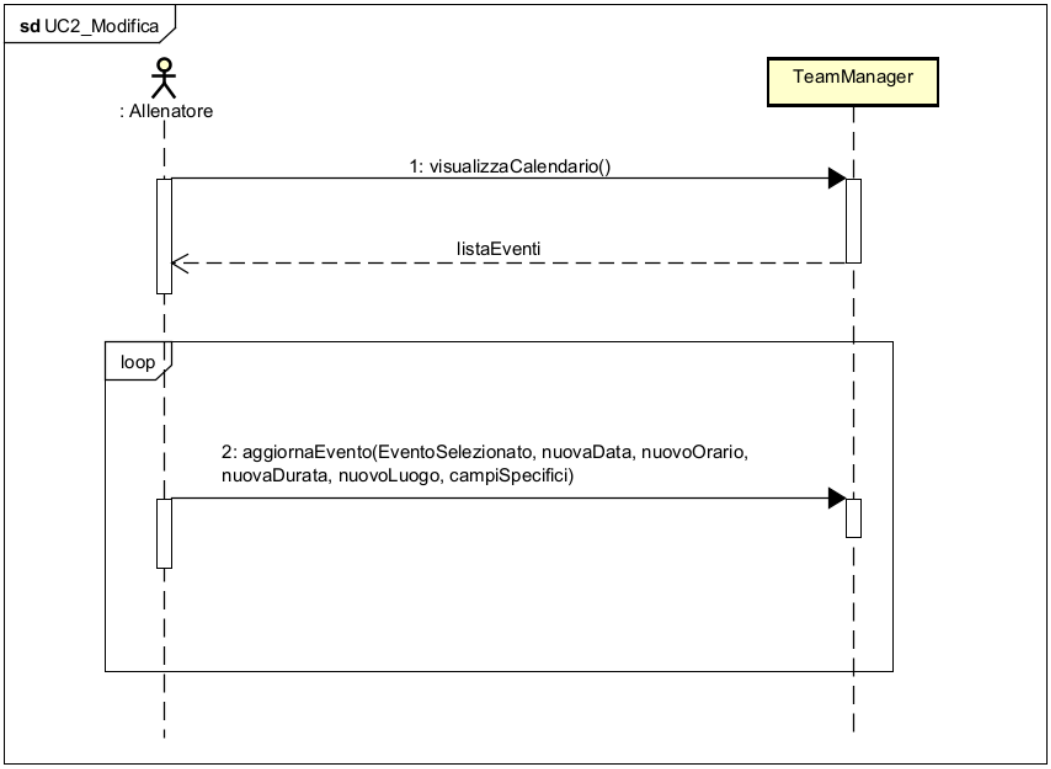
- **Post-condizione:**
    1. I nuovi valori di ruolo e/o stato vengono effettivamente modificati.

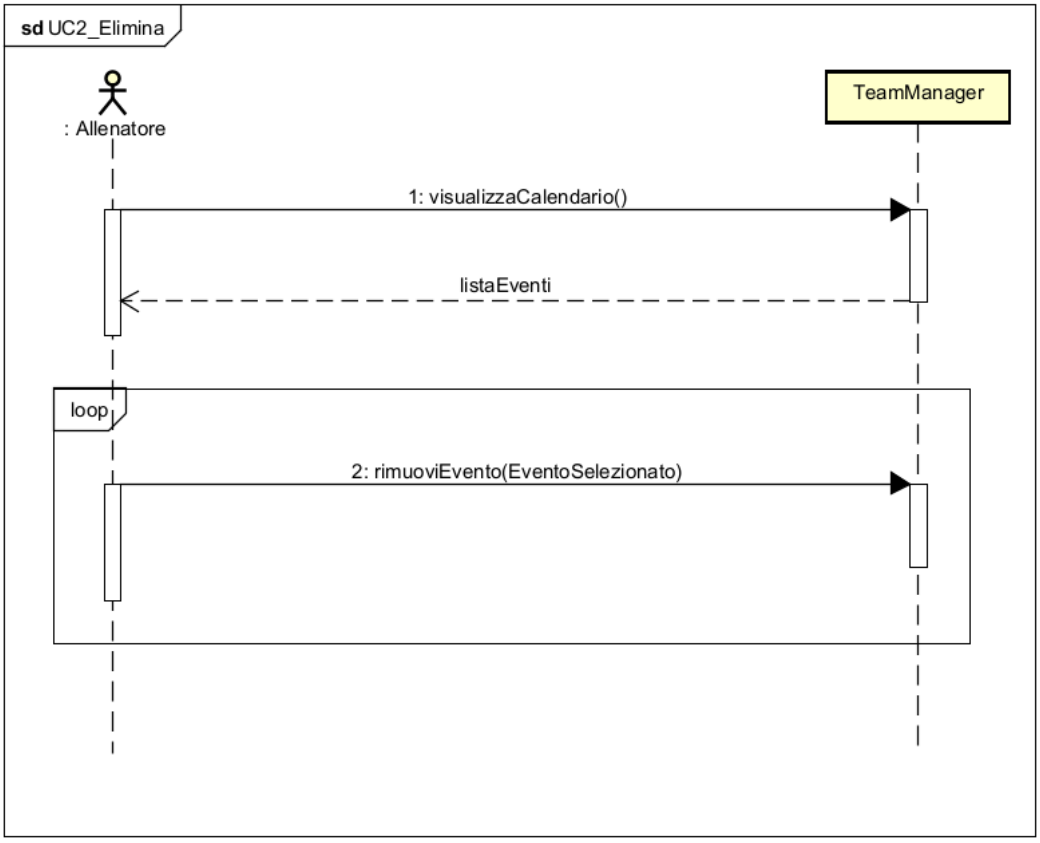
UC2 – Gestisci Evento

I seguenti Diagrammi di Sequenza di Sistema illustrano le interazioni I/O dell'utente con il sistema TeamManager relative al Caso d'Uso 2 con i vari scenari di successo alternativi:






sd UC2_Modifica
: Allenatore
TeamManager
1: visualizzaCalendario()
listaEventi
loop
2: aggiornaEvento(EventoSelezionato, nuovaData, nuovoOrario,
nuovaDurata, nuovoLuogo, campiSpecifici)


sd UC2_Elimina
: Allenatore
TeamManager
1: visualizzaCalendario()
listaEventi
loop
2: rimuoviEvento(EventoSelezionato)

Contratti delle operazioni UC2

CONTRATTO 1 UC2: Gestisci Evento

- **Operazione:** visualizzaCalendario()
- **Riferimento:** UC2-Gestisci Evento
- **Pre-condizione:**
  1. L'allenatore ha effettuato l'accesso a TeamManager e seleziona l'opzione calendario;
  2. TeamManager permette l'accesso al calendario del sistema;
  3. La lista dei vari eventi già pianificati è disponibile nella sezione calendario.
- **Post-condizione:**
  1. Viene restituita una listaEventi
  2. L'allenatore visualizza correttamente il calendario e gli eventi già pianificati
  3. Nessuna modifica viene apportata ai dati

CONTRATTO 2 UC2: Gestisci Evento

- **Operazione:** pianificaAllenamento(data, orario, durata, luogo, tipologia, note);
- **Riferimento:** UC2-Gestisci Evento;
- **Pre-condizione:**
  1. L'allenatore ha selezionato l'opzione "aggiungi allenamento" presente sul calendario;
  2. Il sistema permette la compilazione di un form specifico al tipo di evento;
  3. L'evento non può essere pianificato nella stessa ora dello stesso giorno di altri eventi;
- **Post-condizione:**
  1. L'evento viene pianificato ed è visibile all'interno del calendario nella lista eventi

CONTRATTO 2 UC2: Gestisci Evento

- **Operazione:** pianificaAmichevole(data, orario, durata, luogo, squadraAvversaria);
- **Riferimento:** UC2-Gestisci Evento;
- **Pre-condizione:**
  4. L'allenatore ha selezionato l'opzione "aggiungi amichevole" presente sul calendario;
  5. Il sistema permette la compilazione di un form specifico al tipo di evento;

6. L’evento non può essere pianificato nella stessa ora dello stesso giorno di altri eventi;

- **Post-condizione:**
    1. L’evento viene pianificato ed è visibile all’interno del calendario nella lista eventi

CONTRATTO 3 UC2: Gestisci Evento

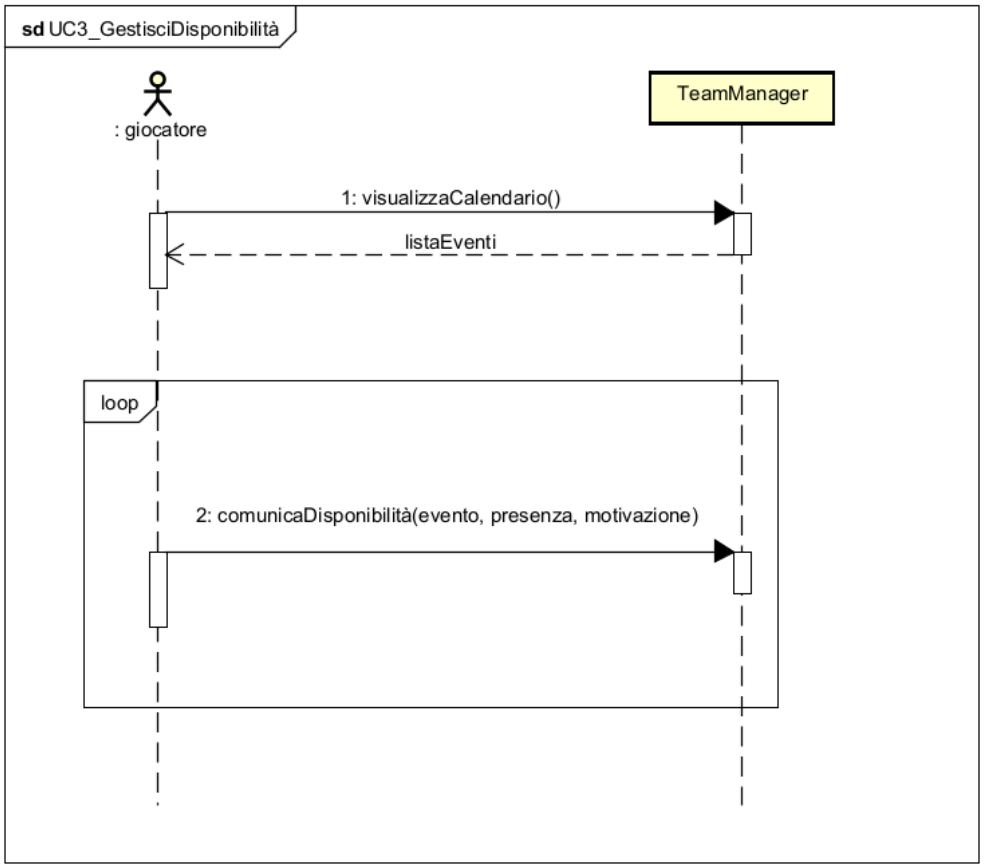
- **Operazione:** aggiornaEvento(eventoSelezionato, nuovaData, nuovoOrario, nuovaDurata, nuovoLuogo, campiSpecifici)
- **Riferimento:** UC2-Gestisci Evento
- **Pre-condizione:**
    1. L’evento risulta pianificato ed è presente all’interno della struttura dati contente i vari eventi;
    2. L’allenatore ha selezionato quel determinato evento;
- **Post-condizione:**
    1. I nuovi valori vengono salvati all’evento associato nella listaEventi di Calendario;
    2. Viene aggiornato il file relativo agli eventi;
    3. Viene restituita la listaEventi aggiornata di calendario;

CONTRATTO 5 UC2: Gestisci Evento

- **Operazione:** rimuoviEvento(EventoSelezionato)
- **Riferimento:** UC2-Gestisci Evento
- **Pre-condizione:**
    1. L’evento risulta pianificato ed è presente all’interno del calendario;
    2. L’allenatore ha selezionato quel determinato evento;
- **Post-condizione:**
    1. L’evento viene rimosso dalla lista degli eventi presenti nel calendario;
    2. Viene restituita la listaEventi aggiornata di calendario;

UC3 – Gestisci Disponibilità

Il seguente Diagramma di Sequenza di Sistema illustra le interazioni I/O dell'utente con il sistema TeamManager relative al Caso d'Uso 3:



Contratti dell'operazioni UC3

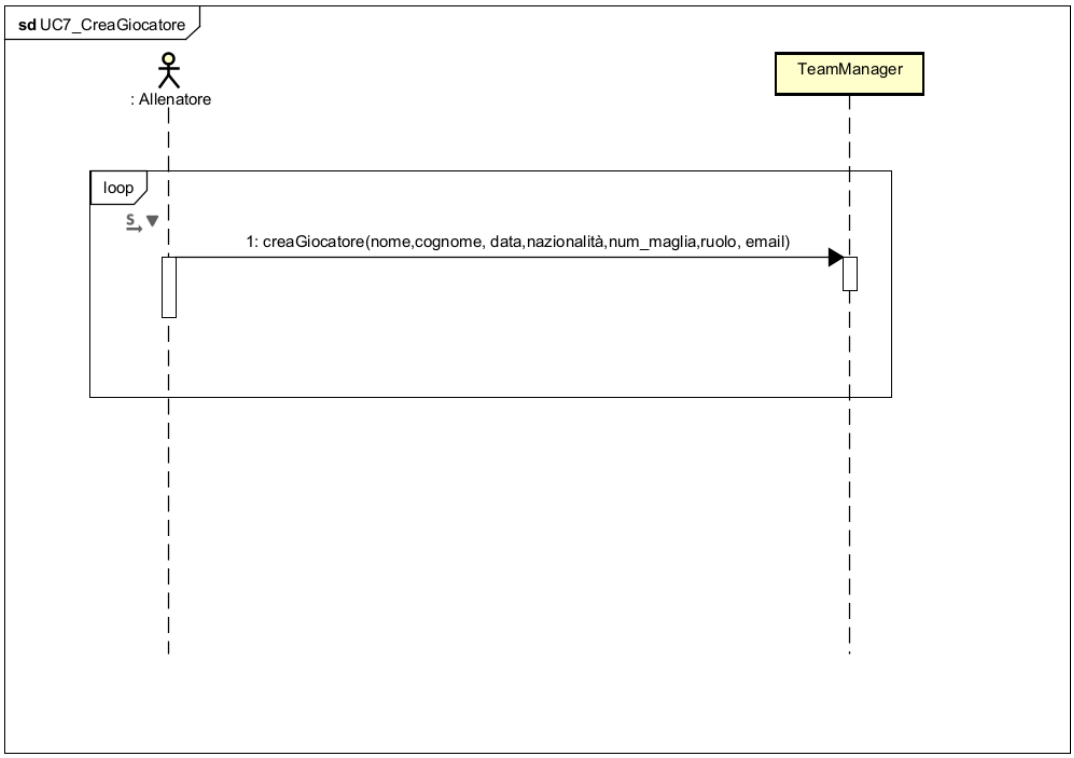
CONTRATTO 1 UC3: Gestisci Disponibilità

- **Operazione:** comunicaDisponibilità(evento, presenza, motivazione)
- **Riferimento:** UC3-Gestisci Disponibilità
- **Pre-condizione:**
  1. L'evento per cui si vuole comunicare la disponibilità è già stato pianificato.
  2. Il sistema ha caricato correttamente la lista degli eventi.
- **Post-condizione:**
  1. Lo stato di disponibilità del giocatore per quell'evento viene aggiornato (presente o assente).

2. Se la disponibilità è impostata su “assente”, viene salvata anche l’eventuale motivazione fornita.

UC7 – Crea Giocatore

Il seguente Diagramma di Sequenza di Sistema illustra le interazioni I/O dell’utente con il sistema TeamManager relative al Caso d’Uso 7:



Contratti delle Operazioni UC7

CONTRATTO 1 UC7: Crea Giocatore

- **Operazione:** creaGiocatore(Nome, Cognome, DataNascita, Nazionalità, NumMaglia, Ruolo, email)
- **Riferimento:** UC7-Crea Giocatore
- **Pre-condizione:**
  1. Tutti i campi richiesti sono compilati correttamente.
  2. Non esiste già un giocatore con lo stesso nome, cognome e data di nascita nell’archivio
- **Post-condizione:**
  1. Un nuovo Giocatore viene aggiunto all’archivio.
  2. I dati del giocatore sono ora disponibili per la selezione nella rosa.

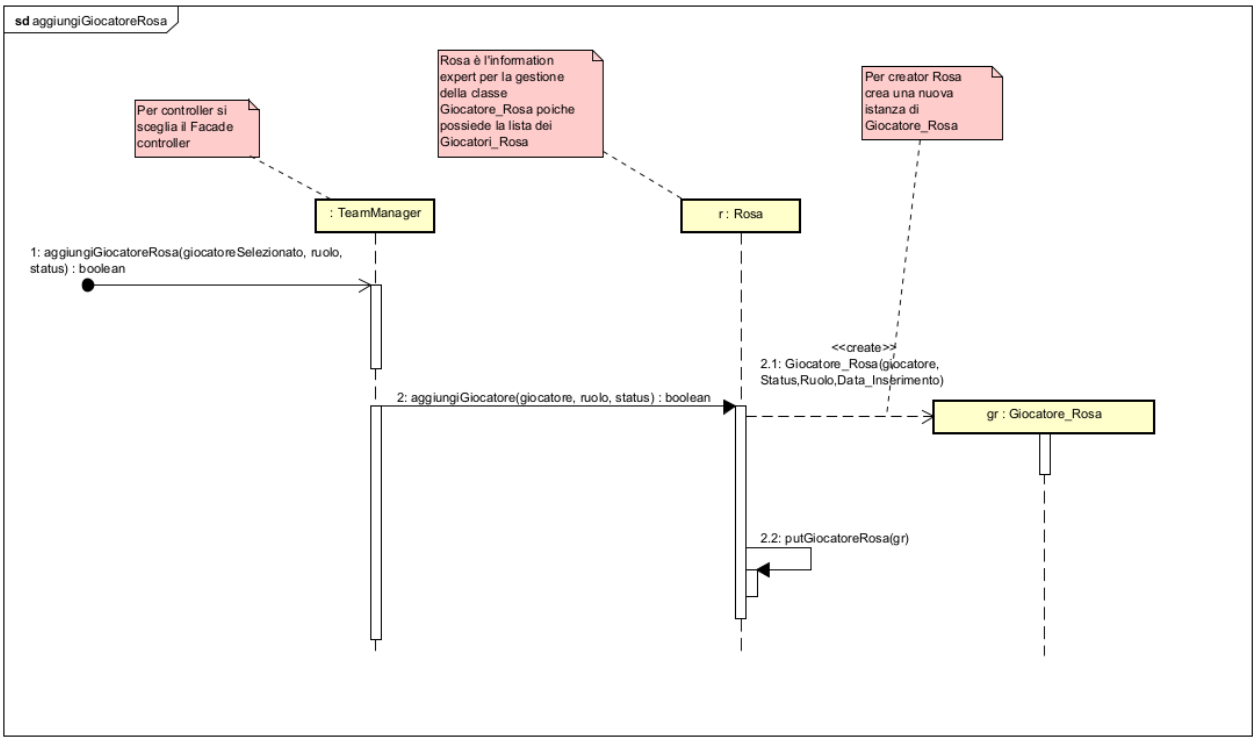
3.0 Fase di Progettazione

Gli elaborati principali della fase di progettazione comprendono i diagrammi di sequenza, utilizzati per descrivere il comportamento dinamico del sistema durante l'esecuzione dei casi d'uso selezionati per la prima iterazione. A supporto di questi, il diagramma delle classi offre una rappresentazione statica del sistema, evidenziando le principali entità coinvolte, i relativi attributi e metodi, nonché le associazioni tra di esse. Di seguito vengono riportati i diagrammi realizzati.

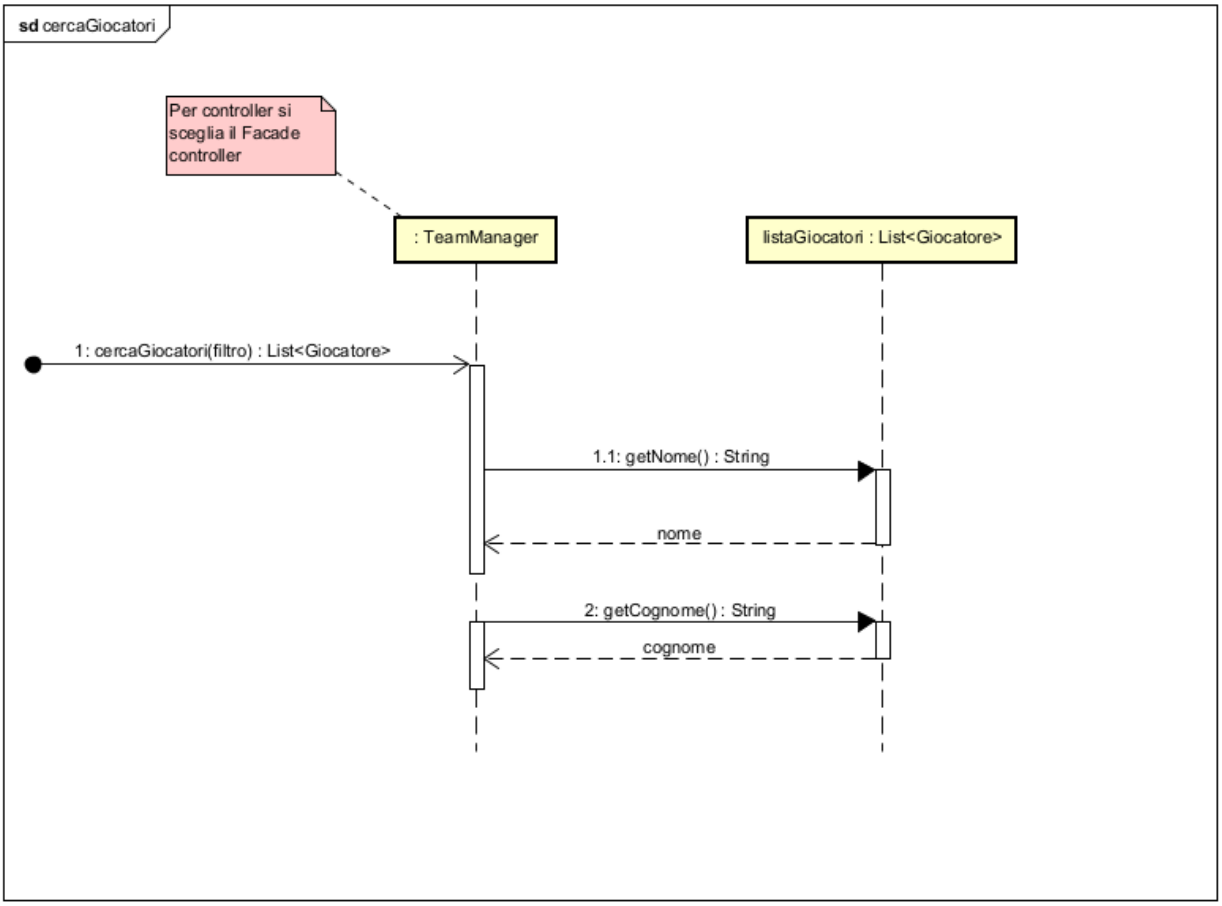
3.1 Diagrammi di Sequenza UC1

Nel sistema TeamManager, la classe **TeamManager** è stata progettata secondo il pattern **GoF Façade Controller**, fungendo da punto di accesso centrale alle funzionalità principali del sistema. In particolare, per il caso d'uso Gestisci Rosa, **TeamManager** coordina l'interazione tra l'interfaccia utente e le classi del dominio, delegando alla classe **Rosa** la gestione concreta dei giocatori in rosa. Seguendo il pattern Creator, è infatti **Rosa** a occuparsi della creazione e della memorizzazione delle istanze di **Giocatore_Rosa**, in quanto responsabile della collezione e del mantenimento dei giocatori attivi nella squadra.

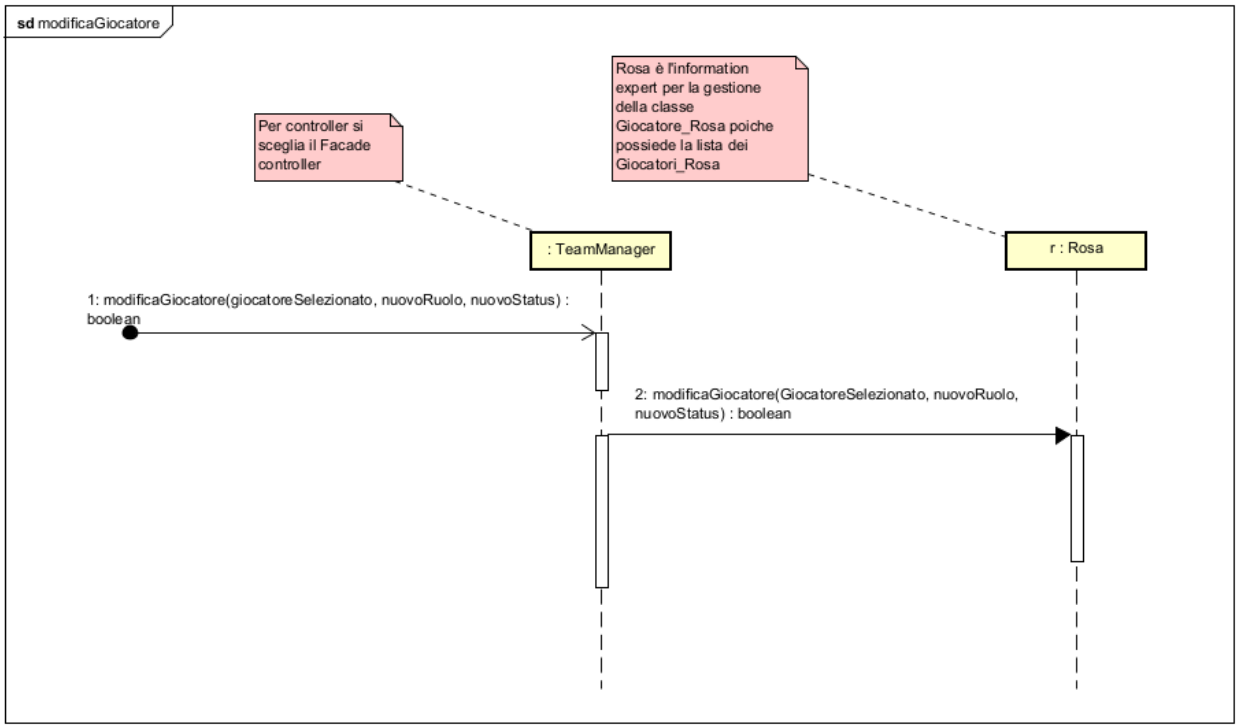
3.1.1 aggiungiGiocatoreRosa(giocatoreSelezionato, ruolo, status)



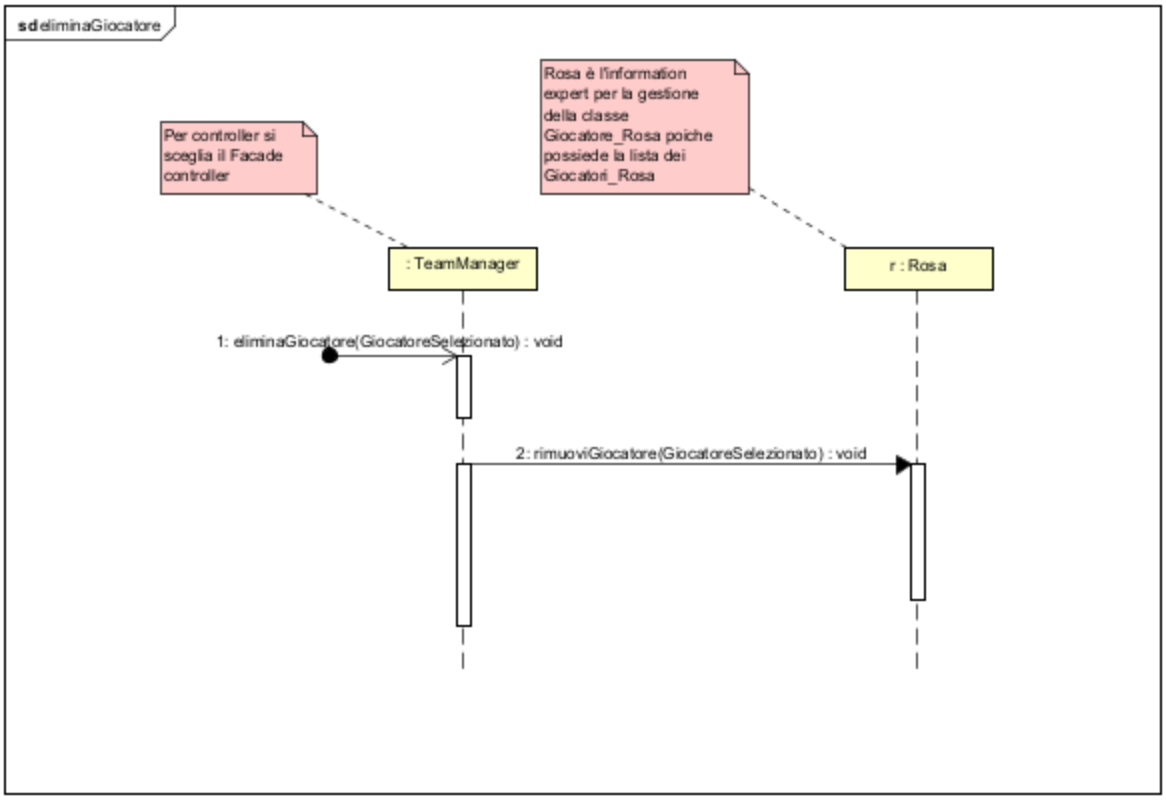
3.1.2 cercaGiocatori(filtro)



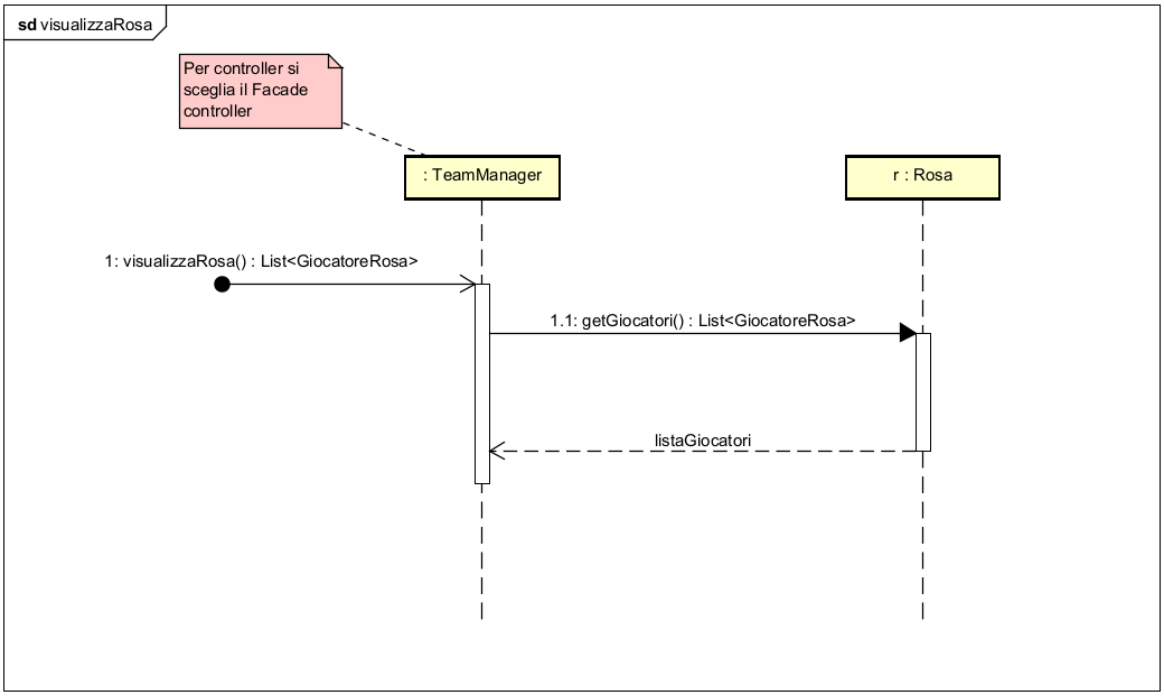
3.1.3 modificaGiocatore(giocatoreSelezionato, nuovoRuolo, nuovoStatus)



3.1.4 eliminaGiocatore(GiocatoreSelezionato)



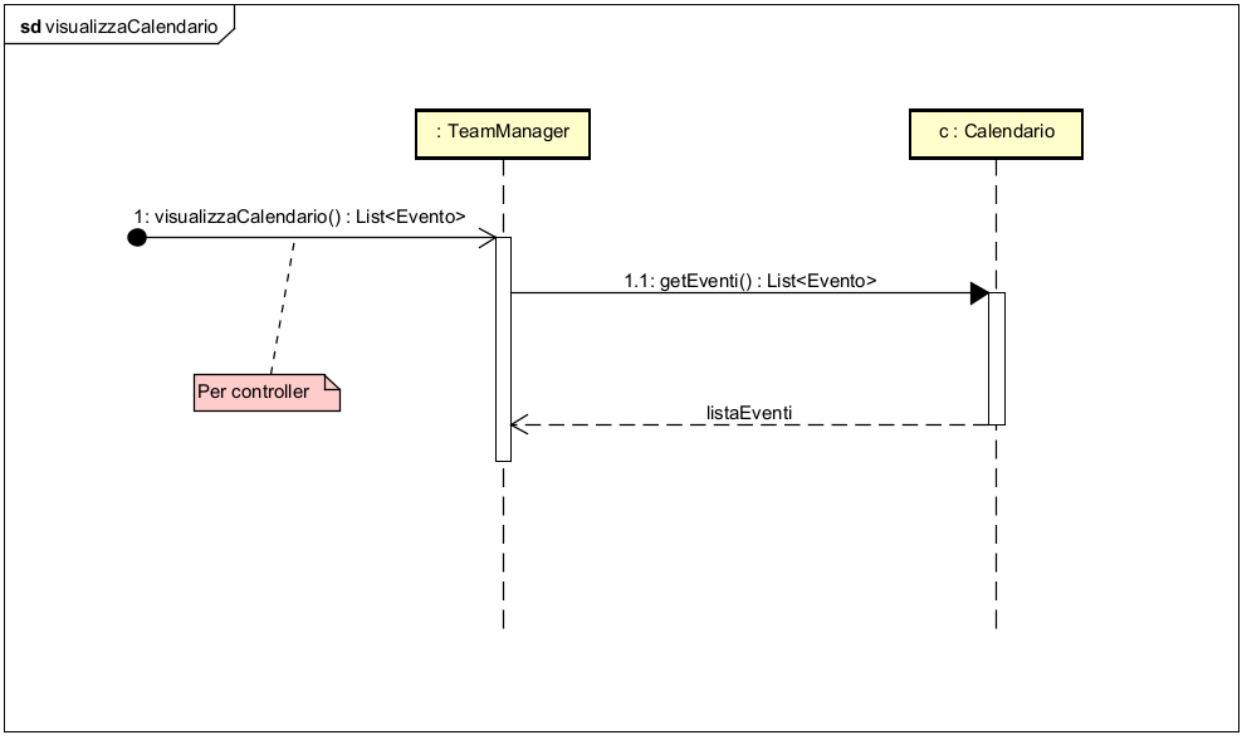
3.1.5 visualizzaRosa()



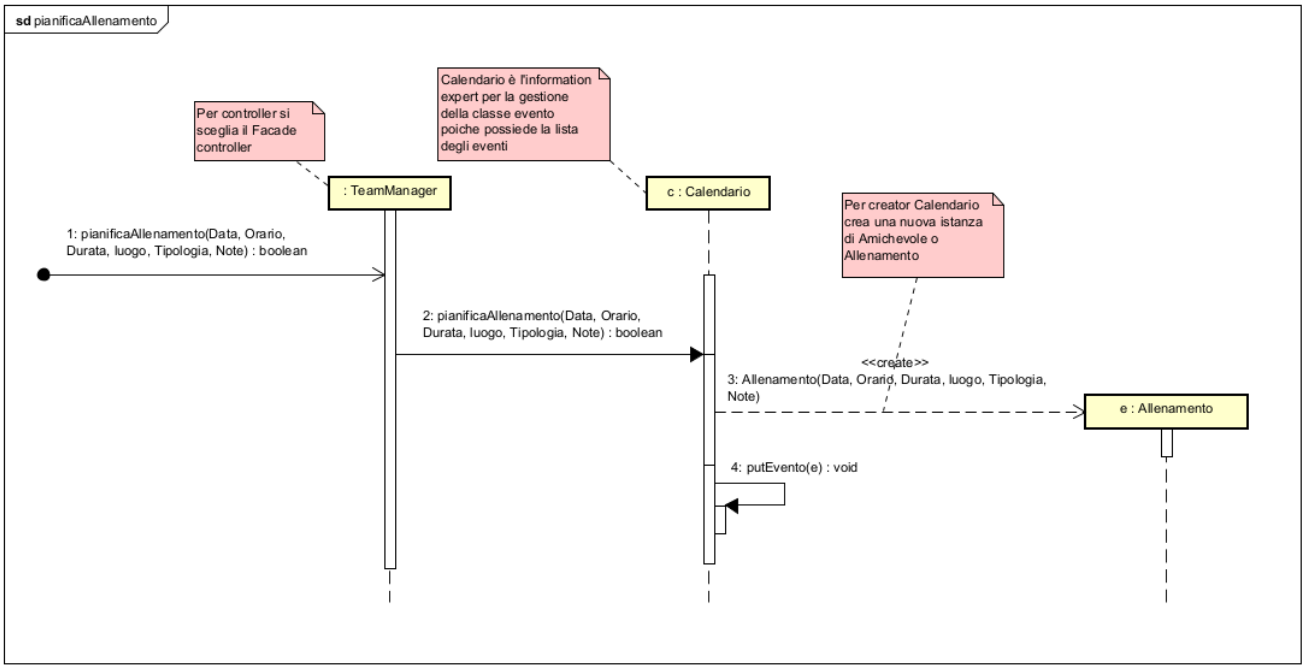
3.2 Diagramma di Sequenza UC2

Nel sistema TeamManager, la gestione degli eventi è affidata al caso d'uso UC2, che ha come obiettivo il tracciamento e la pianificazione delle attività della squadra, tra cui allenamenti e partite amichevoli. Il componente centrale di questa funzionalità è la classe Calendario, incaricata di mantenere aggiornata la lista degli eventi previsti. Tale classe consente la creazione (pattern GoF Controller), la modifica e la rimozione di istanze di eventi. Il controllo dell'interazione è affidato a TeamManager, che funge da Façade Controller e si occupa di orchestrare le operazioni tra l'interfaccia utente e il modello di dominio, delegando a Calendario l'effettiva gestione degli eventi.

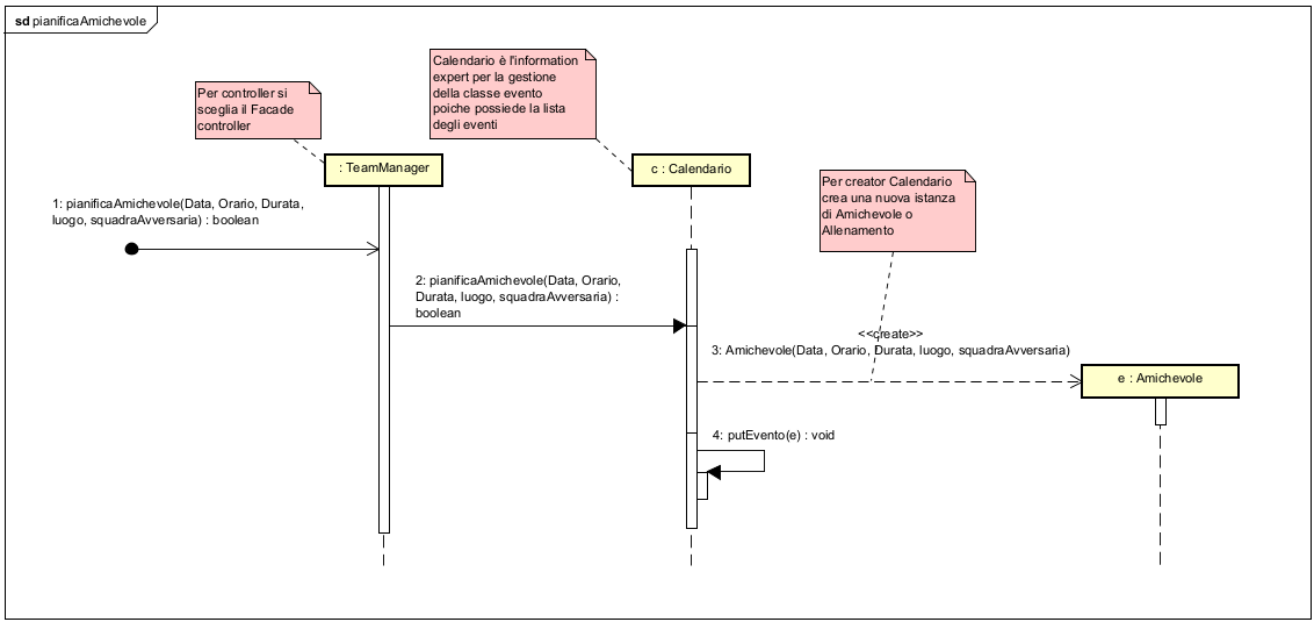
3.2.1 visualizzaCalendario()



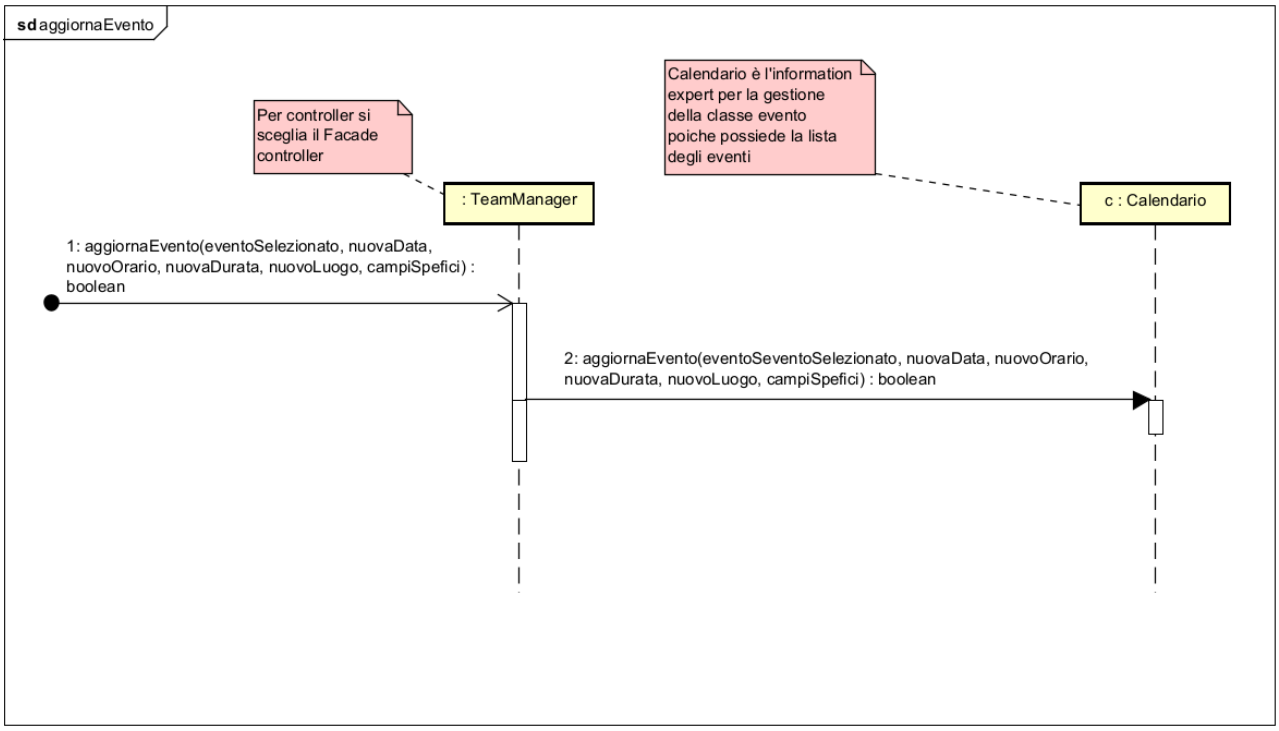
3.2.2 pianificaAllenamento(data, orario, durata, luogo, tipologia, note)



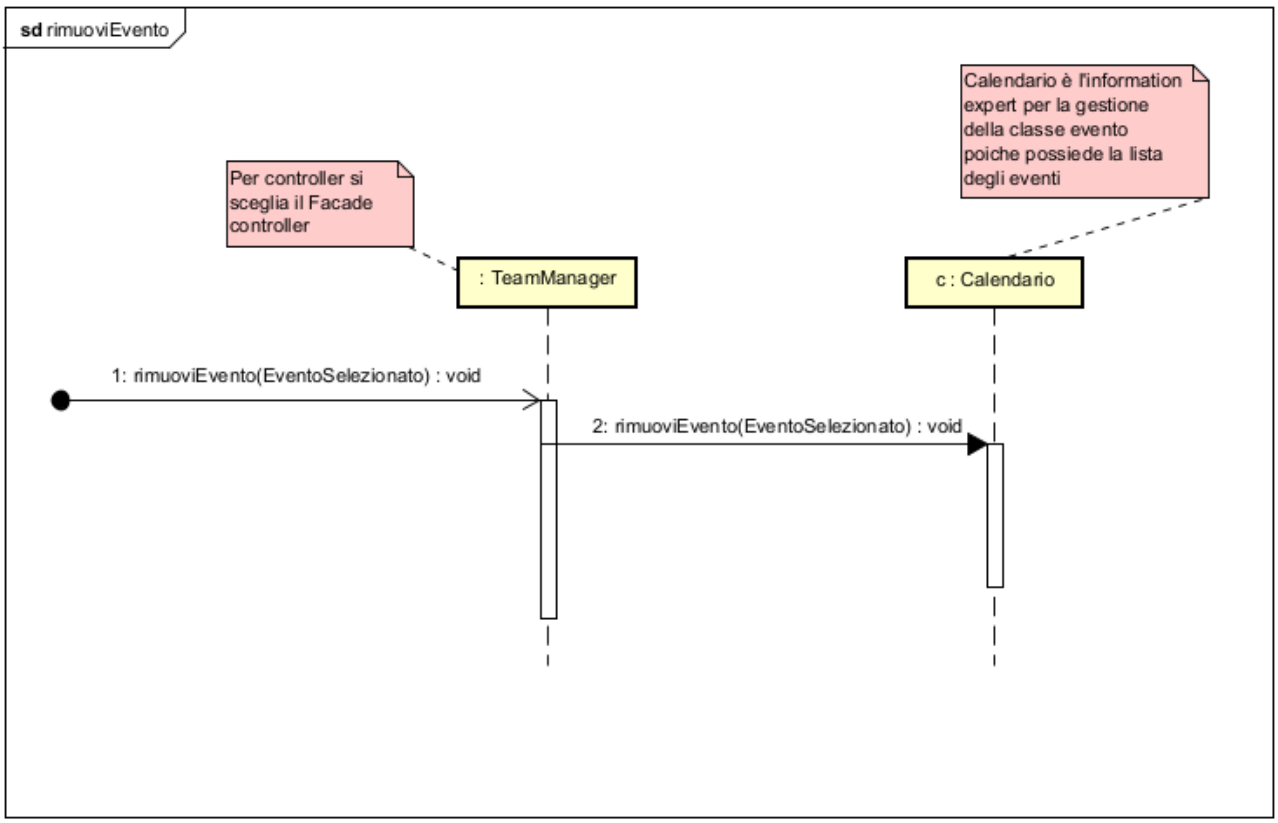
3.2.3 pianificaAmichevole(data, orario, durata, luogo, squadraAvversaria)



3.2.4 aggiornaEvento(eventoSelezionato, nuovaData, nuovoOrario, nuovaDurata, nuovoLuogo, campiSpefici)



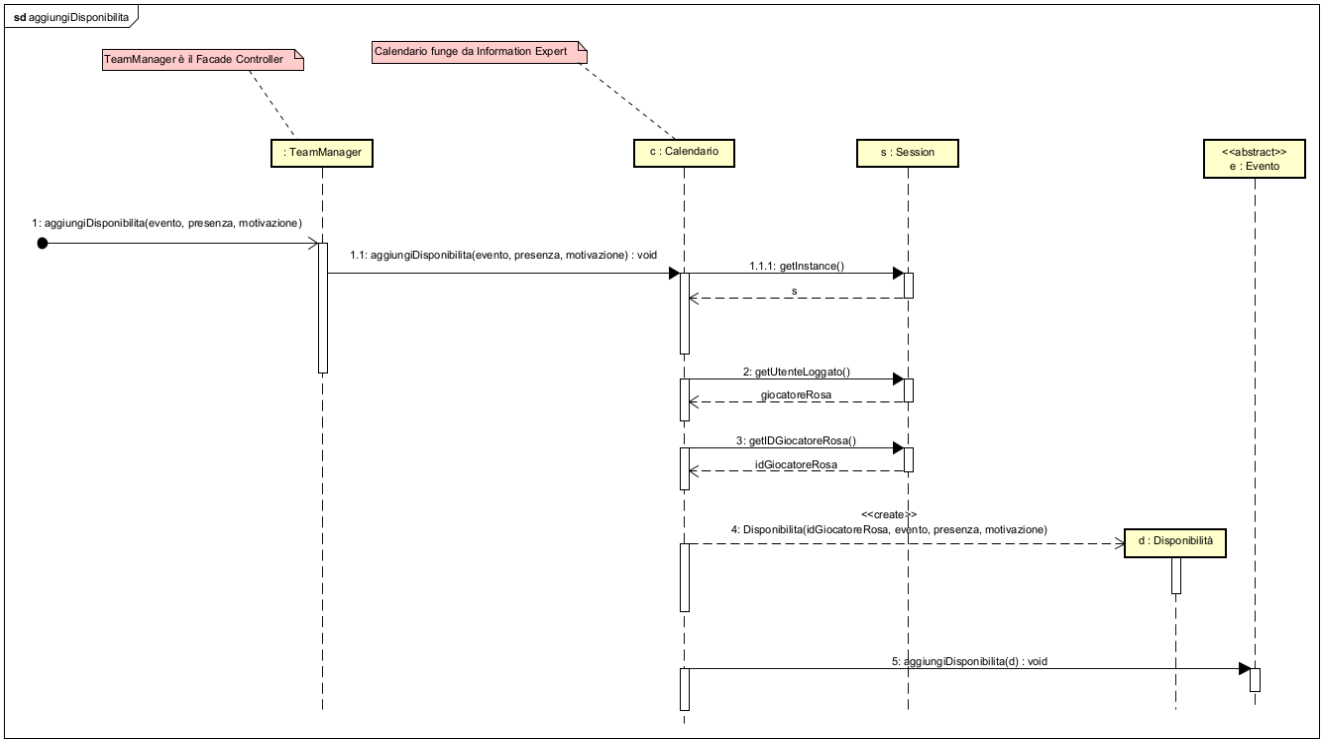
3.2.5 rimuoviEvento(eventoSelezionato)



3.3 Diagramma di Sequenza UC3

Il caso d'uso UC3 consente ai giocatori di interagire con il calendario degli eventi, dichiarando la propria disponibilità per ciascun evento programmato. L'utente può indicare la propria presenza o assenza, specificando, in caso di assenza, una motivazione testuale. Il controllo dell'interazione è gestito da TeamManager, mentre la responsabilità della creazione dell'oggetto Disponibilità è affidata alla classe Calendario, secondo il pattern Creator. Quest'ultima, ricevuti l'evento e il valore booleano di presenza, ricava l'identità del giocatore in base alla sessione attiva e associa tale giocatore all'evento, creando un'istanza di Disponibilità contenente: l'ID del giocatore, l'ID dell'evento, il valore di presenza e, se necessario, la motivazione. L'intera operazione è condizionata dall'esistenza di una sessione attiva che identifichi univocamente l'utente autenticato.

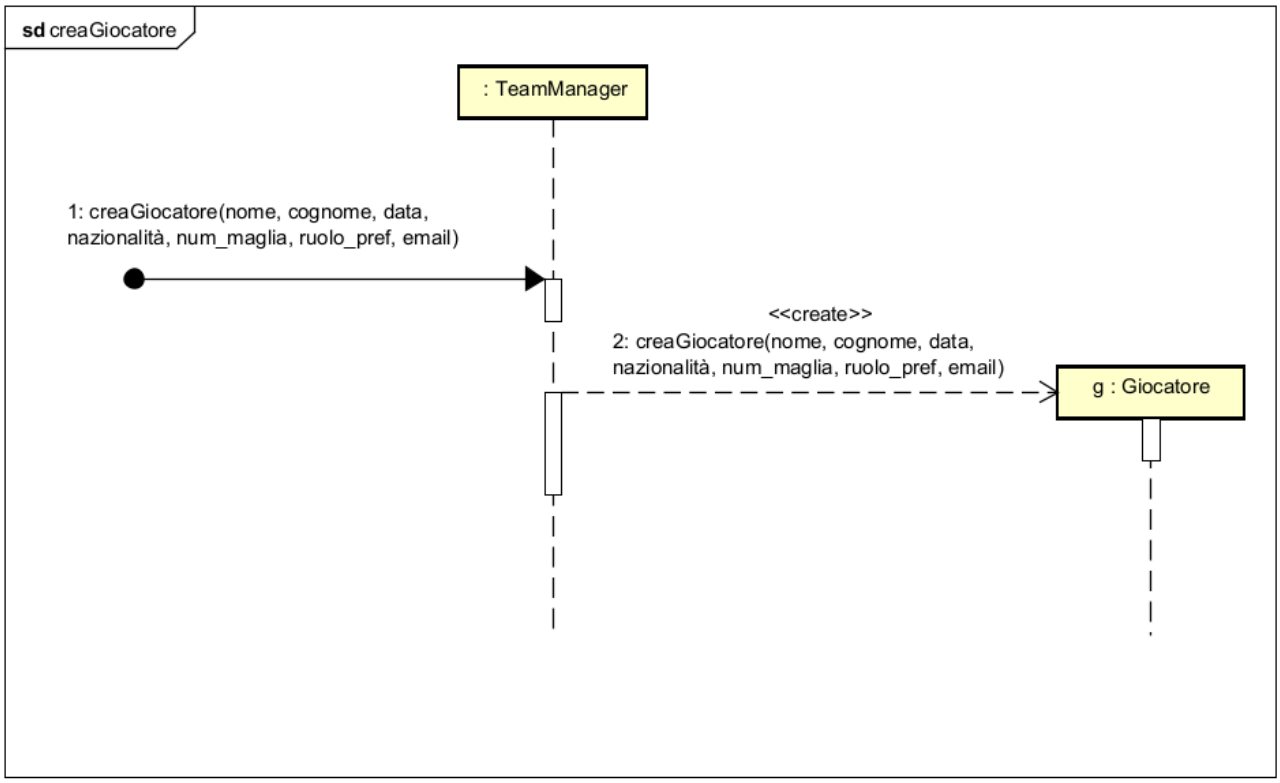
3.3.1 aggiungiDisponibilità(evento, presenza, motivazione)



3.4 Diagrammi di Sequenza UC7

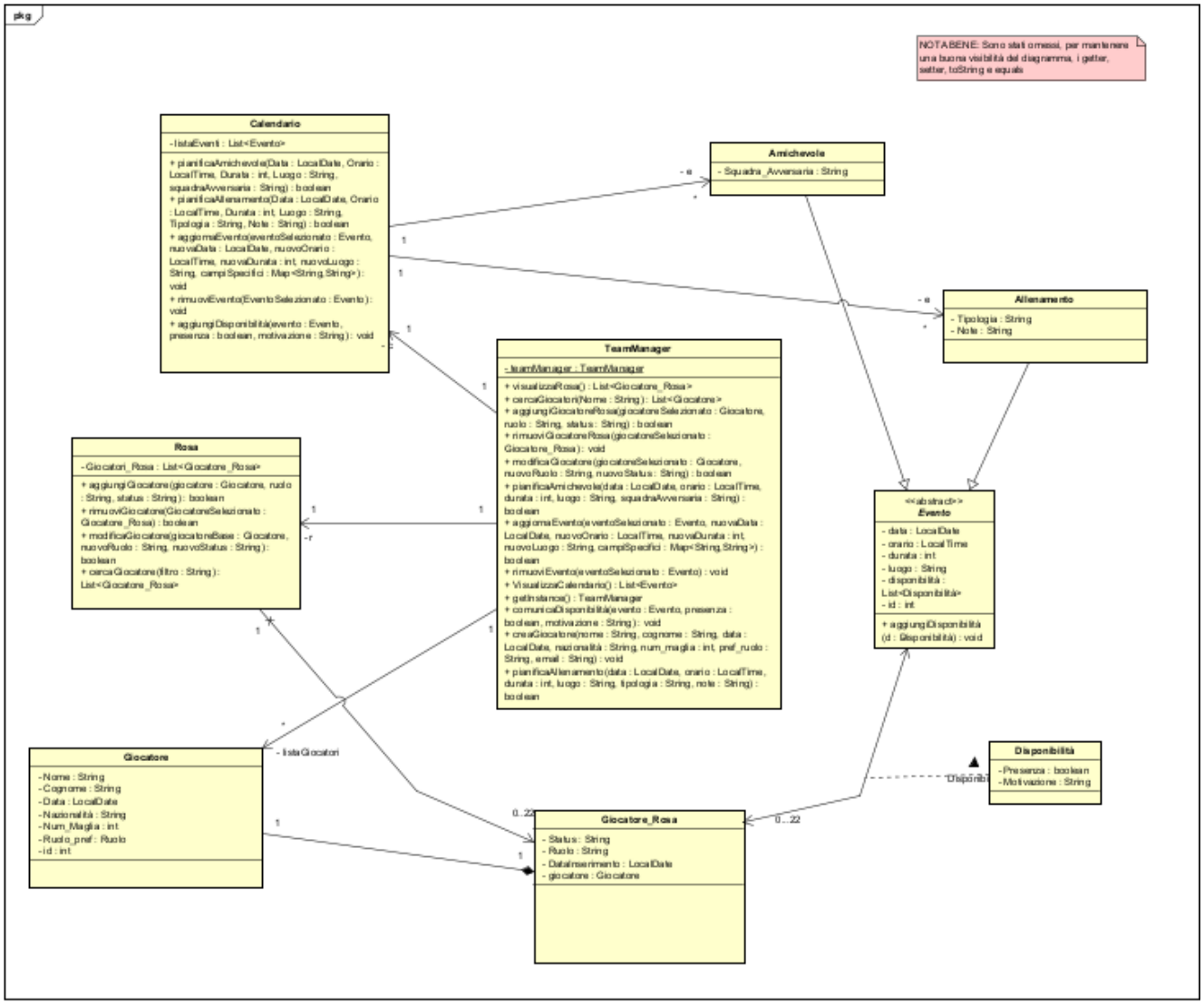
Il caso d'uso UC7 consente all'allenatore di inserire un nuovo giocatore all'interno dell'archivio di giocatori. Una volta creato, il giocatore diventa selezionabile nelle operazioni di gestione della rosa, rendendone possibile l'istanza come giocatore attivo della squadra, Giocatore_Rosa.

3.4.1 creaGiocatore(nome, cognome, data, nazionalità, num_maglia, ruolo_pref, email)



4.0 Diagramma delle Classi

Il diagramma delle classi rappresenta la struttura statica del sistema TeamManager, evidenziando le principali entità coinvolte nella prima iterazione e le relazioni tra esse. La classe TeamManager, progettata come Singleton secondo i pattern GoF, funge da Facade Controller, centralizzando il coordinamento delle operazioni tra l'interfaccia utente e il modello di dominio. Il pattern **Singleton** assicura che una classe abbia un'unica istanza fornendo un unico punto di accesso globale all'unica istanza durante l'esecuzione dell'applicazione. Questo è utile perché assicura coerenza dei dati tra le varie componenti del sistema (es. eventi, giocatori, statistiche); Evita problemi dovuti a duplicazione o inconsistenza nella gestione della rosa o del calendario e semplifica la gestione centralizzata dello stato dell'applicazione. Il pattern **Facade Controller** viene applicato nella progettazione per nascondere la complessità interna del sistema e fornire un'interfaccia semplificata all'interazione dell'utente; Promuove un accoppiamento debole fra cliente e sottosistema, il cliente può comunque, se necessario, usare direttamente le classi del sottosistema; Migliora la manutenibilità, separando in modo netto la logica di coordinamento dalla logica di dominio. In un primo momento si era pensato di utilizzare l'**Adapter** ma durante varie riflessioni si è capito che esso risolve un problema di compatibilità di interfaccia tra due entità mentre il **Facade** risolve un problema di complessità e accoppiamento all'interno di un sottosistema, offrendo un'interfaccia semplificata che più si adatta al nostro applicativo. Per semplicità di rappresentazione, nei diagrammi UML non sono riportati esplicitamente i metodi getter e setter, i quali saranno comunque implementati nel codice per garantire l'accesso controllato agli attributi delle classi.



5.0 Testing

5.1 Introduzione al Testing – White Box

Nella seguente iterazione, dopo aver progettato lo scheletro del codice, si è proceduto nella verifica della correttezza delle classi e dei metodi implementati attraverso la pratica del **Testing**; utilizzando un approccio con accesso ***"White-Box"***, è stato possibile testare direttamente il codice senza ausilio di un'interfaccia grafica, utilizzando il framework **JUnit4**. Tramite JUnit4, è stato possibile creare classi di test associate a quelle implementate, le quali hanno permesso di testare il codice utilizzando parametri "fittizi" e svolgere test di tipo **unitario**; i test di tipo unitario sono test relativi alla singola unità del programma, nel caso in esame le classi. Lo scopo del testing è stato quello di individuare "bug" all'interno del programma, ovvero comportamenti riscontrati che non soddisfano le aspettative

e, inoltre, verificare la robustezza e l'affidabilità del codice, cercando di evidenziare falle in punti critici di alcuni metodi.

5.2 Definizione delle classi e metodi da testare

Nella fase iniziale antecedente al testing in JUnit4, è stato oggetto di interesse definire le principali classi e i metodi da testare, omettendo nell'implementazione del testing metodi semplici come *getter* e *setter*; effettuata tale premessa, sono stati definiti le seguenti classi e metodi di test:

5.2.1 TeamManager

1. **testSingletonInstance()**
   - Si occupa di testare che l'istanza della classe TeamManager sia un *singleton*, ovvero, che tale istanza sia unica e non replicabile.
2. **testCreaGiocatore()**
   - Si occupa di testare che la creazione di un giocatore avvenga con esito positivo. In seguito ad un inserimento corretto, si è proceduto a testare che la lista dei giocatori non sia nulla e che uno stesso giocatore non possa essere inserito nuovamente.
3. **testEliminaGiocatore()**
   - Si occupa di testare la corretta eliminazione di un giocatore dalla lista dei giocatori.
4. **testCercaGiocatori()**
   - Si occupa di testare che una ricerca di un giocatore presente nella lista dei giocatori avvenga con successo, dato un filtro o parole chiave in ingresso al metodo.

5.2.2 Allenamento

1. **testAggiungiDisponibilità()**
   - Si occupa di testare se il metodo *aggiungiDisponibilità()* effettui correttamente l'inserimento delle disponibilità fornite da un giocatore; si verifica anche che la lista delle disponibilità non sia vuota.

5.2.3 Amichevole

1. **testAggiungiDisponibilità()**

- Si occupa di testare se il metodo *aggiungiDisponibilità()* effettui correttamente l'inserimento delle disponibilità fornite da un giocatore; si verifica anche che la lista delle disponibilità non sia vuota.

5.2.4 Calendario

1. **testPianificaAllenamento()**
   - Si occupa di testare la corretta pianificazione di una sessione di allenamento, verificando che l'allenamento inserito non sia nullo.
2. **testPianificaAmichevole()**
   - Si occupa di testare la corretta pianificazione di una partita amichevole, verificando che la lista degli amichevoli inseriti non sia vuota.
3. **testAggiornaEvento()**
   - Si occupa di testare il corretto aggiornamento di un evento, controllando se i vari campi siano stati effettivamente aggiornati.
4. **testRimuoviEvento()**
   - Si occupa di testare la corretta rimozione di un evento presente nella lista degli eventi.
5. **testGetEventi()**
   - Si occupa di testare che la lista restituita non sia nulla

6. **testAggiungiDisponibilità()**
   - Simulando una sessione di login da parte di un giocatore, si verifica la corretta aggiunta della sua disponibilità relativa ad un evento (si ricorda che un evento può essere un amichevole o un allenamento). Infine, si verifica che la disponibilità aggiunta ad un evento non sia nulla

5.2.5 Disponibilità

1. **testMetodoCostruttore()**
   - Si occupa di testare la corretta inizializzazione di un'istanza di **Disponibilità**, a seconda dell'adesione fornita ad un determinato evento (*true* = presenza -> nessuna motivazione; *false* = assenza -> motivazione richiesta).

5.2.6 Rosa

1. **testAggiungiGiocatore()**
   - Si occupa di testare la corretta aggiunta di un giocatore in rosa. Effettuata l'aggiunta di un nuovo giocatore, si è verificato che il sistema impedisca con successo l'aggiunta di uno stesso giocatore aggiunto precedentemente.
2. **testRimuoviGiocatore()**
   - Si occupa di testare la corretta rimozione di un giocatore presente nella rosa.
3. **testModificaGiocatore()**
   - Verifica l'esito positivo di un'operazione di modifica dei parametri di un giocatore presente nella rosa. A seguito della verifica dell'esito, si è testato che i parametri del giocatore siano stati effettivamente modificati con parametri aggiornati.
4. **testCercaGiocatori()**
   - Si occupa di testare l'operazione di ricerca di giocatori presenti in rosa tramite parole chiave, o filtro, fornite in ingresso al metodo. Il test ha permesso di verificare che la ricerca restituisce i giocatori che, nel loro nome o cognome, contengono il valore del filtro.
5. **testGetGiocatori()**
   - Si occupa di testare che la lista di giocatori fornita non sia nulla.