
UNIVERSITÀ
DEGLI STUDI
FIRENZE

UNIVERSITÀ DEGLI STUDI DI FIRENZE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

# Architettura ed Implementazione di un sistema per il gioco del Fantacalcio

*Autori:*
Barbieri Andrea
Cappini Niccolò
Zanni Niccolò

*N° Matricole:*
7078620
7077249
7127236

*Corso principale:*
Ingegneria del Software

*Docente corso:*
Enrico Vicario

# Indice



# Elenco delle figure

# 1 Introduzione generale

Se volete yappare per allungarla fate pure

## 1.1 Statement

Il sistema progettato vuole ricreare una applicazione desktop per il gioco del Fantacalcio. I partecipanti al gioco sono in grado di creare, entrare e gestire le leghe e i propri fanta team: in particolare, sono in grado di schierare una formazione per la successiva partita da giocare. Il sistema inoltre permette a varie testate giornalistiche di registrarsi e, una volta selezionate dall'admin della lega, di dare i voti ai calciatori.

### 1.1.1 Actors

Il sistema prevede tre tipi di utenti distinti, ciascuno con capacità diverse:

- **FantaUser**: l'utente base dell'applicazione è in grado di entrare in leghe già esistenti per competere con gli altri giocatori schierando la formazione migliore in ogni giornata
- **Admin**: è un utente base che è anche admin di una lega. Si occupa di generare il calendario, assegnare o rimuovere i calciatori dai team e di calcolare i risultati delle partite alla fine della giornata
- **Newspaper**: è la testata giornalistica che si occupa di assegnare i voti ai calciatori ad ogni giornata del campionato

## 1.2 Architettura

Aggiungere Figura, probabilmente quella generale non specifica va creata

Il sistema è stato sviluppato in moduli ciascuno dei quali si occupa di compiti specifici. Ciò permette di separare le responsabilità rendendo il codice più organizzato e semplice da manutenere e da estendere.

- **Business Logic**: contiene le classi che si occupano della logica di business. Tra queste ci sono i services che gestiscono le operazioni che i vari attori sono i grado di compiere e le classi che si occupano di gestire le transazioni con **JPA**
- **Domain Model**: contiene le entità annotate dell'applicazione
- **Repositories**: contiene le implementazione concreta dei repositories che si interfacciano con il database

Andre correggimi se sbaglio su Jpa. Inoltre le interfacce dei repository vanno bene qui o è meglio spostarle?

### 1.2.1 Dipendenze

L'applicazione utilizza delle dipendenze esterne per la persistenza dei dati, per l'implementazione dei test e per la realizzazione dell'interfaccia grafica.

- **JPA**: Java Persistence API, specifica standard che consente di gestire in maniera astratta la persistenza dei dati su database relazionali, semplificando l'interazione con le entità tramite annotazioni e query ad alto livello.
- **JUnit**: framework di testing unitario per Java che permette di automatizzare i test delle singole componenti, verificandone il corretto funzionamento e supportando l'integrazione nei processi di build.
- **Mockito**: libreria di supporto ai test che consente di creare oggetti fittizi (mock) per simulare le dipendenze esterne e isolare le unità da testare, favorendo un approccio di testing modulare e controllato.
- **H2 Database**: database relazionale in-memory leggero e veloce, utilizzato durante lo sviluppo e il testing per evitare la dipendenza da un database esterno, garantendo facilità di configurazione e rapidità di esecuzione.
- **Java Swing**: libreria grafica inclusa nel JDK per la realizzazione dell'interfaccia utente, che fornisce componenti GUI (bottoni, menu, finestre, etc...) per costruire applicazioni desktop interattive.

aggiungere dipendenze se mancano

## 2 Strumenti utilizzati

Il codice è stato scritto in Java utilizzando **IntelliJ Idea** ed **Eclipse**. La **GUI** è stata scritta utilizzando **Java Swing** e **Window Builder** Per la costruzione degli **UML** è stato utilizzato **StarUML** mentre per il versionamento del codice è stato utilizzato **Github**. Inoltre sono stati usati vari **LLMS** come aiuto nella scrittura del codice, in particolare **Copilot**, **ChatGpt** e **Gemini**.

Andre aggiungi la descrizione di cosa può fare

# 3 Progettazione

Aggiungere UML, UseCases diagram e template, MockUps, Navigation Diagram ed ER? meglio dire come gestiamo il database in memoria e jpa/hibernate

Andre scrivi il paragrafo 3 sul database ovvero parla di hibernate,transaction manager e come hai annotato le clsse e del database in memoria

Se vuoi scrivi della gui in generale

Nicco crea gli use case template scegline qualcuno che ritieni significativo NON login e register

## 3.1 Use Case Templates

Tabella 1: Crea fantalega

| | |
|---|---|
| **Id** | UC-01 (Crea fantalega) |
| **Scope** | user goal |
| **Descrizione** | L'utente vuole creare una nuova fantalega di cui sarà admin. |
| **Attori** | Admin |
| **Flusso base** | 1. L'utente inserisce il nome e il codice della lega che vuole creare.<br>2. Il sistema verifica che non sia già presente una lega con lo stesso codice.<br>3. Se i dati sono validi, il sistema crea la nuova lega. |
| **Flusso alternativo** | 2.1 Il codice scelto è già utilizzato per un'altra lega.<br>2.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema. |
| **Test** | IT3 |

Tabella 2: Assegna giocatori alle rose

| | |
|---|---|
| **Id** | UC-02 (Assegna giocatori alle rose) |
| **Scope** | user goal |
| **Descrizione** | L'admin vuole assegnare i giocatori corretti alle rose presenti nella lega. |
| **Attori** | Admin |
| **Flusso base** | 1. L'admin inserisce il team e il giocatore che deve essere assegnato a tale team.<br>2. Il sistema verifica che il numero massimo di giocatori nel team non sia già stato raggiunto.<br>3. Il sistema verifica che il numero massimo di giocatori nel team appartenenti allo stesso ruolo del nuovo giocatore non sia già stato raggiunto.<br>4. Il sistema salva il nuovo contratto tra team e giocatore. |
| **Flusso alternativo** | 2.1 Il numero massimo di giocatori nel team è già stato raggiunto.<br>2.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>3.1 Il numero massimo di giocatori nel team appartenenti allo stesso ruolo del nuovo giocatore è già stato raggiunto.<br>3.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema. |
| **Test** | IT5 |

Tabella 3: Inserisci formazione

| | |
|---|---|
| **Id** | UC-03 (Inserisci formazione) |
| **Scope** | user goal |
| **Descrizione** | L'utente vuole inserire la formazione per giocare la partita successiva. |
| **Attori** | FantaUser |
| **Flusso base** | 1. L'utente fornisce la LineUp che vuole utilizzare per la partita successiva.<br>2. Il sistema verifica che la data in cui viene effettuata l'operazione sia precedente alla data della partita.<br>3. Il sistema verifica che la data in cui viene effettuata l'operazione non sia un sabato o una domenica.<br>4. Se la partita non è la prima del campionato, il sistema verifica che i voti per la partita precedente siano già stati calcolati.<br>5. Il sistema verifica che i giocatori appartenenti alla formazione siano giocatori posseduti dall'utente.<br>6. Se l'utente ha già stata inserito una formazione per la partita, il sistema la elimina per poter inserire la nuova formazione.<br>7. Il sistema salva la formazione per la partita successiva. |
| **Flusso alternativo** | 2.1 La data in cui viene effettuata l'operazione è successiva alla data della partita.<br>2.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>3.1 La data in cui viene effettuata l'operazione è un sabato o una domenica.<br>3.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>4.1 I voti per la partita precedente non sono stati ancora calcolati.<br>4.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>5.1 C'è almeno un giocatore all'interno della formazione che non appartiene all'utente.<br>5.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema. |
| **Test** | IT9 |

Tabella 4: Scambia giocatori - Invia proposta

| | |
|---|---|
| **Id** | UC-04 (Scambia giocatori - Invia proposta) |
| **Scope** | user goal |
| **Descrizione** | L'utente vuole inviare una proposta di scambio di giocatori ad un altro utente. |
| **Attori** | FantaUser |
| **Flusso base** | 1. L'utente fornisce i giocatori coinvolti nello scambio e le relative rose.<br>2. Il sistema verifica che i due giocatori abbiano lo stesso ruolo.<br>3. Il sistema verifica che i giocatori coinvolti appartengano alle rose fornite dall'utente.<br>4. Viene creata la nuova proposta e si verifica se è già presente una proposta con le stesse caratteristiche.<br>5. Il sistema salva la nuova proposta. |
| **Flusso alternativo** | 2.1 I due giocatori coinvolti nello scambio non hanno lo stesso ruolo.<br>2.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>3.1 I giocatori coinvolti nello scambio non appartengono alle rose fornite dall'utente<br>3.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>4.1 Nel sistema è già presente una proposta con le stesse caratteristiche.<br>4.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema. |
| **Test** | IT12 |

Tabella 5: Scambia giocatori - Accetta proposta

| | |
|---|---|
| **Id** | UC-05 (Scambia giocatori - Accetta proposta) |
| **Scope** | user goal |
| **Descrizione** | L'utente vuole accettare una proposta di scambio di giocatori inviata da un altro utente. |
| **Attori** | FantaUser |
| **Flusso base** | 1. L'utente fornisce la proposta che vuole accettare e la propria rosa.<br>2. Il sistema verifica che la rosa fornita sia coinvolta nella proposta.<br>3. Il sistema verifica che entrambi i contratti dei giocatori coinvolti siano ancora presenti.<br>4. La proposta è accettata, quindi il sistema provvede a modificare i contratti dei giocatori per cambiare la loro squadra di appartenenza.<br>5. Il sistema salva i nuovi contratti. |
| **Flusso alternativo** | 2.1 La rosa fornita non è coinvolta nella proposta.<br>2.2 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema.<br>3.1 Il contratto di almeno uno dei giocatori coinvolti non è più presente.<br>3.2 Il sistema provvede a rifiutare la proposta dato che non è valida.<br>3.3 Il sistema mostra a schermo un messaggio di errore, specificando la causa del problema. |
| **Test** | IT13 |

# 4 Implementazione

Parlare del domain model, delle annotazioni, dei repository, dei service forse è più adatto qui parlare approfonditamente del databse ed in progettazione fare un introduzione

Qui parla di come hai implementato il transaction manager, E Gui approfondita

# 5 ApplicationTests

## 5.1 Unit Tests

I seguenti test verificano il corretto funzionamento delle singole unità di codice.

### 5.1.1 Test UT1: Registrazione di un FantaUser

Va tenuto qui?

Questo unit test verifica il corretto funzionamento del sistema di registrazione di un FantaUser, tramite mock dei repository, implementando il template UC-01.

```
@Test
void testRegisterFantaUser_SavesWhenNotExists() {
    when(fantaUserRepository.getUser("mail", "pswd")).thenReturn(Optional
        .empty());

    service.registerFantaUser("mail", "pswd");

    verify(fantaUserRepository).saveFantaUser(argThat(user ->
            user.getEmail().equals("mail") && user.getPassword().equals("
                pswd")));
}
```

Similmente sono stati anche prodotti dei test per la registrazione di un NewsPaper.

### 5.1.2 Test UT XXX: Login di un FantaUser

Va tenuto qui?

Questo unit test verifica il corretto funzionamento del sistema di registrazione di un FantaUser, tramite mock dei repository.

```
@Test
void testLoginFantaUser_ReturnsTrueWhenPresent() {
    when(fantaUserRepository.getUser("mail", "pswd"))
            .thenReturn(Optional.of(new FantaUser("mail", "pswd")));

    assertThat(service.loginFantaUser("mail", "pswd")).isTrue();
}
```

Similmente sono stati anche prodotti dei test per il login di un NewsPaper.

### 5.1.3 Test UT XXX: Crea fantalega

Questo unit test verifica il corretto funzionamento del sistema di creazione di una nuova League, tramite mock dei repository.

```
@Test
void testCreateLeague() {
    FantaUser admin = new FantaUser("admin@test.com", "pwd");
    NewsPaper np = new NewsPaper("Gazzetta");
    String leagueCode = "L001";

    // League code does not exist yet
    when(leagueRepository.getLeagueByCode(leagueCode))
        .thenReturn(Optional.empty());

    adminUserService.createLeague("My League", admin, np, leagueCode);

    // Verify that saveLeague was called
    verify(leagueRepository, times(1)).saveLeague(any(League.class));
}
```

### 5.1.4 Test UT XXX: Genera calendario

Questo unit test verifica il corretto funzionamento del sistema di generazione del calendario della League, tramite mock dei repository.

```
@Test
void testGenerateCalendar_SavesMatches() {
    FantaUser admin = new FantaUser(null, null);
    League league = new League(admin, "Serie A", null, null);

    // Create 4 real teams (even number for round-robin)
    FantaTeam team1 = new FantaTeam("Team1", null, 0, null, new HashSet
        <>());
    FantaTeam team2 = new FantaTeam("Team2", null, 0, null, new HashSet
        <>());
    FantaTeam team3 = new FantaTeam("Team3", null, 0, null, new HashSet
        <>());
    FantaTeam team4 = new FantaTeam("Team4", null, 0, null, new HashSet
        <>());
    List<FantaTeam> teams = List.of(team1, team2, team3, team4);

    // 38 match days (real objects are okay)
    List<MatchDaySerieA> matchDays = new ArrayList<>();
    for (int i = 0; i < 38; i++)
        matchDays.add(new MatchDaySerieA("match", LocalDate.now()));

    // Mock repositories
    when(fantaTeamRepository.getAllTeams(league)).thenReturn(teams);
    when(matchDayRepository.getAllMatchDays()).thenReturn(matchDays);

    adminUserService.generateCalendar(league);

    verify(matchRepository, atLeastOnce()).saveMatch(any(Match.class));
}
```

### 5.1.5 Test UT XXX: Assegna giocatori alle rose

Questo unit test verifica il corretto funzionamento del sistema di assegnazione dei Player ai FantaTeam che li hanno acquistati, tramite mock dei repository.

```
@Test
void testSetPlayerToTeam_SavesContract_WhenBelowLimits() {
    FantaTeam team = new FantaTeam("Team", null, 0, null, new HashSet<>()
        );
    Player.Goalkeeper gk = new Player.Goalkeeper("Gigi", "Buffon", Player
        .Club.JUVENTUS);

    adminUserService.setPlayerToTeam(team, gk);

    verify(contractRepository).saveContract(argThat(c -> c.getTeam().
        equals(team) && c.getPlayer().equals(gk)));
}
```

### 5.1.6 Test UT XXX: Calcola voti della giornata

Questo unit test verifica il corretto funzionamento del sistema di calcolo dei voti della giornata una volta terminata, tramite mock dei repository.

```
@Test
void testCalculateGrades_SavesResultsAndUpdatesPoints() {

    FantaUser admin = new FantaUser("admin@example.com", "pwd");
    NewsPaper newspaper = new NewsPaper("Gazzetta");
    League league = new League(admin, "Serie A", newspaper, "1234");

    LocalDate matchDate = LocalDate.of(2025, 9, 21); // Sunday
    MatchDaySerieA prevDay = new MatchDaySerieA("Day0", matchDate.
        minusWeeks(1));
    MatchDaySerieA dayToCalc = new MatchDaySerieA("Day1", matchDate);

    when(matchDayRepository.getPreviousMatchDay(any()))
```

```
13          .thenReturn(Optional.of(prevDay));
14
15      AdminUserService serviceWithFixedDate = new AdminUserService(
            transactionManager) {
16          @Override
17          protected LocalDate today() {
18              return matchDate.plusDays(5);
19          }
20
21          @Override
22          protected Optional<MatchDaySerieA> getNextMatchDayToCalculate(
                LocalDate d, TransactionContext c, League l,
23                  FantaUser u) {
24              return Optional.of(dayToCalc);
25          }
26      };
27
28      // Teams
29      FantaTeam team1 = new FantaTeam("Team1", league, 0, admin, Set.of());
30      FantaTeam team2 = new FantaTeam("Team2", league, 0, admin, Set.of());
31
32      // Match
33      Match match = new Match(dayToCalc, team1, team2);
34      when(matchRepository.getAllMatchesByMatchDay(dayToCalc, league))
35          .thenReturn(List.of(new Match(dayToCalc, team1, team2)));
36
37      // Players
38      Player.Goalkeeper gk1 = new Player.Goalkeeper("G1", "Alpha", Player.
            Club.ATALANTA);
39      Player.Goalkeeper gk2 = new Player.Goalkeeper("G2", "Beta", Player.
            Club.BOLOGNA);
40
41      LineUp lineup1 = new _433LineUp._443LineUpBuilder(match, team1).
            withGoalkeeper(gk1).build();
42      LineUp lineup2 = new _433LineUp._443LineUpBuilder(match, team2).
            withGoalkeeper(gk2).build();
43
44      when(lineUpRepository.getLineUpByMatchAndTeam(match, team1)).
            thenReturn(Optional.of(lineup1));
45      when(lineUpRepository.getLineUpByMatchAndTeam(match, team2)).
            thenReturn(Optional.of(lineup2));
46
47      // Grades
48      Grade grade1 = new Grade(gk1, dayToCalc, 70.0, newspaper);
49      Grade grade2 = new Grade(gk2, dayToCalc, 60.0, newspaper);
50      when(gradeRepository.getAllMatchGrades(match, newspaper)).thenReturn(
            List.of(grade1, grade2));
51
52      // Act
53      serviceWithFixedDate.calculateGrades(admin, league);
54
55      // Assert: Result persisted
56      verify(resultRepository).saveResult(any());
57
58      // Assert: team points updated
59      assertThat(team1.getPoints()).isEqualTo(3);
60      assertThat(team2.getPoints()).isEqualTo(0);
61  }
```

### 5.1.7 Test UT XXX: Unisciti alla lega

Questo unit test verifica il corretto funzionamento del sistema di ingresso in una League, tramite mock dei repository.

```
1  @Test
2  void testJoinLeague() {
```

```
    FantaUser user = new FantaUser("user@test.com", "pwd");
    League league = new League(user, "Test League", new NewsPaper("
        Gazzetta"), "L002");
    FantaTeam team = new FantaTeam("Team A", league, 0, user, Set.of());

    when(leagueRepository.getLeaguesByUser(user))
        .thenReturn(Collections.emptyList());

    userService.joinLeague(team, league);

    verify(teamRepository, times(1)).saveTeam(team);
}
```

### 5.1.8 Test UT XXX: Visualizza calendario

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione del calendario, tramite mock dei repository.

```
@Test
void testGetAllMatches() {
    League league = new League(null, null, null, null);
    MatchDaySerieA day1 = new MatchDaySerieA(null, null);
    Match m1 = new Match(day1, null, null);
    when(context.getMatchDayRepository().getAllMatchDays())
        .thenReturn(List.of(day1));
    when(context.getMatchRepository().getAllMatchesByMatchDay(day1,
        league))
        .thenReturn(List.of(m1));

    Map<MatchDaySerieA, List<Match>> result = userService.getAllMatches(
        league);
    assertThat(result.get(day1)).containsExactly(m1);
    }
```

### 5.1.9 Test UT XXX: Inserisci formazione

Questo unit test verifica il corretto funzionamento del sistema di salvataggio della LineUp, tramite mock dei repository.

```
@Test
void testSaveLineUp() {
    FantaUser user = new FantaUser("user@test.com", "pwd");
    League league = new League(user, "Test League", new NewsPaper("
        Gazzetta"), "L003");
    MatchDaySerieA matchDay = new MatchDaySerieA("MD1", LocalDate.of
        (2025, 9, 15)); // Monday
    FantaTeam team = new FantaTeam("Dream Team", league, 30, user, new
        HashSet<>());
    Match match = new Match(matchDay, team, team);
    LineUp lineUp = new _433LineUp._443LineUpBuilder(match, team).build()
        ;

    UserService spyService = spy(userService);
    doReturn(team).when(spyService).getFantaTeamByUserAndLeague(league,
        user);
    doReturn(LocalDate.of(2025, 9, 15)).when(spyService).today(); //
        Current Monday

    // Stub repos
    when(context.getMatchDayRepository().getPreviousMatchDay(any()))
        .thenReturn(Optional.empty());
    when(context.getLineUpRepository().getLineUpByMatchAndTeam(match,
        team))
        .thenReturn(Optional.empty());

    spyService.saveLineUp(lineUp);
```

```
21
22      verify(context.getLineUpRepository()).saveLineUp(lineUp);
23  }
```

#### 5.1.10 Test UT XXX: Visualizza prossimo Match

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione del prossimo Match, tramite mock dei repository.

```
1   @Test
2   void testGetNextMatch() {
3       League league = new League(null, null, null, null);
4       FantaTeam team = new FantaTeam(null, league, 0, null, null);
5       MatchDaySerieA prev = new MatchDaySerieA(null, null);
6       MatchDaySerieA next = new MatchDaySerieA(null, null);
7       Match prevMatch = new Match(next, team, team);
8       Match nextMatch = new Match(next, team, team);
9
10      when(context.getMatchDayRepository().getPreviousMatchDay(any()))
11          .thenReturn(Optional.of(prev));
12      when(context.getMatchRepository().getMatchByMatchDay(prev, league,
            team))
13          .thenReturn(prevMatch);
14      when(resultRepository.getResult(prevMatch))
15          .thenReturn(Optional.of(mock(Result.class)));
16      when(context.getMatchDayRepository().getNextMatchDay(any()))
17          .thenReturn(Optional.of(next));
18      when(context.getMatchRepository().getMatchByMatchDay(next, league,
            team))
19          .thenReturn(nextMatch);
20
21      Match result = userService.getNextMatch(league, team, LocalDate.now()
            );
22      assertThat(result).isEqualTo(nextMatch);
23  }
```

#### 5.1.11 Test UT XXX: Visualizza classifica

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione della classifica, tramite mock dei repository.

```
1   @Test
2   void testGetStandings() {
3       League league = new League(null, null, null, null);
4       FantaTeam t1 = mock(FantaTeam.class);
5       FantaTeam t2 = mock(FantaTeam.class);
6
7       when(t1.getPoints()).thenReturn(10);
8       when(t2.getPoints()).thenReturn(20);
9
10      UserService spyService = spy(userService);
11      doReturn(List.of(t1, t2)).when(spyService).getAllFantaTeams(league);
12
13      List<FantaTeam> standings = spyService.getStandings(league);
14
15      assertThat(standings).containsExactly(t2, t1);
16  }
```

#### 5.1.12 Test UT XXX: Scambia giocatori - Invia proposta

Questo unit test verifica il corretto funzionamento del sistema di invio di proposte di scambio di Player tra due FantaTeam tramite mock dei repository.

```
1   @Test
2   void testCreateProposal_HappyPath() {
```

```
    League league = new League(null, null, null, null);
    FantaUser user = new FantaUser(null, null);
    FantaTeam myTeam = spy(new FantaTeam("My Team", league, 0, user, new
        HashSet<>()));
    FantaTeam opponentTeam = new FantaTeam("Opponent", league, 0, user,
        new HashSet<>());
    Player offeredPlayer = new Player.Defender(null, null, null);
    Player requestedPlayer = new Player.Defender(null, null, null);

    Contract offeredContract = new Contract(myTeam, offeredPlayer);
    Contract requestedContract = new Contract(opponentTeam,
        requestedPlayer);
    myTeam.getContracts().add(offeredContract);
    opponentTeam.getContracts().add(requestedContract);

    when(context.getProposalRepository().getProposal(offeredContract,
        requestedContract))
        .thenReturn(Optional.empty());
    when(context.getProposalRepository().saveProposal(any()))
        .thenReturn(true);

    assertThat(userService.createProposal(requestedPlayer, offeredPlayer,
         myTeam, opponentTeam)).isTrue();
}
```

### 5.1.13 Test UT XXX: Scambia giocatori - Accetta proposta

Questo unit test verifica il corretto funzionamento del sistema di accettazione di una proposta di scambio di Player tramite mock dei repository.

```
@Test
void testAcceptProposal() {
    // Setup teams and players
    FantaTeam myTeam = mock(FantaTeam.class);
    FantaTeam offeringTeam = new FantaTeam(null, null, 0, null, null);
    Player offeredPlayer = new Player.Forward(null, null, null);
    Player requestedPlayer = new Player.Midfielder(null, null, null);

    // Contracts
    Contract offeredContract = mock(Contract.class);
    when(offeredContract.getTeam()).thenReturn(offeringTeam);
    when(offeredContract.getPlayer()).thenReturn(offeredPlayer);

    Contract requestedContract = mock(Contract.class);
    when(requestedContract.getTeam()).thenReturn(myTeam);
    when(requestedContract.getPlayer()).thenReturn(requestedPlayer);

    // Proposal
    Proposal.PendingProposal proposal = mock(Proposal.PendingProposal.
        class);
    when(proposal.getRequestedContract()).thenReturn(requestedContract);
    when(proposal.getOfferedContract()).thenReturn(offeredContract);

    // Stub isSameTeam
    when(myTeam.isSameTeam(myTeam)).thenReturn(true);

    // Stub searchContract to return non-empty Optionals
    UserService userServiceSpy = spy(userService);
    doReturn(Optional.of(requestedContract))
        .when(userServiceSpy).searchContract(myTeam, requestedPlayer);
    doReturn(Optional.of(offeredContract))
        .when(userServiceSpy).searchContract(offeringTeam, offeredPlayer)
            ;

    // Run test
    userServiceSpy.acceptProposal(proposal, myTeam);
```

```
35
36      // Verify repository interactions
37      verify(context.getContractRepository())
38          .deleteContract(requestedContract);
39      verify(context.getContractRepository())
40          .deleteContract(offeredContract);
41      verify(context.getContractRepository(), times(2)).saveContract(any(
            Contract.class));
42      verify(context.getProposalRepository()).deleteProposal(proposal);
43  }
```

#### 5.1.14 Test UT XXX: Visualizza squadre

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione di tutti i FantaTeam tramite mock dei repository.

```
1  @Test
2  void testGetAllFantaTeams() {
3      League league = new League(null, "My League", new NewsPaper("Gazzetta
           "), "L999");
4      FantaTeam t1 = new FantaTeam("Team 1", league, 0, new FantaUser("u1",
            "pwd"), Set.of());
5      FantaTeam t2 = new FantaTeam("Team 2", league, 0, new FantaUser("u2",
            "pwd"), Set.of());
6
7      when(context.getTeamRepository().getAllTeams(league))
8          .thenReturn(List.of(t1, t2));
9
10     List<FantaTeam> result = userService.getAllFantaTeams(league);
11
12     assertThat(result).containsExactly(t1, t2);
13     verify(context.getTeamRepository(), times(1)).getAllTeams(league);
14 }
```

#### 5.1.15 Test UT XXX: Visualizza listone giocatori

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione di tutti i giocatori tramite mock dei repository.

```
1  @Test
2  void testGetAllPlayers() {
3      Player p1 = new Player.Goalkeeper("", "", null);
4      Player p2 = new Player.Goalkeeper("", "", null);
5      when(context.getPlayerRepository().findAll()).thenReturn(List.of(p1,
           p2));
6
7      List<Player> result = userService.getAllPlayers();
8      assertThat(result).containsExactly(p1, p2);
9  }
```

#### 5.1.16 Test UT XXX: Assegna voti ai giocatori

Questo unit test verifica il corretto funzionamento del sistema di assegnazione dei voti ai Player tramite mock dei repository.

```
1  @Test
2  void testSetVoteToPlayers_MultipleGrades() {
3      Grade grade2 = mock(Grade.class);
4      when(grade2.getMatchDay()).thenReturn(matchDay);
5      when(grade2.getMark()).thenReturn(15.0);
6
7      NewsPaperService spyService = spy(service);
8      doReturn(Optional.of(matchDay)).when(spyService).getMatchDay();
9
10     spyService.setVoteToPlayers(Set.of(grade, grade2));
```

```
11
12      verify(gradeRepository).saveGrade(grade);
13      verify(gradeRepository).saveGrade(grade2);
14  }
```

### 5.1.17 Test relativi ai JpaRepositories

Per ogni repository implementata tramite JPA è stata realizzata una classe di test per verificare la corretta interazione con il database, sia per salvare nuovi elementi, sia per recuperare dati. Di seguito è proposto un test che verifica il corretto salvataggio nel database di un nuovo FantaTeam.

```
1   @Test
2   @DisplayName("saveTeam() should persist correctly")
3   public void testSaveTeam() {
4
5       entityManager.getTransaction().begin();
6
7       FantaUser user = new FantaUser("mail1", "pswd1");
8       FantaTeam team = new FantaTeam("team1", league, 0, user, new HashSet<
            Contract>());
9
10      entityManager.persist(user);
11
12      fantaTeamRepository.saveTeam(team);
13
14      FantaTeam result = entityManager
15          .createQuery("FROM FantaTeam t WHERE t.league = :league AND t.
                fantaManager = :user", FantaTeam.class)
16          .setParameter("league", league).setParameter("user", user).
                getSingleResult();
17
18      assertThat(result).isEqualTo(team);
19
20      entityManager.close();
21  }
```

Di seguito è invece proposto un test che verifica che il recupero di informazioni dal database sia corretto: in particolare, l'obiettivo è recuperare tutti i voti assegnati in un singolo match.

```
1   @Test
2   @DisplayName("getAllMatchGrades() when two grades have been persisted")
3   public void testGetAllMatchGradesWhenTwoGradesExist() {
4       Player player1 = new Player.Goalkeeper("Gigi", "Buffon", Club.
            JUVENTUS);
5       Player player2 = new Player.Forward("Gigi", "Riva", Club.CAGLIARI);
6
7       Grade voto1 = new Grade(player1, matchDay, 6.0, newsPaper);
8       Grade voto2 = new Grade(player1, matchDay, 8.0, newsPaper);
9       Contract contract1 = new Contract(team1, player1);
10      Contract contract2 = new Contract(team1, player2);
11      sessionFactory.inTransaction(session -> {
12          session.persist(player1);
13          session.persist(player2);
14          session.persist(voto1);
15          session.persist(voto2);
16          session.persist(contract1);
17          session.persist(contract2);
18      });
19
20      assertThat(gradeRepository.getAllMatchGrades(match, newsPaper)).
            containsExactly(voto1, voto2);
21  }
```

### 5.1.18 Test relativi al Domain Model

Inoltre, sono stati creati degli unit test per verificare il corretto comportamento di alcuni metodi presenti nelle classi del Domain Model:

- la classe *_433LineUp* usa il pattern **Builder**, perciò si testa che la costruzione di nuovi oggetti sia corretta;
- la classe *FantaTeamViewer*, sfruttando il pattern **Visitor**, si occupa di recuperare gli insiemi di giocatori appartenenti ad un FantaTeam divisi per ruoli basandosi sui contratti del FantaTeam;
- la classe *LineUpViewer*, similmente a *FantaTeamViewer*, si occupa di recuperare gli insiemi di giocatori presenti in una LineUp divisi per ruoli.

## 5.2 Integration Tests

I seguenti test verificano che le varie componenti cooperino correttamente per l'implementazione degli use case richiesti.

Vanno aggiunti i test per il Login?

### 5.2.1 Test IT IT3: Crea fantalega

Questo unit test verifica il corretto funzionamento del sistema di creazione di una nuova League, implementando il template UC-01.

```
@Test
void createLeague() {

    entityManager.getTransaction().begin();

    FantaUser admin = new FantaUser("mail", "pswd");
    fantaUserRepository.saveFantaUser(admin);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
    newspaperRepository.saveNewsPaper(newsPaper);

    entityManager.getTransaction().commit();

    adminUserService.createLeague("lega", admin, newsPaper, "1234");

    Optional<League> result = leagueRepository.getLeagueByCode("1234");

    assertThat(result.isPresent());
    League league = result.get();

    assertThat(league.getName()).isEqualTo("lega");
    assertThat(league.getAdmin()).isEqualTo(admin);
    assertThat(league.getNewsPaper()).isEqualTo(newsPaper);
    assertThat(league.getLeagueCode()).isEqualTo("1234");
}
```

### 5.2.2 Test IT IT4: Genera calendario

Questo unit test verifica il corretto funzionamento del sistema di generazione del calendario della League, implementando il template XXXXXXXX .

collegare al template

```
@Test
void generateCalendar() {

    entityManager.getTransaction().begin();

    FantaUser admin = new FantaUser("mail", "pswd");
    FantaUser user = new FantaUser("user2", "pswd2");
    fantaUserRepository.saveFantaUser(admin);
    fantaUserRepository.saveFantaUser(user);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
    newspaperRepository.saveNewsPaper(newsPaper);

    League league = new League(admin, "lega", newsPaper, "0000");
    leagueRepository.saveLeague(league);

```

```
17      FantaTeam team1 = new FantaTeam("team1", league, 0, admin, new
            HashSet<Contract>());
18      FantaTeam team2 = new FantaTeam("team2", league, 0, user, new HashSet
            <Contract>());
19
20      fantaTeamRepository.saveTeam(team1);
21      fantaTeamRepository.saveTeam(team2);
22
23      List<MatchDaySerieA> matchDays = new ArrayList<MatchDaySerieA>();
24      for (int i = 0; i < 38; i++) {
25          matchDays.add(new MatchDaySerieA("Match " + String.valueOf(i),
                LocalDate.of(2025, 9, 7).plusWeeks(i)));
26      }
27
28      sessionFactory.inTransaction(t -> {
29          for (MatchDaySerieA matchDaySerieA : matchDays) {
30              t.persist(matchDaySerieA);
31          }
32      });
33
34      entityManager.getTransaction().commit();
35
36      adminUserService.generateCalendar(league);
37
38      for (MatchDaySerieA matchDaySerieA : matchDays) {
39          Match matchByMatchDay = matchRepository.getMatchByMatchDay(
                matchDaySerieA, league, team1);
40
41          assertThat(matchByMatchDay.getMatchDaySerieA()).isEqualTo(
                matchDaySerieA);
42          assertThat(matchByMatchDay.getTeam1().equals(team1) ||
                matchByMatchDay.getTeam2().equals(team1)).isTrue();
43      }
44  }
```

### 5.2.3 Test IT IT5: Assegna giocatori alle rose

Questo unit test verifica il corretto funzionamento del sistema di assegnazione dei Player ai FantaTeam che li hanno acquistati, implementando il template UC-02.

```
1   @Test
2   void setPlayersToTeam() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser admin = new FantaUser("mail", "pswd");
7       fantaUserRepository.saveFantaUser(admin);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      League league = new League(admin, "lega", newsPaper, "1234");
13      leagueRepository.saveLeague(league);
14
15      FantaTeam team = new FantaTeam("", league, 0, admin, Set.of());
16      fantaTeamRepository.saveTeam(team);
17
18      Player.Forward player = new Player.Forward("Lionel", "Messi", Club.
            CREMONESE);
19      Player.Goalkeeper player2 = new Player.Goalkeeper("Gigi", "Buffon",
            Club.JUVENTUS);
20      playerRepository.addPlayer(player);
21      playerRepository.addPlayer(player2);
22
23      entityManager.getTransaction().commit();
24
```

```
25      adminUserService.setPlayerToTeam(team, player);
26      adminUserService.setPlayerToTeam(team, player2);
27
28      Optional<Contract> contract = contractRepository.getContract(team,
            player);
29      assertThat(contract).isPresent();
30      Contract found = contract.get();
31      assertThat(found.getPlayer()).isEqualTo(player);
32      assertThat(found.getTeam()).isEqualTo(team);
33
34      Optional<Contract> contract2 = contractRepository.getContract(team,
            player);
35      assertThat(contract2).isPresent();
36      Contract found2 = contract.get();
37      assertThat(found2.getPlayer()).isEqualTo(player);
38      assertThat(found2.getTeam()).isEqualTo(team);
39
40  }
```

### 5.2.4 Test IT IT6: Calcola voti della giornata

Questo unit test verifica il corretto funzionamento del sistema di calcolo dei voti della giornata una volta terminata, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
2   void calculateGrades() {
3
4       entityManager.getTransaction().begin();
5
6       // Users
7       FantaUser admin = new FantaUser("mail", "pswd");
8       FantaUser user = new FantaUser("mail2", "pswd2");
9
10      fantaUserRepository.saveFantaUser(admin);
11      fantaUserRepository.saveFantaUser(user);
12
13      NewsPaper newsPaper = new NewsPaper("Gazzetta");
14      newspaperRepository.saveNewsPaper(newsPaper);
15
16      League league = new League(admin, "lega", newsPaper, "0000");
17      leagueRepository.saveLeague(league);
18
19      // Teams
20      FantaTeam team1 = new FantaTeam("team1", league, 0, admin, new
            HashSet<Contract>());
21      FantaTeam team2 = new FantaTeam("team2", league, 0, user, new HashSet
            <Contract>());
22
23      fantaTeamRepository.saveTeam(team1);
24      fantaTeamRepository.saveTeam(team2);
25
26      // MatchDays
27      LocalDate matchDate = LocalDate.of(2025, 9, 14);
28      MatchDaySerieA prevDay = new MatchDaySerieA("Day0", matchDate.
            minusWeeks(1));
29      MatchDaySerieA dayToCalc = new MatchDaySerieA("Day1", matchDate);
30
31      entityManager.persist(prevDay);
32      entityManager.persist(dayToCalc);
33
34      // Match
35      Match prevMatch = new Match(prevDay, team1, team2);
36      Match match = new Match(dayToCalc, team1, team2);
37
38      matchRepository.saveMatch(prevMatch);
39      matchRepository.saveMatch(match);
```

```
40
41      // Players
42      Goalkeeper gk1 = new Goalkeeper("Gianluigi", "Buffon", Club.JUVENTUS)
            ;
43      Goalkeeper gk2 = new Goalkeeper("Samir", "Handanovic", Club.INTER);
44
45      Defender d1 = new Defender("Paolo", "Maldini", Club.MILAN);
46      Defender d2 = new Defender("Franco", "Baresi", Club.JUVENTUS);
47      Defender d3 = new Defender("Alessandro", "Nesta", Club.LAZIO);
48      Defender d4 = new Defender("Giorgio", "Chiellini", Club.JUVENTUS);
49      Defender d5 = new Defender("Leonardo", "Bonucci", Club.JUVENTUS);
50
51      Midfielder m1 = new Midfielder("Andrea", "Pirlo", Club.JUVENTUS);
52      Midfielder m2 = new Midfielder("Daniele", "De Rossi", Club.ROMA);
53      Midfielder m3 = new Midfielder("Marco", "Verratti", Club.CREMONESE);
54      Midfielder m4 = new Midfielder("Claudio", "Marchisio", Club.JUVENTUS)
            ;
55
56      Forward f1 = new Forward("Roberto", "Baggio", Club.BOLOGNA);
57      Forward f2 = new Forward("Francesco", "Totti", Club.ROMA);
58      Forward f3 = new Forward("Alessandro", "Del Piero", Club.JUVENTUS);
59      Forward f4 = new Forward("Lorenzo", "Insigne", Club.NAPOLI);
60
61      List<Player> players = List.of(gk1, gk2, d1, d2, d3, d4, d5, m1, m2,
            m3, m4, f1, f2, f3, f4);
62
63      players.forEach(playerRepository::addPlayer);
64
65      for (Player player : players) {
66          contractRepository.saveContract(new Contract(team1, player));
67          contractRepository.saveContract(new Contract(team2, player));
68      }
69
70      // LineUps
71      LineUp lineup1 = new _433LineUp._443LineUpBuilder(match, team1).
            withGoalkeeper(gk1)
72              .withDefenders(d1, d2, d3, d4).withMidfielders(m1, m2, m3).
                    withForwards(f1, f2, f3)
73              .withSubstituteGoalkeepers(List.of(gk2)).
                    withSubstituteDefenders(List.of(d5))
74              .withSubstituteMidfielders(List.of(m4)).
                    withSubstituteForwards(List.of(f4)).build();
75      LineUp lineup2 = new _433LineUp._443LineUpBuilder(match, team2).
            withGoalkeeper(gk1)
76              .withDefenders(d1, d2, d3, d4).withMidfielders(m1, m2, m3).
                    withForwards(f1, f2, f3)
77              .withSubstituteGoalkeepers(List.of(gk2)).
                    withSubstituteDefenders(List.of(d5))
78              .withSubstituteMidfielders(List.of(m4)).
                    withSubstituteForwards(List.of(f4)).build();
79
80      lineUpRepository.saveLineUp(lineup1);
81      lineUpRepository.saveLineUp(lineup2);
82
83      // Grades
84      Grade grade1 = new Grade(gk1, dayToCalc, 70.0, newsPaper);
85      Grade grade2 = new Grade(gk2, dayToCalc, 60.0, newsPaper);
86
87      gradeRepository.saveGrade(grade1);
88      gradeRepository.saveGrade(grade2);
89
90      entityManager.getTransaction().commit();
91
92      AdminUserService service = new AdminUserService(transactionManager) {
93          @Override
94          protected LocalDate today() {
```

```
            return LocalDate.of(2025, 9, 16); // after 14/09
        }
    };

    service.calculateGrades(admin, league);

    List<Grade> allMatchGrades = gradeRepository.getAllMatchGrades(match,
        newsPaper);
    assertThat(allMatchGrades.size()).isEqualTo(2);
    assertThat(allMatchGrades.get(0)).isEqualTo(grade1);
    assertThat(allMatchGrades.get(1)).isEqualTo(grade2);

}
```

### 5.2.5 Test IT IT7: Unisciti alla lega

Questo unit test verifica il corretto funzionamento del sistema di ingresso in una League, implementando il template XXXXXXXX .

collegare al template

```
@Test
void joinLeague() {

    entityManager.getTransaction().begin();

    FantaUser user = new FantaUser("user@test.com", "pwd");
    fantaUserRepository.saveFantaUser(user);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
    newspaperRepository.saveNewsPaper(newsPaper);

    League league = new League(user, "Test League", newsPaper, "1234");
    leagueRepository.saveLeague(league);

    FantaTeam team = new FantaTeam("Team A", league, 0, user, new HashSet
        <Contract>());

    entityManager.getTransaction().commit();

    userService.joinLeague(team, league);

    Optional<League> result = leagueRepository.getLeagueByCode("1234");
    assertThat(result).isPresent();

    League resultLeague = result.get();
    assertThat(resultLeague.getAdmin()).isEqualTo(user);
    assertThat(resultLeague.getLeagueCode()).isEqualTo("1234");
    assertThat(resultLeague.getName()).isEqualTo("Test League");
    assertThat(resultLeague.getNewsPaper()).isEqualTo(newsPaper);
}
```

### 5.2.6 Test IT IT8: Visualizza calendario

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione del calendario, cioè della lista dei Match, implementando il template XXXXXXXX .

collegare al template

```
@Test
void testGetAllMatches() {

    entityManager.getTransaction().begin();

    FantaUser user1 = new FantaUser("user1@test.com", "pwd");
    fantaUserRepository.saveFantaUser(user1);
    FantaUser user2 = new FantaUser("user2@test.com", "pwd");
    fantaUserRepository.saveFantaUser(user2);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
```

```
12      newspaperRepository.saveNewsPaper(newsPaper);
13
14      League league = new League(user1, "Test League", newsPaper, "1234");
15      leagueRepository.saveLeague(league);
16
17      FantaTeam team1 = new FantaTeam("Team A", league, 0, user1, new
            HashSet<Contract>());
18      fantaTeamRepository.saveTeam(team1);
19      FantaTeam team2 = new FantaTeam("Team B", league, 0, user2, new
            HashSet<Contract>());
20      fantaTeamRepository.saveTeam(team2);
21
22      MatchDaySerieA day1 = new MatchDaySerieA("MD1", LocalDate.of(2025, 9,
             7));
23      matchDayRepository.saveMatchDay(day1);
24
25      Match m1 = new Match(day1, team1, team2);
26      matchRepository.saveMatch(m1);
27
28      entityManager.getTransaction().commit();
29
30      Map<MatchDaySerieA, List<Match>> result = userService.getAllMatches(
            league);
31
32      assertThat(result.get(day1).size()).isEqualTo(1);
33      Match resultMatch = result.get(day1).get(0);
34
35      assertThat(resultMatch.getMatchDaySerieA().getName()).isEqualTo("MD1"
            );
36  }
```

### 5.2.7 Test IT IT9: Inserisci formazione

Questo unit test verifica il corretto funzionamento del sistema di salvataggio della LineUp, implementando il template UC-03.

```
1   @Test
2   void testSaveLineUp() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser user = new FantaUser("user@test.com", "pwd");
7       fantaUserRepository.saveFantaUser(user);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      League league = new League(user, "Test League", newsPaper, "L003");
13      leagueRepository.saveLeague(league);
14
15      MatchDaySerieA matchDay = new MatchDaySerieA("MD1", LocalDate.now().
            plusWeeks(1)); // Monday
16      matchDayRepository.saveMatchDay(matchDay);
17
18      FantaTeam team = new FantaTeam("Dream Team", league, 30, user, new
            HashSet<>());
19      fantaTeamRepository.saveTeam(team);
20
21      Match match = new Match(matchDay, team, team);
22      matchRepository.saveMatch(match);
23
24      LineUp lineUp = new _433LineUp._443LineUpBuilder(match, team).build()
            ;
25
26      entityManager.getTransaction().commit();
27
```

```
28      userService.saveLineUp(lineUp);
29
30      Optional<LineUp> result = lineUpRepository.getLineUpByMatchAndTeam(
            match, team);
31      assertThat(result).isPresent();
32
33      LineUp resultLineUp = result.get();
34      assertThat(resultLineUp.getMatch()).isEqualTo(match);
35      assertThat(resultLineUp.getTeam()).isEqualTo(team);
36
37  }
```

### 5.2.8 Test IT IT10: Visualizza prossimo Match

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione del prossimo Match, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
2   void testGetNextMatch() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser user = new FantaUser("user@test.com", "pwd");
7       fantaUserRepository.saveFantaUser(user);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      League league = new League(user, "Test League", newsPaper, "L003");
13      leagueRepository.saveLeague(league);
14
15      FantaTeam team = new FantaTeam("Team", league, 30, user, new HashSet
            <>());
16      FantaTeam team2 = new FantaTeam("Team2", league, 40, user, new
            HashSet<>());
17      fantaTeamRepository.saveTeam(team);
18      fantaTeamRepository.saveTeam(team2);
19
20      MatchDaySerieA prevMatchDay = new MatchDaySerieA("MD1", LocalDate.now
            ().minusWeeks(1));
21      MatchDaySerieA nextMatchDay = new MatchDaySerieA("MD2", LocalDate.now
            ().plusWeeks(1));
22      matchDayRepository.saveMatchDay(prevMatchDay);
23      matchDayRepository.saveMatchDay(nextMatchDay);
24
25      Match prevMatch = new Match(prevMatchDay, team, team2);
26      Match nextMatch = new Match(nextMatchDay, team, team2);
27      matchRepository.saveMatch(prevMatch);
28      matchRepository.saveMatch(nextMatch);
29
30      Result prevResults = new Result(20, 50, 1, 2, prevMatch);
31      resultsRepository.saveResult(prevResults);
32
33      entityManager.getTransaction().commit();
34
35      Match result = userService.getNextMatch(league, team, LocalDate.now()
            );
36      assertThat(result.getMatchDaySerieA().getName()).isEqualTo("MD2");
37  }
```

### 5.2.9 Test IT IT11: Visualizza classifica

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione della classifica, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
```

```
void testGetStandings() {

    entityManager.getTransaction().begin();

    FantaUser user = new FantaUser("user@test.com", "pwd");
    fantaUserRepository.saveFantaUser(user);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
    newspaperRepository.saveNewsPaper(newsPaper);

    League league = new League(user, "Test League", newsPaper, "L003");
    leagueRepository.saveLeague(league);

    FantaTeam team1 = new FantaTeam("Team1", league, 10, user, new
        HashSet<>());
    FantaTeam team2 = new FantaTeam("Team2", league, 70, user, new
        HashSet<>());
    fantaTeamRepository.saveTeam(team1);
    fantaTeamRepository.saveTeam(team2);

    entityManager.getTransaction().commit();

    List<FantaTeam> standings = userService.getStandings(league);

    assertThat(standings.get(0).getName()).isEqualTo("Team2");
    assertThat(standings.get(1).getName()).isEqualTo("Team1");
}
```

#### 5.2.10 Test IT IT12: Scambia giocatori - Invia proposta

Questo unit test verifica il corretto funzionamento del sistema di invio di proposte di scambio di Player tra due FantaTeam, implementando il template UC-04.

```
@Test
void testCreateProposal_HappyPath() {

    entityManager.getTransaction().begin();

    FantaUser user = new FantaUser("user@test.com", "pwd");
    fantaUserRepository.saveFantaUser(user);

    NewsPaper newsPaper = new NewsPaper("Gazzetta");
    newspaperRepository.saveNewsPaper(newsPaper);

    League league = new League(user, "Test League", newsPaper, "L003");
    leagueRepository.saveLeague(league);

    FantaTeam myTeam = new FantaTeam("My Team", league, 0, user, new
        HashSet<>());
    FantaTeam opponentTeam = new FantaTeam("Opponent", league, 0, user,
        new HashSet<>());
    fantaTeamRepository.saveTeam(myTeam);
    fantaTeamRepository.saveTeam(opponentTeam);

    Player offeredPlayer = new Player.Defender("Mario", "Rossi", Club.
        ATALANTA);
    Player requestedPlayer = new Player.Defender("Luigi", "Verdi", Club.
        BOLOGNA);
    playerRepository.addPlayer(offeredPlayer);
    playerRepository.addPlayer(requestedPlayer);

    Contract offeredContract = new Contract(myTeam, offeredPlayer);
    Contract requestedContract = new Contract(opponentTeam,
        requestedPlayer);
    myTeam.getContracts().add(offeredContract);
    opponentTeam.getContracts().add(requestedContract);
```

```
29      contractRepository.saveContract(offeredContract);
30      contractRepository.saveContract(requestedContract);
31
32      entityManager.getTransaction().commit();
33
34      assertThat(userService.createProposal(requestedPlayer, offeredPlayer,
             myTeam, opponentTeam)).isTrue();
35
36      Optional<Proposal> result = proposalRepository.getProposal(
            offeredContract, requestedContract);
37      assertThat(result).isPresent();
38      Proposal resultProposal = result.get();
39      assertThat(resultProposal.getOfferedContract()).isEqualTo(
            offeredContract);
40      assertThat(resultProposal.getRequestedContract()).isEqualTo(
            requestedContract);
41  }
```

### 5.2.11 Test IT IT13: Scambia giocatori - Accetta proposta

Questo unit test verifica il corretto funzionamento del sistema di accettazione di una proposta di scambio di Player, implementando il template UC-05.

```
1   @Test
2   void testAcceptProposal() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser user = new FantaUser("user@test.com", "pwd");
7       fantaUserRepository.saveFantaUser(user);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      League league = new League(user, "Test League", newsPaper, "L003");
13      leagueRepository.saveLeague(league);
14
15      FantaTeam myTeam = new FantaTeam("My Team", league, 0, user, new
            HashSet<>());
16      FantaTeam offeringTeam = new FantaTeam("Opponent", league, 0, user,
            new HashSet<>());
17      fantaTeamRepository.saveTeam(myTeam);
18      fantaTeamRepository.saveTeam(offeringTeam);
19
20      Player requestedPlayer = new Player.Defender("Luigi", "Verdi", Club.
            BOLOGNA);
21      Player offeredPlayer = new Player.Defender("Mario", "Rossi", Club.
            ATALANTA);
22      playerRepository.addPlayer(requestedPlayer);
23      playerRepository.addPlayer(offeredPlayer);
24
25      Contract offeredContract = new Contract(offeringTeam, offeredPlayer);
26      Contract requestedContract = new Contract(myTeam, requestedPlayer);
27      offeringTeam.getContracts().add(offeredContract);
28      myTeam.getContracts().add(requestedContract);
29      contractRepository.saveContract(offeredContract);
30      contractRepository.saveContract(requestedContract);
31
32      Proposal.PendingProposal proposal = new PendingProposal(
            offeredContract, requestedContract);
33      proposalRepository.saveProposal(proposal);
34
35      entityManager.getTransaction().commit();
36
37      userService.acceptProposal(proposal, myTeam);
38
```

```
39      assertThat(contractRepository.getContract(offeringTeam, offeredPlayer
            )).isEmpty();
40      assertThat(contractRepository.getContract(offeringTeam,
            requestedPlayer)).isPresent();
41
42      assertThat(contractRepository.getContract(myTeam, requestedPlayer)).
            isEmpty();
43      assertThat(contractRepository.getContract(myTeam, offeredPlayer)).
            isPresent();
44  }
```

### 5.2.12 Test IT IT14: Visualizza squadre

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione di tutti i FantaTeam, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
2   void testGetAllFantaTeams() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser user = new FantaUser("user@test.com", "pwd");
7       fantaUserRepository.saveFantaUser(user);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      League league = new League(user, "Test League", newsPaper, "L003");
13      leagueRepository.saveLeague(league);
14
15      FantaTeam team1 = new FantaTeam("Team1", league, 10, user, new
            HashSet<>());
16      FantaTeam team2 = new FantaTeam("Team2", league, 70, user, new
            HashSet<>());
17      fantaTeamRepository.saveTeam(team1);
18      fantaTeamRepository.saveTeam(team2);
19
20      entityManager.getTransaction().commit();
21
22      List<FantaTeam> result = userService.getAllFantaTeams(league);
23
24      assertThat(result.size()).isEqualTo(2);
25      assertThat(result.get(0).getLeague()).isEqualTo(result.get(1).
            getLeague());
26      assertThat(result.get(0).getName()).isIn("Team1", "Team2");
27      assertThat(result.get(1).getName()).isIn("Team1", "Team2");
28      assertThat(result.get(0).getPoints()).isIn(10, 70);
29      assertThat(result.get(1).getPoints()).isIn(10, 70);
30  }
```

### 5.2.13 Test IT IT15: Visualizza listone giocatori

Questo unit test verifica il corretto funzionamento del sistema di visualizzazione di tutti i giocatori, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
2   void testGetAllPlayers() {
3
4       entityManager.getTransaction().begin();
5
6       Player p1 = new Player.Defender("Mario", "Rossi", Club.ATALANTA);
7       Player p2 = new Player.Defender("Luigi", "Verdi", Club.BOLOGNA);
8       playerRepository.addPlayer(p1);
9       playerRepository.addPlayer(p2);
10
11      entityManager.getTransaction().commit();
```

```
12
13      List<Player> result = userService.getAllPlayers();
14      assertThat(result).containsExactly(p1, p2);
15  }
```

### 5.2.14 Test IT IT16: Assegna voti ai giocatori

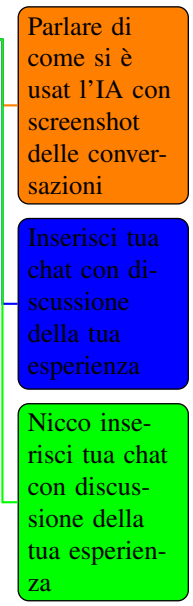
Questo unit test verifica il corretto funzionamento del sistema di assegnazione dei voti ai Player, implementando il template XXXXXXXX .

collegare al template

```
1   @Test
2   public void assignGradesToPlayers() {
3
4       entityManager.getTransaction().begin();
5
6       FantaUser admin = new FantaUser("mail", "pswd");
7       fantaUserRepository.saveFantaUser(admin);
8
9       NewsPaper newsPaper = new NewsPaper("Gazzetta");
10      newspaperRepository.saveNewsPaper(newsPaper);
11
12      Player player = new Player.Forward("player", "1", Club.ATALANTA);
13      Player player2 = new Player.Forward("player", "2", Club.ATALANTA);
14      playerRepository.addPlayer(player);
15      playerRepository.addPlayer(player2);
16
17      MatchDaySerieA previousDay = new MatchDaySerieA("prima giornata",
            LocalDate.of(2020, 1, 13));
18      MatchDaySerieA matchDay = new MatchDaySerieA("seconda giornata",
            LocalDate.of(2025, 9, 20));
19      MatchDaySerieA nextDay = new MatchDaySerieA("terza giornata",
            LocalDate.of(2020, 1, 26));
20      entityManager.persist(previousDay);
21      entityManager.persist(matchDay);
22      entityManager.persist(nextDay);
23
24      League league = new League(admin, "lega", newsPaper, "1234");
25      leagueRepository.saveLeague(league);
26
27      FantaTeam team1 = new FantaTeam("", league, 0, admin, Set.of());
28      FantaTeam team2 = new FantaTeam("", league, 0, admin, Set.of());
29      fantaTeamRepository.saveTeam(team1);
30      fantaTeamRepository.saveTeam(team2);
31
32      Match match = new Match(matchDay, team1, team2);
33      matchRepository.saveMatch(match);
34
35      Grade grade = new Grade(player, matchDay, 25.0, newsPaper);
36      Grade grade2 = new Grade(player2, matchDay, 20.0, newsPaper);
37
38      entityManager.getTransaction().commit();
39
40      newspaperservice = new NewsPaperService(transactionManager) {
41          @Override
42          protected LocalDate today() {
43              return LocalDate.of(2025, 9, 22); // after 21/09
44          }
45      };
46
47      Set<Player> players = newspaperservice.getPlayersToGrade(Player.Club.
            ATALANTA);
48      assertThat(players.size()).isEqualTo(2);
49      assertThat(players).anyMatch(t -> t.getName() == "player" && t.
            getSurname() == "1");
50      assertThat(players).anyMatch(t -> t.getName() == "player" && t.
            getSurname() == "2");
```

```
51
52      newspaperservice.setVoteToPlayers(Set.of(grade, grade2));
53      assertThat(gradeRepository.getAllMatchGrades(match, newsPaper));
54
55  }
```

# 6 Interazione con gli LLMS



# Riferimenti bibliografici

[1]

Cosa ci mettiamo libro del vicario su jpa? poi?