
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

# 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 UT XXX: Registrazione di un FantaUser

Va tenuto qui?

Questo unit test verifica il corretto funzionamento del sistema di registrazione di un FantaUser, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
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, cioè della lista dei Match, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
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, implementando il template XXXXXXXX .

collegare al template

```
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, implementando il template XXXXXXXX .

collegare al template

```
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, implementando il template XXXXXXXX .

collegare al template

```
@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, implementando il template XXXXXXXX .

collegare al template

```
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, implementando il template XXXXXXXX .

collegare al template

```
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, implementando il template XXXXXXXX .

collegare al template

```
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

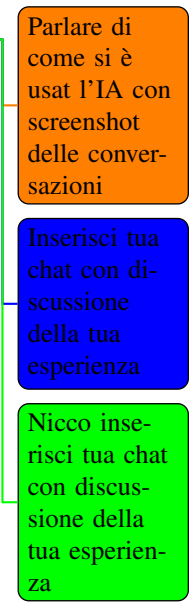
Inoltre, sono stati creati degli unit test per verificare il corretto comportamento di alcuni metodi presenti nelle classi del Domain Model:

- 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

### 5.2.1 Test IT XXX:

# 6 Interazione con gli LLMS



# Riferimenti bibliografici

[1]

Cosa ci mettiamo libro del vicario su jpa? poi?