



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

<b>1</b>	<b>Introduzione generale</b>	<b>2</b>
1.1	Statement . . . . .	2
1.1.1	Actors . . . . .	2
1.2	Architettura . . . . .	2
1.2.1	Dipendenze . . . . .	2
<b>2</b>	<b>Strumenti utilizzati</b>	<b>3</b>
<b>3</b>	<b>Progettazione</b>	<b>4</b>
<b>4</b>	<b>Implementazione</b>	<b>5</b>
<b>5</b>	<b>ApplicationTests</b>	<b>6</b>
<b>6</b>	<b>Interazione con gli LLMS</b>	<b>7</b>

## 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

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 mantenere e da estendere.

Aggiungere  
Figura, pro-  
babilmente  
quella ge-  
nerale non  
specifica va  
creata

- **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 in 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 cor-  
reggimi se  
sbaglio su  
Jpa. Inoltre  
le interfacce  
dei reposi-  
tory vanno  
bene qui o è  
meglio spo-  
starle?

### 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, Use-Cases 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 classe e del database in memoria

Se vuoi scrivi della gui in generale

Nicco crea gli use case template scegli 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 database ed in progettazione fare un'introduzione

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

## 5 ApplicationTests

---

Parlare di  
unit, integra-  
tion

## 6 Interazione con gli LLMS

Parlare di come si è usat l'IA con screenshot delle conversazioni

Inserisci tua chat con discussione della tua esperienza

Nicco inserisci tua chat con discussione della tua esperienza



## Riferimenti bibliografici

[1]

Cosa ci mettiamo libro del vicario su jpa? poi?