Università degli Studi di Salerno
Corso di Ingegneria del Software

FantaUnisa
Object Design Document
Versione 0.4

Data: 20/12/2025

Partecipanti:

Nome	Matricola
Clavino Antonio	0512119692
Corona Francesco	0512119827
Sabetta Francesco	0512118990
Tiberini Monica	0512120226

Scritto da:	Tutti i partecipanti.

Revision History

Data	Versione	Descrizione	Autore
18/12/2025	0.1	Stesura interfacce Control	Sabetta Francesco
19/12/2025	0.2	Stesura interfacce Persistence	Sabetta Francesco
19/12/2025	0.3	Stesura interfacce Utils	Sabetta Francesco
20/12/2025	0.4	Stesura interfacce Model	Sabetta Francesco

Indice

1. INTRODUZIONE

1.1 Object design trade-offs

1.2 Interface documentation guidelines

1.3 Definitions, acronyms, and abbreviations

1.4 References

- Requirements Analysis Document FantaUnisa.
- System Design Document FantaUnisa.

1.5 Overview

2. PACKAGES

3. CLASS INTERFACES

3.1 Control

Questa sezione definisce le Servlet che gestiscono la logica applicativa, separate per responsabilità specifica.

Nome classe	**RegisterServlet**
Descrizione	Gestisce la registrazione di nuovi utenti.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: registerUser*
Pre-condizioni	L'indirizzo e-mail fornito non deve essere già presente nel database.
Post-condizioni	Viene creato un nuovo record nella tabella User e inviata una conferma di avvenuta registrazione.
Invariante	Ogni account creato con successo deve acquisire obbligatoriamente il ruolo di “Fantallenatore” e non può mai possedere permessi amministrativi (es. Gestore).

Nome classe	**LoginServlet**
Descrizione	Gestisce esclusivamente l'autenticazione dell'utente nel sistema.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void

	Action: loginUser
Pre-condizioni	La sessione utente non deve essere già attiva (stato = GUEST).
Post-condizioni	Se le credenziali sono corrette, viene creato un token di sessione e l'utente diventa AUTHENTICATED.
Invariante	La password inserita deve essere verificata confrontando il suo hash, mai in chiaro.

Nome classe	**ProfileServlet**
Descrizione	Gestisce la visualizzazione dei dati del profilo.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: viewProfile*
Pre-condizioni	L'utente deve possedere un token di sessione valido.
Post-condizioni	Vengono caricati e mostrati i dati anagrafici dell'utente.
Invariante	-

Nome classe	**UpdateProfileServlet**
Descrizione	Gestisce specificamente la modifica dei dati del profilo utente.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: updateProfile*
Pre-condizioni	Per modificare la password,

	l'utente deve fornire la vecchia password per conferma.
Post-condizioni	I dati utente sono aggiornati persistentemente nel DB.
Invariante	Non è possibile modificare l'username o l'e-mail dell'utente.

Nome classe	**UserServlet**
Descrizione	Servlet amministrativa per la gestione degli altri utenti.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: banUser, suspendUser*
Pre-condizioni	L'utente richiedente deve avere il ruolo di "Gestore degli utenti".
Post-condizioni	**banUser**: Lo stato dell'utente target viene impostato su "Banned" e la sua sessione invalidata.
Invariante	Un gestore non può bannare se stesso o un altro gestore di pari livello.

Nome classe	**PlayersServlet**
Descrizione	Gestisce il caricamento del file Excel per popolare o aggiornare il database dei calciatori.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: uploadExcel*

Pre-condizioni	Il file in input deve avere estensione .xlsx e rispettare il template delle colonne definito.
Post-condizioni	Il database viene popolato con i nuovi giocatori o aggiornato con le nuove statistiche.
Invariante	Ogni giocatore importato deve avere un ruolo valido (P, D, C, A).

Nome classe	**SquadServlet**
Descrizione	Gestisce la composizione della rosa (aggiunta/rimozione calciatori).
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: addPlayer, removePlayer*
Pre-condizioni	Non può essere inserito un giocatore già inserito precedentemente.
Post-condizioni	Il giocatore viene associato all'ID utente nella tabella Squad.
Invariante	La rosa non può mai contenere più di 25 giocatori totali.

Nome classe	**FormationServlet**
Descrizione	Gestisce il calcolo e il salvataggio della formazione titolare settimanale.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: saveFormation*

Pre-condizioni	I giocatori devono essere inseriti nei rispettivi ruoli.
Post-condizioni	Viene salvata una lista di 11 titolari e relative riserve per la giornata corrente.
Invariante	La formazione deve rispettare i vincoli del modulo (es. 4-4-2 richiede 4 difensori).

Nome classe	**PostServlet**
Descrizione	Gestisce la creazione e l'eliminazione dei post nella community.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: createPost, deletePost*
Pre-condizioni	**createPost**: Il post deve contenere testo o un allegato (formazione) non vuoto.
Post-condizioni	Il post diventa visibile nel feed pubblico di tutti gli utenti.
Invariante	Un utente può eliminare solo i post di cui è autore.

Nome classe	**CommentServlet**
Descrizione	Gestisce l'aggiunta di commenti sotto i post degli utenti.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: addComment*

Pre-condizioni	Il post a cui si riferisce il commento deve esistere e non essere stato eliminato.
Post-condizioni	Il commento viene appeso alla lista dei commenti del post e aggiornato il contatore.
Invariante	-

Nome classe	**ReactionServlet**
Descrizione	Gestisce l'aggiunta o la rimozione delle reazioni ai post.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: addReaction, removeReaction*
Pre-condizioni	-
Post-condizioni	Se la reazione esisteva, viene rimossa; se non esisteva, viene aggiunta.
Invariante	Un utente può mettere al massimo una reazione per ogni singolo post.

Nome classe	**ModuleServlet**
Descrizione	Gestisce esclusivamente la scelta e la modifica del modulo tattico (es. 4-3-3, 3-5-2) utilizzato per il calcolo della formazione.
Signature dei metodi	# doPost(HttpServletRequest, HttpServletResponse): void *Action: setModule*

Pre-condizioni	Il modulo inviato deve appartenere alla lista dei moduli supportati dal sistema.
Post-condizioni	Il modulo selezionato viene associato persistentemente all'utente
Invariante	Un modulo valido deve prevedere sempre la somma di 10 giocatori di movimento + 1 portiere.

3.2 Persistence

Questa sezione definisce le classi responsabili della connessione al database e delle operazioni CRUD sulle entità del sistema.

Nome classe	**DBConnection**
Descrizione	Gestisce la connessione al database.
Signature dei metodi	+ getConnection(): Connection + close(): void
Pre-condizioni	**getConnection**: Il driver JDBC deve essere caricato e le credenziali di accesso al DB configurate correttamente.
Post-condizioni	**getConnection**: Restituisce un oggetto “Connection” valido e aperto. **close**: Rilascia la connessione e le risorse JDBC.
Invariante	Il numero di connessioni attive non deve mai superare il parametro MAX_CONNECTIONS definito.

Nome classe	UserDAO
Descrizione	Interfaccia di accesso alla tabella utenti. Gestisce la persistenza dei dati anagrafici e delle credenziali.
Signature dei metodi	+ doSave(user: User): void + doRetrieveByEmail(email: String): User + doRetrieveById(id: int): User + doUpdateProfile(user: User): void + doDelete(id: int): void
Pre-condizioni	**doSave**: L'oggetto User non deve essere nullo e l'e-mail non deve esistere nel DB.
Post-condizioni	**doRetrieveByEmail**: Restituisce l'oggetto User popolato se trovato, altrimenti null.
Invariante	Ogni operazione di scrittura deve mantenere l'integrità referenziale (es. non duplicare chiavi primarie).

Nome classe	PlayerDAO
Descrizione	Gestisce l'accesso alla tabella calciatori. Permette la lettura della lista e l'aggiornamento dei dati.
Signature dei metodi	+ doSave(player: Player): void + doRetrieveAll(): List<Player> + doRetrieveByRole(role: enum): List<Player> + doUpdate(player: Player): void

Pre-condizioni	**doSave**: Il calciatore deve avere un ruolo valido (P, D, C, A) associato.
Post-condizioni	**doRetrieveAll**: Restituisce la lista completa dei giocatori presenti nel listone corrente.
Invariante	-

Nome classe	**StatisticsDAO**
Descrizione	Gestisce l'accesso alla tabella statistiche (voti, bonus, malus). Usato per storicizzare i dati importati.
Signature dei metodi	+ doSave(stat: Statistic): void + doRetrieveByPlayer(playerID: int): List<Statistic> + doRetrieveByDay(day: int): List<Statistic>
Pre-condizioni	**doSave**: La statistica deve riferirsi a un giocatore esistente.
Post-condizioni	-
Invariante	I valori dei voti devono essere nel range numerico valido (es. 1-10 o valori speciali).

Nome classe	**SquadDAO**
Descrizione	Gestisce la relazione molti-a-molti tra Utenti e Calciatori (tabella rosa).
Signature dei metodi	+ addPlayerToSquad(userID: int, playerID: int): void

	+ removePlayerFromSquad(userID: int, playerID: int): void +doRetrieveSquadByUserId(userID: int): Squad
Pre-condizioni	**addPlayerToSquad**: L'utente e il calciatore devono esistere nelle rispettive tabelle anagrafiche.
Post-condizioni	**doRetrieveSquadByUserId**: Restituisce un oggetto Squad contenente la lista dei giocatori posseduti dall'utente.
Invariante	Un utente non può avere lo stesso calciatore più di una volta nella propria rosa.

Nome classe	**FormationDAO**
Descrizione	Gestisce la persistenza delle formazioni schierate dagli utenti per ogni giornata.
Signature dei metodi	+ doSave(formation: Formation): void + doRetrieveByDay(userID: int, day: int): Formation + doUpdate(formation: Formation): void
Pre-condizioni	**doSave**: La formazione deve contenere esattamente 11 titolari e i panchinari previsti.
Post-condizioni	**doRetrieveByDay**: Restituisce la formazione salvata per quella specifica giornata di campionato.
Invariante	Una formazione è univoca per la

	coppia (utente, giornata).

Nome classe	PostDAO
Descrizione	Gestisce le operazioni CRUD sulla tabella “Post” della community.
Signature dei metodi	+ doSave(post: Post): void + doRetrieveAll(): List<Post> + doDelete(postID: int): void
Pre-condizioni	**doSave**: Il post deve essere associato a un utente autore valido.
Post-condizioni	**doRetrieveAll**: Restituisce i post ordinati per data di pubblicazione (dal più recente).
Invariante	-

Nome classe	CommentDAO
Descrizione	Gestisce la persistenza dei commenti sotto i post.
Signature dei metodi	+ doSave(comment: Comment): void + doRetrieveByPost(postID: int): List<Comment>
Pre-condizioni	Il contenuto del commento non deve essere vuoto.
Post-condizioni	**doRetrieveByPost**: Restituisce tutti i commenti di un post.
Invariante	Un commento, se il post padre viene eliminato, deve essere gestito (es. cancellazione a

	cascata).

Nome classe	ReactionDAO
Descrizione	Gestisce le reazioni ai post.
Signature dei metodi	+ doSave(reaction: Reaction): void + doDelete(userID: int, postID: int): void + countReactions(postID: int): int
Pre-condizioni	-
Post-condizioni	**countReactions**: Restituisce il numero totale di reazioni per un determinato post.
Invariante	Un commento, se il post padre viene eliminato, deve essere gestito (es. cancellazione a cascata)

3.3 Utils

Questa sezione tratta di alcune utilità.

Nome classe	PasswordHasher
Descrizione	Classe di utilità che fornisce algoritmi crittografici (es. SHA-256) per l'hashing sicuro delle password prima della persistenza nel database.
Signature dei metodi	+ hash(password: String): String + verify(password: String, hashedPassword: String): boolean
Pre-condizioni	**hash**: La stringa di input non deve essere nulla o vuota.

Post-condizioni	**hash**: Restituisce una stringa esadecimale di lunghezza fissa che non può essere riconvertita nel testo originale. **verify**: Restituisce true se l'hash della password fornita coincide con quello memorizzato.
Invariante	L'algoritmo di hashing utilizzato deve essere deterministico (lo stesso input produce sempre lo stesso output).

Nome classe	**EmailUtils**
Descrizione	Fornisce metodi statici per la validazione sintattica degli indirizzi email e per l'hashing degli stessi (ove necessario per privacy o log anonimizzati).
Signature dei metodi	+ isValid(email: String): boolean + normalize(email: String): String + hashEmail(email: String): String
Pre-condizioni	**isValid**: La stringa deve contenere caratteri alfanumerici.
Post-condizioni	**isValid**: Restituisce true solo se l'e-mail rispetta il pattern standard RFC 5322 (es. user@domain.com). **normalize**: Rimuove spazi bianchi e converte in minuscolo per garantire unicità nel DB.
Invariante	-

3.4 Model

Questa sezione tratta delle entità persistenti del sistema.

Nome classe	User
Descrizione	Rappresenta l'utente registrato nel sistema (Fantallenatore o Gestore).
Signature dei metodi	+ User(email: String, username: String, password: String, role: Role) + getEmail(): String + getUsername(): String + checkPassword(input: String): boolean + getRole(): Role
Pre-condizioni	-
Post-condizioni	**checkPassword**: Restituisce true solo se l'hash calcolato corrisponde a quello memorizzato.
Invariante	L'attributo “**email**” deve essere un indirizzo sintatticamente valido. L'attributo “**role**” non può essere nullo.

Nome classe	Player
Descrizione	Rappresenta il singolo calciatore di Serie A presente nel database.
Signature dei metodi	+ Player(id: int, nome: String, squadraSerieA: String, ruolo: String) + getRuolo(): String
Pre-condizioni	-
Post-condizioni	-

Invariante	Il ruolo deve essere uno tra: "P" (Portiere), "D" (Difensore), "C" (Centrocampista), "A" (Attaccante).

Nome classe	**Squad**
Descrizione	Rappresenta la "Rosa" virtuale creata da un utente, contenente la lista dei calciatori.
Signature dei metodi	+ getPlayers(): List<Player> + isComplete(): boolean
Pre-condizioni	-
Post-condizioni	**isComplete**: Restituisce true se la rosa contiene esattamente 25 giocatori distribuiti correttamente.
Invariante	Una rosa valida non può mai eccedere i limiti: 3 Portieri, 8 Difensori, 8 Centrocampisti, 6 Attaccanti.

Nome classe	**Formation**
Descrizione	Rappresenta la formazione titolare schierata per una specifica giornata.
Signature dei metodi	+ Formation(giornata: int, modulo: Module, titolari: List<Player>, panchina: List<Player>)
Pre-condizioni	La somma dei giocatori (titolari + panchina) non deve superare il limite consentito.
Post-condizioni	-

Invariante	I titolari devono essere esattamente 11. La disposizione dei ruoli dei titolari deve coincidere con il modulo selezionato.

Nome classe	**Module**
Descrizione	Definisce lo schema tattico (es. 4-4-2).
Signature dei metodi	+ getName(): String + getDefendersCount(): int + getMidfieldersCount(): int + getStrikersCount(): int
Pre-condizioni	-
Post-condizioni	-
Invariante	La somma dei difensori, centrocampisti e attaccanti deve essere sempre uguale a 10 (il portiere è implicito).

Nome classe	**Statistics**
Descrizione	Contiene i dati di performance di un giocatore per una specifica giornata di campionato.
Signature dei metodi	+ Statistic(playerID: int, giornata: int, voto: float, bonus: float, malus: float) + getFantavoto(): float
Pre-condizioni	"**voto**" deve essere nel range 0-10 (o 6 politico).

Post-condizioni	**getFantavoto**: Restituisce la somma algebrica: voto + bonus - malus.
Invariante	Una statistica è univoca per la coppia (PlayerID, Giornata).

Nome classe	**Post**
Descrizione	Rappresenta un contenuto pubblicato da un utente nella sezione community.
Signature dei metodi	+ Post(author: User, text: String, attachment: Formation) + getTimestamp(): DateTime + getReactionCount(): int
Pre-condizioni	Il testo o l'allegato non possono essere entrambi nulli.
Post-condizioni	-
Invariante	Un post deve essere sempre associato a un autore esistente.

Nome classe	**Comment**
Descrizione	Rappresenta un commento di risposta a un post.
Signature dei metodi	+ Comment(author: User, postID: int, text: String) + getText(): String
Pre-condizioni	Il contenuto del commento non deve essere vuoto.
Post-condizioni	-

Invariante	Un commento, se il post padre viene eliminato, deve essere gestito (es. cancellazione a cascata).

Nome classe	**Reaction**
Descrizione	Rappresenta un feedback lasciato su un post.
Signature dei metodi	+ Reaction(user: User, post: Post, type: Enum)
Pre-condizioni	-
Post-condizioni	-
Invariante	Un utente non può duplicare la stessa reazione sullo stesso post.

4. GLOSSARY