

Università degli Studi di Salerno
Corso di Ingegneria del Software

FantaUnisa
Object Design Document
Versione 1.0



Data: 21/01/2026

| | |
|-----------------------------------|------------------|
| Progetto: FantaUnisa | Versione: 1.0 |
| Documento: Object Design Document | Data: 21/01/2026 |

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 |
| 17/01/2026 | 0.5 | Stesura iniziale Introduzione | Sabetta Francesco |
| 18/01/2026 | 0.6 | Aggiunti Trade-offs | Sabetta Francesco |
| 18/01/2026 | 0.7 | Stesura sezione Packages | Sabetta Francesco |
| 19/01/2026 | 0.8 | Riformulazione sezione Class Interfaces | Sabetta Francesco |
| 20/01/2026 | 0.9 | Stesura finale sezione Class Interfaces | Sabetta Francesco |
| 21/01/2026 | 1.0 | Revisione finale | Tutti i partecipanti |

Indice

| | |
|---|----------|
| 1. INTRODUZIONE..... | 4 |
| 1.1 Purpose..... | 4 |
| 1.2 Definitions, acronyms, and abbreviations..... | 4 |
| 1.3 References..... | 4 |
| 1.4 Trade-offs..... | 5 |
| 2. PACKAGES..... | 7 |
| 2.1 Design Pattern | 8 |
| 3. CLASS INTERFACES | 9 |
| 3.1 Subsystem: Access & Profile | 9 |
| 3.2 Subsystem: Team Management | 19 |
| 3.3 Subsystem: Community..... | 23 |
| 3.4 Subsystem: Module Selection | 30 |
| 3.5 Subsystem: Statistics Import..... | 32 |
| 3.6 Subsystem: Statistics Viewer..... | 33 |
| 3.7 Connection | 34 |
| 3.8 Initializer..... | 35 |
| 3.9 Utils | 36 |
| 3.10 Observer Pattern..... | 39 |

1. INTRODUZIONE

1.1 Purpose

Il presente **Object Design Document (ODD)** descrive in dettaglio la progettazione a oggetti del sistema **FantaUnisa**. Lo scopo del documento è definire l'architettura logica finale del software, specificando le interfacce delle classi, le relazioni tra di esse, i pattern utilizzati e le scelte implementative adottate per soddisfare i requisiti funzionali e non funzionali descritti nel **Requirement Analysis Document (RAD)** e l'architettura definita nel **System Design Document (SDD)**.

1.2 Definitions, acronyms, and abbreviations

| NOME | TIPO | DESCRIZIONE |
|-------------|-------|-------------------------------|
| ODD | Abbr. | Object Design Document |
| RAD | Abbr. | Requirement Analysis Document |
| SDD | Abbr. | System Design Document |
| MVC | Abbr. | Model-View-Controller |
| DAO | Abbr. | Data Access Object |
| JSP | Abbr. | JavaServer Pages |
| DB | Abbr. | Database |
| JDBC | Abbr. | Java Database Connectivity |

1.3 References

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

1.4 Trade-offs

Durante la fase di **Object Design**, sono state affrontate diverse decisioni progettuali per **bilanciare i requisiti di coesione, accoppiamento, performance e sicurezza**. Di seguito sono analizzati i principali compromessi adottati.

1.4.1 Reattività vs Affidabilità

- **Contesto:** Il sistema invia notifiche e-mail critiche per la registrazione e la sicurezza (reset password).
- **Conflitto:** L'invio sincrono garantisce che l'e-mail sia partita prima di confermare l'azione all'utente (Affidabilità), ma blocca l'interfaccia per diversi secondi. L'invio asincrono libera subito l'interfaccia (Reattività) ma introduce il rischio di perdita del messaggio in caso di crash immediato.
- **Decisione:** Si è scelto di privilegiare la **Reattività**.
- **Giustificazione:** Nella webapp **FantaUnisa**, la fluidità dell'interazione utente è prioritaria. Le operazioni di invio sono state delegate a processi background non persistenti. Il rischio marginale di mancato invio è stato mitigato fornendo meccanismi di "retry" manuale (es. "Invia di nuovo email di attivazione").

1.4.2 Prestazioni vs Manutenibilità

- **Contesto:** Il sistema deve gestire l'importazione massiva di statistiche (migliaia di record per giornata) da file esterni.
- **Conflitto:** L'uso di un approccio orientato agli oggetti puro (lettura, verifica esistenza, aggiornamento/inserimento a uno a uno) favorirebbe la leggibilità e la manutenibilità del codice, ma degraderebbe drasticamente le prestazioni.
- **Decisione:** Si è scelto di privilegiare le **Prestazioni**.
- **Giustificazione:** È stata implementata una logica di persistenza basata su operazioni *batch* e istruzioni SQL complesse (**Upsert**). Sebbene questo renda il livello di accesso ai dati (**DAO**) più complesso e meno manutenibile rispetto a un approccio ORM standard, riduce il tempo di elaborazione da minuti a secondi, requisito critico per l'usabilità del sistema durante l'aggiornamento dei voti.

1.4.3 Sicurezza vs Usabilità

- **Contesto:** Accesso alla piattaforma e recupero delle credenziali.
- **Conflitto:** L'implementazione di protocolli di sicurezza rigidi (attivazione obbligatoria via e-mail, hash irreversibili, logout automatico) aumenta la sicurezza ma crea frizione nel flusso utente (Usabilità).
- **Decisione:** Si è scelto di privilegiare la **Sicurezza**.
- **Giustificazione:** Trattandosi di un sistema che gestisce competizioni tra utenti e potenzialmente dati sensibili, l'integrità dell'account è prioritaria. È stato accettato un processo di registrazione a più step (Registrazione -> Attesa E-mail -> Click Link) per garantire l'autenticità degli utenti e prevenire la creazione di account falsi (bot), a discapito dell'immediatezza d'accesso.

1.4.4 Semplicità di Sviluppo vs Scalabilità

- **Contesto:** Gestione dello stato dell'utente (sessione) e della configurazione architetturale.
- **Conflitto:** Progettare un sistema stateless (es. basato su token) favorisce la scalabilità orizzontale su più server, ma aumenta la complessità di sviluppo. Un approccio stateful (sessione server-side) è più semplice ma vincola l'utente a un singolo server.
- **Decisione:** Si è scelto di privilegiare la **Semplicità di Sviluppo**.
- **Giustificazione:** Considerando il carico previsto e la natura didattica/monolitica del progetto, l'utilizzo della sessione server-side standard (**HttpSession**) ha permesso di ridurre i tempi di sviluppo e la complessità del codice Client-Server. La scalabilità verticale (potenziamento del singolo server) è stata ritenuta sufficiente per i requisiti attuali rispetto alla scalabilità orizzontale.

1.4.5 Realismo Funzionale vs Fattibilità Implementativa

- **Contesto:** Il requisito funzionale prevede un modulo di "**Intelligenza Artificiale**" capace di suggerire all'utente la formazione ideale, basandosi sull'analisi predittiva delle statistiche storiche dei calciatori.
- **Conflitto:** Lo sviluppo e l'addestramento di un vero modello di Machine Learning richiedono dataset massivi e tempi di sviluppo che eccedono i vincoli del progetto. Tuttavia, l'architettura deve dimostrare la presenza e il funzionamento del flusso di generazione automatica durante la dimostrazione.
- **Decisione:** Si è scelto di privilegiare la **Fattibilità Implementativa**.

- **Giustificazione:** Per garantire una demo funzionante entro la scadenza, **la componente IA è stato sostituito da un algoritmo stocastico (Random/Stub)**. Sebbene questo riduca il **Realismo** dei suggerimenti (la formazione è valida per ruolo ma casuale nei nomi), permette di validare l'integrazione architetturale del sottosistema **module_selection** senza bloccare il rilascio del software.

2. PACKAGES

Il sistema **FantaUnisa** è ingegnerizzato come progetto **Maven**. La gestione delle dipendenze e del ciclo di vita della build è definita nel file **pom.xml** alla radice del progetto. Il codice sorgente Java risiede nella cartella standard **src/main/java**. L'organizzazione interna segue il principio **Package-by-Feature**, che raggruppa le classi per funzionalità di dominio, suddividendole internamente secondo il pattern **MVC**.

Di seguito è riportata l'organizzazione fisica dei package nel file system:

- **src/main/java/connection**
 - Contiene la gestione centralizzata della connessione al database.
 - *Classi:* DBConnection.
- **src/main/java/initializer**
 - Contiene le logiche di bootstrap eseguite all'avvio del server (ServletContextListener).
 - *Classi:* AppInitializer, DBPopulator.
- **src/main/java/utils**
 - Librerie di utilità trasversali condivise tra i vari sottosistemi.
 - *Classi:* EmailSender (invio SMTP), CsvParser (lettura file CSV), PasswordHasher (sicurezza), RandomFormation (algoritmo stub), NavigationUtils, ReactionUtils.
- **src/main/java/observer_pattern**
 - Componenti per la gestione asincrona degli eventi di sistema.
 - *Classi:* Subject (Interfaccia), PasswordChangeObserverInterface, LogObserver, SecurityEmailObserver.
- **src/main/java/subsystems** (Root dei moduli funzionali)
 - **src/main/java/subsystems/access_profile**
 - **/control:** ForgotPasswordServlet, ChangePasswordServlet, ResetPasswordServlet, LoginServlet, RegisterServlet, ProfileServlet, ActivationServlet, DeleteUserServlet, LogoutServlet, UpdateProfileServlet.

- **/model:** User, UserDao, Role (Enum).
- **/service:** ChangePasswordService (Concrete Subject Observer Pattern).
- **src/main/java/subsystems/team_management**
 - **/control:** SquadServlet, FormationServlet, PlayerServlet.
 - **/model:** Player, PlayerDAO, Squad, SquadDAO, Formation, FormationDAO.
- **src/main/java/subsystems/community**
 - **/control:** PostServlet, CommentServlet, ReactionServlet, ReportServlet, DeleteContentServlet.
 - **/model:** Post, PostDAO, Comment, CommentDAO, Reaction, ReactionDAO, Report, ReportDAO.
- **src/main/java/subsystems/module_selection**
 - **/control:** ModuleServlet (Formato JSON).
 - **/model:** Module.
- **src/main/java/subsystems/statistics_import**
 - **/control:** StatisticsImportServlet.
 - **/model:** StatisticheDAO.
- **src/main/java/subsystems/statistics_viewer**
 - **/control:** LoadStatisticsServlet.
 - **/model:** Statistiche

2.1 Design Pattern

Il sistema utilizza un **Observer Pattern** per notificare componenti disaccoppiati (Log, SecurityEmail) quando avviene un evento critico (cambio password), così da avvisare un utente quando la sua password viene modificata.

PasswordChangeObserverInterface rappresenta il contratto a cui si sottoscrivono **LogObserver** e **SecurityEmailObserver** per essere notificati nel momento in cui avviene il cambio password, la quale avviene tramite **ChangePasswordService**, il quale rappresenta il Concrete Subject che implementa **Subject**.

3. CLASS INTERFACES

Qui sono descritte le interfacce delle singole classi.

Nota: I Getter, Setter e Costruttori verranno omessi per alleggerire la lettura ed evitare banalità. Lì dove saranno gli unici metodi (es. negli oggetti POJO), il campo **Signature dei metodi** sarà riempito con “-”.

Anche le classi non aventi attributi avranno il campo **Attributi** riempito con “-”. Infine, per le classi DAO verranno riportati solo alcuni metodi per evitare di sovraccaricare il documento di informazioni ridondanti (es. il metodo **doDelete** ha la stessa finalità per tutte le classi e si comporta allo stesso modo).

3.1 Subsystem: Access & Profile

Questo sottosistema permette all'utente di registrarsi, autenticarsi e di modificare il proprio profilo utente.

3.1.1 Model

| Nome classe | User |
|----------------------|---|
| Descrizione | Rappresenta l'utente registrato nel sistema. Mantiene le informazioni anagrafiche, le credenziali (hash) e lo stato di sicurezza. |
| Attributi | <ul style="list-style-type: none">- email: String- username: String- password: String- nome: String- cognome: String- role: Role- is_active: boolean- verificationToken: String- resetToken: String- resetExpiry: Date |
| Signature dei metodi | - |

| | |
|------------------------|---|
| Pre-condizioni | I campi e-mail, username e password non devono essere nulli o vuoti al momento della creazione. |
| Post-condizioni | getPassword() restituisce sempre la stringa hashata (SHA-256). |
| Invariante | Un utente deve avere sempre un Role assegnato (default: FANTALLENATORE). L'email deve essere univoca. |

| Nome classe | UserDAO |
|--------------------|--|
| Descrizione | Gestisce la persistenza dei dati utente, incluse le operazioni di autenticazione, attivazione account e recupero password. |
| Attributi | - |

| | |
|-----------------------------|--|
| Signature dei metodi | <ul style="list-style-type: none"> + doSave(User user): void + doRetrieveByEmailAndPassword(String email, String pwd): User + doRetrieveByEmail(String email): User + doActivate(String token): boolean + SetResetToken(String token): boolean + findByResetToken(String token): User + updatePassword(String email, String newPass): void <ul style="list-style-type: none"> + doUpdateInfo(User user): void + doDelete(String email): void + mapRowToUser(ResultSet rs): User |
| Pre-condizioni | <p>doRetrieveByEmailAndPassword: La password passata deve essere già hashata.</p> <p>doSave: L'oggetto User deve avere tutti i campi obbligatori popolati.</p> |
| Post-condizioni | <p>doActivate: Se restituisce true, il campo is_active dell'utente nel DB è impostato a 1.</p> <p>doDelete: L'utente e tutti i dati a cascata (rosa, formazioni) vengono rimossi.</p> |
| Invariante | <p>La connessione al database viene sempre chiusa dopo l'operazione (pattern try-with-resources).</p> |

| Nome classe | Role |
|----------------------|--|
| Descrizione | Enumerazione che definisce i livelli di autorizzazione e accesso degli utenti nel sistema. |
| Attributi | <ul style="list-style-type: none"> + FANTALLENATORE + GESTORE_UTENTI + GESTORE_DATI |
| Signature dei metodi | <ul style="list-style-type: none"> + values(): Role[] + valueOf(String name): Role |
| Pre-condizioni | - |
| Post-condizioni | - |

| | |
|-------------------|--|
| Invariante | I ruoli sono costanti e immutabili durante l'esecuzione. |
|-------------------|--|

3.1.2 Control

| Nome classe | | UpdateProfileServlet |
|-----------------------------|---|-----------------------------|
| Descrizione | Gestisce specificamente la modifica dei dati del profilo utente. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| 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 | | RegisterServlet |
|----------------------|---|------------------------|
| Descrizione | Gestisce la creazione account. Genera token di verifica e delega l'invio e-mail a un thread separato. | |
| Attributi | - | |
| Signature dei | # doPost(HttpServletRequest, | |

| | |
|------------------------|---|
| metodi | <pre>HttpServletResponse): void</pre> <pre># doGet(HttpServletRequest, HttpServletResponse): void</pre> |
| Pre-condizioni | L'email non deve essere già presente nel DB. |
| Post-condizioni | Utente creato con stato non attivo. Token generato. |
| Invariante | Tutti gli utenti creati avranno ruolo FANTALENATORE |

| Nome classe | LoginServlet |
|-----------------------------|--|
| Descrizione | Gestisce l'autenticazione. Verifica credenziali, hash password, stato attivazione e reindirizza in base al ruolo. |
| Attributi | - |
| Signature dei metodi | <pre># doPost(HttpServletRequest, HttpServletResponse): void</pre> <pre># doGet(HttpServletRequest, HttpServletResponse): void</pre> |
| Pre-condizioni | E-mail e password non nulli. |
| Post-condizioni | Se successo: Sessione creata e attributo "user" settato. Se errore: Forward a login.jsp. |
| Invariante | Solo utenti attivi (is_active=true) possono completare il login. |

| Nome classe | LogoutServlet |
|-----------------------------|---|
| Descrizione | Gestisce la disconnessione dell'utente. |
| Attributi | - |
| Signature dei metodi | <pre># doPost(HttpServletRequest, HttpServletResponse): void</pre> |

| | |
|------------------------|---|
| | # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | L'utente deve essere loggato nel sistema |
| Post-condizioni | La sessione HTTP viene invalidata |
| Invariante | - |

| Nome classe | | ActivationServlet |
|-----------------------------|---|--------------------------|
| Descrizione | Gestisce l'attivazione dell'account utente tramite il link inviato via e-mail. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | Il parametro token nella URL non deve essere nullo. | |
| Post-condizioni | Se il token è valido, il flag is_active dell'utente nel DB viene impostato a true . | |
| Invariante | Un token può essere utilizzato per attivare un solo account. | |

| Nome classe | | DeleteUserServlet |
|-----------------------------|---|--------------------------|
| Descrizione | Gestisce l'eliminazione di un account. Implementa controlli RBAC (Self-delete o Admin-delete). | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |

| | |
|------------------------|---|
| Pre-condizioni | L'utente richiedente deve essere il proprietario dell'account o avere ruolo GESTORE_UTENTI . |
| Post-condizioni | L'utente e tutti i dati correlati (rosa, formazioni) vengono rimossi dal DB. Se auto-cancellazione, la sessione viene invalidata. |
| Invariante | - |

| Nome classe | | ChangePasswordServlet |
|-----------------------------|---|------------------------------|
| Descrizione | Gestisce il cambio password volontario dall'area privata. Utilizza ChangePasswordService per avvisare gli Observer. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | L'utente deve fornire la vecchia password corretta. La nuova password deve rispettare i criteri di lunghezza. | |
| Post-condizioni | La password viene aggiornata e viene inviata una email di notifica (Observer Pattern). | |
| Invariante | - | |

| Nome classe | | ProfileServlet |
|-----------------------------|--|-----------------------|
| Descrizione | Prepara i dati per la visualizzazione della pagina profilo, recuperando le informazioni aggiornate dal DB. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void | |

| | |
|------------------------|--|
| | # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | Utente loggato in sessione. |
| Post-condizioni | L'attributo userProfile viene aggiunto alla request. |
| Invariante | - |

| Nome classe | | ForgotPasswordServlet |
|-----------------------------|---|---|
| Descrizione | | Avvia la procedura di recupero password generando un token di reset e inviando una email. |
| Attributi | | - |
| Signature dei metodi | | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | L'e-mail fornita deve esistere nel database. | |
| Post-condizioni | Viene salvato un resetToken nel record utente e avviato un thread per l'invio dell'e-mail. | |
| Invariante | Il token diventa invalido allo scadere del tempo. | |

| Nome classe | | ResetPasswordServlet |
|-----------------------------|--|---|
| Descrizione | | Gestisce l'impostazione della nuova password dopo aver cliccato sul link di recupero. |
| Attributi | | - |
| Signature dei metodi | | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |

| | |
|------------------------|--|
| Pre-condizioni | Il token di reset deve essere valido e presente nel DB. |
| Post-condizioni | La vecchia password viene sovrascritta con l'hash della nuova. |
| Invariante | Se il token è scaduto, la procedura fallisce. |

3.1.3 Service

| Nome classe | | ChangePasswordService |
|-----------------------------|---|-----------------------|
| Descrizione | (Concrete Subject) Gestisce la logica di cambio password e notifica gli osservatori registrati. Implementato come Singleton. | |
| Attributi | <ul style="list-style-type: none"> - instance: ChangePasswordService - observers: List<Observer> | |
| Signature dei metodi | <ul style="list-style-type: none"> + static getInstance(): ChangePasswordService + attach (Observer o): void + detach (Observer o): void + updatePassword(User u, String newPass): void + notifyObservers(User u, String eventType): void | |
| Pre-condizioni | La nuova password deve rispettare i requisiti di sicurezza. | |
| Post-condizioni | La password dell'utente viene aggiornata nel DB e tutti gli observer vengono notificati. | |
| Invariante | Esiste una sola istanza del servizio in memoria. | |

3.2 Subsystem: Team Management

Questo sottosistema permette la gestione della propria formazione.

3.2.1 Model

| Nome classe | Player |
|----------------------|---|
| Attributi | <ul style="list-style-type: none"> - id: int - nome: String - squadra: String - ruolo: String - mediaVoto: float - fantamedia: float - golFatti: int - golSubiti: int - assist: int |
| Signature dei metodi | - |
| Pre-condizioni | - |
| Post-condizioni | - |
| Invariante | Il ruolo deve essere uno tra "P", "D", "C", "A". |

| Nome classe | Squad |
|-------------|--|
| Attributi | <ul style="list-style-type: none"> - userEmail: String |
| Desrizione | Rappresenta la rosa dei calciatori di un fantallenatore. |

| | |
|-----------------------------|--|
| | <ul style="list-style-type: none"> - players: List<Player> |
| Signature dei metodi | <ul style="list-style-type: none"> + addPlayer(Player p): void + size(): int + isComplete(): boolean + containsPlayer(int id): boolean |
| Pre-condizioni | - |
| Post-condizioni | isComplete() restituisce true solo se la lista contiene esattamente 25 giocatori. |
| Invariante | Una rosa non può contenere giocatori duplicati. |

| Nome classe | Formation |
|-----------------------------|--|
| Descrizione | Rappresenta la formazione schierata per una specifica giornata, includendo titolari e panchinari. |
| Attributi | <ul style="list-style-type: none"> - id: int - userEmail: String - giornata: int - modulo: String - playersMap: Map<Integer, String> |
| Signature dei metodi | <ul style="list-style-type: none"> + addPlayer(int id, String tipo): void + getModulo(): String |
| Pre-condizioni | Il modulo deve essere una stringa valida (es. "3-4-3"). |
| Post-condizioni | playersMap contiene la distinzione tra "TITOLARE" e "PANCHINA". |
| Invariante | La formazione deve essere associata a un utente esistente. |

| Nome classe | | PlayerDAO |
|-----------------------------|--|-----------|
| Descrizione | Gestisce la persistenza dei dati dei calciatori, permettendone l'aggiornamento massivo. | |
| Attributi | - | |
| Signature dei metodi | + doRetrieveAll(): List<Player> + doRetrieveById(int id): List<Player> + doSaveOrUpdate(Connection con, Player p): void | |
| Pre-condizioni | doSaveOrUpdate: La connessione deve essere aperta e valida. | |
| Post-condizioni | doSaveOrUpdate: L'oggetto viene salvato/aggiornato correttamente all'interno del DB. | |
| Invariante | Ogni operazione di lettura deve chiudere correttamente il ResultSet e lo Statement per evitare memory leak. | |

| Nome classe | | SquadDAO |
|-----------------------------|--|----------|
| Descrizione | Gestisce le operazioni CRUD sulla rosa dell'utente, garantendo l'integrità dei dati tramite transazioni database. | |
| Attributi | - | |
| Signature dei metodi | + doRetrieveByEmail(String email): List<Player> + doUpdateSquad(String email, List<Integer> ids): void | |
| Pre-condizioni | doUpdateSquad: La lista di ID deve contenere 25 elementi validi. | |
| Post-condizioni | doUpdateSquad: Esegue una transazione atomica (Delete All + Insert Batch). In caso di errore esegue il Rollback lasciando la rosa invariata. | |
| Invariante | Un utente può avere una sola rosa attiva nel sistema. | |

| Nome classe | | FormationDAO |
|-----------------------------|---|--------------|
| Descrizione | Gestisce il salvataggio e il recupero delle formazioni. | |
| Attributi | - | |
| Signature dei metodi | + doSave(Formation f): int + doRetrieveDetailById(int id): Map<String, List<Player>> | |
| Pre-condizioni | doSave: L'oggetto Formation deve avere un modulo valido e una lista giocatori popolata. | |
| Post-condizioni | doSave: Restituisce l'ID generato dal database (Auto-Increment). doRetrieveDetailById: Restituisce una mappa con chiavi " TITOLARI " e " PANCHINA ". | |
| Invariante | - | |

3.2.2 Control

| Nome classe | | SquadServlet |
|-----------------------------|---|--------------|
| Descrizione | Gestisce il salvataggio della rosa. Verifica che siano selezionati esattamente 25 giocatori. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | L'array selectedPlayers deve contenere 25 ID univoci. | |
| Post-condizioni | La rosa viene aggiornata atomicamente. La cache LISTA_GIOCATORI_CACHE viene usata se disponibile. | |

| | |
|-------------------|---|
| Invariante | Un utente non può avere più o meno di 25 giocatori. |
|-------------------|---|

| Nome classe | FormationServlet |
|-----------------------------|---|
| Descrizione | Gestisce la creazione della formazione. Delega la logica di scelta dei titolari alla classe RandomFormation . |
| Attributi | - |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | L'utente deve avere una rosa completa. Il modulo scelto deve essere valido. |
| Post-condizioni | Una nuova formazione viene salvata nel DB e mostrata all'utente. |
| Invariante | - |

3.3 Subsystem: Community

Questo sottosistema consente l'interazione sociale tra utenti della piattaforma, attraverso la pubblicazione di formazioni e la possibilità di commentarle.

3.3.1 Model

| Nome classe | Post |
|--------------------|---|
| Descrizione | Rappresenta un contenuto in bacheca. Agisce come aggregatore di commenti e reazioni. |
| Attributi | <ul style="list-style-type: none"> - id: int - userEmail: String - testo: String - dataOra: Timestamp |

| | |
|-----------------------------|--|
| | <ul style="list-style-type: none"> - formationId: Integer - reactionCounts: Map<String, Integer> - comments: List<Comment> - currentUserReaction: String - formationDetails: Map<String, List<Player>> |
| Signature dei metodi | - |
| Pre-condizioni | testo non può essere nullo (ma può essere vuoto se c'è formationId). |
| Post-condizioni | getLikeCount() restituisce 0 se la mappa è vuota. |
| Invariante | Un post deve sempre avere un autore (userEmail) valido. |

| Nome classe | Comment |
|-----------------------------|--|
| Descrizione | Rappresenta una risposta testuale a un Post. Supporta l'allegato facoltativo di una formazione. |
| Attributi | <ul style="list-style-type: none"> - id: int - postId: int - userEmail: String - testo: String - dataOra: Timestamp - formationId: Integer |
| Signature dei metodi | - |
| Pre-condizioni | postId deve esistere. Se formationId è null, testo non può essere vuoto. |
| Post-condizioni | L'oggetto ha il timestamp di creazione settato |

| | |
|-------------------|--|
| | automaticamente. |
| Invariante | Un commento appartiene sempre a un solo post e a un solo autore. |

| Nome classe | Reaction |
|-----------------------------|---|
| Descrizione | Entità di associazione che rappresenta il "Mi Piace" o altre emozioni di un utente verso un post. |
| Attributi | <ul style="list-style-type: none"> - userEmail: String - postId: int - tipo: String |
| Signature dei metodi | - |
| Pre-condizioni | Il tipo deve essere una stringa non vuota. |
| Post-condizioni | La reazione viene assegnata/modificata/tolta dal post. |
| Invariante | La coppia (userEmail, postId) è unica nel DB. |

| Nome classe | Report |
|-----------------------------|--|
| Descrizione | Segnalazione di un contenuto inappropriato inviata agli amministratori. |
| Attributi | <ul style="list-style-type: none"> - id: int - userEmail: String - postId: int - motivo: String - dataOra: Timestamp |
| Signature dei metodi | - |

| | |
|------------------------|--|
| Pre-condizioni | motivo non vuoto. |
| Post-condizioni | La segnalazione è visibile nella dashboard del Gestore Utenti. |
| Invariante | Riferisce sempre a un post esistente. |

| Nome classe | PostDAO |
|-----------------------------|---|
| Descrizione | Gestisce la persistenza dei Post. Recupera anche i dati aggregati (es. ID formazione) ma delega i commenti ad altre query. |
| Attributi | - |
| Signature dei metodi | <ul style="list-style-type: none"> + doSave(Post p): void + doRetrieveAll(): List<Post> + doDelete(int id): void + doRetrieveById(int id): Post |
| Pre-condizioni | doSave: L'oggetto Post deve avere i campi obbligatori valorizzati. |
| Post-condizioni | doRetrieveAll: Restituisce i post ordinati per data decrescente. |
| Invariante | - |

| Nome classe | CommentDAO |
|-----------------------------|---|
| Descrizione | Gestisce CRUD sui commenti. |
| Attributi | - |
| Signature dei metodi | <ul style="list-style-type: none"> + doSave(Comment c): void + doRetrieveByPostId(int postId): List<Comment> + doDelete(int id): void |

| | |
|------------------------|--|
| Pre-condizioni | doDelete: l'id deve essere valido. |
| Post-condizioni | doRetrieveByPostId: Restituisce commenti in ordine cronologico. |
| Invariante | - |

| Nome classe | | ReactionDAO |
|-----------------------------|---|-------------|
| Descrizione | Gestisce le reazioni con logica ottimizzata per evitare duplicati. | |
| Attributi | - | |
| Signature dei metodi | + doSaveOrUpdate(Reaction r): void + doDelete(String email, int postId): void + doRetrieveUserReaction(String email, int postId): String | |
| Pre-condizioni | + doRetrieveUserReaction: L'e-mail e il postId devono essere presenti nel DB. | |
| Post-condizioni | doSaveOrUpdate: Esegue un UPSERT (Inserisce se nuovo, Aggiorna se esistente). | |
| Invariante | Non possono esistere due record per la stessa coppia utente-post. | |

| Nome classe | | ReportDAO |
|-----------------------------|--|-----------|
| Descrizione | Gestione persistenza delle segnalazioni. | |
| Attributi | - | |
| Signature dei metodi | + doSave(Report r): void + doRetrieveAll(): List<Report> + doDelete(int id): void | |
| Pre-condizioni | - | |

| | |
|------------------------|---|
| Post-condizioni | - |
| Invariante | - |

3.3.2 Control

| Nome classe | | PostServlet |
|-----------------------------|--|---|
| Descrizione | | Gestisce la bacheca Community. Aggrega post, commenti e conteggi reazioni prima di passare i dati alla View. |
| Attributi | | - |
| Signature dei metodi | | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | | Utente loggato. |
| Post-condizioni | | La request contiene la lista di Post arricchita con commenti e like. |
| Invariante | | - |

| Nome classe | | CommentServlet |
|-----------------------------|--|---|
| Descrizione | | Gestisce la pubblicazione di un commento a un post. |
| Attributi | | - |
| Signature dei metodi | | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | | Il commento deve avere testo oppure una |

| | |
|------------------------|---|
| | formazione allegata. postId valido. |
| Post-condizioni | Il commento viene salvato con il timestamp attuale. |
| Invariante | - |

| Nome classe | | ReactionServlet |
|-----------------------------|---|-----------------|
| Descrizione | Gestisce i "Mi Piace". Implementa la logica di <i>toggle</i> (aggiungi/rimuovi). | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | Parametri postId e tipo validi. | |
| Post-condizioni | Aggiorna lo stato della reazione e reindirizza usando l'ancora HTML (#postId) per mantenere lo scroll. | |
| Invariante | Un utente può mettere al massimo una reazione per post. | |

| Nome classe | | ReportServlet |
|-----------------------------|---|---------------|
| Descrizione | Gestisce le segnalazioni. doPost per creare segnalazione (Utente), doGet per visualizzare/gestire (Gestore Utenti). | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | doGet: Richiede ruolo GESTORE_UTENTI . | |

| | |
|------------------------|--|
| Post-condizioni | Le segnalazioni vengono salvate o rimosse dal sistema. |
| Invariante | - |

| Nome classe | | DeleteContentServlet |
|-----------------------------|---|-----------------------------|
| Descrizione | Gestisce l'eliminazione di Post o Commenti. Centralizza i controlli di sicurezza. | |
| Attributi | - | |
| Signature dei metodi | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void | |
| Pre-condizioni | Parametri id e type ("post" o "comment") validi. | |
| Post-condizioni | Il contenuto viene eliminato solo se l'utente è l'autore o Gestore Utenti. | |
| Invariante | - | |

3.4 Subsystem: Module Selection

Questo sottosistema permette la configurazione del modulo della propria formazione.

3.4.1 Model

| Nome classe | | Module |
|--------------------|--|---------------|
| Descrizione | Definisce lo schema tattico. | |
| Attributi | - id: String - difensori: int - centrocampisti: int | |

| | |
|-----------------------------|---|
| | - attaccanti: int |
| Signature dei metodi | + static getValidModules(): List<Module> |
| Pre-condizioni | - |
| Post-condizioni | Restituisce lista immutabile dei moduli. |
| Invariante | Somma ruoli (escluso portiere) = 10. |

3.4.2 Control

| Nome classe | | ModuleServlet |
|-----------------------------|--|---|
| Descrizione | | API REST che restituisce la lista dei moduli tattici validi in formato JSON. |
| Attributi | | - |
| Signature dei metodi | | # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |
| Pre-condizioni | | - |
| Post-condizioni | | Scrive una stringa JSON valida nello stream di risposta (application/json). |
| Invariante | | - |

3.5 Subsystem: Statistics Import

Questo sottosistema permette il caricamento e la verifica del file delle statistiche dei giocatori.

3.5.1 Model

| Nome classe | StatisticheDAO |
|-----------------------------|--|
| Descrizione | Gestisce i voti settimanali dei calciatori, supportando l'importazione massiva tramite logica Upsert. |
| Attributi | - |
| Signature dei metodi | <ul style="list-style-type: none"> + doSaveOrUpdate(Connection con, Statistiche s): void + findByPlayerAndRange(playerId, from, to): List<Statistiche> + findLastStatByPlayer(playerId): Statistiche |
| Pre-condizioni | doSaveOrUpdate: Connessione attiva. |
| Post-condizioni | doSaveOrUpdate: Garantisce idempotenza (no duplicati per stessa giornata). |
| Invariante | - |

3.5.2 Control

| Nome classe | StatisticsImportServlet |
|-----------------------------|---|
| Descrizione | Gestisce l'upload del file Excel/CSV e coordina l'importazione massiva dei voti. |
| Attributi | - |
| Signature dei metodi | <ul style="list-style-type: none"> # doPost(HttpServletRequest, HttpServletResponse): void # doGet(HttpServletRequest, HttpServletResponse): void |

| | |
|------------------------|---|
| Pre-condizioni | Il file deve rispettare il formato atteso dal CsvParser . Utente deve essere GESTORE_DATI . |
| Post-condizioni | Il DB viene aggiornato con i nuovi voti. La cache globale dei giocatori viene invalidata e ricaricata. |
| Invariante | L'operazione avviene in una singola transazione database. |

3.6 Subsystem: Statistics Viewer

Questo sottosistema permette la visualizzazione delle varie statistiche inerenti a giocatori e squadre.

3.6.1 Model

| Nome classe | Statistiche |
|-------------|--|
| Descrizione | Modello dati per le prestazioni settimanali di un calciatore |
| Attributi | <ul style="list-style-type: none"> - id: int - idCalciatore: int - giornata: int - partiteVoto: int - mediaVoto: double - fantaMedia: double - golFatti: int - golSubiti: int - rigoriParati: int - rigoriCalciati: int - rigoriSegnati: int |

| | |
|-----------------------------|---|
| | <ul style="list-style-type: none"> - rigoriSbagliati: int - assist: int - espulsioni: int - autogol: int - ammonizioni: int |
| Signature dei metodi | - |
| Pre-condizioni | - |
| Post-condizioni | - |
| Invariante | giornata deve essere positiva. |

3.6.2 Control

| Nome classe | LoadStatisticsServlet |
|-----------------------------|--|
| Descrizione | Gestisce la visualizzazione delle statistiche di un calciatore. |
| Attributi | - |
| Signature dei metodi | <pre># doPost(HttpServletRequest, HttpServletResponse): void</pre> <pre># doGet(HttpServletRequest, HttpServletResponse): void</pre> |
| Pre-condizioni | Parametro playerId obbligatorio. |
| Post-condizioni | Recupera l'ultima statistica singola per il riepilogo. |
| Invariante | - |

3.7 Connection

Tratta della connessione col Database.

| Nome classe | DBConnection |
|-------------|--------------|
|-------------|--------------|

| | |
|-----------------------------|--|
| Descrizione | Gestisce il pool di connessioni al database MySQL. |
| Attributi | - ds: DataSource |
| Signature dei metodi | + static getConnection(): Connection |
| 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. |
| Invariante | Le credenziali di accesso al DB sono configurate nel context.xml o staticamente. |

3.8 Initializer

Servono ad inizializzare e a preparare la webapp all'avvio.

| Nome classe | | AppInitializer |
|-----------------------------|---|----------------|
| Descrizione | (ServletContextListener) Classe di bootstrap che viene eseguita all'avvio del server Tomcat. Inizializza il DB e le risorse globali. | |
| Attributi | | - |
| Signature dei metodi | + contextInitialized(ServletContextEvent sce): void + contextDestroyed(ServletContextEvent sce): void | |
| Pre-condizioni | Il server deve aver caricato il driver JDBC correttamente. | |
| Post-condizioni | Al termine, il database è pronto e le cache globali sono inizializzate. | |

| | |
|-------------------|--|
| Invariante | Viene eseguito una sola volta per ciclo di vita dell'applicazione. |
|-------------------|--|

| Nome classe | DBPopulator |
|-----------------------------|---|
| Descrizione | Classe di utilità invocata dall'AppInitializer per popolare il database con dati di default (es. Gestori) se vuoto. |
| Attributi | - |
| Signature dei metodi | + static populate(): void |
| Pre-condizioni | Connessione al DB attiva. |
| Post-condizioni | Se il DB era vuoto, ora contiene i dati essenziali per il funzionamento. |
| Invariante | Non sovrascrive dati esistenti se il DB è già popolato. |

3.9 Utils

Questa sezione tratta di alcune classi di utilità adoperate all'interno del sistema.

| Nome classe | RandomFormation |
|-----------------------------|--|
| Descrizione | Algoritmo per generare una formazione casuale valida. |
| Attributi | - |
| Signature dei metodi | + static generateRandomLineup(Formation f, List<Player> rosa, Module m): void |
| Pre-condizioni | La rosa deve contenere abbastanza giocatori per coprire il modulo. |
| Post-condizioni | L'oggetto Formation viene popolato con 11 titolari. |
| Invariante | I giocatori scelti sono distinti (no duplicati). |

| Nome classe | EmailSender |
|-----------------------------|--|
| Descrizione | Utility per l'invio di email tramite protocollo SMTP (Gmail). Gestisce messaggi HTML per verifica account, reset password e avvisi di sicurezza. |
| Attributi | <ul style="list-style-type: none"> - HOST: String - PORT: String - USERNAME: String - PASSWORD: String |
| Signature dei metodi | <ul style="list-style-type: none"> + static sendVerificationEmail(to, token): void + static sendResetEmail(to, token): void + static sendSecurityAlert(to): void |
| Pre-condizioni | L'indirizzo email di destinazione deve essere sintatticamente valido. |
| Post-condizioni | Il metodo delega l'invio alla libreria Jakarta Mail (non garantisce la consegna, ma l'invio). |
| Invariante | Le credenziali SMTP non vengono modificate a runtime. |

| Nome classe | PasswordHasher |
|-----------------------------|--|
| Descrizione | Gestisce la crittografia delle password utilizzando l'algoritmo di hashing sicuro SHA-256. |
| Attributi | - |
| Signature dei metodi | <ul style="list-style-type: none"> + static hash(String password): String + static verify(String password, String hash): boolean |

| | |
|------------------------|---|
| Pre-condizioni | La password in input non deve essere nulla o vuota. |
| Post-condizioni | hash() restituisce sempre una stringa esadecimale di lunghezza fissa (64 caratteri). |
| Invariante | L'algoritmo è deterministico (stesso input = stesso output). |

| Nome classe | | CsvParser |
|-----------------------------|--|---|
| Descrizione | | Utility per il parsing di file CSV/Excel contenenti le statistiche. Converte le righe di testo in oggetti di dominio. |
| Attributi | | - |
| Signature dei metodi | | + parse(InputStream is, int giornata): List<ImportData> |
| Pre-condizioni | | Lo stream di input deve essere in formato UTF-8 e seguire la struttura posizionale prevista. |
| Post-condizioni | | Restituisce una lista di oggetti popolati. |
| Invariante | | - |

| Nome classe | | NavigationUtils |
|-----------------------------|--|--|
| Descrizione | | Implementa la logica di reindirizzamento centralizzata basata sul ruolo dell'utente. |
| Attributi | | - |
| Signature dei metodi | | + static redirectBasedOnRole(Role role, HttpServletResponse resp): void |
| Pre-condizioni | | L'oggetto response non deve essere già stato committato. |
| Post-condizioni | | L'utente viene reindirizzato alla pagina home |

| | |
|-------------------|---|
| | specificata per il suo livello di permessi. |
| Invariante | - |

| Nome classe | ReactionUtils |
|-----------------------------|--|
| Descrizione | Utility che esegue query aggregate per calcolare i totali delle reazioni (Like/Dislike) per un determinato post. |
| Attributi | - |
| Signature dei metodi | + static calculateReactionCounts(int postId): Map<String, Integer> |
| Pre-condizioni | - |
| Post-condizioni | Restituisce una Mappa contenente sempre le chiavi "LIKE" e "DISLIKE" (anche con valore 0). |
| Invariante | - |

3.10 Observer Pattern

Di seguito sono riportate le interfacce dei componenti dell'Observer pattern (il Concrete Subject **ChangePasswordService** è riportato nella sezione **service** di **Subsystem: Access & Profile**).

| Nome classe | PasswordChangeObserverInterface |
|-----------------------------|---|
| Descrizione | Interfaccia funzionale che deve essere implementata da qualsiasi classe voglia reagire all'evento di cambio password. |
| Attributi | - |
| Signature dei metodi | + onPasswordUpdate(User user, String eventType): void |
| Pre-condizioni | - |
| Post-condizioni | - |
| Invariante | - |

| Nome classe | | LogObserver |
|-----------------------------|--|---|
| Descrizione | | (Concrete Observer) Implementazione che registra l'evento di cambio password nei log di sistema (console o file) per scopi di auditing. |
| Attributi | | - |
| Signature dei metodi | | + onPasswordUpdate(User user, String eventType): void |
| Pre-condizioni | | L'oggetto user deve essere valido. |
| Post-condizioni | | Viene scritta una riga di log con timestamp e ID utente. |
| Invariante | | Non altera lo stato del sistema (sola lettura). |

| Nome classe | | SecurityEmailObserver |
|-----------------------------|--|--|
| Descrizione | | (Concrete Observer) Invia una e-mail di allerta all'utente quando viene rilevato un cambio password. |
| Attributi | | - |
| Signature dei metodi | | + onPasswordUpdate(User user, String eventType): void |
| Pre-condizioni | | L'oggetto user deve essere valido. |
| Post-condizioni | | Viene avviato un thread separato per l'invio della mail. |
| Invariante | | - |

| Nome classe | | Subject |
|--------------------|--|--|
| Descrizione | | Interfaccia che definisce il contratto per gli oggetti osservabili (Observable). Permette di |

| | |
|-----------------------------|--|
| | registrare e rimuovere osservatori. |
| Attributi | - |
| Signature dei metodi | <p style="text-align: center;">+</p> attach>PasswordChangeObserverInterface(o): void <p style="text-align: center;">+</p> detach>PasswordChangeObserverInterface(o): void <p style="text-align: center;">+ notifyObservers(User user, String eventType): void</p> |
| Pre-condizioni | L'osservatore passato non deve essere nullo. |
| Post-condizioni | La lista interna degli osservatori viene aggiornata. |
| Invariante | - |