Università degli Studi di Salerno

Corso di Ingegneria del Software

UStrike!

ODD

Versione 2.0

Data: 19/01/2026

Informazioni sul Progetto

Progetto: UStrike!
Documento: RAD
Versione: 2.0
Data: 19/01/2026

Coordinatore del progetto:
Nome
Matric
ola
Partecipanti:

Nome	Matricola
Russo Serena	0512119098
Stefanile Andrea	0512119557
Valito Marcello	0512119014

Scritto da:
Russo Serena, Stefanile Andrea, Valito Marcello

Revision History

Data	Versione	Descrizione	Autore
11/12/2025	0.1	Stesura iniziale del documento	Russo Serena, Stefanile Andrea, Valito Marcello
13/12/2025	0.2	Parte introduttiva -object design trade-offs	Stefanile Andrea
14/12/2025	0.3	Iniziale scrittura della class interface – Control, Util	Russo Serena, Valito Marcello
16/12/2025	0.4	Inserimento della sezione Model – DTO e definizione delle Back-end pages	Russo Serena, Valito Marcello
17/12/2025	0.5	Ultimazione della class interface – Service	Stefanile Andrea, Valito Marcello
18/12/2025	1.0	Revisione e correzione del documento	Russo Serena, Stefanile Andrea, Valito Marcello
15/01/2026	1.1	Modifica della sezione Control	Valito Marcello
18/01/2025	1.2	Modifica sezione Front-end	Russo Serena
19/01/2025	2.0	Revisione finale	Russo Serena

Sommario

1.Introduzione

Dopo la realizzazione del Requirement Analysis Document (RAD) e System Design Document (SDD), il presente documento descrive l'**Object Design** del sistema UStrike!, specificando le interfacce delle classi, le operazioni, i tipi, gli argomenti e le signature dei sottosistemi definiti nel SDD.

In particolare, questo documento definisce:

- Decomposizione in packages (back-end/front-end)
- Interfacce di tutte le classi con signature dei metodi
- Pre-condizioni, post-condizioni e invarianti
- Trade-offs di design
- Linee guida per naming convention e struttura del codice

L'obiettivo è fornire una base implementativa coerente con l'architettura MVC a 13 servlet.

2.Object Design Trade-offs

Comprensibilità vs Tempo: il codice deve essere leggibile e manutenibile per facilitare testing e future modifiche. Si utilizzano commenti mirati, naming convention chiari e struttura coerente. Questo comporta un incremento di tempo di sviluppo.

Sicurezza vs Efficienza: la sicurezza è prioritaria (crittografia delle password tramite l'algoritmo PBKDF2 con HmacSHA256., prepared statements, session tokens). Tale priorità comporterà overhead computazionale accettabile per il contesto.

Interfaccia vs Usabilità: l'interfaccia grafica è semplice, chiara e concisa. Flussi di prenotazione e operazioni staff ridotti a 6 passaggi max per ridurre errori utente.

Response time vs Hardware: obiettivo di risposta < 1.5 secondi garantito dalle operazioni di DB ottimizzate e architettura a connection pool. Vincolato dalle risorse della macchina di deploy.

Prestazioni vs Costi: utilizzo di fogli di stile CSS standard (no librerie a pagamento) e dipendenze open-source (JDBC nativo).

3.Linee guida per la documentazione delle interfacce

3.1 Naming Convention

Variabili**:** Iniziano con lettera minuscola, formato camelCase per nomi composti (idPrenotazione, nomeVariabile, statoPrenotazione). Nomi descrittivi, di uso comune, lunghezza medio-corta.

Metodi**:** Iniziano con lettera minuscola, formato camelCase. Nome tipicamente = verbo + oggetto: creaPrenotazione(), getPremi(), cancellaPrenotazione(). Getter/Setter: getNomeVariabile(), setNomeVariabile().

Classi e Pagine**:** Iniziano con lettera maiuscola, formato PascalCase: PrenotazioniServlet, PrenotazioneService, UserModel. Nome fornisce informazione del loro scopo.

3.2 Struttura file sorgente

Ogni file sorgente Java contiene una singola classe, strutturata come segue: (1) Dichiarazione package, (2) Istruzioni import (ordinate alfabeticamente), (3) Breve commento descrittivo della classe (JavaDoc), (4) Codice della classe (campi privati, costruttori, metodi pubblici/privati).

4.Definizioni, acronimi e abbreviazioni

Acronimo	Significato
RAD	Requirement Analysis Document
SDD	System Design Document
ODD	Object Design Document
MVC	Model-View-Controller
UC	Use Case
JDBC	Java Database Connectivity
DTO	Data Transfer Object
JS	JavaScript
ACL	Access Control List
Servlet	Controller HTTP che gestisce request/response
Action	Parametro dispatch per polimorfismo funzionale

5.Riferimenti

Problem Statement UStrike! V2.0 - 13/10/2025

RAD UStrike! V2.0(Russo S., Stefanile A., Valito M.) - 11/11/2025

SDD UStrike! v2.0 (Russo S., Stefanile A., Valito M.) - 19/01/2026

6.Packages

L'implementazione del back-end è organizzata nella directory src/main/java/.Il codice è strutturato all'interno del package com.ustrike, suddiviso nelle seguenti aree logiche:

- **Controller** (com.ustrike.control): contiene le Servlet che gestiscono le richieste e coordinano il flusso tra vista e modello
- **Model** (com.ustrike.model.): questo package gestisce la logica di business e l'accesso ai dati attraverso: com.ustrike.model.dao contenente le classi per l'interfacciamento al database, com.ustrike.model.dto contenente gli oggetti per il trasferimento dei dati, com.ustrike.model.service contenente le classi che contengono la logica applicativa principale
- **Filter** (com.ustrike.filter): contieni il filtro per la gestione degli accessi in base ai ruoli
- **Utility** (com.ustrike.util): package dedicato a classi di supporto come la gestione della connessione al database (DBConnection) e la sicurezza della password (PasswordHasher)

L'implementazione del front-end è organizzata nella directory /WebContent/ come segue

- **View (containers):** le JSP principali che definiscono la struttura delle pagine si trovano in / WebContent /view/jsp
- **Asset statici:** file di supporto sono suddivisi nelle sottocartelle di /WebContent/static, ovvero /css/ per gli stili delle pagine e /images/ per le risorse grafiche
- **Logica Client-side AJAX:** gli script per l'interattività sono situati in / WebContent /static/JavaScript/
- **Configurazione:** le informazioni di deployment e i metadati sono gestiti nelle directory protette WEB-INF e META-INF

7. Back-end Packages

7.1 Control

Classe	Descrizione
LoginServlet	Servlet di login e autenticazione
RegistrazioneServlet	Servlet di registrazione nuovi clienti
LogoutServlet	Servlet di chiusura sessione
RisorseDisponibiliServlet	CRUD risorse fisiche (piste/tavoli)
StaffDashboardServlet	Dashboard staff per accetta/rifiuta prenotazione
AnnullaPrenotazioneServlet	Annullamento prenotazione lato cliente
BiliardoServlet	Visualizzazione role-based della sezione biliardo (Staff/Cliente)
ClienteHomeServlet	Instradamento gli utenti autenticati alla dashboard principale del cliente
CreaPrenotazioneServlet	Gestione del ciclo di vita delle prenotazioni
MiePrenotazioniServlet	Recupero e visualizzazione dello storico delle prenotazioni effettuate dall'utente loggato.

7.2 Util

Classe	Descrizione
DBConnectionPool	Gestione connessioni JDBC (create/get/release)
PasswordHasher	Gestione della sicurezza hash della password

7.3 Filter

Classe	Descrizione
FiltroAccessiRuoli	Filtro autorizzazione (ACL e ruoli)

7.4 Model

7.4.1 DAO

Classe	Descrizione
ClienteDAO	Gestione operazioni CRUD entità cliente
PrenotazioneDAO	Gestione persistenza, recupero, aggiornamento degli stati delle prenotazioni
RisorsaDAO	Gestione delle risorse e verifica della disponibilità
ServizioDAO	Gestione della persistenza e stato di attivazione dei servizi
StaffDAO	Gestisce operazioni CRUD entità staff

7.4.2 DTO

Classe	Descrizione
Cliente	Rappresenta l'entità cliente come oggetto Java per il trasferimento dati tra i livelli
Prenotazione	Modella l'entità prenotazione, dettagli temporali, partecipanti
Risorsa	Rappresenta l'unità risorsa fisica prenotabile, ne definisce capacità, stato operativo e servizio a cui appartiene
Servizio	Modella le categorie di servizi offerti, ne definisce il nome e la disponibilità
Staff	Rappresenta l'entità staff come oggetto Java per il trasferimento dati tra i livelli
Premio	Rappresenta l'oggetto premio, ne definisce le caratteristiche come costo in ticket, disponibilità
Manager	Rappresenta l'entità manager come oggetto Java per il trasferimento dati tra i livelli
Turno	Assegnazione lavorativa del personale, collega staff, manager e fascia oraria

PrenotazioneView	Aggrega dati di prenotazione con quelli di servizio e risorse per la visualizzazione

7.4.3 Service

Classe	Descrizione
UserService	Operazioni su utenti + login/sessione + recupero password
PrenotazioneService	Business logic prenotazioni (stati, deadline)
RisorsaService	Disponibilità/occupazione risorse
ServizioService	Abilita/disabilita/lettura servizi

8. Front-end Packages

/WebContente/view/jsp (View Containers)

JSP	Descrizione
homeGenerale.jsp	Landing page pubblica
login.jsp	Form login
register.jsp	Form registrazione
clienteHome.jsp	Page cliente autenticato
biliardo.jsp	Page dedicata al servizio biliardo
catalogo-prenotazioni.jsp	Dashboard dedicata allo staff
mie-prenotazioni.jsp	Page cliente autenticato con lo storico delle sue prenotazioni
prenotazioni-form.jsp	Form per la prenotazione (servizio-based)

/WebContent/static/css - /WebContent/static/images (Asset statici)

CSS/images	Descrizione

biliardo.css	Foglio di stile esterno per pagina dedicata al servizio biliardo
clienteHome.css	Foglio di stile esterno per la pagina dedicata al cliente autenticato
dashboard.css	Foglio di stile esterno per la dashboard dedicata allo staff
homeGenerale.css	Foglio di stile esterno per la landing page pubblica
login.css	Foglio di stile esterno per il form di login
mie-prenotazioni.css	Foglio di stile esterno per la pagine del cliente autenticato con lo storico delle sue prenotazioni
prenotazioni.css	Foglio di stile esterno per il form per la prenotazione (servizio-based)
register.css	Foglio di stile esterno per il form di registrazione

/WebContent/static/JavaScript (Logica Client-side AJAX)

JS	Descrizione
annullaprenotazione.js	Dialog di conferma annullamento prenotazione per il cliente
dashboard.js	Comparsa di una tendina da cui scegliere il motivo di rifiuto della prenotazione da parte dello staff
prenotazioni.js	Comparsa di campi da compilare coi nomi basata sul numero di partecipanti inserito dal cliente

9.Class Interface

9.1 Control

LoginServlet

Descrizione: Servlet che gestisce l'accesso alla piattaforma UStrike!. Verifica credenziali email/password, crea sessione e reindirizza alla dashboard corretta in base al ruolo.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente non autenticato (o sessione scaduta); Parametri `email` e `password` presenti nel form; Connessione al database disponibile.

Post-condizioni: Se credenziali valide: sessione creata, redirect a dashboard se è staff, alla home se utente. Se credenziali non valide: messaggio di errore e redirect alla login.jsp. Password mai memorizzata/visualizzata in chiaro.

Invariante: Un utente non può avere sessioni duplicate simultanee (una per volta). Password archiviata solo come hash BCrypt.

RegistrazioneServlet

Descrizione: Servlet che gestisce la registrazione di nuovi clienti. Valida i dati, verifica l'univocità dell'account e crea un account con ruolo CLIENTE.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Parametri obbligatori `nome`, `cognome`, `email`, `password`, `confPassword` presenti; Email non già registrata; Password >= 8 caratteri, con obbligo di maiuscola, minuscola e numero; Connessione al database disponibile.

Post-condizioni: Utente creato nel DB con ruolo CLIENTE; Redirect a login.jsp con messaggio di successo; Password delegata allo UserService per l'hashing sicuro prima della scrittura sul DB.

Invariante: Il sistema impedisce la creazione di più utenze con la stessa email. Nessun utente può essere creato senza aver superato la validazione dei pattern definiti.

LogoutServlet

Descrizione: Gestisce la terminazione sicura della sessione utente.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Sessione esistente (oppure richiesta comunque valida).

Post-condizioni: Sessione invalidata e conseguente eliminazione di ogni attributo; Redirect a login.jsp; Nessun dato sensibile residuo in sessione.

Invariante: Logout sempre possibile, anche da stato inconsistente. Il logout è sempre garantito e non genera errori anche se richiamato a sessione già scaduta.

PrenotazioniServlet

Descrizione: Servlet che gestisce l'interfaccia e la logica di creazione delle nuove prenotazioni. Gestisce il pre-caricamento dei servizi e processa le richieste di prenotazione verificando la disponibilità in tempo reale delle risorse.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Autenticazione verificata con ruolo cliente; Parametri richiesti presenti (idServizio, data, orario, etc.); Connessione a database disponibile.

Post-condizioni: Prenotazione con record inserito nel DB con stato iniziale "In attesa". Response JSON con esito operazione. Se la prenotazione fallisce si ha la restituzione di un errore JSON senza interruzione del flusso lato client

Invariante: No overbooking: una risorsa non occupata due volte stesso slot. Solo gli utenti con ruolo "cliente" possono inizializzare una richiesta di prenotazione tramite questa servlet.

RisorseDisponibiliServlet

Descrizione: Servlet orientata al cliente che fornisce in tempo reale l'elenco delle risorse fisiche (piste, kart, ecc.) effettivamente libere per un determinato servizio in una specifica data e fascia oraria.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato con ruolo cliente; Parametri: idServizio, data, ora presenti nella richiesta; Connessione al DB disponibile

Post-condizioni: Restituzione di un oggetto JSON contenente tutte le risorse disponibili, filtrando quelle già occupate da altre prenotazioni confermate. Se fallisce viene restituito un errore JSON descrittivo

Invariante: Una risorsa viene mostrata come disponibile solo se non esiste una prenotazione attiva per lo stesso slot temporale.

StaffDashboardServlet

Descrizione: Servlet dedicata ai membri dello Staff per la gestione operativa delle prenotazioni. Permette di visualizzare le richieste filtrate per stato (In attesa, Completate, All) e di intervenire direttamente tramite l'accettazione o il rifiuto motivato delle stesse.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato come staff; Connessione al DB disponibile. Parametro opzionale filter per determinare le prenotazioni da mostrare

Post-condizioni: Inoltro a catalogo-prenotazioni.jsp con la lista delle prenotazioni da filtrate. Modifica dello stato della prenotazione sul DB; registrazione dell'ID dello staff che ha eseguito l'operazione e dell'eventuale nota di rifiuto. Risposta JSON per le operazioni di aggiornamento (success/error).

Invariante: Coerenza tra ruolo in sessione e contenuto visualizzato. Una prenotazione rifiutata deve obbligatoriamente contenere una motivazione (notaStaff).

MiePrenotazioniServlet

Descrizione: Servlet dedicata all'area personale del cliente. Recupera lo storico e lo stato attuale delle prenotazioni dell'utente loggato, fornendo una visualizzazione arricchita con i dettagli dei servizi e delle risorse.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato con ruolo cliente; Attributo userId presente in sessione; Connessione a database disponibile.

Post-condizioni: Caricamento della lista di oggetti PrenotazioneView come attributo della request; Inoltro (forward) a mie-prenotazioni.jsp per la visualizzazione dei dati.

Invariante: Un cliente può accedere esclusivamente alle prenotazioni associate al proprio ID univoco estratto dalla sessione. Dati sempre coerenti con lo stato attuale del database.

ClienteHomeServlet

Descrizione: Servlet che gestisce l'accesso e la visualizzazione della pagina principale (Dashboard) riservata ai clienti. Funge da punto di ingresso per le funzionalità di prenotazione e gestione profilo.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato con ruolo cliente (verificato dal sistema di filtraggio accessi); Sessione attiva contenente i dati anagrafici di base.

Post-condizioni: Renderizzazione della dashboard cliente tramite inoltro (forward) alla risorsa clienteHome.jsp; Visualizzazione dei collegamenti rapidi ai servizi Bowling e Go-Kart.

Invariante: L'accesso alla risorsa è garantito solo se il ruolo in sessione corrisponde a "cliente", impedendo l'accesso non autorizzato a staff e manager.

BiliardoServlet

Descrizione: Servlet che gestisce la visualizzazione della pagina informativa dedicata al servizio Biliardo. Determina dinamicamente l'indirizzo della "Home" in base al ruolo dell'utente.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Richiesta alla risorsa tramite metodo GET; Sessione esistente analizzata per recuperare il ruolo dell'utente.

Post-condizioni: Inoltro alla pagina biliardo.jsp per la visualizzazione dei contenuti.

Invariante: La navigazione deve sempre garantire un link coerente con lo stato della sessione, evitando che un utente loggato venga riportato alla landing page generale.

AnnullaPrenotazioneServlet

Descrizione: Servlet che gestisce la richiesta di annullamento di una prenotazione esistente da parte di un cliente.

Signature dei metodi:

doPos(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato con ruolo cliente; Parametro idPrenotazione valido e presente nella richiesta; Connessione al database disponibile.

Post-condizioni: Stato della prenotazione aggiornato ad "Annullata"; Inserimento di un messaggio flash in sessione per la notifica all'utente; Redirect verso la servlet MiePrenotazioniServlet per mostrare l'elenco aggiornato.

Invariante: L'operazione deve garantire l'integrità dei dati rimuovendo o aggiornando eventuali cache di sessione.

9.2 Util

PasswordHasher

Descrizione: Classe utility che implementa la crittografia delle password tramite l'algoritmo PBKDF2 con HmacSHA256.

Signature dei metodi:

static hash(String pass) : String
static verify(String pass, String hash) : boolean

Pre-condizioni: Stringa della password non nulla; password lunghezza >= 8.

Post-condizioni: Restituzione di una stringa composta da salt e hash codificati in esadecimale; Il metodo verify restituisce true solo se la password fornita produce il medesimo hash utilizzando il salt estratto.

Invariante: Le password in chiaro non vengono mai memorizzate né restituite dalla classe.

DBConnectionPool

Descrizione: Classe di utility che centralizza la configurazione JDBC e gestisce l'apertura e la chiusura delle connessioni verso il database MySQL.

Signature dei metodi:
static getConnection() : Connection
static closeConnections(onnection conn) : void

Pre-condizioni: MySQL server attivo.; Credenzialo di accesso corrette (root/password).

Post-condizioni: Connection pronta all'uso (open). Chiusura sicurad ella risorsa tramite il metodo di rilascio.

Invariante: Centralizzazione dei parametri di connessione per facilitare la manutenzione del sistema.

9.3 Filter

FiltroAccessiRuoli

Descrizione: Filtro di sicurezza centralizzato che intercetta le richieste dirette alle aree riservate. Verifica la presenza di una sessione attiva e valida il ruolo dell'utente (Cliente, Staff, Manager) garantendo che l'accesso ai percorsi sia limitato esclusivamente ai soggetti autorizzati.

Signature dei metodi:

doFilter(ServletRequest request, ServletResponse response, FilterChain chain) : void

Pre-condizioni: Attributo "ruolo" atteso nella sessione dell'utente; richiesta HTTP verso i pattern /cliente/*, /staff/* o /manager/*.

Post-condizioni: Se il ruolo dell'utente coincide con il prefisso del path richiesto, la richiesta prosegue nella catena (chain.doFilter); In caso di ruolo non autorizzato o sessione assente, la richiesta viene interrotta con reindirizzamento (redirect) alla pagina di login con parametro di errore.

Invariante: N nessun utente può accedere a risorse di una categoria di ruolo diversa dalla propria. I tentativi di accesso non autorizzato vengono tracciati nel log di sistema con timestamp.

9.4 Model

9.4.1 DAO

RisorsaDAO

Descrizione: Data Access Object (DAO) che gestisce tutte le operazioni CRUD (Create, Read, Update, Delete) e le interrogazioni specifiche sulla disponibilità fisica di piste e tavoli nel database.

Signature dei metodi:

insertRisorsa(int stato, int capacita, int idServizio) : int

selectRisorsa(int idRisorsa) : Risorsa

selectAllRisorse() : List<Risorsa>

selectRisorseByServizio(int idServizio) : List<Risorsa>

updateRisorsa(int idRisorsa, int stato, int capacita) : boolean

deleteRisorsa(int idRisorsa) : boolean

isDisponibile(int idRisorsa, Timestamp orario) : boolean

Pre-condizioni: IDServizio esistente nella tabella Servizio per le operazioni di inserimento; Oggetto Timestamp non nullo per il controllo disponibilità.

Post-condizioni: Restituzione di oggetti DTO (Risorsa) mappati correttamente o valori booleani per confermare l'esito delle operazioni di scrittura.

Invariante: Ogni risorsa nel sistema deve essere associata a un servizio valido. Il metodo isDisponibile garantisce la coerenza temporale escludendo dal conteggio le prenotazioni con stato 'Rifiutata' o 'Annullata'.

StaffDAO

Descrizione: Data Access Object (DAO) che gestisce la persistenza dei dati relativi ai dipendenti e al manager, occupandosi delle operazioni di autenticazione e della gestione dei profili autorizzativi.

Signature dei metodi:

doSave(Staff staff) : boolean

doRetrieveByKey(int idStaff) : Staff

doRetrieveByEmail(String email) : Staff

emailExists(String email) : boolean

doRetrieveAll() : List<Staff>

doRetrieveByRuolo(String ruolo) : List<Staff>

doUpdate(Staff staff) : boolean

updatePassword(int idStaff, String newPasswordHash) : boolean

doDelete(int idStaff) : boolean

Pre-condizioni: Oggetto Staff non nullo per le operazioni di salvataggio; Password già processata tramite PasswordHasher (formato PBKDF2).

Post-condizioni: Esecuzione di query SQL parametriche (PreparedStatement) per la manipolazione dei record nella tabella Staff; Restituzione di oggetti DTO mappati correttamente; Inserimento dell'ID autogenerato nell'oggetto Staff passato al metodo doSave.

Invariante: Il ruolo salvato deve corrispondere a uno dei valori ammessi dal sistema.

ServizioDAO

Descrizione: Data Access Object (DAO) che gestisce la persistenza delle tipologie di attività offerte dalla struttura (Bowling, Go-Kart, Biliardo), permettendo la loro attivazione o disattivazione logica all'interno del sistema.

Signature dei metodi:

doSave(Servizio servizio) : boolean

doRetrieveByKey(int idServizio) : Servizio

doRetrieveByNome(String nomeServizio) : Servizio

doRetrieveAll() : List<Servizio>

doRetrieveEnabled() : List<Servizio>

doUpdate(Servizio servizio) : boolean

abilitaServizio(int idServizio) : boolean

disabilitaServizio(int idServizio) : boolean

doDelete(int idServizio) : boolean

Pre-condizioni: Nome del servizio non nullo per le operazioni di salvataggio e ricerca.

Post-condizioni: Restituzione di oggetti DTO Servizio mappati correttamente; Gestione dello stato booleano (StatoServizio) per determinare la disponibilità del servizio all'utenza.

Invariante: Ogni servizio deve possedere un nome univoco. La disattivazione di un servizio (StatoServizio = 0) non elimina i dati storici ma ne impedisce la selezione per nuove prenotazioni.

PrenotazioneDAO

Descrizione: Data Access Object (DAO) che gestisce l'intero ciclo di vita delle prenotazioni. Include metodi per il salvataggio, il recupero filtrato (per cliente, in attesa, completate) e l'aggiornamento degli stati.

Signature dei metodi:

insertPrenotazione(...) : int

selectPrenotazione(int idPrenotazione) : Prenotazione

selectPrenotazioniByCliente(int idCliente) : List<Prenotazione>

selectPrenotazioniInAttesa() : List<Prenotazione>

updateStatoPrenotazione(int idPrenotazione, String nuovoStato, Integer idStaff, String notaStaff) : boolean

selectPrenotazioniByClienteView(int idCliente) : List<PrenotazioneView>

annullaPrenotazioneCliente(int idPrenotazione, int idCliente) : boolean

Pre-condizioni: Vincoli di integrità referenziale soddisfatti (IDCliente, IDServizio e IDRisorsa esistenti); Parametri temporali (Data/Orario) formattati come Timestamp.

Post-condizioni: Recupero di liste filtrate ordinate cronologicamente; Aggiornamento atomico dello stato della prenotazione.

Invariante: Coerenza dello stato: un annullamento o un'approvazione può avvenire solo se la prenotazione si trova nello stato 'In attesa'.

ClienteDAO

Descrizione: Data Access Object (DAO) che gestisce la persistenza dei profili utente, memorizzando le informazioni anagrafiche, le credenziali di accesso (hash)

Signature dei metodi:

selectClienteById(int idCliente) : Cliente

selectClienteByEmail(String email) : Cliente

selectAllClienti() : List<Cliente>

insertCliente(Cliente cliente) : boolean

updateCliente(Cliente cliente) : boolean

deleteCliente(int idCliente) : boolean

Pre-condizioni: Email univoca per ogni registrazione; Password preventivamente cifrata tramite PasswordHasher.

Post-condizioni: Creazione o aggiornamento dei record nella tabella Cliente; Restituzione di oggetti DTO Cliente mappati fedelmente.

Invariante: L'attributo Email funge da identificativo unico per l'autenticazione lato cliente.

9.4.2 DTO

Staff

Descrizione: Data Transfer Object (DTO) che rappresenta il personale della struttura (Staff). Incapsula i dati anagrafici, le credenziali di accesso sicure e le informazioni sul livello di autorizzazione necessario per il controllo degli accessi.

Signature dei metodi:

getIDStaff() : int

setIDStaff(int idStaff) : void

getNomeStaff() : String

setNomeStaff(String nomeStaff) : void

getCognomeStaff() : String

setCognomeStaff(String cognomeStaff) : void

getEmail() : String

setEmail(String email) : void

getPasswordHash() : String

setPasswordHash(String passwordHash) : void

getRuolo() : String

setRuolo(String ruolo) : void

getFullName() : String

Pre-condizioni: Classe serializzabile per la gestione dello stato nella sessione HTTP.

Post-condizioni: Creazione di un oggetto di trasporto dati che funge da contenitore dati tra il DAO e i Controller.

Invariante: L'attributo passwordHash non deve mai contenere la password in chiaro.

Turno

Descrizione: Data Transfer Object (DTO) che rappresenta l'entità Turno all'interno del sistema. Viene utilizzata per incapsulare i dati relativi alla pianificazione lavorativa dello staff, definendo l'associazione tra un dipendente, la data, la fascia oraria e il manager che ha approvato l'assegnazione.

Signature dei metodi:

getIDTurno() : int

setIDTurno(int idTurno) : void

getData() : Timestamp

setData(Timestamp data) : void

getFasciaOraria() : String

setFasciaOraria(String fasciaOraria) : void

getIDStaff() : int

setIDStaff(int idStaff) : void

getIDManager() : int

setIDManager(int idManager) : void

Pre-condizioni: Implementazione dell'interfaccia Serializable per permettere il passaggio dell'oggetto tra i diversi layer o la memorizzazione in sessione.

Post-condizioni: Creazione di un oggetto di trasporto dati che riflette fedelmente i vincoli della tabella Turno del database.

Invariante: L'ID del turno, se valorizzato, deve corrispondere alla chiave primaria univoca sul database.

Servizio

Descrizione: Data Transfer Object (DTO) che rappresenta un'attività o categoria di gioco offerta dalla struttura (es. Bowling, Go-Kart, Biliardo). Viene utilizzata per veicolare le informazioni sull'offerta commerciale e sul relativo stato di operatività tra i vari strati dell'applicazione.

Signature dei metodi:

getIDServizio() : int

setIDServizio(int idServizio) : void

getNomeServizio() : String

setNomeServizio(String nomeServizio) : void

getStatoServizio() : boolean

setStatoServizio(boolean statoServizio) : void

Pre-condizioni: Implementazione dell'interfaccia Serializable per permettere il passaggio dell'oggetto tra i diversi layer.

Post-condizioni: Creazione di un oggetto di trasporto dati che mappa le colonne della tabella Servizio presenti nel database.

Invariante: L'identificativo idServizio deve corrispondere alla chiave primaria univoca del record.

Risorsa

Descrizione: Data Transfer Object (DTO) che rappresenta un'unità fisica specifica legata a un servizio (ad esempio, una singola pista da bowling, un tavolo da biliardo o un go-kart). Viene utilizzata per gestire la capacità e lo stato operativo delle risorse necessarie per completare una prenotazione.

Signature dei metodi:

getIDRisorsa() : int

setIDRisorsa(int idRisorsa) : void

getStato() : int

setStato(int stato) : void

getCapacita() : int

setCapacita(int capacita) : void

getIDServizio() : int

setIDServizio(int idServizio) : void

toString() : String

Pre-condizioni: Implementazione dell'interfaccia Serializable per permettere il passaggio dell'oggetto tra i diversi layer.

Post-condizioni: Creazione di un oggetto DTO che incapsula le proprietà fisiche e lo stato della risorsa recuperata dal database.

Invariante: L'identificativo idRisorsa deve essere unico e positivo. L'attributo idServizio deve puntare a un servizio esistente per garantire che la risorsa sia logicamente classificata.

PrenotazioneView

Descrizione: Data Transfer Object (DTO) specializzata per la visualizzazione dei dati (View Object). Estende le informazioni della prenotazione base includendo dettagli descrittivi come il nome del servizio e la capacità della risorsa.

Signature dei metodi:

getIDPrenotazione() : int

setIDPrenotazione(int IDPrenotazione) : void

getNomeServizio() : String

setNomeServizio(String nomeServizio) : void

getCapacitaRisorsa() : int

setCapacitaRisorsa(int capacitaRisorsa) : void

getIDStaff() : Integer (Nullable)

getNoteStaff() : String (Nullable)

Pre-condizioni: Esecuzione di una query SQL con JOIN tra le tabelle Prenotazione, Servizio e Risorsa; Implementazione di Serializable per la gestione del passaggio dati tra Servlet e pagine JSP.

Post-condizioni: Creazione di un oggetto contenente un set informativo completo pronto per essere renderizzato nelle tabelle "Mie Prenotazioni" o "Gestione Staff".

Invariante: I campi relativi a IDStaff e NoteStaff possono essere nulli se la prenotazione non è ancora stata presa in carico o lavorata dallo staff.

Prenotazione

Descrizione: Data Transfer Object (DTO) che rappresenta l'entità Prenotazione. Incapsula i dati relativi alla richiesta di un cliente (data, ora, partecipanti) e i riferimenti alle chiavi esterne per il servizio e la risorsa allocata. Gestisce inoltre lo stato del workflow (es. "In attesa", "Confermata") e il feedback dello staff.

Signature dei metodi:

getIDPrenotazione() : int

setIDPrenotazione(int IDPrenotazione) : void

getStatoPrenotazione() : String

setStatoPrenotazione(String stato) : void

getIDStaff() : Integer (Nullable)

getNoteStaff() : String (Nullable)

Pre-condizioni: Implementazione di Serializable per la gestione del passaggio dati tra i vari layer.

Post-condizioni: Creazione di un oggetto di trasporto dati che faccia da ponte tra DAO e Service.

Invariante: L'attributo IDPrenotazione deve corrispondere alla chiave primaria univoca.

Premio

Descrizione: Data Transfer Object (DTO) che rappresenta un articolo del catalogo premi. Incapsula le informazioni relative al nome del premio, alla descrizione.

Signature dei metodi:

getIDPremio() : int

setIDPremio(int idPremio) : void

getNomePremio() : String

setNomePremio(String nome) : void

getDescrizione() : String

setDescrizione(String desc) : void

getCostoTicket() : int

setCostoTicket(int costo) : void

Pre-condizioni: Implementazione di Serializable per la gestione del passaggio dati tra i vari layer.

Post-condizioni: Creazione di un oggetto di trasporto dati che mappa la tabella Premio del database.

Invariante: L'identificativo idPremio deve essere unico. Il valore costoTicket deve essere un intero positivo, rappresentante la soglia minima di punti che un cliente deve possedere per richiedere il premio.

Manager

Descrizione: Data Transfer Object (DTO) che rappresenta l'amministratore del sistema. Rispetto allo staff operativo, il manager possiede privilegi estesi per la gestione del catalogo servizi, l'assegnazione dei turni e la configurazione delle risorse fisiche della struttura.

Signature dei metodi:

getIDManager() : int

setIDManager(int idManager) : void

getNomeManager() : String

setNomeManager(String nome) : void

getEmail() : String

setEmail(String email) : void

getPasswordHash() : String

setPasswordHash(String hash) : void

getRuoloManager() : String

getFullName() : String

Pre-condizioni: Implementazione di Serializable per la gestione del passaggio dati tra i vari layer.

Post-condizioni: Creazione di un oggetto di trasporto dati per la gestione del passaggio tra i diversi layer.

Invariante: : L'email deve essere univoca per garantire l'accesso al pannello di amministrazione. La password non viene mai gestita in chiaro, rispettando i criteri di sicurezza basati su hashing definiti nella classe PasswordHasher.

Cliente

Descrizione: Data Transfer Object (DTO) che rappresenta l'utente finale del sistema. Incapsula i dati anagrafici, le credenziali di accesso e il saldo del programma fedeltà. È l'entità principale attorno a cui ruota il sistema di prenotazioni e il catalogo premi.

Signature dei metodi:

getIDCliente() : int setIDCliente(int idCliente) : void

getNomeCliente() : String

setNomeCliente(String nome) : void

getEmail() : String

setEmail(String email) : void

getPuntiTicket() : int

setPuntiTicket(int punti) : void

getFullName() : String

Pre-condizioni: Implementazione di Serializable per la gestione del passaggio dati tra i vari layer; Password già cifrata tramite algoritmo di hashing.

Post-condizioni: Creazione di un oggetto di trasporto dati che mappa la tabella Cliente del database; Disponibilità del saldo punti per la logica di riscatto premi.

Invariante: L'identificativo idCliente deve essere univoco. Il campo email funge da username unico per l'autenticazione.

9.4.3 Service

UserService

Descrizione: Classe che centralizza la gestione degli utenti del sistema Strike. Funge da orchestratore tra il front-end (Servlet) e i layer di persistenza (ClienteDAO, StaffDAO), implementando procedure di sicurezza critiche come il hashing PBKDF2 delle password e la logica di autenticazione multi-ruolo.

Signature dei metodi:

authenticateUser(String email, String passwordPlain) : Object[]

createCliente(Cliente cliente, String passwordPlain) : boolean

createStaff(Staff staff, String passwordPlain) : boolean

updateUser(Object user, String ruolo) : boolean

changePassword(String email, String ruolo, String oldPlain, String newPlain) : boolean

getUserById(int id, String ruolo) : Object

Pre-condizioni: Connessione DB valida; User non nullo (per create/update); Email e password valide.

Post-condizioni: Operazione completata e DB coerente. Le password vengono persistite nel DB esclusivamente sotto forma di hash. Impedisce la creazione di account duplicati verificando l'esistenza dell'email prima dell'inserimento.

Invariante: L'unicità dell'email è garantita a livello di business logic prima della scrittura.

PrenotazioneService

Descrizione: Classe che coordina il ciclo di vita delle prenotazioni. Gestisce la creazione delle richieste, il workflow di approvazione/rifiuto dello staff e l'annullamento da parte del cliente, integrando un meccanismo di invalidazione della cache per garantire la coerenza dei dati in sessione.

Signature dei metodi:

getCatalogoInAttesa() : List<Prenotazione>

getPrenotazioniClienteView(int idCliente, HttpSession session) : List<PrenotazioneView>

creaPrenotazione(Timestamp data, Timestamp orario, String partecipanti, ..., HttpSession session) : int

accettaPrenotazione(int idPrenotazione, int idStaff, HttpSession session) : boolean

rifiutaPrenotazione(int idPrenotazione, int idStaff, String motivo, HttpSession session) : boolean

annullaPrenotazioneCliente(int idPrenotazione, int idCliente, HttpSession session) : boolean

Pre-condizioni: Connessione DB valida; Risorsa disponibile (per create); Staff autenticato (per accetta/rifiuta/assegna); Prenotazione nello stato "In attesa".

Post-condizioni: Rimozione automatica dei dati obsoleti dalla sessione utente; Connessione al DB disponibile.

Invariante: Una prenotazione può essere modificata nello stato solo se attualmente "In attesa".

RisorsaService

Descrizione: Classe che gestisce il ciclo di vita delle risorse fisiche (piste, kart) e la loro disponibilità.

Signature dei metodi:

creaRisorsa(int stato, int capacita, int idServizio) : int

aggiornaRisorsa(int idRisorsa, int nuovoStato, int nuovaCapacita) : boolean

aggiornaStatoRisorsa(int idRisorsa, int nuovoStato) : boolean

getRisorseLibereByServizio(int idServizio) : List<Risorsa>

isRisorsaDisponibile(int idRisorsa, Timestamp dataOra) : boolean

risorsaAppartieneAlServizio(int idRisorsa, int idServizio) : boolean

Pre-condizioni: idServizio esistente nel database.

Post-condizioni: Stato della risorsa persistito su DB.

Invariante: Risorsa sempre associata ad un servizio valido.

ServizioService

Descrizione: Classe dedicata alla gestione del catalogo dei servizi offerti.

Signature dei metodi:

getServiziAbilitati() : List<Servizio>

getTuttiIServizi() : List<Servizio>

getServizioById(int idServizio) : Servizio

getServizioByNome(String nomeServizio) : Servizio

abilitaServizio(int idServizio) : boolean

disabilitaServizio(int idServizio) : boolean

Pre-condizioni: Staff autenticato; Presenza dei 3 servizi nel database

Post-condizioni: Stato servizio aggiornato nel database.

Invariante: 3 servizi mai eliminati fisicamente dal database (Bowling, Go-Kart, Biliardo).