Università degli Studi di Salerno

Corso di Ingegneria del Software

UStrike!

RAD

Versione 1.4

Data: 11/11/2025

Informazioni sul Progetto

Progetto: UStrike!
Documento: RAD
Versione: 1.4
Data: 11/11/2025

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	Stefanile Andrea, Valito Marcello
13/12/2025	0.2		Russo Serena, Valito Marcello
14/12/2025	0.3	Iniziale scrittura della class interface – control, utils e parte della sezione database -	Russo Serena, Stefanile Andrea, Valito Marcello
16/12/2025	0.4	Inserimento di collection e handler nella class interface e completamento della sezione database	Russo Serena, Stefanile Andrea, Valito Marcello
17/12/2025	0.5	Ultimazione della class interface – service, handler	Stefanile Andrea, Valito Marcello
18/12/2025	1.0	Revisione e correzione del documento	Russo Serena, Stefanile Andrea, Valito Marcello

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 (hash password BCrypt, 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 (BCrypt, 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
ACL	Access Control List
Servlet	Controller HTTP che gestisce request/response
Action	Parametro dispatch per polimorfismo funzionale

5. Riferimenti

RAD UStrike! v1.4 (Russo S., Stefanile A., Valito M.) - 11/11/2025

SDD UStrike! v1.0 (Russo S., Stefanile A., Valito M.) - 25/11/2025

Problem Statement UStrike! v1.1 - 13/10/2025

6. Packages

L'implementazione del back-end è organizzata nella directory /java/ con i seguenti package: it.unisa.uStrike.control, it.unisa.uStrike.service, it.unisa.uStrike.model, it.unisa.uStrike.database, it.unisa.uStrike.handler, it.unisa.uStrike.utils, it.unisa.uStrike.collection.

L'implementazione del front-end è organizzata nella directory /webapp/: **View (containers)** in /webapp/ (JSP principali), **View (components)** in /webapp/View/ (JSP componenti), **View (sub-components/AJAX)** in /webapp/View/AJAX_Components/ (JSP AJAX).

7. Back-end Packages

Control

Classe	Descrizione
LoginServlet	Servlet di login e autenticazione
RegistrazioneServlet	Servlet di registrazione nuovi clienti
LogoutServlet	Servlet di chiusura sessione
ProfiloServlet	Servlet impostazioni profilo (dati + cambio password)
PrenotazioniServlet	Gestione prenotazioni (crea/cancella/accetta/rifiuta/assegna)
TurnicaServlet	Gestione turnistica (crea/modifica/visualizza)
RisorseServlet	CRUD risorse fisiche (piste/tavoli)
PremioServlet	CRUD premi + visualizza catalogo
ServizioServlet	Visualizza/abilita/disabilita servizi
DashboardServlet	Dashboard role-based (Cliente/Staff/Manager)
TicketServlet	Riscatto premi tramite ticket
NotificheServlet	Notifiche non lette / marca come lette
FileUploadServlet	Upload immagini premi (multipart/form-data)

Database

Classe	Descrizione
DBConnectionPool	Gestione connessioni JDBC (create/get/release)

Handler

Classe	Descrizione
UStrikeContext	Listener/contesto applicativo per init risorse condivise
AuthFilter	Filtro autenticazione (verifica utente loggato)
RoleFilter	Filtro autorizzazione (ACL e ruoli)
ViewResolver	Dispatcher view in base al ruolo

Utils

Classe	Descrizione
PasswordHasher	Hash password (BCrypt)
InputValidator	Validazione form (email/password/date/orari)
MultipartValidator	Validazione upload (estensione/dimensione file)

Collection

Classe	Descrizione
ParsedPrenotazione	DTO per render leggibile/sicuro di prenotazione
ParsedPremio	DTO per catalogo premi

Service

Classe	Descrizione

UserService	Operazioni su utenti + login/sessione + recupero password
PrenotazioneService	Business logic prenotazioni (stati, deadline, mora)
TurnoService	Operazioni su turni settimanali
PremioService	CRUD premi
RisorsaService	Disponibilità/occupazione risorse
ServizioService	Abilita/disabilita/lettura servizi
TicketService	Riscatto e decremento ticket
NotificaService	Coda/invio/lettura notifiche
UploadService	Salvataggio file e generazione URL immagini

Model

Classe	Descrizione
User	Classe base per utenti
Cliente	Utente cliente con punti/ticket
Staff	Utente staff con ruolo
Manager	Utente manager
Prenotazione	Prenotazione (data, fascia, stato, risorsa, staff, cliente)
Turno	Turno lavoro (data, fascia, staff, manager)
Premio	Premio riscattabile (nome, descrizione, valoreTicket, immagineUrl)
Risorsa	Risorsa fisica (stato, capacità, servizio)
Servizio	Servizio (nome, stato, capacità)
Notifica	Notifica (utente, testo, letta/non letta, data)

8. Front-end Packages

/webapp/ (View Containers)

JSP	Descrizione
index.jsp	Landing page pubblica
login.jsp	Form login
registrazione.jsp	Form registrazione
dashboard.jsp	Container dashboard (include componenti per ruolo)

/webapp/View/ (View Components)

JSP	Descrizione
profilo.jsp	Dati profilo + cambio password
prenotazioni.jsp	Elenco prenotazioni (cliente) o gestione (staff)
turnistica.jsp	Turni (visualizzazione staff / gestione manager)
premi.jsp	Catalogo premi (per clienti)
gestionePremi.jsp	CRUD premi (per staff)
gestioneServizi.jsp	Abilita/disabilita servizi (per staff)
gestioneRisorse.jsp	CRUD risorse (per manager)
notifiche.jsp	Lista notifiche non lette

/webapp/View/AJAX_Components/ (Sub-components)

JSP	Descrizione
prenotazioniTable.jsp	Tabella prenotazioni con filtri/stati

turniGrid.jsp	Griglia turni settimanali (giorni × fasce)
premiList.jsp	Lista premi con descrizione/valore ticket
notificheList.jsp	Lista notifiche non lette
availabilityCalendar.jsp	Calendario disponibilità servizi

9. Class Interface

9.1 Control

LoginServlet

Descrizione: Servlet che gestisce l'accesso alla piattaforma UStrike!. Verifica credenziali email/password, crea sessione e reindirizza a dashboard role-based.

Signature dei metodi:

doPost(HttpServletRequest request, HttpServletResponse response) : void

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

Post-condizioni: Se credenziali valide: sessione creata, redirect a dashboard. Se credenziali non valide: messaggio di errore e refresh 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 dati, verifica email univoca e crea account con ruolo CLIENTE.

Signature dei metodi:

doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Parametri nome, cognome, email, password, confPassword presenti; Email non già registrata; Password >= 8 caratteri, contiene maiuscola/minuscola/numero; Connessione a database disponibile.

Post-condizioni: Utente creato nel DB con ruolo CLIENTE; Email di conferma inviata (opzionale); Redirect a login.jsp con messaggio di successo.

Invariante: Email univoca nel sistema. Password salvata solo come hash.

LogoutServlet

Descrizione: Servlet che gestisce la chiusura 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; Redirect a index.jsp; Nessun dato sensibile residuo in sessione.

Invariante: Logout sempre possibile, anche da stato inconsistente.

ProfiloServlet

Descrizione: Servlet per visualizzare e modificare profilo utente (dati personali e password).

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "modificaDati" | "cambiaPassword"

Pre-condizioni: Utente autenticato (sessione valida); Per POST: azione specificata in parametro action; Connessione a database disponibile.

Post-condizioni: GET: profilo.jsp con dati attuali visualizzati. POST modificaDati: campi nome/cognome/email aggiornati nel DB. POST cambiaPassword: password aggiornata (hash) se attuale corrisponde. Sessione aggiornata se email cambia.

Invariante: Email rimane univoca. Password vecchia deve essere verificata prima di cambio. Password nuova non può essere uguale a precedente.

PrenotazioniServlet

Descrizione: Servlet che gestisce TUTTE le operazioni sulle prenotazioni (UC8–UC20). Utilizza action-based dispatch.

Signature dei metodi:

doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "creaPrenotazione" | "cancellaPrenotazione" | "rifiutaPrenotazione" | "accettaPrenotazione" | "assegnaRisorsa"

Pre-condizioni: Autenticazione verificata (AuthFilter); Autorizzazione coerente: cliente per crea/cancella, staff per accetta/rifiuta/assegna; Parametri richiesti presenti (idServizio, data, orario, etc.); Connessione a database disponibile.

Post-condizioni: Prenotazione creata/modificata/cancellata/accettata/rifiutata/risorsa assegnata. Stato coerente (In attesa → Confermata → Occupata risorsa). Notifiche generate e accodate. Response JSON con esito operazione.

Invariante: No overbooking: una risorsa non occupata due volte stesso slot. Deadline cancellazione 24 ore prima fascia oraria. Mora applicata se cancellazione < 24 ore.

TurnisticaServlet

Descrizione: Servlet che gestisce creazione, modifica e visualizzazione turnistica settimanale.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "creaTurni" | "modificaTurni"

Pre-condizioni: Per GET: staff autenticato; settimana come parametro. Per POST: manager autenticato; settimana + grid turni presenti. Connessione a database disponibile. Staff riferiti esistenti nel DB.

Post-condizioni: GET: turniGrid.jsp con tabella giorni × fasce + staff assegnato. POST create: turni salvati nel DB, visibili a staff. POST modify: turni aggiornati, notifiche inviate per cambiamenti. Response JSON con esito operazione.

Invariante: Turni sempre riferiti a settimana futura (no passato). Ogni cella (giorno, fascia) ha un solo staff assegnato. Manager che crea/modifica tracciato.

RisorseServlet

Descrizione: Servlet che gestisce CRUD delle risorse fisiche (piste bowling, tavoli biliardo, piste go-kart).

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "creaRisorsa" | "modificaRisorsa" | "eliminaRisorsa"

Pre-condizioni: Manager autenticato (AuthFilter + RoleFilter); Parametri: nome, servizio, capacità (per create/modify); Risorsa da eliminare non in uso (prenotazione attiva).

Post-condizioni: Risorsa creata/modificata/eliminata nel DB. Stato iniziale di nuova risorsa: libera. Response JSON con esito operazione.

Invariante: Risorsa sempre associata a servizio valido (Bowling/Go-Kart/Biliardo). Capacità >= 1. Risorsa in uso non eliminabile.

PremioServlet

Descrizione: Servlet che gestisce CRUD premi e visualizzazione catalogo premi.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "catalogo" (default)
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "aggiungi" | "rimuovi" | "modifica"

Pre-condizioni: GET: nessuna autenticazione richiesta. POST: staff autenticato. Parametri per aggiungi/modifica: nome, descrizione, valoreTicket, immagineUrl. Connessione a database disponibile.

Post-condizioni: GET catalogo: JSON lista premi visibile. POST aggiungi: premio salvato nel DB, visibile in catalogo. POST rimuovi: premio rimosso dal catalogo. POST modifica: premio aggiornato nel catalogo. Catalogo sempre aggiornato in tempo reale per tutti i clienti.

Invariante: valoreTicket > 0. Nome premio univoco. immagineUrl deve essere URL valido (salvato da FileUploadServlet).

FileUploadServlet

Descrizione: Servlet che gestisce l'upload multipart di immagini per i premi. Valida estensione/dimensione, salva file e restituisce URL.

Signature dei metodi:

doPost(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Staff autenticato (AuthFilter); Request multipart/form-data; Part "immaginePremio" presente; File estensione .jpg o .png; Dimensione file <= 5MB.

Post-condizioni: File salvato in `/uploads/premi/{timestamp}_{nomeOriginale}.jpg/.png`. Response JSON con `immagineUrl` (percorso relativo/assoluto). URL disponibile per PremioServlet doPost.

Invariante: Solo .jpg/.png accettati (validazione MIME type). Nomi file sanitizzati (no special characters). File salvato con timestamp per unicità.

ServizioServlet

Descrizione: Servlet per visualizzazione servizi (Bowling, Go-Kart, Biliardo) e toggle abilita/disabilita.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "abilita" | "disabilita"

Pre-condizioni: GET: nessuna autenticazione. POST: staff autenticato; idServizio presente. Connessione a database disponibile.

Post-condizioni: GET: JSON lista servizi con stato (abilitato/disabilitato). POST: stato servizio aggiornato nel DB. Clienti non possono prenotare servizio disabilitato.

Invariante: Prenotazioni esistenti non invalidate da disabilitazione servizio. Servizi sempre 3 nel sistema (Bowling, Go-Kart, Biliardo).

DashboardServlet

Descrizione: Servlet che renderizza la dashboard personalizzata in base al ruolo dell'utente (Cliente/Staff/Manager).

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato (AuthFilter); Ruolo determinato dalla sessione.

Post-condizioni: Dashboard.jsp con componenti role-specific. Cliente: prenotazioni personali, punti ticket, catalogo premi. Staff: prenotazioni sospese, turni settimana. Manager: statistiche globali, staff online, ricavi.

Invariante: Coerenza tra ruolo in sessione e contenuto visualizzato. Dati sempre aggiornati al momento della richiesta.

TicketServlet

Descrizione: Servlet che gestisce il riscatto di premi tramite ticket/punti.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "riscatta"

Pre-condizioni: Cliente autenticato (AuthFilter); Parametro idPremio presente; Punti sufficienti (>= valoreTicket del premio); Connessione a database disponibile.

Post-condizioni: Punti decrementati nel DB. Premio riscattato (notifica generata). Response JSON con esito operazione.

Invariante: Punti mai negativi. Ogni riscatto crea una notifica.

NotificheServlet

Descrizione: Servlet per lettura notifiche non lette e marcatura come lette.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void
doPost(HttpServletRequest request, HttpServletResponse response) : void
action parameter: "marcaLette"

Pre-condizioni: Utente autenticato (AuthFilter); Connessione a database disponibile.

Post-condizioni: GET: JSON lista notifiche non lette per l'utente. POST: notifiche specificate marcate come lette nel DB.

Invariante: Notifica appartiene a un solo utente. Non può essere cancellata, solo marcata letta.

9.2 Utils

PasswordHasher

Descrizione: Classe utility che encapsula hashing password con BCrypt per crittografia sicura.

Signature dei metodi:

static hash(String password) : String
static verify(String passwordPlaintext, String passwordHash) : boolean

Pre-condizioni: password non nullo; password lunghezza >= 8.

Post-condizioni: Restituisce hash SHA-256/BCrypt (mai plaintext). verify() ritorna true solo se match.

Invariante: Stesso hash BCrypt può variare tra esecuzioni (salt casuale). verify() sempre idempotente.

InputValidator

Descrizione: Classe utility che valida input form (email, password, date, orari).

Signature dei metodi:

static isValidEmail(String email) : boolean
static isValidPassword(String password) : boolean
static isValidDate(String date) : boolean
static isValidTimeSlot(String time) : boolean
static isValidInteger(String value) : boolean

Pre-condizioni: Parametri non nulli.

Post-condizioni: Restituisce true/false in base a validazione.

Invariante: Email: formato RFC 5322 base. Password: >= 8 car, maiuscola, minuscola, numero. Date: formato YYYY-MM-DD, futura. Orari: HH:00 (00-23).

MultipartValidator

Descrizione: Classe utility che valida file upload (estensione, dimensione).

Signature dei metodi:

static isValidImageExtension(String filename) : boolean
static isValidFileSize(long sizeBytes, long maxBytes) : boolean

Pre-condizioni: filename non nullo; sizeBytes >= 0.

Post-condizioni: Estensione valida: .jpg, .png. Dimensione <= 5MB (5242880 bytes).

Invariante: Validazione lato server (non client).

9.3 Database

DBConnectionPool

Descrizione: Classe che gestisce pool di connessioni JDBC al database MySQL per evitare overhead creazione connessioni.

Signature dei metodi:

static createDBConnection() : Connection
static getConnection() : Connection
static releaseConnection(Connection conn) : void
static closeAllConnections() : void

Pre-condizioni: Configurazione DB (host, user, password) disponibile in .env; MySQL server running.

Post-condizioni: Connection pronta all'uso (open). Connection rilasciato e tornato al pool dopo uso.

Invariante: Pool size massimo (es. 10 connessioni). Nessuna connessione leak.

9.4 Collection

ParsedPrenotazione

Descrizione: Data Transfer Object (DTO) che rappresenta una prenotazione in modo sicuro e leggibile per render su JSP.

Signature dei metodi:

ParsedPrenotazione(Prenotazione p)
getIdPrenotazione() : int
getDataPrenotazione() : String
getOrariFascia() : String
getStatoPrenotazione() : String
getNomeCliente() : String
getNomeServizio() : String
getNumeroRisorsa() : String
getPartecipi() : int
getMora() : double

Pre-condizioni: Prenotazione non nulla.

Post-condizioni: Dati formattati per visualizzazione (es. date in IT locale).

Invariante: Contenuto immutabile (getter only).

ParsedPremio

Descrizione: DTO per render sicuro di un premio nel catalogo (mostra solo campi pubblici).

Signature dei metodi:

ParsedPremio(Premio prize)
getIdPremio() : int
getNomePremio() : String
getDescrizionePremio() : String
getValoreTicket() : int
getImmagineUrl() : String

Pre-condizioni: Premio non nullo.

Post-condizioni: URL immagine disponibile e valido.

Invariante: Contenuto immutabile.

9.5 Handler

UStrikeContext

Descrizione: ServletContextListener che inizializza risorse condivise dell'applicazione (pool DB, config globali).

Signature dei metodi:

contextInitialized(ServletContextEvent sce) : void
contextDestroyed(ServletContextEvent sce) : void

Pre-condizioni: Applicazione in start.

Post-condizioni: Pool DB inizializzato. Risorse globali pronte. Listener registrato in web.xml.

Invariante: Singleton pattern per accesso risorse globali.

AuthFilter

Descrizione: Filter che verifica autenticazione (sessione valida) per accesso a risorse protette.

Signature dei metodi:

init(FilterConfig config) : void
doFilter(ServletRequest req, ServletResponse res, FilterChain chain) : void
destroy() : void

Pre-condizioni: Request contiene cookie di sessione (o no); Sessione valida oppure scaduta.

Post-condizioni: Se sessione valida: forward a risorsa richiesta. Se sessione non valida: redirect a login.jsp. Chain continuata per Filter successivi.

Invariante: Sessione sempre verificata prima di accesso.

RoleFilter

Descrizione: Filter che verifica autorizzazione in base al ruolo dell'utente (ACL). Implementa access control matrix.

Signature dei metodi:

init(FilterConfig config) : void
doFilter(ServletRequest req, ServletResponse res, FilterChain chain) : void
destroy() : void

Pre-condizioni: AuthFilter ha passato (sessione valida); Ruolo utente noto dalla sessione.

Post-condizioni: Se ruolo autorizzato per risorsa: forward. Se ruolo non autorizzato: errore 403 Forbidden. Chain continuata.

Invariante: ACL sempre applicato prima di logica servlet.

ViewResolver

Descrizione: Servlet che risolve qual è il JSP giusto da mostrare in base al ruolo dell'utente.

Signature dei metodi:

doGet(HttpServletRequest request, HttpServletResponse response) : void

Pre-condizioni: Utente autenticato; Ruolo determinato.

Post-condizioni: Include JSP corretto (dashboard.jsp con componenti diversi per Cliente/Staff/Manager).

Invariante: Un ruolo = un set coerente di JSP visualizzabili.

9.6 Service

UserService

Descrizione: Classe che gestisce tutte le operazioni su utenti (CRUD, login, sessione, recupero password).

Signature dei metodi:

UserService(Connection conn)
getUserById(int userId) : User
getUserByEmail(String email) : User
createUser(User user) : boolean
updateUser(User user) : boolean
deleteUser(int userId) : boolean
authenticateUser(String email, String password) : User
createSession(int userId) : String
getSessionUser(String sessionToken) : User
invalidateSession(String sessionToken) : void
resetPassword(String email) : String
getUserRole(int userId) : String

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

Post-condizioni: Operazione completata e DB coerente. Session token restituito (per login).

Invariante: Email univoca. Password mai salvata plaintext.

PrenotazioneService

Descrizione: Classe che gestisce business logic delle prenotazioni (stati, deadline 24h, mora).

Signature dei metodi:

PrenotazioneService(Connection conn)
createPrenotazione(Prenotazione p) : Prenotazione
cancellaPrenotazione(int idPrenotazione) : double
accettaPrenotazione(int idPrenotazione, int idStaff) : boolean

rifiutaPrenotazione(int idPrenotazione, String motivazione) : boolean
assegnaRisorsa(int idPrenotazione, int idRisorsa) : boolean
getPrenotazioniByCliente(int idCliente) : ArrayList<Prenotazione>
getPrenotazioniByStaff(int idStaff) : ArrayList<Prenotazione>
getPrenotazioniByServizio(int idServizio) : ArrayList<Prenotazione>
verifyAvailability(int idServizio, String data, String fascia, int capacita) : boolean
checkDeadlineAndApplyMora(int idPrenotazione) : double

Pre-condizioni: Connessione DB valida; Risorsa disponibile (per create); Staff autenticato (per accetta/rifiuta/assegna).

Post-condizioni: Prenotazione stato coerente (In attesa → Confermata → Occupata). Mora calcolata e addebitata se applicable. Notifiche generate.

Invariante: No overbooking. Deadline 24 ore prima fascia.

TurnoService

Descrizione: Classe che gestisce operazioni su turni settimanali.

Signature dei metodi:

TurnoService(Connection conn)
createTurni(String servizio, String settimana, Map<String, Map<String, Integer>> grid) : boolean
modifyTurni(String servizio, String settimana, Map<String, Map<String, Integer>> grid) : boolean
getTurniBySettimana(String settimana, String servizio) : ArrayList<Turno>
getTurniByStaff(int idStaff) : ArrayList<Turno>
getTurniBySettimanaAndServizio(String settimana, String servizio) : ArrayList<Turno>

Pre-condizioni: Manager autenticato; Settimana futura (no passato); Staff ID validi.

Post-condizioni: Turni salvati nel DB. Notifiche inviate a staff.

Invariante: Ogni cella (giorno, fascia) un solo staff.

PremioService

Descrizione: Classe che gestisce CRUD premi.

Signature dei metodi:

PremioService(Connection conn)
createPremio(Premio p) : Premio
getPremioById(int id) : Premio
getAllPremi() : ArrayList<Premio>
updatePremio(Premio p) : boolean

deletePremio(int id) : boolean
searchPremio(String keyword) : ArrayList<Premio>

Pre-condizioni: Staff autenticato (per create/update/delete); Premio non nullo; valoreTicket > 0.

Post-condizioni: Premio persistito. Catalogo aggiornato per clienti.

Invariante: Nome premio univoco.

RisorsaService

Descrizione: Classe che gestisce risorse fisiche e loro disponibilità.

Signature dei metodi:

RisorsaService(Connection conn)
createRisorsa(Risorsa r) : Risorsa
getRisorsaById(int id) : Risorsa
getRisorseByServizio(int idServizio) : ArrayList<Risorsa>
updateRisorsa(Risorsa r) : boolean
deleteRisorsa(int id) : boolean
checkDisponibilita(int idRisorsa, String data, String fascia) : boolean
occupaRisorsa(int idRisorsa, String data, String fascia) : void
liberaRisorsa(int idRisorsa, String data, String fascia) : void

Pre-condizioni: Manager autenticato; Capacità >= 1.

Post-condizioni: Risorsa creata/modificata/eliminata. Stato (libera/occupata) aggiornato.

Invariante: Risorsa sempre riferita a servizio valido.

ServizioService

Descrizione: Classe che gestisce servizi principali (Bowling, Go-Kart, Biliardo).

Signature dei metodi:

ServizioService(Connection conn)
getServizioById(int id) : Servizio
getAllServizi() : ArrayList<Servizio>
abilitaServizio(int id) : boolean
disabilitaServizio(int id) : boolean
getServizioAbilitato(int id) : boolean

Pre-condizioni: Staff autenticato (per abilita/disabilita).

Post-condizioni: Stato servizio aggiornato.

Invariante: 3 servizi sempre presenti (Bowling, Go-Kart, Biliardo).

TicketService

Descrizione: Classe che gestisce riscatto premi tramite ticket.

Signature dei metodi:

TicketService(Connection conn)
getPuntiCliente(int idCliente) : int
riscattaPremio(int idCliente, int idPremio) : boolean
decrementaTicket(int idCliente, int valore) : boolean
incrementaTicket(int idCliente, int valore) : boolean

Pre-condizioni: Cliente autenticato; Punti >= valoreTicket del premio.

Post-condizioni: Punti decrementati nel DB. Notifica generata.

Invariante: Punti mai negativi.

NotificaService

Descrizione: Classe che gestisce generazione, accodamento e lettura notifiche.

Signature dei metodi:

NotificaService(Connection conn)
creaNotifica(int idUtente, String testo) : Notifica
getNotificheNonLette(int idUtente) : ArrayList<Notifica>
marcaComeLetta(int idNotifica) : boolean
deleteNotifica(int id) : boolean

Pre-condizioni: Utente autenticato; Connessione DB valida.

Post-condizioni: Notifica creata/marcata letta.

Invariante: Notifica appartiene a un utente.

UploadService

Descrizione: Classe che gestisce salvataggio file upload e generazione URL.

Signature dei metodi:

UploadService()
saveFile(Part filePart, String uploadDir) : String
validateFile(Part filePart, long maxSize) : boolean
generateFileName(String originalFilename) : String
deleteFile(String filepath) : boolean

Pre-condizioni: File valido (.jpg/.png); Dimensione <= 5MB.

Post-condizioni: File salvato in `/uploads/premi/{timestamp}_{nomeOriginale}`. URL restituito per salvataggio in DB.

Invariante: Filename sanitizzato (no special char).

9.7 Model

User

Descrizione: Classe base che descrive un utente generico del sistema.

Signature dei metodi:

User(int id, String nome, String cognome, String email, String password)
getId() : int
getNome() : String
getCognome() : String
getEmail() : String
getPassword() : String
setId(int id) : void
setNome(String nome) : void
setCognome(String cognome) : void
setEmail(String email) : void
setPassword(String password) : void
getFullName() : String

Pre-condizioni: Email non nulla e valida; Password non nulla (hash).

Post-condizioni: User istanziato.

Invariante: Email univoca. Password mai plaintext.

Cliente

Descrizione: Classe che estende User per un cliente (utente registrato che fa prenotazioni).

Signature dei metodi:

Cliente(int id, String nome, String cognome, String email, String password, int puntiTicket)
getPuntiTicket() : int
setPuntiTicket(int punti) : void
incrementaPunti(int valore) : void
decrementaPunti(int valore) : void
getPrenotazioni() : ArrayList<Prenotazione>

Pre-condizioni: Eredita da User.

Post-condizioni: Cliente istanziato con punti >= 0.

Invariante: Punti mai negativi.

Staff

Descrizione: Classe che estende User per uno staff member (addetto ai servizi).

Signature dei metodi:

Staff(int id, String nome, String cognome, String email, String password, String ruolo)
getRuolo() : String
setRuolo(String ruolo) : void
getTurni() : ArrayList<Turno>
getPrenotazioniAssegnate() : ArrayList<Prenotazione>

Pre-condizioni: Eredita da User; Ruolo valido.

Post-condizioni: Staff istanziato.

Invariante: Ruolo != null.

Manager

Descrizione: Classe che estende User per un manager (responsabile gestione turni/premi/servizi).

Signature dei metodi:

Manager(int id, String nome, String cognome, String email, String password)
getTurniCreati() : ArrayList<Turno>
getStatistiche() : Map<String, Object>

Pre-condizioni: Eredita da User.

Post-condizioni: Manager istanziato.

Invariante: Accesso a funzioni admin.

Prenotazione

Descrizione: Classe che descrive una prenotazione di servizio.

Signature dei metodi:

Prenotazione(int id, int idCliente, int idServizio, String data, String fascia, ArrayList<String> partecipanti, String stato)
getId() : int
getIdCliente() : int
getIdServizio() : String
getData() : String
getFascia() : String
getPartecipanti() : ArrayList<String>

getStato() : String
getIdRisorsa() : int
getIdStaff() : int
getMora() : double
setId(int id) : void
setStato(String stato) : void
setIdRisorsa(int id) : void
setIdStaff(int id) : void
setMora(double mora) : void

Pre-condizioni: Nulla di nulla.

Post-condizioni: Prenotazione istanziata.

Invariante: Stato in {In attesa, Confermata, Rifiutata, Annullata, Occupata}.

Turno

Descrizione: Classe che descrive un turno di lavoro settimanale.

Signature dei metodi:

Turno(int id, String data, String fascia, int idStaff, int idManager, String servizio)
getId() : int
getData() : String
getFascia() : String
getIdStaff() : int
getIdManager() : int
getServizio() : String
setId(int id) : void
setData(String data) : void
setFascia(String fascia) : void
setIdStaff(int id) : void

Pre-condizioni: Data futura; Fascia valida (Mattina 08–17, Sera 17–01).

Post-condizioni: Turno istanziato.

Invariante: Un turno per cella (giorno × fascia × servizio).

Premio

Descrizione: Classe che descrive un premio riscattabile con ticket.

Signature dei metodi:

Premio(int id, String nome, String descrizione, int valoreTicket, String immagineUrl)
getId() : int
getNome() : String

getDescrizione() : String
getValoreTicket() : int
getImmagineUrl() : String
setId(int id) : void
setNome(String nome) : void
setDescrizione(String desc) : void
setValoreTicket(int valore) : void
setImmagineUrl(String url) : void

Pre-condizioni: valoreTicket > 0; immagineUrl valido.

Post-condizioni: Premio istanziato.

Invariante: Nome univoco. immagineUrl != null.

Risorsa

Descrizione: Classe che descrive una risorsa fisica (pista, tavolo).

Signature dei metodi:

Risorsa(int id, String nome, String stato, int capacita, int idServizio)
getId() : int
getNome() : String
getStato() : String
getCapacita() : int
getIdServizio() : int
setId(int id) : void
setNome(String nome) : void
setStato(String stato) : void
setCapacita(int cap) : void
setIdServizio(int id) : void

Pre-condizioni: capacita >= 1; stato in {libera, occupata}.

Post-condizioni: Risorsa istanziata.

Invariante: Risorsa riferita a servizio valido.

Servizio

Descrizione: Classe che descrive un servizio principale (Bowling, Go-Kart, Biliardo).

Signature dei metodi:

Servizio(int id, String nome, String stato)
getId() : int
getNome() : String
getStato() : String

setId(int id) : void
setNome(String nome) : void
setStato(String stato) : void
isAbilitato() : boolean

Pre-condizioni: Nome in {Bowling, Go-Kart, Biliardo}; stato in {abilitato, disabilitato}.

Post-condizioni: Servizio istanziato.

Invariante: Sempre 3 servizi nel sistema.

Notifica

Descrizione: Classe che descrive una notifica inviata a un utente.

Signature dei metodi:

Notifica(int id, int idUtente, String testo, boolean letta, String data)
getId() : int
getIdUtente() : int
getTesto() : String
isLetta() : boolean
getData() : String
setId(int id) : void
setIdUtente(int id) : void
setTesto(String testo) : void
setLetta(boolean letta) : void
setData(String data) : void

Pre-condizioni: idUtente valido; testo non nullo.

Post-condizioni: Notifica istanziata, letta = false.

Invariante: Notifica appartiene a un utente.