

PoolManager

Alessandro Aldo Raoul Bonciani

June 2024



Contents

1	Introduzione	3
1.1	Obiettivo e descrizione del progetto	3
1.2	Struttura e pratiche utilizzate	3
2	Progettazione	4
2.1	Use case diagram	4
2.2	Use case template	5
2.3	Mockups	9
2.4	Class diagram	12
2.5	ER Diagram e modello relazionale	14
2.6	Navigation Diagram	16
3	Implementazione	18
3.1	Domain Model	18
3.1.1	Object	18
3.1.2	Chair	18
3.1.3	Deckchair	18
3.1.4	Sunbed	18
3.1.5	Table	18
3.1.6	Umbrella	18
3.1.7	TimeRecord	18
3.1.8	Postation	18
3.1.9	User	19
3.1.10	Location	19
3.1.11	Reservation	19
3.2	ORM	19
3.2.1	UserDAO	19
3.2.2	AdminDAO	19
3.2.3	ObjectDAO	19
3.2.4	TimeRecordDAO	19
3.2.5	LocationDAO	19
3.2.6	PostationDAO	20
3.2.7	ReservationDAO	20
3.2.8	ConnectionManager	20
3.3	Business Logic	20
3.3.1	AdminController	20
3.3.2	UserController	20
3.3.3	ResourcesController	20
3.3.4	ReserveController	20
3.4	Database	21
3.5	Interfaccia e CLI	21

1 Introduzione

Elaborato per il superamento dell'esame di Ingegneria del Software, appartenente al modulo Basi di Dati/Ingegneria del Software del corso di laurea Triennale in Ingegneria Informatica presso l'Università degli Studi di Firenze.

Il progetto è stato sviluppato da Alessandro Aldo Raoul Bonciani, matricola 7079209, nel periodo tra maggio - giugno 2024 (a.a. 2023-2024).

Il codice sorgente è disponibile su *GitHub* al seguente indirizzo:
<https://github.com/alebonch/PoolManager>

1.1 Obiettivo e descrizione del progetto

Con questo progetto si sviluppa un programma adibito alla gestione delle prenotazioni di postazioni per la balneazione in un impianto natatorio.

Si ha quindi un sistema in cui, utenti e admin, possono inserire prenotazioni andando a richiedere un numero di risorse per la propria postazione. Inoltre, il sistema di amministrazione, a cui si accede come admin, permette di modificare dati di utenti e la disponibilità delle risorse. Per gestire i dati e salvarli, il programma è stato connesso ad un database.

1.2 Struttura e pratiche utilizzate

Il software è stato sviluppato in Java, mentre per la gestione e il salvataggio dei dati è stato connesso un database PostgreSQL ed è stata utilizzata la libreria JDBC (Java DataBase Connectivity).

Per mantenere una separazione delle responsabilità, la struttura del progetto è stata divisa in tre parti principali: Business Logic, Domain Model e ORM. Questi tre packages si occupano in modo distinto della logica di business, della rappresentazione dei dati e dell'accesso ai dati (Figura 1):

- **Business Logic:** contiene le classi che implementano la logica di business del sistema.
- **Domain Model:** contiene le classi che rappresentano le entità del sistema.
- **ORM:** contiene le classi che implementano l'Object-Relational Mapping. In questo modo è possibile rendere i dati persistenti e recuperarli dal database.

Per utilizzare il software è stata creata un'interfaccia da riga di comando (CLI) che permette di interagire con il sistema in modo semplice e intuitivo.

Gli Use Case Diagram e i Class Diagram seguono lo standard UML (Unified Modeling Language) e sono stati realizzati con il software StarUML.

Le piattaforme e i software utilizzati sono stati:

- **VisualStudio Code:** IDE per lo sviluppo in Java
- **StarUML:** software per la creazione di diagrammi UML

- **PgAdmin**: software per la gestione di database PostgreSQL
- **GitHub**: piattaforma per la condivisione del codice sorgente
- **Lunacy**: software per la creazione di mock-up
- **Overleaf.com**: sito per la creazione di file latex

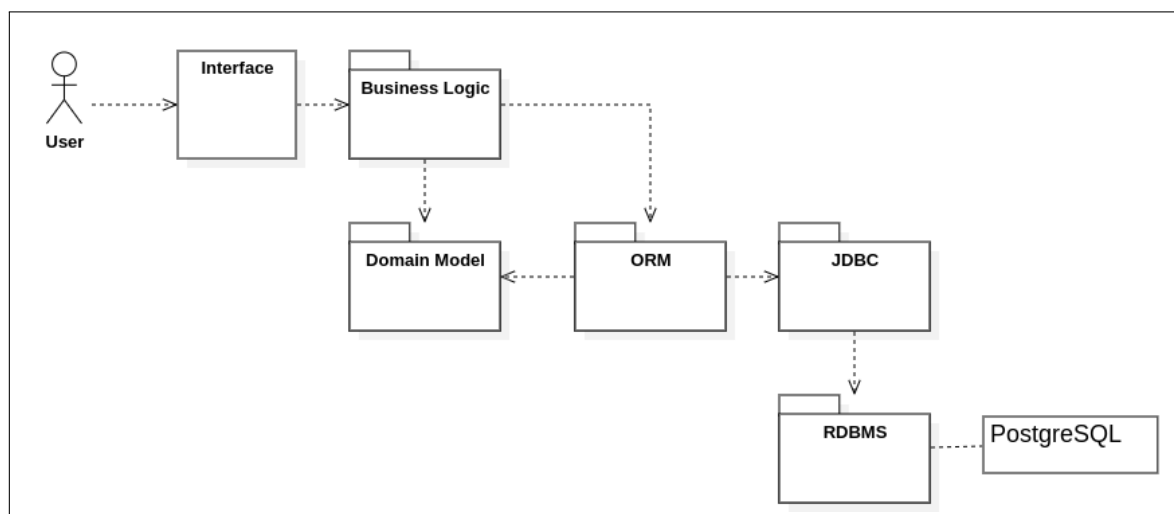


Figure 1: Package Dependency Diagram

2 Progettazione

2.1 Use case diagram

Come precedentemente descritto, il sistema permette agli utenti di effettuare prenotazioni di postazioni in determinati turni di una giornata e di gestire il proprio profilo. La seguente immagine mostra il diagramma dei casi di uso del sistema sia per gli utenti che la parte amministrativa.

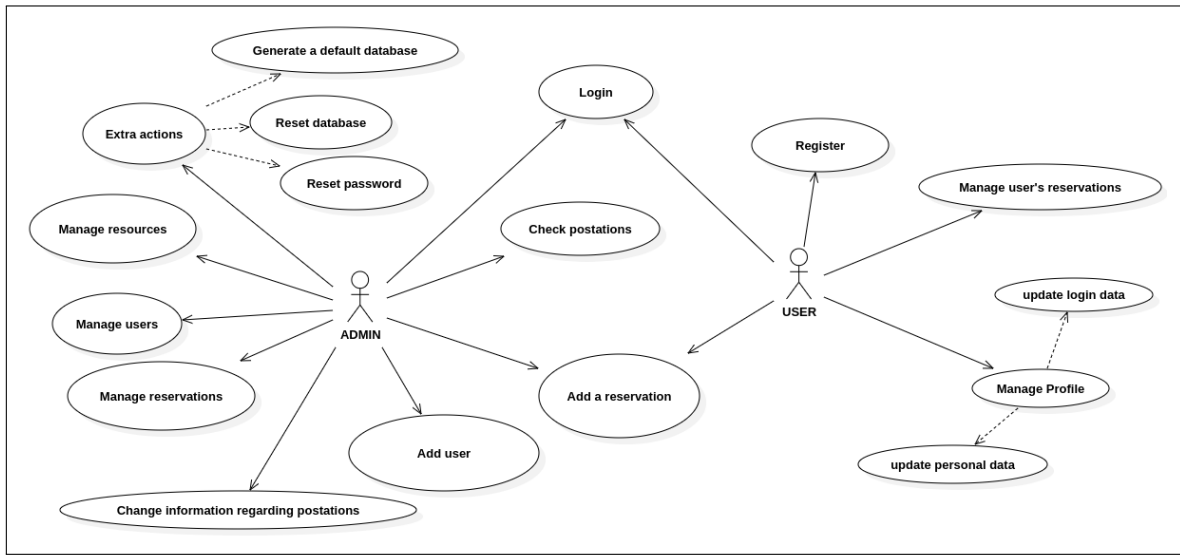


Figure 2: Use Case Diagram

2.2 Use case template

Di seguito sono riportati i template di alcuni dei casi d'uso implementati, in ognuno possiamo trovare: una breve descrizione, il livello del caso d'uso, gli attori coinvolti, le pre-condizioni, le post-condizioni, il flusso base e i flussi alternativi.

Use Case #1	Accesso al sistema (Log-in)
Brief Description	L'utente accede al sistema tramite le proprie credenziali (MK#1, TST#01)
Level	User Goal
Actors	Utente, Admin
Pre-conditions	L'utente deve essere nella pagina iniziale di accesso.
Basic Flow	<ol style="list-style-type: none"> 1. L'utente inserisce le proprie credenziali (username e password) 2. L'utente preme il pulsante di accesso 3. Il sistema verifica le credenziali 4. Il sistema autentica l'utente
Alternative Flows	<ol style="list-style-type: none"> 3a. Se le credenziali fornite non sono corrette, il sistema mostra un messaggio di errore all'utente 4a. Se a fare l'accesso è l'admin, il sistema lo reindirizza alla Admin Page (TST #03)
Post-conditions	L'utente è autenticato nel sistema e ha accesso alle funzionalità riservate

Table 1: Use Case Template 1 (Sign in)

Use Case #2	Registrazione nel sistema (Register)
Brief Description	L'utente si registra nel sistema creando un nuovo account
Level	User Goal
Actors	Utente
Pre-conditions	L'utente deve essere nella pagina iniziale di registrazione
Basic Flow	<ol style="list-style-type: none"> 1. L'utente fornisce i dettagli richiesti per la registrazione 2. L'utente conferma la registrazione 3. Il sistema verifica i dati forniti 4. Il sistema crea un nuovo account per l'utente
Alternative Flows	3a. Se l'utente fornisce dati non validi o se l'email o lo username sono già utilizzati da un altro account, il sistema mostra un messaggio di errore
Post-conditions	L'utente è registrato nel sistema e può accedere utilizzando le credenziali create durante la registrazione

Table 2: Use Case Template 2 (Sign up)

Use Case #3	Creazione di una prenotazione
Brief Description	L'utente prenota una postazione richiedendo un certo numero di risorse.
Level	User Goal
Actors	Utente, Admin
Pre-conditions	L'utente deve essere nella pagina di gestione delle prenotazioni
Basic Flow	<ol style="list-style-type: none"> 1. L'utente fornisce i dettagli richiesti per la prenotazione 2. Il sistema verifica che la richiesta sia accettabile 3. Il sistema calcola il costo della prenotazione 4. Il sistema crea una nuova prenotazione, una postazione e la inserisce nel database
Alternative Flows	3a. Se l'utente richiede troppe risorse, il sistema genera dei messaggi di errore
Post-conditions	La prenotazione viene inserita nel sistema e il numero di risorse disponibili in quel time record diminuirà in base alla richiesta

Table 3: Use Case Template 3 (Reserve)

Use Case #4	Ripristino del database
Brief Description	L'Admin esegue l'operazione di ripristino del database
Level	Function
Actors	Admin
Pre-conditions	L'Admin ha effettuato l'accesso al sistema
Basic Flow	<ol style="list-style-type: none"> 1. L'amministratore seleziona l'opzione per il ripristino del database 2. L'amministratore conferma l'avvio del processo di ripristino 3. Il sistema esegue il ripristino del database 4. Il sistema conferma all'amministratore che il ripristino è stato completato con successo
Alternative Flows	<ol style="list-style-type: none"> 3a. Se il processo di ripristino del database fallisce per qualsiasi motivo, il sistema mostra un messaggio di errore all'amministratore
Post-conditions	Il database è stato ripristinato con successo

Table 4: Use Case Template 4 (Reset database)

2.3 Mockups

Di seguito si inseriscono alcuni possibili Mock-ups relativi alle interfacce grafiche del sistema.



PoolManager

Sign in

username

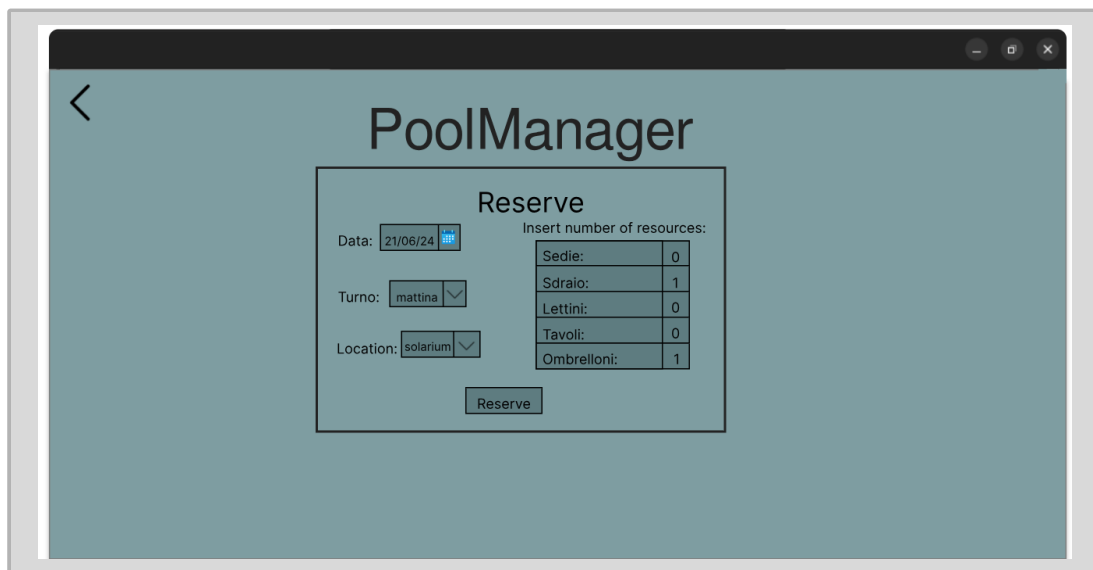
password

LOGIN

New to PoolManager? REGISTER

You're an admin? ADMIN

Figure 3: Prototipo della pagina di login / registrazione



<

PoolManager

Reserve

Data: 21/06/24

Turno: mattina

Location: solarium

Insert number of resources:

Sedie:	0
Sdraio:	1
Lettini:	0
Tavoli:	0
Ombrelloni:	1

Reserve

Figure 4: Prototipo della pagina di prenotazione



Figure 5: Prototipo della pagina di visione delle prenotazioni

2.4 Class diagram

Vista la divisione strutturale in 3 packages, sono stati realizzati 3 diagrammi delle classi distinti, uno per ogni package:

- **Business Logic** (Figura 6): contiene le classi che implementano la logica di business del sistema, ovvero i seguenti controller: quello che gestisce l'accesso e la registrazione dei nuovi utenti (**LoginController**), quello che permette agli utenti aggiornare i propri dati e controllare le proprie prenotazioni (**UserController**), quello che permette di creare e modificare le proprie prenotazioni (**ReserveController**), quello che permette di gestire le risorse disponibili e controllarne la disponibilità (**ResourcesController**), quello che gestisce le azioni dell'admin (**AdminController**)
- **Domain Model** (Figura 7): contiene le classi che rappresentano le entità del sistema, ovvero: **User**, **Object**, **Chair**, **Deckchair**, **Sunbed**, **Table**, **Umbrella**, **Reservation**, **Postation**, **Location** e **TimeRecord**, .
- **ORM** (Figura 8): contiene le classi che implementano l'Object-Relational Mapping, quindi contiene una classe per ogni entità del sistema: **UserDAO**, **ObjectDAO**, **PostationDAO**, **ReservationDAO**, **TimeRecordDAO** e **LocationDAO**. In più contiene anche la classe **AdminDAO** che si occupa della gestione dei dati dell'admin e la classe **ConnectionManager** che si occupa di gestire la connessione al database.

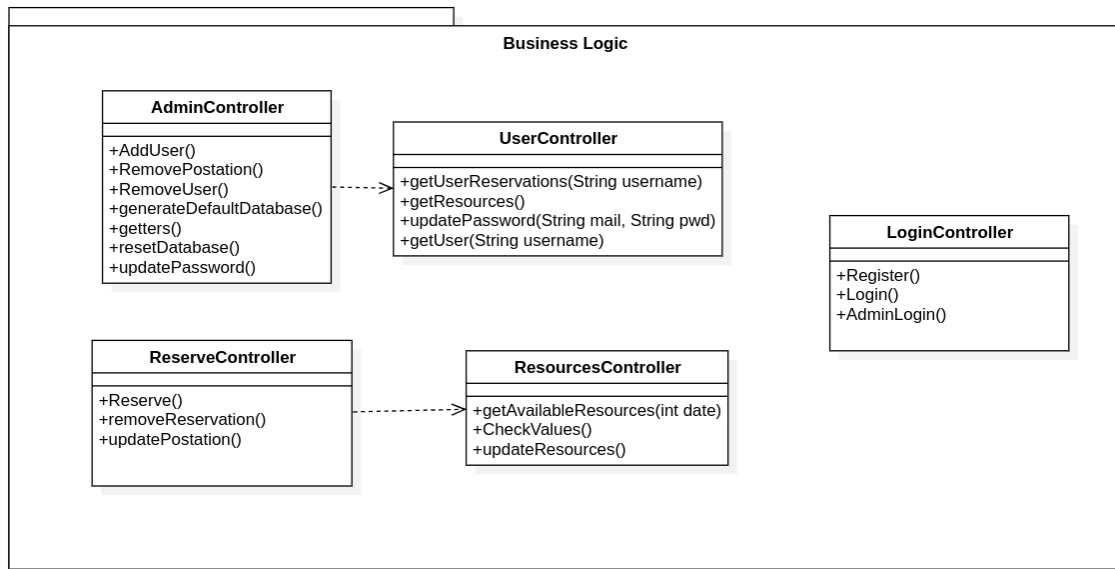


Figure 6: Class Diagram - Business Logic

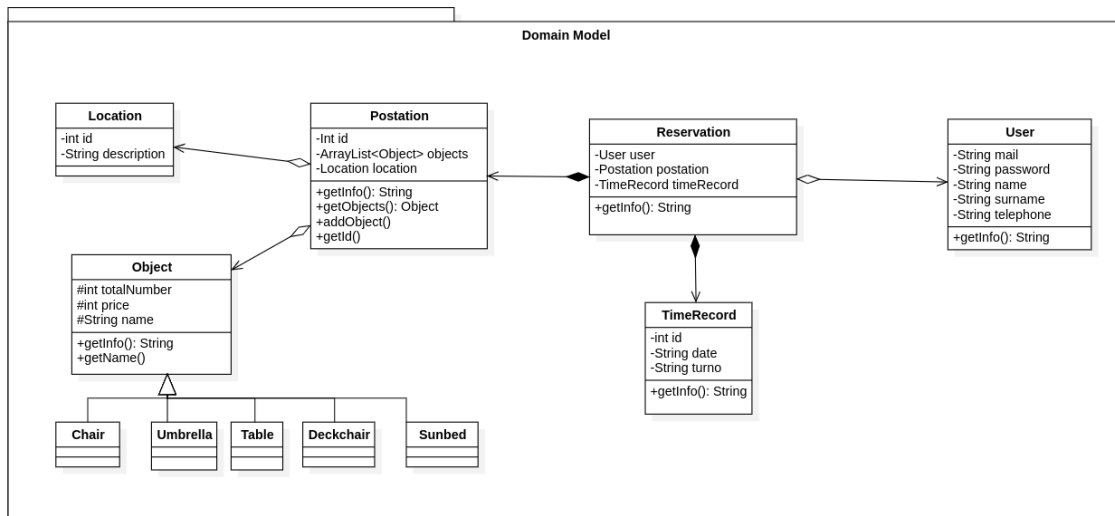


Figure 7: Class Diagram - Domain Model

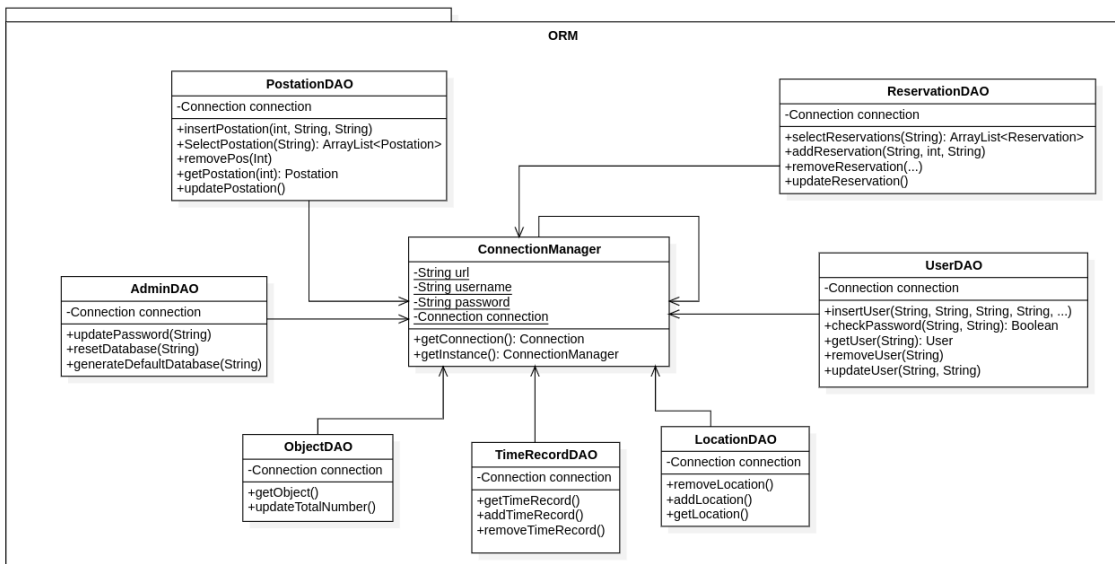


Figure 8: Class Diagram - ORM

2.5 ER Diagram e modello relazionale

Per quanto riguarda la progettazione del database (9), si hanno queste entità:

- **User:** rappresenta l'entità **User**.
- **Postation:** rappresenta l'entità **Postation**. L'ArrayList di oggetti che è presente all'interno della classe (7) è stato inserito inserendo come colonne ogni singolo possibile valore presente tra gli oggetti.
- **Reservation:** rappresenta l'entità **Reservation** e risolve la relazione tra **User**, **TimeRecord** e **Postation**.
- **TimeRecord:** rappresenta l'entità **TimeRecord**.
- **Location:** rappresenta l'entità **Location**
- **Object:** rappresenta l'entità **Object**. A discapito di valori nulli, sono stati inseriti come colonne tutti gli attributi possibili che ogni risorsa può avere.

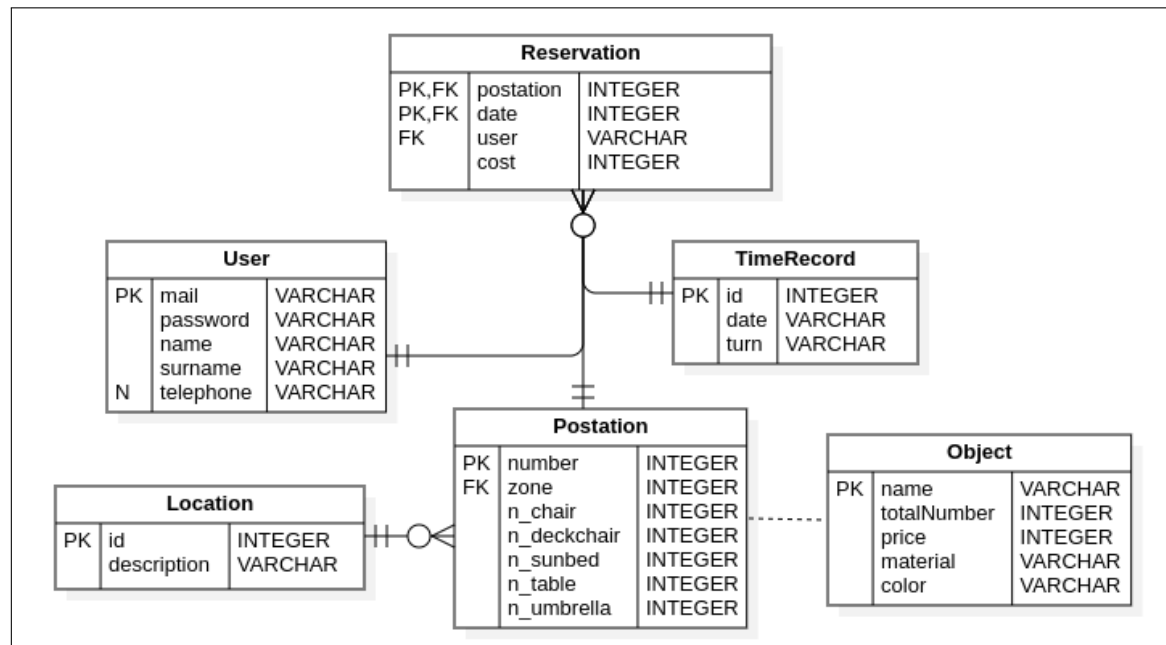


Figure 9: ER Diagram RAW view

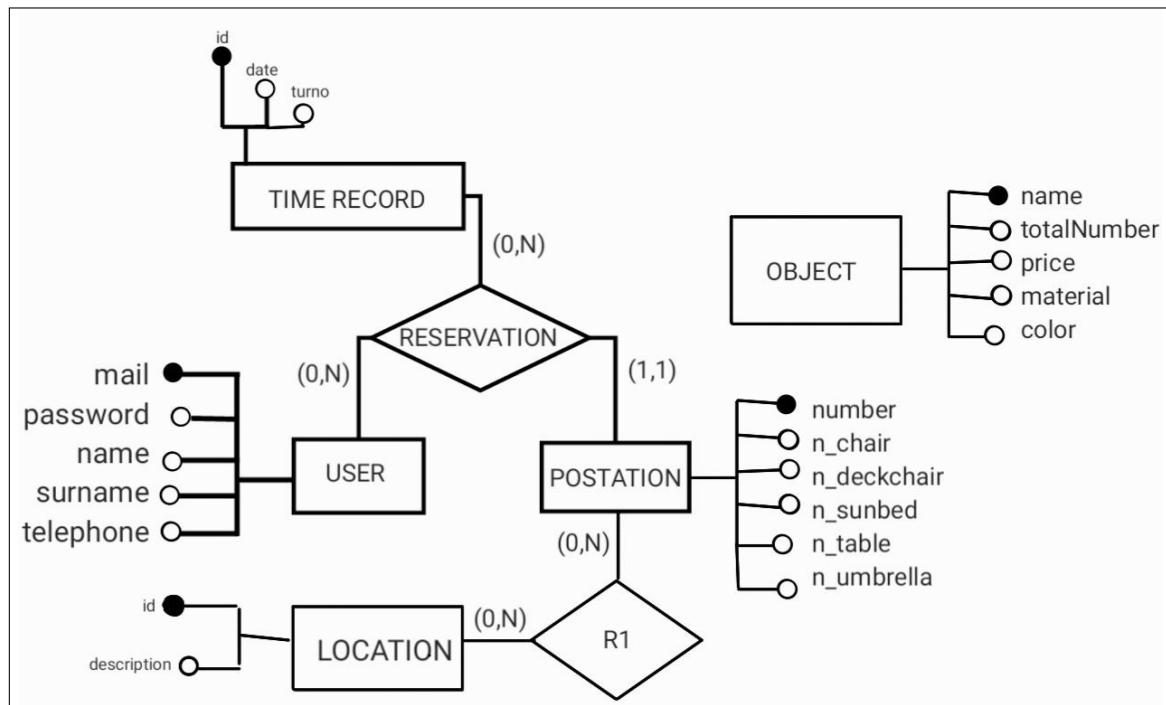


Figure 10: ER Diagram

2.6 Navigation Diagram

Il seguente diagramma (Figura 11) rappresenta quelle che sono le pagine principali del sistema e le possibili azioni che l'utente può compiere. Sono anche rappresentati i modi con cui si può navigare tra le varie pagine.

Alcune delle pagine sono:

- la pagina iniziale **PoolManager**: dove l'utente può registrarsi o effettuare il login.
- **User/Admin Reservation Actions**: in cui si può visualizzare prenotazioni, aggiungerne, rimuoverne o modificarne.
- **Event Editor**: la quale permette di creare un nuovo evento o di modificarne uno esistente.
- **User Profile Actions**: dove è possibile modificare i dati di accesso, cambiare dati personali o controllare i dati attuali.
- **Extra**: dalla quale l'admin può cambiare password o fare il reset del database.

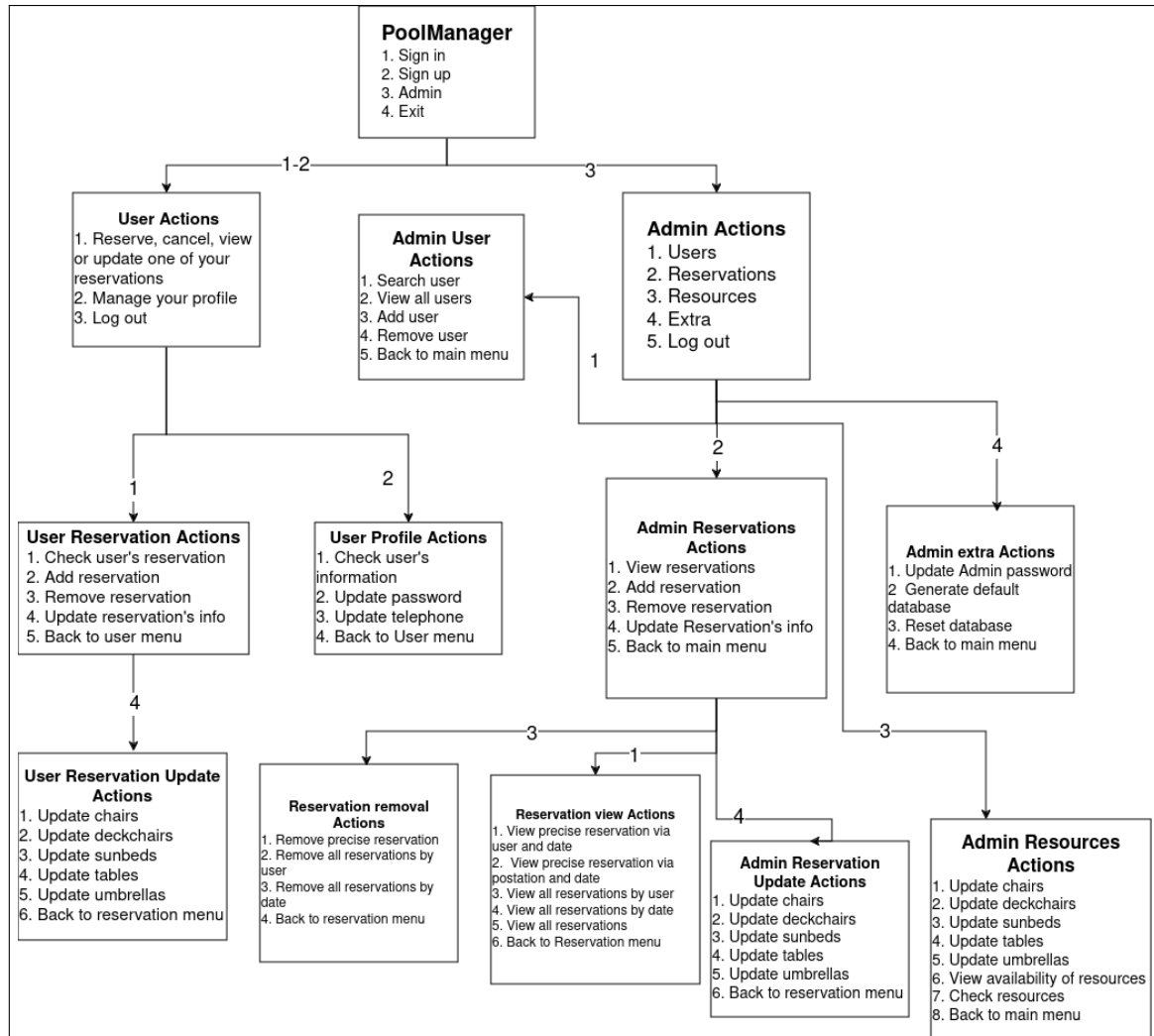


Figure 11: Navigation Diagram

3 Implementazione

L'implementazione dei 3 packages Domain Model, Business Logic e ORM e quella dell'interfaccia sono contenute in `src/main/java`, mentre i file relativi al database sono contenuti in `src/main/sql`.

3.1 Domain Model

Come già accennato nella sezione 2.4, nel package `main.java.DomainModel` (il percorso è: `src/main/java/DomainModel`) sono state implementate le classi che rappresentano le entità del sistema.

3.1.1 Object

Classe astratta che rappresenta le risorse, ovvero ogni singolo oggetto possibilmente prenotabile da un utente. I suoi attributi sono: *totalNumber*, *price* e *name*.

3.1.2 Chair

Specifico oggetto derivato dalla classe `Object`. I suoi attributi sono: *color*, *totalNumber*, *price* e *name*.

3.1.3 Deckchair

Specifico oggetto derivato dalla classe `Object`. I suoi attributi sono: *color*, *material*, *totalNumber*, *price* e *name*.

3.1.4 Sunbed

Specifico oggetto derivato dalla classe `Object`. I suoi attributi sono: *material*, *totalNumber*, *price* e *name*.

3.1.5 Table

Specifico oggetto derivato dalla classe `Object`. I suoi attributi sono: *color*, *material*, *totalNumber*, *price* e *name*.

3.1.6 Umbrella

Specifico oggetto derivato dalla classe `Object`. I suoi attributi sono: *color*, *totalNumber*, *price* e *name*.

3.1.7 TimeRecord

Classe che rappresenta il momento per cui viene fissata la prenotazione. I suoi attributi sono: *id*, *date*, *turn*. Dove *turn* è una stringa il cui valore può essere "mattina" o "pomeriggio".

3.1.8 Postation

Classe che rappresenta la postazione richiesta dall'utente nel momento della prenotazione. I suoi attributi sono: *id*, *objects* (un *ArrayList* di oggetti) e una *location*.

3.1.9 User

Un utente già registrato all'interno del sistema. I campi della classe sono: *mail*, *name*, *surname*, *pwd* e *telephone*.

3.1.10 Location

Classe che rappresenta le possibili scelte tra le zone della piscina. I suoi attributi sono: *id*, *description*.

3.1.11 Reservation

Classe che rappresenta la prenotazione. E' formata da: *postation*, *user* e un *timerecord*.

3.2 ORM

Come già accennato nella sezione 2.4, nel package `main.java.ORM` (il percorso è: `src/main/java/ORM`) sono state implementate le classi che si occupano dell'Object-Relational Mapping, ovvero delle operazioni di lettura e scrittura dei dati nel database. Ogni singola classe fuorché `ConnectionManager` permette alle classi del package `main.java.BusinessLogic` di accedere ai dati salvati nel database e possiede come attributo un'istanza di `ConnectionManager`.

3.2.1 UserDAO

Classe che si occupa della generazione di nuovi utenti attraverso il metodo `insertUser()`, di rimuoverne con `removeUser()`, di aggiornare le informazioni dell'utente e di selezionare utenti. Questa classe si occupa anche del controllo necessario al Login attraverso il metodo `checkPassword()`.

3.2.2 AdminDAO

Classe che si occupa delle Extra actions, ovvero: reset del database, generazione di un database di default e il cambio di password dell'admin.

3.2.3 ObjectDAO

Classe che si occupa della gestione dei dati che riguardano le risorse.

3.2.4 TimeRecordDAO

Classe che si occupa della gestione dei singoli `TimeRecord`, necessaria per calcolare dinamicamente il numero di risorse utilizzate in un dato `TimeRecord`.

3.2.5 LocationDAO

Classe che si occupa dell'inserimento e rimozione delle `Location`.

3.2.6 PostationDAO

Classe che si occupa della gestione delle postazioni, oltre ai metodi per aggiungere, rimuovere, aggiornare e selezionare, è presente un ulteriore metodo *getUsedResources()* necessario al calcolo dinamico delle risorse disponibili.

3.2.7 ReservationDAO

Classe che si occupa della gestione delle prenotazioni, in particolare si è voluto aggiungere diversi metodi di rimozione e selezione di modo da poter andare più o meno nello specifico, in base all'esigenza. E' ovviamente presente anche un metodo per aggiungere una *Reservation*, senza però alcun controllo sulle risorse, cosa che viene fatta dai controller.

3.2.8 ConnectionManager

La classe si occupa di gestire la connessione al database per le altre classi DAO tramite il metodo *getConnection()*. Contiene inoltre i dati di accesso al database, ovvero URL, username e password, che vengono utilizzati per stabilire la connessione. Per garantire che ci sia una sola istanza della classe in tutto il sistema e quindi per evitare conflitti tra connessioni, la classe è implementata come un *singleton*.

3.3 Business Logic

Come già accennato nella sezione 2.4, nel package `main.java.BusinessLogic` (il percorso è: `src/main/java/BusinessLo`) sono state implementate le classi che gestiscono la logica di business del sistema.

3.3.1 AdminController

Controller che gestisce le diverse operazioni che possono essere eseguite dall'Actor *Admin*. Si hanno quindi diversi strumenti di selezione per poter controllare tutte le prenotazioni e le postazioni. Inoltre sono qua presenti le Extra Actions. Per alcune operazioni, chiama *UserController* e ne utilizza i metodi.

3.3.2 UserController

Controller che gestisce le azioni dell'utente, come il controllo delle proprie prenotazioni e la gestione del proprio profilo.

3.3.3 ResourcesController

Controller che si occupa della gestione delle risorse e del calcolo della loro disponibilità. E' inoltre presente il metodo *CheckValues(...)* che restituisce un booleano necessario a sapere se il numero di risorse richieste è prenotabile o no.

3.3.4 ReserveController

Controller che si occupa della creazione, rimozione e modifica delle prenotazione. Chiama *ResourcesController* ogni volta che deve fare un accesso a delle risorse. Inoltre è stato qua inserito

il metodo di calcolo del costo di una prenotazione, che viene effettuato alla creazione della prenotazione e ogni volta che essa viene modificata.

3.4 Database

Come già accennato nella sezione 2.5, il database è stato progettato in modo da rispettare le specifiche del sistema. Sono state create le tabelle `Users`, `Location`, `Object`, `Reservation`, `Postation` e `TimeRecord` e sono state definite come indicato nella Figura 9. I file relativi al database sono contenuti in `src/main/sql` e i principali sono `reset.sql` e `default.sql`: il primo cancella i vecchi dati con il comando `DROP TABLE IF EXISTS` e ricrea le tabelle secondo la struttura sopra citata, mentre il secondo riempie le tabelle con dei dati di default per testare il funzionamento del sistema.

3.5 Interfaccia e CLI

Come introdotto nella sezione 1.2, è stata realizzata un'interfaccia a riga di comando per il sistema, implementata nel file `Main.java` (il percorso è: `src/main/java/Main.java`). L'utente può navigare tra le varie pagine e compiere le funzionalità del programma inserendo i vari comandi indicati dal sistema. Prendendo per esempio la pagina iniziale `PoolManager`, l'utente può scegliere di eseguire l'accesso inserendo il comando 1, di registrarsi inserendo il comando 2, di autenticarsi come admin inserendo il comando 3 o di uscire dal programma inserendo il comando 4. Ogni pagina è implementata con un *do-while* e uno *switch-case* per gestire le varie scelte dell'utente, mentre le varie funzionalità sono implementate con metodi specifici delle classi del package `main.java.BusinessLogic`. La navigazione tra le pagine avviene tramite chiamate agli stessi metodi della classe `Main`.

```

private static void handleLoginAction() throws ClassNotFoundException, SQLException{
    Scanner scanner = new Scanner(System.in);
    LoginController loginController = new LoginController();
    String input;

    do {

        System.out.println(
            """
            \s
            PoolManager
            1. Sign in
            2. Sign up
            3. Admin
            4. Exit
            \s"""
        );

        input = scanner.nextLine();

        switch (input) {
            case "1" -> {

                Scanner scanner1 = new Scanner(System.in);

                System.out.println("\nMail: ");
                String mail = scanner1.nextLine();
                System.out.println("Password: ");
                String password = scanner1.nextLine();

                User user = loginController.login(mail, password);

                if (user != null)
                    handleUserAction(user);
            }
        }
    }
}

```

Snippet: implementazione della pagina iniziale e della funzionalità di login (Sign in)