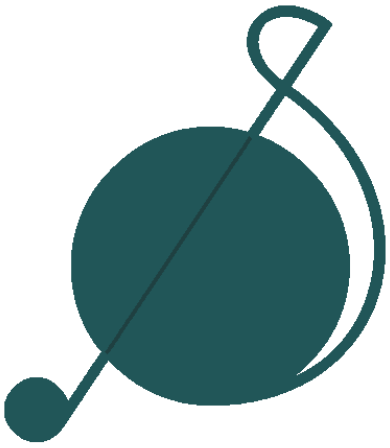
Laurea Triennale in informatica-Università di Salerno
Corso di
Ingegneria del software- Prof. Carmine Gravino



Spotibase

Object Design Document

Spotibase

Docente
Gravino Carmine

Studenti

Silvio Venturino	0512108837
Umberto Della Monica	0512107742
Gagliarde Nicolapio	0512106980
Alessandro Aquino	0512106527

Revision History

Data	Versione	Descrizione	Autori
15/12/2021	0.1	Prima stesura dell'introduzione, della sezione packages e della sezione class interfaces	Tutto il team

Team Members

Nome	Ruolo nel progetto	Email	Acronimo
Alessandro Aquino	Team member	a.aquino33@studenti.unisa.it	AA
Nicolapio Gagliarde	Team member	n.gagliarde@studenti.unisa.it	NG
Umberto Della Monica	Team member	u.dellamonica2@studenti.unisa.it	UM
Silvio Venturino	Team member	s.venturino@studenti.unisa.it	SV

Sommario

1. Introduzione

1.1 Object Design goals

Durante la fase di analisi e di progettazione del sistema abbiamo individuato diversi compromessi per lo sviluppo del sistema. Anche durante la fase di Object Design sorgono diversi compromessi che andremo ad analizzare in questo paragrafo:

Performance:

Il sistema deve garantire tempi di risposta minori di 1 secondo a discapito della memoria utilizzata, come specificato dai requisiti non funzionali (RAD_Spotibase - capitolo:3 - paragrafo:3.3.3) e dai trade off individuati nella fase di System Design (SDD_Spotibase - capitolo:1 - paragrafo:1.2.1). Poiché deve essere inviata una o più risposte ad ogni utente in base al servizio richiesto. Questo verrà ampliato nei design patterns.

Affidabilità:

Il sistema deve risultare robusto, reagendo correttamente a situazioni impreviste attraverso il controllo degli errori e la gestione delle eccezioni. Inoltre il sistema deve garantire un corretto inserimento di dati in input per eventuali accessi minacciosi che non comportino una perdita di dati sensibili (e-mail o password) o una modifica di essi. Concetti espressi nei requisiti non funzionali (RAD_Spotibase - capitolo:3 - paragrafo:3.3.2).

Modificabilità:

Il sistema deve essere scalabile, e facilmente adattabile a nuove funzionalità del dominio applicativo, inoltre deve essere inoltre fortemente modulare in modo da riuscire a modificare la logica di business o quella di sistema senza particolari difficoltà. Concetti espressi nei requisiti non funzionali (RAD_Spotibase - capitolo:3 - paragrafo:3.3.4). Il sistema amplierà il concetto mediante i design patterns.

1.2 Object Trade-Off

Buy vs build:

Abbiamo preferito costruire invece che acquistare. La scelta è stata presa per mancanza di budget economico a differenza del budget di tempo che ci ha permesso di implementare i requisiti richiesti.

Portabilità VS Efficienza:

Abbiamo garantito la portabilità, utilizzando un linguaggio non nativo: Java è indipendente dalla piattaforma, purtroppo questa caratteristica comporta prestazioni inferiori in quanto non viene compilato in linguaggio macchina.

Throughput VS Sicurezza:

Non avendo usato SSL (crittografia), abbiamo aumentato la potenza di Throughput

1.3 Linee Guida per la Documentazione delle Interfacce

In questo paragrafo verranno definite le linee guida a cui un programmatore deve attenersi nell'implementazione del sistema.
In ciascun sotto paragrafo, quando si parla di indentazione ci si riferisce ad una tabulazione.

1.3.1 Classi e Interfacce Java

Le linee guida che abbiamo seguito per le classi ed interfaccia java sono le seguenti:

1. Il nome delle classi è al singolare
2. Il nome della classe è con la lettera maiuscola
3. Il nome della classe e delle variabili rispetta la camelStyle
4. Il nome dei metodi è con la lettera minuscola

1.3.2 Java Servlet Pages (JSP)

Le JSP costruiscono pagine HTML in maniera dinamica che devono rispettare il formato definito nel sotto paragrafo sottostante. Devono anche attenersi alle linee guida per le classi Java definito nel sotto paragrafo soprastante. Inoltre, le JSP devono attenersi alle seguenti puntualizzazioni:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga che non contiene nient'altro;
3. Istruzioni Java che consistono in una sola riga possono contenere il tag di apertura e chiusura sulla stessa riga;

1.3.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5. Inoltre, il codice HTML deve utilizzare l'indentazione per facilitare la lettura e di conseguenza la manutenzione. Le regole di indentazione sono le seguenti:

1. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
2. Ogni tag di chiusura deve avere lo stesso livello di indentazione di quello di apertura;

1.3.4 File JavaScript

Gli script JavaScript verranno scritti nella cartella apposita, separando i file in base alla funzione degli script.
I nomi delle funzioni iniziano con una lettera maiuscola.
I nomi delle variabili rispettano il camelStyle.

1.3.5 Fogli di stile CSS

Ogni regola CSS deve essere formattata nel seguente modo:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.3.6 Script SQL

Le istruzioni e le clausole SQL devono essere costituite da sole lettere maiuscole.

1.4 Definizioni, acronimi e abbreviazioni

Vengono riportati di seguito alcune definizioni presenti nel documento:
Package: raggruppamento di classi, interfacce o file correlati;
Design pattern: template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
Interfaccia: insieme di signature delle operazioni offerte dalla classe;
Javadoc: sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia grafica in modo da rendere la documentazione accessibile e facilmente leggibile.

1.5 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

- **Requirements Analysis Document;**
- **System Design Document;**
- **Test Plan;**
- **Matrice di tracciabilità;**

Libri:

- Object-Oriented Software Engineering (Using UML, Patterns, and Java) Third Edition

2. Packages

In questa sezione viene mostrata la suddivisione del sistema in package.

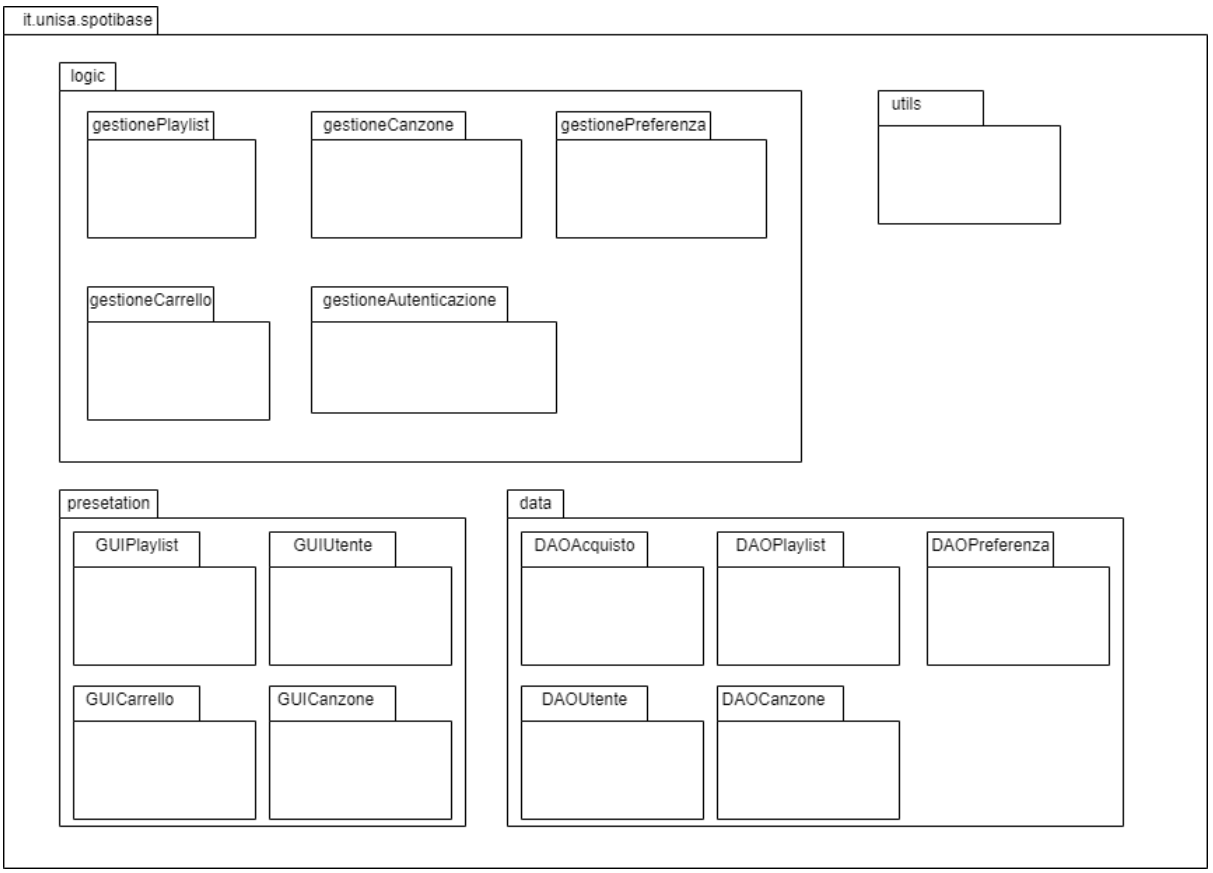
- **.idea**
- **docs**
    - **jacoco**
    - **javadoc**
- **src**: contiene tutti i file sorgente
    - **main**
        - **java:** contiene le classi Java relative alle componenti logic Tier e Data tier
        - **webapp:** contiene i fogli di stile CSS, pagine di errore HTML e gli script JS
            - **WEB-INF:** contiene i file JSP relativi alle componenti Presentation Tier
    - **test:** contiene tutto il necessario per il testing
        - **java:** contiene le classi Java per l'implementazione del testing
- **target:** contiene tutti i file prodotti dal sistema di build

Package Spotibase

Nella presente sezione si mostra la struttura del package principale di Spotibase. La struttura prevede tre package che rispecchiano la struttura three tier:

- package logic: contiene i sottosistemi per gestire le canzoni, le preferenze, le playlist, il carrello e l'autenticazione;
- package presentation: contiene i package per gestire l'interfaccia grafica;
- package data: contiene i DAO, le interfacce dei dao ed i pojo per gestire i dati da salvare nel database.

Inoltre si prevede la presenza del package "utils" per contenere classi software che possono essere utilizzate dagli altri componenti del sistema.



Package logic tier

Package Gestione Autenticazione

- Login
- Registrazione

Package Gestione Canzone

- JsonCanzoneServlet

Package Gestione Playlist

- JsonPlaylistServlet
- ServletPlaylist

Package Gestione Carrello

- ServletCarrello
- JsonCarrelloServlet

Package Gestione Preferenza

- JsonPreferitiServlet

Package Presentation Tier

Package Gui Utente

- **register.jsp**
- **login.jsp**
- **index.jsp**

Package Gui Canzone

- **canzone.jsp**

Package Gui Playlist

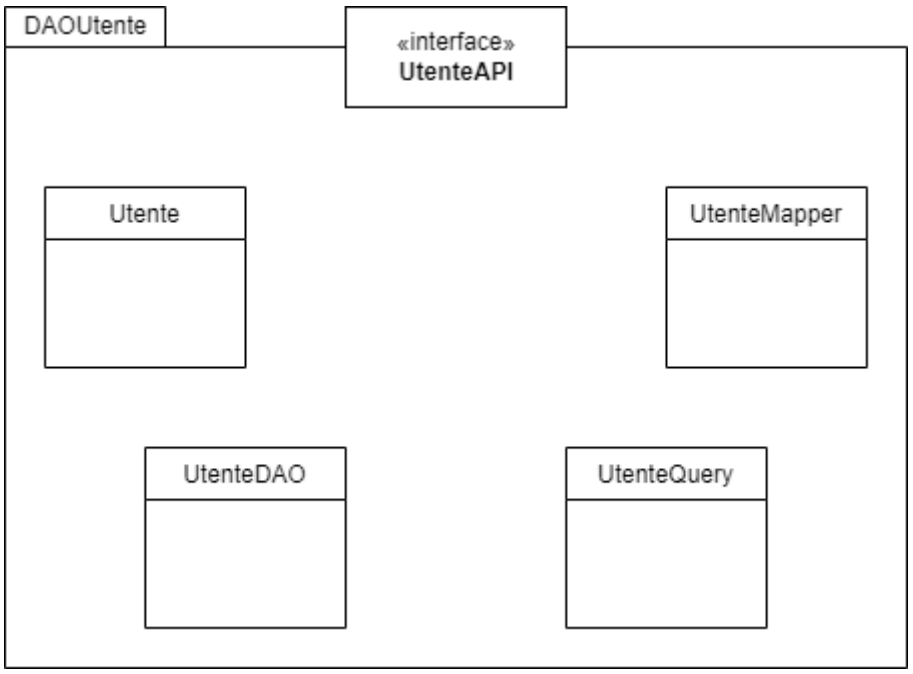
- **playlist.jsp**

Package Gui Carrello

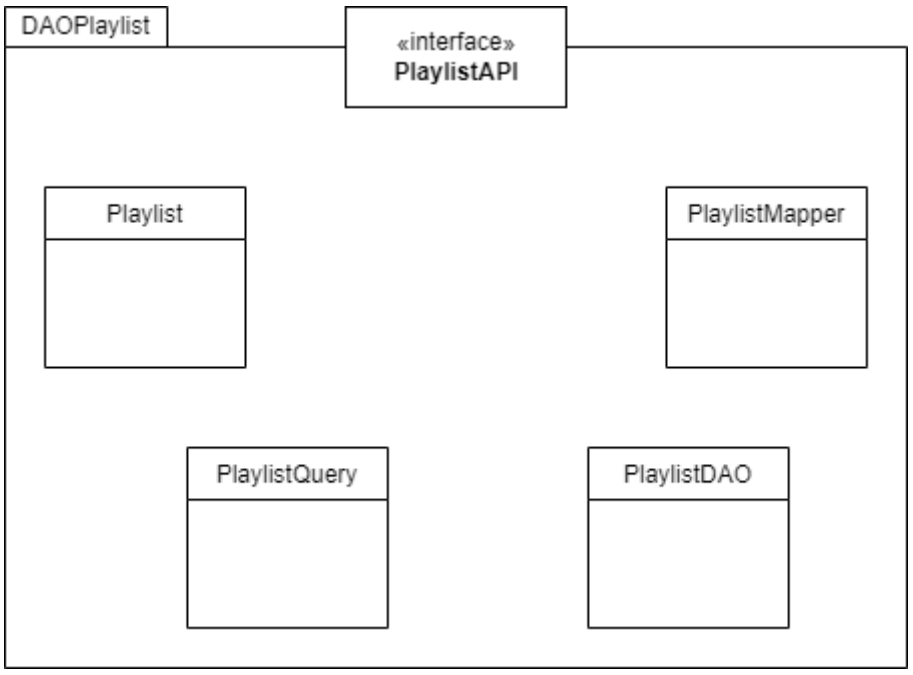
- **carrello.jsp**

Package del Data Tier

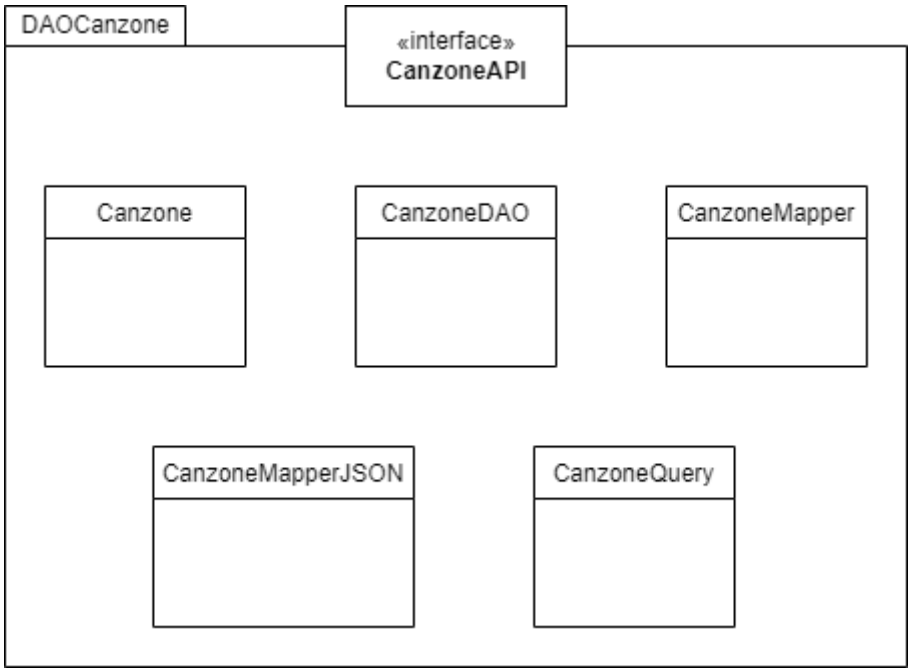
Package DAOUtente



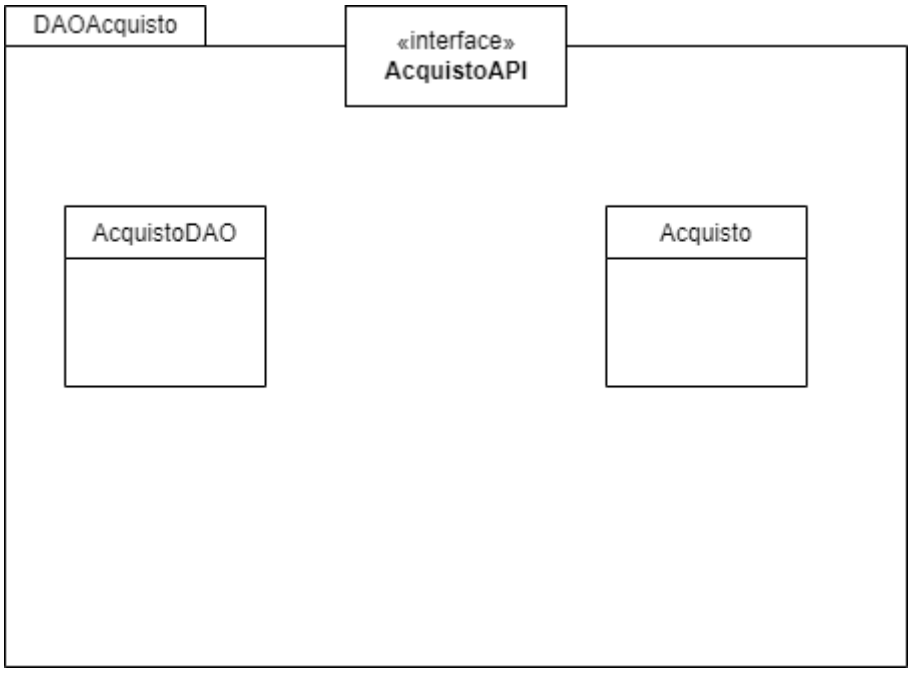
Package DAOPlaylist



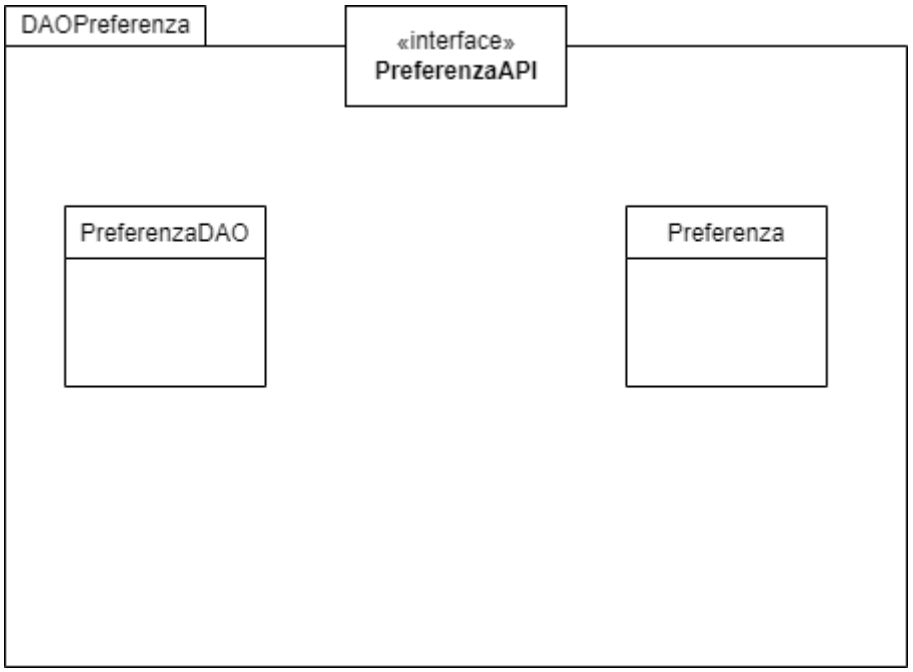
Package DAOCanzone



Package DAOAcquisto



Package DAOPreferenza



3. Class interfaces

Ogni sottopackage del package "data", contiene una classe Java che rappresenta un contenitore di dati, quindi avrà i campi privati con i relativi metodi getter e setter. Una classe DAO, che avrà i metodi per gestire i dati persistenti. Una classe Mapper, che conterrà i metodi che riceveranno il result set dal database e ritorneranno dei pojo.
Inoltre può essere presente una classe MapperJson, che conterrà i metodi che riceveranno il result set e ritorneranno degli oggetti Json. Le classi Mapper e MapperJson consentono il riuso del codice.
Di seguito saranno riportate le interfacce di ciascuna classe, dividendole per package (il package presentation non sarà riportato).

Package DAOUtente

it.unisa.spotibase.data.DAOUtente

Utente

Nome classe	Utente
Descrizione	Questa classe rappresenta il pojo per contenere i dati dell'utente
Campi	-username:String -email:String -passwd:String
Metodi	Metodi getter e setter seguendo lo standard: getNomeCampo(), setNomeCampo(TipoCampo)
Invariante di classe	/

UtenteDAO

Nome classe	UtenteDAO
Descrizione	Questa classe permette di gestire le operazioni relative ai dati persistenti dell'utente.
Metodi	+**doGet**(email: String, password: String): Utente +**findUsers**(field: String, value String): List <Utente> +**doGet**(chiave: String): Utente +**doDelete**(chiave: String): void +**doSave**(oggetto: Utente): void +**exist**(chiave: String):boolean +**isValidEmail**(email:String): boolean +**isValidPasswd**(passwd:String): boolean **+isValidUsername(**username: String**):**boolean
Invariante di classe	/

Nome Metodo	+doGet(email: String, password: String): Utente
Descrizione	Questo metodo consente di prelevare un utente dal database data l'email e la password
Pre-condizione	**context:** UtenteDAO::doGet(email,password) **pre:** email != null && password != null && UtenteDAO.exist(chiave)==true
Post-condizione	/

Nome Metodo	+findUsers(field: String, value: String): List <Utente>
Descrizione	Questo metodo consente di prelevare una lista di utenti, dato un field che sia uguale a value.
Pre-condizione	**context:** UtenteDAO::findUsers(field, value) **pre:** field != null && value != null
Post-condizione	/

Nome Metodo	+doGet(chiave: String): Utente
Descrizione	Questo metodo consente di prelevare un utente, data una chiave
Pre-condizione	**context:** UtenteDAO::doGet(chiave) **pre:** chiave != null && chiave.contains(“;”) == true && UtenteDAO.exist(chiave)==true
Post-condizione	/

Nome Metodo	+doDelete(chiave: String): void
Descrizione	Questo metodo consente di eliminare un utente, data una chiave
Pre-condizione	**context:** UtenteDAO::doDelete(chiave) **pre:** chiave != null && UtenteDAO.exist(chiave)==true
Post-condizione	**context:** UtenteDAO::doDelete(chiave) **post:** UtenteDAO.exist(chiave)==false

Nome Metodo	+doSave(oggetto: Utente): void
Descrizione	Questo metodo consente di salvare un utente
Pre-condizione	**context:** UtenteDAO::doSave(oggetto) **pre:** oggetto!= null && UtenteDAO.exist(chiave) == false && oggetto.getEmail() != null && oggetto.getUsername() != null && oggetto.getPasswd()!= null
Post-condizione	**context:**UtenteDAO::doSave(oggetto) **post:** UtenteDAO.exist(chiave)==true

Nome Metodo	+isValidEmail(email:String):boolean
Descrizione	Questo metodo consente di validare l'email dell'utente
Pre-condizione	**context:** UtenteDAO::isValidEmail(email) **pre:** email != null
Post-condizione	\

Nome Metodo	+isValidPasswd(passwd:String):boolean
Descrizione	Questo metodo consente di validare la password dell'utente
Pre-condizione	**context:** UtenteDAO::isValidPasswd(passwd) **pre:** passwd!= null
Post-condizione	/

Nome Metodo	+**isValidUsername(**username: String**)**:boolean
Descrizione	Questo metodo consente di validare la username dell'utente
Pre-condizione	**context:** UtenteDAO::**isValidUsername(**username) **pre:** username!= null
Post-condizione	/

Nome Metodo	+exist(chiave:String):boolean
Descrizione	Questo metodo consente di verificare la presenza dell'utente all'interno del DB
Pre-condizione	**context:** UtenteDAO::exist(chiave) **pre:** chiave!= null
Post-condizione	\

UtenteMapper

Nome classe	UtenteMapper
Descrizione	Questa classe permette ricavare un utente da un result set proveniente dal database
Metodi	+map(ResultSet rs):Utente
Invariante di classe	/

Nome Metodo	+map(ResultSet rs):Utente
Descrizione	Questo metodo consente di ricavare un utente da un insieme di tuple ritornate dal db
Pre-condizione	**context:** UtenteMapper::map(rs) **pre:** rs.getString("Username")!=null && rs.getString("Passwd")!=null
Post-condizione	/

Package DAOCanzone
it.unisa.spotibase.data.DAOCanzone

Canzone

Nome classe	Canzone
Descrizione	Questa classe rappresenta il pojo per contenere i dati della canzone
Campi	-id -titolo -prezzo -pathImage -pathMP3
Metodi	Metodi getter e setter seguendo lo standard: getNomeCampo(), setNomeCampo(TipoCampo)
Invariante di classe	/

CanzoneDAO

Nome classe	CanzoneDAO
Descrizione	Questa classe permette di gestire le operazioni relative ai dati persistenti della canzone.
Metodi	+**doRetrieveCanzoneWithArtisti**(codiceCanzone: String): Canzone +**doDelete**(chiave: String): void +**doSave**(oggetto: Canzone): void +**doGet**(chiave: String): Canzone +**exist**(chiave:String):boolean
Invariante di classe	/

Nome Metodo	+**doRetrieveCanzoneWithArtisti** (codiceCanzone: String): Canzone
Descrizione	Questo metodo consente di prelevare una Canzone dal database con gli artisti
Pre-condizione	**context** CanzoneDAO::doRetrieveCanzoneWithArtisti(codiceCanzone) **pre:** codiceCanzone != null && CanzoneDAO.exist(Canzone.getId())==true
Post-condizione	/

Nome Metodo	+**doDelete**(chiave: String): void
Descrizione	Questo metodo consente di eliminare una Canzone dal database
Pre-condizione	**context** CanzoneDAO::doDelete(chiave) **pre:** chiave!=null && CanzoneDAO.exist(chiave)==true
Post-condizione	**context:** CanzoneDAO::doDelete(chiave) **post :** CanzoneDAO.exist(chiave) == false

Nome Metodo	+**doSave**(oggetto: Canzone): void
Descrizione	Questo metodo consente di salvare una Canzone dal database
Pre-condizione	**context** CanzoneDAO::doSave(oggetto) **pre:** oggetto != null && CanzoneDAO.exist(oggetto.codice) == false && Canzone.getCodice() != null && Canzone.getTitolo() != null
Post-condizione	**context :** CanzoneDAO::doSave(oggetto) **post:** CanzoneDAO.exist(oggetto.codice)==true

Nome Metodo	+**doGet**(chiave: String): Canzone
Descrizione	Questo metodo consente di prelevare una Canzone dal database
Pre-condizione	**context** CanzoneDAO::doGet(chiave) **pre:** chiave!=null && exist(chiave) == true
Post-condizione	/

Nome Metodo	+**exist**(chiave:String):boolean
Descrizione	Questo metodo consente di verificare la presenza della canzone all'interno del DB
Pre-condizione	**context:** UtenteDAO::exist(chiave) **pre:** chiave!= null
Post-condizione	\

CanzoneMapper

Nome classe	CanzoneMapper
Descrizione	Questa classe permette ricavare una canzone da un result set proveniente dal database
Metodi	+map(ResultSet rs):Canzone
Invariante di classe	/

Nome Metodo	+**map(ResultSet rs)**:Canzone
Descrizione	Questo metodo consente di ricavare una canzone da un insieme di tuple ritornate dal db
Pre-condizione	**context:** CanzoneMapper::map(rs) **pre:** rs.getString(“id”)!=null && rs.getString(“titolo”)!=null && rs.getString(“prezzo”)!=null && rs.getString(“pathImg”)!=null && rs.getString(“pathMP3”)!=null
Post-condizione	/

Package DAOPlaylist
it.unisa.spotibase.data.DAOPlaylist

Playlist

Nome classe	Playlist
Descrizione	Questa classe rappresenta il pojo per contenere i dati della playlist
Campi	-nome -note
Metodi	Metodi getter e setter seguendo lo standard: getNomeCampo(), setNomeCampo(TipoCampo)
Invariante di classe	/

PlaylistDAO

Nome classe	PlaylistDAO
Descrizione	Questa classe permette di gestire le operazioni relative ai dati persistenti della playlist.
Metodi	+**DoRetrievePlaylistByUtente**(username:String,utenteAPI:UtenteApi):List<Playlist> +**doInsertSong**(username:String, titoloPlay:String, codCanzone:String,utenteApi:UtenteAPI,canzoneApi:CanzoneAPI,playlistApi:PlaylistAPI):void +**isPresent**(codiceCanzone:String ,titolo :String, username:String,utenteApi:UtenteAPI,canzoneApi:CanzoneAPI,playlistApi:PlaylistAPI):boolean + **isPresent**(titolo:String,username:String,utenteApi:UtenteAPI):boolean + **doRetrieveNumPlaylistOfUtente**(username:String,utenteApi:UtenteAPI):int + **doDelete**(chiave:String):void + **doGet**(chiave:String):Playlist + **doSave**(oggetto:Playlist): void +**isValidNota**(Nota:String):boolean +**isValidTitolo**(titolo:String):boolean
Invariante di classe	/

Nome Metodo	+**doRetrievePlayListByUtente**(username:String,utenteApi: UtenteAPI): List<Playlist>
Descrizione	Questo metodo consente di prelevare una Playlist dal database
Pre-condizione	**context:** PlaylistDAO::doRetrievePlayListByUtente(username) **pre:** username != null && utenteApi.findusers(username).size!=0
Post-condizione	/

Nome Metodo	+**doInsertSong**(username:String, titoloPlay:String, codCanzone:String,utenteApi:UtenteAPI,canzoneApi:CanzoneAPI ,playlistApi:PlaylistAPI):void
Descrizione	Questo metodo consente di inserire una nuova canzone in PlayList
Pre-condizione	**context:** PlaylistDAO::doInsertSong(username,titotloPlay,codCanzone, utenteApi,canzoneApi,playlistApi) **pre:** username!=null && titoloPlay != null && codCanzone != null && UtenteDAO.exist(username)==true && CanzoneDAO.exist(codCanzone)==true PlayListDAO.isPresent(titoloPlay,username)==true
Post-condizione	**context**: PlaylistDAO::doInsertSong(username,titotloPlay,codCanzone) **post**: PlaylistDAO::isPresent(username,titotloPlay,codCanzone) == true

Nome Metodo	+**isPresent**(codiceCanzone:String ,titolo :String, username:String,utenteApi:UtenteApi,canzoneApi:CanzoneApi,playlis tApi:PlaylistApi):boolean
Descrizione	Questo metodo consente di verificare se la canzone è presente nella playlist
Pre-condizione	**context:** PlaylistDAO::isPresent(username,titolo, codCanzone) **pre:** username!=null && titolo!=null && codCanzone!=null && UtenteDAO.exist(username)==true && CanzoneDAO.exist(codCanzone)==true && PlaylistDAO.isPresent(titolo,username) == true
Post-condizione	/

Nome Metodo	+ **isPresent**(titolo:String , username:String,utenteApi:UtenteApi):boolean
Descrizione	Questo metodo consente di verificare se la playlist è presente nel database
Pre-condizione	**context:** PlaylistDAO:: isPresent(titolo,username) **pre:** username!=null && titolo!=null && UtenteDAO.exist(username) == true
Post-condizione	/

Nome Metodo	+**doRetrieveNumPlaylistOfUtente**(username:String,utenteApi: UtenteApi):int
Descrizione	Questo metodo consente di prelevare il numero di PlayList di un utente
Pre-condizione	**context:** PlaylistDAO::doRetrieveNumPlayListOfUtente(username) **pre:** username != null && UtenteDAO.exist(username) == true
Post-condizione	/

Nome Metodo	+ **doDelete**(chiave:String):void
Descrizione	Questo metodo consente di eliminare la PlayList dato il suo nome
Pre-condizione	**context:** PlaylistDAO::doDelete(chiave) **pre:** chiave!=null && chiave.contains(“;”) == true && PlaylistDAO.isPresent(chiave) == true
Post-condizione	**context:** PlaylistDAO::doDelete(chiave) **post:** PlaylistDAO.isPresent(chiave) == false

Nome Metodo	+ **doGet**(chiave:String):Playlist
Descrizione	Questo metodo consente di ottenere una PlayList dato il suo nome
Pre-condizione	**context:** PlaylistDAO::doGet(chiave) **pre:** chiave!=null && chiave.contains(“;”) == true && PlaylistDAO.isPresent(oggetto.getTitolo(), oggetto.getUsername()) != null
Post-condizione	/

Nome Metodo	+**isValidNota**(nota:String):boolean
Descrizione	Questo metodo consente di validare la nota della Playlist
Pre-condizione	**context:** UtenteDAO::isValidNota(nota) **pre:** nota!= null
Post-condizione	\

Nome Metodo	+**isValidTitolo**(titolo:String):boolean
Descrizione	Questo metodo consente il titolo della playlist
Pre-condizione	**context:** UtenteDAO::isValidTitolo(titolo) **pre:** titolo!= null
Post-condizione	\

Nome Metodo	+ **doSave**(oggetto:PlayList):void
Descrizione	Questo metodo consente di salvare la PlayList all'interno del database
Pre-condizione	**context:** PlaylistDAO::doSave(oggetto) **pre:** oggetto!=null && oggetto.getTitolo() != null && oggetto.getUsername() != null && PlaylistDAO.isPresent(oggetto.getTitolo(), oggetto.getUsername()) == false
Post-condizione	**context:** PlaylistDAO::doSave(oggetto) **post:** PlaylistDAO.isPresent(oggetto.getTitolo(), oggetto,getUsername()) ==true

PlaylistMapper

Nome classe	PlaylistMapper
Descrizione	Questa classe permette ricavare una playlist da un result set proveniente dal database
Metodi	+map(ResultSet rs):Playlist
Invariante di classe	/

Nome Metodo	+**map**(ResultSet rs):Playlist
Descrizione	Questo metodo consente di ricavare una playlist da un insieme di tuple ritornate dal db
Pre-condizione	**context:** PlaylistMapper::map(rs) **pre:** rs.getString("nome")!=null
Post-condizione	/

Package DAOPrefernza

it.unisa.spotibase.data.DAOPreferenza

Preferenza

Nome classe	Preferenza
Descrizione	Questa classe rappresenta l'oggetto per contenere i dati delle preferenze
Campi	-username -codCanzone
Metodi	Metodi getter e setter seguendo lo standard: getNomeCampo(), setNomeCampo(TipoCampo)
Invariante di classe	./

PreferenzaDAO

Nome classe	PreferenzaDAO
Descrizione	Questa classe permette di gestire le operazioni relative ai dati persistenti della preferenza.
Metodi	+ **doRetrieveCodiciCanzoniPreferite**(username:String):List<String> + **doGet(** chiave:String):Preferenza + **doSave**(oggetto:Preferenza) :void + **doDelete**(chiave:String):void + **exist**(codCanzone:String,username:String): boolean
Invariante di classe	/

Nome Metodo	+ **doRetrieveCodiciCanzoniPreferite**(username:String):List<String>
Descrizione	Questo metodo consente di prelevare i codici delle canzoni preferite dall'utente grazie al suo username
Pre-condizione	**context** PreferenzaDAO::**doRetrieveCodiciCanzoniPreferite**(username) **pre:** username!=null && UtenteDAO.exist(username)==true
Post-condizione	/

Nome Metodo	+**doGet(** chiave:String):Preferenza
Descrizione	Questo metodo consente di ottenere una preferenza dal database
Pre-condizione	**context:** PreferenzaDAO:**doGet(** chiave:String):Preferenza **pre:** chiave!=null && chiave.contains(";") && PreferenzaDAO.exist(chiave) == true
Post-condizione	/

Nome Metodo	+ **doSave**(oggetto:Preferenza) :void
Descrizione	Questo metodo consente di salvare una preferenza all'interno del database
Pre-condizione	**context:** PreferenzaDAO:**doSave**(oggetto:Preferenza) :void **pre:** oggetto!=null && oggetto.getCodCanzone()!= null && oggetto.getCodUtente() != null && PreferenzaDAO.exist(chiave) == false
Post-condizione	**context:** PreferenzaDAO:**doSave**(oggetto:Preferenza) :void **post:** PreferenzaDAO.exist(chiave) == true

Nome Metodo	+ **doDelete**(chiave:String):void
Descrizione	Questo metodo consente di cancellare una preferenza dal database
Pre-condizione	**context:** PreferenzaDAO:**doDelete(** chiave:String):boolean **pre:** chiave!=null && PreferenzaDAO.exist(chiave) == true && chiave.contains(“;”) == true
Post-condizione	**context:** PreferenzaDAO:**doDelete(** chiave:String):void **post:** PreferenzaDAO.exist(chiave) == false

Nome Metodo	+**exist**(codCanzone:String,username:String): boolean
Descrizione	Questo metodo consente di verificare la preferenza all’interno del database
Pre-condizione	**context** AcquistoDAO:: exist(codCanzone,username) **pre:** codCanzone!=null && username!=null
Post-condizione	/

Package DAOAcquisto

it.unisa.spotibase.data.DAOAcquisto

Acquisto

Nome classe	Acquisto
Descrizione	Questa classe rappresenta il pojo per contenere i dati relativi ad un acquisto
Campi	-username -codCanzone
Metodi	Metodi getter e setter seguendo lo standard: getNomeCampo(), setNomeCampo(TipoCampo)
Invariante di classe	/

AcquistoDAO

Nome classe	AcquistoDAO
Descrizione	Questa classe permette di gestire le operazioni relative ai dati persistenti degli acquisti.
Metodi	+ **doInsertCanzoneAcquistata**(username:String, codice:String):void +**doRetrieveCodiciCanzoniAcquistate**(username:String):List<String> + **doGet**(chiave:String):Acquisto + **doSave**(oggetto:Acquisto) :void + **doDelete**(chiave:String):void + **exist**(username:String,codCanzone): boolean
Invariante di classe	/

Nome Metodo	+**doInsertCanzoneAcquistata**(username:String, codice:String):void
Descrizione	Questo metodo consente di inserire la canzone acquistata all'interno del Carrello
Pre-condizione	**context** AcquistoDAO::**doInsertCanzoneAcquistata**(username:String, codice:String):void **pre:** codice!=null && username != null && AcquistoDAO.exist(username,codice) == null
Post-condizione	**context**:AcquistoDAO::**doInsertCanzoneAcquistata**(username:String, codice:String):void **post:** AcquistoDAO.exist(username,codice) == true

Nome Metodo	+ **doRetrieveCodiciCanzoniAcquistate**(String username):List<String>
Descrizione	Questo metodo consente di ottenere i codici delle canzoni acquistate mediante l'utilizzo dell'username utente
Pre-condizione	**context** AcquistoDAO:: **doRetrieveCodiciCanzoniAcquistate**(String username):List<String> **pre:** username!=null && UtenteDAO.exist(username) == true
Post-condizione	/

Nome Metodo	+ **doGet**(String chiave):Acquisto
Descrizione	Questo metodo consente di ottenere l'acquisto effettuato dall'utente mediante l'id dell'acquisto
Pre-condizione	**context** AcquistoDAO:: **doGet**(String chiave):Acquisto **pre:** chiave!=null && chiave.contains(";") == true && AcquistoDAO.exist(chiave) == true
Post-condizione	/

Nome Metodo	+ **doSave**(Acquisto oggetto) :void
Descrizione	Questo metodo consente di salvare l'acquisto fatto dall'utente all'interno del database
Pre-condizione	**context** AcquistoDAO:: **doSave**(Acquisto oggetto) :void **pre:** oggetto!=null && oggetto.getCodCanzone() != null && oggetto.getUsename() != null && AcquistoDAO.exist(chiave) == false
Post-Condizione	**context** AcquistoDAO:: **doSave**(Acquisto oggetto) :void **post:** AcquistoDAO.exist(chiave) == true

Nome Metodo	+ **doDelete**(String chiave):void
Descrizione	Questo metodo consente di eliminare l'acquisto fatto dall'utente all'interno del database
Pre-condizione	**context** AcquistoDAO:: **doDelete**(String chiave):void **pre:**chiave!=null && chiave.contains(";") == true && AcquistoDAO.exist(chiave) == true
Post-Condizione	**context** AcquistoDAO:: **doDelete**(Acquisto oggetto) :void **post:** AcquistoDAO.exist(chiave) == false

Nome Metodo	+ **exist**(codCanzone:String, username: String): boolean
Descrizione	Questo metodo consente di verificare l'acquisto all'interno del database
Pre-condizione	**context** AcquistoDAO:: exist(codCanzone,username) **pre:** codCanzone != null && username != null
Post-condizione	/

4. Design patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo di Spotibase. Per ogni pattern si darà:

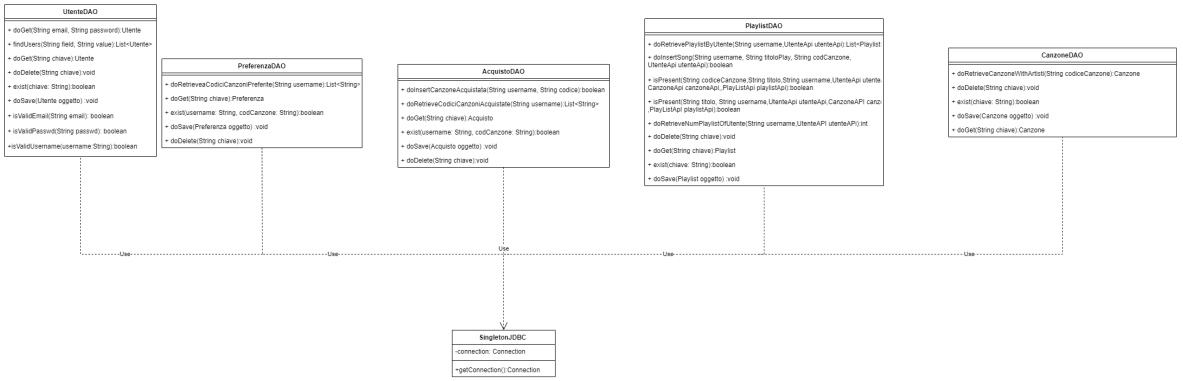
- Una brevissima introduzione teorica;
- Il problema che doveva risolvere all'interno di Spotibase;
- Una brevissima spiegazione di come si è risolto il problema in Spotibase;
- Un grafico della struttura delle classi che implementano il pattern.

Singleton

Singleton è un design pattern creazionale, ossia un design pattern che si occupa dell'istanziazione degli oggetti. Il singleton ha lo scopo di garantire che di una determinata classe venga creata una sola istanza.

Il problema presente nel nostro sistema è legato alla gestione della connessione JDBC, infatti creare e chiudere una connessione al database a ogni invocazione di un metodo DAO porta a una carenza dal punto di vista delle performance.

Il design pattern Singleton risolve questo problema creando una sola connessione JDBC che verrà usata da tutte le invocazioni dei metodi DAO.



Facade

Facade è un design pattern che funge da interfaccia che maschera il codice sottostante.

Il Facade ha lo scopo di migliorare la leggibilità e l'usabilità di una libreria software mascherando l'interazione con componenti più complessi.

Il pattern ha lo scopo di ridurre le dipendenze fra i sottosistemi.

Il Facade , nel nostro progetto , si focalizza sulla riduzione delle dipendenze fra classi affinché il codice possa avere un basso accoppiamento.

L'esempio più evidente è quello del Package DAOUtente, infatti l'interfaccia UtenteAPI viene utilizzata dagli altri package che vogliono usare i metodi dal DAO.

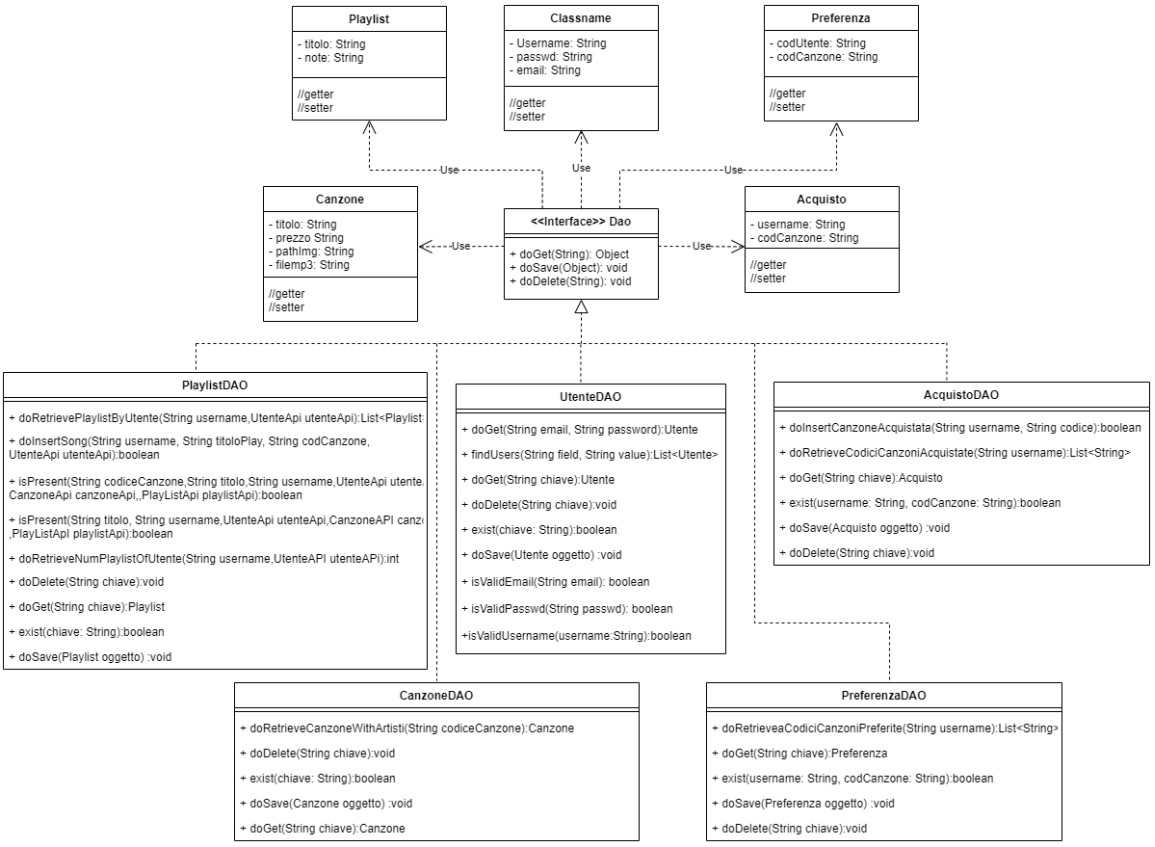
Dao

Dao (Data Access Object) è un pattern utilizzato per la gestione dei dati persistenti in un sistema software con un' architettura di tipo Three tier, permettendo una netta separazione tra il logic tier e il data tier.

Si implementa tramite una classe che rappresenta l'entità tabellare con relativi metodi, i quali permetteranno di eseguire query per estrapolare dati dal DBMS.

I metodi del Dao verranno richiamati dalle classi del logic tier.

In Spotibase, il pattern oltre a realizzare la separazione tra i vari tier e ad estrapolare dati dal DBMS, permette l'aumento della comprensione e della manutenibilità del codice.



5. Class diagram

Il class diagram è suddiviso secondo lo schema Tree Tier. Al seguente link è possibile vedere ogni Tier nello specifico e lo schema generico che li relaziona tutti. Nei class diagram specifici per Tier si evidenziano i collegamenti all'interno di essi.(Es. Nel class diagram del Data Tier viene mostrato come i POJO, classi Mapper, Classi Query, DAO e relative interfacce si relazionano tra di loro). Nel class diagram generale si evidenziano i collegamenti che un tier ha con un altro. (Es. Come il livello Presentation Tier si relazione al Logic Tier oppure il Logic Tier si relazione al Data Tier).

LINK:(Per visionare al meglio si consiglia di scaricare i file e aprirli con draw.io**).**
https://drive.google.com/drive/folders/1_xDIq8rr9QZfmFxRdHbgcIxBD-_uGVon?usp=sharing

6. Glossario

Sigla	Definizione
Package	Raccoglie un insieme di classi le cui interazione assolvono a una certa responsabilità del sistema
Pattern	Soluzione di un problema ricorrente
Singleton	Pattern creazionale che ha lo scopo di creare una e una sola istanza di una classe, e di fornire un punto di accesso globale per quella istanza
Facade	Pattern strutturale che fornisce un'interfaccia per accedere ad un insieme di oggetti che compongono il sottosistema. L'interfaccia definisce tutti i servizi del sottosistema
Dao	Data Access Object è un pattern architetturale che crea un'interfaccia per il data tier. Permette di estrapolare dati dal DBMS attraverso query. Separa in maniera netta il logic tier dal data tier.
Mapper	Classe setter per la costruzione degli oggetti, utilizzata dai Dao.
Three tier	Pattern architetturale che suddivide il sistema in 3 layer: - Presentation layer, interfaccia utente; - Logic tier, implementa la logica di business dell'applicazione; - Data tier per la gestione dei dati persistenti.