

# Swetify

Niccolò Malgeri, Filippo Viti, Alessio Delli Colli

July 2024

## Contents

<b>1</b>	<b>Motivazioni</b>	<b>3</b>
<b>2</b>	<b>Requisiti Funzionali</b>	<b>3</b>
2.1	Funzionalità proposte al cliente . . . . .	3
2.2	Funzionalità proposte all'artista . . . . .	3
2.3	Diagramma dei casi d'uso . . . . .	3
2.4	Template dei casi d'uso . . . . .	3
2.5	Mockups . . . . .	5
2.5.1	Pagina di accesso . . . . .	5
2.5.2	Pagina di ricerca . . . . .	5
2.5.3	Suggerimenti . . . . .	6
2.5.4	Coda di riproduzione . . . . .	6
2.6	Sequence diagram . . . . .	7
<b>3</b>	<b>Scelte di progetto</b>	<b>7</b>
<b>4</b>	<b>Documentazione</b>	<b>8</b>
4.1	Domain model . . . . .	8
4.1.1	Entity . . . . .	8
4.1.2	Customer . . . . .	8
4.1.3	Artist . . . . .	8
4.1.4	Track . . . . .	8
4.1.5	Song . . . . .	8
4.1.6	Podcast . . . . .	8
4.1.7	Playlist . . . . .	9
4.1.8	Album . . . . .	9
4.1.9	Suggestion . . . . .	9
4.1.10	PlaybackQueue . . . . .	9
4.2	Business logic . . . . .	9
4.2.1	Handler . . . . .	9
4.2.2	State . . . . .	9
4.2.3	NavigationManager . . . . .	9
4.2.4	AlbumsHandler . . . . .	9
4.2.5	ArtistInfoHandler . . . . .	9
4.2.6	HomeHandler . . . . .	10
4.2.7	LoginHandler . . . . .	10
4.2.8	PlaybackHandler . . . . .	10
4.2.9	PlaylistHandler . . . . .	10
4.2.10	RegistrationHandler . . . . .	10
4.2.11	SearchHandler . . . . .	10
4.2.12	SuggestionsHandler . . . . .	10
4.2.13	UserPlaylistsHandler . . . . .	10
4.3	Dao . . . . .	10

4.3.1	Dao . . . . .	10
4.3.2	CustomerDao . . . . .	10
4.3.3	ArtistDao . . . . .	10
4.3.4	SongDao . . . . .	11
4.3.5	PodcastDao . . . . .	11
4.3.6	PlaylistDao . . . . .	11
<b>5</b>	<b>Test</b>	<b>11</b>
5.1	Tipologie di test effettuati e organizzazione . . . . .	11
5.2	Test di integrazione . . . . .	12
5.2.1	SongDAOTest . . . . .	12
5.2.2	ArtistDAOTest . . . . .	12
5.2.3	CustomerDAOTest . . . . .	12
5.2.4	SuggestionDAOTest . . . . .	12
5.2.5	SongPlaylistDAOTest . . . . .	13
5.2.6	PodcastPlaylistDAOTest . . . . .	13
5.2.7	SongPlaysCountDAOTest . . . . .	13
5.3	Test funzionali . . . . .	13
5.3.1	HomeHandlerTest . . . . .	13
5.3.2	PlaybackHandlerTest . . . . .	13
5.3.3	RegistrationLoginHandlersTest . . . . .	13
5.3.4	AlbumLoadHandlerTest . . . . .	13

Figure 1: diagramma dei casi d'uso

## 1 Motivazioni

Il nostro intensivo utilizzo di piattaforme di streaming musicali ha suscitato in noi un interesse riguardo la loro struttura e il desiderio di replicarne il funzionamento.

Abbiamo deciso quindi di realizzare un'applicazione che simuli le loro funzionalità da noi denominata Swetify.

## 2 Requisiti Funzionali

Swetify prevede la partecipazione di due tipologie di utenti: cliente e artista.

### 2.1 Funzionalità proposte al cliente

- visualizzare un catalogo musicale che consenta agli utenti di cercare brani, album e artisti tramite una barra di ricerca.
- visualizzare le informazioni dettagliate di un brano, inclusi titolo, artista, album e durata.
- riprodurre, mettere in pausa e saltare i brani.
- creare, modificare ed eliminare le proprie playlist, aggiungere e rimuovere brani da queste playlist.
- ricevere raccomandazioni di brani basate sulla cronologia di ascolto dell'utente
- seguire gli artisti per ricevere aggiornamenti sui nuovi rilasci.

### 2.2 Funzionalità proposte all'artista

- caricare album contenenti canzoni o podcast.

### 2.3 Diagramma dei casi d'uso

### 2.4 Template dei casi d'uso

<b>UC1</b>	<b>riproduzione canzone</b>
livello	user goal
descrizione	l'utente cerca e riproduce una canzone
attori	cliente
pre-condizioni	l'utente deve avere le credenziali per effettuare l'accesso
post-condizioni	la canzone selezionata viene aggiunta alla coda
normale svolgimento	1) l'utente apre la pagina di ricerca 2) inserisce il nome di una canzone 3) seleziona una voce dall'elenco proposto 4) seleziona l'opzione "aggiungi in coda"
svolgimenti alternativi	4b) l'utente seleziona l'opzione "aggiungi in testa"

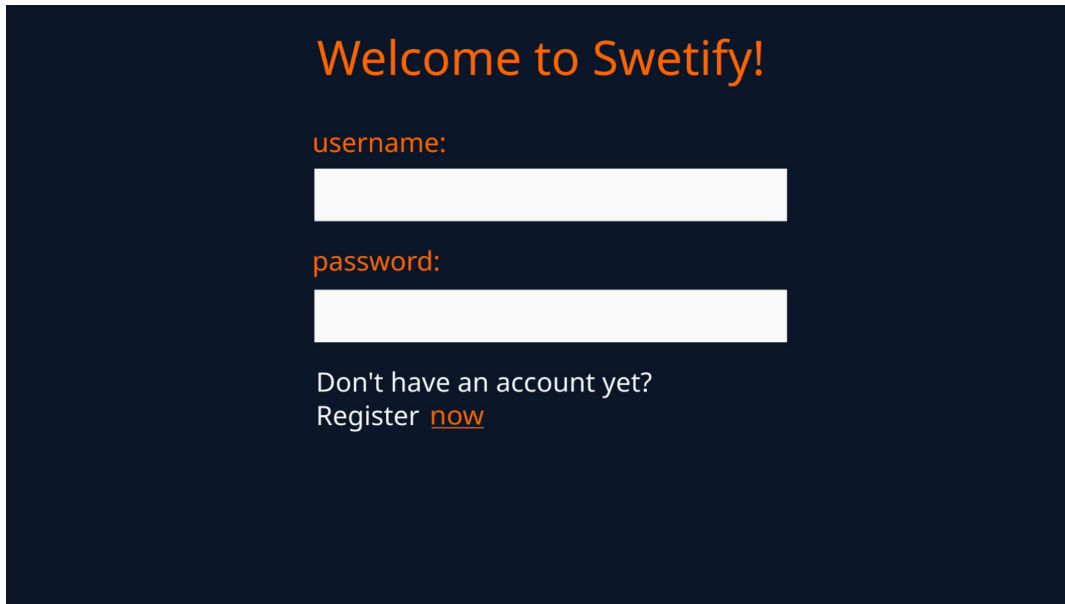
<b>UC2</b>	<b>modifica playlist</b>
livello	user goal
descrizione	l'utente modifica una delle sue playlist personali
attori	cliente
pre-condizioni	l'utente deve avere le credenziali per effettuare l'accesso ed avere una playlist salvata
post-condizioni	la playlist presenta i cambiamenti apportati dall'utente
normale svolgimento	1) l'utente apre la pagina "le mie playlist" 2) l'utente seleziona la playlist da modificare 3) l'utente cerca una canzone da aggiungere 4) l'utente termina salvando le modifiche
svolgimenti alternativi	3b) l'utente seleziona una canzone da rimuovere 4b) l'utente annulla le modifiche alla playlist

<b>UC3</b>	<b>aggiunta album</b>
livello	user goal
descrizione	l'artista carica un nuovo album sul suo profilo
attori	artista
pre-condizioni	l'artista deve avere le credenziali per effettuare l'accesso
post-condizioni	il nuovo album è visibile se cercato dagli utenti
normale svolgimento	1) l'artista seleziona l'opzione carica album 2) inserisce i nomi delle canzoni ed i rispettivi dati 3) l'artista termina l'inserimento salvando l'album
svolgimenti alternativi	3b) l'artista annulla il caricamento dell'album

<b>UC4</b>	<b>iscrizione ad un artista</b>
livello	user goal
descrizione	l'utente aggiunge un artista agli artisti seguiti
attori	cliente
pre-condizioni	l'utente deve avere le credenziali per effettuare l'accesso
post-condizioni	nel momento in cui l'artista carica un nuovo album l'utente può visualizzarlo nella sezione "nuovi rilasci"
normale svolgimento	1) l'utente apre la pagina di ricerca 2) inserisce il nome di una canzone 3) seleziona una voce dall'elenco proposto 4) seleziona l'opzione "aggiungi in coda"

## 2.5 Mockups

### 2.5.1 Pagina di accesso



Mockup of a login page for 'Swetify'. The page has a dark blue background. At the top, the text 'Welcome to Swetify!' is displayed in orange. Below this, the label 'username:' is followed by a white input field. The label 'password:' is followed by another white input field. At the bottom, the text 'Don't have an account yet?' is shown, followed by the word 'Register' and a link 'now' in orange.

Welcome to Swetify!

username:

password:

Don't have an account yet?  
Register [now](#)

### 2.5.2 Pagina di ricerca



Mockup of a search page for 'Swetify'. The page has a dark blue background. At the top, there is a search bar with the placeholder text 'searchName' and a close button (X). Below the search bar, there are three columns of results, each with a title in orange: 'Songs:', 'Podcasts:', and 'Artists:'. Each column contains a list of 9 items, each with a number and a line for the result. At the bottom right, there is a button labeled 'go back' with a right-pointing arrow.

searchName X

**Songs:**

1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_
4. \_\_\_\_\_
5. \_\_\_\_\_
6. \_\_\_\_\_
7. \_\_\_\_\_
8. \_\_\_\_\_
9. \_\_\_\_\_

**Podcasts:**

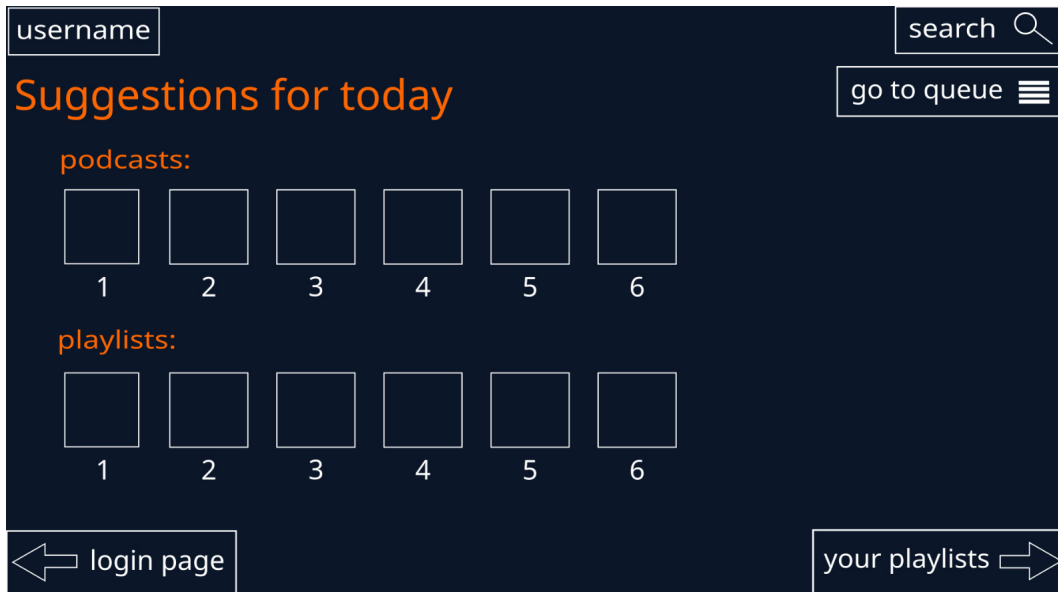
1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_
4. \_\_\_\_\_
5. \_\_\_\_\_
6. \_\_\_\_\_
7. \_\_\_\_\_
8. \_\_\_\_\_
9. \_\_\_\_\_

**Artists:**

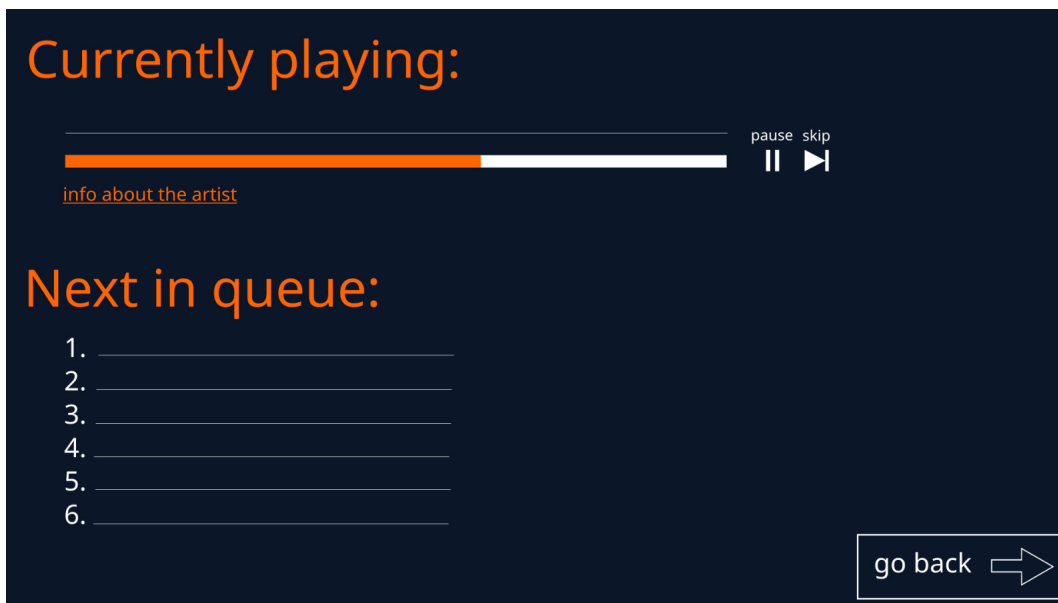
1. \_\_\_\_\_
2. \_\_\_\_\_
3. \_\_\_\_\_
4. \_\_\_\_\_
5. \_\_\_\_\_
6. \_\_\_\_\_
7. \_\_\_\_\_
8. \_\_\_\_\_
9. \_\_\_\_\_

go back ➡

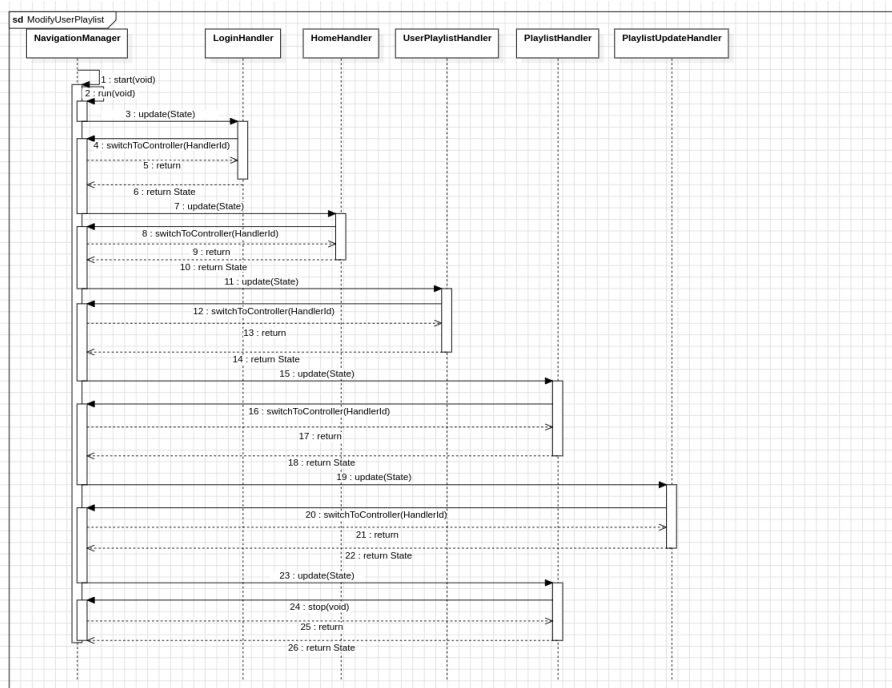
### 2.5.3 Suggestimenti



### 2.5.4 Coda di riproduzione



## 2.6 Sequence diagram



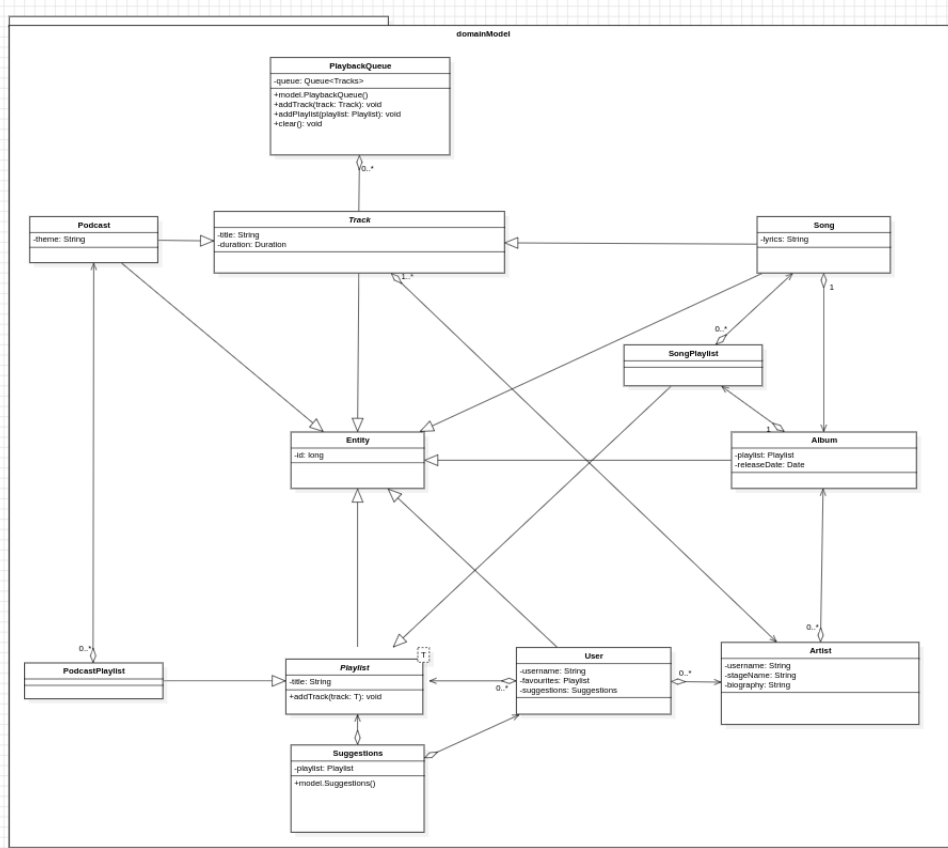
## 3 Scelte di progetto

Abbiamo scelto di strutturare il progetto creando una divisione tra domain model, business logic e data access objects.

- Il package **domainmodel** definisce le classi che simboleggiano gli elementi con cui l'utente interagisce.
- il package **businesslogic** descrive le modalità e le possibilità di interazione che l'utente ha con tali oggetti.
- il **dao** garantisce la persistenza di alcuni oggetti all'interno dell'applicazione.

## 4 Documentazione

## 4.1 Domain model



### 4.1.1 Entity

Classe base astratta che garantisce la presenza di un ID al fine di localizzare gli oggetti nel database.

### 4.1.2 Customer

Rappresenta il cliente e contiene le sue credenziali di accesso oltre alle playlist salvate.

### 4.1.3 Artist

Rappresenta l'artista e contiene le sue credenziali d'accesso e gli album da lui caricati.

#### 4.1.4 Track

Rappresenta un qualsiasi oggetto che può essere aggiunto alla coda di riproduzione.

### 4.1.5 Song

Concretizzazione di Track che rappresenta una canzone.

### 4.1.6 Podcast

Concretizzazione di Track che rappresenta un podcast.



#### 4.1.7 Playlist

Rappresenta una playlist ed espone i metodi per aggiungere o rimuovere delle track in testa oppure in coda.

#### 4.1.8 Album

Permette all'utente di riprodurre le canzoni al suo interno ma è immutabile.

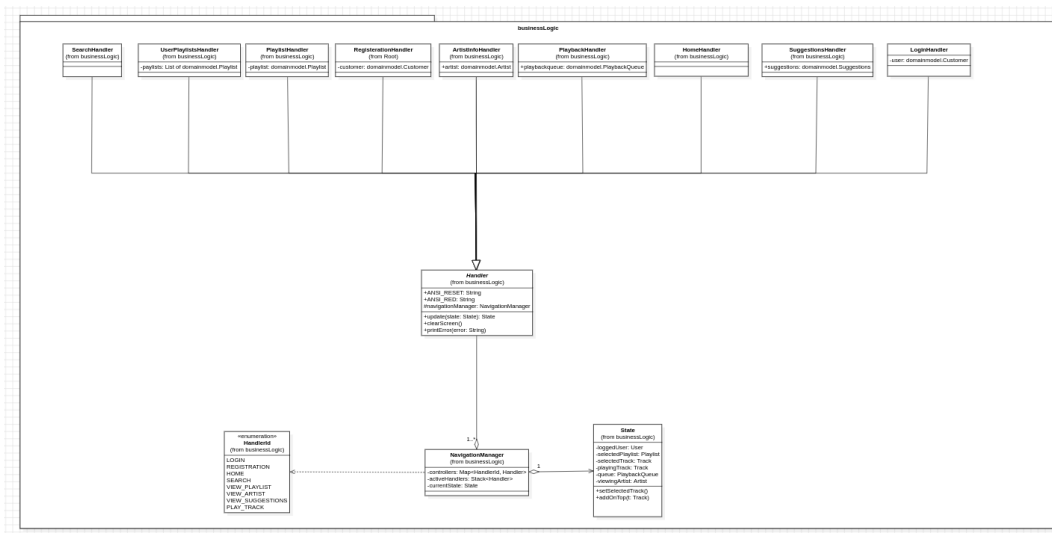
#### 4.1.9 Suggestion

Rappresenta il possibile interesse del cliente verso una track.

#### 4.1.10 PlaybackQueue

Rappresenta la coda di riproduzione e permette di aggiungere in testa ed in coda oltre ad eliminare in testa.

### 4.2 Business logic



#### 4.2.1 Handler

Classe base astratta che rappresenta un handler generico.

Essa garantisce la presenza di un metodo update in ogni handler che contiene la logica di quest'ultimo.

#### 4.2.2 State

Classe che permette ai vari handler di comunicare tra loro dati come l'utente loggato, la canzone o la playlist selezionata

#### 4.2.3 NavigationManager

Gestisce la navigazione tra le pagine passando il controllo ai vari handler.

#### 4.2.4 AlbumsHandler

Permette di visualizzare o riprodurre un album.

#### 4.2.5 ArtistInfoHandler

Mostra le informazioni salienti di un artista.

#### 4.2.6 HomeHandler

Contiene la schermata di ingresso e instrada l'utente verso le varie pagine.

#### 4.2.7 LoginHandler

Permette all'utente di effettuare l'accesso con un nome utente e una password o eventualmente passare alla schermata di registrazione.

#### 4.2.8 PlaybackHandler

Gestisce la coda di riproduzione e permette all'utente di visualizzare i brani contenuti in essa.

#### 4.2.9 PlaylistHandler

Mostra all'utente i brani contenuti in una data playlist e permette ad esso di aggiungerla alla coda di riproduzione.

#### 4.2.10 RegistrationHandler

Permette all'utente di registrarsi all'interno dell'applicazione con un nome utente ed una password.

#### 4.2.11 SearchHandler

Gestisce la ricerca all'interno delle canzoni, dei podcast e degli artisti disponibili.

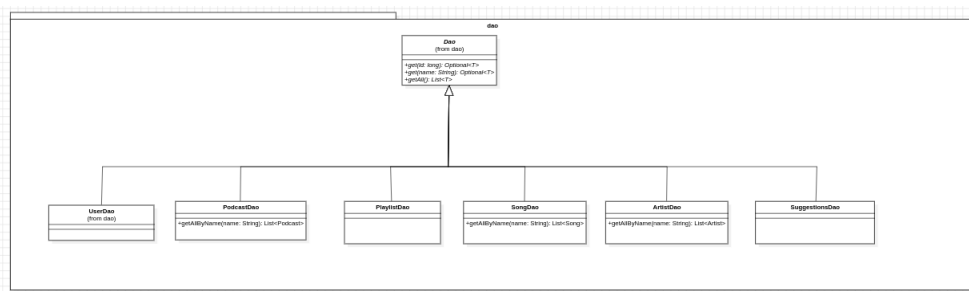
#### 4.2.12 SuggestionsHandler

Mostra all'utente le track suggerite dall'applicazione.

#### 4.2.13 UserPlaylistsHandler

Mostra l'elenco delle playlist create dall'utente.

### 4.3 Dao



#### 4.3.1 Dao

Classe base astratta del data access object che garantisce la capacità di leggere o scrivere oggetti dal database.

#### 4.3.2 CustomerDao

Permette di ottenere oggetti Customer dal database a partire dal loro nome utente o ID.

#### 4.3.3 ArtistDao

Permette di ottenere oggetti Artist dal database a partire dal loro nome utente o ID.

#### 4.3.4 SongDao

Permette di ottenere oggetti Song dal database in base ad una parola chiave.

#### 4.3.5 PodcastDao

Permette di ottenere oggetti Podcast dal database in base ad una parola chiave.

#### 4.3.6 PlaylistDao

Permette di leggere e scrivere le playlist dell'utente dal database.

## 5 Test

### 5.1 Tipologie di test effettuati e organizzazione

Per poter garantire il corretto funzionamento di *Swetify* sono stati effettuate due tipologie di test:

- **test di integrazione:** verificano il comportamento che le componenti dell'applicazione mostrano quando interagiscono tra di loro; nello specifico, vengono testate le operazioni che i DAO utilizzano per interagire con il database di sistema
- **test funzionali:** verificano il comportamento delle singole componenti dell'applicazione in un possibile scenario reale in cui quest'ultima viene utilizzata. Tra i test funzionali rientrano, per esempio, quelli relativi alla navigazione tra una schermata e l'altra dell'applicazione

Figure 2: struttura del package *Test*

In figura 2 viene mostrata l'organizzazione delle classi di test, contenute nel package *Test*. Come si può notare, è presente una classe base ***BaseTest*** contenente le funzionalità comuni a tutte le altre classi, come la funzione ***setUp()***; questa funzione serve per ripulire il database tra un test e l'altro, motivo per il quale viene eseguita prima di ciascun test.

```
@BeforeEach
public void setUp(){
    super.setUp();
    songDatabase = new SongDAO();

    song1.setTitle("title1");
    song1.setDuration(Duration.ofSeconds(10));
    song2.setTitle("title2");
    song2.setDuration(Duration.ofSeconds(20));
    song3.setTitle("title3");
    song3.setDuration(Duration.ofSeconds(15));
    songDatabase.save(song1);
    songDatabase.save(song2);
    songDatabase.save(song3);
}
```

Figure 3: esempio di implementazione di *setUp()* (in questo caso relativa a *SongDAOTest*)

Come si può vedere in figura 3, la *setUp()* della classe base viene chiamata all'inizio della *setUp()* delle classi derivate, seguita dall'inizializzazione del database e degli oggetti necessari per l'esecuzione dei test.

Per la scrittura di tutti i test è stato utilizzato il framework ***Junit 5.0***.

## 5.2 Test di integrazione

Di seguito vengono descritti i metodi presenti nelle classi relative ai test di integrazione; ogni metodo consiste nel verificare il comportamento di una specifica query, scritta in linguaggio SQL, per l'accesso al database.

### 5.2.1 SongDAOTest

1. **testGet()**: verifica che *SongDAO.getByTitle()* e *SongDAO.get()* restituiscano lo stesso risultato, confrontando i titoli delle canzoni (oggetti di tipo *Song*) restituite
2. **testGetByTitle()**: prima viene verificata la presenza di alcuni oggetti di tipo *Song* salvati precedentemente sul database, attraverso *SongDAO.getByTitle()*; successivamente, viene creato un nuovo oggetto *song* di tipo *Song* senza però salvarlo, e viene verificata l'assenza di risultati restituiti da *SongDAO.getByTitle(song.getTitle())*
3. **testGetAll()**: verifica che *SongDAO.getAll()* restituisca il giusto numero di risultati, sulla base di quanti oggetti di tipo *Song* sono presenti nel database

### 5.2.2 ArtistDAOTest

1. **testGet()**: verifica che *ArtistDAO.getByStageName()* e *ArtistDAO.get()* restituiscano lo stesso risultato, confrontando nome, numero di seguaci e biografia degli artisti (oggetti di tipo *Artist*) restituiti
2. **testGetByUserName()**: prima viene verificata la presenza di alcuni oggetti di tipo *Artist* salvati precedentemente sul database, attraverso *ArtistDAO.getByUserName()*; successivamente, viene creato un nuovo oggetto *artist* di tipo *Artist* senza però salvarlo, e viene verificata l'assenza di risultati restituiti da *ArtistDAO.getByUserName(artist.getUserName())*
3. **testByStageName()**: dopo aver creato e salvato due nuovi oggetti di tipo *Artist* con lo stesso nome di uno degli artisti presenti nel database, verifica che *ArtistDAO.getByStageName()* restituisca il giusto numero di risultati
4. **testGetAll()**: prima verifica che *ArtistDAO.getAll()* restituisca il giusto numero di risultati, sulla base di quanti oggetti di tipo *Artist* sono presenti nel database, e successivamente verifica che tali risultati siano corretti, controllando nome, numero di seguaci e biografia per ciascuno di essi

### 5.2.3 CustomerDAOTest

1. **testGet()**: verifica che *CustomerDAO.getByStageName()* e *CustomerDAO.get()* restituiscano lo stesso risultato, confrontando nome utente e password dei customer (oggetti di tipo *Customer*) restituiti
2. **testGetByUserName()**: prima viene verificata la presenza di alcuni oggetti di tipo *Customer* salvati precedentemente sul database, attraverso *CustomerDAO.getByUserName()*; successivamente, viene creato un nuovo oggetto *user* di tipo *Customer* senza però salvarlo, e viene verificata l'assenza di risultati restituiti da *CustomerDAO.getByUserName(user.getUsername())*
3. **testGetAll()**: prima verifica che *CustomerDAO.getAll()* restituisca il giusto numero di risultati, sulla base di quanti oggetti di tipo *Customer* sono presenti nel database, e successivamente verifica che tali risultati siano corretti, controllando nome utente e password per ciascuno di essi

### 5.2.4 SuggestionDAOTest

- **testSuggestions()**: verifica che la query effettuata dalla funzione *getTopSongsBySimilarUsers* della classe *SuggestionDa* sia conforme alla logica desiderata calcolando in maniera indipendente il risultato atteso grazie alle funzioni *getUserTopTen*, *getCustomersWhoListenedTopTenSongs*, *getTopTenSongsByTopTenListeners*.

#### **5.2.5 SongPlaylistDAOTest**

#### **5.2.6 PodcastPlaylistDAOTest**

#### **5.2.7 SongPlaysCountDAOTest**

### **5.3 Test funzionali**

Come per i test di integrazione, di seguito vengono descritti i metodi presenti nelle classi relative ai test funzionali; nella *setUp()* di ciascun metodo viene inizializzato il *NavigationManager* e viene chiamato il metodo *pushHandler()* specificando l'Id dell'handler coinvolto nei test all'interno di una certa classe.

#### **5.3.1 HomeHandlerTest**

#### **5.3.2 PlaybackHandlerTest**

#### **5.3.3 RegistrationLoginHandlersTest**

#### **5.3.4 AlbumLoadHandlerTest**