Music Player Application

Tushar Dixit(PES2201800138)

Abstract

This project is an implementation of a Music Player System written in C++11. It has the concept users that are listeners and admins, each with their own privileges. It supports various functionalities like :

1. Playlist management for listeners
2. Basic Music Player functions like Playing, Pausing, Stopping songs.
3. Song recommendation system
4. Song rating system.

Along with admin functionalities like:
1. Managing Database.
2. Playing Songs, etc.

Module Description:

1. **server.h**: This file declares all the classes and functions in the project.

2. **main.c file**: This is the client file which is using the interface present in server.h file.

3. **server.cpp**: This file defines the functions and attributes of all classes in the project.

4. **users.csv**: This is the database file which contains all the information about the user accounts in the system.
[user id, username, password, favourite genre, playlist(with songid’s separated with ‘/’)]

5. **songs.csv**: This is the database file which acts as an index of songs in

the database.
it consists of :
[song id, name of the song, artist, likes, location of the song in the system, genre, uid of people who liked the song(separated by ‘/’)].

6. **admin.csv**: This is the database file which contains login information about the admin of the system.

7. **songs folder**: This is the database folder which consists of all the audio files in .mp3 format.

Actors:
1. Listeners
2. Admin

Use Cases:
1. Login: login as a listener or an admin

2. Add songs: add a song to playlist for listeners and add song to database for admins

3. Display playlist/database: display user specific playlist for listeners and database for admins.

4. Display song details: display song details of the song id provided.
5. Remove songs: remove songs from playlists for listeners and remove from database for admins.

6. Play playlist: play the playlist in a loop for listeners and database for admins.

7. Play a song: play a song with a song id.

8. Like a song: Like a song functionality for listeners.

9. Recommend songs: listener function to recommend songs based on the favourite genre of listener.

10. Recommend similar songs: listener function to recommend songs similar to the given song id.

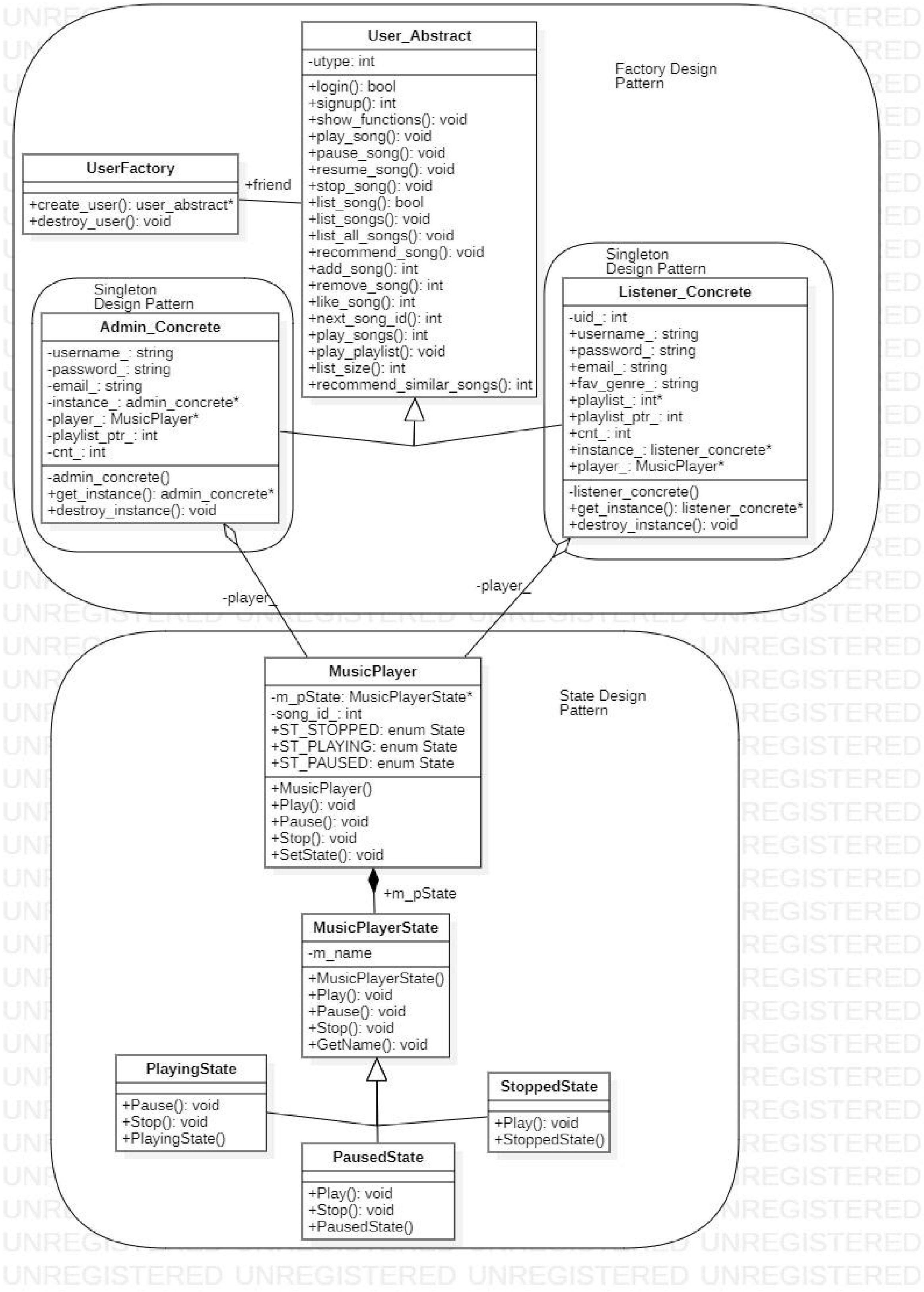
11. Logout

Design Patterns Used:

Design patterns used(See the class diagram below to see how they have been used):

1. Factory: for two different types of users i.e. admin and listener
2. Singleton: to prevent the client from allowing multiple users to login at the same time.
3. State: for different states of the music player namely Playing, Paused, Stopped.

Class Diagram:


Factory Design Pattern
User_Abstract
-utype: int
+login(): bool
+signup(): int
+show_functions(): void
+play_song(): void
+pause_song(): void
+resume_song(): void
+stop_song(): void
+list_song(): bool
+list_songs(): void
+list_all_songs(): void
+recommend_song(): void
+add_song(): int
+remove_song(): int
+like_song(): int
+next_song_id(): int
+play_songs(): int
+play_playlist(): void
+list_size(): int
+recommend_similar_songs(): int
UserFactory
+create_user(): user_abstract*
+destroy_user(): void
+friend
Singleton Design Pattern
Admin_Concrete
-username_: string
-password_: string
-email_: string
-instance_: admin_concrete*
-player_: MusicPlayer*
-playlist_ptr_: int
-cnt_: int
-admin_concrete()
+get_instance(): admin_concrete*
+destroy_instance(): void
Singleton Design Pattern
Listener_Concrete
-uid_: int
+username_: string
+password_: string
+email_: string
+fav_genre_: string
+playlist_: int*
+playlist_ptr_: int
+cnt_: int
+instance_: listener_concrete*
+player_: MusicPlayer*
-listener_concrete()
+get_instance(): listener_concrete*
+destroy_instance(): void
-player_
-player_
State Design Pattern
MusicPlayer
-m_pState: MusicPlayerState*
-song_id_: int
+ST_STOPPED: enum State
+ST_PLAYING: enum State
+ST_PAUSED: enum State
+MusicPlayer()
+Play(): void
+Pause(): void
+Stop(): void
+SetState(): void
+m_pState
MusicPlayerState
-m_name
+MusicPlayerState()
+Play(): void
+Pause(): void
+Stop(): void
+GetName(): void
PlayingState
+Pause(): void
+Stop(): void
+PlayingState()
StoppedState
+Play(): void
+StoppedState()
PausedState
+Play(): void
+Stop(): void
+PausedState()