Music Repository and Player System

Phase 4

1. Project Description

We are about to model a music repository and player system. We aim to develop a search engine and database for songs with a media player to play the music along with some add-on functionalities to enhance the usability of the system. There will be two governing actors in the system - End-User and the Administrator. Both will have certain rights specific to them for which they will have to login to the system.

Songs are the system's main entity that will hold attributes like song_name through which user can will be able to perform search, song_id as primary key and movie_id for mapping songs with their respective Movies . Once search is completed Player can be used to play the songs. Instead of performing search user can make use of the default playlists - SingerPlaylist, AlbumPlayist and MoviePlaylist which are maintained by Administrator. User can create his own User Specific Playlists which will be visible only to him. All these playlists will inherit their functionalities from Playlist. As an add-on feature user can also view the lyrics for the currently playing song.

The system will allow users to perform search and listen to the songs via media player provided in the system. Song search can be done based on two parameters i.e song name and movie name. There will be an add on feature for user to get lyrics of the song he is listening to. Lyrics will be displayed if the user chooses the lyrics options. In addition to this system will provide some default playlist under albums, movies and singers which will be maintained by the Administrator . User can play these playlists directly without searching. User can also create his own playlist for future use and perform few operations like adding/ deleting songs on it. To use this feature, the new user should sign up to the system . Once registered, he can login to the system to create his own playlist and also modify existing playlist if any .

All of this needs to be implemented in such a way that it can be used on laptop/computer . There needs to be a provision for the system to be occasionally connected to a network in cases when the user needs to view the lyrics of the current song.

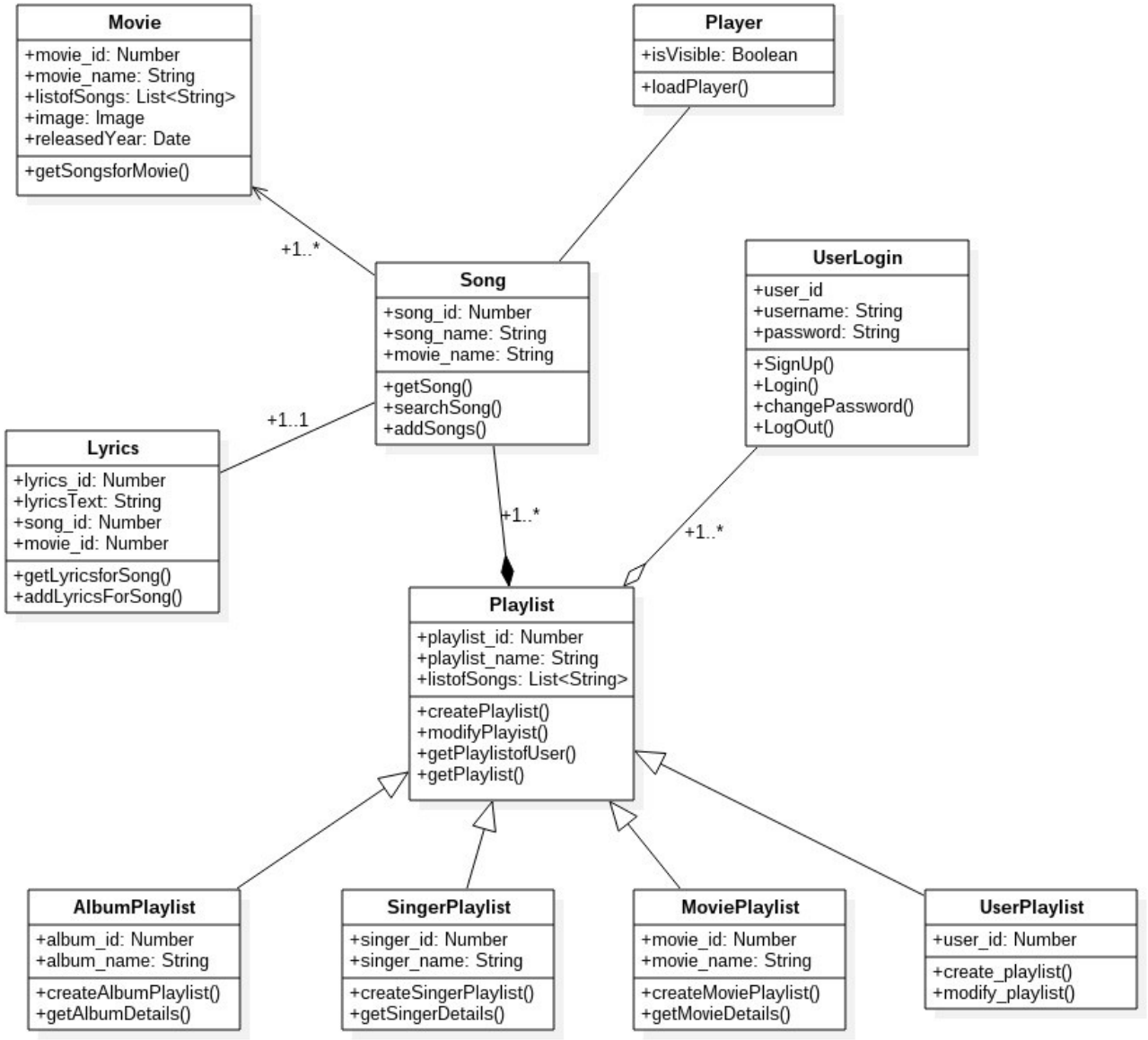
The user base for our system can range from a naive person who is first time user of such a system to a person who uses computer in his daily life . Thus in either case an intuitive user interface is absolutely necessary so that the end user is himself able to navigate through the system and figures out the next step to be taken to proceed.

Color Coding: Actors Noun Verb Phrases Operating System Ease of Use

Use-Cases with their respective actors:

Actors	Use-Cases
1. End User	1. To Login in the system (for existing users) 2. To SignUp in the system (for new users) 3. To search a song by song_name or movie_name 4. To view lyrics of the current song. 5. To create a user playlist 6. To modify a user playlist 7. To change password 8. To play songs 9. To logout from the system
2. Administrator	1. To add new songs to existing database in timely manner 2. To add lyrics for the songs. 3. To create default album playlists. 4. To create default movie playlists. 5. To create default singer playlists. 6. To modify and update existing playlists.

2. Class Diagram



The different relations present in our class diagram are as follows:

1. Aggregation
   - Hard Aggregation between Class Song and Class Playlist → Each playlist will have once object of each song in the list.
   - Soft Aggregation between Class UserLogin and Clas Playlist → Every user can have multiple playlists.
2. Inheritance
   - Classes AlbumPlaylist, SingerPlaylist, MoviePlaylist and UserPlaylist are children of Class Playlist and thus will inherit all the functions from the Playlist class.
3. One to Many Relationships
   - Playlist to Songs, Movie to Songs and User to Playlists
4. One to One Relationships
   - Song to Lyrics

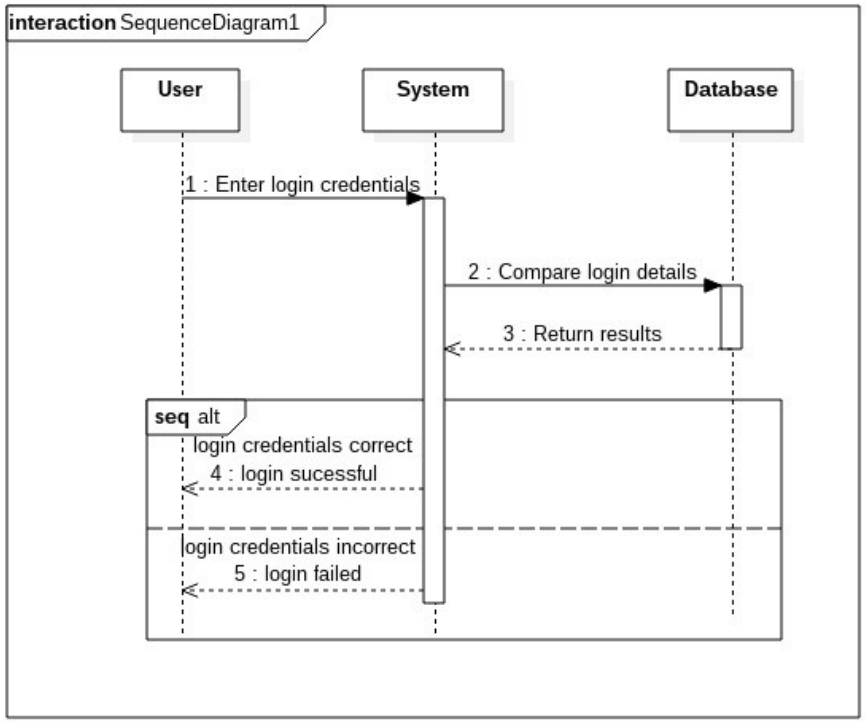
Below are the changes in Class diagram as compared to Phase 1:

1. Class Song : Addition of addSong() → As from time to time administrator might be required to add new songs to the database this function is required.
2. Class Lyrics: Addition of addLyricsForSong() → If new songs are added to database, lyrics for them should also be added by the administrator using this function.
3. Class UserLogin: Addition of Logout() → User will require to logout of the system once he is done listening to songs.
4. Class AlbumPlaylist:
   A) Addition of createAlbumPlaylist() → When administrator wants to create a new default Album playlist for the future use of end-users this function will come into play. Newly created default playlist will have a name and id associated with it to reference to it in database.
   B) Addition of getAlbumPlaylist() → If any existing default Album playlist is called by the user, this function will fetch the playlist from database ready to be played.
5. Class SingerPlaylist: Similar to AlbumPlaylist both functions are added to SingerPlaylist for said purposes.
   A) Addition of createAlbumPlaylist()
   B) Addition of getAlbumPlaylist()
6. Class MoviePlaylist: Similar to AlbumPlaylist both functions are added to MoviePlaylist for said purposes.
   A) Addition of createAlbumPlaylist()
   B) Addition of getAlbumPlaylist()

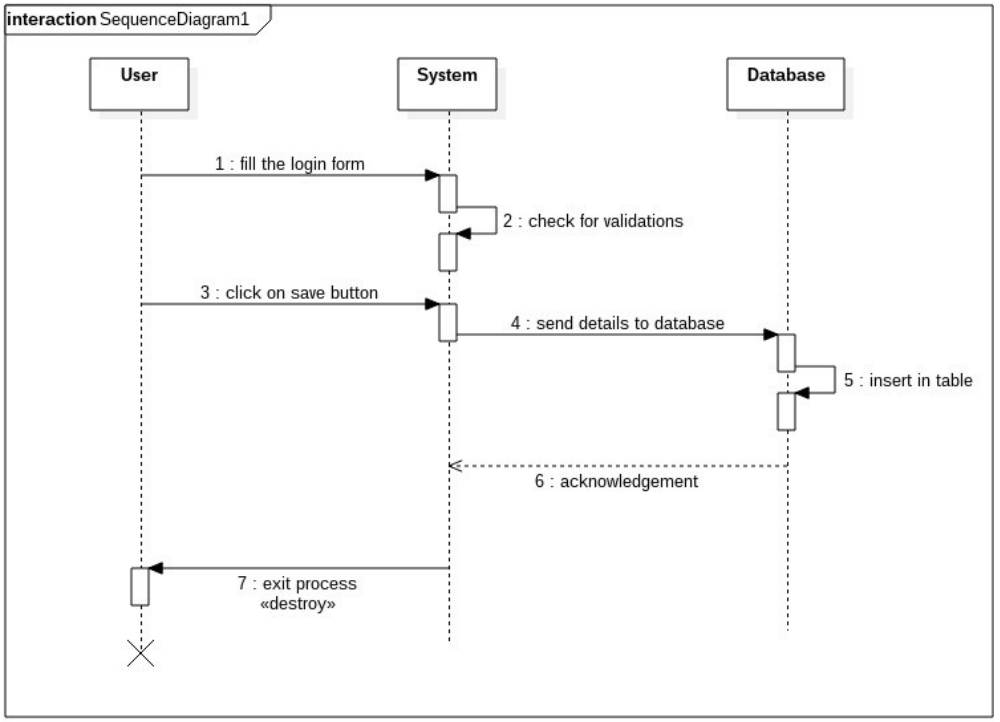
3. Use-Cases (design-time sequence diagrams)

We have used sequence diagram feature of StarUML for creating the interaction for all the 15 usecases in scope of our project. Below are all the illustrated diagrams usecase wise.

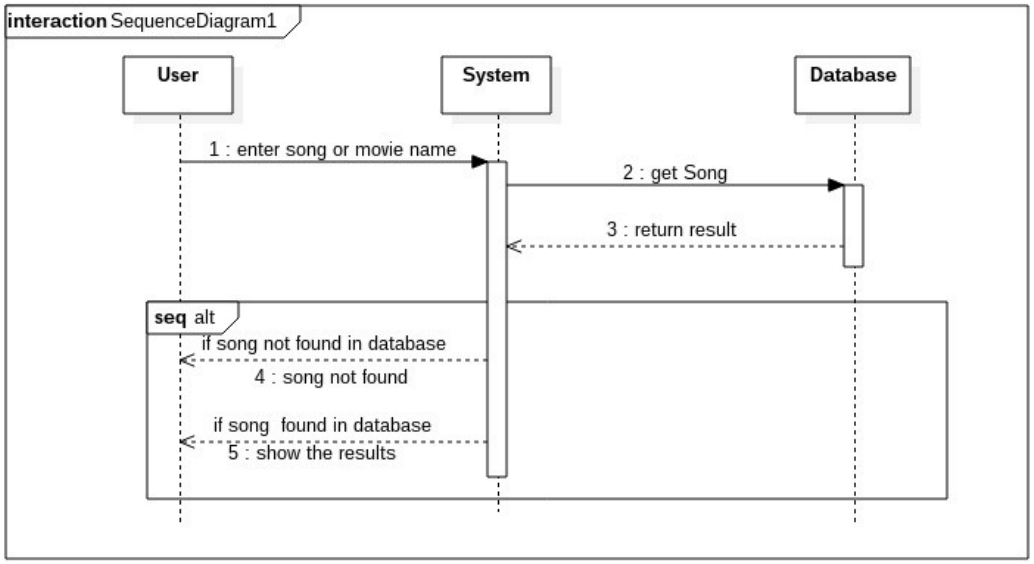
UC01: To login to the system



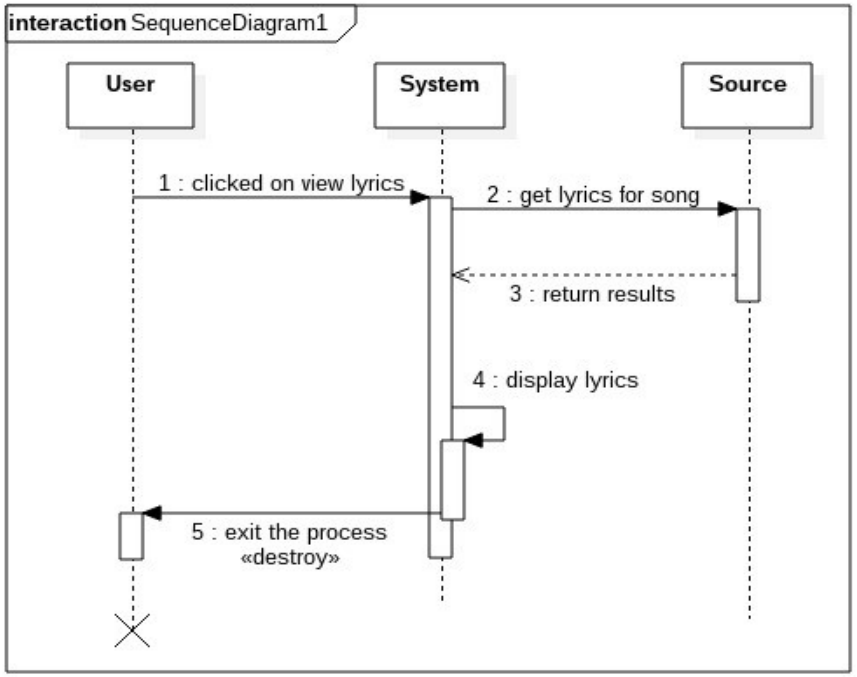
UC02: To signup with the system (in order to create/maintain playlists)



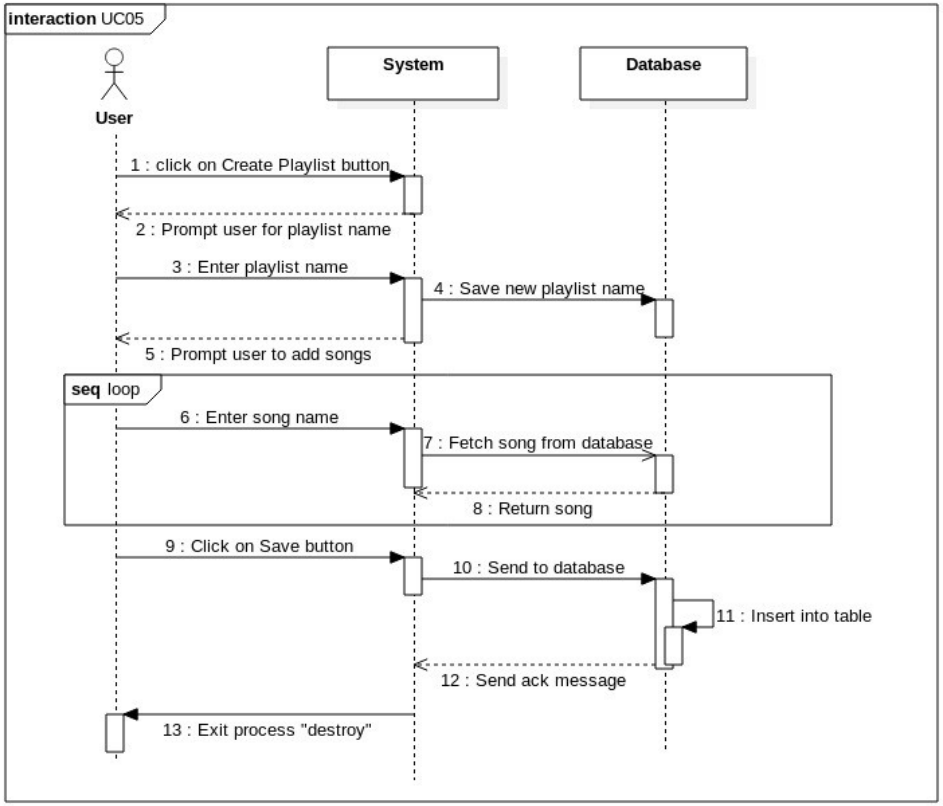
UC03: To search songs by song_name/ movie_name



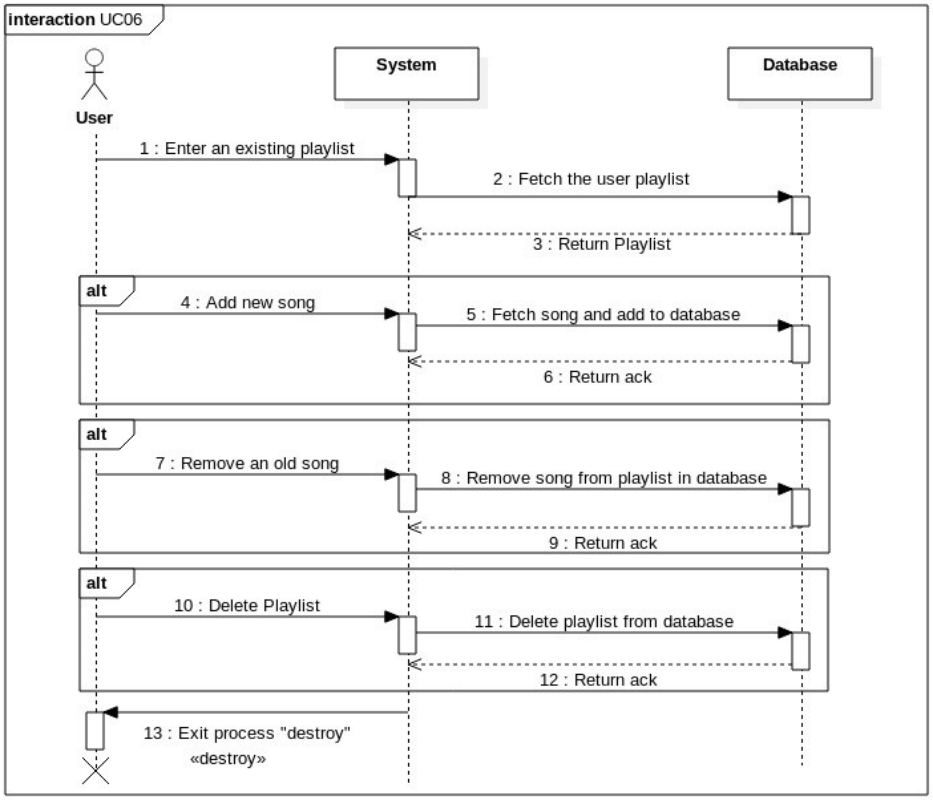
UC04: To view lyrics of a song



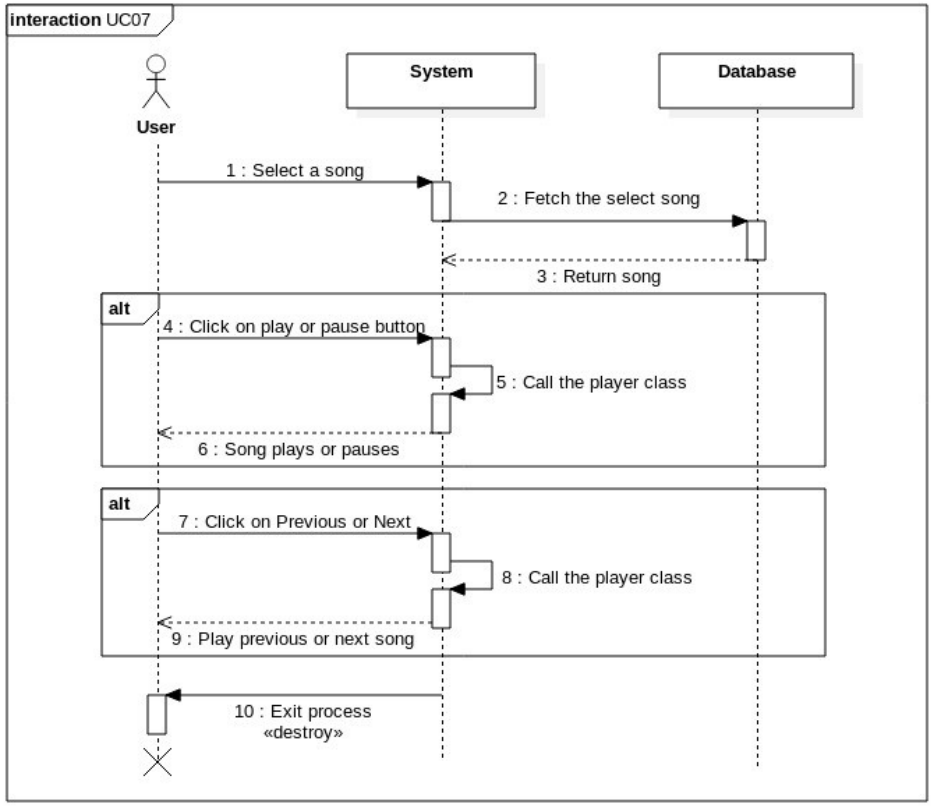
UC05: To create playlist



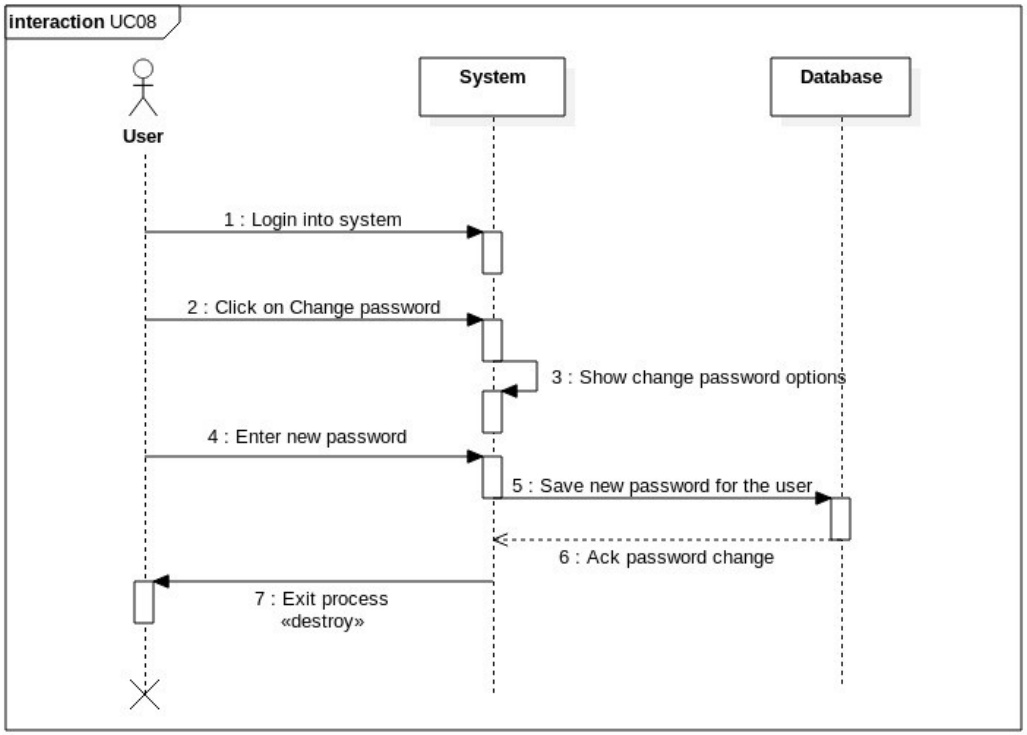
UC06: To modify user playlist



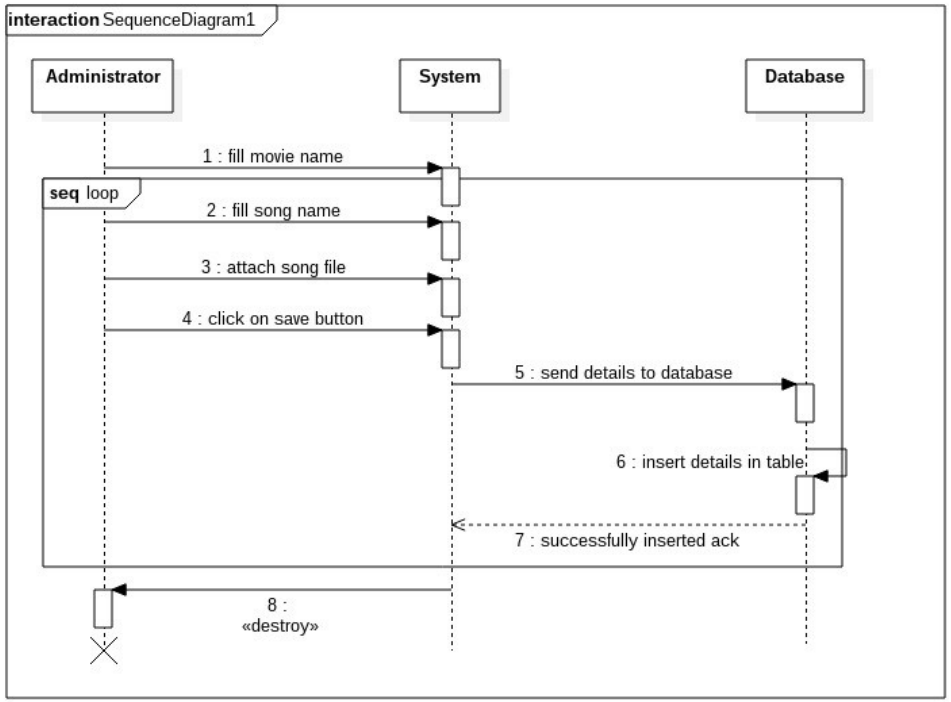
UC07: To play songs



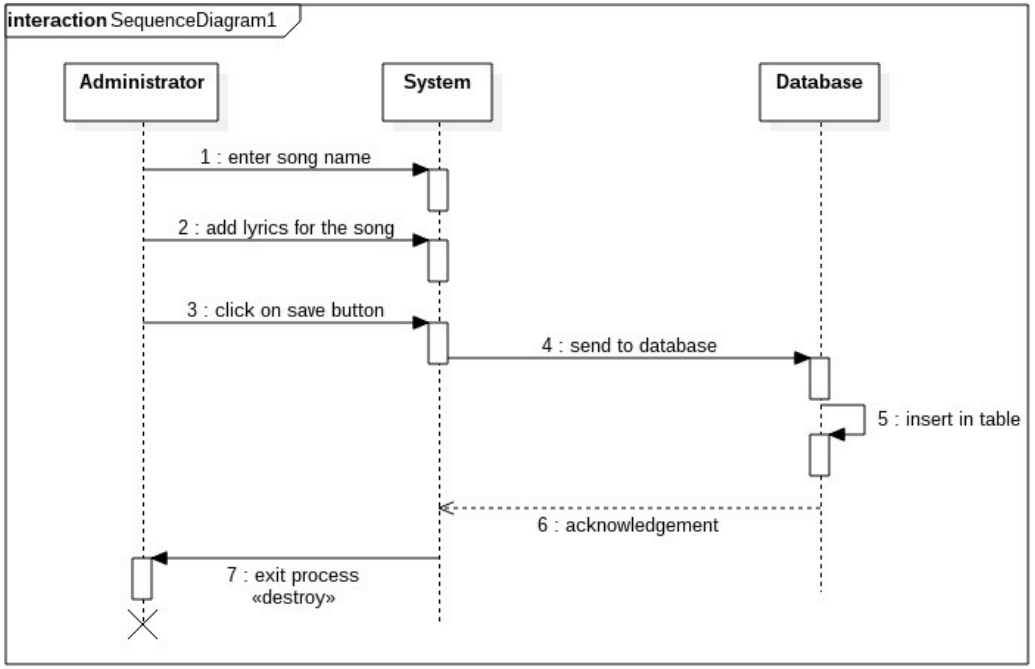
UC08: To change password



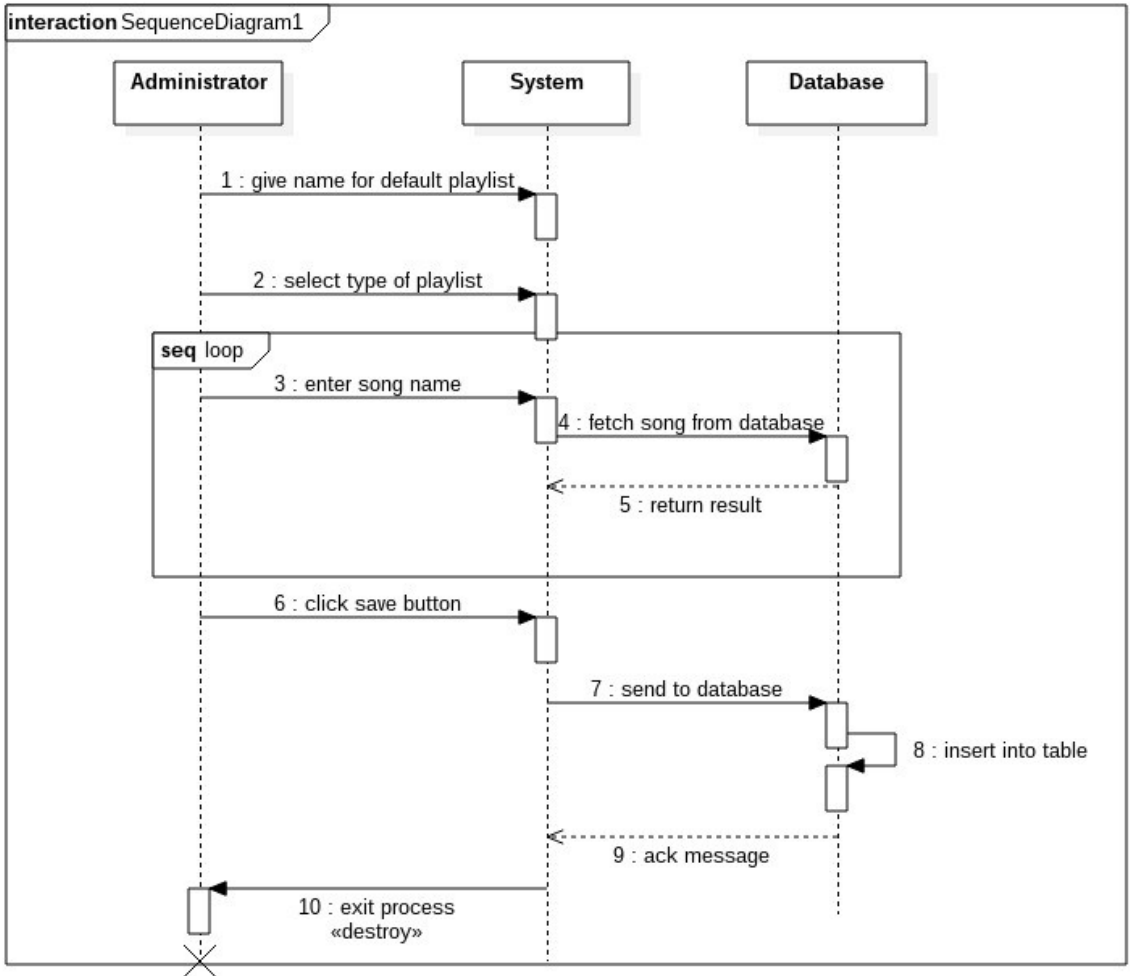
UC09: To update songs database



UC10: To add lyrics for a song in database



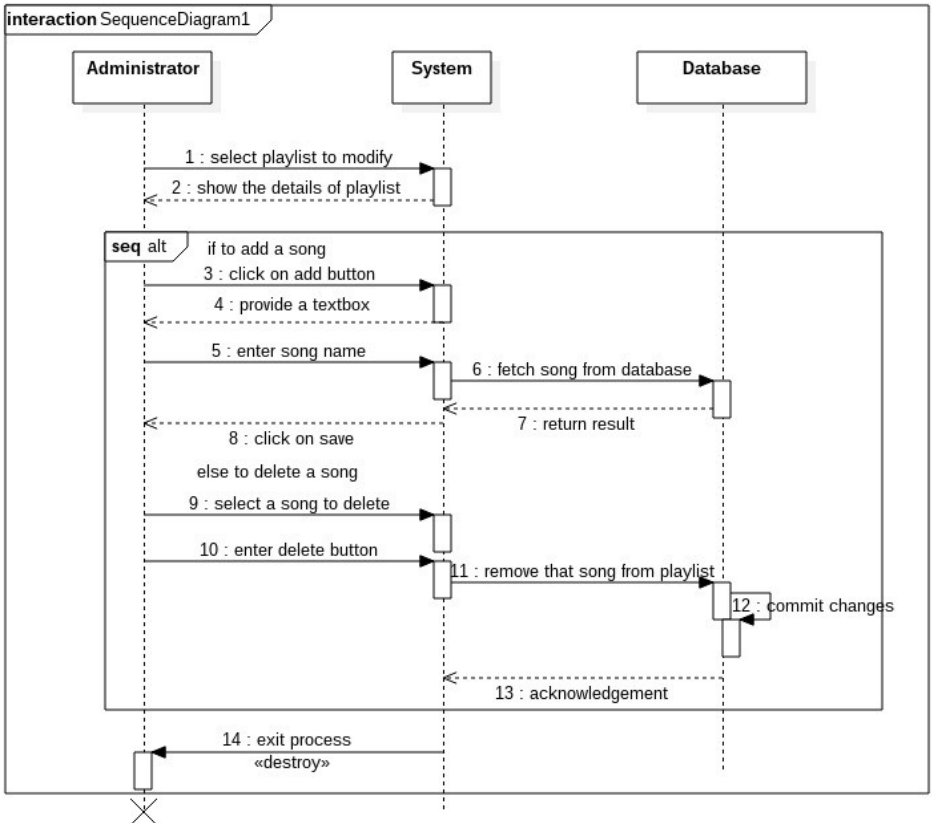
UC11: To create default album playlist



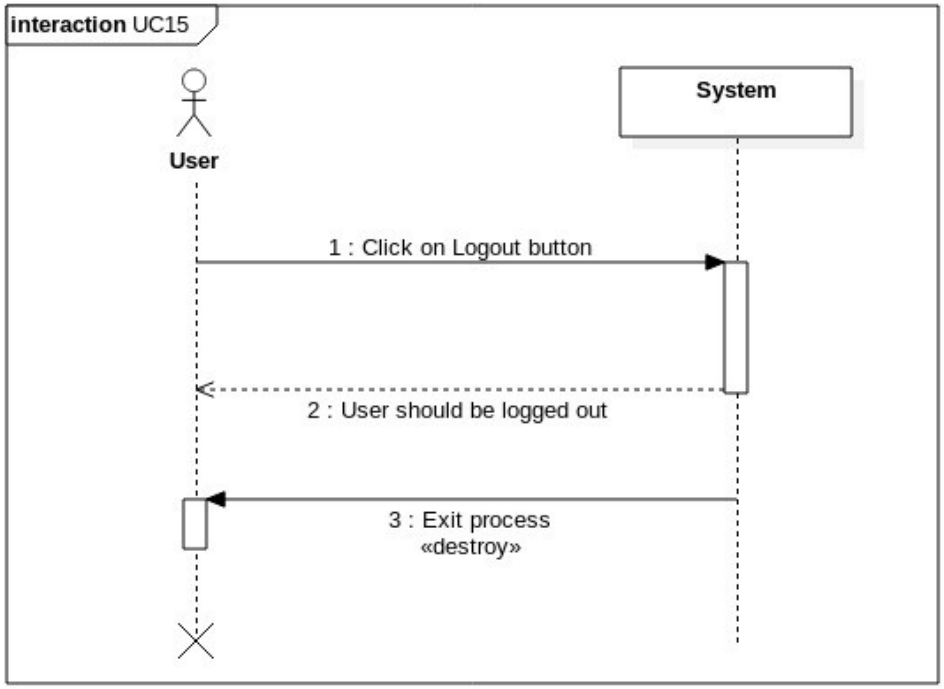
UC12: To create default movie playlist
UC13: To create default singer playlist

For UC 12 and 13 which are for default movie and singer playlists, the flow will be same as that of UC11 thus we have not redrawn the sequence diagram.

UC14: To modify/ update default playlists



UC15: To logout from the system



4. Use-Case Table

Use-Case/ **Class**	C1 : Movie	C2 : Song	C3 : Lyrics	C4 : Playlist	C5 : Player	C6 : User Login	C7 : User Playlist	C8 : Album Playlist	C9 : Movie Playlist	C10 : Singer Playlist
UC1 : To login in the system	-	-	-	-	-	*Login()*	-	-	-	-
UC2 : To signup in system	-	-	-	-	-	*SignUp()*	-	-	-	-
UC3 : To search song by song_name/movie_name	*getSongsForMovie()*	*searchSong()*	-	-	-	-	-	-	-	-
UC4 : To view lyrics of song	-	-	*getLyricsForSong()*	-	-	-	-	-	-	-
UC5 : To create user playlist	-	-	-	-	-	-	*create_playlist()*	-	-	-
UC6 : To modify user playlist	-	-	-	-	-	-	*modify_playlist()*	-	-	-
UC7 : To play songs	-	-	-	-	*loadPlayer()*	-	-	-	-	-
UC8 : To change password	-	-	-	-	-	*changePassword()*	-	-	-	-
UC9 : To	-	*addSongs*	-	-	-	-	-	-	-	-

update songs database		*()*								
UC10 : To add lyrics to song			*addLyrics ForSong()*							
UC11 : To create default album playlist	-	*getSong()*	-	*createPlayist()*	-	-	-	*createAlbumPlaylist()*	-	-
UC12 : To create default movie playlist	*getSongFor Movie()*	-	-	*createPlaylist()*	-	-	-	-	*createMovie Playlist()*	-
UC13 : To create default singer playlist	-	-	-	*createPlaylist()*	-	-	-	-	-	*createSinger Playlist*
UC14 : To modify and update default playlist	-	-	-	*modifyPlaylist()*	-	-	-	-	-	-
UC15 : To logout from system	-	-	-	-	-	*LogOut()*	-	-	-	-

5. Final remarks about your project, including an assessment of the use-case driven methodology for your project

Developing Music Player and Repository system helped us obtain a lot of understanding object oriented methodology and the different concepts involved in it. After working on this project we feel that Use-Case modeling is best and most practical way to development, as use cases are written from outside perspective they are easier to comprehend by anyone, specially the clients. We break the requirements down into shorter scenarios in use cases which are easily understood by the readers.

Use cases focus on the users of the system rather than on the system itself, thus the real system needs are brought to light early on. In our project we had two main actors – EndUser and the Administrator. Thus all our use cases revolved around them and while writing them down, specially while making the integration diagrams, we had to think how the use cases would come into play practically. This made it easier when we started programming as we had some pre-written steps to start with. Each use case describes one way the system is used, but one of the big benefits of use case modeling is that it also describes all of the things that might go wrong. Thus while making the sequence diagrams we even thought of the scenarios when the things do not go as expected or the system fails. Also making the Use-Case table which listed all the classes and methods which will potentially be required in the development enabled us to put in more thought before starting the coding part. We had beforehand analyzed the relationships between classes – inheritance, aggregation and associations.

The main change in our implementation is using an inbuilt database in Java rather than an actual database and the console as the medium for interaction with user rather than a GUI. We could not implement this due to time constraints. This goes to show that there can, very well be changes or modification in the requirements of the system which are discovered in a later stage of a project. But inspite of the changes, Use Case modelling approach enabled us to lay a strong foundation of the requirements before we start the coding phase.