Software Design Document

Project “Smart Music Player”

Prepared by

- Sakshi Agrawal(1812845)
- Sakshi Gupta(1812846)
- Shivangi(1812864)
- Shreya Jain(1812869)

GROUP ID – **CS 32**

Banasthali Vidyapith

21-Feb-2021

Table of contents

Introduction

1.1 Purpose of this document

The purpose of this document is to describe the implementation of the *Smart Music Player* whose requirements have been described in detail in the SRS document.

The *Smart Music Player* as specified earlier, is a music player web application that not only plays music but also recommends music.

The sections in this document will provide guidelines related to the structure and the design of the project and will contain the following too.

- **ER Diagram**: displays the relationship of entity sets stored in a database and help to explain the logical structure of databaseusing : entities, attributes and relationships.
- **Sequence Diagram**: specific information about how objects operate with one another and in what order.
- **Use case Diagram**: specify core functionalities of the system and visualize the interaction diagrams of various things called as actors with the use case.
- **Activity Diagram:** describe the dynamic aspects of the system using a flowchart which represent the flow from one activity to another activity.

1.2 Scope of the development project

- Name of the project is "Smart Music Player". It is a Web Application.
- This is amusic player in which users are given recommendation on the basis of their artist/genre/language preferences, also on the basis of the type of songs the user has liked or disliked. This is an ad free music portal and free of cost.
- The user can create/update his profile.
- User authentication is also achieved by *One Time Password* if he forgets his password.
- Searching music with the help of artist/genre/language/year/album is also possible.
- The user can access music from their local storage and play it in the Music Player.

Advantages:-

- It is not platform or service specific.
- There is no specific audience for this software. Anyone can install it and use it.
- Manage the information of the user.

1.3 Definition, acronyms and abbreviations

- TBD means "To be Decided", these are the components that are not yet decided.
- Terminology definitions are given in the following table.

Term	**Description**
User	Any living being who is interacting with the software is a *user*.
System	The package of all the components which takes input and gives output to demonstrate the features of the software is called System.
Local Storage	This is the collection of songs already present in the user's system.
Database	The set of data which describes and gives information about the sound track.
Recommendation System	A system which takes a track as input and outputs set of tracks closely related to the input.
SQL	The SQL queries are used to manage the database and also for the recommendation of songs.

1.4 References

- http://www.facultyintranet.intercol.edu/onlinecourse/CourseFiles/COMP-401_976/Project/IEEE%20STANDARD%201016.doc
- Fairley, R. (1985). Software engineering concepts. McGraw-Hill, Inc.
- Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.

1.5 Overview of document

This document is divided into sections 2, 3, 4 and 5 with intended readers being, the developers and software managers but sections have been written in a manner that it can be understood by anyone having little knowledge about software.

This *Software Design Specification* also includes:

- System architecture description.
- Detailed description of components. (including the template description for each component)
- Reuse and relationships to other products.
- Pseudo codes of components

The design has been made clear, using the class diagram and sequence diagram.

2. System architecture description

2.1 Overview of modules / components

System contains ten modules that are mentioned below:

LOGIN PAGE

In this page user can login/sign up.

REGISTRATION PAGE

In this page if the user doesn't have an account on the website then he/she can register.

FORGOT PASSWORD

If the user forgot password then this page will open and provide the user an OTP for the verification and changing of the password.

FIRST TIME USER

In this page some information will be taken from the first-time user.

Music Player: Controlling of music

In this component the user can do multiple things like play, pause, play previous song /next song, shuffle the songs, etc.

SEARCH AND HISTORY

In this component user can search the songs either by song name /language/artist/genre.
And user can see the recently played songs by them.

LOCAL STORAGE
In this component user can play the songs from their local system.

DATABASE
This will contain all the data.

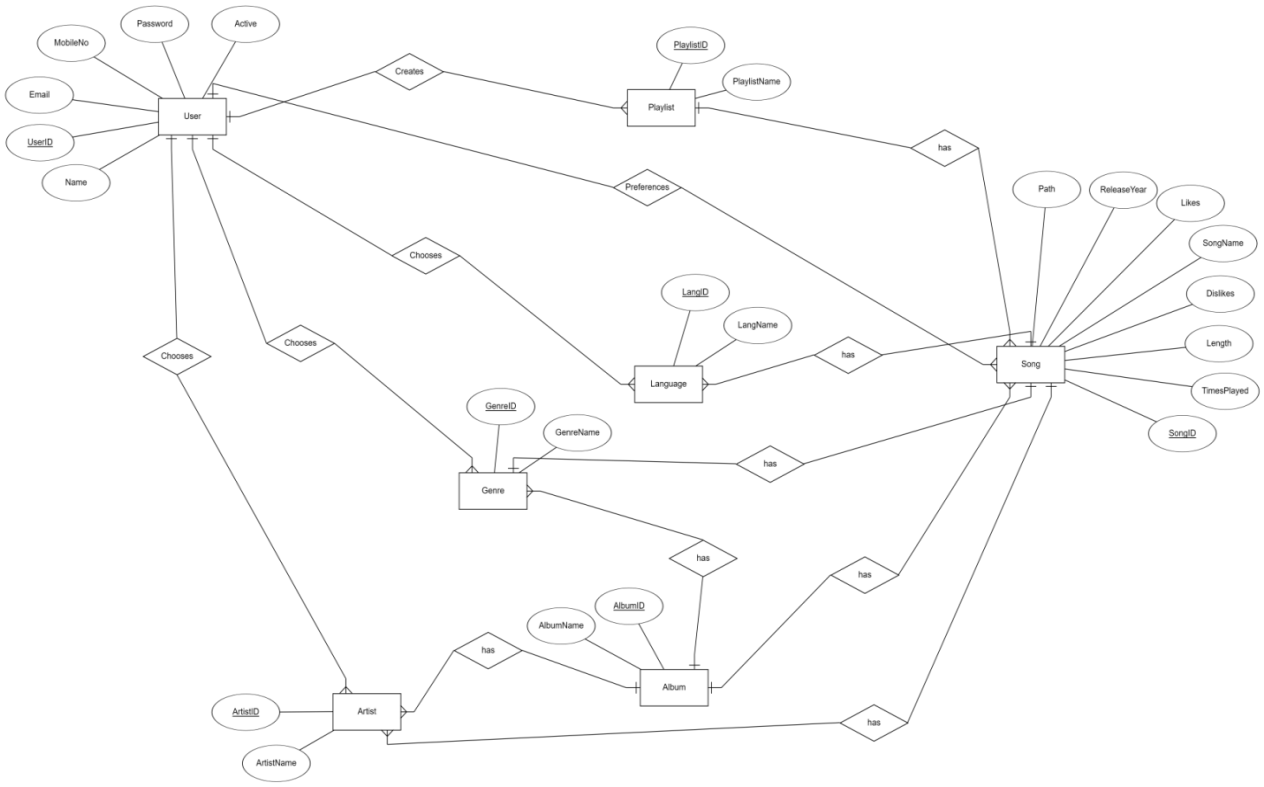
RECOMMENDATION
This component will provide user recommendation on the basis of their favorite language, genre, artist and the songs played recently, their likes and dislikes.

PROFILE
This component is used to edit the information like genre/artist/language, for creating/updating the playlists and for logout.

2.2 Structure and relationships

This Section includes ER Diagram, Sequence Diagram, Use Case Diagram and Activity Diagram of System, that shows Structure and relationships in between components in our System.

ER DIAGRAM OF SYSTEM



Tables

User	
Fields	**Datatype**
UserID	Varchar (100)
Name	Varchar (100)
MobileNo	Varchar (10)
Email	Varchar (100)
Password	Varchar (100)
Active	Int (1)

Song	
Fields	**Datatype**
SongID	Int (10)

Name	Varchar (100)
Path	Varchar (100)
ReleaseYear	Year (4)
Length	Int (3)
Likes	Int (10)
Dislikes	Int (10)
TimesPlayed	Int (10)

Artist	
Fields	**Datatype**
ArtistID	Int (10)
ArtistName	Varchar (100)

Genre	
Fields	**Datatype**
GenreID	Int (10)
GenreName	Varchar (100)

Language	
Fields	**Datatype**
LangID	Int (10)
LangName	Varchar (100)

Album	
Fields	**Datatype**
AlbumID	Int (10)
AlbumName	Varchar (100)

SongArtist	
Fields	**Datatype**
SongID	Int (10)
ArtistID	Int (10)

SongGenre	
Fields	**Datatype**
SongID	Int (10)
GenreID	Int (10)

SongLanguage	
Fields	**Datatype**
SongID	Int (10)
LangID	Int (10)

SongAlbum	
Fields	**Datatype**
SongID	Int (10)
AlbumID	Int (10)

AlbumArtist	
Fields	**Datatype**
AlbumID	Int (10)
ArtistID	Int (10)

AlbumGenre	
Fields	**Datatype**
AlbumID	Int (10)
GenreID	Int (10)

UserSong	
Fields	**Datatype**
UserID	Varchar (100)
SongID	Int (10)
Likes	Int (10)
Dislikes	Int (10)
TimesPlayed	Int (10)

UserArtist	
Fields	**Datatype**
UserID	Varchar (100)
ArtistID	Int (10)

UserGenre	
Fields	**Datatype**
UserID	Varchar (100)
GenreID	Int (10)

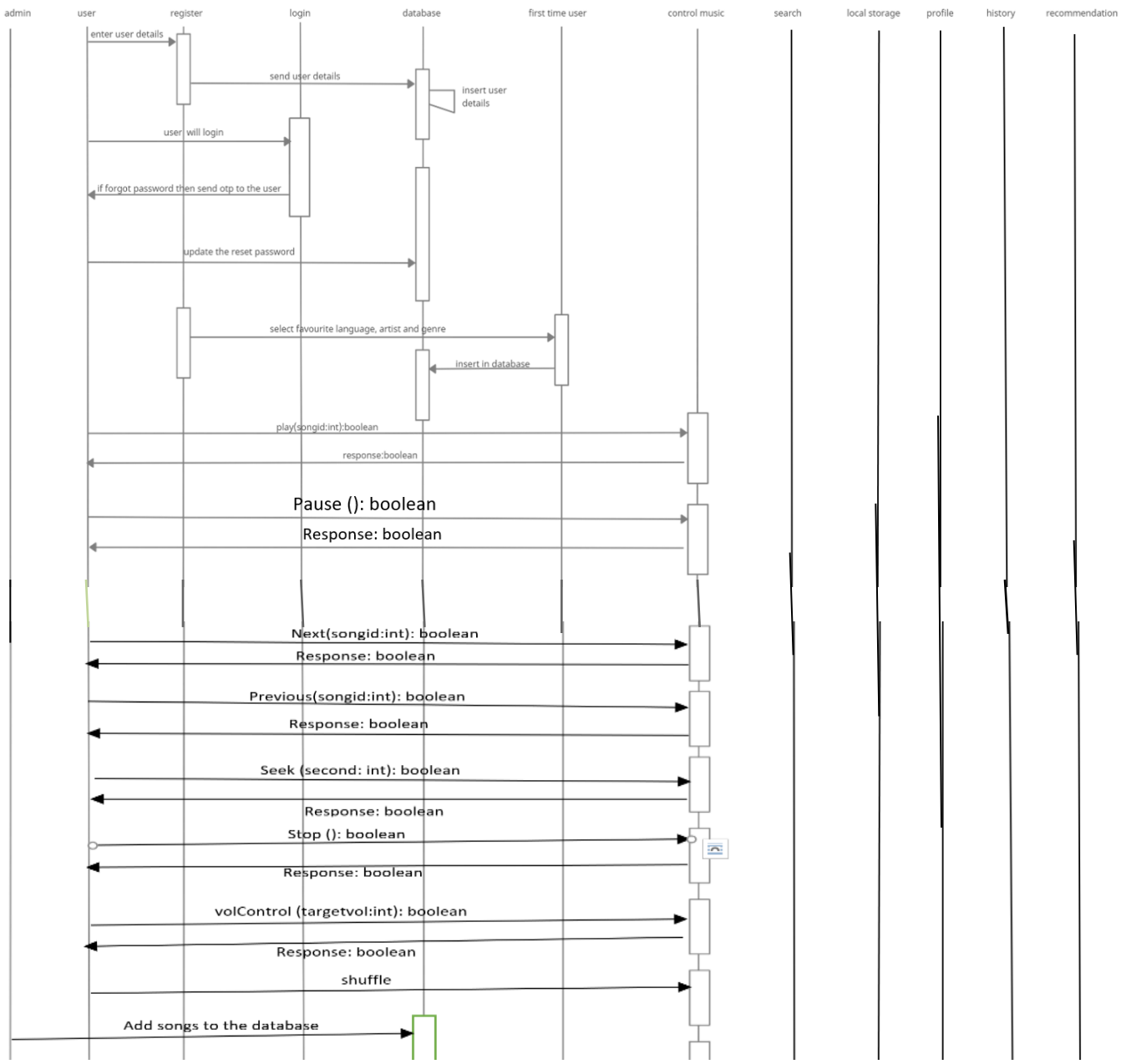
UserLang	
Fields	**Datatype**
UserID	Varchar (100)
LangID	Int (10)

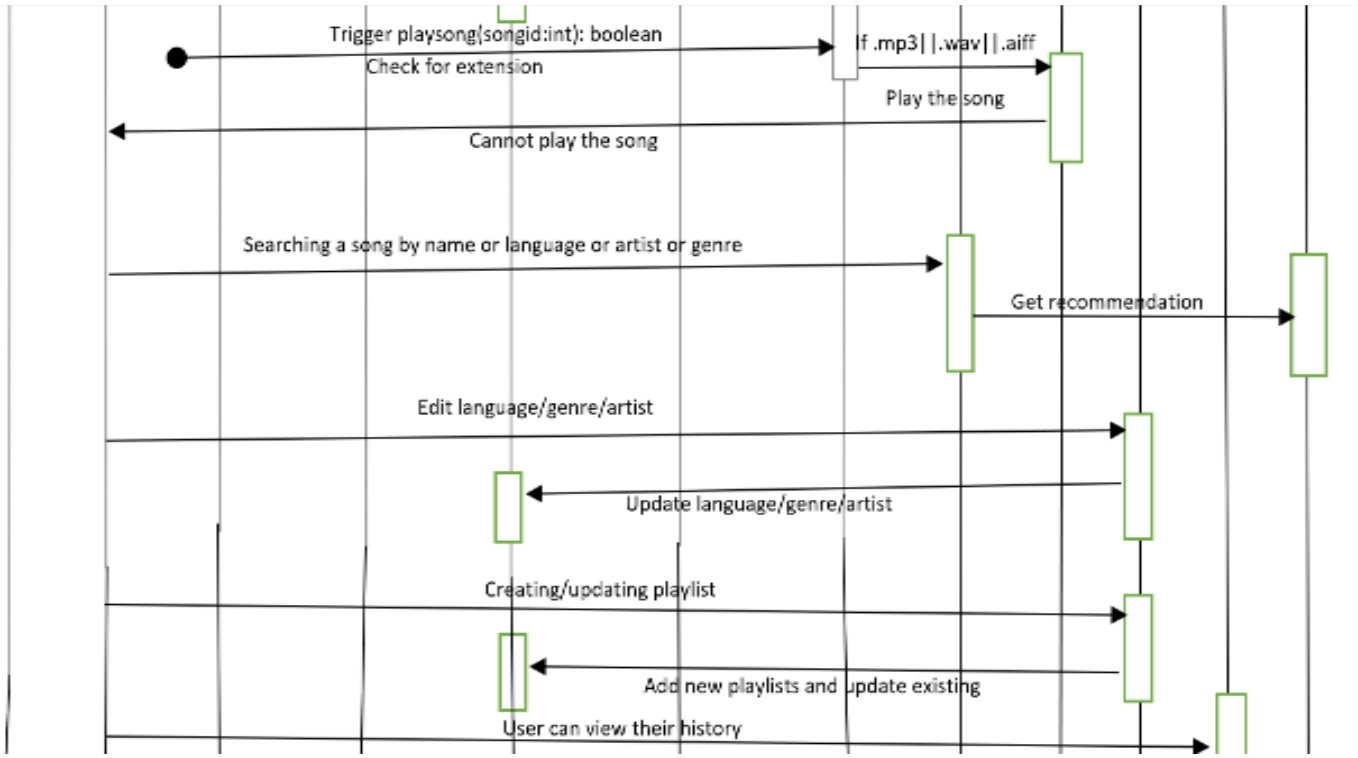
PlaylistUser	
Fields	**Datatype**

PlaylistID	Int (10)
UserID	Varchar (100)
PlaylistName	Varchar (100)

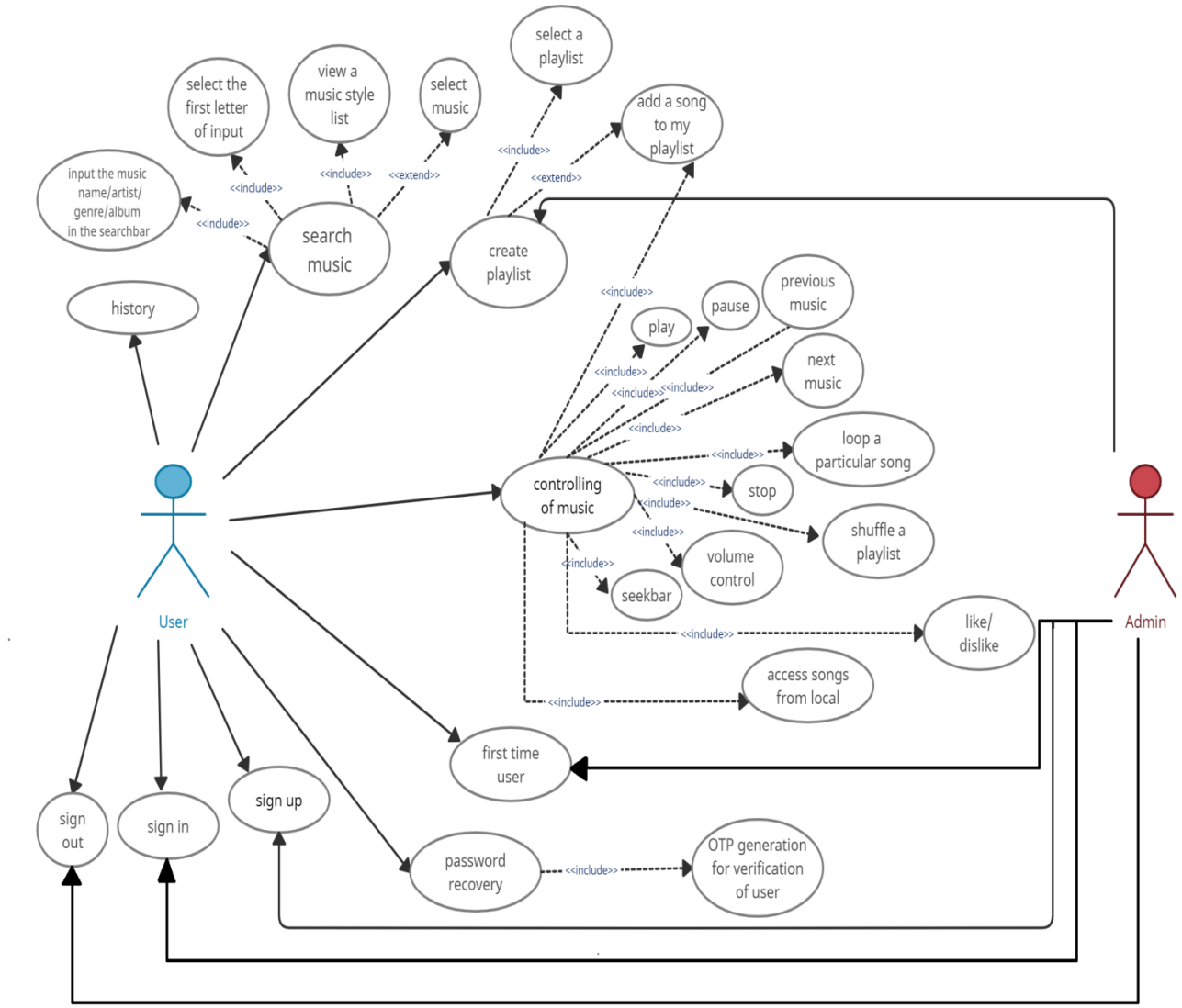
PlaylistSong	
Fields	**Datatype**
PlaylistID	Int (10)
SongID	Int (100)

SEQUENCE DIAGRAM OF THE SYSTEM

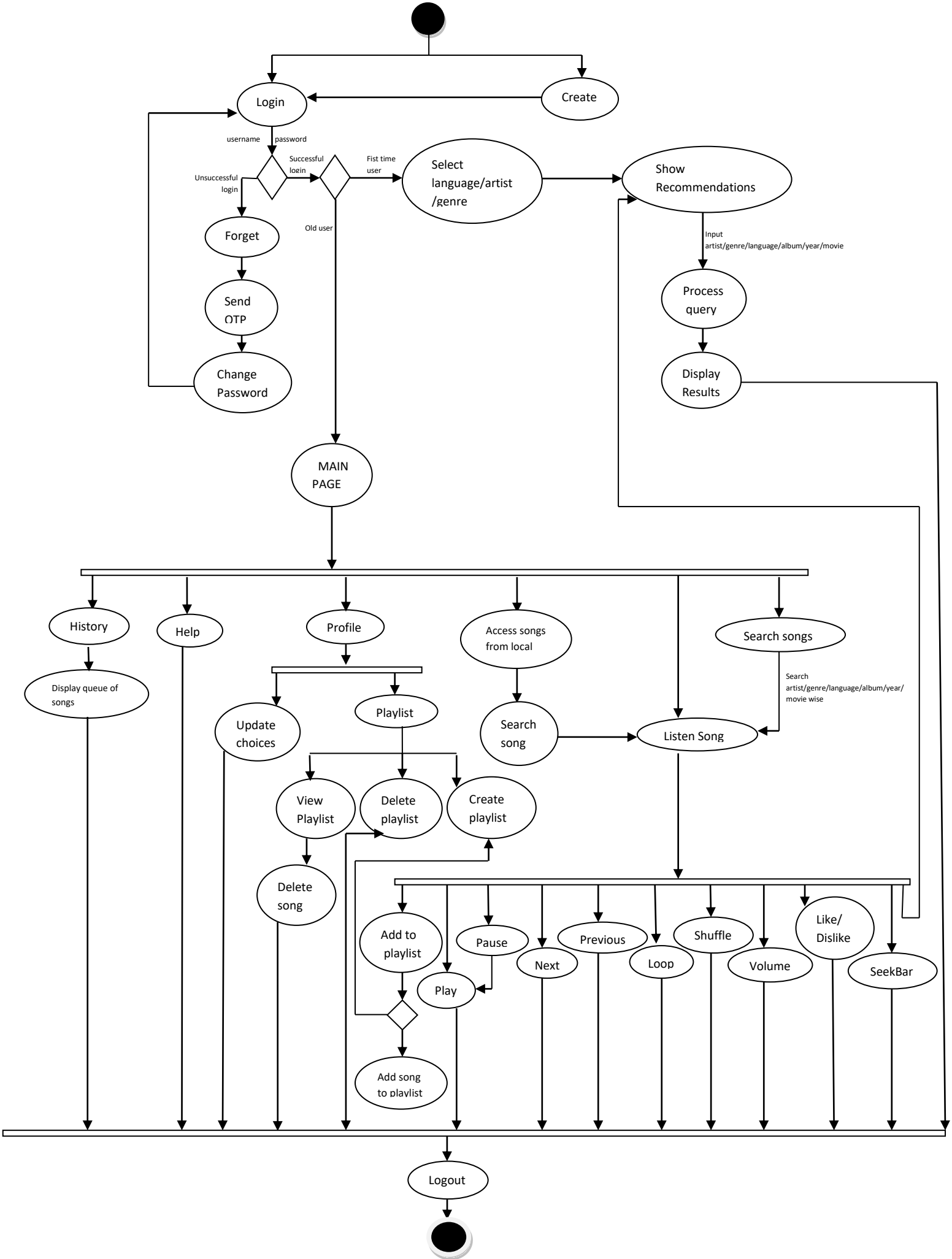

admin
user
register
login
database
first time user
control music
search
local storage
profile
history
recommendation
enter user details
send user details
insert user details
user will login
if forgot password then send otp to the user
update the reset password
select favourite language, artist and genre
insert in database
play(songid:int):boolean
response:boolean
Pause (): boolean
Response: boolean
Next(songid:int): boolean
Response: boolean
Previous(songid:int): boolean
Response: boolean
Seek (second: int): boolean
Response: boolean
Stop (): boolean
Response: boolean
volControl (targetvol:int): boolean
Response: boolean
shuffle
Add songs to the database


Trigger playsong(songid:int): boolean
Check for extension
If .mp3||.wav||.aiff
Play the song
Cannot play the song
Searching a song by name or language or artist or genre
Get recommendation
Edit language/genre/artist
Update language/genre/artist
Creating/updating playlist
Add new playlists and update existing
User can view their history

USE CASE DIAGRAM OF THE SYSTEM



ACTIVITY DIAGRAM OF THE SYSTEM


Login
Create
username
password
Unsuccessful login
Successful login
Fist time user
Old user
Forget
Send OTP
Change Password
Select language/artist /genre
Show Recommendations
Input artist/genre/language/album/year/movie
Process query
Display Results
MAIN PAGE
History
Display queue of songs
Help
Profile
Update choices
Playlist
View Playlist
Delete playlist
Create playlist
Delete song
Access songs from local
Search song
Search songs
Search artist/genre/language/album/year/ movie wise
Listen Song
Add to playlist
Add song to playlist
Play
Pause
Next
Previous
Loop
Shuffle
Volume
Like/ Dislike
SeekBar
Logout

2.3 User interface issues

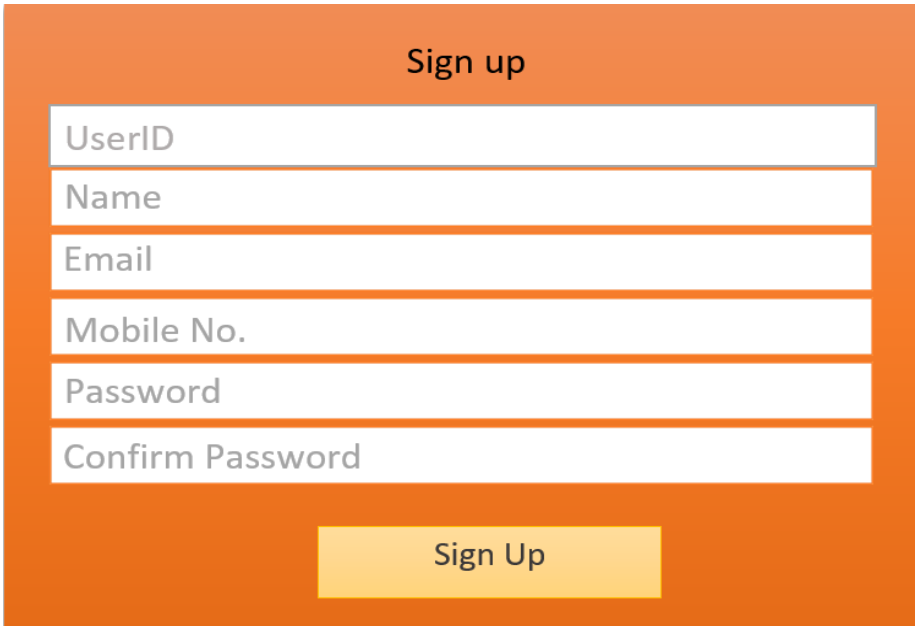
User interface issues

User Interface makes up the greatest component of the web application for the user's experience. Problems with the UI can affect the performance of the web application directly and instantly, becoming a reason for its failure if not checked in time.

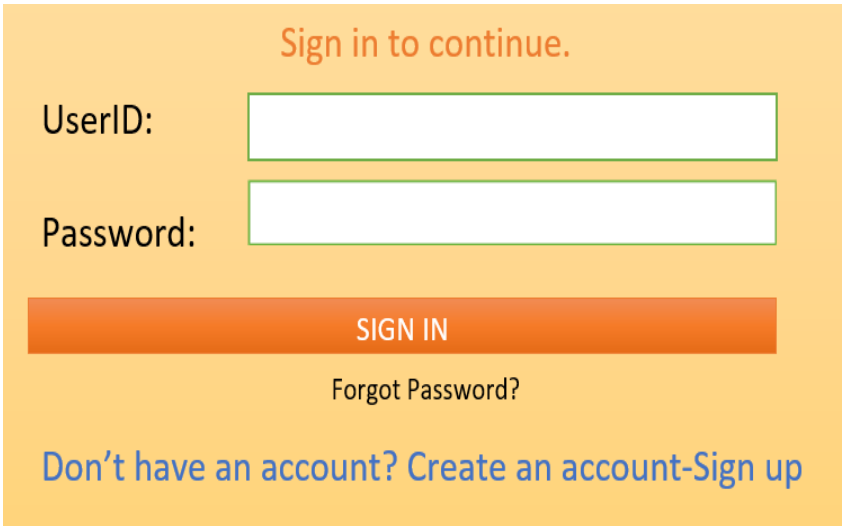
- The UI should be intuitive and UX should be as seamless as possible.
- The UI should be clean and visually engaging with the least cognitive load.
- The web application UI instantly should be able to make the user understand how it works.
- The flow of the app screens should be intuitively oriented with user context and needs.

User interface is implemented in *Java*.

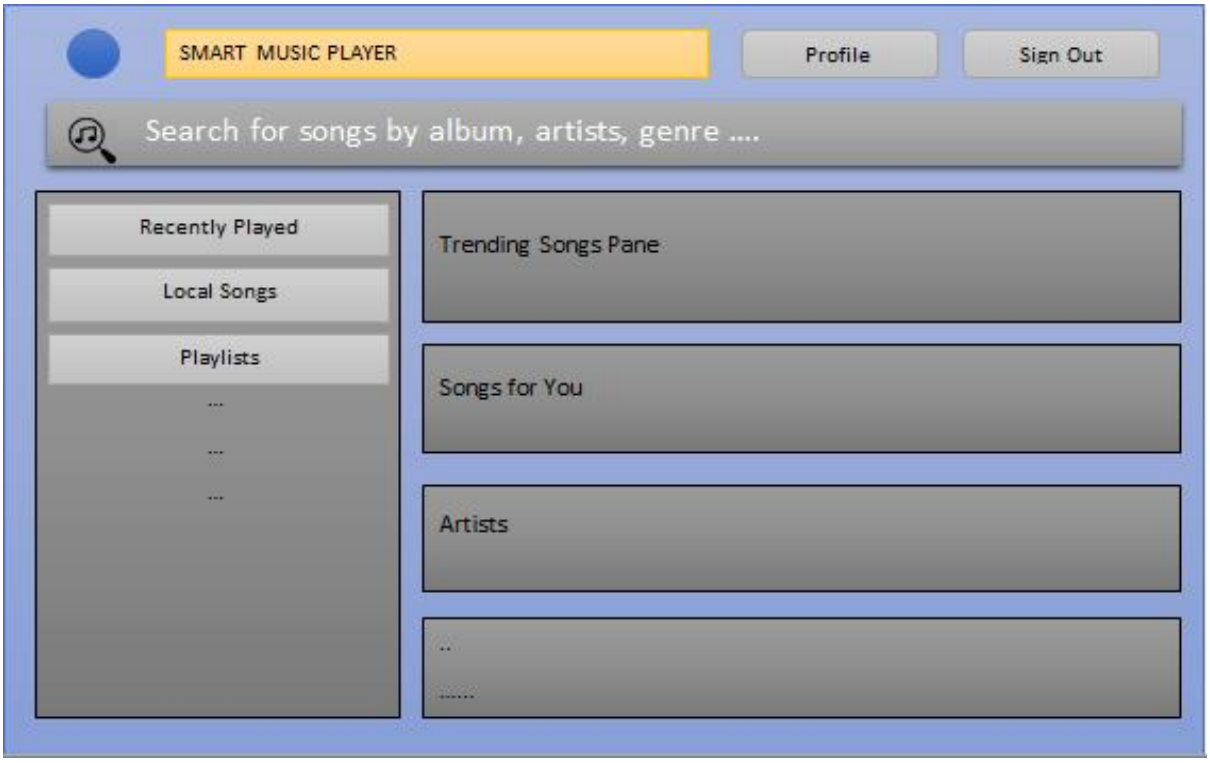
1. CREATE AN ACCOUNT



2. SIGNIN



The front page which interacts with user. It is divided into frames for different functions.



3. Detailed description of components

3.1 Component template description

Description of Component
Component X
Description of Component X
Description of various functions of component X in tabular form

Function	**Description**
Prototype and input parameters	description of function

3.2 MAIN PAGE: Login page/sign in

Function	Description
read (userid: string, password:string):void	Reads the userid and password of the user.
fetch(userdata:string): boolean	Fetch details of user from the database. If the user's details are present in the database then the user is successfully logged in else error is displayed to create the profile.

3.3 Registration page: sign up

Function	Description
read(email: string, username: string, password: string, phoneno: int) :void	Reads the entry of the new user.
write(email:string, username:string, password:string, phoneno:int):Boolean	Writes the new user details in the database. If the details entered by user already exists in database then new entry is not made and user is asked to sign in.

3.4 Forgot Password: password recovery page

Function	Description
read(email:string, phone_no:int): void	Reads the email and phone number of the user.
fetch(phone_no:int):Boolean	If the phone number entered is present in the database then continue else display error as user does not exists.

send_otp(int):Boolean	OTP for password recovery is sent on the registered number. Return true if OTP was send successfully else display error in sending OTP.
read(oldpassword:string, newpassword:string):void	The old and new password is read.
update(newpassword:string): Boolean	Return true if the new password of the user is updated successfully else false.

3.5 First Time User

Function	Description
read(lang1:string, lang2:string, lang3:string): void	Read the language preferences of the user.
write(lang1:string, lang2:string, lang3:string): Boolean	Return true if written in the database successfully; else false and display try again.
read(artist1:string, artist2:string, artist3:string): void	Read the artist preferences of the user.
write(artist1:string, artist2:string, artist3:string): boolean	Return true if written in the database successfully; else false and display try again.
read(genre1:string, genre2:string, genre3:string): void	Read the genre preferences of the user.
write(genre1:string, genre2:string, genre3:string): boolean	Return true if written in the database successfully; else false and display try again.

3.6 Music Player: Controlling of music

Function	Description
Play	Triggered by the user when he/she wants to play a song from his/her list of songs.
Pause	Pause the currently playing music if it is playing.
Next	Play the next song.
Previous	Play the previous song.
Loop	Plays the current song again and again.
Stop	Stops the currently playing song if it is playing.
Shuffle	Shuffles the playlist/recommended song’s randomly.
Volume Control	Controls the volume

Like/Dislike	The user can like/dislike a particular song.
SeekBar	Seeks the currently playing song to the required seconds.
Addtoplaylist	The user's desired song is added in the playlist and the database is updated.
Upload (local songs)	The user can upload a song present in his system, then that song is played by media player.

3.7 SEARCH AND HISTORY

Function	Description
Search(artist,genre, year, album) Fetch(song_id:int):Boolean	The user can search a particular song by the artist name or genre or year or album. Fetches songs with similar song_id/artist_id/language_id/genre_id/album/year
History( )	if any song is being played: add that to the queue of history

3.8 LOCAL STORAGE

Function	Description
Connect():boolean	Creating a connection between the website and the local files that contain music in the user's system
Getsongpath():string	Retrieving the song path from user's file and playing that song using Play() function
Disconnect():boolean	On successfully playing of the song the connection gets disconneted

3.9 DATABASE

Function	Description
Read()	Whenever user searches for a song, calling function passes SongId, it goes and fetches path to that song from the database.
Update()	Calling function passes SongId and Song Metadata. It goes and fetch path to that music file using SongId and performs required

	updation.

3.10 RECOMMENDATION

Function	Description
Fetch(song_id,genre_id,language_id,artist_id, year,duration,album_id):void	takes the current playing track and its information as an input
Recommend(song_database):boolean	outputs set of tracks closely related to the input, fetches songs with similar genres,Singer etc through query

3.11 PROFILE

Function	Description
EditArtist(artist1:string, artist2:string, artist3:string): void	Read the artists preferences from the user and if they are not equal to the previous ones, database will be updated
EditGenre(genre1:string, genre2:string, genre3:string): void	Read the genre preferences from the user and if they are not equal to the previous ones, database will be updated
EditLanguage(lang1:string, lang2:string, lang3:string): void	Read the language preferences from the user and if they are not equal to the previous ones, database will be updated
Createnewplaylist(nameofpl:string):boolean	Creating new playlist for the user by inputing name of the new playlist and then updating user's database
Viewplaylists()	open playlists then shows existing songs from the playlist
Removesongsfromplaylist()	Remove the songs from the playlist
Removeplaylist()	Remove the selected playlist from the playlist
Sign Out/Logout	On clicking the logout button active field will become 0 and the user will be logged out successfully

4.0 Reuse and relationship to other products

If a project is doing some enhancement work, it requires looking into reuse issues. But this project is not doing any enhancement work of existing Software but we are doing enhancement of existing concept of a basic music player, which plays music, with the usual user requirements of playing previous, next songs, adding and removing songs, like and dislike, seek, loop, shuffle and volume control. In addition to that, the software will recommend music too, based on what the user is listening, his artist/genre/languages preferences and the songs the user likes or dislikes.

5 Pseudo codes for components

Login page:

- Get UserID
- Get Password
  - *IF (UserID==EnteredUsername&& Password== EnteredPassword)*
- THEN
  - *Login Successful*
- ELSE
  - *Login Failed*
- ENDIF

Sign up page:

Sign up entities-

- Email field: input type = email, placeholder: “your@email.com”
- UserID field: input type=username, placeholder:” UserID”
- Name field: input type=username, placeholder:” name”
- Password field: input type = password, placeholder: “Password”
- Password confirmation field: input type = password, placeholder: “Password again”
- Phone number field: input type: phone_no, placeholder: “Phone Number”
- Signup submit: value: “Sign Up”, default state, disabled

Logic-

- On leave focus of email field
  - *IF email is blank*
  - Error message: “Please enter an email address.”
  - *ELSE IF email field value is not a valid email address*
  - Error message: “This doesn’t look like an email address. Please try again.”

- On leave focus of username field
  - *IF username is blank*
  - Error message: “Please enter the username.”
  - *ELSE IF username field value already exists*
  - Error message: “Please try another username.”

- On leave focus of password
  - *IF password is not sufficiently strong*
  - Error message: “Please replace with a stronger password.”

- On leave focus of password confirmation
  - *IF password confirmation does not match password*
  - Error message: “Please replace with a stronger password.”

- On leave focus of phone number field
  - *IF phone number is blank*
  - Error message: “Please enter the phone number.”
  - *ELSE IF phone number field value is not a valid email address*
  - Error message: “Please enter a valid 10 digit number.”

- On leave focus of email, username, password , password confirmation or phone number
  - *IF email AND username AND password AND password confirmation AND phone number all contain valid values*

- Enable Signup Submit

Forgot Password:

- Get Email
- Get Phone Number
  - *IF (phone_no==Enteredphone_no)*

- THEN
  - *OTP is sent on the registered phone number*
- ENDIF
  - *IF (OTP==EnteredOTP)*
- THEN
  - *GET old password, new password.*
- ENDIF
- Enable Password Update

First time User:

- Get Language (3 languages) Preferences
- Enable Language Update
- Get Artists (3 artists) Preferences
- Enable Artist Update
- Get Genre (3 genres) Preferences
- Enable Genre Update

Controlling of Music:

- Play Button:
  - Utilizing the addActionListener method, when the user clicks the button it fires an action event.
  - The component with the Listener is registered.
  - The actionPerformed() method is overridden; using play() the song is played.
- Pause Button:
  - The ActionListener interface in the class is implemented.
  - The component with the Listener is registered.
  - The actionPerformed() method is overridden; using pause() the song is paused.
- Next Button:
  - The component with the Listener is registered.
  - The actionPerformed() method is overridden; using next() the next song in queue is played.
- Previous Button:
  - The component with the Listener is registered.
  - The actionPerformed() method is overridden; using previous() the previous song in queue is played.
- Loop Button:

- The component with the Listener is registered.
- The actionPerformed() method is overridden; using loop() the current song is played again and again.

❖ Stop Button:
- The component with the Listener is registered.
- The actionPerformed() method is overridden; using stop() the song is stopped and the current position of the song is set to 0 sec.

❖ Shuffle Button:
- Shuffling of the array of playlist, from index i to index j:
- Shuffle(i,j)
- *if i == j:* return
- Choose a random element, x from i to j
- *Swap a[x] and a[i]*
- *Shuffle(i+1,j)*

❖ Volume Control Slider:
- The initial and final volume values are initialized for the slider.
- Initially the value is set to maximum (1*100 as the value of getVolume() is 1)
- The component with the Listener is registered.
- When the user slides on the volume slider then the value of (getVolume()/100) is set as the new volume.

❖ Like/Dislike Button:
- The component with the Listener is registered.
- If the♡ button is clicked; set the status == “liked”.
- Else; set the status ==“disliked”.

❖ Seekbar Button:
- The component with the Listener is registered.
- Get the duration of the current song in seconds.
- Using seekto(position), seek the music up to the desired position(duration in seconds).

❖ Add to Playlist Button:
- The component with the Listener is registered.
- Get the song’s name and artist, to be added in the playlist.
- Enable update playlist.

❖ Access from Local Button
- The component with the Listener is registered.
- Get the song path.
- Using play(); play the desired song.

❖ Search(name, artist, genre, year, album) TextBox:
- Get the name/artist/genre/year/album of the song.
- Using play(); play the desired song.

Access Local storage:

open a filein your desktop -->click OPEN
check for extension:
if .mp3||.wav||.aiff
play ()
else
ERROR: Can't open the file

History:

if any song is being played:
add that to the queue of history

Search:

- user will type information related to the song like-
  Album Name | Artist Name | Language of the song | Genre of the song | Movie | Year of release in the search bar
- It performs query on the database.
- Fetch the similar song based on either one of them songid/artist/genre/language/year/album and display the best match.

Database:

Admin needs to regularly updateinformation embedded in an audio file that is used to identify the song (like the song title,artist,genre,track length, likes, dislikes etc.) whenever a new song is added in the database.

Read:

- Whenever user searches for a song, calling function passes SongId, it goes and fetches path to that song from the database.
- using that path of file, it reads the Metadata of music file and returns it to calling function.
- It plays the required song.

Update:

- calling function passes SongId and SongMetadata.
- it goes and fetches path to that music file using SongId.
- using that path of file, it updates data about the currently playing file:
  - Whenever user likes/dislikes a song, it should update the database.

Number of times a song has been played to be reflected in the database.

Recommendation:

- When a user is listening to a song, suggestionsof related music for that song appears using it's SongID, ArtistID, Language, Genre, Album, Year, Likes and Dislikes.
- it performs query on the *Database* and fetch relevant songs from it and returns a set of predicted songs
- response *true* on success. The Predicted songs are used to populate the recommended songs view.

Profile:

❖ **EditArtist()**

```
Enter new favourite artist
if new_artist==old_artist
    no changes
else
    artist will be updated
```

❖ **EditGenre()**

```
Enter new favourite genre
if new_gen==old_gen
    no changes
else
    genre will be updated
```

❖ **EditLanguage()**

```
Enter new favourite language
if new_lan==old_lan
    no changes
else
    language will be updated
```

❖ **Createnewplaylist()**

```
Enter new playlist name
then
add songs in the playlist
ask user to add another song
if yes then
    add another song
else
    exit from the playlist
```

- **Addplaylist ()**
  ADD playlist

- **Addsongstoplaylist()**
  add songs to the already exiting playlist

- **Removeplaylist()**
  Remove playlist from the existing playlists

- **Viewplaylists()**
  open playlists then
  all the existing playlists will be displayed

- **Logout()**
  on clicking the logout active will be 0 and the user logged out successfully