PROJECT DESCRIPTION

- ✓ MusicApp is a desktop-based music player and playlist management application developed using Java Swing, with integrated music playback functionality powered by the JavaZoom JLayer library.
- ✓ The primary goal of the app is to allow users to organize, store, and play their favourite songs via playlists, with persistence ensured by connecting the app to a MySQL database.
- ✓ The application provides a simple yet interactive graphical user interface (GUI), making it user-friendly for music enthusiasts who want to create custom playlists and enjoy their music collection without the need for an internet connection.

KEY FEATURES

1. Playlist Management:
   Users can create and delete playlists. Songs can be linked to specific playlists, and playlists are stored in a MySQL database.
2. Song Management:
   Songs can be added by selecting files from the system, and they are stored in the database. Users can also remove songs from playlists and the database.
3. Music Playback:
   The app plays MP3 files from the database using the JLayer library. Playback is handled on a separate thread to keep the interface responsive, with basic controls like play and stop.

TECHNICAL STACK

- **Frontend/GUI:** Java Swing for an intuitive and interactive interface.
- **Backend:** MySQL for database storage of playlists and song metadata.
- **Audio Playback:** JavaZoom JLayer library for MP3 playback functionality.
- **Multithreading:** Playback is managed on a separate thread to ensure the application remains responsive during song playback.

SYSTEM REQUIREMENTS

- **JDK Version :** JDK 17
- **Database :** MySQL
- **IDE :** Visual Studio and Command Prompt

DATABASE DESIGN :

MySQL:

MySQL is an open-source relational database management system (RDBMS) developed by Oracle Corporation.

- It uses Structured Query Language (SQL) for database management and is known for its reliability, speed and ease of use.
- MySQL is widely used for various applications, from small websites to large-scale enterprise systems

1) Database Name: songs

This is the main database that will contain several tables to manage playlists, songs, and the relationships between them.

2) Tables in the songs Database

a. Playlists Table

This table will store the details of each playlist.

Columns:

playlist_id: A unique identifier for each playlist (Primary Key, INT, Auto Increment).

name: The name of the playlist (VARCHAR(255)).

PlayLists table Fields:

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	playlist_id	songs	playlists	INT	binary	11	1	
2	name	songs	playlists	VARCHAR	utf8mb4	255	6	
3	description	songs	playlists	TEXT	utf8mb4	65535	0	
4	created_at	songs	playlists	TIMESTAMP	binary	19	19	

PlayLists table Data:

```
1 •  USE songs;
2 •  select * from playlists;
3
```

Result Grid | Filter Rows: | Edit: | Export/Import:

playlist_id	name	description	created_at
4	karan	NULL	2024-10-13 16:51:35
5	karan2	NULL	2024-10-13 17:02:38
8	image	NULL	2024-10-16 22:36:05
9	img	NULL	2024-10-16 23:03:01
NULL	NULL	NULL	NULL

SQL to Create the Table:

```
CREATE TABLE Playlists (
    playlist_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL
);
```

b. Songs Table

This table will store song details, including the audio file and optional album art.

Columns:

song_id: A unique identifier for each song (Primary Key, INT, Auto Increment).

title: The title of the song (VARCHAR(255)).

song_file: The binary data of the MP3 file (LONGBLOB).

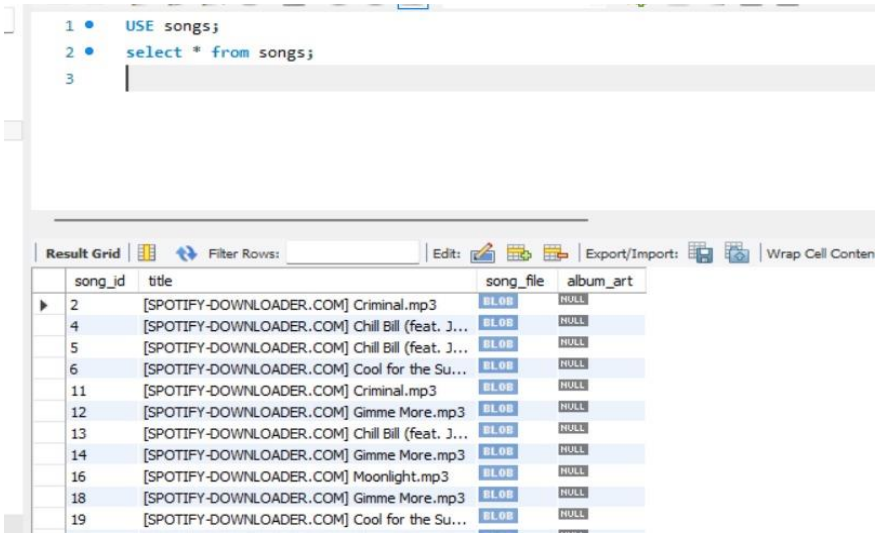
album_art: The binary data for the album art (image) (LONGBLOB).

Songs table fields:

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	song_id	songs	songs	INT	binary	11	2	
2	title	songs	songs	VARCHAR	utf8mb4	255	127	
3	song_file	songs	songs	BLOB	binary	-1	5300017	
4	album_art	songs	songs	BLOB	binary	-1	459947	

Songs table Data:



SQL to Create the Table:

```
CREATE TABLE Songs (
    song_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    song_file LONGBLOB NOT NULL,
    album_art LONGBLOB
 );
```

c. PlaylistSongs Table

This is a junction table that manages the many-to-many relationship between playlists and songs. A playlist can contain multiple songs, and a song can be part of multiple playlists.

Columns:

playlist_id: Foreign key referring to the Playlists table (INT)

song_id: Foreign key referring to the Songs table (INT)

PlaylistSongs table fields:

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	playlist_id	songs	playlistsongs	INT	binary	11	1	
2	song_id	songs	playlistsongs	INT	binary	11	2	

PlayListSongs table Data:

```
1 • USE songs;
2 • select * from playlistsongs;
3
```

Result Grid | Filter Rows: | Edit: | Export/Import:

playlist_id	song_id
5	2
4	4
5	6
9	12
5	21
NULL	NULL

SQL to Create the Table:

```
CREATE TABLE PlaylistSongs (
    playlist_id INT,
    song_id INT,
FOREIGN KEY (playlist_id) REFERENCES Playlists(playlist_id) ON DELETE CASCADE,
FOREIGN KEY (song_id) REFERENCES Songs(song_id) ON DELETE CASCADE,
PRIMARY KEY (playlist_id, song_id)
);
```

3) Relationships

- One-to-Many (Playlists to Songs via PlaylistSongs Table):
- Each playlist can contain multiple songs (many-to-many).
- The PlaylistSongs table bridges the relationship between playlists and songs, mapping each playlist to its associated songs.

4)Database Schema Overview

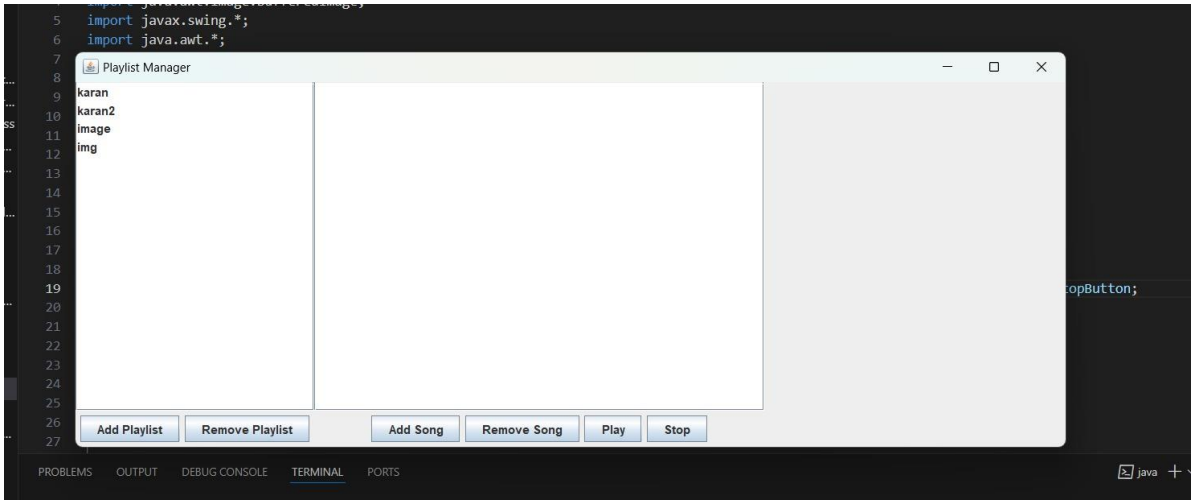
Database: songs

- ├── Playlists (playlist_id, name)
- ├── Songs (song_id, title, song_file, album_art)
- └── PlaylistSongs (playlist_id, song_id)

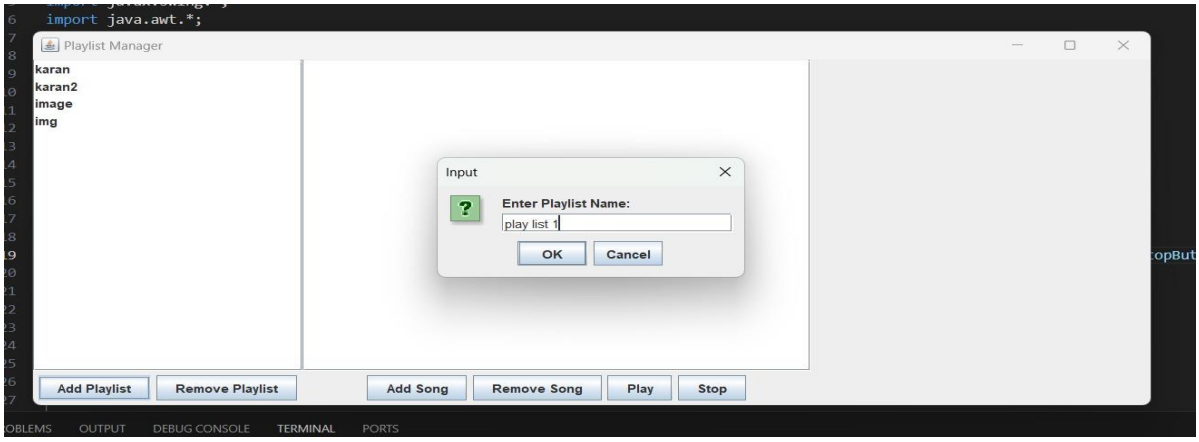
- ✓ Playlists stores the name of each playlist.
- ✓ Songs stores song information, including the title, the actual MP3 file as binary data, and optional album art as binary data.
- ✓ PlaylistSongs is the junction table that links a song with one or more playlists, making it easy to organize songs into different playlists.

GUI DESIGN:

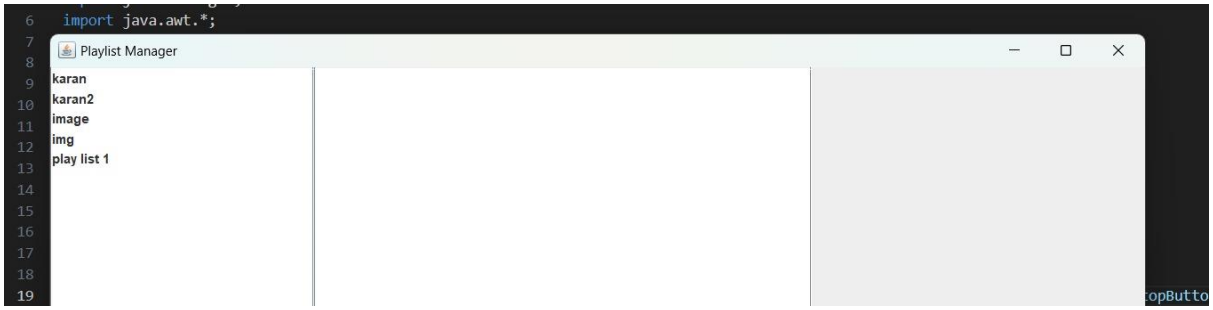
- This is the home page



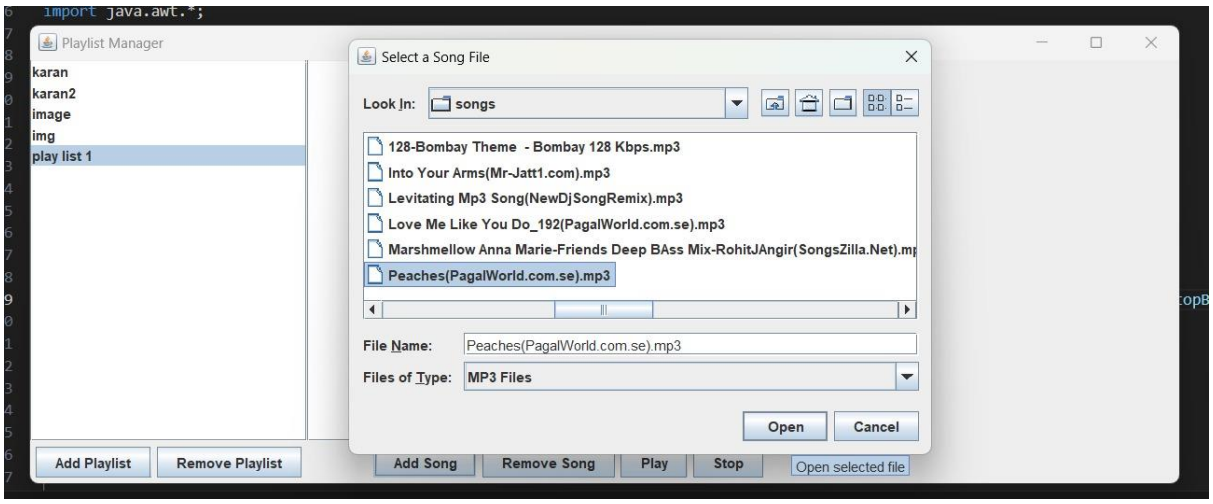
- Creating a playlist



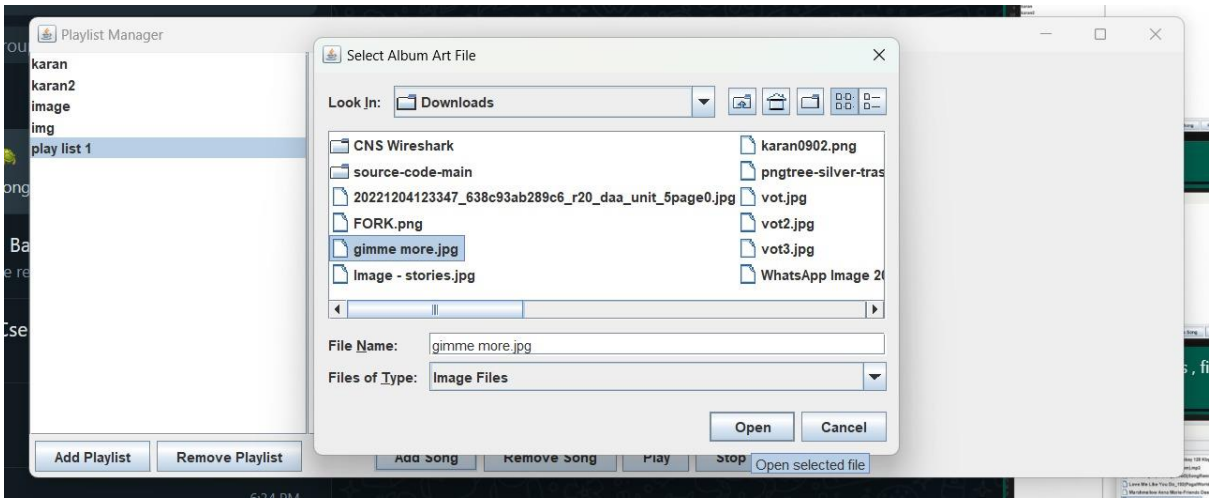
- Playlist was created



- Adding song to playlist , first select the playlist to add the song , then click the song to add

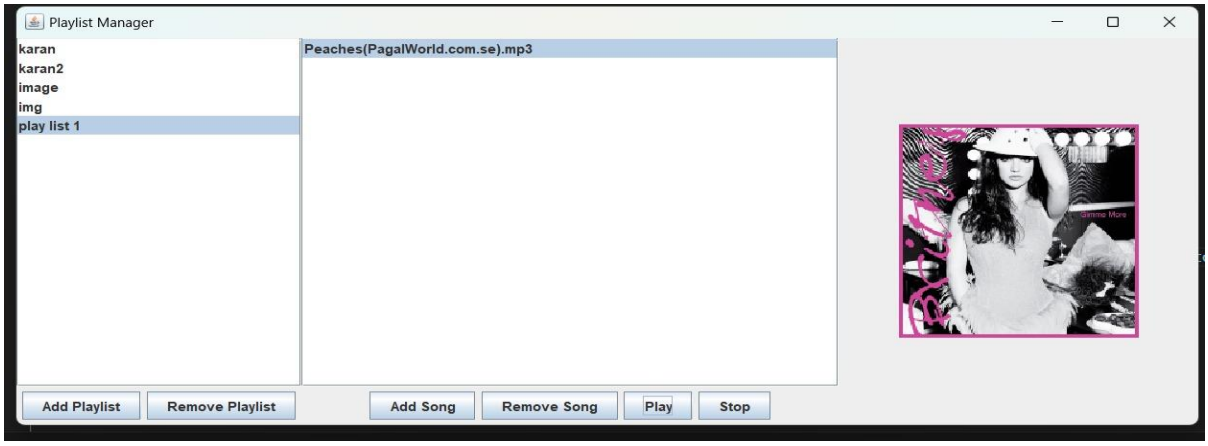


- Adding image to the song

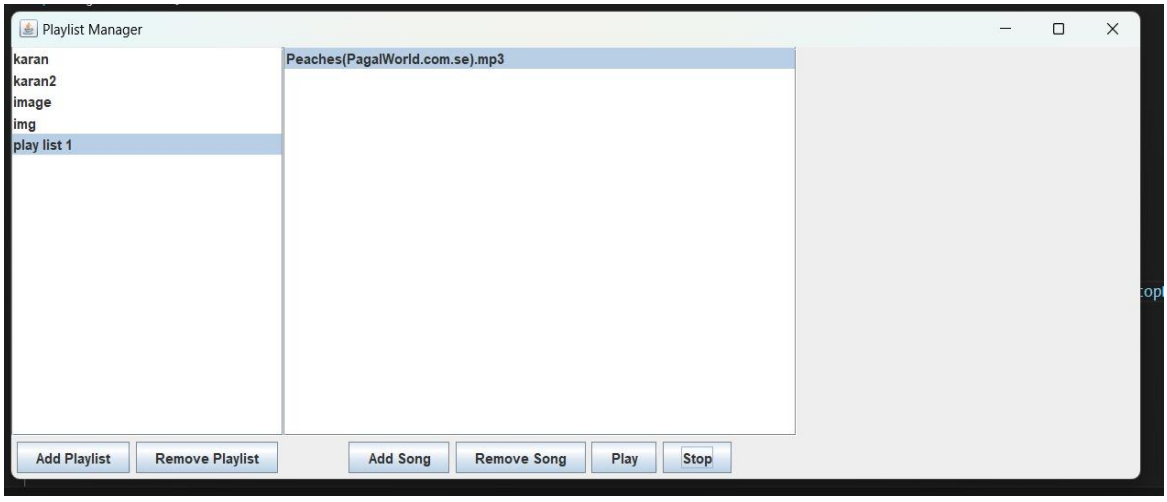


- Song added to Playlist

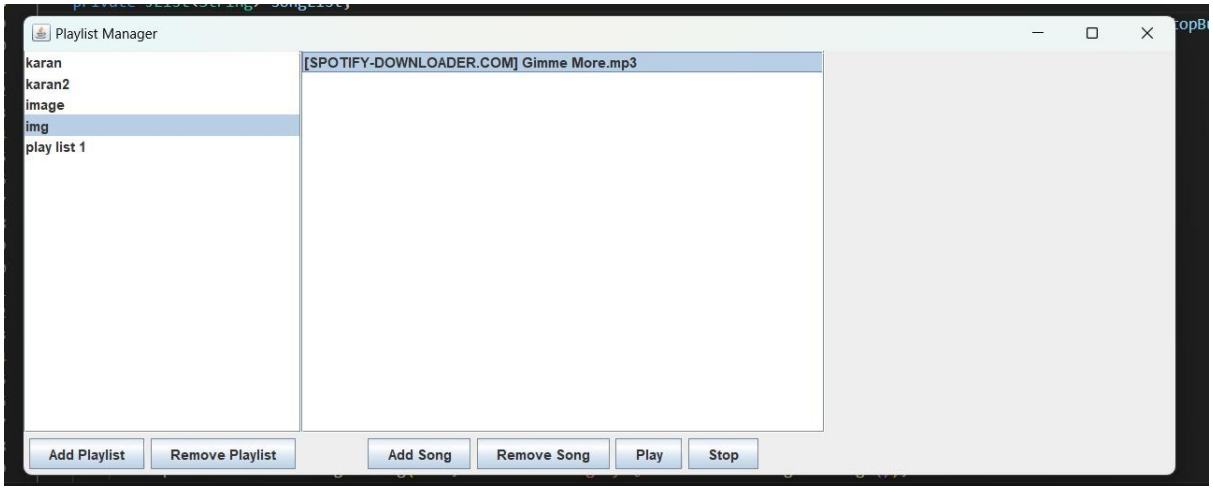
- When song is played , corresponding image will be opened

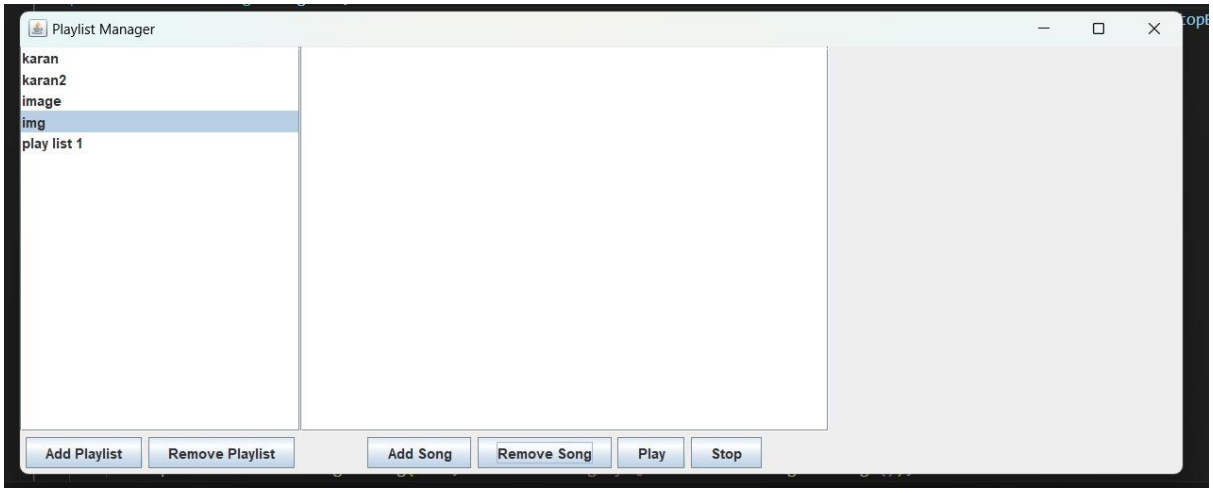


- When you click stop , song and image will be closed

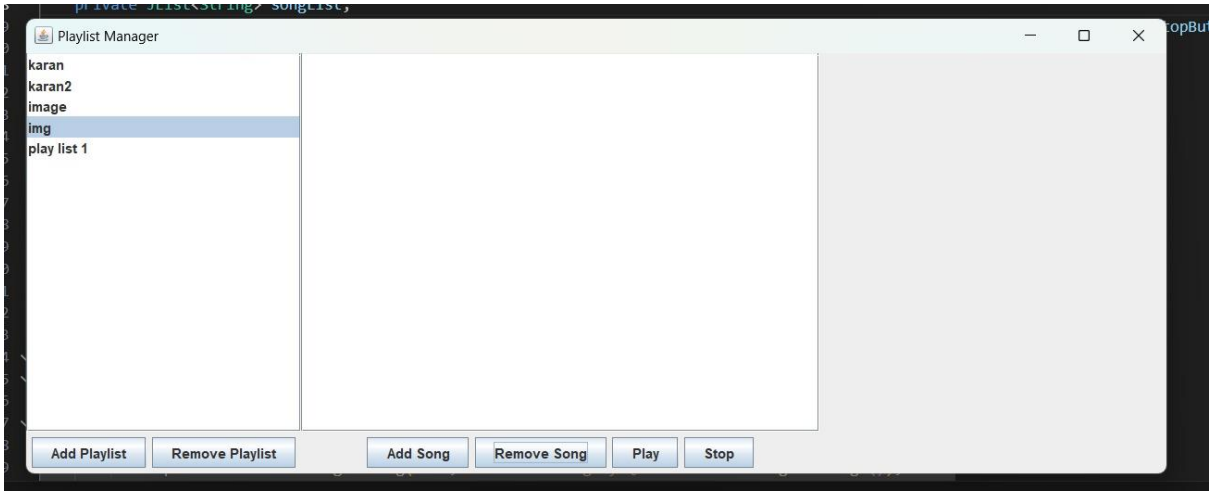


- Removing song
    - Select the song to be removed
    - Click Remove song button . Now the song will be removed





- Removing Playlist
    - Select the Playlist to be removed
    - Click Remove Playlist button . Now the playlist will be removed



CODE:

```
import javazoom.jl.decoder.JavaLayerException;
import javazoom.jl.player.Player;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.sql.*;
import java.util.concurrent.atomic.AtomicBoolean;
import javax.swing.filechooser.FileNameExtensionFilter;

public class MusicApp extends JFrame {
    private DefaultListModel<String> playlistModel;
    private DefaultListModel<String> songModel;
    private JList<String> playlistList;
    private JList<String> songList;
    private JButton addPlaylistButton, removePlaylistButton, addSongButton, removeSongButton, playButton,
togglePauseButton, stopButton;
    private JLabel albumArtLabel;
    private Player mp3Player;
    private Thread playerThread;
    private boolean isPlaying = false;
    private boolean isPaused = false;
    private String currentSongTitle;
    private InputStream currentSongInputStream;
    private AtomicBoolean stopFlag = new AtomicBoolean(false);
    private final String DB_URL = "jdbc:mysql://localhost:3306/songs";
    private final String USER = "root";
    private final String PASSWORD = "karan5686";
    public MusicApp() {
```

```
try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading MySQL Driver: " + e.getMessage());
        }

        setTitle("Playlist Manager");
        setSize(1000, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());

        albumArtLabel = new JLabel();
        albumArtLabel.setHorizontalAlignment(JLabel.CENTER);
        albumArtLabel.setPreferredSize(new Dimension(300, 300)); // Adjust size as needed
        add(albumArtLabel, BorderLayout.EAST);

        JPanel playlistPanel = new JPanel();
        playlistPanel.setLayout(new BorderLayout());

        playlistModel = new DefaultListModel<>();
        playlistList = new JList<>(playlistModel);
        playlistList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        playlistList.addListSelectionListener(e -> {
            if (!e.getValueIsAdjusting()) {
                String selectedPlaylist = playlistList.getSelectedValue();
                loadSongsForPlaylist(selectedPlaylist);
            }
        });

        JScrollPane playlistScroll = new JScrollPane(playlistList);
        playlistPanel.add(playlistScroll, BorderLayout.CENTER);
```

```
        JPanel playlistButtonPanel = new JPanel();
        addPlaylistButton = new JButton("Add Playlist");
        removePlaylistButton = new JButton("Remove Playlist");
        addPlaylistButton.addActionListener(e -> addPlaylist());
        removePlaylistButton.addActionListener(e -> removePlaylist());
        playlistButtonPanel.add(addPlaylistButton);
        playlistButtonPanel.add(removePlaylistButton);
        playlistPanel.add(playlistButtonPanel, BorderLayout.SOUTH);
        add(playlistPanel, BorderLayout.WEST);
        JPanel songPanel = new JPanel();
        songPanel.setLayout(new BorderLayout());
        songModel = new DefaultListModel<>();
        songList = new JList<>(songModel);
        songList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        JScrollPane songScroll = new JScrollPane(songList);
        songPanel.add(songScroll, BorderLayout.CENTER);
        JPanel songButtonPanel = new JPanel();
        addSongButton = new JButton("Add Song");
        removeSongButton = new JButton("Remove Song");
        playButton = new JButton("Play");
        stopButton = new JButton("Stop");
        addSongButton.addActionListener(e -> addSong());
        removeSongButton.addActionListener(e -> removeSong());
        playButton.addActionListener(e -> playSong());
        stopButton.addActionListener(e -> stopSong());
        songButtonPanel.add(addSongButton);
        songButtonPanel.add(removeSongButton);
        songButtonPanel.add(playButton);
        songButtonPanel.add(stopButton);
        songPanel.add(songButtonPanel, BorderLayout.SOUTH);
        add(songPanel, BorderLayout.CENTER);
        loadPlaylists();
    }
    private Connection getConnection() throws SQLException {
```

```
    return DriverManager.getConnection(DB_URL, USER, PASSWORD);
}
private void loadPlaylists() {
    try (Connection conn = getConnection();
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT name FROM Playlists")) {
        while (rs.next()) {
            playlistModel.addElement(rs.getString("name"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error loading playlists: " + e.getMessage());
    }
}

private void loadSongsForPlaylist(String playlistName) {
    songModel.clear();
    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement(
                 "SELECT Songs.title FROM PlaylistSongs " +
                         "JOIN Songs ON PlaylistSongs.song_id = Songs.song_id " +
                         "JOIN Playlists ON PlaylistSongs.playlist_id = Playlists.playlist_id " +
                         "WHERE Playlists.name = ?")) {
        stmt.setString(1, playlistName);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            String songTitle = rs.getString("title");
            songModel.addElement(songTitle);
        }
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error loading songs: " + e.getMessage());
    }    }
private void addPlaylist() {
```

```
    String playlistName = JOptionPane.showInputDialog(this, "Enter Playlist Name:");
    if (playlistName != null && !playlistName.isEmpty()) {
        try (Connection conn = getConnection();
             PreparedStatement stmt = conn.prepareStatement("INSERT INTO Playlists (name) VALUES (?)")) {
            stmt.setString(1, playlistName);
            stmt.executeUpdate();
            playlistModel.addElement(playlistName);
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error adding playlist: " + e.getMessage());
        }
    }
}

private void removePlaylist() {
    int selectedIndex = playlistList.getSelectedIndex();
    if (selectedIndex != -1) {
        String playlistName = playlistModel.get(selectedIndex);
        try (Connection conn = getConnection();
             PreparedStatement stmt = conn.prepareStatement("DELETE FROM Playlists WHERE name = ?")) {
            stmt.setString(1, playlistName);
            stmt.executeUpdate();
            playlistModel.remove(selectedIndex);
            songModel.clear(); // Clear songs when playlist is removed
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error removing playlist: " + e.getMessage());
        }
    }
}

private void addSong() {
 JFileChooser fileChooser = new JFileChooser();
 fileChooser.setDialogTitle("Select a Song File");
```

```
// Create a filter for MP3 files
FileNameExtensionFilter filter = new FileNameExtensionFilter("MP3 Files", "mp3");
fileChooser.setFileFilter(filter);
int userSelection = fileChooser.showOpenDialog(this);
if (userSelection == JFileChooser.APPROVE_OPTION) {
    File songFile = fileChooser.getSelectedFile();
    String title = songFile.getName();
    // Add another JFileChooser for album art selection
    JFileChooser albumArtChooser = new JFileChooser();
    albumArtChooser.setDialogTitle("Select Album Art File");
    FileNameExtensionFilter albumArtFilter = new FileNameExtensionFilter("Image Files", "jpg", "jpeg",
    "png","jfif");
    albumArtChooser.setFileFilter(albumArtFilter);
    File albumArtFile = null; // Initialize albumArtFile variable
    int albumArtSelection = albumArtChooser.showOpenDialog(this);
    if (albumArtSelection == JFileChooser.APPROVE_OPTION) {
        albumArtFile = albumArtChooser.getSelectedFile();
    }
    try (Connection conn = getConnection();
         PreparedStatement stmt = conn.prepareStatement("INSERT INTO Songs (title, song_file, album_art)
         VALUES (?, ?, ?)")) {
        stmt.setString(1, title);
        stmt.setBlob(2, new FileInputStream(songFile)); // Store the song file
        if (albumArtFile != null) {
            stmt.setBlob(3, new FileInputStream(albumArtFile)); // Store the album art
            System.out.println("Album art stored: " + albumArtFile.getName());
        }
         else {
            stmt.setNull(3, java.sql.Types.BLOB); // Set to null if no album art is being added
            System.out.println("No album art provided.");
        }
        stmt.executeUpdate(); // Execute the insert
        songModel.addElement(title); // Update the song model
        // Add the song to the selected playlist
```

```
            String selectedPlaylist = playlistList.getSelectedValue();
            if (selectedPlaylist != null) {
                int songId;
                try (PreparedStatement songIdStmt = conn.prepareStatement("SELECT song_id FROM Songs
                WHERE title = ?")) {
                    songIdStmt.setString(1, title);
                    ResultSet rs = songIdStmt.executeQuery();
                    if (rs.next()) {
                        songId = rs.getInt("song_id");
                        try (PreparedStatement mappingStmt = conn.prepareStatement("INSERT INTO PlaylistSongs
                        (playlist_id, song_id ) VALUES ((SELECT playlist_id FROM Playlists WHERE name = ?), ?)"))
                        {
                            mappingStmt.setString(1, selectedPlaylist);
                            mappingStmt.setInt(2, songId);
                            mappingStmt.executeUpdate();
                        }
                    }
                }
            } catch (SQLException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "Database error adding song: " + e.getMessage());
            } catch (FileNotFoundException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "File not found: " + e.getMessage());
            }
        }
    }

    private void removeSong() {
        int selectedIndex = songList.getSelectedIndex();
        if (selectedIndex != -1) {
            String songTitle = songModel.get(selectedIndex);
            try (Connection conn = getConnection()) {
```

```
            if (isPlaying && songTitle.equals(currentSongTitle)) {
                stopSong(); // Stop the current song if it's the one being removed
            }
            int songId;
            try (PreparedStatement getIdStmt = conn.prepareStatement("SELECT song_id FROM Songs WHERE
            title = ?")) {
                getIdStmt.setString(1, songTitle);
                ResultSet rs = getIdStmt.executeQuery();
                if (rs.next()) {
                    songId = rs.getInt("song_id");
                    try (PreparedStatement deleteMappingStmt = conn.prepareStatement("DELETE FROM
                    PlaylistSongs WHERE song_id = ?")) {
                        deleteMappingStmt.setInt(1, songId);
                        deleteMappingStmt.executeUpdate();
                    }
                    try (PreparedStatement deleteSongStmt = conn.prepareStatement("DELETE FROM Songs
                    WHERE song_id = ?")) {
                        deleteSongStmt.setInt(1, songId);
                        deleteSongStmt.executeUpdate();
                    }
                    songModel.remove(selectedIndex);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error removing song: " + e.getMessage());
        }
    }
}
  private void displayImageFromByteArray(byte[] imageData, JLabel label) {
     ImageIcon icon = new ImageIcon("gimme more.jpg");
if (imageData != null && imageData.length > 0) {
    try {
```

```
            BufferedImage albumArtImage = ImageIO.read(new ByteArrayInputStream(imageData));
            if (albumArtImage != null) {
                // Resize the image if necessary
                Image scaledImage = albumArtImage.getScaledInstance(200, 200, Image.SCALE_SMOOTH);
                label.setIcon(new ImageIcon(scaledImage));
                System.out.println("Image loaded successfully.");
            } else {
                System.out.println("Image is null after reading.");
                label.setIcon(null); // Clear if no image
            }
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(label, "Error loading image: " + e.getMessage());
            label.setIcon(null); // Clear icon on error
        }
    } else {
        System.out.println("Byte array is null or empty.");
        label.setIcon(icon); // Clear if no image
    }
}

private void playSong() {
    int selectedIndex = songList.getSelectedIndex();
    if (selectedIndex != -1) {
        String songTitle = songModel.get(selectedIndex);
        try {
            // Stop the currently playing song if there's one
            if (isPlaying) {
                stopSong();
            }
            // Get the song file and album art from the database
            try (Connection conn = getConnection();
                 PreparedStatement stmt = conn.prepareStatement("SELECT song_file, album_art FROM Songs
                WHERE title = ?")) {
```

```
stmt.setString(1, songTitle);
ResultSet rs = stmt.executeQuery();
if (rs.next()) {
    currentSongInputStream = rs.getBinaryStream("song_file");
    currentSongTitle = songTitle;

    // Load and display album art using the new function
    byte[] albumArtData = rs.getBytes("album_art");
    System.out.print(albumArtData);
    if (albumArtData != null) {
        System.out.println("Album art data retrieved successfully, size: " + albumArtData.length);
    } else {
        System.out.println("Album art data is null or empty.");
    }
    displayImageFromByteArray(albumArtData, albumArtLabel);

    // Create a new Player instance
    try {
        mp3Player = new Player(currentSongInputStream);
        isPlaying = true;
        isPaused = false; // Reset pause state
        stopFlag.set(false);

        // Create a new thread to play the song
        playerThread = new Thread(() -> {
            try {
                mp3Player.play();
            } catch (JavaLayerException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "Error playing song: " + e.getMessage());
            } finally {
                isPlaying = false;
                currentSongInputStream = null; // Clear input stream
                currentSongTitle = null; // Clear current song title
```

```
                        albumArtLabel.setIcon(null); // Clear album art when done
                    }
                });
                playerThread.start();
            } catch (JavaLayerException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "Error initializing player: " + e.getMessage());
            }
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(this, "Error loading song: " + e.getMessage());
}
}

private void stopSong() {
    if (isPlaying) {
        stopFlag.set(true);
        mp3Player.close();
        isPlaying = false;
        currentSongInputStream = null; // Clear input stream
        currentSongTitle = null; // Clear current song title
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        MusicApp app = new MusicApp();
        app.setVisible(true);
    });
}
}
```

CONCLUSION

MusicApp provides a user-friendly solution for managing and playing music with an intuitive interface and robust backend. Users can create and organize playlists, add or remove songs, and play MP3 files. The MySQL database ensures secure and persistent storage of playlists and songs, while the JavaZoom JLayer library enables smooth playback. The efficient database design ensures data consistency, scalability, and reduced redundancy. MusicApp integrates music playback and playlist management into a scalable and functional desktop application.