UE21CS351A - DBMS Mini-Project Report
Project Report
MusicXperience

Team Members:

Name	**SRN**
SHREYA KASHYAP	PES1UG21CS913
SAI MANASA NADIMPALLI	PES1UG21CS910

Table of Contents :

I. Abstract

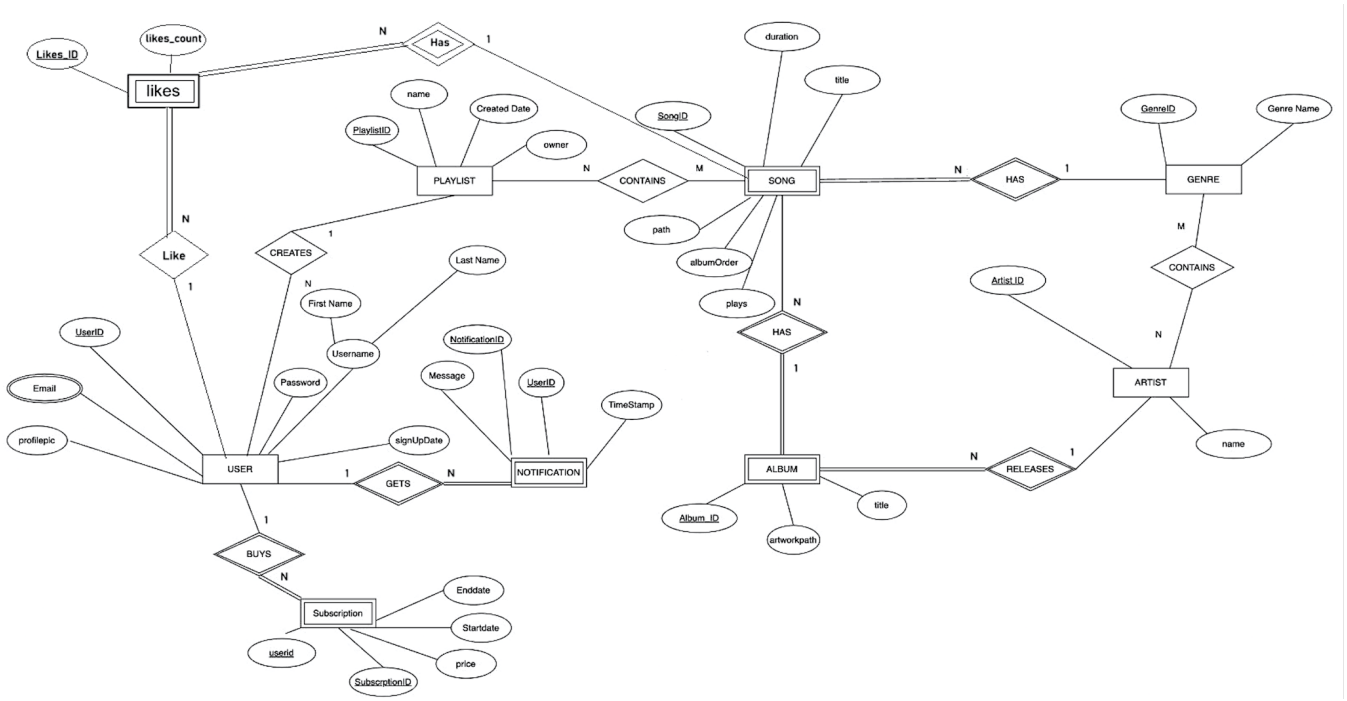
The MusicXperience project is a comprehensive music management system that integrates a robust relational database to efficiently organize and store diverse music-related data, including albums, artists, genres, songs, playlists, and user information. Users can create accounts with unique usernames, providing personal details such as first and last names, email addresses, and profile pictures. The system incorporates user authentication and password protection to ensure secure access.

The music library is enriched with a vast collection of songs, each containing essential details like title, artist, album, genre, duration, path, album order, and play count. Users can explore and enjoy music across various genres and artists. The project facilitates playlist management, allowing users to create and customize playlists with the ability to add songs and define playlist orders. The likes tracking feature records user preferences, with a trigger automatically updating the like count in the songs table after each insertion.

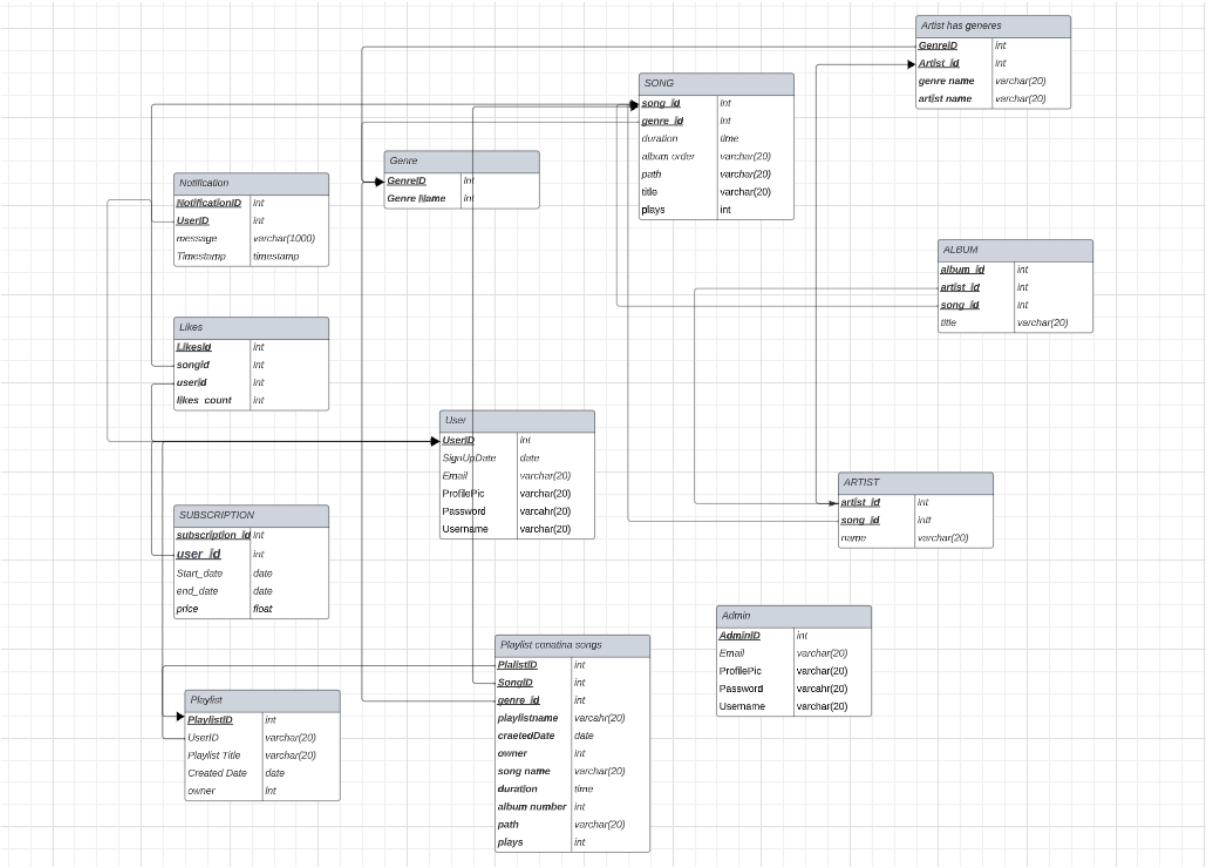
Notifications play a crucial role in keeping users informed about significant events, such as the addition of new artists, songs, or albums, as well as changes in user subscriptions. The subscription management system enables users to subscribe to different plans, each associated with a specific price, start date, and end date. Changes in subscription plans trigger notifications to inform users about modifications.

Additionally, an event scheduler is implemented to automatically delete old notifications daily, ensuring a streamlined and up-to-date notification system. Overall, MusicXperience aims to provide users with a feature-rich platform for managing and enjoying their music preferences, while keeping them informed about relevant activities and updates within the music library.

II. E R Diagram



III. Relational Schema



IV. DDL SQL Commands

```
CREATE TABLE `albums` (
`id` int(11) NOT NULL,
`title` varchar(250) NOT NULL,
`artist` int(11) NOT NULL,
`genre` int(11) NOT NULL,
`artworkPath` varchar(500) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `artists` (
`id` int  NOT NULL,
`name` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `genres` (
`id` int(11) NOT NULL,
`name` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `playlists` (
`id` int(11) NOT NULL,
`name` varchar(50) NOT NULL,
`owner` int(11) NOT NULL,
`dateCreated` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `playlistssongs` (
`id` int(11) NOT NULL,
`songId` int(11) NOT NULL,
`playlistId` int(11) NOT NULL,
`playlistOrder` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
CREATE TABLE `songs` (
`id` int(11) NOT NULL,
`title` varchar(250) NOT NULL,
`artist` int(11) NOT NULL,
`album` int(11) NOT NULL,
`genre` int(11) NOT NULL,
`duration` varchar(8) NOT NULL,
`path` varchar(500) NOT NULL,
`albumOrder` int(11) NOT NULL,
`plays` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
CREATE TABLE `users` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(25) NOT NULL,
`firstName` varchar(50) NOT NULL,
`lastName` varchar(50) NOT NULL,
`email` varchar(200) NOT NULL,
`password` varchar(32) NOT NULL,
`signUpDate` datetime NOT NULL,
`profilePic` varchar(500) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `likes` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`user_id` int(11) NOT NULL,
`song_id` int(11) NOT NULL,
`like_count` int(11) NOT NULL DEFAULT 0,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `admin` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(25) NOT NULL,
```

```
`firstName` varchar(50) NOT NULL,
`lastName` varchar(50) NOT NULL,
`email` varchar(200) NOT NULL,
`password` varchar(32) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `notifications` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`user_id` int(11) NOT NULL,
`message` varchar(255) NOT NULL,
`timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (`id`),
KEY `user_id` (`user_id`),
CONSTRAINT `fk_user_id` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;



CREATE TABLE `subscriptions` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`userid` int(11) NOT NULL,
`planname` varchar(255) NOT NULL,
`price` decimal(10, 2) NOT NULL,
`startdate` date NOT NULL,
`enddate` date NOT NULL,
PRIMARY KEY (`id`),
FOREIGN KEY (`userid`) REFERENCES `users`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `advertisements` (
`ad_id` int(11) NOT NULL AUTO_INCREMENT,
`userid` int(11) NOT NULL,
`ad_name` varchar(255) NOT NULL,
```

```
`company_name` varchar(255) NOT NULL,
`ad_description` text NOT NULL,
`ad_url` varchar(255) NOT NULL,
`ad_clicks` int(11) DEFAULT 0,
PRIMARY KEY (`ad_id`),
FOREIGN KEY (`userid`) REFERENCES `users`(`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
ALTER TABLE `albums`
ADD PRIMARY KEY (`id`);

ALTER TABLE `artists`
ADD PRIMARY KEY (`id`);

ALTER TABLE `genres`
ADD PRIMARY KEY (`id`);

ALTER TABLE `playlists`
ADD PRIMARY KEY (`id`);

ALTER TABLE `playlistssongs`
ADD PRIMARY KEY (`id`);

ALTER TABLE `songs`
ADD PRIMARY KEY (`id`);

ALTER TABLE `albums`
MODIFY `id` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;

ALTER TABLE `genres`
MODIFY `id` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;
```

```
ALTER TABLE `playlists`
MODIFY `id` int NOT NULL AUTO_INCREMENT;

ALTER TABLE `playlistssongs`
MODIFY `id` int NOT NULL AUTO_INCREMENT;

ALTER TABLE `songs`
MODIFY `id` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=32;

ALTER TABLE songs ADD COLUMN likes INT DEFAULT 0;

ALTER TABLE `users`
MODIFY `id` int  NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;COMMIT;
ALTER TABLE playlists
ADD FOREIGN KEY (owner) REFERENCES users(id);

ALTER TABLE `subscriptions`
ADD FOREIGN KEY (`userid`) REFERENCES `users`(`id`);
```

```
--New Creation Notification
DELIMITER //

CREATE TRIGGER `after_insert_user` AFTER INSERT ON `users`
FOR EACH ROW
BEGIN
 DECLARE target_user_id INT;
 SET target_user_id = NEW.id;  -- Assuming you want to track notifications for the newly
inserted user

 IF NEW.id = target_user_id THEN
```

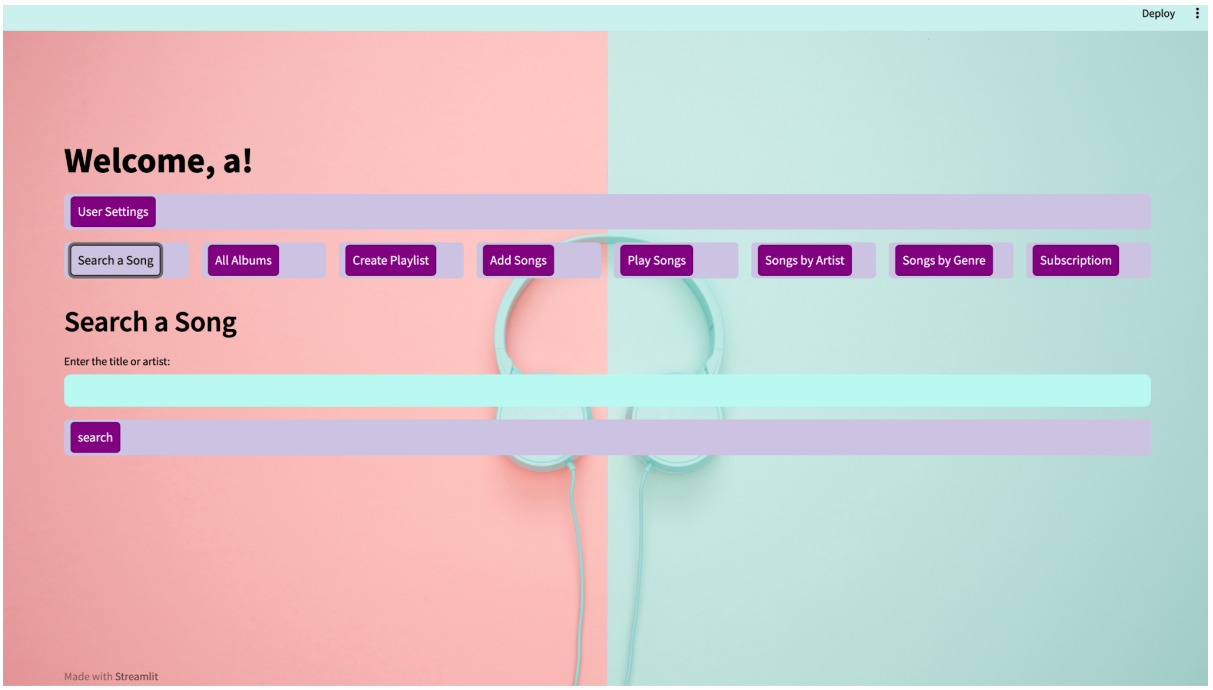
```
  INSERT INTO `notifications` (`user_id`, `message`) VALUES (target_user_id, 'Welcome!
To MusicXperience.');
 END IF;
END //

DELIMITER ;

--Artist Notification
DELIMITER //
CREATE TRIGGER after_insert_artist
AFTER INSERT ON artists
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New artist added: ', NEW.name), NOW()
  FROM users;
END;
//
DELIMITER ;

--Song Notification
DELIMITER //
CREATE TRIGGER after_insert_song
AFTER INSERT ON songs
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New song added: ', NEW.title), NOW()
  FROM users;
END;
//
DELIMITER ;
```

```
--Album notification
DELIMITER //
CREATE TRIGGER after_insert_album
AFTER INSERT ON albums
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New album created: ', NEW.title), NOW()
  FROM users;
END;
//
DELIMITER ;

--Subscription notification
DELIMITER //
CREATE TRIGGER after_insert_subscription
AFTER UPDATE ON subscriptions
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('Subscription changed to: ', NEW.planname), NOW()
  FROM users;
END;
//
DELIMITER ;

--Create a trigger that fires after an insert on the likes table
DELIMITER //
CREATE TRIGGER after_like_insert
AFTER INSERT ON likes
FOR EACH ROW
```

```
BEGIN
  -- Increment the like count in the song table
  UPDATE songs
  SET likes = likes + 1
  WHERE id = NEW.song_id;
END;
//
DELIMITER ;



--Create an event to delete old notifications every day
CREATE EVENT delete_old_notifications
 ON SCHEDULE
  EVERY 1 DAY
  STARTS TIMESTAMP(CURRENT_DATE, '00:00:00')
 DO
  DELETE FROM notifications
  WHERE timestamp < NOW() - INTERVAL 1 DAY;
```

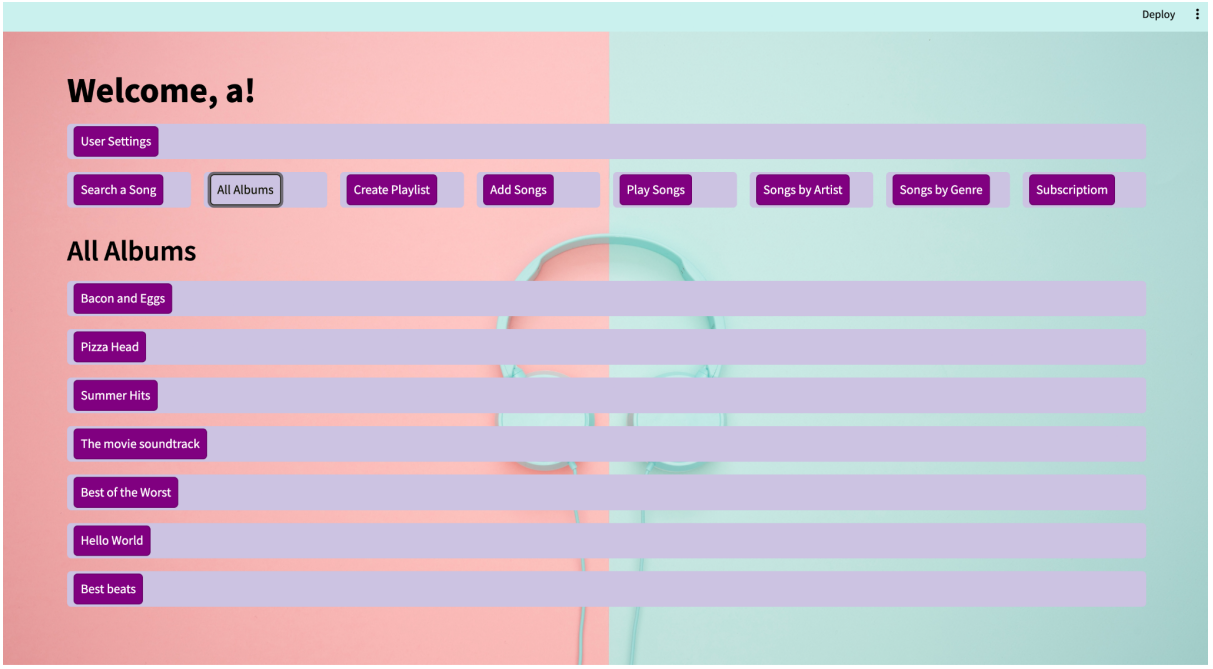
V. CRUD OPERATIONS SCREENSHOTS :

```
def search_song(query):
  cursor = conn.cursor(buffered=True)
  # search_query = f"SELECT * FROM songs WHERE title LIKE '%{query}%' OR artist LIKE '%{query}%';"
  search_query = f"""
     SELECT songs.*, albums.artworkPath as albumArtworkPath
     FROM songs
     JOIN albums ON songs.album = albums.id
     WHERE songs.title LIKE '%{query}%' OR songs.artist LIKE '%{query}%';
  """
  cursor.execute(search_query)
  results = cursor.fetchall()
  cursor.close()
  conn.commit()

  return results
```



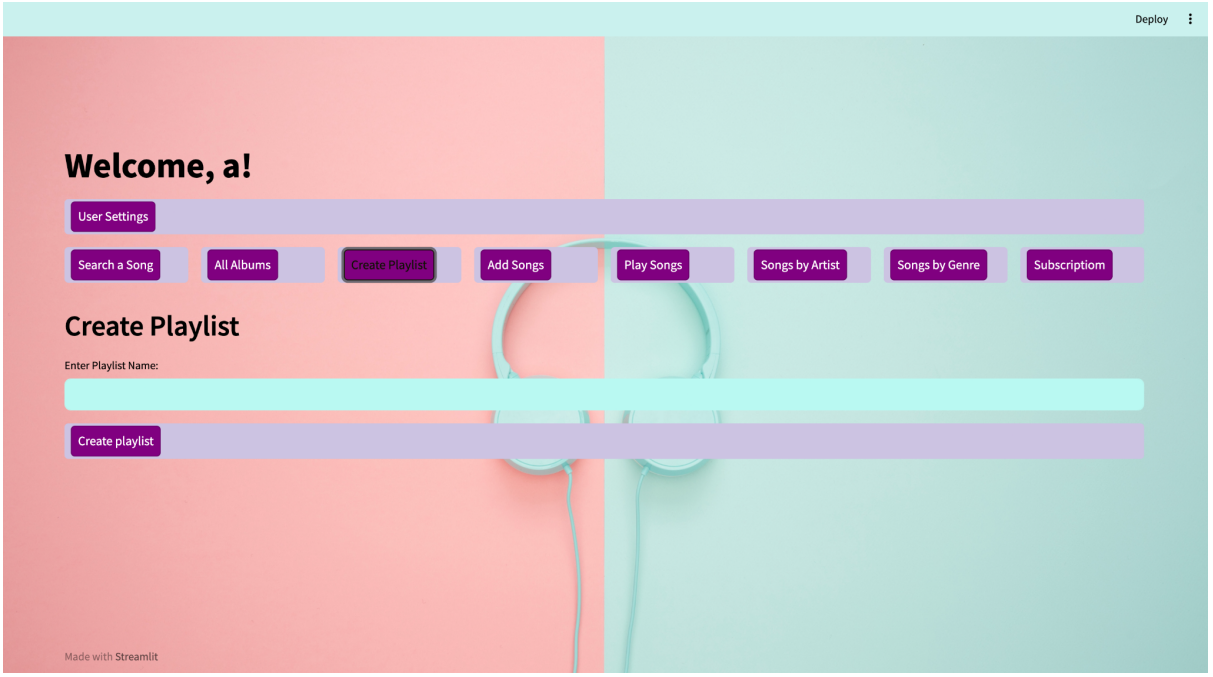

```
def display_all_songs_in_album(album_id,user_id):
```

```
cursor = conn.cursor(buffered=True)
query = "SELECT * FROM songs WHERE album = %s"
cursor.execute(query, (album_id,))
songs = cursor.fetchall()
query = "SELECT * FROM albums WHERE id = %s"
cursor.execute(query, (album_id,))
album_info = cursor.fetchone()
cursor.close()
conn.commit()
st.subheader("Songs in the Album:")
if songs:
    render_songs_in_album(songs, user_id)
else:
    st.info("No songs found for this album.")
```




```
def create_playlist(name, owner):
    date_created = datetime.now()
    cursor = conn.cursor(buffered=True)
```

```
  if is_procedure_exists(cursor, 'InsertPlaylist'):
      cursor.callproc('InsertPlaylist', (name, owner, date_created))
      cursor.fetchall()
      cursor.close()
      conn.commit()
  else:
      cursor.execute('''CREATE PROCEDURE InsertPlaylist(IN playlistName
VARCHAR(255), IN playlistOwner INT, IN playlistDateCreated DATE)
                BEGIN
                    INSERT INTO playlists (name, owner, dateCreated) VALUES (playlistName,
playlistOwner, playlistDateCreated);
                END;''')
      cursor.callproc('InsertPlaylist', (name, owner, date_created))
      cursor.fetchall()
      cursor.close()
      conn.commit()
  st.success(f"Playlist '{name}' created successfully!")
```

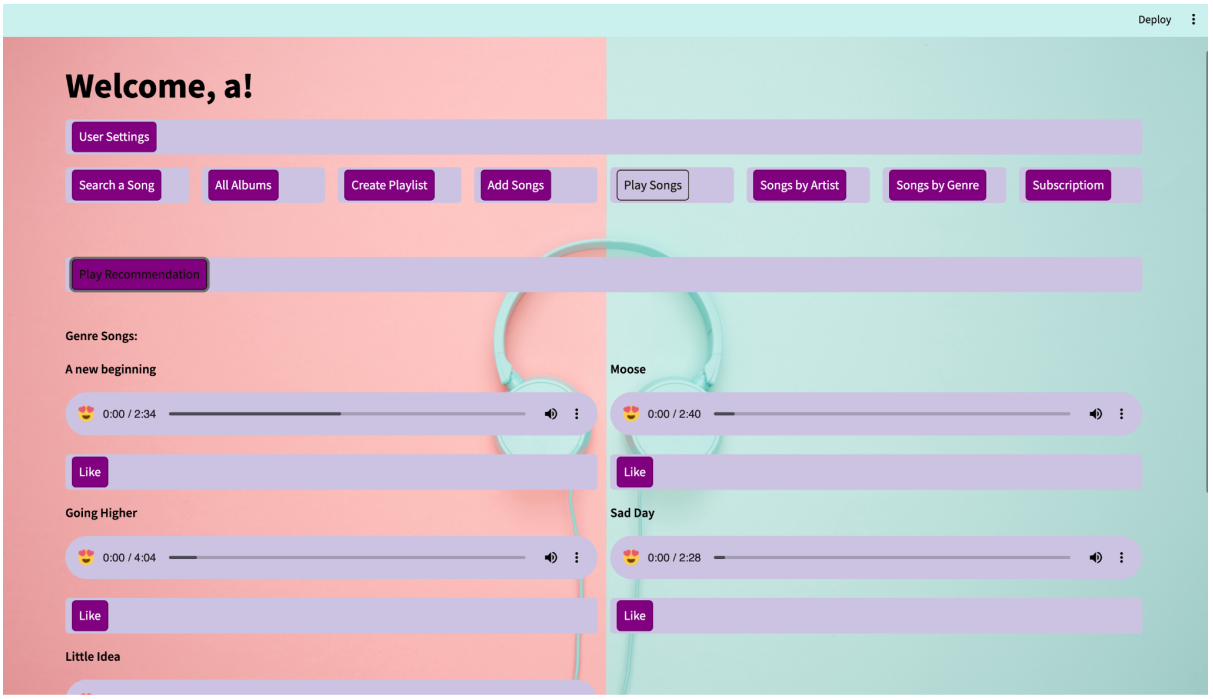



```
def add_songs_to_playlist(playlist_id, song_titles):
  cursor = conn.cursor(buffered=True)
```

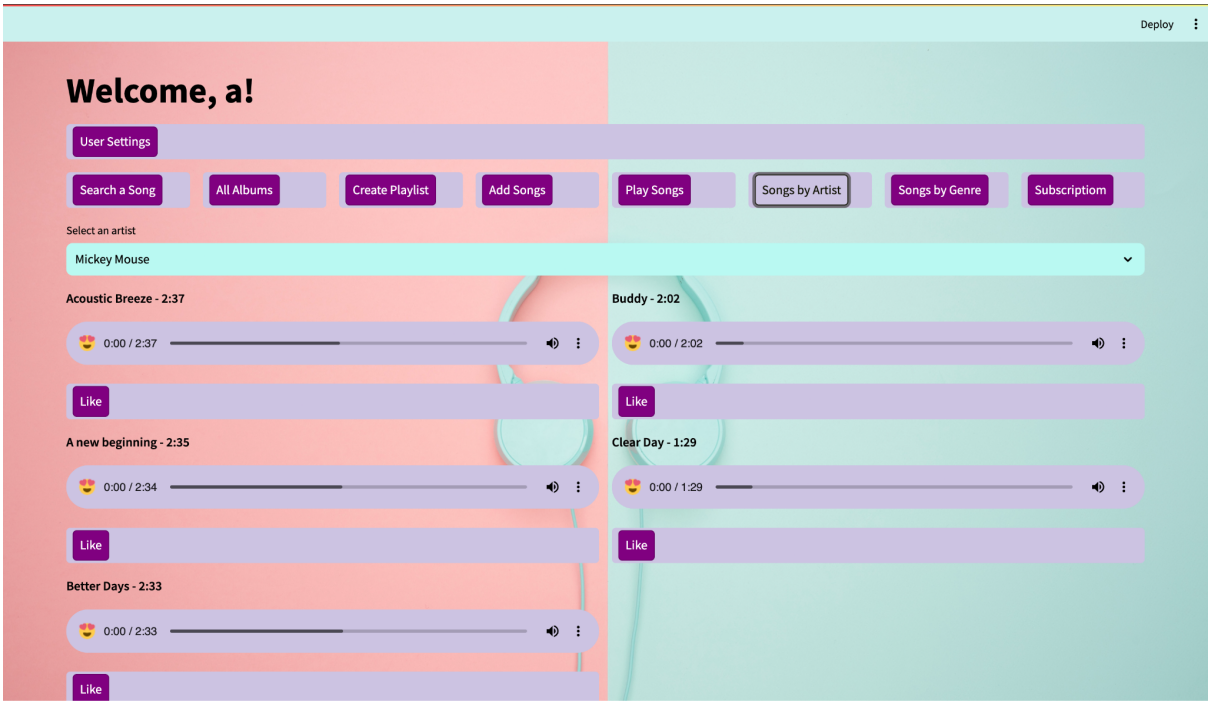
```
for song_title in song_titles:
    # Look up the song_id based on the song_title
    cursor.execute("SELECT id FROM songs WHERE title = %s", (song_title,))
    result = cursor.fetchone()

    if result:
        song_id = result[0]
        cursor.execute(
            "INSERT INTO playlistssongs (songId, playlistId, playlistOrder) VALUES (%s, %s, %s)", (song_id, playlist_id, 0))
    else:
        st.warning(f"Song not found: {song_title}")

cursor.close()
conn.commit()
st.success("Songs added to the playlist successfully!")
```


```
def play_all_songs(playlist_id, user_id, shuffle=False):
    cursor = conn.cursor(buffered=True)
```

```
cursor.execute("""
    SELECT songs.id,songs.title, songs.path, genres.name as genre_name
    FROM songs
    JOIN playlistssongs ON songs.id = playlistssongs.songId
    JOIN genres ON songs.genre = genres.id
    WHERE playlistssongs.playlistId = %s
""", (playlist_id,))
songs = cursor.fetchall()

if shuffle:
    random.shuffle(songs)  # Shuffle the songs if shuffle is True

render_playlist_songs(songs, user_id)
cursor.close()
conn.commit()
```

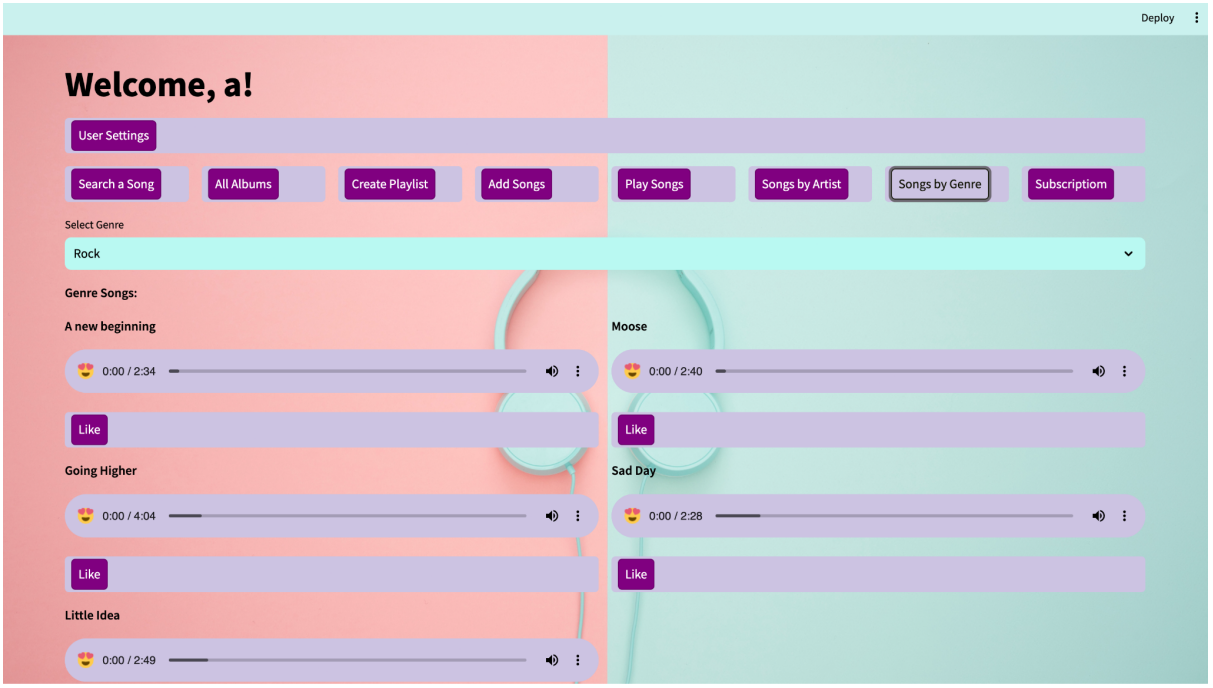


```
def get_songs_by_artist(artist_name):
  cursor = conn.cursor(buffered=True)
  query = "SELECT * FROM songs WHERE artist = (SELECT id FROM artists WHERE
name = %s)"
  cursor.execute(query, (artist_name,))
  songs = cursor.fetchall()
  cursor.close()
  conn.commit()
  return songs
```



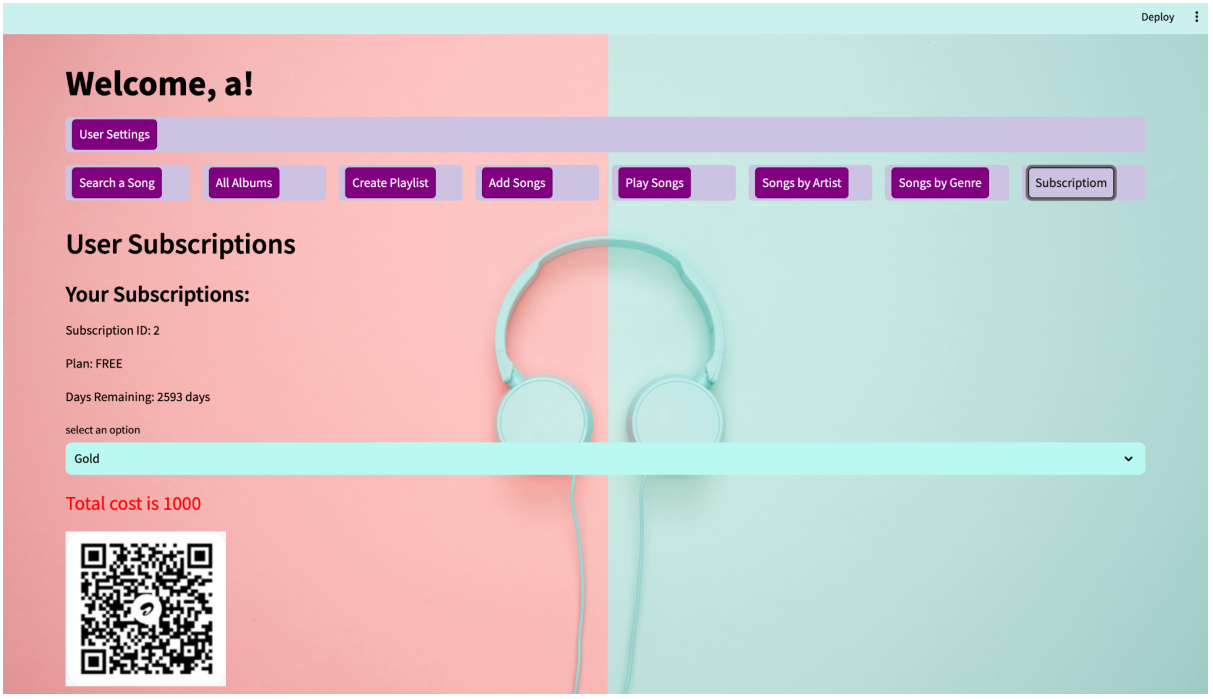

```
def display_songs_by_genre(user_id):
```

```
cursor = conn.cursor(buffered=True)
cursor.execute("SELECT * FROM genres")
genres = cursor.fetchall()
genre_names = [genre[1] for genre in genres]
selected_genre = st.selectbox("Select Genre", genre_names)

genre_id = [genre[0] for genre in genres if genre[1] == selected_genre][0]
cursor = conn.cursor(buffered=True)
cursor.execute("""
    SELECT songs.id,songs.title, songs.path
    FROM songs
    WHERE songs.genre = %s
""", (genre_id,))
songs = cursor.fetchall()

st.write(f"**Genre Songs:**")

if songs:
    render_songs_by_genre(songs, user_id)
else:
    st.info("No songs found for this genre.")
cursor.close()
conn.commit()
```




```
def play_all_songs_page(user_id):
  user_playlists = get_user_playlists(user_id)
  like_playlist = get_like_playlists(user_id)
  genre_recommendation = get_recommendation(user_id)
  if "button8" not in st.session_state:
      st.session_state.button8 = False

  if "button12" not in st.session_state:
      st.session_state.button12 = False
  if like_playlist:
      st.markdown("<br>", unsafe_allow_html=True)
      st.markdown("<br>", unsafe_allow_html=True)
      if st.button("Play Liked Songs Playlist", key=f"{user_id}_like"):
          st.markdown("<br>", unsafe_allow_html=True)
          st.session_state.button8 = not st.session_state.button8
  if st.session_state.button8:
      play_liked_songs(like_playlist)
      st.markdown("<br>", unsafe_allow_html=True)
```

```
if genre_recommendation:
    st.markdown("<br>", unsafe_allow_html=True)
    st.markdown("<br>", unsafe_allow_html=True)
    random_key = random.randint(1, 1000000)
    if st.button("Play Recommendation", key=f"{user_id}_recommendation"):
        st.markdown("<br>", unsafe_allow_html=True)
        st.session_state.button12 = not st.session_state.button12
if st.session_state.button12:
    display_recommmendation_genre(genre_recommendation['genre'], user_id)
    st.markdown("<br>", unsafe_allow_html=True)

if not user_playlists or not like_playlist:
    st.info("You haven't created any playlists yet. Create one to get started!")
    return
st.markdown("<br>", unsafe_allow_html=True)
st.markdown("<br>", unsafe_allow_html=True)
playlist_name = st.selectbox("Select Playlist", user_playlists)
playlist_id = get_playlist_id_by_name(playlist_name)
shuffle_button = st.button("Shuffle Songs")
if (not (st.session_state.button12) and not (st.session_state.button8)):
    if shuffle_button:
        if st.session_state.plan_name != "FREE":
            play_all_songs(playlist_id, user_id, shuffle=True)
        else:
            styled_text = f"<p style='font-size: 24px; color: red;'>You are a FREE user. Buy
GOLD and SILVER subscriptions to avail premium features</p>"
            st.markdown(styled_text, unsafe_allow_html=True)
            play_all_songs(playlist_id, user_id)
    else:
        play_all_songs(playlist_id, user_id)
```




```
def add_song(song_id, title, artist, album, genre, duration, path, album_order, plays):
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO songs (id, title, artist, album, genre, duration, path, albumOrder, plays)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (song_id, title, artist, album, genre, duration, path, album_order, plays))
    conn.commit()
    st.success("Song added successfully!")
```


```python
def delete_song(song_id):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM songs WHERE id = %s", (song_id,))
    conn.commit()
    st.success("Song deleted successfully!")
```

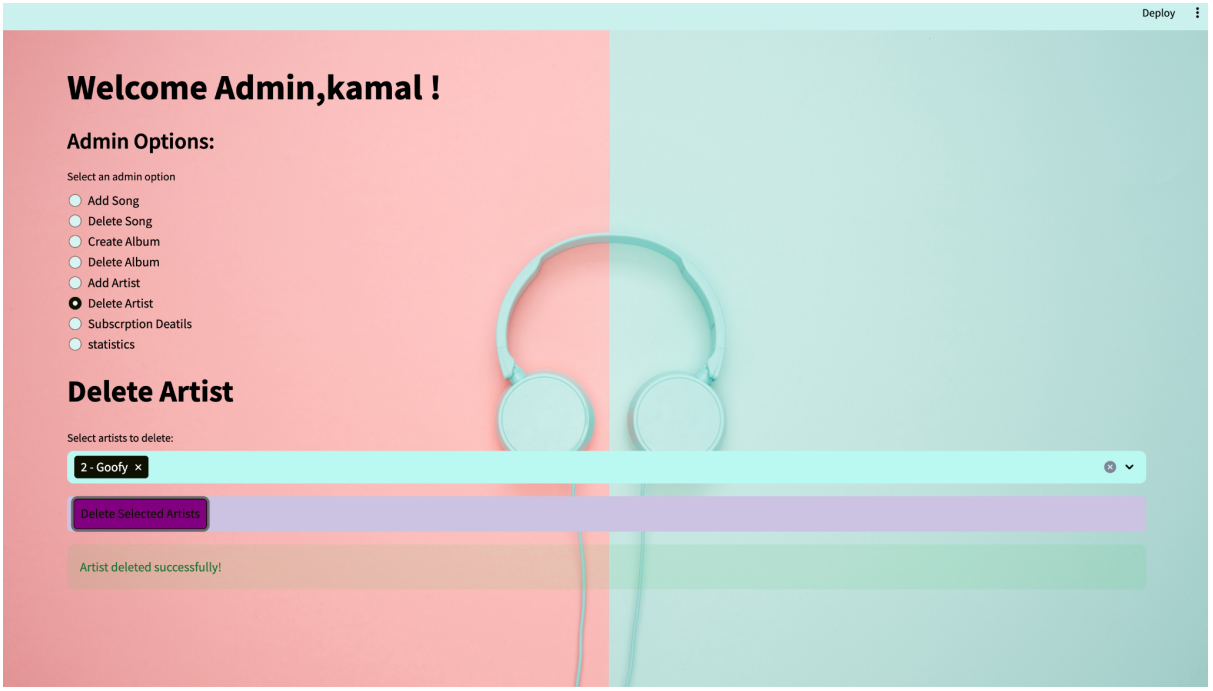
```
def create_album(album_id, title, artist, genre, artwork_path):

  cursor = conn.cursor()
  cursor.execute("""
     INSERT INTO albums (id, title, artist, genre, artworkPath)
     VALUES (%s, %s, %s, %s, %s)
  """, (album_id, title, artist, genre, artwork_path))
  conn.commit()
  st.success("Album created successfully!")
```


```
def add_artist(artist_id, name):
  cursor = conn.cursor()

  # Check if the artist with the given ID already exists
  cursor.execute("SELECT * FROM artists WHERE id = %s", (artist_id,))
  existing_artist = cursor.fetchone()

  if existing_artist:
     # Artist already exists, update the name
     cursor.execute("UPDATE artists SET name = %s WHERE id = %s", (name, artist_id))
     st.success("Artist updated successfully!")
  else:
```

```
    # Artist doesn't exist, insert a new record
    cursor.execute("INSERT INTO artists (id, name) VALUES (%s, %s)", (artist_id, name))
    st.success("Artist added successfully!")

conn.commit()
```

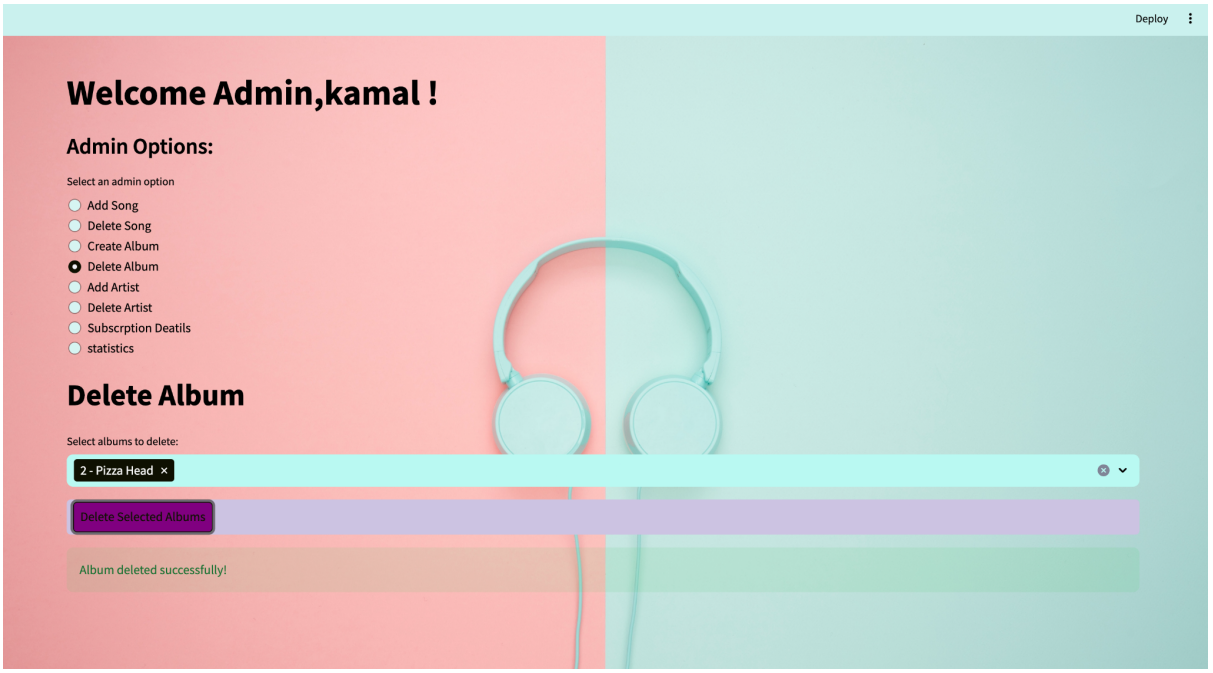

```
def delete_artist(artist_id):
  cursor = conn.cursor(buffered=True)
  if is_procedure_exists(cursor, 'DeleteArtists'):
    # If it exists, call the procedure
    cursor.callproc('DeleteArtist', (artist_id,))
      # Commit the changes
    cursor.fetchall()
    cursor.close()
    conn.commit()
  else:
    cursor.execute('''CREATE PROCEDURE DeleteArtist(IN artistId INT)
                BEGIN
                  DELETE FROM artists WHERE id = artistId;
                END;''')
```

```
    cursor.callproc('DeleteArtist', (artist_id,))
        # Commit the changes
    cursor.fetchall()
    cursor.close()
    conn.commit()
st.success("Artist deleted successfully!")
```



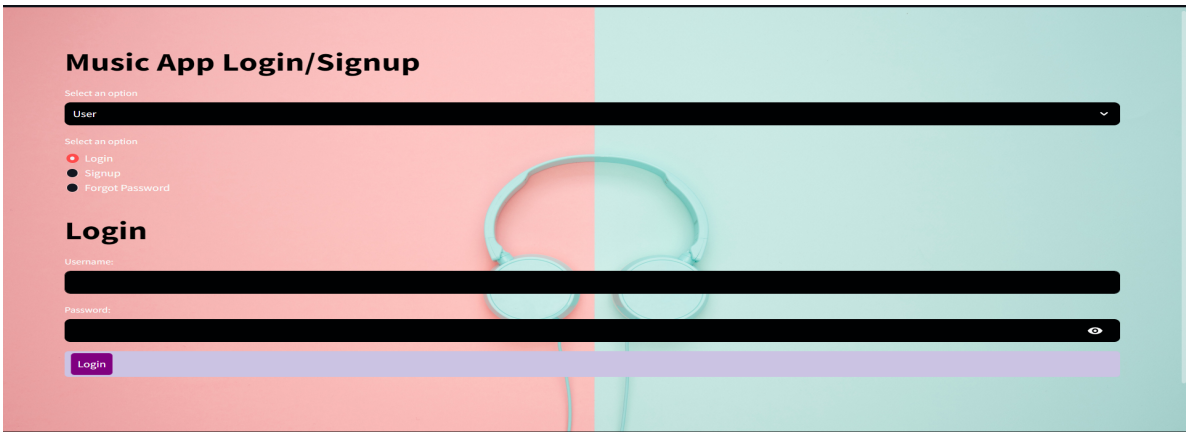

```
def delete_album(album_id):
  cursor = conn.cursor(buffered=True)
  if is_procedure_exists(cursor, 'DeleteAlbum'):
    # If it exists, call the procedure
    cursor.callproc('DeleteAlbum', (album_id,))
        # Commit the changes
    cursor.fetchall()
    cursor.close()
    conn.commit()
  else:
    cursor.execute("'CREATE PROCEDURE DeleteAlbum(IN album_id INT)
                BEGIN
                    DELETE FROM albums WHERE id= album_id;
```

```
                END;'")
    cursor.callproc('DeleteAlbum', (album_id,))
        # Commit the changes
    cursor.fetchall()
    cursor.close()
    conn.commit()
st.success("Album deleted successfully!")
```



VI. List of Functionalities and its associated Query screenshots from front end

1.User Login:



FRONT END:

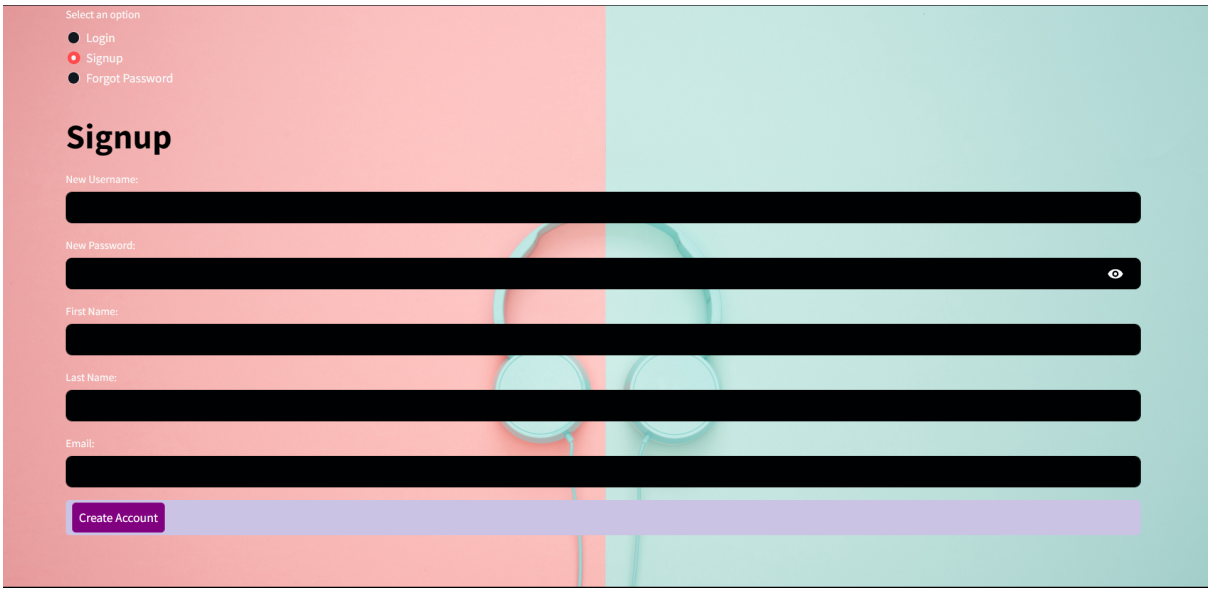
```
if st.session_state.user_id is None:
        st.markdown("# Music Xperience Login/Signup")
        useadmin = st.selectbox("Select an option", ["User", "Admin"])
        if useadmin == "User":
            page = st.radio("Select an option", [
                        "Login", "Signup", "Forgot Password"])
            if page == "Login":
                login_page()
            if page == "Signup":
                signup_page()
            if page == "Forgot Password":
                forgot_password()
        else:
            admin_login_page()
```

BACKEND:

```
def login_page():
    st.title("Login")
    username = st.text_input("Username:")
    password = st.text_input("Password:", type="password")
    if st.button("Login"):

        user = verify_user(username, password)
        if user:
            st.success(f"Welcome, {username}!")
            st.session_state.user_id = user[0]
            st.session_state.username = username
            user_id = user[0]
            cursor = conn.cursor(buffered=True)
            if not user_has_subscription(st.session_state.user_id, cursor):
                today=date.today()
                add_subscription(st.session_state.user_id, "FREE", 0.00, today, "2030-12-31",
cursor, conn)
            user_plan_name = get_user_plan_name(st.session_state.user_id, cursor)
```

```
        st.session_state.plan_name = user_plan_name
        st.experimental_rerun()
    else:
        st.error("Invalid username or password")
```

2.User Sign Up:




```
def signup_page():
    st.title("Signup")
    new_username = st.text_input("New Username:")
```

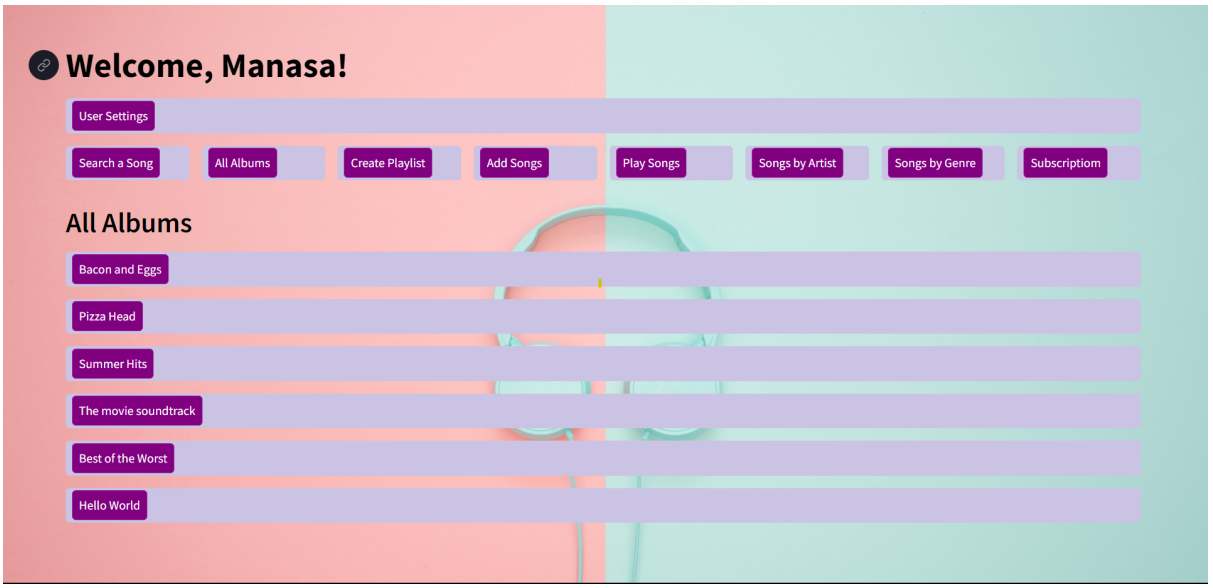
```
    new_password = st.text_input("New Password:", type="password")
    first_name = st.text_input("First Name:")
    last_name = st.text_input("Last Name:")
    email = st.text_input("Email:")
    if st.button("Create Account"):
        create_user(new_username, new_password, first_name, last_name, email)
        # login_page()

    # Increment ad clicks
    cursor = conn.cursor(buffered=True)
    cursor.execute(f"UPDATE Advertisements SET ad_clicks = ad_clicks + 1 WHERE ad_id
= {ad['ad_id']}")
    conn.commit()
```

3.Forgot Password:

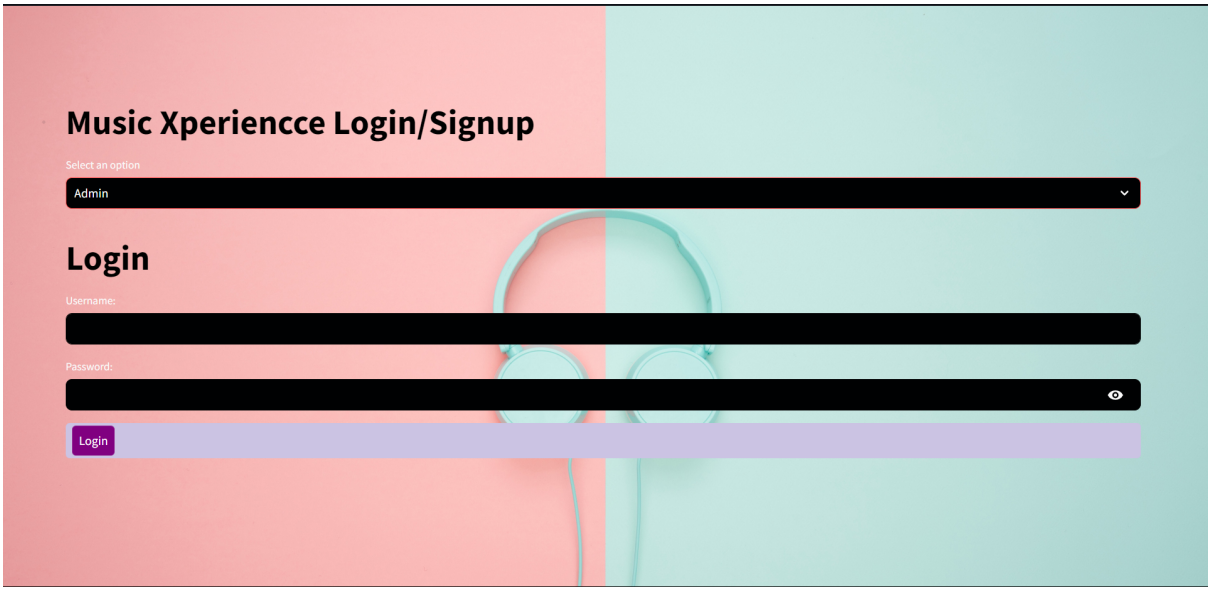

```
def forgot_password():
```

```
st.title("Forgot Password")
username = st.text_input("Username:")
email = st.text_input("Email")
new_password = st.text_input("Enter new password:", type="password")
confirm_password = st.text_input("Confirm new password:", type="password")
submit=st.button('Change Password')

if submit:
    user_data = check_user_credentials(username, email)
    if user_data:
        if new_password == confirm_password:
            hashed_password = hashlib.md5(new_password.encode()).hexdigest()
            update_password_in_database(user_data[0], hashed_password)
            st.success("Password updated successfully!")
        else:
            st.error("Passwords do not match. Please try again.")
    else:
        st.error("Invalid username or email. Please check your credentials.")
```

4. Client Page:



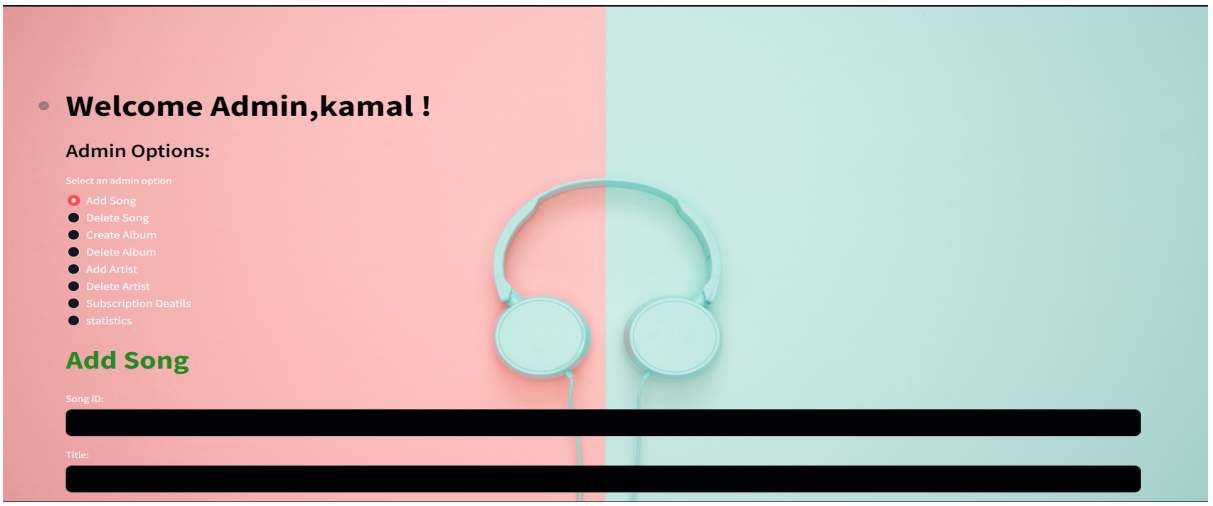
5. Admin Login:



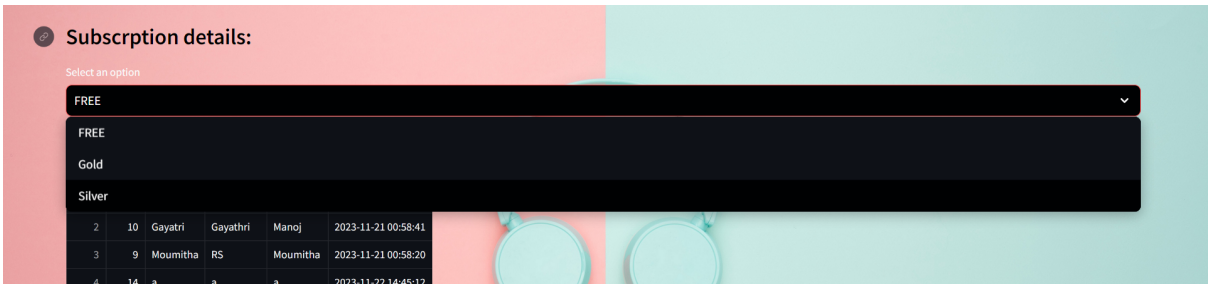

```
def admin_login_page():
    st.title("Login")
    username = st.text_input("Username:")
    password = st.text_input("Password:", type="password")
    if st.button("Login"):

        user = verify_admin(username, password)
        if user:
            st.success(f"Welcome, {username}!")
            st.session_state.user_id = user[0]
            st.session_state.username = username
            user_id = user[0]
            st.experimental_rerun()
        else:
```

```
        st.error("Invalid username or password")
```

6.Admin Page:



7. Subscription Deatils




```
def free_sub():
    cursor = conn.cursor(buffered=True)

    # Execute the SQL query
    query = """
        SELECT id, username, firstName, lastName, signUpDate
        FROM users
        WHERE id IN (
            SELECT userid
            FROM subscriptions
            WHERE planname = 'FREE'
        ) AND id <> 1;
    """
    cursor.execute(query)
```

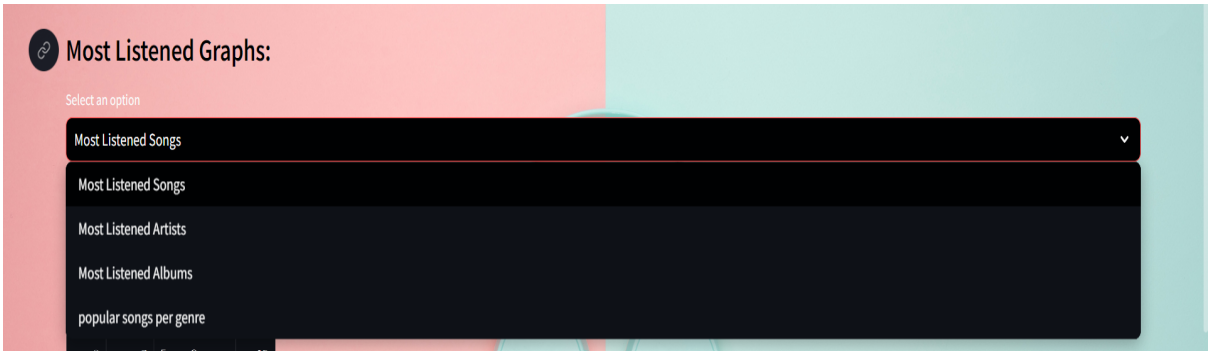
```
    results = cursor.fetchall()

        # Create a DataFrame
    columns = ["ID", "Username", "First Name", "Last Name", "Sign-Up Date"]
    df = pd.DataFrame(results, columns=columns)

        # Display results using Streamlit data table
    st.dataframe(df)
    cursor.execute('''SELECT COUNT(username) AS user_count
                FROM users
                WHERE id IN (
                    SELECT userid
                    FROM subscriptions
                    WHERE planname = 'FREE'
                )AND id <> 1;''')
        # Fetch the results'
    result = cursor.fetchone()



    # Display the result using Streamlit
    st.write(f"Number of users with FREE subscription: {result[0]}")
    cursor.close()
    conn.commit()



def silver_sub():
    cursor = conn.cursor(buffered=True)

        # Execute the SQL query
    query = """
            SELECT  id, username, firstName, lastName, signUpDate
            FROM users
            WHERE id IN (
                SELECT userid
                FROM subscriptions
                WHERE planname = 'Silver'
```

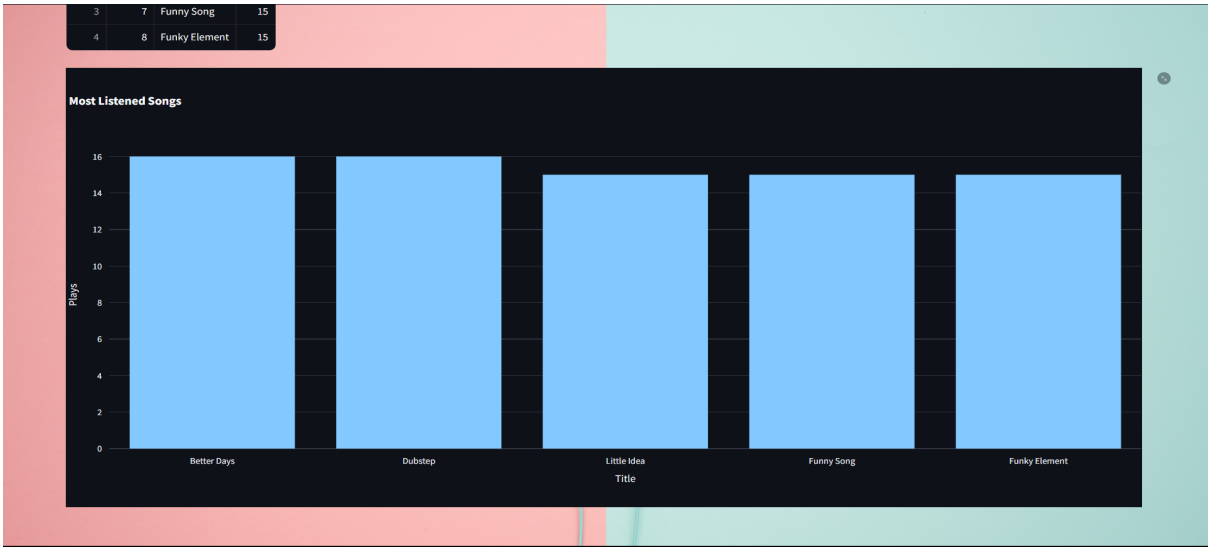
```
        );
    """
    cursor.execute(query)

    # Fetch the results
    results = cursor.fetchall()

    # Create a DataFrame
    columns = ["ID", "Username", "First Name", "Last Name", "Sign-Up Date"]
    df = pd.DataFrame(results, columns=columns)

    # Display results using Streamlit data table
    st.dataframe(df)
    cursor.execute('''SELECT COUNT(username) AS user_count
                FROM users
                WHERE id IN (
                    SELECT userid
                    FROM subscriptions
                    WHERE planname = 'Silver'
                );''')
    # Fetch the results'
    result = cursor.fetchone()


    # Display the result using Streamlit
    st.write(f"Number of users with Silver subscription: {result[0]}")
    cursor.close()
    conn.commit()

def gold_sub():
    cursor = conn.cursor(buffered=True)

    # Execute the SQL query
    query = """
            SELECT  id, username, firstName, lastName, signUpDate
            FROM users
            WHERE id IN (
```

```
            SELECT userid
            FROM subscriptions
            WHERE planname = 'Gold'
        );
    """
    cursor.execute(query)

    # Fetch the results
    results = cursor.fetchall()

    # Create a DataFrame
    columns = ["ID", "Username", "First Name", "Last Name", "Sign-Up Date"]
    df = pd.DataFrame(results, columns=columns)

    # Display results using Streamlit data table
    st.dataframe(df)
    cursor.execute('"SELECT COUNT(username) AS user_count
            FROM users
            WHERE id IN (
                SELECT userid
                FROM subscriptions
                WHERE planname = 'Gold'
            );"')
    # Fetch the results'
    result = cursor.fetchone()


    # Display the result using Streamlit
    st.write(f"Number of users with Gold subscription: {result[0]}")
    cursor.close()
    conn.commit()
def subscription():
    st.subheader("Subscrption details:")
    option = st.selectbox("Select an option", ["FREE", "Gold", "Silver"])
    if option == "FREE":
        free_sub()
    elif option == "Gold":
```

```
    gold_sub()
elif option == "Silver":
    silver_sub()
```

8.Most Listened Graphs

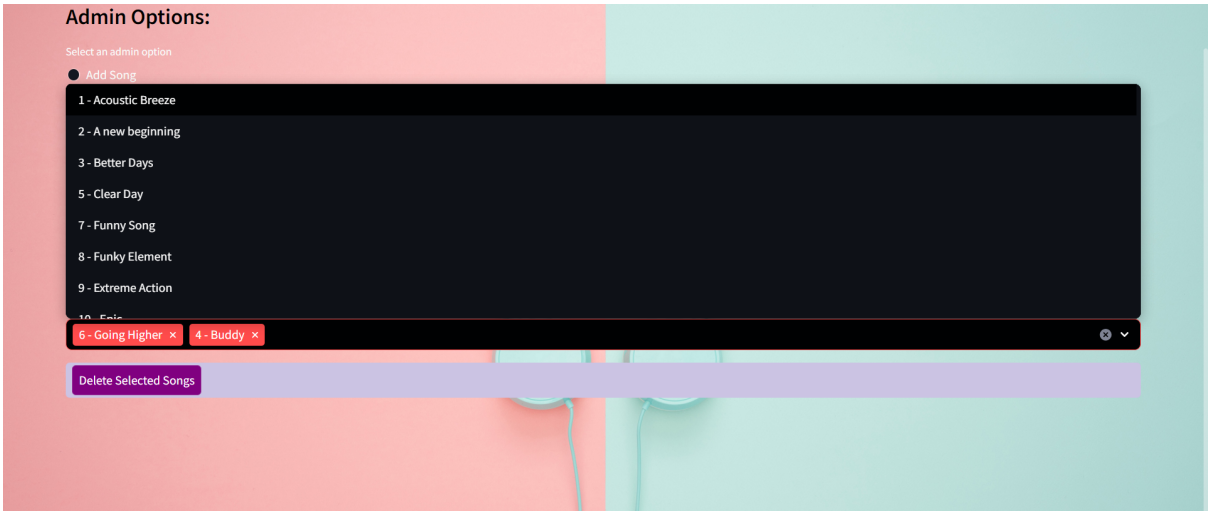





```
def get_most_listened_artists():
    cursor = conn.cursor()
    cursor.execute("""
        SELECT artists.name, SUM(songs.plays) as total_plays
        FROM artists
        JOIN songs ON artists.id = songs.artist
        GROUP BY artists.name
        ORDER BY total_plays DESC
        LIMIT 5
```

```
    """)
    artists_data = cursor.fetchall()
    cursor.close()

    df = pd.DataFrame(artists_data, columns=['Name', 'Total Plays'])
    return df

def display_most_listened_artists():
    # Get the most listened artists
    most_listened_artists_df = get_most_listened_artists()

    # Display the dataframe
    st.dataframe(most_listened_artists_df)

    fig = px.bar(most_listened_artists_df, x='Name', y='Total Plays', title='Most Listened Artists')
    fig.update_layout(height=600, width=1370)  # Adjust the height and width as needed
    st.plotly_chart(fig)
```

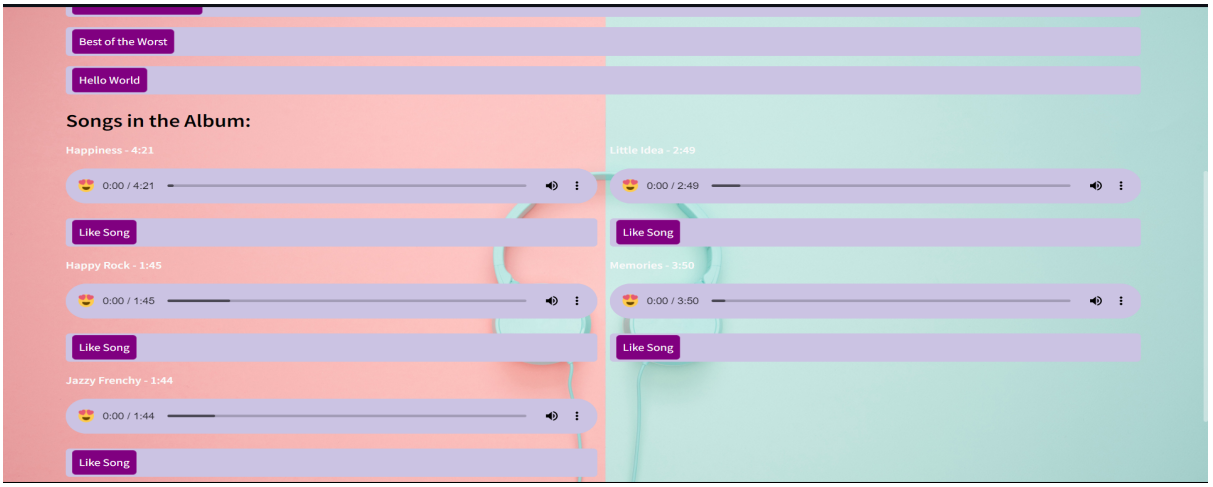
9.Delete Selected Song




```
def delete_song(song_id):
    cursor = conn.cursor()
    cursor.execute("DELETE FROM songs WHERE id = %s", (song_id,))
```

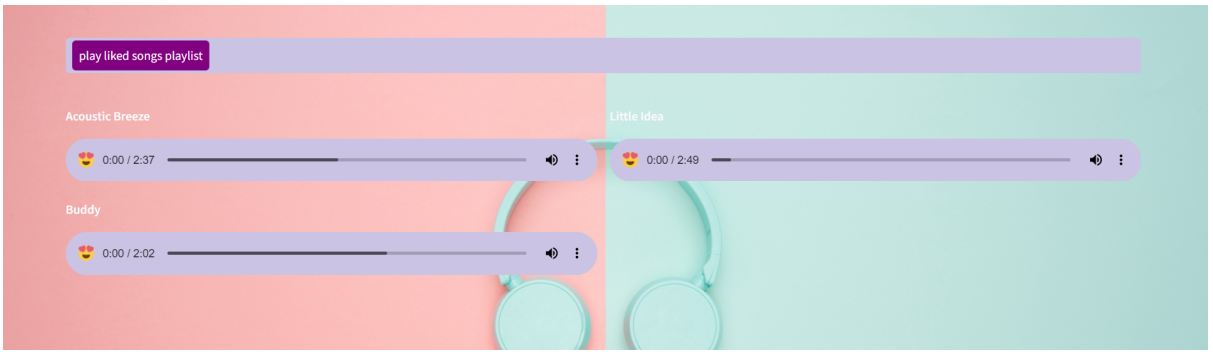
```
conn.commit()
st.success("Song deleted successfully!")
```

10.Display all songs in selected album

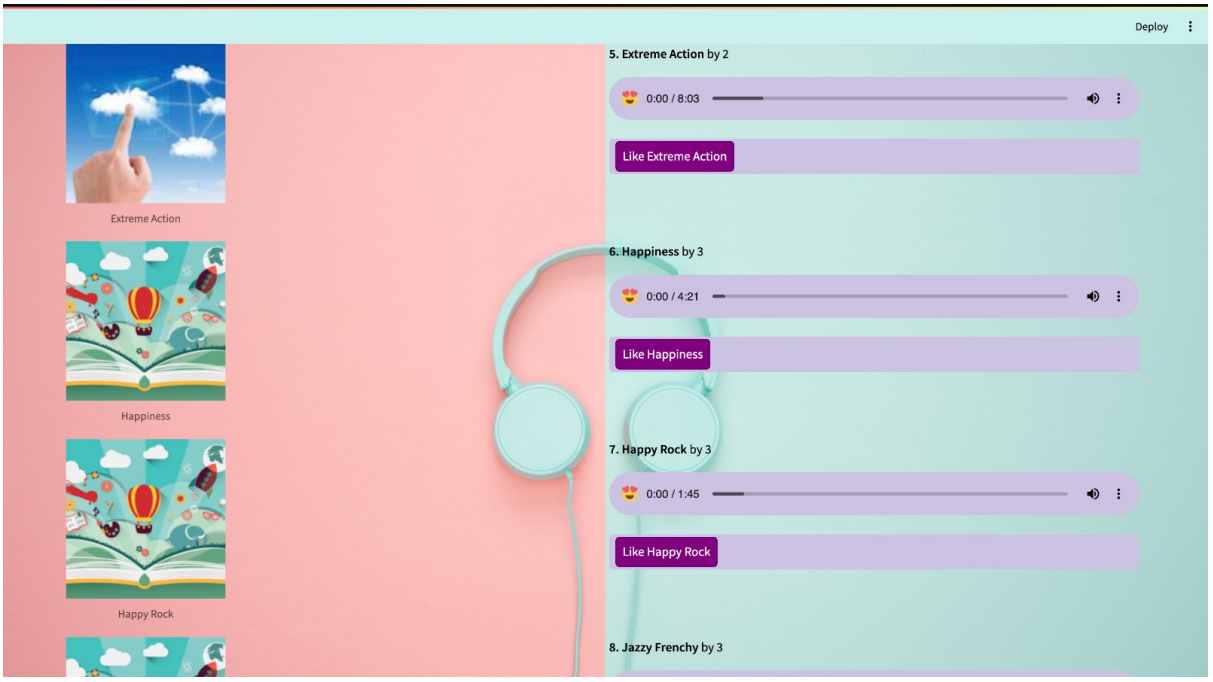


The like button is used to like the song and that song will be added to the users liked playlist and if the user has liked one song the play liked song will be enables.

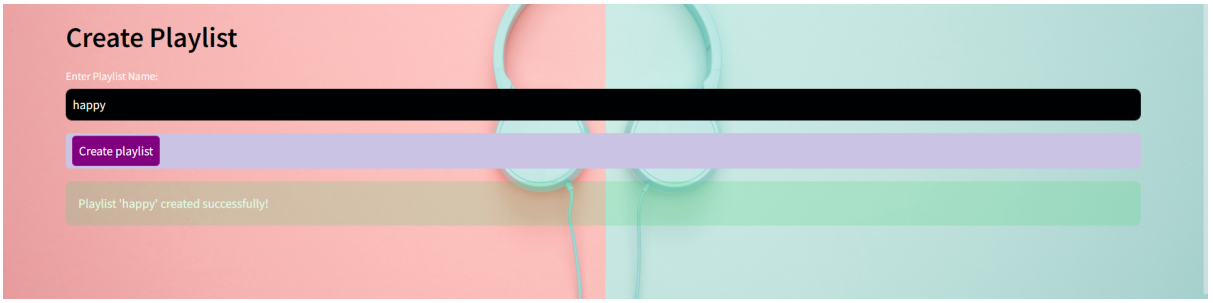
11.Play Liked Song Playlist



12. Search a song



13. Create Playlist




```
def create_playlist(name, owner):
    date_created = datetime.now()
    cursor = conn.cursor(buffered=True)

    if is_procedure_exists(cursor, 'InsertPlaylist'):
        # If it exists, call the procedure
        cursor.callproc('InsertPlaylist', (name, owner, date_created))
```

```
        # Commit the changes
        cursor.fetchall()
        cursor.close()
        conn.commit()
    else:
        cursor.execute('''CREATE PROCEDURE InsertPlaylist(IN playlistName
VARCHAR(255), IN playlistOwner INT, IN playlistDateCreated DATE)
                BEGIN
                    INSERT INTO playlists (name, owner, dateCreated) VALUES (playlistName,
playlistOwner, playlistDateCreated);
                END;''')
        cursor.callproc('InsertPlaylist', (name, owner, date_created))
        # Commit the changes
        cursor.fetchall()
        cursor.close()
        conn.commit()
    st.success(f"Playlist '{name}' created successfully!")
```

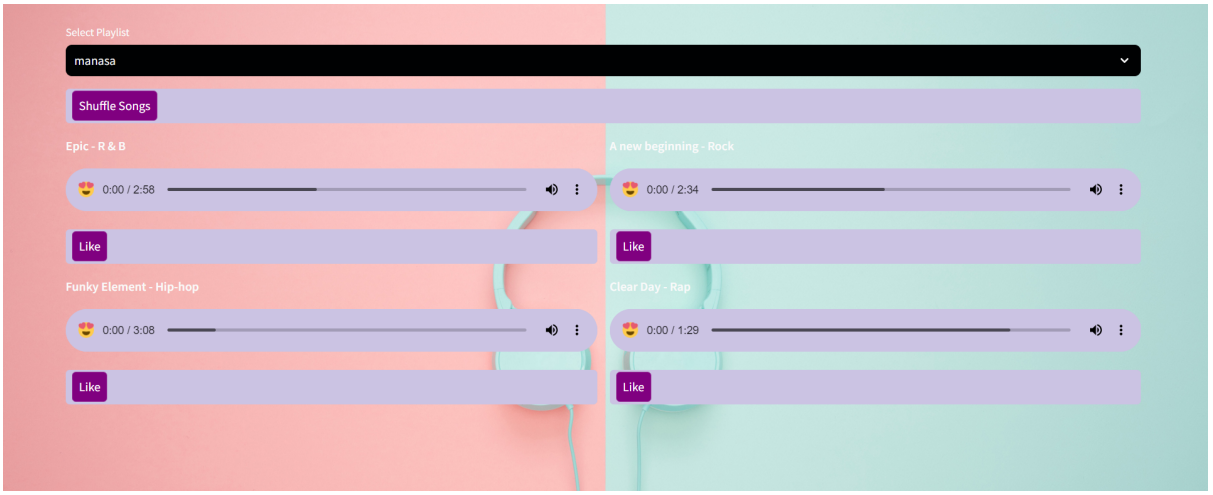
14.Add song to playlist


```
def add_songs_to_playlist(playlist_id, song_titles):
```

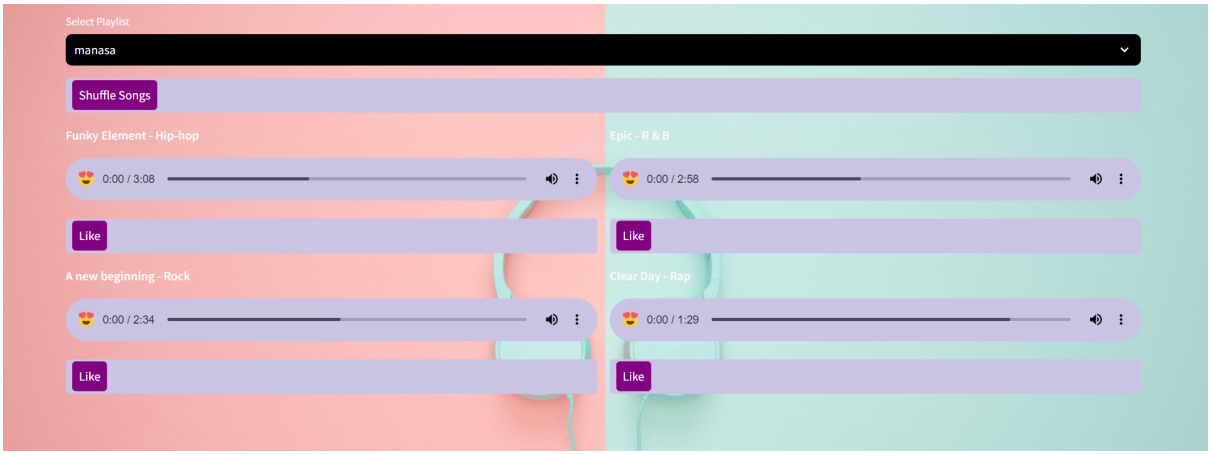
```
  cursor = conn.cursor(buffered=True)
  for song_title in song_titles:
      # Look up the song_id based on the song_title
      cursor.execute("SELECT id FROM songs WHERE title = %s", (song_title,))
      result = cursor.fetchone()

      if result:
          song_id = result[0]
          cursor.execute(
              "INSERT INTO playlistssongs (songId, playlistId, playlistOrder) VALUES (%s,
%s, %s)", (song_id, playlist_id, 0))
      else:
          st.warning(f"Song not found: {song_title}")

  cursor.close()
  conn.commit()
  st.success("Songs added to the playlist successfully!")
```

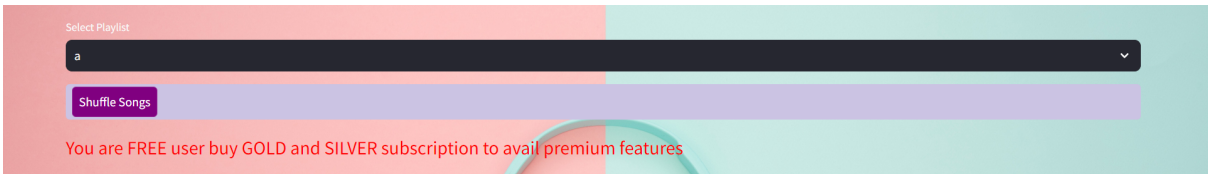
15. Listen to songs in playlist

Before suffle button is clicked



After suffle button is clicked

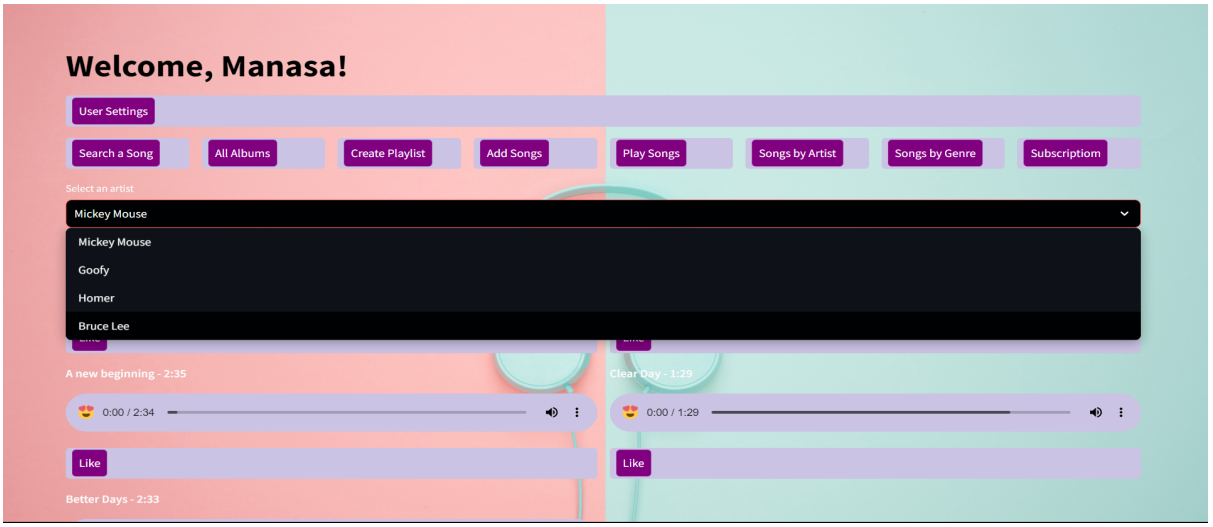




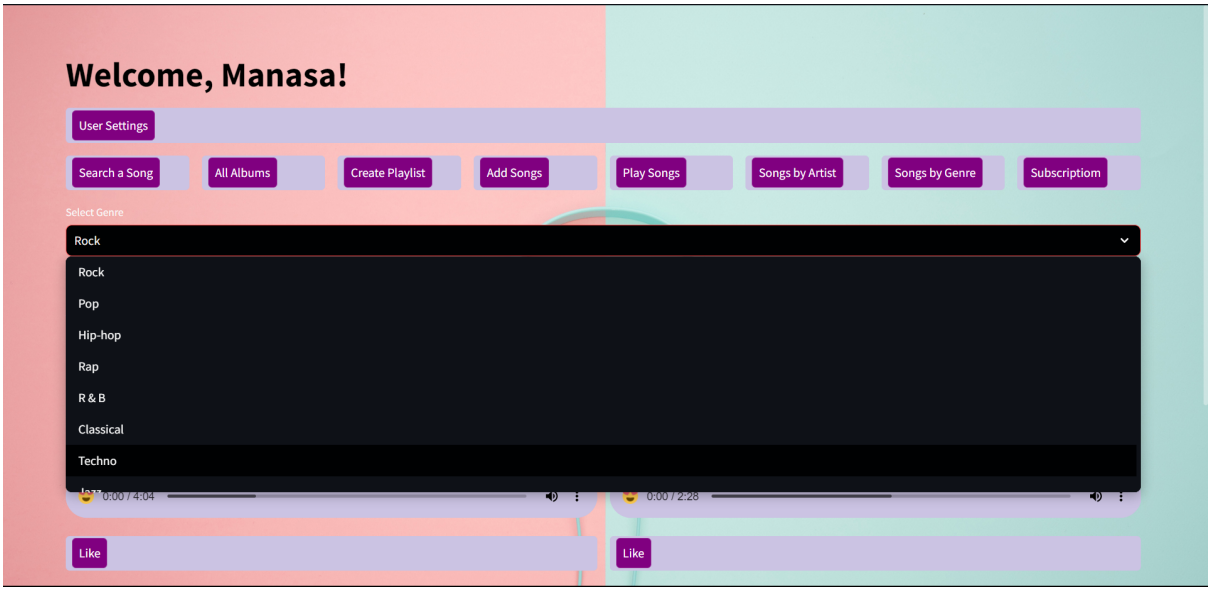
If the user has FREE subscription then the suffle button wont work and will prompt teh user to buy other premium subscription.

```
if (not (st.session_state.button12) and not (st.session_state.button8)):
    if shuffle_button:
        if st.session_state.plan_name != "FREE":
            play_all_songs(playlist_id, user_id, shuffle=True)
        else:
            styled_text = f"<p style='font-size: 24px; color: red;'>You are a FREE user. Buy GOLD and SILVER subscriptions to avail premium features</p>"
            st.markdown(styled_text, unsafe_allow_html=True)
            play_all_songs(playlist_id, user_id)
    else:
        play_all_songs(playlist_id, user_id)
```

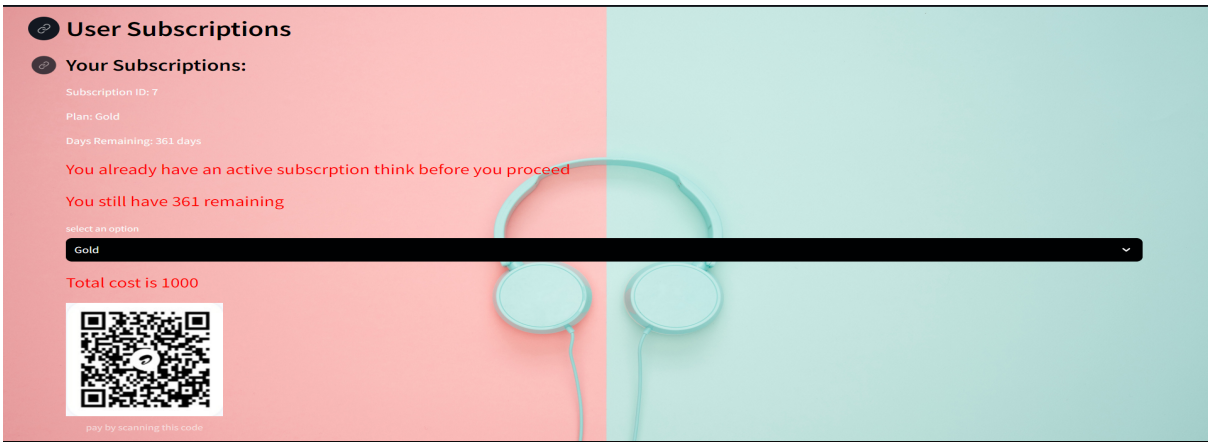
16.Play songs by artist



17.Play songs by genre



18.User Subscription



Different subscriptions have different money to be paid .

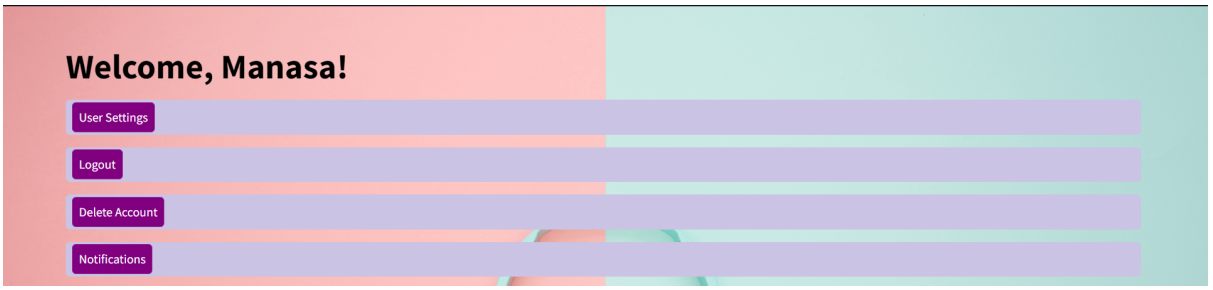
```
def is_stored_procedure_exists(cursor, procedure_name):
    query = """
    SELECT COUNT(*)
    FROM information_schema.ROUTINES
    WHERE ROUTINE_TYPE = 'PROCEDURE' AND ROUTINE_NAME = %s;
    """
    cursor.execute(query, (procedure_name,))
    result = cursor.fetchone()
    return result[0] > 0
```

```
stored_procedure = """
CREATE PROCEDURE UpdateUserSubscription(IN userId INT, IN planName
VARCHAR(255), IN price DECIMAL(10, 2), IN startDate DATE, IN endDate DATE)
BEGIN
    INSERT INTO subscriptions (userid, planname, price, startdate, enddate)
    VALUES (userId, planName, price, startDate, endDate)
    ON DUPLICATE KEY UPDATE
        planname = VALUES(planname),
        price = VALUES(price),
        startdate = VALUES(startdate),
        enddate = VALUES(enddate);
END
"""
cursor = conn.cursor()
if st.session_state.sub:
            cursor = conn.cursor()
            st.header("User Subscriptions")
            cursor.execute(
                "SELECT id, planname, enddate FROM subscriptions WHERE UserID = %s",
(st.session_state.user_id,))
            subscriptions = cursor.fetchall()
            remaining_days = 0
            plan_name = ""
            # Display user subscriptions
            if subscriptions:
                st.subheader("Your Subscriptions:")
                # Calculate when the subscriptions will get over
                today = date.today()
                for subscription in subscriptions:
                    subscription_id, plan_name, end_date = subscription
                    remaining_days = (end_date - today).days
                    st.write(f"Subscription ID: {subscription_id}")
                    st.write(f"Plan: {plan_name}")
                    st.write(f"Days Remaining: {remaining_days} days")

            else:
                st.write("You don't have any active subscriptions.")
```

```
            st.subheader("Buy Subscrption")
        if remaining_days > 0 and (plan_name == 'Gold' or plan_name == 'Silver'):
            styled_text = f"<p style='font-size: 24px; color: red;'>You already have an active
subscrption think before you proceed</p>"
            st.markdown(styled_text, unsafe_allow_html=True)
            styled_text = f"<p style='font-size: 24px; color: red;'>You still have
{remaining_days} remaining</p>"
            st.markdown(styled_text, unsafe_allow_html=True)
        selected_option = st.selectbox(
            "select an option", ['Gold', 'Silver'])
        price = 0
        today = date.today()
        end_date = date.today()
        plan_name = " "
        if selected_option == "Gold":
            price = 1000
            plan_name = "Gold"
            end_date = today + timedelta(days=365)
        if selected_option == "Silver":
            price = 500
            plan_name = "Silver"
            end_date = today + timedelta(days=180)
        styled_text = f"<p style='font-size: 24px; color: red;'>Total cost is {price}</p>"
        st.markdown(styled_text, unsafe_allow_html=True)
        st.image(
            "a12.jpeg", caption="pay by scanning this code", width=200)
        today = date.today()
        con = st.button("Confirm Payment")
        if (con):
            cursor.execute("UPDATE subscriptions SET PlanName = %s, EndDate = %s,
StartDate = %s, Price =%s WHERE UserID = %s ", (
                plan_name, end_date, today, price, st.session_state.user_id))
            conn.commit()
            st.session_state.plan_name = plan_name
            st.success(
                f"Payment for {plan_name} successful! Enjoy your subscription!")
```

```
        cursor.close()
        conn.commit()
```

18.User Settings Button




```
# Display buttons when User Settings is clicked
if user_settings:
    st.session_state.user_settings_open = not st.session_state.user_settings_open

if st.session_state.user_settings_open:

    # Buttons for Logout, Delete Account, and Notifications
    if st.button("Logout"):
        # Handle Logout logic
        logout()
        st.session_state.user_id = None
        st.session_state.username = None

    if st.button("Delete Account"):
        # Handle Delete Account logic
        delete_user_and_entries(st.session_state.user_id)
        st.session_state.user_id = None
        st.session_state.username = None

    if st.button("Notifications"):
        # Handle Notifications logic
        notifications_page(st.session_state.user_id)
        st.session_state.user_settings_open = not st.session_state.user_settings_open
```

19. Notification button

VI. Procedures/ Functions/ Triggers

Notification when a new account is created using triggers:

```
DELIMITER //

CREATE TRIGGER `after_insert_user` AFTER INSERT ON `users`
FOR EACH ROW
BEGIN
 DECLARE target_user_id INT;
 SET target_user_id = NEW.id;  -- Assuming you want to track notifications for the newly inserted user

 IF NEW.id = target_user_id THEN
   INSERT INTO `notifications` (`user_id`, `message`) VALUES (target_user_id, 'Welcome! To MusicXperience.');
 END IF;
END //

DELIMITER ;
```

Notification when a subscription is changed using triggers:

```
--Subscription notification
DELIMITER //
CREATE TRIGGER after_insert_subscription
AFTER UPDATE ON subscriptions
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('Subscription changed to: ', NEW.planname), NOW()
  FROM users;
END;
//
DELIMITER ;
```

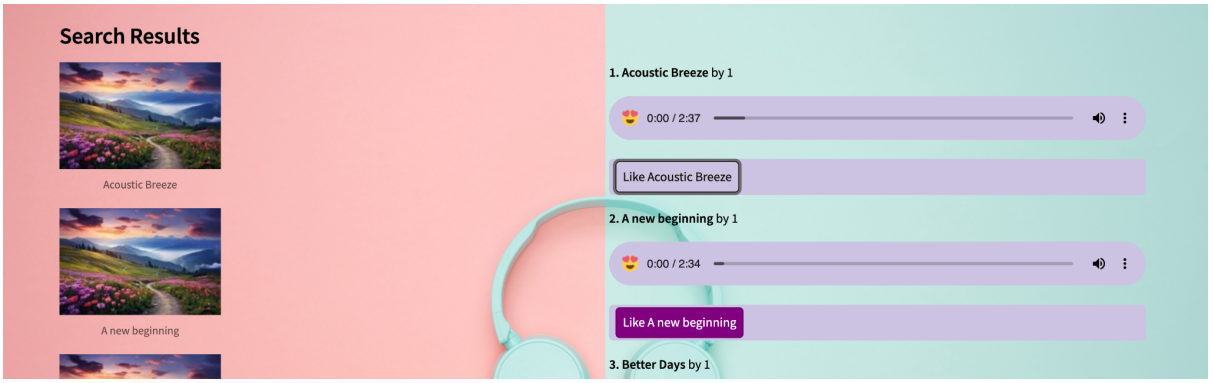
Notification when an artist, album or song is added using triggers:

```
--Artist Notification
DELIMITER //
CREATE TRIGGER after_insert_artist
AFTER INSERT ON artists
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New artist added: ', NEW.name), NOW()
  FROM users;
END;
//
DELIMITER ;

--Song Notification
DELIMITER //
CREATE TRIGGER after_insert_song
AFTER INSERT ON songs
FOR EACH ROW
BEGIN
```

```
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New song added: ', NEW.title), NOW()
  FROM users;
END;
//
DELIMITER ;

--Album notification
DELIMITER //
CREATE TRIGGER after_insert_album
AFTER INSERT ON albums
FOR EACH ROW
BEGIN
  -- Insert a notification for each user
  INSERT INTO notifications (user_id, message, timestamp)
  SELECT id, CONCAT('New album created: ', NEW.title), NOW()
  FROM users;
END;
//
DELIMITER ;
```

Notification when Like count is increment in songs table when like button is pressed:

```sql
-- Create a trigger that fires after an insert on the likes table
DELIMITER //
CREATE TRIGGER after_like_insert
AFTER INSERT ON likes
FOR EACH ROW
BEGIN
  -- Increment the like count in the song table
  UPDATE songs
  SET likes = likes + 1
  WHERE id = NEW.song_id;
END;
//
DELIMITER ;
```



Like count before button click:

```
mysql> select * from songs;
+----+-----------------+--------+-------+-------+----------+---------------------------------------+------------+-------+-------+
| id | title           | artist | album | genre | duration | path                                  | albumOrder | plays | likes |
+----+-----------------+--------+-------+-------+----------+---------------------------------------+------------+-------+-------+
|  1 | Acoustic Breeze |      1 |     5 |     8 | 2:37     | assets/music/bensound-acousticbreeze.mp3 |          1 |    11 |     1 |
|  2 | A new beginning |      1 |     5 |     1 | 2:35     | assets/music/bensound-anewbeginning.mp3 |          2 |     9 |     0 |
|  3 | Better Days     |      1 |     5 |     2 | 2:33     | assets/music/bensound-betterdays.mp3  |          3 |     6 |     0 |
|  4 | Buddy           |      1 |     5 |     3 | 2:02     | assets/music/bensound-buddy.mp3       |          4 |     7 |     0 |
|  5 | Clear Day       |      1 |     5 |     4 | 1:29     | assets/music/bensound-clearday.mp3    |          5 |     7 |     0 |
|  6 | Going Higher    |      2 |     1 |     1 | 4:04     | assets/music/bensound-goinghigher.mp3 |          1 |     9 |     0 |
|  7 | Funny Song      |      2 |     4 |     2 | 3:07     | assets/music/bensound-funnysong.mp3   |          2 |    11 |     0 |
|  8 | Funky Element   |      2 |     1 |     3 | 3:08     | assets/music/bensound-funkyelement.mp3 |          2 |     6 |     0 |
```

Like count after button click:

```
mysql> select * from songs;
+----+-----------------+--------+-------+-------+----------+----------------------------------------+------------+-------+-------+
| id | title           | artist | album | genre | duration | path                                   | albumOrder | plays | likes |
+----+-----------------+--------+-------+-------+----------+----------------------------------------+------------+-------+-------+
|  1 | Acoustic Breeze |      1 |     5 |     8 | 2:37     | assets/music/bensound-acousticbreeze.mp3 |          1 |    12 |     2 |
|  2 | A new beginning |      1 |     5 |     1 | 2:35     | assets/music/bensound-anewbeginning.mp3  |          2 |    10 |     0 |
|  3 | Better Days     |      1 |     5 |     2 | 2:33     | assets/music/bensound-betterdays.mp3     |          3 |     7 |     0 |
|  4 | Buddy           |      1 |     5 |     3 | 2:02     | assets/music/bensound-buddy.mp3          |          4 |     7 |     0 |
|  5 | Clear Day       |      1 |     5 |     4 | 1:29     | assets/music/bensound-clearday.mp3       |          5 |     8 |     0 |
|  6 | Going Higher    |      2 |     1 |     1 | 4:04     | assets/music/bensound-goinghigher.mp3    |          1 |     9 |     0 |
|  7 | Funny Song      |      2 |     4 |     2 | 3:07     | assets/music/bensound-funnysong.mp3      |          2 |    11 |     0 |
|  8 | Funky Element   |      2 |     1 |     3 | 3:08     | assets/music/bensound-funkyelement.mp3   |          2 |     6 |     0 |
|  9 | Extreme Action  |      2 |     1 |     4 | 8:03     | assets/music/bensound-extremeaction.mp3  |          3 |    13 |     0 |
```

Procedure for Album deletion:

```
def delete_album(album_id):
  cursor = conn.cursor(buffered=True)
  if is_procedure_exists(cursor, 'DeleteAlbum'):
     # If it exists, call the procedure
     cursor.callproc('DeleteAlbum', (album_id,))
        # Commit the changes
     cursor.fetchall()
     cursor.close()
     conn.commit()
  else:
     cursor.execute('''CREATE PROCEDURE DeleteAlbum(IN album_id INT)
                BEGIN
                   DELETE FROM albums WHERE id= album_id;
                END;''')
     cursor.callproc('DeleteAlbum', (album_id,))
        # Commit the changes
     cursor.fetchall()
     cursor.close()
     conn.commit()
  st.success("Album deleted successfully!")
```

Procedure for Artist Deletion:

```
def delete_artist(artist_id):
  cursor = conn.cursor(buffered=True)
  if is_procedure_exists(cursor, 'DeleteArtists'):
     # If it exists, call the procedure
     cursor.callproc('DeleteArtist', (artist_id,))
        # Commit the changes
     cursor.fetchall()
     cursor.close()
     conn.commit()
  else:
     cursor.execute('''CREATE PROCEDURE DeleteArtist(IN artistId INT)
                  BEGIN
                     DELETE FROM artists WHERE id = artistId;
                  END;''')
     cursor.callproc('DeleteArtist', (artist_id,))
        # Commit the changes
     cursor.fetchall()
     cursor.close()
     conn.commit()
  st.success("Artist deleted successfully!")
```

Procedure for Playlist Creation:

```
def create_playlist(name, owner):
  date_created = datetime.now()
  cursor = conn.cursor(buffered=True)

  if is_procedure_exists(cursor, 'InsertPlaylist'):
     cursor.callproc('InsertPlaylist', (name, owner, date_created))
     cursor.fetchall()
     cursor.close()
     conn.commit()
  else:
```

```
    cursor.execute('''CREATE PROCEDURE InsertPlaylist(IN playlistName
VARCHAR(255), IN playlistOwner INT, IN playlistDateCreated DATE)
                BEGIN
                    INSERT INTO playlists (name, owner, dateCreated) VALUES (playlistName,
playlistOwner, playlistDateCreated);
                END;''')
    cursor.callproc('InsertPlaylist', (name, owner, date_created))
    cursor.fetchall()
    cursor.close()
    conn.commit()
  st.success(f"Playlist '{name}' created successfully!")
```

Procedure for Playing Songs:

```
def play_song(song_path):
  st.audio(song_path, format="audio/mp3", start_time=0)
  cursor = conn.cursor(buffered=True)
  if is_procedure_exists(cursor, 'IncrementSongPlays'):
    # If it exists, call the procedure
    cursor.callproc('IncrementSongPlays', (song_path,))
    # Commit the changes
    cursor.fetchall()
    conn.commit()
    cursor.close()
  else:
    cursor.execute('''CREATE PROCEDURE IncrementSongPlays(IN songPath
VARCHAR(255))
                BEGIN
                    DECLARE currentPlays INT;

                    -- Get the current number of plays
                    SELECT plays INTO currentPlays FROM songs WHERE path = songPath;
```

```
                -- Update the 'plays' column by incrementing it by one
                UPDATE songs SET plays = currentPlays + 1 WHERE path = songPath;

                -- Commit the changes
                COMMIT;
            END"')
cursor.callproc('IncrementSongPlays', (song_path,))
cursor.fetchall()

conn.commit()
cursor.close()
```