



Bilkent University

Department of Computer Science

---

# CS353 DATABASE SYSTEMS

Sudo: Music listening and sharing platform

## Design Report

Doruk Çakmakçı  
Hakan Sarp Aydemir  
Mehmet Oğuz Göçmen  
Umut Berk Bilgiç

**Course Instructor:** Abdullah Ercüment Çiçek

2.4.2018

## **Table of Contents**

1. Revised E/R Diagram
  - 1.a. Revisions
  - 1.b. The Revised E/R Diagram
2. Set of Relations Resulted From E/R Conversion
3. Design Specifications
  - 3.a. Functional Components
  - 3.b. User Interface Design
  - 3.c. Advanced Database Components
4. Implementation Plan
5. Website

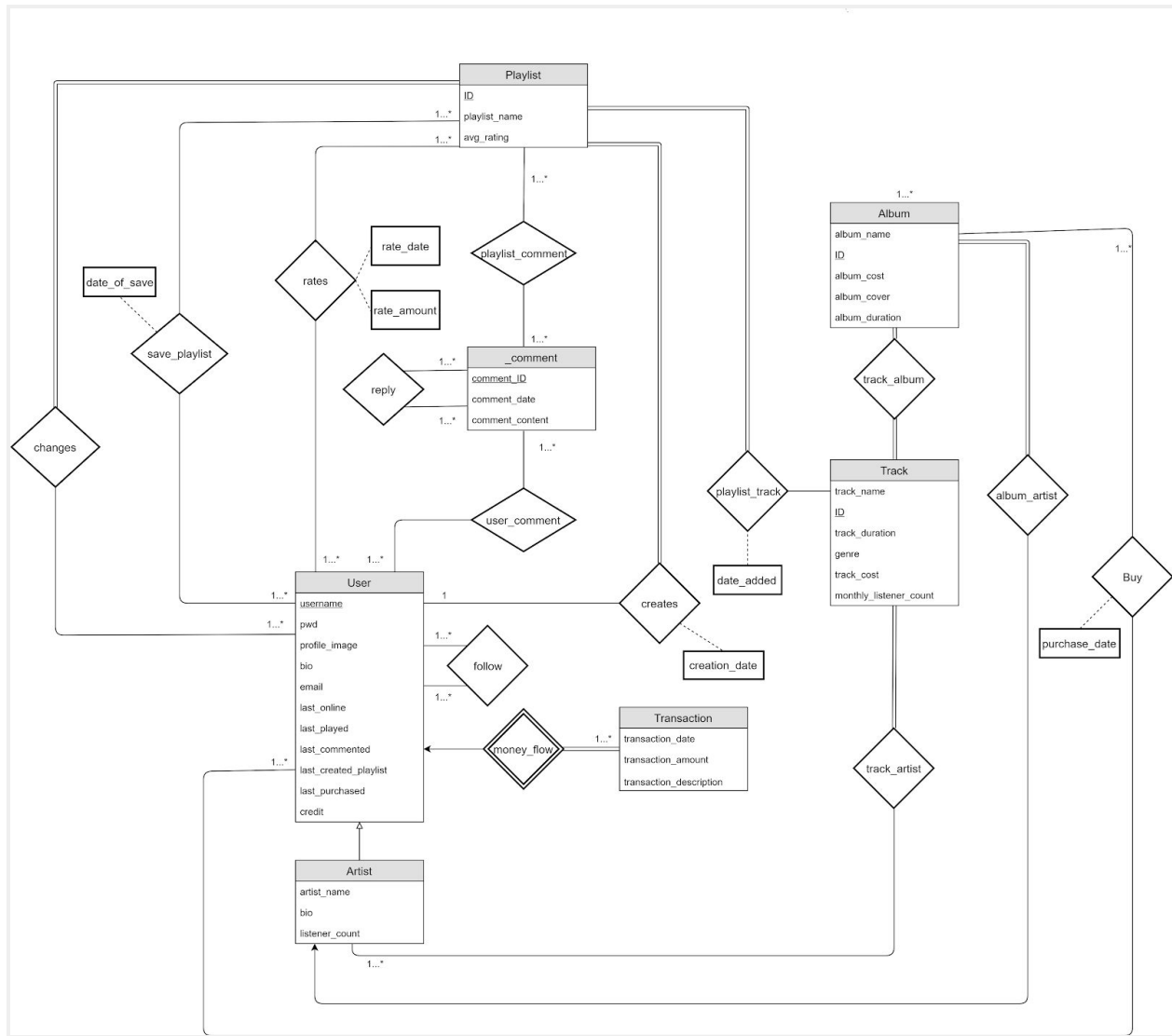
# **1. Revised E/R Diagram**

According to the feedback of Arif Usta for our E/R diagram in project proposal report we have revised the E/R diagram of Sudo. The revised E/R diagram and a list of revisions is given below.

## **1.a Revisions**

- “Library”, “Album” and “Single” entities are removed to eliminate redundancy and presence of artificial entities. Current design treats singles as an edge case of album.
- The entity named “Product” is renamed as “Album”.
- The arrow between “track\_album” relationship and “Product” entity is changed to represent total participation.
- The relationship named “product\_artist” is renamed as “album\_artist”
- The cardinality of “Buy” relationship is changed from one-to-one to many-to-many.
- The relationship named “friends\_with” is renamed as “follow”. Also the cardinality of the relationship is changed from one-to-many to many-to-many.
- The relationship named “comment” with attributes named “date” and “content” is replaced with an entity named “\_comment” with attributes named “comment\_date” and “content”. This entity is connected to “User” and “Playlist” entities with relationships named “user-comment” and “playlist-comment” respectively. The relationships mentioned previously have cardinality of many-to-many.
- A relationship called “reply” is added. This relationship connects comments with role “parent” to comments with role “child”.
- A relationship named “rates” with attributes “rate\_date” and “rate\_amount” and with cardinality of many-to-many on both sides is added to connect “Playlist” and “User” entities. This modification allows users to rate for a playlist.
- “avg\_rating” attribute is added to “Playlist” entity to demonstrate average rating of a playlist.
- A relationship named “save\_playlist” with attributes “date\_of\_save” and with cardinality of many-to-many on both sides is added to connect “Playlist” and “User” entities. This modification allows users to save a playlist to their profile.
- A relationship named “modifies” with cardinality of many-to-many on both sides is added to connect “Playlist” and “User” entities. Whereas, the relationship is total on playlist side since a playlist is modified at least by it’s creator. This modification allows different users to modify a single playlist thus, adds the functionality of collaborated playlist to SUDO.
- The participation of “Playlist” entity in “playlist\_track” relationship is changed from partial to total since, all playlists contain at least one track.
- As an additional functionality to the description in topic document, playlists can have more than one modifier users. This allows collaborated playlists. Also playlists have an average rating stored in “avg\_rating” attribute of playlist.

## 1.b The Revised E/R Diagram:



## 2. Set of Relations Resulted From E/R Conversion

### 2.1 User

#### **Relational Model:**

User( username, pwd, profile\_image, bio, email, last\_online, last\_played, last\_commented, last\_created\_playlist, last\_purchased, credit )

FK: last\_played references Track(ID)

FK: last\_commented references \_comment(comment\_date)

FK: last\_created\_playlist references Playlist(ID)

FK: last\_purchased references Track(ID)

#### **Functional Dependencies:**

username → pwd, profile\_image, bio, email, last\_online, last\_played, last\_commented, last\_created\_playlist, last\_purchased, credit

email → username, pwd, profile\_image, bio, last\_online, last\_played, last\_commented, last\_created\_playlist, last\_purchased, credit

#### **Candidate Keys:**

{ (username), (email) }

#### **Table Definition:**

```
CREATE TABLE User(  
  username          VARCHAR(30) PRIMARY KEY,  
  pwd               VARCHAR(20) NOT NULL,  
  profile_image     LONGBLOB,  
  bio               VARCHAR(140),  
  email             VARCHAR(50) NOT NULL,  
  last_online       DATETIME,  
  last_played       VARCHAR(50),  
  last_commented    DATETIME,  
  last_created_playlist VARCHAR(50),
```

```
last_purchased          VARCHAR(50),
credit                  REAL,
FOREIGN KEY (last_played) references Track(ID)
on delete cascade on update cascade,
FOREIGN KEY (last_commented) references _comment(comment_date)
on delete cascade on update cascade,
FOREIGN KEY (last_created_playlist) references Playlist(ID)
on delete cascade on update cascade,
FOREIGN KEY (last_purchased) references Playlist(ID)
on delete cascade on update cascade,
check(email like ('%@%')),
check(credit >= 0)
);
```

## **2.2 Artist**

### **Relational Model:**

Artist(username, artist\_name, bio, listener\_count)

FK: username references User (username)

### **Functional Dependencies:**

username  $\rightarrow$  artist\_name, bio, listener\_count

### **Candidate Keys:**

{ (username) }

### **Table Definition:**

```
CREATE TABLE Artist(  
  username VARCHAR(30),  
  artist_name VARCHAR(30),  
  bio VARCHAR(140),  
  listener_count INT,  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  PRIMARY KEY (username)  
);
```

## 2.3 Track

### **Relational Model:**

Track( ID, track\_name, track\_duration, genre, track\_cost, monthly\_listener\_count)

### **Functional Dependencies:**

ID  $\rightarrow$  track\_name, duration, genre, cost, monthly\_listener\_count

### **Candidate Keys:**

{ (ID) }

### **Table Definition:**

```
CREATE TABLE Track(  
  ID                VARCHAR(30),  
  track_name        VARCHAR(20),  
  track_duration    TIME NOT NULL,  
  genre             VARCHAR(20),  
  track_cost        REAL,  
  monthly_listener_count INT,  
  check( monthly_listener_count >= 0 )  
);
```



## 2.4 Album

### Relational Model:

Album( ID, album\_name, album\_cost, album\_cover, album\_duration, artist\_username)

FK: artist\_username references Artist(username)

### Functional Dependencies:

ID  $\rightarrow$  album\_name, album\_cost, album\_cover, album\_duration, artist\_username

### Candidate Keys:

{ (ID) }

### Table Definition:

```
CREATE TABLE Album (  
  ID VARCHAR(30) PRIMARY KEY,  
  album_name VARCHAR(20) NOT NULL,  
  album_cost INT,  
  album_duration TIME NOT NULL,  
  artist_username VARCHAR(30),  
  FOREIGN KEY (artist_username) references Artist(username)  
  on delete cascade on update cascade,  
  check(cost >= 0)  
);
```

## **2.5 Playlist**

### **Relational Model:**

Playlist( playlist\_ID, playlist\_name, creator\_username, avg\_rating, creation\_date)

FK: creator\_username references User(username)

### **Functional Dependencies:**

playlist\_ID → playlist\_name, creator\_username, avg\_rating, creation\_date

### **Candidate Keys:**

{ (playlist\_ID) }

### **Table Definition:**

```
CREATE TABLE Playlist(  
    playlist_ID          VARCHAR(30) PRIMARY KEY,  
    playlist_name        VARCHAR(20),  
    creator_username     VARCHAR(20),  
    avg_rating           INT,  
    creation_date        DATETIME,  
    FOREIGN KEY (creator_username) REFERENCES User(username)  
    on delete cascade on update cascade,  
    CHECK ( (avg_rating > 0) AND (avg_rating < 10) )  
);
```

## **2.6 Comment**

### **Relational Model:**

\_comment(comment\_ID, comment\_date, comment\_content)

### **Functional Dependencies:**

comment\_ID  $\rightarrow$  comment\_date, comment\_content

### **Candidate Keys:**

{{comment\_ID}}

### **Table Definition:**

```
CREATE TABLE _comment(  
  comment_ID          VARCHAR(30) PRIMARY KEY AUTO INCREMENT,  
  comment_date        DATETIME,  
  comment_content     VARCHAR(140)  
);
```

## 2.7 Transaction

### **Relational Model:**

Transaction(username, transaction\_description, transaction\_date, transaction\_amount)

FK: username references User (username)

### **Functional Dependencies:**

username, transaction\_description  $\rightarrow$  transaction\_date, transaction\_amount

### **Candidate Keys:**

{ (username) }

### **Table Definition:**

```
CREATE TABLE Transaction(  
  username VARCHAR(30) PRIMARY KEY,  
  transaction_description VARCHAR(140),  
  transaction_date DATETIME,  
  transaction_amount REAL,  
  FOREIGN KEY (username) REFERENCES User(username)  
  on delete cascade on update cascade,  
  check(transaction_amount > 0),  
  PRIMARY KEY (username)  
);
```

## **2.8 Buy**

### **Relational Model:**

Buy(ID, username, purchase\_date)

FK: username references User (username)

FK: ID references Album (ID)

### **Functional Dependencies:**

ID, username  $\rightarrow$  purchase\_date

### **Candidate Keys:**

{ (ID, Username) }

### **Table Definition:**

```
CREATE TABLE Buy (  
  ID          VARCHAR(30),  
  username    VARCHAR(30),  
  purchase_date DATETIME,  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (ID) references Album(ID)  
  on delete cascade on update cascade  
);
```

## **2.9 playlist\_comment**

### **Relational Model:**

playlist\_comment(playlist\_ID, comment\_ID)

FK: playlist\_ID references Playlist (ID)

FK: comment\_ID references \_comment (ID)

### **Functional Dependencies:**

None

### **Candidate Keys:**

{{playlist\_ID, comment\_ID}}

### **Table Definition:**

```
CREATE TABLE playlist_comment(  
  playlist_ID VARCHAR(30),  
  comment_ID VARCHAR(30),  
  FOREIGN KEY (playlist_ID) references Playlist(ID)  
  on delete cascade on update cascade,  
  FOREIGN KEY (comment_ID) references _comment(ID)  
  on delete cascade on update cascade,  
  PRIMARY KEY (playlist_ID,comment_ID)  
);
```

## **2.10 user\_comment**

### **Relational Model:**

user\_comment( username, ID )

FK: username references User (username)

FK: ID references \_comment (ID)

### **Functional Dependencies:**

None

### **Candidate Keys:**

{{username, ID}}

### **Table Definition:**

```
CREATE TABLE user_comment(  
  username VARCHAR(30),  
  ID VARCHAR(30),  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (ID) references _Comment(ID)  
  on delete cascade on update cascade,  
  PRIMARY KEY (ID,username)  
);
```

## **2.11 track\_album**

### **Relational Model:**

track\_album(album\_ID, track\_ID)

FK: album\_ID references Album (ID)

FK: track\_ID references Track (ID)

### **Functional Dependencies:**

None

### **Candidate Keys:**

{{album\_ID, track\_ID}}

### **Table Definition:**

```
CREATE TABLE track_album(  
  album_ID VARCHAR(30),  
  track_ID VARCHAR(30),  
  FOREIGN KEY (album_ID) references Album(ID)  
  on delete cascade on update cascade,  
  FOREIGN KEY (track_ID) references Track(ID)  
  on delete cascade on update cascade,  
  PRIMARY KEY (album_ID, track_ID)  
);
```



## **2.12 track\_artist**

### **Relational Model:**

track\_artist(track\_ID, artist\_username)

FK: artist\_username references User (username)

FK: track\_ID references Track (ID)

### **Functional Dependencies:**

None

### **Candidate Keys:**

{{track\_ID, artist\_username}}

### **Table Definition:**

```
CREATE TABLE track_artist(  
  track_ID VARCHAR(30),  
  artist_username VARCHAR(30),  
  FOREIGN KEY (artist_username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (track_ID) references Track(ID)  
  on delete cascade on update cascade,  
  PRIMARY KEY (track_ID, artist_username)  
);
```

## **2.13 playlist\_track**

### **Relational Model:**

playlist\_track(playlist\_ID, track\_ID, date\_added)

FK: playlist\_ID references Playlist (ID)

FK: track\_ID references Track (ID)

### **Functional Dependencies:**

playlist\_ID, track\_ID → date\_added

### **Candidate Keys:**

{{playlist\_ID, track\_ID}}

### **Table Definition:**

```
CREATE TABLE playlist_track(  
  playlist_ID VARCHAR(30),  
  track_ID VARCHAR(30),  
  date_added DATETIME,  
  FOREIGN KEY (playlist_ID) references Playlist(ID)  
  on delete cascade on update cascade,  
  FOREIGN KEY (track_ID) references Track(ID)  
  on delete cascade on update cascade,  
  PRIMARY KEY (playlist_ID, track_ID)  
);
```

## **2.14 rates**

### **Relational Model:**

rates(playlist\_ID, username, rate\_date, rate\_amount)

FK: username references User(username)

FK: playlist\_ID references to playlist(ID)

### **Functional Dependencies:**

playlist\_ID, username  $\rightarrow$  rate\_date, rate\_amount

### **Candidate Keys:**

{ (playlist\_ID, username) }

### **Table Definition:**

```
CREATE TABLE rates (  
  playlist_ID          VARCHAR(30),  
  username             VARCHAR(30),  
  rate_date            DATETIME,  
  rate_amount          INT,  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (playlist_ID) references Playlist(ID)  
  on delete cascade on update cascade,  
  check( (rate_amount > 0) AND (rate_amount < 10) )  
);
```

## 2.15 changes

### Relational Model:

changes(playlist\_ID, username)

FK: username references User(username)

FK: playlist\_ID references to playlist(ID)

### Functional Dependencies:

NONE

### Candidate Keys:

{ (playlist\_ID, username) }

### Table Definition:

```
CREATE TABLE changes (  
  playlist_ID          VARCHAR(30),  
  username             VARCHAR(30),  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (playlist_ID) references Playlist(ID)  
  on delete cascade on update cascade  
);
```

## **2.16 save\_playlist**

### **Relational Model:**

save\_playlist(playlist\_ID, username, date\_of\_save)

FK: username references User(username)

FK: playlist\_ID references to playlist(ID)

### **Functional Dependencies:**

playlist\_ID, username  $\rightarrow$  date\_of\_save

### **Candidate Keys:**

{ (playlist\_ID, username) }

### **Table Definition:**

```
CREATE TABLE save_playlist(  
  playlist_ID          VARCHAR(30),  
  username             VARCHAR(30),  
  date_of_save         DATETIME,  
  FOREIGN KEY (username) references User(username)  
  on delete cascade on update cascade,  
  FOREIGN KEY (playlist_ID) references Playlist(ID)  
  on delete cascade on update cascade  
);
```

## **2.17 reply**

### **Relational Model:**

reply(id\_parent, id\_child )

FK: id\_parent references \_comment(ID)

FK: id\_child references to \_comment(ID)

### **Functional Dependencies:**

NONE

### **Candidate Keys:**

{ (playlist\_ID, username) }

### **Table Definition:**

```
CREATE TABLE reply (  
  id_parent          VARCHAR(30),  
  id_child           VARCHAR(30),  
  FOREIGN KEY (id_parent) references _comment(ID)  
  on delete cascade on update cascade,  
  FOREIGN KEY (id_child) references _comment(ID)  
  on delete cascade on update cascade  
);
```

## **3. Design Specifications**

### **3.a. Functional Components**

#### **3.a.1. Scenarios**

##### **Registration**

Oğuz opens up the music playing system's website. The landing page of the website has a register part which asks for a username, password and e-mail address in order to complete registration. Oğuz enters credentials to the given text fields and clicks the register button to finish his registration process.

##### **Failed Authentication**

Oğuz opens up the music playing system's website and the system requests a login. Oğuz enters an email address and a password. The system displays an error message indicating that either the username or the password is wrong and prompts the user to try again with new credentials.

##### **Successful Authentication**

Oğuz opens up the music playing system's website and the system requests a login. Oğuz enters an email address and a password. The system directs the user to the main page of the music playing system.

##### **Searching For a Song and Playing It**

After successfully logging into the music playing system, Oğuz enters the song he wants to access into the search bar and presses enter on his keyboard. The system lists the set of possible results according to the string Oğuz entered into the search bar. Oğuz selects the desired song from the list. The song automatically starts to play if he already owns the song.

### **Not Being Able to Play a Non-purchased Song**

After successfully logging into the music playing system, Oğuz enters the song he wants to access into the search bar and presses enter on his keyboard. The system lists the set of possible results according to the string Oğuz entered into the search bar. Oğuz selects the desired song from the list. System creates a pop-up that indicate that the song needs to be purchased first.

### **Selecting a Song From a Playlist and Playing It**

After successfully logging into the music playing system, Oğuz clicks to the “My Playlists” button on the side menu bar. System displays the playlists that the user has access to. Oğuz selects the desired playlist from the list of playlists and clicks on the playlists name. System displays the list of tracks in the selected playlist. Oğuz selects the desired song from the list and clicks on it. The system starts playing the song.

### **Creating a Playlist**

After successfully logging into the music playing system, Oğuz clicks to the “My Playlists” button on the side menu bar. System displays the playlists that the user has access to. On the top right he clicks the “+” button. The system shows a popup with fields that set the playlist name and whether the playlist will be open to collaborators or not. After filling out the fields Oğuz clicks “OK” to finish the playlist creation process.

### **Adding a Song To a Playlist**

After successfully logging into the music playing system, Oğuz selects a song and clicks on it. The system creates a popup menu that shows available actions regarding that song. Oğuz first clicks on the “Add to playlist” button and then selects the playlist he wants from another popup menu next to the first one. After clicking on the playlist he wants to add the song to; the system adds the selected song to the selected playlist.

### **Removing a Song From a Playlist**

After successfully logging into the music playing system, Oğuz clicks to the “My Playlists” button on the side menu bar. System displays the playlists that the user has access to. Oğuz selects the desired playlist from the list of playlists and clicks on the playlists name. System displays the list of tracks in the selected playlist. Oğuz selects the desired song from the list that he wants to remove from the playlist. The system creates



a popup menu that shows the available actions regarding that song. Oğuz clicks the “Remove from playlist” button. The system removes the song from the selected playlist.

### **Commenting to a Playlist**

After successfully logging into the music playing system, Oğuz searches for a playlist by entering the name of the playlist to the search bar. A list of possible results are displayed by the system. Oğuz selects one of them and presses the “Add comment” button next to the selected playlist. Oğuz enters the comment and hits enter on his keyboard in order to save the comment to the playlist.

### **Replying to Another Comment**

After successfully logging into the music playing system, Oğuz searches for a playlist by entering the name of the playlist to the search bar. A list of possible results are displayed by the system. Oğuz selects one of them and presses the “comment” button next to the selected playlist. Comment section is displayed by the system. Oğuz selects the comment he wants to comment to and presses the arrow button (indicating reply) next to the selected comment. A new comment bar that is different from the comment bar reserved for commenting to the playlist is displayed by the system as a drop-down item. Oğuz enters the comment and hits enter on his keyboard in order to save the comment to the playlist

### **Rating a Playlist**

After successfully logging into the music playing system, Oğuz searches for a playlist by entering the name of the playlist to the search bar. A list of possible results are displayed by the system. Oğuz selects one of them and uses the star rating UI element to set a rating of his own. The system saves Oğuz’s rating to the playlist.

### **Following a user**

After successfully logging into the music playing system, Oğuz searches for a user by entering the username of the user to the search bar. A list of possible results are displayed by the system. Oğuz selects one of them and clicks the “Follow” button next to the selected user’s profile image.

### **Purchasing an album**

After successfully logging into the music playing system, Oğuz searches for an album by entering the name of the album to the search bar. A list of possible results are displayed by the system. Oğuz clicks on the desired album and system shows a popup window. Oğuz clicks on “Purchase” in order to purchase the album. The system deducts the cost of the album from Oğuz’s balance.

### **Insufficient Credits**

After successfully logging into the music playing system, Oğuz searches for an album by entering the name of the album to the search bar. A list of possible results are displayed by the system. Oğuz clicks on the desired album and system shows a popup window. Oğuz cannot click on the purchase button since it is disabled. System displays an “Insufficient Funds” message and tells him to add more funds in order to buy the album. No funds are deducted from Oğuz’s balance.

### **Saving Playlist to Local Playlist**

After successfully logging into the music playing system, Oğuz searches for a playlist by entering the name of the playlist into the search bar and pressing enter key. A list of possible results are displayed by the system. Oğuz selects one of them and clicks the “save” button on the popup window. System saves the selected playlist to the user’s local playlists.

### **Publish an Album (Artist)**

After successfully logging into the music playing system Post Malone clicks “Publish Album” button from the side menu. Post is prompted to select a album art image, set a price for the album and add songs to the album. Post also names the album and songs in the album. After he is done with these, he click on the “Publish Album” button to complete the process. The system receives the new album and displays it on Post’s page.

### **View Tracks/Albums of an Artist**

After successfully logging into the music playing system, Oğuz searches for an artist by entering the name of the artist into the search bar and pressing the enter key.

A list of possible results are displayed by the system. Oğuz selects one of them and clicks on the name of the selected artist. System displays the tracks and albums of the artist.

### **Unfollow a User**

After successfully logging into the music playing system, Oğuz presses the “Following” button on the profile page of a user he follows. The system removes this person from Oğuz’s follower list.

### **View Past Transactions**

After successfully logging into the music playing system, Oğuz presses the “past transactions” button on his balance page. The system displays all of the transactions that Oğuz made up until that moment including funds that were added and albums that were purchased.

### **View Your Comments**

After successfully logging into the music playing system, Oğuz presses the “My Comments” button on his profile page. The system displays all of the comments that Oğuz published until that moment and places them in a table.

### **View Followers**

After successfully logging into the music playing system, Oğuz accesses to his profile page and presses on his follower count. System displays a list of all of his followers.

### **Add Credit**

After Oğuz successfully logged in to the system, he clicks his balance on the top right. The system asks for credit card credentials. Oğuz enters these credentials and selects an amount of balance to be added using the balance selection buttons. Oğuz clicks the “Add Credit” button to finalize the process.

### **Delete a Comment**

After a successfully logging in to the system, Oğuz can select one of two ways to delete one or multiple of his comments. First he selects to view all of his previous comments by pressing the profile button. Then Oğuz presses “Comments” button. Then he is directed to a page with all of his comments. Oğuz clicks on the comment he wants to delete, to which the system redirects to. Oğuz clicks on the “Trashcan” icon to delete the comment he made.

Alternatively, after successfully logging into system, Oğuz selects the playlist where the comment which he wanted to delete is made to that playlist. Then Oğuz presses the comments button to view all comments made to that playlist. Oğuz clicks on the “Trashcan” icon to delete the comment he made.

### **Change profile picture**

After successfully logging into the music playing system, Oğuz presses the “settings” button on any page. Settings are displayed by the system. Oğuz presses the “change profile picture” button next to the profile picture icon. Oğuz chooses the picture he wants. System changes the picture accordingly.

### **Change password**

After successfully logging into the music playing system, Oğuz presses the “settings” button on any page. Settings are displayed by the system. Oğuz presses the “change password” button. Oğuz chooses his new password in the pop-up window for password changing that is displayed by the system and accepts. System updates the new password.

### **Change bio**

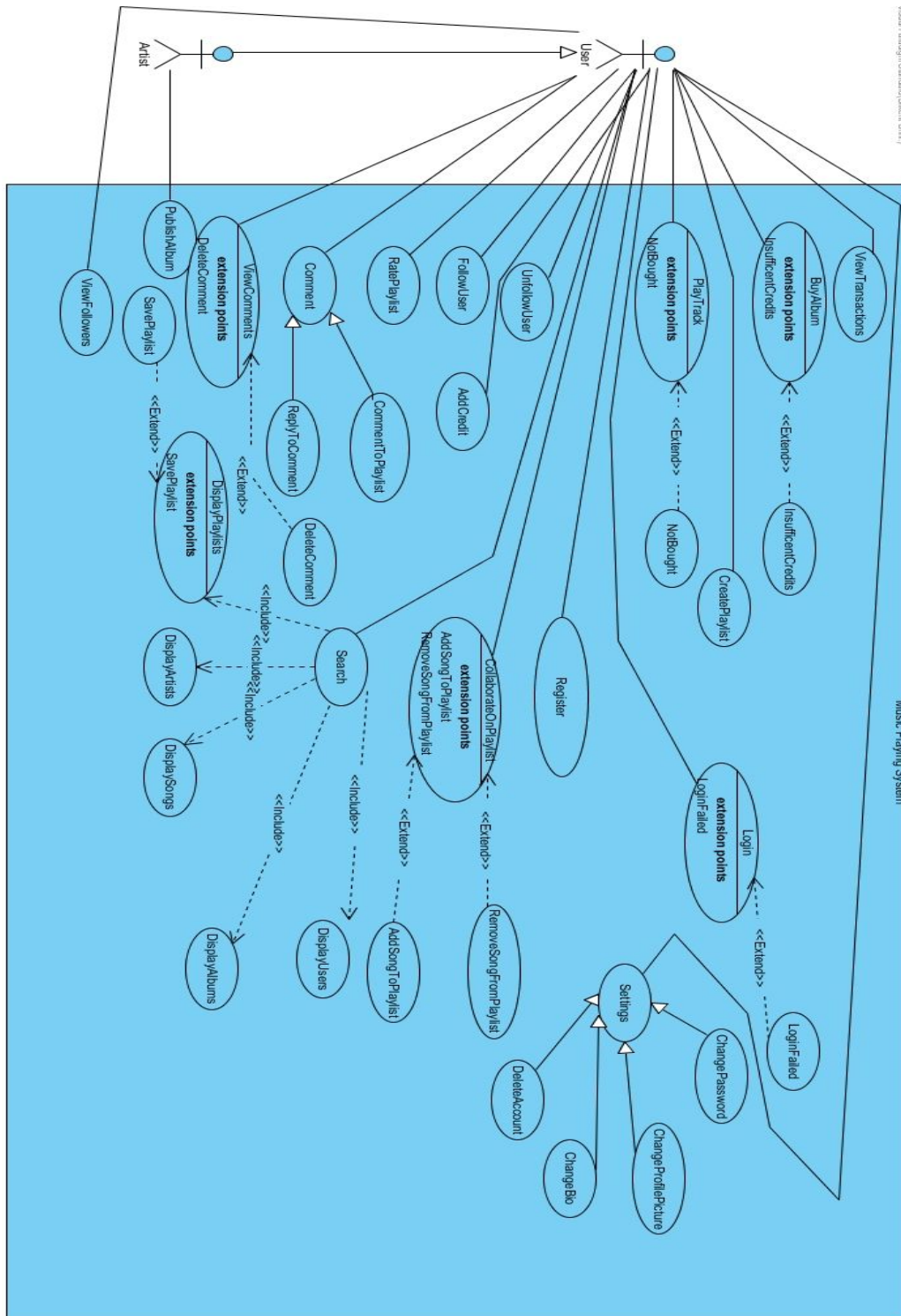
After successfully logging into the music playing system, Oğuz presses the “settings” button on any page. Settings are displayed by the system. Oğuz presses the “change bio” button. Oğuz chooses his new bio in the pop-up window for bio changing that is displayed by the system and accepts. System updates the new bio.

**Delete account**

After successfully logging into the music playing system, Oğuz presses the “settings” button on any page. Settings are displayed by the system. Oğuz presses the “delete my account” button. System deletes the account from the database.

### 3.a.2 Use Case Diagram

A general use case diagram from the point of view of User and Artist is given below:



## **3.a.3 Algorithms**

### **3.a.3.1 Transaction Related Algorithms**

Transactions are done by users. There are two different types of transaction, purchasing credits via a credit card and buying songs using these credits. To purchase credits, user must enter the “credit screen” by clicking on the icon that shows the current available credits that can be found on every page. User can then enter his/her credit card information and amount of credits to be added and proceed. The amount of credits that was chosen is added to the user’s balance. In the case that there is insufficient funds in the credit card entered, credit purchasing operation will be terminated. To buy songs with available credits, the user must select a song and press the purchase button that appears in a pop-up window if the album that the track is in is not bought. In case that there isn’t enough available credits to buy the album that the selected track is in, the purchase button that appears in the pop-up window is displayed as red and a message that indicates that the user doesn’t have enough credits available is displayed and transaction doesn’t occur.

### **3.a.3.2 Search Related algorithms**

Users can use the search bar to search for all kinds of data at once. When user enters a string to the search bar, a sample of each type of data is displayed by the system. These types are users, playlists, albums, tracks and artists.

### **3.a.3.3 Playlist Related Algorithms**

Playlists are a collection of tracks created by users. It has a name, a cover image (generated by using the album covers in the playlist), a creation date and a collaborator list. Any user can comment and reply to comments in a playlist regardless of their collaborative status. The creator can also set it to be a collaborative playlist. Any collaborator of a playlist can add or remove new tracks to the playlist however only the original creator has the ability to remove collaborators after they are added. A playlist also has a star rating. It can be given by any user out of 5 stars. The overall rating of a playlist is determined by counting the amount of stars it has and dividing by the number of total ratings it received. A playlist can only be deleted by the original creator of the playlist.

Once a user starts to play a playlist, the music player only plays songs from that playlist until the entire song list is exhausted or the user switches to a different listening context.

#### **3.a.3.4 Comment Related Algorithms**

Users can comment on playlists and other users' comments. To comment to a playlist or another comment, a user must enter the comment section of that playlist and comment from there. Users can also delete their comments by entering their own "comments" page where all of the comments that user has made are displayed. User must select a specific comment and press the delete comment button to delete the comments they have made.

### **3.a.4 Data Types**

#### **3.a.4.1 Date and Time**

Date and Time related data structures are needed throughout this project. For example: playlist\_track(date\_added), Buy(purchase\_date), rates(rate\_date), save\_playlist(date\_of\_save), transaction(transaction\_date) attributes of the corresponding tables are needed to be stored as date and/or time. In SQL; TIME, DATE, DATETIME data types are needed in the previously mentioned attributes.

#### **3.a.4.2 BLOB**

The album\_cover attribute of Album table and the profile\_image of User will hold images. To hold a large object as image BLOB(binary large object) data type is used.

#### **3.a.4.3 Numeric Types**

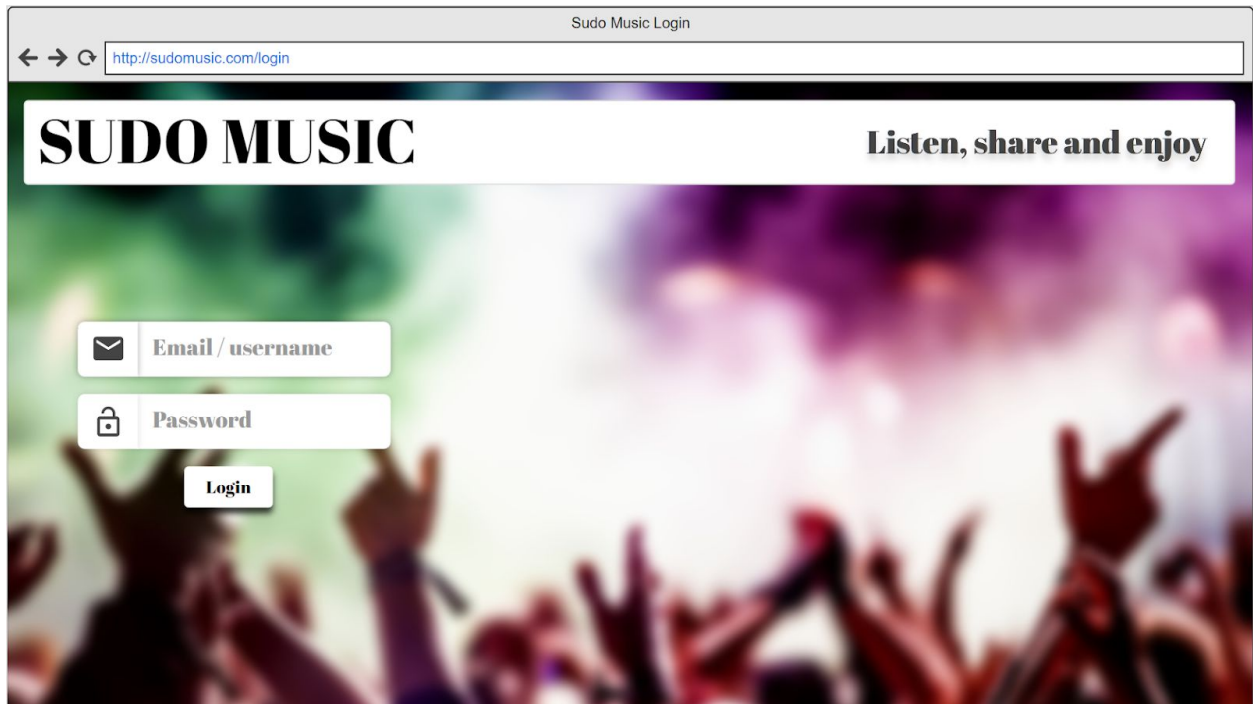
Numeric types are used in rate of playlist, monthly listener count of a track, costs of albums and tracks, listener\_count of an artist and transaction amount of a transaction. As numeric types: INT and REAL are used.

#### **3.a.4.4 String Types**

String types used in SQL statements of this project are CHAR and VARCHAR. All of the attributes other than those stated under BLOB, Numeric Types and Date and Time are VARCHAR or CHAR.



## 3.b User Interface Design



### 3.b.1 Login Screen

#### 3.b.1.1 Login

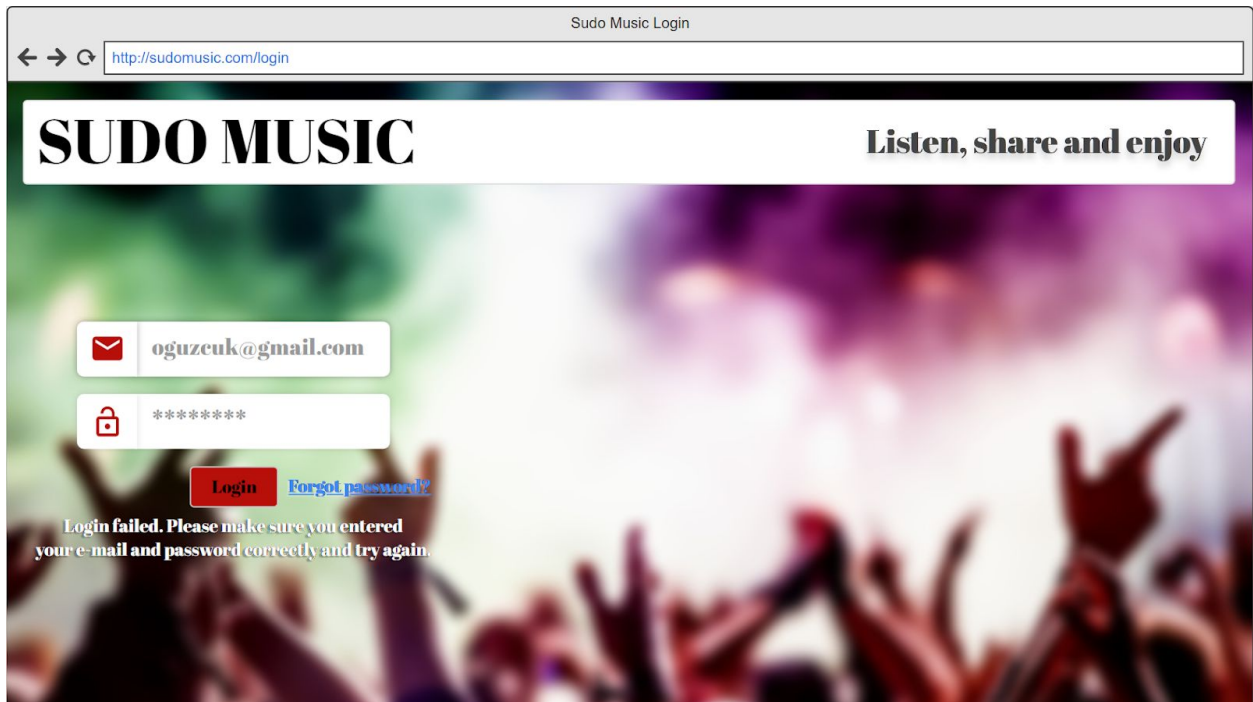
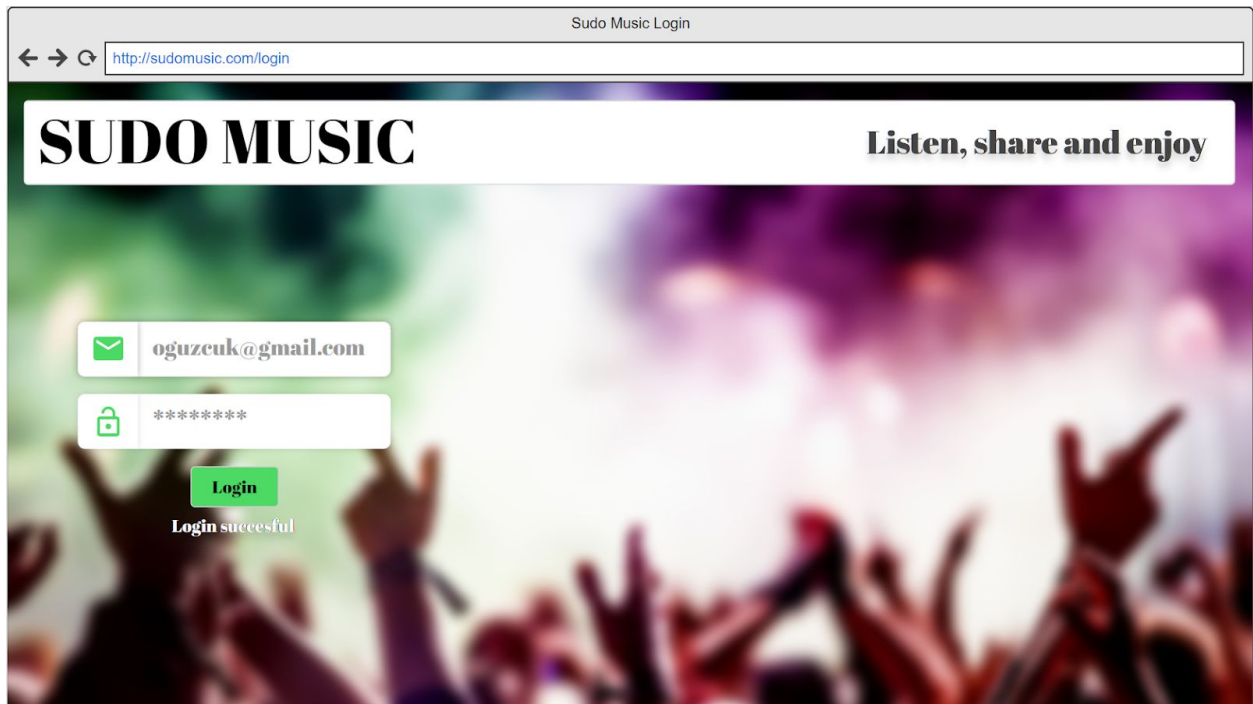
The login attempt of users of SUDO will be determined as successful or unsuccessful according to the result of this query.

Inputs:

@email: email written to the textbox in Login view.

@username: the username written to the textbox in Login view.

```
SELECT *  
FROM User  
WHERE (@email = email OR @username = username)  
      AND @password = pwd;
```



### 3.b.1.2 Forgot Password

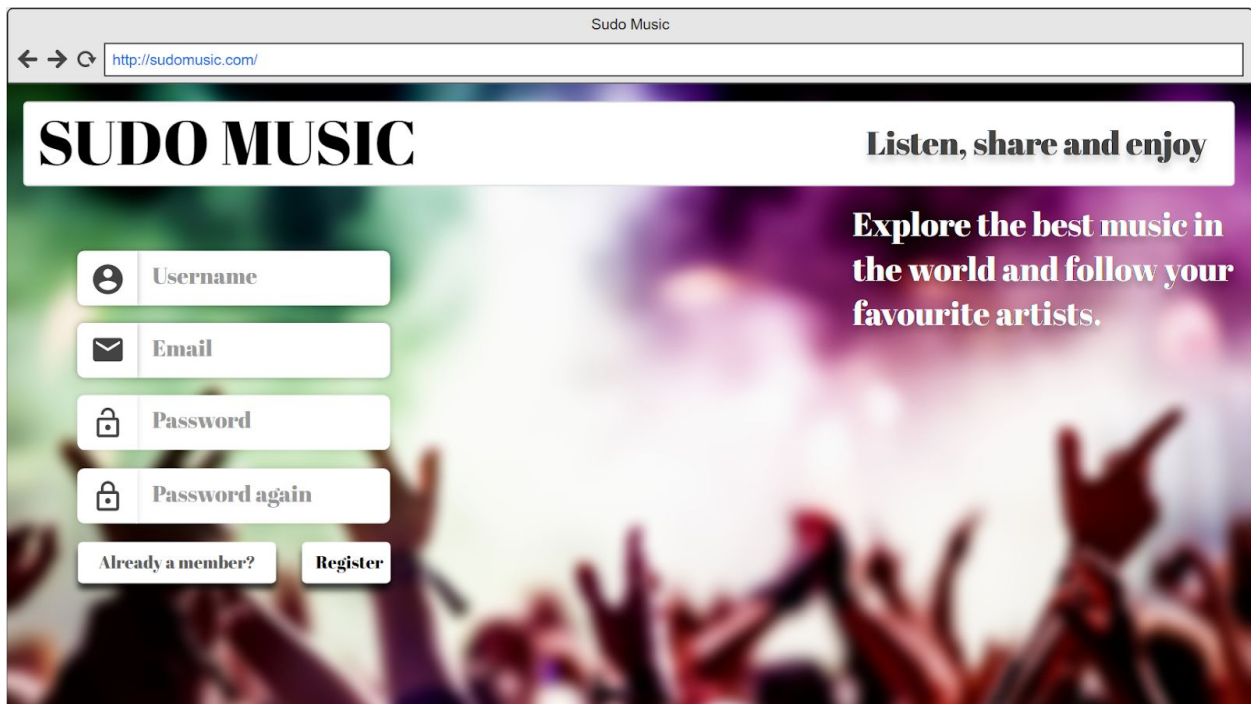
A User's data(password, email, etc.) will be retrieved with this query from the database. For Forgot Password scenario, the username will be used to get the other attributes of the user to remind the user.

Inputs:

@username: username of the current user

```
SELECT *  
FROM User  
WHERE @username = username;
```

### 3.b.2 Register Screen



The screenshot shows a web browser window with the address bar displaying "http://sudomusic.com/". The page title is "Sudo Music". The main heading is "SUDO MUSIC" in large, bold, black letters. To the right of the heading is the tagline "Listen, share and enjoy". Below the heading, there is a registration form with four input fields: "Username" (with a person icon), "Email" (with an envelope icon), "Password" (with a lock icon), and "Password again" (with a lock icon). Below these fields are two buttons: "Already a member?" and "Register". To the right of the form, there is a promotional text: "Explore the best music in the world and follow your favourite artists." The background of the page is a blurred image of a crowd of people at a concert, with many hands raised in the air.

#### 3.b.2.1 Register User

A new user will be added to the database with the query below:

Inputs:

@username: the username written to the textbox in register screen

@email: the email written to the textbox in register screen

@pwd: the password written to the textbox in register screen

```

insert into User(username, email, pwd)
select @username, @email, @pwd
where @username not in (select username from User) and
      @email not in (select email from User);

```

### 3.b.3 My Comments Screen

The screenshot shows the Sudo Music web application. The top navigation bar includes the Sudo Music logo and a balance of \$ 420.69. The left sidebar contains navigation links: My Playlists, Saved Tracks, Saved Albums, Following, Discover, and Settings. The main content area is titled 'Comments' and features a search bar and a 'Back to profile' button. Below this is a table of comments:

▼ On	▼ Comment	▼ Replies	▼ Added
Fire	One of the best pl...	0	1.1.2017
Dope	Quite enjoyed th...	44	2.1.2017
Classics	What is this?	6	4.1.2017
Golden	Delete this ple...	1	4.1.2017

The right sidebar shows 'Recent friend activity' with three entries: dorukeuk purchased Man's Not Hot, sarpisko is currently listening to All Star, and Iknodawae just purchased the Ugandan Shuffle. At the bottom, a music player is visible with a progress bar and a '4:20' duration.

#### 3.b.3.1 View Comments

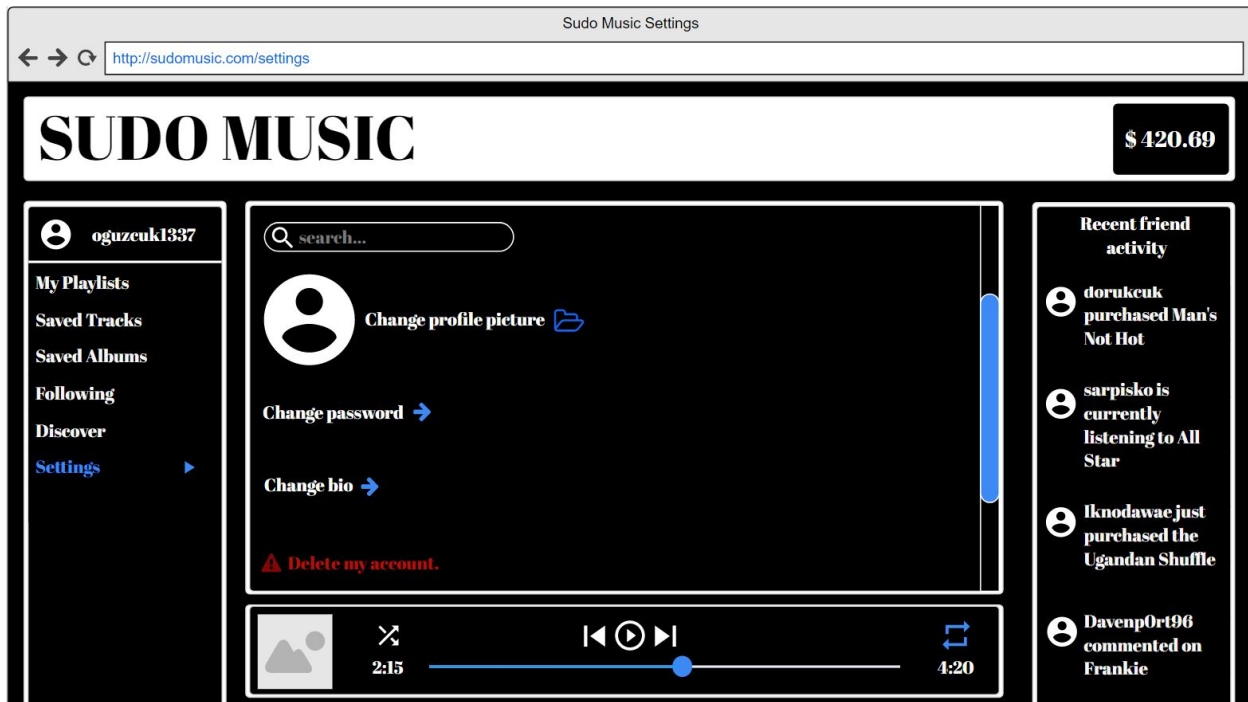
This query will be used for a scenario similar to the scenario named “View Your Comments”.

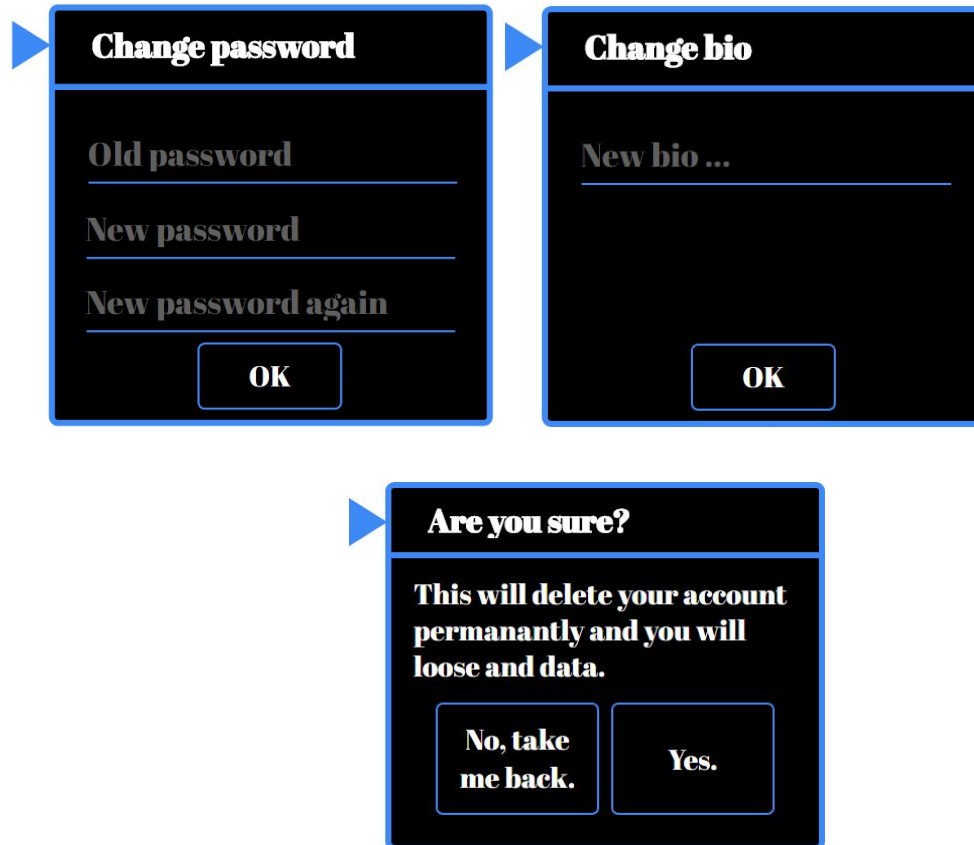
Inputs:

@username: the username of the current user

```
select C.comment_ID, C.comment_date, C.comment_content
from user_comment UC, _comment C
where @username = UC.username and
       C.comment_ID = UC.comment_ID;
```

### 3.b.4 Account(Settings) Screen





### 3.b.4.1 View Details

As mentioned above, the current user's data will be displayed according to the result of this query.

Inputs:

@username: username of the current user

```
SELECT *  
FROM User  
WHERE @username = username;
```

### 3.b.4.2 Change Password

When a user attempts to change the password, this query will be used to update the password of that user. Also the updates will be cascaded.

Inputs:

@username: username of the current user

```
UPDATE *  
FROM User  
SET pwd = @password  
WHERE @username = username;
```

#### 3.b.4.3 Add/Change Bio

When a user attempts to change his/her bio, this query will be used to update the bio of that user. As same as above, the updates will be cascaded to views.

Inputs:

@bio: new biography

@username: username of the current user

```
UPDATE User  
SET bio = @bio  
WHERE username = @username;
```

#### 3.b.4.4 Add/Change Profile Photo

The update request of the profile image of a user will be realized in database by this query.

Inputs:

@image: new profile image

@username: username of the current user

```
UPDATE User  
SET profile_image = @image  
WHERE username = @username;
```

#### 3.b.4.5 Delete Account

This query will erase the account details of current user from the database

Inputs:

@username: username of the current user



```
delete from User
where User.username = @username;
```

### 3.b.5 Credit Screen

Sudo Music My Balance

← → ↻ [http://sudomusic.com/my\\_balance](http://sudomusic.com/my_balance)

# SUDO MUSIC

\$ 420.69

oguzenk1337
 

- My Playlists
- Saved Tracks
- Saved Albums
- Following
- Discover
- Settings

## My Balance

Credit card number :

Exp. date :

CVC:

[View past transactions](#)

[Add credit](#)

## Select Amount

5 10 20 50

## Recent friend activity

- dorukcuk purchased Man's Not Hot
- sarpisko is currently listening to All Star
- lknodawae just purchased the Ugandan Shuffle
- Davenp0rt96 commented on Frankie

2:15

⏮ ⏪ ⏩ ⏭

4:20

Sudo Music Past Transactions

← → ↻ [http://sudomusic.com/past\\_transactions](http://sudomusic.com/past_transactions)

# SUDO MUSIC

\$ 420.69

oguzenk1337
 

- My Playlists
- Saved Tracks
- Saved Albums
- Following
- Discover
- Settings

## Past Transactions

▼ Amount	▼ Description	▼ Date
+\$ 5.0	Add balance	1.1.2017
+\$ 10.0	Add balance	2.1.2017
-\$ 1.99	Purchase: Kendrick Lamar	4.1.2017
-\$ 49.99	Purchase: Taylor Swift	4.1.2017

[Back to balance](#)

## Recent friend activity

- dorukcuk purchased Man's Not Hot
- sarpisko is currently listening to All Star
- lknodawae just purchased the Ugandan Shuffle
- Davenp0rt96 commented on Frankie

2:15

⏮ ⏪ ⏩ ⏭

4:20



### 3.b.5.1 View Credit

The credit of a user will be available to the front end of SUDO with the query below:

Inputs:

@username: the username of the current user

```
select U.credit
from User U
where U.username = @username;
```

### 3.b.5.2 Add Credit

The credit of a user will be updated when “Add Credit” scenario occurs. Also, to set the new credit, first the current credit will be requested from the server with the query presented in “View Credit” section. Then the new credit amount will be added by client. Then, the new balance will be updated in the database with the query below.

Inputs:

@username: username of the current user.

@credit: the new balance

```
update User
set credit = @credit
where username = @username;
```

### 3.b.6 Playlist Screen

← → ↺

http://sudomusic.com/oguzcuk1337

SUDO MUSIC

\$ 420.69

oguzcuk1337

My Playlists

Saved Tracks

Saved Albums

Following

Discover

Settings

search...

My Comments

oguzcuk1337

3.4k followers

Bio: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique,

Playlists

Following

☒ Chill mood

☒ House party theme

☒ Despacito

☒ Gangster Rap

snoophi

ngandanknuckles

lil\_boi

Post Malone

2:15

⏮ ⏪ ⏩ ⏭

4:20

Recent friend activity

dorukcuk purchased Man's Not Hot

sarpisko is currently listening to All Star

Iknodawae just purchased the Ugandan Shuffle

Davenp0rt96 commented on Frankie

← → ↺

http://sudomusic.com/oguzcuk1337/playlists

SUDO MUSIC

\$ 420.69

oguzcuk1337

My Playlists

Saved Tracks

Saved Albums

Following

Discover

Settings

search...

My Playlists

▼ Name	▼ Date	▼ Saves
Fire	1.1.2017	14
Dope	2.1.2017	16
Trax	4.1.2017	8
Bling	4.1.2017	2

2:15

⏮ ⏪ ⏩ ⏭

4:20

Recent friend activity

dorukcuk purchased Man's Not Hot

sarpisko is currently listening to All Star

Iknodawae just purchased the Ugandan Shuffle

Davenp0rt96 commented on Frankie

Create new playlist

Name

Collaborative

☐

OK

Sudo Music snoopboi My Awesome Playlist

← → 🔍

http://sudomusic.com/snoopboi/my\_awesome\_playlist

SUDO MUSIC

\$ 420.69

oguzcuk1337

My Playlists ▶

Saved Tracks

Saved Albums

Following

Discover

Settings

search...

prof\_dr\_dree:

Hey man this is a great playlist. I have been listening to it for hours!

snoopboi:

Thank you very much, I appreciate it a lot. Have fun!

Leave a comment

My Awesome Playlist

Created: 1.1.2017 by snoopboi

▼ Name

▼ Artist

▼ Album

▼ Added

DNA.

Kendrick Lamar

DAMN.

1.1.2017

44 More

Logic

Bobby Tarantino 2

2.1.2017

Bank Account

21 Savage

ISSA Album

4.1.2017

Fly Me to the Moon

Frank Sinatra

Best of Jazz Classics

4.1.2017

2:15

⏮ ⏪ ⏩ ⏭

4:20

Recent friend activity

dorukeuk purchased Man's Not Hot

sarpisko is currently listening to All Star

Iknotdawae just purchased the Ugandan Shuffle

Davenp0rt196 commented on Frankie

43

### 3.b.6.1 Comment On a Playlist

The query below will be used to add the new comment to the database. A comment consists of date, id and content. One of the uses of this query is as in scenario named “Commenting to Playlist”.

Inputs:

@date\_time: timestamp of the comment

@content: the comment content entered to textbox.

```
insert into _comment values(default, @date_time,  
@content);
```

### 3.b.6.4 Add Collaborators to a playlist



This query will be used when a user wants to authenticate some of his followers to a playlist. Authenticated users will have ability to add/remove music from that playlist.

Inputs:

@username: the username of the authenticated user

@playlistname: name of the playlist under consideration

```
insert into changes(username, playlist_id)
```

```
select @username, P.playlist_id
from playlist P
where P.playlist_name = @playlistname;
```

#### 3.b.6.4 Get Average Rating of a playlist

This query will be used to get the current average rating of the playlist under consideration.

Inputs:

@playlist name: the playlist under consideration

```
select P.avg_rating
from P
where P.playlist_name = @playlistname;
```

#### 3.b.6.5 Delete Comment From a Playlist

This query will be used to delete a comment from a playlist.

Inputs:

@comment\_ID: ID of the selected comment from which the comment will be deleted from.

```
DELETE FROM playlist_comment
WHERE comment_ID in (SELECT C.comment_ID
                     FROM _Comment C, playlist_comment PC
                     WHERE C.comment_ID = PC.comment_ID
                     AND C.comment_ID = @comment_ID);
```

#### 3.b.6.6 Get All Comments of a Playlist

This query will be used to return all the comments made to a playlist.

Inputs:

@playlist\_id: ID of the playlist whose comments will be returned.

```
SELECT C.comment_ID, C.comment_date, C.comment_content
FROM _Comment C, Playlist_Comment P
```

```
WHERE C.comment_ID = P.comment_ID AND @playlist_ID =
P.playlist_ID
GROUP BY C.comment_content;
```

### 3.b.6.7 Get All Replies to a Comment

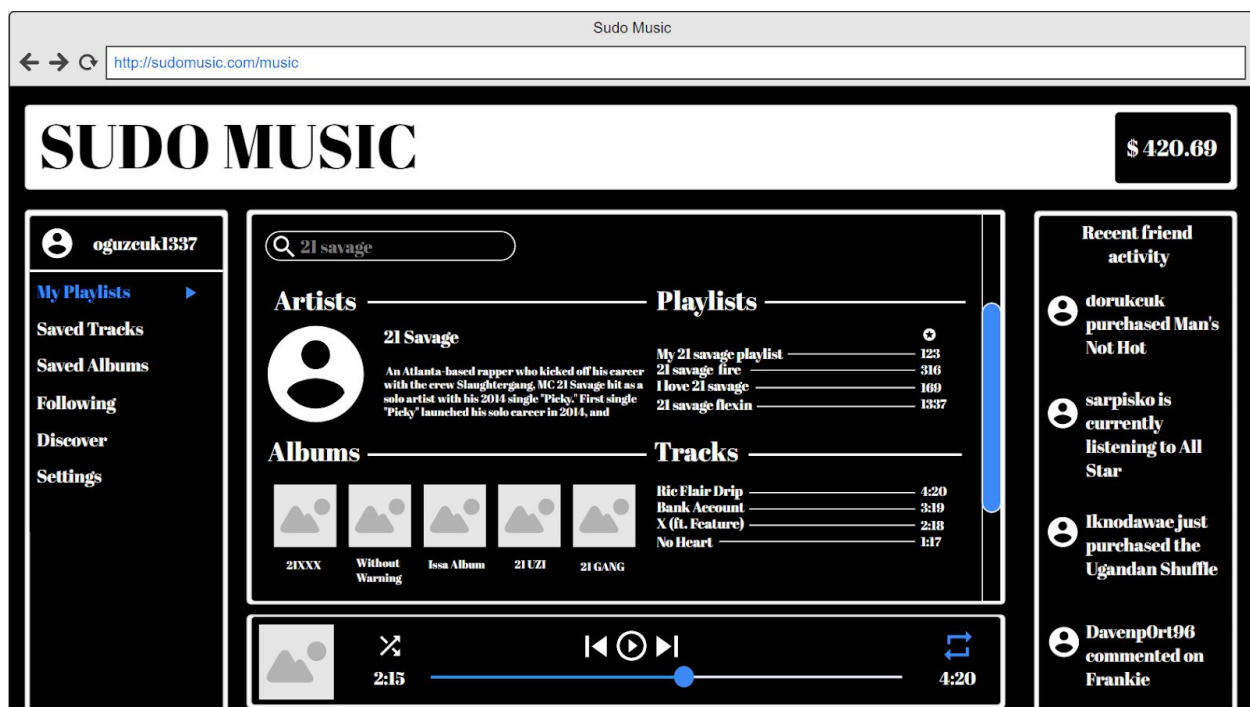
This query will be used to return all the replies made to a specific comment made to a playlist.

Inputs:

@comment\_ID: ID of the parent comment.

```
SELECT C1.comment_ID, C1.comment_date, C1.comment_content
FROM _Comment C1, Reply R1
WHERE C1.comment_ID = R1.id_child
      AND R1.id_parent in (SELECT R2.id_parent
                          FROM _Comment C2, Reply R2
                          WHERE C2.comment_ID =
@comment_ID AND C2.comment_ID = R2.id_parent);
```

### 3.b.7 Search Screen



rockstar by Post Malone

Already purchased!

Save track

View album

View artist

Add to playlist

Fire

Dope

Trax

Bling

Best

Golden

Classics (Jazz)

Stoney by Post Malone

Purchase for \$4.9

Save album

View album

View artist

Stoney by Post Malone

Purchase for \$4.9

Save album

View album

View artist

Insufficient funds. Please add more balance to complete this purchase.

rockstar by Post Malone

Already purchased!

Save track

View album

View artist

Remove

### 3.b.7.1 Search Artist

This search query returns the names of the artists starting with the given string as input to the search bar

Inputs:

@search: the string entered to the search bar

```
select AR.name
from Artist AR
where (temp like (@search || '%')) and
      (AR.name = temp);
```

### 3.b.7.2 Search Playlist

This search query returns the names of the playlists starting with the given string as input to the search bar

Inputs:

@search: the string entered to the search bar

```
select P.playlist_name
from Playlist P
where (temp like (@search || '%')) and
      (P.playlist_name = temp);
```

### 3.b.7.3 Search Track

This search query returns the names of the tracks starting with the given string as input to the search bar

Inputs:

@search: the string entered to the search bar

```
select T.track_name
from Track T
where (temp like (@search || '%')) and
```



```
(T.track_name = temp);
```

#### 3.b.7.4 Search Album

This search query returns the names of the tracks starting with the given string as input to the search bar

Inputs:

@search: the string entered to the search bar

```
select A.album_name
from Album A
where (temp like (@search || '%')) and
      (A.album_name = temp) ;
```

#### 3.b.7.5 Search User

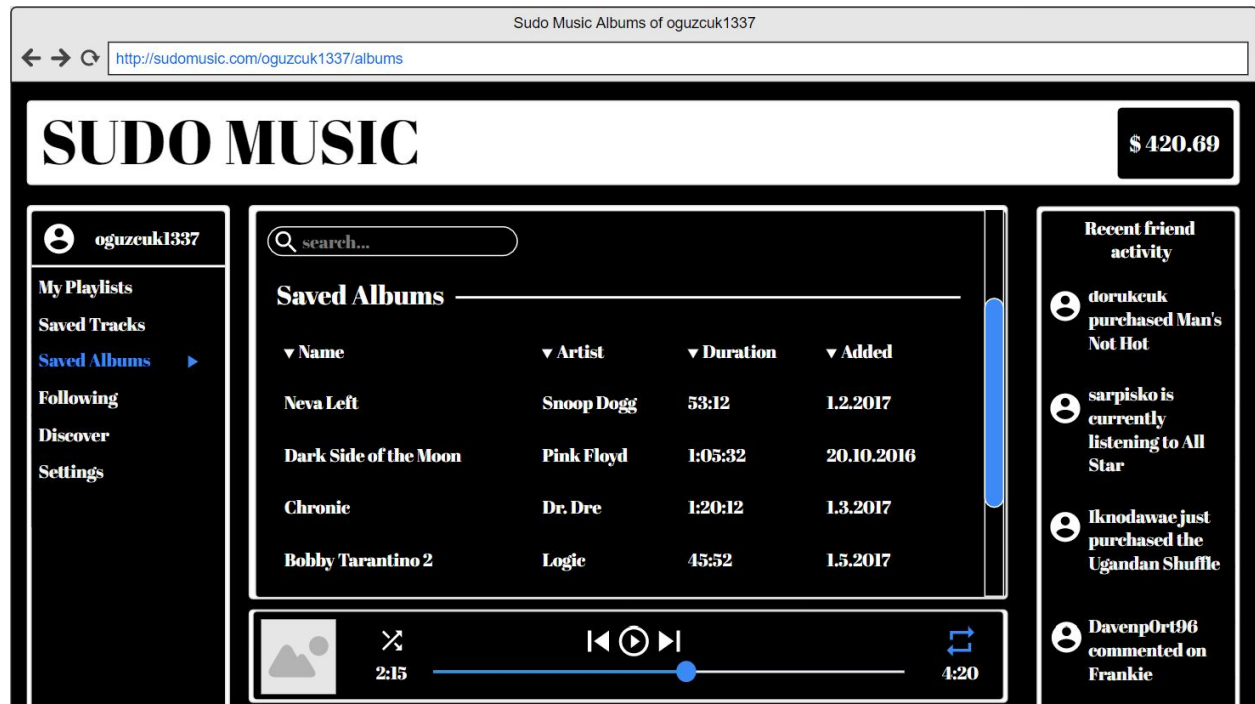
This search query returns the names of the users starting with the given string as input to the search bar

Inputs:

@search: the string entered to the search bar

```
select U.username
from User U
where (temp like (@search || '%')) and
      (U.username = temp);
```

### 3.b.8 Saved Albums Screen



#### 3.b.8.1 View Saved(Purchased) Albums

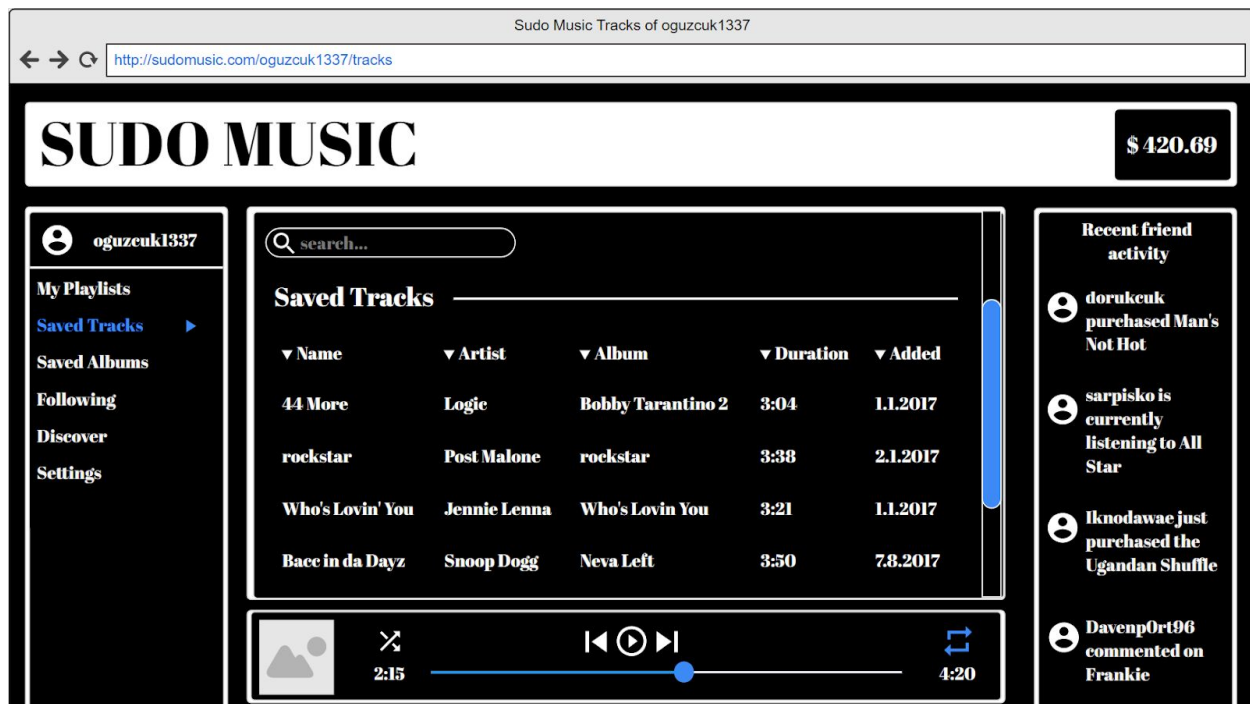
The query below returns saved albums of a user.

Inputs:

@username : username of the current user

```
SELECT A.album_name, A.ID, A.album_cost, A.album_duration
FROM Album A, Buy B
WHERE A.ID = B.ID and B.username = @username
GROUP BY A.album_name;
```

### 3.b.9 Saved Tracks Screen



#### 3.b.9.1 View Saved Tracks

The query below returns saved tracks of a user. The saved tracks will be obtained from the saved albums of a user.

Inputs:

@username : username of the current user

```
select T.track_name, T.ID, T.track_duration, T.genre,
T.track_cost, T.monthly_listener_count
from Track T, track_album TA
where T.ID = TA.track_ID and TA.album_ID in
(select A.ID
from Album A, Buy B
where A.ID = B.ID and B.username = @username
group by A.album_name );
```


### 3.b.10 User/Artist Info Screen

Sudo Music Post Malone

[http://sudomusic.com/post\\_malone](http://sudomusic.com/post_malone)

# SUDO MUSIC

\$ 420.69

 oguzcuk1337

My Playlists

Saved Tracks


Saved Albums

Following

Discover

Settings

search...



**Post Malone**

39m monthly listeners

5.2m followers

Following

Bio: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique.

Top Tracks

Releases

☒ Congratulations ft. Quavo

☒ Psycho

☒ rockstar

☒ I Fall Apart

Stoney

Stoney (Deluxe)


Candy Paint

Congratulations (remix)

\$ 1.0

\$ 11.0

\$ 11.0





2:15


⏮ ⏪ ⏩ ⏭


4:20

Recent friend activity

 dorukcuk purchased Man's Not Hot

 sarpisko is currently listening to All Star

 lknodawae just purchased the Ugandan Shuffle


 Davenp0rt96 commented on Frankie

Sudo Music Oguzcuk 1337

<http://sudomusic.com/oguzcuk1337>

# SUDO MUSIC

\$ 420.69

 oguzcuk1337

My Playlists

Saved Tracks


Saved Albums

Following

Discover

Settings

search...



**sarpisko**

550 followers

Following

Bio: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla quam velit, vulputate eu pharetra nec, mattis ac neque. Duis vulputate commodo lectus, ac blandit elit tincidunt id. Sed rhoncus, tortor sed eleifend tristique.

Playlists

Following

☒ Omae

☒ Wa

☒ Mou


☒ Shindeiru

oguzcuk1337

ugandanknuckles

Post Malone

Taylor Swift





2:15


⏮ ⏪ ⏩ ⏭


4:20

Recent friend activity

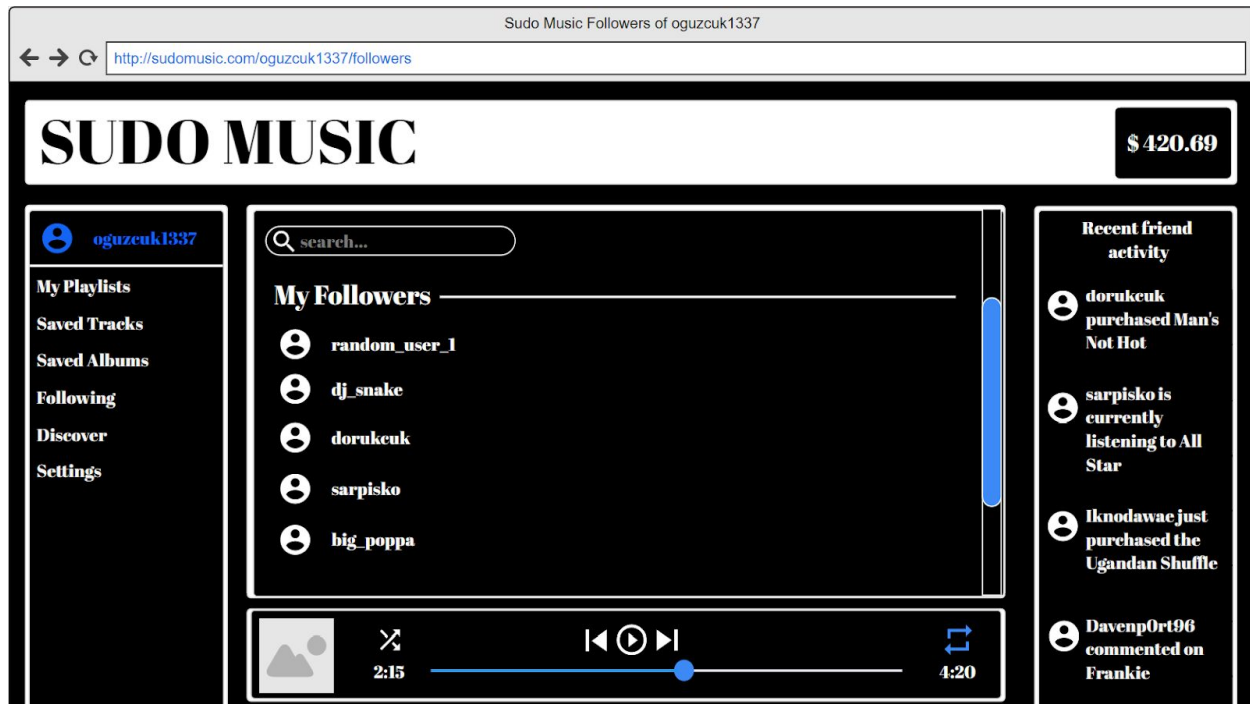
 dorukcuk purchased Man's Not Hot

 sarpisko is currently listening to All Star

 lknodawae just purchased the Ugandan Shuffle

 Davenp0rt96 commented on Frankie

52



### 3.b.10.1 Get Purchased Tracks of User

When clicked on the option, query will return the tracks that are purchased before and available to user.

Inputs:

@username: username of the user.

```
SELECT A.album_name, A.ID, A.album_cost, A.album_duration
FROM Album A, Buy B
WHERE A.ID = B.ID AND B.username = @username
GROUP BY A.album_name;
```

### 3.b.10.2 Get Albums of an Artist

When clicked on the option, query will return the albums of the artist in which user is hovering on right now.

Inputs:

@album\_ID : ID of an album.

```
SELECT T.track_name, T.ID, T.track_duration, T.genre,
T.monthly_listener_count
FROM Track T, track_album TA
```

```
WHERE T.ID= TA.trackID AND @album_ID = TA.album_ID
GROUP BY T.track_name;
```

## 3.c Advanced Database Components

### 3.c.1 Views

#### Past Transactions

Users can see all the transactions they have made up to that date by clicking the past transactions button in the balance screen.

```
CREATE VIEW past_transactions (amount , description ,
transaction_date) AS
SELECT
    T.transaction_amount,
    T.transaction_description,
    T.transaction_date
FROM
    Transaction T
    NATURAL JOIN
    User S
WHERE
    T.username = @username
```

#### Search Page

Users can search for keywords by writing it into the search bar. Artists, playlists, albums and individual tracks related to the searched keyword are displayed.

```
CREATE VIEW search_page (artist_name , artist_bio ,
artist_picture , related_playlists , related_albums ,
related_tracks) AS
SELECT
    A.artist_name,
    A.bio,
```

```

        U.profile_image,
        P.playlist_name,
        A.album_name,
        T.track_name
FROM
    Artist A
    JOIN
    Album Al ON A.username = Al.artist_username
    JOIN
    User U ON A.username = U.username
    JOIN
    Playlist P ON P.creator_username = A.username
    JOIN
    track_album TA ON TA.album_ID = Al.ID
    JOIN
    Track T ON T.ID = TA.track_ID
WHERE
    A.username = @username

```

## My Playlists

Users can see all of their playlists by clicking the “my playlists” button on any page. A list of playlists that are associated with the user (either collaborating with that playlist or created that playlist) are displayed.

```

CREATE VIEW my_playlists (playlist_name , date_added ,
no_of_saves) AS
SELECT
    P.playlist_name,
    P.creation_date,
    transaction_date,
    (SELECT
        COUNT(*)
    FROM
        save_playlist SP
        JOIN
        Playlist P2 ON SP.playlist_ID = P2.playlist_ID
    WHERE

```

```

        U.username = @username)
FROM
    User U
    JOIN
        Playlist P ON U.username = P.creator_username
WHERE
    U.username = @username

```

### 3.c.2 Triggers

- When a playlist is rated, it's avg\_rating will be recomputed and manipulated in playlist table.
- When a music is played by a user, it's monthly\_listener\_count will be incremented.
- When an album is inserted by artist, the Album(ID) and Artist
- When an album is bought, insert a tuple to Transaction where  
Transaction.transaction\_date = Buy.purchase\_date, transaction\_amount = album\_cost  
and transaction\_description will be generated as " @username bought @album\_name  
to \$@transaction\_amount " where:  
    @transaction\_amount: Transaction(transaction\_amount)  
    @username: username of the current user  
    @album\_name: Album(album\_name)

### 3.c.3 Constraints

The constraints given below are included in the create table statements contained in Section 2 of this report.

- Album
  - The cost of an album must be greater than 0
- Playlist
  - The avg\_rating attribute of a playlist must be greater than or equal to 0 and also lower than or equal to 10
- Track
  - The monthly\_listener\_count attribute of a track must be greater than or equal to 0
- rates
  - The rate\_amount attribute of rates table must be greater than or equal to 0, also rate amount must be lower than or equal to 10.
- User
  - The credit attribute of User must be greater than 0
  - The email attribute of a User tuple must contain '@'.



## **4. Implementation Plan**

For the implementation of backend and frontend we intend to use PHP and for the frontend and styling HTML, CSS and Javascript. We will also make use of the Bootstrap and JQuery library. To be able to handle data in the project we will use a MySQL server.

## **5. Website**

Our project's source code and anything else regarding to the project will be hosted on a public GitHub repository. The webpage (at [umutberkbilgic.github.io/Sudo-Music/](https://umutberkbilgic.github.io/Sudo-Music/)) is hosted by GitHub using the GitHub Pages service.