CS - 353
DATABASE SYSTEMS

Nemo Music
github.com/NemoMusic/Nemo/wiki

Design Report

Ali Bulut - 21402408 - Section 2
Kerem Ayöz - 21501569 - Section 1
Musab Erayman - 21401025 - Section 3
Ömer Faruk Karakaya - 21401431 - Section 3

02.04.2018

1. Table Of Content

2. Revised ER Model

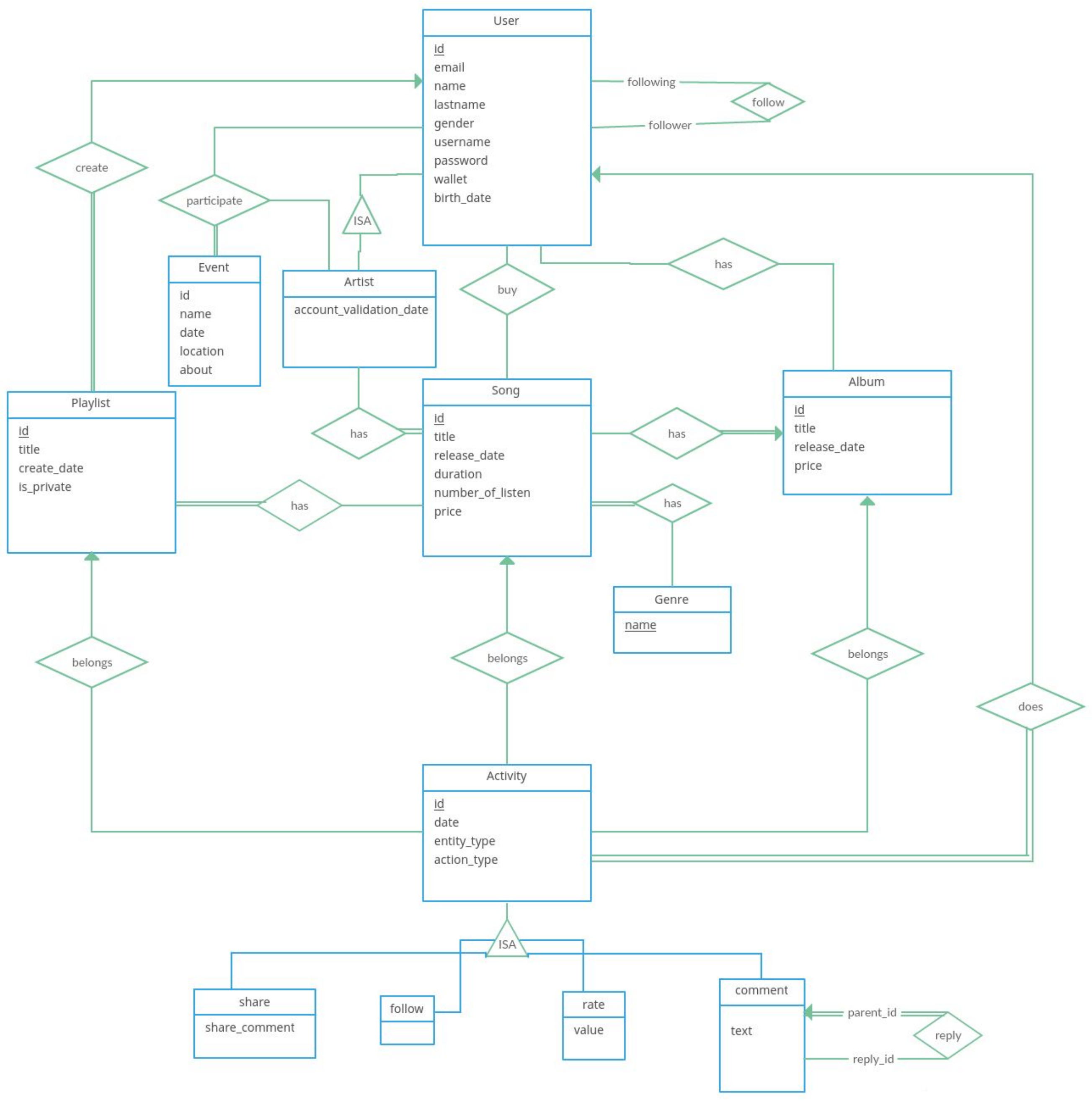


Figure 1. ER Diagram

3. Relational Schemas

3.1. User

Relational Model:

User (ID, email, name, last_name, gender, user_name, password, wallet, birth_date)

Functional Dependencies:

ID -> email name last_name gender user_name password wallet birth_date
user_name -> ID email name last_name gender password wallet birth_date
email -> ID name last_name gender user_name password wallet birth_date

Candidate Keys: {(ID),(user_name)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table user(
        ID              int not null primary key auto_increment,
        email           varchar(50) not null unique,
        name            varchar(20) not null,
        last_name       varchar(20) not null,
        gender          varchar(10),
        user_name       varchar(20) not null unique,
        password        varchar(20) not null,
        wallet          numeric(8,2),
        birth_date      date);
```

3.2. Event

Relational Model:

Event (ID, name, date, location, about)

Functional Dependencies:

ID - > name date location about

Candidate Keys: { (ID) }

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table event(
            ID              varchar(10) primary key not null
auto_increment,
            name            varchar(50) not null,
            date            date not null,
            location        varchar(50) not null,
            about           varchar(140));
```

3.3. Artist

Relational Model:

Artist (user_ID, account_validation_date)

Functional Dependencies:

userID -> account_validation_date

Candidate Keys:

{(user_ID)}

Normal Form:

Boyce-Codd Normal Form

Table Definition:

```
create table artist(
          user_ID                      int  primary key,
          account_validation_date      date,
          foreign key user_ID references User(ID));
```

3.4. Playlist

Relational Model:

Playlist(<u>ID</u>, title,create_date,is_private,User_ID)

Functional Dependencies:

ID -> title create_date is_private User_ID

Candidate Keys:

{(ID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table playlist(
            ID              varchar(10)  not null primary key
auto_increment,
            title           varchar(20) not null,
            create_date     date(20),
            is_private      varchar(10) not null,
            User_ID         varchar(20) not null,
            foreign key User_ID references User(ID));
```

3.5. Song

Relational Model:

Song(<u>ID</u>, title, release_date, duration, number_of_listen, price, album_ID)

Functional Dependencies:

ID -> title release_date duration number_of_listen price album_ID

Candidate Keys:

{(ID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table song(
        ID                  varchar(10)  primary key auto_increment,
        title               varchar(20) not null,
        release_date        date(20) not null,
        duration            time(5) not null,
        number_of_listen    int,
        price               int,
        album_ID            varchar(10) not null,
        foreign key album_ID references Album(ID));
```

3.6. Album

Relational Model:

Album(<u>ID</u>, title, release_date, price)

Functional Dependencies:

ID -> title release_date price

Candidate Keys:

({ID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table album(
            ID              varchar(10)  primary key auto_increment,
            title           varchar(20) not null,
            release_date    date(20),
            price           int not null);
```

3.7. Genre

Relational Model:

Genre(<u>name</u>)

Functional Dependencies:

No functional dependencies

Candidate Keys:

{(name)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table genre(
        name        varchar(10) not null primary key);
```

3.8. Activity

Relational Model:

Activity(<u>ID</u>,date,entity_type,action_type,Entity_ID,UserID)

Functional Dependencies:

ID ->date,entity_type,action_type,Entity_ID

Candidate Keys:

{(ID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table activity(
                ID              varchar(10) not null primary key
auto_increment,
                Date            date(10) not null,
                Entitiy_type    varchar(5) not null,
                Action_type     varchar(5) not null,
                User_ID         varchar(10) not null,
                foreign key User_ID references User(ID));
```

3.9. Share

Relational Model:

Share(activity_id, share_comment)

Functional Dependencies:

activity_id -> share_comment

Candidate Keys: activity_id

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table share(
        activity_id             varchar(10)  primary key not null,
        share_comment           varchar(50),
        foreign key activity_id references Activity(ID));
```

3.10. Follow

Relational Model:

Follow(activity_id)

Functional Dependencies:

No functional dependencies

Candidate Keys:

{(activity_id)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table follow(
        activity_id int not null primary key,
        foreign key activity_id references Activity(ID));
```

3.11. Rate

Relational Model:

Rate(<u>activity_id</u>,value)

Functional Dependencies:

activity_id -> value

Candidate Keys:

{(activitiy_id)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table rate(
            activity_id int not null primary key
            value int not null,
            foreign key activity_id references Album(ID));
```

3.12. Comment

Relational Model:

Comment(activity_id,text)

Functional Dependencies:

activity_id -> text

Candidate Keys:

{(activitiy_id)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table comment(
        activity_id int not null primary key
        text varchar(50) not null,
        foreign key activity_id references Album(ID));
```

3.13. User_follow

Relational Model:

User_follow(follower_id,following_id)

Functional Dependencies:

No functional dependency

Candidate Keys:

{(follower_id,following_id)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table user_follow(
        follower_id  int not null,
        following_id  int not null,
        foreign key follower_id references User(ID)
            on delete cascade on update cascade,
        foreign key following_id references User(ID)
            on delete cascade on update cascade,
        primary key (follower_id, following_id ));
```

3.14. User_song

Relational Model:

User_song(<u>user_id</u>,<u>song_id</u>)

Functional Dependencies:

No functional dependency.

Candidate Keys:

{(<u>user_id</u>,<u>song_id</u>)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table user_follow(
          user_id  int not null,
          song_id  int not null,
          foreign key user_id references User(ID)
               on delete cascade on update cascade,
          foreign key song_id references Song(ID)
               on delete cascade on update cascade,
          primary key (follower_id, following_id ));
```

3.15. User_album

Relational Model:

User_album(<u>user_id</u>,<u>album_id</u>)

Functional Dependencies:

No functional dependency.

Candidate Keys:

{(<u>user_id</u>,<u>album_id</u>)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table user_follow(
        user_id  int not null,
        album_id  int not null,
        foreign key user_id references User(ID)
            on delete cascade on update cascade,
        foreign key album_id references Album(ID)
            on delete cascade on update cascade,
        primary key (user_id, album_id ));
```

3.16. Artist_song

Relational Model:

Artist_song(<u>user_id</u>,<u>song_id</u>)

Functional Dependencies:

No functional dependency

Candidate Keys:

{(<u>user_id</u>,<u>song_id</u>)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table artist_follow(
        user_id  int not null,
        song_id  int not null,
        foreign key user_id references Artist(user_id)
            on delete cascade on update cascade,
        foreign key album_id references Album(ID)
            on delete cascade on update cascade,
        primary key (user_id, album_id ));
```

3.17. Participation

Relational Model:

Participation (<u>user_id</u>, <u>artist_id</u>, <u>event_id</u>)

Functional Dependencies:

No functional dependency.

Candidate Keys: { (user_id, artist_id, event_id) }

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table participation(
        user_id         varchar(10) not null,
        artist_id       varchar(10) not null,
        event_id        varchar(10),
        foreign key(user_id) references User(ID)
            on delete cascade on update cascade,
        foreign key(artist_id) references Artist(user_id)
            on delete cascade on update cascade,
        foreign key(event_id) references Event(ID)
            on delete cascade on update cascade);
```

3.18. Playlist_song

Relational Model:

Playlist_song (<u>song_id</u>, <u>playlist_id</u>)

Functional Dependencies:

No functional dependency

Candidate Keys: { (song_id, playlist_id) }

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table playlist_song(
        song_id             varchar(10) not null,
        playlist_id         varchar(10) not null,
        foreign key(song_id) references Song(ID)
                on delete cascade on update cascade,
        foreign key(playlist_id) references Playlist(ID)
                on delete cascade on update cascade);
```

3.19. Comment_reply

Relational Model:

Comment_reply (<u>reply_id</u>, parent_id)

Functional Dependencies:

reply_id -> parent_id

Candidate Keys: { (reply_id) }

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table comment_reply(
                reply_id                varchar(10) not null,
                parent_id               varchar(10) not null,
                foreign key(parent_id) references Comment(activity_id)
                        on delete cascade on update cascade,);
```

3.20. Song_genre

Relational Model:

Song_genre (genre_name, song_id)

Functional Dependencies:

No dependencies

Candidate Keys: {(song_id, genre_name)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
create table song_genre(
        song_id              varchar(10) not null,
        genre_name           varchar(10) not null,
        foreign key(song_id) references Song(ID)
            on delete cascade on update cascade,
        foreign key(genre_name) references Genre(name)
            on delete cascade on update cascade);
```

4. Functional Components

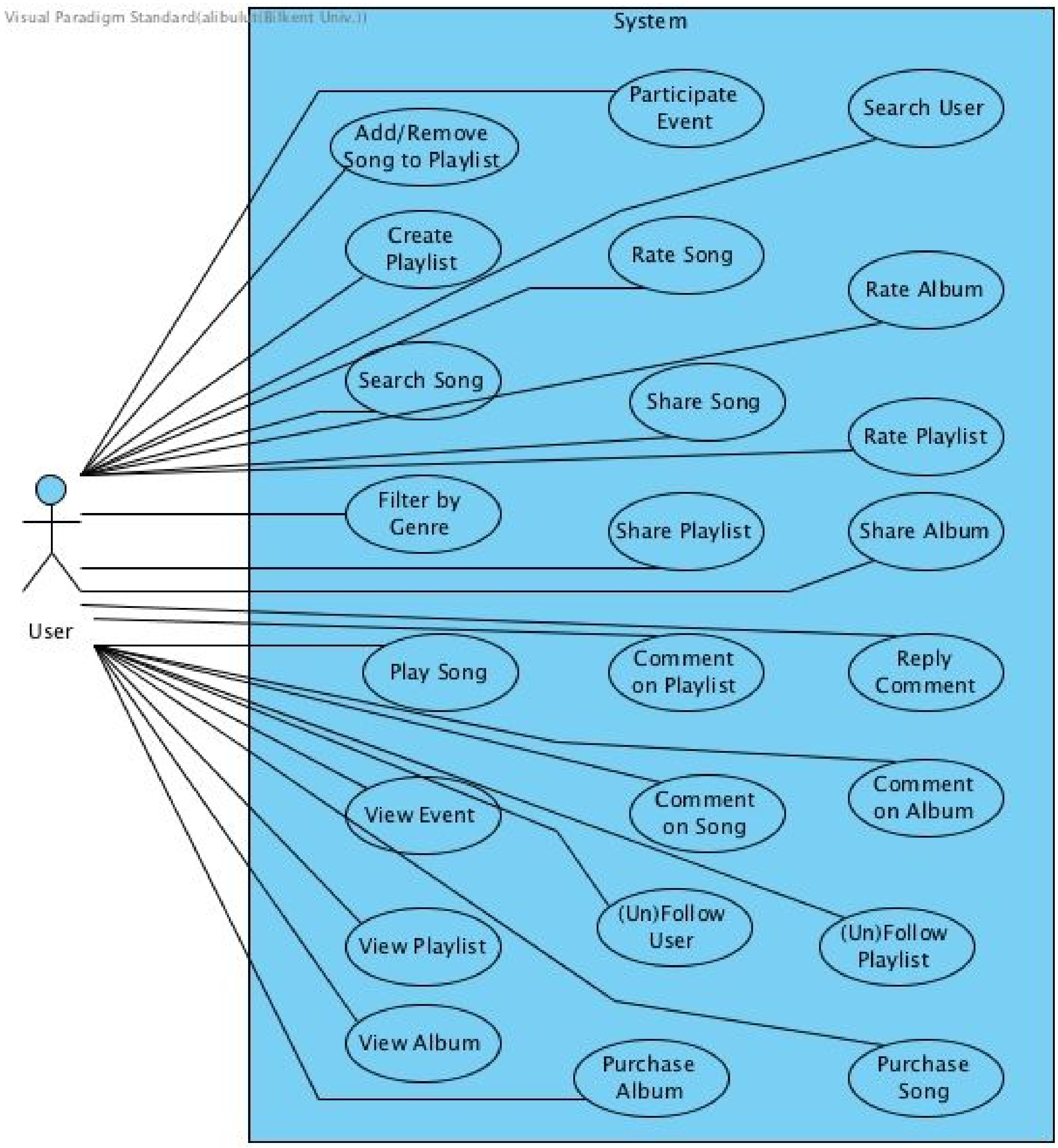


Figure 2. Use Case Diagram

User:

- Every user can create playlist of his/her owned songs and give it a name. Users cannot create a playlist from another user's song library but do it from only his/her song library. Users need to decide on the creating stage whether the playlist will be public or private. Public playlists will be open everyone to be seen, followed, commented on it, shared and listened if the songs are already is owned by other users. Private playlists will be open to only the owner.

- User can add purchased songs to his/her playlists or remove from them.
- User can search a song among database. To do that he/she should give song name or artist name to do system. .
- User can filter songs by their genre. Then the system will return the songs with this genre.
- User can search a user among database. To do that user should give username or user’s name and lastname together.
- Users could listen the musics that they bought from market. They could add these songs their playlists also. Additionally, they could listen song previews before buying it, which gives an idea about the music.
- User can declare that s/he will attend the event or not.
- User can view an upcoming event. Each user can see who will be attending the event and which artists will be participating. Each user can learn details about the event, where it will be placed, when it will be happening. Even more details given can be seen about the event if it is provided such as price, organizators, quota etc.
- User can view any public playlist and his/her own private playlist.
- User can view an album content that includes album name, artist(s), each song name, duration and price.
- In Nemo music system users will be able to rate songs that they own from 1 to 5. They can also change their previous rate. Average rating that some song get will be displayed by all users.When a user rate some song all users that following that user will be notified through their timeline.
- In Nemo music system users will be able to share songs that they own with their followers. All users who follows that users will be notified through their timeline. While a user sharing some son they will be able to post comment with it.

- Users of Nemo will be able to follow public playlist that have been created by other users. When users follow some playlist, that playlist will be shown among users playlist but user won't be able to change followed but not owned playlists.
- Every user will be able to comment on a playlist and if the playlist is publicly opened everyone, anyone can see the comment. If the playlist is private, only owner of the playlist can see the comment.
- In Nemo music system users will be able to comment on the songs that they own. They can also change or delete their previous comments. All comments about a song will be able to displayed by other users. When user make some comment all users that following that user will be notified through their timeline.
- Nemo allows users to interact with other people very easy. Users could find their friends and follow in the Nemo platform. They could see their friends' activities, popular musics, playlists etc. Therefore following other users is a really convenient feature..
- In Nemo Music system users will be able to unfollow already followed users, playlists and artists.
- Each user in Nemo will be able to also buy an album from the market system of Nemo. Any user can purchase any album as the user can afford to buy the album. When the user buys the album, songs of this album will be added to user's song library and also the album itself will be added to the user's owned album part. The user will be able to listen to any song of the album any time when s/he buys it.
- Users of Nemo music system are able to rate albums from 0 to 5. Ratings of each album will be displayed at album page. Users can change their rate whenever they want.
- Users will be able to rate a playlist like comment. However the difference is rating a playlist is giving a integer point from 0 to 5. Users will be able to rate playlist which they own if the playlist is private. Public playlist are open to everyone to be rated.
- In Nemo Music system users are able to share albums with their followers. When they share album followers of users will be noticed through their timeline.

- In Nemo music system users are able to make comments on albums. All comments will be displayed at Album’s page. When users make comment his/her followers will be noticed through their timeline.
- In Nemo music system users will be able to reply comments that made to songs and playlists. Users can only reply comments about a song that they own. When user reply some comment this reply will be displayed with original comment. When user reply some comment, all users that following that user will be notified through their timeline. Also the owner and followers of the owner of the comment which has been replayed by other comment will be notified through their timeline their timeline.
- Each user in Nemo will be able to buy music from the market system of Nemo. Any user can purchase any single music as a user can afford to buy it. When the user buys the single song, this song will be added to user's song library and a user can listen to this song anytime.

5. User Interface Design and Corresponding SQL Statements

5.1. Log-in/Sign-up View

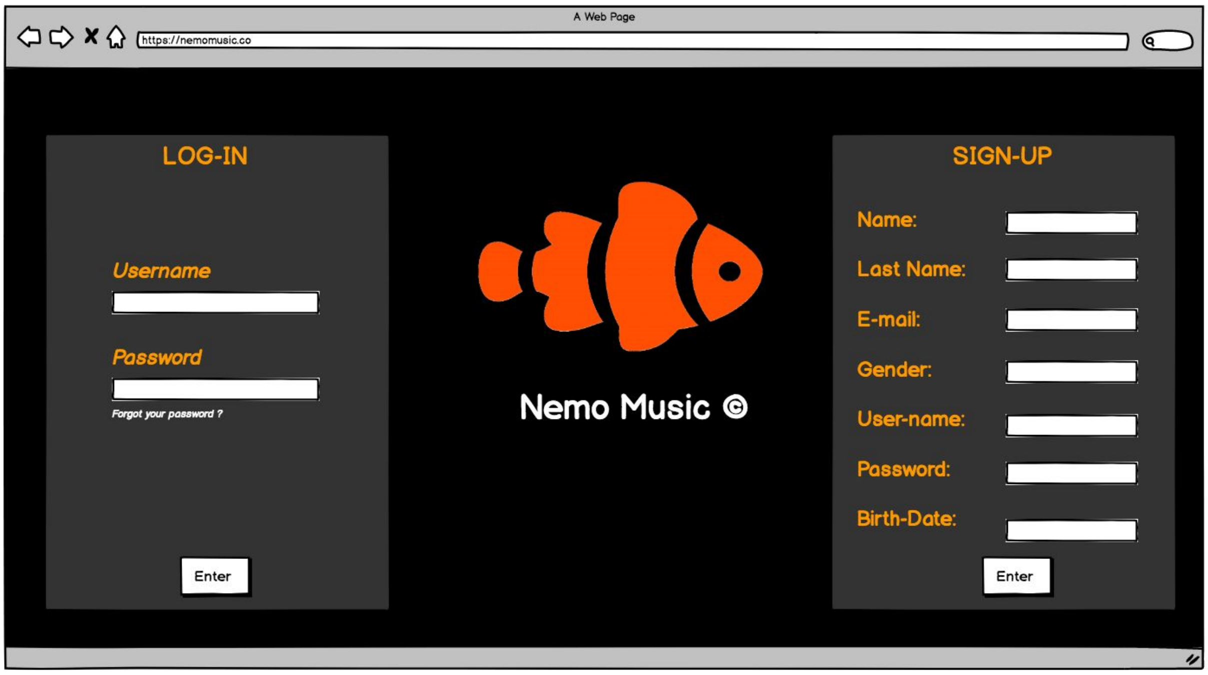


Figure 4. Log-in/Sign-up View

Inputs:

@Username @Password @Name @LastName @Email @Gender @UserName @BirthDate

Process:

When log-in, sign-up screen is opened, login and signup panels are shown to user. If user has an account already, s/he can log in by her/his username and password else user can sign up by giving name, last name, email, gender, username, password, birthdate. Username and e-mail must be unique.

SQL Statements:

Checking If User Exist To Logged User In:

SELECT U.ID, U.Username
FROM User U
WHERE U.username = @Username **and** U.password = @Password

Adding New User To Database:

INSERT into User (<u>ID</u>, email, name, last_name, gender, user_name, password, wallet, birth_date)
VALUE(ID, @Email, @Name, @LastName, @Gender, @Username, @Password, 0, @BirthDate)

5.2. Accounts Settings View

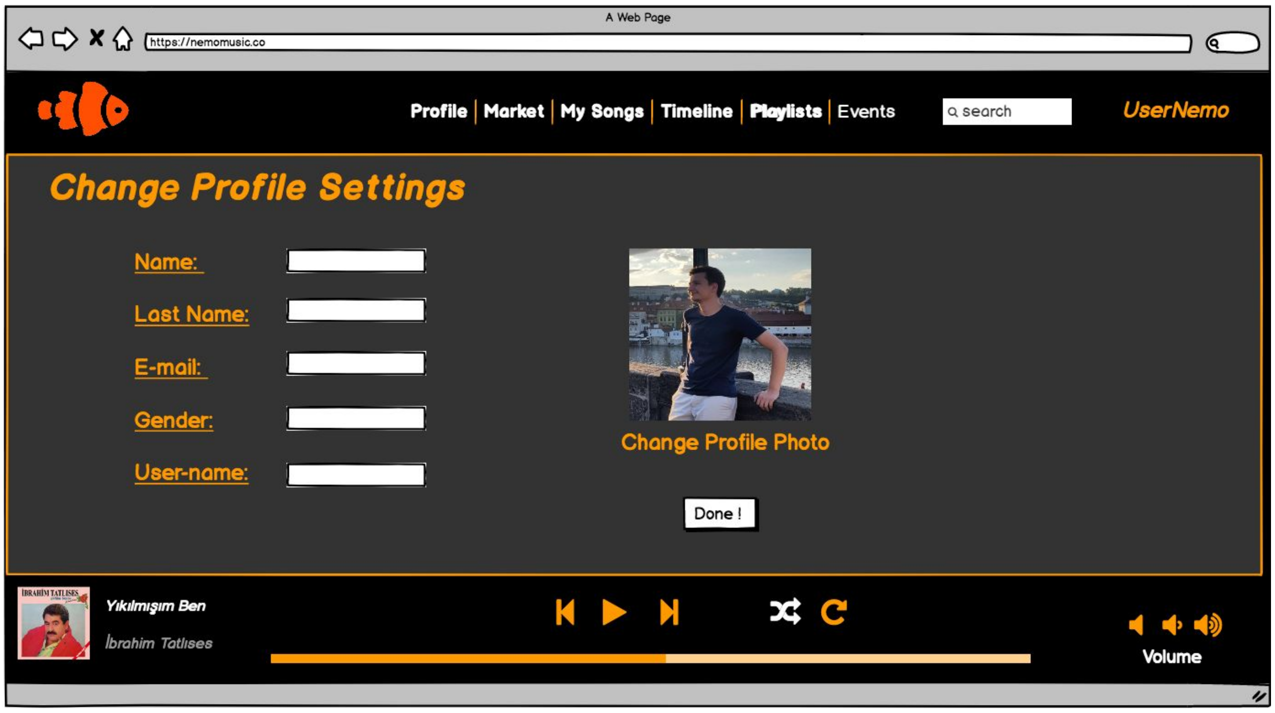


Figure 5. Account Settings View

Inputs:

@OnlineUserID @Name @LastName @Email @Gender @UserName

Process:

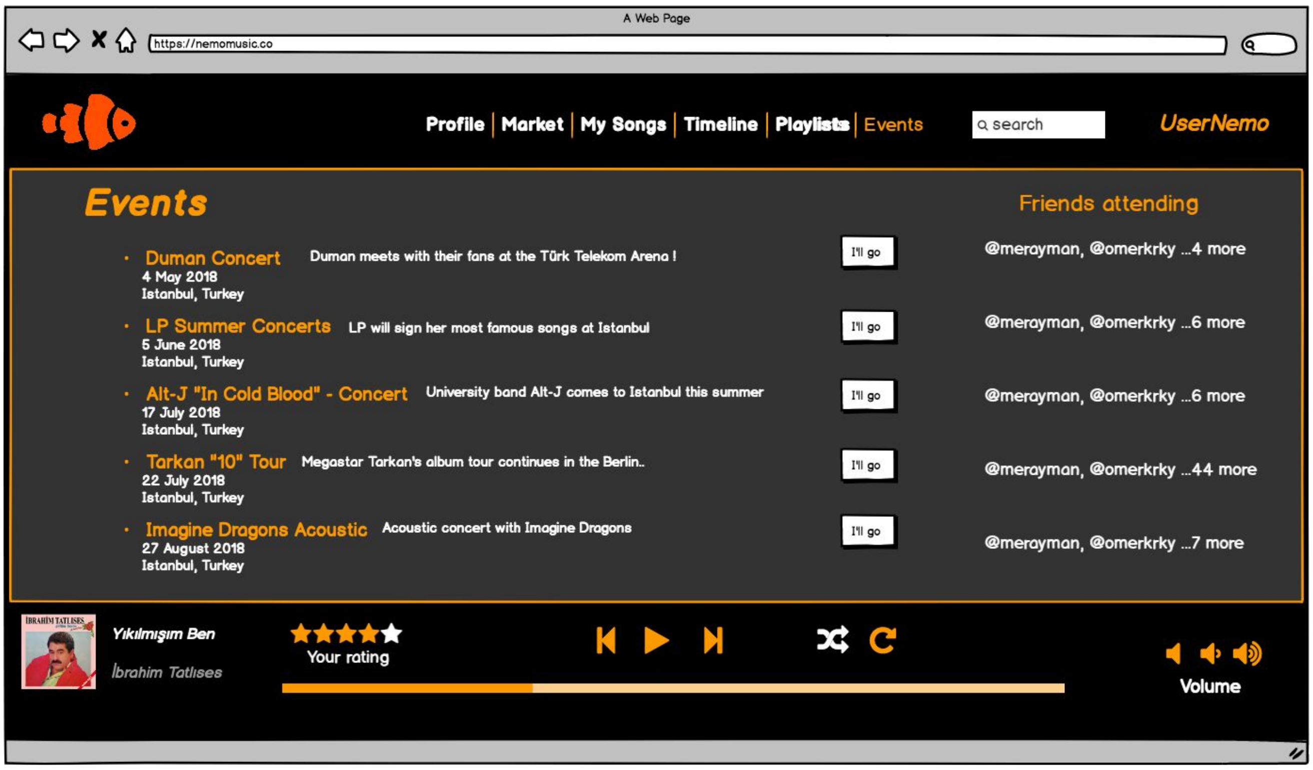
When the Account Settings screen is opened user can change her/his name, last name, email, gender, and username. The new e-mail and username must not be in the user table already(must be unique).

SQL Statements:

Changing Profile Settings:

UPDATE User U
SET U.name = @Name, U.email = @Email, U.last_name = @LastName, U.username = @Username)

5.3. Event View



Events

Friends attending

- Duman Concert Duman meets with their fans at the Türk Telekom Arena !
  4 May 2018
  Istanbul, Turkey
  I'll go — @merayman, @omerkrky ...4 more
- LP Summer Concerts LP will sign her most famous songs at Istanbul
  5 June 2018
  Istanbul, Turkey
  I'll go — @merayman, @omerkrky ...6 more
- Alt-J "In Cold Blood" - Concert University band Alt-J comes to Istanbul this summer
  17 July 2018
  Istanbul, Turkey
  I'll go — @merayman, @omerkrky ...6 more
- Tarkan "10" Tour Megastar Tarkan's album tour continues in the Berlin..
  22 July 2018
  Istanbul, Turkey
  I'll go — @merayman, @omerkrky ...44 more
- Imagine Dragons Acoustic Acoustic concert with Imagine Dragons
  27 August 2018
  Istanbul, Turkey
  I'll go — @merayman, @omerkrky ...7 more

Figure 6. Event View

Inputs:

@OnlineUserID @SelectedEventID

Process:

When event screen is opened, user can see upcoming events. User can say that s/he will participate the event. Also the number of current participants will be shown on the screen.

SQL Statements:

Listing All Events:

```
SELECT *
FROM Event
WHERE E.date > CURDATE()
ORDER BY date desc
```

Participating an Event:

```
INSERT INTO Participation (user_id, artist_id, event_id)
        SELECT OnlineUserID, P.artist_id, SelectedEventID
        FROM Participation P
        WHERE P.event_id = SelectedEventID
```

Listing Participated Friends of Each Event:

```
SELECT P.ID, U.username
FROM User U, Participate P
WHERE U.ID = P.user_id
GROUP BY P.ID
```

5.4. Other User Profile View

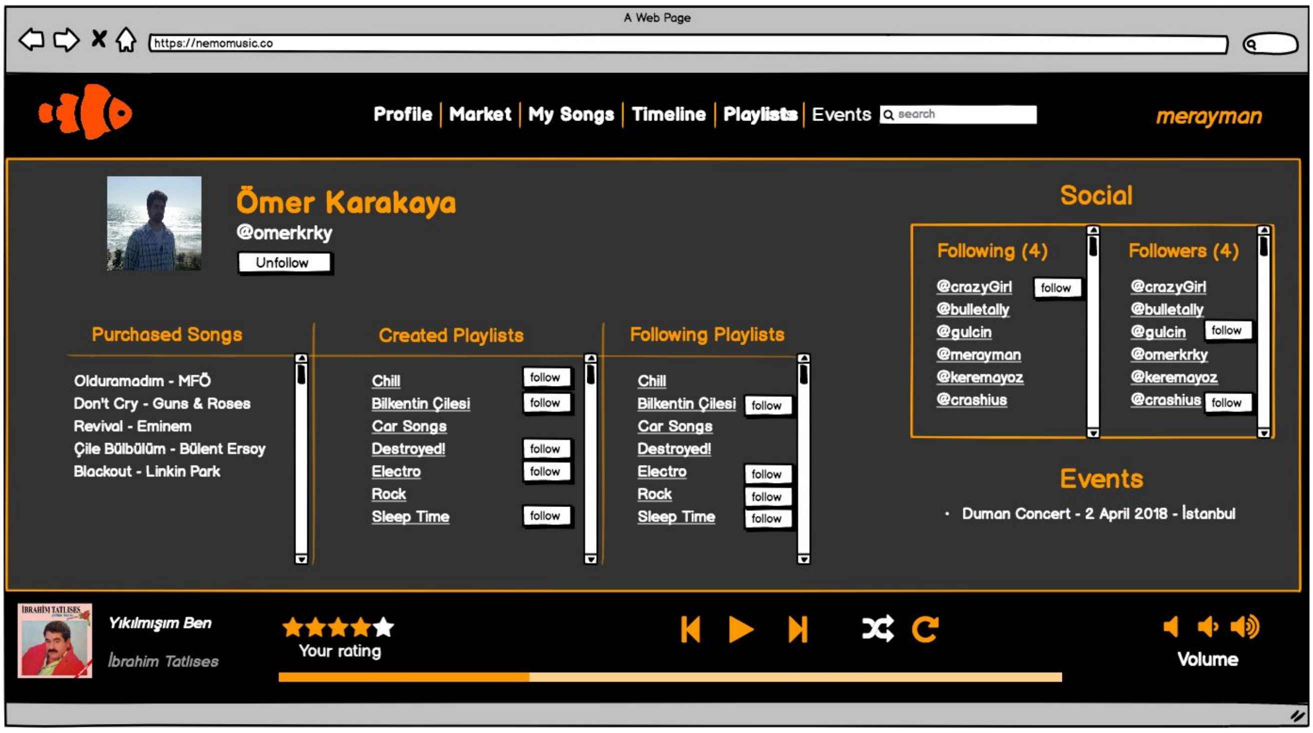


Figure 7. Other User Profile View

Inputs:

@OnlineUserID @SelectedPlaylistID @date @SelectedUserID

Process:

When user display profile page of another user purchased songs, created playlist, followed playlists, participated events, followers and followed users are displayed. When following, followed users and playlist of displayed users are shown follow buttons are displayed if User not followed referencing content yet.

SQL Statements:

Listing Purchased Songs of Specific User:

SELECT S.title, S.ID
FROM Song.S, User_song US
WHERE S.ID = US.user_id **and** S.ID = SelectedUserID);

Listing Created Playlists of Specific User:

SELECT P.title, P.ID
FROM Playlist P, User_playlist UP
WHERE P.ID = UP.user_id **and** UP.user_id = SelectedUserID **and** P.is_private = false) ;

Listing Followed Playlists of Specific User:

```
SELECT P.title, P.ID
FROM  Playlist P
WHERE P.is_private = false and P.ID = any ( SELECT F.entity_id
                                            FROM Activity A , Follow F
                                            WHERE A.entity_id = P.ID and A.ID = F.activity_id and
                                            A.User_id = SelectedUserID);
```

Listing Participated Events of Specific User:

SELECT E.ID, E.about
FROM Event E, Participation P
WHERE E.id = Participation.event_id **and** P.user_id = SelectedUserID;

Listing Followed Users of Specific User:

SELECT U.username, U.ID
FROM User U,User_follow UF
WHERE U.ID = UF.following_id **and** U.ID = SelectedUserID;

Listing Users That Follows Specific User:

SELECT U.username, U.ID
FROM User U,User_follow UF
WHERE U.ID = UF.follower_id **and** UF.follower_id = SelectedUserID;

Following Specific User:

INSERT INTO User_follow(follower_id,following_id)
VALUES (OnlineUserID, SelectedUserID);

5.5. Create Playlist View

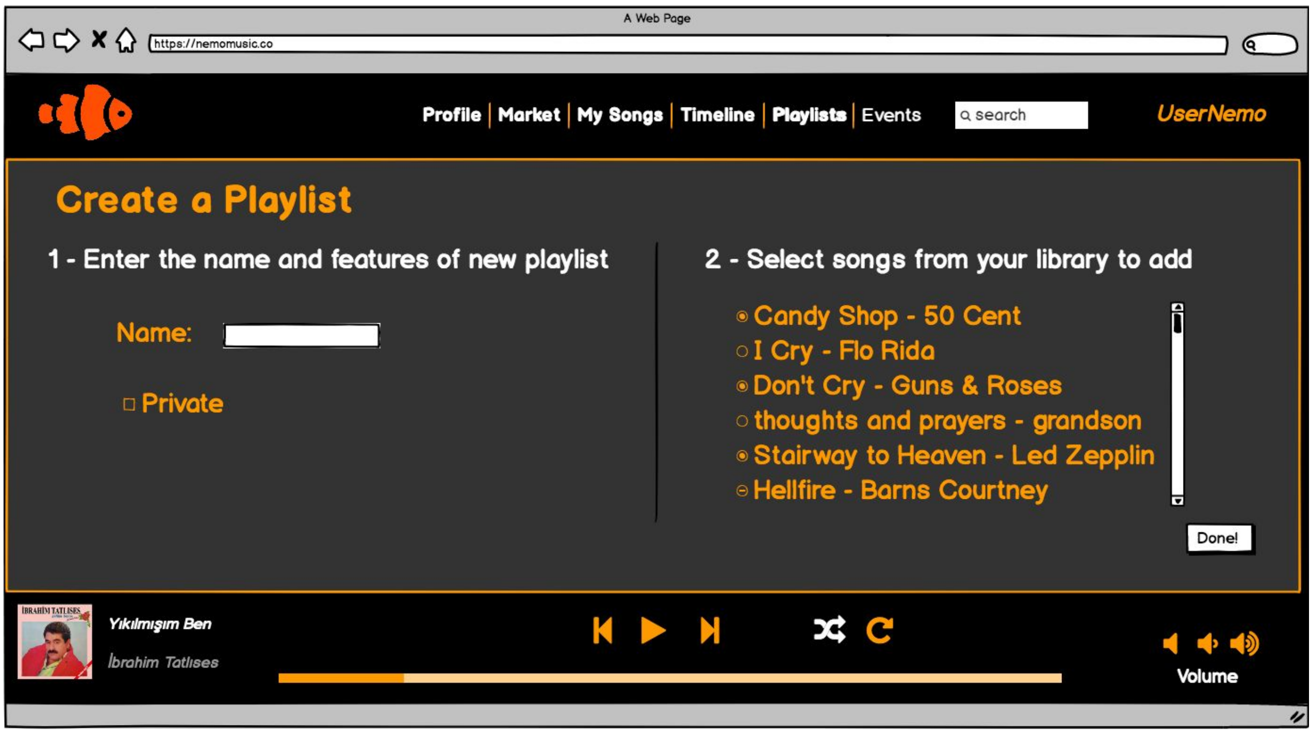

A Web Page
https://nemomusic.co
Profile | Market | My Songs | Timeline | Playlists | Events
search
UserNemo
Create a Playlist
1 - Enter the name and features of new playlist
Name:
Private
2 - Select songs from your library to add
Candy Shop - 50 Cent
I Cry - Flo Rida
Don't Cry - Guns & Roses
thoughts and prayers - grandson
Stairway to Heaven - Led Zepplin
Hellfire - Barns Courtney
Done!
Yıkılmışım Ben
İbrahim Tatlıses
Volume

Figure 8. Create Playlist View

Inputs:

@OnlineUserID @SelectedSongID @date @PlaylistName @isPrivate @createdPlaylistID

Process:

User can create a new playlist by adding song that they have by selecting those songs one by one. User gives name to playlist and selects its is privacy property.

SQL Statements:

Listing All Songs That User Has:

SELECT S.ID, S.title
FROM Songs S, User_song US
WHERE S.ID = US.User_id

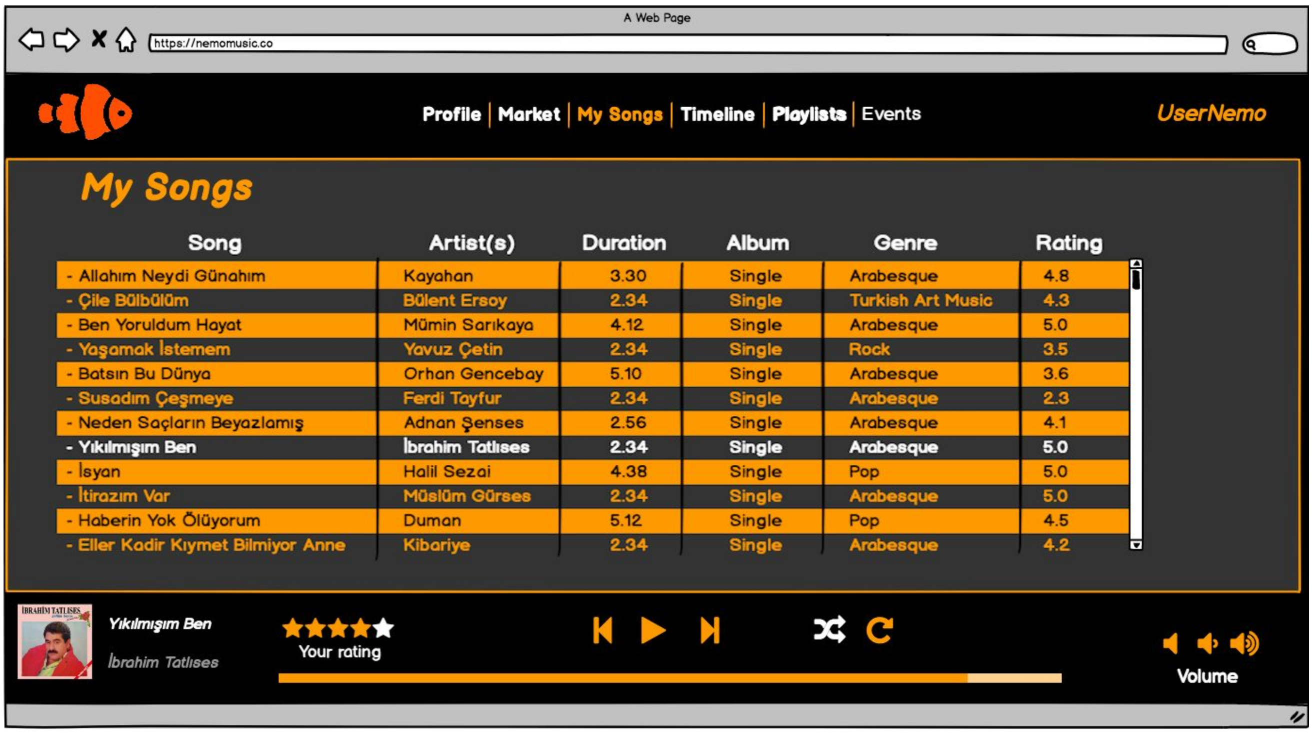
Create New Playlist:

INSERT INTO Playlist(ID, title,create_date,is_private,User_ID)
SET @last_id_in_table = LAST_INSERT_ID();
VALUES (ID, date,isPrivate,OnlineUserID);

Add Song to Playlist:

INSERT Playlist_song
VALUES (SelectedSongID,@last_id_in_table);

5.6. My Songs View



Song	Artist(s)	Duration	Album	Genre	Rating
- Allahım Neydi Günahım	Kayahan	3.30	Single	Arabesque	4.8
- Çile Bülbülüm	Bülent Ersoy	2.34	Single	Turkish Art Music	4.3
- Ben Yoruldum Hayat	Mümin Sarıkaya	4.12	Single	Arabesque	5.0
- Yaşamak İstemem	Yavuz Çetin	2.34	Single	Rock	3.5
- Batsın Bu Dünya	Orhan Gencebay	5.10	Single	Arabesque	3.6
- Susadım Çeşmeye	Ferdi Tayfur	2.34	Single	Arabesque	2.3
- Neden Saçların Beyazlamış	Adnan Şenses	2.56	Single	Arabesque	4.1
- Yıkılmışım Ben	İbrahim Tatlıses	2.34	Single	Arabesque	5.0
- İsyan	Halil Sezai	4.38	Single	Pop	5.0
- İtirazım Var	Müslüm Gürses	2.34	Single	Arabesque	5.0
- Haberin Yok Ölüyorum	Duman	5.12	Single	Pop	4.5
- Eller Kadir Kıymet Bilmiyor Anne	Kibariye	2.34	Single	Arabesque	4.2

Figure 9. My Songs View

Inputs:

@OnlineUserID @SelectedSongID

Process:

Users are able to display all songs that they have. All songs of the user are shown in this screen with songs' title, artist name, duration, belonging album, genre, and rating.

SQL Statements:

Listing All Songs :

SELECT *
FROM User_song US, Song S
WHERE US.user_id = OnlineUserID

5.7. Own Profile View

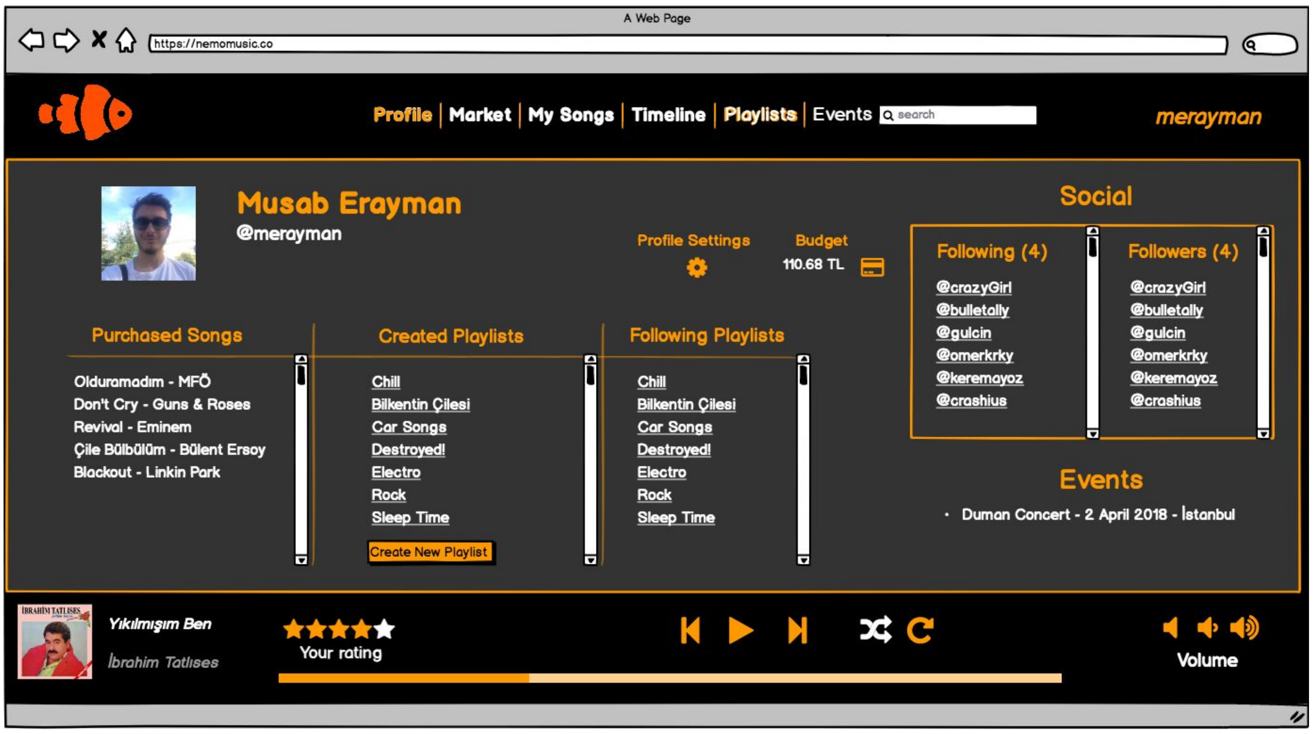


Figure 10. Own Profile View

Inputs:

@OnlineUserID @SelectedSongID @seacrhKey

Process:

When a user opens profile view information about purchased songs, created playlists, following playlists, followers, following users, wallet balance and participating events will be shown. Also users can access their profile settings on this view.

SQL Statements:

Listing Purchased Songs of Specific User:

```
SELECT S.title, S.ID
FROM Song.S, User_song US
WHERE S.ID = US.user_id and S.ID = OnlineUserID);
```

Listing Created Playlists of Specific User:

```
SELECT P.title, P.ID
FROM Playlist P, User_playlist UP
WHERE P.ID = UP.user_id and UP.user_id = OnlineUserID);
```

Listing Followed Playlists of Specific User:

```
SELECT P.title, P.ID
FROM  Playlist P
WHERE P.ID = any ( SELECT F.entity_id
                   FROM Activity A , Follow F
                   WHERE A.entity_id = P.ID and A.ID = F.activity_id and A.User_id =
OnlineUserID);
```

Listing Participated Events of Specific User:

```
SELECT E.ID, E.about
FROM Event E, Participation P
WHERE E.id = Participation.event_id and P.user_id = OnlineUserID
```

Listing Followed Users of Specific User:

```
SELECT U.username, U.ID
FROM User U,User_follow UF
WHERE U.ID = UF.following_id and U.ID = OnlineUserID;
```

Listing Users That Follows Specific User:

```
SELECT U.username, U.ID
FROM User U,User_follow UF
WHERE U.ID = UF.follower_id and UF.follower_id = OnlineUserID;
```

5.8. Market View

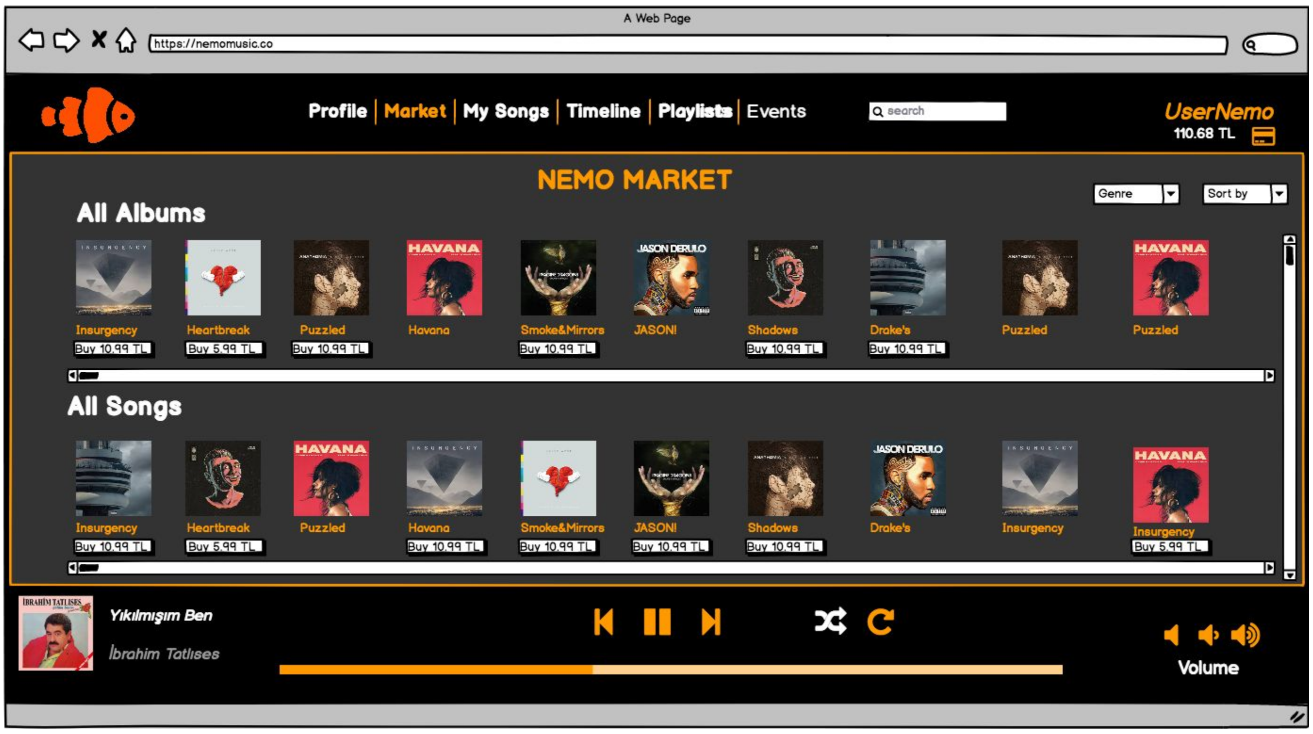


Figure 11. Market View

Inputs:

@OnlineUserID @SelectedSongID @selectedGenreName

Process:

When a user opens market view, all albums and all song will be shown and their default order will be based on their popularity. If the user has not album or song that is listed, there will be buy button for purchasing that song or album. Also, the current credits in the wallet will be shown on the right top of the screen.

SQL Statements:

Listing All Albums :

SELECT *
FROM Album A

Listing All Songs:

SELECT *
FROM Song S

Getting Specific Song by ID:

```
SELECT *
FROM Song
WHERE ID = SelectedSongID;
```

Listing Songs by Genre Filter:

```
SELECT song_id
FROM Song_genre sg
WHERE sg.genre_name = selectedGenreName
```

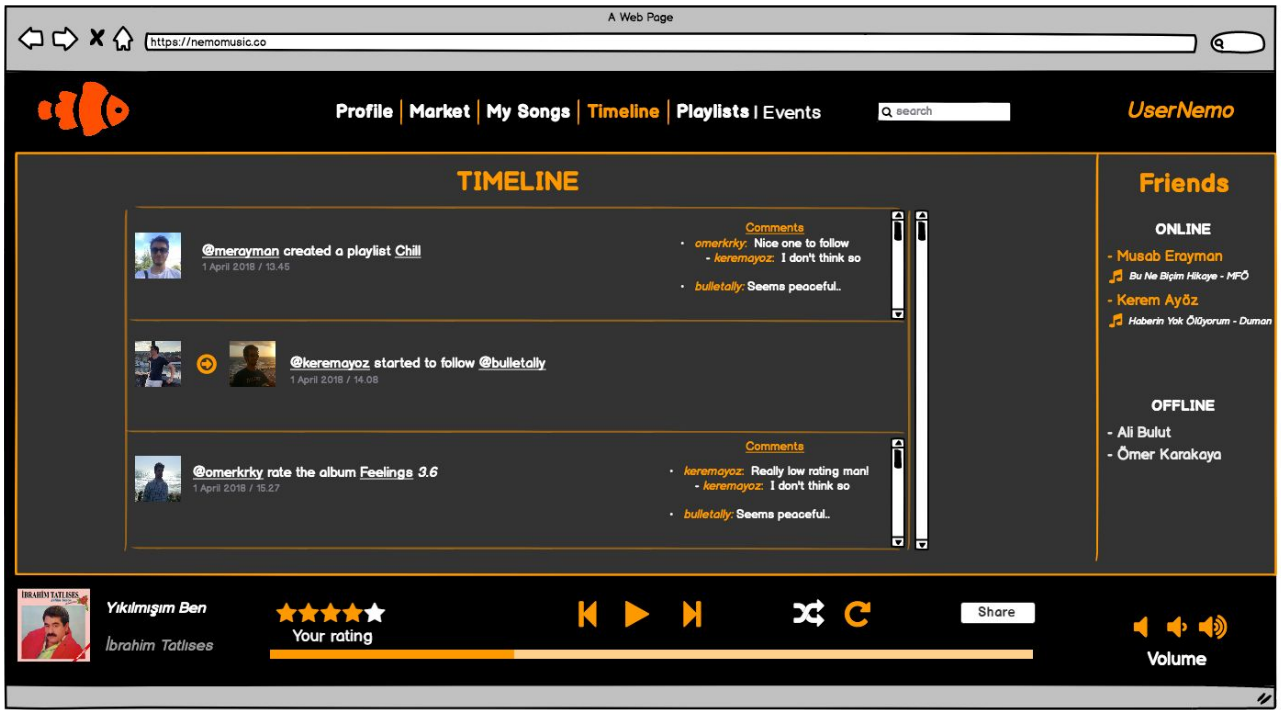
Sort by Release Date:

```
SELECT s.ID, a.ID
FROM Song s, Album a
ORDER BY s.release_date desc, a.release_date desc
```

Sort by Price:

```
SELECT s.ID, a.ID
FROM Song s, Album a
ORDER BY s.price asc, a.price asc
```

5.9. Timeline View



@merayman created a playlist Chill
1 April 2018 / 13.45

Comments
- omerkrky: Nice one to follow
  - keremayoz: I don't think so
- bulletally: Seems peaceful..

@keremayoz started to follow @bulletally
1 April 2018 / 14.08

@omerkrky rate the album Feelings 3.6
1 April 2018 / 15.27

Comments
- keremayoz: Really low rating man!
  - keremayoz: I don't think so
- bulletally: Seems peaceful..

ONLINE
- Musab Erayman
  Bu Ne Biçim Hikaye - MFÖ
- Kerem Ayöz
  Haberin Yok Ölüyorum - Duman

OFFLINE
- Ali Bulut
- Ömer Karakaya

Yıkılmışım Ben
İbrahim Tatlıses

Figure 12. Timeline View

Inputs:

@OnlineUserID @SelectedSongID

Process:

When a user opens timeline view, recent activities from the following users will be shown on the screen. These activities will include recently created comments, follow activities, public playlist creation and rating of album, song and playlist. Also, on the right of screen online users with the currently listening music and offline users will be shown.

SQL Statements:

Listing Timeline of Specific User:

```
SELECT *
FROM Activity A
WHERE A.UserID = any ( SELECT follower_id
                       FROM User_follow
                       WHERE following_id = OnlineUserID);
ORDER BY A.date desc
```

Listing All Users That User Follows:

```
SELECT *
FROM User U
WHERE U.UserID = any ( SELECT following_id
                       FROM User_follow
                       WHERE follower_id = OnlineUserID);
```

Getting Specific Song by ID:

```
SELECT *
FROM Song
WHERE ID = SelectedSongID;
```

5.10. After Search View

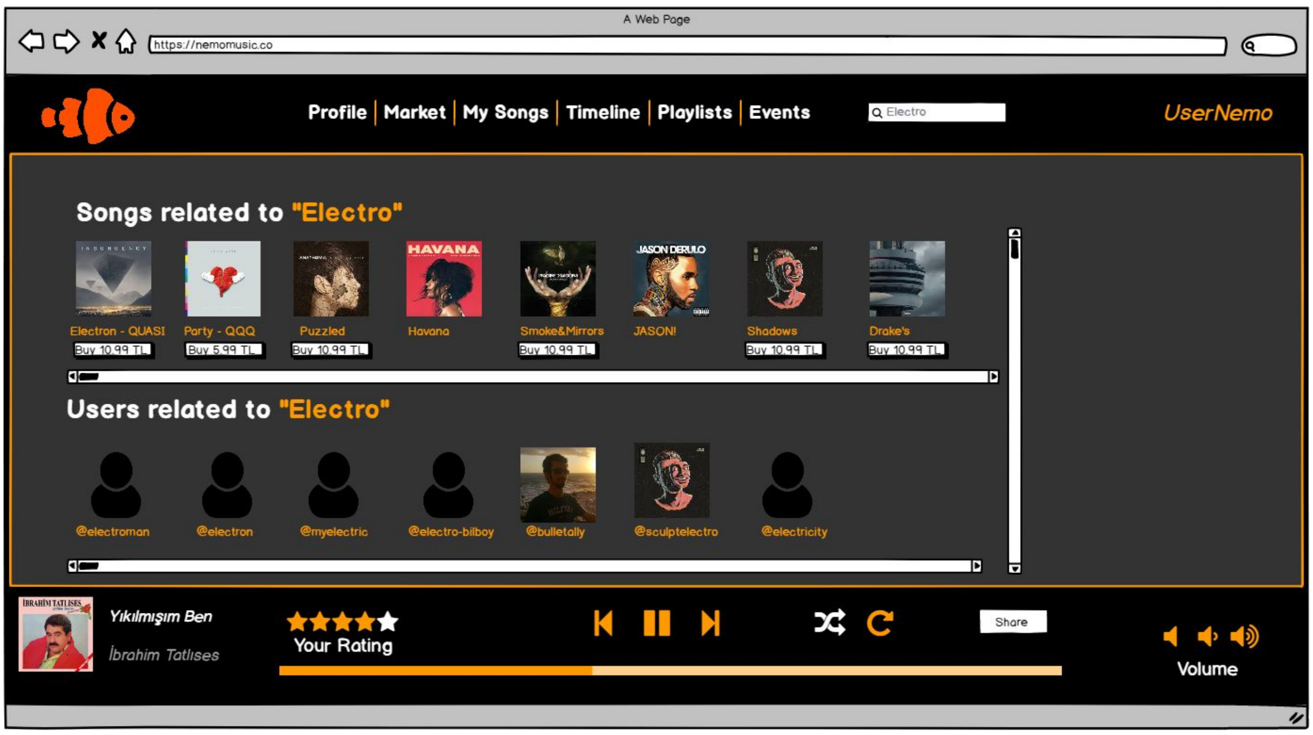


Figure 13. After Search View

Inputs:

@OnlineUserID @SearchKey

Process:

Users are able to search songs, albums, playlists, events and other users with single search box. Searching with multiple tables will be handled by stored procedures and dynamic search conditions.

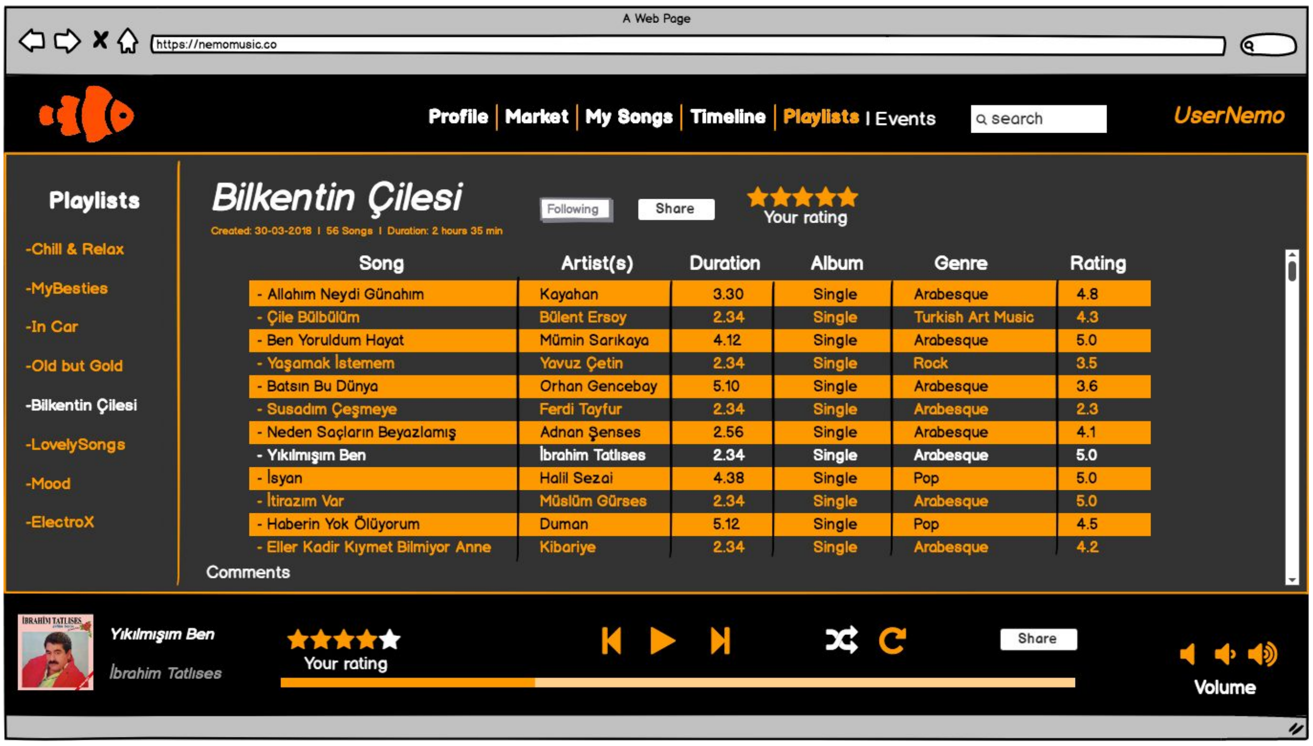
SQL Statements:

Search:

Executing stored procedure:

exec search @song_title = @SearchKey
exec search @event_name = @SearchKey
exec search @playlist_title = @SearchKey
exec search @album_title = @SearchKey
exec search @username = @SearchKey

5.11. Playlist View



Bilkentin Çilesi

Created: 30-03-2018 I 56 Songs I Duration: 2 hours 35 min

Song	Artist(s)	Duration	Album	Genre	Rating
- Allahım Neydi Günahım	Kayahan	3.30	Single	Arabesque	4.8
- Çile Bülbülüm	Bülent Ersoy	2.34	Single	Turkish Art Music	4.3
- Ben Yoruldum Hayat	Mümin Sarıkaya	4.12	Single	Arabesque	5.0
- Yaşamak İstemem	Yavuz Çetin	2.34	Single	Rock	3.5
- Batsın Bu Dünya	Orhan Gencebay	5.10	Single	Arabesque	3.6
- Susadım Çeşmeye	Ferdi Tayfur	2.34	Single	Arabesque	2.3
- Neden Saçların Beyazlamış	Adnan Şenses	2.56	Single	Arabesque	4.1
- Yıkılmışım Ben	İbrahim Tatlıses	2.34	Single	Arabesque	5.0
- İsyan	Halil Sezai	4.38	Single	Pop	5.0
- İtirazım Var	Müslüm Gürses	2.34	Single	Arabesque	5.0
- Haberin Yok Ölüyorum	Duman	5.12	Single	Pop	4.5
- Eller Kadir Kıymet Bilmiyor Anne	Kibariye	2.34	Single	Arabesque	4.2

Figure 14. PlaylistView

Inputs:

@OnlineUserID @SelectedSongID @SelectedPlaylistID @date @rateValue @shareComment

Process:

When a user opens playlist screen, first playlist will be shown as a default. User can choose other playlists as well. On the screen the songs that included in the selected playlist will be shown. Their name, artist(s), duration, album name, genre and rating will be selected from Playlist_song and Song table.

SQL Statements:

Listing All Playlists of Some User:

SELECT *
FROM Playlist P, User U
WHERE U.ID = @OnlineUserID **and** P.UserID = @OnlineUserID

Getting Specific Song:

```
SELECT *
FROM Song
WHERE ID = SelectedSongID;
```

Listing all Songs in Specific Playlist:

```
SELECT *
FROM Song S
WHERE S.ID = ( SELECT ps.song_id
               FROM  Playlist_song ps
               WHERE ps.ID = SelectedPlaylistID);
```

Following a Playlist:

```
INSERT INTO Activity(ID,date,entity_type,action_type,Entity_ID,UserID)
VALUES ( ID,date,Playlist_type,follow,SelectedPlaylistID,OnlineUserID);
```

Unfollowing a Playlist:

```
DELETE FROM Activity A
WHERE A.entityID = SelectedPlaylistID and A.UserID = OnlineUserID and A.ActionType = follow
```

Rate a Playlist:

```
IF EXIST ( SELECT *
        FROM Activity  A
        WHERE A.UserID = OnlineUserID and A.Entity_ID = SelectedPlaylistID
        and A.action_type = rate)

  UPDATE Rate SET (value = rateValue) WHERE activityID = A.ID'

ELSE
  ( INSERT INTO Activity(ID,date,entity_type,action_type,Entity_ID,UserID)
    SET @last_id_in_table = LAST_INSERT_ID();
    VALUES (,date,playlist,rate,SelectedPlaylistID,OnlineUserID)

    INSERT INTO Rate(activity_id,value)
    VALUES (last_id_in_table,rateValue));
```

Share a Playlist:

```
INSERT INTO Activity(ID,date,entity_type,action_type,Entity_ID,UserID)
SET @last_id_in_table = LAST_INSERT_ID();
VALUES(,date,playlist,rate,SelectedPlaylistID,OnlineUserID)

INSERT INTO Share(activity_id,value)
VALUES(last_id_in_table,rateValue,shareComment));
```

Display Comments of a Playlist:

```
SELECT C.text A.date U.username
FROM Comment C, Activity A, User U
WHERE C.activity_id = A.ID and U.ID = A.user_id and A.entity_id = SelectedPlaylistID and
A.action_type = comment);
```

6. Advanced SQL Statements

6.1. Reports

Total Duration of Playlist

```
SELECT sum(S.duration) as totalDuration
FROM Song S
WHERE S.ID = ( SELECT ps.song_id
               FROM  Playlist_song ps
               WHERE ps.ID = SelectedPlaylistID);
```

6.2. Views

No need for views since there is only single user type for accessing directly to database in Nemo, which is the admin of the database.

6.3. Triggers

- When user tries to buy a new song or album which costs more than his/her budget, program will give an message to the user that displays insufficient funds error.
- Assume that user has a single song from an album. When user tries to buy the entire album, program automatically detects that single song and do not add it into the user_song table again.
- When user tries to change his/her email into an existing one, program will give a warning message to user and do not change his/her email until user gives a unique email.
- When user tries to change his/her username into an existing one, program will give a warning message to user and do not change his/her username until user gives a unique username.
- When user creates a playlist, program automatically creates an activity for it and inserts the information about that activity into activity table.

6.4. Constraints

- The system cannot be used without logging in
- An event must have a place and time to exist.
- User must have a unique email and username to sign up.
- A playlist should be created by some user and must be either public or private.
- User should have a password with at least 6 characters that are alpha-numeric.
- Every song need to have an artist and a genre to exist.
- Every activity should belong to a user.

6.5. Stored Procedures

- Stored Procedure will be used for search. With the input from single search textbox, from users, songs, albums, playlist and events this searckey will be searched. Corresponding procedure will be held on Database and server will just execute this with different searchkeys.

```
CREATE PROCEDURE search(
        in song_title varchar(20)
        in event_name varchar(20)
        in playlist_title varchar(20)
        in album_title varchar(20)
        in username varchar(20)
        out S.ID integer
        out S.title varchar(20)
        out E.ID integer
        out E.name varchar(20)
        out P.ID integer
        out P.title varchar(20)
        out A.ID integer
        out S.title varchar(20)
        out U.ID integer
        out U.username varchar(20)
         )
  BEGIN
        SELECT S.ID, S.title, E.ID, E.name, P.ID, P.title, A.ID, A.title, U.ID, U.username
        FROM Song S, Event E, Playlist P, Album A, User U
        WHERE S.title = “%@song_title%” and E.name = “%event_name%” and
                P.title = “%playlist_title%” and A.title =”%title%” and
                U.username = “%username%” ;
  END
```