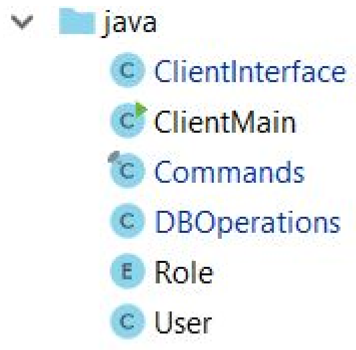
Wojciech Musiatowicz
Matteo Gilardi
Jakub Bielawski

Database Assignment Report#2

1. Introduction
   This report contains information about implementation of the database we designed for the previous assignment. We created a command line program, which takes user input and makes changes to a real database stored on the computer.
   Technologies, that were used:
- Java
- MySQL database engine

2. Running instructions
   In order to run the program, MySQL server must be running and Java JDK must be installed. Database is named 'db' and we used 'root' both for login and password. Database creation script and record insertion script are located in db.txt and inserts.txt. Another way to run the app with database is to copy folder 'db' to MySQL databases directory. The application should be run by entering "java -jar dbApp.jar" in the command prompt (make sure you are in the same directory).

3. File information
   Classes:



- ClientMain - Initializing class. Creates ClientInterface object.

- ClientInterface - Class, which gathers input from user, handles it and processes request to and from DBOperations class.
  fields:
  - private boolean isLoggedIn - indicate if user is logged in
  - private User currentlyLoggedUser - User object keeps currently logged user's data.
  - private DBOperations - instance of DBOperations class, used to access its methods.
- Commands - class containing names of user cases and associating them with user input.
- Role - class, which enumerates user types
- User - class, which contains information about currently logged user.

4. Use cases

Due to the database aspect importance, use cases describe only the expected outcome, omitting side scenarios like already existing username or wrongly entered data.

4.1. Register a new user

4.1.1. Scenario: Unlogged user enters his credentials and chooses his role (admin,editor,user). Program adds new user to database (USERS).

4.1.2. SQL:

```
SELECT COUNT(*)FROM USERS WHERE USERNAME = <username> //check if
user exists
INSERT INTO USERS (USERNAME,PASSWORD,EMAIL,TYPE) VALUES
(<username>, <password>  <email>, <role>)//insert data
```

4.2. Authenticate user

4.2.1. Scenario: unlogged user enters his credentials, database checks them and user is logged in.

4.2.2. SQL:

```
SELECT COUNT(*) FROM USERS WHERE username = <username>
AND password = <password>//should return 1 record
```

4.3. Help (doesn’t use SQL)

4.3.1. Scenario: User types command “help”. Program returns available commands.

4.4. Exit(doesn’t use SQL)

4.4.1. Scenario: User enters command “exit”. Program stops running.

4.5. Log out (doesn't use SQL)

4.5.1. Scenario: User enters command “logout”. Program logs out user.

4.6. Browse all songs

4.6.1. Scenario: user enters command “getAllSongs”, database retrieves list of songs in the database.

4.6.2. SQL:

```
select s.id,s.title,a.album_name,ar.artist_name
from songs s,albums_songs als,albums a,artists ar
where s.id=als.songs_id and als.albums_id = a.id and
a.artists_id=ar.id
```

4.7. Browse all artists

4.7.1. Scenario: user enters command “getAllArtists”, database retrieves list of artists in the database.

4.7.2. SQL:

```
select * from artists
```

4.8. Browse all albums

4.8.1. Scenario: user enters command “getAllAlbums”, database retrieves list of albums in the database.

4.8.2. SQL:

```
select a.id,a.album_name,ar.artist_name
from albums a,artists ar
where a.artists_id=ar.id
```

4.9. Insert new song(s) to album(s)

4.9.1. Scenario: User (editor or admin) enters command “insertSong”, program returns list of albums. User enters albums id(s) to which add song(s). Songs are saved in the database and assigned to previously picked albums.

4.9.2. SQL:

```
SELECT COUNT(*)FROM  songs where id = <id> //check if exists

insert into songs (title, length) values (<title>,<length>)//into
table SONGS
insert into albums_songs values (<album_id>,<song_id>);//into table
<albums_songs>
```

4.10. Insert musician

4.10.1. Scenario: User (editor or admin) enters command “createOneArtist”. User is prompted to enter data about new artist. Data is inserted into artists (ARTISTS)

4.10.2. SQL:

```
insert into artists (artist_name, musicians_full_name) values
(<artistName>,<musicianName>)// inserted data
```

4.11. Insert band

4.11.1. Scenario: User (editor or admin) enters command “createBand”. User is prompted to enter data about new band. Data is inserted into artists (ARTISTS)

4.11.2. SQL:

```
insert into artists (artist_name, bands_creation_date) values
(<band_name>,<date>)/inserted data
```

4.12. Insert new album

4.12.1. Scenario: User (editor or admin) enters command “insertAlbum”, program returns list of artists. User enters artist’s ID, to which the album will be added. Program returns list of publishers. User enters publisher’s ID, to which the album will be added. User enters album data. Album is saved in the database (ALBUMS).

4.12.2. SQL:

```
SELECT COUNT(*)FROM  albums where id = <id> //check if exists
```

```
insert into albums (album_name, creation_date, genre, descr,
publisher_id, artists_id) values
(<id>,<name>,<genre>,<description>,<artist_id>,<publisher_id>)
```

4.13. Update a song

4.13.1. Scenario: User (editor or admin) enters command “updateSong”, program returns list of songs. User enters song’s ID, which will be updated. Record in the database is updated (SONGS)

4.13.2. SQL:

```
SELECT COUNT(*)FROM  songs where id = <id> //check if entered ID
exists

update songs set title = <title> where id = <id_song> //update the
record
```

4.14. Delete a song

4.14.1. Scenario: User (editor or admin) enters command “deleteSong”, program returns list of songs. User enters song’s ID, which will be deleted. Records in the database are updated (SONGS, ALBUMS_SONGS)

4.14.2. SQL:

```
SELECT COUNT(*)FROM  songs where id = <id> //check if entered ID
exists

delete from songs where id = <id_song>
```

4.15. Delete an album

4.15.1. Scenario: User (editor or admin) enters command “deleteAlbum”, program returns list of albums. User enters album’s ID, which will be deleted. Records in the database are deleted(ALBUMS, ALBUMS_SONGS)

4.15.2. SQL:

```
SELECT COUNT(*)FROM  ALBUMS where id = <id> //check if entered ID
exists

delete from albums where id = <id_album>
```

4.16. Delete an artist

4.16.1. Scenario: User (editor or admin) enters command “deleteArtist”, program returns list of artists. User enters artist’s ID, which will be deleted. Records in the database are deleted(ARTISTS,ALBUMS)

4.16.2. SQL:

```
SELECT COUNT(*)FROM  ARTISTS where id = <id> //check if entered ID
exists
```

```
delete from artists where id = <id_artist>
```

4.17. Promote an user

4.17.1. Scenario: User (admin) enters command “promote”, enters promoted user’s username and type (admin, user, editor). Record is updated in the database (USERS)

4.17.2. SQL:

```
SELECT COUNT(*)FROM  ARTISTS where id = <id> //check if entered ID
exists

update users set type = <type> where username = <username>
```

4.18. Show shared files

4.18.1. Scenario: User enters command “getSharedSongs”, program returns list of files that are shared to this user (by group or individually).

4.18.2. SQL:

```
SELECT distinct s.songs_id,s.filename
       FROM songfiles s, usershares u
       where u.users_id = <user_id>and
       s.filename = u.songfiles_filename //all files from user shares
union
select distinct s.songs_id,s.filename
       FROM songfiles s, groupshares g
       where s.filename = g.songfiles_filename and g.groups_id
       in (select ug.groups_id from  users_groups ug where <user_id> =
ug.users_id)//all files from group shares
```

4.19. Upload a song

4.19.1. Scenario: User enters command “sendSong”, program returns list of songs. User picks ID of a song to associate it to file. User enters filename. File name is saved in the database (SONGFILES,USERSHARES)

4.19.2. SQL:

```
SELECT COUNT(*)FROM  songs where id = <id> //check if entered ID
exists

insert into songfiles values
(<filename>,<user_id>,<song_id>)//insert filename

insert into usershares values (<filename><shared_user_id>)//song is
shared with "himself"
```

4.20. Delete user

4.20.1. Scenario: User (admin) enters command “deleteUser”, program returns list of users. User enters user’s username, which will be deleted. User is deleted from the database, as well as groups, which he owns.

4.20.2. SQL:

```
SELECT COUNT(*)FROM users where username= <username> //check if
exists

delete from groops where owner_id = (select id from users where
username= <username>) //get ID from username and delete the group

delete from users where username=<username> //delete the user
```

4.21. Get list of all users

4.21.1. Scenario: User enters command “getAllUsers”, program returns list of users.

4.21.2. SQL:

```
SELECT * FROM USERS //get all users
```

4.22. Share song with a group

4.22.1. Scenario: User enters command “shareSong”, enters “group”. Program returns list of groups. User enters group’s ID, with which he wants to share the file. File is shared with the group. (GROUPSHARES)

4.22.2. SQL:

```
SELECT COUNT(*)FROM users_groups WHERE
groups_id = <group_id> and users_id = <user_id>//check if file is
already shared with this group.

insert into groupshares values (<filename>,<group_id>)//share with
a group
```

4.23. Share song with another user

4.23.1. Scenario: User enters command “shareSong”, enters “user”. Program returns list of users. User enters username, with which he wants to share the file. File is shared with the user. (USERSHARES)

4.23.2. SQL:

```
SELECT COUNT(*)FROM usershares WHERE users_id=<user_id> and
songfiles_filename = <filename> // check if file is already shared
with this user.

insert into usershares values (<filename>,<user_id>)//song is
shared with the user
```

4.24. Search music

4.24.1. Scenario: User enters command “searchMusic”. User enters phrase, program returns list with music records matching given phrase.

4.24.2. SQL:

```
select s.title,a.album_name,ar.artist_name
       from songs s,albums_songs als,albums a,artists ar
       where s.id=als.songs_id and als.albums_id = a.id and
a.artists_id=ar.id
       and (s.title like('%<phrase>%') or a.album_name
like('%<phrase>%') or
       ar.artist_name like('%<phrase>%')) //joins tables to view
all data
```

4.25. Delete a group

4.25.1. Scenario: User enters command “deleteGroup”, program returns list of groups. User enters ID of group he wants to delete. Record is deleted from database (GROOPS, USERS_GROUPS)

4.25.2. SQL:
Checking of which groups currently logged in user is owner:

```
SELECT owner_id, name FROM groops JOIN users ON users.id = owner_id
WHERE users.username = <username>
```

deleting:

```
DELETE FROM groops WHERE id = <id of group that currently logged in
user is owner>
```

4.26. Create a group

4.26.1. Scenario:User enters command “deleteGroup”, program prompts user to enter new group name.

4.26.2. SQL:

```
insert into groops (name, owner_id) values (<name>,<group_id>)
//insert group

insert into users_groups values (<owner_id>,<group_id>,<isEditor>)
//associate user with the group
```

4.27. Add user to existing group

4.27.1. Scenario: User enters command “addUserToGroup”, program returns list of users and groups. User enters username of the user and ID of the group. Program prompts user to decide if he should have group editing rights. User is added to the group.

4.27.2. SQL:

```
SELECT COUNT(*)FROM users where username= <username>//check if
user exists
```

```
SELECT COUNT(*)FROM groups where id= <group_id> //check if group
exists

SELECT COUNT(*)FROM users_groups where isEditor = 1 and groupid =
groups_id and ownerid = users_id //check if user can edit the
group

SELECT COUNT(*)FROM users_groups
where <groupid> = groups_id and <userid> = users_id //check if
added user already belongs to the group

insert into users_groups values (<userid>, <groupid>,
<isEditor>)//insert user into group
```

4.28. Delete user from group

4.28.1. Scenario: User enters command “deleteUserGroup”, program returns list of users and groups.

4.28.2. SQL:

```
SELECT COUNT(*)FROM users where username= <username>//check if
user exists

SELECT COUNT(*)FROM groups where id= <group_id> //check if group
exists

SELECT COUNT(*)FROM users_groups where isEditor = 1 and groupid =
groups_id and ownerid = users_id //check if user is even in this
group

SELECT COUNT(*)FROM groops where <groupid> = id and <userid>
=owner_id//check if deleted user is owner of the group
delete from users_groups where groupid=groups_id and
 users_id =<users_id>//delete user
```

4.29. Create a playlist

4.29.1. Scenario: User enters command “createPlaylist”. Program prompts user to enter a name for the playlist. New playlist is inserted in the database (PLAYLISTS, USERS_PLAYLISTS)

4.29.2. SQL:

```
insert into playlists (name) values (<playlistname>)//insert
playlist

insert into users_playlists values (
<userid>,<playlistid>)//associate playlist with user
```

4.30. Add a song to playlist

4.30.1. Scenario: User enters command “addToPlaylist”. Program views list of playlists that belong to user. User enters ID of the playlist and song in which he wants to insert. The data is inserted into PLAYLISTS_SONGS.

4.30.2. SQL:

```
select p.id,p.name from playlists p, users_playlists up
where up.playlists_id = p.id and up.users_id = <userid>//show all
playlists that belong to this user

SELECT COUNT(*)FROM playlists where playlist = <playlistname>
//check if playlist exists

SELECT COUNT(*)FROM songs where id= <song_id> //check if group
exists

SELECT COUNT(*)FROM users_playlists
       where playlists_id = <idplay> and
       users_id = <idsong>//check if playlist belongs to user

insert into playlists_songs values (<idplay>,<idsong>)//associate
song with playlist
```

4.31. Delete a song from playlist

4.31.1. Scenario: User enters command "deletePlaylist". Program returns list of playlists and prompts user to enter ID of playlist to delete. After entering, the playlist is deleted from the database. (PLAYLISTS,USERS_PLAYLISTS)

4.31.2. SQL:

```
SELECT COUNT(*)FROM songs where id= <playlistid> //check if
playlist with that id exists

SELECT COUNT(*)FROM users_playlists
       where playlists_id = <playlistid> and
       users_id = <userid> //check if playlist was previously added
by the user

delete from playlists where id = <playlistid>//delete
```

4.32. Show all playlists

4.32.1. Scenario: User enters command "getAllPlaylists". Program returns list of all playlists.

4.32.2. SQL:

```
select * from playlists
```

4.33. writeReview

4.33.1. Scenario: User enters command "writeReview". Program returns list of all albums. User enters ID of album to which he wants to write the review, and then enters the review text. Review is inserted into database (REVIEWS)

4.33.2. SQL:

```
SELECT COUNT(*)FROM albums where id= <albumid> //check if album
with that id exists

insert into reviews (text, users_id, albums_id)
       values (<review>,<userid>,<albumid>)//insert the review
```

4.34. Delete a review

4.34.1. Scenario: User (editor or admin) enters command “deleteReview”. Program returns list of all reviews. User enters ID of review which he wants to delete. Review is deleted from database. (REVIEWS)

4.34.2. SQL:

```
SELECT COUNT(*)FROM reviews where id= <reviewid> //check if review
with that id exists

SELECT COUNT(*)FROM reviews
       where id = <revid>  and
       users_id = <userid> //check if that review is written by
this user

delete from reviews where reviews.id= <revid> //delete review
```

4.35. Show my reviews

4.35.1. Scenario: User enters command “showMyReviews”. Program returns list of reviews written by user.

4.35.2. SQL:

```
select r.id,r.text,a.album_name from reviews r,albums a
       where <userid>= r.users_id and r.albums_id = a.id //select
reviews written by user
```

4.36. Show all reviews

4.36.1. Scenario: User enters command “showAllReviews”. Program returns list of all albums. User enters ID of album and program returns list of all reviews written about this album.

4.36.2. SQL:

```
SELECT COUNT(*)FROM albums where id= <albumid> //check if album
with that id exists

select r.id,r.text,r.users_id from reviews r,albums a
       where <albumid> = r.albums_id and r.albums_id = a.id
```

4.37. Show all publishers

4.37.1. Scenario: User enters command “getAllPublishers”. Program returns list of all publishers.

4.37.2. SQL:

```
select * from concerts
```

4.38. Show all concerts

4.38.1. Scenario: User enters command “getAllConcerts”. Program returns list of all publishers.

4.38.2. SQL:

```
select * from concerts
```

4.39. Insert a new publisher

4.39.1. Scenario: User (editor or admin) enters command “insertPublisher”. User is prompted to insert new name. Publisher is inserted into database (PUBLISHERS).

4.39.2. SQL:

```
insert into publisher (name) values <publisher_name>
```

4.40. Delete a concert

4.40.1. Scenario: User (editor or admin) enters command “deleteConcert”. Program returns list of all concerts. User is prompted to enter ID of the concert he wants to delete. Program deletes concert from database. (CONCERTS).

SQL:

```
SELECT COUNT(*)FROM concerts where id= <concertid> //check if
concert with that id exists

delete from concerts where concerts.id= <concertid> //delete
concert
```

4.41. Insert a concert

4.41.1. Scenario: User (editor or admin) enters command “createConcert”. User is prompted to enter data about the concert. Program inserts concert into database (CONCERTS).

4.41.2. SQL:

```
insert into concerts (date_start, date_end, name)
       values (<start_date>,<end_date>,<name>)//insert concert
```

4.42. Add artist to a concert

4.42.1. Scenario: User (editor or admin) enters command “makeLineup”. Program returns list of concerts and artists. User is prompted to enter ID of concert and artist to associate. Artist is associated with concert in the database. (CONCERTS_ARTISTS)

4.42.2. SQL:

```
SELECT COUNT(*)FROM artists where id= <artistid> //check if artist
with that id exists

SELECT COUNT(*)FROM concerts where id= <concertid> //check if
concert with that id exists

SELECT COUNT(*) FROM concerts_artists
       where artists_id = <artist_id> and
       concerts_id = <concertid> //check if artist is already in
the lineup.

insert into concerts_artists
```

```
        values (<concert_id>,<artist_id>)//insert
```

4.43. Show lineup for the concert

4.43.1. Scenario: User enters command “getLineup”. Program returns list of concerts. User is prompted to enter id of concert. Program returns list of artists that will play at that concert.

4.43.2. SQL:

```
SELECT COUNT(*)FROM concerts where id= <concertid> //check if
concert with that id exists

select c.name,a.artist_name
       from concerts c, artists a, concerts_artists ca
       where c.id = ca.concerts_id and ca.artists_id=a.id
       and <concert_id> = ca.concerts_id
```

4.44. Show songs in a playlist

4.45.1 Scenario: User enters command “showPlaylistSongs”. Program returns list of songs for given playlist

4.45.2 SQL:

First showing all playlists for currently logged in user:

```
SELECT name
FROM playlists
JOIN users_playlists ON users_playlists.playlists_id = playlists.id
WHERE users_playlists.users_id = (
             SELECT id
       FROM users
       WHERE users.username = <username>
       )
```

showing all songs for given playlist:

```
SELECT title, length
FROM songs
JOIN playlists_songs ON playlists_songs.songs_id = songs.id
WHERE playlists_songs.playlists_id = (
       SELECT id
    FROM playlists
    WHERE playlists.name = <playlist.name>
    )
```