Web Services

COMP09048

Radoslaw Burkacki

B00309449

Playlist exchange service

Table of Contents

[Declaration(i) 2](#_Toc467771112)

[Team members(i) 2](#_Toc467771113)

[Description of project(ii) 2](#_Toc467771114)

[Interaction pattern(ii) 2](#_Toc467771115)

[Development tools(ii) 2](#_Toc467771116)

[Data store(iii) 2](#_Toc467771117)

[Interface(s) between client and server(iv) 3](#_Toc467771118)

[Design of client and server component (v) 5](#_Toc467771119)

[A table of client functions (vi) 13](#_Toc467771120)

[A table of service endpoints (vii) 14](#_Toc467771121)

[A user guide for your client application(viii) 15](#_Toc467771122)

[A paragraph from each project collaborator(ix) 20](#_Toc467771123)

[PDP(x) 20](#_Toc467771124)

[A section of outstanding issues for the project(xi) 20](#_Toc467771125)

[Source code (xii) 21](#_Toc467771126)

[AccountResource(server): 21](#_Toc467771127)

[PlaylistResource(Server): 23](#_Toc467771128)

[StatisticResource(Server): 26](#_Toc467771129)

[AccountService(Server): 28](#_Toc467771130)

[PlaylistService(Server): 30](#_Toc467771131)

[StatisticService(Server): 37](#_Toc467771132)

[DatabaseClass(Server): 39](#_Toc467771133)

[Account(Server): 40](#_Toc467771134)

[Playlist(Serve): 41](#_Toc467771135)

[Song(Server): 43](#_Toc467771136)

[Statistic(Server): 44](#_Toc467771137)

[Top10Song(Server): 45](#_Toc467771138)

[RestApiClient(Client): 47](#_Toc467771139)

[Account(client): 69](#_Toc467771140)

[Playlist(client): 70](#_Toc467771141)

[Song(client): 71](#_Toc467771142)

[Statistic(Client): 72](#_Toc467771143)

[Top10Song(client): 74](#_Toc467771144)

# Declaration(i)

I declare that work submitted for this coursework is my original work.

# Team members(i)

Radoslaw Burkacki – B00309449

All responsibilities were on myself as I have done this course work on my own. I have done: design, development and testing of client and of the web service itself.

# Description of project(ii)

Purpose of this service based application is to create a web service and a client that will allow users to register, login, view playlists, add playlists, add songs into own playlists and like/dislike playlists. As it is a web service application it will allow anyone that does have access to internet to access my web service.

# Interaction pattern(ii)

The interaction patter that I have used it REST. I have used it because I feel that this pattern is the most suitable for this type of project, also my knowledge about REST is better than about SOAP.

# Development tools(ii)

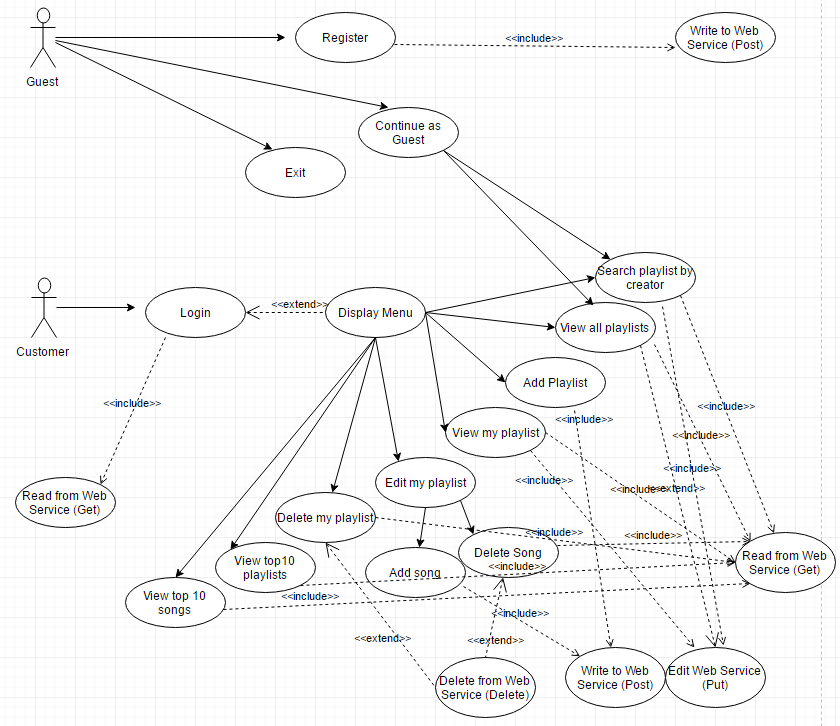
To develop this project, I have used Eclipse Java EE Neon 4.6.0. I have developed the web service with Jersey, to run this web service I have used Tomcat 9.0.

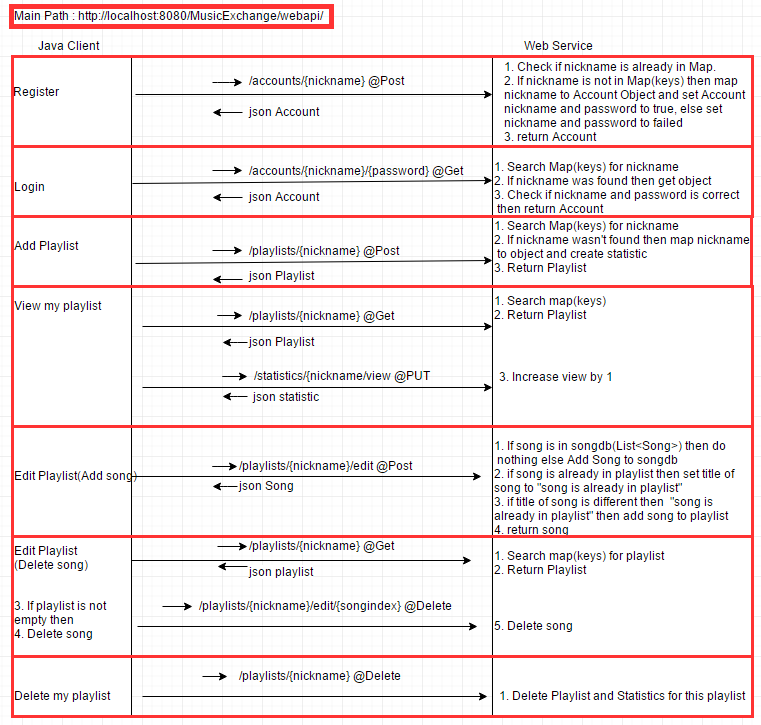
# Data store(iii)

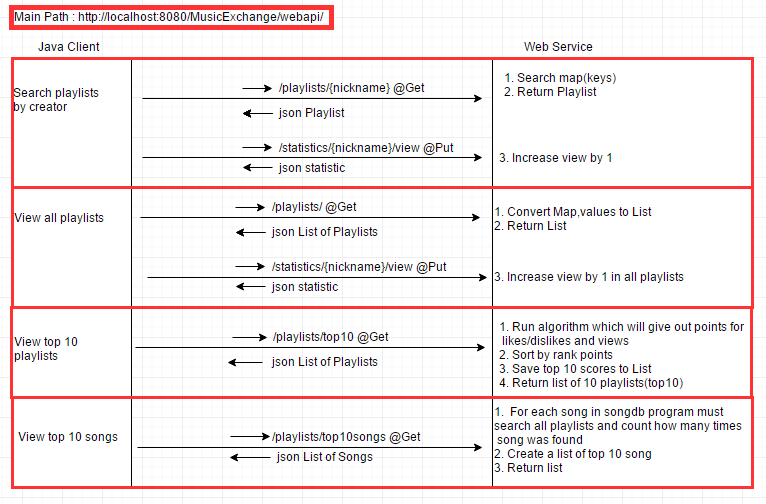
All the data that is stored on the server, is stored in Maps and lists. I haven’t saved this data to database or an xml/json file. Issue that comes with my solution is when the server would be powered off then all the data would be lost. I have created special class in my program to save the data, in side of this class there are 4 main data sources, there are 3 maps and one list.

* Playlists – First data source – To store playlists I have created a Map (String, Playlist). Key for this map is a string which is a nickname of a user. I have mapped nickname to a Playlist object which was created by that user. All playlists are saved in side of this Map.
* Accounts – Second data source – To store account details I have created another Map (String, Account). Key for this map is a string which is a nickname of user. I have mapped nickname to Account object. All accounts are saved in this Map.
* Playlist statistics – Third data source – To store statistics about playlists I have created a Map (String, Statistic). Key for this map is a string which is a Title of a playlist. I have mapped the title of a playlist to a Statistic object. All playlist statistics are saved in this map.
* Song db – Fourth data source – To store a full list of songs I have created a List (Song), it’s a List of Song objects. I have created this to keep track of all songs added to my web service.

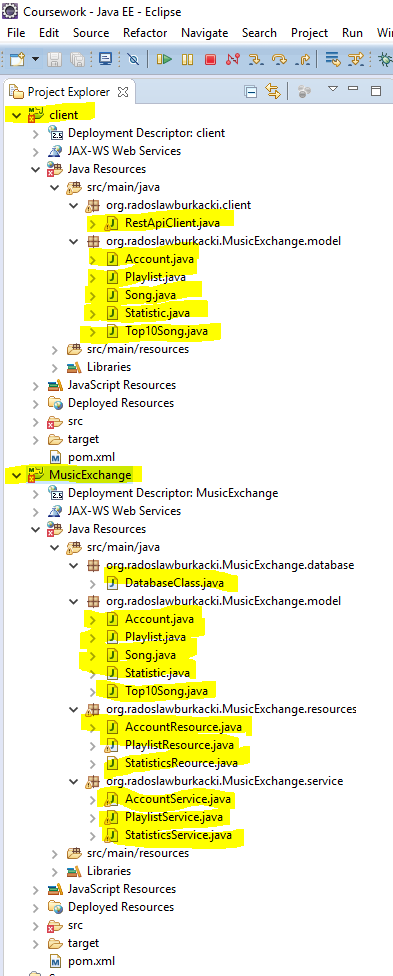
# Interface(s) between client and server(iv)







# Design of client and server component (v)



|  |
| --- |
| Class name: Account Resource **(Server)** |
| Description: This class is main entry for all the calls regarding Accounts, for example when customer is creating new Account, client is connecting with server and client sends post call then this class is called and function responsible for Post calls is run. This function is calling another function which takes care of all the logic. This class creates instance of AccountService which handles all the logic. |
| Attributes: none |
| Functions:   * @GET public List<Account> getAllAccounts() – This function is calling another function (accountservice.getAllAccounts()) and returning list of all accounts. * @POST public Response addAccount(@PathParam("nickname") String nickname, Account acc) – This function is taking parameter from url and saving it to variable. It also expects account to be passed into it when called. This function calls another (accountservice.addAccount(nickname, acc)) function which handled all the logic and passes nickname and account into it. It returns Account object. * @ GET public Account validateAccount(@PathParam("nickname") String nickname, @PathParam("password") String password) – This function is saving two parameters from url. This class is calling another function (accountservice.validateAccount(nickname , password)) which handles validation of account(login). It returns account. |

|  |
| --- |
| Class name: Account Service **(Server)** |
| Description: This class handles all the logic of Accounts. An instance of this class is created in Account Resource class and all the calls care coming from there. |
| Attributes: |
| Functions:   * public AccountService() – This is a constructor, in here I’m creating an instance of Account. I’m using it to do most of the logic. * public List<Account> getAllAccounts () – This class is returning a list of accounts. It is taking values from Map and converts them into List. * public Account addAccount (String nickname, Account acc) – This class expects a string(nickname) and Account object to be passed into it. It checks if Account already exists, if it exists then it sets nickname and password to failed (based on that client knows what happened), if account is not in map then it mapped account to nickname. Returns account. * public Account validateAccount (String nickname, String password) – This class expects two strings (nickname, password) to be passed into it. It creates new account and sets its nickname and password to received strings. It searches Map for this account. If nickname and password is right, then it sets nickname and password to “true” and returns Account. |

|  |
| --- |
| Class name: Playlist Resource **(Server)** |
| Description: This class is main entry for all the calls regarding Playlists and songs. This class is calling functions which are called to server by client, those functions calls other functions which handles logic. This class creates instance of PlaylistService which contains all the logic. |
| Attributes: |
| Functions:   * @GET public List<Playlist> getAllPlaylists() – This function (playlistservice.getAllPlaylists()) calls another function which handles logic, it returns a List of playlists. * @GET public List<Playlist> getTop10Playlists() – This function (playlistservice.getTop10Playlists()) calls another function which handles logic, it returns List of playlists. * @GET public List<Top10Song> getTop10Songs() - This function (playlistservice.getTop10Songs()) calls another function which handles logic, it returns List of top10 songs. * @GET public Response getPlaylist(@PathParam("nickname") String nickname) – This function was created to get one playlist as a response. Users can search playlist by nickname using this function. It calls another function (playlistservice.getPlaylist(nickname)) which is handling the logic. It returns playlist. * @POST public Response addPlaylist(@PathParam("nickname") String nickname,Playlist playlist) – This function is used to create new playlists. It is calling another function (playlistservice.addPlaylist(nickname, playlist)) which handles logic. It returns a Playlist. * @DELETE public void deletePlaylist(@PathParam("nickname")String nickname) – This function calls another function (playlistservice.deletePlatlist(nickname)) which is handling deletion of playlist. Its not returning anything. * @POST public Response addSongToPlaylist(@PathParam("nickname") String nickname,Song s) – This function (playlistservice.addSongToPlaylist(nickname,s)) calls another function which handles logic. This function is used to add songs to playlist. It returns song object. * @DELETE public void removeSongFromPlaylist(@PathParam("nickname")String nickname,@PathParam("songindex")int songindex) – This function is used to delete songs from playlists. It calls another function (playlistservice.deleteSong(nickname, songindex)) which handles logic. Its not returning anything. |

|  |
| --- |
| Class name: Playlist Service**(Server)** |
| Description: This class handles all logic of playlists and songs. An instance of this class is created in Playlist Resource class and all the calls care coming from there. |
| Attributes: |
| Functions:   * public List<Playlist> getAllPlaylists() – This function is used to get all playlists. This function is getting values out of map and converting them to List, returns list. * public Playlist getPlaylist (String nickname) – This function is used to get one playlist using nickname. This function is searching map for nickname of customer, it returns whatever is returned by map. * public List<Playlist> getTop10Playlists() – This function is used to get top10 playlists. It is using an algorithm which uses playlist statistics to give out points to playlists. It sorts those playlists based on number of points. It saves top 10 songs to map. It converts from map to list and returns that list. * public static <String, Integer extends Comparable<? super Integer>> Map<String, Integer> sortTop10( Map<String, Integer> map ) – This function is used to sort top10 songs. It takes map (unsorted) and sorts it. Returns map. * public Playlist addPlaylist(String nickname, Playlist playlist) – This function is used to add playlist to map. First it checks if this user owns a playlist. If not then it checks if playlist title is unique and if its true then it creates new playlist. Returns playlist. * public Song addSongToPlaylist(String nickname, Song s){ // receiving nickname of user and song (title, artist, album genre) – This function is used to add songs to playlist. This function is looping through songdb to see if this song is already in it, if not then it adds the song into songdb, next it checks if song is already in playlist if not then it adds song into playlist. Returns song. * public List<Top10Song> getTop10Songs() – This function Is used to get top 10 songs. Fist it loops through all playlists to see how many time “x” song was in how many playlists. It counts that number for all songs, then it sorts songs bases on that number, makes copy of top 10 songs and returns it. * public void deleteSong(String nickname, int songindex) – This function is used to delete songs from playlists. * public void deletePlatlist(String nickname) – This function is used to delete playlist, when playlist is deleted also statistics for that playlists are deleted. |

|  |
| --- |
| Class name: Statistics Service **(Server)** |
| Description: This class handles all calls that are coming to server about statistics. This class is calling functions which are called to server by client, those functions calls other functions which handles logic. This class creates instance of StatisticsService which contains all logic. |
| Attributes: |
| Functions:   * @GET public Statistic getPlaylistStatistics(@PathParam("nickname") String nickname) – This function is used to get playlist statistics. It calls another function (statisticservice.getPlaylistStatistic(nickname)). It returns object of statistic. * @PUT public void setAllPlaylistView() – This function is used to increase view of all playlists by 1. Its not returning anything. * @PUT public void setPlaylistView(@PathParam("nickname") String nickname, Statistic s) – This function is used to increase view of one playlist by 1. This function calls another function (statisticservice.increaseViewOfOnePlalist(nickname)) which handles all logic. * @POST public Response ratePlaylist(@PathParam("title") String title, @PathParam("nickname") String nickname, Statistic s) – This function is used to add likes/dislikes to playlist statistic. This function calls another function (statisticservice.likePlaylist(title,nickname,s)) which handles all logic. |

|  |
| --- |
| Class name: Statistics Resource **(Server)** |
| Description: This class handles all logic of statistics an instance of this class is created in Statistics Resource class and all the calls care coming from there. |
| Attributes: |
| Functions:   * public Statistic getPlaylistStatistic(String nickname) – This function is used to get statistics of playlist using nickname. * public void increaseViewOfOnePlalist(String nickname) – This function is used to increase view of one playlist. * public void increaseViewOfAllPlaylists() – This function Is used to increase view by 1 of all playlists. * public String likePlaylist(String title, String nickname, Statistic s) – This function Is used to like playlists. * public String dislikePlaylist(String title, String nickname, Statistic s) – This function I used to dislike playlists. |

|  |
| --- |
| Class name: DatabaseClass **(Server)** |
| Description: This class is storing all the data. |
| Attributes: |
| Functions:   * public static Map<String,Account> getAccounts () – This function is used to get all Accounts. * public static Map<String,Playlist> getPlaylists () - This function is used to get all Playlists. * public static Map<String, Statistic> getPlaylistStatistics () - This function is used to get all playlist statistics. * public static List<Song> getsongdb() - This function is used to get all songs. |

|  |
| --- |
| Class name: Account**(Server and client)** |
| Description: This class is a blue-print for Account objects. |
| Attributes: String nickname, String password. |
| Functions:   * public Account() – Empty constructor (requirement for jersey to work properly) * public Account( String nickname, String password) – Constructor, When object is created it is expecting two strings. * public String getNickname() – Used to get nickname. * public void setNickname(String nickname) – Used to set nickname. * public String getPassword() – Used to get password. * public void setPassword (String password) – Used to set password. |

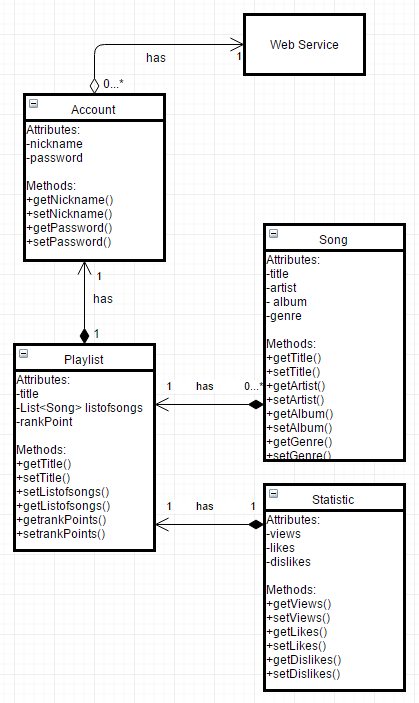
|  |
| --- |
| Class name: Playlist**(Server and client)** |
| Description: This class is a blue-print for Playlist objects. |
| Attributes: String title, List<Song>, int rankPoints |
| Functions:   * public Playlist() - Empty constructor (requirement for jersey to work properly) * public Playlist(String title, List<Song> listofsongs , int rankPoints) – Constructor, When object is created it is expecting string, list of songs and int. * public String getTitle() - Used to get title. * public void setTitle(String title) – Used to set title. * public List<Song> getListofsongs() - Used to get list of songs. * public void setListofsongs(List<Song> listofsongs) - Used to set list of songs. * public int getRankPoints()- Used to get rankPoints. * public void setRankPoints(int rankPoints) Used to set rankPoints. |

|  |
| --- |
| Class name: Song**(Server and client)** |
| Description: This class is a blue-print for Song objects. |
| Attributes: String title, String artist, String album, String genre. |
| Functions:   * public Song() - Empty constructor (requirement for jersey to work properly) * public Song( String title, String artist, String album, String genre) – Constructor, when object is created it is expecting 4 strings * public String getTitle() – used to get title. * public void setTitle(String title) – used to set title. * public String getArtist() – used to get artist. * public void setArtist(String artist) - used to set artist. * public String getAlbum() – used to get album. * public void setAlbum(String album) - used to set album. * public String getGenre() – used to get genre. * public void setGenre(String genre) – used to set genre. |

|  |
| --- |
| Class name: Statistic**(Server and client)** |
| Description: This class is a blue-print for Statistic objects. |
| Attributes: Int views, Int like, Int dislike. |
| Functions:   * public Statistic() - Empty constructor (requirement for jersey to work properly) * public Statistic(int views, int like, int dislike) – Constructor, when object Is created it is expecting 3 int s to be passed. * public int getViews() – used to get views. * public void setViews(int views) - used to set views. * public int getLike() – used to get likes. * public void setLike(int like) – used to set likes. * public int getDislike() – used to getdislikes. * public void setDislike(int dislike) – used to set dislikes. |

|  |
| --- |
| Class name: Top10Song**(Server and client)** |
| Description: This class is a blue-print for Top10Song objects. This class is the same as song but it has one more attribute which is rank. This class is only used to generate and to display top10 songs. |
| Attributes: String title, String artist, String album, String genre, int rank. |
| Functions:   * public Song() - Empty constructor (requirement for jersey to work properly) * public Song( String title, String artist, String album, String genre) – Constructor, when object is created it is expecting 4 strings * public String getTitle() – used to get title. * public void setTitle(String title) – used to set title. * public String getArtist() – used to get artist. * public void setArtist(String artist) - used to set artist. * public String getAlbum() – used to get album. * public void setAlbum(String album) - used to set album. * public String getGenre() – used to get genre. * public void setGenre(String genre) – used to set genre. * public int getRank() – used to get rank. * public void setRank(int rank) – used to set rank. |

|  |
| --- |
| Class name: Top10Song**(client)** |
| Description: This class is a driver program. This class is responsible for interaction with user. This class is mainly connecting to server and handling response from it. |
| Attributes: |
| Functions:   * votePlaylist(String nickname) - This function allows user to like or dislike playlist. * searchPlaylistByNickname() - This function allows user to enter nickname of creator and search for his playlist * setPlaylistView(String nickname) - This function allowed me to keep track of views * getAllPlaylists() - This function is displaying list of all playlists * makeSpace() - Used this function to split text in console to make it easier to read for user. * ToUpperCase(String s) - This function is checking if fist letter of string is upper case, if not then set it to upper case |



# A table of client functions (vi)

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Input | Output | Purpose |
| votePlaylist(String nickname) | answer(Y/N), title, Like/Dislike | Appropriate messages | This function allows user to like or dislike playlist. |
| searchPlaylistByNickname() | Nickname, | Playlist | This function allows user to enter nickname of creator and search for his playlist |
| setPlaylistView(String nickname) | None | None | This function allowed me to keep track of views |
| getAllPlaylists() | None | List of playlists | This function is displaying list of all playlists |
| makeSpace() | None | --------------- | Used this function to split text in console to make it easier to read for user. |
| ToUpperCase(String s) | String | String | This function is checking if fist letter of string is upper case, if not then set it to upper case |

# A table of service endpoints (vii)

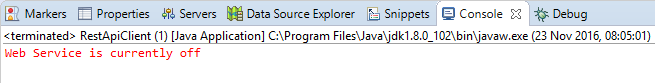
Main: http://localhost:8080/MusicExchange/webapi/

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Input | Output | Purpose |
| /accounts/(@GET) | None | List of Accounts | This endpoint is used to get all accounts. |
| /accounts/{nickname}(@POST) | Account object and string in url | Account | This endpoint is used to add accounts. |
| /accounts/{nickname}/{password}(@GET) | Two strings in url | Account | This endpoint is used validate nickname and password while user is logging in. |
| /playlists/(@GET) | None | List of Playlists | This endpoint is used to get all playlists. |
| /playlists/top10(@GET) | None | List of Playlists | This endpoint is used to get top 10 playlists. |
| /playlists/top10songs(@GET) | None | List of top10songs | This endpoint is used to get top 10 songs. |
| /playlists/{nickname}(@GET) | String in url | Playlist | This endpoint is used to get playlist by nickname. |
| /playlists/{nickname}(@POST) | String in url and playlist | Playlist | This endpoint is used to add playlists. |
| /playlists/{nickname}(@DELETE) | String in url | None | This endpoint is used to delete playlists. |
| /playlists/{nickname}/edit(@POST) | String in url, Song | Song | This endpoint is used to add song to playlist. |
| /playlists/{nickname}/edit/{songindex}  (@DELETE) | String and int in url. | None | This endpoint is used to delete songs from playlist. |
| /statistics/{nickname}(@GET) | String in url | Statistic | This endpoint is used to get statistic. |
| /statistic/view(@PUT) | None | None | This endpoint is used to increase view by 1 in all playlist statistics. |
| /statistic/{nickname}/view(@PUT) | String in url | None | This endpoint is used to increase view by 1 in playlist statistics. |
| /statistic/rate/{nickname}(@POST) | Two strings in url and Statistic | None | This endpoint is used to save like or dislike. |

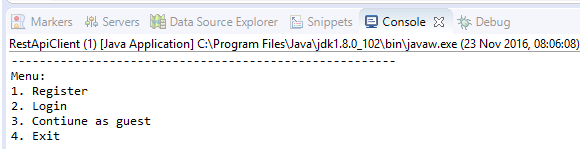
# A user guide for your client application(viii)

Client is a menu driven java program. To pick option user must follow messages.

Error message when server is off

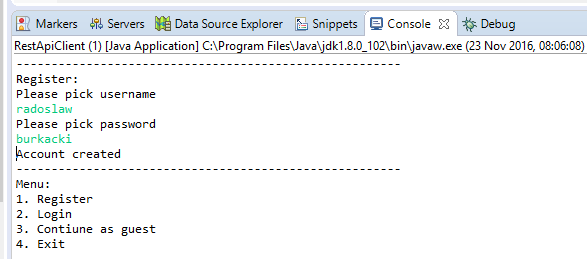


After program is started user will see this:



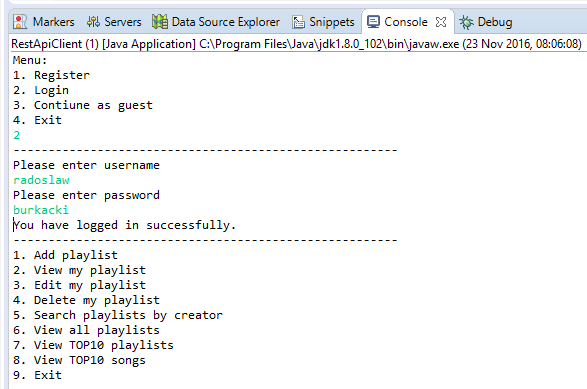
To **register** user must enter 1.

User will be asked to enter nickname and password.

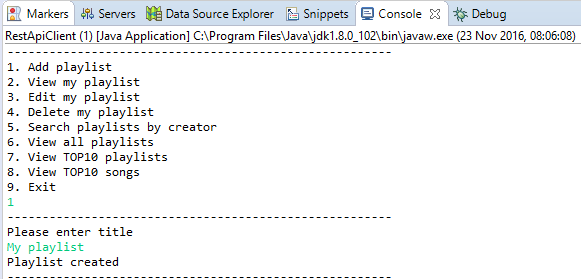


Now user should be able to **login** using chosen credentials.

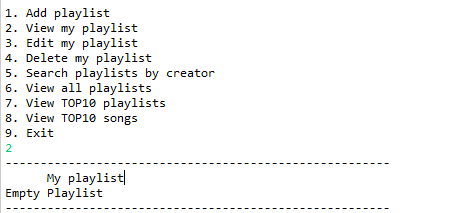
If credentials were correct, then new menu should be displayed.



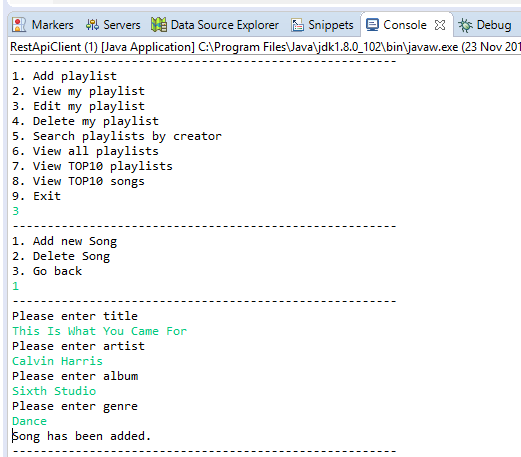
To **create playlist**, user must enter title of playlist.



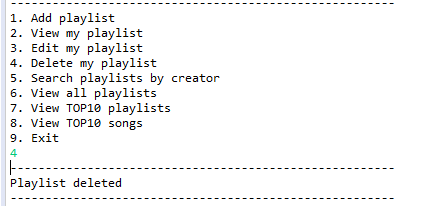
To **view my playlist** user must pick this option and my playlist will be displayed.



To **add new song** into playlist, user must go into **edit playlist** section then select add song. Enter title, artist, album and genre.

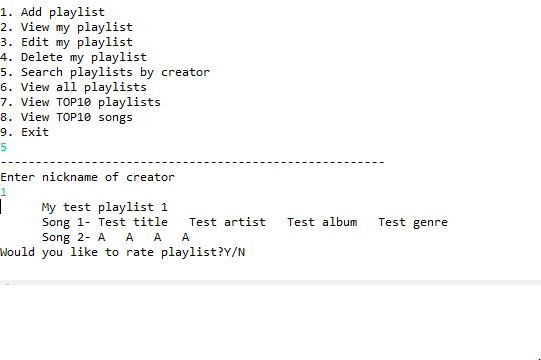


To **delete Playlist** user must pick that option and playlist will be deleted.

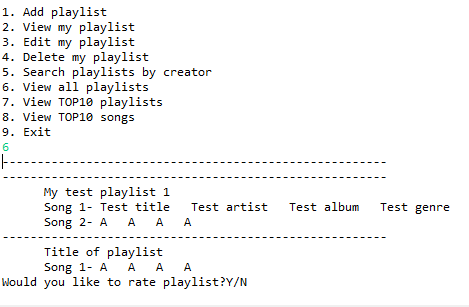


*I have created two playlists and 3 songs on two different customers(1,2) to test option with search playlists by nickname, viewing all playlists, viewing top10 playlists and viewing top10 songs.*

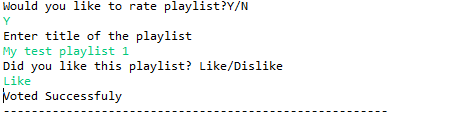
To search playlist by nickname user must pick 5th option and enter nickname of creator.



To **view all playlist** user must pick 6th option.

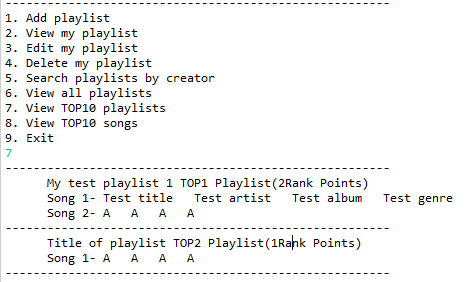


Customer has option to rate playlist to rate enter Y, title of playlist and like or dislike.



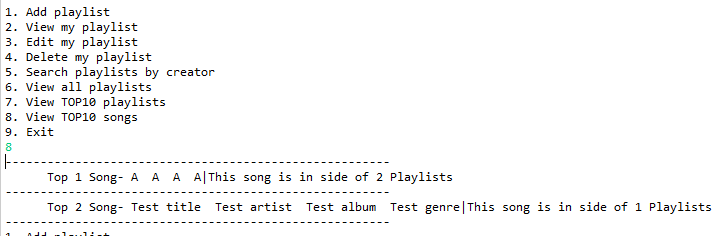
To view top 10 playlist user must pick option number 7.

Top 10 will be displayed.

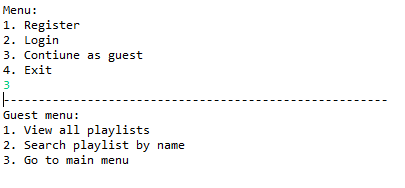


To view top 10 song user must pick option number 8.

Top 10 songs will be displayed.



Guests are also allowed to use the program but they will have only access to two options.



# A paragraph from each project collaborator(ix)

All responsibilities were on myself as I have done this whole project on my own. All project products were created by me.

# PDP(x)

This project has contributed to my Personal Development Plan a lot. Before I started this project, I knew java well but the issue was that I had no idea how to use it in web development, after doing this module I have learned how to use java in web development. I have learned a lot about web services. I enjoyed learning about web services.

# A section of outstanding issues for the project(xi)

This project is very complex, as I have done it only on my own, because of time limits I might have not found some bugs, although I have tested all the functions several times and all known bugs were fixed. If I had some more time I would add more functionality into the web service and create a GUI client.

# Source code (xii)

# AccountResource(server):

package org.radoslawburkacki.MusicExchange.resources;

import java.util.List;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import org.radoslawburkacki.MusicExchange.model.Account;

import org.radoslawburkacki.MusicExchange.service.AccountService;

@Path("/accounts")// set main path

@Consumes(MediaType.APPLICATION\_JSON) // consume json

@Produces(MediaType.APPLICATION\_JSON) // produce json

public class AccountResource {

AccountService accountservice = new AccountService(); // creating new instance

@GET // get function

public List<Account> getAllAccounts(){ // this function will return a list of account objects

return accountservice.getAllAccounts(); // return what ever was returned by function .getallaccounts

}

@POST// post function

@Path("/{nickname}") // path for this function is accounts/{nickname} | nickname is a variable!

public Response addAccount(@PathParam("nickname") String nickname,Account acc){ // this function will return a response/ it also consumes param and saves it to nickname and it consumes an object of account

Account newAccount = accountservice.addAccount(nickname, acc);

if(newAccount.getNickname().equals("failed") & newAccount.getPassword().equals("failed")){

return Response.status(Status.CONFLICT)

.entity(newAccount)

.build();

}

else { return Response.status(Status.CREATED)

.entity(newAccount)

.build();

}

//return accountservice.addAccount(nickname,acc);

}

@GET // get function

@Path("/{nickname}/{password}") // path / accounts/{nickname}/{password} | nickname and password is a variable

public Account validateAccount(@PathParam("nickname")String nickname, @PathParam("password") String password){ // save path params

return accountservice.validateAccount(nickname , password);

}

}

# PlaylistResource(Server):

package org.radoslawburkacki.MusicExchange.resources;

import java.util.List;

import java.util.Map;

import javax.ws.rs.Consumes;

import javax.ws.rs.DELETE;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import org.radoslawburkacki.MusicExchange.model.Account;

import org.radoslawburkacki.MusicExchange.model.Playlist;

import org.radoslawburkacki.MusicExchange.model.Song;

import org.radoslawburkacki.MusicExchange.model.Statistic;

import org.radoslawburkacki.MusicExchange.model.Top10Song;

import org.radoslawburkacki.MusicExchange.service.PlaylistService;

@Path("/playlists") // main path

@Consumes(MediaType.APPLICATION\_JSON) // consumes json

@Produces(MediaType.APPLICATION\_JSON) // produes json

public class PlaylistResource {

PlaylistService playlistservice = new PlaylistService();

@GET

public List<Playlist> getAllPlaylists(){

return playlistservice.getAllPlaylists();

}

@GET

@Path("/top10")

public List<Playlist> getTop10Playlists(){

return playlistservice.getTop10Playlists();

}

@GET

@Path("/top10songs")

public List<Top10Song> getTop10Songs(){

return playlistservice.getTop10Songs();

}

@GET// get one playlist by user nickname

@Path("/{nickname}")

public Response getPlaylist(@PathParam("nickname") String nickname){

Playlist p = playlistservice.getPlaylist(nickname);

if(playlistservice.getPlaylist(nickname)==null){

return Response.status(Status.NO\_CONTENT)

.build();

}

else {return Response.status(Status.FOUND)

.entity(p)

.build();

}

}

@POST

@Path("/{nickname}")

public Response addPlaylist(@PathParam("nickname") String nickname,Playlist playlist){

Playlist newPlaylist = playlistservice.addPlaylist(nickname, playlist);

if(newPlaylist.getTitle().equals("customer already has a playlist error")){

return Response.status(Status.CONFLICT)

.entity(newPlaylist)

.build();

}

else if(newPlaylist.getTitle().equals("name already in use")){

return Response.status(Status.BAD\_REQUEST)

.entity(newPlaylist)

.build();

}

else { return Response.status(Status.CREATED)

.entity(newPlaylist)

.build();

}

}

@DELETE // delete function

@Path("/{nickname}")

public void deletePlaylist(@PathParam("nickname")String nickname){

playlistservice.deletePlatlist(nickname);

}

@POST// post function

@Path("/{nickname}/edit")

public Response addSongToPlaylist(@PathParam("nickname") String nickname,Song s){

Song newSong = playlistservice.addSongToPlaylist(nickname,s);

if(newSong.getTitle().equals("song is already in playlist")){

return Response.status(Status.CONFLICT)

.entity(newSong)

.build();

}

else { return Response.status(Status.CREATED)

.entity(newSong)

.build();

}

}

@DELETE // delete function

@Path("/{nickname}/edit/{songindex}")

public void removeSongFromPlaylist(@PathParam("nickname")String nickname,@PathParam("songindex")int songindex){

playlistservice.deleteSong(nickname, songindex);

}

}

# StatisticResource(Server):

package org.radoslawburkacki.MusicExchange.resources;

import javax.ws.rs.Consumes;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.PUT;

import javax.ws.rs.Path;

import javax.ws.rs.PathParam;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import javax.ws.rs.core.Response.Status;

import org.radoslawburkacki.MusicExchange.model.Statistic;

import org.radoslawburkacki.MusicExchange.service.StatisticsService;

@Path("/statistics")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

public class StatisticsReource {

StatisticsService statisticservice = new StatisticsService();

@GET

@Path("/{nickname}")

public Statistic getPlaylistStatistics(@PathParam("nickname") String nickname){

return statisticservice.getPlaylistStatistic(nickname);

}

@PUT

@Path("/view")

public void setAllPlaylistView(){

statisticservice.increaseViewOfAllPlaylists();

}

@PUT

@Path("/{nickname}/view")

public void setPlaylistView(@PathParam("nickname") String nickname, Statistic s){

statisticservice.increaseViewOfOnePlalist(nickname);

}

@POST

@Path("/rate/{title}/{nickname}")

public Response ratePlaylist(@PathParam("title") String title, @PathParam("nickname") String nickname, Statistic s){

String response = null;

if(s.getLike()==1){

response =statisticservice.likePlaylist(title,nickname,s);

}

else if(s.getDislike()==1){

response =statisticservice.dislikePlaylist(title,nickname,s);

}

if(response.equals("cannot vote for own playlist")){

return Response.status(Status.CONFLICT)

.build();

}

else if(response.equals("error")){

return Response.status(Status.NOT\_FOUND)

.build();

}

else{

return Response.status(Status.CREATED)

.build();

}

}

}

# AccountService(Server):

package org.radoslawburkacki.MusicExchange.service;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import org.radoslawburkacki.MusicExchange.database.DatabaseClass;

import org.radoslawburkacki.MusicExchange.model.Account;

public class AccountService {

private Map<String, Account> accounts = DatabaseClass.getAccounts(); // create new map which equals to what ever is returned by function

public AccountService(){

Account acc = new Account();

}

public List<Account> getAllAccounts(){

return new ArrayList<Account>(accounts.values());

}

public Account addAccount(String nickname, Account acc){

if (accounts.containsKey(nickname)){

acc.setNickname("failed");

acc.setPassword("failed");

}

else if(!accounts.containsKey(nickname))

accounts.put(nickname, acc);

return acc;

}

public Account validateAccount(String nickname, String password){

Account acc = new Account(nickname,password);

Account a= new Account();

if (accounts.containsKey(nickname)){

a = accounts.get(nickname);

if(acc.getNickname().equals(a.getNickname()) && acc.getPassword().equals(a.getPassword())){

acc.setNickname("true");

acc.setPassword("true");

}

}

return acc;

}

}

# PlaylistService(Server):

package org.radoslawburkacki.MusicExchange.service;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.SortedSet;

import java.util.TreeSet;

import org.radoslawburkacki.MusicExchange.database.DatabaseClass;

import org.radoslawburkacki.MusicExchange.model.Playlist;

import org.radoslawburkacki.MusicExchange.model.Song;

import org.radoslawburkacki.MusicExchange.model.Statistic;

import org.radoslawburkacki.MusicExchange.model.Top10Song;

public class PlaylistService {

private Map<String, Playlist> playlists = DatabaseClass.getPlaylists();

private Map<String, Statistic> playliststatistics = DatabaseClass.getPlaylistStatistics();

private List<Song> songdb = DatabaseClass.getsongdb();

public List<Playlist> getAllPlaylists(){

return new ArrayList<Playlist>(playlists.values());

}

public Playlist getPlaylist(String nickname){

return playlists.get(nickname);

}

public List<Playlist> getTop10Playlists(){

Map<String, Integer> playlistRankPoints = new HashMap<String, Integer>();

for ( String key : playliststatistics.keySet() ) {

int like = playliststatistics.get(key).getLike();

int dislike = playliststatistics.get(key).getDislike();

int views = playliststatistics.get(key).getViews()/2;

int points = (like - dislike) + ( views);

playlistRankPoints.put(key, points);

}

playlistRankPoints = sortTop10(playlistRankPoints);

List < Integer> list = new ArrayList<Integer>();

for(String key : playlistRankPoints.keySet()){

list.add(playlistRankPoints.get(key));

}

Map<String, Integer> Top10Playlists = new HashMap<String, Integer>();

if(list.size() >= 11){

int i = list.size();

for(int j =0; j <10 ; j++){

i--;

for(String key : playlistRankPoints.keySet()){

if(playlistRankPoints.get(key) == list.get(i)){

Top10Playlists.put(key, playlistRankPoints.get(key));

}

}

}

}

else if (list.size() < 11){

Top10Playlists = playlistRankPoints;

}

List<Playlist> top10 = new ArrayList<Playlist>();

for(String key : Top10Playlists.keySet()){

playlists.get(key).setRankPoints(Top10Playlists.get(key));

top10.add(playlists.get(key));

}

return top10;

}

public static <String, Integer extends Comparable<? super Integer>> Map<String, Integer> sortTop10( Map<String, Integer> map ){

List<Map.Entry<String, Integer>> list =

new LinkedList<Map.Entry<String, Integer>>( map.entrySet() );

Collections.sort( list, new Comparator<Map.Entry<String, Integer>>()

{

public int compare( Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2 )

{

return (o2.getValue()).compareTo( o1.getValue() );

}

} );

Map<String, Integer> result = new LinkedHashMap<String, Integer>();

for (Map.Entry<String, Integer> entry : list)

{

result.put( entry.getKey(), entry.getValue() );

}

return result;

}

public Playlist addPlaylist(String nickname, Playlist playlist){

if (playlists.containsKey(nickname)){ //checking if user has a playlist

playlist.setTitle("customer already has a playlist error");

}

else if(!playlists.containsKey(nickname)){ // if user doesnt have a playlist then...

for(Playlist s: playlists.values()){ // loop through playlists

if(s.getTitle().equals(playlist.getTitle())){ // if any playlists title is equal to title of current playlist then

playlist.setTitle("name already in use"); // set name of current playlist to "name already in use"

}

}

if((!playlist.getTitle().equals("custnomer already has a playlist error")) && (!playlist.getTitle().equals("name already in use"))){

playliststatistics.put(nickname, new Statistic());

playlists.put(nickname, playlist);

}

}

return playlist;

}

public Song addSongToPlaylist(String nickname, Song s){ // receiving nickname of user and song (title, artist, album genre)

boolean a = false;

for(Song song : songdb) {

if(song.getTitle().equals(s.getTitle()) && song.getArtist().equals(s.getArtist()) && song.getAlbum().equals(s.getAlbum()) && song.getGenre().equals(s.getGenre())){

a = true;

}

}

if (a){

}

else if(!a){

songdb.add(s);

}

for(int i = 0; i < playlists.get(nickname).getListofsongs().size();i++){

if(playlists.get(nickname).getListofsongs().get(i).getTitle().equals(s.getTitle()) && playlists.get(nickname).getListofsongs().get(i).getArtist().equals(s.getArtist()) && playlists.get(nickname).getListofsongs().get(i).getAlbum().equals(s.getAlbum())&& playlists.get(nickname).getListofsongs().get(i).getGenre().equals(s.getGenre())){

s.setTitle("song is already in playlist");

}

}

if(!s.getTitle().equals("song is already in playlist")){

List<Song> songli = new ArrayList<Song>();

songli = playlists.get(nickname).getListofsongs();

songli.add(s);

playlists.get(nickname).setListofsongs(songli);

}

return s;

}

public List<Top10Song> getTop10Songs(){

List <Top10Song> top10songs = new ArrayList<Top10Song>();

for(int i =0; i < songdb.size(); i++){// loop through song database

Top10Song s= new Top10Song();

s.setTitle(songdb.get(i).getTitle()); s.setArtist(songdb.get(i).getArtist()); s.setAlbum(songdb.get(i).getAlbum()); s.setGenre(songdb.get(i).getGenre()); s.setRank(0);

for(String key : playlists.keySet()){// loop through playlists

for(int j=0; j < playlists.get(key).getListofsongs().size(); j++){// loop through songs inside of a playlist

if(playlists.get(key).getListofsongs().get(j).getTitle().equals(s.getTitle()) && playlists.get(key).getListofsongs().get(j).getArtist().equals(s.getArtist()) && playlists.get(key).getListofsongs().get(j).getAlbum().equals(s.getAlbum()) && playlists.get(key).getListofsongs().get(j).getGenre().equals(s.getGenre())){

s.setRank(s.getRank()+1);

}

}

}

top10songs.add(s);

}

int j;

boolean flag = true; // set flag to true to begin first pass

Top10Song temp = new Top10Song();

while ( flag )

{

flag= false; //set flag to false awaiting a possible swap

for( j=0; j < top10songs.size() -1; j++ )

{

if ( top10songs.get(j).getRank() < top10songs.get(j+1).getRank() )

{

temp = top10songs.get(j);

top10songs.add(j, top10songs.get(j+1));

top10songs.remove(j+1);

top10songs.add(j+1, temp);

flag = true;

}

}

}

List <Top10Song> top10songss = new ArrayList<Top10Song>();

if(top10songs.size() > 10){

for(int i =0; i < 10; i++){

top10songss.add(top10songs.get(i));

}

return top10songss;

}

return top10songs;

}

public void deleteSong(String nickname, int songindex){

playlists.get(nickname).getListofsongs().remove(songindex);

}

public void deletePlatlist(String nickname) {

playlists.remove(nickname);

playliststatistics.remove(nickname);

}

}

# StatisticService(Server):

package org.radoslawburkacki.MusicExchange.service;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.radoslawburkacki.MusicExchange.database.DatabaseClass;

import org.radoslawburkacki.MusicExchange.model.Playlist;

import org.radoslawburkacki.MusicExchange.model.Statistic;

public class StatisticsService {

private Map<String, Statistic> playliststatistics = DatabaseClass.getPlaylistStatistics();

private Map<String, Playlist> playlists = DatabaseClass.getPlaylists();

public Statistic getPlaylistStatistic(String nickname){

return playliststatistics.get(nickname);

}

public void increaseViewOfOnePlalist(String nickname) {

playliststatistics.get(nickname).setViews(playliststatistics.get(nickname).getViews()+1);

}

public void increaseViewOfAllPlaylists(){

for ( String key : playliststatistics.keySet() ) {

playliststatistics.get(key).setViews(playliststatistics.get(key).getViews()+1);

}

}

public String likePlaylist(String title, String nickname, Statistic s){

String playlistOwner="";

for ( String key : playlists.keySet() ) {// loop through playlist

if(playlists.get(key).getTitle().equals(title)){// if playlist title is equal to title

playlistOwner = key; // set playlistOwner equal to key (key is nickname of playlist owner)

}

}

for ( String key : playliststatistics.keySet() ) {

if(key.equals(playlistOwner)){

if(playlistOwner.equals(nickname)){// if creator of playlist likes own playlist then do nothing

return "cannot vote for own playlist";

}

else {// if anyone else except creator of playlist is liking then...

playliststatistics.get(key).setLike(playliststatistics.get(key).getLike()+1); // add like

return "success";

}

}

}

return "error";

}

public String dislikePlaylist(String title, String nickname, Statistic s){

String playlistOwner="";

for ( String key : playlists.keySet() ) {// loop through playlist

if(playlists.get(key).getTitle().equals(title)){// if playlist title is equal to title

playlistOwner = key; // set playlistOwner equal to key (key is nickname of playlist owner)

}

}

for ( String key : playliststatistics.keySet() ) {

if(key.equals(playlistOwner)){

if(playlistOwner.equals(nickname)){// if creator of playlist likes own playlist then do nothing

return "cannot vote for own playlist";

}

else {// if anyone else except creator of playlist is liking then...

playliststatistics.get(key).setDislike(playliststatistics.get(key).getDislike()+1); // add like

return "success";

}

}

}

return "error";

}

}

# DatabaseClass(Server):

package org.radoslawburkacki.MusicExchange.database;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import org.radoslawburkacki.MusicExchange.model.Account;

import org.radoslawburkacki.MusicExchange.model.Playlist;

import org.radoslawburkacki.MusicExchange.model.Song;

import org.radoslawburkacki.MusicExchange.model.Statistic;

public class DatabaseClass {

private static Map<String, Playlist> playlists = new HashMap<String, Playlist>();

private static Map<String, Account> accounts = new HashMap<String, Account>();

private static Map<String, Statistic> playliststatistics = new HashMap<String, Statistic>();

private static List<Song> songdb = new ArrayList<Song>();

public static Map<String,Account> getAccounts (){

return accounts;

}

public static Map<String,Playlist> getPlaylists (){

return playlists;

}

public static Map<String, Statistic> getPlaylistStatistics (){

return playliststatistics;

}

public static List<Song> getsongdb(){

return songdb;

}

}

# Account(Server):

package org.radoslawburkacki.MusicExchange.model;

public class Account {

String nickname;

String password;

public Account(){

}

public Account( String nickname, String password) {

this.nickname = nickname;

this.password = password;

}

public String getNickname() {

return nickname;

}

public void setNickname(String nickname) {

this.nickname = nickname;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

# Playlist(Serve):

package org.radoslawburkacki.MusicExchange.model;

import java.util.ArrayList;

import java.util.List;

public class Playlist {

String title;

List<Song> listofsongs = new ArrayList<Song>();

int rankPoints;

public Playlist(){

}

public Playlist(String title, List<Song> listofsongs , int rankPoints) {

this.title = title;

this.listofsongs = listofsongs;

this.rankPoints = rankPoints;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public List<Song> getListofsongs() {

return listofsongs;

}

public void setListofsongs(List<Song> listofsongs) {

this.listofsongs = listofsongs;

}

public int getRankPoints() {

return rankPoints;

}

public void setRankPoints(int rankPoints) {

this.rankPoints = rankPoints;

}

}

# Song(Server):

package org.radoslawburkacki.MusicExchange.model;

public class Song {

String title;

String artist;

String album;

String genre;

public Song(){

}

public Song( String title, String artist, String album, String genre) {

this.title = title;

this.artist = artist;

this.album = album;

this.genre = genre;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getArtist() {

return artist;

}

public void setArtist(String artist) {

this.artist = artist;

}

public String getAlbum() {

return album;

}

public void setAlbum(String album) {

this.album = album;

}

public String getGenre() {

return genre;

}

public void setGenre(String genre) {

this.genre = genre;

}

}

# Statistic(Server):

package org.radoslawburkacki.MusicExchange.model;

public class Statistic {

int views=0;

int like=0;

int dislike =0;

public Statistic(){

}

public Statistic(int views, int like, int dislike) {

super();

this.views = views;

this.like = like;

this.dislike = dislike;

}

public int getViews() {

return views;

}

public void setViews(int views) {

this.views = views;

}

public int getLike() {

return like;

}

public void setLike(int like) {

this.like = like;

}

public int getDislike() {

return dislike;

}

public void setDislike(int dislike) {

this.dislike = dislike;

}

}

# Top10Song(Server):

package org.radoslawburkacki.MusicExchange.model;

public class Top10Song{

String title;

String artist;

String album;

String genre;

int rank;

public Top10Song(){

}

public Top10Song(String title, String artist, String album, String genre, int rank) {

super();

this.title = title;

this.artist = artist;

this.album = album;

this.genre = genre;

this.rank = rank;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getArtist() {

return artist;

}

public void setArtist(String artist) {

this.artist = artist;

}

public String getAlbum() {

return album;

}

public void setAlbum(String album) {

this.album = album;

}

public String getGenre() {

return genre;

}

public void setGenre(String genre) {

this.genre = genre;

}

public int getRank() {

return rank;

}

public void setRank(int rank) {

this.rank = rank;

}

}

# RestApiClient(Client):

package org.radoslawburkacki.client;

import java.io.IOException;

import java.net.HttpURLConnection;

import java.net.URL;

/\*\* Radoslaw Burkacki B00309449

Web services coursework

21/11/2016

\*/

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.Entity;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.GenericType;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import org.radoslawburkacki.MusicExchange.model.Account;

import org.radoslawburkacki.MusicExchange.model.Playlist;

import org.radoslawburkacki.MusicExchange.model.Song;

import org.radoslawburkacki.MusicExchange.model.Statistic;

import org.radoslawburkacki.MusicExchange.model.Top10Song;

public class RestApiClient {// creating new class

public static void main(String[] args) {// calling main function

// this section is about creating url targets //////////////////////////////////////////

Client client = ClientBuilder.newClient(); // creating new instance of client

WebTarget baseTarget = client.target("http://localhost:8080/MusicExchange/webapi/"); // setting main url target

WebTarget accountsTarget = baseTarget.path("accounts"); // setting accounts target

WebTarget nicknameTarget = accountsTarget.path("{nickname}"); // setting nickname target (nickname is a variable)

WebTarget accpassword = nicknameTarget.path("{password}"); // setting password target (password is a variable)

WebTarget playlistsTarget = baseTarget.path("playlists"); // setting playlists target

WebTarget playlisttop10Target = playlistsTarget.path("top10"); // setting top10 target

WebTarget songtop10 = playlistsTarget.path("top10songs"); // setting top10songs target

WebTarget plnickTarget = playlistsTarget.path("{nickname}"); // setting nickname target (nickname is a variable)

WebTarget editplaylistTarget = plnickTarget.path("edit"); // setting edit target

WebTarget removesongTarget = editplaylistTarget.path("{songindex}"); // setting songindex target (songindex is a variable)

WebTarget statisticsTarget = baseTarget.path("statistics"); // setting statistics target

WebTarget rateTarget = statisticsTarget.path("rate"); // setting rate target

// this section is about creating url targets //////////////////////////////////////////

Scanner sc = new Scanner (System.in); // creating new instance of scanner

int menu =0; // creating new variable and setting its value to 0

try {

String strUrl = "http://localhost:8080/MusicExchange/webapi/accounts/";

URL url = new URL(strUrl);

HttpURLConnection urlConn = (HttpURLConnection) url.openConnection();

urlConn.connect();

while(menu != 4){ // starting while loop, this loop will exit when manu will be equal to 4

makeSpace(); // calling function (this function is simply making space in console)

System.out.println("Menu:"); // printing message

System.out.println("1. Register"); // printing message

System.out.println("2. Login"); // printing message

System.out.println("3. Contiune as guest"); // printing message

System.out.println("4. Exit"); // printing message

menu=sc.nextInt(); // whatever user enters is saved to menu variable

switch (menu){ // starting switch statement and saying at which variable it needs to look

case 1: // starting case 1

makeSpace();// calling function (this function is simply making space in console)

String nickname,password; // creating two new variables nickname and password

System.out.println("Register:"); //printing message

sc.nextLine();

System.out.println("Please pick username"); // printing message

nickname = sc.nextLine();

System.out.println("Please pick password"); // printing message

password = sc.nextLine();

Account newAccount = new Account(nickname,password); // creating new instance of Account and passing two variables into it

Response postResponse;

postResponse = nicknameTarget // creating connection to web service and saving response to in into postResponse

.resolveTemplate("nickname", newAccount.getNickname()) // resolving template, passing variable into it

.request(MediaType.APPLICATION\_JSON) // requesting json response

.post(Entity.json(newAccount)); // calling post function on web service, sending account(converting it into json).

if(postResponse.getStatus() == 409) // if response status code is equal to 409 then...

System.out.println("Account already exists");

if(postResponse.getStatus() == 201) // if response status code is equal to 201 then...

System.out.println("Account created");

break; //breaks out of case 1

case 2: // starting case 2

makeSpace(); // calling function

System.out.println("Please enter username");

sc.nextLine();

nickname = sc.nextLine();

if(nickname.equals("")){ // if user havent enterd anything into nickname

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter nickname"); /\*\* Displaying message \*/

nickname=sc.nextLine(); // save input again

}while(nickname.equals("")); // end while loop when nickname is not empty

}

System.out.println("Please enter password");

password = sc.nextLine();

if(password.equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter password"); /\*\* Displaying message \*/

password=sc.nextLine();

}while(password.equals(""));

}

Account acc = accpassword // create connection with server, save esponse to acc

.resolveTemplate("nickname", nickname) // pass nickname

.resolveTemplate("password", password) // pass password

.request(MediaType.APPLICATION\_JSON) // request json response

.get(Account.class); // call get function

if(acc.getNickname().equals("true") & acc.getPassword().equals("true")){

System.out.println("You have logged in successfully.");

while(menu != 9){ // start while loop

makeSpace();

// menu

System.out.println("1. Add playlist");

System.out.println("2. View my playlist");

System.out.println("3. Edit my playlist");

System.out.println("4. Delete my playlist");

System.out.println("5. Search playlists by creator");

System.out.println("6. View all playlists");

System.out.println("7. View TOP10 playlists");

System.out.println("8. View TOP10 songs");

System.out.println("9. Exit");

menu = sc.nextInt();

switch (menu){

case 1: // Create new playlist

makeSpace();

String title;

System.out.println("Please enter title");

sc.nextLine();

title = sc.nextLine();

if(title.equals("")){ /\*\* if title is empty then... \*/

do{ /\*\* start do while loop \*/

System.out.println("Please enter title"); /\*\* Displaying message \*/

title = sc.nextLine(); /\*\* set title equal to input from user \*/

}while(title.equals("")); /\*\* end while loop when title is not empty \*/

}

if(!Character.isUpperCase(title.charAt(0))){ /\*\* if index 0 of title is lower case then...\*/

title = ToUpperCase(title); /\*\* set title equal to(call function ToUpperCase and pass title into it) \*/

}

Playlist newPlaylist = new Playlist();

newPlaylist.setTitle(title);

Response postResponse1 = plnickTarget// create connection with ws, save response

.resolveTemplate("nickname", nickname) // pass nickname

.request(MediaType.APPLICATION\_JSON)

.post(Entity.json(newPlaylist)); // call post function

if(postResponse1.getStatus() == 409)

System.out.println("You already own a playlist");

if(postResponse1.getStatus() == 400)

System.out.println("Name already in use");

if(postResponse1.getStatus() == 201)

System.out.println("Playlist created");

break; // break out of case 1

case 2: // View My playlist

makeSpace();

boolean empty = true;

Response rr = plnickTarget // start new connection, save response

.resolveTemplate("nickname", nickname) // pass nickname

.request(MediaType.APPLICATION\_JSON) // request json response

.get(); // call get function

if(rr.getStatus() == 302){

setPlaylistView(nickname);

Playlist p = rr.readEntity(Playlist.class);

System.out.println(" "+p.getTitle());

for(int i = 0; i < p.getListofsongs().size(); i++) {

empty=false;

System.out.println(" Song "+(i+1) +"- "+ p.getListofsongs().get(i).getTitle() +" "+ p.getListofsongs().get(i).getArtist() +" " + p.getListofsongs().get(i).getAlbum() +" "+ p.getListofsongs().get(i).getGenre());

}

if(empty){

System.out.println("Empty Playlist");

}

}

else if(rr.getStatus() == 204){

System.out.println("Playlist not found");

}

break;

case 3: // edit playlist

makeSpace();

// 1 need to check if user has a playlist if yes then display menu (add song, remove song,go back) if not then display error message

Response r = plnickTarget

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.get();

if(r.getStatus() == 204)

System.out.println("Playlist not found");

if(r.getStatus() == 302){

System.out.println("1. Add new Song");

System.out.println("2. Delete Song");

System.out.println("3. Go back");

menu =sc.nextInt();

switch(menu){

case 1:

makeSpace();

Song s = new Song();

System.out.println("Please enter title");

sc.nextLine();

s.setTitle(sc.nextLine());

System.out.println("Please enter artist");

s.setArtist(sc.nextLine());

System.out.println("Please enter album");

s.setAlbum(sc.nextLine());

System.out.println("Please enter genre");

s.setGenre(sc.nextLine());

if(s.getTitle().equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter title"); /\*\* Displaying message \*/

s.setTitle(sc.nextLine());

}while(s.getTitle().equals(""));

}

if(!Character.isUpperCase(s.getTitle().charAt(0))){ /\*\* if index 0 of s.getTitle() is lower case then...\*/

s.setTitle(ToUpperCase(s.getTitle()));

}

if(s.getArtist().equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter artist"); /\*\* Displaying message \*/

s.setArtist(sc.nextLine());

}while(s.getArtist().equals(""));

}

if(!Character.isUpperCase(s.getArtist().charAt(0))){ /\*\* if index 0 of s.getArtist() is lower case then...\*/

s.setArtist(ToUpperCase(s.getArtist()));

}

if(s.getAlbum().equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter album"); /\*\* Displaying message \*/

s.setAlbum(sc.nextLine());

}while(s.getAlbum().equals(""));

}

if(!Character.isUpperCase(s.getAlbum().charAt(0))){ /\*\* if index 0 of s.getAlbum() is lower case then...\*/

s.setAlbum(ToUpperCase(s.getAlbum()));

}

if(s.getGenre().equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please re-enter genre"); /\*\* Displaying message \*/

s.setGenre(sc.nextLine());

}while(s.getGenre().equals(""));

}

if(!Character.isUpperCase(s.getGenre().charAt(0))){ /\*\* if index 0 of s.getGenre() is lower case then...\*/

s.setGenre(ToUpperCase(s.getGenre()));

}

Response p1=editplaylistTarget

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.post(Entity.json(s));

if (p1.getStatus() == 409){

System.out.println("Song already in playlists.");

}

if (p1.getStatus() == 201){

System.out.println("Song has been added.");

}

break;

case 2:

makeSpace();

Response rrr = plnickTarget

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.get();

if(rrr.getStatus() == 302){

Playlist p = rrr.readEntity(Playlist.class);

System.out.println(" " + p.getTitle() );

for(int i = 0; i < p.getListofsongs().size(); i++) {

System.out.println(" Song "+(i+1)+"- " +p.getListofsongs().get(i).getTitle() + " " +

p.getListofsongs().get(i).getArtist() + " " +

p.getListofsongs().get(i).getAlbum() + " " +

p.getListofsongs().get(i).getGenre());

}

if(!p.getListofsongs().isEmpty()){

System.out.print("Which song would you like to delete?(Enter number)");

int songindex;

songindex = sc.nextInt()-1;

removesongTarget.resolveTemplate("nickname", nickname)

.resolveTemplate("songindex", songindex)

.request(MediaType.APPLICATION\_JSON)

.delete();

}

else{

System.out.println("Empty Playlist");

}

}

break;

}

}

break;

case 4:

makeSpace();

Response resp = plnickTarget

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.get();

plnickTarget.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.delete();

if (resp.getStatus() ==204){

System.out.println("No playlist found");

}

else{

System.out.println("Playlist deleted");

}

break;

case 5: //Search playlist by nickname

makeSpace();

if(searchPlaylistByNickname())

votePlaylist(nickname);

break;

case 6: // view all playlists

makeSpace();

if(getAllPlaylists())

votePlaylist(nickname);// make sure it works after function creation

break;

case 7:// view top10 playlists

List <Playlist>Top10Playlists= playlisttop10Target

.request(MediaType.APPLICATION\_JSON)

.get(new GenericType<List<Playlist>>() {});

for(int i =0; i < Top10Playlists.size(); i++){

System.out.println("-------------------------------------------------------");

System.out.println(" "+Top10Playlists.get(i).getTitle()+ " TOP"+(i+1)+" Playlist("+Top10Playlists.get(i).getRankPoints()+"Rank Points)");

for(int jj=0; jj < Top10Playlists.get(i).getListofsongs().size(); jj++){

System.out.println(" Song "+(jj+1)+"- " +Top10Playlists.get(i).getListofsongs().get(jj).getTitle() + " " +

Top10Playlists.get(i).getListofsongs().get(jj).getArtist() + " " +

Top10Playlists.get(i).getListofsongs().get(jj).getAlbum() + " " +

Top10Playlists.get(i).getListofsongs().get(jj).getGenre());

}

}

if(Top10Playlists.isEmpty()){

System.out.println("No playlists found");

}

break;

case 8:

List<Top10Song> top10songs = new ArrayList<Top10Song>();

top10songs = songtop10.request(MediaType.APPLICATION\_JSON)

.get(new GenericType<List<Top10Song>>() {});

if(top10songs.isEmpty()){

System.out.println("No songs found");

}

int i=0;

for(Top10Song song: top10songs){

System.out.println("-------------------------------------------------------");

System.out.println(" Top "+(i +1)+" Song- "+song.getTitle()+" "+song.getArtist()+" "+song.getAlbum()+" "+song.getGenre() +"|This song is in side of " + song.getRank() +" Playlists");

i++;

}

break;

}

}

}

else{

System.out.println("Wrong nickname/password");

}

break;

case 3:

int menu1 =0;

while(menu1 != 3){

makeSpace();

System.out.println("Guest menu:");

System.out.println("1. View all playlists");

System.out.println("2. Search playlist by name");

System.out.println("3. Go to main menu");

menu1 = sc.nextInt();

switch(menu1){

case 1:

getAllPlaylists();

break;

case 2:

searchPlaylistByNickname();

break;

}

}

break;

case 4:

System.out.println("Closing application");

break;

case 5:

break;

}

}

} catch (IOException e) {

System.err.println("Web Service is currently off");

}

}

public static void votePlaylist(String nickname){

Client client = ClientBuilder.newClient();

WebTarget baseTarget = client.target("http://localhost:8080/MusicExchange/webapi/");

WebTarget statisticsTarget = baseTarget.path("statistics");

WebTarget rateTarget = statisticsTarget.path("rate");

WebTarget titleTarget = rateTarget.path("{title}");

WebTarget nicknameTarget1 = titleTarget.path("{nickname}");

Scanner sc = new Scanner (System.in);

System.out.println("Would you like to rate playlist?Y/N");

String answer;

answer =sc.nextLine();

if(answer.equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Please enter title"); /\*\* Displaying message \*/

answer.equals(sc.nextLine());

}while(answer.equals(""));

}

if(!Character.isUpperCase(answer.charAt(0))){ /\*\* if index 0 of s.getTitle() is lower case then...\*/

answer = answer.substring(0, 1).toUpperCase() + answer.substring(1);

}

if(answer.equals("Y")){

System.out.println("Enter title of the playlist");

String ttitle;

ttitle = sc.nextLine();

if(ttitle.equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Enter title of the playlist"); /\*\* Displaying message \*/

ttitle=sc.nextLine();

}while(ttitle.equals(""));

}

if(!Character.isUpperCase(ttitle.charAt(0))){ /\*\* if index 0 of s.getTitle() is lower case then...\*/

ttitle = ttitle.substring(0, 1).toUpperCase() + ttitle.substring(1);

}

System.out.println("Did you like this playlist? Like/Dislike");

String rate;

rate = sc.nextLine();

if(rate.equals("")){

do{ /\*\* start do while loop \*/

System.out.println("Did you like this playlist? Like/Dislike"); /\*\* Displaying message \*/

rate=sc.nextLine();

}while(rate.equals(""));

}

if(!Character.isUpperCase(rate.charAt(0))){ /\*\* if index 0 of s.getTitle() is lower case then...\*/

rate = rate.substring(0, 1).toUpperCase() + rate.substring(1);

}

Statistic s = new Statistic();

if(rate.equals("Like"))

s.setLike(1);

else if(rate.equals("Dislike"))

s.setDislike(1);

Response r1=nicknameTarget1

.resolveTemplate("title", ttitle)

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.post(Entity.json(s));

if (r1.getStatus() ==409){

System.out.println("Cannot vote for own playlist");

}

else if(r1.getStatus() == 404){

System.out.println("Not Found");

}

else if(r1.getStatus() == 201){

System.out.println("Voted Successfuly");

}

}

}

public static boolean searchPlaylistByNickname(){

Client client = ClientBuilder.newClient();

WebTarget baseTarget = client.target("http://localhost:8080/MusicExchange/webapi/");

WebTarget playlistsTarget = baseTarget.path("playlists");

WebTarget plnickTarget = playlistsTarget.path("{nickname}");

Scanner sc = new Scanner(System.in);

System.out.println("Enter nickname of creator");

//sc.nextLine();

String nick;

nick = sc.nextLine();

setPlaylistView(nick);

boolean emptyy = true;

Response r1 = plnickTarget

.resolveTemplate("nickname", nick)

.request(MediaType.APPLICATION\_JSON)

.get();

if(r1.getStatus() == 302){

Playlist p = r1.readEntity(Playlist.class);

System.out.println(" "+p.getTitle());

for(int i = 0; i < p.getListofsongs().size(); i++) {

emptyy=false;

System.out.println(" Song "+(i+1)+"- " + p.getListofsongs().get(i).getTitle() +" "+ p.getListofsongs().get(i).getArtist() +" " + p.getListofsongs().get(i).getAlbum() +" "+ p.getListofsongs().get(i).getGenre());

}

if(emptyy){

System.out.println("Empty Playlist");

}

return true;

}

else if(r1.getStatus() == 204){

System.out.println("Playlist not found");

}

return false;

}

public static void setPlaylistView(String nickname){

Client client = ClientBuilder.newClient();

WebTarget baseTarget = client.target("http://localhost:8080/MusicExchange/webapi/");

WebTarget statisticsTarget = baseTarget.path("statistics");

WebTarget nicknameTarget = statisticsTarget.path(nickname);

WebTarget viewTarget = nicknameTarget.path("view");

Statistic s = new Statistic();

viewTarget

.resolveTemplate("nickname", nickname)

.request(MediaType.APPLICATION\_JSON)

.put(Entity.entity(s, MediaType.APPLICATION\_JSON));

}

public static boolean getAllPlaylists(){

Client client = ClientBuilder.newClient();

List<Playlist> pl = client.target("http://localhost:8080/MusicExchange/webapi/")

.path("playlists")

.request(MediaType.APPLICATION\_JSON)

.get(new GenericType<List<Playlist>>() {});

for(int i =0; i < pl.size(); i++){

System.out.println("-------------------------------------------------------");

System.out.println(" "+pl.get(i).getTitle());

for(int j=0; j < pl.get(i).getListofsongs().size(); j++){

System.out.println(" Song "+(j+1)+"- " +pl.get(i).getListofsongs().get(j).getTitle() + " " +

pl.get(i).getListofsongs().get(j).getArtist() + " " +

pl.get(i).getListofsongs().get(j).getAlbum() + " " +

pl.get(i).getListofsongs().get(j).getGenre());

}

}

if(!pl.isEmpty()){

Client client1 = ClientBuilder.newClient();

WebTarget baseTarget = client1.target("http://localhost:8080/MusicExchange/webapi/");

WebTarget statisticsTarget = baseTarget.path("statistics");

WebTarget viewTarget = statisticsTarget.path("view");

Statistic s = new Statistic();

viewTarget

.request(MediaType.APPLICATION\_JSON)

.put(Entity.entity(s, MediaType.APPLICATION\_JSON));

return true;

}

if(pl.isEmpty()){

System.out.println("-------------------------------------------------------");

System.out.println("No playlists found");

}

return false;

}

public static void makeSpace(){

for(int i=0; i < 1; i++){

System.out.println("-------------------------------------------------------");

}

}

public static String ToUpperCase(String s){

/\*\*it takes the string and sets the first letter of string to capital letter\*/

String ss = s; /\*\* Create new variable called s which is equal to title\*/

ss = ss.substring(0,1).toUpperCase() + ss.substring(1).toLowerCase();/\*\* setting first letter of s (index 0) to upper case and rest of them to lower\*/

s = ss; /\*\* set s equal to ss \*/

return s; /\*\* @return s\*/

}

}

# Account(client):

package org.radoslawburkacki.MusicExchange.model;

public class Account {

String nickname;

String password;

public Account(){

}

public Account( String nickname, String password) {

this.nickname = nickname;

this.password = password;

}

public String getNickname() {

return nickname;

}

public void setNickname(String nickname) {

this.nickname = nickname;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

# Playlist(client):

package org.radoslawburkacki.MusicExchange.model;

import java.util.ArrayList;

import java.util.List;

public class Playlist {

String title;

List<Song> listofsongs = new ArrayList<Song>();

int rankPoints;

public Playlist(){

}

public Playlist(String title, List<Song> listofsongs , int rankPoints) {

this.title = title;

this.listofsongs = listofsongs;

this.rankPoints = rankPoints;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public List<Song> getListofsongs() {

return listofsongs;

}

public void setListofsongs(List<Song> listofsongs) {

this.listofsongs = listofsongs;

}

public int getRankPoints() {

return rankPoints;

}

public void setRankPoints(int rankPoints) {

this.rankPoints = rankPoints;

}

}

# Song(client):

package org.radoslawburkacki.MusicExchange.model;

public class Song {

String title;

String artist;

String album;

String genre;

public Song(){

}

public Song( String title, String artist, String album, String genre) {

this.title = title;

this.artist = artist;

this.album = album;

this.genre = genre;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getArtist() {

return artist;

}

public void setArtist(String artist) {

this.artist = artist;

}

public String getAlbum() {

return album;

}

public void setAlbum(String album) {

this.album = album;

}

public String getGenre() {

return genre;

}

public void setGenre(String genre) {

this.genre = genre;

}

}

# Statistic(Client):

package org.radoslawburkacki.MusicExchange.model;

public class Statistic {

int views=0;

int like=0;

int dislike =0;

public Statistic(){

}

public Statistic(int views, int like, int dislike) {

super();

this.views = views;

this.like = like;

this.dislike = dislike;

}

public int getViews() {

return views;

}

public void setViews(int views) {

this.views = views;

}

public int getLike() {

return like;

}

public void setLike(int like) {

this.like = like;

}

public int getDislike() {

return dislike;

}

public void setDislike(int dislike) {

this.dislike = dislike;

}

}

# Top10Song(client):

package org.radoslawburkacki.MusicExchange.model;

public class Top10Song{

String title;

String artist;

String album;

String genre;

int rank;

public Top10Song(){

}

public Top10Song(String title, String artist, String album, String genre, int rank) {

super();

this.title = title;

this.artist = artist;

this.album = album;

this.genre = genre;

this.rank = rank;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getArtist() {

return artist;

}

public void setArtist(String artist) {

this.artist = artist;

}

public String getAlbum() {

return album;

}

public void setAlbum(String album) {

this.album = album;

}

public String getGenre() {

return genre;

}

public void setGenre(String genre) {

this.genre = genre;

}

public int getRank() {

return rank;

}

public void setRank(int rank) {

this.rank = rank;

}

}