Software Documentation

Creating the database (one time only, already done, no need to rerun will take a couple hours)

- First you must execute the CREATE-DB-SCRIPT.sql script this will remove all the previous tables and rebuild them.
- Then, cd into the directory with the python script that populates the DB: `cd SRC/API-DATA-RETRIVAL`
- Populate the database: `python populate_db.py`
- This will take a couple hours since a very large database is built from 2 different external sites.

Running the app

- cd into the source directory: `cd SRC/APPLICATION-SOURCE-CODE`
- Configure app_config/config.py according to the credentials of the MySQL server you are using (just configure "user" and "pass"). It is currently configured to run on NOVA.
- From the configuration you may also configure the webserver ip. DO NOT CHANGE the port from 4000 since the client uses it.
- If this is the first time running the app, create the users database: `python create_users_db.py`
- Run the server: `python app.py`
- Open your browser to http://localhost:4000/
- Now you may browse the app.

Code Structure

The code has several parts:

- The database build sql script to create the tables: `SRC/CREATE-DB-SCRIPT.sql`
  - This script was run initially.
- The API data retrieval part: `SRC/API-DATA-RETRIVAL` which has the following modules:
  - `populate_db.py` : The python script which populates the database and uses the following mudules.
  - app_config : Contains all the configuration parameters for the mysql server for the populate_db.
  - musixmatch : Contains the helper class and our user configuration in order to use the MusixMatch API.
  - vohLyrics : Helper module for retrieving lyrics for songs retrieved via the MusixMatch API.
- The application source code. Written in accordance to flask conventions. In `SRC/APPLICATION-SOURCE-CODE` :
  - The client code:
    - templates/ : Directory to hold static .html files rendered by the server (according to Flask convention).
    - static/ : Directory containing the JS, CSS, bootstaps, etc. (in accordance to Flask convention).
  - The server code:
    - app_config : Module containing server configuration.
    - db_operations : Module that contains all the DB operations.
    - `app.py` : Script that runs the application and creates all the routes for the client. These routes check inputs and all the db_operations for the actual operations.

Description of the API use

The server supplied many APIs for the client in order to achieve the app functionality:

- Sign Up
- Log In
- Get Songs Table (or playlist songs table)
- Get Artists Table

- Get Albums Table
- Remove Song from Playlist
- Search (Autocorrect) for Playlist
- Add Song to New or Existing Playlist
- Get User Playlists
- Remove Playlist
- [Get Lyrics of Specific Song] (#get-lyrics-of-specific-song)

Sign Up

Adds a user into the Users database.

- **URL**
  /signUp
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>'
}
```

- **Success Response:**
  - **Code:** 200 OK
- **Error Response:**
  - **Code:** 409 CONFLICT
  - **Content:** `{ error : "User already exists" }`

Log In

Validates user credentials. If ok, client should store them locally and send the headers in every request.

- **URL**
  /login
- **Method:**
  `OPTIONS`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>'
}
```

- **Success Response:**
  - **Code:** 200 OK
- **Error Response:**

- **Code:** 401 UNAUTHORIZED
- **Content:** `{ error : "User and password do not match an existing user" }`

Get Songs Table (or playlist songs table)

Returns the table of all the songs depending on a filter and number of pages to display.

- If 'playlist_name' is defined in the body - then display the songs from that playlist.
- If 'playlist_name' is not defined in the body - then display from regular database
  NOTE: filters are in JSON in the Data Params

- **URL**
  /songs
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'entries_per_page' : <int: How many song entries you wish receive>,
 'page_index' : <int: the page number you wish to receive>
 'lyrics' : '<lyrics user wishes to search by>'
 'order' : '<"desc" or "asc" for descending or ascending order>',
 'field' : '<The field of which the data will be sorted>',
 'filters' : {
             'name' : '<song name>',
             'album' : '<album name>',
             'artist' : '<artist name>'
             'playlist_name' : '<name of a playlist belonging to the user>'
           }
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
    'list' : [
                { 'song': '<song name>', 'track_id': <int track id in the DB>, 'artist': '<artist name>', 'a
                { 'song': '<song name>', 'track_id': <int track id in the DB>, 'artist': '<artist name>', 'a
                { 'song': '<song name>', 'track_id': <int track id in the DB>, 'artist': '<artist name>', 'a
                { 'song': '<song name>', 'track_id': <int track id in the DB>, 'artist': '<artist name>', 'a
                ...
            ],
    'total_rows' : <int: The total number of entries passing the filter>
}
```

- **Error Response:**
  - **Code:** 401 UNAUTHORIZED
    Content: `{ error : "User and password do not match an existing user" }`

Get Artists Table

Returns the table of all the artists depending on a filter and number of pages to display. Currently does not check user for debugging simplicity. NOTE: filters are in JSON in the Data Params

- **URL**
  /artists
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'entries_per_page' : <int: How many song entries you wish receive>,
 'page_index' : <int: the page number you wish to receive>,
 'order' : '<"desc" or "asc" for descending or ascending order>',
 'field' : '<The field of which the data will be sorted>',
 'filters' : {
             'name' : '<artist name>',
             'number_of_songs' : <int: will return all artists with more than number_of_albums albums>,
           }
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
    'list' : [
                { 'name': '<artist name>', 'number_of_songs': <int: number of songs by this artist>},
                { 'name': '<artist name>', 'number_of_songs': <int: number of songs by this artist>},
                { 'name': '<artist name>', 'number_of_songs': <int: number of songs by this artist>},
                { 'name': '<artist name>', 'number_of_songs': <int: number of songs by this artist>}
                ...
            ],
    'total_rows' : <int: The total number of entries passing the filter>
}
```

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Get Albums Table

Returns the table of all the albums depending on a filter and number of pages to display. Currently does not check user for debugging simplicity. NOTE: filters are in JSON in the Data Params

- **URL**
  /albums
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'entries_per_page' : <int: How many song entries you wish receive>,
 'page_index' : <int: the page number you wish to receive>,
```

```
    'order' : '<"desc" or "asc" for descending or ascending order>',
    'field' : '<The field of which the data will be sorted>',
    'filters' : {
                'name' : '<album name>',
                'artist' : '<artist who released the album>',
                'number_of_songs' : <int: will return only the albums with at least this number of songs in it>
              }
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
    'list' : [
                { 'name': '<album name>', 'artist': '<name of artist who released the album>', 'number_of_so
                { 'name': '<album name>', 'artist': '<name of artist who released the album>', 'number_of_so
                { 'name': '<album name>', 'artist': '<name of artist who released the album>', 'number_of_so
                { 'name': '<album name>', 'artist': '<name of artist who released the album>', 'number_of_so
                ...
            ],
    'total_rows' : <int: The total number of entries passing the filter>
}
```

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Search (Autocorrect) for Playlist

Returns an array of `{'playlist_id': <int: id>, 'playlist_name': '<playlist name>'}` where playlist_name is a sub string of an actual playlist of the user Used as autocorrect. when user wishes to add a song to one of his playlists, in the field of playlist that he chooses, each letter typed should trigger this API.

- **URL**
  /searchPlaylist
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'search' : '<substring of a playlist_name that belongs to username>'
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
   'list' : [
              {'name': '<playlist name that the search param in the response is a substring of>'},
              {'name': '<playlist name that the search param in the response is a substring of>'},
              {'name': '<playlist name that the search param in the response is a substring of>'},
              ...
            ],
}
```

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Add Song to New or Existing Playlist

Adds a song to a user's playlist. This API is called after *Search (Autocorrect) for Playlist* API* when user decides on playlist. If the playlist does not exist, a new playlist with this song will be created.

- **URL**
  /addToPlaylist
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'playlist_name' : '<int: The name of the playlist chosen (existing or non existing>'
 'track_id' : <int: The id of the specific song the user chooses from the songs table (the /songs route)>
}
```

- **Success Response:**
  - **Code:** 200 OK
- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Remove Song from Playlist

Removes a song from an existing playlist. If all songs are removed, playlist will be deleted.

- **URL**
  /removeSongFromPlaylist
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'playlist_name' : '<name for the new playlist>',
 'track_id' : <int: The id of the specific song the user chooses from the songs table (the /songs route of a pl
}
```

- **Success Response:**
  - **Code:** 200 OK

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Remove Playlist

Removes a users playlist including all the songs.

- **URL**
  /removePlaylist
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'playlist_name' : '<name of the playlist to be removed>'
}
```

- **Success Response:**
  - **Code:** 200 OK
- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Get User Playlists

This is called when the user wishes to display his playlists. The page will be a list of buttons representing each playlist. Returns a list of `{'name': '<playlist name>', 'number_of_songs' : <int: # songs in playlist>}` corresponding to the playlists that belong to the user. This way the playlists can be displayed in the same way as songs/artists/albums

- **URL**
  /playlists
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'entries_per_page' : <int: How many song entries you wish receive>,
 'page_index' : <int: the page number you wish to receive>,
 'order' : '<"desc" or "asc" for descending or ascending order>',
 'field' : '<The field of which the data will be sorted: "name" or "number_of_songs">'
}
```

- **Success Response:**

- **Code:** 200 OK
- **Content:**

```
{
   'list' : [
              {'name': '<playlist name corresponding to user>', 'number_of_songs' : <int: Number of songs i
              {'name': '<playlist name corresponding to user>', 'number_of_songs' : <int: Number of songs i
              {'name': '<playlist name corresponding to user>', 'number_of_songs' : <int: Number of songs i
              ...
            ]
   'total_rows' : <int: The total number of playlists for the user>
}
```

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

Get Lyrics of Specific Song

This is called when the user wishes to view the lyrics of a single song. Returns a JSON with the lyrics string (check out return code).

- **URL**
  /singleLyrics
- **Method:**
  `POST`
- **HTTP Headers**
  None
- **URL Params**
  None
- **Data Params**

```
{
 'username' : '<username>',
 'password' : '<password>',
 'filters' : {'track_id' : <int: track_id of the song from the songs page>}
}
```

- **Success Response:**
  - **Code:** 200 OK
  - **Content:**

```
{
   'lyrics' : '<The lyrics to the song with >'
}
```

- **Error Response:**
  - **Code:** 500 INTERNAL ERROR
  - **Content:** `{ error : "<An informative error string>" }`

External Packages used

- MusixMatch: For retrieving information about songs, artists, and albums.
- vohLyrics: For retrieving lyrics for the songs.
- Flask: For managing the web server, defining routes, parsing JSON objects, handling HTTP requests, etc.
- MySQLdb: For connecting to the MySQL server, executing queries, and parsing the results.

General Flow of the Application:

1. The database tables must be created with the `SRC/CREATE-DB-SCRIPT.sql` script (one time).
2. The database tables must be populated with the `SRC/API-DATA-RETRIVAL/populate_db.py` script (one time).
3. The app can be run by executing the `app.py` script.
4. User may browse and use the app.