Spotify Group Recommendation System

The Spotify Group Recommendation System will mainly consist of three parts. First, a frontend so users can interact with the system. Second, backend that handles the main business logic for the frontend and acts as an interface and a data parser to the machine learning implementation and the official Spotify API. Third, a machine learning implementation that handles the playlist generation.

The following describes the different use cases and interactions between the frontend and the backend.

# Frontend

## User Interactions

### Accounts

It should be possible to create accounts in the Spotify group recommendation system. Every user needs to be able to join a group and invite other users to the group.

User-specific data, like friends, groups or Spotify connections (tokens) will be saved into a user database. This ensures a consistent reusability for every user.

Every user has access to all of his data and can delete the data at any time.

### Groups

A group consists of a minimum of two users and can have multiple group playlists. Every group member can create a playlist.

It is possible to leave a group. A group itself will only be deleted if every user has left the group. This ensures access to group data even if every other member has left the group.

### Playlists

After a group member has created a playlist, it will be linked to the Spotify account of the creating user. Otherwise, the app would need to create some form of an anonymous/ group-specific user in Spotify.

In addition to the playlist being linked to the user, it will also be saved in the apps database. This prevents loss of data if the creating user decides to remove the playlist from his Spotify account.

### Use Cases

|  |  |
| --- | --- |
| Goal | Create a user account |
| Primary Actor | Unregistered user |
| Scope | Frontend and user database |
| Level | User |
| Precondition | Unregistered user is at registration screen/ page |
| Success end | User account is created |
| Failure end condition | User account is not created |
| Trigger | Unregistered user accesses registration page |
| Main success scenario | 1. Unregistered user enters user name and password 2. User is saved to database 3. System informs user of creation, user has to confirm mail address 4. User is forwarded to account page 5. User continues with setting up Spotify Access (optional) |
| Extensions (Error scenarios) | 1. User name already exists 2. Some form of connectivity issues with the backend |
| Variations (alternate scenarios) |  |

|  |  |
| --- | --- |
| Goal | Get users Spotify access |
| Primary Actor | User |
| Scope | Frontend and backend API |
| Level | User |
| Precondition | User is at account page |
| Success end | App has access to users Spotify account through Spotify API |
| Failure end condition | No access to Spotify API |
| Trigger | User presses button ‘Connect with Spotify’ |
| Main success scenario | 1. User presses button 2. User provides needed data to get user-specific information through the Spotify API 3. Test request to Spotify API returns some form of 200 Status |
| Extensions (Error scenarios) | 1. Data provided by user is wrong 2. Spotify API is unreachable 3. User has no Spotify account |
| Variations (alternate scenarios) |  |

|  |  |
| --- | --- |
| Goal | Creating a playlist |
| Primary Actor | User |
| Scope | Frontend and backend API |
| Level | User |
| Precondition | User is on group page and starts playlist creation |
| Success end | A new group playlist is created |
| Failure end condition | No new playlist created |
| Trigger | User presses Button ‘Create Playlist’ |
| Main success scenario | 1. User starts playlist creation 2. User-specifies playlist conditions    1. E.g. specific genre    2. Main variable that should be applied (song listen count etc.) 3. User confirms selection 4. Playlist is returned as a link and saved to the database |
| Extensions (Error scenarios) | 1. Not all necessary decisions on conditions have been made 2. Spotify is not reachable 3. Database access is unavailable |
| Variations (alternate scenarios) |  |

|  |  |
| --- | --- |
| Goal | Deleting a playlist |
| Primary Actor | User |
| Scope | Frontend and backend API |
| Level | User |
| Precondition | User is at group management site |
| Success end | Playlist is deleted on user account |
| Failure end condition | Playlist not deleted |
| Trigger | User presses ‘Delete Playlist’ |
| Main success scenario | 1. User decides to delete playlist that has been created from his Spotify account 2. User clicks on ‘Delete’ button 3. Playlist is removed from user’s Spotify account but not from the app’s database 4. UI informs about deletion |
| Extensions (Error scenarios) | 1. No Spotify access 2. User has no created playlists |
| Variations (alternate scenarios) |  |

|  |  |
| --- | --- |
| Goal | Leaving a group |
| Primary Actor | User |
| Scope | Frontend and backend API |
| Level | User |
| Precondition | User is at group management site |
| Success end | User has left group |
| Failure end condition | User has not left group |
| Trigger | User presses ‘Leave Group’ button |
| Main success scenario | 1. User chooses to leave group 2. UI informs about success 3. Group is no longer accessible from user’s perspective |
| Extensions (Error scenarios) | 1. No backend API access 2. Other database error |
| Variations (alternate scenarios) | 1. User is the last member of the group    1. User gets removed from group    2. Group itself is deleted as well |

|  |  |
| --- | --- |
| Goal | User satisfaction |
| Primary Actor | Users |
| Scope | Frontend + Backend / Voting interface |
| Level | User |
| Precondition | Users listened to the playlist |
| Success end | Users rated the experience with the playlist 3.5/5 or higher |
| Failure end condition | Rating is lower or non-existent |
| Trigger | Users used integrated voting system to vote on playlist |
| Main success scenario | 1. Users listened to music 2. Users vote 3. Rating is saved and displayed |
| Extensions (Error scenarios) | 1. Rating is made before playlist has been listened to 2. Other users who didn’t have the music playing on their phone but participated in the group can’t vote |
| Variations (alternate scenarios) | No voting happened -> no voting gets saved |

## Communication to the backend

All logic will be handled by the backend API of the group recommendation app.

The backend receives requests from the frontend, parses them and sends them to the ML interface.

### Request/Response Flow

#### User-specific Spotify data

This flow includes user-specific data and thus will probably be handled by multiple endpoints.

Possible requests could be

* Validate user access
* Access user’s Spotify library
* Add playlist to user
* Delete playlist from user
* Get metadata of user’s songs/ playlists

|  |  |
| --- | --- |
| Goal | Get/ Manipulate user data |
| Primary Actor | Frontend |
| Scope | Frontend + backend (connect to Spotify) |
| Level | Internal and external |
| Precondition | Frontend has sent request |
| Success end | Backend returns requested data |
| Failure end condition | Data not available, Spotify unavailable |
| Trigger | Frontend request |
| Main success scenario | 1. Request is received 2. Content is parsed 3. Content is sent to Spotify API 4. Response is received from Spotify 5. Response is parsed and sent back to frontend |
| Extensions (Error scenarios) | 1. Spotify API not available 2. Malformed Request |
| Variations (alternate scenarios) |  |

#### User data

This flow handles user data that is not related to Spotify. This means the users that have registered on the app, created playlists and so on.

|  |  |
| --- | --- |
| Goal | Get/Manipulate user data |
| Primary Actor | Frontend |
| Scope | Frontend + backend (User database) |
| Level | Internal |
| Precondition | Frontend has sent request |
| Success end | Backend returns requested data |
| Failure end condition | Data not available |
| Trigger | Frontend request |
| Main success scenario | 1. Request is received 2. Content is parsed 3. Content is sent to user database 4. Response is received from user database 5. Response is received and status is returned (success/ failure) |
| Extensions (Error scenarios) | 1. Malformed request 2. Data unavailable |
| Variations (alternate scenarios) |  |

#### ML Requests

These flows contain data to produce/ handle the newly generated playlists.

The normal flow is that the user has requested a new playlist and the backend then is responsible for sending the multiple requests and responses.

##### Example Flow:

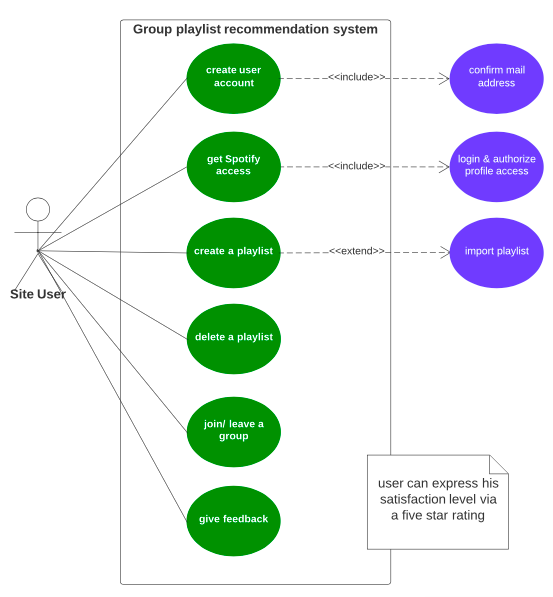
Request Playlist (includes group members and possibly genre)   
-> backend requests user (multiple users) metadata from Spotify   
-> metadata is parsed into a format that the ML interface can handle   
-> send request to ML interface   
-> receive response from ML   
-> possibly parse response   
-> send data to Spotify to create and save a new playlist   
-> send request status back to frontend

|  |  |
| --- | --- |
| Goal | Get new playlist |
| Primary Actor | Frontend and ML |
| Scope | Frontend + ML |
| Level | Internal and external |
| Precondition | Frontend has sent request |
| Success end | Backend returns requested data |
| Failure end condition | Data not available |
| Trigger | Frontend request |
| Main success scenario | 1. Request is received 2. Content is parsed 3. Content is sent to ML interface 4. Response is received from ML interface 5. Playlist is returned to Frontend and sent to Spotify |
| Extensions (Error scenarios) | 1. Malformed request 2. Data unavailable |
| Variations (alternate scenarios) |  |

### Diagram Representation

From our perspective, a UML use case diagram is most appropriate to demonstrate the different ways that a user may interact with the system. The use case diagram should not provide too many details. Instead, it depicts a high-level overview between the actor (site user) and the group recommendation system. In this way, it supplements the written requirements.

UML Website Use Case Diagram



Flow Diagram... (Daniel)