**Requirements Document**

1. Introduction

FaceSpaceIn+ wants to develop a software system that can analyze music files for other metrics to generate playlists as well as relying on user-generated meta-data to build lists.

They would like a back-end system capable of analyzing files and generating metadata, access to the streaming service through a web interface, and a mobile app which has similar features.

1. Audio Analysis
   1. Terms:
      1. Back-end System - The program that the backend user uses to analyze and upload the song as well as provide information to the database.
      2. File storage server - the server to keep the mp3 files after upload
      3. Database - the location where information gathered from analysis is kept
   2. User Information
      1. Characteristics of a common backend user
         1. Tech Savvy
         2. Has knowledge of how the backend program code works
         3. Understands file storage systems and databases
         4. Is able to operate a computer from the command line
      2. The backend user is responsible for analyzing and uploading new music files that will be available for the web/app
      3. Ideal case: The backend user types in one command with one argument (a directory containing new files) and everything below runs automatically without needing further assistance.
         1. Songs should be analyzed
         2. Results should be uploaded
         3. MP3 file should be uploaded
         4. Confirmation should be reported
   3. Assumptions
      1. There need not be a GUI for the backend system
      2. The back-end system should work only for .mp3 files.
      3. The MP3 files that will be analyzed are assumed to have all correct tag information before analysis and upload.
      4. The MP3 files to be analyzed and uploaded should be DRM free.
   4. General Requirements
      1. To analyze and upload songs, the back-end user should simply type a command on a command line.
      2. Should be able to iteratively analyze and upload a directory of many MP3 files at once.
      3. The back-end system should change the filename of the MP3 file to the form: song\_title-artist\_name-album\_name.mp3 to prevent the song titles from conflicting with database information.
   5. Analysis
      1. If song is in an unsupported format it should be reported in an error log.
      2. The back-end system should provide a function that will extract the following information that is stored in the ID3 tags attached to each MP3 and provide the information to the database.
         1. Title
         2. Artist
         3. Album
         4. Genre
         5. Year
         6. Song Length
      3. Analyze song waveforms to obtain additional information and send the information to the database.
         1. Tempo
         2. Key
         3. Loudness
         4. Amount of Bass
      4. Back-end system should log the completion of the analysis and provide the back-end user with a confirmation of completion or error.

* 1. Uploading Song and Information
     1. Songs should be analyzed before upload.
     2. The back-end system should provide an upload service that will add the analyzed songs to a specific file storage server that can be accessed by the the Web/App and Database.
     3. The information from the analysis should be stored to a database.
     4. No song that is not present in the database should exist in the file storage server.
     5. No song that is not present in the file storage server should exist in the database.
     6. The upload service should check if a song already exists in the file storage when uploading.
        1. If it does already exist, it should not upload that song file, exclude that specific song information from being uploaded to database,and report error to back-end user.
     7. The upload service should provide the location that it will store the file to the database.
     8. The upload service should store the file in the same location as it is written in the information given to the database.
     9. The service should log whether or not the information was successfully stored to the correct location.

1. Database

4.1 Needs to be able to generate playlist

4.1.1 Compare song info to find similarities

4.1.1.1 User selects initial song which exists in database through web/app

4.1.1.2 Database compares selected song’s fields to other songs in

database

4.1.1.2.1 Include user like/dislike and custom tags in selection

algorithm

4.1.1.3 Similar songs are compiled into playlist

4.2 Need to communicate with other systems

4.1.1 Algorithm Analysis Data

4.1.1.1 Read output file from analysis

4.1.1.2 Insert data from analysis into approriate tables

4.1.2 Send tag info / playlist to Web/App

4.1.2.1 Generate playlist from user input sent by web/app (see 4.1.1)

4.1.2.2 Timely playlist generation

4.1.2.2.1 Create list and send back to web/app in under 5 sec

4.1.2.3 Add new user account info into approriate tables

4.1.2.4 Authenticate user login info by comparing to stored account info

4.1.2.5 Insert user like/dislike and tagging input into appropriate tables

1. Web/App

5.1 Website

5.1.1 Login/Registration Page

5.1.1.1 Input Fields

5.1.1.1 Username

5.1.1.2 Password

5.1.1.3 Email

5.1.1.4 First/Last Name

5.1.1.2 Submition Button

5.1.1.3 Input Validation

5.1.2 Custom Media Player Page (From most/least priority)

5.1.2.1 User Login Validation: If a user needs to login, they are presented with login on the same page. Otherwise, a registration button is supplied that leads user to the registration page.

5.1.2.2 Media Player

5.1.2.1.1 Play/Pause Button

5.1.2.1.2 Skip Button (Sequential)

5.1.2.1.3 Volume Slider

5.1.2.1.4 Like/Dislike Button

5.1.2.1.5 User-tag Button

5.1.2.1.6 Skip Button (Based on like/dislike/tagging)

5.1.2.3 Sidebar (Tabs) (For mobile & desktop versions)

5.1.2.2.1 Search Bar (For particular songs/playlists?)

5.1.2.2.2 Account options

5.1.2.2.2.1 Change password

5.2 Mobile App/Mobile Browser App (Android)

5.2.1 Audio Player (functionality pulled from 5.1.2)

5.2.2 Hidden Sidebar

5.2.2.1 Change visibility with finger swipe

5.2.2.2 Functionality pulled from 5.1.2.3