Serite Ross, Kaylin Archuleta, Linh Tran

Project Requirements for MusicSpace

MusicSpace functionality includes importation of playlists from several external websites to MusicSpace. There is also a community space where users can share their playlists with others.

User 1: Playlist import

User presses “import” button.

User selects website options from dropdown menu.

User provides credentials to link external website account.

User selects external playlist and presses import.

Testcase 1:

Input:

ImportButton: on

siteList: user select from list

credentials: user text input

expected output: login success

actual output: ?

pass/fail criterion: ExpectedOutput == actualOutput

User 2: Playlist merge

User presses “merge” button.

User selects previously imported playlists.

User enters name for new playlist.

User presses OK.

Testcase 2:

Input:

playlistMerge: on

playlists: user multiselect from list

playlistName: user text input

expected output: new playlist of user entered name consisting of contents of all selected playlists

actual output: ?

pass/fail criterion: ExpectedOutput == actualOutput

User 3: Play playlist

User selects playlist from screen.

User presses play on any of the listed songs.

If user presses shuffle, all songs in the playlist will play in random order.

If user presses repeat once, the entire playlist will continuously repeat.

If user presses repeat twice, the current song will continuously repeat.

Testcase 3:

Input:

playlistSelect: user select from list

songSelect: user select from list

songFunction: play

shuffleToggle: on/off selected by user

repeat: state 1, 2 (repeat all), or 3 (repeat one) selected by user; default to 1

expected output: song sound output based on user selection; consecutive songs play according to toggle & repeat setting selected

actual output: ?

pass/fail criterion: ExpectedOutput == actualOutput

User 4: Trial Period/Access

The user is prompted to make an account for a free trial after first playlist finishes.

User inputs an email, username, and password.

User inputs password again.

The two passwords are compared to see if they match.

If they match, the user is registered.

The date for the end of the trial period is set.

Users in a free trial or subscribed users can access all available features.

The user is prompted to subscribe when the date arrives.

None of the playlists the user made are deleted if the user does not subscribe.

If the user does not subscribe, they are unable to access any features.

Testcase 4:

Input:

playlistEnd: true

credentials: user supplied text

userRegistered: true

accessEnd: set date

expected output: userRegistered == true

actual output: ?

pass/fail criterion: ExpectedOutput == actualOutput

User 5: Subscription

User clicks “subscribe” button.

User clicks either the monthly or yearly option.

User enters their credit card information.

The user is charged with their chosen fee.

The date for the next charge is set.

Test case 5:

Input:

subscripeButton = true

chargeType = "monthly" or "yearly"

creditCard = the user's entered credit card info.

chargedOnce = 1

chargeDate = the date the user will be charged next.

ExpectedOutput: the user’s charge type is set to monthly or yearly, the charge date is chosen based on the charge type. The user’s credit card information is stored (and encrypted), and charged once and only once, indicated by chargedOnce.

ActualOutput: ?

pass/fail criterion:

ExpectedOutput == actualOutput

User 6: Search

User inputs a string into the search.

The site searches for playlists that match the title.

The site shows the page with the list of matching search results.

If the amount exceeds 15, the results are divided into pages.

15 results per page.

Test case 6:

Input:

searchString = the user's inputted string

matchList arrayList of matching results

ExpectedOutput: a display of the results matching the searchString

ActualOutput: ?

pass/fail criterion:

ExpectedOutput == actualOutput

User 7: Create a Playlist

User clicks “Create” button.

User inputs a name before they can put in any songs.

User inputs URLs for each song.

User moves songs to match their desired song order.

User uploads an image file to use as a playlist cover.

If the user does not upload an image, a default one is chosen instead.

Test case 7:

Input:

playlistCreateButton = true

playlistTitle = user's inputted string

songList = list of URLs to each song

playlistCover = user's uploaded image file, or a default image

ExptectedOutput: a new playlist made with the user’s desired title, list of songs, order and cover.

ActualOutput: ?

pass/fail criterion:

ExpectedOutput == actualOutput

User 8: Song recommendations

User is recommended songs on home page based on their recent song searches genres/artists and listening frequencies, which is updated weekly.

Testcase 8:

Input:

weeklyUpdated : true

list 20latestSongSearch

list playingHistory

topFrequency = maxRepeated

top10artists = random artist(20lastestSongSearch || sorted playingHistory(topFrequency))

top10genres = random genres(20lastestSongSearch || sorted playingistory(topFrequency))

Expected Output: random List songsRecommendation from top10artists && top10genres

Actual Output: ?

Pass/Fail Criterion: Actual Output == Expected Output

User 9: Sleep timer

User chooses a desired timer: 15, 30, 45 minutes.

MusicSpace stops playing when the timer is up.

Testcase 9:

Input:

enum Timer{15, 30, 45}

music: playing

timerSelect == enum Timer

if(timerSelect == finished)

Expected Output: music: off

Actual Output: ?

Pass/Fail Criterion: Actual Output == Expected Output

User 10: Most popular songs/playlists and listening statistic

User opens MusicSpace.

User sees recent most popular songs on homepage and a list of 10 songs/playlists he or she has been listening to the most.

Testcase 10:

Input:

List songHistory

counter playingFrequency

songHistory: sorted (base on playingFrequency)

popularSongs = 30songsWithMostListeners

10mostPlayedSongs = top 10 song from songHistory

Expected Output: popularSongs and 10mostPlayedSongs printed

Actual Output: ?

Pass/Fail Criterion: Actual Output == Expected Output