Ursinus
COLLEGE

Test Plan

AUTHORS

Nguyen, Khoi
Santella, Michael
Siders, Mia
Son, Isabelle
Thompson, Eugene

2025-04-04

Contents

1 Introduction

This test plan describes the approach taken to testing the RhythmRun application on iOS and Android.

1.1 Purpose of Document

This document identifies functional unit tests, output tests and user acceptance tests related to the RhythmRun application, serving as the guideline for developing test code.

2 Test Design

2.1 Summary of Test Types

Test Type	Definition
Unit Test	Verifies program (or module) logic based on knowledge of the program structure.
Integration and Deployment Tests (CI/CD)	Test through automated platforms (Xcode Cloud and GitHub Workflows) for the iOS and Android platforms respectively..
User Acceptance Test (User)	Testing the RhythmRun application with beta testers (e.g, other groups in Software Engineering or interested stakeholders) to simulate the application's functionality in the real world.

Table 1: Testing Types and Definitions

3 Tested Features

3.1 Tested Requirements Table

Test	Requirement Key	Test	Expected Behavior
1	importLocalPlaylist(localPlaylist)	Inputs: valid .mp3 .wav songs, invalid mp3 or .wav format	A properly formatted playlist containing the imported playlist is returned. If not, the user receives a pop-up notification if the local playlist is not found on their device. #From the local file, hide all the difference format except the .wav and .mp3
2	importLocalStorageButton()	Test if the button can activate and can connect to importLocalPlaylist function. Inputs: None	The method *importLocalPlaylist* is called on the file
3	importMusicButton()	Verify that the authentication token contains alphanumeric characters (A-Z, a-z, 0-9) and underscores and hyphens (_,-). Inputs: An authentication token that actually contains alphanumeric characters and underscores and hyphens. Unexpected Inputs: Plugging in "[][][][]" as an authentication token	Returns the authentication token
4	addSongToLibrary(trackID)	Try loading some songs from each platform: Apple music and local file into a Spotify "environment" Input: TrackID format Unexpected input: Non-TrackID type	A track is added to the user's Spotify library.
5	skipSong()	Select two songs and called the skip button. Inputs: None	The next song on the user's music queue on Spotify is skipped to.
6	shuffleSongs()	Sample at least 4 songs and call the shuffleSong method multiple times, and print out the upcoming song (or the next state)Inputs: None	Shuffles the user's playlist on if the shuffle status is off or shuffles the user's playlist off if the shuffle status is on.

7	repeatSong()	Sample 2 songs and print out the name of the upcoming song. Inputs: None	Toggles the repeat function of Spotify, either on loop on one song, or on a playlist
8	queueSong(trackID)	Queues a song corresponding with the trackID of the appropriate streaming service (Spotify or Apple Music). Inputs: A song name that is a string; the last element of the queueSong function, which is a string. Unexpected Inputs: A non-string value; empty string; a non-existent trackID	Adds the song that is next in the queue to the current playback session.
9	checkAuthorization(authStatus)	Make the test cases with authorized state and without authorized state. Inputs:Approved. Unexpected input: Denied.	Returns the boolean value: this value is true iff the user has successfully authorized the application to use the API and is false if the auth_status is not authorized otherwise.
10	setMusicQueue(songs, playlist)	Check the added song if it is inside the playlist or not. Inputs: A song that is a string; a playlist that is a string Unexpected Inputs: A non-string value; empty string; a non-existent song; a non-existent playlist	Adds the song that is next in the queue to the current playback session.
11	play()	Test the play() function and compare with expected current song's name. Inputs: None	The most recently played song is returned.
12	trackHeartRate(workoutSession)	Test at least 5 or more times to estimate the difference value of trackHeartRate of this session and the expected value to verify if it's close enough. Inputs: a valid workoutSession (dictionary), an invalid workoutSession, an empty workoutSession, an non-dictionary input	Returns an integer array or json file of the user's heart rate during the workout with timestamps

13	syncFromAppleWatch(heartRate)	Verify all health statistics are properly loaded into the application. Input: expected input - i.e. 90Unexpected Input: A string; a float; a number $x < 0$	The heart rate data is exported over to Apple Health using HealthKit
14	localImport(str fileLocation)	Verify if the file Path exists or not and print out from the name of the playlist that this file path contains.Inputs: a valid file path, a non-existent file path, an empty string, a non-string input	A playlist is imported into the app
15	spotifyImport()	Sample the songs and playlists from Spotify account. Inputs: None	A Spotify playlist is imported into the app.
16	appleImport()	Sample the songs and playlists from Apple account. Inputs: None	The Apple Music playlist that the user selected is imported
17	importButton()	Compare the playlist id with the playlist ID from the import Button()Inputs: None	A playlist is loaded into the app
18	findStartingBeats(Song song)	Test how close to the function method value with the expected value. Inputs: a valid song object, an invalid song object, a default song object (i.e. no song name, length, etc.)	Return the BPM in range 61-150
19	findEndingBeats(Song song)	Test how close to the function method value with the expected value. Inputs: a valid song object, an invalid song object, a default song object (i.e. no song name, length, etc.)	Return the BPM in range 61-150

20	findSongLength(Song song)	Print out the song duration that matches to actual song length from spotify or apple or local. Inputs: a valid song object, an invalid song object, a default song object (i.e. no song name, length, etc.)	Return the time in minutes and seconds from the associate with the number of samples.
21	findAverageBPM(startingBeats, finalBeats, songLength)	Test by comparing output to mean of both parameters. Inputs: A starting beat that is a float; a final beat that is a float; a song length that is a floatUnexpected Inputs: A string; a number $x < 0$; a unit with alphabetic characters; a starting beat, a final beat, and a song length that yields a BPM b such that $b < 61$ or $b > 150$	Return BPM between 61-150.
22	askExperienceLevelButton()	Clicking this button to get the result. Inputs: None	Appropriate experience level is selected.Print out the name of the button and the interval value associated with that.
23	getAvgPace(expLevel)	Verify pace matches with preset pace group associated with user's selected experience level. Input: A pace with only numeric values; a preset pace group with only numeric values; a selected experience level. Unexpected Input: A string; a number $x < 0$; a unit with alphabetic characters	One pace associated with experience level is returned.
24	askPaceButton()	Verify user inputs pace in proper formatting: minutes, seconds, and all values are numeric. Input: A pace with only numeric values. Unexpected Inputs: A string; a number $x < 0$; a unit with alphabetic characters; a unit in miles instead of seconds or minutes	Desired range of paces is returned.

25	calculatePaceIntervals(paceGroup)	Verify if the unit from the function is in minutes per mile and the value makes sense with the real world data.Inputs: Some form of pace group. Unexpected Inputs: A string; a number $x < 0$; 77 meters per second	Range of paces from the average pace group is returned.
26	getHeartRate(heartRate)	Verify if the casting integer and BPM unit is correctly transferring from the sensor.Input: expected input - i.e. 90Unexpected Input: A string; a float; a number $x < 0$	An integer of the heart rate from the Apple Watch is returned.
27	getDistance(distance)	Test the distance, make sure that the unit type is miles. Input: A distance in miles. Unexpected Inputs: A string; a number $x < 0$; a distance in feet or meters	A float of the distance traveled from the Apple Watch is returned.
28	getElevationChange(elevation_chang	Verify if it returns number in appropriate unitInput: Any Float number. Unexpected Inputs: Inputting a number $x < 0$; inputting a string	A float of the change in elevation from the Apple Watch is returned.
29	init(soundFile: soundType)	Test with the first instance which is the song file, and compare the properties with 0 to be equaled.Input: Song file	The length and tempo of the song is initialized to be 0.
30	manualTempo(tempPoint,length, double)	Verify if the song length matches intended time duration.Inputs: Song length, Any float number, Song tempo	Search and filter out the BPM close enough to tempPoint and fit to the time duration. Prioritize the tempo searching over the length of the song.

31	adjustTempo(playlist)	Verify the new tempo of the song which close enough to the requirement tempo. Inputs: a valid playlist, an empty playlist, a playlist with invalid song objects	Return the new song's sample associated with adjusted tempo.
32	songTransition(song1: Song, song2: Song)	Test if the songs are valid the time delay between two songs are as small as possible. Inputs: valid song objects, invalid song objects, default song objects (i.e. no song name, length, etc.), a valid song object and an invalid song object	A transition is applied between two songs that have been provided.
33	repeatSong(song: Song)	Test out if the formatted song loop works.Inputs: a valid song object, an invalid song object, a default song object (i.e. no song name, length, etc.)	A song is repeated.
34	startWorkout()	Test some API functions to verify if it worked properly or not.Inputs: None	The workout for HealthKit is started.
35	endWorkout()	Check the state of work out and also the function contains the Healthkit initialization, then check the changes of the distance, running route, duration of the workout, the pace, and the elevation. Inputs: None	The following are returned: the distance, running route, duration of the workout, the pace, and the elevation changes.
36	changeDuration(duration:double)	Test the default duration variable before and after change. Inputs: expected input - i.e. "short", "b", 0, ""	The duration of the workout is altered to what the user wishes.
37	changeDurationButton()	Check if the button works and the return duration is correct numerically to the user's choice.Inputs: None	*changeDuratio*n is called with an appropriate duration of the user's choice.

38	changeLevel(level)	Tracking the previous and the current level experience.Inputs: expected input - i.e. “low”, “b”, 0, “”	The experience level is changed to what the user desires.
39	changeLevelButton()	Click the button and print out the value of this button to compare with expected value. Inputs: None	*changeLevel* is called with an appropriate level of the user’s choice.
40	generatePlaylist(xp_lvl, intensity, duration)	Dependent on xp_lvl, intensity, duration. Inputs: expected input - i.e. (“beginner”, “medium”, “short”), (“b”, “b”, “b”), (0, 0, 0), (““, ””, ””)	An application-generated playlist with the characteristics that the user requested is returned. If the requested characteristics cannot be fulfilled then the user receives a pop-up notification
41	savePlaylist(playlist)	Verify playlist is in library after selecting save button. Inputs: a valid playlist, an empty playlist, a playlist with invalid song objects	A playlist of the user’s choice is saved..
42	deletePlaylist(playlist)	Verify playlist is removed from library after selecting remove button.Inputs: a valid playlist, an empty playlist, a playlist with invalid song objects	A playlist of the user’s choice is deleted, with the user receiving a pop-up notification to ensure that he or she truly wants to delete the playlist.

Table 2: Tested Requirements

3.2 User Acceptance Testing Scripts

Test	Test Case Name	Steps	Expected Behavior
1	Import the user’s local music into the RhythmRun application	1. Start the RhythmRun App 2. Press the “Continue with Local Music” button on the landing page 3. Press the “Choose Files” button. 4. Start selecting .mp3 or .wav files from your local files or cloud storage 5. Press the “Analyze BPM” button. 6. You see a confirmation popup if it was successfully analyzed!	After the user opens the application, the landing page should transition into the next page that displays the “Choose Files” and “Import” button and the user should be able to see their music files in their appropriate operating system interface. After selecting files, the user should see their selected music files when redirected back to the “Import Songs” page. If the user did not select any songs, then raise a error box that says “No songs were imported”Else, if the user selected songs for analysis, the user will see a confirmation popup that they were successfully analyzed

2	Import the user's playlist from Spotify	Pre-condition: You must have a Spotify Account and have the Spotify app installed on your device. 1. Start the RhythmRun App 2. Press the "Continue with Spotify" button. 3. Redirects to the Spotify app if the app is on the phone. 4. Press "Agree" to allow Spotify to connect with RhythmRun 5. Redirected back to the RhythmRun app. 6. In the "Import Playlists" screen, select playlists from your Spotify account and then click the "Import" button 7. You see a confirmation popup if it was successful!	After the user opens the application, and pressing "Continue with Spotify": • If the user has the Spotify app – If the user does not have an account, it will automatically show a dialogue box that says "Please Log in", with subtext that says "The link will open once you have logged in." – If the user has an account, then Spotify's authentication screen should appear with the choice to either "Agree" or "Cancel" – If the user has an account and agrees then it goes back to the RhythmRun application the user will see their playlists. – If the user has an account and cancels the authentication, they are redirected back to the RhythmRun app, and it presents a dialogue box displaying "Spotify Authentication Failed, Try Again?" with an option to either authenticate again or cancel and return to the landing page. • After the user allows RhythmRun to access Spotify, has an account and the app installed, the user is presented with their playlists to select with checkboxes – If the user selects nothing then raise an error box that says "No playlists were imported" – If the user selects playlists then they should see a confirmation popup

3	Import the user’s playlist from Apple Music	Pre-condition: You must have an Apple Music Account. 1. Start the RhythmRun App 2. Press the “Continue with Apple Music” button 3. Press “Allow” to the request to allow RhythmRun to access Apple Music 4. Select playlists from your Apple Music account and then click the “Import” button 5. User sees a confirmation popup that it was successful!	After the user opens the application, and pressing “Continue with Apple Music”: • A popup will appear that says “‘RhythmRun’ would like to access Apple Music, your music and video activity, and your media library” – If the user selects “Allow,” the user will see their playlists to select for import – If the user selects “Don’t Allow,” the user will see a dialog box that says “Apple Music Authentication Failed” and it has instructions to allow it if the user changes their mind. If the user clicks “Cancel,” they are redirected to the landing page • After the user allows RhythmRun to access Apple Music, the user is presented with their playlists to select with checkboxes – If the user selects nothing then raise an error box that says “No playlists were imported” – If the user selects playlists then they should see a confirmation popup
4	Test the BPM analysis loading screen	Pre-condition: You have imported songs through either local storage, Spotify, or Apple Music. Wait for the BPM analysis to complete, ensure GIF runs properly.	After the user opens the application, and goes through all the steps previously to import their playlist. After pressing the “Import” button, the user is presented with a loading screen of a runner running and after BPM analysis is completed, the user is presented with a page about workout options.

5	Set workout modes (experience level, intensity, duration)	Pre-condition: You have imported songs through either local storage, Spotify, or Apple Music and the BPM analysis is finished. 1. Select experience level. 2. Click the "Next" button. 3. Select intensity level 4. Click the "Next" button. 5. Select the desired duration 6. Click the "Next" button.	The app should adjust the workout playlist based on the parameters inputted by the user.
6	Adjust workout playlist song order	Pre-condition: You have finished all the previous steps and a workout playlist has been generated. 1. Click and drag your desired song to rearrange the song order. 2. Finalize the playlist (save changes) by pressing the "Save" button .	The playlist should reflect the decided playlist order.
7	Pause and resume workout	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the "Pause" button. 2. Observe if music stops. 3. Click the "Play" button. 4. Observe if music plays.	The current workout is paused after pressing "Pause", and the current song stops playing. By pressing the "Play" button, it resumes the workout and the most recent song starts playing again

8	Skip songs	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the “Skip” button (right facing). 2. Observe song change.	Current song is skipped and the next song is played in queue, if there are no songs left, stop playing any music and let the user know in a notification
9	Go back to previous song	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the “Previous” button. 2. Observe song change.	Current song should be the previous song and start playing after clicking the button
10	Loop song	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the “Loop” button. 2. Wait for the song to finish playing 3. Observe playback	Current song should start playing again once the song has finished.
11	Shuffle songs	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the “Shuffle” button. 2. Observe the song playback order.	Songs should play in a randomized order.

Table 3: User Acceptance Testing Scripts

List of Tables