

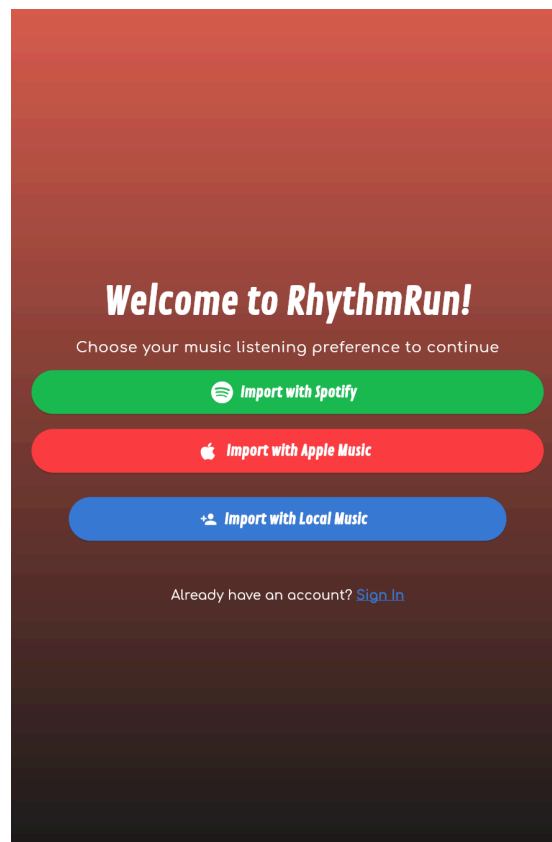
# User Guide

## Introduction:

RhythmRun is an app that creates playlists for you with BPMs that match the pace at which you wish to run. You can either upload your own personal music playlists or link up this app with your playlists from Apple Music or Spotify. Once you upload the relevant music, you can choose an experience level and a pace to determine how fast you want to run and, by extension, how fast-paced your music is.

## Getting Started:

When you first open the app, you will see the home page.

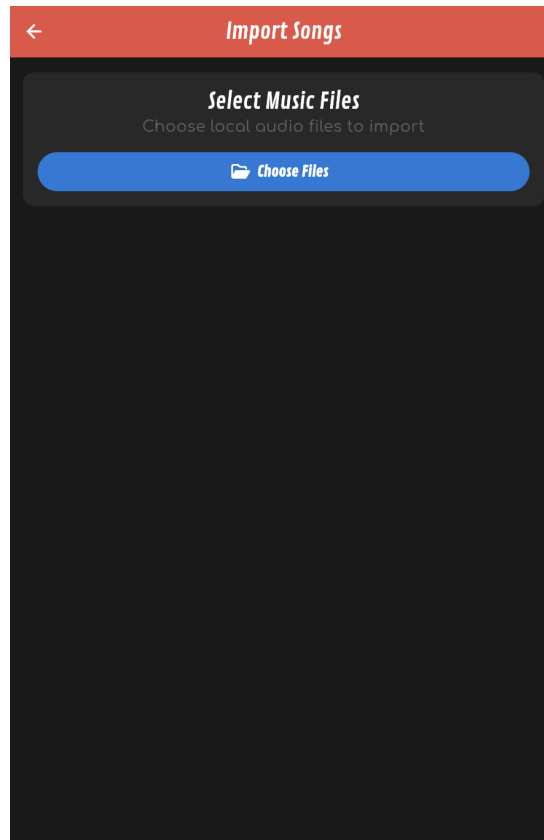


From here, you can choose to either:

- Connect to Spotify
- Connect to Apple Music
- Upload music files from your device

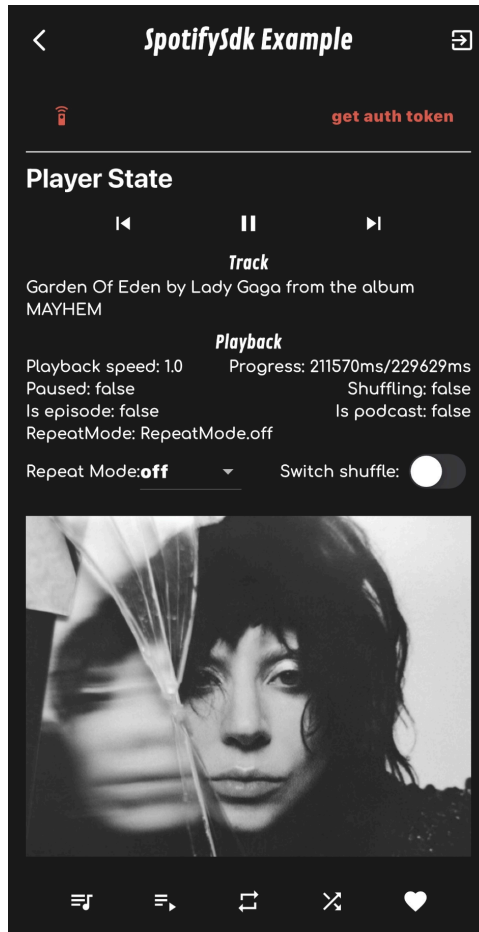
## Uploading Songs:

To upload your local songs, click on the blue “Choose Files” button. Then, select songs from your device’s storage. Once you have selected your songs, confirm your selection, and the app will begin analyzing your songs.



## Connecting to Apple Music or Spotify:

To connect the playlist to your Apple Music or Spotify playlists, select the service you wish to use and follow the prompts to grant the app access. Select the playlist you wish to pull songs from.



### Plugin example app

**DeveloperToken:**

**UserToken:**

**Status:** Instance of  
'MusicAuthorizationStatusAuthorized'

**CountryCode:**

**Subscription:**  
**MusicSubscription(canBecomeSubscriber: false,**  
**canPlayCatalogContent:false,**  
**hasCloudLibraryEnabled:false)**

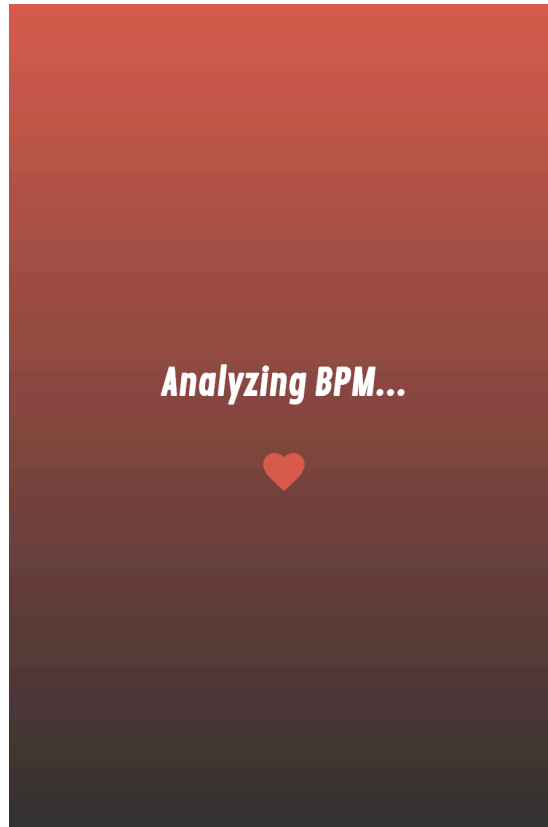
**PlayerState:** MusicPlayerPlaybackStatus.stopped  
**PlayerQueue:** null

**Shuffle**

**Request authorization**

## Loading Screen:

Wait for background processes to execute.



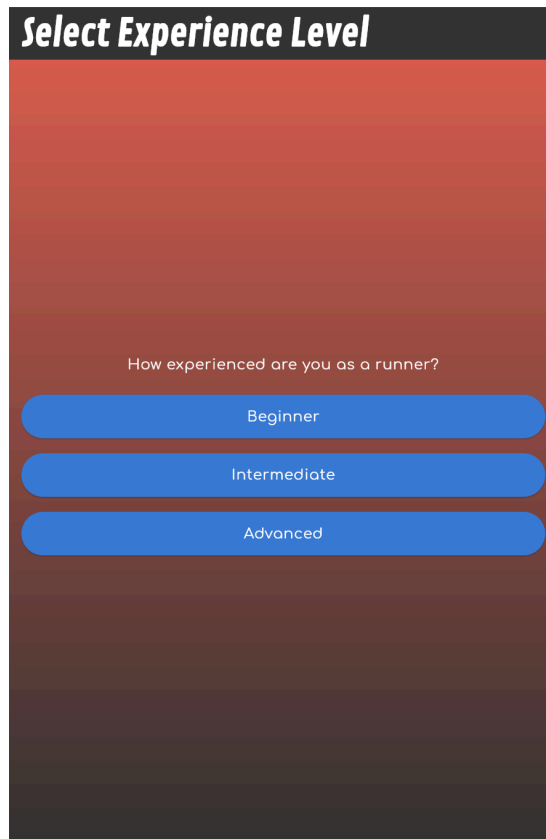
## Configuring Workout:

After uploading or connecting your music, you'll be guided through a few short setup steps:

### a) Select Your Experience Level

Choose the option that best matches your current experience level.

- Beginner
- Intermediate
- Advanced



**Select Experience Level**

How experienced are you as a runner?

Beginner

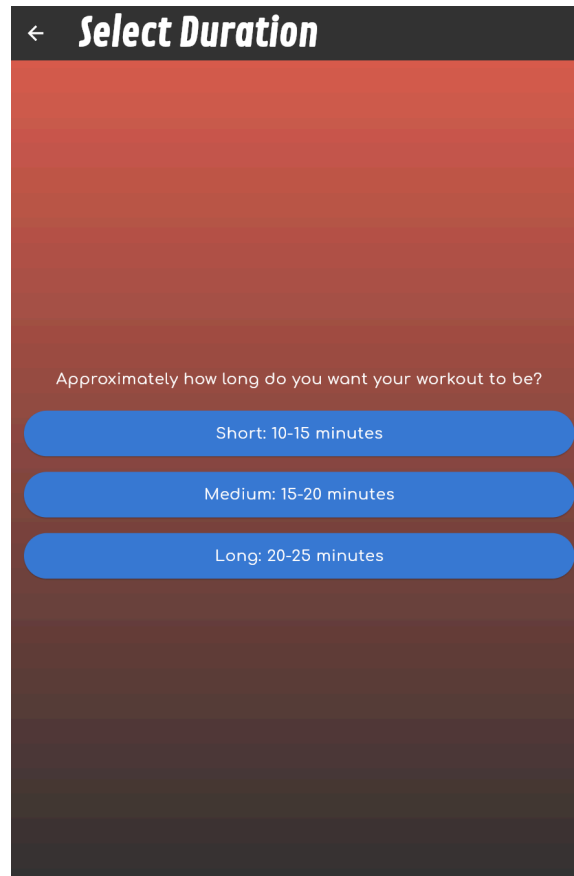
Intermediate

Advanced

## b) Workout Duration

Select how long you want to run.

- 10-15 minutes
- 15-20 minutes
- 20-25 minutes

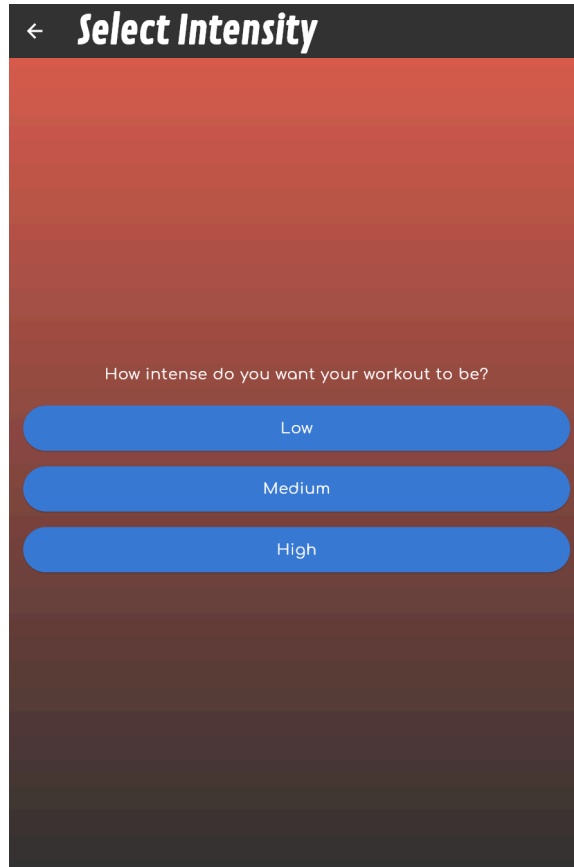


The screenshot shows a mobile application interface for selecting workout duration. At the top, there is a dark header bar with a white back arrow and the title "Select Duration". Below the header, the background is a gradient of red and orange. In the center, the text "Approximately how long do you want your workout to be?" is displayed. Below this text are three blue, rounded rectangular buttons stacked vertically. The first button is labeled "Short: 10-15 minutes", the second is labeled "Medium: 15-20 minutes", and the third is labeled "Long: 20-25 minutes".

c) Workout Intensity

Select how intense you want your workout to be.

- Low intensity
- Medium intensity
- High intensity



The image shows a mobile application interface for selecting workout intensity. At the top, there is a dark header bar with a white back arrow and the text "Select Intensity". Below the header, the background is a gradient of red and orange. In the center, the text "How intense do you want your workout to be?" is displayed. Below this text are three blue, rounded rectangular buttons stacked vertically, labeled "Low", "Medium", and "High" from top to bottom.

## Reviewing the Playlist:

The app generates a tentative playlist based on your choices.

Here you can:

- Swap the order of songs
- Delete songs
- Add suggested songs
- Customize the similarity and number of suggested songs

### Tentative Playlist

**Playlist**

Everything In Its Right Place by Radiohead

=

Simmer by Hayley Williams

=

**Suggested Songs**

The National Anthem - Radiohead

Kid A - Radiohead

Leave It Alone - Hayley Williams

Cinnamon - Hayley Williams

Suggested song similarity (least to most similar)

Number of suggestions per song (1-20)

2 Songs

Ready to finalize

✓ Finalize Playlist

If you are happy with the selection, press Finalize Playlist.

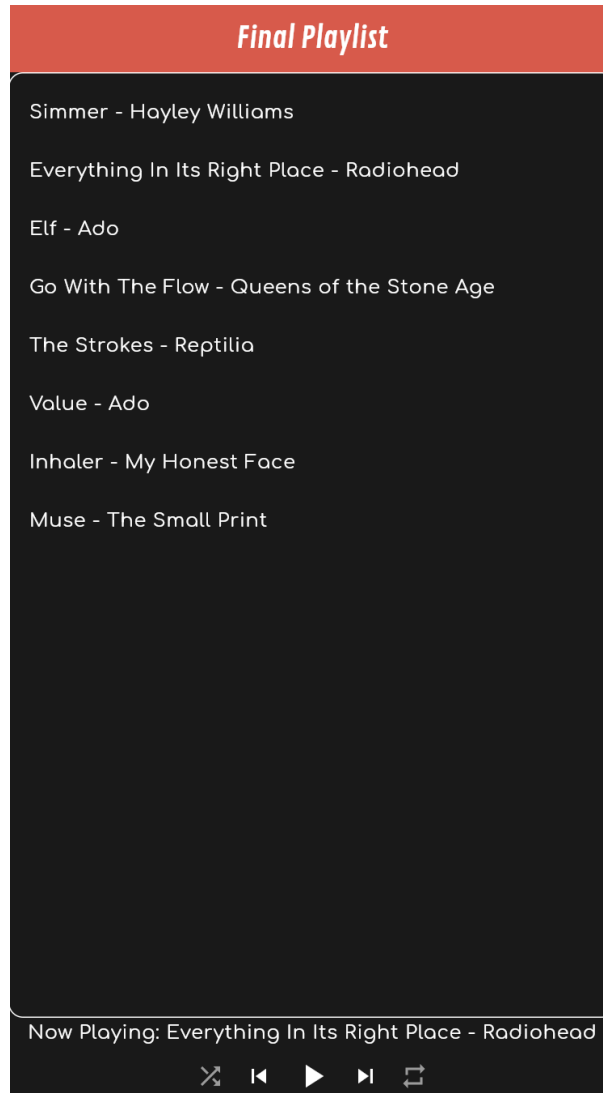


## Final Playlist:

Your Final Playlist is ready for your workout!

Here you can:

- Shuffle song order
- Go back to the previous song
- Play songs
- Skip songs
- Loop songs



# Technical Section

## System Deployment

Platform-Specific Deployment:

**For Android Development:** Make sure to have JDK v17 and NDK v27 or higher.

**For iOS Development:** Deployment only works for devices running iOS 15.6 or higher.

Spotify Setup:

Go to <https://developer.spotify.com/> and sign in, click “Create App,” then fill out all the required information and select the Web API, iOS, and Android, then get the Client ID and Client Secret and put them in an .env file in the assets folder as:

SPOT\_CLIENT\_ID and SPOTIFY\_CLIENT\_SECRET respectively

*Android SHA1 Fingerprint (necessary for Spotify SDK for Android):*

**For Windows Users:** Navigate to the project root folder, and paste the following command into the terminal:

```
keytool -list -v -keystore ".\android\debug.keystore" -alias  
androiddebugkey -storepass android -keypass android
```

**For macOS or Linux Users:** Navigate to the project root folder, and paste the following command into the terminal:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias  
androiddebugkey -storepass android -keypass android
```

**Bundle ID for both Platforms is:** com.hopperhackers.rhythmrn

**Redirect URI:** rhythmrn://callback

### GetSongBPM.com Setup:

Go to <https://getsongbpm.com/api> to get your API Key, make sure that the website you link has an href to <https://getsongbpm.com> and fill out all the relevant information, once you get the email, you should copy your API key and put it in an .env file in the assets folder as:

BPM\_API\_KEY

### Last.FM Setup:

Go to <https://www.last.fm/api/account/create> to create your account for the API, and get the shared secret and key and put it in an .env file in the assets folder as:

LASTFM\_API\_KEY and LASTFM\_SHARED\_SECRET respectively

### Building Codebase:

This presumes that you have already installed Flutter on your machine, if you need help with that go to <https://docs.flutter.dev/get-started/>

First, enter `flutter pub get` to install the dependencies for the app:

Then, use `flutter run` to run natively on Chrome or iOS using the Simulator app if you are running a Mac.

If you are using iOS and you encounter building issues, try using `cd ios` followed by `pod install` to install relevant packages, then try building again!

Otherwise, for all other platforms use `flutter build apk` to run natively on Chrome.

Lastly, if you decide to use some of the APIs and they fail to work, you can always use

`flutter run --dart-define-from-file=api-keys.json` and then put the api-keys.json in the root directory of the app by using the key-value pairs just like the .env file

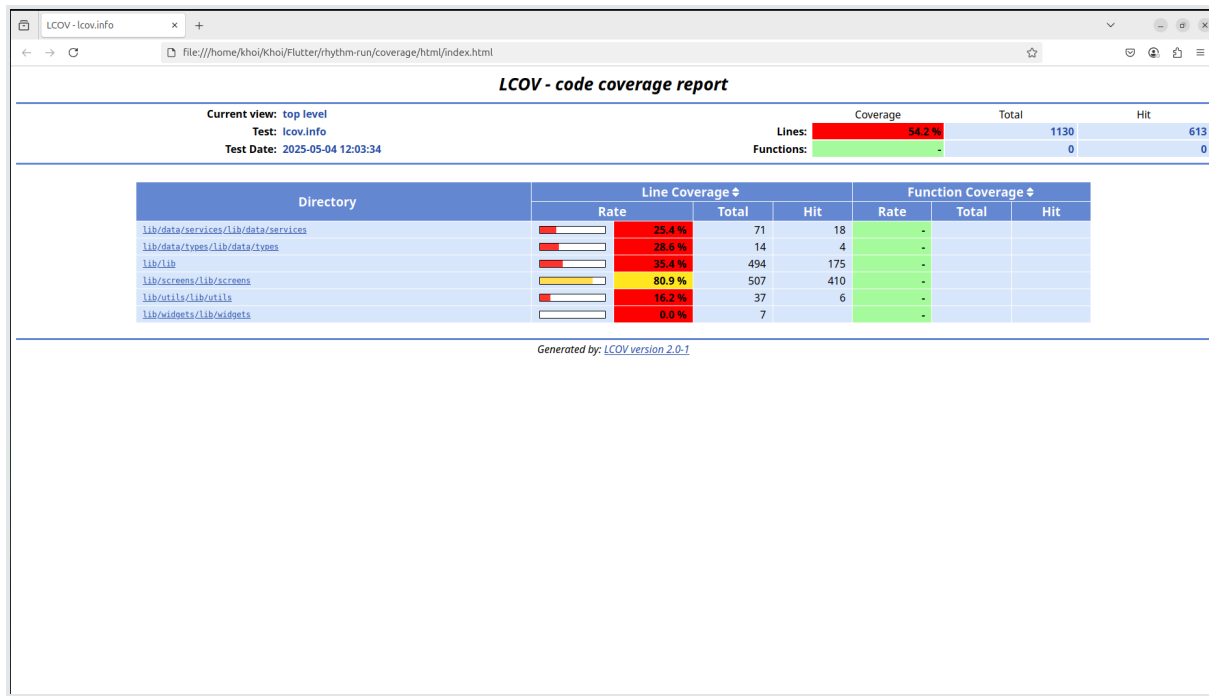
## Code Coverage

To test the codebase coverage, make sure that the test for each requirements is included (unit test) in the test folder of the project

To get the coverage report test, use this command line: `flutter test --coverage`

After running the command, `coverage/lcov.info` file is generated and to visualize the file, first install the tool on linux: `sudo apt-get install lcov`, then to generate on Html use the command: `genhtml coverage/lcov.info -o coverage/html`. Finally, run the `xdg-open coverage/html/index.html` to get a view on html visually.

Result:



## User Acceptance Testing

Check available emulators: `flutter devices`

Run the emulator: `flutter run -d <name of the platform>`

(e.g `flutter run -d android`, `flutter run -d ios`, `flutter run -d chrome`)

Interact with the buttons, box checks of the app.

Test Number	Test Case Name	Steps	Expected Behavior	Actual Behavior
1	Import the user's local music into the RhythmRun	1. Start the RhythmRun App	After the user opens the application, the landing page should transition	1. Success 2. Success, taken to local import page

	application	<ol style="list-style-type: none"> <li>2. Press the "Continue with Local Music" button on the landing page</li> <li>3. Press the "Choose Files" button.</li> <li>4. Start selecting .mp3 or .wav files from your local files or cloud storage</li> <li>5. Press the "Analyze BPM" button.</li> <li>6. You see a confirmation popup if it was successfully analyzed!</li> </ol>	<p>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.</p> <p>After selecting files, the user should see their selected music files when redirected back to the "Import Songs" page.</p> <p>If the user did not select any songs, then raise a error box that says "No songs were imported"</p> <p>Else, if the user selected songs for analysis, the user will be navigated to a loading screen and see a confirmation popup that they were successfully analyzed</p>	<ol style="list-style-type: none"> <li>3. Success, file system pops up</li> <li>4. Success, files selected</li> <li>5. Failure, no loading screen; still on local import page</li> <li>6. Failure, no confirmation</li> </ol>
2	Import the user's playlist from Spotify	<p>Pre-condition: You must have a Spotify Account and have the Spotify app installed on your device.</p> <ol style="list-style-type: none"> <li>1. Start the RhythmRun App</li> <li>2. Press the "Continue with Spotify" button.</li> <li>3. Redirects to the</li> </ol>	<p>After the user opens the application, and pressing "Continue with Spotify":</p> <ul style="list-style-type: none"> <li>• If the user has the Spotify app <ul style="list-style-type: none"> <li>○ If the user does not have an account, it will automatically show a dialogue box that says "Please Log in",</li> </ul> </li> </ul>	<ol style="list-style-type: none"> <li>1. Success</li> <li>2. Success, taken to Spotify page</li> <li>3. Success, taken to the Spotify App.</li> <li>4. Success, button shows up to connect with the App</li> <li>5. Success, redirect to the app</li> <li>6. FAIL, there is no current implementation of selecting playlists,</li> <li>7. FAIL, no pop up exists...</li> </ol>

		<p>Spotify app if the app is on the phone.</p> <ol style="list-style-type: none"> <li>4. Press "Agree" to allow Spotify to connect with RhythmRun</li> <li>5. Redirected back to the RhythmRun app.</li> <li>6. In the "Import Playlists" screen, select playlists from your Spotify account and then click the "Import" button</li> <li>7. You see a confirmation popup if it was successful!</li> </ol>	<p>with subtext that says "The link will open once you have logged in."</p> <ul style="list-style-type: none"> <li>○ If the user has an account, then Spotify's authentication screen should appear with the choice to either "Agree" or "Cancel"</li> <li>○ If the user has an account and agrees then it goes back to the RhythmRun application the user will see their playlists.</li> <li>○ If the user has an account and cancels the authentication, they are redirected back to the</li> </ul>	
--	--	---	---	--

			<p>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.</p> <ul style="list-style-type: none"><li>• 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<ul style="list-style-type: none"><li>○ If the user selects nothing then raise an error box that says "No playlists were imported"</li><li>○ If the user selects playlists then they should see a confirmati</li></ul></li></ul>	
--	--	--	---	--

			on popup	
3	Test the BPM analysis loading screen	Pre-condition: You have imported songs through either local storage, Spotify, or Apple Music. 1. 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.	1. Failure, the GIF runs properly, but the loading does not end
4	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 the desired duration 4. Click the "Next" button. 5. Select intensity level 6. Click the "Next" button.	The app should adjust the workout playlist based on the parameters inputted by the user.	1. Success 2. Success, taken to duration page 3. Success 4. Success, taken to intensity page 5. Success 6. Success, taken to tentative playlist page
5	Adjust workout playlist song order and selected	Pre-condition: You have finished all the previous steps	The playlist should reflect the desired playlist order.	1. Success, order of songs is changed 2. Success, parameters are



	suggested songs	<p>and a workout playlist has been generated.</p> <ol style="list-style-type: none"> <li>1. Click and drag your desired song to rearrange the song order.</li> <li>2. Drag the two sliders to customize the similarity and number of suggested songs</li> <li>3. Using the 'add' icon on a song, add a suggested song to the working playlist</li> <li>4. Using the 'delete' icon on a song, delete a song from the working playlist</li> <li>5. Finalize the playlist (save changes) by pressing the "Save" button .</li> </ol>	<p>When customizing the suggestion parameters, the user should be able to drag the sliders, and the page should update accordingly. The page should not be slow or freeze after a request is made.</p> <p>When adding suggested songs, the song should be removed from the suggested songs list and added to the working playlist.</p> <p>When deleting a song from the working playlist, the song should be removed from the working playlist and added into the suggested song list, in case the user decides to add it back in.</p> <p>When finalizing, the user's choices should be stored and the user should be navigated to the final page.</p>	<p>able to be changed and suggested songs are present and reflect those changes</p> <ol style="list-style-type: none"> <li>3. Success, the selected song is removed from the suggested playlist and added to the tentative playlist</li> <li>4. Success, the selected song is removed from the tentative playlist and added to the suggested playlist</li> <li>5. Success, taken to the final playlist screen</li> </ol>
--	-----------------	--	--	--

6	Pause and resume workout	<p>Pre-condition: A workout playlist has been generated and finalized by the user.</p> <ol style="list-style-type: none"> <li>1. Click the "Pause" button.</li> <li>2. Click the "Play" button.</li> </ol>	<p>The current workout is paused after pressing "Pause", and the current song stops playing.</p> <p>By pressing the "Play" button, it resumes the workout and the most recent song starts playing again</p>	<ol style="list-style-type: none"> <li>1. Success, "pause" button turns into a "play" button, and the music stops playing</li> <li>2. Success, "play" button turns into a "pause" button and the music resumes playing</li> </ol>
7	Skip songs	<p>Pre-condition: A workout playlist has been generated and finalized by the user.</p> <ol style="list-style-type: none"> <li>1. Click the "Skip" button (right facing).</li> </ol>	<p>Current song is skipped and the next song is played in queue, if this is the last song, the next song will be the first song in the playlist.</p>	<ol style="list-style-type: none"> <li>1. Success, the current song stops playing and the next song starts playing. In the case of it being the last song, the first song plays once the skip button is pressed.</li> </ol>
8	Go back to previous song	<p>Pre-condition: A workout playlist has been generated and finalized by the user.</p> <ol style="list-style-type: none"> <li>1. Click the "Previous" button.</li> </ol>	<p>Current song should be the previous song and should start playing after clicking the button</p>	<ol style="list-style-type: none"> <li>1. Success, current song is stopped and the previous song plays</li> </ol> <p>A note—in the case of shuffle being active this is a failure, as the song that is previously listed will play, as opposed to the song that has just played</p>
9	Loop song	<p>Pre-condition: A workout playlist has been generated and finalized by the user.</p> <ol style="list-style-type: none"> <li>1. Click the "Loop" button.</li> <li>2. Wait for the song to finish</li> </ol>	<p>Current song should start playing again once the song has finished.</p>	<ol style="list-style-type: none"> <li>1. Success, once the song has reached its end, it plays again</li> </ol>

		playing		
10	Shuffle songs	Pre-condition: A workout playlist has been generated and finalized by the user. 1. Click the "Shuffle" button.	Songs should play in a randomized order.	1. Success, songs play in a random order

## Stakeholders

Invited to multiple users using Android and iOS to test the app that are either student athletes or interested members of the campus community through TestFlight for iOS, or by supplying the .apk file for Android users.

## Implemented Design

getSimilarSongs(song, lowerThreshold, limit)

- For this function, it would have been nice to have been given the last.fm API from the start to find similar songs as Spotify's Similar Songs API call was deprecated as of last year
- Pre-conditions: Check if the API Key and Shared Secret from last.fm are provided and that the HTTP Response is 200. Also, it checks each similar track to see if the lowerThreshold given by the last.fm API is above or at the threshold.
- Post-conditions: Returns a Playlist with all the similar songs that fit the criteria that the threshold and limit of songs was set as. Otherwise, it will replace songs that were not found as empty strings for the name and artist.

generate\_playlist(xp\_lvl, intensity, duration)

- This function relies on 'songs.db,' our song database
- Pre-conditions: ensure the inputs (xp\_lvl, intensity, duration) are of the correct form (xp\_lvl: "beginner", "intermediate", or "advanced", intensity: "low", "medium", or "high", and duration: "short", "medium", or "long"), and ensure that 'songs.db' exists
- Post-conditions: Returns a list of songs (tuples) of the form (int ID, String song\_name, int song\_BPM, double song\_length), where song\_length is in minutes, that satisfies the BPM range dictated by 'xp\_lvl' and 'intensity', and the duration dictated by 'duration'

### **BPMAnalysis Class for Streamable Songs:**

getBPM(self, artist\_name, track\_name)

- This function relies on two different methods for obtaining the BPM of streamable songs: GetSongBPM.com API and the “Unofficial” Tunebat API
- Pre-conditions: None
- Post-conditions: Runs through all the methods and returns the BPM of the song once a BPM is found or if Tunebat gives the BPM

searchTuneBat(self, artist\_name, track\_name)

- This function relies on the Tunebat API, which has a maximum of about 15 calls, until it needs to take a minute break, this was actually adapted from this GitHub repo: <https://github.com/TheBrenny/tunebat-api/>. Unlike the other APIs, this one can be the most unreliable and it has issues with cloudflare blocking requests..
- Pre-conditions: None
- Post-conditions: Calls the Tunebat API and checks if the response is OK and returns a json of the response, and raises errors if not.

searchGetSongBPM(self, artist\_name, track\_name)

- This function relies on GetSongBPM.com API, which actually has an API Key unlike the other platform being used.
- Pre-conditions: None
- Post-conditions: Calls the GetSongBPM.com API and checks if the response is OK and returns a json of the response, and raises errors if not.

getBPMfromSongBPM(self, artist\_name, track\_name)

- Pre-conditions: Checks if the BPM is in the existing database, and if not it checks if the results are in the correct list format.
- Post-conditions: Returns the BPM from the API Call if successful, otherwise raises errors or returns nothing if not results were found.

getBPMFromTuneBat(self, artist\_name, track\_name)

- Pre-conditions: Checks if the BPM is in the existing database, and if not it checks if the results are there, or some rate limiting happened with CloudFlare
- Post-conditions: Returns the BPM from the API Call if successfully matches the song or a close enough match, otherwise raises errors or returns nothing if no results were found.

### **SongDatabase Class for Streamable Songs:**

*It is worth noting that this class will likely be replaced with Supabase instead of SQLite that will talk with both the node.js backend for BPM analysis and Flutter...*

\_create\_table(self)

- Pre-conditions: Checks if the SQL Table already exists

- Post-conditions: Creates a SQL Table for the songs

save\_song(self, artistName, trackName, BPM)

- Pre-conditions: None
- Post-conditions: Adds the song to the database

fetch\_all\_songs(self)

- Pre-conditions: None
- Post-conditions: Gets all the Songs in the database

fetch\_song(self, artistName, trackName)

- Pre-conditions: None
- Post-conditions: Gets all the Songs in the database

### **Spotify Interface (API)**

*This interface is still a Work in Progress (WIP), but for now it has SDK functionality, it will likely have Web API functionality to access user's playlists. Therefore, this interface primarily works in tandem with the user's Spotify app installed on their device to control playback, set queues, etc.*

initSpotify()

- Pre-conditions: None
- Post-conditions: Tries connecting to Spotify and gets Authentication from the user to access their Spotify account details

setMusicQueue(song)

- Pre-conditions: Checks if the current queue is not empty
- Post-conditions: Adds the song into the queue using the Spotify URI from the StreamableSong object.

addSongToLibrary(song)

- This function should usually be called if the user likes a song during a run and wants to add it to their library
- Pre-conditions: None
- Post-conditions: Adds the song to the User's Library

togglePlayback()

- Pre-conditions: Checks if the Spotify player is on or off and if the player is currently paused.
- Post-conditions: If paused and the player is on, then it will resume the music, otherwise if the player is on and the song is not paused, then it will pause the song

skipSong()

- Pre-conditions: None
- Post-conditions: Skips the song to the next song in the queue

shuffleSongs()

- Pre-conditions: None
- Post-conditions: Toggles between the shuffle being off or on

shuffleSongs()

- Pre-conditions: None
- Post-conditions: Toggles between the shuffle being off or on

repeatSongs(repeat)

- Pre-conditions: None
- Post-conditions: Toggles between turning off repeat mode, repeating the current track, or repeating the current “context” (basically the playlist or album that is currently playing)

connectToSpotifySDK()

- Pre-conditions: None
- Post-conditions: Connects to Spotify’s SDK service

getAuth()

- Pre-conditions: None
- Post-conditions: Gets the access token from the user through the Spotify app to use the rest of the functions listed above.

### **Local Storage:**

*The local import process is done so far and works functionally. Basically adjust to the platform local permission for android, however the framework called file picker handles for ios for accessing to local storage, then pick the multiple .mp3 files or .wav files.*

importLocalPlaylist(localPlaylist)

- Pre-conditions: localPlaylist contains the valid song files as .mp3,.wav
- Post-conditions: The system app receives a copy from the localPlaylist.

importLocalStorageButton()

- Pre-conditions: The users have selected the music file to import.
- Post-conditions: it activates the importLocalPlaylist method like the local playlists are going to the system.

### **UI Widgets:**

*Listed below are all the pages that the user can be navigated to while using the RhythmRun app.*

#### WelcomePageWidget

- Acts as the home page for users opening the app
- Pre-conditions: The app has been initialized
- Post-conditions: Displays a welcome message and a button or action that allows users to proceed to import songs. On interaction, the user is navigated to LocalSongsUploadPage, or either of the API authentication pages.

#### LocalSongsUploadPage

- Allows users to upload local audio files from their device.
- Pre-conditions: The app has file system access and permission to read audio files from the user's device. User has reached this page via the welcome screen.
- Post-conditions: Displays the selected files. Once files are uploaded, the audio files are stored (unimplemented). After finalizing, the user is directed to LoadingPage.

#### LoadingPage

- Informs the user that files are being processed as processes work in the background.
- Pre-conditions: User has uploaded local files. User has reached this page via the local songs upload page, or either of the API pages.
- Post-conditions: After analysis completes, the user is automatically routed to ExperienceLevelPage.

#### ExperienceLevelPage

- Collects user input about their running experience level.
- Pre-conditions: The user has been navigated here from the loading screen.
- Post-conditions: Stores the user's experience level ("beginner", "intermediate", "advanced") (unimplemented) and navigates them to DurationPage.

#### DurationPage

- Collects user input about desired duration.
- Pre-conditions: The user has been navigated here from the experience page screen.
- Post-conditions: Stores the chosen duration ("short", "medium", "long") (unimplemented) and navigates the user to IntensityPage.

#### IntensityPage

- Collects user input about desired intensity.
- Pre-conditions: The user has been navigated here from the duration page screen.
- Post-conditions: Stores the chosen intensity ("low", "medium", "high") (unimplemented) and navigates the user to TempPlaylistPage.

#### TempPlaylistPage

- Displays and allows the user to customize a preview playlist generated based on the user's selected experience level, duration, and intensity.
- Pre-conditions: The playlist has been generated via `generate_playlist()` using the stored user preferences. The user has navigated to this page from the intensity page screen.
- Post-conditions: Shows a list of suggested songs obtained by calling `getSimilarSongs()` with the appropriate arguments. Provides options to add suggested songs, and remove songs in the tentative playlist. When the user finalizes their choice, the playlist will be passed on to `FinalPlaylistPage` (unimplemented).

#### FinalPlaylistPage

- Shows the final playlist the user has finalized and allows playback, skipping forward and backward, shuffling, and looping.
- Pre-conditions: The final playlist has been passed in from `TempPlaylistPage`. The user has navigated to this page from `TempPlaylistPage`.
- Post-conditions: The final playlist page is rendered with the final playlist and the options to play/pause, skip forward, go back to the previous, shuffle songs, and loop songs.