# INDY-02-PlaylistSyncer

## SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

CS4850-02

Spring 2024

Professor Perry

2/13/2024

Team

| **Roles** | **Name** | **Major responsibilities** | **Cell Phone / Alt Email** |
|---|---|---|---|
| Team leader | Nikita Smith | Developer, Manager | 678-628-6076<br>Nikitasmith6@gmail.com |
| Team members | Ben Pallotti | Documentation, Developer | 770-906-3367<br>ben.pallotti@gmail.com |
| | Josh Poore | Programmer, Developer | 770-337-5477<br>joshua.poore@gmail.com |
| Advisor / Instructor | Sharon Perry | Facilitate project progress; advise on project planning and management. | 770-329-3895 |

# Table of Contents

# 1.0 Introduction

## 1.1 Overview

The goal of this document is to assist in defining the software requirements for our PlaylistSyncer mobile app. PlaylistSyncer will allow mobile users to connect to their music streaming platforms and synchronize playlists between platforms. Google's Flutter framework for cross-platform development will execute this task. This document will outline the functional, non-functional, and external interface requirements necessary to develop and launch this software.

## 1.2 Project Goals

As a team our goal is to support a select group of streaming services including Apple Music, Spotify, and YouTube Music. Users with accounts on these platforms can access and transfer playlists among their associated accounts. Additionally, users can review generated playlists prior to finalizing the transfer. During the playlist review process, users receive notifications that if any content from their playlist is unavailable on the platform, they plan to transfer the playlist to. This aids in accomplishing our goals of providing a transparent and smooth experience for users.

## 1.3 Definitions and Acronyms

This section provides definitions for each term and acronym required to interpret this SRS.

### 1.3.1 Definitions

OAuth: An authorization protocol that allows for the request and retrieval of app access tokens.

### 1.3.2 Acronyms

2FA: 2-Factor authentication

API: Application Programming Interface

Mb: Megabyte

## 1.4 Assumptions

Our program relies on the primary assumption that we can access and create data across several streaming platforms. Our app requires each streaming platform to service requests for querying data from their streaming music libraries and authorized user playlists. Additionally, we are operating under the assumption that each platform must be able to service our requests to create playlists based on gathered information from other platforms. Another assumption we're making is that artist and music data are similar enough across each supported platform to interpret and generate platform-specific data properly. While these assumptions could be wrong, after extensive research, we believe that our presuppositions are safe and accurate.

# 2.0 Design Constraints

Our vision for this application operates under a few key constraints and a set scope regarding our environment, user characteristics, and system constraints.

## 2.1 Environment

Our scope is limited by a few factors, especially time and scale constraints. Supporting only three streaming services acts as accurate proof of concept and is an ideally realistic expectation given the

limited amount of time to complete the project. Also, supporting multiple platforms is challenging because of the relatively quick development timeline. Limiting the app to Android and iOS allows us to access a large market while maintaining a small list of supported platforms. This platform limitation is also supported by our usage of the Flutter framework, which allows us to construct Android and iOS applications simultaneously.

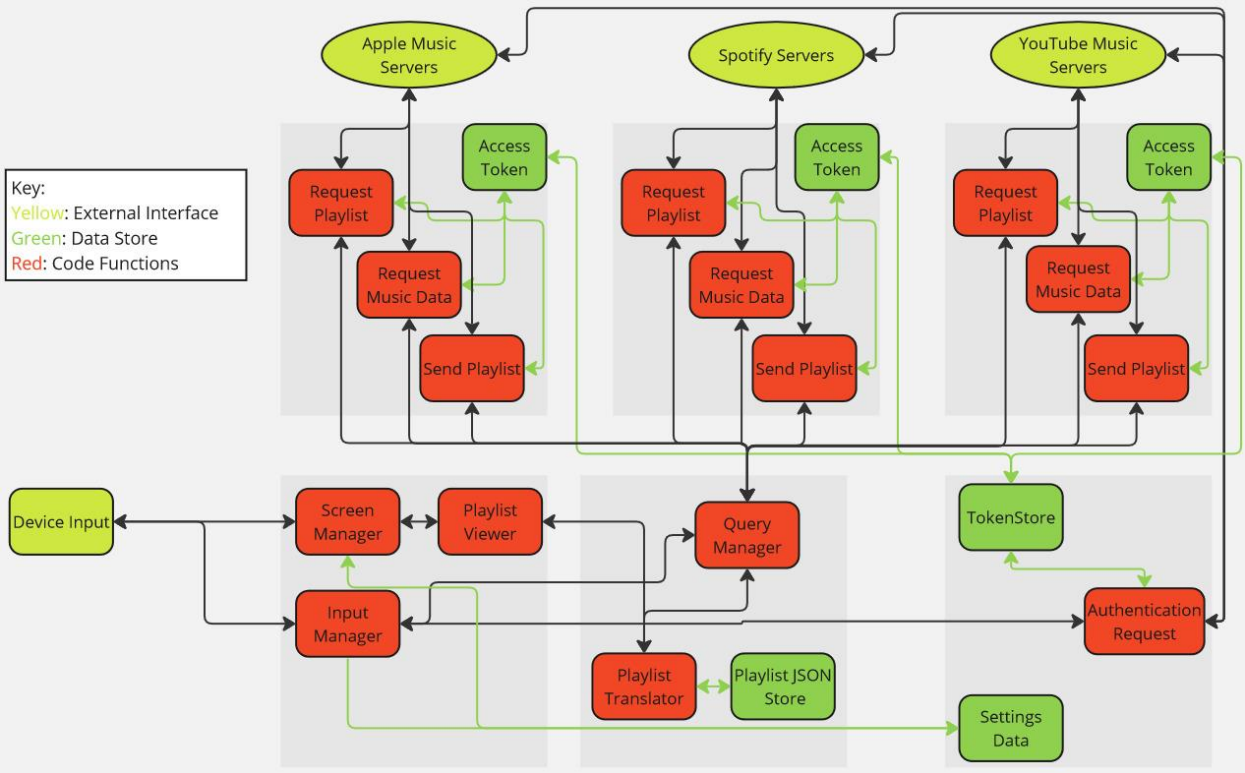
## 2.2 User Characteristics

Our belief is that our users are constantly burdened with expensive, unreliable, and limited streaming platforms. Often, these users remain on the platforms solely because of all the playlists and content they saved as they used the platform. By offering an app that allows users to transfer their content across the three most popular platforms, our scope appropriately services their needs without bloat or unnecessary features. This represents our ideal user; each user can log in to the three music platforms on our application and transfer their playlist data from one to another whilst using the app.

## 2.3 System

Our underlying system powering the app should act as a universal interpreter that handles response data from any of our supported platforms and adequately translates the data into the destination platform's supported format. This means interpreting platform-specific content IDs, artist names, song names, and other relevant playlist information to generate platform-specific playlists. Our system will primarily rely on artists and song names to retrieve streaming service information, as we believe this is the most reliable and consistent data between platforms.

# 3.0 Functional Requirements




*System Architecture Diagram*

## 3.1 Example

Create a working prototype that shows how to use the Flutter app to move playlists from Spotify to Apple Music, Apple Music to YouTube, etc. The playlist selection, successful transfer, and login procedure should all be demonstrated in this prototype.

## 3.2 Authentication

Secure login should be guaranteed with a simple Username and Password login, and 2FA will be considered. Authentication mechanisms such as OAuth should be implemented for secure login. OAuth is a standardized authorization protocol or framework enabling applications to gain secure designated access, so you can give the Playlist Sync App permission to access YouTube, Spotify, and Apple Music.

## 3.3 Display Home Page

The user is prompted to log in. Upon login, users should be directed to a home page where they can see options to access their playlists on each platform to make a centralized entertainment platform.

## 3.4 Navigate Screens

Users should be able to navigate the app seamlessly through the login, transfer, and setting screens. Screen-to-screen navigation is handled by pressing easy-to-locate buttons from the login screen to the transfer screen, and then to the settings screen. Users can also move in reverse, moving from the settings screen back to the transfer screen, and so on.

### 3.5 API and Conversion

Data must be accessible so it may be fetched from the music platform APIs and then converted both to and from a universal JSON data type. This is to ensure that the data is compatible for all the possible music platforms that we are giving data to.

### 3.6 Constraints

1. API Limitations - Each streaming platform likely has limits on the number of API requests per second. In accordance with this issue, our app shall limit requests to avoid being blocked out of calling to the music platform APIs.
2. User Experience – The app should be simple, as its goal is to create an effortless, convenient experience. It should contain minimum buttons/icons/menus that are aesthetically pleasing.
3. Large Amounts of Data – There may be difficulty in finding the most efficient way to transfer and store data from the playlists. Our strategy is centered around storing data locally using JSON files. This data would be used for storing user settings and temporary playlist data, as playlist-specific data should only be stored locally while a transfer is in progress.

## 4.0 Non-Functional Requirements

### 4.1 Security

We will need to create or facilitate some encryption of user data, specifically their login details for their music platform accounts. This could be accomplished via some of the APIs for these music platforms or through encryption of our own creation if the API does not encrypt said data.

### 4.2 Capacity

The capacity of this application should not have to be too large. Approximately 500Mb in storage should be more than enough to store user login information and playlists. This is our ideal upper limit regarding how much device storage is required to store the app and any associated persistent data.

### 4.3 Login and Password

Users will need to create an account for the playlist sync app. They should be able to log in to their Apple Music, Spotify, and YouTube accounts securely through the app. Login information can be deleted through the settings menu if requested by the user.

### 4.4 Usability

1. The user should be able to access all application functions after signing up with at least two offered music platforms.
2. The screen after a user has logged in should be the transfer screen.
3. It should take no longer than 500ms to transfer a playlist from one platform to another (This estimate does not apply to transferring a playlist to all the platforms). This assumes a stable connection to a 5G network while initiating a transfer.

### 4.5 Other

1. An option in the settings menu that allows users to change the theme of the app between a few basic themes.

# 5.0 External Interface Requirements

## 5.1 User Interface Requirements

1. Buttons or interactable pieces of the UI should be easily pressable by the human finger.
2. A drop-down menu that enables the user to select which platform they would like to take playlists from.
    2.1. A checklist allows the user to select multiple platforms to move the playlist to. (excluding the one that is providing the playlist)
3. A settings button in the top left corner of the screen.
4. The font we plan to use for the application is Ageo.

## 5.2 Hardware Interface Requirements

1. The supported devices for this application are Apple and Android devices.
2. To read input, our program will require a touch screen.
3. Our program requires Wi-Fi or Celler Data network capability and connection to work correctly.

## 5.3 Software Interface Requirements

1. This software was developed for Android and iOS.
2. This software connects and utilizes the API calls for Apple Music, YouTube Music, and Spotify.
3. We use Dart to execute our backend code, and Flutter is our development environment.
4. Virtual keyboard input handled by the native OS.

## 5.4 Communication Interface Requirements

1. Communication with Streaming Platform servers via Cellular/Wi-Fi networks
2. APIs (YouTube, Spotify, Apple Music)
3. Databases
4. Flutter

# APPENDICES

**API Links**

YouTube Music: https://pub.dev/packages/youtube_data_api

Apple Music: https://pub.dev/packages/music_kit/example

Spotify Music: https://pub.dev/packages/spotify_sdk

**Development Documentation Links**

Flutter Documentation Page: https://docs.flutter.dev/