INDY-02-PlaylistSyncer
FINAL REPORT

CS4850-02
Spring 2024
Professor Perry

3/20/2024

Lines of Code: 828

Number of Files: 6

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

GitHub Link: https://github.com/Indy02-CS4850

Website Link: https://indy02-cs4850.github.io/PlaylistSyncer.github.io/

Presentation Video:

Table of Contents

1. INTRODUCTION

This document will explain the development process of our web application, Playlist Syncer, including the challenges we faced when creating this as a mobile application. This explanation will include information found on previous documents submitted for this project and how it has changed or was put into effect. We will detail our progress and changes regarding the application's architecture, frameworks explored, and the outcomes of our efforts in creating a playlist synchronization platform. The goal of this document is to give insights into the development process, how we made key decisions, the lessons learned from our trials, and the final product.

1.1 Challenges and Major Changes

We felt it was necessary to discuss the challenges and any major changes in scope early on in this document to preface the many changes from our original scope and design. The main challenges that caused us to change from developing this as a mobile application to a web application was integrating our JavaScript music platform authorization code with our Flutter application.

2. REQUIREMENTS

This section will cover our project requirements that we have laid out in the past and cover if any of them have changed or not been met.

2.1. Functional Requirements

2.1.1. 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.

This requirement has slightly changed to accommodate the removal of YouTube music but is otherwise met in full.

2.1.2. 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.

This requirement was met in full as we decided to use each platform's own website to login in our users, which allowed them to safely log in to their music accounts. **May be more to this.**

2.1.3. 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.

This requirement has changed but is still ultimately met. Our application has a home page that has separate tabs that allows the user to login/authenticate on one tab and go through the transfer process on another tab.

2.1.4. 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.

This requirement is met and improved. Instead of buttons we use tabs that the user can click to move from screen to screen with each screen having its own purpose.

2.1.5. 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.

This requirement has been met and the transfer process is developed completely.

2.1.6. Constraints

1. API Limitations - Each streaming platform 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.

Addressing the API Limitations, we found out that unless a user has a playlist of 10,000 or more songs neither API should have an issue with the amount of API calls a transfer makes. The user experience is simple and easy to understand. We also did end up using JSON to locally store user playlist data. Each of these constraints were met and dealt with accordingly.

2.2. Non-Functional Requirements

2.2.1. 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.

This non-functional requirement was met, but in phase two we plan to go heavily into this requirement as a good portion of our application data could be better protected, such as API client credentials for both Apple and Spotify.

2.2.2. 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.

This non-functional requirement was negated by our pivot over to a web application so local storage is not really an issue anymore, other than caching some of our data within the user authentication websites (i.e., the login websites will remember who logged in for a period after the user has logged in).

2.2.3. 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.

This was originally a non-functional requirement, but we removed it because we do not plan to monetize our application, and it would only complicate our system/application. Like we have said in the past, we want our application to be simple and easy to use, so anything that gets in the way of the main transfer process is not necessary we have gotten rid of.

2.2.4. 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.

This requirement is still the same as before the pivot, with the only non-met requirement being that the user press the tab to move to the transfer screen, but we felt it was close enough to the original that it was passable.

3. Design

3.1. Section Explanation

This section of the report will be about our current design, although we will point out if there are any major differences from what our design used to be before pivoting.

3.2. Architectural Strategies

3.2.1. Programming Language/Framework

For the programming language and framework, we initially intended to use Flutter for our mobile application. However, due to the challenges faced in integrating JavaScript music platform authorization code with Flutter, we transitioned to developing a web application instead. Our web application is built using a combination of HTML, CSS, and JavaScript for the frontend, with Python and Flask for the backend.

3.2.2. API Integration

We used the APIs provided by Apple Music and Spotify to access user playlists and allow playlists to be transferred between the two platforms. The APIs authenticate users and retrieve the playlist data securely.

3.2.3. Data Management

The data that is being handled involves the data fetched from each music platforms' API and then converting it to JSON. We can store this data locally. JSON files are used to store user settings, and playlist data. The playlist data will only be stored temporarily. It should be noted that JSON was chosen to ensure compatibility across all music platforms.

3.2.4. User Interface

The user interface of our web application is designed to be simple and aesthetically pleasing. We use separate tabs for distinct functions, allowing users to navigate between login/authentication, playlist transfer, and settings screens. Minimalistic design elements such as buttons and menus are used to simplify user experience.

3.2.5. Error Handling

Error messages are used to address both expected and unexpected errors. These messages are used to guide the user to the correct result. This is important to ensure that any issues that come up during the data transfer or authentication process are dealt with properly.

3.2.6. Concurrency/Synchronization

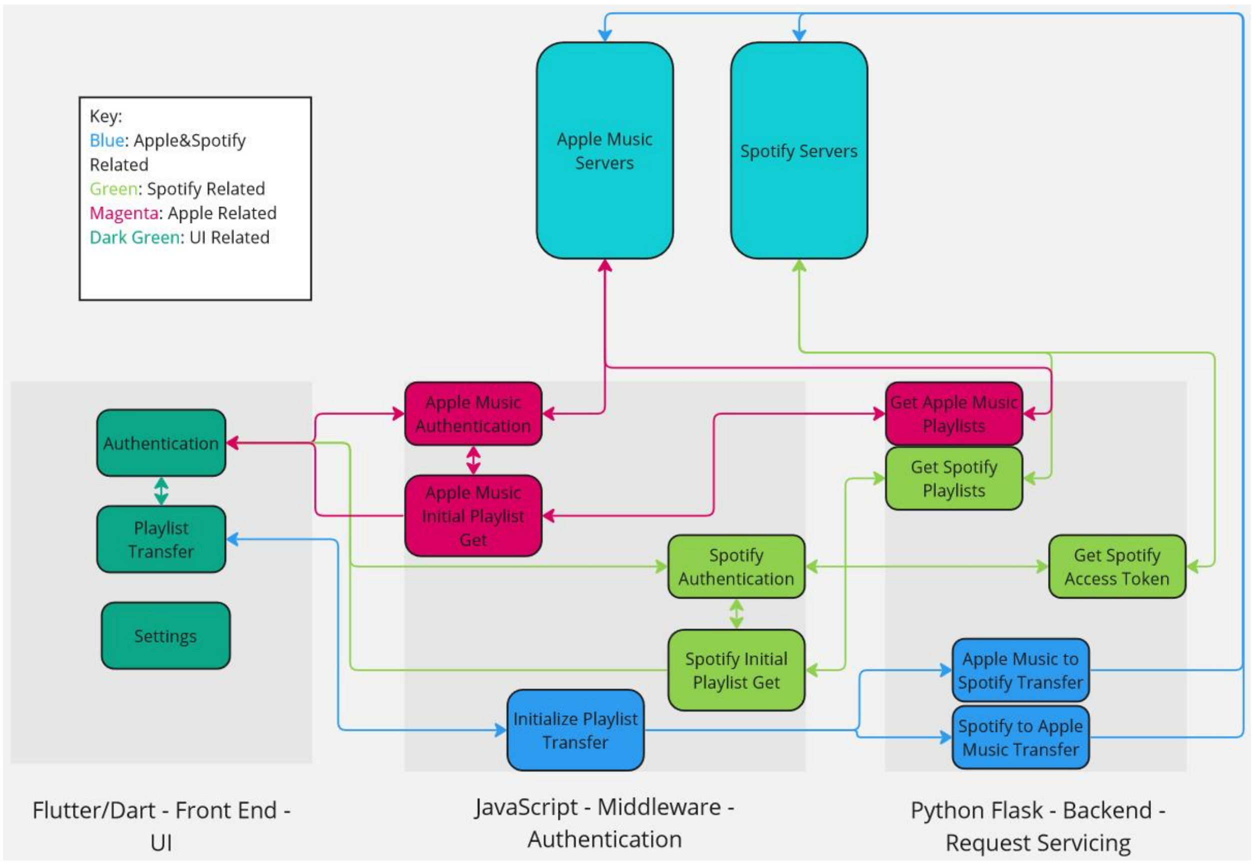
It is difficult to handle concurrency and synchronization efficiently due to user requests potentially compromising performance. The flask server should be efficient to sustain user requests at the expected volume of requests.

3.2.7. Version Control

Git is used to handle version control via the repository hosted on GitHub. This will allow for easy collaboration, synchronized version tracking, and updates. This is how we can maintain code integrity and maintain continuous alterations.

3.3. *System Architecture/Information Flow*

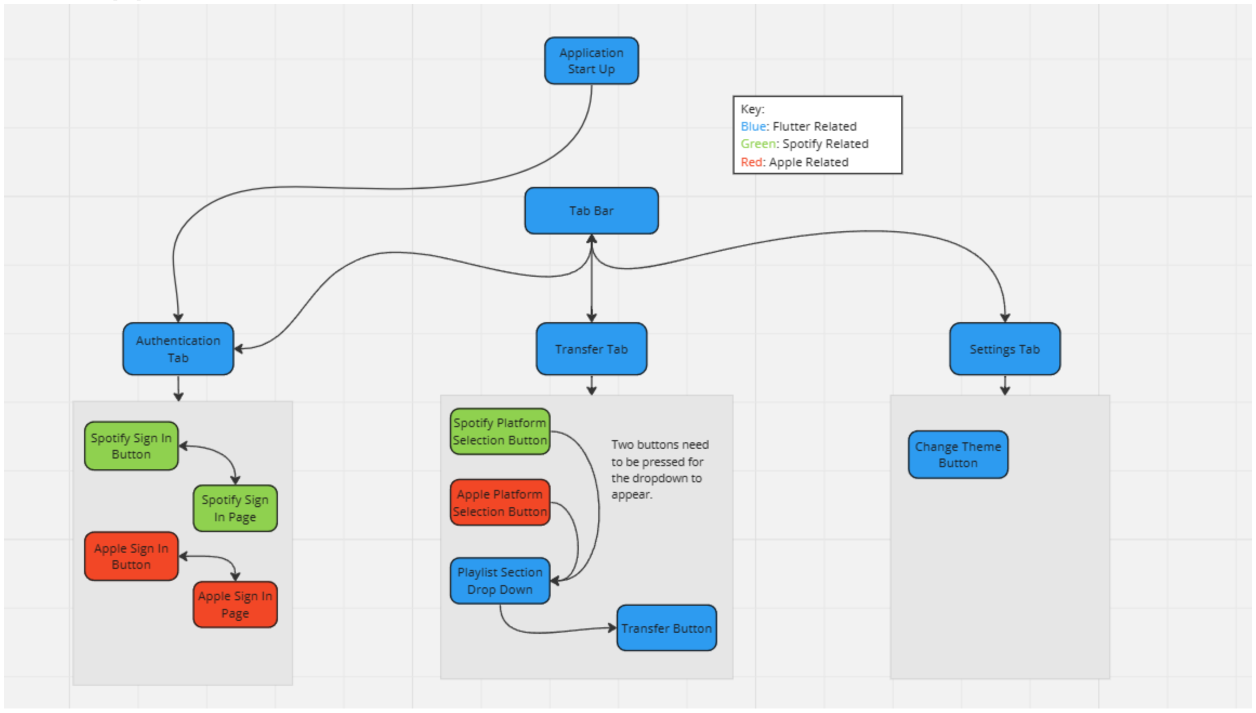
We use a client-server model. The client side uses components that make up the UI; these were developed with JavaScript and HTML. On the server side, we use Python to develop scripts to fetch data and Flask to handle user requests, authentication, and the playlist data transfer.


Key:
Blue: Apple&Spotify Related
Green: Spotify Related
Magenta: Apple Related
Dark Green: UI Related
Apple Music Servers
Spotify Servers
Authentication
Playlist Transfer
Settings
Apple Music Authentication
Apple Music Initial Playlist Get
Spotify Authentication
Spotify Initial Playlist Get
Initialize Playlist Transfer
Get Apple Music Playlists
Get Spotify Playlists
Get Spotify Access Token
Apple Music to Spotify Transfer
Spotify to Apple Music Transfer
Flutter/Dart - Front End - UI
JavaScript - Middleware - Authentication
Python Flask - Backend - Request Servicing

System Architecture Diagram

The diagram above shows our current system architecture and information flow. Each platform has its own authorization flow, and although this diagram simplifies it, after the authentication process to get the initial user access key, the two platforms function identically when it comes to Api calls. We use JavaScript to handle the authentication of both platforms, as it allows us to easily make web requests to the Apis. Then the user access token is given to the flask server, which allows us to pull user data through python Api requests. We decided it would be easier to implement this part of the project in Python because of the ease of data manipulation that Python provides. Once this data is gotten, the JavaScript code that we ran to get that data from the Flask server sets the state of that window to the data and runs the "read data" function that is in the Flutter section again. This allows the Flutter code to pull the data from the JavaScript window state so that we can display the playlists in our dropdown. This allows the user to then select the playlist that they want to transfer via our dropdown so they can initiate the transfer process, which is also handled by the Flask server. That is our system architecture design and information flow.

3.4. Application Flow



Application Flow Diagram

When a user first opens our application, they will begin on the authentication tab. There are two buttons that allow the user to authenticate with each of our currently provided music platforms. Spotify's button will redirect the website to their sign in website when the button is pressed and redirect back to our web application once the sign in process is complete. Apple Music's button will create a pop out of their sign in website that will disappear once the user has signed in. After the user has signed into both platforms, they can move over to the transfer tab to start the playlist transfer process. Once they get to the transfer tab, they will see two buttons and a prompt allowing them to select which platform they wish to transfer from. After a button is clicked, the prompt will change to ask which platform they wish to transfer to. Once another button is clicked, a dropdown menu will appear along with a button to initiate a transfer. The dropdown menu is populated with the user's playlists from the platform they wished to transfer from. Once they have chosen the playlist they wish to transfer, they simply hit the transfer button and the process starts. The final tab is not implemented but serves to allow the user to change the theme of our website if they do not like the default color scheme or want to have it in light mode.

4. DEVELOPMENT

4.1. Preprogramming Phase

After the idea was decided upon and approved by Dr. Perry, we began the project by researching what music platforms would work best for our application based on many factors, including popularity and accessibility of each platform's APIs. Once we decided which APIs would work best for us, we began considering how we could connect the APIs to our backend server and then how to create an appropriate front-end interface.

4.2. Programming Phase

This phase was the challenging work of the project, and many things that were done in this phase had to be done in a specific order, which is as follows.

1. We needed to create a basic frontend that would allow us to test the rest of our code
2. We needed to be able to authenticate users via our JavaScript code so that we could read and manipulate their data
3. We needed to create a sync function that would allow us to make a universal JSON of playlist data that we could convert to match the specifications for any of the music platforms' JSON formats
4. Finally, we needed to integrate all this code together and update the UI of the frontend for the application to function properly

Each of these steps had more micro steps than needed to be completed to finish the step, but these are the main development points we faced when developing this application. We shall be covering these steps in detail momentarily.

For step one, the main thing that was holding us back from completing it was a lack of knowledge on how the Flutter framework functioned, as none of us had used the framework before. We soon after decided to separate our team into specific roles so that we were not all trying to do the same thing, which would end up wasting time if we had. After we all completed the tutorial on how Flutter worked, Joshua Poore took over the front-end development side of our project and was the person who handled the flutter side of our project the most. After he had gotten a better grasp on the fundamentals of Flutter, he created a simple app that had all the functionality that we would need till we started implementing the transfer process. In appendix 2, "Original UI Design", are some screenshots of what the application used to look like. After this functional mockup was completed, we moved on to the next step.

For step two, it was both Joshua Poore and Nikita Smith who handled JavaScript authentication for our two music platforms, Josh on Spotify Music, and Nikita on Apple Music. This step also came with a steep learning curve as neither of us had used web APIs before, let alone the specific ones that we used for this project. This process was started during spring break and was held back because of that. Both authentications had their issues, Spotify has a variety of authorization flows to choose from, and for a while, we were unsure as to which flow we would want to go with but decided on the authorization flow. Apple Music's authorization flow was a bit more abstract comparatively as we had issues with getting an Apple Music developer key, as to get one you need to be a part of a 100-dollar subscription service. In the end, we circumvented this issue as one of our team member's employers were kind enough to allow us to use their developer key.

For step three, we decided it would be easiest to manipulate a pull the data that we needed from Python, so whilst Nikita and Josh were working on user authentication, Ben Pallotti was tasked with creating a Python program that would pull the data that we wanted to manipulate from the Spotify API. Then once we had gotten done with the development of the JavaScript authorization code, Nikita handled the development of the Python code for Apple Music and also handled converting the two JSON files that we got into a singular JSON file type that we could convert into either one of the JSON types to enable a transfer of the playlist data from one music platform to another.

For step four, was more complicated than we had thought it would be. To preface, up until this point we had been testing our Flutter application in Chrome, as it was easier to set up and run. We soon ran into issues when trying to integrate our JavaScript code with our Flutter application, which is why we felt the need to pivot.

4.2.1. The Pivot

There were also a few major changes to our scope because of this challenge that we faced. Originally, we had planned to get our application running with three music platforms available to users, specifically Apple Music, Spotify, and YouTube Music. Due to the amount of time, we took trying to get over the previously mentioned challenge, we decided for this project it would be best that we move implementing all three to phase two of the projects. Currently we do have two music platforms implemented for this project, Apple Music and Spotify and the transfer action for both is functional. As stated, before this section, the main reason for the pivot was due to integration issues. However, there were also other concerns that lead to the pivot happening which will be listed below.

- Concerns about Apple and Google App Store Policies
- Most of our code would work without issue if we changed to a web application, the JavaScript code
- Increased difficulty integrating user authentication
- Wider Accessibility
- Flutter Limitations
- Internal desire for a faster development timeline

Each of these points played a part in our final decision to pivot, so each one will be discussed in a little more detail here. Our concerns over the Apple and Google App store policies were due to these devices allowing a developer to use the local port of that device by an application. Android would have no issue with that whilst Apple has a strict policy against letting developers use the local ports on a device, so it was causing there to be a rift in the development of our application. The second point is referring to the fact that we knew our JavaScript code worked fine, but because of the new environment we were trying to force it into, it was having issues communicating with Flutter. The third point is one mentioned throughout this paper as the main reason we decided to pivot in the first place and is related to many other points in this list. The points it is related to are points two, five, and six which will be explained once they are reached. For point four, we had already been considering pivoting for some time, and we realized that this would allow a user for greater accessibility of our product, as it could be accessed from mobile or desktop devices, not just mobile. Our fifth point is a result of realizing that many of Flutter's open-source packages were not suitable for us to use in terms of pulling data from APIs. Most of them are also not well documented so it became increasingly difficult to find packages in Flutter that could handle user authentication, which is why we decided to handle that in JavaScript instead. The final point is mainly because our team realized this was the final hurdle in creating the application we had set out to create. Given more time, we are confident we could have found a way for the application to work as a mobile application, but we decided for the sake of the project and to serve as a proof of concept that we would create a web application instead.

4.2.2. Where are the External Interface Requirements

The short answer is that since so many of those requirements were changed due to our pivot from a mobile application to a web application, it did not feel needed to include them inside this final report. At first, we had specific interface requirements for mobile applications, and then we found there were compatibility issues with Dart and our chosen data types. This led us to change to a platform that we are more familiar with and allowed us to overcome our issue.

4.3. Finalizing Phase

After we had decided to pivot, we just needed to integrate current JavaScript code with our Flutter web application. This task was handled by Nikita Smith, as his extensive knowledge in handling JavaScript was invaluable during this period. Meanwhile, Joshua Poore worked on updating the team's application UI to work for the entire functionality of the application and to make it look better. Screenshots of the final UI can be found in appendix 2 under "Current UI Design".

5. TESTING

This section will cover how we test our application to make sure we have met our requirements.

5.1. How we test our application

The way we have been testing our application is to adequately check to see if all functional parts of our project work as intended, and then check to see what requirements were met and what were not. Below is an example test report based on the current state of the application. There may be some redundancy in the checks as some general functional checks may overlap with requirements. If you wish to see the requirements used to evaluate them, they are mentioned in section two of this report.

5.2. Example Test Report

General Functional Checks	**True**	**False**
Does the application authenticate Spotify accounts correctly?	X	
Does the application authenticate Apple Music accounts correctly?	X	
Does the application allow the user to choose what platform they are transferring to and from in an easy-to-understand fashion?	X	
Does the application allow the user to select what playlist they want to transfer over?	X	
Does the application handle any potential errors in the transfer process? (i.e., missing songs and things of the sort)	X	
Does it notify the user if an error has occurred?		X
Does the application notify the user when the transfer process is completed?		X
Can a user transfer from Spotify to Apple Music?	X	
Can a user transfer from Apple Music to Spotify?	X	

Requirements Checks	**Met**	**Not**
Functional Requirement 1: Example	X	
Functional Requirement 2: Authentication	X	
Functional Requirement 3: Display Home Page	X	
Functional Requirement 4: Navigate Screens	X	
Functional Requirement 5: API and Conversion	X	
Functional Requirement 6: Constraints	X	
Non-Functional Requirement 1: Security	X	
Non-Functional Requirement 2: Capacity		X
Non-Functional Requirement 3: Login and Password		X
Non-Functional Requirement 4: Usability	X	

CONCLUSION/SUMMARY

At the current state of the project, it stands as a definitive proof of concept for our idea of this application. There are several features and platforms we want to implement into the application in the future, such as YouTube Music as a platform that users can transfer from and to. Another feature we have discussed is allowing users to transfer their whole library at once, but that created new conflicts for a feature that is not as important as its main functionality. Through strategic pivots, problem-solving, and perseverance, we have developed a functional and useful application. We look forward to taking the lessons learned from this experience and applying them to future projects and improvements to this project.

Appendix 1: References

Project Plan

Indy02-PlaylistSyncer-GanttChart.xlsx (sharepoint.com)

API Links

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

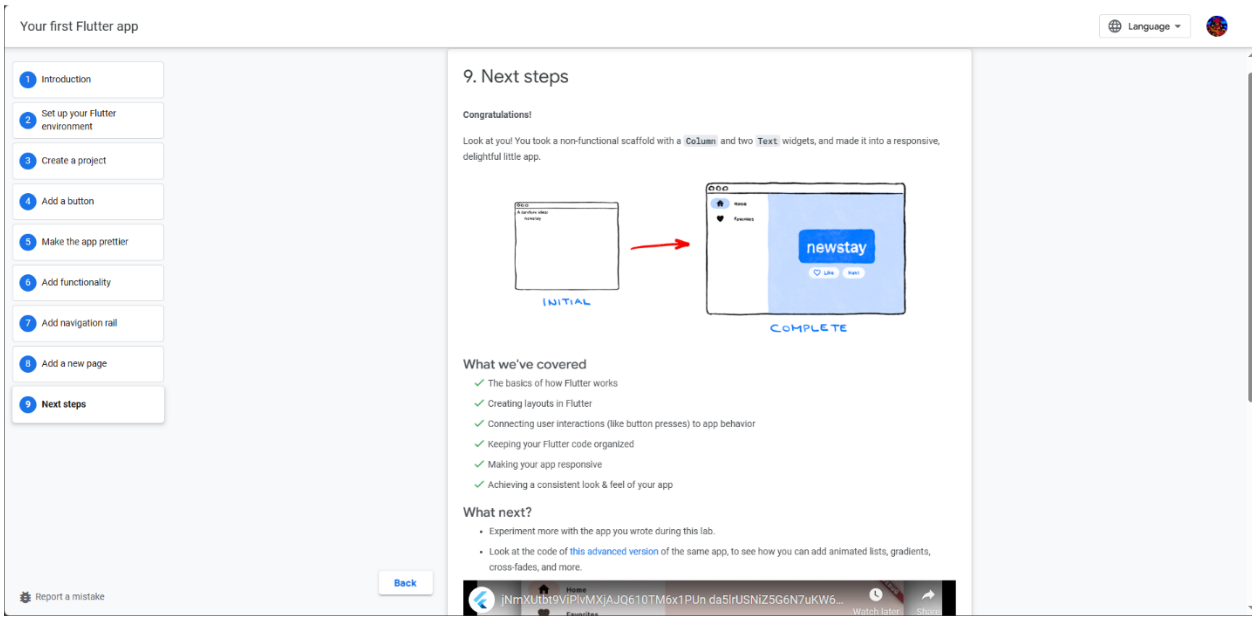
Spotify Music: https://pub.dev/packages/spotify_sdk

Development Documentation Links

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

Appendix 2: Training Verification and Images

Training Verification



Your first Flutter app

Language

1 Introduction
2 Set up your Flutter environment
3 Create a project
4 Add a button
5 Make the app prettier
6 Add functionality
7 Add navigation rail
8 Add a new page
9 **Next steps**

Report a mistake

Back

9. Next steps

Congratulations!

Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.

What we've covered

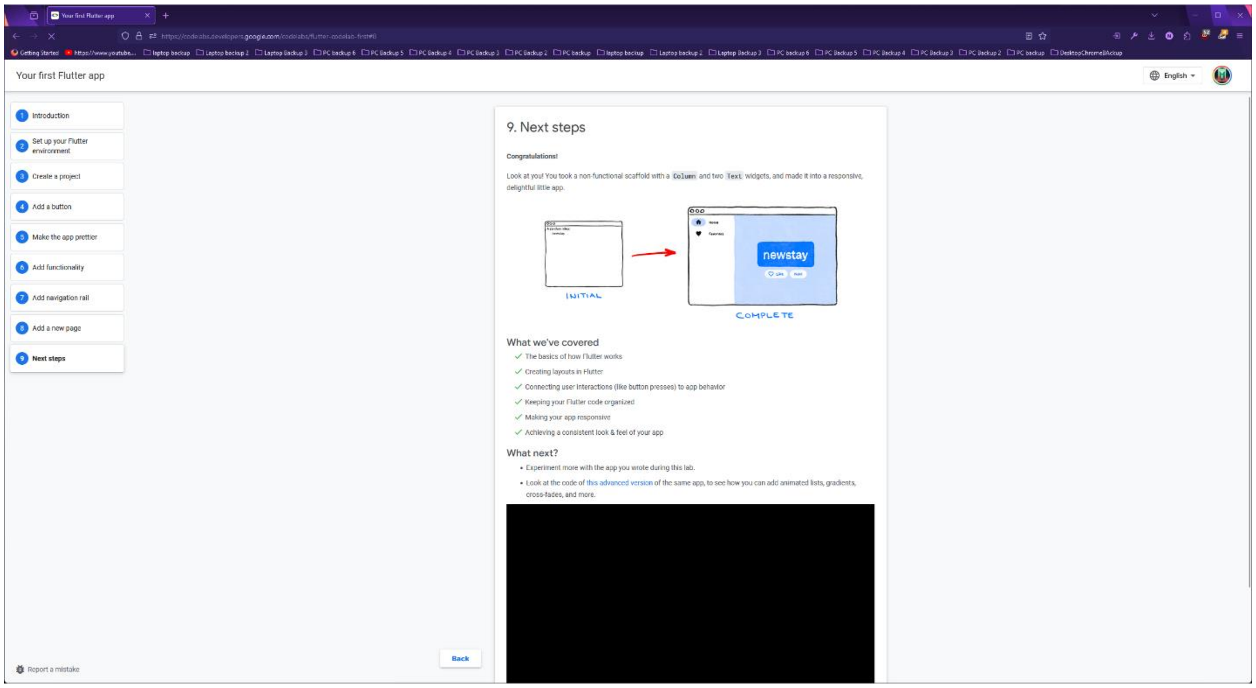
- ✓ The basics of how Flutter works
- ✓ Creating layouts in Flutter
- ✓ Connecting user interactions (like button presses) to app behavior
- ✓ Keeping your Flutter code organized
- ✓ Making your app responsive
- ✓ Achieving a consistent look & feel of your app

What next?

- Experiment more with the app you wrote during this lab.
- Look at the code of this advanced version of the same app, to see how you can add animated lists, gradients, cross-fades, and more.

jNmXUtbt9ViPlvMXjAJQ610TM6x1PUn da5lrUSNiZ5G6N7uKW6...

Joshua Poore's Proof of Tutorial Completion



Your first Flutter app

English

1 Introduction
2 Set up your Flutter environment
3 Create a project
4 Add a button
5 Make the app prettier
6 Add functionality
7 Add navigation rail
8 Add a new page
9 **Next steps**

Report a mistake

Back

9. Next steps

Congratulations!

Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.

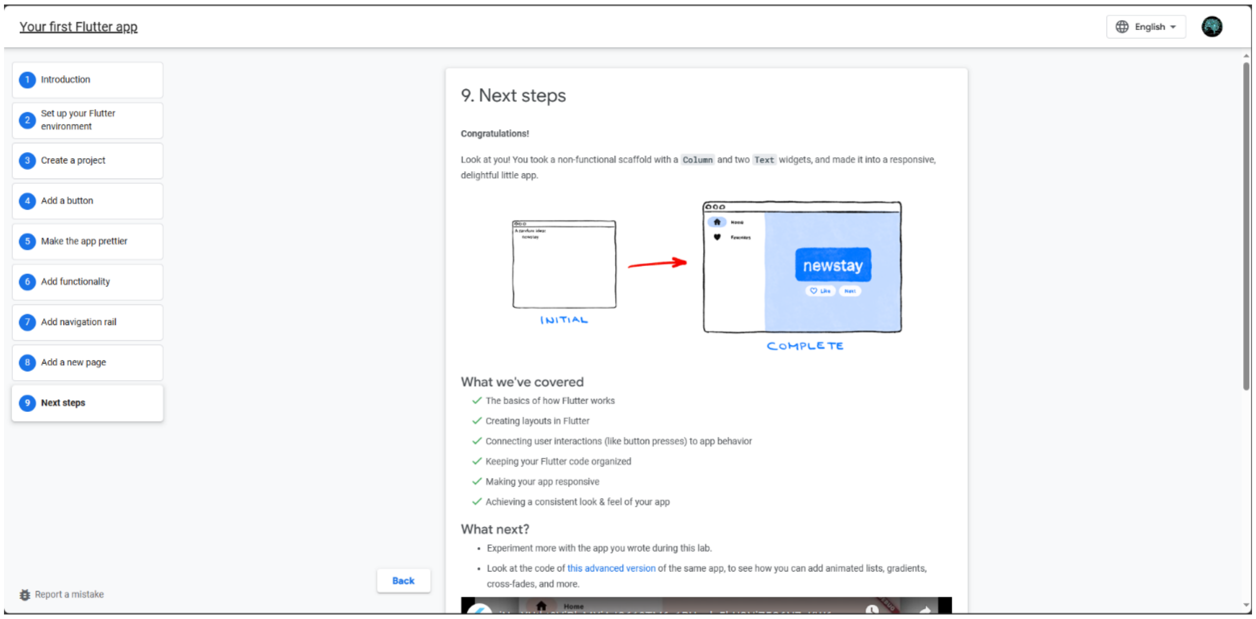
What we've covered

- ✓ The basics of how Flutter works
- ✓ Creating layouts in Flutter
- ✓ Connecting user interactions (like button presses) to app behavior
- ✓ Keeping your Flutter code organized
- ✓ Making your app responsive
- ✓ Achieving a consistent look & feel of your app

What next?

- Experiment more with the app you wrote during this lab.
- Look at the code of this advanced version of the same app, to see how you can add animated lists, gradients, cross-fades, and more.

Nikita Smith's Proof of Tutorial Completion



Your first Flutter app

English

1 Introduction
2 Set up your Flutter environment
3 Create a project
4 Add a button
5 Make the app prettier
6 Add functionality
7 Add navigation rail
8 Add a new page
9 Next steps

Report a mistake

Back

9. Next steps

Congratulations!

Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.

What we've covered

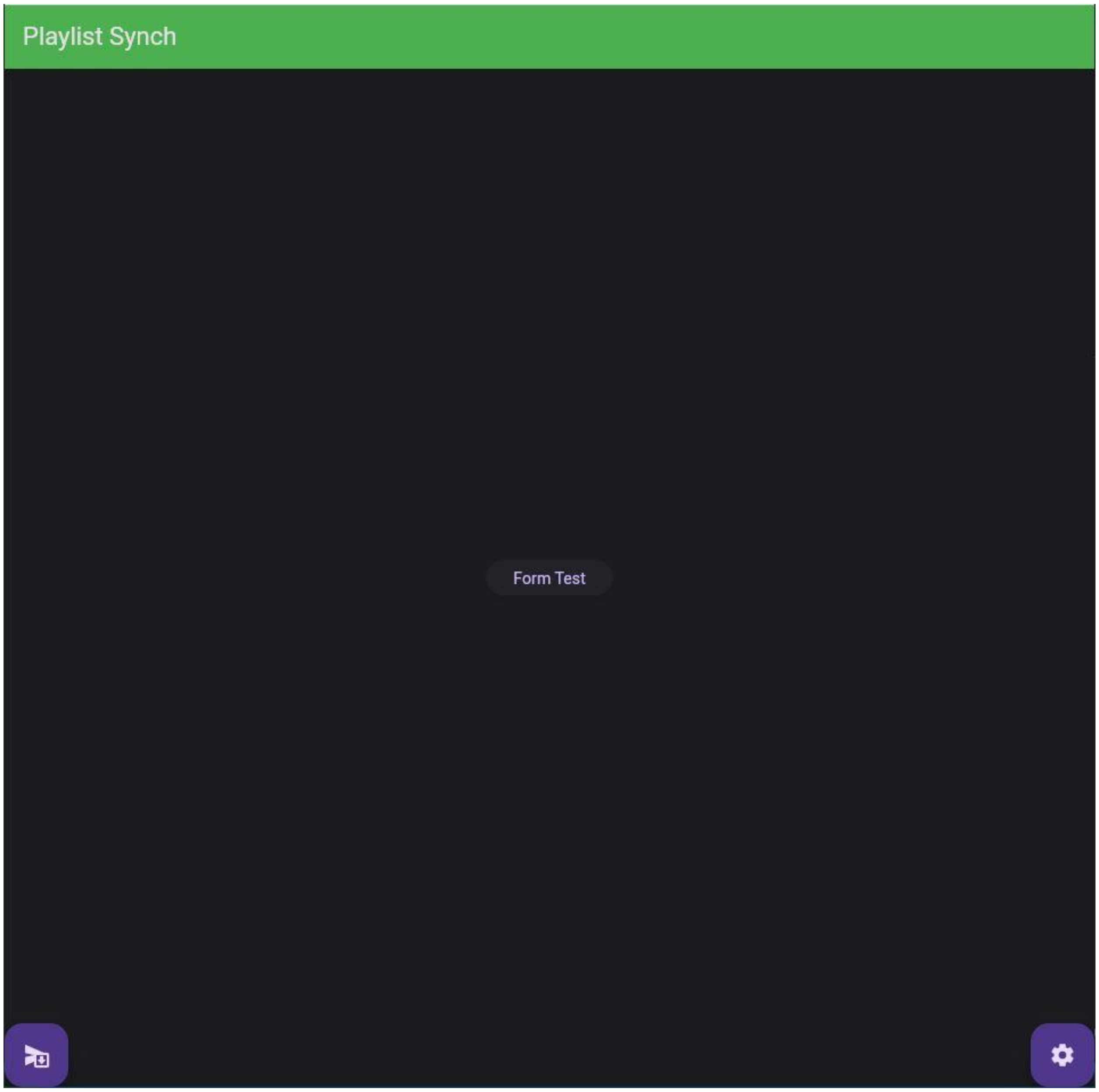
✓ The basics of how Flutter works
✓ Creating layouts in Flutter
✓ Connecting user interactions (like button presses) to app behavior
✓ Keeping your Flutter code organized
✓ Making your app responsive
✓ Achieving a consistent look & feel of your app

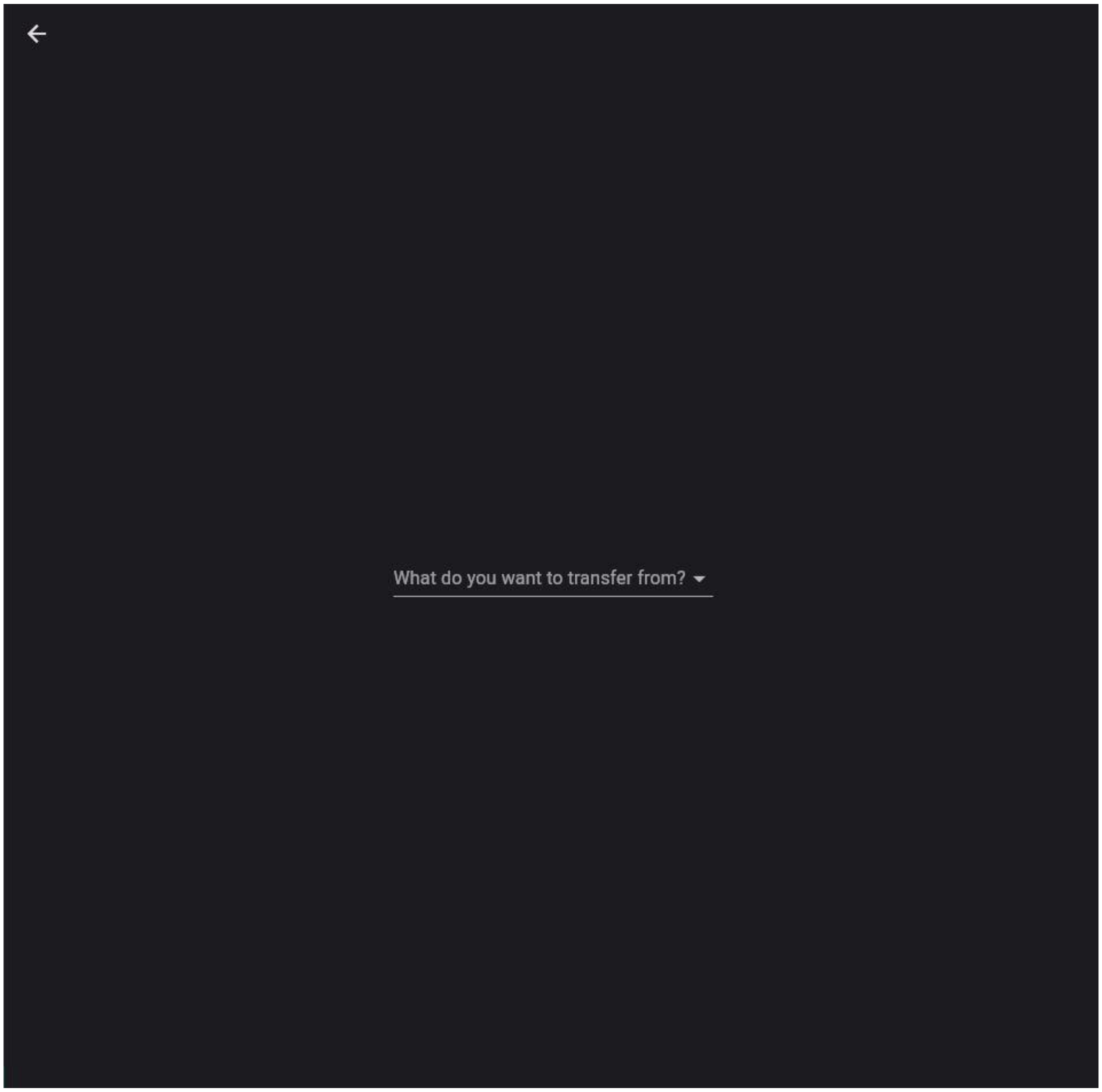
What next?

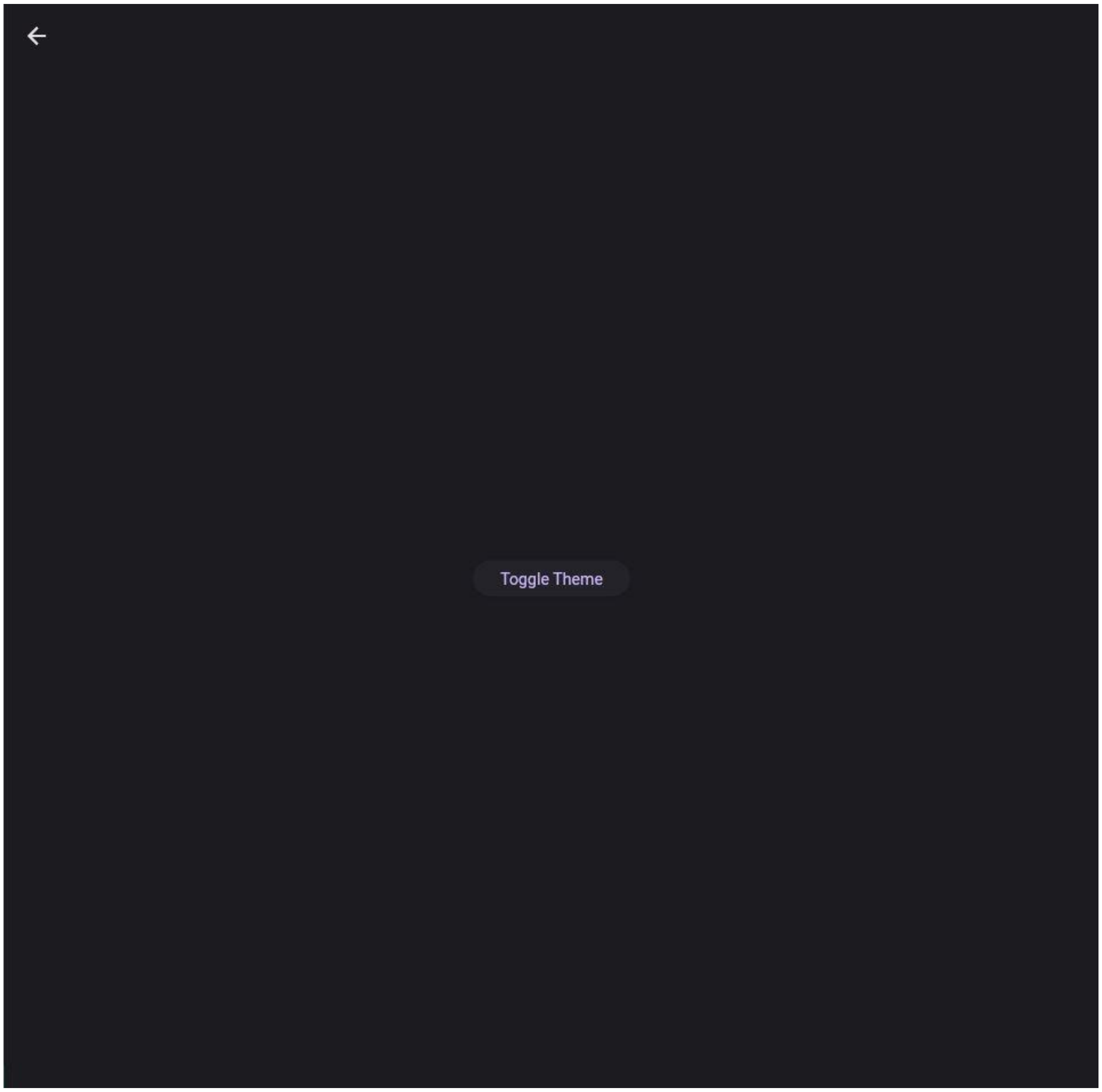
- Experiment more with the app you wrote during this lab.
- Look at the code of this advanced version of the same app, to see how you can add animated lists, gradients, cross-fades, and more.

Ben Pallotti’s Proof of Tutorial Completion

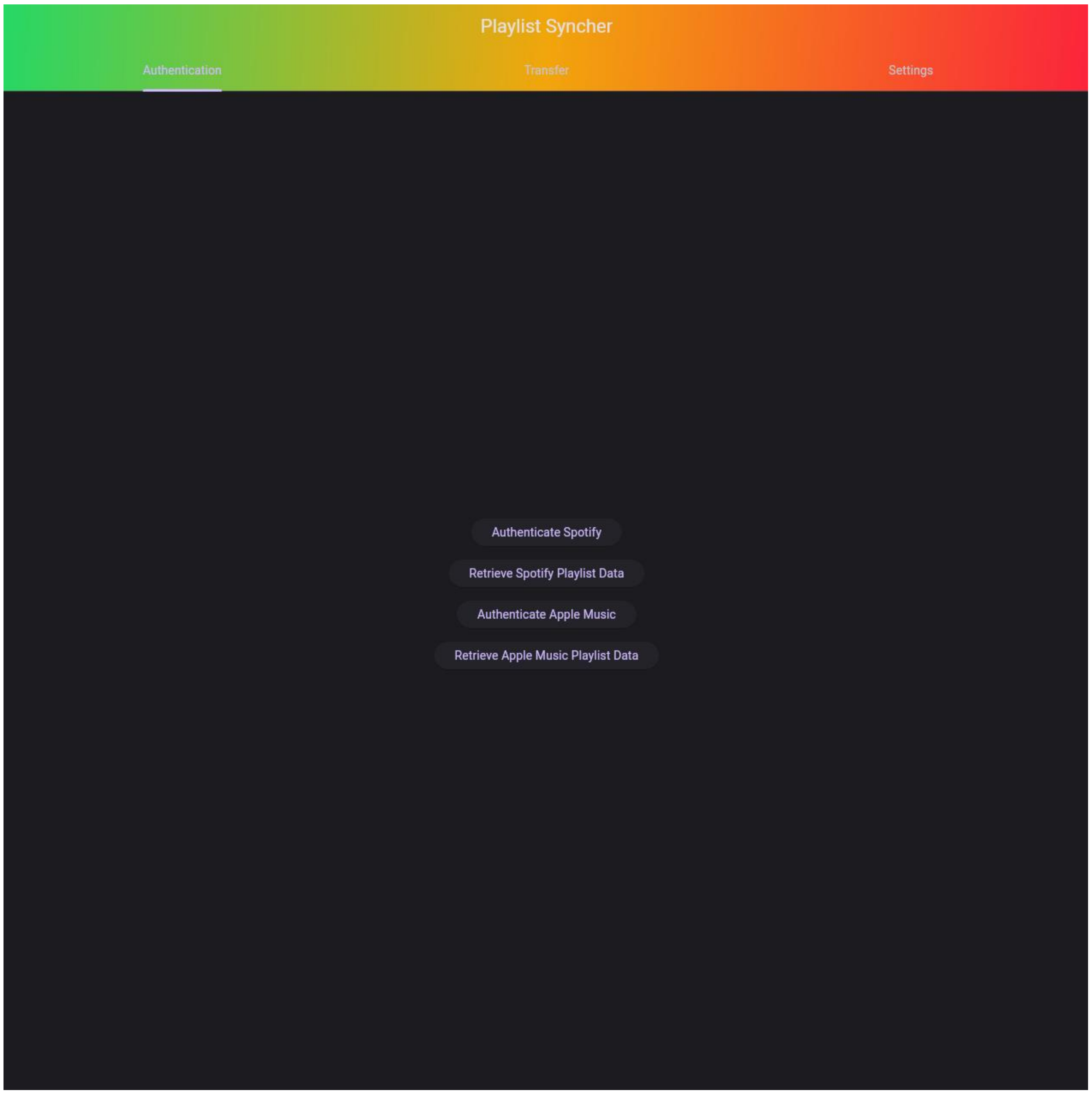
Original UI Designs

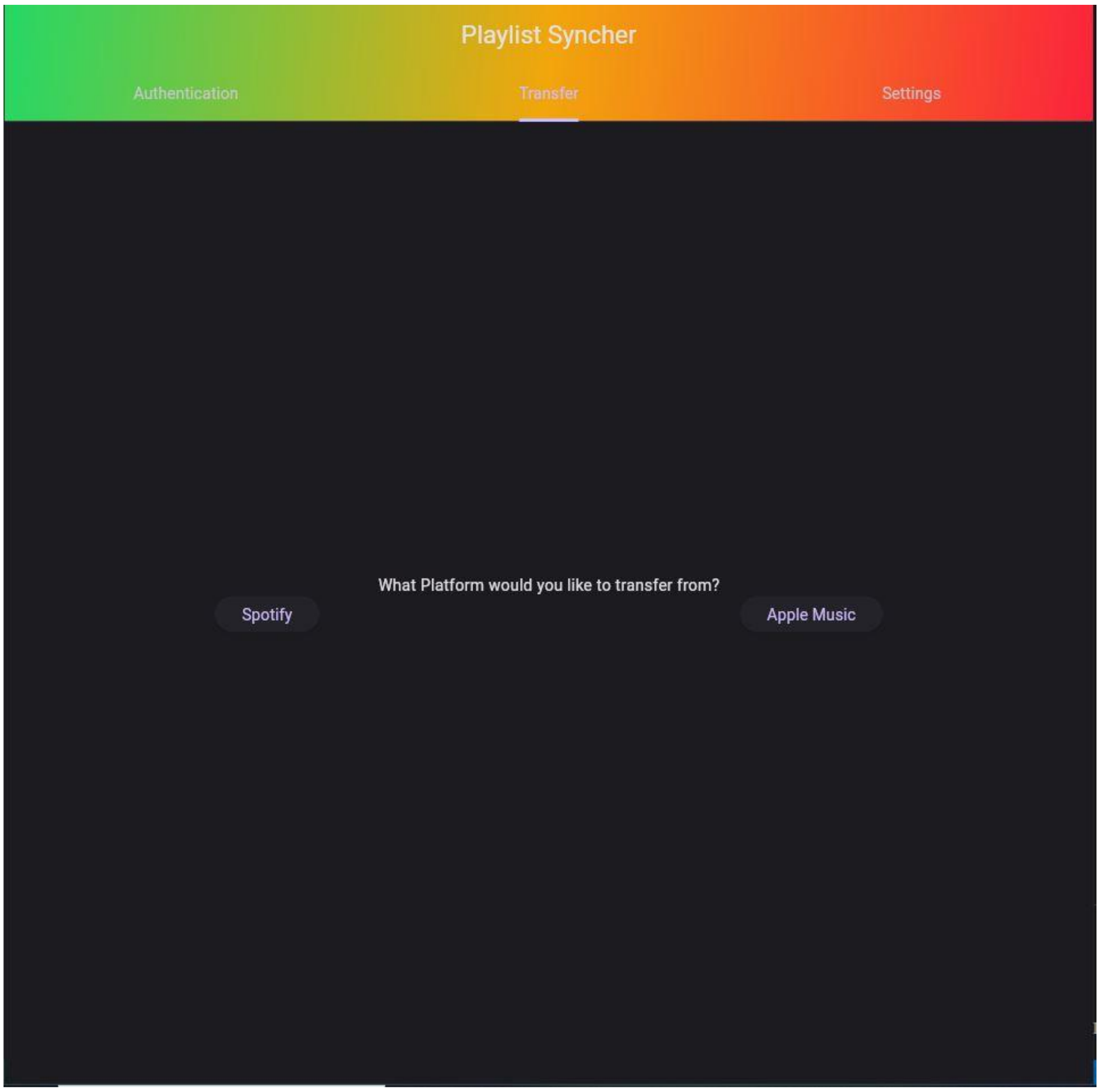


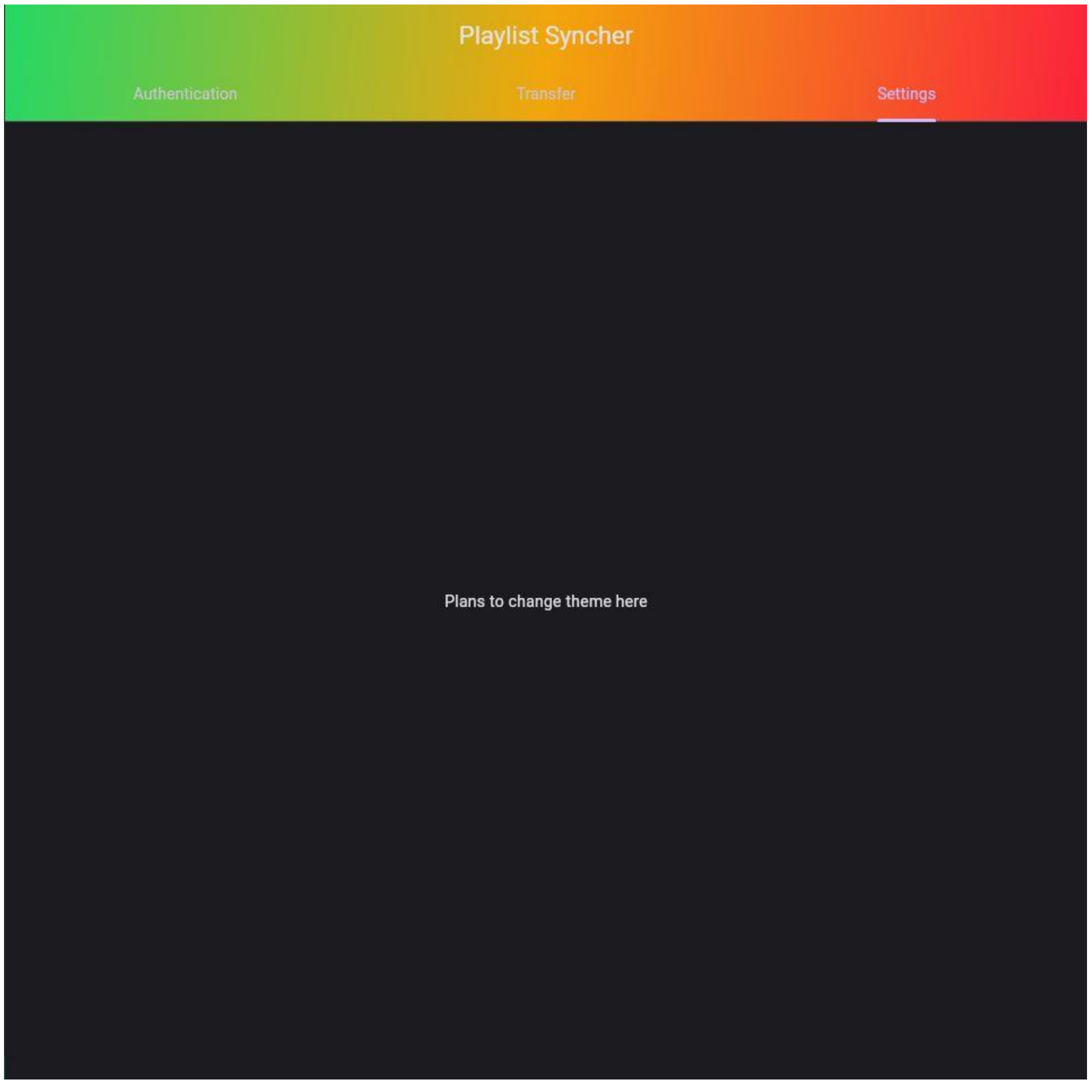

What do you want to transfer from?


Toggle Theme

Current UI Design




Playlist Syncher
Authentication
Transfer
Settings
What Platform would you like to transfer from?
Spotify
Apple Music



Plans to change theme here