CS 4850 Section 02, Fall Semester 2023

SP-29 Green Spotify App

Sam Chandler, Jeremy Ray, Lezly Serrano

Sharon Perry

11/26/2023

Website: https://sp-29-green.github.io

GitHub: https://github.com/Lezlyes/Spotify-Recommendations-App

Project Overview

Abstract

Discover new songs and artists tailored to your interests through an engaging visual interface. Our team's app takes users' Spotify data to provide precise recommendations based on their listening habits. This mobile app utilizes the freely accessible web API for Spotify, the most popular music streaming service, to offer a novel way to receive music recommendations. Additionally, our app provides users with robust functions for viewing and sorting their listening data.

The characteristic that sets this app apart from other recommendation apps is the availability of data. Many recommendation apps portray themselves as magical predicters, obscuring the actual data analysis. This app will give all of the information to the user. Users can see their listening habits, and recommendations will be backed up with numbers from their data. Too many apps hide all of the interesting details, which are often the only reason that users sign up in the first place. This app will satisfy that demand.

Platform

The app was built using Flutter to allow for synchronized iOS and Android development. Flutter is an open-source platform developed by Google that allows apps to be built for both iOS and Android devices. Flutter uses the programming language Dart, which our team became familiar with during the development of this project.

Project Planning

The planning phase of this project involved a project plan document and requirements and design documentation. Such as a System Requirement Specification (SRS) document. An initial System Design Specification (SRS) document. In this phase we first created schedules and a project scope to help us execute tasks.

Scope

*The scope of the project served as a roadmap that outlined the boundaries, objectives, and functionalities of the application.

1. **User Account Access**
    a. Create New User Account
        i. Create username and password
        ii. Import User listening data
        iii. Allow autofill and database searches for user accounts
    b. Modify User Account
        i. Delete account
        ii. View previously existing accounts
2. **User Information Storage**
    a. Collect User Information
    b. Allow modification of user information
    c. Allow removal of user information
3. **GUI Display**
    a. Universal Elements
        i. Account creation screen
        ii. Account login screen
    b. Main Menu
        i. Display tailored user playlists with new song recommendations
        ii. Display new suggested artist list recommendations

GANTT Chart

*The GANTT Chart helped us determine a schedule for each task and outlined the length of our project development.

Spotify Reccomendations App
9/7/2023

					Milestone #1				*Milestone #2*				*Milestone #3*				C-Day	
Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	**09/10**	**09/17**	**09/24**	**10/01**	**10/08**	**10/15**	**10/22**	**10/29**	**11/05**	**11/12**	**11/19**	**11/26**	**11/30**	**12/04**
Requirements	Project Plan	100%	Completed	Everyone	10													
	Weekly Activity Reports	5%		Team Leader	1	1	1	1	1	1	1	1	1	1	1	1		
	Software Requiremnets Specifications (SRS)	15%		Document Writer		10	10											
	Software Design Document (SDD)	0%		Document Writer		10	15											
	Software Test Plan (STP)	0%		Document Writer		10	5											
Project design	Define Tech Requirements	0%		Everyone		10	10											
	Database Design	0%		Everyone		20	15	15										
	App Design	0%		Everyone			20	20	20									
	Website Design	0%		Everyone				10	10	5								
	Develop Working Prototype	0%		Everyone				20	20	15	10							
	Test prototype	0%		Everyone						10	5							
Development	Research Results	0%		Everyone		10	10	5	5	5	5							
	Code Review	0%		Everyone				15	15	15	15	15	15	15	20			
	Final Prototype Testing	0%		Everyone								20	20	20	20	15		
	Design Testing (UX/UI)	0%		Everyone									15	15	15	15	5	
	Website Design Testing	0%		Everyone										10	10	5		
Final report	Review Final Draft Report	0%		Document Writer												15	10	
	Final Project Package Preparations	0%		Everyone												15	10	
	CS Dept. Evaluation Perparations	0%		Everyone												15	15	
	Project Owner Evaluations	0%		Everyone													5	5
			Total work hours	**757**	**11**	**71**	**86**	**86**	**71**	**51**	**36**	**36**	**51**	**61**	**66**	**81**	**45**	**5**

Legend
Planned
Delayed
Number Work: man hours

System Requirement Documentation

1. Introduction

1.1 Overview

Discover new songs and artists tailored to your interests through an engaging visual interface. Our team's app will take users' Spotify data to provide precise recommendations based on their listening habits.

1.2 Project Goals (Scope)

The main goal of the app is to enhance Spotify user's experience with the added functionality that the app provides such as in-depth analysis of their listening habits and recommendations based on that listening data.

1.3 Assumptions

- The availability and stability of the Spotify API
- Internet connectivity for real-time recommendations

2. General Design Constraints

2.1 Product Environment

The Spotify Recommendations app will be a mobile application built using Flutter, an app building software. This app will be compatible with iOS and Android devices. It will utilize the Spotify API to access user playlists and listening history.

2.2 User Characteristics

The primary user classes include:

- Registered Spotify users.
- Administrators for app management

2.3 Mandated Constraints

The app will be built for iOS and Android with the Flutter framework and Spotify API. Constraints brought about due to these technologies apply to this app as well.

3. Nonfunctional Requirements

Nonfunctional requirements are conditions of the app that must be met, such as performance requirements or legal requirements. The functional requirements (the specific characteristics of the application, like a home page) will be discussed later in Section 4.

3.1 Performance

Since the app uses data gathered from users' Spotify accounts, as well as databases relating to musical artists and genres, the app is handling a lot of data at any given time. This can lead to a decrease in performance if handled incorrectly. Users do not want to wait for information to be gathered every few seconds, so data collection will need to be started as soon as possible to ensure consistent performance.

- The app must respond within 2 seconds for most user interactions.
- Recommendations should be generated in real-time or near real-time.

3.2 Security

- User authentication tokens must be securely stored and transmitted.
- User data and preferences must be kept confidential.
- The security of the app must also be adequately known to the users, as they may be hesitant to sign up with their Spotify account if there was any suspicion of danger.

3.3 User Interface

- The app must have an interactive and visually appealing user interface.
- Must support multiple models of iOS and Android devices.

3.4 Compatibility

- The app should work smoothly on a range of devices and screen sizes
- The app should adapt to device settings such as light and dark themes.
- The app's functions must also be compatible with the available tools given by the Spotify Web API.

3.5 Legal Issues

- Since this project utilizes Spotify's API, the team must be careful to conform to Spotify's terms and conditions regarding the API use.
- Non-streaming SDAs are the only services that are allowed to use limited monetization.

4 Functional Requirements

4.1.0 Launch Page

- Upon opening the launch page will display a login and new user button.

4.1.1 Login button redirects to login page

4.1.2 Register button redirects to the Spotify app to create an account

4.2.0 Login Page

- The user enters login credentials, and the system verifies their credentials before redirecting the user to the main page.

4.2.1 Login with username

4.2.1 Password

4.2.3 Recover password (optional)

- Displays a forgotten password button that allows the user to send a password recovery email.

4.3.0 Register - Create Account – Page

The create Spotify account button will redirect a user with no existing Spotify account to the Spotify website to create a listening account.

4.4.0 Homepage

- Displays a hub of curated playlists and new listening recommendations
- Displays button to access user profile

4.4.1 Recommendation Category

- The app generates personalized music recommendations.
- Recommendations based on users listening history

4.4.2 Browsing Recommendation

- Users can view through recommended songs and playlists
- Users can view recommended new and similar artists

4.4.3 Create Playlists

- Users can create playlists based on recommendations
- Users can add and remove songs from playlists

4.5.0 User Profile

4.5.1 Profile management

- Users can view and edit their profile information
- Users can link/unlink their Spotify account

4.6.0 Spotify Data Management

- The app can retrieve the user's listening history, playlists and liked songs from Spotify

System Design Documentation

1. Introduction

1.1 Document Outline

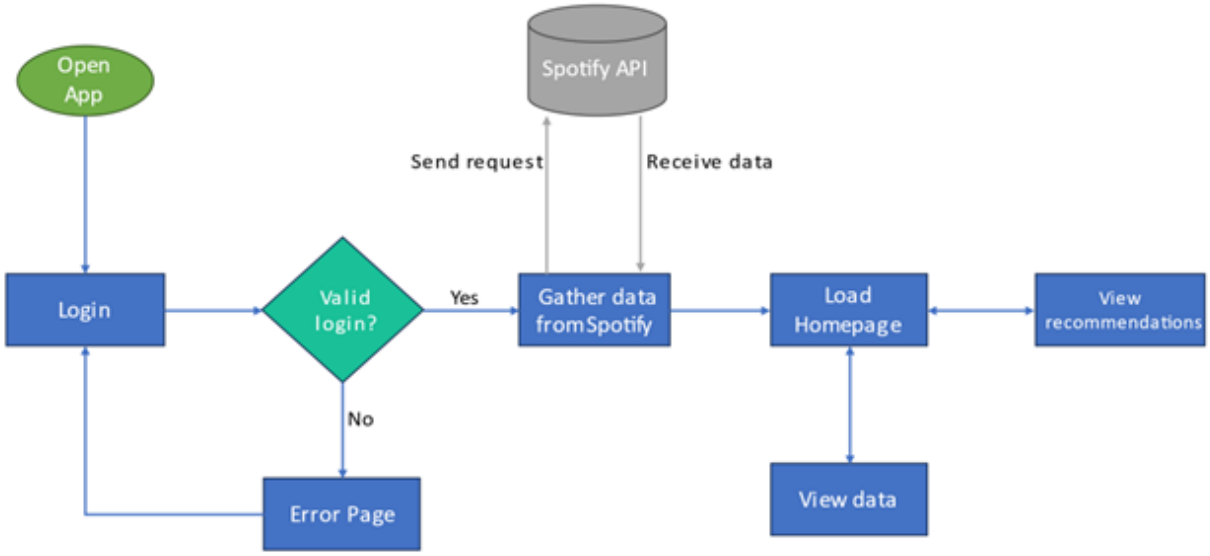
This document outlines the design of SP-29 Green's Spotify app. The designs shown in this paper are for a mobile app that will be compatible on Android and iOS devices. The software Flutter will be used to develop the app simultaneously for Android and iOS devices.

1.2 Document Description

This paper includes a process flow to illustrate the general flow of a user's experience with the app. This document also includes a set of mock-ups to visualize roughly how the app will look. The mockups lack a lot of aesthetic design choices that will be included in the final product, such as album artwork images.

2. Process Flow Diagram

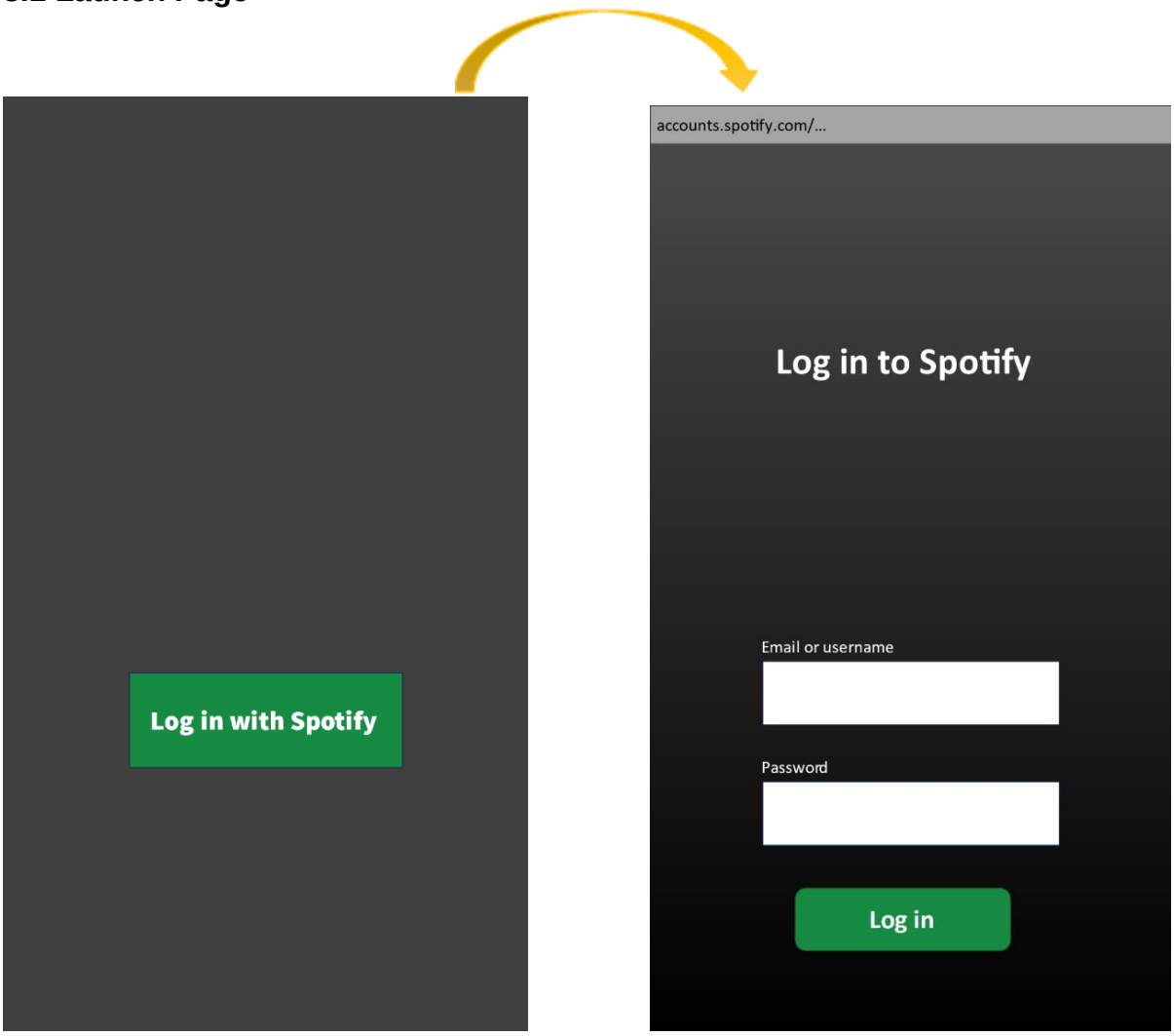
The following diagram shows the general process flow of the mobile app.



This diagram shows an abstracted view of how the app functions. The process begins when the user opens the app. The user proceeds to log into the app using their Spotify account. If the account is not recognized, the user will be shown an error page and then will be set back to the login screen. If the login is successful, then the app will begin gathering data on the user's account from Spotify. Once the information has been received, the homepage is loaded. From the homepage, the user has several options. The options have been simplified down into to categories for the process flow diagram: "View recommendations" and "View data." The normal homepage for the app is loaded again whenever the user leaves the recommendations or data pages, signified by the two-way arrows attaching "Load Homepage" to the two processes.

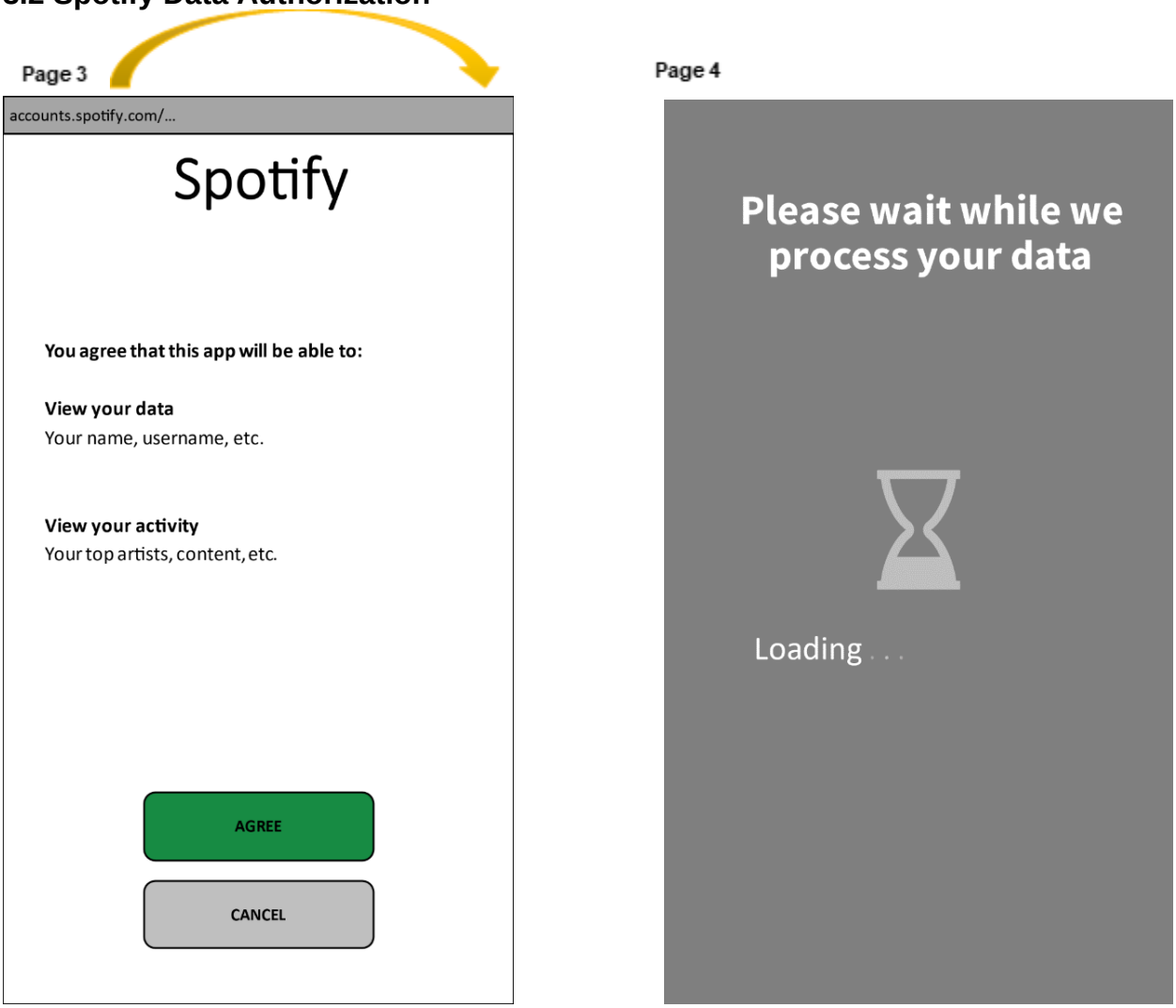
The following section provides a more in-depth view of the app's design with mockups.

3. Mockup Diagrams
3.1 Launch Page



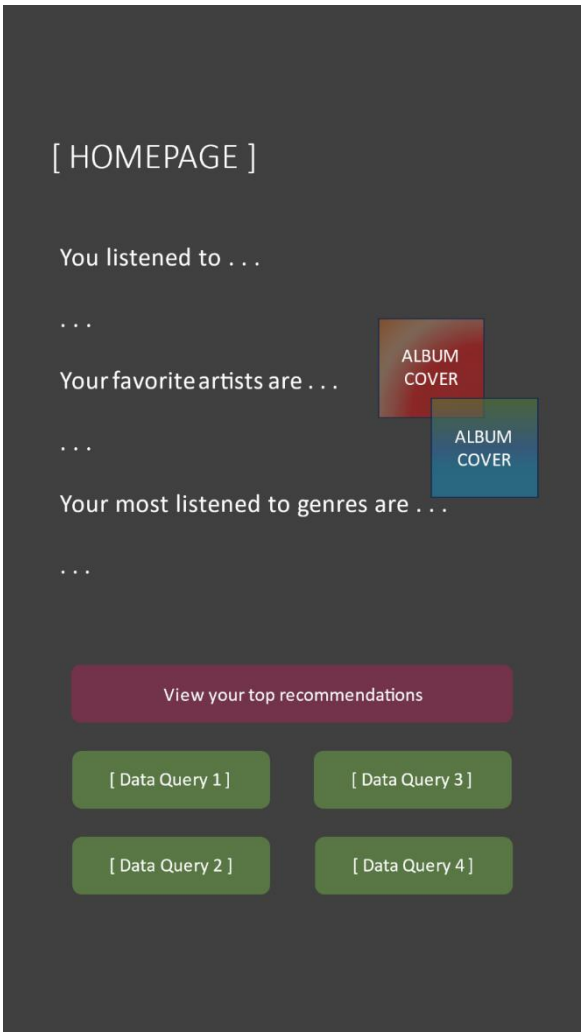
The Launch Page (Page 1) prompts the user to Log in using Spotify. The application uses Spotify's Web API, so logging in with their existing Spotify account is absolutely necessary to begin using the app.
Once the user presses the Log in button, they will be prompted with Spotify's log in page (recreated in Page 2) and log in with their existing email/username and password for their Spotify account.

3.2 Spotify Data Authorization



Once the user has logged in, they will be prompted by Spotify to agree to or decline allowing the app to use their data from Spotify (Page 3).
If the user selects CANCEL, then they will be booted back to the launch page to begin the process over. If the user selects AGREE, then the application will continue with its process.
After pressing agree, there will definitely need to be a loading screen while data is being gathered and processed. Before the user can see the apps.

3.3 Homepage



Once the data has been fully gathered from Spotify and other necessary sources, the user may view the home page.
On the home page, the user is shown a short summary of their listening data. Below the summarized data, the user has many options for what to view, such as their top recommendations and several data queries that are more in depth than the summarized data shown above. Data queries and recommendations will take place on the homepage.

Version Control

For effective collaboration and version control, our team used Git, which is a widely adopted distributed version control system. We established our repository on GitHub, a web-based platform that integrates with Git, providing a collaborative environment for our project. Git allowed us to track changes, manage different versions, and work concurrently on various features. Git has many useful features, such as branches and tools for issue tracking and code review.

GitHub, which hosted our repository, has a very user-friendly interface which made it easy for team members to navigate the project, track changes, and contribute effectively. The project is hosted on a main branch which we can push once we are satisfied with our changes. GitHub had additional security measures and was able to detect when we accidentally included our Client Secret (an important Spotify Web API key) on our public GitHub page. Since we were notified by a GitHub bot, we were able to change the Client Secret and fix the security leak issue.

Narrative Discussion

Our app was built using Flutter and Dart, as well as using the Spotify Web API. Each of these factors added additional limitations and restrictions on what we were able to do and how we were able to do it.

Flutter, like any framework, has limitations on what it is able to accomplish. Not every wish may be feasible to implement with Flutter. Additionally, not every feature implemented with Flutter is available to all devices. There are certain widgets and features that are exclusive to iOS or Android devices. We needed to make sure that our app ran on Android and iOS devices, so we had to think carefully about how we would approach certain problems. We encountered an issue like this that led to a tough decision during programming. During the development of the Log In page, we used a WebView. Unfortunately, WebViews are only compatible with mobile devices (Android and iOS), which meant that we had to lose support for web apps and Chrome apps, which are apps that Flutter supports concurrent development for. Luckily, we were never planning on developing for these, so it was an easier decision to make. You're also limited by the devices that you're using. We developed for iOS and Android, and so we needed to make sure that the features that we were adding were compatible with both of those devices. There are certain things that may only work for Android and certain things that may only work for iOS, so there are additional restrictions for that. And then Dart, like any language, has restrictions as well. Any programming language has limitations based on what the language is capable of. So, with Flutter, we had to add many packages. We had to add the WebView package along with the crypto package and the typed data package. All of these had to be added and used because you cannot rely on the default Flutter to give you all the tools that you need.

In terms of the Spotify Web API, you're limited by that as well because you need to make sure that the app works according to the Spotify Web API. You cannot really do whatever you want; you are limited by what Spotify allows you to do. For example, you can't make a trivia app out of the Spotify data because that goes against Spotify's terms of service. In addition to that, there are many features that we came into because we were never going to make a trivia app that one did not really apply to us. But there were a lot of things that we realized during development that we were limited by the Spotify Web API. For example, you can't show a user the number of times that a user has listened to a specific song. Unfortunately, I assume that that's because Spotify wants to keep that information for themselves; they want to keep that data for their Spotify Wrapped. Spotify Wrapped is an event that Spotify does every year, and so they want to keep a lot of the features and have exclusivity for that. A lot of stuff that we wanted to do was not allowed or possible through the API, so again, you can't explicitly show how many times a user is listening to a song. All you know is which songs they've listened to the most and in what order they've listened to them, what their most listened-to song is over one month, over six months, over all time. You're also limited by the time ranges; one month, six months, and all time are the only ranges that you have for this data. You're not given access to the entirety of Spotify data, which I didn't know when going into this project.

All of this puts an important focus on the analyzing feasibility phase of our software test plan, of our software design life cycle. That's because you can know all of these things if you are actively looking through the Spotify documentation. You will know what you have access to and what you don't have access to. So, you don't get to that difficult position where you're already in the implementation phase and you're designing or coding whatever you want the API to do, and then you realize that it can't do that. It's really important to look at all of the requirements and all of the limitations for Flutter, for Dart, for iOS, for Android, and most importantly, for Spotify Web API because that's the one that seems like you are limited with. Flutter has a lot of flexibility, especially in external packages that you can use. I never really felt too limited by that or that I couldn't do anything. It really feels like for our team that anything that we wanted to do, we weren't really being held back by Flutter. But Flutter did impact how we were able to implement these things, and it's very different from other frameworks and other languages, so you have to get used to that.

This also puts a large emphasis on proper division of work, and I think if we had continued to work on this app and if this got to the point of actual uploading to app stores and such for Apple and Android or Google Play stores, if this got to that point, then we would really need to have a team that had a clear division of work because having such the developers are also having to learn all of the documentation on Spotify and learn how Spotify works is really not efficient. It leads to issues where, you know, a developer wants to implement something and so they go ahead and start designing code, but they don't have all of the access. They're not completely knowing of the Spotify Web API, and so they'll encounter issues. It's very important to have a division of labor where you have someone or a group or a team who is dedicated to understanding the Spotify Web API and then someone who is dedicated to understanding Flutter and someone who's dedicated to understanding iOS so that you can have communication between, and you can have a team who really understands how Spotify Web API works. If the coders, if the programmers have any questions or if they have any ideas for how to implement, they can first go to that team who is knowledgeable on the API and ask them if what they want to do is feasible, if it is viable within the restrictions of the API. And I think that was a big part of this project was just sort of realizing how important the design of something is. It's not just about coding or implementation; there is a lot of logic behind and reasoning for a strict development cycle and a well-thought design plan.

Our app did end up changing a lot from what we had initially designed. The most basic functions are still there, and the entire app is implemented the way that it was intended, but there was a much heavier emphasis on data at the start. And then we had to change that when we realized that Spotify does not offer that much data to developers using the API. So certain functions had to be lost in terms of that. There were also visual things, limitations by Flutter that had to be changed. I initially wanted the homepage to have these low-opacity album covers that the user listens to display in the background, sort of animated and bounce around. But the way that Flutter works is it uses widgets, and so it views an entire page on someone's screen on the phone screen as a widget, or if you use it as a column of widgets. So, you know, a button is its own widget, a piece of text is its own widget, a column containing text would be its own

widget. So, the way it's set up is very structured. There is a clear gallery of widgets that is available when you're using the app. You don't see that; it's basically a grid of widgets of rows and columns and widgets stacked on top of each other. But when you're developing, you're very aware because you have to be aware of that since that's how the Flutter framework works. It's very interesting and a very good idea for making an app. There's really no reason to view an app as layers or stuff stacked on top of each other. It's very smart to view it as, "Okay, this button is its own section, this text is its own section, this image is its own section." But it made implementing the idea of having album art bounced around in the background very difficult because that's not really how Flutter works. You don't have widgets behind widgets. I believe that there's some way to implement it using a stack, but that just gets very complicated, and it's at that point; it's not worth the cute visual aspect for it.

One small and seemingly insignificant feature that we were able to add to the app was the customization of the theme. So there is a settings button at the top right that allows the user to change the theme. I believe some of the themes are red velvet, ice, and cherry blossom. So what this does is when the user changes the theme, if they go into the settings and they change the theme, it'll update the way that the app looks on all pages, on the home page, on the recommendations page, on the top items page; everything gets updated to the new theme. And it just changes the colors; it doesn't change much. It only changes the primary color, which is the background color, and the accent color, which is all the buttons and the headers and such. And so this is a seemingly small change; it just changes colors, it doesn't do much. But it was a very conscious decision because many apps that use the Spotify Web APIs are very gimmicky. They have one gimmick like visualizing your music in terms of circles or visualizing your music as fish in an ocean. And they don't do much beyond that. So what ends up happening is you use a Spotify Web API, a fun little app once, you know, it does something cool, and then you forget about it, and you don't use it again; you log out, you deactivate it, and then you just never use it again. And that's fine for a web application; if you have a website with many games and one of the cute things that you offer is a nice way for someone to log in with their Spotify and view this little program that shows them what their listening data looks like and everything, it's fine for that scale. But when you're making a mobile app, you really want a user to stay. You don't want a user to download the app, use it once, and then delete it. And so this gets into mobile app design philosophy, but one of the ways of accomplishing this is simply by having the user form a relationship with the app and feel like the app is theirs rather than some cool way to visualize their stuff, and then they don't want to use it anymore. So as soon as the user changes the theme to something that they maybe associate more with, like maybe they really like the color red, so they changed it to red velvet, now they have sort of a connection with the app that is drawing them back to it and making them want to use it more and reminding them that they have the app downloaded on their phone. Because if you got to the point where you are monetizing this app and we were adding ads, it does us no good if users are using them once because they're not being shown ads; you want users to get back on. Not necessary for an app like this, not necessarily daily, but you want them to get on every once in a while to see what their listening activities are like. And so if you personalize the environment for them and make it feel like it's theirs, their environment, they're likely to keep returning to this one

app rather than finding an alternative. So a simple settings icon does not do a great job of implementing this philosophy completely, but it is a nice way to, on a very small scale, show that we are thinking about it. And if this app was continuing development and if it got feature updates, that would be something that would be focused on. In addition to showing Spotify Web API stuff to the user, you would want the app to feel fun to use or nice to look at and customizable. So we would add more customization options and more ways that the user could identify with their app and remember to use it because ultimately, one of the biggest parts of an app is getting a user to stay on it and not delete it immediately. You have to be able to sell the app and make them think that it's worth downloading, but then you need to once they download it, keep them on. And so the customization was just a little way to show that.

The recommendations page is interesting because that was maybe something that we had the least knowledge about going into this. It's very easy to understand how you would display the top items to a user and the top tracks to a user, but figuring out how to do the recommendations was probably the most difficult aspect in interacting with the API other than the initial authorization setup and initially learning the API because recommendations have so many factors into it and it's so much more complex than just getting the user's data. The recommendation part of the app actually does not use the token at all when you make a recommendation, but we do have to use the token if we are using the person's favorite tracks. We would have to use the authorization token to get their favorite tracks, but then once we have them, when you make the recommendation API call, it doesn't have anything to do with the user; it's just all on Spotify, all on the general Spotify database. You give them what tracks you want, and then there is a lot of customizing in terms of what you want the recommended tracks to come back as. You can change the valence, which equates to basically how happy the track is. If it's a low valence score, then it's really sad; if it's a high valence score, then it's really happy. So, to the user when they're choosing their recommendations, what recommendations they wanna get, they just see the happiness slider and they get to change that around. There's also instrumentalness, which is just how instrumental a track is, if it has a lot of lyrics or if it's mostly instruments. Another feature that we added was how popular the track was; that's another thing that Spotify can use for recommendations. So, a low popularity score is between 0 and 100. A score of 0 would be a relatively unpopular song, at least by Spotify standards, and then a popularity score of 100 would be a very listened-to song that's very popular right now. And so there are many more features that we could use. You can change the tempo and the loudness of a track, but I figured when I was making, when I was getting these recommendations, that the user does not really care to choose the tempo. So, popularity, how happy it is, and how instrumental it is ended up being what I thought were the most important features to add. And you also want to keep it simple so that the user isn't overwhelmed with options, especially when it's things that they don't understand. That's part of why we changed the term from valence to happiness because the average user is probably not gonna understand what a valence score on a song is. But once the user makes these recommendations and they change the slider and customize it how they want, then they're given a list of songs and artists that are based on their recommendations and their topless and tracks and artists.

So in terms of how this was set up and how each person contributed, Sam Chandler is the team lead, and then Leslie Serrano is the lead document writer, and Jeremy Ray is the lead programmer. And these jobs often blended together as is natural for a team of this size. This was only a team of three; maybe if there were more people, if there were like six people, then we could have really clear lines. But when you're working with such a small group, there's bound to be overlap where you have someone who is supposed to write documents also testing the app because you need that. You need to have more testers; you need to have more programmers. And so, we ended up being very flexible; the job titles did not mean much, but we did try to stick to them.

In addition to making the app, we also made a GitHub Pages website. GitHub Pages is a very useful tool that allows you to create websites. It uses markdown language, and you basically create the website by just editing a readme. So, it's all done in the readme essentially, and if you want something to look like a header, then you use a single hashtag, and that would be heading one, and then two hashtags would be heading 2, and so on. You can customize the way that the webpage looks with that, and then you can use underscores for italics and such, and you can insert images. But it's all done within the readme of the GitHub organization account, which is very interesting, which is very cool. GitHub Pages also uses Jekyll themes, and a Jekyll theme is basically a default theme that you can use with already custom HTML and CSS styling with colors and such that you can just very easily, on the GitHub's organization webpage, you just specify which Jekyll theme you want to use. There are hundreds of themes available online, but GitHub offers maybe 20 default themes to choose from. We ended up going with a theme called Cayman, which looked pretty nice. And if you just make an app using GitHub Pages and using the markdown language on the readme and a Jekyll theme, you can very easily create an app and not require any HTML or any CSS knowledge at all. But since our app needed a little bit more, we needed to have links to our YouTube video and our project plan and such, and we wanted that to display in the header. We did actually have to go into the HTML of the Jekyll theme and change the HTML as well as the CSS files, but that's just because we really cared about the customization and the way that the website looked.

But GitHub Pages can be very useful for anyone, especially if you just use the very basic functions and if you don't have much of a vision for the app. But if you do, like we did, then you do have to have some sort of HTML and CSS knowledge. But it's still, even though we had to edit the HTML and CSS files, it ended up being way easier to just get kickstarted with GitHub Pages than it would be to build a website from the ground up. And that was my first-time using GitHub Pages, so it's actually very interesting and a very cool process. And I think from this project, I will probably use GitHub Pages in the future unless I really wanted to make my own website. But GitHub Pages also allows you to use your own custom URL, so even if the idea of using GitHub Pages for a professional website was a bit looked down upon, you don't have to actually use your GitHub.io URL for it. You can use a custom URL, and at that point, it's just a very useful tool for creating a website and hosting it on your own domain. We did not need to host it on a domain, so we did not care about that. But obviously, just in terms of future personal use, it was very useful.

Challenges, Assumptions, and Risk Assessments

Challenges

1. Algorithmic Complexity

1.1 Personalization Accuracy

Crafting an algorithm that accurately predicts user preferences amidst diverse music tastes, evolving trends, and individual listening behaviors.

1.2 Data Bias

Mitigating biases in the data that could lead to skewed recommendations or exclude certain music styles or artists.

1.3 Handling Data Variability

Dealing with the vastness and variability of music data (genres, moods, artists) to create comprehensive user profiles.

2. User Engagement and Satisfaction

2.1 User Feedback Incorporation

Balancing algorithmic predictions with user feedback to continuously improve recommendations.

2.2 Keeping Users Engaged

Sustaining user interest by consistently providing relevant and appealing recommendations over time.

3. Technical Challenges

3.1 Scalability

Ensuring the app remains responsive and efficient, especially as the user base grows.

3.2 Performance Optimization

Optimizing the app for various devices, ensuring smooth playback and responsiveness across different network conditions.

4. Privacy and Security

4.1 Data Privacy Compliance

Adhering to stringent data privacy regulations while collecting and processing user data for recommendations.

4.2 Security Measures

Implementing robust security measures to protect user data from breaches or unauthorized access.

5. User Interface and Experience

5.1 Intuitive UI/UX

Designing an intuitive and visually appealing interface that encourages user exploration and interaction.

5.2 Cross-Platform Consistency

Ensuring a consistent experience across iOS and Android platforms (mobile).

6. Legal and Licensing Issues

6.1 Music Licensing

Navigating complex licensing agreements and rights management for the music catalog.

6.2 Compliance

Ensuring compliance with copyright laws and licensing agreements in different regions.

Assumptions

1. User Preference Patterns

1.1 Pattern Consistency

Assuming users have consistent patterns in their music preferences, such as genre preferences, favorite artists, or mood-based listening habits.

2. Algorithmic Effectiveness

2.1 Algorithm Accuracy

Assuming the recommendation algorithm, based on user listening history and behaviors, can accurately predict and suggest music that aligns with user tastes.

3. User Engagement

3.1 Interest in Recommendations

Assuming users are willing to explore and engage with recommended music, trusting the app's ability to introduce them to new tracks or artists.

4. Data Availability and Quality

4.1 Sufficient Data

Assuming there's enough diverse and high-quality data available to train the recommendation algorithm effectively.

4.2 User Interaction Data

Assuming users' interactions (skips, likes, playlists) provide sufficient signals for accurate recommendations.

Risk Assessment

1. Algorithm Performance

1.1 Assess the risk of the recommendation algorithm not providing accurate or relevant suggestions, leading to poor user satisfaction.

2. Technical Risks

Address risks related to technical issues such as server downtime, app crashes, or compatibility problems across various devices and platforms.

3. Security Risks

Evaluate the risk of potential data breaches or vulnerabilities that could compromise user data or the app's integrity.

Test Plan and Test Report

Testing the app involved various types of testing to ensure its functionality, performance, usability, and reliability.

Test Plan

1. Functional Testing

User Scenarios

Test various user scenarios such as logging in, searching for songs, creating playlists, and receiving recommendations.

Feature Testing

Test each feature (like recommendations, top songs, etc.) to ensure they work as intended.

Limit Testing

Check the app's behavior at its limits (song title length, characters in song titles, etc.).

2. Usability Testing

- User Interface (UI): Evaluate the app's design, layout, and navigation for user-friendliness.
- User Experience (UX): Test how users interact with the app and their overall experience while using it.

3. Performance Testing

Load Testing

Evaluate the app's performance under varying user loads to ensure it handles traffic well.

Stress Testing

Determine the app's stability under extreme conditions (heavy loads, low memory, etc.).

Response Time Testing: Measure how quickly the app responds to user actions.

4. Compatibility Testing

Device Compatibility

Test the app on iOS and Android devices to ensure it works seamlessly.

5. Security Testing

Data Security

Verify that user data is encrypted and stored securely.

Authentication and Authorization

Test login/logout functionality and user access levels.

6. Regression Testing

Ensuring Changes Don't Break Existing Features after updates or changes, re-test previously working functionalities to ensure they are still operational.

Conclusion

This report highlights the multiple phases of the development of our Spotify Recommendations app. It is aimed to create a unique and personalized listening experience for existing Spotify users. The report outlines development highlights like UX/UI development and the creation of the algorithm using various scheduling and planning methods. It also details prototype testing and validation, by testing the functionality and UI.