Final Report for SP-29 Green - Spotify Recommendations App

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

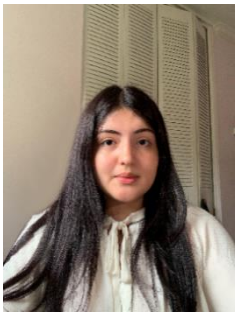
		
Sam Chandler	**Lezly Serrano**	**Jeremy Ray**

Table of Contents

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, such as their top songs and top artists, and customize the features of the recommendations that they are given. 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 recent listening activity
        ii. Display options for viewing top songs, artists, and 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

Figure 1. Gantt Chart

System Requirements

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, top songs and artists.

System Design

1. Introduction

This section 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.

This section 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.

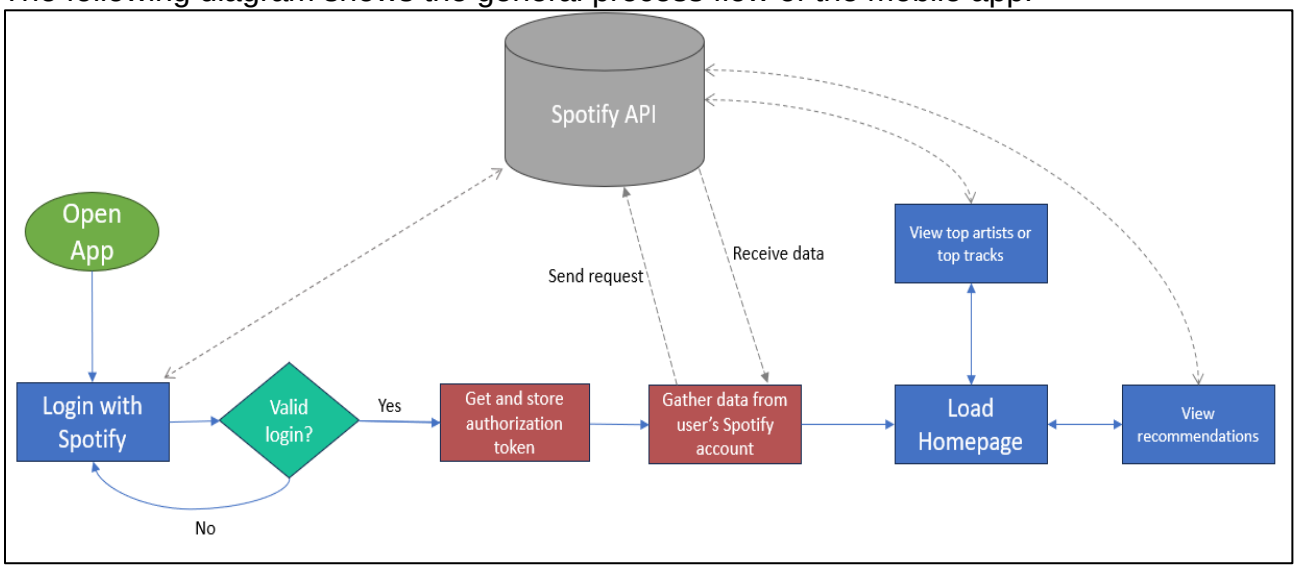
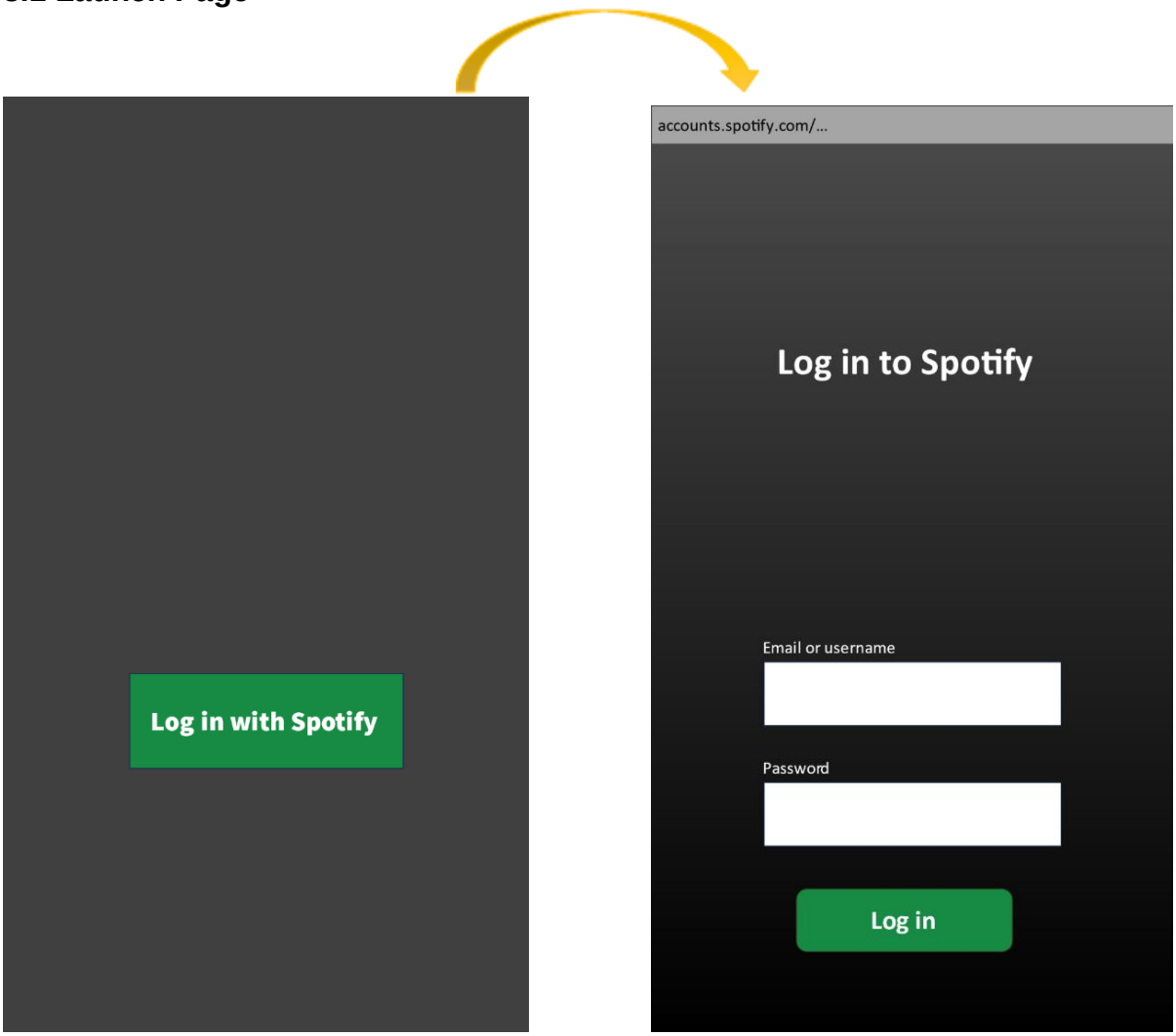


Figure 2. Flow Diagram

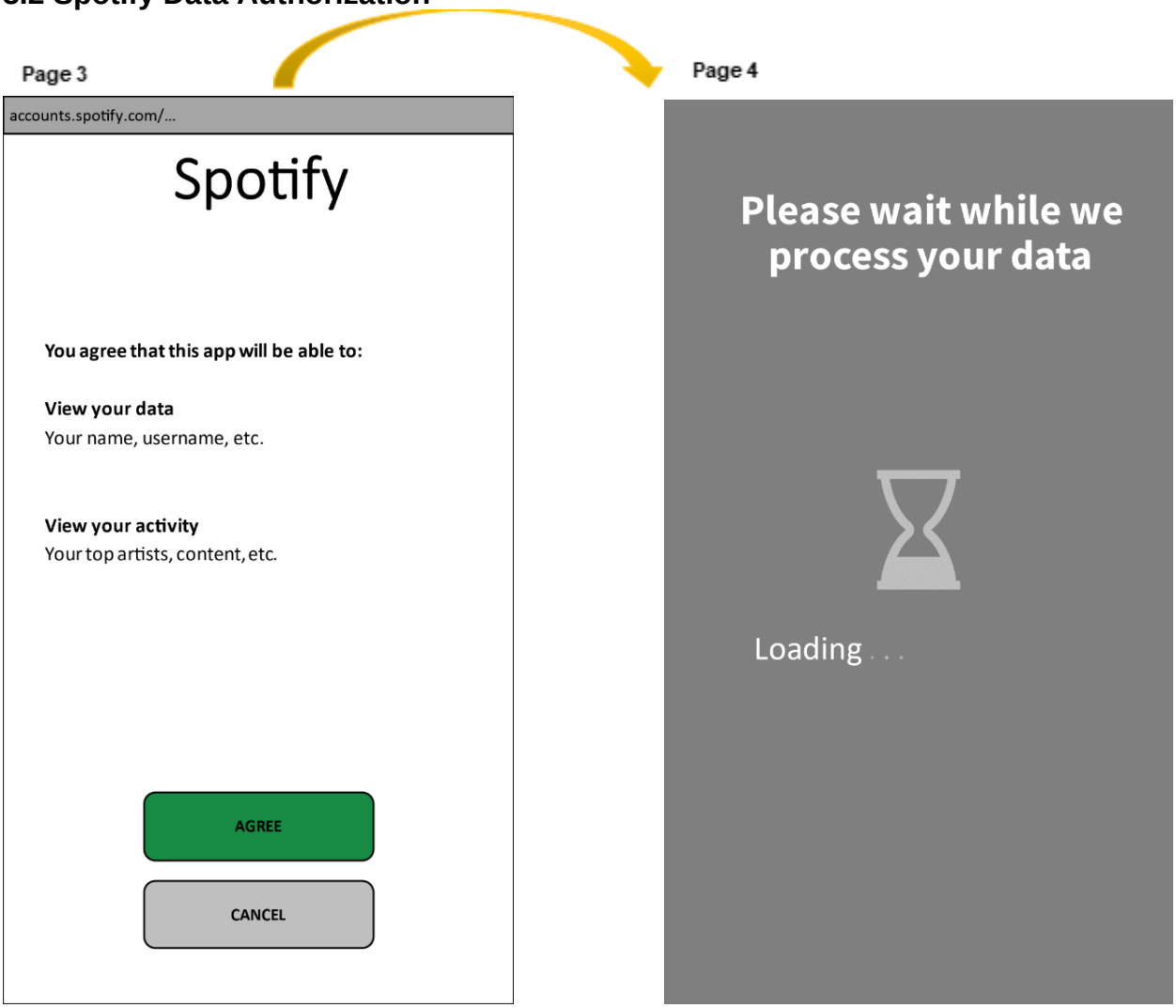
This diagram shows an abstracted view of how the app functions. The process begins when the user opens the app. The user proceeds to log into the app using their Spotify account. If the account is not recognized, the user will be shown an error page and then will be set back to the login screen. If the 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



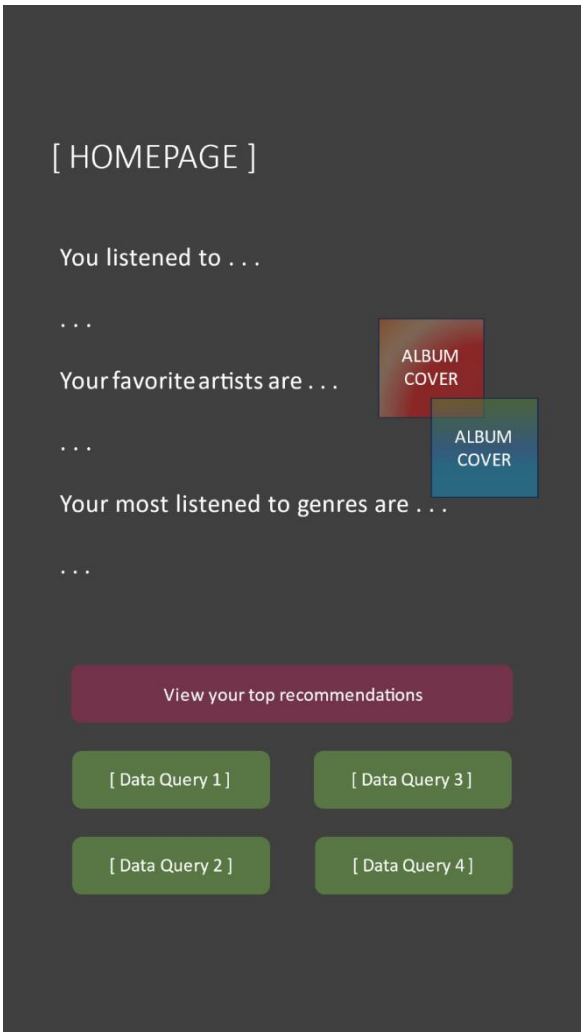
The Launch Page (Page 1) prompts the user to Log in using Spotify. The application uses Spotify's Web API, so logging in with their existing Spotify account is absolutely necessary to begin using the app.
Once the user presses the Log in button, they will be prompted with Spotify's log in page (recreated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

Development

Limitations

The 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 and how the app would be developed.

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. It was decided at the start that the app would run on iOS and Android (that was the primary benefit of using Flutter), so there had to be special care in approaching certain problems. There were many tough decisions that had to be made regarding the platforms that the app would be developed for. During the development of the Log In page, the code used Flutter WebView as the means of allowing the user to log in to their actual Spotify account. Unfortunately, WebViews are only compatible with mobile devices (Android and iOS), which meant that the app could no longer support web apps and Chrome apps. Flutter allows concurrent development for mobile and web apps, and web apps are nice because they can be tested without an emulator. Although web apps could no longer be supported due to the presence of a WebView widget, it ended up being palatable since the app was never planned for a web app release anyway.

Limitations also extend into the different mobile versions, and since the app was being developed for iOS and Android, the team needed to make sure that any added features were compatible with both of those devices. There are certain functions that may only work for Android and certain functions that may only work for iOS, so there are additional restrictions for that. Fortunately, the team was able to test both of these unique devices.

Dart, like any language, has restrictions as well. Any programming language has limitations based on what the language is capable of. So, with Flutter, the team had to add many additional packages in order to implement certain features. The WebView package along with the crypto package and the typed data package all had to be added for the sake of the simple log in. Since Flutter did not support all of these features by default, a lot of work went into learning what each individual package was capable of.

In terms of the Spotify Web API, there are additional limitations since the app is working within the bounds of what Spotify allows. Developers cannot do whatever they want; they are limited by the supported features and Spotify's terms and conditions. For example, developers are not allowed to make a trivia app out of Spotify data because that goes against Spotify's terms of service. Fortunately, this app was never intended to feature any trivia, but there are several restrictions like that example. There were several limitations that the team encountered during the creation of this app that were not initially realized. For example, developers cannot query the actual number of times that a user has listened to a specific song. This limitation is likely the case because Spotify wants to keep that specific data for themselves so that they may use it for the official Spotify Wrapped. Spotify Wrapped is an event that Spotify does every year where users are able to see data about their listening habits, and so Spotify likely wants to keep a lot of the features for themselves and have exclusivity for that. If any app could display this information, then users would not be as excited for the yearly event. All a developer can find out is which songs or artists the user has listened to the most; what their most listened-to song/artist is over one month, over six months, or over all time. These time ranges are also strict and obscure; roughly four weeks, six months, and "several years" are the only ranges that developers have for this data. App creators are not given access to the entirety of Spotify data, which had to be realized during the process of creating the app.

Software Development Life Cycle

All of this puts an important focus on the analyzing viability phase of the software test plan. Spotify's restrictions could be spotted early on if one is actively looking through the Spotify documentation during the analyze phase (right after idea phase). Doing this prevents that difficult position where the team is already in the implementation phase and designing or coding whatever they want the API to do, and then realizes that Spotify API cannot 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 (the one that seemingly had the most limitations). Flutter has a lot of flexibility, especially in external packages available for use. The development process for this app never really felt limited by Flutter or that the framework could not do something. But Flutter did impact how exactly features had to be implemented, and it's very different from other frameworks and other languages, so it all has to be learned through documentation.

This also puts a large emphasis on proper division of work, and if this project continued in development and got to the point of actual uploading to app stores and such for Apple and Android or Google Play stores, then there would really need to be a team that had a clear division of work. Developers having to learn all of the documentation on Spotify and learn how Spotify works is really not efficient, as they should be focused on programming. A lack of division leads to issues where a developer wants to implement something and so they go ahead and start designing code, but they do not have all of the knowledge of the API's capabilities. They're not completely knowing of the Spotify Web API, and so they'll encounter issues that make the coding process even more arduous. It is very important to have a division of labor where there is someone or a group or team that is dedicated to understanding the Spotify Web API and then

someone who is dedicated to understanding Flutter and someone who is dedicated to understanding iOS. Then, communication can flow easily between these individuals and work can be completed with less time spent rewriting or relearning. If the programmers have any questions or if they have any ideas for something 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. 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 thought necessary for a strict development cycle and a well-thought design plan.

Changes to the App

The app did end up changing a lot from what had been 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 specific data at the start. But then the team had to change that after it was realized that Spotify does not offer that much data to developers using the API. So certain functionalities had to be forgotten and left behind because of that. There were also visual things, on the Flutter side, that had to be changed. The design documents initially illustrated the homepage as having these low-opacity album covers that the user listens to display in the background, sort of animated and bouncing around. But the way that Flutter works is it uses widgets, and so it views an entire page on someone's screen as a widget, or a column or row of widgets. So a button is its own widget, a piece of text is its own widget, a column containing text would be its own widget. The way it is designed is very structured. There is a clear gallery of widgets that is available whenever one is using the app. The user may not see or realize that it's basically a grid of widgets of rows and columns and widgets stacked on top of each other. But during development, developers have to be aware of that since that's how the Flutter framework operates. It's very interesting and a very unique way of visualizing the structure applications. But it made implementing the idea of having album art bouncing around in the background very difficult because that's not really how Flutter works. There are no widgets behind widgets. There is some way to implement it using a Stack widget, but that just gets very complicated, and at that point the effort is not worth the cute visual effect.

One small and seemingly insignificant feature that was added to the app was the customization of the theme. There is a settings button at the top right that allows the user to change the theme. Some of the themes included 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. It just changes the colors of some widgets, so it does not change much. It only changes the primary color, which is the background color, and the accent color, which is the color of all the buttons and the headers and such. So this is a seemingly small change; it just changes colors. 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 the user's music in terms of circles or visualizing the user's music as fish in an ocean. And they do not do much beyond that. So, what ends up happening is the user uses the fun little Spotify Web API app once, it does something

cool, and then they forget about it and do not use it again. They just log out, deactivate it, and then never think about it again. And that's fine for a web application; if there is a website with many games and one of the cute things that is offered 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 designing a mobile app, it is really important that a user wants to stay and reuse the app. It is not good for a user to download the app, use it once, and then delete it. 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 the team had gotten to the point where the app is getting monetizing and ads are being included, it does us the app no good if users are using the app once. Users need to return continuously to be shown more ads; mobile developers want users to get back on. For an app like this, it may not necessarily need to be daily visits, but the user should get on every once in a while, to see what their listening activities are like. And so, if the environment is personalized for them and made to feel like it's their environment, they're likely to keep returning to this one app rather than finding an alternative. This 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 the development team was 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, it is a goal and a want for the app to feel fun to use or nice to look at and customizable. If development continued, the team 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. App creators not only have to be able to sell the app and make users think that it's worth downloading, but they need to ensure that once the user downloads it, they keep returning. And so, the customization was just a little way to show that.

The recommendations page is interesting because that was maybe something that the team had the least knowledge about going into this. It's very easy to understand how to 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, it requires a lot of design. The recommendation part of the app actually does not use the token at all when making a recommendation, but it does have to use the token if when using the person's favorite tracks as the seeds for the recommendations. 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. Developers can give Spotify’s API what tracks they want, and then

there is a lot of customizing in terms of what they want the recommended tracks to come back as. It is possible to 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 want to 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, whether a track has a lot of lyrics or mostly instruments. Another feature that was 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 are available for use in the recommendations. It is possible to change the tempo and the loudness of a track, but it seemed like a feature that most users would not care to fiddle around with. Thus, popularity, happiness, and how instrumental the track is ended up being the changeable features that were included since they seemed like the most reasonable options for users to change. It is also important to keep it simple so that the user isn't overwhelmed with options, especially when there are options that they do not understand. That is part of the term valence was changed to happiness, because the average user is probably not going to 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.

The final version of the app takes on a marketable name “Spotitops.” It was decided to give the app a legitimate name in the possibility that it would be added to app stores and need something better than “Spotify Recommendation App.” Spotitops works as a clever, short name as it incorporates Spotify, ‘top’ for top artists and songs, and is a palindrome. This report does not refer to the app as Spotitops throughout or include Spotitops in the title, as this is focused on the development and life cycle of the project and the building of the app, not the app itself.

Development Beyond the App

So, in terms of how this was set up and how each person contributed, Sam Chandler is the team lead, Lezly 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 there could be really clear lines. But when working with such a small group, there's bound to be overlap where there is someone who has a job description of writing documents that is also testing the app. There needs to be more testers and there needs to be more programmers. So, in the end, the team ended up being very flexible; the specific job titles did not amount to much, but they were still considered when delegating tasks.

In addition to making the app, it was also necessary to make a GitHub Pages website. GitHub Pages is a very useful tool that allows users to create websites. It uses markdown language, and users basically create the website by just editing a readme file. All of the website content is added in the readme essentially, and if a user wants a

certain line of text to look like a header, then they can add a single hashtag, and that would be Heading 1, and then two hashtags would be Heading 2, and so on. Users can customize the way that the webpage looks with those tools and can use underscores for italics or insert images or anything a normal website can do. But it's all done within the readme of the GitHub organization account, which is very cool and unique. GitHub Pages also uses Jekyll themes, and a Jekyll theme is basically a default theme that someone can use that has already built-in custom HTML and CSS styling with colors and such. Users can very easily, on GitHub's organization webpage, specify which Jekyll theme they want to use. There are hundreds of themes available online, but GitHub offers maybe 20 default themes to choose from that do not require any special downloading or referencing. The team ended up going with a theme called Cayman, which has a nice aesthetic. 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 the webpage for this project needed a little bit more, such as links to the YouTube video and the project report that display in the header, there needed to be additional work. It ended up being necessary to go into the HTML of the Jekyll theme and change the HTML as well as the CSS files, but that's just because of the specific requirements and wishes for the design of this website. Other websites may not require as many changes, but design was an important factor for this one.

GitHub Pages can be very useful for anyone, especially for individuals that just use the very basic functions and if you don't have much of a vision for the app. But in order to do any real customization of the look of the app (beyond default themes), then it is required to have some sort of HTML and CSS knowledge. But regardless, even though the website built for this project required edits to the design, it ended up being way easier to get kickstarted with GitHub Pages than it would be to build a website from the ground up. And this was the team's first-time using GitHub Pages, so it is actually a very interesting and intuitive process. GitHub Pages also allows users to provide their own custom URL and domain, so even if the idea of using GitHub Pages for a professional website was a bit looked down upon, it does not have to actually use the GitHub.io URL for it. The scope of this project did not require a more professional or advertisable domain, but it would be an important consideration for actual apps. This project, in addition to providing team members with knowledge on Flutter and Spotify Web API, resulted in the discovery of a new and easy way to create websites.

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

Logging In

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

Features

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, artist name length, 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

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

Test Report

1. Functional Testing

1.1 Logging in:

The app was logged in with multiple accounts. Newly created free accounts and premium accounts were tested and both worked. Additionally, multiple methods of logging in were tested (entering user and pass, using cookies, signing in with Google) and all succeeded. As suspected, accounts that are not allowed in the Spotify Developer App access cannot use the app yet. If the app is approved by Spotify, we suspect that it would be allowed.

1.2 Features:

The various features were tested, and all have been verified to be working as intended. These features include:

- ✓ Homepage with user information
- ✓ Top Artists (and all three time periods)
- ✓ Top Items (and all three time periods)
- ✓ Recommendations (with various different slider settings)
- ✓ Settings (with all themes tested)

1.3 Limit Testing:

Initially the limit test for song titles and artist names produced errors. If a song had too long of a title, it would extend beyond the screen and cause an error. Fortunately, the bug causing this has been fixed, and now the text wraps around if it is too long. The limit test was performed a second time after this fix, and this time the results were perfect.

2. Usability Testing

Usability testing included showing the app to others and gauging their reactions and ability to navigate the app. The app's layout is pretty simple and intuitive, so no one had difficulty with navigation. There were some comments on ways to improve the UI, such as including numbers for the items in the Top Artists and Top Tracks sections.

3. Performance Testing

The app performed pretty well under the normal conditions and worked well on both new and old Android devices. The only spikes in lag were experienced when first loading the homepage and first loading the top items pages.

4. Compatibility Testing

The app has been tested on both iOS and Android devices and works on both. There were not any features or widgets added to the app that were exclusively Android or exclusively Apple, so it was safe to assume that it would continue to be compatible on both.

5. Security Testing

5.1 Data Security:

The app uses secure storage for storing the user's authorization token, and never stores the data queried from Spotify anywhere. There are no leaks or potential areas where malicious users could gain access to data. Information is displayed right as it is queried. Since the app also uses Spotify for logging in, the user log-in information is never stored anywhere in the app. This is a major benefit of using the API directly on a WebView widget. There would have been additional security concerns if the user had to enter their Spotify information into the app itself to then be relayed to the Spotify Web API, but fortunately that was not the case.

5.2 Authentication and Authorization

Logging in and logging out both worked functionally well and do not present any leak problems. Once the user logs out from the app, their information is no longer being displayed or stored on the app anywhere (cloud or locally). Of course, if the user wishes to log in again, all the information can be quickly obtained from the Spotify Web API.

6. Regression Testing

The app had to be updated multiple times to add features or make small changes, and the app was never broken by these feature updates. All of the previous test reports were completed during and after the final build of the app. Testing features again after updating is made especially easy with Flutter's 'Hot Reload' functionality, which allows for the app to be tested right after extra coding has been done and without having to rebuild it. Hot Reload proved to be an immensely useful tool throughout the testing of this app.

Conclusion

This report highlights the multiple phases of the development of our Spotify Recommendations app. It is aimed at creating 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.

Designing the app proved to be a great learning experience and taught everyone on the team how to use Flutter and Dart for app development. It was the first time that anyone on the team had even installed the Flutter framework or Android Studio on their computers. Learning how to handle data from the Spotify Web API was also a brand-new experience for the team. Mobile development and API calls are two very common and very useful skills for programmers to have, and no other classes at KSU had taught us how to perform these tasks yet. So, the experience gained during this project was immeasurably valuable for the team.