SP27- Pinky Spotify App

Final Report

CS 4850, 04, Fall, 2024, September 4, 2024, Arthur Choi

Raehyeong Lee
Team Leader/ Developer

Jamia Jackson
Documentation/ Developer

Jordyn Jones
Developer/Testing

Amyr Murray
Developer/Testing

Website: https://spotifyappfall2024.github.io/about/

GitHub: https://github.com/SpotifyAPPFall2024/Spotifyapp

Lines of Code (LOC): 4,420

Number of project components: 16

Total man hours: 150 - 200

Table of Contents

1.0 Introduction

In the ever-evolving landscape of mobile applications and music streaming, our Senior Project team, Pinky Spotify App, in CS 4850 Section 04 for Fall 2024 at Kennesaw State University (KSU), is tackling the challenge of improving the popular Spotify app by integrating advanced features not currently offered. Consisting of team members Raehyeong Lee, Jamia Jackson, Jordyn Jones, and Amyr Murray, and under the guidance of Dr. Choi, our project aims to redefine the way users interact with Spotify through three carefully designed enhancements. These features are intended to enrich user engagement, provide greater customization, and promote collaborative listening experiences.

As one of the most widely used music streaming platforms, Spotify offers users access to millions of songs and podcasts on-demand. Its extensive library and algorithm-driven recommendations have reshaped music consumption worldwide. However, in our study, we identified several opportunities to further elevate user satisfaction and meet unmet needs, particularly in the areas of playlist management, podcast functionality, and real-time interaction among users. Our team aims to bridge these gaps, addressing user demands that go beyond the standard functionalities of the current Spotify experience.

The main objective of our project is to create a mobile app compatible with both Android and iOS platforms, ensuring an accessible and inclusive experience for a broad user base. Our proposed enhancements include a current queue playlist, advanced podcast transcriptions, and real-time collaborative playlists. Each feature has been strategically selected to amplify the app's usability, making it more intuitive and personalized for users.

The current queue playlist will allow users to have greater control over playlist ordering, enabling more flexible listening experiences. Advanced podcast transcription will enhance accessibility, providing on-screen transcriptions for audio content, which is particularly beneficial in diverse listening environments. Real-time collaborative playlists, on the other hand, will offer users the opportunity to curate and share playlists dynamically with friends, fostering a sense of community.

These enhancements will be integrated with the core Spotify app, using Flutter for cross-platform functionality and interfacing with Spotify's API for seamless operation. By undertaking this project, we aim not only to enhance the Spotify experience for current users but also to set a new standard for the app's future development. Our work will contribute valuable insights to the field of music streaming app development, and the end results will serve as a model for introducing user-driven features into established mobile applications. Ultimately, our project is centered on bringing greater personalization, convenience, and interactivity to Spotify users, positioning our team at the forefront of innovative app enhancement for popular digital platforms.

2.0 Project Goals

The three main goals of the Spotify mobile app are:

1. Provide a mobile app for both Android and iOS users to login and access the Spotify app.
2. This app will include three additional features to enhance the original Spotify mobile app.
3. The mobile app will essentially provide a better user experience than the original Spotify app.

3.0 Requirements

3.1 Functional Requirements

Our Spotify App provides a secure and streamlined login and registration experience, a critical component for user accessibility. New users must be able to create an account by entering a username or email address, and password. This ensures that users can securely register and access the app. For added security, new users will need to verify their account through a confirmation email, completing the registration process and ensuring the authenticity of user accounts.

The app's home page is central to the user experience, serving as the main hub for navigation and access to music content. The home page will display personalized options, including recent plays and recommended music, based on the user’s listening habits. This customized display makes it easier for users to access their favorite content and discover new music. Navigation within the app must be intuitive and efficient, allowing users to easily access various pages, such as the artist page, library page, and home page. The artist page enables users to explore the catalog of a specific artist, while the library page provides access to their personal playlists and songs. The home page lets users quickly return to the main interface, where they can see recent plays and recommendations.

Consistency in design is key to creating a cohesive user experience, and thus, the app’s design must maintain uniformity in color schemes and typography across all pages. Furthermore, the app must adhere to the Human Interface Guidelines for iOS and Material Design principles for Android, ensuring a familiar and accessible interface for users on both platforms. The graphic design of the app must also be adaptable, with graphics that can adjust to different screen sizes and orientations. This flexibility enhances the app’s usability across various devices, including smartphones and tablets, while ensuring that all visual elements remain clear and accessible.

Finally, Spotify App must be compatible with major mobile operating systems, specifically Android and iOS. The app must support Android version 9 and above, as well as iOS version 11 and above. This compatibility ensures that the app can reach a broad audience and function

seamlessly on a wide range of devices. Together, these functional requirements form the foundation of the Spotify App, delivering a user-friendly, secure, and visually appealing experience for music lovers across platforms.

3.2 Non-Functional Requirements

The Spotify App's non-functional requirements focus on ensuring user security, accommodating user capacity, and enhancing usability to create a trustworthy and engaging experience. Security is paramount, with Spotify's API providing robust data protection features, including an authorization process for user verification. Upon signing in with an existing Spotify account, users will also encounter terms of service agreement to consent to data usage within the app, aligning with standard data protection practices while fostering user confidence.

To handle user demand, the app will support simultaneous usage by multiple users, with real-time updates for collaborative features. For example, the collaborative playlist functionality will allow up to four users to make live modifications to the same playlist, ensuring that all participants see changes as they occur. This scalability is essential to supporting a seamless, multi-user experience as app usage grows.

Usability is key, ensuring that the app is easy and enjoyable to use. The design will replicate Spotify's color scheme for a familiar aesthetic, and the layout will be simple and intuitive, avoiding clutter and long text blocks that could overwhelm users. The result is a clean interface that helps users access all features quickly and efficiently, making their interactions pleasant and productive.

3.3 External Interface Requirements

The Spotify App's external interface requirements further enhance its usability and compatibility across various devices and platforms. The user interface design will be optimized for the latest smartphone displays, including those with top notches, and it will dynamically adjust to different screen sizes and resolutions on both Android and iOS devices. Hardware compatibility requirements ensure that the app will run smoothly on devices with Android version 9 and above, as well as iOS version 11 and above, maximizing its accessibility and functionality.

Software interface requirements will ensure effective data management through Spotify's API. Variables used to store data retrieved from the API will follow a clear naming convention, accurately reflecting the stored information for easier maintenance and scalability. For communication, the app will facilitate interaction between different instances, allowing data-sharing essential for real-time collaboration and playlist management. Furthermore, regular communication with Spotify's servers will keep user data and preferences up-to-date, thereby enhancing playlist updates, user customization, and other interactions integral to the app's performance and user engagement.

4.0 Analysis

During the analysis phase of the project, we conducted a thorough evaluation of various frameworks to identify the best development environment for our app. The selected framework needed to address not only the technical requirements of the project but also align with the skills and learning capabilities of the team. After researching, we analyzed three frameworks (Ionic, React, and Flutter to determine the best environment for the project.

4.1 Ionic

Ionic is an open-source UI toolkit for building performant, high-quality mobile apps using web technologies such as HTML, CSS, and JavaScript with integration for popular frameworks like Angular, React, and Vue. While Ionic aligns well with the team's existing knowledge in web development, its reliance on web-based technologies was a great limitation. The goal of this project was to build a mobile application for iOS and Android, Ionic's web-based rendering and dependency on WebView was not more suitable than other frameworks.

4.2 React

React is an open-source JavaScript library maintained by Meta, is widely recognized for its component-based architecture and virtual DOM, which enables efficient rendering and dynamic user interfaces. Coupled with its ability to integrate with other libraries and frameworks makes it one of the most adopted UI development environments today. However, React has a significant learning curve that was found to be too steep given the project's time constraints. While React offers great features the need to quickly learn the framework would have greatly delayed progress.

4.3 Flutter

Flutter is an open-source UI software development toolkit created by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. Flutter utilizes the Dart programming language which has incredible performance, a flexible UI, and cross-platform capabilities. These features and other features in the Flutter framework led us to overlook the learning curve for the development of our application.

After evaluating each framework thoroughly, the team decided to choose the flutter framework. This decision was based on its cross-platform efficiency, native-like performance, and comprehensive development tools that aligned closely with the project's requirements. By adopting Flutter, the team is confident in its ability to deliver a high-quality, responsive, and scalable application for both iOS and Android platforms.

5.0 Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a structured framework that guides development teams through various stages of building high-quality software. Our development process followed a systematic approach, ensuring that each phase was executed thoroughly to meet both functional and user experience goals.

Phase 1: Planning

The planning phase marked the initial stage of our project, where the foundation for the entire app was established. During this phase, our team discussed the core objectives of the app, defining its purpose, target audience, and proposed key features. We collectively aligned the goals of the project, ensuring that we had a clear vision of what the app should accomplish. Initial ideas and use cases were brainstormed, and a preliminary project timeline was created to guide the development process. This phase was essential for setting expectations, prioritizing tasks, and ensuring all team members understood the project's scope and deliverables.

Phase 2: Analysis

In the analysis phase, we focused on understanding the specific needs of the app's intended users. Through research and user feedback, we identified potential areas for improvement in the original app and explored new features that would enhance the user experience. The insights gained allowed us to refine our requirements, creating a more concrete understanding of the features and functionality needed in the system. By the end of this phase, we had a clear set of requirements to guide the next steps in the project.

Phase 3: Design

The design phase was dedicated to developing the architecture and user interface of the app. To begin, we created flow maps for each feature. This was great to determine where each feature would seamlessly fit into the Spotify user interface. During this phase, we decided on the technical structure and design methodology that would best support the project. Our team opted to implement a high-level structured BLoC (Business Logic Component) code architecture to ensure clean, modular, and maintainable code. This approach allowed for easy separation of concerns and improved scalability. With the foundational design established, we confirmed that the system's requirements were being met, thus paving the way for the development phase.

Phase 4: Development

The development phase involved the actual coding and implementation of the app. Following the design specifications, we began by building the base app, which served as the foundation for the core functionality of the app. This initial version of the app was developed without the additional features, ensuring that the primary functionality was fully operational. Subsequently, new features were added in phases, following a defined timeline with specific deadlines. Each new

feature was integrated into the base app in a controlled manner to ensure smooth functionality and minimal disruption to existing features. This iterative approach allowed us to track progress, maintain high-quality standards, and quickly address any challenges that arose during development.

Phase 5: Testing

Testing was an ongoing process throughout the development cycle, ensuring that every aspect of the app was thoroughly validated. At the completion of each development milestone, we conducted testing to identify and resolve bugs early in the process. By performing unit tests and integration tests after each task, we ensured that each feature worked as expected before moving into the next phase. At the conclusion if the development phase, a comprehensive system test was performed, executing the entire app to verify that all functionality both core features and additional features worked as intended. This final round of testing confirmed the stability, performance, and overall quality of the app before deployment.

In conclusion, by following the structured phases of the SDLC, our team was able to efficiently develop, test, and refine the app to meet both functional and non-functional requirements as well as user expectations. The iterative nature of the process, with testing at each stage, helped us maintain high standards of quality and ensured the app was free of major bugs. With the successful completion of the development and testing phases, we are now prepared to proceed with deployment, confident that the app is both reliable and user-friendly.

6.0 Design

6.1 Assumptions and Dependencies

The Spotify App relies on specific software, operating systems, and user characteristics for effective performance. This project was developed using Flutter, which serves as the core framework, with a dependency on Spotify’s API to access Spotify’s music catalog and user data. Changes to the API, such as updates or alterations, could impact the app’s functionality, requiring timely adjustments to maintain compatibility. The app is designed to support both iOS and Android platforms, and while it is tested on current OS versions, it is expected to remain compatible with future OS versions. Additionally, users are assumed to have basic smartphone operational skills, ensuring they can easily navigate the app’s interface and utilize its features without needing extensive technical knowledge. However, future updates to Spotify’s app or API may necessitate modifications to the app to align with new functionalities and ensure ongoing compatibility.

6.2 General Constraints

Several constraints impact the app's hardware, software, and operating environment. The app's compatibility extends to devices that currently support Spotify, ensuring users experience minimal issues across various devices and OS versions. It is built to function in diverse network conditions, from high-speed Wi-Fi to slower mobile data, making it accessible in both urban and remote environments. Additionally, the app depends on the constant availability of Spotify's API. Any downtime or updates to the API could affect key features, such as music streaming and playlist management, which rely on a continuous connection to Spotify's resources. Compliance with Spotify's development standards, as outlined in their Developer Terms and Conditions, is essential to avoid issues with third-party integration. The app must also adhere to data protection regulations, including GDPR, to safeguard user privacy and data security.

Given the varied devices that will run the app, memory and processing requirements are carefully managed to ensure smooth performance on devices with limited resources. The app minimizes latency and provides quick responses to enhance user experience. Network usage is optimized, ensuring that data consumption remains low while maintaining reliable streaming and smooth interactions. Verification and validation processes are embedded in the app's development, with extensive testing required to meet both functional and non-functional requirements.

6.3 Architectural Strategies

The architectural strategy behind the Spotify App includes using Flutter as the primary development framework, chosen for its efficiency in producing a cross-platform app with a single codebase. While React Native was considered, Flutter's superior performance ultimately made it the framework of choice, enabling faster development and reduced time-to-market. Integration with Spotify's API is fundamental to the app, as it enables access to Spotify's music catalog, playlists, and user accounts, enhancing the app's functionality. Spotify's API also supports OAuth 2.0, a secure authentication and authorization protocol, which allows users to log in securely, thereby strengthening user trust in the app's data handling practices.

6.4 Enhancing the Software

Several enhancements make the Spotify App a unique and accessible experience for users. Queue playlist management is improved using Flutter's drag-and-drop functionality, which enables users to easily reorder items in their playlists, creating a user-friendly and interactive experience. Podcast transcription features are added to improve accessibility for users who are hard of hearing, utilizing speech-to-text APIs to convert audio into text, making content accessible for a broader audience.

Real-time collaborative playlist functionality enhances the social aspects of the app, enabling friends to share music experiences instantaneously. Firebase is utilized for this feature, enabling real-time updates and synchronized playlist modifications, thereby creating a dynamic, shared experience for users. The app's user interface is built using Flutter's widget-based design, which ensures that the interface is visually appealing and responsive. Custom widgets add to the unique user experience, while adhering to Material Design guidelines, eliminating the need for a custom UI framework.

The app also incorporates robust error detection and recovery mechanisms. Using Flutter's native error handling, along with custom error management for Spotify's API interactions, ensures that runtime exceptions are handled gracefully. When errors occur, informative messages are displayed to help users understand and recover from issues. Flutter's automatic memory management, which includes garbage collection, optimizes memory usage and reduces potential memory leaks, ensuring the app's stable and efficient performance.

Concurrency and synchronization are handled through Flutter's asynchronous programming model, allowing for background processing without blocking the main thread. This enhances the app's responsiveness, especially during long-running tasks like API calls, and supports real-time collaborative playlist functionality by handling updates concurrently. Lastly, communication mechanisms within the app involve HTTP requests for Spotify API interactions and Firebase for real-time features, balancing simplicity with dynamic interactivity to provide a smooth and engaging user experience. Together, these design elements ensure that the Spotify App is reliable, user-friendly, and responsive, offering an enriched music experience for Spotify users.

6.5 Overall Workflow

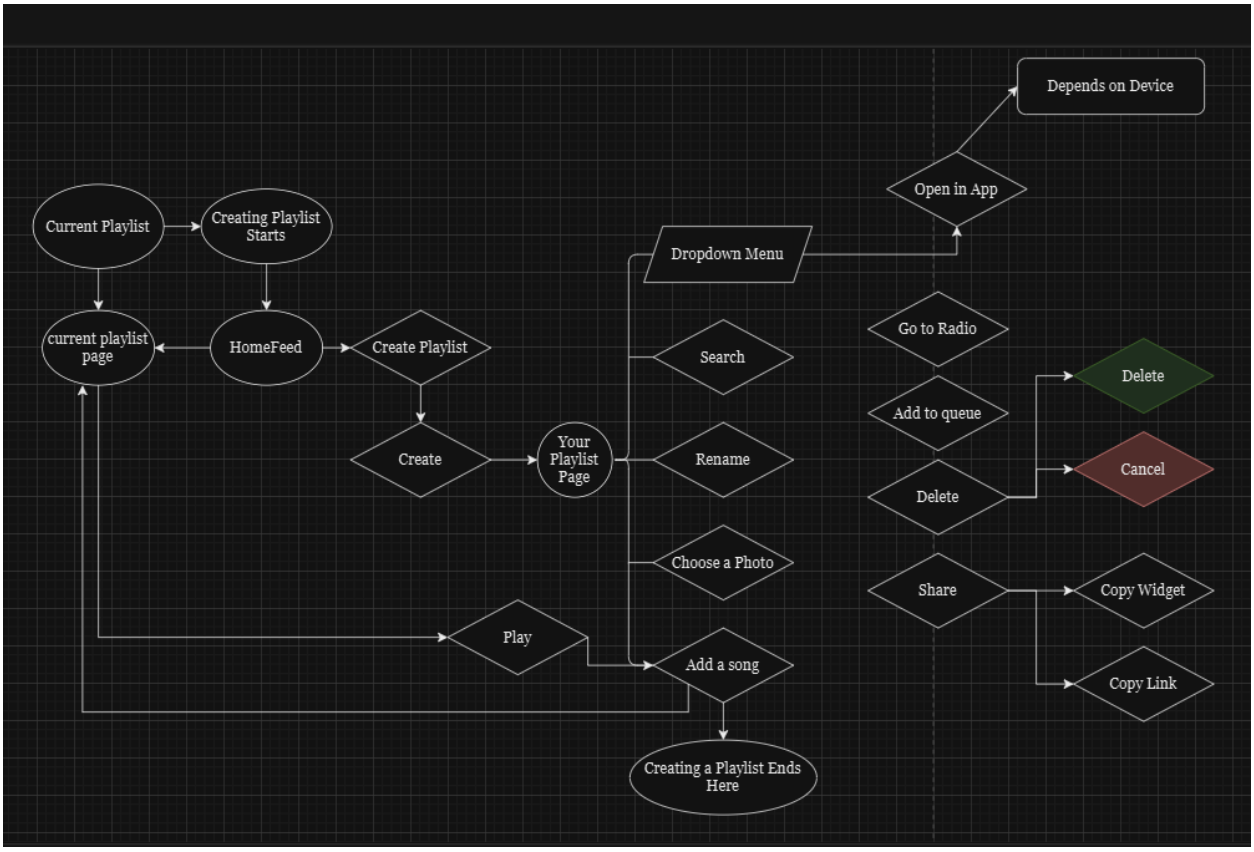


Figure. 2 Overall App Workflow diagram

This diagram outlines the initial workflow of the Spotify app before implementing the project, highlighting the process of creating and managing playlists. It begins with the "Current Playlist" feature, which can be accessed from the "Current Playlist Page" or by starting the playlist creation process. From the "HomeFeed," users can initiate the creation of a new playlist by navigating to the "Create Playlist" option. After choosing "Create," the workflow transitions to "Your Playlist Page," where users can perform actions like renaming the playlist, selecting a cover photo, searching for songs, and adding tracks to the playlist. A dropdown menu offers additional options, such as playing the playlist, sharing it, adding songs to the queue, or deleting the playlist. The "Open in App" option, dependent on the device, provides further functionalities like sharing via widgets or links. The process concludes with adding songs, completing the playlist creation workflow. This structure emphasizes flexibility in playlist management while maintaining user interaction across multiple features.

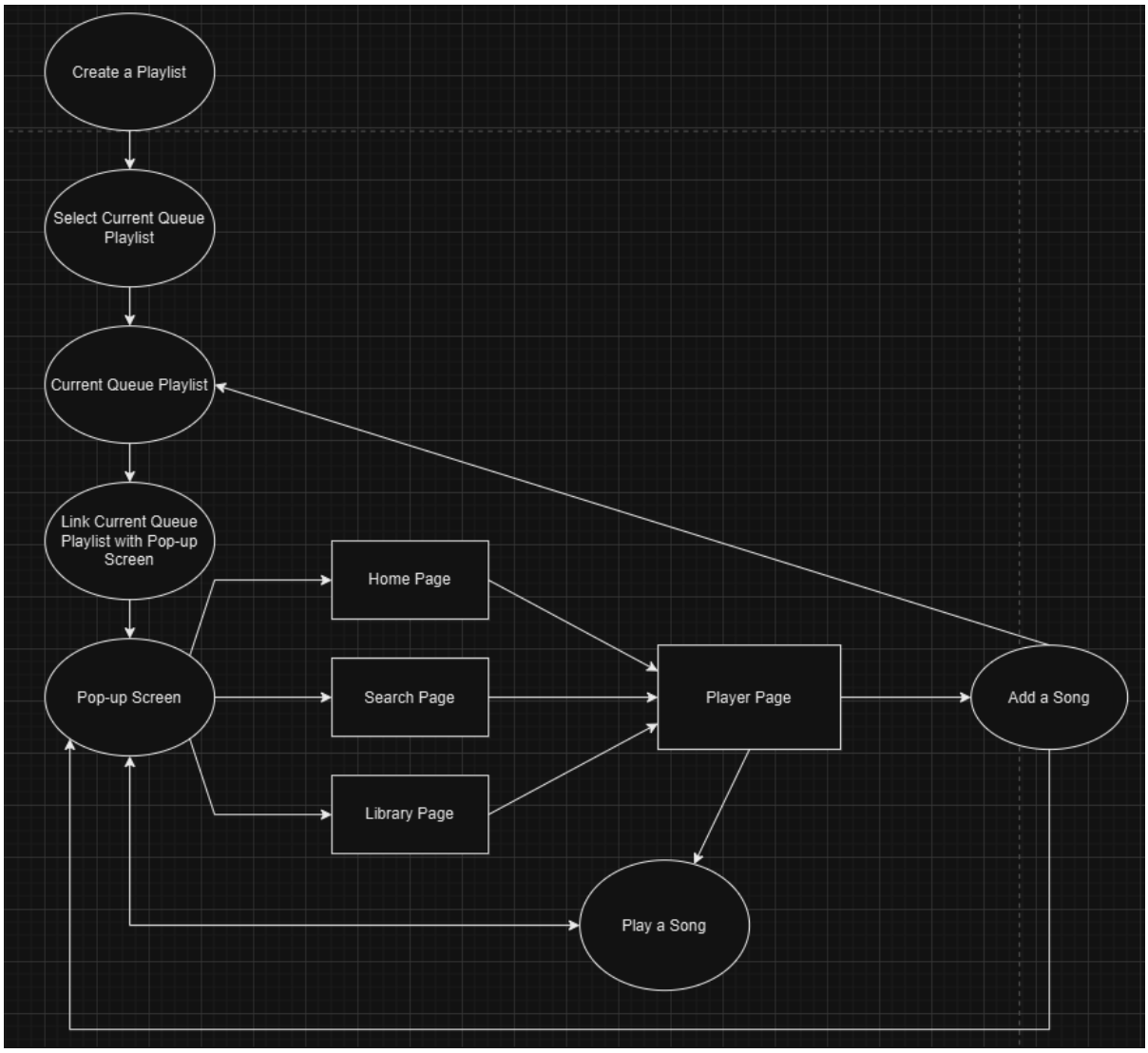


Figure. 3 Current Queue Playlist Workflow Diagram

The diagram illustrates the flow of functionality in the Spotify app, focusing on the integration of a "Current Queue Playlist" with a pop-up screen. The process begins with the creation of a playlist, which is then selected as the "Current Queue Playlist." This playlist is linked to the pop-up screen, ensuring dynamic interaction across various app pages, including the Home Page, Search Page, and Library Page. The pop-up screen serves as a central element, allowing users to manage the Current Queue Playlist seamlessly. It connects to the Player Page, enabling the playback of songs and the addition of new tracks to the playlist. The Player Page further facilitates playing a song while maintaining synchronization with the Current Queue Playlist. The flow ensures that playlist management is accessible and interactive, integrating with key app features for a cohesive user experience.

7.0 Development

To start the development phase of the project we delegated tasks amongst all individuals. To start Raehyeong who is also our team leader was tasked with the development of our project website. Jamia was given the responsibility of developing the base app which is initially the spotify app with no additional features. After Raehyeong concluded the development of the website he then began to develop the search page for the application as well as the queue playlist feature. As we decided on three additional features of the application. The other features went to Jordyn and Amyr with Jordyn tackling the podcast transcription feature and Amyr tackling the Real-Time playlist feature after midterm presentation. The final application at this time is functioning as intended but without the podcast transcription feature due to trouble encountered with Speech to Text APIs.

8.0 Testing

8.1 Test Plan

To ensure the software met the initial requirements defined during the Software Development Life Cycle (SDLC), a comprehensive series of tests were conducted the results of these tests are presented in a structured table format, compromising of three columns: the feature being tested, pass status, and fail status with the failure status also including corrective action. In cases where a feature failed to meet the test criteria, the table also includes a priority level for addressing the issue, ensuring a focused and efficient resolution process.

8.2 Test Report

Feature	Pass	Fail
Login Page/ Sign Up page	X	
After successful login users should be directed towards home page	X	
The system allows user to enter email and password into Sign in pages	X	
When entering user password, it is hidden	X	
Users should be directed to authentication page to provide permission after login	X	
Users can access all features of the app after successful login	X	

9.0 Version Control

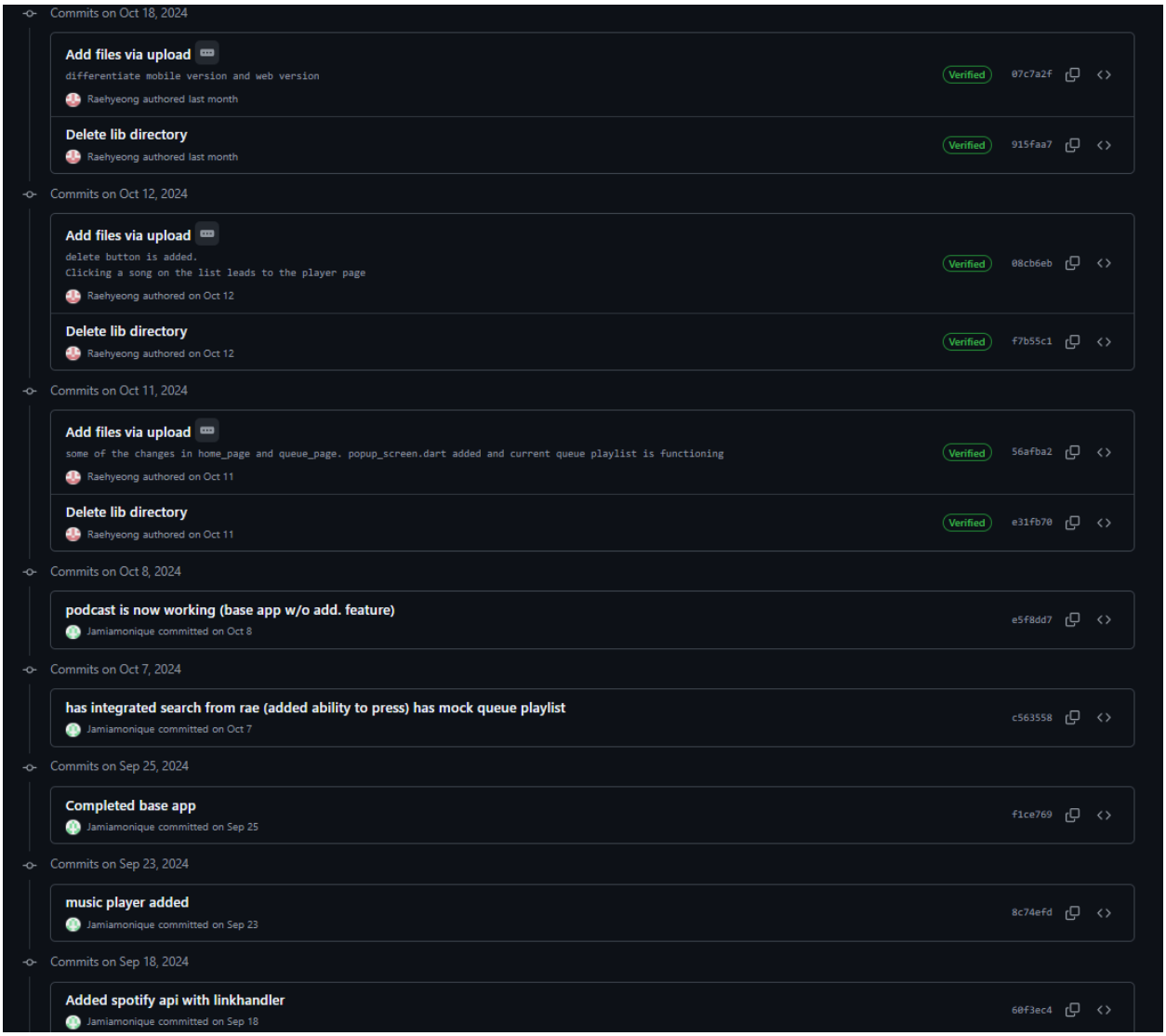


Figure.1 Current Queue Playlist Version Control.

The image displays a version control log for implementing the "Current Queue Playlist" feature in Spotify app. The commit history shows several updates, particularly highlighting the addition and refinement of the Current Queue Playlist functionality, which allows songs to be added to a playlist and displayed on a pop-up screen. Other enhancements include differentiating mobile and web versions, integrating a music player, adding podcast functionality, and allowing users to navigate to the player page by clicking a song. This structured version control process documents the progress and iterative improvements made to the playlist feature.

10.0 Screen Mockups

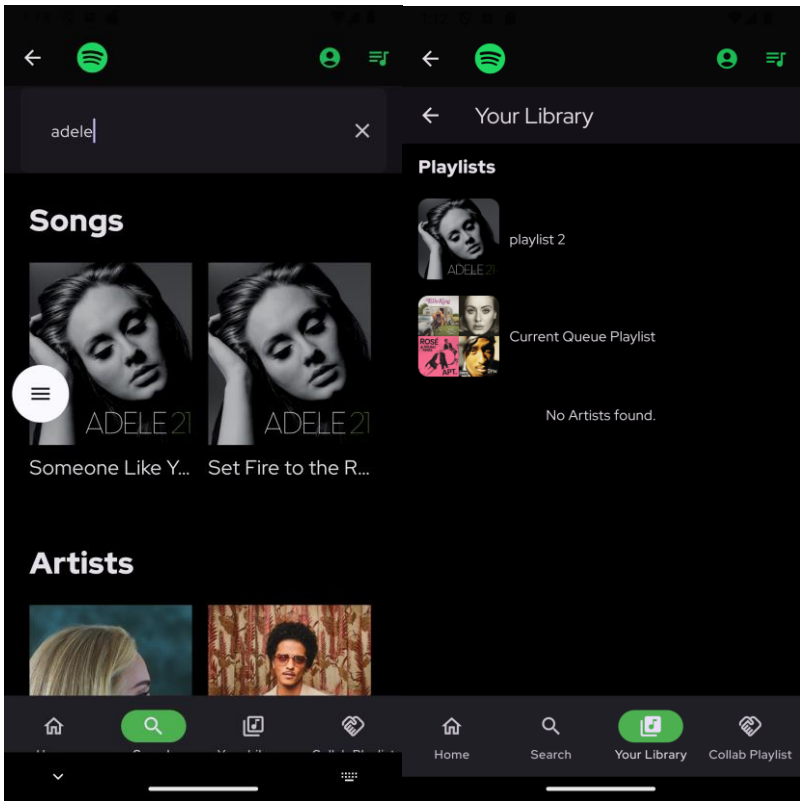


Figure 1. Home Page, Search Page, and Library Page

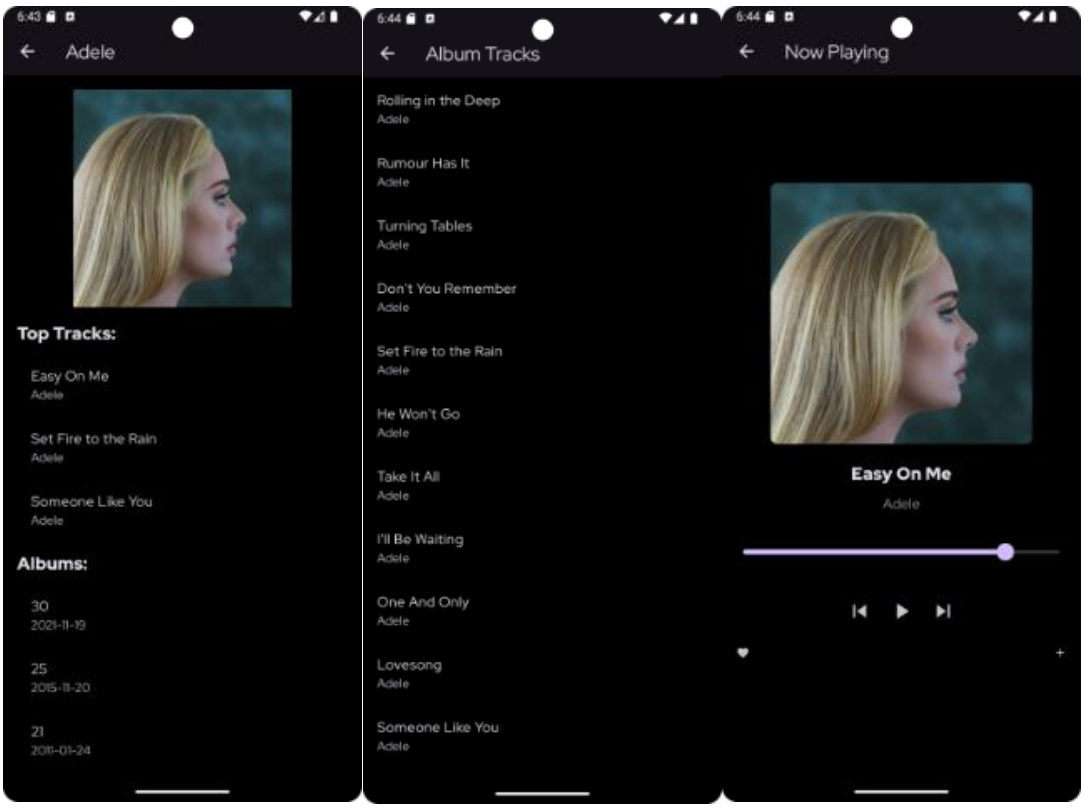


Figure 2. Album/ Playlist pages

sd

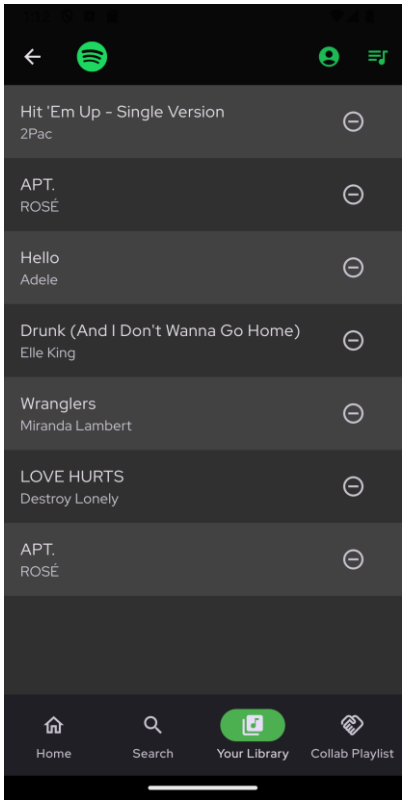


Figure 3. Current queue playlist page

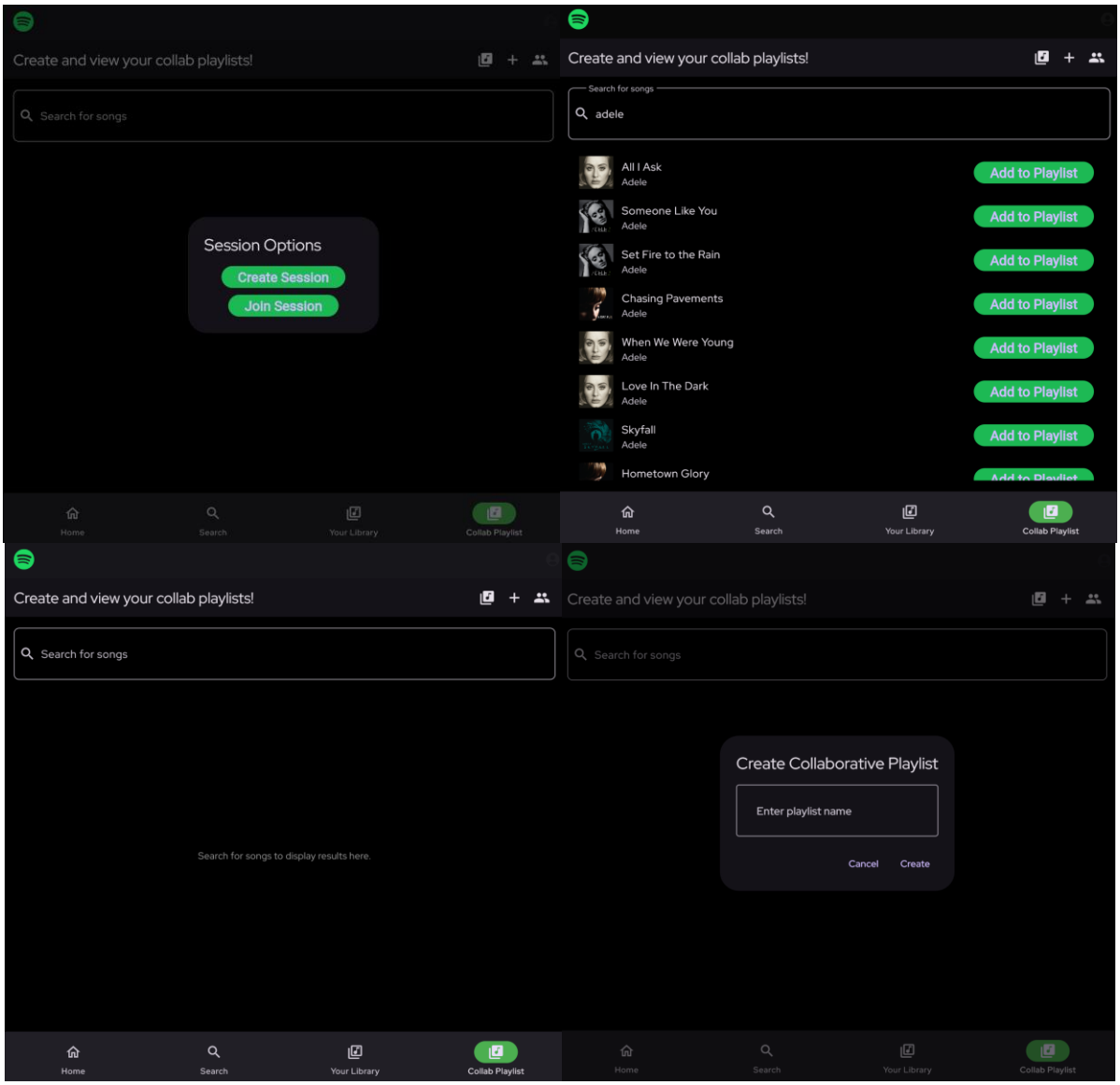


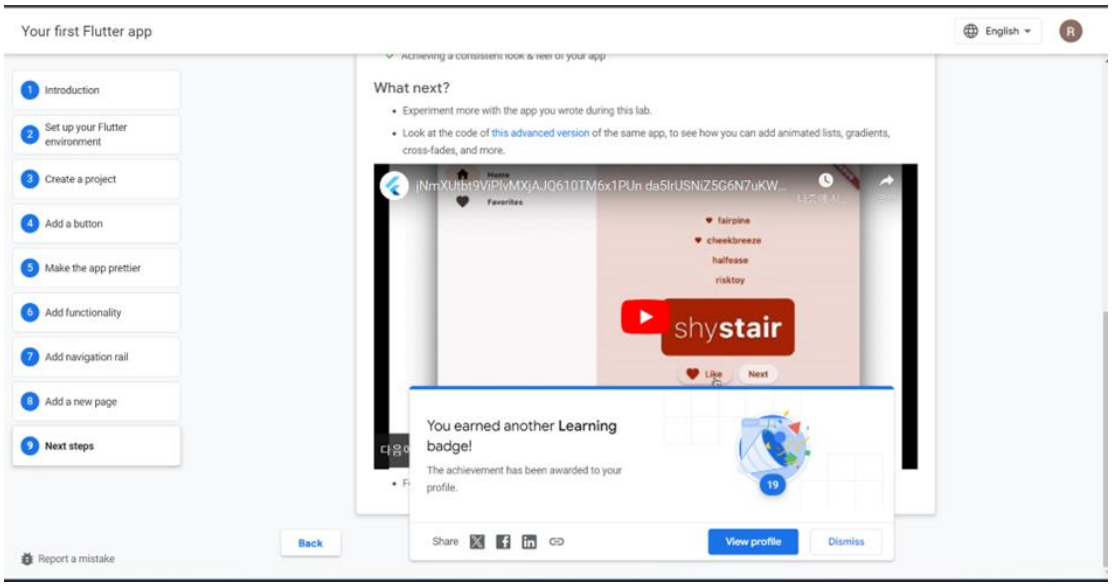
Figure 4. Collaborative Playlist

11.0 Conclusion

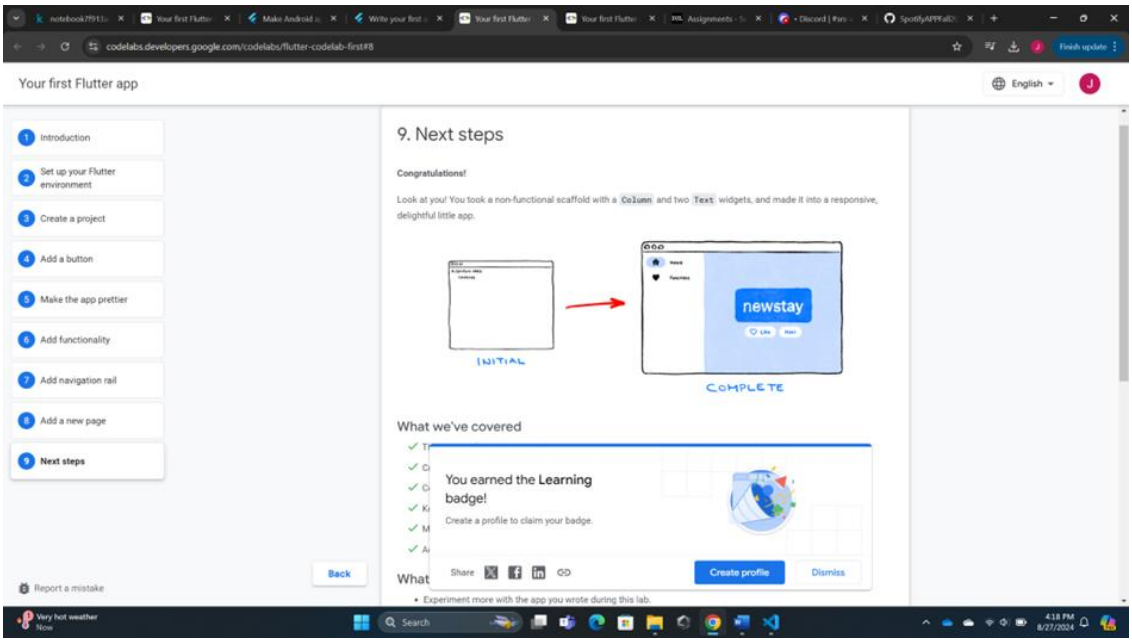
In conclusion, the project was deemed successful at creating a Spotify Mobile app despite the podcast transcriptions feature not being implemented. The team learned valuable new skills surrounding the Flutter framework, Mobile development and Rest APIs. The time spent on developing this project and the satisfaction that we have completed something that we are proud to standby outweighs the dissatisfaction of not being able to implement the podcast transcriptions. We are certain that the podcast transcriptions will be implemented into our app one day along with other features to further enhance our app as well as to continue the growth of our skills in Mobile Application Development.

12.0 Appendix

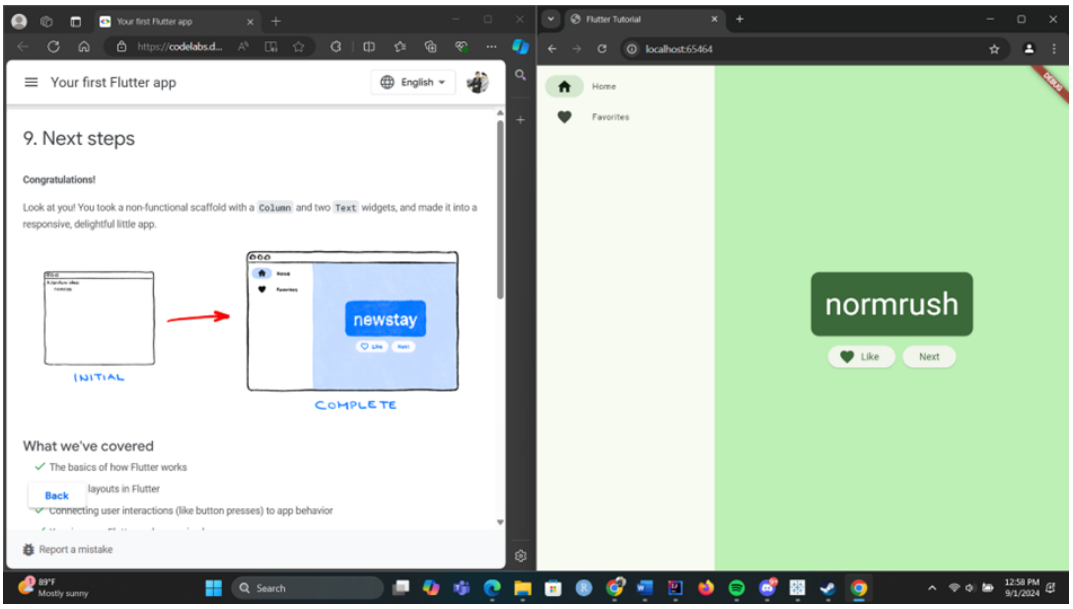
Raehyeong's Flutter Tutorial:



Jamia's Flutter Tutorial:



Amyr's Flutter Tutorial:



Jordyn Jones' Flutter Tutorial: