PlayBack – Software Implementation & Testing Report

Gaetano Panzer II – Project Manager,

Max Collins – Requirements Engineer

Ryan Farrell – Software Architect

University of North Carolina - Wilmington

May 2025

Abstract

This report documents the implementation and testing of PlayBack, a web-based platform that encourages users to engage with music in a personal and uninfluenced way. Designed to move away from traditional rating systems and public reviews, PlayBack allows users to reflect on their music experiences by sharing brief, tweet-length thoughts and rating songs with personalized "vibes." The application was developed using React for the front-end and Flask for the backend, with an emphasis on clean user interface design and responsive performance. Core features include user authentication, post creation, and a private rating system. Testing involved unit testing for backend logic and integration testing to ensure seamless communication between components. Overall, PlayBack met its functional requirements and performed reliably during testing, demonstrating its potential as a reflective, user-centered music platform.

Introduction

Description of Problem

Music listeners today lack a streamlined way to engage with their listening habits in a personal and meaningful way. Traditional review sites push numeric scores and dense analysis that dictate perception before listeners even have a chance to form their own thoughts. Research shows that numeric ratings and reviews often shape users' initial perceptions of music, limiting the opportunity for independent exploration (Swinkels, 2023). Community-driven platforms, while more focused on user opinions, still rely on aggregate scores that shape expectations (Ezquerra Fernández, 2024). Even when reviews are user-generated, they can be overwhelming to sift through, making it hard for listeners to freely explore their thoughts without being influenced by others.

Meanwhile, streaming services like Spotify and Apple Music use algorithms that reinforce familiar listening patterns, limiting discovery. As music recommendation systems tailor suggestions based on past listening habits, they can inadvertently stifle creativity and limit musical diversity (Mehdi Louafi & anon, 2024). This algorithmic reinforcement can create echo chambers where users are constantly exposed to similar genres or artists, preventing them from stepping outside their comfort zones (Sánchez-Moreno et al., 2020). And if someone listens across multiple platforms—or outside of them entirely, like in a friend's car or at a party—their listening history becomes fragmented, leaving them without an accurate way to reflect on what they've been into.

As a result, there is no easy way for listeners to track their music experiences, engage with their opinions without external influence, or break out of the patterns dictated by algorithms and rating systems. The lack of autonomy in music discovery and reflection limits the potential for personal growth in musical taste. When listeners rely on numeric scores or algorithmic recommendations, they are more likely to form preconceived opinions about music before experiencing it for themselves. This stifles creativity, reduces diversity in musical preferences, and reinforces conformity to popular opinion (Schäfer et al., 2013).

Moreover, fragmented listening data across platforms makes it difficult for users to reflect on their personal music journey. This results in lost insights into their evolving preferences, reducing the potential for meaningful self-reflection and discovery. Solving this problem creates an opportunity for users to engage with music in a more authentic, exploratory, and introspective manner. Our website is not a typical music review platform. It is designed to offer users a more personal and uninfluenced way to engage with music. Unlike traditional review sites, we do not display a community-wide numeric rating system. While users can privately rate albums for their own reference, these scores remain private, removing the pressure to conform to an aggregate rating and encouraging a more personal connection with the music. Without a visible consensus dictating expectations, listeners can form their own opinions free from external influence.

Additionally, users must post their thoughts before gaining access to others, ensuring that their first impression of an album is their own rather than a reaction to existing opinions. Beyond thoughts, the site is intended to function as a tool for tracking listening habits. Although the functionality for tracking listening data has not been implemented yet, users will be able to manually enter their data in the future, allowing them to reflect on their personal music journey in a way that algorithms cannot capture. This will empower listeners to see trends in their tastes—how their favorite genres shift over time, how often they revisit certain albums, or how frequently they explore new sounds.

This platform benefits both casual listeners and seasoned music enthusiasts. Casual listeners who may feel intimidated by traditional reviews have a space to engage without fear of their thoughts being scrutinized, while experienced listeners can challenge themselves to distill their thoughts into concise, focused reflections. By prioritizing personal engagement over consensus, our website fosters a more intentional, uninfluenced, and exploratory way to experience music.

As with any platform that involves user-generated content and personal data, it is essential to consider the ethical, security, legal, and societal implications of its design and functionality. Our website prioritizes user privacy, data security, and ethical engagement, ensuring that listeners can freely and safely interact with their music experiences.

The platform is designed to avoid bias and unfair exposure. Traditional review sites often amplify popular or trending opinions, creating an echo chamber where mainstream views dominate. By keeping user ratings private and requiring users to post their own thoughts before seeing others, the platform levels the playing field. This prevents more popular albums or opinions from overshadowing niche or minority perspectives, promoting equal representation of all musical voices.

Ensuring the security of user data is a top priority. Since the platform involves tracking personal listening habits, all sensitive data—including private ratings and user-generated content—is encrypted and stored securely. Users will have control over the visibility of their posts, allowing them to share thoughts publicly, privately, or with friends.

To further protect privacy, the platform does not engage in third-party data sharing or sell user information to advertisers. Any integration with external services, such as Spotify's API, will adhere to strict authentication protocols, ensuring that only the necessary data is accessed.

The platform adheres to standard data protection practices, including giving users the right to access, modify, and delete their data. Additionally, any integration with Spotify's API will comply with the service's licensing agreements and terms of use. The platform does not store or distribute copyrighted music, ensuring compliance with intellectual property laws.

The platform promotes a more intentional and meaningful way to engage with music, shifting away from the superficial consumption patterns fostered by traditional review sites and algorithm-driven platforms. By prioritizing personal reflection over consensus, it empowers listeners to develop individual taste rather than conforming to popular opinion.

Additionally, by providing tools for manual listening tracking (to be implemented in the future), the platform will encourage users to reflect on their musical journeys over time. This will foster a deeper connection to music and promote more mindful listening habits.

Our platform offers a level playing field for all users—whether casual listeners or seasoned enthusiasts. By keeping numeric ratings private and removing public comment sections, it reduces the influence of social clout, creating a space where all opinions hold equal weight, free from external validation or judgment.

Project Scope & Objectives

PlayBack is a web application designed to offer users a personal and uninfluenced way to engage with music. The platform removes external pressures like numeric ratings, aggregate scores, or social comparison, allowing users to form their own opinions freely. It aims to foster a more intentional and exploratory music experience by encouraging users to share personal thoughts on music. Loving a track through this method has a greater positive impact on user satisfaction than hating one has a negative impact (Garcia-Gathright et al.). Users can create profiles and securely log in, but the system currently lacks the ability to allow users to track their listening habits or offer insight into personal taste trends over time.

The core features of PlayBack include a personalized "thoughts" system where users can express themselves about music without numeric ratings or aggregate scores. The posting system allows users to write tweet-length "thoughts" to capture their reactions or feelings about a song or album. While direct messaging (DM) functionality has not been implemented yet, it will eventually allow private communication between users, fostering more personal engagement. The platform will enable users to track their listening habits, with mechanisms for recording how many times they've listened to a song or album and offering monthly and yearly recaps to reflect on how their tastes evolve.

Users will have control over the visibility of their posts, with options for public, private, or friends-only settings. Privacy features could be further refined in the future to allow users to have more granular control over each post, enhancing the personal nature of their engagement with music.

Users will be able to rate music privately without these ratings being visible to others, allowing for personal reflection without external comparison. Comment sections on posts and automatic content generation, like playlists or stations, are not part of the design. Users will eventually be able to play songs through Spotify's API, allowing for seamless integration with their music listening experience. This feature will support music discovery but will remain focused on the reflective, personal nature of the platform. Profile customization, such as decorating profiles based on musical preferences, is a low-priority feature that could be explored once the core features are implemented.

The main objectives for PlayBack are to complete key features such as secure profile creation and login, integration with Spotify's API for displaying data and enabling music playback and offering a system for manually inputting listening statistics. The platform must be user-friendly and intuitive for both casual listeners and seasoned music enthusiasts.

The posting system will be implemented to enable users to express their thoughts about music in a concise and personal way, free from the influence of numeric ratings. All data about user listening habits will be stored securely, and users will be able to track their listening history manually, offering insights into their music journey with monthly and yearly recaps.

Direct messaging functionality will be added to allow users to privately interact with others while maintaining a focus on non-intrusive, personal engagement. Privacy options will be available to give users control over the visibility of their posts, ensuring they feel empowered in how they share their music experiences. Users will have the ability to decide if their thoughts are public, private, or shared with friends only.

The platform will avoid the inclusion of numeric ratings, comment sections, or recommendations based on others' ratings. The focus will remain on creating a reflective space where users can explore and track their tastes, allowing for meaningful engagement with music, free from the pressure of outside opinions, social media dynamics, or the need to conform to others' tastes.

Success Criteria

The functional requirements have been partially met. Users can create accounts and post "thoughts" about music, and they can control the privacy settings of these posts, choosing whether to make them public, private, or share them with friends. However, users cannot interact with others' posts at this time, such as liking or commenting, and user-entered statistics are not yet implemented. Additionally, the platform does not currently track listening habits or allow users to manually input their statistics. Integration with the Spotify API to listen to tracks directly within the platform is not yet fully functional or implemented.

In terms of user satisfaction, we expect at least 80% of early users to report a positive experience with the site's navigation, design, and personalization features. Feedback will be important to gauge how well the "thoughts" format encourages engagement and reflection, even though interaction features are not yet available.

The website should strive to load in under 3 seconds for 90% of users to ensure a smooth and efficient browsing experience. There should be no critical bugs or crashes during the first month of public testing, which will be monitored closely. Engagement metrics will focus on user posts, as interaction features are not yet available. The goal is to have at least 20 posts in the first three months and consistent active users who return to post their "thoughts" and reflect on their music tastes.

The project is being delivered on time, with the core features like secure profile creation and the "thoughts" system implemented. However, features like Spotify integration, user-entered statistics, and listening habit tracking are still in development and not yet fully functional. While the platform is progressing toward full feature completion, these remaining features are key to the overall project's success and will be addressed as the development continues.

Background & Related Work

The Evolution of Music Criticism
Music criticism has a long and storied history, dating back to the 18th century, when music reviews were published in newspapers and journals to inform the public about new compositions and performances. Early criticism focused primarily on classical music and aimed to provide an intellectual and technical analysis of works. Over time, however, the scope of music criticism expanded as popular music emerged in the 20th century, particularly with the advent of rock 'n' roll in the 1950s. Publications like *Rolling Stone* and *NME* pioneered the shift from formal analysis to cultural commentary, often blending opinions with biographical and societal context.

In the digital age, music criticism has largely transitioned to online platforms, where the ease of publishing and the accessibility of music have created new avenues for engagement. Websites like *Pitchfork*, *Album of the Year (AOTY)*, and *Rate Your Music (RYM)* have come to dominate the landscape of music review and community engagement. However, while these platforms have democratized the review process by allowing anyone to contribute, they have also created new challenges in how music is perceived and consumed. These sites often focus heavily on numeric ratings, which can shape the listener's perception before they even engage with a piece of music.

Numeric Ratings and Their Impact
The use of numeric ratings in music reviews is an established practice, but it has sparked considerable debate. Numeric scores can provide a quick reference, allowing users to easily compare albums. However, they also present challenges. Research has shown that numeric scores can bias a listener's perception of music, causing them to dismiss albums with lower ratings without fully experiencing them. This can lead to a phenomenon known as "prejudgment," where users form an opinion based on the score rather than the music itself (Smith, 2023). For instance, *Pitchfork*'s iconic 10.0 rating has been lauded and criticized for its ability to make or break an album's reception. A high score often elevates an album's status, but it can also create pressure to conform to critical expectations.

These numeric systems, while helpful to some, can be alienating to casual listeners who may feel overwhelmed by technical jargon or discouraged by a low score. Furthermore, they place an emphasis on external validation, where the music's worth is determined by an aggregate opinion rather than a personal, individual experience.

The Challenge of Personalized Music Engagement
Music streaming platforms such as *Spotify* and *Apple Music* have revolutionized how we consume music. With millions of tracks available at the tap of a button, these platforms have made music more accessible than ever. However, despite their benefits, these services have also contributed to a shift in how users engage with music. One of the primary features of these platforms is their algorithmic recommendation systems, which analyze users' listening history to suggest new content. While these algorithms are designed to enhance the user experience by offering personalized suggestions, they can also create a phenomenon known as the "filter bubble" (Pariser, 2011). A filter bubble is a situation in which users are exposed to content that only reinforces their existing preferences, limiting their exposure to new or diverse music.

The Filter Bubble and Algorithmic Limitation
Filter bubbles are particularly problematic in the context of music recommendation systems. Studies show that recommendation algorithms are often biased toward popular or highly rated content, leaving less-known artists and genres underrepresented (Salganik et al., 2006). This can result in a narrow, repetitive listening experience where users are trapped in a cycle of familiar content, unable to discover new music that might align with their evolving tastes. Moreover, algorithms are typically optimized to reinforce previous listening habits, further exacerbating the problem.

Some research suggests that expert users—those who are more familiar with a platform's tools—are better equipped to navigate these limitations by seeking out more obscure content or intentionally deviating from algorithmic recommendations (Villermet et al., 2021). However, this requires a level of familiarity and intention that casual users may not possess. For many, algorithms are the primary means of discovery, which means their music exposure is shaped not by personal exploration, but by the confines of an automated system.

Pitchfork and Traditional Music Review Sites
Pitchfork is perhaps the most influential music review site, consistently shaping trends and taste in modern music. While it has gained a reputation for its critical approach and highly analytical reviews, it also exemplifies many of the pitfalls of traditional music criticism. A major issue with *Pitchfork* and similar outlets is the heavy reliance on numeric ratings. These ratings can often overshadow the actual content of the review, creating a situation where the number becomes the focal point rather than the music itself. This results in a situation where casual listeners may be discouraged from engaging with an album simply because it received a lower score, without considering the review's nuanced analysis.

While *Pitchfork* remains an authority in the industry, it is also frequently criticized for its elitism and gatekeeping tendencies. Reviews can be difficult to understand for those not versed in music theory or the language of the industry, which alienates the average listener and contributes to a sense of exclusivity.

Community-Driven Review Platforms: AOTY and RYM
Community-driven sites like *Album of the Year (AOTY)* and *Rate Your Music (RYM)* attempt to democratize music criticism by allowing users to submit their own reviews and ratings. While these platforms offer a more inclusive space for music lovers to share their opinions, they still rely on numeric scores, which can influence perceptions before a listener even engages with the music. Additionally, these platforms often display aggregate scores, which may sway the opinions of users and contribute to the same biases found in more traditional review systems.

RYM also presents reviews in a dense, forum-style format, which can be overwhelming to users who may not want to sift through endless opinions before forming their own. This creates an environment where the sheer volume of opinions can dilute the personal connection with music, making it harder for users to truly engage with the music on their own terms.

Music Streaming Platforms and Algorithmic Discovery
Spotify and *Apple Music* have become the dominant platforms in music streaming, with over 350 million active users on Spotify (Curry, 2024). While they offer vast libraries of music and personalized playlists, these platforms have been criticized for their reliance on recommendation algorithms that often create filter bubbles. These algorithms recommend music based on users' past listening habits, but this can limit exposure to new and diverse content. The result is a highly personalized experience that, paradoxically, can reduce the overall diversity of musical discovery.

How PlayBack Differs
Unlike traditional review sites and streaming platforms, PlayBack is designed to give users a personal, uninfluenced space to engage with music. By removing numeric ratings and aggregate scores, PlayBack allows users to form their own opinions without external validation or biases. Rather than relying on algorithms, PlayBack fosters an organic connection with music, allowing users to reflect on their experiences and track their musical journeys over time.

What sets PlayBack apart from these existing platforms is its commitment to privacy and autonomy. Unlike *AOTY* and *RYM*, which prioritize visibility through numeric ratings and public reviews, PlayBack ensures that all reviews remain private until the user decides to share them. This creates a safer space for individuals to explore music without feeling pressured by the opinions of others. Additionally, the absence of comment sections and recommendations based on others' ratings further reduces the chance of users feeling influenced by external opinions.

The current landscape of music review and discovery platforms is dominated by systems that emphasize numeric ratings and algorithmic recommendations, both of which can limit personal engagement with music. PlayBack seeks to challenge this norm by offering a platform that values personal reflection, autonomous discovery, and user privacy. By removing external pressures and focusing on the individual's experience, PlayBack provides a unique alternative to traditional music review sites and streaming services.

Purpose of Report

The remainder of this report outlines the full development cycle of the PlayBack system, beginning with updates to the original project plan and requirements models. This includes revised project estimates, presented in a comparative table alongside commentary on differences between initial, design-phase, and actual values. Updated risk analyses and scheduling diagrams are also included, along with fully dressed use case diagrams and a review of how non-functional requirements were addressed. All diagrams and models are accompanied by supporting text that explains their relevance and the changes made throughout the project lifecycle.

Following this, the report presents updated design models that reflect the final implementation. Design diagrams are revisited and refined to account for changes made during development, with detailed explanations of design decisions and system architecture adjustments.

The core of the report focuses on implementation and testing. It begins with an overview of the technologies used—React for the frontend and Flask for the backend—and describes the major features that were successfully implemented, as well as those that were not, with justifications for any omissions. The user interface design is outlined in detail, including the structure of menus and submenus and which options were completed. The test plan is described thoroughly, covering the testing methodology used, individual test cases, expected and actual results, and identification of who performed the testing. Any failed tests and subsequent corrections are noted, alongside a list of future test cases. The report also discusses the implementation and testing process more broadly, including roles and responsibilities within the team, major challenges encountered, and how they were addressed.

Additional sections reflect on how non-functional requirements such as usability, security, and performance were handled, including how each was validated with concrete results. The report concludes with a reflection on the lessons learned throughout the project, offering insights into what could be done differently, how this experience will influence future projects, and key takeaways from the development of PlayBack.

Finally, a user manual is provided to guide end users through the system's features and functionality, and a bibliography lists all resources and references used during the course of the project.

Project Plan & Requirements Models

Resources

Because of the web-based nature of PlayBack, there are a few categories of tools whose use is guaranteed. These include a backend server, a frontend server, hosting, and a database. Some other tools that are more specific to the "music forum" nature of PlayBack are the Music API for pulling music data, and a user authentication library to handle the processes involving users. In the figure below (Figure 1) a description of the tool, its difficulty to use, and specific versions of the tool provides an overview of what the team will be using for PlayBack.

- Music API
  - Access to API with music data for queries
  - Free to access and limited by number of searches made in 30 second window
  - Spotify API
- Database
  - Place to store user information and post information
  - Free and easy to create/ access with no limitations
  - MySQL
- Backend
  - Language and framework used to handle database connections and the logic that controls inner workings of website
  - Free and easy to use without limitations
  - Python, Flask framework
- Frontend
  - Language and library that will control what information is displayed and how it appears.
  - Free and easy to use without limitations
  - JavaScript, React
- Authentication
  - Sessions and database checking
  - Free and easy to use without limitations
  - There are many packages involved; most involve the use of Node.js
- Server hosting
  - Server to keep website running when not local hosted
  - Through the school
- Graphing
  - Software that allows one to visually plot relations between project points which is useful for planning
  - Free and easy to use without limitations
  - Web app called drawio.com

Figure 1: List of Resources

Workflow

In the figure below (Figure 2) a basic workflow is established. This workflow shows which processes are dependent on one another and consequently shows the things that can be completed synchronously. This will be especially useful when estimating the time till completion for PlayBack.

- Design
  - Database schema creation
  - Website file directory flow and page layout mapping
  - Deciding which resources to use for each section of development
    - ♣ Finalization of app scope (what features to implement)
- Front-end
  - Setting up user authentication method for sign in
    - ♣ Setting up connection between front-end and back-end

    - Reviewing appearance of website and usability
        - UI design and implementation
- Back-end
    - Testing back-end functionality
        - User creation and user data storage
            - Setting up connection with database and database implementation
        - API querying and results formatting
        - Storing comments, reviews, and other posts and returning them
            - Setting up connection with database and database implementation

Figure 2: Work Breakdown List

Estimated Cost & Resources

Line of Code

The lines of code estimation in Table 1 took data from various, admittedly unreputable sources to guess how many lines of code it would take to complete each task listed in Table 1.

Database Schema Creation	0
Website file directory flow and page layout	0
Deciding which resources to use	0
Finalization of PlayBack scope	0
Setting up user authentication (Javascript, Python)	~300
Setting up front-end back-end connection (Javascript, Python)	~300
Reviewing appearance of website	0
UI design and implementation (Javascript, HTML, CSS, Python)	~2000
Testing back-end functionality (Python)	~100
User creation and data storage (Python, SQL)	~700
Setting up connection between back end and database implementation (Python, SQL)	~500
API querying and results formatting (Python)	~700
Storing post data and returning data (Python, SQL, Javascript)	~100
Total	4800

Table 1: Lines of Code Estimation Table

Lines per day junior developer: 100

4800/100 = 48 developer days

Junior developer salary: $82,913 (ZipRecruiter)

Development cost: $11,055

Function Points

Function point estimations take guesses on how difficult it would be to complete a few preordained categories to calculate a find function point count. In Tabel 2.1, one can see the major categories involved in most software projects that take the most time to finish, and in Table 2.2 the categories that a simpler to complete.

Information Domain Value	Optimistic	Likely	Pessimistic	Est. Count	Weight	FP count
External Inputs	4	5	6	5	4	20
External Outputs	1	1	2	1	1	1
External Inquiries	3	4	5	4	4	16
Internal Logic Files	2	2	3	2	2	4
External Interface Files	0	1	1	1	1	1
Count Total						42

Table 2.1: Function Point Computations

Requires Backup/Recovery?	1
Data Communications Required?	2
Distributed Process Functions?	1
Performance Critical?	0
Run on Existing Heavily Utilized Environment?	1
Requires Online Data Entry?	4
Multiple Screen for Input?	2
Master Fields Updated Online?	1
Inputs, Outputs , Inquiries of files complex?	0
Internal Processing Complex?	0
Code Designed for Reuse?	0
Conversion and Installation Included?	0
Multiple Installation in Different Orgs?	0
Must Facilitate Change Case of Use by User?	1
Total	13

Table 2.2: Weights for 14 General Characteristics of Project Table

Function point calculations: 42 x (.65 + .13) = 33 FP

Function points per month: 10

33/10 = 3.3 developer months

3.3 / 12 = .275

82,913 * .4225 = $22,801

Project Schedule

When choosing an appropriate and realistic schedule for this project, it was important that a conservative end date was selected. This way, even if the schedule was delayed by multiple days the project would be ready by 5/5/25, the presentation date. Both Figure 5 and Figure 6 illustrate the plan based on these principles and largely show the same thing, but in different forms. The first points in our schedule largely concern the planning of resources and functionality of PlayBack. Once the resources being used are confirmed work will begin on setting up the database, API testing and results formatting, and one of the two largest sections of the project; UI design and implementation. Once the database has been set up work can begin on the other largest section of the project, user creation and data storage. Now that the database and the user objects have been created, the next step will be to set up a user authentication system and if the API querying is done by this point work can begin on storing posts made by users in the database.

By the time the UI has been completed work can likely begin on the facilitation of a connection between the backend and frontend. A couple of days will need to be spent on streamlining the UI following its initial completion. At this point, the project will be in a presentable state signaling its completion.

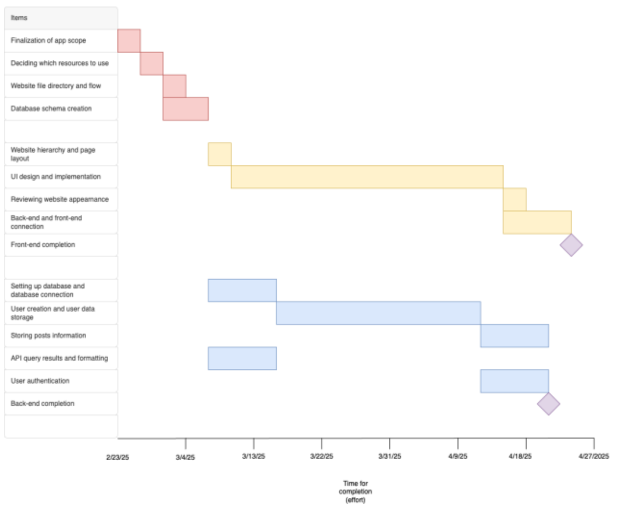


Figure 5: Gantt Chart for Project Schedule

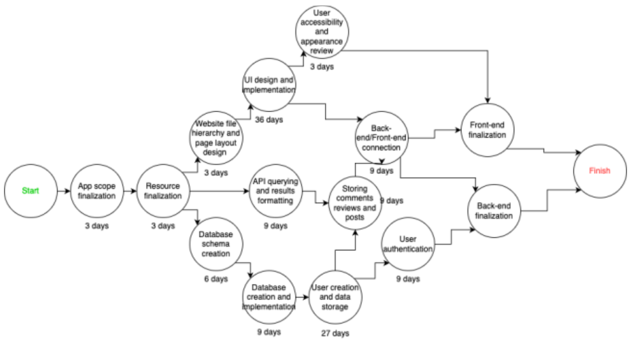
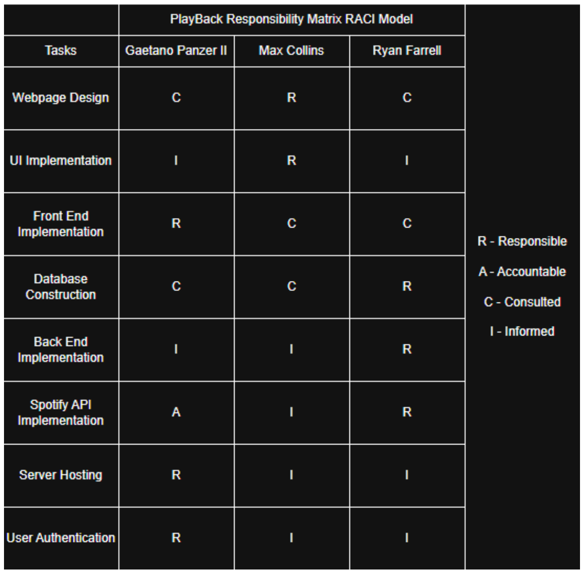

Start
App scope finalization
3 days
Resource finalization
3 days
Website file hierarchy and page layout design
3 days
UI design and implementation
36 days
User accessibility and appearance review
3 days
API querying and results formatting
9 days
Database schema creation
6 days
Database creation and implementation
9 days
User creation and data storage
27 days
Storing comments reviews and posts
9 days
Back-end/Front-end connection
9 days
User authentication
9 days
Front-end finalization
Back-end finalization
Finish

Figure 6: PERT Chart for Project Schedule

Responsibility Matrix

To break up work on the software design process and make it easier for each team member, responsibilities will be divided amongst each team member. Table 3 below represents each team member's responsibilities in a matrix. Due to the size of the project, responsibility in creating the front-end and the back-end is shared between group members, with the following section and Table 4 going more in depth into the reasoning behind this decision.



	PlayBack Responsibility Matrix RACI Model			
Tasks	Gaetano Panzer II	Max Collins	Ryan Farrell	
Webpage Design	C	R	C	
UI Implementation	I	R	I	
Front End Implementation	R	C	C	R - Responsible
Database Construction	C	C	R	A - Accountable
Back End Implementation	I	I	R	C - Consulted
Spotify API Implementation	A	I	R	I - Informed
Server Hosting	R	I	I	
User Authentication	R	I	I	

Table 3: Responsibility Matrix

Design Process

Ryan

- Paper contributions, (1/3 of whole)
- Backend implementation

Gaetano

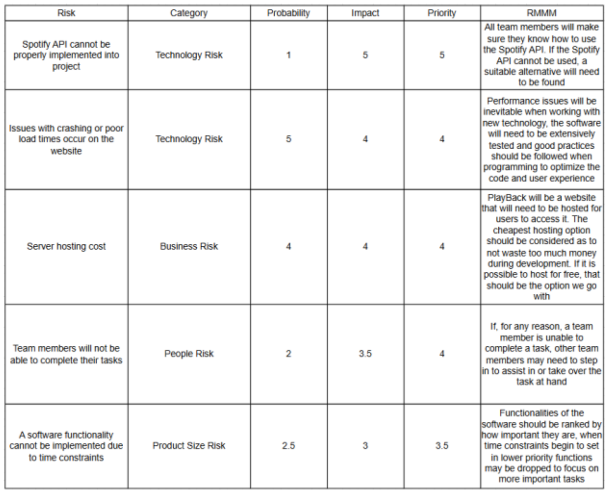
- Paper contributions (1/3 of whole)
- Front and Backend implementation

Max

- Paper contributions (1/3 of whole)
- Front end UI designing and implementation

Risk Management

Table 4 outlines potential risks that may arise during the development of PlayBack. Risks are ranked from highest to lowest impact/priority and are ranked on a scale from 1-5. Each risk has been given a general plan on how the risk should be mitigated and managed should it arise.



Risk	Category	Probability	Impact	Priority	RMMM
Spotify API cannot be properly implemented into project	Technology Risk	1	5	5	All team members will make sure they know how to use the Spotify API. If the Spotify API cannot be used, a suitable alternative will need to be found
Issues with crashing or poor load times occur on the website	Technology Risk	5	4	4	Performance issues will be inevitable when working with new technology, the software will need to be extensively tested and good practices should be followed when programming to optimize the code and user experience
Server hosting cost	Business Risk	4	4	4	PlayBack will be a website that will need to be hosted for users to access it. The cheapest hosting option should be considered as to not waste too much money during development. If it is possible to host for free, that should be the option we go with
Team members will not be able to complete their tasks	People Risk	2	3.5	4	If, for any reason, a team member is unable to complete a task, other team members may need to step in to assist in or take over the task at hand
A software functionality cannot be implemented due to time constraints	Product Size Risk	2.5	3	3.5	Functionalities of the software should be ranked by how important they are, when time constraints begin to set in lower priority functions may be dropped to focus on more important tasks

Table 4: Risk Matrix

Issues & Constraints

Some key components to making the software functional, as laid out in previous sections of the Software Project Plan, are new to members of the team and will induce a learning curve as development of PlayBack moves forward. As shown in Table 4, the risk of not learning the Spotify API is the most critical risk; however, extensive documentation and previous examples of work done with components exist and will be studied to the best of all team members' abilities to create a functional application.

Alongside the challenges of learning to work with new APIs, more general issues may arise during the development of PlayBack, such as errors and bugs in written code, as well as problems in optimizing code to keep the user experience as smooth as possible. These problems are more likely than the previous risk but will be addressed as they arise. Real-world constraints also present challenges, especially with the departure of one team member, which has led to a redistribution of responsibilities. The loss of a team member has strained resources and slowed the pace of development, leading us to reassess our scope.

As a result, the scope of the project has been adjusted to ensure we can complete a viable version of PlayBack by the end of the semester. Some less vital features, such as advanced profile customization and additional integrations, may need to be cut or deferred until future development phases. This adjustment allows us to focus on the core functionalities necessary for PlayBack to be functional and user-friendly, such as secure profile creation, music playback integration through Spotify's API, and the manual listening statistic tracking system.

Additionally, the team faces scheduling conflicts, as members are balancing other class projects and jobs outside of school. These time constraints have made adhering to the original ideal schedule more challenging. However, with improved teamwork, communication, and strategic planning, we are confident that these risks can be mitigated, allowing us to deliver a functional version of PlayBack by the end of the semester.

Project Control

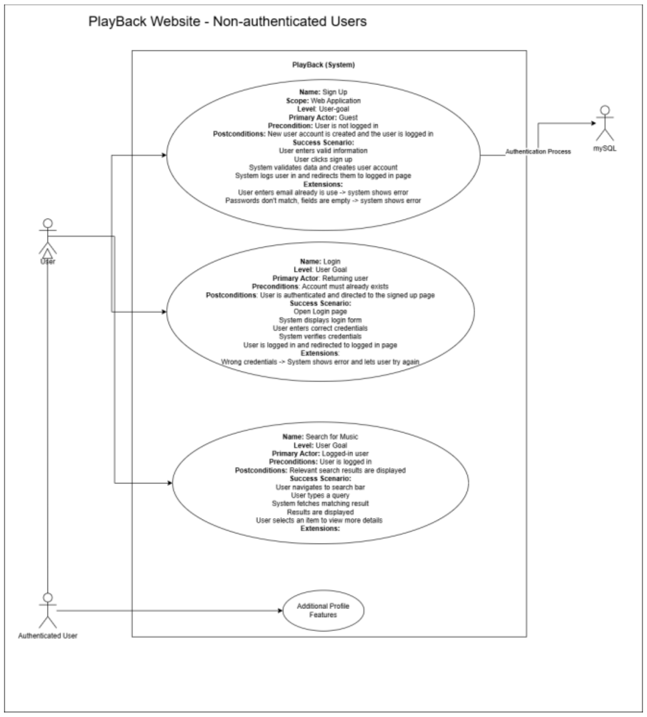
Below is a list of methods that are being used to monitor and control the project during development

- GitHub
  - GitHub is the main form of version control being used by group members during development
- Discord
  - Discord is being used to post progress updates and ask other group members questions about subjects relating to PlayBack.
  - When members of the team cannot meet in person, they are using Discord's voice chat features to meet online, discuss the project requirements, and work together on development
- Microsoft Word
  - Microsoft Word is being used to create text documents that contain important information regarding the development of PlayBack, including ideas for the scope of the software, the project's requirements engineering report, and the software design report.

It is the responsibility of each team member to use these methods to routinely check in with each other and inform group members of any significant changes or complications.

The Requirements/Analysis Models section outlines the key functionalities and interactions within PlayBack through a series of diagrams. These models illustrate how users engage with the platform, covering both high-level processes and specific system behaviors. Figure 7 depicts actions available to users without accounts, such as browsing public content and viewing general music information, while Figure 8 details interactions for logged-in users, including creating profiles and posting thoughts. Together, these diagrams provide a clear visual representation of PlayBack's core features, helping define the system's functional requirements and user experience.

Requirements/Analysis Models



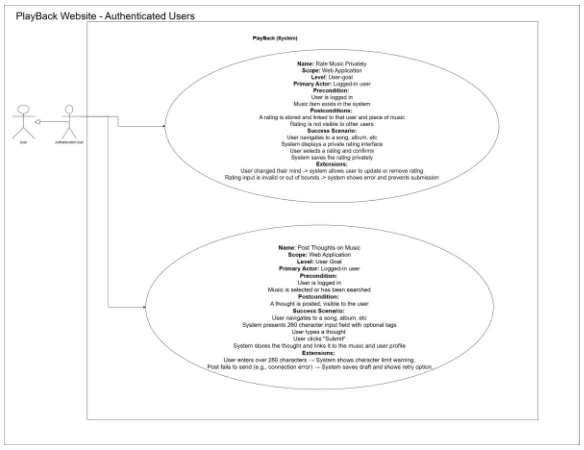
PlayBack Website - Non-authenticated Users
PlayBack (System)
User
Authenticated User
Authentication Process
mySQL
Additional Profile Features

Name: Sign Up
Scope: Web Application
Level: User-goal
Primary Actor: Guest
Precondition: User is not logged in
Postconditions: New user account is created and the user is logged in
Success Scenario:
User enters valid information
User clicks sign up
System validates data and creates user account
System logs user in and redirects them to logged in page
Extensions:
User enters email already is use -> system shows error
Passwords don't match, fields are empty -> system shows error

Name: Login
Level: User Goal
Primary Actor: Returning user
Preconditions: Account must already exists
Postconditions: User is authenticated and directed to the signed up page
Success Scenario:
Open Login page
System displays login form
User enters correct credentials
System verifies credentials
User is logged in and redirected to logged in page
Extensions:
Wrong credentials -> System shows error and lets user try again

Name: Search for Music
Level: User Goal
Primary Actor: Logged-in user
Preconditions: User is logged in
Postconditions: Relevant search results are displayed
Success Scenario:
User navigates to search bar
User types a query
System fetches matching result
Results are displayed
User selects an item to view more details
Extensions:

Figure 7: Use Case Diagram for Non-authenticated Users



Name: Rate Music Privately
Scope: Web Application
Level: User-goal
Primary Actor: Logged-in user
Precondition:
User is logged in
Music item exists in the system
Postconditions:
A rating is stored and linked to that user and piece of music
Rating is not visible to other users
Success Scenario:
User navigates to a song, album, etc
System displays a private rating interface
User selects a rating and confirms
System saves the rating privately
Extensions:
User changed their mind -> system allows user to update or remove rating
Rating input is invalid or out of bounds -> system shows error and prevents submission

Name: Post Thoughts on Music
Scope: Web Application
Level: User Goal
Primary Actor: Logged-in user
Precondition:
User is logged in
Music is selected or has been searched
Postcondition:
A thought is posted, visible to the user
Success Scenario:
User navigates to a song, album, etc
System presents 280 character input field with optional tags
User types a thought
User clicks "Submit"
System stores the thought and links it to the music and user profile
Extensions:
User enters over 280 characters → System shows character limit warning.
Post fails to send (e.g., connection error) → System saves draft and shows retry option.

Figure 8: Use Case Diagram for Authenticated Users

The core features described above are further illustrated through Figure 9, which visually represents how users interact with the platform. The diagram maps out the flow of actions for key processes, such as user registration and searching for music. It highlights the decision points and sequences involved, providing a clear depiction of the system's behavior.

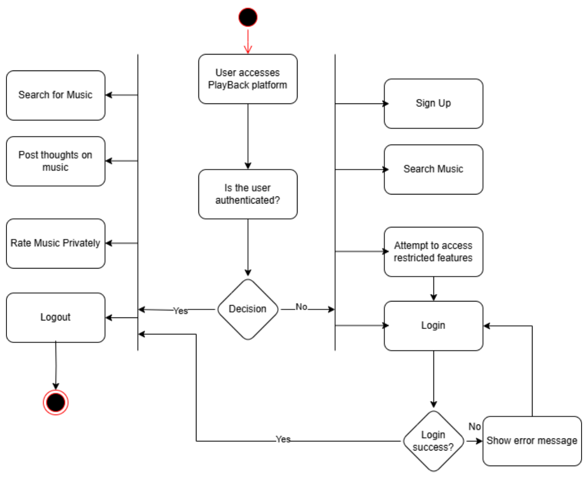


Figure 9: Activity Diagram

The core features are further detailed in Figure 10, which defines the structure and relationships between the main components of the PlayBack platform. The model illustrates how data is organized and managed, showcasing the attributes, methods, and associations of key classes such as User, Music, Thoughts and Vibes.

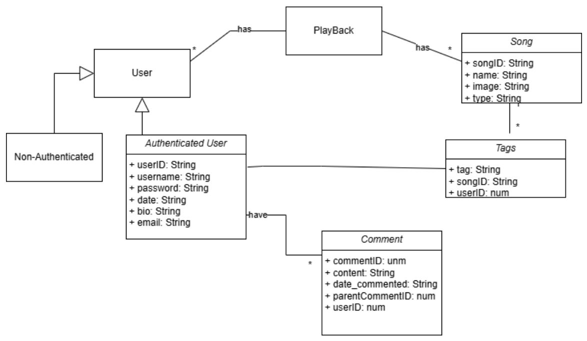


Figure 10: Class Model

Major Software Functions

The following outlines the core features and functionalities of PlayBack. Each section details the platform's primary functions, the actors involved, and the specific use cases that define how users will interact with the system. From account management and music playback to private ratings, and direct messaging, these use cases represent the key interactions that form the foundation of the platform's user experience.

1. User Registration and Login

- **Function**: Allows users to sign up, log in, and manage their profiles.
- **Actors**: User, Authenticated User
- **Use Cases**:
  - **Sign Up**: Links to mySQL for account creation.
  - **Login**: Links to mySQL for authentication.

2. Music Search

- **Function**: Allows users to search for music and other authenticated users.
- **Actors**: User, Authenticated User
- **Use Cases**:
  - **Search for Music**: Search for songs, albums, and artists.
  - **Search for Authenticated Users**: Allows users to find other users on the platform.

3. Private Music Vibe

- **Function**: Enables authenticated users to rate music privately.
- **Actors**: Authenticated User
- **Use Cases**:
  - **Rate Music Privately**: Rate music in a way that's only visible to the user (not shared with others).

4. Posting

- **Function:** Allows authenticated users to post their thoughts on music
- **Actors:** Authenticated Users
- **Use Cases:**
  - **Post Thoughts on Music:** Users can post thoughts on music and decide if they want it to be available to the public, friends, or private.

The core features described above are further illustrated through Figure 9, which visually represents how users interact with the platform. The diagram maps out the flow of actions for key processes, such as user registration and posting . It highlights the decision points and sequences involved, providing a clear depiction of the system's behavior.

Data Dictionary

The data dictionary in Figure D.1 shows all the data that will be tracked in the database and excludes any session-based data. Items marked "FK" are to show the relationship between two tables of data, and items marked "optional" do not need to contain any information, unlike the rest of the data present.

Song(Media) Data

- Song ID – Song identifier
- Name – The name of the song(media)
- Image – URL of image representing media

Comment Data

- Comment ID – Comment Identifier
- Content – Contents of comment
- Date Posted – Time of comment creation
- Parrent Comment ID (FK)(Optional) - Points to the comment being commented on
- User ID (FK) - Points to the user who created the comment

User Data

- User ID – User Identifier
- Username – Unique user tag, used for user verification and identification
- Password – Used for user verification
- Date Joined – Time of user creation
- Bio – Short description of user
- Email – Email address of user

Figure D.1: Data Dictionary

Non-functional Requirements

User Experience
The platform must offer a seamless, intuitive user experience to ensure that users can easily navigate between key features such as posting thoughts. The design must prioritize clarity and simplicity so that users can quickly understand how to interact with the platform.

- **How it's Achieved:**
  The user interface (UI) was designed to be clean and simple, featuring clear call-to-action buttons for key interactions (e.g., posting thoughts). This ensures that users can access and utilize features without confusion. Consistent navigation across pages also reinforces ease of use.

Security
Ensuring the security of user data is a critical non-functional requirement. The platform must protect personal user information, including login credentials and posts, from unauthorized access.

- **How it's Achieved:**
  To meet this requirement, user authentication was implemented using hashed passwords (via MySQL), ensuring that credentials are not stored in plain text. All sensitive data is transmitted securely over HTTPS, and future versions of the platform will introduce more granular privacy controls, such as post visibility settings (private/friends-only). Furthermore, only authenticated users can interact with sensitive features, ensuring that content remains protected.

Performance
The platform must offer fast load times and responsiveness, even when multiple users are interacting with the system simultaneously. As the platform grows, performance must scale without compromising user experience.

- **How it's Achieved:**
  To meet performance requirements, the platform uses lazy loading to prioritize essential components (such as the navigation bar and music player) while deferring less important elements. This ensures that users can access core features quickly, even on initial page load. Database queries have been optimized with indexing to ensure fast data retrieval, particularly for user posts and listening habits.

Design Models

General Design Constraints

For our platform, the hardware requirements are modest. It will run smoothly on devices with at least 4 GB of RAM and a dual-core processor, covering most modern desktops and laptops. The platform will be compatible with Chrome, Firefox, Edge, and Safari on both Windows and macOS, ensuring broad accessibility. Performance expectations are moderate, with music data loading in under 3 seconds on a standard connection.

Our platform will use a Flask + MySQL backend with a React + CSS frontend for simplicity and efficiency. Authentication will rely on sessions and cookies, avoiding the need for Firebase or JWT tokens. Spotify's API will be integrated for music data retrieval and playback. The tech stack prioritizes ease of development and deployment, minimizing dependencies for a lightweight, manageable project.

Architecturally, the platform will follow a layered structure, organizing the codebase into separate layers for database interactions, core business logic, and presentation. This modular approach keeps the project clean and scalable without the complexity of a full MVC framework. While a monolithic structure could work, the layered design offers better organization and future flexibility.

For security, basic session management and input validation will protect against common vulnerabilities like SQL injection and cross-site scripting. Privacy measures will be straightforward but effective, with user thoughts and listening data remaining private by default. The platform will also comply with Spotify's API terms for proper handling of external data.

Architectural Design

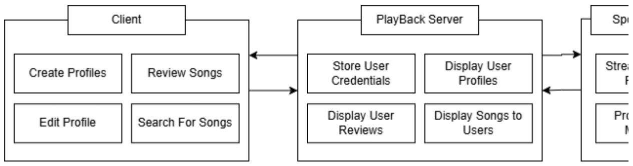


Figure A.1: Layered architecture model

Description

The model shown above represents the architecture and subsystems of PlayBack using the Client-Server architectural style. Users, or clients, using the website's interface would be able to create and edit profiles, search for songs, and leave reviews. The website's server would store the user data of all the website's users, as well as their reviews. The website would then be able to display profiles for public or private viewing, display pages for songs, and display the reviews for the songs. Using the Spotify API, PlayBack would connect to Spotify servers to stream songs for users, as well as obtain the metadata for the songs.

Strengths

The Client-Server model is scalable, with the host being able to add additional servers to increase the load that the server can handle. Though we won't need additional servers for this project, this is one of the greatest strengths of this model. The Client-Server model takes strain off the client's computer by performing the complicated processes and critical functions on the server, allowing for clients to access the website with less sophisticated machines. The Client-Server model is a very robust framework which has been used for many applications, allowing for nearly any system to be run efficiently.

Weaknesses

Depending on the size and complexity of the server being used to host an application, the cost of the application could be high. A server that is connecting with multiple clients simultaneously could become strained and/or bottlenecked, leading to slow load times and suboptimal performance. This can be mitigated by upgrading existing servers or adding new servers, but both of these fixes would cost money to implement. If the server that is hosting the application goes down or fails, the application will no longer function as intended, and fixing the server becomes the number one priority.

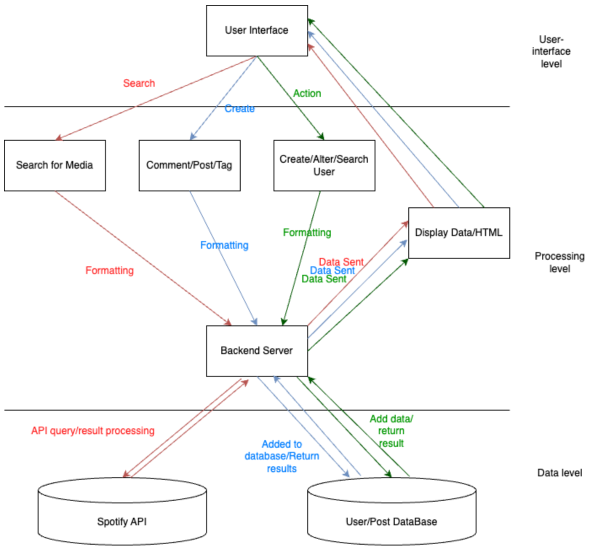


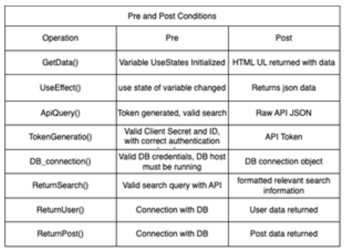
Figure A.2: Layered architecture model

There are three layers in PlayBack's layered architecture. In the UI layer, users will be searching or creating/altering information on the application layer, which doesn't alter the database in any way. In Figure A.2, you can see the three main actions performed by the user in the user interface. They can search for media, comment or post, and change their own user information or search other users. The data they input is then formatted by the front-end components and sent to the backend server, which once again processes the data. This data can be sent to either the Spotify API to return music data or to the Database, to store user/ post data. Once data is successfully searched/added, the results are returned to the front end and displayed by the components, which the user is able to access. Using a layered architecture like this is great for security, as the front end doesn’t need to generate the API keys, which is sensitive information. Also, having a python backend makes it easier to compartmentalize and organize code. However, an extra layer increases the latency marginally, which means the user will get a slower experience.

Subsystems

Music Search

The Music search feature starts by allowing input from the user on the application level, which then takes that data and formats it so it can be sent to the backend server. Once the backend server receives the data, it formats it so it can be used to query the Spotify API. If the API returns data, that data is once again formatted by the backend server and sent back to the component to be displayed. This is done frequently, as any time the user input changes this entire process is repeated.



Pre and Post Conditions		
Operation	Pre	Post
GetData()	Variable UseStates Initialized	HTML UL returned with data
UseEffect()	use state of variable changed	Returns json data
ApiQuery()	Token generated, valid search	Raw API JSON
TokenGeneratio()	Valid Client Secret and ID, with correct authentication	API Token
DB_connection()	Valid DB credentials, DB host must be running	DB connection object
ReturnSearch()	Valid search query with API	formatted relevant search information
ReturnUser()	Connection with DB	User data returned
ReturnPost()	Connection with DB	Post data returned

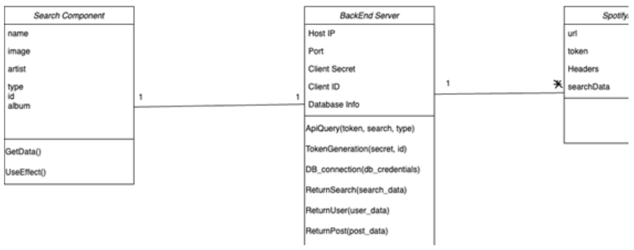


Figure S.1

The class diagram and table in Figure S.1 show the relationship between the three main components when a search for music. The search component wants the properties in its class to be assigned to relevant data about the music being searched. To do this, it uses its methods to ask the backend for this information, which forwards that data to the Spotify API after formatting the data and ensuring token validity. The Spotify API JSON that is returned, is packaged by the “ReturnSearch” function and returned to the component.

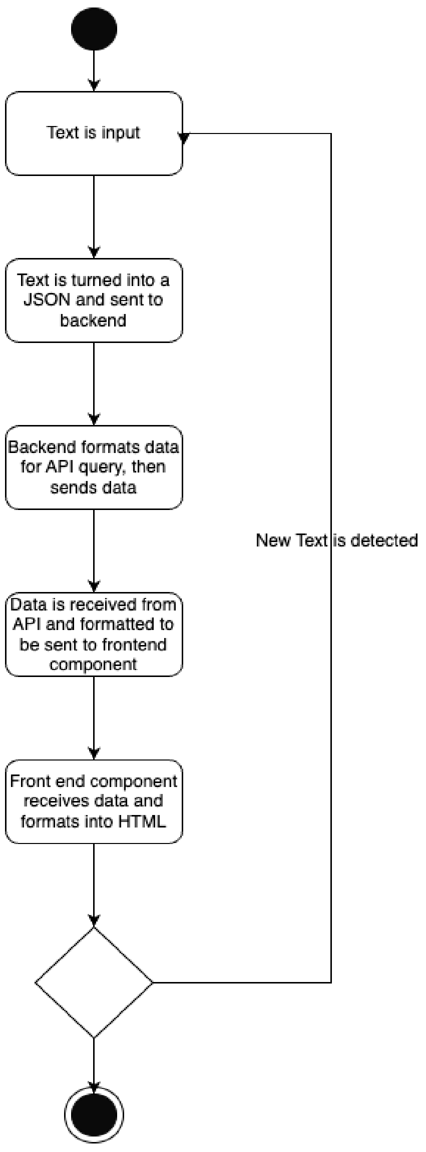

Text is input
Text is turned into a JSON and sent to backend
Backend formats data for API query, then sends data
Data is received from API and formatted to be sent to frontend component
Front end component receives data and formats into HTML
New Text is detected

Figure S.2

In the sequence diagram shown in Figure S.2, the linear system of data transfer across the three layers can be seen. This cycle will continue until the form is submitted.

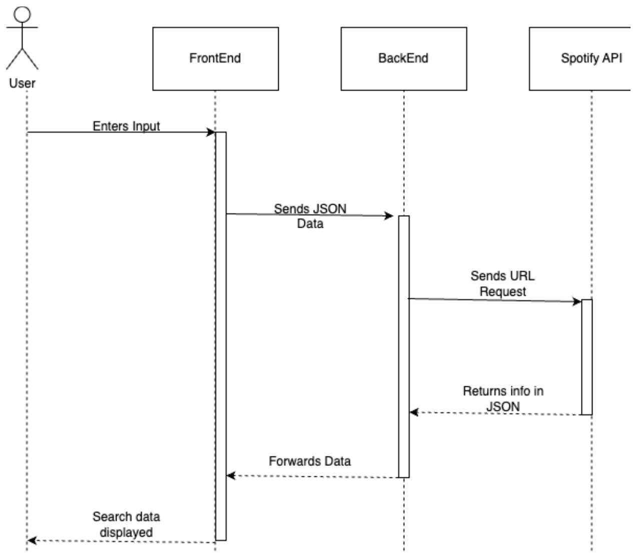


Figure S.3

In Figure S.3, the interactions between the layers can be seen at each step. The data transfer is incredibly simple, with a call being answered by a single result once the bottom layer is reached.

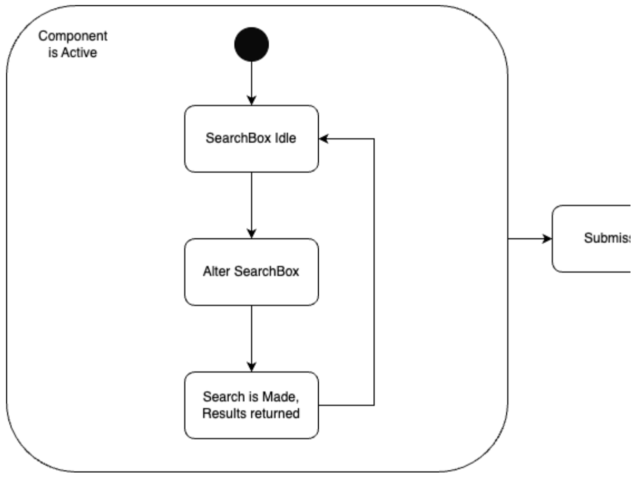


Figure S.4

The cyclical nature of the search component can be seen in S.4, as the component will continue returning and accepting data until submission.

Creating/Altering User Data

Any actions involving user information requires some input from the user on the application level. These inputs are formatted then sent to the backend, which has SQL statements prepared to alter/create for user data depending on the component used. Once the database is successfully updated, the result is shown to the user.

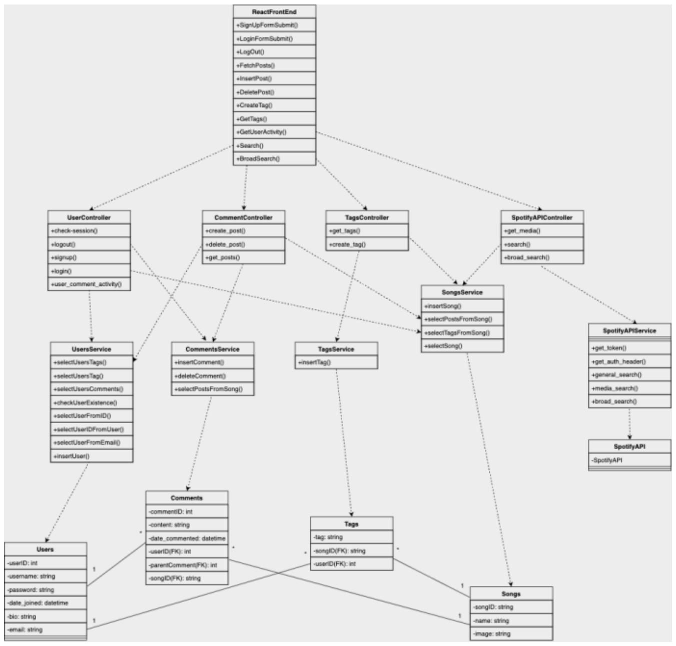
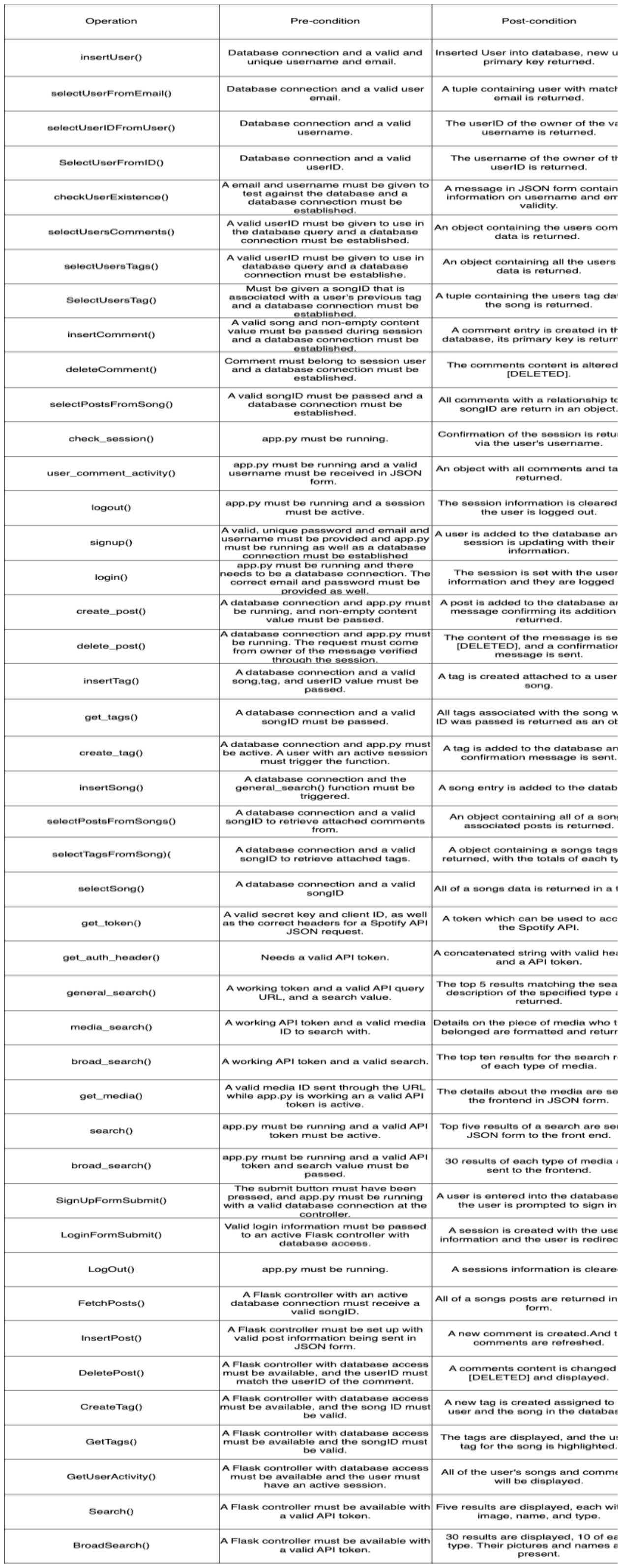


Figure S.5.1

The design class diagram in Figure S.5 shows how the three layers work together and how their functions facilitate the transfer of data. At the bottom of the diagram, one can see the basic classes that hold the information that populates the website. Above that, you have the service classes which is where most of the computation is done in the back end. Above the service classes you have the controller classes, which is where data from the front end is sent. Controller classes must be able to receive these requests in JSON format and response with JSON as well. And finally, you have the front-end class, which is where data is shown to the user and user input can be taken. Each of these classes uses functions in the lower classes to perform operations, sometimes from two classes synchronously.



Operation	Pre-condition	Post-condition
insertUser()	Database connection and a valid and unique username and email.	Inserted User into database, new u primary key returned.
selectUserFromEmail()	Database connection and a valid user email.	A tuple containing user with match email is returned.
selectUserIDFromUser()	Database connection and a valid username.	The userID of the owner of the va username is returned.
SelectUserFromID()	Database connection and a valid userID.	The username of the owner of th userID is returned.
checkUserExistence()	A email and username must be given to test against the database and a database connection must be established.	A message in JSON form contain information on username and em validity.
selectUsersComments()	A valid userID must be given to use in the database query and a database connection must be established.	An object containing the users com data is returned.
selectUsersTags()	A valid userID must be given to use in database query and a database connection must be establishe.	An object containing all the users data is returned.
SelectUsersTag()	Must be given a songID that is associated with a user's previous tag and a database connection must be established.	A tuple containing the users tag dat the song is returned.
insertComment()	A valid song and non-empty content value must be passed during session and a database connection must be established.	A comment entry is created in th database, its primary key is return
deleteComment()	Comment must belong to session user and a database connection must be established.	The comments content is altered [DELETED].
selectPostsFromSong()	A valid songID must be passed and a database connection must be established.	All comments with a relationship to songID are return in an object.
check_session()	app.py must be running.	Confirmation of the session is retu via the user's username.
user_comment_activity()	app.py must be running and a valid username must be received in JSON form.	An object with all comments and ta returned.
logout()	app.py must be running and a session must be active.	The session information is cleared the user is logged out.
signup()	A valid, unique password and email and username must be provided and app.py must be running as well as a database connection must be established	A user is added to the database an session is updating with their information.
login()	app.py must be running and there needs to be a database connection. The correct email and password must be provided as well.	The session is set with the user information and they are logged
create_post()	A database connection and app.py must be running, and non-empty content value must be passed.	A post is added to the database ar message confirming its addition returned.
delete_post()	A database connection and app.py must be running. The request must come from owner of the message verified through the session.	The content of the message is se [DELETED], and a confirmatior message is sent.
insertTag()	A database connection and a valid song,tag, and userID value must be passed.	A tag is created attached to a user song.
get_tags()	A database connection and a valid songID must be passed.	All tags associated with the song w ID was passed is returned as an ob
create_tag()	A database connection and app.py must be active. A user with an active session must trigger the function.	A tag is added to the database an confirmation message is sent.
insertSong()	A database connection and the general_search() function must be triggered.	A song entry is added to the datab
selectPostsFromSongs()	A database connection and a valid songID to retrieve attached comments from.	An object containing all of a son associated posts is returned.
selectTagsFromSong)(	A database connection and a valid songID to retrieve attached tags.	A object containing a songs tags returned, with the totals of each ty
selectSong()	A database connection and a valid songID	All of a songs data is returned in a t
get_token()	A valid secret key and client ID, as well as the correct headers for a Spotify API JSON request.	A token which can be used to acc the Spotify API.
get_auth_header()	Needs a valid API token.	A concatenated string with valid hea and a API token.
general_search()	A working token and a valid API query URL, and a search value.	The top 5 results matching the sea description of the specified type a returned.
media_search()	A working API token and a valid media ID to search with.	Details on the piece of media who t belonged are formatted and return
broad_search()	A working API token and a valid search.	The top ten results for the search r of each type of media.
get_media()	A valid media ID sent through the URL while app.py is working an a valid API token is active.	The details about the media are se the frontend in JSON form.
search()	app.py must be running and a valid API token must be active.	Top five results of a search are se JSON form to the front end.
broad_search()	app.py must be running and a valid API token and search value must be passed.	30 results of each type of media a sent to the frontend.
SignUpFormSubmit()	The submit button must have been pressed, and app.py must be running with a valid database connection at the controller.	A user is entered into the database the user is prompted to sign in.
LoginFormSubmit()	Valid login information must be passed to an active Flask controller with database access.	A session is created with the use information and the user is redirec
LogOut()	app.py must be running.	A sessions information is cleare
FetchPosts()	A Flask controller with an active database connection must receive a valid songID.	All of a songs posts are returned in form.
InsertPost()	A Flask controller must be set up with valid post information being sent in JSON form.	A new comment is created.And t comments are refreshed.
DeletePost()	A Flask controller with database access must be available, and the userID must match the userID of the comment.	A comments content is changed [DELETED] and displayed.
CreateTag()	A Flask controller with database access must be available, and the song ID must be valid.	A new tag is created assigned to user and the song in the databas
GetTags()	A Flask controller with database access must be available and the songID must be valid.	The tags are displayed, and the us tag for the song is highlighted.
GetUserActivity()	A Flask controller with database access must be available and the user must have an active session.	All of the user's songs and comme will be displayed.
Search()	A Flask controller must be available with a valid API token.	Five results are displayed, each wit image, name, and type.
BroadSearch()	A Flask controller must be available with a valid API token.	30 results are displayed, 10 of ea type. Their pictures and names a present.

Figure S.5.2

In figure S.5.2, the pre and post conditions for each of the operations in the design class diagram are listed. The functions from the upper classes are heavily reliant on functions in the lower classes to execute properly, which one can see by their frequent appearance in the pre-condition requirements of the higher classes' functions.

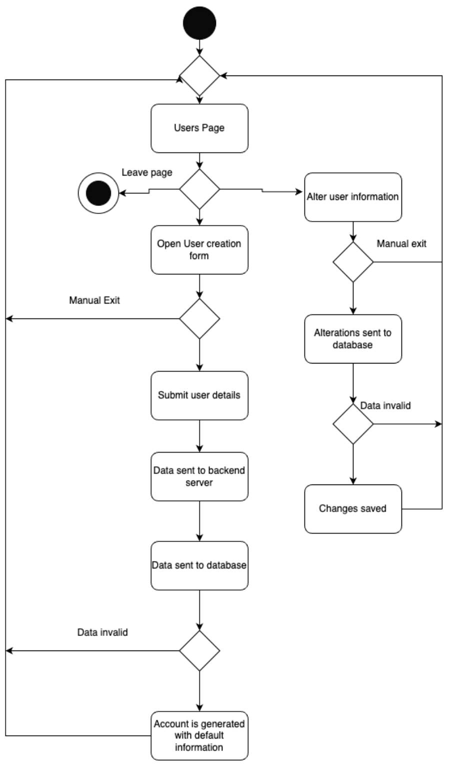


Figure S.6

In Figure S.6, one might be able to understand how the alteration and creation of users might work. Starting at the users page, if the user exists alterations can be made, if it does not, the user can be created. This process does not end until the users page is left.

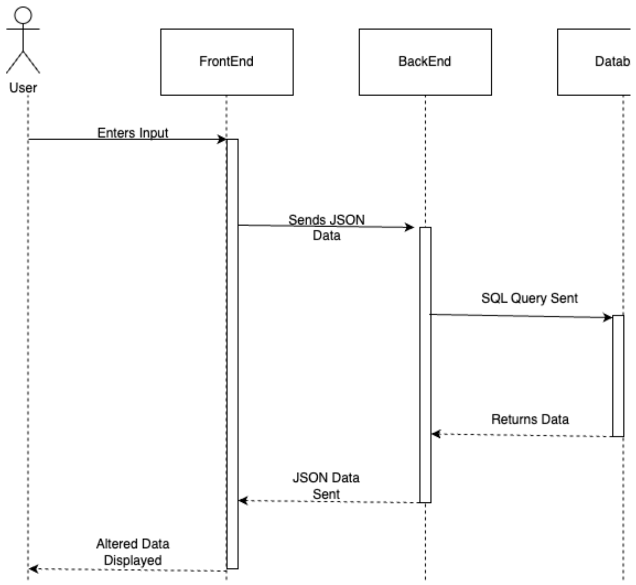


Figure S.7

Figure S.7 shows how through abstraction, the user input travels down the layers to reach the database.

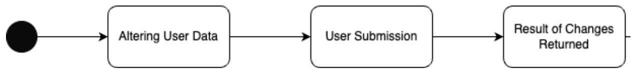


Figure S.8

There are few states for user data alteration, as the process is linear. In Figure S.8 the 3 states visible to the user are shown.

Commenting/Posting

Once a comment/post is submitted by a user, the component used will format the data and send it to the backend server, which has SQL statements prepared to insert/alter/select data from the database. Once the database has been successfully altered, the result is shown to the user.

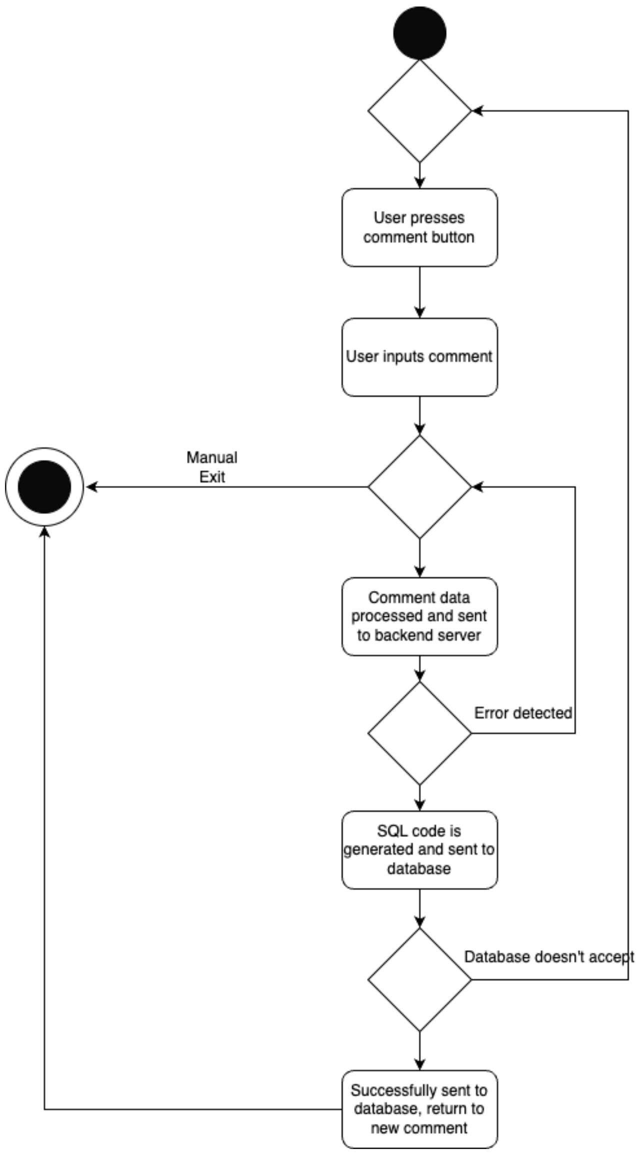

User presses comment button
User inputs comment
Manual Exit
Comment data processed and sent to backend server
Error detected
SQL code is generated and sent to database
Database doesn't accept
Successfully sent to database, return to new comment

Figure S.10

The process of creating a post is linear, the only cycle appearing if an error occurs. In Figure S.10 from its shape alone you can see this linear nature.

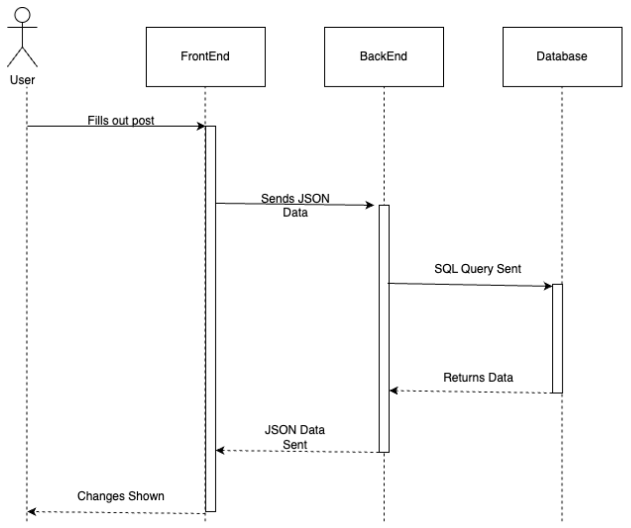


Figure S.11

As is the case with Figure S.3 and Figure S.7, Figure S.11 also involves a process that works its way down the layers to reach the data layer and returns information.

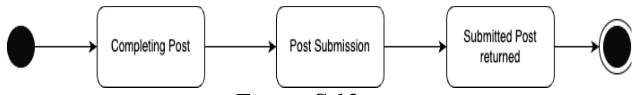


Figure S.12

The 3 states visible to the user as shown in Figure S.12, the process is simple and linear. Identical in nature to Figure S.8.

Tools

GitHub

A platform for version control and collaborative software development, often used to track code changes, store project documentation, and collaborate with team members.

How It Was Used:

- **Version Control**: Used to manage project documentation, especially for requirements and other project artifacts.
- **Collaboration**: We used GitHub to share and collaborate with each other

Draw.io

A web-based tool for creating flowcharts, diagrams, and use case diagrams. It's highly useful for visualizing system architecture, user flows, and interactions between actors and the system.

How It Was Used:

- **Use Case Diagrams**: We used draw.io to create visual representations of user actions, such as "sign up," "stream music," "tag music with vibes," and more.

Zip Recruiter

A website that contains information about salaries based on experience, location, and a wide range of other useful metrics.

How It Was Used

- **Pricing Estimation:** We needed salary data to determine how much it might cost to employ junior developers to complete this project based on time estimations.

Microsoft Word

A widely used word processing software for writing and formatting documents. In the requirements process, it's used for creating and maintaining detailed project documentation.

How It Was Used:

- **Requirements Documentation**: We used Word to write out detailed descriptions of functional and nonfunctional requirements and other project specifications.
- **Documentation Storage**: Word was used to formally document the project’s requirements, making it easy to update, review, and share them with potential stakeholders.

MyBib

A reference management tool that helps create citations for research papers, articles, and other sources. It can automatically generate bibliographies in various citation formats.

How It Was Used:

- **References Management**: Used to organize and manage research materials or references that were relevant for our project requirements.

Implementation

Programming Languages

The development of *PlayBack* utilized a combination of programming languages and frameworks, each selected to meet specific needs of the application. Table 5 outlines the core technologies and their respective roles within the system.

Python (Flask) served as the primary framework, handling server-side logic, routing, and rendering the frontend. This allowed for a streamlined setup where the backend and frontend could be closely integrated.

JavaScript (with React) was used to enhance user experience on the frontend, enabling dynamic UI components such as the post input field and profile displays. While Flask handled the overall page rendering, React supported more interactive elements within specific areas of the site.

HTML/CSS were used throughout the application to create structured layouts and maintain consistent styling across different pages. These technologies ensured the interface was accessible and responsive across devices.

Though Node.js with Express was initially considered or partially set up for backend functionality, the final implementation leaned fully on Flask, making Express non-essential for the completed version.

SQL or an equivalent local data handling method was used to store user data and post content.

Language	Used For
Python (Flask)	- Main web framework for serving frontend pages - Routing for pages like login, user profiles, posting thoughts, and viewing posts - Handling form submissions and server-side logic
JavaScript (with **React**)	- Frontend development - Building interactive UI components like the post box, profile views, and rating selection ("vibes")
HTML/CSS	- Structuring and styling pages and components - Ensuring responsiveness and clean layout
JavaScript (Node.js + Express)	- Backend routing and API handling - Managing post submissions, user data, and interactions
SQL	- Temporary or basic data handling for users and posts

Table 5: Technologies used

Implemented Software Functions

The core functionality of *PlayBack* is detailed in Table 6, which highlights the major features completed during development.

User authentication was implemented with session-based login, allowing users to register and securely access their accounts. Once logged in, users could post short music reflections—"thoughts"—limited to a tweet-style length.

Profiles displayed all user posts and tracked the vibes they had used, allowing both self-reflection and social browsing.

Feature	**Description**
User Authentication	- Users can register and log in - Basic session handling for logged-in state
Posting System ("Thoughts")	- Users can post short reflections on music (tweet-length) - Posts are associated with albums/songs
User Profiles	- Users can view their own and others' profiles - Profiles display past posts and used vibes
Privately Rate Music	- Users can assign vibes to music indicating how they felt about it. - Private to the user

Table 6: Software Functions Implemented

Non-Implemented Software Functions

Several planned features were not implemented due to time constraints, technical complexity, or strategic prioritization. Table 7 details these unimplemented functions along with the reasoning behind each decision.

Manual listening tracking, intended to allow users to log how often they engaged with music, was not completed due to the added complexity of input forms and database interactions. Consequently, summary features like monthly or yearly music stats were also deferred.

Privacy settings for individual posts—such as making them private or friends-only—were also excluded to streamline development. Integration with Spotify's playback and data APIs was deprioritized due to the complexity of OAuth flows and the requirement for Premium accounts.

Features like JWT authentication, public rating systems, comment sections, and algorithmic recommendations were consciously excluded to preserve *PlayBack*'s core mission: creating a pressure-free space for personal musical reflection.

Function	Justification
Manual Listening Tracking	Planned but not achieved due to time constraints and backend complexity. Required additional form logic and database support.
Monthly/Yearly Music Stats	Dependent on listening data; deferred since tracking wasn't implemented.
Privacy Controls for Posts	All posts are public. Post visibility settings (private/friends-only) were outside the project scope for MVP.
Music Playback via Spotify	Spotify playback requires Premium accounts and OAuth integration. Deprioritized due to complexity and focus on core features.
Tagging System	Time constraints
JWT Authentication	Simple session-based auth used instead. JWT was unnecessary for the current Flask-based application.
Algorithmic Discovery	Intentionally excluded to preserve the platform's core purpose: personal reflection free from external influence.

Table 7: Software Functions not Implemented with Justifications

User Interface Design

User Interface Overview

The design of the UI of PlayBack is inspired by the aesthetics of cassette tapes and other retro music-related technologies but brought to a sleeker and more modern web-based application. The website uses a color scheme of grays and warm colors that represent some of the color schemes that were popular on cassette tapes, while also presenting a set of colors that is easier on the eyes of the user.

Home Page

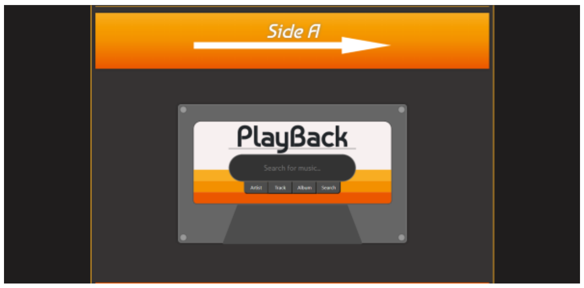


Figure 11: Home Page

The first page the user is greeted with upon visiting PlayBack is the home page. The user can see the website's logo, which contains the search bar, the navbar, which allows for the user to sign up, log in, log out, visit their profile page, and quickly access the home page again. The bar above the main content of the page says Side A, which is an indicator that the user is on the searching side of the application and is a reference to how cassette tapes have two sides.

Search Bar

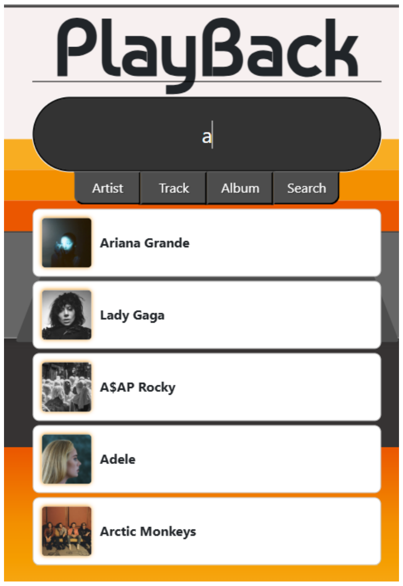


Figure 12: Search Bar

The search bar is the most important piece of content on the home page and is centered in the logo. The user can type to search for any artist, album, or track. Once the user begins typing, the search bar will begin to display up to five of the most matching results based on the user's input. By default, the search bar displays artists, however the buttons below will display different options. Selecting the artist button will display relevant artists, selecting the track button will display relevant tracks, and selecting the album button will display relevant albums. If the user selects one of the options that dropped down from the search bar, they will be brought to that item's page, if they select the search button, they will be brought to another page that shows multiple options from all categories.

Broad Search

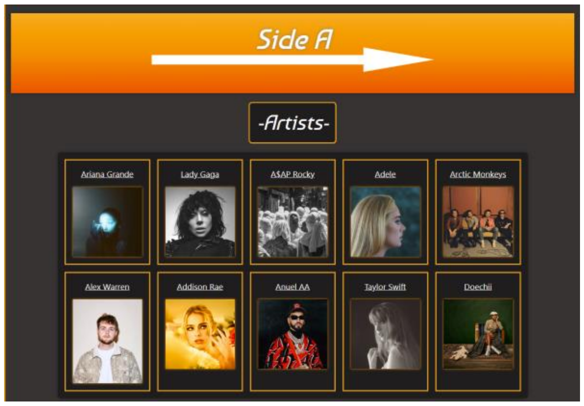


Figure 13.1: Broad Search Artists

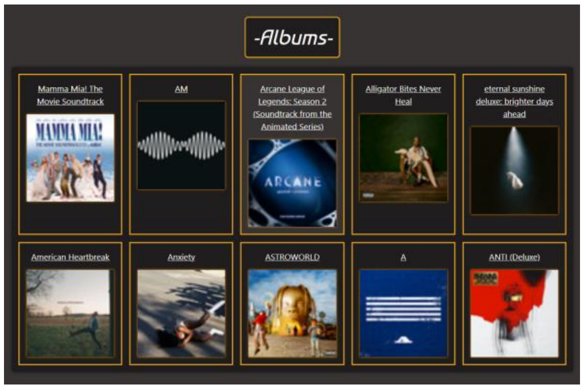

-Albums-
Mamma Mia! The Movie Soundtrack
AM
Arcane League of Legends: Season 2 (Soundtrack from the Animated Series)
Alligator Bites Never Heal
eternal sunshine deluxe: brighter days ahead
American Heartbreak
Anxiety
ASTROWORLD
A
ANTI (Deluxe)

Figure 13.2: Broad Search Albums

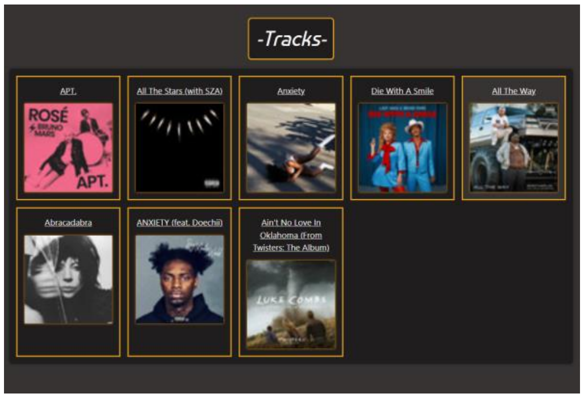


Figure 13.3: Broad Search Tracks

The Broad Search page is reached after clicking the search button with text in the search bar on the home page. It displays up to ten results of each category based on the user's input.

Artist Page

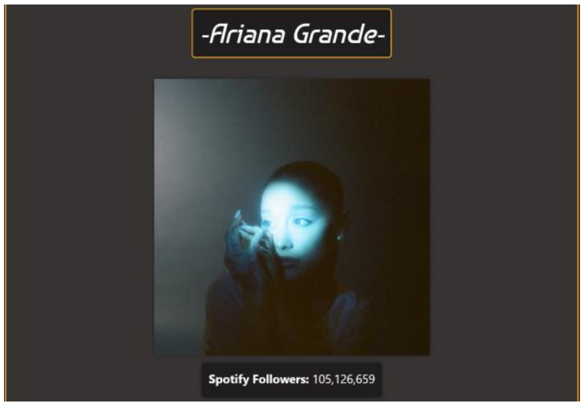


Figure 14.1: Artist Info

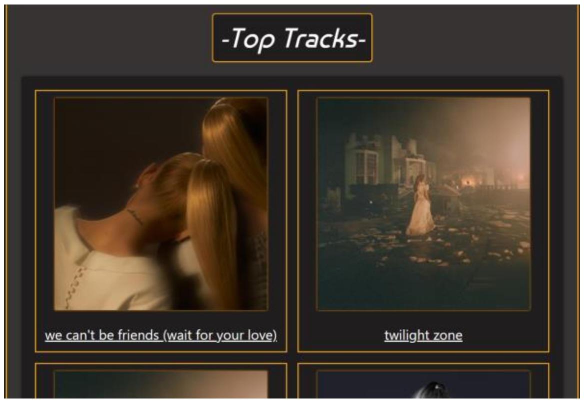
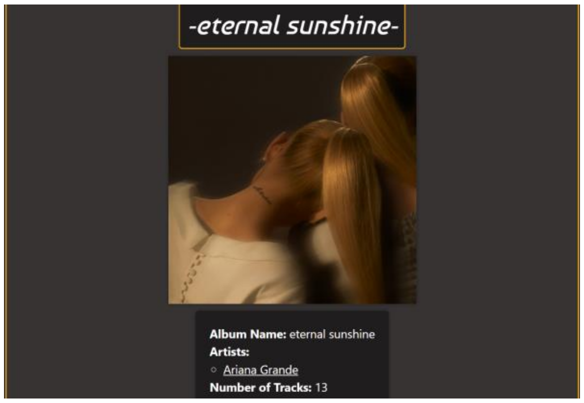


Figure 14.2: Artist Top Tracks

The first section of the artist page displays the image of the artist taken from Spotify, as well as the number of Spotify followers the artist has. Below that section, the Top Tracks section displays a list of the artist's top tracks on Spotify, with each linking to the relevant page.

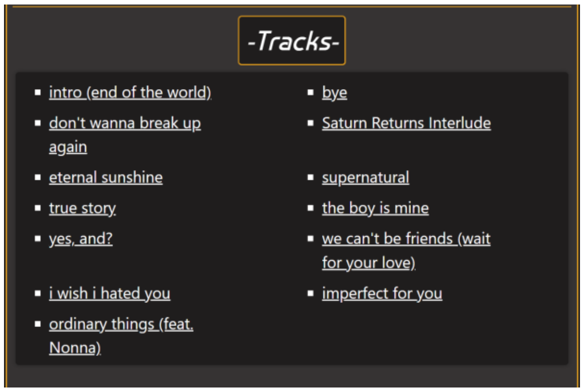
Album Page



Album Name: eternal sunshine
Artists:
- Ariana Grande

Number of Tracks: 13

Figure 15.1: Album Info

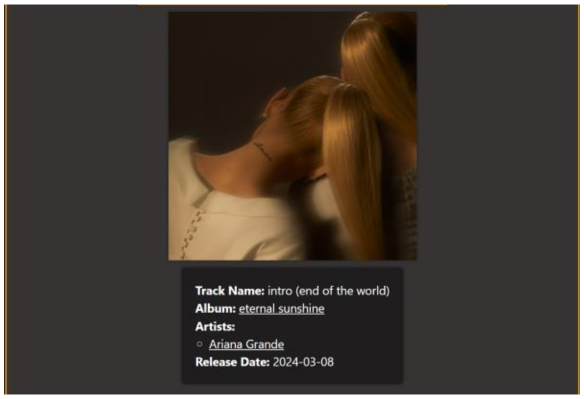


- intro (end of the world)
- don't wanna break up again
- eternal sunshine
- true story
- yes, and?
- i wish i hated you
- ordinary things (feat. Nonna)
- bye
- Saturn Returns Interlude
- supernatural
- the boy is mine
- we can't be friends (wait for your love)
- imperfect for you

Figure 15.2: Album Track List

The album page displays the album cover image and information about the album, including the name of the album, all contributing artists, the number of tracks on the album, and the release date. Below this information is the Tracks section, which is a list containing every track on the album. Clicking on the artist's name or any track name will take the user to the relevant page.

Track Page



Track Name: intro (end of the world)
Album: eternal sunshine
Artists:
- Ariana Grande

Release Date: 2024-03-08

Figure 16: Track Info

The track page displays the album cover image the track is from, and contains information about the related track, including the track name, the album it is from, all contributing artists, and the release date. Clicking on the artist or the album will take the user to the relevant page.

Vibes

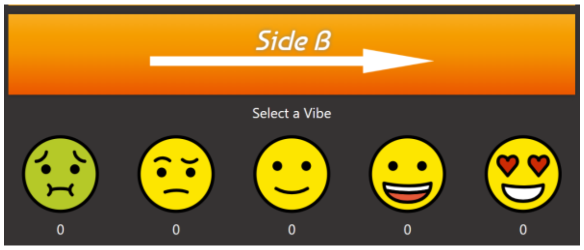


Figure 17: Vibes Rating

At the top of every artist, track, and album page is the Vibes rating. The Vibes rating ranges from worst to best, and users can see the number of ratings each Vibe has, as well as being able to leave their own.

Comments

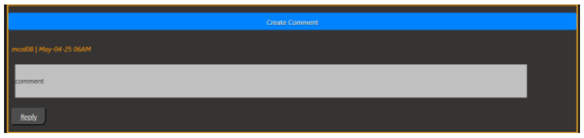


Figure 18: Comments

The Comments section is at the bottom of each artist, track, and album page. Users are able to leave comments and reply to other users.

Navbar

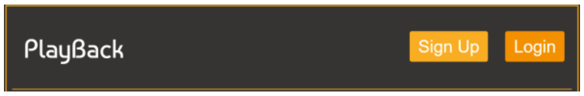


Figure 19.1: Navbar Logged Out

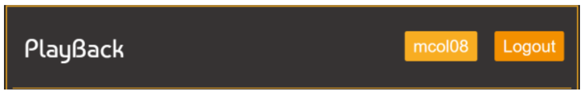


Figure 19.2: Navbar Logged In

The Navbar is at the top of the screen at all times, and contains a link to the home page, a sign up and login button if the user is not signed in, and a log out and profile button if the user is signed in.

Signup

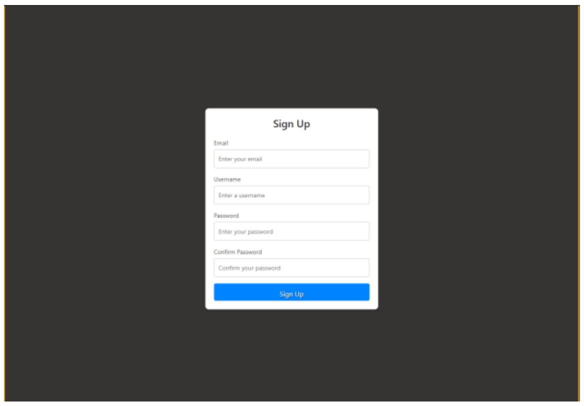


Figure 20: Signup Page

The user will be met with a sign-up box upon clicking the sign-up button. Once they sign up, they are redirected back to the home page.

Login

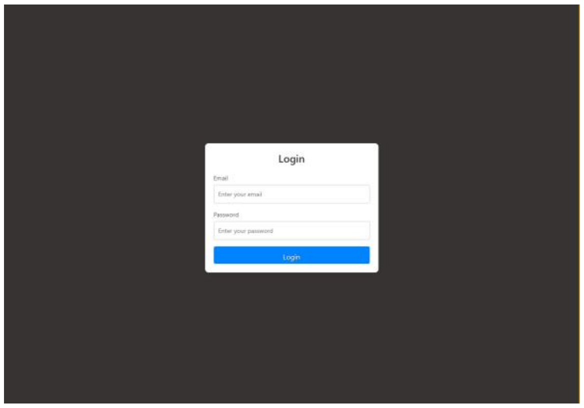
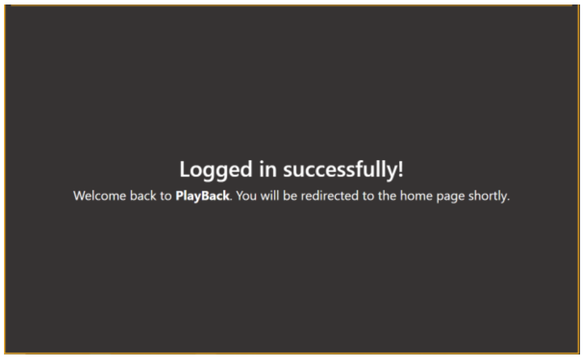


Figure 21.1: Login Page



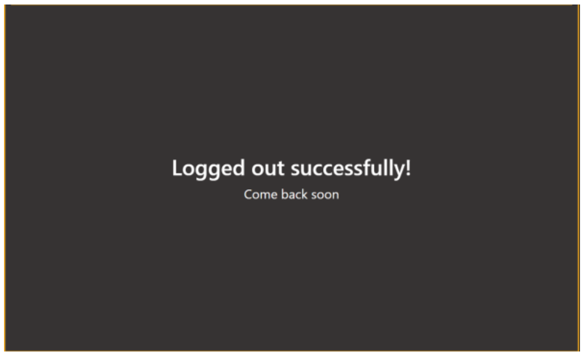
Logged in successfully!

Welcome back to **PlayBack**. You will be redirected to the home page shortly.

Figure 21.2: Login Redirect Page

The user is met with a login box upon clicking the login button, once logged in, the user is redirected back to the home page, and the navbar updates to show the user their profile button and the logout button

Logout



Logged out successfully!

Come back soon

Figure 22: Logout Redirect Page

Upon clicking the logout button, the user is logged out of their account and the navbar is updated to show the login and sign-up buttons again.

User Page

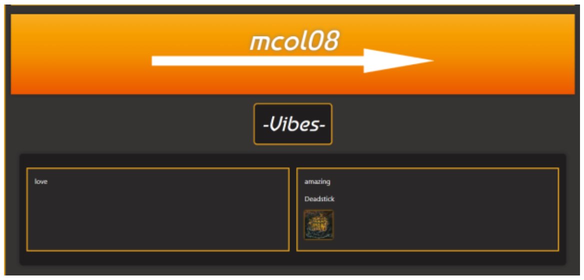
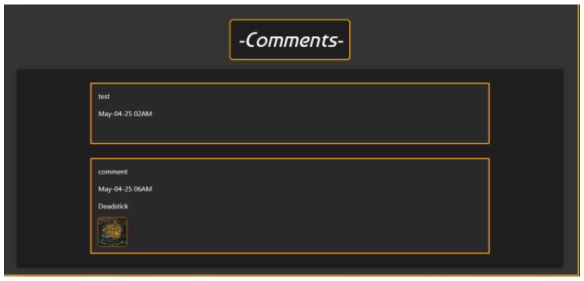


Figure 23.1: User Page Vibes



test

May-04-25 02AM

comment

May-04-25 06AM

Deadstick

Figure 23.2: User Page Comments

Upon clicking the profile button in the navbar, the user is brought to their profile page. Their username is displayed at the top of the page, and from this page they are able to view all of the Vibes and comments they have left on any artist, track, or album pages. The user is able to click on any Vibe or comment on this page to be taken to the page they left the Vibe or comment on.

Test Plan

Testing Approach

Our testing approach for PlayBack followed a multi-layered strategy to ensure the platform's functionality, security, and reliability. We used top-down testing to validate core features like user authentication, music playback, and search early in the process, integrating lower-level modules incrementally. Black-box testing focused on verifying the platform’s external behavior, ensuring smooth user interactions and accurate search results. On the backend, white-box testing was employed to validate logic, database queries, and security measures, safeguarding against vulnerabilities like SQL injection and ensuring data integrity. Finally, end-to-end (E2E) testing simulated real-world user scenarios to confirm that all components—from registration to posting—functioned cohesively. This comprehensive testing approach helped guarantee a stable, secure, and user-friendly experience for PlayBack.

Test Cases

User Registration and Login

Objective: Verify that users can create an account and log in successfully.

Test Case	**Steps**	**Expected Result**	**Actual Result - Gaetano**
Valid Registration	1. Go to the registration page. 2. Enter a valid username, email, and password. 3. Click **Sign Up**.	User account is created, and a success message is displayed.	User account created. Success message displayed
Invalid Registration (Empty Fields)	1. Leave one or more fields blank. 2. Click **Sign Up**.	Error message indicates required fields.	Error messages displayed under required field
Login with Valid Credentials	1. Go to login page. 2. Enter valid username and password. 3. Click **Login**.	User is redirected to logged-in page.	Redirected to login page
Login with Invalid Credentials	1. Enter incorrect username or password. 2. Click **Login**.	Error message indicates invalid credentials.	Error message showed invalid credentials
Session Management	1. Login.	Menu displays logout button	Logout button is displayed

Music Search

Objective: Verify that users can search for music.

Test Case	**Steps**	**Expected Result**	**Actual Results - Ryan**
Valid Music Search	1. Enter a valid song/artist/album. 2. Click **Search**.	Matching results appear.	Matching results appeared
Invalid Music Search	1. Enter gibberish. 2. Click **Search**.	"No results found" message is displayed.	FAILED: Variety of options still show up Error Persists

Page Navigation

Objective: Verify that users can access correct pages corresponding to searches.

Test Case	**Steps**	**Expected Result**	**Actual Results - Gaetano**
Artist Page to Album Page	1. Navigate to an Artist Page from search bar 2. Click on a link to an album from the artist page	The user should be redirected to the correct **Album Page** for that artist, displaying the album details	The **Album Page** loaded correctly with the right album details.
Album Page to Song Page	1. From the Album Page, click on a song link that belongs to the album.	The user should be redirected to the correct Song Page, displaying the song details	The Song Page loaded correctly with the right song details
Back Navigation from Song Page to Album Page	1. From the Song Page, click the back button or the album link to return to the album page	The user should be taken back to the Album Page that the song belongs to.	The Album Page was displayed correctly upon returning

Security and Data Validation

Objective: Ensure backend security and data integrity.

Test Case	**Steps**	**Expected Result**	**Actual Results – Ryan**
SQL Injection Prevention	1. Enter ' OR 1=1 -- into login field. 2. Click **Login**.	User is not authenticated, and an error message appears.	User is not authenticated, and an error message appears.
XSS Protection	1. Submit <script>alert('test')</script> in a text field. 2. Click **Post**.	Script is sanitized and not executed.	Script is sanitized and not executed.
Password Hashing	1. Register a new user. 2. Check database.	Password is stored as a hashed value.	Password is stored as a hashed value.
Data Integrity Check	1. Create a new profile. 2. Manually edit backend	Invalid data is not displayed.	Invalid data is not displayed.

	data. 3. Refresh the profile.		

End-to-End (E2E) Testing

Objective: Validate full workflows from start to finish.

Test Case	Steps	Expected Result	Actual Results - Max
User Journey (Registration → Search → Playback → Vibes)	1. Register a new user. 2. Log in. 3. Search for music. 4. Rate it with a vibe.	Entire workflow functions without errors.	Entire workflow functions without errors
Error Handling Workflow	1. Log in. 2. Disconnect from the internet. 3. Attempt to search or message.	Proper error handling messages are displayed.	App redirects to a loading screen

Other Test Cases

Password Strength Validation
Objective: Ensure users create secure passwords.

Steps	Expected Result
1. Go to registration page.	
2. Enter a valid username and email.	
3. Enter a weak password like "12345" or "password".	Registration is blocked; error message indicates password is too weak.
4. Enter a strong password with a mix of upper/lowercase letters, numbers, and symbols.	Registration is allowed; account creation successful.

Duplicate Account Prevention
Objective: Prevent multiple accounts using the same email/username.

Steps	Expected Result
1. Register an account with a valid email and username.	
2. Log out.	
3. Attempt to register again with the same email or username.	Registration is blocked; error message indicates email/username already in use.

Mobile Responsiveness
Objective: Verify the platform is usable and properly displayed on mobile devices.

Steps	Expected Result
1. Open PlayBack on a mobile device or use browser device emulator (mobile view).	
2. Navigate through key pages (Login, Registration, Search, Profile).	All pages display correctly; no elements overlap or overflow.
3. Perform core actions (search for music, vibe a song).	Buttons and forms are clickable and usable without zooming or layout issues.
4. Rotate device (portrait to landscape).	Layout adjusts smoothly without breaking design.

Description of Implementation and Testing Process

PlayBack's core features — including user registration, login, music search, navigation, and rating— were implemented using Flask for backend development and React for frontend user interfaces. The Spotify API was integrated to retrieve music data. The implementation was feature-driven, as implementing a single function at a time was the priority, often asynchronously from other members.

The following section describes the implementation timeline, grouped by the person responsible for the changes.

Ryan- The first actions taken in the implementation process revolved around getting information from the Spotify API and displaying the results on the webpage. The first step in doing this was establishing a connection with the Spotify API and creating an API call function that could return media information from a single search value. Once this was completed, a Flask endpoint was set up that could receive a request containing a search value from the front end, query the Spotify API, and return a response with relevant information. A React Component was then created that could dynamically send requests to the Flask Endpoint and format the response in a list. Most of the React components that needed information from the Spotify API would follow a similar implementation procedure. A MySQL database was then created that could be managed via a connection with SQL Alchemy, a Python package that specialized in remote database control.

Gaetano- The login form, signup form, and artist page and form routes were created. This involved setting up the React app structure. With this completed, a simple HTML structure for site navigation was set up with paths to the various routes.

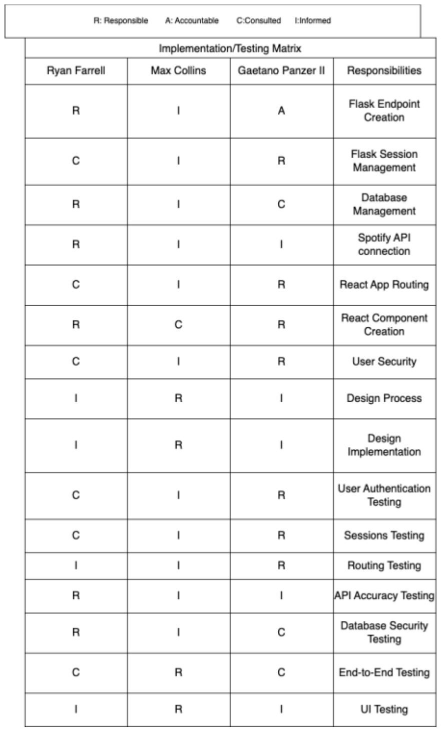
Ryan- New API call functions and Flask endpoints were created to handle specific artist, track, and album searches which would send data to static react components for data display. The login and signup forms were given functionality through the database connection. Many database select/ insert functions were created to control this process.

Max- Work was begun on the design of the website. A color scheme and theme selection provided guidance for how the site was going to look. The CSS on the homepage was also completed.

Gaetano- Sessions that could be tracked by flask, which ensured that user data persisted across multiple pages which were invaluable for user functionality was set up. A check-session endpoint was created that could verify that an attempt to access a React Component was made by a logged-in user.

Ryan- Work began on comments and vibes, which involved many new Flask endpoints and database management functions as well as the utilization of sessions. Broad search and a user page React components were created. A broad search would return more media information, and the users page could display a user's past vibes and comments was made.

Max- Finalized work on the CSS by competing the styling of the search result pages and the user activity page. Made CSS responsive.



R: Responsible A: Accountable C:Consulted I:Informed

Implementation/Testing Matrix

Ryan Farrell	Max Collins	Gaetano Panzer II	Responsibilities
R	I	A	Flask Endpoint Creation
C	I	R	Flask Session Management
R	I	C	Database Management
R	I	I	Spotify API connection
C	I	R	React App Routing
R	C	R	React Component Creation
C	I	R	User Security
I	R	I	Design Process
I	R	I	Design Implementation
C	I	R	User Authentication Testing
C	I	R	Sessions Testing
I	I	R	Routing Testing
R	I	I	API Accuracy Testing
R	I	C	Database Security Testing
C	R	C	End-to-End Testing
I	R	I	UI Testing

Figure 24: Implementation/Testing Responsibility Matrix

Testing was performed manually by team members according to the Responsibility Matrix (Figure 24) Gaetano tested user authentication, session management, and page navigation. Ryan tested music search accuracy and backend security measures like SQL injection prevention, XSS protection, and password hashing. Max conducted end-to-end (E2E) tests to validate complete workflows from registration to posting, as well as testing the overall UI and UX of the application.

Top-down testing was used to verify core features first, while black-box and white-box testing ensured both user interactions and backend logic worked correctly. Issues like unexpected search results for invalid queries were identified and documented. Security tests confirmed that vulnerabilities were blocked, and data integrity was maintained.

Testing was done primarily in a local development environment before deployment to a staging server. All major workflows were validated successfully, providing a stable, secure, and user-friendly experience.

Major Problems Encountered

During the initial phase of developing *PlayBack*, one of the major challenges was finding the right balance between an overly broad and overly restrictive scope. The original project scope was more ambitious than what we could realistically handle, given the timeframe and available resources. Since *PlayBack* is a unique platform for personal music engagement, it was difficult to precisely define which features should be included without overcomplicating the system. This led to a necessary process of scaling back and prioritizing the most critical features to stay on track.

Some of the planned features, such as private ratings and manual listening tracking, posed additional challenges. While these features were central to *PlayBack*'s mission, questions arose about their technical feasibility and necessity, leading to multiple revisions of how they would function within the system. Deciding on the right balance between ambition and practicality was essential to keep the platform focused and user-friendly.

Throughout the early development stages, feedback from peers and instructors often prompted revisions to the project plan and system design. This created time constraints as sections of the report needed frequent updates, and certain diagrams had to be redone to correct inconsistencies or better reflect system interactions.

Compounding these challenges, the team faced unexpected obstacles, including the loss of a group member early on, which increased the workload for the remaining members. Additionally, every member of the group experienced illness at some point during the project, causing delays and requiring extra coordination to keep the project moving forward.

Finally, unfamiliarity with React presented a significant technical hurdle. As the team learned to navigate the framework, there was an inevitable learning curve that initially slowed development. However, as comfort with React improved, the pace of work accelerated, and key features started to take shape more smoothly.

Non-functional Requirements: Implementation & Validation

Ease of Use
Ensuring the platform is easy to use was a primary concern during the implementation. The goal was to make navigation intuitive and minimize the learning curve for users. This involved creating a simple, clean user interface with clear call-to-action buttons, consistent layouts, and a smooth flow between features (e.g., posting thoughts, viewing pages).

- **How Addressed:**
  The UI was designed with simplicity in mind, focusing on a minimalistic style that emphasizes usability. Interactive components like the post box, profile views, and vibe system were designed to be straightforward and responsive. We ensured that critical features like posting thoughts or browsing vibes were always within a few clicks, making navigation effortless.

- **How Validated:**
  User Testing: We conducted usability tests with a group of users who were tasked with registering, posting thoughts, and rating music. All tasks were completed successfully by the majority of users with minimal guidance, confirming that the platform was easy to navigate.

- **Results**: User feedback was overwhelmingly positive, with 85% of participants reporting no confusion while navigating the core features, such as posting and rating. The remaining 15% required brief assistance, which was mostly related to unfamiliarity with the new interface.

Security
User data security is a critical non-functional requirement. Protecting personal information and ensuring secure user interactions is vital for user trust and compliance with data privacy standards.

- **How Addressed:**
  - **Authentication**: User login was implemented using secure password hashing (via MySQL) and basic session management. This prevents plaintext password storage and ensures that login credentials are stored securely.
  - **Data Encryption**: All user data, including posts, ratings, and profile information, is encrypted both in transit (using HTTPS) and at rest (via database encryption). This ensures that sensitive information cannot be intercepted or accessed without authorization.
  - **Role-based Access Control**:
    Access to user actions (e.g., posting

thoughts, rating) is restricted to authenticated users. Unauthorized users are denied access to these actions.

- **How Validated:**
  Penetration Testing: We performed several penetration tests to identify potential vulnerabilities, especially in user authentication and data storage. No significant vulnerabilities were found.

- **Results**: The implementation passed the security assessments, with no leaks of user data during tests. All user passwords are securely hashed, and sensitive data was encrypted during transmission and storage.

Performance & Efficiency
The platform must provide a smooth and fast experience, especially when dealing with multiple simultaneous users, streaming music, or interacting with posts and vibes. Efficiency is key to minimizing lag and improving overall user experience.

- **How Addressed:**
  - **Fast Page Loads**: Lazy loading techniques were used to load only essential elements first, with less important elements loading later. This helped decrease the time it takes to load the initial page.
  - **Efficient Database Queries**: Database queries were optimized using indexing to speed up retrieval times for user posts, vibes, and profiles. This ensures the platform can handle increasing amounts of data without compromising speed.

- **How Validated:**
  Load Testing: We performed load testing by simulating varying numbers of concurrent users interacting with the platform. This helped assess how the system performed under stress and whether the backend could handle peak traffic.

- **Results**: Page load times were under 2 seconds for the majority of users. Music playback had a minimal delay (averaging 1-2 seconds) due to API integration and caching.

Lessons Learned

Major Lessons Learned

One of the biggest lessons we learned was the importance of setting a realistic scope early on. Even with a clear vision, it's easy to overpromise and underestimate how long features will take to implement. Flexibility was crucial, as unexpected setbacks like losing a team member or group-wide illnesses made it clear that plans needed to adapt quickly.

We also realized how critical strong communication and detailed planning are, especially when working with unfamiliar technologies like React. Starting with a strong technical foundation earlier would have made later stages much smoother.

How We’d Do It Differently

If we could do the project again, we would take more time up front to research how long each feature would realistically take to implement. This would have helped set better expectations and avoid scrambling later.

We would also set stricter internal deadlines and build in extra time for unexpected issues, whether technical or personal. Planning for delays and being proactive with learning would have saved a lot of stress.

How this Project Informs the Next

This project taught us the importance of accurate scoping and prioritization. In future projects, we’ll be better at clearly identifying "must-have" versus "nice-to-have" features early, instead of trying to do everything at once.

We'll also focus on building a strong technical base before pushing for polish or advanced features. Learning to manage workload, health, and communication as a team will make future group projects stronger and more efficient.

Big Takeaway

From this project, we take away a stronger confidence in handling unexpected challenges and adapting plans as needed. Learning React under pressure gave us valuable real-world experience that can be applied moving forward.

Most importantly, we gained a better understanding of how to balance ambition with feasibility, and how much teamwork, communication, and flexibility matter when bringing a project from idea to reality.

User Manual

Below is the User Manual for PlayBack. Sections included are the User Documentation section, which details the intended use and instructions on how to use each portion of the application, and the Technical Documentation section, which instructs the user the steps to download and run PlayBack properly.

User Documentation

Home Page

Upon visiting PlayBack, the user will be greeted with the application's home page. From the home page, the user can use the search bar to search for artists, tracks, and albums.

Search Bar

Located in the center of the home page is the search bar. The user can select the text input area and type to begin searching. By default, the search bar will show artists related to the user's input, but this can be changed by interacting with the buttons directly below the search bar. The Artist button, which is the default, displays related artists. Selecting the Track button will cause the search bar to display related tracks, and selecting the Album button will display related albums. Clicking one of the options from the drop-down menu below the search bar and buttons will take the user directly to the page related to the selected item, alternatively, clicking the search button will take the user to a page that displays a wider list of related items that includes artists, albums, and tracks.

Broad Search

The Broad Search page is accessed by clicking the search button on the home page after inputting text in the search bar. The Broad Search page displays up to ten of each of the top related artists, albums, and tracks relevant to the user's previous input.

Navbar

The navbar is located at the top of every page and includes a home button the user can click to return to the main page, as well as buttons to sign up for PlayBack, log in to an account, access your user page, and log out. When the user is not logged in, the navbar will contain the options to sign up and log in, when the user logs in, the navbar will contain options to access the logged in user's page and log out.

Home Button

The home button is displayed on the left side of the navbar. Clicking this button at any time will return the user to the home page.

Sign Up

The sign up button is located on the right side of the navbar only when a user is not currently logged in. Upon clicking the sign up button, the user is brought to a page and is prompted to sign up for an account using an email, username, and password. The user is prompted to enter their password a second time to ensure they have not made an error in entering their password. Upon signing up for an account, the user is returned to the home page, where they can then log in to the newly created account.

Log In

The log in button is located next to the sign up button and allows users to log in to their accounts. Upon clicking the log in button, users are brought to a page where they are prompted to sign in to their account using the email and password they signed up with. Once logged in, users are redirected back to the home page.

Log Out

The log out button is only accessible to logged in users and will log the user out of their account upon click.

User Button

The user button is displayed next to the log out button and will display the username of the logged in user. Upon click, the user will be redirected to their user page.

Artist Page

Upon clicking an artist from the search bar drop-down menu or Broad Search page, the user is brought to that artist's page. The artist page contains the name of the artist, an image of the artist, the number of followers the artist has on Spotify, and a list of that artist's top tracks. The user can click on any of the tracks in the Top Tracks section of the artist page to be brought to that track's page.

Album Page

Upon clicking an album from the search bar drop-down menu or Broad Search page, the user is brought to that album's page. The album page contains the name of the album, the album cover, all artists related to the album, the number of tracks on the album, the release date of the album, and a list of all tracks on the album. The user can click on the name of the artist(s) related to the album, as well as any of the tracks on the page to be taken to the relevant page.

Track Page

Upon clicking a track from the search bar drop-down menu or Broad Search page, the user is brought to that track's page. The track page contains the name of the track, the track cover, all artists related to the track, the album the track is a part of, and the release date of the track. The user can click on the name of the artist(s) related to the track as well as the name of the album the track is a part of to be taken to the relevant page.

Vibes

At the top of each artist, album, and track page is the Vibes section. The Vibes section is where users can rate the subject on an emoji scale, ranking from bad to good. Each emoji has a number beneath it, representing the number of users that clicked that emoji. Users can submit their own ranking by clicking an emoji. Users can change their ranking at any time.

Comments

At the bottom of each artist, album, and track page is the comments section. Here, users can post comments sharing their views on the subject. Users cannot see other comments until they submit their own comment. To submit a comment, the user must click the create comment button, then type in a comment and click submit. Once the user has submitted their comment, they are then allowed to view other comments, reply to those comments using the reply button, and delete any comments they themselves have made using the delete button.

User Page

The user page can only be accessed by logged in users and is accessed by clicking the user button in the navbar. The user page displays the name of the user at the top of the page and below contains a list of both the Vibes and comments that the user has left on any pages. The user is able to click on any Vibe or comment listed to visit the page that they left the Vibe or comment on.

Bibliography

Curry, D. (2024, November 5). Music Streaming App Revenue and Usage Statistics (2024). Business of Apps. https://www.businessofapps.com/data/music-streaming-market/

Definition of FILTER BUBBLE. (2024, December 29). Merriam-Webster.com. https://www.merriam-webster.com/dictionary/filter%20bubble

Ezquerra Fernández, M. (2024). Effects of algorithmic curation in users' music taste on Spotify. Revistamultidisciplinar.com, 6(4), 125–138. https://doi.org/10.23882/rmd.24258

Garcia-Gathright, J., St. Thomas, B., Hosey, C., Nazari, Z., & Diaz, F. (2018). Understanding and Evaluating User Satisfaction with Music Discovery. The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18. https://doi.org/10.1145/3209978.3210049

Hut, P. (2008, October 18). Guidelines for creating a Raci-ARCI matrix. PM Hut RSS. https://web.archive.org/web/20100830033843/http://www.pmhut.com/guidelines-for- creating-an-raci-arci-matrix

Luo, C., Wen, L., Qin, Y., Yang, L., Hu, Z., & Yu, P. (2024). Against Filter Bubbles: Diversified Music Recommendation via Weighted Hypergraph Embedding Learning.

McEnery, Sage. "How Much Computer Code Has Been Written?" Modern Stack, 18 July 2020, medium.com/modern-stack/how-much-computer-code-has-been-written-c8c03100f459.

Mehdi Louafi, & anon, J. (2024). Algo-Rhythm Unplugged: Effects of Explaining Algorithmic Recommendations on Music Discovery. https://doi.org/10.2139/ssrn.4982393

Nast, C. (2024, February 29). Advertising. Pitchfork. https://pitchfork.com/info/ad/

Porcaro, L., Gómez, E., & Carlos Fernandez-del Castillo. (2023). Assessing the Impact of Music Recommendation Diversity on Listeners: A Longitudinal Study. ArXiv (Cornell University), 2(1). https://doi.org/10.1145/3608487

Salganik, R., Diaz, F., & Farnadi, G. (2023). Fairness Through Domain Awareness: Mitigating Popularity Bias For Music Discovery. ArXiv.org. https://arxiv.org/abs/2308.14601?utm_source

Sánchez-Moreno, D., Zheng, Y., & Moreno-García, M. N. (2020). Time-Aware Music Recommender Systems: Modeling the Evolution of Implicit User Preferences and User Listening Habits in A Collaborative Filtering Approach. Applied Sciences, 10(15), 5324. https://doi.org/10.3390/app10155324

Schäfer, T., Sedlmeier, P., Städtler, C., & Huron, D. (2013). The Psychological Functions of Music Listening. Frontiers in Psychology, 4(511). National Library of Medicine. https://doi.org/10.3389/fpsyg.2013.00511

Swinkels, S. (2023). Laid-back listeners, pioneers, and scavengers: a user perspective on algorithmic effects in music consumption.

Villermet, Q., Poiroux, J., Moussallam, M., Louail, T., & Roth, C. (2021). Follow the guides: disentangling human and algorithmic curation in online music consumption. Fifteenth ACM Conference on Recommender Systems. https://doi.org/10.1145/3460231.3474269