PlayBack – Software Design Report

Gaetano Panzer II – Project Manager, Max Collins – Requirements Engineer

Ryan Farrell – Software Architect

University of North Carolina - Wilmington

April 2025

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 give 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. Users can rate albums for their reference, but these scores remain private. This removes the pressure to conform to an aggregate rating and encourages 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 reviews before gaining access to others, ensuring that their first impression of an album is their own rather than a reaction to existing opinions. Beyond reviews, the site also functions as a tool for tracking listening habits. Users manually enter their data, allowing them to reflect on their personal music journey in a way that algorithms cannot capture. This empowers 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 reviewers 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 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, adheres 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 complies 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, the platform encourages users to reflect on their musical journeys over time. This fosters a deeper connection to music and promotes 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

Scope

PlayBack is a web application designed to give 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 track their listening habits, share personal thoughts on music, and discover new music through user-driven tags (referred to as "vibes"). 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 will be able to create profiles, securely log in, and engage with others through direct messages. The system will also allow them to track how often they've listened to a particular piece of music, offering 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 will allow users to write tweet-length "thoughts" to capture their reactions or feelings about a song or album. Direct messaging (DM) functionality will allow private communication between users, fostering more personal engagement. The platform will enable users to track their listening habits, with mechanisms to record how many times they've listened to a song or album and offer 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, further enhancing the personal nature of their engagement with music. A tagging system (vibes) will help group similar music, facilitating discovery without relying on algorithms or external recommendations. While the platform will not recommend music based on others' ratings, there could be an option to display users' top media or preferred music categories on their profiles as an additional feature after core features are solidified.

The platform will include a numeric rating system, but it will be private to the user. Users will be able to rate music privately, without these ratings being visible to others, allowing for personal reflection without external comparison. The platform will not feature comment sections on posts or automatic content generation (like playlists or stations). The discovery aspect of the platform will be driven by user-generated tags (“vibes”) that help group similar music and foster organic music exploration. Additionally, users will 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 it 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.

Objectives

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. Additionally, a tagging ("vibes") system will be developed to allow users to group and discover similar music.

The posting system will be implemented to enable users to express their thoughts about music in a concise and personal way, without 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 should 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

Functional Requirements Met

Users should be able to create accounts, post “thoughts” about music, and interact with others’ posts. User-entered statistics must be accurately saved, displayed, and editable. The system will allow users to manually track their listening habits and review personal trends. Additionally, users will have control over the privacy settings of their thoughts, choosing whether to make them public, private, or share them with friends. Spotify API integration will function seamlessly, allowing users to listen to tracks directly within the platform.

User Satisfaction

At least 80% of early users should report a positive experience with the site’s navigation, design, and personalization features. Feedback should indicate that the “thoughts” format encourages more engagement and reflection compared to traditional review formats, with users feeling that the site’s design fosters a more personalized, exploratory music experience. The platform’s “vibes” and tagging system should resonate well with users, promoting organic music discovery and meaningful reflection.

Performance and Reliability

The website must load in under 3 seconds for 90% of users to ensure smooth and efficient browsing. No critical bugs or crashes should occur during the first month of public testing.

Engagement Metrics

There should be a set number of user posts and interactions, for example, 20 “thoughts” posted in the first three months. There should also be consistent daily or weekly active users, with users regularly returning to reflect on their evolving music tastes.

Project Completion

The project must be delivered on time, with all planned features working as intended, including secure profile creation, the “thoughts” system, Spotify integration, and manual listening data tracking.

Background and Related Work

Background

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.

Related Work

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. The platform's focus on manually inputted listening data and its tag-based discovery system encourages users to explore music in a way that is both intentional and personal. 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

This report outlines the requirements, scope, and objectives of a music engagement platform aimed at allowing users to share their personal thoughts on music without the influence of external opinions. It details both functional and non-functional requirements, success criteria, and project constraints to guide the development of a platform that prioritizes personal reflection and authentic engagement with music.

Software Project Plan

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
  - External Library that allows users to sign in using google/ other accounts
  - 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
    - Easy to set up but will cost money.
    - There are many website hosting options, we will likely go with whatever is cheapest
- 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

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

Estimated Resources

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.

GANTT Chart

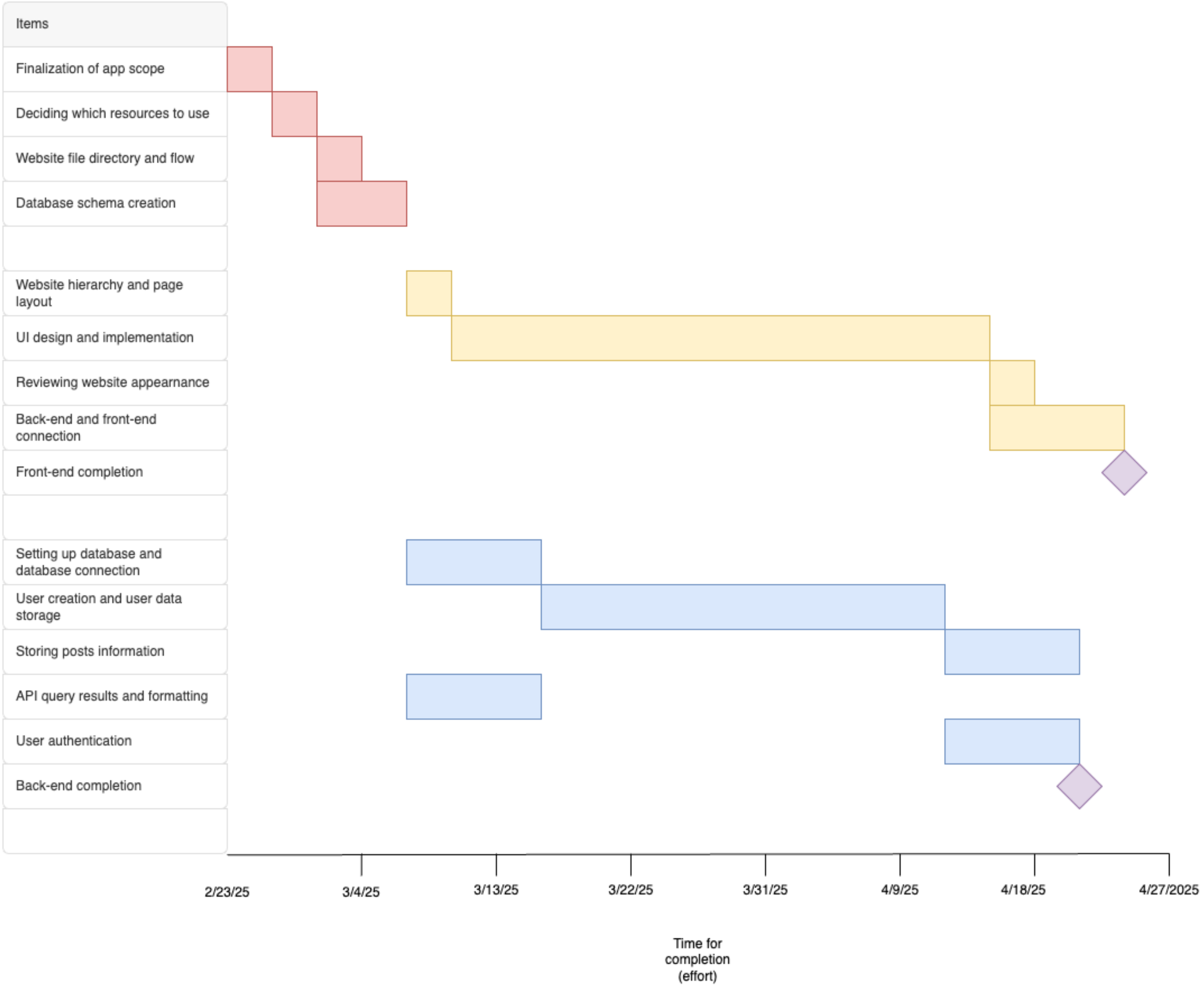


Figure 5: Gantt Chart for Project Schedule

CPM or PERT Chart

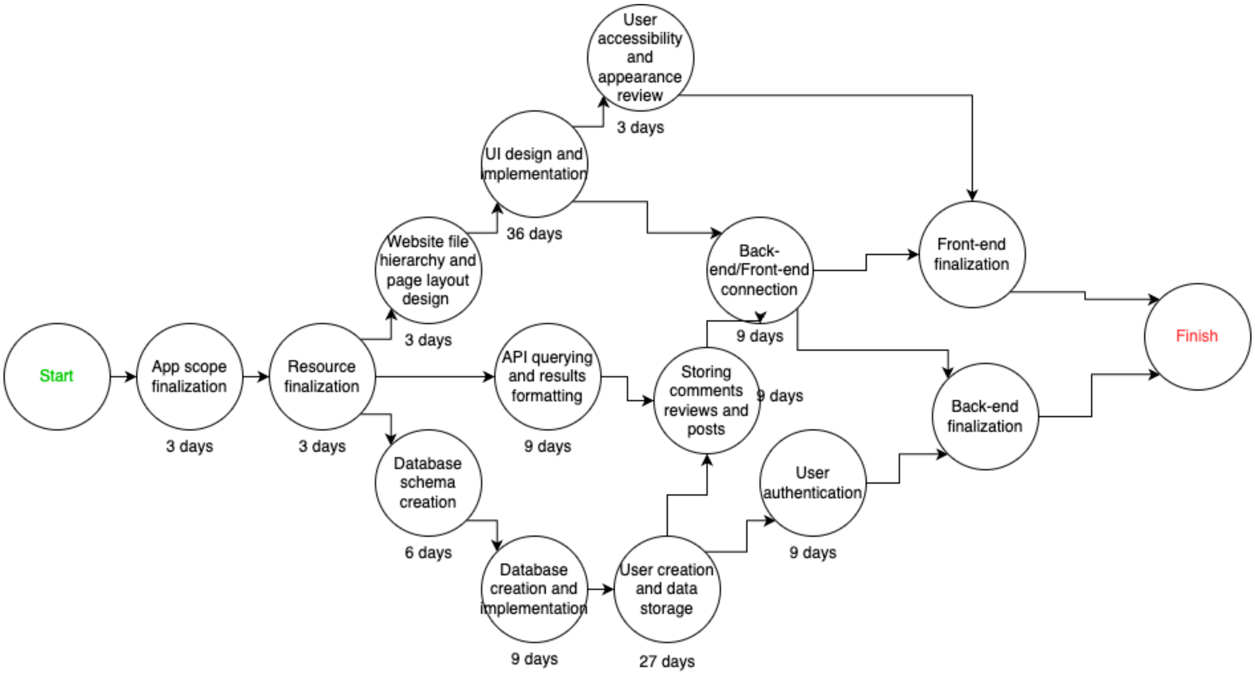
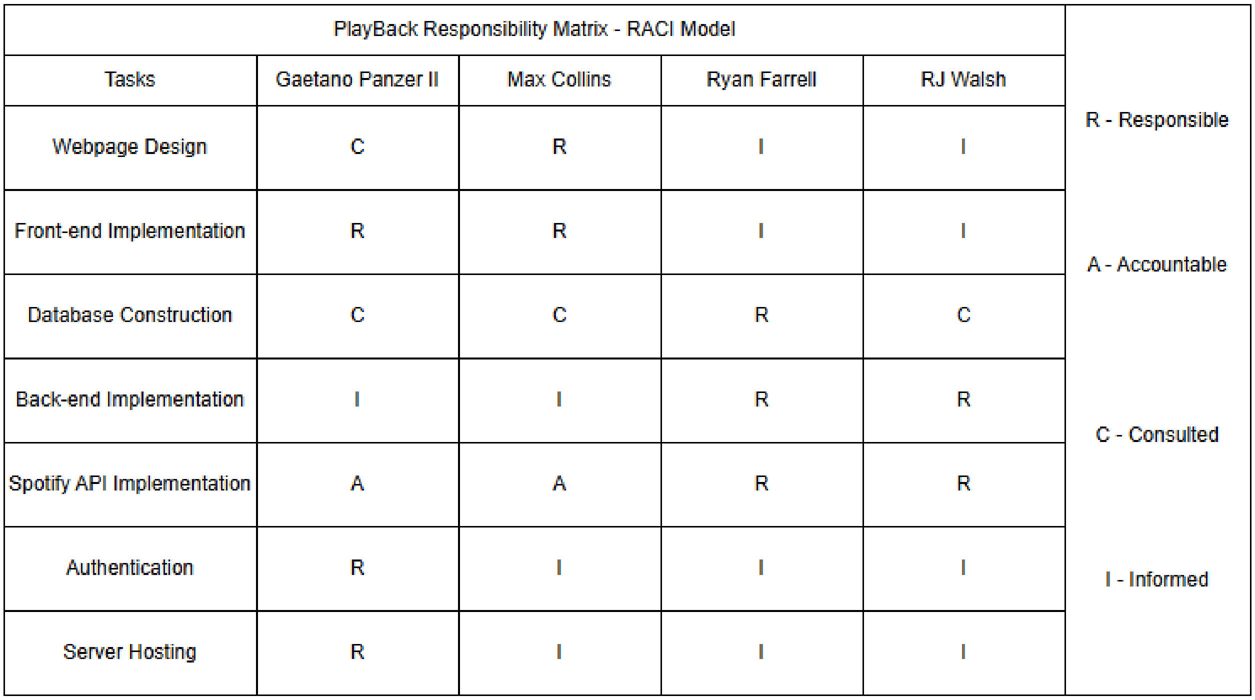

User accessibility and appearance review
3 days
UI design and implementation
36 days
Website file hierarchy and page layout design
3 days
Back-end/Front-end connection
9 days
Front-end finalization
Finish
Start
App scope finalization
3 days
Resource finalization
3 days
API querying and results formatting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

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	RJ Walsh	
Webpage Design	C	R	I	I	R - Responsible
Front-end Implementation	R	R	I	I	A - Accountable
Database Construction	C	C	R	C	
Back-end Implementation	I	I	R	R	C - Consulted
Spotify API Implementation	A	A	R	R	
Authentication	R	I	I	I	I - Informed
Server Hosting	R	I	I	I	

Table 3: Responsibility Matrix

Design Process as of 3/31

Ryan

- Paper contributions, (1/3 of whole)
- Started on backend

Gaetano

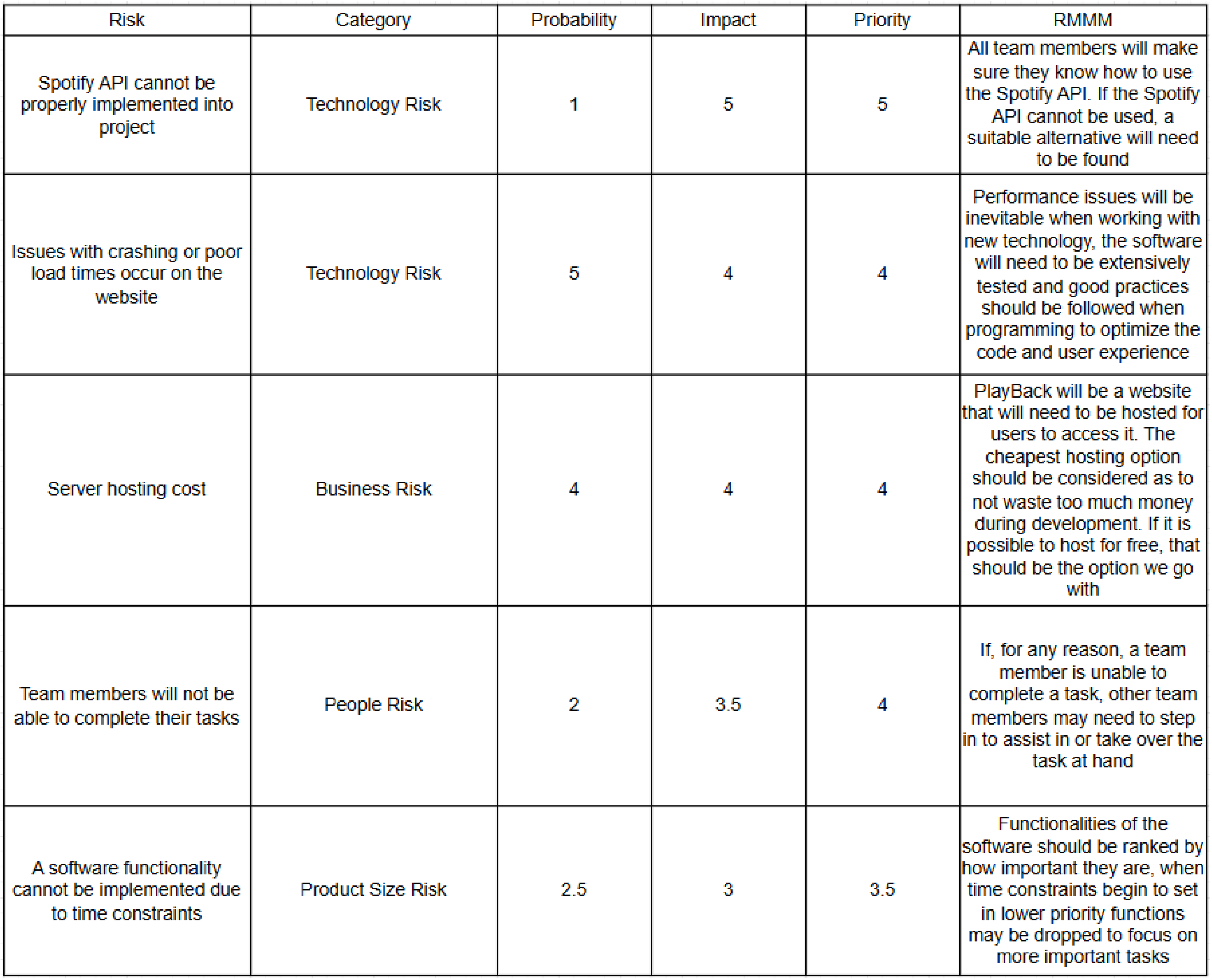
- Paper contributions (1/3 of whole)
- Generating ideas

Max

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

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

Anticipated Issues and 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 therefore will induce a learning curve as development of PlayBack moves forward. As shown in Tabl 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 issues that may rise with 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 problem but will be addressed when they arise. Some problems may also arise from real-world constraints. Since development time of PlayBack is currently limited to the amount of time remaining in the current semester, some less vital functions may have to be cut to allow us to focus on the main functions of the software. Group members also have other class projects and jobs outside of school which will more likely than not cause conflicts with an ideal schedule for programming this software, however with proper teamwork and planning, this risk should be able to be mitigated.

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.

Requirements/Analysis Models

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, posting thoughts, and managing music libraries. Together, these diagrams provide a clear visual representation of PlayBack’s core features, helping define the system’s functional requirements and user experience.

Use Case Diagrams

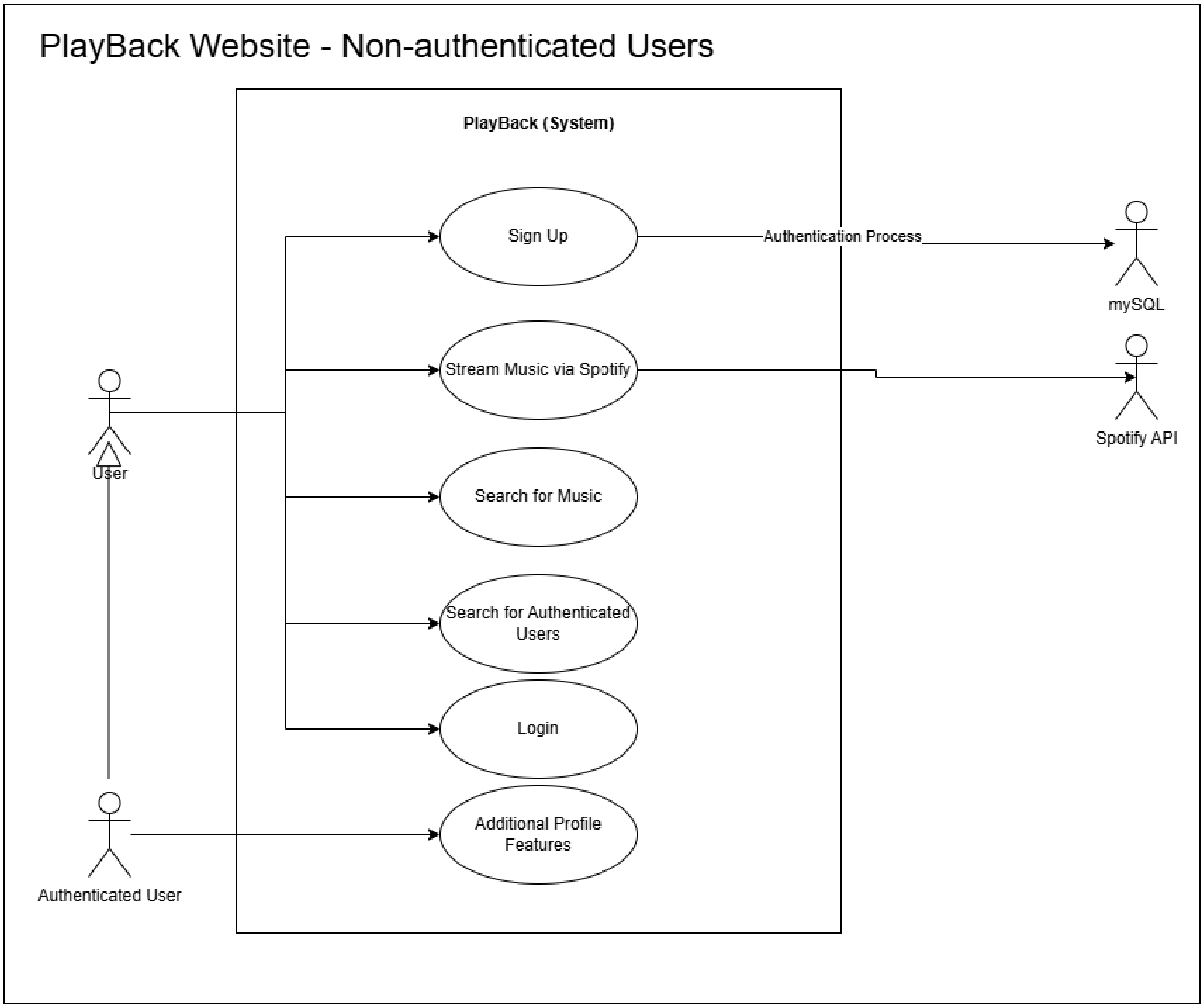


Figure 7: Use Case Diagram for Non-authenticated Users

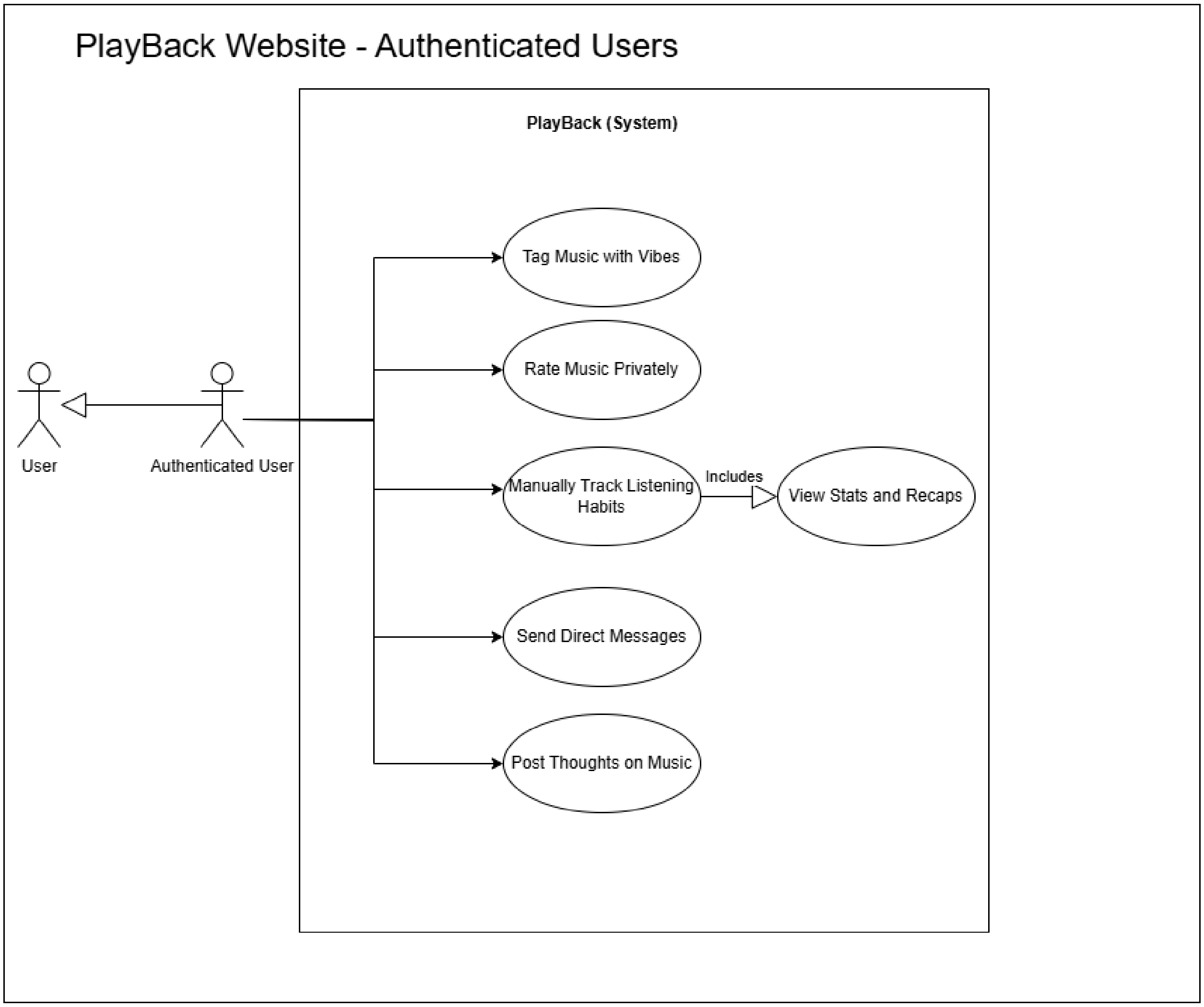


Figure 8: Use Case Diagram for Authenticated Users

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, tagging, 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 Playback

- **Function**: Enables users to stream music through Spotify's API.
- **Actors**: User, Authenticated User
- **Use Cases**:
  - **Stream Music via Spotify**: Links to Spotify API for playing music.

3. 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.

4. Music Tagging (Vibes)

- **Function**: Lets authenticated users tag music with descriptive "vibes."
- **Actors**: Authenticated User
- **Use Cases**:

    - **Tag Music with Vibes**: Apply user-generated tags to music for categorization.

5. Private Music Rating

- **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).

6. Listening Habit Tracking

- **Function**: Tracks user listening data and generates personalized insights.
- **Actors**: Authenticated User
- **Use Cases**:
    - **Manually Track Listening Habits**: Input and log how often a user listens to a particular song or album.
    - **View Stats and Recaps**: View monthly or yearly listening summaries to track trends.

7. Direct Messaging (DM)

- **Function**: Allows authenticated users to send private messages to each other.
- **Actors**: Authenticated User
- **Use Cases**:
    - **Send Direct Messages**: Users can communicate privately with each other.
    - **Receive Direct Messages**: Users can view messages from others.

8. 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, music playback, tagging, and direct messaging. It highlights the decision points and sequences involved, providing a clear depiction of the system's behavior.

Activity Diagram

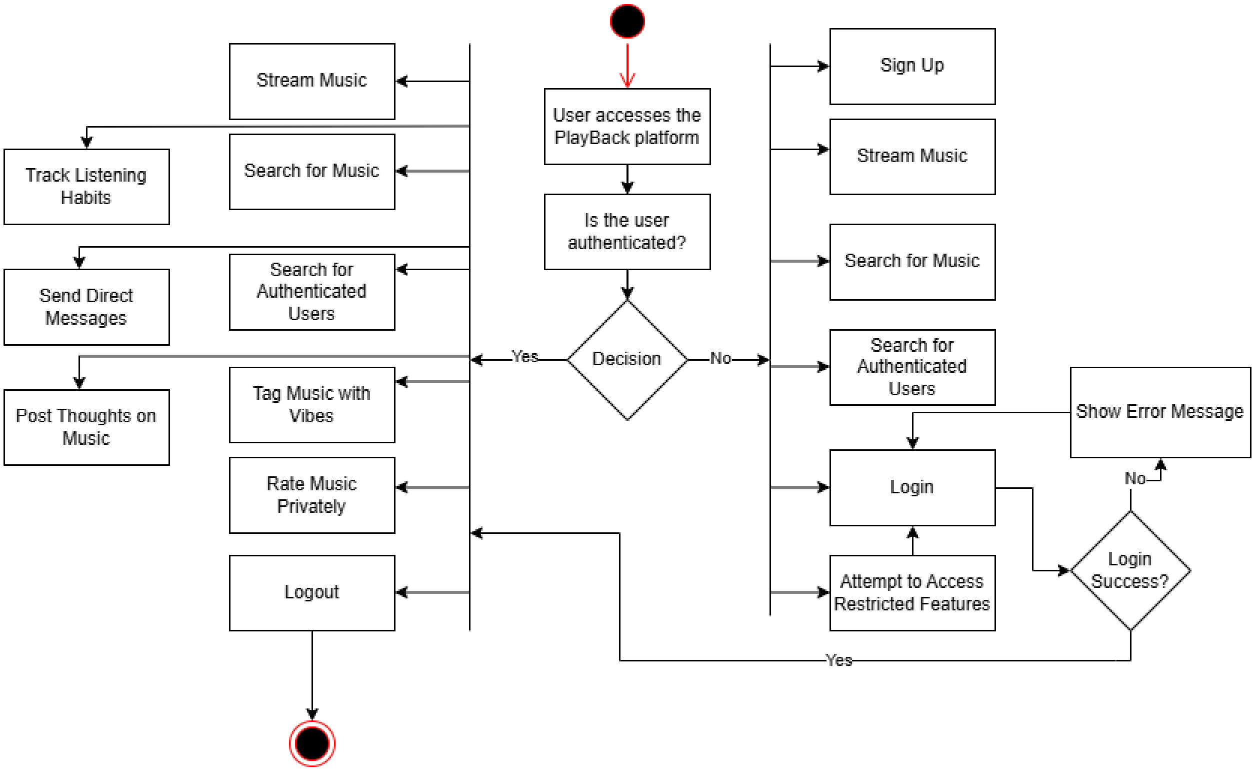


Figure 9: Activity Diagram

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.

Requirements Class Models

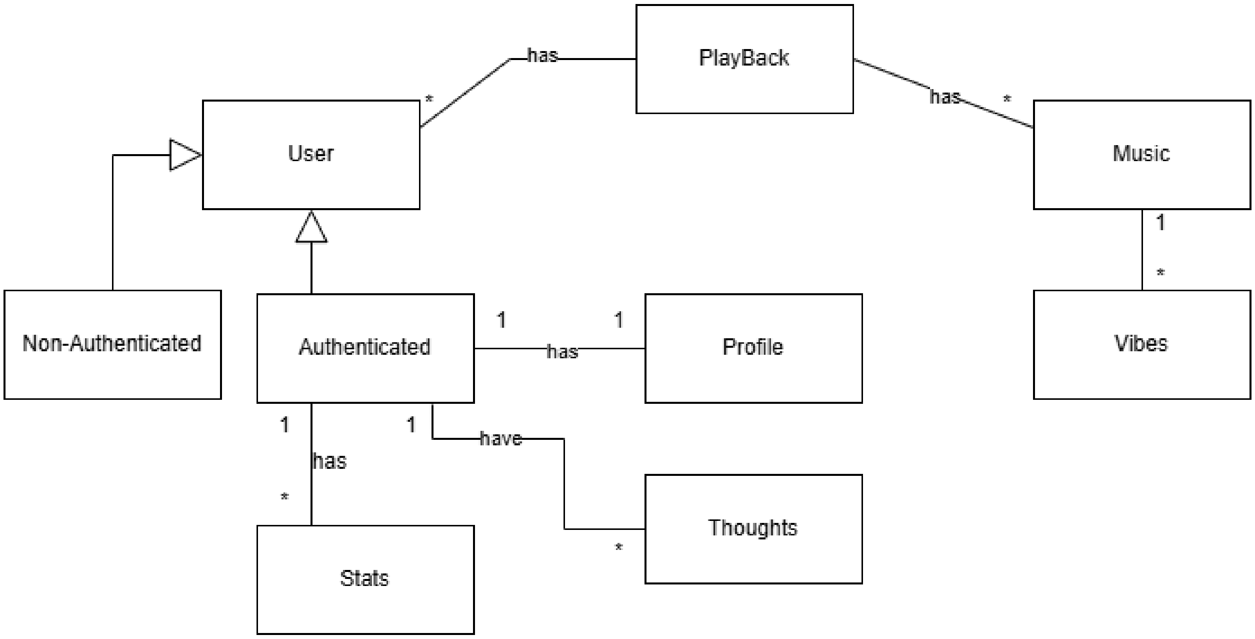


Figure 10: Class Model

Requirements/Data Dictionary

The data dictionary in Figure 11 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 Data

- <u>Song ID</u> – Song identifier

Album Data

- <u>Album ID</u> – Album identifier

Post Data

- <u>Post ID</u> – Post identifier

- Date Posted – Time post was created
- Header – Header of post
- Content – Content of post
- User ID (FK) - Points to the user who created the post
- Song ID (FK) - Points to the song the post is about

Comment Data

- Comment ID – Comment Identifier
- Content – Contents of comment
- Date Posted – Time of comment creation
- Post ID (FK)(Optional) - Points to the post being commented on
- 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
- Profile Picture – Visual representation of user

Figure 11: Data Dictionary

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 PHP + MySQL backend with an HTML + CSS frontend for simplicity and efficiency. Authentication will rely on PHP 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 PHP 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

Client-Server Model

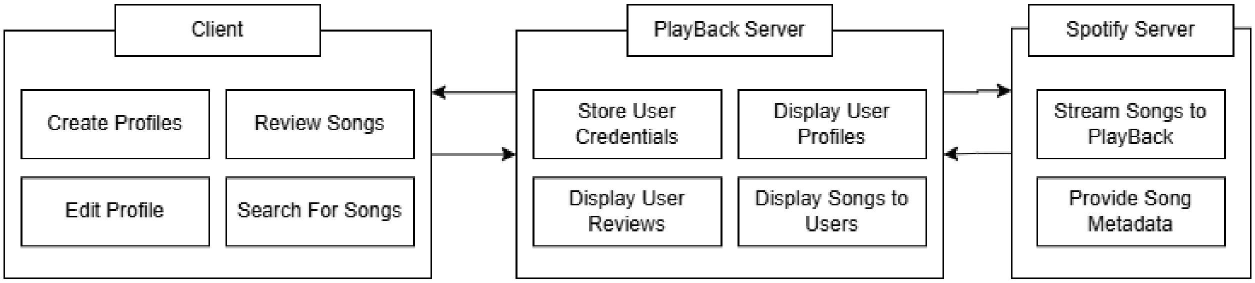


Figure 12: Layered architecture model

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.

Layered Architecture

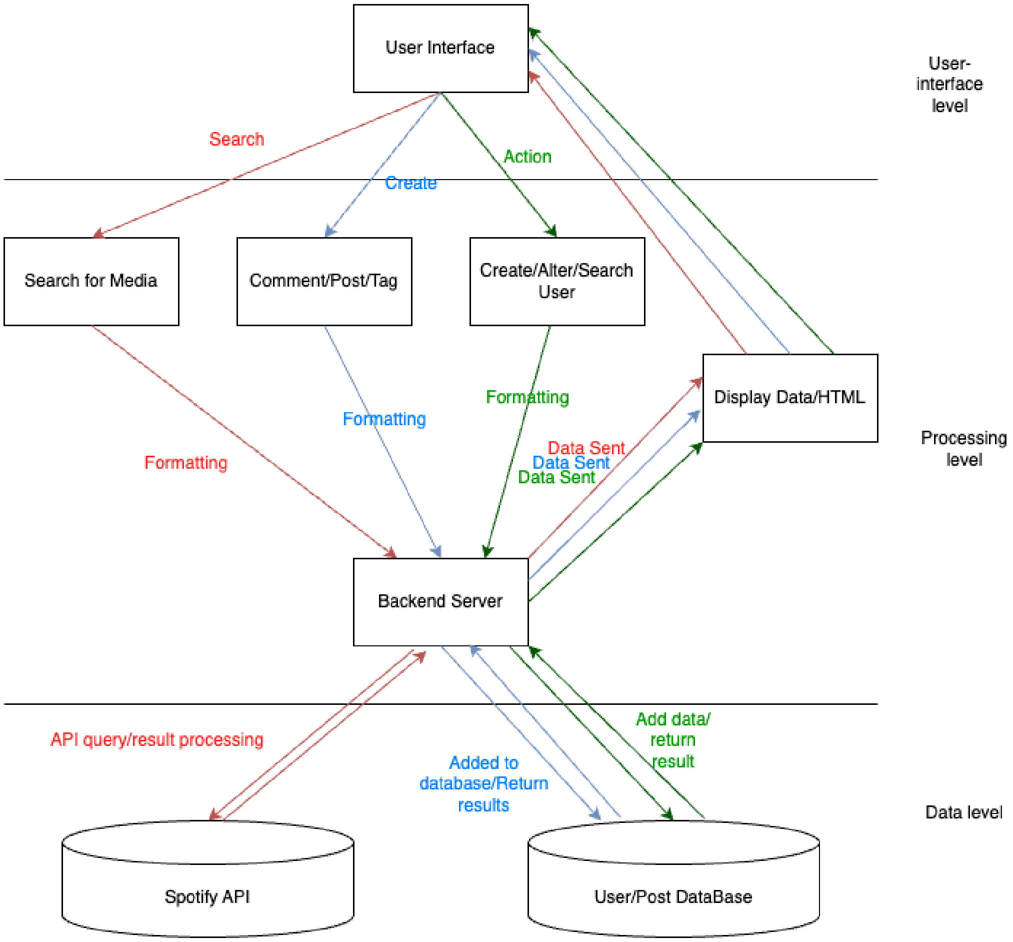


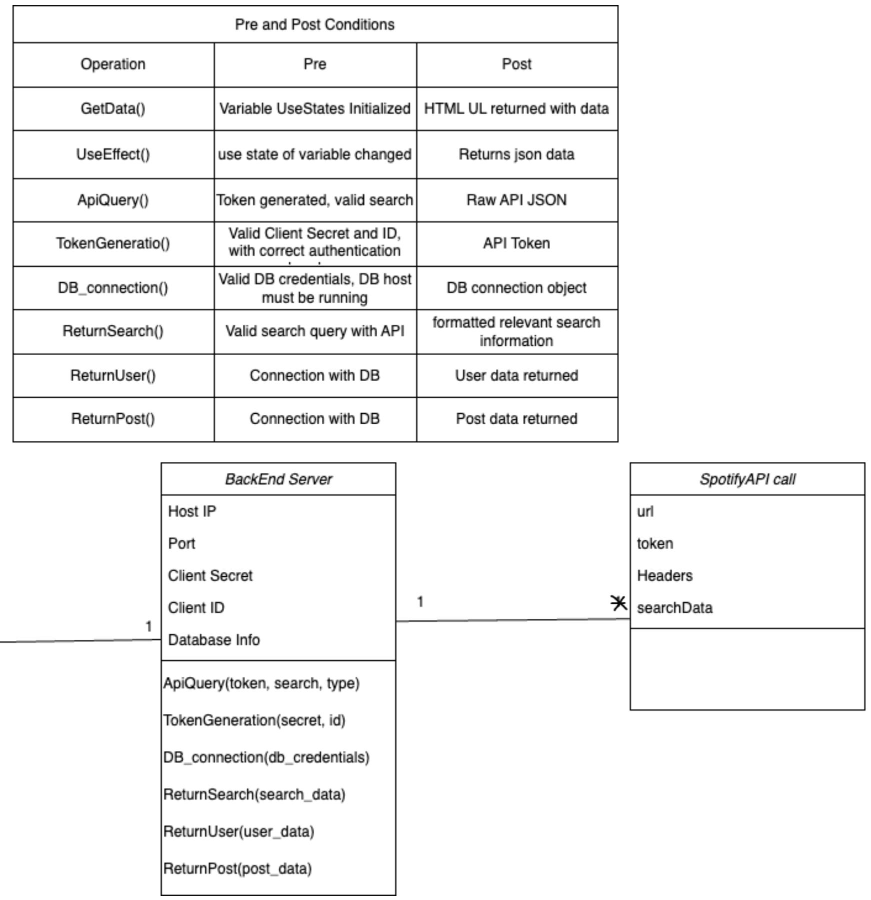
Figure 13: Layered architecture model

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 13, 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 14.1

The class diagram and table in Figure 14.1 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.

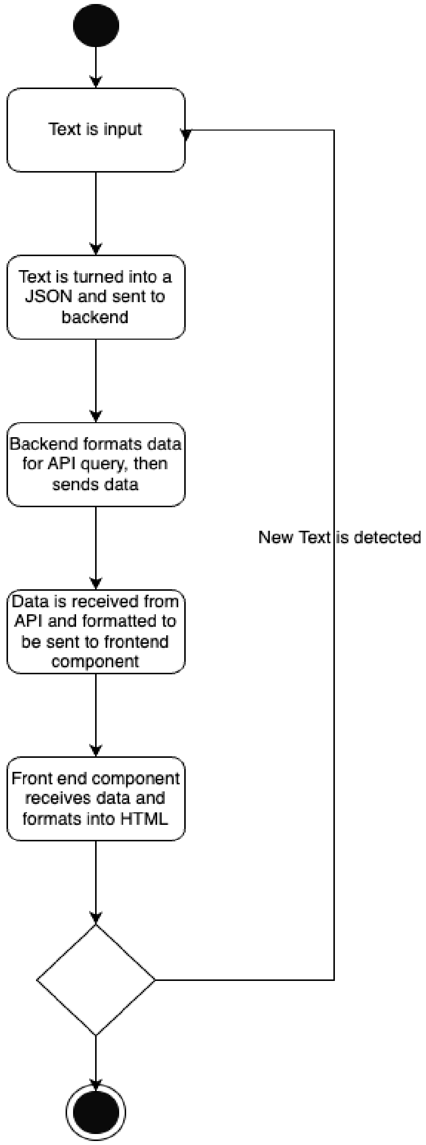

Text is input
Text is turned into a JSON and sent to backend
Backend formats data for API query, then sends data
Data is received from API and formatted to be sent to frontend component
Front end component receives data and formats into HTML
New Text is detected

Figure 14.2

In the sequence diagram shown in Figure 14.2, the linear system of data transfer across the three layers can be seen. This cycle will continue until the form is submitted.

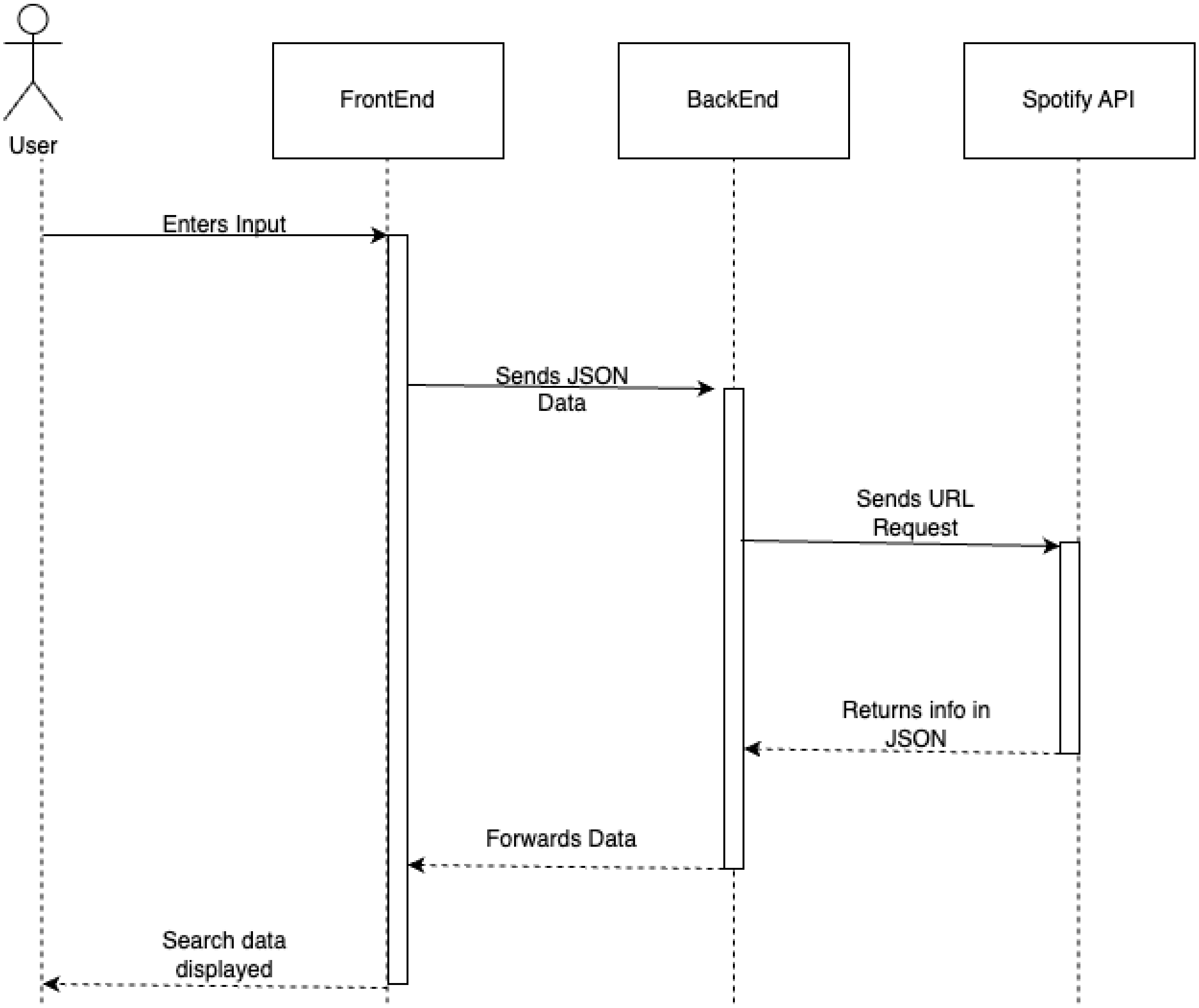


Figure 14.3

In Figure 14.3, 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.

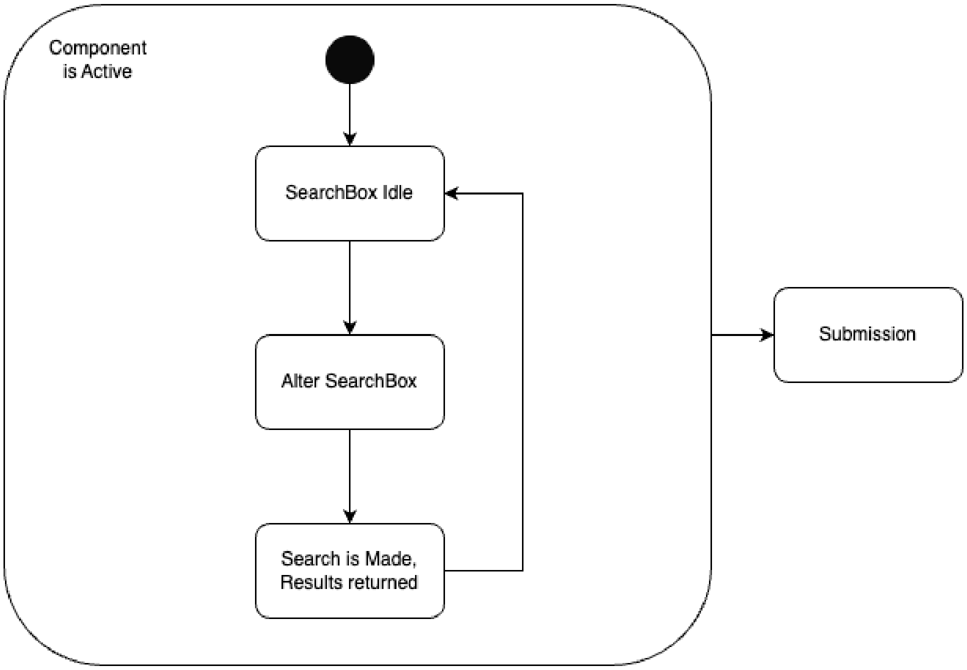
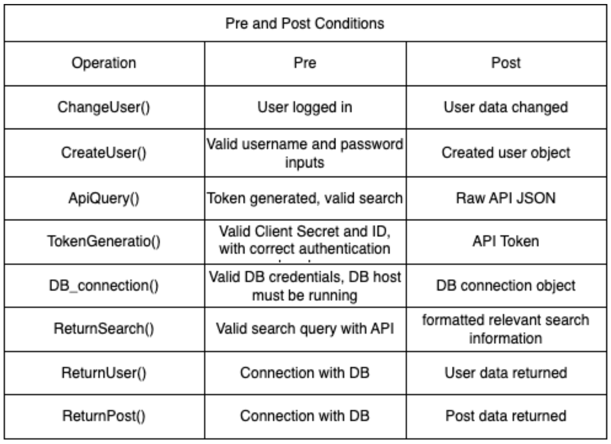


Figure 14.4

The cyclical nature of the search component can be seen in 14.4, 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.



Pre and Post Conditions		
Operation	Pre	Post
ChangeUser()	User logged in	User data changed
CreateUser()	Valid username and password inputs	Created user object
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

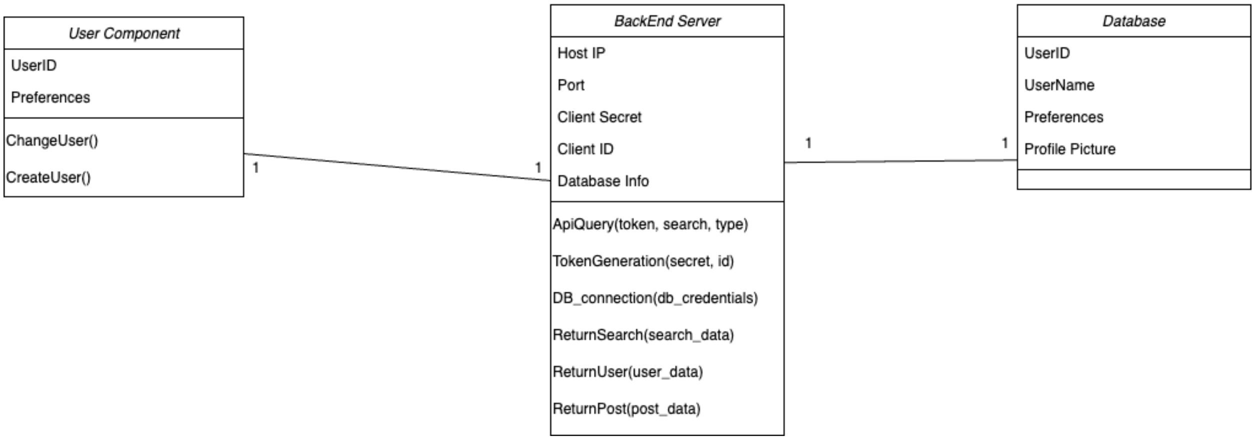


Figure 15.1

The class diagram in Figure 15.1 shows how the three layers work together and how their functions facilitate the transfer of data. Most of the functions are present in the backend server as that's where most of the computations occur, while the database doesn't innately return any information to the backend, so it has no methods. The results of a SQL query being returned is the result of the python package SQLAlchemy.

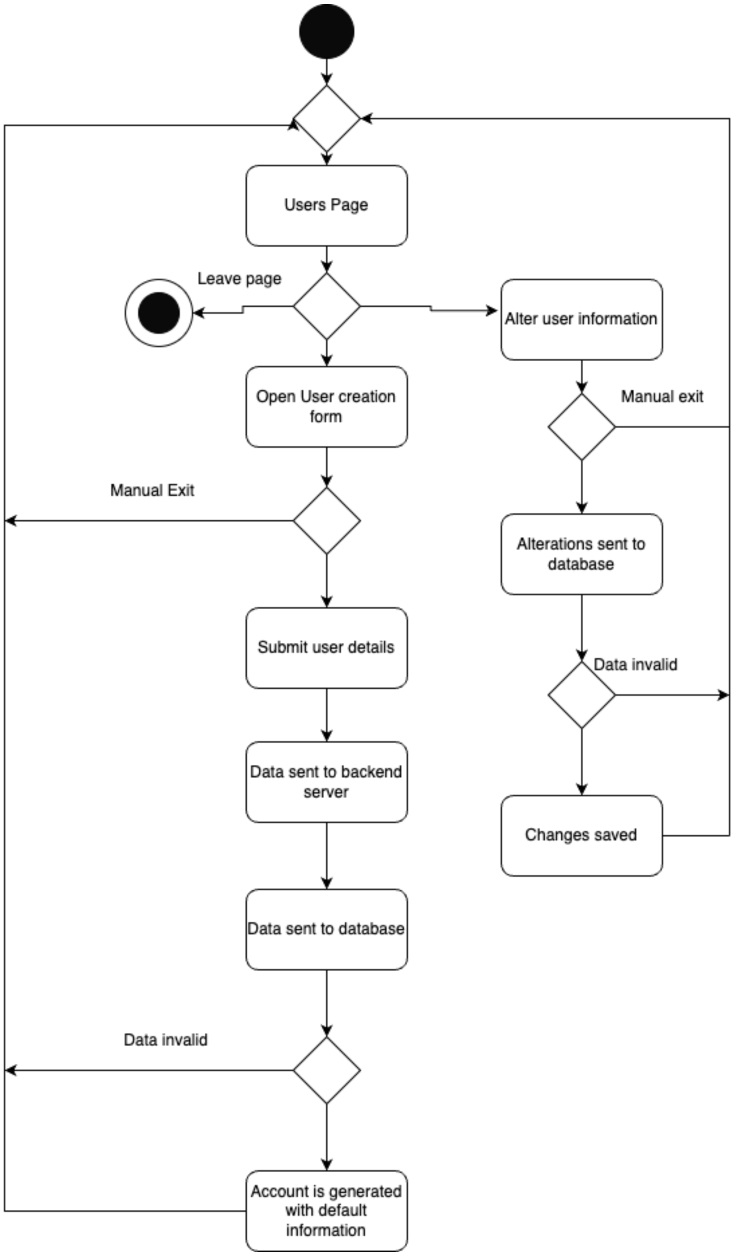


Figure 15.2

In Figure 15.2, 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.

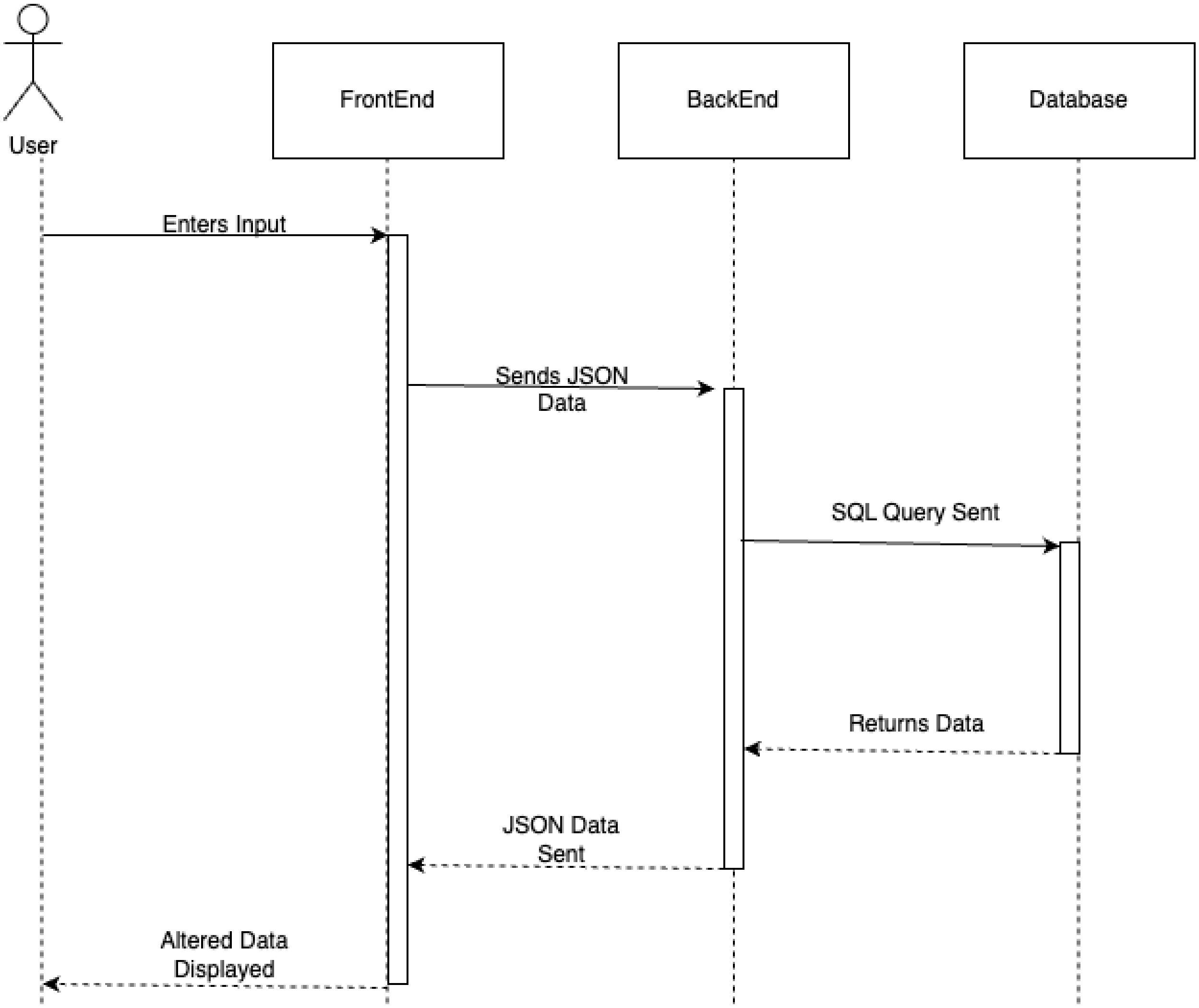


Figure 15.3

Figure 15.3 shows how through abstraction, the user input travels down the layers to reach the database.

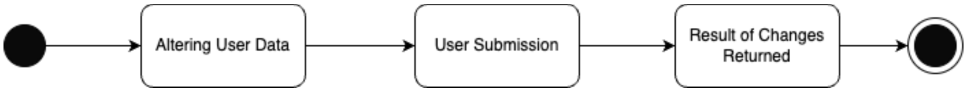
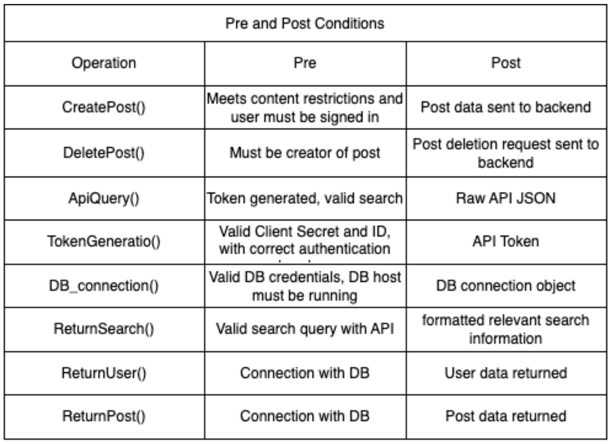


Figure 15.4

There are few states for user data alteration, as the process is linear. In Figure 14.4 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.



Pre and Post Conditions		
Operation	Pre	Post
CreatePost()	Meets content restrictions and user must be signed in	Post data sent to backend
DeletePost()	Must be creator of post	Post deletion request sent to backend
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

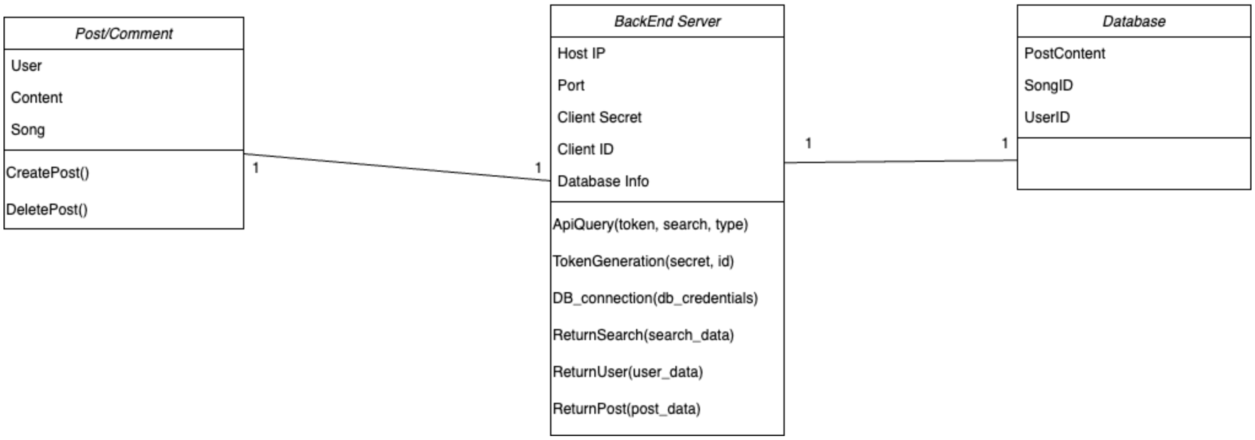


Figure 16.1

The process of creating posts is extremely similar to that of working with user data, as both follow a similar path with the end goal of altering the database. In Figure 16.1 there are only a few changes to the properties and methods of the classes from Figure 15.1. The reasoning behind its structure is the same as that listed in Figure 15.1.

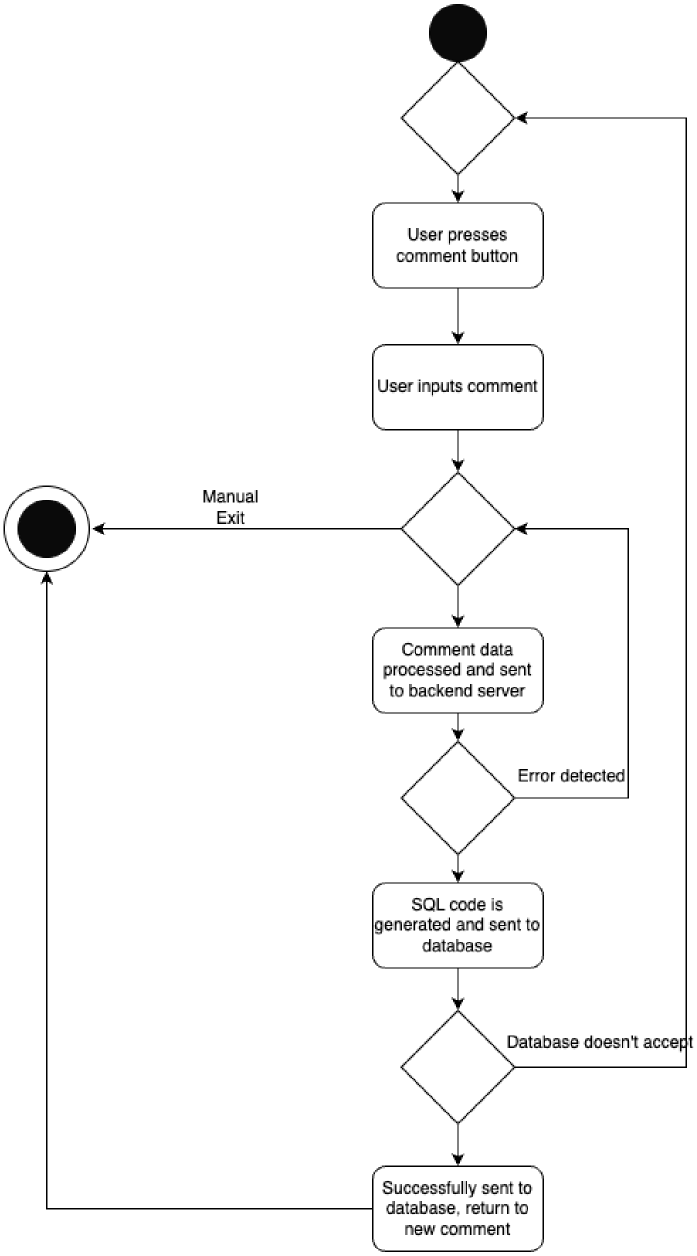

User presses comment button
User inputs comment
Manual Exit
Comment data processed and sent to backend server
Error detected
SQL code is generated and sent to database
Database doesn't accept
Successfully sent to database, return to new comment

Figure 16.2

The process of creating a post is linear, the only cycle appearing if an error occurs. In Figure 16.2 from its shape alone you can see this linear nature.

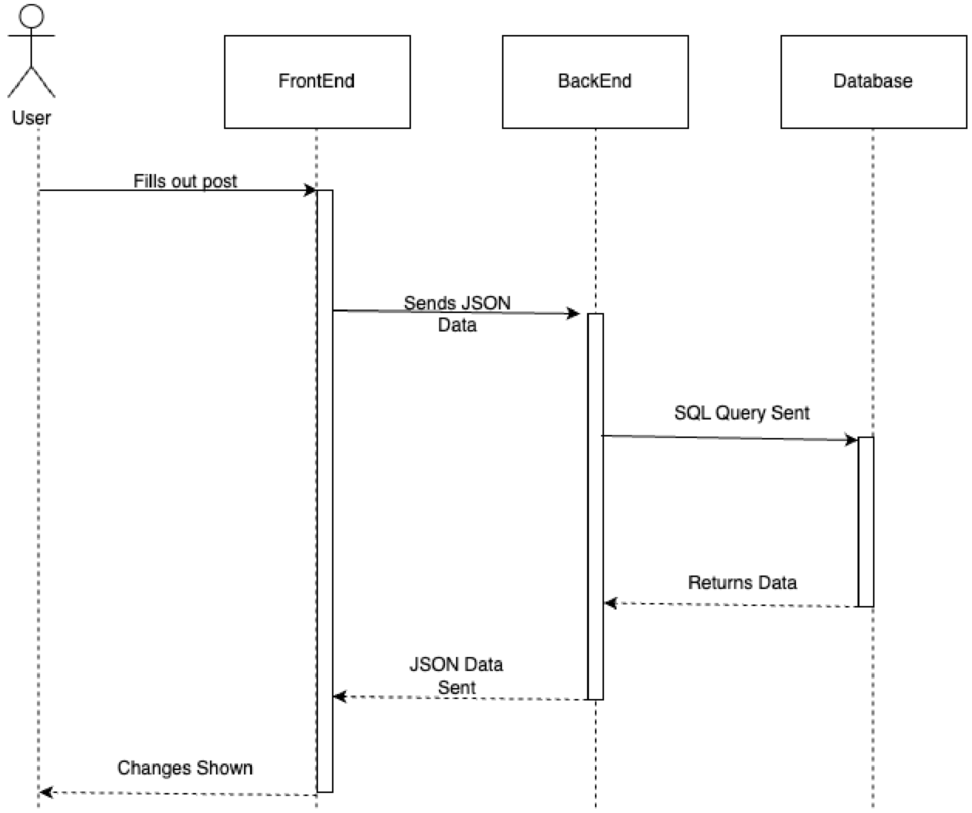


Figure 16.3

As is the case with Figure 14.3 and Figure 15.3, Figure 16.3 also involves a process that works its way down the layers to reach the data layer and returns information.

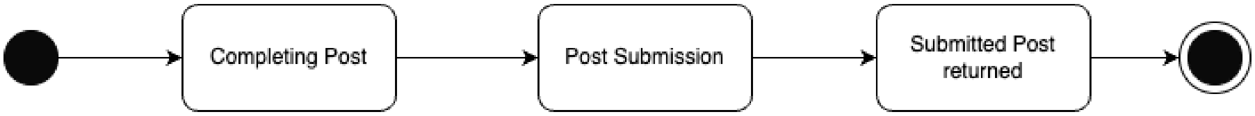


Figure 16.4

The 3 states visible to the user as shown in Figure 16.4, the process is simple and linear. Identical in nature to Figure 15.4.

Nonfunctional Requirements

Ease of Use

The platform must be intuitive and straightforward to navigate, ensuring users can easily access features such as music playback, tagging, direct messaging, and stats tracking without confusion.

How It Will Be Addressed:

- **User Interface (UI) Design**: The UI will be simple, clean, and modern, with clear call-to-action buttons and easy-to-understand layouts. For example, music controls (play, pause, skip) will be prominently displayed, and user profile management will be simple to navigate.
- **Onboarding Process**: New users will have an easy-to-follow tutorial or guide upon sign-up, walking them through core features, such as tracking music, tagging, and posting thoughts.
- **Consistency**: The platform's navigation structure will follow a consistent flow, with easy access to key sections like personal stats, music discovery, and messaging.

How It Will Be Validated:

- **User Testing**: Perform usability tests with a diverse group of users to ensure they can complete tasks such as signing up, searching for music, and posting thoughts without assistance.
- **Surveys/Feedback**: After using the app, gather user feedback regarding the ease of use, particularly on how easily they can navigate the platform and use key features.
- **Analytics**: Monitor usage patterns to see if users are completing key actions (like posting thoughts or interacting with music playback) without abandoning the flow.

Security

PlayBack must secure user data, including personal information and listening habits, and ensure private communications are protected.

How It Will Be Addressed:

- **Authentication**: Secure user login will be handled through mySQL hashing. Passwords will be encrypted, and sessions will expire after inactivity.
- **Data Encryption**: All user data (including thoughts, tags, ratings, and private messages) will be encrypted both in transit (using HTTPS) and at rest (using database-level encryption).

- **Role-based Access Control**: Only authenticated users will be able to perform actions such as posting thoughts, tagging music, and sending direct messages.
- **Privacy Controls**: Users will be able to set their posts and personal data visibility, allowing them to keep certain information or posts private.

How It Will Be Validated:

- **Penetration Testing**: Conduct security assessments and penetration testing to identify vulnerabilities in user authentication, data storage, and data transmission.
- **User Privacy Reports**: Provide users with transparency regarding how their data is stored and used and periodically update them on security measures and breaches (if any).

Performance/Efficiency

The platform must load quickly and perform smoothly, even with multiple users interacting simultaneously. Users should experience minimal lag when streaming music, navigating the site, or posting content.

How It Will Be Addressed:

- **Optimized Music Playback**: By integrating Spotify's API, music will be streamed directly from their servers, ensuring fast and efficient delivery. PlayBack will minimize delays between clicking a song and starting playback by caching frequently accessed songs.
- **Fast Page Loads**: Implement lazy loading, where only critical elements of the page load first (e.g., the navigation bar, music player), followed by less essential components (e.g., user stats, profile customizations).
- **Efficient Database Queries**: Use indexing and query optimization to handle user data efficiently, ensuring minimal load times for fetching listening habits, stats, and thoughts.
- **Scalable Infrastructure**: The backend infrastructure will be scalable, using services like mySQL, ensuring that it can handle increasing user traffic without degradation in performance.

How It Will Be Validated:

- **Load Testing**: Conduct load testing with varying numbers of concurrent users to assess whether the platform remains responsive when usage spikes (e.g., during peak times).
- **Response Time Monitoring**: Track response times for key actions (e.g., loading pages, posting thoughts, streaming music) and ensure they remain within acceptable limits.
- **User Feedback**: Gather performance-related feedback from users to identify any perceived lag or slowdowns during interaction with the platform, especially during peak usage.

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.

Analysis Process

The requirements analysis process for PlayBack involved gathering, organizing, and defining the system’s key functionalities to ensure a clear development roadmap. The process began with Panzer II drafting the **Introduction**, which established the purpose, scope, and objectives of the platform. This required thorough research into existing music engagement tools to highlight PlayBack's unique approach—providing a personal, uninfluenced way to reflect on music.

Following the introduction, **Use Case Models** were developed to outline the interactions between users and the system. This involved identifying key user roles, such as standard users and administrators, and mapping out their interactions with PlayBack's core features. Each use case was documented with descriptions of user actions, system responses, and alternative flows to ensure a comprehensive understanding of how the system would function.

The process included reviewing industry standards for system modeling, using UML diagrams to visually represent use cases, and iterating on the analysis based on peer and instructor feedback. All documentation was compiled and refined to ensure clarity and completeness before finalizing the report.

Major Problems Encountered

One of the challenges in the introduction phase was ensuring the scope was neither too broad nor too restrictive. Since PlayBack is a unique music engagement platform, it was difficult to precisely define what should and shouldn't be included without overcomplicating the system.

Some planned features, like private ratings and manual tracking, raised questions about their feasibility and necessity. This led to multiple revisions in defining how users would interact with these features.

Feedback from peers and instructors often required revising sections of the report, leading to time constraints. Some diagrams had to be redone to correct inconsistencies or better reflect system interactions.

Project Testing and Validation

General Testing Approach

Our testing approach for PlayBack follows a multi-layered strategy to ensure the platform's functionality, security, and reliability. We will use 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 will focus on verifying the platform’s external behavior, ensuring smooth user interactions, accurate search results, and proper music playback through the Spotify API. On the backend, white-box testing will be employed to validate PHP logic, database queries, and security measures, safeguarding against vulnerabilities like SQL injection and ensuring data integrity. Finally, end-to-end (E2E) testing will simulate real-world user scenarios to confirm that all components—from registration to music tagging—function cohesively. This comprehensive testing approach will help 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
Valid Registration	1. Go to registration page. 2. Enter valid username, email, and password. 3. Click **Sign Up**.	User account is created, and a success message is displayed.
Invalid Registration (Empty Fields)	1. Leave one or more fields blank. 2. Click **Sign Up**.	Error message indicates required fields.
Login with Valid Credentials	1. Go to login page. 2. Enter valid username and password. 3. Click **Login**.	User is redirected to their profile page.
Login with Invalid Credentials	1. Enter incorrect username or password. 2. Click **Login**.	Error message indicates invalid credentials.
Session Management	1. Login. 2. Close the browser. 3. Reopen and access the site.	User is still logged in if "Remember Me" was checked.

Music Playback (Spotify API)

Objective: Verify that users can search, play, and control music through Spotify.

Test Case	Steps	Expected Result
Music Search Functionality	1. Enter a valid song/artist in the search bar. 2. Click **Search**.	Matching results are displayed.
Invalid Search	1. Enter random text (nonsense characters). 2. Click **Search**.	"No results found" message appears.
Music Playback	1. Select a song from the search results. 2. Click **Play**.	The song starts playing with correct metadata.
Pause and Resume Playback	1. Play a song. 2. Click **Pause**. 3. Click **Play** again.	Playback pauses and resumes successfully.
Playback Error Handling	1. Disconnect from the internet. 2. Attempt to play a song.	Error message: "Unable to connect" appears.

Music Search and Tagging

Objective: Verify that users can search for music and apply tags (vibes).

Test Case	Steps	Expected Result
Valid Music Search	1. Enter a valid song/artist. 2. Click **Search**.	Matching results appear.
Invalid Music Search	1. Enter gibberish. 2. Click **Search**.	"No results found" message is displayed.
Add a Tag (Vibe)	1. Select a song. 2. Click **Add Vibe**. 3. Enter tag name and save.	Tag is applied and saved.
Remove a Tag	1. Select a song with an existing tag. 2. Click **Remove Tag**.	Tag is successfully removed.
Duplicate Tag Prevention	1. Add the same tag twice. 2. Click **Save**.	Error message: "Tag already exists".

Security and Data Validation

Objective: Ensure backend security and data integrity.

Test Case	Steps	Expected Result

SQL Injection Prevention	1. Enter ' OR 1=1 -- into login field. 2. Click **Login**.	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.
Password Hashing	1. Register a new user. 2. Check database.	Password is stored as a hashed value.
Session Timeout	1. Log in. 2. Stay inactive for X minutes. 3. Attempt to access a protected page.	User is automatically logged out.
Data Integrity Check	1. Create a new profile. 2. Manually edit backend data. 3. Refresh the profile.	Invalid data is not displayed.

User Profiles and Direct Messaging

Objective: Verify profile customization and direct messaging functionality.

Test Case	Steps	Expected Result
Profile Update	1. Go to **Profile Settings**. 2. Edit bio and save.	Changes are reflected on the profile.
Invalid Profile Input	1. Enter 500+ characters in bio. 2. Click **Save**.	Error message: "Character limit exceeded".
Send a DM	1. Select a user. 2. Type message and click **Send**.	Message is sent and appears in chat history.
Receive a DM	1. Have another user send a message. 2. Check inbox.	New message appears in the inbox.
DM Error Handling	1. Send DM while offline. 2. Check inbox.	Message is queued or error displayed.

End-to-End (E2E) Testing

Objective: Validate full workflows from start to finish.

Test Case	Steps	Expected Result

User Journey (Registration → Search → Playback → Tagging)	1. Register a new user. 2. Log in. 3. Search for music. 4. Play a song. 5. Tag it with a vibe.	Entire workflow functions without errors.
Profile + DM Integration	1. Log in. 2. Edit profile. 3. Send a DM to another user. 4. Verify they receive it.	Profile updates and DM system works as expected.
Error Handling Workflow	1. Log in. 2. Disconnect from the internet. 3. Attempt to search or message.	Proper error handling messages are displayed.

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