PlayBack - Requirements Engineering Report

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

Ryan Farrell – Software Architect, RJ Walsh – Quality Assurance Manager

University of North Carolina - Wilmington

February 2025

Introduction 3

- Description of Problem 3
- Project Scope & Objectives 5
- Success Criteria 7
- Purpose of Report 8

Software Project Plan 8

Resources 8
Workflow 9
Estimated Cost 10
Line of Code 10
Function Points 12
Figure 2.2: Weights for 14 General Characteristics of Project Table 13
Estimated Resources 13
Project Schedule 13
GANTT Chart 14
CPM or PERT Chart 15
Responsibility Matrix 15
Risk Management 16
Anticipated Issues and Constraints 17
Project Control 18
Requirements/Analysis Models 19
Use Case Diagrams 19
Activity Diagram 23
Requirements Class Models 23
Requirements/Data Dictionary 23
Nonfunctional Requirements 25
Ease of Use 25
Security 26
Performance/Efficiency 26
Tools 27
Analysis Process 29
Major Problems Encountered 29
Bibliography 30

Introduction

Description of Problem

Traditional music review websites have become substitutions for personal engagement with music. Pitchfork, for example, is the most predominant and widely used music review site with more than 7 million monthly listeners (Nast, 2024). The first thing users see on a review for any album on Pitchfork's website is a numeric rating next to the cover of the album. While this provides a quick reference for how the project was received, it can also shape how users feel about a project before they even read the review.

The reviews themselves tend to be lengthy, analytical, and occasionally pretentious. While this style appeals to those familiar with industry jargon and technical aspects of music, it can feel intimidating and inaccessible to casual listeners. Additionally, these types of reviews often prioritize technical analysis and comparisons to artists' previous works or the works of their peers, rather than fostering a personal connection with the music itself.

Community-driven review sites like Album of the Year (AOTY) and Rate Your Music (RYM), shift the focus toward user perspectives but still fall into similar trappings. Namely, prominently displaying a numeric score. While AOTY displays an aggregate score from both critics and user reviews as opposed to a singular critic, it still influences perceptions before listeners engage with the music themselves. RYM, while entirely user-driven, presents reviews in a dense, forum board style, leaving users with an endless ocean of opinions before they form their own ideas.

Meanwhile, music streaming platforms like Apple Music and Spotify dominate the industry, with Spotify holding the title of the most popular platform globally, boasting over 350 million users and 150 million subscribers. Apple Music, while trailing behind in overall user count, leads in foreign markets (Curry, 2024).

Music recommendation algorithms on these platforms rely heavily on users' past listening data, which can trap listeners in an echo chamber, limiting their exposure to new genres and artists. This reduces the opportunity for discovery, as recommendations tend to reinforce existing preferences rather than expand them.

Researchers have done important work in analyzing and improving recommendation algorithms to mitigate what is known as a 'filter bubble.' According to Merriam-Webster, a filter bubble is an environment—especially online—where users are exposed only to information and opinions that align with their existing beliefs (Definition of FILTER BUBBLE, 2024). Despite these efforts, today's music landscape still produces 'personalized outcomes that exceed user expectations, thereby exposing them to the pitfalls of filter bubbles' (Luo et al., 2024).

Additionally, if users listen to music across multiple platforms, their listening data is fragmented, leading to an incomplete picture of their music engagement. This issue is even more pronounced when music is played outside of any tracking platform—such as in a friend's car or at a listening party—resulting in gaps in a user's listening history that algorithms fail to capture.

Music listeners today don't have 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. Community-driven platforms, while more focused on user opinions, still rely on aggregate scores that shape expectations. 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.

On the other hand, streaming services like Spotify and Apple Music use algorithms that reinforce familiar listening patterns, limiting discovery. 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.

There's 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.

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.

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"). 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.

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

- 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

Workflow

Items are dependent on completion of sub-bulleted points before their own completion

- 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

Estimated Cost

Line of Code

THINGS TO CONSIDER

- Most of our time will likely be spent on learning how to use the tool, not writing code
- These estimates come from scarce sources and forum posts and should be treated with skepticism.

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

Figure 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

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

Figure 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

Figure 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. 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

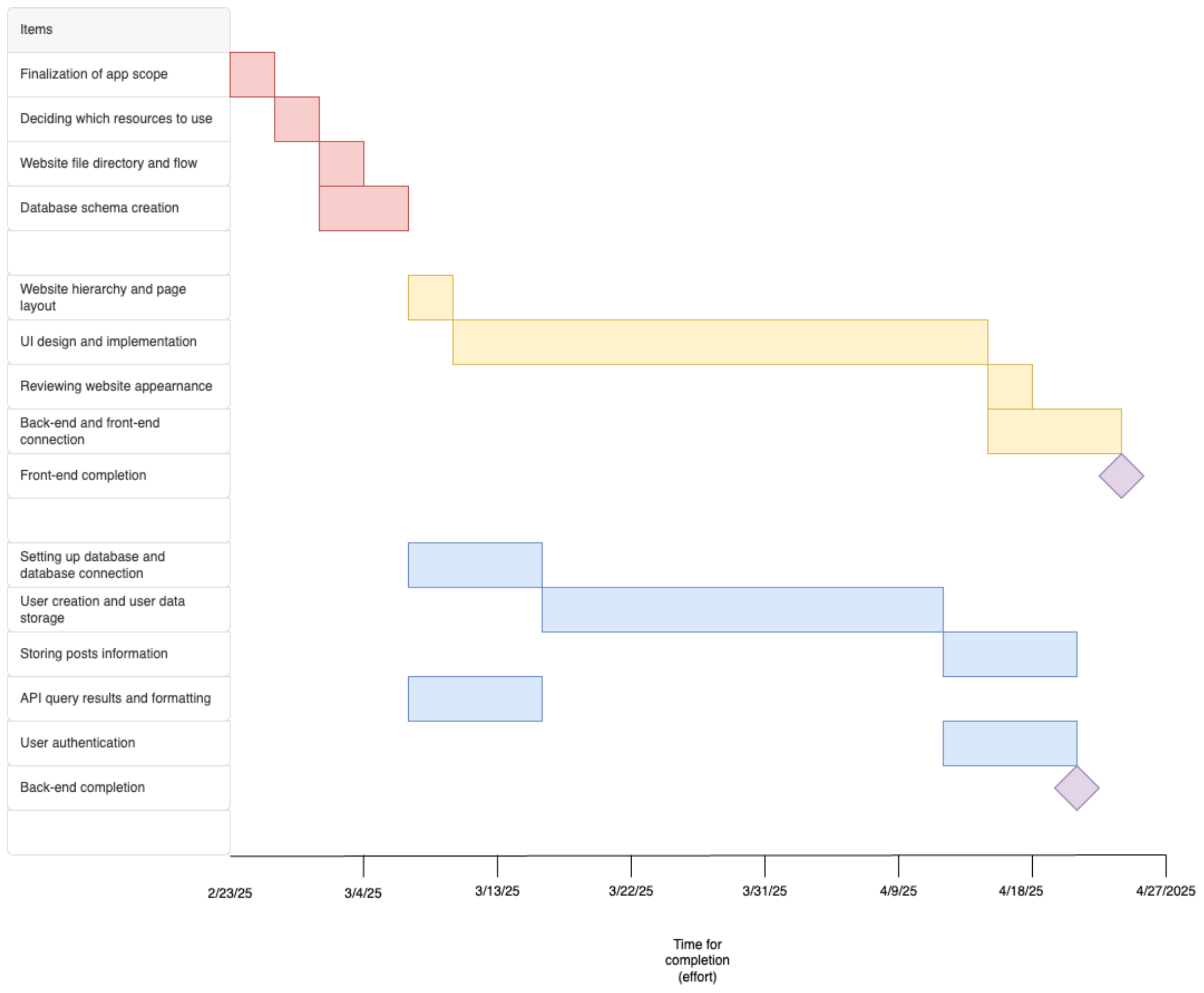


Figure 3: Gantt Chart for Project Schedule

CPM or PERT Chart

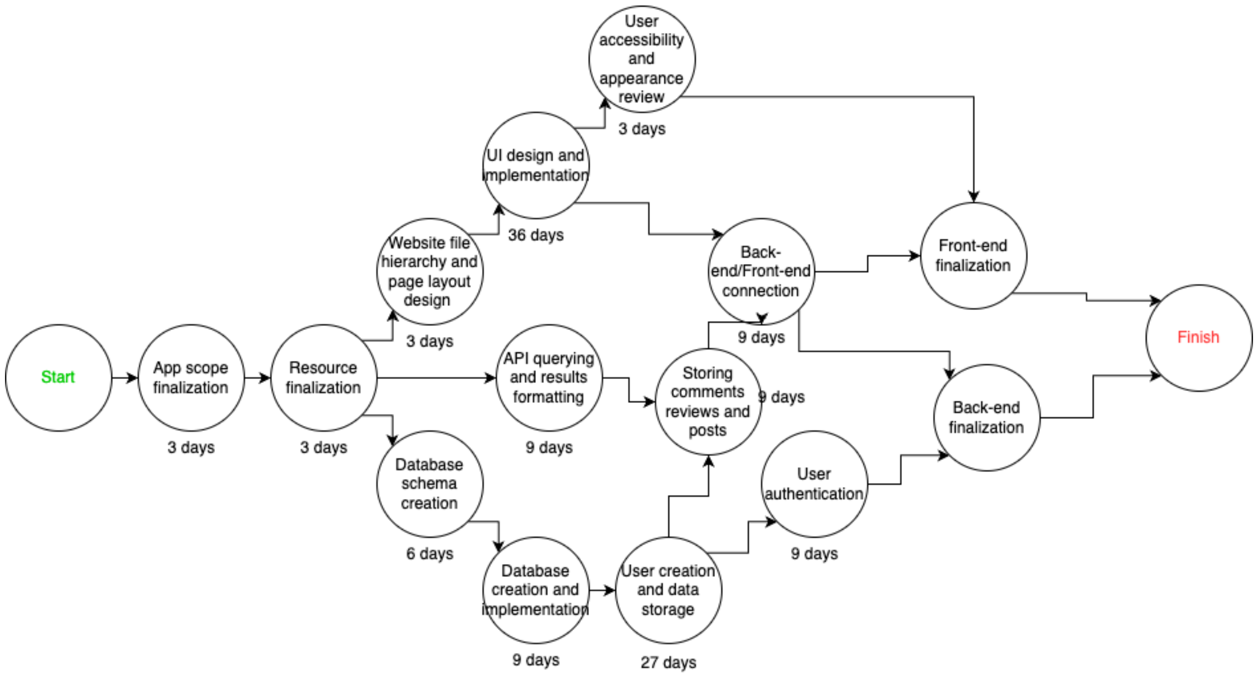
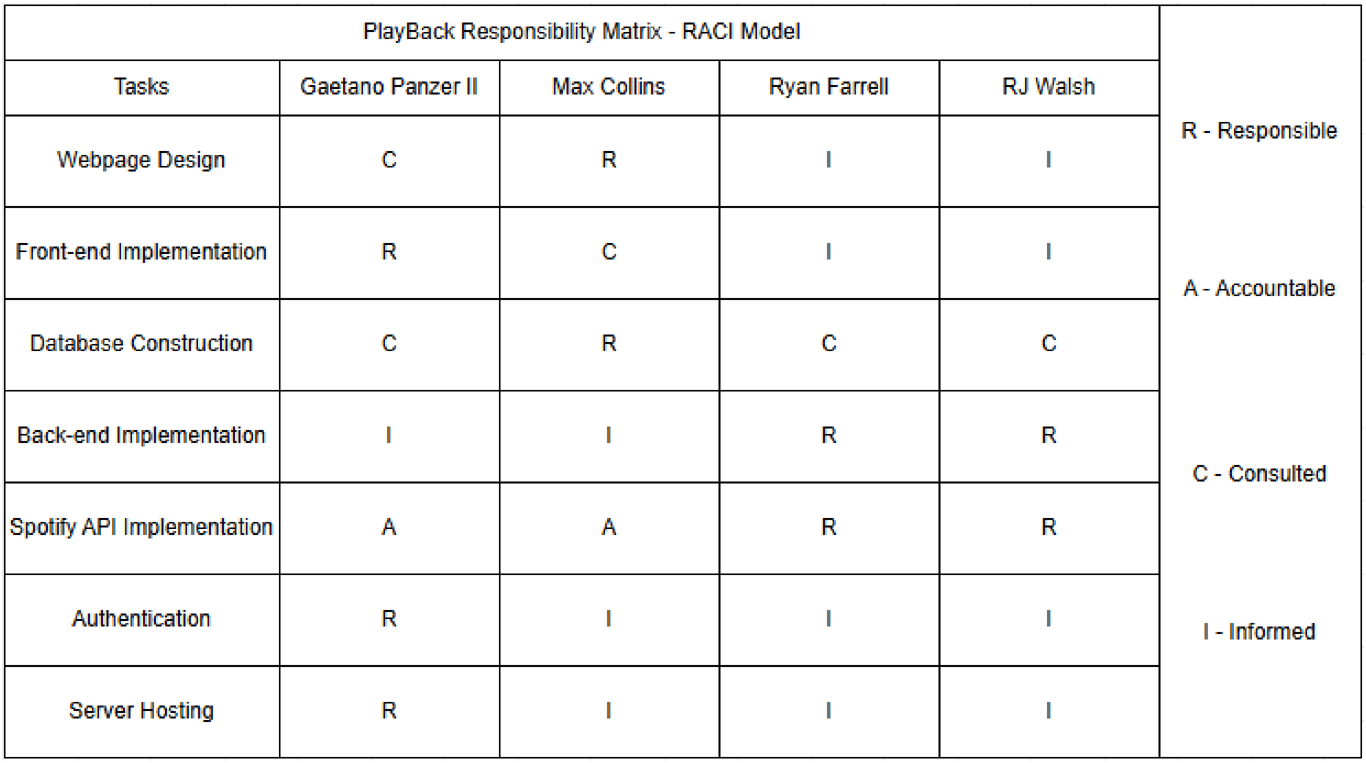

Start
App scope finalization
3 days
Resource finalization
3 days
Website file hierarchy and page layout design
3 days
UI design and implementation
36 days
User accessibility and appearance review
3 days
API querying and results formatting
9 days
Database schema creation
6 days
Database creation and implementation
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 4: 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, with the following diagram representing an early rendition of each team member's responsibilities.



PlayBack Responsibility Matrix - RACI Model

Tasks	Gaetano Panzer II	Max Collins	Ryan Farrell	RJ Walsh
Webpage Design	C	R	I	I
Front-end Implementation	R	C	I	I
Database Construction	C	R	C	C
Back-end Implementation	I	I	R	R
Spotify API Implementation	A	A	R	R
Authentication	R	I	I	I
Server Hosting	R	I	I	I

R - Responsible

A - Accountable

C - Consulted

I - Informed

Figure 5: Responsibility Matrix

Risk Management

Below is a table outlining 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

Figure 6: 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. Extensive documentation and previous examples of work done with components, such as Spotify API and secure user login with Google 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. 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.

Project Control

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

- GitHub
    - GitHub will be the main form of version control used during development
- Discord
    - Discord will be used as a form of voice and text communication between team members.
    - Members will send messages asking questions and updating team members on development progress
    - When members of the team cannot meet in person, they will be able to use Discord's voice chat features to meet online and discuss and work together on development
- Microsoft Word
    - Microsoft Word will be used to create text documents that contain important information regarding the development of PlayBack, including ideas for the scope of the software, and the project's requirements engineering 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

Use Case Diagrams

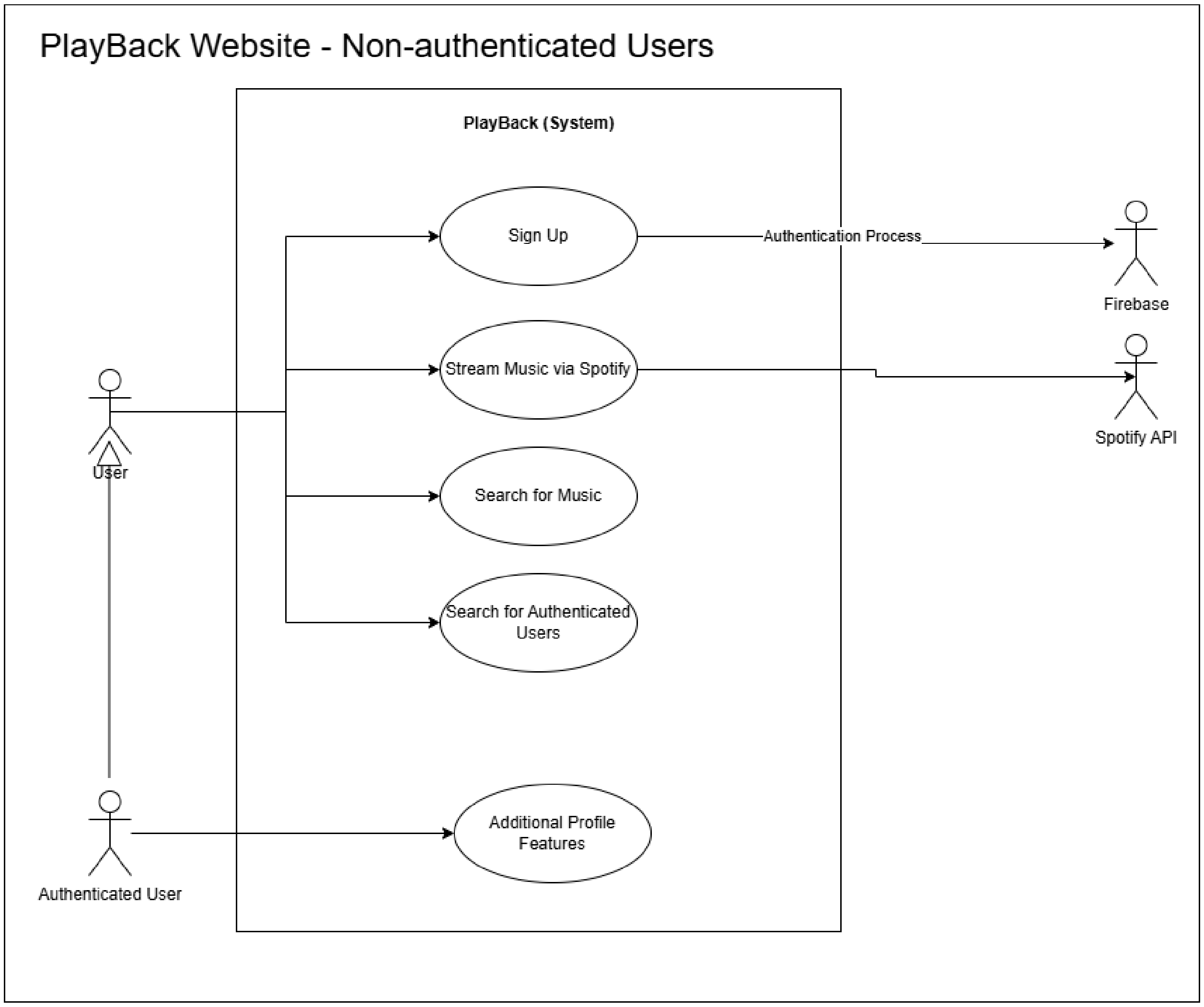


Figure 7: Use Case Diagram for Non-authenticated Users

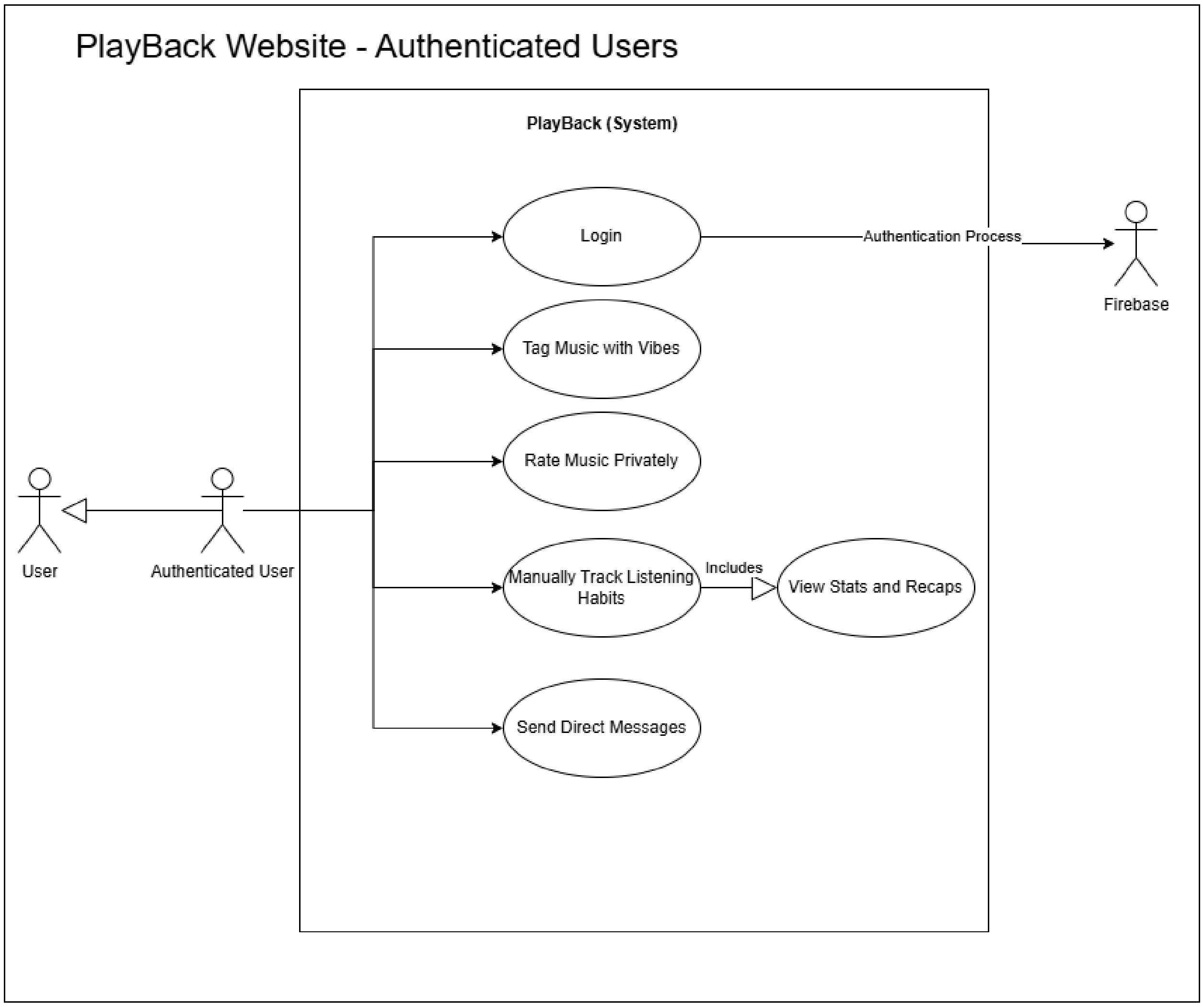


Figure 8: Use Case Diagram for Authenticated Users

Major Software Functions

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 Firebase for account creation.
  - **Login**: Links to Firebase 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.

Activity Diagram

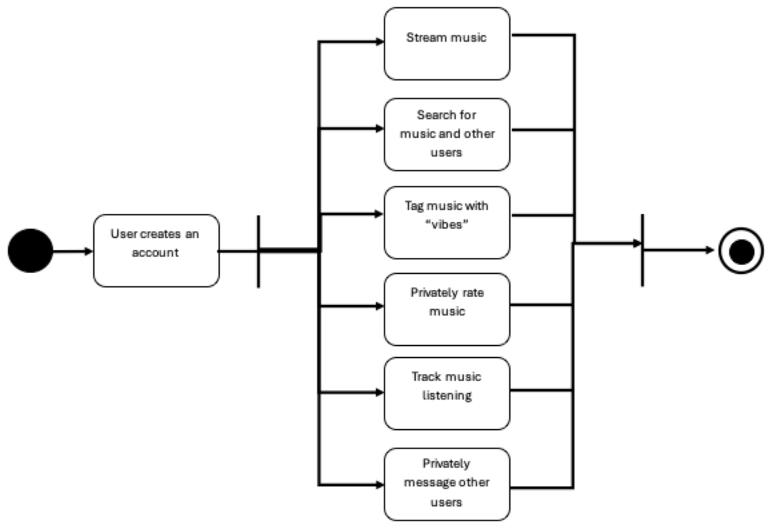
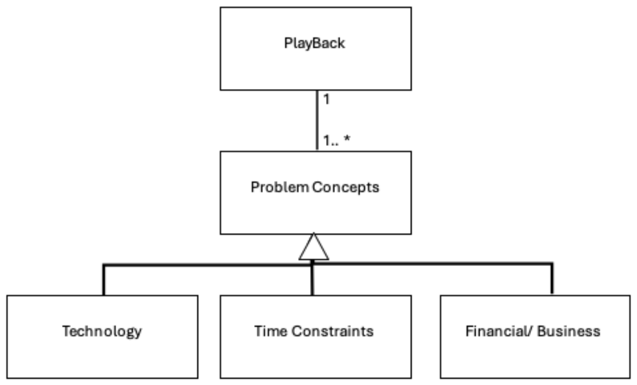


Figure 9: Activity Diagram

Requirements Class Models



Requirements/Data Dictionary

User Data

Data Name	Type	Data Format	Num. Of Bytes	Description	Example	Limitations
User Post	String	255 character string	256	Basic user post	“I like this song!”	Less than 255 characters.
User reply	String	255 character string	256	User reply to post	“You’re wrong!”	Less than 255 characters.

Data Name	Type	Data Format	Num. Of Bytes	Description	Example	Limitations
Genre Tags	String Array	6 element string array	~	List of song tags	Rock, rap...	Less than 6 elements
Song interactions	INT		4	Tracks engagement with a song	Posts: 10	

User Data

Data Name	Type	Data Format	Num. Of Bytes	Description	Example	Limitations
Username	String		20	User identifier	CatLover96	UTF characters, less than 20 characters, must be unique
User Password	String		20	User validation key	Password!	UTF characters, more than 5 characters, less than

						20 characters
Date Joined	Float	MM/DD/YYYY	4	Tracks when user joined	02/10/2025	
Post History	Array		~	Points to all user posts		

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 Firebase Authentication. 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 cloud-based services like Firebase, 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.

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

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*.

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

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.