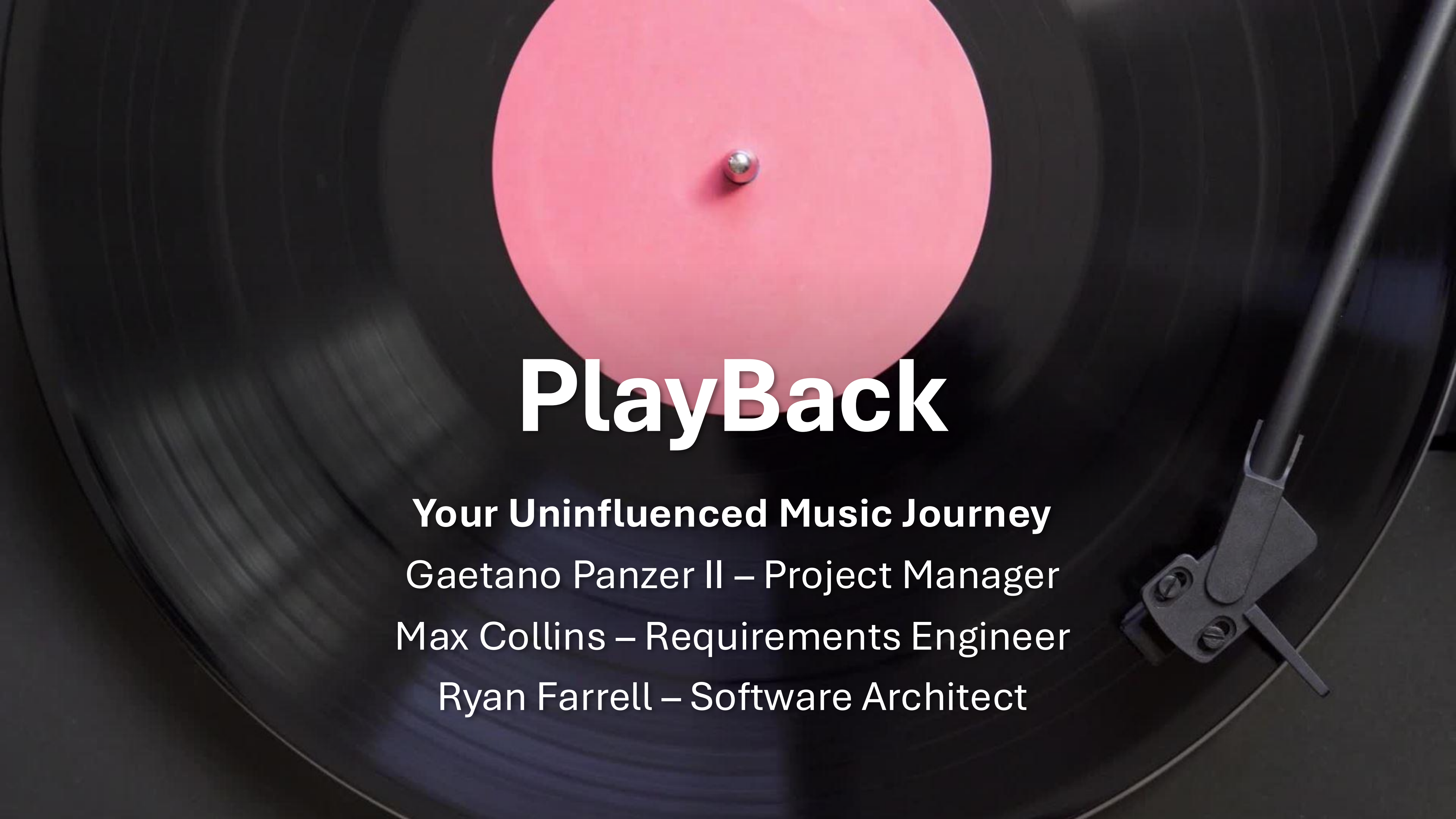


PlayBack

Your Uninfluenced Music Journey

Gaetano Panzer II – Project Manager

Max Collins – Requirements Engineer

Ryan Farrell – Software Architect

Problem Description

Numeric scores, dense reviews, and algorithmic recommendations

Consensus over personal experience

Fragmented listening data, limiting reflection on one’s musical journey

Objectives

Removes external influence: No public scores, no aggregate ratings

Short-form comments: Express genuine reactions without pressure

Private vibes: For personal reflection only

Manual tracking (Future Implementation: Users log plays to enable monthly/yearly taste recaps

Success Criteria

🔐 Core Account Features

Users can create secure accounts and log in/out.

Users can search for music with corresponding pages by type

Users can post comments (short-form reflections) about music.

🛠 Still in Development

Manual input of listening statistics

Monthly and yearly listening recaps.

Spotify API integration for track playback.

Success Criteria cont.

⚙ Technical Requirements

Website should load in < 3 seconds for 90% of users.

No critical bugs or crashes during testing phase.

Backend must securely store user data and post metadata.

Presentation Outline

- Team Member Responsibilities
- Requirements Models
- Design Models
- Implementation

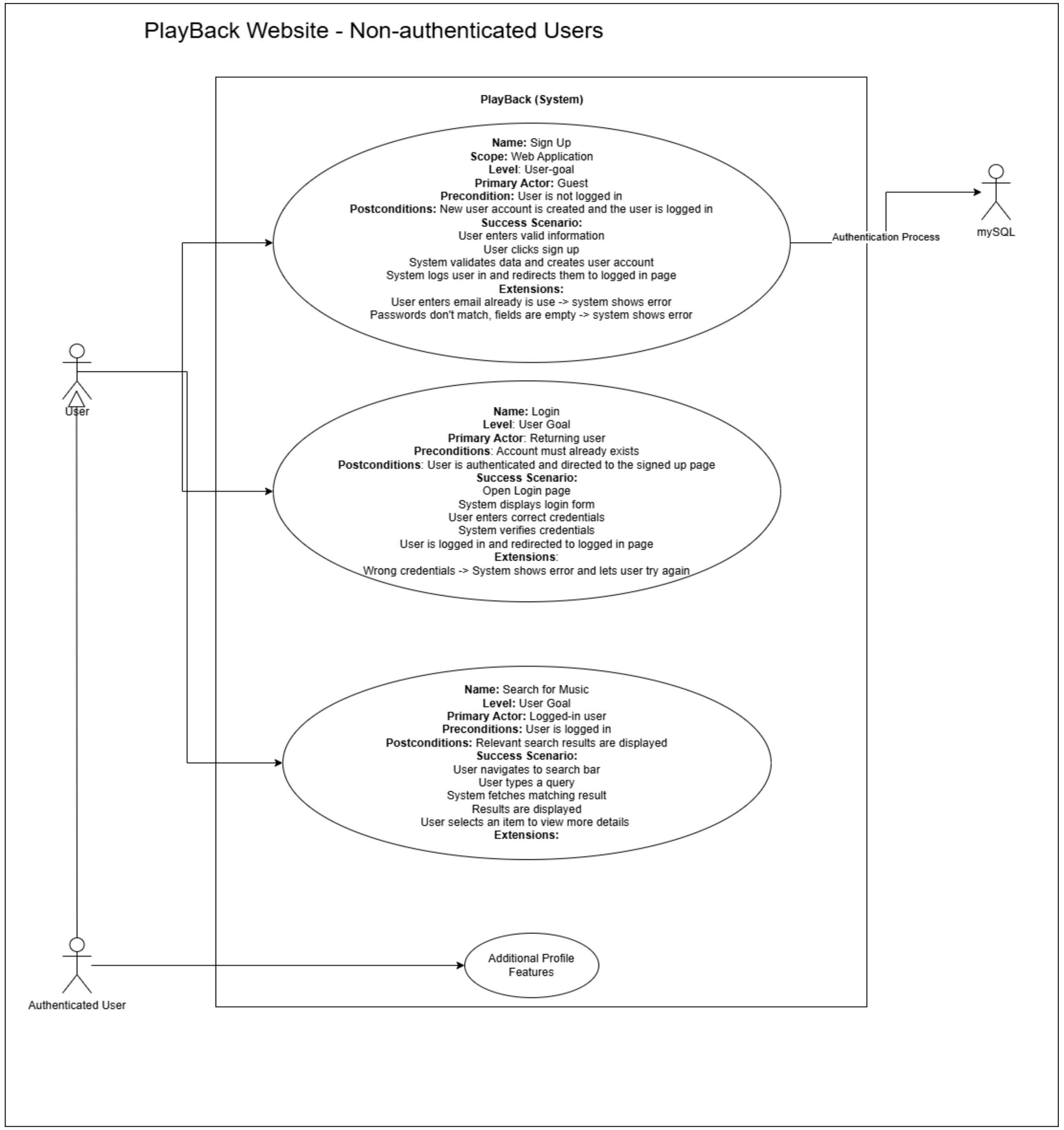
Team Member Responsibilities

GAETANO – LOGIN/SIGNUP FORMS, ROUTES, SESSIONS

RYAN – API ENDPOINTS, DATABASE, COMMENTS, VIBES, SEARCHING

MAX – WEBSITE DESIGN AND IMPLEMENTATION WITH CSS

Use Case Models

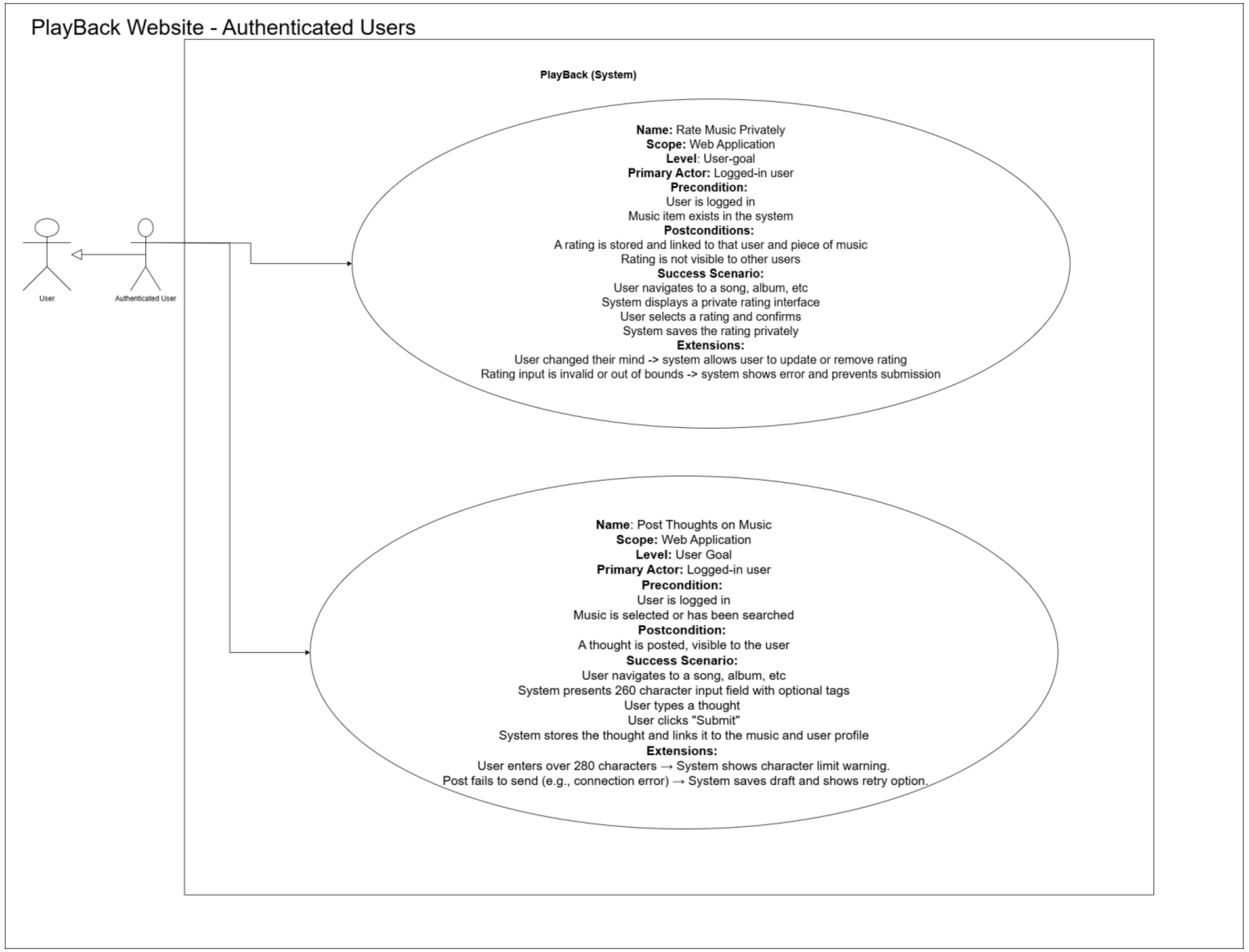

PlayBack Website - Non-authenticated Users
PlayBack (System)
User
Authenticated User
mySQL
Authentication Process
Additional Profile Features

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

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

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

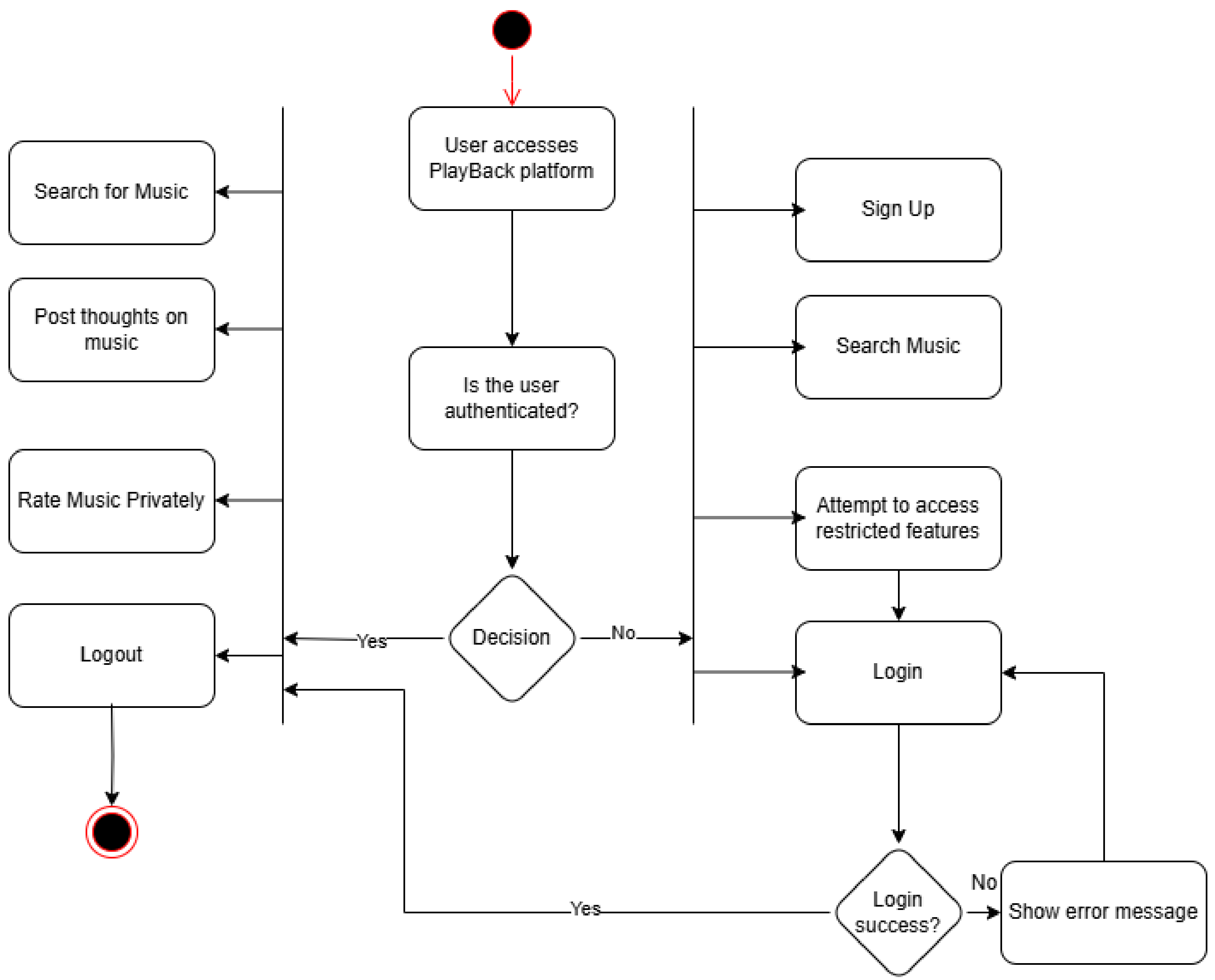
Use Case Models



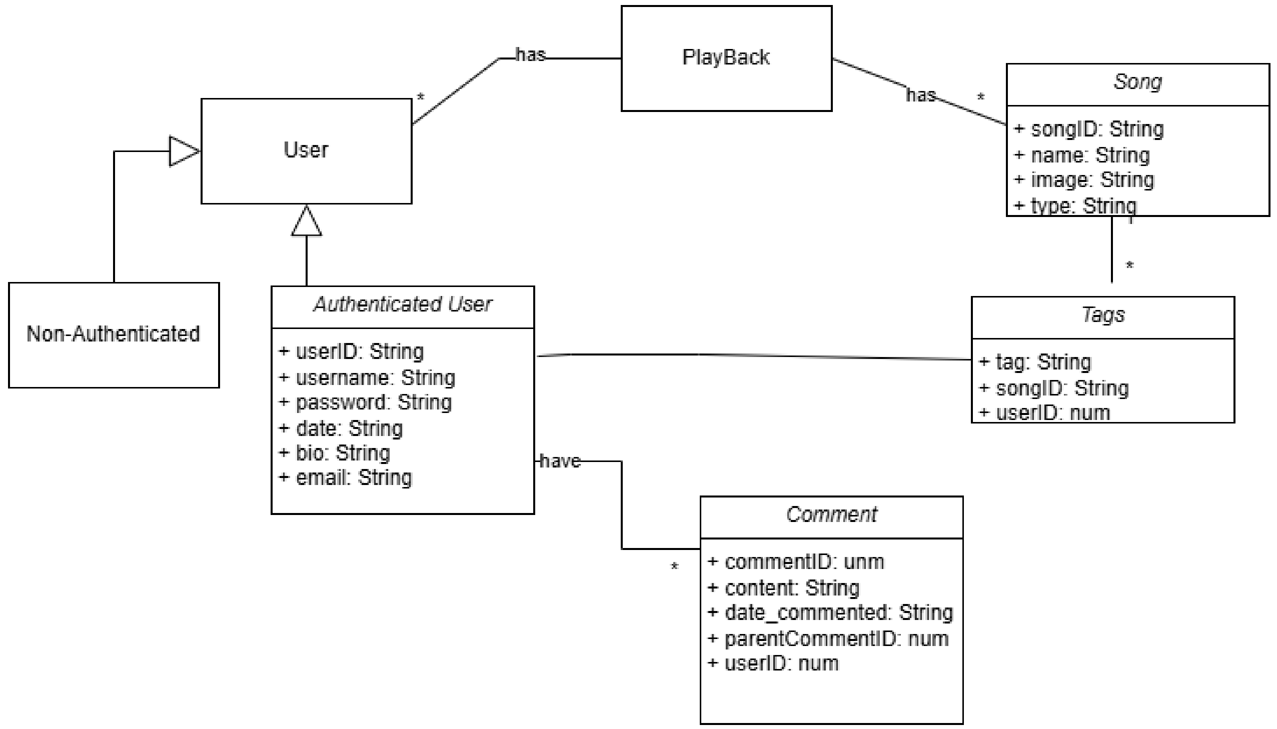
Name: Rate Music Privately
Scope: Web Application
Level: User-goal
Primary Actor: Logged-in user
Precondition:
User is logged in
Music item exists in the system
Postconditions:
A rating is stored and linked to that user and piece of music
Rating is not visible to other users
Success Scenario:
User navigates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

Name: Post Thoughts on Music
Scope: Web Application
Level: User Goal
Primary Actor: Logged-in user
Precondition:
User is logged in
Music is selected or has been searched
Postcondition:
A thought is posted, visible to the user
Success Scenario:
User navigates to a song, album, etc
System presents 260 character input field with optional tags
User types a thought
User clicks "Submit"
System stores the thought and links it to the music and user profile
Extensions:
User enters over 280 characters → System shows character limit warning.
Post fails to send (e.g., connection error) → System saves draft and shows retry option.

Activity Diagram

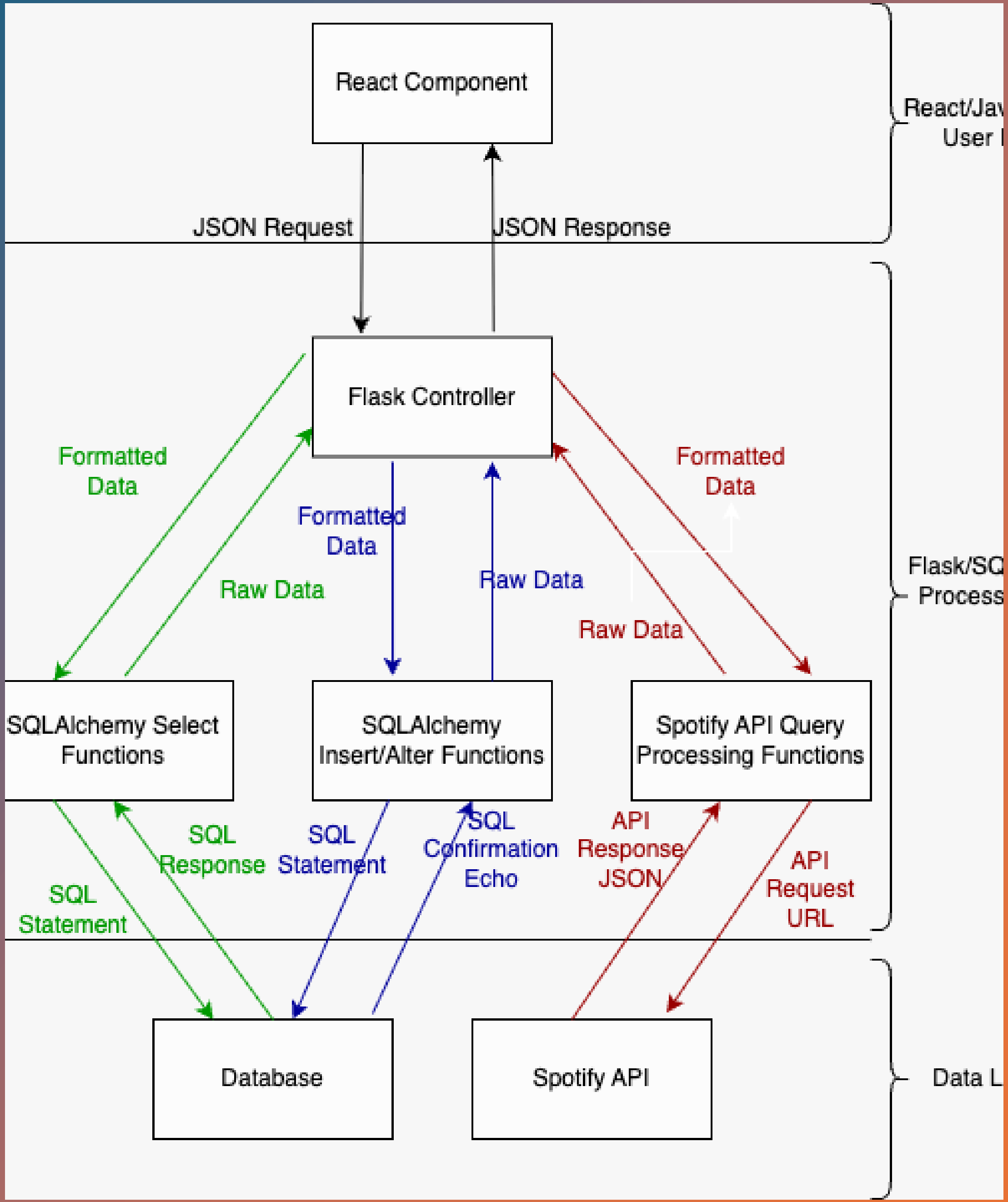


Class Model



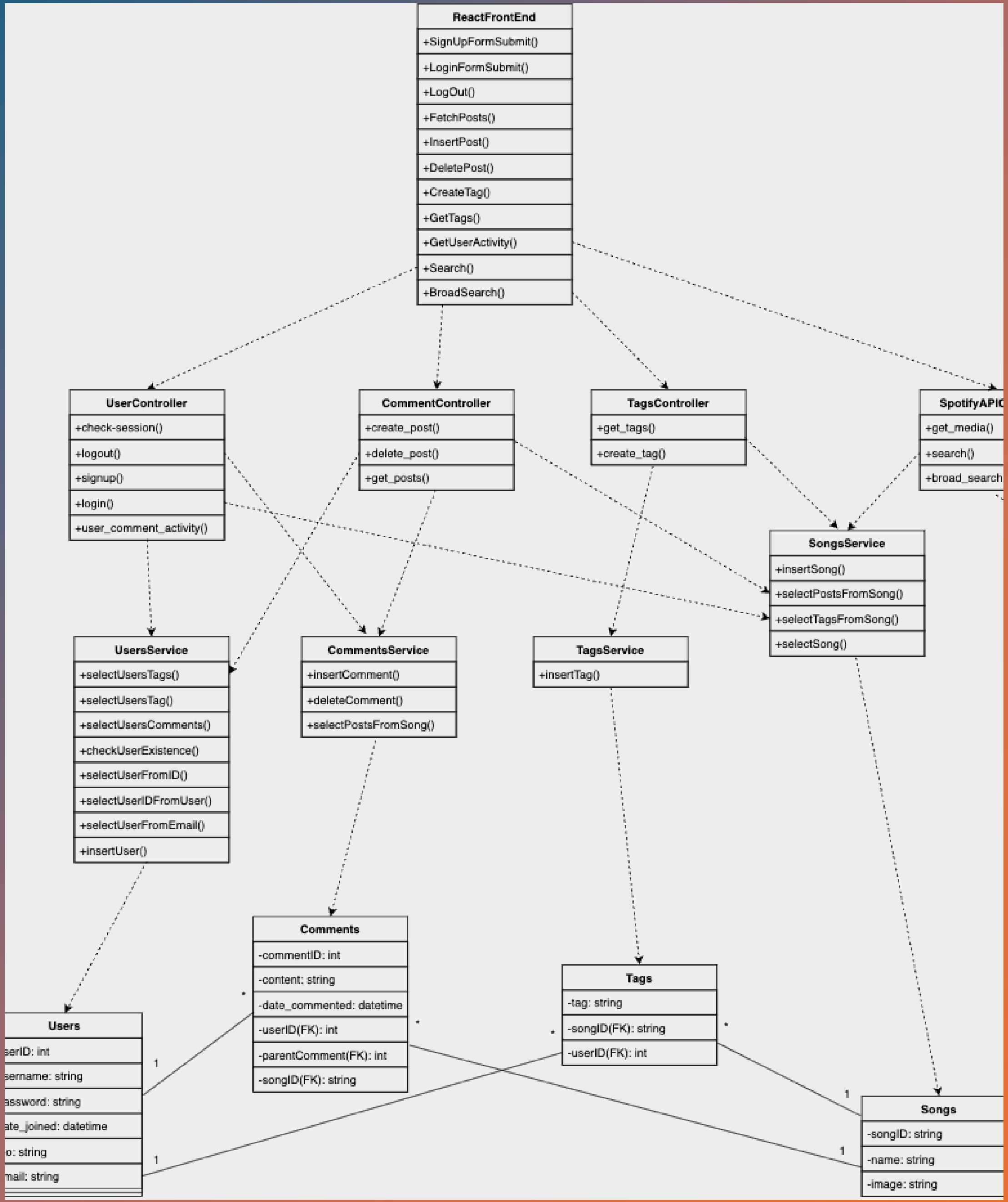
PlayBack
has
has
*
*
User
Song
+ songID: String
+ name: String
+ image: String
+ type: String
*
Non-Authenticated
Authenticated User
+ userID: String
+ username: String
+ password: String
+ date: String
+ bio: String
+ email: String
Tags
+ tag: String
+ songID: String
+ userID: num
have
*
Comment
+ commentID: unm
+ content: String
+ date_commented: String
+ parentCommentID: num
+ userID: num

Design Models



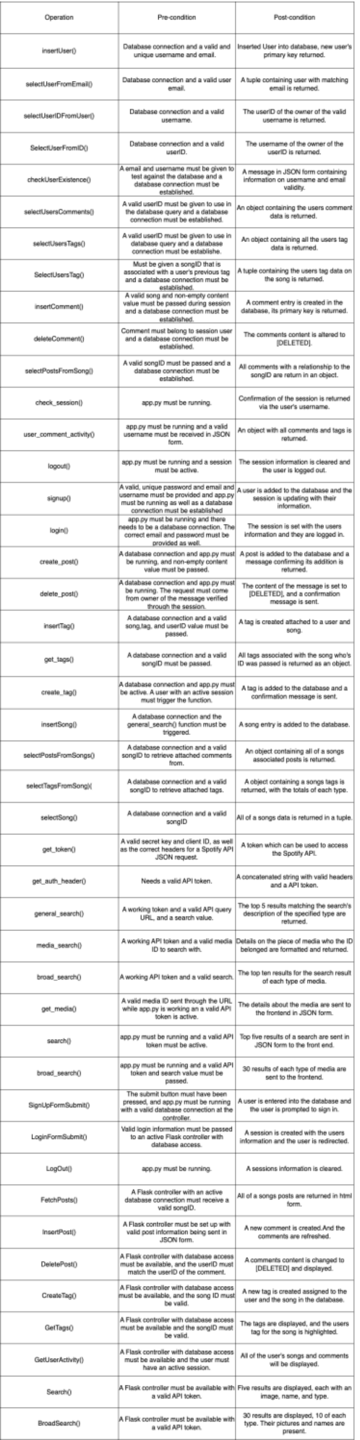
Architectural Design

- Layered Architecture
- Three-tiered
- Great for a component focused frontend.



Design Class Diagram

- Whole structure is built off the database values and Spotify API responses.
- Service classes structure and manipulate that data through functions. Keeps Flask routes clean.
- Controller classes
  - Receive requests from the frontend.
  - Parses request data.
  - Calls the service classes for logic.
  - Returns response to the frontend.
- The react frontend class takes user input and sends requests to the Controller classes.



Operation	Pre-condition	Post-condition
insertUser()	Database connection and a valid and unique username and email.	Inserted User into database, new user's primary key returned.
selectUserFromEmail()	Database connection and a valid user email.	A tuple containing user with matching email is returned.
selectUserIDFromUser()	Database connection and a valid username.	The userID of the owner of the valid username is returned.
SelectUserFromID()	Database connection and a valid userID.	The username of the owner of the userID is returned.
checkUserExistence()	A email and username must be given to test against the database and a database connection must be established.	A message in JSON form containing information on username and email validity.
selectUsersComments()	A valid userID must be given to use in the database query and a database connection must be established.	An object containing the users comment data is returned.
selectUsersTags()	A valid userID must be given to use in database query and a database connection must be establishe.	An object containing all the users tag data is returned.
SelectUsersTag()	Must be given a songID that is associated with a user's previous tag and a database connection must be established.	A tuple containing the users tag data on the song is returned.
insertComment()	A valid song and non-empty content value must be passed during session and a database connection must be established.	A comment entry is created in the database, its primary key is returned.
deleteComment()	Comment must belong to session user and a database connection must be established.	The comments content is altered to [DELETED].
selectPostsFromSong()	A valid songID must be passed and a database connection must be established.	All comments with a relationship to the songID are return in an object.
check_session()	app.py must be running.	Confirmation of the session is returned via the user's username.
user_comment_activity()	app.py must be running and a valid username must be received in JSON form.	An object with all comments and tags is returned.
logout()	app.py must be running and a session must be active.	The session information is cleared and the user is logged out.
signup()	A valid, unique password and email and username must be provided and app.py must be running as well as a database connection must be established	A user is added to the database and the session is updating with their information.
login()	app.py must be running and there needs to be a database connection. The correct email and password must be provided as well.	The session is set with the users information and they are logged in.
create_post()	A database connection and app.py must be running, and non-empty content value must be passed.	A post is added to the database and a message confirming its addition is returned.
delete_post()	A database connection and app.py must be running. The request must come from owner of the message verified through the session.	The content of the message is set to [DELETED], and a confirmation message is sent.
insertTag()	A database connection and a valid song,tag, and userID value must be passed.	A tag is created attached to a user and song.
get_tags()	A database connection and a valid songID must be passed.	All tags associated with the song who's ID was passed is returned as an object.
create_tag()	A database connection and app.py must be active. A user with an active session must trigger the function.	A tag is added to the database and a confirmation message is sent.
insertSong()	A database connection and the general_search() function must be triggered.	A song entry is added to the database.
selectPostsFromSongs()	A database connection and a valid songID to retrieve attached comments from.	An object containing all of a songs associated posts is returned.
selectTagsFromSong)(	A database connection and a valid songID to retrieve attached tags.	A object containing a songs tags is returned, with the totals of each type.
selectSong()	A database connection and a valid songID	All of a songs data is returned in a tuple.
get_token()	A valid secret key and client ID, as well as the correct headers for a Spotify API JSON request.	A token which can be used to access the Spotify API.
get_auth_header()	Needs a valid API token.	A concatenated string with valid headers and a API token.
general_search()	A working token and a valid API query URL, and a search value.	The top 5 results matching the search's description of the specified type are returned.
media_search()	A working API token and a valid media ID to search with.	Details on the piece of media who the ID belonged are formatted and returned.
broad_search()	A working API token and a valid search.	The top ten results for the search result of each type of media.
get_media()	A valid media ID sent through the URL while app.py is working an a valid API token is active.	The details about the media are sent to the frontend in JSON form.
search()	app.py must be running and a valid API token must be active.	Top five results of a search are sent in JSON form to the front end.
broad_search()	app.py must be running and a valid API token and search value must be passed.	30 results of each type of media are sent to the frontend.
SignUpFormSubmit()	The submit button must have been pressed, and app.py must be running with a valid database connection at the controller.	A user is entered into the database and the user is prompted to sign in.
LoginFormSubmit()	Valid login information must be passed to an active Flask controller with database access.	A session is created with the users information and the user is redirected.
LogOut()	app.py must be running.	A sessions information is cleared.
FetchPosts()	A Flask controller with an active database connection must receive a valid songID.	All of a songs posts are returned in html form.
InsertPost()	A Flask controller must be set up with valid post information being sent in JSON form.	A new comment is created.And the comments are refreshed.
DeletePost()	A Flask controller with database access must be available, and the userID must match the userID of the comment.	A comments content is changed to [DELETED] and displayed.
CreateTag()	A Flask controller with database access must be available, and the song ID must be valid.	A new tag is created assigned to the user and the song in the database.
GetTags()	A Flask controller with database access must be available and the songID must be valid.	The tags are displayed, and the users tag for the song is highlighted.
GetUserActivity()	A Flask controller with database access must be available and the user must have an active session.	All of the user's songs and comments will be displayed.
Search()	A Flask controller must be available with a valid API token.	Five results are displayed, each with an image, name, and type.
BroadSearch()	A Flask controller must be available with a valid API token.	30 results are displayed, 10 of each type. Their pictures and names are present.

Example pre and post conditions

Get_token()

Preconditions

- *Valid client secret and client ID.*
- *Valid Spotify API URL endpoint*

Postconditions

- Spotify API token is created and returned, valid for ~30 minutes.

General_search(token, search, type, limit)

Preconditions

- Up to date Spotify API token
- Type that is 'artist', 'track', or 'album'
- Search value that is not null
- A valid engine object connection to the database.

Postconditions

- A list of dictionaries containing n=limit elements, with media IDs, image data, media names, and more.
- N=limit new song entries in the database.

Search()

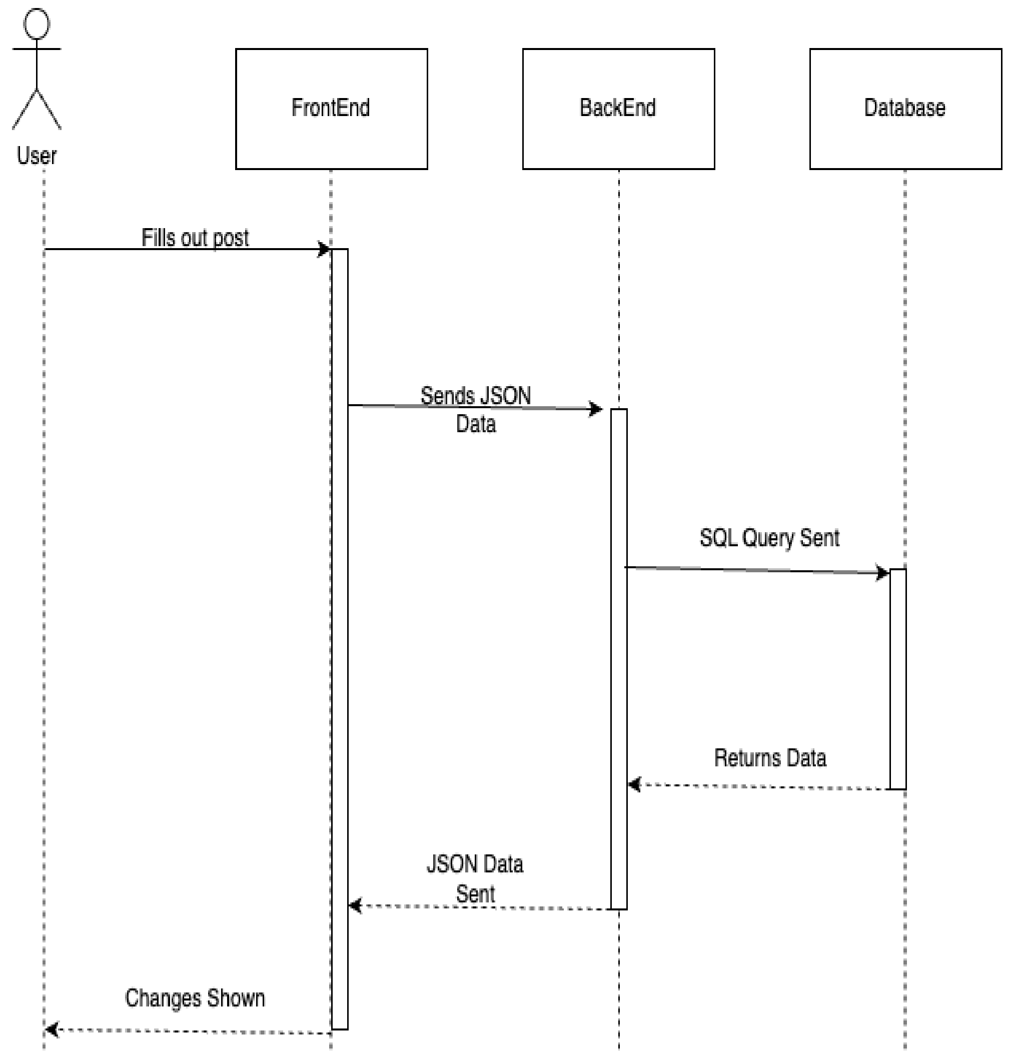
Preconditions

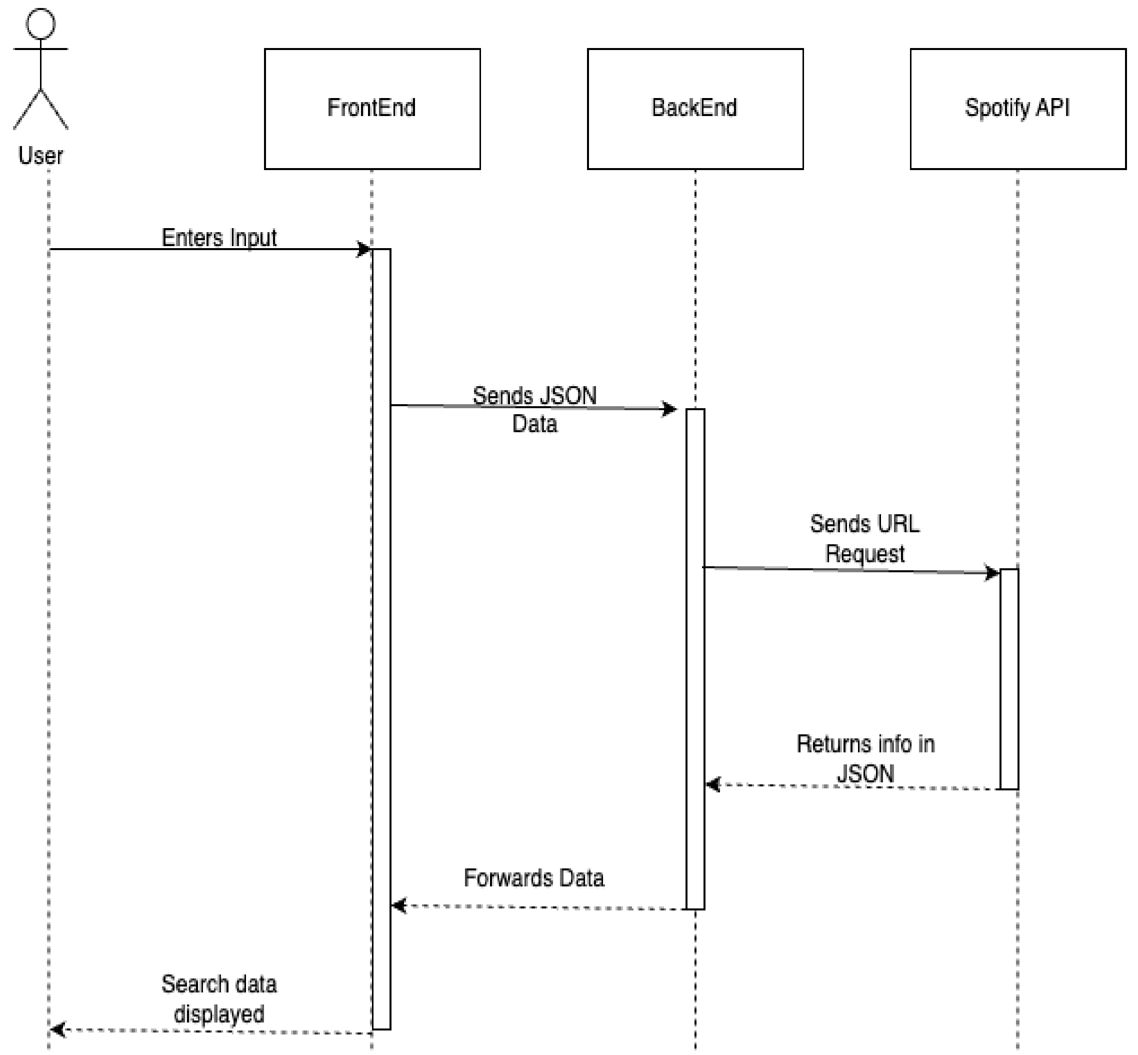
- Valid Spotify API token
- Non-null search value request from frontend
- App.py(backend server) running

Postconditions

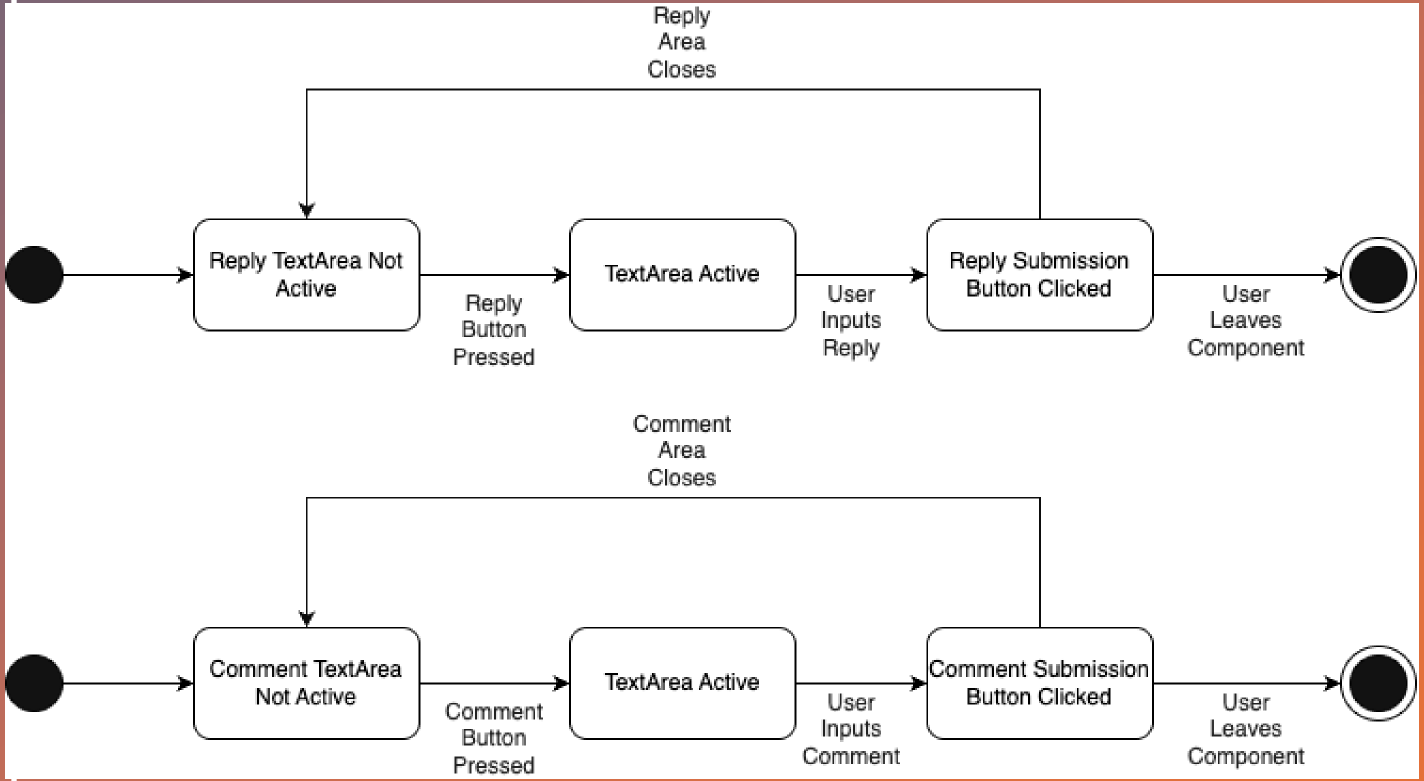
- Print message is executed to show that end point has been hit.
- List of dictionaries in JSON form sent to frontend component OR error message with error is sent.

Sequence Diagrams

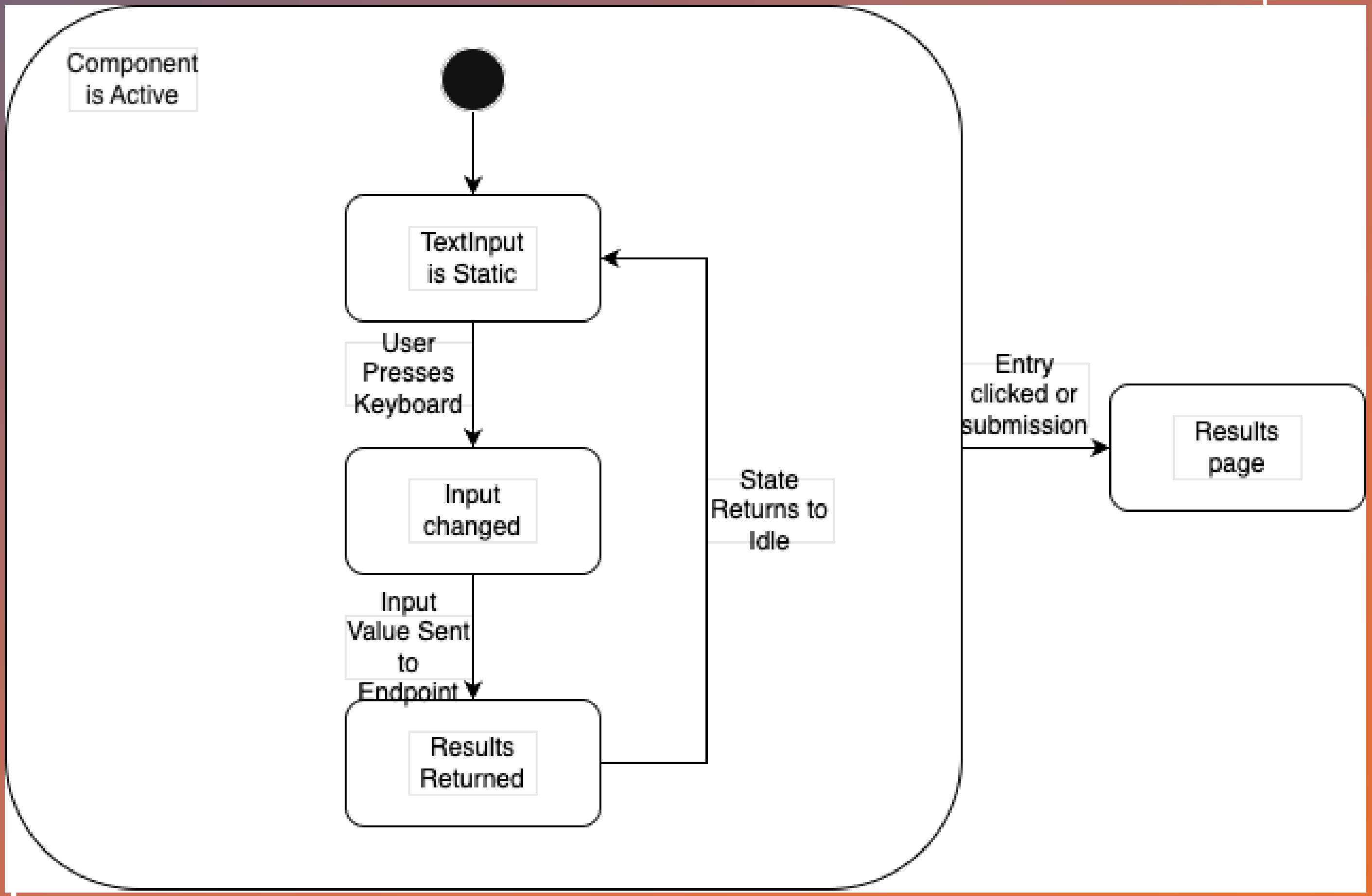




State Diagrams (Replying/ Commenting)



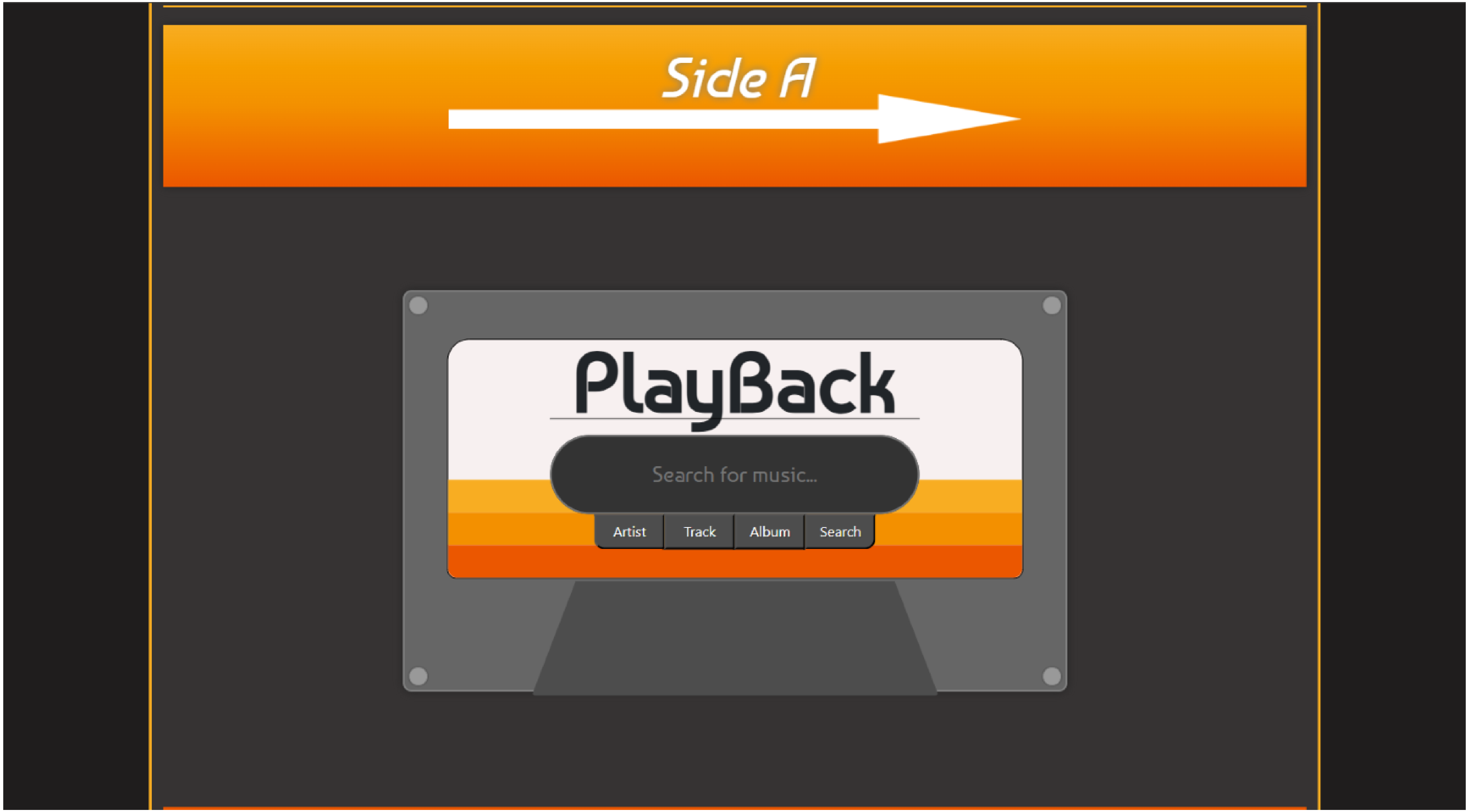
State Diagram (useEffect)

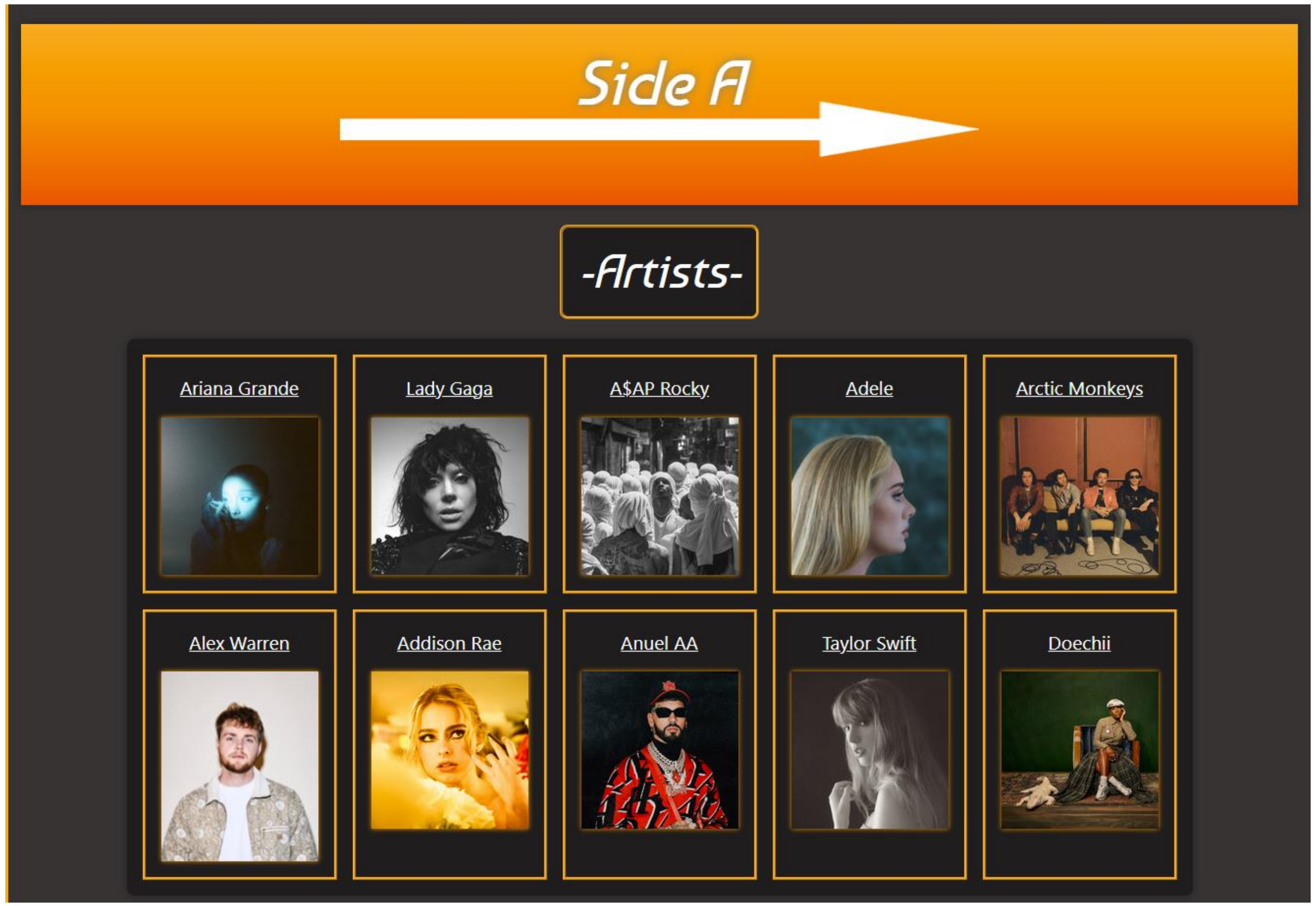


Implementation

User Interface

- Design inspired by cassette tapes
- Retro feel, sleeker look
- Warm colors and grays for pleasant viewing experience
- Sections on each page are clear and concise
- Examples on following slides

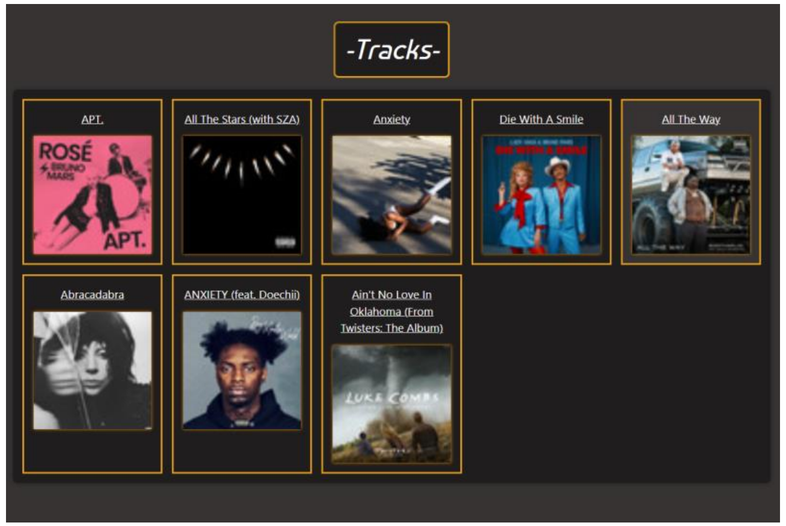

Side A
PlayBack
Search for music...
Artist
Track
Album
Search



Side A

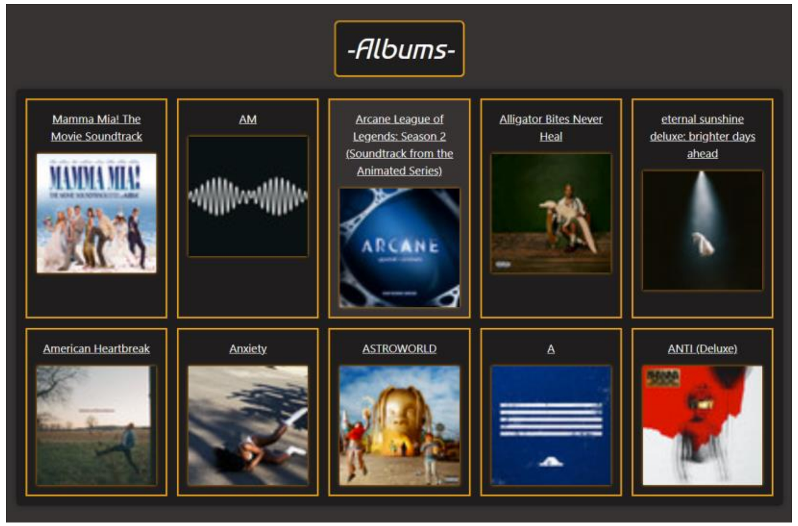
-Artists-

Ariana Grande
Lady Gaga
A$AP Rocky
Adele
Arctic Monkeys
Alex Warren
Addison Rae
Anuel AA
Taylor Swift
Doechii



-Tracks-

APT.
All The Stars (with SZA)
Anxiety
Die With A Smile
All The Way
Abracadabra
ANXIETY (feat. Doechii)
Ain't No Love In Oklahoma (From Twisters: The Album)



-Albums-

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

Testing Overview

- Multi-layered strategy to ensure proper functionality
- Top-down testing
  - User authentication
  - Early search functionality
- Black-box testing
  - User interactions
  - Proper search results
- White-box testing
  - Validate logic
  - Database queries
  - Security measures
- E2E testing
  - Application user flow
  - Cohesive functionality

Test Cases

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

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

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

Test Case	Steps	Expected Result	Actual Results – Ryan
SQL Injection Prevention	1. Enter ' OR 1=1 -- into login field. 2. Click **Login**.	User is not authenticated, and an error message appears.	User is not authenticated, and an error message appears.
XSS Protection	1. Submit <script>alert('test')</script> in a text field. 2. Click **Post**.	Script is sanitized and not executed.	Script is sanitized and not executed.
Password Hashing	1. Register a new user. 2. Check database.	Password is stored as a hashed value.	Password is stored as a hashed value.
Data Integrity Check	1. Create a new profile. 2. Manually edit backend data. 3. Refresh the profile.	Invalid data is not displayed.	Invalid data is not displayed.

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

Steps	Expected Result	Actual Result - Gaetano
1. Register an account with a valid email and username.		Account Registered
2. Log out.		
3. Attempt to register again with the same email or username.	Registration is blocked; error message indicates email/username already in use.	Error message indicating cannot use email address already in use

Lessons Learned

- Setting a realistic scope
  - Don't set expectations too high early on
- Stronger communication
  - Makes planning easier
- Starting with a stronger technical foundation
  - More research on project components
- Stricter deadlines
  - Leave room for unexpected issues
- Increased proactivity
  - Thinking ahead

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

Demo