PlayBack Software Design Report

Gaetano Panzer II – Project Manager, Max Collins – Requirements Engineer, Ryan Farrell – Software Architect

April 2025

Foundational Responsibilities

- Ethical
  - Avoids bias and mainstream conformity by keeping ratings private.
  - Users must post their own thoughts before viewing others' opinions.
  - Ensures equal representation for all musical perspectives.
  - No engagement-driven algorithms to prevent toxic discourse.
  - Reduces bandwagoning and amplification of controversial content.
- Security
  - Encrypts all sensitive user data, including private ratings.
  - Users control post visibility (public, private, or friends-only).
  - No data selling or third-party tracking for advertising.
  - Strict authentication protocols for external integrations (e.g., Spotify API).

- Legal
  - Complies with data protection laws .
  - Users can access, modify, and delete their data.
  - No storage or distribution of copyrighted music.
  - Adheres to licensing agreements of integrated services like Spotify.
  - Transparent terms of service and privacy policies.

- Societal Impact
  - Encourages independent, intentional music engagement.
  - Helps users break out of algorithm-driven echo chambers.
  - Expands exposure to diverse music beyond popularity metrics.
  - Fosters a judgment-free space for personal music exploration.
  - Promotes a healthier connection to artistic expression.

Success Criteria

Core Functionality and UX

- Users can create and log into accounts securely
- Users can manually track their listening habits and see personal insights.
- Users can write and post their thoughts on music without external influence.
- Private rating system allows for reflection without public pressure.
- Users can tag music with "vibes" for easy organization and discovery.
- Spotify API integration enables song playback and metadata display.
- Users can customize their profile with privacy settings and visibility options.

Success Criteria

Security & Data Protection

- User data is encrypted and stored securely, following **North Carolina privacy laws**.
- Users can access, modify, and delete their data.
- No third-party data tracking or unauthorized data sharing.
- Strong authentication protocols protect accounts and linked services.
- MySQL hashing will secure data

Success Criteria

Ethical & Legal Compliance

- Users can create and log into accounts securely
- Users can manually track their listening habits and see personal insights.
- Users can write and post their thoughts on music without external influence.
- Private rating system allows for reflection without public pressure.
- Users can tag music with "vibes" for easy organization and discovery.
- Spotify API integration enables song playback and metadata display.
- Users can customize their profile with privacy settings and visibility options.

Success Criteria

Community & Engagement

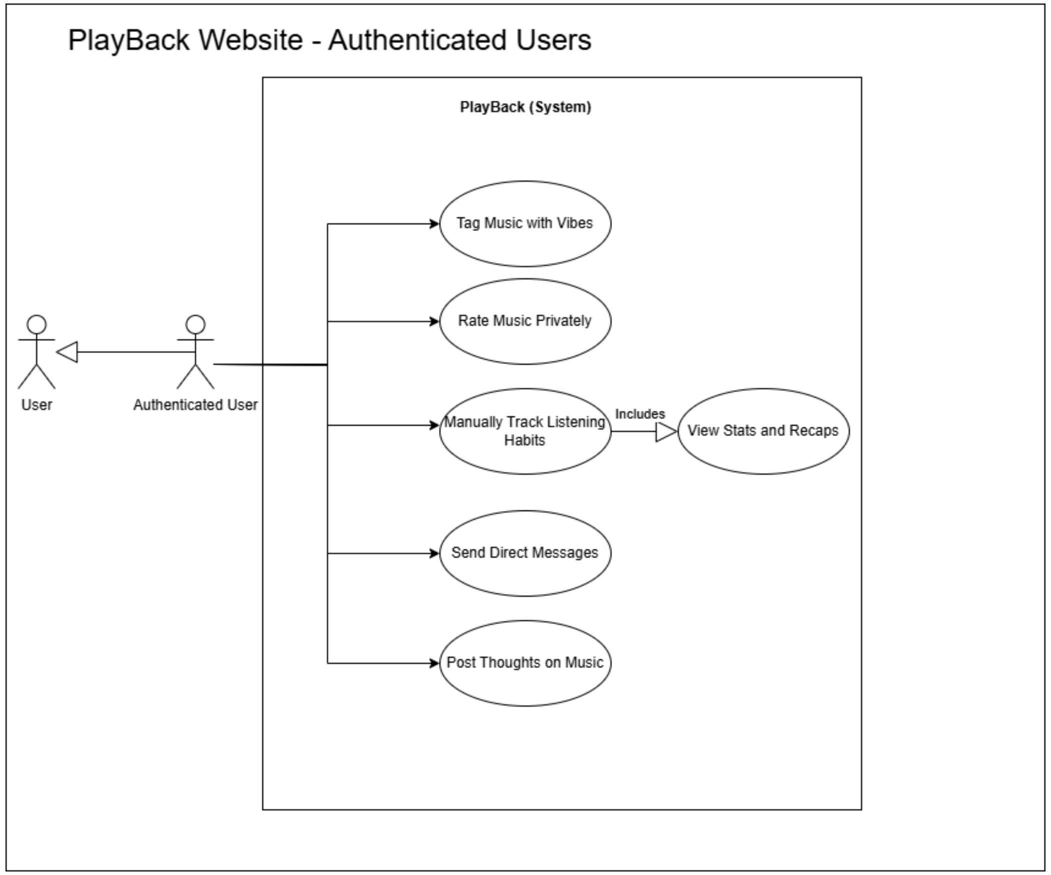
- Users can view personal history and listening trends over time.
- Direct messaging works for private user interactions.
- Community interactions remain **non-intrusive**, keeping the focus on personal engagement.
- The platform fosters a diverse and inclusive space for all music perspectives.

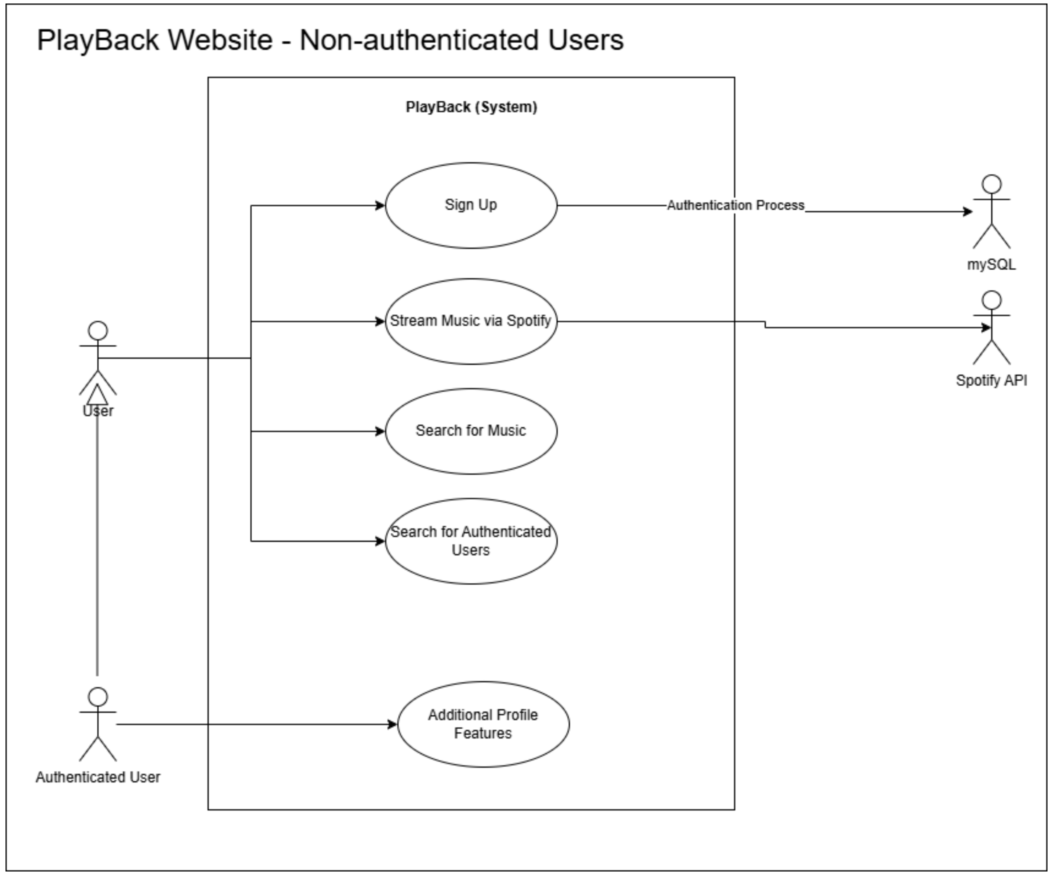
Success Criteria

Performance & Scalability

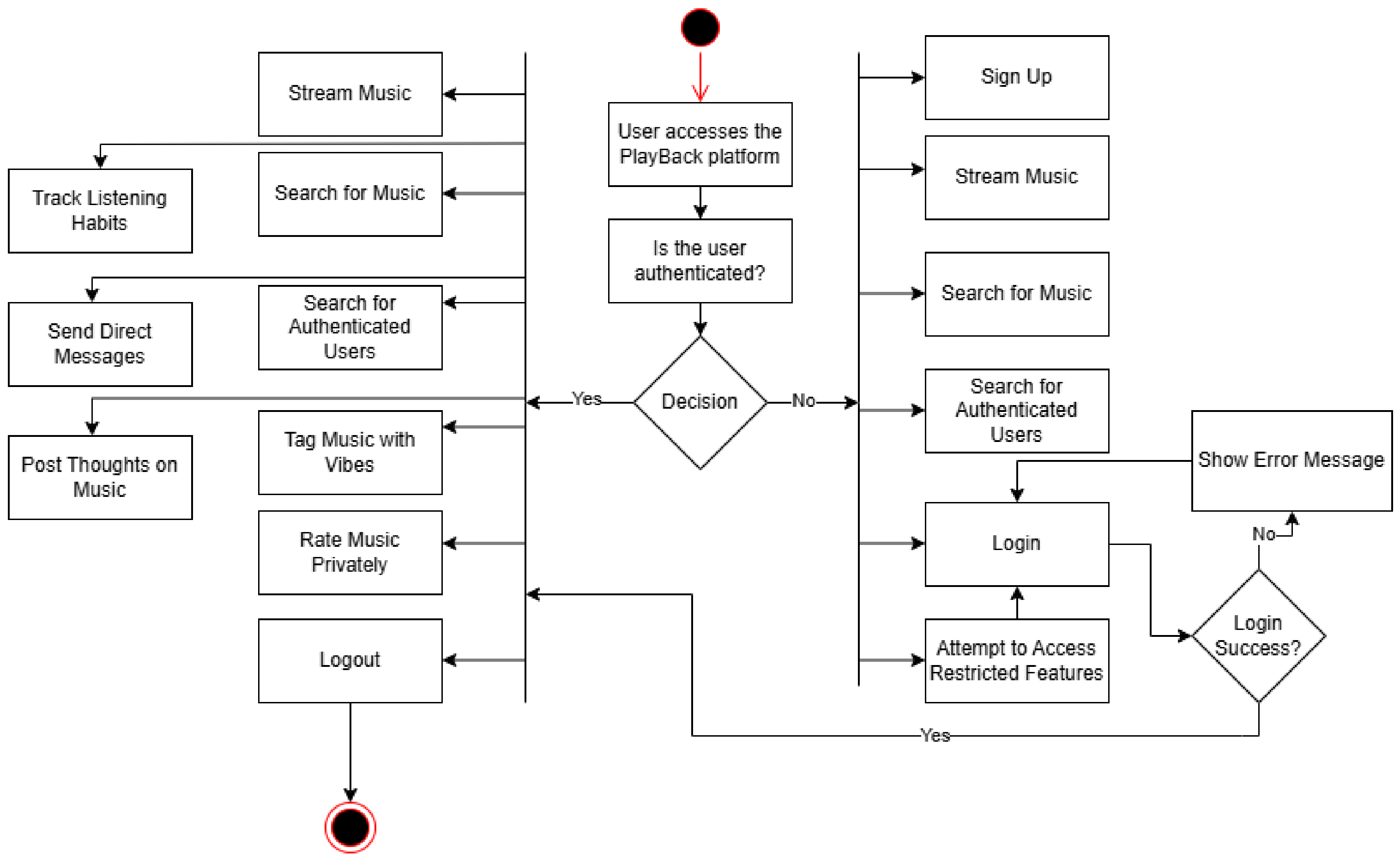
- The platform loads efficiently with minimal downtime.
- Database handles increasing users and posts without performance issues.
- Future feature expansions (e.g., profile customization) remain feasible without overcomplication.

Use Case Models

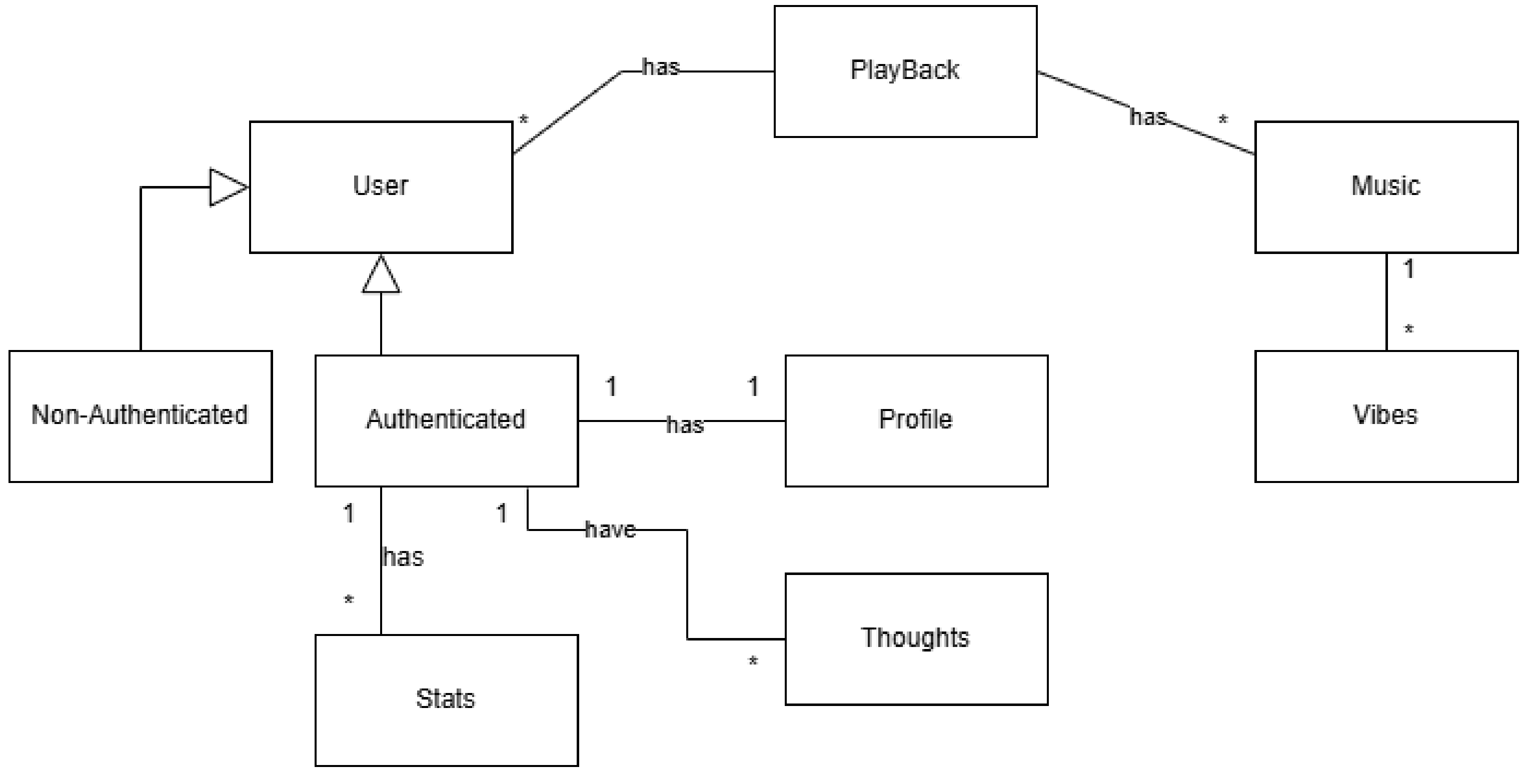




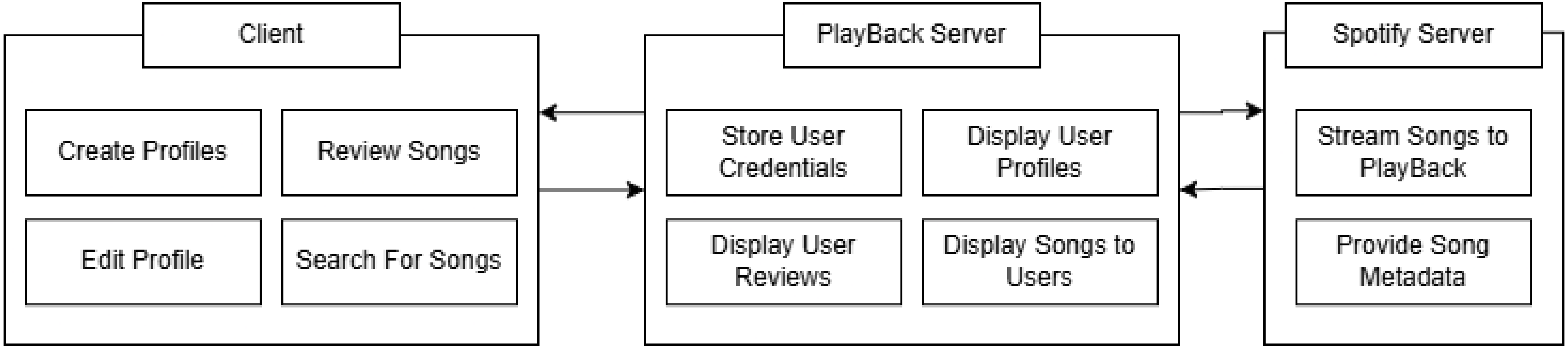
Activity Diagram



Class Model


PlayBack
has
*
has
*
User
Music
1
*
Non-Authenticated
Authenticated
1
has
1
Profile
Vibes
1
has
*
1
have
*
Thoughts
Stats

Architectural Models



Advantages
- Scalable
- Reduce strain on client
- Robust framework

Disadvantages
- Potentially costly
- Server strain/bottlenecking
- Server failure

Client-Server

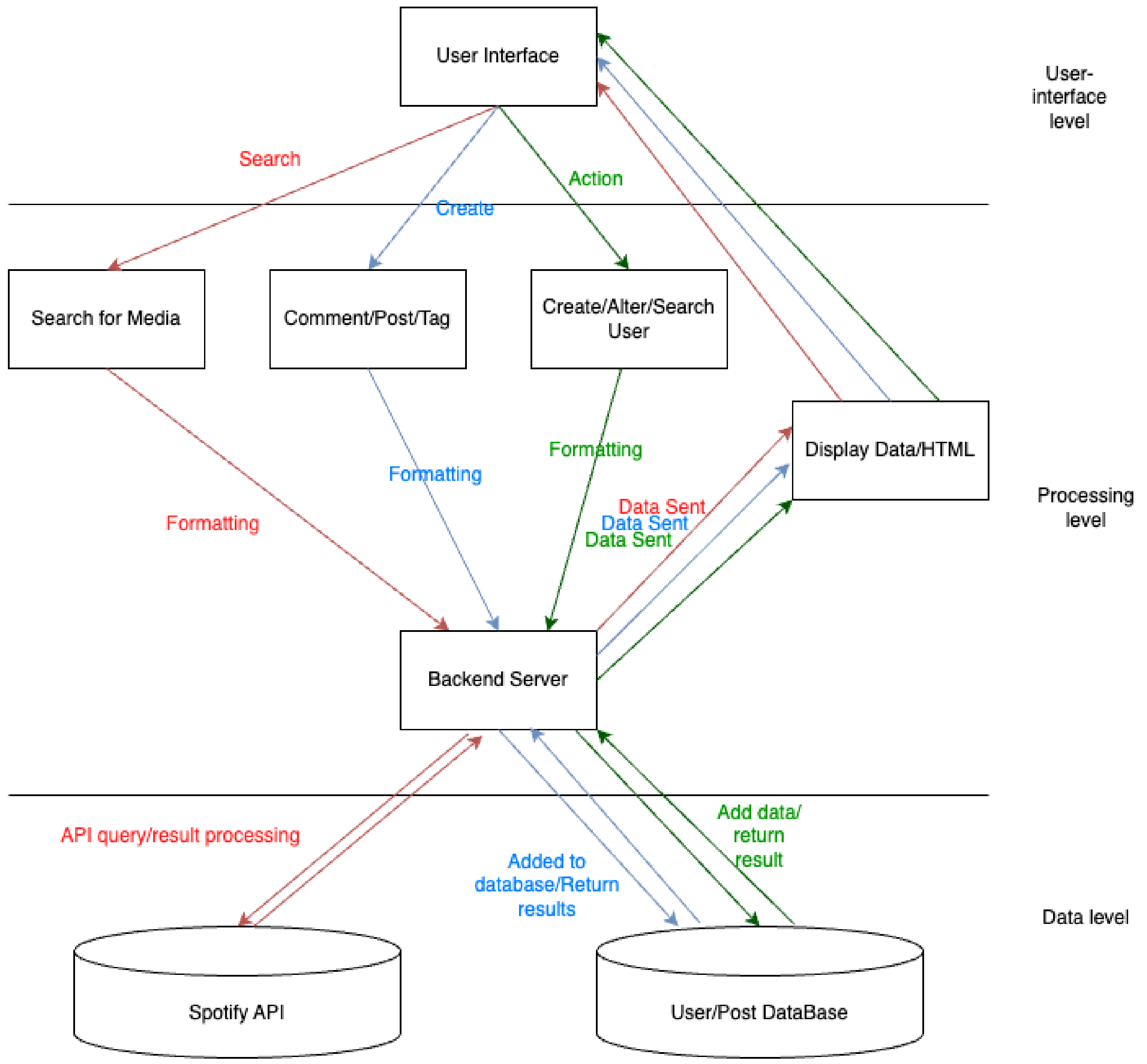
Layered Architecture

Advantages

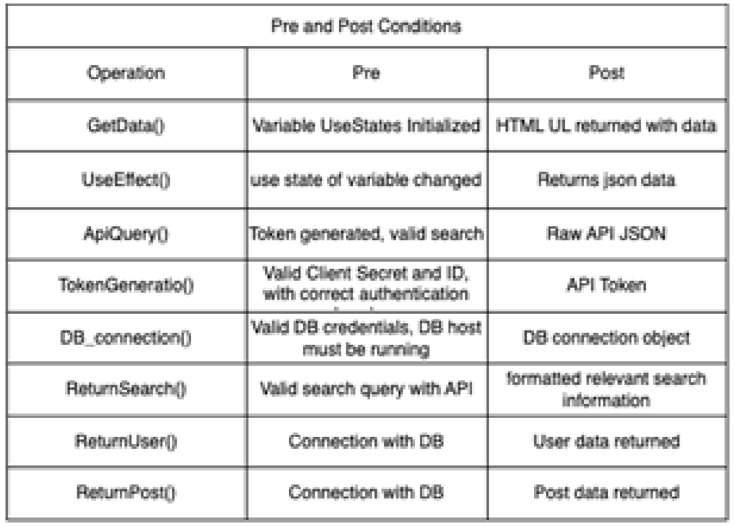
- Organized
- More secure
- Easy to maintain
- Python libraries

Disadvantages

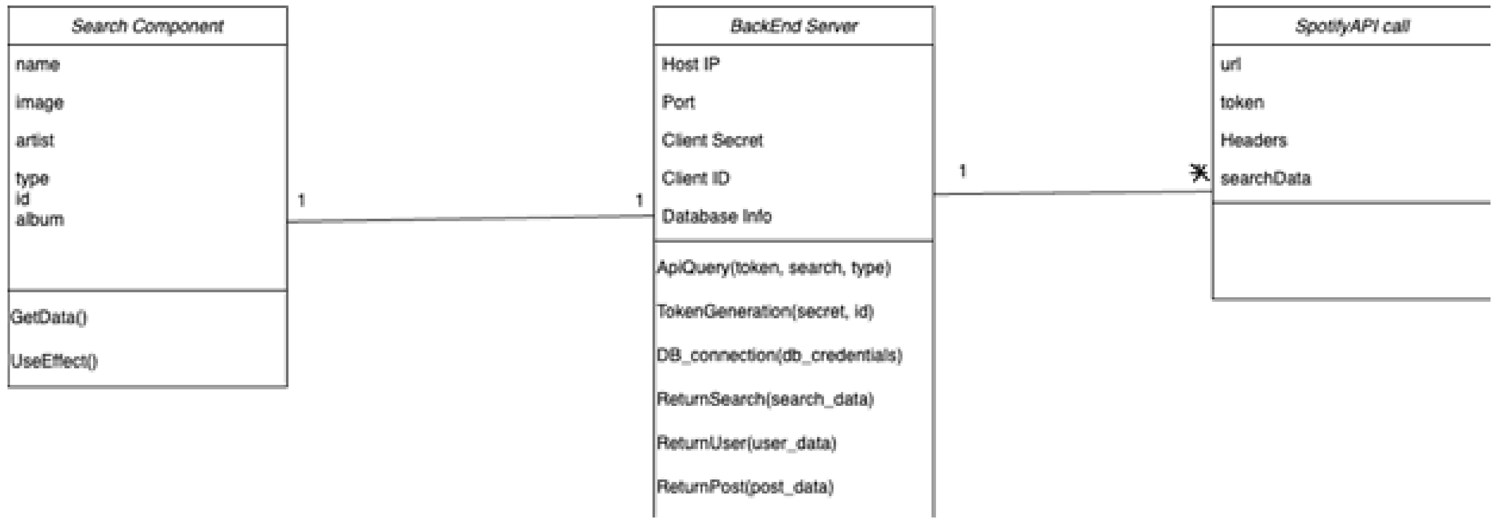
- More latency
- Single point of failure
- More work (theoretically)



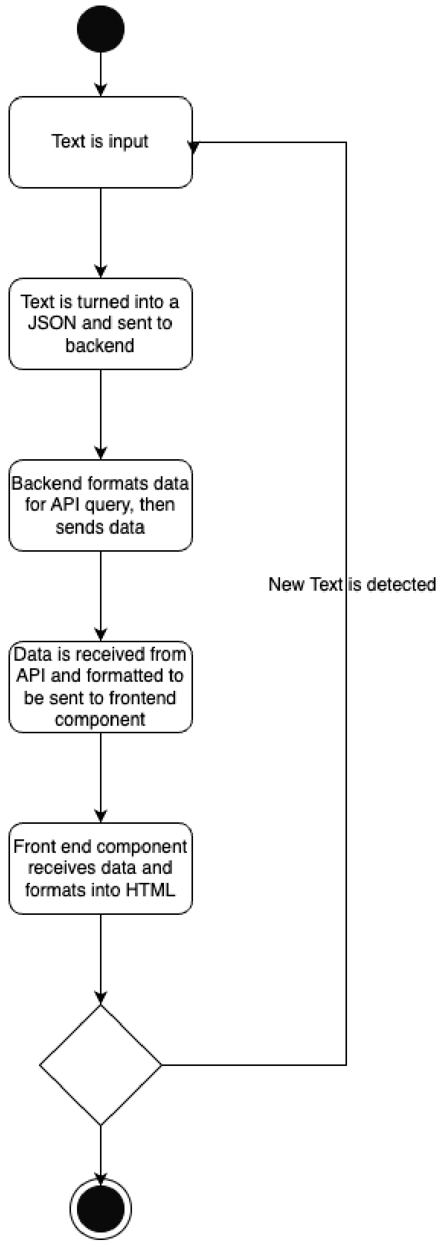
Subsystems – Music Search



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

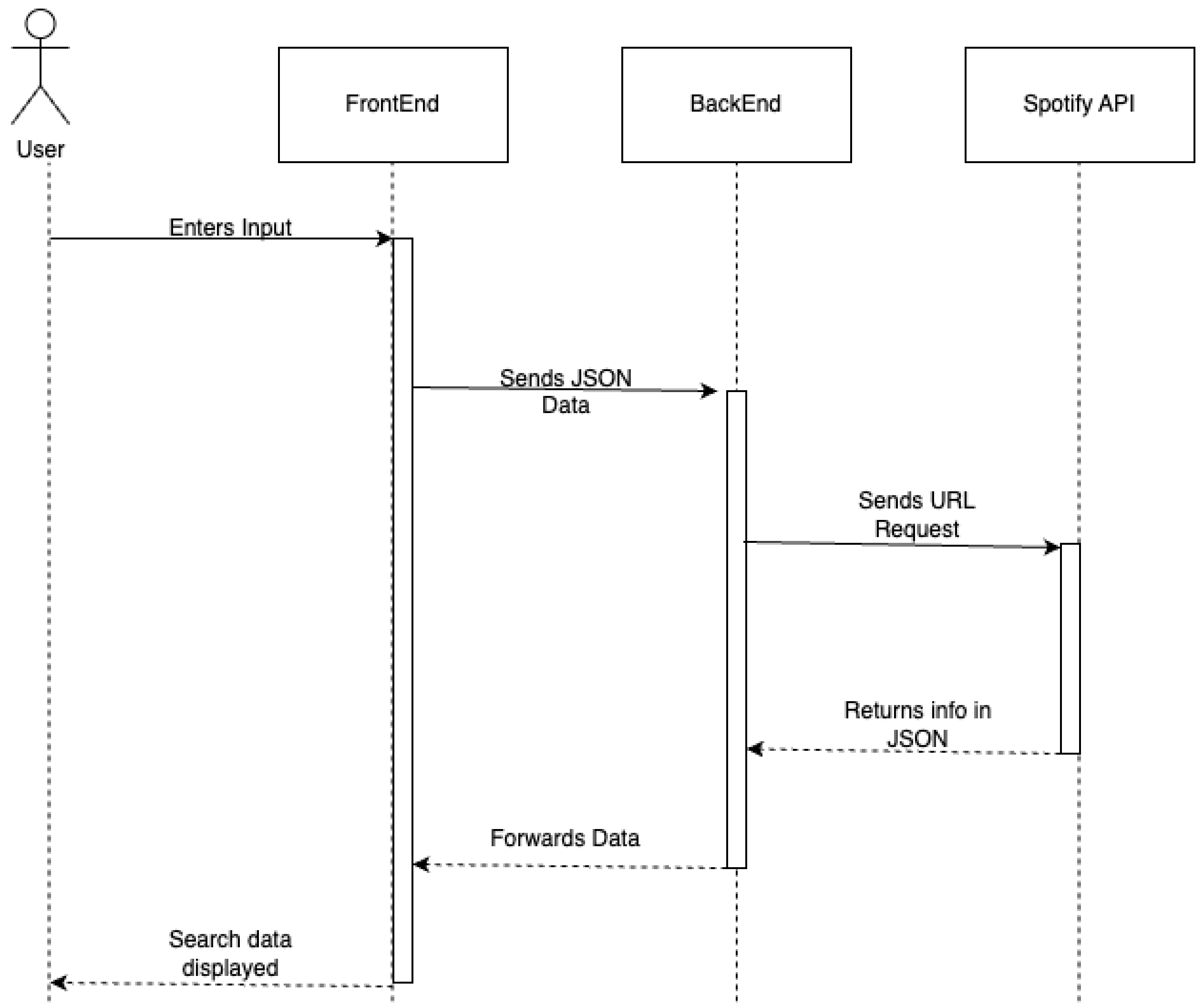


API usage Class Diagram

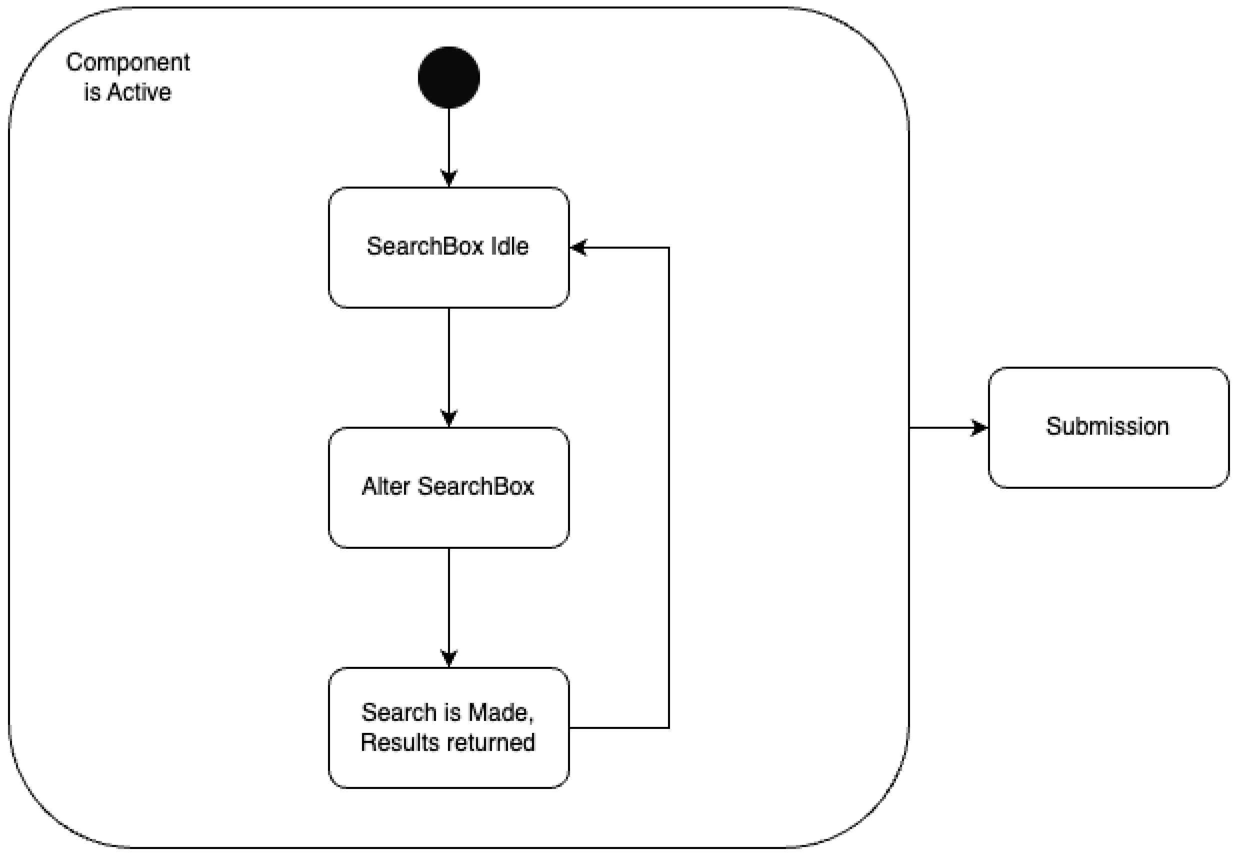

Text is input
Text is turned into a JSON and sent to backend
Backend formats data for API query, then sends data
Data is received from API and formatted to be sent to frontend component
Front end component receives data and formats into HTML
New Text is detected

Search bar component activity diagram

Subsystems – Music Search

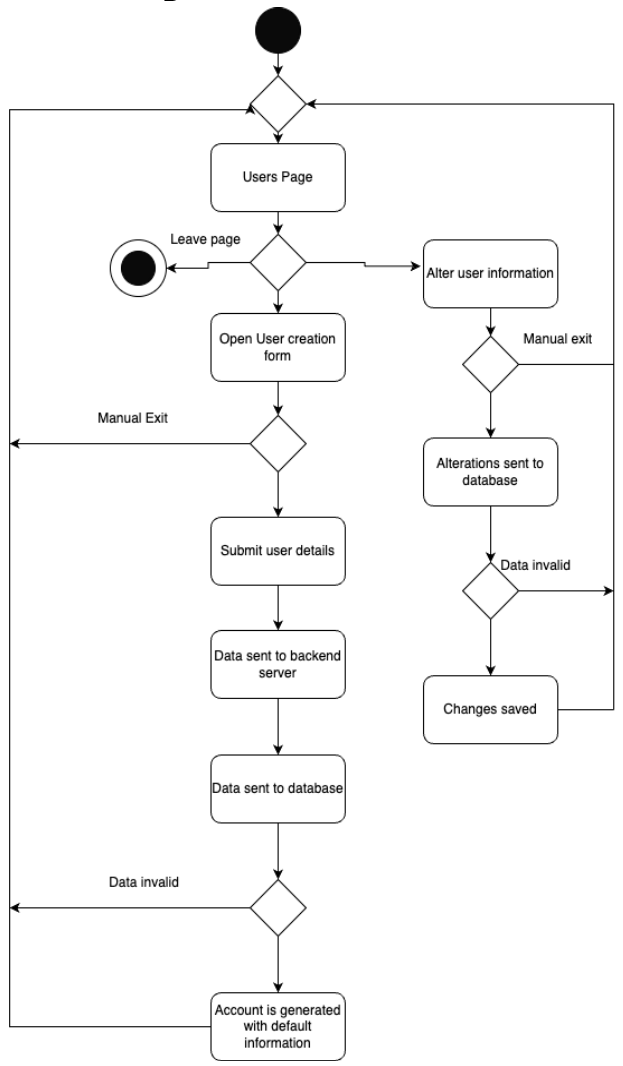


Music search sequence diagram

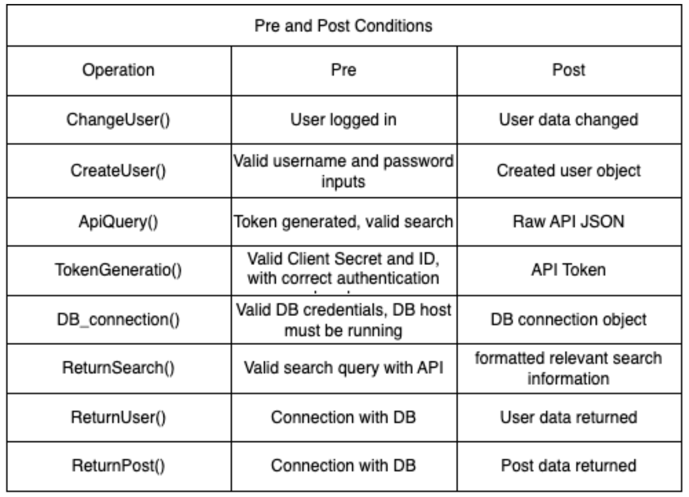


Search bar component state diagram

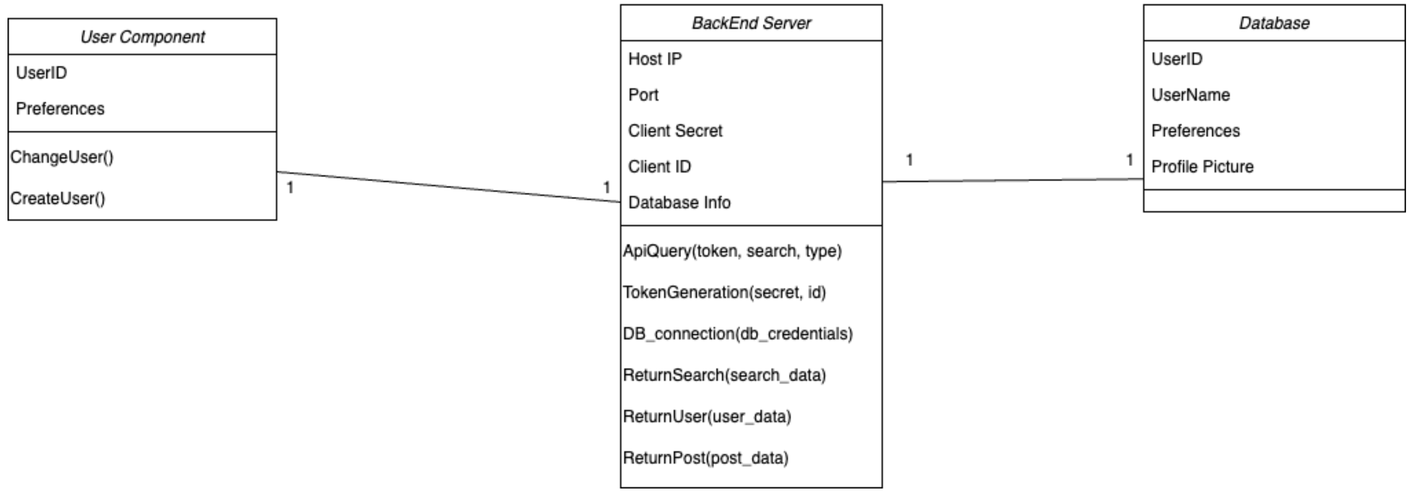
Subsystems – Creating/Altering User Data



User creation/alteration activity diagram

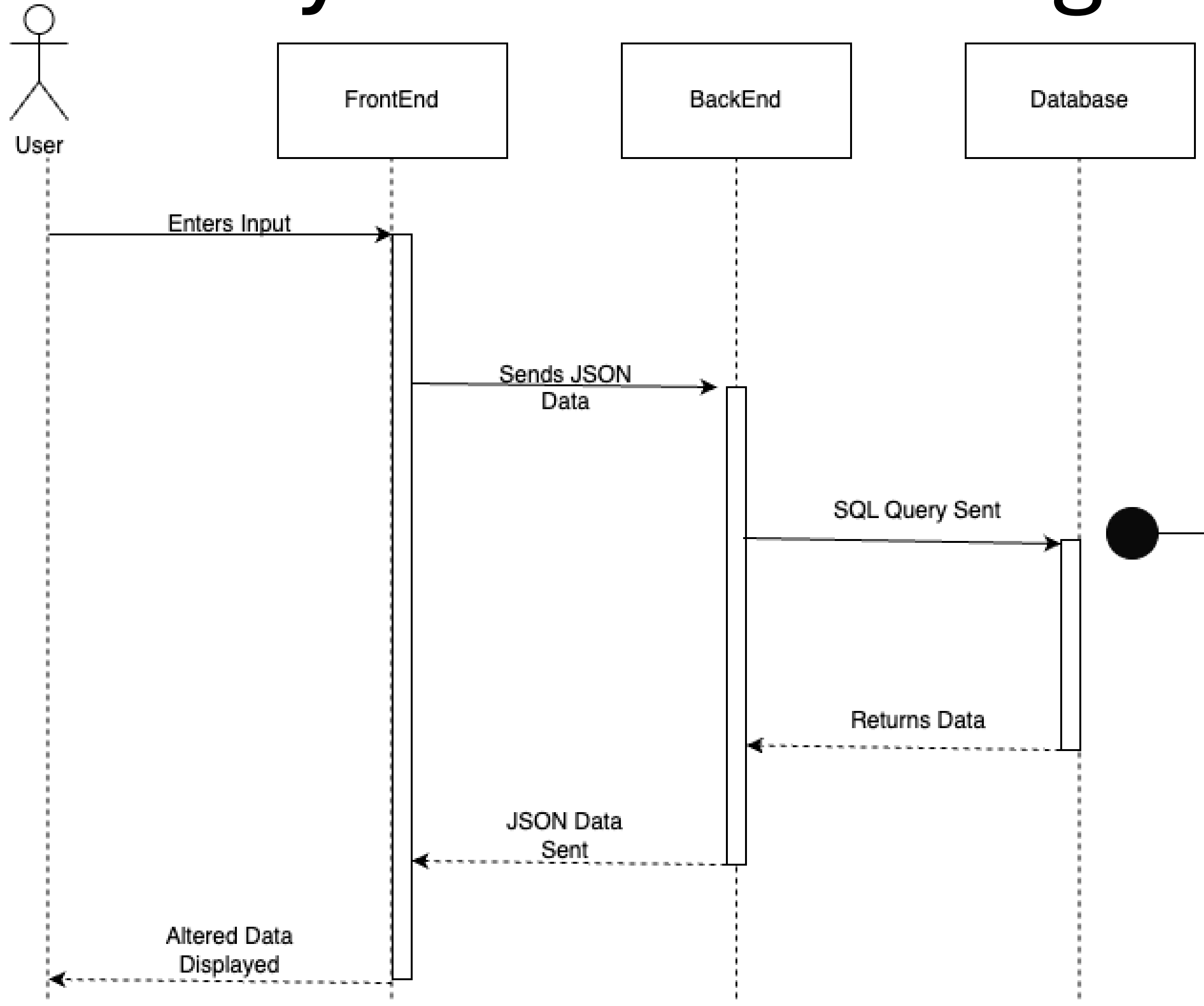


Pre and Post Conditions		
Operation	Pre	Post
ChangeUser()	User logged in	User data changed
CreateUser()	Valid username and password inputs	Created user object
ApiQuery()	Token generated, valid search	Raw API JSON
TokenGeneratio()	Valid Client Secret and ID, with correct authentication	API Token
DB_connection()	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

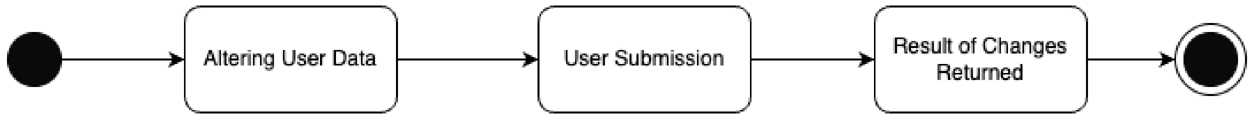


User creation/alteration class diagram

Subsystems – Creating/Altering User Data

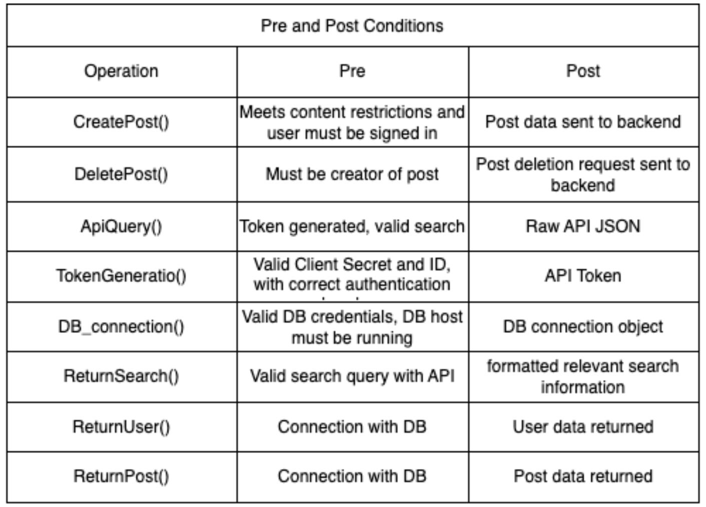


User creation/alteration sequence diagram

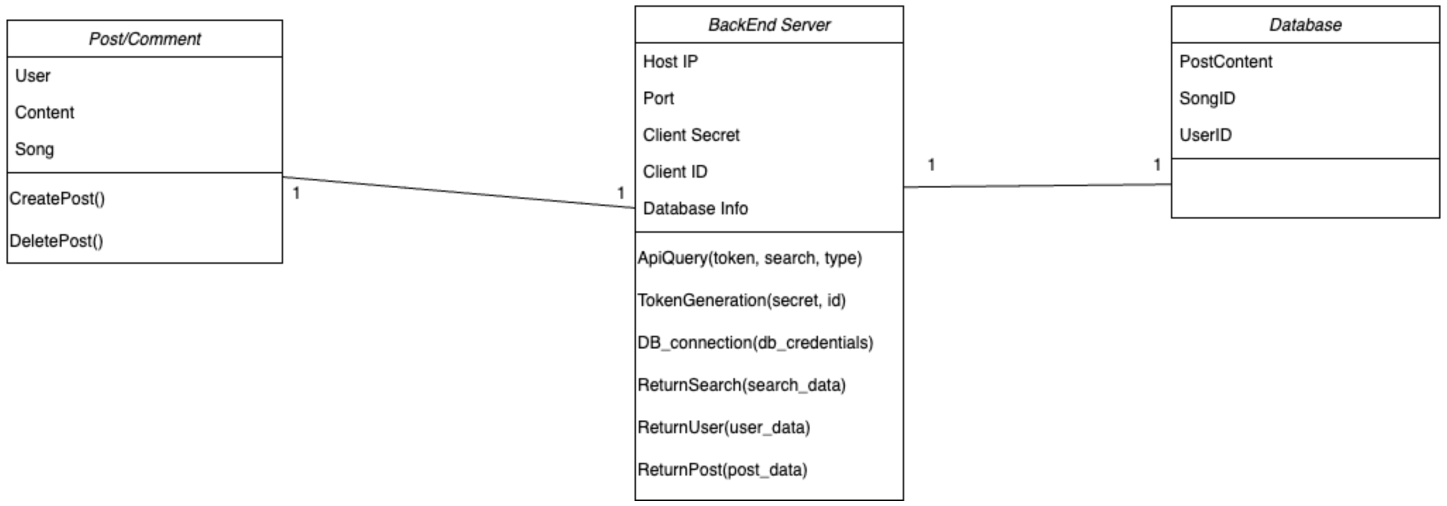


User form submission state diagram

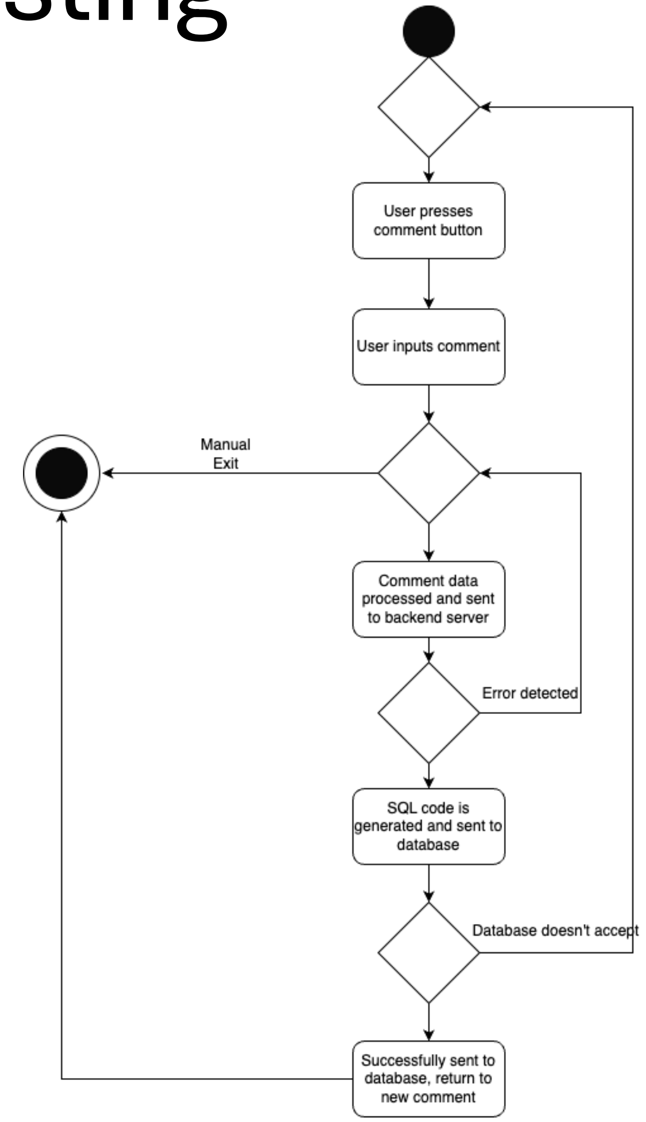
Subsystems – Commenting/Posting



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

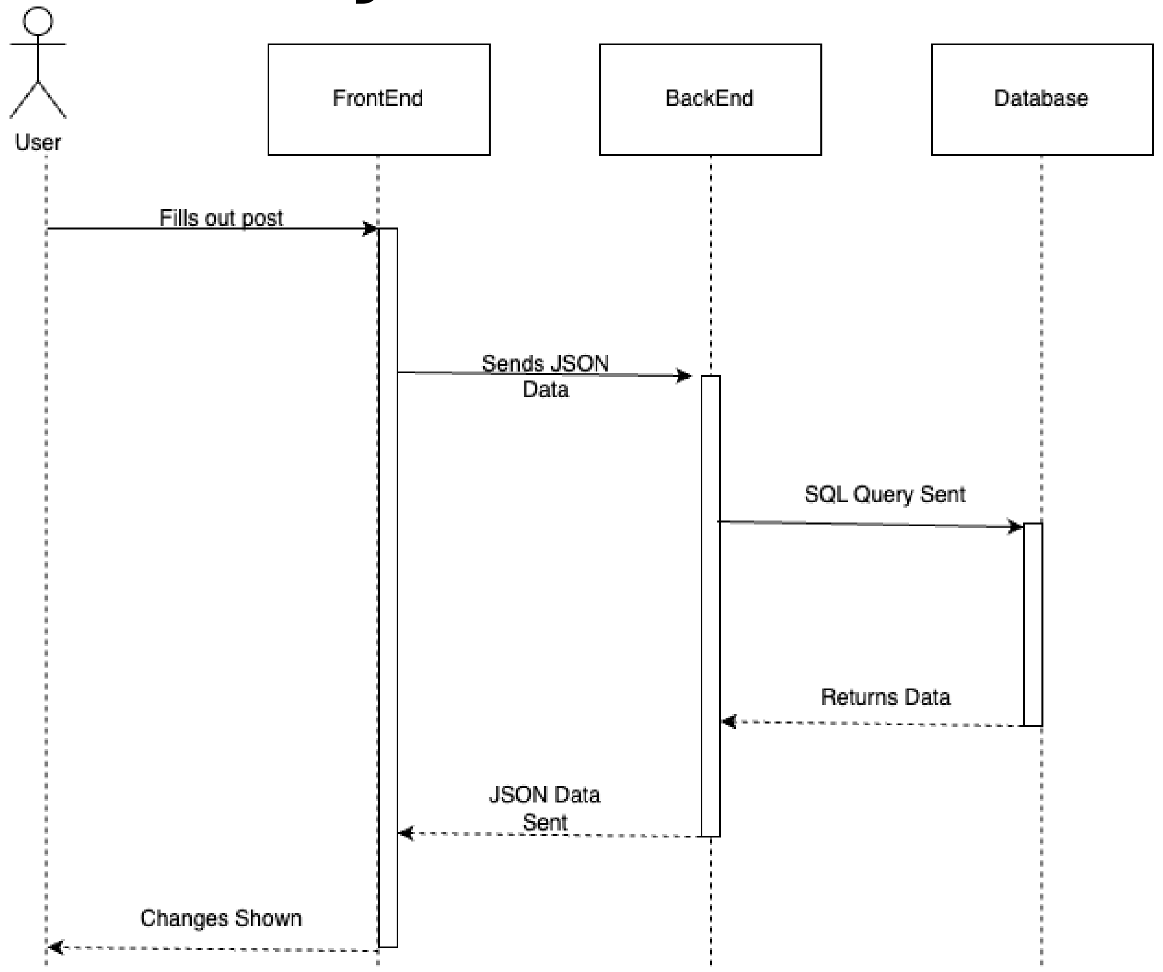


Creating a comment/post class diagram

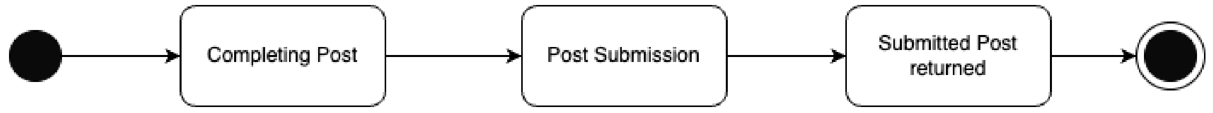

User presses comment button
User inputs comment
Manual Exit
Comment data processed and sent to backend server
Error detected
SQL code is generated and sent to database
Database doesn't accept
Successfully sent to database, return to new comment

Comment creation activity diagram

Subsystems – Commenting/Posting



Post/comment creation sequence diagram



Filling out post form state diagram

Project Testing and Validation

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.

Project Testing and Validation

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.

Project Testing and Validation

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".

Project Testing and Validation

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.

Project Testing and Validation

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.

Project Testing and Validation

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.