Music Social Media Website

Zachary Faulkner
California State University, Fullerton

Abstract

The project I am proposing is a music review website that also includes a multitude of features to enhance both the listening and community components within music. This project is being made with the intent of offering users a tool to strengthen the communities of their favorite artists. Music is an experience that is amplified when shared with others, this platform aims to encourage users to find and engage with other communities of different artists. While music recommendation algorithms nowadays are quite advanced, this platform will focus on users sharing music amongst themselves. Put simply, the goal of this project is to create a hub for music enjoyers to chat about, listen to, discuss, or find music.

Table Of Contents

Introduction

The goal of this project is to create a website for music enjoyers to gather to share thoughts, ideas, and connect with others that share similar music tastes. The website will have multiple features geared towards making this goal a reality. This website will consist of three main features that will serve as the pillars of the platform. These pillars being the music review feature, the statistics feature, and the listening party feature. It is through these key features that I envision users will be able to create and foster strong communities for their favorite genres and artists. In order to make these features both functional and efficient in order to maximize the user experience, I plan on integrating the Spotify API within this platform in order to preserve the goals that I have set.

The first feature that will function as the core functionality of the platform will be the music review feature. This platform will function moreso as a social media website than a traditional music review website. With the main difference being that this platform will encourage discussion through the design of the website. For example, I want user profiles to function similarly to those found on traditional social media websites such as Instagram or Twitter/X. What this means is that instead of a post history that you would find on the aforementioned traditional social media platforms, the post history would consist of both music reviews as well as another tab of a user's comments on another user's music review. Another functionality of other social media websites that I want to implement is the ability to follow and direct message other users. The follow button functionality would allow users to follow another user who's opinion they may agree with. However, I believe the main benefit that will come from implementing a follow button will be that users can follow another user who has a taste in music they enjoy, thus if the latter user reviews an album positively, the former user will be highly inclined to check it out.

The second feature that this platform will revolve around will be the statistics feature. This will allow users to see an in-depth breakdown of their music listening history. Users will be

able to select a specific timeframe for which their music listening history will be analyzed. The goal of this feature is to both allow users some insight into what music they frequent as well as to serve as a benchmark for users to look back on how their music taste has evolved throughout their time using the platform. As mentioned previously, expanding the music taste of users through the sharing of music is one of the goals of this platform. Thus, being able to quantify this change is important. This feature could also encourage users to seek out communities that they may have not considered previously. For example, a user may notice on the statistics page that they have been listening to R&B much more frequently than they ever have before, this may encourage the user to seek out communities to continue expanding their understanding of the genre.

The final feature I want to implement is the listening party feature. This feature will allow users to sync their Spotify accounts with other users and play music simultaneously. For example, this feature would allow a community to share the experience of listening to their favorite artists' new album for the first time. In order to facilitate the community environment within this feature, a chatbox will be implemented so that users can communicate and react to the current music being played in realtime.

As for the utility of this platform, I believe this could be a great tool for users and artists alike. As mentioned before, this platform aims to be a hub for people from all over the world to listen to, share, and discuss music. As for artists, using features such as the “listening party” to create a unique experience for their fans could help strengthen their communities.

Project Plan

Team: Zach Faulkner

Project Schedule:

February

- Pitch and Research
    - Capstone Pitch - Present initial idea for capstone project to class, gather feedback as well as taking in ideas from other students.
    - Designate initial requirements - Find the initial requirements necessary for the project to function as designed (API Integration, Features, Potential Issues).
    - Research similar entities - Use similar entities as a springboard for both integrating key features as well as iterating said features so that the projects are distinct.
- Milestone 1
    - Research Completion - Complete research of initial requirements as well as similar entities.

March

- First Project Proposal
    - Requirements Modeling - Create user scenarios/use-cases as well as diagrams modeling what is needed for the project to function as intended.
    - Iterate on Requirements from Analysis - Using the initial requirements discovered as the baseline, build upon said requirements to enhance features as well as user experience.
    - Use Cases - Create user scenarios/use cases illustrating what should be the standard for the average user experience for the website.

    - Persona Cases - Create cases that demonstrate how a user may interact and the patterns they may exhibit while using the website.

April

- Project Proposal 2
    - Design Project Architecture - Research and integrate both an architecture and design pattern to follow while developing the website.
    - UX Design - Research and integrate UX architectures, design patterns, frameworks, and aesthetics for the development of the website.

May

- Project Proposal 2 cont.
    - Prototype - Create a mockup of the website, demonstrating how the website will populate data using the Spotify API as well as how user data will be stored and managed.
    - Final Iteration of Requirements - Design, approach, and necessary features should be fully planned out and researched in order to fully implement them within the website.
    - Project Architecture Design - Use chosen architecture and design patterns to serve as a guideline for developing the project.
    - UX Design - Use aforementioned chosen UX architectures, design patterns, frameworks, and aesthetics to develop an intuitive and seamless user experience.

August/September

- Begin Development
    - UI/UX Design - Begin using the chosen aesthetics and frameworks to begin designing an appealing and intuitive UI while also using the chosen architecture and design patterns to create a UX that is simple and enjoyable for the user to navigate.
    - Posting and Commenting Features with User History - Users should be able to post their own reviews as well as comment on other users reviews, with their posting and commenting data being attached to their account and viewable by other users.
    - Following Feature - Users should be able to follow each other on the platform, the main review page should then be populated with reviews and/or comments from followed users.

October

- Development Continued

- Direct Messaging Feature - Users should be able to directly message each other, users should be able to specify who can direct message them, this can be filtered by if the users follow each other or the user may leave their direct message inbox open to anybody.
- Spotify API Integration - Integrate the Spotify API to populate data for the review board as well as allowing for users to access the statistics and listening party feature.
- Statistics Feature - Users can link their Spotify accounts and see the data of their listening history spanning from albums, songs, and artists. Users can filter the data by time period (Example: Only use data from the last 6 months).

November

- Finishing Development
  - Listening Party Feature - Users can create listening parties for other users to join and listen along to the music. USers can specify if the listening party is private or not. If the listening party is specified as private, the user will be given a link and/or code to distribute as an invite for the listening party.
  - Live Chat Box Feature creation - Will be implemented within the listening party feature, will allow users to communicate about the music in real time.

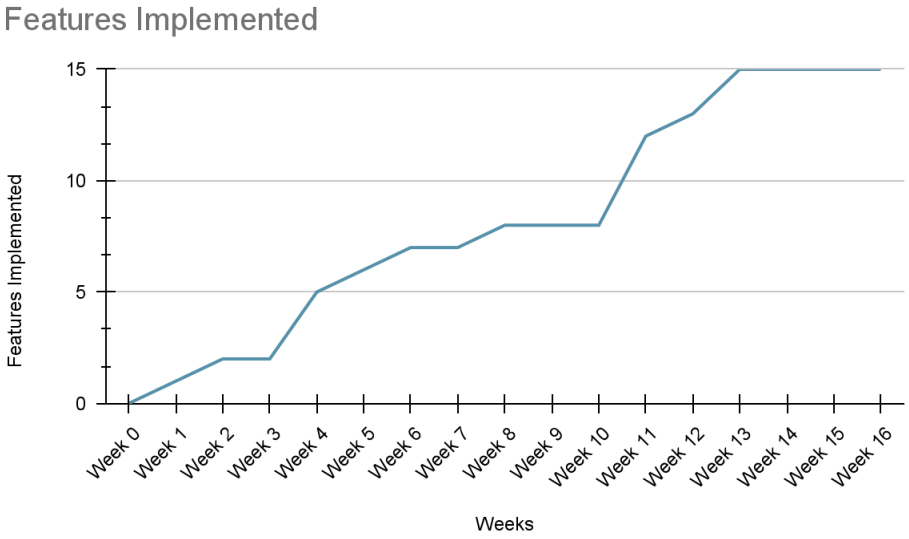
December

- Submission
  - Complete Testing - Failed Tests at this point should be zero. The project should be fully functional.
  - Documentation - Create documentation detailing how the project was developed as well as making it accessible for others to iterate on the project if they choose to do so.
  - User Guides - Guides for users to navigate and use the project and it’s features
  - Run-Books - Create a guide for users to debug and/or navigate the procedures within the website

Metrics

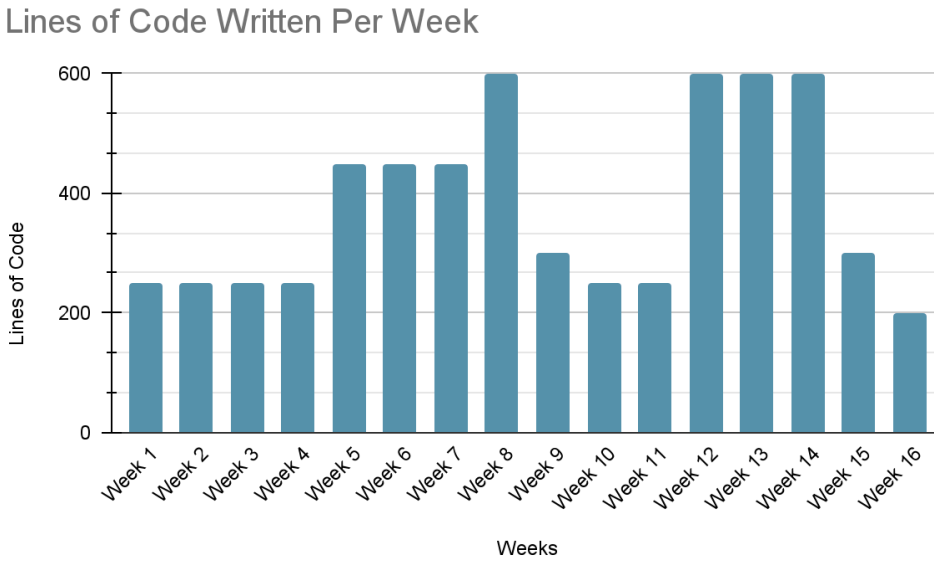
Feature Integration

My goal for feature integration is to complete a feature every 2-4 weeks at maximum. Feature integration serves as a benchmark for how close the project is to completion. Below is the complete graph of my features integration from this semester.



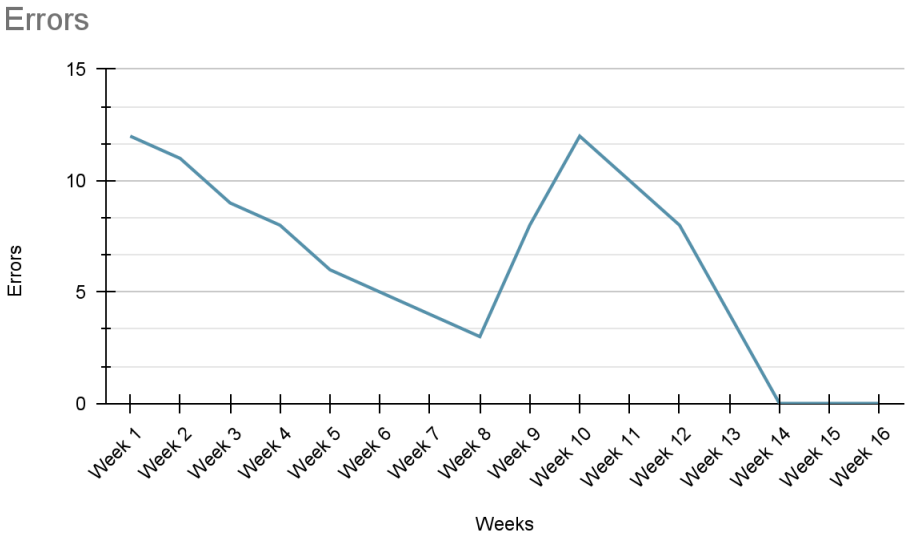
Lines Of Code Per Week

This metric shows how much code will be written per week, this serves as a way to quantify progress towards project completion. Below is the complete graph of my lines of code written from this semester.



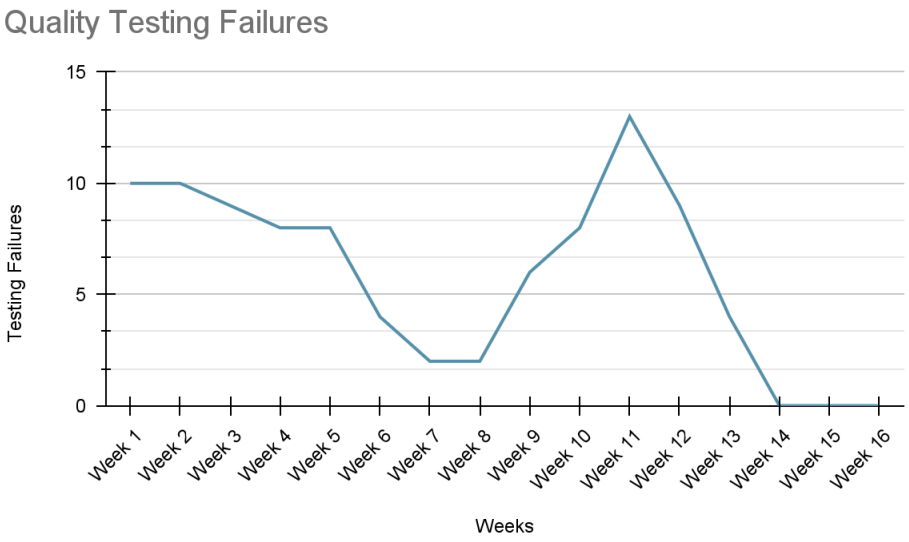
Errors

This metric displays a rough estimate of how many errors will pop up and be fixed week to week. Errors will vary in how often they occur and are fixed, however the total count should be 0 by the submission date. Below is my graph I calculated using error count from throughout the semester.



Failed Tests

Failed Tests would consist of things such as buttons that don't work or data not being displayed correctly. These failures should decrease consistently throughout the duration of the project lifecycle. Below is the graph of quality testing failures from this semester.



Requirements Engineering

Use Cases

Use Case Title: Creating an Account
Actor: User
Precondition: User has navigated to website sign up page
Success Guarantee: User can create and account
Main Success Scenario:

1. User enters email address
2. System validates email exists
3. User enters password
4. User confirms password
5. User submits email and password fields
6. User's data is inserted into database
7. User's account is now created

Use Case Title: Changing a Password
Actor: User
Precondition: User has navigated to profile page
Success Guarantee: User can change password
Main Success Scenario:

1. User clicks on change password option on profile page
2. User enters old password
3. User enters new password

4. User confirms new password
5. User submits password fields
6. User;s data is updated in database
7. User has now changed password

Use Case Title: Changing EMail
Actor: User
Precondition: User has navigated to profile page
Success Guarantee: User can EMail
Main Success Scenario:

1. User clicks on change password option on profile page
2. User enters new email
3. System validates email exists
4. User's data is changed in database
5. User has now changed EMail

Use Case Title: Delete Account
Actor: User
Precondition: User has navigated to profile page
Success Guarantee: User can delete account
Main Success Scenario:

1. User clicks on delete account option
2. User must enter string "I want to delete my account" into text field to confirm deletion
3. User submits text
4. User is prompted with final prompt confirming deletion
5. User submits prompt
6. User's data is removed from database
7. User has now deleted account

Use Case Title: Forgot Password
Actor: User
Precondition: User has navigated to log in page
Success Guarantee: User can recover account
Main Success Scenario:

1. User clicks on "Forgot Password" link
2. User enters EMail associated with their account
3. EMail is sent to the address containing a link to reset their password
4. User navigates to change password page through link
5. User enters new password
6. User confirms new password

7. User's data is updated in database
8. User has now recovered account

Use Case Title: Posting Reviews
Actor: User
Precondition: User has navigated to review forum
Success Guarantee: User can change password
Main Success Scenario:

1. User will be shown a collection of the current popular albums as well as a search bar to search for specific albums.
2. User navigates to the album they wish to review.
3. User will be shown the album's current community score as well as reviews from other users.
4. User clicks the "Create Review" button.
5. User is prompted with a slider allowing the user to choose a score from 0.0-5.0 as well as a text field to write a more detailed review.
6. User sets score with slider and writes review
7. User submits score and review
8. Review and score are saved to the database and attached to the user's profile
9. User's review is posted on forum

Use Case Title: Posting Comments
Actor: User
Precondition: User has navigated to review page
Success Guarantee: User can post comment
Main Success Scenario:

1. User will be shown a collection of the current popular albums as well as a search bar to search for specific albums.
2. User navigates to the album they wish to review
3. User will be shown the album's current community score as well as reviews from other users.
4. User clicks on a review
5. User will be prompted with the review in full as well as a text box to post a comment under the review
6. User submits comment
7. User's comment is stored in the database and attached to their profile
8. User's comment is posted under the review

Use Case Title: Direct Messaging
Actor: User

Precondition: User has navigated to direct messages tab
Success Guarantee: User can directly message someone
Main Success Scenario:

1. User can search through their following list to select recipient
2. User selects recipient
3. User is prompted with text box for message
4. User submits message
5. Message is sent to recipient
6. Conversation is saved to the direct messaging tab for easier access

Use Case Title: Privatizing Direct Message Inbox
Actor: User
Precondition: User has navigated to direct messages tab
Success Guarantee: User can privatize inbox
Main Success Scenario:

1. User clicks on setting icon within tab
2. User can select from options such as “From followers only” or “From mutual followers” or “Turn Off”.
3. User submits selection
4. User’s request is processed and updated in database
5. User’s changes are complete

Use Case Title: Viewing Statistics
Actor: User
Precondition: User has navigated to statistics, user has linked Spotify Account
Success Guarantee: User can view listening statistics
Main Success Scenario:

1. User will be shown all-time listening history by default with tabs pertaining to artists, albums, and songs.
2. User can filter the data used in the calculation by time period (1 Month, 1 Year, etc.)

Use Case Title: Listening Party
Actor: User
Precondition: User has navigated to listening party page, user has linked a spotify premium account
Success Guarantee: User can create a listening party
Main Success Scenario:

1. User is shown currently public listening parties that they can join as well as a “Create Party” button.
2. User clicks “Create Party”

3. Listening Party is created and connected to User's Spotify
4. New listening party is now viewable and searchable from the listening party page

Use Case Title: Private Listening Party
Actor: User
Precondition: User has navigated to listening party page, user has linked a spotify premium account
Success Guarantee: User can create private listening party
Main Success Scenario:

1. User is shown currently public listening parties that they can join as well as a "Create Party" button.
2. User clicks "Create Party"
3. User selects "private party" option
4. Listening Party is created and connected to User's Spotify
5. User is prompted with a code and/or link that they can share to give others access to the listening party

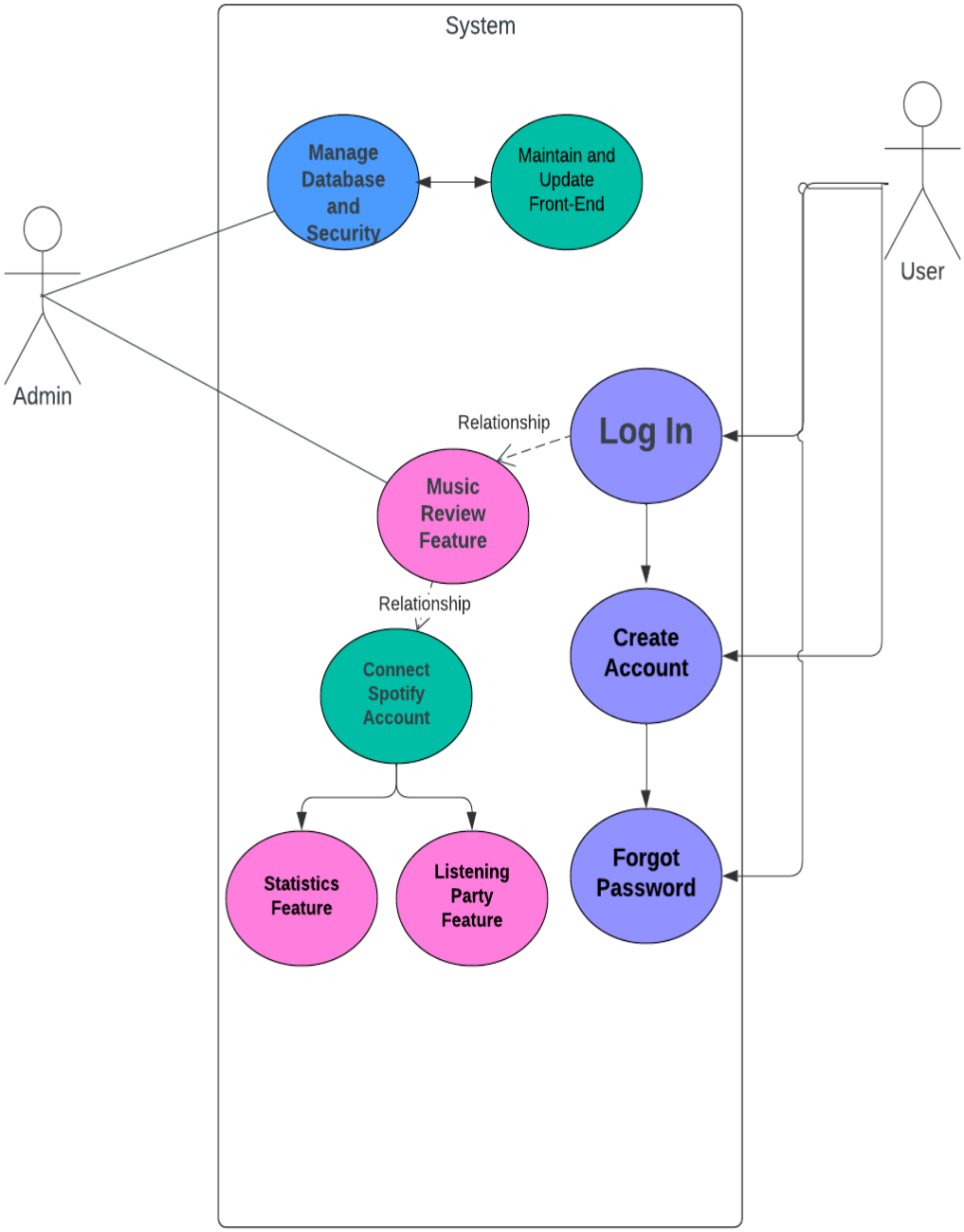
Use Case Title: Chatbox Feature
Actor: User
Precondition: User has navigated to listening party page, user has linked a spotify premium account
Success Guarantee: User can chat with others through chatbox
Main Success Scenario:

1. User is shown currently public listening parties that they can join as well as a "Create Party" button.
2. User joins a listening party
3. User can chat with others through a public chatbox

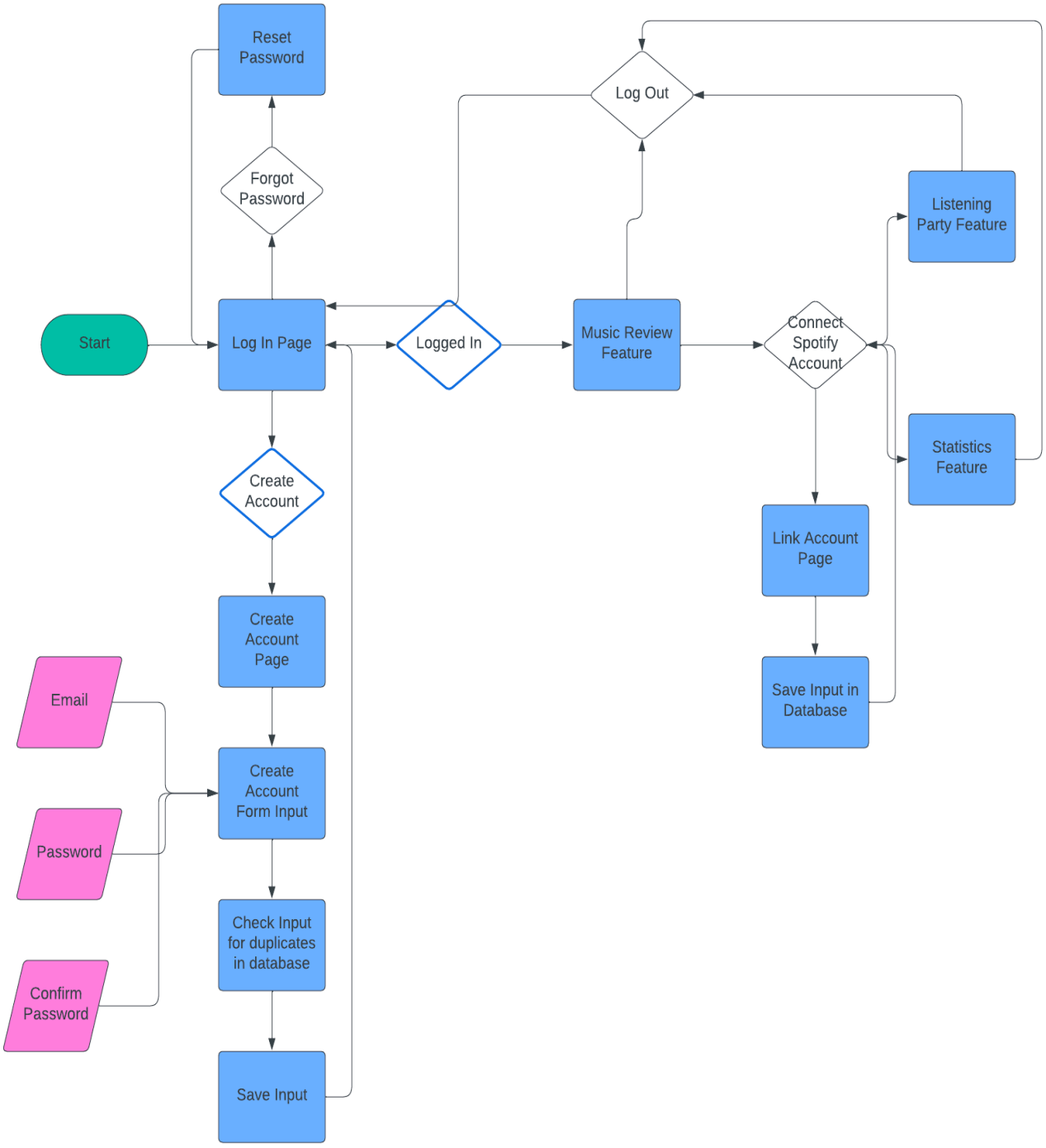
Use Case Title: Linking Spotify
Actor: User
Precondition: User has navigated to profile page
Success Guarantee: User can link their spotify account
Main Success Scenario:

1. User clicks on "Link Spotify" button.
2. User is prompted with Spotify login page
3. User submits credentials
4. Users credentials are added to their session ID within database
5. User's spotify account is now linked

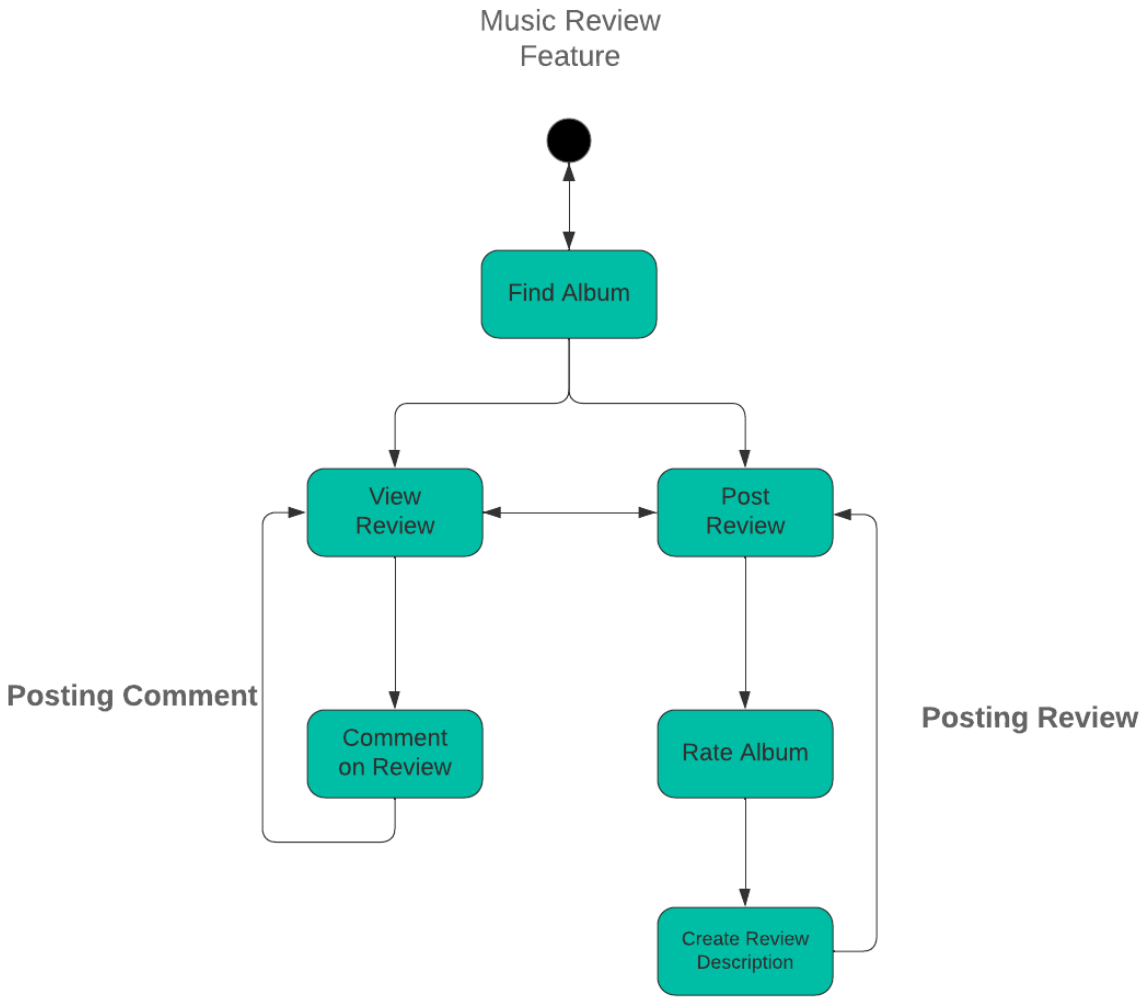
Use Case Diagram

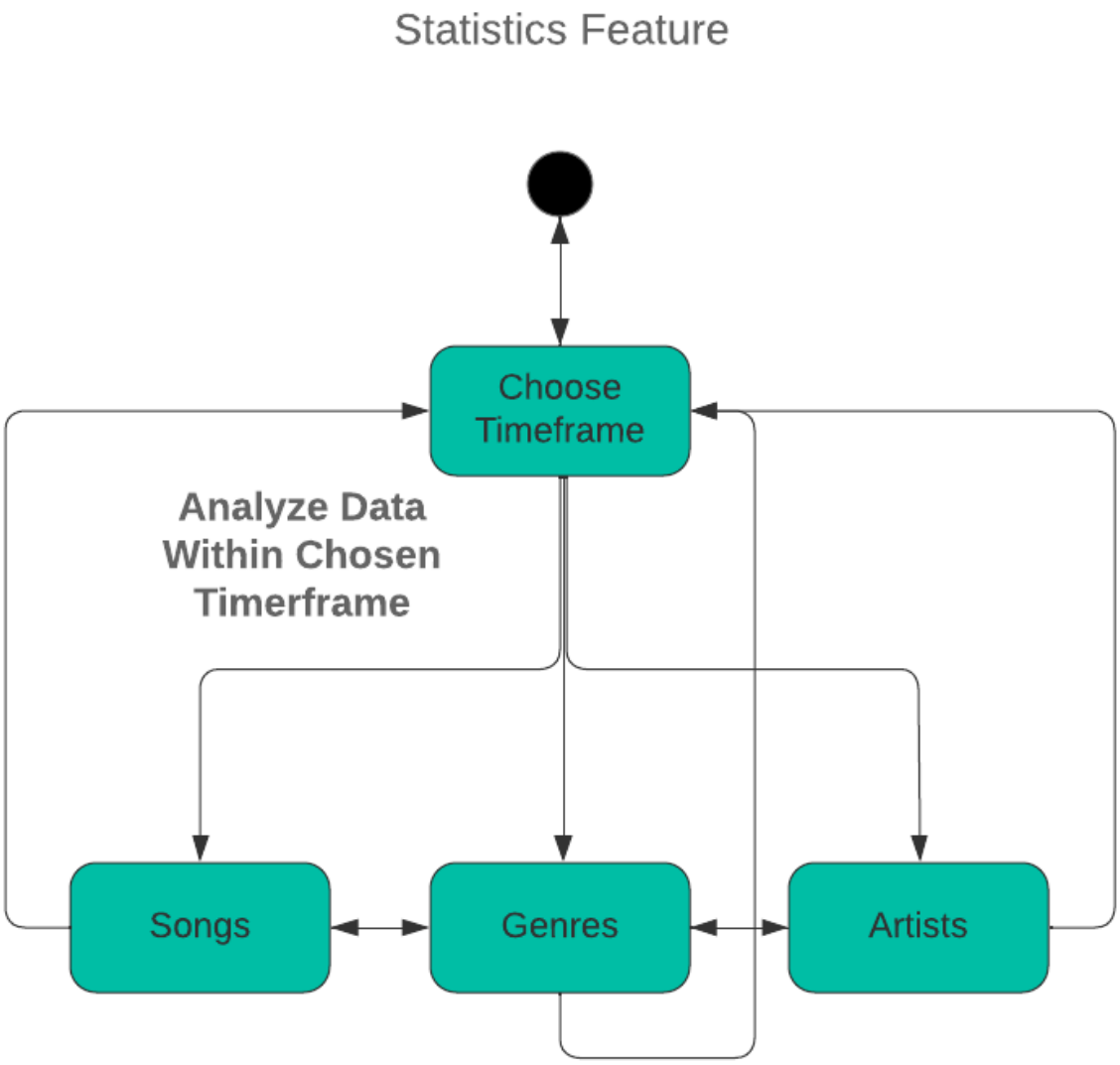


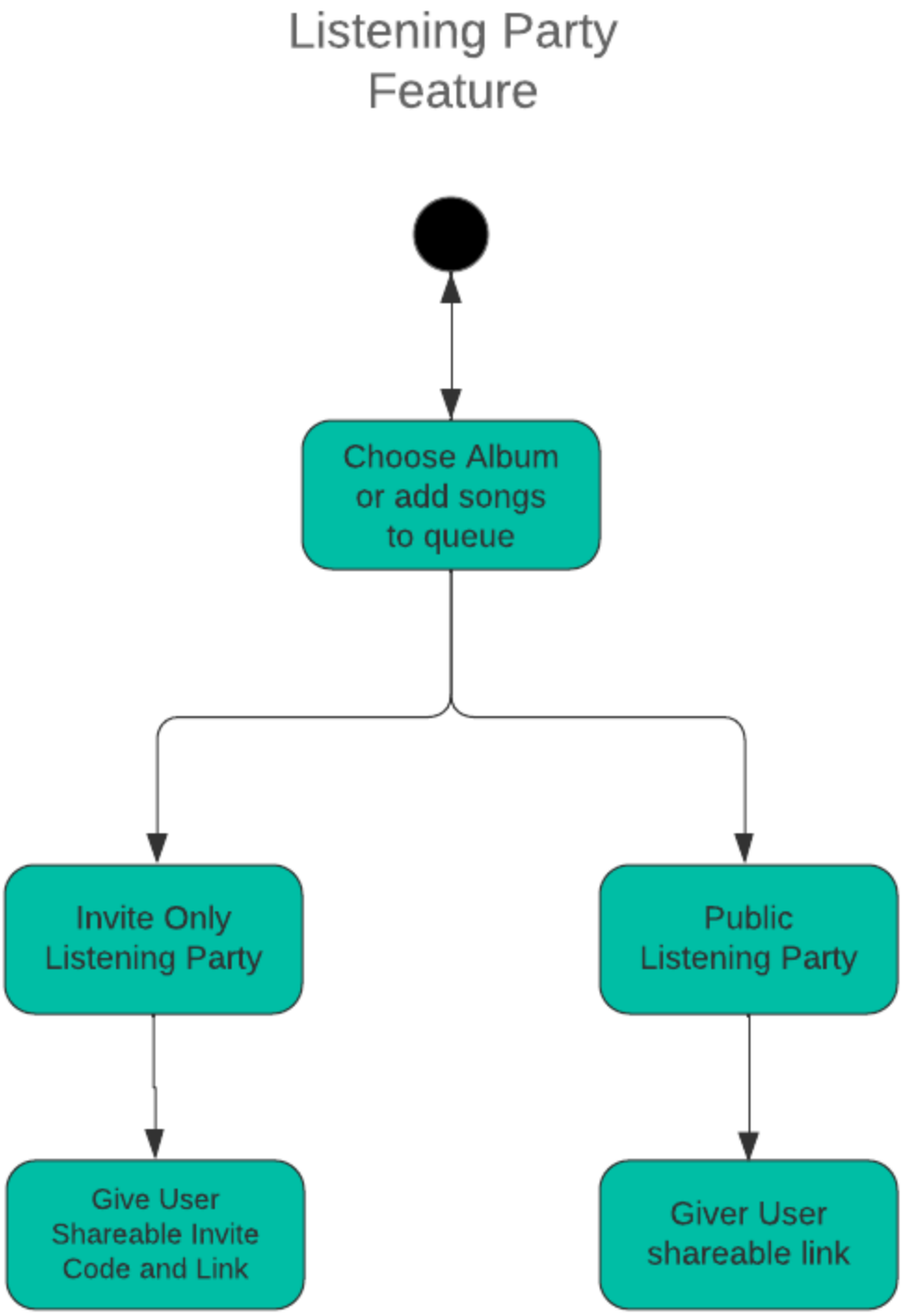
Behavioral Diagram



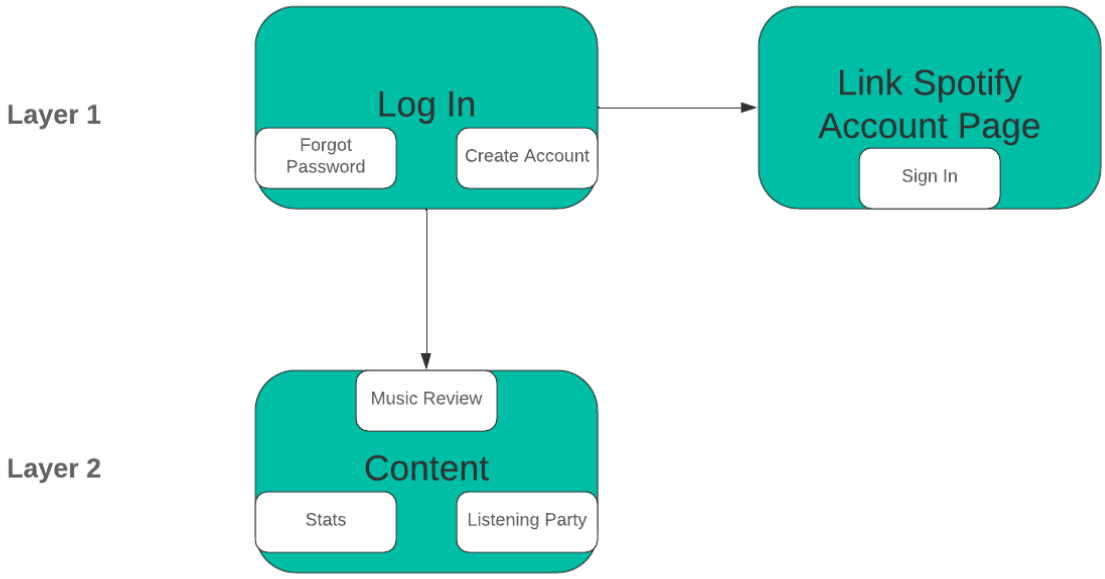
State Diagrams


Music Review Feature
Find Album
View Review
Post Review
Comment on Review
Rate Album
Create Review Description
Posting Comment
Posting Review


Statistics Feature
Choose Timeframe
Analyze Data Within Chosen Timerframe
Songs
Genres
Artists


Listening Party Feature
Choose Album or add songs to queue
Invite Only Listening Party
Public Listening Party
Give User Shareable Invite Code and Link
Giver User shareable link

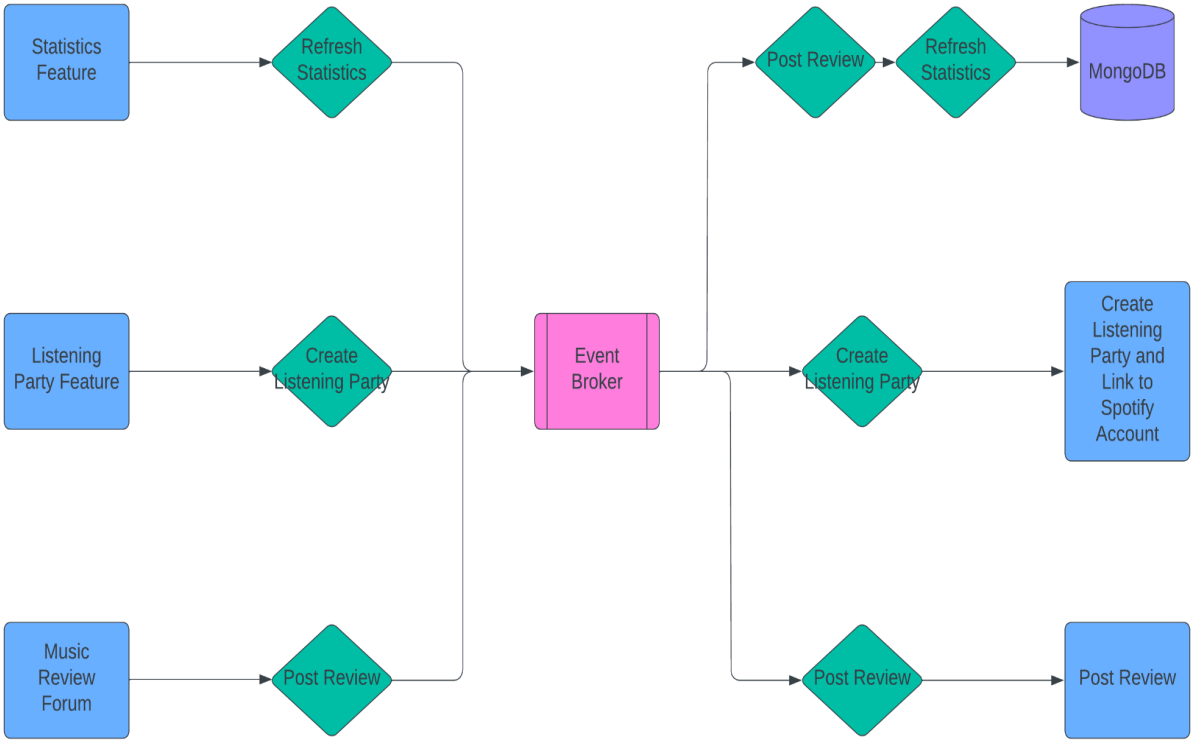
Storyboard Diagram



Architecture and Design

For this project I will be employing an “Event-Driven” architecture pattern. There are many reasons for choosing this pattern but there are three main points that pushed me towards choosing this pattern. The first and probably most important reason for me is that as of right now I feel that a lot of this project will be written using JavaScript and Python while also utilizing Node.JS. One of the main potential problems with this project is that there are a multitude of different events happening concurrently. This could in turn lead to scalability issues. With Node.JS however I can minimize this issue in an event-driven environment as Node.JS was created as an event-driven environment itself which means I can upscale easily by adding more instances and processes (Parmar, 2023). Both JavaScript and Python are event-driven as well, meaning their integration into the application will be seamless. This website will contain a multitude of extremely different features, thus by integrating Python with Node.JS I can mesh the scalability and efficiency of Node.JS with Python’s versatility and endless supply of libraries and frameworks (Hardy, 2024). While I don’t believe creating this project using only HTML/CSS and JavaScript is impossible, adding Python to the mix will make it much more feasible. Another reason for choosing the Event-Driven architecture is that it offers increased modularity (GeeksforGeeks, 2024a). This means the complex components such as the listening party feature or the review forum can be separated into much more maintainable and accessible modules to work on. This is necessary because the features for this website are so distinct from each other, I need to be able to launch certain features while others are still being developed. An event-driven architecture allows me to do this without ruining the user experience for other features (GeeksforGeeks, 2024a). Lastly, an Event-Driven architecture pattern allows for increased flexibility and responsiveness (GeeksforGeeks, 2024a). This is extremely important for this project as creating a seamless user experience is paramount to creating an accessible and enjoyable website. With so many distinct and complex features, processing these may cause delays and interruptions within the website. Thus by employing an Event-Driven architecture pattern I can minimize these potential adverse effects. Below is a simple example diagram showing what the flow of the architecture may look like for the main features. Obviously, this does not contain all the necessary function calls, etc., however it illustrates how an Event-Driven architecture functions in general.

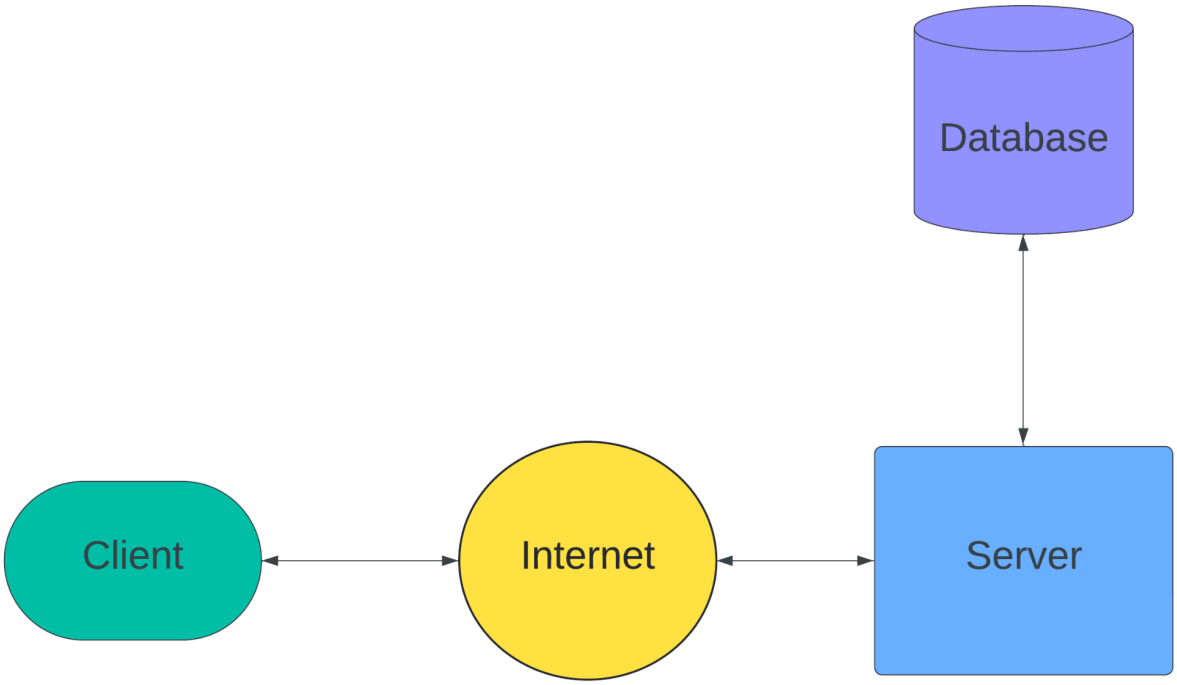
Example Event-Driven Architecture Pattern Diagram



What an Event-Driven architecture is takes some explaining. As seen in the diagram above, an Event-Driven architecture contains a process called an “Event Broker/Router”. The function of this process is to route the requests/events from users to the necessary systems. All events must pass through the event broker. As mentioned earlier and as shown in the diagram, everything within an Event-Driven architecture pattern is modular, which means that the event broker acts as a bridge of sorts to deliver the outputs from events to their necessary destination.

I will also be employing a client-server architecture for the development of this project. The project will be deployed on a server that hosts and delivers most of the events/requests from clients. As mentioned before, scalability can be a concern with this project. With a client-server architecture it gives me the ability to also scale horizontally as well as vertically (Terra, 2023). A client-server network model also enables a higher level of efficiency and reliability that other architectures can’t provide. As mentioned previously, there are features that require the system to be able to send and receive data simultaneously, with a client-server architecture this becomes possible (Terra, 2023). I have created a diagram below that provides a simple model of how the client-server architecture should function with the project.

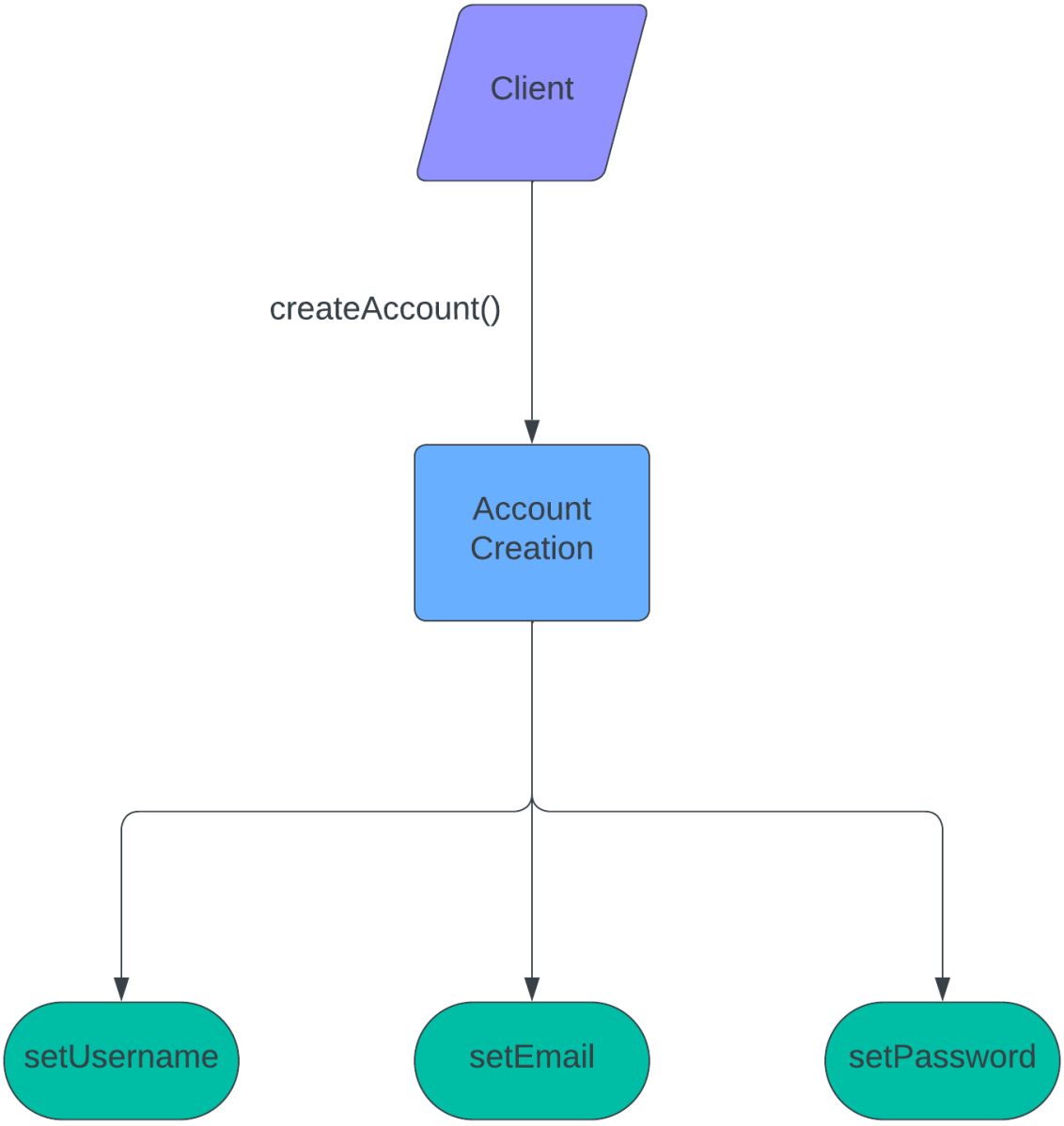
Client Server Architecture Example



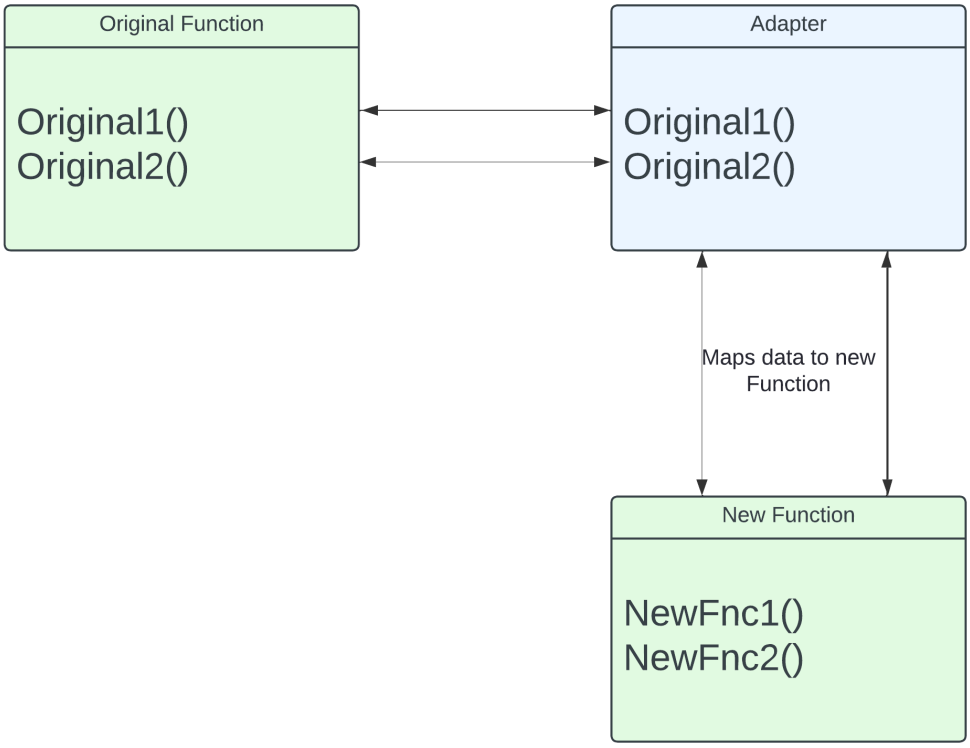
Design Pattern

The creational design pattern I want to implement for this project is the “factory method” design pattern. The reasoning for choosing the “factory method” design pattern is that because I will be creating objects in one place my code will be much cleaner and simple to integrate into other features (Cocca, 2022). For the structural design pattern I plan on implementing an “adapter” pattern. This is because I will be working with the Spotify API for many of the features on this website. The Spotify API may return data in a format that is not compatible with other aspects of my project. For this reason I will be employing the adapter pattern so that my website can process these differences in data on the fly, maximizing the user experience (Cocca, 2022). Lastly, I will be employing the “chain of responsibility” behavioral design pattern. This will keep my code efficient and simple to read, as event handlers will hand off the process output to the next event handler when necessary (GeeksforGeeks, 2024b).

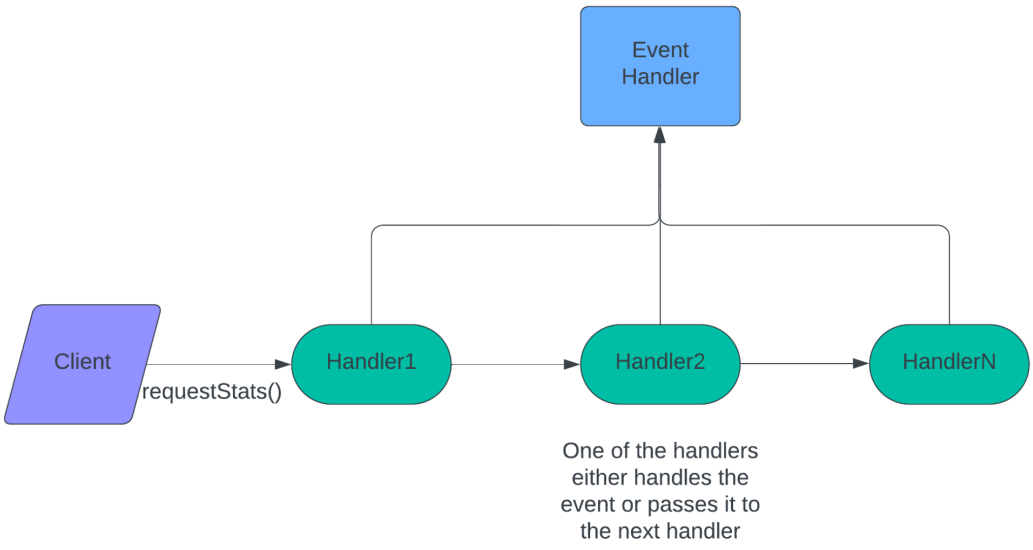
Factory Method Creational Design Pattern Example



Adapter Method Structural Design Pattern Example

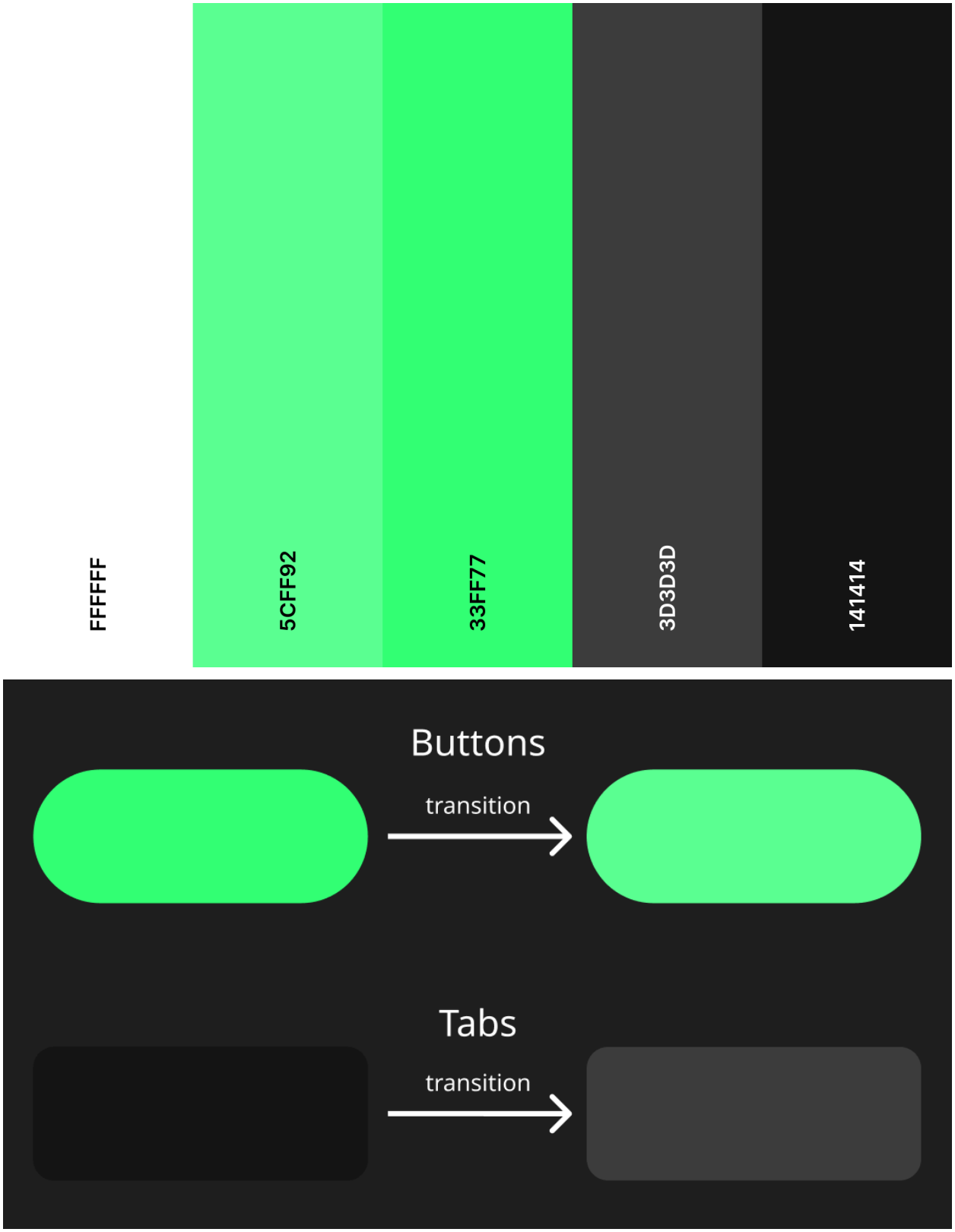


Chain of Responsibility Behavioral Design Pattern



UX Design

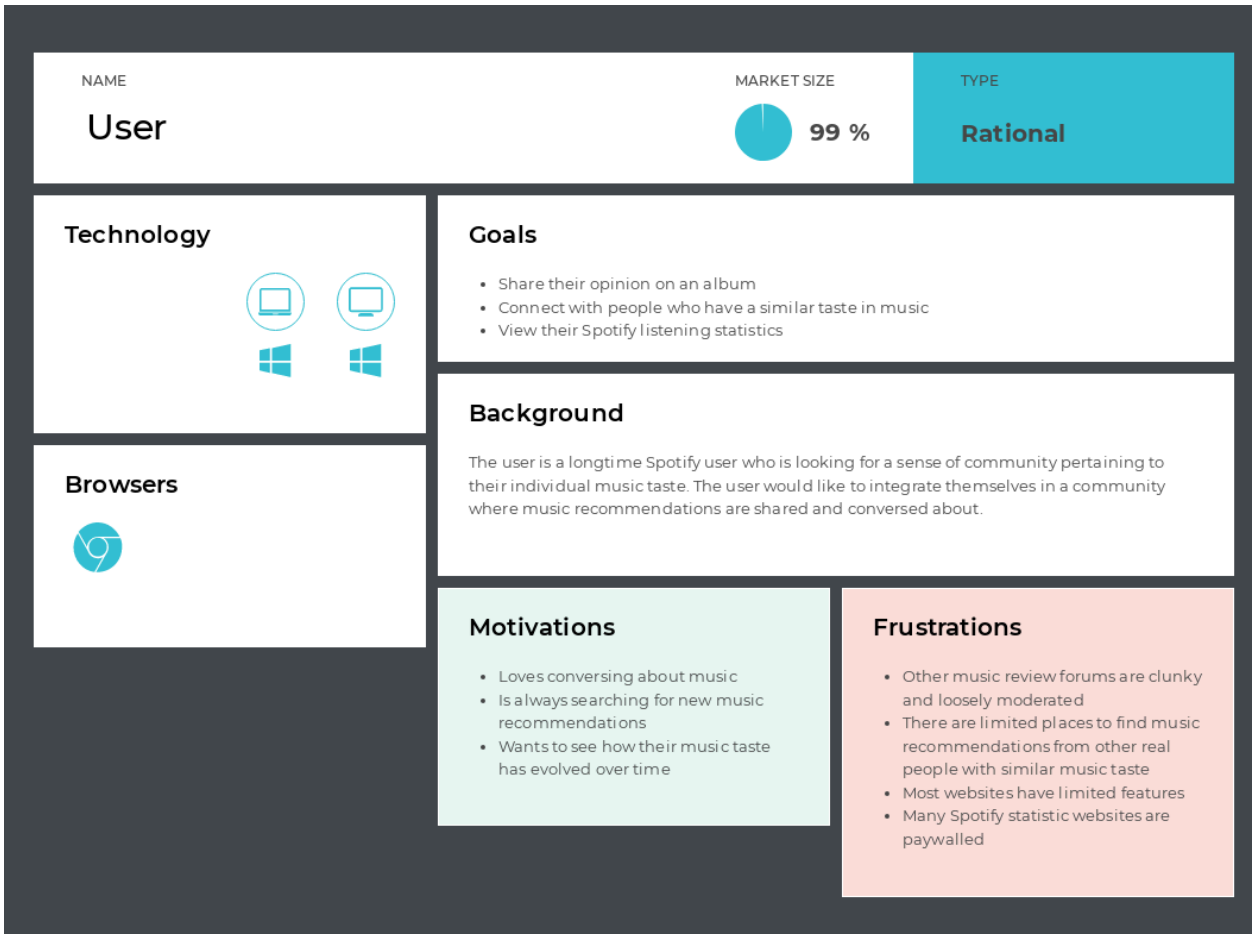
In order to create an engaging product for the consumer, having an intuitive and visually appealing UI/UX is extremely important to accomplish that goal. Aesthetics are largely a matter of subjective opinion, thus it is impossible to create an aesthetic that is visually appealing to everybody. However, something I have noticed throughout my computer science career is that having harmony within the design of the UI/UX is the key to creating a platform that spans most peoples aesthetic preferences. I also believe users are much more likely to weather issues with the website if it is visually appealing. So with that in mind, my goal with the aesthetic of my website is to prioritize the harmony between objects such as the background, buttons, tabs, etc. For the color palette I want to focus on mostly darker shades with splashes of brighter colors used as accents throughout the website. Because this website is being developed to be used in tandem with Spotify, sticking to the Black, Grey, White, and Green color palette with some tweaks is the plan as of now. Buttons will have transitions to lighter colors on the spectrum when hovered over to indicate to the user that the button is both functional as well as increasing the overall aesthetic of the website. Most of the website will consist of softer shapes. I plan on using harsh edges very sparingly in instances when they are needed to create contrast between items on a page, websites such as YouTube employ this strategy as well. In order to prioritize the user experience, I want items on a page to have as little visual weight as possible, which means the project will use a more minimalistic design. Users may become confused or stressed when there is clutter within the website so keeping pages as minimalistic as possible to keep the focus on the main features is key, an example of this can be seen with the Google homepage which solely focuses on the logo and search bar. Movement is another aesthetic function that I want to employ for this website. As stated previously, minimalism is a key aspect of this project's design, thus movement within the website will be used sparingly. I will use it mainly to draw attention to features on the website and to help user navigation. Put simply, websites with some movement are much more engaging than websites with static web pages. Lastly, I want the scale of items to be in line with their importance. For example, the main aspects of the listening party feature are the chat box and the album being played. Thus the chat box and the album cover will be the most prominent items on the page with items such as the host’s username and the next song being much smaller in scale due to their lack of importance. Below is an image of the potential color palette for the website as well as some early designs for buttons, transitions, etc.

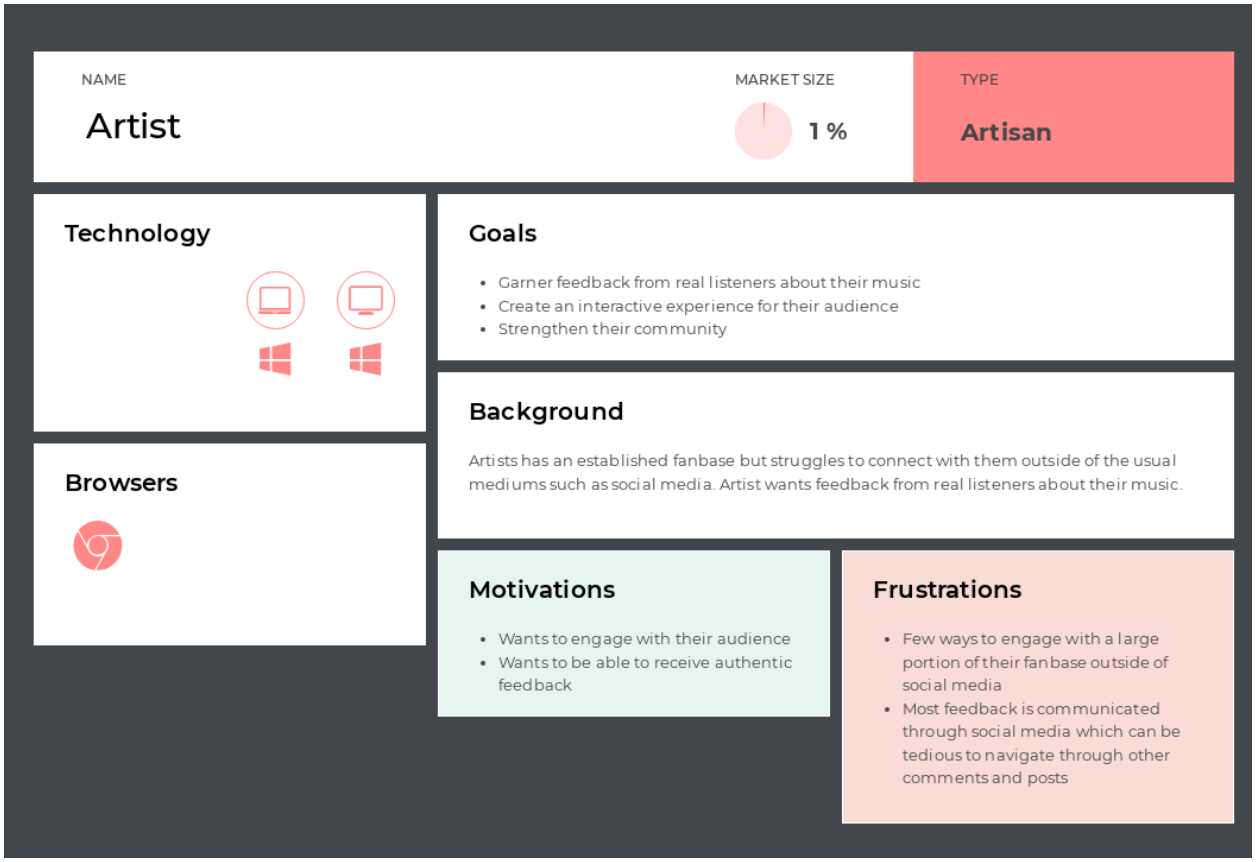

FFFFFF
5CFF92
33FF77
3D3D3D
141414
Buttons
transition
Tabs
transition

User Research

This project is designed to enhance the experience of Spotify users. So the stakeholders in this system will largely be those who are already using Spotify in some capacity. The two main stakeholders for this system will be the average user as well as artists. The average user may be someone who just wants to know their Spotify stats. However, the average user may have more complex needs such as the user has an opinion on an album that they'd like to share, or they are looking for a community of people with similar music taste. Because many alternatives are either inefficient and clunky or only contain limited features, this website will allow the user to solve their needs efficiently and in one place. On the other side of the coin, this website could be a great tool for artists looking for feedback or to strengthen their community. The review forum on the website will allow the artist to gain real feedback from real listeners, as well as to converse with other users about their work. The listening party feature will allow artists to host their own listening party for either new or preexisting material to create an interactive experience for their fanbase.

User Persona





UX Frameworks

The UX Frameworks I plan on integrating throughout the development of this project are the Design Thinking Process and the User Centered Design frameworks. The Design Thinking Process consists of 5 different phases (Flowmapp, 2024). The first phase is to empathize with the user to foresee some potential struggles or issues they may have while perusing the website. This allows me to anticipate future problems while designing the UX. The second phase is to define what issues the users defined in our personas may need and how they should function. For example, the average user wants to find a sense of community based on their music taste, thus making sure communication between users is efficient and simple is necessary to satisfy this issue. The next phase is to ideate on how I can solve problems garnered from the user personas. The communication issue I posed previously can be satisfied by the creation of the commenting and direct messaging features. After ideating on how to solve user issues, creating a prototype to better visualize these issues come next. Some flaws become much more obvious once you actually try working around them. After creating a prototype, testing new patches to user issues is the last step in the process. As mentioned earlier, I will also be integrating the User Centered Design framework. The Design Thinking Process and User Centered Design frameworks are extremely similar in their philosophies, where they differ is that the User Centered Design

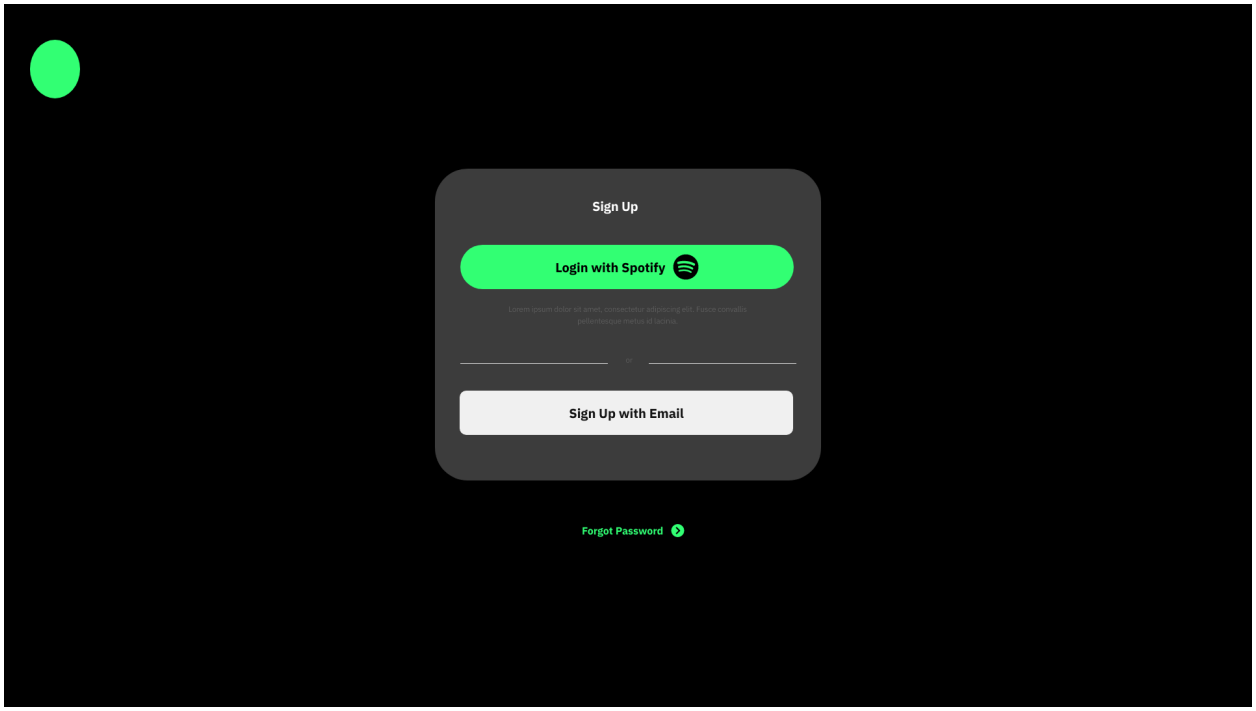
framework places an emphasis on continually improving the experience for users after deployment (UXPin, 2024).

UX Architecture and Design

In order to visualize every possible path for users to take on the website, I will be using the Information Architecture to complete this goal. By creating a map of the website infrastructure I can more efficiently diagnose certain problems depending on where they occur as well as making sure the flow of the website remains intuitive and simple (Eppy, 2022). The map developed during this process also helps serve as a springboard for implementing new features in the future (Eppy, 2022). The design pattern for this website can be broken up into three pieces, the input and output, the navigation, and the content structuring. For the input and output, there will be text fields to submit text when necessary such as when writing a review/comment or when creating an account. Buttons and tabs will also have visual transitions as illustrated above indicating to the user that they are functional. Secondly, navigation will be heavily prioritized throughout the design of this project. Because I am enlisting a minimalist approach to this project's UX I want to keep the amount of tabs limited to only the necessary functions, with smaller but still easily viewable icons for pages such as the profile or setting page. Lastly, the content structuring of this website will be heavily modular but still extremely intuitive. Settings pertaining to certain features will only be available within the page for those certain features in order to limit confusion when navigating other features.

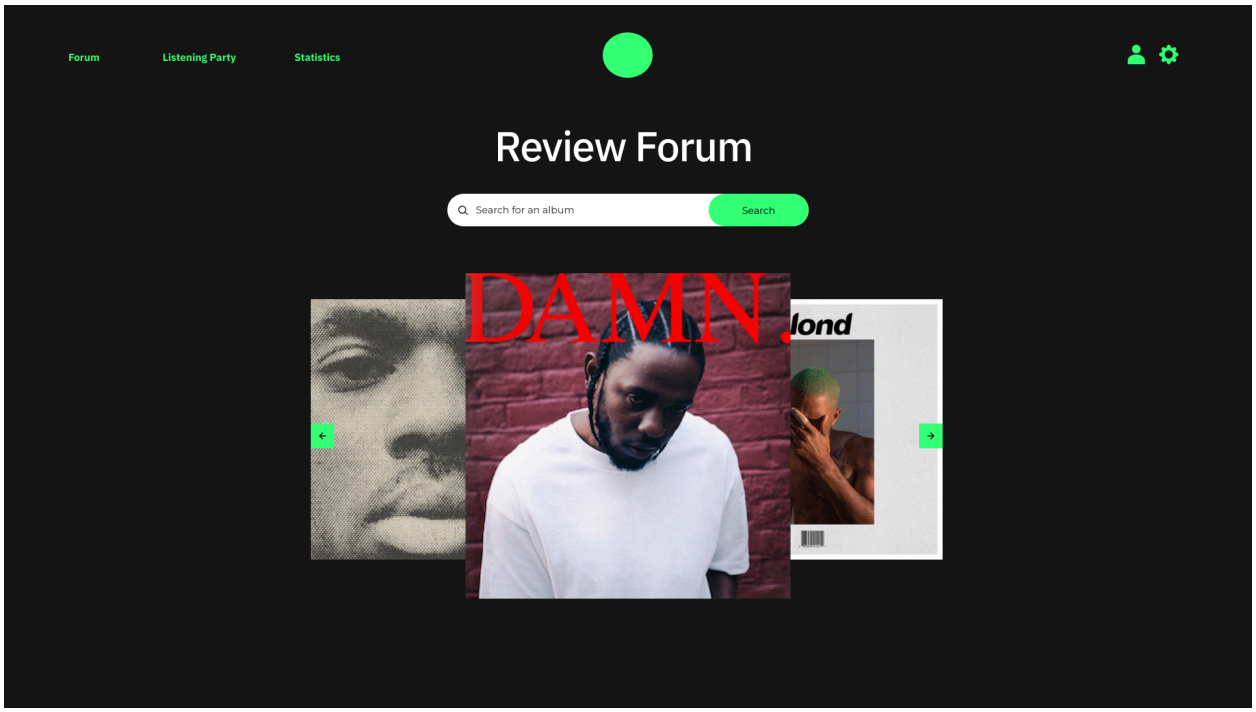
UX Mockup

Login Screen

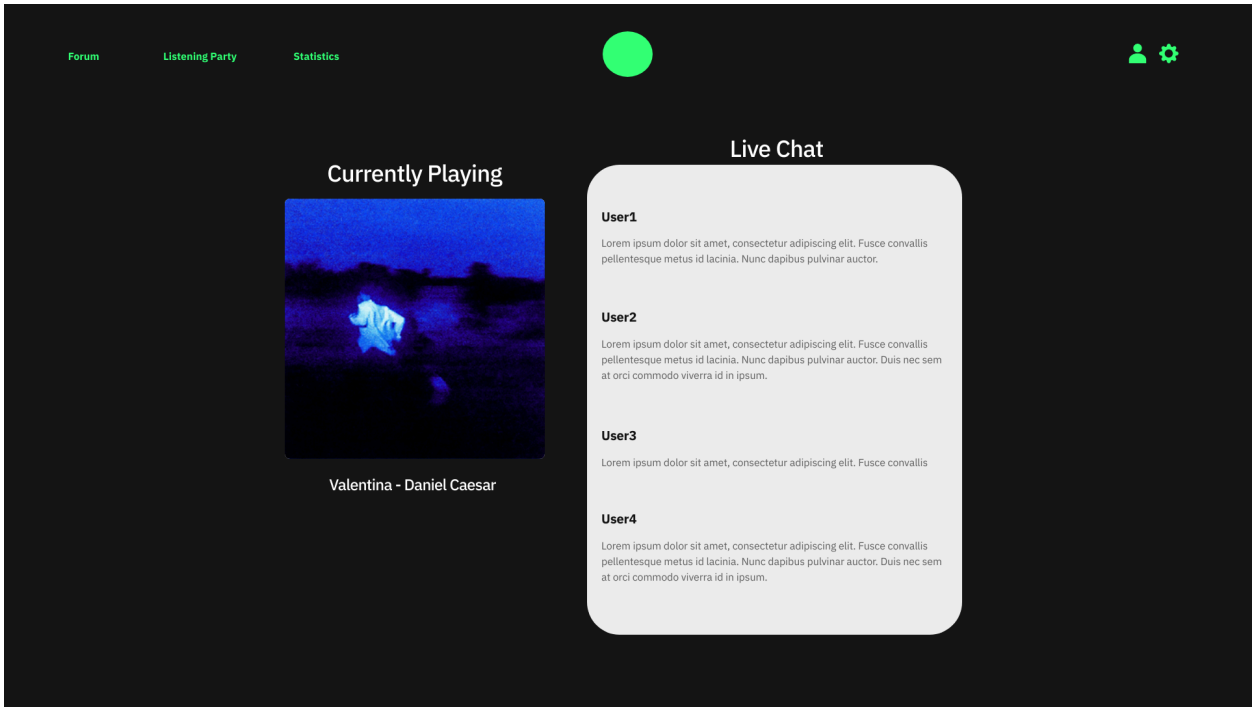


Login Screen 2

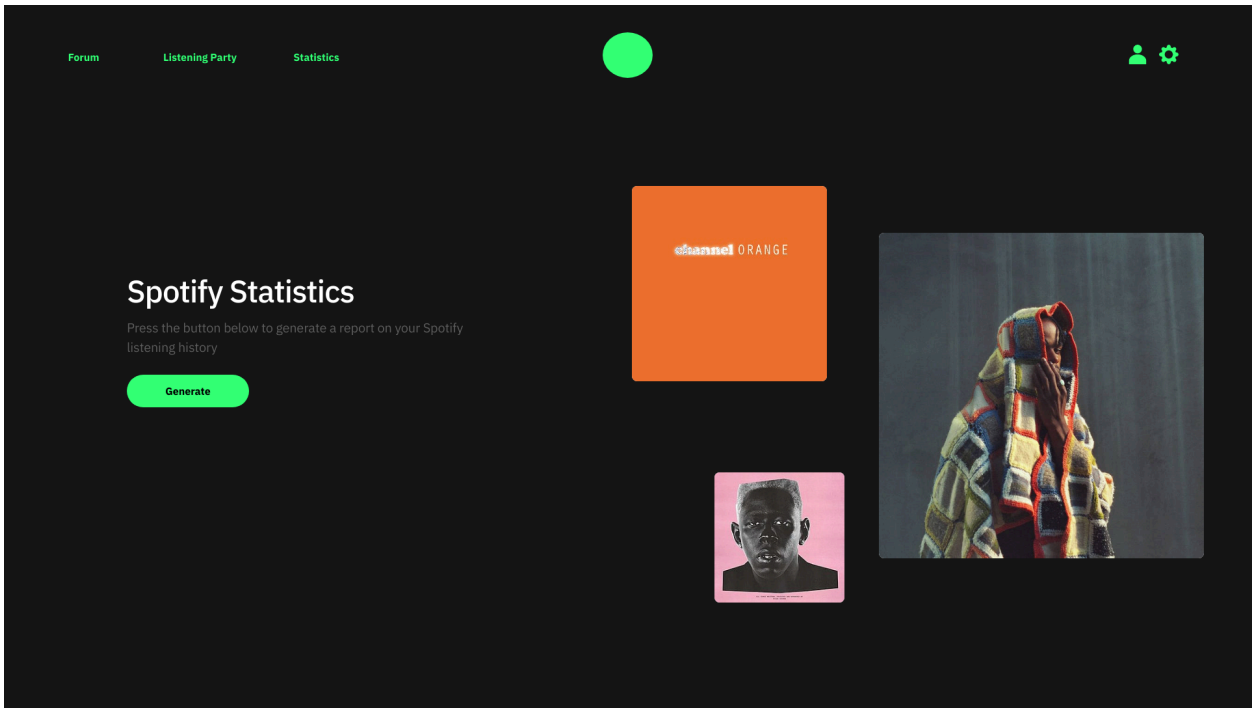
Review Forum (Home Page)



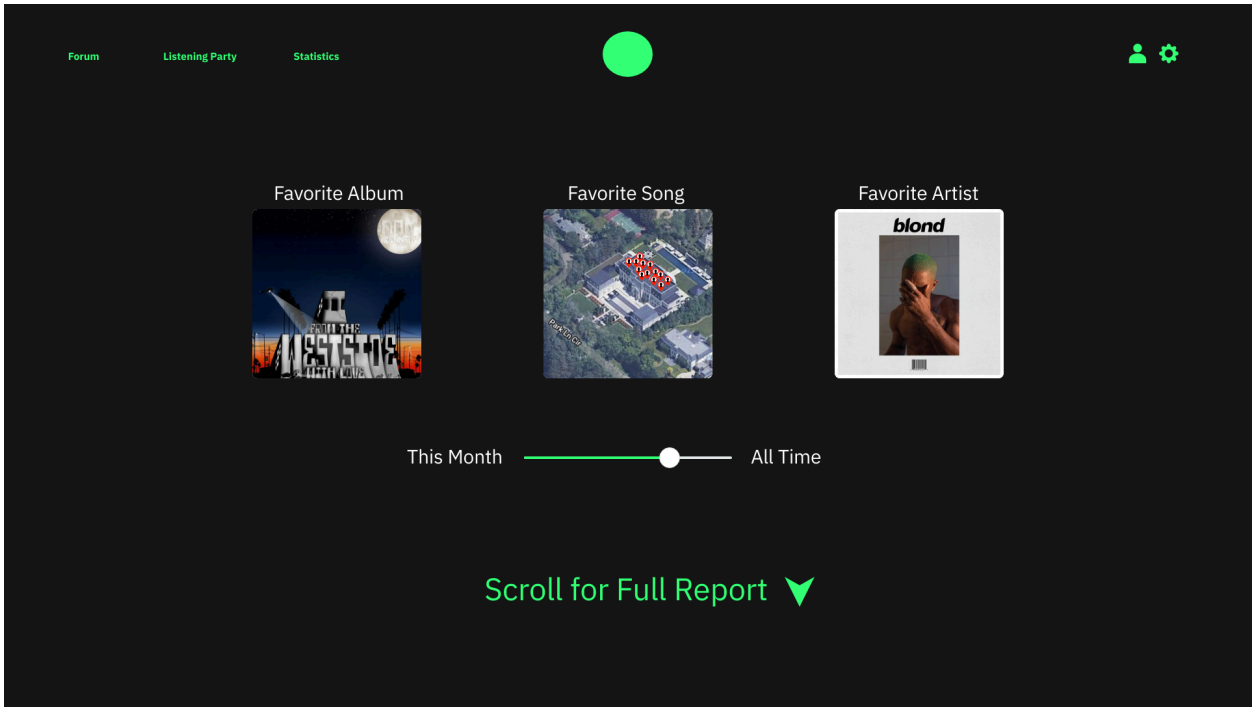
Listening Party Feature Rough Design



Stats Origin Page



Generated Stats Page



Prototype

Submitted with 490 project proposal doc on Canvas

GitHub Link

https://github.com/zacahryy/CapstoneFinal.git

Test Plans and Results

Unit Tests

In order to determine whether the user is receiving the proper outputs from the application I created a set of unit tests to determine whether or not the application is working correctly. I also integrated some unit tests throughout the entire program. For example, in order for a page to load it must receive a proper HTTP response (200, 201, 204) depending on the item being rendered, anything else will throw an error. Thus by simply navigating the website I can see that it passes all rendering unit tests. I still defined some user tests to showcase the website's functionality.

Rendering Unit Test Example

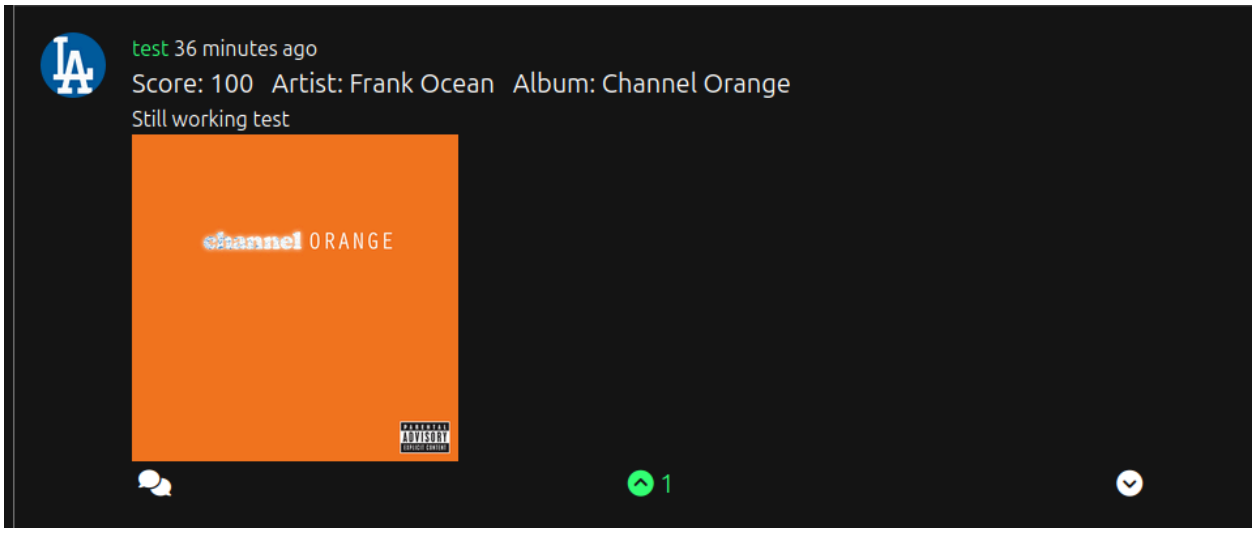
```
router.get("/", (req, res, next) => {

    var payload = createPayload(req.session.user)

    res.status(200).render("searchPage", payload);
})
```

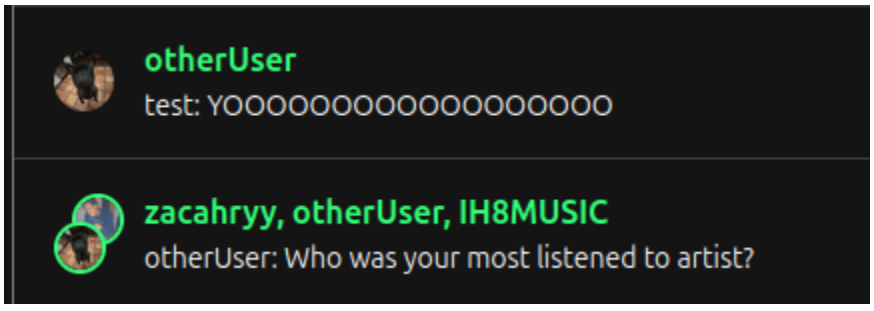
Posting Unit Test

The user should be able to create a post with a score, artist name, and an album name. The post will then be inserted into the database and rendered for the user and anybody that follows them. The user should also be able to upvote and downvote posts, this number will be displayed along with the post. As you can see below this functionality works in the application.



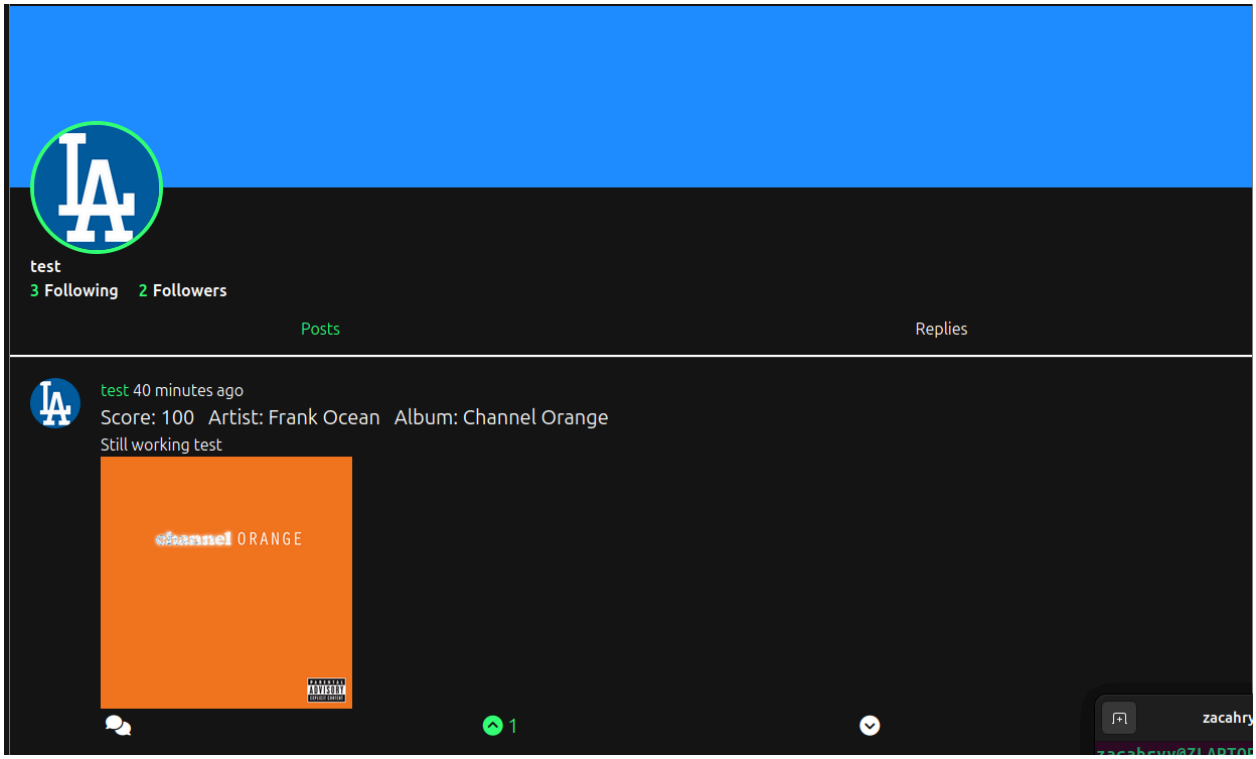
Direct Message Unit Test

The user should also be able to send direct messages to other users. As you can see below this functionality exists as well.



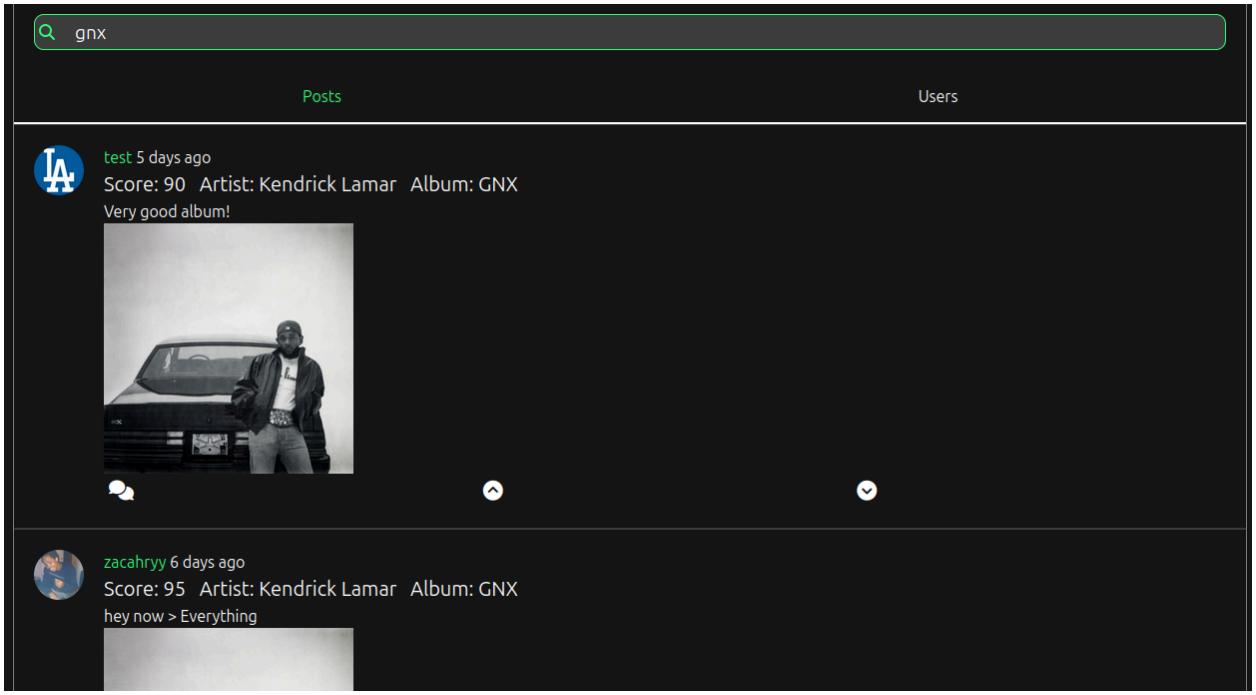
Profile Unit Test

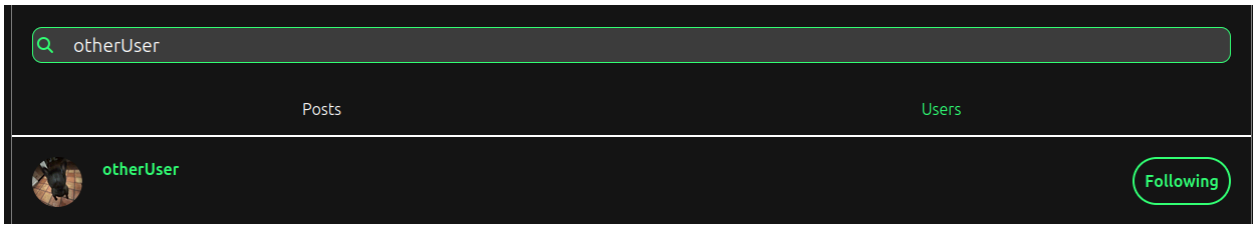
When navigating to a profile page users should be able to see the selected users follower and following count, their posts and replies, as well as a follow button. If the user is viewing their own profile, the user will not see a follow button but they will be able to change their profile picture and profile banner. Below is an example of this, better evidence of this unit test will be provided in the demo video



Search Function Unit Test

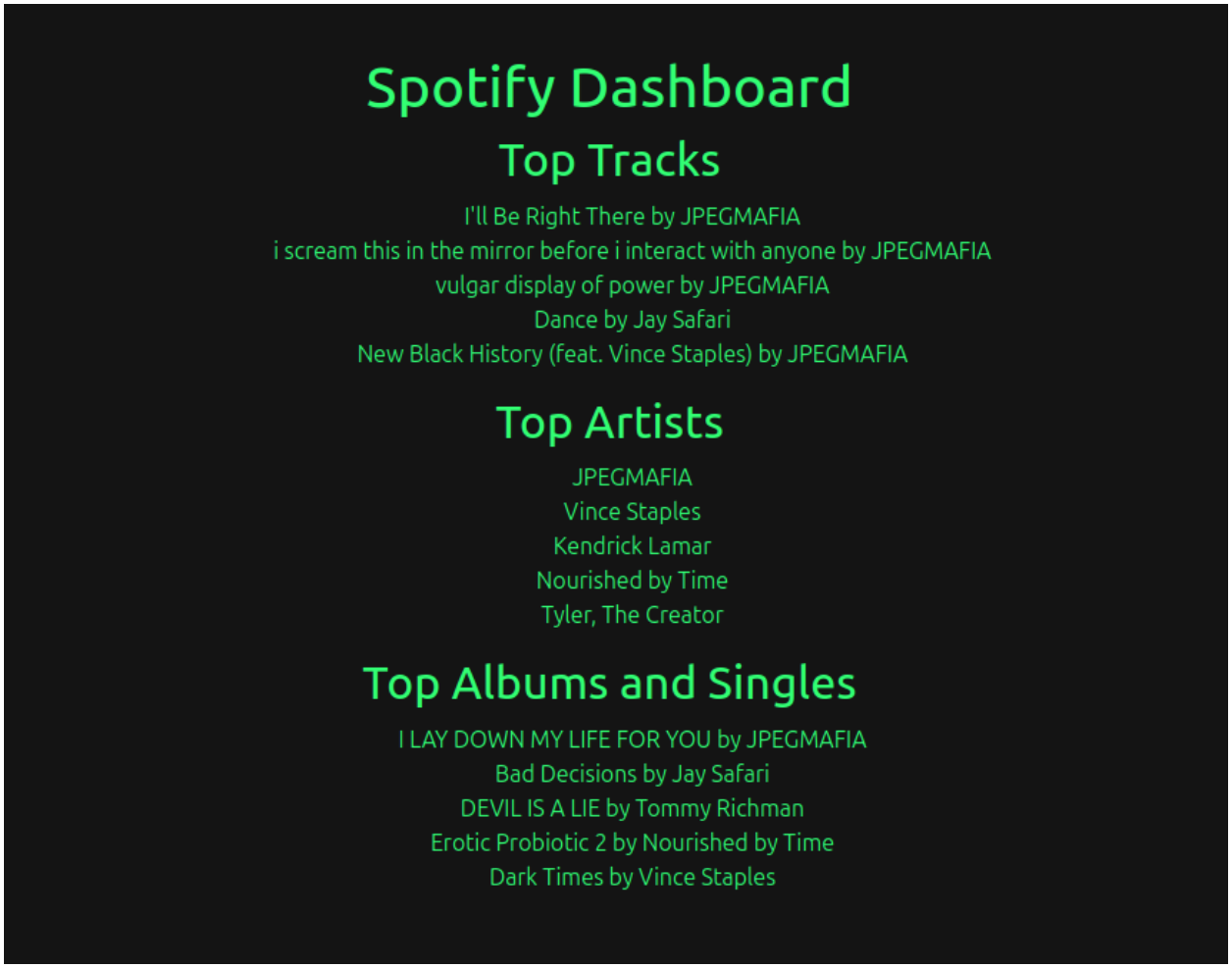
Users should be able to search for posts and other users on the website. The user can then navigate to another user's profile via the posts or user tabs if they feel inclined to do so. A better demonstration of this unit test will be shown in the demo video.





Spotify Statistics Unit Test

By clicking on the spotify logo on the homepage, the user should be prompted with the spotify login screen, a successful login should show the listening statistics for there account from the past 6 months.



Performance Tests

Because this application needs to handle multiple users at once, I tested the platform using the direct messaging feature, showing that the application can handle multiple users sending and receiving requests to the MongoDB database. I have attached a video below showing this test, where I queried the database rapidly using multiple users.

PerformanceTest.mp4

Speed and UI Tests

The goal of this test is to make sure that the user can progress through the application smoothly, and the webpage should be responsive and not lag when resizing the window or navigating to new pages. I have attached a video below demonstrating this test. The video proves that all navigation buttons on the website are fully functional and redirect the user to the proper page. With functions like loading album covers or getting spotify statistics (spotify log in page didn't appear because my log in data was already cached) there is a miniscule delay as the website has to ping the Spotify API and await the response, however this delay only affects some cosmetics for a fraction of a second.

SpeedUITest.mp4

Security Tests

In order to navigate to other pages outside of the /register and /login pages, the user must be logged in. I accomplished this by creating a middleware function that checks if the user is logged in and there has been a session created, if both these parameters are true the user can navigate the website freely, if either of them fail the user is redirected to the login page so they can either log in or create an account. Below are some screenshots of the different pages the user can navigate to, the middleware function that checks for the logged in user, and a video showing the require login function in action.

Routes

```
app.use('/login', loginRoutes);
app.use('/register', registerRoutes);
app.use('/posts', middleware.requireLogin, postRoutes);
app.use('/auth', authRoutes);
app.use('/spotify-dashboard', spotifyDashboardRoutes);
app.use('/profile', middleware.requireLogin, profileRoutes);
app.use('/uploads', uploadRoutes);
app.use('/search', middleware.requireLogin, searchRoutes);
app.use('/messages', middleware.requireLogin, messagesRoutes);
```

Middleware Function

```
exports.requireLogin = (req, res, next) => {
    if (req.session && req.session.user) {
        return next();

    }else{
        return res.redirect('/login');

    }
}
```

Navigation Security Test

This video shows me attempting to navigate to different parts of the website using the url, as you can see users will always be redirected to the login page if they have not logged in no matter what. Once the user is logged in, they can navigate freely.

SecurityTest.mp4

SQL Injections and Password Hashing Test

Because I am using MongoDB for this project, which is a NoSQL database, it is resistant to SQL injections. Even if a user were to enter MongoDB commands, the data is saved as a string in most cases, meaning MongoDB does not parse the data, thus the commands would not work. I also hash all user passwords entered into the database using Bcrypt, thus even in the instance an attacker was able to gain access to the database, the passwords would be useless anyways. Below is an example of the parameters I set for the register function that helps prevent this attack.

```
    var data = req.body;

    data.password = await bcrypt.hash(password, 10);
    User.create(data)
    .then((user) => {
        req.session.user = user;
        return res.redirect('/');
    })
```

Test Cases

I have defined some test cases below to give some examples of what users can do and how the website functions at a base level.

Test Case #1
Test Scenario: Registering a new account
Steps:

User clicks on “Need an account? Register here.” button
User enters a unique username
User enters a unique email
User enters a password and confirms it
User is redirected to home page upon successful account creation

Prerequisites: Unique username and password
Expected Results: Entered data is saved within the users collection in the database using the user schema. If this succeeds the user is redirected to the home page.
Test Status: Pass

Test Case #2
Test Scenario: Logging In
Steps:

User enters username
User enters password
User is redirected to the home page if login is successful

Prerequisites: Username and password
Expected Results: Entered data is matched with preexisting data within the users collection, if there is a username match, the hash of the entered password is compared with the hashed password associated with the user in the database. If both of these conditions pass the user is redirected to the home page.
Test Status: Pass

Test Case #3
Test Scenario: Creating a Post
Steps:
- User enters score
- User enters artist name
- User enters album name
- User enters review

Prerequisites: User is logged in
Expected Results: If all of these fields are filled, the post will be created and be displayed for the posting user and anybody that follows them. If the album name is spelled correctly (case does not matter) the application will ping the Spotify API and insert the album cover image into the review.
Test Status: Pass

Test Case #5
Test Scenario: Replying to a Post
Steps:
- User clicks on reply button
- User enters reply into textbox
- User clicks “Save Changes”
- Reply is posted and page refreshes
- Reply is now attached to the post and displayed

Prerequisites: User is logged in
Expected Results: When a user replies to a review it will be linked to that review and appear if another user clicks on the main review
Test Status: Pass

Test Case #6
Test Scenario: Creating a Direct Message
Steps:
- User navigates to inbox page
- User clicks on “Create new message” button
- User enters the usernames of other users they want to add to message
- User creates chat
- Users are then free to message each other

Prerequisites: User is logged in
Expected Results: A new message will display within the inbox page for all users added to the chat.
Test Status: Pass

Test Case #7
Test Scenario: Using Search Feature
Steps:
User navigates to search page
User clicks on either “Posts” or “Users” (posts is selected by default)
If posts is selected, the user enters either the album name, artist name, or they can search by the review content, if users is selected enter the username
Application displays results related to search
Prerequisites: User is logged in
Expected Results: Depending on which part of the search feature is selected, the webpage will display results according to input.
Test Status: Pass

Test Case #8
Test Scenario: Getting Spotify Statistics
Steps:
User clicks on Spotify Button
User enters their Spotify account information
Application displays listening history results from the past 6 months
Prerequisites: User is logged in, User has a Spotify account with listening history
Expected Results: When the user clicks on the Spotify button they will be prompted with the Spotify login screen, if there login is successful the application will load the users listening history results from the past 6 months.
Test Status: Pass

Test Case #9
Test Scenario: User Edits Profile Images
Steps:
User clicks on profile icon in top right corner of application
User hovers over either their profile picture or profile banner
User clicks the camera button
User uploads and crops picture
Related attribute is changed in database
Prerequisites: User is logged in
Expected Results: When the user navigates to their own account page they should have the ability to change their profile picture or banner using Cropper.JS, the changes will then be uploaded to the database.
Test Status: Pass

Bugs

- User profile picture not displaying when replying or viewing replies
- Spotify Stats not appearing
- Direct Messages sticking to one side
- Direct messages not displaying for every user
- Album covers not loading for posts
- Replies having headers when they weren't supposed to
- Follower count going up by one every time the user refreshes

Above are a handful of bugs that I encountered during development, these aren't close to every bug I encountered but I have listed them as they were the most tedious to fix.

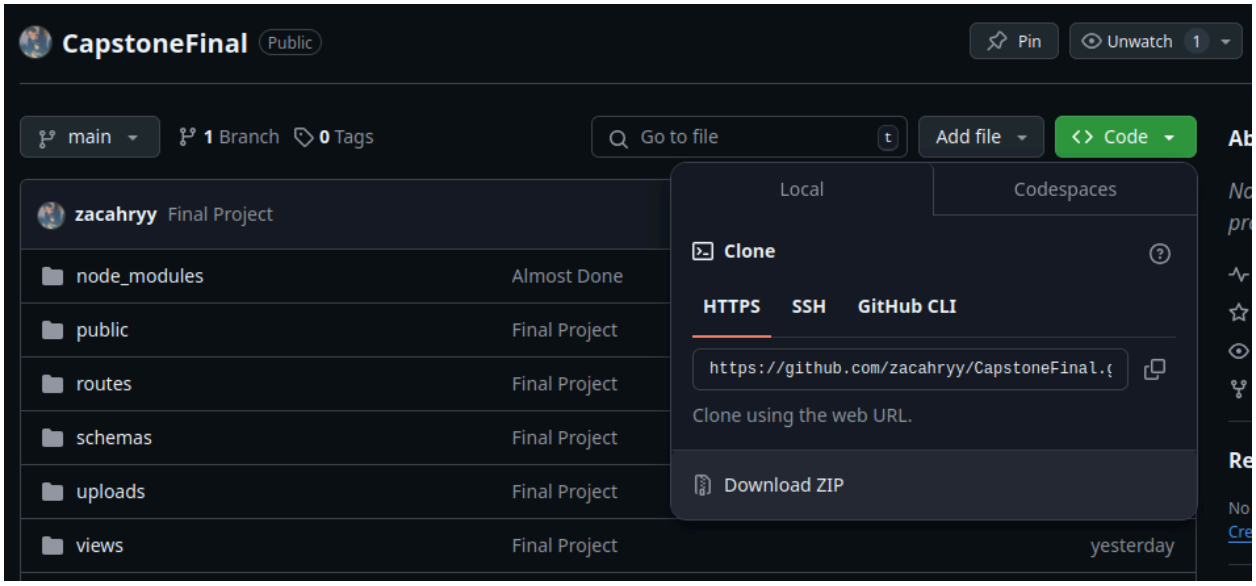
Installation and Setup Guide

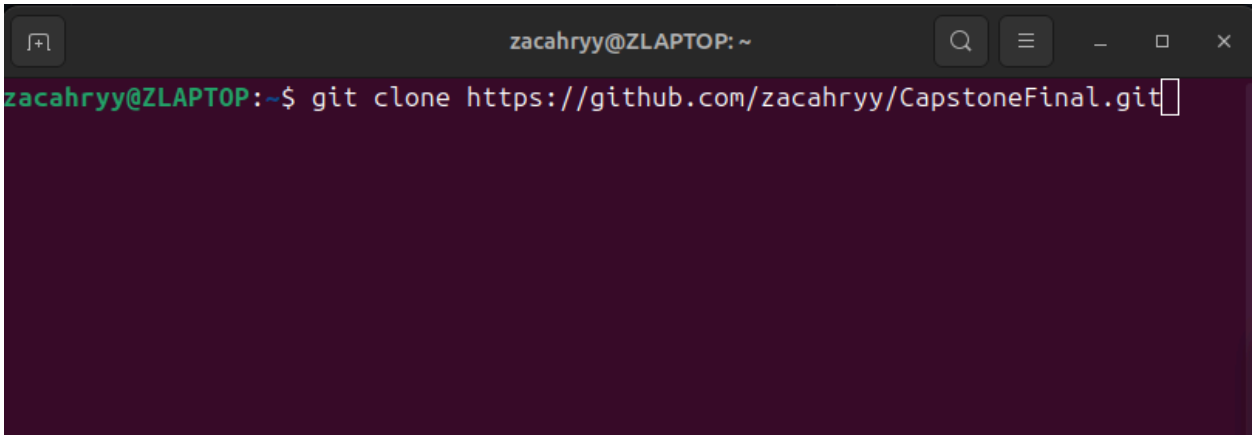
Step 1

The first step in setting up this project for use is making sure you have Node.js installed on your machine. You can navigate to https://nodejs.org/en to download the runtime environment.

Step 2

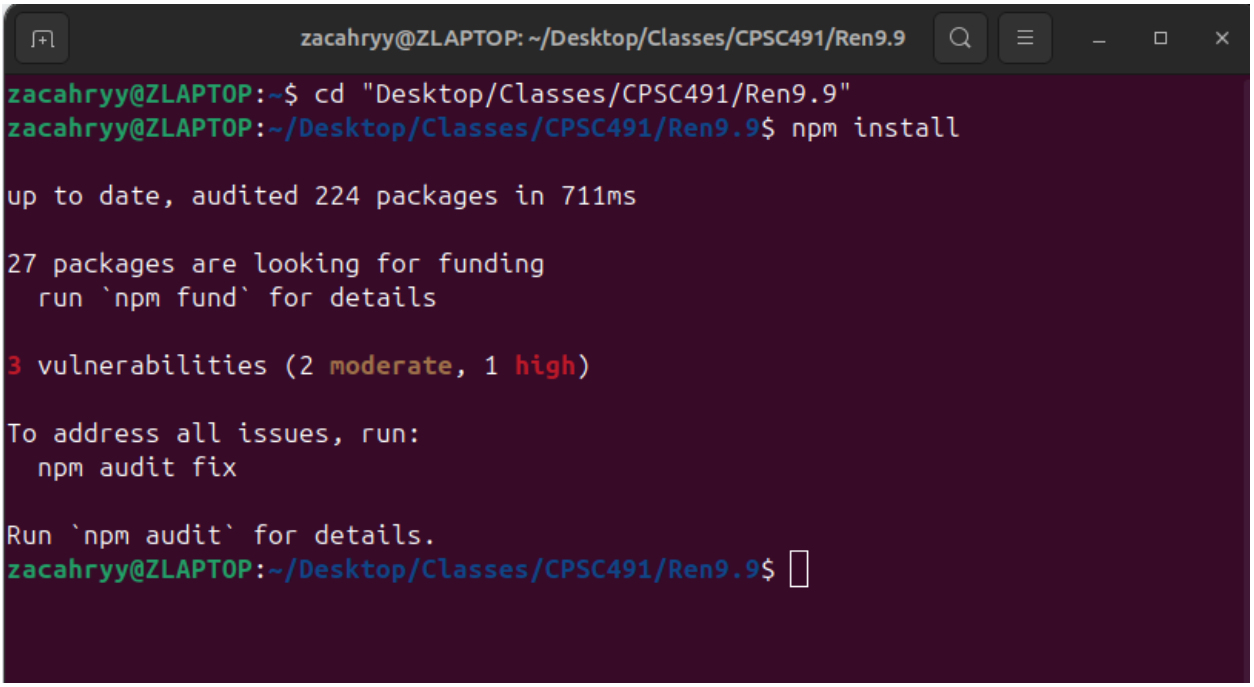
The next step would be to either download the .zip file from the github provided above or if you have git installed use the command “git clone <repo link>” in the desired folder.





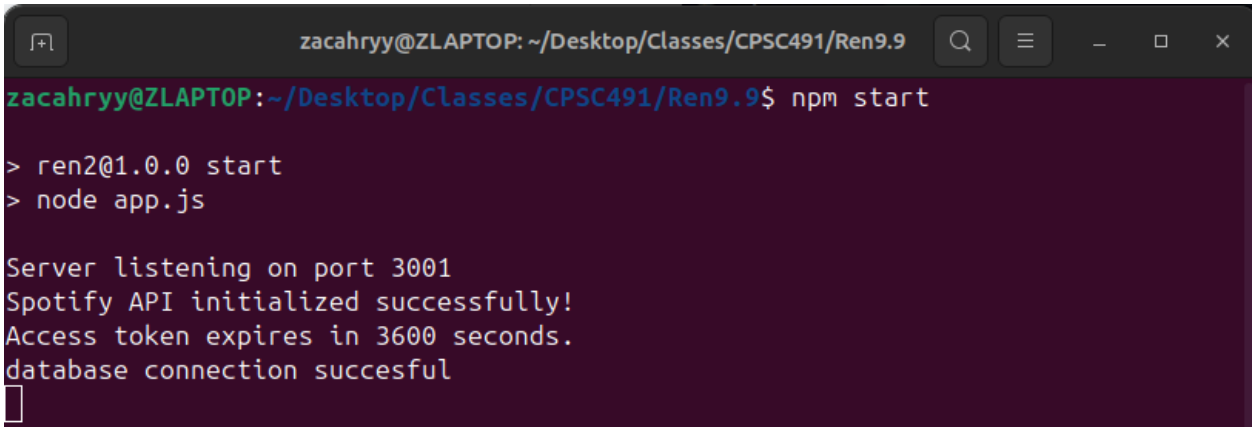
Step 3

You can then cd into the folder you have placed the project, and install the necessary dependencies by using the command “npm install”, you must have Node.js installed in order for this to work.



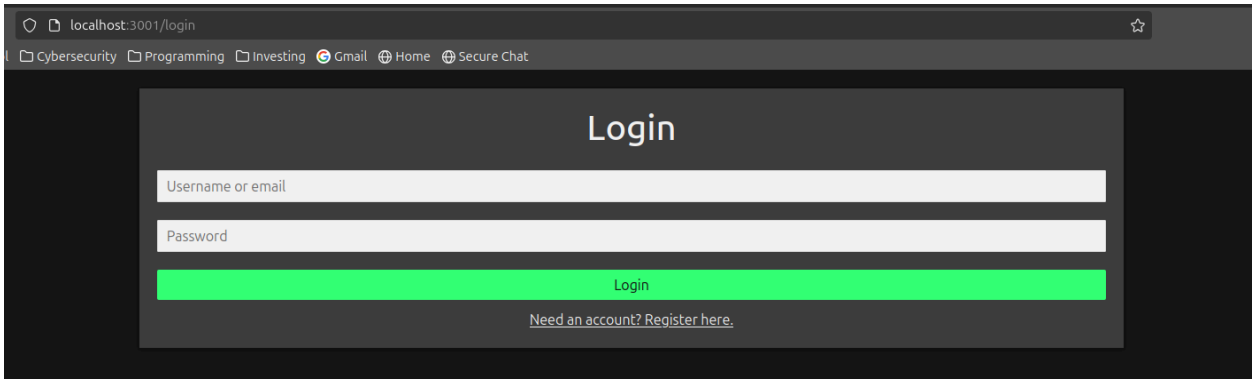
Step 4

Lastly you can run the program using the command “npm start”. This will start the application. As you can see, no database setup work is needed because I am using a MongoDB Atlas Cluster that is already deployed. The application runs on localhost:3001.



Step 5

Navigating to “localhost:3001” will take you to the login page for the application, the application is fully functioning at this point.



Run and Operation Books

How to Deploy Project

The application currently only runs locally at the moment. However, deploying the application should be quite simple as the database is already deployed separately. All that is needed is to sign up using a website hosting service. There will be some bugs with the SpotifyAPI, as they require you to define the urls for your website in order to use the API, however the fix is as simple as copying the new website url and callback url and pasting them where they are needed. When determining where to host, security considerations such as using HTTPS should be a focal point.

DDoS Recovery

Because of limited funds, I cannot host this website on my own server. Thus as mentioned previously I would have to use a website hosting service. This would mean I would have very little power in preventing a DDoS attack, as I am only a customer of the service being used and would have to wait for the service to fix the issue. To somewhat remediate this, I would keep a clear line of communication updating users on the status of the website via other social media services.

Database Breach

If the database is breached, this would indicate the password for the database is compromised. My next steps would be to block all IPs except my own from interacting with the database, and to change the password for the database. This would mean service downtime, however I feel this is necessary to protect user information. Because of this possibility, all passwords are hashed before being entered into the database as a layer of protection.

Disaster Recovery

If this plan is used, it indicates that something catastrophic has happened to the servers hosting the website or in a hypothetical world the workplace if employees were ever hired to work on this project. Depending on the severity of the damage to the servers, there are multiple different paths I can take. If the service hosting the website relays to me that the downtime will be extensive, I can redeploy the site using the most recent GitHub release on a new website hosting platform. As for the hypothetical employees, prioritizing their safety and well-being would be paramount to shortening the downtime.

Business Continuity

The business continuity plan consists of keeping the downtime to a minimum. As mentioned previously, prioritizing the well-being of the hypothetical employees would come first. This would mean either creating a temporary workspace or switching to work from home until the original workspace is fixed or a permanent new one is found. For the website downtime, I would like to keep it to a few hours at the max. If the hosting service explains that the downtime will be longer than around a day. I will consider my options pertaining to deploying using a new service. Ideally throughout this whole process I will be giving status updates via another social media platform to our users.

Backup/Restore

Because this is a solo project and I have limited funds, I cannot create backups for the database as these would drain my bank account. Ideally, depending on funding, I would be able to create viable backups via MongoDB on a set interval (i.e Once a Week, Once a Day, Once an Hour, etc.). In the instance of the current database being compromised, I could switch to the most recent viable backup.

Service Level Indicators

My request based service level indicators would be availability, request latency, and error rate. These metrics allow me to create a picture on whether or not the performance of the application is where it should be.

Service Level Objectives

The threshold for availability would be that the application is fully functioning 97% of the time. This means that the website will likely be fully functional whenever a user wants to access it, the 3% it is not online will be used for crucial maintenance that requires downtime. The average request latency threshold for the website should be <400ms. This would mean that all requests to the application are operating at quick speeds. All of the requests should take under 1 second no matter what. Lastly the error rate threshold should be <1%. This would mean that errors only occur in extremely unusual situations.

Service Level Agreement

The user should be able to access the application 97% of the time. If downtime is necessary, users will be given a notice containing information about the outage such as date, time, and duration, within the application at least a week in advance. In occurrences where unplanned downtime occurs, users should expect the application to be functioning again within 24 hours. Request latency should be less than 1s no matter the circumstances, if the website fails to meet this requirement, considerations will be made to upgrade the server the application is being hosted on. The error rate for users should be less than 1% anything higher than this indicates a large problem within the application, and status updates will be given to the user pertaining to upcoming fixes for the error.

User Manual

Prerequisites

This user manual assumes that the user has followed the setup guide above and has navigated to “localhost:3001”.

Account Creation and Login

When navigating to the website, users will first be prompted with the login screen. If the user already has an account created, they can enter either their username or email along with their password. If the login is successful they will be redirected to the homepage. If the user is new to the website, they can click on the “Need an Account? Register Here.” link. The user will then be redirected to the register page, where they can enter their username, email, and password. If the account creation is successful, they will be redirected to the homepage.

Using the Review Page

Once the user is logged in they can use the music review page. To post a review, they can fill out the textbox fields near the top of the page. The user has to enter a score (1-100), the artist name, album name, and the review content. The user should double check that the album title is spelled correctly so that the proper album cover displays with their review. If all these fields are filled out the user can then click the post button and the post will now be displayed to them and anybody that follows them. The user has three ways to interact with other user’s posts on the website. They can either upvote, downvote or reply. The user can reply via the reply button to the left of the upvote button. Clicking on this button will open the reply modal where the user can enter their reply. Once the user is done creating their reply, they can click on the “Save Changes” button which will post the reply.

Using the Search Feature

The home page only displays users that the logged in user is following (along with the logged in user's own posts). To discover other users and reviews the user can navigate to the search page by clicking on the magnifying glass icon. By default the search page is set to search posts, the user can search using the artist name, album name, or review content. The user can then scroll through the results related to their search term. If the user wants to search for other users, they can switch to the users tab located on the same page. This will display users related to the logged in user's search term. They can follow from this page or navigate to the other profiles by clicking on their username to view more reviews from that user.

Creating and Viewing Messages

To view their inbox, the user can click on the inbox button under the magnifying glass icon within the navbar. This will take users to the inbox page, where all group chats or direct messages will be displayed. To create a new message the user can click on the "Create new message" button on the screen. This will prompt the user with a search bar to search for other users to add to the message. Once the desired number of users are added to the message, the user can click on the "Create Chat" button. This will then create the group chat or direct message and add the message room to the inbox page of every user added to the chat. The user can then type in their message at the bottom of the page and send them to other users. Messages from the logged in user will always be on the right side of the screen with a green background. Messages from other users will appear on the left side of the screen with a white background and the username of the user who sent the message.

Viewing Spotify Statistics

In order for the user to view their Spotify statistics they must have a Spotify account. Assuming this is true, the user can click on the Spotify logo in the navbar. This will take the user to the Spotify login page where the user can enter their account information. If this login is successful, the user will be redirected to the Spotify Stats Dashboard which will display the user's top 5 artists, songs, and albums/singles from the past 6 months.

Customizing User Account

To customize their account, the user can click on the profile icon towards the top right of the website. This will redirect the user to the profile page. To change the profile picture or banner image, the user can hover over the item and click on the camera button. This will display the Cropper.JS modal where the user can upload a picture, crop it, and save their changes. Once a change is made it will display to all other users on the website.

Conclusion

I believe I accomplished my goal of creating a functional music social media website. From the beginning the target demographic for this project was to create a music review website with an intuitive UI akin to modern social media websites. The main feature of the website is the review board. This serves as the vehicle for the user to share their favorite music (or least favorite if they feel inclined to do so) to other users on the platform. The goal of this feature is for users to find new music via real human reviews. Users can create a review by entering a score of 1-100, the artist name, the album title, and the review content. By seeing the passion a person may have for an album, it might inspire another user to go listen to that album. In instances like this, I feel like it is paramount for the users to be able to discuss the music on a more personal level. For this reason, I added the reply and direct messaging feature. For example, if a user was inspired by another user to listen to a specific album, the user can then go the review where they first heard about the album and tell the poster their thoughts on the music. If the two users want to have a more intricate conversation about the music, they can create a group chat with themselves and other like minded users to discuss the music. I believe this creates an environment where users will want to share and discuss music with others, as throughout this process they may find people with similar tastes and can swap music recommendations. Not everybody wants to have a conversation however, which is why I added the upvote and downvote buttons, where users can express their opinion of a review or reply via a simple click. Music can be a sensitive subject for some, so upvotes and downvotes are anonymous. If the user wants to search for a specific album or a specific user, they can use the search feature to find what they're looking for. To fully express their personality, users can also change their profile picture and profile banner to something that represents them. This helps add character to each interaction on the website. Lastly, user's may be curious as to what their Spotify listening history looks like from the past six months. For this I implemented a feature that works with the SpotifyAPI to fetch the user's data and display it in a concise readable format. I firmly believe that this project is a technical challenge that displays the knowledge of computer science principles I have gained throughout my undergraduate career. This project encompasses multiple different areas of the computer science field. Every post, message, profile picture/banner change, etc., must interact with the MongoDB database I deployed. In order for many of the features on the website to function I had to create well defined schemas and use the MongoDB JavaScript commands when necessary so that data inserted into the database can be indexed and displayed quickly and properly, or inserted into the database. This portion of the project displays my knowledge of databases. As mentioned previously, I created multiple frontend features such as posting, messaging, searching, and profile customization. Each of these features required extensive JavaScript code that interacts with the DOM using JQuery and defining functions to route users and data. I also had to handle all of the .css for the application in order to make it responsive and aesthetically pleasing. For the SpotifyAPI, I had to learn how to interact with the API and reformat the output so that it displays for the user. For profile customization I imported tools such as Cropper.JS so that users can upload and crop images they

want to display on their account. These areas combined exemplify what I have learned pertaining to frontend development and algorithms. This project also took a large amount of application security into consideration. Features such as hashing passwords using Bcrypt before they are inserted into the database helps drastically with security. I also created a middleware authentication function that checks if a user is logged in before they can access any pages outside of the login/register screen so that users can't gain unauthorized access to other parts of the website. Lastly, this project also shows a decent amount of networking knowledge. I used the .pug view engine as an HTML substitute for this project because data has to be rendered dynamically. In order for a webpage to be loaded, it must receive the proper HTTP response. For example, many of the .pug files require an HTTP 200 (HTTP OK) response in order to render the page, anything else will throw an error. Ajax is used extensively throughout the project as well via the "GET" and "POST" calls when necessary. There are some features that I did not have time to finish or could not finish due to outside factors. The listening party feature was causing API rate limit errors that I could not circumvent, which led me to scrap the feature. Some small features such as private accounts and the live chat were scrapped as well due to time constraints. However, I feel this project is more than sufficient enough to prove that I tackled a large enough challenge for this project.

Project Stack

Frontend:

- HTML/CSS
- JavaScript
- Pug View Engine
- Express.js
- Body-Parser
- Cropper.js
- Axios
- SpotifyAPI
- Bootstrap

Backend:

- MongoDB
- Node.js
- JavaScript
- BCrypt
- Multer

Demo Video

Demo.mp4

References

Cocca, G. (2022, June 23). *JavaScript design patterns – explained with examples*. freeCodeCamp.org. https://www.freecodecamp.org/news/javascript-design-patterns-explained/#factory-method-pattern

Eppy, A. (2022, March 18). *What is information architecture? (UX Tips and examples): Envato tuts+*. Web Design Envato Tuts+. https://webdesign.tutsplus.com/what-is-information-architecture-ux-tips-and-examples--cms-40064t

flowmapp. (2024). *What are UX frameworks and how to use them?*. FlowMapp design blog. https://www.flowmapp.com/blog/qa/ux-frameworks

GeeksforGeeks. (2024a, February 21). *Chain of responsibility design pattern*. https://www.geeksforgeeks.org/chain-responsibility-design-pattern/

GeeksforGeeks. (2024b, March 12). *Event-driven architecture - system design*. https://www.geeksforgeeks.org/event-driven-architecture-system-design/

Hardy, I. (n.d.). *Node.js and Python Integration: Powering versatile and Scalable Applications*. Node.js and Python Integration: Powering Versatile and Scalable Applications - Chef Supermarket. https://supermarket.chef.io/tools/node-js-and-python-integration-powering-versatile-and-scalable-applications#:~:text=The%20combination%20of%20Node.,key%20advantages%20of%20using%20Node.

Parmar, A. (2023, September 26). *Know everything about event-driven architecture in Node.js*. Prismetric. https://www.prismetric.com/event-driven-architecture-in-node-js/#:~:text=JS's%20event%2Ddriven%20architecture%20excels,large%20number%20of%20concurrent%20events.

Terra, J. (2023, August 7). *What is client-server architecture? everything you should know: Simplilearn*. Simplilearn.com. https://www.simplilearn.com/what-is-client-server-architecture-article#:~:text=The%20client%2Dserver%20architecture%20refers,model%20or%20client%20server%20network.

UXPin. (2024, March 8). *UX design frameworks – what are the most useful ones?*. Studio by UXPin. https://www.uxpin.com/studio/blog/design-frameworks/

(2023, June 24). *JavaScript design patterns: Adapter*. Joe Zim's JavaScript Corner. https://www.joezimjs.com/javascript/javascript-design-patterns-adapter/

Google. (n.d.). *Sre Fundamentals: Slas vs slos vs slis | google cloud blog*. Google. https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-slis-slas-and-slos

Jones, C., Wilkes, J., Murphy, N., & Smith, C. (n.d.). *Service level objectives*. Google. https://sre.google/sre-book/service-level-objectives/

Google SRE. (n.d.). *Example slo document*. Google SRE - SLO Documents: Game Services API, HTTP, Score. https://sre.google/workbook/slo-document/