MUSILINKY

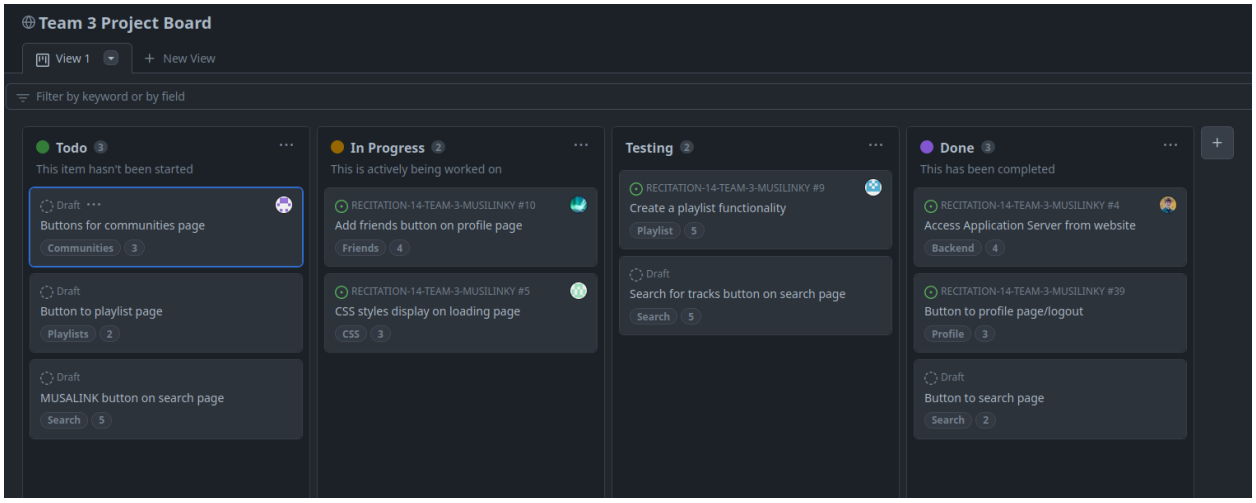
Silas Khan, Tyler Chung, Eric Gosnell,
Alex Burch, Cade Williams, Patrick Flemming

Project Description:

For our project, we decided to create a music service where users can search for songs, be provided a link to that song through Deezer, and then use that link to generate links to other music services such as Spotify and Apple Music. Users can then add and delete songs into a playlist and access their unique playlist through their profile page. The playlist feature as well as the search feature filters songs and then displays them according to song id, song name, song artist, song album, a 30-second song preview, and finally the Deezer song link. This website comprises our unique functionality of music links inputted and music links outputted as the idea is that users can share links to songs with their friends for a communal effect to music. Our songs are stored on an external database which we've called through the Deezer API. Further API calls such as DuckDuckGo, Spotify, MusicAPI, and Soundcloud helped us to complete our website. Other than that, simple everyday features such as login, logout, register, and profile pages were implemented to create a more well-rounded user experience.

Project Tracker - GitHub project board:

https://github.com/users/EricGosnell/projects/1/views/1



Video Demonstration:

https://clipchamp.com/watch/cnqZbCACNtT

Version Control System:

https://github.com/EricGosnell/RECITATION-14-TEAM-3-MUSILINKY

Contributions:

Eric Gosnell:

I contributed primarily to the backend of Musilinky. In the first week, I created the repository and project board, and included the framework for file structure, along with writing the code to run the docker container and website. I then created the API routes for login, register, and logout, along with the database tables to store the data. With this, I created a simple UI to be able to log in and navigate through the website. Finally, I created new database tables and API routes to allow users to add friends, however, this had to be abandoned due to time constraints.

Tyler Chung:

I contributed extensively to the front-end development. I started creating mock pages based on Eric's initial design. As the project progressed, I focused on different aspects of the

website, such as creating the homepage, link search page, playlist page, profile page, log in, and registration page. Each page was developed using three different programming languages - HTML, CSS, and JavaScript - to ensure the website's user interface was running smoothly and effectively. Once each page was created, I presented it to the team for feedback and incorporated any necessary changes. Most of the mock pages were created using pure HTML, so in the final weeks, I integrated the HTML into the EJS pages. I ensured that the front end and back end were correctly incorporated to create a functional and visually appealing website. In the last week of the project, I collaborated with the team to make final adjustments and ensure the testing aspect was working properly.

Alex Burch:

For contributions, I largely worked with Silas on the playlist and API incorporation plus overall contributions with meetings, the project board, and the use case diagram. Added in the skeleton API code for the Music API which helped to provide the external links we used for our website. Provided the skeleton code for the playlist and helped to integrate with further commits and CSS implementation. Helped the team with CSS implementation, debugs with the playlist, login, and register pages, and helped to lead discussions and meetings. Helped to update the project board along with Silas and revise epics and user stories along the way. Created the video demo for the project as well and worked with Silas on the final implementation of the use-case diagram.

Silas Khan:

At the start of the project, I added a search endpoint on the backend that allows users to search for songs via different filters. This endpoint uses the Deezer external API, and I made an accompanying front-end table for its results as well. Following this, I made a Musalink endpoint

in the backend that allows users to retrieve song links on Apple Music, Spotify, Soundcloud, Deezer, and Youtube Music. This endpoint uses external APIs that are outlined in the below use case diagram. I made an accompanying front-end table for its results as well. At the end of the project, I implemented a functionality that allows users to add songs from the Deezer API to a playlist.

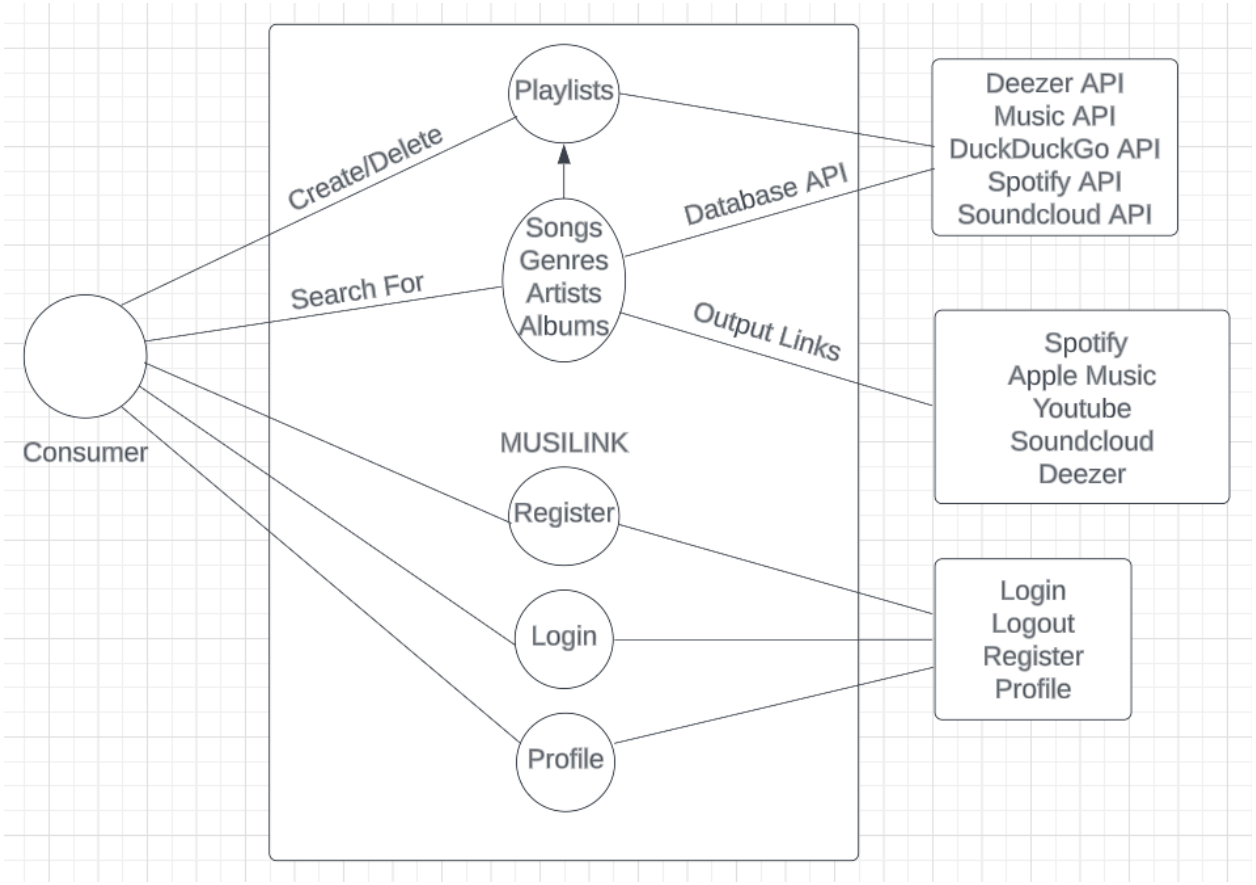
Patrick:

My main contribution to this project was the communities page, for which I programmed the SQL components, the endpoints in the index.js file, and all of the front end components of the page. To get this page up and running, I programmed 4 endpoints in the index.js file and two tables in the create.sql file, where information relating to the communities and the users that could join them would be stored in the database. For the front end, I had to undertake some problem solving to properly display the data from the database, using nested loops in a table. I also got the project deployment up and running using Microsoft Azure according to what was outlined in Lab 13.

Cade:

My main contribution was to the front-end of the application, focusing on the presentation of specific pages and site elements such as the footer. I experimented with many different designs and approaches within the HTML code in order to ensure my contributions flowed well with the project's look as a whole. I collaborated extensively with Tyler on the design of the profile page, where we both had different designs and had to decide which style would ultimately compliment the project best. I also managed outside elements such as the project board.

Use Case Diagram:



Test Results:

- **Use Case 1:** Users can access a database of songs
    - What are the users doing?
        - **The users are clicking on song links on other platforms after retrieving them by using the Musalink API.**
    - What is the user's reasoning for their actions?
        - **Most users do this because they want to listen to songs on platforms that they do have. Other users are doing this because they have accounts on more than one platform and thus want to broaden their listening experience.**
    - Is their behavior consistent with the use case?
        - **This behavior is consistent with the use case. The intended purpose of this feature is to grant users the freedom of listening to songs on various platforms. Since we did not want to restrict the users to only**

one platform, they are using the retrieved links to expand their music experience on other platforms, which is exactly what we intended.

- If there is a deviation from the expected actions, what is the reason for that?
  - **In some cases, the users would receive incorrect or non-existent song links based on their query. The reasons for this are as follows. When we first began developing the Musalink API, we found that song data would consistently not be returned if the user entered an Amazon Music link. We also found that incorrect song data would often be returned if the user entered a Soundcloud link.**
- Did you use that to make changes to your application? If so, what changes did you make?
  - **Yes, we used this to make changes to our application. In this case, we decided to drop the support of Amazon Music in our app. Although we did our best to get Amazon Music links working, we found that the MusicAPI simply did not work when these were used. Another change we made was altering the external API that parses Soundcloud links. Originally we used the MusicAPI for this purpose, but upon further testing, we found a Soundcloud external API that reliably returns song data when official Soundcloud links are entered.**

- **Use Case 2:** Users can add songs to a playlist
  - What are the users doing?
    - **The users are clicking the "ADD" button to add songs to their playlist after first searching for them using the Deezer external API.**
  - What is the user's reasoning for their actions?
    - **They do this because they want to save a select number of songs for future use. That is, they want to be able to listen to their favorite songs whenever they want without having to search for them again.**
  - Is their behavior consistent with the use case?
    - **This behavior is consistent with the use case. The intended purpose of this feature is to allow users the opportunity to reserve certain songs for their future listening pleasure. Since the users are adding songs to their playlist for this exact purpose, their behavior is consistent with the design.**
  - If there is a deviation from the expected actions, what is the reason for that?
    - **There was no deviation from the expected action during our testing. This was likely due to there being no exploitable bug with regard to this feature. However, some users requested that the playlist user interface be altered so that they can more easily see their songs.**
  - Did you use that to make changes to your application? If so, what changes did you make?

        - **Yes, we used this to make changes to our application. In this case, we altered the CSS for the playlist page so that there is a better blend between the background and display table. We also made some slight alterations to the playlist table so that the song data is more readable and easier on the eyes.**
- **Use Case 3:** Users can search for songs via title, album, genre, or artist
    - What are the users doing?
        - **The users are inputting song titles, albums, genres, or artists into the search bar. They then click the Search button to see the returned results in a table.**
    - What is the user's reasoning for their actions?
        - **They do this because they want to listen to the returned songs on their respective Deezer links. The users are also searching for songs via various filters so that they can add the songs to their playlist.**
    - Is their behavior consistent with the use case?
        - **This behavior is consistent with the use case. The intended purpose of this feature was to grant users the flexibility of searching for songs with different filters. We understand that some users are more familiar with music than others. Thus, adding the ability to search for songs via different filters allows users to broaden or specify their search queries depending on their comfort level.**
    - If there is a deviation from the expected actions, what is the reason for that?
        - **Occasionally, a user would submit the search form without adding any text to the search box. This would cause the external API call to Deezer to fail. There were various reasons for this deviation. Some users were curious about what would happen, and others would make the genuine mistake of submitting the form without entering any content.**
    - Did you use that to make changes to your application? If so, what changes did you make?
        - **Yes, we used this to make changes to our application. In this case, the necessary change was adding the required attribute to the search box. This ensured that a blank form could not be submitted. If the user attempted to submit a blank form, then a message would be displayed.**
- **Use Case 4:** Users can access the provided links when using the Musalink API
    - What are the users doing?
        - **The users are inputting song links from Spotify, Apple Music, Soundcloud, Youtube Music, and Deezer into the Musalink search**

bar. They then click the Musalink button to get the links to these songs on other sites.

- What is the user's reasoning for their actions?
  - **They are doing this because they want to listen to songs on platforms that they do have. From our testing, this means that they input a song link on a platform that they do not have so that they can listen to it elsewhere.**
- Is their behavior consistent with the use case?
  - **This behavior is consistent with the use case. The intended purpose of this feature was to allow users to connect with friends who send them song links on platforms that they do not have. Since the user is then using these links to generate other links on platforms that they do have, their behavior is consistent with how we wanted the app to be used.**
- If there is a deviation from the expected actions, what is the reason for that?
  - **Some users would occasionally input incorrect links, which would cause the external API call to fail. The reason for this is that the application did not specify the expected format of the inputted links.**
- Did you use that to make changes to your application? If so, what changes did you make?
  - **Yes, we used this to make changes to the application. In our case, we added a regular expression on the form that checks if the inputted URL is valid. If it is not valid, then a message is displayed to the user.**

Deployment (with screenshot):

http://recitation-14-team-03.eastus.cloudapp.azure.com:3000/login