SPOT U, Social Media that Excites

Zev Kunianski - chinchillajo
Bobby Burns - bobby-burns
Mason Choi - mason-ch01
Aaron Nong - Aaronnong02
Tawhid Ather - Redsuperninja
Nikky Merkel - NikkyM3

Project Description:

Spot U is a music social media application that allows users to connect based on their music preferences. Users are able to interact with each other by following, and unfollowing different accounts. These accounts will populate the users explore page, so they can see what other users have posted. In addition, users are able to comment on others posts to give their feedback on their song choice. There is also a search feature, where someone can look up a specific song on our website. From the search feature or from someone's posts, there is a link that allows the user to be able to open spotify through the click of a link. In order to customize their profile picture, they are able to select from 5 pre-determined photos which come from the xbox 360 accounts in order to pay tribute to where a lot of our social connections started online. This application will hopefully allow users to find new friends and expand their music taste my explore genres and artists they have never heard before.

VCS:
https://github.com/Mason-ch01/SpotU-012-group-1

Github Project Board:
https://github.com/users/Mason-ch01/projects/1/views/1?filterQuery=

Video:
https://drive.google.com/file/d/1PMx4_1J2D-GINScHLw1IUliSoyzdCC5S/view?usp=sharing

Contributions:

Zev Kunianski:

I was focused on the profile page and did full stack development to create the layout and the features for it. I built the layout for it, and at first I worked on formatting the posts, but it was taking too much time so I switched to adding the sql calls in order to get the data from the database. This allowed me to get the information for the followers, following, and posts for a specific user. I also added the change profile feature which also used queries.

Nikky Merkel:

I worked on the save_song function in index.js, which was used to get details about a song like its ID, name, image URL, artist, and link. Later, this function was replaced with the searchSong function. I also made changes to search.hbs and new_posts.hbs to improve how things looked and worked. I added a button to the song cards in the search feature, making it easy to add songs to posts. This saved the song's image, name, post title, and caption in the database. These posts then showed up on the explore page, making everything work smoothly.

Bobby Burns:

I was responsible for all things spotify. This includes the entire OAuth2 authentication flow as well as storing and getting the authentication token. I also set up functions in our codebase that made communication between the spotify API endpoints a smooth process. With those, I set up our spotify developer portal with all the callbacks and permission scopes, as well as the user accounts for all the members of the project in order to get authentication working for all of us. I then helped our team through merge conflicts, and bug solving, and then finally piecing together the final product.

Aaron Nong:

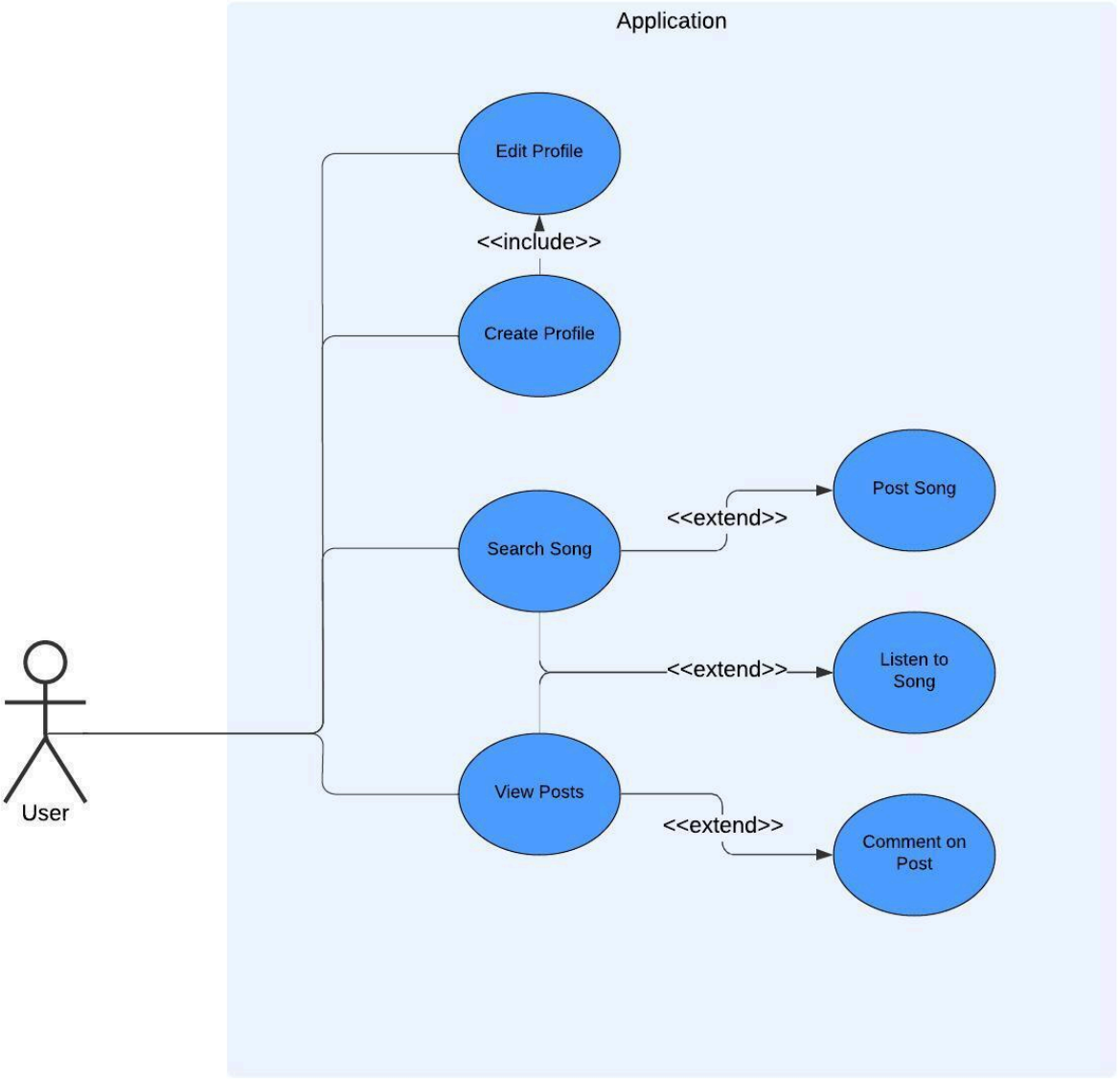
I was mostly working on the login and explore page. For the login page, I created the route to query the database to check that the inputted username and password had a matching user in the database, while also including hashing using bcrypt. I also created the initial design of the login page. For the explore page, I did some minor work on the HTML/CSS formatting and design, and then I created the comment dropdown and route to allow users to write and read comments on a post.

Mason Choi:

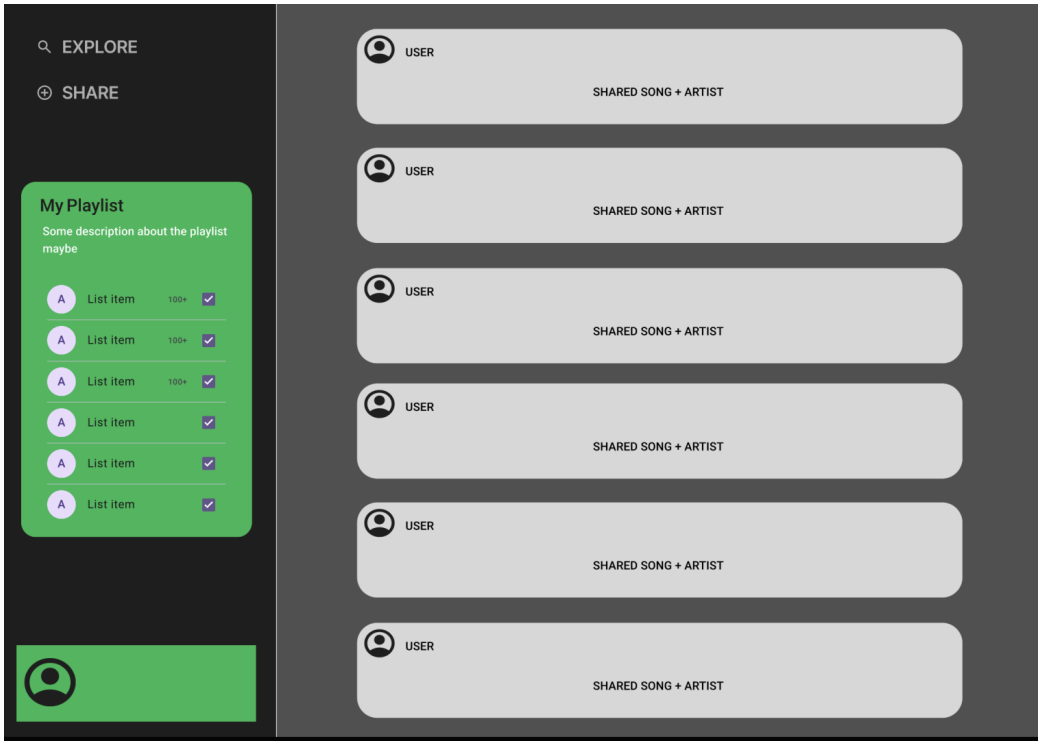
I designed and implemented the database architecture using lucidChart and PostgreSQL in order to ensure efficient data storage and retrieval for the functionality needed in our app. Additionally, I developed a lot of the front and back end for the search page which allowed users to search and find songs in Spotify’s library. This functionality was built using HTML, handlebars, and javascript. I also contributed to the discover page by creating some of the front end cards and creating the queries to display the data to said cards with the same tools used for the search page.

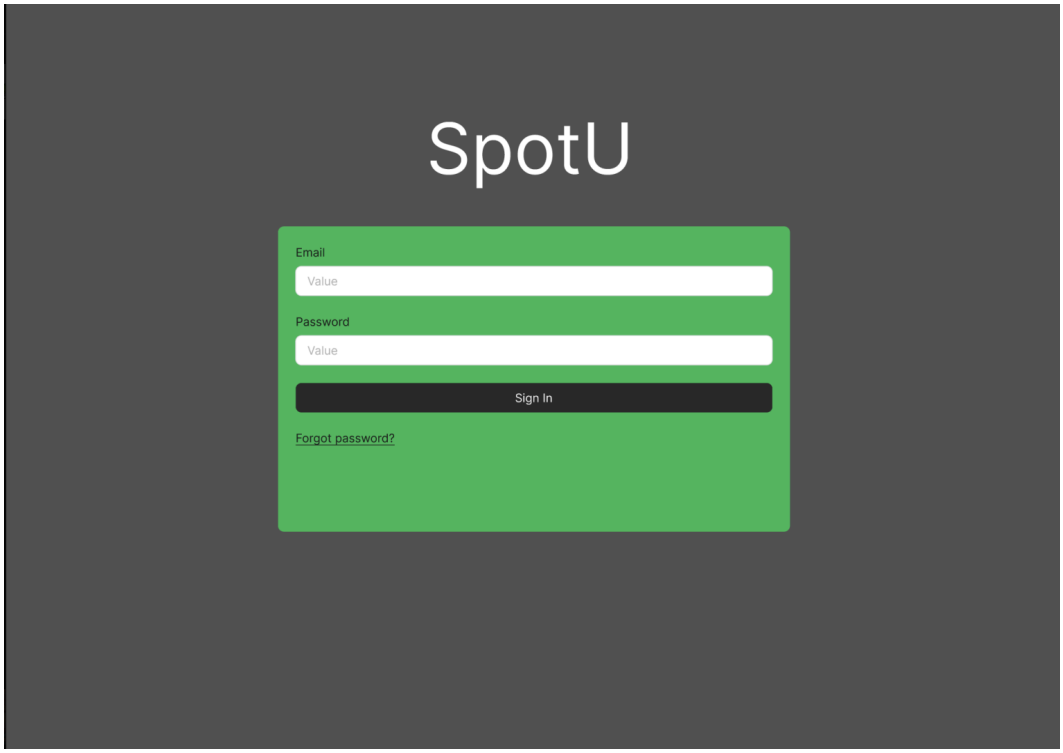
Tawhid Ather:
I focused on enhancing the navigation bar and the integration of multiple pages, ensuring consistent formatting and functionality between routes and calls. I also designed the styles for the registration and login pages and updated calls to work with the new database which was expanded as the scope of our feature changed. Additionally, I implemented unit tests to secure features like comments and posts, preventing unauthorized access when users were not logged in.

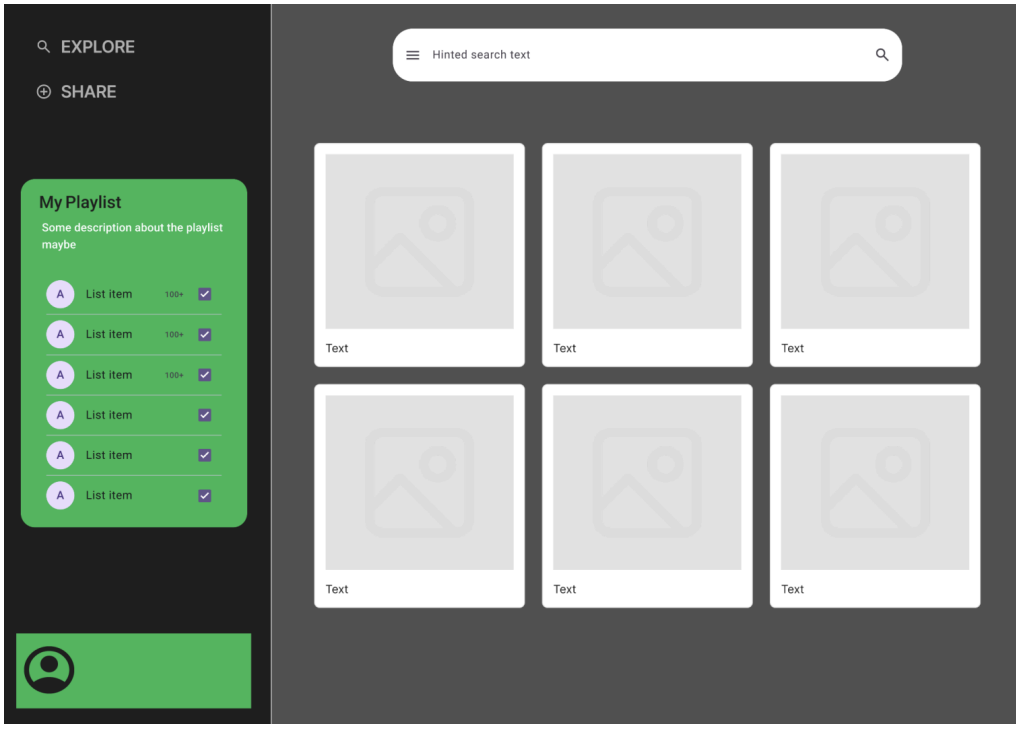
Use Case Diagram:



WireFrames:


EXPLORE
SHARE
My Playlist
Some description about the playlist maybe
A List item 100+
A List item 100+
A List item 100+
A List item
A List item
A List item
USER
SHARED SONG + ARTIST
USER
SHARED SONG + ARTIST
USER
SHARED SONG + ARTIST
USER
SHARED SONG + ARTIST
USER
SHARED SONG + ARTIST
USER
SHARED SONG + ARTIST


SpotU
Email
Value
Password
Value
Sign In
Forgot password?



Test Results:

Test Cases

Below are the main test cases we will be testing based on the provided API routes:

1. **Default Welcome Test Case**
    - **Use Case**: Test the /welcome endpoint to check if the server responds with a default welcome message.
    - **Expected Behavior**: The server should return a status of 200 and a message of "Welcome!".
2. **Add User API (POST /add_user)**
    - **Use Case**: Test the /add_user endpoint by sending a user object with id, name, and dob.
    - **Expected Behavior**: The API should return a status of 200 and a message of "Success".
3. **Login API (GET /login and POST /login)**
    - **Use Case**: Test the login functionality using both GET and POST methods.
        - For GET: Check if the login page is rendered.
        - For POST: Test a valid user login and verify redirection.
    - **Expected Behavior**: GET should return a login form, and POST should authenticate the user and redirect them.

4. **Register User API (GET /register and POST /register)**
    - **Use Case**: Test the registration functionality using both GET and POST methods.
        - For GET: Check if the registration page is rendered.
        - For POST: Test creating a new user by sending valid registration data.
    - **Expected Behavior**: GET should render the registration page, and POST should return a successful registration and redirect.
5. **Spotify Connect API (GET /spotify_connect)**
    - **Use Case**: Test the /spotify_connect endpoint to check for Spotify authorization redirection.
    - **Expected Behavior**: The API should redirect to the Spotify OAuth authorization page.
6. **Create New Post API (POST /new_post)**
    - **Use Case**: Test the creation of a new post by sending information about a song.
    - **Expected Behavior**: The API should successfully create the post and redirect to the /explore page.
7. **New Comment API (POST /new_comment)**
    - **Use Case**: Test posting a new comment on a post.
    - **Expected Behavior**: The API should return an error if the user is not logged in (status 500).
8. **Logout API (GET /logout)**
    - **Use Case**: Test the logout functionality.
    - **Expected Behavior**: The user should be logged out and redirected to the login page.

The use cases that were tested were the user registering and logging in, creating a post, commenting on a new and existing post, and going to and modifying the profile page. For the registration and login, one of the users attempted to log in before registering as they didn't read that they were on the login page but they were prevented from logging in as they had not signed up which is consistent. They also tried to use the nav bar before logging in which was not expected but prompted us to ensure that any use of it would prevent you from accessing the pages if you were not logged in. The post and comment use cases were consistent where the users tried to add the post as intended and commented on them as well. Some of them just wanted to look at comments or listen to the songs of pre-existing posts which was an expected use case that we had accounted for. As for the profile page, many of them were not expecting their posts to be directly underneath on the same page but didn't see any issue with it but one did ask for post removal which we would think about adding to our application in the future. Other than that the updating of the photo went smoothly and worked well.

Deployment:
https://spotu-012-group-1.onrender.com