SingleStream

Team 8 – John Fantell, Robert Rotering, Ryan Flynn,
Roshni Vachhani, and Will Stone

Spring 2017

Inspiration

Each member of our team has a passion for music, and naturally, we all have multiple platforms we use to listen to our favorite songs. Although Spotify and Apple Music have a lot of the big hits, they sometimes don't have some of the less popular songs that we also enjoy listening to. This leads to the problem of having different playlists scattered across multiple platforms, making it difficult to listen to all of your favorite music at once. This sparked the idea for SingleStream, an application that aims to centralize libraries from different music listening platforms. We also felt the need to give users the ability to interact with other users, leading to the "social media" aspect of the application. We additionally wanted to add an analytics capability to differentiate our applications from the other ones out there.

Our Solution

To address the problem today's music streaming industry has, our Team decided to create a platform to unify popular streaming sites. Developing a singular site to stream your favorite music had to have a few key features that create a unique but familiar site. First, the main feature, the site had to have playlists that supported all streaming services. Second, the site needed search features that incorporate all streaming services. Third, the site needed a clean UI to help users understand the site functionality. And finally, a social aspect that helped users find new music through their friends. Combining all of these features together creates a platform that includes the best parts of popular streaming sites.

Front End

We decided to utilize a simple, sleek design when building the interface for SingleStream, The login page introduces the color palette for the rest of the application, and makes use of a bootstrap-driven minimalistic design, providing information without being visually overwhelming. A consistent top menu/sidebar is used throughout the rest of the application in order to create an intuitive user interface. The sidebar uses font-awesome icons to represent the other pages, and the names for those pages are accessible upon opening the sidebar or hovering over the icons.

Additionally, we changed the design of the playlists page since the final presentations in order to adopt a more modern music streaming design, similar to that of Spotify. We also moved the playlists to be displayed with the links on the sidebar menu, and each playlist can now be represented by any font-awesome icon the creator chooses (a link to the total list of font-awesome icons is provided during the playlist creation process). The look and feel of any website is important, and the layout and design pallete were chosen carefully for this project. Modern sites tend to use either very dark (Spotify) or very white (iTunes) designs. We went with darker colors when building the website, and were careful to implement them in an approachable manner, and not leave the application looking cluttered, confusing, or outdated.

Back End

For the back end, we utilized the MEAN stack technologies to build our site. This included Node.js and Express to create the server and handle all routes and HTTP requests from the client. These requests also include the many different calls to the API endpoints for streaming services. For our database setup, we utilized MongoDB with Mongoose for interaction with Node.js. We wrote a Playlist schema which contained an array of track objects and a field for a user_id as well as some additional data. We also wrote a User Schema to hold local login data as well as all the authentication information for each of our streaming services. This schema also had Following and Followers arrays to support our social features on the site.

For managing the communication between our server code and the front end we utilized Angular.js to initialize and send all the HTTP requests depending on the user's behaviour on the webpage. This was mainly essential in handling events like when a user searched for new songs, created a new playlist, followed a user, and requested data analytics. Angular was also helpful for managing the different elements on the page and when they would be shown or hidden from the user. We also used EJS format for templating and rendering pages from the server instead of regular html files.

Challenges

Throughout the duration of this project our team ran into many challenges that forced us to work flexibly and keep an open mind about our expectations for the final product. Among the first roadblocks was the issue of settling on the different music streaming API's that we were going to utilize for our application. Our original goal was to use API's for Spotify, SoundCloud, and YouTube as we believed those three would produce a very diverse library of music and satisfy a wide range of user preferences. However, as we began work on incorporating these, we were faced with the fact that Spotify is very restrictive about streaming full tracks in it's web API and we would not be able to incorporate it in the way we wanted. In addition, we discovered SoundCloud had also recently restricted its API usage and after applying for access, we were denied after a whole month of waiting for a response. This situation forced us to continue searching for an additional streaming API that we could pair with YouTube, and this is when we discovered Napster.

Napster seemed to work similarly to Spotify's services and we set out on incorporating the API only to discover a new challenge. Napster requires Adobe Flash to be enabled in the web browser in order to stream music. This was a variable out of our control, but seriously impeded our ability to have it work smoothly with YouTube and caused issues with cross-browser compatibility as well as mobile support. Nonetheless, we were able to write functions to play songs from both services on the same page and were able to construct a very basic auto-play feature that paces through the tracks regardless of source.

Overall our challenges were mainly in working with the API's from Google and Napster, and the main effect is how well we are able to achieve a functional cohesion between the two. YouTube is meant for video streaming and thus does not offer isolated audio for its web API and can produce much longer buffering time compared to Napster. However, our team feels that we were able to work adaptively against these challenges and produce an application that, while slightly handicapped, does functionally demonstrate the main idea behind SingleStream.

Future

After completing the project for the semester our group is very excited about what we completed. We invested a lot of time and effort into the project and see potential for a great product in the future. The team will likely continue development on the project to improve existing features and add more.

We would like to add more music services to our platform to make it a more robust tool. We would also like to change Napster's music player from Flash to an HTML player when they release their next API update. Finally, we'd like to improve the playlist functionality to have a few more features such as song skip, replay, pause, and shuffle library. There are a lot of improvements we can make to the project but now we're no longer bound in a time frame. We'll work on the project in our own time and add features that we feel would improve the site.

Closing Thoughts

Looking back on the project, we're very proud of what SingleStream has become. We set out to complete a very ambitious project and have met almost all the features we planned for. Aside from API roadblocks, our project completed with the main idea still intact. We created a working application that combined different music streaming apps into one area.

More importantly, we solved a real world problem that many people have. That's an exciting feeling. SingleStream right now is just a small application made by five students, but it could grow to be so much more. We've created a prototype that could be adapted to fit more API's and include more features. The idea we turned into a product could be a game-changer in the music industry, especially as more platforms differentiate from each other.

Development of the project was excellent. Our group divided the work according to knowledge sets but still worked as a group. Pair programming was used often to help mitigate bugs, merge conflicts, as well as keep everyone on the same page. We also had a great group dynamic. Working on the project was rarely stressful because we did most of the work together and genuinely enjoyed working together.