Merge Sort Music Final Report

Team Name: Merge Sort Music
Team Members: Shreyank Patel, Georgia Stricklen, Jonathan Ting, John Carmack

1. Introduction

1.1. Project Description

While many media hosting services do allow users to manage and organize their personal content into playlists, the functionality of such services can be lacking in both efficiency and efficacy. When users of these services wish to partake in the media offered, rarely do they wish to spend time battling the cumbersome user interface for building playlists that the services offer. In addition, many of these services lack the ability to sort the playlist of media on criteria that many users find significant (such as user rating).

We propose a web application that will allow users to import libraries of media from these media services and then organize them into playlists based on user-generated criteria. These playlists will then be able to be exported to the media services that they came from. Users will also be able to store these playlists with their account such that it will serve as a backup external to the media hosting platform that it's for. As far as the team is aware there are already a number of similar applications that provide this service, but they offer it for specific, obscure media hosting services. Our proposed application would offer similar services for Spotify, which is a far more mainstream service.

In order to implement and improve some of the designs within our project, changes to some of the fundamental algorithms on both the front and backend were necessary. One of the major changes we encountered while we were designing the website were changes to the merge sort algorithm. We decided that we wanted the user to be able to stop and continue sorting the playlist at a later date. This means that the state of the playlist needs to be saved. A recursive merge sort algorithm will not accomplish this easily. To remedy this, we implemented an interactive selection sort algorithm that does not rely on querying the stack to save the state of the playlist. The interactive merge sort method, much like the recursive method, split up the playlist into multiple smaller arrays and asked the user to decide between two songs. The size of these sub lists grew until the entire playlist was compared. This change also allowed us to easily calculate and save the amount of comparisons needed to be made. The website can display this information to the user. Another change we made with regards to the merge sort algorithm was its placement within our project. We originally decided it should be in the back end. This way the playlist could be stored in a database. When trying to implement the algorithm, we discovered

that getting querying the user for input would require the backend to be interrupted. This was decided to be too complex for our project. It was then decided that the merge sort algorithm was best suited to be placed in the frontend where input from the user could easily be captured.

This decision was made late in the development process, and due to John and Shreyank's inexperience with React the merge sort had to be substituted for an interactive selection sort. This sorting was implemented and made it to the live version of the project.

Our goal was to deliver a web application that allowed users to sort their Spotify playlists with a merge-sorting algorithm. While we were not able to fully implement a merge sort, a different sorting algorithm was substituted and users are able to import a Spotify playlist, sort it by preference, and then export it back to Spotify.

1.2. Team Description

Jonathan Ting is a senior studying Computer Engineering currently doing undergraduate research in hardware security. However, as a hobby, he also enjoys dabbling in web development, both with backend frameworks such as Django and frontend frameworks such as Bootstrap. This project will fulfill a wish of his, to optimize the process of sorting music playlists, of which there are not many great tools for at the moment.

Shreyank Patel is a senior majoring in Computer Science and Economics. He enjoys running, and cooking. His areas of interest include Machine Learning and Cryptography. He has prior experience designing websites using HTML, CSS, Javascript and React. Through this project he hopes to gain meaningful experience in Bootstrap, Django and other frameworks.

Georgia Stricklen is a Junior at the University of Tennessee, majoring in Computer Science. Her main focus is C++ and Java with some experience in Python. She enjoys reading and video games. Solving real world problems while bettering existing structures is what she hopes to work towards in her career as a Computer Scientist.

John Carmack is a Senior majoring in computer science. Web development is a primary interest for him, though he has only ever worked on fairly static web sites with limited user interactivity. He enjoys rock climbing, brazillian jiu-jitsu and works as a commissioned artist in his spare time.

2. Customer Value

We realized by mid April that a merge sort would be too difficult to implement by the deadline.

The decision to pivot to a selection sort was made around the 20th of April and it was fully implemented by the next day. It was the team's feeling that a selection sort would not negatively impact customer value, and it proved to be substantially easier to implement and allowed the team to do so by the deadline.

3. Technology & Tools

Our tech stack did not change from the technology listed in the final project proposal.

Our backend is a combination of Django, Django REST Framework, and SQLite, and our frontend is primarily React and React Router with Material-UI components.

All components function as expected with minor visual bugs.

Each element of the frontend (main page, user page, react components) was tested incrementally with testing accounts.

The backend testing involved primarily testing the functionality of the API calls, which was done through Django REST Framework's built-in testing tools, such as being able to make API calls without a frontend. Once the API calls were verified given multiple test cases, they were also verified with a proper frontend which made RESTful API calls.

When the merging of the frontend and backend had taken place both John and Shreyank bug tested the application and met to discuss bug fixes and implement them.

4. Team

These assignments are relatively flexible and subject to the requirements of the project. During the duration of development, our roles remained static. Jonathan Ting developed the majority of the backend for our project. Georgia Stricklen developed the iterative merge sort algorithm implemented at first in the backend and then moved to the front end. Shreyank Patel and John Carmack developed the frontend for our projects.

4.1. Georgia Stricklen

Merge Sort Algorithm and back end.

4.2. Jonathan Ting

Django Database Design and Backend API. Project manager.

4.3. John Carmack

Front end design and development. Documentation. User advocate

4.4. Sheyank Patel

Front end design and development.

5. Project Management

With regards to our goals, we did not have enough time to implement collaborative sorting for our project, or fully implement the merge sorting algorithm for sorting.

The primary reason for these goals not being met was the team's general inexperience with the technology that the application was built on, and a underestimation of the complexity of the implementation required to fulfill the goals completely.

6. Reflection

In general our team was very respectful of each other's ideas. We were able to adapt to each other's experience levels and step up to fill gaps in understanding or experience.

We were very good at using meeting time effectively, which was important because as the semester wore on it became difficult to adhere to a consistent meeting schedule. Being able to capitalize effectively on what meeting time we did have was instrumental in the project's success.

We used meeting times to discuss the results of testing done over the preceding week as well as to determine what our next steps needed to be for each iteration. This might have been one of the weaker points of our project, though.

We tended to run headlong into implementation without fully considering the overall goal of our project, and that ultimately resulted in the substitution of a merge sorting algorithm for an insertion sort.

Even so, we consider our project to be a success. Each of us were able to explore technology that we're interested in, and at the end of it all we were able to deliver a web application that allows users to sort their Spotify playlists through a more helpful interface.