Assignment 2 Report

Ashwin Kamalakannan - 100584423

Details of application

For my assignment 2, I created a distributed mp3 player application using Java RMI. My goal was to have a pure GUI based application that is intuitive and straight forward for a non tech savvy person to use. I wanted multiple users to be able to connect and use the server at the same time. I also wanted the users to be able to update the available tracks on the server. This essentially means that my application is a mix between Spotify, where users can download songs from a server and have them on their local devices to listen to, and Google Cloud where users can upload their own files and sync them accross their devices. I felt that the combination of these two features made for a truly novel application.

Process

I started building the project using the existing code that I had made for Lab 2. This included a client/server applcation communicating through an rmi registry. I then focused on looking for a decent mp3 library that suited the scope of this application. I wasn't looking for anything overly complex, since this project focused on building an RMI application and given the time constraints. I settled on using the jaco mp3 library, which is a simple pure java based mp3 implementation capable of decoding and playing mp3 files. I spent the first couple days building the basic player functionality including `play`, `pause`, `forward`, `backward` and saving playlists. Next, I started work on the gui for the application. I went with a simple JFrame with buttons for each function, labels for the text and a table to display the playlist and currently playing track. I also wanted to include an upload feature which would allow you to search your computer for a track you want to upload. For this, I used a JFileChooser which uses the native file explorer to search for files. To add the functionality to all components of the JFrame, I used action listeners to check for button presses, etc.

For the 5 RMI methods in my file interface class, I decided to use methods that would help move the music files to and from the server storage. Rather than stream the song to the user, which would be slow and unecessary since the files are fairly small, I simply keep the tracks synchronized on both the server and client. Whenever a new track is added to the server, I use the `download(string)` method to send that file to the client. If the client wants to upload a new track, they will use `upload(byte[], string)` method which reads a file on the clients file system and sends the data over to the server to be stored. If the client wants to remove a track from the playlist, they press the delete button which calls the `delete(string)` method. When the client first starts, I call `checkAvailableSongs()` to help synchronize the server's songs with the clients. If there are any new or removed tracks, these updates are then reflected on the client side. Finally, I wanted the client to display the album art for their particular song. To accomplish this, I call the `downloadImage(int, string)` method, which uses an image api to search for a random image based on size and keyword. This api is extremely basic and it does not allow me to look up real album art, instead I can get a picture of the artist by taking a keyword from the track name. One of the benefits of using Java RMI, is that I did not have to implement any kind of multi threading or resource locking since Java RMI already uses this in its implementation. Because of this, my workload was significantly reduced and I was able to focus on building the rest of my application.

Novel features

My first novel feature is simply that my application exists as a stand alone gui app. If I build it down to a .jar file, I could easily distribute it and the client wouldn't have to have any knowledge of Java to compile and run it. This greatly improves the usability of my application for all users and will result in less unexpected behavior and improved security. This is because there is less room for error when interacting with buttons and lists than text input forms.

The second novel feature is that my distributed music player essentially combines apps like Spotify and Google Drive into one. Meaning that a user can upload their own music to a central server and synchronize it across their devices, while still listening to the music with the same app. I have yet to see a similar app that allows you to do this and I believe this is a stand out feature of my project.

Challenges & Solutions

The main challenge I faced when building this app was the combination of having to learn so many new concepts while juggling my time with midterms and other assignments. For this project, I used Java RMI, GUIs, a new mp3 library and several other concepts that I am not familiar with. For example, I don't have a lot of experience working with the `byte[]` data type and the RMI interface required that I pass data through it in this format. This meant that I would have to perform several conversions as I read files, converted them to byte arrays and them converted them back to mp3 files with the correct encoding. The mp3 library I used helped me with this. Also, I found a lot of documentation on using JFrames which proved helpful when adding complex components such as the JFileChooser.

Conclusion

This assignment 2 was very interesting and posed a significant challenge for me to complete. I think by increasing the scope and complexity of my project idea for this assignment over assignment 1 was helpful for me, since I now have all of the knowledge and resources required to complete my final project which is similar to this but involves video streaming. If I could go back and update my code, I would add the ability to auto synchronize the playlist data instead of having to sync when it is opened. This would have further increased my usability, however I didn't think of this feature at the time even though it is essentially already implemented and I would just have to call the method periodically.