The artifact I worked on this week is a jukebox application. The jukebox used to function based on a large number of Java classes to draw student and playlist info. On top of this, the only functionality of the application was to print out the songs in a given playlist. This was created in the summer of 2023 and was made collaboratively with other students via BitBucket. This collaborative nature explains the fairly unintuitive way the application was set up. Each student could easily add a new class that showed proficiency in using Git. I chose to migrate this over to a MySQL database for my database category. I also chose to focus on adding a ton of extra user functionality to the system to satisfy the software engineering category. The previous system only allowed the user to type in a specific playlist name to print out the name of every song within a given playlist. I focused on expanding this functionality to allow the user to properly interact with artists/songs/and playlists in ways aside from just generic viewing.

The main reason I chose to work with this artifact was its massive potential. A music application holds a massive amount of information ranging from song information to user account information. The initial implementation was very simple, but it did hold the basic frame to begin expanding into what can be a very reasonable music application. Due to this, I know I could expand this application a massive amount in an indefinite number of ways. The application had the potential for database migration, additional user interactivity, new variable components (albums), and even the possibility of a UI. This is all without an actual need to make the songs play audio. This massive amount of potential and the different directions I could take with the application is why I chose it for my first artifact. I believe that the ability to improve the way that data is stored (database category) and make the application much more usable for an average user (software engineering category) provides me with a great opportunity to vastly improve the application in multiple ways.

The proposed changes for this specific artifact were broken up between goals targeted toward database migration and ones targeted towards adding new user functionality to fulfill software engineering requirements. On this note, all the proposed functions were added in addition to more. I proposed adding viewing and generation of artists/playlists/songs and the addition of private playlists. I went beyond this by adding editing functions. This allows the user to make changes to files already existing within the application. The new editing functionality includes the ability to edit names, remove components, and alter the songs in a playlist. This creates a very clean set of functionality that I don’t plan to make any major expansions on, but I will work on cleaning it up more and perhaps make minor additions during the refinement phase and database artifact milestone.

Focusing mainly on the software engineering component of the application. I used this as an opportunity to refresh myself on how to route the user through a complex set of menus. I was also able to use this as an opportunity to use a small set of classes (artist/song/playlist) and their related functions to vastly improve the system complexity. Finally, this segment forced me to look into how to implement SHA-based hashing into Java. This is an important skill that I may be able to make use of later.

Applying this directly to course outcomes leads me to the three primary goals that I listed in my plan. These goals include course outcome 1, 3, and 5. Although I did make progress on these three goals working specifically on the non-database components of this artifact, I will touch upon them again in reference to specifically the database components.

Object 1 is based on creating collaborative environments that allow diverse audiences to interact with the project. I tried to keep other developers in mind throughout the process of adding enhancements and expanding documentation throughout this artifact. The entire project is well organized with several levels of sorting for the functions. For example, the UI functionality is broken down into editing, login, processing, and several other groups. This makes it much easier to get to the part of the document you are working with or need a function from by giving you a high level category to look for. Along with this, every few lines are broken up by a comment that briefly describes what the next few lines of code will be doing. These additions make it much easier for a new developer to jump in and understand what different parts of the program do.

Objective 3 revolves around coming up with a solution making use of valid computer science principles to solve the problem effectively while managing related trade-offs. In this case, I chose to make use of individual classes for students, songs, and playlists. I initially started developing the application using simple string communication between the database and UI and decided that although the class system added more complexity to the system, it would also add a massive amount of expandability, and make the code more readable for a new programmer.

Object 5 is related to the security principles that are included in the application. In this case, I decided that implementing a login system was a great idea for keeping the users’ playlists private from other students. This includes keeping the user’s password secure via SHA hashing. The system can also have its algorithm changed very easily, with one function dictating what algorithm is used. Overall, this increases the security of an individual user tremendously.

The only real challenges I faced during this segment were learning SHA hashing alongside developing two different components separately. This will likely be mentioned in my database milestone review as well, but I must note that it was interesting to decide to work on updating songs and having to think about two parts. I had to think about implementing the menu routing and dealing with the logic on the front end while also going through how I wanted to get the required information from the database. This leads to a very informative and interesting process where you really think about all the different ways you can implement the system. Even now, I have different ideas for how this could be implemented.

One in particular that comes to mind is binding the component classes (artist/song/playlist) to the database to vastly reduce the amount of logic required in the command processor. This would mean that a song class would be initialized with a database connection that it carries as long as it's held. It would use this connection whenever any updates are made to it. For example, let's say you make a new song. That song would take the DB connection and immediately push the new song to the database. It would also automatically make changes as you change its internal values. So, if you changed the name, the database would be automatically updated. The main reason I avoided this was to prevent situations where you want to work with the data within these classes without touching the database.

Functionality added Overview:

* Login System (New or existing user) – Includes SHA-1
* Artist Viewing
* Song Viewing
* Playlist Viewing
* Artist Publishing
* Song Publishing
* Playlist Publishing
* Artist Editing (Including Removal)
* Song Editing (Including Removal)
* Playlist Editing (Including Removal)
* Private Playlists
* Cascading Components (Could be considered DB function)