1. **Please list out changes in directions of your project if the final project is different from your original proposal (based on your stage 1 proposal submission).**

Our final project was almost identical to our original proposal. In regards to the overall idea and use for our application, our final project and original proposal remained mostly the same. The biggest difference between our final project and our original proposal was the removal of the recommendation feature for users who are not currently listening to a song. However, there were some features that we added and removed throughout our project that we will discuss later in this report.

2. **Discuss what you think your application achieved or failed to achieve regarding its usefulness.**

Back in Stage 1, we stated (with regards to our application's usefulness) that our application allows users to share and connect through music, as well as inspire users to explore new music tastes by learning about songs that their friends listen to or recommend. Our application, MusicReal, achieves this through allowing users to post and share the current song they are listening to on Spotify. Our homepage allows their friends to see what song they are listening to and also listen to a short preview of the song.

On the other hand, MusicReal failed to allow users to recommend songs to other users when they are not listening to a song, or see suggested friends based on mutual friends or other factors. Nonetheless, MusicReal can still be a personalized and engaging experience for our users because they can share the current song they are listening to, and explore songs that their peers are listening to. They can view the past songs that they and their friends have posted, and are also able to see the common songs that they and their friends listen to, and how many other users of MusicReal listen to the same songs.

3. **Discuss if you changed the schema or source of the data for your application**

For our schema, we changed it quite a bit. To start off, we got rid of the History table since we were able to get a user's history by calling a query that obtains all the posts in the Post table that match the username we were looking for. Additionally, we changed FriendsRelationships to just Friends and added an Id for each relationship. This is mostly because without the Id we would not have a primary key for the Friends table. We also changed the Post table to Post(postId, time, songId, userName). Instead of obtaining the time late and the date separately, we decided it would be more efficient to just stick to the time that the user had

posted. Lastly, we had three tables that related to the user: UserRef(userToken, userName), Profile(userName, email), and Login(userName, password).

For the source of the data for our application, we stated in our initial proposal that we would use the Spotify API as our source of data. Initially, we weren't able to successfully connect the Spotify API to the GCP, and hence, we used “dummy/sample data” by manually inputting placeholder data, such as usernames and album cover images for the midterm demo. However, once we were able to successfully connect the Spotify API and GCP, we changed our source of data to the Spotify API.

4. **Discuss what you change to your ER diagram and/or your table implementations. What are some differences between the original design and the final design? Why? What do you think is a more suitable design?**

- Added a Friends table that contains id, username, and friendUser
    - To provide a relationship between a user and their friends
- Post:
    - Changed from having postId, TimeLate, and Date to postId, songId, userName, time → instead of obtaining the time late and the date separately, we decided it would be more efficient to just stick to the time that the user had posted, and include additional details such as songId and the username that posted the song.
- Profile:
    - Changed from having userId, name, email to userName, email → We got rid of userId because we were able to use username as a primary key
- Song:
    - Remain unchanged
- UserRef
    - Added userName to table → to link Spotify-generated user token to a specific user in our database

Our relations between tables such as one-one, one-to-many, many-to-one remained the same. The newer design is more suitable since we modified it to function according to our application. In our original design, we would not have been able to join specific tables to implement our features because we did not include shared attributes to those tables. For instance, the Post table did not have songId, so we wouldn’t have been able to map a song to a post.

5. **Discuss what functionalities you added or removed. Why?**

We added a feature to display the shared songs between a user and each of their friends on a friend's user profile. While we were not able to implement the recommendation feature, we still wanted to provide more insights for the user for a more personalized and engaging experience, so we added this feature. Additionally, we also added this feature to incorporate a stored procedure within our backend, thus removing the suggested friends list which was our original idea for the procedure.

Moreover, the functionalities we removed are:

- Allow users to recommend songs if they are not currently listening to a song
- See other users' listening history → replaced with shared songs feature
- Saving song to listen to later
- Share a song from a post on home page
- Suggested friends list
- Number of mutual friends
- Managing password within app → removed because we connected Spotify API so the users use their Spotify account to login, so Spotify manages password changes

Additionally, we removed these features due to a few other reasons. One reason was that we were over ambitious given the time constraints. Another reason was that all of our team members got really sick at some point throughout the semester, creating challenges to collaborate together during some parts of the semester.

6. **Explain how you think your advanced database programs complement your application.**

By using a well established data management system such as SQL that was introduced through this course, our group was able to implement advanced functionalities such as stored procedures and triggers and write efficient SQL queries to access data effectively to project and insert data from our MusicReal application.

7. **Each team member should describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project.**

Rachel: One technical challenge the team encountered was implementing the stored procedure. Our goal for the stored procedure was to return the name, songId, and count of how many shared songs a user had with a friend. Initially, our stored procedure kept outputting null values in all or certain columns. To address this issue, we ran the individual subqueries in MySQL Workbench to ensure they are outputting the correct data, and also ensuring that our variable names are different/easily distinguishable to not confuse ourselves and to prevent the query from reading the wrong variable. For example, differentiating the variables we declared at the top of the stored procedure and the variables we declared within the table within the stored procedure.

Anjolee: A technical challenge that we encountered was when we were implementing shared songs between users. The front-end was calling the API to obtain a list of all the shared songs between the two users and then storing that information into a useState. However, the state was only showing the first element in the array. This is due to the fact that useState updates asynchronously. Therefore, using a temp array and repeatedly setting the state to that array did not work. In order to fix this, instead of using a temp array, we used a spread operator and set the state to that along with the new data from the API.

Leo: A challenge we encountered while developing the backend services for our fullstack project was debugging our code. After deploying our backend service to Google Cloud Platform, we could not obtain the error logs. This is because Google Cloud Platform only considers a subset of errors to be errors, ie. only some errors from our backend server show up on the Google Platform App Engine console. This means that when designing the backend, it is useful to have an environment that can run in both the Google Cloud Platform version as well as a local backend version. When encountering issues, we could run the backend locally and view any errors in the console, which we used to debug difficult-to-find bugs and errors.

Aumkar: A challenge that we faced initially was connecting to the Spotify API using GCP. We weren't sure how to utilize the temporary tokens that were generated after a user is authenticated through Spotify, and how to access aspects such as the current song information through this. We had to figure out an approach where we stored this information and linked it to the username of the user in our database.

8. **Are there other things that changed comparing the final application with the original proposal?**

   There were no other changes we made between the original proposal and final application, besides the ones mentioned in other questions throughout this report.

9. **Describe future work that you think, other than the interface, that the application can improve on.**

Since we are extensively using relations between friends in our application, we believe if we were recreating the application, we would have explored Neo4j as our database management system instead of SQL. Additionally, certain components such as the output of the backend returned JSON objects, and since NoSQL solutions such as MongoDB are extensively used and are more efficient for JSON objects compared to SQL based solutions, we would have also explored this.

In our backend service, we need to give users access tokens for their spotify API calls to authorize. However, if a user idles on the app for a while, the app will error out because all access tokens have a timeout counter. If we had more time, we would have included code to check time spent on the app for each user, and assign refresh access tokens for those affected users. We could also clean out expired users from our cache every 6 hours.

10. **Describe the final division of labor and how well you managed teamwork.**

Rachel and Anjolee mainly worked on the front-end, whereas Leo and Aumkar mainly worked on back-end; however, we all helped with various aspects of the project. For example, all of us worked on the stored procedure and connecting the API and GCP in the back-end. Overall, we managed teamwork very well and helped each other with debugging code and resolving errors when needed. Additionally, with team members being sick at various times during the project, we were able to step in and help each other.