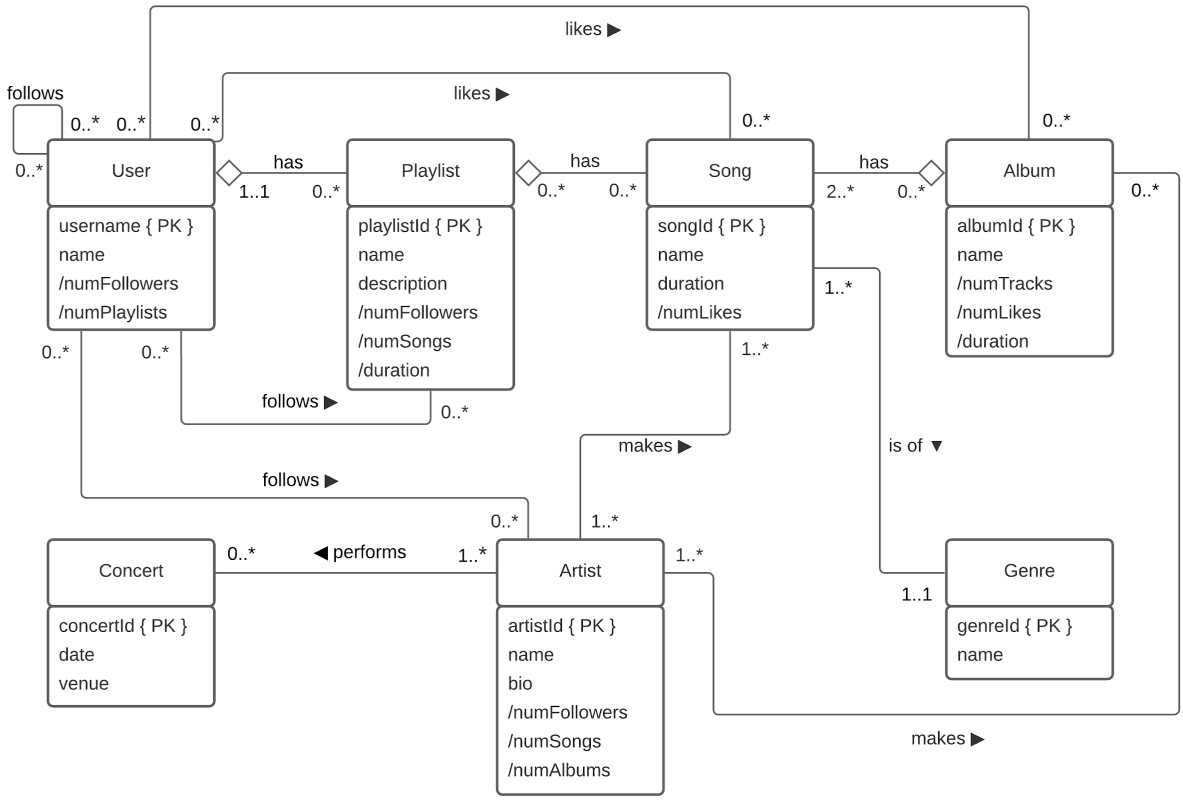
README

In order to successfully build and run the project, first you will need to recreate the database schema locally using the dump file or init_db.sql script. Next, you will need to edit the application.properties file (/src/main/java/edu/northeastern/resources/application.properties). Spring.datasource.username and spring.datasource.password should be updated to reflect your local MySql user credentials. If the local instance was not started on port 3306 or the generated database is not named ‘playlist’, you should update the spring.datasource.url property to reflect these differences.
Finally, you can build and run the project. Gradle Wrapper makes it so Gradle is automatically installed the first time you run the build command. On Unix based systems like Linux and Mac, you can simply run **./gradlew build** followed by **./gradlew bootRun** in the root folder and on Windows, **gradlew.bat build** and **gradlew.bat bootRun** can be used. If the build fails, still attempt the bootRun command as there may be dependency classpath issues. When the project is successfully built and running, navigate to http://localhost:8080/api/swagger-ui/index.html, replacing 8080 with a different port if its busy on your machine. From here, you can view entity models, controllers, and provide payloads to use each REST endpoint.

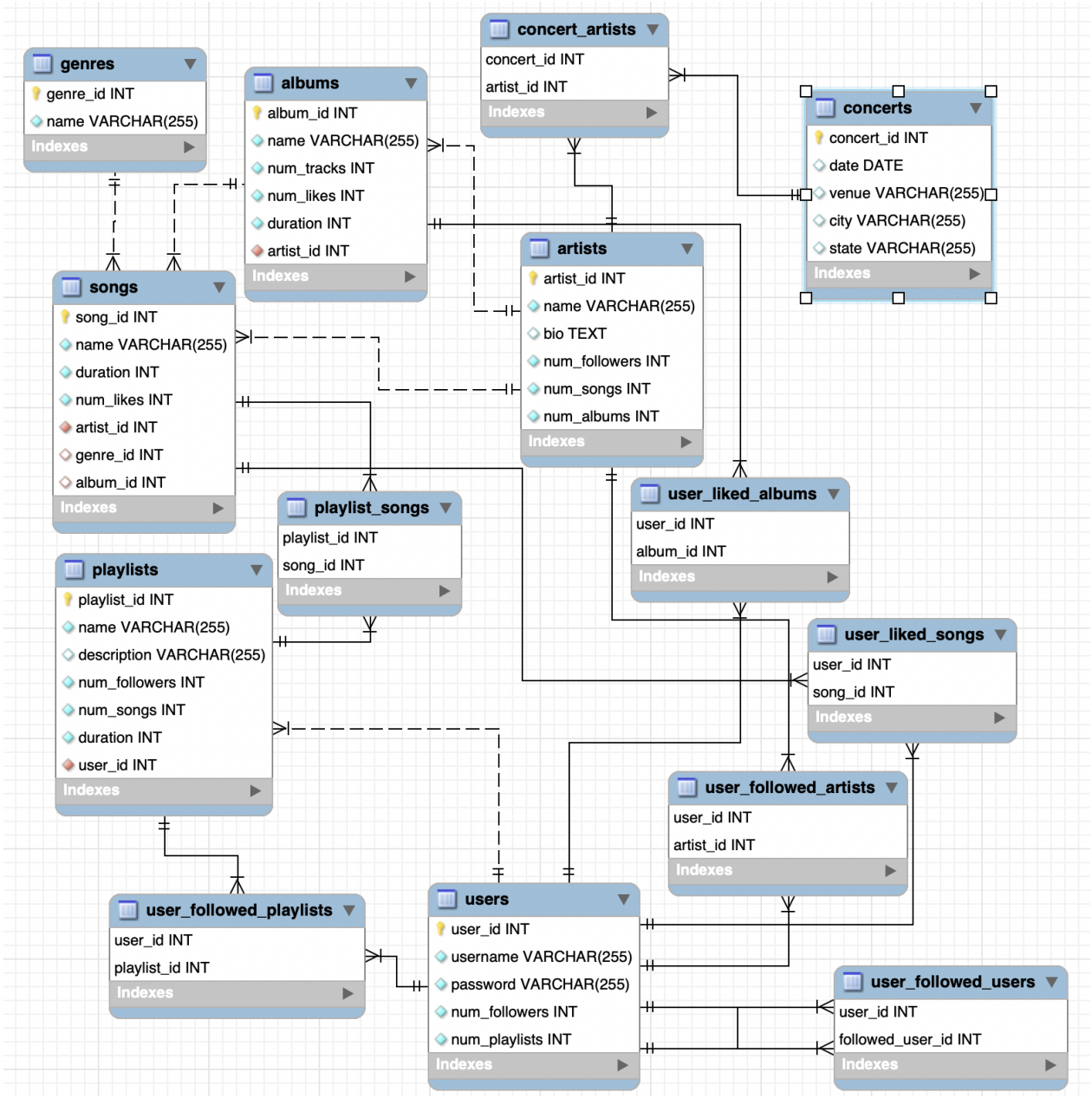
Technical Specifications

- Database
    - Relational
    - MySql DBMS
- Backend
    - Java 11 (OpenJDK)
    - Dependencies
        - spring-boot-starter-data-jpa → Java persistence API
        - mysql:mysql-connector-java:8.0.23 → MySQL JDBC driver
        - org.springframework.boot:spring-boot-starter-web → Spring MVC for RESTful applications
        - org.projectlombok:lombok:1.18.20 → entity annotation processor
        - com.fasterxml.jackson.core:jackson-core → JSON object mapper
        - io.springfox:springfox-boot-starter:3.0.0 → Swagger UI for displaying documentation and using RESTful endpoints
        - Others include log4j logging and default JUnit testing framework
- Build and Automation
    - Gradle v6.8.3

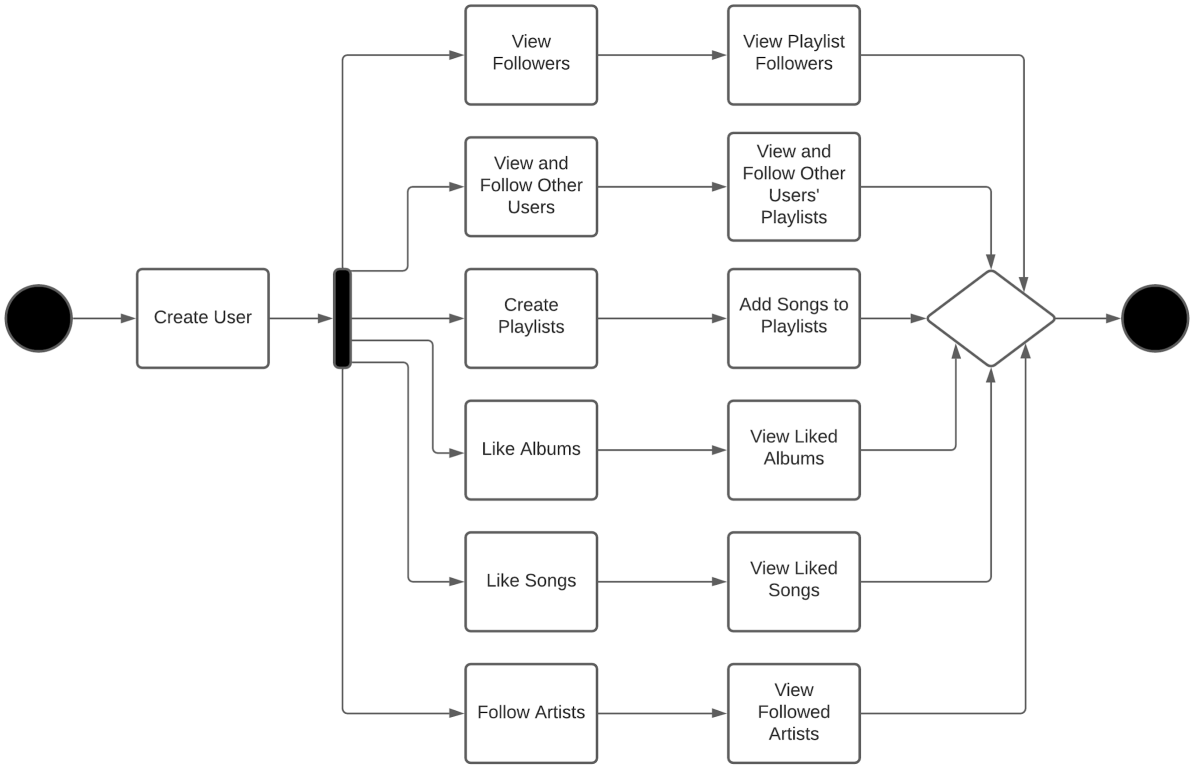
UML Diagram


likes ▶
follows
likes ▶
0..*
0..*
0..*
0..*
0..*
User
has
1..1
0..*
Playlist
has
0..*
0..*
Song
has
2..*
0..*
Album
0..*
username { PK }
name
/numFollowers
/numPlaylists
playlistId { PK }
name
description
/numFollowers
/numSongs
/duration
songId { PK }
name
duration
/numLikes
albumId { PK }
name
/numTracks
/numLikes
/duration
0..*
0..*
follows ▶
0..*
1..*
1..*
makes ▶
is of ▼
follows ▶
0..*
1..*
Concert
0..*
◀ performs
1..*
Artist
1..*
Genre
1..1
concertId { PK }
date
venue
artistId { PK }
name
bio
/numFollowers
/numSongs
/numAlbums
genreId { PK }
name
makes ▶

Logical Design



User Flow



The user flow is particularly supported by the user controller in the backend, which uses a repository object to communicate with the database tables and objects such as procedures. Since this project a REST API, post mappings are used to create users, add songs, albums, playlists, concerts, genres, and artists. Get mappings on the controller are used to view user followers, view users followed, created playlists, liked albums, liked songs, and followers artists. Put mappings were used to update these tables, and delete mappings are used to remove records from these tables.

Lessons Learned

The main technical expertise I gained throughout this project was learning how to efficiently design databases and use them in meaningful ways. Another aspect was the importance of database objects, something I didn't know existed before this course. Database triggers and procedures are super important and helpful in separating database and application functionality, and what operations are exposed to external systems such as the project backend. Time management was the main issue I faced during this project and it showed me the importance of effective project management in real world scenarios - whether it be sprint cycles, defining requirements, or preventing scope creep. If I were to redesign or redo this project, I feel as if a NoSQL database would have been more efficient for the music data domain due to the sheer number of circular references and lists such as playlists. The document-based model would have made more sense for storing large amounts of this embedded data.

Future Work

The database can be used by people who want to keep track of their artists, songs, albums, and playlists independent of a specific music streaming platform. There are many areas for potential added functionality. Time limitations pushed building a website out of the scope of the project, so adding a more interactive UI would help better user experience. Role management would be useful in distinguishing between users who can write specific records such as songs and albums to the database and users that can only read these records.. Another area for increased data reliability is by using Spring Boot supported transaction processing and using a reactive programming library to increase efficiency and perform more operations asynchronously (such as the reactor-core library). Logging can be updated to be more informative and Spring security can also be configured to secure the API endpoints. Finally, a search endpoint would be really useful to users due to the sheer number of potential records in songs, albums, artists, users, and playlists, and this functionality can potentially be implemented with third party endpoints, such as Spotify's developer search API.