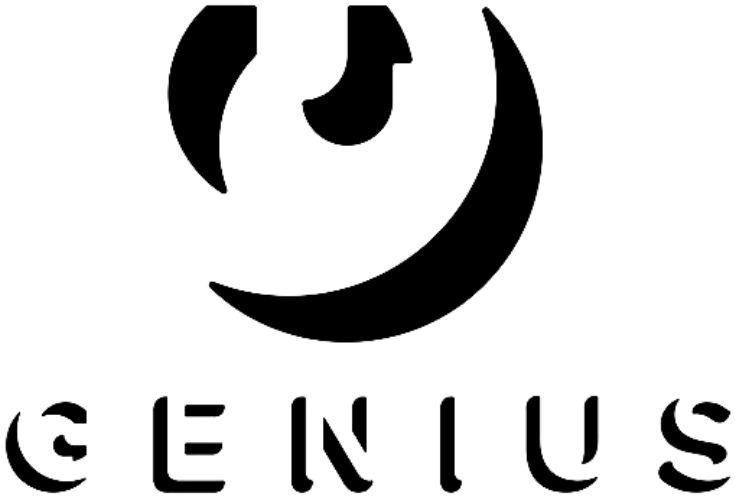
Genius Project Document

Advanced Programming - Spring 2025

Course Instructor: Dr. Saeed Reza Kheradpisheh



Introduction

In this project, you must use Java to design and develop an application inspired by **Genius**, which is an online music platform. Genius is a collection of song lyrics and musical knowledge where users can view content as lyrics and information about songs and provide their own annotations. They can also contribute by voting, editing and adding information.

Note that this is a solo project meant to be undertaken by each student individually.

Objective

A Major goal of this project for you, is to exercise your creativity and implement your own unique approach to building the application. No template code will be provided for this project as you are expected to build it from start to finish.

You are allowed to use any Java library to accomplish the project goals, but you are expected to demonstrate a strong understanding of core OOP concepts such as **inheritance**, **polymorphism**, **abstraction**, and **encapsulation**.

Exploring Genius

Before diving into development, take a moment to visit **Genius.com** and explore its features. I strongly recommend to keep an eye on the website while reading the document.

Main Tasks

Your goal is to create an app and implement a **well-structured and practical object-oriented design** based on the principles you have learned. This means representing data as objects and establishing meaningful relationships between them. The way you design your app is completely up to you. Here is the explanation of the project and the **mandatory features** you should implement:

Accounts

In this project, users can create accounts with different roles, which determine their permissions and interactions within the system. The two primary roles are **User** and **Artist**.

All accounts share common attributes such as **name, age, email, username, and password**. To ensure efficient account management, the application should leverage **object-oriented principles**, defining a base **Account** or **Person** class from which different user types can inherit.

Account Features

- **User Authentication**: All users must be able to **sign up** and **log in**, with a unique identifier such as a **username**.
- **Role Selection**: The role (**User** or **Artist**) is chosen during registration and determines available features.

Roles

User (Regular Listener)

This is the most common type of account, representing music enthusiasts who primarily consume content but also contribute by improving lyric accuracy.

Permissions & Features:

- **View Song Lyrics**: Users can browse and search for lyrics of various songs.
- **Suggest Edits**: If a lyric is incorrect, a user can propose an edit. These edits require approval from the respective artist to maintain quality and authenticity.
- **Follow Artists**: Users can follow their favorite artists or songs to receive updates on new releases or accepted lyric edits.
- **Commenting**: Users can comment on the songs to share their opinion.

- **User Interface:**
    - Users should be able to **view the list of artists they follow**.

Artist

Artists are the verified creators of songs, granting them control over their content.

Permissions & Features:

- **Edit Their Own Lyrics**: Unlike regular users, artists can directly update their song lyrics without approval.
- **Approve or Reject Edits**: Artists have the authority to review user-submitted edits and decide whether to accept or reject them.
- **Create New Songs**: Artists can add new songs to the platform, ensuring a constantly growing database of content.
- **Create New Albums**: Artists can add new albums containing multiple songs to the platform.
- **Artists** can only be followed and they **can't follow other users**.
- **User Interface:** They have a **profile page** that must display:
    - Main information about them.
    - Their most popular songs.
    - A list of all their songs and albums if needed.

Admin (System Manager)

The **Admin** role is crucial for platform maintenance and managing user roles. If user-generated lyric edits become a major part of the platform, an **Admin** role could help manage content.

Permissions & Features:

- Review and approve/reject user-submitted lyric edits when artists are inactive.
- Approve or verify artists: When an artist registers in the application, a request is being sent to an admin for approval. After admin approval the registered artist can log in and start activity.

Albums

The platform should support **albums**, allowing artists to organize their music into structured collections.

Features:

- **Creation** (Artist Exclusive)
    - Artists can create **new albums** and add songs to them.
    - Each album has **only one artist**.
    - Album includes details such as **title, release date, and tracklist**.
    - **Tracklist** contains the songs of an album in a special order
- **User Interface**
    - Users can **browse albums** and view all songs within them.

Songs

Songs are the core content of the platform, each containing detailed metadata, lyrics, and artist information.

Features:

- **Creation** (Artist Exclusive)
    - Artists can create **new songs**.
    - Each song is **single** or **only** belongs to **one album**.
    - Song includes details such as **title, album, artists, lyric, genre, tags, views count, comments, release date**.
    - If a song is **single** it **doesn't belong** to any **album(you can set the album field to a unique object or string called single dependant on your design)**.
    - By **each time** viewing the song, the **views count** increases.
- **Featured Artists**
    - Songs can have **multiple artists**, permissions can be set so that **co-artists can edit their respective lyrics**.
- **Lyric Contribution**
    - Users can **suggest lyric edits**, which require artist approval.
    - Admins can **review** lyric edits when artists are **inactive**.

Comments

The comment system allows users to discuss, interpret, and provide feedback on lyrics, songs, and albums.

Comment Features:

- **Engagement**
  - Users can comment **only** on **songs**.
- **User Interface**
  - Comments should be displayed in a separate **section** sorted by **date**.
  - The **username** of the commenter should be displayed.

Essential Features for Your Application

1. Display Entities

- Once again your app must be able to show Account page, Album Page and song page and their information to users
- Users must be able to Navigating through this Entities

2. Charts

- Display a **list of top songs** based on **views**.
- Show **view counts** for each song.

3. Search Functionality

- Allow users to **search for an artist, song, or album**.
- Display relevant **results** with quick access to details.

4. See following Artists

- Allow users to see list of artists they follow
- See new song or new album of their following artists

Seed Data and Initialization

Importance of Seed Data

To **demonstrate the application's functionality** and ensure that its structure works correctly, your program **must generate initial data automatically**. This includes:

- **Multiple user accounts** (both regular users and artists).
- **Several artists** with their profiles.
- **Multiple songs and albums**.
- **A set of comments** on different songs.

Why Is This Critical?

- It **validates** that all relationships and features work as intended.
- It **improves the project's clarity** by making it easy to test functionalities.
- **Grading heavily depends on this**, so **do not skip this step**.

Notes

- Make sure to use a Package Manager for your project (Maven or Gradle)
- Create a new Git branch "Develop". Any additional branches you create should be named appropriately and merged into Develop. Do not delete old branches after merging them.
- You must override certain methods such as toString() and equals() in some of the classes you implement.
- Use Design Patterns - You should use appropriate design patterns to ensure your code is modular and maintainable.

Project Readme

You are expected to write a detailed GitHub README for the project. You must write it in **Markdown** format.

A README file is a guide that gives users a description of the project you have worked on, with guidelines on how to use a project. It is the first file a person will see when they encounter your project, so it should be fairly brief but detailed. A well-written README can also help **you** focus on what your project needs to deliver and how. your project's README should be able to answer the **what**, **how**, and most importantly the **why** of the project.

Your README should include the following sections:

- **Project Title:** State the project's name.
- **Table of Contents:** A table of contents can help users navigate to specific sections.
- **Description:** Provide an overview of what the project does and its purpose.
- **Usage:** Explain how to use the project. Include code examples where you see fit.
- **Demo/GIFs/Images:** Provide visuals to showcase the project in action.
- **Credits:** Acknowledge and attribute any contributors, libraries, dependencies, or resources used in the project.
- **Changelog:** Document changes made in each version of the project.
- **Contact Information:** Offer ways for users to reach out for help or support.

For reference, you can explore the following GitHub repositories with well-structured READMEs:

1. Express
2. AI Face Recognition Attendance System
3. AI English Tutor

Bonus Tasks

Below you can find a collection of Additional features you can implement in your project. While this is our recommended list, you are free to add more optional and challenging features, **your creative effort won't go unnoticed.**

Remember to mention any bonus task you finish when you're presenting your code.

1. Improve existing **mandatory features** for example you can create a more advanced search system or improve user experience.
2. Ensure the data used by your application is **stored** properly. You can use **static lists,** simple **file** systems to achieve this or utilize a **database management system** such as PostgreSQL or MongoDB. We recommend using a file system or database.
3. Design a GUI (Graphical User Interface) for your application using **JavaFX**. The GUI must offer as many options as the CLI version of the project. Using images and icons or adding animations is optional.
4. Learn how to use **password hashing** and utilize it to increase the security level of account creation and user authentication.
5. Add an **articles** section to your application where users can create, view, and interact with written content. Each article has an **author**(as a new role), a title and a description.
6. Add the option to **like** or **dislike** a comment.
7. Add the **Questions & Answers** section, where some frequently asked questions about songs are answered by artists and/or contributors.
8. Allow users to **follow** artists. Songs from the following artists should be displayed on the front page.

9. Add a **lyric analysis** section. Users can click on a part of the lyric and view an analysis revolving the meaning of the lyric written by the artist or contributors.
10. Implement a **history** section that allows users to view their recently viewed songs.
11. Implement a **notification** system for new tracks from followed artists and updates on lyric change requests.
12. Create a **recommendation system** to suggest songs to the user based on the songs they viewed in the past.
13. You can assign a UUID(Universal Unique Identity) to entities.
14. Online data Updating : Use Online API to update and expand your data this requires using **files** or **database** in this case if data must be stored in online mode and used in offline mode.
15. your project can have **visually appealing interface (UI)**
16. Consider clear navigation, responsive design, and **user-friendly interaction (UX)**

Useful Tools & Packages

You can find a selection of helpful libraries, apps, and tools here. **You are not required to use any of them**, but they may prove useful in your project.

- **UUID:** The Universally Unique Identifier class can be used to generate IDs for objects that need to be uniquely identified. Note that using a UUID for every class might not always be the best approach.
    - What is a UUID?
    - UUID in Java

- **Scene Builder:** A visual layout tool that allows developers to design a JavaFX UI by dragging and dropping components onto a scene, and generating FXML markup code that can be used in Java applications.
    - Scene Builder installation tutorial
- **JavaFX:** A Java library used for building applications with a modern user interface. JavaFX supports CSS for styling, FXML for UI design, and integrates well with Java to create responsive applications.
    - A Playlist for learning JavaFX
- **PostgreSQL:** A powerful open-source object-relational database management system that provides robust data integrity and advanced query capabilities.
    - SQL Tutorial
- **Gson:** Gson is a Java library for converting Java objects to JSON format and vice versa. It allows developers to serialize and deserialize Java objects to JSON strings and parse JSON strings back into Java objects.
    - Gson - Object Serialization

Evaluation

- Your code should compile and run without any errors.
- Your code should be well-organized, readable, properly commented, and should follow clean code principles. Use appropriate Java naming conventions.
- Your code should follow **OOP** principles. Concepts such as **polymorphism** and **inheritance** should be efficiently used in your code.
- You should use **Git** for version control and include meaningful commit messages. Use as many branches as necessary.

- You will be required to present your code to a team of judges at a later date for the final evaluation.
- You need to write a well structured README.md for your project. This not only helps you to create a better repository for one of your more serious projects in this course but also provides a better way to judge and evaluate your code.

Submission

- Push your code to GitHub
- Complete the README file
- Merge your “Develop” branch with the main branch of your fork

The deadline for submitting your project is **Sunday, April 6 (17th of Farvardin)**. Any commit made after this date will not affect your score.

Good Luck & Keep On Learning!