Student: Juliane Alvarado Salgado

Course: SWENG 837, SP II – 2024

Professor: Santosh Nalubandhu

Class Project

Chosen Software Problem:

Build a recommendation engine for a streaming service: Utilize machine learning and user data to recommend personalized content to users.

Problem Statement and Requirements

Business Requirements:

Clearly define the problem the system aims to solve.

Build a recommendation engine for a music streaming service.
Utilize machine learning and user data to recommend personalized content to users. Please note that this design assumes the streaming service already exists – users can query for specific songs. This recommendation service will analyze user specific search, customization decisions, and feedback to identify predicable behavior and generate personalized content. Recommendation Engine will also use existing data for the given user demographic (age, location, likes, provided feedback) and compare to others similar to the user to show recommended content.

Specify the functionalities the system needs to provide.

a. Analytics – use this tool to gain insight into user behavior, content performance, and trends – allowing data-driven decisions to be made to enhance the experience.
b. Security – Protect sensitive personalized data generated by machine learning of the user. Restrict access from suspicious third-party integrations and domain. Ensure all media and content has no violation of copyrights/community guidelines to not show to user.
c. Scalability – be able to support an enormous user base – be able to host millions of users and eliminate any glitches, bugs because server can't hold the growing user base.
d. Customization Options – allow users to have control over platform (multi-language support, genres to display, specific decade of songs, etc.) Included are parental controls (if underage person using engine, block explicit streaming content)
e. Feedback – Consider Ratings (5 stars) on the content. Recommendation Engine will take that data to further personalize content.
f. Algorithm – machine learning component for content recommendations.
g. Global Content and Library – offer a wide variety of music to cater to different tastes and preferences.

Identify the target users and their needs.

Defining the target users and their needs can benefit the overall system design as understanding the needs allows for developing a recommendation engine that can cater to different usage of the system. The system's target users and their needs are:

- *Music enthusiasts* – need a platform to recommend interesting and new music that is based on their tastes and preferences.
- *Fitness enthusiasts* – need a platform that can curate high–tempo, high energy music based on their exercise routines.
- *Casual Listeners* – need a platform that offers a curated selection of current popular songs they may enjoy just for relaxation and entertainment. They don't want to spend too much time exploring and discovering new music, it can be presented to them.

Outline any business goals the system should support.

The recommendation engine should support several key business goals to ensure its success in the maker. The system should support the following two goals:

- *Customer Satisfaction*: enhance user retention by providing a valuable tool that recommendations satisfy the customer.
- *Growth:* increase user engagement and expand the user base.

Non-Functional Requirements:

Define performance requirements like scalability, response time, and throughput.

1. Load Balancing –The system will evenly distribute incoming traffic across multiple servers/instances. This distribution will ensure optimal resource utilization and prevent any overloading on individual components.
2. Database Optimization – The system will use database optimization techniques to improve database performance and reduce query execution times.
3. Asynchronous Processing – The system will utilize asynchronous processing for non-blocking operations to reduce and improve latency and system responsiveness, respectively.
4. Horizontal and Vertical Scaling – The system will support adding more servers/instance and increasing the resources of existing servers/instances to handle traffic fluctuations and accommodate future growth.
5. Scalability Testing – The system will implement conduct scalability testing to evaluate the horizontal and vertical scaling.

Specify security requirements like authentication, authorization, and data encryption.

1. Session Management – The system will generate unique session identifiers, store data securely, and invalidate sessions after a period of inactivity to prevent hijacking and fixation attacks.
2. Data Privacy and Compliance – The system will comply with data privacy regulations such as HIPPA to provide transparency and control over personal data.

3. Authentication – The system will require the users to authenticate themselves before accessing the system with a multi-factor authentication.
4. Data Encryption: The system will protect the data transmitted between client and server using HTTPS/TLS to protect against man-in-the-middle attacks.

Outline maintainability requirements like code modularity, documentation, and testing strategies.

1. Code Modularity and Reusability - The system will be designed with modular and reusable code for maintainability and extension.
2. Continuous Integration and Continuous Deployment - The system will implement CI/CD pipelines to automate build/test/deployment processes.
3. Logging and Monitoring - The system will implement logging and monitoring solutions to capture system events, errors, and performance metrics.
4. Containerization – the system will use containerized application components using Docker to simplify deployment.

Indicate any other non-functional requirements relevant to the system's success.

1. Cross-Platform Compatibility – The system shall be compatible with wide range of devices, OS and web browsers.
2. Localization – The system shall support multiple languages and cultures to cater to a global audience.
3. Feedback – The system shall incorporate feedback mechanisms to collect user feedback, track interactions and system performance for product development decisions.

System Design using Domain Modeling

UML Use Case Diagram

This section covers the visual representation of the system's actors (users and external systems) and their interactions with the system. The primary, supporting, and offstage actors are defined in Table 1.

Table 1. Use Case Actors

Type	Actor	Goal Description
Primary	User	The user uses the streaming service.
	Recommendation Engine	Utilizes user customization, feedback, analytics and algorithms to generate personalized content for user.
Supporting	Music Database – Apache Cassandra	Contains wide variety of music from genres, style, history (date).
	Streaming Service	Provides the interface user uses to stream data.
	Machine Learning	Creates the general-purpose algorithms that are used to obtain the useful user information for personalized content creation.
	Other User	Other user data affects (and can be affected) other users' data as the engine learns from all users.
Offstage	Copyrights and Violations Department	Streaming Service content will need to follow copyright and violations validation.

	Regulatory Compliance	AI and ML need to align with specific regulations relevant to the industry.

Now, Use Cases must be defined. The following lists use case names for the software system. The naming convention of the use cases follows "verb-phrase" + "noun-phrase". All use cases are *partially dressed;* essential steps and variations are written in detail and there are supporting sections. For a briefing of the use cases, focus on scope, primary actors and main success scenario which are sequence of steps that must be executed by the actor.

Use Cases:

1. **Use Case Name: Generate Playlist**
    - Scope: Recommendation Engine will generate a playlist based on the algorithms learned by machine learning about the user to the user.
    - Level: System
    - Primary Actor: User
    - Preconditions: Recommendation Engine should already have data from user to use the algorithms for personal content display.
    - Main Success Scenario:
      1.1.User Hits 'Personalized Playlist'
      1.2.Recommendation Engine gets user data from database.
      1.3.Recommendation Engine uses algorithms to generate the list.
      1.4.List is displayed to the user.
2. **Use case Name: Display Top Content**
    - Scope: Recommendation Engine will generate the top content of user based on the data that engine knows about the user.
    - Level: System
    - Primary Actor: User
    - Preconditions: Recommendation Engine should already have data from user to use the algorithms for personal content display.
    - Main Success Scenario:
      2.1.User hits 'View Top Content'.
      2.2.Recommendation Engine gets user data from Database.
      2.3.Recommendation Engine uses algorithms given user data (usage, and history) to generate the list.
      2.4.List is displayed to the user.
3. **User Case Name: Display New Artists**
    - Scope: Recommendation Engine will generate new artists of user based on the data that engine knows about the user.
    - Level: System
    - Primary Actor: User
    - Preconditions: Recommendation Engine should have already have data from user to use the algorithms for personal content display.

- Main Success Scenario:
  3.1. User hits ‘View New Artists’.
  3.2. Recommendation engine queries user data and queries the database for recent artists add-ins to the system.
  3.3. Recommendation Engine using algorithms filters through the data and generates a list for user based on model parameters and content library.
  3.4. List is displayed to the user.

4. **Show Similar Playlists**
   - Scope: Recommendation Engine will generate a list based on what other users like user are listening to.
     - Level: System
     - Primary Actor: Recommendation Engine
     - Preconditions: Recommendation Engine should already have data from users to use the algorithms for personal content display.
     - Main Success Scenario:

     4.1. User hits ‘What are others Listening to’.
     4.2. Recommendation engine finds all users like user, sorts, and filters for top content.
     4.3. List is generated using the algorithms from the recommendation engine and history about the users.
     4.4. List is displayed to the user.
5. **User Case Name: Search For Songs**
   - Scope: User will search for songs based on chosen categories (genre, style, data), recommendation engine will generate list based on the parameters and data already learned from user.
   - Level: System
   - Primary Actor: User
   - Preconditions: Recommendation Engine should already have data from users to use the algorithms for personal content display.
   - User is looking for Song, of a specific genre, style, and no specific date.
   - Main Success Scenario:
     5.1. User selects song, genre, style, and specific date to search for.
     5.2. User hits ‘Search’ button.
     5.3. Recommendation engine takes the parameters provided by user, and using algorithms, generates the list based on learned behavior/data.
     5.4. List is displayed to the user.

The use cases have been identified and defined; now the Use Case Diagram is created as shown in Figure 1. All Use cases are inside the system boundary “Streaming Service System”. All the primary and supporting actors are outside of the system boundary, and each actor has at least one connection to one use case of the diagram. There are no offstage actors in the diagram.

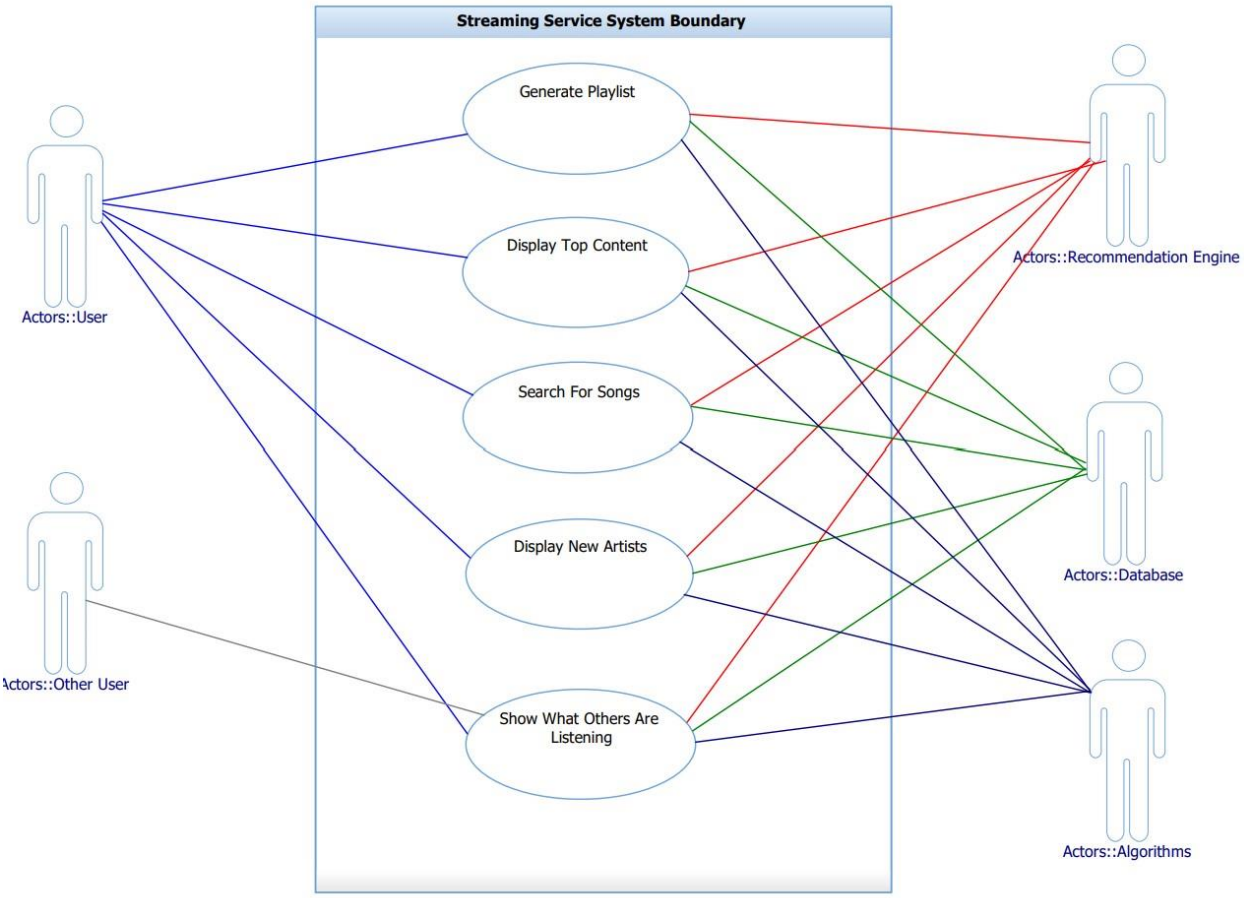

Figure 1. Use Case Diagram.

Each use case defined has a system sequence diagram (SSD). The system is treated as a black box, so all internal operations inside the system are not shown here. It is a very high-level diagram showing basic interaction between the actors to the system. In creating the use cases main case scenario, the overall steps are the same for all use cases. It starts with:

1. User takes action.
2. System is triggered to act and communicates with the Recommendation Engine.
3. Recommendation engine extracts data from Database and feeds it to the Algorithm.
4. The algorithm generates the personalized list.
5. List is displayed to the user.

Figure 2 demonstrates this generic sequence that is applicable to all use cases in this system.

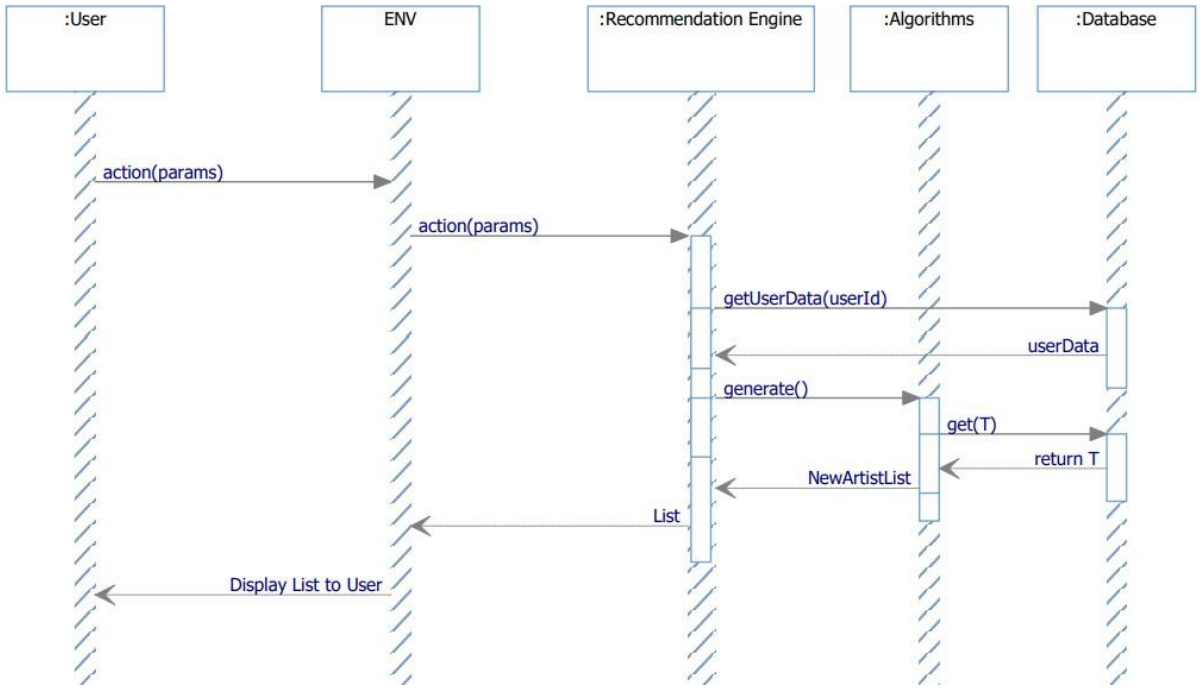

Figure 2. Generic SDD for Use Cases.

UML Domain Model

The following model identifies key entities and their relationships within the problem domain, independent of any specific technology.

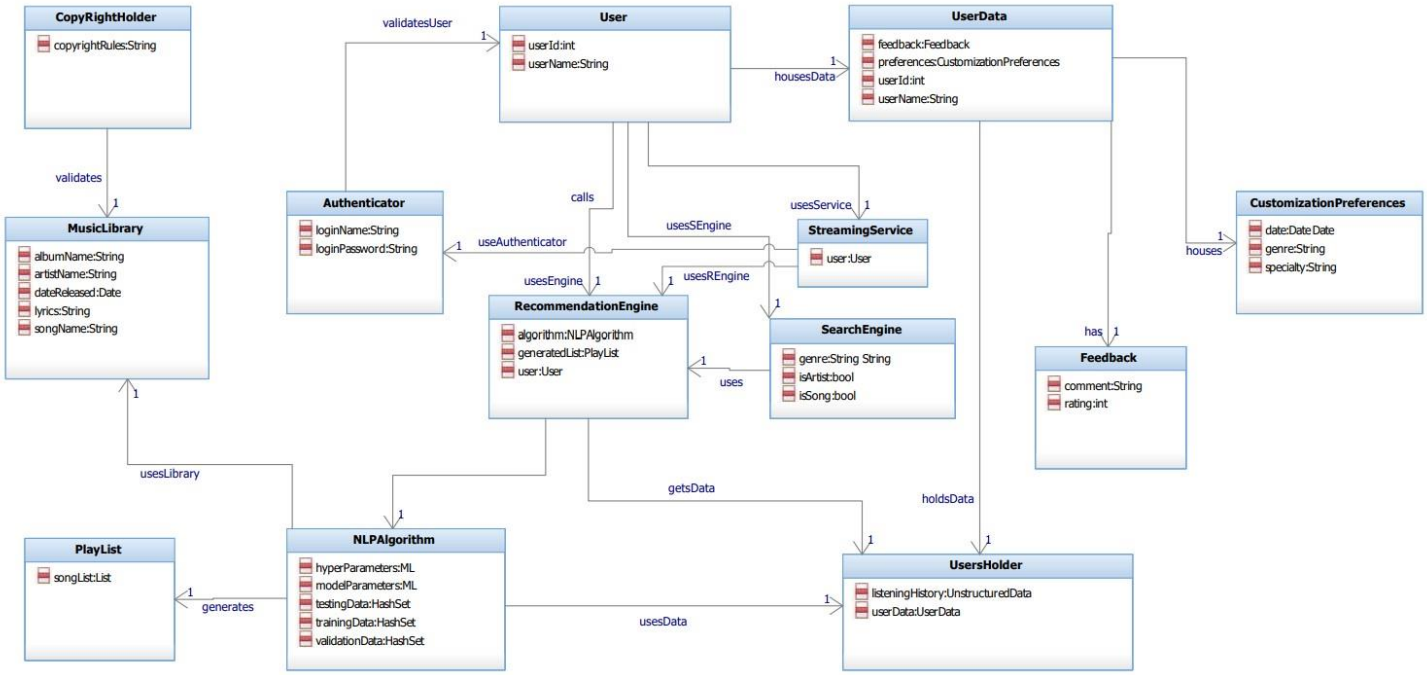

Figure 3. Domain Model

The domain model was created using the “Conceptual Class Category” list. The purpose is to create potential classes based on the list. There are categories such as: roles of people, events, processes, catalogs, records, etc.... Once the table has been filled, classes that are redundant, vague or irrelevant are pruned. After classes have been pruned and only “good” classes are left, attributes can be defined. Class attributes are logical data values for the object.

Figure 3 shows that important conceptual classes are User (UserData, Feedback, CustomizationPreference), Streaming Service and its Engines, Algorithm, and Music and Users Holder. All of which are essential as they are needed for the system to function.

The system cannot function without the User and its Data and there needs to be UsersHolder to hold the data. The system needs a Recommendation Engine that uses Machine Learning NLPAlgorithm to generate personalized data. The NLPAlgorithm needs user Feedback and any Customized Preferences to train and update the model that generates the data. This system is a streaming music service, so it needs a Music Library to hold all that music data.

UML Class Diagram

Figure 4 translates the domain model into a set of classes, their attributes, and relationships, reflecting the system's functionality.

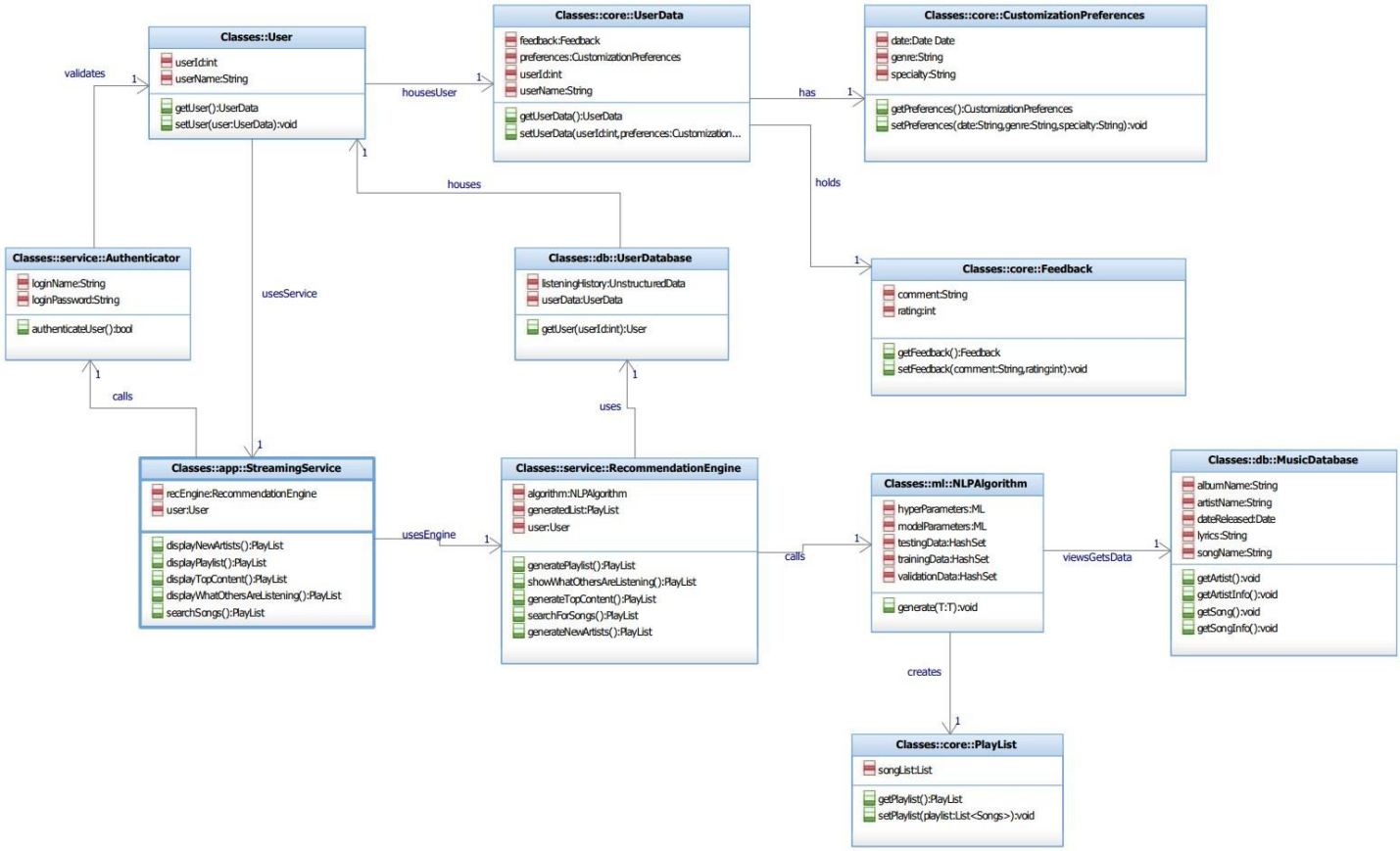

Figure 4. Class Diagram.

From the domain model to the class diagram, a couple of things have changed:

- The MusicLibrary is called the MusicDatabase that houses all the music data system uses and needs.
- The UsersHolder has become the UserDatabase, same explanation as the MusicDatabase.
- SearchEngine has been removed from class diagram as it serves no real purpose and acts same way as recommendation engine (from the domain model, SearchEngine called the RecommendationEngine).
- CopyRightHolder has been removed from the class diagram as it holds no value to the overall system.

The class diagram also holds methods and further defines the attributes. The methods correlate to the Sequence Diagrams that are defined in the following section.

UML Sequence Diagrams

The following diagrams demonstrate the message flow between objects in the recommendation engine domain that participate in specific use cases, depicting the interaction sequence. Note that the interaction sequence is similar for all in terms of the participating conceptual objects and message flows. This is due to the design pattern of creating reusable code. Due to streaming service that uses the algorithm, the code can be templated in a way that the system can take any parameter and know how to feed to the algorithm to generate the personalized data.

SOC 1

Table 2 breaks down a System Operational Contract for Use Case 1 of the system. The operation, responsibilities, outputs, and any pre- and post-conditions are defined. Figure 5 shows the interaction between the conceptual classes defined in the Domain Model.

Table 2. UC1 SOC.

Name Of Operation:	**generatePlayList(userId:int)**
Responsibilities:	Streaming Service displays recommended playlist to User using the Machine Learning Algorithms.
Type:	System
Cross Reference:	Use Case 1
Exception:	If the database is not available, indicate that there was an error.
Outputs:	System Displays List.
Pre-Conditions	Users' data already exists in UserHolder; NLPAlgorithm already has a model to generate the desired data.
Post-Conditions:	NLPAlgorithm takes in any feedback from User and uses it enhanced its model.

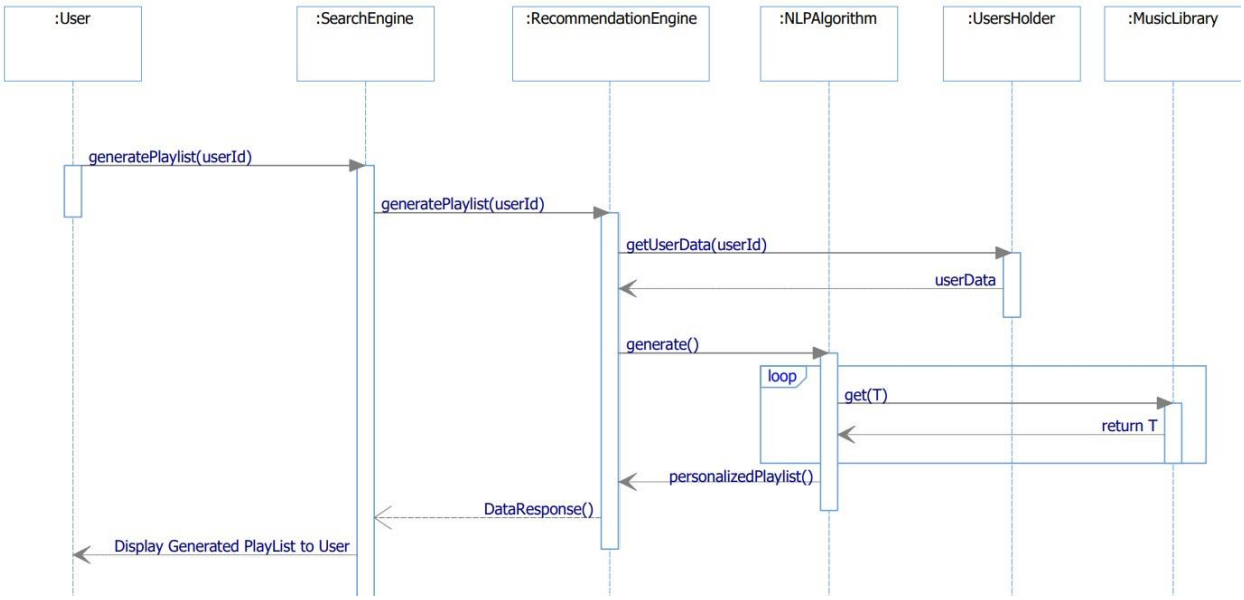

Figure 5. UC1 SOC Interaction Diagram.

SOC 2

Table 3 breaks down a System Operational Contract for Use Case 2 of the system. The operation, responsibilities, outputs, and any pre- and post-conditions are defined. Figure 6 shows the interaction between the conceptual classes defined in the Domain Model.

Table 3. UC2 SOC.

Name Of Operation:	**displayTopContent(userId:int)**
Responsibilities:	Streaming Service displays users top content List to User using the Machine Learning Algorithms.
Type:	System
Cross Reference:	Use Case 2
Exception:	If database is not available, indicate that there was an error.
Outputs:	System Displays List
Pre-Conditions	Users' data already exists in UserHolder; NLPAlgorithm already has a model to generate the desired data.
Post-Conditions:	NLPAlgorithm takes in any feedback from User and uses it enhanced its model.

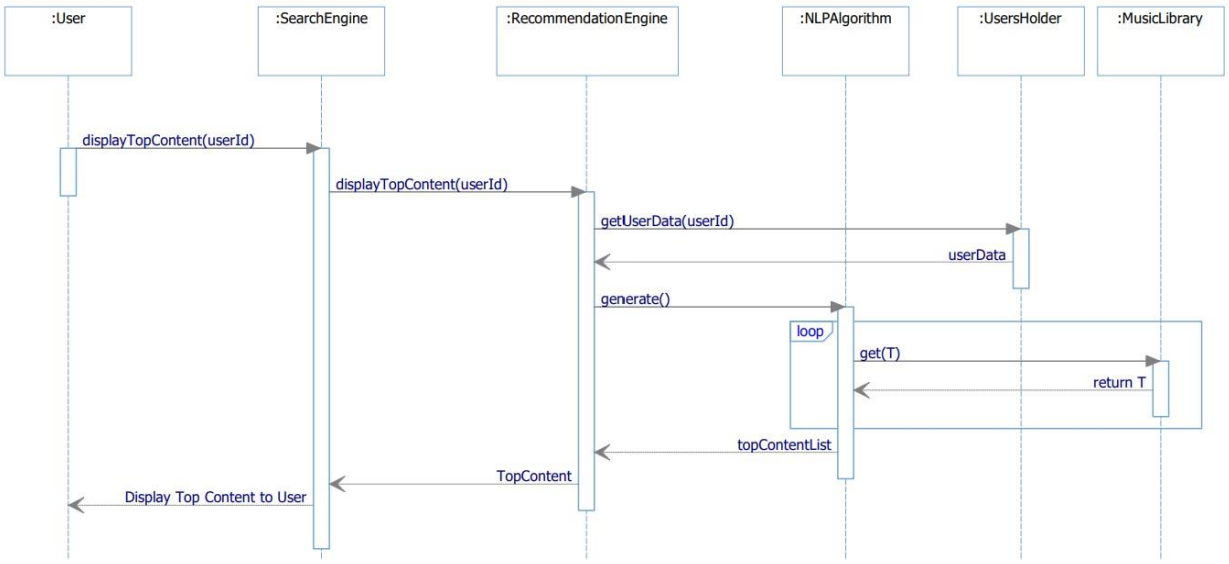

Figure 6. UC2 SOC Interaction Diagram.

SOC 3

Table 4 breaks down a System Operational Contract for Use Case 3 of the system. The operation, responsibilities, outputs, and any pre- and post-conditions are defined. Figure 7 shows the interaction between the conceptual classes defined in the Domain Model.

Table 4. UC3 SOC.

Name Of Operation:	**displayNewArtists(userId:int)**
Responsibilities:	Streaming Service displays new artists List to User using the Machine Learning Algorithms
Type:	System
Cross Reference:	Use Case 3
Exception:	If database is not available, indicate that there was an error.
Outputs:	System Displays List
Pre-Conditions	Users' data already exists in UserHolder; NLPAlgorithm already has a model to generate the desired data.
Post-Conditions:	NLPAlgorithm takes in any feedback from User and uses it enhanced its model.

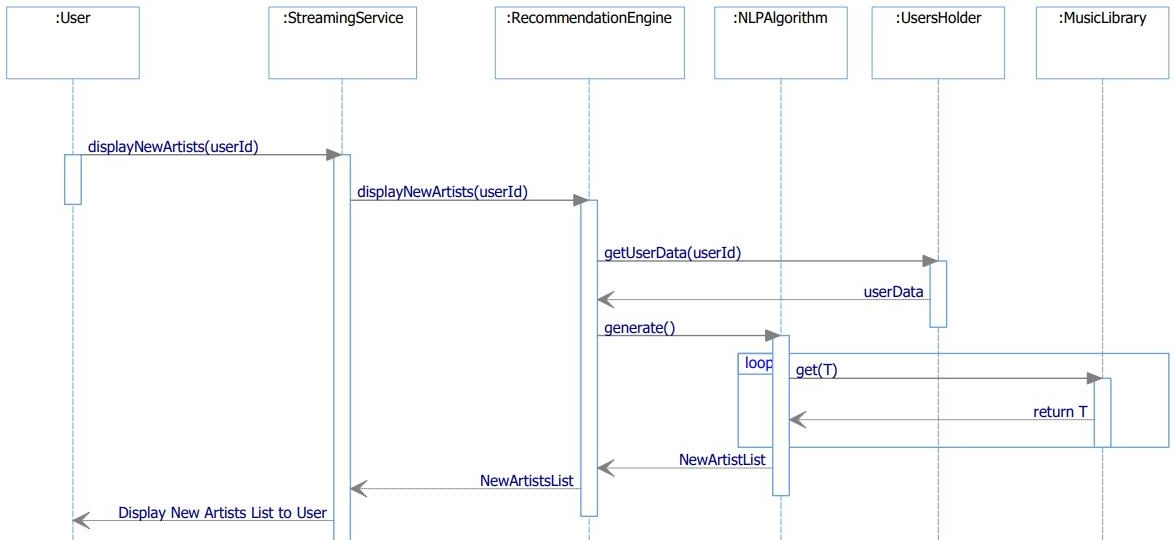

Figure 7. UC3 SOC Interaction Diagram.

SOC 4

Table 5 breaks down a System Operational Contract for Use Case 4 of the system. The operation, responsibilities, outputs, and any pre- and post-conditions are defined. Figure 8 shows the interaction between the conceptual classes defined in the Domain Model

Table 5. UC4 SOC.

Name Of Operation:	**showSimilarPlaylist(userId:int)**
Responsibilities:	Streaming Service displays similar playlists from other users to User using the Machine Learning
Type:	System
Cross Reference:	Use Case 4
Exception:	If database is not available, indicate that there was an error.
Outputs:	System Displays List
Pre-Conditions	Users' data already exists in UserHolder; NLPAlgorithm already has a model to generate the desired data.
Post-Conditions:	NLPAlgorithm takes in any feedback from User and uses it enhanced its model.

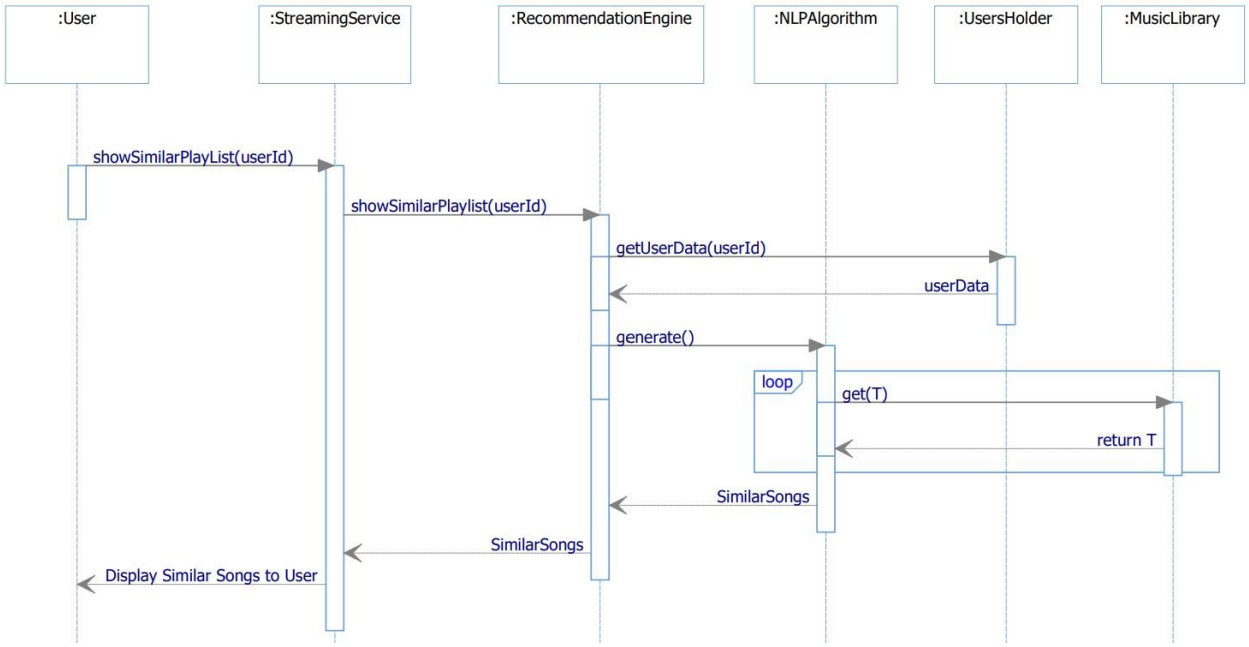

Figure 8. UC4 SOC Interaction Diagram.

SOC 5

Table 6 breaks down a System Operational Contract for Use Case 5 of the system. The operation, responsibilities, outputs, and any pre- and post-conditions are defined. Figure 9 shows the interaction between the conceptual classes defined in the Domain Model

Table 6. UC5 SOC

Name Of Operation:	**searchSongs(userId:int, type:String, style:String, genre:String, date:Date)**
Responsibilities:	Search Service displays Songs to User using the Machine Learning Algorithms.
Type:	System
Cross Reference:	Use Case 5
Exception:	If database is not available, indicate that there was an error.
Outputs:	System Displays List
Pre-Conditions	Users' data already exists in UserHolder; NLPAlgorithm already has a model to generate the desired data.
Post-Conditions:	NLPAlgorithm takes in any feedback from User and uses it enhanced its model.

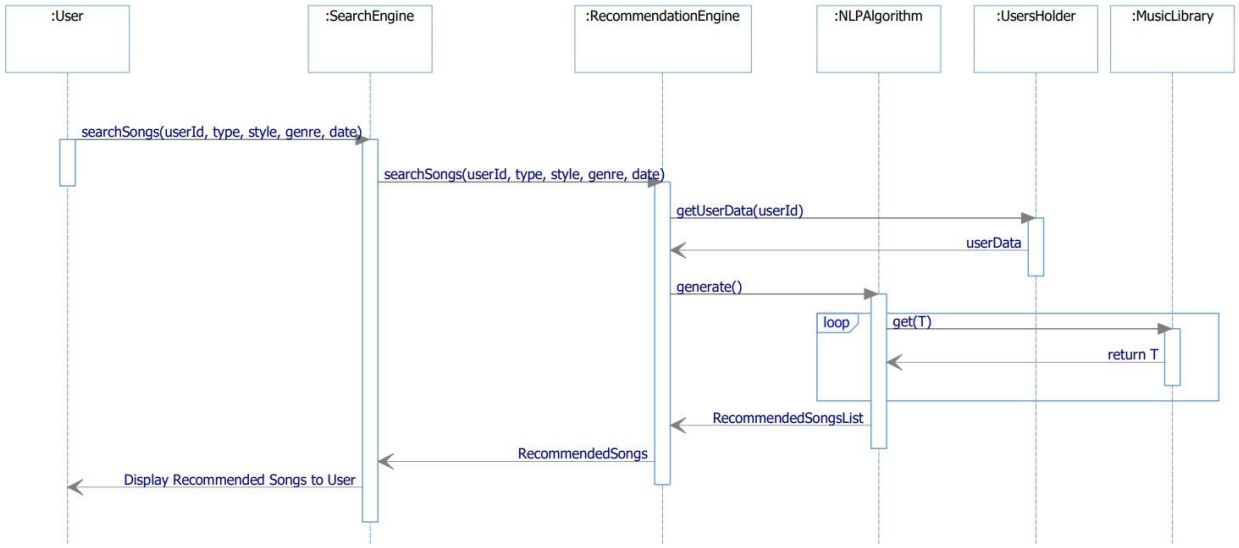

Figure 9. UC5 SOC Interaction Diagram.

UML State Diagram

Figure 10 illustrates the possible states and transitions an object can undergo throughout its lifecycle within the system.

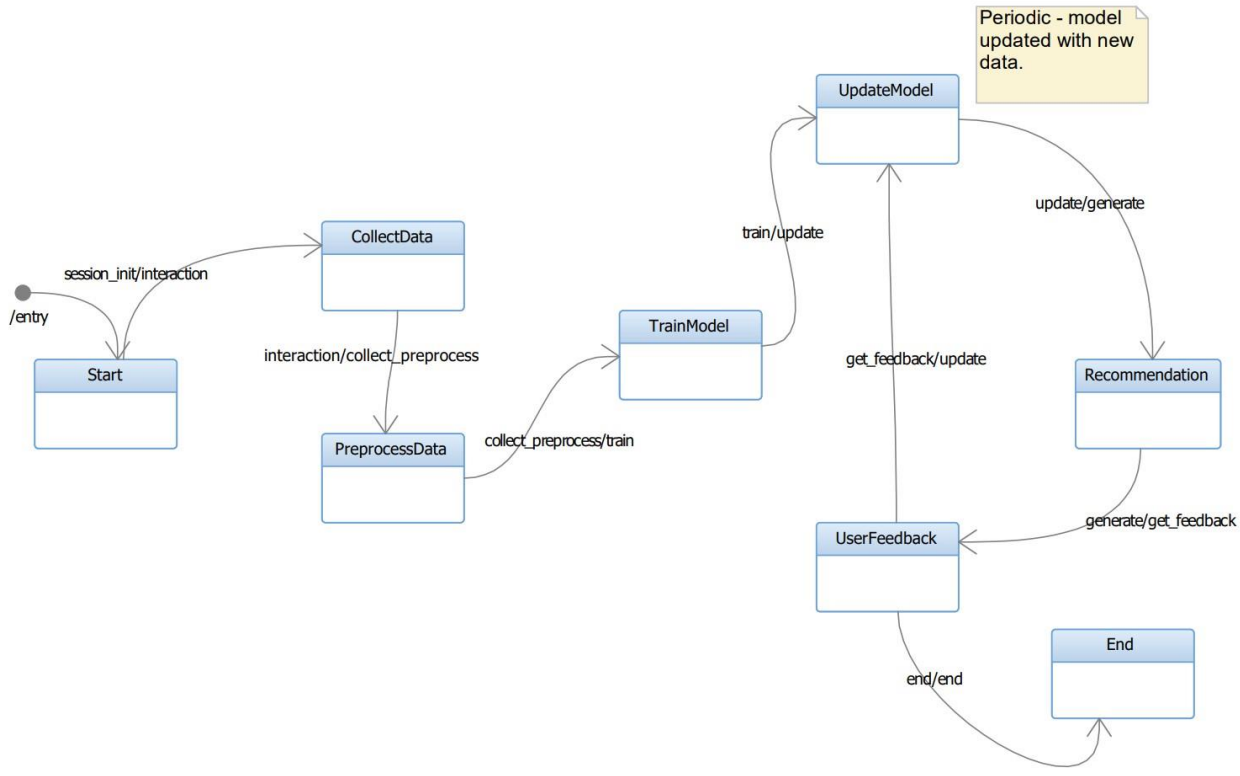

Figure 10. State Diagram.

The entry point (Start) begins with a user interaction. It transitions from the initial state to then servicing the interface the user accesses. The system collects and preprocesses the user data to

feed to the training model for any updates. The system will generate the recommendation and present it to the user. If the user has any feedback, the system will feed that feedback to the model and it will update the model accordingly. The 'UpdateModel' component will periodically update the model.

UML Activity Diagram (Swimlane Diagram)

Figure 11 visually represent the activities and flows within the recommendation engine system, highlighting the responsibility of different actors using swim lanes.

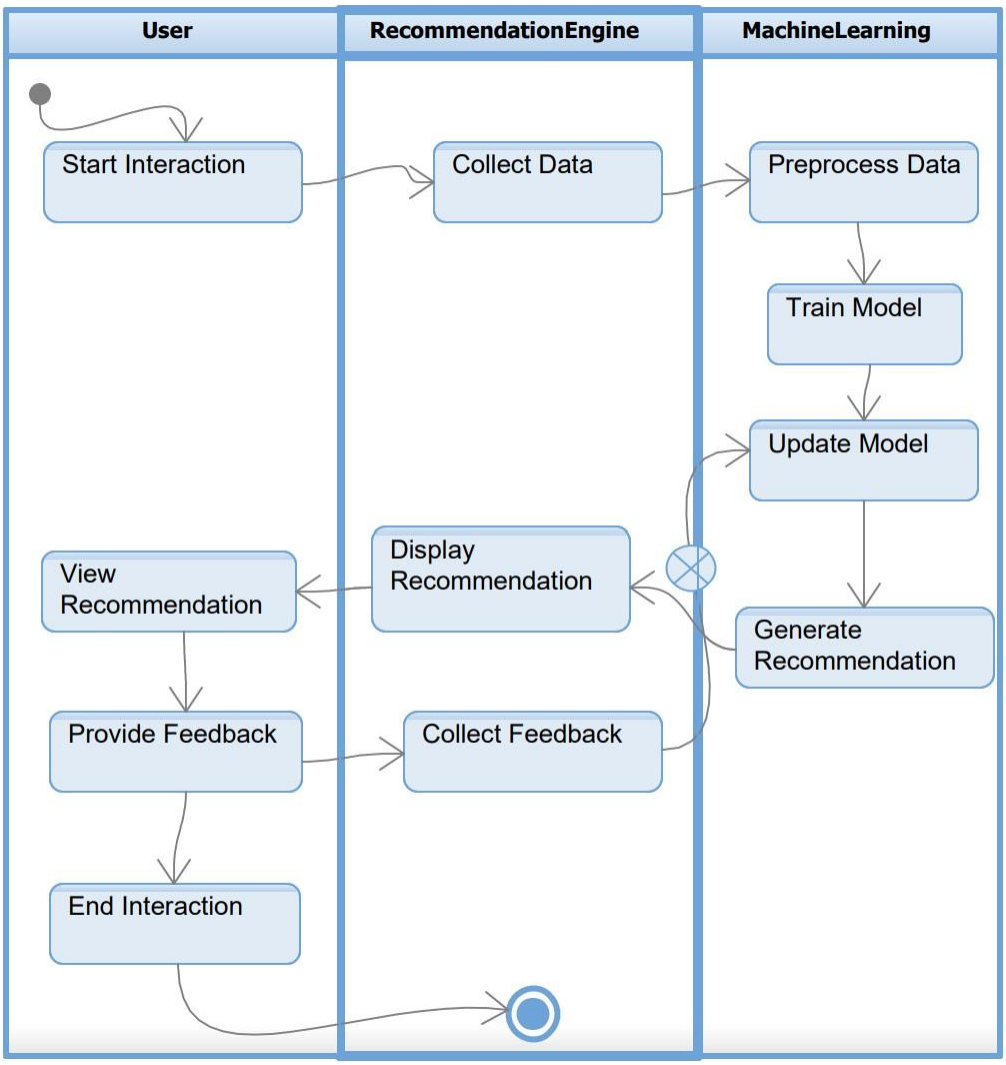

Figure 11. Activate (Swimlane) Diagram.

For the given diagram we have actors: User, Recommendation Engine (RE), and MachineLearning model (ML). The User is the actor that interacts with the system. The RE is what serves as the backend and frontend server for the system. It processes user interaction and interacts with ML model. RE collects the data for the ML model which will train and update the model to generate the recommendation. The RE will display the results to the User who will

provide feedback and end the interaction. If User offers feedback, the feedback will be collected by RE, and analyzed by the ML model.

UML Component Diagram

Figure 12 depicts the system's physical components and their dependencies, providing a high-level architectural view.

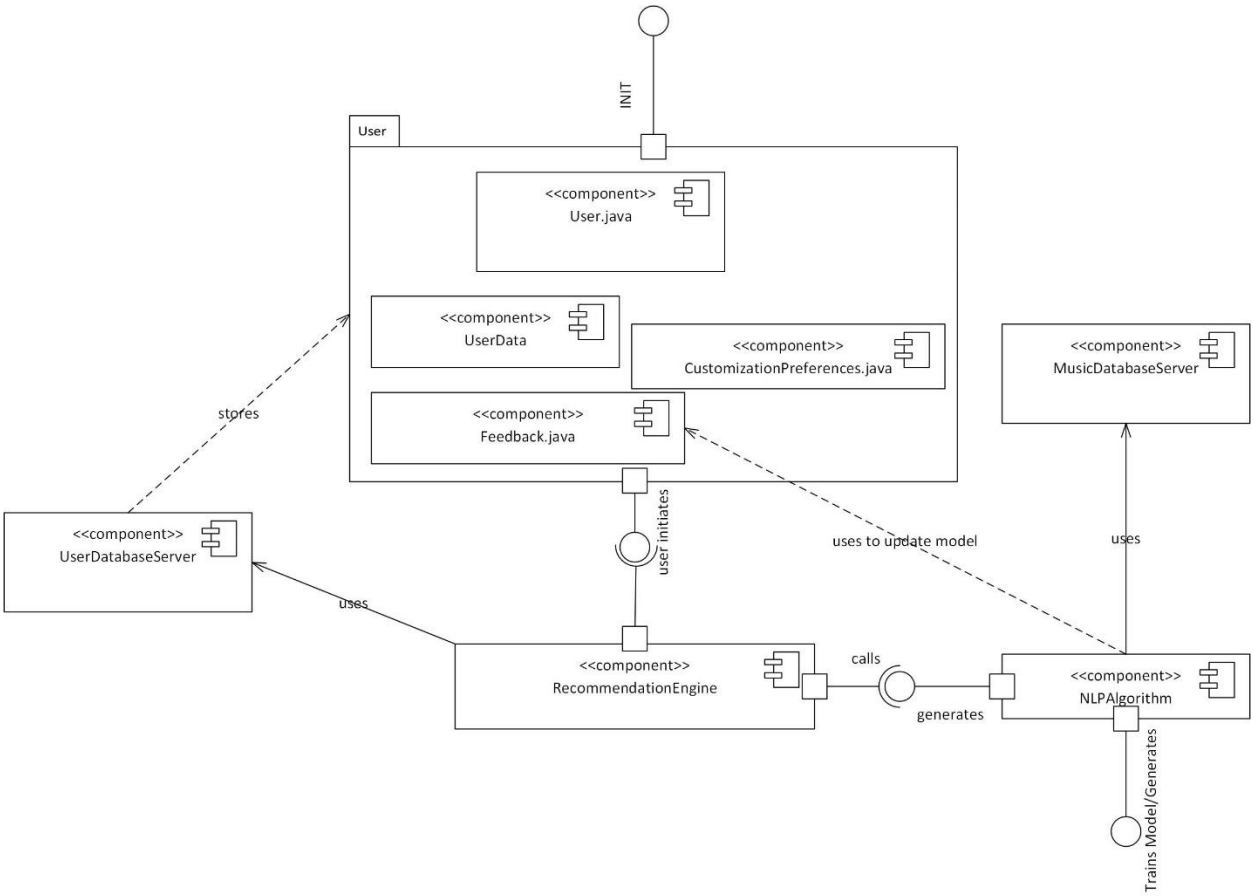

Figure 12. Component Diagram.

From the diagram, the important components are User package which contains all user specific information, the Recommendation Engine component that handles all frontend and backend processing; the User DB and Music DB that stores the user and music information the Algorithm uses respectively; and the NLPAlgorithm component which is the core of the generating the music recommendation by training and updating its model. The User interacts with the RecommendationEngine component which uses UserDatabase to collect the data. The RecommendationEngine calls the NLPAgorithm component that the generates the personalized data. NLPAlgorithm uses MusicDatabase to create the data. Also, NLPAlgorithm uses Feedback component to update its model. Finally, the system is initiated with INIT which is connected to the User demonstrating that a user interaction is what kickstarts the system.

Cloud Deployment Diagram

Figure 13 illustrates the AWS cloud platform and how the system's components will be deployed within it.

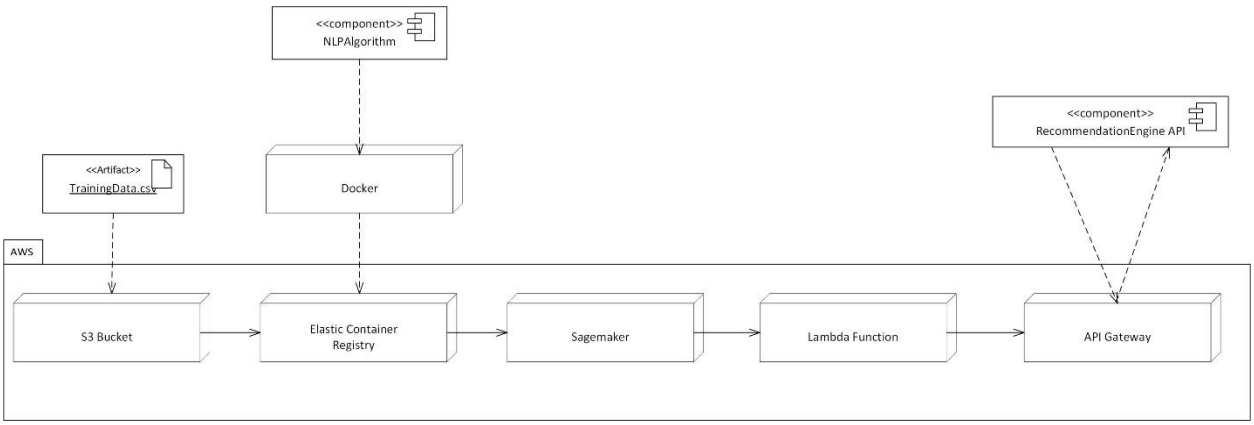

Figure 13. Deployment Diagram.

For this system, an ML pipeline is created using AWS tools. The service will be deployed through an API that can receive HTTP requests and use the machine learning model to make predictions and generate personalized music streaming data for user. Below are the tools:

- The NLPAlgorithm will use python libraries such as Pandas for the ML algorithm.
- Docker will be used for containerizing the algorithm.
- Amazon S3 for storing datasets and the trained model.
- Amazon ECR for hosting custom algorithm in the Docker container.
- Amazon Sagemaker for training models and making predictions using the Docker Container.
- AWS Lambda Function for invoking Sagemaker Endpoint to deploy the model, and enriching response.
- API Gateway for pushing the public API service to the Internet.

Skeleton Classes and Tables Definition

Provide basic outlines of the main classes involved, including their attributes and methods.

Please reference Section UML Class Diagram and view Figure 4 for basic outlines of the main classes involved and their attributes/methods.

Define the structure of any database tables required to store system data.

1. users_t
    - userId (primary key)
    - username
    - email
    - password

- preferences (JSON)
- listening_history (stored as relational data – association – with separate table)
- feedback (stored as relational data – association - with separate table)

2. songs_t
    - songId (primary key)
    - title
    - artist
    - album
    - genre
    - duration
    - release_date
    - lyrics
3. feedback_t
    - id (primary key)
    - userId (foreign key referencing users_t)
    - songId (foreign key registered songs_t)
    - rating
    - comment
    - timestamp
4. listerning_history_t (many-to-many relationship between users and songs)
    - historyId (primary key)
    - userId (foreign key referencing users_t)
    - songId (foreign key referencing songs_t)
    - playcount
    - last_timestamp

Design Patterns

This section explains any design patterns and best practices implemented in the design, justifying their use for specific scenarios.

Architectures implemented in this design are cloud-native applications and monolithic system. Cloud-native applications was chosen for this system because based on research of existing streaming services (Netflix for example) the benefits for speed, safety and scalability is what is best for recommendation engine. Most importantly, cloud applications allow for containerization and its orchestration. Containerization packs the application and dependencies into a container and can be run consistently in different computing environments because they are portable.

KISS (Keep-It-Simple-Stupid) design pattern and monolithic architecture was chosen because the recommendation engine has a simple architecture, and only one service will be needed to be deployed. A microservice architecture does not fit this system. Although scalability is reduced in a monolithic architecture, cloud-native application combats that. The use cases all follow same flow : user requests action, engine collects data, feeds data to algorithm, algorithm generates recommendation, engine presents to user. KISS pattern was chosen to keep the simplicity of the design and reduce any unnecessary complexities which is why a monolithic approach was chosen, both go hand in hand in terms of decision.

KISS pattern gives the system maintainability, scalability and clarity. This pattern approach allows the recommendation engine to be designed to only focus on core functionalities and provide a clean recommendation that prioritizes the users preference and feedback. There’s no need for over-complicated features in this system.