Northwestern | SCHOOL OF PROFESSIONAL STUDIES

# PERSONALIZED MUSIC RECOMMENDATION SYSTEM

Project and Testing Plan

MSDSP 498 - Capstone

Abhigna Mallepally, Anishka Agarwal, Edwin Daniels, Sachin Sharma

January 19, 2025

# Project Plan

## Project Overview

The Personalized Music Recommendation System aims to enhance user experience by offering tailored music suggestions using Spotify data. Combining collaborative filtering, content-based filtering, matrix factorization, and neural network techniques, the system delivers personalized recommendations and addresses cold-start issues for new users. Key features include playlist generation, sentiment-based filtering, and AI-driven conversational recommendations.

## Project Objectives

1. **Personalized Song Recommendations:** Provide tailored song suggestions based on users' last 50 listened songs.
2. **Cold-Start Recommendations:** Recommend songs for new users without historical interaction data by leveraging genre and artist preferences.
3. **Playlist Generation:** Generate playlists of 50 songs based on user's preferred genres or artists.
4. **Single Song Recommendations:** Suggest the top 5 similar songs based on a single song input.
5. **Sentiment Filter:** Incorporate sentiment analysis to refine recommendations.
6. **User Engagement Features:** Add functionalities like like/dislike and add-to-playlist options.
7. **Trending Now Feature:** Leverage Spotify's trending music data for recommendations.
8. **AI Search/ChatBot:** Implement interactive search or chatbot features for enhanced user interaction.
9. **Geography-based Recommendations:** Tailor recommendations based on locale, language, geography, and time (optional).

## Phases of Implementation

### Phase 1: Research and Planning

- **Tasks**:
    - Collect user interaction data using Spotify REST API.

  - Source and preprocess the Spotify 1 Million Songs dataset from Kaggle.
  - Conduct feasibility analysis for selected ML/DL algorithms.
- **Deliverables**:
  - Dataset ready for training and testing.
  - Requirements document and technical feasibility report.

**Phase 2: Development**

- **Tasks**:
  - Implement and evaluate models: KNN, Matrix Factorization, Neural Collaborative Filtering, and LightFM.
  - Develop the recommendation engine backend.
  - Build the frontend web application interface.
- **Deliverables**:
  - Functional backend with tested recommendation algorithms.
  - User interface with core features.

**Phase 3: Deployment**

- **Tasks**:
  - Deploy the application on a cloud platform (e.g., AWS or Azure).
  - Set up APIs for integration with the Spotify service.
- **Deliverables**:
  - Fully operational web application.
  - Scalable and reliable cloud infrastructure.

**Phase 4: Evaluation and Refinement**

- **Tasks**:
  - Conduct user acceptance testing (UAT) and gather feedback.
  - Optimize models and application performance based on test results.
- **Deliverables**:
  - Optimized and validated recommendation system.
  - Final project report with evaluation metrics.

# Testing Plan

## Testing Approach

The testing process will validate the system's accuracy, scalability, and user experience. Tests are divided into the following categories:

**1. Unit Testing**

- **Objective**: Validate individual components for correctness.
- **Scope**:
    - Test data ingestion modules to ensure they correctly process Spotify data.
    - Validate the implementation of algorithms (e.g., KNN, LightFM) with test datasets.
    - Confirm model output consistency under various input conditions.
- **Tools**: Pytest, Unittest (Python).
- **Example Test Case**:
    - **Test Input**: A sample dataset with 10 songs and known interactions.
    - **Expected Output**: Correct user-item similarity scores.

**2. Functional Testing**

- **Objective**: Validate individual components and features for correctness.
- **Scope**:
    - Test the accuracy of recommendations for various user profiles.
    - Validate the cold-start solution for new users.
    - Confirm the behavior of the sentiment filter and AI chatbot.
    - Test features like "Trending Now" and engagement functionalities (likes, dislikes, add-to-playlist).
- **Tools**: Pytest, Unittest.
- **Example Test Case**:
    - **Test Input**: User with 50 song interaction records.
    - **Expected Output**: Personalized recommendations based on the last 50 songs.
    - **Test Input**: New user selects a preferred genre (e.g., "Pop").

- **Expected Output**: Recommendations for songs within the selected genre.
- **Test Input**: User selects a “happy” sentiment filter.
- **Expected Output**: Recommended songs with high valence scores.
- **Test Input**: User interacts with the “Trending Now” feature.
- **Expected Output**: Display of trending songs dynamically updated based on Spotify data.

**3. Integration Testing**

- **Objective**: Ensure seamless interaction between components.
- **Scope**:
  - Verify that the backend communicates correctly with the frontend.
  - Test the integration of Spotify API data into the system.
- **Tools**: Postman, Selenium.
- **Example Test Case**:
  - **Test Input**: User searches for a song.
  - **Expected Output**: Backend returns accurate recommendations to the frontend.

**4. Usability Testing**

- **Objective**: Ensure the system is user-friendly and intuitive.
- **Scope**:
  - Gather user feedback on the interface and features.
  - Test the chatbot’s conversational flow for natural and engaging interactions.
  - Validate the ease of navigation for generating playlists or accessing recommendations.
- **Tools**: UsabilityHub, Surveys.
- **Example Test Case**:
  - **Test Input**: Users are asked to generate a playlist by selecting their favorite genre and interact with the chatbot to discover songs.
  - **Expected Output**: Users successfully create playlists without confusion, find the interface easy to navigate, and rate the chatbot experience as intuitive.

**5. System Testing**

- **Objective**: Validate the complete system against requirements.
- **Scope**:
    - Assess performance, scalability, and functionality.
    - Validate workflows like playlist generation and single-song recommendations.
- **Tools**: JMeter, Locust (for load testing).
- **Example Test Case**:
    - **Test Input**: 1,000 concurrent users accessing the application.
    - **Expected Output**: Stable performance with response times under 2 seconds.

**6. Performance Testing**

- **Objective**: Assess the system's ability to handle load and scale efficiently.
- **Scope**:
    - Load Testing: Simulate high user traffic to test system performance.
    - Stress Testing: Push the system to its limits to identify breaking points.
    - Latency Testing: Measure response times for recommendation generation.
- **Tools**: JMeter, Locust.
- **Example Test Case**:
    - **Test Input**: Simulate 5,000 concurrent users generating playlists and accessing recommendations.
    - **Expected Output**: Response times remain under 2 seconds, and the system handles the load without crashing or significant delays.

**7. Security Testing**

- **Objective**: Identify and resolve security vulnerabilities.
- **Scope**:
    - Validate secure handling of user data and API calls.
    - Test for vulnerabilities like SQL injection and unauthorized access.
- **Tools**: OWASP ZAP, Browser Console.
- **Example Test Case**:

- **Test Input**: Attempt to access user data by simulating SQL injection attacks on the login page.
- **Expected Output**: The system rejects malicious queries, protects user data, and logs the attempt for security analysis.

**8. User Acceptance Testing (UAT)**

- **Objective**: Ensure the system meets user needs and expectations.
- **Scope**:
  - Deploy the system to a small group of beta testers for real-world evaluation.
  - Collect feedback on recommendation accuracy, usability, and satisfaction.
- **Tools**: Surveys, Usability testing tools.
- **Example Test Case**:
  - **Test Input**: Users interact with the chatbot to discover music.
  - **Expected Output**: Positive feedback on usability and relevance of recommendations.

## Test Scenarios and Cases

| Scenario | Description | Test Case | Input | Expected Output |
|---|---|---|---|---|
| Personalized Song Recommendations | Validate recommendations based on the last 50 songs listened to. | Recommendations tailored to user's recent listening history. | User with 50 interaction records. | Songs matching user preferences. |
| Search Song/Artist | Validate song and artist search is working. | Find song/artist based on user input. | Input contains Song/Artist name. | Find Song/Artist track details. |
| Cold-Start Recommendations | Provide recommendations for new users. | Genre-based recommendations for new users. | New user with no interaction data. | Recommendations based on preferred genres. |
| Find Similar Song | Provide similar recommendation to user based on single song selection. | Top 5 Similar song recommendation. | Song Name. | Top 5 Similar Songs using KNN. |

| | | | | |
|---|---|---|---|---|
| Sentiment-Based Filtering | Filter recommendations based on mood. | Tracks filtered by selected mood (e.g., “happy”). | User selects “happy” mood. | Songs with positive sentiment. |
| Scalability | Assess performance under high load. | Application’s stability under concurrent users. | 5,000 concurrent users. | Stable performance without crashes. |
| Trending Now Feature | Display trending songs dynamically. | Generate and display trending song lists. | Real-time Spotify data. | Updated trending songs list. |
| AI Chat Bot | Songs results based on the user query | Search the database and recommend songs/generate playlists. | User query in text format | Relevant search results. |

## Evaluation and Metrics

### Accuracy Metrics

- **Precision**: Measures the proportion of recommended songs that are relevant.
- **Recall**: Assesses the proportion of relevant songs successfully recommended.
- **F1-Score**: Balances precision and recall to provide an overall accuracy measure.

### Performance Metrics

- **Latency**: Measure response times under different loads.
- **Throughput**: Tracks the number of recommendations generated per second.

### User Experience Metrics

- **User Satisfaction**: Collected via surveys and feedback forms.
- **Engagement Rate**: Tracks user interactions with recommendations (e.g., clicks, likes, playlist additions).

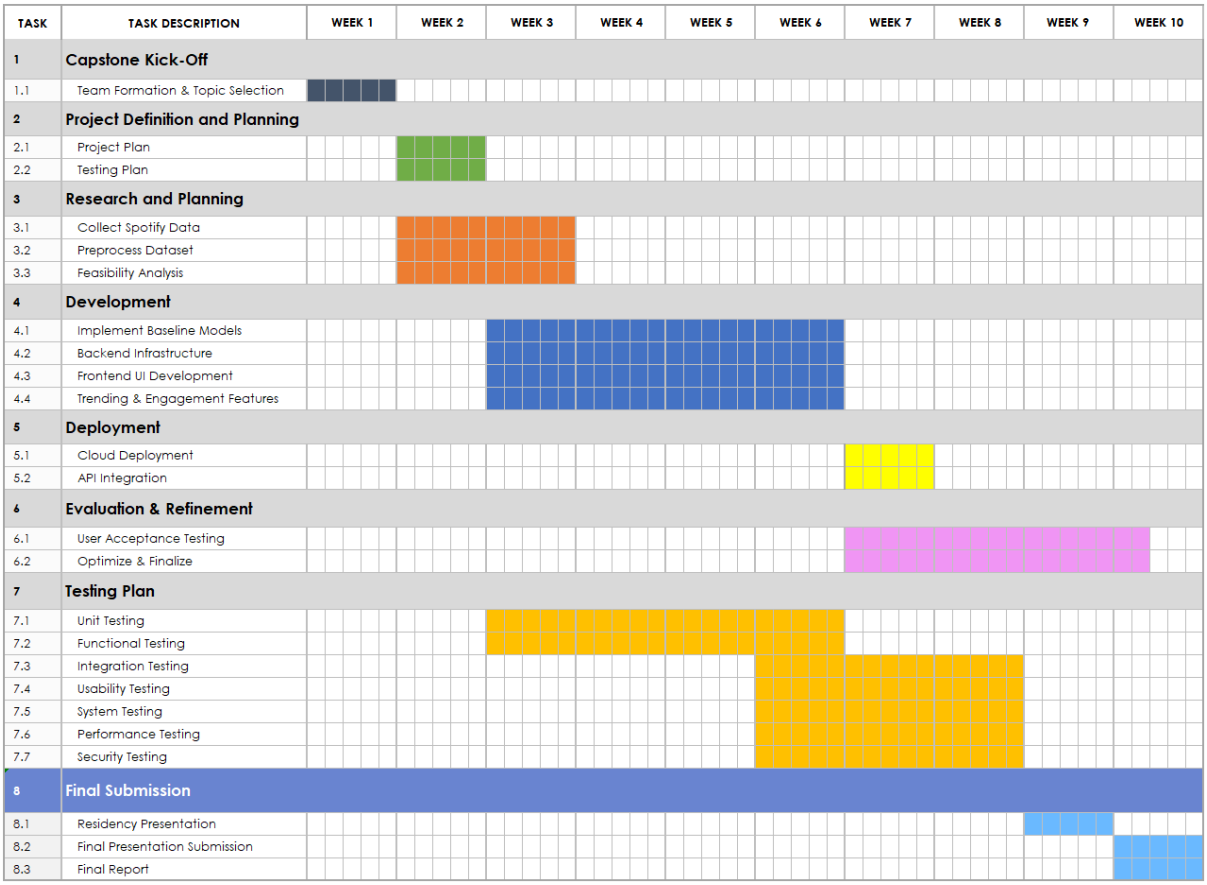
### Scalability Metrics

- **Load Handling**: Assesses system performance with increasing concurrent users.
- **Resource Utilization**: Monitors CPU, memory, and network usage.

## Test Reports

- Daily status reports during testing phases.
- Test execution summary with inputs, outputs, and issue logs.

## Project and Testing Timelines




# Sample Data Schema

### User Interaction Data

| Column | Type | Description |
|---|---|---|
| UserID | String | Unique identifier for the user. |
| SongID | String | Unique identifier for the song. |
| Timestamp | Timestamp | Time of interaction. |
| Song Audio Features | String | Audio Features. |

### Song Metadata

| Column | Type | Description |
|---|---|---|
| SongID | String | Unique identifier for the song. |
| Title | String | Title of the song. |
| Artist | String | Artist of the song. |
| Genre | String | Genre of the song. |

| | | |
|---|---|---|
| Sentiment | String | Sentiment label (happy, sad, etc.). |
| Popularity | Integer | Track popularity (0 to 100). |
| Year | Integer | Year released (2000 to 2023). |
| Danceability | Float | Track suitability for dancing (0.0 to 1.0). |
| Energy | Float | Intensity and activity (0.0 to 1.0). |
| Key | Integer | Musical key (-1 to -11). |
| Loudness | Float | Loudness in decibels (-60 to 0 dB). |
| Mode | Integer | Major ('1') or Minor ('0'). |
| Speechiness | Float | Presence of spoken words (0.0 to 1.0). |
| Acousticness | Float | Acoustic confidence (0.0 to 1.0). |
| Instrumentalness | Float | Instrumental content (0.0 to 1.0). |
| Liveness | Float | Audience presence (0.0 to 1.0). |
| Valence | Float | Musical positiveness (0.0 to 1.0). |
| Tempo | Float | Tempo in beats per minute (BPM). |
| Time_signature | Integer | Estimated time signature (3 to 7). |
| Duration_ms | Integer | Duration of track in milliseconds. |

## Tools and Frameworks

### Development Tools

- **Python Libraries**: Pandas, NumPy, Scikit-learn, TensorFlow, LightFM.
- **Web Framework**: Flask/Django for backend, React/Angular for frontend.

### Testing Tools

- **Unit Testing**: Pytest, Unittest.
- **Integration Testing**: Postman, Selenium.
- **Performance Testing**: JMeter, Locust.
- **Usability Testing**: UsabilityHub, Surveys.
- **Security Testing**: OWASP ZAP, Burp Suite.

By following this detailed project and testing plan, we aim to deliver a reliable, scalable, and user-friendly recommendation system that redefines music discovery.