

Project Management Practices:

1.	Modularity and Organization:	<ul style="list-style-type: none">• The code is organized into functions and endpoints, promoting modularity.• Each endpoint seems to handle a specific feature or functionality, contributing to a clear project structure.
2.	Error Handling:	<ul style="list-style-type: none">• Proper error handling is implemented using try-except blocks in critical sections of the code.• The application returns appropriate HTTP status codes and error messages to the client.
3.	Documentation:	<ul style="list-style-type: none">• Inline comments are used to explain complex sections of code or provide context.• Descriptive function and variable names contribute to code readability.
4.	Session Management:	<ul style="list-style-type: none">• The use of the <code>session</code> object for storing the current user's information indicates a stateful approach, which is essential for maintaining user sessions.
5.	Database Abstraction:	<ul style="list-style-type: none">• The code abstracts database operations, making use of functions to connect to the MongoDB database, interact with collections, and perform queries.

Codebase Management Practices:

1.	Version Control:	<ul style="list-style-type: none">• using a version control system (e.g., Git) is crucial for codebase management. Commits and branches help track changes and collaborate effectively.
2.	Code Readability:	<ul style="list-style-type: none">• The code follows PEP 8 conventions for Python, contributing to code consistency and readability.• Proper indentation and spacing enhance code readability.
3.	Separation of Concerns:	<ul style="list-style-type: none">• Functionality is separated into different routes/endpoints, and the logic is distributed across multiple functions, promoting the separation of concerns.
4.	Environment Configuration:	

	<ul style="list-style-type: none"> The application have a configuration for running on a specific host and port, which is essential for deployment flexibility.
5.	Security Considerations: <ul style="list-style-type: none"> While the codebase does handle user authentication,we made sure to follow best practices for securing sensitive information, such as secret keys and user credentials.
6.	Testing: <ul style="list-style-type: none"> Unit test here.
7.	Dependency Management: <ul style="list-style-type: none"> Flask==2.0.1 certifi==2021.5.30 requests==2.26.0 pymongo==3.12.0 bson==0.5.10

REQUIREMENTS SPESIFICATONS:

1.	User Stories in Code Comments: <ul style="list-style-type: none"> The code includes comments that describe the purpose and functionality of certain blocks of code. This is a good practice to provide context and explanations for future developers.
2.	Descriptive Endpoint Names: <ul style="list-style-type: none"> Endpoint names such as <code>/register</code>, <code>/login</code>, <code>/recommend_song</code> indicate the functionality they serve. Descriptive endpoint names help in understanding the purpose of each API.
3.	Consistent Variable Naming: <ul style="list-style-type: none"> Variable names are relatively consistent and descriptive. For example, <code>username</code>, <code>user_document</code>, <code>liked_songs</code>. This contributes to code readability and understanding.
4.	Error Handling: <ul style="list-style-type: none"> The code includes error handling for potential exceptions, providing informative error messages. This is crucial for debugging and understanding the cause of issues.
5.	Input Validation: <ul style="list-style-type: none"> There are checks for missing parameters in query parameters and form data. For example, in <code>/get_higher Rated Genre</code> and <code>/get_users_liked_songs</code>, the code checks if the <code>username</code> parameter is present.
6.	Usage of Session for User Authentication: <ul style="list-style-type: none"> Session usage (<code>session.get('username')</code>) indicates an attempt to manage user authentication and maintain user state across requests.
7.	RESTful Endpoint Design: <ul style="list-style-type: none"> The endpoints are designed following RESTful principles, with HTTP methods (<code>GET</code>, <code>POST</code>) corresponding to the intended actions.
8.	Separation of Concerns:

	<ul style="list-style-type: none"> The code separates concerns by having distinct endpoints for different functionalities, such as authentication, recommendation, search, and user-related activities.
9.	Structured JSON Responses: <ul style="list-style-type: none"> The API responses are structured in JSON format, making it easy for clients to parse and interpret the data.
10.	Documentation in Comments: <ul style="list-style-type: none"> There are comments that explain the purpose of specific code blocks, which can serve as a form of documentation for developers who review or maintain the code.

Analysis/Design Practices:

1.	Separation of Concerns: <ul style="list-style-type: none"> The code is organized into different functions and endpoints, indicating a separation of concerns. There are separate functions/endpoints for user authentication, song analysis, recommendations, friendship activities, song management, search functionality, and more.
2.	Modularization: <ul style="list-style-type: none"> The code is modular, with different functionalities organized into separate routes and functions. Modularization promotes code readability, maintainability, and reusability.
3.	Error Handling: <ul style="list-style-type: none"> Error handling is implemented using try-except blocks in each route to catch and handle exceptions. Appropriate error messages are returned in case of exceptions, providing meaningful feedback.
4.	RESTful Design: <ul style="list-style-type: none"> The API endpoints follow RESTful principles, using HTTP methods (GET, POST, DELETE) for different actions. Meaningful and consistent endpoint names are used for each resource or action.
5.	Data Validation: <ul style="list-style-type: none"> Input validation is performed in several places to ensure that required parameters are present and have the correct format. For example, checking if the username is present, validating file uploads, and ensuring the existence of required fields in forms.
6.	Database Interaction: <ul style="list-style-type: none"> MongoDB is used as the database, and interactions with the database are handled using the PyMongo library.

- CRUD operations (Create, Read, Update, Delete) are performed on the database, such as adding tracks, deleting tracks, and retrieving user data.

Technologies/Frameworks:

1. **Flask:**

- Flask is used as the web framework for building the backend of the application.
- Flask is known for its simplicity and ease of use in building web applications and APIs.

2. **MongoDB:**

- MongoDB is the chosen NoSQL database for storing user information, tracks, and other data.
- PyMongo is used as the Python driver to interact with MongoDB from the Flask application.

3. **Python:**

- The backend is implemented in Python, taking advantage of its versatility and ease of integration with web frameworks and databases.

4. **RESTful API:**

- The backend follows the principles of RESTful design, providing a set of stateless and uniform endpoints for interacting with the application.