**Contents Page**

**Introduction**

Facing a difficulty in discovering new artists that align with the preferences is a common concern among music listeners. Many people I know have experienced music fatigue with the artists they were listening to. This problem can prevent one’s ability to fully enjoy music.

\*NAME\* application software has been developed for people who love music and are interested in discovering more artists similar to the ones they listen to in order to improve user experience. My website is suitable for people of all ages and music tastes, it is user-friendly and interactive.

Users will create accounts on the website in order to store the progress. Website’s interactivity includes a questionnaire with genres and artists. Based on the answers from the questionnaire, the algorithm creates a playlist with music that the user will like. Users will be able to listen to a short piece of music, if they like it, there will be a redirection to Apple Music or Spotify music broadcasting platforms for users to log in and add the song to their playlists or explore the artist further. Based on the amount of songs the user marks as “favourite”, the algorithm will display more similar music.

For the ones that are very keen on music’s history, the website communicates with \*website name\* and will display selected songs’ history and information about artists.

SOURCES TO READ WHICH MAY HELP WITH DEVELOPMENT:

AI playlist generator <https://playlistable.io>

<https://www.chosic.com/playlist-generator/>

<https://www.google.com.ua/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjuhIPEm5iBAxWZUUEAHfGmApw4ChC3AnoECA4QAg&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D3vvvjdmBoyc&usg=AOvVaw0UWarQj-U2pWX6OhTDV98O&opi=89978449>

<https://python.plainenglish.io/building-a-cli-spotify-playlist-generator-using-python-spotipy-3b32b63a25da>

FIREBASE <https://extensions.dev>

AppleMusic Developer for APIs

<https://developer.apple.com/documentation/applemusicapi/get_a_library_song_s_relationship_directly_by_name>

<https://www.appcoda.com/musickit-music-api/>

<https://leemartin.dev/creating-an-apple-music-api-token-e0e5067e4281>

<https://www.google.com.ua/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwiPyIXu6LX_AhXyhv0HHQJjDocQFnoECAcQAQ&url=https%3A%2F%2Fwww.benfrederickson.com%2Fmatrix-factorization%2F&usg=AOvVaw04aZh6WS1cu3sobhnvyf5U>

**3.1. Analysis of the problem (10 marks)**

* 3.1.1 Problem identification
  1. *Describe and justify the features that make the problem solvable by computational methods.*
  2. *Explain why the problem is amenable to a computational approach.*

When developing a playlist-generating website, numerous challenges must be considered. One of the most significant obstacles is efficiently handling and processing vast quantities of music data, including genres, artists, albums, and user preferences. However, computational methods offer a robust solution to this issue. By utilising advanced algorithms, analysing this data and generating personalised playlists for each user based on their unique preferences is possible.

To accomplish this, it is essential to comprehend the complex relationships and patterns between music genres, artists, and user preferences. This is where machine learning algorithms come into play. These algorithms can accurately predict and recommend personalised playlists by extracting patterns from data and learning from user behaviour. They can identify connections and dependencies between music attributes and user preferences, leading to highly customised recommendations.

Moreover, a computational approach enables iterative learning and improvement. The playlist-generating algorithm can continuously refine and optimise its recommendations by collecting user feedback and preferences. This means the system can adapt and adjust based on this feedback, improving playlist recommendations.

Scalability is also a crucial factor. Once the algorithm is developed and optimised, it can generate personalised playlists for a large number of users simultaneously, delivering recommendations in real time. This scalability allows us to cater to a growing user base and provide users with a seamless and responsive experience.

Finally, computational methods facilitate the integration of the playlist-generating website with external music platforms, such as Apple Music or Spotify. By leveraging APIs provided by these platforms, the website can access additional music data and seamlessly redirect users to their preferred music platform for continuous listening. This integration allows for the handling of data exchanges and the integration of different systems.

In summary, a computational approach offers a powerful solution to the challenges of creating a playlist-generating website. We can provide personalised and accurate playlist recommendations by processing large amounts of data, identifying complex relationships and patterns, learning iteratively, scaling efficiently, and integrating with external platforms, enhancing the user's music discovery and listening experience.

* 3.1.2 Stakeholders
  1. *Identify and describe those who will have an interest in the solution explaining how the solution is appropriate to their needs (this may be named individuals, groups or persona that describes the target end user).*

| User Demographic Statistics | | Apple Music | Spotify |
| --- | --- | --- | --- |
| Age | 18-24 |  | 26% |
| 25-34 |  | 29% |
| 35-44 |  | 16% |
| 45-54 |  | 11% |
| 55+ |  | 19% |
| Gender |  |  |  |
|  |  |  |

The clients and demographic for this software would be users of any device with internet access and a subscription to Apple Music or Spotify to listen to the full versions of songs generated in the playlist. The stakeholders of the \*NAME\* are people of all ages keen on music and exploring new artists that align with their music taste. Due to the user-friendly software, older people and children can easily navigate through and use it; therefore, the website does not have age restrictions. The stakeholders will be interested in a solution allowing them to discover and listen to new artists and songs explicitly tailored for them based on their music preferences.

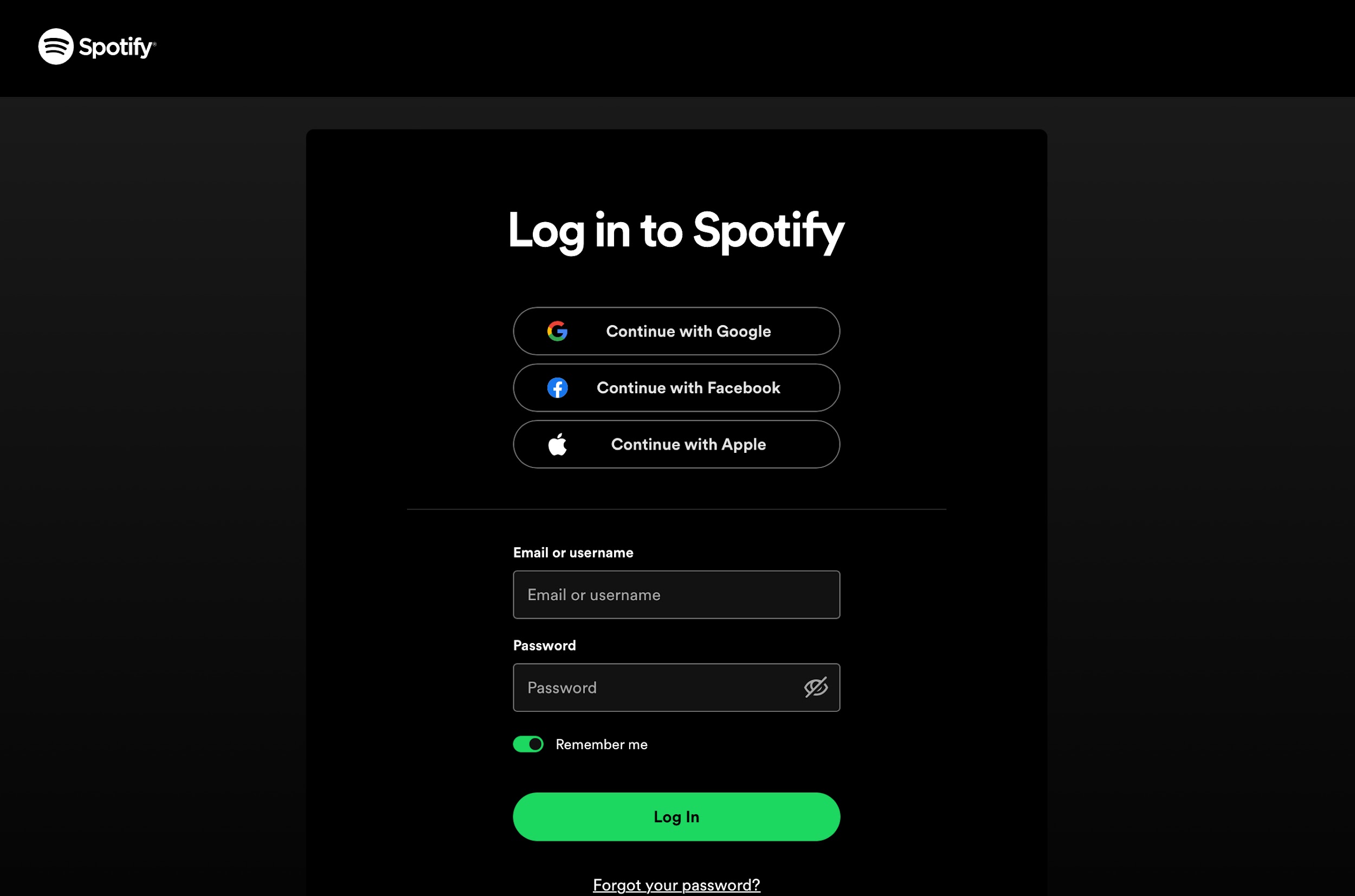
Music enthusiasts, such as avid listeners or collectors, who have a strong interest in expanding their music library will also be interested in the \*NAME\* software because it provides them with a curated playlist-generation algorithm that enhances music discovery by considering their preferences and feedback to ensure that the playlist fulfils users’ expectations, which caters to the interests of music enthusiasts.

Artists and bands can also be the stakeholders for the software, as the playlist-generating algorithm can expose users to artists and bands they may not have been aware of before. Therefore less-known music creators will be interested in the software that will allow them to increase their exposure and popularity; assist them in reaching their target audience.

Music streaming platforms such as Apple Music or Spotify can be interested in the solution because it integrates their services and directs users to their platforms. The links to platforms that allow users to continue listening to music will increase user engagement and lead to higher subscription rates and usage of their services. Thus the solution will benefit both users and music streaming platforms because users will enjoy a seamless experience. In contrast, platforms will increase their analytics, increasing the platforms’ profitability. Overall, the website is a valuable extension of the music streaming platforms, enhancing their offerings and fostering a deeper connection with users.

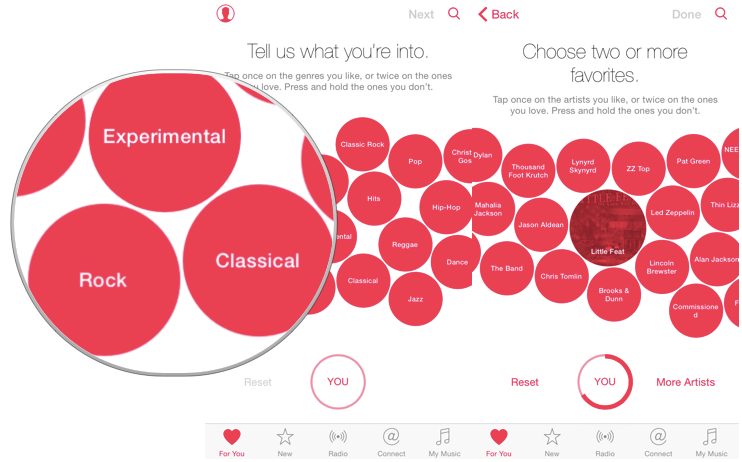
The music industry and labels are vested in promoting their artists and music catalogue. The solution will expose users to a wide range of artists, including both emerging and already established ones, which will help promote their music and increase sales or streaming revenue.

* 3.1.3 Research the problem
  1. Research the problem and solutions to similar problems to identify and justify suitable approaches to a solution.
  2. Describe the essential features of a computational solution explaining these choices.
  3. Explain the limitations of the proposed solution.



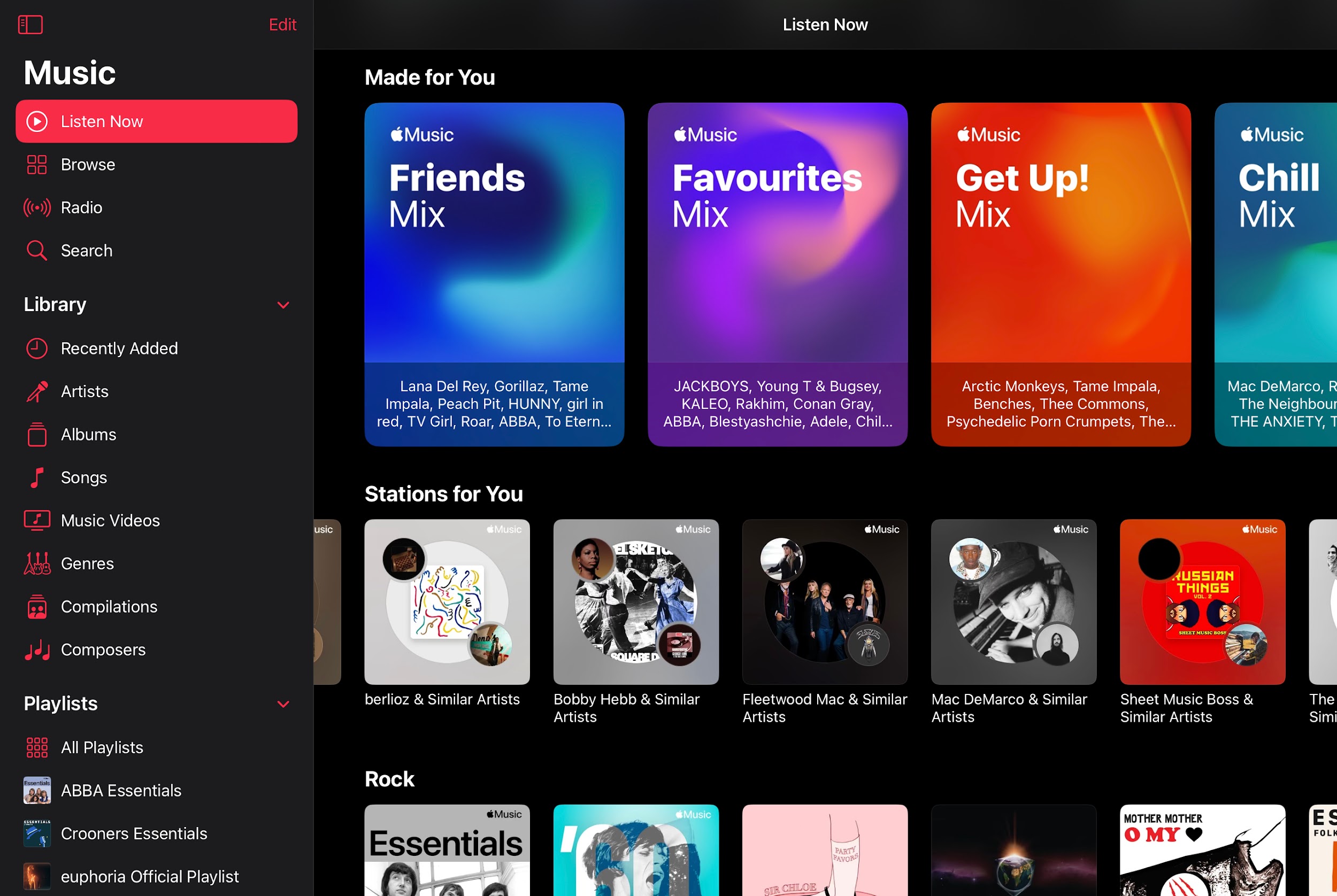
User Authentication and Account Management:

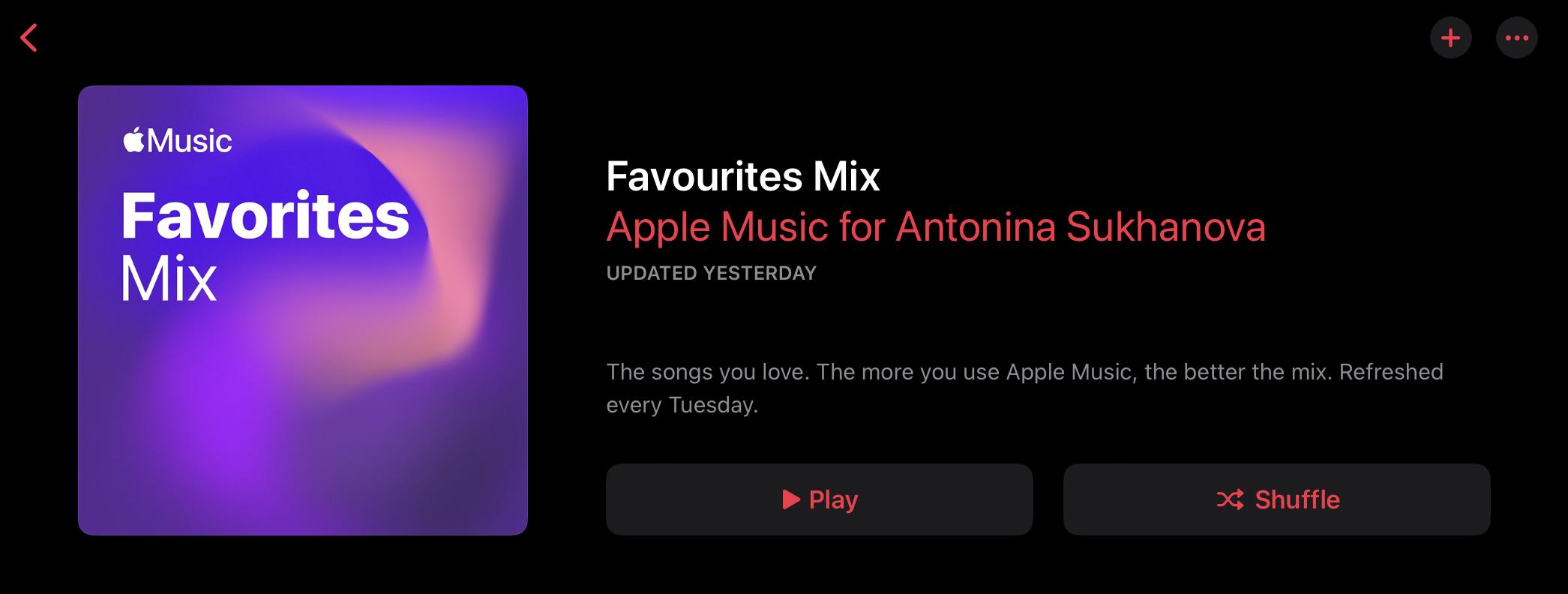
* Essential Feature: User registration, login, and account management functionalities to allow users to create and manage their profiles, preferences, and playlists.
* Choice: Implementing a secure user authentication system, such as username/password or social media login integration, to protect user data and ensure personalised experiences.
* Limitations: The solution may require users to provide personal information during registration, which raises privacy concerns. Ensuring proper data security measures and obtaining user consent is crucial.

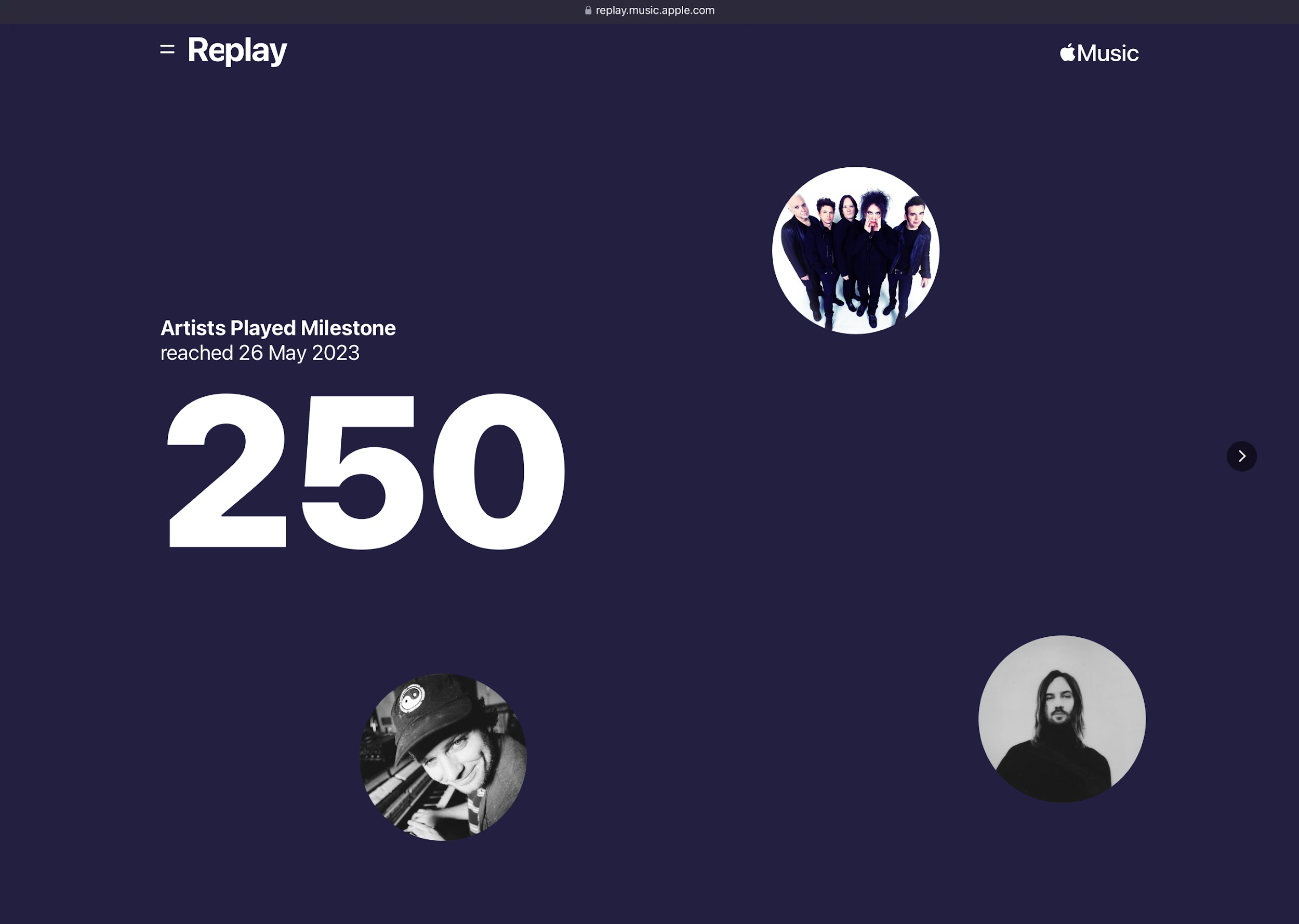
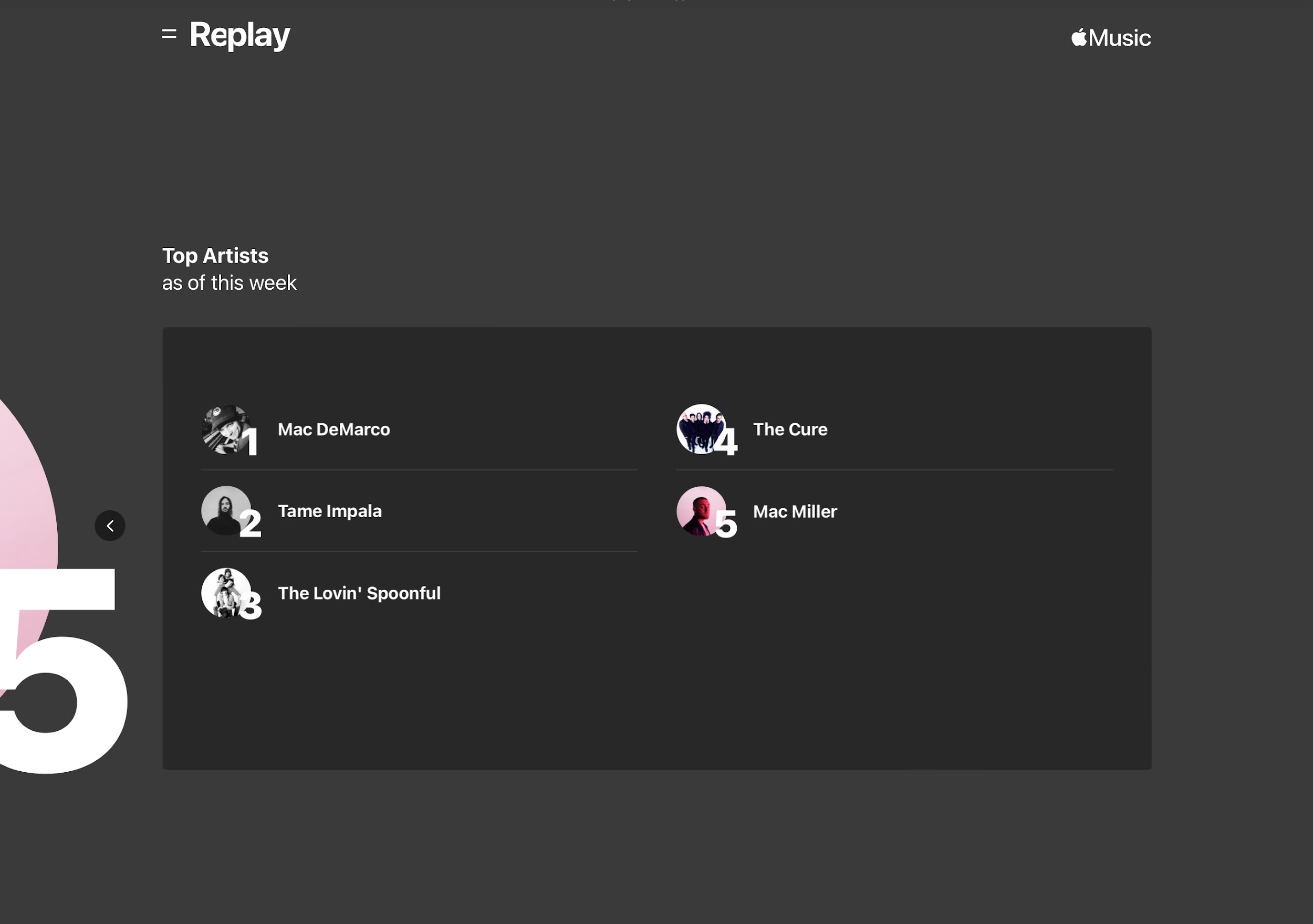
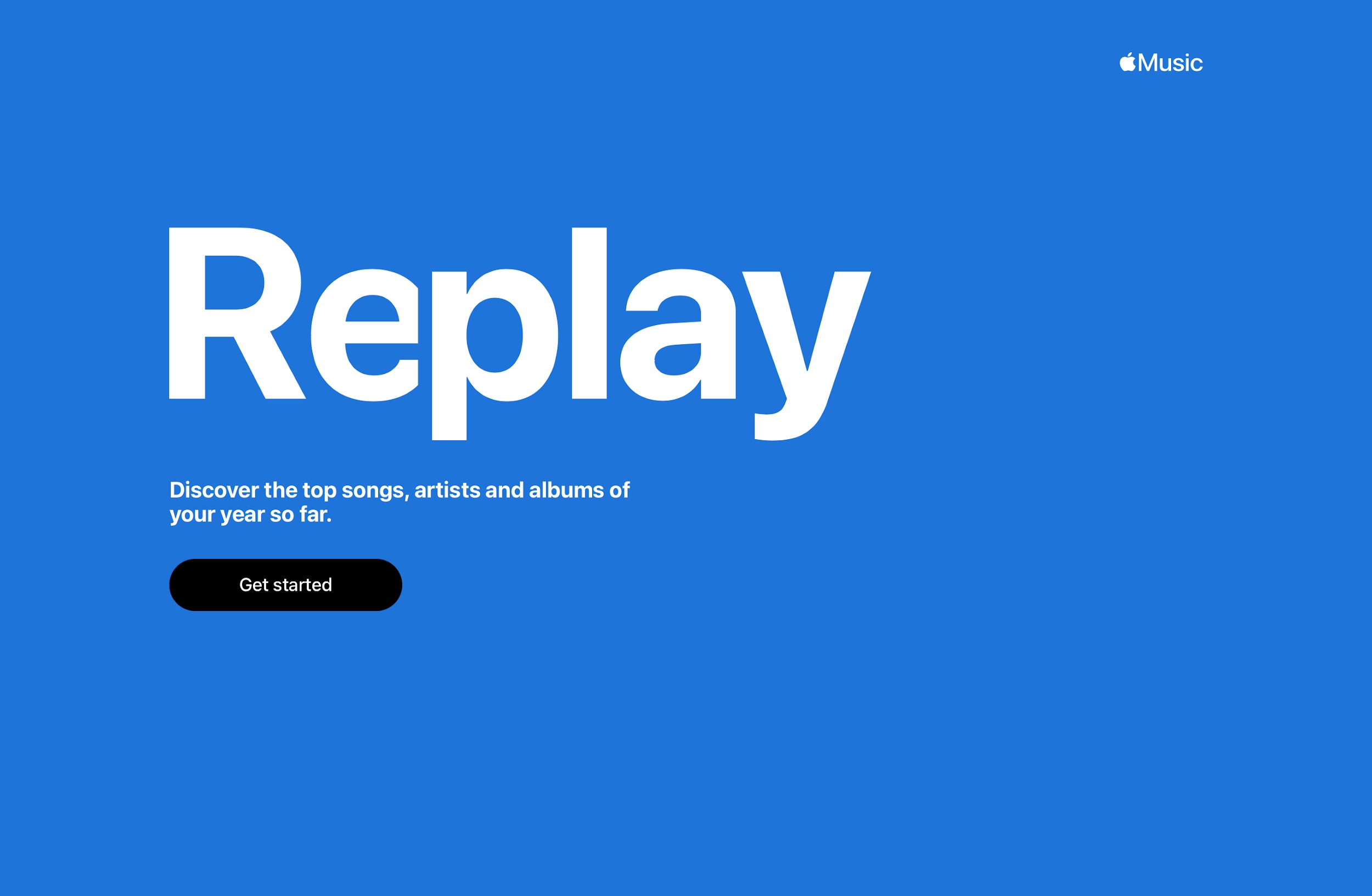
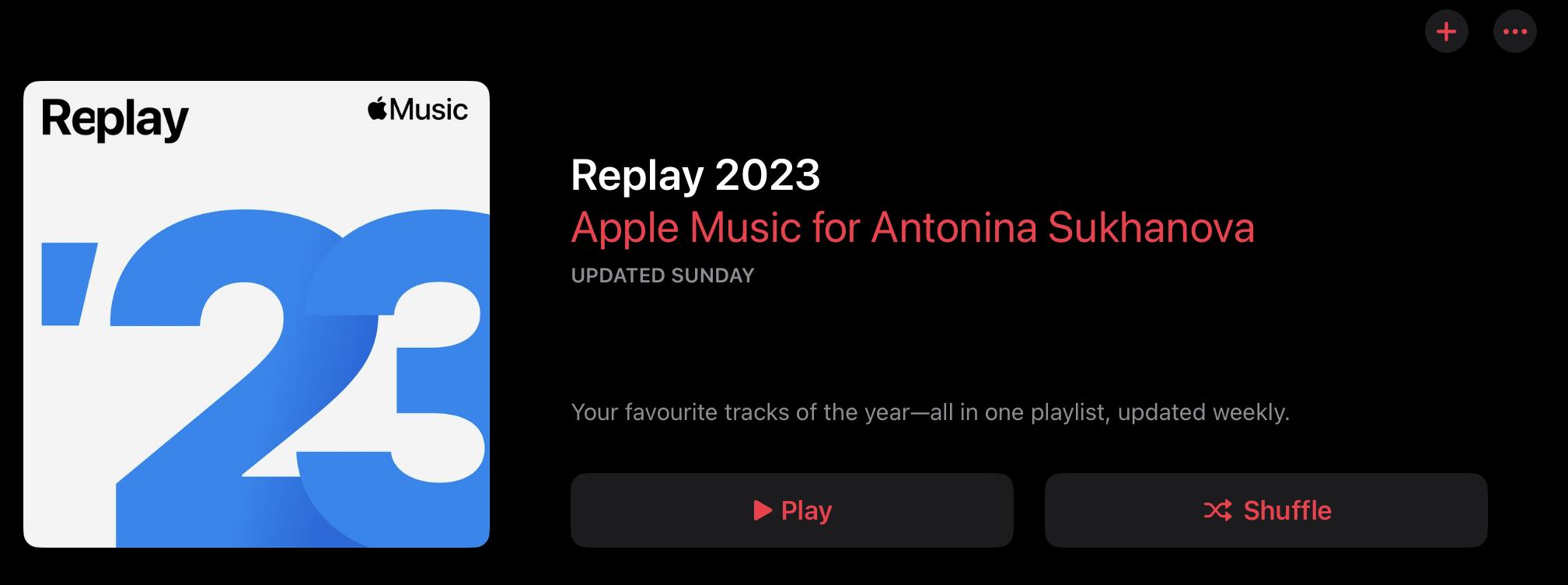


Preference Capture and Customization:

* Essential Feature: A user-friendly interface to capture and store user preferences, such as favourite music genres and liked/disliked artists.
* Choice: Providing intuitive forms or interactive elements to gather user preferences, ensuring ease of use and accurate data capture.
* Limitations: The solution relies on user-provided data, which can be subjective or may not fully reflect their true preferences. Balancing user input with algorithmic recommendations is essential to improve the accuracy of the generated playlists.

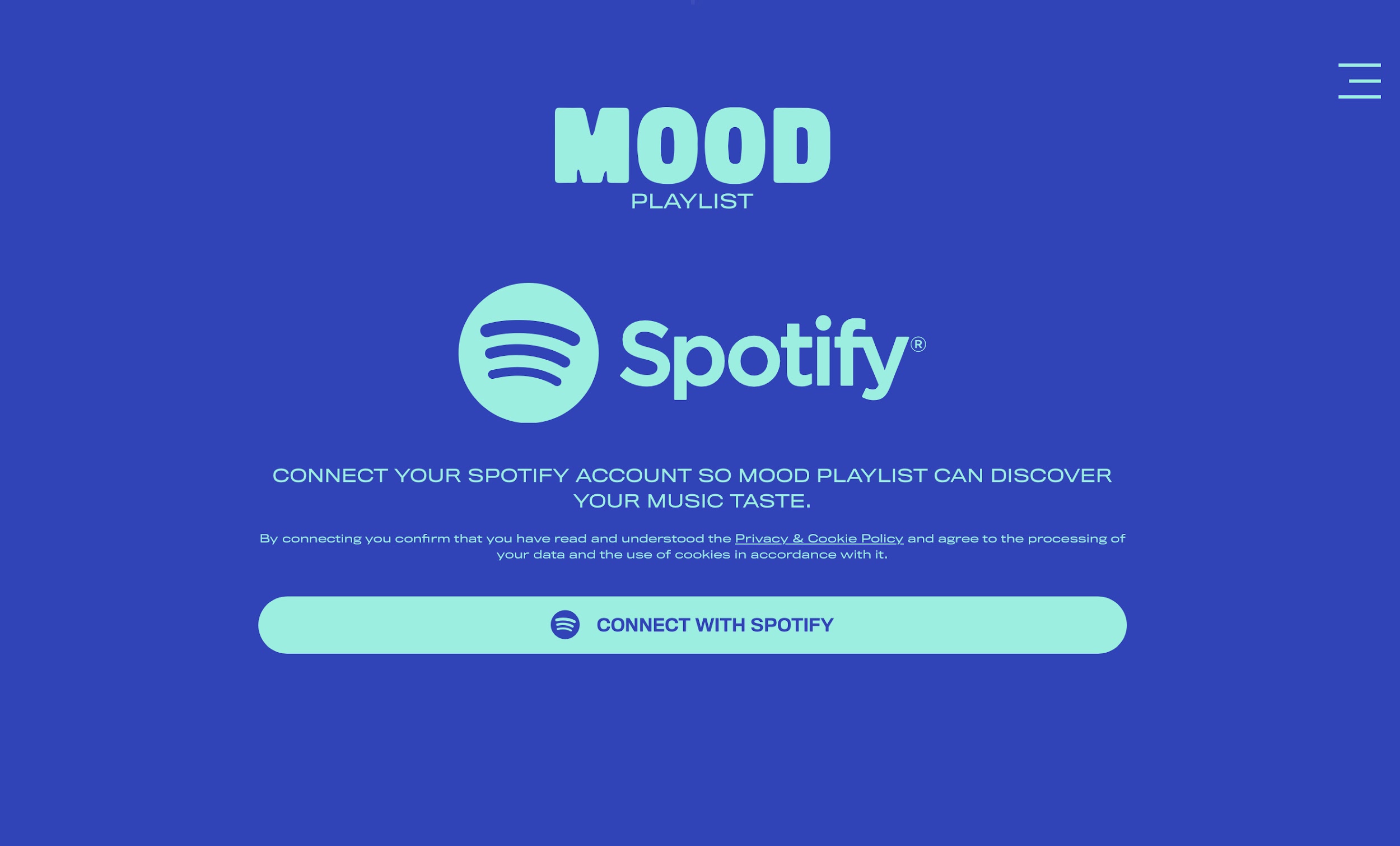






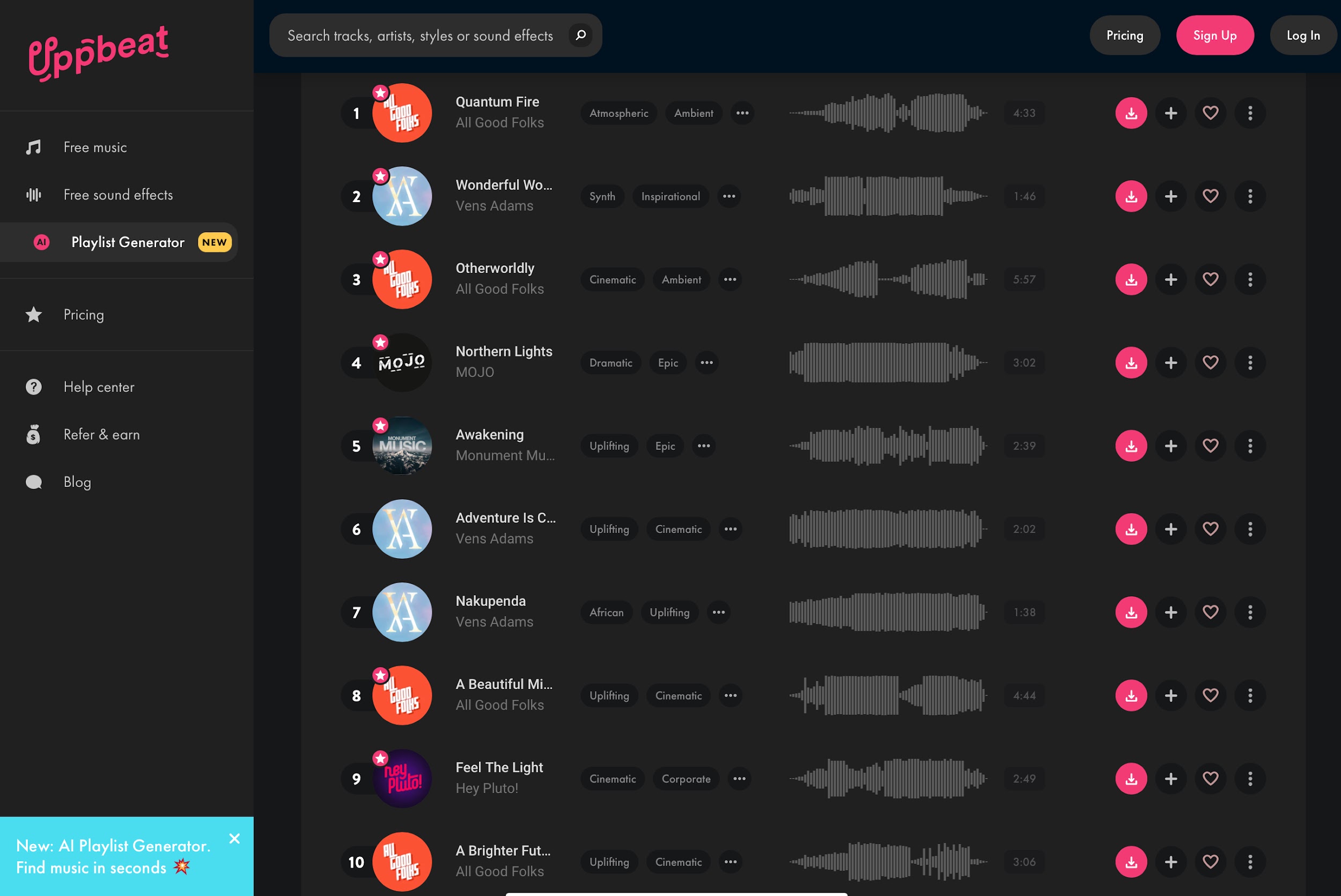
Playlist Generation Algorithm:

* Essential Feature: Developing an algorithm that takes into account user preferences, such as favourite genres and liked/disliked artists, to generate personalised playlists.
* Choice: Employing machine learning techniques, collaborative filtering, or content-based filtering to analyse user preferences and generate accurate playlist recommendations.
* Limitations: The accuracy of the algorithm's recommendations depends on the quality and diversity of user input data. Lack of user feedback or limited data can affect the algorithm's ability to generate highly personalised playlists.



External Music Platform Integration:

* Essential Feature: Integration with external music platforms, such as Apple Music or Spotify, to allow users to continue listening to recommended songs seamlessly.
* Choice: Utilizing the APIs provided by the music platforms to generate URLs or requests that open specific songs, albums, or playlists in the respective platform's app or website.
* Limitations: The solution relies on the availability and functionality of the external music platform APIs. Any changes or restrictions imposed by the platform providers can impact the integration and user experience.



Feedback and Learning Mechanism:

* Essential Feature: Incorporating a feedback mechanism where users can rate or provide feedback on the recommended songs to improve future playlist generation.
* Choice: Implementing a feedback system that collects user ratings, listens to user feedback, and uses this information to refine the playlist generation algorithm.
* Limitations: The effectiveness of the learning mechanism depends on user engagement and the quantity/quality of feedback received. Encouraging users to provide feedback and continuously updating and refining the algorithm are essential for better playlist generation.

3.1.4 Specify the proposed solution

* 1. Specify and justify the solution requirements including hardware and software configuration (if appropriate).
  2. Identify and justify measurable success criteria for the proposed solution.

The proposed solution for the website that generates personalized playlists and integrates with external music platforms requires the following solution requirements:

**Hardware Configuration:**

Web Server:

A reliable web server capable of handling user requests, hosting the website, and storing user data securely. To justify this choice, the playlist-generating website requires a web server to function and be accessible to the audience.

Database Server:

A database server to store user profiles, preferences, playlist data, and feedback information. The website requires a database server to store the data efficiently in one place.

**Software Configuration:**

Programming Languages:

Backend development using languages such as Python, Java, or Node.js to implement the server-side logic and algorithms. It is essential for website coding.

Frameworks and Libraries:

Utilizing appropriate frameworks or libraries for web development (e.g., Django, Flask, Express.js) and data processing (e.g., pandas, scikit-learn) to expedite development and leverage existing tools.

APIs:

Integrating with external music platforms (e.g., Apple Music, Spotify) using their provided APIs to fetch music data and redirect users to their platforms.

User Interface (UI) Design:

Creating an intuitive and user-friendly interface for user registration, login, preference input, playlist display, and feedback submission. Designing responsive UI components to ensure compatibility across different devices and screen sizes.

Playlist Generation Algorithm:

Developing a recommendation algorithm using machine learning techniques such as collaborative filtering, content-based filtering to generate personalised playlists based on user preferences.

Implementing an iterative learning mechanism that adapts the algorithm based on user feedback and continuously improves playlist recommendations.

This solution will ensure that website generates personalised recommendations to each user and adapts to the feedback given.

External Platform Integration:

Leveraging the APIs provided by external music platforms (e.g., Apple Music, Spotify) to seamlessly integrate with their services.

Implementing functionality to generate deep links or requests that open specific songs, albums, or playlists on the respective platform's app or website.

**Measurable Success Criteria for the Proposed Solution:**

User Satisfaction:

Measuring user satisfaction through user feedback surveys or ratings to gauge how well the generated playlists align with their preferences and provide an enjoyable listening experience.

Playlist Relevance:

Evaluating the relevance of the generated playlists by comparing user feedback and ratings with the recommendations. Higher ratings and positive feedback indicate more accurate and relevant recommendations.

Engagement and Usage Metrics:

Tracking user engagement metrics, such as the number of active users, playlist creation frequency, average listening time, and user retention rates, to assess the solution's effectiveness and user adoption.

Seamless Integration:

Monitoring the successful redirection of users to external music platforms and tracking the number of click-throughs to Apple Music or Spotify, indicating the ease of integration and user satisfaction with the external platform experience.

Algorithm Improvement:

Assessing the algorithm's performance over time by measuring the increase in playlist relevance, user satisfaction, and user engagement as the algorithm learns from user feedback and continuously improves its recommendations.

These success criteria provide measurable indicators to evaluate the effectiveness of the proposed solution, ensuring that it meets user expectations, generates relevant playlists, integrates seamlessly with external platforms, and continually improves over time.

**3.2 Design of the solution (15 marks)**

* 3.2.1 Decompose the problem
  1. Break down the problem into smaller parts suitable for computational solutions justifying any decisions made.
* 3.2.2 Describe the solution
  1. Explain and justify the structure of the solution.
  2. Describe the parts of the solution using algorithms justifying how these algorithms form a complete solution to the problem.
  3. Describe usability features to be included in the solution
  4. Identify key variables / data structures / classes justifying choices and any necessary validation.
* 3.2.3 Describe the approach to testing
  1. Identify the test data to be used during the iterative development and post development phases and justify the choice of this test data.

**3.2.1 Decompose the problem:**

To decompose the problem of generating personalised playlists and integrating with external music platforms, we can break it down into smaller parts suitable for computational solutions:

User Authentication and Account Management:

Registration: Collect user information such as username, password, and email address for account creation.

Login: Verify user credentials and grant access to personalized features.

Account Management: Allow users to update their profile, preferences, and manage their playlists.

Preference Capture:

Genre Selection: Provide a list of genres or interactive elements for users to select their favourite genres.

Artist Preference: Allow users to specify their liked/disliked artists through search or selection.

Playlist Generation:

Algorithm Implementation: Design and implement a playlist generation algorithm that takes into account user preferences, genre popularity, artist popularity, and possibly other factors like song popularity or release date.

Algorithm Training: Utilize machine learning techniques to continuously learn from user feedback and improve the accuracy of playlist recommendations.

External Platform Integration:

API Integration: Utilize the APIs provided by external music platforms (e.g., Apple Music, Spotify) to fetch music data, create playlists, and redirect users to the respective platform for continued listening.

**3.2.2 Describe the solution:**

The solution structure will consist of different components/modules that work together to achieve the desired functionality:

User Authentication and Account Management:

Algorithm: Use standard authentication algorithms and frameworks (e.g., bcrypt, OAuth) for secure user registration, login, and account management.

Preference Capture:

Algorithm: Implement algorithms to capture and store user preferences, such as favorite genres and liked/disliked artists, using appropriate data structures.

Playlist Generation:

Algorithm: Develop a playlist generation algorithm that incorporates user preferences, genre popularity, artist popularity, and other relevant factors. This algorithm will generate personalized playlists based on user input and can be continuously improved through machine learning techniques.

External Platform Integration:

API Integration: Utilize the APIs provided by external music platforms to fetch music data and create deep links or requests that redirect users to the respective platforms for continued listening.

Usability features to be included in the solution:

User-Friendly Interface: Design an intuitive and visually appealing user interface with clear instructions and interactive elements for a seamless user experience.

Playlist Preview: Provide users with a preview or short snippets of the recommended songs to allow them to make informed choices.

Feedback Mechanism: Implement a feedback system that allows users to rate and provide feedback on the recommended songs to improve future playlist generation.

Key variables/data structures/classes:

User Profile: A data structure to store user information such as username, password, email, and preferences.

Playlist: A data structure or class to represent a playlist, including information such as the list of songs, artist, genre, and external platform links.

Preference Data: Data structures to store user preferences, such as favorite genres and liked/disliked artists, which will be utilized by the playlist generation algorithm.

**3.2.3 Describe the approach to testing:**

During the iterative development and post-development phases, it is important to conduct thorough testing to ensure the correctness and effectiveness of the solution. The following test data can be used:

Unit Testing:

Test individual components, algorithms, and functions using mock data to ensure they perform as expected. This can involve generating sample playlists based on specific user preferences and verifying the accuracy of the generated results.

Integration Testing:

Test the integration between different modules, such as user authentication, preference capture, playlist generation, and external platform integration. This involves simulating user interactions and verifying that data flows correctly between the modules.

User Acceptance Testing:

Involve real users or representative users to provide feedback on the usability, intuitiveness, and effectiveness of the solution. Users can be given specific scenarios to test, such as providing feedback on playlist recommendations and assessing the integration with external music platforms.

Performance Testing:

Assess the solution's performance under different loads, including a large number of users and concurrent playlist generation requests. This ensures that the solution can handle the expected user traffic and provide a responsive experience.

The choice of test data should cover a variety of scenarios and edge cases, including different genres, artist preferences, and feedback scenarios. It should also include data that represents different user profiles and preferences to ensure the solution caters to a diverse user base.

**3.3 Developing the solution (25 marks)**

* 3.3.1 Iterative development process
  1. Provide annotated evidence of each stage of the iterative development process justifying any decision made.
  2. Provide annotated evidence of prototype solutions justifying any decision made.
* 3.3.2 Testing to inform development
  1. Provide annotated evidence for testing at each stage justifying the reason for the test.
  2. Provide annotated evidence of any remedial actions taken justifying the decision made.

**3.3.1 Iterative development process:**

The iterative development process involves breaking down the development into smaller iterations or sprints, where each iteration focuses on implementing and testing a subset of features. Here is an overview of the iterative development process:

Requirement Gathering:

Identify and document the requirements for each iteration based on the project specifications and user needs.

Design:

Create a design for the features to be implemented in the current iteration. This includes designing the user interface, data structures, algorithms, and integration with external platforms.

Implementation:

Develop the features based on the design. Start by implementing the core functionalities, such as user authentication, preference capture, and playlist generation. Use the chosen programming language, frameworks, and libraries to build the solution.

Testing:

Conduct testing during the development process to identify and fix issues early. This includes unit testing, integration testing, and user acceptance testing. Test cases should cover different scenarios and edge cases to ensure the solution works as expected.

Review and Feedback:

Seek feedback from stakeholders, users, or peers to evaluate the implemented features and gather suggestions for improvements. Incorporate the feedback and make necessary adjustments to the solution.

Documentation:

Document the changes made during the iteration, including any modifications to the requirements, design, and implementation. Update user documentation, such as user guides and API documentation, as necessary.

Deployment and Evaluation:

Deploy the updated solution in a test environment or to a subset of users for evaluation. Collect feedback and evaluate the solution's performance, usability, and effectiveness. Use this feedback to inform future iterations.

Repeat:

Repeat the above steps for subsequent iterations, incorporating new features and addressing any issues or enhancements identified during the previous iterations.

**3.3.2 Testing to inform development:**

Testing plays a crucial role in informing the development process and ensuring the quality and reliability of the solution. Here are some key testing activities and their justifications:

Unit Testing:

Perform unit tests on individual components, algorithms, and functions to verify their correctness and identify any bugs or issues. These tests help catch errors early in the development process and ensure the components function as expected.

Integration Testing:

Test the integration between different modules to ensure data flows correctly and the functionalities work together seamlessly. This testing helps identify any issues arising from the interaction between different parts of the system.

User Acceptance Testing:

Involve real users or representative users to test the solution and provide feedback on its usability and effectiveness. This testing helps identify any usability issues, gather user feedback for improvements, and ensure the solution meets user expectations.

Performance Testing:

Evaluate the performance of the solution under different loads and stress conditions. This testing helps identify performance bottlenecks, scalability issues, and ensures the solution can handle the expected user traffic.

Regression Testing:

Perform regression testing after making changes or adding new features to ensure that existing functionalities have not been affected by the updates. This helps prevent regression bugs and ensures the overall stability of the solution.

During the testing process, if any issues or bugs are identified, appropriate remedial actions should be taken. These actions may include debugging the code, making necessary fixes, and retesting the affected components. The decisions regarding remedial actions should be based on the severity of the issues, impact on the functionality, and project priorities.

**3.4 Evaluation (20 marks)**

* 3.4.1 Testing to inform evaluation
  1. Provide annotated evidence of testing the solution of robustness at the end of the development process.
  2. Provide annotated evidence of usability testing (user feedback)
* 3.4.2 Success of the solution
  1. Use the test evidence from the development and post development process to evaluate the solution against the success criteria from the analysis.
* 3.4.3 Describe the final product
  1. Provide annotated evidence of the usability features from the design, commenting on their effectiveness.
* 3.4.4 Maintenance and development
  1. Discuss the maintainability of the solution.
  2. Discuss potential further development of the solution.

**3.4.1 Testing to inform evaluation:**

To evaluate the solution, it is important to conduct robustness testing and gather user feedback through usability testing. Here is an outline of the testing activities:

Robustness Testing: Test the solution's robustness by subjecting it to various stress conditions and edge cases. This includes testing the solution's performance under heavy loads, testing error handling and recovery mechanisms, and ensuring the solution handles unexpected inputs gracefully. Annotated evidence can include test cases, test results, and any issues or bugs encountered during the testing process.

Usability Testing: Conduct usability testing with real users or representative users to gather feedback on the solution's usability and user experience. This can be done through user interviews, surveys, or observation of user interactions with the solution. Annotated evidence can include user feedback, usability test results, and any observations or insights gained from the testing process.

**3.4.2 Success of the solution:**

Evaluate the solution's success based on the evidence gathered from testing during the development and post-development phases. Compare the results against the success criteria established during the analysis phase. Annotated evidence can include comparisons of the test results against the success criteria, highlighting the areas where the solution meets or exceeds expectations, as well as areas where improvements can be made.

**3.4.3 Describe the final product:**

Describe the final product, highlighting the usability features identified in the design phase and commenting on their effectiveness. Annotated evidence can include screenshots or prototypes of the user interface, user feedback on specific usability features, and any iterations or improvements made based on user testing and feedback.

For example, if one of the usability features is a playlist preview, provide evidence of how users interacted with the feature, their feedback on its usefulness, and any adjustments made to enhance the user experience.

**3.4.4 Maintenance and development:**

Discuss the maintainability of the solution, considering factors such as code maintainability, scalability, and extensibility. Annotated evidence can include documentation or code snippets that demonstrate good coding practices, modular design, and proper documentation.

Additionally, discuss potential further development of the solution. Identify areas where the solution can be enhanced or expanded in the future based on user feedback, technological advancements, or evolving user needs. Provide insights and suggestions on how the solution can be further developed to better serve its intended purpose.

Overall, the evaluation stage focuses on assessing the solution's robustness, usability, success against the established criteria, and its potential for future maintenance and development. It involves gathering evidence from testing and user feedback to make informed judgments about the solution's effectiveness and identify areas for improvement.