1. Introduction 1.1 Purpose The purpose of the Music Streaming web application is to provide users with a seamless and enjoyable experience for discovering, listening to, and managing music content online.

1.2 Scope The application will include features such as user authentication, personalized playlists, a vast music library, social sharing, offline mode, and recommendations based on user preferences.

1.3 Stakeholders Stakeholders include end-users, artists, record labels, advertisers, and the development team.

2. System Overview 2.1 System Architecture Adopt a microservices architecture with components such as user authentication, content delivery, recommendation engine, and payment processing.

2.2 Technologies Used Frontend: React.js for a dynamic and responsive user interface. Backend: Node.js with Express.js for server-side logic. Database: MongoDB for metadata storage, Amazon S3 for audio file storage. Authentication: OAuth 2.0 for secure user authentication. Streaming: Use a content delivery network (CDN) for efficient media streaming. 3. Functional Requirements 3.1 User Stories As a user, I can create an account and log in. As a user, I can search for songs, albums, and artists. As a user, I can create and manage playlists. As a user, I can follow artists and friends. As a user, I can listen to music in offline mode. 3.2 Use Cases Search for Music Users can search for specific songs, albums, or artists. Create Playlist Users can create, edit, and delete playlists. Follow Artists Users can follow their favorite artists for updates. Offline Mode Users can download music for offline listening. 4. Non-Functional Requirements 4.1 Performance Ensure low latency for streaming, aiming for an average load time of under 2 seconds. The system should support at least 100,000 concurrent users. 4.2 Security Implement secure authentication and authorization mechanisms. Encrypt communication between the client and server. 4.3 Usability The user interface should be intuitive, aesthetically pleasing, and accessible. Support multiple devices and screen sizes. 5. User Interface Design 5.1 Wireframes Include wireframes for key screens, such as the homepage, music player, user profile, and playlist management.

5.2 Navigation Describe the navigation flow, including menus, buttons, and gestures.

6. Database Design 6.1 Entity-Relationship Diagram (ERD) Provide an ERD illustrating the relationships between entities like users, songs, playlists, and artists.

6.2 Data Storage Specify how metadata and audio files will be stored, retrieved, and updated.

7. System Components 7.1 Frontend Detail the frontend components, libraries, and frameworks.

7.2 Backend Specify the backend components, microservices, and APIs.

7.3 Third-Party Integrations List and describe any third-party services or APIs to be integrated, such as payment gateways or social media sharing.

8. Testing Plan 8.1 Unit Testing Detail the strategy for unit testing individual components.

8.2 Integration Testing Explain how different components will be tested together.

8.3 User Acceptance Testing (UAT) Outline the plan for involving end-users in testing to validate the system.

9. Deployment Plan 9.1 Deployment Architecture Specify the deployment environment, including hosting services, server configurations, and scalability considerations.

9.2 Release Plan Provide a detailed timeline for releases, considering beta or pilot phases and any rollbacks in case of issues.

10. Maintenance and Support 10.1 Bug Tracking Describe the process for tracking and resolving bugs, including tools and workflows.

10.2 Updates and Enhancements Outline the procedures for rolling out updates, incorporating user feedback, and implementing new features.

11. Conclusion Summarize the key points, emphasizing the importance of delivering a secure, efficient, and user-friendly Music Streaming web application.