Test Design Document

for

Music Streaming System

Version 2.0 approved

Prepared by Group 5

2324II INT2208E 23, VNU-UET

April 9, 2024

Revision History

Date	Version	Description	Author
May 10th 2024	1.0	First draft with section titles	Nguyễn Đức Khánh
May 15th 2024	1.1	Update document	Trần Thế Mạnh

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this document is to present the initial design for a comprehensive Music Streaming System. This system is envisioned as a robust, user-friendly platform that will enable music enthusiasts to stream their favorite tracks, discover new music, and curate personalized playlists. The primary purpose of this Test Design Document is to provide a systematic and structured approach to testing the functionality, performance, and reliability of our Music Streaming System. It aims to ensure that the system meets its specified requirements and provides a high-quality experience for our users. This document will serve as a guide for our testing team, outlining the test cases, procedures, and criteria for each feature of the system. It will help ensure that all aspects of the system are thoroughly tested, from the user interface to the backend services, and that any defects or issues are identified and addressed before the system is released to our users.

1.2. Scope

The Music Streaming System is a web-based application that will allow users to listen to music from a vast library of tracks spanning various genres, moods, and eras. Users will be able to search for songs, albums, or artists, create and manage their own playlists, and explore music based on their listening history and preferences. The system will also include features for user authentication, music recommendation, and social interaction among users.

2. Test Strategy and Approach

Our test strategy for the Music Streaming WebApp is designed to ensure comprehensive coverage of all features and functionalities. We will employ a multi-tiered testing approach, starting from the smallest units of code and gradually integrating them to test in a combined environment. Our approach will ensure that the application performs optimally under various loads and provides a seamless user experience.

2.1. Test Types

- **Unit Tests**: Unit tests will be our first line of defense, verifying the correctness of individual components in isolation. Developers will write and run these tests as they code, catching and fixing issues early in the development cycle. We will use a unit testing framework suitable for our development language.
- **Integration Tests**: Once our components are tested individually, we will perform integration tests. These tests will check how different parts of the application work together, from interactions between different services to data flow between the frontend and backend.

- **Acceptance Tests**: Acceptance tests will be performed to ensure that the system fulfills its requirements and provides the intended functionality. These tests will be based on the user stories and requirements specified for the system. They will be executed in an environment that closely mirrors the production environment.
- **Performance Tests**: Performance tests will be conducted to ensure that our Music Streaming WebApp can handle a large number of users simultaneously without compromising on speed or quality of service. We will simulate various loads on the system and measure its response time, throughput, and resource utilization.

2.2. Test Techniques

Black-box testing is a method where the tester evaluates the functionality of an application without peering into its internal structures or workings. Test cases are derived from the external description of the software—namely, the specifications and requirements. Here's how test cases can be derived using black-box testing for our system:

- *Understanding Requirements*: Begin by thoroughly understanding the functional requirements of the application. For a music streaming app, this could include user stories or use cases for features like account creation, music playback, playlist management, and search functionality.
- *Identifying Test Scenarios*: Based on the requirements, identify various scenarios that the application is expected to handle. For instance, creating a playlist, adding songs to a playlist, searching for a specific artist, or handling multiple simultaneous streams.
- *Defining Input Data*: Determine the input data needed to test each scenario. This could involve different genres of music, various user actions, or different search terms.
- *Determining Expected Outputs*: For each input, define the expected output. For example, when a user searches for a specific artist, the expected output would be a list of songs or albums by that artist.
- *Creating Test Cases*: Combine the scenarios, input data, and expected outputs to create detailed test cases. Each test case should include the steps to execute the test, the input data to be used, and the expected result.
- *Executing Test Cases*: Run the tests by providing the input data and comparing the actual output with the expected output. Any discrepancies should be logged as defects.

- *Reviewing Results*: Analyze the test results to ensure that the application behaves as expected across all test cases.

By focusing on the inputs and outputs, black-box testing can validate that the application is functioning correctly from the user's perspective, regardless of the internal code structure. This method is particularly useful for ensuring that the application meets the user's needs and adheres to the specified requirements.

2.3. Test Environments

- **Hardware**: The primary testing device will be equipped with an AMD Ryzen 5 5600H with Radeon Graphics, 3.30 GHz processor and an Intel i5 12500H with GeForce RTX 3050 4GB and 16.0 GB RAM. This configuration is representative of a high-performance user device, ensuring that our tests reflect the performance that our users can expect to experience. It will allow us to test the system's performance under heavy load and ensure that it remains responsive even when handling complex tasks.

- **Software**: The tests will be conducted on a device running Windows 11 Home Single Language. This operating system is widely used among our target user base, and testing on this platform will help us ensure that our application is fully compatible and performs optimally on it. The tests will be performed using the Mozilla Firefox browser. Mozilla Firefox is one of the most popular web browsers, known for its speed and reliability. By testing our application on this browser, we can ensure that it works smoothly and correctly for a large portion of our users.

- **Network Configurations**: Our tests will be conducted over a WiFi connection. This will allow us to test the app's performance under typical network conditions. We will simulate different network speeds and signal strengths to understand how these factors might impact the user's experience. For instance, we will test how the app performs when the network connection is weak or unstable, and ensure that it can handle such situations gracefully.

3. Test Cases

Test Case ID	Test Steps	Test Description	Inputs	Expected Outputs	Pass/Fail Criteria	Priority	Dependencies
FLOT001	Verify registration functionality	1. Open the music web application. 2. Navigate to the registration page. 3. Enter valid user details. 4. Click the register button.	Email: "newuser@example.com" Password: "Password123" Confirm Password: "Password123"	User should be registered and redirected to the home page or login page.	User is successfully registered and redirected appropriately.	High	None
FLOT002	Verify registration with invalid email format	1. Open the music web application. 2.Navigate to the registration page. 3. Enter an invalid email format and other required details. 4. Click the "Register" button.	Email:"invalidemail" Password: "Password123" Confirm Password: "Password123"	An error message indicating the email format is invalid should be displayed.	The error message "Invalid email format" is shown.	Medium	None
FLOT003	Verify registration with passwords not matching	1. Open the music web application. 2. Navigate to the registration	Email: "newuser@example.com" Password:" assword123	An error message indicating the passwords do not match should be	The error message "Confirm password must match with password"	Medium	None

		page. 3. Enter valid email and password details but different passwords in the "Password" and "Confirm Password" fields. 4. Click the "Register" button.	" Confirm Password: "Password321"	displayed.	is shown.		
FLOT004	Verify registration with weak password	1.Open the music web application. 2. Navigate to the registration page. 3. Enter valid email but a weak password. 4.Click the "Register" button.	Email: "newuser@example.com" Password: "123" Confirm Password: "123"	An error message indicating the password is too weak should be displayed.	The error message "invalid password form" is shown.	Medium	None
FLOT005	Verify registration with empty fields	1. Open the music web application. 2. Navigate to the registration page. 3. Leave all required fields empty. 4. Click the	N/A	An error message indicating that all fields are required should be displayed.	The error message "invalid entry" is shown.	High	None

		"Register" button.					
FLOT006	Verify login functionality for valid credentials.	1. Navigate to the login page. 2. Enter the email: "testuser@gmail.com" and password "Password123". 3. Click the "Login" button. 4. Verify successful login.	Valid email: "testuser@gmail.com" Valid Password: "Password123"	User should be redirected to the home page on valid login.	User successfully logs in with valid credentials.	High	Pre-existing user account "testuser" with valid credentials.
FLOT007	Verify login functionality with invalid email	1. Open the music web application. 2. Navigate to the login page. 3. Enter an invalid email and valid password. 4. Click the "Login" button.	Email: "invalidemail" Password: "Password123"	An error message indicating invalid email format should be displayed.	The error message "Incorrect email or password" is shown.	Medium	None
FLOT008	Verify login functionality with unregistered email	1. Open the music web application. 2. Navigate to the login page. 3. Enter an unregistered email and	Email: "unregistered@gmail.com" Password: "Password123"	An error message indicating the email is not registered should be displayed.	The error message "Incorrect email or password" is shown.	High	None

		password. 4. Click the "Login" button.					
FLOT009	Verify logout functionality	1. Open the music web application. 2. Log in with valid credentials. 3. Click the logout button.	N/A	User should be redirected to the login page.	User is successfully logged out and redirected to the login page.	High	User must be logged in.
FLOT010	Verify play track functionality	1. Open the music web application. 2. Log in with valid credentials. 3. Navigate to the search bar. 4. Enter the track name and search. 5. Select the track from the search results. 6. Click the play button.	Valid Username: "user@example.com" Valid Password: "Password123" Track Name: "Song Title"	The selected track should start playing.	The track starts playing upon clicking the play button.	High	User account with a valid subscription.
FLOT011	Verify pausing a track	1. Open the music web application. 2. Log in with valid credentials. 3. Play a	N/A	The playing track should be paused.	The track playback is paused successfully.	High	Track must be playing.

		track. 4. Click the pause button.					
FLOT012	Verify seeking forward a track	1.Open the music web application. 2.Log in with valid credentials. 3. Play a track. 4. Drag the seek bar forward to a specific time.	Seek Time: Forward to 1:30	The track should jump to the specified forward time (1:30).	The track resumes playback from the specified time (1:30).	Medium	Track must be playing.
FLOT013	Verify seeking backward a track	1.Open the music web application. 2. Log in with valid credentials. 3. Play a track. 4. Drag the seek bar backward to a specific time.	Seek Time: Backward to 0:30	The track should jump to the specified backward time (0:30).	The track resumes playback from the specified time (0:30).	Medium	Track must be playing.
FLOT014	Verify setting the volume functionality	1. Open the music web application. 2. Log in with valid credentials. 3. Play a track.	Volume: 50%	The volume should be set to 50% and reflected in the UI.	The volume level changes to 50% and the change is audible.	High	Track must be playing.

		4. Locate the volume control slider. 5. Adjust the volume to 50%. 6. Verify the volume level.					
FLOT015	Verify removing a track from a playlist	1. Open the music web application. 2. Log in with valid credentials. 3.Navigate to "My Playlists". 4. Select a playlist containing the track. 5. Click on the track's options menu. 6. Select "Remove from Playlist".	Track Name: "Song Title" Playlist Name: "My Playlist"	The track should be removed from the playlist.	The track no longer appears in the playlist.	Medium	User must be logged in. Playlist must contain the track.
FLOT016	Verify add to playlist functionality	1. Open the music web application. 2. Log in with valid credentials 3. Search for a track. 4. Select	Track Name: "Song Title" Playlist Name: "My Playlist"	The track should be added to the selected playlist.	The track appears in the specified playlist.	Medium	User must be logged in. A playlist must exist or the ability to create one must be enabled.

		the track from search results. 5. Click the "Add to Playlist" button. 6. Select an existing playlist or create a new one.					
FLOT017	Verify creating a playlist	1. Open the music web application. 2. Log in with valid credentials. 3. Navigate to the "Playlists" section. 4. Click the "Create Playlist" button. 5. Enter a name for the playlist. 6. Click "Create".	Playlist Name: "My Playlist"	The new playlist should be created and visible in the playlists section.	The playlist "My Playlist" is created and visible.	High	User must be logged in.
FLOT018	Verify removing a playlist	1. Open the music web application. 2. Log in with valid credentials. 3. Navigate to the "Playlists"	Playlist Name: "My Playlist"	The selected playlist should be removed and no longer visible in the playlists section.	The playlist "My Playlist" is removed and no longer visible.	High	User must be logged in. Playlist "My Playlist" must exist.

		section. 4. Select the playlist to be removed. 5. Click the "Delete" button. 6. Confirm the deletion.					
FLOT019	Verify creating a playlist with no name	1. Open the music web application. 2. Log in with valid credentials. 3. Navigate to the "Playlists" section. 4. Click the "Create Playlist" button. 5. Leave the playlist name field empty. 6. Click "Save".	N/A	An error message indicating the playlist name is required should be displayed.	The error message "Playlist name is required" is shown.	Medium	User must be logged in.
FLOT020	Verify search functionality	1. Open the music web application. 2. Log in with valid credentials. 3.Navigate to the	Track Name: "Song Title"	Relevant tracks should be displayed in the search results.	The search results include the track matching the search input.	High	User must be logged in.

		search bar. 4. Enter a track name. 5. Press the search button.					
FLOT021	Verify search functionality	1. Open the music web application. 2. Log in with valid credentials. 3.Navigate to the search bar. 4. Enter a track name. 5. Press the search button.	Track Name: "Song Title"	Relevant tracks should be displayed in the search results.	The search results include the track matching the search input.	High	none

4. Test Data

4.1. Data Sources

The test data for the Music Streaming System will be sourced from a variety of sources to comprehensively evaluate its performance and robustness. We use:

- Generated Data: Synthetic data will be generated to simulate various user scenarios, such as user interactions, playback behaviors, and search queries. This allows for controlled testing of specific functionalities and edge cases.

- Real-world Data: Actual user data obtained from the live environment of the Music Streaming System will be utilized to replicate authentic usage patterns and assess system performance under real-world conditions. This includes historical user activity, content preferences, and user demographics.

4.2. Data Preparation

To ensure accurate and effective testing, the test data will undergo specific preparation and formatting procedures:

- Normalization: Raw data from different sources will be normalized to ensure consistency and uniformity in the test dataset. This includes standardizing data formats, encoding schemes, and data representations.

- Anonymization: User-sensitive information will be anonymized to protect privacy and confidentiality. Personal identifiers such as names, email addresses, and location data will be replaced with anonymized placeholders to comply with privacy regulations and best practices.

- Data Augmentation: In certain cases, data augmentation techniques may be applied to enrich the test dataset with diverse scenarios and variations. This involves introducing synthetic modifications or variations to existing data samples to simulate realistic user interactions and system responses.

5. Test Execution

5.1. Test Schedule

Test Step	Start Date	End Date	Responsibility

(A) Information gathering			
a1) Prepare for Interview	15/5/2004	15/5/2004	Nguyễn Đức Khánh
a2) Conduct Interview	15/5/2004	15/5/2004	Trần Thế Mạnh
a3) Summarize Findings	15/5/2004	15/5/2004	Trần Thế Mạnh
(B) Test Planning			
b1) Build Test Plan	15/5/2004	16/5/2004	Nguyễn Đức Khánh
b2) Define the Metric Objectives	16/5/2004	16/5/2004	Nguyễn Đức Khánh
b3) Review / Approve Plan	17/5/2004	17/5/2004	Nguyễn Đức Khánh
(C) Test Case Design			
c1) Design Function Tests	17/5/2004	17/5/2004	Ngô Lê Hoàng
c2) Design GUI Tests	17/5/2004	17/5/2004	Đỗ Minh Quang
c3) Define the System / Acceptance Tests	17/5/2004	17/5/2004	Ngô Lê Hoàng
Review / Approve Design	17/5/2004	17/5/2004	Ngô Lê Hoàng, Đỗ Minh Quang
(D) Test Development			
d1) Develop Test Scripts	17/5/2004	17/5/2004	Trần Thế Mạnh

d2) Review / Approve Test Development	17/5/2004	17/5/2004	Trần Thế Mạnh
(E) Test Execution/Evaluation			
e1) Setup and Testing	18/5/2004	18/5/2004	Ngô Lê Hoàng
e2) Evaluation	18/5/2004	18/5/2004	Ngô Lê Hoàng
(F) Conduct System Testing			
f1) Complete System Test Plan	19/5/2004	19/5/2004	Trần Thế Mạnh
f2) Complete System Test Cases	19/5/2004	19/5/2004	Ngô Lê Hoàng
f3) Review / Approve System Tests	19/5/2004	19/5/2004	Ngô Lê Hoàng
f4) Execute the System Tests	20/5/2004	20/5/2004	Ngô Lê Hoàng
(G) Conduct Acceptance Testing			
g1) Complete Acceptance Test Plan	20/5/2004	20/5/2004	Trần Thế Mạnh
g2) Complete Acceptance Test Cases	20/5/2004	20/5/2004	Đỗ Minh Quang
g3) Review / Approve Acceptance Test Plan	20/5/2004	20/5/2004	Nguyễn Đức Khánh

g4)Execute the Acceptance Tests	21/5/2004	21/5/2004	Ngô Lê Hoàng
(H) Summarize/Report Test Results			
h1) Perform Data Reduction	21/5/2004	21/5/2004	Ngô Lê Hoàng
h2) Prepare Final Test Report	21/5/2004	21/5/2004	Trần Thế Mạnh
k3) Review / Approve the Final Test Report	21/5/2004	21/5/2004	Đỗ Minh Quang

5.2. Test Tools

5.2.1. Selenium: Automation Testing Tools

Selenium is a powerful and widely-used open-source automation testing framework primarily used for web applications. It provides a suite of tools for automating web browsers across multiple platforms and supports various programming languages, including Java, Python, C#, Ruby, and JavaScript. Selenium WebDriver, the most popular component of Selenium, enables users to interact with web elements, simulate user actions, and automate testing workflows.

Key Features of Selenium:

Cross-Browser Compatibility: Selenium WebDriver allows testing across multiple browsers such as Chrome, Firefox, Safari, Edge, and Internet Explorer. This ensures that web applications behave consistently across different browser environments.

Multiple Language Support: Selenium WebDriver supports multiple programming languages, providing flexibility for testers and developers to choose the language they are most comfortable with. Popular languages include Java, Python, C#, Ruby, and JavaScript.

Element Interaction: Selenium WebDriver allows users to interact with web elements such as buttons, links, input fields, and dropdowns. It provides methods to

perform actions like clicking, typing text, selecting options, and verifying element properties.

Handling Dynamic Web Elements: Selenium WebDriver offers mechanisms to handle dynamic web elements that change state or position on the web page. It includes techniques like waiting for elements to become visible, clickable, or available before performing actions on them.

Testing Framework Integration: Selenium can be integrated with various testing frameworks such as JUnit, TestNG, NUnit, and Pytest, enabling structured and organized test automation projects. These frameworks provide features for test case management, execution, and reporting.

Parallel Test Execution: Selenium Grid allows running tests in parallel across multiple browsers and environments, reducing test execution time and improving efficiency in large-scale test automation projects.

5.2.2. TestRail: Test Case Management

TestRail is a comprehensive test management software designed to help testing teams organize their testing efforts, manage test cases, track test results, and report on testing progress. It provides a centralized platform for all testing-related activities, enabling teams to collaborate effectively and streamline their testing processes.

Key Features:

Test Case Management: TestRail allows users to create, organize, and manage test cases in a hierarchical structure. Test cases can be categorized by modules, features, or functional areas, making it easy to navigate and locate specific tests.

Test Planning and Scheduling: Teams can create test plans and schedules within TestRail, defining which tests need to be executed for each release or iteration. Test runs can be scheduled, assigned to testers, and prioritized based on business requirements.

Test Execution and Tracking: TestRail provides a user-friendly interface for executing test cases, recording test results, and tracking testing progress in real-time. Testers can mark tests as pass, fail, blocked, or untested, and add comments or attachments to provide additional context.

Defect Management Integration: TestRail seamlessly integrates with popular issue tracking systems like Jira, Bugzilla, and GitHub, allowing testers to link test results

directly to defects and vice versa. This facilitates efficient defect management and traceability between tests and issues.

Customizable Reporting: TestRail offers customizable dashboards and reports to visualize testing progress, test coverage, and defect metrics. Users can create custom reports using predefined templates or build ad-hoc reports tailored to their specific needs.

5.3. Defect Tracking

5.3.1. Reporting Bugs:

Identification: Testers or users identify bugs while using the software, through manual testing, automated testing, or user feedback.

Reproduction: Testers attempt to reproduce the bug in a controlled environment, noting the exact steps required to trigger the issue.

Description: Testers provide a clear and concise description of the bug, including:

- What behavior was expected.
- What behavior was observed.
- Any error messages or warnings.
- Environment details (e.g., operating system, browser version).

Attachments: Testers may attach screenshots, screen recordings, or logs that help developers understand and reproduce the issue.

Priority and Severity: Testers assign priority and severity levels to the bug based on its impact and urgency:

Priority: How urgently the bug needs to be fixed relative to other tasks.

Severity: How much the bug affects the functionality or usability of the software.

Reporting: Testers enter the bug details into the defect tracking system, ensuring all relevant information is provided.

5.3.2. Triaging Bugs:

Initial Assessment: A designated triage team or individual reviews newly reported bugs to validate and understand the issue.

Classification: Bugs are categorized based on factors such as their impact, area of the software affected, and frequency of occurrence.

Prioritization: Bugs are prioritized based on their severity, priority, and potential impact on users or project timelines.

Assignment: Bugs are assigned to developers or development teams for investigation and resolution. Clear ownership and accountability are established for each bug.

5.3.3. Resolving Bugs:

Investigation: Developers analyze the bug, reviewing the reported details and attempting to reproduce the issue in their development environment.

Diagnosis: Developers identify the root cause of the bug, tracing it back to the underlying code or configuration issue.

Fixing: Developers develop a fix for the bug, following coding standards and best practices. They may write new code, modify existing code, or make configuration changes.

Testing: Once the fix is implemented, developers conduct unit testing to ensure that the bug has been addressed and that no new issues have been introduced.

Verification: Testers verify the fix by retesting the affected functionality, ensuring that the bug is no longer present and that the software behaves as expected.

Validation: Testers and stakeholders review the fixed bug to validate that it meets the expected behavior and quality standards.

Closure: After validation, the bug is marked as closed or resolved in the defect tracking system. Any relevant documentation or release notes are updated to reflect the resolution.

6. Test Reporting

6.1. Test Metrics

- Schedule slippage = (Actual end date – Estimated end date) / (Planned End Date – Planned Start Date) X 100 = 1.1
- Schedule Variance = (Actual Date of Delivery – Planned Date of Delivery)
- Passed Test Cases Percentage = (Number of Passed Tests/Total number of tests executed) X 100 = 0.9

- Failed Test Cases Percentage = (Number of Failed Tests/Total number of tests executed) X 100 = 0.1
- Fixed Defects Percentage = (Defects Fixed/Defects Reported) X 100 = 0.8
- Accepted Defects Percentage = (Defects Accepted as Valid by Dev Team /Total Defects Reported) X 100 = 0.1
- Defects Deferred Percentage = (Defects deferred for future releases /Total Defects Reported) X 100 = 0.1

6.2. Test Summary Report

- Purpose of the Document

This document provides a comprehensive overview of the various activities performed during the testing phase of our application. It serves as a high-level report, summarizing the results and providing insights into the effectiveness of our testing strategies.

- Application Overview

The music application under review provides a comprehensive platform for streaming, discovering, and managing music. It offers users access to a vast library of songs across various genres, personalized playlist creation. The app is designed for a seamless user experience. This overview highlights the core functionalities and user-centric design, forming the basis for our detailed test evaluation.

- Test Environment Details

The testing was conducted on a Windows 10 operating system, with the application hosted on a local server. The application URL is internal and used for testing purposes only. We used PostgreSQL version 14.0 for the database.

- Scope of Testing

The scope of the testing effort included all major modules of the application such as User Registration, Login, Playing Track, Creating Playlist, Modifying Playlist. Due to time constraints, some minor features like user profile editing and history viewing were not tested.

- Types of Testing Performed and Test Results

Several types of software testing were performed including Functional testing, Performance testing, Usability testing, Integration testing, and Regression testing. The

majority of the tests passed successfully, with a few minor bugs identified and logged for future resolution.

- Lessons Learned Throughout Testing

Throughout the testing process, we learned the importance of early bug detection and the effectiveness of our chosen testing tools. We encountered issues with load testing which will be addressed in future testing cycles. The use of automated testing significantly improved the efficiency of our testing effort.

- Conclusion

Overall, the testing phase was successful in identifying and addressing key issues within the application. A few minor bugs remain and will be addressed in the next development cycle. Based on the test results, the software is deemed ready for release, pending resolution of the remaining issues.