Socially Anxious Hub

Test Plan

Student: Virag Szabo (4727444)

Date: Summer of 2025

Subject: Threading in C#

School: NHL Stenden

Table of contents

Contents

1 Introduction

The Test Plan for the **Socially Anxious Hub** outlines the strategy and approach for validating the functionality, performance, and reliability of the application. This document provides a structured framework for testing various components and features to ensure that the app meets the specified requirements and user expectations.

2 Objectives

The primary objectives of the Test Plan are:

- **Validate Core Functionality:** Ensure the app's essential features, including Spotify authentication, playlist management, and the memory board, work as expected.
- **Verify Cross-Platform Compatibility:** Confirm the app works seamlessly across Android and iOS devices with a responsive design.
- **Assess Usability and UX:** Evaluate the app's user experience through targeted testing to identify and resolve any usability issues.
- **Measure Performance:** Test the app's responsiveness, load times, and battery consumption to ensure a professional and smooth user experience.
- **Ensure Security:** Verify that the app adheres to security best practices, protecting user data and mitigating vulnerabilities.
- **Validate Error Handling:** Confirm the app handles errors gracefully and provides clear, informative messages.
- **Perform Comprehensive Testing:** Conduct various testing types—Unit, Integration, Performance, and User Acceptance (Alpha/Beta)—to ensure overall quality.

3 User Stories

The app's features and functionalities are driven by the following user stories:

3.1 Must-Have

Title	Description
potify Authentication	As a user, I want to log in securely with my Spotify account.
Song Search	As a user, I want to be able to search for songs.
Playlist Management	As a user, I want to create a personalized playlist (add/remove songs).
Memory Creation	As a user, I want to create a new memory item with a title and a description.
Local Data Persistence	As a user, I want my playlists and memory board data to be available again when I reopen the app.

3.2 Should-Have

Title	Description
Image Attachment	As a user, I want to attach a picture to a memory to make it more personal.
Playlist Controls	As a user, I should be able to sort my playlist by artist, title, or album.
Memory Editing	As a user, I should be able to edit or delete my memories.
Responsive Design	As a user, I should be able to use the app comfortably on different screen sizes and orientations.

3.3 Could-Have

Title	Description
Image Cropping	As a user, I could crop or resize an image before adding it to a memory.
Sentiment Analysis	As a user, I could get a simple sentiment analysis of my memory's description to understand my feelings.
Multi-User Collaboration	As a user, I will not be able to share my playlists or memories with other users in the app.

3.4 Won't-Have

Title	Description
Advanced Music Controls	As a user, I will not have access to features like music streaming or playback controls within the app.

4 Test Cases

The following test cases are categorized by testing type and functionality.

4.1 Unit Testing

Verify that individual methods and classes work as expected.

4.1.1 SpotifyService

- Verify that it returns a cryptographically random string of the correct length.
- Verify that it correctly hashes and Base64-URL-encodes the code verifier.
- Verify that it correctly deserializes a JSON string into a List<Song>.

4.1.2 DatabaseService

- Verify that a List<Song> is correctly serialized and saved to SQLite.
- Verify that it correctly retrieves and deserializes the data from the database.
- Verify that a MemoryItem is correctly saved to the database.

4.2 Integration Testing

Ensure that components work together as a cohesive whole.

4.2.1 Authentication Flow

Test Case: A user clicks the "Authenticate with Spotify" button.

Expected Result: The system browser opens, the user logs in, and the app successfully exchanges the authorization code for an access token.

4.2.2 Playlist Functionality

Test Case: A user searches for a song, adds it to the playlist, closes the app, and reopens it.

Expected Result: The song is successfully saved to the local playlist and is still present after the app restarts.

4.2.3 Memory Board Functionality

Test Case: A user creates a new memory with a title, description, and an image from their device, then closes and reopens the app.

Expected Result: The new memory, including its image, is correctly saved in the local database and loads correctly on the memory board page.

4.3 User Acceptance Testing (Alpha/Beta)

Validate that the app meets user requirements and provides a good user experience in a real-world environment.

4.3.1 Alpha Testing

Alpha Testers: A small, internal group (friends, family, classmates).

Focus on core functionality. Can testers easily log in? Do they find any crashes when using the main features? Is the UI confusing?

4.3.2 Beta Testing

Beta Testers: A larger, external group from the public.

Focus on real-world use. How does the app perform on older devices? Does it use too much battery? Is the experience smooth across different network connections?

4.4 Performance Testing

Assess the app's performance under various conditions.

4.4.1 Profiling Tools

Use Visual Studio's Diagnostic Tools or command-line tools like dotnet-trace to monitor CPU and memory usage.

Test Cases

- **Startup Time:** Measure the time it takes for the app to launch from a cold start on both Android and iOS devices.
- **UI Responsiveness:** Scroll through a long list of songs (e.g., 500+ songs) and measure if the UI is smooth and responsive.
- **API Load Time:** Measure the time it takes to get a response from the Spotify search API under normal and poor network conditions.

4.5 Security

Ensure that the app protects user data and is resistant to attacks.

4.5.1 Authentication

Test Case: Attempt to bypass the PKCE flow by intercepting the authorization code without the code verifier.

Expected Result: The token exchange fails, preventing an attacker from getting an access token.

4.5.2 Data Encryption

Test Case: Inspect the local storage on a device to confirm that sensitive information like refresh tokens are not stored in plain text.

Expected Result: All sensitive data is encrypted by SecureStorage or the SQLite database.