Test Plan

Unit Testing - GeneratePlaylist Class

Targeted Class: GeneratePlaylist
Features Tested: The recommendation of songs based on user preferences.
Test Sets/Vectors:

- Test 1: Verify that songs are recommended based on the favorite song type.
- Test 2: Verify that songs are recommended based on the favorite artist.
- Test 3: Verify that songs are recommended based on the favorite song duration.
- Test 4: Verify that songs are recommended based on the favorite song BPM.

Scope: Unit tests will focus on individual methods and attributes of the GeneratePlaylist class. These tests ensure that the class can accurately recommend songs based on user preferences.

<u>TESTS THAT PASS</u>

```
public class GeneratePlaylistTest {

    @Test 1
    public void testRecommendSongsWithValidUserPreferences() {
        GeneratePlaylist generatePlaylist = new GeneratePlaylist();
        UserPreferences userPreferences = new UserPreferences();
        userPreferences.setFavSongType("Pop");
        userPreferences.setFavArtist("Artist1");
        userPreferences.setFavSongDuration(180);
        userPreferences.setFavSongBPM(120);
     Song[] recommendedSongs =
     generatePlaylist.recommendSongs(userPreferences);

     assertTrue(recommendedSongs.length > 0, "At least one song
     should be recommended with valid user preferences.");
    }
}
```

<u>TESTS THAT FAIL</u>

```
public class GeneratePlaylistTest {

    @Test 2
    public void testRecommendSongsWithInvalidFavoriteGenre() {
        GeneratePlaylist generatePlaylist = new GeneratePlaylist();
        UserPreferences userPreferences = new UserPreferences();
        userPreferences.setFavSongType("NonexistentGenre");
     Song[] recommendedSongs =
     generatePlaylist.recommendSongs(userPreferences);
```

```
        assertEquals(0, recommendedSongs.length, "No songs should be
        recommended due to an invalid favorite genre.");
    }

    @Test 3
    public void testRecommendSongsWithInvalidUserPreferences() {
        GeneratePlaylist generatePlaylist = new GeneratePlaylist();
        UserPreferences userPreferences = new UserPreferences();
        userPreferences.setFavSongType(null);
        userPreferences.setFavArtist(null);
        userPreferences.setFavSongDuration(-1);
        userPreferences.setFavSongBPM(-1);
        Song[] recommendedSongs =
        generatePlaylist.recommendSongs(userPreferences);

        assertEquals(0, recommendedSongs.length, "No songs should be
        recommended due to invalid user preferences.");
    }
}
```

Functional Testing - SpotifyAPIManager Class

Targeted Class: SpotifyAPIManager
Features Tested: Interaction with the Spotify API for authentication, authorization, and catalog retrieval.
Test Sets/Vectors:

- Test 1: Verify that the Spotify API authentication process is initiated successfully.
- Test 2: Verify that the Spotify API authorization request is made with specified scopes.
- Test 3: Verify that songs are fetched from Spotify's catalog based on a search query.
- Test 4: Verify that a track is successfully added to the user's Spotify playlist.
- Test 5: Verify that the user's Spotify playlists are retrieved.

Scope: Functional tests ensure that the SpotifyAPIManager class can interact correctly with Spotify's APIs, including authentication, authorization, and data retrieval.

<u>TESTS THAT PASS</u>

```
public class SpotifyAPIManagerTest {

    @Test 1
    public void testFetchSpotifyCatalogWithValidQuery() {
        SpotifyAPIManager spotifyAPIManager = new
SpotifyAPIManager();
        String query = "SongName";
        SpotifyTrack[] spotifyTracks =
        spotifyAPIManager.fetchSpotifyCatalog(query);
```

```
     assertTrue(spotifyTracks.length > 0, "At least one Spotify
     track should be fetched with a valid query.");
    }

    @Test 2
    public void testAddTrackToValidSpotifyUserPlaylist() {
        SpotifyAPIManager spotifyAPIManager = new
SpotifyAPIManager();
        int playlistId = 1;  // Valid playlist ID
        int trackId = 12345;  // Valid track ID
     boolean result =
     spotifyAPIManager.addTrackToSpotifyUserPlaylist(playlistId,
     trackId);

     assertTrue(result, "Adding a track to a valid playlist should
     be successful.");
    }
}
```

TESTS THAT FAIL

```
public class SpotifyAPIManagerTest {

    @Test 1
    public void testFetchSpotifyCatalogWithInvalidQuery() {
        SpotifyAPIManager spotifyAPIManager = new
SpotifyAPIManager();
        String query = "InvalidQuery";
     SpotifyTrack[] spotifyTracks =
     spotifyAPIManager.fetchSpotifyCatalog(query);

     assertEquals(0, spotifyTracks.length, "No Spotify tracks should
     be fetched due to an invalid query.");
    }

    @Test 2
    public void testAddTrackToInvalidSpotifyUserPlaylist() {
        SpotifyAPIManager spotifyAPIManager = new
SpotifyAPIManager();
        int playlistId = -1;  // Invalid playlist ID
        int trackId = 12345;  // Valid track ID
     boolean result =
     spotifyAPIManager.addTrackToSpotifyUserPlaylist(playlistId,
     trackId);
```

```
      assertFalse(result, "Adding a track to an invalid playlist
      should fail.");
    }
}
```

System Testing - DatabaseManager Class

Targeted Class: DatabaseManager
Features Tested: Storing and retrieving user and playlist data in the database.
Test Sets/Vectors:

- Test 1: Verify that user data is successfully stored in the database.
- Test 2: Verify that user data can be retrieved from the database based on username or email.
- Test 3: Verify that playlist data is successfully stored in the database.
- Test 4: Verify that playlist data can be retrieved from the database based on the playlist identifier.
- Test 5: Verify that collaborative playlist data is successfully stored in the database.
- Test 6: Verify that collaborative playlist data can be retrieved from the database based on the playlist identifier.

Scope: System tests validate the complete functionality of the DatabaseManager class, ensuring that user and playlist data can be stored and retrieved accurately from the database.

<u>TESTS THAT PASS/FAIL</u>

```
public class DatabaseManagerTest {

    @Test 1
    public void testRetrieveNonexistentUserData() {
        DatabaseManager databaseManager = new DatabaseManager();
      UserAccount retrievedUser =
      databaseManager.getUser("nonexistent_user");

      assertNull(retrievedUser, "Retrieving nonexistent user data
      should return null.");
    }

    @Test 2
    public void testStoreUserWithDuplicateUsername() {
        DatabaseManager databaseManager = new DatabaseManager();
      UserAccount user1 = new UserAccount("duplicate_user",
      "user1@example.com", "securepass1");
      UserAccount user2 = new UserAccount("duplicate_user",
      "user2@example.com", "securepass2");
```

```
        boolean result1 = databaseManager.storeUser(user1);
        boolean result2 = databaseManager.storeUser(user2);

        assertTrue(result1, "User 1 should be successfully stored.");
     assertFalse(result2, "User 2 should fail to store due to
     duplicate username.");
    }
}
```