Unit Test Plan

In our Unit Test Plan, the classes that will be tested will be the functional classes of the system:

- HomeAudioSystemController by testing each methods of the controller (i.e. addSong)
- PersistenceHomeAudioSystem by making sure the classes are written the right way in the persistence

While the following classes won't be tested since they are the presentation and design classes:

- HomeAudioSystemPage (View)
- HAS: Song, Location, Artist, Playlist, Album, AlbumGenre (Models)
- DateLabelFormatter, InvalidInputException (helper classes)

In other words, the classes that will be tested in the Unit Test Plan will mostly be the classes in the controller and the persistence while the classes in the model and the view will not be tested. The reason is that the goal of the Unit test is to test the program components, i.e. the different functions and methods, and making sure they return the desired result using different input parameters. In our system, these methods and functions are mostly present in the controller, and most interactions are handled by the controller as well. While the view contains all the information presented to the users, it does not perform any modifications on this information. The view sends all its information to the controller, where it is processed and transferred back to the view. Therefore, the view don't need to be unit tested (the GUI will be tested in another plan). In the case of the model, it consists of classes that dictate the structure of the system, and contain its logic and rules. However, as with the view, the model which contains the information on the system's behavior is mostly modified and instantiated by the controller. It does not perform any change or provide any functionality by itself. Therefore, the controller is still in charge of the user interaction and the model does not need to be unit tested.

To derive the test cases for the Unit Tests, there are two techniques that we will be using: The Partition Testing and Guideline-based Testing. First, the Partition Testing makes group of inputs to our tests with common characteristics that will have a valid output and group of inputs that will have invalid outputs. Each group only needs tests for one input, thus saving testing time. For example, when we change the volume to a location in the application, the volume input would have to be a positive integer with a value below 100. This would represent a valid group of input, while an invalid group of inputs would be a negative integer or greater than 100. The second testing strategies, Guideline-based Testing, will consist of writing test cases based on testing guidelines reflecting common errors found by programmers such as changing the size of the collection, choosing inputs generating error messages, making the buffer overflow, test the one-value collections and so on. An example in our application would be to test the functionality of the addSong method by entering an empty string or a spacebar in the song name when deriving the test case (this should output an appropriate error message).

The tool used for the unit testing will be JUnit and PHPUnit both specialized unit testing frameworks based on xUnit. The JUnit framework gives us the ability to automate the unit testing in Java. By using an automation framework, we can create generic test classes containing test cases that can be all run automatically with a detailed report at the end detailing which methods passed or failed, and if it is the case, why it failed. This gives us the possibility to run an entire record of tests after a change has being amended in the program. The PHPUnit has exactly the same functionality as the JUnit, but in the PHP language

The test coverage of our application should be at least 90% for the classes of the controller (as explained some classes don't need to be tested). The % of test coverage for Unit Testing is the higher compared to integration testing and system testing, since Unit testing is a low level testing method. It aims for precise methods and components of different classes, thus it is easier to test precise line of codes and to simulate an error in this environment. However, it is not necessary to reach the 100% test coverage as the focus should also be put into logic and functionality testing instead of testing every lines of the source code.

These tests should be run in two occasions: When a pertinent change is brought to the program, and before the program is launched into production. Since the unit cases are automated, all the tests cases can be run rapidly. Therefore, they can be prompted after each pertinent change without creating a time issue.

Finally, the Unit Test for the desktop app and the mobile app should essentially be the same, as they use the same jar file (they have the same model and controller). However, the difference will be while testing the view (the GUI), which will be done manually, not in the Unit Test. A difference to note will be between the desktop app and the web app, since they use two different automation frameworks. Basically, the testing content will be the same, but it will be written in two different fashions.

Integration Strategy

Systems integration: the independently developed subsystems are put together to make up a complete system.

There are 3 different approaches that can be taken to test the following system:

- **Big Bang Approach:** all the subsystems are integrated at the same time
- **Bottom Up Approach**: testing is conducted from sub module to main module. If the main module is not developed a temporary program called "drivers" is used to simulate the main module.
- **Top Down Approach:** testing is conducted from main module to sub module. If the sub module is not developed a temporary program called "stub" is used for simulate the sub-module.

For our system, an incremental integration process where subsystems are integrated one at a time is the best approach. Therefore, we will not be using the Big Bang Approach. As well, since most of all of our systems depend directly on subsystems, we will be using the **Bottom Up Approach**. This integration strategy is the best suit for our system because it is especially useful if major flaws occur toward the bottom of the program. As well, the test conditions are easier to create and observing the tests is even easier.

The order in which we will be testing our classes

1. **Genre**
2. a) **Album** (Contains Genre)
   b) **Artist**
3. **Song** (Contains Album and Artist)
4. **Playlist** (Contains Song)
5. **Location** (Contains Song, Album and Playlist)
6. **HAS** (Contains the Location, Song, Album and Playlist)
7. a) **InvalidInputException**
   b) **PersistenceXStream**
8. **PersistenceHomeAudioSystem** (Contains PersistenceXStream)
9. **HomeAudioSystemController** (Contains HAS, InvalidInputException, PersistenceHomeAudioSystem)
10. **DateLabelFormatter**
11. **HomeAudioSystemPage** (Contains DateLabelFormatter)
12. **HomeAudioSytem** (Contains HomeAudioSystemPage and PersistenceHomeAudioSystem)

In addition to the unit tests, an example of a test that would be run in the integration strategy would be a persistence test. In other words, we would create a JUnit test in which we would create a song in the @Before method, test to see whether it is present in the XML file in the @Test method and later delete it in the @After method. This would be a perfect example of an integration test because it is verifying that the Song class and the Persistence classes are behaving in synchronicity. This type of test can of course be repeated for the Genre, Album, Artist, Playlist and Location classes. The test coverage of our system is essentially the effectiveness of system tests in testing the code of an entire system. Our testing goal is 85%. The tool that we will be using to support the testing will be JUnit, which is a testing framework for the Java programming language. As well, there are no discernable differences between the integration strategy of the desktop, web and mobile applications. Since all three types of applications follow near identical class diagrams and contain the same types dependencies between systems and their subsystems, the same Bottom Up Approach will be used for each type of application.

System Test Plan

Testing the system is not obvious since there could be emergent behavior when creating the system. In other words, some elements of system functionality only becomes obvious when you actually put the components together. Thus our system testing mainly focuses on testing the interactions between the components and objects that form the system and because of its focus on interactions, we developed the tests based on the sequence diagram. Furthermore, all system functions that can be accessed through the interface should be tested with correct and incorrect input to all functions to examine correct output. In this test, there might be unexpected situations where system would fail, however, our expectation is to achieve 99 % test coverage since all the methods are being tested thoroughly.

Test Situations – Features of the system to be manually tested

Description:

The following test situations describe the manual tests to be performed on the complete system. These tests apply for the three apps (desktop, mobile and web) and they are divided into pairs for the situations with correct and with incorrect inputs. The descriptions include the expected results that need to be verified in the different components of the system.

1. Adding an album to the library with all the fields (name, genre, release Date) filled correctly. After pressing the "AddAlbum" button, verify that:
    a. In the application: the album name appears in the albums spinner.
    b. In the persistence file: The album object is present with all the correct fields.

2. Adding an album to the library with one or more fields (name, genre, release Date) filled incorrectly.
    2.1. The name field is empty or just filled with spaces "   ".
    2.2. The genre is not selected.
    2.3. The date is not selected.
    After pressing the "AddAlbum" button, verify that:
    a. In the application: The correct error message is shown.
    b. In the application: the album name does not appear in the albums spinner.
    c. In the persistence file: The album object or any of its fields are not present.

3. Add an artist to the library with the name field filled. After pressing the "AddArtist" button, verify that:
    a. In the application: the artist name appears in the artists spinner.
    b. In the persistence file: The artist object is present with the artist name.

4. Add an artist to the library with the name field empty filled or only with spaces "   ". After pressing the "AddArtist" button, verify that:
    a. In the application: The correct error message is shown.
    b. In the application: the artist name does not appear in the artists spinner.
    c. In the persistence file: The artist object or its name are not present.

5. Add a song to the library with all the fields (title, duration, position in album, album, artist) filled correctly.
   After pressing the “AddSong” button, verify that:
   a. In the application: the song name appears in the songs spinner.
   b. In the persistence file: The song object is present with all the correct fields.

6. Add a song to the library with one or more fields (title, duration, position in album, album, artist) filled incorrectly.
   6.1. The title is field is empty or just filled with spaces “   “.
   6.2. The duration is not selected.
   6.3. The position in album is empty or zero.
   6.4. The album is not selected in the album spinner.
   6.5. The artist is not selected in the artist spinner.
   After pressing the “AddSong” button, verify that:
   a. In the application: The correct error message is shown.
   b. In the application: the song name does not appear in the songs spinner.
   c. In the persistence file: The song object or any of its fields are not present.

7. Add a playlist to the library with the name field filled.
   After pressing the “AddPlaylist” button, verify that:
   a. In the application: the playlist name appears in the playlists spinner.
   b. In the persistence file: The playlist object is present with the playlist name.

8. Add a playlist to the library with the name field empty filled or only with spaces “   “.
   After pressing the “AddPlaylist” button, verify that:
   a. In the application: The correct error message is shown.
   b. In the application: the playlist name does not appear in the playlists spinner.
   c. In the persistence file: The playlist object or its name are not present.

9. Add a location with the name field filled.
   After pressing the “AddLocation” button, verify that:
   a. In the application: the location name appears in the locations spinner.
   b. In the persistence file: The location object is present with the location name.

10. Add a location to the library with the name field empty filled or only with spaces “   “.
    After pressing the “AddLocation” button, verify that:
    a. In the application: The correct error message is shown.
    b. In the application: the location name does not appear in the location spinner.
    c. In the persistence file: The location object or its name are not present.

11. Add a song to an existing playlist (both song and playlist are specified)
    After pressing the “AddSongToPlaylist” button, verify that:
    a. In the persistence file: The specified song’s ID appears within the specified playlist.

12. Add a song to an existing playlist (song and/or playlist are not specified)
    After pressing the “AddSongToPlaylist” button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: The specified song’s ID does not appear within the specified playlist.

13. Assign a song to an existing location (both song and location are specified)
    After pressing the “AssignSongToLocation” button, verify that:
    a. In the persistence file: The specified song’s ID appears within the specified location.

14. Assign a song to an existing location (song and/or location are not specified)
    After pressing the “AssignSongToLocation” button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: The specified song’s ID does not appear within the specified playlist.

15. Assign an album to an existing location (both album and location are specified)
    After pressing the “AssignAlbumToLocation” button, verify that:
    a. In the persistence file: The specified album’s ID appears within the specified location.

16. Assign an album to an existing location (album and/or location are not specified)
    After pressing the “AssignAlbumToLocation” button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: The specified album’s ID does not appear within the specified playlist.

17. Assign a playlist to an existing location (both playlist and location are specified)
    After pressing the “AssignPlaylistToLocation” button, verify that:
    a. In the persistence file: The specified playlist’s ID appears within the specified location.

18. Assign a playlist to an existing location (playlist and/or location are not specified)
    After pressing the “AssignPlaylistToLocation” button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: The specified playlist’s ID does not appear within the specified playlist.

19. Clear previously assigned songs/albums/playlist from all locations (location specified)
    After pressing the “ClearAllLocations” button, verify that:
    a. In the persistence file: The specified locations do not have any song, album or playlist assigned (there is no IDs inside of the locations).

20. Clear previously assigned songs/albums/playlist from all locations (location not specified)
    After pressing the “ClearAllLocations” button, verify that:
    a. In the application: The correct error message is shown.

    b. In the persistence file: All locations have the previously assigned songs, albums or playlists within them.

21. Play currently assigned songs/albums/playlist at locations with music assigned (one or more location has music assigned)
    After pressing the "PlayAtSelectedLocations" button, verify that:
    a. In the application: Correct messages displayed, shown which songs/albums/playlist are being played in which locations.

22. Play currently assigned songs/albums/playlist at locations with music assigned (no locations have music assigned)
    After pressing the "PlayAtSelectedLocations" button, verify that:
    a. In the application: The correct error message is shown.

23. Change the volume of an existing location (location and volume specified)
    After pressing the "ChangeLocationVolume" button, verify that:
    a. In the persistence file: The volume value has been changed for the specified location.

24. Change the volume of an existing location (location and/or volume not specified)
    After pressing the "ChangeLocationVolume" button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: The volume value has not been changed for any location.

25. Mute an existing location (location specified)
    After pressing the "MuteLocation" button, verify that:
    a. In the persistence file: "isMuted" variable set to true for the specified location.

26. Mute an existing location (location not specified)
    After pressing the "MuteLocation" button, verify that:
    a. In the application: The correct error message is shown.
    b. In the persistence file: "isMuted" variable's value not changed for any location.

Detailed System Test Cases

Assign empty album to a location

1. Add an album to the library:
    a. Name and genre field must be properly filled out (cannot be empty or filled with white spaces).
    b. Release date must be picked from date picker.
    c. Press addAlbum button.
2. Verify the album has been successfully added:
    a. Check that the album exists in the album selection spinner.
    b. In the persistence file: The album object is present with all the correct fields.
3. Add a location to the library:
    a. Name field must be properly filled out (cannot be empty or filled with white spaces).
    b. Press addLocation button.
4. Verify the location has been successfully added:
    a. In the application: the location name appears in the locations spinner.
    b. In the persistence file: The location object is present with the location name.
5. Assign album to existing location:
    a. Album and location must be specified
    b. Press AssignAlbumToLocation button
6. Verify the album has not been assigned to the proper location:
    a. The application shows the proper error message.
    b. In the persistence file: The specified album's ID does not appear within the specified location.

Clear previously assigned songs/albums/playlist from all locations

1. Add an album to the library:
    a. Make sure all the fields (name, genre, release Date) are filled correctly (cannot be empty or filled with white spaces).
    b. Press the "AddAlbum" button,
2. Verify that album has been successfully added:
    a. In the application: the album name appears in the albums spinner.
    b. In the persistence file: The album object is present with all the correct fields.
3. Add a song to the library:
    a. Make sure all the fields (title, duration, position in album, album, artist) filled correctly (cannot be empty or filled with white spaces).
    b. Press the "AddSong" button
4. Verify that the song was added correctly:
    a. In the application: the song name appears in the songs spinner.
    b. In the persistence file: The song object is present with all the correct fields.

5. Add a playlist to the library:
    a. Name field must be properly filled (cannot be empty or filled with white spaces).
    b. Press the “AddPlaylist” button.
6. Verify the playlist was added correctly:
    a. In the application: the playlist name appears in the playlists spinner.
    b. In the persistence file: The playlist object is present with the playlist name.
7. Add a location to the library:
    a. Name field must be filled properly (cannot be empty or filled with white spaces).
    b. Press “AddLocation” button.
8. Verify that the location was properly added:
    a. In the application: the location name appears in the locations spinner.
    b. In the persistence file: The location object is present with the location name.
9. Add a song to an existing playlist:
    a. Both song and playlist are must be specified
    b. Pressing the “AddSongToPlaylist” button
10. Verify that the song was properly added:
    a. In the persistence file: The specified song’s ID appears within the specified playlist.
11. Add a song to an existing album:
    a. Both song and album must be specified
    b. Pressthe “AddSongToAlbum” button
12. Verify that the song was properly added:
    a. In the persistence file: The specified song’s ID appears within the specified album.
13. Assign a song to an existing location:
    a. Both song and location must be specified
    b. Press the “AssignSongToLocation” button
14. Verify that the song was properly added:
    a. In the persistence file: The specified song’s ID appears within the specified location.
15. Assign an album to an existing location:
    a. Both album and location must be specified
    b. Press the “AssignAlbumToLocation” button
16. Verify that album was properly added:
    a. In the persistence file: The specified album’s ID appears within the specified location.
17. Assign a playlist to an existing location:
    a. Both playlist and location must be specified
    b. Press the “AssignPlaylistToLocation” button
18. Verify that the playlist was properly added:

    a. In the persistence file: The specified playlist's ID appears within the specified location.

19. Clear previously assigned songs/albums/playlist from all locations:
    a. The location must specified
    b. Press the "ClearAllLocations" button
20. Verify that assigned songs/albums/playlists were properly cleared:
    a. In the persistence file: The specified locations do not have any song, album or playlist assigned (there is no IDs inside of the locations).

Task	Start Date	Expected Completion Date	isCompleted	Effort (hrs.)
Deliverable 1 – Requirements Document and Prototype				
Functional and non-functional system requirements	02/10/2016	02/12/2016	yes	n/a
Domain Model	02/13/2016	02/15/2016	yes	n/a
Use Cases	02/16/2016	02/20/2016	yes	n/a
Requirements-level sequence diagram for “Add Album” use case	02/16/2016	02/20/2016	yes	n/a
Source code of prototype implementation of “Add Album” use case on each supported platform	02/16/2016	02/20/2016	yes	n/a
Implementation-level sequence diagram for “Add Album” use case for each supported platform	02/16/2016	02/20/2016	yes	n/a
Work plan for remaining iterations	02/20/2016	02/21/2016	yes	n/a
Make sure everything is in order for submission of deliverable	02/21/2016	02/22/2016	yes	n/a
Deliverable 2 – Design Specification				
Description of architecture of proposed solution including block diagram	02/26/2016	02/29/2016	yes	3
Description of detailed design of proposed solution including class diagram	02/26/2016	02/29/2016	yes	2.5
Implement Requirement "HAS7706"	02/27/2016	03/06/2016	yes	2*
Implement Requirement "HAS7707"	02/27/2016	03/06/2016	yes	1.5*
Implement Requirement "HAS7708"	02/27/2016	03/06/2016	yes	2*
Update of work plan	03/05/2016	03/06/2016	yes	0.5
Make sure everything is in order for submission of deliverable	03/06/2016	03/07/2016	yes	0.25
Deliverable 3 – Quality Assurance Plan				
Description of Unit Test Plan	03/08/2016	03/17/2016	no	3
Description of Integration Test Plan	03/08/2016	03/17/2016	no	2
Description of System Test Plan	03/08/2016	03/17/2016	no	2
Implement Requirement "HAS7702"	03/22/2016	03/25/2016	no	1*
Implement Requirement "HAS7703"	03/10/2016	03/18/2016	no	1*
Implement Requirement "HAS7704"	03/10/2016	03/18/2016	no	2*
Implement Requirement "HAS7705"	03/10/2016	03/18/2016	no	0.5*
Implement Requirement "HAS7709"	03/10/2016	03/18/2016	no	1*
Implement Requirement "HAS7711"	03/10/2016	03/18/2016	no	2*
Update of work plan	03/19/2016	03/20/2016	no	0.5
Make sure everything is in order for submission of deliverable	03/20/2016	03/21/2016	no	0.25
Deliverable 4 – Release Pipeline Plan				
Description of release pipeline	03/22/2016	03/25/2016	no	2.5
Implement Requirement "HAS7710"	03/22/2016	03/25/2016	no	1.5*
Implement Requirement "HAS7712"	03/22/2016	03/25/2016	no	1.5*
Implement Requirement "HAS7713"	03/22/2016	03/25/2016	no	2*
Implement Requirement "HAS7714"	03/22/2016	03/25/2016	no	1.5*
Implement Requirement "HAS7715"	03/22/2016	03/25/2016	no	1*
Update of work plan	03/26/2016	03/27/2016	no	0.5
Make sure everything is in order for submission of deliverable	03/27/2016	03/27/2016	no	0.3

Deliverable 5 – Presentation				
Prepare Powerpoint presentation	03/28/2016	04/09/2016	no	4
Prepare script to say	03/28/2016	04/09/2016	no	3
Fully test application	03/28/2016	04/13/2016	no	4*
Present project	04/14/2016	04/14/2016	no	0.4
Deliverable 6 – Final Application				
Source code of full implementation on each supported platform	04/15/2016	04/15/2016	no	5*

* Hours are calculated based on an average amount of time worked on the iteration by EACH of the three (3) software developpers for the web, mobile and desktop applications