Adjustable Audio Test Plan

ECE 493 Group 5

Rizwan Qureshi, Alexandra Tyrrell and Stephen Zuk

Revision History

Name	**Date**	**Reason For Change**	**Version**

Table of Contents

Introduction

The objective of this document is to provide a detailed overview on how the testing for the Adjustable Audio software will be carried out. An analysis for each major module of this software will be carried out and a plan will be derived on how to verify its correctness. Since this is an Android based project, it is inevitable that we will use the following testing frameworks:

- RoboElectric Testing Framework
- Espresso Testing Framework

As well, we will use the UIAutomator Testing Framework to test user interactions with our app outside of the application.

Test Analysis

Audio Settings

The user is able to create, edit, or remove equalizer settings and audio balance settings. These settings can be created in two places: from the hearing test results or manually by the user in the settings fragment. Moreover these settings can be applied to two types of audio playback devices: the media player and the microphone player. As such, the audio settings is the most pivotal component of the software since almost every feature relates to it in some form.

The following functionality of the audio settings are expected:

ID	Functionality and Analysis	Testing Required? (Y/N)
1	Saving an equalizer preset into storage. Tests: - Verify that the equalizer preset file exists in the storage. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SaveController - EqualizerPresetListController - Saver - EqualizerPreset - EqualizerModel Covered in the following tests:	Y

	- Covered by the Espresso test ("AddEqualizerPreset" file) - Covered by the Unit test("savePresetTest") - Covered by the Unit test("loadPresetTest") - Covered by the Unit test("updatePresetTest") - Covered by the Unit test("deletePresetTest")	
2	Loading an equalizer preset from storage. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SaveController - EqualizerPresetListController - Saver - EqualizerPreset - EqualizerModel Tests: - Verify that if no equalizer preset file exists, a default file is created and loaded. - Verify that if an equalizer preset file exists, it is loaded into a valid equalizer preset object. - A valid equalizer preset object has: - a name that is not an empty string. - 5 equalizer band settings where the frequency is >= 30 Hz and <= 16000 Hz and the decibels are in the range [-15 dB, 15dB] - A leftRightRatio that is in the range (0, 10000000) Covered in the following tests: - Covered by the Espresso test ("RemoveEqualizerPreset") - Covered by the Espresso test ("EditEqualizerPreset") - Covered by the Unit test("savePresetTest") - Covered by the Unit test("loadPresetTest") - Covered by the Unit test("updatePresetTest") - Covered by the Unit test("deletePresetTest")	Y
3	Converting an equalizer preset into a JSON object. Tests: - Verify that the JSON object conforms to the following	Y

	schema: - The name key exists and maps to a string - The lightRightRatio key exists and maps to an integer - The equalizerSettings key exists and maps to a dictionary of int to ints. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - EqualizerPreset Covered in the following tests: - Covered by the Espresso test (“AddEqualizerPreset”) - Covered by the Unit test (“toJsonEqualizerPreset”)	
4	Converting a JSON object into an equalizer preset. Tests: - Verify that deserializing the JSON creates a valid equalizer preset object. - A valid equalizer preset object has: - a name that is not an empty string. - 5 equalizer band settings where the frequency is >= 30 Hz and <= 16000 Hz and the decibels are in the range [-15 dB, 15dB] - A leftRightRatio that is in the range (0, 10000000) Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - EqualizerPreset Covered in the following tests: - Covered by the Espresso test (“RemoveEqualizerPreset”) - Covered by the Espresso test (“EditEqualizerPreset”) - Covered by the Unit test (“fromJsonEqualizerPreset”)	Y
5	Encrypting the equalizer preset.	Y

	Tests: - Verify that deserializing the encrypted JSONized equalizer preset is not possible (eg. triggers an exception). Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Encrypter - Jsonizer - EqualizerPreset Covered in the following tests: - Covered by the Espresso test (“AddEqualizerPreset”) - Covered by the Unit test (“encryptEqualizerPresetTest”)	
6	Decrypting the equalizer preset. Tests: - Verify that deserializing the decrypted JSONized equalizer preset is possible (eg. does not trigger and exception). Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Encrypter - Jsonizer - EqualizerPreset Covered in the following tests: - Covered by the Espresso test (“RemoveEqualizerPreset”) - Covered by the Espresso test (“EditEqualizerPreset”) - Covered by the Unit test (“decryptEqualizerPresetTest”)	Y
7	Modifying the frequency band of an equalizer preset. Tests: - Verify that a frequency band in a EqualizerPreset can be set. - Verify that a frequency band in a AudioData can be set. Test Criteria: - Statement coverage will be performed for the following	Y

	classes to test this functionality: - EqualizerPreset - EqualizerModel - AudioData Covered in the following tests: - Covered in Espresso Test (located in "SettingsFragmentTest") - Covered in Espresso Test (located in "EditEqualizerPresetTest")	
8	Modifying the audio balance ratio of an equalizer preset. Tests: - Verify that the audio balance ratio can be set for a EqualizerPreset. - Verify that the audio balance ratio can be set for a AudioData. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - EqualizerPreset - EqualizerModel - AudioData Covered in the following tests: - Covered in Espresso Test (located in "SettingsFragmentTest")	Y
9	Applying an equalizer preset to an audio device (such as the media player and microphone player). This means that the audio device reflects the settings provided from the preset, such as the equalizer bands and audio balance. Tests: - Verify that the AudioController causes the MediaPlayerAdapter to set the equalizer setting. - Verify that the AudioController causes the MicrophonePlayerAdapter to set the equalizer setting. - Verify that the AudioController causes the MediaPlayerAdapter to set the audio balance. - Verify that the AudioController causes the MicrophonePlayerAdapter to set the audio balance.	Y

	Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - AudioController - IAudioDevice (MicrophonePlayerAdapter, MediaPlayerAdapter, MusicServiceInteractor, MicrophoneServiceInteractor)	
10	In the Settings fragment, allow viewing and modifying the equalizer settings using a 5 band graphic equalizer existing on the UI. Tests: - Verify that changing the decibel value of a band causes the corresponding band in the equalizer preset object to change. - Verify that the UI reflects the equalizer band decibel values from the equalizer preset object. - Verify that it is not possible to set decibel values outside of the range [-15dB, 15dB]. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - Equivalence Partitioning will be used to ensure that no decibel value outside of the range [-15dB, 15dB] can be set by the UI. Covered in the following tests: - Covered in Espresso Test (located in “SettingsFragmentTest”)	Y
11	In the Settings fragment, allow viewing and modifying the audio balance ratio using a seekbar. Tests: - Verify that the audio balance ratio is presented as percentage. - Verify that changing the audio balance using a seek bar will convert the result into a left-right ratio in the range (0, 1000000) and store it in the equalizer preset object. Test Criteria: - Statement coverage will be performed for the following	Y

	classes to test this functionality: - SettingsFragment - Equivalence Partitioning will be used to ensure that no audio balance ratio outside of the range [0, 1000000] can be set by the UI. This means that the left and right audio percentage must remain in the range [0, 100%] Covered in the following tests: - Covered in Espresso Test (located in "SettingsFragmentTest")	
12	In the Media Player fragment, allow viewing and modifying the audio balance ratio using a seekbar. Tests: - Verify that the audio balance ratio is presented as percentage. - Verify that changing the audio balance using a seek bar will convert the result into a left-right ratio in the range (0, 1000000) and store it in the equalizer preset object. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - Equivalence Partitioning will be used to ensure that no audio balance ratio outside of the range [0, 1000000] can be set by the UI. This means that the left and right audio percentage must remain in the range [0, 100%] Covered in the following tests: - Covered in Espresso Test (located in "MediaPlayerFragmentTest")	Y
13	In the Settings fragment, allow switching between different equalizer presets. Tests: - Verify that any unsaved changes to the previous equalizer preset are not persisted. - Verify that the preset name is changed to new preset's name. - Verify that the UI's equalizer band values are the same as the preset's band value. - Verify that the UI's audio balance ratio in percent form	Y

	is equivalent to the preset's audio balance ratio. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel - EqualizerPreset Covered in the following tests: - Covered in Espresso Test ("EditEqualizerPresetTest")	
14	In the Settings fragment, display the name of the currently selected equalizer preset. Tests: - Verify that the text displayed in UI is equivalent to the equalizer preset's name. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel - EqualizerPreset Covered in the following tests: - Covered in Espresso Test ("SettingsFragmentTest") - Covered in Espresso Test ("EditEqualizerPresetTest")	Y
15	In the Settings fragment, allow the creation of an equalizer preset. Tests: - Verify that the newly created equalizer preset will persisted (eg. the equalizer preset can be loaded from storage). - Verify that the attributes of the new equalizer preset are valid: - The name is not empty - There are 5 band frequencies with decibel values between [-15 db, 15dB] - The leftRightRatio is between [0, 1000000] Test Criteria:	Y

	- Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel - EqualizerPreset Covered in the following tests: - Covered in Espresso Test (“AddEqualizerPresetTest”)	
16	In the Settings fragment, allow the renaming of an equalizer preset. Tests: - Verify that the new name is not an empty string. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel - EqualizerPreset - Equivalence Partitioning will be used to verify that the length of the name is > 0. Covered in the following tests: - Covered in Espresso Test (“EditEqualizerPresetTest”)	Y
17	In the Settings fragment, allow the deleting of an equalizer preset. Tests: - Verify that the current equalizer preset is not the default preset. - Verify that the previous equalizer preset is loaded after deletion. - Verify that the UI shows the previous equalizer preset. - Verify that the deleted equalizer preset is persisted (eg. loading the deleted equalizer preset from the storage is not possible) Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel	Y

	- SaveController - Save - Jsonizer - Encryptor - EqualizerPresetListController.java - EqualizerPreset - Equivalence Partitioning will be used to verify that the length of the equalizer preset list is always >= 1. Covered in the following tests: - Covered in Espresso Test (“RemoveEqualizerPresetTest”)	
18	In the Settings fragment, allow reverting changes of an unsaved equalizer preset. Tests: - Verify that the equalizer preset object has the same attributes of the persisted equalizer preset object. - Verify that the UI displays the reverted preset. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SettingsFragment - EqualizerModel - SaveController - Save - Jsonizer - Encryptor - EqualizerPresetListController.java - EqualizerPreset - Equivalence Partitioning will be used to verify that the length of the equalizer preset list is always >= 1. Covered in the following tests: - Covered in Espresso Test (“EditEqualizerPresetTest”)	Y

Hearing Test

The user is able to perform hearing tests which can generate hearing test results. The hearing test results can be viewed as an audiogram to help the user evaluate their hearing. These hearing test results can also be converted into equalizer settings. Additionally the audiogram may be forwarded over email or SMS. The hearing test will evaluate hearing of both ears by recording the dBHL values the user reacts to for varying frequencies. The Hearing Test component has the least amount of interaction with other components in the system.

The following functionality of the hearing test component are expected:

ID	Functionality and Analysis	Testing Required? (Y/N)
19	Saving hearing test results into storage. Tests: - Verify that the hearing test results file exists in the storage. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SaveController - HearingTestResultListController - Saver - HearingTestResult - ToneData Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test) - Covered by “saveResultTest” (Unit Test) - Covered by “loadResultTest” (Unit Test) - Covered by “deleteResulTest” (Unit Test)	Y
20	Loading hearing test results from storage. Tests: - Verify that if no hearing test results file exists, a default file is created and loaded. - Verify that if a hearing test results file exists, it is loaded into a valid hearing test result object. - A valid hearing test result object has: - a name that is not an empty string. - A date - A list of tone data containing: - The frequency of the tone >= 30 and <= 16000 - The decibel value of the tone >= -5 and <= 100	Y

	Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - SaveController - HearingTestResultListController - Saver - HearingTestResult - ToneData Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test) - Covered by “HearingTestResultFragmentTest” (EspressoTest) - Covered by “saveResultTest” (Unit Test) - Covered by “loadResultTest” (Unit Test) - Covered by “deleteResulTest” (Unit Test)	
21	Converting a hearing test result into a JSON object. Tests: - Verify that the JSON object conforms to the following schema: - The testName key exists and maps to a non empty string - The testResults key exists and maps to a list of dictionaries - Each dictionary contains a frequency key mapping to a number, an lHeardAtDB key mapping to a number, an rHeardAtDB key mapping to a number, and a dBHL key mapping to a number. - The testDate key mapping to a date. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - HearingTestResultListController - HearingTestResult - ToneData Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test)	Y

	- Covered by “toJsonHearingTestResult” (Unit Test)	
22	Converting a JSON object into a hearing test result. Tests: - Verify that deserializing the JSON creates a hearing test result object. - A valid hearing test result object has: - a name that is not an empty string. - A date - A list of tone data containing: - The frequency of the tone >= 30 and <= 16000 - The decibel value of the tone >= -5 and <= 100 Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - HearingTestResultListController - HearingTestResult - ToneData Covered in the following tests: - Covered by “HearingTestResultFragmentTest” (EspressoTest) - Covered by “fromJsonHearingTestResult” (Unit Test)	Y
23	Encrypting the hearing test results. Tests: - Verify that deserializing the encrypted JSONized hearing test result is not possible (eg. triggers an exception). Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - Encrypter - HearingTestResultListController - HearingTestResult - ToneData	Y

	Covered in the following tests: - Covered by "HearingTestActivity" (Espresso Test) - Covered by "encryptHearingTestResult" (Unit Test)	
24	Decrypting the hearing tests results. Tests: - Verify that deserializing the decrypted JSONized hearing test result is possible (eg. does not trigger and exception). Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - Jsonizer - Encrypter - HearingTestResultListController - HearingTestResult - ToneData Covered in the following tests: - Covered by "HearingTestResultFragmentTest" (EspressoTest) - Covered by "decryptHearingTestResult" (Unit Test)	Y
25	Converting hearing test results into equalizer and audio balance settings. Tests: - Verify that resulting equalizer preset object is valid: - A valid equalizer preset object has: - a name that is not an empty string. - 5 equalizer band settings where the frequency is >= 30 Hz and <= 16000 Hz and the decibels are in the range [-15 dB, 15dB] - A leftRightRatio that is in the range (0, 10000000) - Verify that the calculation does not cause a divide by 0 exception when the mean decibel value is 0. - Verify that a toast appears confirming that the equalizer preset was generated. - Verify that the new equalizer preset is persisted with the same name as the hearing test result.	Y

	Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestResult - EqualizerPreset - SaveController - ToneData - HearingTestResultFragment - Branch coverage will be performed for the HearingTestResultFragment.generateEqualizerPreset() method to test edge cases such as dividing by 0. Covered in the following tests: - Covered by “HearingTestResultFragmentTest” (EspressoTest)	
26	Forwarding the hearing tests results as an audiogram to another app (such email and MMS). Tests: - Verify that the UI can create an intent to forward the audiogram. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestResult - HearingTestResultFragment Covered in the following tests: - Covered by “HearingTestResultFragmentTest” (EspressoTest)	Y
27	Viewing the hearing test results as an audiogram. Tests: - Verify that a hearing test result can be generated into an audiogram. - Verify that the HearingTestResult fragment UI is displaying the audiogram. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality:	Y

	- HearingTestResult - ToneData - HearingTestResultFragment Covered in the following tests: - Covered by “HearingTestResultFragmentTest” (EspressoTest)	
28	Deleting a hearing test result. Tests: - Verify that deleting a hearing test result is persisted (eg. reloading the hearing test results will not contain the deleted hearing test result.) - Verify that HearingTestResult UI allows deletion of the hearing test result. - Verify that deleting a hearing test result under the HearingTestResult fragment causes the UI to return to the HearingTestList UI. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestResultFragment - HearingTestFragment - SaveController - HearingTestResultListController Covered in the following tests: - Covered by “DeleteHearingTestResultTest” (EspressoTest)	Y
29	Renaming a hearing test result. Tests: - Verify that renaming a hearing test causes the HearingTestResult object’s name attribute to be changed to the new name. - Verify that the new name is persisted. - Verify that the new name is reflected by the HearingTestResult UI. - Verify that the new name is a non empty string. Test Criteria: - Statement coverage will be performed for the following	Y

	classes to test this functionality: - HearingTestResultFragment - SaveController - HearingTestResultListController - Equivalence Partitioning will be used to verify that only names with length > 0 are acceptable. Covered in the following tests: - Covered by “HearingTestResultFragmentTest” (EspressoTest)	
30	Viewing the list of hearing test results generated from previous hearing tests. Tests: - Verify that the HearingTestList UI displays a list of hearing tests results with the name and date of each hearing test result (if the list is non empty). - Verify that selecting a hearing test result in the HearingTestList UI will display the HearingTestResult UI. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestFragment - HearingTestResult Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test)	Y
31	Viewing instructions on performing the hearing test. Tests: - Verify that instructions for performing the UI are displayed on the HearingTest UI. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test)	Y

32	Generating tones for a hearing test. The tone will can be targeted to a specific ear (eg. left or right). The following tones must be generatable (in Hz): 30, 90, 233, 250, 347, 500, 907, 1000, 1353, 2000, 3533, 4000, 5267, 8000, 11333, 15667. Tests: - Verify that the above listed tones are generatable. - Verify that the hearing test will generate the above listed tones for both ears. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel Covered in the following tests: - Covered by “generateSamplesTest” (Unit Test)	Y
33	Tone volume is incremented by 5dB every two seconds when a Hearing Test occurs. Tests: - Verify that the tone volume is incremented by 5dB every two seconds. - Verify that the tone volume does not increment beyond 100 dbHL. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel - Boundary-value analysis will be used to verify that the tone volume does not increment past 100 dBHL and moves to the next frequency instead. Covered in the following tests: - Covered by “hearingTestStateTest”,	Y

	“hearingTestVolumeMaxTest”, “hearingTestVolumeSetTest” (Unit Tests)	
34	Tone volume resets to -5 dBHL whenever the frequency is changed during a hearing test. Tests: - Verify that the tone volume is initially -5 dbHL whenever the frequency is changed. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel - Boundary-value analysis will be used to verify that the tone volume is initialized at -5 dbHL at the start of every frequency test. Covered in the following tests: - Covered by “hearingTestVolumeMaxTest”, “hearingTestVolumeSetTest” (Unit Tests)	Y
35	Tone volume and frequency is recorded when the tone sound is acknowledged. Tests: - Verify that whenever a sound acknowledgment is triggered the ToneData is recorded. - Verify that the ToneData is valid: - The decibel values are between 0 and 100 dbHL - The frequency value is between 30 and 16000 Hz. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel Covered in the following tests:	Y

36	Tone frequency is changed to the next (higher) frequency, if possible, when the tone sound is acknowledged. Tests: - Verify that the frequency is increased after a sound acknowledgment. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel Covered in the following tests: - Covered by “hearingTestVolumeMaxTest”, “hearingTestVolumeSetTest” (Unit Tests)	Y
37	The hearing test is performed for both ears. Tests: - Verify that on the completion of a hearing test, 16 ToneData entries exist and each ToneData has a valid decibel value for both ears. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test) - Covered by “hearingTestStateTest” (Unit Test)	Y
38	The hearing test result is stored when the hearing test is completed and confirmed. Tests: - Verify that the HearingTest UI requests the name of the hearing test result upon the completion of the test. - Verify that the hearing test result is persisted with the entered name, the current date, and the 16 ToneData	Y

	entries. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - HearingTestActivity - ToneData - ToneGenerator - HearingTestModel - Equivalence partitioning and Boundary value analysis is used to ensure the length of the name is > 0. Covered in the following tests: - Covered by “HearingTestActivity” (Espresso Test)	

<u>Media Player</u>

The media player allows the user to play audio files. The audio files must first be queued into the playlist before it can be played. The user can switch between and remove audio files in the playlist. Additionally, the media player can play the audio files in the background and control the state using a navigation bar.

The following functionality of the media player is expected:

ID	Functionality and Analysis	Testing Required? (Y/N)
39	From the Media Player fragment, allow browsing for an audio file to add to the playlist. Tests: - Verify that the MediaPlayer UI contains a button to add more songs. - Verify that the MediaPlayer UI launches a file browser intent when the add song button is pressed. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - Song Covered in the following tests: - Covered in “MediaQueueViewTest” (Espresso Test)	Y

40	From the Media Player fragment, allow adding an audio file to the playlist. Tests: - Verify that if the Add Song Intent returns a song, the song is added to the playlist in the MusicService and the song is visible in the MediaPlayer UI. - Verify that if the added song is the only song in the playlist, it is automatically selected as the current song on the MediaPlayer UI and in the MusicService. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - MusicService - MusicServiceInteractor - MediaPlayerAdapter - MediaData - Song - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaQueueViewTest” (Espresso Test)	Y
41	From the Media Player fragment, allow removing an audio file from the playlist from the Media Player fragment. Tests: - Verify that removing a song from the playlist will remove it from both the MediaPlayer UI and the MusicService. - Verify that if the currently selected song is removed, the next song is selected. If there is no next song, the previous song is selected. If there is no previous song, the playlist is empty and no metadata of the song is displayed over the MediaPlayer UI. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment	Y

	- MusicService - MusicServiceInteractor - MediaPlayerAdapter - MediaData - MediaPlayerQueueAdapter - Song - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaQueueViewTest” (Espresso Test)	
42	From the Media Player fragment. allow playing an audio file in the playlist Tests: - Verify that if no song is playing from the MusicService, the play button is visible on the MediaPlayer UI. - Verify that if the play button is pressed on the MediaPlayer UI when no song is selected, nothing happens. - Verify that if the play button is pressed on the MediaPlayer UI when a song is selected, the MusicService will play that song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - MusicService - MusicServiceInteractor - MediaPlayerAdapter - MediaData - Song - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaPlayerFragmentTest” (Espresso Test) - Covered in “MediaQueueViewTest” (Espresso Test)	Y
43	From the Media Player fragment, allow pausing an audio file that is currently playing	Y

	Tests: - Verify that when the MusicService is playing a song, the MediaPlayer UI displays the pause button. - Verify that when the pause button is pressed on the MediaPlayer UI, the music stops playing. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - MusicService - MusicServiceInteractor - MediaPlayerAdapter - Song - MediaData - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaPlayerFragmentTest” (EspressoTest) - Covered in “MediaQueueViewTest” (Espresso Test)	
44	From the Media Player fragment, allow skipping to the next audio file in the playlist Tets: - Verify that pressing the skip to next button, when no next song exists in the playlist, will cause nothing to happen. - Verify that pressing the skip to next button from the MediaPlayer UI, when a next song exists in the playlist, the next song will be selected both in MediaPlayer UI and the MusicService. If the MusicService is currently playing a song, it will also play the next song automatically. Otherwise it will pause at the beginning of the next song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - MusicService	

	- MusicServiceInteractor - MediaPlayerAdapter - MediaData - Song - MediaPlayerQueueAdapter - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaPlayerFragmentTest” (Espresso Test) - Covered in “MediaQueueViewTest” (Espresso Test)	
45	From the Media Player fragment, allow skipping to the previous audio file in the playlist Tests: - Verify that pressing the skip to previous button, when no previous song exists in the playlist, will cause nothing to happen. - Verify that pressing the skip to previous button from the MediaPlayer UI, when a previous song exists in the playlist, the previous song will be selected both in MediaPlayer UI and the MusicService. If the MusicService is currently playing a song, it will also play the previous song automatically. Otherwise it will pause at the beginning of the previous song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MediaPlayerFragment - MusicService - MusicServiceInteractor - MediaPlayerAdapter - MediaData - Song - MediaPlayerQueueAdapter - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaPlayerFragmentTest” (EspressoTest)	

	- Covered in “MediaQueueViewTest” (Espresso Test)	
46	From the Notification, allow skipping to the next audio file in the playlist Tets: - Verify that no next button exists in Notification UI, when no next song exists in the playlist. - Verify that pressing the skip to next button from the Notification UI, when a next song exists in the playlist, the next song will be selected in the MediaPlayer UI, Notification UI and the MusicService. If the MusicService is currently playing a song, it will also play the next song automatically. Otherwise it will pause at the beginning of the next song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MusicNotificationManager - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in MusicNotificationTest (Espresso and UIAutomator)	Y
47	From the Notification, allow skipping to the previous audio file in the playlist Tests: - Verify that no previous button exists in Notification UI, when no previous song exists in the playlist. - Verify that pressing the skip to previous button from the Notification UI, when a previous song exists in the playlist, the previous song will be selected in the MediaPlayer UI, Notification UI and the MusicService. If the MusicService is currently playing a song, it will also play the previous song automatically. Otherwise it will pause at the beginning of the previous song.	Y

	Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MusicNotificationManager - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in MusicNotificationTest (Espresso and UIAutomator)	
48	From the Notification, allow playing of an audio file in the playlist. Tests: - Verify that when no song is playing from the MusicService, the Notification UI displays the play button. - Verify that when the play button is pressed from the Notification UI, it causes the MusicService to play the song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MusicNotificationManager - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in MusicNotificationTest (Espresso and UIAutomator)	Y
49	From the Notification, allow pausing of an audio file that is	Y

	currently playing. Tests: - Verify that when a song is playing from the MusicService, the Notification UI displays the pause button. - Verify that when the pause button is pressed from the Notification UI, it causes the MusicService to pause the song. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MusicNotificationManager - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in MusicNotificationTest (Espresso and UIAutomator)	
50	In the Media Player fragment, display the title and artist of the audio file that is currently selected in the playlist. Tests: - Verify that the MediaPlayer UI displays the song title and artist if a song is currently selected. - Verify that if no song is selected, the MediaPlayer UI displays a blank field for the song title and artist. - Verify that the song title and artist displayed on the MediaPlayer UI matches the song title and artist of the currently played song from the MusicService. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData	Y

	- Song - MediaPlayerFragment - State transition table shown in Figure 1 under the Appendix will be used to test all the acceptable states of the MediaPlayer. Covered in the following tests: - Covered in “MediaQueueViewTest” (Espresso Test)	
51	In the Media Player fragment, display the elapsed and total duration of the currently selected audio file in the playlist. The elapsed duration should update when an audio file is playing. Tests: - Verify that when a song is playing from the MusicService, the elapsed duration MediaData is incremented every second. - Verify that when the MediaData’s elapsed duration changes, the MediaPlayer UI displays the elapsed and total duration of the song in the form (HH:MM:SS / HH:MM:SS). Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MediaPlayerFragment - Boundary Value Analysis will be used to ensure that the HH:MM:SS format is correctly displayed when the duration is 9 seconds, 60 seconds, and 3600 seconds. Covered in the following tests:	Y
52	In the Media Player fragment, the elapsed duration and total duration is reflected by a seekbar. The seekbar should update its position when an audio file is playing. Tests: - Verify that when the MediaData’s elapsed duration changes, the MediaPlayer UI updates the seekbar with the new progress. - Verify that when a song completes, the seekbar is reset	Y

	to the beginning. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MediaPlayerFragment Covered in the following tests:	
53	In the Media Player fragment, allow seeking to another duration position of the currently selected audio file using the seekbar. Tests: - Verify that the MediaPlayer UI does not allow seeking outside of the range (0, total duration of song). - Verify that the MediaPlayer allows seeking to a specific position in the song, when a song is selected. - Verify that when a song is playing and a seek is performed, the song continues to play. Otherwise the song does not play after the seek is performed. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MediaPlayerAdapter - MediaData - Song - MediaPlayerFragment - Equivalence Partitioning will be used to ensure that it is impossible to seek before 0 seconds or past the duration of the song. Covered in the following tests: - Covered by “MediaQueueViewTest” (Espresso Test)	Y
54	The Media Player should remain active as a background service. This means that the audio file should continue to play in the background while the app is not open. Tests:	Y

	- Verify that when the MusicService is playing a song, it can be heard in the background while the app is not running in the foreground. - Verify that when the song playlist is nonempty, and the app is running in the background, the Notification is visible. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MusicService - MainActivity - MediaPlayerAdapter Covered in the following tests: - Covered in “MusicNotificationTest “ (Espresso Test)	

<u>Microphone Player</u>

The microphone player will playback the microphone input of the device.

The following functionality is expected from the microphone player.

ID	Functionality and Analysis	Testing Required? (Y/N)
55	Only one of the following modes can be selected at a time: *Normal, Speech Focus, Noise Suppression* Tests: - Verify that only one microphone mode can be selected at any time in the MicrophoneService and Microphone UI. - Verify that when the MicrophoneService changes the mode, the Microphone UI will highlight the new mode that is selected. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService - MicrophonePlayerAdapter - MicrophoneData	Y

	Covered in the following tests: - Covered by “MicrophoneFragmentTest” (Espresso) → only asserts if a mode is selected or not	
56	When the microphone is enabled in Normal mode, the microphone playback should be streamed in real time. Tests: - Verify that the microphone playback is streamed, unfiltered, if the current mode is Normal and the microphone is enabled. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService - MicrophonePlayerAdapter - MicrophoneData Covered in the following tests:	Y
57	When the microphone is enabled in Noise Suppression mode, the echoing and background noise should be filtered from the microphone playback stream. Tests: - Verify that the microphone playback is streamed, with the noise and acoustic echoing filtered out, when the current mode is Noise Suppression and the microphone is enabled. - Verify that if Noise Suppression is not supported, the Microphone UI displays a toast notifying that this is the case. - Verify that if Noise Suppression is not supported by the device, it cannot be selected. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService	N

	- MicrophonePlayerAdapter - MicrophoneData This is not testable due to insufficient testing tools. No emulator seems to exist with support of noise suppression.	
58	When the microphone is enabled in Speech Focus mode, the microphone playback stream will be active when a voice is captured. Tests: - Verify that the microphone playback is streamed when voice is heard and the current mode is Speech Focus, and the microphone is enabled. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService - MicrophonePlayerAdapter - MicrophoneData - Boundary Value Testing will be used to verify voice activity threshold. Covered in the following tests:	Y
59	The Microphone Player should remain active as a background service. This means that the microphone should continue to playback in the background while the app is not open. Tests: - Verify that the microphone playing while the app is in the background and the microphone is enabled. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneService - MicrophonePlayerAdapter Covered in the following tests:	Y
60	The Microphone fragment allows a user to switch between the	Y

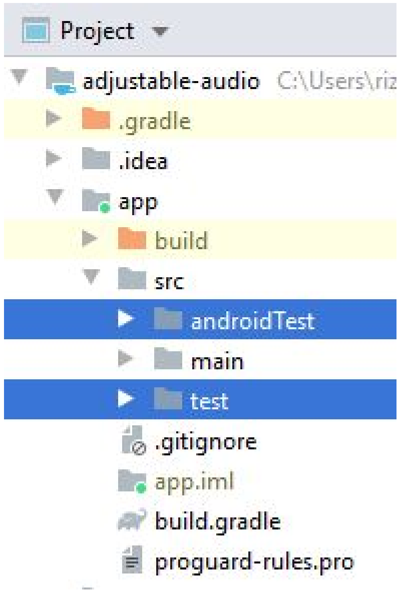
	modes. Tests: - Verify that when a mode is selected from the Microphone UI, the same mode is selected in the MicrophoneService. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService - MicrophonePlayerAdapter - MicrophoneData Covered in the following tests: - Covered by “MicrophoneFragmentTest” (Espresso) → only asserts if a mode is selected or not	
61	The Microphone fragment allows a user to enable and disable the playback of the microphone input. Tests: - Verify that when the microphone is disabled, the microphone button is not highlighted in the Microphone UI. - Verify that when the microphone is enabled, the microphone button is highlighted in the microphone UI. Test Criteria: - Statement coverage will be performed for the following classes to test this functionality: - MicrophoneFragment - MicrophoneServiceInteractor - MicrophoneService - MicrophonePlayerAdapter - MicrophoneData Covered in the following tests: - Covered by “MicrophoneFragmentTest” (Espresso) → only tests if the button is toggled or not	Y

Instructions Running the Test Suite

Android Studio is required for running the test suite. All test suites are available under the Departmental git server, along with the Adjustable Audio Android project. Clone the project into a local directory and open the project via Android Studio.

Before you can run the test, an Android emulator is required. We recommend the Pixel 2 emulator. In addition to the emulator, the test requires at least 1 song to exist on the emulator. It is possible to drag and drop an audio file (.mp3 is preferrable) into the emulator from your own machine. Otherwise you can download it directly from the emulator's web browser. Any audio file will be acceptable for performing the test.

There are two main categories of tests: Unit tests, and Android tests. Unit tests (meant for Java independent class tests) can be located under the test directory while Android test (meant for packages requiring Android) can be located in the androidTest directory as shown below:



You can run these tests by right clicking the test directory and hitting the play button. For further details, please refer to:

https://developer.android.com/studio/test#run_a_test

Appendix

Figure 1 - State-Transition Table for Media Player

State	Has Next Song?	Has Previous Song	Has Current Song?	Is Playing?

s1	0	0	0	0
s2	0	0	1	0
s3	0	1	1	0
s4	1	0	1	0
s5	1	1	1	0
s7	0	0	1	1
s8	0	1	1	1
s9	1	0	1	1
s10	1	1	1	1

Input	**Current State**	**Next State**	**Output**
Play	s1	s1	None
Add song	s1	s2	Song added to playlist and is selected
Remove song	s2	s1	Song removed from playlist. Nothing is selected.
Play	s2	s7	Current Song is played.
Add song	s2	s4	Song is queued to playlist.
Pause	s7	s2	Current song is paused
Remove song	s7	s1	Song removed from playlist. Nothing is selected. Song stops playing.
Add song	s7	s9	Song is queued to playlist.

Skip to next song	s4	s3	Next song is selected.
Skip to previous song	s3	s4	Previous song is selected.
Add song	s3	s5	Song is queued to playlist.
Remove song	s3	s2	Select previous song
Remove song	s4	s2	Select next song
Remove song	s5	s3	Select next song
Skip to next song	s9	s8	Next song is selected. Next song automatically plays.
Skip to previous song	s8	s9	Previous song is selected. Previous song automatically plays.
Add song	s8	s10	Song is queued to playlist.
Remove song	s8	s7	Select previous song. Previous song automatically plays.
Remove song	s9	s7	Select next song. Next song automatically plays.
Remove song	s10	s8	Select next song. Next song automatically plays.
Song completes	s7	s2	Song no longer playing.
Song completes	s8	s3	Song no longer playing.
Song completes	s9	s8	Play next song.
Song completes	s10	s10	Play next song.