# Overall Test Plan

Our app has 3 different components that need to communicate with each other: the app, the server, and the player. Since most of our functionality lies in the server, the majority of our tests are in that area. The server's endpoints are tested using a program called Postman, which allows you to easily make and send http requests. The player has a built-in cli that allows for manually inputting commands for testing. The mobile app must be tested by performing normal app functions, like searching and logging in.

# Test Case Descriptions

S1.1 **Server Test 1**
S1.2 This test will ensure that the server is able to start normally.
S1.3 This test will start the server then run a simple health check web query against it to ensure that the client-side web server has started.
S1.4 Inputs: None
S1.5 Outputs: An echo response
S1.6 Normal
S1.7 Whitebox
S1.8 Functional
S1.9 Integration

S2.1 **Server Test 2**
S2.2 This test will ensure that the server is able to authenticate the user normally with a correct password.
S2.3 The user will use a supported authentication mechanism to authenticate with the admin page.
S2.4 Inputs: Password or other supported authentication mechanism.
S2.5 Outputs: An authentication token and ability to edit administrator details.
S2.6 Normal
S2.7 Blackbox
S2.8 Functional
S2.9 Unit

S3.1 **Server Test 3**
S3.2 This test will ensure that the server is able to deny authentication to the user with an incorrect password.
S3.3 The user will use a supported authentication mechanism to authenticate with the admin page with an incorrect password. The server will deny the authentication.
S3.4 Inputs: Incorrect password or other supported authentication mechanism.
S3.5 Outputs: An error
S3.6 Abnormal
S3.7 Blackbox
S3.8 Functional
S3.9 Unit

S4.1 **Server Test 4**
S4.2 This test will ensure that the server is able to return to a client the settings page.
S4.3 A correctly authenticated user will be able to view the settings page when requested.
S4.4 Inputs: Authenticated user
S4.5 Outputs: The current settings of the user
S4.6 Normal
S4.7 Blackbox
S4.8 Functional
S4.9 Unit

S5.1 **Server Test 5**
S5.2 This test will ensure that the server will not provide the current settings page if the user is not authenticated.
S5.3 An incorrectly or unauthenticated user will not be able to view the current settings page when it is requested. They will receive an error.
S5.4 Inputs: Unauthenticated or incorrectly authenticated user.
S5.5 Outputs: Unauthenticated error
S5.6 Abnormal
S5.7 Blackbox
S5.8 Functional
S5.9 Unit

S6.1 **Server Test 6**
S6.2 This test will ensure that the server is able to set new settings when correctly authenticated.
S6.3 A correctly authenticated user will be able to change settings on the server.
S6.4 Inputs: Authenticated user
S6.5 Outputs: The current settings of the user
S6.6 Normal
S6.7 Blackbox
S6.8 Functional
S6.9 Unit

S7.1 **Server Test 7**
S7.2 This test will ensure that the server is not able to set new settings when incorrectly or unauthenticated.
S7.3 An incorrectly or unauthenticated user will not be able to change the settings of a server. This should instead result in an error.
S7.4 Inputs: Unauthenticated or incorrectly authenticated user.
S7.5 Outputs: Unauthenticated error
S7.6 Abnormal
S7.7 Blackbox
S7.8 Functional
S7.9 Unit

S8.1 **Server Test 8**
S8.2 This test will ensure that the server is able to query the local file collection via search functionality.
S8.3 A user will query the server via the app for a song located on the local file collection via traits such as Title, Author, or Album.
S8.4 Inputs: A title, author, or album name.
S8.5 Outputs: Songs which loosely correlate with a combination of the title, author, and album names.
S8.6 Normal
S8.7 Blackbox
S8.8 Functional
S8.9 Integration

S9.1 **Server Test 9**
S9.2 This test will ensure that the server is able to have songs added to the queue.
S9.3 A user will add a song to the queue by passing a song URI to the server.
S9.4 Inputs: Song URI
S9.5 Outputs: The song with the associated URI is added to the queue
S9.6 Normal
S9.7 Blackbox
S9.8 Functional
S9.9 Unit

S10.1 **Server Test 10**
S10.2 This test will ensure that the server is able to remove items from the queue by an administrator.
S10.3 A properly authenticated administrator will request to remove a song from HeyJuke's internal song queue and the queue will remove the song.
S10.4 Inputs: Song URI, properly authenticated user.
S10.5 Outputs: The song is removed from the queue
S10.6 Normal
S10.7 Blackbox
S10.8 Functional
S10.9 Unit

S11.1 **Server Test 11**
S11.2 This test will ensure that the server is able to remove items from the queue by the person who originally added the song to the queue.
S11.3 A user who previously requested a song be added to the queue will request to remove a song from HeyJuke's internal song queue and the queue will remove the song.
S11.4 Inputs: Song URI, user with previously submitted song.
S11.5 Outputs: The song is removed from the queue
S11.6 Normal
S11.7 Blackbox

S11.8 Functional
S11.9 Unit

S12.1 **Server Test 12**
S12.2 This test will ensure users that are not administrators and have not previously added a song are unable to remove a song from a queue.
S12.3 A user who is neither an administrator or previously added a song to the queue will request to remove a song. This will result in the server returning an unauthenticated error.
S12.4 Inputs: Song URI, user who is neither a previously
S12.5 Outputs: The song is removed from the queue
S12.6 Normal
S12.7 Blackbox
S12.8 Functional
S12.9 Unit

S13.1 **Server Test 13**
S13.2 This test will ensure that the server is able to return the current queue with additional metadata required.
S13.3 A user will request the current queue of songs. The server should return the current list of songs with additional metadata which may be required if not available from other sources.
S13.4 Inputs: None
S13.5 Outputs: List of songs within the queue with additional metadata when necessary.
S13.6 Normal
S13.7 Blackbox
S13.8 Functional
S13.9 Unit

S14.1 **Server Test 14**
S14.2 This test will ensure that the server is able to return network credentials to the user when queried via NFC.
S14.3 A user will use a device to query the NFC sensor on the server. The server will in turn respond with various data required for connecting to the server, including network credentials.
S14.4 Inputs: None
S14.5 Outputs: Network credentials
S14.6 Normal
S14.7 Blackbox
S14.8 Functional
S14.9 Integration

S15.1 **Server Test 15**
S15.2 This test will ensure that the server is able to pass new songs to the player.
S15.3 A user will add a song to an empty queue. The server will then immediately notify the player of the song in the queue.

S15.4 Inputs: Song URI
S15.5 Outputs: Server notifies player of new song to play
S15.6 Normal
S15.7 Blackbox
S15.8 Functional
S15.9 Integration

S16.1 **Server Test 16**
S16.2 This test will ensure that the server is able to send a new song when the previous song is finished.
S16.3 A user will add a song to a non-empty queue. Once the currently playing song is finished as notified by the player, the next song in the queue should be sent to the player to start playing.
S16.4 Inputs: Song URI
S16.5 Outputs: Server notifies player of new song to play
S16.6 Normal
S16.7 Blackbox
S16.8 Functional
S16.9 Integration

S17.1 **Server Test 17**
S17.2 This test will ensure that the server is able to stop a song on the player and queue a new song if it is removed from the queue while being played.
S17.3 A user will add a song to the queue. When the song starts playing, the same user (or an administrator) will remove the song from the queue. When the song is removed, the song should stop playing on the player.
S17.4 Inputs: Queued then removed song.
S17.5 Outputs: Song starts then stops playing.
S17.6 Boundary
S17.7 Blackbox
S17.8 Functional
S17.9 Integration

M1.1 **Mediaplayer Test 1**
M1.2 This test will ensure that the mediaplayer connects to the server.
M1.3 This test will start the mediaplayer once the server is started and check to make sure that the server receives the websocket initiation message.
M1.4 Inputs: None
M1.5 Outputs: A websocket message
M1.6 Normal
M1.7 Blackbox
M1.8 Functional
M1.9 Integration

M2.1 **Mediaplayer Test 2**
M2.2 This test will ensure that the mediaplayer safely rejects corrupted requests.

M2.3 This test will send random data to the mediaplayer’s open websocket
M2.4 Inputs: Random data as a string
M2.5 Outputs: A websocket message indicating error
M2.6 Abnormal
M2.7 Blackbox
M2.8 Functional
M2.9 Unit

M3.1 **Mediaplayer Test 3**
M3.2 This test will ensure that the mediaplayer safely rejects unsupported commands
M3.3 This test will send an incorrect command to the mediaplayer’s open websocket
M3.4 Inputs: A message over websocket specifying a command that doesn’t exist
M3.5 Outputs: A websocket message indicating error
M3.6 Abnormal
M3.7 Blackbox
M3.8 Functional
M3.9 Unit

M4.1 **Mediaplayer Test 4**
M4.2 This test will ensure that the mediaplayer safely rejects unsupported play media
M4.3 This test will send an incorrect command to the mediaplayer’s open websocket
M4.4 Inputs: A message over websocket with a given song and provider that isn’t supported
M4.5 Outputs: A websocket message indicating error
M4.6 Abnormal
M4.7 Blackbox
M4.8 Functional
M4.9 Unit

M5.1 **Mediaplayer Test 5**
M5.2 This test will ensure that the mediaplayer plays a local file
M5.3 This test will send a correct play command to the mediaplayer’s open websocket
M5.4 Inputs: A message over websocket specifying a command to play a file on the server
M5.5 Outputs: A websocket message indicating successful playback, as well as sound
M5.6 Normal
M5.7 Blackbox
M5.8 Functional
M5.9 Unit

M6.1 **Mediaplayer Test 6**
M6.2 This test will ensure that the mediaplayer stops when prompted
M6.3 This test will send a correct stop command to the mediaplayer’s open websocket
M6.4 Inputs: A message over websocket specifying a pause command
M6.5 Outputs: A websocket message indicating playback has been paused
M6.6 Normal

M6.7 Blackbox
M6.8 Functional
M6.9 Unit

M7.1 **Mediaplayer Test 7**
M7.2 This test will ensure that the mediaplayer continues playback when prompted
M7.3 This test will send a correct resume command to the mediaplayer's open websocket
M7.4 Inputs: A message over websocket specifying a resume command
M7.5 Outputs: A websocket message indicating playback has resumed, and sound of the current song starting from the position when the music was paused
M7.6 Normal
M7.7 Blackbox
M7.8 Functional
M7.9 Unit

M8.1 **Mediaplayer Test 8**
M8.2 This test will ensure that the mediaplayer can start a new song while another one is playing
M8.3 This test will send a correct play-song command to the mediaplayer's open websocket while another song is playing
M8.4 Inputs: A message over websocket specifying a file on the server, sent while another file is being played
M8.5 Outputs: A websocket message indicating successful playback, as well as audio of the new song and not the old.
M8.6 Normal
M8.7 Blackbox
M8.8 Functional
M8.9 Unit

A1.1 **Mobile App Test 1**
A1.2 validate search functionality
A1.3 This test will run perform a search using the name of an artist
known to be on the utilized streaming provider and will receive the results
from the server.
A1.4 Inputs: The search query will be the input
A1.5 Outputs: The list of expected search results for the given search term /
provider will be the output
A1.6 Normal
A1.7 Blackbox
A1.8 Functional
A1.9 Integration

A2.1 **Mobile App Test 2**
A2.2 Validate "now playing" functionality
A2.3 This test will ensure that the player UI updates when the track is

changed on the server
A2.4 Inputs: The server will send a track change event to the mobile app as the input
A2.5 Outputs: The “now playing” screen UI will update to indicate the new track
that has begun playing
A2.6 Normal
A2.7 Whitebox
A2.8 Functional
A2.9 Integration

A3.1 **Mobile App Test 3**
A3.2 Validate Spotify authentication
A3.3 This test will ensure that Spotify validation works as expected
and returns the necessary tokens
A3.4 Inputs: User logs into the Spotify login page with their Spotify Premium
credentials
A3.5 Outputs: The app receives authentication tokens from Spotify and can
successfully perform a request against the Spotify API
A3.6 Normal
A3.7 Blackbox
A3.8 Functional
A3.9 Integration

| | **Normal / Abnormal** | **Blackbox / Whitebox** | **Functional / Performance** | **Unit / Integration** |
|---|---|---|---|---|
| **S1** | Normal | Whitebox | Functional | Integration |
| **S2** | Normal | Blackbox | Functional | Unit |
| **S3** | Abnormal | Blackbox | Functional | Unit |
| **S4** | Normal | Blackbox | Functional | Unit |
| **S5** | Abnormal | Blackbox | Functional | Unit |
| **S6** | Normal | Blackbox | Functional | Unit |
| **S7** | Abnormal | Blackbox | Functional | Unit |
| **S8** | Normal | Blackbox | Functional | Integration |
| **S9** | Normal | Blackbox | Functional | Unit |
| **S10** | Normal | Blackbox | Functional | Unit |

| | | | | |
|---|---|---|---|---|
| **S11** | Normal | Blackbox | Functional | Unit |
| **S12** | Normal | Blackbox | Functional | Unit |
| **S13** | Normal | Blackbox | Functional | Integration |
| **S14** | Normal | Blackbox | Functional | Integration |
| **S15** | Normal | Blackbox | Functional | Integration |
| **S16** | Normal | Blackbox | Functional | Integration |
| **S17** | Boundary | Blackbox | Functional | Integration |
| **M1** | Normal | Blackbox | Functional | Integration |
| **M2** | Abnormal | Blackbox | Functional | Unit |
| **M3** | Abnormal | Blackbox | Functional | Unit |
| **M4** | Abnormal | Blackbox | Functional | Unit |
| **M5** | Normal | Blackbox | Functional | Unit |
| **M6** | Normal | Blackbox | Functional | Unit |
| **M7** | Normal | Blackbox | Functional | Unit |
| **M8** | Normal | Blackbox | Functional | Unit |
| **A1** | Normal | Blackbox | Functional | Integration |
| **A2** | Normal | Whitebox | Functional | Integration |
| **A3** | Normal | Blackbox | Functional | Integration |