Jukebox Testing Documentation

4/29/22

Version 2.0

Derek Muse - 4/29/2022
Noah Beliveau - 4/29/2022
Bernie Sander - 4/29/2022

Table of Contents:

Project description:

Music NFTs have been identified as an emerging and disruptive market that can help artists retain the most value from their music as well as bring fans closer to artists. NFTs are a relatively new vehicle for artists to distribute their music, which means the user experience when interacting with these music NFTs is still a work in progress for both artists and fans. Turning music into an NFT can be a complicated process for artists who may not be tech-savvy and fans that purchase music NFTs desire a UI that allows them to easily listen to the music NFTs, create playlists and buy/sell music NFTs. In addition, NFTs create an opportunity to build a community through ownership. This sense of community in the music space is currently lacking. Artists do not have a good medium to interact with their fans on the current streaming platforms (Spotify, Apple Music, Soundcloud) and fans do not have a place where they can interact with each other on these platforms outside of collaborative playlists.

Jukebox aims to deliver a platform where artists, fans, and music NFT enthusiasts can listen to and create playlists with their music NFTs, buy/sell music NFTs, as well as interact and follow each other.

Jukebox will initially be launched as a web application, but will eventually be available on iOS and Android mobile devices as well. One of the main goals is to make this application as accessible as current streaming services, allowing the average artist or fan to seamlessly interact with music NFTs and all of its potential utility.

Static Tests:

- The following lists represent React components that make up our UI
    - AudioControls
        - Sole purpose: Handles the audio execution in response to user action (play, pause, rewind, fast forward)
        - Modular: Yes
    - Navbar
        - Sole purpose: Persist through the entire site. Display links to home, profile, and show loaded songs
        - Modular: Yes
    - List Components
        - Sole Purpose: Display List items in a list format
        - Modular: Yes
    - List Item Components
        - Sole Purpose: Display song, playlist, user information to be used in the list component
        - Modular: Yes
    - Home Component
        - Sole Purpose: Display general information
        - Modular: Yes
    - Playlist Component
        - Sole Purpose: Display songs and comments in a playlist
        - Modular: No
    - Profile Component
        - Sole Purpose: Display user information
        - Modular: No
    - Song Component
        - Sole Purpose: Display user songs
        - Modular: No
    - Edit/Components
        - Sole Purpose: Update corresponding component (Playlist, Profile)
        - Modular: No
- Moralis code follows best practices by making each function asynchronous and using the ACL (access control list) provided by Moralis to enforce authorization

Unit Tests for Cloud functions:

Method	Input	Expected Result	Actual Result
getEqualTo()	object, field, value	item of an object given a value of a field	Matches expected result
getAll()	object	all items of a given object	Matches expected result
getObjectRelations()	object, objectId, relation	objects in a relation given objectId	Matches expected result
queryRelation()	object, objectId, relation, itemId	item of a given object	Matches expected result
getMusicNFTs()	chain, address	get a music nft from Moralis given chain and address	Matches expected result
addItemToRelation()	object, objectId, item, itemId, relation	add an item to a relation	Matches expected result
removeItemFromRelation()	object, objectId, item, itemId, relation	removes an item from a relation	Matches expected result
updateObject()	object, objectId, updates	updates an object based upon entered updates	Matches expected result
newPlaylist()	name, collaborative, public	create a new playlist	Matches expected result
newUpdate()	content	create a new update	Matches expected result
newComment()	data, parentObj, parentId	create a new comment	Matches expected result

deleteObject()	object, objectId	delete an object based upon objectID	Matches expected result

Integration Tests:

We will test our react component integration using a bottom up method, with drivers to simulate higher level components. Our Moralis-React integration will also be tested through unit testing the React State after making requests to our database, to ensure both that Moralis is returning the correct data, and React is updating the State correctly. Finally we will test to ensure that our CSS styling integrates with our React components through a combination unit testing and fault-injection for various use cases.

Our React components are integrated properly, and function as expected. Our UI correctly implements our Moralis cloud functions, and updates the React state accordingly. These have been confirmed using bottom up integration tests, to ensure that data is passed through components correctly. Our CSS styling integration was tested with a combination of fault-injection and bottom up integration methods to ensure that the components are rendered correctly in all possible cases.

Validation Tests:

We will review our product to ensure that all requirements listed in our SRS are accounted for by our final product. We will also perform stress tests by simulating edge case scenarios that could cause faults in our software. Finally, we will provide our software to our clients and a group of stakeholders for feedback and review.

Our product meets all of the base requirements detailed in our SRS. After performing stress tests of both the Cloud Functions and UI, we can confirm that our software can handle errors and exceptions, and that no potential end user use cases considered by the development team cause our software to break. Our product will be put up for review to our clients and a group of select stakeholders after the semester.

System Tests:

The remainder of each section will be completed upon completion of the testing.

- *Security*
    - *Test ID*
        - *1*
    - *Time/Date*
        - *4/29/22*
    - *Requirements ID*
        - *4.3*
    - *Tester*
        - *Derek Muse*
    - *Environment*
        - *Chrome, Safari, Firefox*
    - *Goal*
        - *Ensure that only users are able to edit their profiles and that users are able to authenticate themselves through their wallet*
    - *Testing Procedure*
        - *Each member of the group has authenticated themselves through MetaMask wallets and tested that we can edit our own profiles, but have no editing abilities on other user's profiles.*
    - *Expected results*
        - *Users are able to authenticate themselves through their wallet, edit their own profiles, but not other user's profiles.*
    - *Actual results*
        - *Matches expected results.*
    - *Status*
        - *Complete*
    - *Comments*
        - *Profile page is coming along nicely!*

- *Recovery (handling promises)*
    - *Test ID*
        - *2*
    - *Time/Date*
        - *4/29/22*
    - *Tester*
        - *Bernie Sander*

    - *Environment*
        - *Chrome, Safari, Firefox*
    - *Goal*
        - *Ensure that promises are handled correctly (i.e., there is both fulfilled and unfulfilled logic)*
    - *Testing Procedure*
        - *Run each use case to ensure promises are handled correctly*
    - *Expected results*
        - *Fulfilled promises are shown through the user interface and all unfulfilled promises are caught by the Error component*
    - *Actual results*
        - *Matches expected result*
    - *Status*
        - *Complete*
    - *Comments*
        - *Site runs smoothly!*

- *Deployment testing (different browsers)*
    - *Test ID*
        - *3*
    - *Time/Date*
        - 4/29/22
    - *Tester*
        - *Noah Beliveau*
    - *Environment*
    - *Goal*
        - *Ensure that application runs on Chrome, Safari, Firefox*
    - *Testing Procedure*
        - *Observation of application performance for various use cases on alternate browsers.*
    - *Expected results*
        - *Application will work similarly across browsers.*
    - *Actual results*
        - *Matches expected result*
    - *Status*
        - *Complete*
    - *Comments*
        - *No issues here!*