**BeatBlocks Project Report**

**Class:** CIS4930/CIS6930

**Date:** 4/22/18

**Github Project Repo Link:** https://github.com/beatblocks/beatblocks\_web

**Group Members:** Trenton Fleming, Esteban Zapata, Byron George

**Project Summary:**

This project combines IPFS (InterPlanetary File System) and the Ethereum network to create a decentralized music streaming platform. This allows musicians and artists to have control of their content. In addition, they will have the freedom to publish and set the price for their work. Users and listeners are able to subscribe and enjoy artist’s content.

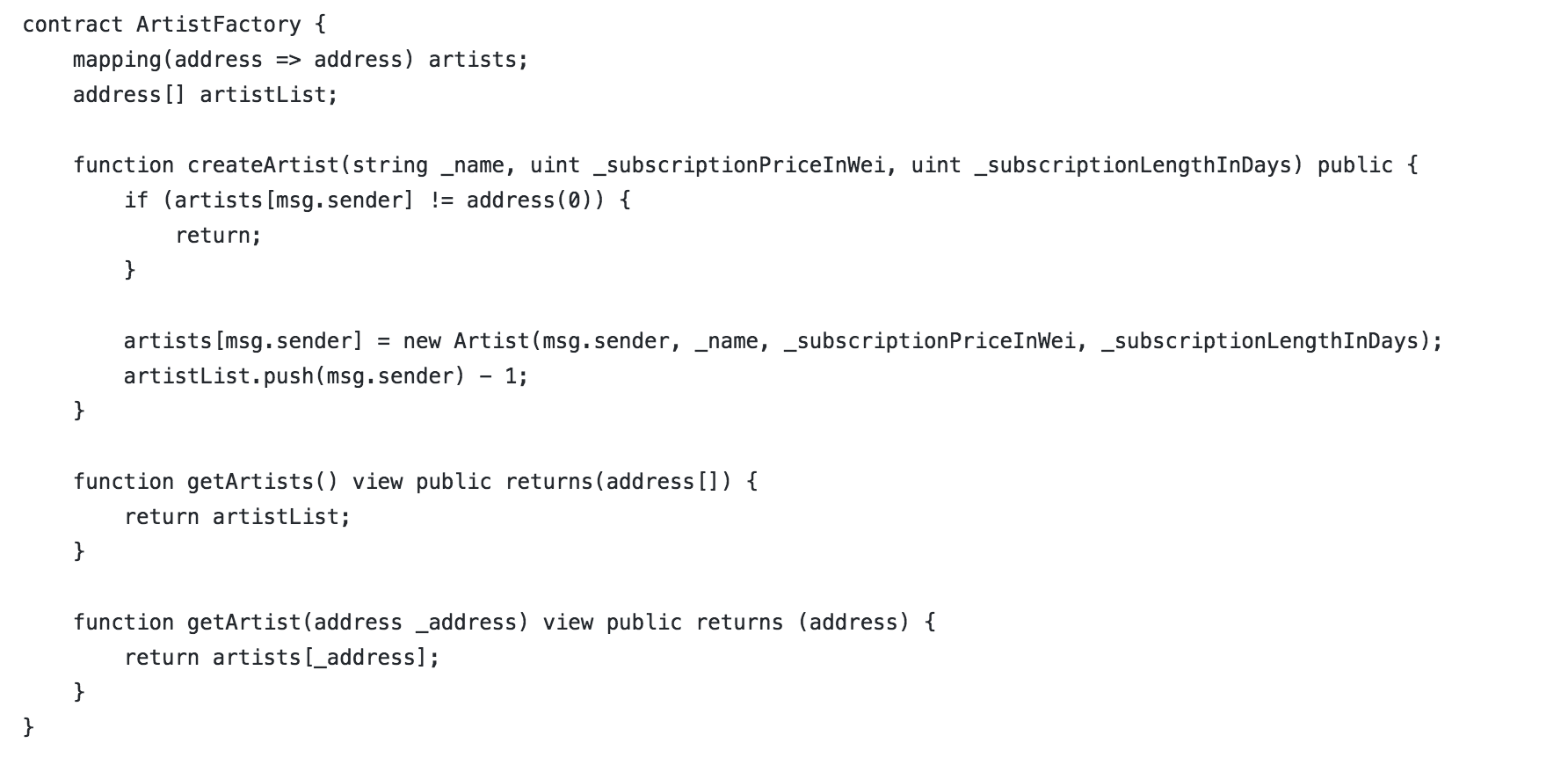
**Implementation:**

In order to ensure the artist’s content would be immutable and directly manageable while still allowing other users to subscribe directly to the artist, Ethereum smart contracts were used. This is done by allowing artists to create and add what are called collections. Collections are one or many songs associated with a number of track names and audio files. The data for these songs are then stored in what is called IPFS. IPFS will store a collection and return a hash of its location for storage in the smart contract. In order to access the Ethereum network and IPFS we used infura’s access nodes (<https://infura.io/>). To make the application independent, the front-end is a single page application built with React, which is a Javascript library for building user interfaces, and a chrome plugin called metamask.

Metamask is a chrome plugin that manages several accounts and their private keys for users. To allow communication to the ethereum network metamask will inject its own instance of web3 into the browser’s window. This creates a global instance of web3 and allows users to interact with our Dapp. In addition, metamask simplifies the user interacton with the Dapp. Allowing all the addressing to be done behind the scenes securely and without the need of exposing their private keys.

**Ethereum:**

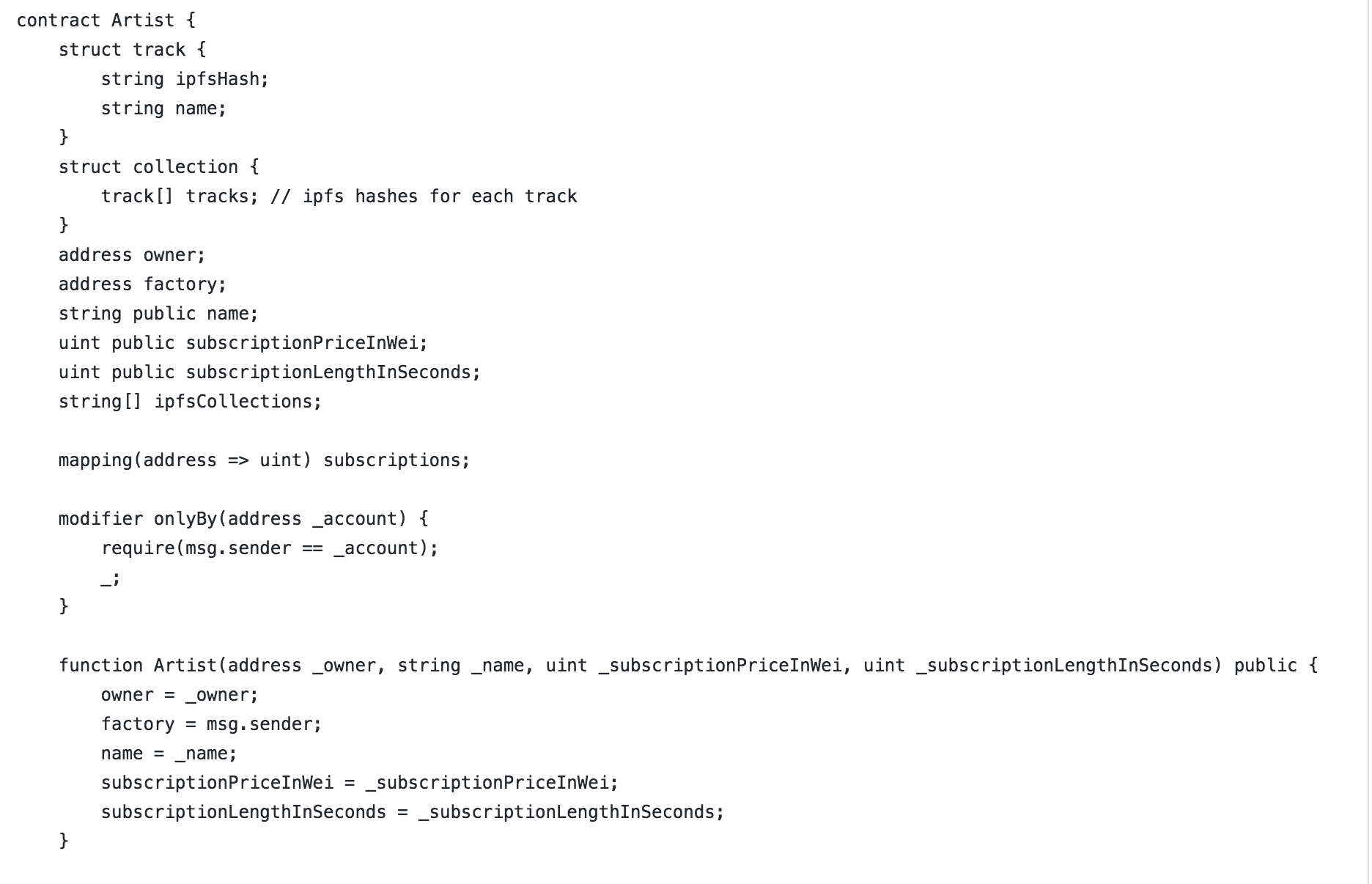
To make sure the artists would get their money we developed smart contracts and stored them into blocks on the ethereum network. In total, we created two smart contracts. The first contract is called the artist factory. This contract manages the artists. It has functions that allow creation of artist, get all the artists, and get a specific artist. The address of the artists is stored into an array and in addition the artist is mapped to an address. The array is used to retrieve all artists using the contract and the mapping allows one to retrieve information about a specific artist.



*Snippet 1: ArtistFactory smart contract*

The second smart contract defines the artist object, and defines core functionality for managing collections and subscriptions. Once an artist is signed up they can begin adding their collections. A collection is a set of tracks with associated metadata like track names and the collection image. Storing a collection on IPFS will return a hash of it’s address. That hash is then stored into an array in the smart contract and used to query/manage collections. There are functions to get the IPFS collection size, get the specific IPFS collection, add and remove IPFS collections, and update IPFS collections. When it comes to the remove, add, and update functions it is important to note that IPFS acts as one big version control; content written to it is immutable and never removed. When one wishes to update a collection, the old collection will remain, but the new collection will produce a new hash. Thus, the new hash replaces the old hash in the smart contract and becomes the new reference.

The remaining functions deal with subscriptions, the artist, and updating general information. Subscriptions are temporal meaning a user will specify how long they wish to subscribe to the content. The unit for this time period is in seconds. The price is in wei and is set by the artist when they are adding the collection. If the user does not have enough wei and they go to subscribe the transaction will not go through and the user does not get the subscription. If they do have the funds, that amount of wei is sent to the artist and a timestamp of the transaction going through plus the subscription length in seconds is stored into a mapping along with the user’s address. That way whenever a user wishes to listen to a track in the collection, a function to check that user’s subscription time is called and if the time returned is greater than the current time, the user is allowed to listen. If the time returned is less than the current time, the user’s subscription is considered up and they are not able to listen to the music. Artist’s also are allowed to change their subscription price and subscription length whenever they wish to.



*Snippet 2: Artist smart contract*

**The Block:**

In addition to the smart contracts, some other information is stored onto the block. This information is the artist name, subscription price. IPFS hash, and the users subscribed.

**IPFS:**

InterPlanetary File System (IPFS) is a protocol and network we use to store the artists collection data and the block header. As explained above, whenever a collection is published/written to IPFS it returns a hash or reference to that index. That hash is then stored into the block and used to find that specific collection when requested. What gets returned from IPFS is a hash of the collection header.



*Snippet 3: IPFS collection header*

As seen in snippet 3, the collection header contains everything for the collection. The tracks hashes are all addresses to the specific songs in the collection with respect to the track names.

The reason for using IPFS rather than storing everything in the blockchain is that it would be way too expensive. IPFS gives us a free, distributed, and secure way to handle this storage problem while minimizing Ethereum related fees for artists.

**Limitations:**

While working on this project we came across limitations of the technologies we were using. One big limitation for us was the fact that Ethereum can’t handle returning datatypes with varying memory (structs, arrays of strings, etc.). This was big problem for us when trying to figure out how to store and retrieve the IPFS hashes to access the collections. Our work around for this problem was to return the count N of the array and then make subsequent N requests for each element. This forces the creation of many network requests all at once, adding unnecessary congestion between the client and Ethereum’s network.

Another limitation is the disadvantage of waiting for a transaction to be verified. Each transaction takes on average 15 seconds of time before it goes through because that is the block time for Ethereum. From a user point of view the time taken is not optimal, but there is nothing that can be done until Ethereum moves to PoS.

Lastly, the application will not work without Metamask. Metamask is a chrome plugin that manages accounts and the private keys associated with those given accounts. It handles signing transactions without exposing user’s private keys to Dapps. It is also largely responsible for reducing overhead in adding that interaction. This is currently the greatest limitation, limiting users only to Google’s chrome and the plugin.

**To Do:**

There are still many optimizations and work to be done on this project. One thing to be done is to optimize the gas cost. We did our best to keep the gas cost low, but there are still optimizations we can make that can make it even lower. Some of these include specifying the uint size for certain data, trying to make functions view only, and storing information in a more efficient manner. For example, the IPFS can be base58 encoded and remove the first 2 bytes (Every IPFS hash starts with Qm). This will reduce the cost by 1/3.

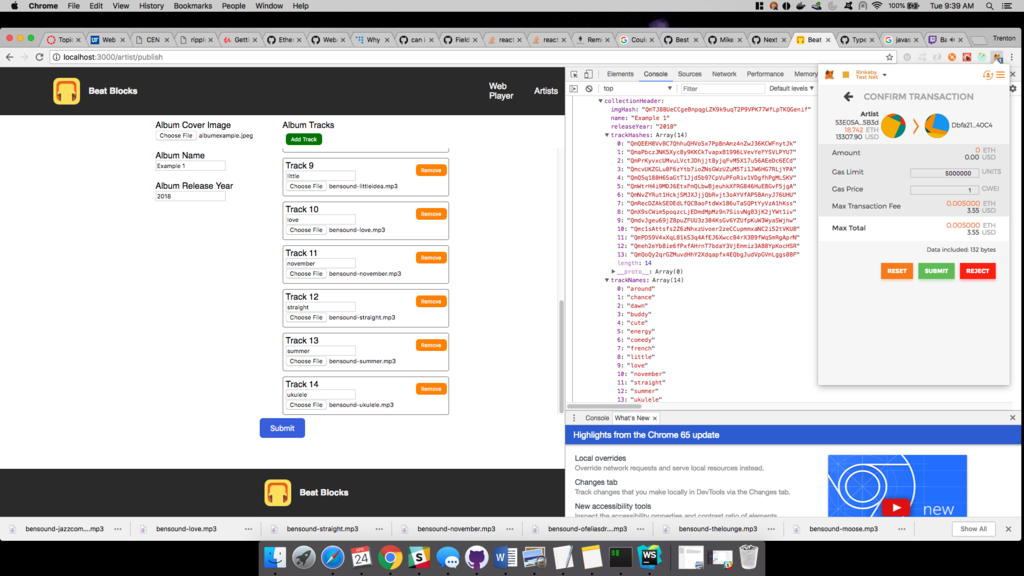
In addition to gas optimizations, security optimizations can be made. One such example is that if someone has the collection header they have the hashes to all the tracks. This issue allows the exploitation of the service and defeats the purpose of trying to secure artists’ content.

Error checking and handling needs to be implemented more. As of now there are not many checks for errors. This hinders the user experience because if an error occurs they won’t know unless they really look for it.

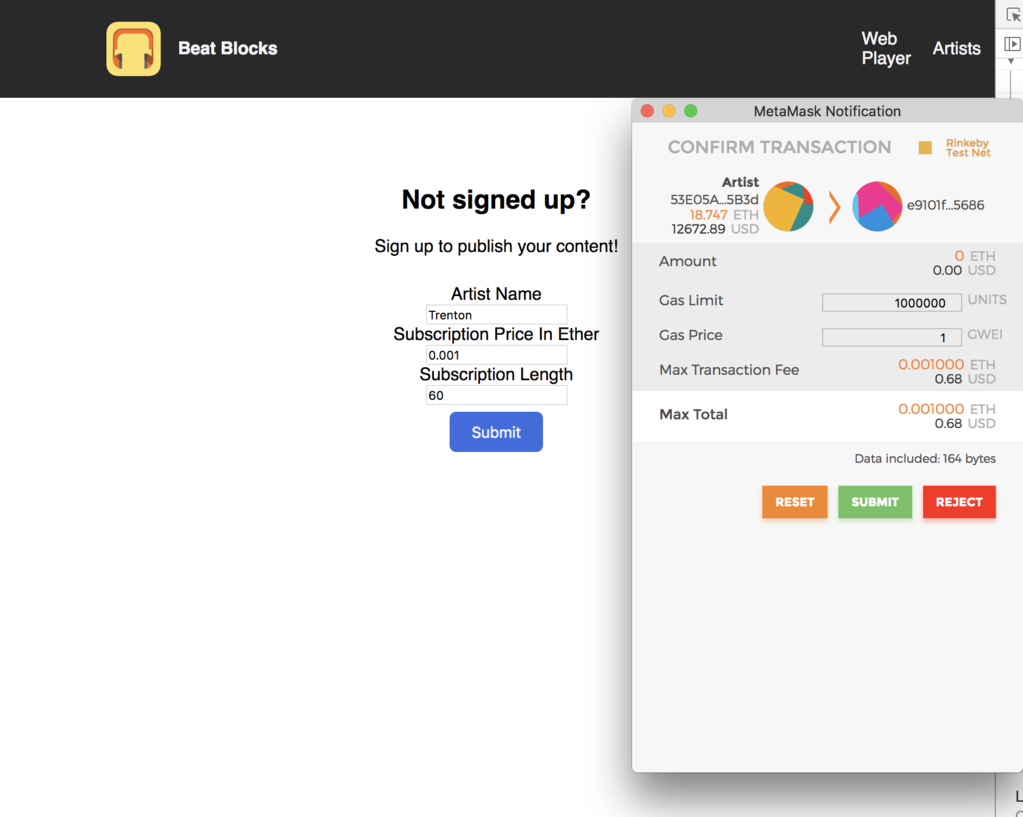
Another thing to do is find a way to manage contracts/addresses ie contract migration. For example, if we deploy a new contract all the artists/users that were using the previous contract will no longer be able use the service since that contract is no longer being used. This problem arises because of the block chain’s immutable nature. Because nothing can be changed the only way to make modifications and apply them is to deploy a new smart contract and then have those previous addresses point to the new one.

Lastly, something to do is eventually make this be able to run on its own and find a way to remove the metamask requirement.

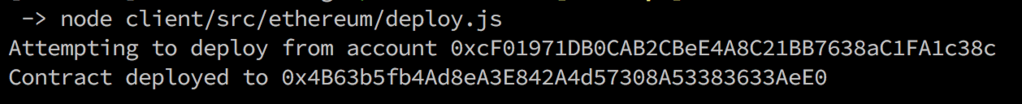
**Screenshots:**



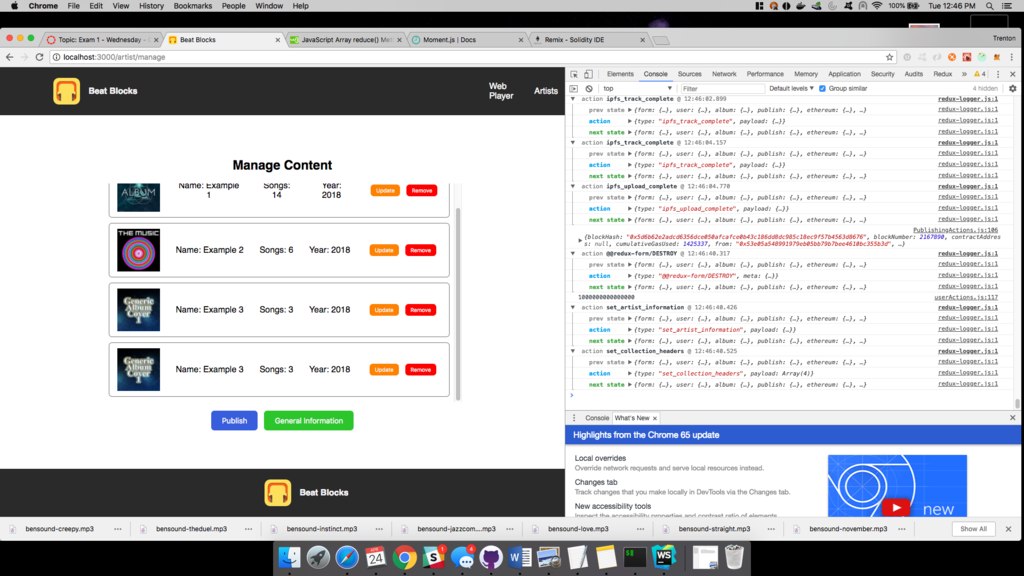
*Snippet 4: User uploading content. Users are able to publish what they want, whenever they want*

**

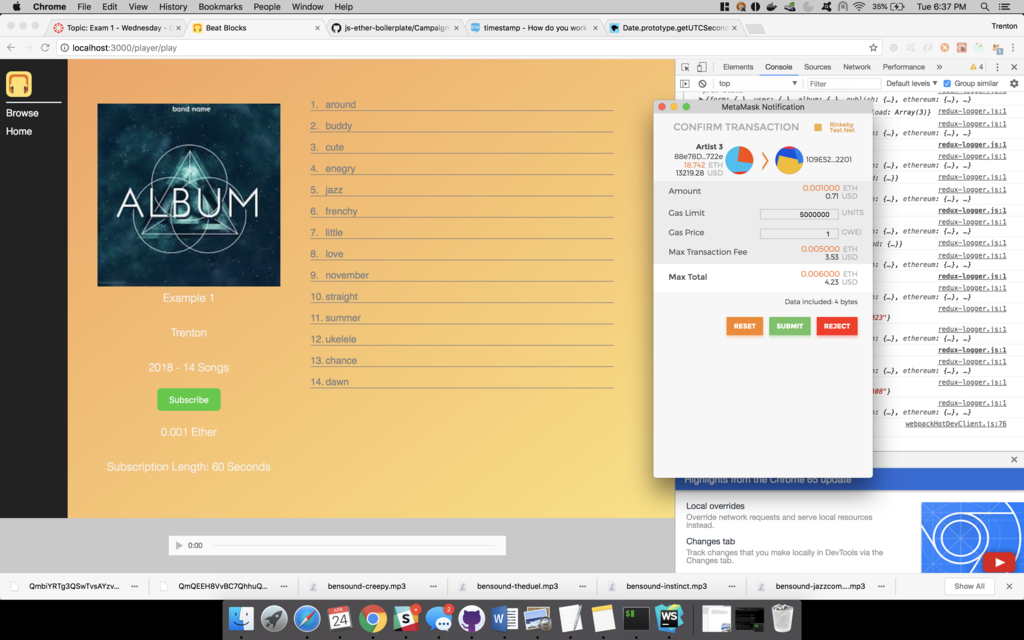
*Snippet 5: Anyone can sign up (requires metamask)*

**

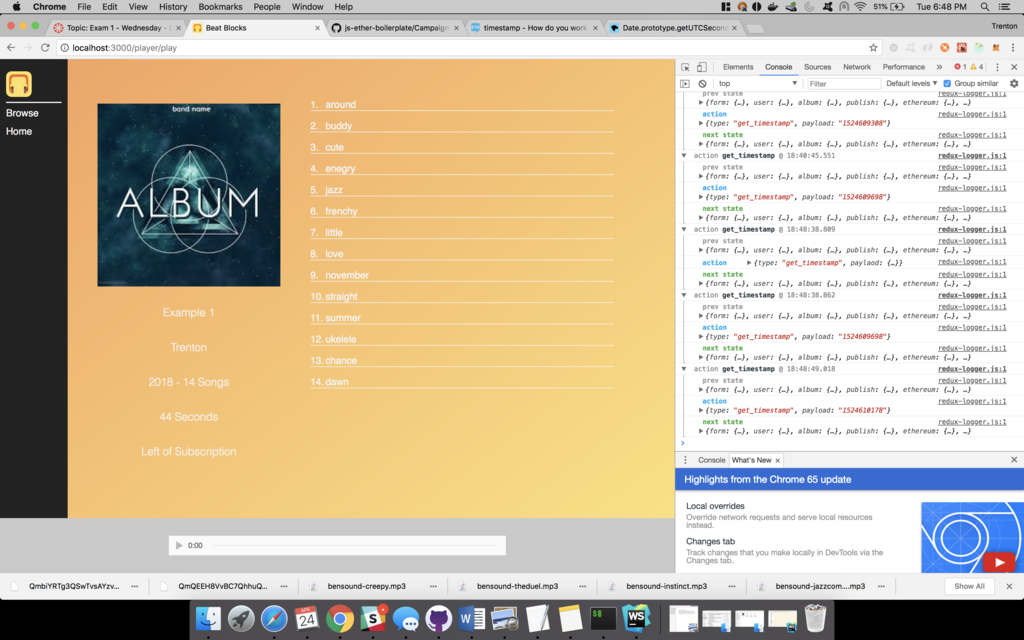
*Snippet 6: The deployed factory*

**

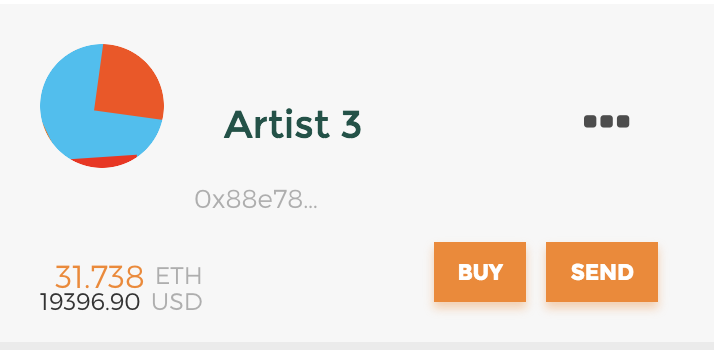
*snippet 7: Users have complete control*

**

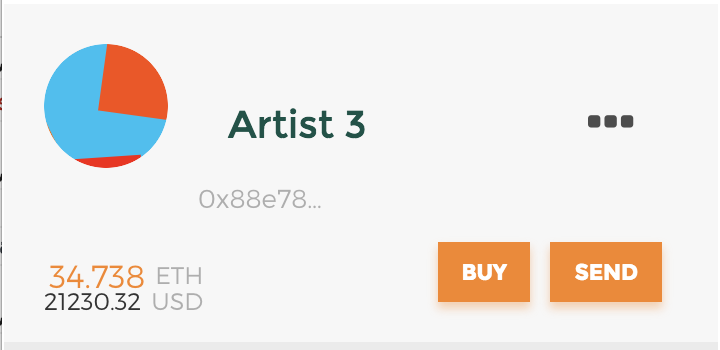
*Snippet 8: Before subscribing, gray tracks mean disabled*

**

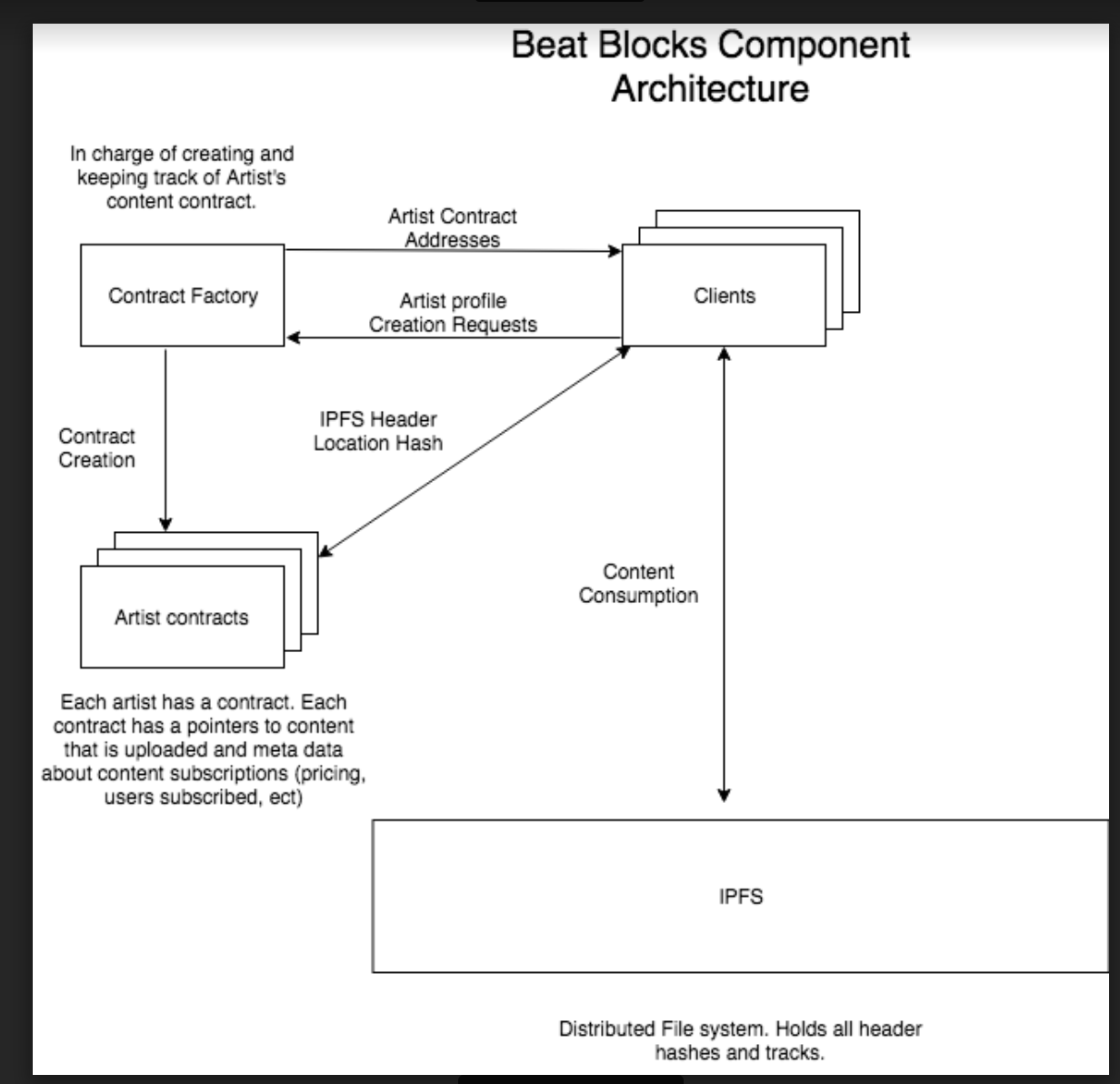
*Snippet 9: After subscribing*

**

*Snippet 10: Artist 3 balance prior to customer subscribing*

**

*Snippet 11: Artist 3 balance after customer subscribes*

**

*Snippet 12: Beat Blocks Component Architecture*