Music Streaming Decentralized Application JJAX

Jake Grieves, Andrew McWilliam, Xinrui He, Jiayi Zhong
Financial Technology with Data Science
University of Bristol
Bristol, U.K.

I. Introduction

The advent of music streaming platforms has completely transformed the way we purchase and consume music, but it has also given rise to several issues. While it is extremely convenient for music enthusiasts to use these platforms and enjoy music with just a click, high fees and restricted content access prevent some users from availing the service. On the other hand, for music creators, especially for those who are not mainstream, these platforms provide an opportunity to showcase their work to a wider audience. However, uploading music to traditional streaming apps can be a costly and challenging task. Music creators need to use distributing services to upload their music and face issues such as high fees, low income, and copyright concerns.

Our decentralized streaming music platform JJAX attempts to solve these problems with blockchain technology. Our platform is built on Ethereum, utilizing smart contracts, tokenization, and NFTs to provide a more open and transparent environment for all users.

For creators, JJAX offers a fairer and more transparent revenue sharing system. We provide free music file uploading services, enabling original creators to upload their music to our platform, set their own prices, and receive the majority of their earnings in cryptocurrency when other users purchase their music, all without worrying about piracy. Secondary creators can purchase music at a lower price on our platform, remix it, and then upload their creations for listening or resale, with original creators receiving a share of the earnings. Users can easily browse and search for new music on our platform and purchase their favorite tracks at a lower price.

This report will first discuss the background and motivation behind creating JJAX. Then, we'll describe the overall system architecture and user interface design. We'll also delve into the technical details of how JJAX operates, including the use of smart contracts, tokenization, and NFTs. We'll evaluate the effectiveness of JJAX and discuss future directions for improvement. Finally, we'll summarize the development process and the challenges we faced along the way.

In summary, JJAX represents an innovative solution to the problems plaguing traditional music streaming platforms, and we believe it has the potential to disrupt the industry and create a fairer and more sustainable music consumption model.

II. BACKGROUND

With the growth of the internet and the popularity of music, composers and musicians are beginning to have more and more markets to distribute their music, but at the same time the income generated by their works is becoming less and less due to the excessive number of third-party music markets on the internet, and can also be affected by copyright issues for the music.

Listeners dominate the market as there are several different platforms for music to choose from. However, most of these music platforms are centralised, so there are multiple formalities between creator and listener transactions, and significant fees are charged, i.e. the price of the creator's own work plus intermediate fees is the actual price paid by the consumer. In particular, the monopoly of some music platforms on well-known music works has led to high prices, with most of the benefits of this monopoly going to the intermediary platforms, while the profits for creators remain limited, giving rise to a lot of pirated music resources, which further undermines the interests of creators. Moreover, these central organisations collect data on users' behaviour and analyse their habits, which in turn violates consumers' privacy to a certain extent.

As a result, decentralised trading software has emerged. For the music market, decentralised applications allow for direct trading between music creators and users, making it easier for new creators to promote their work, and for users to sell their music at a lower price due to fewer fees. In addition, the decentralised music platform ensures that the music purchased by users is copyrighted, and for fans of specific creators or music lovers, it ensures that the music they purchase is authoritative. Based on current blockchain technology and smart contracts, it is possible to implement this decentralised music platform where the blockchain and smart contracts act as a third-party organisation where users can directly purchase copyrighted music.

The best example of the current development in Dapp is Audius, a platform for decentralised music sharing and streaming protocols, a format that dramatically increases the ability of creators to express themselves freely and to communicate directly with their fans, while ensuring that music creators receive the majority of the revenue from the benefits created by their work. 90% of the platform's revenue will go directly to the artist. Audius is designed to give creators more control over their music, as opposed to other third-party platforms, where only 12% of revenue goes to the artist. By uploading their music to Audius, creators can leave a permanent record of their work, which will be protected by a decentralised network of nodes.

A key difference between Audius and other streaming music platforms such as Spotify is that it operates in a decentralised way using blockchain technology, such as NFT gating, which allows artists to distribute exclusive content to NFT holders. 2 can "gate" their content here, so only token holders of the corresponding collection can access it. Audius effectively solves the copyright challenges faced by creators in the music industry, including music rights, royalties and ownership, giving every user the freedom to publish or listen to music, while ensuring the rights of both creators and consumers.

The profitability of Dapps is also limited due to the decentralised model of Dapps which simplifies the transaction model and eliminates many fees. Many Dapps will be profitable on a drawback basis, such as simulated stock market or gambling Dapps where users are required to buy or hold certain assets or tokens to participate. When the price of these assets or tokens changes, the Dapp re-calculates the value of the user's assets and pays the holder a corresponding bonus, from which the Dapp can take a percentage of the bonus as a source of profit. Or it can take a direct commission when users perform trading activities and top-ups. Alternatively, the most straightforward way to monetise an app is to include advertising in the app and receive a fee for doing so. This model is also applicable to music Dapps.

In the music Dapp, users can buy individual tracks or entire albums to access music services and the Dapp can earn revenue from the sale of the music, for example Audius earns a 10fee on each sale. In addition to this Dapp can also combine music related products and services with its music services, for example selling music tickets, selling album or songwriter

III. Design Documentation

A. Initial Idea

The initial idea for JJAX, outlined a blockchain Dapp similar to the online record store Bandcamp [3]. In the idea, users would be able to upload and buy digital music, with features existing for the streaming of said music. Adding to this point the implementation of auction-like sales was explored, in which users can bid to buy unique pieces of music.

The idea while sound in nature runs into issues when considering its use within the Ethereum blockchain. Storing files on the blockchain is incredibly expensive and therefore alternate solutions were required, this led to the research of the IPFS system, however, this in turn led to further issues. Without a dedicated node or server, the IFPS system can be incredibly slow which is detrimental to the "Streaming" ethos. Considering the auctions section of the idea, the project runs into the "free rider problem". While the user who won the auction will get a digital file of the "unique piece of music", should a third party get access to the file, there is no way to prove ownership and in effect, the first and third-party users have exactly the same product. Therefore users are likely to feel auctions are unjustified and are unlikely to use the product. With these two problems in mind, the group came to the consensus that the project fails to justify its own creation, as we would simply be creating a lesser version of a product that already exists.

B. The Solution

NFTs (Non-Fungible-Tokens) are well explored in the creation and selling of digital art and act as proof of ownership for digital files. While typically the technology exists as a way for artists to "mint" and sell digital image files, the technology is incredibly general and can be used in conjunction with the initial idea. By combining the original idea with the NFT technology, we pivot into creating a more proof-of-ownership-orientated, music distribution platform.

C. Evaluated Idea

Building from the evaluation of the initial idea, a project idea was drawn up to better combine the research and ideas the group had. This idea outlines a decentralized music distribution app, heavily relying on NFT technology to validate and prove ownership of music.

The main functionality of the app will allow an artist to upload a unique piece of music to the IPFS system, in which a token will be created representing the uploaded content. This token can be exchanged between accounts and auctioned off for a certain amount of Ethereum and will follow the NFT principles. This token will also be associated with only one address and therefore the ownership of the token can be validated through the blockchain.

The main user base of the project is intended to be that of artists and dedicated music fans. While there exist numerous existing music platforms for artists to share their work. The project introduces the novel idea of proof of ownership, therefore allowing the project to operate in an untapped niche. To create music, musicians often iterate upon musical ideas in order to create the end product, however, the initial ideas or demos are often unreleased and therefore aren't monetized. While some artists see some moderate success in releasing these demos alongside album releases, this often leads to critique from the more casual consumers, as the music is often unpolished and or unfinished. Through the ideas proposed, an artist would be able to tokenize and mint a musical demo before being able to sell the product in an auction-like sale. The dedicated music fans would be able to own a unique piece of music from the artist, and the artist would be able to monetize their demos without the risk of tarnishing their legacy by releasing unpolished/unfinished music.

D. NFT Standards

NFTs are a widely used technology within the blockchain and because of this, there exist numerous standards that projects can make use of. Therefore as an important first step, a viable standard should be identified and built upon in order to create the project. As there are so many options a few well-used standards are listed and described below:

- **ERC 20** "The Token Standard". The original standard for fungible tokens, an ERC-20 Token acts just like ETH, and because of this, each token is made equal. While not an NFT standard this standard may be useful to implement voting and or governance tokens within the project. [4]

- **ERC 721** A ”Non-Fungible Token Standard”, in this standard tokens are unique and distinguishable from one another and each ERC721 token has a distinct identifier (token ID). [5]
- **ERC 1155** The ”Multi-Token Standard”. This standard is similar to the ERC 721 standard with the caveat that multiples of each token can exist. For example, a user could own x amount of a given NFT and is given the ability to send a specified number of them to another user. [6]
- **ERC 998** ”Composable NFTs”. Extending upon the ERC 721 standard users are able to own ERC 721 and ERC 20 tokens but users are now given the ability to ”merge several NFTs into one NFT”. [7]

While a plethora of standards exist within the NFT space, the goal of the project is to create a viable product, and therefore a single standard should be picked and stuck to in order to reduce complexity (and therefore wasted resources) within the project. Out of the ERC standards listed, ERC 721 and ERC 1155 are the obvious picks, offering the functionality outlined in the ”evaluated idea” section. While ERC 1155 allows for proof of ownership and in the example of the project, would allow musicians to mint and sell multiple versions of a demo. This somewhat goes against the project idea. The main brief outlines the uniqueness of the music being sold, and therefore by allowing demos to be minted multiple times, this uniqueness is diluted. Along with this, consumers of music will find no added benefits of owning multiples of the same demo. Owning the song 100 times will have the same user experience as owning the song once, and therefore this is likely to fuel a secondary market of song resellers, which goes against the principles of the project. The main goal again is to facilitate the selling of unique music from musicians to dedicated fans. Because of the reasons listed, the ERC 721 ”Non-Fungible Token Standard” will be used as the backbone for the project.

E. The ERC 721 Standard

The ERC 721 Standard defines a standard interface for creating and managing unique tokens and will be used as the basis for the project. Every ERC-721 compliant contract must implement the ERC 721 interface, defined by the following functions, as listed on the OpenZeppelin website [8].

- **event Transfer(from,to,tokenId):** Emits when ownership of a NFT changes.
- **event Approval(owner, approved, tokenId)** emits when the approved address for a NFT is changed.
- **event ApprovalForAll(owner, operator, approved)** emits when an operator is enabled or disabled for a owner
- **function balanceOf(owner)** returns a count of all NFTs assigned to a owner
- **function ownerOf(tokenId)** returns the address associated to a given token.
- **function safeTransferFrom(from, to, tokenId)** Transfers ownership of an NFT from one address to another.
- **function safeTransferFrom(from, to, tokenId, data)** Same as the other safe transfer function, but with the added ability to send data to the receiver.
- **function TransferFrom(from, to , tokenId)** Same as safetransfer however if the receiver address is incapable of receiving a token, the token may be permanently lost.
- **function approve(to, tokenId)** can change the approved address for an NFT, i.e. allows the approved to control the specified NFT
- **function setApprovalForAll(operator, _approved)** allows a third party to maange all of the sender’s assets.
- **function getApproved(tokenId)** Gets the approved address for a single NFT
- **function isApprovedForAll(owner, operator)** Query if an address is an authorized operator for another address

These functions act as the basis for the ERC 721 standard and therefore the basis of the project. While manual implementation is possible, there exists the open-source library OpenZeppelin which contains implementation for industry standards such as ERC 20, ERC 777 and in the case of the project ERC 721. Therefore through the use of contract inheritance, the project will make use of the pre-defined open zeppelin contracts. The project makes use of 5 open-souce contracts, which are as defined:

- **ERC721.sol:** This smart contract contains basic implementation of the functions listed in the ERC 721 standard, and therefore allows for the creation and distribution of NFTs. The implementation however lacks a lot of the functionality listed in the brief. Within the contract, tokens are non-fungible, however, the ID must be defined by the user, and the tokens have no way of linking to real-world data.
- **Ownable.sol** This contract acts as an extension of the ERC 721 contract and adds the safemint function. This function when called allows approved users to create a new token. The functionality within the contract is incredibly basic and therefore adjustments will need to be made to edit the functionality in line with the end goal.
- **Counters.sol** Again acting as an extension of the ERC 721 smart contract, the smart contract adds enumeration features, which allow for newly minted tokens to be given a unique id.
- **Enumerable.sol** This extension allows for on-chain enumeration of all tokens.
- **URIStorage.sol** This extension edits the safemint function, and the tokens as a whole by now permitting the passing of a IPFS URI. This extension allows the tokens to be linked to a object within the IPFS system and therefore adds the main functionality of the NFT system.

With the inheritance in place, the backbone of the project

```
{ "Attributes": [
{ "trait_type": "Artist",
"value": [ARTIST] },
{ "trait_type": "Genre",
"value": [GENRE] } ],
"description": [DESCRIPTION],
"image": [IMAGE CID],
"mp3": [MP3 CID],
"name": [TITLE] }
```

Listing 1: JSON example

is complete, and work can now begin on the project.

F. IPFS Implementation

The solidity blockchain is both slow and expensive and therefore the storage of the NFT files on chain is out of the question, therefore a workaround is needed. IPFS is a ”modular suite of protocols” that facilitates the organizing and transferring of data, in a decentralized fashion [9], and will be the protocol the project uses for file storage and sharing. The protocol is a decentralized and distributed file-sharing system, that creates a permanent and censor-resistant method for file sharing/storage. The interesting part of the protocol is its use of a content-addressable system in which files are identified and accessed based on content rather than location (such as on a server). IPFS will act as the file storage section of the project and will host the metadata and content of the NFTs.

There exist three components for a song, the mp3 (or file), the album art, and the metadata. In this section, I will describe the process by which the songs will be uploaded and minted as an NFT.

1) Through the use of a third-party API, the song's mp3 data and album art are uploaded to the IPFS system. The CID for both are returned and stored as variables.
2) A JSON object is created, containing the CID for both the mp3 and the album art, along with any relevant metadata. e.g. Title of the song, genre, ect.
3) The JSON object is uploaded to the IPFS system and the CID is returned and stored as a variable.
4) The SafeMint function is called and the CID of the JSON function is passed as a parameter.
5) The NFT is now created and can be shared and sold at will.

Every JSON file used within the project will follow the predefined structure, see Listing: 1. Which allows a given NFT to contain the files necessary for the end product.

Of note, this subsection outlines the added features required by the front end of the system, and therefore while brief, documents the design that will be further elaborated upon in the next section(s).

G. Auctions

This functionality will be hardcoded into the smart contract and will give users the ability to list/sell tokens and to buy up tokens. These listings will have two main options: 1) a fixed price listing, facilitating the transfer of a predefined amount of ether for a given token.

2) auction-based listing: within a given time frame buyers can bid on tokens, and once the time limit is up the token and money are transferred.

These functions are fairly simple and will be further elaborated upon in further sections.

H. General Use Case

What follows is the idea use case for both musicians and fans.

Musicians:

1) The musician has finished work on the demo and aims to sell it to a fan.
2) Through the use of the app, the music file and album art are uploaded to IPFS making use of an appropriate API
3) The musician clicks ”Mint” and Mints the metadata into an NFT, with the function returning the unique identifier for said token.
4) The musician is now able to sell
5) making use of the create_listing function the musician can create a listing of the given NFT specifying price and the type of auction.
6) If a fixed listing is chosen the listing is set at a given price, and once a buyer is found the payments will go through and the NFT ownership will be transferred. Should an auction be chosen, at the start of the first bid (supposing the bid is above the minimum price) an auction will begin, and once over the highest bidder will receive the NFT and the musician will receive the given price.

Users:

1) The user can first begin by looking through currently owned NFTs and current listings.
2) Should a listing take the owner's fancy, the owner will be able to buy the NFT with a listed amount of ether.
3) depending on the listing and the time frame, should a transaction be accepted the NFT will transfer ownership to the user, and will now be visible to the user.
4) through the use of IPFS the user can download the associated metadata, and enjoy the product.

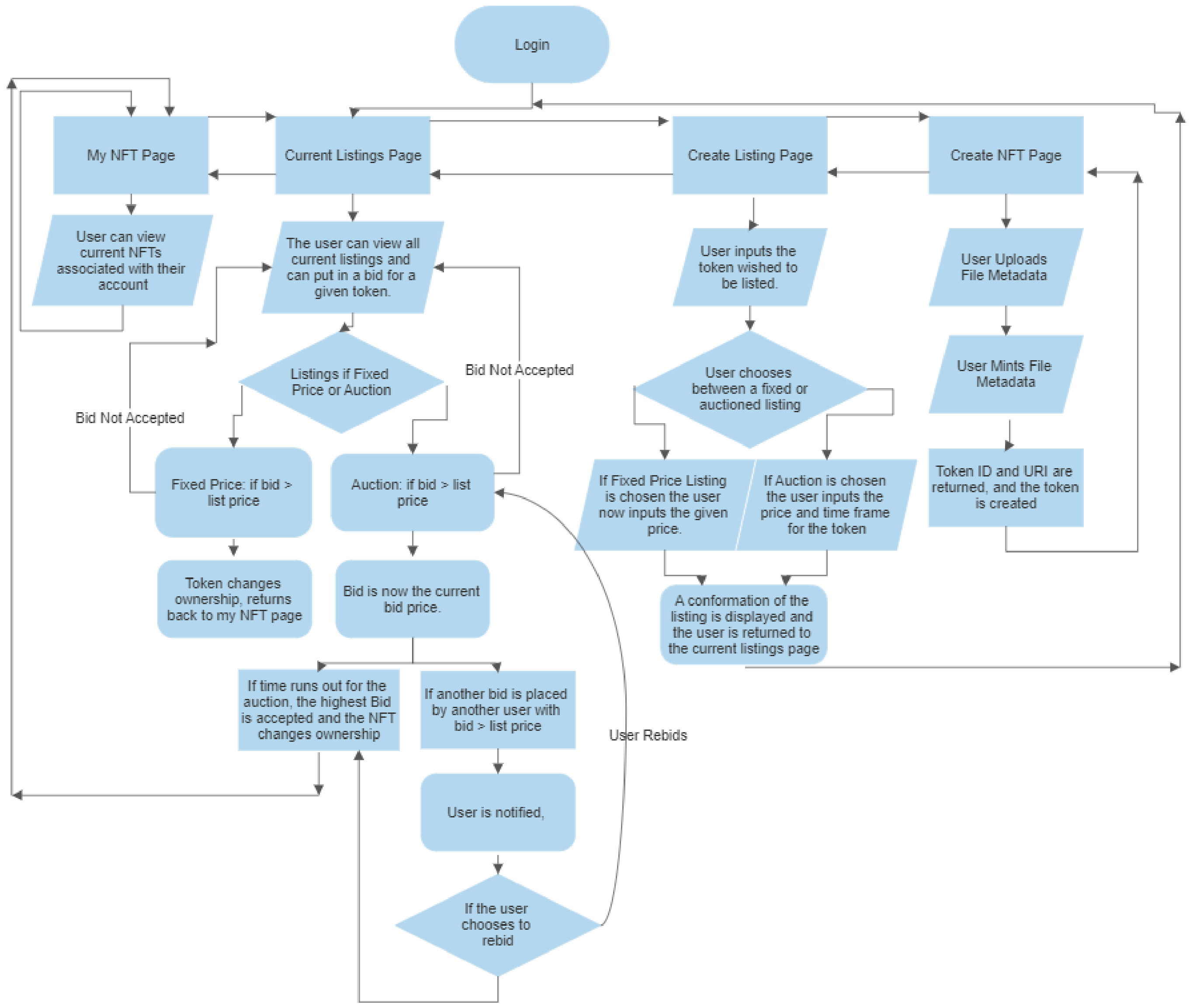
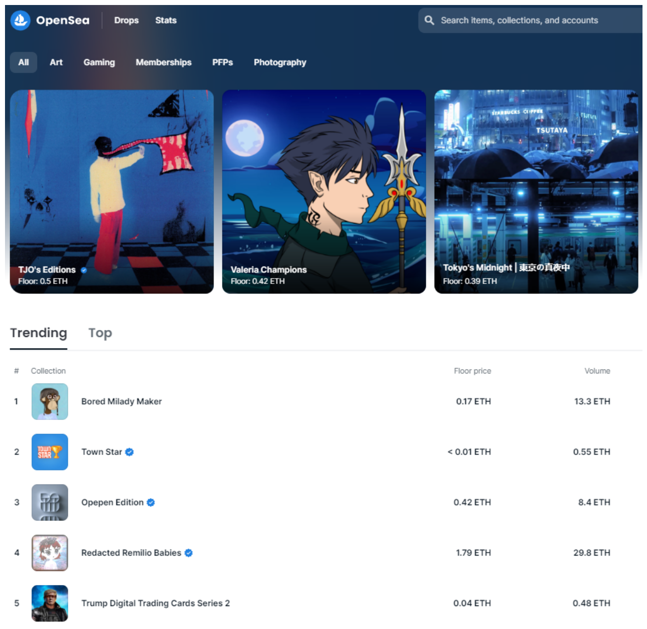

Login
My NFT Page
Current Listings Page
Create Listing Page
Create NFT Page
User can view current NFTs associated with their account
The user can view all current listings and can put in a bid for a given token.
User inputs the token wished to be listed.
User Uploads File Metadata
Listings if Fixed Price or Auction
Bid Not Accepted
Bid Not Accepted
User chooses between a fixed or auctioned listing
User Mints File Metadata
Fixed Price: if bid > list price
Auction: if bid > list price
If Fixed Price Listing is chosen the user now inputs the given price.
If Auction is chosen the user inputs the price and time frame for the token
Token ID and URI are returned, and the token is created
Token changes ownership, returns back to my NFT page
Bid is now the current bid price.
A conformation of the listing is displayed and the user is returned to the current listings page
If time runs out for the auction, the highest Bid is accepted and the NFT changes ownership
If another bid is placed by another user with bid > list price
User Rebids
User is notified,
If the user chooses to rebid

Fig. 1. Flowchart showing idea usage within the DApp

Fig. 1 shows a flow diagram illustrating the idea usage case within our project. Upon entering into the DApp, the user is presented with the current listings page. This is a deliberate decision and since so much of the project will revolve around listings and auctions of NFTs, the decision was made within the group to display this information at the forefront. This is in direct reference to the Web3 NFT Marketplace, Openseas [10]. Fig 2. shows the webpage presented to the user upon entry, and much like what is discussed within this section, the user is shown current and trending listings. The four pages at the top each represent a section of the front end, and therefore a part of the functionality of the DApp. Users will at any point within the app be able to freely move between these pages and explore further functionality within each section.

My NFT Page: This page is simply a listing of the current NFTs associated with the current account and allows the user to view both the tokens and the associated metadata. This page is overall the simplest and is returned to after a successful bid. In the current iteration of the project, the goal is for each NFT to have the associated metadata (i.e. Music and album art) displayed alongside the CID and token identifier, in order to help add legitimacy to the product. Further work in the front end will be explored to add streaming functionality within this page, meaning that a user can listen to their NFTs on app.

Current Listings Page: This page as suggested allows the users to view all current listings and from this allows the user to place a bid for a chosen token. Actings as the forefront for the DApp, the page will be split into two shows on one side fixed price listings, and on the other auction-based listings.



#	Collection	Floor price	Volume
1	Bored Milady Maker	0.17 ETH	13.3 ETH
2	Town Star	< 0.01 ETH	0.55 ETH
3	Opepen Edition	0.42 ETH	8.4 ETH
4	Redacted Remilio Babies	1.79 ETH	29.8 ETH
5	Trump Digital Trading Cards Series 2	0.04 ETH	0.48 ETH

Fig. 2. OpenSea Marketplace Website

The reasoning behind being to clearly show and display the different types of listings. Depending on the type of listing, the page will display the necessary information and some metadata from the tokens. For a fixed price listing, the listing will show the seller, price, album art, and a snippet of the music (i.e. the first 30 seconds), allowing the user to get a sense of the product they are purchasing. Also included will be a bid feature for each listing. For an auction-based listing, the same metrics will be used, however, the price indicates the current highest bid, and an added feature of time left is added to represent the time left within the auction. By choosing the NFT (based on the token ID) and by sending a defined of ether the user can bid on an NFT, and attempt to buy ownership. as presented in the flowchart there are two cases for this.
1) Fixed Price Listing: In this case, supposing the bid is sufficient, the ether is transferred and the NFT will change ownership, taking the user back to the ”My NFT Page”.
2) Auction Listing: Supposing the bid is higher than the current highest bid, (or listing price), the bid becomes the new current bid price. The transfer is far from complete however, and should another user bid on the token, the new bid will take priority, and the user will have to rebid in order to buy ownership of the token. This will be marked by a notification to any parties who have previously bid on the token. Once the time runs out for an auction-based listing, ownership is transferred to the highest bidder, for the listed bid price. The user is then taken to the ”My NFT Page” where the newly bought NFT will be listed and available to consume.

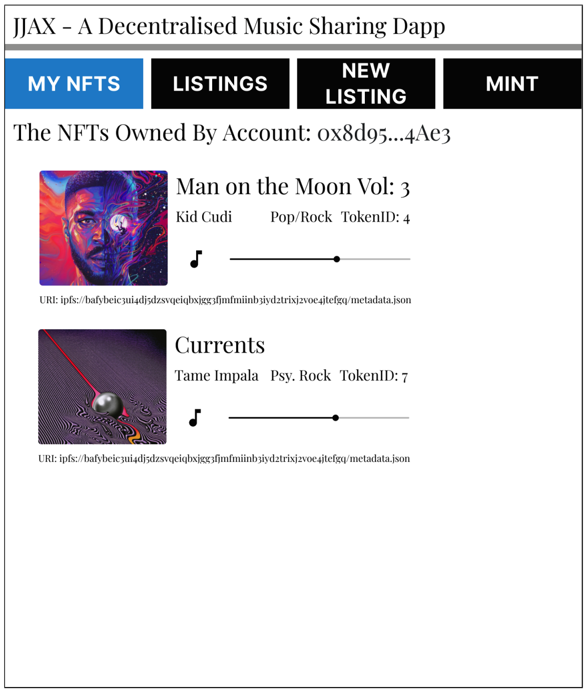
Create Listing Page: This exists for the (re)selling of tokens on the listing marketplace. The page will first take in the id of the token to be sold, to which ownership will be checked. Should the token-user information align, the user will be presented with the options for creating a fixed price or auction-based listing, to which relevant information will be imputed and the token will be listed on the current listings page. Of note the token should already be minted and have the associated metadata already contained, should the user want to create an NFT the ”Create NFT Page” Should be used.

Create NFT Page: This page acts as the basis for the project and allows for the creation of new NFTs within the system. First, the user will upload the album art and music, and the associated metadata (i.e. Title, Genre, Artist) will be manually imputed by the user. From this, the user is ready to mint. The mint function will take a small amount of ether (to discourage users from spamming the mint function) and will first, through the use of the NFT storage API, upload the album art and music file. Upon successful upload, a JSON file containing the relevant metadata and the CIDs of the music and album art will be uploaded. Through the use of the safemint function, the JSON URI will be supplied and minted, creating an NFT. The given token ID and URI will be supplied to the user, and the user is now free to do what they please with the token.

I. Data Flow?

J. User Interface

This section will highlight the initial idea for the DApp UI and may be subject to changes within the final execution. The following figures act as a mockup presented to the group in order to justify front-end ideals. While likely the finished product will be somewhat different from these mockups they act as the front end in an ideal solution, and are used more to justify further front-end design. fig 3,4,5 & 6 showcase the initial UI designed for the project. As apparent the UI is made to be as minimal as possible in order to show off the NFTs. The four buttons on the top allow for navigation and directly correspond to the user interaction flowchart, see fig. 1. Within the NFT page (fig. 3) the user tokens are displayed, with the album art, title, genre, token ID and IPFS link being clearly displayed to the user. Along with this, a music bar is added allowing users to listen to purchased music. The IPFS URI is included to allow users to access and download the metadata associated with the file, in effect allowing the user to consume the media purchased.
Fig 4. Follows simple design principles, showing available listings with the list price in a large font allowing for easy visual communication with the user. The listings are separated into Fixed price and Auction based listings to avoid user confusion, with the use of a grey line to solidly separate the types.
Lastly, Fig 5 and 6 (The New Listings and Mint Functions) follow the same general design, any input fields are highlighted in grey, explaining their functionality, and the function button (Begin listing and MINT) are given in bold colors to indicate separation from the rest of the page. The goal of these

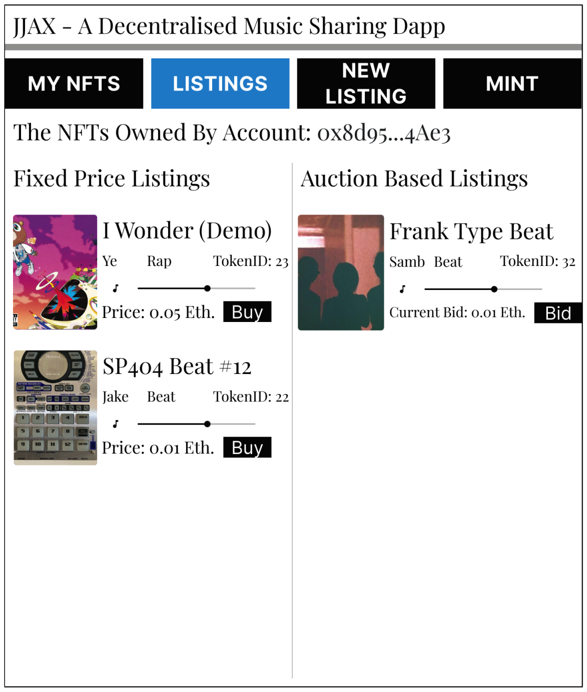


The NFTs Owned By Account: 0x8d95...4Ae3

Man on the Moon Vol: 3
Kid Cudi Pop/Rock TokenID: 4
URI: ipfs://bafybeic3ui4dj5dzsvqeiqbxjgg3fjmfmiinb3iyd2trixj2voe4jtefgq/metadata.json

Currents
Tame Impala Psy. Rock TokenID: 7
URI: ipfs://bafybeic3ui4dj5dzsvqeiqbxjgg3fjmfmiinb3iyd2trixj2voe4jtefgq/metadata.json

Fig. 3. My NFTs Page Mockup



The NFTs Owned By Account: 0x8d95...4Ae3

Fixed Price Listings

I Wonder (Demo)
Ye Rap TokenID: 23
Price: 0.05 Eth. Buy

SP404 Beat #12
Jake Beat TokenID: 22
Price: 0.01 Eth. Buy

Auction Based Listings

Frank Type Beat
Samb Beat TokenID: 32
Current Bid: 0.01 Eth. Bid

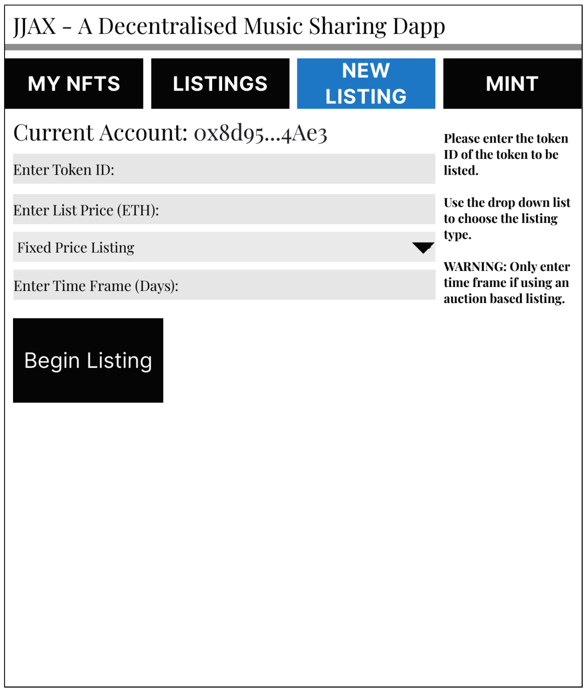
Fig. 4. Listings Page Mockup

mockups is to create an easy visual language to be explored by the user.

IV. Project Execution

A. Minting and Transfer Functionality

Referring back to section 1E (The ERC 721 Standard), the main minting and transfer functions are created by modifying

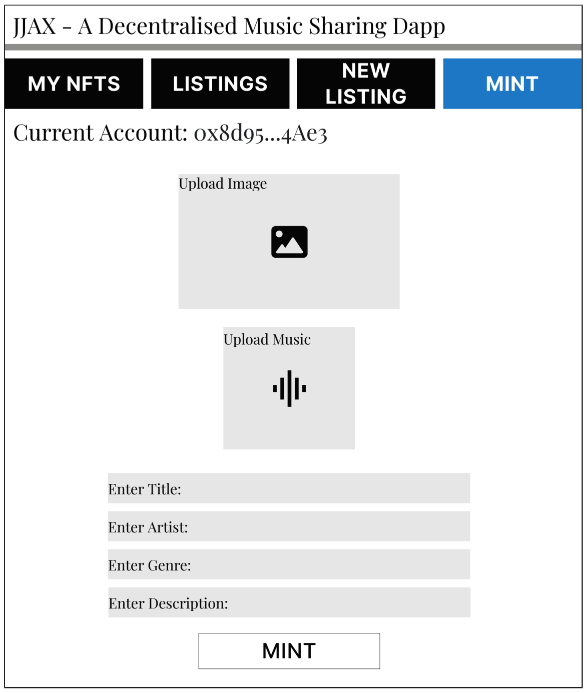


Current Account: 0x8d95...4Ae3

Enter Token ID:
Enter List Price (ETH):
Fixed Price Listing
Enter Time Frame (Days):
Begin Listing

Please enter the token ID of the token to be listed.
Use the drop down list to choose the listing type.
WARNING: Only enter time frame if using an auction based listing.

Fig. 5. New Listings Page Mockup



Current Account: 0x8d95...4Ae3

Upload Image
Upload Music
Enter Title:
Enter Artist:
Enter Genre:
Enter Description:
MINT

Fig. 6. Mint Page Mockup

and combining OpenZeppelin Functions in order to create the functionality required by the app. For the sake of brevity, the ERC functions won't be reiterated, however, will be referenced in the code analysis.

```
contract JJAX2 is ERC721, ERC721Enumerable,
ERC721URIStorage, ERC721Burnable, Ownable {
    using Counters for Counters.Counter;
```

The above code is used to initialize the contract, inheriting from several OpenZepplin smart contracts. Along with this, the OpenZeppelin Counters library is it is used to keep track of the token IDs assigned to the NFTs in the contract.

With these libraries alone, the smart contract has the functionality to create and transfer tokens, however, lacks the ability to mint tokens with the IPFS URI and lacks any sort of monetary transfer within the contract.

For the main minting functionality, two constants are declared:

```
uint256 public MINT_PRICE = 0 ether;
uint256 public MAX_SUPPLY = 100;
```

This allows for a Max Supply of NFTs within the app and adds the ability to charge when minting an NFT, and both will be referenced by the mint function.

```
function safeMint(address to, string
memory uri) public payable
```

The safeMint function acts as the backbone of the project and is used to create an NFT within the contract. Two checks are used to ensure the max supply of tokens isn't met and that the msg contains enough ether to mint the function. The _tokenIdCounter function is used in conjuncture with the counters library to get the current tokenID, and increments. The _safemint function is passed the user address and tokenID, and the new token is minted. This however lacks the IPFS integration so the _setTokenURI function is used (from ERC721URIStorage.sol). This function takes the tokenID and URI, and associated the given token with the IPFS metadata allowing for integration. This URI will be associated with a JSON file following the general design seen in Listing 1.
The smart contract now has the functionality to mint tokens with IPFS data, however, lacks any sort of auction functionality, which will be discussed in the following section.

B. Listing/ Auction Functionality

Stuctures Enum ListingType Struct Listing Struct Auction Struct Auctionview

Fixed Price Listings: These listings make use of the Listing struct which specifies the general form needed. This struct takes in tokenID and the Price (as uint256) and the address of the seller, and is used as a convenient way to store the listing information. Along with this the mapping

```
mapping(uint256 => Listing) public
fixedPriceListings;
```

is used to map each respective listing and purchasing process. Three functions are used within the fixed-price listings to facilitate the creation, buying, and viewing of the tokens.
The **createFixedPriceListing** takes the tokenID and Price as parameters. This function is incredibly simple and makes three checks being 1) the msg sender owns the token, 2) the token isnt already listed, and 3) the token isn't in the auction. Should the requirements be met, the fixedPriceListings mapping is passed the tokenId and associated with a listing Struct containing the relevant information.
The **buyFixedPriceListing** function takes in the TokenID as a parameter. This function first access and stores the listing information and makes two checks, checking first if the listing exists and second if the msg.value contains enough ether. Should the requirements be met, the money is transfered and the nft is transferred through the _transfer function (From ERC721.sol). After the fixedPriceListing associated with the tokenID is deleted, removing the listing.
Lastly, the **ViewFixedPriceListings** function can be used to view current listings. Due to the use of the counters.sol and ERC721Enumerable.sol the smart contract has an on-chain enumeration of all tokens, therefore to find current listings, the possible tokenIDs are iterated through, and should a corresponding listing exist, said listing is returned within a list. This is fairly inefficient and is one of the main reasons for a max supply of tokens, however, is the only viable solution found.

Auctions follow the same general structure as the fixed price listings, however, have added complexity which will be discussed within this section. A struct is used again to store the data associated with each Auction, and a mapping is used to associate each tokenID with the given auction struct. The Auction struct has the form:

```
    struct Auction {
        uint256 tokenId;
        uint256 startTime;
        uint256 bidStartTime;
        uint256 duration;
        uint256 highestBid;
        address highestBidder;
        address seller;
        bool hasStarted;
    }
```

This struct doesn't include the time left within the auction since this needs to be calculated dynamically, therefore a separate struct (AuctionView) is used, being nearly identical to the Auction struct but with an added uint256 timeLeft. Along with this a modifier is used which is called whenever a listing/auction function is called that checks the given token, and calculates the time left. Should there be no time left, the endAuction function is called and the auction ended.
To create a new auction the **CreateAuction** function is used, taking the tokenId, minPrice and auctionDuration as parameters. This follows the same structure as the listing version,

making the required checks and creating the mapping between the tokenID and the Auction listing.
The **bidOnAuction** function is used to allow buyers to bid on a given token. Past requirements checking the msg.value and the auction parameters. If the bidding has already begun, the highest bidder is returned their bid, and the new user takes their place. If the bidding has not started, the bidStartTime is set to the current timestamp and the auction parameter hasStarted is set to True.
The **endAuction** is used to end auctions once the time is up. This checks the auction time is up, and should the highestBidder be different from the owner, the token transfers ownership, and the highest bid is sent to the original owner. Should there be no bid placed, the auction is simply deleted, with no transfer of cash.
Lastly, the **viewAuctions** function is used to visualize current auctions and follows the same general structure as the listing equivalent. By looping through tokenIDs current auctions are identified, and the relevant auction struct is called. From this the timeLeft is calculated, and the auction and timeLeft fields are added to an AuctionView struct, which is appended to a list. After the tokens have been iterated the results are returned, showing current listings as an AuctionView struct.
This marks the end of the auction functionality within the smart contract, and as a current iteration contains all the functionality required by the project.

C. Uploading Functionality/ Front End

Elaboration upon section 1F (IPFS Implementation) this section will discuss the use of the NFT Storage API to upload to the IPFS network.
There are several reasons for the use of the NFT storage API within the application, however, the main being that 1) the API is simple to use and requires little IPFS knowledge to implement and 2) It is free to use. Following from point 2) the original implementation for the front end used the Pinata API [11] however for the functionality required a cost of £20 per month was needed, and therefore alternate solutions were identified.

D. Front End UI:

While far different from the "Model implementation" seen in section 2J, the user interface is presented as follows:

E. Testing Procedure

F. User Guide

For users intending to use the product, please refer to fig.1. This flowchart details the possible interactions between the user and the machine, and gives a good overview of how the user will use the product.

V. Critical Evaluation

A. Problems with NFTs

Overall the project has some great functionality (particularly within the backend) which allows for a robust and efficient solution to the problem at hand. This is, however, not without some caveats. The project is built upon the standards set out by the ERC 721 "Non-Fungible Standard", and in particular makes use of functionality within the OpenZeppelin implementation of said standard (see section 2E). These libraries are well-tested and therefore the project's smart contract can make use of the functionality without risking the security of the project. There is however issues within the project's implementation of said libraries. As all the payment and transfer functionality is built upon the ERC 721 technology, there is no real risk of reentrancy or front-running attacks. However, the project runs into a big issue in regard to what will be referred to as "double minting". This is a term described in the Medium article "Solving the NFT Double Minting Problem With Computer Vision" [12] and describes the issue seen within the project. In essence, there are no checks in place to stop a musician from minting multiple tokens with the same metadata, and should a user do so, will likely degrade the value of said tokens and takes away from the "uniqueness" described within the idea brief (See section 3C). Building upon this issue is the idea of NFT fraud. Although several types of fraud exist, the main types are rug-pulls and Plagiarized NFTs. Rug-pull scams are fairly simple and work as follows: A user will upload an NFT and mint said NFT with a URI linking to the associated metadata, in this scam the scammer will have full control over the URI and or server hosting the metadata, and on purchase of the NFT, the scammer will change the metadata, meaning that the customer will receive either an empty token or an NFT they didn't purchase. This is a very common scam with examples such as the "Frosties freeze rug pull" scamming consumers out of the equivalent of 1.3 million [13]. This however is unable to happen within the project due to the use of the IPFS system. Since IPFS works through the use of a decentralized CID system, the content associated with each NFT is tied to the CID, meaning the files will be immutable. However the project is susceptible to plagiarized NFTs, much like with double spending there is again no copyright checks to indicate ownership of the metadata to be minted, and therefore with continued use the project is likely to run into issues with plagiarism. This is a difficult issue to solve with OpenSea stating (in regards to their free NFT minting tool) "Over 80% of the items created with [the] tool were plagiarized works, fake collections, and spam." While this is a big issue within the project, further academic research is required before a suitable solution can be identified and presented. Due to the newness of the NFT technology, there are several problems within the smart contract, that may see abuse by the community. While work has been done to mitigate issues within the project, due to the scope and size of the project, there are a few flaws within the working logic that may lead to long-term consequences.

B.

(safe because of erc 721) (security) Wash Trading (gas usage)
(double uploading)
(problems with NFTs)

VI. Conclusion

References

[1]

[2]

[3] https://bandcamp.com/about

[4] https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[5] https://eips.ethereum.org/EIPS/eip-721

[6] https://ethereum.org/en/developers/docs/standards/tokens/erc-1155/

[7] https://eips.ethereum.org/EIPS/eip-998

[8] https://docs.openzeppelin.com/contracts/2.x/api/token/erc721

[9] https://docs.ipfs.tech/concepts/what-is-ipfs/defining-ipfs

[10] https://www.google.com/search?q=openseas+nftrlz=1C1ONGR$_en - GBGB938GB938oq = openseasaqs = chrome.0.69i59j46i199i465i512j0i512j46i10i512j46i10i199i465i512j0i512l2j69i60.1663j0j7sourceid = chromeie = UTF - 8https : //nft.storage/docs/$

[11] https://www.pinata.cloud/

[12] https://medium.com/geekculture/solving-the-nft-double-minting-problem-with-computer-vision-c57bbbb4652d

[13] https://nftnow.com/features/the-biggest-rug-pulls-in-nft-history/

[14] OpenSea @opensea. (2022) [Twitter] Jan 27th. Available at: https://twitter.com/opensea/status/1486843204062236676 (15/05/2023)