**The convenience of NFTs and an NFT marketplace**

NFTs seem to have come a long way from the first ever minted NFT by artist Kevin McCoy back in 2014. At that time, NFTs’ use case was more or less seen as confined to digital art. Fast forward to 2019 when the National Basketball Association (“NBA”), the NBA Players Association and Dapper Labs came together and collaborated to create NBA Top Shot, which they describe as a “revolutionary new experience in which jaw-dropping plays and unforgettable highlights become collectibles that you can own forever”. To put it in simpler terms, this is just about digitized or virtual trading cards but instead of just having a picture of the player and his career stats, each virtual card will show a particular play or highlight like for example a dagger shot that wins a game. (Source: <https://ca.nba.com/news/what-is-nba-top-shot-explaining-the-blockchain-nba-highlight-collectables/18nram5ye1ub01hres3lkk3xvd#:~:text=In%20a%20joint%20venture%20that,you%20can%20own%20forever%2C%22%20per>)

The next question would have been beyond digital art and virtual trading cards, what are NFTs’ other use cases? Perhaps the correct way of answering that question is to begin with a discussion on what NFTs are. An NFT – short for “non-fungible token” – is basically a digital file with verified identity and ownership. It is in the verification part where blockchain is involved. As everyone would know, blockchain technology is an un-hackable system based on the mathematics of cryptography. It is non-fungible because it is an asset that is not interchangeable because it is unique (unlike money and Bitcoins). The value of an NFT lies in providing the ability to securely value, purchase and sell digital art using a ledger that is open to everyone. One problem that should be addressed is the issue on a marketplace where a secured transfer of NFTs may be made.

**The Project**

The Project is about how to build a blockchain that is more general purpose and a better fit for the NFT use cases using IPFS as a storing and sharing. This smart contracts in this Project will be built on the Flow blockchain.

**Process flow**

Summary:

1. NFT minting process – creating the smart contract and minting a token;
2. Creating an app to view NFTs created through the smart contract
3. Backing the NFT metadata and assets on IPFS – creating a marketplace to transfer NFTs to others while also transferring the NFT’s underlying assets on IPFS

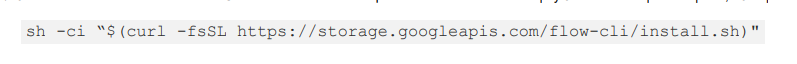
Step-by-step:

1. **Minting process**
2. Setting up
3. Install Flow CLI

* macOS :

(1A)

* Linux:

(1B)

* Windows



(1C)

1. Sign up for a Pinata account and grab API Key. This will make storing asset files on IPFS easier.
2. Install NodeJS. A text editor is also needed that can help with syntax highlighting of Flow smart contract code. Note that the smart contract code is written in Cadence. VS Code has an extension that supports Cadence.
3. Create a directory for the project.

(1D)

1. Change to that directory and initialize a new Flow project.

Text

Description automatically generated(1E)

1. Open project in code editor.
2. In the “flow.json” file, create a folder “cadence”. Within “cadence” create a folder “contracts”. Within, “contracts”, create a file called “PinataPartyContract.cdc”.
3. From this point, everything that will be done in relation to the Flow blockchain will be done on the emulator.
4. Set up the file for the emulator environment.
5. Start writing the smart contract.
6. Writing the smart contract

The contract is able to do the following: (a) mint NFTs, (b) associate metadata to the NFT, and (c) ensure that metadate points to the underlying asset stored on IPFS.

1. Open the file “PinataPartyContract.cdc”.
2. Define the contract and within that create a “resource”.
3. Create a resource interface to define the capabilities made available to others – i.e. people who are not the contract owner. Put the following under the NFT “resource” code:

Graphical user interface, application

Description automatically generated(1F)

1. Define the token collection interface. In this resource, create these two variables: (a) “ownedNFTs” and (b) “metadataObjs”. Initialise these variables.
2. Create a “collection” resource and an “NFTMinter” resource. Below the “NFTMinter” resource, add the main contract initialiser which is called only when the contract is deployed.
3. Deploy the contract and run the following from the command line:

Text

Description automatically generated with low confidence(1G)

1. Then run this command:

A picture containing text

Description automatically generated(1H)

1. Minting the NFT

Cadence scripts and the command line will be used to mint and show how metadata will work with the NFTs on Flow.

1. Create a new directory within the root of the “pinata-party” project and call it “transactions”. Within “transactions”, create a new file called “MintPinataParty.cdc”.
2. Upload a file to IPFS via Pinata, which in this project is XXX. To be able to write a transaction, a file is needed to reference in the metadata provided to the NFT. Once a file is uploaded, an IPFS hash will be provided (often referred to as a content identifier or CID). Copy the has as this will be used in the minting process.
3. Inside “MintPinataParty.cdc”, add the following:
4. Replace the account received with the account address from the deployment.
5. Define the transaction – define the “receiverRef” and “minterRef” variables.
6. “Prepare” function – this takes the account information of the person executing the transaction and does validation.
7. “Execute” function - where the metadate is built for the NFT, the NFT is minted, and where the metadata is associated before depositing the NFT in the account.
8. Prepare the account – create a private key for signing. Run the following:

Logo

Description automatically generated with medium confidence(1I)

This will provide you with a public and a private key. The private key will be used to sign the transaction. The signing algorithm should also be specified.

1. Send the transaction using the following command:

Text

Description automatically generated with low confidence(1J)

1. Verify if the token is in the account and fetch the metadata.
2. From the root of the project, create a new folder called “scripts”, then create inside of that a file called “CheckTokenMetadata.cdc”.
3. In the “script”, the contract is imported from the deployed address. This is actually defining a “main” function which is the required function name for a script to run.
4. Inside the “main” function, define the following:
5. nftOwner – the account that owns the NFT
6. capability – there is a need to “borrow” the available capabilities or functions from the deployed contact – capabilities from the “NFTReceiver” resource.
7. receiverRef – this variable takes the capability and tells the script to borrow from the deployed contract.
8. Run the script. Run the following on the command line:

(1K)

1. Building the React application

This part relates to:

1. the steps on how to build an application that interfaces with the Flow smart contract created in Part I of this Project to authenticate and to fetch NFTs owned by a user; and
2. resolve the NFT’s metadata to get the IPFS location of the NFT’s underlying digital asset.
3. Setting Up
4. Ensure Flow emulator is running.
5. Ensure NodeJS is installed.
6. Setting up React and Dependencies
7. Create the app in a directory. (in this case, the app will be created in the “pinata-party” directory.
8. Run:

Graphical user interface, text

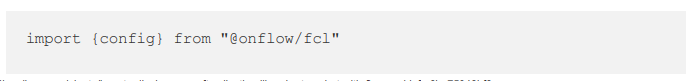
Description automatically generated with medium confidence(2A)

1. Switch to the new directory that will be created.
2. Install the following dependencies:
3. Flow JS SDK. Run the following:

A picture containing logo

Description automatically generated(2B)

1. Create an “.env” file and set key value pairs where the keys are prefixed with “REACT\_APP”.
2. Create a file in the “src” directory called “config.js” and add the following:



Text, Word

Description automatically generated

(2C and 2D)

1. To make the configuration file available throughout the app, open up the “index.js” file and add this line:

Chart, scatter chart

Description automatically generated

1. Authentication – this is relevant in transferring NFT assets.
2. Create a file called “AuthCluster.js” file in the “src” directory.
3. Replace the “App.js” file with the following:

Graphical user interface, text, application

Description automatically generated(2F)

1. Fetching NFTs from Flow

The objective here is to be able to communicate with the Flow emulator using JavaScript.

1. Create a new component that will allow to fetch both data and display the NFT data.
2. Create a file called “TokenData.js” in the “src” directory. In this file a component is created that has buttons that (1) fetch the token data and (2) clear token data. The function “fetchTokenData” is called when the button is clicked and it uses the Flow JS SDK to execute the script executed in Part I of this ReadMe file from the command line, but in React. The result of the execution will be set to a state variable called “nftInfo”.
3. Add CSS by replaing “App.css” with adding a new component to “App.js” below the “AuthCluster” component like the following:

Text, letter

Description automatically generated(2G)

1. Check the app and fetch the token data.
2. Getting media from IPFS
3. Update the “TokenData.js” file as follows:
4. “We’ve added a video element with the source pointing to the file on IPFS. The “uri” value was split to get the IPFS hash to enable fetching the content from the UPFS gateway.
5. Note on “uri” – one that was created looks like “ipfs://Qm...”, the reason being the IPFS desktop client will, by default, allow you to click on and open the link that looks like that. The Brave browser likewise supports pasting in links that look like that. However, what is need in this case is the hash so that content from the IPFS gateway can be fetched. The link should look something like this:

(2H)

1. Creating an NFT Marketplace on Flow with IPFS

The purpose of creating a marketplace is to be able to transfer NFTS.

1. Setting Up
2. Open the project and ensure emulator is running.
3. Creating contracts
4. Marketplace will require:
5. Payment mechanism (i.e. fungible tokens)
6. Token transfer capabilities
7. Token supply settings
8. Minting and transferring tokens
9. Go into the “cadence/contracts” folder under the “pinata-party” directory and create a new file called “PinnieToken.cdc”, which will the fungible token.
10. Define the empty contract as follows:

Graphical user interface, text, application

Description automatically generated(3A)

1. Add the following inside the empty contract:

Graphical user interface, text, application, email

Description automatically generated(3B)

The resource interface “Provider” defines a function that is public, but it will still be callable only by the account owner from whom the withdraw is being executed against.

1. Define the “Receiver” and “Balance” public resource interfaces as follows:

Graphical user interface, application

Description automatically generated(3C)

The “Receiver” interface includes a function that can be executable by anyone. The “Balance” resource will return a balance of the new token for any given account.

1. Create a “Vault” resource by adding the following below the “Balance” resource:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated(3D and 3E)

The “Vault” resource is important because without it, the account will be unable to receive particular tokens, which also means that the account will be unable to send tokens as well. If this is the case, the account will be unable to buy NFTs.

1. Ensure account gets access to the “Vault” interface by adding the following:

Text

Description automatically generated with low confidence(3F)

1. Set up minting capability by adding the following below the “createEmptyVault” function:

Graphical user interface, text, application

Description automatically generated(3G)

1. Initialise the contract by adding the following after the “VaultMinter” resource:

Graphical user interface, text, application

Description automatically generated(3H)

1. Deploying and minting tokens
2. Updating the “flow.json” file
3. Ensure that the “flow.json” references the new contract and has the “emulator-account” key reference as follows:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

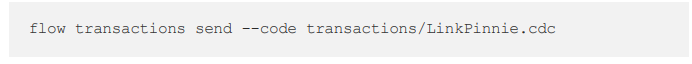
Description automatically generated(3I and 3J)

1. Run “flowprojectdeploy” from within the “pinata-party” project directory in another terminal window.
2. Test the minting function. Update the “flow.json”. Under the “emulator-account”, change the json back to the following:

Graphical user interface, text, application

Description automatically generated(3K)

1. Create a link.
2. Add a file called “LinkPinnie.cdc” file inside the transactions folder. After the transaction is created, run it by adding the terminal the following at the root of the project:

(3L)

1. Mint
2. Add the following code to a file called “MintPinnie.cdc” within the transactions folder and run:

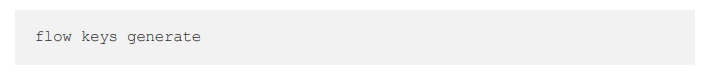
Graphical user interface, text

Description automatically generated(3M)

1. Create a script to check balance by creating inside the scripts folder of the project a file called “CheckPinnieBalance.cdc”. Run the script with the following command:

(3N)

1. Generate a new keypair by running the following:

(3O)

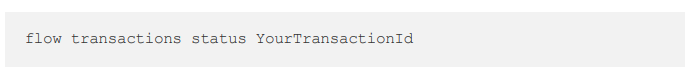
Use the public key to generate a new account and use the private key to update “flow.json”.

1. Create a new account by running the following:

Graphical user interface, text, application

Description automatically generated(3P)

1. Copy the transaction ID generated from running the above command, and copy the transaction ID and run the following:

(3Q)

The outcome should provide a new account address which will be used to update the “flow.json” file.

1. Use the second account to send Pinnie tokens.
2. Sending fungible tokens
3. Create an empty vault by creating a file called “CreateEmptyVault.cdc” inside the “transactions” folder.
4. Run the transaction in the root of the project:

A picture containing graphical user interface

Description automatically generated(3R)

1. Link to the Pinnie Token resource by running the following:

Graphical user interface, text

Description automatically generated with medium confidence(3S)

1. Transfer tokens from the “emulator-account” to the “second-account” by creating a file called “TransferPinnieToken.cdc” inside the “transactions” folder.
2. Execute the transaction by running the following:

Graphical user interface, text, application

Description automatically generated(3T)

1. Prove the two accounts with tokens by opening the “CheckPinnieBalance” script and replacing the account address on line 3 with the address for the “second-account”. Save and run the script as per below:

Graphical user interface

Description automatically generated with medium confidence(3U)

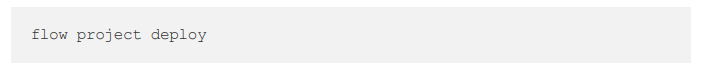
1. Creating the marketplace
2. Create a contract that can handle the marketplace creation and management.
3. Create a new file called “MarketplaceContract.cdc” inside the “cadence/contracts” folder. Add the following to the file:

“We are importing both our NFT contract and our fungible token contract as they will both work in conjunction with this marketplace contract. Inside the contract definition, we define four events: ForSale (indicates the NFT is for sale), PriceChanged (indicates a change for the NFT), TokenPurchased (indicates an NFT was purchased), and SaleWithdrawn (indicates an NFT was removed from the marketplace).”

1. Below the event emitters, there is a resource interface called “SalePublic”. Below this, add a “SaleCollection” resource.
2. Within the “SaleCollection” resource, define the following variables:
   * “forSale” – mapping of tokens for sale
   * “prices” - mapping of prices for each token for sale
   * “ownerVault” – protected variable that can only be accessed by the contract owner
3. Initialise the above variables.
4. Define the following functions:

* withdraw
* listForSale
* changePrice
* purchase
* idPrice
* getIDs
* destroy

1. Final part is of the marketplace function is “createSaleCollection” function.
2. Deploy the contract in the emulator account by running the following:

(3V)

1. Updating the frontend
2. Change to the “frontend” directory in the project and look at the “App.js” file.
3. Open the “TokenData.js” file and replace everything with the following:

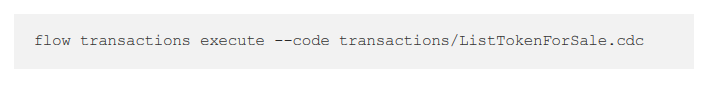
“We’re hard coding some values so in a real app be sure to think about how to dramatically get information like the account addresses. Notice that in the “checkMarketplace” function, everything is wrapped in a try/catch.”

A “NO NFTSs FOR SALE” will appear if you start your frontend app by changing into the frontend directory and running “npmstart”. This will need to be fixed.

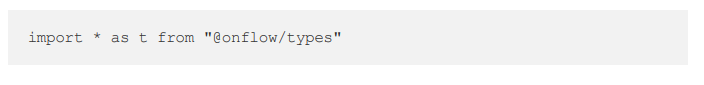
1. Listing the minted NFT for sale via Flow CLI tool
2. In the root pinata-party project and inside the “transactions” folder, create a file called “ListTokenForSale.cdc” and add the following:

“In this transaction, the three contracts created will be imported. The PinnieToken receiver is required for accepting payment in PinnieTokens. Access to the “createSaleCollection” function on the MarketPlace contract is also required.

1. Reference the NFT that will be put up for sale. Withdraw the NFT, list of for sale and save it into the NFTSale storage path.
2. Run the following to be able to list the NFT minted earlier:

(3W)

1. Go back to the React App page and refresh.
2. Before adding code to the React App, add the following import to the top of the “TokenData.js” file to allow to pass in arguments to the scripts we send using fcl:

(3X)

1. Inside the “TokenData.js” file and within the “checkMarketplace” function, add the following after the decoded variable:

“In the console, you will see an array of metadata associated specifically with the tokens that are for sale. The last thing needed to be done before we can render anything is to find out the prices for the listed tokens”.

1. Right below the “decodedMetadat” variable and before the “marketplaceMetadata.push(decodedMetadata)” function, add the following:

“We are getting the price of the each NFT that has been listed and when the price is received, we add it to the token metadata before pushing that metadata into the “marketplaceMetadata” array.

1. Render listed tokens and show the price
2. Under the console.lg statement showing the “marketplaceMetadata” array, add the following:

Graphical user interface

Description automatically generated(3Y)

1. Add the following right below the start of the “TokenData” main function declaration:

Graphical user interface, text, application, Word

Description automatically generated(3Z)

1. In the return statement, add something like the following:

Graphical user interface, text, application

Description automatically generated(3AA)

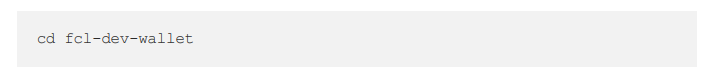
1. Buying the NFT

Note that we are working with a local Flow emulator and therefore, we need to run a local dev wallet and update the React app’s environment variables.

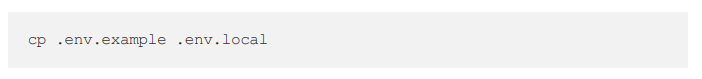
1. Clone the local dev wallet. From the root of the pinata-party project, run the following:

(3BB)

1. Change into the folder:

(3CC)

1. Copy the sample env file and create local env file that the dev wallet will use:

(3DD)

1. Install dependencies:

Rectangle

Description automatically generated with medium confidence(3EE)

1. Open the “.env.local” file.

Note that you will find references to an account and a private key. Remember that a new account was created earlier that would be buying the NFT from the marketplace. Change the account in the “.env.local” file to match the new account that was created. Change the private and public keys too. For the “FLOW\_ACCOUNT\_KEY\_ID” environment variable, change it to 1. The emulator-account is 0.

1. Run the “npm run dev” to start the wallet server.
2. In the “frontend” directory of the project, find the “.env” file and update “REACT\_APP\_WALLET\_DISCOVERY” to point to <http://localhost:3000/fcl/authz>.
3. Restart the React app.
4. Wire the frontend Buy Now button to send a transaction to purchase the token. Open the “TokenData.js” file and create a buyToken function like the following:

“Add an “onClick” handler for the Buy Now button. Update the button to look like this:

A picture containing graphical user interface

Description automatically generated(3FF)

1. Go to the React app and click the Buy Now button.