NFT Marketplace & Generator The 21st century Picasso

The future of ART!

Joel Carballo, Dan Ladner, Anthony Barone,
& Babajide Ademola

Technology used

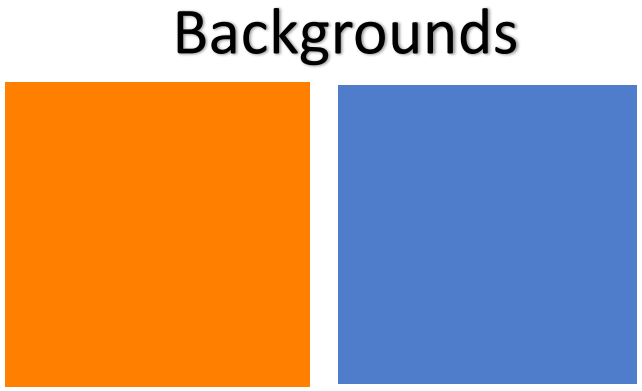
- Packages:
  - PIL from Image
  - Ipython from display
  - Random
  - Json

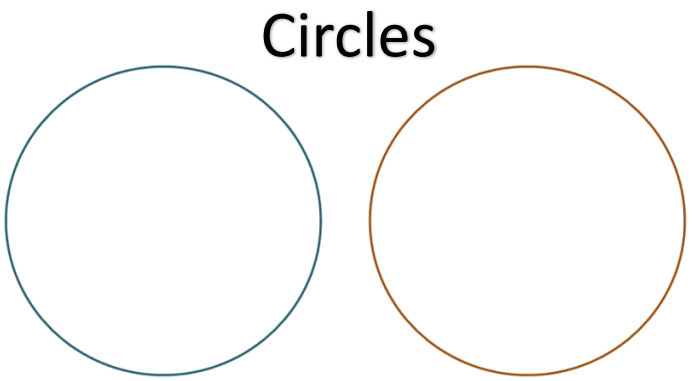
- Jupyter Lab
- Power shell
- NexJS polygon
- Metamask
- Web3modal
- Solidarity
- JavaScript
- Open zeppelin
- Tokenonmics
- Hard hat

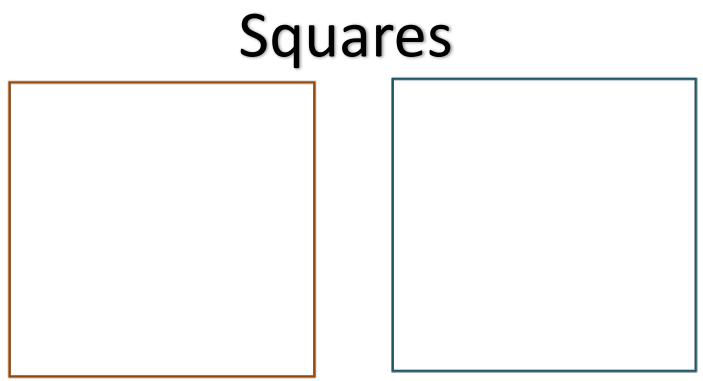
Challenges

Summary – NFT Generator

- NFT Generator: Takes in a variety of traits and randomly combines them into a unique photo. The more traits the more unique possibilities.
  - Crypto Punks: 10,000 unique Punks (6,039 males / 3,840 females)
  - 3,200 owners / 728.9k ETH (volume traded)
  - Launched in mid -2017 and become one of the inspiration for the ERC-721 standard.







Code/Solutions

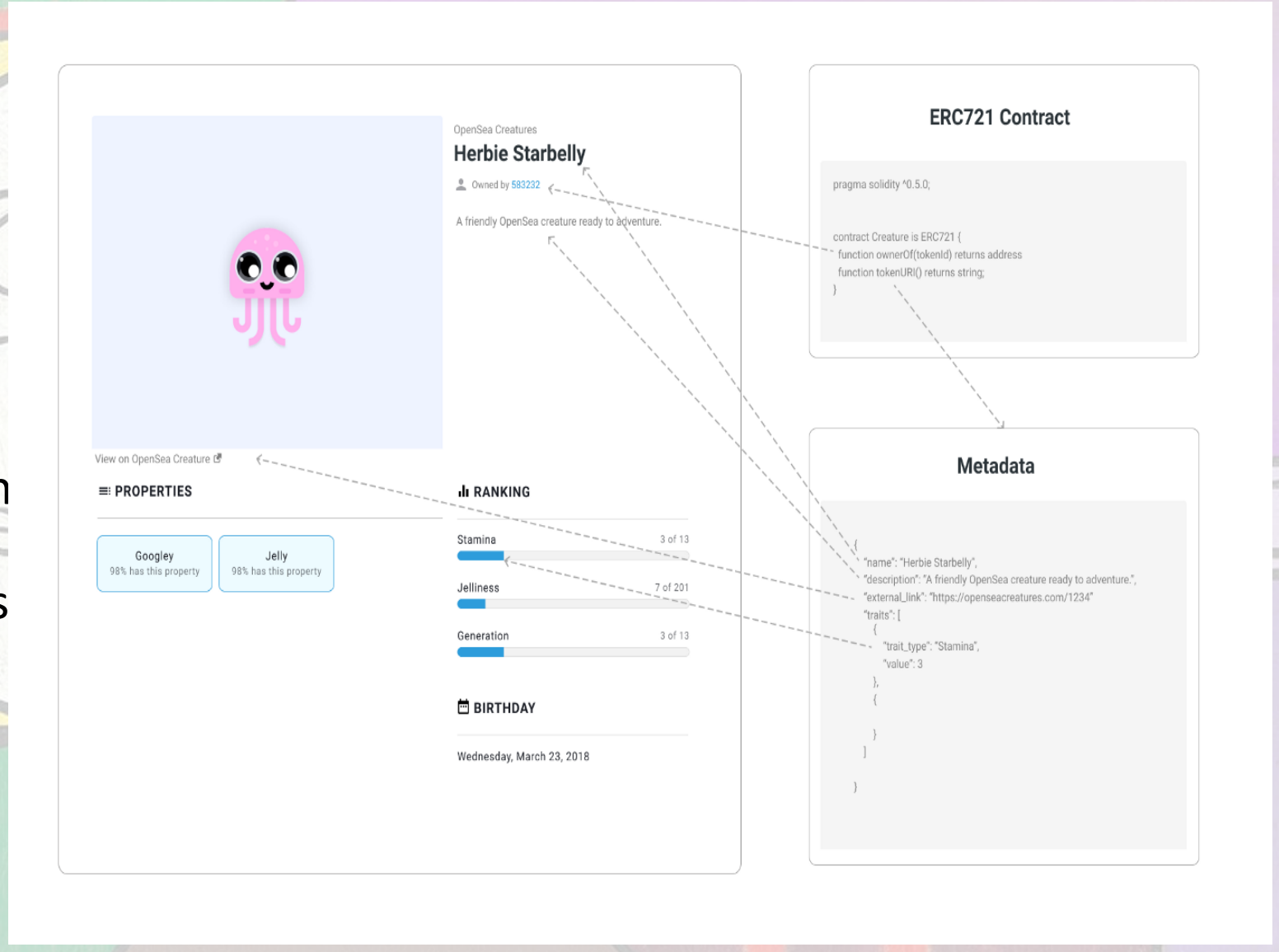
- Using the random package in python, we were able to generate images, in which stores the metadata of the generated image into a JSON file

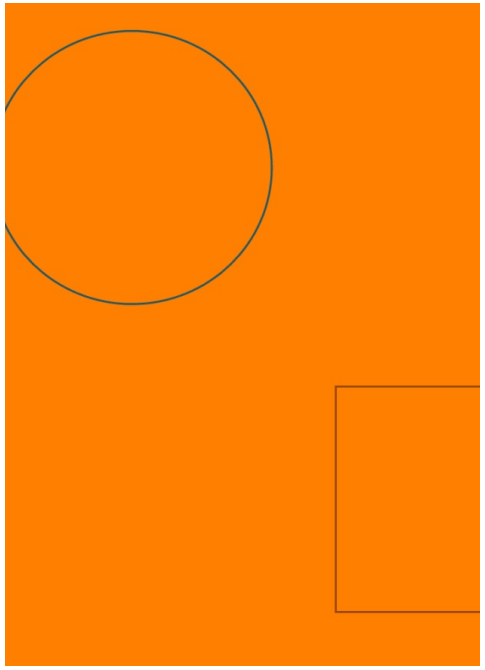
```
background = ["Blue", "Orange"]
background_weights = [30,15]

circle = ["Blue", "Orange"]
circle_weights = [30,15]

square = ["Blue", "Orange"]
square_weights = [30,15]

background_files = {
    "Blue" : "blue",
    "Orange" : "orange",
}

square_files = {
    "Blue" : "blue-square",
    "Orange" : "orange-square",
}

circle_files = {
    "Blue" : "blue-circle",
    "Orange" : "orange-circle",
}
```

```
TOTAL_IMAGES = 5

all_images = []

def create_new_image():
    new_image = {}

    #For each feature category, select a random feature according to the weight
    new_image ["Background"] = random.choices(background, background_weights)[0]
    new_image ["Circle"] = random.choices(circle, circle_weights)[0]
    new_image ["Square"] = random.choices(square, square_weights)[0]

    if new_image in all_images:
        return create_new_image()
    else:
        return new_image

for i in range(TOTAL_IMAGES):
    new_trait_image = create_new_image()
    all_images.append(new_trait_image)
```

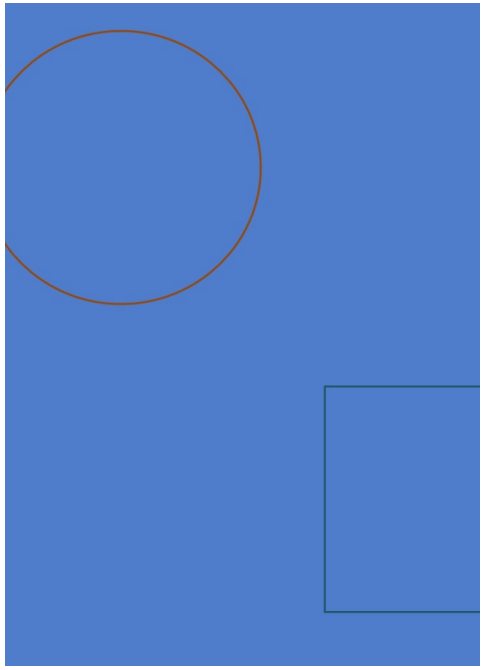
Metadata

- Metadata is the core of an NFT, it is a JSON file that often contains: NFT's name, Description, attributes, & link to the hosted image.
- Uploading data to a blockchain can be expensive, for this reason NFTs store their images somewhere else and a link to this hosted image is included in the Metadata.

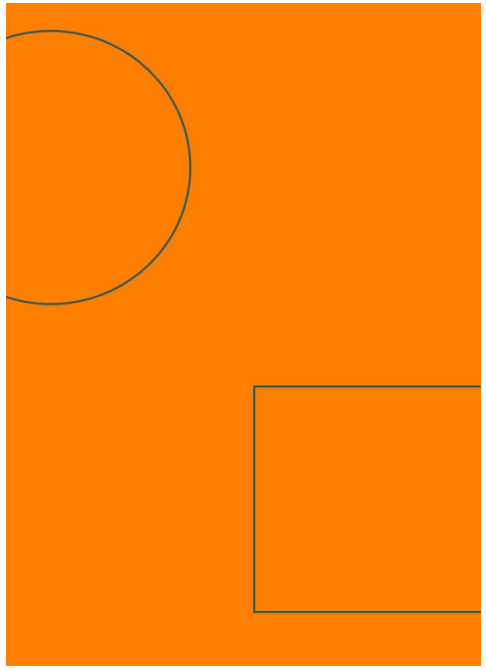




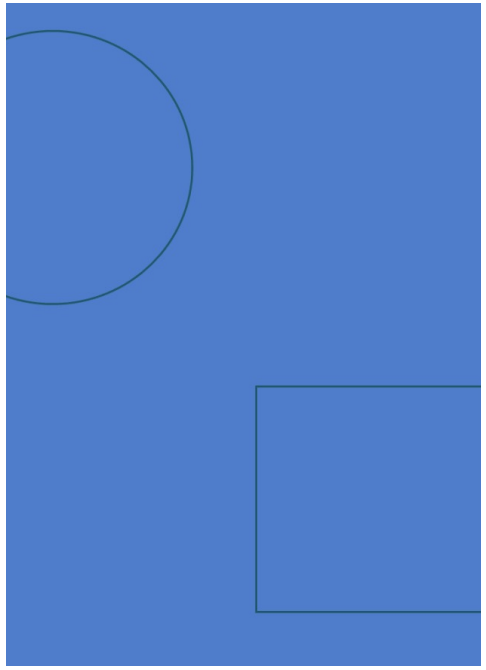
Picasso0



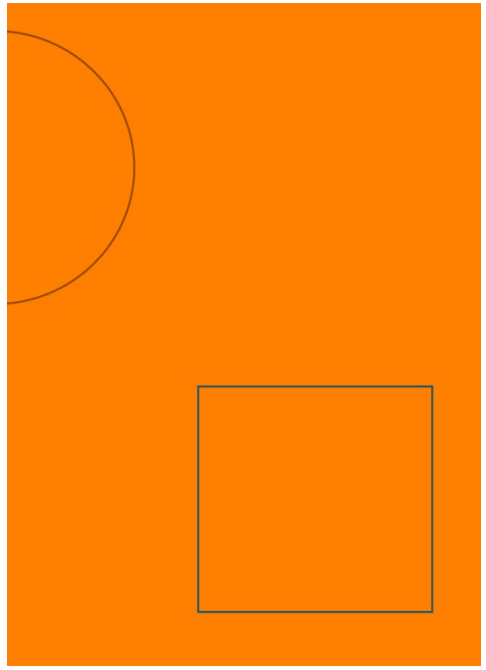
Picasso1



Picasso2

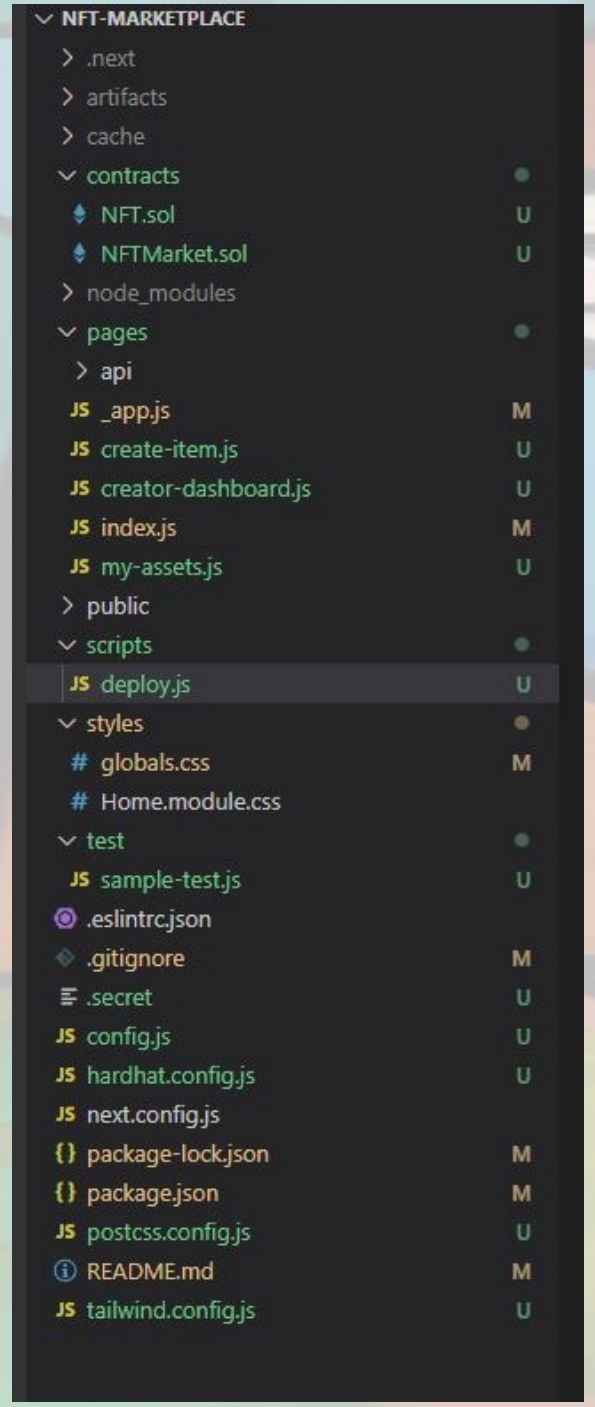


Picasso3



Picasso4


```
{
    "image": "https://gateway.pinata.cloud/ipfs/QmdNgj4jsQmbFSWCrkuXMNnyuiGNoDg9ra47uvvGpVaXib",
    "tokenId": 0,
    "name": "Picass0",
    "attributes": [
        {
            "trait_type": "Background",
            "value": "Orange"
        },
        {
            "trait_type": "Circle",
            "value": "Blue"
        },
        {
            "trait_type": "Square",
            "value": "Blue"
        }
    ]
}
```



NFT Marketplace - Code

```
export default function CreateItem() {
  const [fileUrl, setFileUrl] = useState(null)
  const [formInput, updateFormInput] = useState({ price: '', name: '', description: '' })
  const router = useRouter()

  async function onChange(e) {
    const file = e.target.files[0]
    try {
      const added = await client.add(
        file,
        {
          progress: (prog) => console.log(`received: ${prog}`)
        }
      )
      const url = `https://ipfs.infura.io/ipfs/${added.path}`
      setFileUrl(url)
    } catch (error) {
      console.log('Error uploading file: ', error)
    }
  }
  async function createMarket() {
    const { name, description, price } = formInput
    if (!name || !description || !price || !fileUrl) return
    /* first, upload to IPFS */
    const data = JSON.stringify({
      name, description, image: fileUrl
    })
    try {
      const added = await client.add(data)
      const url = `https://ipfs.infura.io/ipfs/${added.path}`
      /* after file is uploaded to IPFS, pass the URL to save it on Polygon */
      createSale(url)
    } catch (error) {
      console.log('Error uploading file: ', error)
    }
  }
```

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "hardhat/console.sol";

contract NFT is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    address contractAddress;

    constructor(address marketplaceAddress) ERC721("Metaverse", "METT") {
        contractAddress = marketplaceAddress;
    }

    function createToken(string memory tokenURI) public returns (uint) {
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();

        _mint(msg.sender, newItemId);
        _setTokenURI(newItemId, tokenURI);
        setApprovalForAll(contractAddress, true);
        return newItemId;
    }
}
```

```sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "hardhat/console.sol";

contract NFTMarket is ReentrancyGuard {
  using Counters for Counters.Counter;
  Counters.Counter private _itemIds;
  Counters.Counter private _itemsSold;

  address payable owner;
  uint256 listingPrice = 0.025 ether;

  constructor() {
    owner = payable(msg.sender);
  }

  struct MarketItem {
    uint itemId;
    address nftContract;
    uint256 tokenId;
    address payable seller;
    address payable owner;
    uint256 price;
    bool sold;
  }

  mapping(uint256 => MarketItem) private idToMarketItem;

  event MarketItemCreated (
    uint indexed itemId,
    address indexed nftContract,
    uint256 indexed tokenId,
    address seller,
    address owner,
    uint256 price,
    bool sold
  );
```

```
/* Returns the listing price of the contract */
function getListingPrice() public view returns (uint256) {
  return listingPrice;
}

/* Places an item for sale on the marketplace */
function createMarketItem(
  address nftContract,
  uint256 tokenId,
  uint256 price
) public payable nonReentrant {
  require(price > 0, "Price must be at least 1 wei");
  require(msg.value == listingPrice, "Price must be equal to listing price");

  _itemIds.increment();
  uint256 itemId = _itemIds.current();

  idToMarketItem[itemId] =  MarketItem(
    itemId,
    nftContract,
    tokenId,
    payable(msg.sender),
    payable(address(0)),
    price,
    false
  );

  IERC721(nftContract).transferFrom(msg.sender, address(this), tokenId);

  emit MarketItemCreated(
    itemId,
    nftContract,
    tokenId,
    msg.sender,
    address(0),
    price,
    false
  );
}
```

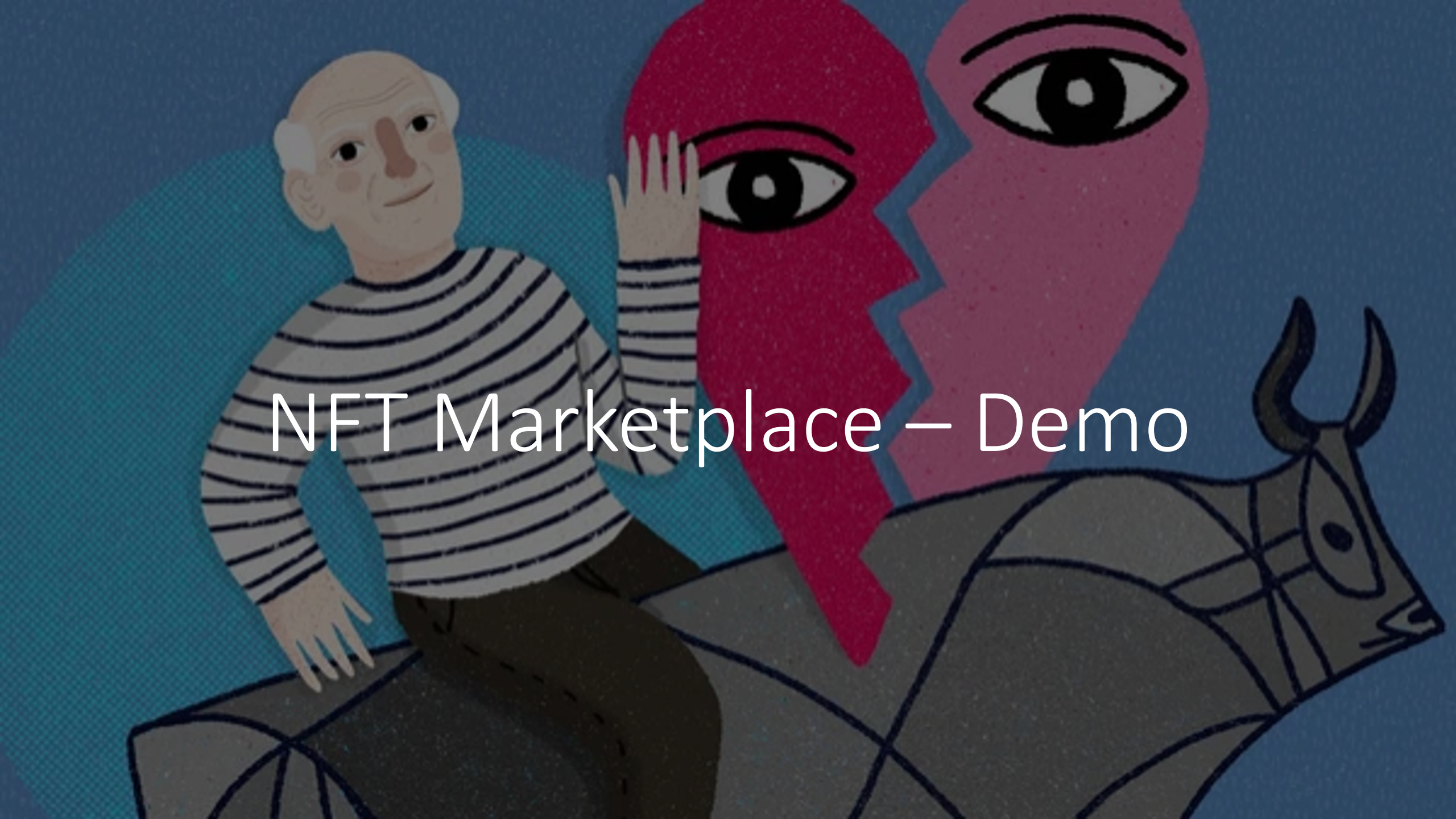
```
/* Creates the sale of a marketplace item */
/* Transfers ownership of the item, as well as funds between parties */
function createMarketSale(
  address nftContract,
  uint256 itemId
  ) public payable nonReentrant {
  uint price = idToMarketItem[itemId].price;
  uint tokenId = idToMarketItem[itemId].tokenId;
  require(msg.value == price, "Please submit the asking price in order to complete the purchase");

  idToMarketItem[itemId].seller.transfer(msg.value);
  IERC721(nftContract).transferFrom(address(this), msg.sender, tokenId);
  idToMarketItem[itemId].owner = payable(msg.sender);
  idToMarketItem[itemId].sold = true;
  _itemsSold.increment();
  payable(owner).transfer(listingPrice);
}

/* Returns all unsold market items */
function fetchMarketItems() public view returns (MarketItem[] memory) {
  uint itemCount = _itemIds.current();
  uint unsoldItemCount = _itemIds.current() - _itemsSold.current();
  uint currentIndex = 0;

  MarketItem[] memory items = new MarketItem[](unsoldItemCount);
  for (uint i = 0; i < itemCount; i++) {
    if (idToMarketItem[i + 1].owner == address(0)) {
      uint currentId = i + 1;
      MarketItem storage currentItem = idToMarketItem[currentId];
      items[currentIndex] = currentItem;
      currentIndex += 1;
    }
  }
  return items;
}
```

```
/* Returnsonlyl items that a user has purchased */
function fetchMyNFTs() public view returns (MarketItem[] memory) {
  uint totalItemCount = _itemIds.current();
  uint itemCount = 0;
  uint currentIndex = 0;

  for (uint i = 0; i < totalItemCount; i++) {
    if (idToMarketItem[i + 1].owner == msg.sender) {
      itemCount += 1;
    }
  }

  MarketItem[] memory items = new MarketItem[](itemCount);
  for (uint i = 0; i < totalItemCount; i++) {
    if (idToMarketItem[i + 1].owner == msg.sender) {
      uint currentId = i + 1;
      MarketItem storage currentItem = idToMarketItem[currentId];
      items[currentIndex] = currentItem;
      currentIndex += 1;
    }
  }
  return items;
}
```

```
/* Returns only items a user has created */
function fetchItemsCreated() public view returns (MarketItem[] memory) {
  uint totalItemCount = _itemIds.current();
  uint itemCount = 0;
  uint currentIndex = 0;

  for (uint i = 0; i < totalItemCount; i++) {
    if (idToMarketItem[i + 1].seller == msg.sender) {
      itemCount += 1;
    }
  }

  MarketItem[] memory items = new MarketItem[](itemCount);
  for (uint i = 0; i < totalItemCount; i++) {
    if (idToMarketItem[i + 1].seller == msg.sender) {
      uint currentId = i + 1;
      MarketItem storage currentItem = idToMarketItem[currentId];
      items[currentIndex] = currentItem;
      currentIndex += 1;
    }
  }
  return items;
}
}
```

```
export default function CreateItem() {
  const [fileUrl, setFileUrl] = useState(null)
  const [formInput, updateFormInput] = useState({ price: '', name: '', description: '' })
  const router = useRouter()

  async function onChange(e) {
    const file = e.target.files[0]
    try {
      const added = await client.add(
        file,
        {
          progress: (prog) => console.log(`received: ${prog}`)
        }
      )
      const url = `https://ipfs.infura.io/ipfs/${added.path}`
      setFileUrl(url)
    } catch (error) {
      console.log('Error uploading file: ', error)
    }
  }
  async function createMarket() {
    const { name, description, price } = formInput
    if (!name || !description || !price || !fileUrl) return
    /* first, upload to IPFS */
    const data = JSON.stringify({
      name, description, image: fileUrl
    })
    try {
      const added = await client.add(data)
      const url = `https://ipfs.infura.io/ipfs/${added.path}`
      /* after file is uploaded to IPFS, pass the URL to save it on Polygon */
      createSale(url)
    } catch (error) {
      console.log('Error uploading file: ', error)
    }
  }
```

```
async function createSale(url) {
  const web3Modal = new Web3Modal()
  const connection = await web3Modal.connect()
  const provider = new ethers.providers.Web3Provider(connection)
  const signer = provider.getSigner()

  /* next, create the item */
  let contract = new ethers.Contract(nftaddress, NFT.abi, signer)
  let transaction = await contract.createToken(url)
  let tx = await transaction.wait()
  let event = tx.events[0]
  let value = event.args[2]
  let tokenId = value.toNumber()

  const price = ethers.utils.parseUnits(formInput.price, 'ether')

  /* then list the item for sale on the marketplace */
  contract = new ethers.Contract(nftmarketaddress, Market.abi, signer)
  let listingPrice = await contract.getListingPrice()
  listingPrice = listingPrice.toString()

  transaction = await contract.createMarketItem(nftaddress, tokenId, price, { value: listingPrice })
  await transaction.wait()
  router.push('/')
}
```



NFT Marketplace – Demo

Future Plans

Links

- Github Link: https://github.com/jcarballo321/Project_3