

The background of the slide features a vibrant, abstract painting. It consists of bold, angular brushstrokes in shades of teal, dark brown, and grey. The composition is dynamic, with thick, layered strokes creating a sense of depth and movement. In the upper right corner, there are some faint, illegible markings that appear to be Arabic script.

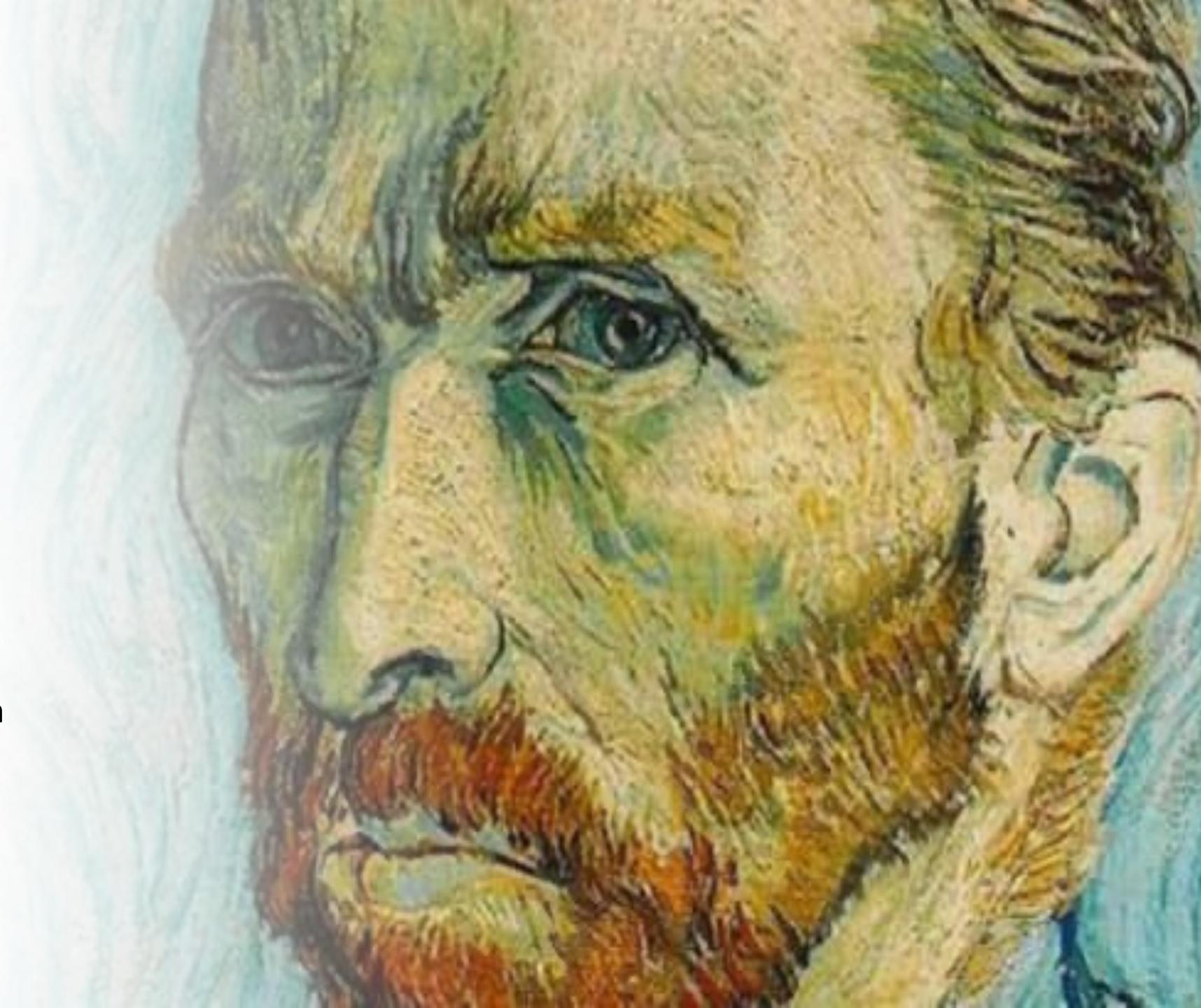
# NFT Marketplace & Generator The 21<sup>st</sup> century Picasso

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
- Tokenomics
- Hard hat



# Challenges

---



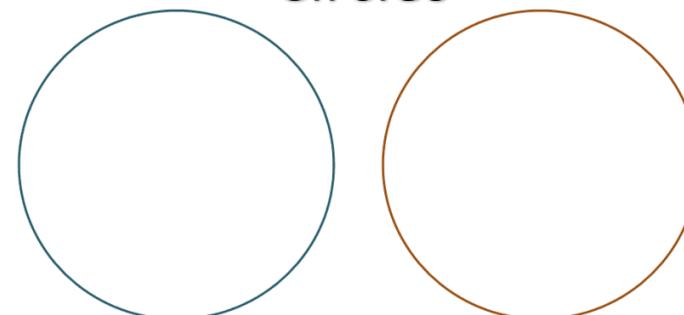
# Summary

- Marketplace: Allows a user to upload a picture to the blockchain, buy an NFT from the market and store them in their profile.
- 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.

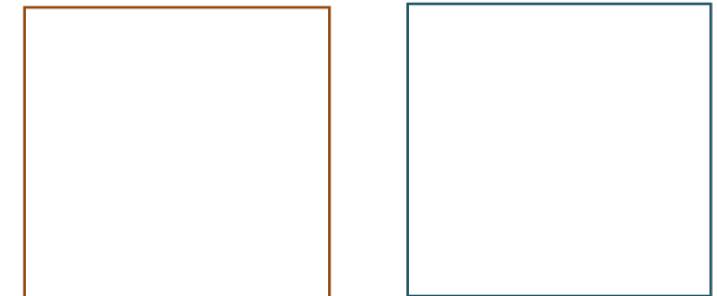
Backgrounds



Circles



Squares



# 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 = {}

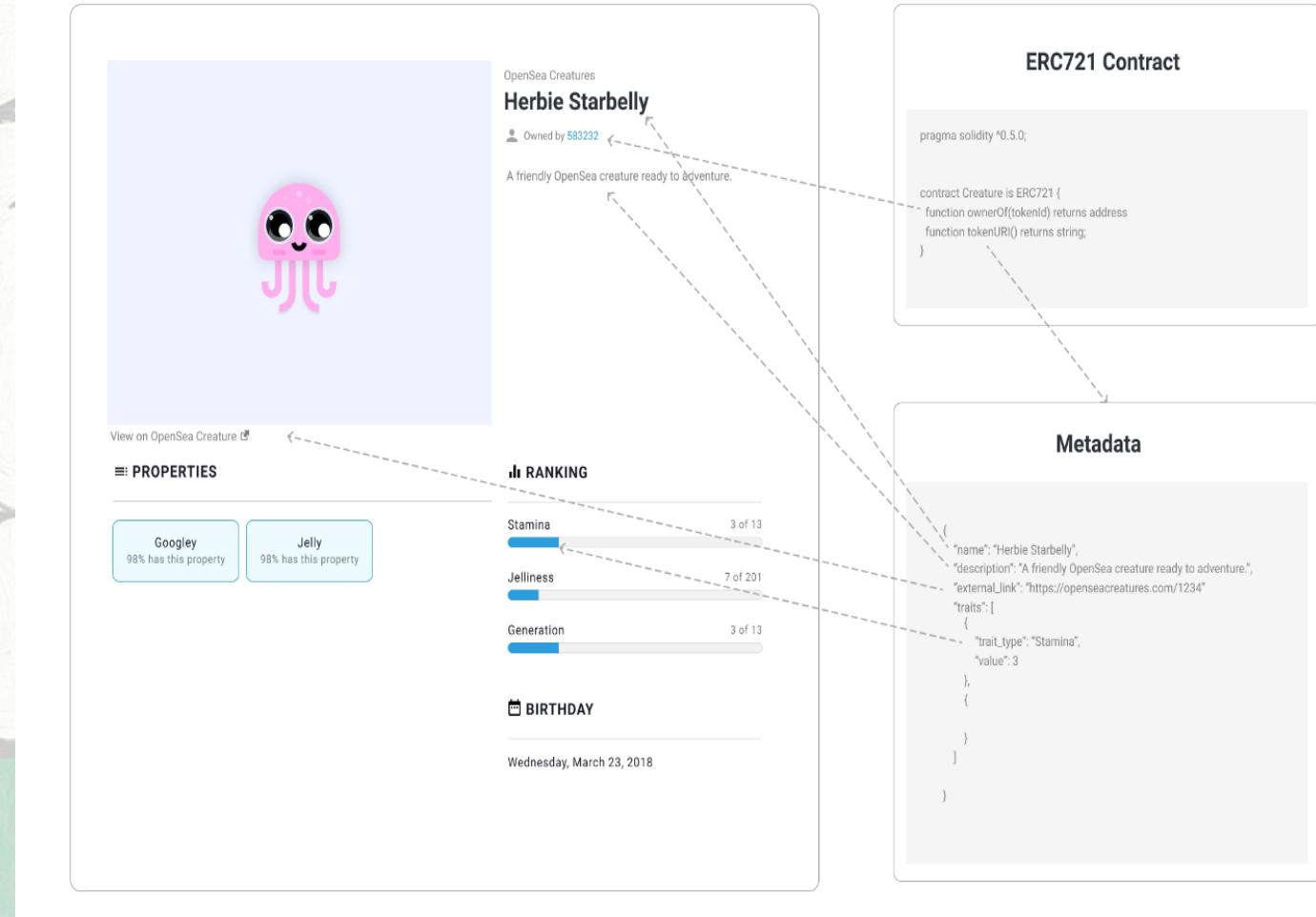
    #For each feature category, select a random feature according to the weight
    new_image ["Background"] = random.choices(background, background_weights)[0]
    new_image ["Circle"] = random.choices(circle, circle_weights)[0]
    new_image ["Square"] = random.choices(square, square_weights)[0]

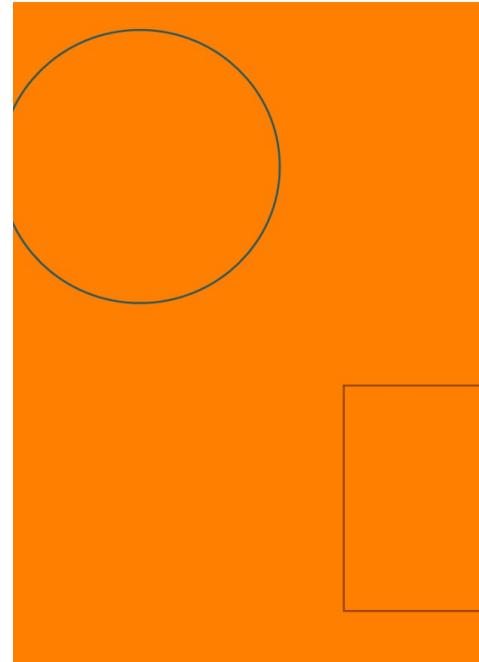
    if new_image in all_images:
        return create_new_image()
    else:
        return new_image

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

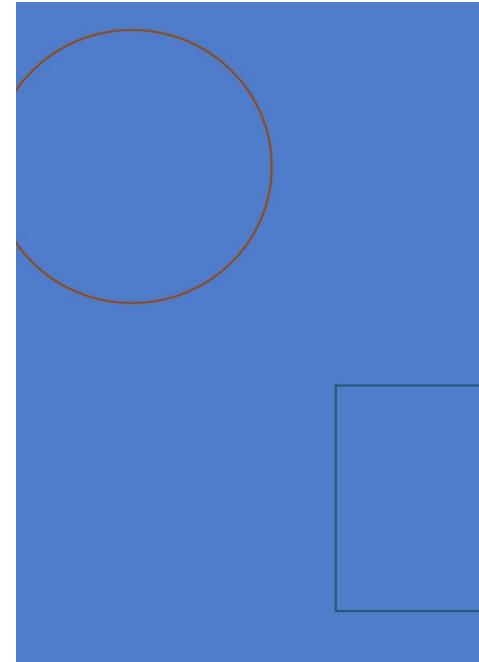
# Metadata

- Metadata is the core of an NFT, it is a JSON file that often contains: NFT's name, description, attributes, & a 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 the hosted image is included in the Metadata.

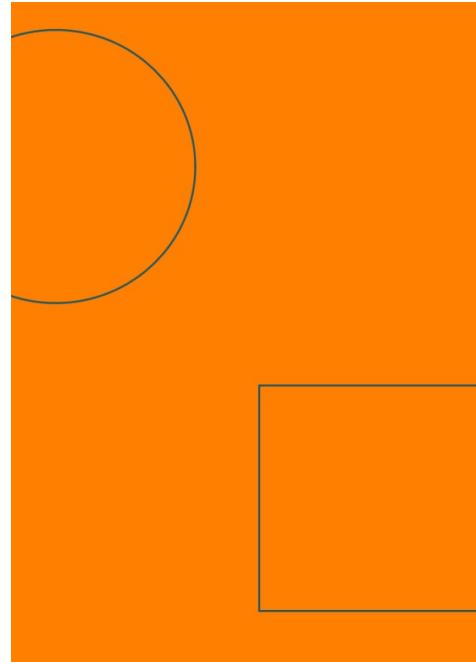




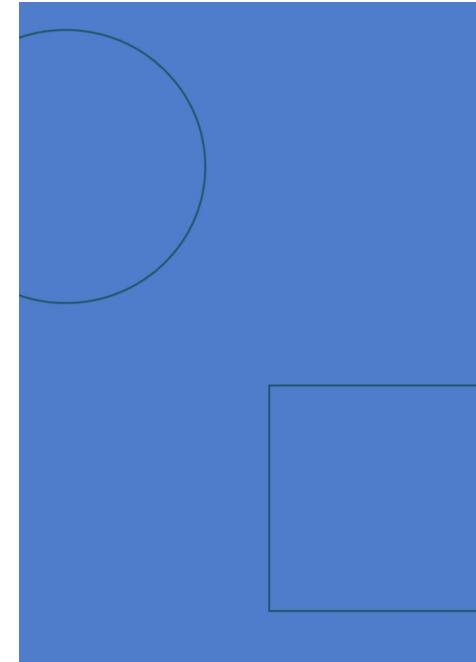
Picasso0



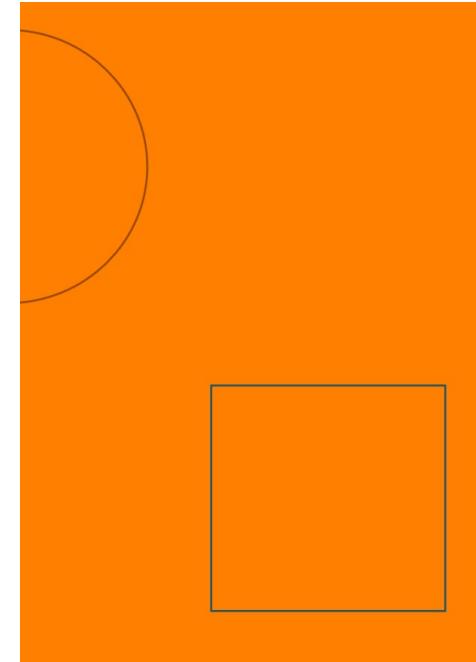
Picasso1



Picasso2



Picasso3



Picasso4

```
{  
  "image": "https://gateway.pinata.cloud/ipfs/QmdNgj4jsQmbFSWCrukXMNnyuiGNodg9ra47uvvGpVaXib",  
  "tokenId": 0,  
  "name": "Picass0",  
  "attributes": [  
    {  
      "trait_type": "Background",  
      "value": "Orange"  
    },  
    {  
      "trait_type": "Circle",  
      "value": "Blue"  
    },  
    {  
      "trait_type": "Square",  
      "value": "Blue"  
    }  
  ]  
}
```

NFT-MARKETPLACE	
> .next	
> artifacts	
> cache	
✓ contracts	
◆ NFT.sol	U
◆ NFTMarket.sol	U
> node_modules	
✓ pages	
> api	
JS _app.js	M
JS create-item.js	U
JS creator-dashboard.js	U
JS index.js	M
JS my-assets.js	U
> public	
✓ scripts	
JS deploy.js	U
✓ styles	
# globals.css	M
# Home.module.css	
✓ test	
JS sample-test.js	U
① .eslintrc.json	
◆ .gitignore	M
Ξ .secret	U
JS config.js	U
JS hardhat.config.js	U
JS next.config.js	
{} package-lock.json	M
{} package.json	M
JS postcss.config.js	U
① README.md	M
JS tailwind.config.js	U

# 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)
    }
  }
}
```

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.4;
3
4 import "@openzeppelin/contracts/utils/Counters.sol";
5 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
6 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
7
8 import "hardhat/console.sol";
9
10 contract NFT is ERC721URIStorage {
11     using Counters for Counters.Counter;
12     Counters.Counter private _tokenIds;
13     address contractAddress;
14
15     constructor(address marketplaceAddress) ERC721("Metaverse", "METT") {
16         contractAddress = marketplaceAddress;
17     }
18
19     function createToken(string memory tokenURI) public returns (uint) {
20         _tokenIds.increment();
21         uint256 newItemId = _tokenIds.current();
22
23         mint(msg.sender, newItemId);
24         setTokenURI(newItemId, tokenURI);
25         setApprovalForAll(contractAddress, true);
26         return newItemId;
27     }
28 }
```

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.4;
3
4 import "@openzeppelin/contracts/utils/Counters.sol";
5 import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
6 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
7
8 import "hardhat/console.sol";
9
10 contract NFTMarket is ReentrancyGuard {
11     using Counters for Counters.Counter;
12     Counters.Counter private _itemIds;
13     Counters.Counter private _itemsSold;
14
15     address payable owner;
16     uint256 listingPrice = 0.025 ether;
17
18     constructor() {
19         owner = payable(msg.sender);
20     }
21
22     struct MarketItem {
23         uint itemId;
24         address nftContract;
25         uint256 tokenId;
26         address payable seller;
27         address payable owner;
28         uint256 price;
29         bool sold;
30     }
31
32     mapping(uint256 => MarketItem) private idToMarketItem;
33
34     event MarketItemCreated (
35         uint indexed itemId,
36         address indexed nftContract,
37         uint256 indexed tokenId,
38         address seller,
39         address owner,
40         uint256 price,
41         bool sold
42     );
43 }
```

```
/* 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;
}
```

```
/* Returns only 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('/')
}
```

A stylized illustration featuring a man with a bald head and a striped shirt, looking towards the right. To his right is a large, red, angular shape containing two large, black-outlined eyes. Below the man is a dark, textured shape resembling a horse's head. The background is a dark blue.

# NFT Marketplace – Demo

# Future Plans

---



# Links

---

- Github Link:  
[https://github.com/jcarballo321/  
Project\\_3](https://github.com/jcarballo321/Project_3)

