

KELVIN KISSI

# NFT DAPP - PROFIT PAYMENT SPLITTER



---

# EXECUTIVE SUMMARY

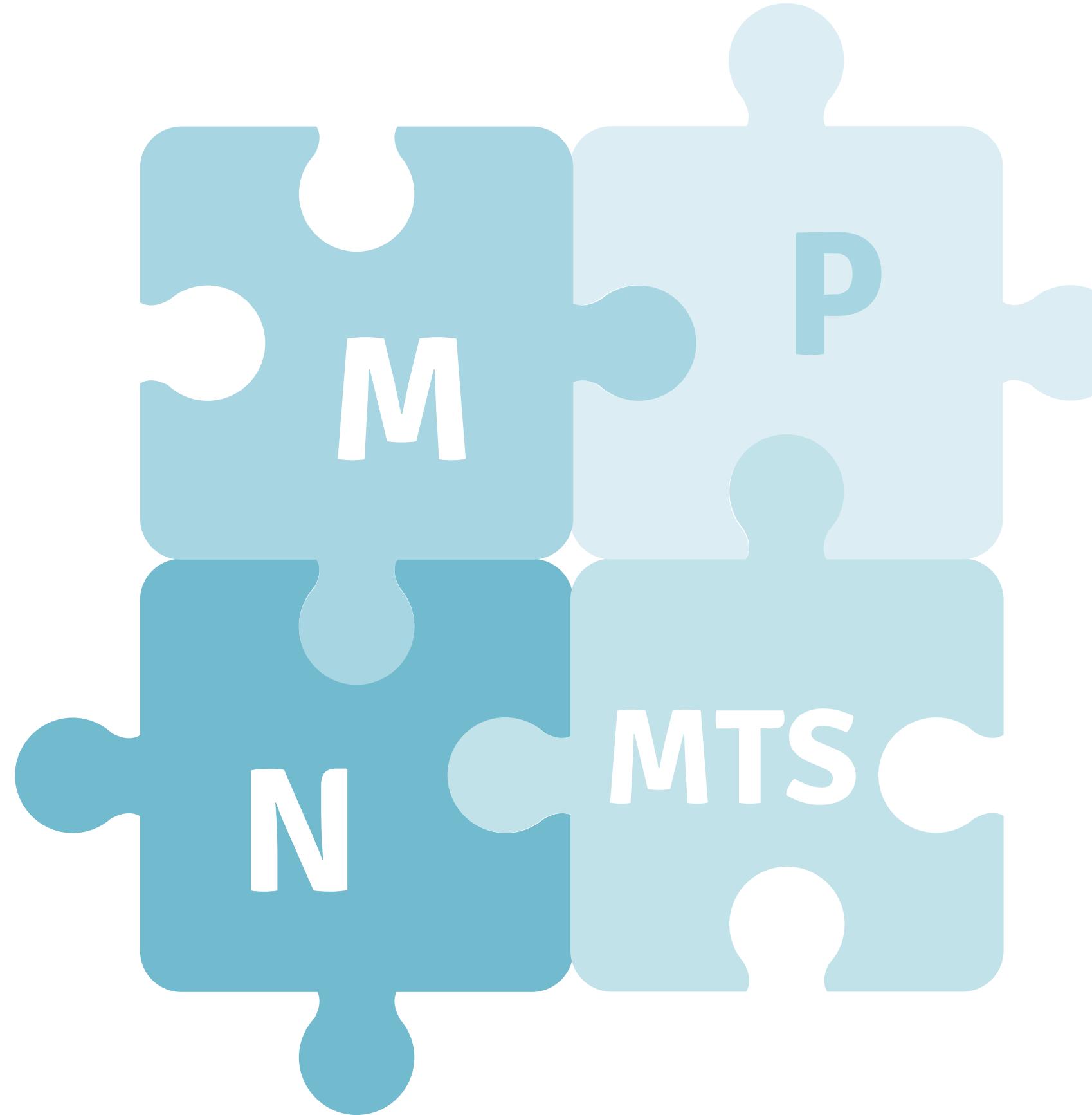
- In this project I used the programming language Solidity to showcase the launch of an actual NFT collection on OpenSea. The ERC-721 tokens from the NFT collection on OpenSea will be used as proof of ticket ownership of an exclusive virtual event. The proceed of those tickets sale will be divided into three profit shares (70,20,10) percent based on their input on the project. I used the Rinkeby test network to deploy my smart contracts and to create a ticket payment model. This model help us identify one of many potential use cases of the blockchain technology.
- I used the Openzeppelin libraries to create the different smart contracts (NFT & Payment Splitter) and Node.js for the minting Dapp. I also decided to add another smart contract feature "MultiSig wallet" for extra security for the largest share holder.
- I picked the Rinkeby network to test my source codes, but the model (smart contract)could be used on the Polygon network as well as ETH mainnet.

- As a front end specialist we understand how crucial it is to have a custom UI for people that are not very technical and have very little knowledge about what is taking place when interacting with the blockchain. This is why I decided to not only integrate a minting Dapp in my project but allow everyone to understand what is really taking place.
- It is important to note that the end user will not interact with the smart contract directly, they use the front end of the app to interact with the blockchain.

---

# SMART CONTRACTS

- Smart contracts are computer programs that can run on a blockchain. This means that people can use them to build decentralized applications (dApps) that can run code in a trustworthy way.
- Because smart contracts exist in a blockchain, they inherit two valuable properties. The first is that smart contracts are immutable, meaning that once a smart contract is created and validated, it can never be changed.
- The second property is that smart contracts are distributed. This means that everyone in the blockchain network validates the terms of the contract. If one party to the contract or member of the blockchain network tries to override the terms of the contract—for example, by trying to trigger an early release of funds—the other network members will recognize the change to the contract as invalid and prevent the action.



---

## PLAN ANALYTICS

### MINTING DAPP

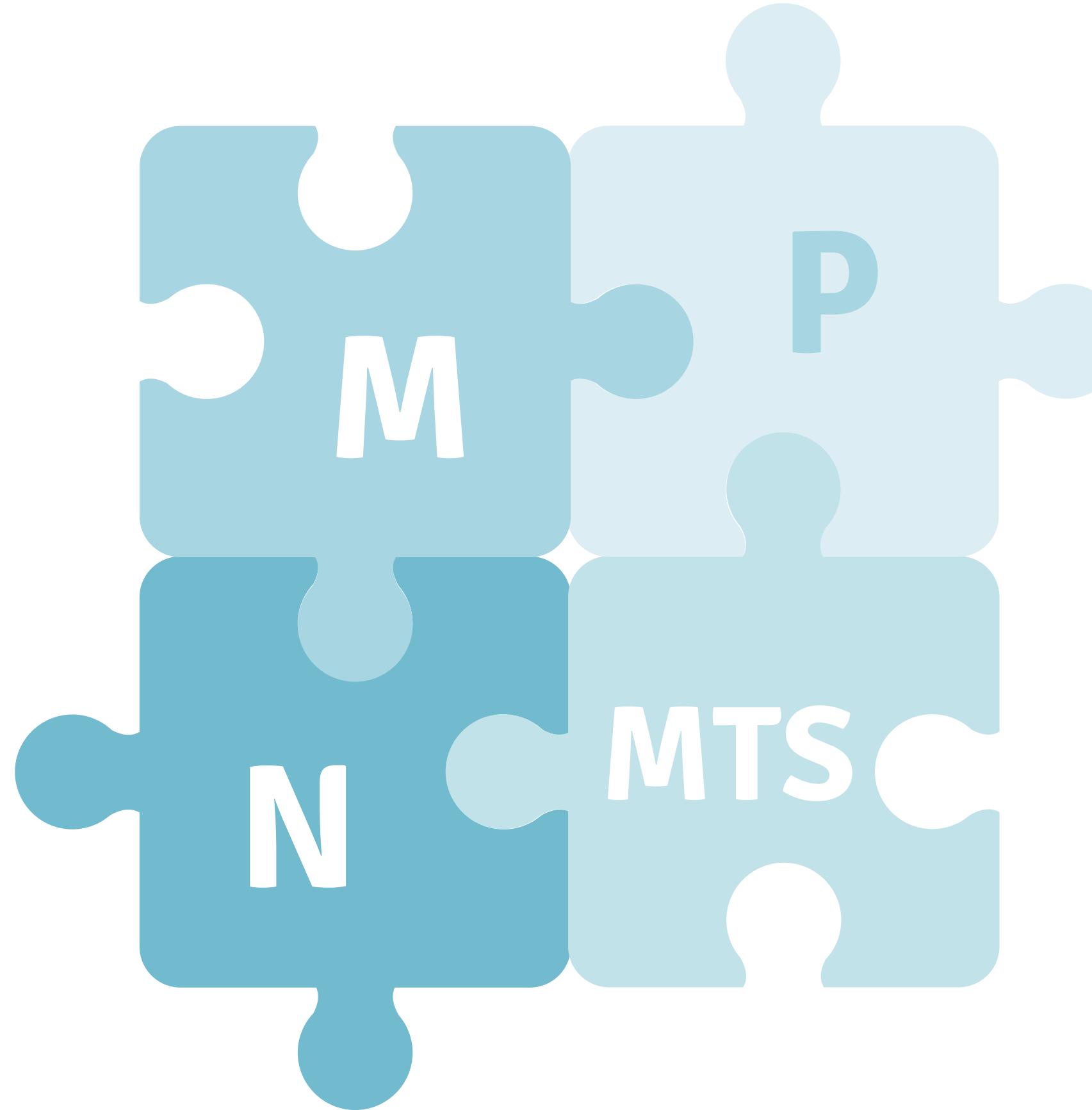
NFT minter DApp is a simple UI (user interface) where you can input information about your NFT or digital creation. Ability to mint NFT from personalized website.

### PAYMENT SPLITTER CONTRACT

This contract allows to split Ether payments among a group of accounts. The sender does not need to be aware that the Ether will be split in this way, since it is handled transparently by the contract.

### NFT SMART CONTRAT

An NFT smart contract is a mechanism for implementing a sale agreement between the NFT owner and the buyer.

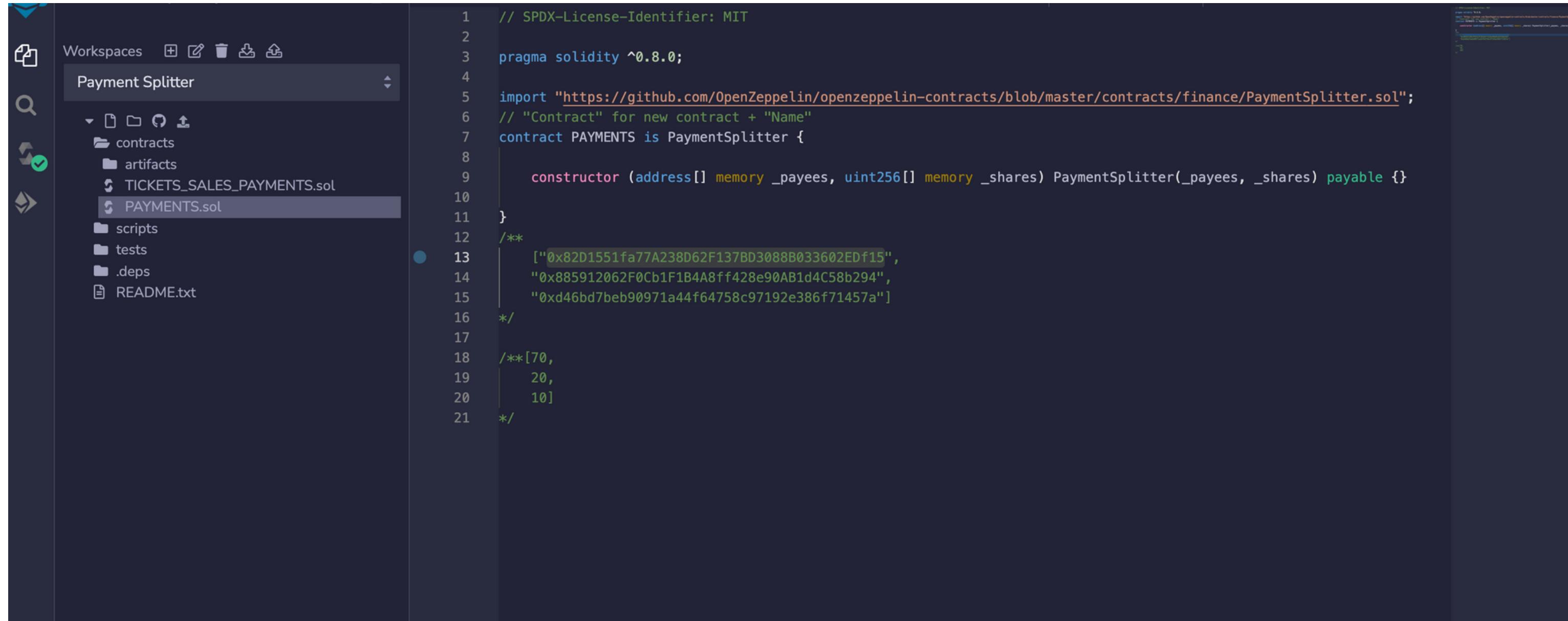


---

## PLAN ANALYTICS

### **MULTISIG WALLET - SMART CONTRACT**

A MultiSig wallet is a digital wallet that operates with multisignature addresses. This means that it requires more than one private key to sign and authorize a crypto transaction or, in some cases, that several different keys can be used to generate a signature.



The screenshot shows a Solidity smart contract named "PAYMENTS.sol" in a workspace titled "Payment Splitter". The code defines a contract "PAYMENTS" that inherits from "PaymentSplitter". It imports the "PaymentSplitter" contract from OpenZeppelin. The constructor takes two arrays of addresses: "\_payees" and "\_shares", and initializes the contract with these values. The code also includes a comment block with three wallet addresses and their corresponding shares.

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
4
5 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/finance/PaymentSplitter.sol";
6 // "Contract" for new contract + "Name"
7 contract PAYMENTS is PaymentSplitter {
8
9     constructor (address[] memory _payees, uint256[] memory _shares) PaymentSplitter(_payees, _shares) payable {}
10
11 }
12 /**
13  * ["0x82D1551fa77A238D62F137BD3088B033602EDf15",
14  * "0x885912062F0Cb1F1B4A8ff428e90AB1d4C58b294",
15  * "0xd46bd7beb90971a44f64758c97192e386f71457a"]
16 */
17
18 /**
19  * [70,
20  *   20,
21  *   10]
22 */
```

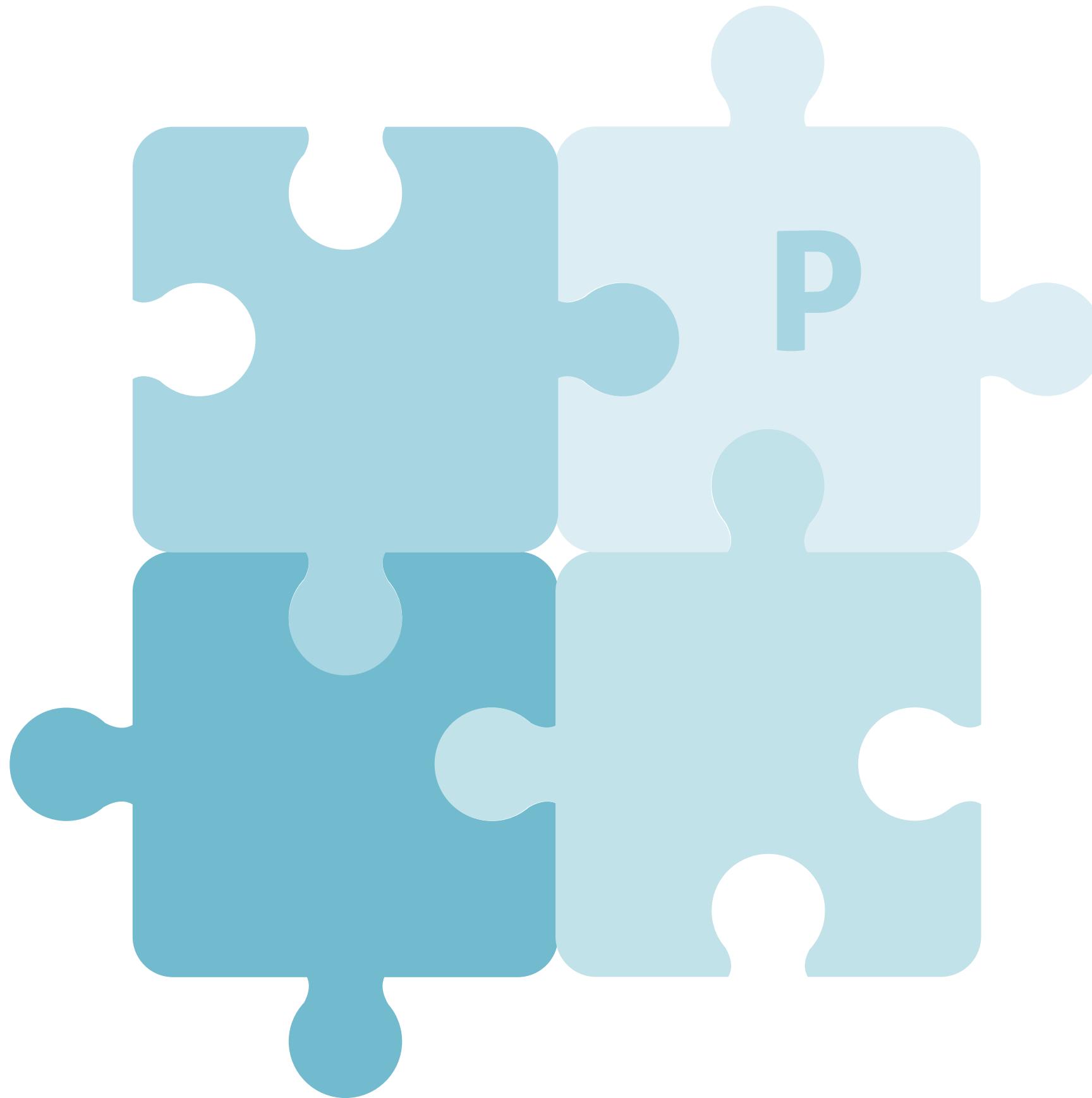
# **PAYMENT SPLITTER SMART CONTRACT**

**WALLET ADDRESSES**

OWNERS OF THE PROJECT - INVESTORS

**SHARES**

OWNERS SHARES BASED ON INPUT



---

## PULL METHOD

Pull Method: Require the User to pull the amount of share (ETH) that is allowed to them.

The screenshot shows a blockchain development interface. On the left is the 'FILE EXPLORERS' panel, which lists workspaces and files. The 'Payment Splitter' workspace is selected, showing a folder structure with 'contracts', 'artifacts', 'scripts', 'tests', '.deps', and 'README.txt'. Inside 'contracts', 'TICKETS\_SALES\_PAYMENTS.sol' and 'PAYMENTS.sol' are listed. The 'artifacts' folder contains a file named 'TICKETS\_SALES\_PAYMENTS.sol'. On the right is a code editor displaying a Solidity smart contract named 'TICKETS\_SALES\_PAYMENTS'. The code defines a contract that inherits from 'ERC721Enumerable' and 'Ownable'. It includes variables for baseURI, baseExtension, cost (0.02 ether), maxSupply (10), maxMintAmount (5), paused status, whitelisted addresses, and a payable payments address. The constructor initializes the name, symbol, and baseURI, sets the payments address, and mints two NFTs to the wallet owner.

```
3
4 pragma solidity >=0.7.0 <0.9.0;
5
6 import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
7 import "@openzeppelin/contracts/access/Ownable.sol";
8
9 contract TICKETS_SALES_PAYMENTS is ERC721Enumerable, Ownable {
10     using Strings for uint256;
11
12     string public baseURI;
13     string public baseExtension = ".json";
14     uint256 public cost = 0.02 ether;
15     uint256 public maxSupply = 10;
16     uint256 public maxMintAmount = 5;
17     bool public paused = false;
18     mapping(address => bool) public whitelisted;
19     address payable public payments;
20
21     constructor(
22         string memory _name,
23         string memory _symbol,
24         string memory _initBaseURI,
25         address _payments
26     ) ERC721(_name, _symbol) {
27         setBaseURI(_initBaseURI);
28         payments = payable(_payments);
29         mint(msg.sender, 2);
30     }
31 }
```

# NFT SMART CONTRACT

## CONTRACT DETAILS

- 0.02 ETH - \$22.19 USD
- TOTAL SUPPLY = 10 NFTS
- MAX MINT PER WALLET = 5 NFTS

## TICKETS

2 TICKETS MINTED FOR THE WALLET OWNER

# TOKENSALES CONTRACT - ETHERSCAN

Etherscan

Rinkeby Testnet Network

All Filters Search by Address / Txn Hash / Block / Token / Ens

Home Blockchain Tokens Misc Rinkeby

Token Ticketsales ⓘ

Overview [ERC-721]

Max Total Supply: 3 ts ⓘ

Holders: 1

Transfers: 3

Profile Summary

Contract: 0x346622443110a9386c18ca6a6f2eb396de7ff49a

Transfers Holders Contract

A total of 3 transactions found

First < Page 1 of 1 > Last

Txn Hash	Method ⓘ	Age	From	To	TokenID
0xf09a22af2103087e6fc...	Mint	37 secs ago	0x000000000000000000000000...	0x885912062f0cb1f1b4a...	3
0x086d41e451aba54998...	0x60806040	9 mins ago	0x000000000000000000000000...	0x885912062f0cb1f1b4a...	2
0x086d41e451aba54998...	0x60806040	9 mins ago	0x000000000000000000000000...	0x885912062f0cb1f1b4a...	1

This website uses cookies to improve your experience and has an updated Privacy Policy. Got It

[ Download CSV Export ]

# OPENSEA NFT DISPLAY

The screenshot shows the OpenSea Testnets interface for the 'Ticketsales' collection. At the top, there's a navigation bar with back, forward, and search icons, followed by the URL 'testnets.opensea.io/collection/ticketsales'. Below the URL is the OpenSea logo with 'Testnets' underneath. A search bar contains the placeholder 'Search items, collections, and accounts'. To the right of the search bar are links for 'Explore', 'Stats', 'Resources', and 'Create', along with a profile icon and a copy icon.

The main content area features a large, empty gray placeholder box where NFT items would normally be displayed. Below this, the title 'Ticketsales' is prominently shown in bold black text. To the right of the title are icons for a chart, a star, a left arrow, and three dots. A welcome message reads: 'Welcome to the home of Ticketsales on OpenSea. Discover the best items in this collection. ...' with a 'See more ▾' link.

Below the welcome message, key statistics are displayed: '3 items', '1 owners', 'floor price ⚡ ---', and 'total volume ⚡ 0.00'. There are also tabs for 'Items' (which is selected) and 'Activity'.



**0 / 10**

0x346622443110a...

1 TS costs 0.02 Ether.

Excluding gas fees.

Connect to the Ethereum network

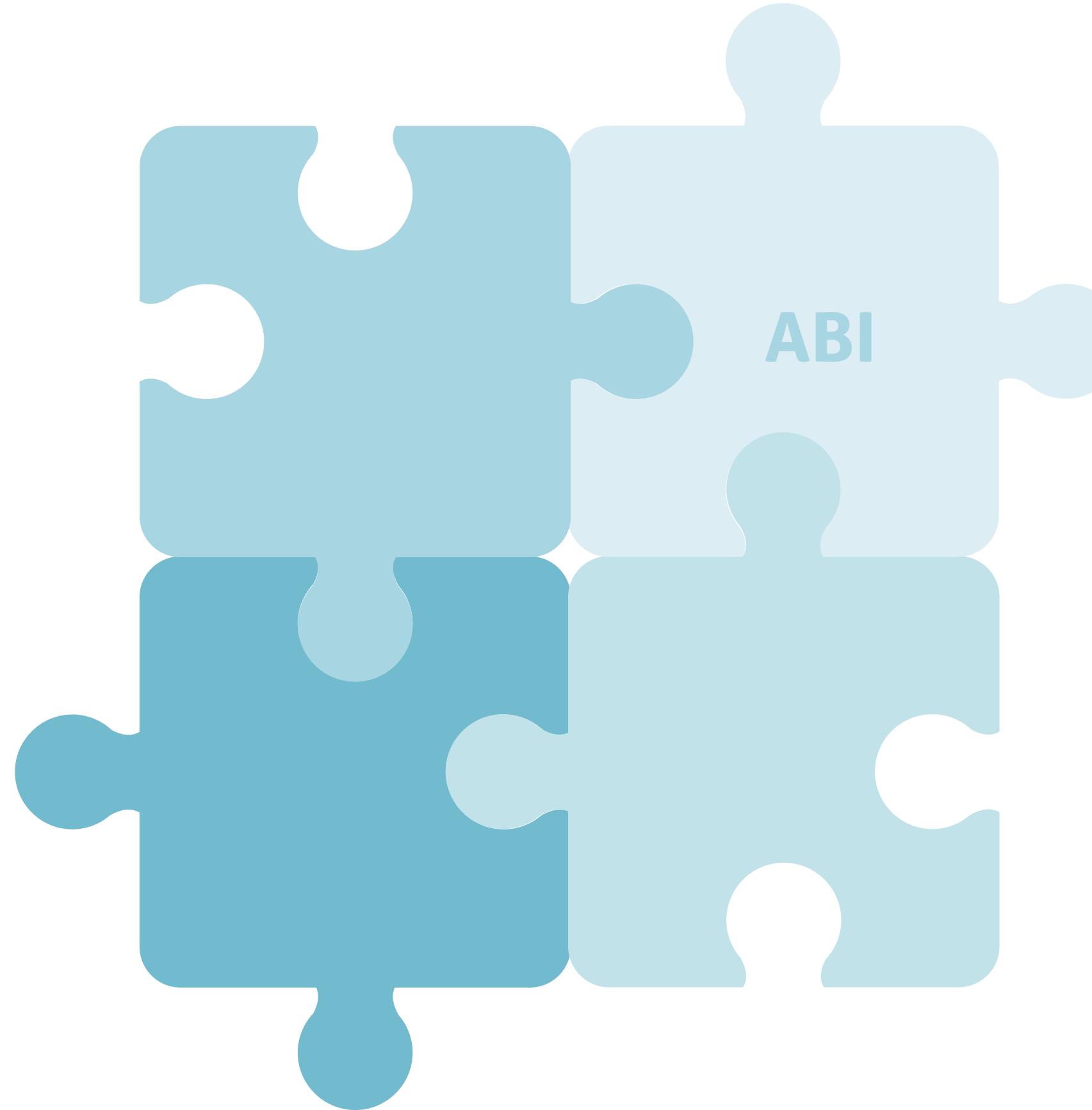
CONNECT



# MINTING DAPP

## DAPP DETAILS

- FRIENDLY META USER INTERFACE
- TOTAL SUPPLY = 10 NFTS
- CONTRACT ADDRESS
- MINTING CAPABILITY



---

## ABI

The Contract Application Binary Interface (ABI) is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction.

---

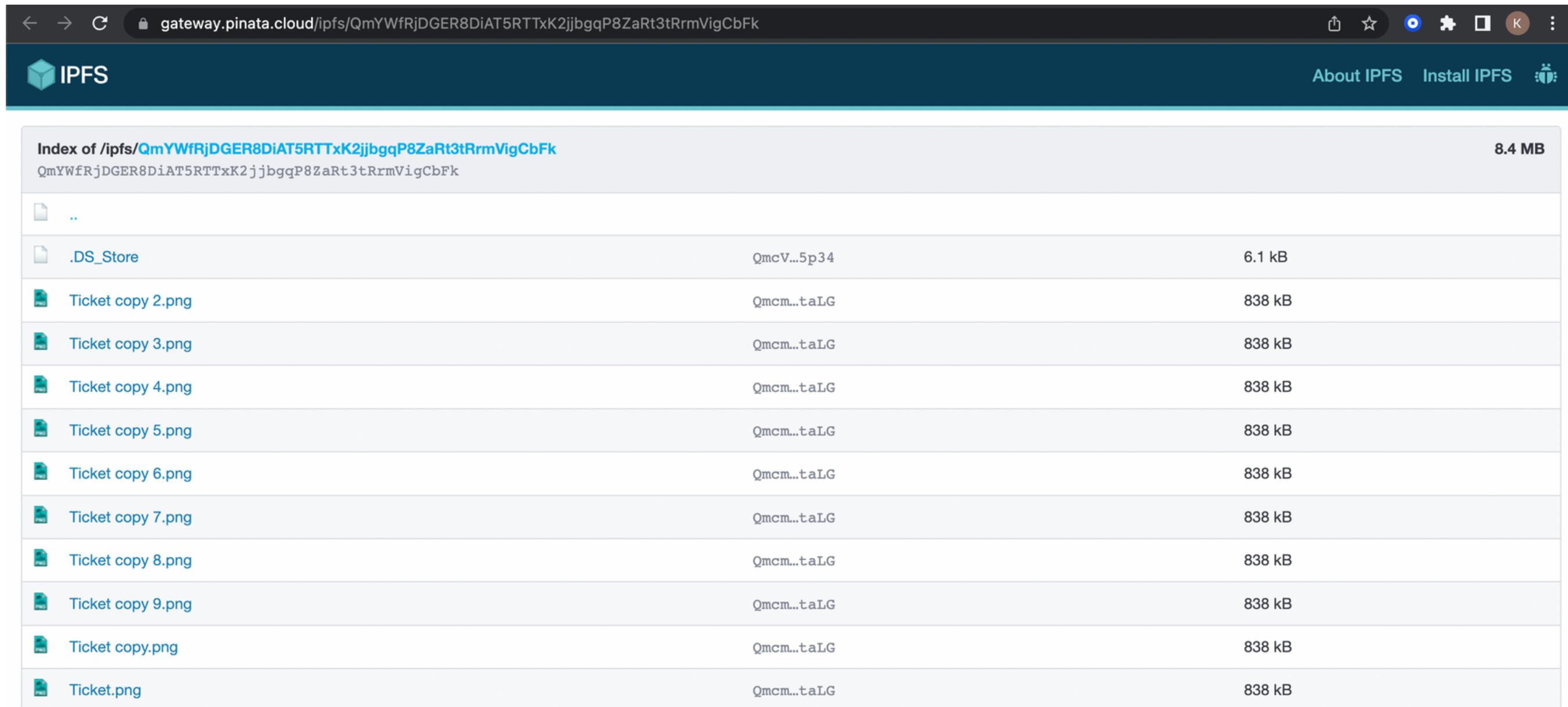
# DATA COLLECTION & EXPLORATION

The data used in my model is being pulled from IPFS and uploaded via Pinata.



The InterPlanetary File System is a protocol and peer-to-peer network for storing and sharing data in a distributed file system. IPFS uses content-addressing to uniquely identify each file in a global namespace connecting all computing devices.

# DATA COLLECTION & EXPLORATION



The screenshot shows a web browser window displaying an IPFS file listing. The URL in the address bar is `gateway.pinata.cloud/ipfs/QmYWfRjDGER8DiAT5RTTxK2jjbgqP8ZaRt3tRrmVigCbFk`. The page has a dark blue header with the IPFS logo and navigation icons. Below the header is a table listing files from the specified IPFS path.

Index of /ipfs/QmYWfRjDGER8DiAT5RTTxK2jjbgqP8ZaRt3tRrmVigCbFk		
	QmYWfRjDGER8DiAT5RTTxK2jjbgqP8ZaRt3tRrmVigCbFk	8.4 MB
..		
..		
.DS_Store	QmcV...5p34	6.1 kB
Ticket copy 2.png	Qmc...taLG	838 kB
Ticket copy 3.png	Qmc...taLG	838 kB
Ticket copy 4.png	Qmc...taLG	838 kB
Ticket copy 5.png	Qmc...taLG	838 kB
Ticket copy 6.png	Qmc...taLG	838 kB
Ticket copy 7.png	Qmc...taLG	838 kB
Ticket copy 8.png	Qmc...taLG	838 kB
Ticket copy 9.png	Qmc...taLG	838 kB
Ticket copy.png	Qmc...taLG	838 kB
Ticket.png	Qmc...taLG	838 kB

---

# APPROACH TO ACHIEVING GOALS

This project was initially inspired by my desire to understand how launch an NFT collection. I combined my front end web knowledge with the solidity backend to create a minting Dapp for NFTs that could be used as ticket to an exclusive virtual event and a payment splitter system for the different wallet addresses involved.

In the future I will most likely create and implement different features into the smart contract to have more of a DAO system model.

---

# ISSUE



## ERROR MINTING DAPP

An error message appear when attempting to mint an NFT from the UI.

Internal error due to network configuration, will set time aside to fix issue and create a brief on how to avoid problems for potential new users

# CONCLUSIONS



## POTENTIAL FOR ALL-IN-ONE ANALYSIS

Created a solidity smart contract and front end framework that can eventually be a one-stop-shop for all nft launches.

## EASILY CUSTOMIZABLE

Easy implementation and seamless process.

## NEXT STEPS

- PULL METHOD
- MINTING DAPP
- PAYMENT SPLITTER
- NFT SMART CONTRACT



---

# REFERENCES

<https://docs.openzeppelin.com/contracts/2.x/api/payment>

<https://github.com/kissikelvin/Payment-Splitter-NFT-Dapp-Launch>

<https://ipfs.io/>

<https://www.pinata.cloud/>