

# FALLEN STAR

(Trading Meteor fragments as NFT's)

Jahnavi Rudraraju

UBID: 50464467

jahnavir@buffalo.edu

## **MOTIVATION:**

The main idea of this project is to save the space history, meteor fragments itself is an antique collection piece. Users (Here users are customers). If users do not have an active certification of the type of meteoroid, they are holding they can create an NFT based on the exact details of the material. In this way we can keep a track of rarest elements. Example: Cheylabinsk meteor: fragments are sold in auction, the person who bought in the auction was given a certification of the type of meteoroid and his details on it. We are implementing the same technique by providing certification in the form of NFT to everyone who comes with the meteoroids.

## **WHY TO BUY THESE PRODUCTS:**

Meteoroids are considered as a best investment in pawn shops based on the element present inside the fragment, we are doing the same process but in a digital marketplace. The product values depend upon the age of the meteoroid, if the meteoroid is held as a family heirloom the value can be increased and at a time it can decrease because, even if the meteoroid is old and kept for a long time on earth when comparing the actual age of the meteor to the new meteor entering earth may be older than the one kept as a family heirloom.

The value of these fragments entirely depends on

- Age of the meteor (Includes the spacetime)
- Material rarity
- Liability issues for uncertified meteor fragments (uncertified fragments are risky because people can easily start a lawsuit about the ownership of the fragment).

If the meteor age is older, then it is more valuable the demand for obtaining one will increase thus increased its share price. We will base the rarity of the meteor to increase and decrease stock price.

## **DETAIL DESCRIPTION OF HOW SYSTEM WORKS:**

The system works in a simple way- Trading of meteoroids as NFT's. we create a user-friendly interface where users(customers) can sell or purchase NFTs. They can buy the whole NFT, or they can buy shares (shares are fragments of an NFT split into pieces and sold). We have two types of customers

- Customers with valid certification, they can directly start selling and purchasing shares because we don't get any issues if they have a valid certificate. We will create a NFT for their meteoroid. The image on NFT will be a replica of the meteoroid even the cuts, shapes, dents everything In-order to maintain the uniqueness and rarity.
- Customers without valid certification, they will also get the same benefits and perks as the other category customers, but they should sign a contract stating that they will be responsible for all the liability issues in the future.

Users are required to use ETH token when purchasing NFT shares, or they can trade with their own NFT fragment with another person's fragment.

Users can also sell their NFT as a whole or they can fragment their NFT into pieces we will set a threshold of 20 pieces in an NFT. Because we are dealing with image of the meteoroid if we allow more than 20 then the uniqueness and rarity of the meteoroid cannot be traced, or we will be met with a lot of legal issues because if we split into a million pieces the ownership of every fragment is difficult to keep in track and if the meteoroid is black, we cannot trace the details which can describe the history of the meteoroid.

## **MONETIZATION:**

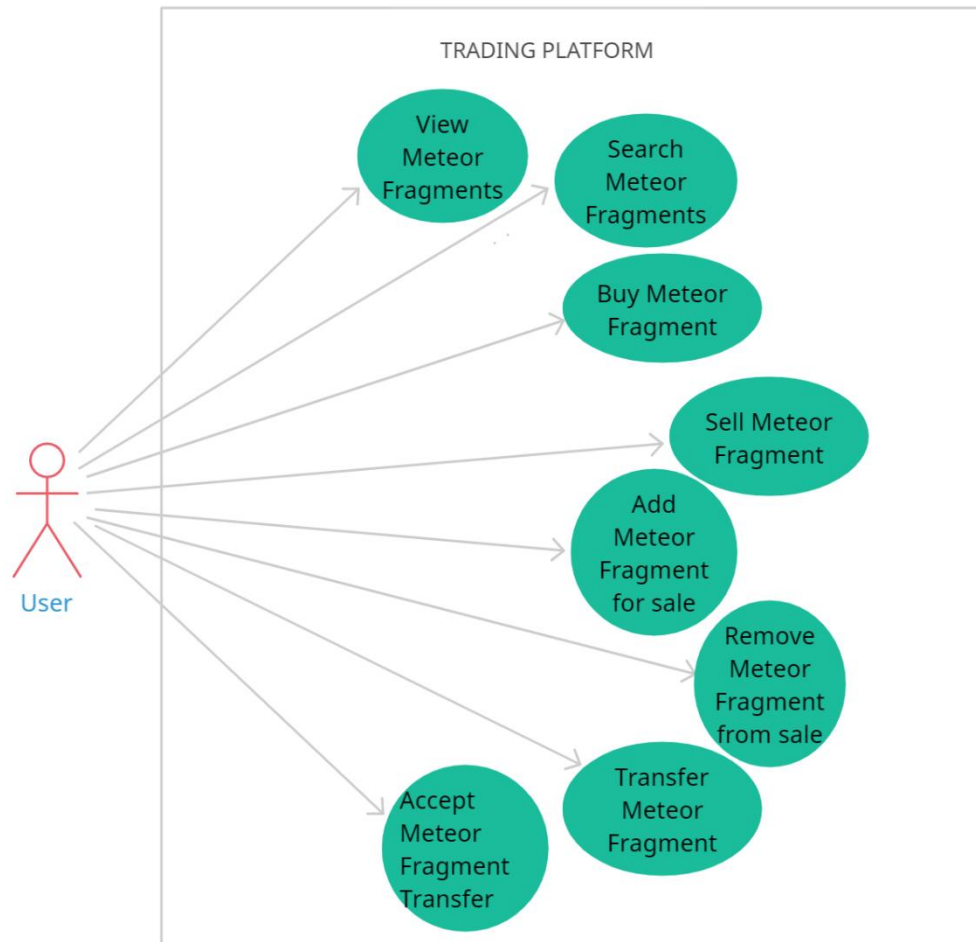
We create customers NFT's we will take some percentage; customers can give us ETH tokens the amount will be determined based on the quality checks of the material. Customers can fragment their created NFT's and sell them as shares- they can offer these shares to us instead of ETH tokens, they should provide valid ownership for the piece of fragment if they decide to give the shares as commission.

## **BACKGROUND CHECK (Who are eligible to join):**

Before launching the marketplace, we are checking with some meteoroid specialists who can help us understand how to check and what to check in the piece of rock.

Users should agree to the terms and conditions where we state that if any lawsuits or liability issues occur for the uncertified meteoroids in the future the original seller will be held liable for all the damages.

## USE CASE DIAGRAM EXPLAINING PROJECT IDEA:

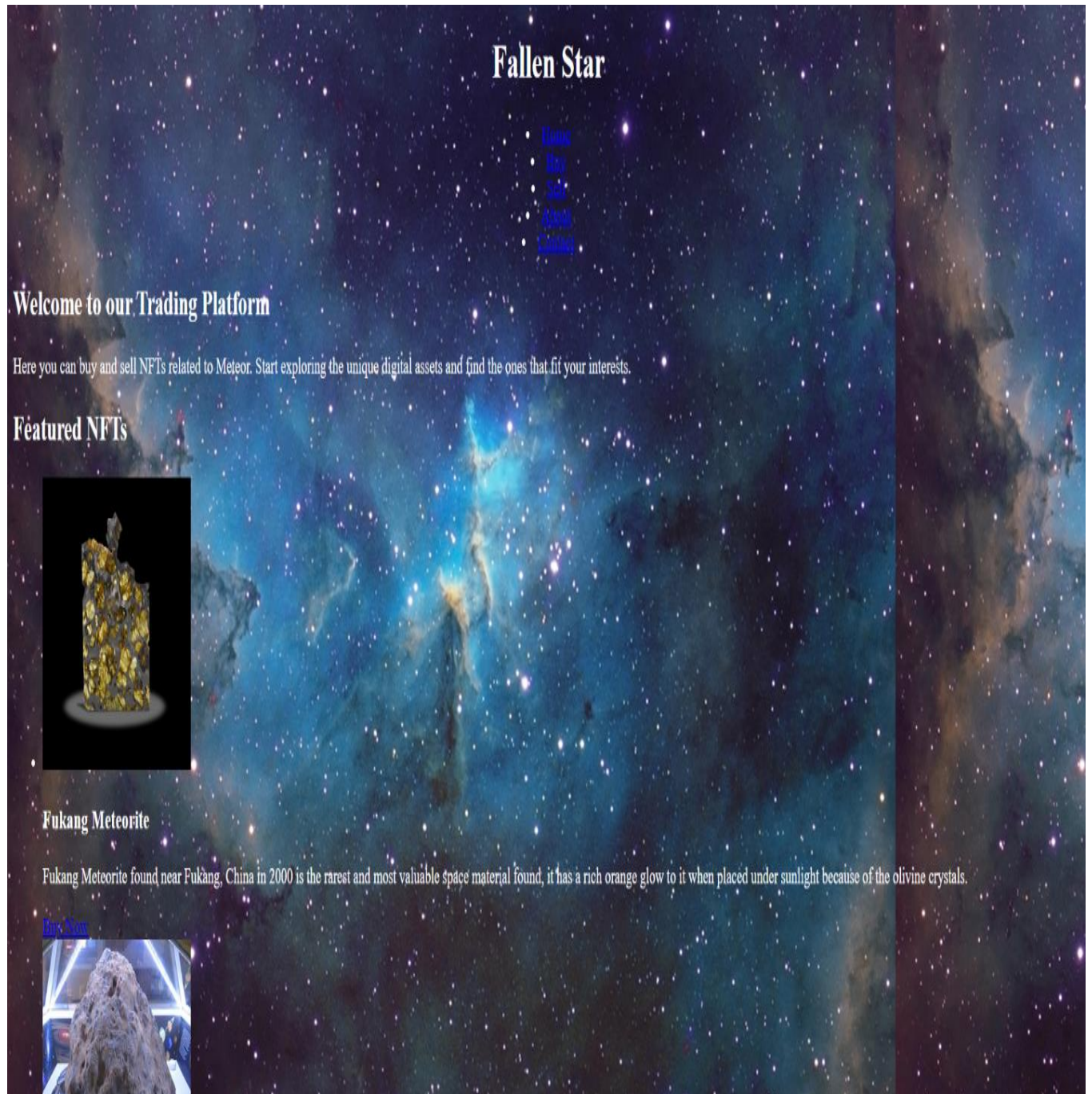


The User is the person who interacts with the DApp (Trading Platform) to buy, sell, or transfer meteor fragments as NFTs. The Trading Platform is the system that facilitates these transactions.

- **View Meteor Fragments:** This use case allows the User to view a list of all meteor fragments currently available for sale on the Trading Platform.
- **Search Meteor Fragments:** This use case allows the User to search for specific meteor fragments by name, date, or other criteria.
- **Buy Meteor Fragment:** This use case allows the User to purchase a meteor fragment from the Trading Platform.
- **Sell Meteor Fragment:** This use case allows the User to list a meteor fragment for sale on the Trading Platform.
- **Add Meteor Fragment for sale:** This use case allows the User to add a meteor fragment to the Trading Platform's list of items for sale.
- **Remove Meteor Fragment from sale:** This use case allows the User to remove a meteor fragment from the Trading Platform's list of items for sale.
- **Transfer Meteor Fragment:** This use case allows the User to transfer ownership of a meteor fragment to another User.

- **Accept Meteor Fragment Transfer:** This use case allows the Trading Platform to confirm the transfer of ownership of a meteor fragment from one User to another.

### UI Framework:







### Chelyabinsk Meteorite

Chelyabinsk Meteorite entered Earth's atmosphere in 2013 near Ural region in Russia, the meteorite is made up of chondrite (metal iron, olivine, sulfides).

[Buy Now](#)



### Zagami Martian Meteorite

Zagami Martian Meteorite discovered in 1962 near Katsina Province, Nigeria is a precious meteorite containing martian atmosphere trapped inside it. This meteorite is made up of melted glass.

[Buy Now](#)

## About Us

We are a team of passionate individuals who believe that NFTs have the potential to revolutionize the way we trade digital assets. Our platform is dedicated to bringing the best Meteor NFTs to collectors and investors around the world.

© 2023 Meteor NFT Trading Platform. All rights reserved.

## **REFERENCES:**

1. B. Ramamurthy. Blockchain in Action (BIA), book
2. Lecture Notes
3. [Introduction to smart contracts | ethereum.org](#)
4. UI creation : [HTML Styles CSS \(w3schools.com\)](#)

## APPROVED BY:

ZHAOFENG

## CODE IMPLEMENTATION OF THE DIGITAL ASSETS-TOKEN SMART CONTRACTS:

```
1 //SPDX-License-Identifier: UNLICENSED
2
3 pragma solidity ^0.8.0;
4
5 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
6 contract TradingMeteorFragment is ERC721 {
7     // Define a struct to store information about each meteor fragment
8     struct MeteorFragment {
9         string name;
10        uint256 rarity; // Rarity score (0-100)
11        uint256 price; // Price in ether
12    }
13
14    // Array of all meteor fragments
15    MeteorFragment[] public meteorFragments;
16
17    // Mapping from meteor fragment index to owner address
18    mapping (uint256 => address) public meteorFragmentToOwner;
19
20    constructor() ERC721("TradingMeteorFragment", "TMF") {}
21
22    // Function to create a new meteor fragment and tokenize it
23    function createMeteorFragment(string memory _name, uint256 _rarity, uint256 _price) public {
24        uint256 meteorFragmentId = meteorFragments.length;
25        meteorFragments.push(MeteorFragment(_name, _rarity, _price));
26        _mint(msg.sender, meteorFragmentId);
27        meteorFragmentToOwner[meteorFragmentId] = msg.sender;
28    }
29
30    // Function to buy a meteor fragment
31    function buyMeteorFragment(uint256 _meteorFragmentId) public payable {
```

```

29
30 // Function to buy a meteor fragment
31 function buyMeteorFragment(uint256 _meteorFragmentId) public payable {
32     require(meteorFragmentToOwner[_meteorFragmentId] != address(0), "Meteor fragment does not exist");
33     address payable owner = payable(meteorFragmentToOwner[_meteorFragmentId]);
34     require(msg.value >= meteorFragments[_meteorFragmentId].price, "Insufficient funds");
35     _transfer(owner, msg.sender, _meteorFragmentId);
36     meteorFragmentToOwner[_meteorFragmentId] = msg.sender;
37     owner.transfer(msg.value);
38 }
39
40 function sellMeteorFragment(uint256 meteorFragmentId, uint256 price) public {
41     require(_exists(meteorFragmentId), "meteorFragment Id does not exist");
42     require(msg.sender == ownerOf(meteorFragmentId), "Only the owner can sell");
43     meteorFragments[meteorFragmentId].price=price;
44 }
45 }
46

```

0 ☐ listen on all transactions

✓ [block:8 txIndex:0] from: 0xE9E...31487 to: TradingMeteorFragment.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0xa53...8a7a5

Debug ▼

# **CSE 526 BLOCKCHAIN APPLICATION DEVELOPMENT PROJECT PHASE-II**

## **‘Fallen Star’ NFT trading Platform**

**Jahnavi Rudraraju**

**50464467**

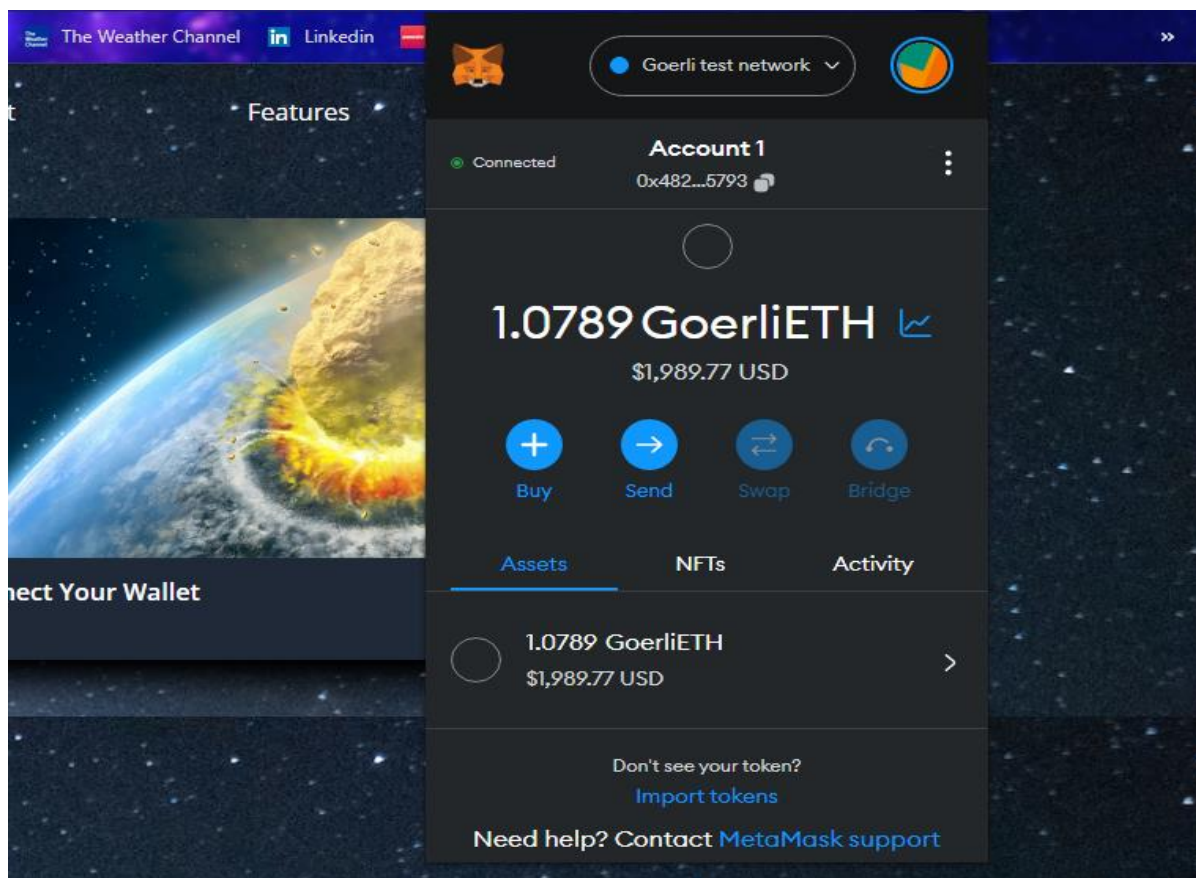
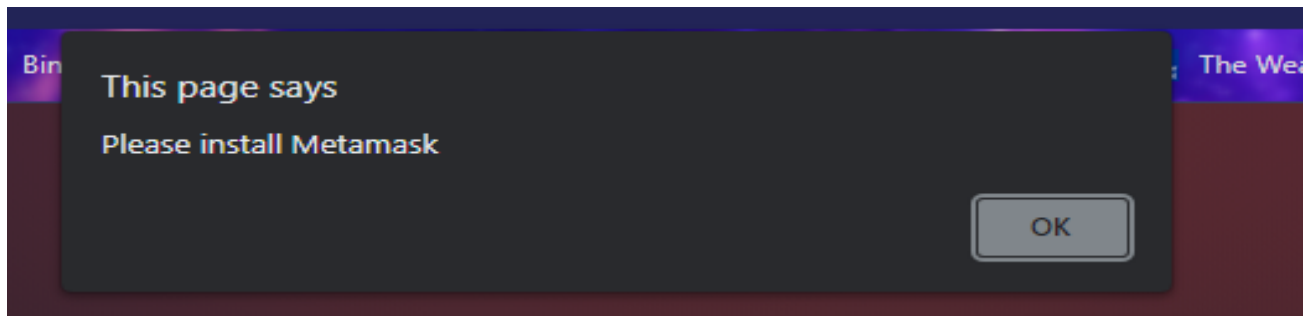
**[jahnavir@buffalo.edu](mailto:jahnavir@buffalo.edu)**

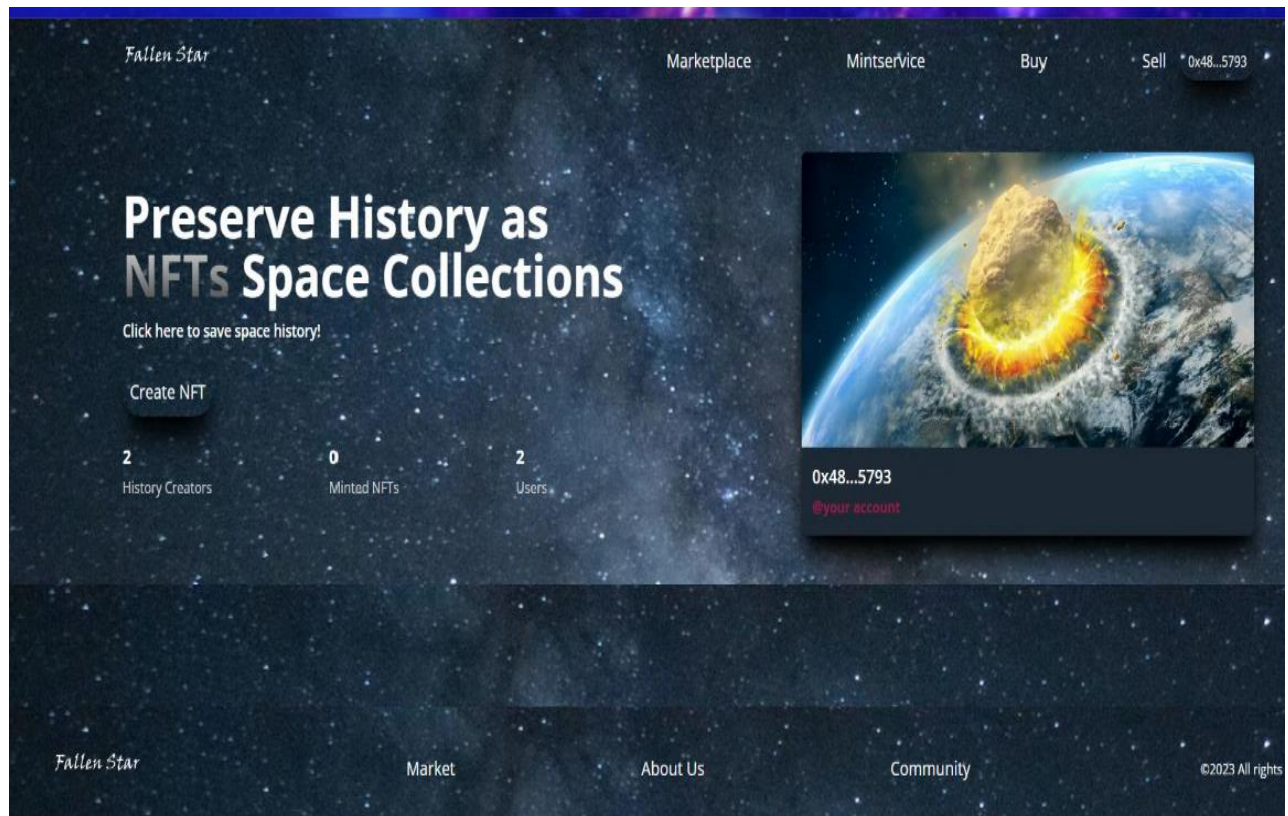


## Implementation:

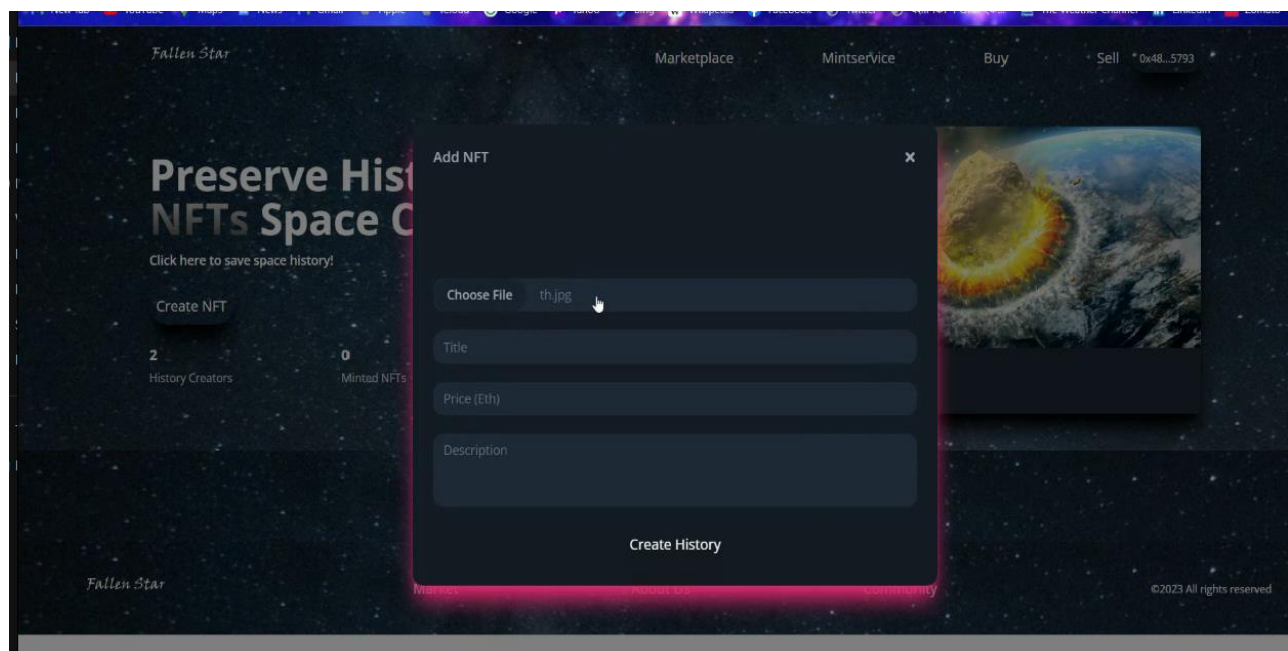
The trading platform is built using react application, the activities running in the web site are designed using truffle, tailwind, and some solidity smart contracts for transactions, connecting accounts etc.

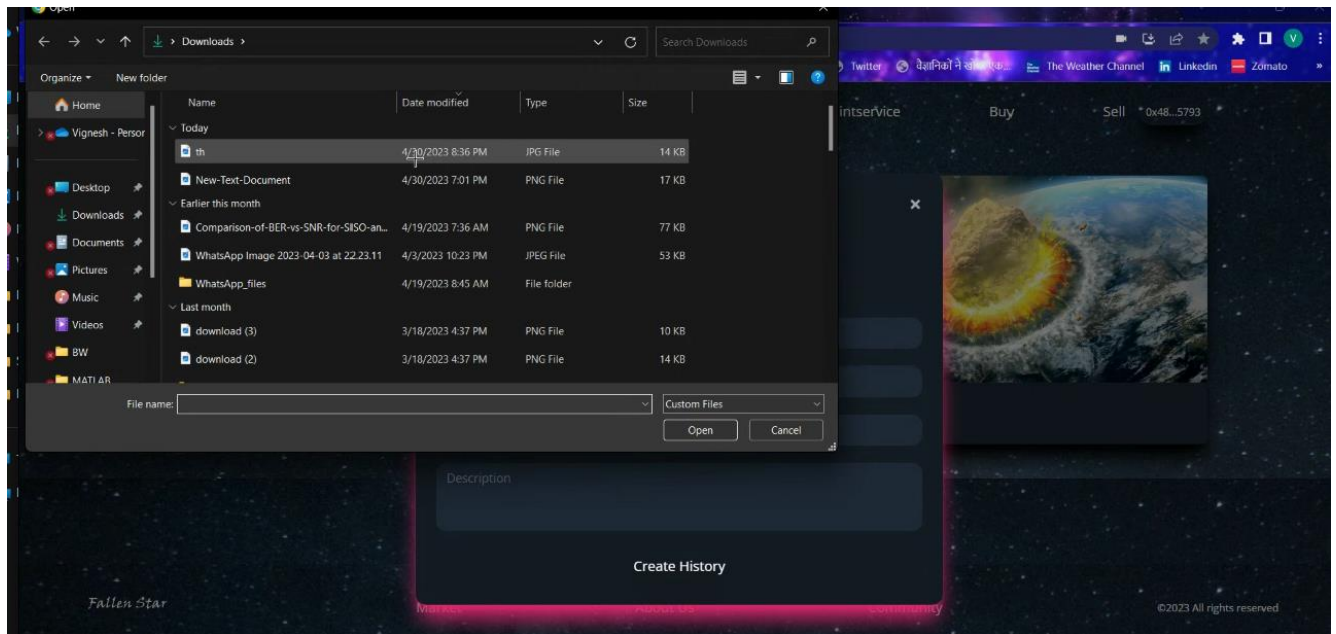
First we need to connect our metamask wallet to the web page, here we have two ways the site works, if metamask is unavailable in your system then it shows 'Please install metamask'. If you already have metamask it brings a prompt where you can connect to metamask account.



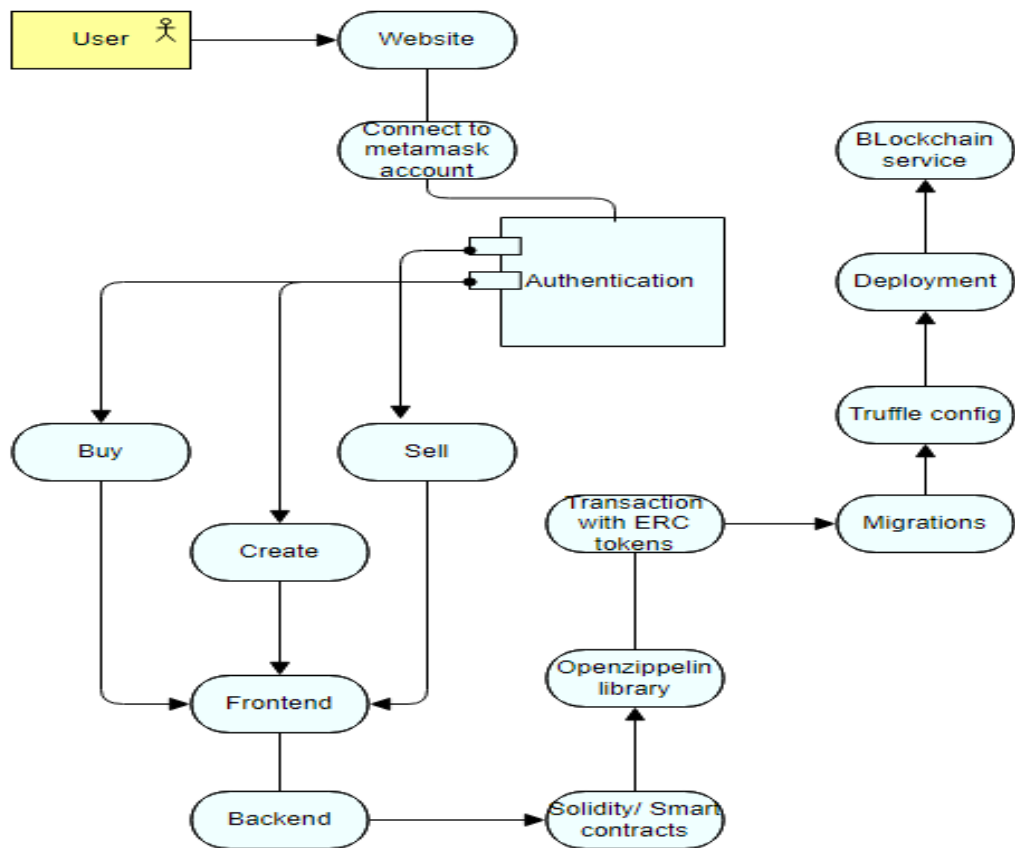


Now our account is connected to the web site, we can start creating NFT's, buying and selling in marketplace. Below is my activity diagram from phase 1



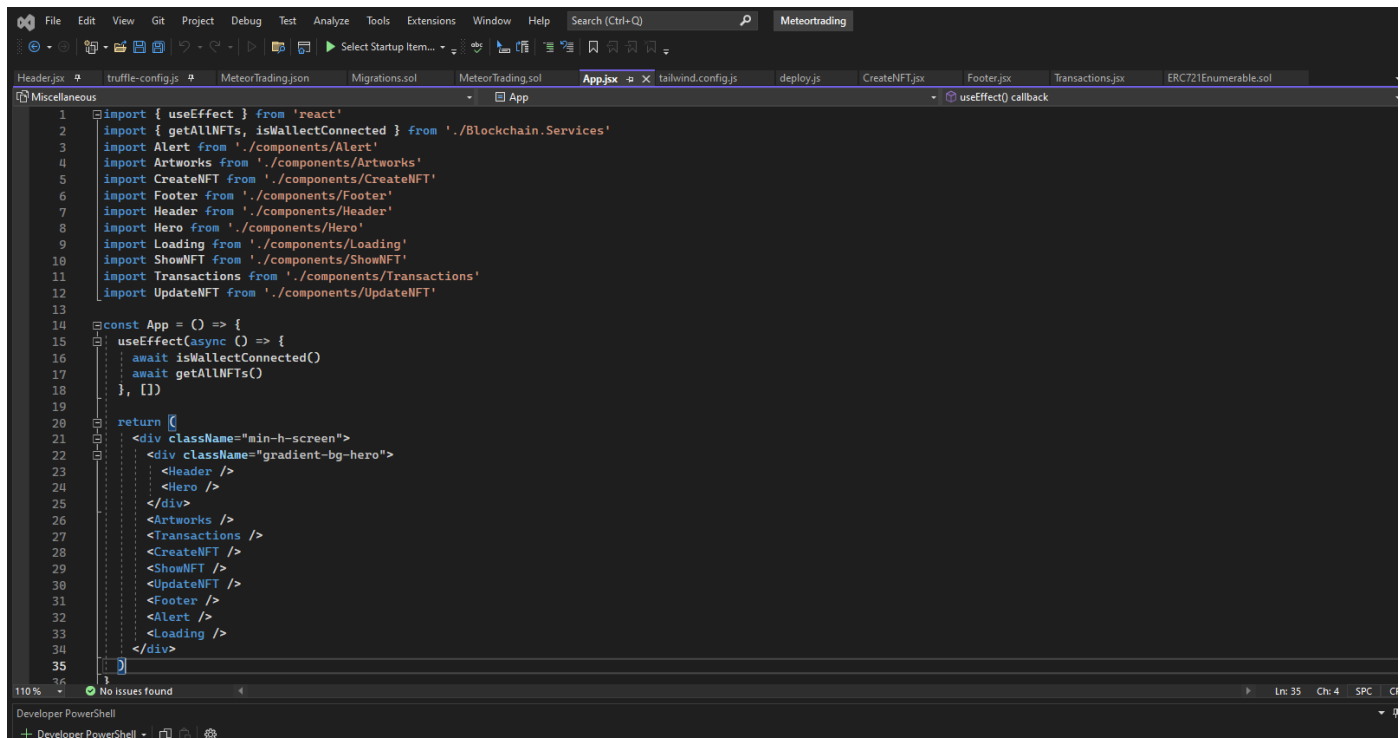


### Architecture Diagram:

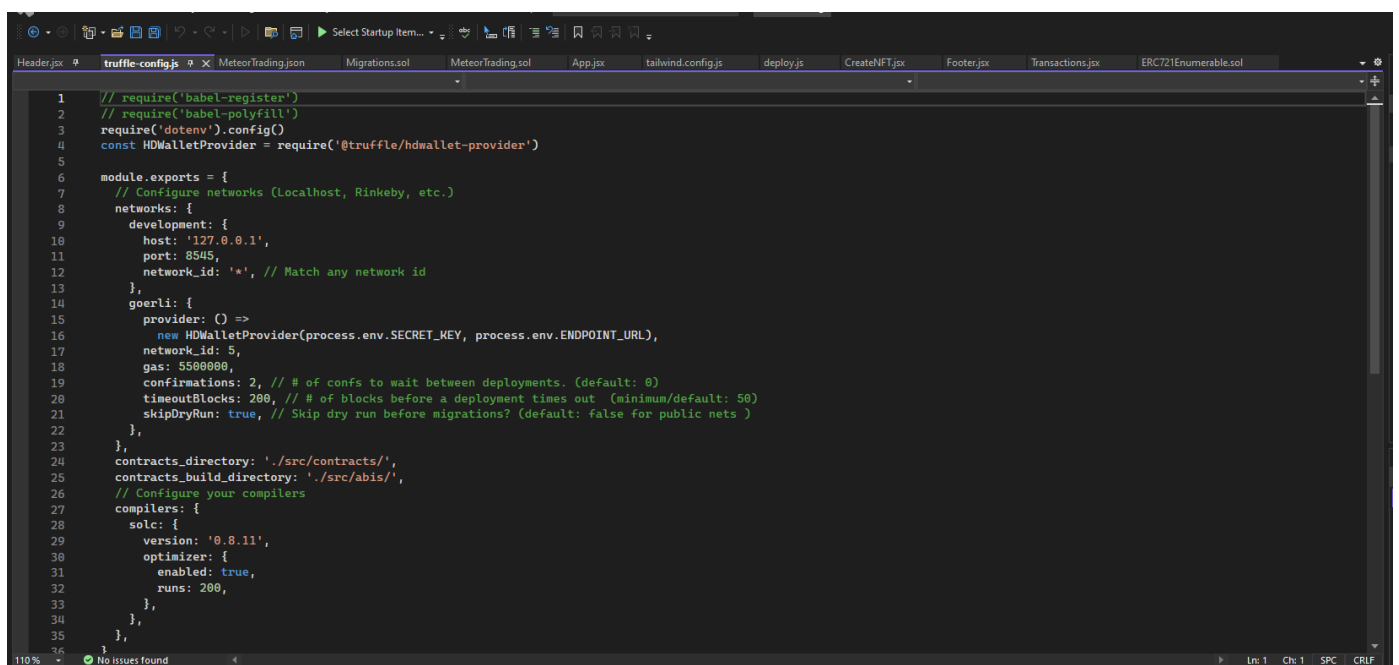


## Code:

Here is the implementation of smart contracts and code for all the frontend and backend process happening while the transaction are running. The coordination of all the programs are displayed in the above architecture diagram.



```
1 import { useEffect } from 'react'
2 import { getAllNFTs, isWalletConnected } from './Blockchain.Services'
3 import Alert from './components/Alert'
4 import Artworks from './components/Artworks'
5 import CreateNFT from './components/CreateNFT'
6 import Footer from './components/Footer'
7 import Header from './components/Header'
8 import Hero from './components/Hero'
9 import Loading from './components/Loading'
10 import ShowNFT from './components/ShowNFT'
11 import Transactions from './components/Transactions'
12 import UpdateNFT from './components/UpdateNFT'
13
14 const App = () => {
15   useEffect(async () => {
16     await isWalletConnected()
17     await getAllNFTs()
18   }, [])
19
20   return (
21     <div className="min-h-screen">
22       <div className="gradient-bg-hero">
23         <Header />
24         <Hero />
25       </div>
26       <Artworks />
27       <Transactions />
28       <CreateNFT />
29       <ShowNFT />
30       <UpdateNFT />
31       <Footer />
32       <Alert />
33       <Loading />
34     </div>
35   )
36 }
```



```
1 // require('babel-register')
2 // require('babel-polyfill')
3 require('dotenv').config()
4 const HDWalletProvider = require('@truffle/hdwallet-provider')
5
6 module.exports = {
7   // Configure networks (localhost, Rinkeby, etc.)
8   networks: {
9     development: {
10       host: '127.0.0.1',
11       port: 8545,
12       network_id: '*', // Match any network id
13     },
14     goerli: {
15       provider: () =>
16         new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL),
17       network_id: 5,
18       gas: 5500000,
19       confirmations: 2, // # of confs to wait between deployments. (default: 0)
20       timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)
21       skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
22     },
23   },
24   contracts_directory: './src/contracts/',
25   contracts_build_directory: './src/abis/',
26   // Configure your compilers
27   compilers: {
28     solc: {
29       version: '0.8.11',
30       optimizer: {
31         enabled: true,
32         runs: 200,
33       },
34     },
35   },
36 }
```

```

1  async function main() {
2      const [deployer, feeAccount] = await ethers.getSigners();
3
4      console.log("Deploying contracts with the account:", deployer.address);
5      console.log("Assigning Fee Account with:", feeAccount.address);
6      console.log("Deployer balance:", (await deployer.getBalance()).toString());
7      console.log("FeeAccount balance:", (await feeAccount.getBalance()).toString());
8
9      // Get the ContractFactories and Signers here.
10     const Store = await ethers.getContractFactory("Store");
11
12     // deploy contracts
13     const store = await Store.deploy('Freshers', feeAccount, 10);
14
15     // Save copies of each contracts abi and address to the frontend.
16     saveFrontendUtils(store, "Store");
17 }
18
19 function saveFrontendUtils(contract, name) {
20     const fs = require("fs");
21     const utilsDir = __dirname + "../src/utils";
22
23     if (!fs.existsSync(utilsDir)) {
24         fs.mkdirSync(utilsDir);
25     }
26
27     fs.writeFileSync(
28         utilsDir + '/' + [name] + '-address.json',
29         JSON.stringify({ address: contract.address }, undefined, 2)
30     );
31
32     const contractArtifact = artifacts.readArtifactSync(name);
33
34     fs.writeFileSync(
35         utilsDir + '/' + [name] + '.json',
36         JSON.stringify(contractArtifact, null, 2)
37     );
38 }

```

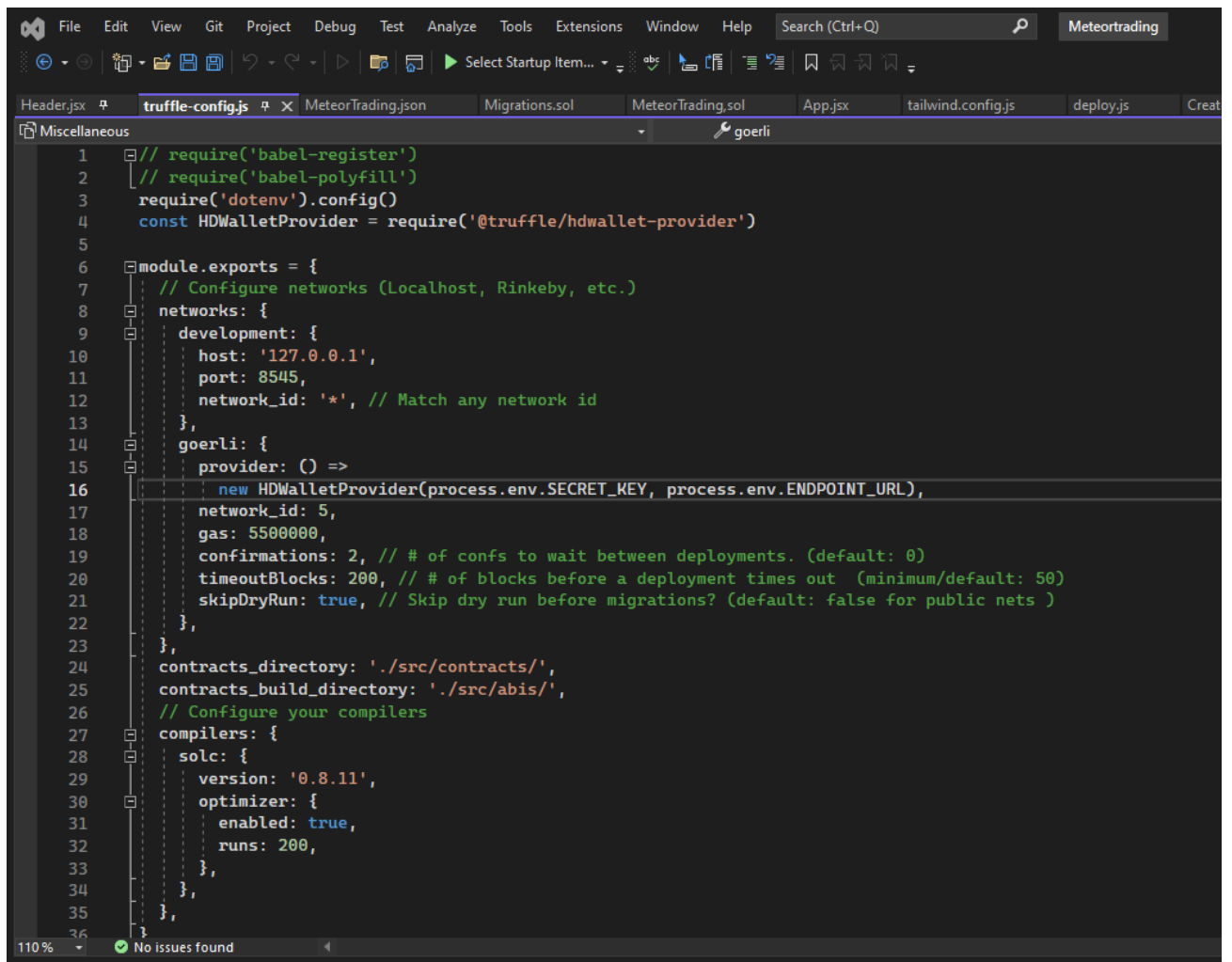
110% No issues found

```

Header.jsx  #  truffle-config.js  MeteorTrading.json  Migrations.sol  MeteorTrading.sol  App.jsx  tailwind.config.js  deploy.js  CreateNFT.js  Footer.jsx  Transactions.jsx  ERC721Enumerable.sol
1  // SPDX-License-Identifier: MIT
2  // OpenZeppelin Contracts v4.4.1 (token/ERC721/extensions/ERC721Enumerable.sol)
3
4  pragma solidity ^0.8.0;
5
6  import "./ERC721.sol";
7  import "./IERC721Enumerable.sol";
8
9  /**
10   * @dev This implements an optional extension of {ERC721} defined in the EIP that adds
11   * enumerability of all the token ids in the contract as well as all token ids owned by each
12   * account.
13   */
14  abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
15      // Mapping from owner to list of owned token IDs
16      mapping(address => mapping(uint256 => uint256)) private _ownedTokens;
17
18      // Mapping from token ID to index of the owner tokens list
19      mapping(uint256 => uint256) private _ownedTokensIndex;
20
21      // Array with all token ids, used for enumeration
22      uint256[] private _allTokens;
23
24      // Mapping from token id to position in the allTokens array
25      mapping(uint256 => uint256) private _allTokensIndex;
26
27      /**
28       * @dev See {IERC165-supportsInterface}.
29       */
30      function supportsInterface(bytes4 interfaceId)
31          public
32          view
33          virtual
34          override(IERC165, ERC721)
35          returns (bool)
36      {
37          return

```

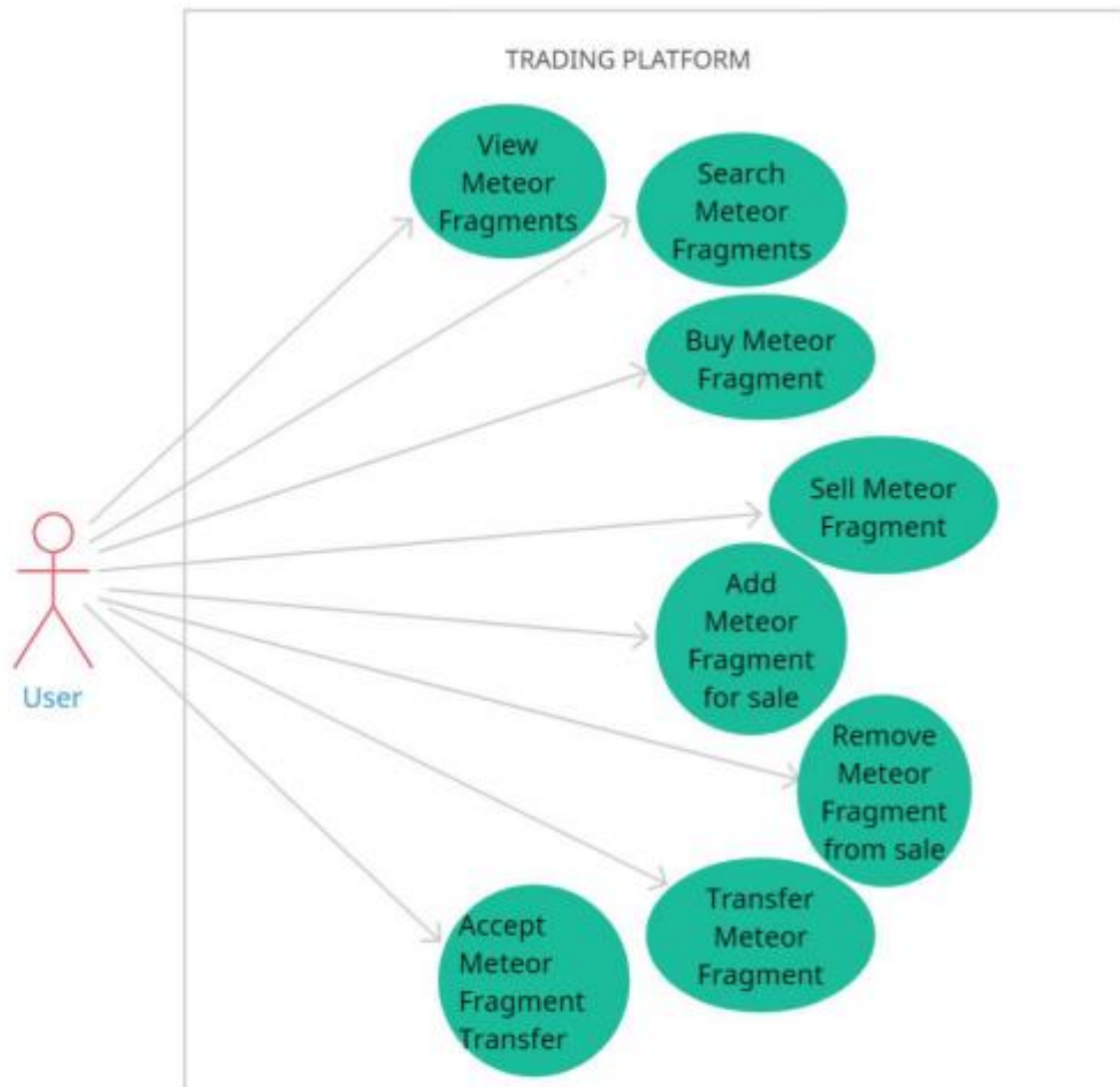




```
1 // require('babel-register')
2 // require('babel-polyfill')
3 require('dotenv').config()
4 const HDWalletProvider = require('@truffle/hdwallet-provider')
5
6 module.exports = {
7   // Configure networks (Localhost, Rinkeby, etc.)
8   networks: {
9     development: {
10       host: '127.0.0.1',
11       port: 8545,
12       network_id: '*', // Match any network id
13     },
14     goerli: {
15       provider: () =>
16         new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL),
17       network_id: 5,
18       gas: 5500000,
19       confirmations: 2, // # of confs to wait between deployments. (default: 0)
20       timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)
21       skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
22     },
23   },
24   contracts_directory: './src/contracts/',
25   contracts_build_directory: './src/abis/',
26   // Configure your compilers
27   compilers: {
28     solc: {
29       version: '0.8.11',
30       optimizer: {
31         enabled: true,
32         runs: 200,
33       },
34     },
35   },
36 }
```

110 % No issues found

Contract Diagram:



## **Merits and Demerits:**

### **Merits:**

- we can save or preserve the space history by converting them into NFTs
- It is easier to identify the owner of the meteoroid and maintain the locations of the meteoroid, by using their account description or in their information page.

### **Demerits:**

- Some people can replicate the picture of the meteoroid and demand false ownership
- Fake meteoroids will be more, we need to carefully examine the meteoroid before minting it.

# **CSE 526 BLOCKCHAIN APPLICATION DEVELOPMENT PROJECT PHASE-III**

## **'Fallen Star' NFT trading Platform**

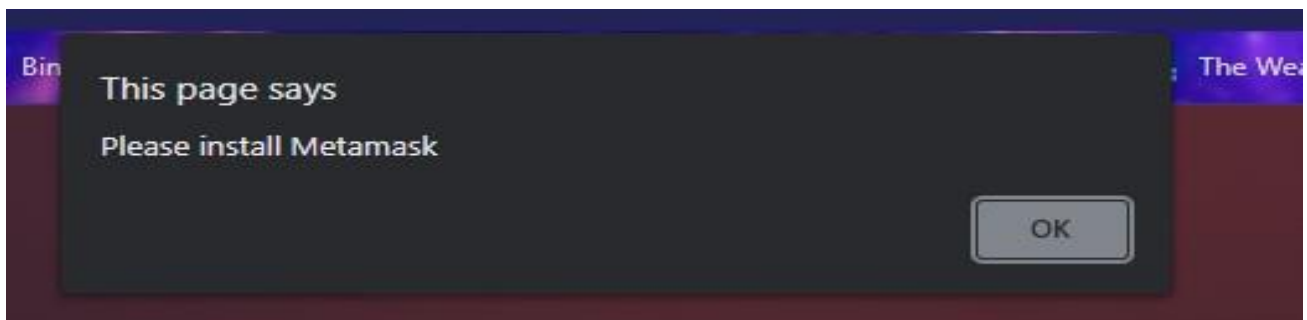
**Jahnavi Rudraraju**

**50464467**

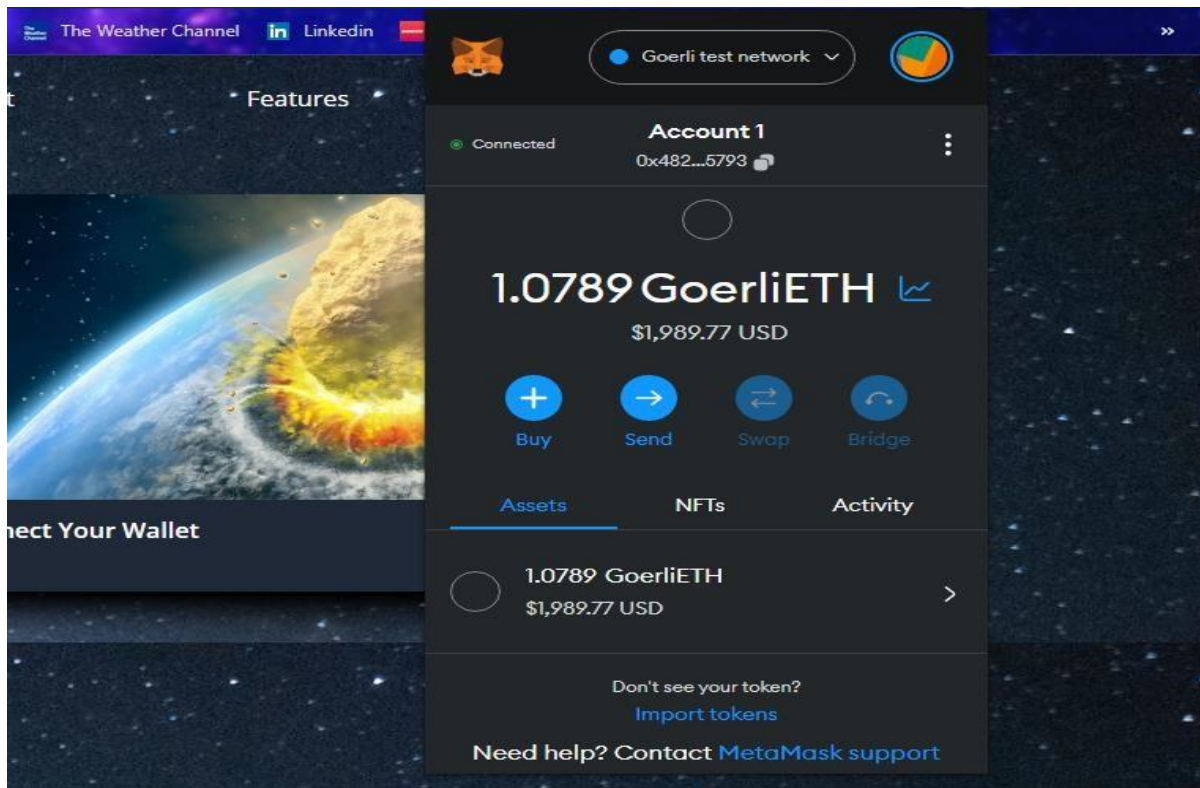
**[jahnavir@buffalo.edu](mailto:jahnavir@buffalo.edu)**

## Implementation:

This Dapp is built by react application all the styling and labelling of the website is created using tailwind.css, and for the transactions part I used ERC721 tokens with the help of truffle wallet handler. All the migrations and deployment code is mentioned below, I used INFURIA-API with the help of react application to deploy the website. Openzeppelin library is used for the migrations and the transaction of ERC721 solidity codes, all the applications mentioned in the architecture diagram and the contract diagram are mentioned in the package.json file. Here are some of the pictures of the working of website. If the metamask extension is unavailable to access a prompt shows up stating install metamask.

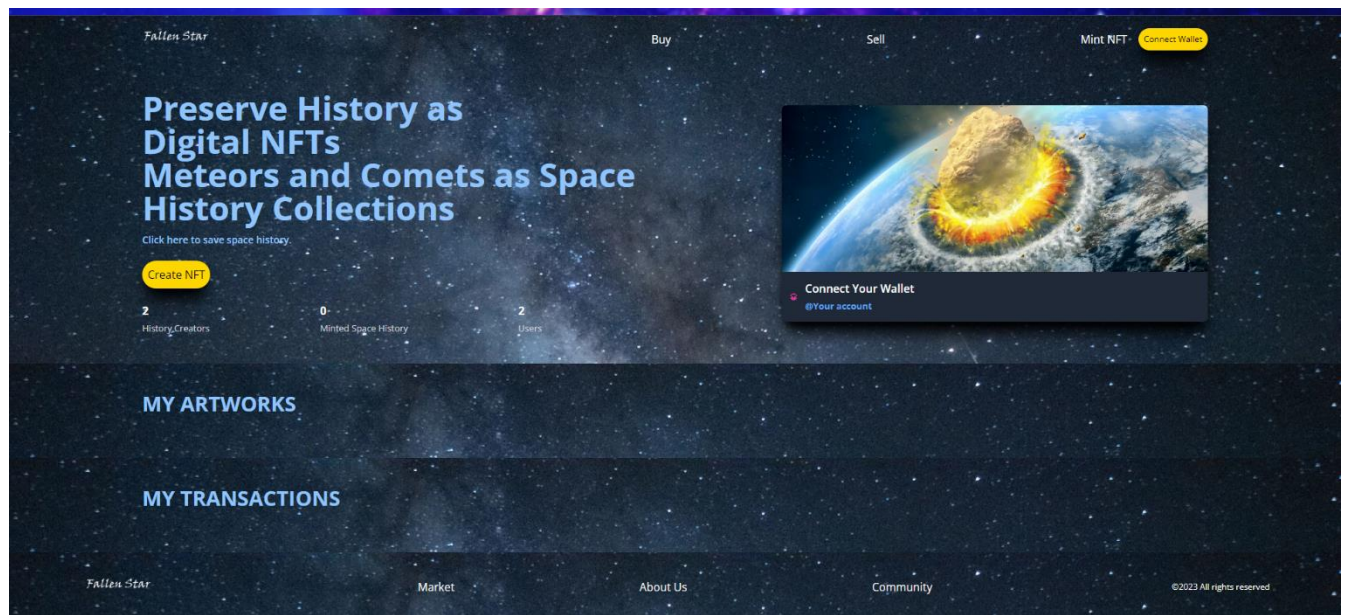


If we have metamask extension we get a prompt for connection to wallet

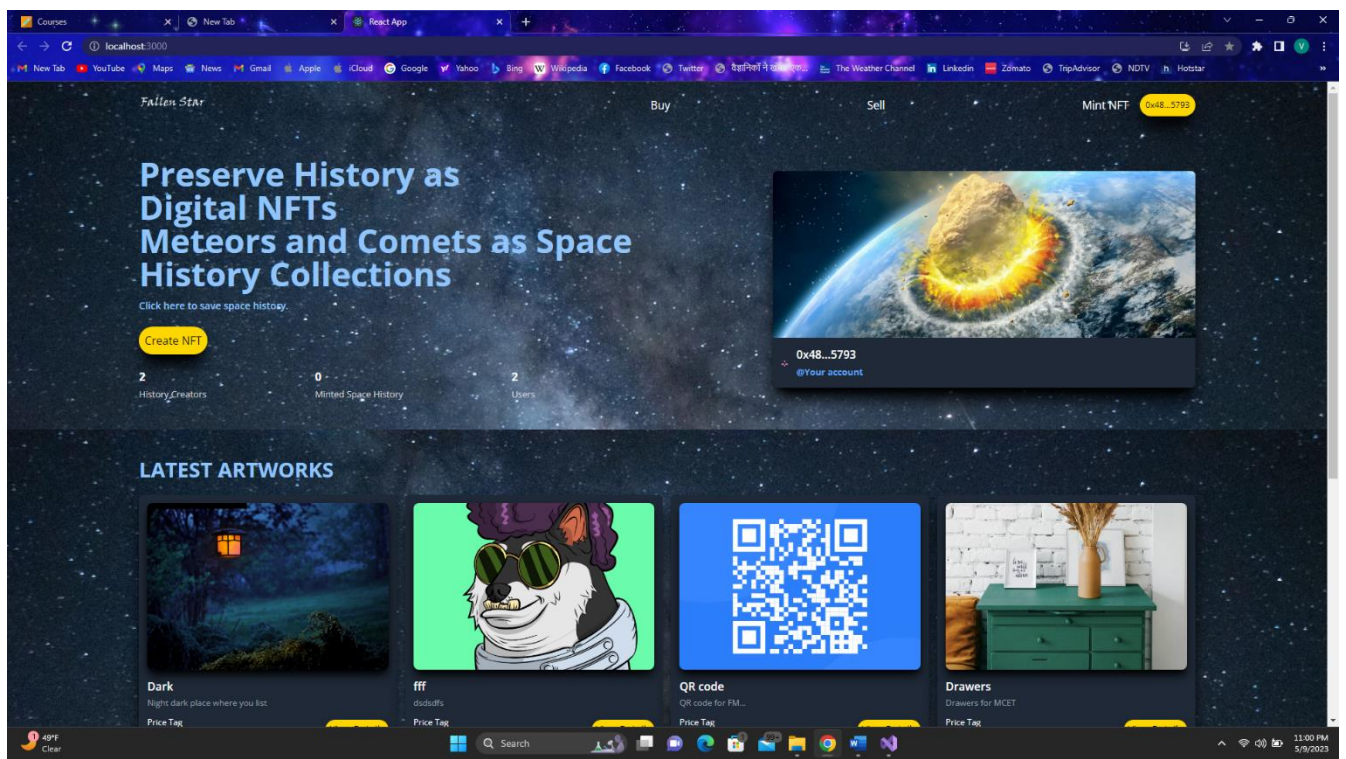




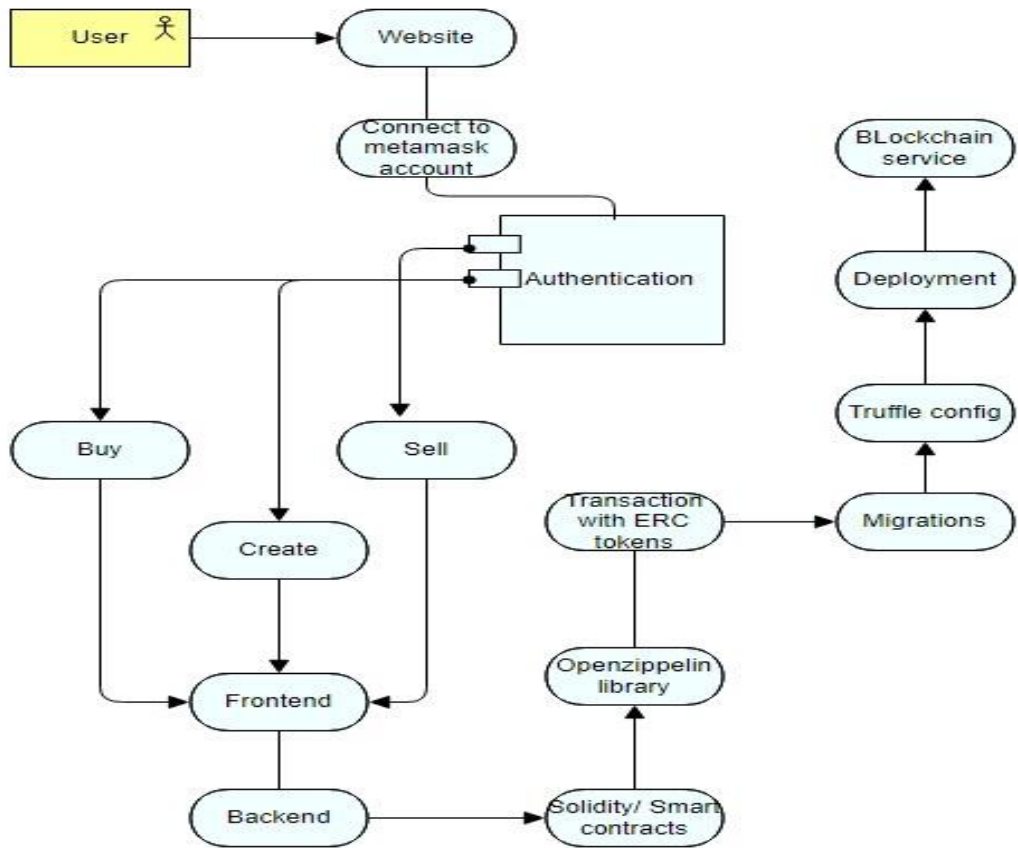
This is a screenshot of the website here we can create, buy and sell NFTs.



Now the account is connected to the web site, we can start creating NFT's, buying and selling in marketplace.

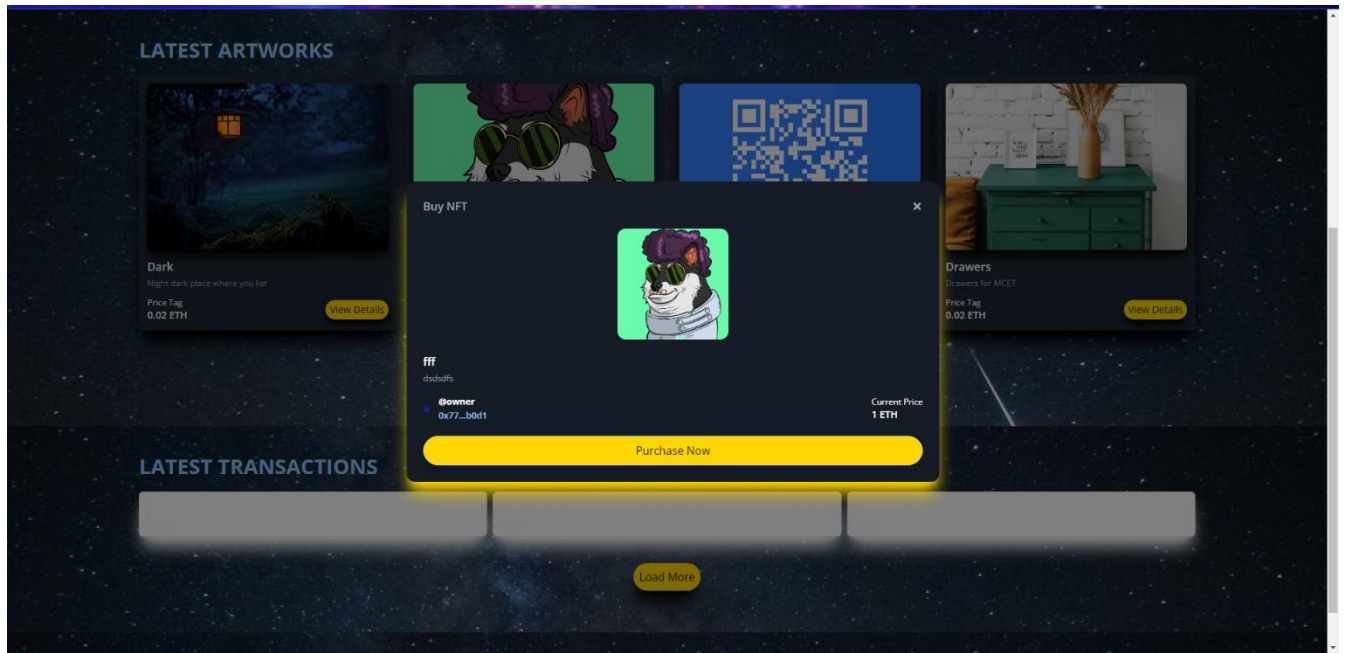


## Architecture Diagram:

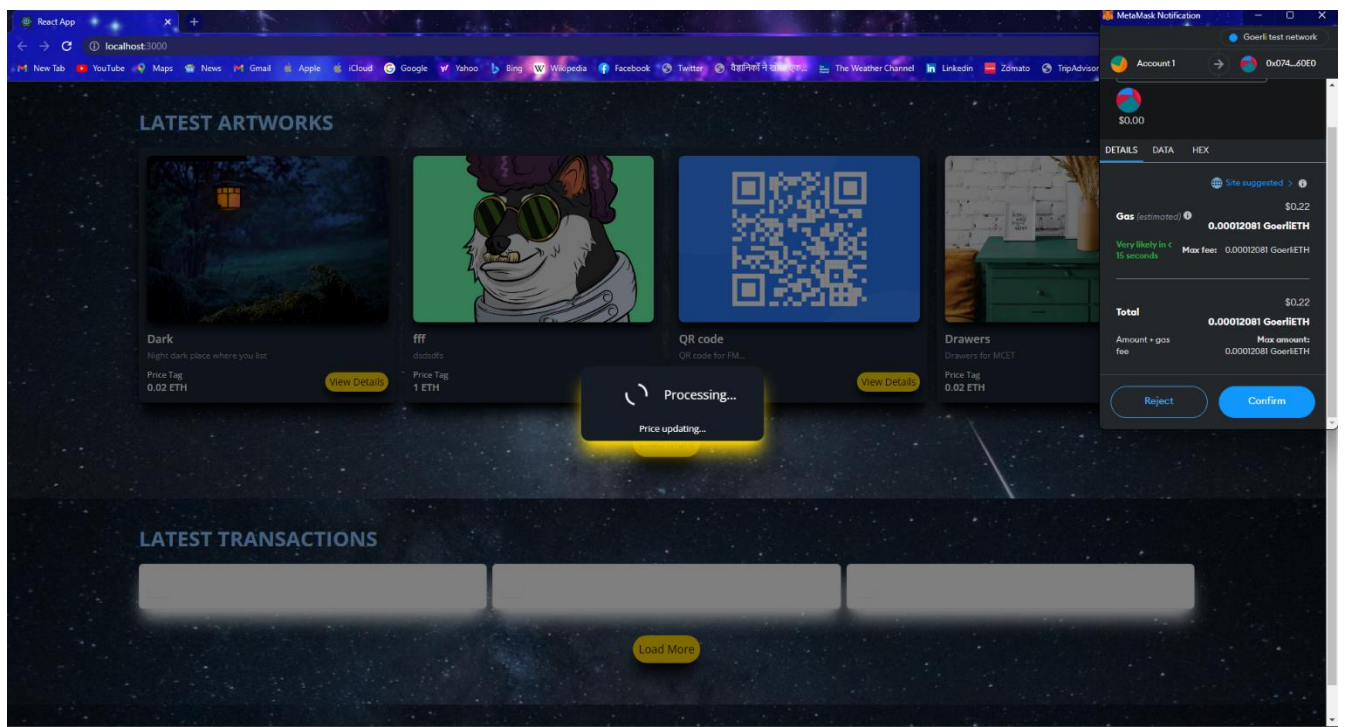


## Applications of the website:

These are some of the screenshots of the working of the website, Below is an example for buying an Existing NFT In the market of our website.

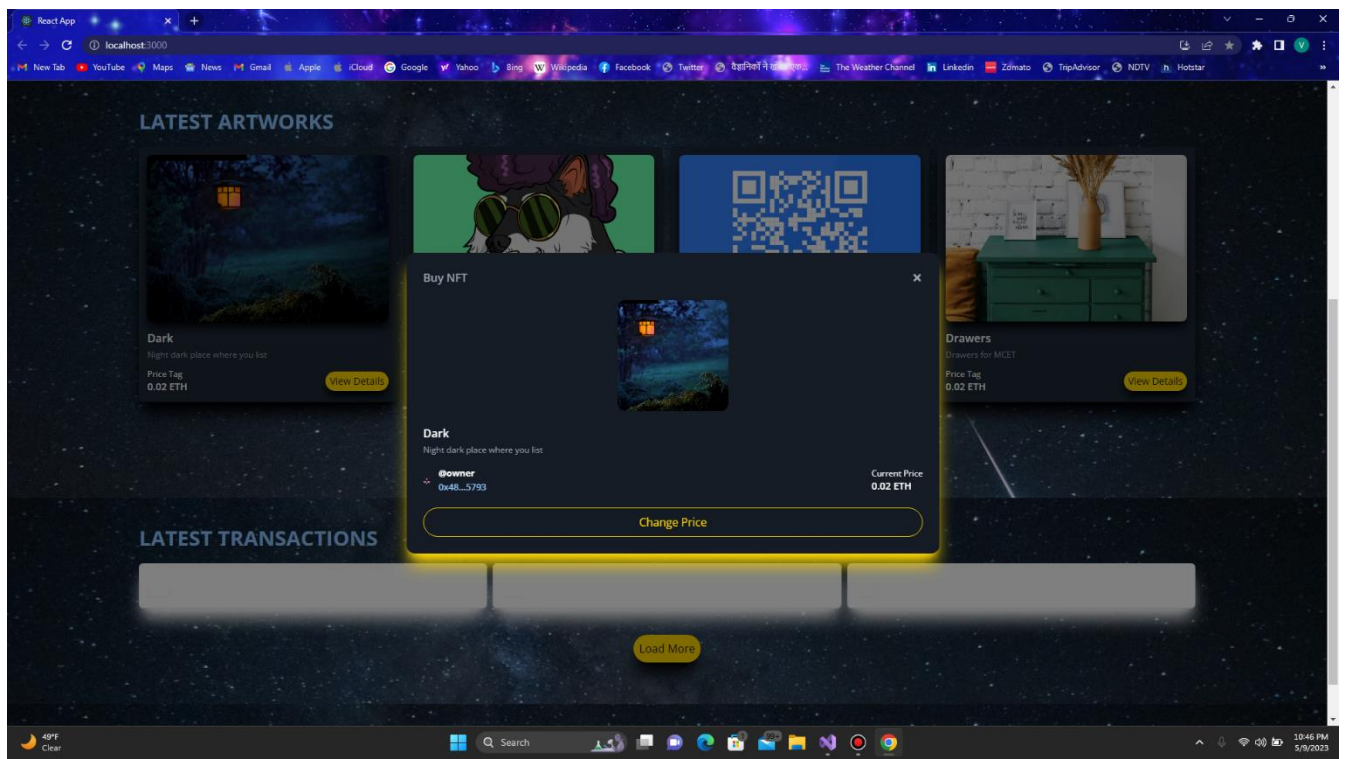


After selecting desired NFT it shows the confirmation and the Gas price.

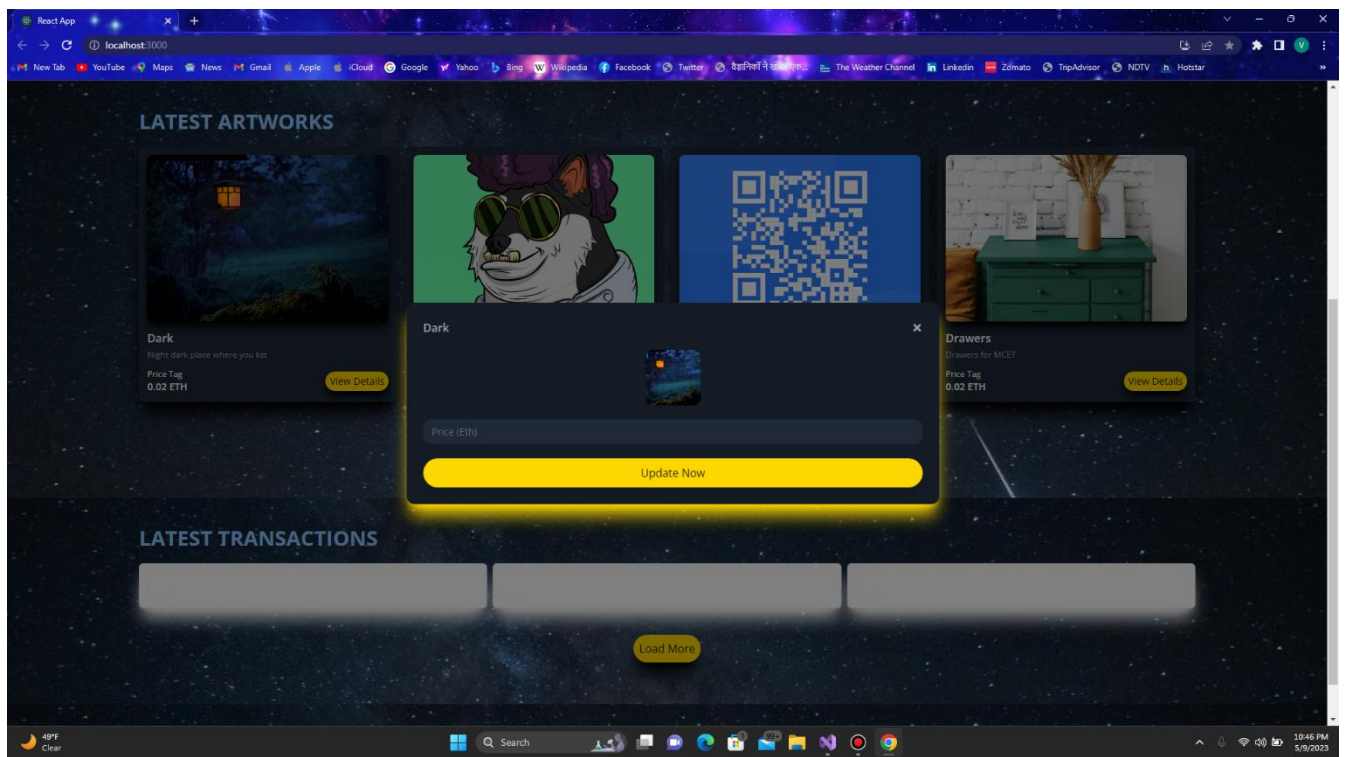




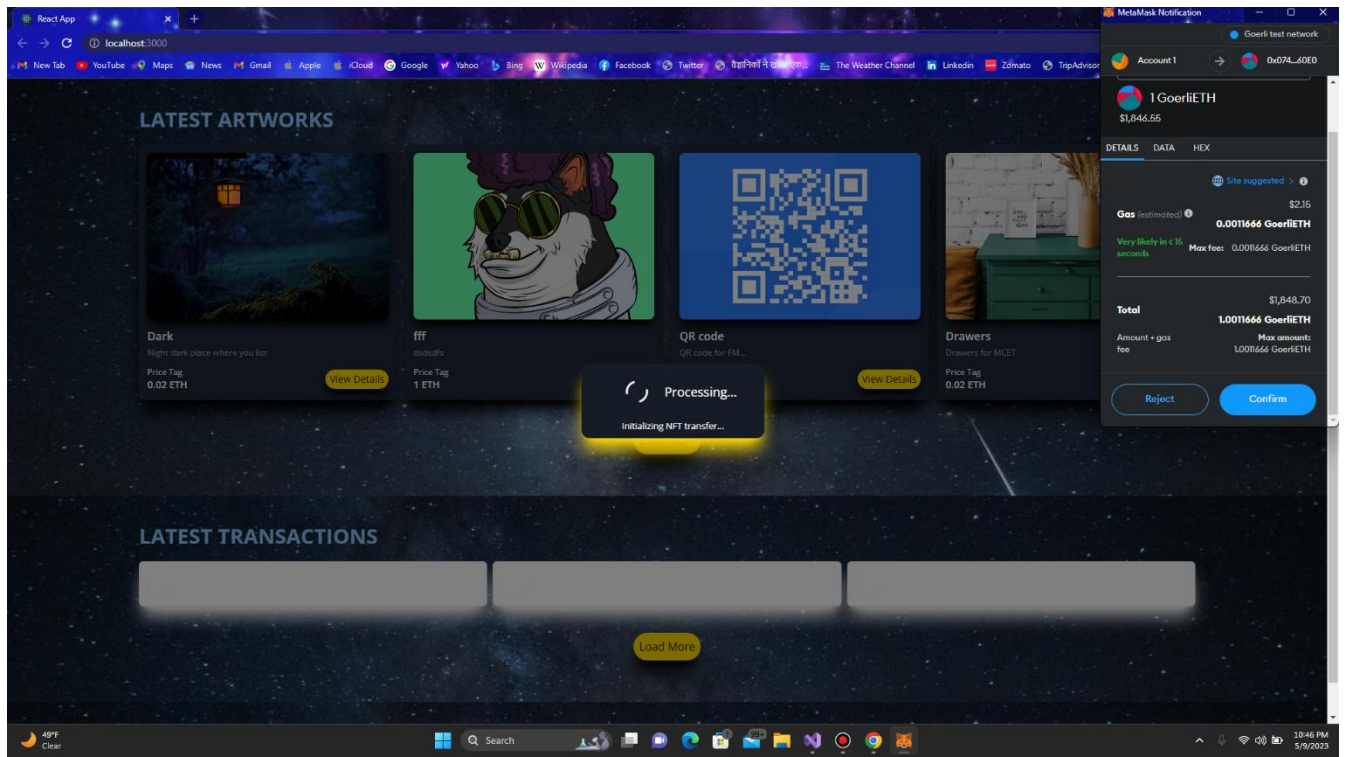
This is an example of owned NFTs we can change the price of the NFT and sell it in the market



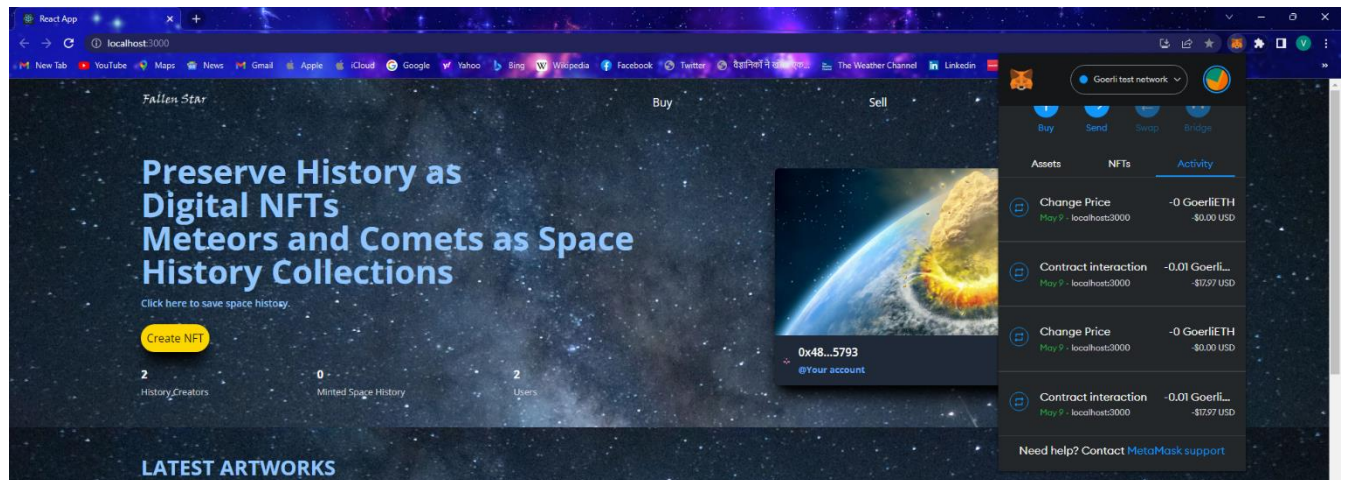
By clicking “Change Price” we get a prompt to increase value and update the value of the NFT.



We get the same transaction confirmation and the Gas price of the update.



Here is the list of transactions of all the NFT exchanges in my account.





## Code:

Here is the implementation of smart contracts and code for all the frontend and backend process happening while the transaction are running. The coordination of all the programs are displayed in the above architecture diagram.

The screenshot displays the Visual Studio Code editor with two main files open: `truffle-config.js` and `package.json`.

**truffle-config.js** (lines 1-35):

```
1 // require('babel-register')
2 // require('babel-polyfill')
3 require('dotenv').config()
4 const HDWalletProvider = require('@truffle/hdwallet-provider')
5
6 module.exports = {
7   // Configure networks (Localhost, Rinkeby, etc.)
8   networks: {
9     development: {
10       host: '127.0.0.1',
11       port: 8545,
12       network_id: '*', // Match any network id
13     },
14     goerli: {
15       provider: () =>
16         new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL),
17       network_id: 5,
18       gas: 5500000,
19       confirmations: 2, // # of confs to wait between deployments. (default: 0)
20       timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)
21       skipDryRun: true, // Skip dry run before migrations? (default: false for public nets)
22     },
23   },
24   contracts_directory: './src/contracts/',
25   contracts_build_directory: './src/abis/',
26   // Configure your compilers
27   compilers: {
28     solc: {
29       version: '0.8.11',
30       optimizer: {
31         enabled: true,
32         runs: 200,
33       },
34     },
35   },
36 }
```

**package.json** (lines 1-35):

```
1 {
2   "name": "MeteorTrading",
3   "private": true,
4   "version": "0.0.0",
5   "scripts": {
6     "start": "react-app-rewired start",
7     "build": "react-app-rewired build",
8     "test": "react-app-rewired test",
9     "eject": "react-scripts eject"
10  },
11  "dependencies": {
12    "ipfs-http-client": "^59.0.0",
13    "react": "^17.0.2",
14    "react-dom": "^17.0.2",
15    "react-hooks-global-state": "^1.0.2",
16    "react-icons": "^4.3.1",
17    "react-identicons": "^1.2.5",
18    "react-router-dom": "^6",
19    "react-scripts": "5.0.0",
20    "web-vitals": "^2.1.0",
21    "web3": "^1.7.1"
22  },
23  "devDependencies": {
24    "@openzeppelin/contracts": "^4.5.0",
25    "@tailwindcss/forms": "^0.4.0",
26    "@truffle/hdwallet-provider": "^2.1.10",
27    "assert": "^2.0.0",
28    "autoprefixer": "10.4.2",
29    "babel-polyfill": "6.26.0",
30    "babel-preset-env": "1.7.0",
31    "babel-preset-es2015": "6.24.1",
32    "babel-preset-stage-2": "6.24.1",
33    "babel-preset-stage-3": "6.24.1",
34    "babel-register": "6.26.0",
35    "buffer": "6.0.3",
36  },
37  "browserslist": [
38    "last 1 chrome version",
39    "last 1 firefox version",
40    "last 1 safari version"
41  ]
42 }
```

The bottom panel shows the **Developer PowerShell** terminal with the following output:

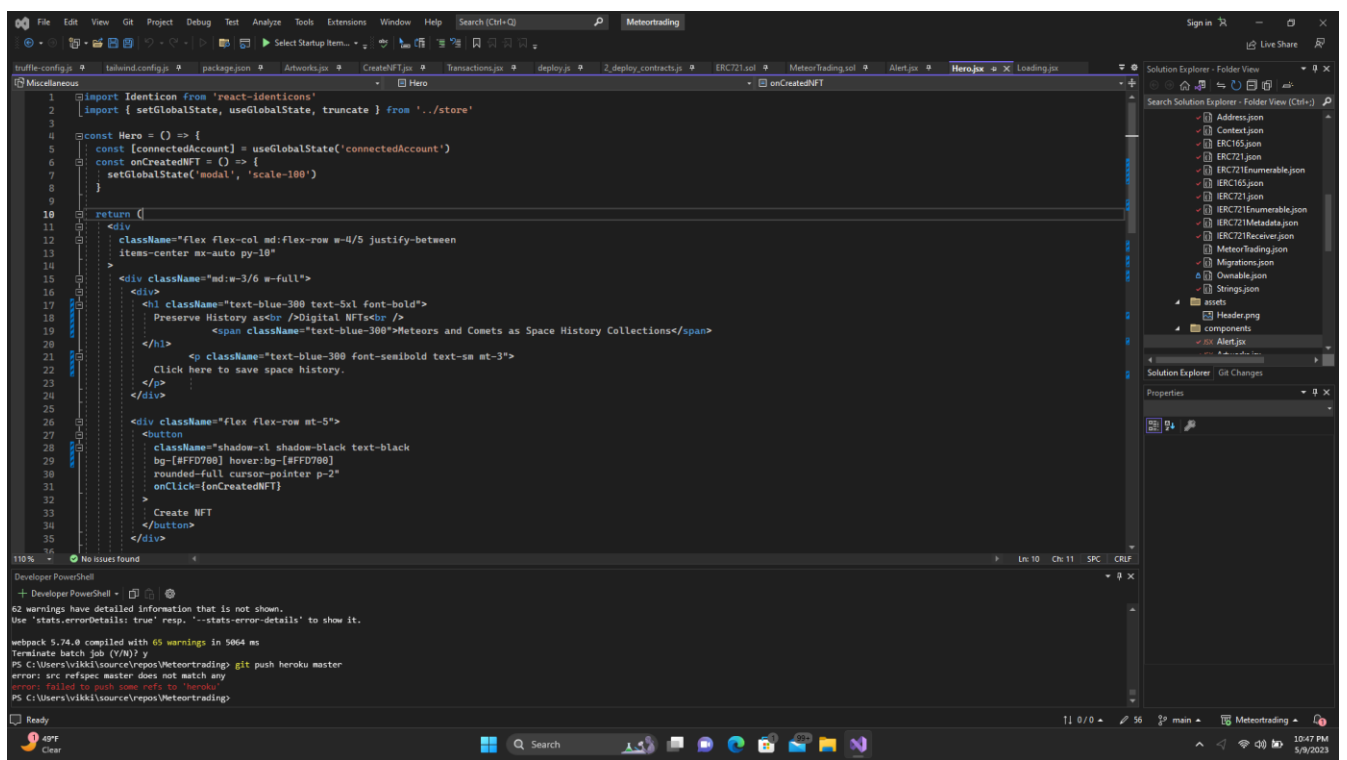
```
webpack 5.74.0 compiled with 65 warnings in 5064 ms
Terminate batch job (Y/N)? y
PS C:\Users\vikki\source\repos\MeteorTrading> git push heroku master
error: src refspec master does not match any
error: failed to push some refs to 'heroku'
PS C:\Users\vikki\source\repos\MeteorTrading>
```

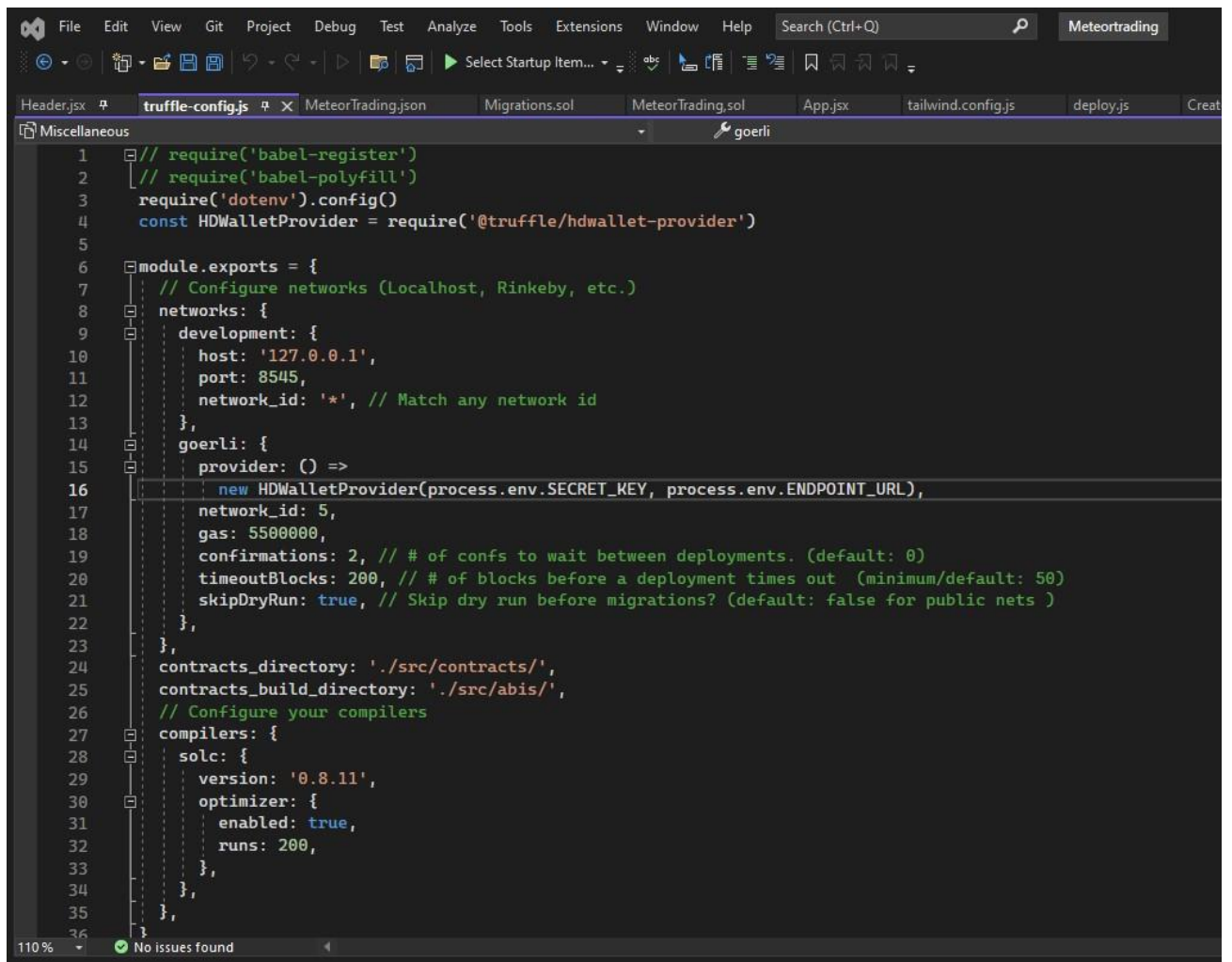
The right sidebar shows the **Solution Explorer** with a folder view of the project structure, including files like `Address.json`, `Context.json`, `ERC165.json`, `ERC721.json`, `ERC721Enumerable.json`, `ERC721Metadata.json`, `ERC721Receiver.json`, `Migrations.json`, `Ownable.json`, `Strings.json`, `assets`, `Header.png`, `components`, `Alert.jsx`, and `Properties`.

```
File Edit View Git Project Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Meteortrading
Header.jsx truffle-config.js MeteorTrading.json Migrations.sol App.jsx tailwind.config.js deploy.js CreateNFT.jsx Footer.jsx Transactions.jsx ERC721Enumerable.sol
Miscellaneous
1 import { useEffect } from 'react'
2 import { getAllNFTs, isWalletConnected } from './Blockchain.Services'
3 import Alert from './components/Alert'
4 import Artworks from './components/Artworks'
5 import CreateNFT from './components/CreateNFT'
6 import Footer from './components/Footer'
7 import Header from './components/Header'
8 import Hero from './components/Hero'
9 import Loading from './components/Loading'
10 import ShowNFT from './components/ShowNFT'
11 import Transactions from './components/Transactions'
12 import UpdateNFT from './components/UpdateNFT'
13
14 const App = () => {
15   useEffect(async () => {
16     await isWalletConnected()
17     await getAllNFTs()
18   }, [])
19
20   return (
21     <div className="min-h-screen">
22       <div className="gradient-bg-hero">
23         <Header />
24         <Hero />
25       </div>
26       <Artworks />
27       <Transactions />
28       <CreateNFT />
29       <ShowNFT />
30       <UpdateNFT />
31       <Footer />
32       <Alert />
33       <Loading />
34     </div>
35   )
36 }
110% No issues found Ln: 35 Ch: 4 SPC C
Developer PowerShell
+ Developer PowerShell
```

```
1 async function main() {
2   const [deployer, feeAccount] = await ethers.getSigners();
3
4   console.log("Deploying contracts with the account:", deployer.address);
5   console.log("Assigning Fee Account with:", feeAccount.address);
6   console.log("Deployer balance:", (await deployer.getBalance()).toString());
7   console.log("FeeAccount balance:", (await feeAccount.getBalance()).toString());
8
9   // Get the ContractFactories and Signers here.
10  const Store = await ethers.getContractFactory("Store");
11
12  // deploy contracts
13  const store = await Store.deploy('Freshers', feeAccount, 10);
14
15  // Save copies of each contracts abi and address to the frontend.
16  saveFrontendUtils(store, "Store");
17 }
18
19 function saveFrontendUtils(contract, name) {
20   const fs = require("fs");
21   const utilsDir = __dirname + "/src/utils";
22
23   if (!fs.existsSync(utilsDir)) {
24     fs.mkdirSync(utilsDir);
25   }
26
27   fs.writeFileSync(
28     utilsDir + `/${name}-address.json`,
29     JSON.stringify({ address: contract.address }, undefined, 2)
30   );
31
32   const contractArtifact = artifacts.readArtifactSync(name);
33
34   fs.writeFileSync(
35     utilsDir + `/${name}.json`,
36     JSON.stringify(contractArtifact, null, 2)
37   )
110% No issues found Ln: 20 Ch: 28 SPC CRL
```

```
Header.jsx  truffle-config.js  MeteorTrading.json  Migrations.sol  MeteorTrading.sol  App.jsx  tailwind.config.js  deploy.js  CreateNFT.jsx  Footer.jsx  Transactions.jsx  ERC721Enumerable.sol  X
1 // SPDX-License-Identifier: MIT
2 // OpenZeppelin Contracts v4.4.1 (token/ERC721/extensions/ERC721Enumerable.sol)
3
4 pragma solidity ^0.8.0;
5
6 import './ERC721.sol';
7 import './IERC721Enumerable.sol';
8
9 /**
10  * @dev This implements an optional extension of {ERC721} defined in the EIP that adds
11  * enumerability of all the token ids in the contract as well as all token ids owned by each
12  * account.
13  */
14 abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
15     // Mapping from owner to list of owned token IDs
16     mapping(address => mapping(uint256 => uint256)) private _ownedTokens;
17
18     // Mapping from token ID to index of the owner tokens list
19     mapping(uint256 => uint256) private _ownedTokensIndex;
20
21     // Array with all token ids, used for enumeration
22     uint256[] private _allTokens;
23
24     // Mapping from token id to position in the allTokens array
25     mapping(uint256 => uint256) private _allTokensIndex;
26
27     /**
28      * @dev See {IERC165-supportsInterface}.
29      */
30     function supportsInterface(bytes4 interfaceId)
31     public
32     view
33     virtual
34     override(IERC165, ERC721)
35     returns (bool)
36     {
37         return
```

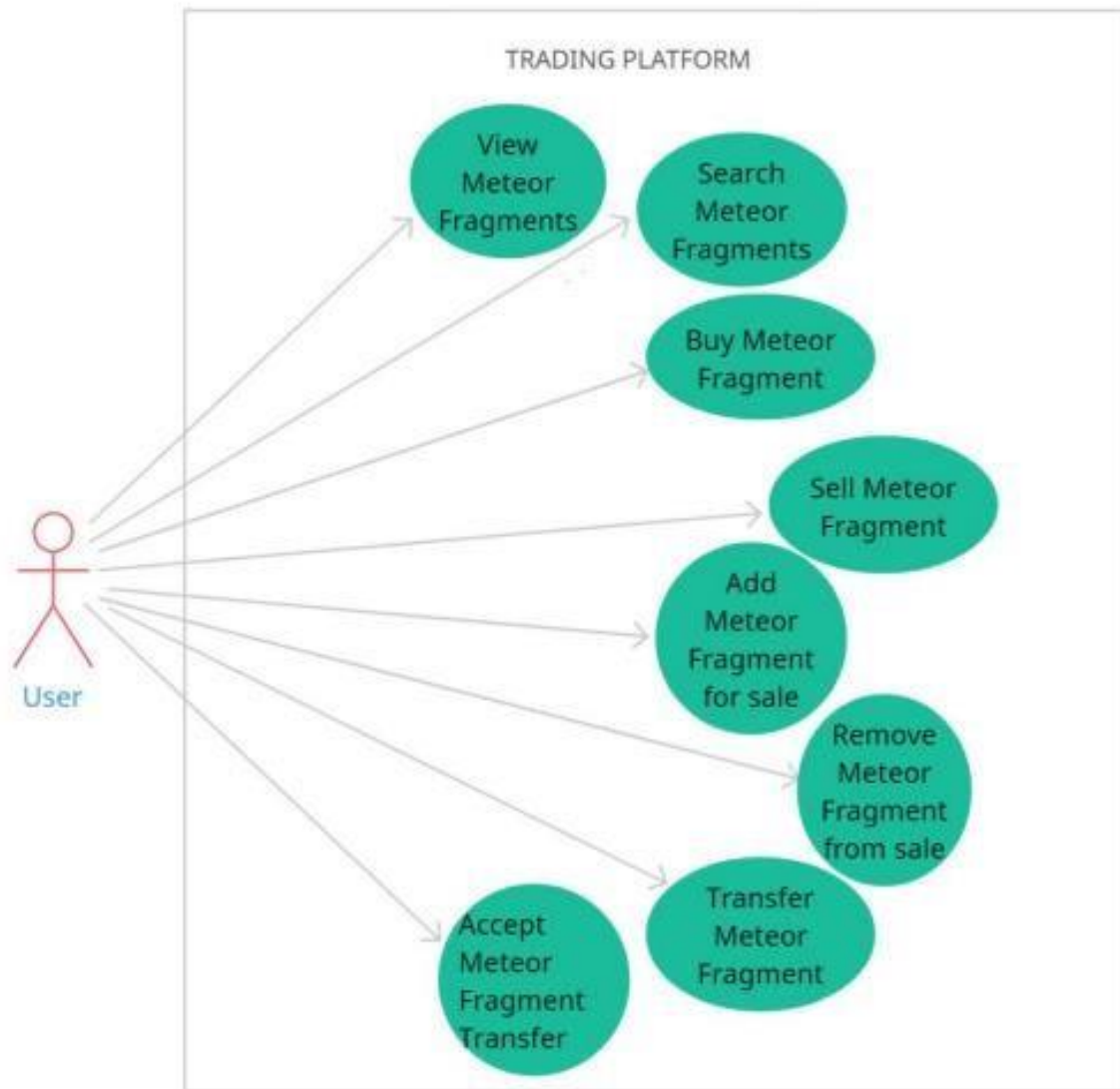




```
1 // require('babel-register')
2 // require('babel-polyfill')
3 require('dotenv').config()
4 const HDWalletProvider = require('@truffle/hdwallet-provider')
5
6 module.exports = {
7   // Configure networks (Localhost, Rinkeby, etc.)
8   networks: {
9     development: {
10       host: '127.0.0.1',
11       port: 8545,
12       network_id: '*', // Match any network id
13     },
14     goerli: {
15       provider: () =>
16         new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL,
17           network_id: 5,
18           gas: 5500000,
19           confirmations: 2, // # of confs to wait between deployments. (default: 0)
20           timeoutBlocks: 200, // # of blocks before a deployment times out (minimum/default: 50)
21           skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
22         ),
23     },
24     contracts_directory: './src/contracts/',
25     contracts_build_directory: './src/abis/',
26     // Configure your compilers
27     compilers: {
28       solc: {
29         version: '0.8.11',
30         optimizer: {
31           enabled: true,
32           runs: 200,
33         },
34       },
35     },
36   }
37 }
```

110 % No issues found

Contract Diagram:





## **Merits and Demerits:**

### **Merits:**

- we can save or preserve the space history by converting them into NFTs
- It is easier to identify the owner of the meteoroid and maintain the locations of the meteoroid, by using their account description or in their information page.

### **Demerits:**

- Some people can replicate the picture of the meteoroid and demand false ownership
- Fake meteoroids will be more, we need to carefully examine the meteoroid before minting it.

### **References:**

Professors references

<https://www.npmjs.com/package/react-app-rewired>

<https://moralis.io/how-to-build-decentralized-apps-dapps-quickly-and-easily/>

<https://www.dappuniversity.com/articles/the-ultimate-ethereum-dapp-tutorial>