CSE 526 BLOCKCHAIN APPLICATION DEVELOPMENT PROJECT PHASE-II

‘Fallen Star’

NFT trading Platform

Jahnavi Rudraraju

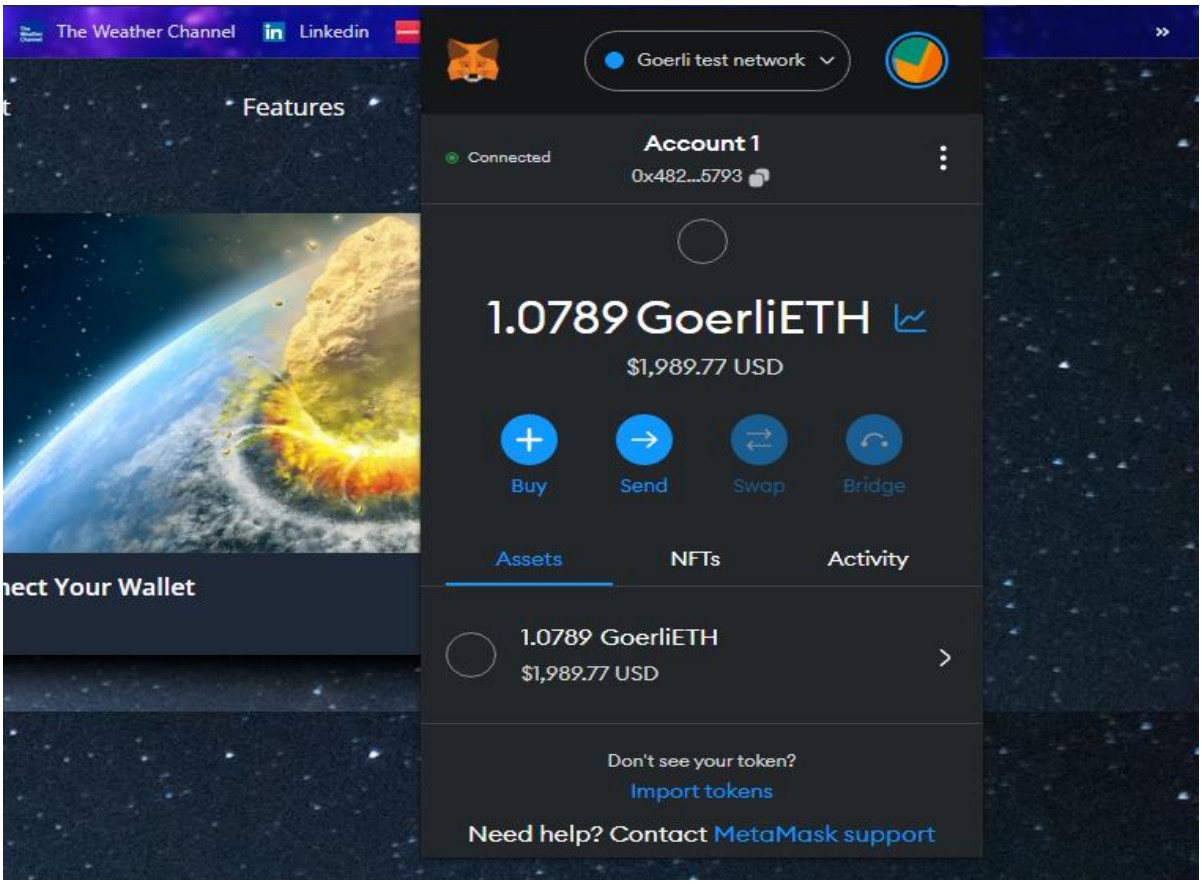
50464467

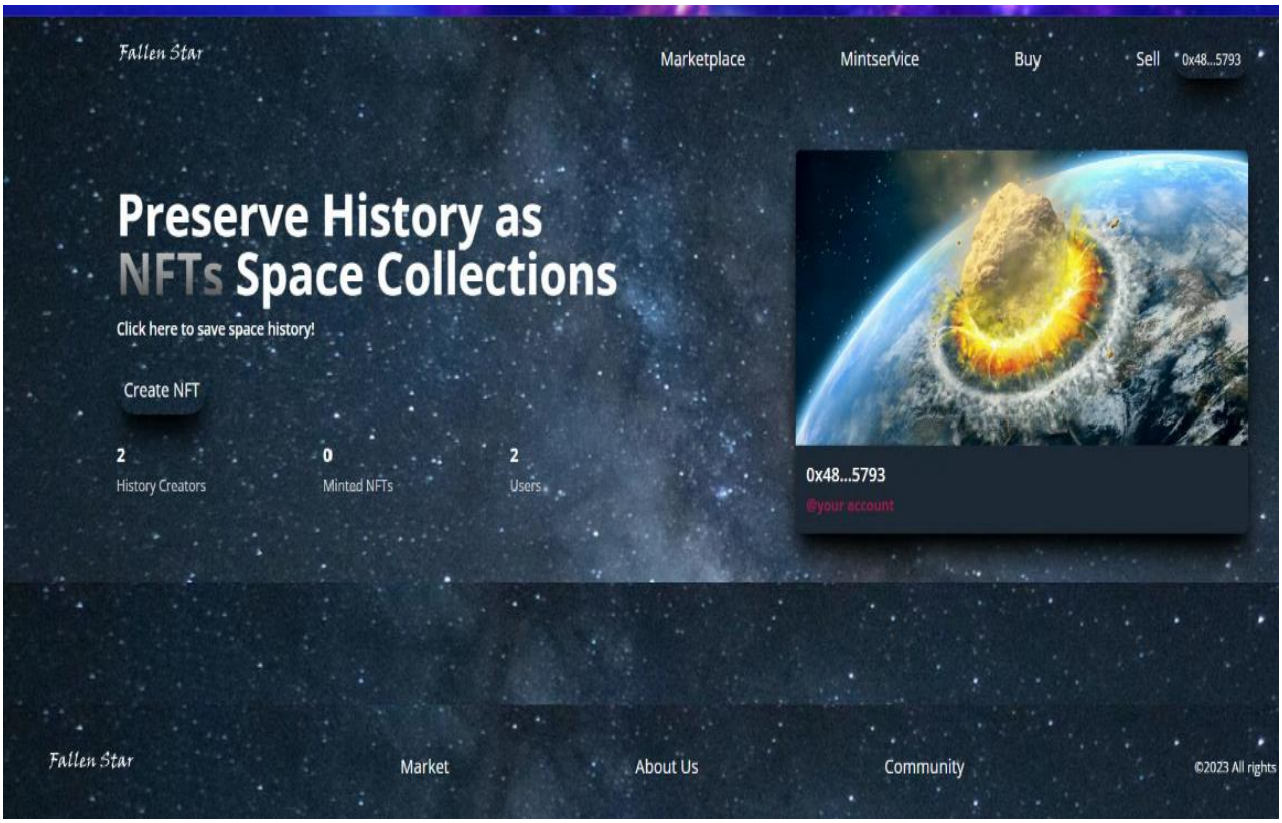
jahnavir@buffalo.edu

Implementation:

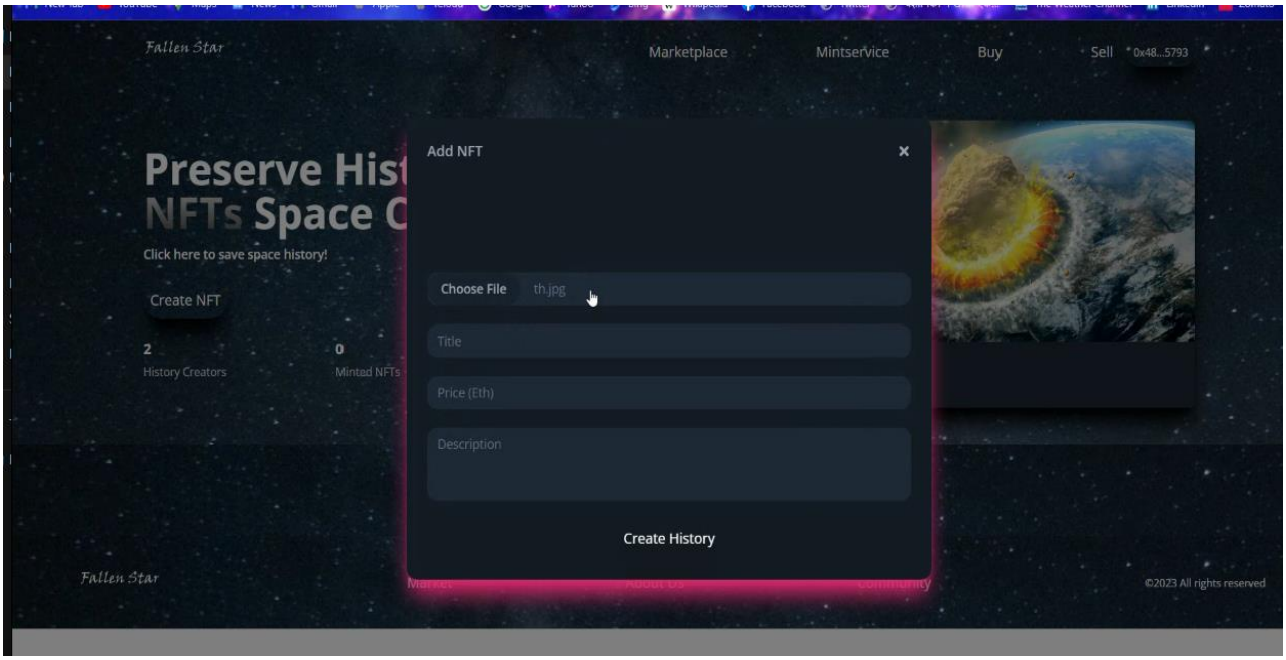
The trading platform is built using react application, the activities running in the web site are designed using truffle, tailwind, and some solidity smart contracts for transactions, connecting accounts etc.

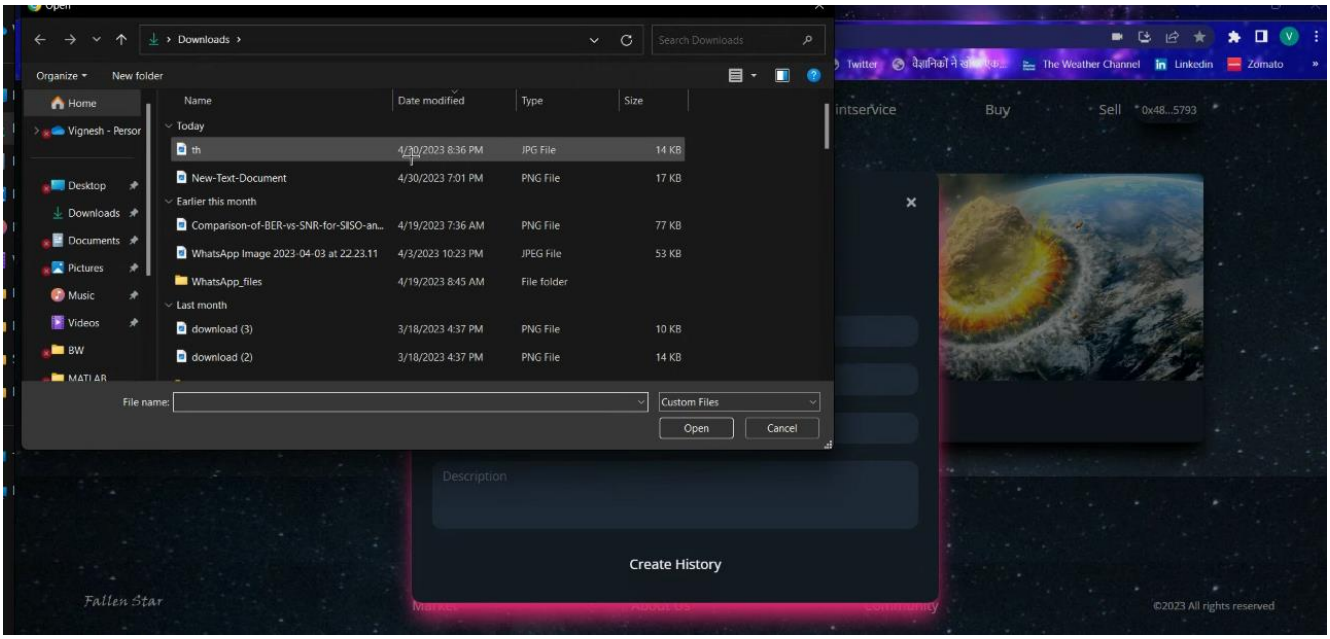
First we need to connect our metamask wallet to the web page, here we have two ways the site works, if metamask is unavailable in your system then it shows 'Please install metamask'. If you already have metamask it brings a prompt where you can connect to metamask account.



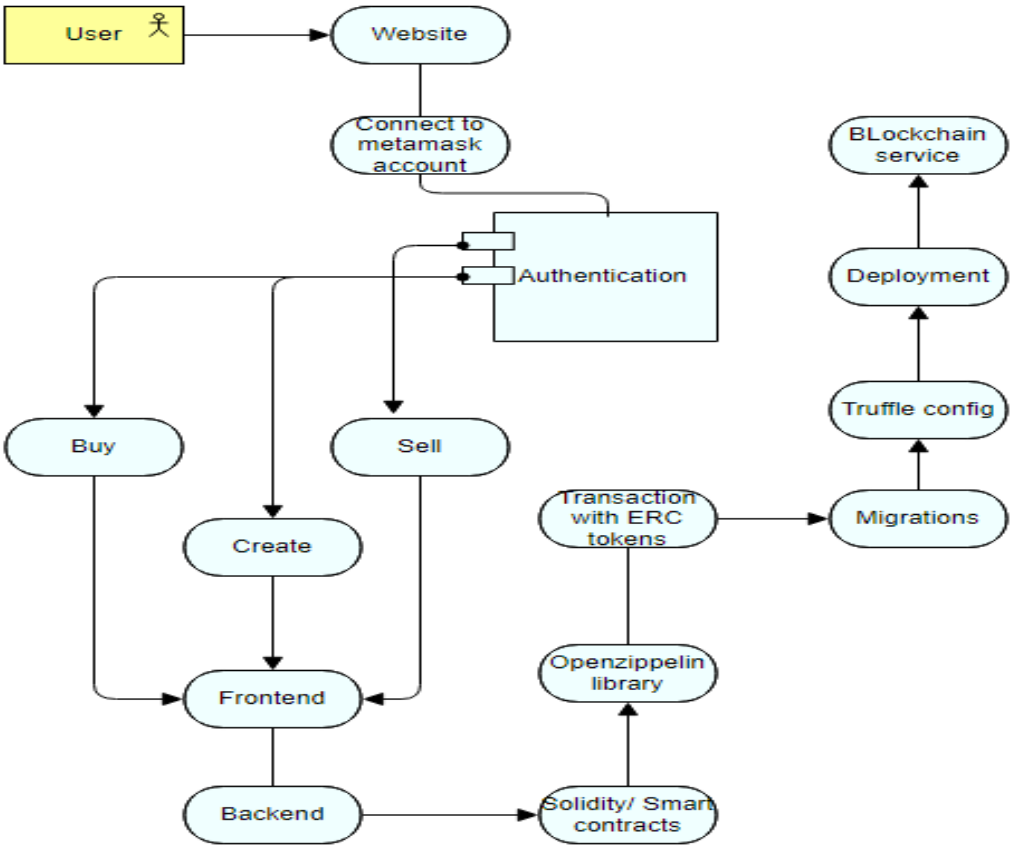


Now our account is connected to the web site, we can start creating NFT's, buying and selling in marketplace. Below is my activity diagram from phase 1





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.

```jsx
import { useEffect } from 'react'
import { getAllNFTs, isWallectConnected } from './Blockchain.Services'
import Alert from './components/Alert'
import Artworks from './components/Artworks'
import CreateNFT from './components/CreateNFT'
import Footer from './components/Footer'
import Header from './components/Header'
import Hero from './components/Hero'
import Loading from './components/Loading'
import ShowNFT from './components/ShowNFT'
import Transactions from './components/Transactions'
import UpdateNFT from './components/UpdateNFT'

const App = () => {
  useEffect(async () => {
    await isWallectConnected()
    await getAllNFTs()
  }, [])

  return (
    <div className="min-h-screen">
      <div className="gradient-bg-hero">
        <Header />
        <Hero />
      </div>
      <Artworks />
      <Transactions />
      <CreateNFT />
      <ShowNFT />
      <UpdateNFT />
      <Footer />
      <Alert />
      <Loading />
    </div>
  )
}
```

```js
// require('babel-register')
// require('babel-polyfill')
require('dotenv').config()
const HDWalletProvider = require('@truffle/hdwallet-provider')

module.exports = {
  // Configure networks (Localhost, Rinkeby, etc.)
  networks: {
    development: {
      host: '127.0.0.1',
      port: 8545,
      network_id: '*', // Match any network id
    },
    goerli: {
      provider: () =>
        new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL),
      network_id: 5,
      gas: 5500000,
      confirmations: 2, // # of confs to wait between deployments. (default: 0)
      timeoutBlocks: 200, // # of blocks before a deployment times out  (minimum/default: 50)
      skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
    },
  },
  contracts_directory: './src/contracts/',
  contracts_build_directory: './src/abis/',
  // Configure your compilers
  compilers: {
    solc: {
      version: '0.8.11',
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
}
```

```javascript
async function main() {
  const [deployer, feeAccount] = await ethers.getSigners();

  console.log("Deploying contracts with the account:", deployer.address);
  console.log("Assigning Fee Account with:", feeAccount.address);
  console.log("Deployer balance:", (await deployer.getBalance()).toString());
  console.log("FeeAccount balance:", (await feeAccount.getBalance()).toString());

  // Get the ContractFactories and Signers here.
  const Store = await ethers.getContractFactory("Store");

  // deploy contracts
  const store = await Store.deploy('Freshers', feeAccount, 10);

  // Save copies of each contracts abi and address to the frontend.
  saveFrontendUtils(store , "Store");
}

function saveFrontendUtils(contract, name) {
  const fs = require("fs");
  const utilsDir = __dirname + "./src/utils";

  if (!fs.existsSync(utilsDir)) {
    fs.mkdirSync(utilsDir);
  }

  fs.writeFileSync(
    utilsDir + `/${name}-address.json`,
    JSON.stringify({ address: contract.address }, undefined, 2)
  );

  const contractArtifact = artifacts.readArtifactSync(name);

  fs.writeFileSync(
    utilsDir + `/${name}.json`,
    JSON.stringify(contractArtifact, null, 2)
```

```
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (token/ERC721/extensions/ERC721Enumerable.sol)

pragma solidity ^0.8.0;

import "./ERC721.sol";
import "./IERC721Enumerable.sol";

/**
 * @dev This implements an optional extension of {ERC721} defined in the EIP that adds
 * enumerability of all the token ids in the contract as well as all token ids owned by each
 * account.
 */
abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
    // Mapping from owner to list of owned token IDs
    mapping(address => mapping(uint256 => uint256)) private _ownedTokens;

    // Mapping from token ID to index of the owner tokens list
    mapping(uint256 => uint256) private _ownedTokensIndex;

    // Array with all token ids, used for enumeration
    uint256[] private _allTokens;

    // Mapping from token id to position in the allTokens array
    mapping(uint256 => uint256) private _allTokensIndex;

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId)
        public
        view
        virtual
        override(IERC165, ERC721)
        returns (bool)
    {
        return
```

File Edit View Git Project Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q) Meteortrading

Select Startup Item...

Header.jsx | truffle-config.js | MeteorTrading.json | Migrations.sol | MeteorTrading.sol | App.jsx | tailwind.config.js | deploy.js | Creat

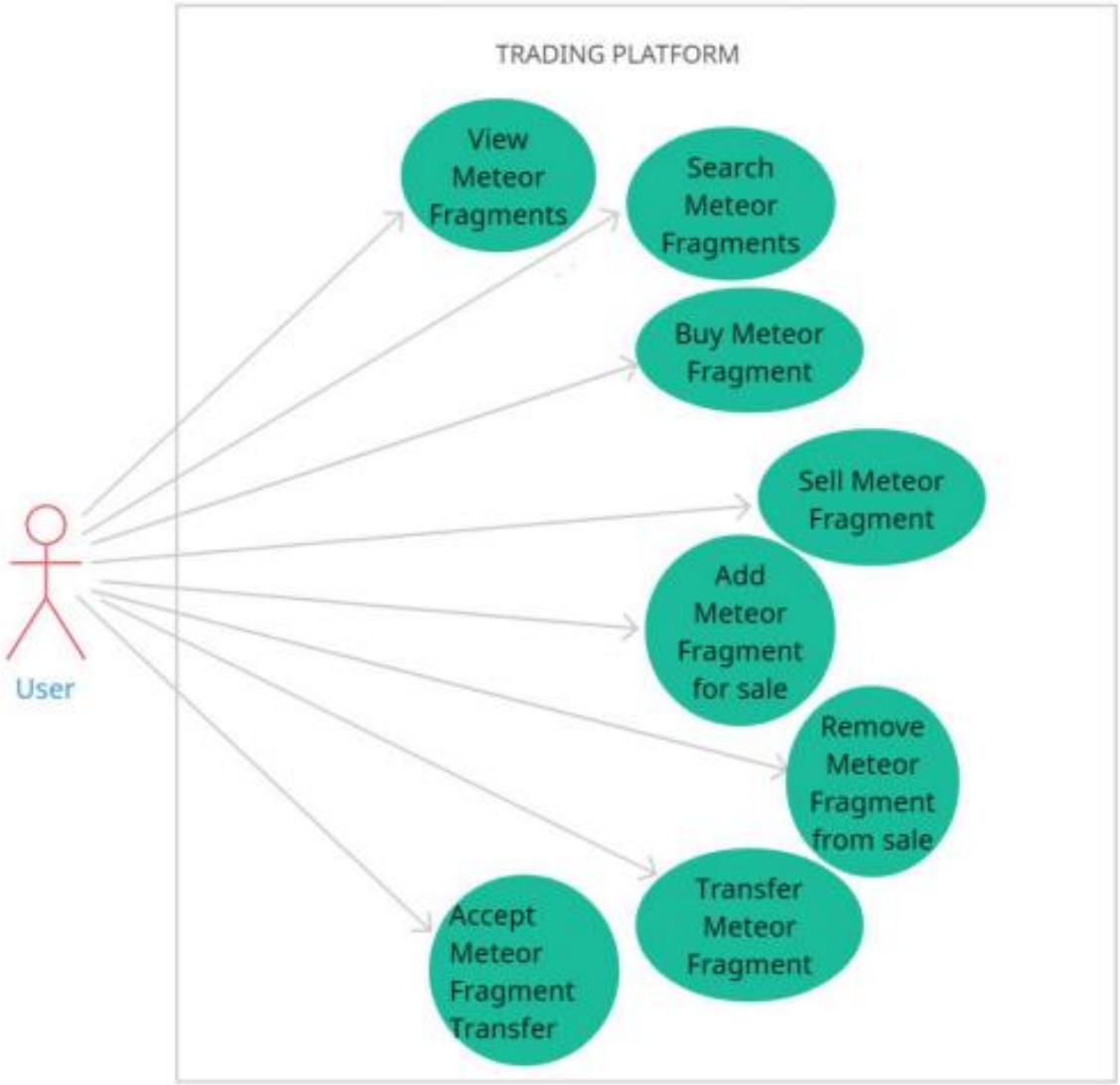
Miscellaneous | goerli

```
// require('babel-register')
// require('babel-polyfill')
require('dotenv').config()
const HDWalletProvider = require('@truffle/hdwallet-provider')

module.exports = {
  // Configure networks (Localhost, Rinkeby, etc.)
  networks: {
    development: {
      host: '127.0.0.1',
      port: 8545,
      network_id: '*', // Match any network id
    },
    goerli: {
      provider: () =>
        new HDWalletProvider(process.env.SECRET_KEY, process.env.ENDPOINT_URL),
      network_id: 5,
      gas: 5500000,
      confirmations: 2, // # of confs to wait between deployments. (default: 0)
      timeoutBlocks: 200, // # of blocks before a deployment times out  (minimum/default: 50)
      skipDryRun: true, // Skip dry run before migrations? (default: false for public nets )
    },
  },
  contracts_directory: './src/contracts/',
  contracts_build_directory: './src/abis/',
  // Configure your compilers
  compilers: {
    solc: {
      version: '0.8.11',
      optimizer: {
        enabled: true,
        runs: 200,
      },
    },
  },
}
```

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.