This is an instantiation of a proof-of-concept to illustrate an NFT mechanism for which the tokens have multiple owners, that have different share percentages over the token. The code consists of two parts:

- A javascript frontend interacting with the ethereum blockchain using the web3 API.
- Implementation of contract logic in Solidity.

In short, the solidity code resides in *Explorer.sol* and is represented by two contracts: *ExplorerToken* and *ExplorerSupply*. *ExplorerToken* (1) represents an "internal" contract and is responsible for keeping track of the tokens themselves and their properties. *ExplorerSupply* (2) is the contract that is meant to interact with the "exterior" and mainly wraps the creation method for the tokens from ExplorerToken.

Besides the above mentioned contracts, a significant part of functionality is given by the (openzeppelin) contracts that are being inherited:

- *Ownable* provides a mechanism by which the contracts have an owner that has exclusive access to some specific functionality, in our case being the creation of new tokens.
- *Counters* library exposes a restrictive and (more) secure implementation of a counter, used for keeping track of the tokens created.
- *ERC721Enumerable* is an extension of ERC721 standard (NFTs) that originally permitter the creation of non fungible tokens, and provided means of transferring them across (authorised) blockchain addresses. The "enumerable" extension permits indexing tokens, feature used for the frontend module.
- *ERC1155* is a contract that implements the standard with the same name, which allows to have multiple token types with one / more units for each. In this particular case, it is used to keep track of each token's shares distribution among multiple owners.
- *IERC721Receiver* is an interface, that, according to openzeppelin documentation, it is neccessary for implementing a standardised way of receiving ERC721 tokens.

The functionality itself is provided by combining the *ERC721* and *ERC1155* functionality: when a new location token is created, it is first "minted" in the *ExplorerToken*, then it is also "minted" in the ERC1155 - this poses the distadvantage of spending more gas per token, and also the need for synchronising the two contract states arises.

The javascript component uses web3 API to interact with the blockchain and with the contracts' method themselves:

- *web3.eth.Contract()* to "cast" a given address to the contract type that is used (in our case, the *ExplorerSupply* and *ExplorerToken* instances)
- *ExplorerToken.totalSupply().call()* to get the total number of NTFs in existence
- *ExplorerToken.getLocation().call()* to get a specific NFT based on an index

- *ExplorerSupply.createToken().estimateGas()* to estimate the gas needed to execute the following operation

```

contract ExplorerToken is ERC721Enumerable, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    struct Location {
        int24 latitude;
        int24 longitude;
        string name;
    }

    mapping(int24 => mapping(int24 => bool)) _coords;

    mapping(uint256 => Location) _places;

    constructor() ERC721("Explorer", "EXP") {}

    function createToken(
        int24 latitude,
        int24 longitude,
        string memory name
    ) external onlyOwner returns (uint256) {
        require(
            _coords[latitude][longitude] == false,
            "This location has already been registered!"
        );

        _tokenIds.increment();
        uint256 locationId = _tokenIds.current();
        _safeMint(this.owner(), locationId);
        _places[locationId] = Location(latitude, longitude, name);
        _coords[latitude][longitude] = true;
        return locationId;
    }

    function getLocation(uint256 tokenId)
        public
        view
        returns (Location memory)
    {
        _exists(tokenId);
        return _places[tokenId];
    }
}
```

Listing 1: ExplorerToken

```

contract ExplorerSupply is ERC1155, IERC721Receiver, Ownable {
    address NFT;

    constructor() ERC1155("") {
        ExplorerToken _NFT = new ExplorerToken();
        NFT = address(_NFT);
    }

    function ownedToken() external view returns (address _NFT) {
        return NFT;
    }

    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    ) external override returns (bytes4) {
        _mint(this.owner(), tokenId, 128, data);
        return IERC721Receiver.onERC721Received.selector;
    }

    function createToken(
        int24 latitude,
        int24 longitude,
        string memory name
    ) public onlyOwner returns (uint256) {
        ExplorerToken(NFT).createToken(latitude, longitude, name);
        return 0;
    }
}
```

Listing 2: ExplorerSupply