ERC1155CONTRACT

SELLCONTRACT

MAIN CONTRACT DESIGN

SERCAN ÇELENK

30 July 2021

CONTENTS

1 Functions

1.1 Batch Transfer

Batch transactions and transfers; it's about saving time and removing bugs rather than saving gas. Batch tools are useful for any operation that requires tokens to be sent to multiple addresses.

For example; hebys.io wants to distribute 1 percent of the token they hold as a loyalty reward to users who have kept hebys tokens in their wallet for 1 year. The easiest way to do this is with Batch Transfers.

In another scenario; Suppose the company is damaged due to hacking or flash loan attacks.

A short time ago, "belt.fi" was subjected to such an attack and a large amount of dollars was stolen in 4-5 seconds. Most people who provided liquidity in Belt.fi pools had also experienced a deposit meltdown of nearly 20 percent.

As a remedy, belt.fi released a token called r4Belt and airdropped this token in a short time.

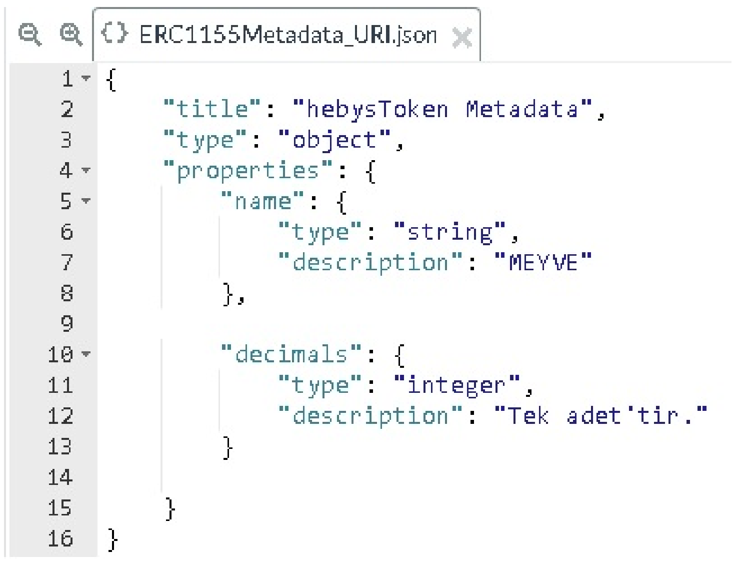
Shortly after the Bitfinex exchange was hacked, it released a coin called LEO and made an airdrob of it.

Such airdrops would not have been possible if the contract owner had not included Batch Transfer in the contract.

Batch Transfer is vital in some cases. The examples I gave are the best proof of this.

1.2 NFT Support

When a supply is given as only 1, the token is essentially a non fungible token(NFT). If we say it cannot be changed, the ERC standard that defines it is 721. Each private (single) token must have an NFT support and a metadataURL must be defined for each.

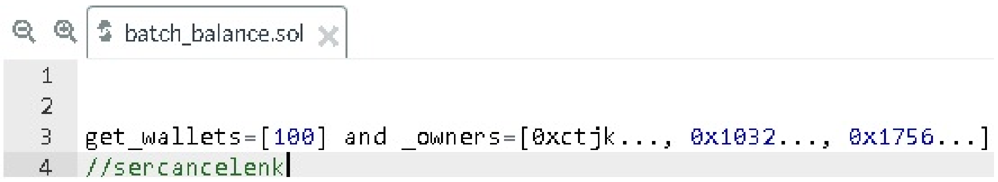

ERC1155Metadata_URI.json

```
{
    "title": "hebysToken Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description": "MEYVE"
        },

        "decimals": {
            "type": "integer",
            "description": "Tek adet'tir."
        }

    }
}
```

1.3 Batch Balance

Toplu Bakiye işlemi elimizdeki sorgu durumuna göre istediğimiz cüzdanları listelememize yarar. Listeleme sonrasında istediğimiz aksiyonu alabiliriz.

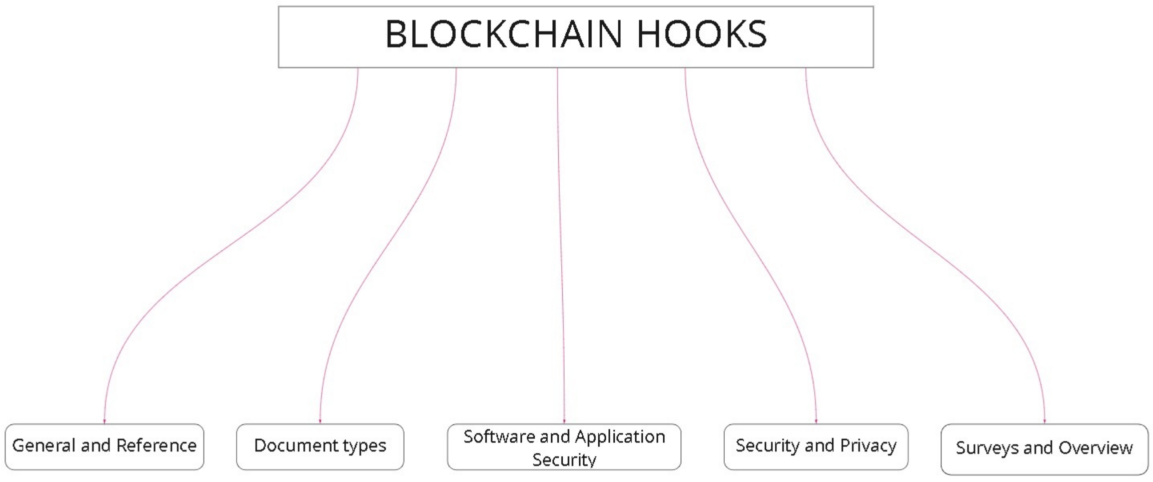
The Batch Balance operation allows us to list the wallets we want according to the query situation we have. After listing, we can take the action we want.


batch_balance.sol

```


get_wallets=[100] and _owners=[0xctjk..., 0x1032..., 0x1756...]
//sercancelenk
```

1.4 Safe Transfer Rule

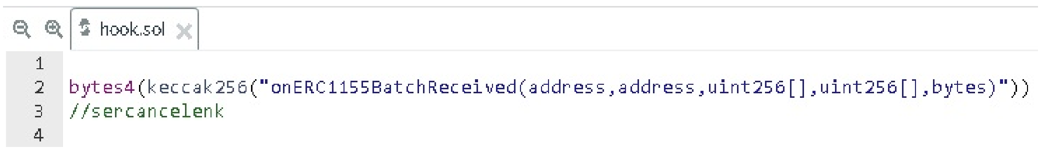
The most important rule of safe transfer is in the from function; The caller must have approved the contract in order to spend (send) the token in his hand. Return conditions when the caller makes a call; If the address balance is 0,
If the called value and the value in the wallet do not match,
If the wallet balance is insufficient, including pending transactions waiting to be posted,
And various other reasons,



1.5 Hooks

Prevent malicious transactions from entering the ledger; A runtime hook is essential for synchronizing and analyzing pending processes.(Runtime Hook)

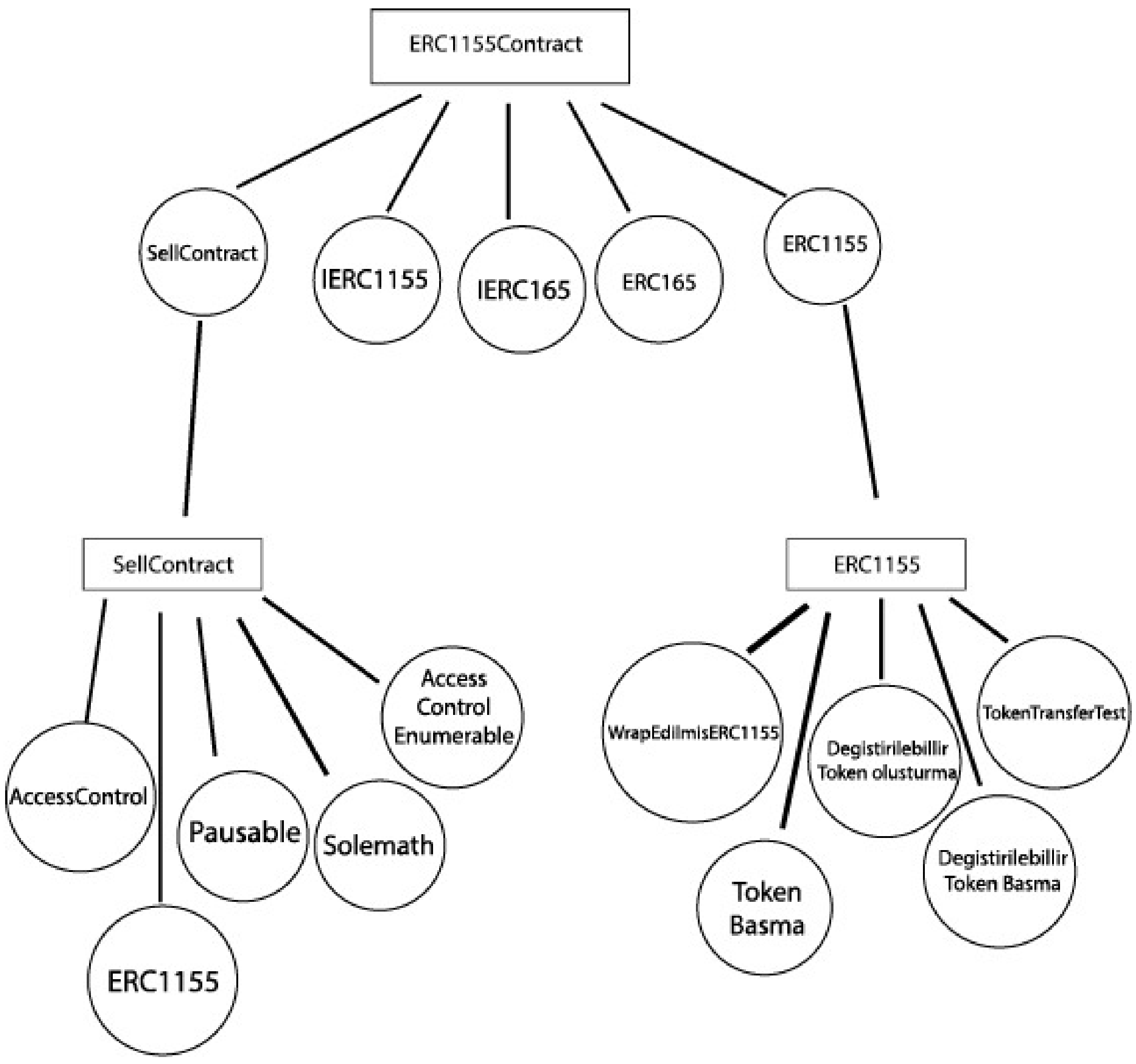
With this hook, we can prevent malicious transactions from taking place and prevent the signer, that is, the attacker, from accessing the contract.(flag)



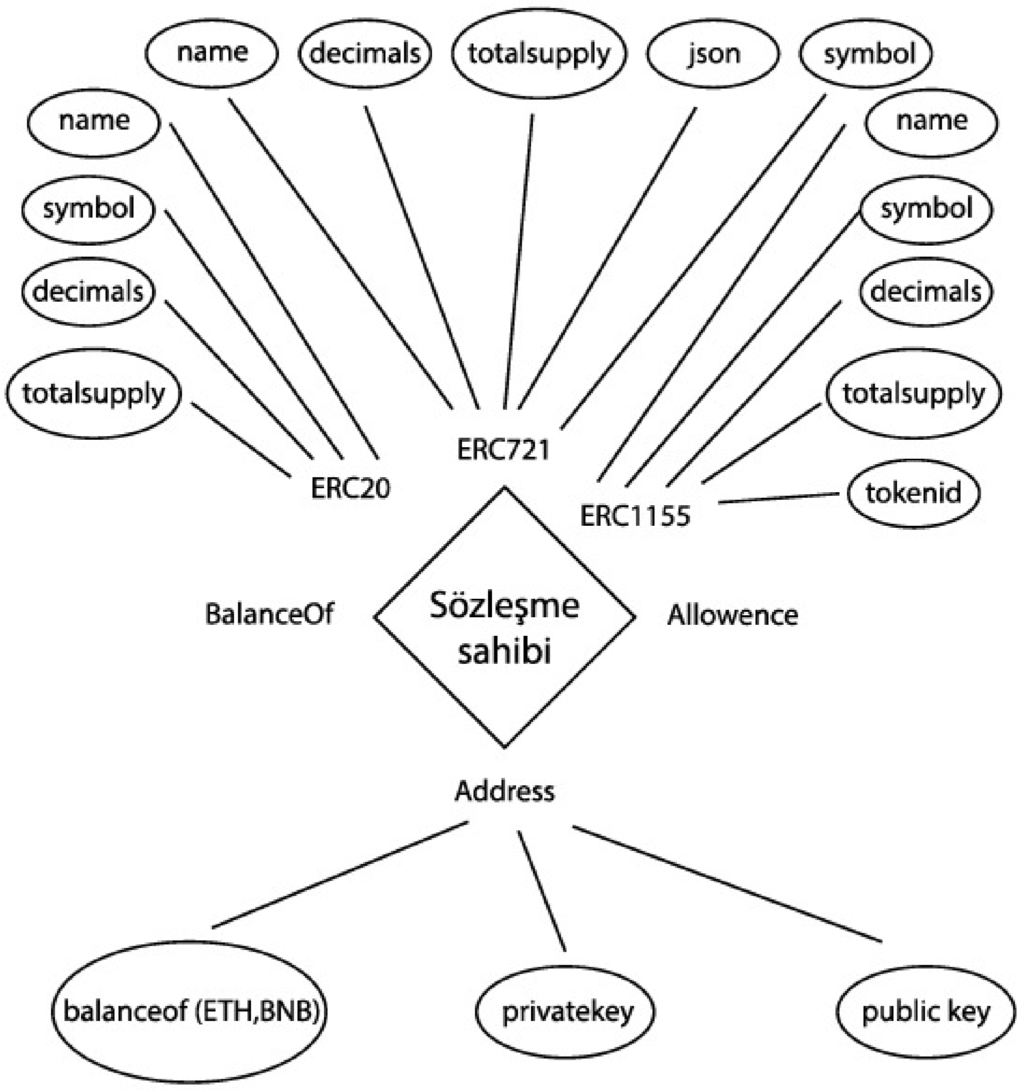
```

bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"))
//sercancelenk

```

2 Diagram Schemas

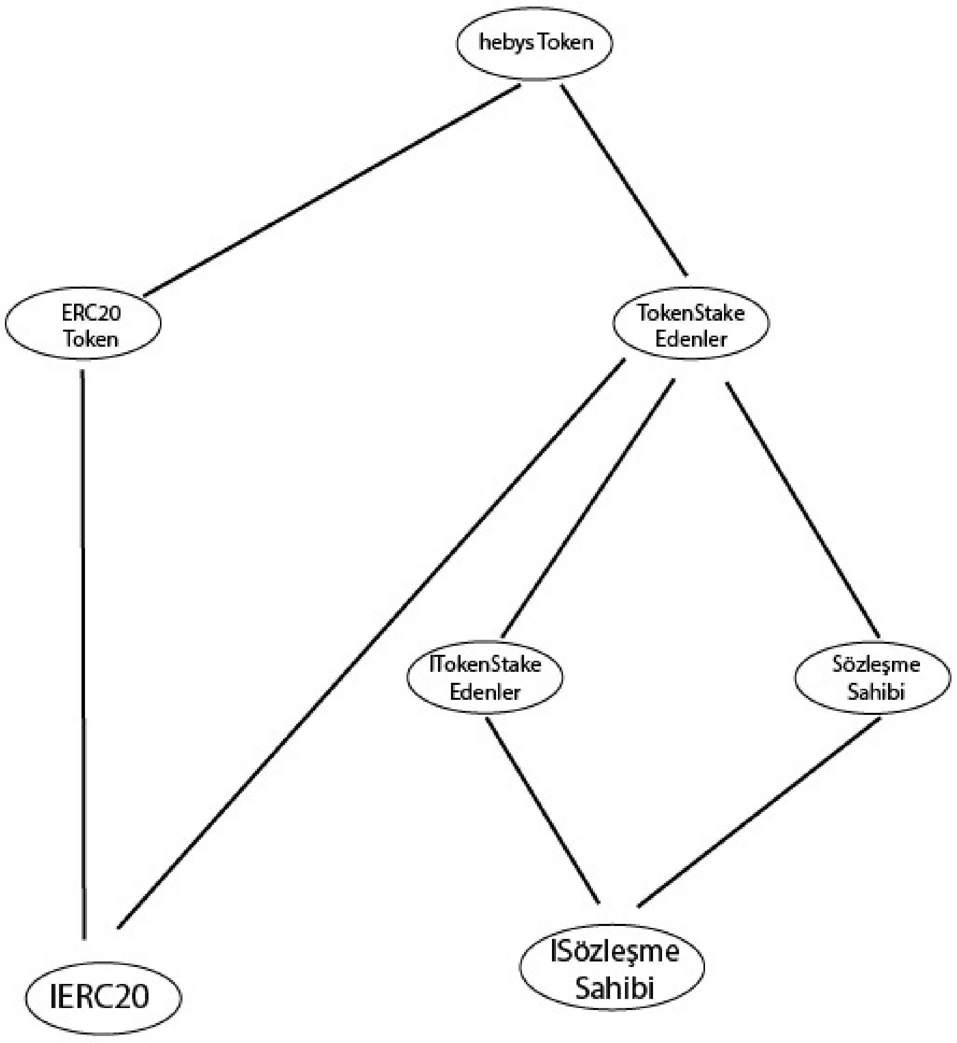
2.1 Contract Functions and Called Contract



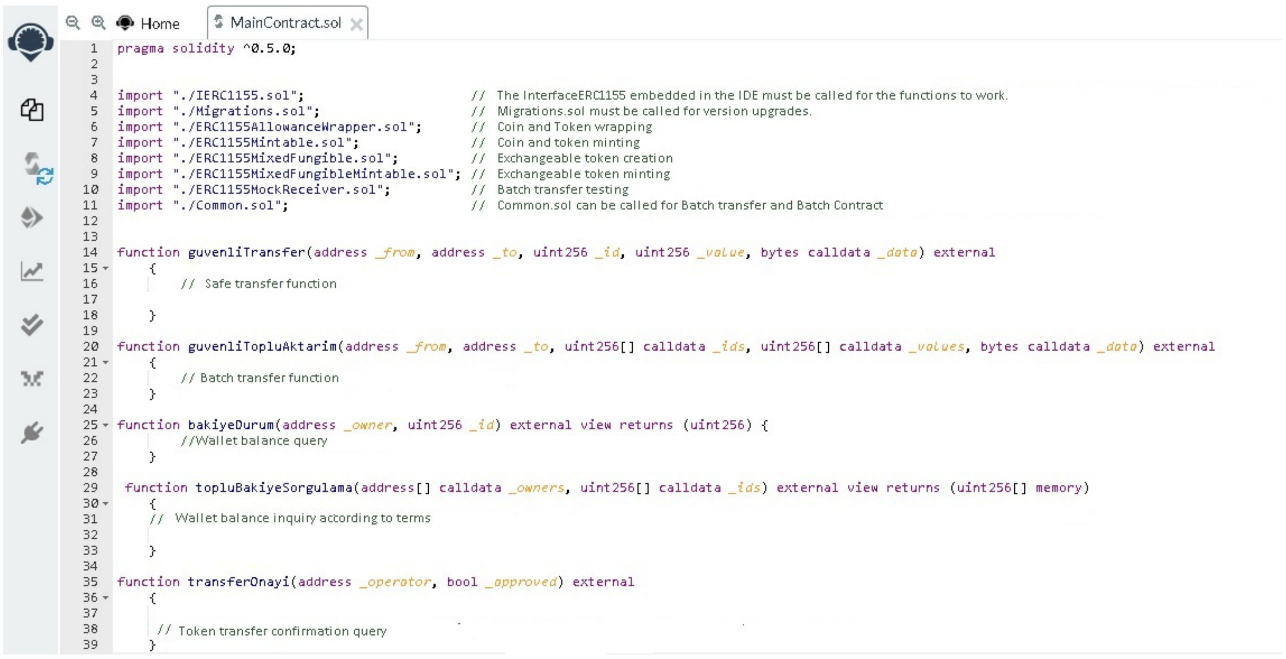
2.2 Flow Chart



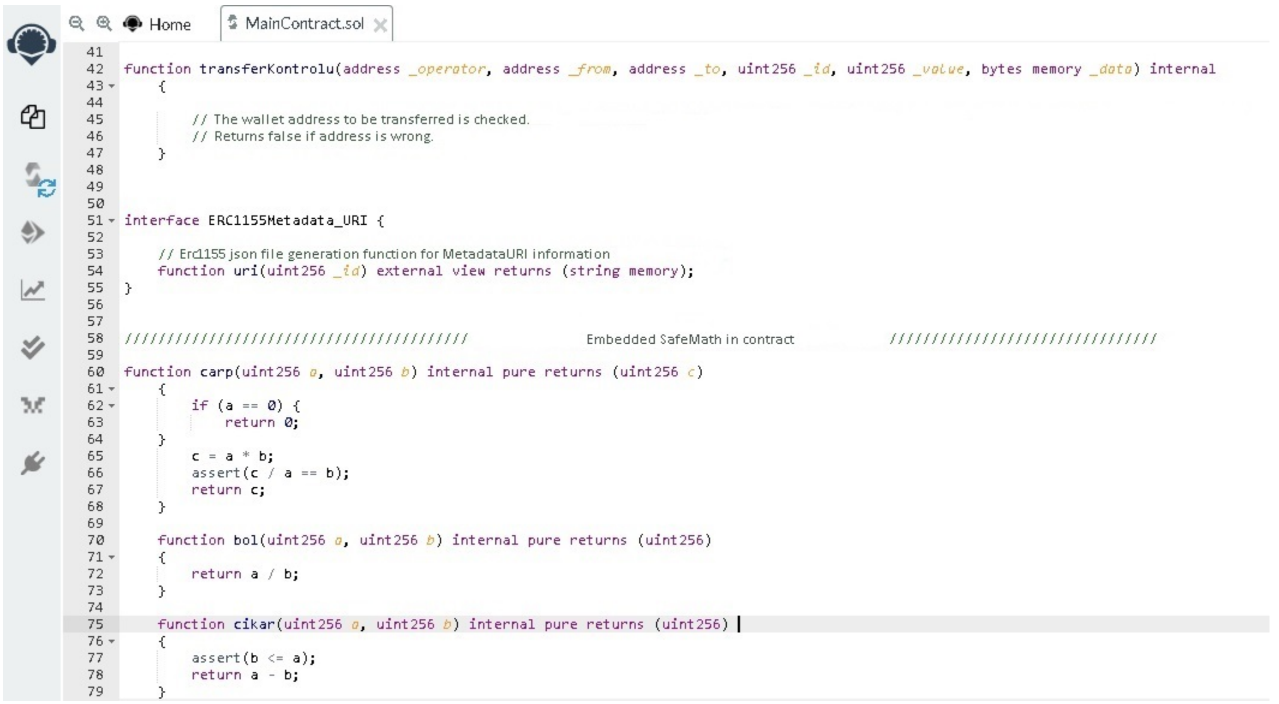
2.3 Staking Function Flow Chart



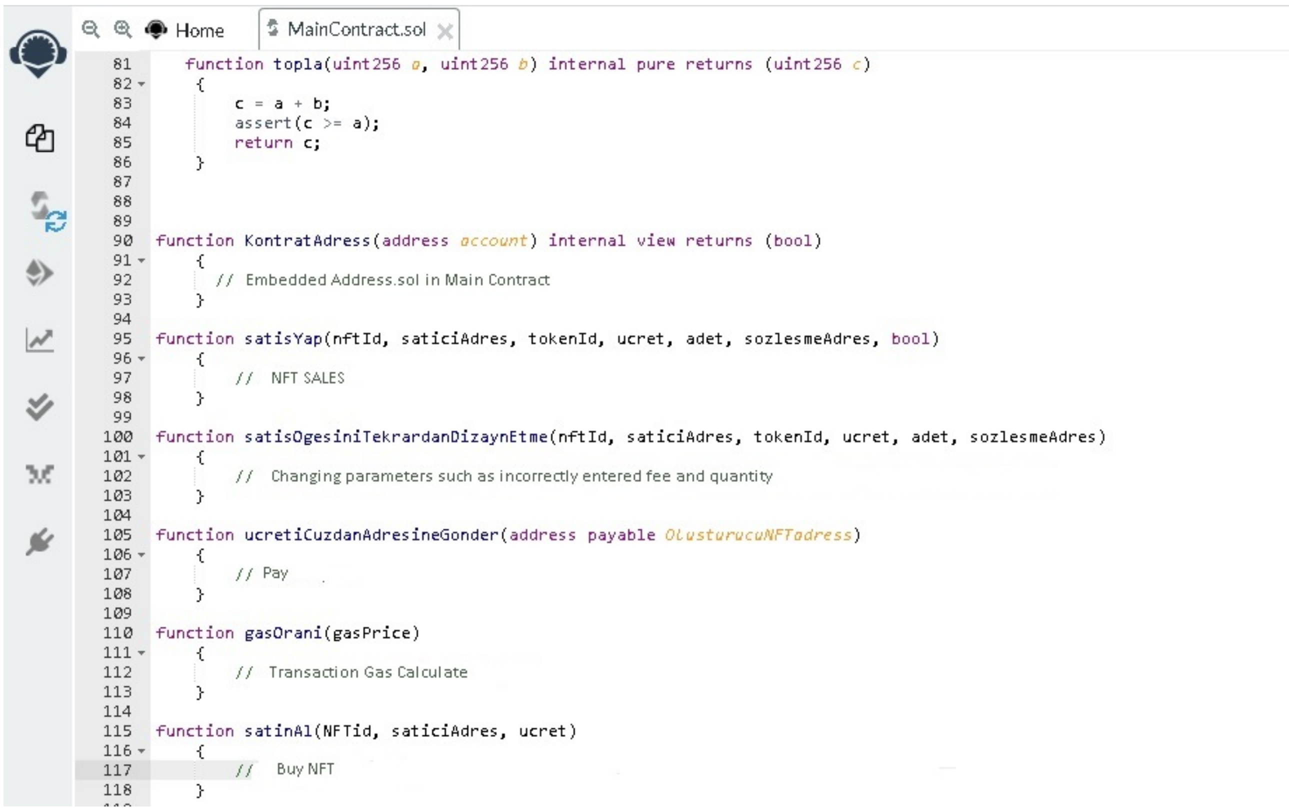
2.4 Main Contract Code Body



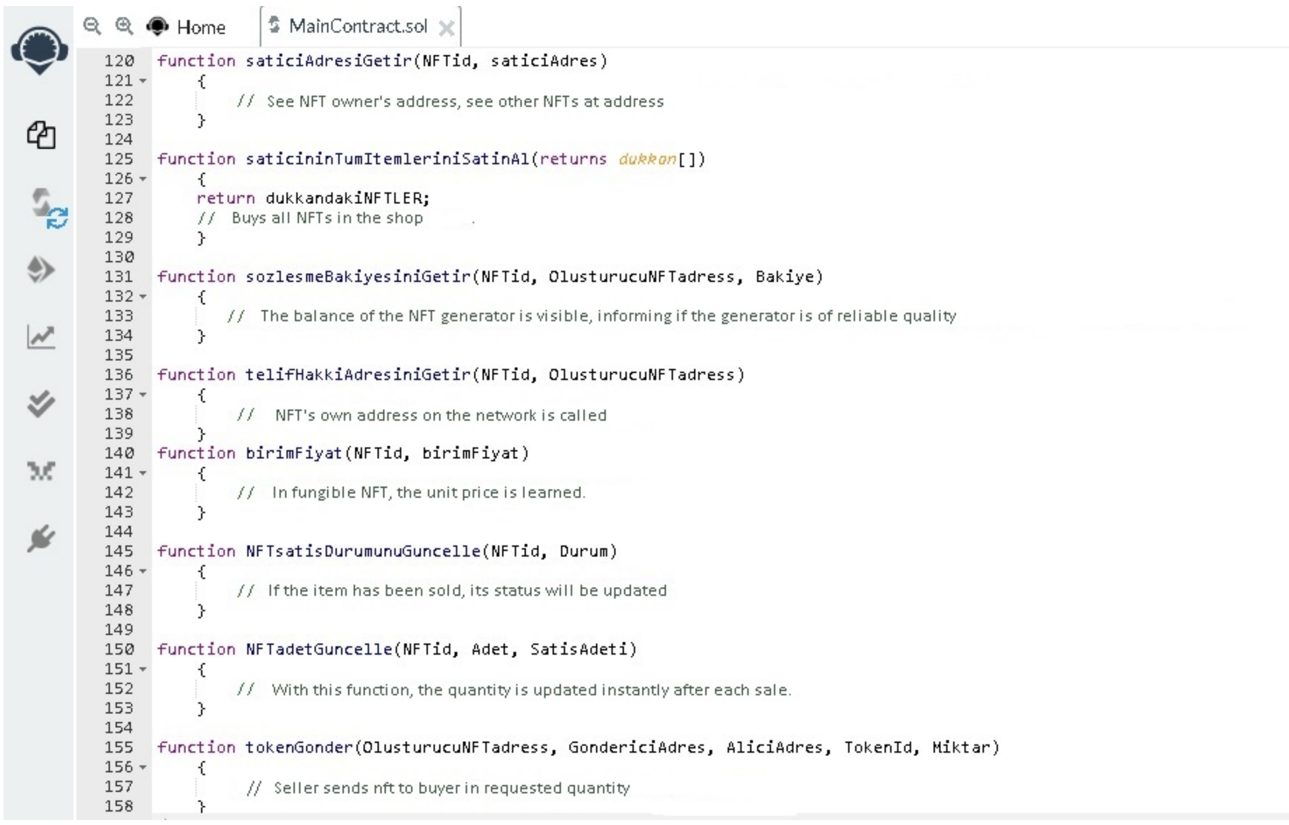
```
pragma solidity ^0.5.0;


import "./IERC1155.sol";                      //  The InterfaceERC1155 embedded in the IDE must be called for the functions to work.
import "./Migrations.sol";                    //  Migrations.sol must be called for version upgrades.
import "./ERC1155AllowanceWrapper.sol";       //  Coin and Token wrapping
import "./ERC1155Mintable.sol";               //  Coin and token minting
import "./ERC1155MixedFungible.sol";          //  Exchangeable token creation
import "./ERC1155MixedFungibleMintable.sol"; //  Exchangeable token minting
import "./ERC1155MockReceiver.sol";           //  Batch transfer testing
import "./Common.sol";                        //  Common.sol can be called for Batch transfer and Batch Contract


function guvenliTransfer(address _from, address _to, uint256 _id, uint256 _value, bytes calldata _data) external
    {
        //  Safe transfer function

    }

function guvenliTopluAktarim(address _from, address _to, uint256[] calldata _ids, uint256[] calldata _values, bytes calldata _data) external
    {
        // Batch transfer function
    }

function bakiyeDurum(address _owner, uint256 _id) external view returns (uint256) {
        //Wallet balance query
    }

 function topluBakiyeSorgulama(address[] calldata _owners, uint256[] calldata _ids) external view returns (uint256[] memory)
    {
    //  Wallet balance inquiry according to terms

    }

function transferOnayi(address _operator, bool _approved) external
    {

     // Token transfer confirmation query
    }
```



```

function transferKontrolu(address _operator, address _from, address _to, uint256 _id, uint256 _value, bytes memory _data) internal
    {

        // The wallet address to be transferred is checked.
        // Returns false if address is wrong.
    }



interface ERC1155Metadata_URI {

    // Erc1155 json file generation function for MetadataURI information
    function uri(uint256 _id) external view returns (string memory);
}


/////////////////////////////////////////                Embedded SafeMath in contract            ////////////////////////////////

function carp(uint256 a, uint256 b) internal pure returns (uint256 c)
    {
        if (a == 0) {
            return 0;
    }
        c = a * b;
        assert(c / a == b);
        return c;
    }

    function bol(uint256 a, uint256 b) internal pure returns (uint256)
    {
        return a / b;
    }

    function cikar(uint256 a, uint256 b) internal pure returns (uint256)
    {
        assert(b <= a);
        return a - b;
    }
```



```
   function topla(uint256 a, uint256 b) internal pure returns (uint256 c)
    {
        c = a + b;
        assert(c >= a);
        return c;
    }



function KontratAdress(address account) internal view returns (bool)
    {
      //  Embedded Address.sol in Main Contract
    }

function satisYap(nftId, saticiAdres, tokenId, ucret, adet, sozlesmeAdres, bool)
    {
        //   NFT SALES
    }

function satisOgesiniTekrardanDizaynEtme(nftId, saticiAdres, tokenId, ucret, adet, sozlesmeAdres)
    {
        //   Changing parameters such as incorrectly entered fee and quantity
    }

function ucretiCuzdanAdresineGonder(address payable OlusturucuNFTadress)
    {
        // Pay
    }

function gasOrani(gasPrice)
    {
        //  Transaction Gas Calculate
    }

function satinAl(NFTid, saticiAdres, ucret)
    {
        //    Buy NFT
    }
```



```
function saticiAdresiGetir(NFTid, saticiAdres)
    {
        //  See NFT owner's address, see other NFTs at address
    }

function saticininTumItemleriniSatinAl(returns dukkan[])
    {
    return dukkandakiNFTLER;
    //  Buys all NFTs in the shop
    }

function sozlesmeBakiyesiniGetir(NFTid, OlusturucuNFTadress, Bakiye)
    {
       //  The balance of the NFT generator is visible, informing if the generator is of reliable quality
    }

function telifHakkiAdresiniGetir(NFTid, OlusturucuNFTadress)
    {
        //   NFT's own address on the network is called
    }
function birimFiyat(NFTid, birimFiyat)
    {
        //  In fungible NFT, the unit price is learned.
    }

function NFTsatisDurumunuGuncelle(NFTid, Durum)
    {
        //  If the item has been sold, its status will be updated
    }

function NFTadetGuncelle(NFTid, Adet, SatisAdeti)
    {
        //  With this function, the quantity is updated instantly after each sale.
    }

function tokenGonder(OlusturucuNFTadress, GondericiAdres, AliciAdres, TokenId, Miktar)
    {
         //  Seller sends nft to buyer in requested quantity
    }
```