

Final Project

Deploy 'ERC20Token Contract':

The screenshot displays the Remix IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' sidebar is active. It shows the environment set to 'JavaScript VM', an account with 100 ether, a gas limit of 3,000,000, and a value of 0 wei. The contract selected is 'ERC20Token - browser/SupplyChain'. The '_NAME' field is set to 'ERC20Token', and the 'transact' button is highlighted. Below this, there are options to 'PUBLISH TO IPFS' or 'At Address', and a section for 'Transactions recorded' and 'Deployed Contracts'.

The main editor shows the Solidity code for 'SupplyChain.sol'. The code defines an 'ERC20Token' contract with a 'name' and 'balances' mapping, and a 'SupplyCoin' contract that inherits from 'ERC20Token' and adds an 'owners' array and a 'mint' function.

```
1 pragma solidity ^0.5.16;
2
3 //This is the standard of the token ERC20
4 contract ERC20Token {
5
6     string public name;
7     mapping(address => uint256) public balances;
8
9     constructor(string memory _name) public { name = _name; }
10    function mint(uint256 refund) public { balances[msg.sender] += refund; }
11
12 }
13
14 //This is the token of the supply chain
15 contract SupplyCoin is ERC20Token {
16
17     string public symbol;
18
19     address[] public owners;
20     uint256 public ownerCount;
21
22     constructor(string memory _name, string memory _symbol)
23
24     ERC20Token(_name) public { symbol = _symbol; }
25
26     //Añadir tokens
27     function mint(uint256 refund) public {
28         super.mint(refund);
29         ownerCount ++;
30         owners.push(msg.sender);
31     }
32
33 }
```

The bottom panel shows the terminal output, which includes a search bar and a list of functions: 'remix.loadurl(url)', 'remix.setproviderurl(url)', 'remix.execute(filepath)', 'remix.exeCurrent()', 'remix.help()', and 'remix.debugHelp()'. It also displays a welcome message for Remix v0.10.1.

Deploy 'SupplyCoin Contract' :

The screenshot displays the Remix IDE interface during the deployment of the SupplyCoin contract. The left sidebar contains the 'DEPLOY & RUN TRANSACTIONS' panel, which is configured with the following settings:

- ENVIRONMENT:** JavaScript VM
- ACCOUNT:** 0x1e2...b3897 (99.9999999)
- GAS LIMIT:** 3000000
- VALUE:** 0 wei
- CONTRACT:** SupplyCoin - browser/SupplyChain.sol
- DEPLOY:** _NAME: SupplyCoin, _SYMBOL: Group6
- PUBLISH TO IPFS:** Unchecked
- At Address:** Load contract from Address
- Transactions recorded:** 1
- Deployed Contracts:** ERC20TOKEN AT 0X643...075D8 (MEMORY)

The main editor shows the Solidity code for the SupplyChain.sol file:

```
1 pragma solidity ^0.5.16;
2
3 //This is the standard of the token ERC20
4 contract ERC20Token {
5
6     string public name;
7     mapping(address => uint256) public balances;
8
9     constructor(string memory _name) public { name = _name; }
10    function mint(uint256 refund) public { balances[msg.sender] += refund; }
11 }
12
13 //This is the token of the supply chain
14 contract SupplyCoin is ERC20Token {
15
16     string public symbol;
17
18     address[] public owners;
19     uint256 public ownerCount;
20
21     constructor(string memory _name, string memory _symbol)
22
23     ERC20Token(_name) public { symbol = _symbol; }
24
25     //Añadir tokens
26     function mint(uint256 refund) public {
27         super.mint(refund);
28         ownerCount ++;
29         owners.push(msg.sender);
30     }
31 }
```

The bottom panel shows the terminal output, which includes the following messages:

- remix.help(): Display this help message
- remix.debugHelp(): Display help message for debugging
- Welcome to Remix v0.10.1 -
- You can use this terminal for:

 - Checking transactions details and start debugging.
 - Running JavaScript scripts. The following libraries are accessible:
 - web3 version 1.0.0
 - ethers.js
 - swarmgw
 - remix (run remix.help() for more info)
 - Executing common command to interact with the Remix interface (see list of commands)
 - Use exports/.register(key, obj)/.remove(key)/.clear() to register and reuse objects

Deploy 'supply_Chain Contract':

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

JavaScript VM

ACCOUNT

0x1e2...b3897 (99.9999999)

GAS LIMIT

3000000

VALUE

0 wei

CONTRACT

supply_Chain - browser/SupplyChain

DEPLOY

_WALLET: 0x1e2fC3636D1a090c6Ba4D2

_TOKEN: 0x9A7062a411f7fBD1B6308d

transact

☐ PUBLISH TO IPFS

OR

At Address Load contract from Address

Transactions recorded 2

Deployed Contracts

> ERC20TOKEN AT 0X643...075D8 (MEMORY)

> SUPPLYCOIN AT 0X9A7...F550E (MEMORY)

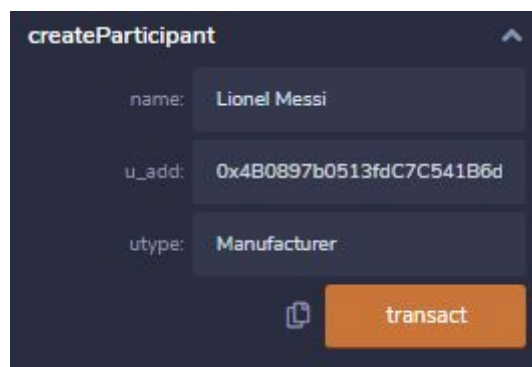
Supply chain interaction

Participants and roles

We create 3 participant on the supply chain with three different roles :

- Lionel Messi
 - Address : 0x4B0897b0513fdC7C541B6d9D7E929C4e5364D2dB
 - UserType : Manufacturer
- Marie Curie
 - Address : 0x583031D1113aD414F02576BD6afaBfb302140225
 - UserType : Supplier
- Albert Einstein
 - Address : 0xD870fA1b7C4700F2BD7f44238821C26f7392148
 - UserType : Customer

Here an example of a new participant creation :



```
[vm] from:0x4b0...4d2db to:supply_chain.createParticipant(string,address,string) 0x8ff...76f8a value:0 wei data:0x40c...0000 logs:0 hash:0xf74...f7df2

status                                0x1 Transaction mined and execution succeed
transaction hash                       0xf74115b6425f993a388f0d3d047de0324f0ba1bbd197b69a1e6e57c903ff7df2
from                                   0x4b0897b0513fdC7C541B6d9D7E929C4e5364D2dB
to                                     supply_chain.createParticipant(string,address,string) 0x8ffdb98daffa804c912a6224460b1a7c63776f8a
gas                                    3000000 gas
transaction cost                       110535 gas
execution cost                         85295 gas
hash                                   0xf74115b6425f993a388f0d3d047de0324f0ba1bbd197b69a1e6e57c903ff7df2
input                                  0x40c...0000
decoded input                          {
  "string name": "Lionel Messi",
  "address u_add": "0x4B0897b0513fdC7C541B6d9D7E929C4e5364D2dB",
  "string utype": "Manufacturer"
}
decoded output                         {
  "0": "uint256: 0"
}
logs                                   []
value                                  0 wei
```

Here we can see that there are 3 participants and the information of the last one with Id:2 is:

The screenshot shows a web interface with a dark theme. At the top, there is a dropdown menu labeled 'participants' with the value '2' selected. Below this, the details for participant 2 are displayed: '0: string: participantName Albert Einstein', '1: address: participantAddress 0xdD870fA1b7C4700F2BD7f44238821C26f7392148', and '2: string: userType Customer'. Below these details, there are three buttons: 'products' (selected), 'token', and 'totalParticipan...'. The 'products' button shows a value of 'uint256'. The 'totalParticipan...' button shows a value of '0: uint256: 3'.

After creating the participants, we have to create a new product which will be transferred on the supply chain.

Products

Product :

- OwnerId : 0 (Manufacturer - Lionel Messi)
- Product Name : 1984
- Product Cost : 20000000
- Product Description : 1984 is a modern, critical and very current novel.

New Product transaction :

The screenshot shows a web interface for creating a new product transaction. The title is 'newProduct'. There are four input fields: 'own_id' with the value '0', 'prod_name' with the value '"1984"', 'prod_cost' with the value '20000000', and 'prod_description' with the value 'n, critical and very current novel.'. At the bottom right, there is an orange button labeled 'transact'.

```
[vm] from:0x4b0...4d2db to:supply_Chain.newProduct(uint256,string,uint256,string) 0x8ff...76f8a value:0 wei data:0x960...00000 logs:0 hash:0xab0...a1cfa

status                                0x1 Transaction mined and execution succeed
transaction hash                       0xab0d3c00233e063db461968bbd25cccc4f5109ccd97c91c68fb04b5bd15a1cfa ⓘ
from                                  0x4b0897b0513fdc7c541b6d9d7e929c4e5364d2db ⓘ
to                                    supply_Chain.newProduct(uint256,string,uint256,string) 0x8ffdb98daffa804c912a6224460b1a7c63776f8a ⓘ
gas                                    3000000 gas ⓘ
transaction cost                       195045 gas ⓘ
execution cost                         168717 gas ⓘ
hash                                  0xab0d3c00233e063db461968bbd25cccc4f5109ccd97c91c68fb04b5bd15a1cfa ⓘ
input                                  0x960...00000 ⓘ
decoded input                          {
  "uint256 own_id": {
    "_hex": "0x00"
  },
  "string prod_name": "1984",
  "uint256 prod_cost": {
    "_hex": "0x01312d00"
  },
  "string prod_description": "1984 is a modern, critical and very current novel."
} ⓘ
decoded output                         {
  "0": "uint256: 0"
} ⓘ
logs                                   [] ⓘ ⓘ
value                                  0 wei ⓘ
```

In this image we can see the information of the new product on the chain with ID:0.

```
products 0 ▼
0: string: productName 1984
1: uint256: productCost 20000000
2: string: productDescription 1984 is a modern,
critical and very current novel.
3: address: productOwner
0x4B0897b0513fdC7C541B6d9D7E929C4e5364D
2dB
4: uint256: manufactureDate 1591361639
```

Buy SupplyCoins

Let's transfer the product between different owners.

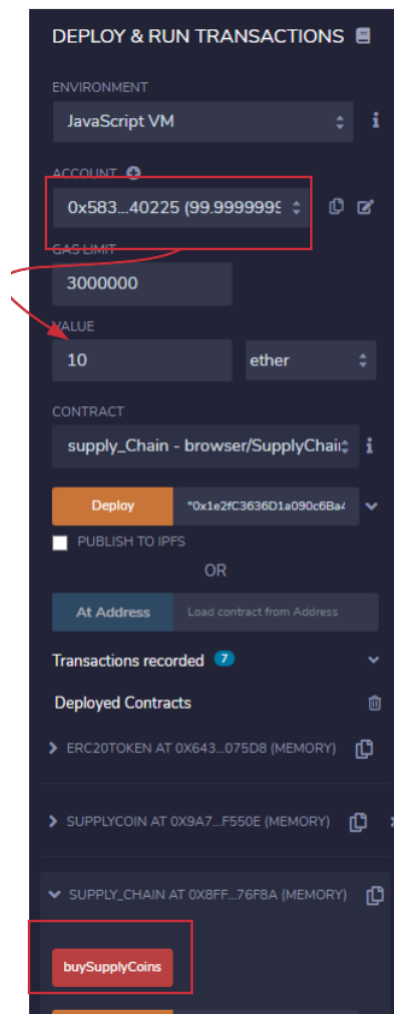
Manufacturer → Supplier → Customer

Before transferring products, the participants must have enough SupplyCoins to pay the transfers.

The supplier and the consumer have to buy SupplyCoins. This will be possible using function *buySupplyCoins()*.

This function will exchange from Ethereum to SupplyCoins with the value that the user decide.

Here an example of how Marie Curie (supplier) get 10 SupplyCoins :



In the image below it is available to check how the Marie Curie's SupplyCoins balance has changed.

The screenshot shows a blockchain interface with a dark theme. At the top, a red box highlights the address `0x583...40225` with a balance of `(89.9999999)`. Below this, the `GAS LIMIT` is set to `3000000` and the `VALUE` is `0` in `ether`. The `CONTRACT` section shows `supply_Chain - browser/SupplyChain` with a `Deploy` button and a dropdown menu showing `*0x1e2fC3636D1a090c6B54`. There is a checkbox for `PUBLISH TO IPFS` and an `OR` section with `At Address` and `Load contract from Address` buttons. Below this, there are sections for `Transactions recorded` and `Deployed Contracts`. The `Deployed Contracts` section lists `ERC20TOKEN AT 0X643...075D8 (MEMORY)` and `SUPPLYCOIN AT 0X9A7...F550E (MEMORY)`. The `SUPPLYCOIN` contract has buttons for `mint`, `moveTokens`, and `balances`. The `mint` button is highlighted in orange. Below these, a red box highlights the `getAccountBal...` function with a dropdown menu showing `0x583031D1113eD414F02f`. Below this, there is a `name` field and a red box highlighting the `ownerCount` field with a value of `0: uint256: 1`. At the bottom, there are fields for `owners` and `symbol`.

0x583...40225 (89.9999999)

GAS LIMIT

3000000

VALUE

0 ether

CONTRACT

supply_Chain - browser/SupplyChain

Deploy *0x1e2fC3636D1a090c6B54

☐ PUBLISH TO IPFS

OR

At Address Load contract from Address

Transactions recorded 0

Deployed Contracts

> ERC20TOKEN AT 0X643...075D8 (MEMORY)

> SUPPLYCOIN AT 0X9A7...F550E (MEMORY)

mint uint256 refund

moveTokens uint256 amount, address _to

balances address

getAccountBal... 0x583031D1113eD414F02f

0: uint256: 1000000000000000000

name

ownerCount 0: uint256: 1

owners uint256

symbol

Product transfer

After buying tokens for making possible the ownership transfer between parties, let's how to do it.

If the owner wants to transfer the ownership of a product have to use the function `transferOwnership_product()`.

The screenshot shows a web interface for the `transferOwnership_product` function. It has three input fields: `user1_id` with value 0, `user2_id` with value 1, and `prod_id` with value 0. Below these fields is a blue button labeled `transact`.

This function is only available to use by the owner of the product, if it is executed by another participant will be reverted as in this case is shown :

The screenshot shows a web application interface with a sidebar on the left and a main content area on the right. The sidebar contains sections for ACCOUNT, GAS LIMIT, VALUE, CONTRACT, Transactions recorded, and Deployed Contracts. The main content area displays the source code for the `transferOwnership_product` function. The function is defined as follows:

```
function transferOwnership_product(uint256 user1_id, uint256 user2_id, uint256 prod_id) public {
    //The functions with this modifier will be only able to execute by the owner of the product
    modifier onlyOwner(prod_id) {
        require(msg.sender == products[prod_id].productOwner, "Error: Only the owner can transfer a product to another participant.");
    }

    //Transfer the ownership of the product
    function createParticipant(string memory name, address _add, string memory utype) public returns (uint) {
        uint user_id = totalParticipants++;
        participant[user_id].participantName = name;
        participant[user_id].participantAddress = _add;
        participant[user_id].userType = utype;
        return user_id;
    }

    //Create a new product in the supply chain
    function newProduct(uint256 user_id, string memory prod_name, uint256 prod_cost, string memory prod_description) public returns (uint) {
        (uint256 user_id, string memory prod_name, uint256 prod_cost, string memory prod_description) = createParticipant(msg.sender, _add, utype);
        uint product_id = totalProducts++;
        products[product_id].productName = prod_name;
        products[product_id].productCost = prod_cost;
        products[product_id].productDescription = prod_description;
        products[product_id].productOwner = participant[user_id].participantAddress;
        products[product_id].manufactureDate = ...;
        return product_id;
    }

    return #;
}
```

The transaction log shows the following sequence of events:

- transact to supplyChain.createParticipant pending ...
- [w] from:0x83...40225 to:supplyChain.createParticipant(string,address,string) @0x83...ad97f value:0 wei data:0x40c...00000 logs:0 hash:0x260...d9f1b
- transact to supplyChain.buySupplyCoins pending ...
- [w] from:0x83...40225 to:supplyChain.buySupplyCoins() @0x83...ad97f value:1000000000000000000 wei data:0xc01...26990 logs:0 hash:0xc65...0c9a8
- call to SupplyCoin.getAccountBalance
- call [call] from:0x583810113a0414f82576806afab302140225 to:SupplyCoin.getAccountBalance(address) data:0x934...40225
- call to SupplyCoin.balances
- call [call] from:0x48889708513f6c7c54186d907e929c4c536402d8 to:SupplyCoin.balances(address) data:0x27e...40225
- call to SupplyCoin.balances
- call [call] from:0x48889708513f6c7c54186d907e929c4c536402d8 to:SupplyCoin.balances(address) data:0x27e...402d8
- transact to supplyChain.transferOwnership_product pending ...
- [w] from:0x83...531ea to:supplyChain.transferOwnership_product(uint256,uint256,uint256) @0x83...ad97f value:0 wei data:0xa08...00000 logs:0 hash:0xb0c1...50720
- transact to supplyChain.transferOwnership_product errored: VM error: revert.
- revert The transaction has been reverted to the initial state.
- Reason provided by the contract: "Error: Only the owner can transfer a product to another participant."
- Debug the transaction to get more information.

```

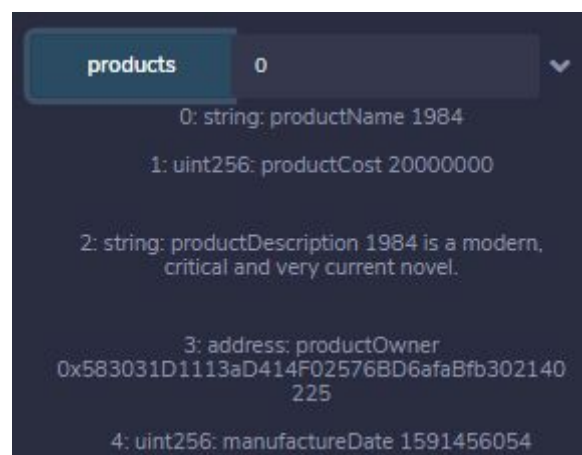
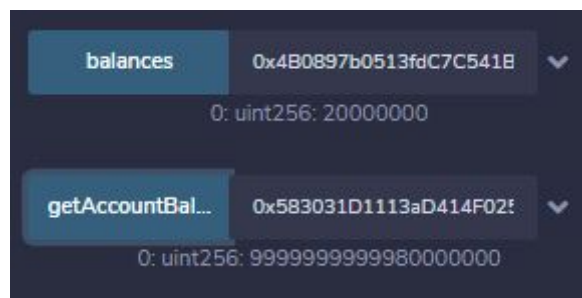
[vm] from:0x5d3...531ea to:supply_Chain.transferOwnership_product(uint256,uint256,uint256) 0xb43...ad97f value:0 wei data:0xa68...00000 logs:0 hash:0xbc1...50720

status                                0x0 Transaction mined but execution failed
transaction hash                       0xbc16bc464e86a6bebf4d2189ec5c769dc506d38a30223c87d13f014c7f50720
from                                  0x5d3a020cda74196484da5150ad811cc7d3531ea
to                                    supply_Chain.transferOwnership_product(uint256,uint256,uint256) 0xb43020bf4b2df84dabb594251580c2b05a2a897f
gas                                    30000000 gas
transaction cost                       23156 gas
execution cost                         1436 gas
hash                                  0xbc16bc464e86a6bebf4d2189ec5c769dc506d38a30223c87d13f014c7f50720
input                                  0xa68...00000
decoded input                          {
  "uint256_user1_id": {
    "hex": "0xa68"
  },
  "uint256_user2_id": {
    "hex": "0xa61"
  },
  "uint256_prod_id": {
    "hex": "0xa60"
  }
}
decoded output                         {
  "v": "bool: true"
}
logs                                   []
value                                  0 wei

transact to supply_Chain.transferOwnership_product errored: VM error: revert.
revert The transaction has been reverted to the initial state.
Reason provided by the contract: "Error: Only the owner can transfer a product to another participant.".
Debug the transaction to get more information.

```

Whereas if it is the owner who executes it, the transfer will be done without problems. The owner's SupplyCoins balance will be increased with the cost of the product.



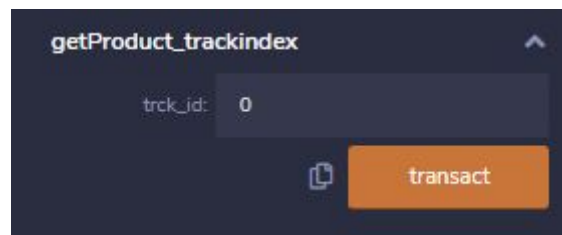
The productOwner address has changed. Now appear Marie Curie's address.

Product track

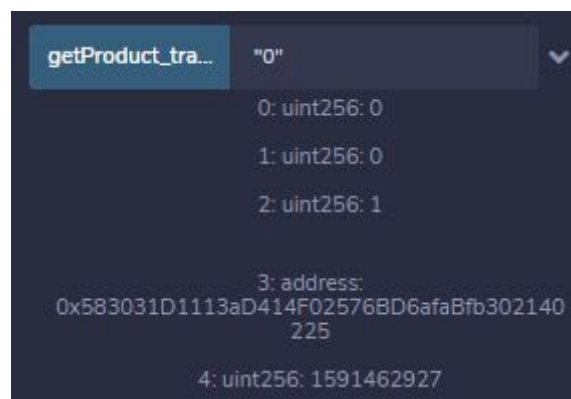
After transferring the ownership of a product, we can locate who currently owns a product. Every transfer between parties is registered at the tracks mapping.

To do this we only need to know the ID of the product to get the information of the transaction. In this case the ID is : 0.

Here an example :



Using it we'll be able to get the information of the track.



The value that we can see here are :

- Product ID
- Previous Owner ID
- Current Owner ID
- Current Owner Address
- The time stamp of the transaction

If we transfer the product again we'll see how these information change.

Let's transfer between Marie Curie (Supplier) and Albert Einstein (Customer).

The balances and the ownership have changed :

