

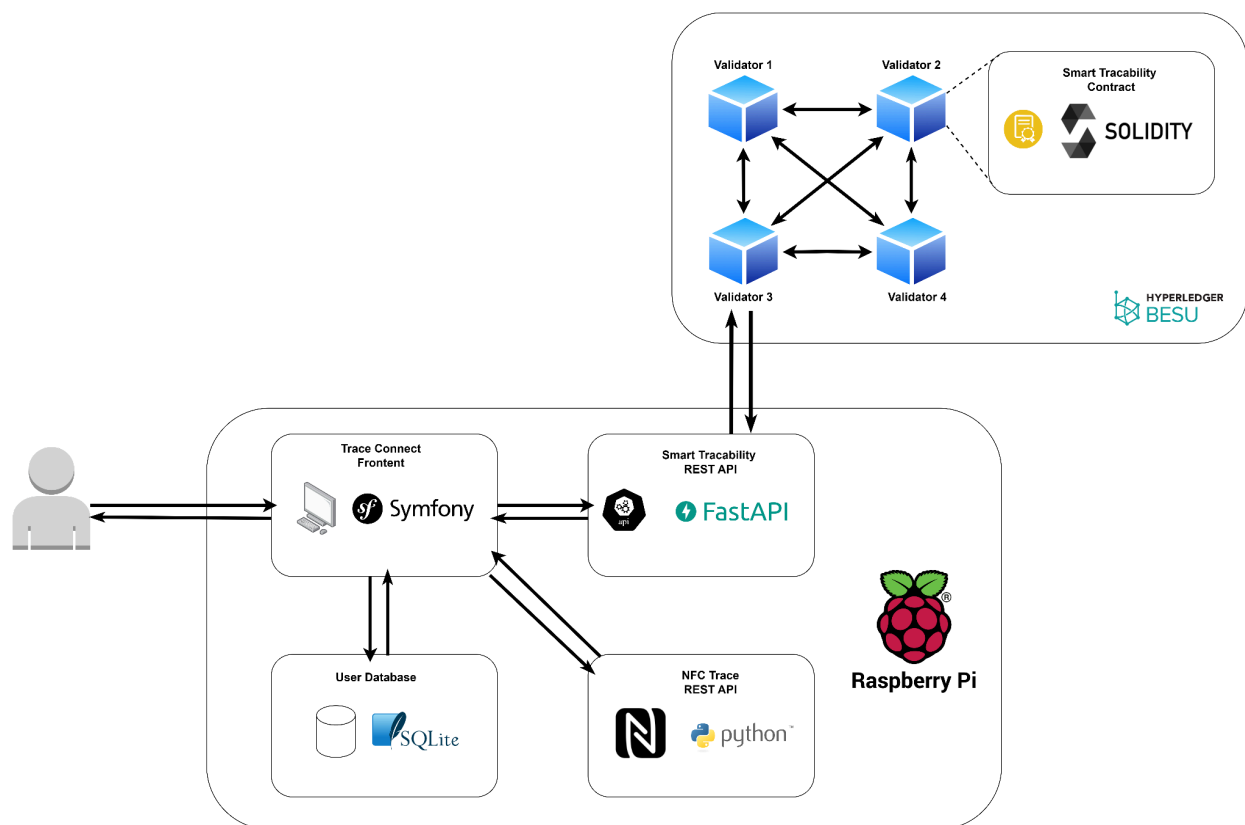
BC24 Technical Documentation

Overview

The BC24 project consists of several microservices developed by multiple student groups. This documentation should give a brief overview, subsequently focusing on the backbone project Smart Tracability (Smart contract and API).

The project revolves around the BC24-Contract, a Solidity smart contract designed for managing resources on a private Ethereum blockchain.

General Architecture



NFC-Trace

Rest API that will let the caller write to or read from a NFC tag.

Additional functionality includes two modules to capture the temperature as well as getting the current GPS location.

Endpoint	Function	Description
/startReader	GET	<p>Gets the following information from the NFC and from the Temperature and GPS modules:</p> <p>Example response</p> <pre>{ "NFC_ID": "X", "NFT_tokenID": "Y", "creationDate": "2023-04-01", "lastModificationDate": "2023-04-10", "scanDate": "2023-04-15", "temperature": "25.5", "gps": { "longitude": "-122.4194", "latitude": "37.7749", "altitude": "15" } }</pre>
/write	POST	<p>Writes the following information on the NFC tag:</p> <pre>{ "NFC_ID": "X", "NFT_tokenID": "Y", "creationDate": "2023-04-20", "lastModificationDate": "2023-04-20" }</pre>

All additional information can be found here: [NFC-Trace Notion Workspace](#)

Trace-connect

Describes the graphical interface for the defined usecase. Interacts with all the different microservices.

All additional information can be found here: [Trace Connect Notion Workspace](#)

Smart Traceability Contract

Technologies and environment

Environment with Hardhat

Hardhat is a comprehensive development environment designed to facilitate the creation, testing, and deployment of Ethereum-based smart contracts. It provides a suite of tools and features that streamline the development process for blockchain developers.

The following features have been extensively used:

- Local network
- Solidity contract compilation & Artifact builds
- Testing suites
- Local and prod testing
- Deployments

Overall hardhat has been a valuable environment which was prepared by OpenZeppelin contract forge.

Contract Development

OpenZeppelin

OpenZeppelin is an open-source framework and library for developing secure smart contracts on Ethereum and other blockchain platforms.

It is the designated starting point since it allows to create repositories with already established hardhat dev environments.

The Smart Traceability smart contract integrates several concepts. At its base it is a contract that allows fungible as well as non-fungible tokens. This is perfect if client resource templates use different unit measurements (for example 1 sheep vs 10 kg of sheep shoulder)

ERC1155Burnable

The [ERC1155Burnable](#) token is an extension of the [ERC-1155](#) multi-token standard, enabling efficient creation, management, and burning of multiple token types within a single contract.

AccessControl

[AccessControl](#) is a flexible and extensible authorization mechanism that allows the assignment of roles to different accounts. This contract facilitates role-based access control, ensuring only authorized accounts can execute specific functions. Roles can be granted and revoked dynamically, enhancing security and management within the smart contract.

The potential roles are dynamically defined in the templates (see below).

The owner of the contract holds the admin role and only this role can distribute roles to other users / wallet addresses.

Generalization of traceability

The contract is built to allow user specific supply chain traceability projects. This means that on contract deployment it is the client who will define the following:

- Resources that can be minted
- Access Control
 - For minting
 - Setting and changing metadata
 - Transferring to new owner

The resource templates is a list of json objects of the following form:

```
{
  ressource_id: 1,
  ressource_name: "Sheep",
  ressource_type: "Animal",
  needed_resources: [],
  needed_resources_amounts: [],
  initial_amount_minted: 1,
  required_role: "BREEDER",
  produces_resources: [],
  produces_resources_amounts: [],
}
```

ressource_id: This is the unique identifier for the resource. In this case, the ID is 1.

ressource_name: This is the name of the resource. In this case, the resource name is "Sheep"

ressource_type: This is the type of the resource. In this case, the resource type is "Animal"

needed_resources: This is an array of resource IDs that are required to produce this resource. In this case, the array is empty, which means no other resources are needed to produce this resource.

needed_resources_amounts: This is an array that specifies the amount of each needed resource required to produce this resource. The indices in this array correspond to the indices in the needed_resources array. In this case, the array is empty, which is consistent with needed_resources being empty.

initial_amount_minted: This is the initial amount of this resource that exists. In this case, the creation of a sheep resource will create one NFT, which represents this resource.

required_role: This is the role required to produce this resource. In this case, the role is BREEDER.

produces_resources: This is an array of resource IDs that this resource can produce. In this case, the array is empty, which means this resource doesn't produce any other resources. This has been introduced to handle more complex resource processing. A resource that is used to create multiple and different NFTs needs to keep track of those. To replicate this on the blockchain with the creation of tokens a function is provided which will automatically create those NFTs defined in this array.
See the example.

produces_resources_amounts: This is an array that specifies the amount of each produced resource that this resource can produce. The indices in this array correspond to the indices in the produces_resources array. In this case, the array is empty, which is consistent with produces_resources being empty

The following is an example:

- 1) A breeder can mint a token when a sheep is born
 - a) it creates a NFT of type sheep with quantity 1
- 2) A slaughterer can then kill the sheep and create a sheep carcass then cut a sheep carcass in 2 demi-carcass
- 3) A manufacturer can take this carcass and process it, resulting in the creation of meat or the creation of a product from a recipe

For our Sheep the meat produced in our POC would be :

- a) a sheep shoulder with weight 2500 g
- b) a sheep hip with weight 5000g
- c) etc

The sheep can be used in a recipe, for exemple : merguez patty

```
export const ressourceTemplates = [
  // <-----> Animal
  {
    ressource_id: 42,
    ressource_name: "Sheep",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 1,
    required_role: "BREEDER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type: "Animal",
  },
  // <-----> Carcass
  {
    ressource_id: 45,
    ressource_name: "Sheep carcass",
    needed_resources: [42],
    needed_resources_amounts: [1],
    initial_amount_minted: 1,
    required_role: "SLAUGHTERER",
    produces_resources: [48,49],
    produces_resources_amounts: [1,1],
    ressource_type : "Carcass"
  },
  // <-----> Demi Carcass
  {
    ressource_id: 48,
    ressource_name: "Sheep Left Demi Carcass",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 1,
```

```

    required_role: "SLAUGHTERER",
    produces_resources: [54, 55, 56, 57, 58],
    produces_resources_amounts: [1, 1, 1, 1, 1],
    ressource_type : "Demi Carcass"
},
{
    ressource_id: 49,
    ressource_name: "Sheep Right Demi Carcass",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 1,
    required_role: "SLAUGHTERER",
    produces_resources: [54, 55, 56, 57, 58],
    produces_resources_amounts: [1, 1, 1, 1, 1],
    ressource_type : "Demi Carcass"
},

// <-----> Meat
{
    ressource_id: 54,
    ressource_name: "Sheep shoulder",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 2500,
    required_role: "MANUFACTURER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type: "Meat",
},
{
    ressource_id: 55,
    ressource_name: "Sheep hip",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 5000,
    required_role: "MANUFACTURER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type : "Meat"
},
{
    ressource_id: 56,
    ressource_name: "Sheep back",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 5000,
    required_role: "MANUFACTURER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type : "Meat"
},
{
    ressource_id: 57,
    ressource_name: "Sheep ribs",

```



```

        needed_resources: [],
        needed_resources_amounts: [],
        initial_amount_minted: 7500,
        required_role: "MANUFACTURER",
        produces_resources: [],
        produces_resources_amounts: [],
        ressource_type : "Meat"
    },
    {
        ressource_id: 58,
        ressource_name: "Sheep brains",
        needed_resources: [],
        needed_resources_amounts: [],
        initial_amount_minted: 700,
        required_role: "MANUFACTURER",
        produces_resources: [],
        produces_resources_amounts: [],
        ressource_type : "Meat"
    }
];

```

From these meat in combination with beef meat :

```

{
    ressource_id: 59,
    ressource_name: "Beef",
    needed_resources: [],
    needed_resources_amounts: [],
    initial_amount_minted: 3500,
    required_role: "MANUFACTURER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type : "Meat"
}

```

you can create **Merguez Patty** which takes 30g of **Sheep shoulder** and 70g of **Beef**

```

{
    ressource_id: 66,
    ressource_name: "Merguez Patty",
    needed_resources: [54, 59],
    needed_resources_amounts: [30, 70],
    initial_amount_minted: 100,
    required_role: "MANUFACTURER",
    produces_resources: [],
    produces_resources_amounts: [],
    ressource_type : "Meat"
},

```

if you mint 2 you will have 200g of Merguez Patty.

This is an illustration on how the templates work and can be extended depending on the usecase. We can imagine the usecase for jewelry ; with silver and diamond you can create a ring which will be traceable.

Installation and Setup

Please see the repository Readme for more information regarding running the Smart Contract:
<https://github.com/bc24-miage-dev/BC24-Contract.git>

Smart Traceability API

To facilitate easier communication with the smart contract we implemented a Rest API:

<https://github.com/bc24-miage-dev/BC24-API>

Documentation

Swagger UI API documentation is used (see README)

Please see the documentation of the repository for more information.

Testing and Validation

Testing

All the testing can be reproduced by following the Readme of the project.

Unit testing

There are 26 unit tests that verify the correct functionality of the Smart Contract.

Test Coverage

Our test cover:

- 100% of statements
- 83% of branches
- 90% of functions

100% Statements 84/84 83.33% Branches 35/42 90.91% Functions 35/31 100% Lines 308/308

File ▾		Statements ▾		Branches ▾		Functions ▾		Lines ▾						
contracts/	<div></div>	100%		84/84		83.33%		35/42		90.91%		10/11		100%

Branches:

Not all branches are tested since some of our code checks conditions that can apply to several roles. As such we have some edge cases that are not covered.

Functions:

There is a function which needs to be overridden because of inheritance. This function is not tested.

Performance

Stress testing

Stress testing has been done via the python API. We used locust to automatically and randomly create requests.

Case 1

See [Case 1 report](#) for more information

Setting	
Number of concurrent users	1
Function call delays	1-4 seconds
Number of metadata	1 key value pair

Key findings	
Number of requests	94
Failing requests	8 (9%)

2	POST	/resource/mintToMany	HTTPError('500 Server Error: Internal Server Error for url: /resource/mintToMany')
6	POST	/resource/transfer	HTTPError('500 Server Error: Internal Server Error for url: /resource/transfer')

- Transfer failures due to implementation details and timing issues in the stressTest.py.
 - They are neglectable but for the sake of completeness are still listed
- mintToMany failures could not be clearly identified but are assumed connected to transfer errors.
 - Resources that failed to properly transfer to a new owner will subsequently throw errors if a user wants to mint something with resources that are not associated with the correct wallet address

Average time to complete	5503.45 ms
Max time to complete	16148 ms

Case 2

See [Case 2 report](#) for more information

Setting	
Number of concurrent users	3
Function call delays	1-4 seconds
Number of metadata	1 key value pair

Key findings	
Number of requests	97
Failing requests	22 (23%)

5	POST	/resource/mint	HTTPError('500 Server Error. Internal Server Error for url: /resource/mint')
1	POST	/resource/mintToMany	HTTPError('500 Server Error. Internal Server Error for url: /resource/mintToMany')
16	POST	/resource/transfer	HTTPError('500 Server Error. Internal Server Error for url: /resource/transfer')

- Transfer failures due to implementation details and timing issues in the stressTest.py.
 - They are neglectable but for the sake of completeness are still listed
- mintToMany failures could not be clearly identified but are assumed connected to transfer errors.
 - Resources that failed to properly transfer to a new owner will subsequently throw errors if a user wants to mint something with resources that are not associated with the correct wallet address

Average time to complete	14183.87 ms
Max time to complete	42027 ms

Considerations

The minton function cannot be optimised.

The way on how the metaData is recuperated could be further enhanced by implementing a caching system on the fastAPI side. This is not in the scope and is only intended to give an idea for future work.

Maintenance and Troubleshooting

There are several system interacting with each other and all of them need to be maintained.

BESU network

The besu network is maintained by Nicolas Herbaut.

There are functionalities for restarting the network via:

<https://explorer.bc24.miage.dev/explorer/explorer>

The local blockchain node stati and block explorer can be found here:

<https://explorer.bc24.miage.dev/explorer/nodes>

Smart Contract

The smart contract needs to be deployed on one of the validator nodes from the network.

In the following cases the contract needs to be relaunched and will be in the initial state with no tokens:

1. Besu network reset
2. Contract changes and updates
 - a. the contract has fixed resource templates
3. New use case / client with new resources

Smart Traceability API

The python api will need to be updated with the correct contract address and the new abi, each time a new contract is launched on the chain.