



ÉCOLE  
**D'INGÉNIEURS**  
PARIS-LA DÉFENSE

# Monnaies Numériques



**Rendu TD6 : ERC20**

**AOUES GAYA**

## I. Git Repository

<https://github.com/Gayardo/ERC721>

Le git a été partagé avec l'enseignant Henri Lieutaud par mail.

## II. On crée un ERC721 Token Contract :

Dans le dossier contracts, on crée un nouveau contrat du nom de Token.sol. On suit le modèle de erc721 sur le github de la librairie Open Zeppelin.

On compile avec la commande :

**Truffle compile**

Pour vérifier qu'il n'y pas d'erreurs, On fait attention aux fichiers d'import et aux erreurs de versions de solidity.

## III. On implémente toutes les fonctions d'un ERC721 :

Le code est disponible sur git dans contracts/Token.sol

Les fonctions implémentées :

- `balanceOf(address owner)`
- `ownerOf(uint256 tokenId)`
- `approve(address to, uint256 tokenId)`
- `getApproved(uint256 tokenId)`
- `isApprovedForAll(address owner, address operator)`
- `safeTransferFrom(address from, address to, uint256 tokenId)`
- `_exists(uint256 tokenId)`
- `_isApprovedOrOwner(address spender, uint256 tokenId)`
- `_safeMint(address to, uint256 tokenId)`
- `_mint(address to, uint256 tokenId)`
- `_burn(address owner, uint256 tokenId)`
- `_transferFrom(address from, address to, uint256 tokenId)`
- `_checkOnERC721Received(address from, address to, uint256 tokenId, bytes memory _data)`
- `_clearApproval(uint256 tokenId)`

⇒ On teste avec la commande `truffle compile`, pour voir si le contrat compile et n'a pas d'erreurs.

### Migration vers Ganache :

Une fois que ganache est installé, on clique sur quick Start. Afin d'effectuer la migration, on modifie le fichier `truffle-config.js` pour lui signifier la façon dont on se connecte à ganache.

On enlève les deux barres de commentaires dans la section **Développement** pour modifier les champs suivants :

Host : « 127.0.0.1 »

Port : 7545

Network id : « \* »

### Script pour la migration :

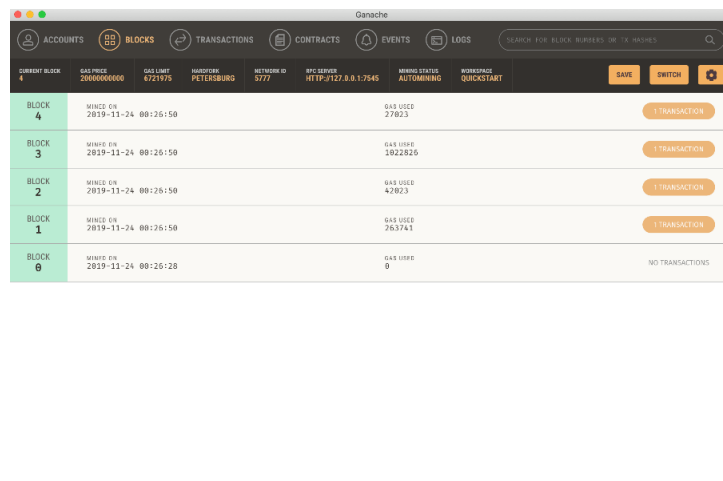
Dans le dossier migrations, on crée un fichier `2_deploy_contracts.js` dans lequel on met ce bout de code.

```
1 const Token = artifacts.require('Token');
2 module.exports = (deployer) => {
3   deployer.deploy(Token);
4 };
```

Ensuite, on lance la migration avec la commande :

**truffle migrate**

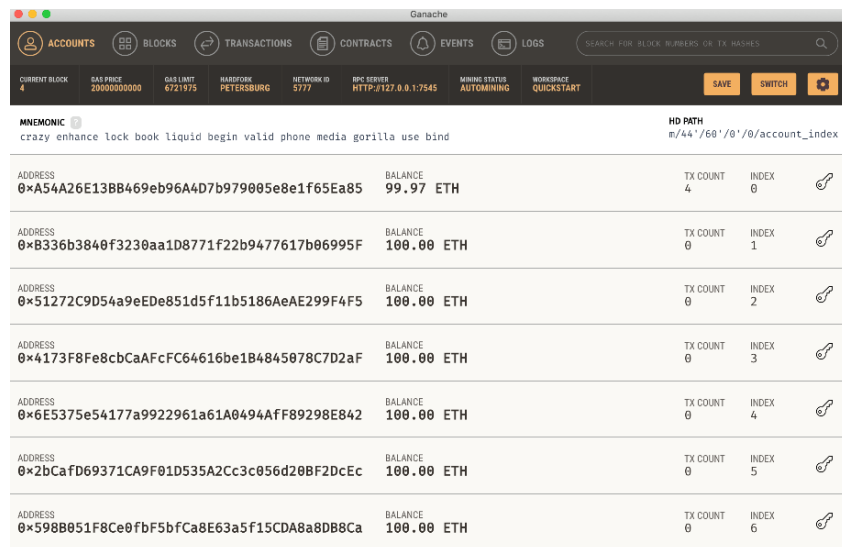
On peut alors voir les blocks qui ont été créés sur ganache :



The screenshot shows the Ganache application window. The 'BLOCKS' tab is selected in the top navigation bar. Below the navigation bar, there is a status bar with various metrics: Current Block #, Gas Price, Gas Limit, Hardfork, Network ID, RPC Server, Mining Status, and Workspaces. The main area displays a table of blocks. The table has columns for Block number, Block range, Block time, Gas used, and a button to view the transaction. The blocks are numbered 0 to 4, with block 0 having no transactions and blocks 1 to 4 each having one transaction.

Block	Block Range	Block Time	Gas Used	Transaction
BLOCK 4	2819-11-24	00:26:150	27823	TRANSACTION
BLOCK 3	2819-11-24	00:26:150	282325	TRANSACTION
BLOCK 2	2819-11-24	00:26:150	42823	TRANSACTION
BLOCK 1	2819-11-24	00:26:150	263741	TRANSACTION
BLOCK 0	2819-11-24	00:26:128	0	NO TRANSACTIONS

Et la réduction des ethers du premier account.



The screenshot shows the Ganache application window. At the top, there are tabs for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below these tabs is a search bar and a status bar with various metrics like CURRENT BLOCK, GAS PRICE, GAS LIMIT, NETWORK, and more. The main area displays a list of accounts with their addresses, balances, transaction counts, and indices. The first account has a balance of 99.97 ETH, while the others have 100.00 ETH.

ADDRESS	BALANCE	TX COUNT	INDEX
0xA54A26E13BB469eb96A4D7b979085e8e1f65Ea85	99.97 ETH	4	0
0xB336b3840f3230aa1D8771f22b9477617b06995F	100.00 ETH	0	1
0x51272C9D54a9eEDe851d5f11b5186AeAE299F4F5	100.00 ETH	0	2
0x4173F8Fe8cbCaAFcFC64616be1B4845078C7D2aF	100.00 ETH	0	3
0x6E5375e54177a9922961a61A0494AfF89298E842	100.00 ETH	0	4
0x2bCaFD69371CA9F01D535A2Cc3c056d20BF2DcEc	100.00 ETH	0	5
0x598B051F8Ce0fbF5bFCa8E63a5f15CDA8a8DB8Ca	100.00 ETH	0	6

## IV. Register Breeder:

RegisterBreeder : On déclare un éleveur. Cela ressemble à la whitelist implémentée au TD5.

Pour l'implémenter, on utilise les contrats : **Roles.sol**, **WhitelistAdminRole.sol** et **WhitelistedRole.sol**

On fait hériter notre contrat **Token.sol** du contrat **WhitelistedRole.sol** qui lui-même hérite du contrat **WhitelistAdminRole.sol**. Enfin, les deux derniers contrats utilisent **Roles.sol**.

Dans notre contrat **Token**, on rajoute dans le constructeur la fonction `__addWhitelistAdmin(msgSender())`. Quand le contrat est déployé, on est au moins sûr d'avoir un admin qui va pouvoir enregistrer les nouveaux éleveurs.

On crée alors la fonction `registerBreeder(address account)` dans laquelle on ajoute un éleveur à notre whitelist.

```
function registerBreeder(address account) public {
    addWhitelisted(account);
}
```

## V. Declare animal :

On va créer une structure pour un animal avec 5 caractéristiques :

```
struct Animal {  
    uint id;  
    typeAnimal race;  
    uint age;  
    Color color;  
    uint rarity;  
    string name;  
}
```

Type Animal et Color correspondent à des énumérations.

```
enum typeAnimal { Cow, Horse, Chicken, Pig, Sheep, Donkey, Rabbit }  
enum Color { Brown, Black, White, Red, Blue }
```

On va ensuite créer un mapping pour ranger nos animaux dans une liste. Chaque éleveur possède une liste d'animaux vide ou non.

```
mapping (address => Animal[]) public _animalsOfOwner;
```

On crée enfin notre fonction `DéclareAnimal(...)`

Qui ne peut être exécutée que par un éleveur enregistré et pour ça, on ajoute **onlyWhitelisted** dans la signature de la fonction.

```
function declareAnimal(address to, typeAnimal race, uint age, Color color, uint rarity, string memory name)  
    public onlyWhitelisted() returns (bool) {  
    _currentId++;  
    Animal memory animal = Animal(_currentId, race, age, color, rarity, name);  
    _animalsOfOwner[msg.sender].push(animal);  
  
    _mint(to, _currentId);  
    return true;  
}
```

## VI. Dead animal :

Pour tuer un animal, il suffit de l'enlever du mapping et de détruire le token avec la fonction `_burn` du contrat ERC721.

Voici le code :

```
function deadAnimal(uint id) public onlyWhitelisted() {
    _burn(msg.sender, id);

    address owner=msg.sender;
    uint size = _animalsOfOwner[owner].length;
    for (uint index = 0; index < size; index++) {
        Animal storage animal = _animalsOfOwner[owner][index];
        if (animal.id == id) {
            if (index < size - 1) {
                _animalsOfOwner[owner][index] = _animalsOfOwner[owner][size - 1];
            }
            delete _animalsOfOwner[owner][size - 1];
        }
    }
}
```

Quand on supprime de la liste, on décale tous les animaux après le notre à gauche d'une position.

## VII. Déploiement sur Rinkeby :

Pour déployer sur le testnet rinkeby, on utilise **Infura**.

On crée un compte puis un projet.

On installe truffle hd wallet provider :

```
npm install --save truffle-hdwallet-provider
```

On modifie notre truffle-config.js en ajoutant le network rinkeby, on spécifiant la infura key dans le lien de connexion et on renseigne notre mnemonic qu'on peut trouver sur metamask.

```
const HDWalletProvider = require('truffle-hdwallet-provider');
// vous m'avez dit la dernière fois de ne jamais l'écrire quelque
// part, donc je l'ai enlevé du code

const mnemonic = "mon mnemonic en 12 mots";
module.exports = {
    networks: {
        rinkeby: {
            provider: function() {
                var str1 = "https://";
                str1.concat("/infura.io/project/5e360948fa294be3aaf23c463195e2e3") ;
                return new HDWalletProvider(mnemonic, str1) ;
            }
        }
    }
}
```

```
        },  
        network_id: 1  
    }  
},  
}
```

On effectue ensuite la migration avec la commande :

**truffle migrate -f 2 --network rinkeby**

En output, on a alors l'adresse de notre contrat.