



ÉCOLE
D'INGÉNIEURS
PARIS-LA DÉFENSE

Monnaies Numériques



Rendu TD7 : UX

AOUES GAYA

I. Git Repository

https://github.com/Gayardo/Ux_Ethereum.git

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

II. On crée un ERC721 Token Contract et implémente ses fonctions:

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.

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 };

```

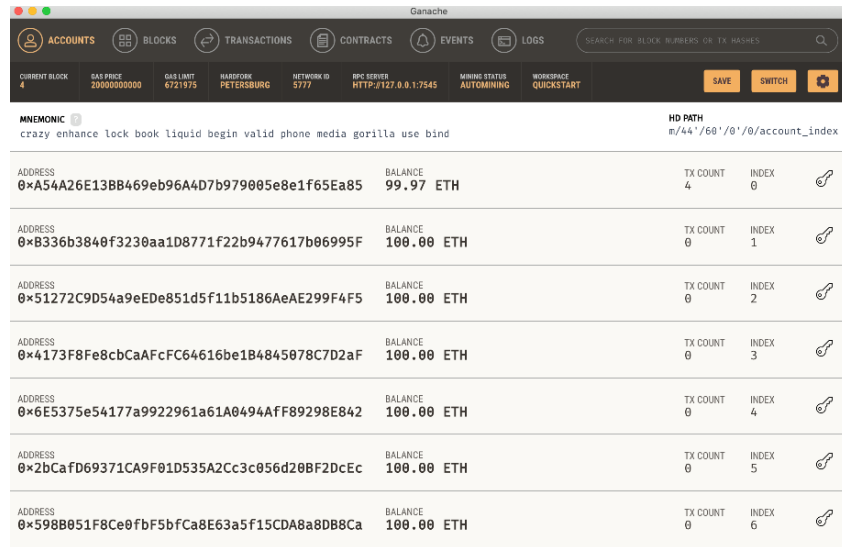
Ensuite, on lance la migration avec la commande :

```
truffle migrate
```

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

Block	Hash	Time	Size	Transactions
4	0x1234567890123456789012345678901234567890123456789012345678901234	2019-11-24 00:26:50	645 USED 27823	1 TRANSACTION
3	0x1234567890123456789012345678901234567890123456789012345678901234	2019-11-24 00:26:50	645 USED 1022826	1 TRANSACTION
2	0x1234567890123456789012345678901234567890123456789012345678901234	2019-11-24 00:26:50	645 USED 42823	1 TRANSACTION
1	0x1234567890123456789012345678901234567890123456789012345678901234	2019-11-24 00:26:50	645 USED 763761	1 TRANSACTION
0	0x1234567890123456789012345678901234567890123456789012345678901234	2019-11-24 00:26:28	645 USED 0	NO TRANSACTIONS

Et la réduction des ethers du premier account.



ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS
CURRENT BLOCK: 4 GAS PRICE: 20000000000 GAS LIMIT: 6721975 HARDFORK: PETERSBURG NETWORK ID: 5777 RPC SERVER: HTTP://127.0.0.1:7545 MINING STATUS: AUTOMINING WORKSPACE: QUICKSTART					
MNEMONIC: crazy enhance lock book liquid begin valid phone media gorilla use bind HD PATH: m/44'/60'/0'/0/account_index					
ADDRESS	BALANCE	TX COUNT	INDEX		
0xA54A26E13BB469eb96A4D7b979005e8e1f65Ea85	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		

III. Nom du Token Registry et compteur de tokens créés:

On crée la variable name, pour le nom du token registry :

```
string public name;
```

On crée la variable currentId pour compter le nombre de tokens créés :

```
uint public currentId;
```

Enfin, dans le constructeur du contrat, on initialise le nom du registre.

```
constructor () public {  
    // register the supported interfaces to conform to ERC721 via ERC165  
    _registerInterface(_INTERFACE_ID_ERC721);  
    name="GayardoRegistry";  
    counter=0;  
}
```

IV. Création de la fonction mint utilisable que par ceux ayant payé 0.1 ethers :

Dans cette fonction, on appelle la fonction `_mint(address to, uint256 tokenId)` définie dans le contrat. Mais avant la création on incrémente le compteur de tokens `_currentId`.

Pour faire en sorte que la fonction ne soit utilisable que si on a payé 0.1 ethers, on ajoute **payable** dans la signature de la fonction et un **require** dans l'implémentation avec comme condition

Msg.value == 0.1 ethers

```
function mintToken(address to) payable public returns (bool){  
  
    require(msg.value == 0.1 ether);  
    _currentId++;  
    _mint(to, _currentId);  
  
    return true;  
}
```

V. Création d'une react App :

D'abord, je crée un dossier myReact pour mon application.

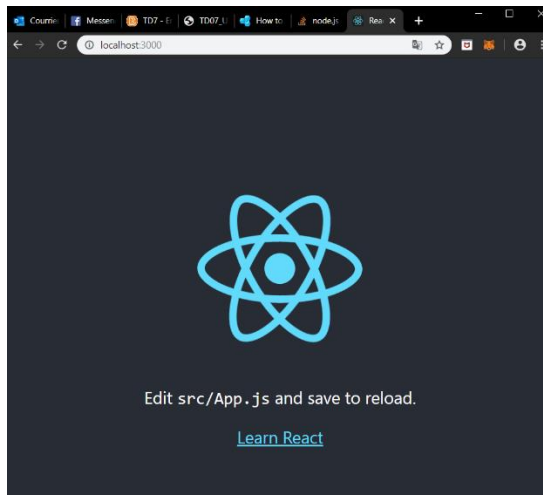
A l'intérieur, j'installe create-react-app :
npm install create-react-app

Je crée donc l'application avec cette commande :

create-react-app myapp
cd myapp

On lance le webserver :
npm run start

Notre navigateur s'ouvre automatiquement pour faire apparaître notre app real comme sur l'image ci-dessous :



On modifie le fichier package.json pour ajouter web3 dans les dependencies :

```
"dependencies": {  
  "bootstrap": "^4.3.1",  
  "react": "^16.8.3",  
  "react-bootstrap": "^1.0.0-beta.5",  
  "react-dom": "^16.8.3",  
  "react-scripts": "2.1.5",  
  "web3": "^1.0.0-beta.46"  
},
```

On code alors le backend de notre page sur App.js :

VI. Connexion à la blockchain et affichage du chainId et du numéro du dernier block :

On déploie d'abord notre contrat sur ganache pour s'assurer d'avoir quelques blocks. Sur ganache on a 4 blocks

On affiche le chainID (Peut être que vous demandez le network Id mais dans mon cas j'ai pris l'adresse du premier account)

Pour avoir la liste des comptes :

```
const accounts = await web3.eth.getAccounts()
```

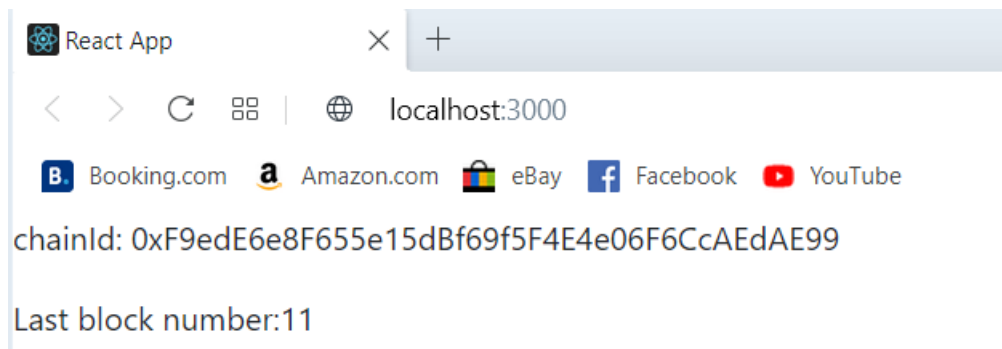
Pour avoir le numéro du dernier block

```
const lastBlock= await web3.eth.getBlockNumber()
```

Voici le code complet de notre App.js

```
1  import React, { Component } from 'react'
2  import Web3 from 'web3'
3  import './App.css'
4
5  class App extends Component {
6    componentWillMount() {
7      this.loadBlockchainData()
8    }
9
10   async loadBlockchainData() {
11     const web3 = new Web3(Web3.givenProvider || "http://localhost:8545")
12     const accounts = await web3.eth.getAccounts()
13     const lastBlock= await web3.eth.getBlockNumber()
14     this.setState({ account: accounts[0] })
15     this.setState({ BlockNumber: lastBlock})
16   }
17   constructor(props) {
18     super(props)
19     this.state = { account: '',BlockNumber: 0 }
20   }
21   render() {
22     return (
23       <div>
24         <p>chainId: {this.state.account}</p>
25         <p>Last block number:{this.state.BlockNumber} </p>
26       </div>
27     );
28   }
29 }
30 }
```

Et voici notre affichage :



VII. . Affichage du nom du token registry de notre contrat et du nombre de tokens créés.

On s'assure d'avoir d'abord créé quelques tokens en utilisant la fonction mint de la question 4 pour vérifier que le compteur s'incrémente. On crée 7 tokens.

- Pour pouvoir utiliser notre contrat déployé sur ganache dans l'application, on récupère **l'ABI** du contrat dans le dossier build/contracts/Token.json et aussi l'adresse du contrat qu'on lit lorsqu'on migre le contrat avec truffle.

```
Replacing 'Token'
-----
> transaction hash: 0x78e461971a3c441fa50198395454357bb51dbb6a375e52a52c47d45e59305ae8
> Blocks: 0 Seconds: 0
> contract address: 0x360216200c6A04337a3632ca64FC3Fe691Ab8186
> block number: 3
> block timestamp: 1575737992
> account: 0xF9edE6e8F655e15dBf69f5F4E4e06F6CcAEdAE99
> balance: 99.93848436
> gas used: 2770018
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.05540036 ETH
```

- Une fois qu'on a l'ABI et l'adresse du contrat, on les stocke dans un fichier config.js situé au même niveau que notre App.js.

```
1
2 export const TOKEN_ADDRESS = '0x360216200c6A04337a3632ca6
3 export const TOKEN_ABI = [
4   {
5     "inputs": [],
6     "payable": false,
7     "stateMutability": "nonpayable",
8     "type": "constructor"
9   },
10  {
11    "anonymous": false,
12    "inputs": [
13      {
14        "indexed": true,
15        "internalType": "address",
16        "name": "owner",
17        "type": "address"
18      },
```

- On importe les données dans notre app.js en ajoutant la ligne :
`import { TOKEN_ABI, TOKEN_ADDRESS } from './config'`

- On récupère les informations de notre contrat pour pouvoir exécuter ses fonctions :

```
const myToken = new web3.eth.Contract(TOKEN_ABI, TOKEN_ADDRESS)

this.setState({ myToken })
const name = await myToken.methods.name().call()
this.setState({ name })
const counter = await myToken.methods._currentId().call()
this.setState({ counter })
```

- On affiche alors le contenu du nom du registre et du compteur de Tokens créés :

```
<p> Registry Name: {this.state.name} </p>
<p> Counter : {this.state.counter} </p>
```

Voilà à quoi ressemble l'affichage pour les deux dernières questions :

