RAPPORT TD4

1. Create a Git repository & share it with the teacher (2 pts)

2. Create a functional ERC721 token contract (2 pts)

Même contrat token ERC721 que le TD3, avec les autres contrats nécessaires à son fonctionnement, tous extraits d'Open Zeppelin, : https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC721/ERC721.sol

3. Create a 'name' public variable to give a name to your token registry, and a counter to show total number of created token (2 pts)

a. Create a 'name' public variable to give a name to your token:

Dans le contrat ERC721 :

- Déclaration de la variable string « name » en public.
- Dans le constructor, attribution du nom de notre token : «myToken » à la variable « name ».

```
string public name; // name token
uint256 public counterToken;

constructor () public {
    name = "myToken";
    counterToken = 0;

    // register the supported interfaces to conform to ERC721 via ERC165
    _registerInterface(_INTERFACE_ID_ERC721);
}
```

b. Create a counter to show total number of created token:

Dans le contrat ERC721 :

- Déclaration de la variable uint256 « counterToken » en public.
- Dans le constructor, initialisation du « counterToken » à 0.

```
string public name; // name token
uint256 public counterToken;

constructor () public {
    name = "myToken";
    counterToken = 0;

    // register the supported interfaces to conform to ERC721 via ERC165
    _registerInterface(_INTERFACE_ID_ERC721);
}
```

- Dans la fonction declareMyToken, incrémentation du « counterToken » à chaque fois qu'un nouveau token est créé (voir partie 4.a. pour la description de la fonction declareMyToken).

```
function declareMyToken(uint _id, address _owner) public {
    //require(isWhitelisted(_msgSender()), "WhitelistedRole: caller does not have the Whitelisted role");
    myToken memory _totalToken;

    _totalToken.id=_id;
    _totalToken.owner=_owner;
    _tokenOwner[_id] = _owner; //mint

    counterToken +=1;
    _id +=1; //token ID incrémenté de 1 à chaque fois

    totalToken.push(_totalToken);
}
```

4. Create a 'mint' function to create new tokens, usable by anyone paying 0.1 ETH (2 pts)

a. Create a 'mint' function to create new tokens

Dans le contrat ERC721 :

- Création de la liste de tous les tokens avec « myToken[] », déclaré en public (myToken est le nom donné à notre token, voir partie 3.a.).
- Déclaration de la structure de notre token « myToken » : une variable uint256 correspondant à l'identifiant « id » du token, et une variable address correspondant à l'adresse du propriétaire « _owner » du token.
- Création d'une fonction declareMyToken qui prend en paramètres l'identifiant et l'adresse du propriétaire du token. Cette fonction permet de créer un nouveau token, et d'affecter à l'adresse du propriétaire du token, l'identifiant de ce nouveau token.

```
myToken[] public totalToken;

struct myToken{
    uint256 id;
    address owner;
}

function declareMyToken(uint _id, address _owner) public {
    //require(isWhitelisted(_msgSender()), "WhitelistedRole: caller does not have the Whitelisted role");
    myToken memory _totalToken;

    _totalToken.id=_id;
    _totalToken.owner=_owner;
    _tokenOwner[_id] = _owner; //mint

    counterToken +=1;
    _id +=1; //token ID incrémenté de 1 à chaque fois

    totalToken.push(_totalToken);
}
```

5. Create a react app (2 pts)

J'ai utilisé la commande npm pour installer les modules nécessaires à la création d'une application ReactJS :

```
C:\Users\Admin\TD4_Ethereum>npm install create-react-app
```

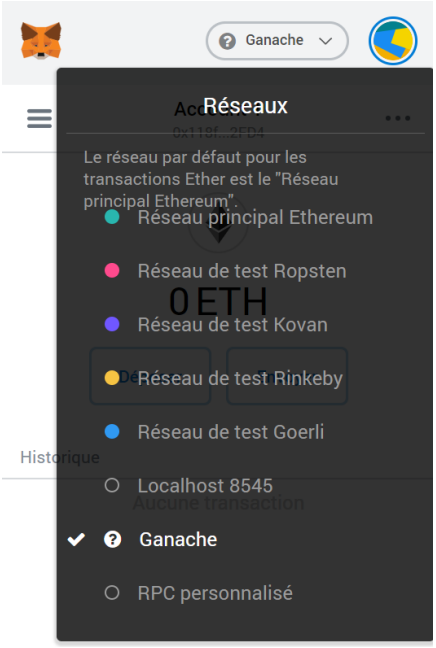
Pour créer une nouvelle application :

```
C:\Users\Admin\TD4_Ethereum>create-react-app Application
```

Pour lancer l'application ReactJS :

```
C:\Users\Admin\TD4_Ethereum>npm start
```

Ensuite, il faut configurer Metamask pour se connecter à Ganache.



On utilise web3 pour interagir avec Ethereum.

```
import Web3 from 'web3'
```

Ensuite, lorsqu'on se connecte à son application ReactJs, on regarde si une extension Ethereum est présente (exemple Metamask), si ce n'est pas le cas on affiche le message d'erreur avec « alert ».

6. Connect your app to the blockchain and display the ChainId and last block number (2 pts)

&

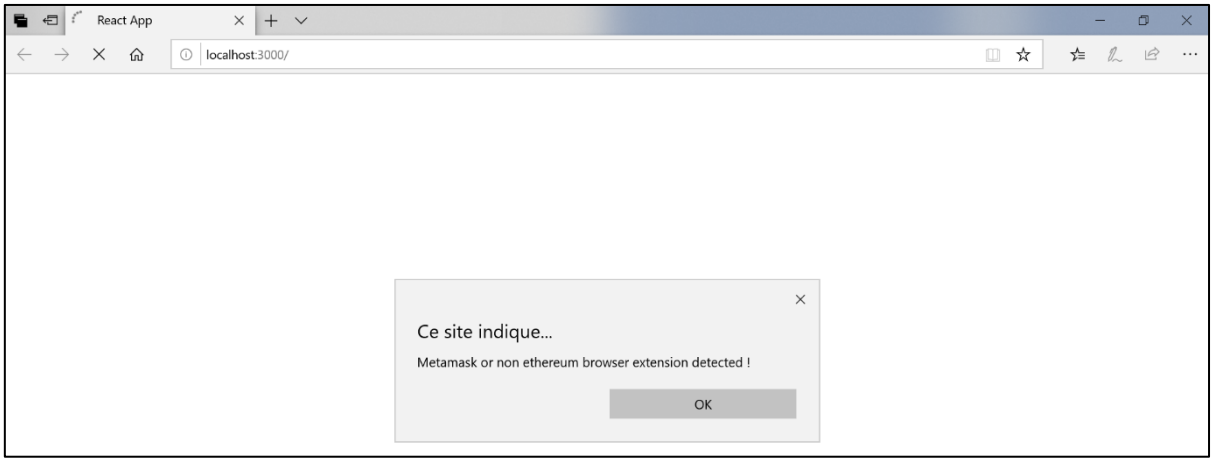
7. Display the token registry name and the total token number in the app (2 pts)

Connection de l'application à Metamask :

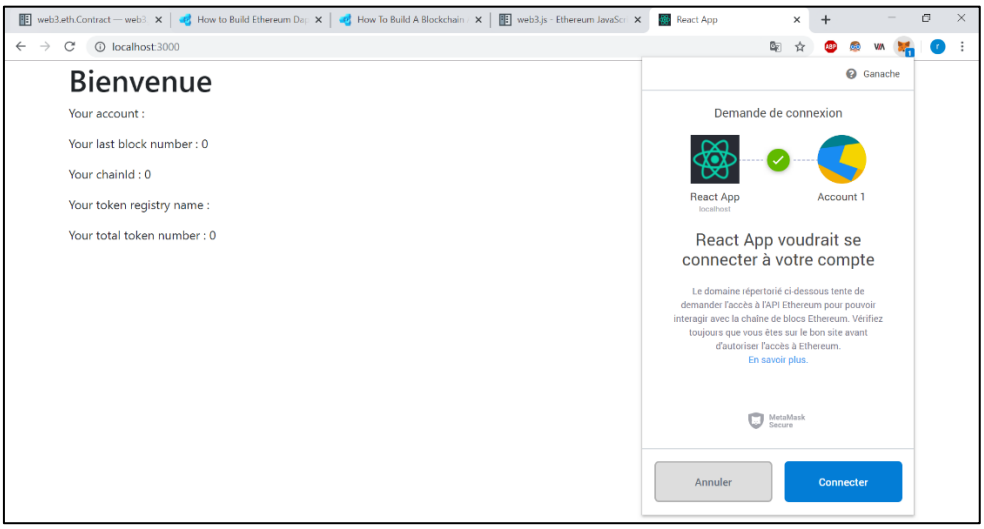
```
// Use Component function to load all blockchain data
class App extends Component {
    async componentWillMount(){
        await this.isEthereumBrowser()
        await this.loadBlockchainData()
    }

    async isEthereumBrowser() {
        if (window.ethereum) {
            window.web3 = new Web3(window.ethreum)
            await window.ethereum.enable()
        }
        else if (window.web3) {
            window.web3 = new Web3(window.web3.currentProvider)
        }
        else {
            window.alert('Metamask or non ethereum browser extension detected !')
        }
    }
```

Résultat :

1) Internet Explorer n'a pas de module Ethereum présent donc message d'erreur :



2) Chrome a un module Ethereum, donc ReactJs demande à Metamask s'il peut accéder à ses informations.



Affichage de « chainId », « lastBlock », « nameToken », « totalTokenNumber » :

Ensuite, nous déployons le contrat de l'application and obtenons les informations du contrat à savoir « chainId », « last_block », « nameToken », « totalTokenNumber ».

- Récupération des adresses de Ganache avec « web3.eth.getAccounts() » puis on récupère la première adresse.
- chainId : utilisation de la fonction « web3.eth.getChainId() »
- Numéro du dernier block : « web3.eth.getBlockNumer() ».

```
async loadBlockchainData() {
    // Connect to the blockchain, passing as well URL to ganache
    const web3 = new Web3(Web3.givenProvider || "http://localhost:7545")
    // Ensuite on recupere le compte qui est actuellement connecté et enregistre l'état de l'objet pour pouvoir suivre l'évolution du composant
    const accounts = await web3.eth.getAccounts()
    this.setState({ account: accounts[0] })
    const lastBlock = await web3.eth.getBlockNumber()
    this.setState({ last_block: lastBlock })
    const chainId = await web3.eth.getChainId()
    this.setState({ chainId: chainId })
    const ERC721 = new web3.eth.Contract(CONTRACT_ABI, CONTRACT_ADDRESS)
    this.setState({ ERC721 })
    const nameToken = await ERC721.methods.name().call()
    this.setState({ nameToken })
    const totalTokenNumber = await ERC721.methods.counterToken().call()
    this.setState({ totalTokenNumber })
}
```

Concernant le nom du token et le nombre total de token, nous devons d'abord déployer le contrat sur l'application.

- Pour cela on utilise la fonction « Contract (abi, address) » qui prend en paramètre l'ABI du contrat à savoir celui présent dans ERC721.json et l'adresse du contrat ERC721 que l'on peut obtenir en faisant :

```
C:\Users\Admin\TD4_Ethereum>truffle console
truffle(development)> ERC721.address
'0x6B2a2D97A04101B8558d06E7E06a058433170C6A'
truffle(development)>
```

- L'ABI et l'adresse du contrat ont été ajoutés dans un fichier externe « config » que l'on importe dans le code de l'application ReactJs « App.js ».

```
export const CONTRACT_ADDRESS = '0x6B2a2D97A04101B8558d06E7E06a058433170C6A'

export const CONTRACT_ABI = [
    {
      "inputs": [],
      "payable": false,
      "stateMutability": "nonpayable",
      "type": "constructor"
    },
    {
      "anonymous": false,
```

Après, cela nous renvoie un objet qui contient toutes les méthodes et variables.

- Nom du token : « ERC721.methods.name().call() » pour récupérer la valeur de la variable « name » dans le contrat ERC721.
- Nombre total de tokens: « ERC721.methods.counterToken().call() » pour récupérer la valeur de la variable contenue dans le contrat.

Ensuite, on doit pouvoir afficher et mettre en forme les valeurs à afficher dans l'application. On le fait grâce à « render() ».

```
render(){
    return (
        <div className="container">
            <h1>Bienvenue</h1>
            <p>Your account : {this.state.account}</p>
            <p>Your last block number : {this.state.last_block}</p>
            <p>Your chainId : {this.state.chainId}</p>
            <p>Your token registry name : {this.state.nameToken}</p>
            <p>Your total token number : {this.state.totalTokenNumber}</p>
        </div>
    );
}
```

Résultat :