TP Blockchain Ethereum - Partie I

- Le support de cours de Léo : Introduction à la Blockchain (../img/blockchain-intro.pdf) (version 1.0 2019)
- Le support de cours de Jp : Introduction à Ethereum (../img/ethereum-intro-cm-jpgelas.pdf) (version 1.1 2021)

Premières transactions

Objectif: Créer son premier Wallet et réaliser ses premières transactions d'ETH sur un réseau de test.

- Installer l'extension Metamask dans votre navigateur (Chrome ou Firefox)
- Créer un compte et copier sur une feuille (ou sauvegarder) les douze mots.
- Connecter Metamask au réseau de test **Ropsten** .
- Récuperer 1.0 ETH via le *faucet*.
- Surveiller via un explorateur de blockchain (etherscan) la confirmation de la transaction.
- Envoyer 1 ETH au faucet. Pourquoi la transaction est rejetée ?
- Envoyer 0.1 ETH à un autre compte vous appartenant. Vous devrez le créer au préalable.
- Envoyer 0.1 ETH vers le compte d'un camarade.

Remarque : Si vous constatiez que les transactions prennent trop de temps à être validée (plus de 1 ou 2 minutes) n'hésitez pas à utiliser un réseau de test alternatif à *Ropsten* comme par exemple **Kovan**, **Rinkeby** ou même **Görli**.

Travail à rendre :

1. Lorsque vous avez envoyé 1 ETH pourquoi la transaction a t'elle été rejetée ou bien le destinataire n'a pas reçu exactement 1 ETH ?
2. Combien coûte en GAS puis en Ether (ou GWei) l'envoie d'Ether (transaction simple d'ETH d'un compte à l'autre) ?
3. Les frais en GAS sont-il proportionnel à la quantité d'Ether envoyé ?
4. Quel est la particularité de *Ropsten* ?

Premières interaction avec la Blockchain Ethereum

Objectif: Faire sa première requête sur le *mainnet*.

- Se créer un compte sur infura (https://infura.io/) (sauvegardez votre identifiant)
- Faire une requête *curl* permettant de connaitre le numéro du dernier bloque miné sur le *mainnet*.

Liens utiles : https://eth.wiki/json-rpc/API#eth_blocknumber ()

Travail à rendre :

1. Quel est le numéro du dernier bloque miné en **décimal** sur le *mainnet* (notez la date et l'heure de relevé) ?

Premier Smart Contract

Objectif: Compiler et déployer un premier Smart Contract sur un réseau de test.

- Copier/coller le Smart Contract (Hello.sol) en Solidity fournit ci-dessous dans l'IDE Remix (https://remix.ethereum.org ()).
- Compiler et déployer le Smart Contract sur le réseau *Ropsten* (Injected Web3 / Ropsten).
- Combien a couté le déploiement du contrat ? Sa taille influe t-elle sur le coût ?
- Retrouver l'adresse du contract sur Etherscan (ropsten) (http://ropsten.etherscan.io). Quel est la démarche ?
- Essayer de modifier la chaine de caractères *message*. Combien cela coûte t-il ?
- Écrire un contrat similaire à *Hello.sol* pour gérer cette fois-ci un entier `uint256`.

```
pragma solidity ^0.5.7;

contract Hello {
  string private message;
  constructor(string memory _message) public {
    message = _message;
  }
  function getMessage() public view returns (string memory) {
    return message;
  }
  function setMessage(string memory _message) public {
    message = _message;
  }
}
```

Smart Contract simple

Objectif: Création d'un premier Token basé sur un contrat simple.

- Lire, comprendre et déployer sur un réseau de test (*Ropsten*) le contrat ci-dessous.
- Qui est le propriétaire initial des tokens ?
- Transférer quelques tokens d'un compte vers un autre compte.
- Visualiser le nombre de tokens appartenant à chacun des comptes sur www.myetherwallet.com (https://www.myetherwallet.com) (le nombre de décimal est 0) ou MyCrypto.com (https://mycrypto.com).

```
pragma solidity >=0.4.22 <0.6.0;

contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    constructor( uint256 initialSupply ) public {
        balanceOf[msg.sender] = initialSupply;
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
        return true;
    }
}
```

TP Blockchain Ethereum - Partie II

Introduction aux *Events*

> Dans un Smart Contract, une fonction appelée peut émettre un événement (*event*). Les *events* en plus d'être inscrit dans la blockchain, peuvent être récupéré par un *Listener* dans l'application en *front-end*. Cela permet par exemple de notifier l'utilisateur qu'une opération à bien eu lieu.

Pour les travaux qui suivent nous allons nous appuyer sur la librairie *ethers.js* (au lieu de *web3.js*). L'installation se fait à l'aide de la commande `npm i ethers` (Attention au proxy Lyon 1 : `npm config set proxy "http://proxy.univ-lyon1.fr:3128"`).

Programmation asynchrone

Pour pouvoir mieux appréhender la suite des exercices proposés il est important de comprendre la programmation en mode asynchrone de JavaScript (*Promise*, *async/await*). Voici un premier exemple qui permet de récupérer le numéro du dernier bloc miné. Ce bout de code peut être exécuté directement dans la console de *node* ou avec la commande *node* suivit du *nom du fichier* javascript.

```
const ethers = require('ethers');
const provider = ethers.getDefaultProvider(); // homstead (mainnet)

provider.getBlockNumber()
        .then(blockNumber => {
                console.log("Block number : " + blockNumber);
        });
```

ou avec le support de *Infura*

```
const ethers = require('ethers');
const provider = ethers.providers.InfuraProvider('homestead','64da4byourprojectid.....');

provider.getBlockNumber()
        .then(blockNumber => {
                console.log("Block number : " + blockNumber);
        });
```

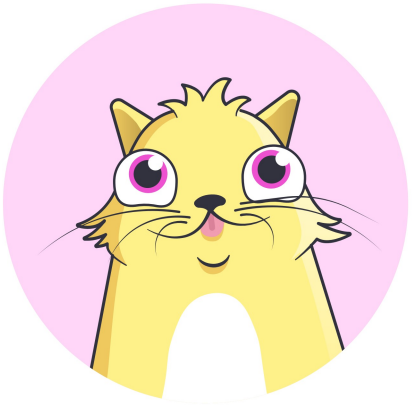
Observons l'évolution de la blockchain

La première étape consiste a observer en "temps réel" la création de nouveaux blocs dans la blockchain *homestead* (mainnet). L'apparition d'un nouveau bloc est un *event* en soi. Soit le fichier `index.js`.

```
const ethers = require('ethers');
const provider = ethers.getDefaultProvider(); // homstead (mainnet)

provider.on('block', (blockNumber) => {
    console.log('-- New Block: ' + blockNumber);
});
```

Observons la naissance des chats (CK)



Il existe un Smart Contract (ERC-721) à l'addresse `0x06012c8cf97BEaD5deAe237070F9587f8E7A266d` qui permet de collectionner des *CryptoKitties* (chat électronique). L'application frontal (https://www.cryptokitties.co/ ()) permet aux utilisateurs d'en acheter, d'en vendre (spéculation), ou de les faire se reproduire afin de créer de nouvelles générations de "chats". Le code source du Smart Contract qui permet cela est disponible à l'adresse 0x06012c8cf97bead5deae237070f9587f8e7a266d (https://etherscan.io/address/0x06012c8cf97bead5deae237070f9587f8e7a266d#code).

Si vous parcourez le code Solidity de ce Smart Contract (lignes 226 et 415) vous observerez qu'un évenement `Birth` est émis à chaque fois qu'un "chat" naît. Le code source JavaScript suivant permet d'observer la naissance des chats en temps réel.

```
const ethers = require('ethers');
const provider = ethers.getDefaultProvider(); // homstead (mainnet)

let abi = [
    "event Birth(address owner, uint256 kittyId, uint256 matronId, uint256 sireId, uint256 genes)"
];
let contractAddress = "0x06012c8cf97BEaD5deAe237070F9587f8E7A266d";
let contract = new ethers.Contract(contractAddress, abi, provider);

contract.on("Birth", (owner, kittyId, matronId, sireId, genes , event) => {
    console.log("# block # : " + event.blockNumber);
    console.log("# Owner : "   + owner);
    console.log("# kittyId : " + kittyId);
});
```

Vous devrez peut être patienter quelques longues minutes avant qu'un "heureux événement" ne se produise. Ensuite, avec l'addresse du propriétaire et `etherscan.io` vous devriez pouvoir observer la transaction sur la blockchain, son coût et surtout à quoi ressemble le "chat" nouveau-né (cf. onglet *Inventory*).

Travail facultatif :

1. Écrivez un bout de code qui affiche le numéro de chaque nouveau bloc miné, et
2. si ce bloc contient la naissance d'un chat, affichez l'adresse Ethereum de son propriétaire et l'Id du chaton.

A vous d'observer ! (ERC-20)

Les contrats ERC-20 définissent un événement `Transfer(address indexed _from, address indexed _to, uint _value)` appelé par toutes les fonctions de transfert de Token d'un compte à un autre (i.e. `transfer(...)` et `transferFrom(...)`).

Choisissez le Token de votre choix (ex: BAT, ZRX, ...) et en vous inspirant des exemples précédents, écrivez un bout de code Javascript qui affiche le nombre de Token transmis (value) entre les comptes en temps réel.

Travail à rendre :

1. Écrivez un bout de code qui affiche le numéro de chaque nouveau bloc miné
2. Si ce nouvea bloc contient une transaction du Token que vous avez choisi profitez en pour afficher les adresses de l'émetteur et du récepteur ainsi que la valeur transmise.

3. Pourquoi pour ce second TP nous n'avons pas eu besoin de récupérer des *Ethers* au préalable ?

Travail de réflexion :

Comparez le coût en unité de GAS de l'OP_CODE **SSTORE** et de l'OP_CODE **LOG** (utilisé par la fonction *emit* d'un événement) :

1. Qu'est ce que cela vous inspire ?
2. Quelle est la limitation ?

Projet Contrat de Mariage (A RENDRE)

Projet à faire en **binôme** evidemment :-)

Ce projet est à réaliser en deux étapes:

1. La première étape consiste à concevoir et rédiger les Smart Contract nécessaires pour officialiser à tout jamais l'union de deux personnes (sur la blockchain Ropsten).
2. La seconde étape consiste à concevoir une interface web utilisateur (simple) pour qu'un couple puisse (a) soumettre leurs prénoms et récupérer en échange de quelques fractions d'Ether l'adresse du Smart Contract dans lequel leurs prénoms sont gravés à jamais. (b) Soumettre l'adresse de leur Smart Contract pour vérifier que leur prénoms ont bien été inscrits dans la blockchain.

Le vidéo ci-dessous illustre le sujet de ce projet.

0:00 / 0:48

Hints : J'ai rédigé un premier Contrat (WeddingCertificate) qui me permet de stocker le nom des deux partenaires, d'initialiser ces deux champs avec le constructeur, et une fonction accesseur qui me retourne les deux noms. Puis j'ai rédigé un second contrat (WeddingCertificateFactory) qui me permet de générer des *WeddingCertificate* lorsqu'un couple en fait la demande.

Travail à rendre le MARDI 1 JUIN 2021 (au soir) :

1. Les codes sources de votre application.
2. Au choix une démo faite en salle de TP ou une URL qui pointe vers un *screencast* de votre démo.

TP Blockchain Ethereum - Partie III

ENS (Ethereum Name Service)

> Objectif : Ce TP propose d'aider à comprendre le principe de fonctionnement du service de nommage d'Ethereum. Pour obtenir la version *hashée* des noms de domaines nous avons mis en ligne l'outil nécessaire sur codepen (https://codepen.io/jpgelas/pen/pBOxry).

Enregistrer un domaine *.test*

La première étape consiste à trouver (https://docs.ens.domains/ens-deployments) le contrat déjà déployé sur le testnet *Ropsten* qui nous permettra :

1. de vérifier que notre nom de domaine choisi sur *.test* est libre,
2. d'enregistrer notre nom de domaine.

> Le registre sur Ropsten (https://ropsten.etherscan.io/address/0x112234455c3a32fd11230c42e7bccd4a84e02010) est déployé à l'adresse : `0x112234455C3a32FD11230C42E7Bccd4A84e02010`

- Trouver le resolver de .test avec le *namehash* (full domain) (https://docs.ens.domains/contract-api-reference/name-processing) (aussi appelé node name) de "test" (0x04f740db81dc36c853ab4205bddd785f46e79ccedca351fc6dfcbd8cc9a33dd6) sur la fonction `owner()` du contrat.
- Celui-ci devrait vous renvoyer : 0x21397c1A1F4aCD9132fE36Df011610564b87E24b (https://ropsten.etherscan.io/address/0x21397c1A1F4aCD9132fE36Df011610564b87E24b)
- Allez dans la section *Read contract* du contrat ci-dessus, et appelez `expiryTimes()` avec le *labelhash* de votre nom de domaine (ex: jpgelas => 0x38598a04245de05ed5da8546c3b31c8cb4003867b0542f0518b15de36471b042).

> **REMARQUE:** : Le `expiryTimes` peut être non-nul mais le domaine peut quand même être disponible. Les `.test` sont "verrouillés" pour 28 jours seulement, si `expiryTimes < Date.now()` le domaine est disponible.

Seconde étape : Register your name (0x21397c1...)

- Dans la section *write contract*, soumettez le *labelhash* de votre subdomain de *.test* (ex: jpgelas => 0x38598a04245de05ed5da8546c3b31c8cb4003867b0542f0518b15de36471b042) et votre addresse Ethereum (cf. Metamask) dans la fonction `register()`.
- Dans la section *Read contract*, appelé `expiryTimes()` avec le namehash de votre nom de domaine. Si non nul c'est que cela a fonctionné.
- Vérifiez que 'owner()' du registrar (0x1122344...) renvoit bien votre adresse Metamask quand vous lui passé le *namehash* de votre domaine (ex: jpgelas.test => 0xc631d23fa0668d018e25d5fee3c9b3930c394b8a40a2a4637006d4f9ee4c959c)

A présent nous sommes propriétaire d'un nom (ex: `jpgelas.test`). Nous allons donc lui associer un resolveur. Nous allons en utiliser un existant mais nous pourrions déployer notre propre résolveur.

> Un resolveur publique disponible sur Ropsten (https://ropsten.etherscan.io/address/0x4c641fb9bad9b60ef180c31f56051ce826d21a9a#writeContract) est déployé à l'adresse : `0x4c641fb9bad9b60ef180c31f56051ce826d21a9a`
>
> **REMARQUE** : Si l'interface de etherscan.io ne vous propose pas d'accéder aux fonctions du contrat vous pouvez utiliser les services de myetherwallet.com (https://myetherwallet.com) > contract > ENS Registry ou bien de mycrypto.com (https://mycrypto.com/contracts/interact).

Plus que deux étapes :

- Dans le registry (0x112234455...)
  - Appelez la fonction `setResolver(namehash, resolverAddress)`
- Dans le resolver (0x4c641f...)
  - Appelez `setAddr(namehash, myOwnAddress)`

Étape ultime : Envoyez de l'ether via un nom de domaine :)

Pour allez plus loin :

1. Créez des sous domaines à votre domaine.
2. Configurez la résolution inverse (reverse resolution)