RAPPORT TRAVAUX PRATIQUE – SUJET 1

Charité et Blockchain

Étape 1 – état des lieux

Logiciels utilisés :

- PhpStorm
- Postman

Récupération du projet avec : git clone https://github.com/laurentgiustignano/hachage-tp1

Lancement du projet :

- Lancer le fichier server.js
- http://localhost:3000/blockchain

Test de fonctionnement a effectué :

Pour vérifier le bon fonctionnement de l'environnement de travail on vérifie si les requêtes GET et POST fonctionne correctement :

Code mis en place :

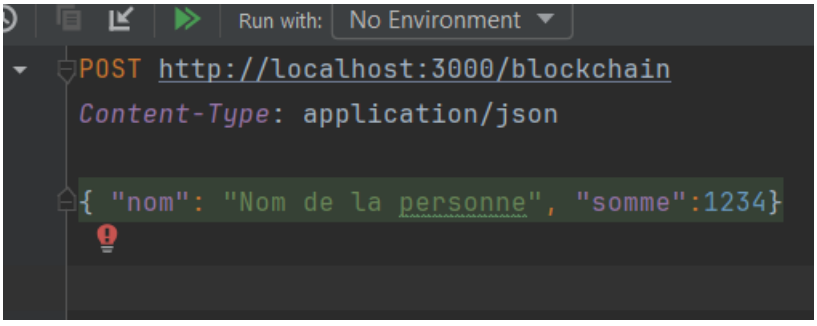
```
switch (endpoint) {
    case 'GET:/blockchain':
        results = await liste(req, res, url)
        console.log('test');
        break
    case 'POST:/blockchain':
        results = await create(req, res)
        console.log('test2');
        break
    default :
        res.writeHead(404)
}
```

résultat obtenu dans la console :

```
test
test2
```

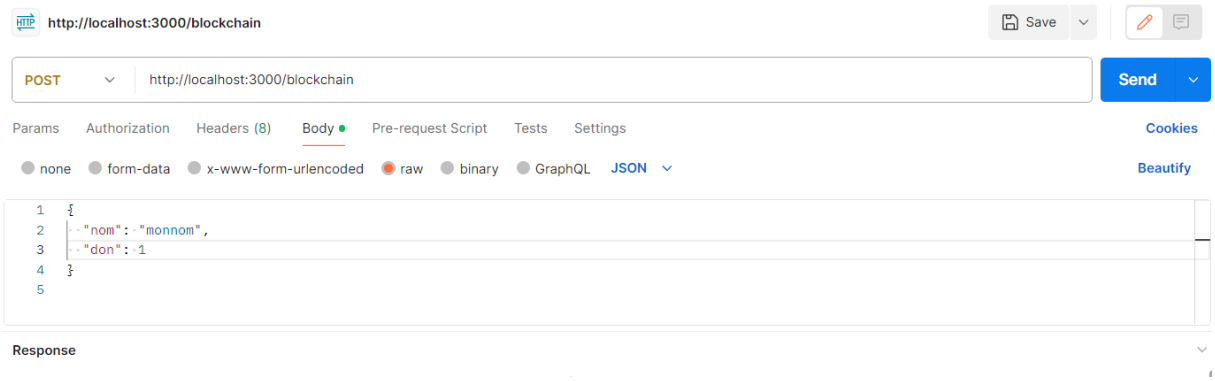
Pour effectuer le post il y a deux façons :

La première directement via PhpStorm en allant dans l'onglet Tools-> HTTP Client -> Create Request in HTTP Client.



La seconde à l'aide de l'outil Postman.

On entre l'url on se met en POST, Body et raw et on entre notre contenu Json



Étape 2 – apprenons à lire

Tout d'abord j'ai créer un fichier blockchain.json dans un dossier data

J'ai ensuite développé la fonction findBlocks qui retourne tous les Blocks. Après mettre renseigner sur la syntaxe et sur la fonction readFile() je n'ai pas eu trop de mal à faire cette fonction.

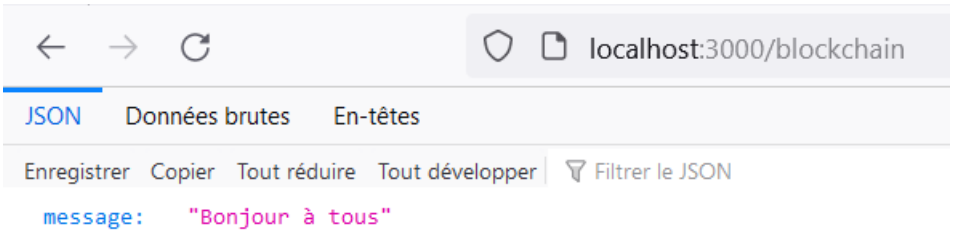
```
export async function findBlocks() {
    try {
        const response = await readFile(path);

        return new Promise( executor: (resolve, reject) => {
            if (response.length === 0) {
                resolve( value: []);
            } else {
                resolve(JSON.parse(response));
            }
        });
    } catch (err) {
        return Promise.reject( reason: []);
    }
}
```

Cette fonction findBlocks est appelée dans la fonction liste qui est elle-même appelé dans le fichier sever.js.

```
export async function liste(req, res, url) {
    return findBlocks()
}
```

```
case 'GET:/blockchain':
    results = await liste(req, res, url)
    //console.log(results)
    break
```

Et voici le résultat de ces opérations :



Étape 3 – une brique après l'autre

La fonction uuidv4() est appelé après avoir importer le module à l'aide de npm

```
export async function createBlock(contenu) {
    const obj : {…}  = {
        id: uuidv4(),
        nom: contenu.nom,
        don: contenu.don,
        date: getDate(),
    }
```

Il a ensuite fallu mettre en place la fonction getDate().

Pour cette fonction j'ai utilisé ChatGPT et je l'ai adapté pour obtenir la date sous le format voulu.

```
export function getDate() : string  {
    const now : Date  = new Date();

    const year : number  = now.getFullYear();
    const month : string  = String( value: now.getMonth() + 1).padStart(2, '0');
    const day : string  = String(now.getDate()).padStart(2, '0');

    const hours : string  = String(now.getHours()).padStart(2, '0');
    const minutes : string  = String(now.getMinutes()).padStart(2, '0');
    const seconds : string  = String(now.getSeconds()).padStart(2, '0');

    return `${year}${month}${day}-${hours}:${minutes}:${seconds}`;
}
```

```
20240125-15:54:58
```

Maintenant que ces deux fonctions sont prêtes à être utilisé je peux coder la fonction createBlock(contenu). Cette fonction a été longue à faire à cause de ma nécessité à devoir me documenter de plein de petites erreurs et de quelques fausses pistes sur Internet.

```
export async function createBlock(contenu) {
    const obj :{…}  = {
        id: uuidv4(),
        nom: contenu.nom,
        don: contenu.don,
        date: getDate(),
    }
    try{
        const blockAll = await findBlocks() || [];

        const arr :(…)[]  = [blockAll, obj];
        const json :string  = JSON.stringify(arr, replacer: null, space: 2);
        await writeFile(path, json);
        return new Promise( executor: resolve => resolve(arr));
    }catch(err){
        console.log(err);
```

Et voici le résultat :

Je peux ajouter des Blocks lorsque j'utilise Post.


```
▼ 0:
    id:     "3d7cd843-a7ce-4844-b013-d1fbcd62b09e"
    nom:    "test"
    don:    5
    date:   "20240125-16:23:06"
▼ 1:
    id:     "cae6fab5-a809-4d3c-9bca-ba5dc2b88653"
    nom:    "exemple"
    don:    "quelquechose"
    date:   "20240129-11:16:42"
```


Et Voici à quoi cela ressemble dans server.js :

```
case 'POST:/blockchain':
    results = await create(req, res)
    console.log(`POST : ${JSON.stringify(results)}`);
    break
default :
```

Étape 4 – « Vers l'infini et au-delà ! »

La fonction findLastBlock() n'a pas été très dure à faire car se qu'il fallait faire me paraissait assez intuitif, il fallait juste trouver la bonne syntaxe.

```
export async function findLastBlock() {
    const blocks = await findBlocks();
    const lastBlock : null | any  = (blocks === null || undefined) ? null : blocks[blocks.length - 1];
    return new Promise( executor: resolve => resolve(lastBlock));

}
```

```
POST : [[[[[[{"id":"3d7cd843-a7ce-4844-b013-d1fbcd62b09e","nom":"test","don":5,"date":"20240125-16:23:06"},{"id":"cae6fab5-a809-4d3c-9bca-ba5dc2b88653","nom":"exemple","don":"quelq
echose","date":"20240129-11:16:42"}],{"id":"f48db790-1668-4542-b583-360c46143a61","nom":"toto","don":200,"date":"20240129-11:17:42"}],{"id":"18b447d2-8632-4ff7-bf83-dc8de5a49d72","
om":"toto","don":200,"date":"20240129-11:26:10"}],{"id":"ad4aec74-f1b0-4736-b002-9fac189bed7e","nom":"dernier ajout","don":2000000,"date":"20240129-11:26:47"}],{"id":"fc08db33-3db9
4093-9e7a-b5fa25dcbe41","nom":"toto","don":200,"date":"20240129-11:27:47"}],{"id":"0589b79e-7f26-4f75-9d21-f8b69153e650","nom":"dernier ajout","don":2000000,"date":"20240129-11:27-
7"}]
dernier ajout : {"id":"0589b79e-7f26-4f75-9d21-f8b69153e650","nom":"dernier ajout","don":2000000,"date":"20240129-11:27:57"}
```

Conclusion :

J'ai eu un peu de mal au démarrage car hormis pour la SAE je n'avais jamais codé en javascript, mais il existe plein de documentation sur cette technologie qui m'ont permis de surmonter mes problèmes (principalement de syntaxe). L'installation de l'environnement de travail ainsi que le déroulé du TP était claire et facile à comprendre.

J'ai fait quelques tentatives pour les fonctions en plus mais je n'ai pas réussi à les faire fonctionner.