TP Sujet 1 Charité et Blockchain

Nom : GUICHARD
Prénom : Julien
Groupe : 301

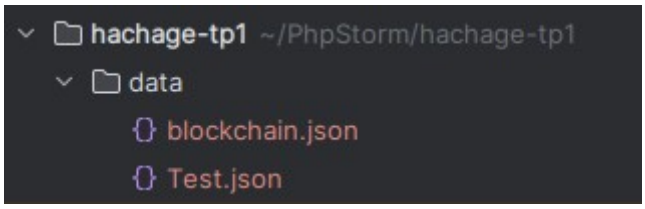
Première etape
Récupération du projet sur github à l'adresse suivante :
https://github.com/laurentgiustignano/hachage-tp1

Ensuite il est demandé de regarder la structure du code server.js. Ce code source permet de lancer la fonction createServer directement grâce à nodeJS sur le port 3000.

Cette fonction à deux entrée une pour le GET et l'autre pour le POST pour le chemin « /blockchaine ».

J'ai rajouter des console.log pour le post et le get afin de vérifier en « debug » le résultat retourné en cas de problème.
Suivant le résultat la fonction **res.write(*JSON*.stringify(results))** permet de visualiser le retour sous postman du résultat au format d'une chaîne de caractère et l'afficher.

Deuxième étape
Création d'un répertoire data dans le projet :

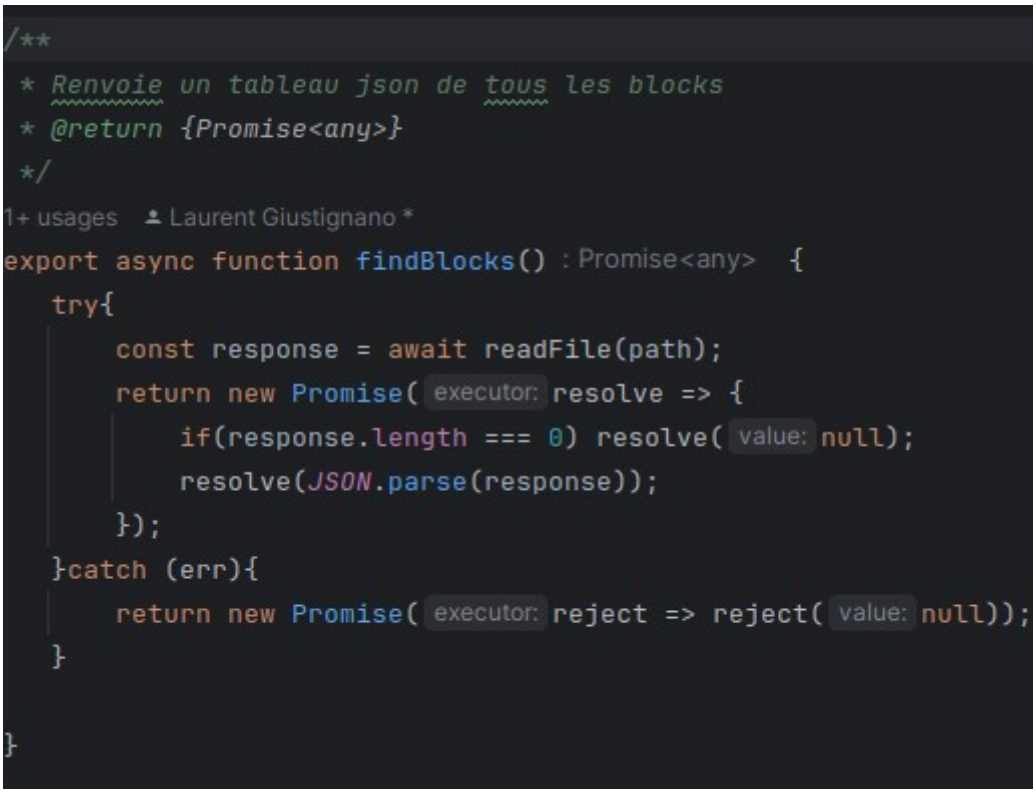


Le fichier blockchain.json à été créé pour les besoins du projet.
J'ai créé un autre fichier Test.json afin de pouvoir tester les checkSum et faire des copier/coller pour les tests.

En ce qui concerne le chemin pour accéder au fichier j'ai eu des problèmes sur linux donc j'ai mis le chemin absolu car le chemin relatif
« ../data/blockchain.json » me revoyait l'erreur suivante :

```
[Error: ENOENT: no such file or directory, open '../data/blockchain.json'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: '../data/blockchain.json'
}
```

Codage de la fonction findBlocks() :



```
/**
 * Renvoie un tableau json de tous les blocks
 * @return {Promise<any>}
 */
1+ usages   Laurent Giustignano *
export async function findBlocks() : Promise<any>  {
    try{
        const response = await readFile(path);
        return new Promise( executor: resolve => {
            if(response.length === 0) resolve( value: null);
            resolve(JSON.parse(response));
        });
    }catch (err){
        return new Promise( executor: reject => reject( value: null));
    }

}
```

On utilise un try catch pour gérer les erreurs liés à la lecture du fichier.
On vérifie que le fichier n'est pas vide sinon la promise sera à nulle, si il n'y a pas d'erreur on renvoie un objet javascript via JSON.parse.

Troisième étape

Pour la création du bloc « createBlock » il faut installer le paquet uuidv4 avec la commande « npm install uuidv4 » et mettre l'import comme dans le lien suivant : https://www.npmjs.com/package/uuidv4
Dans cette fonction on a besoin d'un objet javascript pour créer la structure avec :

- l'id (générer grâce à uuidv4 qui sera unique)
- le nom (nom donnée)
- le don (le montant du don)
- la date (la date générer au moment de la création)

Ensuite on récupère cette objet et on le concatène avec tous les blocs pour l'enregistrer dans le fichier blockchain.json.
La fonction JSON.stringify(arr, null, 2) permet de formater le json dans le fichier blockchain.json et on retourne une promise avec tous les blocs à postman.

```
/**
 * Creation d'un block depuis le contenu json
 * @param contenu
 * @return {Promise<Block[]>}
 */
1+ usages  Laurent Giustignano *
export async function createBlock(contenu) : Promise<...>  {
    const obj : {...}  = {
        id: uuidv4(),
        nom: contenu.nom,
        don: contenu.don,
        date: getDate(),
    }
    try{
        const isOk : boolean  = await verifBlocks();
        console.log("Check blocks : ", isOk);
        const blockAll = await findBlocks();
        obj.hash = hashMe(await findLastBlock(), obj);
        const arr : ... | ...  = (blockAll === null || blockAll === undefined) ? [obj] : [...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);
    }
}
```

Dans le fichier src/divers.js on doit créer la fonction getDate

```
/**
 * @description Retourne un timestamp au format aaaammjj-hh:mm:ss
 * @return {string}
 */
1+ usages  Laurent Giustignano *
export function getDate() : string  {
    const d : Date  = new Date();
    const mm : string | number  = (d.getMonth() + 1 < 10) ? "0" + (d.getMonth() + 1) : d.getMonth() + 1;
    const dd : string | number  = (d.getDate() < 10) ? "0" + d.getDate() : d.getDate();
    const yyyymmdd = d.getFullYear() + mm + dd;
    const time : string  = d.getHours() + ':' +
        ((d.getMinutes() < 10) ? "0" + d.getMinutes() : d.getMinutes() )+ ':' +
        ((d.getSeconds() < 10) ? "0" + d.getSeconds() : d.getSeconds());
    return yyyymmdd + '-' + time;
}
```

Lors de la création de la date il faut vérifier si les minutes sont inférieur à dix et de même pour les secondes pour rajouter un zéro devant ; car ces deux fonctions retourne un nombre donc pour un chiffre inférieur à dix on aura pas 0X mais juste X comme nombre.

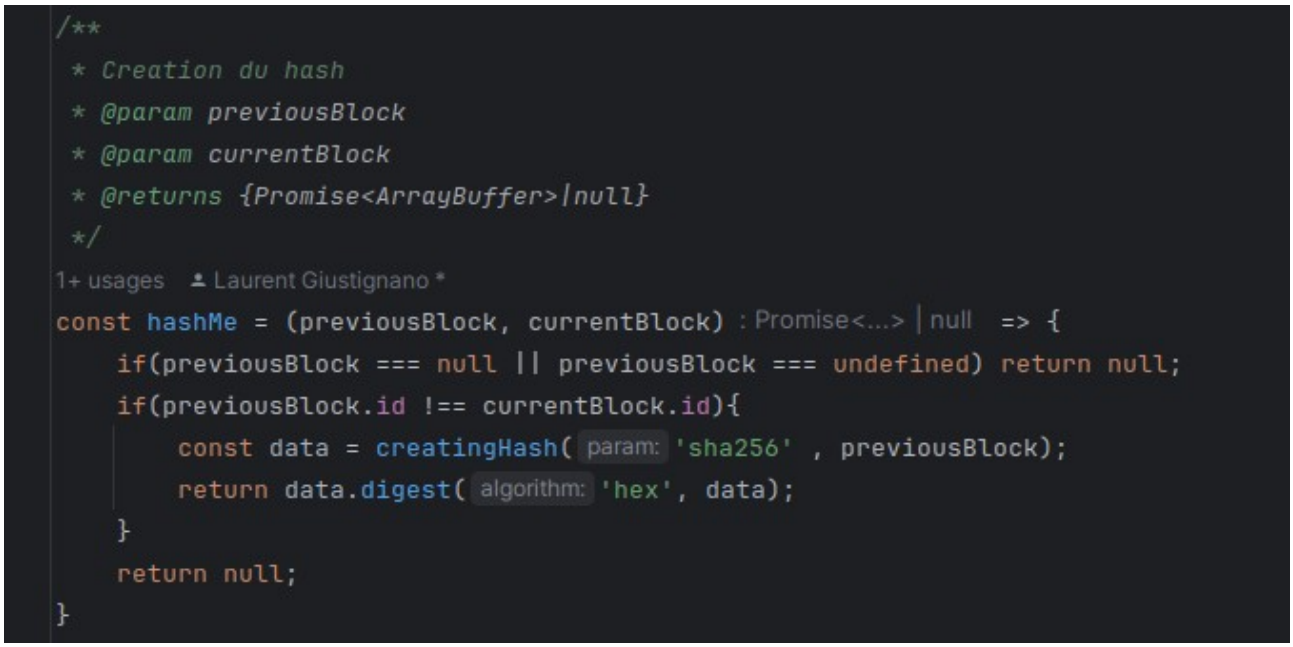
<u>Quatrième étape</u>

Création de la fonction findLastBlock.
Cette fonction va retourné le dernier élement grâce à la fonction findBlocks et on retourne ce bloc via une promise.

```
/**
 * Trouve le dernier block de la chaine
 * @return {Promise<Block|null>}
 */
1+ usages  Laurent Giustignano *
export async function findLastBlock() : Promise<...>  {
    const blocks = await findBlocks();
    const lastBlock : null | any  = (blocks === null || undefined) ? null : blocks[blocks.length - 1];
    return new Promise( executor: resolve => resolve(lastBlock));

}
```

Pour l'encodage j'ai crée une fonction hashMe



```
/**
 * Creation du hash
 * @param previousBlock
 * @param currentBlock
 * @returns {Promise<ArrayBuffer>|null}
 */
1+ usages  Laurent Giustignano *
const hashMe = (previousBlock, currentBlock) : Promise<...> | null  => {
    if(previousBlock === null || previousBlock === undefined) return null;
    if(previousBlock.id !== currentBlock.id){
        const data = creatingHash( param: 'sha256' , previousBlock);
        return data.digest( algorithm: 'hex', data);
    }
    return null;
}
```

Cette fonction va récupérer le block précédent pour la création d'un sha256 dans le bloc courant.

<u>Cinquième étape</u>

On parcours tous les éléments à partir du deuxième élément et on récupère l'élément précédent afin de vérifier si le sha256 et correcte.
On va se servir d'une autre fonction creatingHash qui va générer le hash.

```
/**
 * Vérification de tout les block json
 * @return {boolean}
 */
1+ usages  new *
const verifBlocks = async () : boolean  => {
    const blockAll = await findBlocks();
    for(let i : number  = 1; i < blockAll.length; ++i){
        const previousBlock = blockAll[i - 1];
        const hashData = creatingHash( param: 'sha256', previousBlock);
        const value : Promise<ArrayBuffer>  = hashData.digest( algorithm: 'hex', hashData);
        if(value !== blockAll[i].hash) {
            const dataTest : Block[]  = await findBlock(blockAll[i]);
            console.log("Block error", dataTest);
            return false;
        }
    }
    return true;
}
```

Fonction de création du hash

```
/**
 * Creation du hash
 * @param param
 * @param block
 * @returns {any}
 */
1+ usages  new *
const creatingHash = (param, block) => {
    return createHash(param).update(JSON.stringify(block));
}
```

La dernière fonction à coder findBlock(id) on va parcourir tous les blocs et trouver le bloc précédent grâce à la fonction find. Ensuite on vérifie si le hash n’est pas correcte alors on retourne le bloc au format json dans une promise sinon on retourne nulle.

```
/**
 * Trouve un block à partir de son id
 * @param partialBlock
 * @return {Promise<Block[]>}
 */
1+ usages  Laurent Giustignano *
export async function findBlock(partialBlock) : Promise<...>  {
    const blocks = await findBlocks();
    let prev : undefined  = undefined;
    blocks.find(f => {
        if(f.id !== partialBlock.id) prev = f;
        else return f;
    });

    return new Promise( executor: resolve => {
        const hashData = creatingHash( param: 'sha256', prev);
        const value : Promise<ArrayBuffer>  = hashData.digest( algorithm: 'hex', hashData);
        if(value !== prev.hash){
            return resolve(partialBlock);
        }else{
            return resolve( value: null);
        }
    });

}
```