Rapport développement avancé

Exercice 1 :

Pour pouvoir vérifier que les requêtes se réalisent, on a besoin d'une trace que ce soit par une réponse ou par une entrée dans la console. Pour éviter de se compliquer la tâche et simplement vérifier que les requêtes fonctionnent j'ai ajouté un console.log() pour chacune des requêtes.

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

J'ai ensuite lancé le serveur ainsi que postman pour réaliser les requêtes get et post sur le serveur et vérifier que les messages apparaissent bien sur la console.

Exercice 2 :

Après la création du dossier data et du fichier blockchain.json ainsi que l'attribution de la constante path à '../data/blockchain.json' qui est le chemin relatif du script blockchainStorage.js au fichier de stockage de données blockchain.json, j'ai pu coder la fonction findBlocks().

Pour cela je me suis appuyé sur la documentation de la fonctions readFile() de la bibliothèque node:fs/promises en créant alors d'abord une URL à partir de la constante path initialisé auparavant puis en appelant la méthode readFile pour récupérer le contenu du fichier json en String. Afin de renvoyer le résultat à la requête il faut d'abord convertir le String en format json, afin de faire cela je m'appuie sur la méthode JSON.parse() afin de faire la conversion, je retourne ensuite le résultat de cette conversion.

Il existe un cas où le json.Parse ne fonctionnera pas, si le fichier de sauvegarde est vide l'argument de la fonction sera alors une chaîne de caractère vide. La fonction ne trouvera alors pas de contenu JSON dans le string et retournera une erreur. Afin d'éviter cela j'ai vérifié que le résultat de la lecture de fichier soit vide ou non, si le fichier est vide le résultat retourné sera une liste vide ce qui aidera pour la réalisation de future fonctionnalité.

```
[
    {
        "id": "29a6a120-6b57-4c39-85e0-db0522131d3f",
        "date": "20240123-16:52:21",
        "nom": "ilo",
        "don": 1,
        "hash": null
    },
    {
        "id": "d9917ef3-f491-477d-90ac-618ceb3da792",
        "date": "20240123-16:52:29",
        "nom": "akash",
        "don": 15,
        "hash": "7824eabc0d2c3083e13010214e280713db7586f659c45d4b51ef67e42948027d"
    }
]
```


Réaliser la requête GET renvoie maintenant un résultat JSON, le résultat vu ici correspond à la version après finalisation de l'étape 4.

Exercice 3 :

Afin de créer un bloc, on aura besoin de deux données fournies par la requête, le nom et le don. L'id ainsi que la date seront générés directement dans la fonction. Pour récupérer le nom et le don rien de plus simple que de récupérer l'information directement dans le json fournit par l'utilisateur, une implémentation de la vérification des données d'entrées serait préférable, que ce soit que ce qui est fourni est bien un json et que le nom soit bien un string et le don un entier.

```
const id : string  = uuid();
const date : string  = getDate();
const nom = contenu.nom;
const don = contenu.don;
```

Pour la génération de l'id, l'utilisation de uuid de la bibliothèque uuidv4 a été utilisée comme conseillé dans la consigne. Ceci permettra de générer un id aléatoire pour chaque bloc.

Pour la date, une fonction récupère la date actuelle et la formate au format demandé. Pour cela je me suis basé du format renvoyé par la commande toISOString qui est la suivante "aaaa-mm-jjThh:mm:ss.sssZ " afin d'obtenir le format "aaaammjj-hh:mm:ss" j'ai d'abord retirer tout les tirets pour avoir ce résultat "aaaammjjThh:mm:ss.sssZ" puis changé le T majuscule en un tiret avant de retirer tout ce qui se trouve après le dernier point.

```
export function getDate() : string  {
    return new Date().toISOString().
    replaceAll( searchValue: "-",  replaceValue: '').
    replace( searchValue: /T/, replaceValue: '-').
    replace( searchValue: /\..+/,  replaceValue: '')
}
```

J'ai rangé ensuite toute les valeurs dans une constante json qui correspond alors au bloc.

```
const data :{...}  = {
    id : id,
    date : date,
    nom : nom,
    don : don,
    hash : hash
}
```

Je récupère ensuite toute les données dans le fichier avec la fonction findBlocks codé auparavant et je rajoute à ce dernier le nouveau bloc qu'on vient juste de créer dans la constante res. Je remplace alors le contenu du fichier actuel avec notre nouvelle liste d'objet JSON en prenant le soin de le transformer en string avant puis je retourne la liste des blocs pour que le client puisse les récupérer.

```
const content = await findBlocks();

const res :any[]  = [...content,data]
writeFile(path,JSON.stringify(res),'utf8');
return res;
```

Exercice 4 :

Le fonction findBlocks permet de récupérer le dernier bloc dans le fichier de stockage, pour cela on utilise la fonction findBlocks pour lire le fichier et récupérer alors la list de tous les blocs, si cette liste est vide alors on renvoie null pour signifier qu'il n'existe pas de bloc précédent sinon on retourne le dernier objet de cette liste afin de faire cela on soustrait 1 à la longueur de la liste pour l'utiliser comme index du dernier élément.

```
export async function findLastBlock() : Promise<...>  {
    const contents = await findBlocks();
    if (contents === []){
        return null;
    }
    return contents[contents.length-1];
}
```

La fonction findLastBlock sera donc ensuite utiliser dans createBlock pour compléter la création d'un bloc avec son dernier attribut, le hash, ce dernier sera créer à partir du retour de la fonction findLastBlock, si le retour est null le hash le sera aussi cependant si le retour est un bloc on le transformera directement en string pour s'en servir pour créer le hash. On s'aide de l'algorithme sha256 pour créer le hash à partir du string obtenu. La fonction digest qui est utilisée à la fin permet de compacter le résultat dans un format plus court. On ajoute à la fin cette nouvelle valeur dans le bloc json du bloc qui sera ajouté dans le fichier pour la sauvegarde.

```
let lastBlockString : string  = JSON.stringify(await findLastBlock());

let hash
if(lastBlockString == null){
    hash = null;
}
else{
    hash = createHash('sha256').update(lastBlockString).digest( algorithm: 'hex');
}
```

Conclusion :

Ce TP n'a pas été si compliqué mais il m'a permis de travailler et d'apprendre de nouvelle notion, j'ai pu revoir le regex et donc comment manipuler un format de string pour le transformer en un autre et aussi apprendre la syntaxe "…liste" pour segmenter une liste et pouvoir lui ajouter un nouvel élément facilement.