IUT de Paris - Rives de Seine
Université Paris Cité

BUT 3 INFORMATIQUE

Parcours A FI

Réalisations d'applications

RAPPORT TP1

Charité et Blockchain

Prénom et Nom de l'étudiant : Akash Selvaratnam

Groupe : 303

Promotion : 2023-2024

Sommaire

Table des matières

Étape 1 – « Avengers, rassemblement ! »

Pour l'étape 1, je me suis créé un compte sur le site de Marvel avec mon mail institutionnelle, puis j'ai pu découvrir l'ensemble des API de test dans le répertoire interactive documentation, j'ai pu tester l'API suivante /v1/public/characters qui ma retourner une erreur car je n'ai pas spécifié mon api KEY. J'ai également testé mon API avec le paramètre name qui m'a également retourner une erreur, car je n'ai pas spécifié l'API key.

Request URL

```
https://gateway.marvel.com:443/v1/public/characters?name=ironman&apikey=
```

Response Body

```
{
  "code": "InvalidCredentials",
  "message": "The passed API key is invalid."
}
```

Response Code

```
401
```

Response Headers

```
{
  "Content-Type": "application/json; charset=utf-8",
  "Date": "Tue, 30 Jan 2024 15:07:45 GMT"
}
```

Figure 1 : Requête personnage Marvel (iron man) erreur 401

Étape 2 – Toujours plus loin

Pour l'étape 2, j'ai créé une nouvelle collection nommée Marvel sur Postman, j'ai ensuite effectué la requête suivante, http://gateway.marvel.com/v1/public/characters. Pour pouvoir faire exécuter cette requête, je me suis dirigé vers Pre-request script et indiquer les valeurs de différentes variables telles que la clé privé, la clé publique, le timestamp et le hachage.

```
const publicKey = "de2fca9a1c8f20f60536992bf0ded52a"
const privateKey = "c2233cdd3bf4e1006f23b4849767a357f7fe3b09"
const date = new Date()
const ts = date.toISOString()
const hash = CryptoJS.MD5(ts + privateKey + publicKey).toString()

console.log(ts)


pm.environment.set("publicKey", publicKey);//variable d'environnement
pm.environment.set("ts", ts);//variable d'environnement
pm.environment.set("hash", hash);//variable d'environnement
```

Figure 2 : Variable d'environnement

J’ai ensuite indiqué dans les paramètres de la requête, la clé, le timestamp et le hachage, pour pouvoir spécifier ces valeurs dans les paramètres, il faut les indiquer en tant que variable d’environnement dans le pre-request-script.

Figure 3 : Paramètre de la requête avec les valeur des variables d'environnement

J’ai appuyé le bouton send pour pouvoir exécuter ma requête http et cela m'a retourner un statue 200 pour m’indiquer que la requête à bien été effectué.

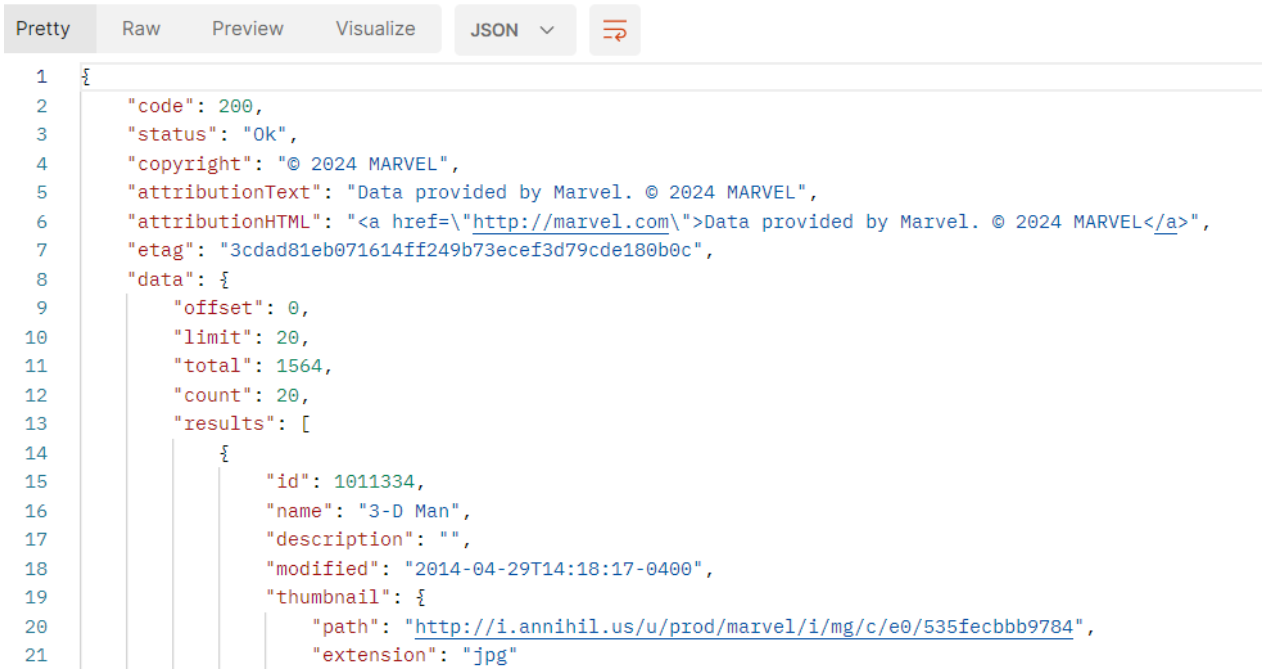

Figure 4 : Résultat de la requête http://gateway.marvel.com/v1/public/characters

Étape 3 – Tu veux ma photo ?

Pour l'étape 3, j'ai récupéré le code source de l'application en réalisant un git clone, ensuite, j'ai installé le module fetch afin de pouvoir faire exécuter l'api Marvel suivante http://gateway.marvel.com/v1/public/characters et d'obtenir ces résultats.

Dans le fichier api.js, il existe deux différentes fonctions getData(), fonction permettant de faire appel à la requête Marvel de la même façon que réaliser avec Postman, c'est-à-dire avec le timestamp, la clé publique et le hachage entre la clé publique, la clé privée et le timestamp en MD5 fourni par la fonction getHash() à réaliser également.

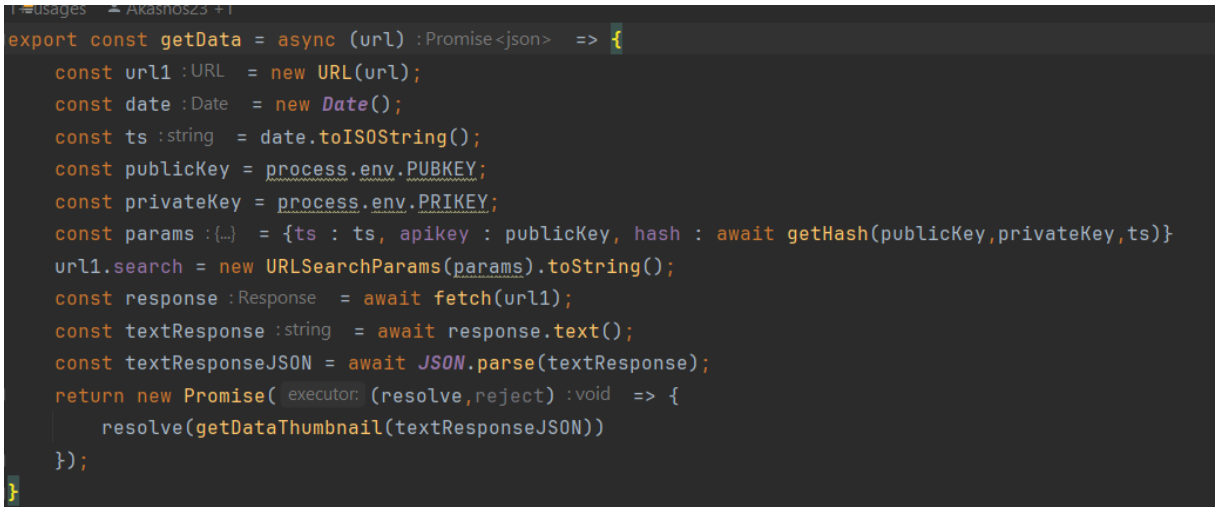

Figure 5 : Fonction getData()

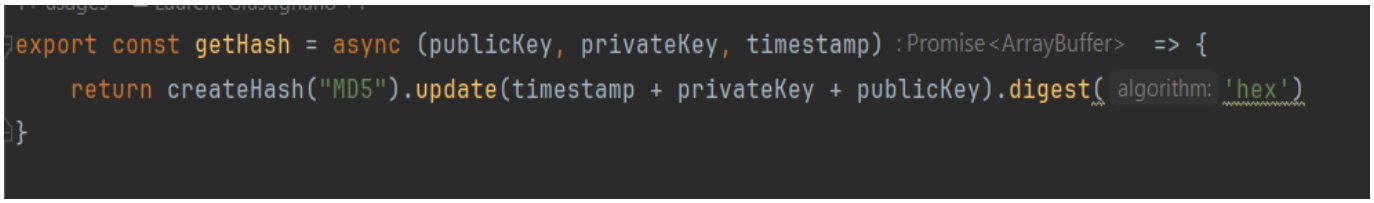

Figure 6 : fonction getHash()

La fonction getDataThumbnail() qu'on peut apercevoir dans la fonction getData() permet de sélectionner les personnages ayant une image valide, c'est-à-dire, les personnages dont l'URL de l'image ne contient pas image_not_available. Ensuite, nous retournons le nom du personnage, sa description si elle existe ainsi que son image en ajoutant .jpg à la fin de l'url de l'image afin qu'elle puisse être affiché pour la prochaine étape, pour l'ensemble des personnages ayant une image valide.

```
export const getDataThumbnail = async (resultat) : Promise<…>  => {
    const tab : any[]  = [];
    for(let i : number  = 0; i < resultat.data.results.length; i++){
        if(!resultat.data.results[i].thumbnail.path.includes("image_not_available")) {
            let tabJSON : {…}  = {
                name : resultat.data.results[i].name,
                description : resultat.data.results[i].description,
                imageURL : resultat.data.results[i].thumbnail.path.concat(".jpg")
            }
            tab[i] = tabJSON;
        }
    }
    return tab;
}
```

Figure 7 : fonction getDataThumbnail()

Avec cette étape, j'ai pu apprendre comment gérer des API via le module fetch (indiquer les paramètres, récupérer la réponse de la requête …) ainsi que de manipuler des fichiers JSON.

Étape 4 – On s'accroche au guidon

Pour l'étape 4, j'ai tout d'abord installé les modules Fastify, @Fastify/view et handlebars avec la commande npm, j'ai ensuite configuré fastify view en indiquant mon moteur handlebars ainsi que mes fichiers partiels, le header et le footer.

```
app.register(fastifyView, {
        engine: {
                handlebars: handlebars,
        },
        includeViewExtension: true,
        options: {
                partials: {
                        header: '../../templates/header.hbs',
                        footer: '../../templates/footer.hbs'
                }
        },
        templates : 'views'
});
```

Figure 8 : Configuration de Fastify view

Ensuite, j'ai réalisé une requête http get afin de pouvoir récupérer les informations des personnages (nom du personnage, description si elle existe et l'image du personnage) comme définis dans l'étape précèdent, j'indique que la requête HTTP concerne la page index et je renvoie les information des personnages récupéré à la page index.

```
app.get('/', (re : FastifyRequest<RouteGeneric, http.Server, http.IncomingMessage, SchemaCompiler, FastifyTypeProviderDefault, ContextConfi
        const resultat : Promise<json>  = getData( url: "http://gateway.marvel.com/v1/public/characters");
        resultat.then((retour :json ) => {
                return res.view("../../templates/index.hbs", {retour : retour});
        }
        )
})
```

Figure 9 : Requête HTTP permettant de récupérer les informations des personnages et d'envoyer ces information à la page index

Dans la page index, j'indique au début le composant header et à la fin le composant footer, je réalise ensuite un for each sur la variable renvoyé dans notre cas, il s'agit des information des personnages Marvel et j'affiche pour chaque son nom, sa description et son image grâce à la balise img.

```
{{> header}}
<div class="container-fluid m-2 p-5 bg-primary text-center">
    <h1>Les Marvels</h1>
</div>
<div class="container m-2 p-5 bg-primary  text-center">
    <h2>Mes comics</h2>

</div>
<div class="row">
        {{#each retour}}
            <p>{{name}} - {{description}}</p>
            <img src="{{imageURL}}"/>
        {{/each}}

</div>
{{> footer}}
```

Figure 10 : Page index.hbs

Ensuite, j'appuie le bouton exécuté cela fait tourner mon serveur sur le port 3000 afin de pouvoir visualiser le résultat obtenu sur mon navigateur via la requête suivante http://localhost:3000 .

Avec cette étape, j'ai pu apprendre comment fonctionner le module Fastify et comment gérer des fichiers Handlebars.

Étape 5 – C'est dans la boite !

Pour l'étape 5, j'ai créé deux différents fichiers le Dockerfile et .dockerignore, dans le dockerignore j'ai indiqué l'entrée node_modules et npm_debug.log afin qu'il soit ignoré lorqu'e je réaliserai mon conteneur docker. Dans le Dockerfile, j'ai créé un répertoire menant vers /home/node/app/node_modules avec la commande MKDIR, j'ai ensuite indiqué le chemin vers le dossier app grâce à la commande Workdir pour définir mon dossier de travail. J'ai copié mes fichiers source (templates et src), mon package.json, mon package-lock.json et mon fichier .env avec la commande COPY. J'ai exécuté la commande npm install pour qu'il puisse installer l'ensemble des modules, j'expose le port 3000 avec la commande Expose et je fais lancer mon serveur avec la commande CMD.

```
FROM node:lts-bullseye-slim

LABEL authors="akash"

RUN mkdir -p /home/node/app/node_modules
RUN chown node /home/node/app/node_modules
WORKDIR /home/node/app
COPY package*.json ./
COPY ./.env ../
RUN npm install
COPY ./src ./src
COPY ./templates ../templates
EXPOSE 3000
CMD ["node", "src/server.js"]
```

Figure 11 : Fichier Dockerfile

J'exécute la commande suivante docker build . -t nomDeLImage sur mon terminale en indiquant un nom d'image afin qu'il puisse construire l'application que je spécifie dans mon fichier Dockerfile et j'exécute ensuite docker run suivie du nom de l'image afin de pouvoir visualiser sur mon navigateur. Il faut également indiquer au serveur la fonction listen le host : « 0.0.0.0 ».

Je peux ensuite visualiser le résultat obtenu sur mon navigateur avec le lien suivant http://localhost:3000.

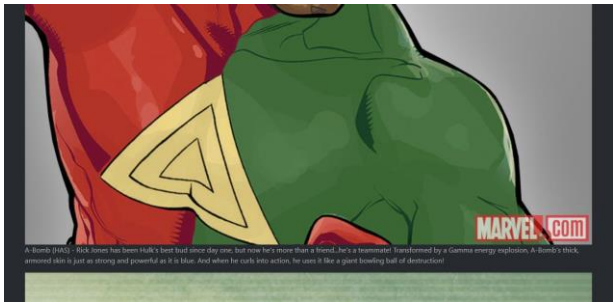

Figure 12 : Résultat obtenu sur le navigateur au lien suivant http;//localhost:3000

J’ai également crée un fichier .env regroupant deux variables d’environnement la clé publique et privé de l’API Marvel. J’utilise ces variables d’environnement dans le fichier api.js grâce aux modules dotenv dans laquelle j’ai configuré le chemin vers le fichier .env afin de pouvoir récupérer les variables d’environnement qu’il contiennent et les utiliser dans mon fichier api.js.

```
const publicKey = process.env.PUBKEY;
const privateKey = process.env.PRIKEY;
```

Figure 13 : Initialisation des variables public key et private key avec les variables d'environnements

Grâce a cette étape, j’ai pu apprendre comment crée un fichier docker, la constitution de ce fichier et comment lancé un fichier docker, j’ai également pu apprendre comment créer un fichier des variables d’environnement et comment les utiliser avec le module dotenv.