Rapport

Les Marvels

Introduction

Ce TP a pour but de nous initier aux thèmes suivants :

- Thèmes
- Accès API REST distante
- Le module Fastify
- Le moteur de template Handlebars
- Dockerisation d'une application

Il consiste en la récupération de données via une API, leur affichage avec Handlebars puis la dockerisation de l'application.

Implémentation

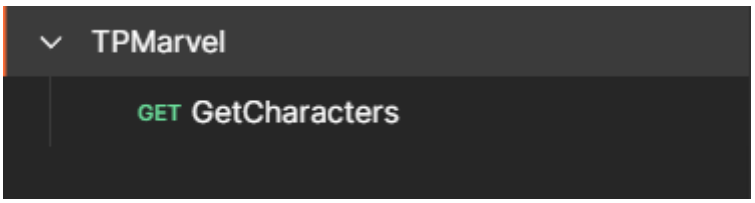
Étape 1

La première étape consistait en la récupération de l'accès à l'API de Marvel (à l'adresse https://developer.marvel.com).

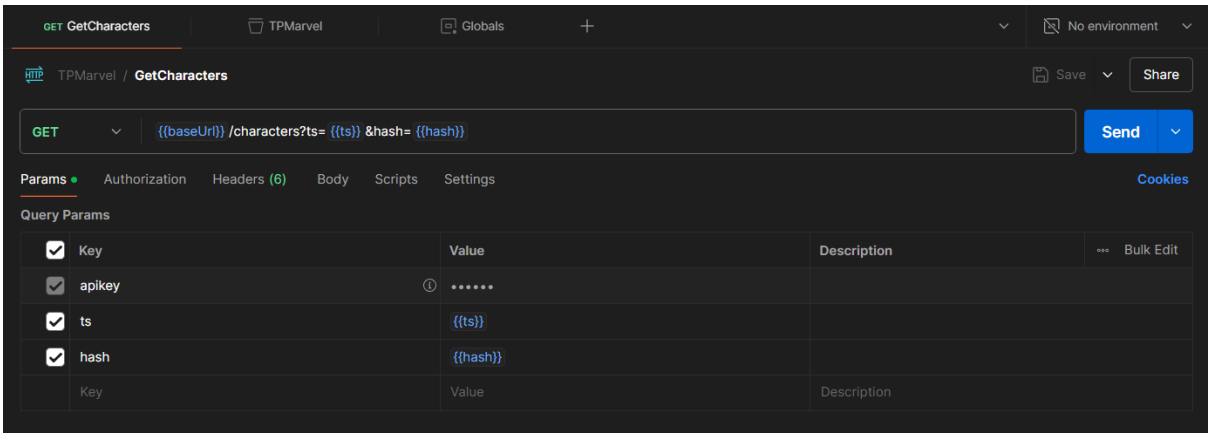
Pour cela, j'ai créé un compte sur le site. Cependant, à cause de changements sur celui-ci, la poursuite de cette manipulation n'est plus possible. Ainsi, je suis directement passée à l'étape 2.

Étape 2

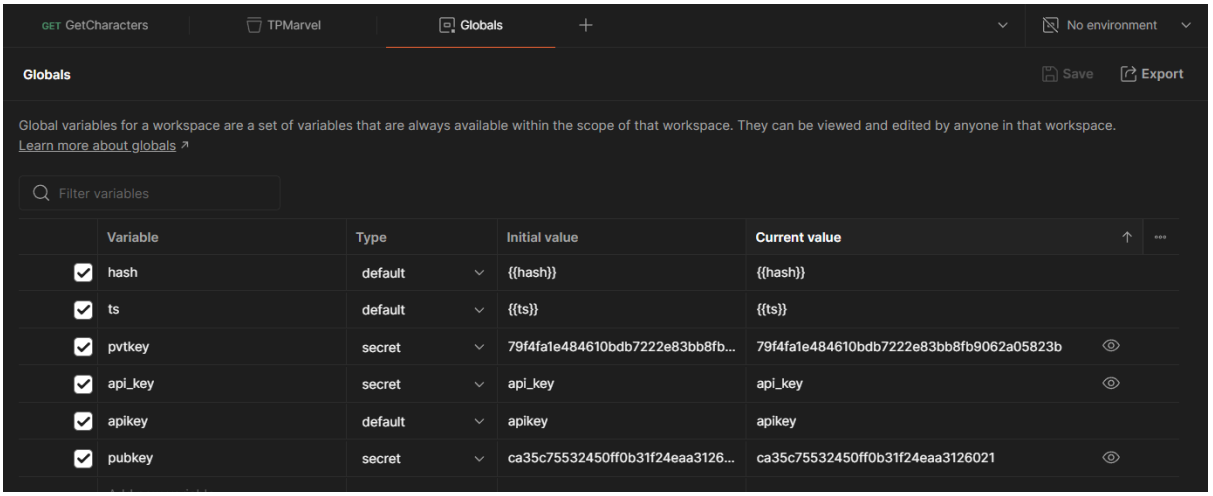
Pour cette étape, j'ai utilisé Postman. Dessus, j'ai créé une nouvelle collection nommée "TPMarvel" puis une nouvelle requête GET, utilisant l'url et l'endpoint correspondants afin d'obtenir la liste des personnages.



Collection TPMarvel et Requête GET



Requête GET utilisant l'URL correspondant ainsi que les paramètres nécessaires



Variables globales définies (url, clés publique et privée, etc…)

Enfin, il a été nécessaire d'ajouter une Pre-Request-Script contenant mes identifiants et les stockant dans les variables correspondantes.

TPMarvel

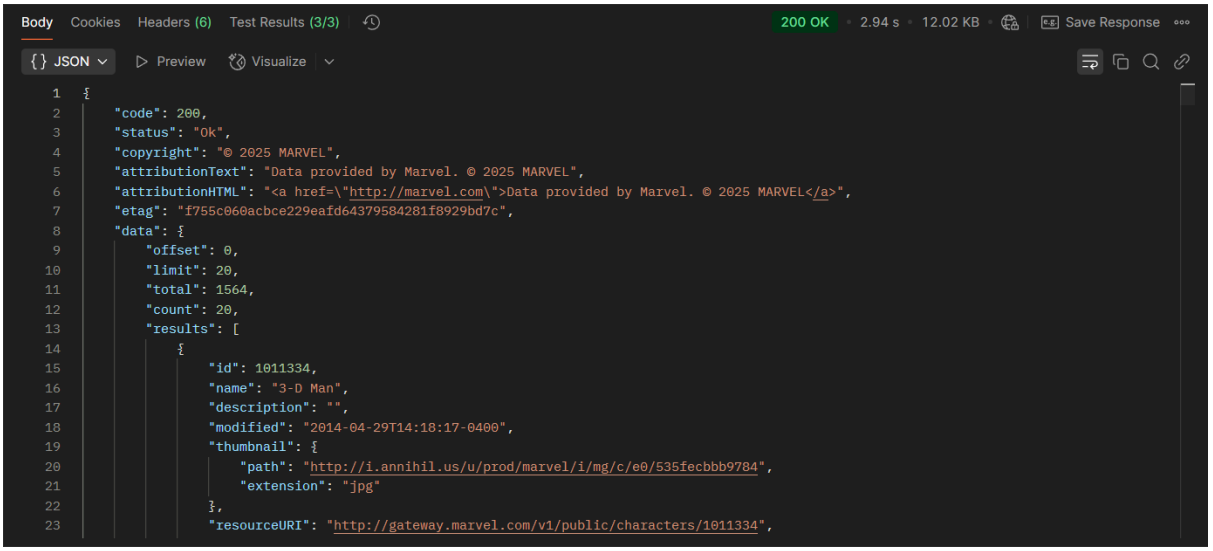
Overview Authorization Scripts Variables Runs

Pre-request

Post-response

```
/*

Pre-request script:
=> Generates MD5 hash using public key, private key & timestamp
=> Used for all requests in collection
=> `pubkey`, `pvtkey` & `ts` parameters stored in collection(s)

*/

var pubkey = "ca35c75532450ff0b31f24eaa3126021";
var pvtkey = "79f4fa1e484610bdb7222e83bb8fb9062a05823b";

var ts = new Date().getTime();

var message = ts+pvtkey+pubkey;
var a = CryptoJS.MD5(message);

pm.environment.set("ts", ts)
pm.environment.set("apikey", pubkey)
pm.environment.set("hash", a.toString())

```

Une fois ces étapes effectuées, nous pouvons observer les réponses de l'API.



Étape 3

L'étape 3 sert à créer notre propre affichage des personnages obtenus grâce à l'API. Pour cela, j'ai d'abord récupéré un squelette de l'application à l'adresse https://github.com/laurentgiustignano/LesMarvels.git

Ensuite, il a été nécessaire d'installer le module *node:fetch*, puis de coder les fonctions *getHash()* et *getData()*, respectant les mêmes spécifications que l'authentification faite via Postman.

Je n'ai récupéré que les personnages avec des images (*thumbnail*) valides; soit qui ne sont pas "image_not_available". Enfin, j'ai créé un tableau de personnages contenant pour le champ *imageUrl* la destination de la version *portrait_xlarge* de l'image.

Code :

```
/**
 * Récupère les données de l'endpoint en utilisant les identifiants
 * particuliers developer.marvels.com
 * @param url l'end-point
 * @return {Promise<json>}
 */
export const getData = async (url) => {
    try {
        const ts = new Date().getTime().toString();
        const hash = await getHash(PUBLIC_KEY, PRIVATE_KEY, ts);

        const noParams = url.indexOf('?') === -1;
        const debutParam = noParams ? '?' : '&';

        const finalUrl = url + debutParam + 'ts=' + ts + '&apikey=' +
PUBLIC_KEY + '&hash=' + hash;

        const response = await fetch(finalUrl);
        if (!response.ok) {
            throw new Error(`L'API a répondu avec le statut :
${response.status}`);
        }

        const res = await response.json();
        if (!res.data || !res.data.results) {
            throw new Error('Format de réponse inattendu');
        }

        const triThumbnailValid = res.data.results.filter((char) => {
            return char.thumbnail && char.thumbnail.path &&
                   char.thumbnail.path.indexOf('image_not_available')
=== -1;
        });
```

```
        return triThumbnailValid.map((char) => ({
            ...char,
            imageUrl:
`${char.thumbnail.path}.${char.thumbnail.extension}`
        }));
    } catch (error) {
        console.error('Erreur', error);
        throw error;
    }
}

/**
 * Calcul la valeur md5 dans l'ordre : timestamp+privateKey+publicKey
 * cf documentation developer.marvels.com
 * @param publicKey
 * @param privateKey
 * @param timestamp
 * @return {Promise<ArrayBuffer>} en hexadecimal
 */
export const getHash = async (publicKey, privateKey, timestamp) => {
    return
createHash('md5').update(`${timestamp}${privateKey}${publicKey}`).digest
('hex');
}
```

Étape 4

Cette étape sert à afficher nos données sur un site web. Pour cela, il fallait se servir des modules *Fastify* (et son plugin *@fastify/view*) et *Handlebars*. Afin que la mise en page soit uniforme, des fichiers *header.hbs* et *footer.hbs* étaient fournis. J'ai également utilisé des balises *<li>* pour énumérer les personnages.

Tout d'abord, il fallait enregistrer le moteur Handlebars dans fastifyview puis ajouter les fichiers *header.hbs* et *footer.hbs* comme partials dans la configuration.

```
const app = Fastify();
const __dirname = path.resolve();

app.register(fastifyStatic, {
    root: path.join(__dirname, 'templates')
});

app.register(fastifyView, {
    engine: {
```

```
        handlebars: handlebars
    },
    templates: 'templates',
    options: {
        partials: {
            header: path.join('header.hbs'),
            footer: path.join('footer.hbs')
        }
    }
});
```

Enfin, il restait à insérer dans le fichier *index.hbs* insérer les fichiers partials déclarés et compléter * * pour obtenir l'affichage des personnages. De plus, il a été nécessaire d'ajouter un appel à nos fonctions créées en étape 2, dans *server.js,* pour obtenir les données à afficher.

```
// Appel dans server.js
 app.get('/', async (request, reply) => {
    try {
        const data = await
getData("https://gateway.marvel.com:443/v1/public/characters");
        return reply.view("index.hbs", { data: data });
    } catch (err) {
        console.error("Erreur : ", err);
        return reply.status(500).send(err.message);
    }
});
```

// Fichier idex.hbs

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

    <div class="row">
        <ul>
            {{#each data}}
                <li>
                    <img src={{this.imageUrl}} alt={{this.name}}>
                    <h5>{{this.name}}</h5>
                </li>
            {{/each}}
        </ul>
    </div>
</div>
{{> footer}}
```

Étape 5

Cette dernière étape avait pour objectif de conteneuriser notre application pour la déployer sur différentes machines sans se soucier d'installer node et ses modules. Pour cela, j'ai d'abord créé un fichier *Dockefile* et *.dockerignore*.

Dans dockerignore, j'ai placé les entrées node_modules et npm-debug.log, qui ne sont pas pertinentes pour une image destinée à la production.

```
node_modules
npm-debug.log
```

Dans *Dockefile*, j'ai d'abord commencé par préciser la version de node *node:lts-bookworm-slim*, puis j'ai créé l'arborescence de destination */home/node/app/node_module* en précisant node comme propriétaire. Ensuite, j'ai défini le chemin d'accès jusqu'à *app* avant de copier les fichiers *package*.json* dans le répertoire de travail. Il ne restait maintenant qu'à lancer la commande *npm install* pour installer les modules désignés dans *package.json*, effectuer la copie des fichiers sources, et, à la fin du *Dockerfile*, lancer la commande de démarrage de l'application node avec *CMD*.

```
FROM node:lts-bookworm-slim

WORKDIR /usr/src/app

RUN mkdir -p /home/node/app/node_modules && chown -R node:node
/home/node/app

COPY package*.json ./

RUN npm install

COPY . .

CMD ["node", "src/server.js"]
```

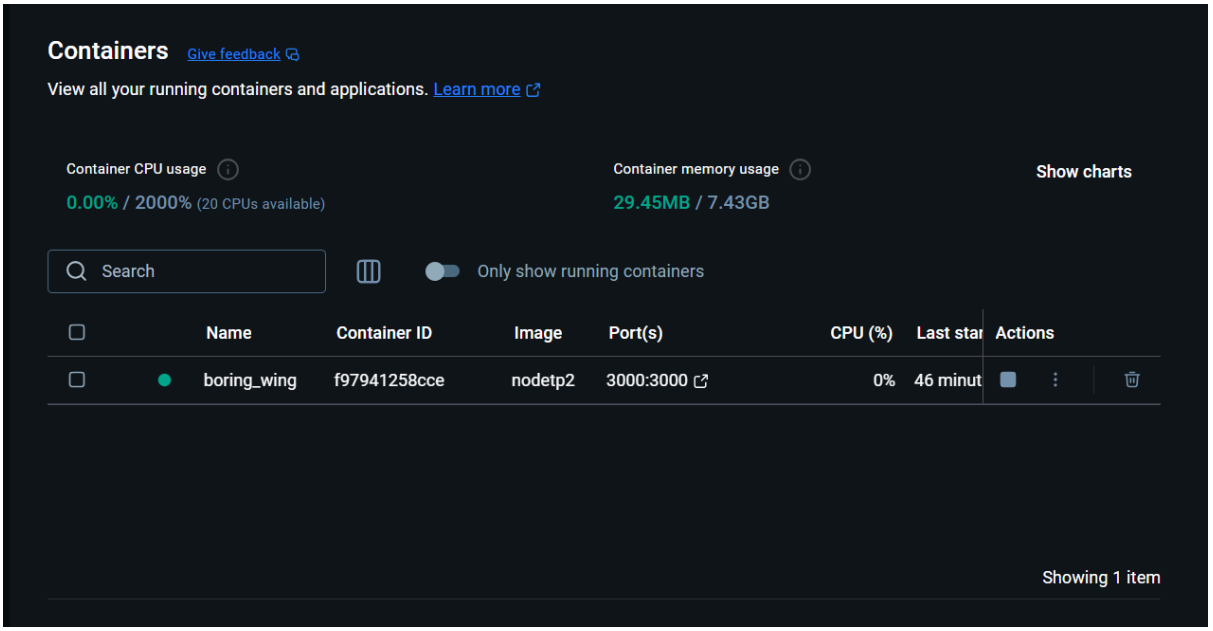
Une dernière étape consistait à "cacher" nos clés d'API dans un fichier .env puis l'injecter dans le code NodeJS lors de son exécution.

```
### Configuration API Marvels
PUBKEY = "ca3...6021"
PRIKEY = "79f...823b"
```

Une fois ce fichier créé, j'ai remplacé les clés dans *api.js* par des variables obtenues grâce au module *dotenv.*

```
import dotenv from 'dotenv';

dotenv.config();

const PUBLIC_KEY = process.env.PUBKEY;
const PRIVATE_KEY = process.env.PRIKEY;
```

L'image Docker doit être régénérée et peut être relancée.



Conclusion

Pour conclure, ce projet est intéressant pour mieux comprendre le fonctionnement d'API et de Docker. J'ai également pu apprendre à utiliser le moteur Handlebars ainsi que le module Fastify pour l'affichage.