2024 - TP2

RAPPORT DÉVELOPPEMENT AVANCÉ

Mathéo BIENVENU

SOMMAIRE

1 - EXPLICATIONS DE MES CHOIX ET DIFFICULTÉS RENCONTRÉES

Etape 1 :

Au sein de cette étape, rien de très compliqué, il m'a suffi de suivre les consignes et, ayant déjà utilisé beaucoup d'APIs, je n'ai pas eu de problème à comprendre celle-ci et ce qu'elle retourne.

Etape 2 :

Lors de cette étape, c'était assez nouveau pour moi d'utiliser Postman de cette manière. J'ai souvent fait mes tests directement via un bouton sur mon site, par exemple, pour essayer mes requêtes **POST**. J'ai suivi les consignes, j'ai eu un peu de mal à comprendre où je devais renseigner les variables d'environnement, mais j'ai trouvé et j'ai su facilement les mettre en place puis les utiliser au sein des paramètres de la requête.

Etape 3 :

Cette étape a été la plus compliquée selon moi, mais comme dit précédemment, j'ai utilisé beaucoup d'APIs auparavant lors de projets personnels, alors je savais exactement comment procéder. J'ai directement décidé de cloner le projet comme demandé, puis d'installer **node-fetch** avec **npm i node-fetch.**

Je savais que **getHash()** allait être très rapide à faire, alors j'ai simplement importé le module **createHash** de **node:crypto**, j'ai écrit en une ligne pour faire très simple et très lisible la création et le retour du hash.

Ensuite, j'ai attaqué le développement de la fonction **getData()**. J'ai d'abord mis en place le timestamp en utilisant **Date.now()**, que j'ai bien suivi d'un **.toString()**, car nous avons besoin de concaténer ce timestamp avec la **publicKey** et la **privateKey**, alors il est nécessaire que ça ne soit pas un **Integer** mais bien un **String**.

J'ai ensuite fait ma requête **fetch** avec l'url en ajoutant "?", car j'ai pensé au fait que si on améliorait l'application et que l'utilisateur ou nous-mêmes devions donner d'autres URL plus tard, cela ne va pas forcément de soi de rajouter le point d'interrogation. Puis, je rajoute à l'URL les différents paramètres grâce à la méthode **URLSearchParams** que j'avais déjà utilisée auparavant, à laquelle j'ai donné en paramètre l'**apiKey**, le **timestamp** et le **hash** qui sont tous les trois nécessaires à l'envoi de cette requête.

Etape 3 (suite) :

Une fois la requête effectuée, je la récupère puis je mets la réponse en format **JSON** directement que je stocke dans une variable datas. Une fois cette étape finie, j'ai mis un console.log de datas pour me permettre de voir comment étaient structurées les données à l'intérieur. J'ai vu qu'il y avait un champ nommé "**data**" qui lui-même contenait "**results**", qui contenait les résultats que nous cherchions !

Une fois avoir pris connaissance de toute cette structure, cela a été assez simple. J'ai utilisé la méthode "**filter**" qui m'a permis de retourner dans un tableau initialisé précédemment tous les résultats qui contenaient une image.

J'ai ensuite utilisé la méthode "**map**" afin de retourner seulement les informations dont j'avais besoin, c'est-à-dire : le nom, la description et l'URL de l'image. Pour la variable **imageUrl**, j'ai concaténé l'url avec "**/portrait_xlarge.jpg**", qui était donné dans la documentation de l'API et qui permet d'obtenir l'image sous un format spécifique. J'ai mis cette chaîne de caractères dans une variable pour faciliter une mise à jour si nous voulons maintenant utiliser des images d'un autre format que XLARGE.

Je n'ai pas précisé mais j'ai créé un fichier "**apiKeys.js**" qui contient la clé publique ainsi que la clé privée pour faciliter le changement de celles-ci si besoin mais également pour ne pas les avoir en "dur" au sein du fichier "**api.js**". J'aurais pu en faire un fichier **.env** mais comme j'avais déjà séparé ces clés dans un fichier, j'ai laissé cela comme ça !

Etape 4 :

Lors de l'étape 4, je me suis aidé du cours et du TD faits précédemment. Je n'ai pas rencontré de grosse difficulté car j'ai déjà utilisé **Express**. C'est assez similaire, mais le système de handlebars était nouveau pour moi, car je n'en avais jamais utilisé avec Express. J'ai simplement fait les installations et les imports nécessaires puis utilisé "**handlebars**" comme moteur et établi les templates "**header**" et "**footer**".

J'ai ensuite simplement utilisé "**app.get('/')**", et dedans, j'appelle **getData()** que j'ai développé précédemment pour ensuite récupérer la liste des personnages. Je retourne une réponse avec "**res.view**" vers le template index et je n'oublie pas de fournir la liste de personnages également.

Je suis ensuite allé dans index et j'ai ajouté le **header** et le **footer** et utilisé une balise de liste dans laquelle j'ai utilisé **#each**, qui m'a permis pour chaque personnage d'afficher son nom, sa description, et son image.

2 - AMÉLIORATIONS POSSIBLES

Vis-à-vis des améliorations possibles, je n'en vois pas du côté technique. Tout est bien exécuté et simplement. En revanche, si on venait à améliorer l'application web pour afficher l'entièreté des personnages, il serait vraiment sympa de rajouter une flèche permettant de renvoyer une requête en modifiant l'offset dans l'URL. Par défaut, l'offset est à 0 lorsque nous ne le donnons pas en paramètre, alors nous nous retrouvons avec des personnages inconnus et nous ne pouvons en voir d'autre sur la page web.

3 - CONCLUSION

En conclusion, j'ai trouvé ce TP très intéressant et je n'ai pas rencontré de grosses difficultés. Cependant, je n'avais pas connaissance de l'onglet "**pre-request-script**" dans Postman, qui s'avère au final très utile et que je réutiliserai très probablement sur de prochains projets. Je pense également remplacer Express par Fastify pour mes prochains projets, car j'ai adoré le système de schéma qui est très utile et qu'on n'a effectivement pas sur Express.