

## ÉNONCE TRAVAUX PRATIQUE - SUJET 2

### Les Marvels

#### Thèmes

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

#### Présentation

Le principe d'une API REST consiste à proposer l'accès à des données en utilisant une connexion http. De la même manière que nous utilisons un navigateur pour accéder à des données sur un serveur distant, un programme peut accéder aux mêmes données à travers une API. Cette interface règle et régule les accès, en fonction différents éléments, comme un chemin d'accès, une authentification, un verbe http, etc... Beaucoup d'entreprises et de ministères proposent l'accès à l'OpenData, par le biais d'API gratuite ou payante, restreinte ou totalement ouverte.

Pour ce sujet, nous allons utiliser l'API de la société Marvels qui propose de fournir, à but pédagogique, les informations de sa base de données sur les Comics qui ont été édités. L'accès est restreint, c'est à dire qu'il faut s'enregistrer sur le site <https://developer.marvels.com> et ainsi obtenir une clé d'API qui permettra d'effectuer jusqu'à 3000 requêtes par jour.

Avec le nom et la couverture des différents albums que nous recevrons, par exemple, nous fabriquerons une page web avec **Fastify** et utiliserons le moteur de template **handlebars** pour faciliter notre affichage.

Quand notre applicatif sera fonctionnel, nous pourrons le conteneuriser pour rendre l'applicatif transportable et déployable facilement.

#### Étape 1 - « Avengers, rassemblement ! »

La première étape consiste à récupérer l'accès à l'API de Marvels, et en vérifier son bon fonctionnement. Une fois que vous aurez un compte valide sur la plateforme, vous pouvez accéder à l'Interactive Documentation qui vous permet de tester différentes requêtes.

- ◆ Démarrer la création de compte sur le site : <https://developer.marvels.com>. Vous pouvez utiliser votre email institutionnel ou votre préféré, il n'y a aucune incidence. Par contre, il faut pouvoir y accéder car un email de vérification est envoyé. Il est nécessaire pour valider votre compte et vous connecter.
- ◆ Depuis l'Interactive Documentation, déployez la première requête, **/v1/public/characters** qui permet de récupérer les données des personnages Marvels. Choisissez **Model Schema** dans le type de réponse pour observer le format de la réponse. Identifier l'organisation de la réponse pour retrouver les noms des personnages, leurs descriptions, l'image miniature correspondante.
- ◆ Lancez la requête sans rien préciser en cliquant sur le bouton « **Try it out !** ». Notez l'URL de la demande.
- ◆ Ajoutez un paramètre dans les champs proposés, et observez le changement dans l'URL.

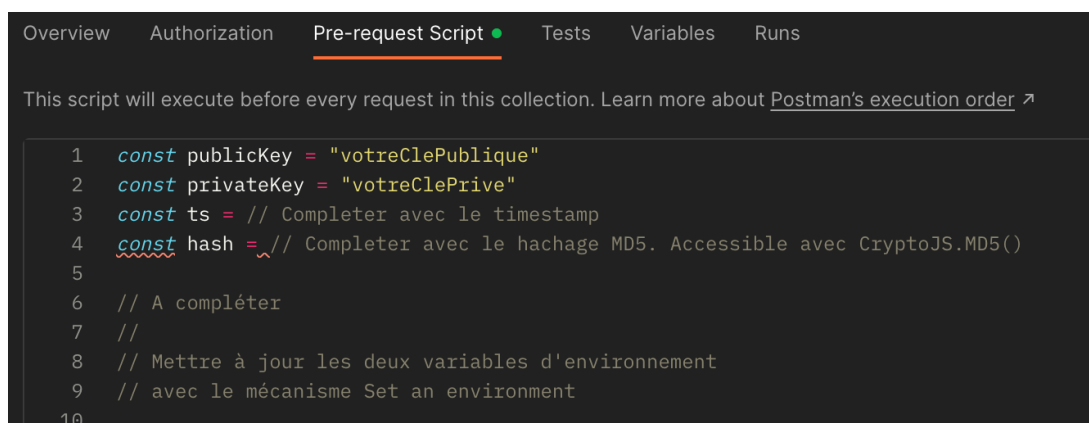
#### Étape 2 - Toujours plus loin

Pour l'instant, les résultats s'affichent dans la page web, mais cela ne nous intéresse pas pour nos travaux, nous souhaitons y avoir accès de manière distante. Pour y accéder, nous avons besoin d'une authentification qui est fourni dans l'onglet **My Developer Account**. Il s'agit d'un chiffrement asymétrique qui est composé d'une clé publique et d'une clé privée. Dans la partie **How-Tos / Authorization**, vous trouverez la mécanique qui permet de les utiliser. Nous utiliserons le logiciel **Postman** pour réaliser les requêtes, ainsi que le traitement qui permet l'authentification. Un calcul de

hachage doit être effectué sur notre ordinateur entre la clé publique, la clé privée et un timestamp, puis ce résultat avec le timestamp et la clé publique seront envoyés avec la requête. **Marvels** fera le même calcul, avec la clé privée qu'il a conservée et qui n'est pas transmise, pour vérifier votre identité.

**Postman** effectuera donc ce calcul avant chaque demande, grâce à son mécanisme de **pre-request script**. Cette fonctionnalité existe pour chaque requête, mais il est plus intéressant de le faire pour toutes les requêtes Marvel. Il convient de créer une **Collection** qui permet de traiter un ensemble de requêtes avec les mêmes conditions. Le **pre-request-script** est un code écrit en **JavaScript**, et le résultat du calcul sera échangé par un mécanisme de variable d'environnement.

- ◆ Créez une nouvelle **collection Marvels**.
- ◆ Créez une nouvelle requête dans votre **collection Marvels**. Choisissez la méthode **GET**, et copiez au niveau de l'URL le même **end-point** qu'à l'étape précédente. Dans l'onglet **params**, ajoutez les noms des clés spécifiés dans la documentation **developer.marvels.com**. Pour lier la valeur d'un paramètre avec une variable d'environnement, il faut utiliser la syntaxe suivante : **{{nomVariableEnvironnement}}**. Par exemple, vous pourrez appeler vos variables d'environnement **maintenant** et **hachage**. Vous noterez qu'à l'ajout de chaque nouveau paramètre, il est rajouté dans la requête.
- ◆ Dans l'onglet **pre-request-script** de la **collection**, complétez le code suivant pour qu'il puisse correspondre à vos identifiants. Attention les variables dans le JavaScript ne sont pas les variables d'environnement.



```

Overview  Authorization  Pre-request Script  Tests  Variables  Runs

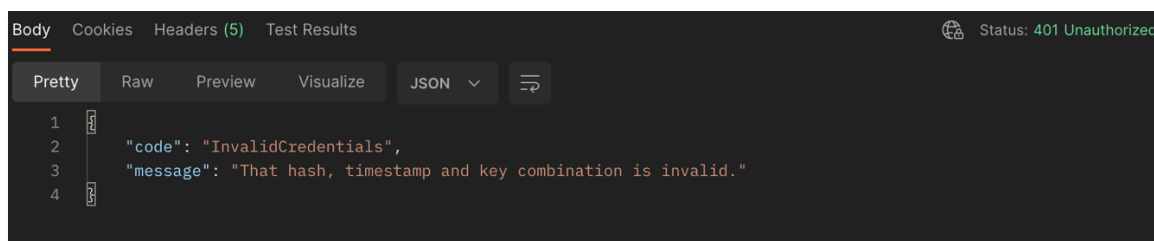
This script will execute before every request in this collection. Learn more about Postman's execution order ↗

1  const publicKey = "votreClePublique"
2  const privateKey = "votreClePrive"
3  const ts = // Compléter avec le timestamp
4  const hash = // Compléter avec le hachage MD5. Accessible avec CryptoJS.MD5()
5
6  // A compléter
7  //
8  // Mettre à jour les deux variables d'environnement
9  // avec le mécanisme Set an environment
10

```

Figure 1- Structure de code à compléter

Conseils : Postman possède aussi une console. Vous pouvez y afficher le résultat des différentes opérations que vous y ferez. En cas de mauvaise authentification, la réponse à votre requête ressemblera au message ci-dessous.



```

Body  Cookies  Headers (5)  Test Results  Status: 401 Unauthorized

Pretty  Raw  Preview  Visualize  JSON  ↕

1  [
2    "code": "InvalidCredentials",
3    "message": "That hash, timestamp and key combination is invalid."
4  ]

```

Figure 2 - Exemple d'authentification invalide

### Étape 3 - Tu veux ma photo ?

A présent que nous savons accéder à l'API Marvels depuis **Postman**, il nous reste à écrire le code de notre application **NodeJS** qui affichera, par exemple, le nom des personnages et l'image associée dans la base Marvel, à travers un navigateur. Nous aurons besoin de reconstruire notre authentification avec les mêmes éléments, dont nous avons prouvé le fonctionnement. La méthode **fetch()** est une fonction

asynchrone incluse dans **node-fetch** qui permet de réaliser des requêtes vers des end-points. Il sera aussi nécessaire d'utiliser la méthode **createHash()** du module **node:crypto**, pour réaliser le hachage **md5** avec notre clé d'API Marvels.

Comme cela a pu être observé depuis **Postman**, la quantité de données renvoyées est imposante. Si l'on fait une requête sur les personnages, on se contentera d'utiliser son nom, une rapide description si elle existe, et une image. Beaucoup de personnage n'ont pas d'images disponible, et dans le champ **thumbnail**, la valeur est **image\_not\_available**. On éliminera les personnages sans visuel, pour constituer en retour un tableau de **Personnage**.

- ◆ Récupérez le dépôt de démarrage à l'URL suivante avec la commande : **git clone https://github.com/laurentgiustignano/LesMarvels.git**.
- ◆ Installer le module **node-fetch** avec la commande **npm**.
- ◆ Compléter dans le fichier **src/api.js** le corps des fonctions **getData()** et **getHash()** avec les mêmes spécifications que l'authentification **Postman**.
- ◆ Effectuer la sélection des résultats qui possèdent une image **thumbnail** valide.
- ◆ Constituer un tableau de **Personnage** contenant pour le champ **imageUrl** la destination de la version **portrait\_xlarge** de l'image.

## Étape 4 - On s'accroche au guidon

Ensuite, pour présenter les données sur la page web, nous allons nous servir de **Fastify** et de son plugin **@Fastify/view** qui s'occupera du rendu visuel avec **Handlebars**. L'objectif étant d'afficher sur la page, toutes les informations récupérées sur les personnages. Des fichiers **header.hbs** et **footer.hbs** sont fournis pour injecter un **Bootstrap** dans toutes les pages, et ainsi garder le même visuel sur l'ensemble du site. Les éléments pourront être affichés soit par des balises **<li>**, soit par des **card** de **Bootstrap**. L'usage des **helpers** de **Handlebars** facilitera le travail.

- ◆ Installer les modules **Fastify**, **@Fastify/view** et **Handlebars** avec la commande **npm**.
- ◆ Enregistrer le moteur **Handlebars** dans **fastifyview**. Rajouter les fichiers **header.hbs** et **footer.hbs** comme **partials** dans la configuration.
- ◆ Dans le fichier **index.hbs**, insérer les fichiers **partials** déclarés. Compléter, dans la **<div class='row'>** de ce fichier et dans **server.js**, pour obtenir l'affichage des personnages.

## Étape 5 - C'est dans la boîte !

A présent, nous allons conteneuriser cette application en créant un **Dockerfile**. Ainsi, nous pourrons la déployer sur différentes machines sans se soucier d'installer **node** et ses modules. Dans un premier temps, nous procéderons de manière directe pour vérifier le comportement de notre **Dockerfile**. Ensuite, nous verrons comment améliorer la création de notre image.

- ◆ En haut de l'arborescence de notre projet, créer un fichier **Dockefile** et **.dockerignore**. Ces fichiers n'ont pas d'extension.
- ◆ Dans le fichier **.dockerignore**, placer les entrées **node\_modules** et **npm-debug.log**. En effet, ces deux fichiers sont inintéressants dans une image destinée à la production.
- ◆ Dans le fichier **Dockerfile**, compléter en commençant par **FROM node :lts-bulleyes-slim**.
- ◆ Avec la commande **RUN**, créer l'arborescence de destination **/home/node/app/node\_module** et spécifier que le propriétaire et groupe du propriétaire est **node**.
- ◆ Définir avec la commande **WORKDIR** le chemin d'accès jusqu'à **app**.
- ◆ Copier les fichiers **package\*.json** dans le répertoire de travail.
- ◆ Lancer la commande **npm install** pour les modules désignés dans **package.json** s'installe dans le conteneur.
- ◆ A présent, effectuer la copie des fichiers sources
- ◆ Et à la fin du **Dockerfile**, lancer la commande de démarrage de l'application node avec **CMD**. Pour lancer la construction de votre image, il faut se placer dans un terminal au niveau de votre projet et lancer la commande : **docker build . -t nomDeLImage**.

- ♦ Avec la commande `docker run`, lancer votre conteneur pour observer le comportement de votre application.

*Remarque :* Notez que le point entre **build** et **-t** n'est pas une faute de frappe. Il désigne le répertoire courant où se trouve le fichier **Dockerfile**. Vérifiez également dans l'initialisation de **Fastify** de préciser dans les options de **listen()**, le champ **host** avec la valeur **0.0.0.0**. Comme beaucoup de serveur web, **Fastify** lie son serveur avec l'adresse **127.0.0.1** qui est valide lors de l'exécution sur nos ordinateurs, mais ne correspond pas quand **node** l'exécute dans son conteneur, où il doit écouter depuis n'importe quelle adresse.

Pour améliorer la construction de notre application et son déploiement, nous remarquons que les identifiants de l'API sont inclus dans le code. Il serait plus élégant de faire contenir ses informations dans un fichier externe, et qu'il soit injecté dans le code **NodeJS** lors de son exécution. Ainsi, le **Dockerfile** et le code peut être distribué sans fournir des identifiants personnels.

- ♦ Rajouter un fichier **.env** dans votre projet et rajouter des entrées similaires à l'image ci-dessous.
- ♦ Intégrer le module **dotenv** avec **npm** et en vous inspirant de la documentation du module, remplacez vos clé par les valeurs de **.env**.
- ♦ Régénérez l'image **docker** et lancer l'application.

```
### Configuration API Marvels
PUBKEY="maCléPublique"
PRIKEY="maCléPrivée"
```

## Pour aller plus loin...

Vous pouvez :

- ♦ Séparer les deux affichages en créant une page **li.html** pour la représentation sous forme de liste, et dans une page **card.html** la représentation sous forme de vignettes.
- ♦ Améliorer la création de l'image docker en utilisant un **build** en « **multistage** ». La documentation officielle **Docker** (<https://docs.docker.com>) propose un guide sur ce sujet. Vous pourrez ainsi encore diminuer la taille de l'image finale de votre application.

Bon courage...

