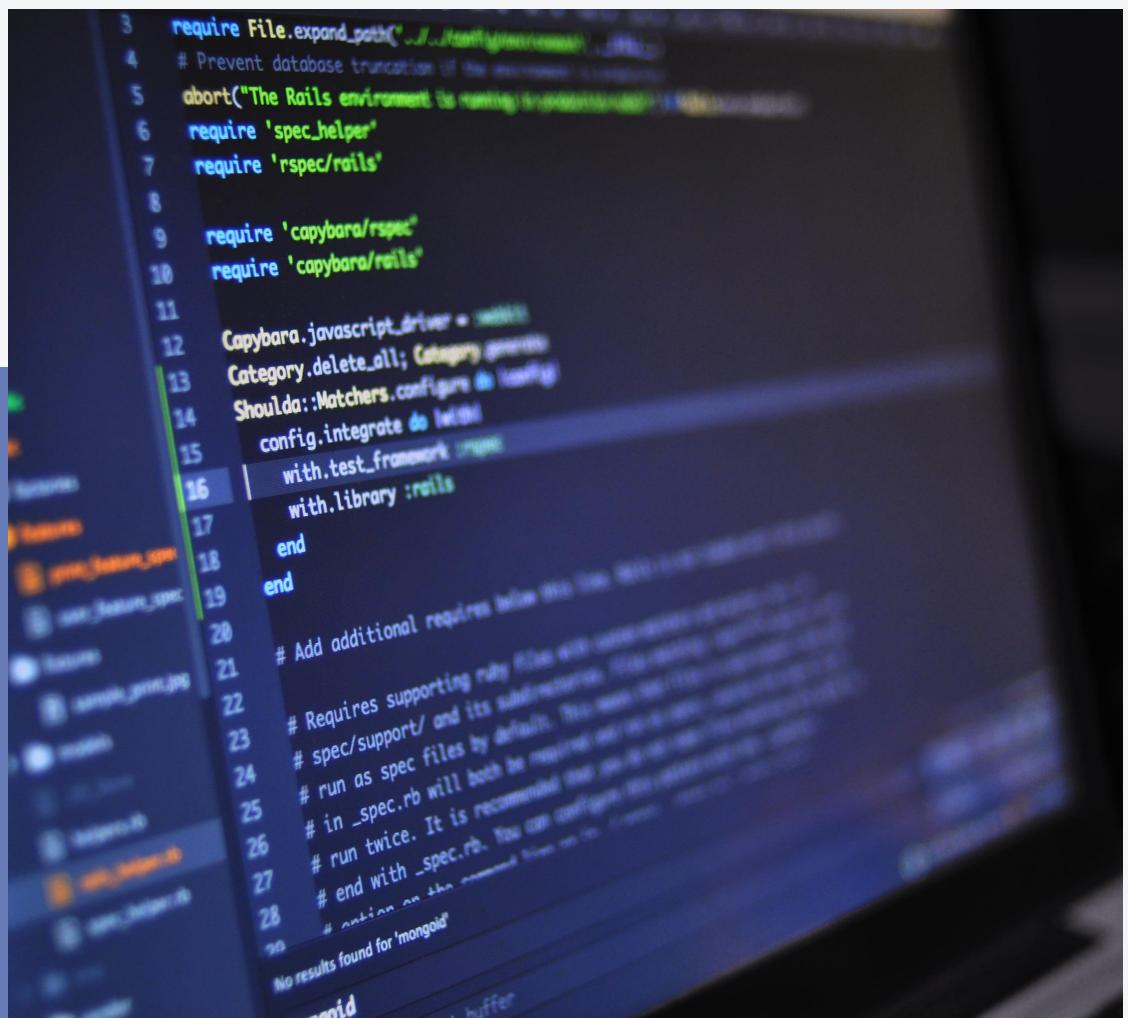


COMPTE RENDU

TP2



A screenshot of a code editor displaying a Ruby script. The code includes require statements for File, spec_helper, rspec/rails, capybara/rspec, and capybara/rails. It also defines a context block for 'Category' with various configuration options like javascript_driver, delete_all, and Matchers. A note at the bottom suggests adding additional requires for Mongoid and its submodules. The code editor has a dark theme with syntax highlighting in blue, green, and yellow.

```
3  require File.expand_path('../config/environment', __FILE__)
4  # Prevent database truncation if the environment is :test
5  abort("The Rails environment is running in production mode") if Rails.env == "production"
6  require 'spec_helper'
7  require 'rspec/rails'
8
9  require 'capybara/rspec'
10 require 'capybara/rails'
11
12 Capybara.javascript_driver = :webkit
13 Category.delete_all; Category.create!(name: "Default")
14 Shoulda::Matchers.configure do |config|
15   config.integrate do |with|
16     with.test_framework :rspec
17     with.library :rails
18   end
19 end
20
21 # Add additional requires below this line if you need them
22
23 # Requires supporting ruby files with custom matchers and helpers
24 # in spec/support/ and its subdirectories. This directory is
25 # in spec.rb will both be required automatically.
26 # run twice. It is recommended that you do this in your test suite
27 # end with _spec.rb. You can configure the :only option in
28 # action on the command line or in
29
30 # no results found for 'mongoid'
31
32 # buffer
```

CHAIMAE EL MATTARI

304

ÉTAPE 1 – « AVENGERS, RASSEMBLEMENT ! »

- Identifier l'organisation de la réponse pour retrouver les noms des personnages, leurs descriptions, l'image miniature correspondante :

Les informations sur les personnages Marvel se trouvent dans le tableau Results, qui se trouve dans le champ Data.

Les éléments du tableau représentent un personnage Marvel avec des propriétés telles que son id, son nom, sa description. On y retrouve également un champ thumbnail dans lequel on retrouve donc le path et l'extension de l'objet représentant la miniature du personnage.

Il faut donc parcourir le tableau results pour récupérer les différentes informations concernant un personnage Marvel.

- URL de la demande:

Cette URL contient uniquement le paramètre obligatoire apikey.

`https://gateway.marvel.com:443/v1/public/characters?`

`apikey=507b26042b7eb8fd48ac4281919811e1`

- URL de la demande avec le paramètre name = Spider-Man :

Cette URL contient le paramètre name avec la valeur "Spider-Man", en plus du paramètre obligatoire apikey

`https://gateway.marvel.com:443/v1/public/characters?name=Spider-`

`Man&apikey=507b26042b7eb8fd48ac4281919811e1`

ÉTAPE 2 – TOUJOURS PLUS LOIN

- La première étape a été de créer une collection nommée "Marvels" dans Postman.

Cette collection permet de traiter un ensemble de requêtes partageant les mêmes conditions.

- J'ai ensuite ajouté une requête GET dans la collection en utilisant l'endpoint que me proposait l'API pour la récupération des personnages Marvel. L'URL, dans ce cas, était :

`https://gateway.marvel.com:443/v1/public/characters?`

`apikey=507b26042b7eb8fd48ac4281919811e1.`

- J'ai ensuite utilisé des variables d'environnements, afin de simplifier la gestion des paramètres de la requête. Ces variables comprennent l'apikey, le timestamp (ts), et le hash. La syntaxe **`{{nomVariableEnvironnement}}`** m'a permis de lier ces variables à l'URL de la requête.

- Puis, j'ai écrit le pre-request script dans l'onglet Collection directement pour que celui-ci soit pris en compte par toutes les requêtes dans la collection. Ce script génère automatiquement le timestamp et le hash en fonction des clés publique et privée, et les associe aux variables d'environnement.

• Résultats et Difficultés :

Au début, j'ai initialement placé le script dans chaque requête individuelle, ce qui a conduit à une duplication inutile de code. Cependant, en déplaçant le script au niveau de la collection, j'ai assuré son exécution pour toutes les requêtes.

La documentation de l'API Marvel Comics a été essentielle et m'a permis de comprendre les paramètres nécessaires à l'authentification et à la création des requêtes. Les tests avec des requêtes telles que "get comics" et "get characters" ont été un succès, démontrant le bon fonctionnement de l'authentification.

```

1 const publicKey = "507b26042b7eb8fd48ac4281919811e1";
2 const privateKey = "f7a9610f5aae980370e8375c5a40f3d4f587af91";
3 const ts = Math.floor(Date.now() / 1000);
4 const hash = CryptoJS.MD5(ts + privateKey + publicKey).toString();
5
6 pm.environment.set("timestamp", ts);
7 pm.environment.set("hash", hash);
8
  
```

Pre-scripts dans la collection

Get comics

```

1 {
2   "code": 200,
3   "status": "Ok",
4   "copyright": "© 2024 MARVEL",
5   "attributionText": "Data provided by Marvel. © 2024 MARVEL",
6   "attributionHTML": "<a href=\"http://marvel.com\">Data provided by Marvel. © 2024 MARVEL</a>",
7   "etag": "3cad81eb071614ff249b73ecef3d79de180b0c",
8   "data": {
9     "offset": 0,
10    "limit": 20,
11    "total": 1564,
12    "count": 20,
13    "results": [
14      {
15        "id": 1011334,
16        "name": "Marvel Previews (2017)"
17      }
18    ]
19  }
20}
  
```

Get characters

```

1 {
2   "code": 200,
3   "status": "Ok",
4   "copyright": "© 2024 MARVEL",
5   "attributionText": "Data provided by Marvel. © 2024 MARVEL",
6   "attributionHTML": "<a href=\"http://marvel.com\">Data provided by Marvel. © 2024 MARVEL</a>",
7   "etag": "3cad81eb071614ff249b73ecef3d79de180b0c",
8   "data": {
9     "offset": 0,
10    "limit": 20,
11    "total": 1564,
12    "count": 20,
13    "results": [
14      {
15        "id": 1011334,
16        "name": "Marvel Previews (2017)"
17      }
18    ]
19  }
20}
  
```

ÉTAPE 3 – TU VEUX MA PHOTO ?

Après avoir réussi à accéder à l'API Marvels depuis Postman, l'exercice consiste donc à y accéder depuis une application Node.js à travers un navigateur. Pour ce faire j'ai reconstruit le mécanisme d'authentification en utilisant les éléments du pre-script dans Postman.

- La première étape a été de compléter les fonctions getData() et getHash() dans le fichier api.js, en utilisant les même spécifications que celles sur Postman. Ces fonctions sont responsables de la récupération des données depuis l'API Marvels en générant le timestamp, le hash MD5 et effectue la requête.
- Ensuite, il a été question de filtrer les résultats pour s'assurer que seuls les personnages avec une image 'thumbnail' valide soient inclus dans les données, et ce en vérifiant que le chemin de l'image n'est pas égal à 'image_not_available'.
- Ensuite j'ai créé un tableau charactersData de personnage, en ne gardant que les informations pertinentes telles que le nom, la description, le 'thumbnail' et l'url de l'image. À savoir que pour l'url de l'image, j'ai construit le chemin complet en utilisant d'une part le chemin du 'thumbnail' et d'autre part l'extension de l'image.

• Réussites :

- L'utilisation de 'node-fetch' pour la réalisation de requêtes asynchrones à l'API Marvel a été fluide
- Le filtrage des résultats pour éliminer les personnages sans images a été un succès.
- La création du tableau de personnage avec les informations nécessaires est également un succès.

• Difficultés :

- J'ai eu quelques difficultés avec la récupération du code de démarrage et avec la mise en place du projet Github.

```
// Test avec les personnages
const apiUrl :string = 'https://gateway.marvel.com/v1/public/characters?limit=10';

(async () :Promise<void> => {
    try {
        const charactersData :json = await getData(apiUrl);
        console.log('Characters Data:', charactersData);
    } catch (error) {
        console.error('Error:', error);
    }
})();
```

Code du server.js pour tester la récupération des données via la requête (ici les personnages)

```
Characters Data: [
  {
    name: '3-D Man',
    description: '',
    thumbnail: {
      path: 'http://i.annihil.us/u/prod/marvel/i/mg/c/e0/535fecbbb9784',
      extension: 'jpg'
    },
    imageUrl: 'http://i.annihil.us/u/prod/marvel/i/mg/c/e0/535fecbbb9784/portrait_xlarge.jpg'
  },
  {
    name: 'A-Bomb (HAS)',
    description: "Rick Jones has been Hulk's best bud since day one, but now he's more than a friend...he's a teammate! Transformed by a Gamma energy explosion, A-Bomb's thick, armored skin is just as strong and powerful as it is blue. And when he curls into action, he uses it like a giant bowling ball of destruction!",
    thumbnail: {
      path: 'http://i.annihil.us/u/prod/marvel/i/mg/3/20/5232158de5b16',
      extension: 'jpg'
    }
  }
]
```

Résultats, tableau de personnages

ÉTAPE 4 – ON S'ACCROCHE AU GUIDON

L'exercice suivant a pour objectif de rendre visuellement les données récupérées depuis l'API Marvel sur une page web, pour ce faire il a été question de mettre en place l'utilisation de Fastify et de son plugin @Fastify/view avec Handlebars.

- Tout d'abord j'ai travaillé sur le fichier server.js, où dans un premier temps j'ai créé une instance de Fastify qui permet de gérer le serveur.
- Ensuite, j'ai enregistré le plugin fastify-view dans l'instance Fastify, ce qui permet d'intégrer la gestion des vues dans l'application.
- Concernant la configuration du plugin, j'y ai précisé le moteur de template à utiliser, soit handlebars. J'y ai ensuite ajouté les partials comme demandé, qui seront inclus dans le fichier modèle.
- Ensuite, une partie de code permet la définition de la route, récupérant les données depuis l'API pour les transmettre à la vue.
- Le serveur est ensuite lancé sur le port 3000
- Puis, concernant le fichier index, celui-ci contient dans un premier temps des inclusions des partials définies dans le server, aux endroits où ils apparaîtront dans le fichier (header et footer). Ensuite dans ce fichier, j'ai utilisé une boucle Handlebars qui itère sur chaque éléments du tableau 'characters', où chaque itération désigne un personnage, ce qui permet l'affichage de ces personnages avec les informations voulu.

• Réussites :

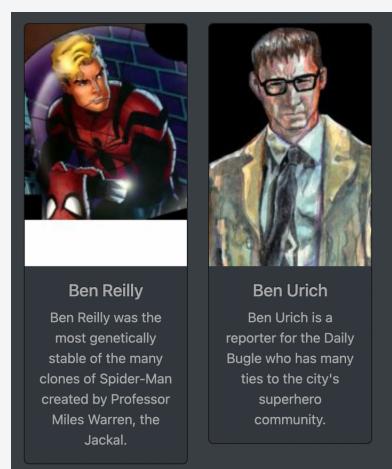
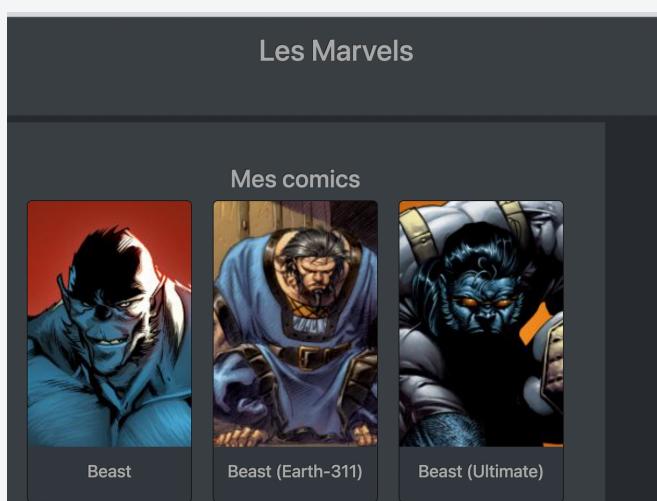
- Les données récupérées depuis l'API Marvel sont transmises à la vue et affichées correctement sur la page.

• Difficultés :

- J'ai eu quelques difficultés avec l'utilisation de Fastify.
- Des problèmes liés aux chemins des différents fichiers.

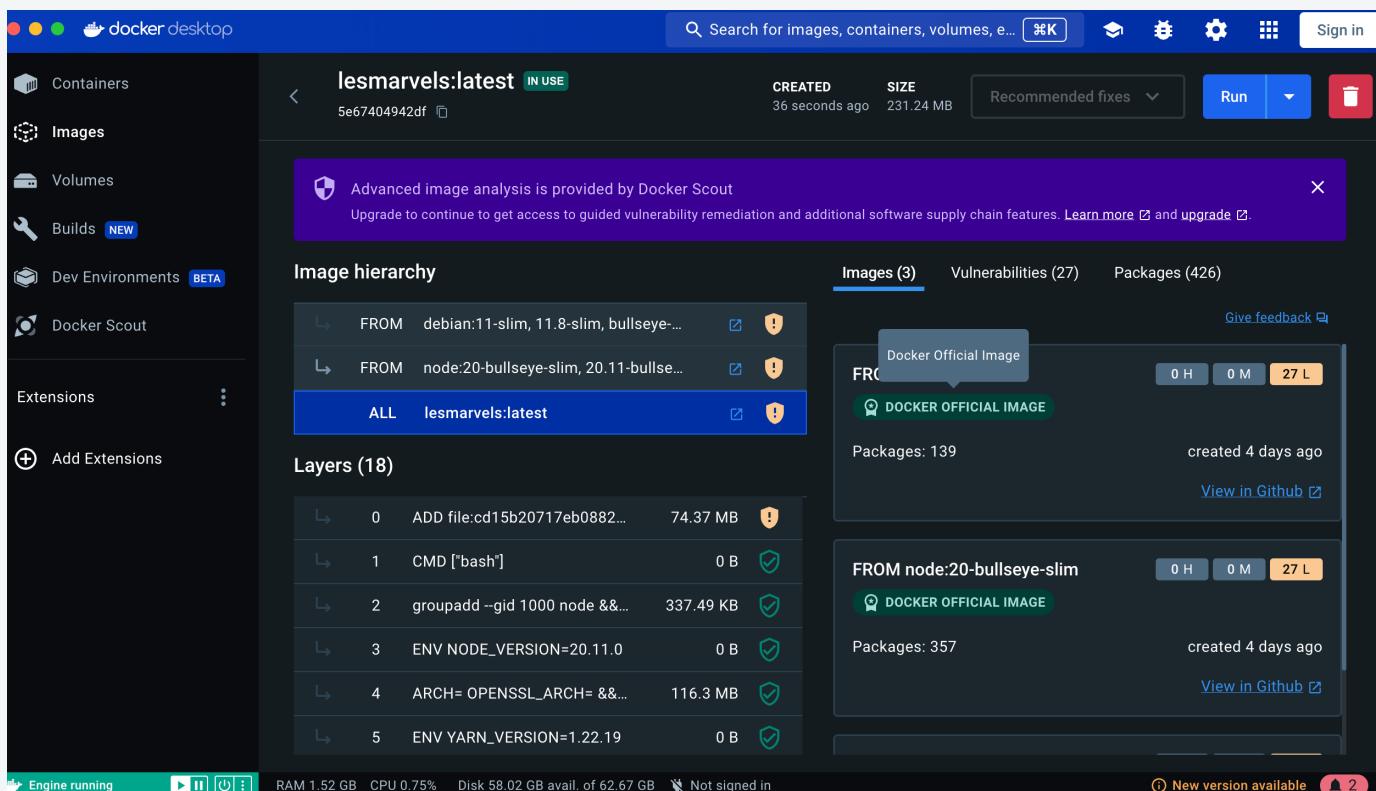
• Améliorations :

- Une amélioration possible serait d'ajouter une fonctionnalité de navigation à travers les pages en utilisant l'offset. Cela permettrait aux utilisateurs de parcourir différentes pages de personnages.



ÉTAPE 5 – C'EST DANS LA BOITE !

Pour cette exercice permettant la contenirisation de l'application avec Docker, toutes les étapes ont été suivies avec succès. L'application web Marvels fonctionne correctement dans le conteneur Docker, et l'image a été créée avec succès. Cet étape permet de faciliter le déploiement de l'application sur différentes machines, indépendamment des détails d'installation de Node.js et de ses modules.



L'image lesmarvels de notre application a bien été créée

Containers						
Container CPU usage		Container memory usage		Show charts		
0.00% / 800% (8 cores available)		39.39MB / 3.75GB				
	Search					
	Name	Image	Status	CPU (%)	Port(s)	Last started
<input type="checkbox"/>	unruffled_le e7f20616824c	lesmarvels	Running	0%	3000:3000	47 minutes ago

Containers de notre application