

Rapport de développement « API rest Ruby rails »

Projet réalisé et développer
par Théo Michellon

Sommaire

Table des matières

I) 🚧 Contexte.....	3
II) 🎯 Objectifs	3
III) ✔ Exigences.....	3
IV) Réalisation du projet.....	3
a) Installation du projet	4
b) Création du modèle et du controller	4
c) Ajout des routes - CRUD	4
d) Configuration des retours Api :.....	5
e) Modification affichage rendu :.....	6
f) Test API	6
g) Postamnn Script.....	8
A votre tour de jouer	8

I) 🚀 Contexte

Ce projet consiste à développer une API en Ruby on Rails qui gère deux types de données : les Pokémon, retournant des données au format JSON, et les champions de League of Legends (LoL), retournant des données au format XML. Ce projet peut être réalisé individuellement ou en groupe.

II) 🎯 Objectifs

- ❖ API Pokémon (JSON): Créer des routes pour gérer les informations sur les Pokémon (CRUD) et retourner les réponses en format JSON. 🐾
- ❖ API Champions de LoL (XML): Mettre en place des routes pour gérer les données des champions de LoL (CRUD) avec des réponses en format XML. 🍷

Pour les attributs des Pokémon et des champions de LoL, vous pouvez vous référer aux exemples ci-dessous :

Exemple de données pour un Pokémon

```
{
  "id": 1,
  "name": "Bublizarre",
  "type": "Grass",
  "level": 5,
  "evolution": "Herbizarre"
}
```

Exemple de données pour un champion de LoL

```
<champion>
  <id>1</id>
  <name>Graves</name>
  <role>Jungler</role>
  <difficulty>4</difficulty>
  <price>6300</price>
</champion>
```

III) ✅ Exigences

- ❖ Utiliser Ruby on Rails pour le backend. 💎
- ❖ L'API doit respecter les principes RESTful. 🔄
- ❖ Documenter clairement les routes, incluant des exemples de requêtes et de réponses pour chaque type de données. 📄

IV) 🏗️ Réalisation du projet

Le projet a été réalisé selon les étapes suivantes :

- Installation du projet
- Création du modèle et controller
- Ajout des routes
- Configuration des retours API
- Modification affiche des retours
- Test API
- Script Postman

a) Installation du projet

Une fois l'installation et la configuration de Ruby on rails effectuée vous pouvez :
Créer un nouveau répertoire pour le projet. Depuis une invite de commande à l'intérieur. Exécutez la commande suivante pour initialiser votre projet. (Pour lancer le serveur il suffit de lancer la commande : « rails server » dans un cmd à la racine du projet)

```
rails new mon_app  
cd mon_app
```

b) Création du modèle et du controller

Afin de réaliser cette étape j'ai utilisé les commandes « scaffold »

Après avoir exécuté cette commande, Rails générera automatiquement les fichiers suivants :

- ❖ Un modèle Pokemon/Champion avec les attributs spécifiés dans votre commande.
- ❖ Un contrôleur Pokemons/Champion avec des actions CRUD standard pour manipuler les données Pokemon.
- ❖ Des fichiers de vues pour afficher et manipuler les données Pokemon/Champion.
- ❖ Un fichier de migration de base de données pour créer la table Pokemon/Champion dans la base de données.

```
rails generate scaffold Pokemon name:string pokemon_type:string size:float weight:float date_capture:datetime
```

```
rails generate scaffold Champion name:string pv:float damage:float speed:float shield:float
```

c) Ajout des routes- CRUD

- Create (Créer) - Route POST :
 - Endpoint : /pokemons
 - Description : Cette route est utilisée pour créer une nouvelle ressource. Les données sont envoyées sous le format JSON
- Read (Lire) - Routes GET :
 - Endpoint : /pokemons (ou /pokemons/:id pour une ressource spécifique)
 - Description : Ces routes permettent de récupérer des informations sur une ou plusieurs ressources. Si aucun identifiant n'est spécifié alors toutes les données de la route sont renvoyées
- Update (Mettre à jour) - Route PUT ou PATCH :
 - Endpoint : /pokemons/:id
 - Description : Cette route est utilisée pour mettre à jour une ressource existante.
- Delete (Supprimer) - Route DELETE :
 - Endpoint : /pokemons/:id
 - Description : Cette route est utilisée pour supprimer une ressource spécifique.

Au cours du projet j'ai donc réalisé les routes suivantes :

- ❖ POST /pokemons : Créer un nouveau pokemon. Avec la méthode « create »
- ❖ GET /pokemons : Récupérer la liste complète des pokemons. Avec la méthode « index »
- ❖ GET /pokemons /:id : Récupérer un pokemon en particulier. Avec la méthode « show »
- ❖ PUT /pokemons /:id : Modifier les données d'un pokemon existante. Avec la méthode « update »
- ❖ DELETE /pokemons /:id : Supprimer un pokemon spécifique. Avec la méthode « destroy »

Il n'est pas nécessaire de définir les routes dans le fichier du controller lorsque ces dernières sont correctement définies.

d) Configuration des retours Api :

Au cours du projet une exigence était demandée ; Il s'agit de réaliser les retours Api de Pokemon en JSON et ceux des champions de LOL en XML pour réaliser cela j'ai développé les parties suivantes :

Rendu avec LOL en XML lors du render

```
# POST /champions
# Creates a new champion based on the provided parameters.
def create
  # Initialize a new Champion object with parameters from the request.
  @champion = Champion.new(champion_params)

  # Check if the champion is successfully saved to the database.
  if @champion.save
    # Render the details of the created champion in XML format.
    render xml: @champion.as_json
  else
    # Render errors in XML format if there are validation errors.
    render xml: @champion.errors.as_json
  end
end
```

Rendu avec Pokemon en JSON lors du render

```
# POST /pokemons
# Creates a new pokemon based on the provided parameters.
def create
  # Initialize a new Pokemon object with parameters from the request.
  @pokemon = Pokemon.new(pokemon_params)

  # Check if the pokemon is successfully saved to the database.
  if @pokemon.save
    # Render the details of the created pokemon in JSON format.
    render json: @pokemon, status: :created, location: @pokemon
  else
    # Render errors in JSON format if there are validation errors.
    render json: @pokemon.errors, status: :unprocessable_entity
  end
end
```

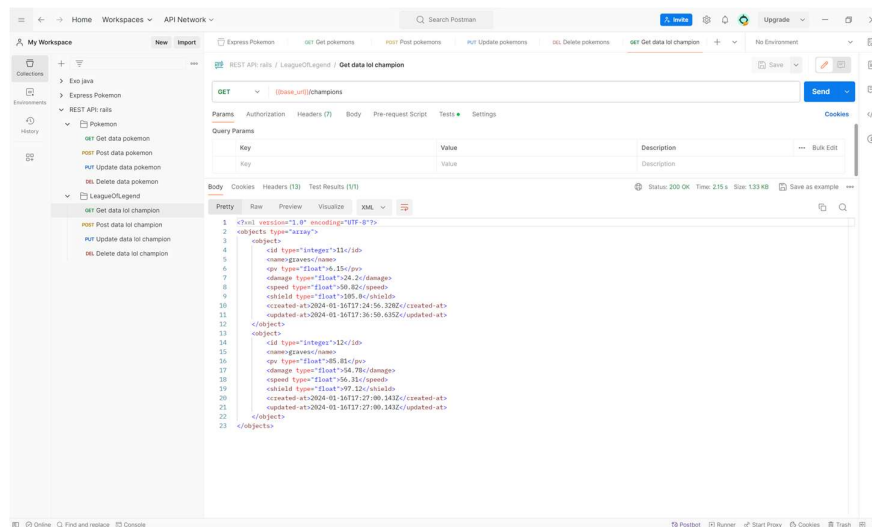
e) Modification affichage rendu :

Lors du développement je me suis aperçu que je recevais des balise inutile en XML j'ai donc développé le correctif suivant dans le but de modifier mon rendu :

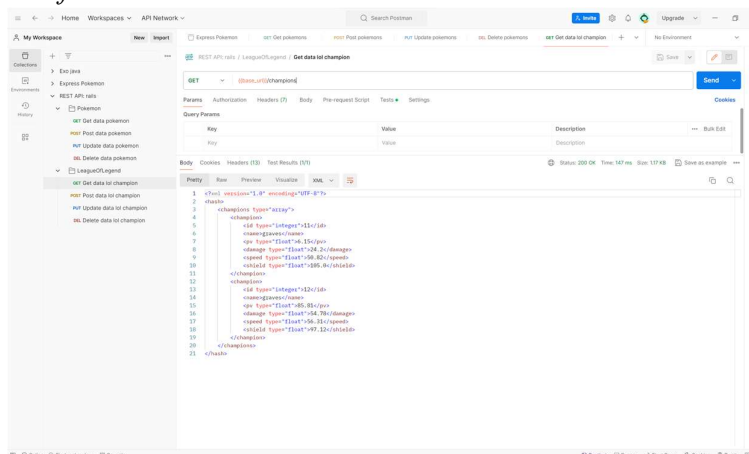
```
# GET /champions
# Retrieves a list of all champions.
def index
  # Fetch all champions from the database.
  @champions = Champion.all

  # Render the list of champions in XML format.
  render xml: { champions: @champions.as_json(except: [:created_at, :updated_at]) }
end
```

Avant la modification dans le code :



Après l'ajout de la modification



f) Test API

Postman est une plateforme API permettant de créer et d'utiliser des API. Postman simplifie chaque étape du cycle de vie des API et rationalise la collaboration afin que vous puissiez créer de meilleures API plus rapidement.

Vous trouverez ci-dessous deux exemple d'utilisation de cette API :

The image displays two screenshots of the Postman API client interface, demonstrating API calls to a League of Legends API.

Top Screenshot: GET Request

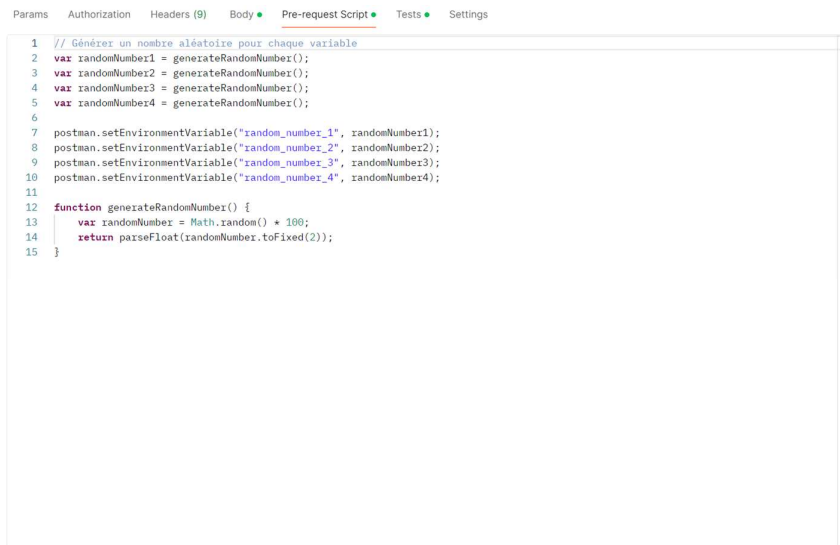
- Method:** GET
- URL:** `{{base_url}}/champions`
- Query Params:** A table with two columns: Key and Value. The first row shows 'Key' and 'Value'.
- Body:** The response is shown in the 'Pretty' tab as an XML document. It contains a root element `<?xml version="1.0" encoding="UTF-8"?>` followed by a `<hash>` element. Inside the hash, there is a `<champions type="array">` element containing two `<champion>` elements. Each champion element contains attributes for `<id type="integer">`, `<name>`, `<pv type="float">`, `<damage type="float">`, `<speed type="float">`, and `<shield type="float">`.

Bottom Screenshot: PUT Request

- Method:** PUT
- URL:** `{{base_url}}/champions/11`
- Body:** The request body is shown in the 'Raw' tab as a JSON object. It contains a `"name": "graves"`, a `"pv": {{random_number_1}}`, a `"damage": 24.2`, a `"speed": {{random_number_3}}`, and a `"shield": 105.0`.
- Body:** The response is shown in the 'Pretty' tab as an XML document. It contains a root element `<?xml version="1.0" encoding="UTF-8"?>` followed by a `<hash>` element. Inside the hash, there is a `<name>` element, a `<pv type="float">` element, a `<damage type="float">` element, a `<speed type="float">` element, a `<shield type="float">` element, a `<id type="integer">` element, a `<created-at>` element, and a `<updated-at>` element.

g) Postman Script

Afin de faciliter la saisie de données, j'ai créé différents scripts permettant d'initialiser des valeurs au cours des projets avec par exemple :



```
1 // Générer un nombre aléatoire pour chaque variable
2 var randomNumber1 = generateRandomNumber();
3 var randomNumber2 = generateRandomNumber();
4 var randomNumber3 = generateRandomNumber();
5 var randomNumber4 = generateRandomNumber();
6
7 postman.setEnvironmentVariable("random_number_1", randomNumber1);
8 postman.setEnvironmentVariable("random_number_2", randomNumber2);
9 postman.setEnvironmentVariable("random_number_3", randomNumber3);
10 postman.setEnvironmentVariable("random_number_4", randomNumber4);
11
12 function generateRandomNumber() {
13     var randomNumber = Math.random() * 100;
14     return parseFloat(randomNumber.toFixed(2));
15 }
```

V) A votre tour de jouer

Pour cela rien de plus simple installer le repository ; importer le fichier de config Postman dans votre logiciel et amusez-vous.