Adoptez les API REST pour vos projets web

4 heures Facile

Mis à jour le 29/01/2024

Utilisez les ressources et collections REST

01:37

Maintenant que vous savez que les API REST servent d'intermédiaires et aident les développeurs à manipuler des données, regardons de plus près à quoi ressemblent réellement ces données.

Appréhendez les données REST via l'utilisation des ressources

Les données REST sont représentées dans ce qu'on appelle des ***ressources***.

Une ressource peut être tout type d'objet **nominal** (on lui attribue un nom) que vous pouvez utiliser pour représenter les données dans votre application. Vous savez, une personne, un lieu, ou autre chose ! 😉 Pour faire simple, voyez les ressources comme des boîtes dans lesquelles vous rangerez des objets par catégorie et sur lesquelles vous collez une étiquette pour savoir quoi mettre dedans.

Vous trouvez que c'est abstrait ? C'est le but, afin que vous puissiez représenter n'importe quel élément de donnée sous la forme que vous souhaitez.

Chaque **ressource** comporte des informations supplémentaires sur les données contenues. Si on prend l'exemple d'une application qui liste les héros Marvel, une des ressources pourrait être *Superhero* et on pourrait avoir par exemple un nom, une description, etc., comme information supplémentaire.

Les ressources sont regroupées dans un groupe que l'on appelle une **collection**. On s'y réfère avec la forme au **pluriel** du nom de la ressource. Par exemple une ressource superhero donnerait superheroes.

> Par convention, tous les champs d'une ressource et le nom d'une collection sont en anglais. Ils sont traduits ici en français pour une meilleure compréhension du cours, mais privilégiez toujours l'anglais !

Imaginons que vous créez une API pour qu'une boutique de skateboards ait un service de livraison en ligne. Ce que vous voulez, c'est que d'un côté vos clients puissent acheter des skateboards sur le site web, et de l'autre que vos salariés puissent ajouter des produits et mettre l'inventaire à jour.

Procédons étape par étape et déterminons ensemble les ressources, leurs informations supplémentaires et les collections.

Pour **une boutique de skateboards,** vos ressources pourraient être :

- Client ;
- Staff ;
- Basket (panier) ;
- Skateboard ;
- Inventory (inventaire).

Une ressource *Skateboard* pourrait comporter comme informations supplémentaires : nom, marque, id, prix.

Vos collections seraient donc :

Ressource	Collection
skateboard	skateboards
client	clients

Cette liste est un exemple et ne contient pas tous les exemples cités précédemment.

Bien ! Nous avons nos collections ainsi que les ressources correspondantes et leurs informations supplémentaires. Continuons avec notre exemple de boutique de skateboards et suivons ensemble le parcours de la requête d'un client via notre API.

Quand un client achète un skateboard en utilisant votre application web, cela donne :

- votre API envoie la requête du navigateur (le client) aux serveurs de l'application pour l'achat d'un skateboard ;
- la requête met à jour l'**inventaire** pour qu'il y ait un skateboard de moins ;
- la requête met à jour l'historique de commandes du **client** pour ajouter le skateboard à son historique d'achats. 😁

Super ! Vous savez maintenant comment et sous quelle forme *stocker* les données que vous voulez utiliser dans votre API via des ressources et des collections. Mais du coup, comment pouvez-vous y *accéder* ? Comment savoir où les *récupérer*, ces données ?

URI et Endpoints

Le **path (ou *chemin*)** que vous donnez à votre API lui permet de savoir exactement **où** se trouvent les données que vous voulez récupérer. Vous pouvez imaginer cela comme le fait de parcourir vos propres fichiers sur votre ordinateur. Vous devez aller de dossier en dossier pour trouver vos données, et chaque photo ou document que vous sauvegardez a son propre path, ou *chemin de fichier*, en français. Par exemple, votre photo de série préférée pourrait se trouver au bout de ce path :

```
MyComputer/Images/Series/gameofthrones.jpg
```

Les API REST stockent également les données de façon similaire, et un URI constitue le chemin pour y arriver.

Si une ressource est l'objet qui stocke vos données, pour les récupérer vous allez avoir besoin d'un identifiant de ressource uniforme, ou ***URI*** pour ***U****niform* ***R****esource* ***I****dentifier*. L'URI est le moyen d'identifier votre ressource, comme une étiquette.

Imaginons que vous créez une API pour un site web qui présenterait toutes les informations de Game of Thrones, que ce soit sur le livre ou la série. L'URI qui listerait tous les personnages pourrait être la suivante :

```
/characters
```

Si vous voulez voir les informations sur *un seul* personnage, qui porte l'ID 123, votre URI serait le suivant :

```
/characters/123
```

Tout comme les paths pour les fichiers, les URI peuvent avoir des ressources **imbriquées**. Si vous voulez obtenir uniquement *le nom* du personnage qui vous intéresse, votre URI pourrait ressembler à ceci :

```
/characters/123/description
```

Wouhou ! Voilà du progrès ! Le souci, c'est que sans l'adresse réelle du site web, l'API ne saura pas du tout où chercher l'URI pour commencer ! C'est là que les **endpoints** (ou *points de terminaison*, en français) interviennent !

Un endpoint est une URL/URI qui fait partie d'une API. Si un URI est comme un chemin de fichier, alors un endpoint est comme l'adresse complète du fichier. Il vous suffit d'ajouter votre **nom de domaine** au début de votre URI, et vous avez un endpoint ! Par exemple, si le nom de domaine de notre app est *gameofthrones-informations.com*, nous aurons :

```
https://gameofthrones-informations.com/characters
```

> Houla, attends deux secondes, c'est quoi la différence entre URI et URL ?

Toutes les URL sont des URI, mais toutes les URI ne sont pas des URL 🤯. L'URI permet d'identifier une ressource tandis que l'URL permet de la localiser.

On confond souvent les deux. On va tout simplifier avec un exemple ! Reprenons notre site de Game of Thrones. Si le personnage de Jon Snow a pour ID 890, alors l'URI serait /characters/890. L'URL serait https://gameofthrones-informations.com/characters/890

L'URL de la requête est l'endpoint complet que vous utilisez pour votre requête. Il associe le nom de domaine + le path de votre ressource. À présent, vous savez comment accéder aux données que vous souhaitez !

Au fait, pas besoin de créer votre propre API de GoT – il en existe déjà une et vous pouvez la découvrir ici : **anapioficeandfire.com**. ⚔️

Distinguez XML et JSON

Une fois que vous avez le bon endpoint sur lequel faire votre requête, il est temps pour vous d'obtenir vos données ! C'est là que vous obtenez les informations sur les ressources que vous avez créées.

Le terme *données* est un terme général qui décrit toute information envoyée ou reçue, tandis que que le terme *ressource* décrit plus précisément les **éléments** qui sont contenus dans cette information.

Les données des API REST peuvent utiliser deux langages : XML et JSON. Si une API renvoie un set de données en XML ou en JSON, le contenu restera le même, mais la forme change. Le format de données est différent.

Le XML

En **XML**, chaque élément de donnée a une balise ouvrante et une balise fermante qui peut également avoir des balises imbriquées :

xml

```xml
<series>
  <serie>    <titre>Game Of Thrones</titre>
    <realisateur>Random</realisateur>
  </serie>
       <serie>    <titre>Peaky Blinders</titre>
<realisateur>Random</realisateur>  </serie>
</series>


<series>
  <serie>
     <titre>Game Of Thrones</titre>
     <realisateur>Random</realisateur>
  </serie>
  <serie>
     <titre>Peaky Blinders</titre>
     <realisateur>Random</realisateur>
  </serie>
</series>
```

Vous pouvez voir une balise ouvrante pour la **collection** en haut – series – entre crochets < >. La balise fermante à la fin est identique, sauf qu'on y ajoute une barre oblique / au début : </series> , qui indique que c'est une balise fermante. Chaque **ressource** listée a la balise ouvrante <serie> et la balise fermante </serie> . Au sein de chaque ressource se trouvent davantage d'informations, comme "titre" et "réalisateur".

Le JSON

Le **JSON** stocke les données sous un format de clé-valeur avec comme clé le type de données, suivi de deux points **:** , suivi de la valeur de la donnée. Les données JSON sont entourées d'accolades { }, et chaque paire clé-valeur est envoyée comme chaîne de caractères avec des guillemets autour "".

Ce qui nous donne ceci :

json

```json
{ "titre" :"Game of Thrones"}
```

Les tableaux, ou listes, en JSON sont entourées de crochets []. L'exemple ci-dessous montre comment une liste complète peut être considérée comme la valeur de la clé "series". Les mêmes données en XML ci-dessus seraient représentées ainsi en JSON :

json

```json
{ "series": [
  {   "titre": "Game Of Thrones",
    "realisateur": "Alan Taylor" },  {   "titre": "Peaky Blinders",
```

```
4     "realisateur": "Otto Bathurst" }  ]
5 }
```

Le JSON est généralement considéré comme :

1. Plus facile à analyser avec du code.
2. Plus court.
3. Plus rapide à lire et à écrire pour les machines.
4. Très "léger" et efficace grâce à sa structure en arborescence et sa syntaxe simple.

Voici quelques exemples d'API réelles qui renvoient du JSON et du XML :

- Penguin Random House : XML.
- Potter API : JSON.
- Ghibli API: JSON

Comme vous pouvez le constater, le JSON est le langage de données le plus utilisé ; c'est pour cette raison que nous l'utiliserons dans le reste de ce cours !

En résumé

- Une ressource est un objet de type nominal utilisé pour sauvegarder des données dans une API.
- Une ressource peut contenir des informations supplémentaires.
- Les ressources sont regroupées en collection et sont nommées au pluriel.
- Vous pouvez accéder aux ressources dans les API avec des URI.
- Les données REST peuvent être en langage JSON ou XML, mais le JSON est le plus courant.

Et maintenant, revenons sur tout ce que nous avons appris avec le quiz de cette première partie.

Une fois que vous aurez terminé ce quiz, je vous retrouve dans la seconde partie, dans laquelle nous allons sauter dans le grand bain et utiliser une API !

J'ai terminé ce chapitre et je passe au suivant

Être orienté

Comparez nos types de formation

Identifiez les avantages d'une API REST

Quiz : Servez-vous des API REST pour vos projets de code

Les professeurs

Kassandre Pedro

Backend Engineer at Pexels | Ruby on Rails Developer

Raye Schiller

Raye Schiller is a backend software engineer based in New York City and has an MEng. in Computer Science from Cornell University 🙏💻