

Lab 4 : Création de services web RESTful avec Spring Boot

Rapport de Ousmane KA

Dans ce lab, nous allons créer un service web RESTful en utilisant Spring Data REST pour fournir des fonctionnalités CRUD automatiques, en s'appuyant sur l'application de base de données du lab précédent et en utilisant des IDEs comme IntelliJ IDEA ou Eclipse avec Spring Tool Suite.

Exercice 1

Le code source de l'exercice 1 est disponible au niveau du github :

https://github.com/Ousoka/AGL_Springboot_RESTful/tree/master/sb-online-shop-lab5

Les services web sont des applications qui échangent des données sur Internet via le protocole HTTP, basées sur diverses architectures partageant le même principe fondamental de communication.

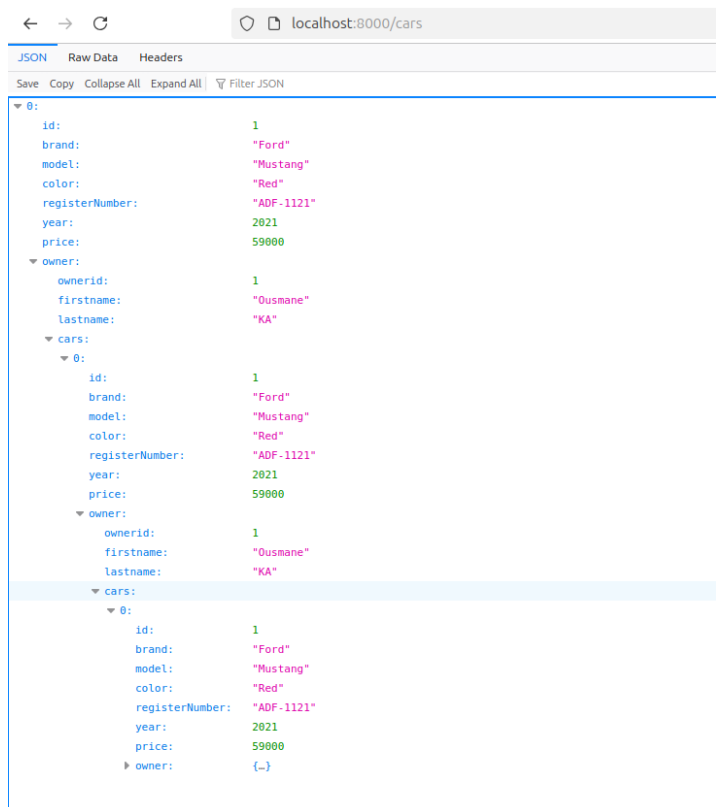
Les bases d'un service web RESTful

Representational State Transfer (REST) est un style architectural pour créer des services web, défini par six contraintes principales : stateless (pas de stockage d'état côté serveur), client-serveur (indépendance entre client et serveur), cacheable (réponses mises en cache pour optimiser les performances), interface uniforme (requêtes uniformes pour différents clients), système en couches (architecture modulaire), et code à la demande (optionnel). Une interface REST inclut l'identification des ressources par des URLs claires, la manipulation via des représentations (JSON ou XML), des messages auto-descriptifs, et HATEOAS (liens pour naviguer dans l'application). Le service web RESTful développé dans ce lab applique ces principes.

Création d'un service web RESTful avec Spring Boot

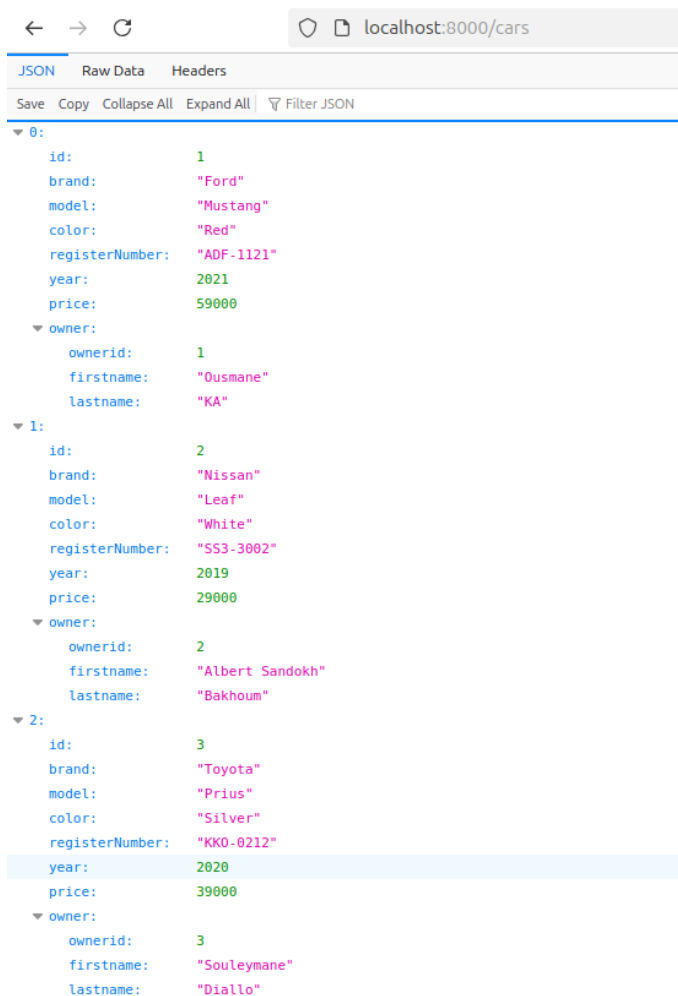
Pour créer un service web RESTful, un nouveau package `com.dsti.uam.sb-online-shop.web` est créé, contenant une classe `CarController` annotée avec `@RestController`, qui définit le contrôleur. La méthode `getCars()`, associée au point de terminaison `/cars` via `@RequestMapping` ou `@GetMapping`, gère les requêtes HTTP GET et renvoie une liste de voitures convertie en JSON grâce à la bibliothèque Jackson. En injectant `CarRepository`, la méthode `findAll()` permet de récupérer les voitures depuis la base de données. Enfin, en lançant l'application, le service est accessible via `localhost:8000/cars`.

[illegible]



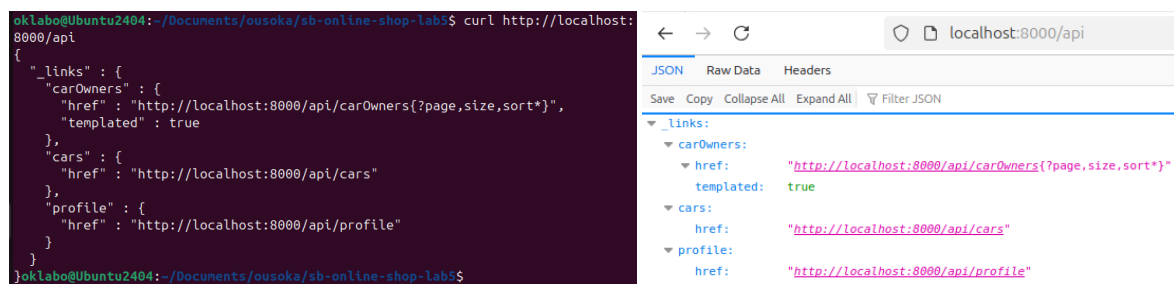
Une boucle infinie peut survenir en raison de la relation un-à-plusieurs entre les tables *voiture* et *propriétaire*. Lors de la sérialisation, une voiture inclut son propriétaire, qui inclut à son tour ses voitures, et ainsi de suite. Pour résoudre ce problème, l'annotation `@JsonIgnore` peut être appliquée au champ `cars` dans la classe `Owner` pour exclure ce champ de la sérialisation. De plus, `@JsonIgnoreProperties` permet d'ignorer les champs générés par Hibernate. Une fois ces ajustements effectués, le point de terminaison `/cars` renvoie correctement toutes les voitures en JSON via `localhost:8000/cars`. Après avoir implémenté ce service web RESTful, il est possible de supprimer la classe `CarController` pour explorer d'autres fonctionnalités de Spring Boot.

```
oklabo@Ubuntu2404:~/Documents/ousoka/sb-online-shop-lab5$ curl http://localhost:8000/cars
[{"id":1,"brand":"Ford","model":"Mustang","color":"Red","registerNumber":"ADF-1121","year":2021,"price":59000.00,"owner":{"ownerid":1,"firstname":"Ousmane","lastname":"KA"}},{ "id":2,"brand":"Nissan","model":"Leaf","color":"White","registerNumber":"SS3-3002","year":2019,"price":29000.00,"owner":{"ownerid":2,"firstname":"Albert Sandokh","lastname":"Bakhoum"}},{ "id":3,"brand":"Toyota","model":"Prius","color":"Silver","registerNumber":"KKO-0212","year":2020,"price":39000.00,"owner":{"ownerid":3,"firstname":"Souleymane","lastname":"Diallo"}}]oklabo@Ubuntu2404:~/Documents/ousoka/sb-online-shop-lab5$
```

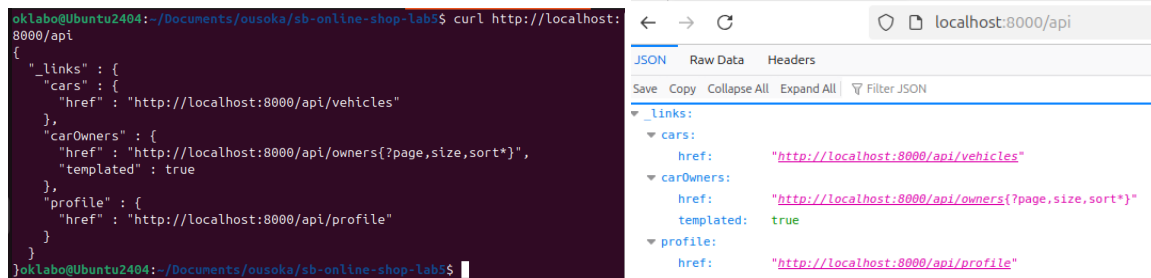


Utilisation de Spring Data REST

Spring Data REST, partie du projet Spring Data, permet de créer rapidement des services web RESTful en générant automatiquement des endpoints pour les entités définies dans les répertoires publics, comme CarRepository et OwnerRepository. Après avoir ajouté la dépendance au fichier pom.xml et configuré l'endpoint dans application.properties, vous pouvez accéder aux services via localhost:8000/api. Les données renvoyées sont au format JSON HAL (Hypertext Application Language), facilitant l'intégration avec des applications frontend grâce à l'inclusion d'hyperliens standardisés.



Spring Data REST génère des liens vers les services des entités en utilisant un nom de chemin dérivé du nom de la classe d'entité, converti en pluriel et non capitalisé (ex. : Car devient cars que l'on changera en vehicles). Le lien de profil inclut des métadonnées spécifiques à l'application. Pour personnaliser le nom de chemin, il est possible d'utiliser l'annotation `@RepositoryRestResource` dans la classe de répertoire correspondante.



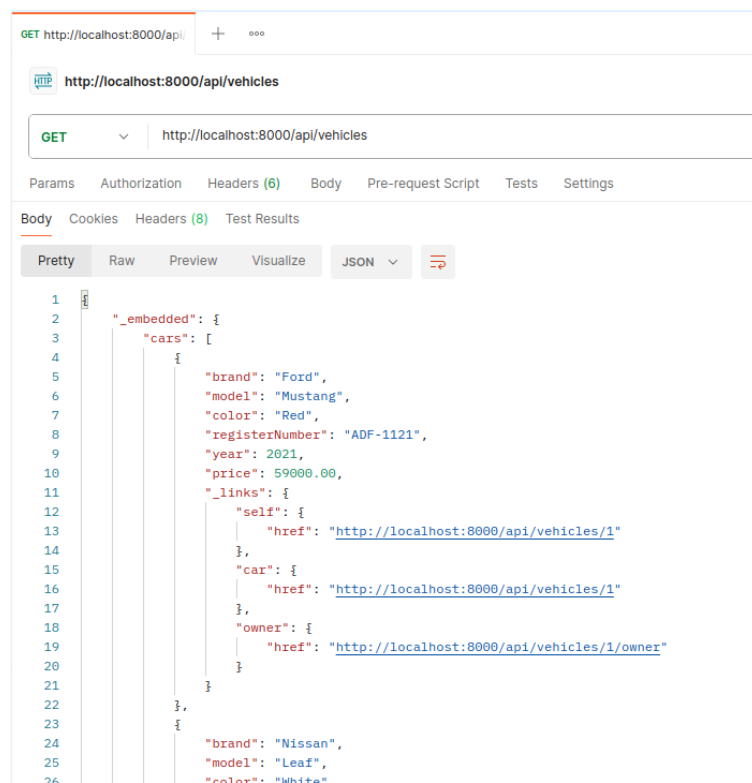
The image shows a terminal window on the left and a web browser on the right. The terminal displays the output of a curl command to http://localhost:8000/api, showing a JSON response with links to cars, car owners, and a profile. The browser on the right shows the same JSON response in its developer tools, with the links expanded to show their href values.

```
oklabo@Ubuntu2404:~/Documents/ousoka/sb-online-shop-lab$ curl http://localhost:8000/api
{
  "_links": {
    "cars": {
      "href": "http://localhost:8000/api/vehicles"
    },
    "carOwners": {
      "href": "http://localhost:8000/api/owners?page,size,sort*",
      "templated": true
    },
    "profile": {
      "href": "http://localhost:8000/api/profile"
    }
  }
}
oklabo@Ubuntu2404:~/Documents/ousoka/sb-online-shop-lab$
```

The browser shows the following JSON structure:

```
{
  "_links": {
    "cars": {
      "href": "http://localhost:8000/api/vehicles"
    },
    "carOwners": {
      "href": "http://localhost:8000/api/owners?page,size,sort*",
      "templated": true
    },
    "profile": {
      "href": "http://localhost:8000/api/profile"
    }
  }
}
```

Pour interagir avec les services RESTful générés par Spring Data REST, vous pouvez utiliser des outils comme Postman, Httpie ou Curl. En effectuant une requête GET à l'endpoint par défaut /cars (par exemple, via `http://localhost:8000/api/vehicles`), vous obtiendrez une liste de toutes les voitures disponibles. Les requêtes GET peuvent également être effectuées directement depuis un navigateur web.



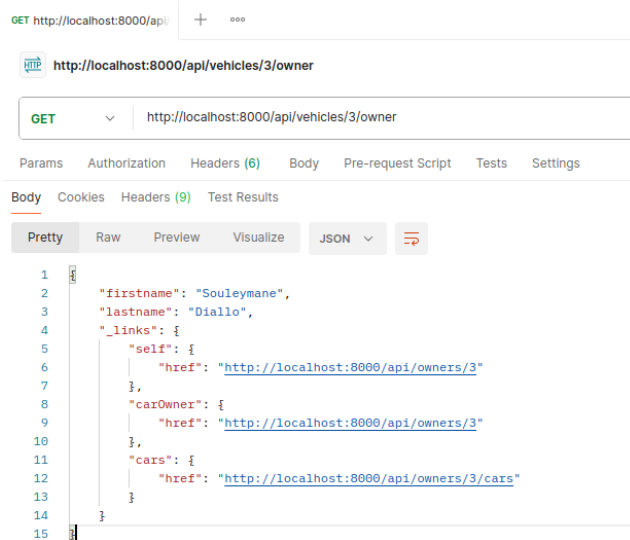
The image shows a Postman interface with a GET request to http://localhost:8000/api/vehicles. The response is a JSON array of car objects. Each car object has fields for brand, model, color, registerNumber, year, and price. It also has a _links object containing a self link and a car link. The first car is a Ford Mustang, and the second is a Nissan Leaf.

```
GET http://localhost:8000/api/vehicles
http://localhost:8000/api/vehicles
GET http://localhost:8000/api/vehicles

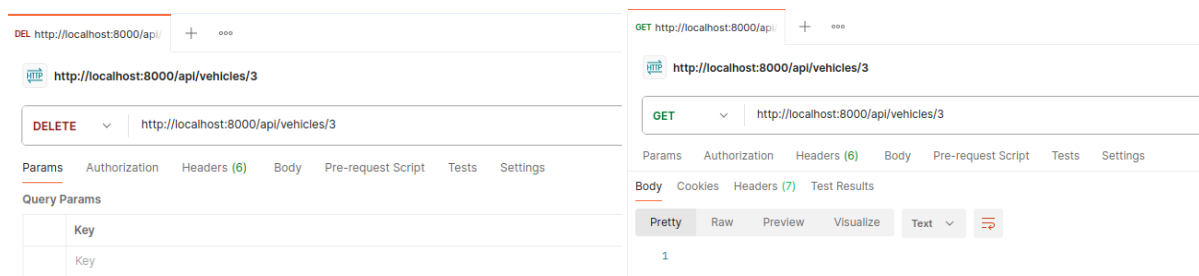
{
  "_embedded": {
    "cars": [
      {
        "brand": "Ford",
        "model": "Mustang",
        "color": "Red",
        "registerNumber": "ADF-1121",
        "year": 2021,
        "price": 59000.00,
        "_links": {
          "self": {
            "href": "http://localhost:8000/api/vehicles/1"
          },
          "car": {
            "href": "http://localhost:8000/api/vehicles/1"
          },
          "owner": {
            "href": "http://localhost:8000/api/vehicles/1/owner"
          }
        }
      },
      {
        "brand": "Nissan",
        "model": "Leaf",
        "color": "White"
      }
    ]
  }
}
```

Dans la réponse JSON de Spring Data REST, chaque voiture est représentée avec ses attributs spécifiques et un champ `_links` contenant des hyperliens. Ces liens permettent d'accéder à des ressources associées, comme les détails d'une voiture spécifique via `/api/vehicles/{id}` ou son propriétaire via `/api/vehicles/{id}/owner`. Par exemple, une requête GET à `/api/vehicles/3/owner`

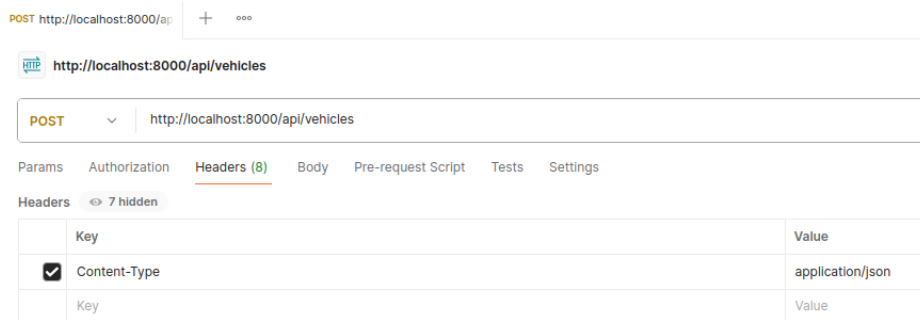
retourne les informations du propriétaire de la voiture avec l'ID 3, ainsi que des liens vers les autres voitures de ce propriétaire.



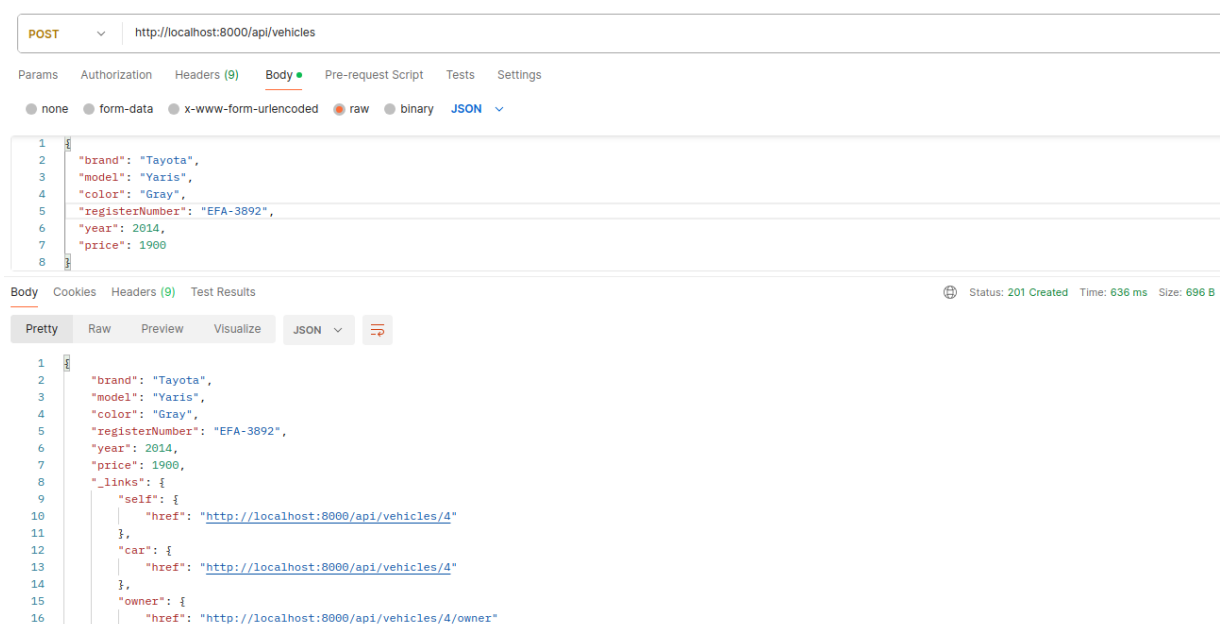
Spring Data REST supporte toutes les opérations CRUD via des méthodes HTTP standard. Par exemple, pour supprimer une voiture de la base de données, utilisez la méthode DELETE avec l'URL correspondant à la voiture ciblée, comme `/api/vehicles/{id}`. Avec Postman, sélectionnez la méthode DELETE, saisissez l'URL appropriée (en remplaçant `{id}` par l'identifiant de la voiture), puis cliquez sur *Envoyer*. Cette requête supprimera la voiture spécifiée de la base de données.



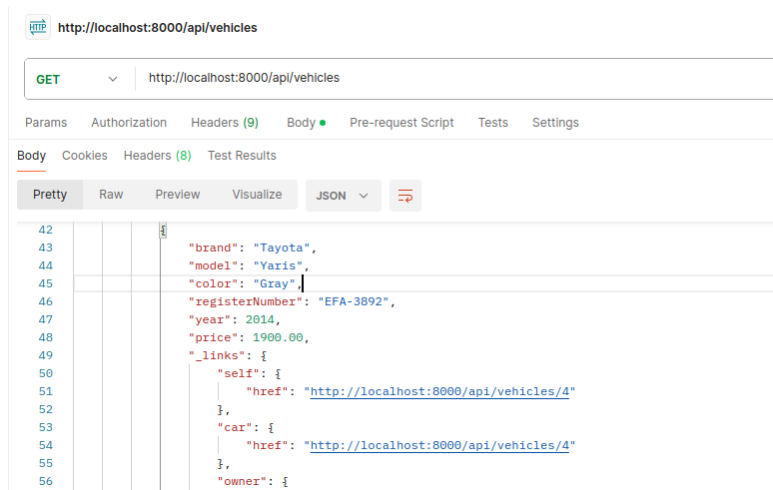
Lors de l'ajout d'une nouvelle voiture à la base de données avec Spring Data REST, vous utilisez la méthode HTTP **POST** avec l'URL `http://localhost:8000/api/vehicles`. Assurez-vous d'inclure l'en-tête Content-Type avec la valeur `application/json`. Le corps de la requête doit contenir l'objet de la voiture au format JSON. Si l'ajout est réussi, vous recevrez une réponse indiquant que la voiture a été ajoutée. Si vous vérifiez avec une requête **GET** à `http://localhost:8000/api/vehicles`, vous verrez la nouvelle voiture dans la liste des voitures disponibles.



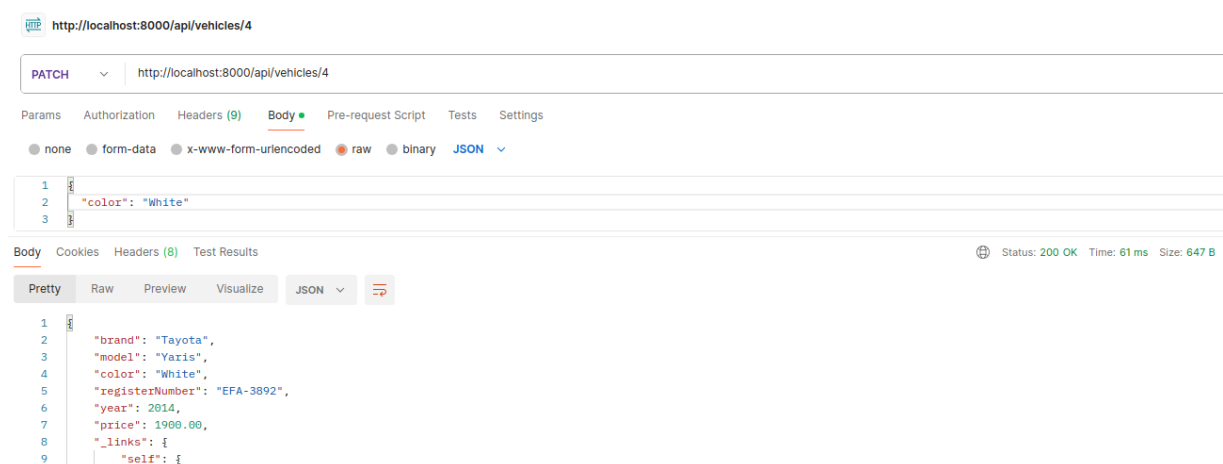
Dans Postman, pour ajouter une nouvelle voiture, cliquez sur l'onglet **Body**, sélectionnez **raw**, puis saisissez l'objet JSON représentant la voiture dans cet espace. Cela permet de formater et d'envoyer les données de la voiture au service web via une requête **POST**.



Après avoir ajouté une nouvelle voiture via une requête **POST**, la réponse renverra l'objet de la voiture créée ainsi qu'un code de statut **201 Created**. Si la requête est réussie, une requête **GET** à l'endpoint <http://localhost:8000/api/vehicles> affichera la voiture nouvellement ajoutée dans la base de données.

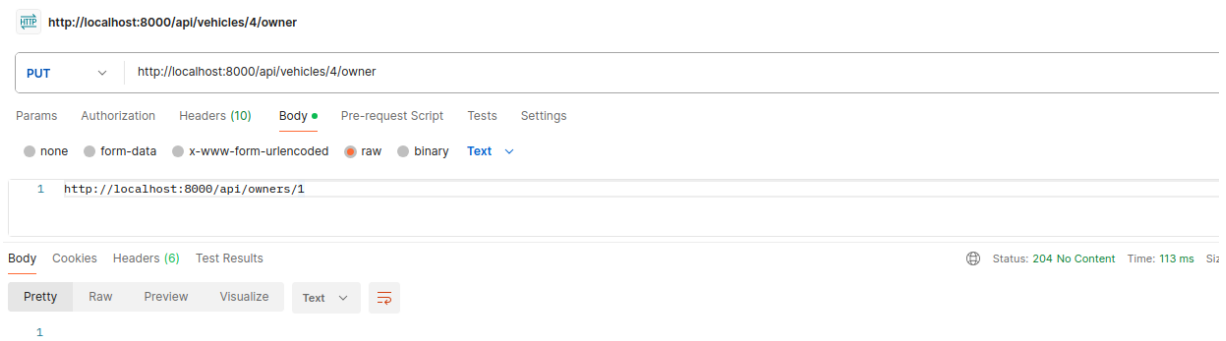


Pour mettre à jour une entité, on utilise la méthode **PATCH** avec l'URL de la voiture à modifier (`http://localhost:8000/api/vehicles/{id}`). L'en-tête doit spécifier **Content-Type: application/json**, et seuls les champs modifiés doivent être inclus dans le corps de la requête. En revanche, avec la méthode **PUT**, tous les champs doivent être envoyés. Par exemple, pour modifier la couleur d'une voiture en blanc, on enverra une requête **PATCH** avec les données mises à jour.

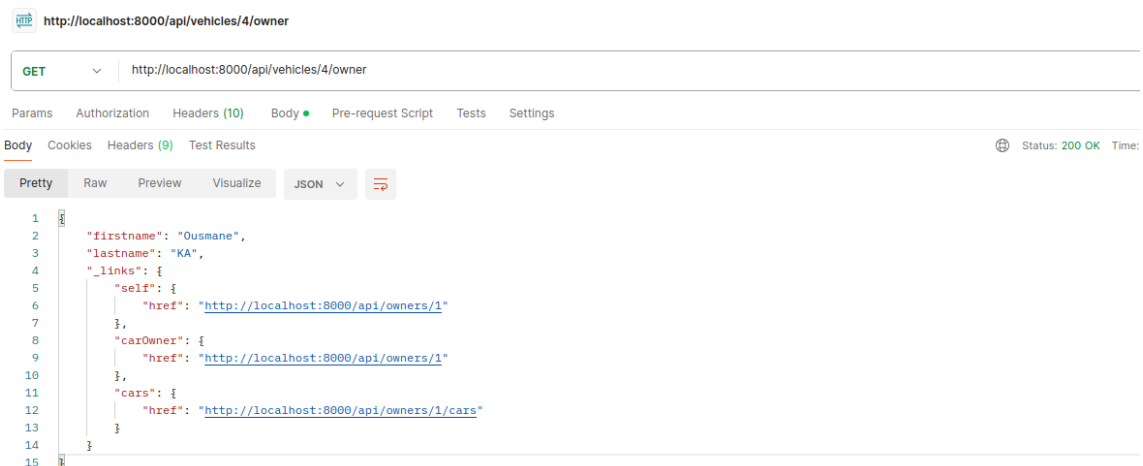


Après avoir effectué une mise à jour réussie d'une voiture, la réponse aura un statut **200 OK**. En faisant une nouvelle requête **GET**, vous pourrez vérifier que la modification (comme le changement de couleur) a bien été effectuée. Pour ajouter un propriétaire à une voiture, on utilise la méthode **PUT** avec l'URL `http://localhost:8000/api/vehicles/{id}/owner`, où `{id}` est l'identifiant de la voiture. Le corps de la requête contient l'URL du propriétaire à associer, et l'en-tête **Content-Type** doit être défini sur **text/uri-list**.

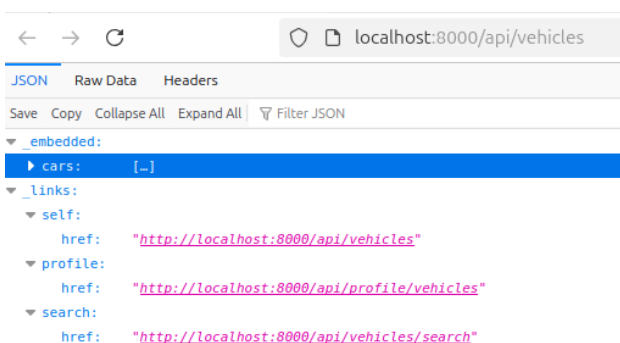
Key	Value
<input type="checkbox"/> Content-Type	application/json
<input checked="" type="checkbox"/> Content-Type	text/uri-list

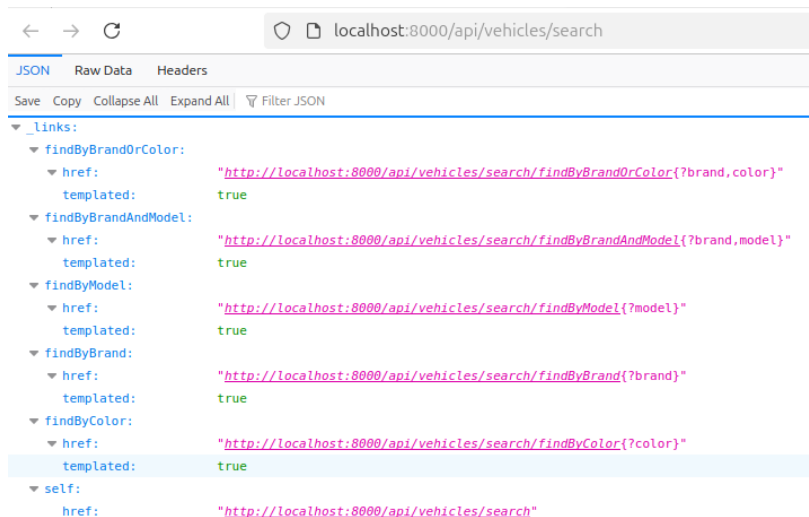


Après avoir effectué la requête **PUT** pour lier un propriétaire à la voiture, vous pouvez effectuer une requête **GET** pour vérifier que le propriétaire est bien associé à la voiture. La réponse indiquera que le lien entre la voiture et son propriétaire a été établi avec succès, montrant ainsi que les entités sont correctement reliées.

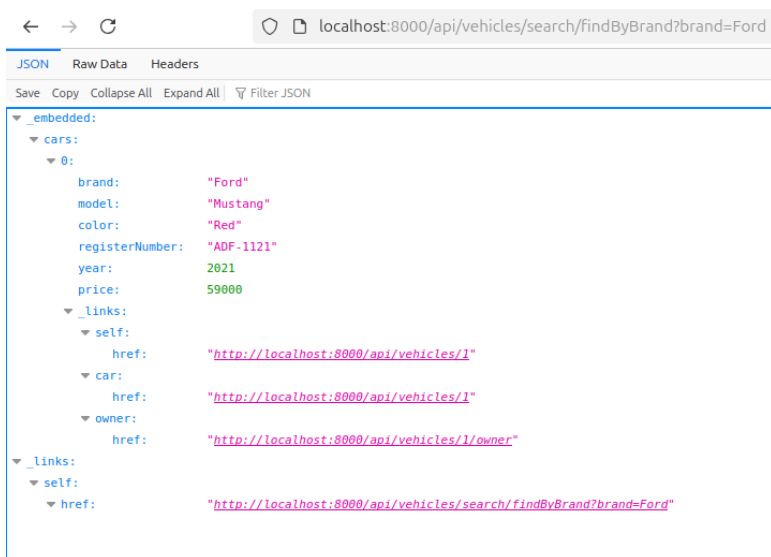


Dans le lab précédent, des requêtes ont été créées pour le repository. Ces requêtes peuvent être intégrées dans un service en ajoutant l'annotation `@RepositoryRestResource` à la classe du repository. Les paramètres des requêtes sont annotés avec `@Param`. Après avoir ajouté ces annotations, lorsqu'une requête GET est effectuée sur l'endpoint `http://localhost:8000/api/vehicles`, un nouvel endpoint `/search` devient accessible. En appelant `http://localhost:8000/api/vehicles/search`, la réponse renverra les résultats associés à la requête de recherche spécifiée.





À partir de la réponse, il est possible de constater que deux requêtes sont désormais disponibles dans le service. Par exemple, pour récupérer des voitures par marque, vous pouvez utiliser l'URL <http://localhost:8000/api/vehicles/search/findByBrand?brand=Ford>. Cette étape permet de créer l'API RESTful pour le backend, qui sera ensuite consommée par le frontend React dans une étape ultérieure.



Résumé

Dans ce lab, nous avons créé un service web RESTful avec Spring Boot. Nous avons commencé par créer un contrôleur et une méthode pour renvoyer toutes les voitures au format JSON. Ensuite, nous avons utilisé Spring Data REST pour obtenir un service web pleinement fonctionnel, offrant toutes les fonctionnalités CRUD. Nous avons exploré différentes requêtes HTTP nécessaires pour manipuler les entités, notamment pour la création, la lecture, la mise à jour et la suppression des voitures. Enfin, nous avons intégré des requêtes personnalisées dans le service web RESTful.