self-service-machine-api - STEP 11

Mise en place de la recherche

Nous allons maintenant améliorer notre API REST en ajouter une recherche à travers nos produits.

Pour cela nous allons modifier le code de la route GET /api/products.

En effet, nous allons apprendre à utiliser les paramètres de requêtes GET.

On commence par ajouter dans le fichier `mock-product.mjs` le produit suivant :

```
...

{
  id: 10,
  name: "Big Tasty",
  price: 5.99,
  created: new Date(),
},

...
```

Maintenant nous disposons de 2 produits avec le mot 'big'.

Nous aimerions faire une recherche à travers pour les produits en basant cette recherche sur le nom du produit.

GET /api/products?name=big

Nous aimerions que cela retourne tous les produits qui contiennent le mot 'big'.

Voila le code qui permet de faire cela :

```
...
import { ValidationError, Op } from "sequelize";
...

productsRouter.get("/", (req, res) => {
  if (req.query.name) {
    return Product.findAll({
      where: { name: { [Op.like]: `%${req.query.name}%` } },
    }).then((products) => {
      const message = `Il y a ${products.length} produits qui correspondent au terme de la recherche`;
      res.json(success(message, products));
    });
  }
  Product.findAll()
    .then((products) => {
      const message = "La liste des produits a bien été récupérée.";
      res.json(success(message, products));
    })
    .catch((error) => {
      const message =
        "La liste des produits n'a pas pu être récupérée. Merci de réessayer dans quelques instants.";
      res.status(500).json({ message, data: error });
    });
});
```

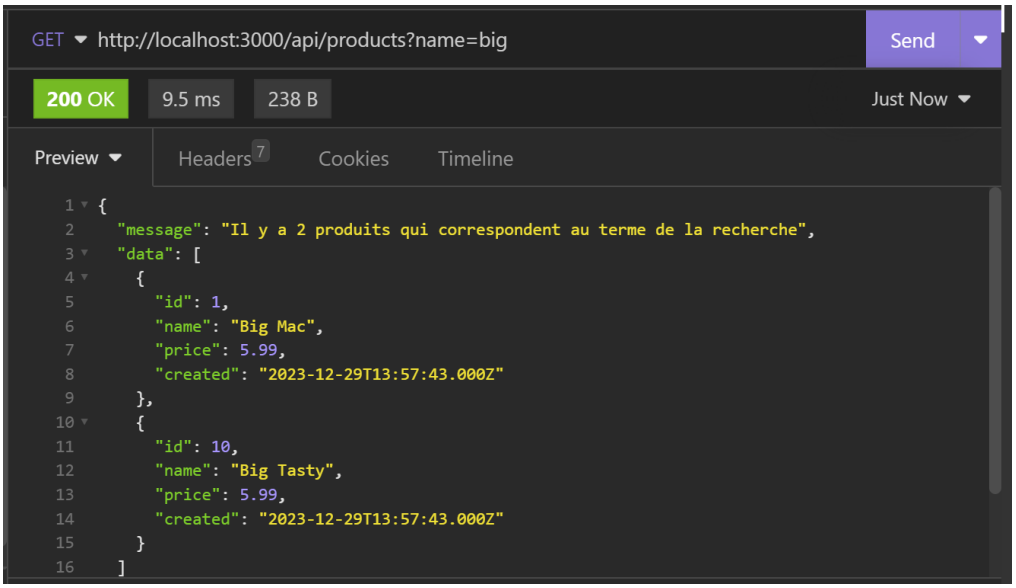
```
...
```

Vous connaissez déjà l'opérateur LIKE utilisé dans le code ci-dessus. En effet si nous devions faire cette requête en SQL, nous ferions :

```
SELECT *
FROM products
WHERE name LIKE '%big%';
```

C'est exactement ce que nous faisons mais en utilisant l'ORM sequelize.

Nous pouvons vérifier que ce code fonctionne bien :



Mise en place d'une limitation pour la recherche

Avec une grande quantité de produits, chaque recherche via l'appel HTTP GET /api/products?name=… qui retournent un certain nombre de produits pourrait mettre un temps conséquent à s'exécuter.

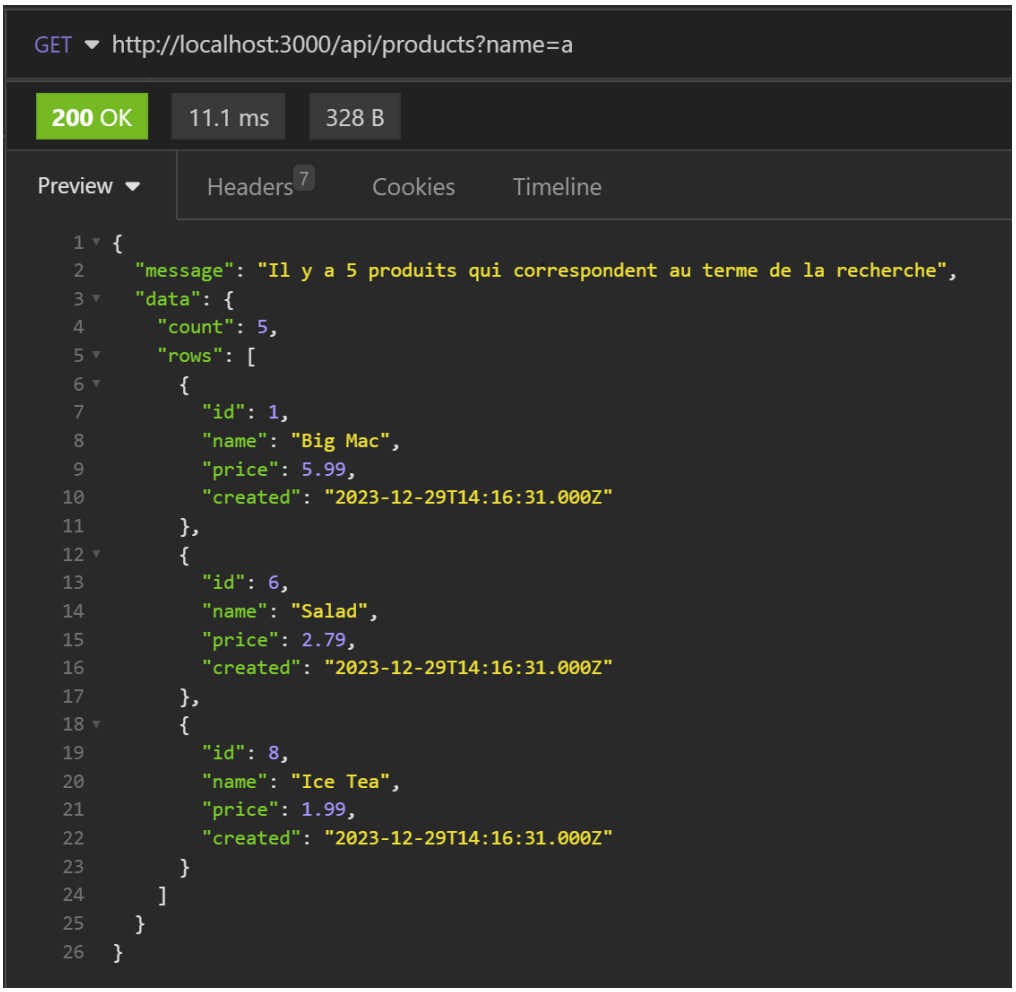
Nous allons donc limiter le nombre de résultats de manière arbitraire à 3 pour le moment. Nous rendrons cela dynamique par la suite.

Nous allons également vouloir afficher le nombre total de produits qui correspondent à la recherche même si à cause de la limitation, le consommateur ne verra que les 3 premiers pour l'instant.

```
...
productsRouter.get("/", (req, res) => {
  if (req.query.name) {
    return Product.findAndCountAll({
      where: { name: { [Op.like]: `%${req.query.name}%` } },
      limit: 3,
    }).then((products) => {
      const message = `Il y a ${products.count} produits qui correspondent au terme de la recherche`;
      res.json(success(message, products));
    });
  }
  Product.findAll()
    .then((products) => {
```

```
      const message = "La liste des produits a bien été récupérée.";
      res.json(success(message, products));
    })
    .catch((error) => {
      const message =
        "La liste des produits n'a pas pu être récupérée. Merci de réessayer dans quelques instants.";
      res.status(500).json({ message, data: error });
    });
});
...
```

Nous pouvons vérifier que ce code fonctionne bien :



Nous indiquons bien que 5 produits correspondent à la recherche même si nous limiter à 3 le résultat de la recherche.

Exercice :

Le consommateur de notre API REST a besoin de choisir le nombre de produits qu'il souhaite. Par exemple dans le cas d'une pagination.

A vous de rendre cette limitation dynamique c'est à dire que pour le cas de la recherche, le consommateur peut saisir l'URL suivante : GET /api/products?name=a&limit=1

A vous de mettre en place le code permettant de prendre en charge ce nouveau paramètre GET limit.

Solution de l'exercice :

```
productsRouter.get("/", (req, res) => {
  if (req.query.name) {
    let limit = 3;
    if (req.query.limit) {
      limit = parseInt(req.query.limit);
    }
    return Product.findAndCountAll({
      where: { name: { [Op.like]: `%${req.query.name}%` } },
      limit: limit,
    }).then((products) => {
      const message = `Il y a ${products.count} produits qui correspondent au terme de la recherche`;
      res.json(success(message, products));
    });
  }
  Product.findAll()
    .then((products) => {
      const message = "La liste des produits a bien été récupérée.";
      res.json(success(message, products));
    })
    .catch((error) => {
      const message =
        "La liste des produits n'a pas pu être récupérée. Merci de réessayer dans quelques instants.";
      res.status(500).json({ message, data: error });
    });
});
```

Passons à l'étape n°12