self-service-machine-api - STEP 7

Maintenant nous allons pouvoir modifier nos routes afin que ces dernières utilisent l'ORM sequelize plutôt qu'une liste statique de produits.

Sequelize a quelques méthodes de base que nous allons utiliser :

- Model.create()
- Model.findAll()
- Model.findByPk()
- Model.update()
- Model.destroy()

Comme toujours, la documentation officielle doit être consultée :

https://sequelize.org/docs/v6/core-concepts/model-querying-basics/

Modification des routes

Route GET /api/products/

Pour récupérer tous les produits présents en base de données, nous allons utiliser la méthode `findAll()` de l'ORM `sequelize`.

Pour ce faire, nous allons appeler cette méthode `findAll()` sur le modèle `Product` créé dans l'étape précédente.

Il est important de comprendre que chaque méthode de l'ORM Sequelize est une promesse javascript.

Définition d'une promesse :

Une promesse est un objet (Promise) qui représente la complétion (la réussite) ou l'échec d'une opération asynchrone.

Vous devez donc bien connaître cette notion js pour comprendre le code que nous allons écrire maintenant :

Théorie sur les promesses en JS

Voilà ce que cela donne dans le code du fichier `routes/products.mjs` :

```
productsRouter.get("/", (req, res) => {
  Product.findAll().then((products) => {
    const message = "La liste des produits a bien été récupérée.";
    res.json(success(message, products));
  });
});
```

Route GET /api/products/:id

Pour récupérer un produit dont l'id est passé en paramètre, nous allons cette fois utiliser la méthode `findByPk()` sur le modèle `Product`. Cette méthode prend en paramètre la clé primaire (Primary Key (Pk) en anglais) du produit que l'on souhaite récupérer.

Voilà ce que cela donne dans le code du fichier `routes/products.mjs` :

```
productsRouter.get("/:id", (req, res) => {
  Product.findByPk(req.params.id).then((product) => {
```

```
    const message = `Le produit dont l'id vaut ${product.id} a bien été récupéré.`;
    res.json(success(message, product));
  });
});
```

Route POST /api/products/

Pour l'ajout d'un nouveau produit, nous allons utiliser la méthode `create()` en passant en paramètre les données fournies par l'utilisateur. Comme d'habitude, on récupère les données dans le corps de la requête HTTP grâce à `req.body`.

Vous vous souvenez que grâce au middleware `express.json()` le json fourni a été transformé en javascript !

Attention ! Ce n'est pas une bonne pratique de ne pas vérifier ou valider les données envoyées par l'utilisateur. Mais pour l'instant on commence par faire fonctionner notre API REST avec l'ORM `Sequilize` et la DB `MySQL`. Dans une prochaine étape, nous allons ajouter de la validation sur les données du client.

Voilà ce que cela donne dans le code du fichier `routes/products.mjs` :

```
productsRouter.post("/", (req, res) => {
  Product.create(req.body).then((createdProduct) => {
    // Définir un message pour le consommateur de l'API REST
    const message = `Le produit ${createdProduct.name} a bien été créé !`;

    // Retourner la réponse HTTP en json avec le msg et le produit créé
    res.json(success(message, createdProduct));
  });
});
```

Route DELETE /api/products/:id

Pour la suppression d'un produit, nous allons utiliser 2 méthodes de l'ORM :

La 1ère est la méthode `findByPk()` que nous allons déjà utiliser. Nous allons commencer par récupérer le produit que nous souhaitons supprimer. Ainsi nous pourrons afficher des informations sur ce produit supprimé à notre consommateur (notre client).

La 2ème méthode est la méthode `destroy()` qui permet de supprimer le produit en question. La syntaxe est un peu différente des méthodes utilisées jusqu'à présent. En effet, nous devons fournir l'id du produit à supprimer mais à une propriété `where`.

Un dernier point concernant l'enchaînement des promesses. En effet, nous avons une 1ère promesse pour la récupération du produit à supprimer. Ensuite une 2ème promesse pour supprimer le produit.

Nous avons donc 2 `then` que s'enchaînent.

Le principe du chaînage des promesses est une notion importante en JS que vous devez maîtriser.

```
productsRouter.delete("/:id", (req, res) => {
  Product.findByPk(req.params.id).then((deletedProduct) => {
    Product.destroy({
      where: { id: deletedProduct.id },
    }).then((_) => {
      // Définir un message pour le consommateur de l'API REST
      const message = `Le produit ${deletedProduct.name} a bien été supprimé !`;

      // Retourner la réponse HTTP en json avec le msg et le produit créé
      res.json(success(message, deletedProduct));
    });
```

```
  });
});
```

Route PUT /api/products/:id

Pour la mise à jour d'un produit nous allons commencer par utiliser la méthode `update()` en lui passant :

- le `req.body`
- l'id du produit à mettre à jour

Comme pour la suppression, il y a un chaînage des promesses.

```
productsRouter.put("/:id", (req, res) => {
  const productId = req.params.id;
  Product.update(req.body, { where: { id: productId } }).then((_) => {
    Product.findByPk(productId).then((updatedProduct) => {
      // Définir un message pour l'utilisateur de l'API REST
      const message = `Le produit ${updatedProduct.name} dont l'id vaut ${updatedProduct.id} a été mis à jour avec succè

      // Retourner la réponse HTTP en json avec le msg et le produit créé
      res.json(success(message, updatedProduct));
    });
  });
});
```

Ci-dessous, vous trouverez le code complet pour le fichier `routes/products.mjs`

```
import express from "express";

import { Product } from "../db/sequelize.mjs";

import { success } from "./helper.mjs";

const productsRouter = express();

productsRouter.get("/", (req, res) => {
  Product.findAll().then((products) => {
    const message = "La liste des produits a bien été récupérée.";
    res.json(success(message, products));
  });
});

productsRouter.get("/:id", (req, res) => {
  Product.findByPk(req.params.id).then((product) => {
    const message = `Le produit dont l'id vaut ${product.id} a bien été récupéré.`;
    res.json(success(message, product));
  });
});

productsRouter.post("/", (req, res) => {
  Product.create(req.body).then((createdProduct) => {
    // Définir un message pour le consommateur de l'API REST
    const message = `Le produit ${createdProduct.name} a bien été créé !`;

    // Retourner la réponse HTTP en json avec le msg et le produit créé
    res.json(success(message, createdProduct));
  });
});

productsRouter.delete("/:id", (req, res) => {
  Product.findByPk(req.params.id).then((deletedProduct) => {
    Product.destroy({
      where: { id: deletedProduct.id },
```

```
    }).then((_) => {
      // Définir un message pour le consommateur de l'API REST
      const message = `Le produit ${deletedProduct.name} a bien été supprimé !`;

      // Retourner la réponse HTTP en json avec le msg et le produit créé
      res.json(success(message, deletedProduct));
    });
  });
});

productsRouter.put("/:id", (req, res) => {
  const productId = req.params.id;
  Product.update(req.body, { where: { id: productId } }).then((_) => {
    Product.findByPk(productId).then((updatedProduct) => {
      // Définir un message pour l'utilisateur de l'API REST
      const message = `Le produit ${updatedProduct.name} dont l'id vaut ${updatedProduct.id} a été mis à jour avec succè

      // Retourner la réponse HTTP en json avec le msg et le produit créé
      res.json(success(message, updatedProduct));
    });
  });
});

export { productsRouter };
```

Suppression du code mort

Après avoir vérifier que votre API fonctionne à nouveau en testant vos routes avec Insomnia, il nous reste à supprimer le code mort.

Dans le fichier `mock-product.mjs` on peut supprimer toutes les fonctions :

```
...

/**
 * Récupère le produit dont l'id vaut `productId`
 * @param {*} productId
 */
const getProduct = (productId) => {
  return products.find((product) => product.id == productId);
};

/**
 * Supprime le produit dont l'id vaut `productId`
 * @param {*} productId
 */
const removeProduct = (productId) => {
  products = products.filter((product) => product.id != productId);
};

/**
 * Met à jour le produit dont l'id vaut `productId`
 * @param {*} productId
 * @param {*} updatedProduct
 */
const updateProduct = (productId, updatedProduct) => {
  products = products.map((product) =>
    product.id == productId ? updatedProduct : product
  );
};

/**
 * Génère et retourne le prochain id des produits
 * @param {*} products
 */
const getUniqueId = () => {
```

```
  const productsIds = products.map((product) => product.id);
  const maxId = productsIds.reduce((a, b) => Math.max(a, b));
  const uniqueId = maxId + 1;

  return uniqueId;
};

...
```

C'était une étape importante ! Notre API REST fonctionne maintenant avec une base de données !
Passons à l'étape n°8