self-service-machine-api - STEP 4

Dans cette étape nous allons mettre en place 2 routes :

- la route permettant de mettre à jour un produit
- la route permettant de supprimer un produit

Nous allons également apprendre à utiliser un outil qui facilite l'exécution de requêtes HTTP pour tester notre backend.

Route DELETE /api/products/id

```
productsRouter.delete("/:id", (req, res) => {
  const productId = req.params.id;

  let deletedProduct = getProduct(productId);

  removeProduct(productId);

  // 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));
});
```

Pour tester le code `curl -X DELETE http://localhost:3000/api/products/1`

```
Greg@LAPTOP-5335QTOF MINGW64 ~/OneDrive - Education Vaud/295/self-service-machine-api (main)
$ curl -X DELETE http://localhost:3000/api/products/1
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   140  100   140    0     0   8561      0 --:--:-- --:--:-- --:--:--  9333{"message":"Le produit Big
Mac a bien été supprimé !","data":{"id":1,"name":"Big Mac","price":5.99,"created":"2023-12-27T11:59:01.23
3Z"}}
```

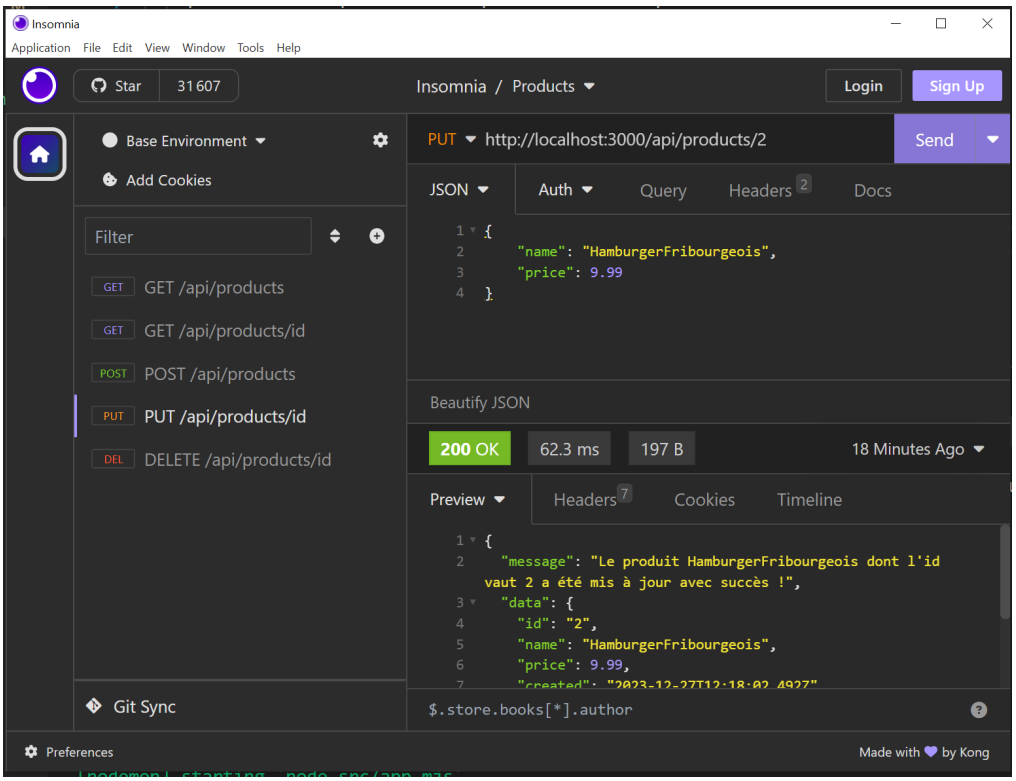
Route PUT /api/products/id

```
productsRouter.put("/:id", (req, res) => {
  const productId = req.params.id;

  const product = getProduct(productId);

  // Mise à jour du produit
  // A noter que la propriété 'created' n'étant pas modifiée, sera conservée telle quelle.
  const updatedProduct = {
    id: productId,
    ...req.body,
    created: product.created,
  };
  updateProduct(productId, updatedProduct);

  // Définir un message pour l'utilisateur de l'API REST
  const message = `Le produit ${updatedProduct.name} dont l'id vaut ${productId} a été mis à jour avec succès !`;

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

Nous avons placé certaines fonctions liées aux produits dans le fichier `mock-product.mjs`

```
...

/**
 * 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;
};

export { products, getProduct, removeProduct, updateProduct, getUniqueId };
```

Tester notre API REST avec insomnia



Vous trouverez dans le répertoire `/test/insomnia` un export json permettant de tester les différentes routes.

Passons à l'étape n°5