

## Objectif de l'application

L'application que nous allons créer a pour objectif de gérer des produits, en ajouter, les modifier ou en supprimer.

N.B. : Cette application est la suite de l'exercice Ecommerce en TypeScript.

## Un rappel sur Node JS

Node.js est une plateforme logicielle libre et événementielle en JavaScript. Elle permet de réaliser une application complète en Javascript que ce soit du côté du client ou du serveur.

Un bon nombre de plateformes web connues à fort trafic tournent sur Node JS, on peut citer LinkedIn, Yahoo !, Rakuten,...

De nombreux frameworks Javascript tournent sur Node JS, on peut citer notamment [Express](#) et [Meteor](#)

Avec Node JS, on peut construire des applications mobiles cross-platform (avec [Ionic](#)) ou des applications desktop cross-platform (avec [electron](#))

Les meilleurs exemples dans ce domaine sont [Discord](#) et [Slack](#), des outils de collaboration intégrant la discussion instantanée notamment utilisés par les gamers et les développeurs.

## Au préalable

Pour acquérir cette compétence, vous avez besoin de connaître le HTML et CSS et d'être à l'aise avec la programmation en Javascript.

## De l'aide

La documentation est essentielle pour bien comprendre le fonctionnement et le développement. Comme tout bon développeur, ayez le réflexe d'avoir la documentation de Node JS ou encore le site des communautés à disposition.

[Site officiel](#)

[Une documentation de développeur](#)

## Installation de NPM (Node Package Manager) et Node JS

### NPM (Node Package Manager)

**NPM** est un outil en ligne de commande permettant de gérer les dépendances en Javascript (Node JS ici).

Il est à télécharger et à installer. L'installateur Node JS installe également NPM : <https://nodejs.org/en/download/>

**NPM** permet de :

- Garder à jour les différentes bibliothèques dont un projet a besoin pour fonctionner.
- Gérer les dépendances entre bibliothèques (installation & mises à jour).
- Générer un fichier de chargement automatique commun à toutes les bibliothèques générées.

Centralisation des informations de chaque bibliothèque :

- Sur le site [www.npmjs.com](https://www.npmjs.com)
- Accès à Internet requis (configurer pour le proxy).

Pour vérifier si NPM est bien installé, on ouvre un terminal et on exécute la ligne de commande :

**npm -v**

```
C:\>npm -v
2.15.10
C:\>
```

## Node JS

Pour installer la plateforme Node JS,

Elle est à télécharger et à installer. L'installateur Node JS installe également NPM :  
<https://nodejs.org/en/download/>

Pour vérifier si Node JS est bien installé, on ouvre un terminal et on exécute la ligne de commande :

**node -v**

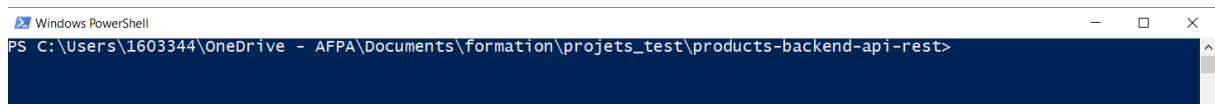
```
C:\>node -v
v10.15.3
C:\>
```

## Créer le projet d'application

### Installation des dépendances

Créer le répertoire **products-backend-api-rest**

Dans un terminal, se mettre sur ce répertoire



*Terminal sur le dossier de projet*

Nous allons initialiser le projet avec la commande : **npm init**

Cette commande a pour but de créer un package.json avec les informations du projet.

Laissez tout par défaut.

Ensuite on va installer les librairies dont nous avons besoin :

- Framework Express (serveur web et router)
- Mongoose (ODM : Object-Document Mapping) (Cela ressemble à ORM n'est-ce pas)
- CORS (pour gérer les erreurs de sécurité CORS)

Pour installer tout ça nous allons exécuter la ligne de commande :

**npm install express mongoose cors**

## Création du premier fichier de code

### 🕒 Créer un serveur web en Express

A la racine du projet, créer un fichier **main.js**  
qui contiendra ce code

```
// @/main.js
const express = require("express");

const app = express();

app.use(express.json());

app.get("/", async (req, res) => {
  return res.json({ message: "Hello, World 🌎" });
});

const start = async () => {
  try {
    app.listen(3000, () => console.log("Server started on port 3000"));
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};

start();
```

*Création d'un fichier de code permettant de démarrer un serveur Web en Express*

## Résultat

### 🕒 Lancement dans le navigateur

Dans le terminal à la racine du projet, lancez la commande : **node main.js**  
et ce message apparaît

```
Server started on port 3000
```

*Launch server Node Express*

Direction un navigateur à l'adresse : [localhost:3000](http://localhost:3000)



*Lancement du serveur web et résultat sur le navigateur*

Nous avons la réponse en JSON.

## Création de la base de données MongoDB

Au préalable, il faut avoir suivi la ressource sur MongoDB

Soit avec une installation locale ou l'utilisation du cloud mongodb nommé Atlas permettant d'avoir la base de données accessible en ligne.

Avec Compass, nous allons créer la base de données qui contiendra nos produits.

A screenshot of the 'Create Database' dialog in MongoDB Compass. The dialog has a title bar with a close button (X). It contains two input fields: 'Database Name' with the value 'ecommerce' and 'Collection Name' with the value 'products'. Below these fields is a checkbox labeled 'Time-Series' which is unchecked. Underneath the checkbox is a description: 'Time-series collections efficiently store sequences of measurements over a period of time. Learn More' with a link icon. At the bottom, there is a section titled 'Additional preferences (e.g. Custom collation, Capped, Clustered collections)' with a right-pointing arrow. At the very bottom are two buttons: 'Cancel' and 'Create Database'.

*Création de la base données Mongoddb Ecommerce*

Ensuite nous allons créer manuellement un produit.

## Insert Document

To Collection ecommerce.products

VIEW  

```
1 ▾ /**
2  * Paste one or more documents here
3  */
4  ▾ {
5  ▾   "_id": {
6  |     "$oid": "660d834a0df92bb0e63ba51f"
7  |   },
8  |   "name": "MICHAËL GREGORIO",
9  |   "price": 43.00,
10 |   "promotion": false,
11 |   "discount": 20.00,
12 ▾   "category": {
13 |     "name": "Spectacles Rodez"
14 |   }
15 }
```

Cancel

Insert

*Insertion d'un produit dans la base de données MongoDB*

## Le codage des interactions avec la base de données

### 🕒 La connexion à la base de données

On va mettre en place la connexion à la base de données au lancement de l'application.

On va écrire dans le fichier **main.js** se situant à la racine du projet.

Dans ce fichier, on va mettre après les différents **require**, les 3 lignes pour intégrer les bibliothèques et spécifier sur quelle base de données on souhaite se connecter

```
// @/main.js
const express = require("express");
const mongoose = require("mongoose");

const app = express();

// ...

const start = async () => {
  try {
    await mongoose.connect(
      "mongodb://root:root@localhost:27017/mongoose?authSource=admin"
    );
    app.listen(3000, () => console.log("Server started on port 3000"));
  } catch (error) {
    console.error(error);
    process.exit(1);
  }
};

start();
```

*Code with MongoDB connection*



## Rendre accessible la connexion à la base de données par le Router

On va mettre en place le fait que le Router puisse avoir accès à la connexion à la base de données.  
On va écrire dans le fichier **main.js** se situant à la racine du projet.

```
let db = mongoose.connection;

// Make our db accessible to our router
app.use(function(req,res,next){
  req.db = db;
  next();
});
```

*Make our db accessible to our router*

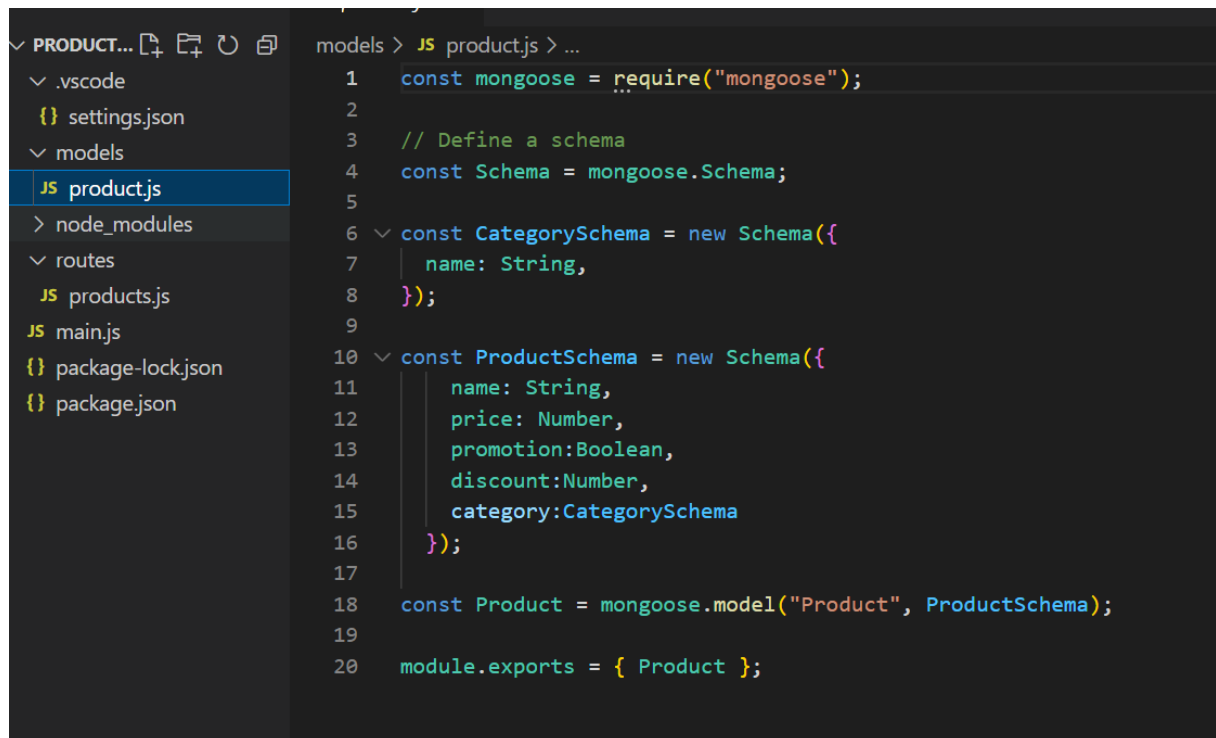
Cela permettra qu'à chaque action effectuée sur une route on puisse récupérer la connexion à la base de données et agir dessus.

Par exemple, sur la route "AddProduct", on va pouvoir ajouter un utilisateur en base de données.

## 🕒 Définition du modèle de données

Déclaration du modèle de données MongoDB avec Mongoose

On va créer un répertoire **models** dans lequel on va mettre un fichier **product.js**



```
1  const mongoose = require("mongoose");
2
3  // Define a schema
4  const Schema = mongoose.Schema;
5
6  const CategorySchema = new Schema({
7    name: String,
8  });
9
10 const ProductSchema = new Schema({
11   name: String,
12   price: Number,
13   promotion: Boolean,
14   discount: Number,
15   category: CategorySchema
16 });
17
18 const Product = mongoose.model("Product", ProductSchema);
19
20 module.exports = { Product };
```

*Define a database schema*

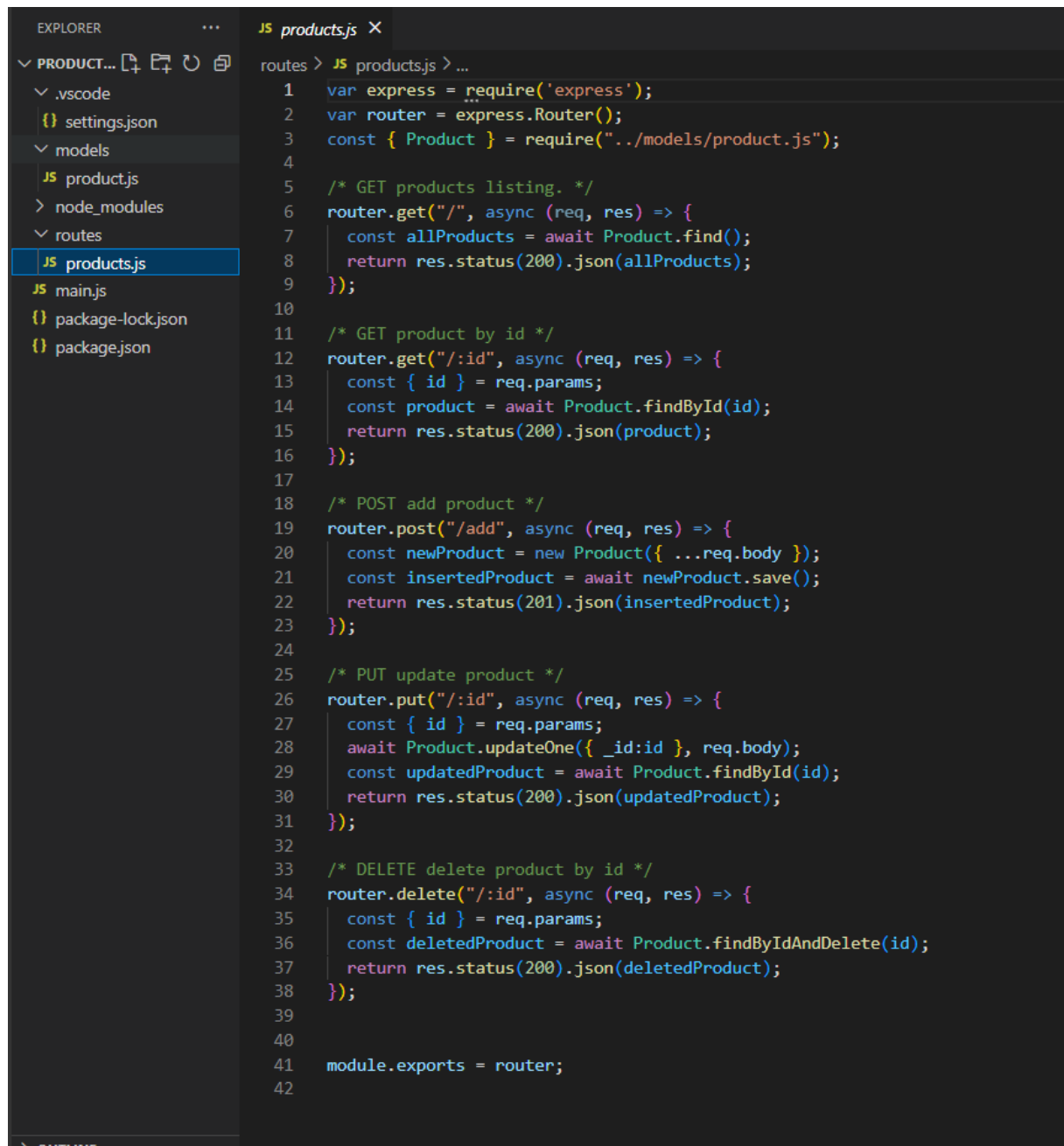
## 🕒 Déclaration des routes

Toutes les routes qui vont être mises pour les actions à réaliser sont à inscrire avant **module.exports = router** ;

Cela va inscrire les actions et les routes dans le système.

On va créer un répertoire **routes** où on mettra un fichier **products.js**

Ce fichier, on va le compléter avec les opérations de base sur les données (CRUD)



```
EXPLORER
  PRODUCT...
  .vscode
  {} settings.json
  models
  JS product.js
  > node_modules
  routes
  JS products.js
  JS main.js
  {} package-lock.json
  {} package.json

routes > JS products.js > ...
1  var express = require('express');
2  var router = express.Router();
3  const { Product } = require("../models/product.js");
4
5  /* GET products listing. */
6  router.get("/", async (req, res) => {
7    const allProducts = await Product.find();
8    return res.status(200).json(allProducts);
9  });
10
11 /* GET product by id */
12 router.get("/:id", async (req, res) => {
13   const { id } = req.params;
14   const product = await Product.findById(id);
15   return res.status(200).json(product);
16 });
17
18 /* POST add product */
19 router.post("/add", async (req, res) => {
20   const newProduct = new Product({ ...req.body });
21   const insertedProduct = await newProduct.save();
22   return res.status(201).json(insertedProduct);
23 });
24
25 /* PUT update product */
26 router.put("/:id", async (req, res) => {
27   const { id } = req.params;
28   await Product.updateOne({ _id: id }, req.body);
29   const updatedProduct = await Product.findById(id);
30   return res.status(200).json(updatedProduct);
31 });
32
33 /* DELETE delete product by id */
34 router.delete("/:id", async (req, res) => {
35   const { id } = req.params;
36   const deletedProduct = await Product.findByIdAndDelete(id);
37   return res.status(200).json(deletedProduct);
38 });
39
40
41 module.exports = router;
42
```

CRUD Routes

## 🕒 On améliore et complète le fichier main.js

On met une sécurité sur la base mongodb

On intègre la gestion cors pour éviter des problèmes lors des appels API en raison des restrictions de sécurité liées au navigateur

On inclut notre fichier de routes



```
const express = require("express");
const mongoose = require("mongoose");
var cors = require('cors')

var productsRouter = require('./routes/products');

const app = express();

// Set `strictQuery: false` to globally opt into filtering by properties that aren't in the schema
// Included because it removes preparatory warnings for Mongoose 7.
// See: https://mongoosejs.com/docs/migrating\_to\_6.html#strictquery-is-removed-and-replaced-by-strict
mongoose.set("strictQuery", false);

app.use(express.json());
app.use(cors())

let db = mongoose.connection;

// Make our db accessible to our router
app.use(function(req,res,next){
  req.db = db;|
  next();
});

app.use('/products', productsRouter);
```

*Configuration in main.js*

On gère les erreurs éventuelles (page non trouvée, exception)

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.json('error');
});
```

*Errors handling*

## Les tests

Les tests sont importants dans une application. nous allons tester notre REST API sur Navigateur pour les routes pour lequel c'est possible (GET), sur POSTMAN (outil pour tester les APIs), et visualiser le résultat en base de données MongoDB

## 🕒 Test sur navigateur

Nous ne pouvons tester ici que la récupération des produits au format JSON (GET)



```
[
  {
    "_id": "660e09f57609569f4a5fef04",
    "name": "DANIEL GUICHARD",
    "price": 43,
    "promotion": false,
    "discount": 0,
    "category": {
      "name": "Spectacles Rodez",
      "_id": "660e09f57609569f4a5fef05"
    },
    "__v": 0
  },
  {
    "_id": "660e0b73b6dcd8fce7d43776",
    "name": "MICHAËL GREGORIO",
    "price": 43,
    "promotion": false,
    "discount": 20,
    "category": {
      "name": "Spectacles Rodez",
      "_id": "660e0b73b6dcd8fce7d43777"
    },
    "__v": 0
  }
]
```

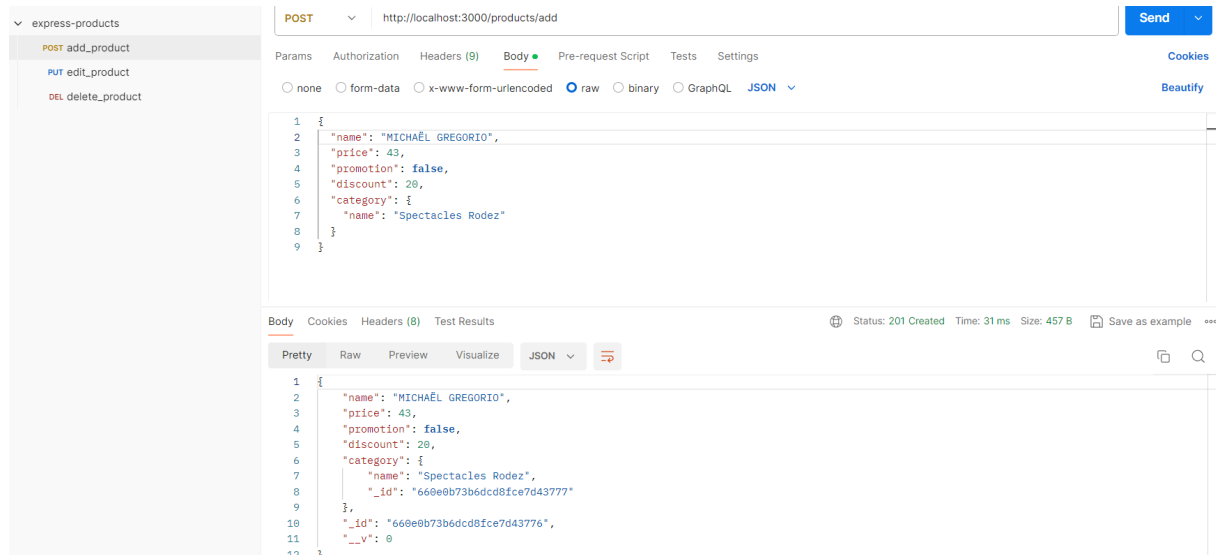
*Test products on browser*

## 🕒 Tests sur POSTMAN

on peut tester toutes les routes avec POSTMAN

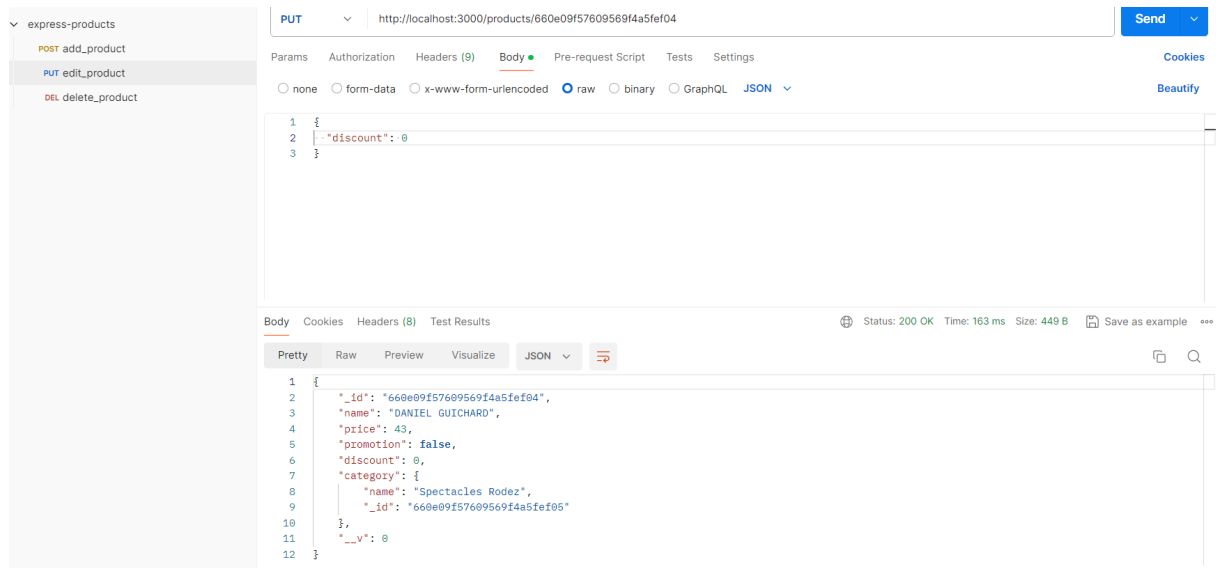
L'ajout

# Réaliser un backend en NodeJS Express sous forme de REST API en JavaScript



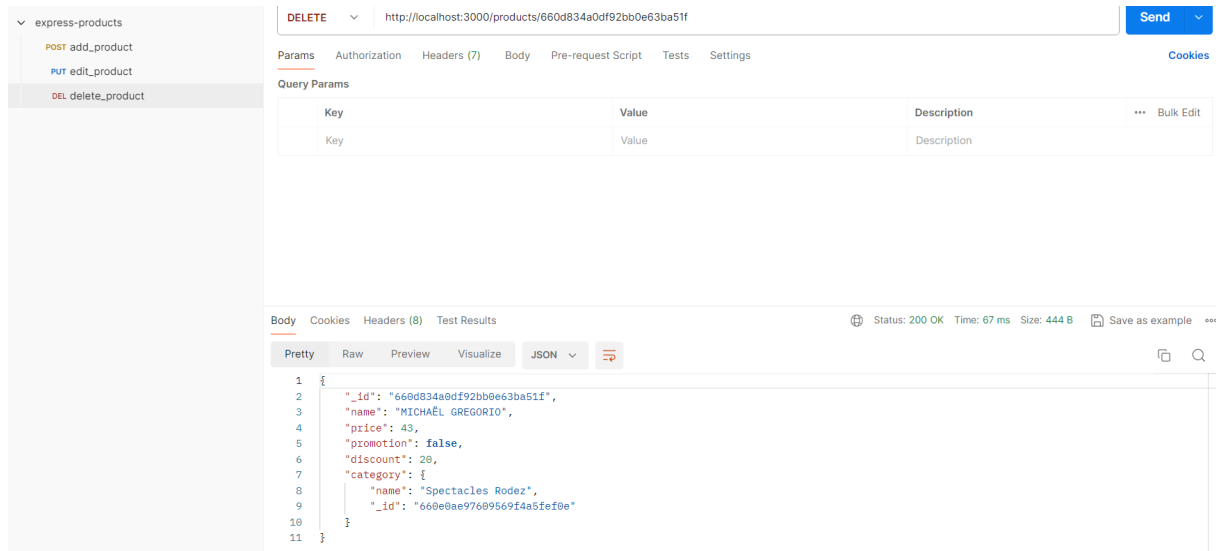
Test POSTMAN Add Product

## L'update



Test POSTMAN Update

## Le Delete



Test POSTMAN Delete

## Test en base de données avec Compass

On peut voir les résultats de nos opérations avec Compass



Test Compass

## Le code source de l'exemple

Le code est dans ce  [node-api-rest.zip](#) (cf. *Node Express REST API code*)

## Pour aller plus loin

Nous avons mis en place l'API.

Celle-ci peut être exploitable par plusieurs applications construites en différentes technologies.

On peut envisager de réaliser le front en Vue pour faire la stack MEVN

On peut envisager de réaliser le front en REACT pour faire la stack MERN

Cela peut être tout à fait avec de l'Angular ou encore en Electron ou encore en Application Mobile native, etc...

Nous n'avons pas sécurisé notre REST API. Il faut mettre en place une gestion d'authentification et des tokens d'accès. Elle est actuellement parfaitement publique sans restrictions d'utilisation.



## **Œuvre collective de l'AFPA**

Sous le pilotage de la Direction de l'Ingénierie.

© AFPA

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la transformation, l'arrangement ou la reproduction par un art ou un procédé quelconque. »