Mise en place d'une API Node.js Express.js avec ORM Prisma et MySQL

Dans le monde du développement Web, la création d'un back end robuste et efficace pour votre application est cruciale. Heureusement, les outils et Framework modernes rendent cette tâche plus facile que jamais. Dans cet atelier, nous allons créer une application RESTful Express.js en utilisant Prisma comme ORM de base de données, avec MySQL comme base de données de choix. Nous allons aborder les concepts et les étapes impliquées dans la création d'une API CRUD (Créer, Lire, Mettre à jour, Supprimer).

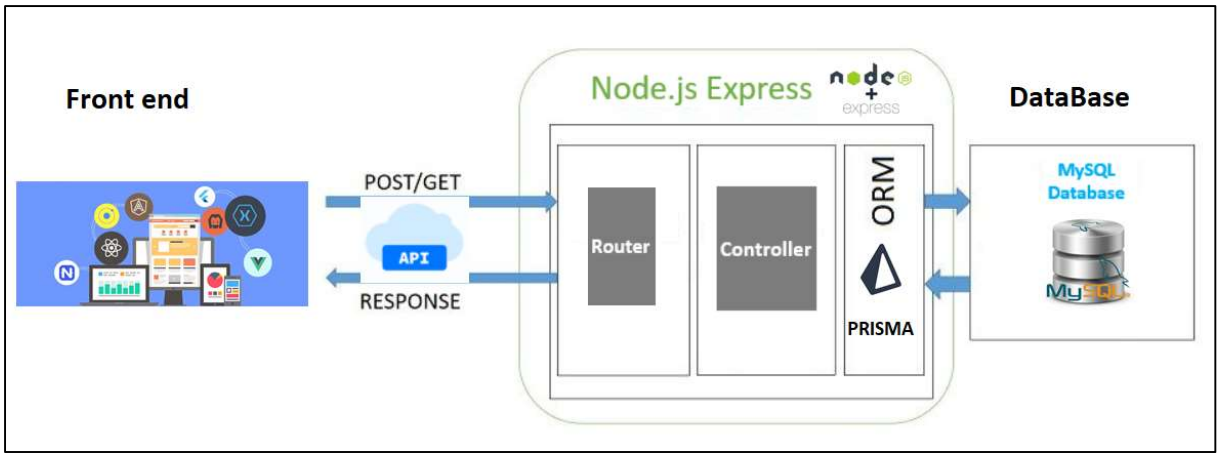
Avant de passer au code, expliquons brièvement pourquoi Prisma est un excellent choix pour travailler avec des bases de données dans vos applications Node.js. Prisma est un outil de mappage objet-relationnel (ORM) qui simplifie les interactions avec les bases de données en vous permettant de travailler avec des bases de données à l'aide d'un langage de requête de type sécurisé.

D'autre part, Node.JS est une technologie très développée ces dernières années et le langage JavaScript a évolué avec son temps. Désormais, il semble incontournable de savoir créer une application utilisant Node.JS. Nous allons, dans cet atelier, vous montrer comment créer simplement un serveur ExpressJS en JavaScript.

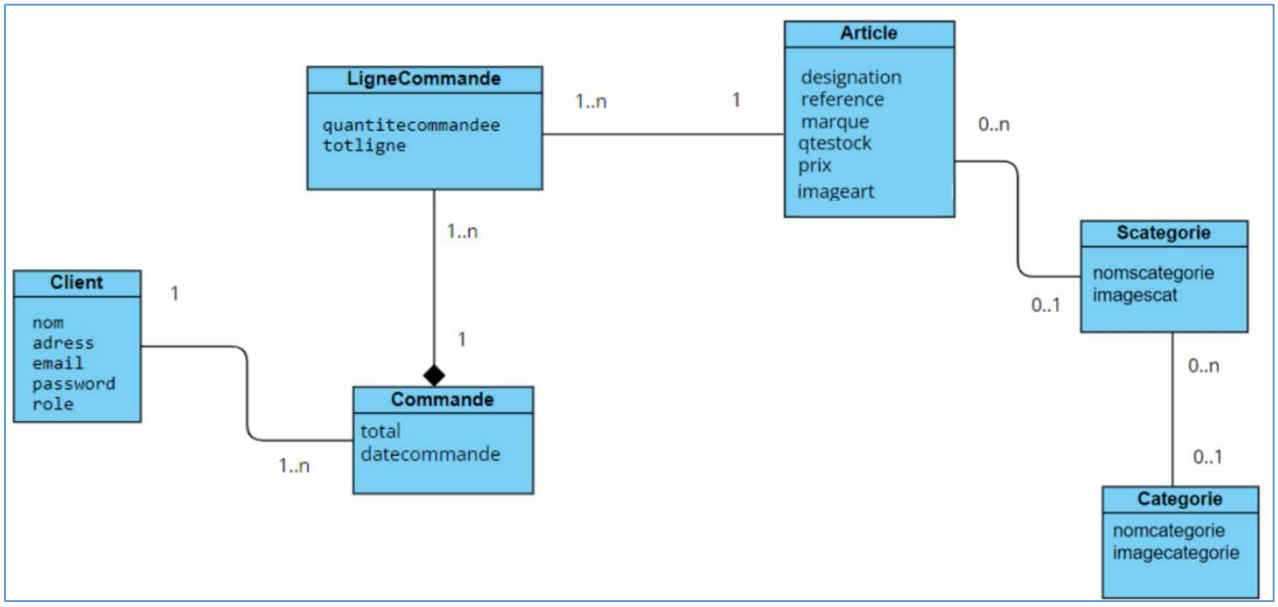
Express.js est un Framework minimaliste pour Node.js. Il permet de créer facilement une application web. Son côté minimaliste le rend peu pratique pour créer des applications de taille importante. Nous allons nous en servir pour développer des applications jouant un rôle de serveur de données pour nos applications en back end.

Avec Node.js, on n'utilise pas de serveur web HTTP comme Apache. En fait, c'est à nous de créer le serveur.

L'architecture de l'application est la suivante :



Lors de la mise en place des modèles, on considèrera la base de données déduite d'une partie du diagramme de classes UML :



Mise en place du projet

Pour commencer, assurez-vous que Node.js est installé sur votre machine. Sinon faire le téléchargement à partir du site : https://nodejs.org

Egalement il faut disposer de mySQL et tâcher que le serveur correspondant est démarré.

Remarque : Vous pouvez installer xampp ou wampserver pour utiliser phpMyAdmin en vue d'une création plus simple de la base de données.

Créez un nouveau répertoire pour votre projet et initialisez-le :

```
npm init -y
```

```
Wrote to D:\Sandra\Nodejs\2024\backend-prisma-mysql-ecommerce\package.json:

{
  "name": "backend-prisma-mysql-ecommerce",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Ensuite, installez les dépendances nécessaires : Express et Prisma

```
npm install express cors prisma @prisma/client
```

Initialiser le projet Prisma pour créer automatiquement .env et /prisma/schema.prisma

```
npx prisma init
```

```
✔ Your Prisma schema was created at prisma/schema.prisma
  You can now open it in your favorite editor.

Next steps:
1. Set the DATABASE_URL in the .env file to point to your existing database. If your database has no tables yet, read https://pris.ly/d/getting-started
2. Set the provider of the datasource block in schema.prisma to match your database: postgresql, mysql, sqlite, sqlserver, mongodb or cockroachdb.
3. Run prisma db pull to turn your database schema into a Prisma schema.
4. Run prisma generate to generate the Prisma Client. You can then start querying your database.

More information in our documentation:
https://pris.ly/d/getting-started
```

Installer aussi de façon globale nodemon

```
npm install -g nodemon
```

Création du schéma de base de données

Pour cet exemple, créons un schéma de base de données simple pour gérer une application e-commerce. Dans le fichier schema.prisma on va définir le schéma relatif au diagramme ci-dessus.


```
DATABASE_URL="mysql://root@localhost:3306/dbcommerce"
PORT=3001
```


```
generator client {
  provider = "prisma-client-js"
```

```
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model articles {
  id          Int      @id @default(autoincrement()) @db.UnsignedInt
  designation  String     @db.VarChar(30)
  marque       String     @db.VarChar(20)
  reference    String     @db.VarChar(30)
  qtestock     Int
  prix         Float
  imageart     String     @db.VarChar(255)
  scategorieID Int      @db.UnsignedInt
  created_at   String? @db.VarChar(24)
  updated_at   String? @db.VarChar(24)
  scategories  scategories @relation(fields: [scategorieID], references: [id], onUpdate: Restrict, map: "articles_scategorieid_foreign")

  @@index([scategorieID], map: "articles_scategorieid_foreign")
}

model categories {
  id           Int       @id @default(autoincrement()) @db.UnsignedInt
  nomcategorie   String       @unique(map: "categories_nomcategorie_unique") @db.VarChar(30)
  imagecategorie String       @db.VarChar(255)
  created_at     String? @db.VarChar(24)
  updated_at     String? @db.VarChar(24)
  scategories    scategories[]
}
```

```
model scategories {
  id            Int     @id @default(autoincrement()) @db.UnsignedInt
  nomscategorie   String     @db.VarChar(255)
  imagescategorie String     @db.VarChar(255)
  categorieID     Int      @db.UnsignedInt
  created_at      String? @db.VarChar(24)
  updated_at      String? @db.VarChar(24)
  articles       articles[]
  categories       categories @relation(fields: [categorieID], references: [id], onUpdate: Restrict, map: "scategories_categorieid_foreign")

  @@index([categorieID], map: "scategories_categorieid_foreign")
}

model users {
  id              Int    @id @default(autoincrement()) @db.UnsignedInt
  name            String    @db.VarChar(255)
  email           String    @unique(map: "users_email_unique") @db.VarChar(255)
  password        String    @db.VarChar(255)
  created_at      String? @db.VarChar(24)
  updated_at      String? @db.VarChar(24)
}
```

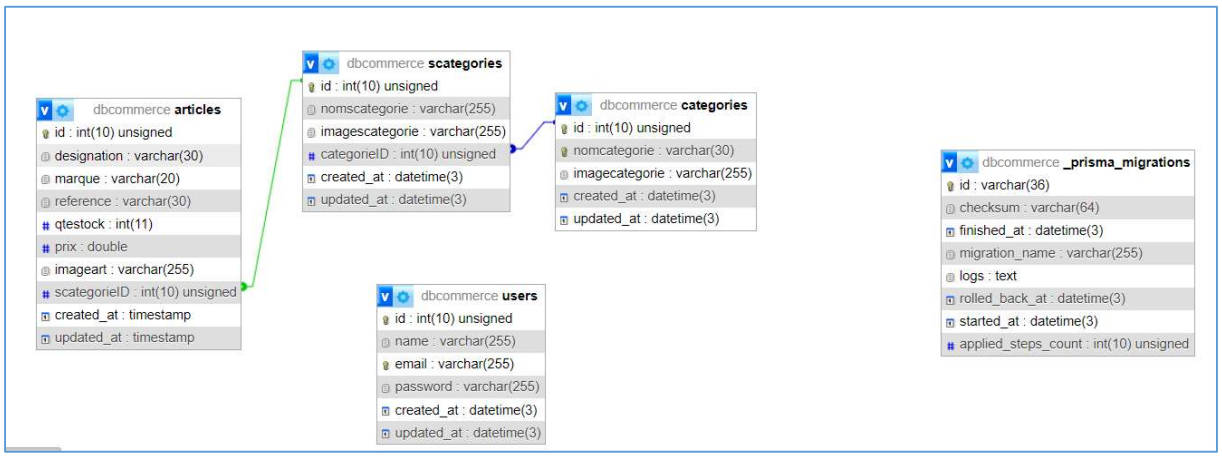
Migration

Exécutez la migration pour créer les tables de base de données :

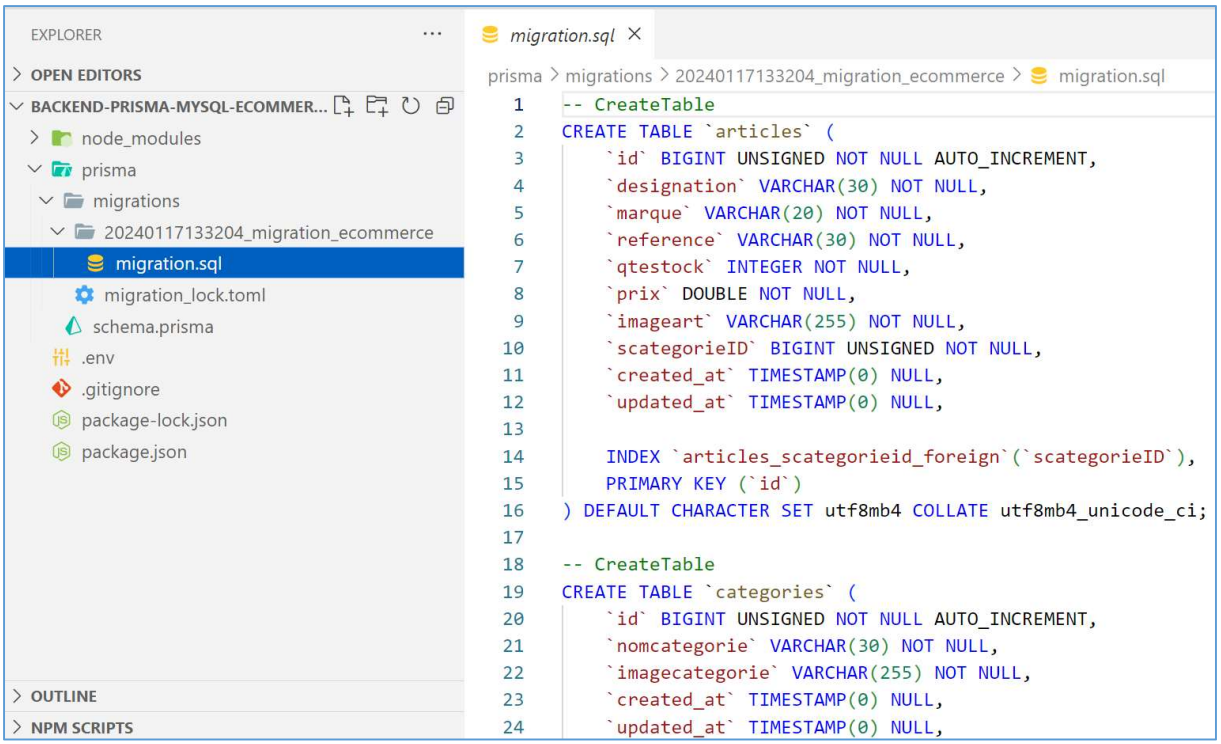
```
npx prisma migrate dev
```

```
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": MySQL database "dbcommerce" at "localhost:3306"

MySQL database dbcommerce created at localhost:3306

√ Enter a name for the new migration: ... migration_ecommerce
Applying migration `20240117133204_migration_ecommerce`

The following migration(s) have been created and applied from new schema changes:

migrations/
  └─ 20240117133204_migration_ecommerce/
    └─ migration.sql

Your database is now in sync with your schema.

✔ Generated Prisma Client (v5.8.1) to .\node_modules\@prisma\client in 289ms
```

Base de données automatiquement générée.



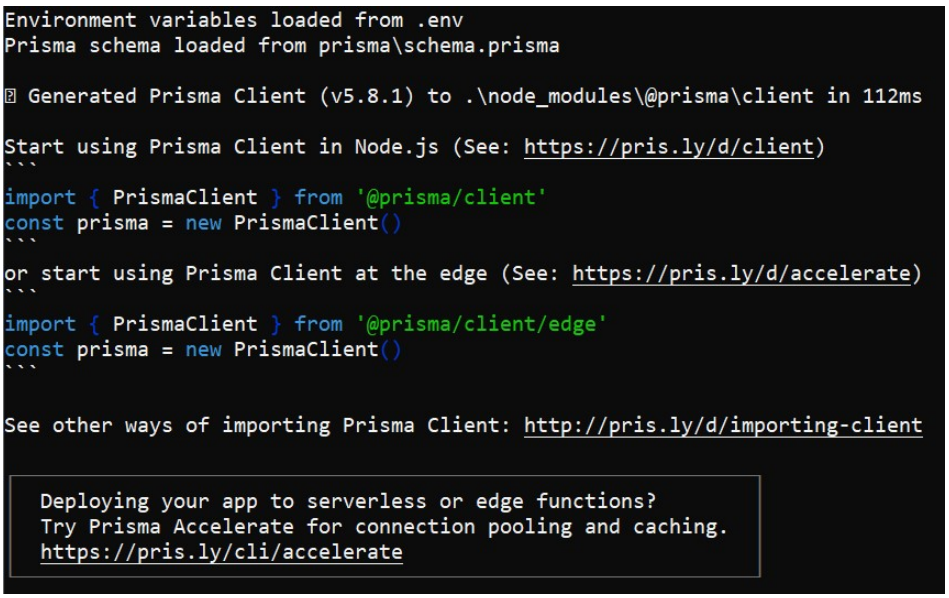
Fichier de migration automatiquement créé.



Générez le client Prisma :

```
npx prisma generate
```

La commande que nous venons d'exécuter lit notre schema.prisma et génère tout ce qu'il nous faut pour interagir avec la base de données dans le dossier suivant node_modules/@prisma/client. Voici ce qui vous devez obtenir :



Le cycle est le suivant : Chaque fois que vous modifiez la structure des tables, vous devez lancer la commande précédente pour prendre en compte vos modifications.

Une fois que vous êtes satisfait du résultat, vous devez exécuter npx prisma migrate afin de générer les scripts SQL de migration de votre base de données.

Remarque :

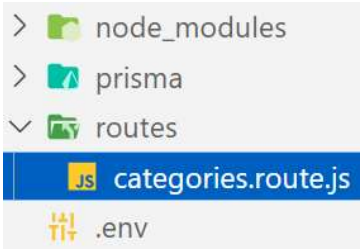
Si nous avons la base de données Mysql déjà créée, dans l'invite de commande on tape

```
npx prisma db pull
```

Automatiquement nous aurons le code de schema.prisma.

Création de la route vers les APIs categories

Dans la racine du projet, créer le dossier /routes puis le fichier categories.route.js



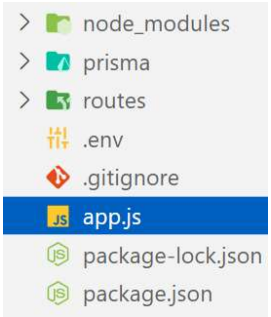

```
const express = require('express');
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()
const router = express.Router();

//Ajout d'une categorie
router.post('/', async (req, res, )=> {
    const {nomcategorie,imagecategorie}=req.body
    try {
        const categorie = await prisma.categories.create({
            data: {
                nomcategorie: nomcategorie,
                imagecategorie: imagecategorie
            }
        })

        res.json(categorie)
      } catch (error) {
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});

// afficher la liste des catégories.
router.get('/', async (req, res, )=> {
    try {
        // categories est le nom du model précisé dans prisma.schema
        const catgeories = await prisma.categories.findMany()
```

```
         res.json(categories)
        } catch (error) {
         res.status(500).json({
        message: "Something went wrong",
       })
       }
});

// afficher une categorie.
router.get('/:id', async (req, res, )=> {
    const { id } = req.params

    try {
        const categorie = await prisma.categories.findUnique({
            where: {
                id: Number(id),
            }
        })

       res.json(categorie)
      } catch (error) {
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});



// modifier une categorie
router.put('/:id', async (req, res)=> {
    const {nomcategorie,imagecategorie}=req.body
    const id  = req.params.id;

    try {
     const categorie = await prisma.categories.update({
        data: {
            nomcategorie: nomcategorie,
            imagecategorie: imagecategorie
        },
            where: { id: Number(id)},
        })
    res.json(categorie);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});
```

```
// Supprimer un categories
router.delete('/:id', async (req, res)=> {
    const  id  = req.params.id;
    try {
     await prisma.categories.delete({
        where: { id: Number(id) },
    })

    res.json({ message: "category "+ id +" deleted successfully." });
} catch (error) {
    res.status(404).json({ message: error.message });
    }
});

module.exports = router;
```

Création du serveur NodeJS

Dans la racine du projet, créer le fichier app.js où on va créer notre serveur




```
const express=require('express');
const cors=require('cors')

const app = express()

//BodyParser Middleware
app.use(express.json());
app.use(cors())

// requête
app.get("/",(req,res)=>{
    res.send("bonjour");
    });

// Appel de routes
const categoriesRouter =require("./routes/categories.route")
```

```
app.use('/api/categories', categoriesRouter);

const PORT = process.env.PORT || 3001
app.listen(PORT, () => console.log(`Server is running on port ${PORT}`))
```

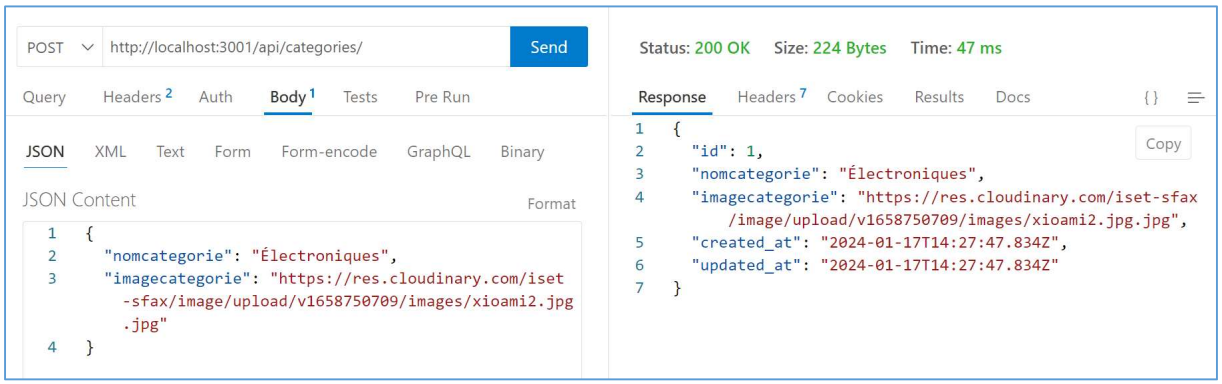
Exécuter dans le terminal

```
nodemon app
```

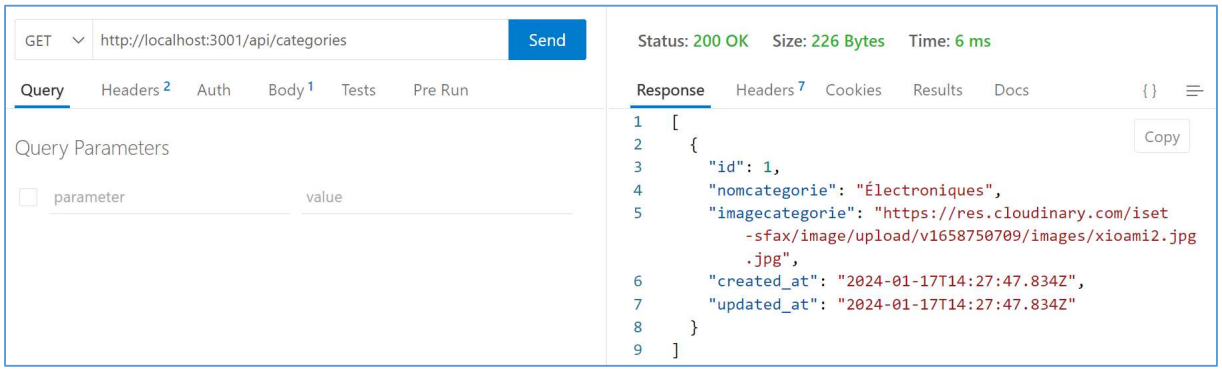
```
[nodemon] starting `node app app.js`
Server is running on port 3001
```

Test des APIs

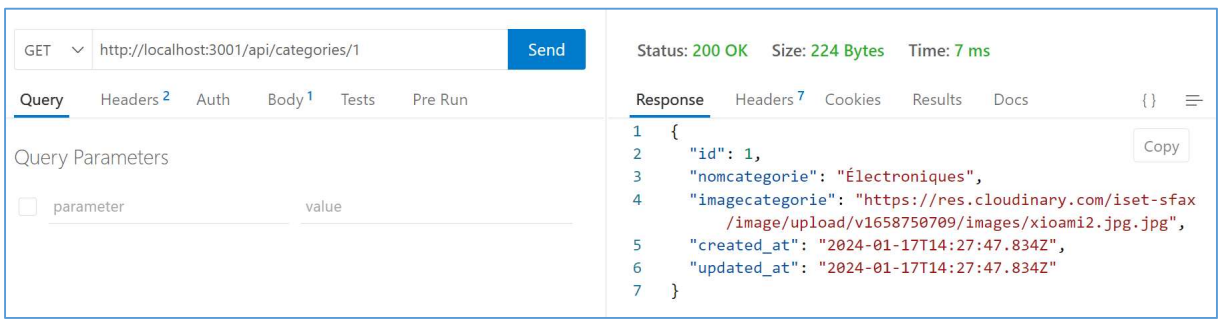
POST : http://localhost:3001/api/categories/



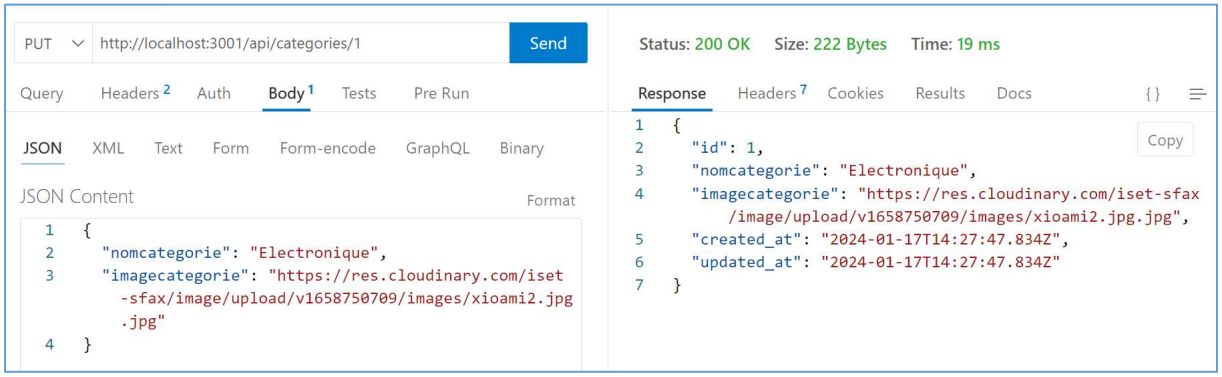
GET : http://localhost:3001/api/categories/



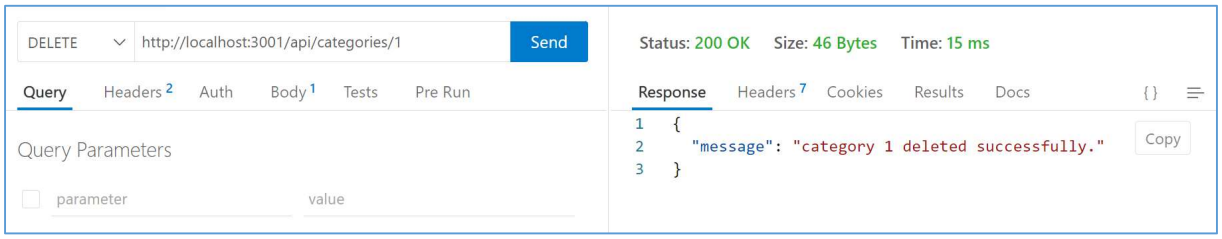
GET : http://localhost:3001/api/categories/1



PUT : http://localhost:3001/api/categories/1

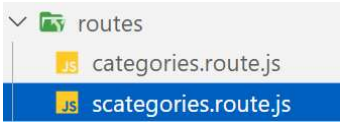


DELETE : http://localhost:3001/api/categories/1



Création de la route vers les APIs scategories

Sous le dossier /routes créer le fichier scategories.route.js




```
const express = require('express');
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()
const router = express.Router();

//Ajout d'une scategorie
router.post('/', async (req, res, )=> {
    const {nomscategorie,imagescategorie,categorieID}=req.body
    try {
        const scategorie = await prisma.scategories.create({
            data: {
                nomscategorie: nomscategorie,
                imagescategorie: imagescategorie,
                categorieID:categorieID
            }
        })

        res.json(scategorie)
      } catch (error) {
```

```
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});

// afficher la liste des catégories.
router.get('/', async (req, res, )=> {
    try {
        // scategories est le nom du model précisé dans prisma.schema
        const scategories = await prisma.scategories.findMany({
            include: {
            categories: {
                select: {
                    nomcategorie: true,
                },
          }
        }
        })
         res.json(scategories)
        } catch (error) {
         res.status(500).json({
        message: "Something went wrong",
       })
       }
});

// afficher une scategorie.
router.get('/:id', async (req, res, )=> {
    const { id } = req.params

    try {
        const scategorie = await prisma.scategories.findUnique({
            where: {
                id: Number(id),
            }
        })

       res.json(scategorie)
      } catch (error) {
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});
```

```
// modifier une scategorie
router.put('/:id', async (req, res)=> {
    const {nomscategorie,imagescategorie,categorieID}=req.body
    const id  = req.params.id;

    try {
     const scategorie = await prisma.scategories.update({
        data: {
            nomscategorie: nomscategorie,
            imagescategorie: imagescategorie,
            categorieID:categorieID
        },
            where: { id: Number(id)},
        })
    res.json(scategorie);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});

// Supprimer un scategories
router.delete('/:id', async (req, res)=> {
    const  id  = req.params.id;
    try {
     await prisma.scategories.delete({
        where: { id: Number(id) },
    })

    res.json({ message: "scategory "+ id +" deleted successfully." });
} catch (error) {
    res.status(404).json({ message: error.message });
    }
});

module.exports = router;
```

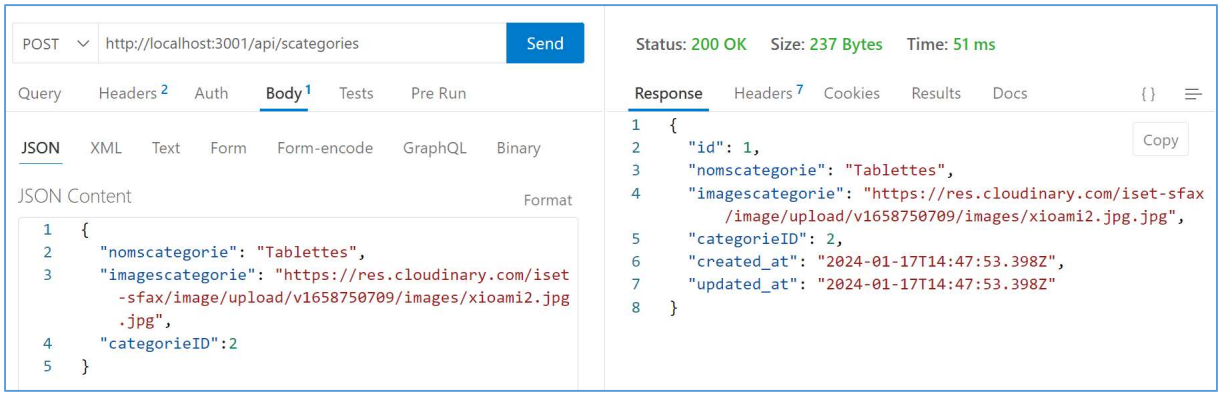
Modifier app.js pour faire appel à la route

app.js >

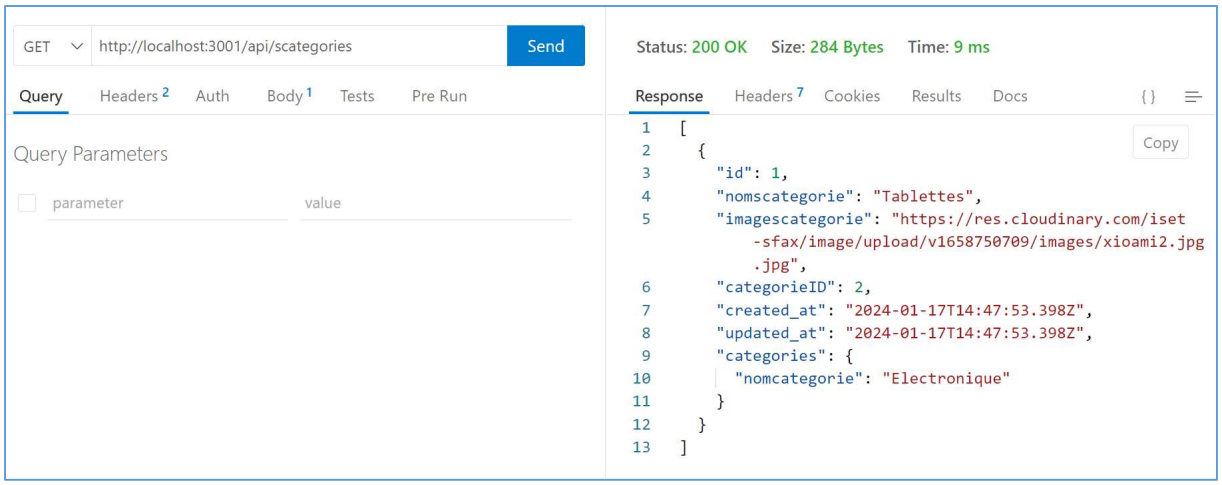
```
const scategoriesRouter =require("./routes/scategories.route")
app.use('/api/scategories', scategoriesRouter);
```

Test des APIs

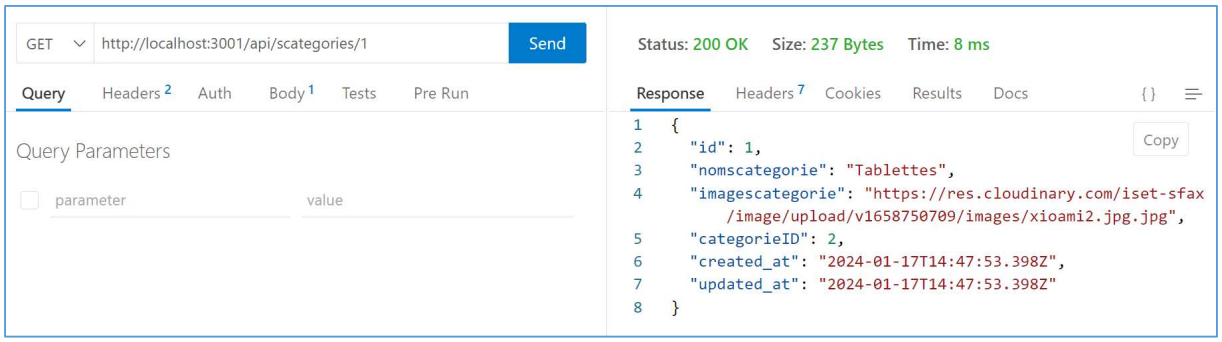
POST : http://localhost:3001/api/scategories/



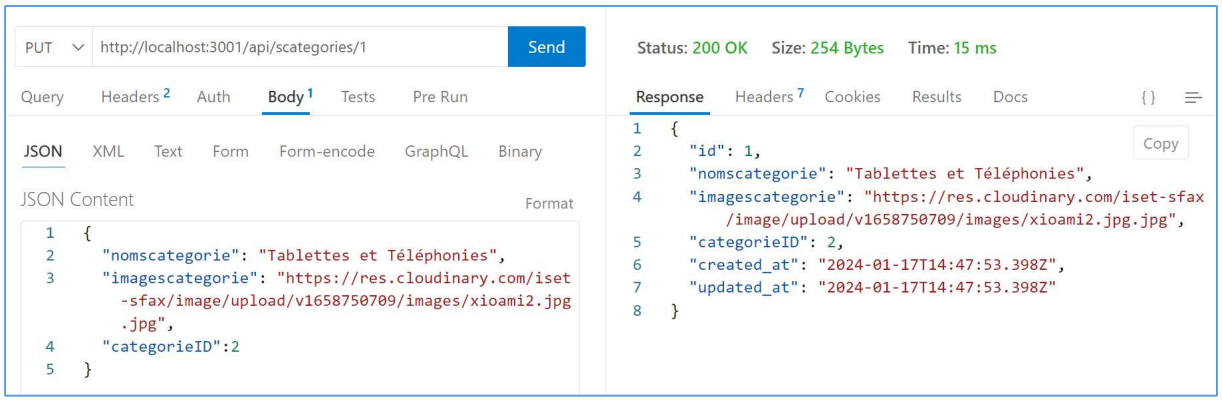
GET : http://localhost:3001/api/scategories/



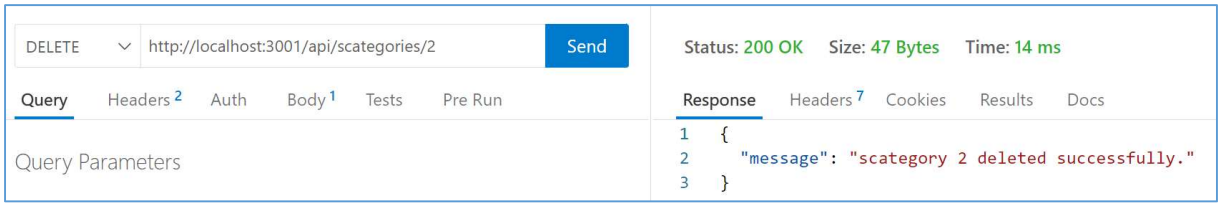
GET : http://localhost:3001/api/scategories/1



PUT : http://localhost:3001/api/scategories/1

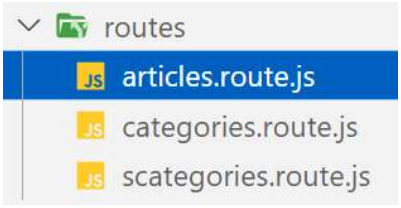


DELETE : http://localhost:3001/api/scategories/1



Création de la route vers les APIs articles

Sous le dossier /routes créer le fichier articles.route.js




```
const express = require('express');
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()
const router = express.Router();

//Ajout d'un article
router.post('/', async (req, res, )=> {
    const { designation ,  marque , reference , qtestock , prix , imageart , scategorieID }=req.body
    try {
        const article = await prisma.articles.create({
            data: {
                designation : designation,
                marque      : marque,
                reference   : reference,
                qtestock    : qtestock,
                prix        : prix,
```

```
                imageart     : imageart,
                scategorieID : scategorieID
            }
        })

        res.json(article)
      } catch (error) {
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});

// afficher la liste des articles.
router.get('/', async (req, res, )=> {
    try {
        // articles est le nom du model précisé dans prisma.schema
        const Articles = await prisma.articles.findMany({
            include: {
                scategories: {
                  include: {
                    categories: true,
                  },
                },
              },
          })

            res.json(Articles)

        } catch (error) {
         res.status(500).json({
        message: "Something went wrong",
       })
       }
});

// afficher un article.
router.get('/:id', async (req, res, )=> {
    const { id } = req.params

    try {
        const article = await prisma.articles.findUnique({
            where: {
                id: Number(id),
            }
        })

       res.json(article)
```

```
      } catch (error) {
        res.status(500).json({
          message: "Something went wrong",
        })
      }

});

// modifier un article

router.put('/:id', async (req, res)=> {
    const { designation ,  marque , reference , qtestock , prix , imageart ,
scategorieID }=req.body
    const id  = req.params.id;

    try {
     const article = await prisma.articles.update({
        data: {
            designation : designation,
            marque       : marque,
            reference    : reference,
            qtestock     : qtestock,
            prix         : prix,
            imageart     : imageart,
            scategorieID : scategorieID
        },
            where: { id: Number(id)},
        })
    res.json(article);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});

// Supprimer un article
router.delete('/:id', async (req, res)=> {
    const  id  = req.params.id;
    try {
     await prisma.articles.delete({
        where: { id: Number(id) },
    })

    res.json({ message: "article "+ id +" deleted successfully." });
} catch (error) {
    res.status(404).json({ message: error.message });
    }
});

module.exports = router;
```

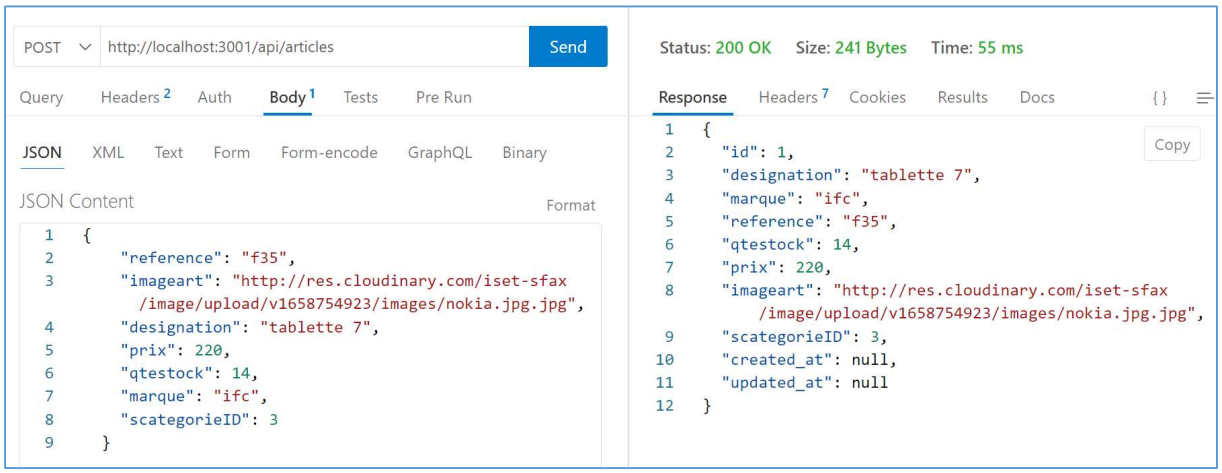
Modifier app.js pour faire appel à la route

app.js >

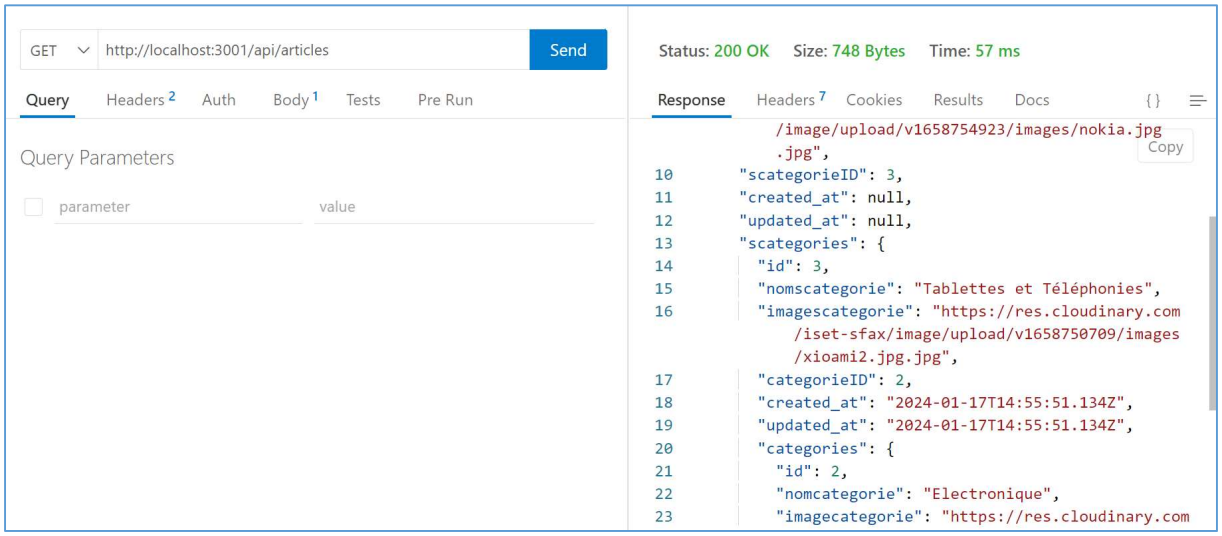
```
const articlesRouter =require("./routes/articles.route")
app.use('/api/articles', articlesRouter);
```

Test des APIs

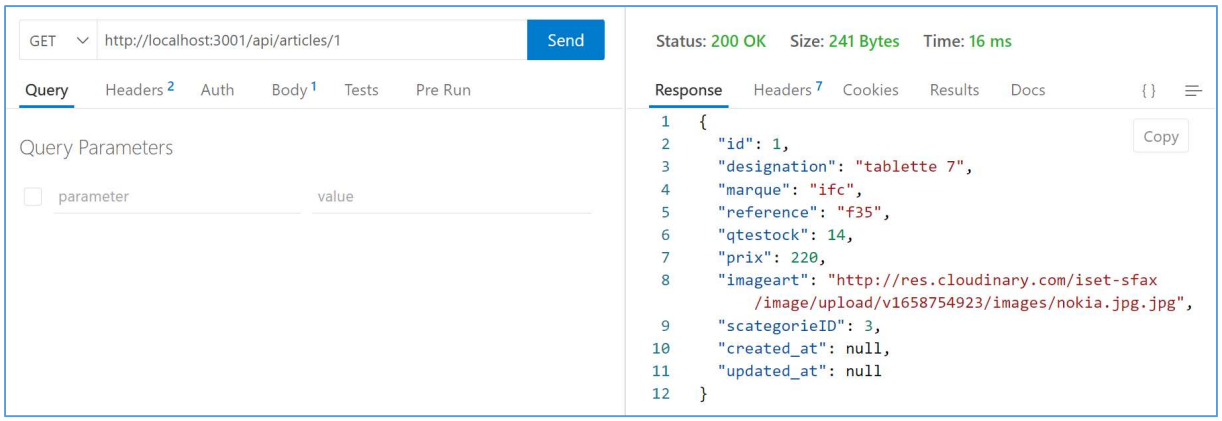
POST : http://localhost:3001/api/articles/



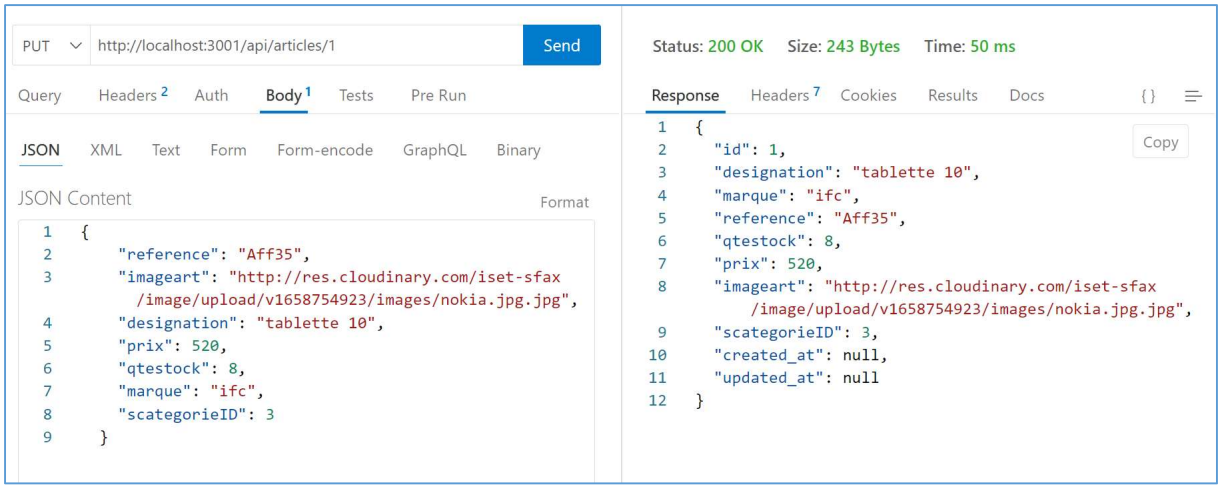
GET : http://localhost:3001/api/articles/



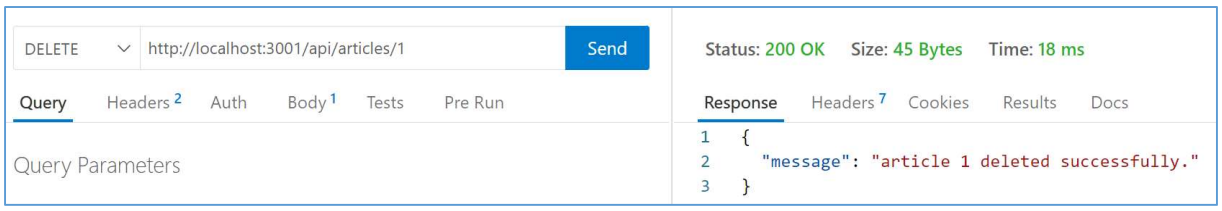
GET : http://localhost:3001/api/articles/1



PUT : http://localhost:3001/api/articles/1



DELETE : http://localhost:3001/api/articles/1



Affichage avec pagination

On va modifier la requête GET pour afficher les données avec pagination et limite. Si c'est valeur ne sont pas indiquées, la totalité des enregistrements seront affichées.

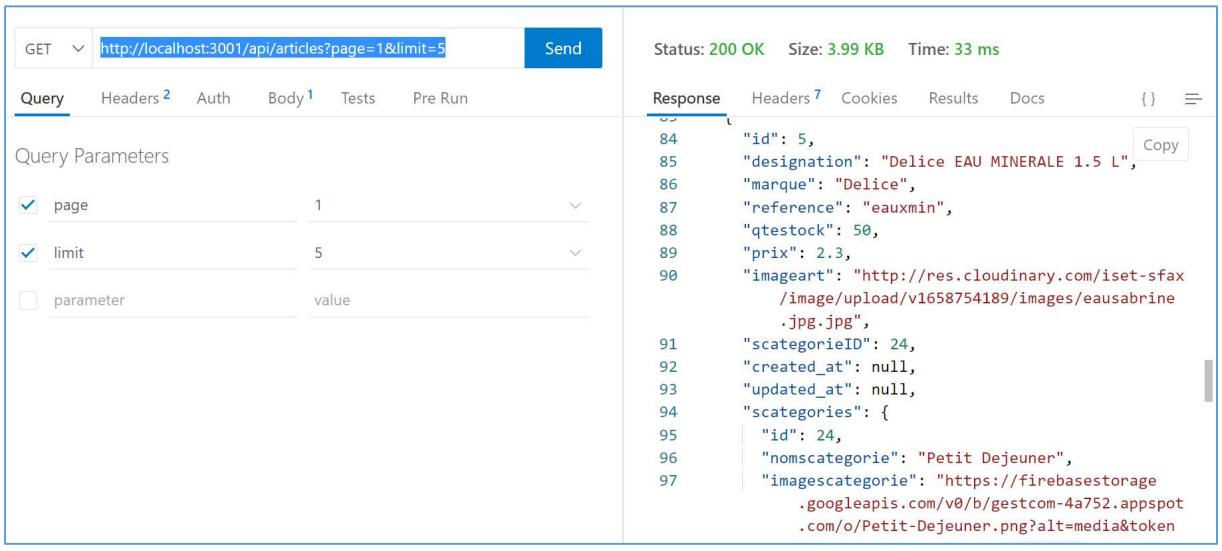
routes > articles.route.js >

```
router.get('/', async (req, res) => {

   try {
        const page_str = req.query.page;
```

```
    const limit_str = req.query.limit;

    const page = page_str ? parseInt(page_str, 10) : 1;
    const limit = limit_str ? parseInt(limit_str, 10) : 10;
    const skip = (page - 1) * limit;
      const articles = await prisma.articles.findMany({
        skip,
        take: limit,
        include: {
          scategories: {
            include: {
              categories: true,
            },
          },
        },
      });

      res.json(articles);
    } catch (error) {
      res.status(500).json({
        message: "Something went wrong",
        error: error.message,
      });
    }
  });
```

Test de l'API

GET : http://localhost:3001/api/articles?page=1&limit=5



Liste des articles pour une catégorie

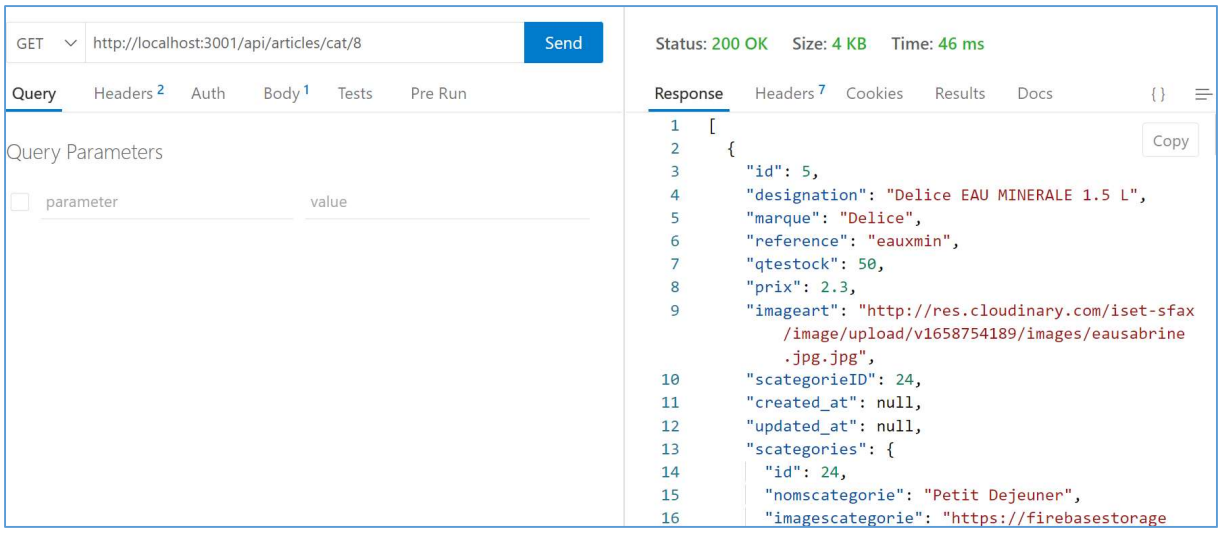
On veut afficher les articles reliés à travers les sous catégories à une catégorie donnée via la requête.

routes > JS articles.route.js >

```
//liste des articles pour une catégorie donnée
 router.get('/cat/:idCateg', async (req, res) => {
   const { idCateg } = req.params;

   try {
       const articles = await prisma.articles.findMany({
           where: {
               scategories: {
                   categorieID: Number(idCateg)
               }
           },
           include: {
               scategories: true,
             },
       });

       res.json(articles);
   } catch (error) {
       res.status(500).json({
           message: "Something went wrong",
           error: error.message,
       });
   }
});
```

Test de l'API

GET : http://localhost:3001/api/articles/cat/8



GET : http://localhost:3001/api/articles/cat/2