Atelier N°10 Application MERN : Authentification avec JWT

Application : Site de Commerce en ligne

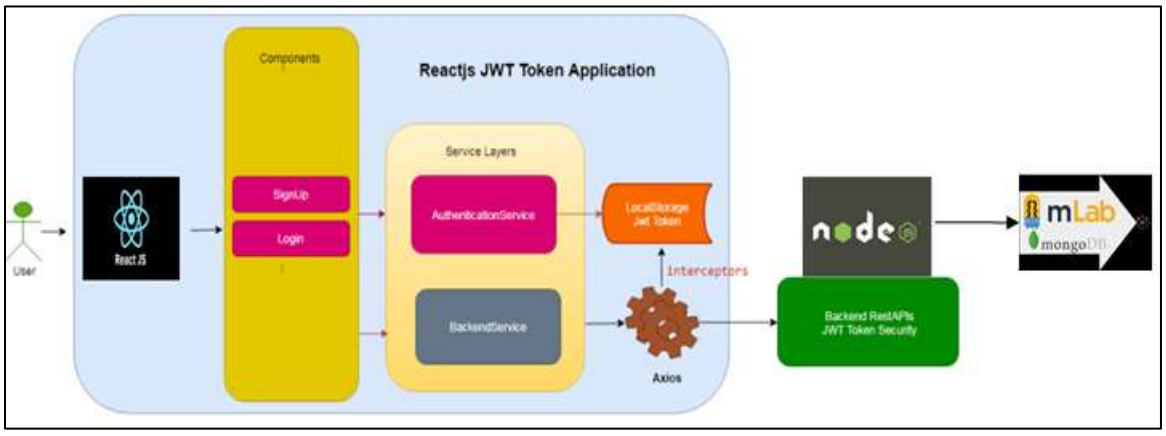
Dans cet atelier, nous allons mettre en place une application full-stack MERN, qui va intégrer la technique de JWT pour l'opération d'authentification.

JSON Web Token est un standard ouvert défini dans la RFC 7519. Il permet l'échange sécurisé de jetons entre plusieurs parties. Cette sécurité de l'échange se traduit par la vérification de l'intégrité des données à l'aide d'une signature numérique.

JWT est un jeton permettant d'échanger des informations de manière sécurisée. Ce jeton est composé de trois parties, dont la dernière, la signature, permet d'en vérifier la légitimité. JWT est souvent utilisé pour offrir une authentification stateless au sein des applications.

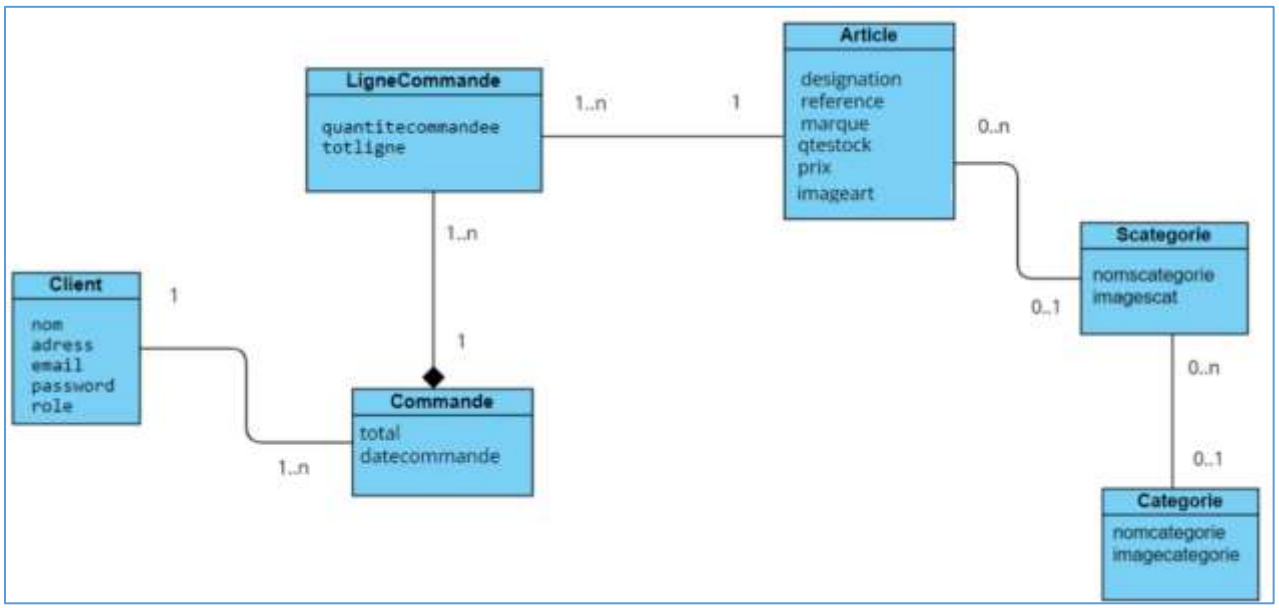
Au niveau de la base de données, on va utiliser mLab qui est un service de base de données cloud entièrement géré qui héberge les bases de données MongoDB. mLab fonctionne sur les fournisseurs de cloud Amazon, Google et Microsoft Azure, et s'est associé à des fournisseurs de plate-forme en tant que service.

Concernant le design de la partie front end, on va utiliser la bibliothèque Material UI. Le Material Design est un ensemble de règles de design proposées par Google et qui s'appliquent à l'interface graphique des logiciels et applications.



Partie I : Backend avec NodeJS

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



1. Tout d'abord, on doit créer un nouveau projet pour gérer le back end de notre application.

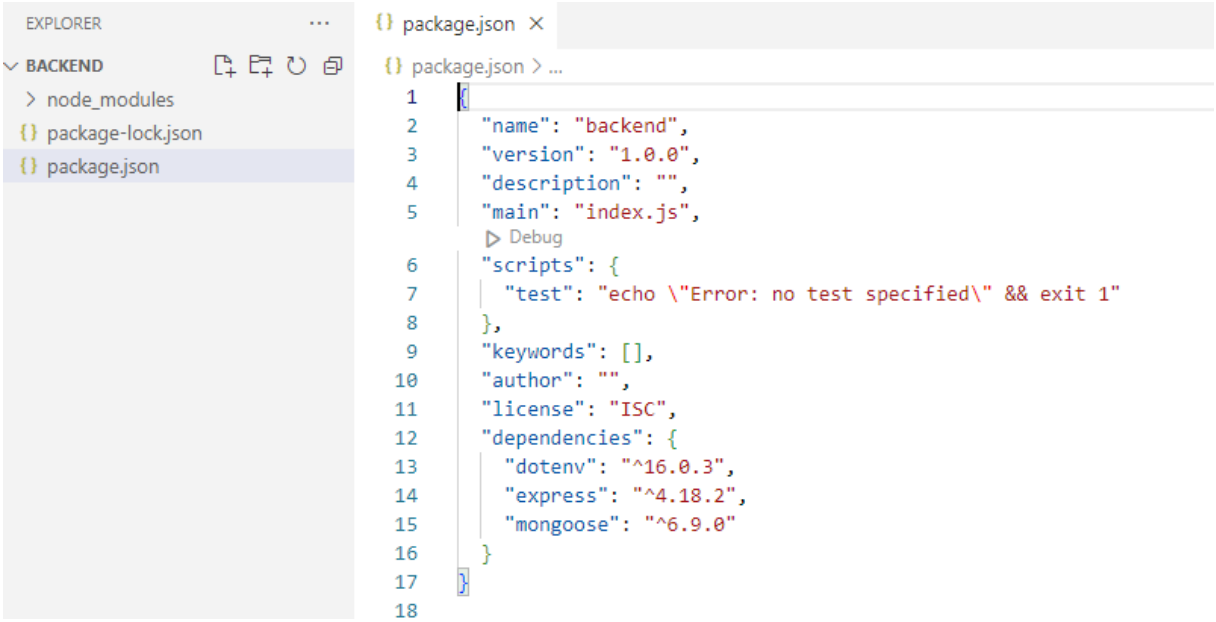
```
mkdir ecommerce\backend

cd ecommerce\backend

c:\ecommerce\backend>npm init -y
```

2. Démarrer l'application avec visual studio code

```
code .
```

3. Faire l'installation des dépendances suivantes :
   Menu terminal→new terminal

```
npm i -g nodemon

npm i express mongoose  dotenv
```



4. Commencer par préparer les variables d'environnement dans un fichier intitulé .env

```
ENV=DEVELOPMENT
DATABASE=mongodb://127.0.0.1:27017/DatabaseCommerce
DATABASECLOUD=mongodb+srv://hassan:messk2015@cluster0.yfhhu.mongodb.net/DatabaseCommerce?retryWrites=true&w=majority
PORT=3001
```

5. Les routes à indiquer dans le problème :

Classe Route	Route	Méthodes
Authentification	/api/auth	/login /register /user
User.route.js	/api/users	get
categorie.route.js	/api/categories /api/categories /:id	post,get put,delete
scategorie.route	/api/scategories /api/scategories/:id	post,get put,delete
article.route.js	/api/articles /api/articles/:id	post,get put,delete
commande.route.js	/api/commandes	post,get

	/api/commandes/:id	put,delete
panier.route.js	/api/panier /api/panier/:id	post ,get put,delete

6. Mettre le code suivant dans un nouveau fichier appelé app.js :

```
const express=require('express');
const  mongoose =require("mongoose")
const dotenv =require('dotenv')
dotenv.config()
const app = express();

//BodyParser Middleware
app.use(express.json());
mongoose.set("strictQuery", false);
// Connexion à la base données
mongoose.connect(process.env.DATABASE,{
 useNewUrlParser: true,
 useUnifiedTopology: true
 })
 .then(() => {console.log("Connexion à la base de données réussie");
}).catch(err => {
 console.log('Impossible de se connecter à la base de données', err);
 process.exit();
});

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

app.listen(process.env.PORT, () => {
 console.log(`Server is listening on port ${process.env.PORT}`); });
export default app
```

Exécuter le code avec la commande nodemon app

7. Créer les modèles
   Créer le dossier models dans lequel on mettra tout notre modèle de base de données

models/categorie.js

```
const mongoose =require("mongoose")
const categorieSchema=mongoose.Schema({
    nomcategorie:{ type: String, required: true,unique:true },
```

```
    imagecategorie :{ type: String, required: false }
    })
module.exports=mongoose.model('categorie',categorieSchema)
```

models/scategorie.js

```
const  mongoose =require("mongoose")
const Categorie =require("./categorie.js");
const scategorieSchema=mongoose.Schema({
    nomscategorie:{ type: String, required: true },
    imagescat :{ type: String, required: false },
    categorieID: {type:mongoose.Schema.Types.ObjectId,
    ref:Categorie}
    })

    module.exports=mongoose.model('scategorie',scategorieSchema)
```

models/article.js

```
const  mongoose  =require("mongoose")

const Scategorie =require("./scategorie.js");

const articleSchema=mongoose.Schema({

    reference:{ type: String, required: true,unique:true },

    designation:{ type: String, required: true,unique:true },

    prix:{ type: Number, required: false },

    marque:{ type: String, required: true },

    qtestock:{ type: Number, required: false },

    imageart:{ type: String, required: false },

    scategorieID: {type:mongoose.Schema.Types.ObjectId,

    ref:Scategorie}

    })

    module.exports=mongoose.model('article',articleSchema)
```

I. CRUD categories

1. On va maintenant créer les routes pour la classe **categorie** :

routes/categorie.route.js

```
var express = require('express');
var router = express.Router();


// afficher la liste des categories.
router.get('/', async (req, res, )=> {
});
// créer un nouvelle catégorie
router.post('/', async (req, res) =>  {
});
// chercher une catégorie
router.get('/:categorieId',async(req, res)=>{
});
// modifier une catégorie
router.put('/:categorieId', async (req, res)=> {
});
// Supprimer une catégorie
router.delete('/:categorieId', async (req, res)=> {

});
module.exports = router;
```

2. Méthode pour afficher la liste des catégories

```
const express = require('express');
const router = express.Router();
const Categorie=require("../models/categorie")

// afficher la liste des categories.
router.get('/', async (req, res )=> {
```

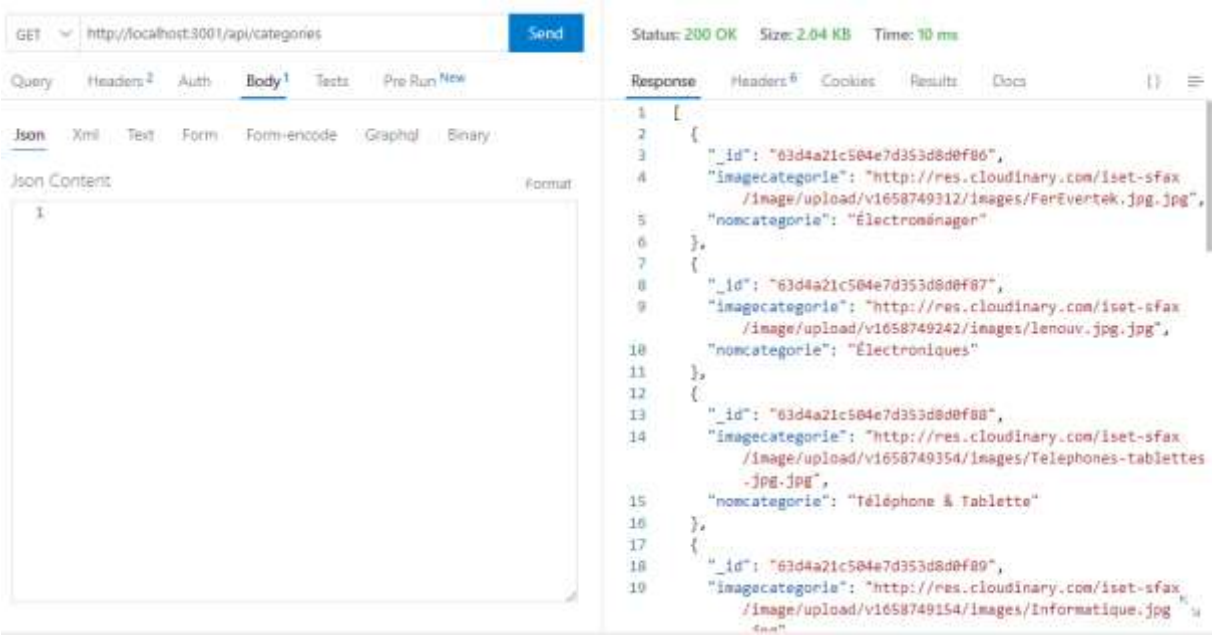
```
    try {
        const cat = await Categorie.find();

        res.status(200).json(cat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
```

Dans le fichier app.js ajouter la route suivante :

```
const categorieRouter =require("./routes/categorie.route")
app.use('/api/categories', categorieRouter);
```

Tester le service crée

Installer dans visual studio code l'utilitaire **thunder client**

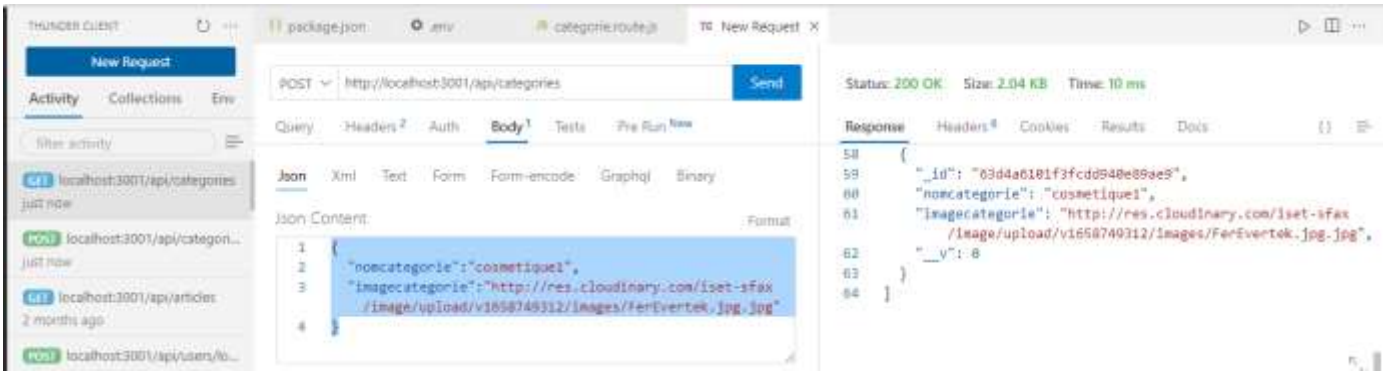


3. Méthode pour créer une catégorie

```
// créer un nouvelle catégorie
router.post('/', async (req, res) =>  {
    const { nomcategorie, imagecategorie} = req.body;
    const newCategorie = new Categorie({nomcategorie:nomcategorie,
imagecategorie:imagecategorie})
```

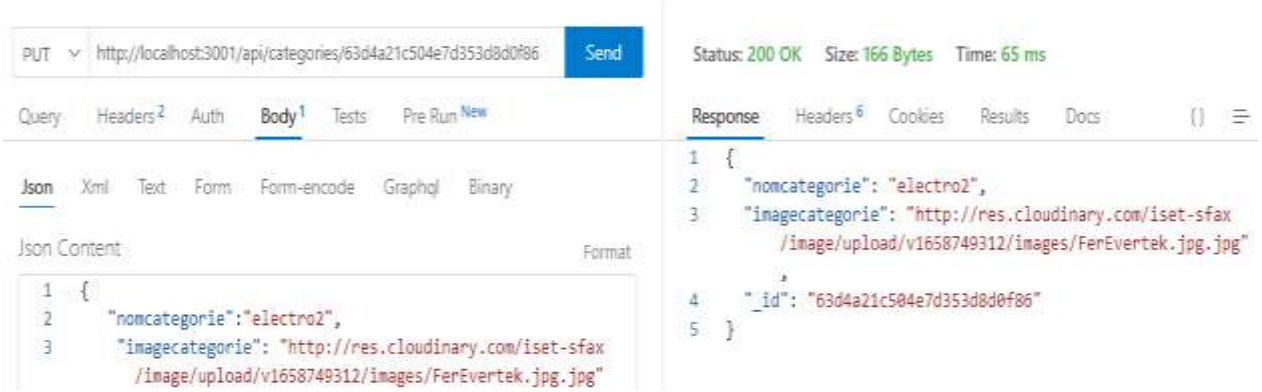
```
    try {
        await newCategorie.save();
        res.status(200).json(newCategorie );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
```

Démarrer thunder client pour tester l'ajout dans la base de données



4. Méthode pour modifier une catégorie

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

    try {

    const cat1 = {
nomcategorie:nomcategorie,imagecategorie:imagecategorie, _id:id };
    await Categorie.findByIdAndUpdate(id, cat1);

    res.json(cat1);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});
```



5. Méthode pour supprimer une catégorie

```
// Supprimer une catégorie
router.delete('/:categorieId', async (req, res)=> {
    const  id  = req.params.categorieId;
    await Categorie.findByIdAndDelete(id);
    res.json({ message: "categorie deleted successfully." });

});
```

6. Méthode pour chercher une catégorie

```
// chercher une catégorie
router.get('/:categorieId',async(req, res)=>{
    try {
        const cat = await Categorie.findById(req.params.categorieId);

        res.status(200).json(cat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
```

Fichier categorie.route.js complet

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

```
const router = express.Router();
const Categorie=require("../models/categorie")

// afficher la liste des categories.
router.get('/', async (req, res, )=> {
    try {
        const cat = await Categorie.find();

        res.status(200).json(cat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }

});
// créer un nouvelle catégorie
router.post('/', async (req, res) =>  {
    const { nomcategorie, imagecategorie} = req.body;
    const newCategorie = new Categorie({nomcategorie:nomcategorie,
imagecategorie:imagecategorie})

    try {
        await newCategorie.save();

        res.status(200).json(newCategorie );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }


});
// chercher une catégorie
router.get('/:categorieId',async(req, res)=>{
    try {
        const cat = await Categorie.findById(req.params.categorieId);
```

```
        res.status(200).json(cat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
// modifier une catégorie
router.put('/:categorieId', async (req, res)=> {
    const { nomcategorie, imagecategorie} = req.body;
    const id  = req.params.categorieId;
    try {
    const cat1 = {
nomcategorie:nomcategorie,imagecategorie:imagecategorie, _id:id };
    await Categorie.findByIdAndUpdate(id, cat1);
     res.json(cat1);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});
// Supprimer une catégorie
router.delete('/:categorieId', async (req, res)=> {
    const  id  = req.params.categorieId;
    await Categorie.findByIdAndDelete(id);

    res.json({ message: "categorie deleted successfully." });
});
module.exports = router;
```

II. CRUD Sous categories

créer le fichier routes/scategorie.route.js

```
const express = require('express');
const router = express.Router();
const SCategorie=require("../models/scategorie")
```

```
// afficher la liste des categories.
router.get('/', async (req, res, )=> {
    try {
        const scat = await SCategorie.find();

        res.status(200).json(scat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }

});
// créer un nouvelle catégorie
router.post('/', async (req, res) =>  {
    const { nomscategorie, imagescat,categorieID} = req.body;
    const newSCategorie = new SCategorie({nomscategorie:nomscategorie,
imagescat:imagescat,categorieID:categorieID })

    try {
        await newSCategorie.save();

        res.status(200).json(newSCategorie );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }


});
// chercher une sous catégorie
router.get('/:scategorieId',async(req, res)=>{
    try {
        const scat = await
SCategorie.findById(req.params.scategorieId);
```

```
        res.status(200).json(scat);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
// modifier une catégorie
router.put('/:scategorieId', async (req, res)=> {
    const { nomscategorie, imagescat,categorieID} = req.body;
    const id  = req.params.scategorieId;

    try {

    const scat1 = {
nomscategorie:nomscategorie,imagescat:imagescat,categorieID:categorieID
, _id:id };

    await SCategorie.findByIdAndUpdate(id, scat1);

    res.json(scat1);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});
// Supprimer une catégorie
router.delete('/:scategorieId', async (req, res)=> {
    const  id  = req.params.scategorieId;
    await SCategorie.findByIdAndDelete(id);

    res.json({ message: "sous categorie deleted successfully." });

});
module.exports = router;
```

ajouter la route du fichier dans app.js

```
const scategorieRouter =require("./routes/scategorie.route")

app.use('/api/scategories', scategorieRouter);
```

Tester les CRUDs en utilisant **thunder client**

III. CRUD articles

routes/article.route.js

```
const express = require('express');
const router = express.Router();
const Article=require("../models/article")

// afficher la liste des articles.
router.get('/', async (req, res, )=> {
    try {
        const articles = await Article.find();

        res.status(200).json(articles);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }

});
// créer un nouvel article
router.post('/', async (req, res) =>  {

    const nouvarticle = new Article(req.body)

    try {
        await nouvarticle.save();

        res.status(200).json(nouvarticle );
    } catch (error) {
        res.status(404).json({ message: error.message });
    }


});
// chercher un article
router.get('/:articleId',async(req, res)=>{
    try {
        const art = await Article.findById(req.params.articleId);
```

```
        res.status(200).json(art);
    } catch (error) {
        res.status(404).json({ message: error.message });
    }
});
// modifier un article


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

    try {

    const art1 = {
reference:reference,designation:designation,prix:prix,marque:marque,qte
stock:qtestock,imageart:imageart,scategorieID:scategorieID, _id:id };

    await Article.findByIdAndUpdate(id, art1);

    res.json(art1);
    } catch (error) {
    res.status(404).json({ message: error.message });
    }
});
// Supprimer un article
router.delete('/:articleId', async (req, res)=> {
    const  id  = req.params.articleId;
    await Article.findByIdAndDelete(id);

    res.json({ message: "article deleted successfully." });

});
module.exports = router;
```

Dans le fichier app.js ajouter la route de article

```
const articleRouter =require("./routes/article.route")

app.use('/api/articles', articleRouter);
```

Tester les CRUDs de article en utilisant thunder client