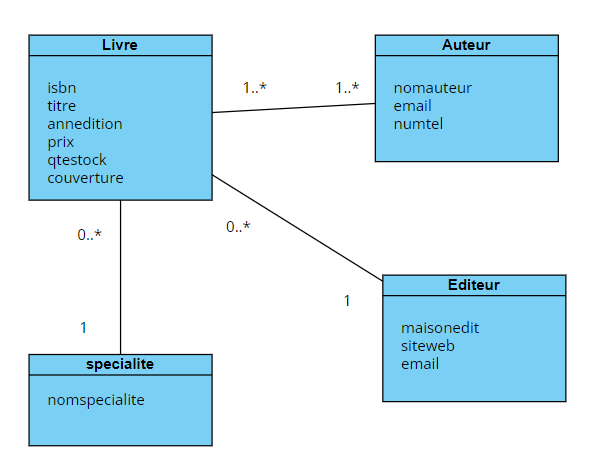
Atelier 2 NextJS : Connexion à MongoDB

Notre exemple d’API portera sur un catalogue de livres. Plus précisément, en plus des informations basiques sur un livre, notre API fournira les données relatives à la spécialité, l’auteur et l’éditeur. A cet effet, on considère la base de données déduite d’une partie du diagramme de classes UML de l’application bibliothèque :



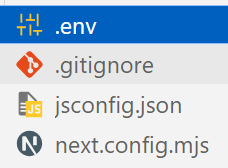
# Installation des packages

npm install mongoose

npm install axios

# Fichier .env

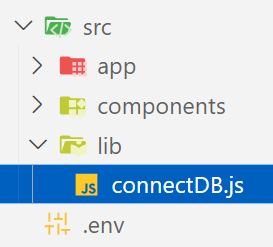
Créez un fichier .env dans la racine de votre projet Next.js et mettez-y la chaîne de connexion MongoDB.



NEXT\_PUBLIC\_DATABASE\_URL=mongodb://127.0.0.1:27017/bibliothequedb

# Fichier de configuration

Dans votre projet, créez un dossier lib sous le dossier /src. Dans ce dossier lib, créez un fichier nommé connectDB.js



import mongoose from "mongoose";

const DATABASE\_URL = process.env.NEXT\_PUBLIC\_DATABASE\_URL;

const connectDB=async()=> {

try{

await mongoose.connect(DATABASE\_URL);

console.log("connexion a la base de donnée réussie")

}catch(error){

    throw new Error("erreur de connexion a la base de données")

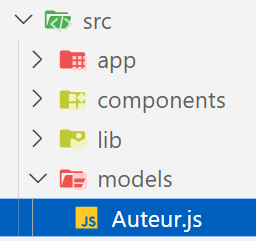
}

}

export default connectDB;

# Le Model

Maintenant, sous /src, créez un dossier nommé models pour stocker tous vos schémas. Dans le dossier models, créez un fichier nommé Auteur.js



import mongoose from "mongoose"

var auteurSchema = mongoose.Schema({

    nomauteur: String,

    email: String,

    numtel: String

},

    {

        timestamps: true

    }

);

const Auteur = mongoose.models.Auteur || mongoose.model('Auteur', auteurSchema);

//ce sont les auteurs du modèle Mongo s'ils existent ou un modèle mongoose

//le schéma auteur est déjà là donc il y a une vérification s'il existe déjà il ne faut pas essayer de le créer

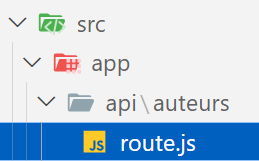
//pour éviter l'erreur OverwriteModelError: Cannot overwrite `Auteur` model once compiled.

export default Auteur

# API Routes

Définir les points de terminaison de l'API : dans le répertoire app/api, créez des fichiers pour chacune de vos routes API.

Dans notre cas, app/api/auteurs/route.js



import { HttpStatusCode } from 'axios';

import Auteur from '@/models/Auteur';

import {  NextResponse } from 'next/server';

import connectDB from '@/lib/connectDB';

export async function POST(req) {

    try {

        const body = await req.json();

        const newAuteur = new Auteur(body)

        const auteur = await newAuteur.save();

        return NextResponse.json(

                { auteur, message: 'Your author has been created' },

                { status: HttpStatusCode.Created },

            );

    } catch (error) {

        return NextResponse.json({ message: error }, { status: HttpStatusCode.BadRequest });

    }

}

export async function GET() {

    try {

        await connectDB();

        const auteurs = await Auteur.find({}, null, {sort: {'\_id': -1}});

        return NextResponse.json(auteurs );

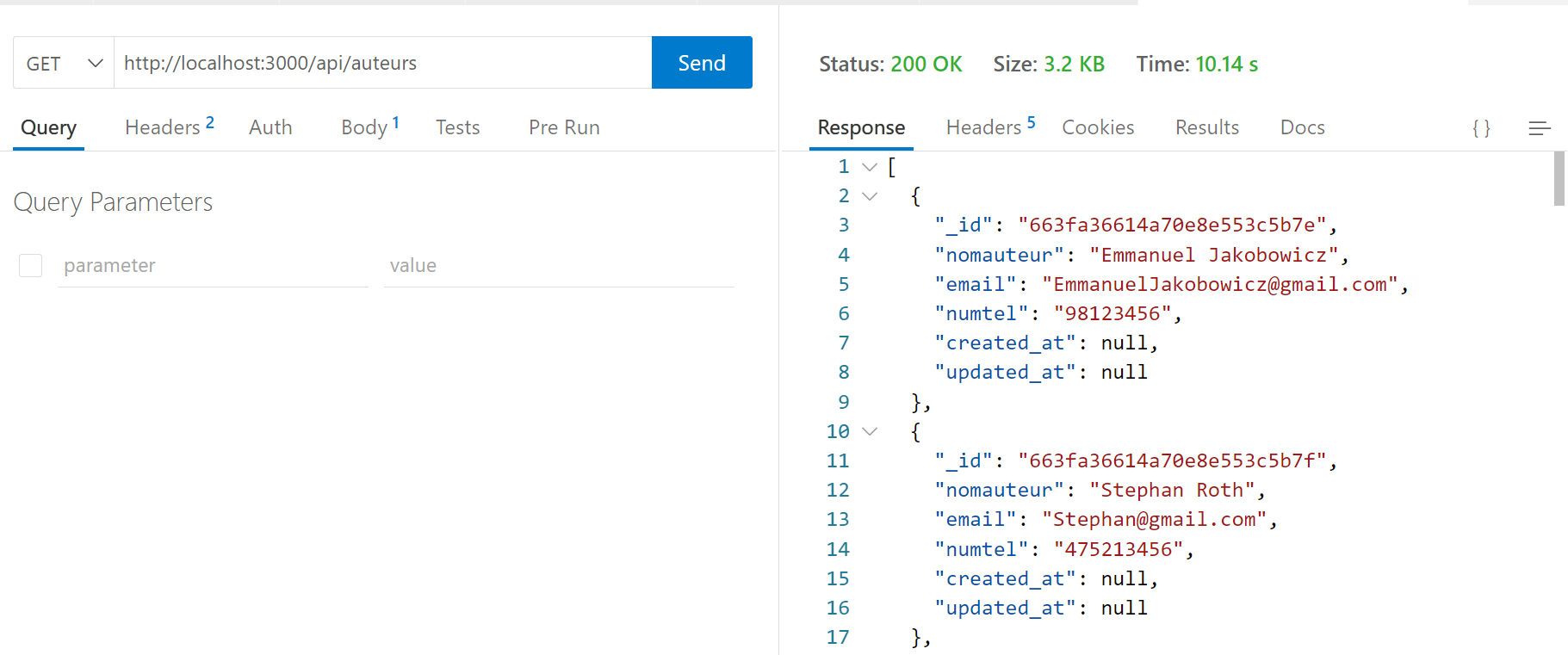
    } catch (error) {

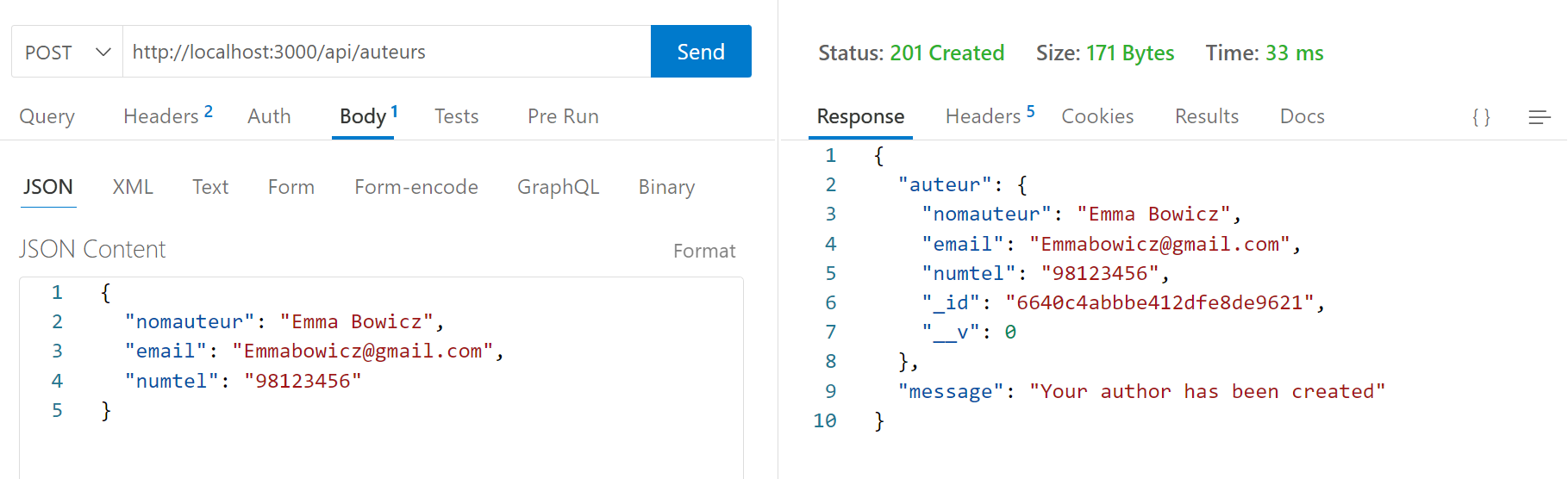
        return NextResponse.json({ error });

    }

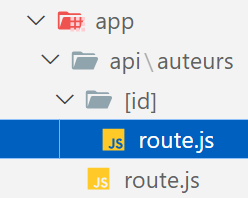
}

Test Api





Dans le répertoire app/api/auteurs créer le dossier [id] ayant le fichier route.js



import { HttpStatusCode } from 'axios';

import Auteur from '@/models/Auteur';

import {  NextResponse } from 'next/server';

export async function GET(\_, { params }) {

    try {

        const auteur = await Auteur.findById(params.id);

        if (auteur) {

            return NextResponse.json(auteur);

        }

        return NextResponse.json({ message: `Auteur ${params.id} not found` }, { status: HttpStatusCode.NotFound });

    } catch (error) {

        return NextResponse.json({ message: error }, { status: HttpStatusCode.BadRequest });

    }

}

export async function PUT(req, { params }) {

    try {

            const body= await req.json();

            const auteur = await Auteur.findByIdAndUpdate(

                params.id,

                { $set: body },

              { new: true }

            );

            return NextResponse.json({ auteur });

        } catch (error) {

        return NextResponse.json({ message: error }, { status: HttpStatusCode.BadRequest });

    }

}

export async function DELETE(\_, { params }) {

    try {

        await Auteur.findByIdAndDelete(params.id);

        return NextResponse.json({ message: `Auteur ${params.id} has been deleted` });

   } catch (error) {

        return NextResponse.json({ message: error }, { status: HttpStatusCode.BadRequest });

    }

}

Test Api

