API REST express/node js

Documentation de l'api rest

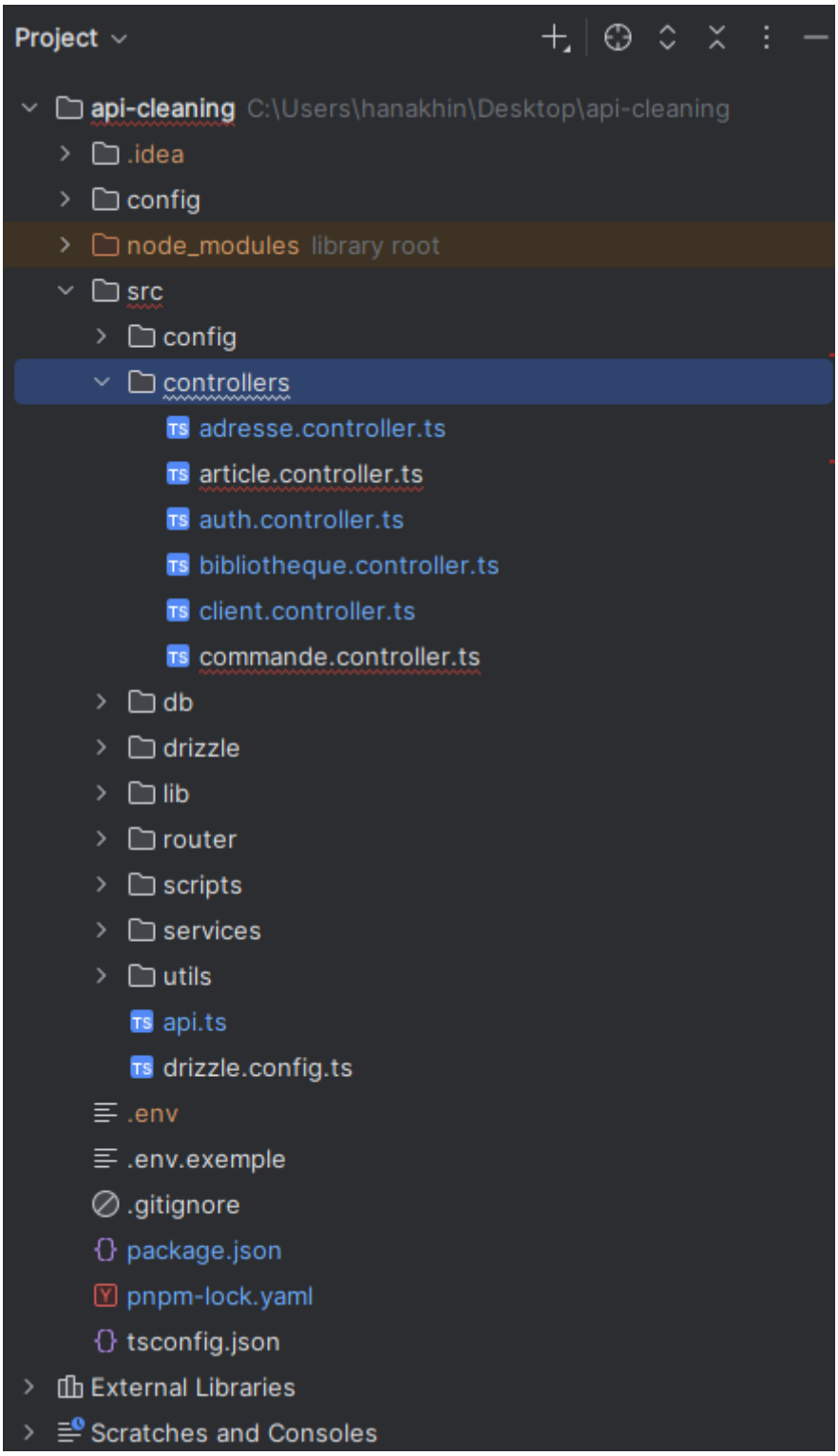
Avoir installé npm sur son poste avec ce lien https://nodejs.org/fr/download
installer pnpm avec 'npm i -g pnpm'

lancer le serveur de l'api avec la commande presente dans le package.json a la racine du projet

exemple : 'pnpm start' pour l'api E-CARE cleaning

E-CARE API CLEANING

Processus de création d'une route -> controller -> service



1)Créer le controller

```
import {clientService} from "../services/client.service";
import {getDb} from "../lib/mssql";
import {sendError, sendSuccess} from "../utils/helpers/response.helper";

export const clientController = {  Show usages  hanakhin *

    all: async (req:any,res:any) : Promise<Response<any, Record<string, any>... => {
        try {
            const pool : ConnectionPool  = getDb(Number( value: 1))
            const result : { cli: IResult<any>; cde: IResult<any> }  = await clientService.getAll(pool)
            return sendSuccess(res, result,  status: 200)
        } catch (err) {
            console.log(err)
            return sendError(res, err,  status: 500)
        }
    },
}
```

un controller contient tout ce qui est en rapport avec les requetes/responses, c'est ici qu'on fait le try catch et qu'on renseigne par exemple le pool (db) etc .

on oublie pas de le rendre exportable avec le "export" const , et ensuite on crée nos methodes.

On peut le faire de plusieurs façons comme vu ici

```
import {clientService} from "../services/client.service";
import {getDb} from "../lib/mssql";
import {sendError, sendSuccess} from "../utils/helpers/response.helper";

export const clientController = {  Show usages  hanakhin *

    all: async (req:any,res:any) : Promise<Response<any, Record<string, any>... => {...},

}
export async function all (req:any,res:any) : Promise<void>  {}  no usages  new *

export const all2 : () => void  = () : void => {}  no usages  new *
```

trois méthodes differentes qui font la meme chose , question de préference.

donc dans le controller on appelle le service clientService.[la methode qu'on veut] ([le parametre que la methode attends]).

```
const result : { cli: IResult<any>; cde: IResult<any> }  = await clientService.getAll(pool)
```

2)Créer le service

```
export const clientService = {  Show usages  hanakhin *
    getAll: async (pool: mssql.ConnectionPool) : Promise<{ cli: IResult<any>; cde: IResult... => {
            const cliResult : IResult<any> = await pool.request().query( command: `
            SELECT TOP 10 cli.*
            FROM dbo.cli AS cli
            LEFT JOIN dbo.tst AS tst ON tst.ctiers = COALESCE(NULLIF(cli.code, ''), cli.censeigne)
            LEFT JOIN dbo.tsd AS tsd ON tsd.ctiers = COALESCE(NULLIF(cli.code, ''), cli.censeigne)

        `);
            const cdeResult : IResult<any> = await pool.request().query( command: `
            SELECT TOP 10 cde.*, vtl.*
            FROM dbo.cde AS cde
            INNER JOIN dbo.vtl AS vtl ON vtl.numero = cde.numero
            INNER JOIN dbo.cli AS cli ON cli.code = cde.cclient
            WHERE cde.categorie = 'F'
        `);

            return {'cli':cliResult,'cde': cdeResult};
    }
}
```

dans le service on met tout ce qui est appel en db, et on retourne le resultat, le controller ce charge de la gestion d'erreur etc .

On fait toujours comme ca :

```
        const cdeResult : IResult<any> = await pool.request().query( command: `
        SELECT TOP 10 cde.*, vtl.*
        FROM dbo.cde AS cde
        INNER JOIN dbo.vtl AS vtl ON vtl.numero = cde.numero
        INNER JOIN dbo.cli AS cli ON cli.code = cde.cclient
        WHERE cde.categorie = 'F'
    `);
```

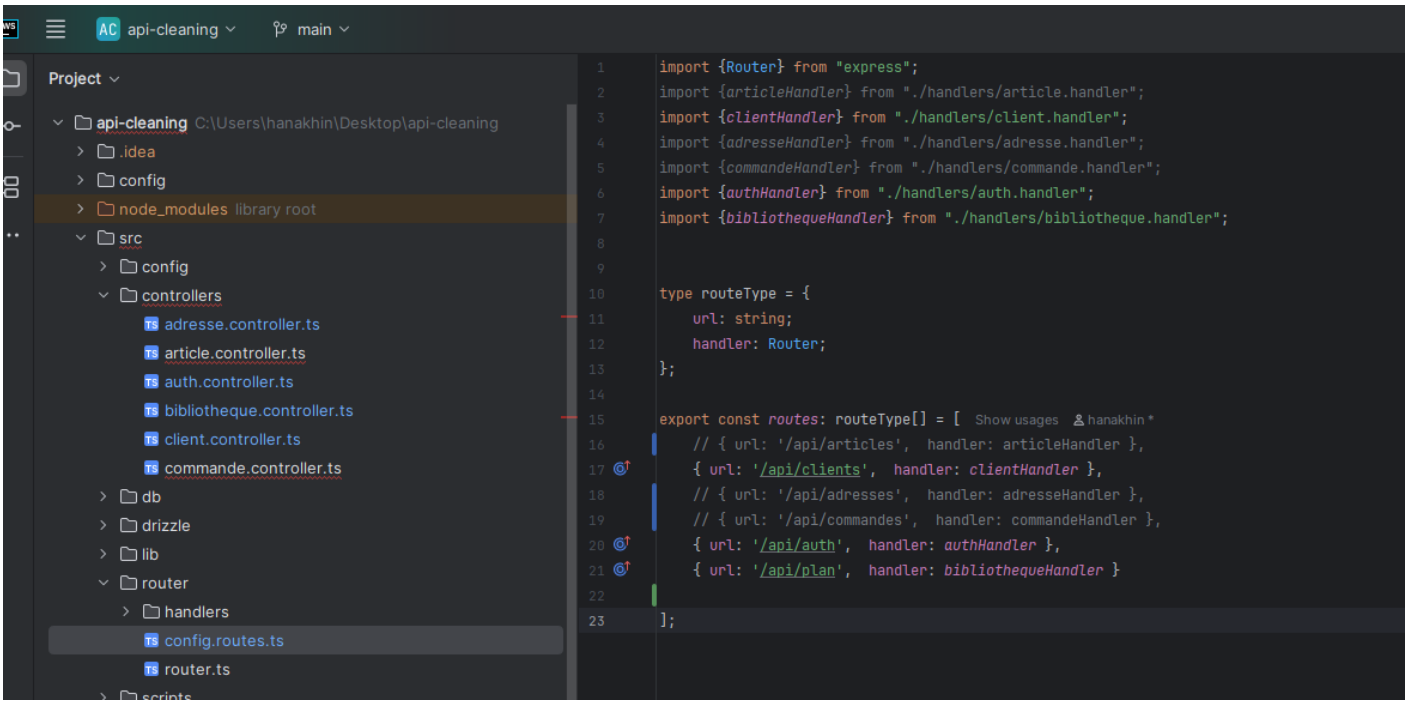
const [nom] = await (pour l'asynchrone) pool(parametre qu'on a typé).request().query(`[notre requete]`);

et on return le resultat

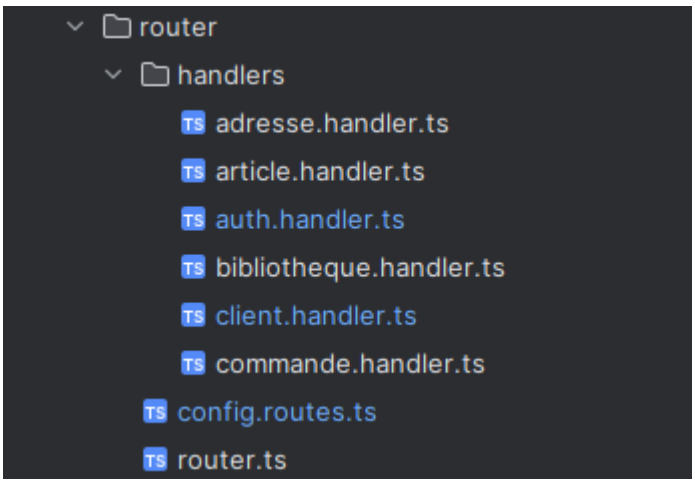
```
        return {'cli':cliResult,'cde': cdeResult};
```

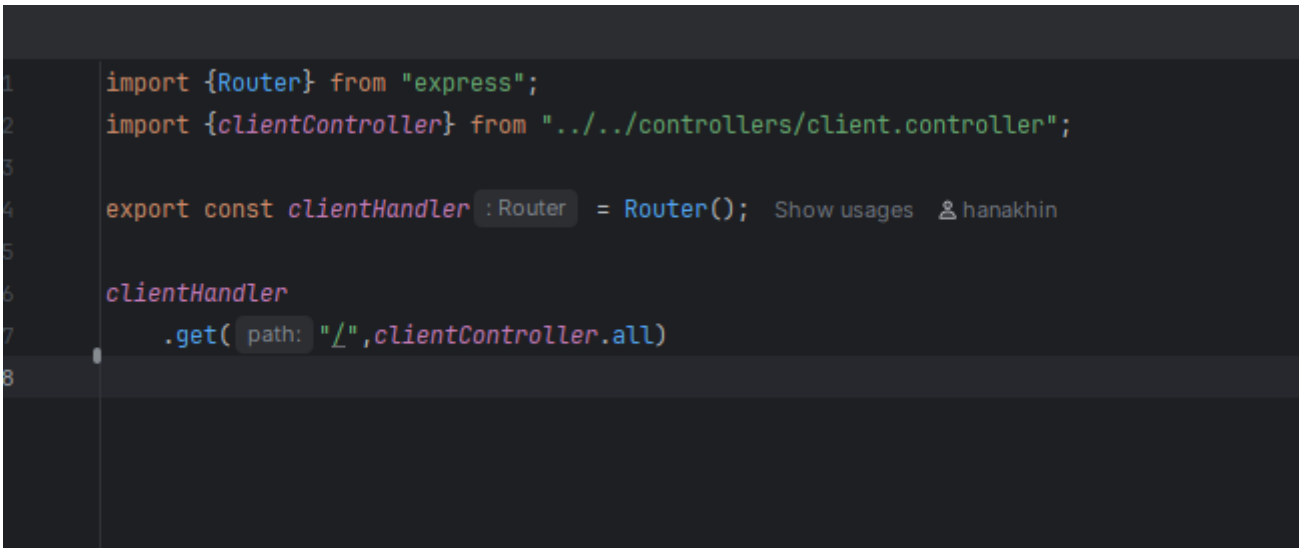
3)Créer la route

dans src/router/config.routes.ts on peut voir un tableau de routes, on duplique une des lignes et on change les valeurs par la route qu'on veut , url = [route de base], handler = [le router qui va gerer les routes]



ensuite dans router/handlers on crée un handler, [le nom qu'on veut].handler.ts





Dans ce handler, on crée notre export const [nom du handler] = Router() (crée une instance du router d'express.JS)

puis on appelle ce router fraichement crée pour lui definir ses endpoints , ici on voit .get , ca crée donc la route /api/clients/ , en method GET , qui executera le code du controller qu'on mettra en second parametre (ici clientController.all)

avec tout ca , on a crée notre route d'api /api/clients qui execute la methode all du controller .

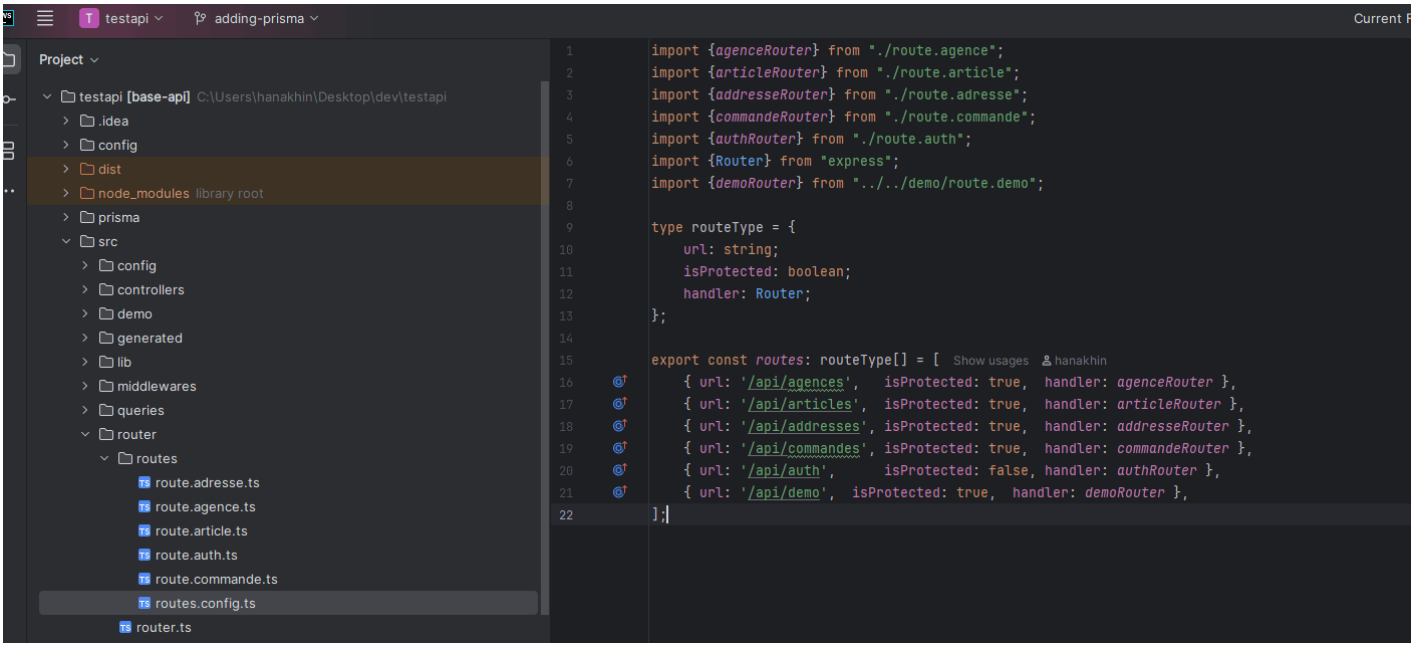
Pour tester on peut soit faire la route 'http://localhost:8000/api/clients' en get sur postman , ou simplement aller sur une page internet et mettre cette url dans la barre de recherche.

Créer une route

Pour creer une route il faut aller dans src/router/routes/routes.config.ts et y ajouter sa route dans le tableau de routes. Une route attends 3 paramètres:

L'url en string, Le Booleen "isProtected" qui définit si la route est accessible hors connexion ou non , et le router qui lui est associé.

Le paramétré "url" définit la route de base du router associé , donc par exemple si on met "/api/test" dans url et le router "testRouter" dans le handler; toutes les routes du router commenceront par "/api/test"



une fois que c'est fait , il faut aller créer le router associé , on va utiliser le "DemoRouter" pour cet exemple.

dans le fichier "src/router/routes" on crée un ficher ts qu'on appelle route.[le nom du router].ts

```ts
import {Router} from "express";
import {CommandeController} from "../../controllers/commande.controller";

export const commandeRouter :Router = Router(); Show usages hanakhin

commandeRouter
    .get( path: "/", CommandeController.getAllCommandesByCcli)
    .post( path: "/",CommandeController.createPcde)

```

ensuite on crée une instance de Router qui est importé depuis "express"

puis , on définit les routes du router, donc comme on a dit avant toutes les routes du demoRouter on par défaut le prefix "/api/demo", donc il nous suffit de lui mettre l'endpoint qu'on veut précédé de la méthode de la route.

dans notre cas ici on a une get et une post , suivi du controlleur, et de la méthode qu'on veut utiliser.

L'étape suivante est de créer le controlleur associé comme ci dessous.

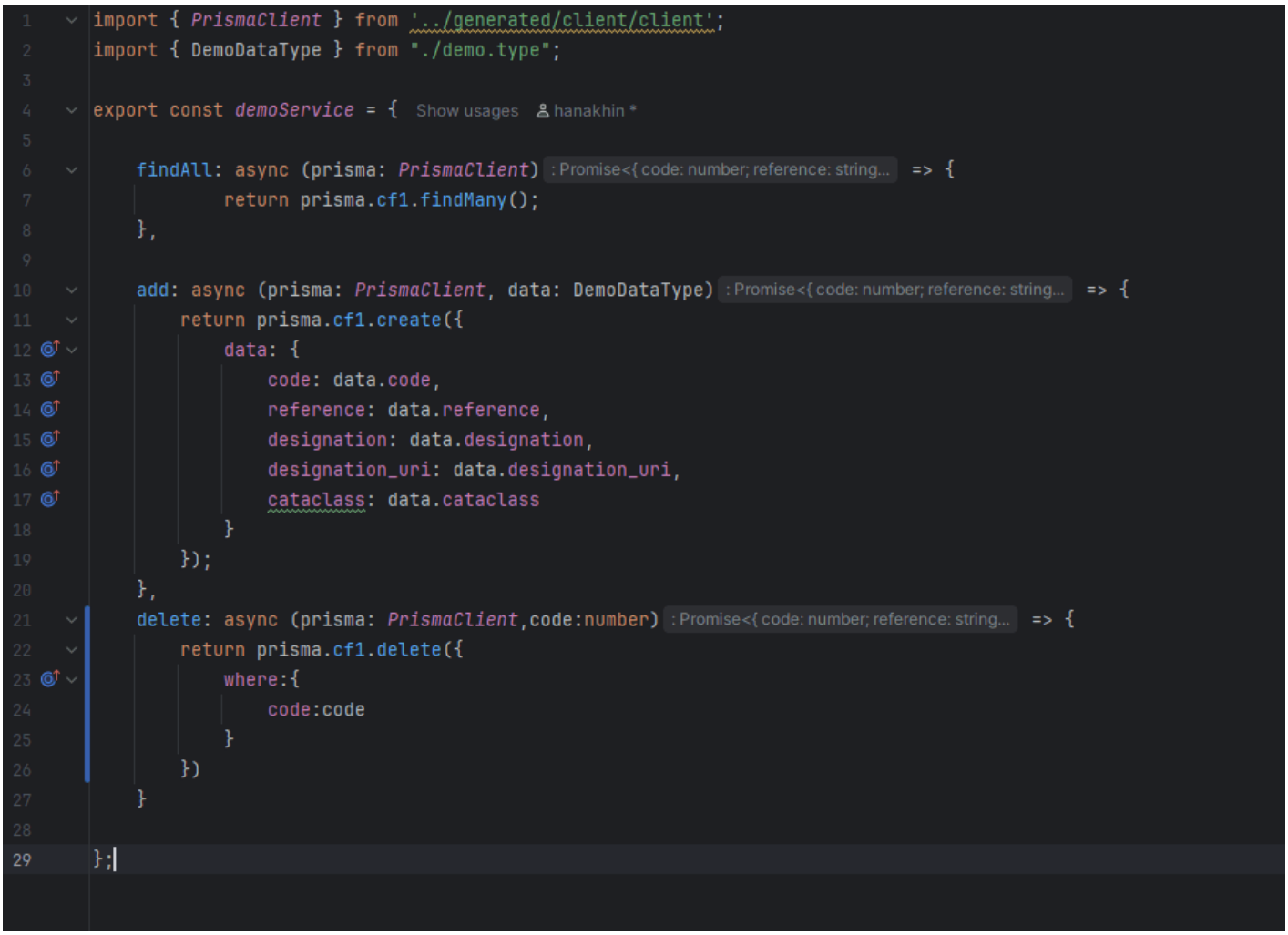
```ts
import { RequestHandler } from "express";
import { demoService } from "./demo.service";

export const DemoController :Record<string, RequestHandler>= { Show usages hanakhin *
    getAllData: async (req: any, res: any) :Promise<any> => {
        try {
            const prisma :any = req.prisma;
            const data :{ code: number; reference: string | null;... = await demoService.findAll(prisma);
            return res.status(200).json(data);
        } catch (err: any) {
            console.error( message: "Error fetching items:", err);
            return res.status(500).json({ error: err.message });
        }
    },

    addData: async (req: any, res: any) :Promise<any> => {...},
    deleteData: async (req: any, res: any) :Promise<any> => {...}
};
```

le controlleur contient la méthode qui permet d'aller chercher le service qui communique avec la base de donnée , lui il se charge de faire l'appel du service et de retourner une réponse .

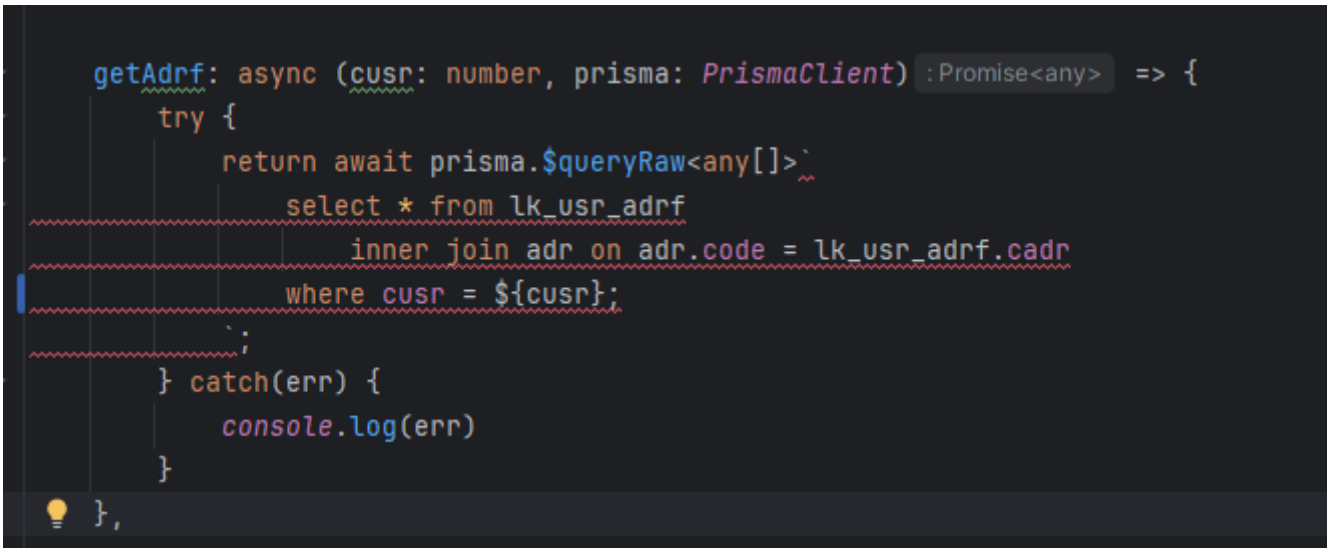
soit une 200 si la requête est acceptée , soit une 500 si il y a un souci, soit une 401 si pas autorisé ... etc , ici on gère uniquement les services.

on crée ce controlleur dans "src/controller" et on l'importe dans le router.

La dernière étape est de créer le service.



Le service ce charge des appel en base de donnée , ici on utilise prisma , mais il est possible de faire du sql ou autres directement .



on crée ce fichier dans "src/services"

pour faire une méthode , il faut suivre une syntaxe claire :

[nom de la méthode] : async ([les paramètres]) =>{[La méthode]}

Donc une fois que tout ceci est fait , la route devrais être accessible .

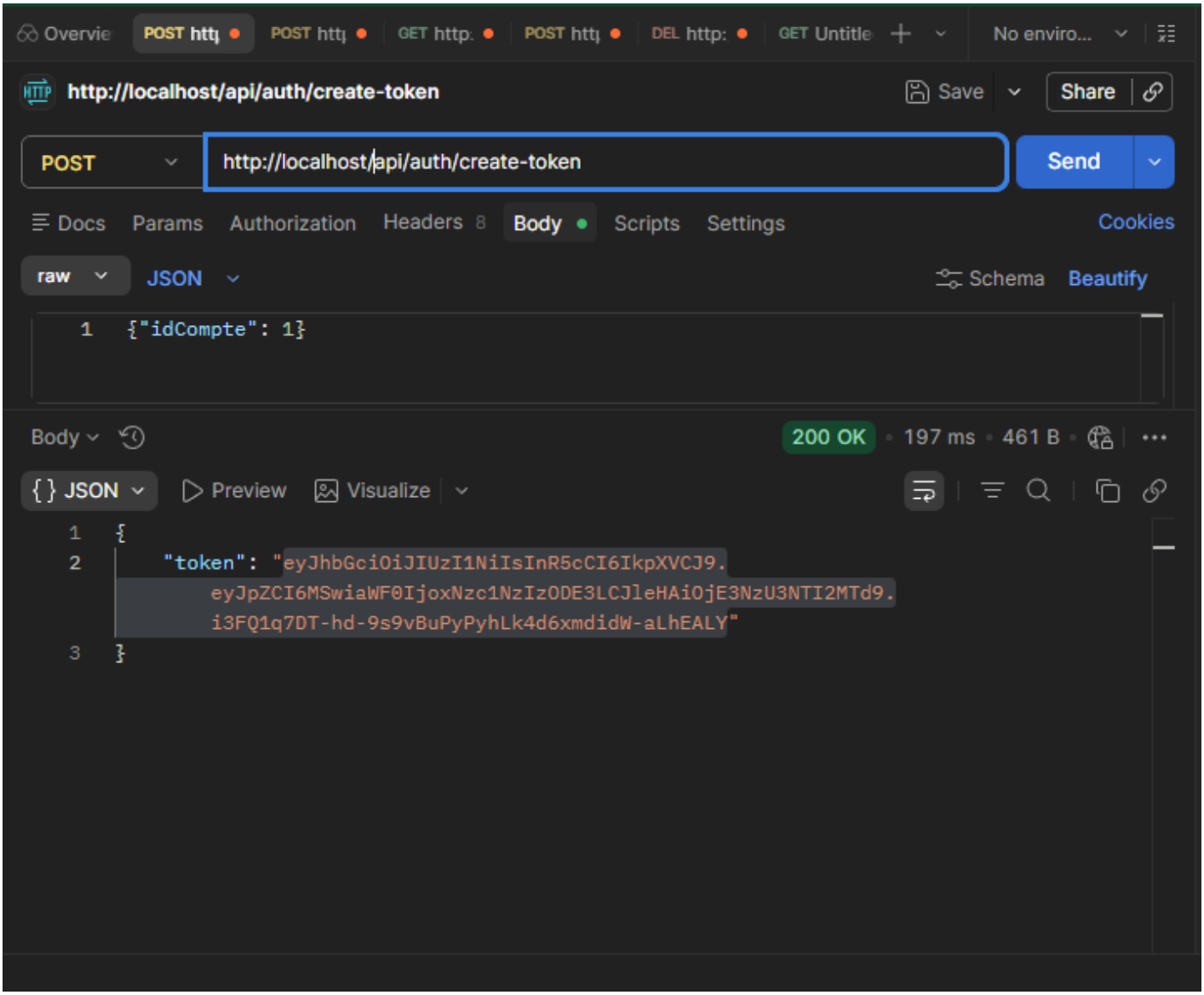
Le process est le suivant :

Route->Router->Controlleur->Service->Appel db

Tester sa route avec postman

Pour tester la route sur postman il y a 2 étapes a faire :

1:

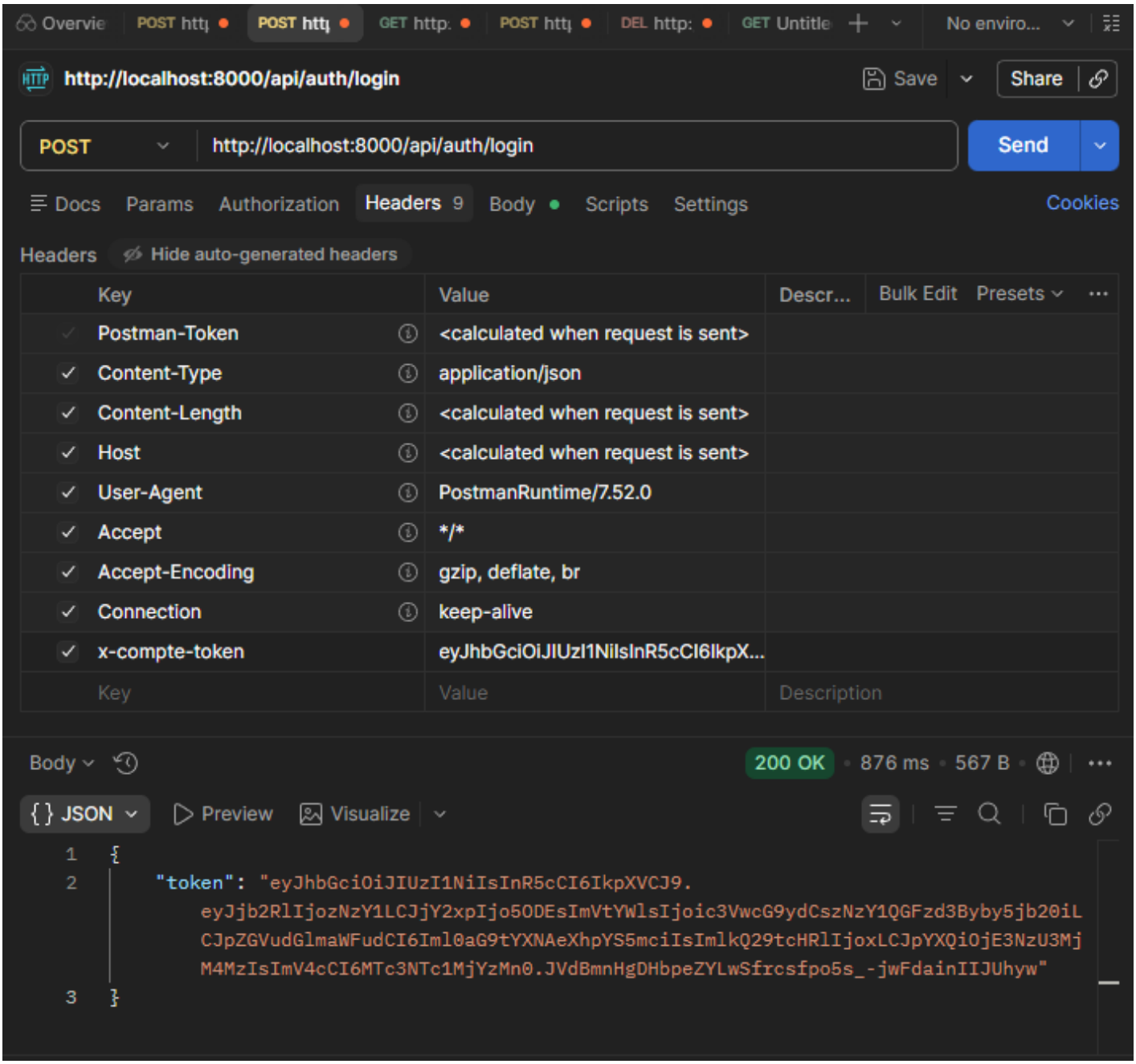


il faut generer le compte-token en faisant un appel sur cette route

"http://localhost:8000/api/auth/create-token"*

copier le token qui sera généré.

2:

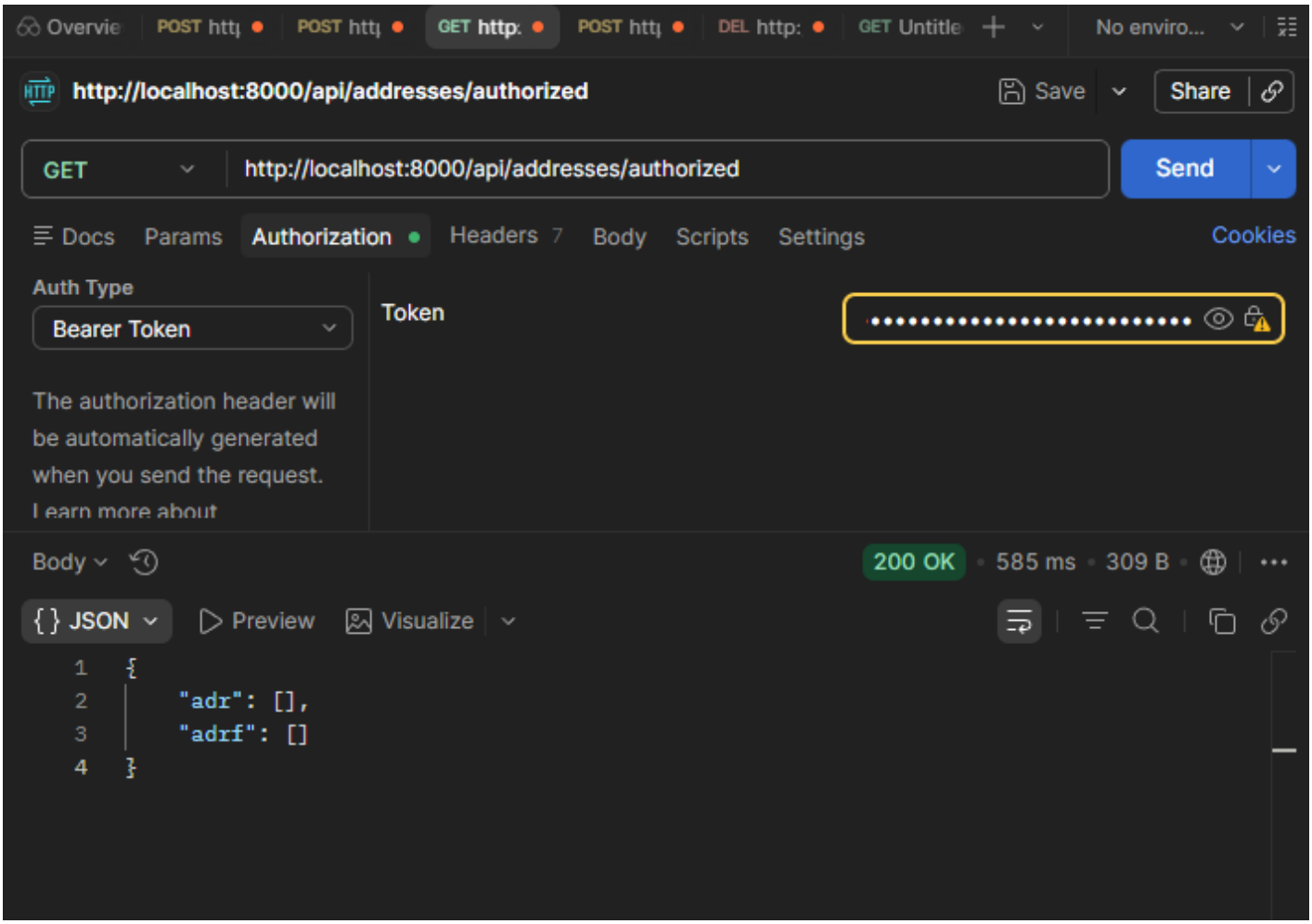


Aller sur la route "http://localhost:8000/api/auth/login"

passer les identifiants d'un utilisateur sur la db , en metant le token copié precedement dans un header "x-compte-token" : "token copié"

ca generera un autre token qu'il faudra copier a son tour et coller dans le champs

"bearer token" de l'auth type postman.



cette manipulation est a faire une fois , puis il faut simplement copié le token utilisateur généré au login dans chaque requete sur une route protégéé comme on a vu précédemment.