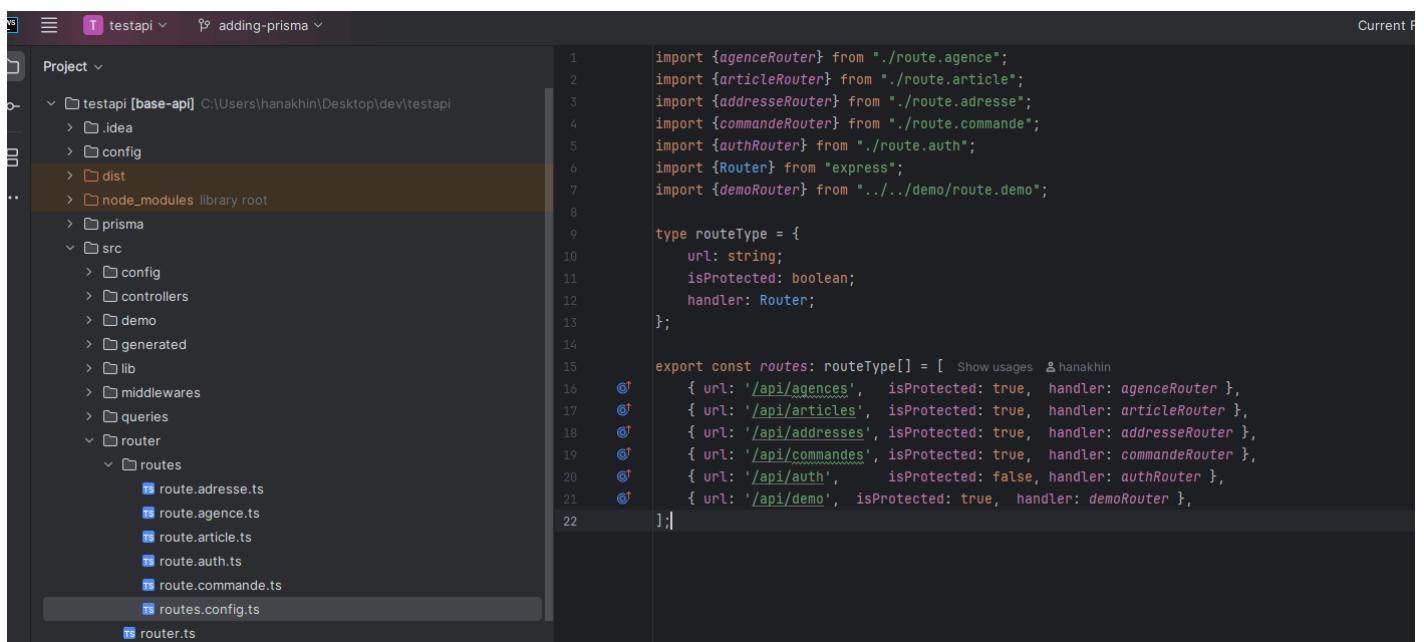


Créer une route

Pour créer une route il faut aller dans `src/router/routes/routes.config.ts` et y ajouter sa route dans le tableau de routes. Une route attend 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 fichier ts qu'on appelle route.[le nom du router].ts

```

1 import {Router} from "express";
2 import {CommandeController} from "../../controllers/commande.controller";
3
4 export const commandeRouter :Router = Router(); Show usages  ⓘ hanakhin
5
6 commandeRouter
7   .get( path: "/", CommandeController.getAllCommandesByCli)
8   .post( path: "/", CommandeController.createPcde)
9

```

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.

```

1  import { RequestHandler } from "express";
2  import { demoService } from "../demo.service";
3
4  export const DemoController :Record<string, RequestHandler>= { Show usages  ⓘ hanakhin *
5    getAllData: async (req: any, res: any) : Promise<any> => {
6      try {
7        const prisma :any = req.prisma;
8        const data :{code: number; reference: string | null;...} = await demoService.findAll(prisma);
9        return res.status(200).json(data);
10     } catch (err: any) {
11       console.error( message: "Error fetching items:", err);
12       return res.status(500).json({ error: err.message });
13     }
14   },
15
16   addData: async (req: any, res: any) : Promise<any> => {...},
17   deleteData: async (req: any, res: any) : Promise<any> => {...}
18 };
19

```

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.

```
1  import { PrismaClient } from '../generated/client/client';
2  import { DemoDataType } from '../demo.type';
3
4  export const demoService = { Show usages  hanakhin *
5
6    findAll: async (prisma: PrismaClient) : Promise<{ code: number; reference: string... => {
7      return prisma.cf1.findMany();
8    },
9
10   add: async (prisma: PrismaClient, data: DemoDataType) : Promise<{ code: number; reference: string... => {
11     return prisma.cf1.create({
12       data: {
13         code: data.code,
14         reference: data.reference,
15         designation: data.designation,
16         designation_uri: data.designation_uri,
17         cataclass: data.cataclass
18       }
19     });
20   },
21   delete: async (prisma: PrismaClient, code: number) : Promise<{ code: number; reference: string... => {
22     return prisma.cf1.delete({
23       where: {
24         code: code
25       }
26     })
27   }
28
29 };|
```

Le service se charge des appels en base de données, ici on utilise prisma, mais il est possible de faire du sql ou autres directement.

```

getAdrf: async (cusr: number, prisma: PrismaClient) : Promise<any> => {
  try {
    return await prisma.$queryRaw<any[]>`
      select * from lk_usr_adrf
      inner join adr on adr.code = lk_usr_adrf.cadr
      where cusr = ${cusr};
    `;
  } catch(err) {
    console.log(err)
  }
},

```

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

Revision #1

Created 2026-04-10 15:05:04 UTC by Hanakhin

Updated 2026-04-10 15:06:08 UTC by Hanakhin