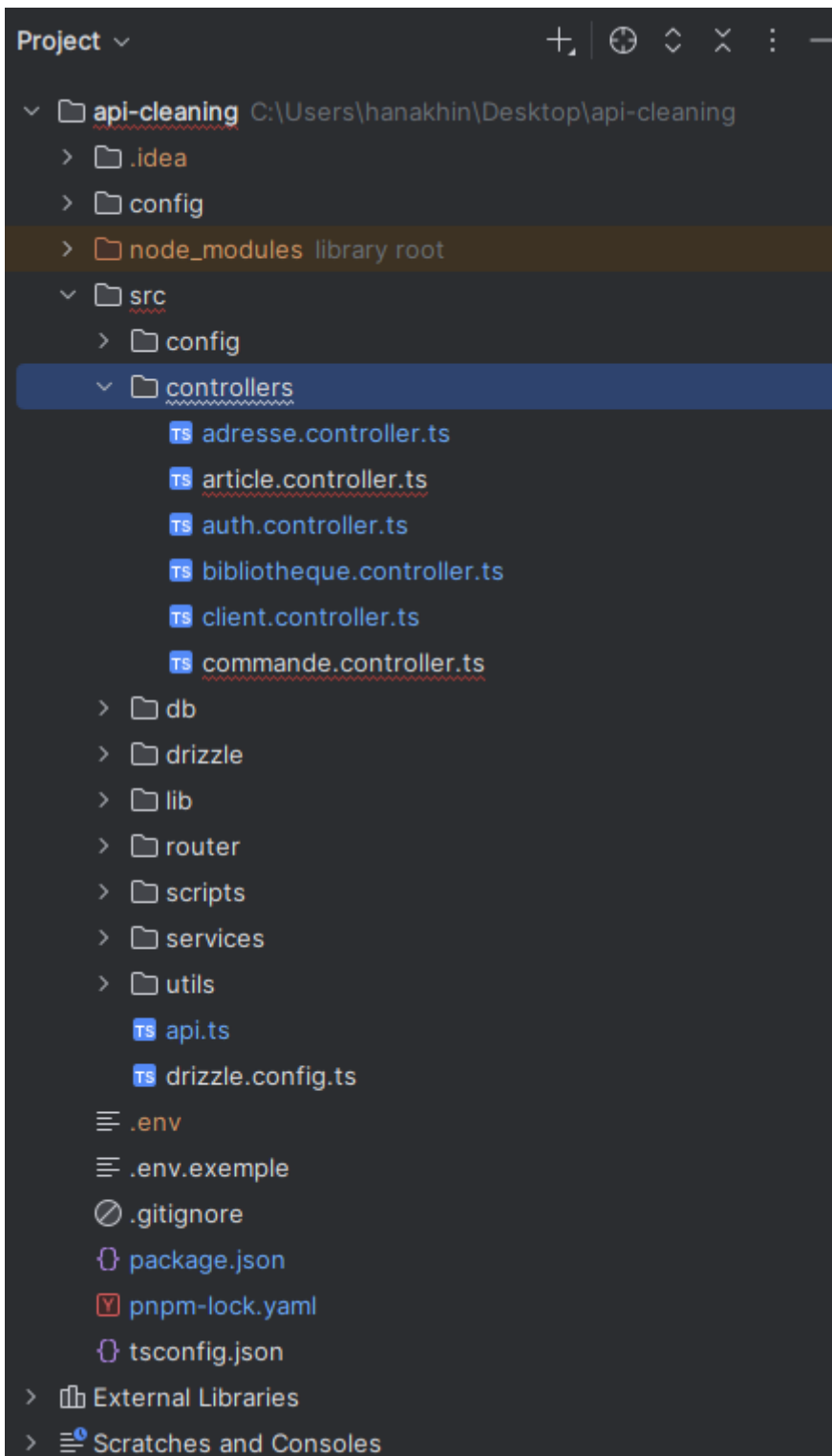


E-CARE API CLEANING

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



1)Créer le controller

```

1  import {clientService} from "../services/client.service";
2  import {getDb} from "../lib/mssql";
3  import {sendError, sendSuccess} from "../utils/helpers/response.helper";
4
5  export const clientController = { Show usages  & hanakhin *
6
7      all: async (req:any,res:any) : Promise<Response<any, Record<string, any>...> => {
8          try {
9              const pool : ConnectionPool = getDb(Number( value: 1))
10             const result : { cli: IResult<any>; cde: IResult<any>} = await clientService.getAll(pool)
11             return sendSuccess(res, result, status: 200)
12         } catch (err) {
13             console.log(err)
14             return sendError(res, err, status: 500)
15         }
16     },
17 }

```

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 différentes qui font la même chose , question de préférence.

donc dans le controller on appelle le service clientService.[la méthode qu'on veut] ([le paramètre que la méthode attend]).

```
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

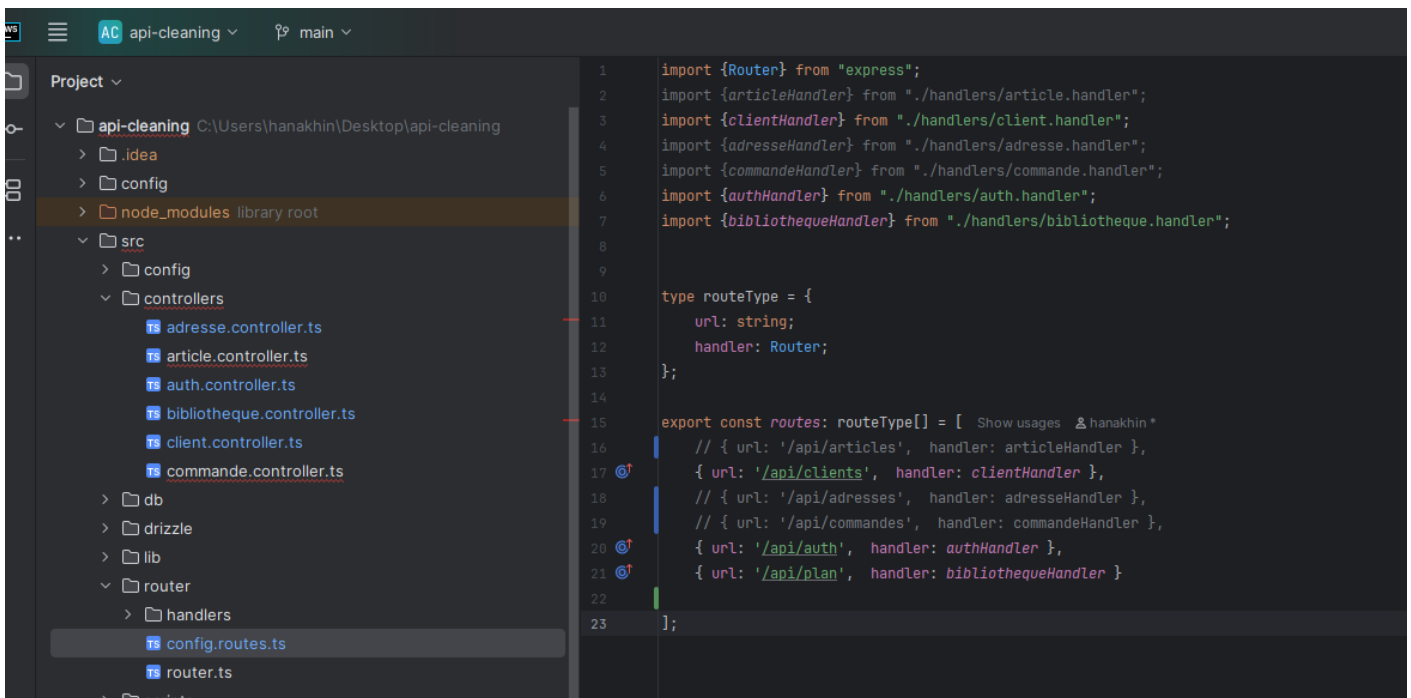
```

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

```

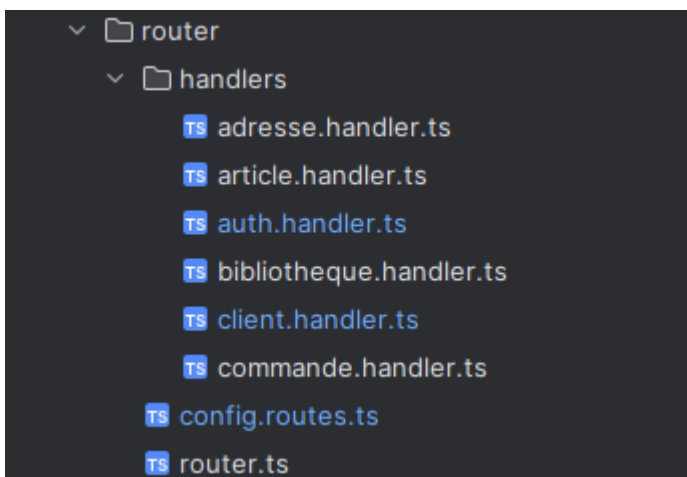
3)Créer la route

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]



```
1 import {Router} from "express";
2 import {articleHandler} from "../handlers/article.handler";
3 import {clientHandler} from "../handlers/client.handler";
4 import {adresseHandler} from "../handlers/adresse.handler";
5 import {commandeHandler} from "../handlers/commande.handler";
6 import {authHandler} from "../handlers/auth.handler";
7 import {bibliothequeHandler} from "../handlers/bibliotheque.handler";
8
9
10 type routeType = {
11   url: string;
12   handler: Router;
13 };
14
15 export const routes: routeType[] = [ Show usages & hanakhin "
16 // { url: '/api/articles', handler: articleHandler },
17 { url: '/api/clients', handler: clientHandler },
18 // { url: '/api/adresses', handler: adresseHandler },
19 // { url: '/api/commandes', handler: commandeHandler },
20 { url: '/api/auth', handler: authHandler },
21 { url: '/api/plan', handler: bibliothequeHandler }
22 ];
23 ];
```

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



- router
 - handlers
 - adresse.handler.ts
 - article.handler.ts
 - auth.handler.ts
 - bibliotheque.handler.ts
 - client.handler.ts
 - commande.handler.ts
 - config.routes.ts
 - router.ts

```
1 import {Router} from "express";
2 import {clientController} from "../../controllers/client.controller";
3
4 export const clientHandler : Router = Router(); Show usages & hanakhin
5
6 clientHandler
7   .get( path: "/", clientController.all)
8
```

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

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

avec tout ça , on a créé notre route d'api /api/clients qui exécute la méthode 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.

Revision #4

Created 2026-04-23 13:22:35 UTC by Hanakhin

Updated 2026-04-23 13:52:19 UTC by Hanakhin