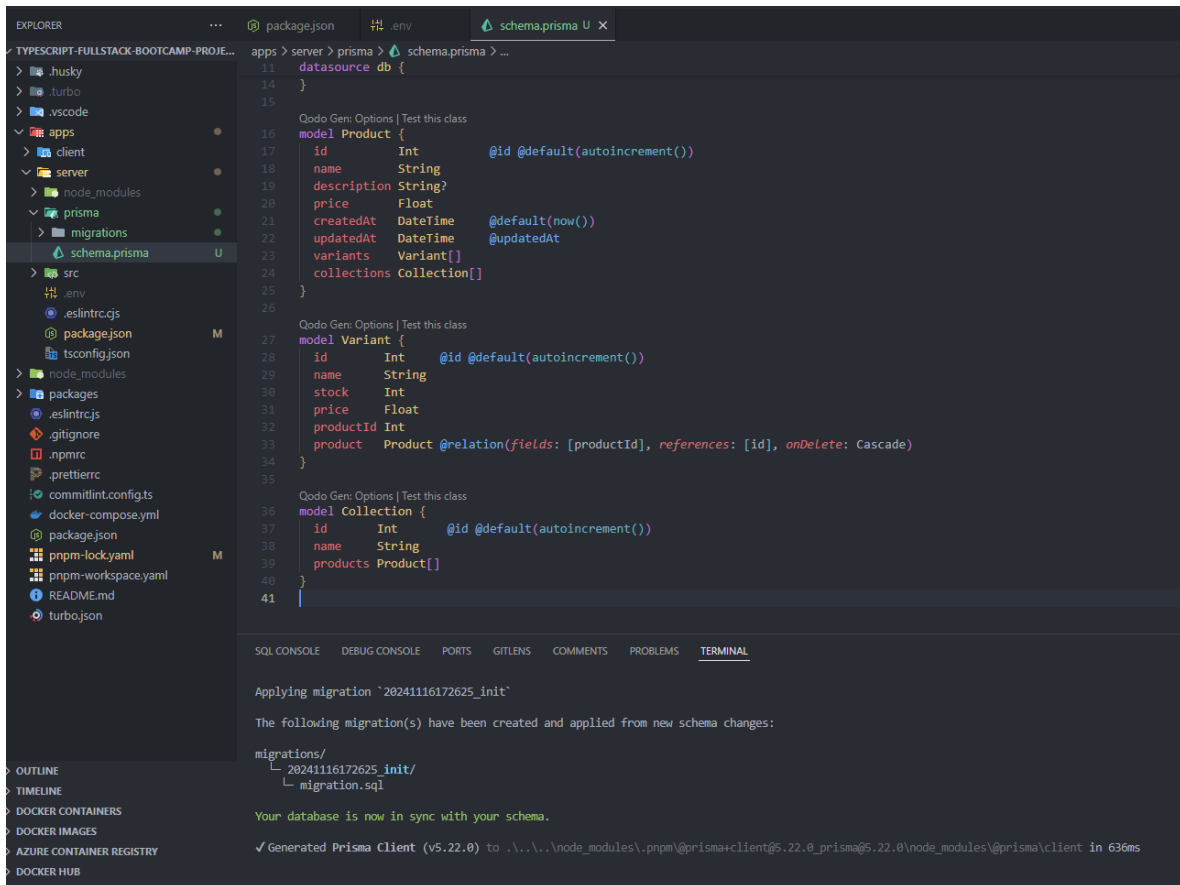


Set Up prisma



The screenshot shows a VS Code editor with the following components:

- EXPLORER:** Displays the project structure. The `schema.prisma` file is selected under the `prisma` folder.
- SCHEMA.PRISMA:** Contains the Prisma schema definition:

```
11 datasource db {
12   url      = env('DATABASE_URL')
13   provider = 'postgresql'
14 }
15
16 Qodo Gen: Options | Test this class
17 model Product {
18   id          Int           @id @default(autoincrement())
19   name        String
20   description  String?
21   price       Float
22   createdAt   DateTime      @default(now())
23   updatedAt   DateTime      @updatedAt
24   variants    Variant[]
25   collections Collection[]
26 }
27
28 Qodo Gen: Options | Test this class
29 model Variant {
30   id          Int           @id @default(autoincrement())
31   name        String
32   stock       Int
33   price       Float
34   productId   Int
35   product     Product @relation(fields: [productId], references: [id], onDelete: Cascade)
36 }
37
38 Qodo Gen: Options | Test this class
39 model Collection {
40   id          Int           @id @default(autoincrement())
41   name        String
42   products    Product[]
43 }
```
- TERMINAL:** Shows the application of a migration:

```
Applying migration '20241116172625_init'

The following migration(s) have been created and applied from new schema changes:

migrations/
├─ 20241116172625_init/
│   └─ migration.sql
└─

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.22.0) to .\..\..\node_modules\.pnpm\@prisma+client@5.22.0_prisma@5.22.0\node_modules\@prisma\client in 636ms
```

Repository

The screenshot shows a VS Code editor with three files open: `productTypes.ts`, `productRepository.ts`, and `productController.ts`. The `productTypes.ts` file defines interfaces for product creation and update, and a product interface. The `productRepository.ts` file implements the repository interface with methods for creating, getting, updating, and deleting products. The `productController.ts` file implements the controller interface with methods for creating, getting, updating, and deleting products. The terminal shows the command `prisma generate` and the output indicating that the Prisma client was generated successfully.

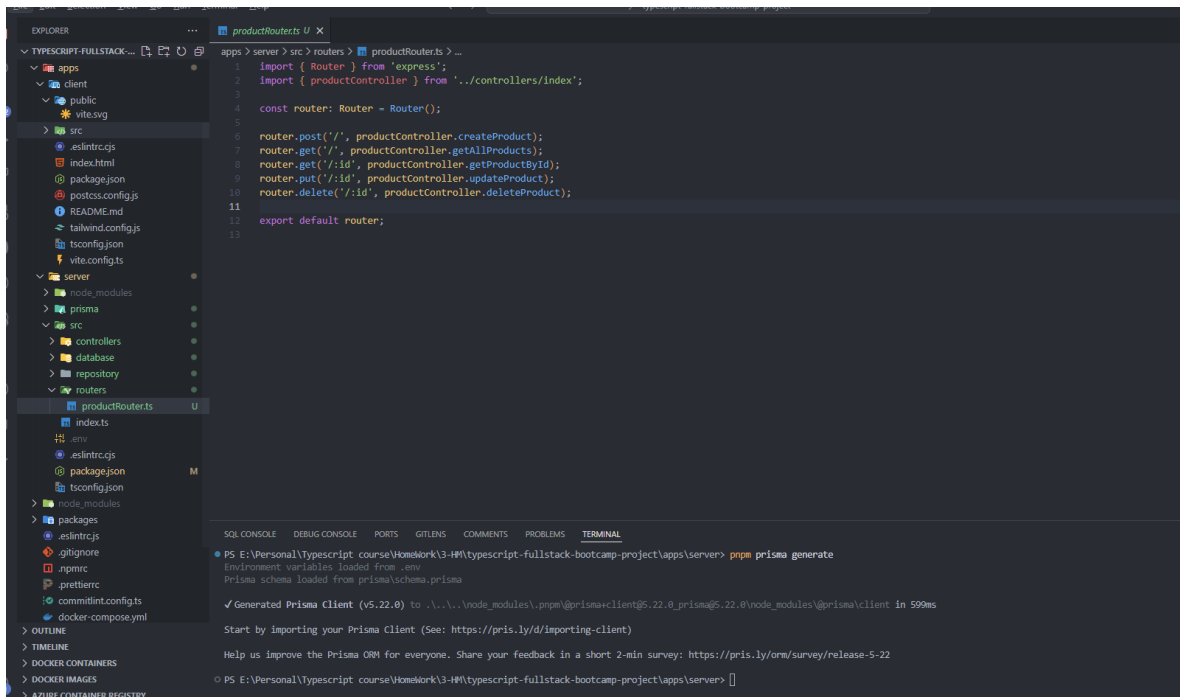
```
productTypes U X
1 export interface ProductCreateInput {
2   name?: string;
3   description?: string;
4   price: number;
5 }
6
7 export interface ProductUpdateInput {
8   name?: string;
9   description?: string;
10  price?: number;
11 }
12
13 export interface Product {
14   id: number;
15   name: string;
16   description?: string;
17   price: number;
18   createdAt: Date;
19   updatedAt: Date;
20 }
21
productRepository U X
1 import { Product } from '@prisma/client';
2 import { ProductCreateInput, ProductUpdateInput } from '../controllers/productCont';
3 import { prisma } from '../database/db-client';
4
5 export const productRepository = {
6   async createProduct(data: ProductCreateInput): Promise<Product> {
7     return await prisma.product.create({ data });
8   },
9
10  async getAllProducts(): Promise<Product[]> {
11    return await prisma.product.findMany();
12  },
13
14  async getProductById(id: number): Promise<Product | null> {
15    return await prisma.product.findUnique({ where: { id } });
16  },
17
18  async updateProduct(id: number, data: ProductUpdateInput): Promise<Product> {
19    return await prisma.product.update({ where: { id }, data });
20  },
21
22  async deleteProduct(id: number): Promise<Product> {
23    return await prisma.product.delete({ where: { id } });
24  },
25 };
26
productController U X
1 import { Request, Response } from 'express';
2 import { ProductCreateInput, ProductUpdateInput } from '../types';
3 import { productRepository } from '../repository';
4
5 export const productController = {
6   async createProduct(req: Request, res: Response) {
7     const { name, description, price } = ProductCreateInput;
8
9     try {
10       const product = await productRepository.createPro
11       res.status(201).json(product);
12     } catch (error) {
13       res.status(500).json({ error: 'Error creating pro
14     }
15   },
16
17   async getAllProducts(req: Request, res: Response) {
18     try {
19       const products = await productRepository.getAllPr
20       res.json(products);
21     } catch (error) {
22       res.status(500).json({ error: 'Error fetching pro
23     }
24   },
25
26   async getProductById(req: Request, res: Response) {
27     const id = Number(req.params.id);
28
29     try {
30       const product = await productRepository.getProduc
31       if (!product) {
32         return res.status(404).json({ error: 'Product
33       }
34     } catch (error) {
35       res.status(500).json({ error: 'Error fetching product
36     }
37   },
38
39   async updateProduct(req: Request, res: Response) {
40     const id = Number(req.params.id);
41     const { name, description, price } = ProductUpdateInput;
42
43     try {
44       const product = await productRepository.getProduct
45       if (!product) {
46         return res.status(404).json({ error: 'Product not found
47       }
48       const updatedProduct = await productRepository.updateProduct(id, {
49         name,
50         description,
51         price
52       });
53       res.status(200).json(updatedProduct);
54     } catch (error) {
55       res.status(500).json({ error: 'Error updating product
56     }
57   },
58
59   async deleteProduct(req: Request, res: Response) {
60     const id = Number(req.params.id);
61
62     try {
63       const product = await productRepository.getProduct
64       if (!product) {
65         return res.status(404).json({ error: 'Product not found
66       }
67       await productRepository.deleteProduct(id);
68       res.status(200).json({ message: 'Product deleted successfully
69     } catch (error) {
70       res.status(500).json({ error: 'Error deleting product
71     }
72   },
73 };
74
75 export { productController };
76
Router U X
1 import { Router } from 'express';
2 import { productController } from './productController';
3
4 const router = Router();
5
6 router.post('/products', productController.createProduct);
7 router.get('/products', productController.getAllProducts);
8 router.get('/products/:id', productController.getProductById);
9 router.put('/products/:id', productController.updateProduct);
10 router.delete('/products/:id', productController.deleteProduct);
11
12 export { router };
13
index U X
1 export * from './productController/types/productType';
2 export * from './productController/productController';
3
SQL CONSOLE DEBUG CONSOLE PORTS GITLens COMMENTS PROBLEMS TERMINAL
PS E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server> prisma generate
Prisma schema loaded from prisma/schema.prisma
✓ Generated Prisma Client (v5.22.0) to E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server\node_modules\@prisma\client in 599ms
Start by importing your Prisma Client (See: https://pris.ly/d/importing-client)
Help us improve the Prisma ORM for everyone. Share your feedback in a short 2-min survey: https://pris.ly/orm/survey/release-5-22
PS E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server>
```

Controller

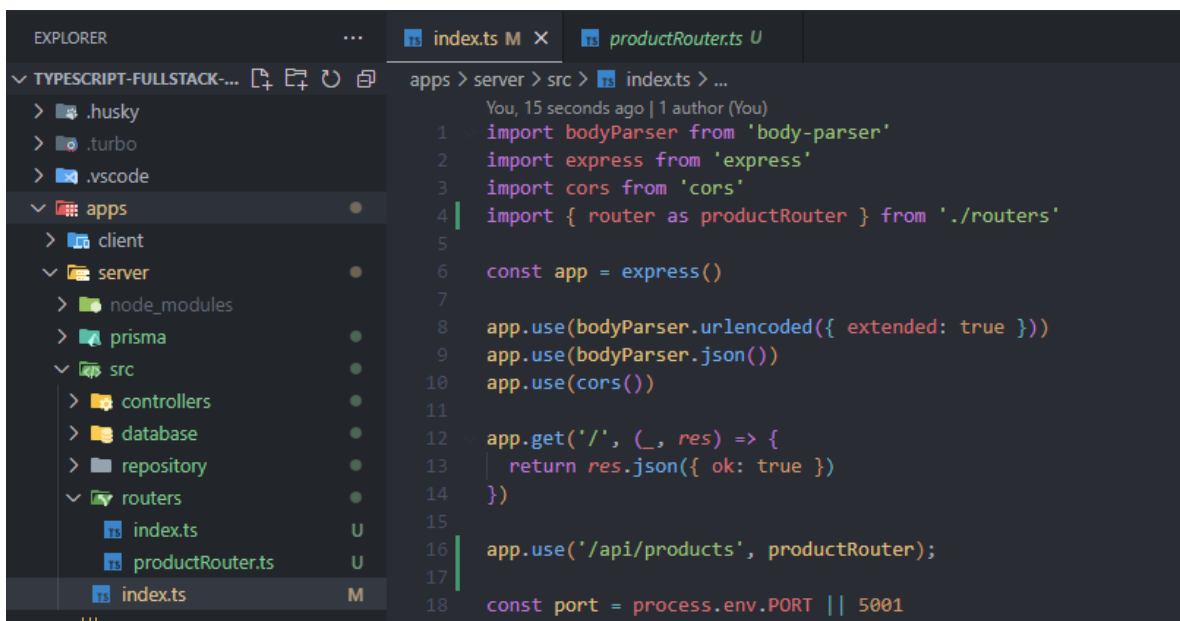
The screenshot shows a VS Code editor with three files open: `productTypes.ts`, `productRepository.ts`, and `productController.ts`. The `productTypes.ts` file defines interfaces for product creation and update, and a product interface. The `productRepository.ts` file implements the repository interface with methods for creating, getting, updating, and deleting products. The `productController.ts` file implements the controller interface with methods for creating, getting, updating, and deleting products. The terminal shows the command `prisma generate` and the output indicating that the Prisma client was generated successfully.

```
productTypes U X
1 export interface ProductCreateInput {
2   name: string;
3   description?: string;
4   price: number;
5 }
6
7 export interface ProductUpdateInput {
8   name?: string;
9   description?: string;
10  price?: number;
11 }
12
13 export interface Product {
14   id: number;
15   name: string;
16   description?: string;
17   price: number;
18   createdAt: Date;
19   updatedAt: Date;
20 }
21
productRepository U X
1 import { Product } from '@prisma/client';
2 import { ProductCreateInput, ProductUpdateInput } from '../controllers/productCont';
3 import { prisma } from '../database/db-client';
4
5 export const productRepository = {
6   async createProduct(data: ProductCreateInput): Promise<Product> {
7     return await prisma.product.create({ data });
8   },
9
10  async getAllProducts(): Promise<Product[]> {
11    return await prisma.product.findMany();
12  },
13
14  async getProductById(id: number): Promise<Product | null> {
15    return await prisma.product.findUnique({ where: { id } });
16  },
17
18  async updateProduct(id: number, data: ProductUpdateInput): Promise<Product> {
19    return await prisma.product.update({ where: { id }, data });
20  },
21
22  async deleteProduct(id: number): Promise<Product> {
23    return await prisma.product.delete({ where: { id } });
24  },
25 };
26
productController U X
1 import { Request, Response } from 'express';
2 import { ProductCreateInput, ProductUpdateInput } from '../types';
3 import { productRepository } from '../repository';
4
5 export const productController = {
6   async createProduct(req: Request, res: Response) {
7     const { name, description, price } = ProductCreateInput;
8
9     try {
10       const product = await productRepository.createPro
11       res.status(201).json(product);
12     } catch (error) {
13       res.status(500).json({ error: 'Error creating pro
14     }
15   },
16
17   async getAllProducts(req: Request, res: Response) {
18     try {
19       const products = await productRepository.getAllPr
20       res.json(products);
21     } catch (error) {
22       res.status(500).json({ error: 'Error fetching pro
23     }
24   },
25
26   async getProductById(req: Request, res: Response) {
27     const id = Number(req.params.id);
28
29     try {
30       const product = await productRepository.getProduc
31       if (!product) {
32         return res.status(404).json({ error: 'Product
33       }
34     } catch (error) {
35       res.status(500).json({ error: 'Error fetching product
36     }
37   },
38
39   async updateProduct(req: Request, res: Response) {
40     const id = Number(req.params.id);
41     const { name, description, price } = ProductUpdateInput;
42
43     try {
44       const product = await productRepository.getProduct
45       if (!product) {
46         return res.status(404).json({ error: 'Product not found
47       }
48       const updatedProduct = await productRepository.updateProduct(id, {
49         name,
50         description,
51         price
52       });
53       res.status(200).json(updatedProduct);
54     } catch (error) {
55       res.status(500).json({ error: 'Error updating product
56     }
57   },
58
59   async deleteProduct(req: Request, res: Response) {
60     const id = Number(req.params.id);
61
62     try {
63       const product = await productRepository.getProduct
64       if (!product) {
65         return res.status(404).json({ error: 'Product not found
66       }
67       await productRepository.deleteProduct(id);
68       res.status(200).json({ message: 'Product deleted successfully
69     } catch (error) {
70       res.status(500).json({ error: 'Error deleting product
71     }
72   },
73 };
74
75 export { productController };
76
Router U X
1 import { Router } from 'express';
2 import { productController } from './productController';
3
4 const router = Router();
5
6 router.post('/products', productController.createProduct);
7 router.get('/products', productController.getAllProducts);
8 router.get('/products/:id', productController.getProductById);
9 router.put('/products/:id', productController.updateProduct);
10 router.delete('/products/:id', productController.deleteProduct);
11
12 export { router };
13
index U X
1 export * from './productController/types/productType';
2 export * from './productController/productController';
3
SQL CONSOLE DEBUG CONSOLE PORTS GITLens COMMENTS PROBLEMS TERMINAL
PS E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server> prisma generate
Prisma schema loaded from prisma/schema.prisma
✓ Generated Prisma Client (v5.22.0) to E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server\node_modules\@prisma\client in 599ms
Start by importing your Prisma Client (See: https://pris.ly/d/importing-client)
Help us improve the Prisma ORM for everyone. Share your feedback in a short 2-min survey: https://pris.ly/orm/survey/release-5-22
PS E:\Personal\Typescript\course\workspace\3-11\typescript-fullstack-bootcamp-project\apps\server>
```

Router

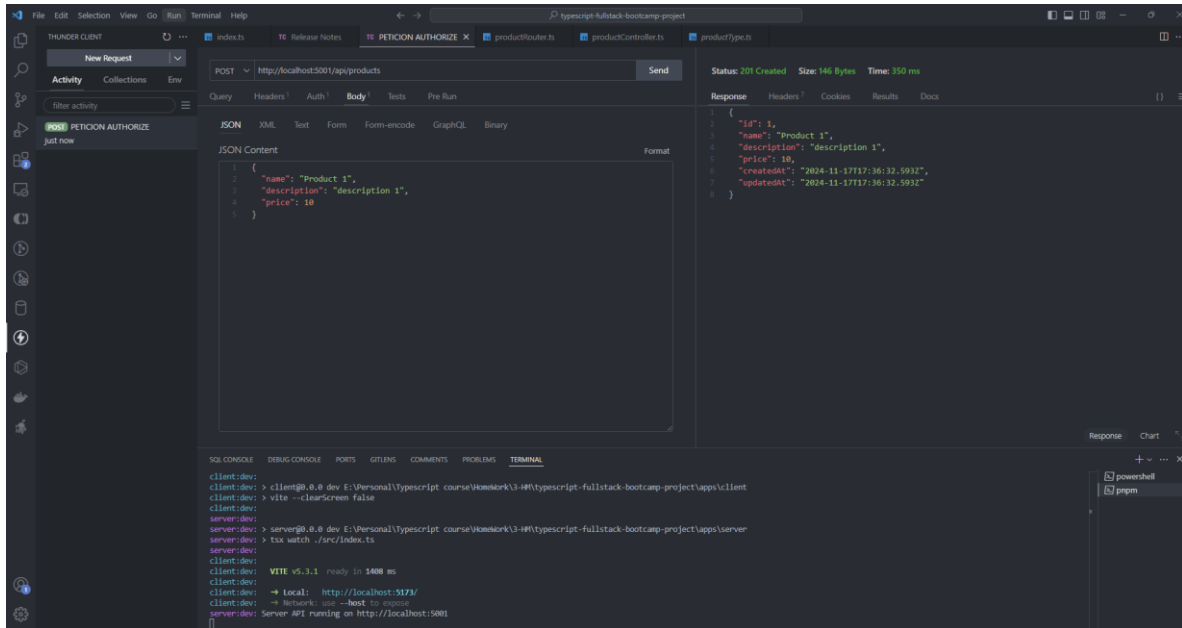


Set up apis

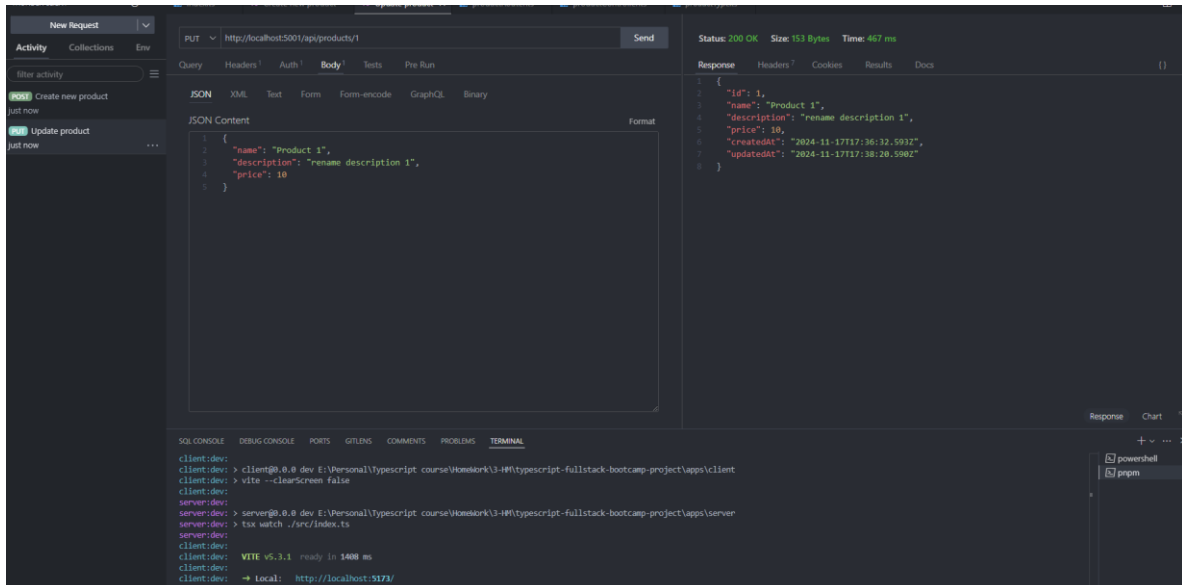


CRUD

CREATE EXAMPLE



UPDATE EXAMPLE



GET BY ID

Thunder Client interface showing a GET request to `http://localhost:5001/api/products/1`. The request is successful (Status: 200 OK, Size: 153 Bytes, Time: 447 ms). The response body is JSON:

```
{
  "id": 1,
  "name": "Product 1",
  "description": "rename description 1",
  "price": 10,
  "createdAt": "2024-11-17T17:36:32.593Z",
  "updatedAt": "2024-11-17T17:38:20.590Z"
}
```

The bottom terminal shows the following commands and output:

```
client:dev:
client:dev: > client@0.8.0 dev E:\Personal\Typescript course\Homework3-4\@typescript-fullstack-bootcamp-project\apps\client
client:dev: > vite --clearScreen false
client:dev:
server:dev:
server:dev: > server@0.8.0 dev E:\Personal\Typescript course\Homework3-4\@typescript-fullstack-bootcamp-project\apps\server
server:dev: > tsx watch ./src/index.ts
server:dev:
client:dev:
client:dev: VITE v5.3.1 ready in 1408 ms
client:dev:
client:dev: ➔ Local: http://localhost:5173/
client:dev: ➔ Network: use --host to expose
server:dev: Server API running on http://localhost:5001
[]
```

GET ALL

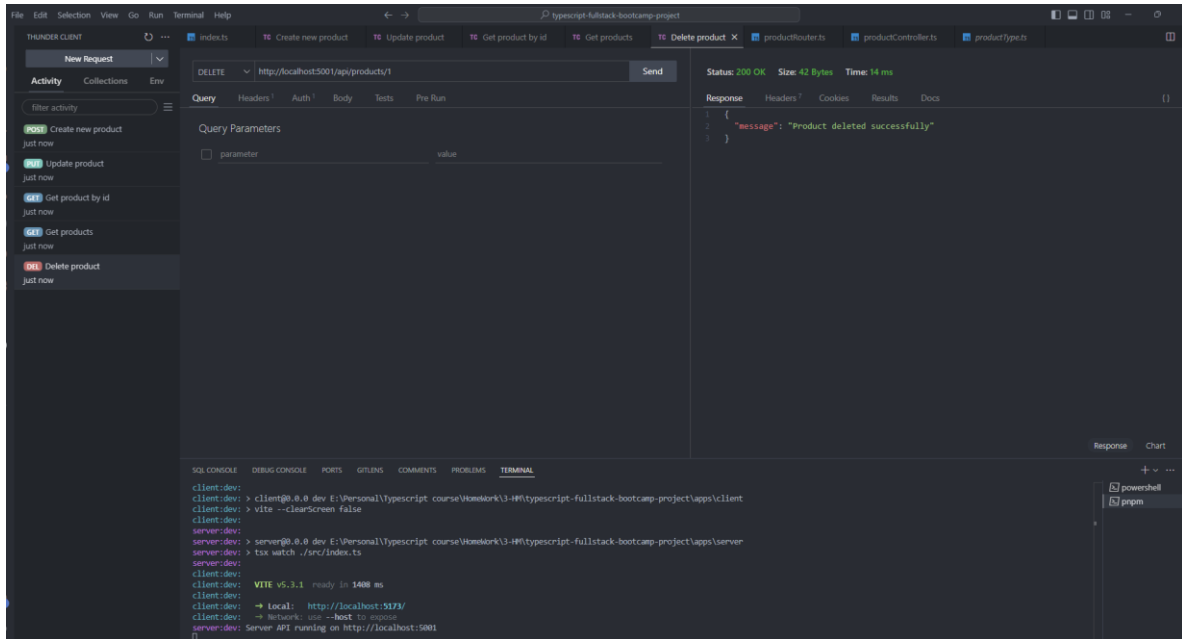
Thunder Client interface showing a GET request to `http://localhost:5001/api/products`. The request is successful (Status: 200 OK, Size: 155 Bytes, Time: 6 ms). The response body is JSON:

```
[
  {
    "id": 1,
    "name": "Product 1",
    "description": "rename description 1",
    "price": 10,
    "createdAt": "2024-11-17T17:36:32.593Z",
    "updatedAt": "2024-11-17T17:38:20.590Z"
  }
]
```

The bottom terminal shows the following commands and output:

```
client:dev:
client:dev: > client@0.8.0 dev E:\Personal\Typescript course\Homework3-4\@typescript-fullstack-bootcamp-project\apps\client
client:dev: > vite --clearScreen false
client:dev:
server:dev:
server:dev: > server@0.8.0 dev E:\Personal\Typescript course\Homework3-4\@typescript-fullstack-bootcamp-project\apps\server
server:dev: > tsx watch ./src/index.ts
server:dev:
client:dev:
client:dev: VITE v5.3.1 ready in 1408 ms
client:dev:
client:dev: ➔ Local: http://localhost:5173/
client:dev: ➔ Network: use --host to expose
server:dev: Server API running on http://localhost:5001
[]
```

DELETE



GET ALL RESULT AFTER DELETE

