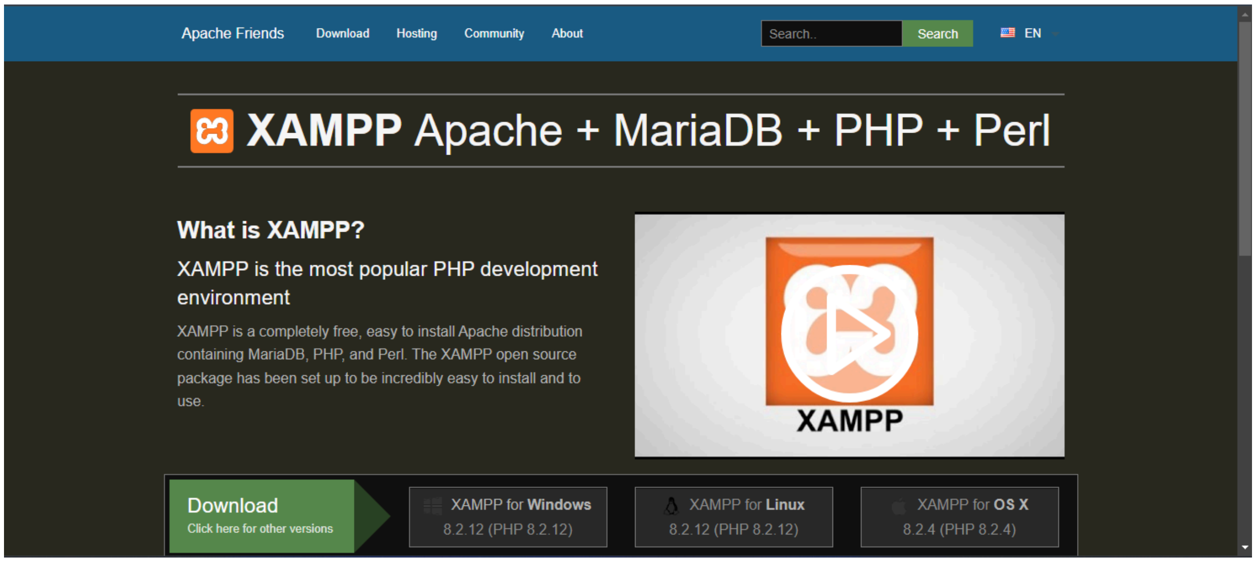
Mathew James M. Dagle IT - INTPROG32 BSIT - 3

Build a REST API with Typescript, NodeJS, ExpressJS and MySQL as storage.

Step 1: Install XAMPP to use MySQL.


Apache Friends
Download
Hosting
Community
About
Search
EN

XAMPP Apache + MariaDB + PHP + Perl

What is XAMPP?

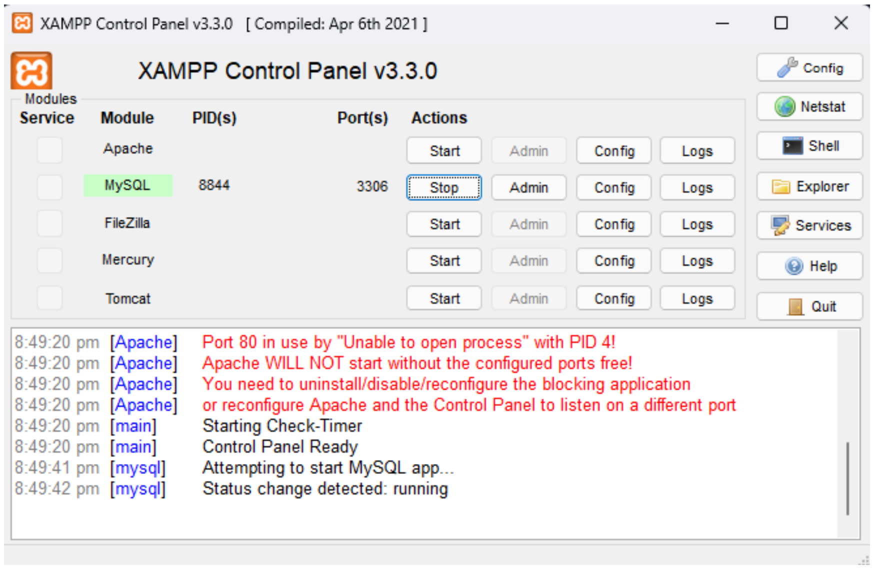
XAMPP is the most popular PHP development environment

XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.

XAMPP
Download
Click here for other versions
XAMPP for Windows 8.2.12 (PHP 8.2.12)
XAMPP for Linux 8.2.12 (PHP 8.2.12)
XAMPP for OS X 8.2.4 (PHP 8.2.4)

XAMPP
Download link/portal: https://www.apachefriends.org/

Step 2: Getting Started with MySQL on XAMPP


XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

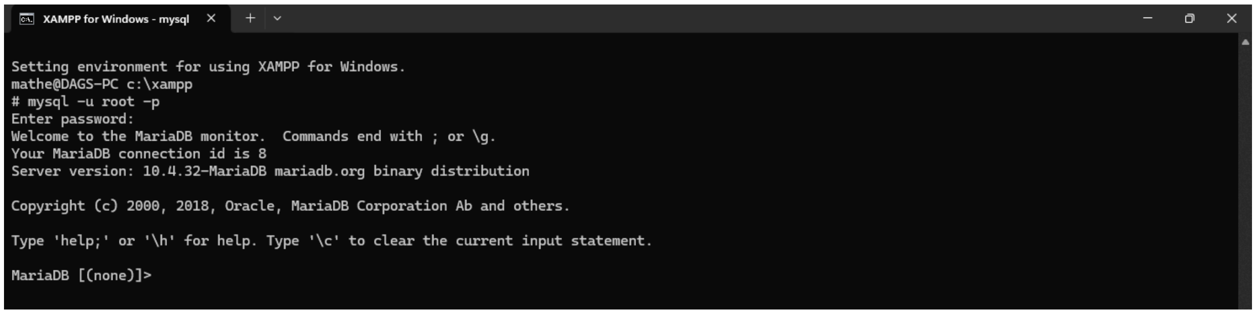
XAMPP Control Panel v3.3.0

Service	Module	PID(s)	Port(s)	Actions			
	Apache			Start	Admin	Config	Logs
	MySQL	8844	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
8:49:20 pm  [Apache]   Port 80 in use by "Unable to open process" with PID 4!
8:49:20 pm  [Apache]   Apache WILL NOT start without the configured ports free!
8:49:20 pm  [Apache]   You need to uninstall/disable/reconfigure the blocking application
8:49:20 pm  [Apache]   or reconfigure Apache and the Control Panel to listen on a different port
8:49:20 pm  [main]     Starting Check-Timer
8:49:20 pm  [main]     Control Panel Ready
8:49:41 pm  [mysql]    Attempting to start MySQL app...
8:49:42 pm  [mysql]    Status change detected: running
```

XAMPP Control Panel
When you run the app different modules from the apache friends will be seen, this time we will use MySQL. Get started by clicking the start button right beside MySQL. Then afterwards click shell on the rightmost navigation bar.

You will be greeted with the message: “Setting environment for using XAMPP for Windows” upon clicking the shell/terminal on the XAMPP control panel.

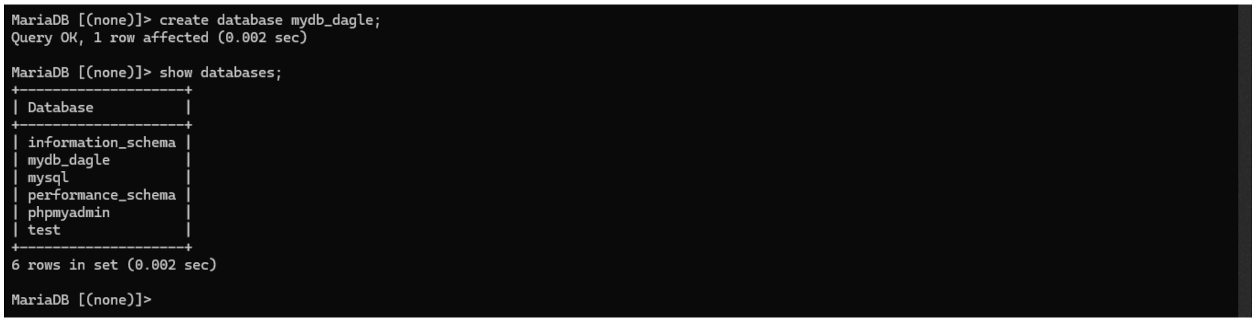


XAMPP for Windows - mysql

```
Setting environment for using XAMPP for Windows.
mathe@DAGS-PC c:\xampp
# mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

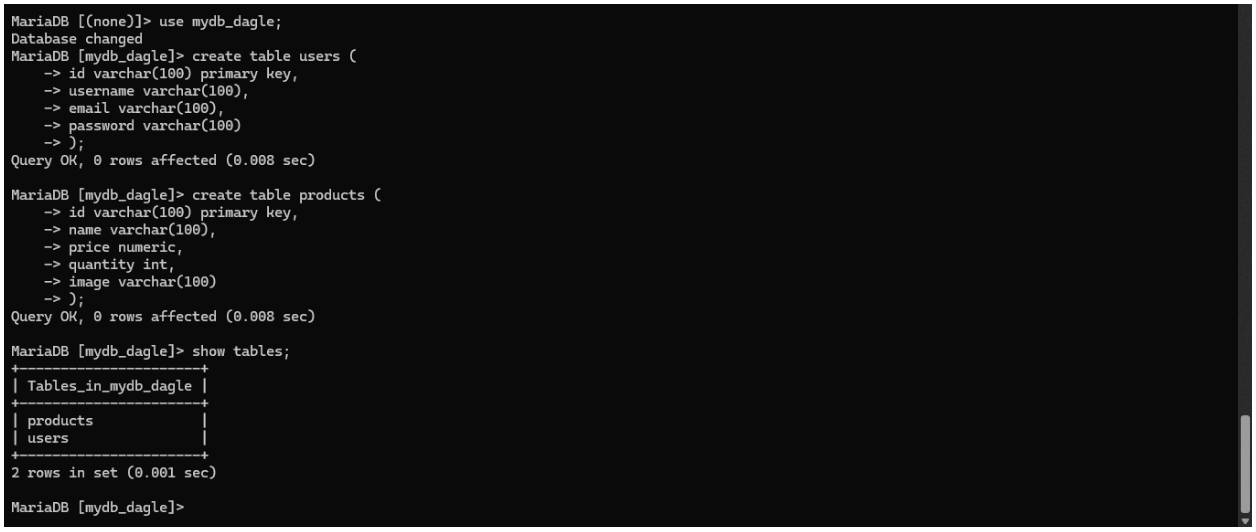
To continue type the following command:

- mysql -u root -p



```
MariaDB [(none)]> create database mydb_dagle;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb_dagle         |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.002 sec)

MariaDB [(none)]>
```

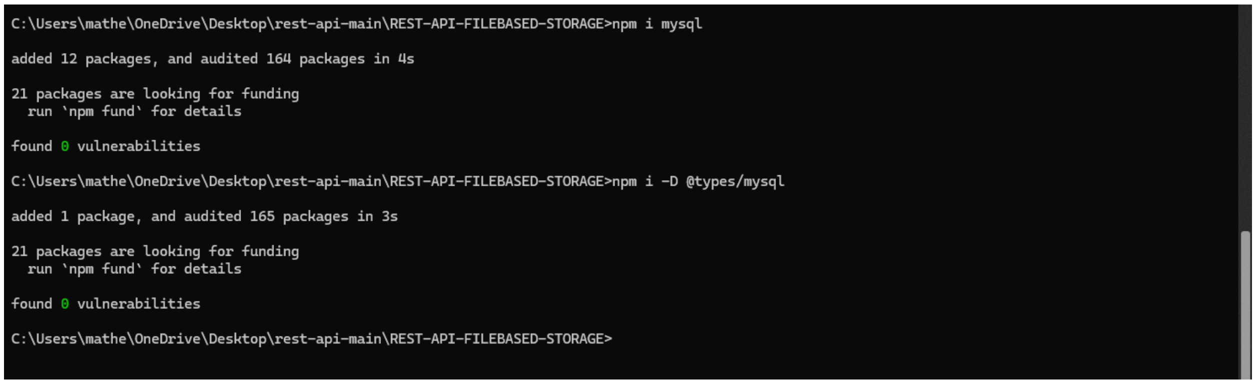
To create a database use the command “create database” followed by the database name. And check on the created database using show databases;



```
MariaDB [(none)]> use mydb_dagle;
Database changed
MariaDB [mydb_dagle]> create table users (
    -> id varchar(100) primary key,
    -> username varchar(100),
    -> email varchar(100),
    -> password varchar(100)
    -> );
Query OK, 0 rows affected (0.008 sec)

MariaDB [mydb_dagle]> create table products (
    -> id varchar(100) primary key,
    -> name varchar(100),
    -> price numeric,
    -> quantity int,
    -> image varchar(100)
    -> );
Query OK, 0 rows affected (0.008 sec)

MariaDB [mydb_dagle]> show tables;
+----------------------+
| Tables_in_mydb_dagle |
+----------------------+
| products             |
| users                |
+----------------------+
2 rows in set (0.001 sec)

MariaDB [mydb_dagle]>
```

Navigate to your database by typing ”use [database_name];” and from there you can create tables with corresponding columns. Type “show tables;” to see the tables created

Step 3: Installing Project Dependency



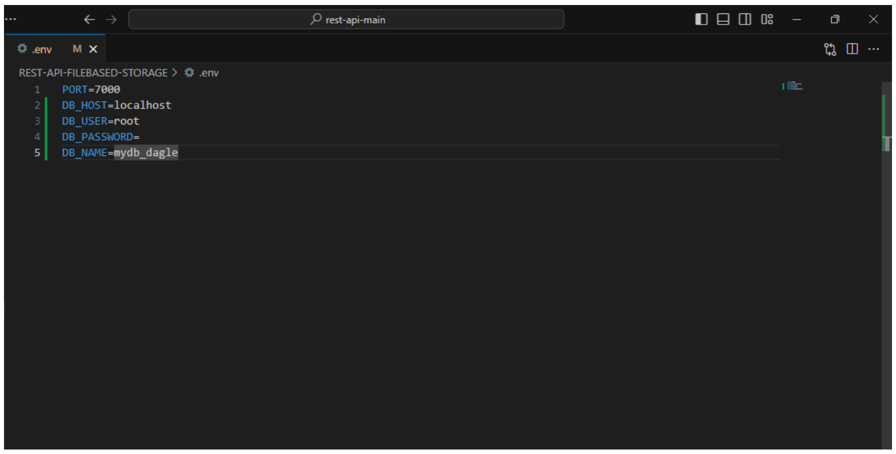
```
C:\Users\mathe\OneDrive\Desktop\rest-api-main\REST-API-FILEBASED-STORAGE>npm i mysql

added 12 packages, and audited 164 packages in 4s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\mathe\OneDrive\Desktop\rest-api-main\REST-API-FILEBASED-STORAGE>npm i -D @types/mysql

added 1 package, and audited 165 packages in 3s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\mathe\OneDrive\Desktop\rest-api-main\REST-API-FILEBASED-STORAGE>
```

Your Node.js project requires a dependency to be able to interact with MySQL. On your terminal, install it like so:

- npm install mysql

To use TypeScript effectively, you need to install the type definition for the package you installed previously:

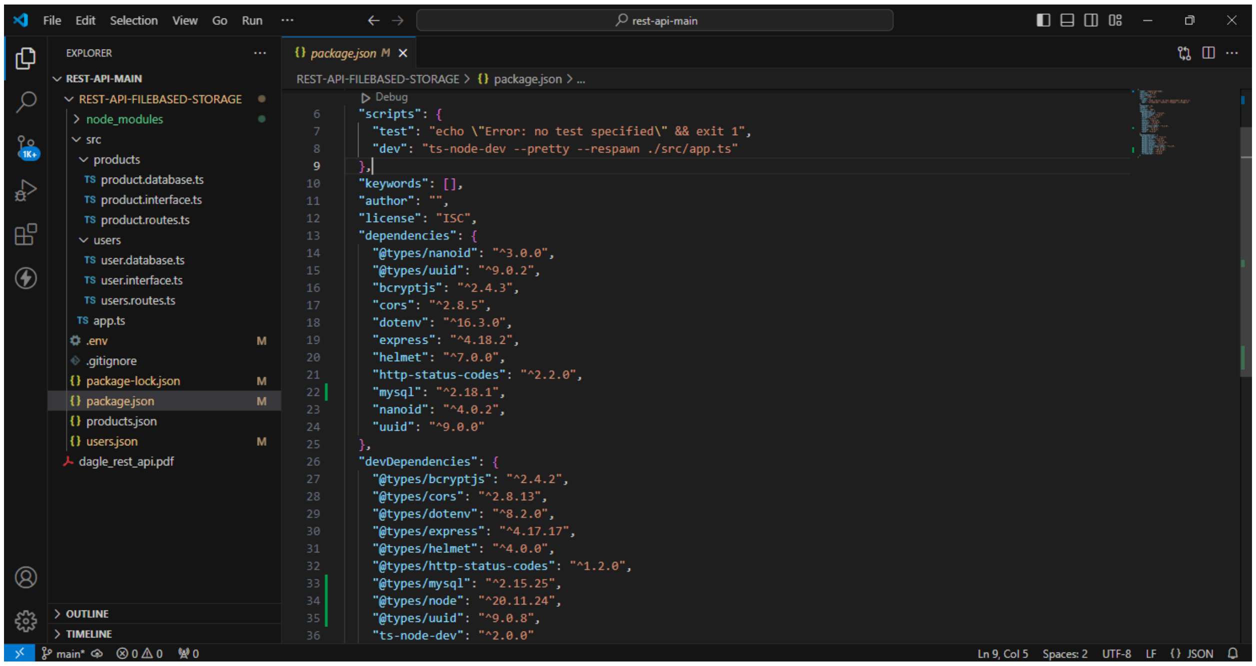
- npm i -D @types/mysql



```
PORT=7000
DB_HOST=localhost
DB_USER=root
DB_PASSWORD=
DB_NAME=mydb_dagle
```

Repopulate the .env file with a variable called **DB_HOST** with a value of “localhost”**,** **DB_USER** with a value of “root”**,** **DB_PASSWORD** and **DB_NAME** with a value of your database name.

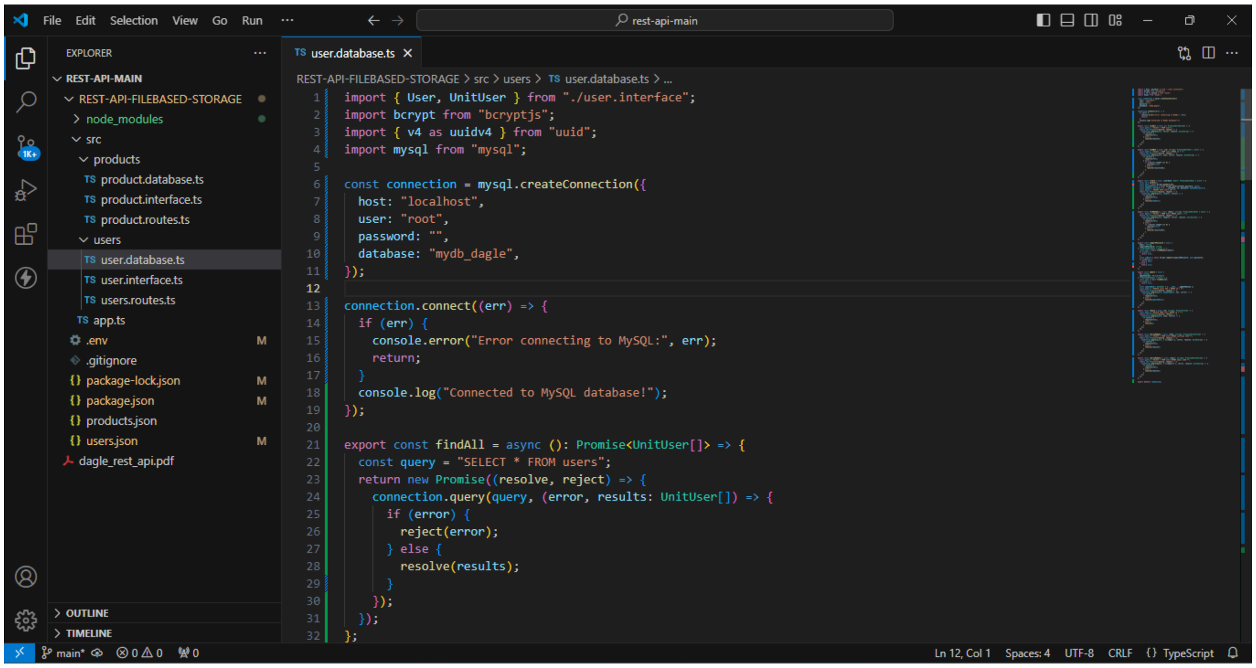
After installing the dependency, your package.json file will be updated and should look like this:



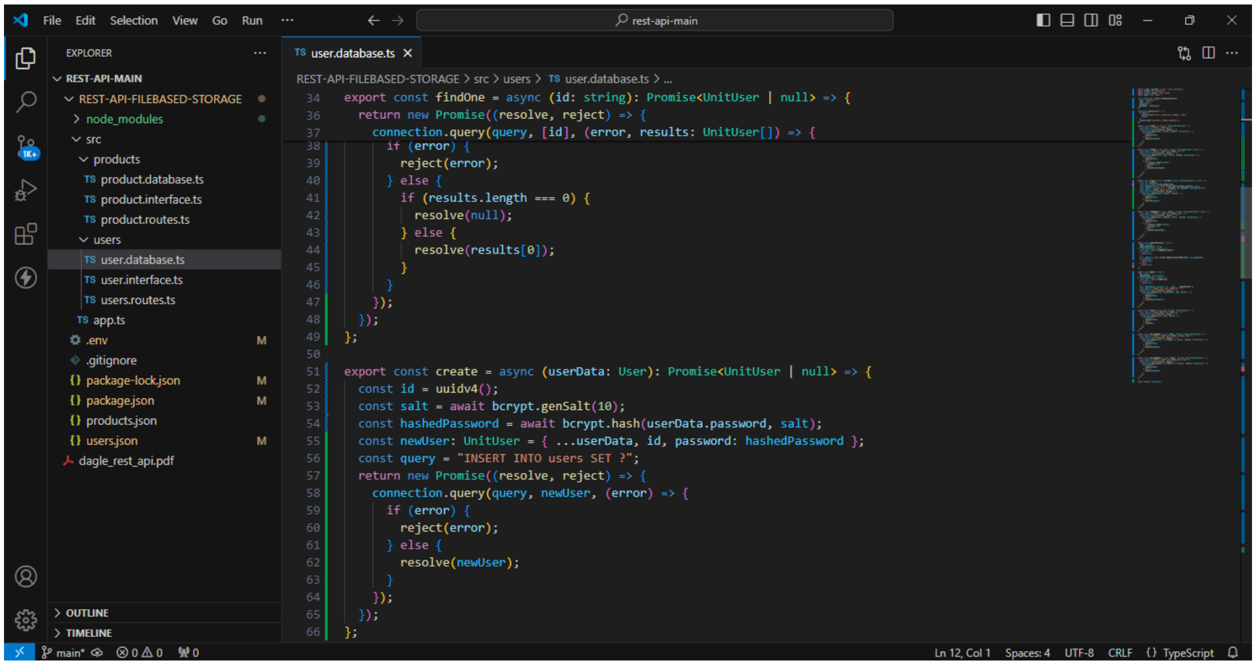
```json
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "ts-node-dev --pretty --respawn ./src/app.ts"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@types/nanoid": "^3.0.0",
    "@types/uuid": "^9.0.2",
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.0",
    "express": "^4.18.2",
    "helmet": "^7.0.0",
    "http-status-codes": "^2.2.0",
    "mysql": "^2.18.1",
    "nanoid": "^4.0.2",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "@types/bcryptjs": "^2.4.2",
    "@types/cors": "^2.8.13",
    "@types/dotenv": "^8.2.0",
    "@types/express": "^4.17.17",
    "@types/helmet": "^4.0.0",
    "@types/http-status-codes": "^1.2.0",
    "@types/mysql": "^2.15.25",
    "@types/node": "^20.11.24",
    "@types/uuid": "^9.0.8",
    "ts-node-dev": "^2.0.0"
```

Step 4: Update Users and Products Modules /Users

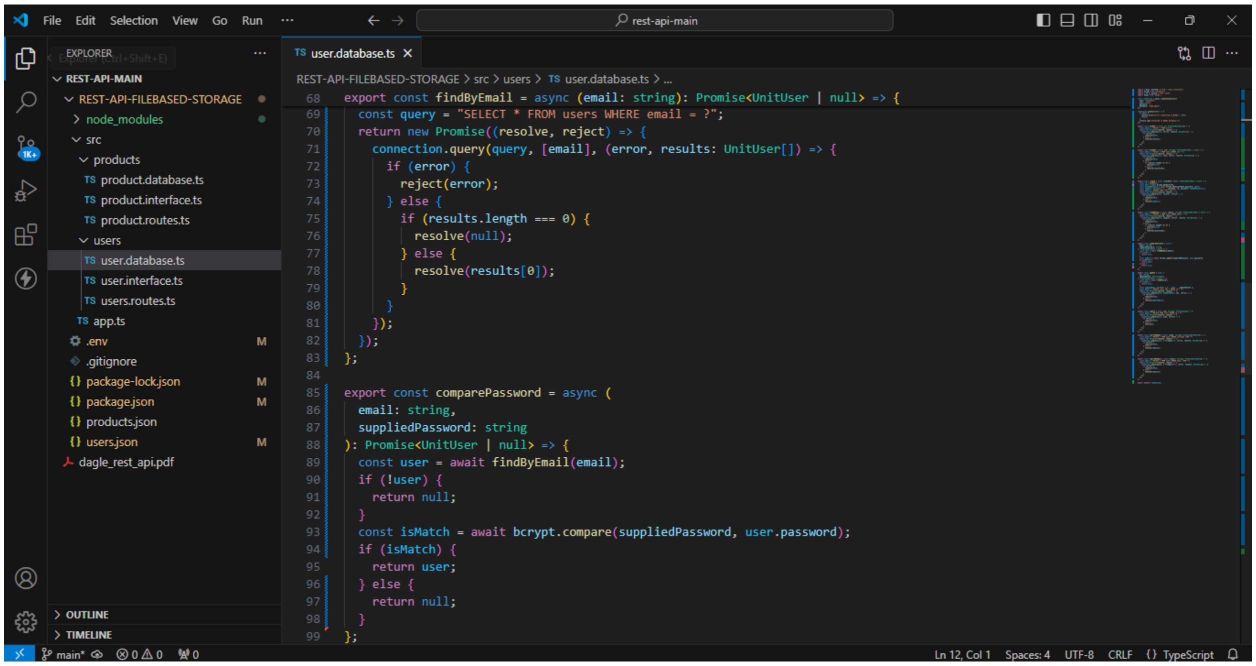
Next, we will update the config file for our database. Where you can set up the connection and be able to use the functions.
Repopulate src/users/user.database.ts with the following code:



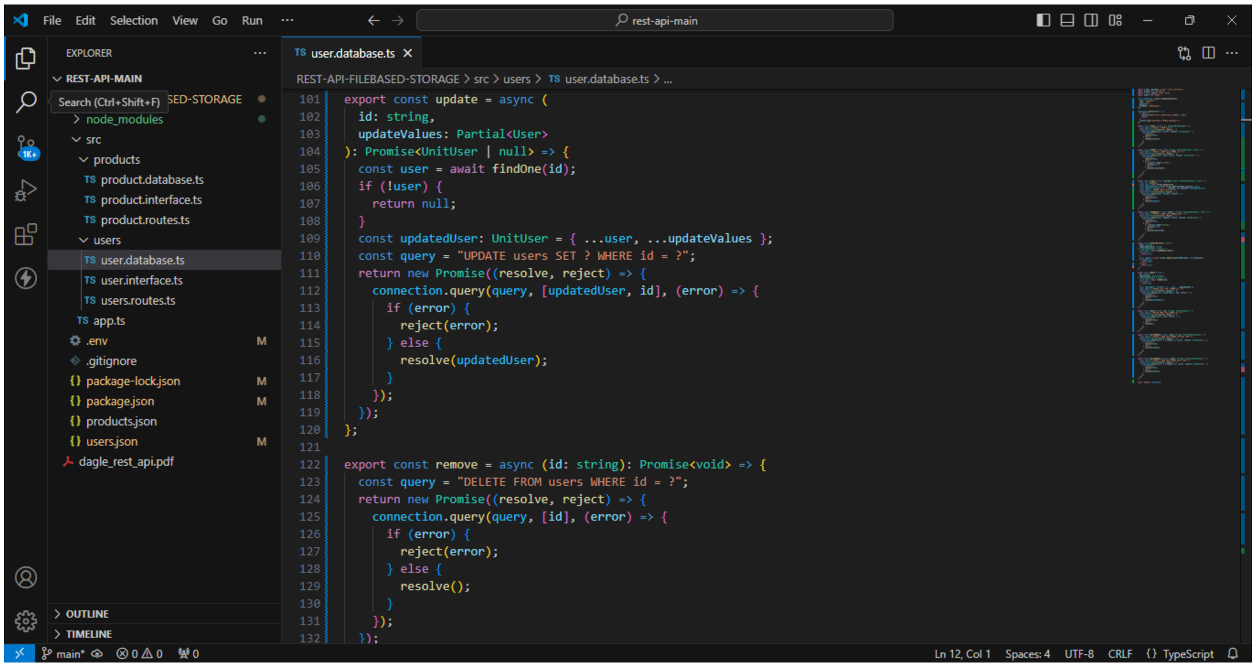
```ts
import { User, UnitUser } from "./user.interface";
import bcrypt from "bcryptjs";
import { v4 as uuidv4 } from "uuid";
import mysql from "mysql";

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "mydb_dagle",
});

connection.connect((err) => {
  if (err) {
    console.error("Error connecting to MySQL:", err);
    return;
  }
  console.log("Connected to MySQL database!");
});

export const findAll = async (): Promise<UnitUser[]> => {
  const query = "SELECT * FROM users";
  return new Promise((resolve, reject) => {
    connection.query(query, (error, results: UnitUser[]) => {
      if (error) {
        reject(error);
      } else {
        resolve(results);
      }
    });
  });
};
```



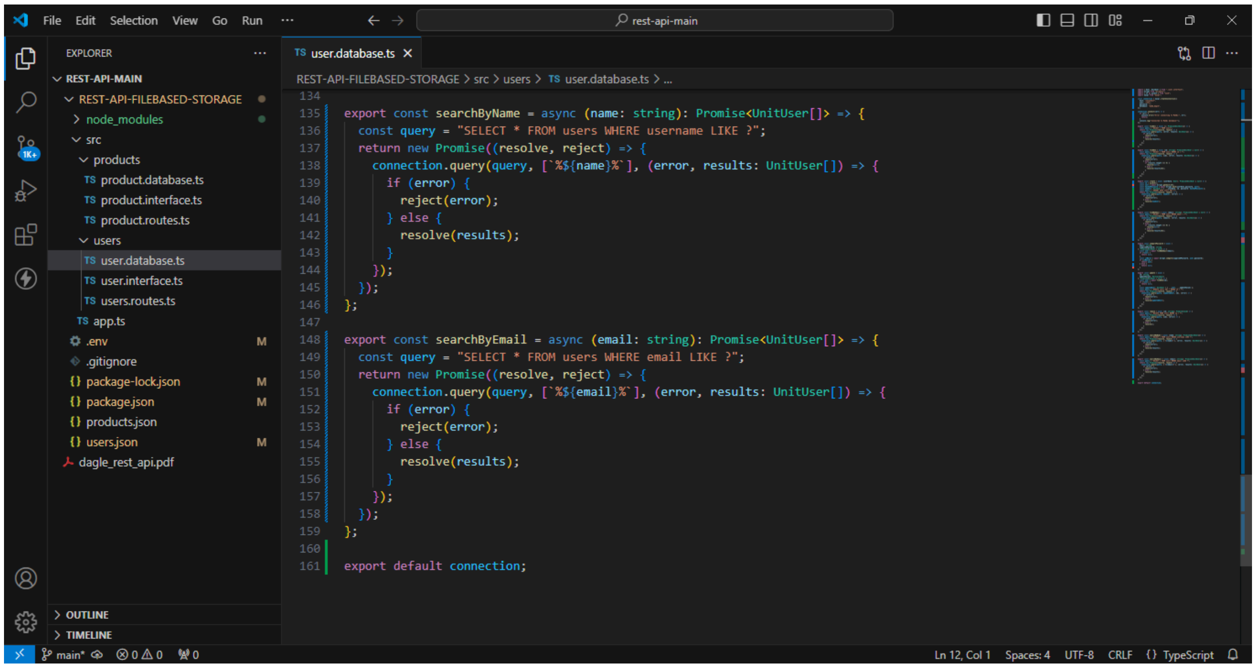
```typescript
export const findOne = async (id: string): Promise<UnitUser | null> => {
  return new Promise((resolve, reject) => {
    connection.query(query, [id], (error, results: UnitUser[]) => {
      if (error) {
        reject(error);
      } else {
        if (results.length === 0) {
          resolve(null);
        } else {
          resolve(results[0]);
        }
      }
    });
  });
};

export const create = async (userData: User): Promise<UnitUser | null> => {
  const id = uuidv4();
  const salt = await bcrypt.genSalt(10);
  const hashedPassword = await bcrypt.hash(userData.password, salt);
  const newUser: UnitUser = { ...userData, id, password: hashedPassword };
  const query = "INSERT INTO users SET ?";
  return new Promise((resolve, reject) => {
    connection.query(query, newUser, (error) => {
      if (error) {
        reject(error);
      } else {
        resolve(newUser);
      }
    });
  });
};
```



```typescript
export const findByEmail = async (email: string): Promise<UnitUser | null> => {
  const query = "SELECT * FROM users WHERE email = ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [email], (error, results: UnitUser[]) => {
      if (error) {
        reject(error);
      } else {
        if (results.length === 0) {
          resolve(null);
        } else {
          resolve(results[0]);
        }
      }
    });
  });
};

export const comparePassword = async (
  email: string,
  suppliedPassword: string
): Promise<UnitUser | null> => {
  const user = await findByEmail(email);
  if (!user) {
    return null;
  }
  const isMatch = await bcrypt.compare(suppliedPassword, user.password);
  if (isMatch) {
    return user;
  } else {
    return null;
  }
};
```

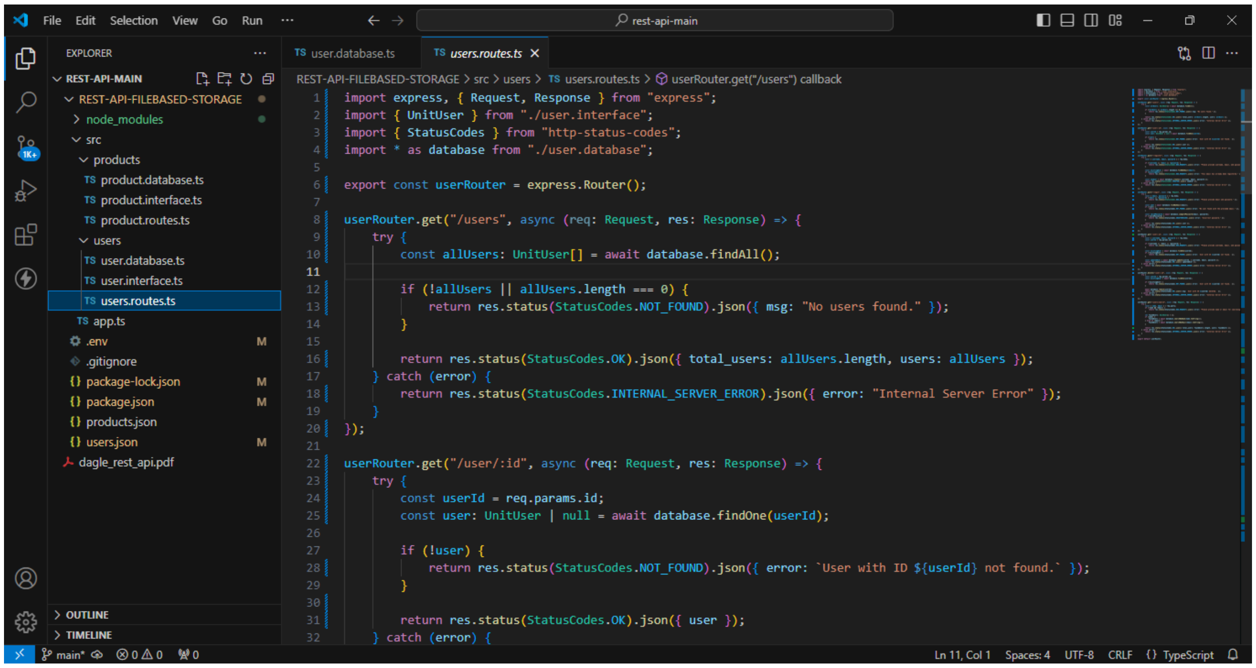


```
export const update = async (
  id: string,
  updateValues: Partial<User>
): Promise<UnitUser | null> => {
  const user = await findOne(id);
  if (!user) {
    return null;
  }
  const updatedUser: UnitUser = { ...user, ...updateValues };
  const query = "UPDATE users SET ? WHERE id = ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [updatedUser, id], (error) => {
      if (error) {
        reject(error);
      } else {
        resolve(updatedUser);
      }
    });
  });
};

export const remove = async (id: string): Promise<void> => {
  const query = "DELETE FROM users WHERE id = ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [id], (error) => {
      if (error) {
        reject(error);
      } else {
        resolve();
      }
    });
  });
```

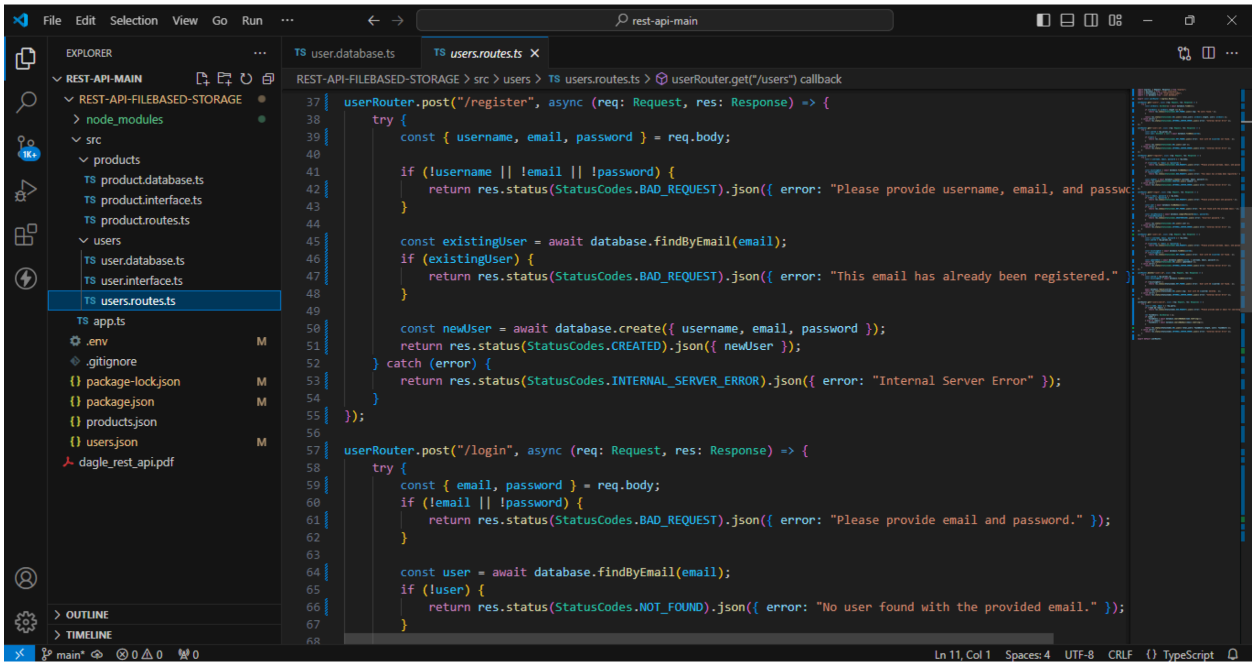


```

export const searchByName = async (name: string): Promise<UnitUser[]> => {
  const query = "SELECT * FROM users WHERE username LIKE ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [`%${name}%`], (error, results: UnitUser[]) => {
      if (error) {
        reject(error);
      } else {
        resolve(results);
      }
    });
  });
};

export const searchByEmail = async (email: string): Promise<UnitUser[]> => {
  const query = "SELECT * FROM users WHERE email LIKE ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [`%${email}%`], (error, results: UnitUser[]) => {
      if (error) {
        reject(error);
      } else {
        resolve(results);
      }
    });
  });
};

export default connection;
```

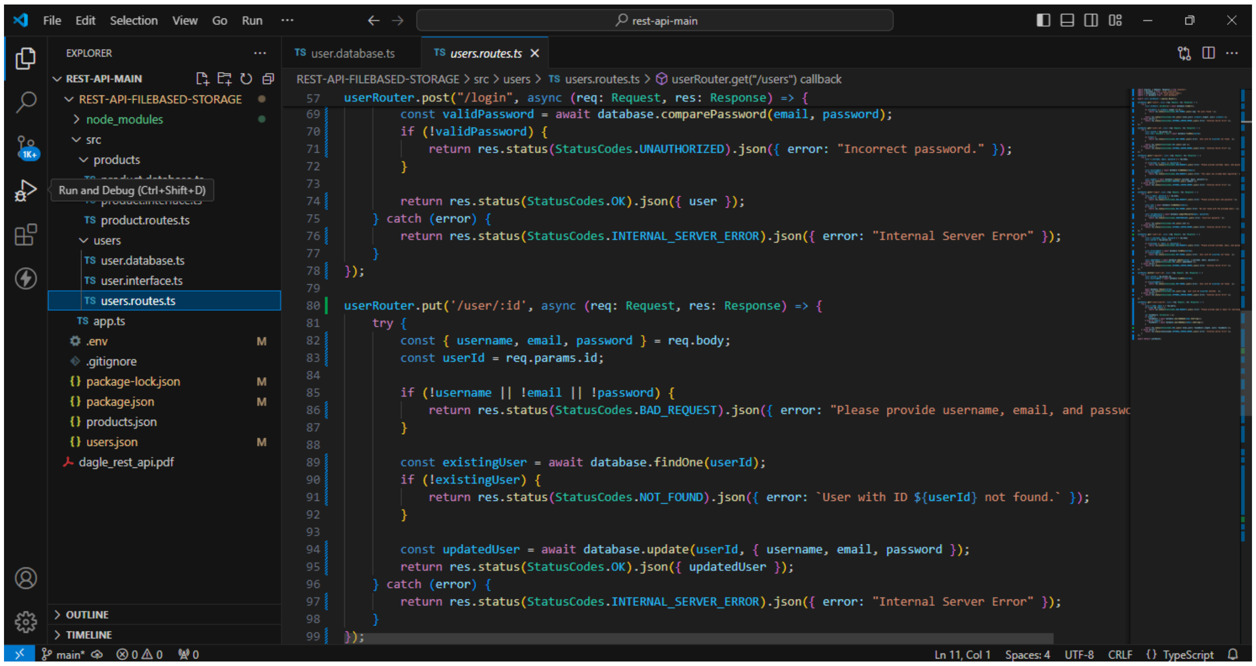
Next, let's update all the required functions and modules into the routes file ./src/users.routes.ts and repopulate as follows:



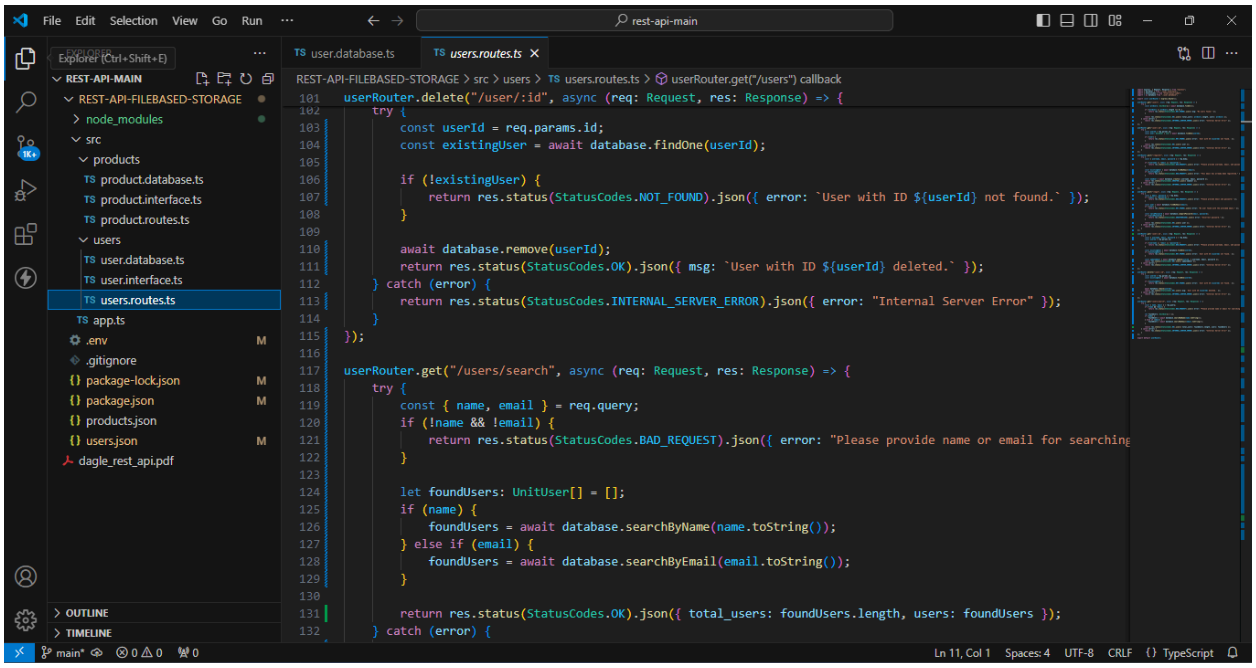
```
import express, { Request, Response } from "express";
import { UnitUser } from "./user.interface";
import { StatusCodes } from "http-status-codes";
import * as database from "./user.database";

export const userRouter = express.Router();

userRouter.get("/users", async (req: Request, res: Response) => {
    try {
        const allUsers: UnitUser[] = await database.findAll();

        if (!allUsers || allUsers.length === 0) {
            return res.status(StatusCodes.NOT_FOUND).json({ msg: "No users found." });
        }

        return res.status(StatusCodes.OK).json({ total_users: allUsers.length, users: allUsers });
    } catch (error) {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});

userRouter.get("/user/:id", async (req: Request, res: Response) => {
    try {
        const userId = req.params.id;
        const user: UnitUser | null = await database.findOne(userId);

        if (!user) {
            return res.status(StatusCodes.NOT_FOUND).json({ error: `User with ID ${userId} not found.` });
        }

        return res.status(StatusCodes.OK).json({ user });
    } catch (error) {
```



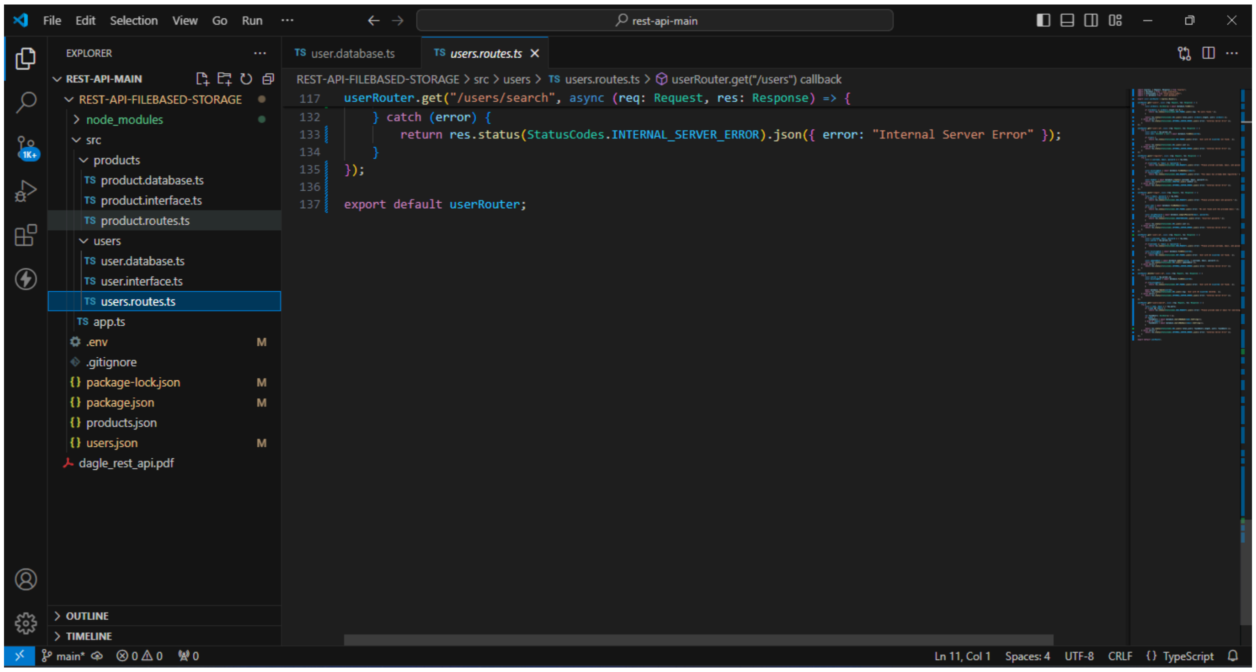
```
userRouter.post("/register", async (req: Request, res: Response) => {
    try {
        const { username, email, password } = req.body;

        if (!username || !email || !password) {
            return res.status(StatusCodes.BAD_REQUEST).json({ error: "Please provide username, email, and passwc
        }

        const existingUser = await database.findByEmail(email);
        if (existingUser) {
            return res.status(StatusCodes.BAD_REQUEST).json({ error: "This email has already been registered." }
        }

        const newUser = await database.create({ username, email, password });
        return res.status(StatusCodes.CREATED).json({ newUser });
    } catch (error) {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});

userRouter.post("/login", async (req: Request, res: Response) => {
    try {
        const { email, password } = req.body;
        if (!email || !password) {
            return res.status(StatusCodes.BAD_REQUEST).json({ error: "Please provide email and password." });
        }

        const user = await database.findByEmail(email);
        if (!user) {
            return res.status(StatusCodes.NOT_FOUND).json({ error: "No user found with the provided email." });
        }
```



```
userRouter.post("/login", async (req: Request, res: Response) => {
        const validPassword = await database.comparePassword(email, password);
        if (!validPassword) {
            return res.status(StatusCodes.UNAUTHORIZED).json({ error: "Incorrect password." });
        }

        return res.status(StatusCodes.OK).json({ user });
    } catch (error) {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});

userRouter.put('/user/:id', async (req: Request, res: Response) => {
    try {
        const { username, email, password } = req.body;
        const userId = req.params.id;

        if (!username || !email || !password) {
            return res.status(StatusCodes.BAD_REQUEST).json({ error: "Please provide username, email, and passwo
        }

        const existingUser = await database.findOne(userId);
        if (!existingUser) {
            return res.status(StatusCodes.NOT_FOUND).json({ error: `User with ID ${userId} not found.` });
        }

        const updatedUser = await database.update(userId, { username, email, password });
        return res.status(StatusCodes.OK).json({ updatedUser });
    } catch (error) {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});
```



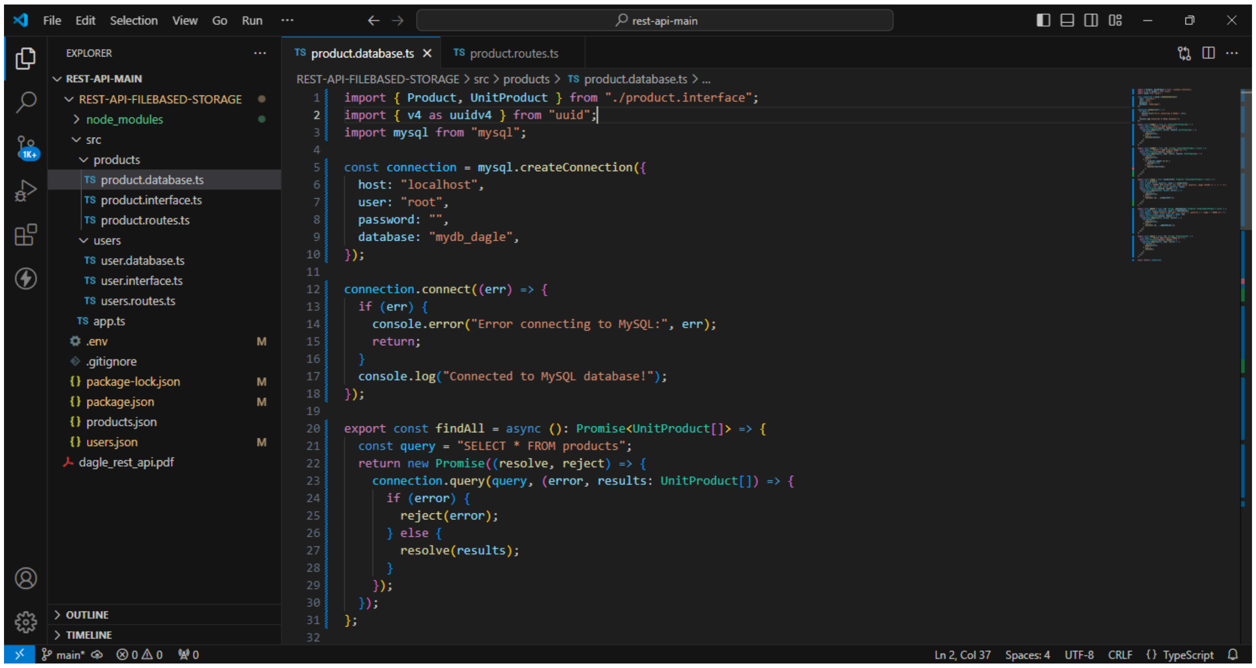
```
userRouter.delete("/user/:id", async (req: Request, res: Response) => {
    try {
        const userId = req.params.id;
        const existingUser = await database.findOne(userId);

        if (!existingUser) {
            return res.status(StatusCodes.NOT_FOUND).json({ error: `User with ID ${userId} not found.` });
        }

        await database.remove(userId);
        return res.status(StatusCodes.OK).json({ msg: `User with ID ${userId} deleted.` });
    } catch (error) {
        return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});

userRouter.get("/users/search", async (req: Request, res: Response) => {
    try {
        const { name, email } = req.query;
        if (!name && !email) {
            return res.status(StatusCodes.BAD_REQUEST).json({ error: "Please provide name or email for searching
        }

        let foundUsers: UnitUser[] = [];
        if (name) {
            foundUsers = await database.searchByName(name.toString());
        } else if (email) {
            foundUsers = await database.searchByEmail(email.toString());
        }

        return res.status(StatusCodes.OK).json({ total_users: foundUsers.length, users: foundUsers });
    } catch (error) {
```



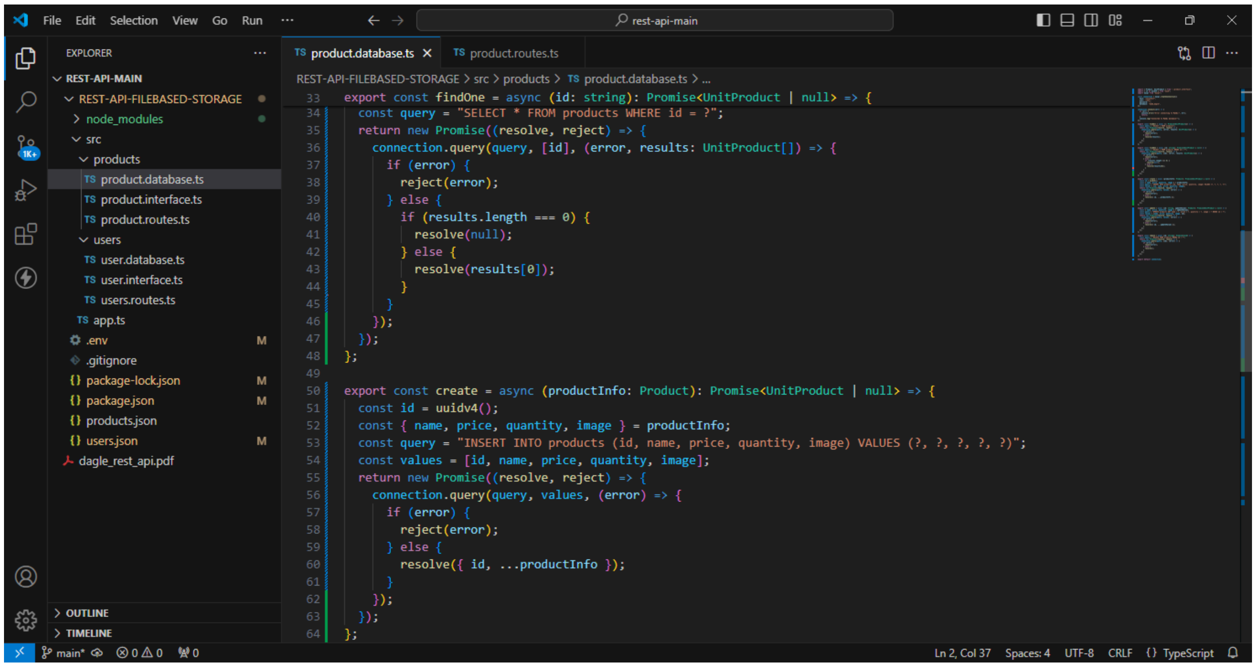
```
userRouter.get("/users/search", async (req: Request, res: Response) => {
    } catch (error) {
      return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error: "Internal Server Error" });
    }
});

export default userRouter;
```

/Products

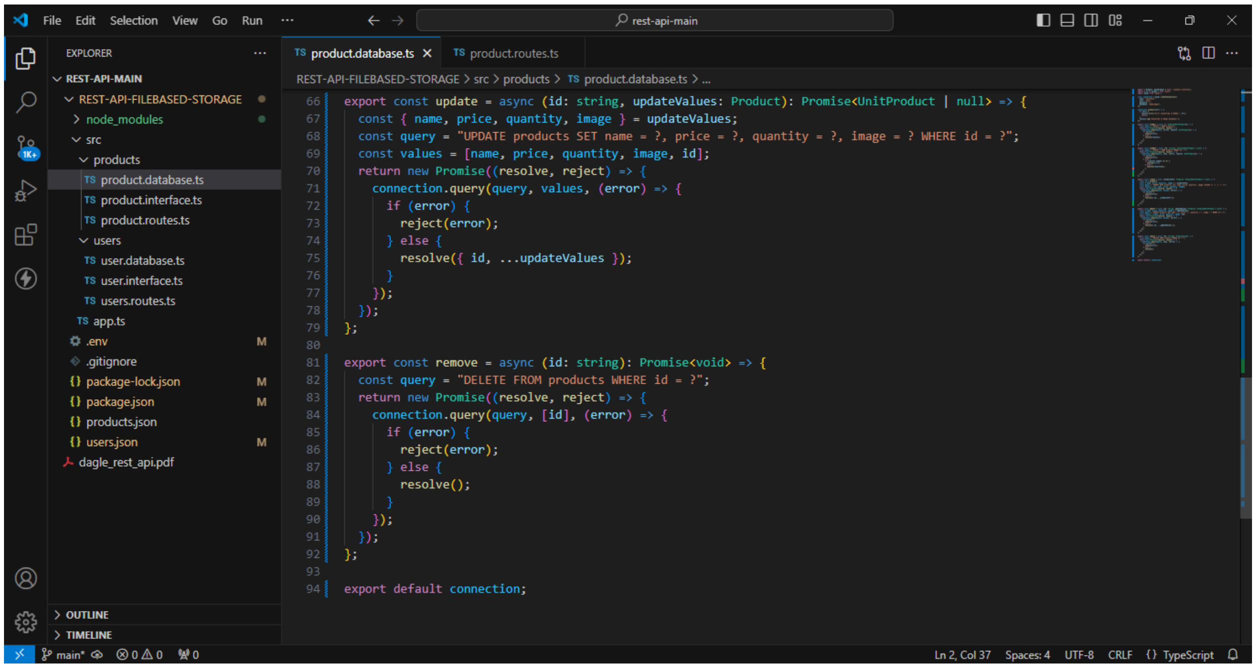
Next, just like in the ./src/users.database.ts file, let us populate the ./src/product.database.ts with a similar logic.



```
import { Product, UnitProduct } from "./product.interface";
import { v4 as uuidv4 } from "uuid";
import mysql from "mysql";

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "mydb_dagle",
});

connection.connect((err) => {
  if (err) {
    console.error("Error connecting to MySQL:", err);
    return;
  }
  console.log("Connected to MySQL database!");
});

export const findAll = async (): Promise<UnitProduct[]> => {
  const query = "SELECT * FROM products";
  return new Promise((resolve, reject) => {
    connection.query(query, (error, results: UnitProduct[]) => {
      if (error) {
        reject(error);
      } else {
        resolve(results);
      }
    });
  });
};
```

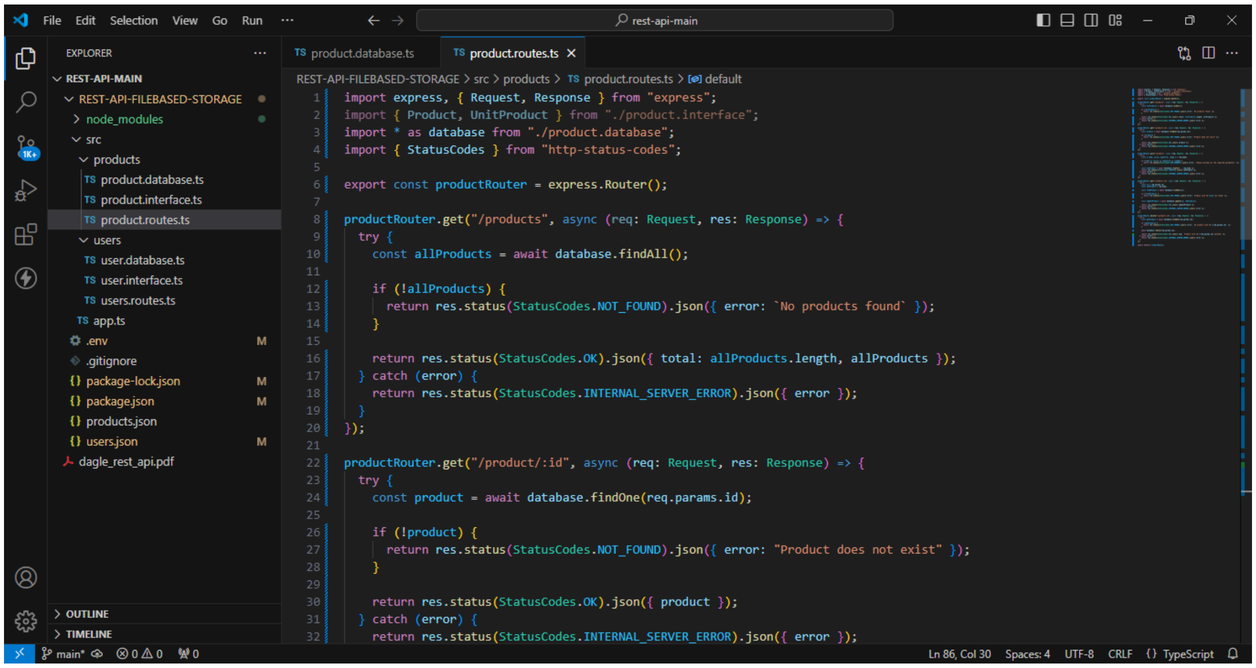


```typescript
export const findOne = async (id: string): Promise<UnitProduct | null> => {
  const query = "SELECT * FROM products WHERE id = ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [id], (error, results: UnitProduct[]) => {
      if (error) {
        reject(error);
      } else {
        if (results.length === 0) {
          resolve(null);
        } else {
          resolve(results[0]);
        }
      }
    });
  });
};

export const create = async (productInfo: Product): Promise<UnitProduct | null> => {
  const id = uuidv4();
  const { name, price, quantity, image } = productInfo;
  const query = "INSERT INTO products (id, name, price, quantity, image) VALUES (?, ?, ?, ?, ?)";
  const values = [id, name, price, quantity, image];
  return new Promise((resolve, reject) => {
    connection.query(query, values, (error) => {
      if (error) {
        reject(error);
      } else {
        resolve({ id, ...productInfo });
      }
    });
  });
};
```

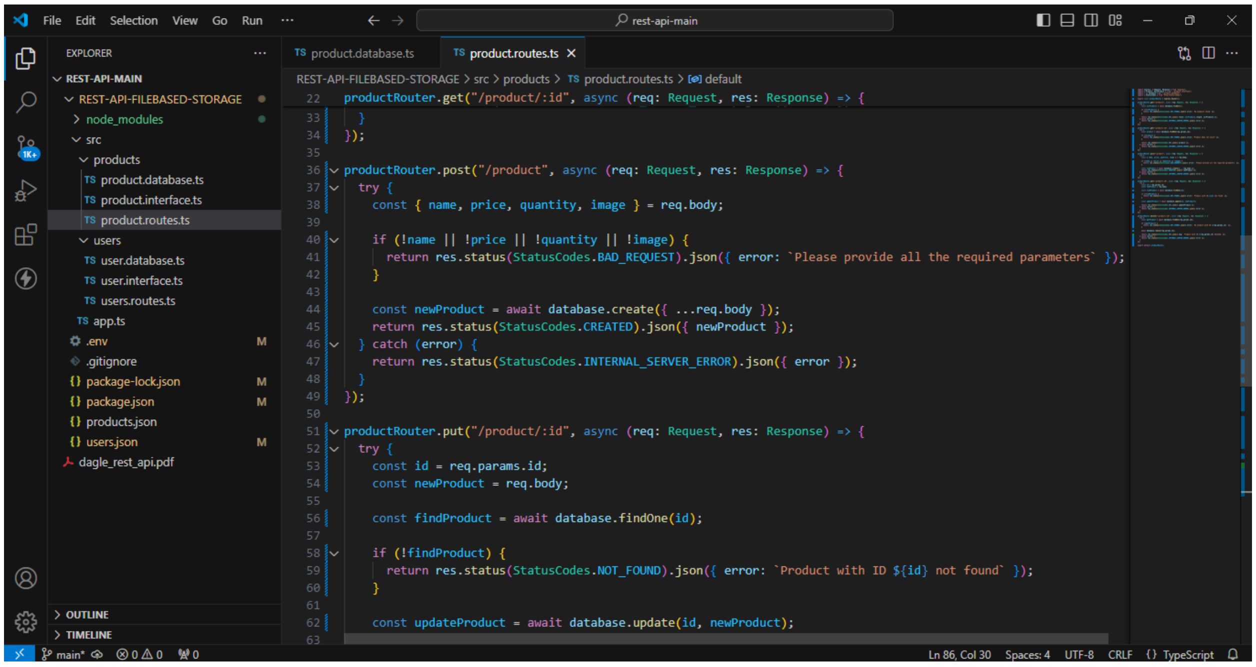


```typescript
export const update = async (id: string, updateValues: Product): Promise<UnitProduct | null> => {
  const { name, price, quantity, image } = updateValues;
  const query = "UPDATE products SET name = ?, price = ?, quantity = ?, image = ? WHERE id = ?";
  const values = [name, price, quantity, image, id];
  return new Promise((resolve, reject) => {
    connection.query(query, values, (error) => {
      if (error) {
        reject(error);
      } else {
        resolve({ id, ...updateValues });
      }
    });
  });
};

export const remove = async (id: string): Promise<void> => {
  const query = "DELETE FROM products WHERE id = ?";
  return new Promise((resolve, reject) => {
    connection.query(query, [id], (error) => {
      if (error) {
        reject(error);
      } else {
        resolve();
      }
    });
  });
};

export default connection;
```

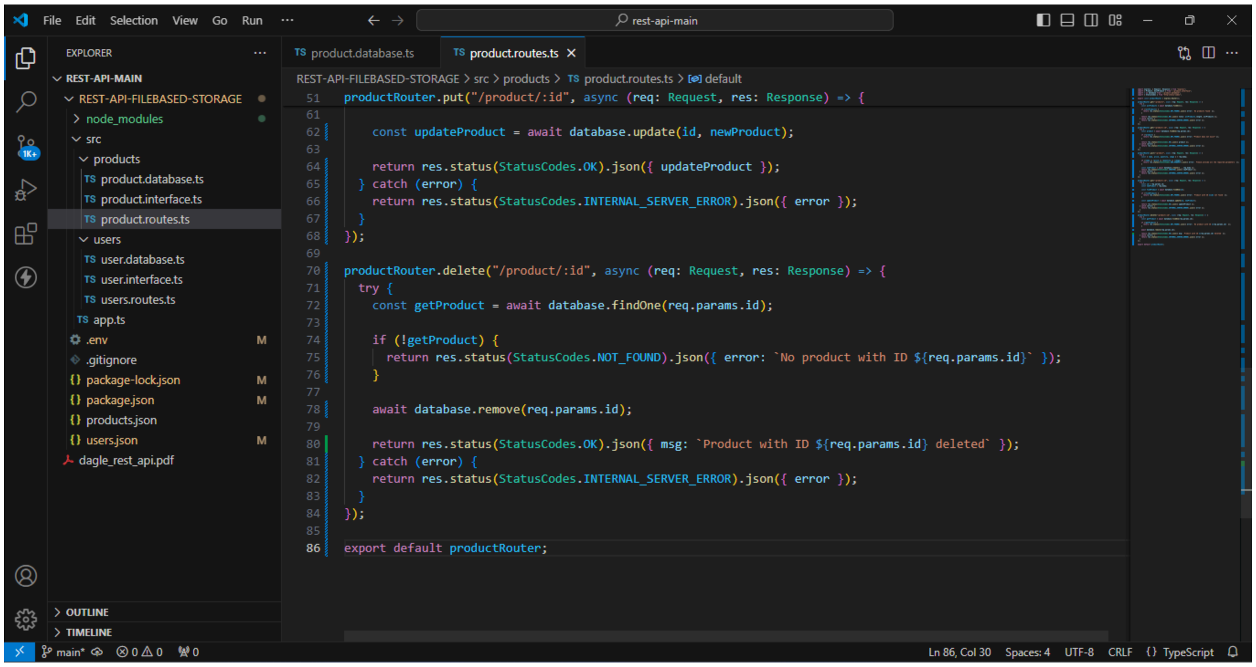
Once our logic checks out, it's time to implement the routes for our products. Populate the ./src/product.routes.ts file with the following code :


rest-api-main
product.database.ts
product.routes.ts
REST-API-FILEBASED-STORAGE > src > products > product.routes.ts > default
Ln 86, Col 30 Spaces: 4 UTF-8 CRLF TypeScript

```typescript
import express, { Request, Response } from "express";
import { Product, UnitProduct } from "./product.interface";
import * as database from "./product.database";
import { StatusCodes } from "http-status-codes";

export const productRouter = express.Router();

productRouter.get("/products", async (req: Request, res: Response) => {
  try {
    const allProducts = await database.findAll();

    if (!allProducts) {
      return res.status(StatusCodes.NOT_FOUND).json({ error: `No products found` });
    }

    return res.status(StatusCodes.OK).json({ total: allProducts.length, allProducts });
  } catch (error) {
    return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error });
  }
});

productRouter.get("/product/:id", async (req: Request, res: Response) => {
  try {
    const product = await database.findOne(req.params.id);

    if (!product) {
      return res.status(StatusCodes.NOT_FOUND).json({ error: "Product does not exist" });
    }

    return res.status(StatusCodes.OK).json({ product });
  } catch (error) {
    return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error });
```


rest-api-main
product.database.ts
product.routes.ts
REST-API-FILEBASED-STORAGE > src > products > product.routes.ts > default
Ln 86, Col 30 Spaces: 4 UTF-8 CRLF TypeScript

```typescript
productRouter.get("/product/:id", async (req: Request, res: Response) => {
  }
});

productRouter.post("/product", async (req: Request, res: Response) => {
  try {
    const { name, price, quantity, image } = req.body;

    if (!name || !price || !quantity || !image) {
      return res.status(StatusCodes.BAD_REQUEST).json({ error: `Please provide all the required parameters` });
    }

    const newProduct = await database.create({ ...req.body });
    return res.status(StatusCodes.CREATED).json({ newProduct });
  } catch (error) {
    return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error });
  }
});

productRouter.put("/product/:id", async (req: Request, res: Response) => {
  try {
    const id = req.params.id;
    const newProduct = req.body;

    const findProduct = await database.findOne(id);

    if (!findProduct) {
      return res.status(StatusCodes.NOT_FOUND).json({ error: `Product with ID ${id} not found` });
    }

    const updateProduct = await database.update(id, newProduct);
```

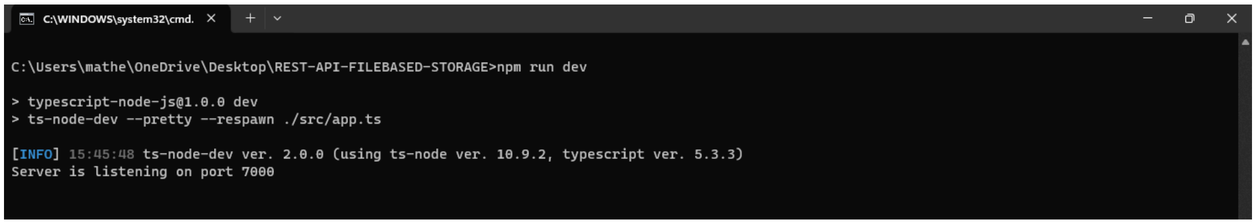


```typescript
productRouter.put("/product/:id", async (req: Request, res: Response) => {

    const updateProduct = await database.update(id, newProduct);

    return res.status(StatusCodes.OK).json({ updateProduct });
  } catch (error) {
    return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error });
  }
});

productRouter.delete("/product/:id", async (req: Request, res: Response) => {
  try {
    const getProduct = await database.findOne(req.params.id);

    if (!getProduct) {
      return res.status(StatusCodes.NOT_FOUND).json({ error: `No product with ID ${req.params.id}` });
    }

    await database.remove(req.params.id);

    return res.status(StatusCodes.OK).json({ msg: `Product with ID ${req.params.id} deleted` });
  } catch (error) {
    return res.status(StatusCodes.INTERNAL_SERVER_ERROR).json({ error });
  }
});

export default productRouter;
```

Step 5: Testing the API

Start the server and test our API using Thunder Client (VS Code Extension).
Note: You can use any other app to test the API if you had used a different IDE.

- run the npm run dev command in your terminal
- If there are no errors, the server should be listening to port 7000

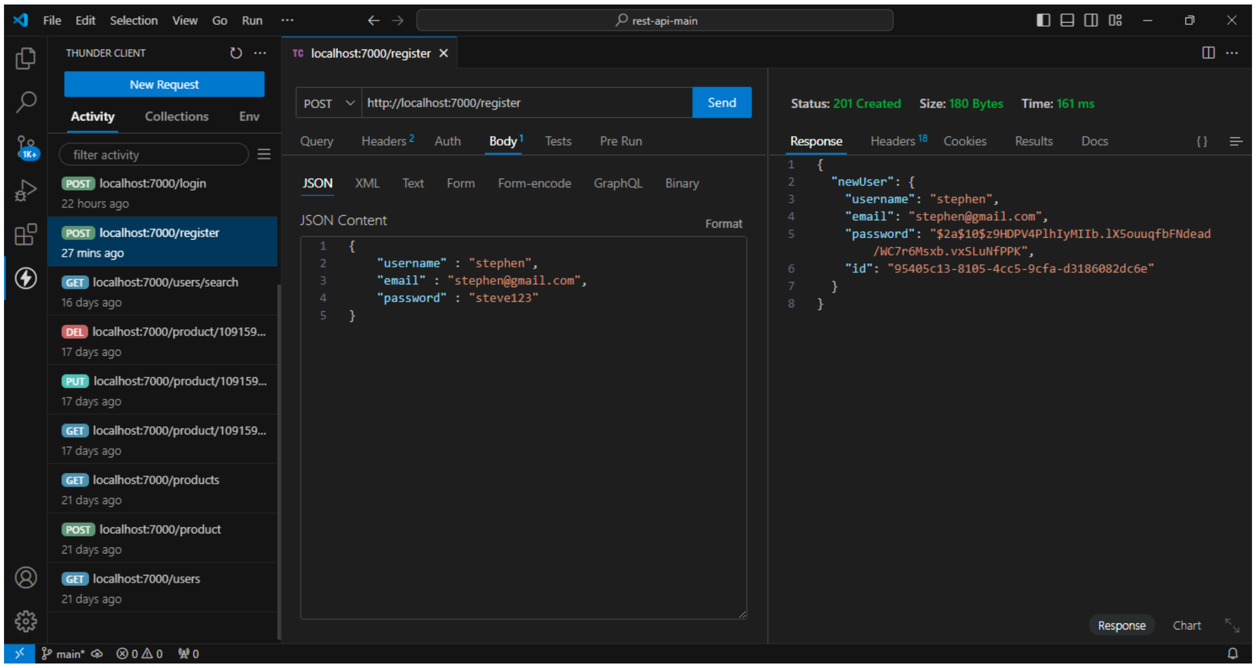


```
C:\Users\mathe\OneDrive\Desktop\REST-API-FILEBASED-STORAGE>npm run dev

> typescript-node-js@1.0.0 dev
> ts-node-dev --pretty --respawn ./src/app.ts

[INFO] 15:45:48 ts-node-dev ver. 2.0.0 (using ts-node ver. 10.9.2, typescript ver. 5.3.3)
Server is listening on port 7000
```

Making Requests (Thunder Client)

- Click on the new request button
- You can configure the request url and what type of request you want to send on the page next to the activity/collections/env tab.
- If you are done with the configurations then click the send button.

Register a user (And Response after sending the request)



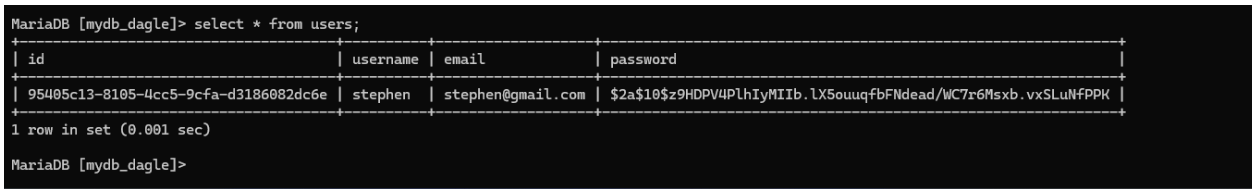
```
POST  http://localhost:7000/register  Send
```

```
{
    "username" : "stephen",
    "email" : "stephen@gmail.com",
    "password" : "steve123"
}
```

Status: 201 Created Size: 180 Bytes Time: 161 ms

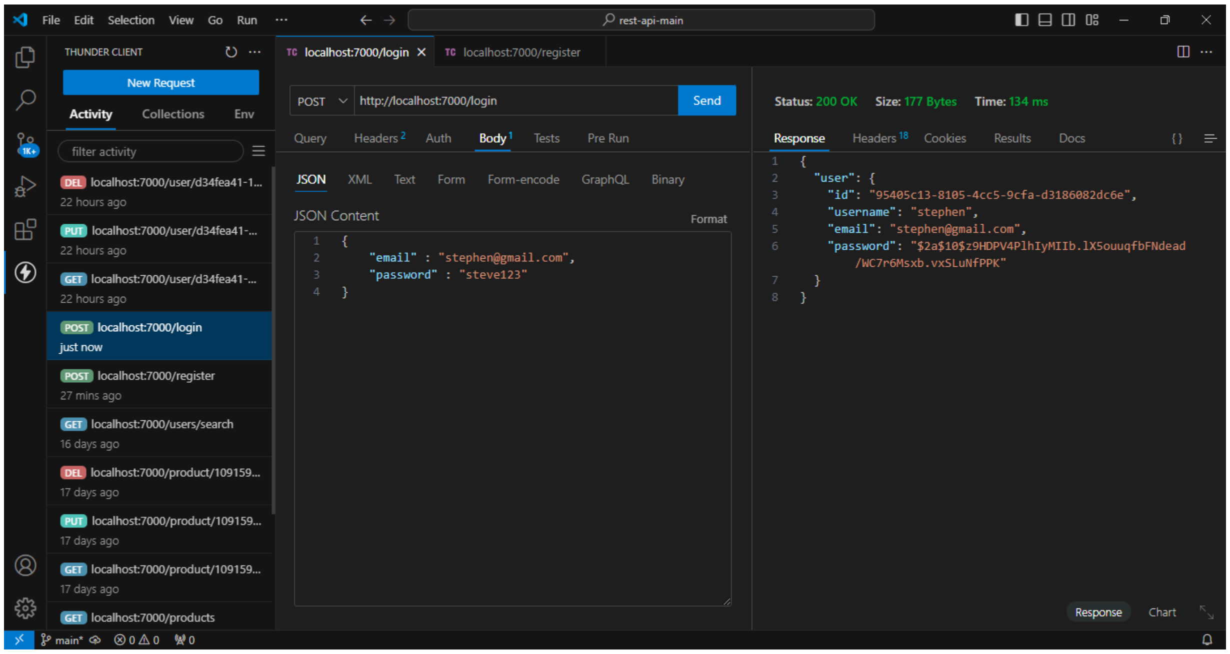
```
{
  "newUser": {
    "username": "stephen",
    "email": "stephen@gmail.com",
    "password": "$2a$10$z9HDPV4PlhIyMIIb.lX5ouuqfbFNdead
      /WC7r6Msxb.vxSLuNfPPK",
    "id": "95405c13-8105-4cc5-9cfa-d3186082dc6e"
  }
}
```

Register a user (Changes reflected on the database after request has been done)
Note: Use “select * from [tablename]” to show the contents of the table.



```
MariaDB [mydb_dagle]> select * from users;
+--------------------------------------+----------+-------------------+--------------------------------------------------------------+
| id                                   | username | email             | password                                                     |
+--------------------------------------+----------+-------------------+--------------------------------------------------------------+
| 95405c13-8105-4cc5-9cfa-d3186082dc6e | stephen  | stephen@gmail.com | $2a$10$z9HDPV4PlhIyMIIb.lX5ouuqfbFNdead/WC7r6Msxb.vxSLuNfPPK |
+--------------------------------------+----------+-------------------+--------------------------------------------------------------+
1 row in set (0.001 sec)

MariaDB [mydb_dagle]>
```

Login user (And Response after sending the request)



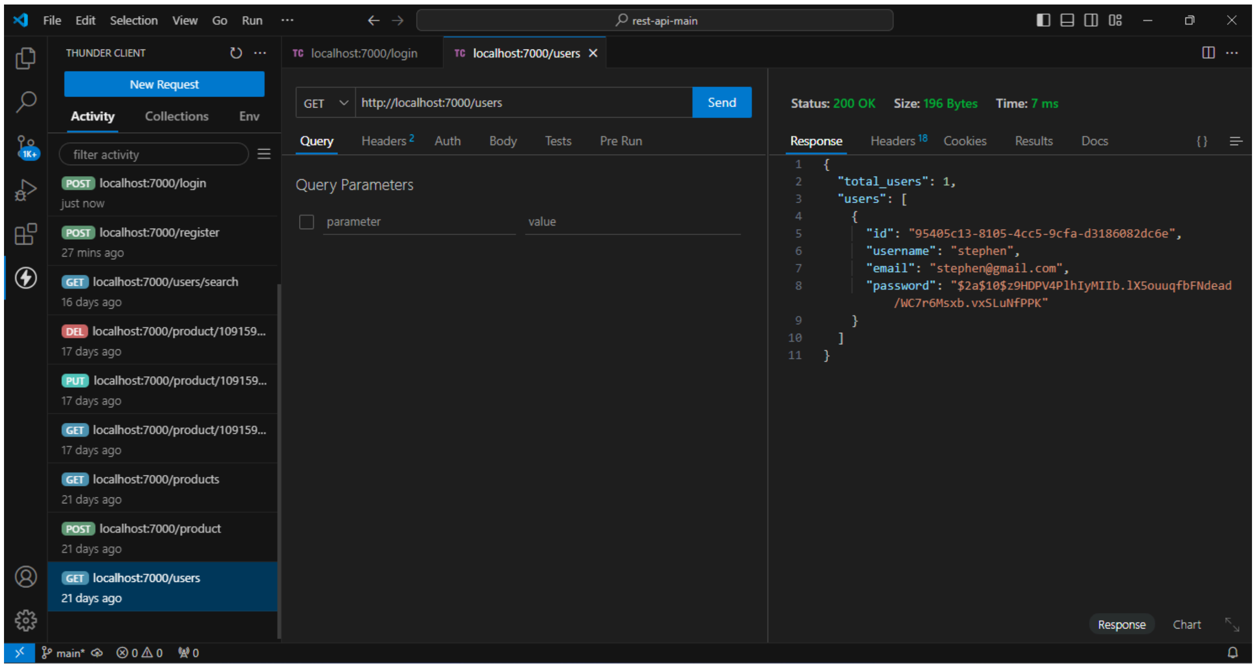
```
POST  http://localhost:7000/login  Send
```

```
{
    "email" : "stephen@gmail.com",
    "password" : "steve123"
}
```

Status: 200 OK Size: 177 Bytes Time: 134 ms

```
{
  "user": {
    "id": "95405c13-8105-4cc5-9cfa-d3186082dc6e",
    "username": "stephen",
    "email": "stephen@gmail.com",
    "password": "$2a$10$z9HDPV4PlhIyMIIb.lX5ouuqfbFNdead
      /WC7r6Msxb.vxSLuNfPPK"
  }
}
```

Get all users (List all the registered users)

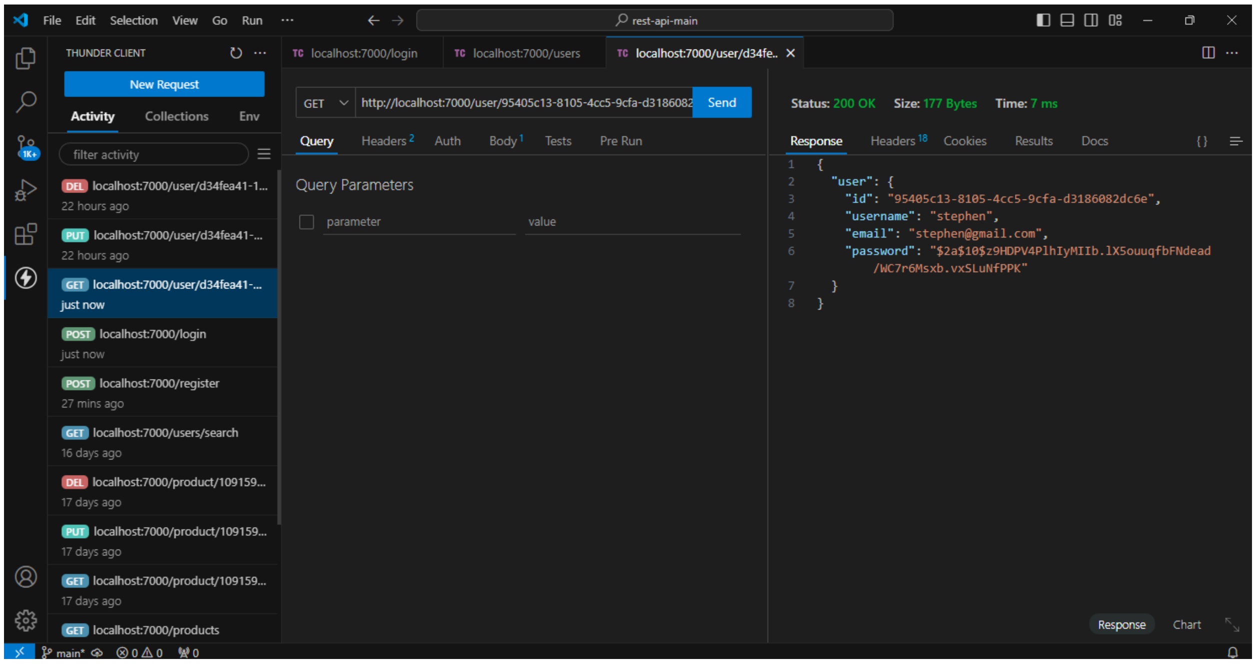


GET http://localhost:7000/users

Status: 200 OK Size: 196 Bytes Time: 7 ms

```
{
  "total_users": 1,
  "users": [
    {
      "id": "95405c13-8105-4cc5-9cfa-d3186082dc6e",
      "username": "stephen",
      "email": "stephen@gmail.com",
      "password": "$2a$10$z9HDPV4PlhIyMIIb.lX5ouuqfbFNdead
        /WC7r6Msxb.vxSLuNfPPK"
    }
  ]
}
```

Get a single user (By ID)

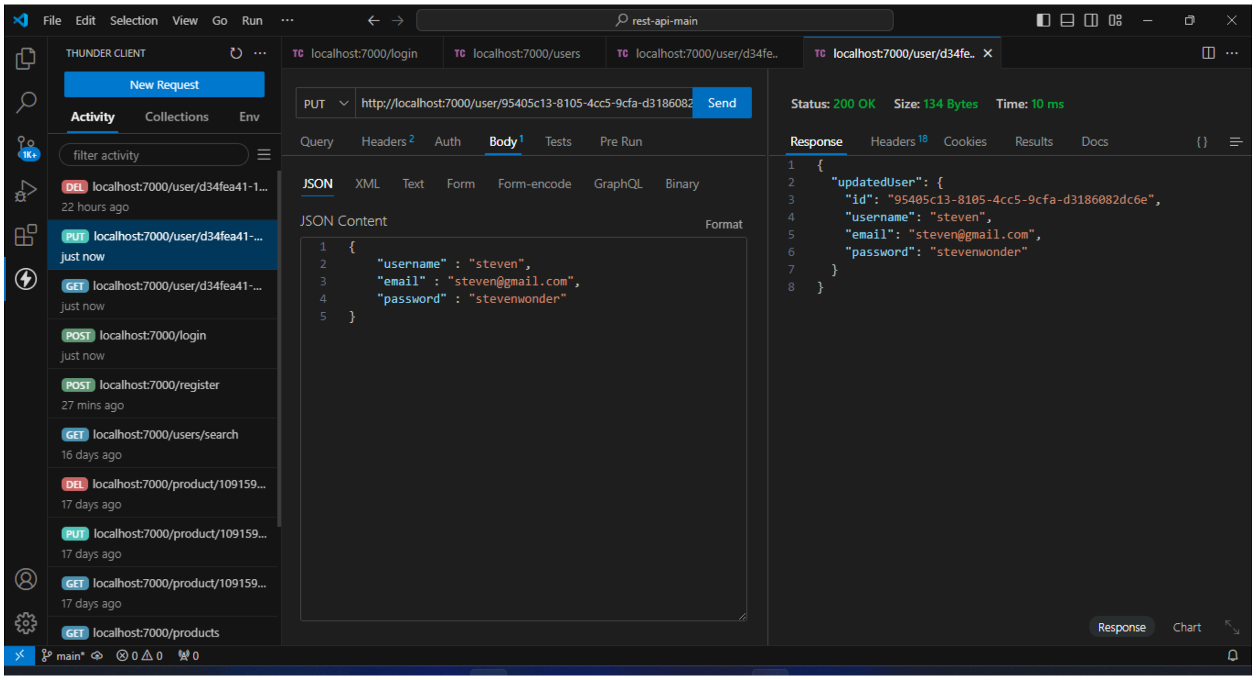


GET http://localhost:7000/user/95405c13-8105-4cc5-9cfa-d3186082

Status: 200 OK Size: 177 Bytes Time: 7 ms

```
{
  "user": {
    "id": "95405c13-8105-4cc5-9cfa-d3186082dc6e",
    "username": "stephen",
    "email": "stephen@gmail.com",
    "password": "$2a$10$z9HDPV4PlhIyMIIb.lX5ouuqfbFNdead
      /WC7r6Msxb.vxSLuNfPPK"
  }
}
```

Update a User (By ID) And Response after sending the request.



```
PUT  http://localhost:7000/user/95405c13-8105-4cc5-9cfa-d3186082  Send
```

JSON Content

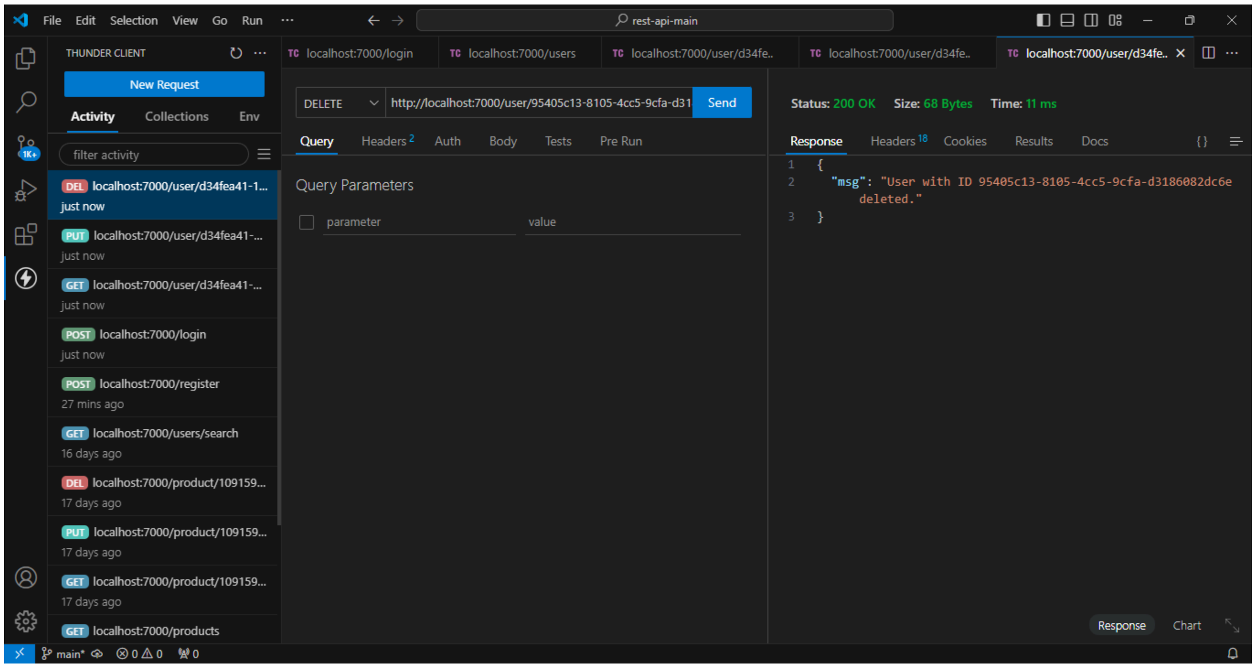
```
{
    "username" : "steven",
    "email" : "steven@gmail.com",
    "password" : "stevenwonder"
}
```

Status: 200 OK Size: 134 Bytes Time: 10 ms

```
{
  "updatedUser": {
    "id": "95405c13-8105-4cc5-9cfa-d3186082dc6e",
    "username": "steven",
    "email": "steven@gmail.com",
    "password": "stevenwonder"
  }
}
```

```
DEL localhost:7000/user/d34fea41-1...   22 hours ago
PUT localhost:7000/user/d34fea41-...    just now
GET localhost:7000/user/d34fea41-...    just now
POST localhost:7000/login               just now
POST localhost:7000/register            27 mins ago
GET localhost:7000/users/search         16 days ago
DEL localhost:7000/product/109159...    17 days ago
PUT localhost:7000/product/109159...    17 days ago
GET localhost:7000/product/109159...    17 days ago
GET localhost:7000/products
```

Delete a User (By ID)



```
DELETE  http://localhost:7000/user/95405c13-8105-4cc5-9cfa-d31  Send
```

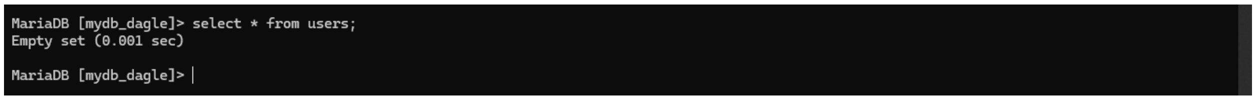
Query Parameters

Status: 200 OK Size: 68 Bytes Time: 11 ms

```
{
  "msg": "User with ID 95405c13-8105-4cc5-9cfa-d3186082dc6e
    deleted."
}
```

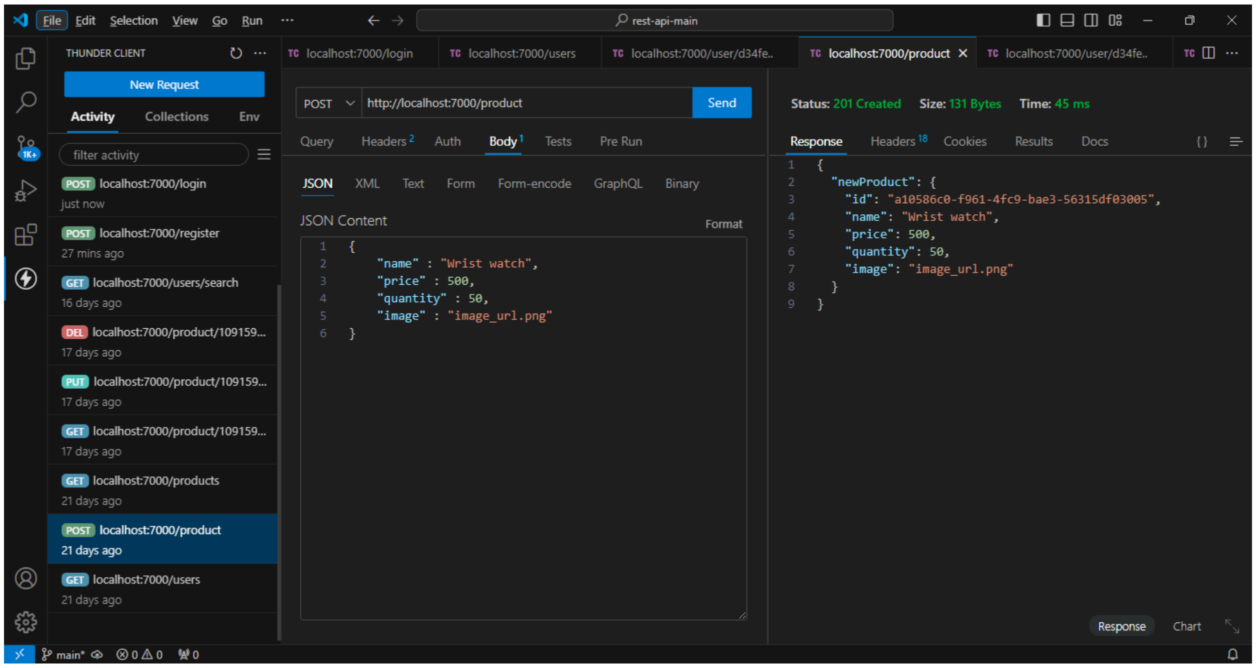
```
DEL localhost:7000/user/d34fea41-1...   just now
PUT localhost:7000/user/d34fea41-...    just now
GET localhost:7000/user/d34fea41-...    just now
POST localhost:7000/login               just now
POST localhost:7000/register            27 mins ago
GET localhost:7000/users/search         16 days ago
DEL localhost:7000/product/109159...    17 days ago
PUT localhost:7000/product/109159...    17 days ago
GET localhost:7000/product/109159...    17 days ago
GET localhost:7000/products
```

Delete a User (Changes reflected on the database after request has been done)
Note: Use “select * from [tablename]” to show the contents of the table.



```
MariaDB [mydb_dagle]> select * from users;
Empty set (0.001 sec)

MariaDB [mydb_dagle]> |
```

Create a product (And Response after sending the request)



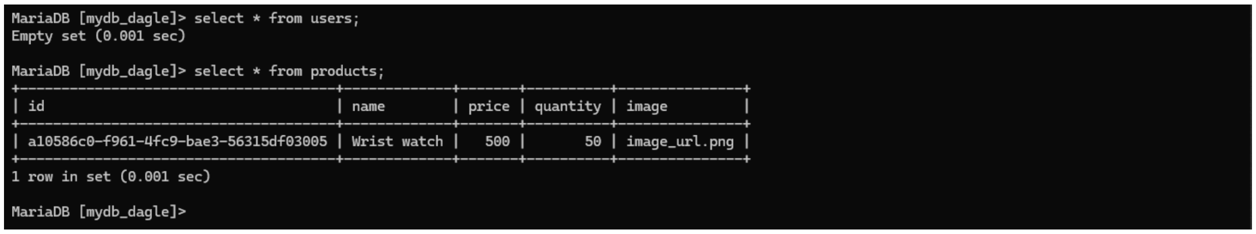
POST http://localhost:7000/product Send

```
{
    "name" : "Wrist watch",
    "price" : 500,
    "quantity" : 50,
    "image" : "image_url.png"
}
```

Status: 201 Created Size: 131 Bytes Time: 45 ms

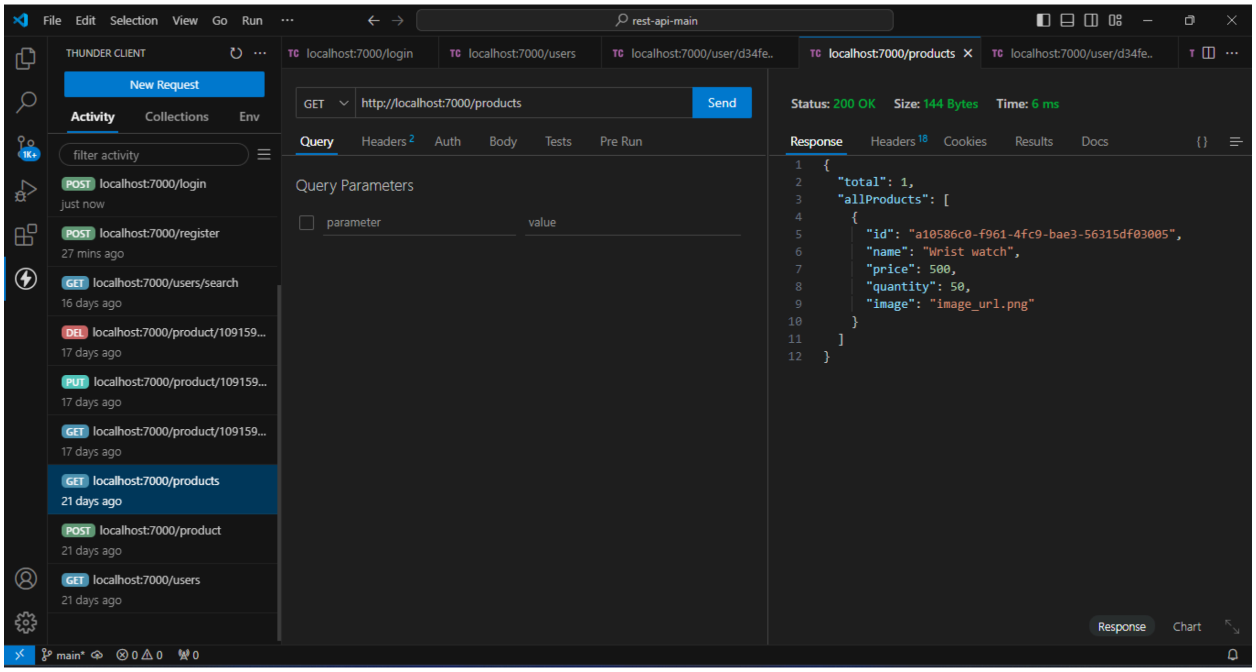
```
{
  "newProduct": {
    "id": "a10586c0-f961-4fc9-bae3-56315df03005",
    "name": "Wrist watch",
    "price": 500,
    "quantity": 50,
    "image": "image_url.png"
  }
}
```

Create a product (Changes reflected on the database after request has been done)
Note: Use “select * from [tablename]” to show the contents of the table.



```
MariaDB [mydb_dagle]> select * from users;
Empty set (0.001 sec)

MariaDB [mydb_dagle]> select * from products;
+--------------------------------------+-------------+-------+----------+---------------+
| id                                   | name        | price | quantity | image         |
+--------------------------------------+-------------+-------+----------+---------------+
| a10586c0-f961-4fc9-bae3-56315df03005 | Wrist watch |   500 |       50 | image_url.png |
+--------------------------------------+-------------+-------+----------+---------------+
1 row in set (0.001 sec)

MariaDB [mydb_dagle]>
```

Get all products (List all the created products)



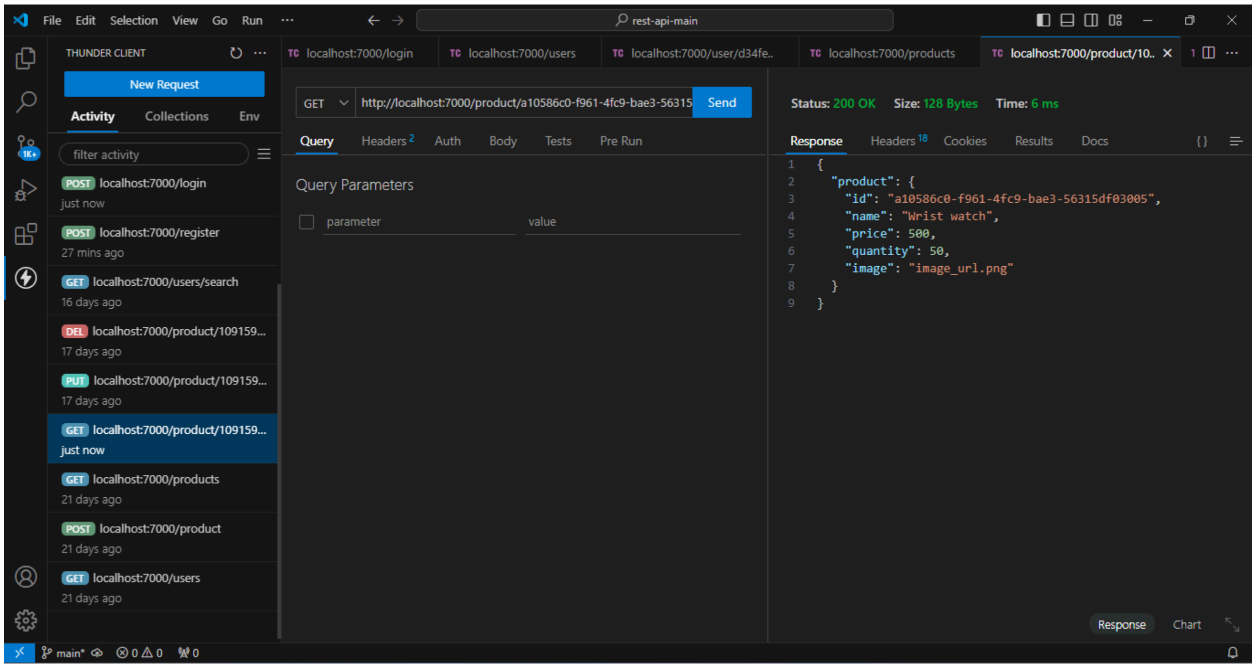
File Edit Selection View Go Run ···
rest-api-main
THUNDER CLIENT
New Request
Activity Collections Env
filter activity
POST localhost:7000/login just now
POST localhost:7000/register 27 mins ago
GET localhost:7000/users/search 16 days ago
DEL localhost:7000/product/109159... 17 days ago
PUT localhost:7000/product/109159... 17 days ago
GET localhost:7000/product/109159... 17 days ago
GET localhost:7000/products 21 days ago
POST localhost:7000/product 21 days ago
GET localhost:7000/users 21 days ago
localhost:7000/login
localhost:7000/users
localhost:7000/user/d34fe..
localhost:7000/products
localhost:7000/user/d34fe..
Query Headers 2 Auth Body Tests Pre Run
Query Parameters
parameter value
Response Headers 18 Cookies Results Docs
Response Chart
main*

GET http://localhost:7000/products Send

Status: 200 OK Size: 144 Bytes Time: 6 ms

```
{
  "total": 1,
  "allProducts": [
    {
      "id": "a10586c0-f961-4fc9-bae3-56315df03005",
      "name": "Wrist watch",
      "price": 500,
      "quantity": 50,
      "image": "image_url.png"
    }
  ]
}
```

Get a single product (By ID)



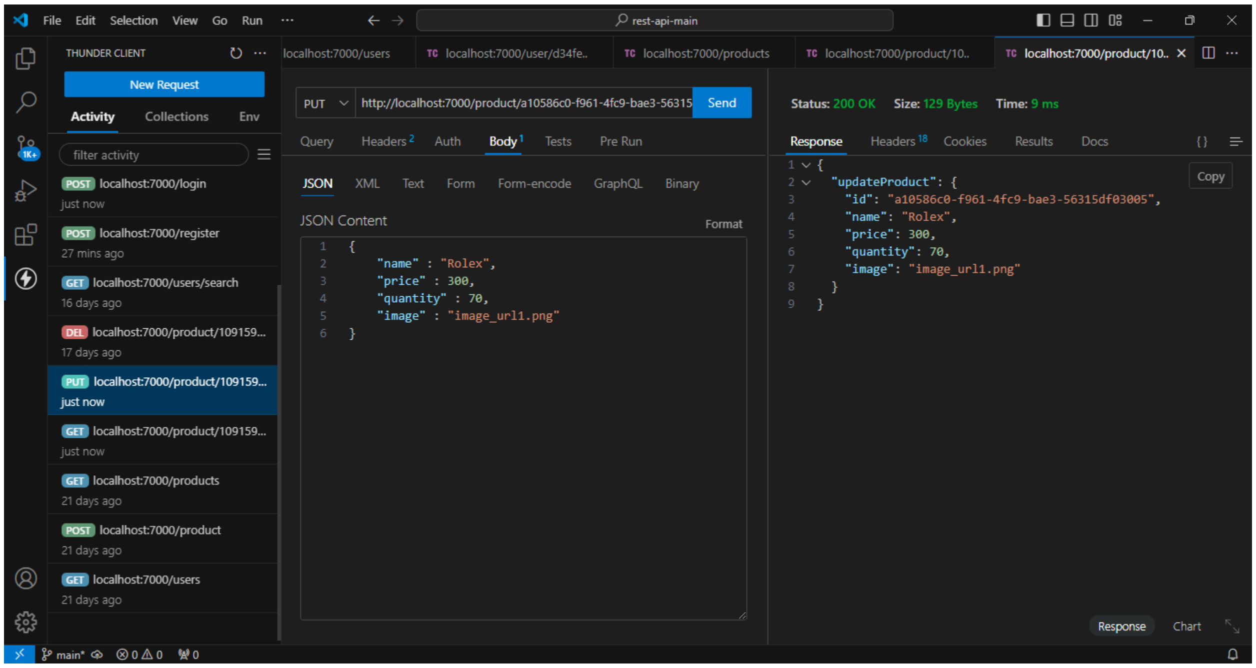
File Edit Selection View Go Run ···
rest-api-main
THUNDER CLIENT
New Request
Activity Collections Env
filter activity
POST localhost:7000/login just now
POST localhost:7000/register 27 mins ago
GET localhost:7000/users/search 16 days ago
DEL localhost:7000/product/109159... 17 days ago
PUT localhost:7000/product/109159... 17 days ago
GET localhost:7000/product/109159... just now
GET localhost:7000/products 21 days ago
POST localhost:7000/product 21 days ago
GET localhost:7000/users 21 days ago
localhost:7000/login
localhost:7000/users
localhost:7000/user/d34fe..
localhost:7000/products
localhost:7000/product/10..
Query Headers 2 Auth Body Tests Pre Run
Query Parameters
parameter value
Response Headers 18 Cookies Results Docs
Response Chart
main*

GET http://localhost:7000/product/a10586c0-f961-4fc9-bae3-56315 Send

Status: 200 OK Size: 128 Bytes Time: 6 ms

```
{
  "product": {
    "id": "a10586c0-f961-4fc9-bae3-56315df03005",
    "name": "Wrist watch",
    "price": 500,
    "quantity": 50,
    "image": "image_url.png"
  }
}
```

Update a product (By ID) And response after sending the request.



PUT http://localhost:7000/product/a10586c0-f961-4fc9-bae3-56315

Status: 200 OK Size: 129 Bytes Time: 9 ms

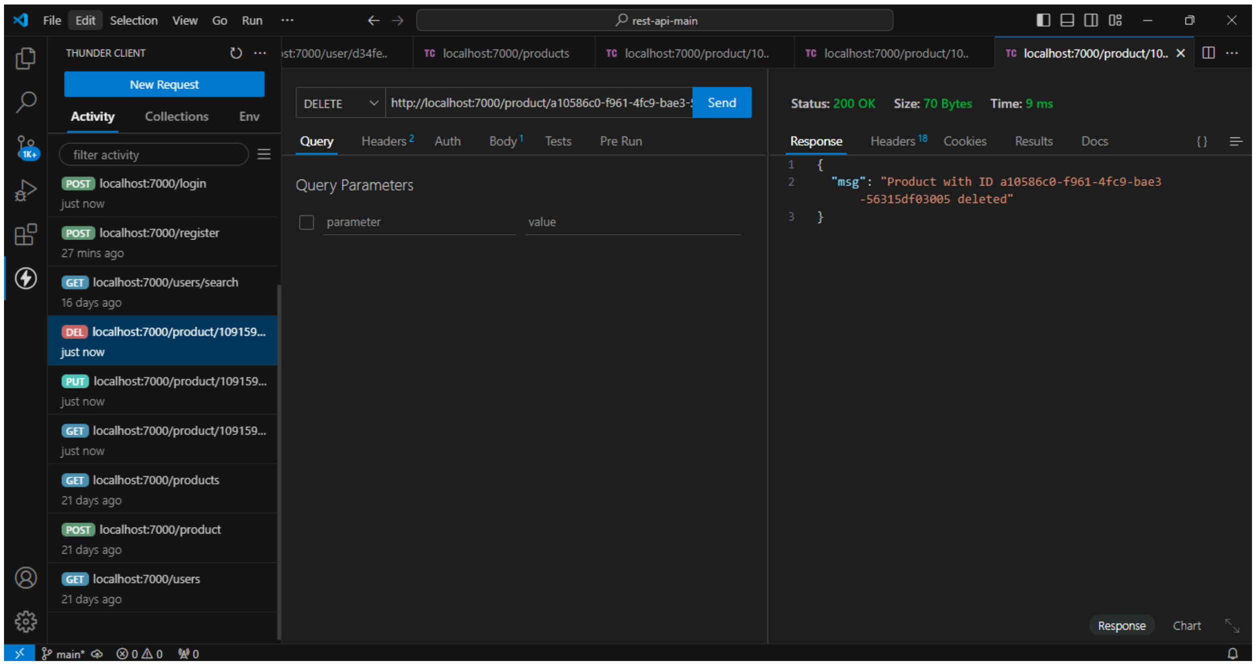
JSON Content

```
{
    "name" : "Rolex",
    "price" : 300,
    "quantity" : 70,
    "image" : "image_url1.png"
}
```

Response

```
{
  "updateProduct": {
    "id": "a10586c0-f961-4fc9-bae3-56315df03005",
    "name": "Rolex",
    "price": 300,
    "quantity": 70,
    "image": "image_url1.png"
  }
}
```

Delete a product (By ID)



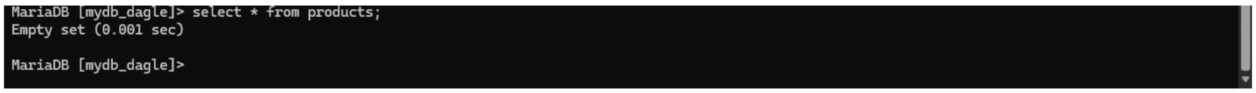
DELETE http://localhost:7000/product/a10586c0-f961-4fc9-bae3-

Status: 200 OK Size: 70 Bytes Time: 9 ms

Response

```
{
  "msg": "Product with ID a10586c0-f961-4fc9-bae3
    -56315df03005 deleted"
}
```

Delete a product (Changes reflected on the database after request has been done)
Note: Use “select * from [tablename]” to show the contents of the table.



```
MariaDB [mydb_dagle]> select * from products;
Empty set (0.001 sec)

MariaDB [mydb_dagle]>
```