YNOV

Documentation de L'api

Trio D'homme forts

Introduction

Notre école nous a chargé de réaliser une application web de e-commerce en ligne, pour sa réalisation nous avons dû créer une API contenant toutes les informations de chaque produit mis en vente et les comptes utilisateurs des clients sur notre application web, dans cette documentation, nous allons vous expliquer de manière détaillée la réalisation de l'API avec les méthode et outil utilisé durant le projet.

Sommaire

- **Création et outils**
  - **Outils**
  - **Ressources**
  - **Création**

- **Fonctionnement**
  - **Structure générale**
  - **Structure détaillée**

- **Liste des Requêtes**
  - **User :**
    - **Post**
    - **Get**
    - **Put**
    - **Del**
  - **Product :**
    - **Post**
    - **Get**
    - **Put**
    - **Del**

Création et outils

Dans cette section, nous détaillerons la genèse de l'api et les outils utilisés pour sa création, ainsi nous verrons les étapes cruciales pour mettre en place l'API pour pouvoir l'utiliser à son plein potentiel .

Outils

L'api est une API HTTP ayant des dépendances minimales, elle utilise .NET core 7, elle est donc portée par du c#, le framework qui porte le projet s'intitule ASP.net core, c'est un framework open source développé par Microsoft. L'éditeur de texte utilisé dans le projet est visual studio code.

Ressources

- https://github.com/dotnet/aspnetcore
- https://learn.microsoft.com/fr-fr/aspnet/core/tutorials/min-web-api?view=aspnetcore-7.0&source=recommendations&tabs=visual-studio-code
- https://dotnet.microsoft.com/en-us/download
- https://github.com/AxelSevenS/sevenet-pham-chorro-ymmersionb3

Création

Avant la création de l'API, il est nécessaire de télécharger .Net core (7 ou équivalent).

Pour notre part nous avons décidé de créer le projet depuis VScode, pour ce faire il est recommandé d'avoir l'extension c# pour visual studio.

Les tests des appels de l'API se font par PostMan

Pour commencer nous avons ouvert le terminal de commande window (cmd) et défini le répertoire du projet, par la suite nous avons tapé dans l'invite de commande ces commandes :

- dotnet new web -o *"le nom du projet"*
- cd *"le nom du projet"*
- code -r ../*"le nom du projet"* (ouvre vscode)
- dotnet dev-certs https --trust

Nous avons approuvé toutes les fenêtres s'ouvrant durant le processus.

Pour le projet nous avons besoin de packages supplémentaires qui serviront à prendre en charge la base de données (NuGet) :

- dotnet add package Microsoft.EntityFrameworkCore.InMemory
- dotnet add package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore

Le raccourci pour lancer le projet est **CTRL + F5**

Fonctionnement

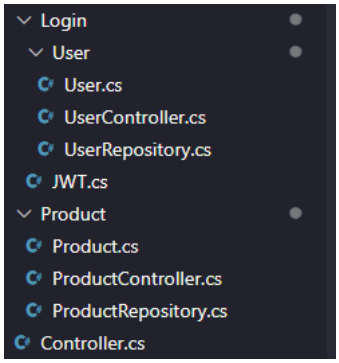
La structure et le fonctionnement de l'API est décrit ici, de plus les moyens nécessaires pour son exploitation optimal.

L'API est hébergé sur l'adresse **Localhost:5001**

Structure générale

La structure de l'api est basée sur plusieurs fichiers pour chaque élément qui nécessite d'une base de données (User ou product) . Chaque élément est lié aux autres et interagit avec ces derniers, ils ont une fonction différente chacun. Ici Element correspond à un terme générique.

Nous avons un contrôleur (ElementController.cs) qui permet l'interaction avec l'api depuis le front, les différentes requêtes sont listées dans ce dernier, ensuite nous avons un fichier (ElementRepository.cs) qui interagit avec la mémoire tampon et la base de données en la modifiant (Un json) pour finir nous avons un modèle de classe de contexte pour structurer les données (Element.cs).



Exemple de structure des fichiers

Structure détaillée

Le système de fichier comme dit précédemment est basé sur un trio de fichiers.
Pour expliquer en détail, nous nous devons de faire un zoom sur chaque fichier et décrire son fonctionnement.

ElementController.cs:

- C'est le fichier appelé lors d'une requête de l'app (GET/POST etc)
- Ce dernier permet d'effectuer des actions tels que l'authentification d'un User ou la mise en ligne d'un produit.
- Le fichier est dépendant de ElementRepository dès que le besoin d'interagir avec les différentes données (Mémoires tampon de l'app ou base de données) se fait ressentir.
- Le fichier dépend de Element.cs .

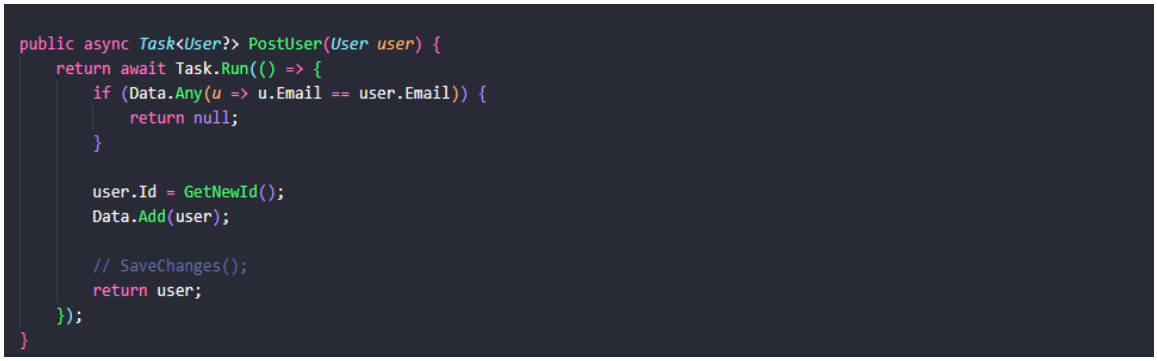
```
[HttpPost]
public async Task<ActionResult<Product>> Post([FromForm] Product Product)
{
    Product.Id = repository.GetNewId();

    await repository.PostProductAsync(Product);

    await PostImages(Product.Id);

    repository.SaveChanges();
    return Ok(Product);
}
```

Dans ce code, on peut observer une fonction correspondant à une requête qui vise à poster un produit dans la base de données, les fonctions appelées dans cette fonction (Requête) correspondent à des fonctions dans ElementRepository qui vont s'occuper de lier la mémoire cache avec la base de donnée.

(Hormis PostImages qui est une autre fonction de ElementController)

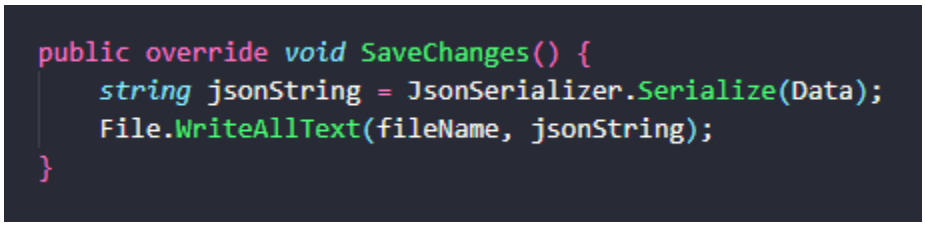
ElementRepository.cs:

- Le fichier est appelé par ElementContoller lorsqu' il est nécessaire d'accéder à la base de données se fait ressentir.
- C'est le fichier qui agit sur la base de données en utilisant les données transmises par ElementController , par ce fait il permet d'écrire/modifier dans la base de données et lier les données transmises dans les requêtes aux éléments de la db.
- Le fichier dépends de Element.cs



```
public async Task<User?> PostUser(User user) {
    return await Task.Run(() => {
        if (Data.Any(u => u.Email == user.Email)) {
            return null;
        }

        user.Id = GetNewId();
        Data.Add(user);

        // SaveChanges();
        return user;
    });
}
```

Dans cette capture on peut observer une fonction servant à ajouter un utilisateur dans la db, cette fonction interagit avec la mémoire tampon de la webapp (Data) et permet de la modifier avant d'écrire les modifications dans le json qui nous sert de base de données.



```
public override void SaveChanges() {
    string jsonString = JsonSerializer.Serialize(Data);
    File.WriteAllText(fileName, jsonString);
}
```

(`SaveChanges()` est la fonction qui permet de save la data (mémoire tampon) dans le json elle est appelée à chaque opération dans ElementController mais elle est situé dans Element.repository)

Element.cs:

- Le fichier permet de structurer les données avant de faire la jointure de la base de données avec la mémoire volatile de la webapp et vice versa.
- Il est nécessaire aussi bien pour ElementController que ElementRepository

```
namespace ApiThf;

public record User
{
    public uint Id { get; set; }
    public string? Email { get; set; }
    public string? Password { get; set; }

}
```

exemple de class pour structurer un utilisateur

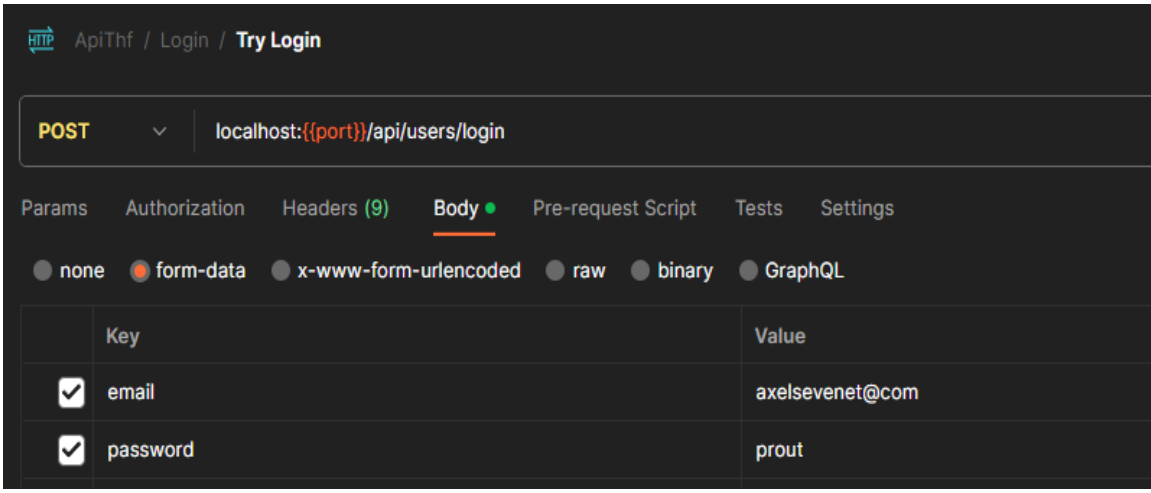
Requêtes:

Dans cette partie nous répertorions toutes les requêtes de l'api actuellement utilisées dans le projet chaque requête sera accompagnée d'une capture d'écran PostMan qui sert à illustrer le propos.

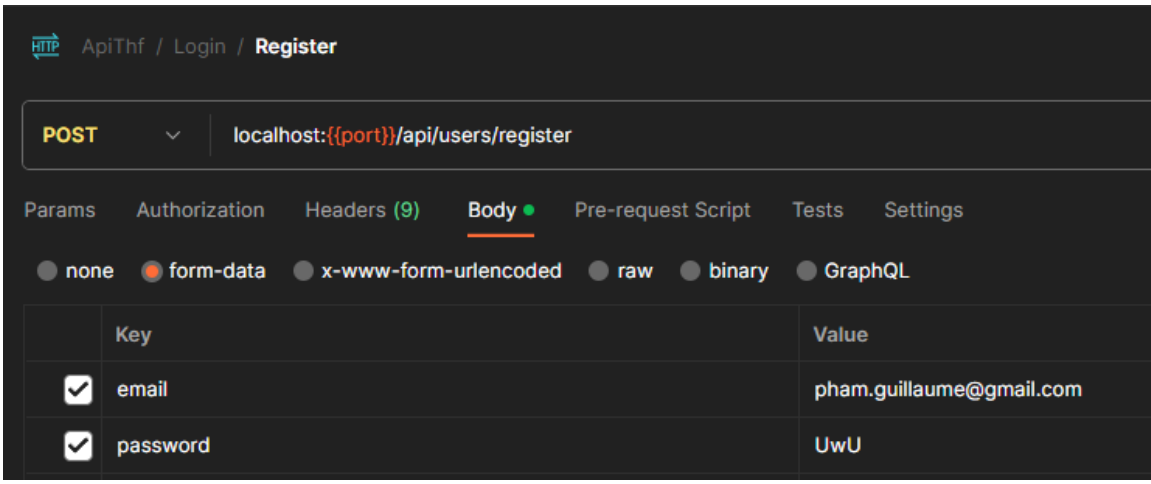
Dans le contexte présent "**Id"** correspond à un terme non défini unique qui correspond à un élément dans la base de données, dans chaque contexte précis le 'id' doit être remplacé par l'utilisateur en fonction de son besoin.

Utilisateurs:

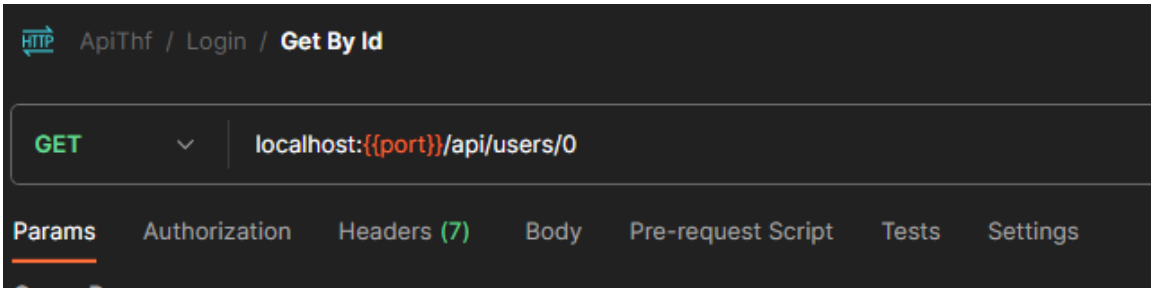
- **POST:**
  - **localhost:5001/api/users/login** permet de vérifier l'authentification d'un user avec un body contenant les informations fournies par un user éventuelle du site .



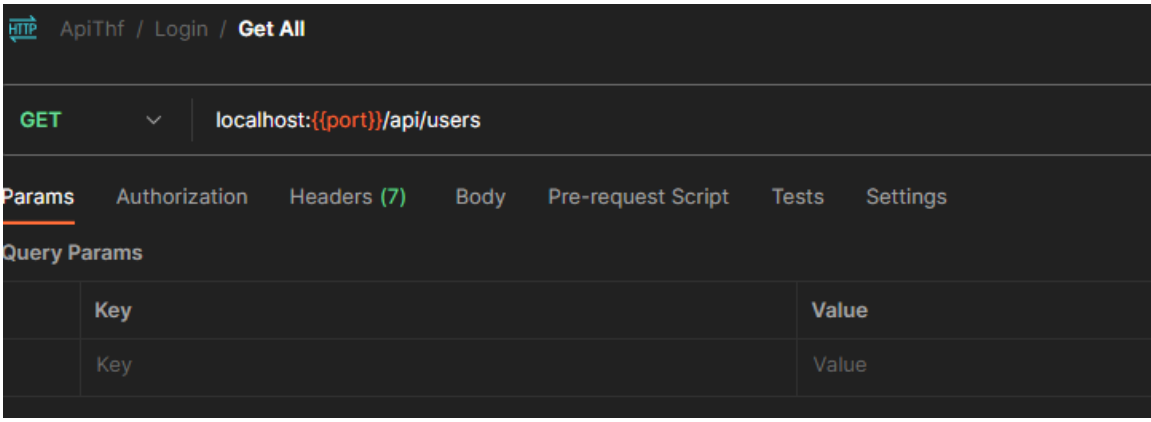
  - **localhost:5001/api/users/register** permet de s'inscrire dans la base d'utilisateurs contenant un body des informations du nouvel utilisateur de la plateforme.



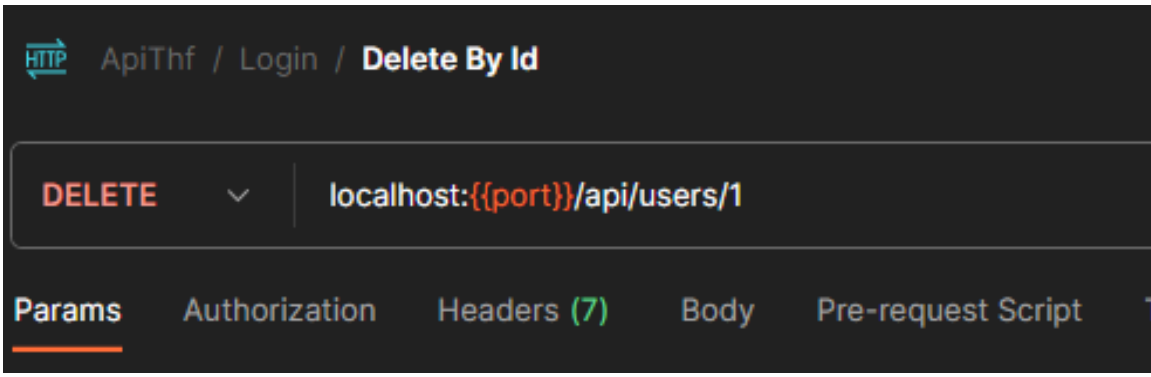
- **GET:**
  - **localhost:5001/api/users/user/"id"** permet de récupérer un utilisateur avec un id spécifique



  - **localhost:5001/api/users** permet de récupérer tous les utilisateurs de la base de données

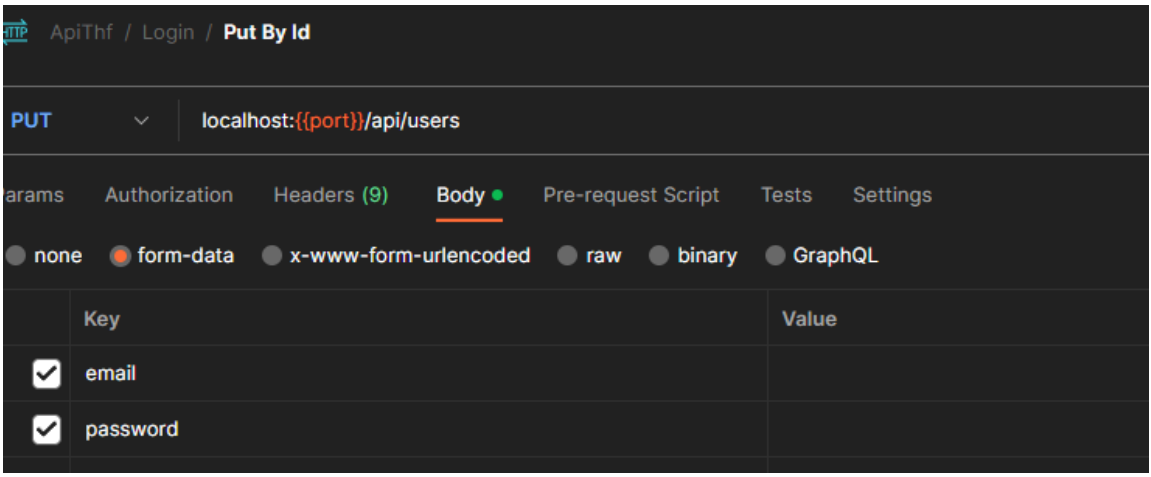


- **Del:**
  - **localhost:5001/api/users/"id"** permet de supprimer un user grâce à son id unique fourni dans la requête .



- **Put:**

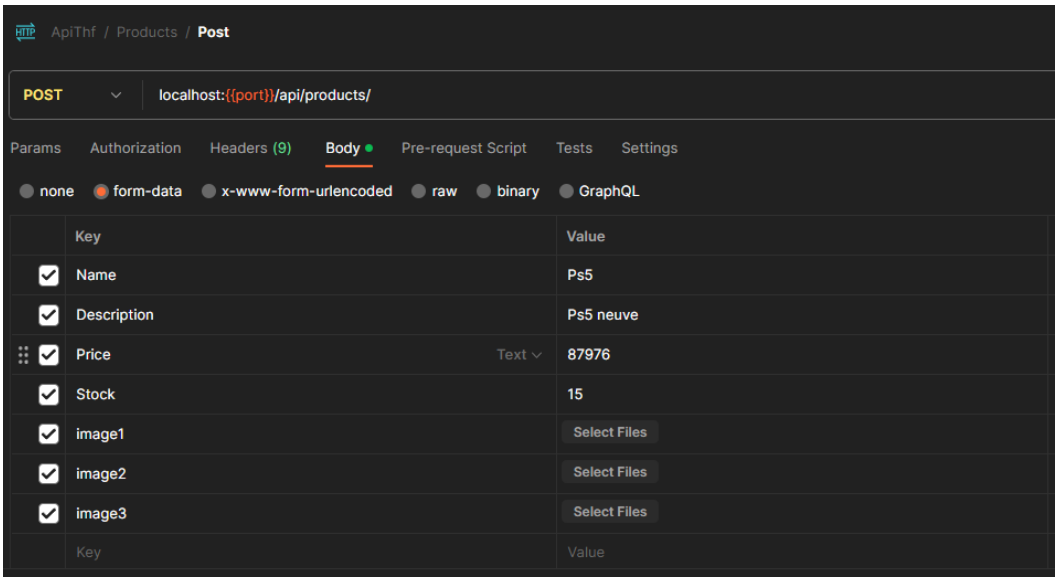
  localhost:5001/api/users/"id" permet de modifier des informations d'un utilisateur avec son id spécifique.



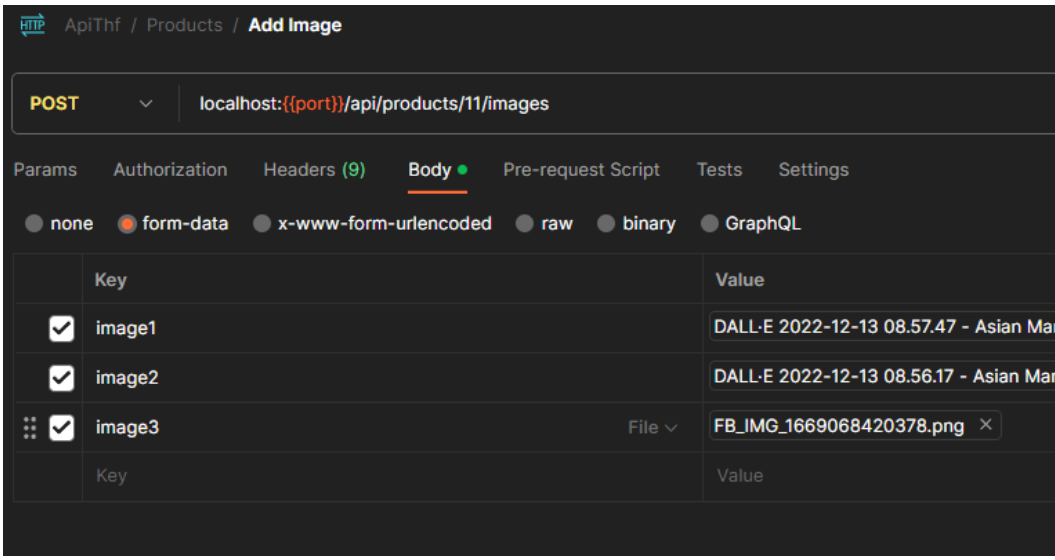
Products:

Les requêtes ici sont très similaires aux précédentes car elles appliquent la même structure que celles des utilisateurs, même si Product a ses spécificités (les images notamment).

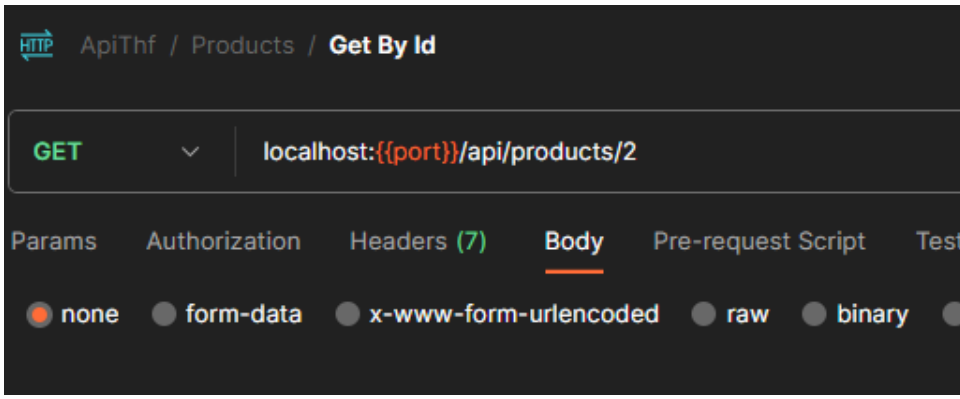
- **POST:**
  - **localhost:5001/api/products/** permet tout simplement d'ajouter un produit dans la base de données.



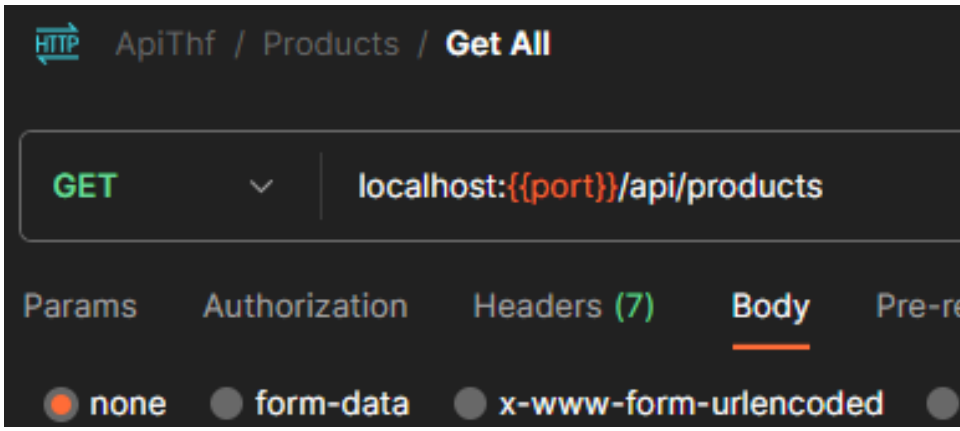
  - **localhost:5001/api/"id"/images** permet d'ajouter une image ou des images pour chaque produit le body correspond ici à chaque photo du produit à insérer dans la base de données



- **GET:**
  - **localhost:5001/api/products/"id"** permet de récupérer un produit avec son id correspondant.



  - **localhost:5001/api/products/** permet de récupérer tous les utilisateurs.



- **PUT:**
  - **localhost:5001/api/products/"id"** permet de modifier ce qu'on veut dans un produit avec son id (hormis les images) en fonction du body fourni.

- **DELETE:**
  - **localhost:5001/api/products/"id"** permet de supprimer un produit avec son id correspondant .