Université Nord Américaine privée
Institut International Technologie

Atelier Framework Spring-05
Service Web REST - Partie01

Objectifs

Développer et tester un Web Service REST avec ARC (Advanced REST Client)

- **Créer un service Web REST**
  - Définir des paths avec des méthodes GET,POST,PUT et DELETE
- **Installer l'extension « Advanced REST Client » pour Google Chrome**
  - Installer et utiliser l'outil ARC
  - Lancer une requête http avec ARC avec la méthode « GET »
  - Configurer ARC pour envoyer des requête avec les méthodes « POST », « PUT » et « DELETE ».

A. Développement d'un service web REST

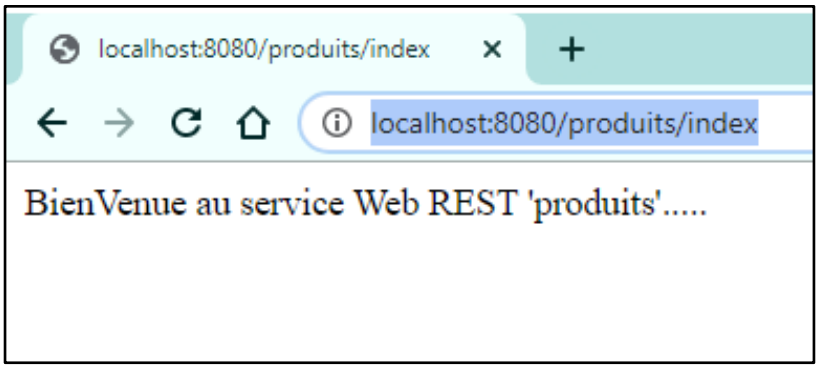
1. Prendre une copie du projet «**jpa-spring-boot-Thymeleaf7»** et le nommer «**jpa-spring-boot-ServiceWeb-REST».**
2. Référencer dans le fichier « **application.properties** » vers une nouvelle base de données «**SpringJpaSW**».
3. Créer, dans le package «**spring.jpa.controller**», la classe qui définit le service web REST et qui est nommée «**ProduitRESTController**» :

```
package spring.jpa.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController // pour déclarer un service web de type REST
@RequestMapping("/produits")  //    http://localhost:8080/produits
public class ProduitRESTController {
```

```
        //  Message d'accueil
        //  http://localhost:8080/produits/index  (GET)
       @RequestMapping(value ="/index" ,method = RequestMethod.GET)
       public String accueil() {
             return "BienVenue au service Web REST 'produits'.....";
       }
}
```

4. Exécuter le projet pour déployer le service web.
5. Aller au navigateur web pour afficher le message d'accueil en utilisant l'url suivant :

http://localhost:8080/produits/index



6. Avec l'annotation « **@RequestMapping** » permet de spécifier :
   - Le path à travers l'attribut « **value** »
   - La méthode **GET** pour la requête http à travers l'attribut « **method** »

- Il est possible d'avoir le même comportement en remplaçant l'annotation «**@RequestMapping**» associée à la méthode « **accueil** » par l'annotation «**@GetMapping**» comme suit :

```
@GetMapping(value ="/index" )
```

- Enregistrer la modification et vérifier le résultat sur le navigateur web.

B. Afficher des données avec la méthode GET au format JSON

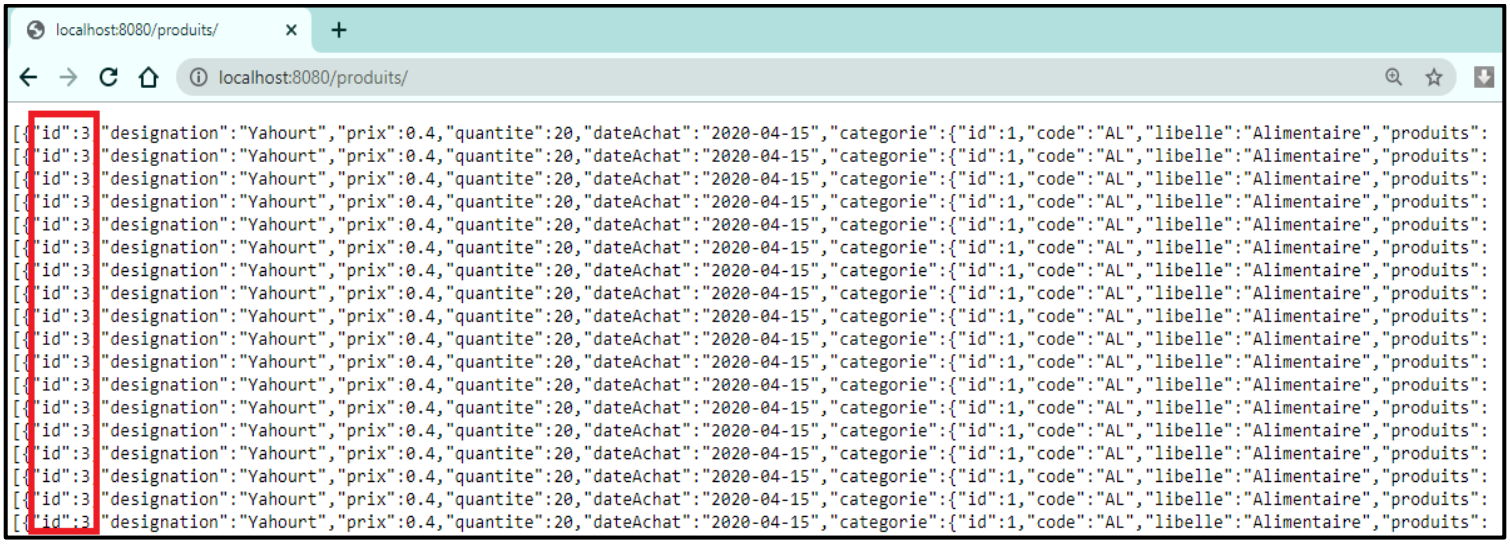
7. Ajouter dans la classe «**ProduitRESTController**» une méthode « **getAllProduits** » qui permet de retourner la liste des produits :
   - Elle porte le path « **/** »
   - Utilise la méthode **GET** du http
   - Utilise la méthode «**findAll**» de l'interface «**ProduitRepository**» pour récupérer tous les produits
   - Et retourne les données sous le format **JSON** (par défaut)

Prendre cette nouvelle version du service web :

```
package spring.jpa.controller;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import spring.jpa.model.Produit;
import spring.jpa.repository.ProduitRepository;

@RestController // pour déclarer un service web de type REST
@RequestMapping("/produits")  //    http://localhost:8080/produits
public class ProduitRESTController {
      @Autowired // pour l'injection de dépendances
      private ProduitRepository produitRepos;

      //  Message d'accueil
      //  http://localhost:8080/produits/index  (GET)
      @GetMapping(value ="/index" )
      public String accueil() {
            return "BienVenue au service Web REST 'produits'.....";
      }

      //Afficher la liste des produits
        //  http://localhost:8080/produits/ (GET)
          @GetMapping(
                        // spécifier le path de la méthode
                        value= "/")
          public  List<Produit> getAllProduits() {
               return produitRepos.findAll();
          }
}
```

8. Enregistrer et lancer l'affichage des produits via le lien suivant :

http://localhost:8080/produits/



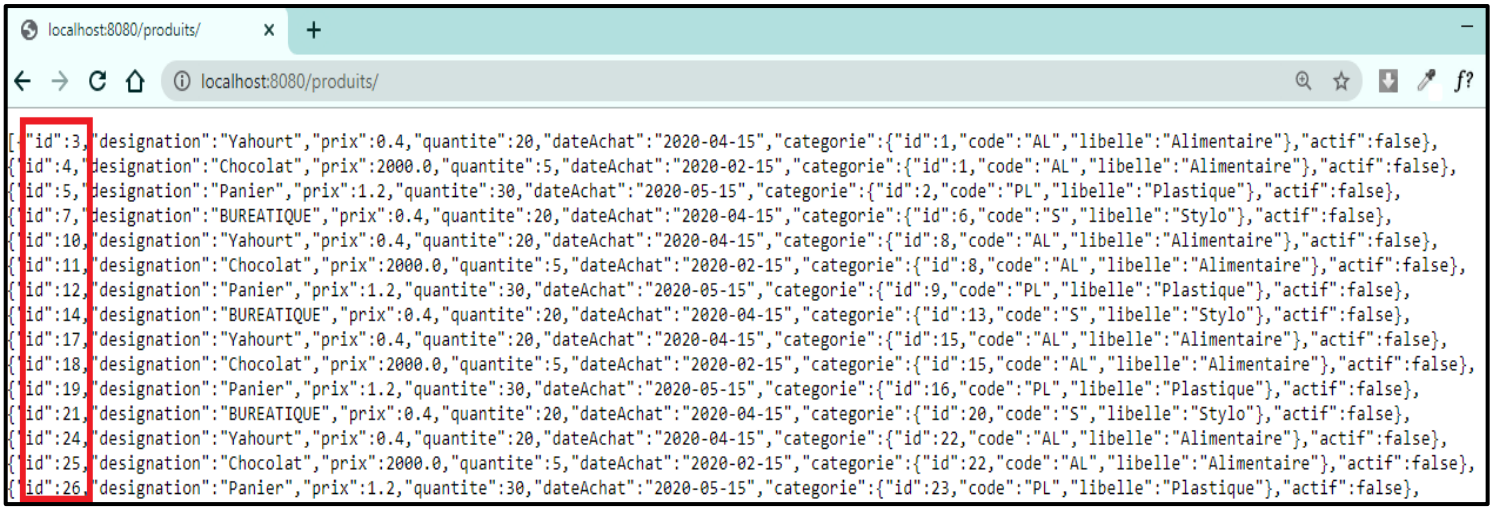
Remarquer un blocage lors de l'affichage dû à la relation bidirectionnelle entre l'entité « **Categorie** » et l'entité « **Produit** ». Ceci provoque des appels récursifs bloquant l'affichage.

9. Pour résoudre ce problème, aller à la classe «**Categorie** » pour indiquer à **spring** d'ignorer les appels aux produits en mode JSON : Ajouter l'annotation « **@JsonIgnore** » juste avant la définition de la méthode « **getProduits** » (Càd ignorer la liste des produits lors de la représentation JSON d'un objet «**Categorie**») :

```
@JsonIgnore
public Collection<Produit> getProduits() {
    return produits;
}
```

10. Enregistrer la modification et aller de nouveau au lien :

http://localhost:8080/produits/



11. Ajouter dans la classe «**ProduitRESTController**» une méthode « **getProduit** » qui permet de retourner un objet de type « Produit » ayant un « id » passé en paramètre :

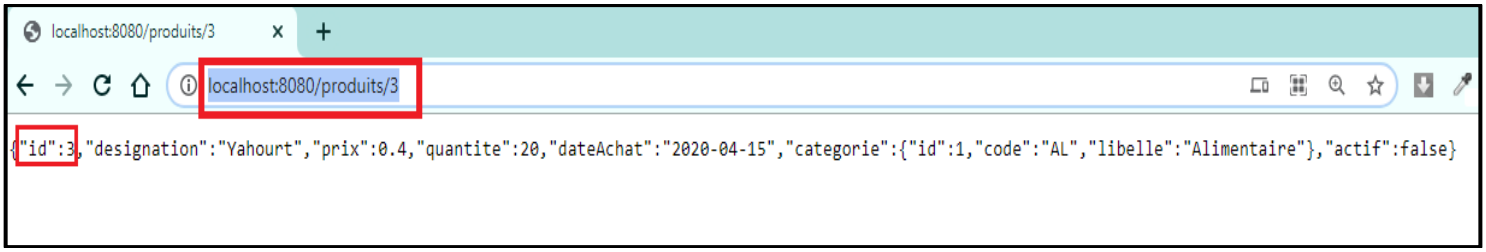
- Elle porte le path « **/{id}** » (Le paramètre « **id** » du path est associé à l'argument « **id** » de la méthode en utilisant l'annotation « **@PathVariable** »)
- Utilise la méthode **GET** du http
- Utilise la méthode «**findById**» de l'interface «**ProduitRepository**» pour récupérer le produit donné.
- Et retourne les données sous le format **JSON** (par défaut)

Voici le code la méthode « **getProduit** » :

```
//  Afficher un produit en spécifiant son 'id'
//  http://localhost:8080/produits/{id} (GET)
@GetMapping(
    value= "/{id}"  )
public Produit getProduit(@PathVariable Long id)
{
    Produit p =produitRepos.findById(id).get();
    return p;
}
```

12. Enregistrer la modification et visualiser l'affichage au format **JSON** (format par défaut) du produit ayant la valeur « **3** » pour l'attribut « **id** » à travers le lien suivant :

http://localhost:8080/produits/3



C. Afficher des données avec la méthode GET au format XML

13. Pour générer les représentations XML des données traitées par le service web REST, il est nécessaire d'ajouter la déclaration de la dépendance « **jackson-dataformat-xml** » dans le fichier « **pom.xml** » :

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

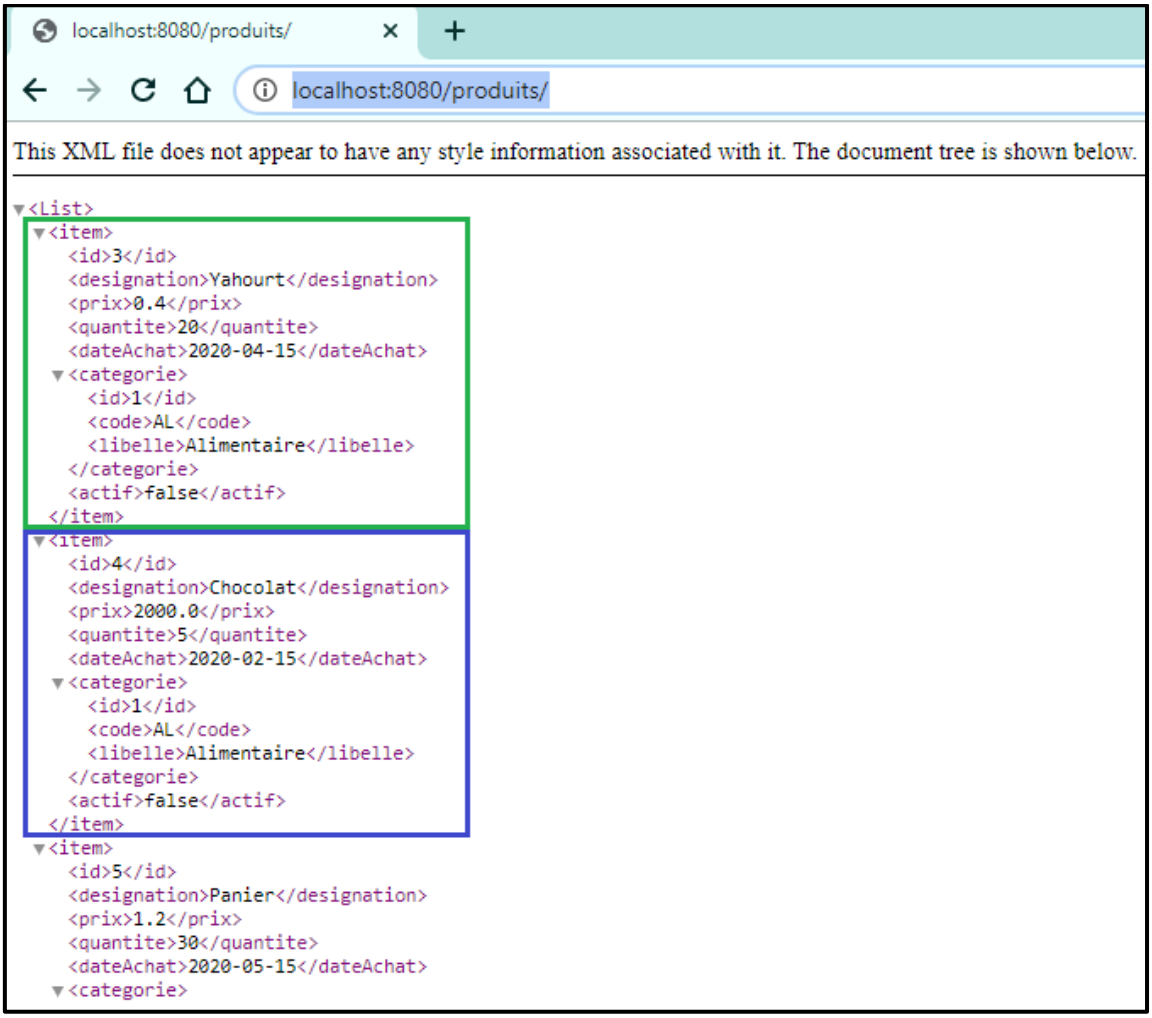
14. Relancer l'exécution du projet pour prendre en considération la nouvelle dépendance.

15. Ensuite spécifier explicitement le format XML pour la représentation des données retournées par le service web (**produces = MediaType.APPLICATION_XML_VALUE**). Changer le code deux méthodes « **getAllProduits** » et « **getProduit** » comme suit :

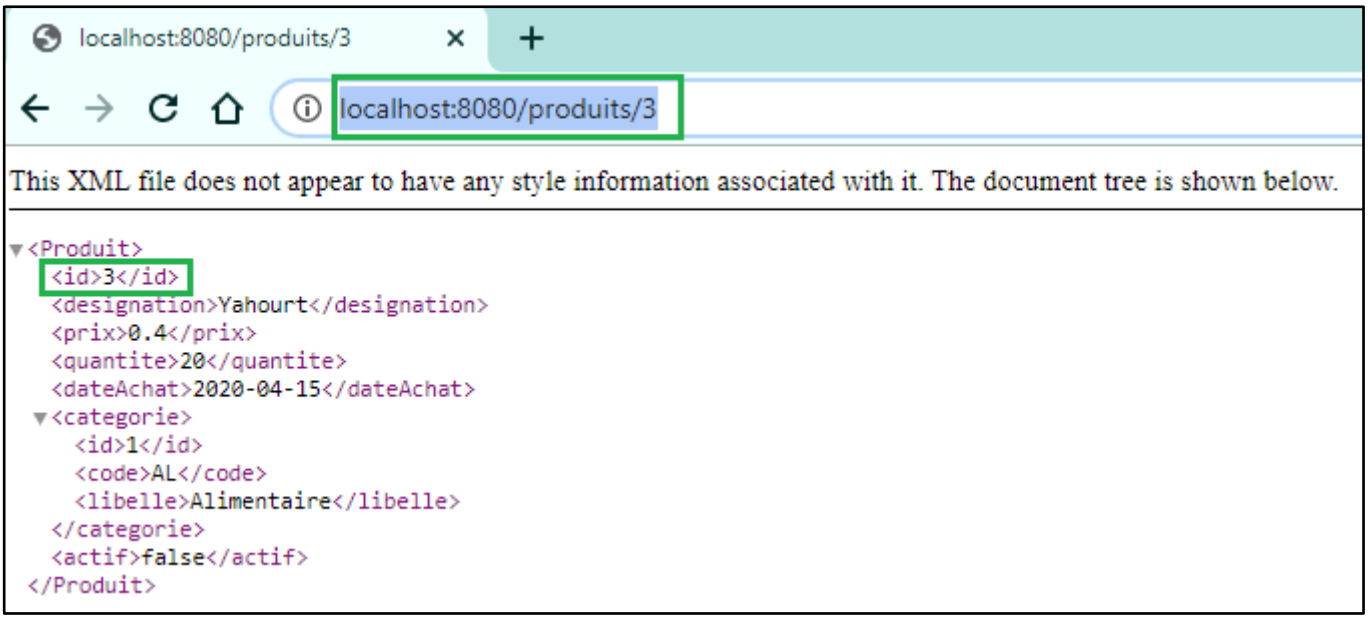
```
    //  Afficher la liste des produits
    //  http://localhost:8080/produits/ (GET)
        @GetMapping(
                        // spécifier le path de la méthode
                        value= "/",
                    // spécifier le format de retour en XML
                        produces = MediaType.APPLICATION_XML_VALUE
                    )
        public  List<Produit> getAllProduits() {
              return produitRepos.findAll();

        }

     //  Afficher un produit en spécifiant son 'id'
     //  http://localhost:8080/produits/{id} (GET)
        @GetMapping(
                        // spécifier le path de la méthode qui englobe un paramètre
                        value= "/{id}" ,
                    // spécifier le format de retour en XML
                        produces = MediaType.APPLICATION_XML_VALUE
                   )
        public Produit getProduit(@PathVariable Long id) {
              Produit p =produitRepos.findById(id).get();
              return p;
        }
```

NB : importer le package « org.springframework.http » pour la classe MediaType

16. Enregistrer la modification et relancer l'affichage de tous les produits (cette fois au format XML) :



17. Afficher le produit ayant la valeur « **3** » pour l'attribut « **id** » (au format XML) :



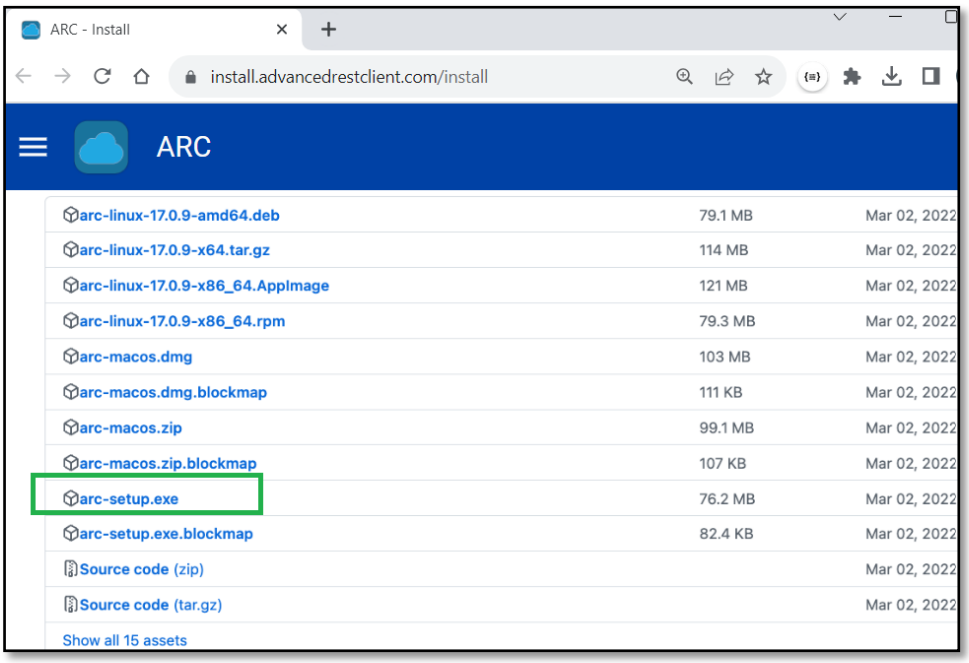
D. Test du service web REST à travers l'outil ARC du Google Chrome

Nous utilisons l'outil **RESTClient** (version Desktop) pour permettre de personnaliser les requêtes envoyées à un service RESTful. Il aide les programmeurs à développer l'application de test RESTful Service pour leurs services.

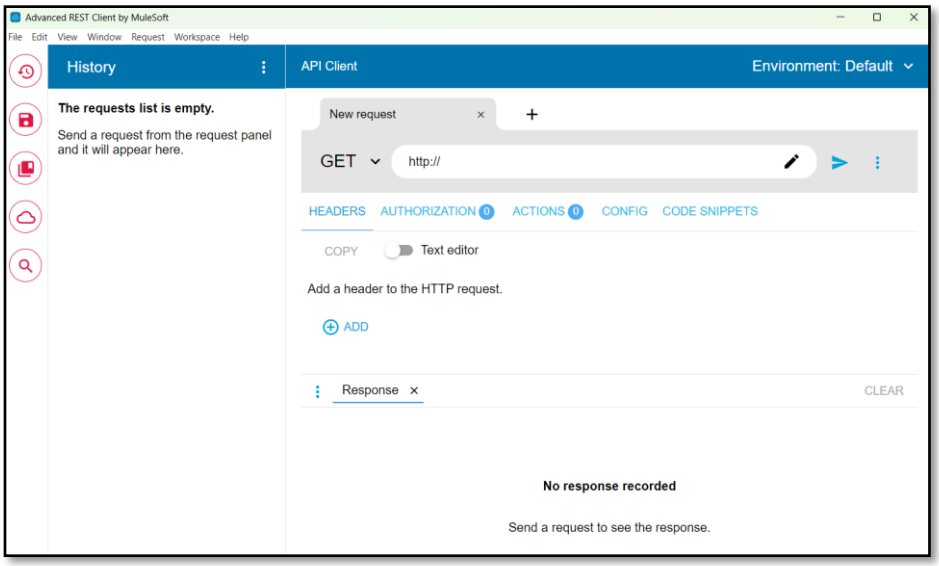
18. Accéder à l'adresse suivante :

https://github.com/advanced-rest-client/arc-electron/releases

19. Choisir la version Windows :

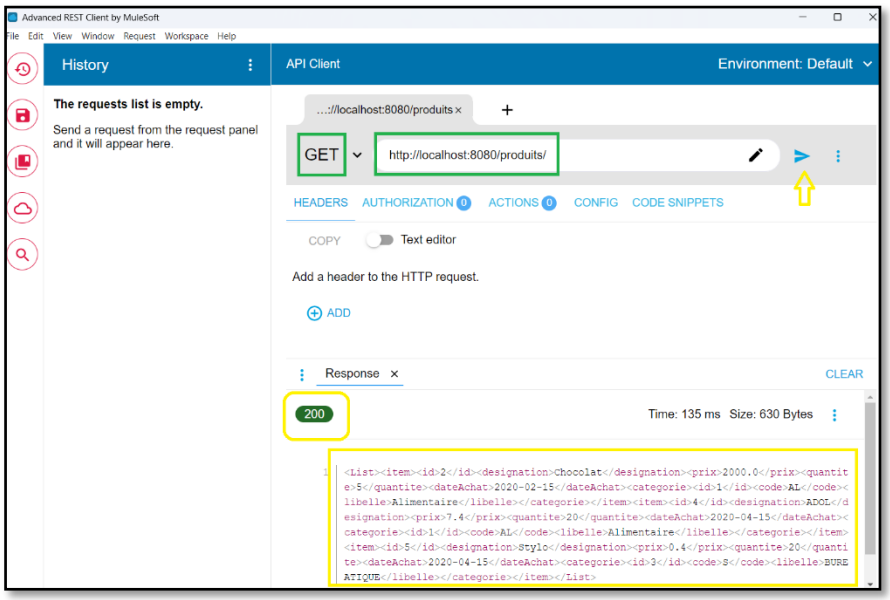


20.Télécharger **l'outil ARC** et l'installer pour avoir cette page d'accueil :



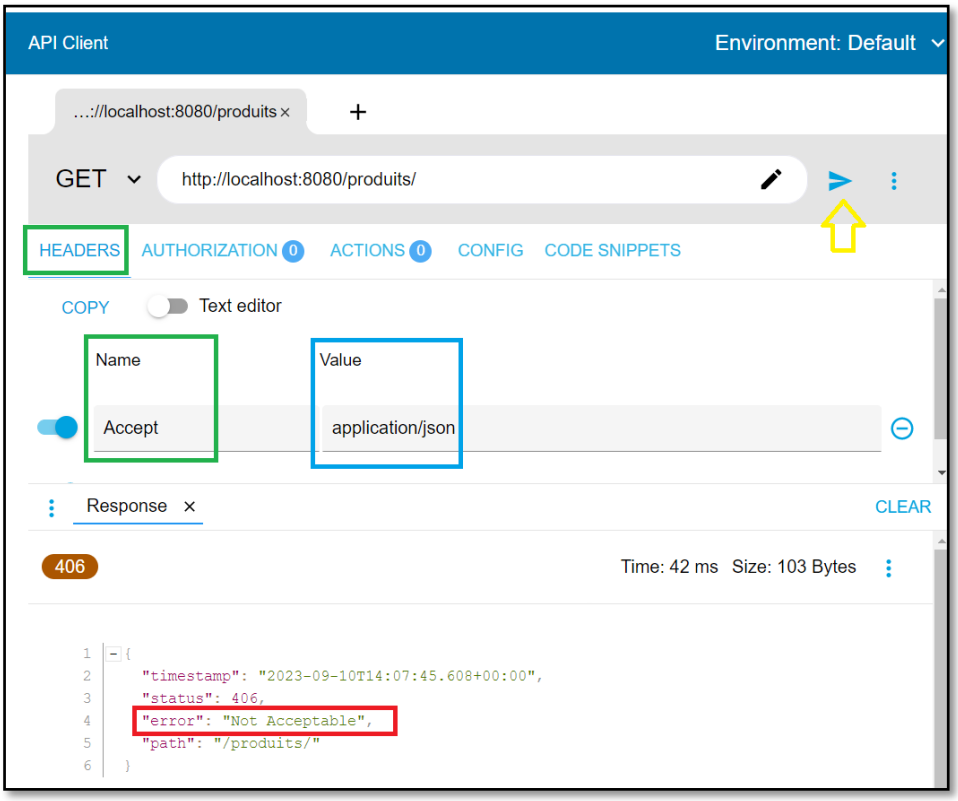
21. Passation de requête avec la méthode **GET**

- Spécifier le type de la méthode : **GET**
- Spécifier Le path de la requête : http://localhost:8080/produits/
- Pui envoyer la requête :



22. Dans la zone « **Headers** », ajouter une propriété à l'entête de la requête pour indiquer le format JSON pour la réponse:

- Header name = **Accept (format des données de la réponse)**
- Header value = **application/JSON**



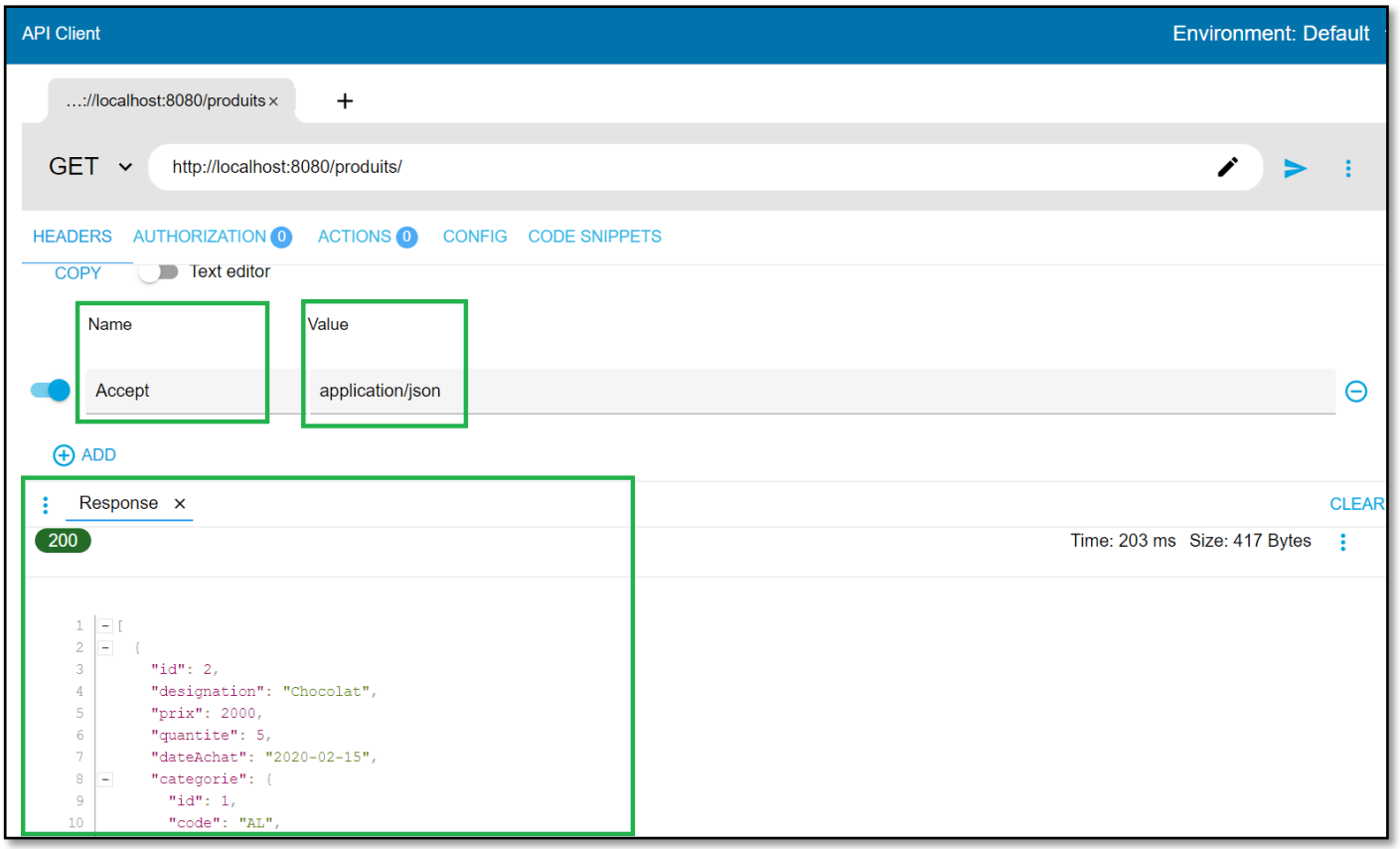
Le lancement de la requête déclenche une erreur 406 dans la réponse du serveur pour indiquer que le format JSON n'est pas supporté.

23. Pour avoir un affichage au format JSON :

- Ajouter, au niveau de l'annotation «**@GetMapping**» de la méthode « **getAllProduits**» le type du média JSON comme suit :

```
produces = {MediaType.APPLICATION_XML_VALUE , MediaType.APPLICATION_JSON_VALUE })
```

- Ainsi, le service web est capable d'envoyer des résultats avec les deux formats XML et JSON.
- Redémarrer Spring.
- Puis relancer la même requête en choisissant le format JSON (dans le header Accept) :



24. Réaliser les modifications nécessaires pour afficher, en utilisant l'outil ARC, le produit ayant la valeur « **2**» comme « **id** » au format XML puis au format **JSON :**

E. Supprimer un produit avec la méthode GET

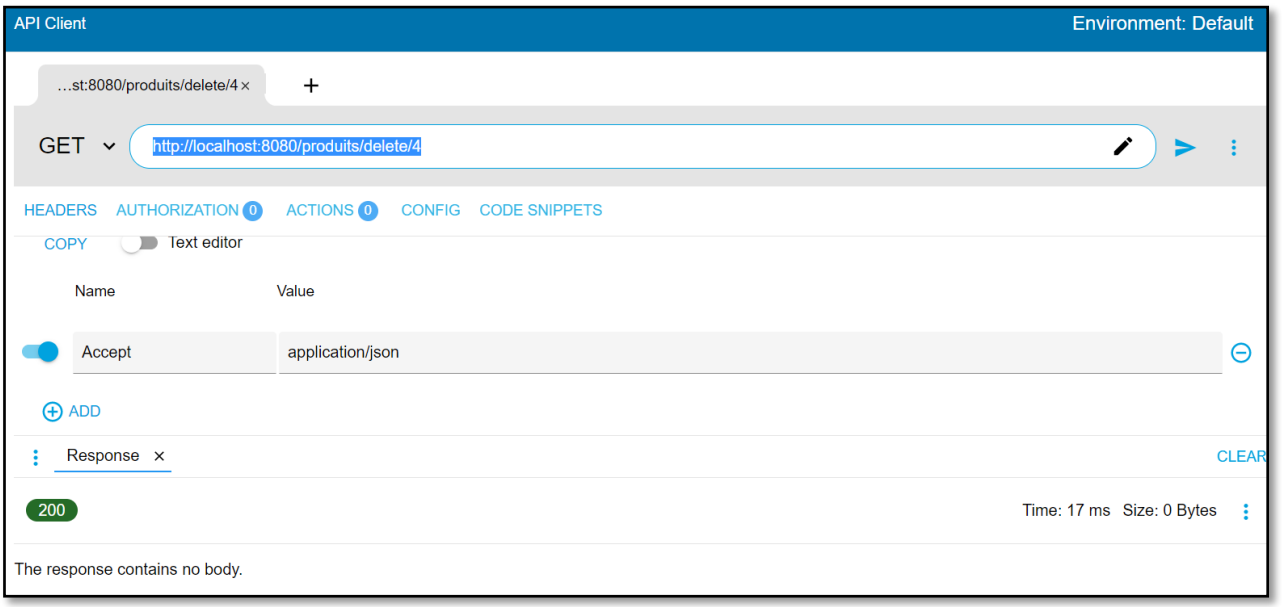
25. Ajouter dans la classe «**ProduitRESTController**» une méthode «**deleteProduit** » qui permet de supprimer un objet de type « **Produit** » ayant un « **id** » passé en paramètre :

- Elle porte le path « **/delete/{id}** » (Le paramètre « **id** » du path est associé à l'argument « **id** » de la méthode en utilisant l'annotation « **@PathVariable** »)
- Utilise la méthode **GET** du http
- Utilise la méthode «**deleteById**» de l'interface «**ProduitRepository**» pour supprimer le produit ayant l''attribut « id » donné en paramètre.
- Voici le code la méthode «**deleteProduit**» :

```
// Supprimer un produit par 'id' avec la méthode 'GET'
//  http://localhost:8080/produits/delete/{id}  (GET)
    @GetMapping(
              // spécifier le path de la méthode
              value = "/delete/{id}")
    public void deleteProduit(@PathVariable Long id)
        {
         produitRepos.deleteById(id);
        }
```

26. Enregistrer la modification et tester la méthode «**deleteProduit**» avec la méthode **GET** à travers le lien suivant (pour supprimer le produit ayant l'id « **4** »):

http://localhost:8080/produits/delete/4



F. Ajouter un nouveau produit en format JSON avec la méthode POST

27. Ajouter dans la classe «**ProduitRESTController**» une méthode «**addProduit**» qui permet d'ajouter un objet de type « **Produit** » passé en paramètre :

- Elle porte le path « **/** »
- Utilise la méthode **POST** du http
- Reçoit un argument de type «**Produit** » précédé de l'annotation «**@RequestBody**» :

```
//  ajouter un produit avec la méthode "POST"
//  http://localhost:8080/produits/   (POST)
        @PostMapping(
                    // spécifier le path de la méthode
                    value = "/"  ,
                    //spécifier le format de retour
                    produces = { MediaType.APPLICATION_JSON_VALUE,
MediaType.APPLICATION_XML_VALUE  }
                    )
        public Produit saveProduit(@RequestBody Produit p)
        {
             return produitRepos.save(p);
        }
```

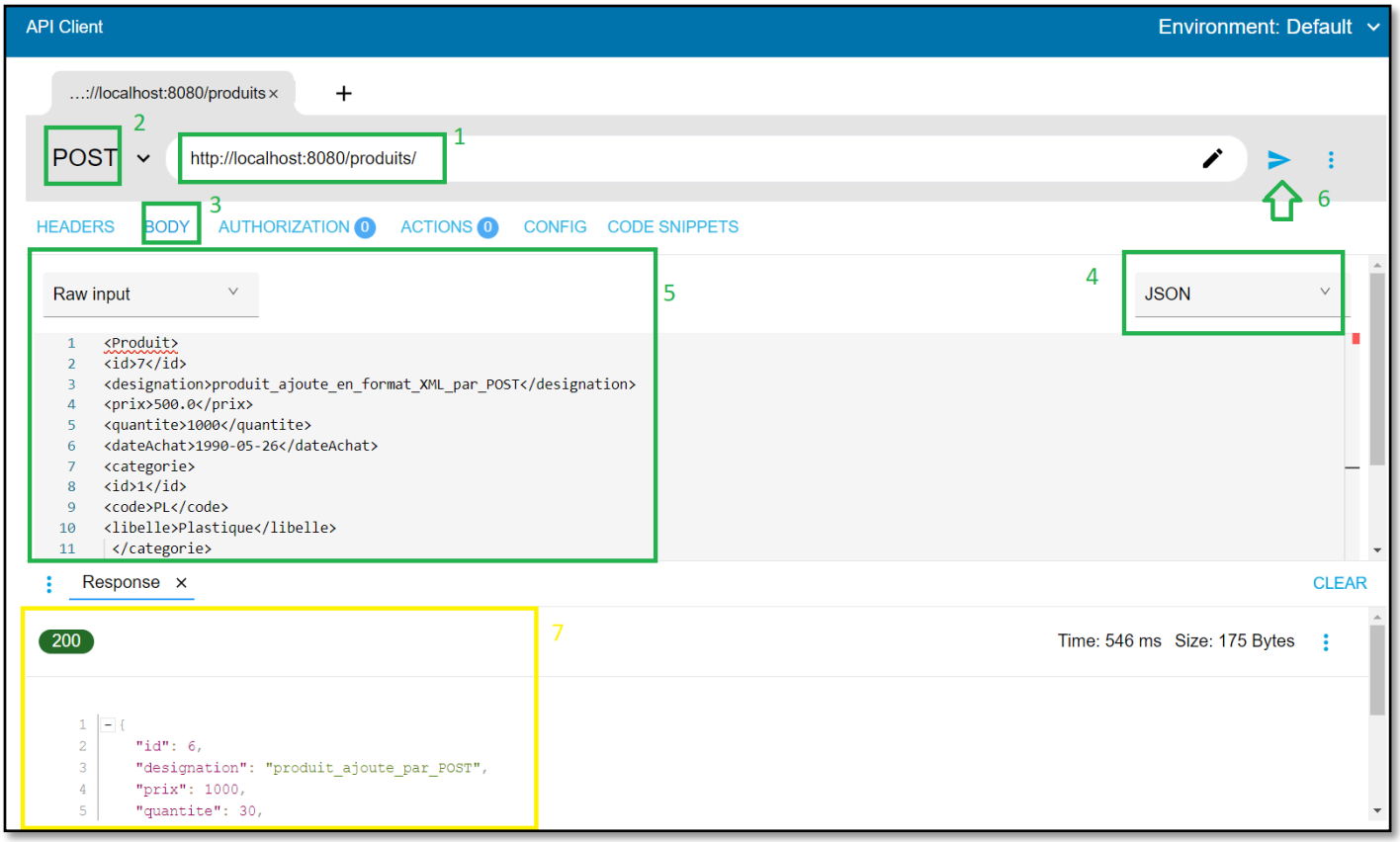
NB : importer le package « org.springframework.web.bind.annotation» pour la classe **RequestBody**

- Redémarrer Spring
- Au niveau de ARC , créer une nouvelle requête pour insérer le produit suivant au format JSON :

```
{ "designation": "produit_ajoute_par_POST",
  "prix": 1000,
  "quantite": 30,
  "dateAchat": "2020-05-15",
  "categorie": {
    "id": 1,
    "code": "PL",
    "libelle": "Plastique"
  }
}
```

- Spécifier l'URL suivante avec la méthode **POST** :

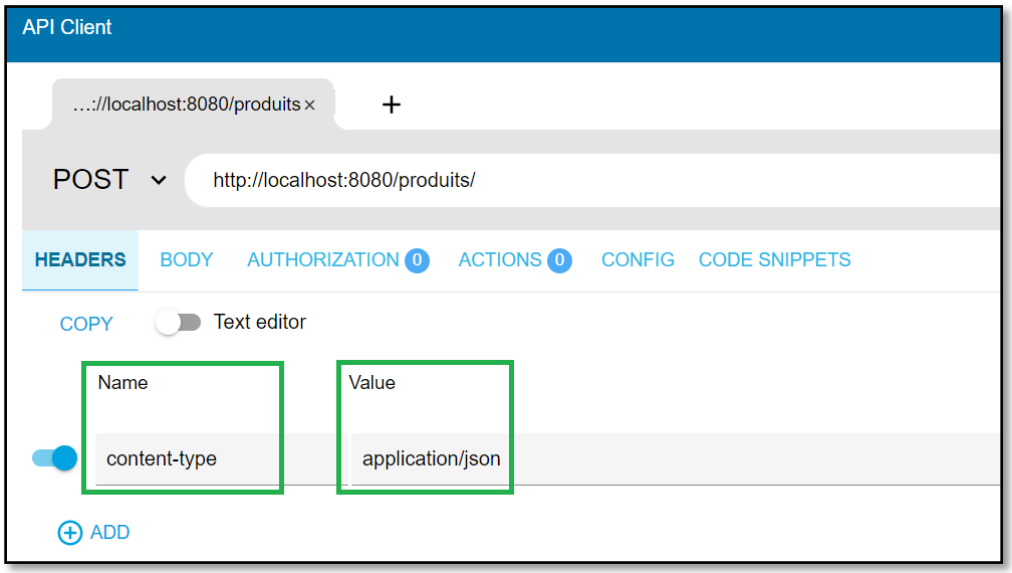
http://localhost:8080/produits/



28. Remarquer que l'outil ARC a ajouté un paramètre d'entête de la requête comme suit :

- Header name = **Content-Type (format des données de la requête)**
- Header value = **application/json**

Ceci est utile pour spécifier au serveur le format des données à recevoir :



29. Remarquer que la réponse est donnée avec la représentation **JSON** (par défaut) en associant au produit une valeur automatique à l'attribut « **id** ». Réaliser les configurations nécessaires pourque la réponse du serveur soit au format XML.

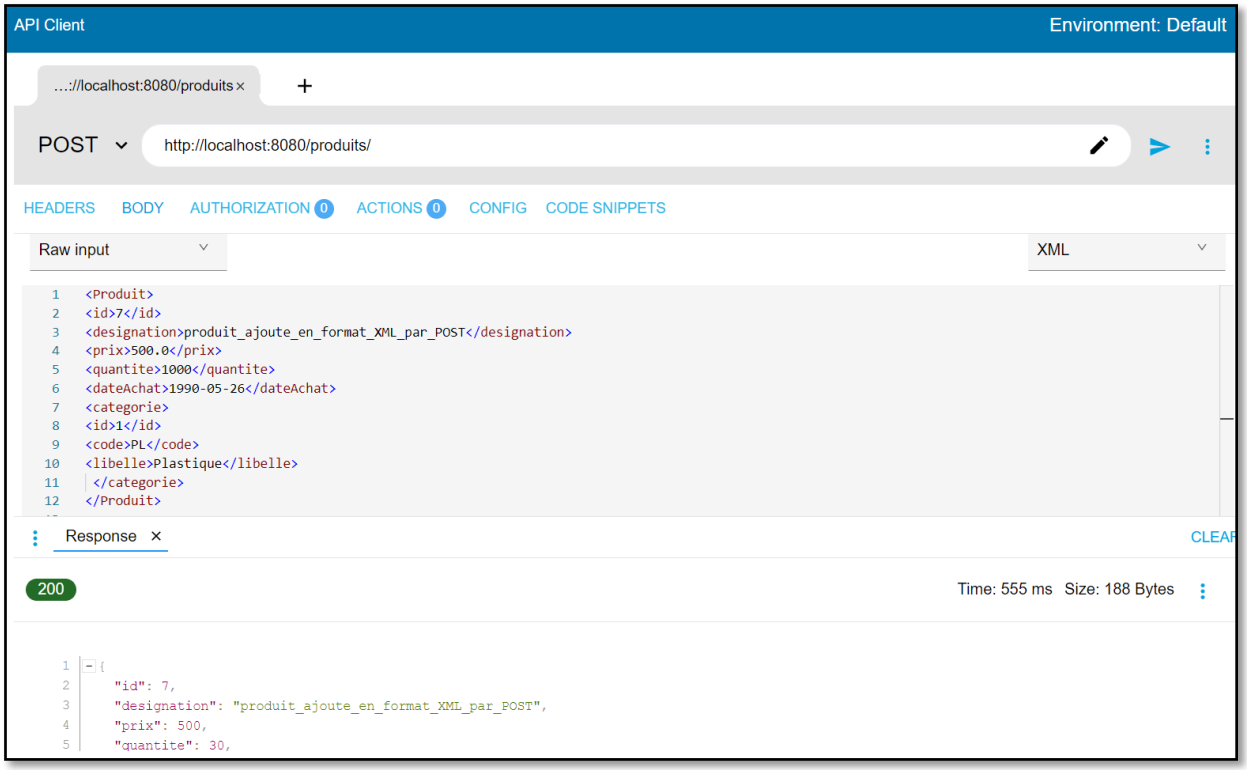
G. Ajouter un nouveau produit en format XML avec la méthode POST

30. Utiliser la même requête pour un nouveau produit en format XML :

- Spécifier la méthode POST
- Saisir l'URL
- Ajouter dans le volet « **Body** » un produit ayant les caractéristiques suivantes :

```
<Produit>
<designation>produit_ajoute_en_format_XML_par_POST</designation>
<prix>500</prix>
<quantite>30</quantite>
<dateAchat>1990-05-26</dateAchat>
<categorie>
<id>1</id>
<code>PL</code>
<libelle>Plastique</libelle>
 </categorie>
 </Produit>
```

- Envoyer la requête pour recevoir le résultat suivant :



31.Réaliser la modification nécessaire pour avoir une réponse en format JSON

H. Modifier un produit existant en format XML avec la méthode PUT

32. Ajouter dans la classe «**ProduitRESTController**» une méthode «**updateProduit**» qui permet de modifier un objet de type « **Produit** » passé en paramètre :

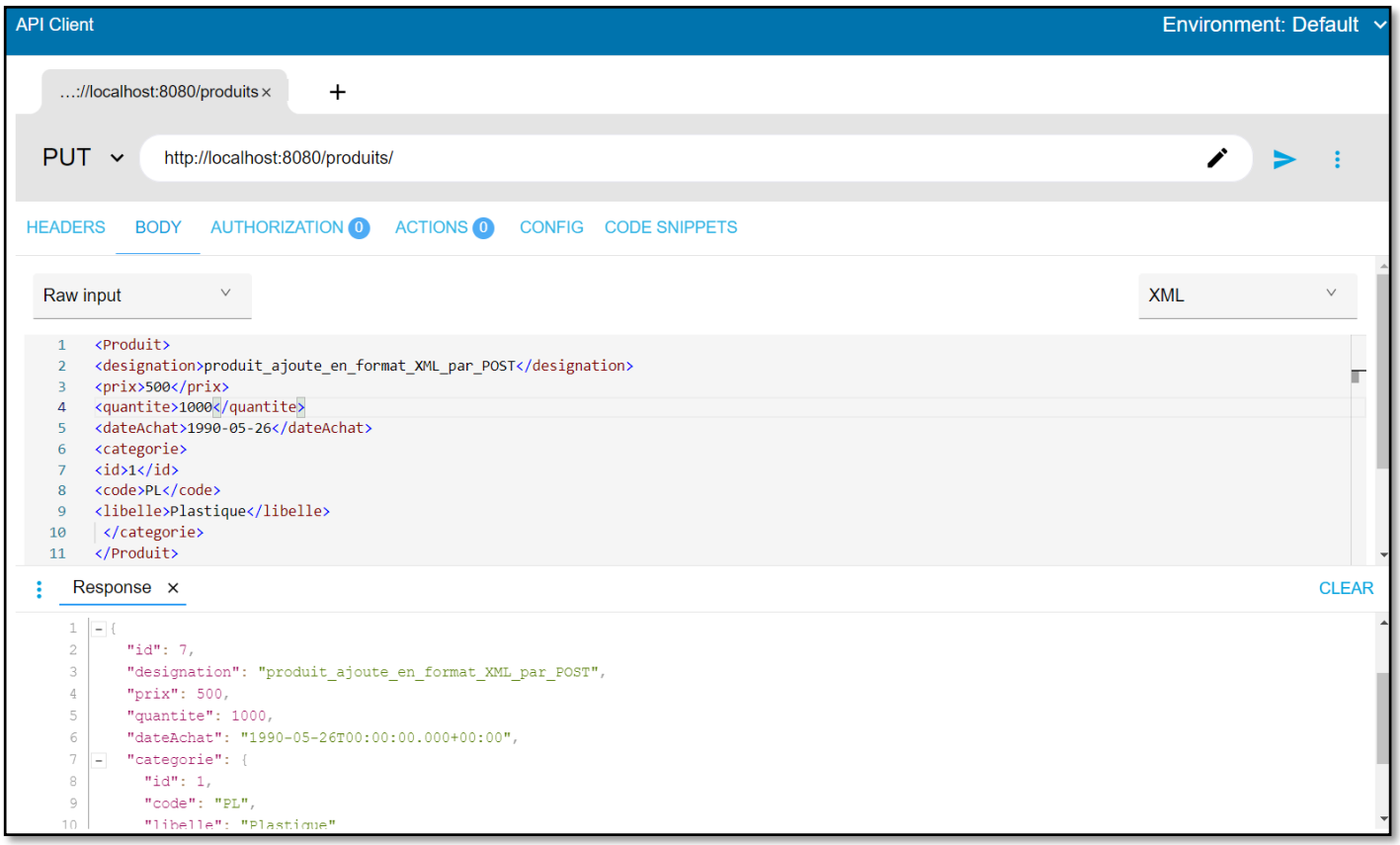
- Elle porte le path « **/** »
- Utilise la méthode **PUT** du http
- Reçoit un argument de type «**Produit** » précédé de l'annotation « **@RequestBody** »
- Utilise la méthode «**save**» de l'interface «**ProduitRepository**» :

```
//  modifier un produit avec la méthode "PUT"
//  http://localhost:8080/produits/   (PUT)
@PutMapping(
      // spécifier le path de la méthode
      value = "/"  ,
      //spécifier le format de retour
    produces = { MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE  }
       )
      public Produit updateProduit(@RequestBody Produit p)
        {
            return produitRepos.save(p);
        }
```

- Enregistrer la modification et redémarrer Spring
- Tester la méthode «**updateProduit**» à travers l'outil ARC :
  - Utiliser la méthode PUT
  - Spécifier l'URL suivante :

    http://localhost:8080/produits/
  - Envoyer dans le body le produit suivant au format XML :

```
<Produit>
<id>7</id>
<designation>produit_ajoute_en_format_XML_par_POST</designation>
<prix>500.0</prix>
<quantite>100</quantite>
<dateAchat>1990-05-26</dateAchat>
<categorie>
<id>1</id>
<code>PL</code>
<libelle>Plastique</libelle>
 </categorie>
 </Produit>
```



2. Réaliser une modification d'un produit donné avec une représentation JSON.

I. Supprimer un produit existant en format XML avec la méthode DELETE

3. Ajouter dans la classe «**ProduitRESTController**» une méthode «**deleteProduit**» qui permet de supprimer un objet de type « **Produit** » passé en paramètre :
   - Elle porte le path « **/** »
   - Utilise la méthode **DELETE** du http
   - Reçoit un argument de type «**Produit** » précédé de l'annotation « **@RequestBody** »
- Utilise la méthode «**delete**» de l'interface «**ProduitRepository**» pour supprimer le produit dernièrement créé en format XML :

```
<Produit>
<id>7</id>
<designation>produit_ajoute_en_format_XML_par_POST</designation>
<prix>500.0</prix>
<quantite>1000</quantite>
<dateAchat>1990-05-26</dateAchat>
<categorie>
<id>1</id>
<code>PL</code>
<libelle>Plastique</libelle>
 </categorie>
</Produit>
```

- Voici le code de la méthode « **deleteProduit** » :

```
// Supprimer un produit  avec la méthode 'DELETE'
//  http://localhost:8080/produits/   (DELETE)
@DeleteMapping(
// spécifier le path de la méthode
    value = "/")
public void deleteProduit(@RequestBody Produit p)
{
    produitRepos.delete(p);
}
```

- Enregistrer la modification et redémarrer Spring
- Passer au test avec ARC :