

## TP : Développer un service web Rest avec Spring Data Rest et Spring BOOT

Architecture des composants d'entreprise

## Table des matières

I.	Objectif du TP.....	2
II.	Prérequis .....	2
III.	C'est quoi Spring Data Rest .....	2
IV.	Développement du WS avec Spring Data Rest .....	4
a.	Objet de l'atelier .....	4
b.	Création du projet Maven .....	4
c.	Le fichier pom.xml .....	7
d.	Le fichier application.properties .....	9
e.	Les classes modèles « Article » et « Catégorie » .....	10
f.	L'interface ArticleDTO.....	12
g.	Les interfaces DAO : ArticleRepository et CatégorieRepository .....	13
h.	La classe MainApplication .....	17
V.	Premier test.....	18
VI.	Tests avec Postman.....	21
i.	Tester la méthode GET pour toute la liste .....	21
j.	Tester la méthode GET pour un article donnée et une catégorie donnée .....	22
k.	Tester la méthode POST pour un article donnée et une catégorie donnée .....	22
l.	Tester la méthode GET pour consulter un article par son ID en utilisant la projection .....	24
m.	Tester le service byCategorie .....	25
n.	Tester la méthode PUT pour associer une catégorie à un article .....	25
VII.	Configuration de l'api OpenAPI et Swagger.....	26
1.	Configuration de l'api OpenAPI.....	26
2.	Intégration avec Swagger UI.....	28
3.	Tester la méthode Get avec Swagger UI .....	29
4.	Tester la méthode Put avec Swagger UI .....	30
	Conclusion .....	33

## I. Objectif du TP

- Utiliser le starter *Spring Data Rest* pour exposer une modèle via l'architecture Rest.
- Utiliser Spring Data JPA pour générer les méthodes CRUD.
- Utiliser la base de données H2.
- Configurer l'api *OpenAPI version 3* pour rendre votre service web compréhensible à tout le monde.
- Utiliser l'interface de Swagger pour tester votre service web.

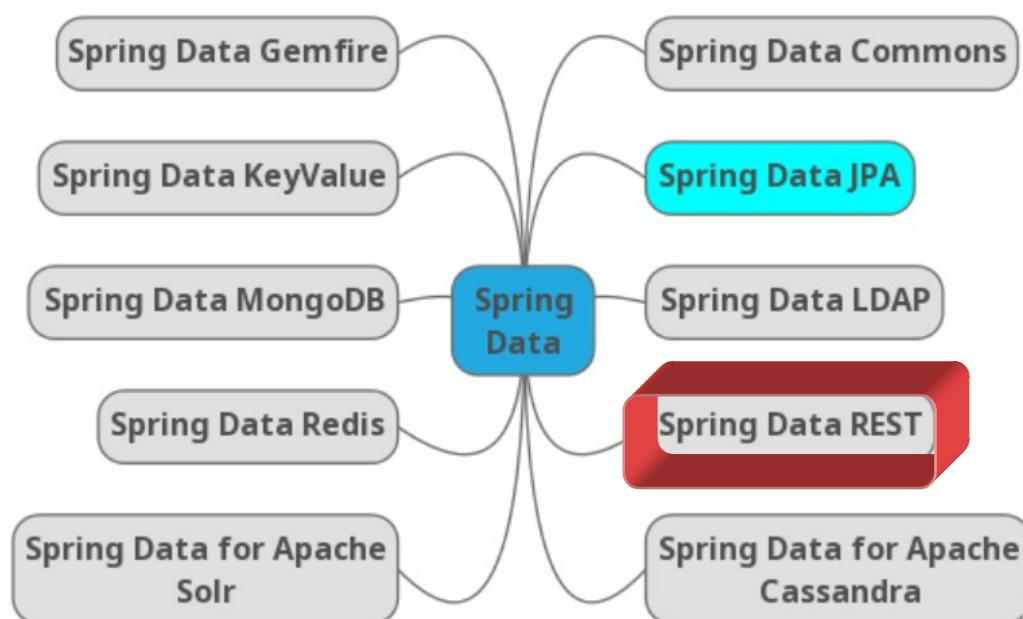
## II. Prérequis

- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.
- Postman pour effectuer les tests.

**NB :** Ce TP a été réalisé avec IntelliJ IDEA 2023.2.3 (Ultimate Edition).

## III. C'est quoi Spring Data Rest

- *Spring Data REST* fait partie du projet *Spring Data « Umbrella Project »* et facilite la création des services Web « hypermedia-driven REST ».

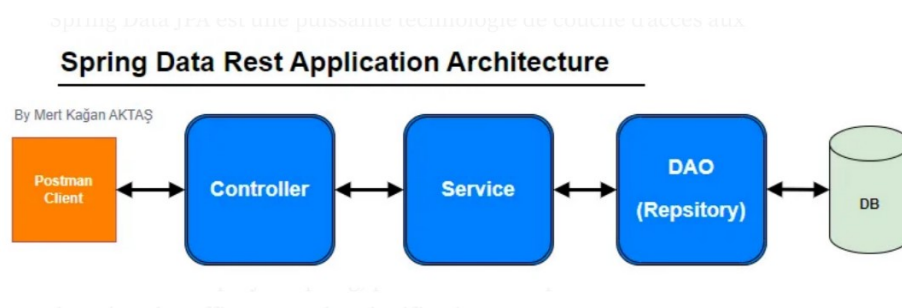


- Spring Data REST s'appuie sur les référentiels Spring Data, analyse le modèle de domaine de votre application et expose les ressources HTTP pour les agrégats contenus dans le modèle.
- Parmi les fonctionnalités fournies par Spring Data Rest :
  - Expose une API REST pour votre classe modèle en utilisant HAL (*Json Hypertext Application Language*) comme *media type*.

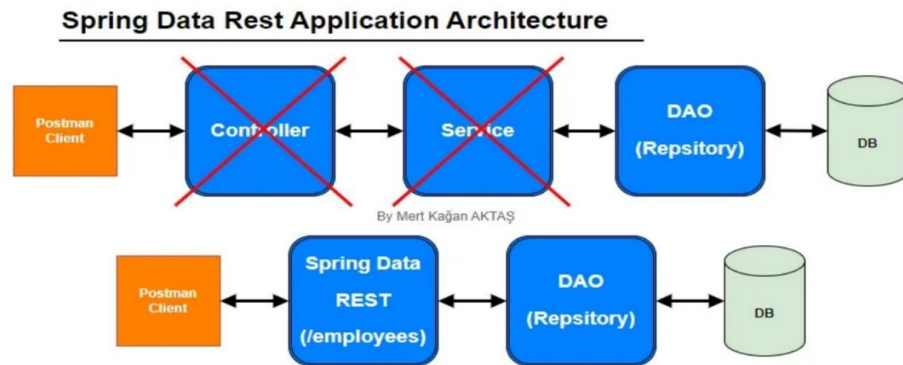
- Resource Discoverability (ou découverte des ressources) : Par défaut, Spring Data REST utilise **HAL** pour restituer les réponses. HAL définit les liens à contenir dans une propriété du document renvoyé.
- Expose les collections et les associations représentant votre modèle.
- Supporte la pagination via des liens de navigation.
- Supporte actuellement JPA, MongoDB, Neo4j et Cassandra.
- Le site officiel de *Spring Data Rest* est <https://spring.io/projects/spring-data-rest>.
  - Plusieurs exemples sont disponibles sur le site officiel : <https://spring.io/projects/spring-data-rest#samples>.

#### IV. Pourquoi Spring Data Rest ?

- Dans une application typique basée sur Spring, nous créons une architecture à trois couches comme ci-dessous :



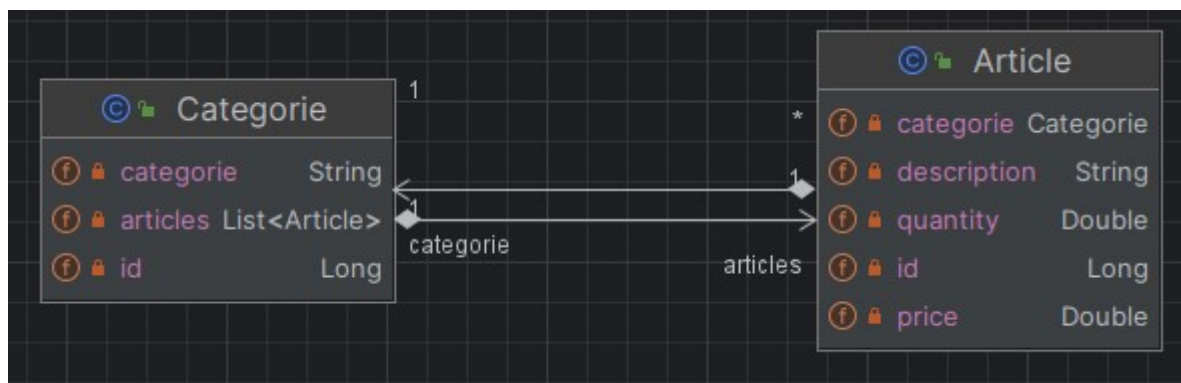
- Quel est le problème ?  
 Si nous voulons créer une API REST CRUD pour une entité, nous devons créer une classe de contrôleur et écrire manuellement les API REST (créer, mettre à jour, supprimer, obtenir, paginer, trier, etc.).  
 Si nous voulons créer une API REST CRUD pour une autre entité : par exemple , Département , Projet , Société , Utilisateur , etc. Ensuite, nous devons créer un contrôleur pour toutes ces entités et créer des API REST.
- ➔ Constat : **Nous répétons encore le même code....**
- Solution :  
 Spring Data REST est la solution. il utilise des interfaces qui étendent **JpaRepository** et fournit des API REST CRUD pour les entités avec un minimum d'efforts. Cela permet de minimiser le code de la couche contrôleur standard.



## V. Développement du WS avec Spring Data Rest

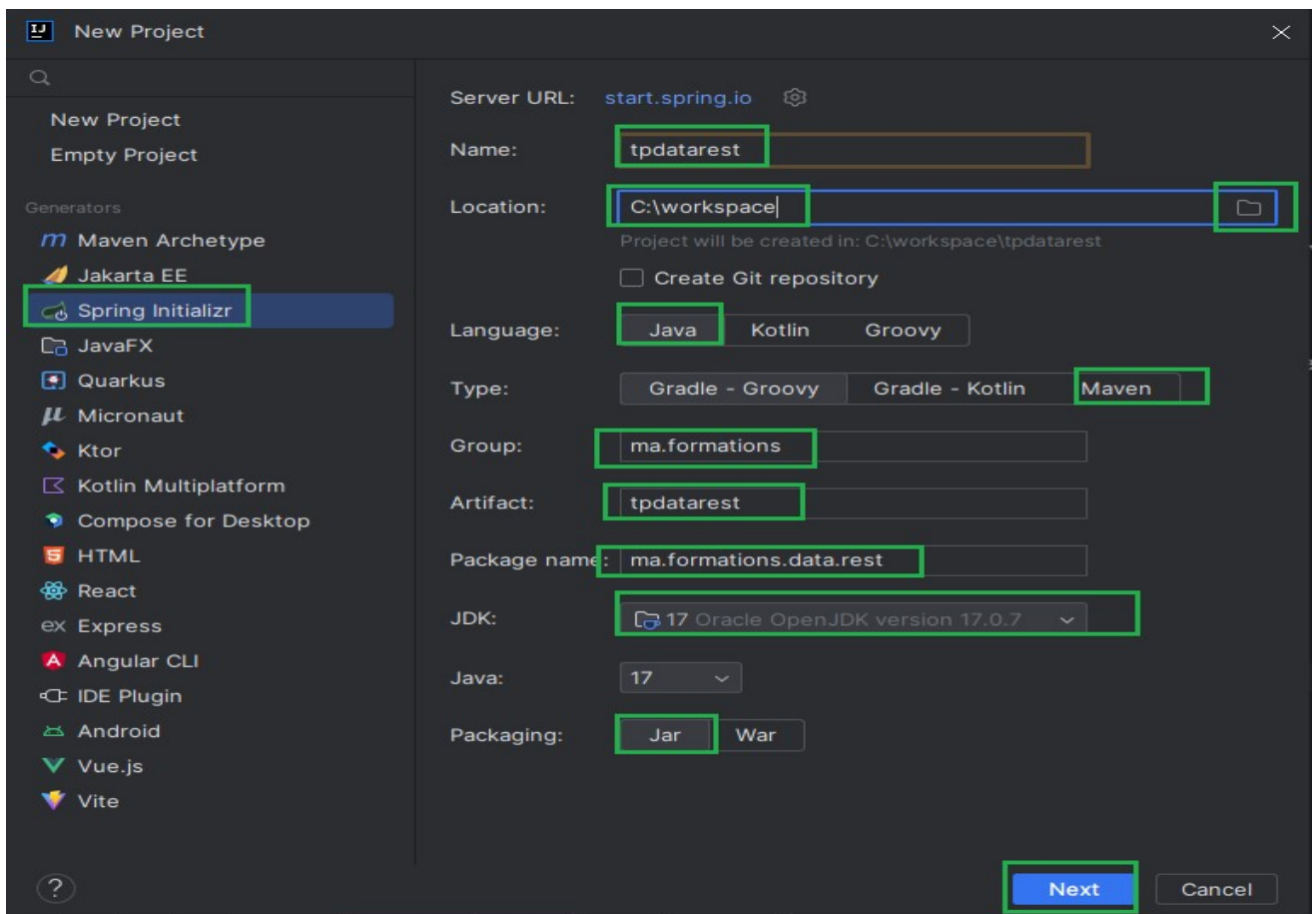
### a. Objet de l'atelier

- Nous allons développer un service web qui permet d'effectuer des opérations CRUD sur les articles contenus dans une base de données H2. Le diagramme de classe est le suivant :

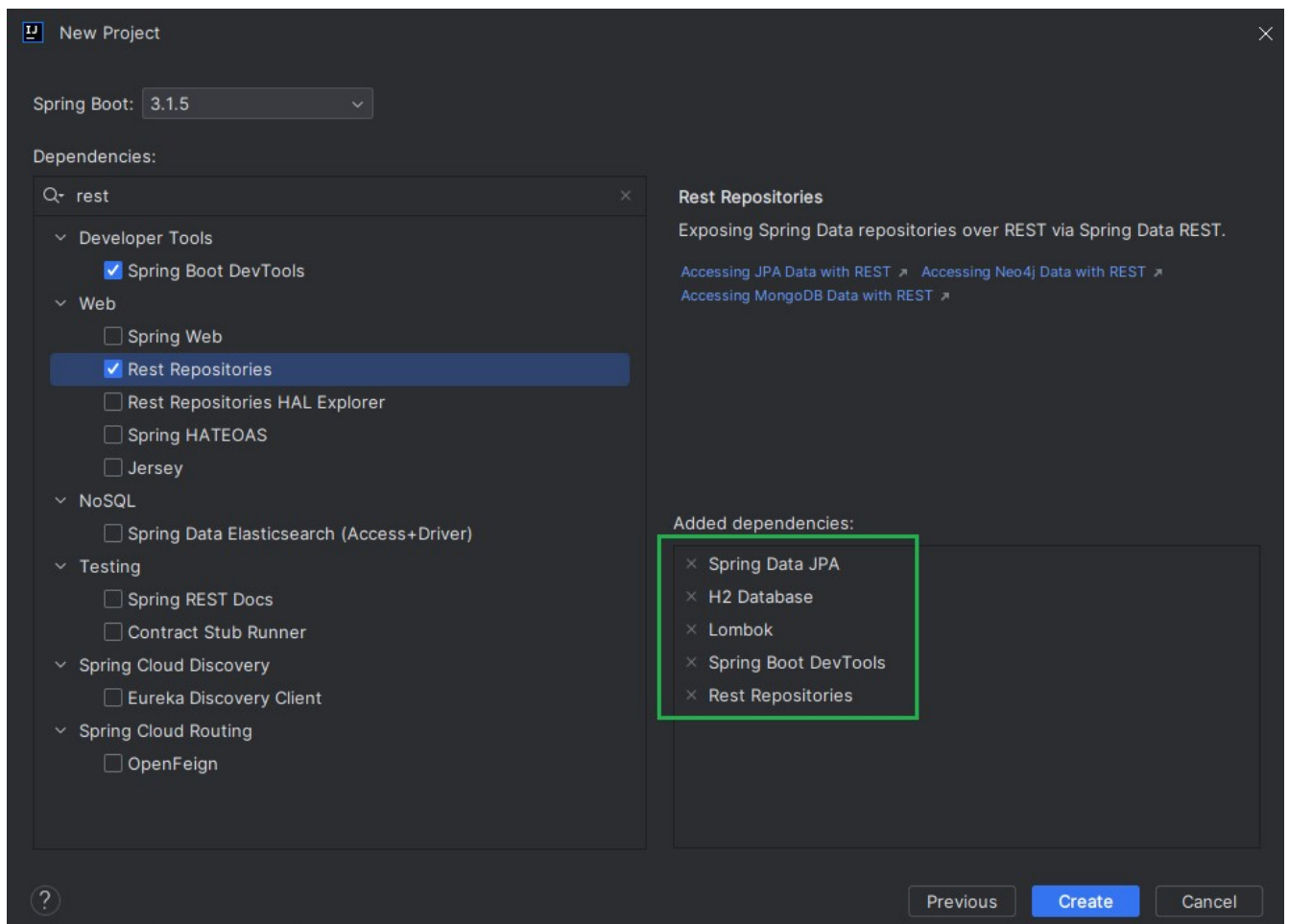


### b. Création du projet Maven

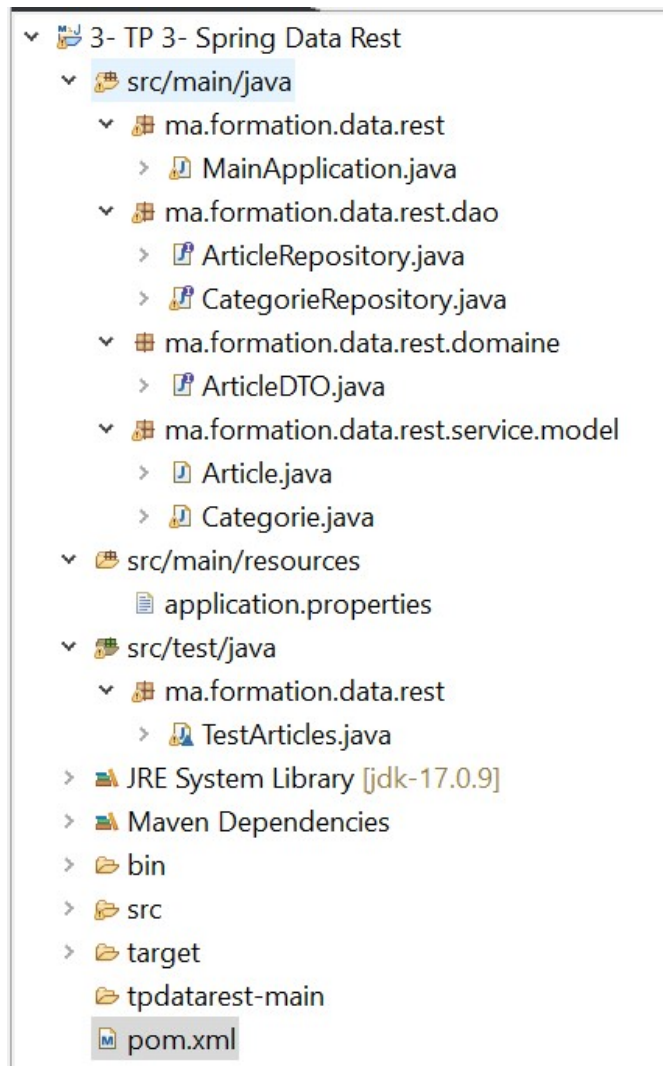
- Créer un nouveau projet comme le montre la fenêtre suivante :



1. Entrer le nom de votre projet.
2. Dans Location, préciser le dossier dans lequel le projet sera créé (par exemple : c:\workspace).
3. Dans Language, cliquer sur Java.
4. Dans Type, cliquer sur Maven.
5. Entrer le Group et l'Artifact.
6. Entrer le nom du package.
7. Choisir JDK 17.
8. Dans Packaging, cliquer sur Jar. Enfin, cliquer sur Next. La fenêtre suivante s'affiche :



- Ajouter les dépendances suivantes : Spring Data JPA, H2 database, Lombok, Spring Boot DevTools et Rest Repositories et cliquer ensuite sur Create.
- Ou bien via Eclipse :



### c. Le fichier pom.xml

- Vérifier que votre fichier pom.xml généré contient les dépendances suivantes :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
```



```

    <scope>runtime</scope>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>

```

#### **Explications :**

- La dépendance :

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>

```

- ✓ Il s'agit du starter de *Spring Data Rest*. Ce dernier permet d'ajouter toutes les dépendances nécessaires et permet également d'auto-configurer le Framework *Spring Data Rest*.

- La dépendance :

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

```

- ✓ Il s'agit du starter de *Spring Data JPA*. Ce dernier permet de d'ajouter toutes les dépendances nécessaires et permet également d'auto-configurer l'api *JPA*. Le starter *spring-boot-starter-data-jpa* utilise comme implémentation par défaut de l'api *JPA*, le Framework *Hibernate*. Pour rappel, l'api *JPA* est implémentée par 03 Framework : *Hibernate*, *TopLink* et *EclipseLink*.

- La dépendance :

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

- ✓ Il s'agit de la base de données virtuelle H2. H2 est une base de données embarquée qui se démarre et s'arrête avec l'application.

#### d. Le fichier `application.properties`

- Copier les lignes suivantes au niveau du fichier `application.properties` :

```
# The name of the H2 database :
spring.datasource.url=jdbc:h2:mem:testdb
# The H2 Driver :
spring.datasource.driverClassName=org.h2.Driver
spring.data.jpa.repositories.bootstrap-mode=default
spring.datasource.username=sa
spring.datasource.password=
# The Dialect : java => SQL compatible with H2
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
# automatic creation and modification of tables
spring.jpa.hibernate.ddl-auto=update
# Activate the H2 console :
spring.h2.console.enabled=true
# For customizing the console URL
spring.h2.console.path=/h2
```

#### Explications :

- ✓ Au niveau de ce fichier, nous avons configuré la base de données H2 : le driver, l'URL, le compte utilisateur, le dialecte permettant la transformation du code java vers SQL et l'activation de la création et de la modification automatique des tables.
- ✓ Vous pouvez ne rien préciser au niveau de ce fichier. En effet, par défaut *Spring Boot* configurera automatiquement la base de données H2. Dans ce cas :
  - Le nom de la base de données sera généré aléatoirement par *Spring Boot* comme illustré ci-après :

```
url=jdbc:h2:mem:a8bdfc89-46f7-49cc-aeel-129f6879c2eb user=SA
```

- localhost:8080/h2-console/test.do?jsessionid=335c27629807c75cff4f5c0f495197a1

Test successful

```

@Entity
@Data
@NoArgsConstructor
@Builder
@AllArgsConstructor
public class Article {
    @Id
    @GeneratedValue
    private Long id;
    private String description;
    private Double price;
    private Double quantity;

    @ManyToOne
    @JoinColumn(name = "categorie_id")
    private Categorie categorie;
}

```

- La classe **Categorie** :

```

package ma.formation.data.rest.service.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Categorie {
    @Id
    @GeneratedValue
    private Long id;
    @Column(unique = true)
    private String categorie;

    @OneToMany(mappedBy = "categorie", cascade = CascadeType.ALL)
    private List<Article> articles = new ArrayList<>();
}

```

```

public void addArticle(Article article) {
    article.setCategorie(this);
    articles.add(article);
}
}

```

#### f. L'interface ArticleDTO

```

package ma.formation.data.rest.domaine;

import ma.formation.data.rest.service.model.Article;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.rest.core.config.Projection;

@Projection(name = "articleDTO", types = Article.class)
public interface ArticleDTO {
    Long getId();

    @Value("#{target.description}")
    String getDesc();

    Double getPrice();

    @Value("#{target.quantity}")
    Double getQuant();

    @Value("#{target.categorie.categorie}")
    String getCat();
}

```

#### Explications :

- ✓ L'annotation `@Projection` de *Spring Data Rest* permet de préciser les projections des données à transférer via Rest. Dans cet exemple, nous allons transmettre : l'identifiant de l'article (champ : *id*), la description (champ : *desc*), le prix (champ : *price*), la quantité (champ : *quant*) et la catégorie (champs : *cat*). Par défaut, *Spring Data Rest* n'envoie pas l'id dans la réponse Rest.
- ✓ Par défaut, *Spring Data Rest* injectera la valeur du champ ayant le même nom au niveau de la classe *Article* (classe précisée dans l'attribut *types* de l'annotation `@Projection`).

- ✓ Si aucun champ ayant le même nom ne se trouve dans la classe *Article*, il faut annoter ce dernier par *@Value* en précisant le nom du champ correspondant au niveau de la classe *Article* en utilisant le mot clé: *target*

## g. Les interfaces DAO : *ArticleRepository* et *CategorieRepository*

- L'interface ***ArticleRepository*** :

```
package ma.formation.data.rest.dao;

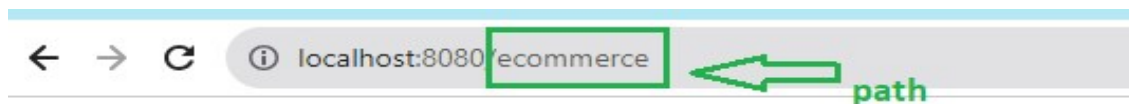
import ma.formation.data.rest.domaine.ArticleDTO;
import ma.formation.data.rest.service.model.Article;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;

import java.util.List;

@RepositoryRestResource(collectionResourceRel = "articles", path = "ecommerce",
excerptProjection = ArticleDTO.class)
public interface ArticleRepository extends JpaRepository<Article, Long> {
    @RestResource(path = "byCategorie", rel = "customFindByDescription")
    List<Article> findByCategorie_Categorie(@Param("categorie") String description);
}
```

### Explications :

- ✓ L'annotation *@RepositoryRestResource* permet d'exposer le Repository *ArticleRepository* sous forme d'Api Rest.
- ✓ L'attribut *collectionResourceRel* sert à configurer le nom du champ qui représentera la collection des articles dans le message JSON ou XML.
- ✓ L'attribut *path* sert à configurer le nom de l'Endpoint dans l'URL. Voir ci-dessous pour plus de précisions :



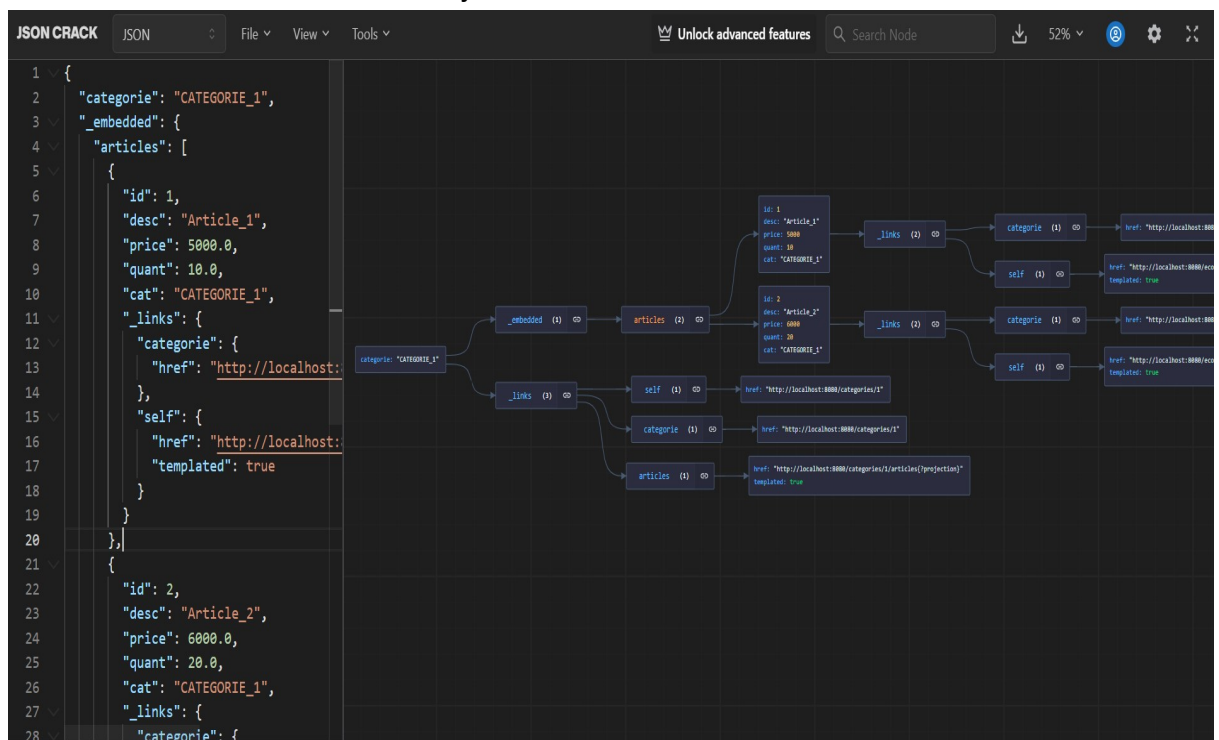
```
{
  "embedded": {
    "articles": [
      {
        "id": 1,
        "desc": "Article_1",
        "price": 120.0,
        "quant": 10.0,
        "cat": "CATEGORIE_1",
        "_links": {
          "self": {
            "href": "http://localhost:8080/ecommerce/1"
          },
          "article": {
            "href": "http://localhost:8080/ecommerce/1{?projection}",
            "templated": true
          },
          "categorie": {
            "href": "http://localhost:8080/ecommerce/1/categorie"
          }
        }
      }
    ]
  }
}
```

**collectionResourceRel**

- ✓ Remarquer la pagination en fin de page

```
"page" : {
  "size" : 20,
  "totalElements" : 5,
  "totalPages" : 1,
  "number" : 0
}
```

- ✓ Vous pouvez utiliser le lien <https://jsoncrack.com/editor> pour une meilleure visualisation de l'arborescence json :





- ✓ L'attribut *excerptProjection* ou (*l'extrait de la projection*) permet de préciser les projections que l'api peut exposer aux clients. Dans l'exemple ci-dessus, c'est les données de l'interface *ArticleDTO* qui seront transmises aux clients :

```
{
  "_embedded" : {
    "articles" : [ {
      "id" : 1,
      "desc" : "Article_1",
      "price" : 120.0,
      "quant" : 10.0,
      "cat" : "CATEGORIE_1",
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/ecommerce/1"
        },
        "article" : {
          "href" : "http://localhost:8080/ecommerce/1{?projection}",
          "templated" : true
        },
        "categorie" : {
          "href" : "http://localhost:8080/ecommerce/1/categorie"
        }
      }
    }
  ]
}, {
```

← Les attributs de l'interface ArticleDTO

- Cette projection est utilisée par défaut dans la consultation de l'ensemble des articles. Par ailleurs, cette même projection n'est pas utilisée si vous consultez un article par son id :

<http://localhost:8080/ecommerce/1>

```
← → ↻ ⓘ localhost:8080/ecommerce/1

{
  "description" : "Article_1",
  "price" : 120.0,
  "quantity" : 10.0,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/ecommerce/1"
    },
    "article" : {
      "href" : "http://localhost:8080/ecommerce/1{?projection}",
      "templated" : true
    },
    "categorie" : {
      "href" : "http://localhost:8080/ecommerce/1/categorie"
    }
  }
}
```

← Categorie n'apparaît pas

- Pour utiliser la projection *ArticleDTO*, il faut utiliser le lien <http://localhost:8080/ecommerce/1?projection=articleDTO> :



```

{
  "id" : 1,
  "desc" : "Article_1",
  "price" : 120.0,
  "cat" : "CATEGORIE_1",
  "quant" : 10.0,
  "_links" : {
    "self" : {
      "href" : "http://localhost:8080/ecommerce/1"
    },
    "article" : {
      "href" : "http://localhost:8080/ecommerce/1{?projection}",
      "templated" : true
    },
    "categorie" : {
      "href" : "http://localhost:8080/ecommerce/1/categorie"
    }
  }
}

```

- ✓ Les services définis dans le Repository sont accessibles via le lien : <http://localhost:8080/ecommerce/search>.
- ✓ Pour chaque service défini dans l'interface Repository, l'annotation `@RestResource` permet de personnaliser le nom du champ qui sera produit au niveau du message JSON/XML ainsi que le nom de l'URL pour accéder au service en question. Voir ci-dessous pour plus de précisions :

GET <http://localhost:8080/ecommerce/search>

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

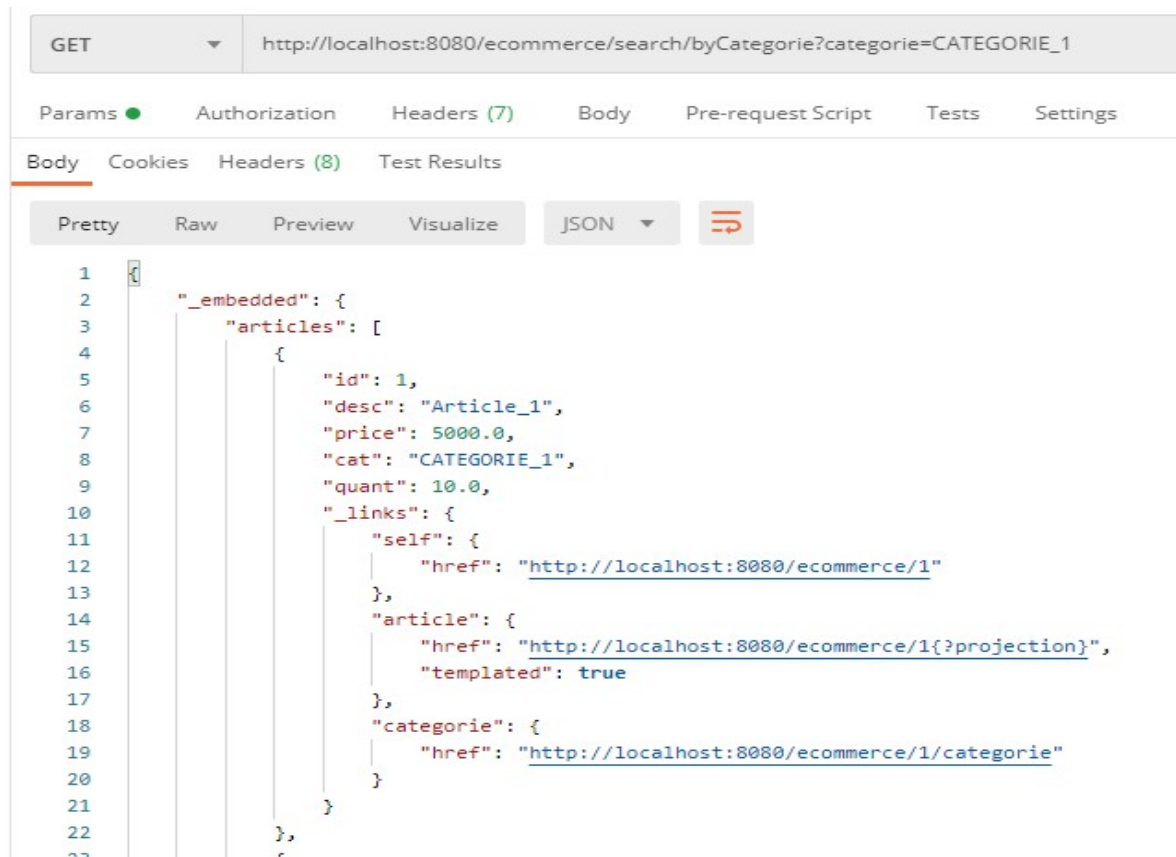
1 {
2   "_links": {
3     "customFindByDescription": {
4       "href": "http://localhost:8080/ecommerce/search/byCategorie?categorie=projection",
5       "templated": true
6     },
7     "self": {
8       "href": "http://localhost:8080/ecommerce/search"
9     }
10  }
11 }

```

C'est ce nom qui est défini dans l'attribut rel de l'annotation `@RestResource`

C'est ce nom qui est défini dans l'attribut path de l'annotation `@RestResource`

- ✓ Le lien [http://localhost:8080/ecommerce/search/byCategorie?categorie=CATEGORIE\\_1](http://localhost:8080/ecommerce/search/byCategorie?categorie=CATEGORIE_1) permet d'exécuter le service `findByCategorie_Categorie` :



- L'interface **CategorieRepository** :

```

package ma.formation.data.rest.dao;

import ma.formation.data.rest.service.model.Categorie;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(collectionResourceRel = "categories", path = "categories")
public interface CategorieRepository extends JpaRepository<Categorie, Long> {
}

```

#### h. La classe MainApplication

- Au niveau de votre classe de démarrage de Spring BOOT, ajouter la méthode suivante :

```

@Bean
public CommandLineRunner initDatabase(CategorieRepository categorieRepository,
                                      ArticleRepository articleRepository) {

    return (a) -> {
        var categorie1 = Categorie.builder().categorie("CATEGORIE_1").build();
        var categorie2 = Categorie.builder().categorie("CATEGORIE_2").build();
        var categorie3 = Categorie.builder().categorie("CATEGORIE_3").build();
    };
}

```

```

var article1 = Article.builder().description("Article_1").price(5000.0).quantity(10.0).build();
var article2 = Article.builder().description("Article_2").price(6000.0).quantity(20.0).build();
var article3 = Article.builder().description("Article_3").price(7000.0).quantity(30.0).build();
var article4 = Article.builder().description("Article_4").price(8000.0).quantity(40.0).build();
var article5 = Article.builder().description("Article_5").price(9000.0).quantity(50.0).build();

```

```

categorie1 = categorieRepository.save(categorie1);
categorie2 = categorieRepository.save(categorie2);
categorie3 = categorieRepository.save(categorie3);

```

```

article1=articleRepository.save(article1);
article2=articleRepository.save(article2);
article3=articleRepository.save(article3);
article4=articleRepository.save(article4);
article5=articleRepository.save(article5);

```

```

categorie1.setArticles(new ArrayList<>());
categorie1.addArticle(article1);
categorie1.addArticle(article2);

```

```

categorie2.setArticles(new ArrayList<>());
categorie2.addArticle(article3);
categorie2.addArticle(article4);

```

```

categorie3.setArticles(new ArrayList<>());
categorie3.addArticle(article5);

```

```

articleRepository.save(article1);
articleRepository.save(article2);
articleRepository.save(article3);
articleRepository.save(article4);
articleRepository.save(article5);

```

```

};
}

```

### **Explications :**

- ✓ La méthode *initDatabase* est annotée par *@Bean* et retourne une instance de type *CommandLineRunner* : sera exécutée après le démarrage de l'application.
- ✓ Spring injectera les deux objets *articleRepository* et *categorieRepository* étant donné que ces derniers ont été passés en paramètre de la méthode *initDatabase()*.

## **VI. Premier test**

- Démarrer la méthode *main* de votre classe de démarrage de Spring Boot.
- Accéder à la console via le lien <http://localhost:8080/h2> :

- Cliquer sur le bouton Connect pour accéder à la base de données :

- Vérifier que les deux tables Article et Categorie ont été bien créés et initialisées :

Run	Run Selected	Auto complete
SELECT * FROM CATEGORIE		
SELECT * FROM CATEGORIE;		
ID	CATEGORIE	
1	CATEGORIE_1	
2	CATEGORIE_2	
3	CATEGORIE_3	
(3 rows, 1 ms)		

Run	Run Selected	Auto complete	Clear	SQL statement:
SELECT * FROM ARTICLE				
SELECT * FROM ARTICLE;				
ID	DESCRIPTION	PRICE	QUANTITY	CATEGORIE_ID
1	Article_1	5000.0	10.0	1
2	Article_2	6000.0	20.0	1
3	Article_3	7000.0	30.0	2
4	Article_4	8000.0	40.0	2
5	Article_5	9000.0	50.0	3
(5 rows, 2 ms)				
Edit				

- Dans test/java, créer la classe **TestArticles** suivante pour effectuer les tests unitaires des deux méthodes GET (findAll et findById) du Repository *ArticleRepository* :

```
package ma.formation.data.rest;

import ma.formation.data.rest.service.model.Article;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class TestArticles {

    @Autowired
    WebApplicationContext context;

    private MockMvc mvc;

    @BeforeEach
    void setUp() {
        this.mvc = MockMvcBuilders.webAppContextSetup(context).build();
    }
}
```

```

@Test
void testGetAllArticles() throws Exception {

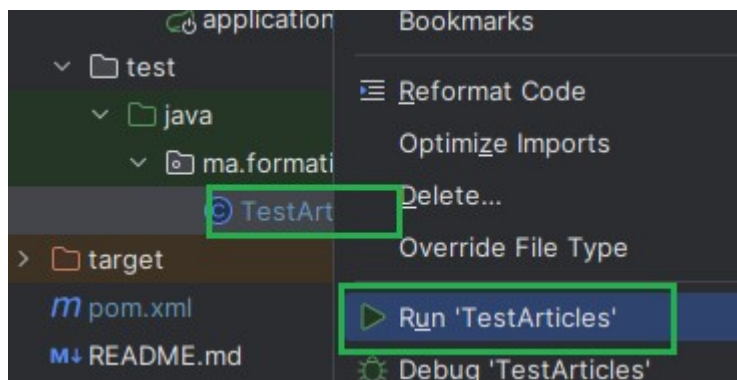
mvc.perform(get("/ecommerce").contentType(MediaType.APPLICATION_JSON).accept(MediaType.APPLICATION_JSON))
    .andExpect(status().isOk())
    .andExpect(jsonPath("$. _embedded.articles[0].desc").value("Article_1"))
    .andExpect(jsonPath("$. _embedded.articles[0].price").value(5000))
    .andExpect(jsonPath("$. _embedded.articles[0].quant").value(10));
}

@Test
void testGetArticleById() throws Exception {

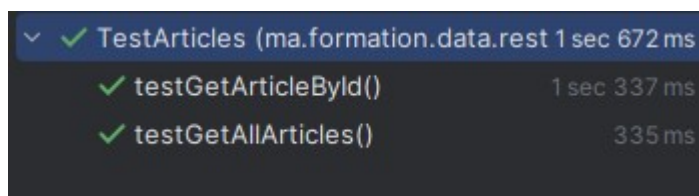
mvc.perform(get("/ecommerce/1").contentType(MediaType.APPLICATION_JSON).accept(MediaType.APPLICATION_JSON))
    .andExpect(status().isOk())
    .andExpect(jsonPath("$.description").value("Article_1"))
    .andExpect(jsonPath("$.price").value(5000))
    .andExpect(jsonPath("$.quantity").value(10));
}
}

```

- Lancer le test comme montré ci-après :



- Vérifier que le résultat est concluant :



## VII. Tests avec Postman

### i. Tester la méthode GET pour toute la liste

- Le lien est <http://localhost:8080/ecommerce> :



```

1  {
2    "_embedded": {
3      "articles": [
4        {
5          "id": 1,
6          "desc": "Article_1",
7          "price": 5000.0,
8          "cat": "CATEGORIE_1",
9          "quant": 10.0,
10         "_links": {
11           "self": {
12             "href": "http://localhost:8080/ecommerce/1"
13           },
14           "article": {
15             "href": "http://localhost:8080/ecommerce/1{?projection}",
16             "templated": true
17           },
18           "categorie": {
19             "href": "http://localhost:8080/ecommerce/1/categorie"
20           }
21         }
22       ],
23     },
24     {
25       "id": 2,
26       "desc": "Article_2",
27       "price": 6000.0,
28       "cat": "CATEGORIE_1",

```

Remarquer que la projection ArticleDTO qui est utilisée

#### j. Tester la méthode GET pour un article donnée et une catégorie donnée

- Le lien est <http://localhost:8080/ecommerce/1> :

```

1  {
2    "description": "Article_1",
3    "price": 5000.0,
4    "quantity": 10.0,
5    "_links": {
6      "self": {
7        "href": "http://localhost:8080/ecommerce/1"
8      },
9      "article": {
10       "href": "http://localhost:8080/ecommerce/1{?projection}",
11       "templated": true
12     },
13     "categorie": {
14       "href": "http://localhost:8080/ecommerce/1/categorie"
15     }
16   }
17 }

```

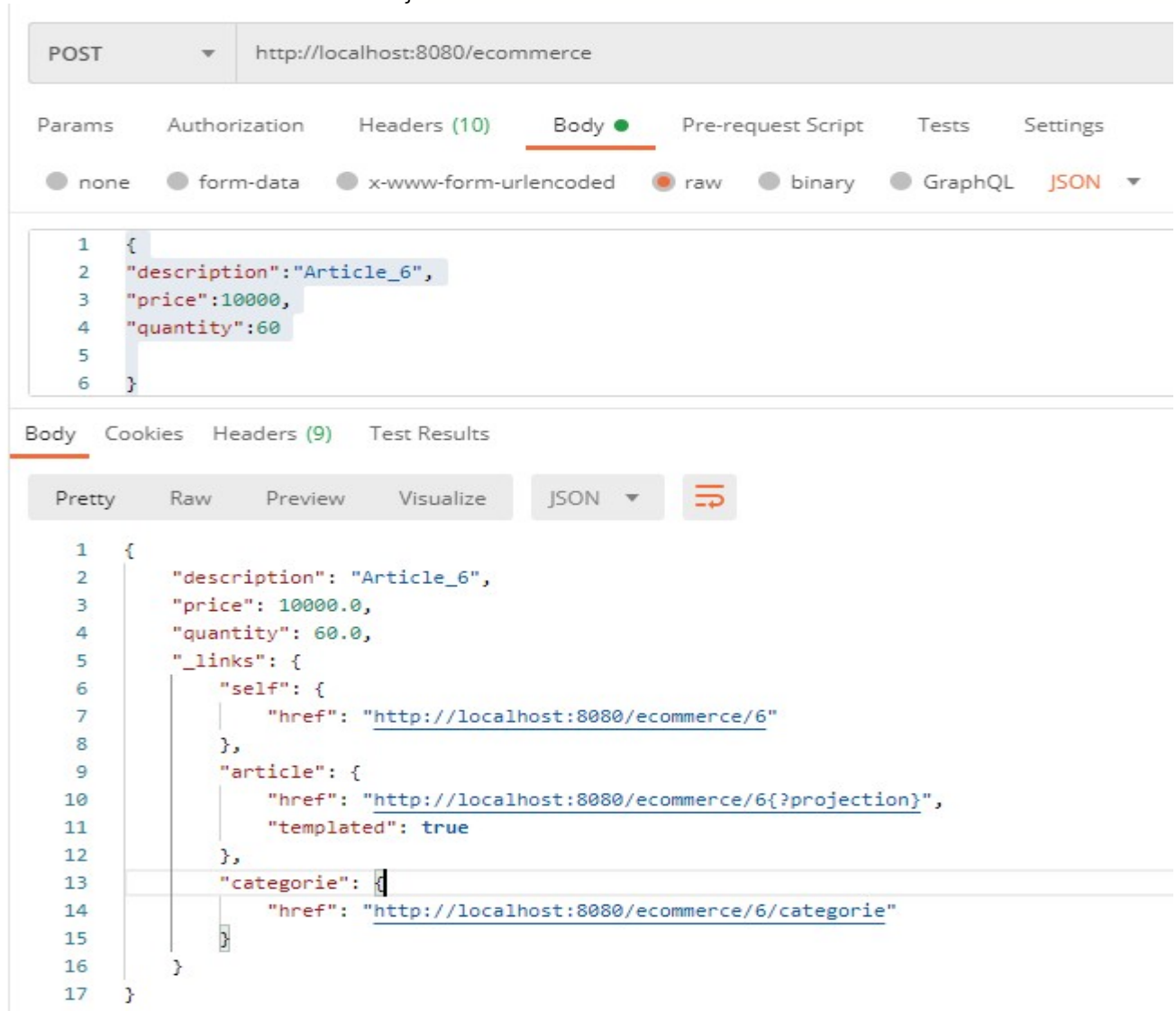
Remarquer que lorsque vous consultez un article par son ID, la projection ArticleDTO n'est pas utilisée.

#### k. Tester la méthode POST pour un article donnée et une catégorie donnée

##### Créer un nouvel article :

- Pour créer un nouveau article, voici le lien : <http://localhost:8080/ecommerce>.
- Dans le *header* ajouter le paramètre *Accept* avec comme valeur *application/json*.
- Dans le *body*, préciser le message JSON suivant :

```
{
  "description": "Article_6",
  "price": 10000,
  "quantity": 60
}
```



### **Créer une nouvelle catégorie :**

- Pour créer une nouvelle catégorie, voici le lien : <http://localhost:8080/categories>.
- Dans le header ajouter le paramètre *Accept* avec comme valeur *application/json*.
- Dans le *body*, préciser le message JSON suivant :

```
{
  "categorie": "CATEGORIE_4"
}
```



}

**POST** <http://localhost:8080/categories>

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1 {
2   "categorie": "CATEGORIE_4"
3 }

```

Body Cookies Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "categorie": "CATEGORIE_4",
3   "_links": {
4     "self": {
5       "href": "http://localhost:8080/categories/4"
6     },
7     "categorie": {
8       "href": "http://localhost:8080/categories/4"
9     },
10    "articles": {
11      "href": "http://localhost:8080/categories/4/articles{?projection}",
12      "templated": true
13    }
14  }
15 }

```

# I. Tester la méthode GET pour consulter un article par son ID en utilisant la projection

- Le lien est <http://localhost:8080/ecommerce/1?projection=articleDTO> :

**GET** <http://localhost:8080/ecommerce/1?projection=articleDTO>

Params Authorization Headers (7) Body Pre-request Script Test

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 1,
3   "desc": "Article_1",
4   "price": 5000.0,
5   "cat": "CATEGORIE_1",
6   "quant": 10.0,
7   "_links": {
8     "self": {
9       "href": "http://localhost:8080/ecommerce/1"
10    },
11    "article": {
12      "href": "http://localhost:8080/ecommerce/1{?projection}",
13      "templated": true
14    },
15    "categorie": {
16      "href": "http://localhost:8080/ecommerce/1/categorie"
17    }
18  }
19 }

```

### m. Tester le service byCategorie

- Le lien est [http://localhost:8080/ecommerce/search/byCategorie?categorie=CATEGORIE\\_1](http://localhost:8080/ecommerce/search/byCategorie?categorie=CATEGORIE_1) :

```
1 {
2   "_embedded": {
3     "articles": [
4       {
5         "id": 1,
6         "desc": "Article_1",
7         "price": 5000.0,
8         "cat": "CATEGORIE_1",
9         "quant": 10.0,
10        "_links": {
11          "self": {
12            "href": "http://localhost:8080/ecommerce/1"
13          },
14          "article": {
15            "href": "http://localhost:8080/ecommerce/1{?projection}",
16            "templated": true
17          },
18          "categorie": {
19            "href": "http://localhost:8080/ecommerce/1/categorie"
20          }
21        }
22      },
23      {
24        "id": 2,
```

### n. Tester la méthode PUT pour associer une catégorie à un article

Pour associer la catégorie CATEGORIE\_4 (ID=4) à l'article Article\_6 (ID=6), suivre les étapes suivantes :

- La méthode est PUT.
- Le lien est <http://localhost:8080/ecommerce/6/categorie>.
- Dans *Headers*, ajouter le paramètre **Content-Type** avec comme valeur **text/uri-list** :

<input checked="" type="checkbox"/>	Content-Type	text/uri-list
-------------------------------------	--------------	---------------

- Dans *Body*, entrer le lien <http://localhost:8080/categories/4>

- Exécuter la requête et vérifier que le code Status est 204 et que la catégorie a été bien associée à l'article :

Run	Run Selected	Auto complete	Clear	SQL statement:
SELECT * FROM ARTICLE				
SELECT * FROM ARTICLE;				
ID	DESCRIPTION	PRICE	QUANTITY	CATEGORIE_ID
1	Article_1	5000.0	10.0	1
2	Article_2	6000.0	20.0	1
3	Article_3	7000.0	30.0	2
4	Article_4	8000.0	40.0	2
5	Article_5	9000.0	50.0	3
6	Article_6	10000.0	60.0	4
(6 rows, 1 ms)				

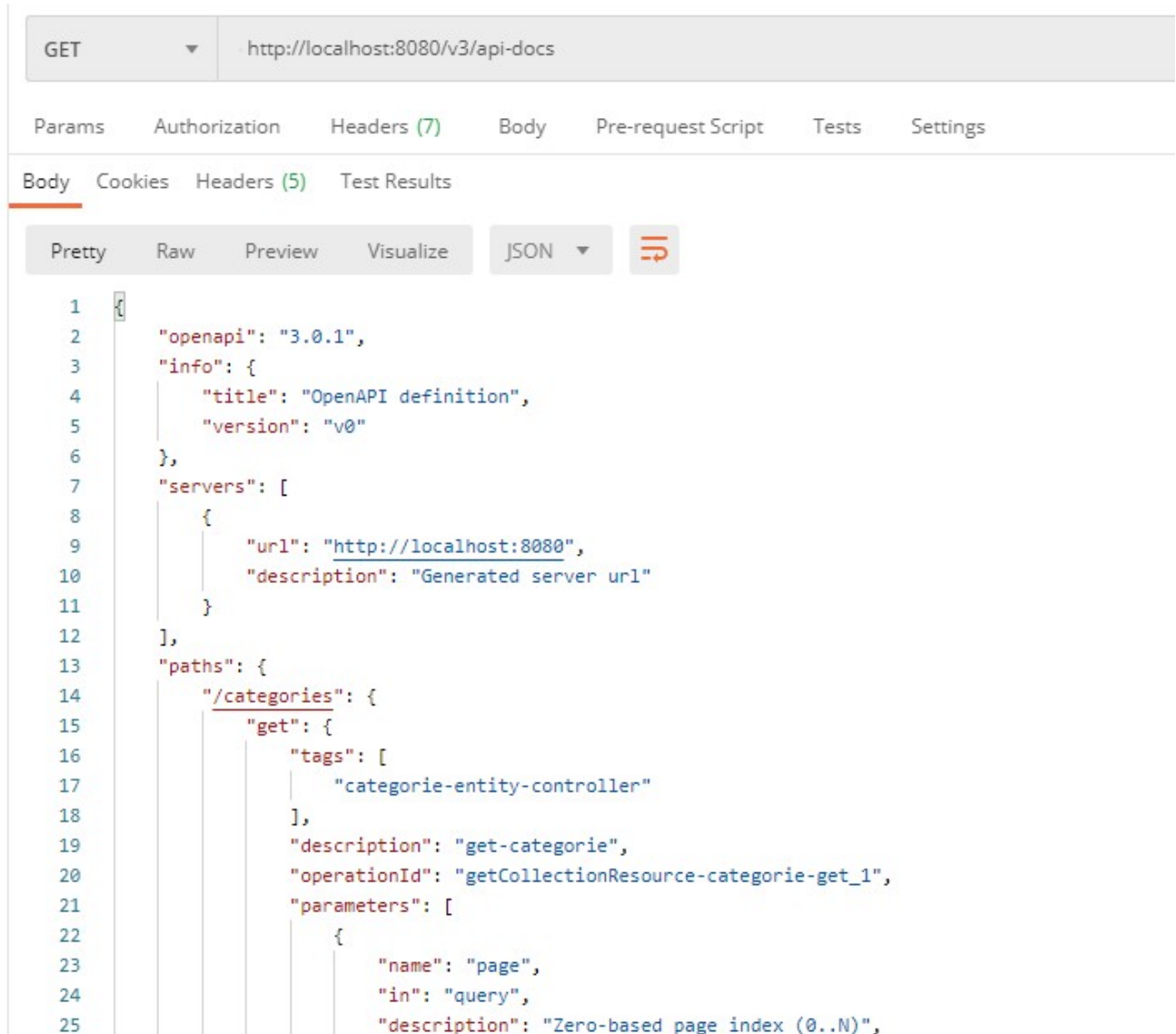
## VIII. Configuration de l'api OpenAPI et Swagger

### 1. Configuration de l'api OpenAPI

- Ajouter la dépendance suivante au niveau du fichier pom.xml :

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.2.0</version>
</dependency>
```

- Démarrer votre application et vérifier que la documentation en format JSON est accessible via le lien suivant : <http://localhost:8080/v3/api-docs> (n'oublier pas de changer votre port au cas où vous n'avez pas utilisé le port 8080) :



- Vous pouvez personnaliser le lien pour accéder à la documentation générée par OpenAPI en ajoutant la clé suivante au niveau du fichier *application.properties* :

```
springdoc.api-docs.path=/api-docs
```

Désormais, le lien de la documentation est <http://localhost:8080/api-docs>

- Le format de la documentation générée par l'api OpenAPI est par défaut JSON. Vous pouvez consulter la documentation en format YAML en accédant au lien suivant :

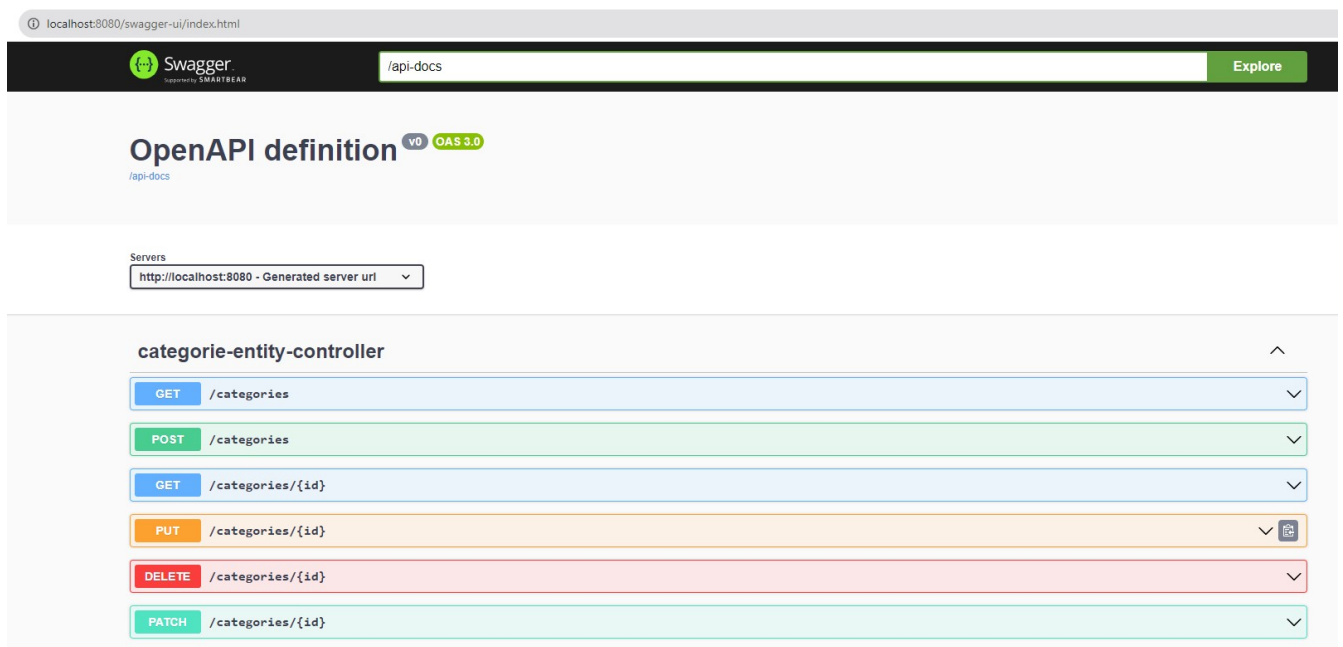
<http://localhost:8080/api-docs.yaml>

## 2. Intégration avec Swagger UI

- En plus de générer la spécification OpenAPI 3, nous pouvons intégrer *springdoc-openapi* à *Swagger UI* pour interagir avec notre spécification API et tester les Endpoints.
- La dépendance *springdoc-openapi* inclut déjà *Swagger UI*, cette dernière est accessible via le lien suivant : <http://localhost:8080/swagger-ui/index.html>.
- Vous pouvez également personnaliser le lien pour accéder à *Swagger UI* en ajoutant la clé suivante au niveau du fichier *application.properties* :

```
springdoc.swagger-ui.path=/docs-ui
```

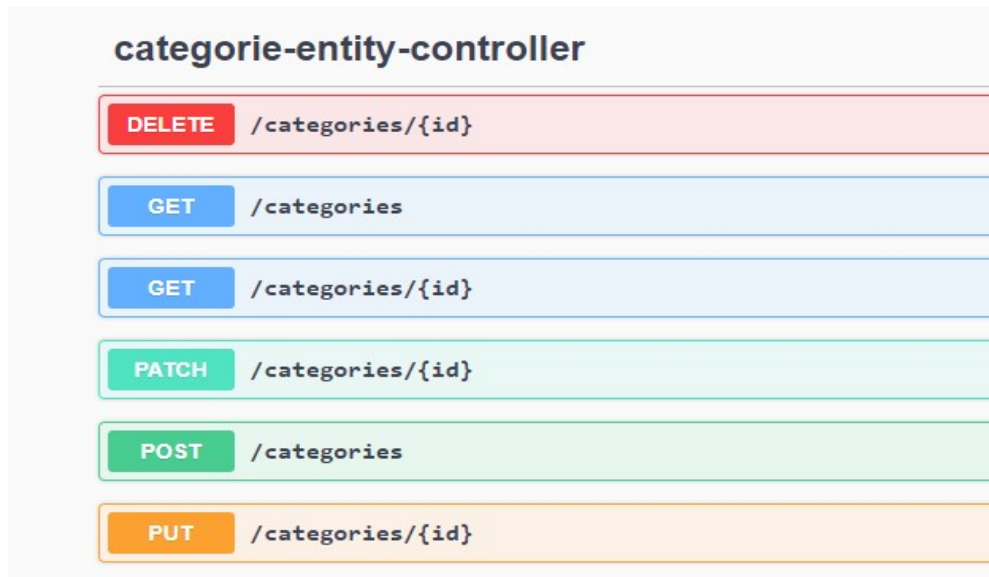
- Démarrer votre application et accéder au lien <http://localhost:8080/docs-ui> :



- Vous pouvez trier les méthodes au niveau de Swagger UI en ajoutant la clé suivante au niveau du fichier *application.properties* :

```
springdoc.swagger-ui.operationsSorter=method
```

- Démarrer votre application et accéder au lien <http://localhost:8080/docs-ui> :



### 3. Tester la méthode Get avec Swagger UI

- Vous pouvez tester par exemple l'Endpoint « *Get /categories* » en suivant les étapes suivantes :
  - ✓ Cliquer sur **Get**.
  - ✓ Cliquer ensuite sur le bouton **Try it out** :

The image shows the 'Try it out' interface in Swagger UI for the GET /categories endpoint. The endpoint name 'get-categorie' is at the top. Below it, the 'Parameters' section is expanded, showing a table with columns 'Name' and 'Description'. The parameters are: 'page' (integer, query) with a value of 0, 'size' (integer, query) with a value of 2, and 'sort' (array[string], query) with a description 'Sorting criteria in the format: property(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.' and an 'Add string item' button. At the bottom, there are 'Execute' and 'Clear' buttons. A 'Cancel' button is also visible in the top right of the parameters section.

- ✓ Entrer le numéro de la page et le size (ici on souhaite consulter les deux premières catégories).
- ✓ Cliquer ensuite sur **Execute**, le résultat est le suivant :

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:8080/categories?page=0&size=2' \
  -H 'accept: application/hal+json'
```

Request URL

```
http://localhost:8080/categories?page=0&size=2
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "_embedded": {     "categories": [       {         "categorie": "CAT\u00c9GORIE_1",         "_embedded": {           "articles": [             {               "id": 1,               "desc": "Article_1",               "price": 120,               "cat": "CAT\u00c9GORIE_1",               "quant": 10,               "_links": {                 "categorie": {                   "href": "http://localhost:8080/ecommerce/1/categorie"                 },                 "self": {                   "href": "http://localhost:8080/ecommerce/1/{?projection}",                   "templated": true                 }               }             },             {               "id": 2,               "desc": "Article_2",               "price": 6000,               "cat": "CAT\u00c9GORIE_1",               "quant": 10,               "_links": {                 "categorie": {                   "href": "http://localhost:8080/ecommerce/1/categorie"                 },                 "self": {                   "href": "http://localhost:8080/ecommerce/1/{?projection}",                   "templated": true                 }               }             }           ]         }       }     ]   } }</pre> <p>Response headers</p>

#### 4. Tester la m\u00e9thode Put avec Swagger UI

- Pour modifier la cat\u00e9gorie de l'article dont l'id est \u00e9gal \u00e0 1, suivre les \u00e9tapes suivantes :
  - \u2713 Cliquer sur « **PUT ecommerce/{id}/categorie** » suivante :



- \u2713 Cliquer ensuite sur **Try it out** :

PUT /ecommerce/{id}/categorie

update-categorie-by-article-id

Parameters

Name	Description
id * required	
string (path)	1

Request body required

text/uri-list

http://localhost:8080/categories/2

- \u2713 Entrer l'id de l'article.
- \u2713 Dans **Request body**, choisir **text/uri-list**
- \u2713 Entrer le lien <http://localhost:8080/categories/2>
- \u2713 Ensuite cliquer sur **Ex\u00e9cute**.

✓ Puis vérifier que l'update s'est bien passé :

GET /categories

get-categorie

Parameters

Name	Description
page integer (query)	Zero-based page index (0..N) <input type="text" value="0"/>
size integer (query)	The size of the page to be returned <input type="text" value="10"/>
sort array[string] (query)	Sorting criteria in the format: property.(asc desc). Default sort order is ascending. Multiple sort criteria are supported. <input type="button" value="Add string item"/>

Execute

```
{
  "categorie": "CATEGORIE_2",
  "_embedded": {
    "articles": [
      {
        "id": 1,
        "desc": "Article 1",
        "cat": "CATEGORIE_2",
        "quant": 10,
        "price": 5000,
        "_links": {
          "categorie": {
            "href": "http://localhost:8080/ecommerce/1/categorie"
          },
          "self": {
            "href": "http://localhost:8080/ecommerce/1{?projection}",
            "templated": true
          }
        }
      }
    ]
  }
},
```

Ou bien : <http://localhost:8080/ecommerce>

localhost:8080/ecommerce

Impression élégante

```
{
  "_embedded": {
    "articles": [ {
      "id" : 1,
      "desc" : "Article 1",
      "cat" : "CATEGORIE_2",
      "quant" : 10.0,
      "price" : 5000.0,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8080/ecommerce/1"
        },
        "article" : {
          "href" : "http://localhost:8080/ecommerce/1{?projection}",
          "templated" : true
        }
      },
      "categorie" : {
        "href" : "http://localhost:8080/ecommerce/1/categorie"
      }
    }, {
      "id" : 2,
      "desc" : "Article 2",
      "cat" : "CATEGORIE_1",
```



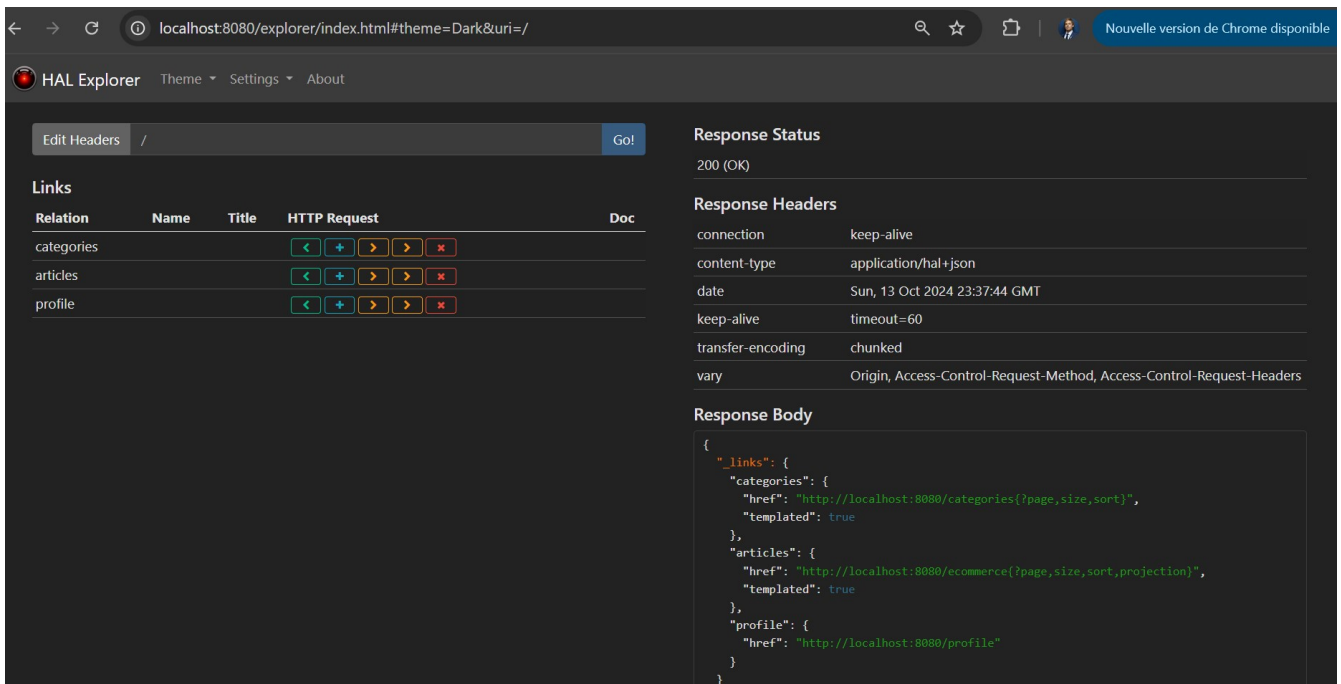
## IX. HAL Explorer

Kai Tödter a créé une application utile : HAL Explorer. Il s'agit d'une application Web basée sur Angular qui vous permet d'explorer facilement les réponses HTTP basées sur HAL et HAL-FORMS. Elle prend également en charge les profils Spring générés par Spring Data REST. Vous pouvez la pointer vers n'importe quelle API Spring Data REST et l'utiliser pour naviguer dans l'application et créer de nouvelles ressources

La liste suivante montre comment ajouter la dépendance dans Maven :

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

Si vous utilisez Spring Boot ou Spring Data, vous n'avez pas besoin de spécifier la version. Cette dépendance configure automatiquement HAL Explorer pour qu'il soit servi lorsque vous visitez l'URI racine de votre application dans un navigateur. (REMARQUE : <http://localhost:8080/> a été connecté au navigateur et a été redirigé vers l'URL affichée dans l'image suivante.)



The screenshot shows the HAL Explorer web application running in a browser at `localhost:8080/explorer/index.html#theme=Dark&uri=/`. The interface is dark-themed and includes a navigation bar with 'Theme', 'Settings', and 'About' options. A 'Go!' button is next to the 'Edit Headers' input field. The main content area is divided into two panels. The left panel, titled 'Links', contains a table with columns 'Relation', 'Name', 'Title', 'HTTP Request', and 'Doc'. It lists three links: 'categories', 'articles', and 'profile', each with a set of navigation buttons (back, forward, search, etc.). The right panel displays the 'Response Status' (200 OK), 'Response Headers' (including 'keep-alive', 'content-type: application/hal+json', 'date', 'keep-alive: timeout=60', 'transfer-encoding: chunked', and 'vary'), and the 'Response Body' which is a JSON object containing links to 'categories', 'articles', and 'profile'.

Relation	Name	Title	HTTP Request	Doc
categories			<a href="#">←</a> <a href="#">+</a> <a href="#">→</a> <a href="#">↻</a> <a href="#">✕</a>	
articles			<a href="#">←</a> <a href="#">+</a> <a href="#">→</a> <a href="#">↻</a> <a href="#">✕</a>	
profile			<a href="#">←</a> <a href="#">+</a> <a href="#">→</a> <a href="#">↻</a> <a href="#">✕</a>	

```
{
  "_links": {
    "categories": {
      "href": "http://localhost:8080/categories?page,size,sort",
      "templated": true
    },
    "articles": {
      "href": "http://localhost:8080/ecommmerce?page,size,sort,projection",
      "templated": true
    },
    "profile": {
      "href": "http://localhost:8080/profile"
    }
  }
}
```

**HAL Explorer** Theme ▾ Settings ▾ About

Edit Headers /ecommerce Go!

**JSON Properties**

```
{
  "page": {
    "size": 20,
    "totalElements": 5,
    "totalPages": 1,
    "number": 0
  }
}
```

**Links**

Relation	Name	Title	HTTP Request	Doc
self			< > < > < >	
profile			< > < > < >	
search			< > < > < >	

**Embedded Resources**

- articles
  - articles [0]
  - articles [1]
  - articles [2]
  - articles [3]
  - articles [4]

**Response Status**

200 (OK)

**Response Headers**

connection	keep-alive
content-type	application/hal+json
date	Sun, 13 Oct 2024 23:40:27 GMT
keep-alive	timeout=60
transfer-encoding	chunked
vary	Origin, Access-Control-Request-Method, Access-Control-Request-Headers

**Response Body**

```
{
  "_embedded": {
    "articles": [
      {
        "id": 1,
        "desc": "Article 1",
        "cat": "CATEGORIE_2",
        "quant": 10,
        "price": 5000,
        "_links": {
          "self": {
            "href": "http://localhost:8080/ecommerce/1"
          },
          "article": {
            "href": "http://localhost:8080/ecommerce/1{?projection}",
            "templated": true
          },
          "categorie": {
            "href": "http://localhost:8080/ecommerce/1/categorie"
          }
        }
      }
    ]
  }
}
```

Documentation officielle de HAL : <https://datatracker.ietf.org/doc/html/draft-kelly-json-hal>

## Conclusion

Le code source du TP est disponible sur GITHUB :

<https://github.com/abbouformations/tpdatarest.git>.