Architecture orientée services (SOA) REST (Spring Boot)

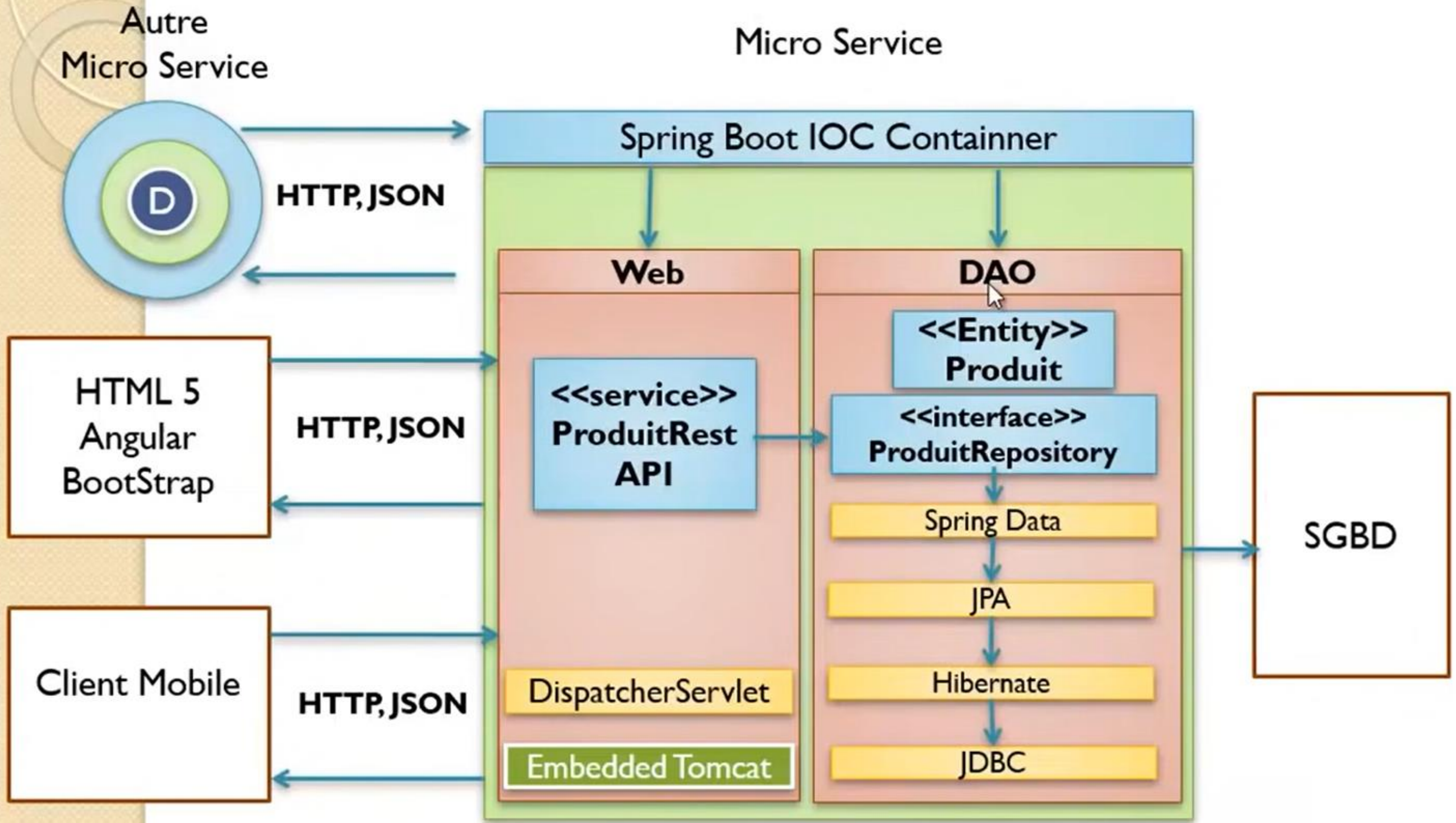
Producing a Restful web service (Spring Boot) - Partie I

Service Rest (Spring Boot)

- Pour créer Rest service avec spring Boot
  1. **Créer avec Rest controler(@Controller + @Responsebody)**
  2. Se baser sur spring data rest (@RepositoryRestResource)
  3. Jaxrs avec jersy

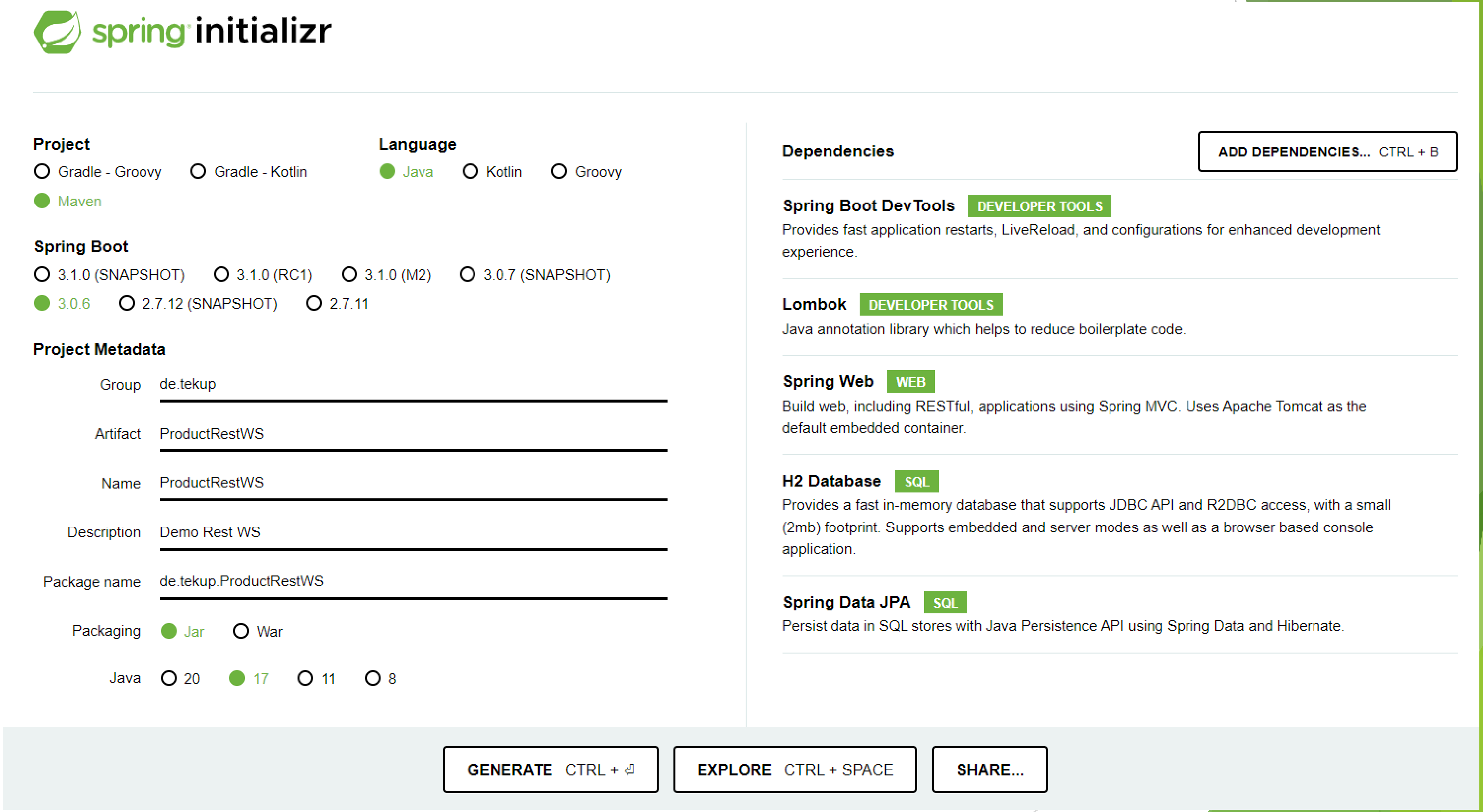
Exemple

- Nous allons créer un service simple qui gère les produits en stock. Nous allons stocker les objets produits dans une base de données (H2 en mémoire) et y accéder


Autre Micro Service
D
Micro Service
Spring Boot IOC Containner
HTTP, JSON
Web
DAO
<<Entity>> Produit
HTML 5
Angular
BootStrap
HTTP, JSON
<<service>> ProduitRest API
<<interface>> ProduitRepository
Spring Data
SGBD
JPA
Client Mobile
Hibernate
HTTP, JSON
DispatcherServlet
Embedded Tomcat
JDBC

Technology Stack

- JDK 17, Intellij IDEA, Maven – Development environment
- Spring-boot – Underlying application framework
- jackson-dataformat-xml – for XMl
- Postman – for testing our service
- dependencies to a project:
  - Web
  - JPA
  - H2
  - Lombok



spring initializr

Project

○ Gradle - Groovy ○ Gradle - Kotlin [✓ Maven]

Language

[✓ Java] ○ Kotlin ○ Groovy

Spring Boot

○ 3.1.0 (SNAPSHOT) ○ 3.1.0 (RC1) ○ 3.1.0 (M2) ○ 3.0.7 (SNAPSHOT) [✓ 3.0.6] ○ 2.7.12 (SNAPSHOT) ○ 2.7.11

Project Metadata

Group	de.tekup
Artifact	ProductRestWS
Name	ProductRestWS
Description	Demo Rest WS
Package name	de.tekup.ProductRestWS
Packaging	[✓ Jar] ○ War
Java	○ 20 [✓ 17] ○ 11 ○ 8

Dependencies

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

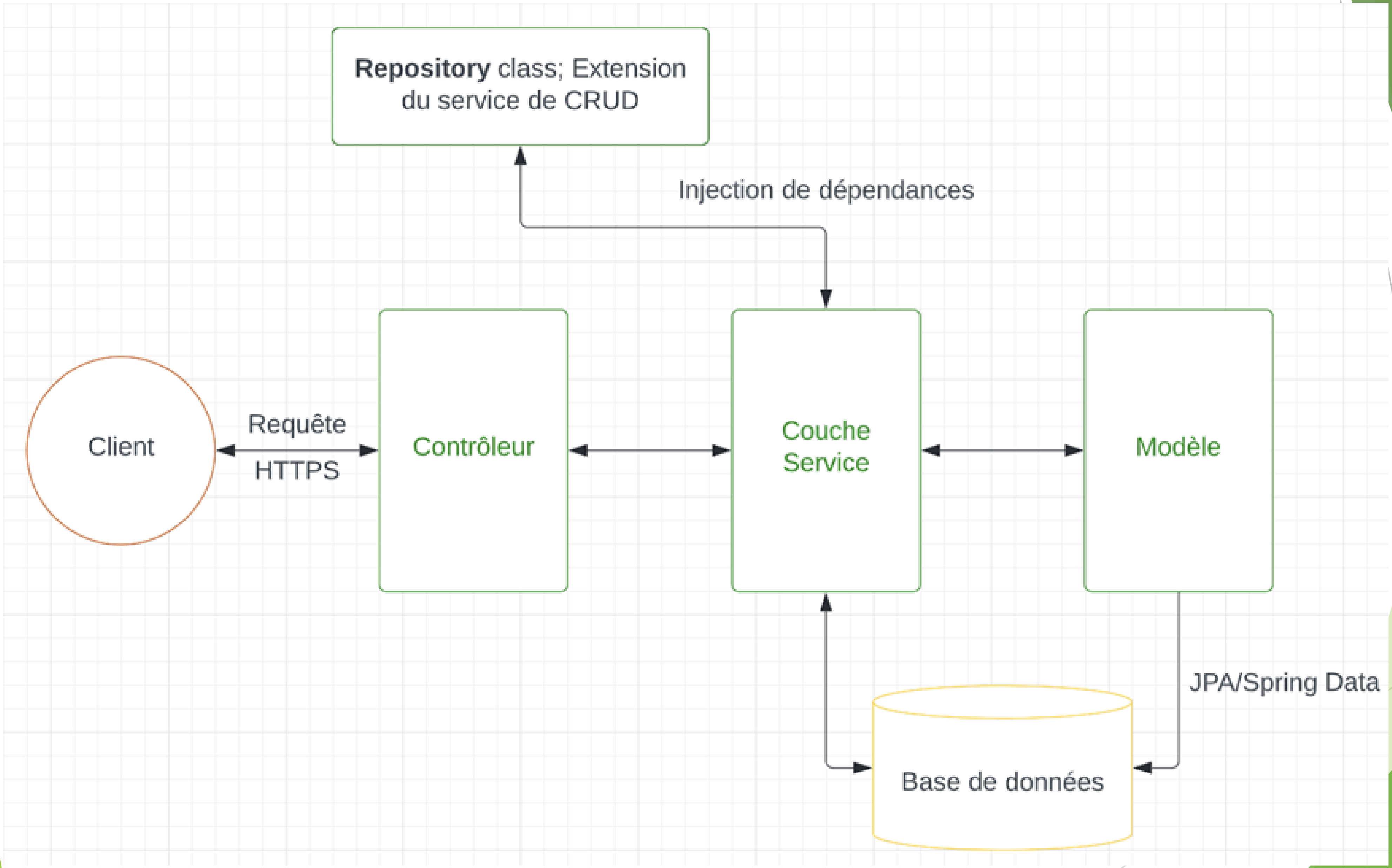
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

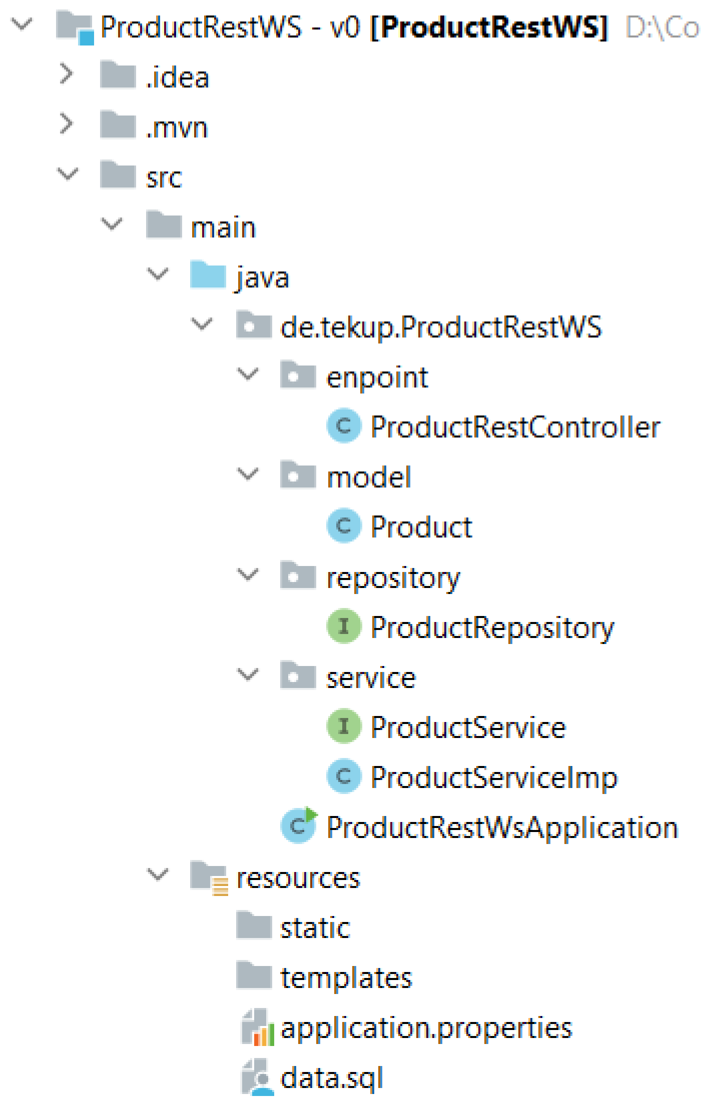
H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Ou cloner à partir du Git

- git clone https://github.com/ines-mouakher/ProductRestWS-init.git


Repository class; Extension du service de CRUD
Injection de dépendances
Client
Requête
HTTPS
Contrôleur
Couche Service
Modèle
JPA/Spring Data
Base de données



```
ProductRestWS - v0 [ProductRestWS] D:\Co
  .idea
  .mvn
  src
    main
      java
        de.tekup.ProductRestWS
          enpoint
            ProductRestController
          model
            Product
          repository
            ProductRepository
          service
            ProductService
            ProductServiceImp
          ProductRestWsApplication
      resources
        static
        templates
        application.properties
        data.sql
```

Domain Classes: Produit

- La classe Produit représente un produit.

```
@Entity
@Data //Getters et Setters
@NoArgsConstructor
@AllArgsConstructor
@ToString

public class Product {
   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)

   private Long id;
   private String designation;
   private double price;
   private int quantity;
}
```

Produit Repository

- With this domain object definition, we can now turn to Spring Data JPA to handle the tedious database interactions.
- Spring Data JPA repositories are interfaces with methods supporting creating, reading, updating, and deleting records against a back end data store.
- To get all this free functionality, all we had to do was declare an interface which extends Spring Data JPA's JpaRepository, specifying the domain type as Produit and the id type as Long.

```
@Repository
public interface ProductRepository  extends JpaRepository<Product,Long> {
    public List<Product> findByPrice(double price);
}
```

Consulter BD

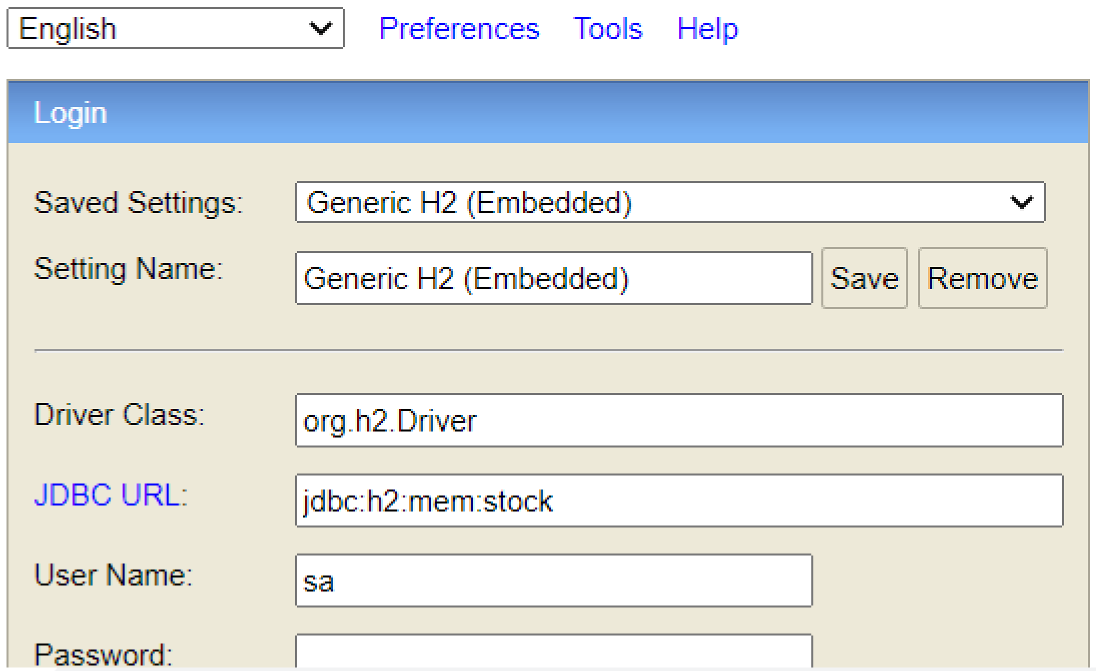
- In the file application.properties we add :

```
spring.jpa.defer-datasource-initialization=true
spring.datasource.url=jdbc:h2:mem:stock
spring.h2.console.enable=true
```

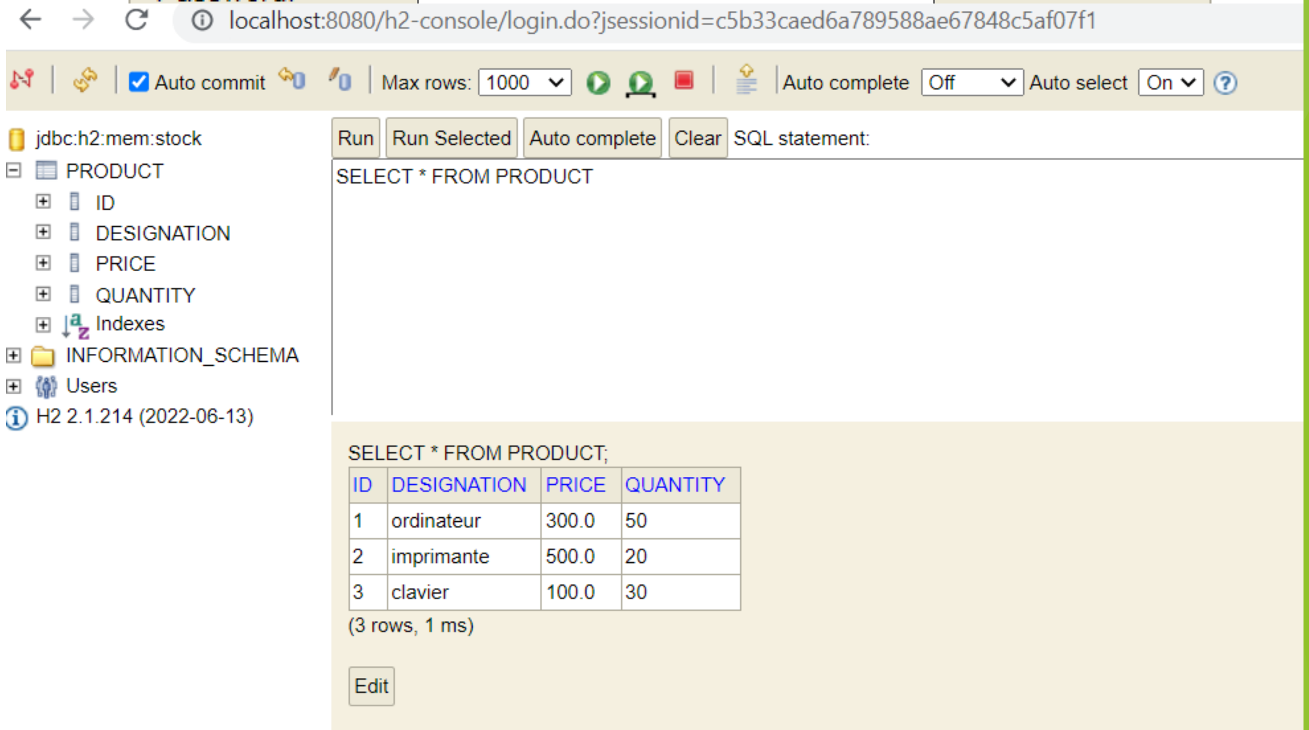
- Dans resources créer un fichier data.sql

```
--INITIALISATION TABLE PRODUIT
INSERT INTO  Product(designation, price, quantity) values ('ordinateur', 300, 50);
INSERT INTO Product(designation, price, quantity) values ('imprimante', 500, 20);
INSERT INTO Product(designation, price, quantity) values ('clavier', 100, 30);
```

- Consulter BD
  - http://localhost:8080/h2-console/



Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:stock
User Name:	sa
Password:	



```
SELECT * FROM PRODUCT
```

SELECT * FROM PRODUCT;

ID	DESIGNATION	PRICE	QUANTITY
1	ordinateur	300.0	50
2	imprimante	500.0	20
3	clavier	100.0	30

(3 rows, 1 ms)

Service

```java
public interface ProductService {
    public List<Product> productList();
    public Product getOne(Long id);
    public Product save(Product produit);
    public Product update(Product produit, Long id);
    public void delete(Long id);
    public List<Product> findByPrice(double price);
}
```

```java
@Service
public class ProductServiceImp implements ProductService{

    @Autowired
    private ProductRepository productRepository;

    @Override
    public List<Product> productList()
        return productRepository.findAll();
    }

    @Override
    public List<Product> findByPrice(double price)
    {

        return productRepository.findByPrice(price);
    }


    @Override
    public Product getOne(Long id){
        return productRepository.findById(id).get();}

    @Override
    public Product save(Product product){
        return productRepository.save(product);
    }

    @Override
    public Product update(Product product, Long id){
        product.setId(id);
        return productRepository.save(product);
    }

    @Override
     public void delete(Long id){
        productRepository.deleteById(id);
    }
}
```

Endpoint

- Notre application est comme une maison. Si on n'y met aucune porte, impossible d’y accéder ! Les portes de notre application sont ce qu’on appelle **des *endpoints***. Un endpoint est associé à une URL. Lorsqu'on appelle cette URL, on reçoit une réponse, et cet échange se fait en **HTTP**.
- Nos controllers @RestController nous permettent de définir des URL et le code à exécuter quand ces dernières sont requêtées.

Rest controller

```
@RestController
@RequestMapping("/apiv1/stock")// specifier un path pour le web service
public class ProductRestController {

    @Autowired
    private ProductService productService;

    @GetMapping(path = "/products")
    public List<Product> produitList() {

        return productService.productList();
    }

    @GetMapping(path = "/products/ByPrice/{price}")
    public List<Product> findByPrice(@PathVariable double price) {
        return productService.findByPrice(price);
    }

    @GetMapping(path = "/products/{id}")
   public Product getOne(@PathVariable Long id) {
         return productService.getOne(id);
    }
    ….
}
```

Rest controller

Annotation	
@RestController	used to define the RESTful web services. It serves JSON, XML and custom response.
@RequestMapping	used to define the Request URI to access the REST Endpoints. We can define Request method to consume and produce object. The default request method is GET.
@RequestBody	used to define the request body content type.
@PathVariable	annotation is used to define the custom or dynamic request URI. The Path variable in request URI is defined as curly braces {}
@RequestParam	used to read the request parameters from the Request URL. By default, it is a required parameter.

https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_services.htm

@RequestParam vs. @PathVariable

- @RequestParam captures values from the query parameters or form parameters in the URL, which are usually appended to the URL after a “?” symbol. For example: “/users?name=john&age=30”.
- On the other hand, @PathVariable captures values from the URL path itself, which is usually separated by slashes (“/”). For example: “/users/{id}”.
- Optional vs. Required: By default, @RequestParam parameters are considered optional, as they can have a default value specified and can be omitted from the URL without causing an error. However, @PathVariable parameters are considered required, as they are part of the URL path and must be provided in the URL for the mapping to match.

La négociation de contenu

- HTTP permet la négociation de contenu proactive. Cela signifie qu’un client peut envoyer ses préférences au serveur. Ce dernier doit répondre au mieux en fonction des préférences reçues et de ses capacités. Une négociation possible porte sur le format de représentation. Cela peut s’avérer utile pour une API Web destinée à des clients très divers. Par exemple, certains clients peuvent privilégier le XML et d’autres le JSON.
- La négociation proactive pour le type de représentation est réalisée par le client qui envoie dans sa requête un en-tête Accept donnant la liste des types MIME qu’il préfère. Avec Spring Web MVC, cette négociation est automatiquement gérée par le contrôleur.
- Nous pouvons maintenant faire évoluer notre contrôleur pour indiquer qu’il peut produire du JSON ou du XML en déclarant un tableau de types MIME dans l’attribut produces
- Par défaut, ce contrôleur produit toujours du JSON (car le type MIME JSON est placé en premier dans la liste) mais un client peut indiquer qu’il préfère une représentation XML grâce à l’en-tête Accept :

Format de représentation de resource

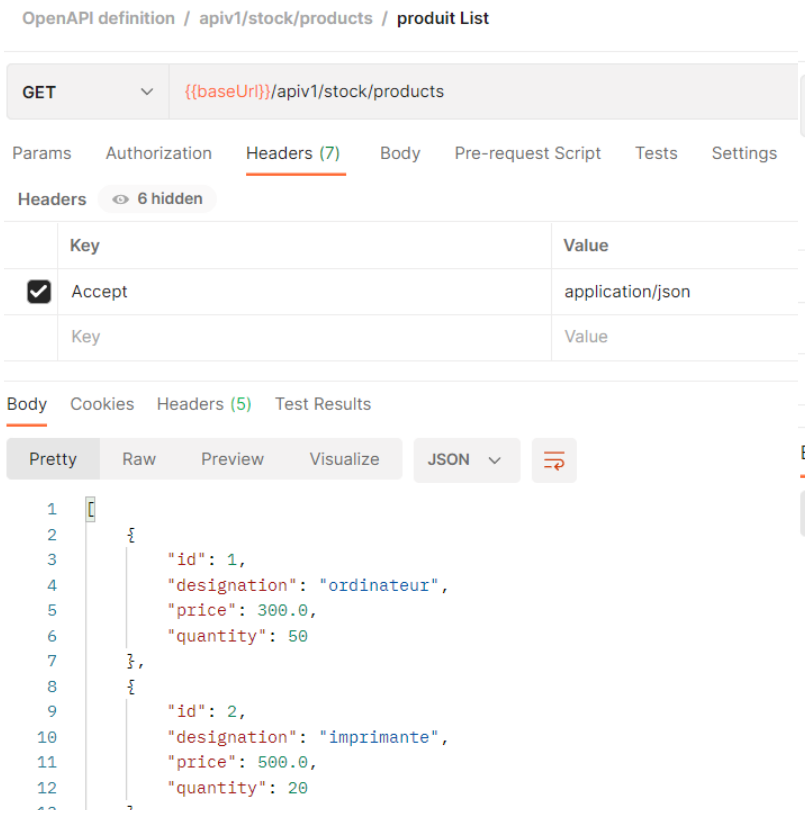
- produces = { MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }
- L'attribut produces sert à spécifier un URL qui créera seulement (renverra au utilisateur) les données sous quel format.

```
@GetMapping(path = "/products", produces = {"application/json", "application/xml"})
//mettre deux formats possibles peut generer erreur dans navigateur car il a  besoin de header accept
// produire soit Jsom soit xml
 public List<Product> produitList() {

        return productService.productList();
   }
```

Add dependency to support XML format

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

Tester avec Postman



OpenAPI definition / apiv1/stock/products / produit List

GET {{baseUrl}}/apiv1/stock/products

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

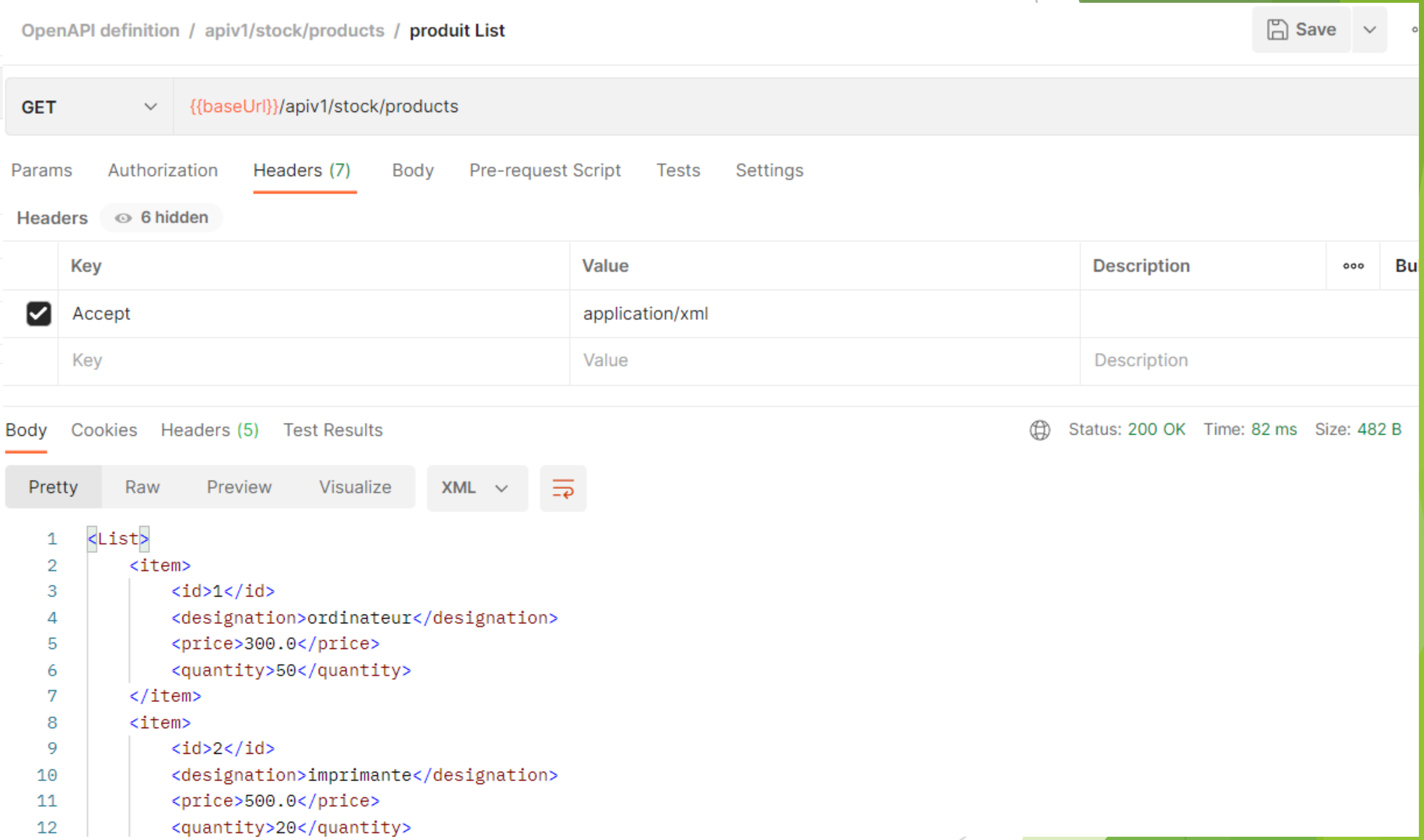
Headers 6 hidden

	Key	Value
✓	Accept	application/json
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "designation": "ordinateur",
        "price": 300.0,
        "quantity": 50
    },
    {
        "id": 2,
        "designation": "imprimante",
        "price": 500.0,
        "quantity": 20
```



OpenAPI definition / apiv1/stock/products / produit List

Save

GET {{baseUrl}}/apiv1/stock/products

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

Headers 6 hidden

	Key	Value	Description
✓	Accept	application/xml	
	Key	Value	Description

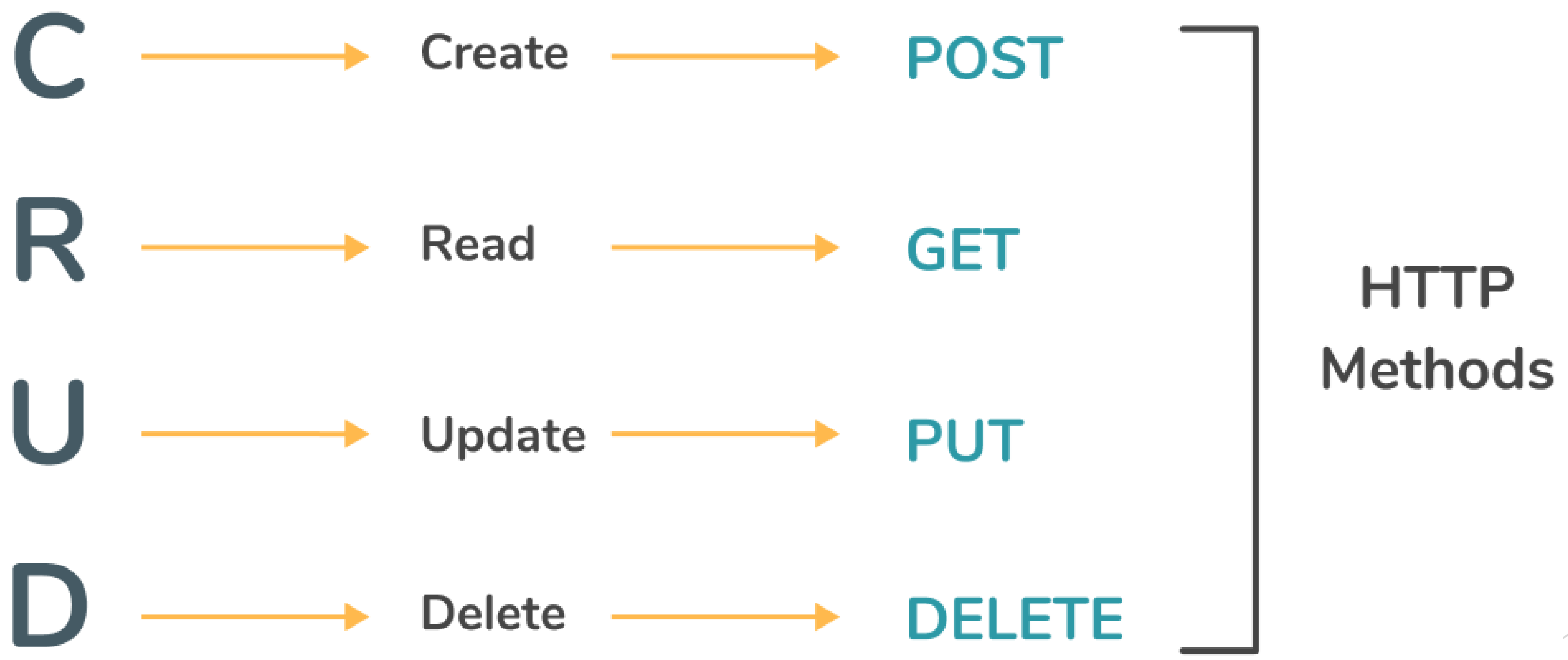
Body Cookies Headers (5) Test Results

Status: 200 OK Time: 82 ms Size: 482 B

Pretty Raw Preview Visualize XML

```
<List>
    <item>
        <id>1</id>
        <designation>ordinateur</designation>
        <price>300.0</price>
        <quantity>50</quantity>
    </item>
    <item>
        <id>2</id>
        <designation>imprimante</designation>
        <price>500.0</price>
        <quantity>20</quantity>
```

HTTP methods



POST API

- The HTTP POST request is used to create a resource. This method contains the Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

```
@PostMapping(path = "/products")
    public Product save(@RequestBody Product product) {

        return productService.save(product);
    }
```

PUT API

- The HTTP PUT request is used to update the existing resource. This method contains a Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

```
@PutMapping(path = "/products/{id}")
  public Product update(@RequestBody Product product, @PathVariable Long id) {
    return productService.update(product,id);
  }
```

DELETE API

- The HTTP Delete request is used to delete the existing resource. This method does not contain any Request Body. We can send request parameters and path variables to define the custom or dynamic URL.

```
@DeleteMapping(path = "/products/{id}")
 public void delete(@PathVariable Long id) {
     productService.delete(id);
 }
```

Document your RESTful

Why do we need to document your RESTful API?

- The most important design principle for RESTful Services is
    - Think about consumer of the API.
- REST does not specify a documentation standard or a contract like SOAP (WSDL). REST gives you the flexibility to choose your documentation format and approach. But that does not mean “No documentation”

How do you document your RESTful API?

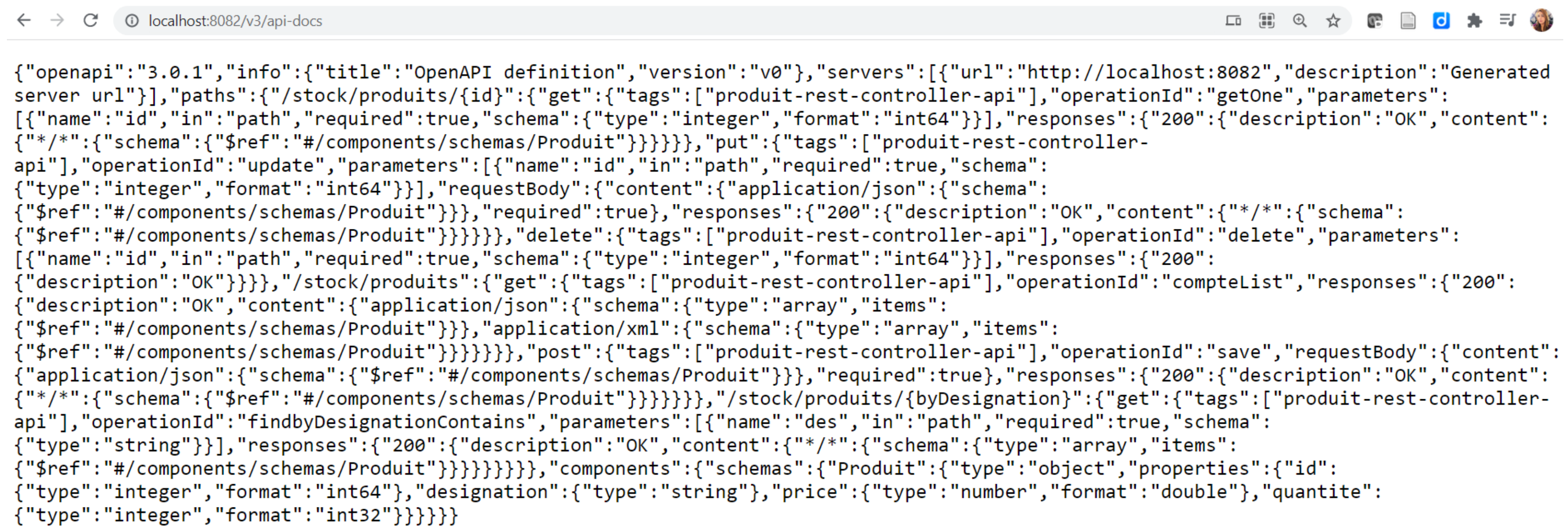
- One option is to maintain documentation manually. But that gets outdated quickly.
- Other option is to generate documentation from code. And that's the approach we would discuss in this guide.
- There are multiple approaches to documenting your RESTful API
  - WADL
  - RESTDocs
  - Swagger or OpenDocs
  - Swagger has picked up momentum in the last couple of years and is now the most popular REST API documentation standard. We will use Swagger in this guide.

How to document SpringBoot Rest APIs using Open API and Swagger?

- Ajouter au fichier pom.xml les *dependencies* suivantes:

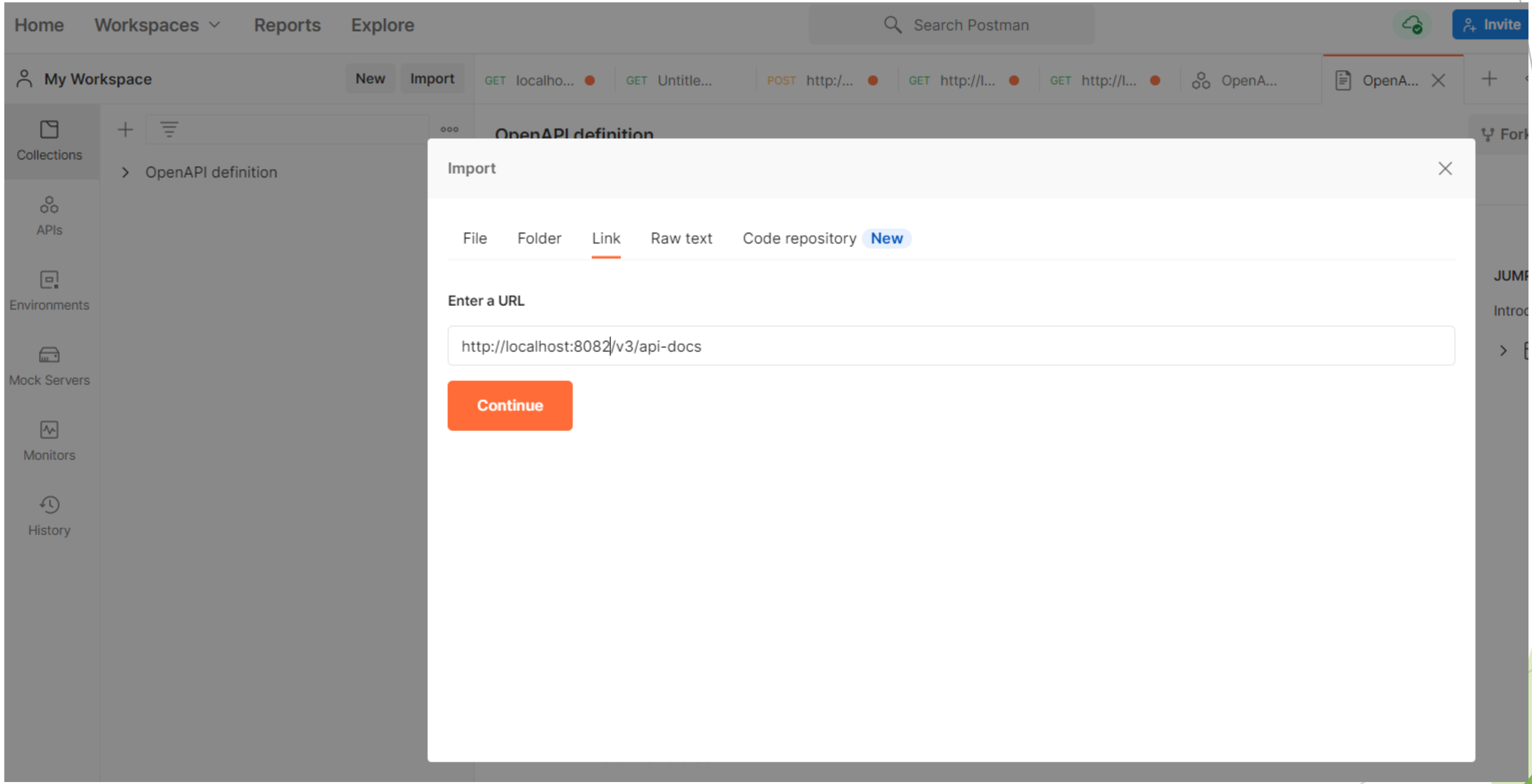
```
<!-- openapi-ui dependency-->
<dependency>
   <groupId>org.springdoc</groupId>
   <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
   <version>2.1.0</version>
</dependency>

<!-- openapi-ui dependency-->


<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-validator -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-validator</artifactId>
   <version>8.0.0.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/jakarta.validation/jakarta.validation-api -->
<dependency>
   <groupId>jakarta.validation</groupId>
   <artifactId>jakarta.validation-api</artifactId>
   <version>3.0.2</version>
</dependency>
```

http://localhost:8080/v3/api-docs



```
{"openapi":"3.0.1","info":{"title":"OpenAPI definition","version":"v0"},"servers":[{"url":"http://localhost:8082","description":"Generated
server url"}],"paths":{"/stock/produits/{id}":{"get":{"tags":["produit-rest-controller-api"],"operationId":"getOne","parameters":
[{"name":"id","in":"path","required":true,"schema":{"type":"integer","format":"int64"}}],"responses":{"200":{"description":"OK","content":
{"*/*":{"schema":{"$ref":"#/components/schemas/Produit"}}}}}},"put":{"tags":["produit-rest-controller-
api"],"operationId":"update","parameters":[{"name":"id","in":"path","required":true,"schema":
{"type":"integer","format":"int64"}}],"requestBody":{"content":{"application/json":{"schema":
{"$ref":"#/components/schemas/Produit"}}},"required":true},"responses":{"200":{"description":"OK","content":{"*/*":{"schema":
{"$ref":"#/components/schemas/Produit"}}}}}},"delete":{"tags":["produit-rest-controller-api"],"operationId":"delete","parameters":
[{"name":"id","in":"path","required":true,"schema":{"type":"integer","format":"int64"}}],"responses":{"200":
{"description":"OK"}}}},"/stock/produits":{"get":{"tags":["produit-rest-controller-api"],"operationId":"compteList","responses":{"200":
{"description":"OK","content":{"application/json":{"schema":{"type":"array","items":
{"$ref":"#/components/schemas/Produit"}}},"application/xml":{"schema":{"type":"array","items":
{"$ref":"#/components/schemas/Produit"}}}}}}},"post":{"tags":["produit-rest-controller-api"],"operationId":"save","requestBody":{"content":
{"application/json":{"schema":{"$ref":"#/components/schemas/Produit"}}},"required":true},"responses":{"200":{"description":"OK","content":
{"*/*":{"schema":{"$ref":"#/components/schemas/Produit"}}}}}}},"/stock/produits/{byDesignation}":{"get":{"tags":["produit-rest-controller-
api"],"operationId":"findbyDesignationContains","parameters":[{"name":"des","in":"path","required":true,"schema":
{"type":"string"}}],"responses":{"200":{"description":"OK","content":{"*/*":{"schema":{"type":"array","items":
{"$ref":"#/components/schemas/Produit"}}}}}}}}},"components":{"schemas":{"Produit":{"type":"object","properties":{"id":
{"type":"integer","format":"int64"},"designation":{"type":"string"},"price":{"type":"number","format":"double"},"quantite":
{"type":"integer","format":"int32"}}}}}}
```

Importer dans Postman



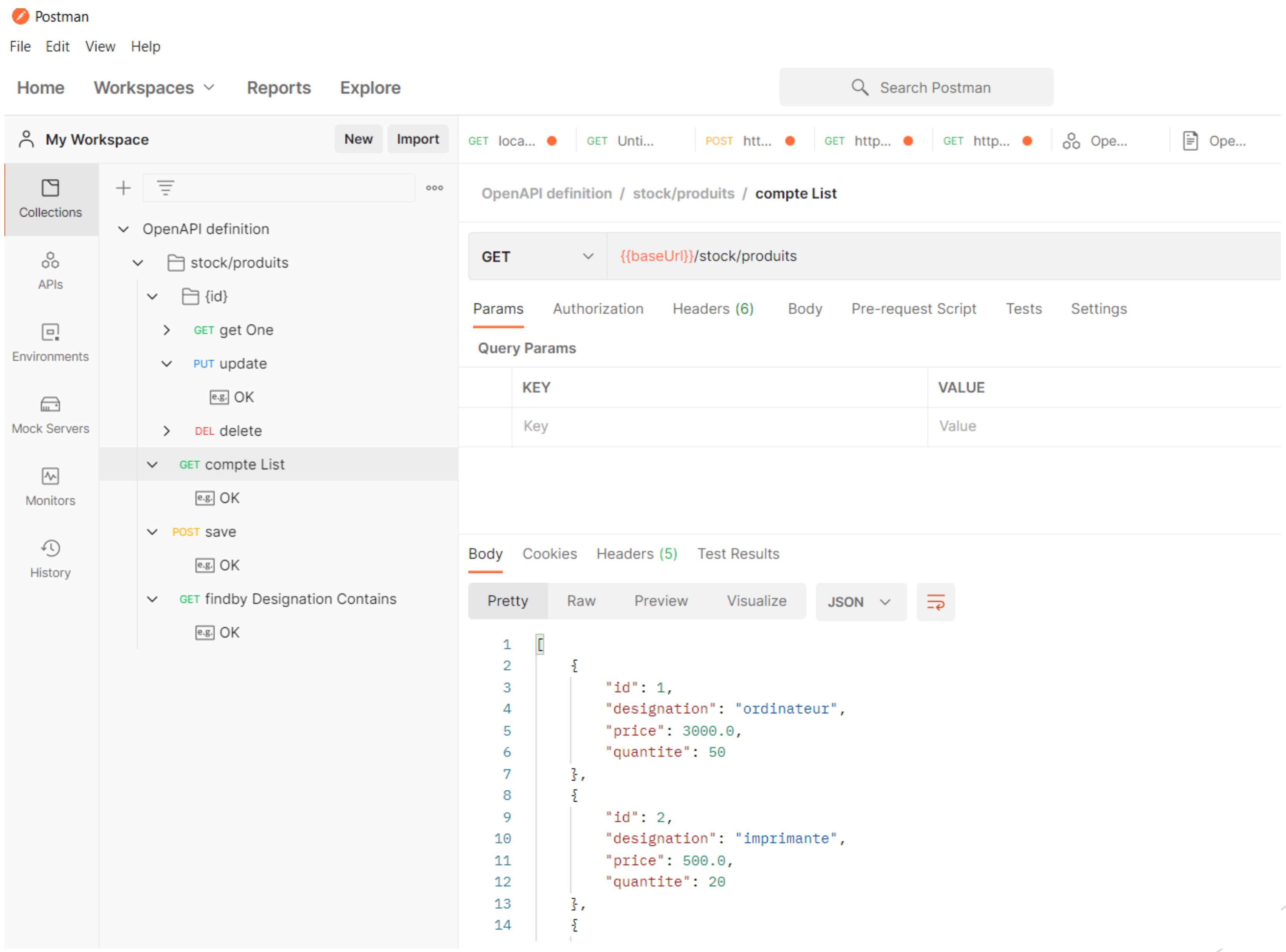
Import

File Folder Link Raw text Code repository New

Enter a URL

http://localhost:8082/v3/api-docs

Continue


Postman
File Edit View Help
Home Workspaces Reports Explore
Search Postman
My Workspace New Import
GET loca... GET Unti... POST htt... GET http... GET http... Ope... Ope...
Collections
APIs
Environments
Mock Servers
Monitors
History

OpenAPI definition
- stock/produits
  - {id}
    - GET get One
    - PUT update
      - OK
    - DEL delete
  - GET compte List
    - OK
  - POST save
    - OK
  - GET findby Designation Contains
    - OK

OpenAPI definition / stock/produits / **compte List**

GET {{baseUrl}}/stock/produits

Params Authorization Headers (6) Body Pre-request Script Tests Settings

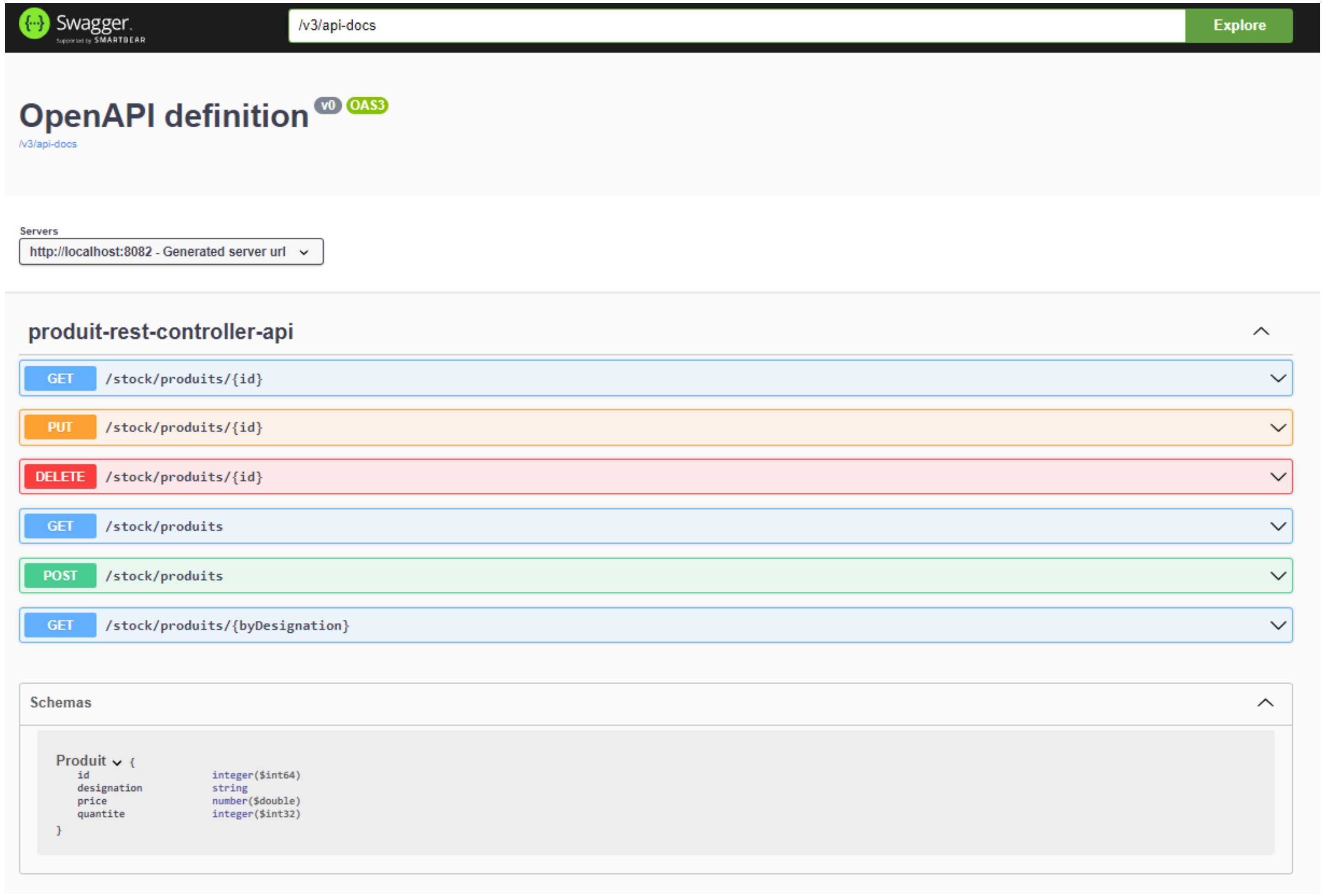
Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results
Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "designation": "ordinateur",
        "price": 3000.0,
        "quantite": 50
    },
    {
        "id": 2,
        "designation": "imprimante",
        "price": 500.0,
        "quantite": 20
    },
    {
```

http://localhost:8080/swagger-ui.html



OpenAPI definition v0 OAS3

/v3/api-docs

Servers

http://localhost:8082 - Generated server url

produit-rest-controller-api

GET /stock/produits/{id}

PUT /stock/produits/{id}

DELETE /stock/produits/{id}

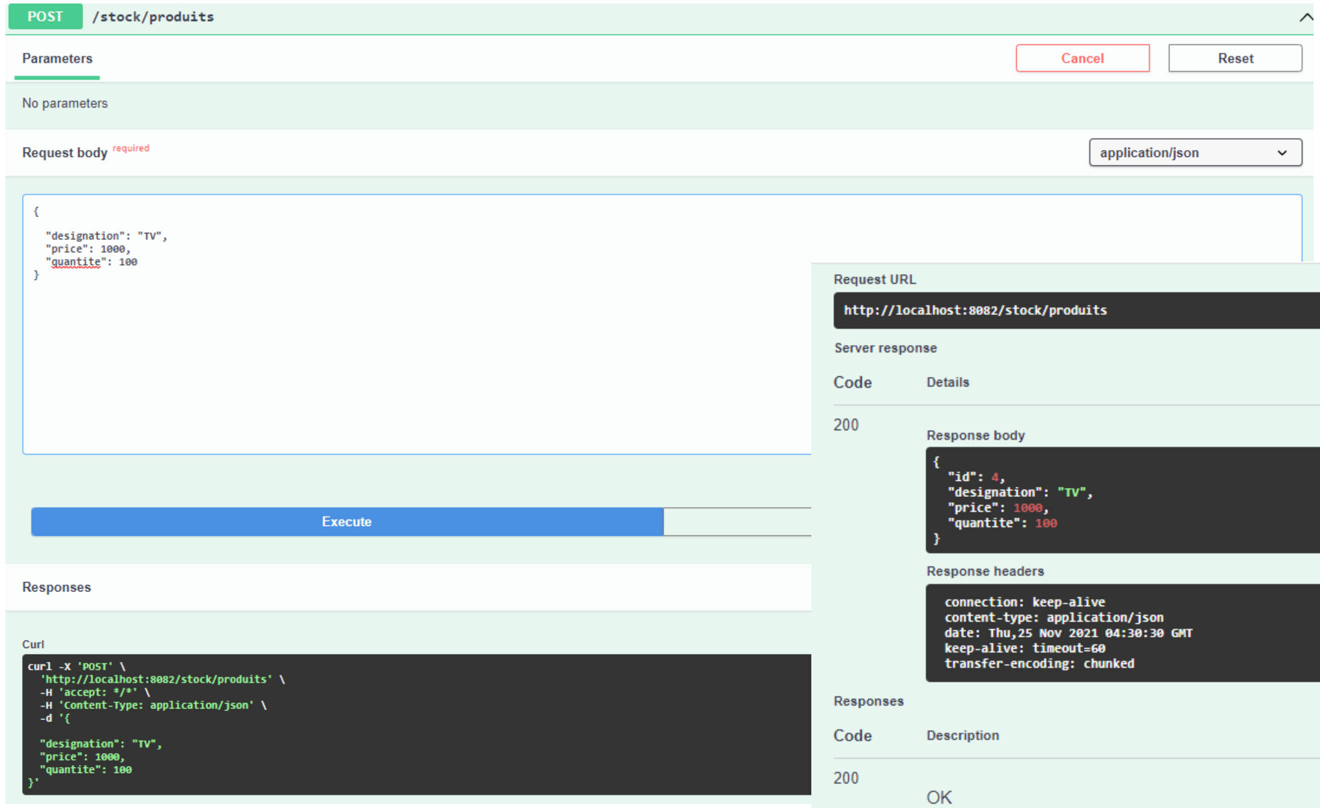
GET /stock/produits

POST /stock/produits

GET /stock/produits/{byDesignation}

Schemas

```
Produit {
    id              integer($int64)
    designation     string
    price           number($double)
    quantite        integer($int32)
}
```



POST /stock/produits

Parameters

Cancel Reset

No parameters

Request body required

application/json

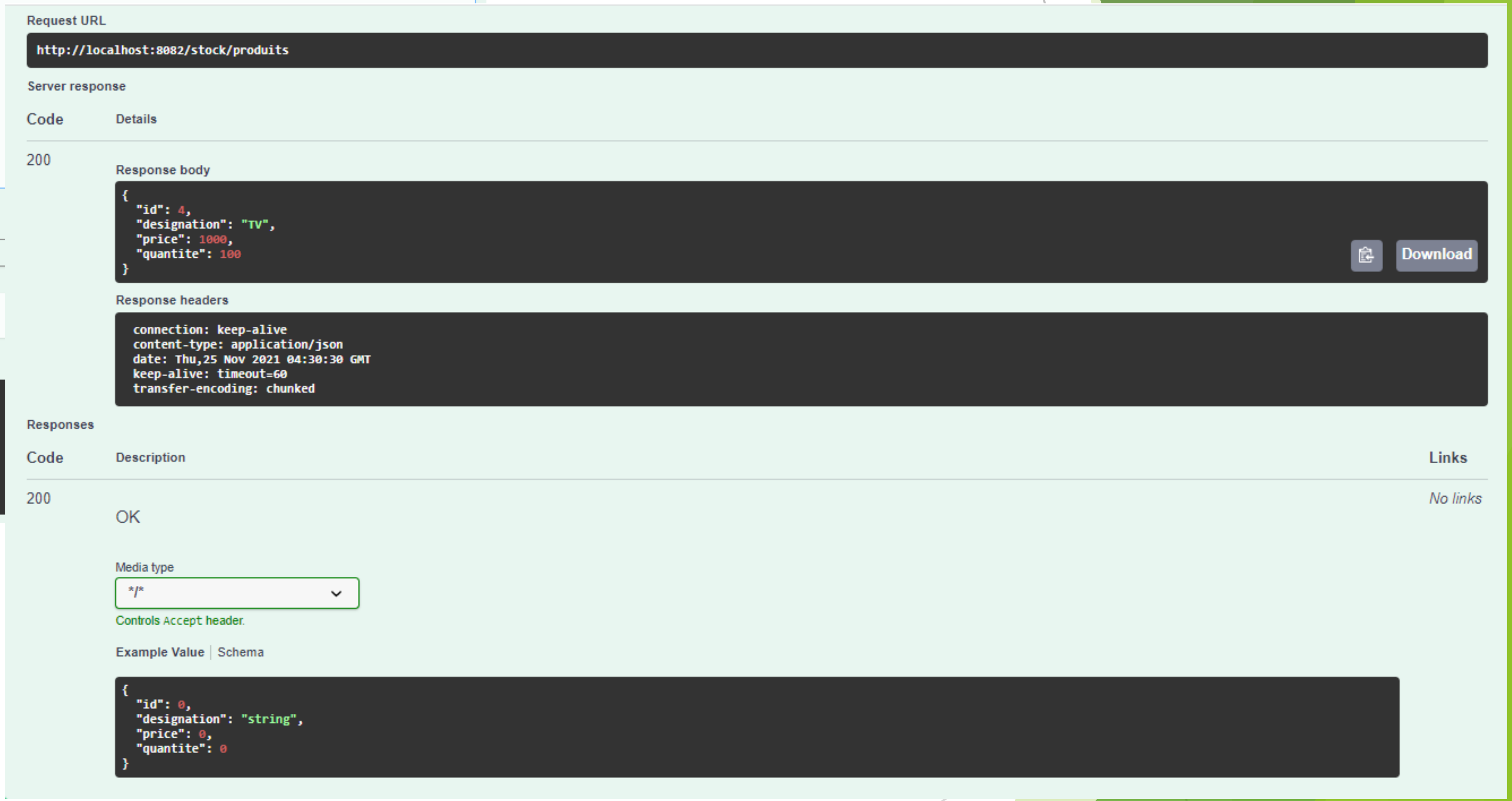
```
{
  "designation": "TV",
  "price": 1000,
  "quantite": 100
}
```

Execute

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8082/stock/produits' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{

  "designation": "TV",
  "price": 1000,
  "quantite": 100
}'
```



Request URL

```
http://localhost:8082/stock/produits
```

Server response

Code	Details
200	Response body

```
{
  "id": 4,
  "designation": "TV",
  "price": 1000,
  "quantite": 100
}
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Thu,25 Nov 2021 04:30:30 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	OK	No links

Media type

/

Controls Accept header.

Example Value | Schema

```
{
  "id": 0,
  "designation": "string",
  "price": 0,
  "quantite": 0
}
```

Consuming a Restful web service (Spring Boot)

Exemple

- On va développer un Rest service (CategoryRestWs) qui gère les catégories et qui va être fournie dans http://localhost:8081/
- On reprend notre premier exemple : ProduitRestWs (archive ProduitRestWS-Exemple1-v1) et on va le modifier en ajoutant un nouveau endpoint : Put produits/calculPrix qui permet de calculer le prix en utilisant les informations du Rest service Category(voir exmple 2 du service produit). Pour cela on va consommer le service web qui gère les catégories qui doit être accessible.

Microservice Category

Git clone https://github.com/ines-mouakher/CategoryRestWS.git

Microservice Category

```
@Entity
@Data //Getters et Setters
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class Category  {
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;
    private String designation;
    private float tax;
}
```

```
  public interface CategoryRepository  extends JpaRepository<Category,Long> {
  }
```

Microservice Category

```java
public interface CategoryService {
    public List<Category> categoryList();
    public Category getOne(Long id);
    public Category save(Category category);
    public Category update(Category category, Long id);
    public void delete(Long id);
}
```

```java
@Service
public class CategoryServiceImp implements CategoryService{

    @Autowired
    private CategoryRepository categoryRepository;

    @Override
    public List<Category> categoryList(){
        return categoryRepository.findAll();
    }

    @Override
    public Category getOne(Long id){
        return categoryRepository.findById(id).get();}

    @Override
    public Category save(Category category){
        return categoryRepository.save(category);
    }

    @Override
    public Category update(Category category, Long id){
        category.setId(id);
        return categoryRepository.save(category);
    }

    @Override
    public void delete(Long id){
        categoryRepository.deleteById(id);
    }
}
```

- Ajouter dans le fichier application.properties

```
spring.jpa.defer-datasource-initialization=true
spring.datasource.url=jdbc:h2:mem:cat
spring.h2.console.enable=true
server.port=8081
```

- Ajouter dans resources data.sql pour initialiser la BD

```sql
--INITIALISATION TABLE CATEGORY
INSERT INTO  Category(designation, tax) values ('alimentaire',15);
INSERT INTO  Category(designation, tax) values ('soin',10);
INSERT INTO  Category(designation, tax) values ('hytech',25);
```

Microservice Category

```java
@RestController
@RequestMapping("/apiv1/categories")// specifier un path pour le web service
public class CategoryRestController {

    @Autowired
    private CategoryService categoryService;


    @GetMapping
    public List<Category> categoryList() {

        return categoryService.categoryList();
    }


    @GetMapping(path = "/{id}")
    public Category getOne(@PathVariable Long id) {
        return categoryService.getOne(id);
    }

    @PostMapping
    public Category save(@RequestBody Category product) {

        return categoryService.save(product);
    }

    @PutMapping(path = "/{id}")
    public Category update(@RequestBody Category category, @PathVariable Long id) {
        return categoryService.update(category,id);
    }


    @DeleteMapping(path = "/{id}")
    public void delete(@PathVariable Long id) {
        categoryService.delete(id);
    }
}
```

Modifier Microservice Produit

Premier exemple Microservice Produit

- On reprend l' exemple 1 du Rest service Produit et on ajoute l'attribut categoryId dans la classe produit

```
@Entity
@Data //Getters et Setters
@NoArgsConstructor
@AllArgsConstructor
@ToString

public class Product {
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)

    private Long id;
    private String designation;
    private double price;
    private int quantity;
    private Long categoryId;
}
```

- Pour initialiser BD on modifie le fichier data.sql

```
--INITIALISATION TABLE PRODUCT
INSERT INTO  Product(designation, price, quantity,CATEGORY_ID) values ('ordinateur', 300, 50,1L);
INSERT INTO Product(designation, price, quantity,CATEGORY_ID) values ('imprimante', 500, 20,2L);
INSERT INTO Product(designation, price, quantity,CATEGORY_ID) values ('clavier', 100, 30,1L);
```

Consommer un REST web service

- Sprinboot vous simplifie la tache en proposant un proxy via la classe RestTemplate. Cette classe propose plusieurs méthodes, notamment:
  - getForObject(url, classType)
  - postForObject(url, request, classType)
  - put(url, request)
  - delete(url)
- Exemple:

```
RestTemplate restTemplate = new RestTemplate();
String result = restTemplate.getForObject(uri, String.class);
```

- Vous avez également des méthodes du type getForEntity(url, responseType) et execute(url, httpMethod, requestCallback, responseExtractor) vous permettant de récupérer des réponses complètes.

Create a domain class Category

- First, you need to create a domain class Category to contain the data that you need.

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Category {
    private Long id;
    private String designation;
    private float tax;
}
```

Ajouter l’opération calculPrix dans la couche service

- Dans la couche service ajouter l’opération calcul prix
  - Injecter template

```
@Autowired   private RestTemplate restTemplate;
```

  - Appeler l’opération gerForObjects

```
@Override
public double calculPrix(Long code, int qt) {
    Product p1= productRepository.findById(code).get();

    Category category =
           restTemplate.getForObject("http://localhost:8081/apiv1/categories/"+p1.getCategoryId(), Category.class);


    double res=(1+ (category.getTax()/100))*p1.getPrice()*qt;

    p1.setQuantity(p1.getQuantity()-qt);
         productRepository.save(p1);
    return res;}
```

Modifier le controleur

- Ajouter la classe CalculPrixRequestDTO

```
@Data
public class CalculPrixRequestDTO {
    private Long code;
    private int qt;
}
```

- Ajouter la nouvelle resouce URI

```
@PutMapping  (path="products/calculPrix")
public  double calculPrix(@RequestBody CalculPrixRequestDTO request){
    return
            productService.calculPrix(request.getCode(), request.getQt());
}
```

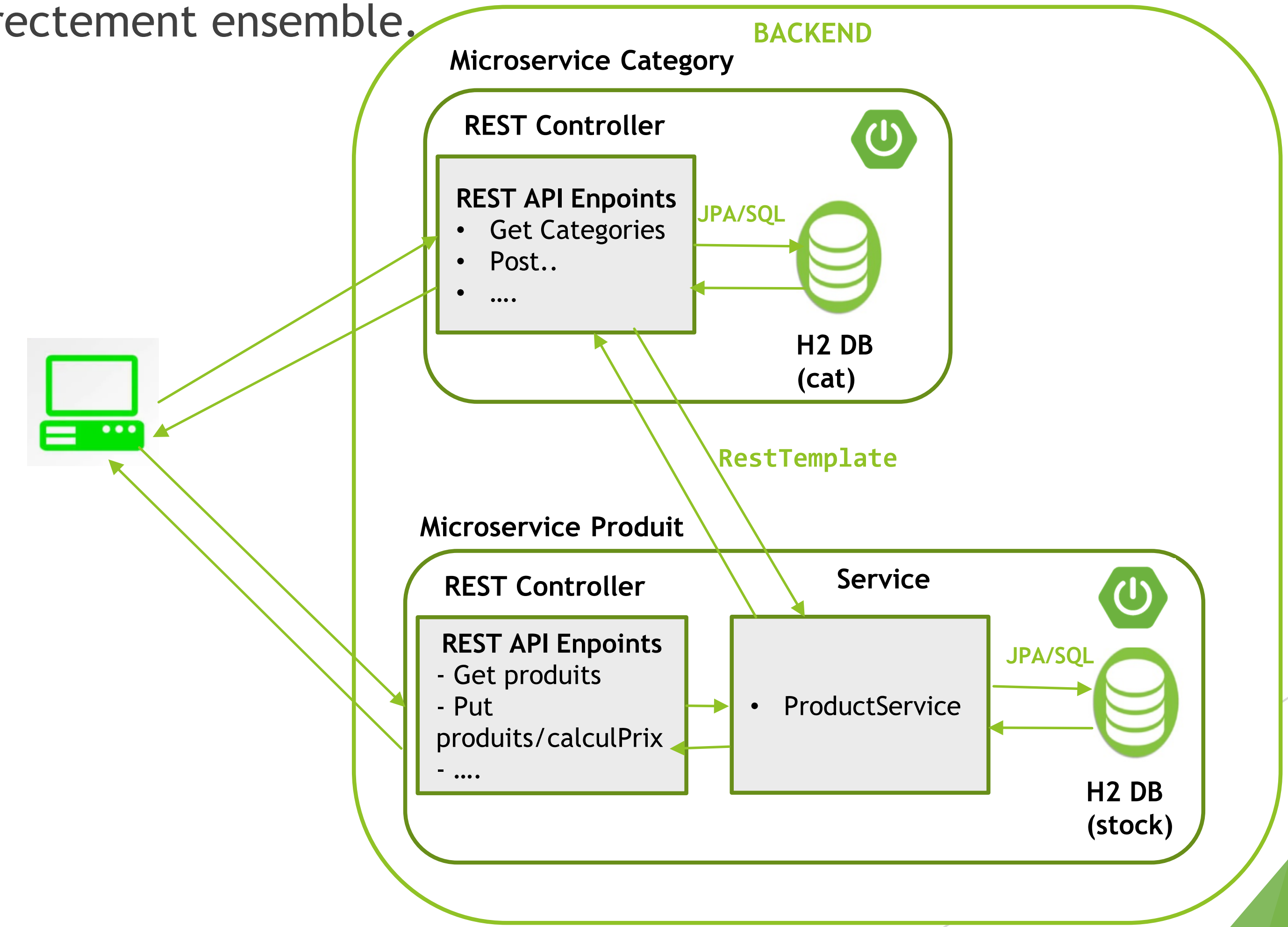
Add RestTemplate

- Now you need to add a few other things to the ProductRestWsApplication class:
  - A RestTemplate, which uses the Jackson JSON processing library to process the incoming data.

```
@Bean
public RestTemplate restTemplate(){
  return new RestTemplate();
}
```

Architecture

- Dans cette exemple on a développé deux microservice : product et category qui communiquent directement ensemble.



Producing a Restful web service (Spring Boot) - Partie II

Service Rest product (Spring Boot)

- Assume that we've already had a Spring Boot Application that exposes Rest APIs for a Tutorial application:

Methods	Urls	Actions
POST	/api/v1/products	create new Product
GET	/api/v1/products	retrieve all Products
GET	/api/v1/products/:id	retrieve a Product by :id
PUT	/api/v1/products/:id	update a Product by :id
DELETE	/api/v1/products/:id	delete a Product by :id

- git clone https://github.com/ines-mouakher/ProductRestWS-v0.git

API Parameters

- There are three common types of parameters to consider for your API:
- Filtering: Return only results that match a filter by using field age as parameters.
  - For example: GET /users?age=30
- Pagination: Don’t overload clients and servers by providing everything. Instead, set a limit and provide prev and next links in your response.
  - Example: GET /users?page=3&results_per_page=20
- Sorting: Provide a way to sort or some use cases will still require paging through all results to find what’s needed.
  - Example: GET /users?sort_by=first_name&order=asc
- These three approaches can be used together to support very specific queries. Understanding your use cases will help determine the complexity of your parameters.

Filtering

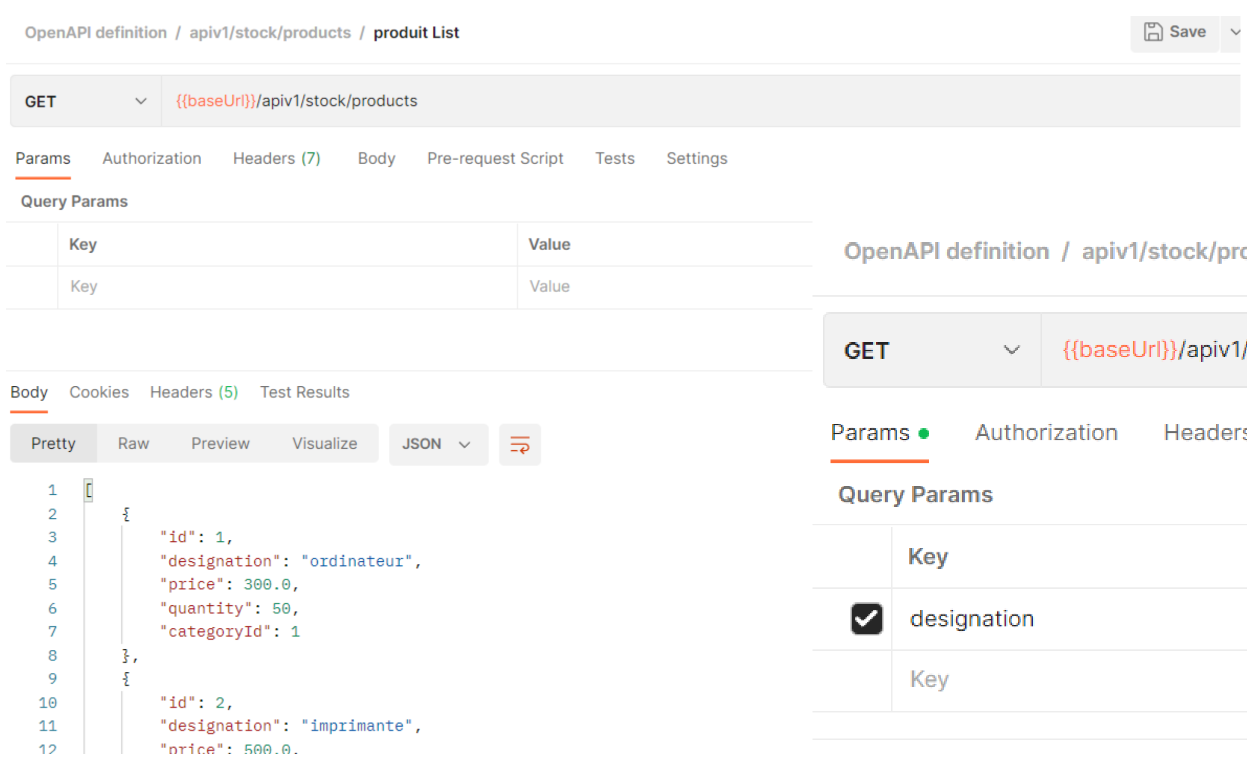
Methods	Urls	Actions
GET	/api/v1/products?designation=[keyword]	find all Producs which designation contains keyword

Filtering

```
@Repository
public interface ProductRepository  extends JpaRepository<Product,Long> {
public List<Product> findByDesignationContaining(String designation);
}
```

```
@Override
public List<Product> productList(){
   return productRepository.findAll();
}
@Override
public List<Product> findByDesignation(String designation){
   return productRepository.findByDesignationContaining(designation);
}
```

```
@Autowired
private ProductService productService;

@GetMapping(path = "/products", produces = {"application/json", "application/xml"})
public List<Product> produitList(@RequestParam(required = false) String designation) {
    if (designation== null)
        return productService.productList();
    else
        return productService.findByDesignation(designation);

}
```

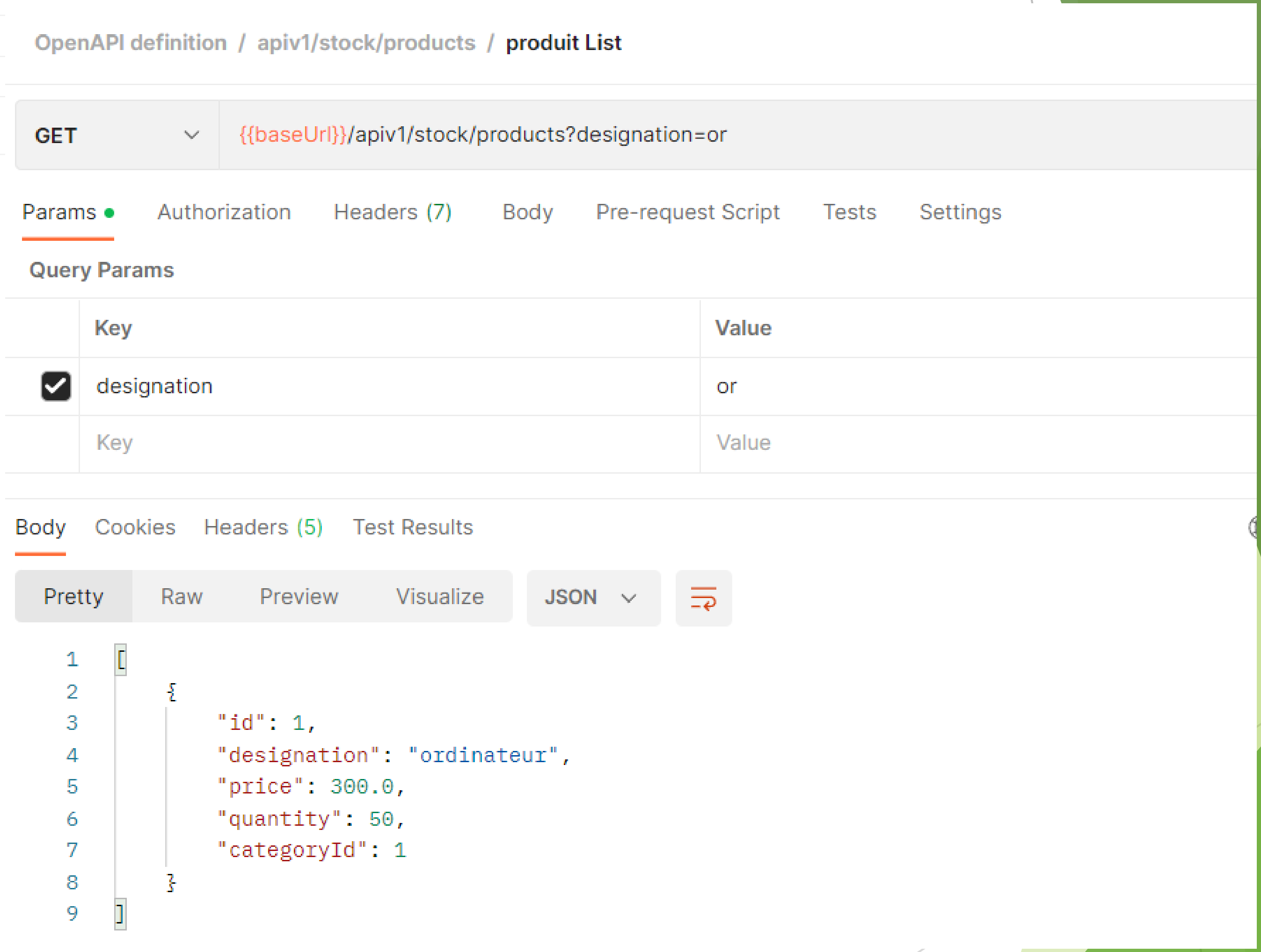

OpenAPI definition / apiv1/stock/products / produit List

`{{baseUrl}}/apiv1/stock/products`

Query Params

Key	Value
Key	Value

```
[
    {
        "id": 1,
        "designation": "ordinateur",
        "price": 300.0,
        "quantity": 50,
        "categoryId": 1
    },
    {
        "id": 2,
        "designation": "imprimante",
        "price": 500.0,
```


OpenAPI definition / apiv1/stock/products / produit List

`{{baseUrl}}/apiv1/stock/products?designation=or`

Query Params

	Key	Value
✓	designation	or
	Key	Value

```
[
    {
        "id": 1,
        "designation": "ordinateur",
        "price": 300.0,
        "quantity": 50,
        "categoryId": 1
    }
]
```

Pagination & Sorting

Methods	Urls	Actions
GET	/api/v1/products?designation=[keyword]&pageNo=0&pageSize=3&sortBy=id&sortDir=desc	find all Producs which designation contains keyword

Pagination & Sorting

```
@Repository
public interface ProductRepository  extends JpaRepository<Product,Long> {
   public Page<Product> findByDesignationContaining(String designation, Pageable pageable);
}
```

```
  @Override
  public ProductResponse productList(int pageNo,int pageSize,String sortBy,String sortDir){
      Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
              : Sort.by(sortBy).descending();

      // create Pageable instance
      Pageable paging = PageRequest.of(pageNo,pageSize, sort);
      Page<Product> pageProds =productRepository.findAll(paging);

      // get content for page object
      List<Product> listofProduct = pageProds.getContent();


      ProductResponse response = new ProductResponse();
      response.setContent(listofProduct);
      response.setPageNo(pageProds.getNumber());
      response.setPageSize(pageProds.getSize());
      response.setTotalElements(pageProds.getTotalElements());
      response.setTotalPages(pageProds.getTotalPages());
      response.setLast(pageProds.isLast());


      return response ;
  }
```

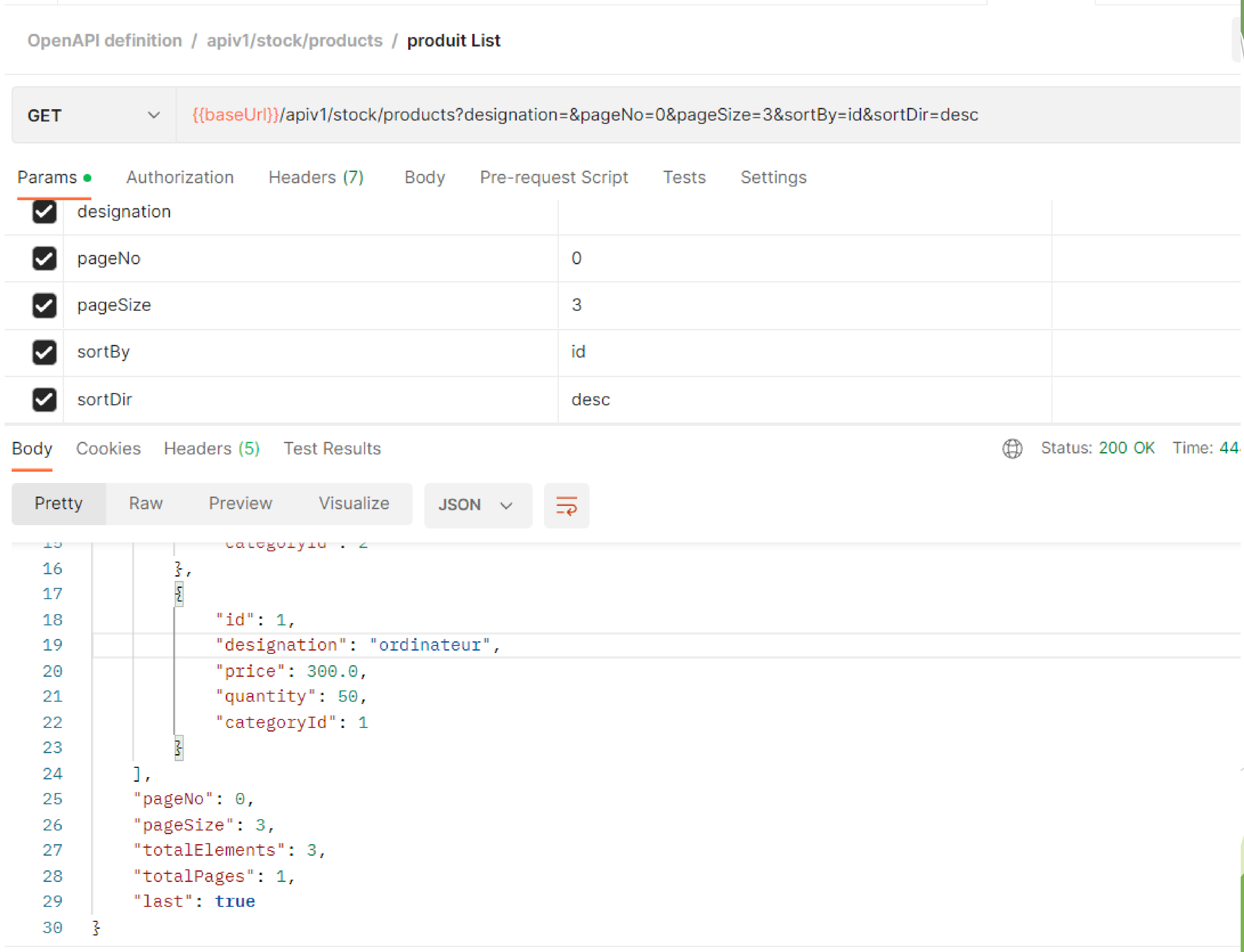
```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class ProductResponse {
    private List<Product> content;
    private int pageNo;
    private int pageSize;
    private long totalElements;
    private int totalPages;
    private boolean last;
}
```

Pagination & Sorting

```java
@Override
public ProductResponse findByDesignation(String designation, int pageNo,int pageSize,String sortBy,String sortDir){
    Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
            : Sort.by(sortBy).descending();

    // create Pageable instance
    Pageable paging = PageRequest.of(pageNo,pageSize, sort);
    Page<Product> pageProds =productRepository.findByDesignationContaining(designation, paging);

    // get content for page object
    List<Product> listofProduct = pageProds.getContent();


    ProductResponse response = new ProductResponse();
    response.setContent(listofProduct);
    response.setPageNo(pageProds.getNumber());
    response.setPageSize(pageProds.getSize());
    response.setTotalElements(pageProds.getTotalElements());
    response.setTotalPages(pageProds.getTotalPages());
    response.setLast(pageProds.isLast());
    return response ;
}
```

Pagination & Sorting

```
@GetMapping(path = "/products", produces = {"application/json", "application/xml"})
public ProductResponse produitList(
        @RequestParam(required = false) String designation,
        @RequestParam(defaultValue = "0", required = false) int pageNo,
        @RequestParam(defaultValue = "3", required = false) int pageSize,
        @RequestParam(defaultValue = "id", required = false) String sortBy,
        @RequestParam(defaultValue = "desc", required = false) String sortDir )

{
  if (designation== null)
     return productService.productList(pageNo,pageSize, sortBy, sortDir);
  else
      return productService.findByDesignation(designation, pageNo,pageSize, sortBy, sortDir);

}
```



OpenAPI definition / apiv1/stock/products / produit List

GET {{baseUrl}}/apiv1/stock/products?designation=&pageNo=0&pageSize=3&sortBy=id&sortDir=desc

✓ designation	
✓ pageNo	0
✓ pageSize	3
✓ sortBy	id
✓ sortDir	desc

Status: 200 OK Time: 44

```
            "categoryId": 2
        },
        {
            "id": 1,
            "designation": "ordinateur",
            "price": 300.0,
            "quantity": 50,
            "categoryId": 1
        }
    ],
    "pageNo": 0,
    "pageSize": 3,
    "totalElements": 3,
    "totalPages": 1,
    "last": true
}
```

DTO (Data Transfer Object)

- À l’heure actuelle, notre API web expose les entités de base de données au client. Le client reçoit des données qui correspondent directement à vos tables de base de données. Cependant, ce n’est pas toujours une bonne idée. Parfois, vous souhaitez modifier la forme des données que vous envoyez au client. Vous pouvez, par exemple, souhaiter effectuer les opérations suivantes :
  - Masquer des propriétés particulières que les clients ne sont pas censés afficher.
  - Omettez certaines propriétés pour réduire la taille de la charge utile.
  - Évitez les vulnérabilités de « sur-publication ».
  - Dissociez votre couche de service de votre couche de base de données….
- Pour ce faire, vous pouvez définir un *objet de transfert de données* (DTO). Un DTO est un objet qui définit la façon dont les données seront envoyées sur le réseau. Voyons comment cela fonctionne avec l’entité Product.

DTO (Data Transfer Object)

- On ajoute la dépendance suivante dans pom.xml

```
<!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
<dependency>
   <groupId>org.modelmapper</groupId>
   <artifactId>modelmapper</artifactId>
   <version>3.1.1</version>
</dependency>
```

- On crée une nouvelle classe ProductDTO

```
@Data //Getters et Setters
@NoArgsConstructor
@AllArgsConstructor

public class ProductDTO {
    private Long id;
    private String designation;
    private double price;
    private int quantity;
}
```

- Dans la classe ProductRestWsApplication on ajoute:

```
@Bean
public ModelMapper modelMapper() {
   return new ModelMapper();
}
```

DTO - Couche controler

```
 @GetMapping(path = "/products/{id}")
public ProductDTO getOne(@PathVariable Long id) {
       return productService.getOne(id);
 }

 @PostMapping(path = "/products")
 public ProductDTO save(@RequestBody ProductDTO productDto) {

     return productService.save(productDto);
 }
 @PutMapping(path = "/products/{id}")
 public ProductDTO update(@RequestBody ProductDTO productDto, @PathVariable Long id) {
    return productService.update(productDto,id);
 }
```

DTO - Couche service

```java
@Override
public ProductResponse productList(int pageNo,int pageSize,String sortBy,String sortDir){
    Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()
            : Sort.by(sortBy).descending();

    // create Pageable instance
    Pageable paging = PageRequest.of(pageNo,pageSize, sort);
    Page<Product> pageProds =productRepository.findAll(paging);

    // get content for page object
    List<Product> listofProduct = pageProds.getContent();


    List <ProductDTO> listofProductDto = new ArrayList< ProductDTO >();
    if (!listofProduct.isEmpty()) {
        for (Product p: listofProduct ) {
            listofProductDto.add(modelMapper.map(p, ProductDTO.class));
        }
    }
    ProductResponse response = new ProductResponse();
    response.setContent(listofProductDto);
    response.setPageNo(pageProds.getNumber());
    response.setPageSize(pageProds.getSize());
    response.setTotalElements(pageProds.getTotalElements());
    response.setTotalPages(pageProds.getTotalPages());
    response.setLast(pageProds.isLast());


    return response ;
}
```

```java
@Override
public ProductDTO  getOne(Long id){
    Product product = productRepository.findById(id).get();

    // convert entity to DTO
    ProductDTO productResponse = modelMapper.map(product ,
ProductDTO.class);
    return productResponse;
}

@Override
public ProductDTO save(ProductDTO productDto){
    // convert entity to DTO
    Product product = modelMapper.map(productDto, Product.class);
    Product newproduct  = productRepository.save(product);

    // convert entity to DTO
    ProductDTO productResponse = modelMapper.map(newproduct ,
ProductDTO.class);
    return productResponse;
}
```

Renvoyez les bons codes HTTP

- Quand vous avez testé votre méthode save, vous avez dû remarquer que celle-ci renvoie un code 200 OK une fois exécutée. Or, selon les standards du protocole HTTP, en cas de requête POST pour créer une ressource sur le serveur distant, il faut définir, dans l'en-tête de la réponse :
  - le code 201 Created
- Tout d'abord, nous remplaçons la valeur de retour ProductDTO de la méthode ajouterProduit par ResponseEntity< ProductDTO>.
  - ResponseEntity est une classe qui hérite de HttpEntity, qui permet de définir le code HTTP à retourner. L'intérêt de ResponseEntity est de nous donner la main pour personnaliser le code facilement.

Renvoyez les bons codes HTTP

```
@GetMapping(path = "/products/{id}")
public  ResponseEntity<ProductDTO> getOne(@PathVariable Long id) {
    try {

       Optional<ProductDTO> p =productService.getOne(id);
        if (!p.isEmpty())
            return new ResponseEntity<>(p.get(),HttpStatus.OK);
        else
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);

    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}

 @PostMapping(path = "/products")
 public ResponseEntity<ProductDTO> save(@RequestBody ProductDTO productDto) {
    try {
    return new ResponseEntity<>(productService.save(productDto), HttpStatus.CREATED);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

```
@GetMapping(path = "/products", produces = {"application/json", "application/xml"})
public ResponseEntity<ProductResponse> produitList(
        @RequestParam(required = false) String designation,
        @RequestParam(defaultValue = "0", required = false) int pageNo,
        @RequestParam(defaultValue = "3", required = false) int pageSize,
        @RequestParam(defaultValue = "id", required = false) String sortBy,
        @RequestParam(defaultValue = "desc", required = false) String sortDir )

{ try {
    ProductResponse res;
  if (designation== null)
      res=productService.productList(pageNo,pageSize, sortBy, sortDir);
  else
       res=productService.findByDesignation(designation, pageNo,pageSize, sortBy, sortDir);

    if (res.getContent().isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    else {
        return new ResponseEntity<>(res, HttpStatus.OK);
    }

} catch (Exception e) {
    return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
}
}
```

Hypermedia As The Engine Of Application State (HATEOAS)

- Hypermedia is an important aspect of REST. It lets you build services that decouple client and server to a large extent and let them evolve independently. The representations returned for REST resources contain not only data but also links to related resources. Thus, the design of the representations is crucial to the design of the overall service.
- **Spring HATEOAS** provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC. The core problem it tries to address is link creation and representation assembly. (https://spring.io/projects/spring-hateoas)

Hypermedia As The Engine Of Application State (HATEOAS)

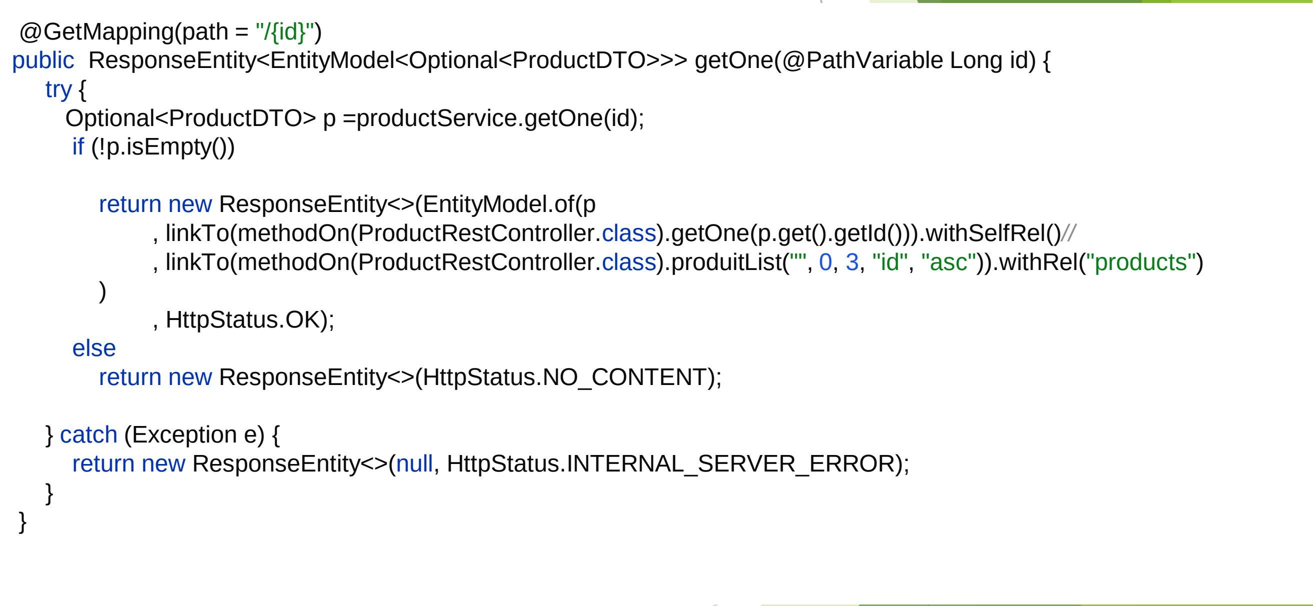
```
<!-- hateoas dependency-->

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

<!-- hateoas dependency-->
```

Spring HATEOAS

- Spring HATEOAS defines a generic EntityModel<T> container that lets you store any domain object (Produit in this example), and add additional links.

Spring HATEOAS

- It retrieves a collection of Produit objects, streams through a Java 8 spliterator, and converts them into a collection of EntityModel<Employee> objects by using Spring HATEOAS's linkTo and methodOn helpers to build links.
- The natural convention with REST endpoints is to serve a self link (denoted by the .withSelfRel() call).
- It's also useful for any single item resource to include a link back to the aggregate (denoted by the .withRel("produits")).
- The whole collection of single item resources is then wrapped in a Spring HATEOAS Resources type

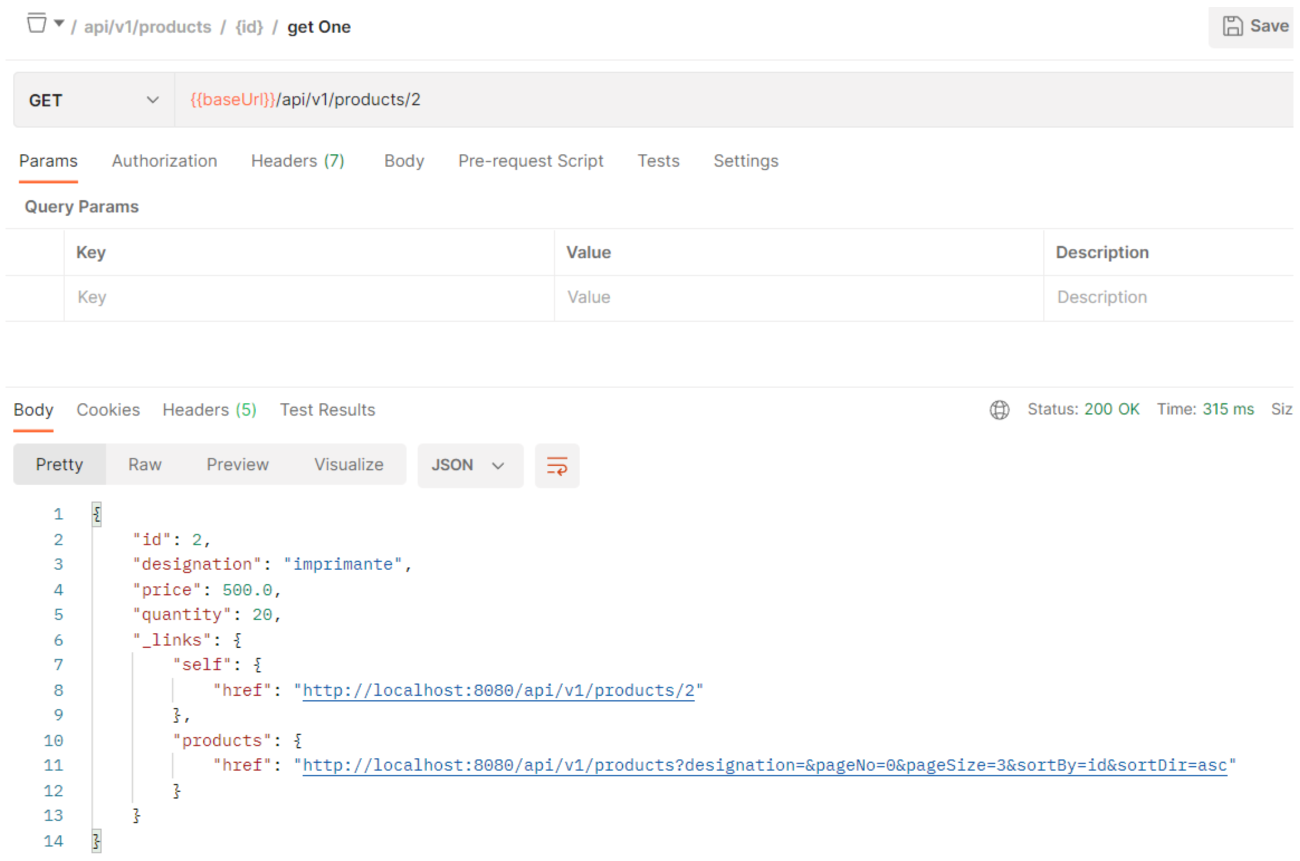


```
 @GetMapping(path = "/{id}")
public  ResponseEntity<EntityModel<Optional<ProductDTO>>> getOne(@PathVariable Long id) {
    try {
       Optional<ProductDTO> p =productService.getOne(id);
        if (!p.isEmpty())

            return new ResponseEntity<>(EntityModel.of(p
                    , linkTo(methodOn(ProductRestController.class).getOne(p.get().getId())).withSelfRel()//
                    , linkTo(methodOn(ProductRestController.class).produitList("", 0, 3, "id", "asc")).withRel("products")
            )
                    , HttpStatus.OK);
        else
            return new ResponseEntity<>(HttpStatus.NO_CONTENT);

    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
 }
```

Spring HATEOAS

```
@GetMapping(produces = {"application/json", "application/xml"})
public  ResponseEntity<CollectionModel<EntityModel<ProductDTO>>> produitList(
        @RequestParam(required = false) String designation,
        @RequestParam(defaultValue = "0", required = false) int pageNo,
        @RequestParam(defaultValue = "3", required = false) int pageSize,
        @RequestParam(defaultValue = "id", required = false) String sortBy,
        @RequestParam(defaultValue = "desc", required = false) String sortDir )
{ try {
    ProductResponse res;
  if (designation== null)
      res=productService.productList(pageNo,pageSize, sortBy, sortDir);
  else
       res=productService.findByDesignation(designation, pageNo,pageSize, sortBy, sortDir);

    if (res.getContent().isEmpty()) {
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    else {
        List<EntityModel<ProductDTO>> produits =
                StreamSupport.stream(res.getContent().spliterator(), false)
                .map(produit -> EntityModel.of(produit, //
                        linkTo(methodOn(ProductRestController.class).getOne(produit.getId())).withSelfRel(), //
                        linkTo(methodOn(ProductRestController.class).produitList("",0,3,"id","asc")).withRel("products")
                )) //
                .collect(Collectors.toList());

        return new ResponseEntity<>(//
              CollectionModel.of(produits, //
                        linkTo(methodOn(ProductRestController.class).produitList("",0,3,"id","asc")).withRel("products")
                )
                , HttpStatus.OK);
    }

} catch (Exception e) {
    return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
}
}
```

Tester avec Postman : http://localhost:8080/api/v1/products/2

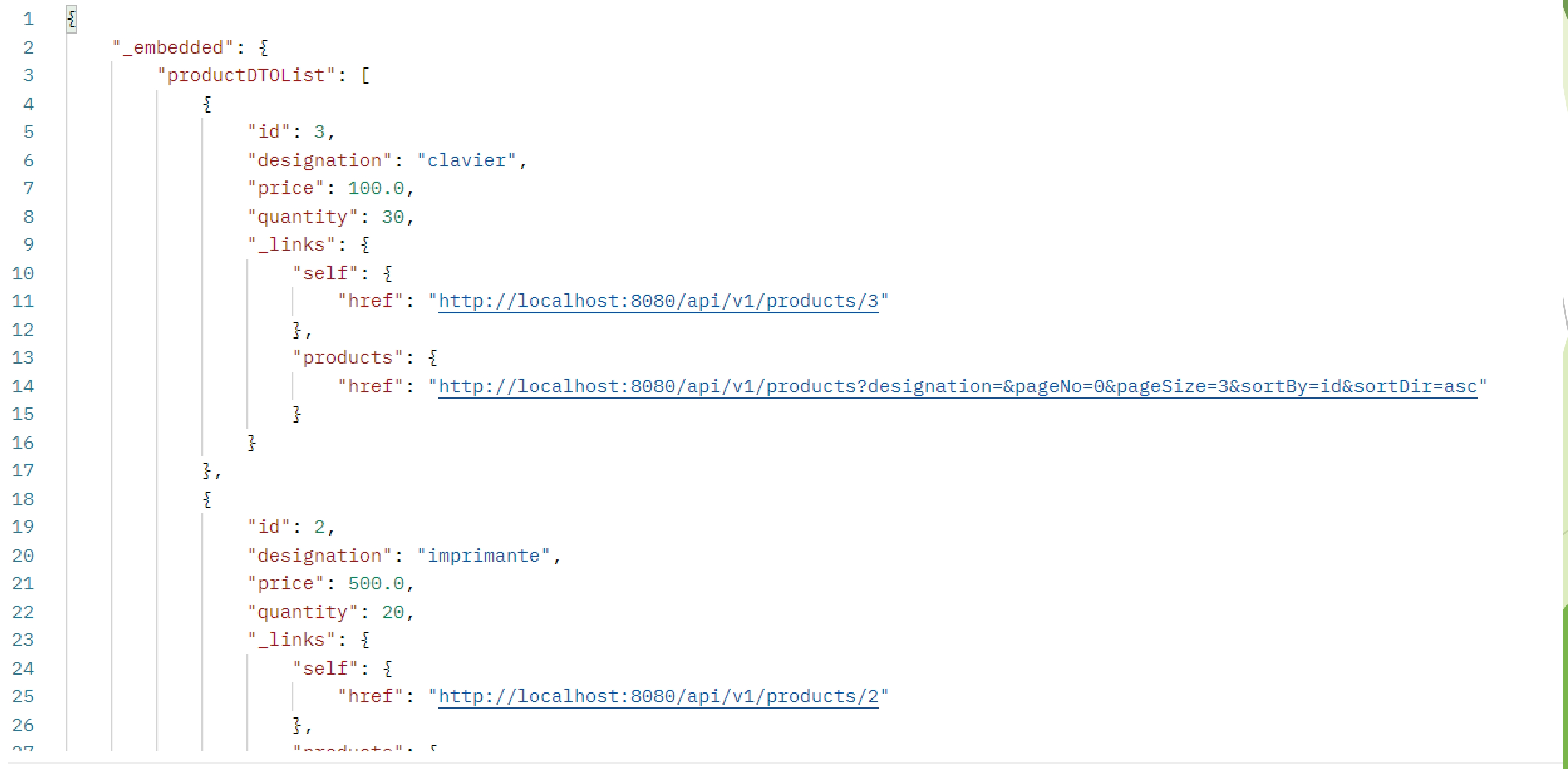


GET {{baseUrl}}/api/v1/products/2

Status: 200 OK Time: 315 ms Siz

```
{
    "id": 2,
    "designation": "imprimante",
    "price": 500.0,
    "quantity": 20,
    "_links": {
        "self": {
            "href": "http://localhost:8080/api/v1/products/2"
        },
        "products": {
            "href": "http://localhost:8080/api/v1/products?designation=&pageNo=0&pageSize=3&sortBy=id&sortDir=asc"
        }
    }
}
```

Tester avec Postman : http://localhost:8080/api/v1/products



```
{
    "_embedded": {
        "productDTOList": [
            {
                "id": 3,
                "designation": "clavier",
                "price": 100.0,
                "quantity": 30,
                "_links": {
                    "self": {
                        "href": "http://localhost:8080/api/v1/products/3"
                    },
                    "products": {
                        "href": "http://localhost:8080/api/v1/products?designation=&pageNo=0&pageSize=3&sortBy=id&sortDir=asc"
                    }
                }
            },
            {
                "id": 2,
                "designation": "imprimante",
                "price": 500.0,
                "quantity": 20,
                "_links": {
                    "self": {
                        "href": "http://localhost:8080/api/v1/products/2"
                    },
```

Lien

- https://spring.io/guides/gs/rest-service/
- https://spring.io/guides/gs/rest-hateoas/
- https://spring.io/projects/spring-hateoas#learn