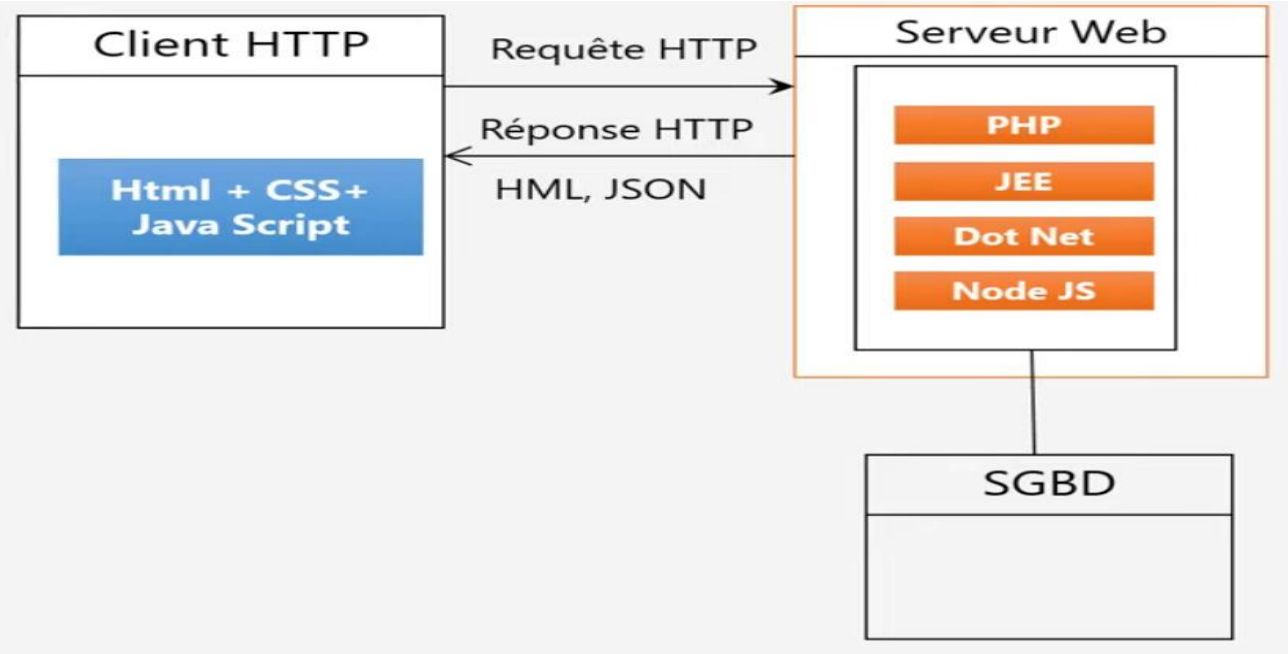
Formation full stack Spring boot- Angular

Spring boot

Architecture Web



- Une application web se compose de deux parties:
  - La partie **Backend** : S'occupe des traitements effectués coté serveur : Technologies utilisées : PHP, JEE, .Net, Node JS
  - La partie **Frontend** : S'occupe de la présentations des IHM coté Client : Langages utilisés : HTML, CSS, Java Script
- La communication entre la partie Frontend et la partie backend se fait en utilisant le protocole HTTP

Exigence d'un projet

- Exigences Fonctionnelles (métier):satisfaire les besoins fonctionnels de l'entreprise
- Exigences Techniques
  - Performances : Temps de réponse, **Problème de montée en charge**: scalabilité, Equilibrage de charge
  - Maintenance: l'application doit être facile à maintenir (évoluer dans le temps, l'application doit être fermée à la modification et ouverte à l'extension)
  - Sécurité
  - Persistances des données, Gestion des transactions
  - Versions: Web, Mobile, Desktop
- Exigences financières

Constat

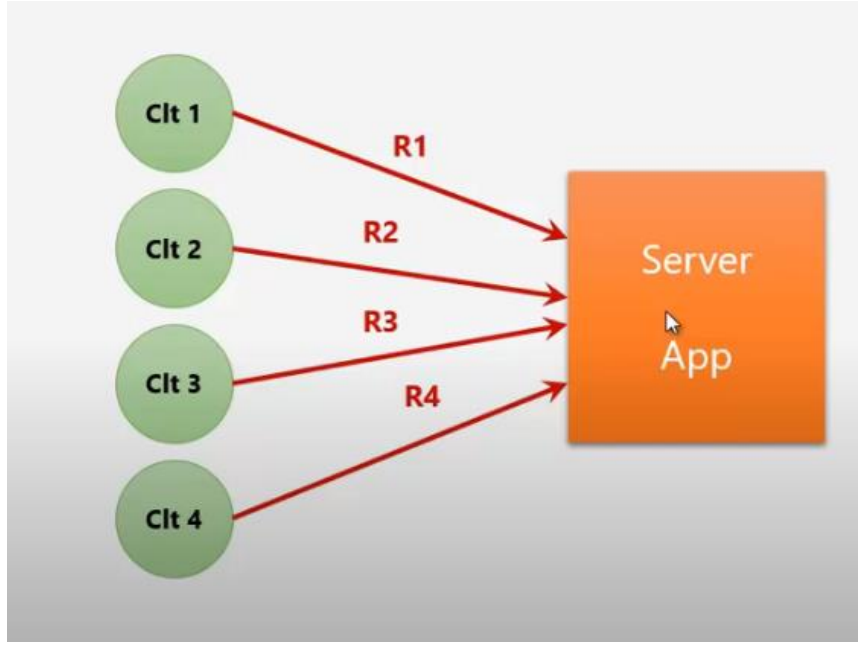
- Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres :
- Bâtir l'application sur une architecture d'entreprise: JEE

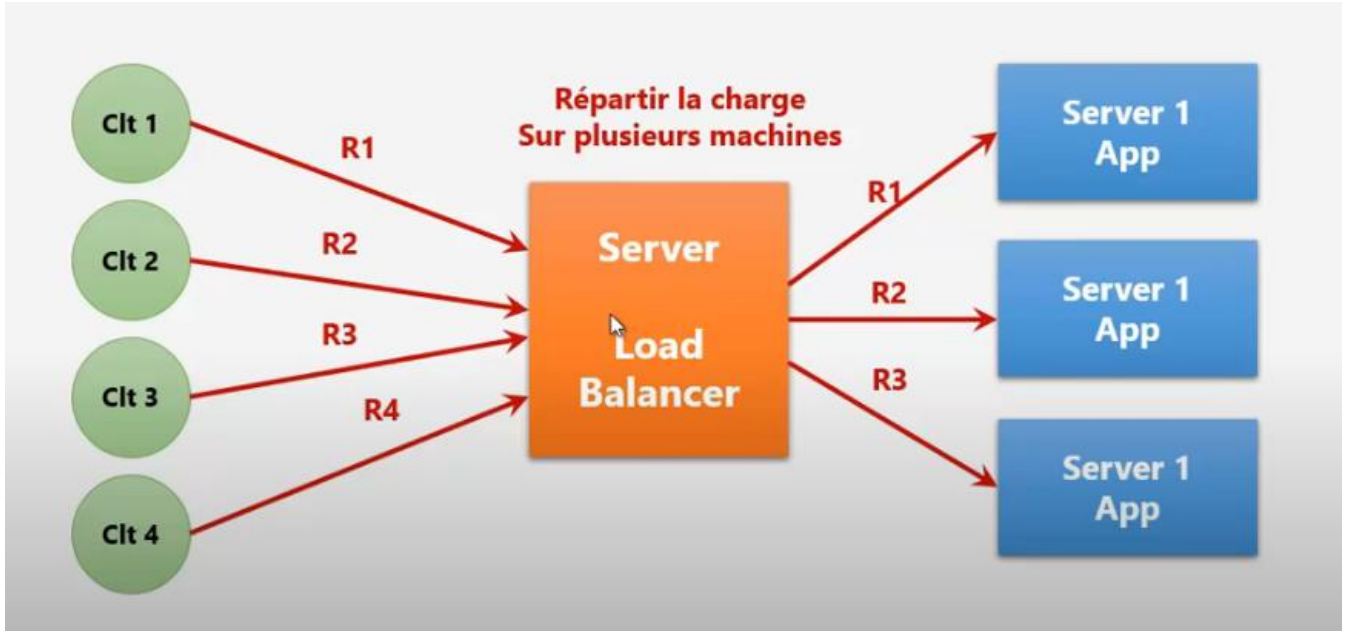
  Framework pour l'Inversion de contrôle:

  Permettre au développeur de s'occuper uniquement du code métier (Exigences fonctionnelles) et c'est le Framework qui s'occupe du code technique (Exigences Techniques)

  - **Spring (Conteneur léger)**
  - **EJB (Conteneur lourd)**
- Frameworks :
  - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
  - Applications Web : Struts, JSF, SpringMVC

Problème de montée en charge





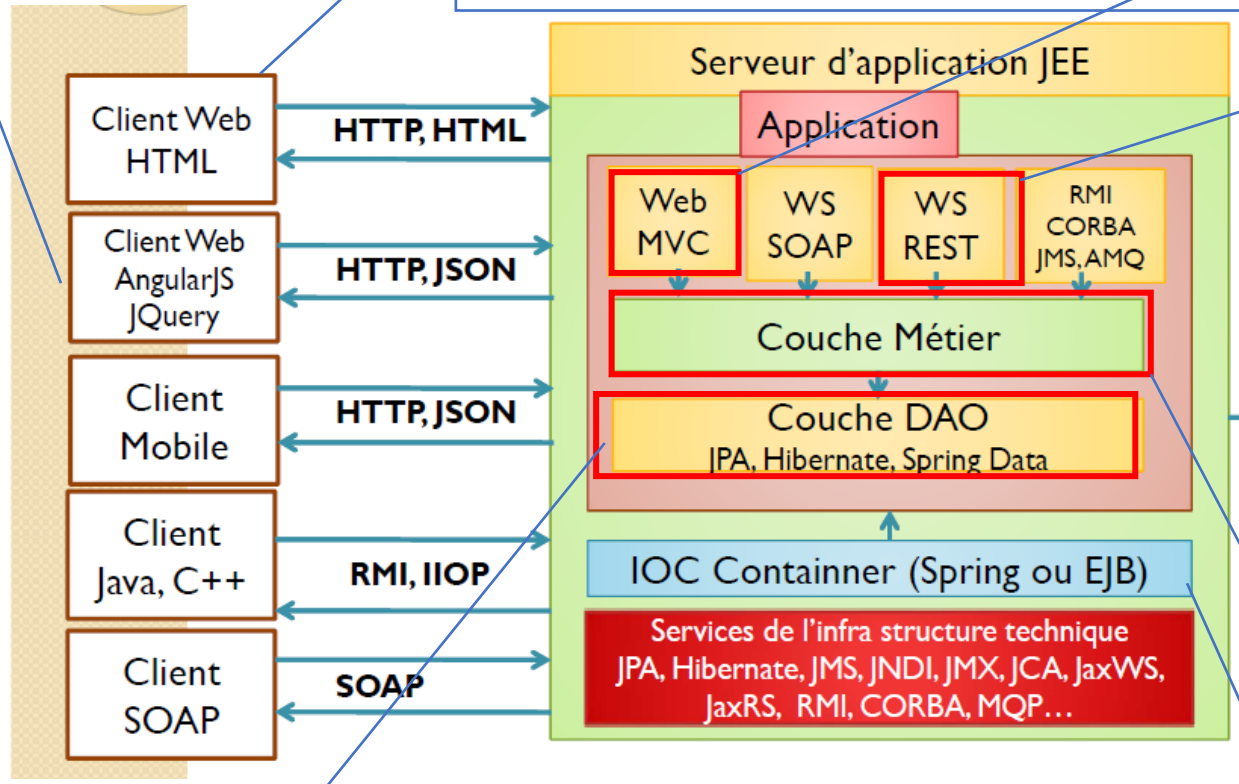
Architecture d'une application

- Envoie les requête HTTP au service REST
- Ce dernier retourne une réponse HTTP qui contient des données au format JSON
- Dans ce cas c'est le client HTTP qui s'occupe de la présentation
- Souvent on utilise un Framework Java Script comme Angular

Envoie les requête HTTP à la couche Web de l'application
Reçois les réponse HTTP contenant du HTML, CSS et Java Script
Dans ce scénario, tout est généré coté serveur

- Permet d'implémenter la logique présentation Web de l'application (Servlet et JSP)
- Dans cette partie on utilise des Framework MVC comme Spring MVC
- Cette couche reçoit le requêtes HTTP du client web
- Faire appel à la couche métier pour effectuer les traitement
- Envoie un résultat HTML au client, en utilisant un moteur de Template comme Tymeleaf dans une réponse HTTP

Un serveur d'application qui permet de déployer les applications



Couche Web Services REST full:

- Permet de définir un web service qui permet à d'autres applications développées avec d'autres langages de faire appel, à distance, aux fonctionnalités de l'application
- Le service REST reçoit des requête HTTP
- Exécute des traitement et renvoie le résultat au client avec différents formats (JSON, XML)

SGBDR Ou NoSQL

Permet de stocker les données de l'application.
Si la quantité de données sont très importante, on utilise des SGBD NoSQL (Not Only SQL) comme MangoDB, …

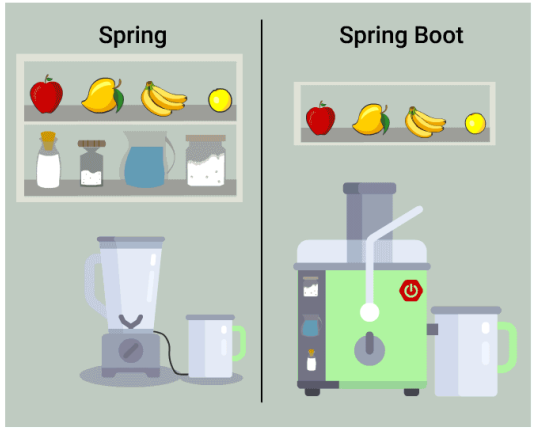
Permet d'implémenter la logique métier de l'application

Couche technique qui représente la couche d'accès aux données de l'application.
Si les données sont stockées dans une base de données relationnelle, cette couche utilise un Framework de Mapping Objet relationnel implémentant la spécification JPA comme Hibernate, TopLink, etc..

Représente le Framework qui permet de faire l'inversion de contrôle

- Permet à l'application de se concentrer uniquement sur le code métier (Exigences fonctionnelles)
- Le Container IOC offre à l'applications les services techniques (Sécurité, Gestion de transaction, ORM, etc..)

Spring vs Spring Boot



Spring	Spring Boot
• **L**e framework Spring est l'un des frameworks les plus populaires pour le développement des applications en Java. • Il vise à simplifier le développement Java EE qui rend les développeurs plus productifs. • Les principales caractéristiques de Spring Framework sont **l'inversion de control et** l' **injection de dépendances** . • Cela aide à simplifier les choses en nous permettant de développer **des** applications **faiblement couplées** . • Pour tester le projet Spring, nous devons configurer le serveur explicitement. • Les développeurs définissent manuellement les dépendances pour le projet Spring dans **pom.xml** .	• Spring Boot est une extension du framework Spring, • **Spring Boot Framework** est largement utilisé pour développer **des API REST** • Il vise à raccourcir la longueur du code et à fournir le moyen le plus simple de développer **des applications Web** . • La principale caractéristique de Spring Boot est la **configuration automatique** . Il configure automatiquement les classes en fonction des besoins. • Il permet de créer une application **autonome** avec moins de configuration • Spring Boot propose **des serveurs embarqués** tels que **Jetty** et **Tomcat** , etc. • Spring Boot est livré avec le concept de **démarreur** dans le fichier pom.xml qui prend en charge en interne le téléchargement des **JAR de** dépendances en fonction des exigences de Spring Boot.

Application

On souhaite créer une application qui permet de gérer un catalogue de produits. Ce catalogue est formé par des produits classés par catégorie qui sont stockés dans une base de données MYSQL.

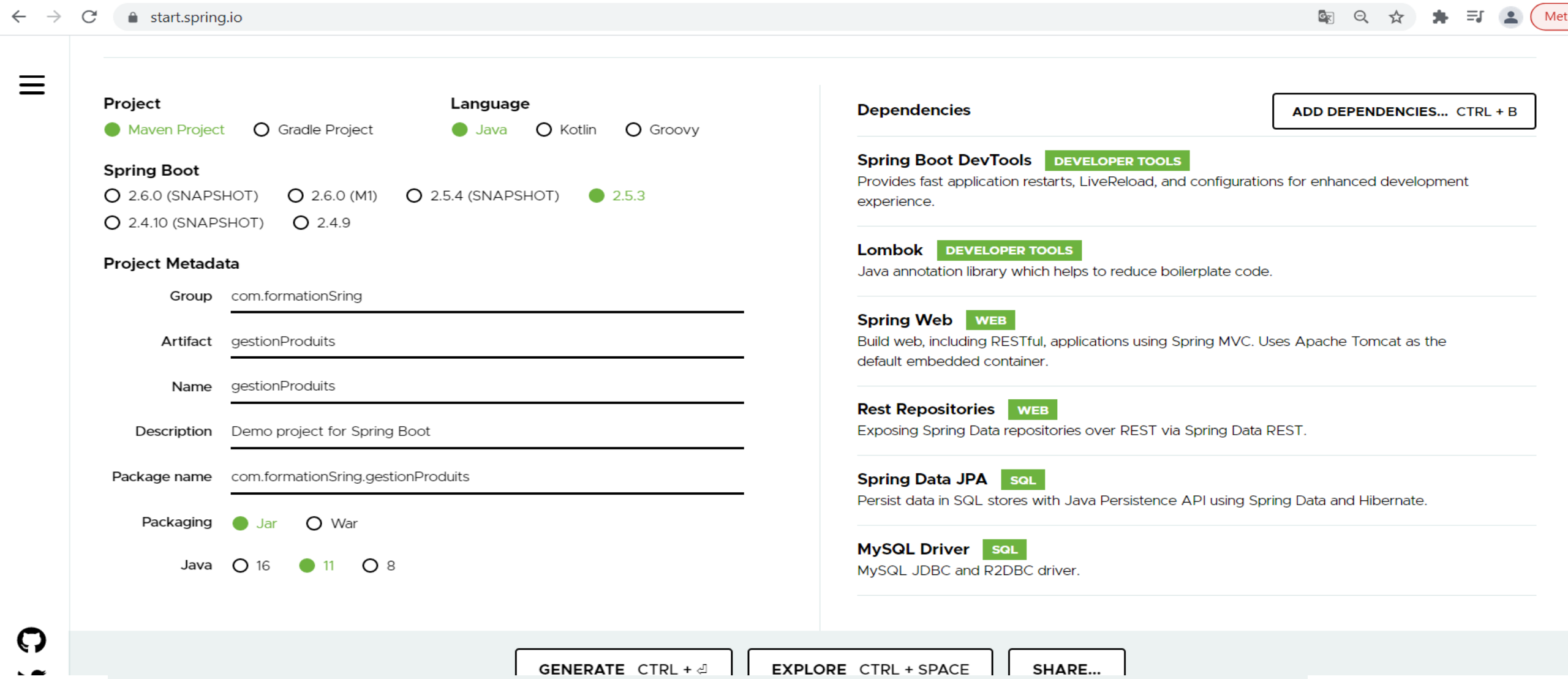
Une catégorie est définie par :

- Un code de type INT, clé primaire, Auto_Increment
- Le nom de catégorie de type VARCHAR.

Un produit appartenant à une catégorie est défini par :

- Une référence de type int, clé primaire
- Le nom du produit
- Le prix du produit
- La quantité en stock
- La photo du produit

Création d'un projet Spring Boot (https://start.spring.io/)



Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.
0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.5.2</version>

<relativePath/> <!-- lookup parent from repository -
->

</parent>

<groupId>com.formationEte</groupId>

<artifactId>gestionProduit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>gestionProduit</name>

<description>Demo project for Spring
Boot</description>

<properties>

<java.version>11</java.version>

</properties>
```

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

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-
devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-
java</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>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId
>
<artifactId>spring-boot-maven-
plugin</artifactId>
<configuration>
<excludes>
<exclude>
<groupId>org.projectlombok</groupId>
<artifactId>lombok</artifactId>
</exclude>
</excludes>
</configuration>
</plugin>
</plugins>
</build>

</project>
```

applications.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/catalogue?serverTimezone=UTC
spring.datasource.username=root
sprin.datasource.password=
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
spring.jpa.show-sql=true
server.port=9595
```

Association OneToMany, ManyToOne, OneToOne : Exemple Rendez-Vous Médecins , Patients

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Patient {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
    @OneToMany(mappedBy = "patient")
    private Collection<RendezVous> rendezVous;
}
```

PATIENT
- ID
- EMAIL
- NOM

```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Medecin {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String specialite;
    private String email;
    @OneToMany(mappedBy = "medecin")
    private Collection<RendezVous> rendezVous;
}
```

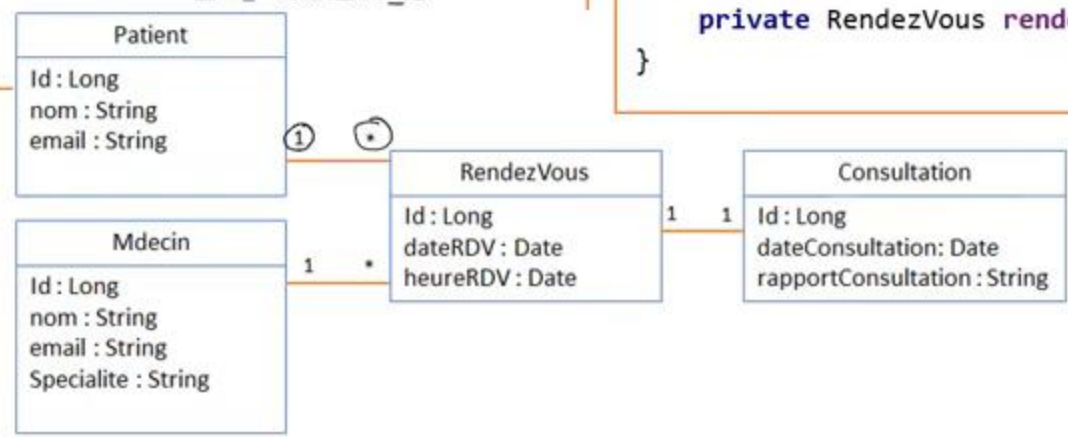
MEDECIN
- ID
- EMAIL
- NOM
- SPECIALITE

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class RendezVous {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateRendezVous;
    @ManyToOne
    private Medecin medecin;
    @ManyToOne
    private Patient patient;
    @OneToOne
    private Consultation consultation;
}
```

RENDEZ_VOUS
- ID
- DATE_RENDEZ_VOUS
- CONSULTATION_ID
- MEDECIN_ID
- PATIENT_ID

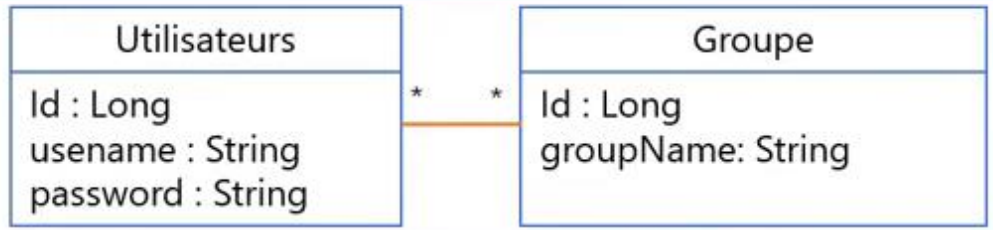
```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Consultation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateConsultation;
    private String rapportConsultation;
    private double prixConsultation;
    @OneToOne(mappedBy = "consultation")
    private RendezVous rendezVous;
}
```

CONSULTATION
- ID
- DATE_CONSULTATION
- PRIX_CONSULTATION
- RAPPORT_CONSULTATION



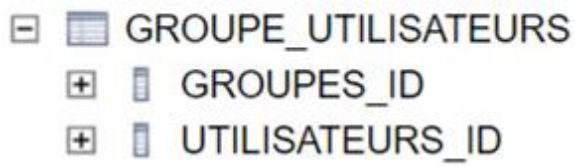
Cas de ManyMany

- On suppose que l'on souhaite de créer une application qui permet de gérer des Utilisateurs appartenant à des groupes. Chaque Groupe peut contenir plusieurs utilisateurs.




```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Utilisateur {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true)
    private String userName;
    private String password;
    @ManyToMany(mappedBy = "utilisateurs",fetch = FetchType.EAGER)
    private Collection<Groupe> groupes=new ArrayList<>();
}
```

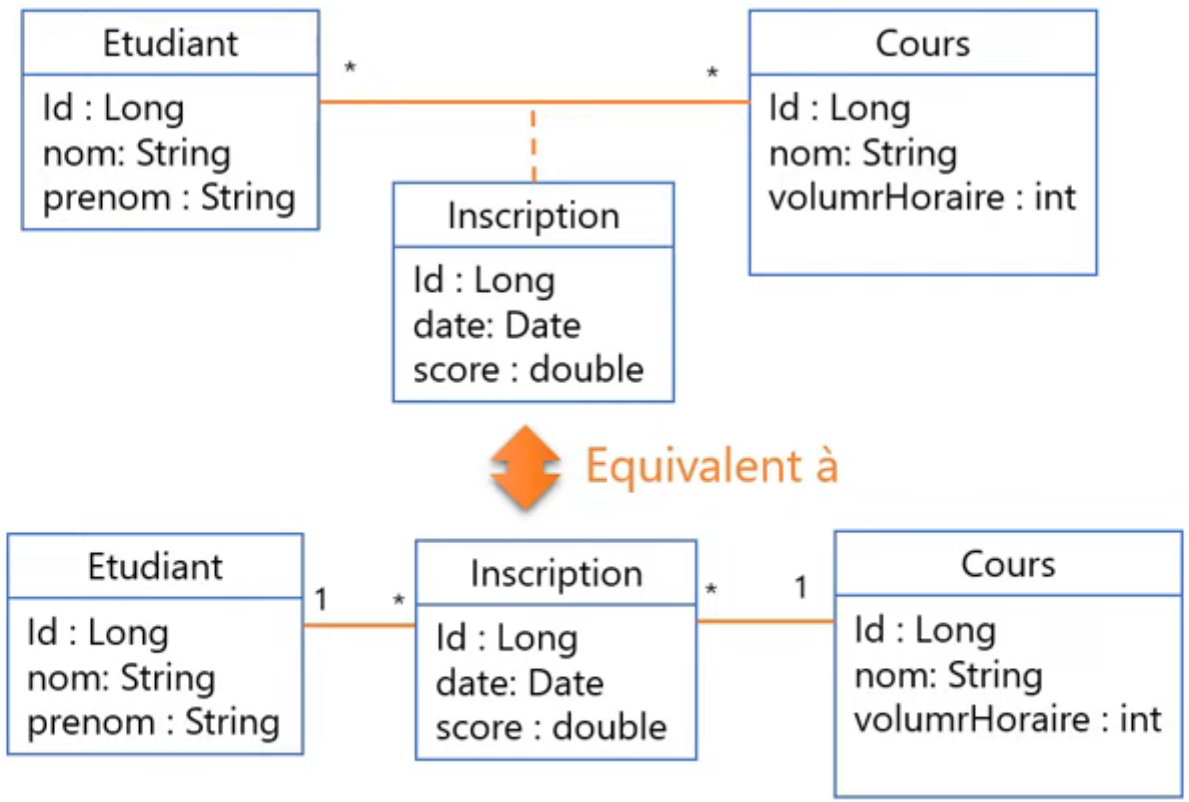
```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Groupe {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique = true)
    private String groupName;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<Utilisateur> utilisateurs=new ArrayList<>();
}
```



Cas de ManyMany avec Classe d'association

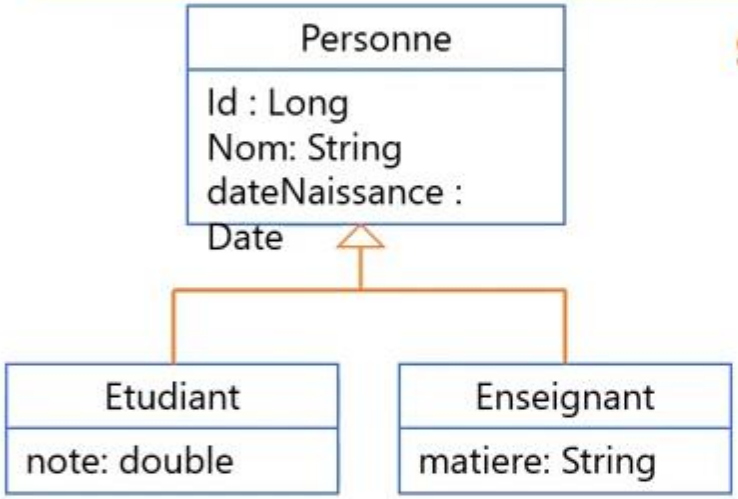
Exemple :

- Un Etudiant peut s'inscrire dans plusieurs Cours à une date donnée avec un score obtenu. Un cours concerne plusieurs Inscriptions. (Plusieurs à Plusieurs avec Classe d'association Inscription)
- Equivalent à : Un Etudiant peut effectuer Plusieurs Inscription. Chaque Inscription Concerne un Cours
- Deux Associations : Un à Plusieurs + Plusieurs à 1



Mapping de l'héritage

- Différentes stratégies de Mapping de l'héritage
- Une table par hiérarchie (SINGLE_TABLE)
- Une table pour chaque classe concrète (TABLE_PER_CLASS)
- Une table pour la classe parente et une table pour chaque classe fille (JOINED_TABLE)



SINGLE_TABLE : PERSONNES

ID	TYPE	NOM	DATE_NAISSANCE	NOTE	MAIETRE
1	ET	N1	d1	15	NULL
2	PROF	N2	d2	NULL	MATH

TABLE_PER_CLASS

ETUDIANTS

ID	NOM	DATE_NAISSANCE	NOTE
1	N1	d1	15

ENSEIGNANTS

ID	NOM	DATE_NAISSANCE	MATIERE
2	N2	d2	MATH

JOINED_TABLE

ETUDIANTS

NOTE	#ID
15	1

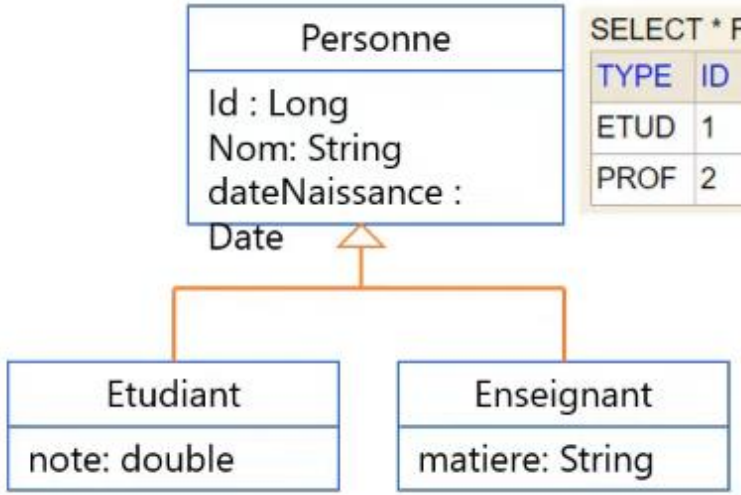
ENSEIGNANTS

MATIERE	#ID
MATH	2

PERSONNES

ID	NOM	DATE_NAISSANCE
1	N1	d1
2	N2	d2

Mapping de l'héritage : Stratégie SINGLE_TABLE



SELECT * FROM PERSONNE;

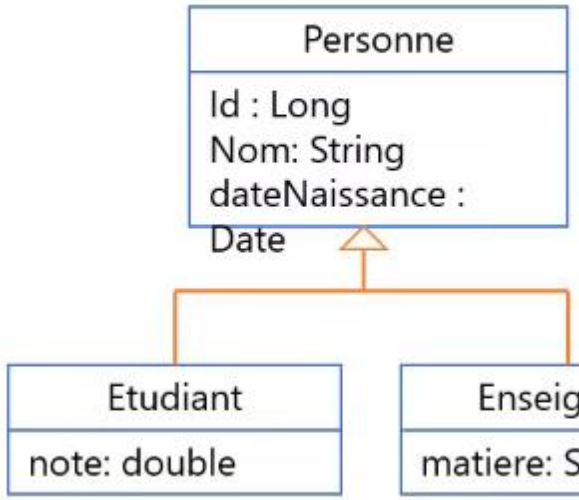
TYPE	ID	DATE_NAISSANCE	NOM	MATIERE	NOTE
ETUD	1	2020-09-03 20:05:28.358	Hassan	*null*	14.0
PROF	2	2020-09-03 20:05:28.362	Mohamed	MATH	*null*

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE",length = 4)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@DiscriminatorValue("ETUD")
public class Etudiant extends Personne {
    private double note;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@DiscriminatorValue("PROF")
public class Enseignant extends Personne {
    private String matiere;
}
```

Mapping de l'héritage : Stratégie TABLE_PER_CLASS



SELECT * FROM ETUDIANT;

ID	DATE_NAISSANCE	NOM	NOTE
1	2020-09-03 20:17:39.486	Hassan	14.0

SELECT * FROM ENSEIGNANT;

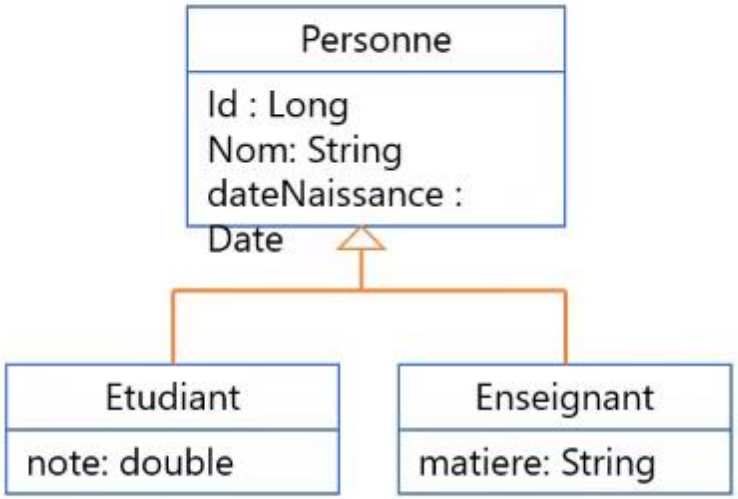
ID	DATE_NAISSANCE	NOM	MATIERE
2	2020-09-03 20:17:39.497	Mohamed	MATH

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Etudiant extends Personne {
    private double note;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Enseignant extends Personne {
    private String matiere;
}
```

Mapping de l'héritage : Stratégie JOINED_TABLE



SELECT * FROM PERSONNE;

ID	DATE_NAISSANCE	NOM
1	2020-09-03 20:25:17.668	Hassan
2	2020-09-03 20:25:17.679	Mohamed

SELECT * FROM ETUDIANT;

NOTE	ID
14.0	1

SELECT * FROM ENSEIGNANT;

MATIERE	ID
MATH	2

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Personne {
    @Id @GeneratedValue(strategy = GenerationType.TABLE)
    private Long id;
    private String nom;
    private Date dateNaissance;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Etudiant extends Personne {
    private double note;
}
```

```java
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Enseignant extends Personne {
    private String matiere;
}
```

Entité Categorie

```
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Categorie {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String nom;
@OneToMany(mappedBy = "catalogue", fetch = FetchType.EAGER)// cascade = CascadeType.ALL
@JsonProperty(access = Access.WRITE_ONLY)
Collection<Produit> produits;
public Catalogue(String nom) {
super();
this.nom = nom;
}
@Override
public String toString() {
return « Categorie [id=" + id + ", nom=" + nom + ", produits=" + produits + "]";
}
}
```

Entité Produit

```
@Entity
@Data
@AllArgsConstructor @NoArgsConstructor @Table(name="produit")
public class Produit {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
@Column(name="nom_product")
private String nom;
private int quantite;
private double prix;
private String photo;
@ManyToOne
private Catalogue catalogue;
@Override
public String toString() {
return "Produit [id=" + id + ", nom=" + nom + ", quantite=" + quantite + ", prix=" + prix + ", photo=" + photo
+ "  catalogue: "+catalogue.getNom()+ "]";
}
}
```

Categorie Repository

```
@CrossOrigin("*")
@RepositoryRestResource
public interface CategorieRepository extends JpaRepository<Categorie, Long>  {
    @RestResource(path = "/catname")
    public List<Categorie> findByNomContains(@Param("cat") String nomcat);

    @RestResource(path="/productsByCat")
    @Query("select p from Produit p where p.categorie.id=:x")

    public List<Produit> getProducts(@Param("x")Long id);
}
```

Produit Repository

```
@CrossOrigin("*")
@RepositoryRestResource

public interface ProduitRepository extends JpaRepository<Produit, Long> {
@RestResource(path="/CatalogueName")
@Query(value="select p from Produit p where p.catalogue.nom like %:mc%")
public List<Produit> findByNameCategorie(@Param("mc") String mc);
@RestResource(path="/products")
@Query("select p from Produit p where p.catalogue.id=:x" )
public List<Produit> listeProduits(@Param("x")Long id);

@RestResource(path = "/rechercheNomPrix")
public List<Produit> findByNomContainsAndPrix(@Param("mc")String mc,@Param("prix")double
prix);
}
```

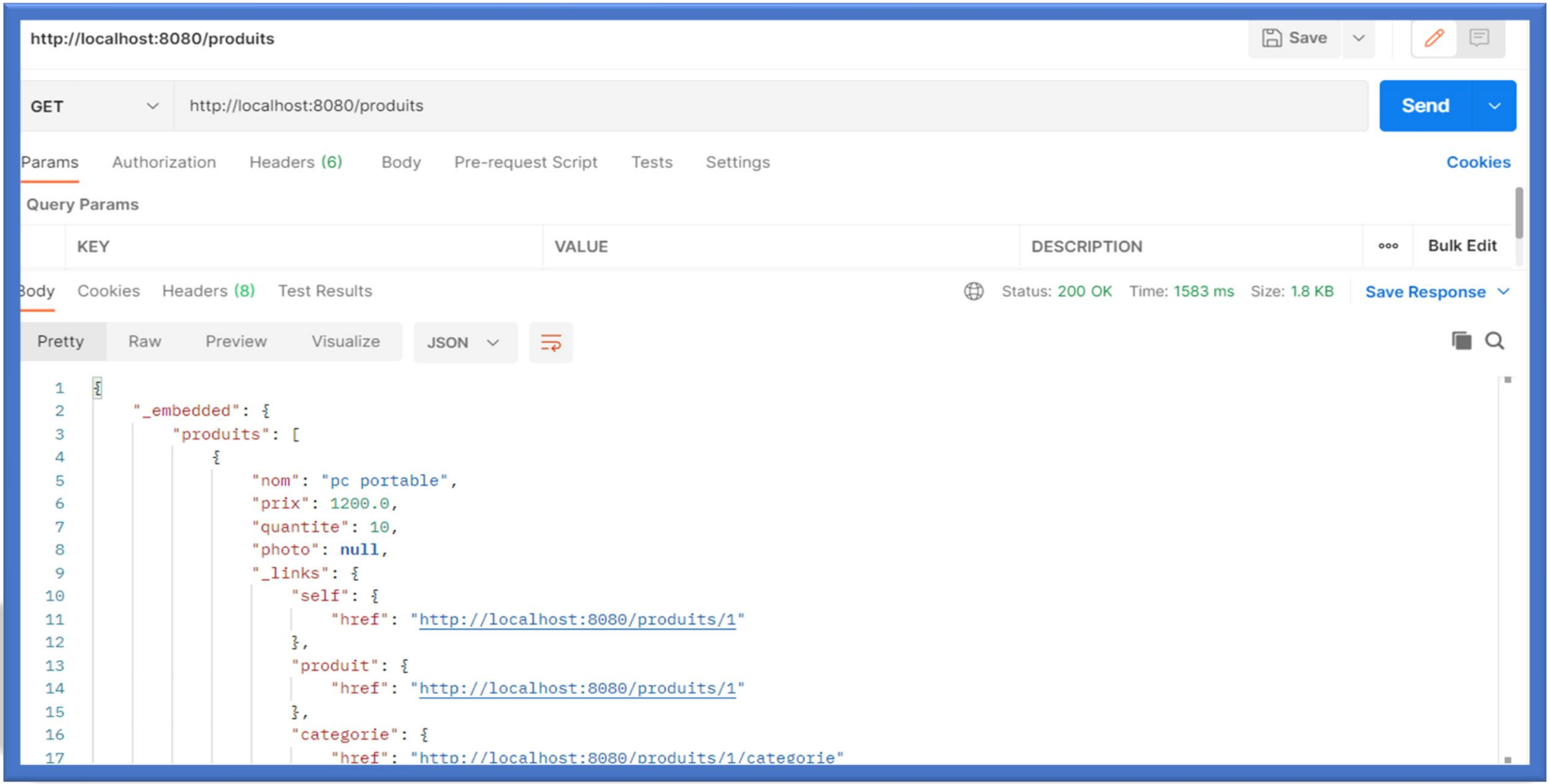
GestionProduitApplication

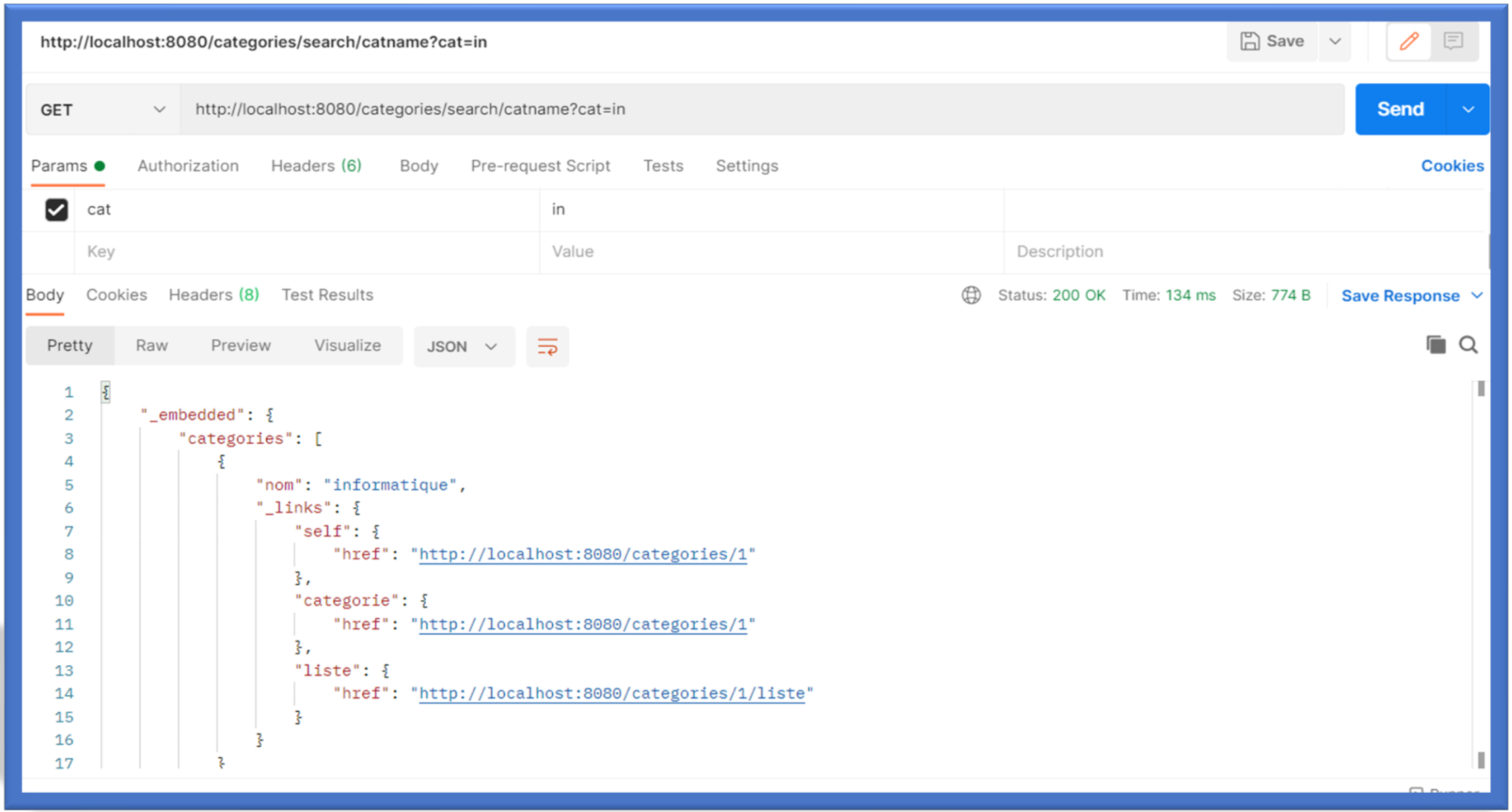
```java
@SpringBootApplication
public class GestionProduitApplication implements CommandLineRunner {


public static void main(String[] args) {
SpringApplication.run(GestionProduitApplication.class, args);
}
@Autowired
RepositoryRestConfiguration rc;
@Autowired
ProduitRepository pr;
@Autowired
CategorieRepository cr;
@Override
public void run(String... args) throws Exception {
rc.exposeIdsFor(Produit.class);
rc.exposeIdsFor(Categorie.class);
Categorie c1=new Categorie(null, "informatique", null);
Categorie c2=new Categorie(null, "electronique", null);
Categorie c3=new Categorie(null, "telephonie", null);
```

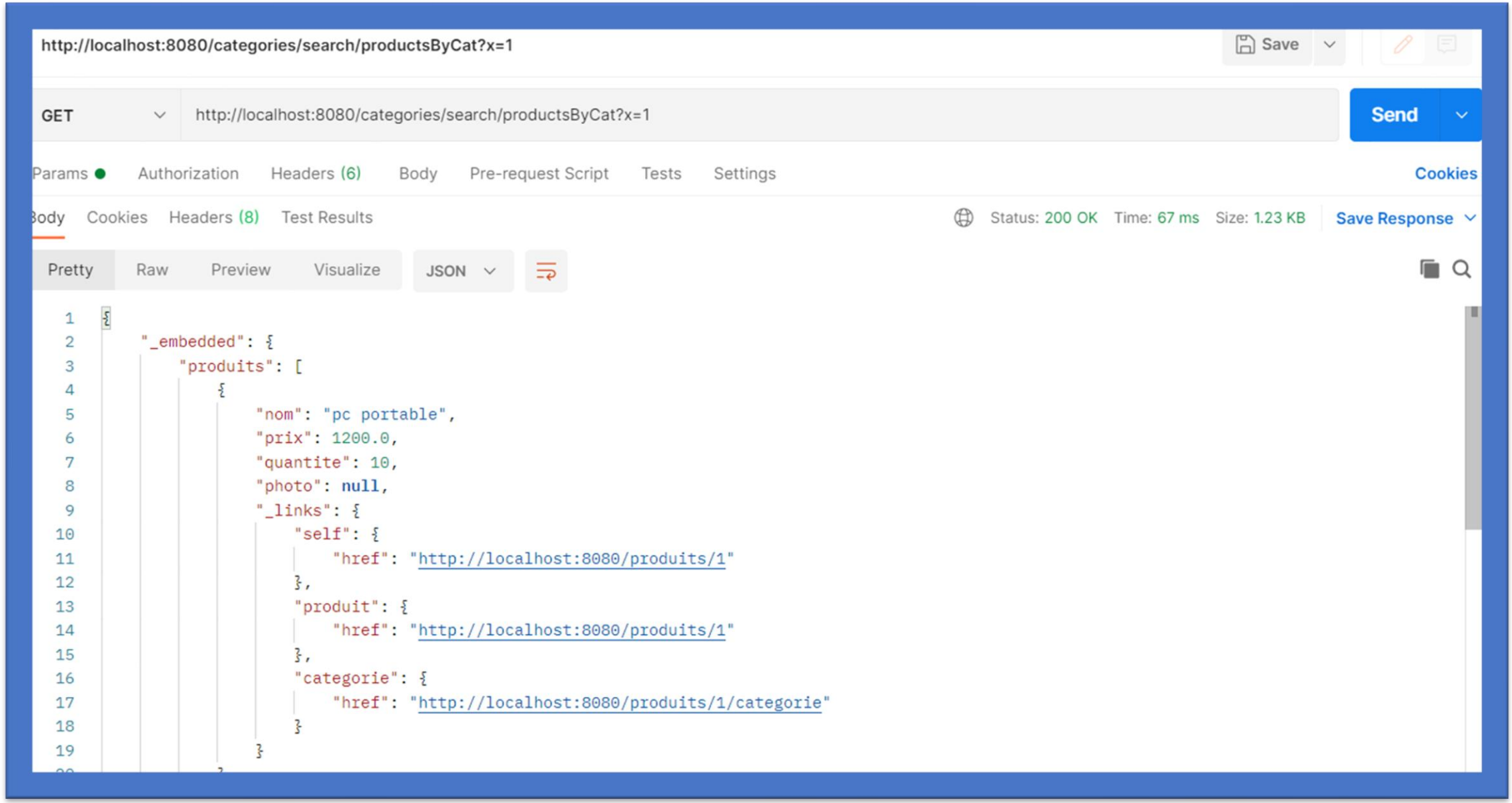
```java
cr.save(c1);
cr.save(c2);
cr.save(c3);
pr.save(new Produit(null, "pc portable", 1200, 10, null, c1));
pr.save(new Produit(null, "clavier", 12, 10, null, c1));
pr.save(new Produit(null, "video projecteur", 1200, 10, null, c2));

Categorie cc=cr.findById(1L).get();
cc.getListe().forEach(x->System.out.println(x.getNom()));
}

}
```

```
.   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.2)

2021-08-05 15:57:08.068  INFO 44556 --- [  restartedMain] c.f.g.GestionProduitApplication          : Starting GestionProduitApplication using Java 15.0.2 on
2021-08-05 15:57:08.072  INFO 44556 --- [  restartedMain] c.f.g.GestionProduitApplication          : No active profile set, falling back to default profiles
2021-08-05 15:57:08.204  INFO 44556 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools
2021-08-05 15:57:08.206  INFO 44556 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the
2021-08-05 15:57:09.879  INFO 44556 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT m
2021-08-05 15:57:10.047  INFO 44556 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 145 ms. Fou
2021-08-05 15:57:12.377  INFO 44556 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-08-05 15:57:12.400  INFO 44556 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-08-05 15:57:12.400  INFO 44556 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.48]
2021-08-05 15:57:12.659  INFO 44556 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-08-05 15:57:12.660  INFO 44556 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in
2021-08-05 15:57:13.499  INFO 44556 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: defaul
2021-08-05 15:57:13.642  INFO 44556 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.32.Final
2021-08-05 15:57:14.003  INFO 44556 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final
2021-08-05 15:57:14.267  INFO 44556 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2021-08-05 15:57:14.688  INFO 44556 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2021-08-05 15:57:14.751  INFO 44556 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL55
2021-08-05 15:57:15.931  INFO 44556 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hiber
2021-08-05 15:57:15.942  INFO 44556 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence un
2021-08-05 15:57:17.540  WARN 44556 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefor
2021-08-05 15:57:20.613  WARN 44556 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : Unable to start LiveReload server
2021-08-05 15:57:20.762  INFO 44556 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context pat
2021-08-05 15:57:20.785  INFO 44556 --- [  restartedMain] c.f.g.GestionProduitApplication          : Started GestionProduitApplication in 13.467 seconds (JV
c portable
lavier
```

Tester les APIs avec Postman




http://localhost:8080/categories/search/catname?cat=in
Save
GET
http://localhost:8080/categories/search/catname?cat=in
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
cat
in
Key
Value
Description
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 134 ms
Size: 774 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_embedded": {
"categories": [
{
"nom": "informatique",
"_links": {
"self": {
"href": "http://localhost:8080/categories/1"
},
"categorie": {
"href": "http://localhost:8080/categories/1"
},
"liste": {
"href": "http://localhost:8080/categories/1/liste"
}
}
}


http://localhost:8080/categories/search/productsByCat?x=1
Save
GET
http://localhost:8080/categories/search/productsByCat?x=1
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 67 ms
Size: 1.23 KB
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"_embedded": {
"produits": [
{
"nom": "pc portable",
"prix": 1200.0,
"quantite": 10,
"photo": null,
"_links": {
"self": {
"href": "http://localhost:8080/produits/1"
},
"produit": {
"href": "http://localhost:8080/produits/1"
},
"categorie": {
"href": "http://localhost:8080/produits/1/categorie"
}
}

Service (Interface)

```
public interface IAppService {
public void ajouterCategorie(Categorie c);
public void ajouterProduit(String produit, MultipartFile file)throws Exception;
public List<Categorie> getAllCategories();
public List<Produit> getALlProduits();
public List <Produit> getProduitByCat(Long idCat);
public void supprimerCategorie(Long idCat);
public Produit getProduitById(Long idProduit);
public void supprimerProduit(Long idProduit);
public byte[] getImage(Long IdProduit)throws IOException;
public Categorie getCategorieById(Long idCat);
public void updateProduit1(String produit,Long idProduit,MultipartFile file) throws Exception;
public void updateProduit2(Produit produit,Long idProduit) ;


}
```

Service (implementation)

```
@Transactional @Service
public class AppService implements IAppService {
ProduitRepository pr;
public AppService(ProduitRepository pr) {
this.pr = pr;}
@Autowired
CategorieRepository cr;
@Override
public void ajouterCategorie(Categorie c) {
cr.save(c);}
private String saveImage(MultipartFile file) throws
IOException {
String filename=file.getOriginalFilename();
String tab[]=filename.split("\\.");
String
filenameModif=tab[0]+"_"+System.currentTimeMillis()+"."+ta
b[1];
File f=new
File(System.getProperty("user.home")+"/images/"+filenameMo
dif);
FileOutputStream fos=new FileOutputStream(f);
fos.write(file.getBytes());
//FileUtils.writeByteArrayToFile(f, file.getBytes());
return filenameModif;}
```

```
@Override
public void ajouterProduit(String produit,
MultipartFile file) throws Exception {
Produit p=new ObjectMapper().readValue(produit,
Produit.class);
String nomImage=saveImage(file);
p.setPhoto(nomImage);
pr.save(p);}

@Override
public List<Categorie> getAllCategories() {
return cr.findAll();
}

@Override
public List<Produit> getALlProduits() {
return pr.findAll();
}
```

Service (implementation)

```
@Override
public List<Produit> getProduitByCat(Long idCat)
{
Categorie cc=cr.findById(idCat).get();
return cc.getListe();
}

@Override
public void supprimerCategorie(Long idCat) {
Categorie cc=cr.findById(idCat).get();
     List<Produit>liste=cc.getListe();
      for(Produit p:liste)
     p.setCategorie(null);

    cr.delete(cc);}
@Override
public Produit getProduitById(Long idProduit) {
// TODO Auto-generated method stub
return pr.findById(idProduit).get();
}
```

```
@Override
public void supprimerProduit(Long idProduit) {
pr.deleteById(idProduit);}
@Override
public byte[] getImage(Long IdProduit) throws IOException {
String photo=pr.getById(IdProduit).getPhoto();
File f=new File(System.getProperty("user.home"));
Path p=Paths.get(f+"/images/"+photo);
return Files.readAllBytes(p);
}
@Override
public Categorie getCategorieById(Long idCat) {
// TODO Auto-generated method stub
return cr.findById(idCat).get();
}
```

Service (implementation)

```
• @Override
public void updateProduit1(String produit, Long idProduit,
MultipartFile file) throws Exception {

String fileName=saveImage(file);
Produit p=new ObjectMapper().readValue(produit,Produit.class);
p.setId(idProduit);
p.setPhoto(fileName);
pr.save(p);
}

@Override
public void updateProduit2(Produit produit, Long idProduit) {

produit.setId(idProduit);
pr.save(produit);

}
```

Controleur (CategorieControler)

```java
@CrossOrigin("*")
@RestController
@RequestMapping("/api")
public class CategorieControleur {
@Autowired
IAppService service;

@GetMapping("/categories")
public List<Categorie> getAllCategories()
{
return service.getAllCategories();

}
@GetMapping("/categorie/{id}")
public Categorie getCategorie(@PathVariable Long id)
{
return service.getCategorieById(id);

}
```

```java
@GetMapping("/productsByCat/{idcat}")
List<Produit> getProduitsByCat(@PathVariable Long idcat)
{
return service.getProduitByCat(idcat);
}
@PostMapping("/listecategories")
public void addCat(@RequestBody Categorie c)
{
service.ajouterCategorie(c);
}

@DeleteMapping("/listecategories/{idc}")
public void supprimerCategorie(@PathVariable Long idc)
{    service.supprimerCategorie(idc);
}

}
```

Controleur(ProduitControler)

```
@CrossOrigin("*")
@RestController
@RequestMapping("/api")
public class ProduitControleur {
@Autowired
IAppService service;
@Autowired
ProduitRepository pr;
@Autowired
CategorieRepository cr;
@GetMapping("products")
   public List<Produit>getAllProduct()
   {   return service.getALlProduits();   }
@GetMapping("/product/{id}")
public Produit getProduit(@PathVariable Long id)
{return service.getProduitById(id);}
```

```
@PostMapping("/addProduct")
public void addProduit(@RequestParam("produit")
String prod, @RequestParam("file") MultipartFile
file)throws Exception {
service.ajouterProduit(prod, file);


}
@DeleteMapping("/deleteProduct/{id}")
public void delete(@PathVariable("id") Long id)
{service.supprimerCategorie(id);}
```

Controleur(ProduitControler)

```
public byte[] getImage(@PathVariable Long id)throws Exception
{
  return service.getImage(id);
}


@PutMapping("/update2/{id}")
public void modifier(@RequestBody Produit p, @PathVariable("id")
Long id) throws Exception {
  service.updateProduit2(p,id);
}


@PutMapping("/update/{id}")
public void modifier(@RequestParam("produit") String prod,
@RequestParam("file") MultipartFile file,
@PathVariable("id") Long id) throws Exception {
  service.updateProduit1(prod, id, file);


}
```