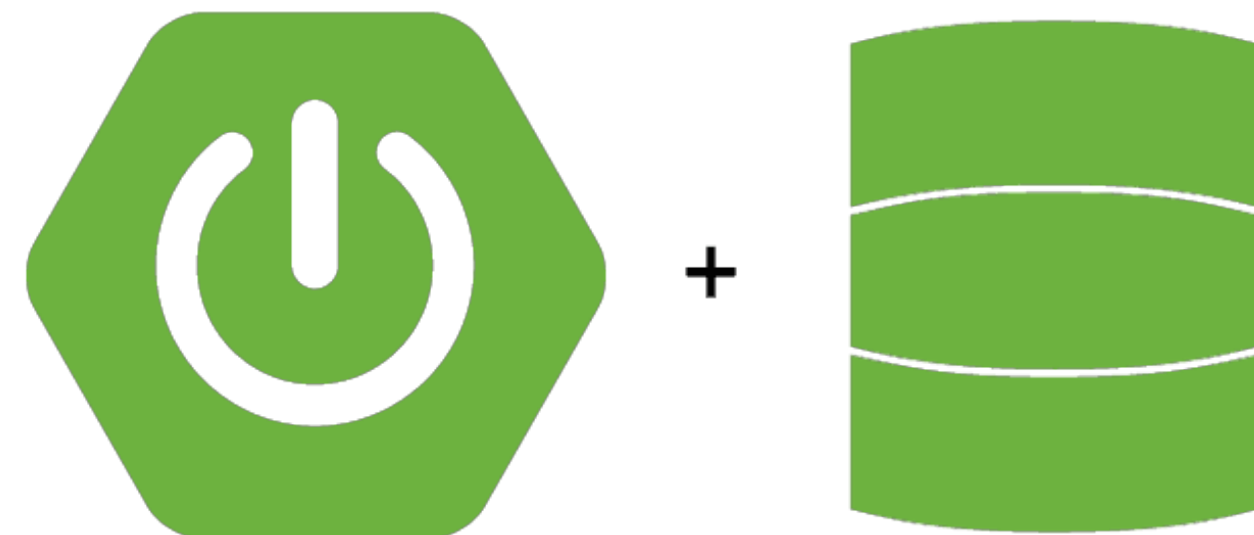




SPRING DATA JPA



01 - Définition

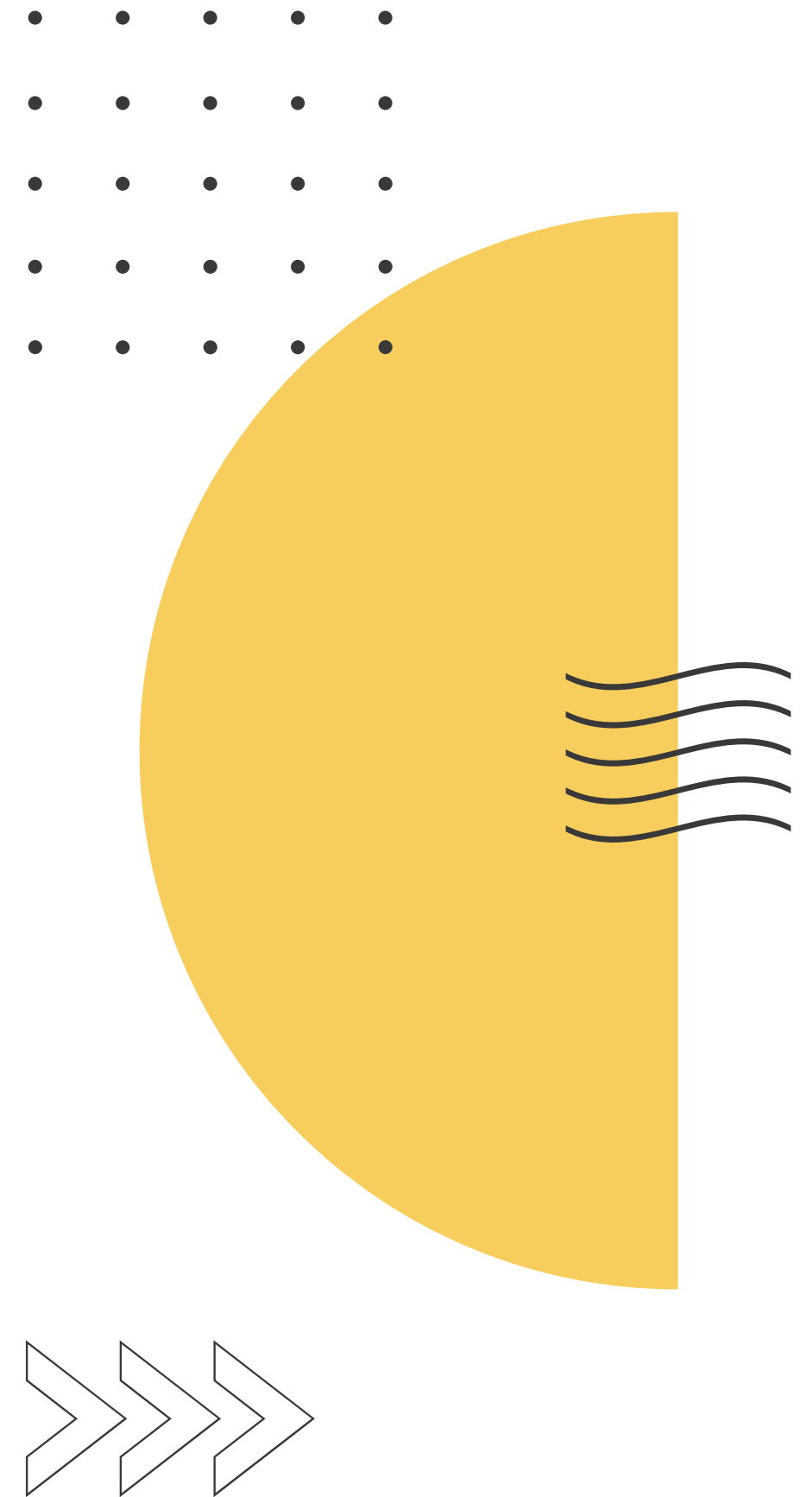
Un framework est, un “cadre de travail”. L’objectif est généralement de simplifier le travail des développeurs informatiques , en leur offrant une architecture “prête à l’emploi” et qui leur permette de ne pas repartir de zéro à chaque nouveau projet.

Un ORM (Object-Relational Mapping), sont des frameworks qui permettent de créer une correspondance entre un modèle objet et un modèle relationnel de base de données.

En Java, les frameworks ORM se basent sur une spécification nommée JPA (Java Persistence API). Cette spécification est une API qui décrit les comportements nécessaires au bon fonctionnement d’un framework ORM.

Grâce à ces spécifications, de nombreuses implémentations ont vu le jour, comme :

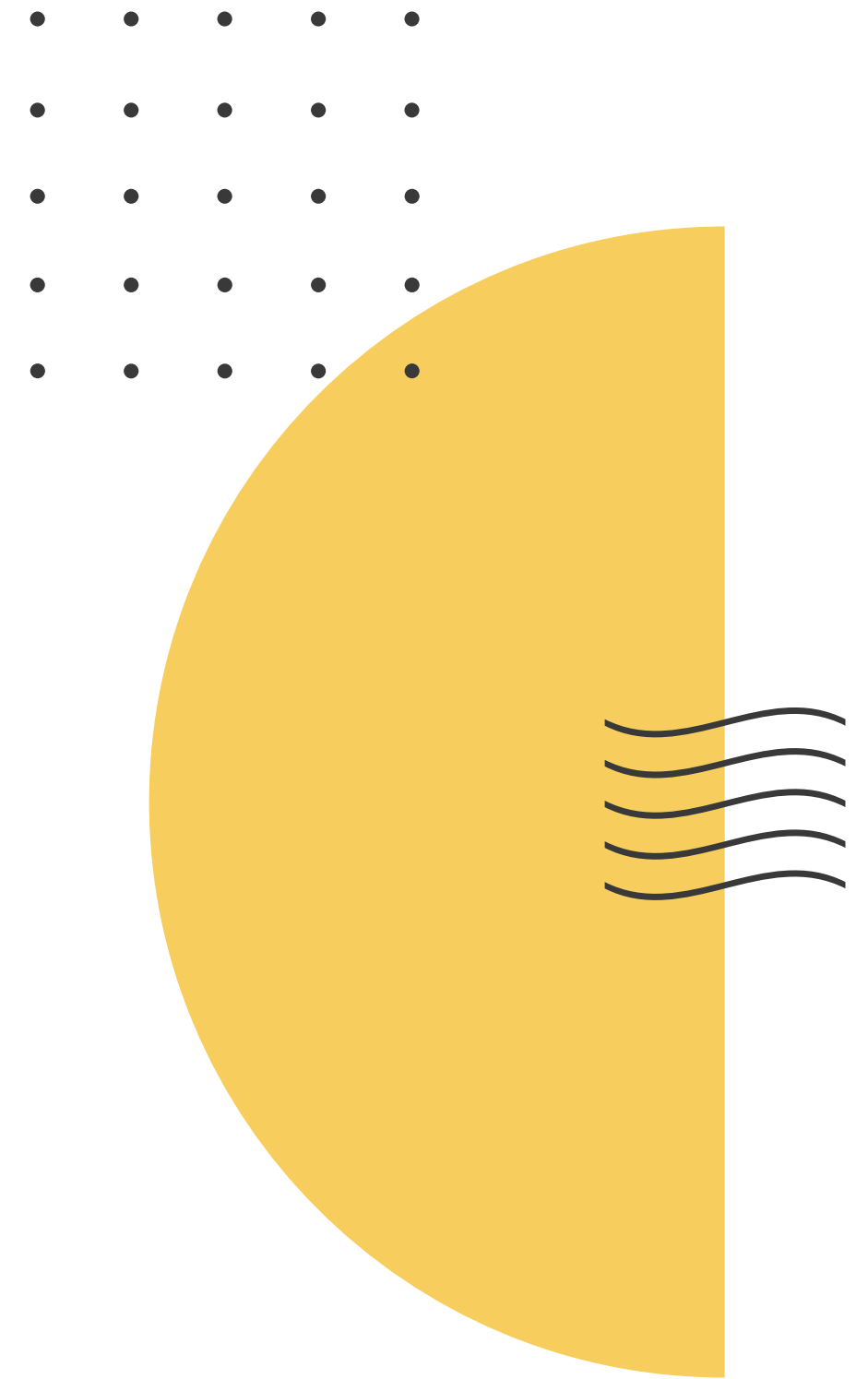
- EclipseLink ;
- Spring Data JPA ;
- OpenJPA ;
- Hibernate.



02 - Pourquoi Spring Data JPA ?

- Sa simplicité : Assez simple d'utilisation
- C'est un composant Spring, et le framework Spring est un excellent framework pour implémenter des applications Java robustes et performantes.
- Il n'y a qu'un pas après avoir appris Spring Data JPA pour apprendre Hibernate, un autre framework ORM très utilisé sur le marché.

Spring Data JPA est un framework ORM performant qui vous permet d'interagir avec une base de données.



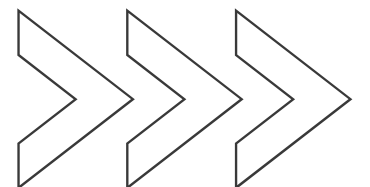
03 - Configuration

Nous allons créer la structure de notre projet Spring Boot sur <https://start.spring.io/>, que nous importerons ensuite dans notre IDE.

The screenshot shows the Spring Initializr web application interface. The browser address bar displays 'start.spring.io'. The page features a sidebar with a hamburger menu and a 'spring initializr' logo. The main content area is divided into sections for project configuration:

- Project:** Includes radio buttons for 'Maven Project' (selected), 'Gradle Project', and 'Language' options: 'Java' (selected), 'Kotlin', and 'Groovy'.
- Spring Boot:** Includes radio buttons for versions: '2.5.1 (SNAPSHOT)', '2.5.0' (selected), '2.4.7 (SNAPSHOT)', '2.4.6', '2.3.12 (SNAPSHOT)', and '2.3.11'.
- Project Metadata:** Includes input fields for 'Group' (com.jpa), 'Artifact' (spring), 'Name' (spring), 'Description' (Demo project for Spring Boot), and 'Package name' (com.jpa.spring). It also has a 'Packaging' section with 'Jar' (selected) and 'War' options, and a 'Java' version section with '16', '11', and '8' (selected).
- Dependencies:** Includes a button 'ADD DEPENDENCIES... CTRL + B' and two dependency cards: 'Spring Data JPA' (SQL) and 'MySQL Driver' (SQL).

At the bottom of the page, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. A social media icon (Twitter) is visible in the bottom left corner.

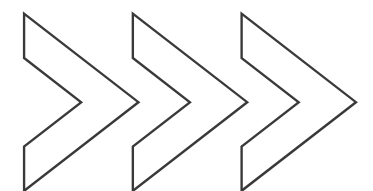
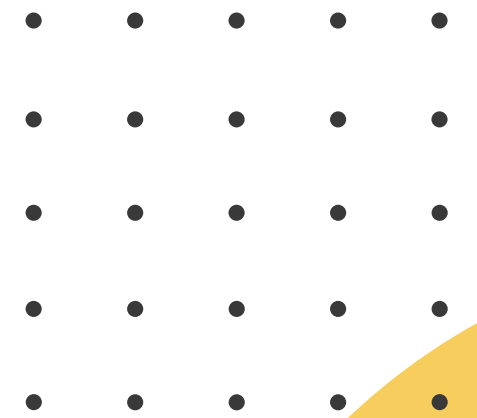


Lors de la génération du projet, le répertoire `/src/main/resources` contient un fichier par défaut nommé `applications.properties`

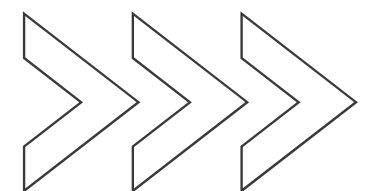
Ce fichier est considéré par Spring comme étant une source de propriétés. Au démarrage de l'application, ce fichier sera lu, et les informations présentes pourront être exploitées par le code de l'application.

Voici le code nécessaire à la connexion de la base de données. Il s'agit d'ouvrir une connexion JDBC (Java DataBase Connectivity) vers notre base de données.

```
</> application.properties ×  
1  
2  spring.datasource.url=jdbc:mysql://localhost:3306/assuranceAuto?serverTimezone=UTC  
3  spring.datasource.username=globlehi  
4  spring.datasource.password=globlehi  
5  spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect  
6
```

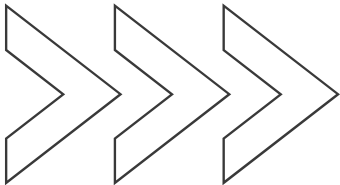
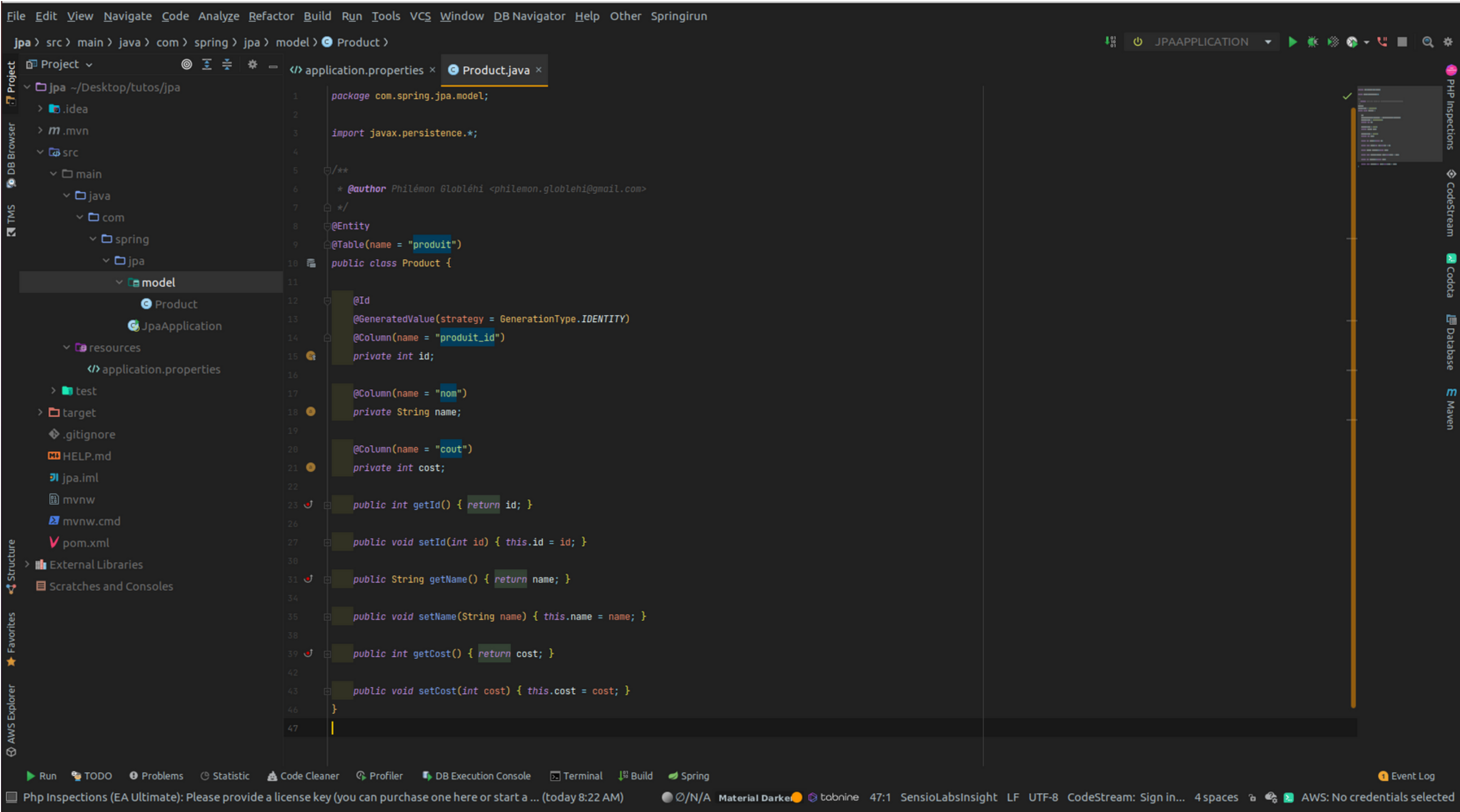


- `spring.datasource.url` : indique l'URL de connexion à la base de données. Cette URL possède plusieurs informations clés :
 - `jdbc:mysql` : le protocole utilisé.
 - `localhost` : la machine où la base de données est installée.
 - `3306` : le port de la base de données sur votre poste.
 - `/assuranceAuto` : le nom de la base de données.
 - `?serverTimezone=UTC` : ce paramètre permet de résoudre une erreur courante avec le driver MySQL.
- `spring.datasource.username` : Nom de l'utilisateur pour se connecter à la base de données.
- `spring.datasource.password` : Mot de passe de l'utilisateur
- `spring.jpa.properties.hibernate.dialect` : Spécifie au framework ORM le dialecte SQL à utiliser, cela permet d'optimiser les traitements du framework.



04 - Entité

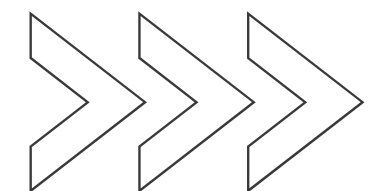
Nous allons implémenter la classe **Product** dans un package nommé **model**.



- `@Entity` est une annotation qui indique que la classe correspond à une table de la base de données.
- `@Table(name="produit")` est une annotation qui indique le nom de la table associée.
- `@Id` est une annotation qui indique la clé primaire de la table .
- id étant auto-incrémenté, on ajoute l'annotation `@GeneratedValue(strategy = GenerationType.IDENTITY)`.
- `@Column` est une annotation qui permet de faire le lien entre les attributs de la classe avec le nom du champ de la table dans la base de données.

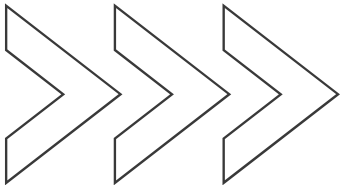
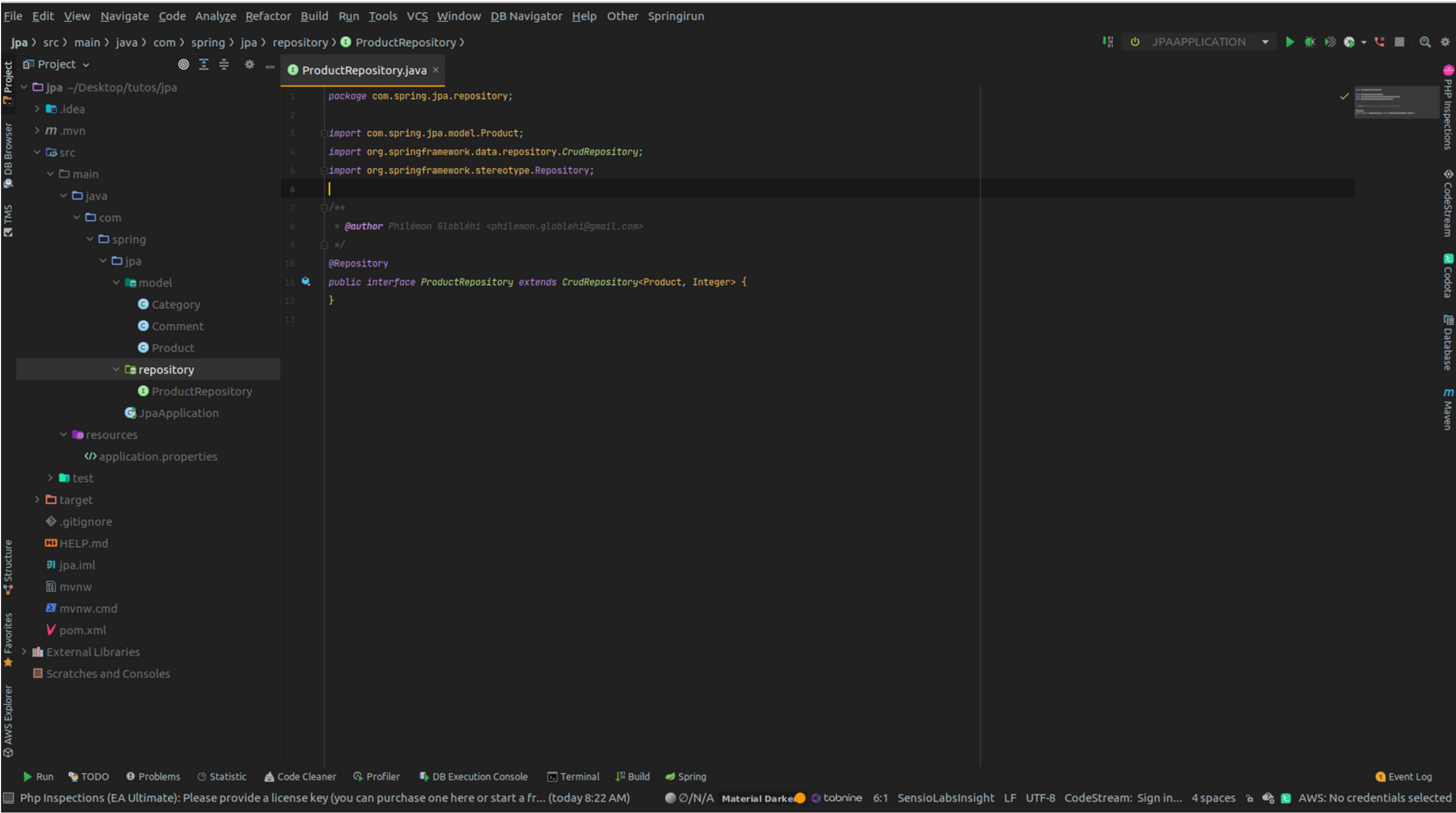


Il est important de noter que Spring Data JPA est capable de faire l'association automatique entre le nom de l'entité et le nom de la table, ainsi qu'entre le nom des attributs et le nom des colonnes. Donc, si les noms sont similaires, les annotations `@Table` et `@Column` deviennent facultatives.



05 - Repository

Nous allons implémenter l'interface **ProductRepository** dans un package nommé **repository**.

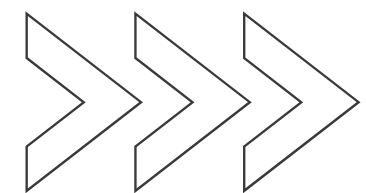
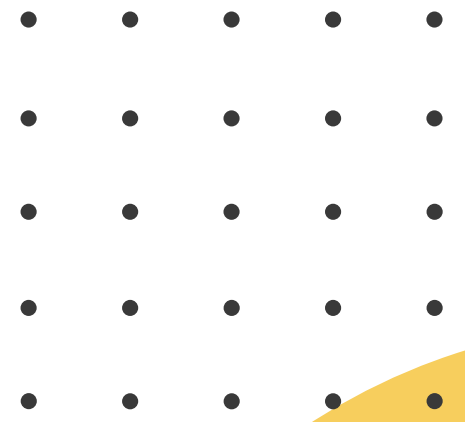


- `@Repository` est une annotation Spring pour indiquer que la classe a pour rôle de communiquer avec une source de données (en l'occurrence la base de données).

L'interface étend `CrudRepository`.

C'est une interface qui est fournie par Spring. Elle spécifie des méthodes pour manipuler notre entité. Elle utilise la généricité pour que son code soit applicable à n'importe quelle entité, d'où la syntaxe "`CrudRepository<Product, Integer>`".

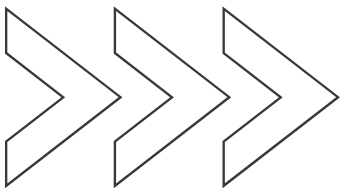
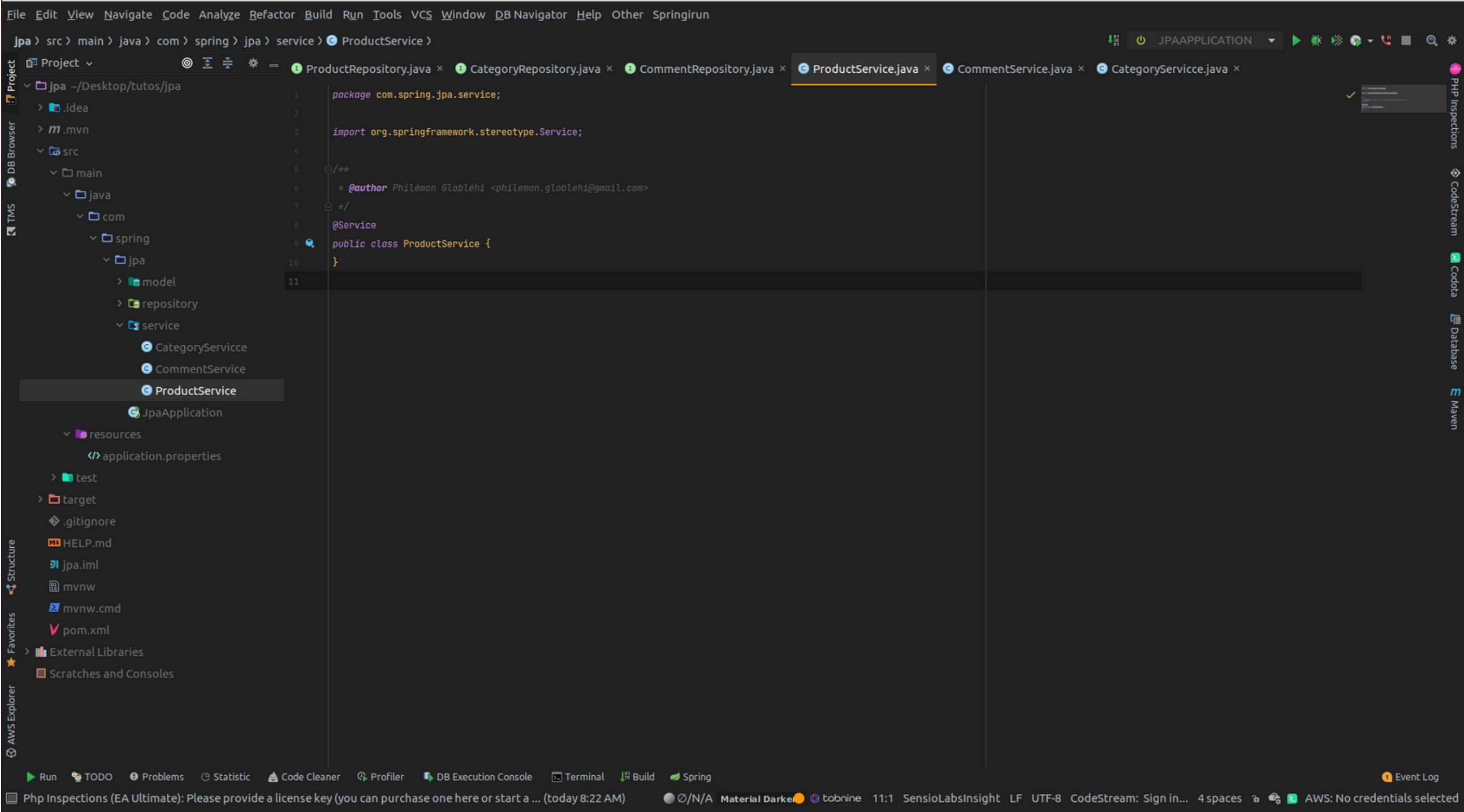
`@Repository` est une spécialisation de l'annotation `@Component`. Elle permet de déclarer auprès du Context d'Application Spring qu'une classe est un bean à exploiter. Par son nom, elle fournit une indication sémantique sur le rôle de la classe.



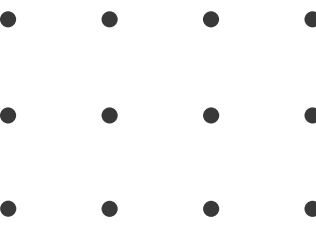
06 - Service

Pour suivre l'architecture en couche, la couche métier est mise en place. Elle permet d'appliquer des traitements spécifiques sur les données avant l'interaction avec la base de données.

Chaque classe aura l'annotation `@Service` (une autre spécialisation de `@Component`).



07 - Contrôleur

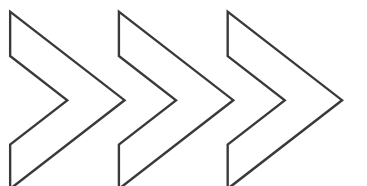


Pour exposer le contenu de nos différentes ressources, nous allons créer les contrôleurs dans le package **controller**. Déjà ajouter dans le pom.xml la dépendance **Spring Web** dans la section `<dependencies>...</dependencies>`

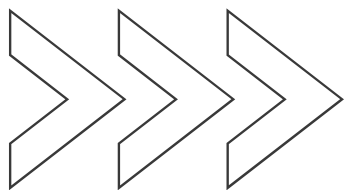
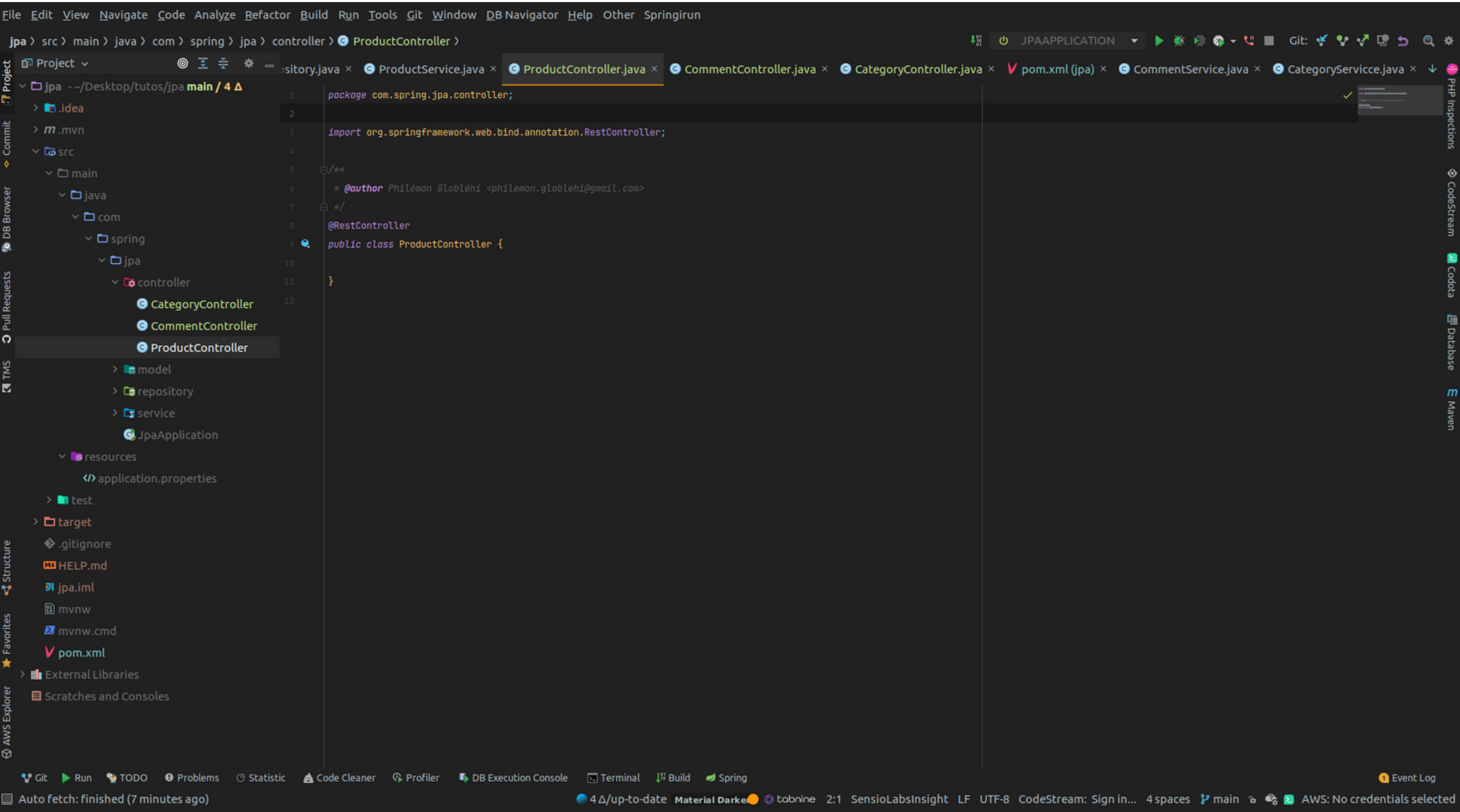
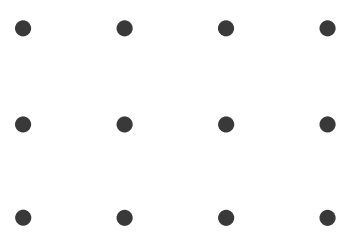
Ensuite lancer la build de l'application avec la commande **mvn install**



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

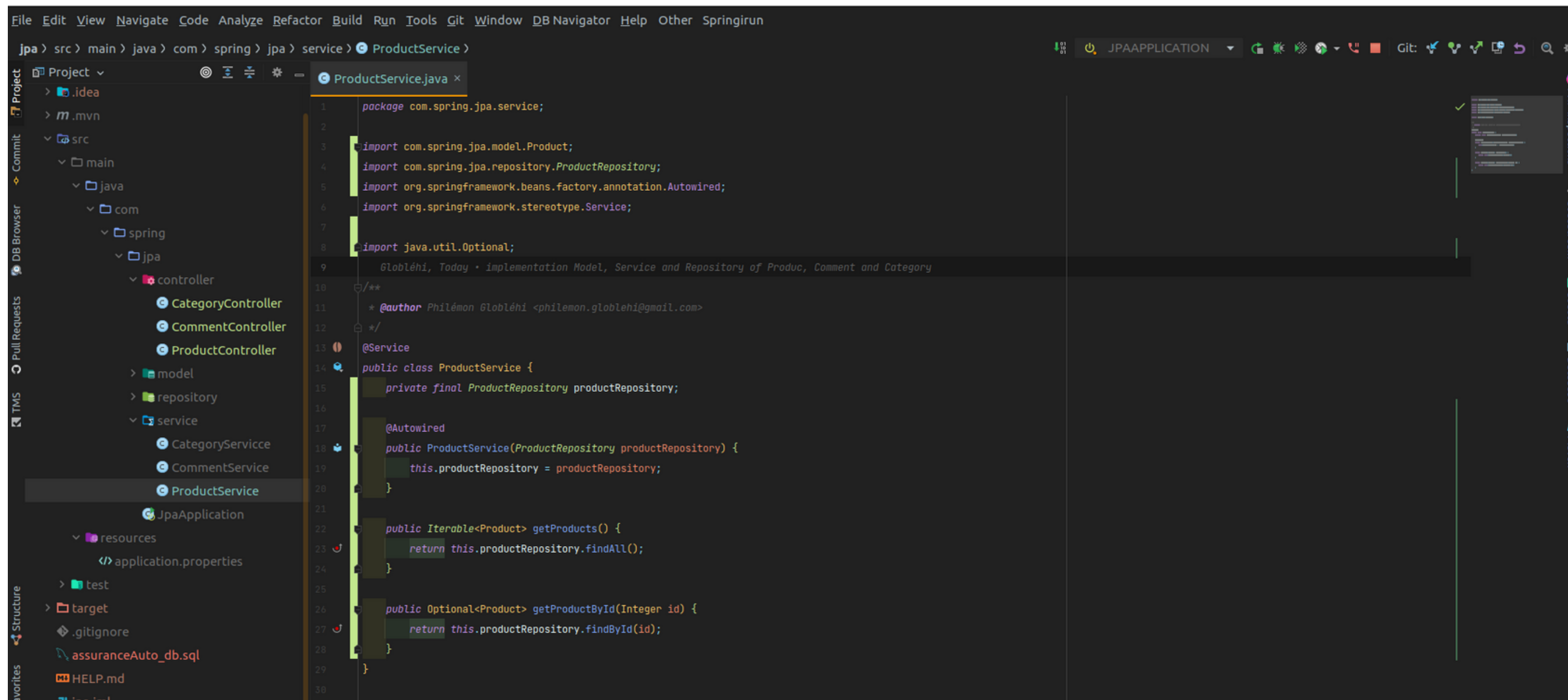


Chaque classe aura l'annotation **@RestController** (une autre spécialisation de @Component). Elle indique à Spring d'insérer le retour de la méthode au format JSON dans le corps de la réponse HTTP. Grâce à cela, les applications qui vont communiquer avec votre API accéderont au résultat de leur requête en parsant la réponse HTTP.



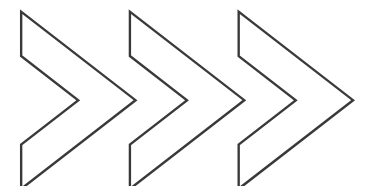
08 - Récupérer une ressource

- L'interface CrudRepository donne accès à de nombreuses méthodes pour interagir avec la base de données.
- Pour récupérer un ensemble de données, nous avons utilisé `findAll()` qui renvoie une liste d'objets correspondant à toutes les données de la table associée à l'entité concernée.
- Pour récupérer une donnée précise, nous avons utilisé `findById(Integer id)`, qui renvoie un unique objet correspondant à l'ID demandé.

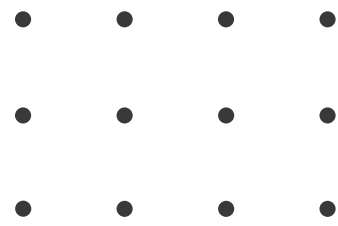
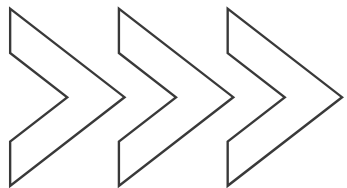


```
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window DB Navigator Help Other Springirun
jpa > src > main > java > com > spring > jpa > service > ProductService >
Project
> .idea
> m.mvn
src
  main
    java
      com
        spring
          jpa
            controller
              CategoryController
              CommentController
              ProductController
            model
            repository
            service
              CategoryService
              CommentService
              ProductService
JpaApplication
resources
  application.properties
test
target
.gitignore
assuranceAuto_db.sql
HELP.md
ina.iml

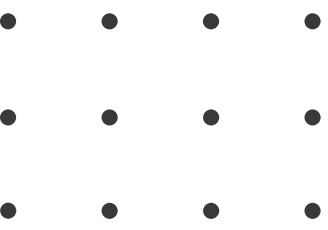
1 package com.spring.jpa.service;
2
3 import com.spring.jpa.model.Product;
4 import com.spring.jpa.repository.ProductRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.Optional;
9
10 /**
11  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
12  */
13 @Service
14 public class ProductService {
15     private final ProductRepository productRepository;
16
17     @Autowired
18     public ProductService(ProductRepository productRepository) {
19         this.productRepository = productRepository;
20     }
21
22     public Iterable<Product> getProducts() {
23         return this.productRepository.findAll();
24     }
25
26     public Optional<Product> getProductById(Integer id) {
27         return this.productRepository.findById(id);
28     }
29 }
30
```



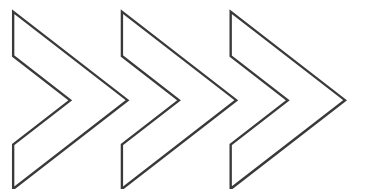
- Exposer la ressource via le controlleur

A screenshot of an IDE (IntelliJ IDEA) showing the code for ProductController.java. The file is located in the package com.spring.jpa.controller. It imports com.spring.jpa.model.Product, com.spring.jpa.service.ProductService, and various Spring annotations like @Autowired, @GetMapping, @PathVariable, @RequestMapping, and @RestController. It also imports java.util.Optional. The class ProductController is annotated with @RestController and @RequestMapping(path = "/api/v1/rest/products", name = "app_products_"). It has a constructor that takes a ProductService and sets it as a private final field. There are two methods: list() which returns an Iterable<Product> and show() which returns an Optional<Product> based on an id path variable. The IDE interface includes a project structure on the left, a toolbar at the top, and a status bar at the bottom with various tool icons and system information.

10 - Relations unidirectionnelles(OneToMany)



- `@OneToMany` : cette annotation permet de spécifier une relation 'une à plusieurs'. Pour un produit, il y a plusieurs commentaires possibles.
- La propriété `'cascade'` permet de définir quel impact l'action sur une entité aura sur son entité associée. Le type `'ALL'` signifie que toutes les actions sur l'entité Produit seront propagées sur l'entité Commentaires. Exemple : si on supprime le produit, les commentaires associés seront également supprimés.
- La propriété `orphanRemoval=true` permet d'activer un mécanisme qui garantit la non-existence de commentaire orphelin de son produit. Si on supprime un commentaire de la liste des commentaires du Product, alors le commentaire devient orphelin, et il est supprimé de la base de données.
- La propriété `fetch` possède la valeur `EAGER`, et cela signifie qu'à la récupération du produit, tous les commentaires seront également récupérés.
- `@JoinColumn` : Cette annotation permet d'indiquer le nom de la clé étrangère dans la table de l'entité concernée.



FileEditViewNavigateCodeAnalyzeRefactorBuildRunToolsGitWindowDB NavigatorHelpOtherSpringirun

jpa > src > main > java > com > spring > jpa > n

Project

Project

Commit

DB Browser

Pull Requests

TMS

Structure

Favorites

AWS Explorer

Product

main / 6 Δ

src

main

java

com

spring

jpa

controller

model

Category

Comment

Product

repository

service

JpaApplication

resources

application.properties

test

target

.gitignore

assuranceAuto_db.sql

HELP.md

jpa.iml

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

36

37

40

41

44

45

48

49

52

53

56

57

58

59

60

@Entity

@Table(name = "produit")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "produit_id")

private int id;

@Column(name = "nom")

private String name;

@Column(name = "cout")

private int cost;

@OneToMany(

cascade = CascadeType.ALL,

orphanRemoval = true,

fetch = FetchType.EAGER

)

@JoinColumn(name = "produit_id")

private final List<Comment> comments = new ArrayList<>();

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public int getCost() { return cost; }

public void setCost(int cost) { this.cost = cost; }

public List<Comment> getComments() {

return comments;

}

JPAAPPLICATION

PHP Inspections

CodeStream

Codota

Database

Maven

Git

Run

TODO

Problems

Statistic

Code Cleaner

Profiler

DB Execution Console

Terminal

Build

Spring

Event Log

Auto fetch: finished (12 minutes ago)

6 Δ/up-to-date

Blame: Globléhi 5/29/21, 12:00 PM

Material Darke

tobnine

60:2

SensioLabsInsight

LF

UTF-8

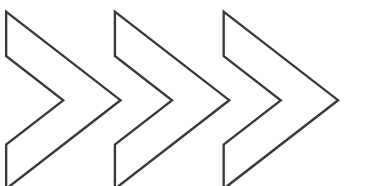
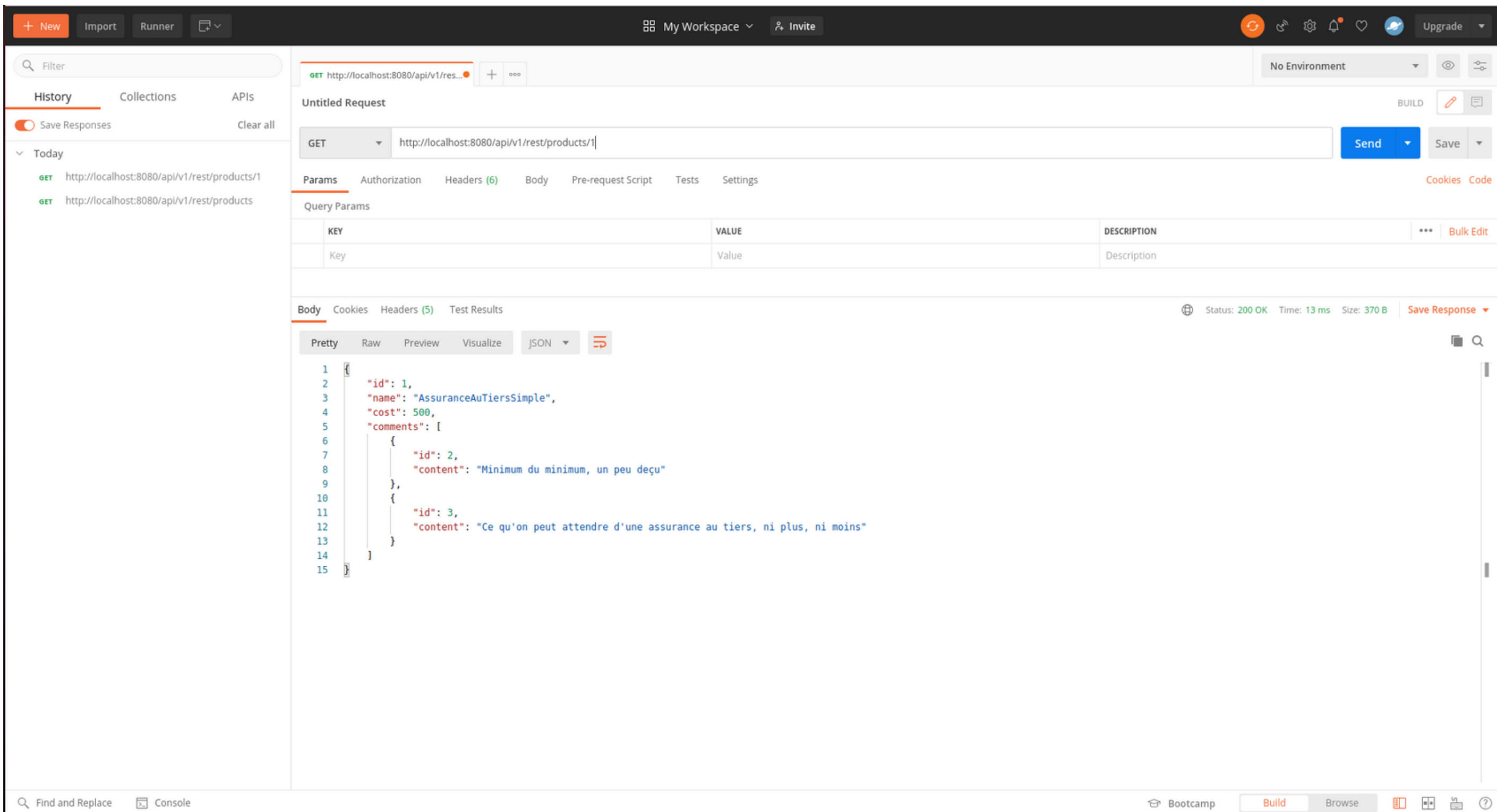
CodeStream: Sign in...

4 spaces

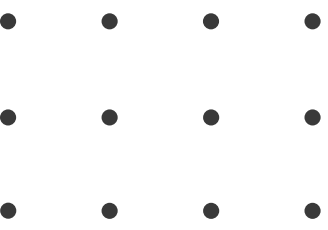
main

AWS: No credentials selected

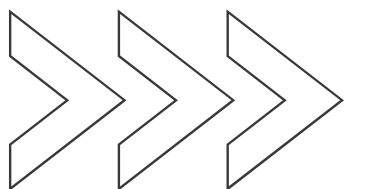
Test via postman



11 - Relations unidirectionnelles(ManyToMany)



- `@ManyToMany` : cette annotation permet de spécifier une relation ‘plusieurs à plusieurs’. Pour un produit, il y a plusieurs catégories possibles et pour une catégorie, il y a plusieurs produits possibles.
- La propriété `fetch` possède la valeur `LAZY`, et cela signifie qu’à la récupération de la catégorie, les produits ne sont pas récupérés. Par voie de conséquence, les performances sont meilleures (la requête est plus légère)
- `Cascade` : Je spécifie donc uniquement `PERSIST` et `MERGE`, la cascade s’applique donc tant en création qu’en modification.
- Il est également nécessaire d’utiliser l’annotation `@JoinTable` avec les propriétés suivantes :
 - `name` : correspond au nom de la table de jointure en base de données ;
 - `joinColumns` correspond à la clé étrangère dans la table de jointure ;
 - `inverseJoinColumns` correspond à la clé étrangère dans la table de jointure de la seconde entité concernée par la relation.



FileEditViewNavigateCodeAnalyzeRefactorBuildRunToolsGitWindowDB NavigatorHelpOtherSpringrun

jpa > src > main > java > com > spring > jpa > model > Category >

Project

jpa --~/Desktop/tutos/jpa main / 7 Δ

> .idea

> m.mvn

> src

main

java

com

spring

jpa

> controller

> model

Category

Comment

Product

> repository

> service

JpaApplication

> resources

</> application.properties

> test

> target

.gitignore

assuranceAuto_db.sql

HELP.md

jpa.iml

mvnw

mvnw.cmd

✓ pom.xml

> External Libraries

Scratches and Consoles

Commit

DB Browser

Pull Requests

TMS

Structure

Favorites

AWS Explorer

Category.java x

11 @table(name = "categorie")

12 public class Category {

13

14 @Id

15 @GeneratedValue(strategy = GenerationType.IDENTITY)

16 @Column(name="categorie_id")

17 private int id;

18

19 @Column(name="nom")

20 private String name;

21

22 @ManyToMany

23 fetch = FetchType.LAZY,

24 cascade = {

25 CascadeType.PERSIST,

26 CascadeType.MERGE

27 }

28 }

29 @JoinTable

30 name = "categorie_produit",

31 joinColumns = @JoinColumn(name = "categorie_id"),

32 inverseJoinColumns = @JoinColumn(name = "produit_id")

33 }

34 private final List<Product> products = new ArrayList<>();

35

36 public int getId() { return id; }

37

38

39

40 public void setId(int id) { this.id = id; }

41

42

43

44 public String getName() { return name; }

45

46

47

48 public void setName(String name) { this.name = name; }

49

50

51

52 public List<Product> getProducts() {

53 return products;

54 }

55 }

6 Globléhi, Today • implementation Model, Service and Repository of Produc, Comment and Category

7 Δ/up-to-date Blame: Globléhi 5/29/21, 12:00 PM Material Darke tabnine 55:2 SensioLabsInsight LF UTF-8 CodeStream: Sign in... 4 spaces main AWS: No credentials selected

JPAAPPLICATION

PHP Inspections

CodeStream

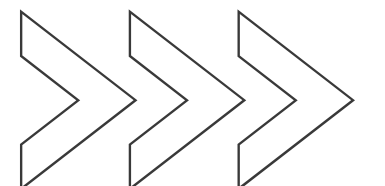
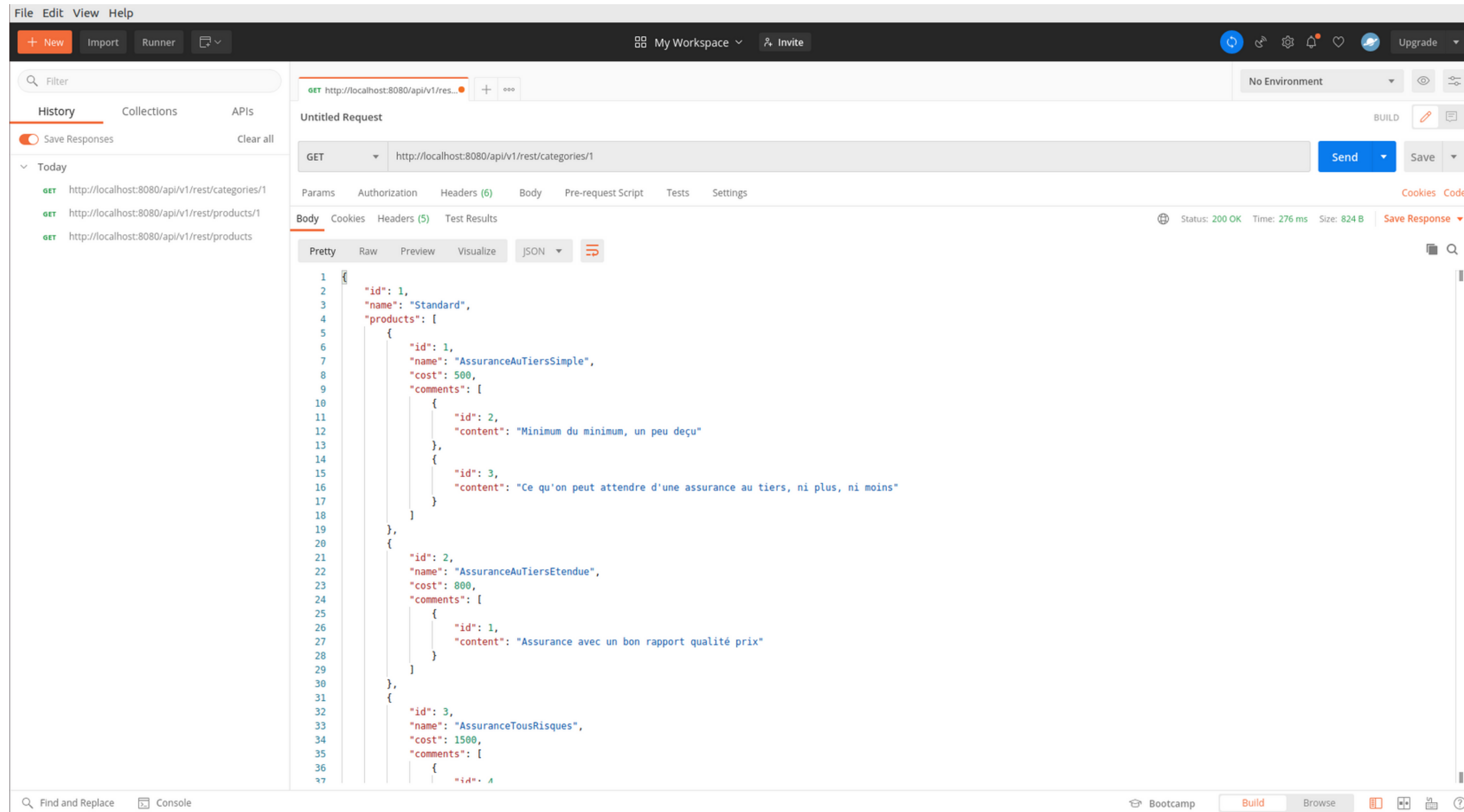
Codota

Database

Maven

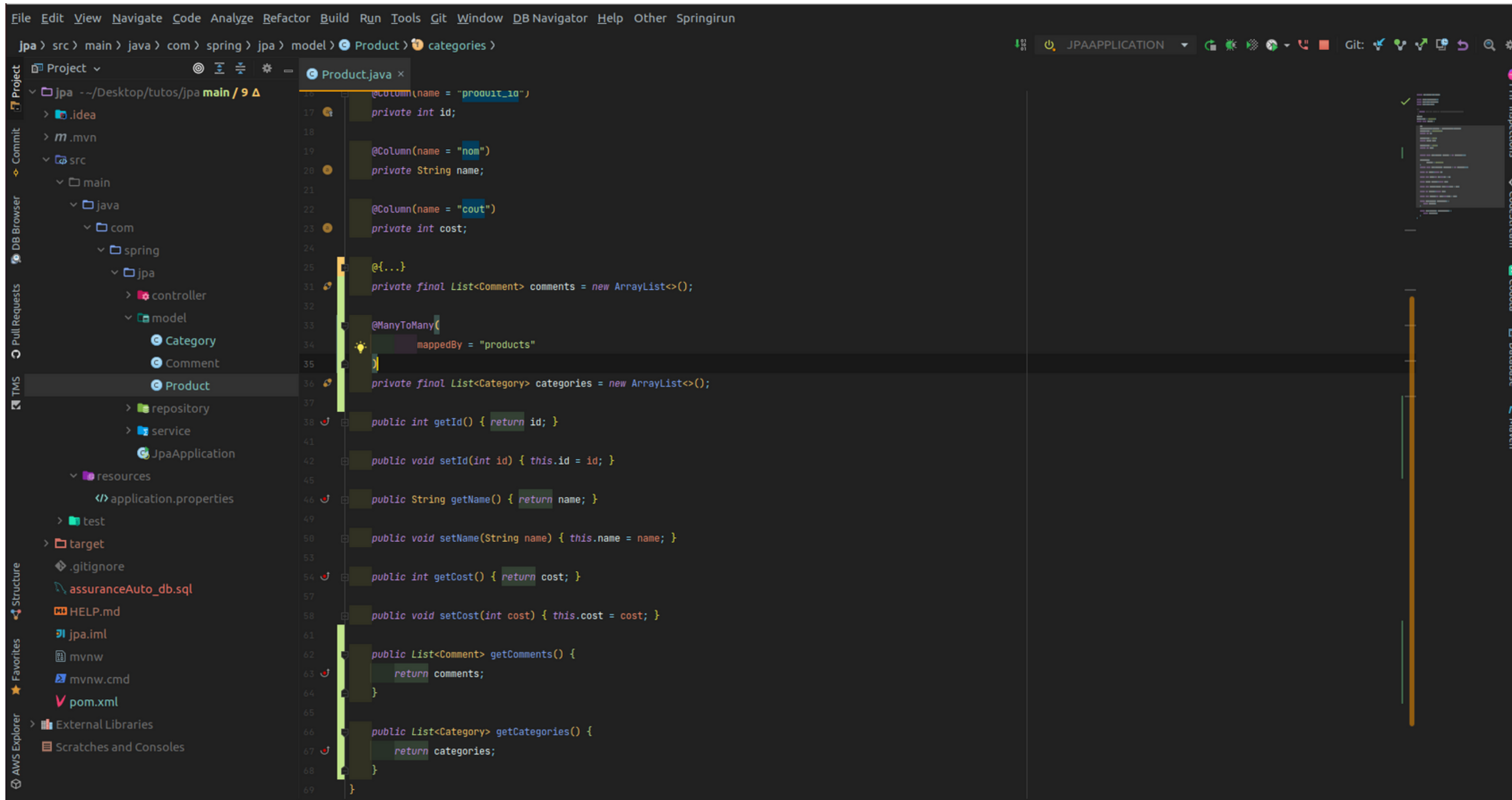
Event Log

Test via postman



12 - Relations bidirectionnelles(ManyToMany)

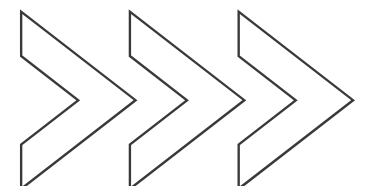
@ManyToMany est de rigueur, et l'unique propriété à utiliser est mappedBy. Cette propriété est valide dans le contexte de la bidirectionnalité, car elle indique le nom de l'attribut dans l'autre entité. Étant donné que dans la classe Category, l'annotation @ManyToMany est apposée sur l'attribut nommé “products”, alors la valeur de la propriété mappedBy est “products”.



```
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window DB Navigator Help Other Springirun
jpa > src > main > java > com > spring > jpa > model > Product > categories >

Project
  Jpa --/Desktop/tutos/jpa main / 9 Δ
    .idea
    m.mvn
    src
      main
        java
          com
            spring
              jpa
                controller
                model
                  Category
                  Comment
                  Product
                repository
                service
                JpaApplication
            resources
              application.properties
            test
            target
            .gitignore
            assuranceAuto_db.sql
            HELP.md
            jpa.iml
            mvnw
            mvnw.cmd
            pom.xml
          External Libraries
          Scratches and Consoles

Product.java x
  16 @Column(name = "produit_id")
  17 private int id;
  18
  19 @Column(name = "nom")
  20 private String name;
  21
  22 @Column(name = "cout")
  23 private int cost;
  24
  25 @{...}
  31 private final List<Comment> comments = new ArrayList<>();
  32
  33 @ManyToMany(
  34     mappedBy = "products"
  35 )
  36 private final List<Category> categories = new ArrayList<>();
  37
  38 public int getId() { return id; }
  41
  42 public void setId(int id) { this.id = id; }
  45
  46 public String getName() { return name; }
  49
  50 public void setName(String name) { this.name = name; }
  53
  54 public int getCost() { return cost; }
  57
  58 public void setCost(int cost) { this.cost = cost; }
  61
  62 public List<Comment> getComments() {
  63     return comments;
  64 }
  65
  66 public List<Category> getCategories() {
  67     return categories;
  68 }
  69 }
```

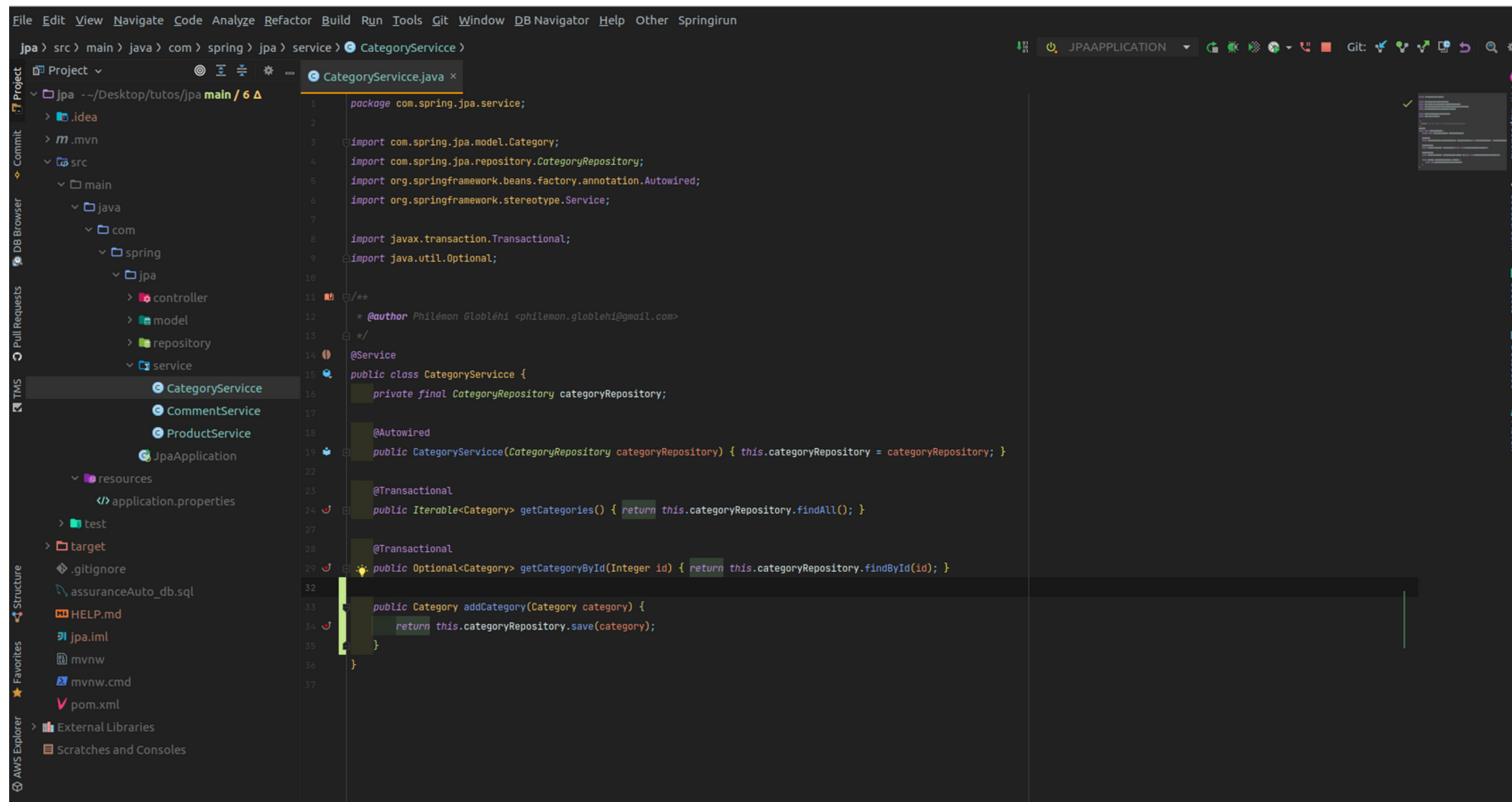


13 - Persister des données

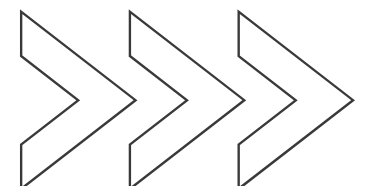
Spring Data JPA va grandement nous simplifier la tâche, et ainsi nous faire gagner du temps !

Tout comme pour la récupération de données, l'interface CrudRepository nous offre des méthodes déjà implémentées.

Les méthodes **save** et **saveAll** permettent de sauvegarder (ou persister) en base de données



```
1 package com.spring.jpa.service;
2
3 import com.spring.jpa.model.Category;
4 import com.spring.jpa.repository.CategoryRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import javax.transaction.Transactional;
9 import java.util.Optional;
10
11 /**
12  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
13  */
14 @Service
15 public class CategoryService {
16     private final CategoryRepository categoryRepository;
17
18     @Autowired
19     public CategoryService(CategoryRepository categoryRepository) { this.categoryRepository = categoryRepository; }
20
21     @Transactional
22     public Iterable<Category> getCategories() { return this.categoryRepository.findAll(); }
23
24     @Transactional
25     public Optional<Category> getCategoryById(Integer id) { return this.categoryRepository.findById(id); }
26
27     public Category addCategory(Category category) {
28         return this.categoryRepository.save(category);
29     }
30 }
```



FileEditViewNavigateCodeAnalyzeRefactorBuildRunToolsGitWindowDB NavigatorHelpOtherSpringirun

jpa > src > main > java > com > spring > jpa > controller > CategoryController >

Project

jpa --/Desktop/tutos/jpa main / 6 Δ

.idea

.mvn

src

main

java

com

spring

jpa

controller

CategoryController

CommentController

ProductController

model

repository

service

JpaApplication

resources

application.properties

test

target

.gitignore

assuranceAuto_db.sql

HELP.md

jpa.iml

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

Commit

DB Browser

Pull Requests

TMS

Structure

Favorites

AWS Explorer

CategoryController.java x

```
1 package com.spring.jpa.controller;
2
3 import com.spring.jpa.model.Category;
4 import com.spring.jpa.service.CategoryService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.Optional;
9
10 /**
11  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
12  */
13 @RestController
14 @RequestMapping(path = "/api/v1/rest/categories", name = "app_categories_")
15 public class CategoryController {
16     private final CategoryService categoryService;
17
18     @Autowired
19     public CategoryController(CategoryService categoryService) { this.categoryService = categoryService; }
20
21     @GetMapping(name = "list")
22     public Iterable<Category> list() { return this.categoryService.getCategories(); }
23
24     @GetMapping(path =("/{id}", name = "details")
25     public Optional<Category> show(@PathVariable int id) { return this.categoryService.getCategoryById(id); }
26
27     @PostMapping(name = "add")
28     public Category save(@RequestBody Category category) { return this.categoryService.addCategory(category); }
29 }
30
```

PHP Inspections

CodeStream

Codota

Database

Maven

Git

Run

TODO

Problems

Statistic

Code Cleaner

Profiler

DB Execution Console

Terminal

Build

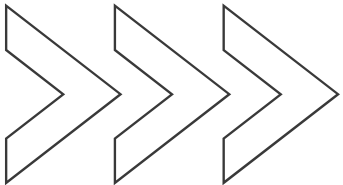
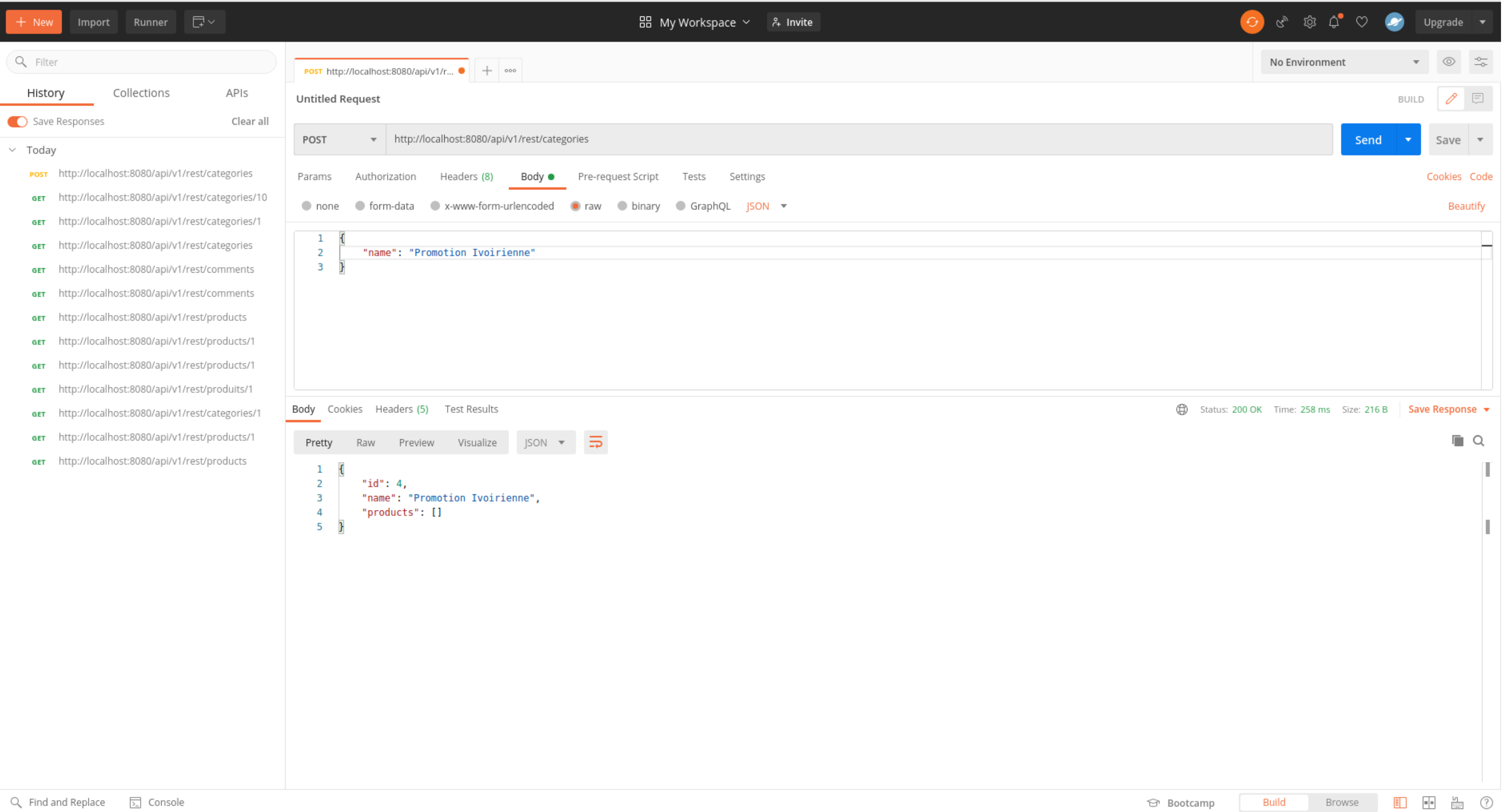
Spring

Event Log

Build completed successfully in 3 sec, 380 ms (7 minutes ago)

6 Δ/up-to-date Material Darker tobnine 38:1 SensioLabsInsight LF UTF-8 CodeStream: Sign in... 4 spaces main AWS: No credentials selected

Test via postman

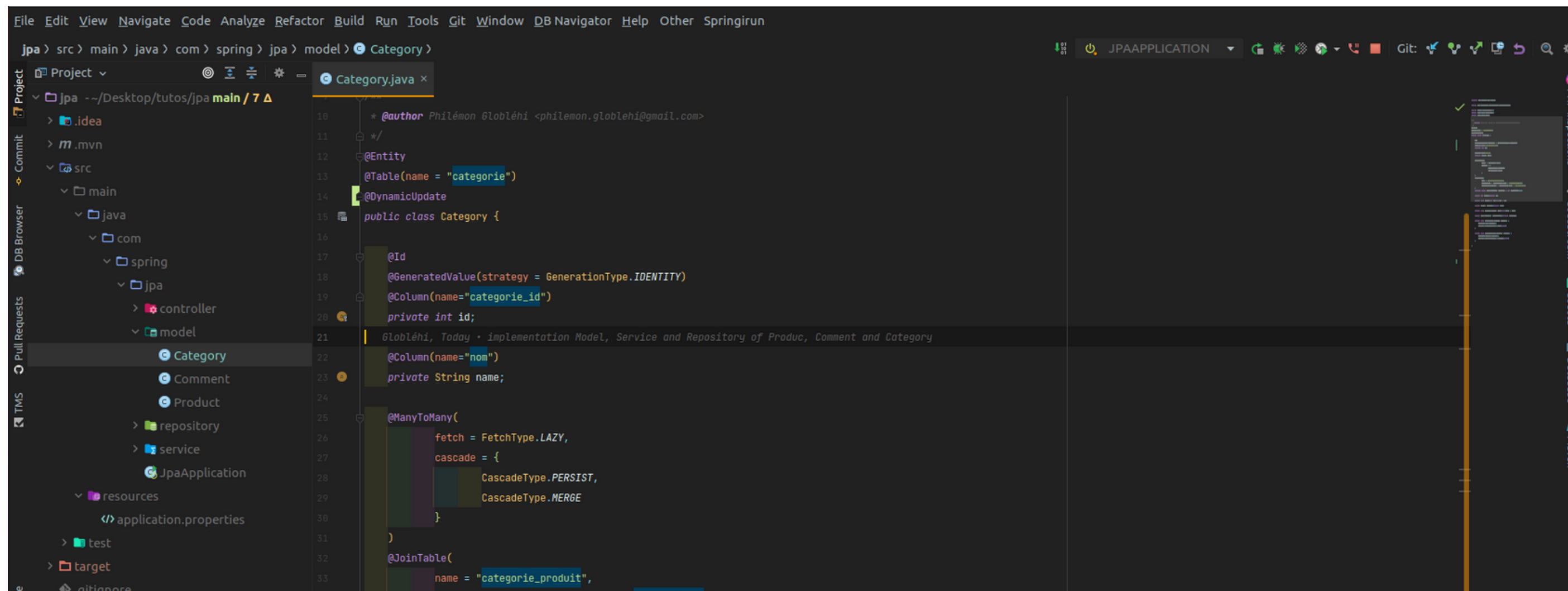


14 - Modifier des données

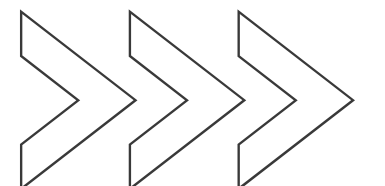
La création et la mise à jour d'un objet sont gérées par les mêmes méthodes `save(S entity)` et `saveAll(Iterable<S> entities)`.

Spring Data JPA va inspecter le contenu de l'objet que nous lui fournissons, et pourra identifier s'il est déjà existant en base ou non, grâce à son identifiant unique (la clé primaire).

Ajoutons l'annotation `@DynamicUpdate` à nos models. Cette annotation doit être appliquée sur l'entité (dans notre exemple, la classe `Category`). Si elle est présente, alors la requête SQL update exécutera un set uniquement sur les valeurs qui ont changé et permet de booster vos performances



```
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window DB Navigator Help Other Springrun
jpa > src > main > java > com > spring > jpa > model > Category >
Project
  jpa --/Desktop/tutos/jpa main / 7 Δ
  .idea
  .mvn
  src
    main
      java
        com
          spring
            jpa
              controller
              model
                Category
                Comment
                Product
              repository
              service
                JpaApplication
              resources
                application.properties
              test
              target
              gitignore
Commit
DB Browser
Pull Requests
TMS
Category.java x
10  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
11  */
12  @Entity
13  @Table(name = "categorie")
14  @DynamicUpdate
15  public class Category {
16
17      @Id
18      @GeneratedValue(strategy = GenerationType.IDENTITY)
19      @Column(name="categorie_id")
20      private int id;
21
22      Globléhi, Today • implementation Model, Service and Repository of Produc, Comment and Category
23      @Column(name="nom")
24      private String name;
25
26      @ManyToMany(
27          fetch = FetchType.LAZY,
28          cascade = {
29              CascadeType.PERSIST,
30              CascadeType.MERGE
31          }
32      )
33      @JoinTable(
34          name = "categorie_produit",
```



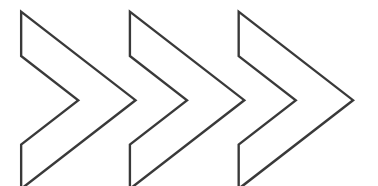
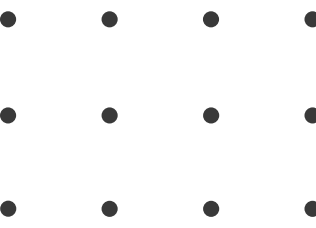
15 - Supprimer des données

Spring Data JPA tient vraiment à nous faciliter la tâche pour la suppression

Pas moins de 4 méthodes sont à notre disposition :

- `delete(S entity)`
- `deleteAll()`
- `deleteAll(Iterable<S> entities)`
- `deleteById(Integer id)`

Il est important, dans le cas de la suppression, de bien tenir compte du type de CASCADE qui a été spécifié au niveau des relations. Un `CascadeType.ALL` ou un `CascadeType.REMOVE` déclenche une suppression en cascade en fonction des associations. Et cela peut avoir de graves conséquences !



FileEditViewNavigateCodeAnalyzeRefactorBuildRunToolsGitWindowDB NavigatorHelpOtherSpringirun

jpa > src > main > java > com > spring > jpa > service > CommentService >

Project

jpa -~/Desktop/tutos/jpa main / 1 Δ

.idea

m.mvn

src

main

java

com

spring

jpa

controller

model

repository

service

CategoryService

CommentService

ProductService

JpaApplication

resources

application.properties

test

target

.gitignore

assuranceAuto_db.sql

HELP.md

jpa.iml

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

CategoryService.java x

CommentService.java x

```
6 import org.springframework.stereotype.Service;
7
8 import javax.transaction.Transactional;
9 import java.util.Optional;
10
11 /**
12  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
13  */
14 @Service
15 public class CommentService {
16     private final CommentRepository commentRepository;
17
18     @Autowired
19     public CommentService(CommentRepository commentRepository) { this.commentRepository = commentRepository; }
20
21
22     @Transactional
23     public Iterable<Comment> getComments() { return this.commentRepository.findAll(); }
24
25     Globléhi, Today • Implementation comments and categories endpoints
26
27     @Transactional
28     public Optional<Comment> getCommentById(Integer id) { return this.commentRepository.findById(id); }
29
30
31     public Comment addComment(Comment comment) { return this.commentRepository.save(comment); }
32
33
34     public void removeComment(Comment comment) {
35         this.commentRepository.delete(comment);
36     }
37
38     public void removeCommentById(Integer id) {
39         this.commentRepository.deleteById(id);
40     }
41
42 }
43
44
45
```

PHP Inspections

CodeStream

Codota

Database

Maven

Git

Run

TODO

Problems

Statistic

Code Cleaner

Profiler

DB Execution Console

Terminal

Build

Spring

Auto fetch: finished (a minute ago)

1 Δ/up-to-date

Blame: Globléhi 5/29/21, 5:15 PM

Material Darker

tobnine

27:1

SensioLabsInsight

LF

UTF-8

CodeStream: Sign in...

4 spaces

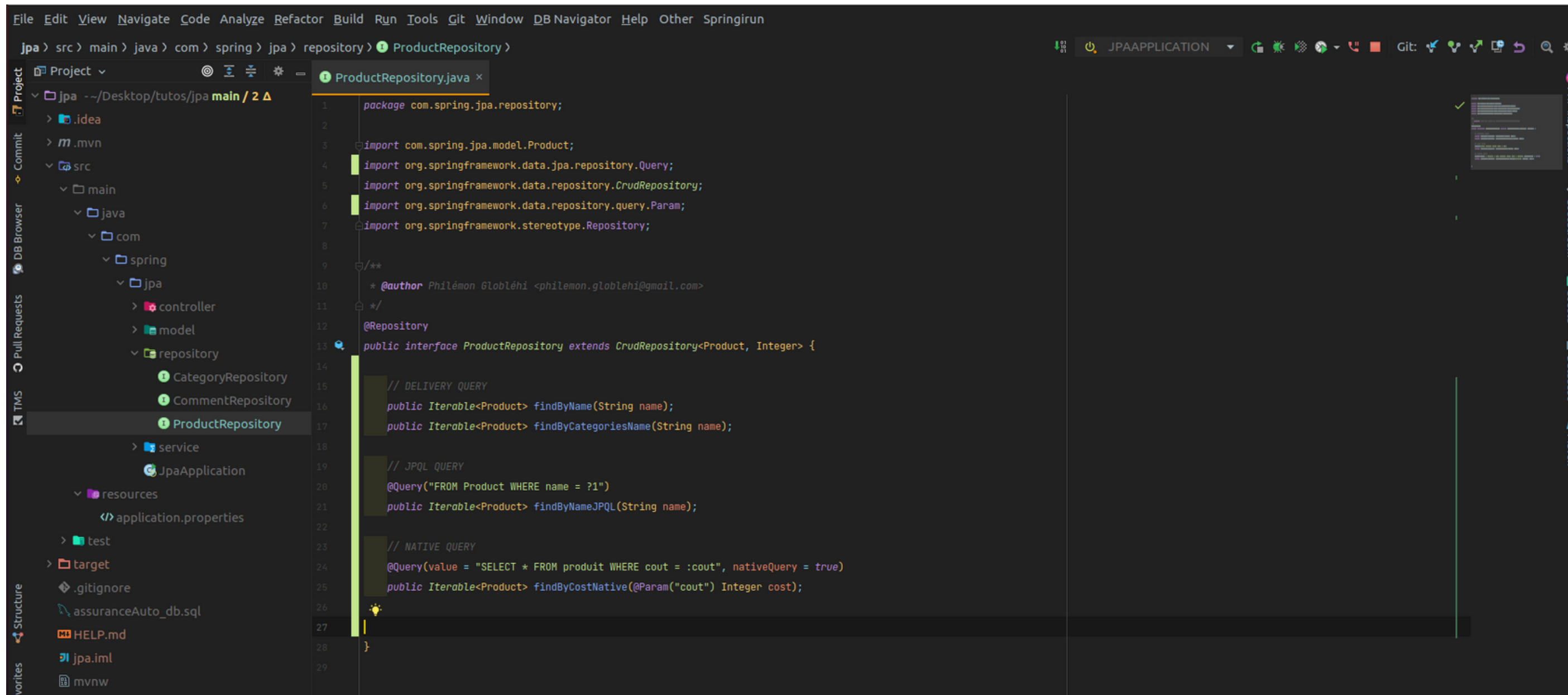
main

AWS: No credentials selected

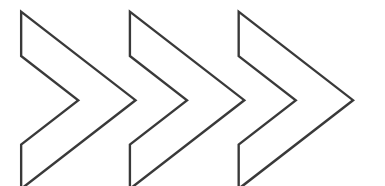
16 - Requêtes personnalisées

Les requêtes personnalisées sont des requêtes qui ne sont pas incluses par défaut dans l'interface CrudRepository, on a :

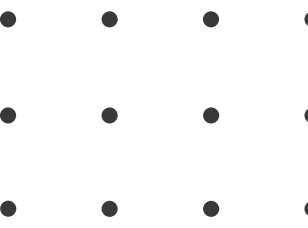
- les requêtes dérivées;
- les requêtes JPQL;
- les requêtes natives SQL.



```
File Edit View Navigate Code Analyze Refactor Build Run Tools Git Window DB Navigator Help Other Springrun
jpa > src > main > java > com > spring > jpa > repository > ProductRepository >
ProductRepository.java x
1 package com.spring.jpa.repository;
2
3 import com.spring.jpa.model.Product;
4 import org.springframework.data.jpa.repository.Query;
5 import org.springframework.data.repository.CrudRepository;
6 import org.springframework.data.repository.query.Param;
7 import org.springframework.stereotype.Repository;
8
9 /**
10  * @author Philémon Globléhi <philemon.globlehi@gmail.com>
11  */
12 @Repository
13 public interface ProductRepository extends CrudRepository<Product, Integer> {
14
15     // DELIVERY QUERY
16     public Iterable<Product> findByName(String name);
17     public Iterable<Product> findByCategoriesName(String name);
18
19     // JPQL QUERY
20     @Query("FROM Product WHERE name = ?1")
21     public Iterable<Product> findByNameJPQL(String name);
22
23     // NATIVE QUERY
24     @Query(value = "SELECT * FROM produit WHERE cout = :cout", nativeQuery = true)
25     public Iterable<Product> findByCostNative(@Param("cout") Integer cost);
26
27 }
28
29
```



Les requêtes dérivées



À la ligne 14, j'ai rajouté le prototype d'une nouvelle méthode, `public Iterable<Product> findByName(String name);`

Le nom de cette méthode `findByName` n'a pas été construit au hasard :

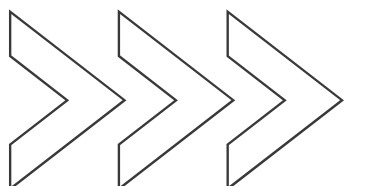
- Le mot `find` correspond à la récupération de données.
- Le mot `By` est un mot clé utilisé par Spring Data JPA pour indiquer que la requête aura un paramètre.
- Le mot `Name` correspond à l'attribut de l'entité qui est notre critère de recherche.

Cette méthode est une requête dérivée (Derived Query) car nous n'allons pas en écrire l'implémentation. Spring Data JPA est capable de générer le code nécessaire à partir du prototype de la méthode.

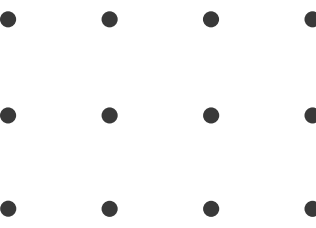
les requêtes dérivées ont la capacité de traverser les relations .

La ligne 15 définit une nouvelle méthode `findByCategoriesName` :

- `find` pour la récupération de données ;
- `By` pour introduire des critères ;
- `Categories` car l'attribut au sein de l'entité `Product` se nomme `categories` ;
- `Name` car au sein de l'entité `Category` il y a l'attribut `name`.



Les requêtes JPQL



JPQL signifie Java Persistence Query Language. Ce langage de requête est défini au sein de l'API JPA, et permet d'exécuter des requêtes via les entités et non via les tables directement.

La ligne 21 définit une méthode `findByNameJPQL` utilise l'annotation `@Query` avec une requête JPQL. On note qu'il n'y a pas de clause `SELECT`, et ce n'est pas le nom de la table mais le nom de l'entité qui est indiqué après le `FROM`.

Les requêtes natives

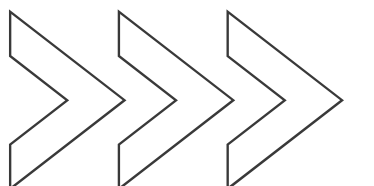
Les requêtes SQL natives permettent d'exécuter des requêtes SQL classiques en manipulant les tables directement.

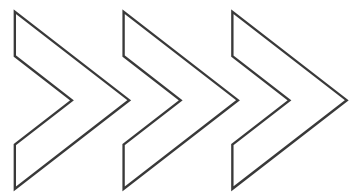
La ligne 25 définit une méthode `findByNameJPQL` utilise l'annotation `@Query` avec une requête JPQL

La méthode `findByCostNative` utilise l'annotation `@Query` avec une requête SQL native.

3 aspects particuliers :

1. La requête est dans une propriété value de l'annotation.
2. La propriété `nativeQuery` de l'annotation doit être à `true`.
3. Le paramètre au sein de la méthode doit être annoté `@Param` afin que l'association soit faite entre le paramètre et la requête SQL.





Merci

Philémon Globléhi, Back-end Developer & DevOps Lover
Github: <https://github.com/philemongloblehi>
Linkedin: <https://www.linkedin.com/in/philemon-globlehi/>