Atelier Framework Spring-04
Thymeleaf – Partie 01

Objectifs

- **Utilisation du module Spring Web avec le moteur Thymeleaf**
  - Ajouter les dépendances MAVEN
  - Gérer un contrôleur et utiliser Spring MVC
  - Afficher une vue avec Thymeleaf

A. Création d'un projet Spring Boot avec Thymeleaf

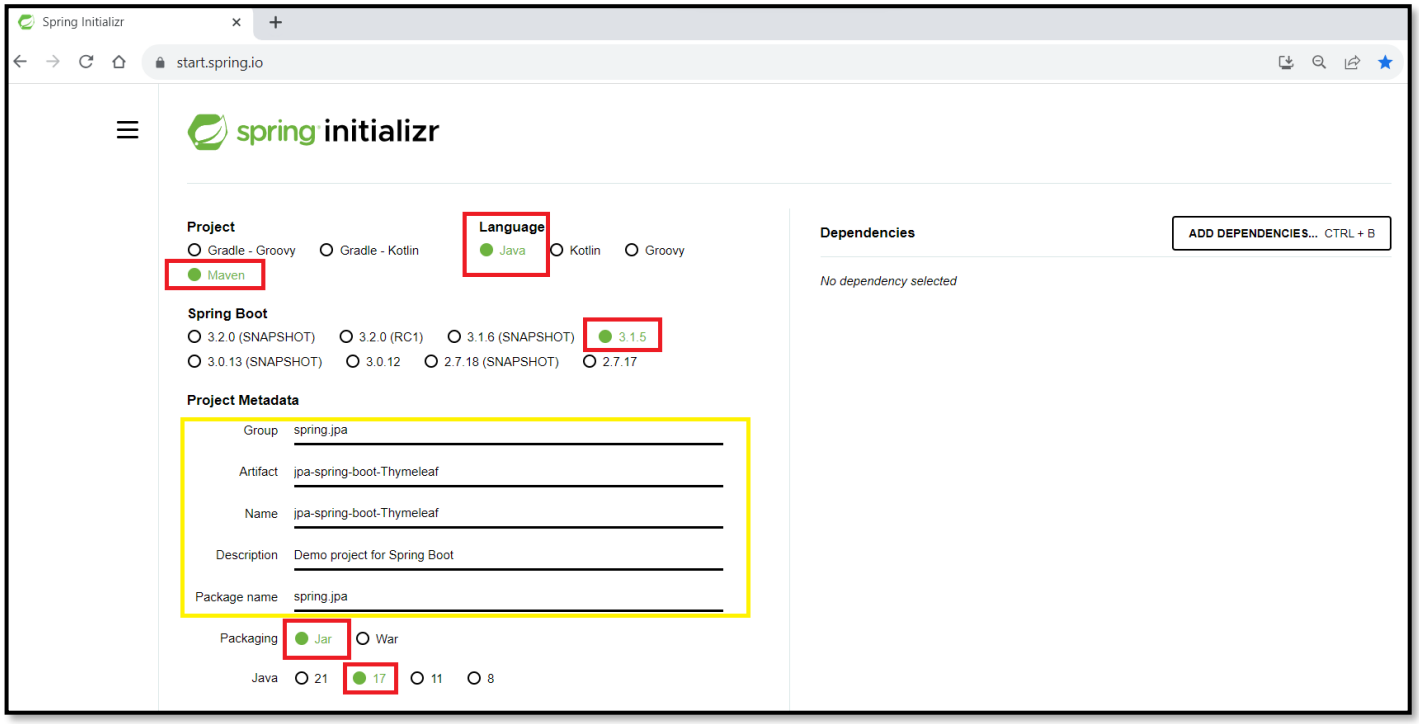
1. Pour générer rapidement un projet Spring Boot, il est possible, aussi, d'utiliser l'API de **Spring Initialz** à travers le lien suivant :

https://start.spring.io/

a) Créer un projet **Spring boot** avec les paramètres suivants :

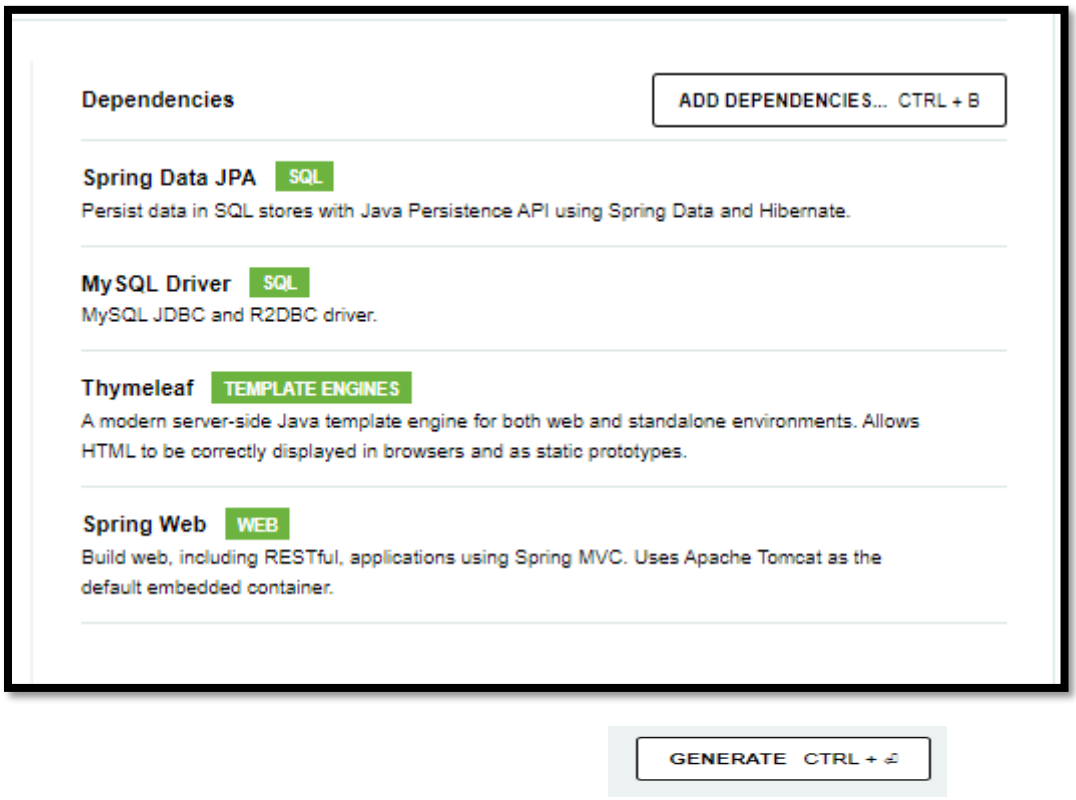
- ✓ Groupe : **spring.jpa**
- ✓ Artifact :**jpa-spring-boot-Thymeleaf**
- ✓ Nom : **jpa-spring-boot-Thymeleaf**
- ✓ Description : **Gestion des produits avec Thymeleaf**
- ✓ Package : **spring.jpa**

- ✓ **Project : Maven**
- ✓ **Language : Java**
- ✓ **Spring Boot : 3.1.5**
- ✓ **Packaging : jar**
- ✓ **Java : 17**



b) Spécifier les dépendances suivantes après avoir appuyé, chaque fois, sur le bouton: ADD DEPENDENCIES... CTRL + B

- **Spring Data JPA**
- **MySQL Driver**
- **Thymeleaf (dans la catégorie « Template Engines »)**
- **Spring Web**

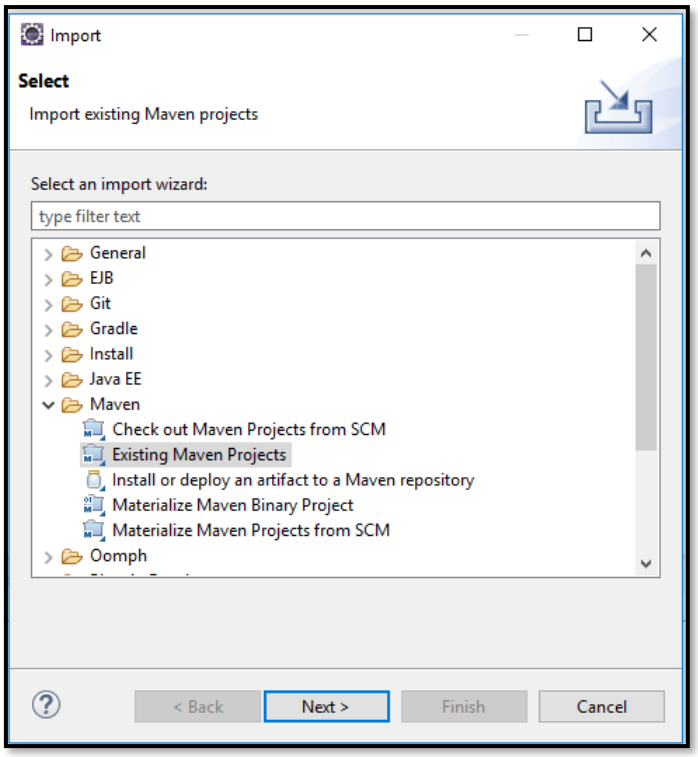


c) Appuyer, finalement, sur le bouton pour télécharger le projet sous un format compressé.

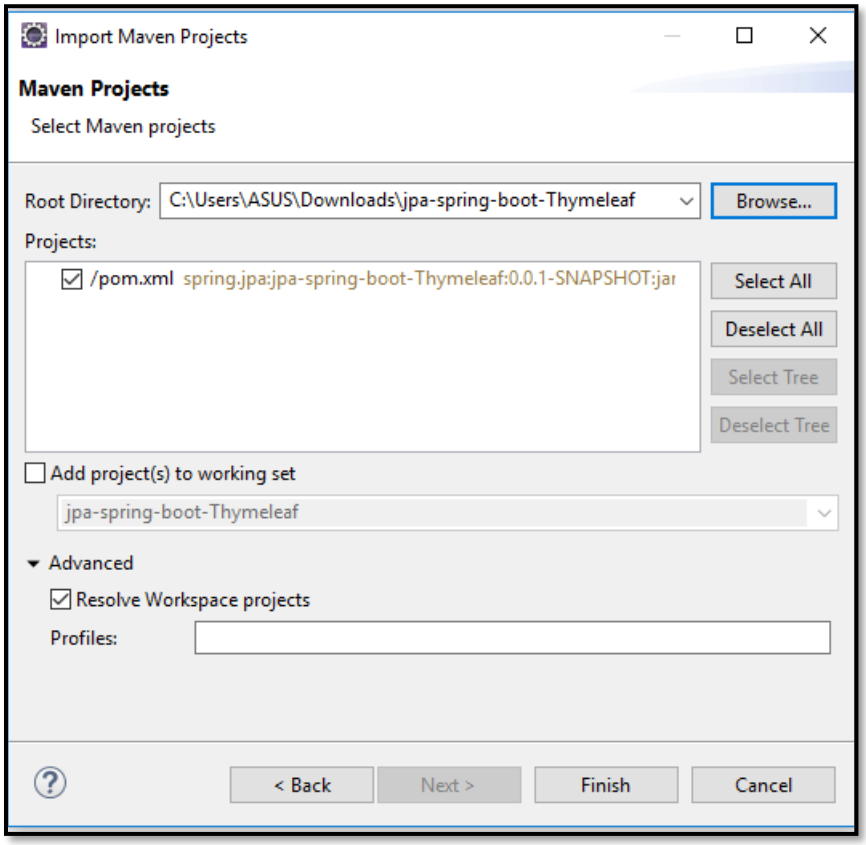
d) Décompresser le projet

e) Aller à eclipse pour importer le projet compressé :

- Aller au menu «**File/Import/Maven/Existing Maven Projects**»

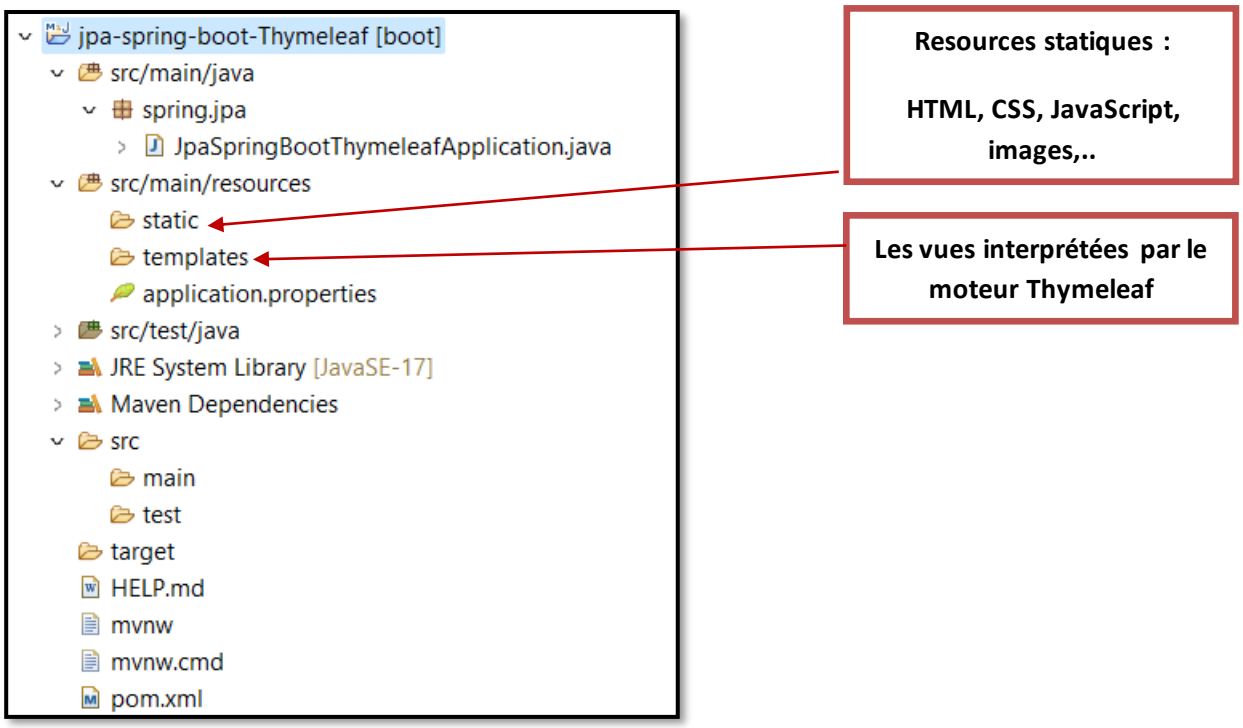


- Appuyer sur «**Next**» pour sélection le projet : «**jpa-spring-boot-Thymeleaf**» :

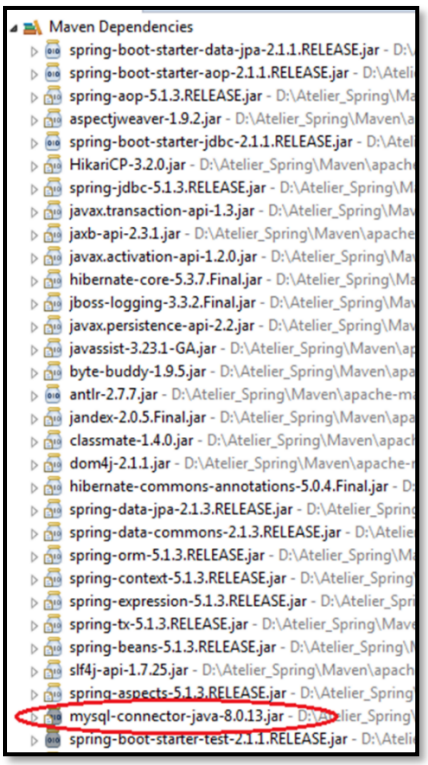


- Appuyer sur « **Finish** » pour achever l'opération d'importation

2. Voici la structure du projet ainsi créé :



3. Remarquer, dans le volet « **Project Explorer** », la présence des dépendances suivantes : (Exemple : lorsque vous indiquez le besoin du module JPA, Spring Boot va télécharger toutes les JAR nécessaires à cette spécification.



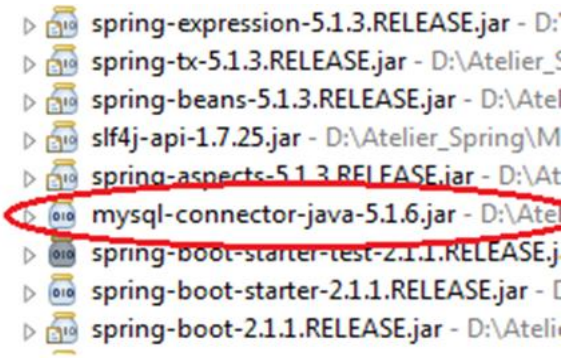
4. Ouvrir le fichier « **pom.xml** »
5. Sélectionner la dépendance suivante de « **mysql** »

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

6. La remplacer par la déclaration suivante : (afin de choisir la version 5)

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
</dependency>
```

7. Remarquer le changement dans la liste des dépendances de Maven :



B. Configurer la couche "JPA"

8. Ajouter dans dossier «**src/main/resources**» le fichier de configuration « **application.properties** » ayant le code suivant :

```
#database Configuration:
spring.datasource.driverClassName=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/springJpaThymeleaf?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
#Hibernate Configuration:
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
#spring.main.banner-mode=off
spring.jpa.properties.hibernate.enable_lazy_load_no_trans=true
```

NB : il s'agit d'une nouvelle base de données «**springJpaThymeleaf**». Avec cette configuration, Spring crée la base de données si elle n'est pas existante.

9. Ajouter dans dossier «**src/main/java**» le package «**spring.jpa.model**».

10. Créer dans le package «**spring.jpa.model**» les deux entités suivantes :

a) Entité « **Categorie** » :

```
package spring.jpa.model;

import java.util.ArrayList;
import java.util.Collection;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
```

```java
import jakarta.persistence.OneToMany;

@Entity
public class Categorie
{
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String code;

	@Column(length = 50)
	private String libelle;


	@OneToMany (mappedBy = "categorie" ,cascade = {CascadeType.REMOVE,
CascadeType.MERGE, CascadeType.PERSIST} )
    private Collection <Produit> produits = new ArrayList<Produit>();

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

	public void setProduits(Collection<Produit> produits) {
		this.produits = produits;
	}

	public Categorie(String code, String libelle) {
		super();
		this.code = code;
		this.libelle = libelle;
	}

	@Override
	public String toString() {
		return "Categorie [id=" + id + ", code=" + code + ", libelle=" +
libelle + "]";
	}

	public Categorie() {}

	public Long getId() {
		return id;
	}

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

```
	public String getCode() {
		return code;
	}

	public void setCode(String code) {
		this.code = code;
	}

	public String getLibelle() {
		return libelle;
	}

	public void setLibelle(String libelle) {
		this.libelle = libelle;
	}
}
```

- Entité « **Produit** » :

```
package spring.jpa.model;


import java.util.Date;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Temporal;
import jakarta.persistence.TemporalType;

@Entity
public class Produit {
	@Id
	@GeneratedValue
	private Long id;

	@Column(length = 50)
	private String designation;

	private double prix;
	private int quantite;

	@Temporal(TemporalType.DATE)
	java.util.Date dateAchat;
```

```
	@ManyToOne
	private Categorie categorie;


	public Produit(String designation, double prix, int quantite, Date
dateAchat, Categorie categorie) {
		super();
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
		this.dateAchat = dateAchat;
		this.categorie = categorie;
	}

	public Categorie getCategorie() {
		return categorie;
	}

	public void setCategorie(Categorie categorie) {
		this.categorie = categorie;
	}

	public java.util.Date getDateAchat() {
		return dateAchat;
	}

	public void setDateAchat(java.util.Date dateAchat) {
		this.dateAchat = dateAchat;
	}

	public Produit(String designation, double prix, int quantite, Date
dateAchat) {
		super();
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
		this.dateAchat = dateAchat;
	}

	public Long getId() {
		return id;
	}

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

	public String getDesignation() {
		return designation;
	}
```

```
	public void setDesignation(String designation) {
		this.designation = designation;
	}

	public double getPrix() {
		return prix;
	}

	public void setPrix(double prix) {
		this.prix = prix;
	}

	public int getQuantite() {
		return quantite;
	}

	public void setQuantite(int quantite) {
		this.quantite = quantite;
	}
	public Produit() {
		super();

	}
	public Produit(String designation, double prix, int quantite) {
		super();
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
	}
	@Override
	public String toString() {
		return "Produit [id=" + id + ", designation=" + designation +
", prix=" + prix + ", quantite=" + quantite + ", dateAchat=" + dateAchat +
 ", categorie=" + categorie + "]";
	}
	public Produit(Long id, String designation, double prix, int quantite) {
		super();
		this.id = id;
		this.designation = designation;
		this.prix = prix;
		this.quantite = quantite;
	}
}
```

11. Ajouter dans dossier «**src/main/java**» le package «**spring.jpa.repository**».

12. Créer dans le package «**spring.jpa.repository**» les deux interfaces suivantes :

- Interface «**ProduitRepository**» :

```
package spring.jpa.repository;
```

```
import org.springframework.data.jpa.repository.JpaRepository;

import spring.jpa.model.Produit;

public interface ProduitRepository  extends JpaRepository<Produit, Long>  {}
```

- Interface «**CategorieRepository**» :

```
package spring.jpa.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import spring.jpa.model.Categorie;

public interface CategorieRepository extends JpaRepository<Categorie, Long> {}
```

13. Lancer l'exécution du projet pour générer le schéma des tables correspondantes :

14. Remarquer le démarrage du serveur web « **Tomcat** » dans le volet « **Console** » :

```
main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
main] ion$DefaultTemplateResolverConfiguration : Cannot find template location: classpath:/templates/ (please add some templates or ch
main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
main] i.m.s.JpaSpringBootThymeleafApplication  : Started JpaSpringBootThymeleafApplication in 7.78 seconds (JVM running for 8.556)
```

15. Pour interdire l'affichage des « logs » dans la console, Ajouter dans dossier «**src/main/resources**» le fichier «**logBack.xml**» ayant le code suivant :

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

C. Création de la couche "contrôleur"

16. Ajouter dans dossier «**src/main/java**» le package «**spring.jpa.controller**».

17. Créer dans le package «**spring.jpa.controller**» la classe «**ProduitController**» ayant le code suivant :

```
package spring.jpa.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
```

```java
import spring.jpa.repository.ProduitRepository;

@Controller                          // pour déclarer un contrôleur
@RequestMapping (value = "/produit")  // nom logique dans l'URL pour accéder au contrôleur
public class ProduitController
{
      @Autowired       // pour l'injection de dépendances
      private ProduitRepository produitRepos;


@RequestMapping (value = "/index")   // nom logique pour accéder à l'action "index" ou méthode "index
      public String index ()
      {
            return "produits";    //retourner le nom de la vue WEB à afficher
      }
}
```

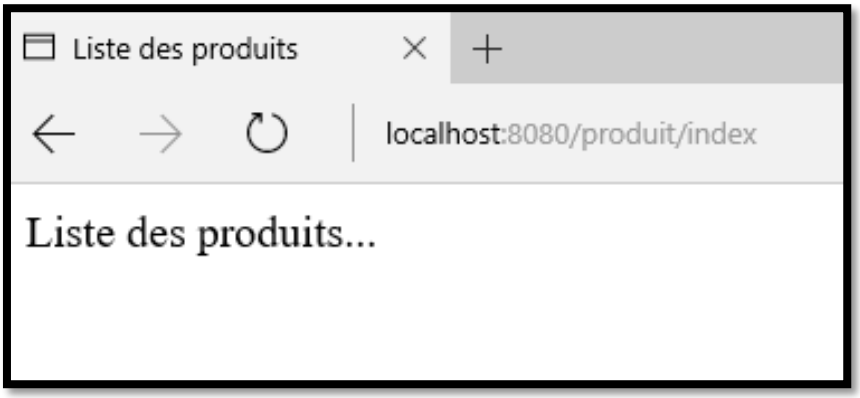
18. Créer dans le dossier «**src/main/resources/templates**» le fichier «**produits.html**» ayant le simple code suivant :

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>Liste des produits </title>
</head>
<body>
     Liste des produits...
</body>
</html>
```

19. Relancer l'exécution du projet puis aller au navigateur web et saisir l'URL suivante :

http://localhost:8080/produit/index

20. Vérifier l'exécution suivante :



D. Affichage des données (Spring MVC)

21. Retourner au contrôleur «**ProduitController**» et utiliser la nouvelle version suivante :

```
package spring.jpa.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import spring.jpa.model.Produit;
import spring.jpa.repository.ProduitRepository;

@Controller       // pour déclarer un contrôleur
@RequestMapping (value = "/produit")  // nom logique dans l'URL pour accéder au contrôleur
public class ProduitController  {
      @Autowired       // pour l'injection de dépendances
      private ProduitRepository produitRepos;

// nom logique pour accéder à l'action "index" ou méthode "index
@RequestMapping (value = "/index")
public String index (Model model )
      {
      //récupérer la liste des produits à partir de la couche "service"
      List <Produit> lp = produitRepos.findAll();
      // placer la liste des produits dans le "Model" comme un attribut"
      model.addAttribute("produits", lp);
      //retourner le nom de la vue WEB à afficher
      return "produits";
      }
}
```

22. Prendre une nouvelle version de la vue «**produits.html**» pour afficher sous forme d'un tableau la liste des produits existants dans la BD (Cette version utilise le moteur web **Thymeleaf**) :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8"/>
<title>Liste des Produits</title>

</head>
<body>
<h3>Liste des produits</h3>

<table>
  <thead>
   <tr>
   <th>ID</th>
   <th>Désignation</th>
   <th>Prix</th>
   <th>Quantité</th>
   <th>Date Achat</th>
   <th>Action</th>
  </thead>
  <tbody>

      <tr th:each="p:${produits}">

            <td th:text="${p.id}" ></td>
            <td th:text="${p.designation}" ></td>
            <td th:text="${p.prix}" ></td>
            <td th:text="${p.quantite}" ></td>
            <td th:text="${p.dateAchat}" ></td>
      </tr>
  </tbody>
</table>
</body>
</html>
```

23. Prendre une nouvelle version de la classe «**JpaSpringBootThymeleafApplication**» pour implémenter un exemple d'insertion de données :

```
package spring.jpa;
```

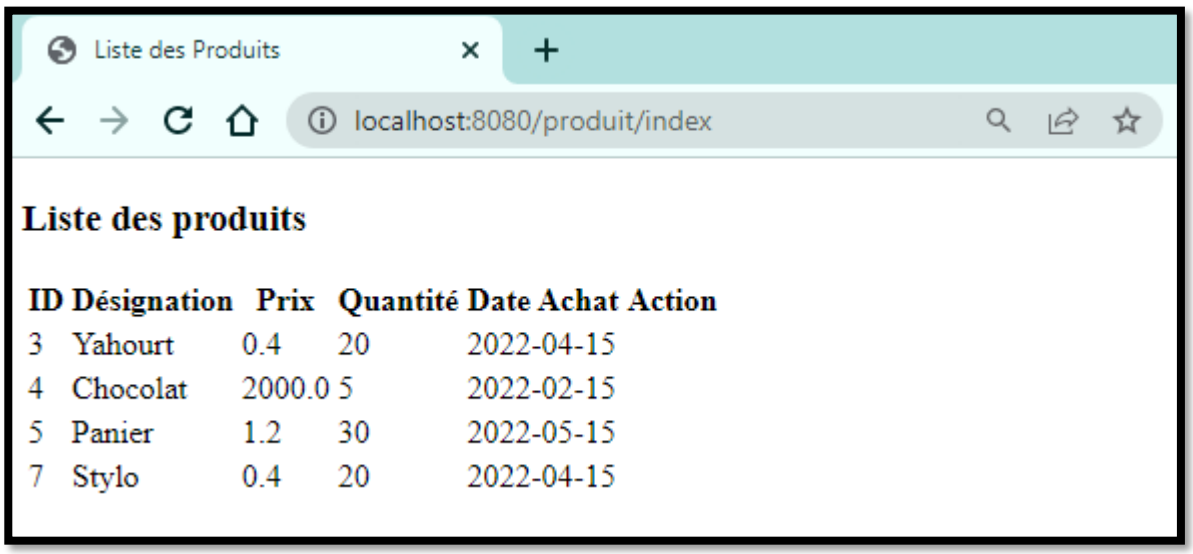
```java
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

import spring.jpa.model.Categorie;
import spring.jpa.model.Produit;
import spring.jpa.repository.CategorieRepository;
import spring.jpa.repository.ProduitRepository;

@SpringBootApplication
public class JpaSpringBootThymeleafApplication {

	// Déclarer une référence de l'interface " ProduitRepository"
	static ProduitRepository produitRepos ;

	// Déclarer une référence de l'interface "CategorieRepository"
	static CategorieRepository categorieRepos;


	public static void main(String[] args) {

		// référencer le contexte
		ApplicationContext contexte =
SpringApplication.run(JpaSpringBootThymeleafApplication.class, args);
// Récupérer une implémentation de l'interface "ProduitRepository" par injection de dépendance
		produitRepos =contexte.getBean(ProduitRepository.class);

// Récupérer une implémentation de l'interface "CategorieRepository" par injection de dépendance
		categorieRepos =contexte.getBean(CategorieRepository.class);


		// créer deux catégories;
		Categorie cat1 = new Categorie("AL", "Alimentaire");
		Categorie cat2 = new Categorie("PL", "Plastique");

		//Attacher les deux catégories à la BD (insertion)
		categorieRepos.save(cat1);
		categorieRepos.save(cat2);


		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
		java.util.Date date1 = null;
		java.util.Date date2 = null;
		java.util.Date date3 = null;


		try {
			date1 = sdf.parse("2022-04-15");
```

```
                date2 = sdf.parse("2022-02-15");
                date3 = sdf.parse("2022-05-15");
            } catch (ParseException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }

            // Insérer 3 produits
            Produit p1 =new Produit("Yahourt", 0.400, 20, date1 , cat1);
            Produit p2 =new Produit("Chocolat", 2000.0, 5, date2, cat1);
            Produit p3 =new Produit("Panier", 1.200, 30, date3, cat2);
            produitRepos.save(p1);
            produitRepos.save(p2);
            produitRepos.save(p3);

            // Insérer une nouvelle  catégorie  avec l'ajout d'un nouveu produit
             Produit p4 =new Produit("Stylo", 0.400, 20, date1 );
             Categorie cat3 = new Categorie("BR", "BUREAUTIQUE");
             p4.setCategorie(cat3);
             cat3.getProduits().add(p4);
             categorieRepos.save(cat3);

            //Afficher la liste des produits
             afficherTousLesProduits();
      }

      static void afficherTousLesProduits()
      {
            System.out.println("****************************************");
            // Lister l'ensemble des produits
            System.out.println("Afficher tous les produits...");
            System.out.println("****************************************");
            List<Produit> lp = produitRepos.findAll();
            for (Produit p : lp)
            {
                System.out.println(p);
            }
            System.out.println("****************************************");
      }
}
```

24. Relancer l'exécution du projet puis taper l'url suivante :

http://localhost:8080/produit/index

25. Vérifier l'exécution suivante :


Liste des Produits
localhost:8080/produit/index
Liste des produits
ID Désignation Prix Quantité Date Achat Action
3 Yahourt 0.4 20 2022-04-15
4 Chocolat 2000.0 5 2022-02-15
5 Panier 1.2 30 2022-05-15
7 Stylo 0.4 20 2022-04-15