

Cours Spring MVC

Thymeleaf

Plan

- 1. Notion JEE**
- 2. Introduction au Principe de l'inversion de Contrôle et Injection de Dépendance**
 - a. Couplage Fort
 - b. Couplage Faible
- 3. Maven**
- 4. Framework Spring**
 - a. Présentation
 - b. Quelques Projets Spring
 - c. Architecture
- 5. Spring MVC et Thymeleaf**
 - a. Présentation de Spring MVC
 - b. Architecture d'un Projet Spring
- 6. Projet Fil Rouge**
 - a. Expression de Besoin
 - b. Modélisation du Besoin
- 7. Réalisation du Projet Fil Rouge**
 - a. Création du Projet
 - i. Ajout des Dépendances
 - ii. Configuration de Spring DevTools
 - iii. Configurer Spring Boot & JPA
 - b. Architecture de notre Projet
 - i. Couche Entity

- ii. Couche Repository**
- iii. Couche Service**
- iv. Couche Controller**
- v. Vues**
 - 1. Intégration de Thymeleaf**
 - 2. Intégration du JavaScript**

Cours 1 :

Exercice Application : (15h40)

Réaliser une Application de Gestion des Approvisionnement avec le Menu suivant :

1. Lister les Clients
2. Lister les toutes Commandes
3. Lister les Commandes Par Type
 - a. Encours
 - b. Valider
4. Ajouter une commande
5. Quitter

NB :

- Un Client est caractérisé par son et son prenom
- Une commande est caractérisé par sa date et son montant et son état
- Un client peut avoir plusieurs commandes et une commande est faite par un client
- Les données sont Stockées dans une Liste

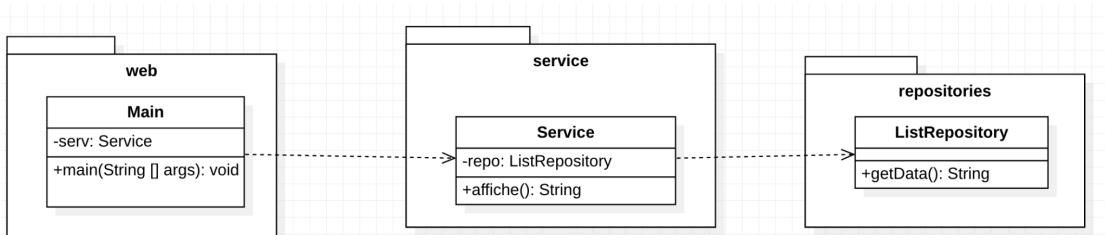
TAF : Réaliser le Projet en utilisant :

1. **Couplage Faible ,Inversion Contrôle et Injection de Dépendance**
2. **Design Pattern**
 - a. Fabrique
 - b. Singleton

1. Introduction au Principe de l'inversion de Contrôle et Injection de Dépendance

a. Couplage Fort

Exemple :



Application:

- **Packadge repository**

```
public class ListRepository {  
    public String getData(){  
        return "Les Donnees proviennent d'une Liste";  
    }  
}
```

- **Packadge service**

```

public class Service {
    //Couplage Fort
    ListRepository repo;
    public Service() {}
    //Injection de Dépendance Par Constructeur
    public Service(ListRepository repo) {
        this.repo = repo;
    }
    //Injection de Dépendance Par Setter
    public void setRepo(ListRepository repo) {
        this.repo = repo;
    }
    public String affiche(){
        return repo.getData();
    }
}

```

● Packadge view

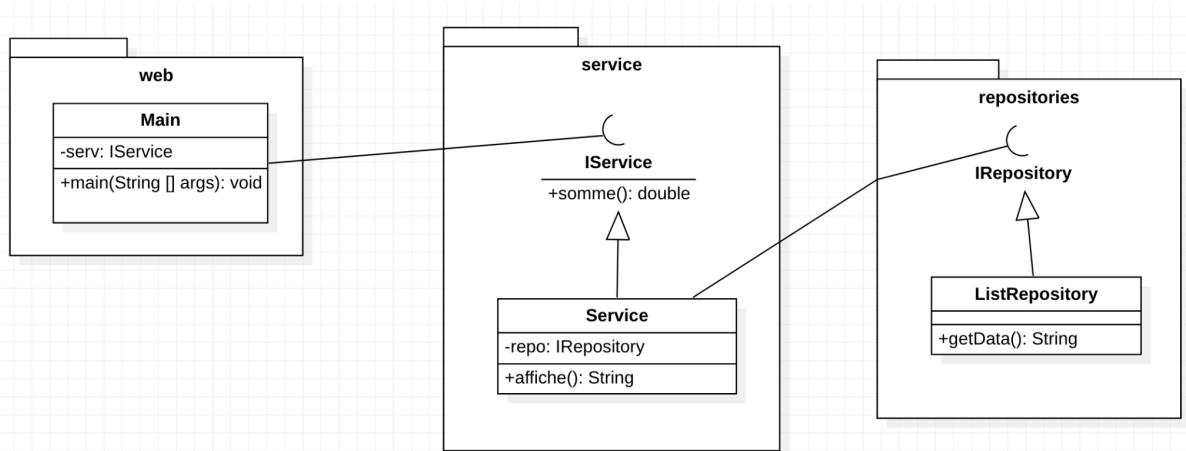
```

public class Main {
    public static void main(String[] args) {
        ListRepository repo=new ListRepository();
        Service service=new Service(repo);
        System.out.println(service.affiche());
    }
}

```

Inconvénients : Cette architecture est ouverte à la modification et peu évolutive .

b. Couplage Faible(IOC)



Avantages : Cette architecture est fermée à la modification et ouverte à l'évolution.

Application:

- Packadge repository
 - IRepository

```
public interface IRepository {  
    public String getData();  
}
```

- ListRepository

```
public class ListRepository implements IRepository{  
    @Override  
    public String getData(){  
        return "Les Donnees proviennent d'une Liste";  
    }  
}
```

- Packadge service
 - IService

```
public interface IService {  
    public String affiche();  
    public void setRepo(IRepository repo);  
}
```

- **Service**

```
public class Service implements IService{  
    //Couplage Faible  
    IRepository repo;  
    public Service() {}  
    //Injection de Dépendance Par Constructeur  
    public Service(IRepository repo) {  
        this.repo = repo;  
    }  
    //Injection de Dépendance Par Setter  
    @Override  
    public void setRepo(IRepository repo) {  
        this.repo = repo;  
    }  
    @Override  
    public String affiche() { return repo.getData(); }  
}
```

- **Packadge view**

```

    ]
    public static void main(String[] args) {
        IRepository repository=new ListRepository();
        IService service=new Service();
        service.setRepo(repository);
        System.out.println(service.affiche());
    }
}

```

2. Maven

a. Présentation de Maven

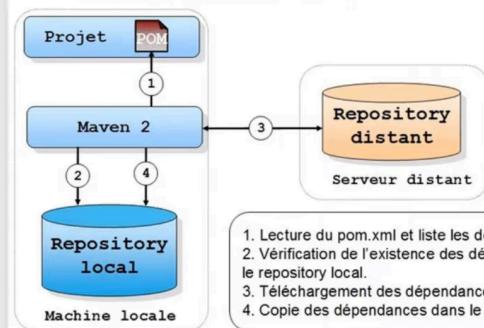
Maven : POM

Maven utilise un paradigme connu sous le nom de **Project Object Model (POM)** afin de :

- Décrire un projet logiciel,
- Ses dépendances avec des modules externes
- et l'ordre à suivre pour sa production.

Il est livré avec un grand nombre de tâches (**GOLS**) prédefinies, comme la compilation du code Java ou encore sa modularisation.

Gestion des dépendances par Maven 2



Framework

Spring

a. Présentation

Spring est ainsi un des frameworks les plus répandus dans le monde Java : sa popularité a grandi au profit de la **complexité de Java EE notamment pour ses versions antérieures à la version 5** mais aussi grâce à la qualité et la richesse des fonctionnalités qu'il propose :

- son cœur reposant sur un conteneur de type **IoC(Inversion de Contrôle et Injection de Dépendance)** assure la gestion du cycle de vie des beans et l'injection des dépendances
- l'utilisation de l'AOP(Programmation Orientée Aspect)
- des projets pour faciliter l'intégration avec de nombreux projets open source ou API de Java EE

Spring était un framework applicatif à ses débuts mais maintenant c'est un véritable écosystème composé du framework Spring, de projets qui couvrent de nombreux besoins et de middlewares.

Spring permet une grande flexibilité dans les fonctionnalités et les projets utilisés dans une application.

Spring est associé à la notion de conteneur léger (*lightweight container*) par opposition aux conteneurs lourds que sont les serveurs d'applications Java EE.

Les Avantages de Spring :

- Interagir avec une base de données.

- **Traiter des requêtes HTTP et écrire des réponses HTTP.**
- **Exécuter des traitements par lots (batch).**
- **Préparer un API REST**
- **Gérer la sécurité de l'application.**

Inversion de Contrôle avec Spring

1. Définition du Projet

IoC (Inversion of control), est un processus qui définit les dépendances d'un objet sans avoir à les créer. C'est lors de la création des objets, que Spring va injecter les Beans entre eux afin d'avoir toutes leurs dépendances .

2. Présentation des Dépendances

- a. **Spring Core** : Ce composant est la base de l'écosystème Spring. Il contient le “core container” (ce qui permet l'injection de dépendances).
- b. **Spring Beans** : Spring utilise un système de Bean. Un Bean est un objet qui est instancié, assemblé et géré par Spring IOC Container.
- c. **Spring Context** : La dépendance **Spring Context** définit un format XML qui permet de déclarer un ensemble de beans, c'est-à-dire un ensemble d'objets à créer pour l'application.

3. Réalisation de l'inversion de Contrôle avec Fichier XML

a. Etape 1: Création d'un projet Maven

b. Etape 2: Ajout des dépendances dans pom.xml

```
<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-core</artifactId>

    <version>5.3.13</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-context</artifactId>

    <version>5.3.13</version>

</dependency>

<dependency>

    <groupId>org.springframework</groupId>

    <artifactId>spring-beans</artifactId>

    <version>5.3.13</version>

</dependency>
```

c. Etape 3

Création d'un fichier de configuration spring dans :

ressources>ApplicationContext.xml

```

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation=" http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- bean definitions here -->

    <bean id="repoCommande" class="org.example.repositories.CommandeRepository"></bean>

    <bean id="repoClient" class="org.example.repositories.ClientRepository"></bean>

    <bean id="service" class="org.example.service.Service">

        <property name="repo" ref="repoClient"></property>

        <constructor-arg ref="repoCommande" ></constructor-arg>

    </bean>

</beans>

```

Description

Dans le Noeud Bean :

- **Définition :** Un Bean est un objet qui est instancié, assemblé et géré par Spring IOC Container.
- **id :** correspond à l' identifiant du bean qui permettra de l'identifier lors d'une injection.
- **class :** correspond à la classe d'implémentation a injecter.
- **property :** est un noeud enfant de bean, il permet de faire l'injection de dépendance a partir du setter ,ses ses attributs sont
 - **name:**La méthode setter appelée pour faire l'injection
 - **ref:**identifiant de la dépendance à injecter

- **constructor-arg**: est un noeud enfant de bean, il permet de faire l'injection de dépendance à partir du constructeur , ses attributs sont :
 - **ref**: identifiant de la dépendance à injecter

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/
<bean id="repo" class="repositories.ListRepository"></bean>
<bean id="repo1" class="repositories.BdRepository"></bean>
<bean id="service" class="service.Service">
    <constructor-arg ref="repo1"></constructor-arg>
</bean>
</beans>
```

- **Etape 2: Dans la vue**

```
public static void main( String[] args ){
    ApplicationContext context=
        new ClassPathXmlApplicationContext("ApplicationContext.xml");
    IService service=( IService ) context.getBean("service");
}
```

4. Réalisation de l'inversion de Contrôle avec Annotation

- **Étape 1:** Définir les composants (classe) que le conteneur de spring doit créer au démarrage. On utilisera l'annotation **@Component("nomBean")**.

```
@Component("repoClientList")
public class ClientRepository implements IClientRepository{
    4 usages
    private List<Client> clients=new ArrayList();
```

```
@Component("repoClientBd")
public class ClientRepositoryBD implements IClientRepository{
    no usages
    public Client findByTelephone(String tel) { return null; }
```

```
@Component()
public class Service implements IService{
```

- **Étape 2: Injection de Dépendance**
 - **Par constructeur**

```
2 usages
private ICommandeRepository commandeRepository;
no usages
public Service(ICommandeRepository commandeRepository) {
    this.commandeRepository = commandeRepository;
}
```

- **Par @Autowired**

```
@Autowired  
@Qualifier("repoClientBd")  
private IClientRepository repo;
```

NB : Lorsqu'on a plusieurs implementation alors l'injection de dépendance se fera comme suit :

- 1. Identifier chaque implémentation avec l'annotation `@Component("nom")`**
 - 2. Lors de l'injection avec `@Autowired` ajouter l'annotation `@Qualifier("identifiant")`**
- Etape 3: Dans la vue**

```
ApplicationContext context=new  
AnnotationConfigApplicationContext("org.example.repositories",  
"org.example.service");  
  
IService service=( IService ) context.getBean("");
```

Notion ORM en JAVA

1. Notion de JPA

Introduction

- Travailler dans les deux univers que sont l'orienté objet et la base de données relationnelle peut être lourd et consommateur en temps dans le monde de l'entreprise d'aujourd'hui.
- EclipseLink est un outil de mapping objet/relationnel pour le monde Java.
- Le terme mapping objet/relationnel (ORM) décrit la technique consistant à faire le lien entre la représentation objet des données et sa représentation relationnelle basée sur un schéma SQL.

Application Orientée Objet

```
larticle:List
```

```
id:1  
reference:xxxx  
libelle:lait  
qtestock:10  
pu:1500  
categorie:{id=1,  
libelle :Alimentaire}  
  
id:2  
reference:xxxy  
libelle:sucre  
qtestock:10  
pu:2000  
categorie:{id=1,  
categorie:Alimentaire}  
  
id:23  
reference:xxxx  
libelle:Lait de corps  
qtestock:20  
pu:3000  
categorie:{id=2,  
categorie:Cosmétique}
```

```
public List<Article> findAll() {  
    List<Article> larticle=null;  
    DaoMysql dao=DaoMysql.getInstance();  
    dao.initPS(SQL_SELECT_ALL);  
    ResultSet rs= dao.executeSelect();  
    try {  
        larticle=new ArrayList<>();  
        while(rs.next()){  
            Article art =new Article();  
            art.setId(rs.getInt(1));  
            art.setLibelle(rs.getString(3));  
            art.setReference(rs.getString(2));  
            art.setQtestock(rs.getDouble(4));  
            art.setPu(rs.getDouble(5));  
  
            art.setCategorie(Fabrique.getInsCategorieCont  
roller().findById(rs.getInt(6)));  
            larticle.add(art);  
        }  
        rs.close();  
        dao.CloseConnection();  
    } catch (SQLException ex) {  
        Logger.getLogger(CategorieController.class.get  
Name()).log(Level.SEVERE, null, ex);  
    }  
    return larticle;  
}
```

```
public class Article {  
    private int id;  
    private String reference;  
    private String libelle;  
    private double qtestock;  
}
```

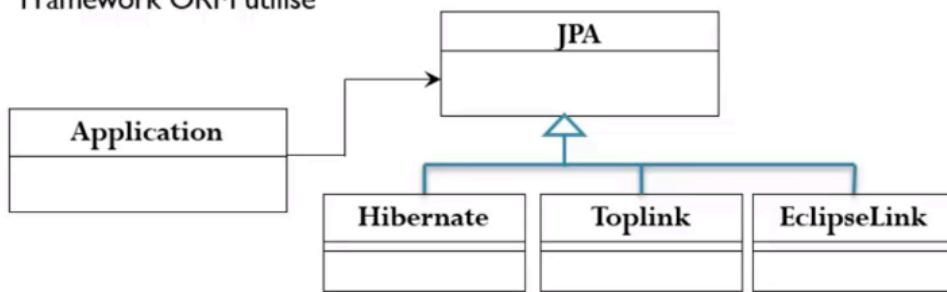
ORM:Mapping Object Relationnel

SGBDR:Table article

id	reference	libelle	pu	qtestock	id_categorie
21	Test	test	56	45	1
3	xxf	Lait de Corps	1500	50	1
4	rrr	rrr	12	12	1

JPA : Java Persistence API

- JPA est une spécification créée par Sun pour standardiser le mapping Objet relationnel.
- JPA définit un ensemble d'interfaces, de classes abstraites et d'annotations qui permettent la description du mapping objet relationnel
- Il existe plusieurs implémentation de JPA:
 - Hibernate
 - Toplink
 - iBatis
 - EclipseLink
- L'utilisation de JPA permet à votre application d'être indépendante du Framework ORM utilisé



NB: Dans notre cours nous utiliserons implémentation avec
Hibernate

Remarque :

Hibernate

- framework ORM (le premier) pour Java implémentant les spécifications de l'API JPA
- Open-source
- Créé par **JBoss** (Entreprise productrice de serveurs d'application JEE JBoss)
- Possédant une extension **NHibernate** pour la plateforme .NET (de **Microsoft**)
- Pouvant être utilisé dans tout type de projet Java (Web, Client lourd...)

2. Notion Entity

a. Définition

JPA permet de définir des entités (entities). Une entité est simplement une instance d'une classe qui sera persistante (que l'on pourra sauvegarder dans / charger depuis une base de données relationnelle).

Une entité est signalée par l'annotation **@Entity** sur la classe. De plus, une entité JPA doit disposer d'un ou plusieurs attributs définissant un identifiant grâce à l'annotation **@Id**. Cet identifiant correspond à la clé primaire dans la table associée.

b. Quelques Annotations

Quelques annotations JPA de Mapping des Entités

- **@Table**
 - Préciser le nom de la table concernée par le mapping. Par défaut c'est le nom de la classe qui sera considérée
- **@Column**
 - Associer un champ de la colonne à la propriété. Par défaut c'est le nom de la propriété qui sera considérée.
- **@Id**
 - Associer un champ de la table à la propriété en tant que clé primaire
- **@GeneratedValue**
 - Demander la génération automatique de la clé primaire au besoin
- **@Basic**
 - Représenter la forme de mapping la plus simple. Cette annotation est utilisée par défaut
- **@Transient**
 - Demander de ne pas tenir compte du champ lors du mapping
- **@OneToMany, @ManyToOne**
 - Pour décrire une association de type un à plusieurs et plusieurs à un
- **@JoinedColumn**
 - Pour décrire une clé étrangère dans une table
- **@ManyToMany**
 - Pour décrire une association plusieurs à plusieurs

3. Application

a. Dépendances dans POM XML

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.25</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.1.Final</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>2.6.0</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version> 5.3.22</version>
</dependency>
```

b. Création du Fichier de persistance dans :

resources>META-INF>persistence.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence
    xmlns="https://jakarta.ee/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
    https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
        version="3.0">
    <persistence-unit name="ismJpa">
        <properties>
            <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:8889/db_test_orm?useSSL=false" />
            <property name="javax.persistence.jdbc.user"
value="root" />
            <property
name="javax.persistence.jdbc.password" value="root" />
```

```

<property
name="javax.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver" />
    <property
name="javax.persistence.schema-generation.database.action"
value="drop" />
        <property name="hibernate.show_sql"
value="true" />
            <property name="hibernate.format_sql"
value="true" />
        </properties>
    </persistence-unit>

</persistence>

```

NB : Les valeurs de `javax.persistence.schema-generation.database.action` sont:

- `create`
- `drop`
- `drop-and-create`
- `update`

c. Configuration de EntityManager

Spring fournit un support ORM (JPA, Hibernate, etc.) dans son module `spring-orm`. Spring `LocalEntityManagerFactoryBean` peut être utilisé pour enregistrer JPA `EntityManagerFactory` en tant que bean. Cette méthode de chargement de JPA est conforme au contrat d'amorçage autonome standard de JPA.

config>PersistenceConfig

```

@Configuration
@EnableJpaRepositories(basePackages =
{"org.example.repositories"})
public class PersistenceConfig {
    @Bean
    public LocalEntityManagerFactoryBean entityManagerFactory() {
        LocalEntityManagerFactoryBean factoryBean = new
LocalEntityManagerFactoryBean();
        factoryBean.setPersistenceUnitName("ismJpa");
        return factoryBean;
    }

    @Bean
    public JpaTransactionManager
transactionManager(EntityManagerFactory entityManagerFactory) {

```

```

        JpaTransactionManager transactionManager = new
JpaTransactionManager();

transactionManager.setEntityManagerFactory(entityManagerFactory);
    return transactionManager;
}

}

```

d. Injection de Dépendance

```

AnnotationConfigApplicationContext appContext = new
AnnotationConfigApplicationContext();
appContext.scan("org.example");
appContext.refresh();
IService service=( IService ) appContext.getBean("service");

```

e. Crédit des Entités

```

@Entity
public class Client {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long Id;
}

```

4. Notion de Repository

c. Définition

Spring Data s'organise autour de la notion de repository. Il fournit une interface marqueur générique **Repository<T, ID>**. Le type T correspond au type de l'objet géré par le repository. Le type ID correspond au type de l'objet qui représente la clé d'un objet.

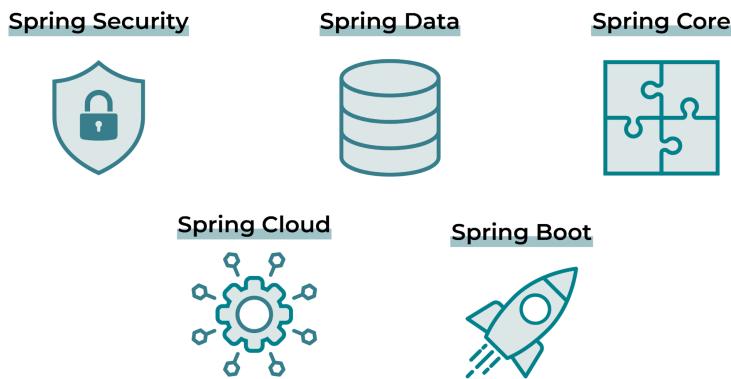
L'interface **CrudRepository<T, ID>** hérite de **Repository<T, ID>** et fournit un ensemble d'opérations élémentaires pour la manipulation des objets.

Pour une intégration de Spring Data avec JPA, il existe également l'interface **JpaRepository<T, ID>** qui hérite indirectement de **CrudRepository<T, ID>** et qui fournit un ensemble de méthodes pour interagir avec une base de données.

Pour créer un repository, il suffit de créer une interface qui hérite d'une des interfaces ci-dessus.

Quelques Composants Spring

Spring est un écosystème formé de plusieurs composants, les plus importants sont :



- **Spring Core**

Ce composant est la base de l'écosystème Spring. Il contient le “core container” (ce qui permet l'injection de dépendances vue précédemment), mais il contient également Spring MVC qui permet de faire du web et de Data Access qui fournit des éléments fondamentaux pour la communication avec les bases de données.

- **Spring Data**

Ce composant permet de communiquer avec de nombreux types de bases de données. Par exemple, il offre la capacité de communiquer avec une base de données en implémentant uniquement des interfaces grâce à des conventions de nommage .

- **Spring Security**

Ce composant permet de gérer l'authentification et l'autorisation mais aussi la sécurité des API. Il est l'un des composants les plus critiques de Spring Framework, bien qu'il soit aussi l'un des plus complexes.

- **Spring Cloud**

Ce composant permet d'implémenter l'architecture microservice et de répondre toutes aux contraintes de cette architecture logicielle .

• **Spring Boot**

C'est un composant très particulier de Spring Framework, dans la mesure où il nous permet de mettre en œuvre tous les autres. Ce cours vous montrera comment tirer profit de la puissance de Spring Boot, et de ses avantages qui sont :

- l'autoconfiguration automatique de Spring ;
- des starters de dépendances ;
- des endpoints Actuator pour fournir des données sur l'application.

Spring Boot

• **Spring Boot**

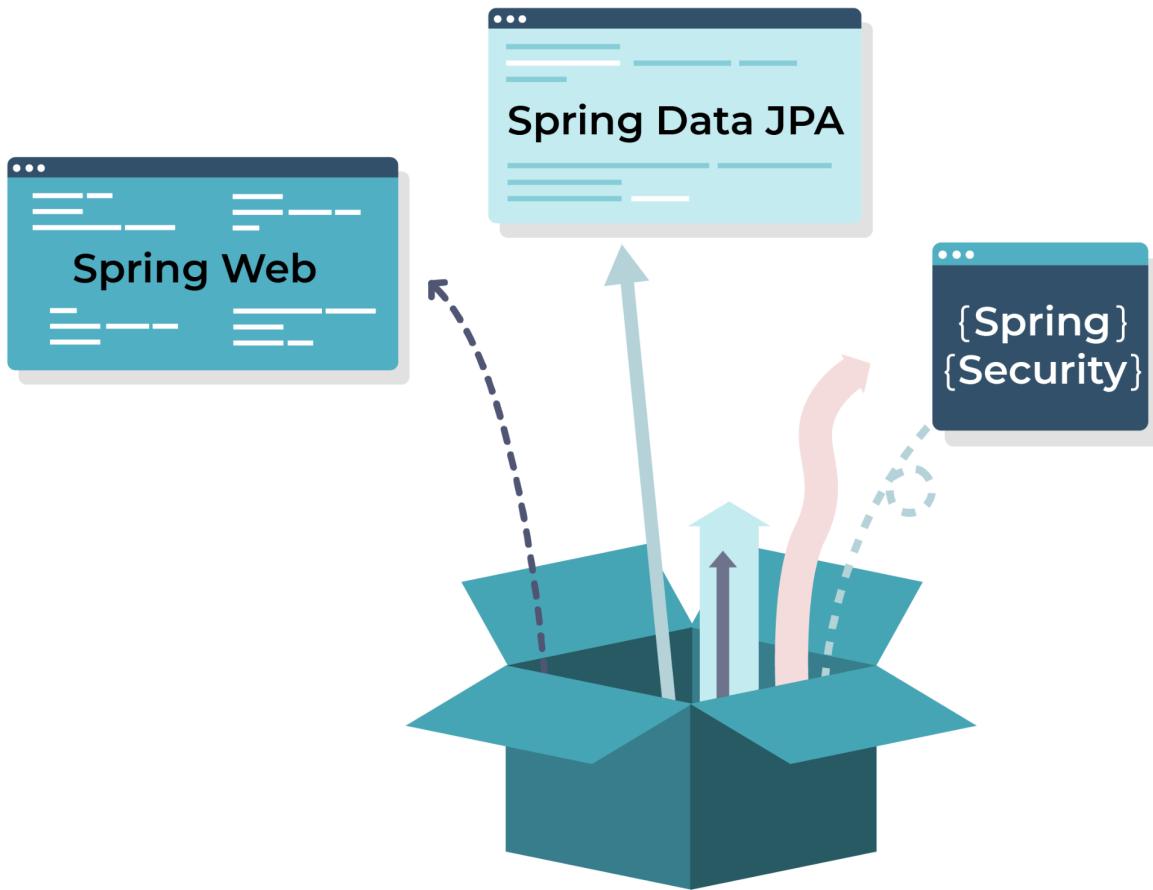
C'est un composant très particulier de Spring Framework, dans la mesure où il nous permet de mettre en œuvre tous les autres. Ce cours vous montrera comment tirer profit de la puissance de Spring Boot, et de ses avantages qui sont :

- l'autoconfiguration automatique de Spring ;
- des starters de dépendances ;
- des endpoints Actuator pour fournir des données sur l'application.

Les Avantages de Spring Boot

Avantage n° 1 : optimisation de la gestion des dépendances

Spring Boot nous fournit des starters, qui correspondent à un ensemble de dépendances homogénéisées (associations, versions). On peut les comparer à des kits de dépendances.



NB : Nul besoin de définir les versions des dépendances explicitement dans le pom.xml : Maven les déduit grâce à la version de Spring Boot utilisée.

Avantage n° 2 : l'autoconfiguration

C'est peut-être l'avantage le plus important de Spring Boot ! L'utilisation de Spring Boot, et l'annotation `@SpringBootApplication` placée au niveau de la classe principale de notre projet (celle générée automatiquement), déclenchent automatiquement de nombreuses opérations en background qui nous sont nécessaires.

Le développeur peut alors **se concentrer sur le code métier** au lieu de passer un temps fou à configurer le framework qu'il utilise.

Avantage n° 3 : la gestion des propriétés

Spring Boot permet de gérer les propriétés au sein d'un programme en toute simplicité.

Par exemple, les informations de connexion et de configuration de Spring seront saisies dans un fichier **properties** et qui seront prises en compte par

certaines classes, sans que nous ayons besoin d'agir. Ce fichier est l'un des éléments clés pour la gestion des propriétés de notre Application.

Avantage n° 4 : le monitoring et la gestion du programme

Spring Boot qui permet de monitorer et de manager notre programme pendant qu'il est en cours d'exécution.

Avantage n° 5 : le déploiement

Spring Boot produit un simple fichier JAR pour le déploiement.

• Projet Fil Rouge

Cas Auchan

Soit une gestion commerciale , un client a la possibilité d'effectuer des commandes de produits.Après chaque commande une facture est éditée et cette facture peut faire l'objet de plusieurs paiements.

On voudrait connaître :

- Les produits d'une commandes
- La facture d'une commande
- Les factures payées ainsi que les détails d'un paiement c'est-à- dire les factures de ce paiement.
- Le vendeur qui a reçu le paiement

• Création d'un projet Spring Boot

a. Créer un Projet Spring Initializr

b. Ajouter les Stater Suivants:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</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-data-jpa</artifactId>
</dependency>
```

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

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

    <artifactId>spring-boot-starter-thymeleaf</artifactId>
>
</dependency>
```

NB :

Le module **spring-boot-devtools** inclut un serveur LiveReload intégré qui est utilisé pour déclencher une actualisation du navigateur lorsqu'une ressource est modifiée.

Pour que cela se produise dans le navigateur, nous devons installer le plugin LiveReload, une de ces implémentations est **Remote Live Reload** pour Chrome.

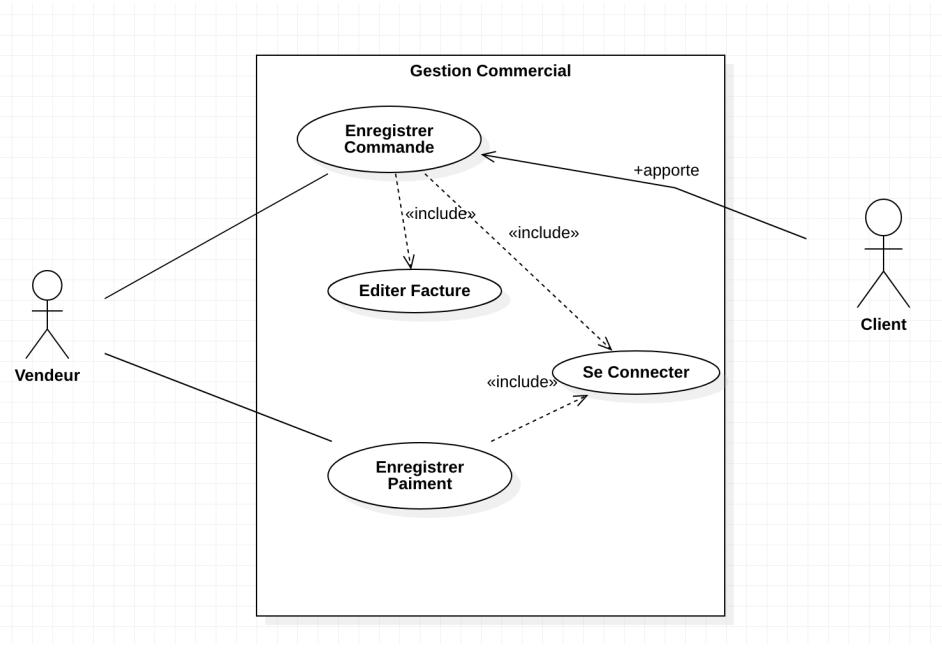
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
```

• Application

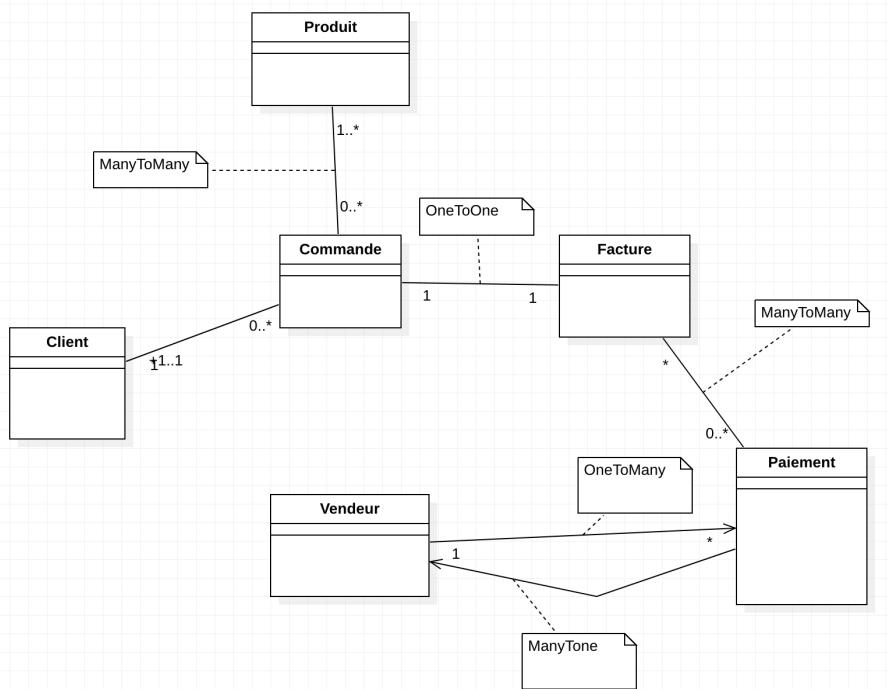
Réalisation de la couche entity et Repository du Projet Fil Rouge

a. Modélisation du Projet Fil Rouge

i. Diagramme de Use Case



ii. Diagramme de Classe



b. Crédation de la Couche Entité

La couche entité est créée à partir du diagramme de classe

c. Crédation de la Base de Donnée

La base de donnée est générée à partir de la couche entité

#package src/main/ressources/**application.properties**

```
spring.datasource.url=jdbc:mysql://server:port/db_name  
spring.datasource.username=root  
spring.datasource.password=  
#com.mysql.cj.jdbc.Driver  
spring.datasource.driver-class-name =com.mysql.jdbc.Driver  
spring.jpa.database=MYSQL  
spring.jpa.show-sql=false  
#create,update,create-drop  
spring.jpa.hibernate.ddl-auto=create  
spring.jpa.properties.dialect=org.hibernate.dialect.MySQL57Dialect
```

NB : Pour que spring puisse créer la BD

```
spring.datasource.url=jdbc:mysql://server:port/db-name?autoReconnect=true&useSSL=false&createDatabaseIfNotExist=true
```

d. Les Repositories

```
4 usages  
public interface ClientRepository extends JpaRepository<Client, Long> {  
    1 usage
```

e. Insérer des Fausses données

```
@SpringBootApplication
public class IsmApplication implements CommandLineRunner {
    no usages
    public static void main(String[] args) { SpringApplication.run(IsmApplication.class, args); }
    public void run(String... args) throws Exception {
    }
}
```

f. Requêtes Personnalisées

L'interface **JpaRepository<T, ID>** déclare beaucoup de méthodes mais elles suffisent rarement pour implémenter les fonctionnalités attendues d'une application.

i. Query Methods

Spring Data JPA utilise une convention de nommage pour générer automatiquement le code sous-jacent et exécuter la requête. La requête est déduite de la signature de la méthode (**on parle de query methods**).

La convention est la suivante : Spring Data JPA supprime du début de la méthode les préfixes **find**, **findAll**, **read**, **query**, **count** et **get** et recherche la présence du mot **By** pour marquer le début des critères de filtre.

Le terme après **By** fait référence à un attribut de l'entité JPA pour lequel on veut appliquer un filtre. Chaque critère doit correspondre à un paramètre de la méthode en respectant l'ordre.

Exemple:

```
Optional<Client> findByTelephone(String telephone);  
1 usage  
List<Client> findByNomCompletLike(String telephone);
```

```
public interface CommandeRepository extends JpaRepository<Commande, Long> {  
    no usages  
    public Optional<Commande> findByNumero(String numero);  
    no usages  
    public List<Commande> findByDateCmdBetween(Date debut, Date fin);  
    no usages  
    public List<Commande> findByNumeroOrMontant(String numero, double Montant);  
    no usages  
    • public Long countByDateCmd(Date date);
```

ii. Utilisation de @Query

L'annotation **@Query** permet de préciser la requête directement sur la méthode elle-même .

TAF 1 : Définir la requêtes suivantes dans CommandeRepository :

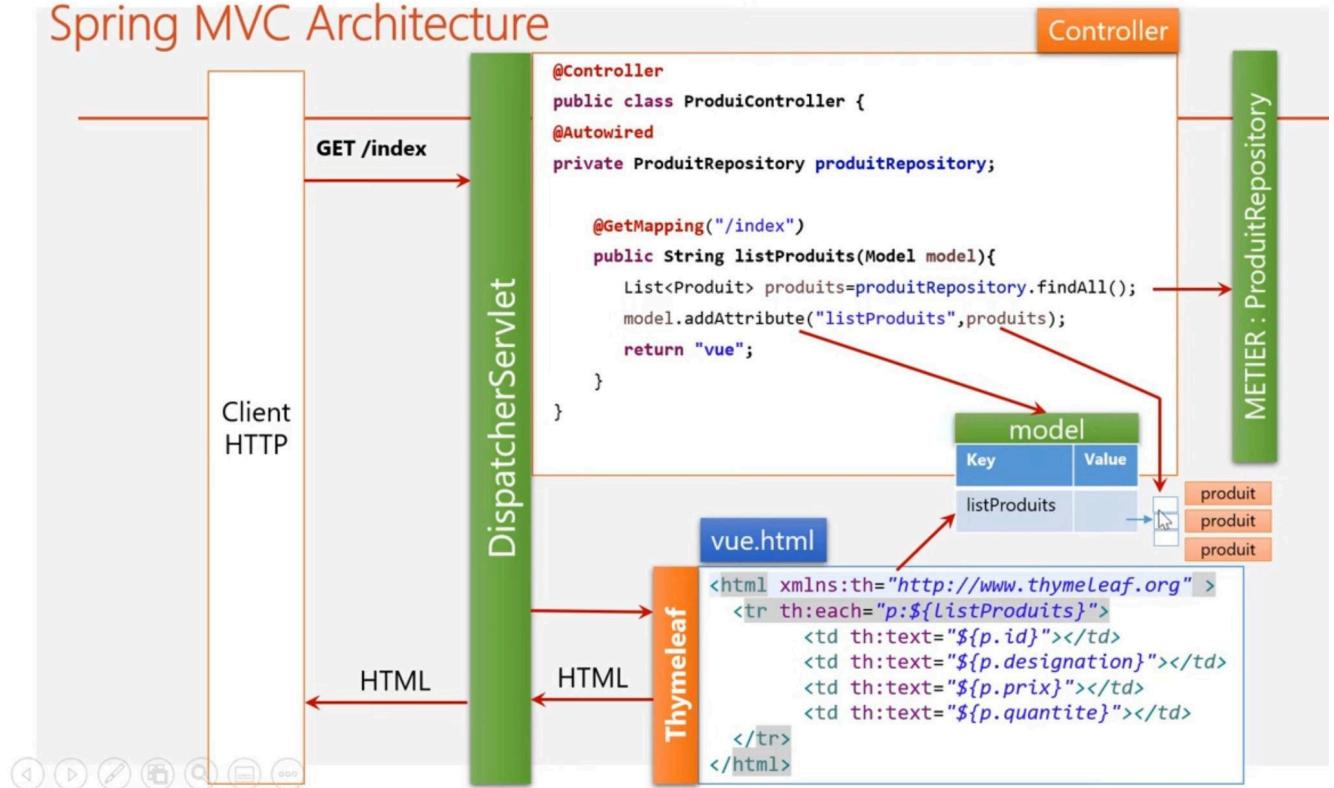
- Afficher les commandes dont le montant est compris entre 100000 et 150000

```
no usages  
@Query(name = "select c from Commande c where c.montant>=:min and c.montant<=:mx")  
public List<Commande> rechercherCmdeParMontant(@Param("min") double montant1,  
                                              @Param("max") double montant2);
```

NB : Remarquez la présence de l'annotation **@Param** qui permet d'associer le paramètre de la méthode au paramètre de la requête nommée.

g. Architecture MVC

Spring MVC Architecture



h. DispatcherServlet

Tout d'abord, la requête est reçue par le **DispatcherServlet**, qui est chargé de traiter toutes les requêtes URI entrantes et de les mapper à leurs gestionnaires correspondants sous la forme de méthodes de contrôleur. Une fois la méthode du contrôleur exécutée, la ressource est ensuite traitée comme une réponse qui peut être Html ou JSON ou XML.

i. Controller

L'annotation Spring **@Controller** est également **une spécialisation** de l'annotation **@Component**.

L'annotation **@Controller** indique qu'une classe particulière joue le rôle d'un contrôleur c'est dire reçoit une **Request HTTP** et produit une **Réponse qui est généralement une vue**.

L'annotation Spring Controller est généralement utilisée en combinaison avec des méthodes de gestionnaire annotées basées sur l' annotation

1. **@RequestMapping** :est utilisé pour mapper les requêtes HTTP (**GET ou POST**) aux méthodes du contrôleur.
2. **@GetMapping** est un raccourci pour **@RequestMapping(method = RequestMethod.GET)** et est

utilisé pour mapper `GET` les requêtes HTTP aux méthodes de contrôleur mappées.

3. `@PostMapping` est un raccourci pour `@RequestMapping(method = RequestMethod.POST)` et est utilisé pour mapper `POST` les requêtes HTTP aux méthodes de contrôleur mappées.

NB : Avec Spring MVC, une méthode mappée retourne un String qui correspond au nom de la vue à retourner.

j. Model

Spring Framework fournit une interface appelée **Model(I)** pour travailler avec les données.

`Il est utilisé pour transférer des données entre la vue et le contrôleur de l'application Spring MVC.`

L'interface du modèle est disponible dans le package `org.springframework.ui`.

Il agit comme un conteneur de données qui contient les données de l'application.

Ces données stockées peuvent être de n'importe quelle forme, comme une chaîne, un objet, des données de la base de données, etc.

k. Vues

Spring Framework permet réaliser les vues avec `jsp` ou en utilisant un moteur de templates .Dans ce cours nous utiliserons le moteur de templates **Thymeleaf** .

NB:

Par défaut, les modèles **Thymeleaf** doivent être placés dans le répertoire `src/main/resources/templates` de votre projet. Les fichiers statiques (non transformés par **Thymeleaf**) doivent être placés dans le répertoire `src/main/resources/static` de votre projet.

i. Thymeleaf

i. Présentation

Thymeleaf est un moteur de modèle Java côté serveur moderne pour les environnements Web et autonomes, capable de traiter HTML, XML, JavaScript, CSS et même du texte brut.

L'objectif principal de Thymeleaf est de fournir un moyen élégant et hautement maintenable pour créer des vues.

Pour ce faire, il s'appuie sur le concept de modèles naturels pour injecter sa logique dans les fichiers de vue d'une manière qui n'empêche pas le modèle d'être utilisé comme prototype de conception.

Thymeleaf a également été conçu dès le départ avec les normes Web à l'esprit

- en particulier HTML5
- vous permettant de créer des modèles de validation complets si vous en avez besoin.

ii. Utilisation

Ajoutons un `xmlns:th` attribut à notre balise `<html>`, pour utiliser les expressions de Thymeleaf :

```
<html xmlns:th="http://www.thymeleaf.org">
```

- **`th:text`** : afficher du texte dans une balise
- **`th:class`** : ajouter une classe dans une balise
- **`th:each="value,index:${tableau}"`** : parcourir un tableau
- **`th:href="@{path(queryParams=${variable})}"`**
- **`th:if="${expression}"`**

- Expressions simples :
 - Expressions variables :\${...}
 - Expressions variables de sélection :*{...}
 - Expressions des messages :#{...}
 - Expressions d'URL de lien :@{...}
 - Expressions fragmentées :~{...}
- Littéraux
 - Textes littéraux : 'one text', 'Another one!', ...
 - Littéraux numériques : 0, 34, 3.0, 12.3, ...

- **Littéraux booléens** : true, false
 - **Littéral nul** : null
- **Opérations de texte :**
 - **Concaténation de chaînes** : +
 - **Substitutions littérales** : |The name is \${name}|
- **Opérations arithmétiques:**
 - **Opérateurs binaires** : +, -, *, /, %
 - **Signe moins (opérateur unaire)** : -
- **Opérations booléennes :**
 - **Opérateurs binaires** : and, or
 - **Négation booléenne (opérateur unaire)** : !, not
- **Comparaisons et égalité :**
 - **Comparateurs** : >, <, >=, <=(gt, lt, ge, le)
 - **Opérateurs d'égalité** : ==, !=(eq, ne)
- **Opérateurs conditionnels :**
 - **Si donc**: (if) ? (then)
 - **Si-alors-sinon** : (if) ? (then) : (else)
 - **Défaut**: (value) ?: (defaultvalue)

m. Ajouter Bootstrap

i. Ajouter la Dépendance

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>5.2.3</version>
</dependency>
```

ii. AJouter le Link

```
href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css"
```

n. Notion de Dto

o. Design Pattern Builder

• Réalisation du Projet Fil Rouge

a. Lister Clients

i. Création des entités

```
@Entity
@Table(name="users")
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "role")
@DiscriminatorValue(value="User")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public abstract class User {
    @Id
    @GeneratedValue(strategy =
GenerationType.AUTO)
    protected Long id;
    protected String nomComplet;
    public User(String nomComplet) {
        this.nomComplet = nomComplet;
    }
}
```

```
@Embeddable
@Data
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class Adresse {
```

```
private String ville;
private String quartier;
private String numeroVilla;

@Override
public String toString() {
    return
        ville + " | " +
        quartier + " | " +
        numeroVilla
    ;
}

}
```

```
public enum Etat {
    Encours, Terminer, Payer, Livrer
}
```

```
@Entity
@Table(name = "clients")
@DiscriminatorValue(value="Client")
@Data
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class Client extends User {
    @Column(nullable = false)
    private String telephone;
    @Embedded()
    private Adresse adresse;
```

```
@OneToOne(mappedBy = "client", fetch =
FetchType.LAZY)
List<Commande> commandes;

    public Client(String nomComplet, String
telephone) {
        super(nomComplet);
        this.telephone = telephone;
    }

    public Client(String nomComplet, String
telephone, Adresse adresse) {
        super(nomComplet);
        this.telephone = telephone;
        this.adresse = adresse;
    }

@Override
public String toString() {
    return "Client{" +
        "telephone='" + telephone + '\'' +
        ", adresse=" + adresse +
        ", id=" + id +
        ", nomComplet='" + nomComplet +
        '\'' +
        '}';
}
}
```

ii. Créer les Repository

```
public interface ClientRepository extends  
JpaRepository<Client, Long> {  
Optional<Client> findByTelephone(String telephone);  
List<Client> findByNomCompletLike(String telephone);  
}
```

iii. Remplir la Base

```
@Autowired  
ClientRepository clientRepository;  
@Override  
public void run(String... args) throws Exception {  
  
for (int i = 0; i < 10 ; i++) {  
if(i%2==0){  
clientRepository.save(new Client("Diop"+i,"77867101"+i));  
}else{  
clientRepository.save(new Client("Ndiaye"+i,"77867101"+i,  
new Adresse("Dakar","Point E","Villa00"+i)));  
}  
}  
}
```

iv. Créer le Contrôleur

```
@Controller  
public class ClientController {  
@Autowired  
ClientRepository clientRepository;  
@GetMapping("/liste-clients")  
public String listerClient(Model model){  
List<Client> clients=  
clientRepository.findAll();  
  
//clients.forEach(System.out::println);  
  
model.addAttribute("clients",clients);  
return "client";
```

```
    }
}
```

V. Vue

```
<!DOCTYPE html >
<html lang="en"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet"
          href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
</head>
<body>
<div class="container mt-2">
    <nav class="navbar navbar-expand-lg navbar-light bg-primary
mb-3 " >
        <div class="container-fluid">
            <a class="navbar-brand text-white" href="#">Navbar</a>
            <button class="navbar-toggler" type="button"
data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                    <li class="nav-item">
                        <a class="nav-link active text-white"
aria-current="page" href="#">Home</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link text-white"
href="#">Link</a>
                    </li>
                    <li class="nav-item dropdown">
                        <a class="nav-link dropdown-toggle"
href="#" id="navbarDropdown" role="button"
data-bs-toggle="dropdown" aria-expanded="false">
```

```
        Dropdown
    </a>
    <ul class="dropdown-menu"
aria-labelledby="navbarDropdown">
        <li><a class="dropdown-item"
href="#">Action</a></li>
        <li><a class="dropdown-item"
href="#">Another action</a></li>
        <li><hr
class="dropdown-divider"></li>
        <li><a class="dropdown-item"
href="#">Something else here</a></li>
        </ul>
    </li>
    <li class="nav-item">
        <a class="nav-link disabled" href="#">Disabled</a>
    </li>
</ul>
<form class="d-flex">
    <input class="form-control me-2"
type="search" placeholder="Search" aria-label="Search">
    <button class="btn btn-outline-success"
type="submit">Search</button>
</form>
</div>
</div>
</nav>
<div class="card mt-1">
    <div class="card-header">
        <h3>Liste des Clients</h3>
    </div>
    <div class="card-body">
        <table class="table table-bordered">
            <thead>
                <th>Nom et Prenom</th>
                <th>Telephone</th>
                <th>Adresse</th>
                <th>Commandes</th>
            </thead>
            <tbody>
                <tr th:each="cl:${clients}">
                    <td th:text="${cl.nomComplet}"></td>
                    <td th:text="${cl.telephone}"></td>
```

```

        <td th:text="${cl.adresse}"></td>
        <td ><a href="#" class="btn
btn-outline-success">Mes Commandes</a></td>
    </tr>
</tbody>
</table>
</div>

</div>
<!-- Content here -->
</div>
</body>
</html>
```

b. Paginer la Liste des Clients

i. Dans le Controller

```

@GetMapping("/liste-clients")
public String listerClient(Model model,
    @RequestParam(name = "page",defaultValue = "0") int page,
    @RequestParam(name = "size",defaultValue = "5") int size) {
    Page<Client> pageClients= clientRepository.findAll(
        PageRequest.of(page,size)
    );

    //clients.forEach(System.out::println);

    model.addAttribute("clients",pageClients.getContent());
    model.addAttribute("pages",new
    int[pageClients.getTotalPages()]);
    model.addAttribute("currentPage",page);

    return "client";
}
```

ii. Dans la Vue

```
<nav aria-label="Page navigation example mt-3 ">
  <ul class="pagination float-end">
    <li class="page-item"><a class="page-link"
href="#">Previous</a></li>
    <li class="page-item" th:each="p,status:${pages}"
><a th:class="${status.index==currentPage ? 'page-link
active': 'page-link '}"
      th:text="${status.index+1}"
      th:href="@{liste-clients(page=${status.index})}">
    </a>
  </li>

    <li class="page-item"><a class="page-link"
href="#">Next</a></li>
  </ul>
</nav>
```

c. Ajouter la Recherche sur la liste des Clients

i. Dans la vue

```
<form class="d-flex my-2" method="get"
th:action="@{/liste-clients}">
  <input class="form-control me-2 w-50"
type="search"
      name="keyword"
      placeholder="Taper un telephone"
aria-label="Search">
  <button class="btn btn-outline-success"
type="submit">Chercher</button>
</form>
```

ii. Dans le Repository

```
Page<Client> findByTelephoneContains(String telephone,  
Pageable pageable);
```

iii. Dans le Controllers

```
@GetMapping("/liste-clients")  
  
public String listerClient(Model model,  
  
                           @RequestParam(name =  
"page", defaultValue = "0") int page,  
  
                           @RequestParam(name =  
"size", defaultValue = "5") int size,  
  
                           @RequestParam(name =  
"keyword", defaultValue = "") String keyword) {  
  
    Page<Client> pageClients= clientRepository  
  
        .findByTelephoneContains(keyword,  
PageRequest.of(page, size)  
  
    );  
}
```

d. Paginer la Rechercher

i. Dans le Controller

```
@GetMapping("/liste-clients")  
public String listerClient(Model model,  
                           @RequestParam(name = "page", defaultValue = "0")  
int page,  
                           @RequestParam(name = "size", defaultValue = "5")  
int size,  
                           @RequestParam(name = "keyword", defaultValue =  
"") String keyword){  
    Page<Client> pageClients= clientRepository  
        .findByTelephoneContains(keyword,  
PageRequest.of(page, size)  
    );  
}
```

```

        //clients.forEach(System.out::println);

model.addAttribute("clients",pageClients.getContent());
        model.addAttribute("pages",new
int[pageClients.getTotalPages()]);
        model.addAttribute("keyword",keyword);

        return "client";
}

```

ii. Dans la Vue

1. Sur le Formulaire

```

<form class="d-flex my-2" method="get"
th:action="@{/liste-clients}">
    <input class="form-control me-2 w-50" type="search"
        th:value="${keyword}"
        name="keyword"
        placeholder="Taper un telephone"
aria-label="Search">
        <button class="btn btn-outline-success"
type="submit">Chercher</button>
</form>

```

2. Sur le lien de Pagination

```

<a th:class="${status.index==status.current ?
'page-link active': 'page-link '}"
th:text="${status.index+1}"
th:href="@{liste-clients(page=${status.index},
                           keyword=${keyword}) }"
></a>

```

e. Lien de Redirection, dans le Controller

```
@GetMapping("/")
public String index() {
    return "redirect:/liste-clients";
}
```

f. Créer un Base Layout

Exemple de template: layout.html

The diagram illustrates the structure of a Thymeleaf layout template. It shows the following components:

- HEADER FIXE**: A blue box containing the header section of the template.
- CONTENT VARIABLE**: An orange box containing the content section, which is defined as a fragment (`layout:fragment="content"`).
- FOOTER FIXE**: A grey box containing the footer section of the template.

```
<!DOCTYPE html>
<html
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/Layout">
<head>
    <meta charset="utf-8"/>
    <title>Gestion des produits</title>
    <link rel="stylesheet" type="text/css" href="../static/css/bootstrap.min.css"
          th:href="@{/css/bootstrap.min.css}" />
    <link rel="stylesheet" type="text/css" href="../static/css/style.css"
          th:href="@{/css/style.css}" />
</head>
<body>
    <header> Header FIXE </header>
    <section layout:fragment="content">
        </section>
    <footer class="navbar-fixed-bottom">
        </footer>FOOTER FIXE
    </body>
</html>
```

i. Dépendance : Thymeleaf Layout Dialect

```
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
```

ii. Crédit de la page Template

```

<!DOCTYPE html>

<html lang="en"    xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

    <head>

        <meta charset="UTF-8">

        <link rel="stylesheet"
        href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">

        <script
        src="/webjars/bootstrap/5.2.3/js/bootstrap.bundle.js">

    </script>

        <title>Gestion de Stock Ism</title>

    </head>

    <body>

        <div class="container mt-2">

            <section layout:fragment="contentForView">

            </section>

        </div>

    </body>

```

iii. Dans les Vues

```

<html lang="en"
      xmlns:th="http://www.thymeleaf.org"

      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
                  layout:decorate="template"
>

<div layout:fragment="contentForView">

```

```
</div>
```

g. Lister les Commandes d'un client

i. Création Entity Commande

1. Les Dates

```
@Temporal(TemporalType.DATE)  
@DateTimeFormat(pattern = "yyyy-MM-dd")  
private Date dateCmd;
```

NB :

Lorsqu'on veut appliquer le même format de date dans tout le projet, on fait la configurer dans le fichier properties

```
spring.mvc.format.date=yyyy-MM-dd
```

ii. Fixtures Commandes

```
Client  
cl=clientRepository.findByTelephone("77867101"+i).get();  
  
Commande commande=new Commande(null,"Com00"+i  
,new Date(),  
100000,  
Etat.Encours,cl);  
commandeRepository.save(commande);
```

iii. Lien sur Clients

```
<td><a th:href="@{liste-cmds-client(id=${cl.id})}" class="btn  
btn-outline-success">Mes Commandes</a></td>
```

iv. Contrôleurs de Commandes

```
@GetMapping("/liste-cmds-client")  
public String listerCommandeUnClient(  
    Model model,  
    @RequestParam(value = "id", defaultValue = "") Long  
idClient) {  
    List<Commande>  
commandes=clientRepository.findById(idClient).get().getCommandes();  
    model.addAttribute("commandes", commandes);
```

```
        return "commande";
    }
```

h. Créer un Client

i. Chargement du Formulaire

1. Vue Liste Client: Ajouter le Bouton Nouveau

```
<div class=" float-end">
    <a th:href="@{/form-client}"
       class="btn btn-outline-info ">Nouveau</a>
</div>
```

2. Crédation de la Classe Client DTO

```
@Data
@Builder
public class ClientDto {
    protected Long id;
    protected String nomComplet;
    private String telephone;
    private String ville;
    private String quartier;
    private String numeroVilla;
    public Client toEntity(){
        return new Client(
                    id,
                    nomComplet,
                    telephone,
                    new Adresse(ville,
                                quartier,
                                numeroVilla));
    }
}
```

3. Dans le Controller

```
@GetMapping("/form-client")
public String loadFormClient(Model model){
    model.addAttribute("client",
                      ClientDto .builder() .build());
    return "add.client";
}
```

4. Vue Ajout Client

```
<form class="col-md-8 offset-2" method="post"
th:action="@{/add-client}">
<div class="mb-3">
    <label for="nomPrenom" class="form-label">Nom et Prenom</label>
    <input type="text" class="form-control" id="nomPrenom"
        th:value="${client.nomComplet}"
        name="nomComplet">
    <div class="form-text text-danger">
```

ii. Enregistrement du Client

1. Dans le Controller

```
@PostMapping("/add-client")

public String addClient(Model model, ClientDto clientDto) {

    Client client=clientDto.toEntity();

    clientRepository.save(client);

    return "redirect:/form-client";

}
```

i. Créer un Client avec Validation

i. Dépendance : Spring Boot Starter Validation

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

ii. Validation De Entite

```
public class ClientDto {
    protected Long id;
    @NotEmpty(message = "erreur")
    @NotNull
    protected String nomComplet;
```

```
@NotEmpty  
@NotNull
```

iii. Dans le Controller

```
@PostMapping("/save-client")  
public String addClient(Model model, @Valid ClientDto clientDto,  
                        BindingResult bindingResult,  
                        RedirectAttributes redirectAttributes){  
  
    if(bindingResult.hasErrors()) {  
        redirectAttributes  
            .addFlashAttribute("type","error");  
        redirectAttributes  
            .addFlashAttribute("sms","Erreur sur les Champs");  
    }else{  
  
        Client client=clientDto.toEntity();  
        clientRepository.save(client);  
        redirectAttributes  
            .addFlashAttribute("type", "success");  
        redirectAttributes  
            .addFlashAttribute("sms", "Client Enregistre avec Succes");  
    }  
    return "redirect:/form-client";  
}
```

iv. Dans la vue

```
<div th:if="${type}">  
    <div class="col-md-8 offset-2"  
        th:text="${sms}"  
        th:class="${type=='error' ? 'alert alert-danger': 'alert  
        alert-success'}" role="alert">  
    </div>  
</div>
```

j. Enregistrement la Commande d'un Client

i. Crédation des Entités

```
public class Article {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    @Column(nullable = false)  
    private String libelle;  
}
```

```
public class LigneCommande {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
    @Column(nullable = false)  
    private double prix ;  
    @Column()  
    private double montant ;  
    @Column()  
    private int qte ;
```

ii. Crédation des Repository

```
public interface ArticleRepository extends  
JpaRepository<Article, Long> {  
}
```

iii. Crédation des Repository

```
public interface LigneCommandeRepository extends  
JpaRepository<LigneCommande, Long> {  
}
```

iv. Fixtures

```
for (int i = 1; i <=10 ; i++) {  
    Client cl=clientRepository.findByTelephone("77867101"+i);  
    Article  
    article=articleRepository.findById(Long.valueOf(i)).get();  
    Commande commande=new Commande(null,"Com00"+i  
        ,new Date(),  
        100000,  
        Etat.Encours,cl,null);
```

```

        commandeRepository.save(commande);
        ligneCommandeRepository.save(
            new LigneCommande(
                null, 10000,
                10000*100, 100,
                commande,
                article));
    }
}

```

v. Dans la Vue Client

```

<a th:href="@{form-cmds-client(id=${cl.id})}" class="btn
btn-sm btn-outline-success">Add Commande</a></td>

```

vi. Maquette Ajout de la Commande

Ndiaye1	778671011	Dakar Point E Villa001	
Choisir un Ar	0	0.0	Ajouter
Article	Prix	Quantite	Montant
Total: 0.0			
Valider la Commande			

vii. Dans le Controller de Commande

1. Définition des Attributs de Session

```

@SessionAttributes({"client"})
public class CommandeController {

```

2. La Méthode Chargement du Formulaire

```

@GetMapping("/form-cmds-client")
public String loadFormCommande(
    Model model,
    @RequestParam(value = "id", defaultValue = "") Long idClient) {
    // Stockage du client dans la session
    Client client=clientRepository.findById(idClient).get();
    model.addAttribute("client",client);
}

```

```

// Recuperation de la Liste des articles pour le
chargement des Select d'Article
    List<Article>
articleList=articleRepository.findAll();
    model.addAttribute("articles",articleList);
// Mapper un Objet Ligne Commande au Formulaire
d'ajout au panier;
    model.addAttribute("ligneCommande",
LigneArticleDto.builder().build());
}

        return "add.commande";
}

```

viii. Vue Ajout de la Commande

1. Affichage des Données du client

```

<div class=" card col-md-10 offset-1">
    <div class="card-body">
        <div class="row">
            <div class="col">
                <input type="text"
                    th:value="${client.nomComplet}"
                    readonly class="form-control"
placeholder="Nom et Prenom" aria-label="First name">
            </div>
            <div class="col">
                <input
                    th:value="${client.telephone}"
                    type="text" readonly
class="form-control" placeholder="Telephone" aria-label="First
name">
            </div>
            <div class="col">
                <input type="text"
                    th:value="${client.adresse}"
                    readonly class="form-control"
placeholder="Adresse" aria-label="Last name">
            </div>
        </div>
    </div>
</div>

```

2. Formulaire d'ajout Article dans le Panier

```
<div class=" card col-md-10 offset-1 my-2">
    <div class="card-body w-100">
        <form class="d-flex my-2 w-100" method="post"
            th:object="${ligneCommande}"
            th:action="@{/add-panier}">

            <div class="row w-100">
                <div class="col">
                    <select name="idArticle"
                        th:field="*{idArticle}"
                        class="form-select" id="specificSizeSelect">
                        <option selected value="0">Choisir un
Article</option>
                        <option th:each="article:${articles}"
                            th:text="${article.libelle}"
                            th:value="${article.id}">
                            </option>
                    </select>
                </div>
                <div class="col-md">
                    <input type="text"
                        name="qte"
                        th:field="*{qte}"
                        class="form-control" placeholder="Quantite"
aria-label="Last name">

                </div>
                <div class="col-md">
                    <input type="text"
                        name="prix"
                        th:field="*{prix}"
                        class="form-control" placeholder="PrixAchat"
aria-label="Last name">

                </div>
                <div class="col">
                    <button class="btn btn-outline-success"
type="submit">Ajouter</button>
                </div>

            </div>
        </form>
    </div>
</div>
```

ix. Ajout dans le panier

1. Définition du panier et du montant total de la commande dans le panier

```
@SessionAttributes({"panier", "total", "client"})
```

```
@ModelAttribute("panier")
public List<LigneArticleDto> panier() {
    return new ArrayList<>();
}

ModelAttribute("total")
public double total() {
    return 0;
}
```

2. Dans la Méthode Ajout dans le Panier

```
@PostMapping ("/add-panier")
public String addArticleToPanier(Model model,
                                  LigneArticleDto ligneArticleDto,
                                  BindingResult bindingResult,
                                  @ModelAttribute("panier")
List<LigneArticleDto> panier,
                                  @ModelAttribute("total") double total,
                                  @ModelAttribute("client") Client client) {
    Article article=
articleRepository.findById(ligneArticleDto.getIdArticle()).get();
    ligneArticleDto.setLibelle(article.getLibelle());
    total+=ligneArticleDto.getMONTANT();
    panier.add(ligneArticleDto);

    model.addAttribute("panier",panier);
    model.addAttribute("total",total);
    return "redirect:/form-cmds-client?id="+client.getId();
}
```

3. Affichage du Panier dans la vue

```
<table class="table table-bordered mt-1 ">
    <thead>
        <th>Article</th>
        <th>Prix</th>
        <th>Quantite</th>
        <th>Montant</th>
    </thead>
    <tbody>
        <tr th:each="article:${panier}">
            <td th:text="${article.libelle}"></td>
            <td th:text="${article.qte}"></td>
            <td th:text="${article.prix}"></td>
            <td th:text="${article.montant}"></td>
        </tr>
    </tbody>
</table>
<div class=" row float-start w-25">
    <button type="button " class="btn btn-info fs-3">
        Total: <span class="badge text-white font-monospace fs-3"
th:text="${total}"> </span>
    </button>
</div>
<div class="row float-end w-25 mt-5">
    <a th:href="@{/add-commande}" type="button " class="btn
btn-info ">
        Valider la Commande
    </a>
</div>
</div>
</div>
```

X. Ajout de la Commande

```
@GetMapping ("/add-commande")
public String saveCommande(Model model,
                           @ModelAttribute("panier")
List<LigneArticleDto> panier,
                           @ModelAttribute("total") double total,
                           @ModelAttribute("client") Client client)
{

    Commande commande=new Commande(null,
        "Com00"+client.getId(),
        new Date(),
        total,
        Etat.Encours,client,null);
    commandeRepository.save(commande);
    panier.forEach(p->
        ligneCommandeRepository.save(
            new LigneCommande(
                null,p.getPrix(),
                total,p.getQte(),
                commande,
                articleRepository.findById(p.getIdArticle()).get()));

    });
    return "redirect:/liste-cmds-client?id="+client.getId();
}
```

k. Enregistrement la Commande d'un Client en utilisant un Service

i. Définir un PanierDto

```
@Data  
@Builder  
public class PanierDto {  
    private double total;  
    private List<LigneArticleDto> articles;  
  
    private Client client;  
  
}
```

ii. Dans le Service Commande

```
@Service  
public class CommandeService {  
  
    public void saveCommande(PanierDto panier) {}
```

iii. Dans Commande Controller

```
@Controller  
@SessionAttributes({"panier"})  
public class CommandeController  
  
    @ModelAttribute("panier")  
    public PanierDto panier() {  
        return PanierDto  
            .builder()  
            .total(0)  
            .articles(new ArrayList<>())  
            .client(null)  
            .build();  
    }
```

```
@GetMapping("/form-cmds-client")
```

```

public String loadFormCommande(
    Model model,
    @RequestParam(value = "id", defaultValue = "") Long idClient,
    @ModelAttribute("panier") PanierDto panier ){
    Client client=clientRepository.findById(idClient).get();
    panier.setClient(client);
    model.addAttribute("panier",panier);
    List<Article> articleList=articleRepository.findAll();
    model.addAttribute("articles",articleList);
    model.addAttribute("ligneCommande",
LigneArticleDto.builder().build());
    return "add.commande";
}

```

```

@GetMapping ("/add-commande")
public String saveCommande(Model model,
                           @ModelAttribute("panier") PanierDto
panier
                           )
{
    commandeService.saveCommande(panier);
    //Réinitialiser le Panier de Commande
    model.addAttribute("panier",this.panier());
    return
"redirect:/liste-cmds-client?id="+panier.getClient().getId();
}

```

iv. Dans ArticleDto

```

public int addQteToArticle(int qte){
    this.qte+=qte;
    return this.qte;
}

```

v. Définir le Controller Panier

```

@Controller
@SessionAttributes({"panier"})
public class PanierController {

```

```

@PostMapping("/add-panier")
public String addArticleToPanier(Model model,

```

```

        @Valid LigneArticleDto ligneArticleDto,
        BindingResult bindingResult,
        @ModelAttribute("panier") PanierDto panier
    ) {

    Article article=
    articleRepository.findById(ligneArticleDto.getIdArticle()).get();
    ligneArticleDto.setLibelle(article.getLibelle());
    panier.setTotal(panier.getTotal()+ligneArticleDto.getMontant());
    //Recherche si l'article existe déjà dans le panier
    OptionalInt pos= IntStream
        .range(0, panier.getArticles().size())
        .filter(i ->

panier.getArticles().get(i).getIdArticle()==ligneArticleDto.getIdArticle()
        ) .findFirst();

    if(pos.isPresent()){
        panier.getArticles()

        .get(pos.getAsInt()).addQteToArticle(ligneArticleDto.getQte());
    }else{
        panier.getArticles().add(ligneArticleDto);
    }
}

model.addAttribute("panier",panier);
return "redirect:/form-cmds-client?id="+panier.getClient().getId();
}

```

vi. Vue Commande

- Affichage des données du client

```

<div class="col">
    <input type="text"
           th:value="${panier.client.nomComplet}"
           readonly class="form-control" placeholder="Nom
et Prenom" aria-label="First name">
</div>

```

- Affichage du Panier

```

<tbody>
<tr th:each="article:${panier.getArticles() }">
    <td th:text="${article.libelle}"></td>
    <td th:text="${article.prix}"></td>
    <td th:text="${article.qte}"></td>
    <td th:text="${article.montant}"></td>
</tr>
</tbody>

```

a. Paginer les Commandes d'un Client

i. Dans le Repository de Commande

```
Page<Commande> getByClient(Client client, Pageable pageable);
```

ii. Dans le Controller

```
@GetMapping("/liste-cmds-client")
public String listerCommandeUnClient(
    Model model,
    @RequestParam(value = "id", defaultValue = "") Long idClient,
    @RequestParam(name = "page", defaultValue = "0") int page,
    @RequestParam(name = "size", defaultValue = "5") int size
) {
    Client client=new Client();
    client.setId(idClient);
    Page<Commande>
pageCommandes=commandeRepository.getByClient(client, PageRequest.of(page, size));
    model.addAttribute("commandes",pageCommandes.getContent());
    model.addAttribute("pages",new
int[pageCommandes.getTotalPages()]);
    model.addAttribute("currentPage",page);
    model.addAttribute("path","liste-cmds-client?id="+idClient);
    return "commande";
}
```

iii. Vue

```
<nav aria-label="Page navigation example mt-3 ">
    <ul class="pagination float-end">
        <li class="page-item"><a class="page-link"
href="#">Previous</a></li>
        <li class="page-item" th:each="p,status:${pages}"
><a th:class="${status.index==currentPage ? 'page-link active': 'page-link '}"
th:text="${status.index+1}"
th:href="@{${path}(page=${status.index})}">
        </a>
        </li>

        <li class="page-item"><a class="page-link" href="#">Next</a></li>
    </ul>
</nav>
```

b. Modifier État d'une Commande

Dans la Vue

```
<a th:if="${cmde.etat.name() != 'Terminer'}"  
th:href="@{form-cmds-paiement(id=${cmde.id})}" class="btn  
btn-sm btn-outline-success">Payer Commande</a>
```

Dans le Controller

```
@GetMapping("/form-cmds-paiement")  
public String payerCommande(Model model,  
                           @RequestParam(value = "id", defaultValue =  
                           "") Long idCommande  
                           ) {  
  
    Optional<Commande>  
    commande=commandeRepository.findById(idCommande);  
    if(!commande.isPresent()) {  
        return "redirect:/liste-cmds" ;  
    }  
    model.addAttribute("commande",commande.get());  
    return "add.paiement.commande";  
}
```

Dans l'énumération

```
public static Etat toEnum(int id) {  
    for (Etat type : values()) {  
        if (type.ordinal()== id) {  
            return type;  
        }  
    }  
    return null;  
}
```

```

@GetMapping("/commande-change-etat")
public String changeEtatCommande( Model model,
                                  @RequestParam(value
= "id",defaultValue = "") Long idCommande,
                                  @RequestParam(value
= "etat",defaultValue = "") int etat){
    Optional<Commande>
OptCommande=commandeRepository.findById(idCommande);
    if(OptCommande.isPresent()){

        Commande cmde= OptCommande.get();
        System.out.println(etat);
        System.out.println(Etat.toEnum(etat).name());
        cmde.setEtat(Etat.toEnum(etat));
        commandeRepository.save(cmde);
    }
    return "redirect:/liste-cmds";
}

```

Dans la vue

```

<form class="d-flex col-md-10 offset-1 my-1" method="get"
      th:action="@{/commande-change-etat}"
>
    <div class="row w-100">
        <div class="col">
            <select name="etat"
                    class="form-select" id="specificSizeSelect"
            >
                <option selected value="0">Estat</option>
                <option value="1" >Terminer</option>
                <option value="2" >Payer</option>
            </select>
        </div>
        <div class="col">
            <button class="btn btn-outline-success"
type="submit">Faire Paiement Commande</button>
        </div>
    </div>

```

```
        <input type="hidden" name="id" th:value="${commande.id}">
    </div>
</form>

<tr th:each="article:${commande.ligneCommandes}">
    <td th:text="${article.getArticle().libelle}"></td>
    <td th:text="${article.prix}"></td>
    <td th:text="${article.qte}"></td>
    <td th:text="${article.montant}"></td>
</tr>
```

c. Valider le Formulaire de Commande

Spring Security

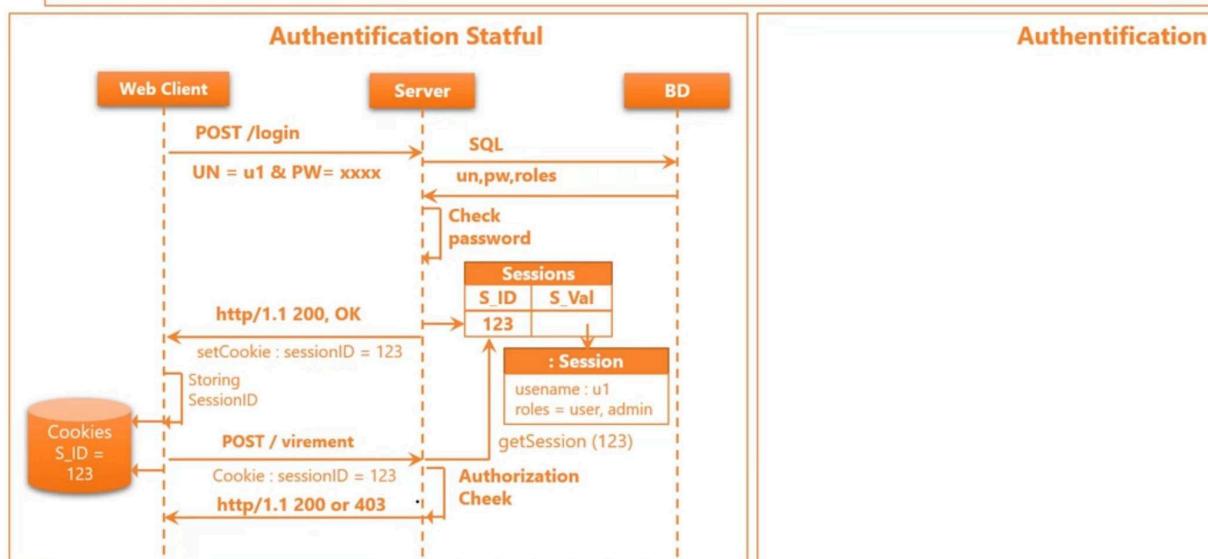
1. Présentation

Ce composant permet de gérer l'authentification et l'autorisation mais aussi la sécurité des API. Il est l'un des composants les plus critiques de Spring Framework, bien qu'il soit aussi l'un des plus complexes.

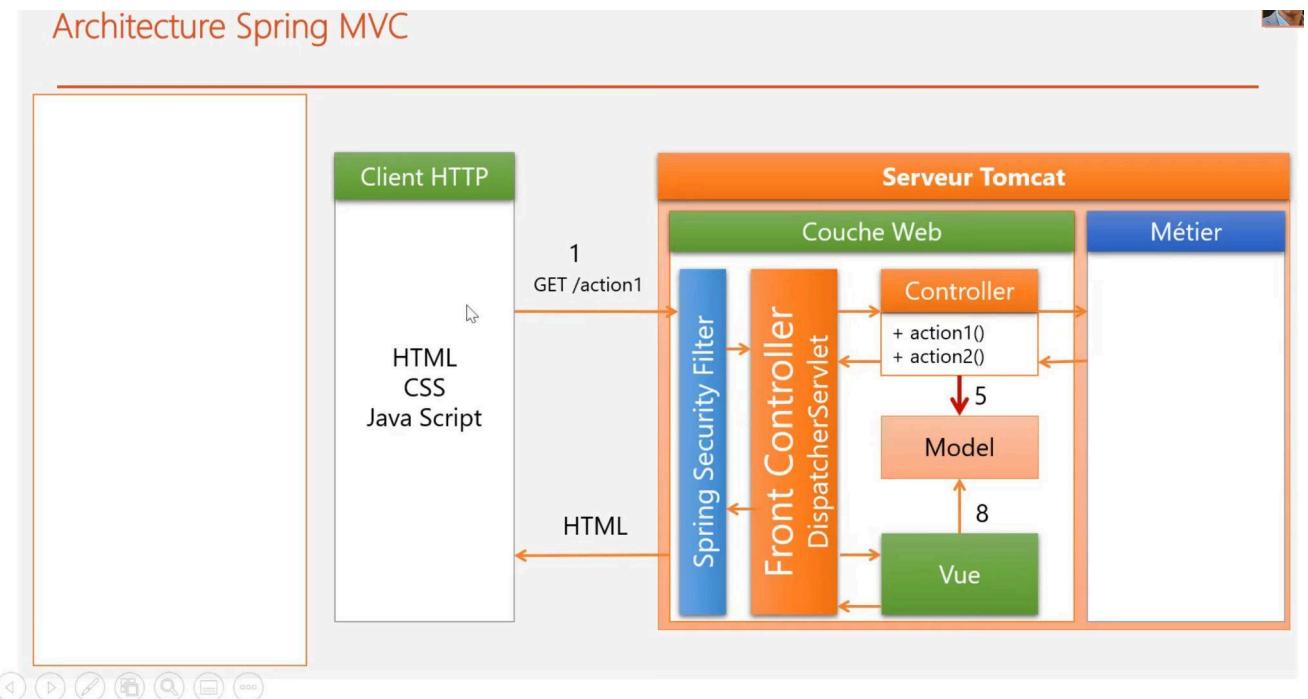
2. Notion Authentication Statefull

Systèmes d'authentification

- Deux types de modèles d'authentification :
- **Statful** : Les données de la session sont enregistrés côté serveur d'authentification
- **Statless** : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.



3. Authentification avec Spring



4. Dépendances

Spring Security

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

Dépendance pour la Gestion des autorisations sur Thymeleaf

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
    <version>3.1.1.RELEASE</version>
</dependency>
```

5. Création de la Classe de configuration

```
@Configuration  
  
@EnableWebSecurity  
  
public class SecurityConfig {
```

@EnableWebSecurity : permet d'activer la sécurité web sur Spring

6. Authentification InMemory

```
@Bean  
  
public InMemoryUserDetailsManager userDetailsService() {  
  
    UserDetails user = User.withDefaultPasswordEncoder()  
  
        .username("user")  
  
        .password("passer")  
  
        .roles("USER", "ADMIN")  
  
        .build();  
  
    return new InMemoryUserDetailsManager(user);  
}
```

7. Gestion des Autorisation

a. Dans Security Config

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
  
    http.formLogin();  
  
    http.authorizeHttpRequests().anyRequest().authenticated();  
  
    return http.build();  
}
```

anyRequest().authenticated() : signifie que toutes les pages sont accessibles après authentification

b. Au niveau des Vues

```
xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity6"
```

```
<li class="nav-item" sec:authorize="hasRole('ADMIN')">  
    <a class="nav-link text-white" th:href="@{/liste-clients}">Client</a>  
</li>
```

NB : Lorsque qu'on veut qu'un élément soit accessible après la connexion on met :

```
sec:authorize="isAuthenticated()"
```

8. Le nom de l'utilisateur Connecté

```
<ul class="navbar-nav mb-2 mb-lg-0">  
    <li class="nav-item float-end">  
        <a class="nav-link text-white" th:href="@{/liste-clients}">  
            <span th:remove="tag" sec:authentication="name"></span></a>  
    </li>  
</ul>
```

9. La Déconnexion

```
<form class="d-flex"  
      th:action="@{/logout}" method="post">  
    <button class="btn btn-outline-white text-white" type="submit">Deconnexion</button>
```

```
</form>
```

10. Protège les URL

```
http.authorizeHttpRequests().requestMatchers("/liste-clients",
"/liste-cmds").hasRole("ADMIN").anyRequest().authenticated();
```

NB :

- Lorsqu'on veut qu'une page soit accessible sans connexion

```
http.authorizeHttpRequests()
    .requestMatchers("/url", "/url").permitAll()
```

- Pour éviter d'avoir à valider une liste très longue Url , on peut les organiser par rôle.

```
http.authorizeHttpRequests().requestMatchers("/admin/**").hasRole("ADMIN").anyRequest().authenticated();
http.authorizeHttpRequests().requestMatchers("/client/**").hasRole("CLIENT").anyRequest().authenticated();
```

11. Formater la Page Erreur

//SecurityConfig

```
http.exceptionHandling().accessDeniedPage("/403");
```

//Controller Error

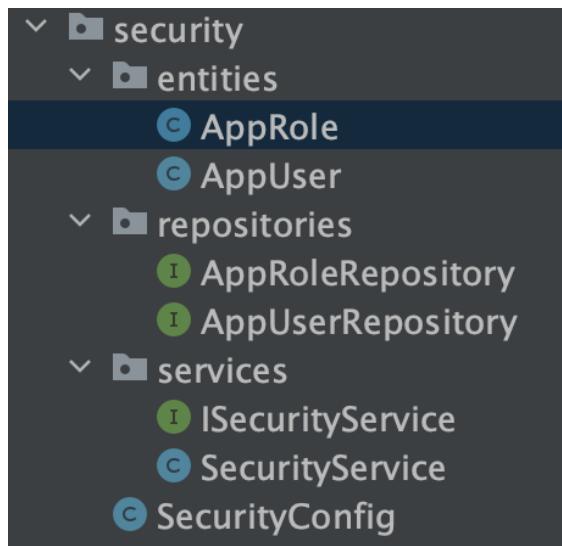
```
@Controller
```

```
public class ErrorController {
    @GetMapping("/403")
    public String error403() {
```

```
        return "403";
    }
}
```

Spring Security: Authentication avec une Base de Donnée

1. Création du package security



2. Création du package entity

```
public class AppRole {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    protected Long id;

    @NotEmpty()
    protected String roleName;

    @ManyToMany(mappedBy = "roles")
    List<AppUser> users;

}
```

```
public class AppUser {  
  
    @Id  
  
    @GeneratedValue(strategy = GenerationType.AUTO)  
  
    protected Long id;  
  
    @NotEmpty()  
  
    protected String nomComplet;  
  
    @Column(unique = true)  
  
    protected String username;  
  
    protected String password;  
  
    protected boolean active;  
  
    @ManyToMany(fetch = FetchType.EAGER)  
  
    @JoinTable(  
  
        name = "users_roles",  
  
        joinColumns = @JoinColumn(name = "users_id"),  
  
        inverseJoinColumns = @JoinColumn(name =  
"roles_id"))  
  
    List<AppRole> roles=new ArrayList<>();
```

NB : On peut avoir des problèmes de référence circulaire, on demandera à spring de les ignorés :

```
spring.main.allow-circular-references= true
```

3.Création du package repositories

```
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {  
  
    AppRole findByRoleName(String roleName);  
  
}
```

```
public interface AppUserRepository extends JpaRepository<AppUser, Long> {  
  
    AppUser findByUsername(String username);  
  
}
```

4.Création du package service

```
public interface ISecurityService {  
  
    AppUser getUserByUserName(String username);  
  
    AppRole saveNewRole(String roleName);  
  
    void addRoleToUser(String username, String roleName);  
  
    void removeRoleToUser(String username, String roleName);  
  
    AppUser saveAdmin(String nomComplet, String username, String password);  
  
}
```

```
@Service  
  
@Transactional  
  
public class SecurityService implements ISecurityService {  
  
    @Autowired  
  
    private AppRoleRepository appRoleRepository;  
  
    @Autowired  
  
    private AppUserRepository appUserRepository;  
  
    @Autowired  
  
    private PasswordEncoder passwordEncoder;
```

```
public SecurityService(AppRoleRepository appRoleRepository,
AppUserRepository appUserRepository) {

    this.appRoleRepository = appRoleRepository;

    this.appUserRepository = appUserRepository;

}

@Override

public AppUser getUserByUserName(String username) {

    return appUserRepository.findByUsername(username);
}

@Override

public AppRole saveNewRole(String roleName) {

    AppRole role =appRoleRepository.findByRoleName(roleName);

    if(role!=null) throw new RuntimeException("Ce Role "+roleName+" existe deja");

    return appRoleRepository.save(new AppRole(null,roleName,null));
}

@Override

public void addRoleToUser(String username, String roleName) {

    AppUser user=appUserRepository.findByUsername(username);

    if(user==null) throw new RuntimeException("Cet Utilisateur n'existe pas");

    AppRole role =appRoleRepository.findByRoleName(roleName);

    if(role==null) throw new RuntimeException("Cet Role n'existe pas");

    user.getRoles().add(role);

    appUserRepository.save(user);
}
```

```

    }

    @Override

    public void removeRoleToUser(String username, String roleName) {

        AppUser user=appUserRepository.findByUsername(username);

        if(user==null) throw new RuntimeException("Cet Utilisateur n'existe pas");

        AppRole role =appRoleRepository.findByRoleName(roleName);

        if(role==null) throw new RuntimeException("Cet Role n'existe pas");

        user.getRoles().remove(role);

        appUserRepository.save(user);

    }

    @Override

    public AppUser saveAdmin(String nomComplet, String username, String password) {

        AppUser admin=new AppUser(nomComplet);

        admin.setUsername(username);

        admin.setPassword(passwordEncoder.encode(password));

        admin.setActive(true);

        return appUserRepository.save(admin);

    }

}

```

5.Dans le Main

```

@Bean

PasswordEncoder passwordEncoder() {

    return new BCryptPasswordEncoder();

}

```

```
@Bean

CommandLineRunner commandLineRunner(ClientRepository
clientRepository,AppRoleRepository appRoleRepository){

    return args -> {

        //Creation des Roles

        AppRole role1=appRoleRepository.save(new
AppRole(null,"CLIENT",null));

        AppRole role2=appRoleRepository.save(new
AppRole(null,"ADMIN",null));

        AppUser admin=service.saveAdmin("Birane Baila
Wane","admin","passer");

        service.addRoleToUser("admin","CLIENT");

        service.addRoleToUser("admin","ADMIN");

        Client client;

        for (int i = 1; i <=10 ; i++) {

            if (i % 2 == 0) {

                client=new Client("Diop" + i, "77867101" + i);

            } else {

                client=new Client("Ndiaye" + i, "77867101" + i, new
Adresse("Dakar", "Point E", "Villa00" + i));

            }

            client.setUsername("client" + i);

            client.setPassword(passwordEncoder().encode("passer"));

            client.getRoles().add(role1);

            client.setActive(true);

            clientRepository.save(client);

            articleRepository.save(new Article(null,"Article"+i));
        }
    }
}
```

```

    }

    for (int i = 1; i <=10 ; i++) {

        Client cl=clientRepository.findByTelephone("77867101"+i);

        Article
article=articleRepository.findById(Long.valueOf(i)).get();

        Commande commande=new Commande(null,"Com00"+i

            ,new Date(),

            100000,

            Etat.Encours,cl,null);

        commandeRepository.save(commande);

        ligneCommandeRepository.save(

            new LigneCommande(

                null,10000,

                10000*100,100,

                commande,

                article));
    }

}

```

6. SecurityConfig

a. Méthode Authentication

```

@Bean

public AuthenticationProvider authenticationProvider() {

    DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();

    authenticationProvider.setUserDetailsService(userDetailsService);

    authenticationProvider.setPasswordEncoder(passwordEncoder);

    return authenticationProvider;
}

```

b. Les Autorisations

Remplace les `hasRole("ADMIN")` to `hasAuthority("ADMIN")`

7.UserDetailsServiceimpl

```
import  
org.springframework.security.core.userdetails.User;  
  
@Override  
  
public UserDetails loadUserByUsername(String username) throws  
UsernameNotFoundException {  
  
    AppUser userConnect = service.getUserByName(username);  
  
    return new User(  
  
        userConnect.getUsername(),  
        userConnect.getPassword(),  
        userConnect.getRoles()  
  
        .stream()  
  
        .map(role-> new  
SimpleGrantedAuthority(role.getRoleName()))  
  
        .collect(Collectors.toList())  
    );  
}
```

8. SecurityController

```
@GetMapping("/")
public String index( @AuthenticationPrincipal UserDetails userDetails){
    String path="redirect:/login";

    if(userDetails.getAuthorities().stream().filter(role->role.getAuthority().compareTo("ADMIN")==0).findFirst().isPresent()){
        path= "redirect:/liste-clients";
    }

    if(userDetails.getAuthorities().stream().filter(role->role.getAuthority().compareTo("CLIENT")==0).findFirst().isPresent()){
        Client client=
        clientRepository.findByUsername(userDetails.getUsername());
        path="redirect:/liste-cmds-client?id="+client.getId();
    }
    return path;
}
```