TP3 : Les fragments Thymeleaf et les requêtes dérivées.

I- Utilisatrion des fragments avec Thymeleaf :

Il est possible de définir des fragments de code qu'on peut réutiliser plusieurs fois dans les vues Thymeleaf. Ces fragments correspondent au code commun dans les différents vues (par exemple une barre de navigation, importation des CDN Bootstrap etc.

Thymeleaf propose les éléments suivants :

- `th:fragment` qui est utilisé pour définir un fragment à réutiliser. Il est défini dans une page HTML considérée comme le template des vues.
- `th:insert` qui permet d'insérer le contenu dans la balise ou elle est placée. La syntaxe est : `<balise th:insert="nom_page_template :: nom_fragment"></balise>`
- `th:replace` qui permet de remplacer (et écrase) le contenu de la balise courante par le contenu de la balise qui définit le fragment. La syntaxe est :

`<balise th:replace="nom_page_template :: nom_fragment"></balise>`

- `th:include` qui permet d'inclure le contenu d'un fragment dans le contenu de la balise courante sans écrasement. La syntaxe est :

`<th:block th:include="nom_page_template :: nom_fragment"></th:block>`

Application au TP :

Créer une page thymeleaf `template.html` qui définit 2 fragments :

- un fragment nommé `header` pour le contenu de la barre de navigation au début du body

```
<body>
<div th:fragment="header">
    <nav class="navbar navbar-expand-sm bg-dark navbar-dark">
        <div class="container-fluid">
    </nav>
</div>

</body>
```

- un fragment nommé `header-file` pour les importations des CDN Bootstrap

```
<head th:fragment="headerfiles">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap
</head>
```

Utiliser les éléments nécessaires pour intégrer ces 2 fragments dans les vues de l'application.

II- Les requêtes dérivées

Les requêtes dérivées (Derived Query) : à partir du nom de la méthode mentionnée dans l'interface Repository, Spring Data JPA est capable d'exécuter des requêtes de sélection avec des critères spécifiques. Considérons l'entité User suivante comme exemple :

```
@Entity
class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;
    private int age;
    private Boolean active;

    // standard getters and setters
}
```

Voici son Repository :

```
interface UserRepository extends JpaRepository<User, Integer> {

}
```

Dans ce Repository on peut déclarer les prototypes des requêtes dérivées suivantes selon la condition désirée :

- Condition d'égalité sur un attribut:
  - `List<User> findByName(String name);`
  - `List<User> findByAge(int age);`
- Condition d'inégalité :
  - `List<User> findByNameIsNot(String name);`
- IS NULL criteria :
  - `List<User> findByNameIsNull();`
  - `List<User> findByNameIsNotNull();`
- Condition d'égalité Booléenne :
  - `List<User> findByActiveTrue();`
  - `List<User> findByActiveFalse();`
- Condition de similarité (attrubut chaine de caractère) :
  - `List<User> findByNameContaining(String name);`
  - `List<User> findByNameStartingWith(String prefix);`
  - `List<User> findByNameEndingWith(String suffix);`
- Condition de comparaison (attribut numérique) :
  - `List<User> findByAgeLessThan(int age);`
  - `List<User> findByAgeLessThanEqual(int age);`
  - `List<User> findByAgeGreaterThan(int age);`
  - `List<User> findByAgeGreaterThanEqual(int age);`
  - `List<User> findByAgeBetween(int startAge, int endAge);`
- Conditions multiples :
  - `List<User> findByNameOrAge(String name, int age);`

- `List<User> findByNameOrAgeAndActive(String name, int age, Boolean active);`

- Pour des requêtes complexes, on utilise l'annotation @Query:
  - `@Query("select u from User u")`
    `List<User> toutLesUsers();`
  - `@Query(" select u from User u where u.age > :x")`
    `List<User> rechercheparage(@Param(value = "x") int age);`

Application au TP :

1. Recherche des films par titre

On souhaite ajouter un formulaire de recherche des films par titre (complet ou partiel) dans la vue `afiche.html`.

- Ajouter dans `FilmRepository` le prototype de la requête dérivée correspondante.
- Ajouter le nécessaire dans la couche service (interface et classe).
- Ajouter dans la vue `affiche` un formulaire pour saisir le titre et un bouton de soummision.
- Ajouter l'action dans `FilmController` qui traite la recherche et affiche son résultat dans la même vue `affiche`.

2. Filtrage des films par catégorie

De la même manière que la recherche par titre, on va définir le filtrage des films par catégorie (sélectionnée à partir d'une liste déroulante dans la vue `affiche`)

- On commence par ajouter le prototype de la méthode qui va sélectionner les films d'une même catégorie dans l'interface `FilmRepository`.
- Déclarer et implémenter une méthode métier pour cette méthode.
- Modifier l'action `all` dans le contrôleur `FilmController` pour envoyer à la vue `affiche` la liste des catégories à afficher.
- Dans la vue `affiche.html`, ajouter à côté du formulaire de de recherche des films par titre le code HTML qui permet d'afficher un autre formulaire ayant la méthode `post` contenant une liste déroulante des catégories avec un première option "Toutes les catégories" (ayant la valeur 0). Le changement d'un choix dans la liste (évènement `onChange`) va soumettre le formulaire (en utilisant la fonction Javascript `submit()` appliquée sur l'élément formulaire ayant un id) pour envoyer la requête de filtrage.
- Il reste maintenant à définir une action dans le contrôleur `FilmController` qui récupère la valeur du paramètre de la requête `idcat` puis selon cette valeur (0 ou non) va rechercher la liste des films par catégorie ou bien toute la liste des films pour les afficher dans la même vue `affiche`. On renvoie aussi à la vue ; la valeur de `idcat` pour savoir quel était la catégorie sélectionnée dans la liste déroulante.