# TP5 : Pagination et Tri des entités affichées dans une vue

Ce TP a pour objectif de réaliser la pagination et le tri selon l'ordre croissant ou décroissant des valeurs d'un attribut des entités affichées dans une vue Thymeleaf.

## Etape 1 : Pagination

Lorsque la liste des entités à afficher, dans une vue, est assez long, il est judicieux de diviser cet affichage sur plusieurs pages et de mettre en place un système simple de navigation entre ces pages commençant par 1 : c'est le principe de la pagination.

En Spring Boot, cette opération est facile à mettre en œuvre grâce à la déclaration de l'interface `JpaRepository` qui hérite d'une interface générique nommé : `PagingAndSortingRepository` :

```
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>
```

Cette interface contient la signature d'une variante de la méthode `findAll()` qui retourne une page d'entité selon des informations passées en argument à la méthode de type Pageable :

```
Page<T> findAll(Pageable pageable);
```

Donc aucun changement au niveau de l'interface `FilmRepository`.

Dans la couche métier, on va ajouter la signature d'une méthode dans `IServiceFilm` et son implémentation dans la classe `ServiceFilm`. Cette méthode retourne une page d'entités Film et possède 2 paramètres : le numéro de la page (commençant à partir de 0) et la nombre d'entité par page :

```
public Page<Film> findPaginetedFilms(int pageNum, int pageSize);
```

L'interface Page est importée depuis :

```
import org.springframework.data.domain.Page;
```

L'implémentation de cette méthode peut être comme suit :

```
@Override
public Page<Film> findPaginetedFilms(int pageNum, int pageSize) {
    Pageable pageable = PageRequest.of(pageNum-1, pageSize);
    return filmRepository.findAll(pageable);
}
```

`Pageable` est une interface qui représente les informations sur la pagination :

- Le contenu de la page courante d'entité : `getContent()`
- Le nombre total des entités : `getTotalElements()`
- Le nombre total des pages : `getTotalPages()`

`PageRequest` est une classe qui implémente `Pageable`. La méthode statique `of` de cette classe permet d'instancier un objet avec les 2 paramètres : numéro de la page et sa taille.

On passe à la méthode pageNum-1 pour correspondre entre la page résultat et l'affichage.

Puis on retourne la page des films avec l'appel de la variante de `findAll()`.

Dans le contrôleur `FilmController`, on va ajouter une action qui traite une requête HTTP envoyé par get ayant un paramètre dans le path correspondant au numéro de la page à afficher (on va fixer le nombre des films par page à 4 par exemple) :

```
@GetMapping("page/{pageNum}")
public String findPagineted(@PathVariable int pageNum, Model model) {
    int pageSize=4;
    Page<Film> page = iServiceFilm.findPaginetedFilms(pageNum, pageSize);
    model.addAttribute("films", page.getContent());
    model.addAttribute("totalPages", page.getTotalPages());
    model.addAttribute("currentPage", pageNum);
    return "affiche";
}
```

Le contenu de la page des entités Film sera affiché dans la même vue affiche. On transmet à cette vue le numéro de la page courante pour effectuer certains contrôles.

Cette méthode devrait initialiser l'affichage des Films avec une première page (1), donc l'action `all` sera modifié comme suit :

```
@GetMapping("all")
public String listeFilms(Model model) {
    model.addAttribute("categories", iServiceCategorie.findAllCategories());
    return findPagineted(1, model);
}
```

An niveau de la vue `affiche.html`, on va ajouter en bas du tableau HTML, l'affichage qui permet d'afficher les numéros des différentes pages qui serviront de navigation entre les pages.

**Précisions sur la syntaxe Thymeleaf :**

Pour afficher une valeur textuelle d'une expression directement dans le code HTML, on peut utiliser la syntaxe : `[[${expr}]]`

Pour effectuer un test avec Thymeleaf, on utilise la syntaxe :

```
<balise th:if="${cond bool}">[[${expr}]]</balise>
<balise2 th:unless="${cond bool}">[[${expr}]]</balise2>
```

th:unless contient la même condition que th:if et représente son else. Balise et balise2 peuvent être différentes mais doit être consécutives.

Sachant que la barre de navigation entre les pages n'apparait que si le nombre de pages est strictement supérieur à 1, voici le code qui permet d'afficher la séquence des numéros des pages :

```
<div class="text-center">
    <div th:if="${totalPages > 1}">
        <span th:each="i : ${#numbers.sequence(1,totalPages)}">
            [[${i}]]   
        </span>
    </div>
</div>
```

Ajouter dans a l'intérieur de la balise `span`, un test portant sur la condition si i est différent de `currentPage` alors il sera un lien hypertexte pointant sur l'action ajoutée dans le contrôleur, sinon il sera affiché comme un simple texte.

Ajouter 2 liens hypertextes Suivant et Précédent permettant de naviguer vers la page suivante ou précédente.

Appliquer en fin le style Bootstrap pour la barre de navigation apparait dans sa finalité comme suit (https://www.w3schools.com/bootstrap5/bootstrap_pagination.php)


## Etape 2 : Tri de la liste des films selon un attribut

Le tri peut être appliqué avec la pagination dans la même méthode, puisque l'interface `JpaRepository` hérite de l'interface : `PagingAndSortingRepository`.

De ce fait, on va ajouter dans la signature et l'implémentation de la méthode `findPaginetedFilms()` deux arguments :

- `sortField` : correspondant à l'attribut sur lequel le tri sera réalisé
- `sortDir` : correspondant au sens du tri, qui prend l'une des valeur `asc` ou `desc`.

```
@Override
public Page<Film> findPaginetedFilms(int pageNum, int pageSize, String sortField, String sortDir) {
    Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name())?
            Sort.by(sortField).ascending() : Sort.by(sortField).descending();
    Pageable pageable = PageRequest.of(pageNum-1, pageSize,sort);
    return filmRepository.findAll(pageable);
}
```

Dans cette méthode on affecte, dans une référence de type Classe `Sort`, le résultat du tri selon sa direction. Ce résultat sera ajouté comme argument à la méthode statique `of` de `PageRequest`.

Un exemple d'URL qui fera référence sur la pagination et le tri aura la forme suivante :

`/page/1?sortField=titre&sortDir=asc`

Ici on demande la page 1 des Films qui sont trié selon l'attribut `titre` ascendant. `sortField` et `sortDir` sont passés comme paramètres dans la requête.

De ce fait, l'action dans le contrôleur `FilmController` subira les modifications suivantes :

```
@GetMapping("page/{pageNum}")
public String findPagineted(@PathVariable int pageNum,
        @RequestParam String sortField, @RequestParam String sortDir, Model model) {
    int pageSize=4;
    Page<Film> page = iServiceFilm.findPaginetedFilms(pageNum, pageSize, sortField, sortDir);
    model.addAttribute("films", page.getContent());
    model.addAttribute("totalPages", page.getTotalPages());
    model.addAttribute("currentPage", pageNum);

    model.addAttribute("sortField", sortField);
    model.addAttribute("sortDir", sortDir);
    model.addAttribute("reverseSortDir", sortDir.equals("asc")? "desc" : "asc");

    return "affiche";
}
```

L'action `all` qui initialisera la pagination et le tri sera modifié comme suit :

```
@GetMapping("all")
public String listeFilms(Model model) {
    model.addAttribute("categories", iServiceCategorie.findAllCategories());
    return findPagineted(1, "titre", "asc", model);
}
```

Dans la vue affiche.html, Pour appliquer le tri selon les valeurs d'un attribut, on va appliquer un lien hypertexte sur le nom de la colonne dans le tableau HTML :

```
<th><a th:href="@{'/film/page/' + ${currentPage} + '?sortField=titre&sortDir=' + ${reverseSortDir}}">TITRE</a></th
```

Compléter le lien hypertexte permettant le tri par année de parution.

Modifier aussi les liens hypertextes dans la barre de navigation de la pagination pour contenir `sortField` et `sortDir`.