Filtrage des films et upload d'image

Ce TP a pour objectif de réaliser le filtrage des films affichés par catégorie dans la vue `affiche` ainsi que réaliser l'upload d'une image d'un film pendant son ajout puis l'affichage de l'image dans une vue.

Etape 1 : Filtrage des films

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 la signature de la méthode qui va sélectionner les films d'une même catégorie (par son id) dans l'interface `FilmRepository`.

```
List<Film> findByCategorieId(int id);
```

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 :

```
@GetMapping("all")
public String listeFilms(Model model) {
    model.addAttribute("films", iServiceFilm.findAllFilms());
    model.addAttribute("categories", iServiceCategorie.findAllCategories());
    return "affiche";
}
```

Dans la vue `affiche.html`, ajouter à coté 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 renvoi aussi à la vue ; la valeur de idcat pour savoir quel était la catégorie sélectionnée dans la liste déroulante.

Etape 2 : Uploade et affichage d'une image d'un film

Upload de l'image

On commence par ajouter un attribut photo dans l'entité Film :

```
private String photo;
```

Ceci a pour conséquence d'ajouter une colonne `photo` dans la table `film` au prochain reload du projet.

Ensuite on va ajouter dans le formulaire d'ajout d'un produit `ajout.html` :

L'attribut `enctype` dans la balise `form` :

```
<form action="/produit/add" method="post" enctype="multipart/form-data">
```

Un input de type `file` pour uploader une fichier image :

```
<label class="form-label">Photos: </label>
<input type="file" name="file" accept="image/png, image/jpeg" class="form-control" />
```

Dans le contrôleur `FilmController`, on va définir le chemin du dossier dans lequel seront uplodées les photos.

```
private String uploadDirectory = System.getProperty("user.dir")+"\\src\\main\\resources\\static\\photos";
```

`System.getProperty("user.dir")` : retourne le chemin sur disque de la racine du projet courant (chemin du répertoire du projet)

Le chemin des images est complété jusqu'à le dossier `photos` **(à créer)** sous `static`.

Toujours dans le contrôleur, on va modifier le code de l'action `add` comme suit :

```
@PostMapping("add")
public String add(Film f, Model model, @RequestParam("file") MultipartFile multipartFile) {

    String fileName = multipartFile.getOriginalFilename();
    Path fileNameAndPath = Paths.get(uploadDirectory, fileName);

    try {
        Files.write(fileNameAndPath, multipartFile.getBytes());
    } catch (IOException e) {
        e.printStackTrace();
    }

    f.setPhoto(fileName);
    iServiceFilm.createFilm(f);
    return "redirect:/film/all";
}
```

L'action prend comme argument le paramètre file de type `MultipartFile` envoyé avec la requête HTTP.

On récupère dans `fileName` le nom original de la photo.

On définit une variable de type `Path fileNameAndPath` à partir du chemin `uploadDirectory` avec le nom de la photo.

Ensuite on génère physiquement la photo dans son emplacement en effectuant son écriture à partir de son fichier binaire.

Enfin avant de créer le film f on définit la valeur de son attribut photo.

On doit importer les classes utilisées à partir des packages suivants :

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
```

Test l'upload d'une image d'un film.

Affichage de l'image d'un produit

On va ajouter pour chaque film un lien hypertexte Détails dont le clic doit afficher la photo, le titre, la description, les acteurs de ce film. Voici le code pour afficher la photo :

```
<img  th:src="@{/photos/{img} (img=${produit.photo})}">
```

Pour les films qui n'ont pas de photo, on peut afficher, en remplacement, une autre image standard placé dans le dossier photos en faisant le test suivant :

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
<title>Détails d'un film</title>
</head>
<body>
<div class="container">
<span th:if="${film.photo == '' || film.photo == NULL}"><img src="/photos/noimage.jpg"></span>
<span th:unless="${film.photo == ''}">
<img  th:src="@{/photos/{img} (img=${film.photo})}"></span>
<h2 th:text="${film.titre}"></h2>
<h4>Description : </h4>
<p th:text="${film.description}"></p>
<h4>Acteurs : </h4>
    <ul>
        <li th:each="acteur: ${film.acteurs}" th:text="${acteur.nom + ' ' + acteur.prenom}"></li>
    </ul>
</div>
</body>
</html>
```

Etape 3 : Effectuer les modifications nécessaires dans la vue et l'action de modification d'un film en tenant (ou pas) compte de l'upload d'images