Dans cette partie vous permettrez à l'utilisateur de cliquer sur un titre de film.

Les liens mèneront vers une méthode de **MovieController** qui devra avoir été au préalable déplacée dans votre application web.

Vous devrez préfixer les urls par **/movie** pour accéder au contrôleur pour respecter les bonnes pratiques liées aux bonnes pratiques du standard REST.

Vous réceptionnerez l'id de l'url généré pour vos titres, par une méthode nommée **displayMovieDetails()**, variabilisée en entrée de cette méthode.

Votre entité Movie ne disposant pas pour le moment d'identifiant; vous devrez ajouter une propriété privée **id** de type **Long** avec ses getters/setters ainsi qu'un attribut pour la **description** du film.

Vous ajouterez au modèle le film dont on veut afficher le détail. Les informations liées à chaque film seront récupérées grâce à une méthode de service **getMovieById()** qui invoquera une méthode de repository **getById()**;

Vous passerez par la classe **MemoryMovieRepository** et utiliserez l'API stream pour filtrer en mémoire le film lié à l'id récupéré dans l'url :

```
@Override
public Movie getById(long id) {
	return movies.stream().
	filter(m -> m.getId()==id).
	findFirst().get();
}
```

Si vous passez par la classe **FileMovieRepository**, vous adapterez le code suivant :

```
@Override
public Movie getById(long id) {

	final Movie movie = new Movie();
	movie.setId(id);
	try(BufferedReader br = new BufferedReader(new FileReader(file))) {

	for(String line; (line = br.readLine()) != null; ) {

		final String[] allProperties = line.split("\\;");
		final long nextMovieId=Long.parseLong(allProperties[0]);
		if (nextMovieId==id) {
			movie.setTitle(allProperties[1]);
			movie.setGenre(allProperties[2]);
			movie.setDescription(allProperties[3]);
			return movie;
		}
	}

	} catch (FileNotFoundException e) {
		e.printStackTrace();
	} catch (IOException e) {
		e.printStackTrace();
	} catch (NumberFormatException e) {
		System.err.println("A movie from the file does not have a proper id");
		e.printStackTrace();
	}

	movie.setTitle("UNKNOWN");
```

```
        movie.setGenre("UNKNOWN");
        movie.setDescription("UNKNOWN");
        return movie;
}
```

Pour ajouter le film au modèle, vous pouvez au choix utiliser un **ModelAndView** ou un **Model**.

La page qui affichera le détail d'un film s'appellera **movie-details.html**, elle permettra entre autres d'afficher le titre, le genre et la description du film en question.

Votre fichier csv ne comportant pour le moment que les informations relatives au titre et au genre, vous devrez rajouter à la main l'id et la description de vos films dans ce fichier :

- 1; Training day; Action; Super film policier.

Dans la page d'accueil, les liens sur les titres mènent à une URL qui contient donc l'identifiant du film. Les films de la liste devront disposer de cet identifiant, vous devrez donc adapter la méthode list() du repository.

Pour générer un lien autour du titre, préférez la syntaxe exploitant un @ qui suit afin de pouvoir inclure le préfixe d'URL /movie :

```
<a th:href="@{/movie/{id}(id=${movie.id})}" th:text="${movie.title}"/>
```

NB :

Attention, nous avons ici rajouté un id et un description à notre entité Movie, ainsi qu'aux films dans notre csv.
Nous allons donc devoir apporter des modifications repository FileMovieRepository qui récupérait l'ensemble des films dans notre fichier csv et adapter le code pour setter pour chaque film récupéré un id et un description pour pouvoir afficher correctement notre tableau html.

D'ailleurs, vous devrez donc rajouter deux colonnes à votre tableau html dans bluraystore-home.html pour afficher l'id et la description de chaque film.