Introduction

L’objectif de ce TP est de réaliser une application dans des circonstances aussi semblables que possible à un réel projet de développement en entreprise.

Expression du besoin

Le gérant d’un magasin de vente de DVD souhaite **rénover** une application permettant de vendre ses DVD en ligne.

Cette application lui permet après authentification de :

- gérer son stock de DVD
- gérer les informations concernant chaque film proposé
  - titre
  - genre
  - nombre d'exemplaires disponible
  - résumé
  - acteur principal
  - acteurs secondaires éventuellement

Les clients de leur côté, peuvent retrouver le film de leur choix en accédant à la liste complète des films :

- l'utilisateur pourra obtenir un descriptif qui reprend toutes les informations du film.

L'application actuelle à rénover ne permet pas à l'utilisateur de commander en ligne. L'utilisateur effectue sa commande par téléphone.

Le gérant du magasin souhaite profiter de la refonte de l'application pour proposer à ses clients de commander en ligne les DVD qu’ils ont choisi :

- le client devra disposer d’un caddie d’achat
- le client pourra s’inscrire en donnant ses coordonnées
- le client pourra commander le contenu de son caddie

Sources

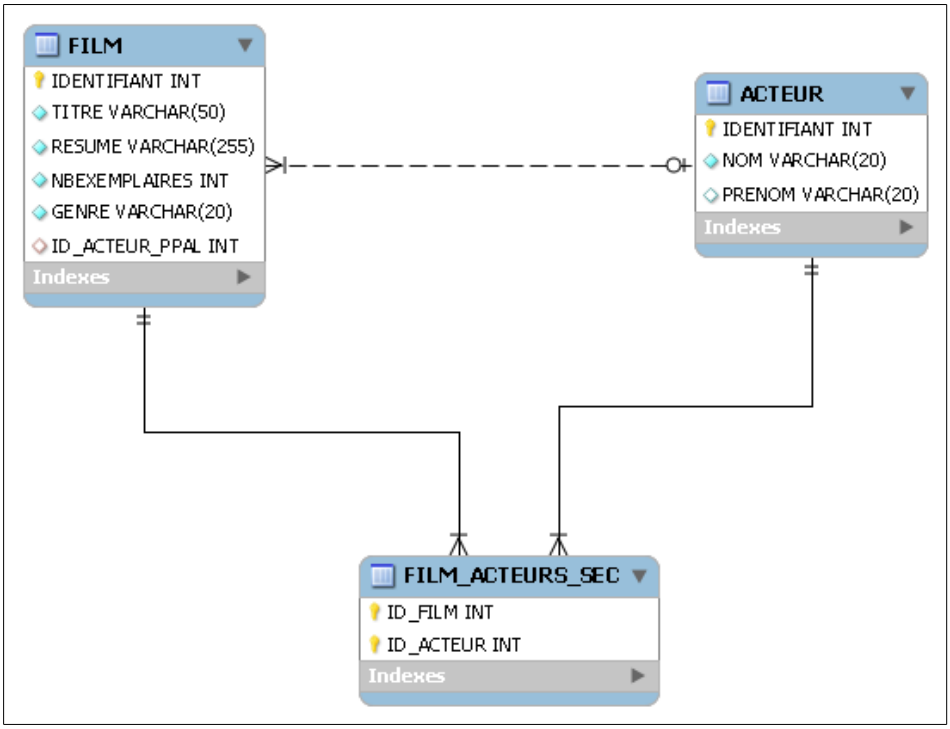
Cette application a été initialement écrite en Modèle JSP 1. Les sources vous sont partiellement fournies.

Etape n°1

1.Création de la base de données. Créez un utilisateur VIDEO / VIDEO ayant une base de données à son nom (Vous pouvez simplement exécuter les lignes de commandes du fichier **1_database.txt** fourni).

2.Pour commencer, nous allons créer les tables ACTEUR, FILM et FILM_ACTEURS_SEC (Vous pouvez simplement exécuter les lignes de commandes du fichier **2_create.sql** fourni).

La base de donnée est constituée de la manière suivante :



3.Insérer un échantillon de données dans la base de données (Vous pouvez simplement exécuter les lignes de commandes du fichier **3_insert.sql** fourni).

4.Intégrez les sources de l'application à Eclipse (répertoire **jsp-model-1**). L'application est composée :

- D'un projet Java contenant les classes du modèle métier
- D'une application Web Dynamique dédiée au front-office (interfaces utilisateur)
- D'une application Web Dynamique dédiée au back-office (interfaces administrateur)

Note : Attention aux chemins des sources de TP1-Commons.

5.Testez l'application :

- Déposez la librairie du Driver MySQL dans le répertoire lib de Tomcat.
- Modifiez les META-INF/context.xml si nécessaire.
- Déployez l'application front-office et l'application back-office dans une instance de

Tomcat grâce à Eclipse.
- Ouvrez un navigateur Web et accédez aux différentes interfaces :
  - http://localhost:8080/TP1-BackOffice
  - http://localhost:8080/TP1-FrontOffice

6.Examinez la structure de l'application

Etape n°2

Transformez les applications en JSP Modèle 2 :

1.Intégrez l'application web « jsp-model-2 » à votre IDE (Eclipse, NetBeans,...).
Cette application fait appel à la méthode à base de « Presentation objects ».

2.Déployez ensuite les applications au sein de l'instance Tomcat et testez-les.

3.Étudiez les sources de cette dernière et la décomposition architecturale mise en œuvre (JSP Model 2).

Etape n°2 Suite

A vous de jouer.

Ajouter au Back-Office une page JSP (`authentification-form.jsp`) permettant à l'administrateur de s'authentifier (login / password).
Ce formulaire doit être soumis à une Servlet `AthentificationServlet` qui doit :

- vérifier que le login est **formation** et le mot de passe **struts2**
- en cas d'erreur, retournez vers la page JSP initiale avec un message "*Mauvais login / mot de passe*" (attention ce message est fourni par la servlet dans un scope sous l'identifiant "`errormessage`")
- retourner vers `home.jsp` en cas de succès.

Etape n°3

Dans les applications frontoffice et backoffice de l'étape 2, faites en sorte que grâce à struts 2 l'url **home.action** dirige l'utilisateur vers la vue **home.jsp**.

Vous placerez le fichier `struts.xml` à la racine des sources de chaque projet (front-office et back-office) et y déclarer **un seul namespace** (« / » fera très bien l'affaire).

Nous placerons le filtre Http sur le mapping /*.
Attention : Le fait d'ajouter un mapping du filtre vers /* va mettre en défaut les mapping de servlet existantes.
Hormis la page d'accueil, les fonctionnalités de l'application deviendront donc inopérantes.

Déployez et testez.

Remarque : Comme indiqué dans le cours, par défaut, l'extension **.action** et l'**absence d'extension** permettent de joindre la même action. Ce comportement est modifiable avec la constante `struts.action.extension`.

Etape n°3 Suite

Dans l'application **backoffice**, faites en sorte que l'url **home.action** ou **home** soit traitée par une action `HomeAction`.

Cette action va inscrire dans la console (`System.out`) :
- Tentative d'utilisation du Back-office le : *dd-MM-yyyy HH:mm*

Déployez et testez.

Etape n°4

Dans le back-office, positionnez la page jsp de **home.action** résultat de sous forme de `global-result`.
Le nom de ce `global-result` sera « home ». Vous devrez donc également adapter l'action.

Etape n°4 Suite

Remplacez maintenant la servlet `ListeServlet` par l'action `ListeAction` côté front et back-office (faites un copier coller à la fin).
L'action héritera de `ActionSupport` et réécrira sa méthode `execute`

Elle sera accessible sous l'uri **liste.action**

Conservez la JSTL en complément éventuelle de la Taglib Struts pour l'itération sur la liste.

La liste à afficher constituera donc une propriété de l'Action

Etape n°5

Créez côté front-office et back-office (faites un copier coller à la fin) l'action `DescriptifAction` héritant de `ActionSupport` et réécrivant sa méthode `execute`. Elle sera accessible sous l'uri **descriptif.action**.

Déplacez les traitements effectués par la servlet `DecriptifServlet` dans la méthode `execute` de l'action. Vous pouvez désormais supprimer `DecriptifServlet` des sources si vous le souhaitez.

N'ayant plus accès à l'instance de `HttpServletRequest`, vous n'êtes plus en mesure de connaître l'identifiant du film à afficher.

Récupérez l'identifiant du film à afficher en utilisant les mécanismes automatiques de translation paramètre → propriétés. Ajoutez donc une propriété (`filmId`) de type int dans l'action sans oublier son accesseur / mutateur.

La transformation en type entier est maintenant automatique, vous ne pourrez plus pour l'instant gérer le cas de l'identifiant non numérique.

Attention : Le nom de la variable de type Film dont on souhaite afficher les propriétés

dans la page JSP à peut-être changé.

N'oubliez pas de modifier les liens vers l'action `DescriptifAction` dans `liste.jsp`.

Déployez et testez.

Etape n°6

Créez l'action `AjoutFilmAction` héritant de `ActionSupport` et réécrivant sa méthode `execute`.
Elle sera accessible sous l'uri **ajoutfilm.action**.

Déplacez les traitements effectués par la servlet `AjoutFilmServlet` dans la méthode `execute` de l'action. Vous pouvez désormais supprimer `AjoutFilmServlet` des sources si vous le souhaitez.

Récupérez le nombre d'exemplaire sous forme de `String` et effectuez la conversion dans le corps de la méthode `execute()`. Nous allons améliorerez ceci plus tard.
Ne vous préoccupez pas pour le moment des erreurs de saisie utilisateur (commentez les zone de code afférentes).

En cas de succès, retournez vers `ajoutok.jsp` grâce au `result` « success ».
Attention, cette page affiche également l'identifiant du film nouvellement créé. Vous allez devoir récupérer cet identifiant à l'aide de la balise `<s:property>`.

En cas d'exception, retournez vers `ajoutnok.jsp` grâce au `result` « error ».

N'oubliez pas de mettre à jour l'attribut `action` du formulaire de la page `ajoutfilmform.jsp`.

Déployez et testez.

Groupe en avance :
Permettre à l'administrateur de modifier un film (acteurs non compris pour l'instant)

Etape n°7

Nous allons maintenant exploiter le scope « session » au travers de `l'ActionContext`.
Pour ce faire, nous allons engager la mise en place d'un caddie d'achat.

1.Vous devez permettre à l'utilisateur du front-office de composer le caddie d'achat.
Pour cela, vous devrez stocker en session les films que l'utilisateur aura sélectionnés.

Nous allons considérer que l'utilisateur ajoute les films un par un et qu'il n'y a pas de notion de quantité.

A partir de la fiche descriptive du film, proposez un formulaire avec un simple bouton « ajouter au caddie ».

L'ajout au caddie sera pris en charge par l'action **ajoutcaddie.action**.
Cette action va ajouter une instance de Film (le film sélectionné bien-sûr) à une instance d'un nouvel objet : le caddie.
Il s'agira d'une instance d'une classe `Caddie` (nouvelle classe à créer). Cette classe ayant

un propriété contenu sous forme de `ArrayList<FilmLightPO>`.
Après ajout, l'utilisateur est reconduit à la page d'accueil.

Note : L'utilisateur qui souhaite commander 2 exemplaires d'un même film utilisera 2 fois le bouton « ajouter au caddie ».

2.Dans un second temps, vous devrez afficher dans la page d'accueil, le nombre de films actuellement présents dans le caddie. Pour cela ajoutez une méthode `getTaille()` qui fera office d'accesseur fictif à une propriété `taille` qui n'existe pas. Cette méthode retourne la taille de la propriété `contenu`.

Note : Ne tenez pas compte du cas où le caddie est vide à l'origine, nous améliorerons la fonctionnalité plus tard.

Groupe en avance :

Transformez le formulaire d'authentification de manière à exploiter une action Struts 2 au lieu d'une servlet.

Etape n°7 Suite

Récupérez / créez l'instance de `Caddie` en exploitant l'interface « Aware » dédié à la session Http.

Etape n°8

Modifiez la page de formulaire de `AjoutFilmAction` afin d'exploiter la balise `<s:form>`.

Les champs titre, nombre d'exemplaire, genre et nom de l'acteur principal doivent être signifiés obligatoires à l'utilisateur.
Attention : La page JSP du formulaire ne peut plus être invoquée sans passer par l'action !
Pour l'instant nous allons :

- Créer un nouveau chemin **ajoutfilmform.action** vers le même type d'action (il faudra donc déclarer une nouvelle balise `<action>`)
- ajouter une méthode à l'action, cette méthode existe déjà au niveau `ActionSupport`, c'est `input()` et elle retourne bien-sûr la vue INPUT.
- Modifier le lien dans la page d'accueil

Déployez et testez et...examinez la source HTML du resultat !

Le contenu HTML généré apparaît sous forme de tableau.
Ceci est configurable, nous en reparlerons plus tard.

Groupe en avance :
Modifier également le formulaire d'ajout au caddie.

Etape n°9

Modifiez à nouveau la page de formulaire de `AjoutFilmAction`.

Cette fois nous allons proposer à l'utilisateur de choisir le genre du film parmi une liste de genre prédéfinis.

4 genres seront disponibles :

- Action
- Aventure
- Comédie
- Fantastique

Le contenu du menu déroulant doit provenir d'une liste positionnée en tant que nouvel attribut de l'`ActionContext` : `#genres`.

Cette liste a vocation à provenir d'une base de données mais dans le cas présent, nous l'alimenterons par le biais de valeurs en mémoire (collection `static` par exemple).

Etape n°10

Remplacez maintenant la servlet `ListeServlet` par l'action `ListeAction` (côté front et back-office).
Remplacez dans toutes les pages la JSTL par les balises de la Taglib Struts.

Groupe en avance :

Modifier `descriptif.jsp` afin d'utiliser exclusivement la tagilb struts.
N'affichez la zone « acteurs secondaires » que s'il existe réellement des acteurs secondaires pour ce film.

Etape n°11

Dans les pages `home.jsp` et `liste.jsp`, utilisez `s:url` et `s:a` pour afficher les différents liens.

Etape n°12

Faites en sorte que les pages d'erreur soient retournées par redirection plutôt que par forward. Ceci permettant à l'utilisateur de constater dans l'url qu'il est reconduit vers la page d'erreur.

Pour simuler une erreur, nous allons ajouter dans l'action `DescriptifAction` un contrôle vérifiant que l'identifiant du film demandé est bien un identifiant correspondant à un film existant.
Nous forcerons ensuite l'erreur en manipulant l'URL d'appel du descriptif dans notre navigateur.

Etape n°12 Suite

Considérons que nous souhaitons publier notre liste de films sur une plateforme de commercialisation de la société X.
La société nous demande simplement de leur mettre à disposition une URL fournissant la liste des titres de film au format XML :

```
<?xml version\"1.0\" encoding=\"UTF-8\"?>
<films>
      <film titre="toto"/>
      <film titre....
</films>
```

Créez côté front-office un nouveau type de résultat `B2BFilmListResult` permettant de fournit cette liste à ce format.

Déclarez ce nouveau type.

Ajoutez dans la configuration `struts.xml` une nouvelle action `listeB2B` pointant toujours vers `ListeAction` mais fournissant le résultat à ce nouveau format.

Testez ce nouveau service grâce à votre navigateur.

Etape n°13

Valider le formulaire de création de Film au moyen de la méthode `validate()` de `AjoutFilmAction`.
Ne vous préoccupez pas pour l'instant du nombre d'exemplaires.

Attention : Une erreur de validation vous conduira directement vers le result « input » sans passer par la méthode `input()`. N'oubliez donc pas de faire le nécessaire pour que l'ensemble des informations nécessaires à l'affichage du formulaire soient disponibles (liste des genres).

Etape n°14

Obtenez le même résultat que l'exercice précédent avec les fichiers XML.

Etape n°14 Suite

Ajouter la validation côté client.
Vous devez vous apercevoir qu'aucune requête au serveur n'est effectuée tant qu'il subsiste des erreurs de validation.

Note : Examiner la source HTML générée de votre vue pour vous assurer de la présence de ces scripts de validation Javascript.

Etape n°15

Transformez l'attribut nombre d'exemplaires en numérique (`int`).
Tentez de saisir une valeur non numérique.
Personnalisez ensuite le message d'erreur.

Etape n°16

Vérifier avant la tentative d'insertion en base qu'il n'existe pas déjà un film ayant le même titre dans la base de données.
Le cas échéant, retournez un message d'erreur à afficher en en-tête de formulaire.

Etape n°17

Reprenez l'exercice précédent.
Cette fois, faites en sorte que, si un film ayant le même titre existe déjà, une `FilmAlreadyExistsException` est alors retournée par la méthode.

Faites également en sorte que ce genre d'exception conduise l'utilisateur vers une nouvelle page d'erreur spécifique `duplicateEntry.jsp`.

Etape n°18

Dans le front-office, factorisez `DescriptifAction` et `ListeAction` au sein de la même action `FilmAction` en suivant le modèle exposé dans le cours.
Les pages jsp seront nommées body.jsp et seront placées dans des répertoires /Film/detail et /Film/liste
Les URLs seront `/Film-liste` (vers la méthode `liste`) et `/Film-detail` (vers la méthode `detail`)

Groupe en avance : Ajoutez dans la home un nouvel item permettant de conduire à la liste des Acteurs et leur détail. Utilisez une nouvelle action `ActeurAction` que vous placerez dans le même package que `FilmAction`.

Etape n°19

Dans le formulaire d'ajout de film, internationalisez les libellés des champs de saisie et des erreurs de saisie vers la langue de votre choix. Considérez que le fichier par défaut package.properties vaut pour le français.

Tester la page en appelant l'url par défaut puis en ajoutant manuellement le paramètre `request_locale`.

Etape n°20

Appliquer à l'ensemble du formulaire d'ajout de film le thème « simple ».
Vous constaterez que nous ne disposons plus :

- Des libellés de champs, vous utiliserez `<s:property>` pour les replacer devant les champs.
- Des * marquant les champs obligatoires, vous ajouterez manuellement ces *.
- Du positionnement des champs en tableau, vous effectuerez un saut de ligne à chaque champ : `<br/>`
- Des messages d'erreur, vous ajouterez en fin de page les messages d'erreur grâce à `<s:fielderror>`

Note : Le fait de passer en thème « simple » va désactiver la fonction de validation côté client. Revenez au thème xhtml si vous le souhaitez.

Etape n°21

Dans le front-office, faites en sorte de rajouter dans toutes pages de liste et de détail

- un texte indiquant en en-tête de quelle entité la page présente la liste ou le détail.
- En bas de page un lien de retour vers la liste pour le détail et vers la page d'accueil pour la liste

Etape n°22

Rappelez-vous, dans l'étape « 2 Suite » nous avions créé une Servlet permettant d'authentifier l'administrateur de l'application Back-Office.
Jusque ici, rien n'oblige l'utilisateur à passer dans ce formulaire d'authentification.

Nous allons désormais grâce à un nouvel intercepteur `UtilisateurNonIdentifieInterceptor` contraindre l'utilisateur à passer sur le formulaire d'authentification si ce dernier n'est pas encore identifié.
Pour cela nous allons simplement placer un booléen en session lorsque l'authentification est un succès, booléen dont la présence sera vérifié par l'intercepteur.

Attention : La servlet d'authentification retournait jusque là vers `home.jsp` en cas de succès. Rappelez-vous que ceci ne peut plus fonctionner depuis que cette page exploite la taglib struts, il faudra donc modifier la servlet afin qu'elle retourne vers `home.action`.

Groupe en avance :
Si vous ne l'avez pas déjà fait au chapitre 7, transformez la servlet en action.
A l'issue d'une authentification fructueuse, ajouter en scope session un objet de type `Utilisateur` plutôt qu'un simple booléen. `Utilisateur` étant une nouvelle classe ayant uniquement une propriété `login`.

Etape n°23

Supprimez l'héritage d'`ActionSupport` dans les classes `HomeAction`.
Tentez la même opération sur la classe `AjoutFilmAction`.

Etape n°24

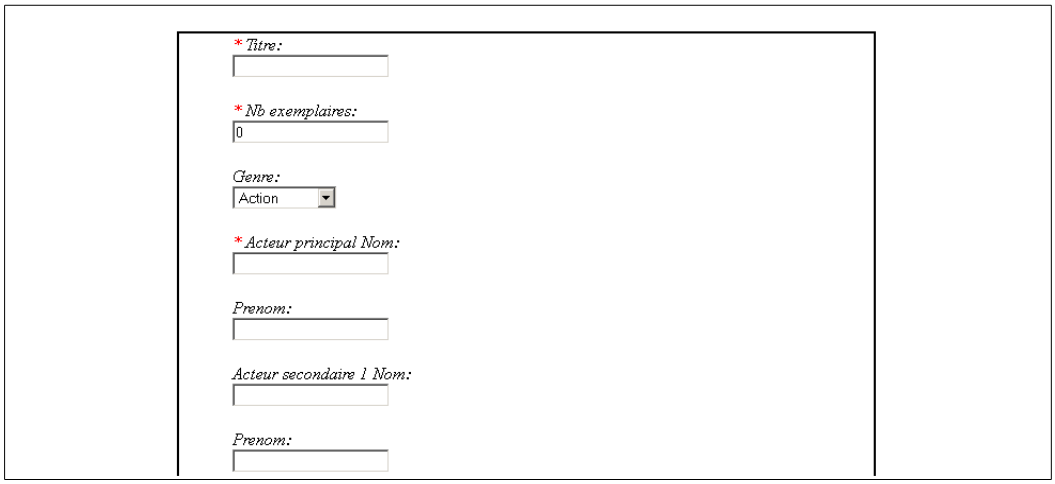
Appliquez le theme css_xhtml à l'ensemble de l'application front et back-office.

Etape n°25

Ajoutez la feuille de style standard proposée par Struts 2 dans l'ensemble des pages.

Etape n°26

Centrez et encadrez d'une bordure fine le formulaire d'ajout de film.
Donnez lui une largeur de 650px.
Placer les éléments du formulaire à 50px du bord gauche.



Groupe en avance :
Apporter d'autres améliorations visuelles si vous le souhaitez.

Etape n°27

Créez le thème « formation » sous le répertoire template à la racine de l'application back-office.
Dupliquez-y l'ensemble des fichiers présent dans le template css_xhtml de la librairie `struts2-core`.

Notre thème aura la particularité de demander à l'utilisateur une confirmation lorsqu'il choisit un bouton submit.
Il s'agira concrètement de faire appel à la fonction `confirm` de Javascript.

Ecrasez le fichier submit.ftl de votre thème par celui fourni dans les ressources du tp.

- Jetez-y un oeil tout de même pour information

Positionner l'attribut `theme="formation"` dans la balise `<s:form>` du formulaire d'ajout de film.

Déployer et vérifier que notre thème fonctionne comme prévu.

Etape n°28

Transformer l'action `DescriptionAction` afin de respecter la méthodologie « Model Driven ».

Groupe en avance :
Faites de même pour `AjoutFilmAction`. Vous allez être confronté aux problématiques de format comme par exemple pour les acteurs secondaires qui, dans la classe `Film` sont représentés sous forme de Collection !
Attention : Dans `ajoutok.jsp`, `${param.}` ne fonctionne pas si l'attribut comporte un « . », utiliser plutôt `<s:property>`

Etape n°29

Déplacer la vérification de doublon de titre effectuée étape 17 dans la méthode `prepare()` de `Preparable`.
Faites en sorte que, si le titre existe déjà, d'ajouter un suffixe à ce titre « _1 » et cette fois de permettre l'enregistrement du film (ne plus retourner vers ERROR).

Attention : Le fait que le backing bean ne soit pas encore alimenté au moment des méthodes `prepare` va vous imposer de récupérer le titre saisi par un autre biais. Utiliser l'interface `ServletRequestAware` afin de récupérer le titre dans les paramètres de `ServletRequest`.

Etape n°30

Dans le back-office, commentez les actions `HomeAction`, `ListeAction` et `DescriptifAction` dans `struts.xml` et adaptez l'application afin que celle-ci continue à fonctionner.
Placer les nouvelles vues dans un répertoire (content ou autre) mais conservez les vues originales situées sous la racine
Ne tenez pas compte des cas d'erreur pour le moment.

Note : Vous remarquerez que l'intercepteur d'authentification est devenu inopérant pour ces actions.

Etape n°31

Dans le back-office, nous allons ajouter une page spécifique dans le cas d'erreur sur `DescriptifAction` (id inconnu).
Cette page devra bien entendu suivre les conventions de nommage permettant à Struts 2 de la retrouver.

Etape n°32

Supprimez le répertoire spécifique créé étape 30 et annotez les actions afin d'exploiter à nouveau les pages de vue d'origine.

Etape n°33

Commentez dans le fichier struts.xml les actions relatives à l'ajout de film et annotez la classe `AjoutFilmAction`.

Etape n°34

Dans la page qui liste les films côté back-office, faites en sorte que le « click » sur le lien « descriptif » affiche le descriptif dans la page courante, juste en dessous du lien.

Etape n°35

Dans la page de descriptif côté front-office, faites en sorte que le « click » sur le bouton d'ajout ne vous conduise plus à la page d'accueil mais en revanche vous affiche directement en dessous du bouton le nombre de films au caddie.

Etape n°36

Obtenez le même résultat avec `<sx:submit>`.

Etape Finale – Groupe en avance

Vous devrez de plus proposer :

1) Un bouton pour vider le contenu du caddie.
2) La possibilité de supprimer un DVD particulier
3) La possibilité de modifier la quantité pour chaque DVD du caddie.

Achever l'application en mettant en oeuvre tous les concepts vu jusque ici.

1.Permettre l'authentification des utilisateurs du front-office. Vous devrez créer une table USER.
2.Permettre à l'utilisateur de commander son caddie. Pour ceci vous allez créer les table COMMANDE et COMMANDE_DETAIL