JDEV 2015 - Atelier T7.A04 - AngularJS

Thierry Chatel

Cet atelier va vous permettre de créer une boutique en ligne, très simplifiée, de vente de livres.

Outils

- **Node.js**
- **Git**
- **WebStorm** ou autre IDE ou éditeur de texte pour les fichiers HTML & JavaScript

JavaScript : quelques rappels et explications

- La ligne suivante au tout début de chaque fichier JavaScript permet de passer en mode strict, un mode ou l'interpréteur est plus rigoureux et signale davantage d'erreurs :
  `"use strict";`

- Pour définir une variable : `var a = value;`
- Pour créer un objet : `{ key1: val1, key2: val2 }` objet vide : `{}`
- Pour créer un tableau : `[elem0, elem1, elem2]` tableau vide : `[]`

- Pour créer une fonction anonyme inline : `function (arg1, arg2) { ...code... }`

- **Conseil important : toujours afficher la console JavaScript, pour ne pas rater les messages d'erreur**

Etape INIT : configuration

1) Charger et décompresser le fichier **angular-1.4.1.zip**, ce qui va créer un sous-répertoire **angular-1.4.1**

2) Cloner le **dépôt Git**, ce qui va créer un sous-répertoire **JDEV2015-T7A04** :

```
git clone https://github.com/tchatel/JDEV2015-T7A04.git
```

3) Copier le fichier **JDEV2015-T7A04/web-server-rewrite.js** dans le répertoire **angular-1.4.1**

4) Dans le répertoire **angular-1.4.1**, lancer un serveur Node.js sur le port **9999** pour avoir AngularJS et sa documentation en local, en tapant :

```
node web-server-rewrite.js 9999
```

et ouvrez la doc dans le navigateur à l'adresse http://localhost:9999/docs/index.html

5) Dans le répertoire **JDEV2015-T7A04**, lancer un second serveur Node.js sur le port **8000** pour l'application web, en tapant cette fois :

```
node web-server.js 8000
```

et ouvrez l'application web dans le navigateur à l'adresse http://localhost:8000/index.html

Arborescence du projet :

css/	feuille de style
data/	simule en local une API serveur pour les données des livres, et les images
img/	images pour la directive *rating*
js/	fichiers JavaScript de l'application, dont le fichier app.js qui contient le module principal avec la définition du routage
templates/	répertoire des templates des différentes routes
index.html	fichier HTML principal

La solution de chaque étape de l'atelier se trouve dans le dépôt Git, dans une branche portant le nom de l'étape (par exemble, branche **A1** pour l'étape qui suit).

Démarrez l'atelier sur la branche **master**, ou sur une branche créée à partir de **master**.

ATTENTION : le copier-coller depuis le fichier PDF a tendance à rajouter des caractères invisibles qui provoquent des erreurs difficiles à comprendre, donc il vaut mieux éviter.

Etape A1 : chargement du catalogue des livres par une requête HTTP

Dans le fichier **js/app-controllers.js**, injecter le service $http au contrôleur **CatalogController**, en lui ajoutant un paramètre nommé `$http`.

> *Explications :*
>
> *AngularJS contient un ensemble de services standards, dont le nom commence par $ pour les distinguer des services créés par les utilisateurs du framework. Par exemple le service **$http** pour faire une requête HTTP, et si le résultat est au format JSON, il est automatiquement parsé et transformé en objet (ou tableau) JavaScript.*
>
> *Un contrôleur est une fonction, appelée comme constructeur, qui sert à initialiser le contexte (**$scope**) de la vue, en y publiant les données et fonctions utilisées dans le template.*
>
> *Injection de dépendances dans un contrôleur : à la fonction contrôleur, indiquer des paramètres dont le nom correspond à la dépendance à injecter, comme $scope pour le contexte de l'élément associé dans le template, ou comme des noms de services. L'ordre des paramètres n'a aucune importance, c'est leur nom qui est déterminant.*

Faire dans le contrôleur une requête GET en appelant :

```
$http.get('/someUrl')
    .success(successFn)
    .error(errorFn);
```

L'URL à appeler est, en relatif : data/catalog.json
ou en absolu (mais avec du coup le numéro de port en dur) : http://localhost:8000/data/catalog.json

Les méthodes **success()** et **error()** prennent une fonction callback qui reçoit 4 paramètres, dont le premier contient les données de la réponse :

```
function(data, status, headers, config) {
    // this callback will be called asynchronously
    // when the response is available
}
```

Dans la fonction callback de succès, publier les données reçues dans une propriété **books** de l'objet $scope.
Dans la fonction callback d'erreur, écrire un message avec `console.log()`.

Dans **catalog.html**, afficher `{{books}}`, et vérifier dans le navigateur web que les données JSON s'affichent.

Etape A2 : affichage du catalogue des livres

Dans le fichier **templates/catalog.html**, la table contient un `<tr>` avec un exemple d'affichage d'un résultat.
Répeter cet élément `<tr>` pour chacun des livres, en utilisant la directive ngRepeat, sous la forme d'un attribut `ng-repeat="book in books"`.

> *Les **directives** d'AngularJS sont des indications pour le compilateur de templates, qui ajoutent des comportements au HTML standard. Elles ont un nom en "camelCase", sous lequel on les trouve dans la documentation, mais il faut les indiquer dans le template HTML sous la forme d'attributs (ou parfois d'éléments) **entièrement en minuscules avec les mots séparés par des tirets**. Donc la directive `ngRepeat` est déclenchée par la présence d'un attribut `ng-repeat`.*
>
> *La directive ngRepeat permet de répéter un élément HTML en fonction des valeurs d'une collection (éléments d'un tableau ou valeurs des propriétés d'un objet), fournie par l'expression qui suit le mot-clef '**in**'.*
> *Elle crée un scope enfant pour chacun des éléments HTML répétés, qui hérite des données du scope parent. Et elle publie la valeur courante de la collection dans une propriété du scope enfant, dont le nom est indiqué avant le mot-clef '**in**'.*

A l'intérieur de l'élément répété `<tr>`, remplacer les données en dur pour afficher pour le livre courant :

- le titre (propriété '**title**')
- le nom de l'auteur (propriété '**author**')
- le prix (propriété '**price**')
- la note (propriété '**rating**')
- le nombre de votes (propriété '**votes**')

Utiliser pour cela une expression entre doubles accolades `{{...}}`, pointant sur la propriété voulue de l'objet `book`.

> *Une **expression** AngularJS a une syntaxe similaire à une expression JavaScript, et elle est toujours évaluée sur le contexte courant (l'objet $scope dans le contrôleur) en fonction de sa position dans le template.*

Pour afficher "*votes*" au singulier ou au pluriel suivant le nombre de votes, vous pouvez utiliser deux éléments `<span>`, un pour chaque version, et les conditionner avec les directives ngShow et éventuellement ngHide, qui prennent une expression qui indique quand l'élément doit être visible / caché.
Un des livres a justement un seul vote.

Pour formater correctement le prix, vous allez utiliser le filtre `currency`. Il faut l'indiquer après une barre verticale (symbole 'pipe' : |), à l'intérieur de l'expression dans les doubles accolades.

> *Un **filtre** AngularJS est une fonction globale qui peut être appliquée à n'importe quelle expression dans un template. Le filtre est indiqué par un caractère **|** suivi du nom du filtre, puis éventuellement par des expressions donnant la valeur de ses paramètres, précédées d'un caractère **:***

Etape A3 : affichage des images des couvertures des livres

Dans l'élément **<img>**, remplacer la référence en dur de l'image dans l'attribut **src** (les 10 caractères avant l'extension .jpg) par le code ISBN-10 du livre courant, qui se trouve dans la propriété '**isbn10**' de l'objet `book`. Il faut mettre là aussi l'expression entre doubles accolades `{{...}}`, juste pour la partie du chemin de l'image qui doit être calculée.

Vérifiez que les images s'affichent bien.

Regardez les messages dans la console. L'erreur est due à une requête intempestive du navigateur lors du chargement du template. Elle peut être corrigée en utilisant la directive ngSrc, sous la forme d'un attribut `ng-src`, à la place de l'attribut `src`. Il n'y a rien d'autre à changer, la valeur de l'attribut reste la même, car contrairement à la plupart des directives d'AngularJS qui attendent une expression (sans doubles accolades), la directive **ngSrc** attend une valeur texte, dans laquelle il peut y avoir une ou plusieurs expressions entre doubles accolades.

Etape B1 : filtrage des données (recherche full-text en local)

Vous aller utiliser le filtre **filter** pour faire une recherche en local dans le texte de toutes le propriétés des objets livres (même les propriétés qui ne sont pas affichées).

Créez un formulaire (balise `<form>`) au-dessus de la table affichant le catalogue des livres, contenant un paragraphe `<p>` avec un champ de saise `<input>` à l'intérieur. Mettez sur ce champ `<input>` une directive **ngModel**, sous la forme d'un attribut `ng-model="searchText"`. Sa valeur est une expression évaluée sur le scope. Cette directive crée un *binding bidirectionnel* entre la valeur dans le champ de saisie et la propriété indiquée par l'expression ; ici une propriété `searchText` de $scope, qui sera mise à jour à chaque changement de la valeur dans le champ de saisie.

Utilisez ensuite le filtre ***filter*** dans l'attribut **ng-repeat** de l'élément <tr>, avec le bon paramètre.

> *Rappel : la syntaxe est* **| nomDuFiltre : valeurDuPremierParamètre**

Etape B2 : création d'un filtre (intervalle de prix)

Ajouter dans un autre paragraphe du même formulaire deux champs pour que l'utilisateur puisse saisir un prix minimum et un prix maximum.

Dans le fichier **js/app-filters.js**, sur le module `'app.filters'` déjà présent, créer un filtre '**range**' prenant deux paramètres, le prix minimum et le prix maximum, en appelant la **méthode** `.filter()` **du module** avec une syntaxe de ce type :

```
.filter('range', function () {
    // return the filter function
    return function(input, param1, param2) {
        // return filter result
    };
})
```

Attention :

- les noms des arguments de la fonction de filtrage sont libres, donc choisissez des noms évocateurs
- le premier argument que reçoit la fonction de filtrage est le tableau des livres, et non pas un seul livre
- tant que les données ne sont pas arrivées du serveur, le premier argument que reçoit la fonction de filtrage vaut **undefined** ; il ne faut pas que la fonction plante quand c'est le cas
- les arguments suivants de la fonction sont les paramètres passés au filtre dans l'expression du template
- la fonction de filtrage doit créer et **renvoyer un nouveau tableau** contenant les livres valides par rapport au filtre
- le filtre doit fonctionner correctement même si l'une ou l'autre des bornes n'a pas de valeur

Ecrire d'abord ce filtre en faisant porter la condition sur la propriété **'price'** en dur de chacun des livres.
Utiliser le filtre '**range**' dans l'attribut **ng-repeat** du <tr>, avec les paramètres adéquats.

Filtre 'range' générique (optionnel, si vous avez bien avancé dans l'atelier) :
Une fois que le filtre fonctionne avec la propriété en dur, on va le transformer pour le rendre générique, en lui passant un paramètre supplémentaire qui est une chaîne de caractères contenant une expression AngularJS. Cette expression sert à calculer la valeur qui doit être comprise entre les deux bornes.
Ici, l'expression à fournir sera donc **'price'**, mais le fait que ce soit une expression plutôt qu'un simple nom de propriété permet par exemple d'accéder à un sous-objet ou d'appeler une méthode.

Dans le filter, pour évaluer l'expression AngularJS, on va utiliser le service **$parse**, qu'il faut injecter (en mettant un argument **$parse**) à la fonction *factory* du filtre, celle passée en second argument de la méthode **filter** et qui renvoie la fonction de filtrage.
L'évaluation de l'expression se fait en deux phases :

- parsing de l'expression, qui renvoie une fonction *getter* correspondant à l'expression :
  `var getter = $parse(expression);`
- évaluation sur un objet servant de contexte, en l'occurrence l'élément courant du tableau :
  `var value = getter(item);`

Etape B3 : validation des critères de recherche

Nommez le formulaire et les trois champs de saisie, en leur mettant un attribut `name` avec les noms suivants : **form**, **searchText**, **minPrice**, **maxPrice**.

> *AngularJS se base sur les valeurs des attributs* **name** *pour publier dans le scope :*
> - *un objet* `form` *contenant notamment les informations de validité du formulaire*
> - *des objets* `form.searchText`, `form.minPrice` *et* `form.maxPrice` *(c'est-à-dire des sous-objets de celui correspondant au formulaire) contenant notamment les informations de validité des champs de saisie*

Mettez, si ce n'est pas déjà fait, un attribut `type="number"` sur les deux champs de saisie de l'intervalle de prix. AngularJS va ainsi valider que la saisie est bien numérique.

On veut 3 caractères au minimum pour faire la recherche *full-text*. Mettez sur le champ de recherche un attribut `ng-minlength="3"`, qui va déclencher la validation d'une longueur minimale.

> *Dans chacun des objets du scope* `form.searchText`, `form.minPrice` *et* `form.maxPrice`, *AngularJS alimente les propriétés suivantes :*
> - `$valid` *qui vaut* **true** *quand l'objet est valide (idem pour le formulaire)*
> - `$invalid` *qui vaut* **true** *quand l'objet est invalide (idem pour le formulaire)*
> - `$error` *qui contient un objet JavaScript, avec comme nom de propriété les clefs de validation (ici :* `number` *et* `minlength`*), et comme valeur de propriété un booléen indiquant s'il y a une erreur pour le validateur correspondant*
>
> *AngularJS place aussi sur chaque champ de saisie, et sur le formulaire lui-même, une classe CSS* `ng-invalid` *lorsqu'il est invalide.*

Dans le fichier ***css/app.css***, ajoutez une règle utilisant cette classe CSS `ng-invalid` pour donner un style différent aux champs en erreur, par exemple une bordure rouge ou un fond coloré.

On veut maintenant afficher un message d'erreur explicite à côté de chaque champ invalide.
Utilisez la directive ngShow pour conditionner la visibilité d'un élément <**span**> contenant le message. La valeur de l'attribut **ng-show** est une expression booléenne. Dans notre cas elle va pointer sur la propriété booléenne correspondant au validateur dans l'objet `$error` associé au champ de saisie, par exemple :

```
ng-show="form.searchText.$error.minlength"
```

Vous pouvez mettre une classe CSS **error**, qui est déjà définie dans le fichier CSS, sur le <**span**> du message d'erreur.

Etape C1 : vue des détails d'un livre

Dans le fichier **js/app.js**, définir une seconde route `'/book/:id'`, où **id** est un paramètre qui recevra l'identifiant du livre sur lequel l'utilisateur a cliqué, en appelant la méthode `$routeProvider.when()` comme pour la route du catalogue déjà présente. Lui associer un template **templates/book.html**, et un contrôleur **BookController**.

> *Dans AngularJS, pour chaque service il existe un **provider** qui est chargé d'instancier le service, et peut permettre de le configurer. Le provider a le même nom que le service, suffixé par 'Provider'. **$routeProvider** permet ainsi de configurer le routage interne à l'application, qui est géré par le service **$route** disponible dans le module optionnel **ngRoute**. La configuration des services doit obligatoirement se faire avant qu'ils soient créés, dans une fonction callback passée à la méthode **config()** d'un module.*
>
> *La route ainsi définie indique le template qui sera chargé dans l'élément `<ng-view>` du fichier index.html (directive **ngView** d'AngularJS), et le contrôleur qui sera utilisé pour initialiser son scope.*

Créer le nouveau contrôleur dans le même module 'app.controllers' du fichier **js/app-controllers.js**, en appelant une seconde fois la méthode **controller()** à la suite de la première. On peut les enchaîner, car comme toute fonction appelée sur l'objet module, la méthode **controller()** renvoie le module sur lequel on l'appelle.
Utiliser le service **$routeParams** (à injecter au contrôleur) pour récupérer la valeur courante du paramètre **id** défini dans la route.

> *Le service $routeParams est un objet qui contient des propriétés avec les valeurs des paramètres définis dans la route.*

Dans ce contrôleur **BookController**, faire une requête HTTP à l'adresse (relative) suivante :
data/2070344134.json où la partie en rouge doit être remplacée par l'identifiant du livre.
Publier les données renvoyées par la requête dans une propriété **book** du scope.

Créer le fichier template, et reprendre le contenu du `<tr>` répété du catalogue, en remplaçant les `<td>` par des `<div>`. Vous pouvez vérifier le résultat dans le navigateur en indiquant manuellement la route `/book/2070344134` dans la partie fragment de l'URL.

Dans le template du catalogue, mettre au niveau de l'image un lien vers la vue des détails du livre. Il faut utiliser un attribut **href** (ou **ng-href**) du type `"#/book/2070344134"`. Le # est indispensable pour que le navigateur crée un lien interne à la page web. Et il faut bien sûr y mettre le bon identifiant de livre, que l'on trouve dans la propriété `isbn10` des objets livres du catalogue. Vous pouvez mettre le même lien sur le titre du livre.

Depuis la vue de détail d'un livre, vous pouvez cliquer sur le lien **Catalogue** du menu, ou revenir à la page précédente dans le navigateur web, pour retrouver le catalogue.

Etape C2 : affichage de la description en HTML

*Les objets livres reçus du serveur contiennent une propriété **description**, dont le contenu est formaté en HTML.*

Dans la vue des détails d'un livre, regarder ce qu'il se passe si on affiche la propriété `{{book.description}}`.

Charger et activer le module optionnel **ngSanitize** fourni avec AngularJS. Il faut pour cela :

- dans **index.html**, charger le fichier **angular-sanitize.js**, comme celui du module **ngRoute** déjà présent
- dans **js/app.js**, mettre le nom du module **ngSanitize** dans la liste des dépendances du module principal **app**, là aussi comme le module **ngRoute** déjà présent

Afficher alors la description en utilisant sur un élément la directive **ngBindHtml**, comme un attribut **ng-bind-html**, qui prend comme valeur l'expression AngularJS indiquant le binding (sans accolades).

> *La directive ngBindHtml refuse par défaut d'insérer dans la page web du code HTML d'origine inconnue, pour éviter de créer une faille de sécurité si le HTML est susceptible de contenir des éléments dangereux comme du JavaScript.*
> *Mais si le module optionnel ngSanitize a été chargé, alors la directive ngBindHtml utilise automatiquement le service $sanitize qu'il contient pour épurer le HTML avant de l'insérer.*

Etape C3 : titre des vues

On veut maintenant que le titre de la page web, visible sur l'onglet et quand on clique avec le bouton droit de la souris sur le bouton "Page précédente" du navigateur, affiche quelque chose de différent en fonction de la vue courante :

- "Catalogue des livres" *sur la vue catalogue*
- "Livre : ...titre du livre…" *sur la vue des détails d'un livre.*

Le titre est défini par l'élément <title>, dans le <head> du fichier **index.html**.
Le problème est que cet élément se trouve à l'extérieur du **<ng-view>**, et n'a donc pas accès aux données publiées dans son scope par les contrôleurs de chaque route.

> *Dans une application AngularJS, les scopes forment une arborescence. Un scope enfant est créé par la directive **ngController**, ou par **ngView**, ou encore pour chaque élément répété par **ngRepeat**. Les scopes enfants héritent des propriétés du scope parent (héritage par prototype de JavaScript, parce que chaque scope enfant a pour prototype son scope parent). Le scope racine de l'arbre est publié comme un service sous le nom **$rootScope**.*

On va commencer par créer un **service** AngularJS, dans le fichier **js/app-services.js**.

> *Les services AngularJS sont similaires aux services utilisés côtés serveur, avec des frameworks comme Spring par exemple.*
> *Un service AngularJS est un singleton qui peut contenir n'importe quel valeur, un objet quelconque (y compris tableau, fonction) ou même un type primitif : chaîne de caractères, nombre, booléen.*
> *Le service est instancié une seule fois par AngularJS, et injecté à tous les contrôleurs, services et mêmes directives ou filtres où il est indiqué comme paramètre.*
> *Un service est créé dans un module. L'objet module dispose de 5 méthodes permettant de créer un service, qui correspondent à des cas d'utilisation et des besoins différents : provider(), factory(), service(), value() et constant().*

Appeler la méthode **value()** sur le module **'app-services'**, en lui passant comme premier argument le nom du service, ici **'pageInfo'**, et comme second argument l'objet service : un objet JavaScript avec une unique propriété **title**, alimentée avec un titre par défaut, par exemple *"Bookstore"*.

Dans les contrôleurs **CatalogController** et **BookController**, injectez ce service **pageInfo** (en ajoutant un argument nommé ainsi à la fonction contrôleur), et dans le code de la fonction affectez la valeur voulue à sa propriété **title**. Faites attention à mettre la ligne au bon endroit dans **BookController** pour avoir le titre du livre.

Maintenant pour récupérer la valeur du titre dans l'élément **<head>**, on va mettre un nouveau contrôleur nommé **HeadController** sur cet élément **<head>** (dans le fichier **index.html**), en utilisant la directive ngController, via un attribut : **ng-controller="HeadController"**

Créez ensuite ce nouveau contrôleur dans le fichier **js/app-controllers.js**, de la même façon que les deux précédents, en injectant à la fonction **$scope** et le service **pageInfo**.
Ce contrôleur doit créer sur son scope (c'est-à-dire sur l'objet **$scope** qui lui est injecté) une fonction **getPageTitle** qui renvoie la valeur de la propriété **title** du service **pageInfo**. Ça ne peut pas être une simple affectation faite lors de l'exécution de ce contrôleur au démarrage de l'application, il faut créer une fonction car la valeur du titre va évoluer. En appelant la fonction, on récupérera toujours la bonne valeur.

Comme contenu de l'élément **<title>**, mettez une expression entre doubles accolades pour afficher la valeur retour de la fonction **getPageTitle**, sans oublier les parenthèses pour appeler la fonction.

Vérifiez que le titre est correct sur l'onglet du navigateur web, et dans l'historique (clic droit sur le bouton "Page précédente") lorsque vous avez enchaîné plusieurs vues en utilisant le lien du menu pour revenir au catalogue.

Etape D1 : directive pour afficher la note du livre sous la forme d'étoiles

Pour l'instant la propriété 'rating' est affichée comme une valeur numérique. On veut afficher à la place une série d'étoiles, grâce à une directive spécifique qu'il va falloir créer.

Mettre dans les templates, là où vous voulez afficher le rating sous la forme d'étoiles, un élément sans contenu avec un attribut **rating** dont la valeur est l'expression AngularJS :

```
<span rating="book.rating"></span>
```

Dans le fichier **js/app-directives.js**, créer la directive **'rating'** dans le module **'app-directives'** existant, en utilisant la méthode **directive()** du module, d'une façon similaire à l'exemple suivant qui reproduit le fonctionnement de la directive **ngBind** (qu'on peut utiliser directement, ou qui est utilisée lorsqu'on met une expression entre doubles accolades) :

```
.directive('ngBind', function () {
    return {
        restrict: 'A',
        link: function(scope, element, attrs) {
            scope.$watch(attrs.ngBind, function (value) {
                element.text(value == undefined ? '' : value);
            });
        }
    };
})
```

*La méthode **directive()** du module prend comme paramètres :*
- *le nom de la directive (en camel case)*
- *une fonction qui sert de factory, à laquelle on peut injecter des dépendances si nécessaire, et qui renvoie un objet de définition de la directive*

*La propriété **restrict** de cet objet de définition de la directive peut contenir les lettres :*
- ***A** pour indiquer que la directive est utilisable comme attribut HTML*
- ***E** pour indiquer qu'elle est utilisable comme élément HTML*

*La fonction **link** reçoit en paramètre dans cet ordre (ce sont des paramètres positionnels, pas de l'injection de dépendances, donc leur nom n'a aucune importance, seul l'ordre compte) :*
- *le scope de l'élément HTML sur lequel est placée la directive*
- *l'élément lui-même, encapsulé dans un objet jQuery (ou l'implémentation interne jqLite d'AngularJS si jQuery n'est pas chargé)*
- *un objet contenant les valeurs des attributs de l'élément HTML, et dont les noms des propriétés sont les noms des attributs convertis en camel case*

*La méthode **$watch()** disponible sur tous les scopes permet de déclencher une fonction callback quand la valeur d'une expression est modifiée. Son premier argument doit être une chaîne de caractères contenant l'expression à surveiller, qui sera évaluée sur le scope sur lequel on appelle $watch(). La fonction callback en second argument reçoit quand elle est appelée la nouvelle valeur de l'expression évaluée.*

Dans votre directive **rating**, utilisez la méthode de jQuery **element.html(...)** pour affecter à l'élément le contenu HTML voulu, ici une série de balises ****. Construisez d'abord le contenu, dans une chaîne de caractères, avec le bon nombre de balises <img>, puis faites un seul appel à la méthode **element.html(...)** car elle écrase le contenu précédent.

Si ça ne marche pas du premier coup, n'hésitez pas à utiliser les outils de debug du navigateur, pour regarder ce qui a été généré dans le DOM et/ou mettre un point d'arrêt dans la fonction link() de la directive.

Etape E1 : tri du catalogue des livres

On veut pouvoir choisir dans une liste déroulante comment doivent être triés les livres du catalogue, parmi différentes options : prix croissant, prix décroissant, titre, note décroissante.

Il va falloir pour cela utiliser :

- le filtre **orderBy**, qui prend comme paramètres :
  - une chaîne de caractères contenant une expression AngularJS pour calculer la valeur sur laquelle les objets du tableau doivent être triées
  - un booléen *reverse*, pour inverser l'ordre du tri quand il est à *true*

- un champ **<select>** avec
  - une directive **ngModel** pour indiquer la propriété cible du scope à alimenter avec l'option de la liste choisie par l'utilisateur
  - une directive **ngOptions**, dont la valeur explique comment construire la liste des options

Commencez par créer un service **sortOptions**, dans le fichier **js/app-services.js**, en appelant la méthode **value()** sur le module, et en lui fournissant comme service un tableau d'objets, avec dans chaque objet :

- le libellé de l'option
- le premier paramètre pour le filtre orderBy : l'**expression sur laquelle trier**
  (en pratique l'expression est ici réduite à une simple propriété)
- le premier paramètre pour le filtre orderBy : le **booléen reverse** pour un tri inversé

Injectez ce service au contrôleur **CatalogController**, et publiez-le dans son scope.

Dans le template **catalog.html**, créez la liste déroulante **<select>** avec ses attributs `ng-model` (propriété qui recevra l'option choisie) et `ng-options` (pour construire la liste des options). **Attention**, `ng-options` n'a aucun effet tant que `ng-model` n'est pas présent.
Utilisez pour cet attribut `ng-options` une syntaxe du type :

```
ng-options="label for item in array"
```

où `label` est une expression dont la valeur le libellé à afficher,
`item` est le nom de la variable qui reçoit chacun des éléments du tableau,
et `array` une expression dont la valeur est le tableau des objets représentant les options.
Et comme pour chaque option on ne spécifie que le libellé à afficher, c'est tout l'objet JavaScript de l'option qui alimentera la propriété cible indiquée dans `ng-model`.

Utilisez le filtre **orderBy** en lui fournissant les valeurs de ses deux paramètres, à partir de l'objet récupéré dans la propriété indiquée dans l'attribut `ng-model` du **<select>**. Utilisez le caractère **:** pour séparer chaque paramètre du précédent, ou du nom du filtre.

Quand on arrive sur le catalogue, on veut que le tri se fasse par défaut par **prix croissant**. Faites le nécessaire pour sélectionner l'option voulue.

Etape E2 : service pour conserver les critères de recherche

Si vous saisissez un ou plusieurs critères de recherche sur la vue catalogue, puis que vous cliquez sur l'image d'un livre pour voir les détails, quand vous revenez au catalogue les critères sont perdus.
Pour les conserver, vous allez les stocker dans un service. Ils seront donc conservés en mémoire tant qu'on reste dans l'application, mais perdus si l'on recharge la page web dans le navigateur.

On veut créer un service générique conservant l'état d'une vue de l'application : critères de recherche, pagination, etc.
Le but est de pouvoir l'utiliser ainsi dans le contrôleur :

```
$scope.state = state('/catalog', {});
```

L'objet publié dans **$scope.state** est l'état de la vue **'/catalog'**, renvoyé par la fonction service **state**. On appelle cette fonction **state** en lui passant comme argument un identifiant de la vue courante, et une valeur initiale pour l'état de cette vue lorsqu'il n'est pas trouvé dans le cache.

Dans le fichier **js/app-services.js**, vous allez utiliser la méthode **factory()** du module pour publier une fonction comme un service nommé **state**.
Le premier paramètre de la méthode **factory()** est le nom du service, donc **'state'**. Le second est une fonction *factory*, qui sera exécutée pour instancier le service, elle doit donc créer et renvoyer le service. Le service est ici une seconde fonction, prenant comme argument l'identifiant de la vue, et la valeur initiale de l'état qui est pour l'instant un objet vide.

La fonction factory doit donc renvoyer la fonction service. Pour stocker l'état des différentes vues, créez simplement un objet **cache** vide, utilisé comme une map avec la notation **cache[viewId]**.

Injectez ensuite ce service **state** au contrôleur du catalogue, et utilisez-le comme prévu pour publier dans le scope l'état sauvegardé de la page :

```
$scope.state = state('/catalog', {});
```

Ainsi l'objet **state** du scope de la vue catalogue va être conservé. Il ne reste plus qu'à faire pointer dans le template **catalog.html** toutes les données à conserver vers des propriétés de cet objet **state** :

- dans les expressions des directives **ngModel**
- et dans les arguments passés aux filtres, pour que ça corresponde

Il faut aussi conserver l'ordre de tri dans la liste déroulante, ce qui va nécessiter une petite modification supplémentaire. En effet, il ne faut plus sélectionner systématiquement l'option "Prix croissant" dans le contrôleur, sinon on écrase la valeur conservée. Faire le nécessaire pour que l'option "Prix croissant" soit sélectionnée la première fois qu'on affiche le catalogue, et seulement dans ce cas.

Etape F1 : service de gestion du panier

Il s'agit maintenant de compléter ce catalogue de livres pour en faire un site e-commerce, en permettant d'ajouter les articles au panier, avant de passer commande.

Créer une nouvelle route `'/cart'` avec son contrôleur `CartController` et son template `templates/cart.html`.

Dans le module **'app-services'**, créer un service `'cart'` de gestion du panier. Ce service va regrouper tout ce qui concerne le panier : les données qui seront ainsi conservées, et les méthodes pour calculer le montant total, pour ajout un article au panier, pour supprimer une ligne.

Définir dans le service 'cart' une propriété **rows**, qui contiendra les lignes du panier.
Pour simplifier le code, la structure suivante est recommandée :

- initialiser **rows** avec un objet vide `{}`
- stocker dans **rows**, indexés par l'identifiant du livre, des objets lignes contenant un livre et une quantité :

```
'2070344134': {
    book: {
        title: '...',
        ...
    },
    quantity: 1
}
```

- pour ajouter un livre *book*, avec une quantité à 1, où *id* est son identifiant :

```
this.rows[id] = { book: book, quantity: 1 };
```

- pour supprimer une ligne row :

```
delete this.rows[id];
```

Définissez dans le service **cart** des méthodes pour :

- ajouter un livre, avec une quantité à 1
- supprimer une ligne
- calculer le montant total du panier, en parcourant les clefs de l'objet avec `Object.keys(this.rows)`
- savoir si le panier est vide : si `Object.keys(this.rows).length == 0`

Ajouter dans la vue des détails d'un livre un bouton *"Ajouter au panier"* (élément **<button>**).
Utiliser sur ce bouton la directive **ngClick** pour déclencher une action quand l'utilisateur clique sur le bouton. La valeur de l'attribut **ng-click** est une expression AngularJS, dans laquelle vous appelez une fonction (exactement comme en JS) qui a été publiée dans le scope par le contrôleur, en lui passant le paramètre nécessaire.

La fonction en question ajoute le livre au panier, et affiche la vue panier. Pour cela, elle fait une redirection vers la route interne du panier, en utilisant le service **$location** (à injecter au contrôleur) :

```
$location.url('/cart');
```

Publier dans le contrôleur **CartController** le service '**cart**' comme une propriété **cart** du scope, puisque le template associé va avoir besoin aussi bien de ses données que de ses méthodes. Et dans le template, afficher un tableau avec pour chaque ligne :

- la photo de couverture, le titre du livre, le nom de l'auteur
- le prix unitaire
- la quantité, dans un champ de saisie pour permettre de la modifier
- un bouton supprimer, avec un directive **ngClick** qui déclenche la suppression de la ligne du panier

Afficher également le montant total du panier.

Utiliser la directive **ngShow** pour afficher un message "*Aucun article*" à la place du tableau quand le panier est vide.