**UniversalList**

**V2.6.0**

Table des matières

[1 CREATION D’UN LISTING SIMPLE 4](#_Toc3471770)

[1.1 Créations de la page HTML et des données 4](#_Toc3471771)

[1.2 Création des colonnes du listing 6](#_Toc3471772)

[1.3 Remplissage des colonnes par les données 7](#_Toc3471773)

[1.4 Gestion du listing 7](#_Toc3471774)

[1.5 Récupération des données 8](#_Toc3471775)

[1.6 Affichage du listing 8](#_Toc3471776)

[1.7 Modification de la largeur des colonnes 10](#_Toc3471777)

[1.8 Récupération de données nombreuses 11](#_Toc3471778)

[2 CREATION D’UN LISTING COMPLEXE 14](#_Toc3471779)

[2.1 Filtres de colonne 14](#_Toc3471780)

[2.2 Fichiers et organisations 15](#_Toc3471781)

[2.3 Surcharge de la classe UniversalList () 15](#_Toc3471782)

[2.3.1 Structure de base 15](#_Toc3471783)

[2.3.2 Définition des colonnes 15](#_Toc3471784)

[2.3.3 Remplissage des colonnes par les données 21](#_Toc3471785)

[2.3.4 Récupération des données. 21](#_Toc3471786)

[2.4 Page principale de notre listing 26](#_Toc3471787)

[2.5 Filtres externes 32](#_Toc3471788)

[2.5.1 Filtre de recherche simple 32](#_Toc3471789)

[2.5.2 Filtre de recherche multiple 34](#_Toc3471790)

[2.5.3 Filtre case à cocher 34](#_Toc3471791)

[2.5.4 Abécédaire 35](#_Toc3471792)

[3 Annexes 38](#_Toc3471793)

[3.1 Modifications 38](#_Toc3471794)

[3.2 Méthodes publiques de la classe UniversalList 40](#_Toc3471795)

[3.3 Méthodes publiques de la classe UniversalListFiltreExterne 57](#_Toc3471796)

[3.4 Code complet 60](#_Toc3471797)

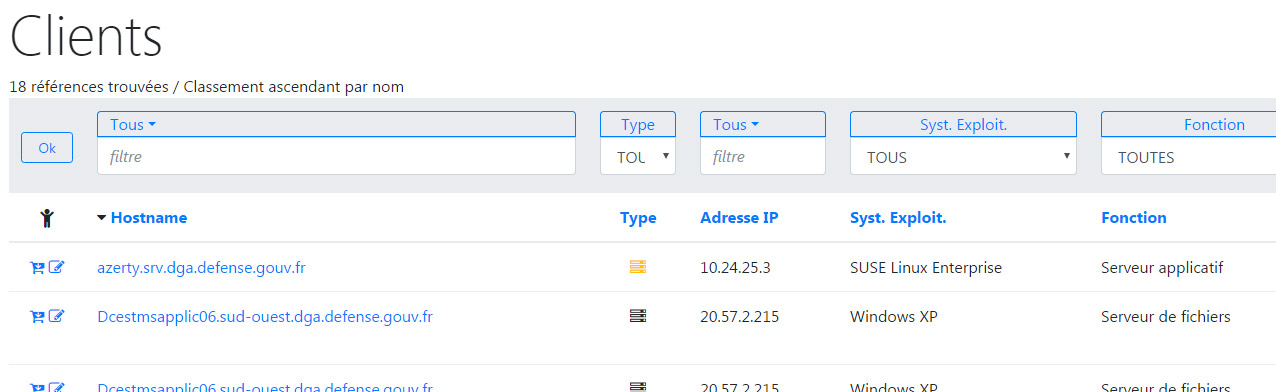
Ce document à pour but de proposer une méthodologie pour construire des listes et des listings en PHP et utilisant la mise en forme du Framework Boostrap V4 et le squelette de développement UniversalWeb (qui contient les classes étudiées) afin de simplifier la construction des pages HTML. Il est supposé que le lecteur en connaissent les usages de base.

Afin de créer rapidement des listes et listing il a été créé deux classes différentes : la première, statique, SimpleListingHelper() fourni un certain nombre de méthodes dont le but est d’aider à la construction de listes et listings simples, c’est-à-dire multi-colonnes avec tris ascendants et descendants possibles.



Figure

La deuxième, UniversalList(), permettra de confectionner des listes et listings plus complexes avec des filtres par colonnes et/ou des filtres externes. Notons que ce type de listing fait aussi appel (mais ce, de manière transparente) aux classes UniversalForm[[1]](#footnote-1), en particulier pour la construction des filtres.



Figure

Dans ce tutoriel, nous allons apprendre à créer ces liste par l’exemple.

# CREATION D’UN LISTING SIMPLE

Nous allons tout d’abord créer un listing simple, c’est-à-dire sans filtres en se faisant aider de la classe SimpleListingHelper(). Nous afficherons dans ce listing une liste de films composée des champs suivant :

* Titre du film (255 caractères)
* Année de distribution (entier)
* Réalisateur (255 caractères)
* Visuel (booléen) (1 pour film en couleur, 0 pour un film en noir & blanc)
* Genre (64 caractères)

## Créations de la page HTML et des données

Nous allons utiliser les données issue de la table « films » d’une base de données créée pour l’occasion. Voici le code de création de cette table qui comporte également l’ajout de quelques titres de films. Notons que nous avons créé des index sur les champs titre, annee et genre. Notons aussi que le codage de cette table est en UTF-8 afin de rester compatible avec le codage UniversalWeb. Ce code est disponible dans le fichier exemple\_listes.sql.

CREATE TABLE IF NOT EXISTS `films` (

`titre` varchar(255) NOT NULL,

`annee` int(4) unsigned NOT NULL,

`realisateur` varchar(255) NOT NULL,

`visuel` tinyint(1) unsigned NOT NULL DEFAULT '1',

`genre` varchar(64) NOT NULL

) ENGINE=MyISAM DEFAULT CHARSET=utf8;

-- Contenu de la table `films`

INSERT INTO `films` (`titre`, `annee`, `realisateur`, `visuel`, `genre`) VALUES

('2001: l''odyssée de l''espace', 1968, 'Stanley Kubrick', 1, 'Science-fiction'),

('Amityville, la maison du diable', 1979, 'Stuart Rosenberg', 1, 'Horreur'),

('Chantons sous la pluie', 1952, 'Stanley Donen', 1, 'Comédie musicale'),

('Il était une fois dans l''Ouest', 1968, 'Sergio Leone', 1, 'Western'),

('L''Empire contre-attaque', 1980, 'Irvin Kershner', 1, 'Science-fiction'),

('La Guerre des étoiles', 1977, 'George Lucas', 1, 'Science-fiction'),

('La planète des singes', 1968, 'Franklin J. Schaffner', 1, 'Science-fiction'),

('La prisonnière du désert', 1956, 'John Ford', 1, 'Western'),

('Le bon, la brute et le truand', 1966, 'Sergio Leone', 1, 'Western'),

('Les parapluies de Cherbourg', 1964, 'Jacques Demy', 1, 'Comédie musicale'),

('Quo Vadis', 1951, 'Mervyn LeRoy', 1, 'Drame'),

('Autant en emporte le vent', 1939, 'Victor Fleming', 0, 'Drame'),

('Le jour le plus long', 1962, 'Ken Annakin', 0, 'Guerre'),

('La bête humaine', 1938, 'Jean Renoir', 0, 'Drame');

-- Index pour la table `films`

ALTER TABLE `films`

ADD UNIQUE KEY `titre` (`titre`),

ADD KEY `annee` (`annee`),

ADD KEY `genre` (`genre`);

Pour les besoins de ce tutoriel, partons sur le code ci-dessous : Il s’agit juste de créer la page HTML qui va s’occuper d’afficher notre listing. Nous ne reviendrons pas sur cette partie et tout le code que nous allons maintenant écrire se trouvera après les commentaires « Code personnalisé préparatoire à écrire ICI » et « Code personnalisé de la page commence ICI ». Entrons ce code dans le fichier exemple\_liste\_simple.php.

<?php

//-----------------------------------------------------------

// INDEX

//-----------------------------------------------------------

// ééàç : pour sauvegarde du fichier en utf-8

//-----------------------------------------------------------

**require\_once**('libs/common.inc.php');

//-----------------------------------------

//Code personnalisé préparatoire à écrire ICI

//-----------------------------------------

//---------------------------------------------------------

// head

//---------------------------------------------------------

$titrePage = \_APP\_TITLE\_;

$scriptSup = '';

$fJquery = '';

echo **writeHTMLHeader**($titrePage, '', '');

//---------------------------------------------------------

// body

//---------------------------------------------------------

echo '<body>';

//---------------------------------------------------------

// Corps APPLI

//---------------------------------------------------------

echo '<div class="container">';

echo '<div class="row" style="min-height:65rem">';

//------------ colonne centrale ---------------

//lg et xl -> taille 10 colonnes

//en dessous prend toute la largeur de la fenetre (12 colonnes)

echo '<div class="col-12">';

//panel de l'application

echo '<section class="bg-light px-3 souligne">';

echo '<div class="row">';

//marque

echo '<div class="col-12">';

echo '<p class="h1">Listings</p>';

echo '<p><span class="lead">Liste simple</span></p>';

echo '</div>';

echo '</div>';

echo '</section>';

//code propre à la page

echo '<article style="margin-bottom:3rem;">';

echo '<div class="row">';

echo '<div class="col-12">';

//-----------------------------------------

//Code personnalisé de la page commence ICI

//-----------------------------------------

echo '</div>'; //col

echo '</div>'; //row

echo '</article>';

echo '</div>';

//------ fin colonne centrale -----------

echo '</div>'; //row

//---------------------------------------------------------

// Footer APPLI

//---------------------------------------------------------

**include\_once**(\_BRIQUE\_FOOTER\_);

echo '</div>'; //container

echo '</body>';

echo '</html>';

## Création des colonnes du listing

Sauf instruction contraire, tout le code que nous allons écrire maintenant devra l’être dans la partie « Code personnalisé préparatoire ». Nous allons maintenant créer les colonnes du listing. Voici le code de création de la première colonne de notre listing de films :

$cols['titre'] = **SimpleListingHelper::createCol**(array('name' => 'Titre', 'size' => 40, 'tri' => 'titre'));

Explications :

Il est fait appel à la méthode statique createCol(**array** $parametres) de la classe SimpleListingHelper()**.** La méthode reçoit un tableau de paramètres qui vont caractériser la colonne. Dans notre exemple, ce sont les paramètres suivant :

name : indique le titre à afficher dans l’entête de la colonne.

size : la taille de la colonne en pourcentage (la totalité des colonnes doit faire 100%).

tri : le champ SQL de la table « films » qui sera utilisé pour trier sur cette colonne de notre liste.

Il existe d’autre paramètres qui sont :

align : alignement du texte dans la colonne (choisir parmi left, center, right).

sens : le sens du tri (ASC ascendant, DESC descendant) dans le cas d’une colonne triée.

title : le texte qui apparaitra en info bulle au survol de la colonne.

header : booléen (true / false) qui définit la colonne comme entête pour la ligne (défaut false).

Aucun paramètre n’est obligatoire. Dans le pire des cas, si l’on entre une array() vide en guise de paramètres de la colonne, la colonne ne permettra pas de tri, aura comme le titre 'Nom Colonne ?' (pour bien marquer l’oubli), une taille de 10% et un alignement du texte à gauche (left).

Le résultat de cette création (retour de la méthode) doit aller dans un tableau à laquelle nous donnerons un titre parlant : ici $cols['titre'] soit la colonne ‘titre’ … quoi de plus naturel ?

Construisons les colonnes suivantes :

$cols['titre'] = **SimpleListingHelper**::**createCol**(array

('name' => 'Titre',

'size' => 40,

'tri' => 'titre',

'header' => true));

$cols['annee'] = **SimpleListingHelper**::**createCol**(array

('name' => 'Année',

'size' => 10,

'align' => 'center',

'tri' => 'annee'));

$cols['real'] = **SimpleListingHelper**::**createCol**(array

('name' => 'Réalisateur',

'size' => 25,

'title' => 'Nom du réalisateur'));

$cols['visuel'] = **SimpleListingHelper**::**createCol**(array

('name' => '<span class="fas fa-tv"></span>',

'size' => 10,

'align' => 'center'));

$cols['genre'] = **SimpleListingHelper**::**createCol**(array

('name' => 'genre',

'size' => 25,

'tri' => 'genre'));

Et voilà ! Nous avons maintenant à disposition un tableau $cols qui définie notre liste. Idéalement la somme des tailles des colonnes devrait arriver à 100%. Mais tout fonctionnera quand même bien si ce n’est pas le cas. Noter l’utilisation de code font-awesome pour afficher une petite icone sur le titre de la colonne « visuel » à la place de texte conventionnel.

## Remplissage des colonnes par les données

Nous allons maintenant définir comment les données vont venir s’afficher dans les colonnes. Pour cela nous allons créer une fonction par colonne à afficher. Chacune de ces fonction doit avoir la syntaxe suivante :

Col\_[clé\_de\_la\_colonne]**(array** $donnee**, int** $page**)**

Ainsi, la fonction qui va se charger d’afficher la colonne ‘titre’ va s’écrire ainsi :

function **Col\_titre**($donnee, $page) {  
 echo $donnee['titre'] ;  
}

Le premier paramètre ($donnee) contient le tuple à afficher issu de la base de données. Le deuxième paramètre ($page) contient le numéro de la page en cours d’affichage. Ces informations seront remplies ultérieurement.

Ainsi le code nécessaire au remplissage de toutes les colonnes de notre listing de films est le suivant :

function **Col\_titre**($donnee, $page) {

echo $donnee['titre'];

}

function **Col\_annee**($donnee, $page) {

echo $donnee['annee'];

}

function **Col\_real**($donnee, $page) {

echo $donnee['realisateur'];

}

function **Col\_visuel**($donnee, $page) {

echo $donnee['visuel'];

}

function **Col\_genre**($donnee, $page) {

echo $donnee['genre'];

}

## Gestion du listing

Puisque nous parlons de notions de page, le moment est venu d’aborder la gestion de ce listing. Celle-ci est confiée à la méthode statique getParams(). Elle a pour rôle de préparer l’affichage aux tris possibles. Cette méthode doit être appelée après la création des colonnes et avant avant l’appel aux données (discuté plus bas). Voici la syntaxe d’appel de cette méthode :

**SimpleListingHelper::getParams**(**string** $idColonne, **array** &$cols, **int** &$page, **string** &$tri, **string** &$sens)

* idColonne : identifiant de la colonne par défaut sur laquelle trier (ex : ‘titre’). Le sens du tri de la colonne est pris en compte. Si aucun champ de tri n’est disponible pour la colonne par défaut, l’affichage de la liste peut conduire à une erreur SQL dans le cas où la requête qui sera exécutée utiliserait une clause ORDER BY.
* cols : notre tableau de colonnes définit plus haut.
* page : la page à afficher en cours.
* tri : le champ SQL de tri en cours.
* sens : le sens du tri en cours (ASC ascendant ou DESC descendant).

Les paramètres cols, page, tri et sens sont des variables passées par adresse, ce qui signifie que c’est la méthode qui leur donne (ou modifie) une valeur en retour de son appel.

Voici le code pour notre listing cinématographique

//-------------------------------------

// Gestion du listing

// Le premier paramètre est l’identifiant de la colonne par défaut sur laquelle trier

//-------------------------------------

**SimpleListingHelper::getParams**('titre', $cols, $page, $tri, $sens);

## Récupération des données

Point de listing sans données. Voici la phase de récupération des données. Bien entendu cela va se faire via une requête SQL.

Voici dans notre exemple basés sur la table « films » la requête SQL qui convient :

**SELECT** titre, annee, realisateur, visuel, genre

**FROM** films

**ORDER BY** [tri] [sens]

Avec [tri] le champ SQL (donc la colonne) sur lequel notre listing va être trié et [sens]le sens du tri.

Une fonction peut se charger de la récupération de ces données :

function **getListe**($tri, $sens, &$laListe) {

$laListe = array();

$requete = "SELECT titre, annee, realisateur, visuel, genre ";

$requete.= "FROM films ";  
 $requete.= "ORDER BY ".$tri." ".$sens." ";

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $nombre;

}

$laListe = null;

return false;

**}**

**Attention** : Si votre liste ne contient aucun champ trié, ne pas renseigner la clause ORDER BY.

Noter que - et ceci est important -, la fonction renvoie le nombre de lignes trouvées par la requête. Ce nombre va nous servir plus tard. L’appel doit donc être du style :

//-------------------------------------

// Récupération des données à afficher

//-------------------------------------

$totalLignes = **getListe**($tri, $sens, $donnees);

## Affichage du listing

Reste à afficher notre listing. Rien de plus simple avec la classe SimpleListingHelper(). Il suffit de placer le code suivant dans la zone « Code personnalisé de la page » de notre page HTML.

//affichage du nombre de lignes trouvés

//--------------------------------------

SimpleListingHelper::**drawTotal**($totalLignes);

//affichage du tableau

//--------------------------------------

echo '<table class="table table-hover table-striped">';

//affichage de l'entete

**SimpleListingHelper**::**drawHead**($cols, $tri, $sens);

//affichage du corps du tableau : donnees

**SimpleListingHelper**::**drawBody**($cols, $donnees, $page);

echo'</table>';

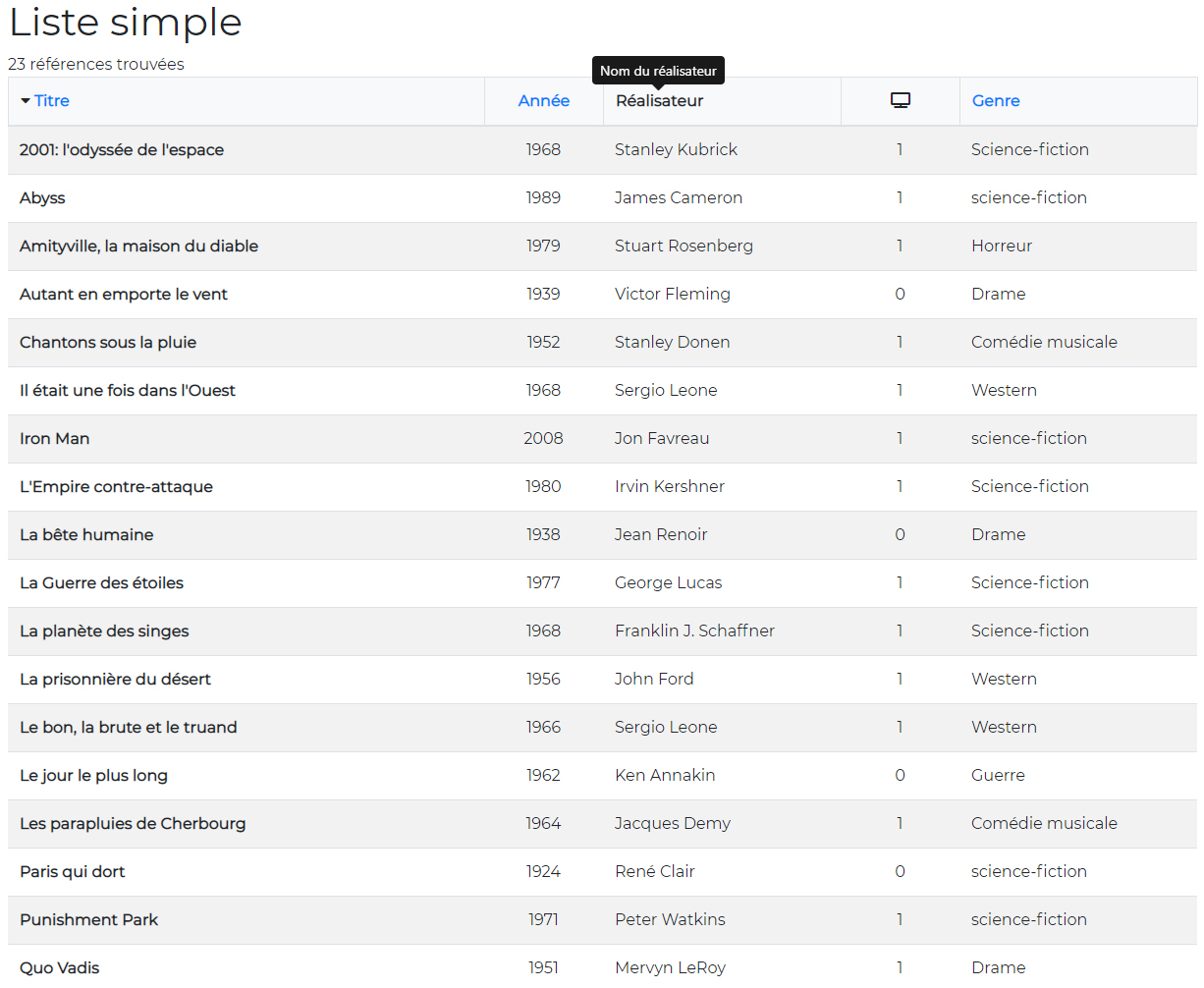
Tout d’abord, la méthode statique drawTotal(**int** $nb\_lignes)est appelée. Elle a pour rôle d’afficher en haut de notre liste le nombre de lignes ramenées. On lui passe comme paramètre $totalLignes qui a été renvoyé par l’appel SQL via la fonction getListe() vue juste avant.

Ensuite, on va procéder à l’affichage du listing proprement dit. Le listing est en fait une table au sens HTML, agrémentées de CSSs Bootstrap.

La classe statique drawHead(**array** $cols, **string** $tri, **string** $sens) est chargée de dessiner l’entête de la liste (nom des colonnes et informations de tris). Un 4ème paramètre optionnel permet de modifier la classe CSS de l’entête de la liste. Exemple d’usage : drawHead(**array** $cols, **string** $tri, **string** $sens, ‘bg-light’).

La classe statique drawBody(**array** $cols, **array** $donnees, **int** $page) est chargée d’afficher le contenu (les données) de la liste.

Voici le résultat. Magique non ? On remarquera que le survol du titre de la colonne « réalisateur » affiche une bulle d’information au survol de la souris … c’était demandé par le paramètre « title » lors de la construction de la colonne ! Enfin, la colonne « Titre » a été définie comme entête de la liste (paramètre header), c’est pourquoi, les titres de film s’affichent de manière plus prononcés.



Figure

Notez aussi la petite flèche descendante sur la colonne « Titre » à côté du libellé. Ceci signifie que notre liste est actuellement triée sur la colonne « titre » dans l’ordre Ascendant, qui est l’ordre par défaut. Si l’on avait voulu débuter l’affichage par un ordre inverse, il aurait fallu passer le paramètres ‘sens’ => ‘DESC’ à la colonne « titre » lors de sa conception.

Un clic sur chaque colonne triable modifie l’ordre du tri.

## Modification de la largeur des colonnes

L’utilisateur peut également modifier la largeur de chaque colonne une fois la liste affichée lors de l’exécution du script. Pour cela nous allons poser une petite touche de Javascript avec le code qui se trouve dans le fichier js/resizable.min.js, fichier qui doit bien entendu être chargé par la page (en utilisant UniversalWeb, ceci est fait automatiquement lors de l’appel à la fonction writeHTMLHeader).

Pour activer ce code il est tout d’abord nécessaire de nommer votre liste avec un identifiant. C’est ce qui est fait ici par ajout du code id=''tableId'' au tag html table.

echo '<table **id="tableId"** class="table table-hover table-striped">';

Il faut ensuite « brancher » le code javascript permettant la manipulation des colonnes à votre liste. Pour ceci, ajouter le code javascript suivant au paramètre $scriptSup de la page dessinant la liste.

$scriptSup = '';

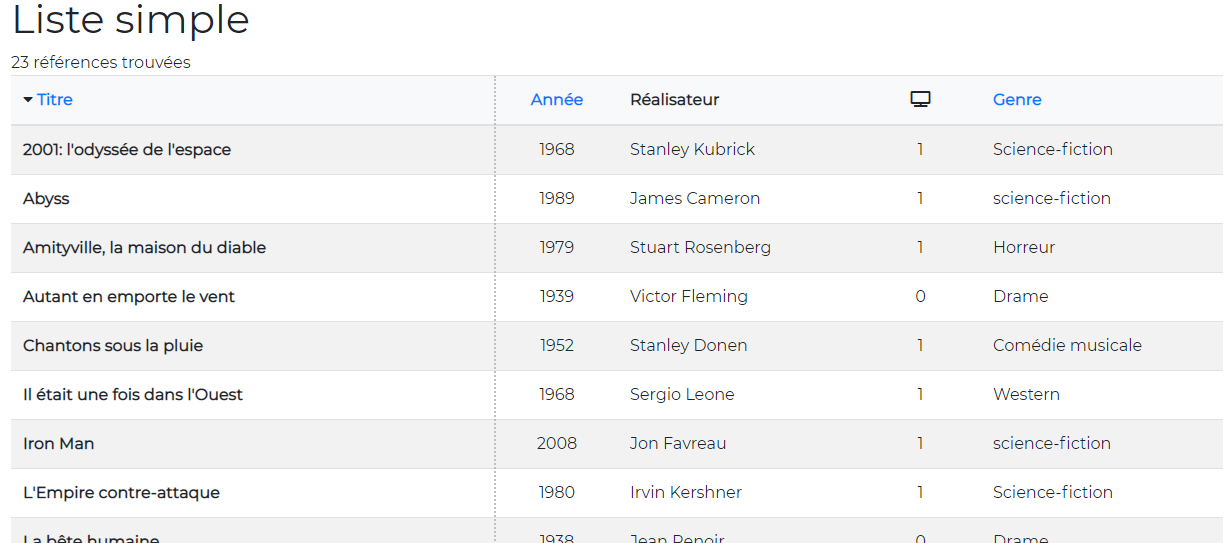
$scriptSup.= '<script>';

$scriptSup.= 'var table = document.getElementById(\'**tableId**\');';

$scriptSup.= 'resizableGrid(table);';

$scriptSup.= '</script>';

Pour voir le résultat, un clic sur une inter-colonne permet par simple drag & drop de modifier la taille de la colonne. La ligne représentant l’inter-colonne apparaît alors en pointillés. Ci-dessous un exemple de cette manipulation.



Figure

Tout cela c’est bien beau mais si j’ai des centaines de lignes à ramener, cela va faire un listing très long non ? Oui c’est vrai. Pour cela nous allons modifier notre code…

## Récupération de données nombreuses

Dans le cas où notre table devrait ramener un grand nombre de données, il est nécessaire de penser à la pagination de notre liste. Afin de ne pas ralentir le script, nous allons optimiser la requête pour qu’elle ne ramène QUE les données qui vont être affichées sur la page en cours. Voyons comment cela va fonctionner.

Voici la requête dans le cas où la table contiendrait un grand nombre de données, l’idée étant de ne ramener QUE les données visibles sur la page du listing en cours…

**SELECT** titre, annee, realisateur, visuel, genre   
**FROM** films   
**ORDER BY** [tri] [sens]   
**LIMIT** [debut], [nombre]

Avec [tri] le champ SQL (donc la colonne) sur lequel notre listing va être trié, [sens] le sens du tri, [debut] la première ligne à ramener en fonctions de la page en cours et [nombre] le nombre de lignes à ramener (c’est-à-dire le nombre maximal de lignes à afficher dans le listing). La fonction qui peut se charger de la récupération de ces données pourrait être :

function **getListe**($tri, $sens, $start, $nb\_lignes, &$laListe) {

$laListe = array();

$requete = "SELECT titre, annee, realisateur, visuel, genre ";

$requete.= "FROM films ";

$requete.= "ORDER BY ".$tri." ".$sens." ";

$requete.= "LIMIT ".$start.", ".$nb\_lignes;

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $nombre;

}

$laListe = null;

return false;

**}**

L’appel pourrait donc être du style

//-------------------------------------

// Récupération des données à afficher

//-------------------------------------

$totalLignes = **getListe**($tri, $sens, $start, $nb\_lignes, $donnees);

A voir l’appel de cette fonction, on constate qu’il nous faut définir le tuple de départ de l’affichage sur la page en cours (paramètre $start). Sur la première page, pas de problème, le tuple n°1 fera l’affaire, mais sur les pages suivantes ? On ne peut résoudre ce problème qu’en spécifiant le nombre de lignes à ramener par page et en prenant en compte le numéro de la page en cours affichée. Et puis il nous faut aussi un système de navigation entre les pages !

Et bien pour réaliser tout cela nous allons employer la classe PageNavigator() également disponible dans le squelette UniversalWeb. Nous n’allons pas ici rentrer dans les détails de cette classe (se reporter à sa documentation pour cela) mais allons simplement la mettre en œuvre. Juste pour rappel la classe affiche un navigateur de page en page construit de boutons comme le montre la figure 1 au début de ce document.

Voici sa syntaxe d’appel :

**PageNavigator**(**int** $total\_data, **int** $nb\_lignes\_par\_page, **int** $nb\_item\_menu, **int** $page\_encours, [**int** $schema], [**int** $indicePage], [**string** $langue]);

Sans entrer ici dans les détails

* $total\_data : nombre total de lignes que peut ramener la liste.
* $nb\_lignes\_par\_page : nombre de lignes à afficher par page.
* $nb\_item\_menu : c’est le nombre de boutons de navigation à afficher au maximum entre les boutons de début et de fin… c’est un paramètre juste cosmétique !
* $page\_encours : c’est le numéro de la page en cours.

Nous connaissons tous les paramètres sauf $total\_data. Il faut donc faire appel à une autre requête SQL qui va nous renvoyer ce nombre. Voici la fonction qui va faire office :

function **getListeNombre**() {

$requete = "SELECT count(\*) nombre ";

$requete.= "FROM films";

$res = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($res !== false) {

if ($nombre != 0) {

return $res[0]['nombre'];

}

else return $nombre;

}

return false;

**}**

Voici enfin le code à afficher :

//-------------------------------------

// Récupération des données à afficher

//-------------------------------------

$totalLignes = **getListeNombre**();

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, 5, 2, $page);

$debut = $pn->**getItemDebut**();

$pn->setPageOff();

//lecture des articles

$nombreLignes = **getListe**($tri, $sens, $debut, 5, $donnees);

Explications :

La fonction getListeNombre() renvoie donc le nombre total de tuple de la table « films » dans la variable $totalLignes.

Cette valeur est utilisée par la classe PageNavigator() qui est instanciée dans la variable $pn. On a au passage demandé à l’objet d’afficher un maximum de 5 lignes par page et 2 boutons intercalaires. Point d’orgue, c’est l’objet $pn qui nous renvoie l’indice du premier tuple à afficher en haut de la page en cours via l’appel à la méthode getItemDebut(). Nous avons donc maintenant toutes les données pour demander à notre requête SQL getListe() de nous retourner les seules données qui seront affichées sur la page en cours et restituées dans la variable $donnees.

$nombreLignes = **getListe**($tri, $sens, $debut, 5, $donnees);

Reste à afficher notre liste multipages. Et là rien de plus à faire par rapport à un listing mono-page, si ce n’est de demander l’affichage du navigateur de pages juste avant celui de la liste...

//affichage barre de navigation

//--------------------------------------

echo $pn->**draw**();

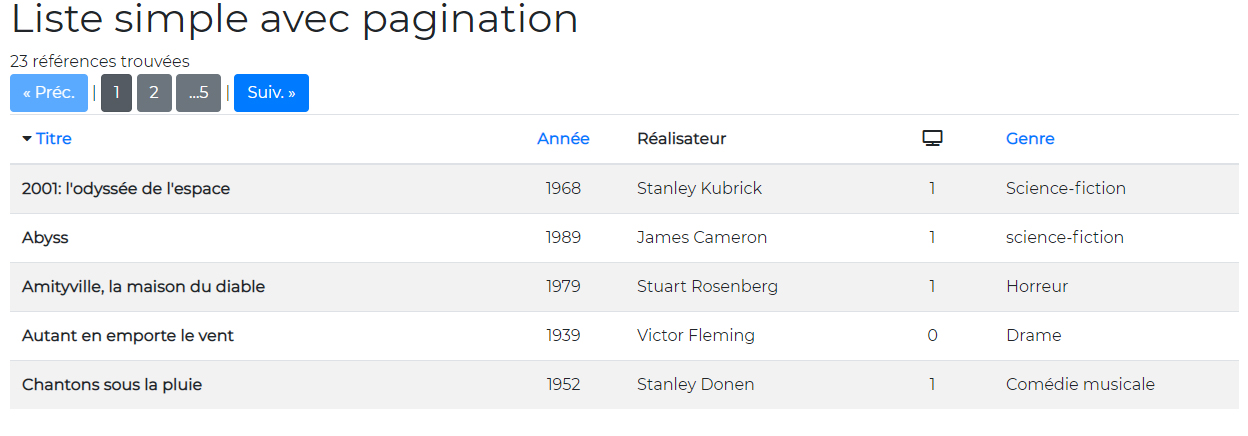


Figure 5

Enjoy !

Récapitulatif de la classe SimpleListingHelper()

class **SimpleListingHelper** {  
 public function **createCol**(**array** $tableau)   
 public function **getSize**(**array** $tableau)   
 public function **sizeAlarm**(**array** $tableau){  
 public function **getParams**(**string** $idColonne, **array** &$cols, **int** &$page, **string** &$tri, **string** &$sens)  
 public function **drawTotal**(int $nombreLignes)  
 public function **drawHead**(**array** $cols, **string** $tri, **string** $sens, [**string** $css])  
 public function **drawBody**(**array** $cols, **array** $listing, **int** $page)  
}

Autres méthodes disponibles

getSize()  
int **getSize**(**array** $tableau)

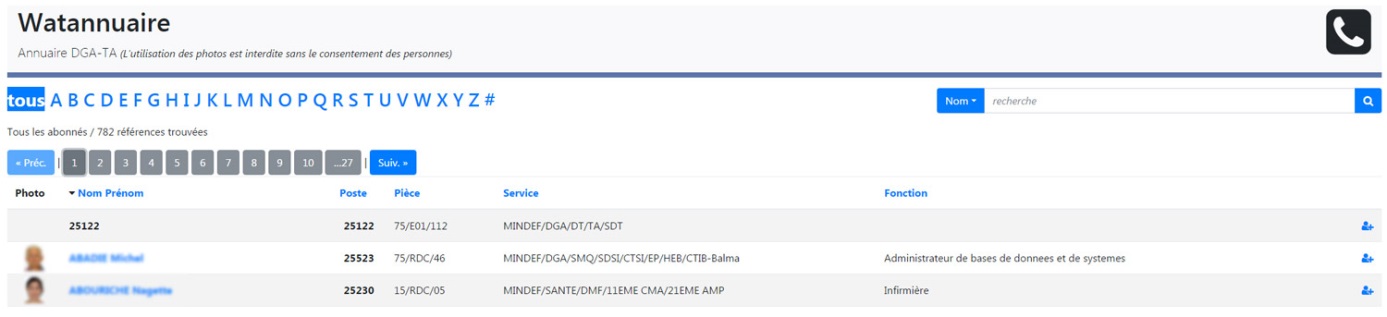
Additionne les tailles de toutes les colonnes pour info.   
ex : echo **getSize**($cols);

sizeAlarm()  
**sizeAlarm**(**array** $tableau)

Envoi un message en DEBUG\_ lorsque la taille totale des colonne est différente de 100%.   
ex : **sizeAlarm**($cols);

# CREATION D’UN LISTING COMPLEXE

Comme nous le disions un listing dit complexe est composé de filtres de recherche, soit sur les colonnes (figure 2), soit externe (figure 6), soit les deux.



Figure

## Filtres de colonne

Nous allons construire une nouvelle liste avec filtres de colonne. En conservant notre exemple de travail sur des films, nous allons poser un filtre sur la colonne « titre » , un filtre sur la colonne « visuel » et un filtre sur la colonne « genre ». Il existe trois types de filtres :

|  |  |
| --- | --- |
| **Filtre de type texte** : Avec sa zone de texte le filtre permet de chercher une information saisie par l’utilisateur … ou une partie de cette information si l’on ajoute des champs d’affinage tels que « Tous », « Egale à », « Commence par », etc. Ces paramètres d’affinages sont libres et paramétrables … | Figure |
| **Filtre de type select** : Ici pas de zone de saisie, mais un sélecteur avec des choix prédéfinis. | Figure |
| **Filtre de type checkbox** : Une simple boit à cocher va permettre de réaliser un filtre binaire. | Figure |

Ces trois types de filtres héritent des classes UniversalForm. Depuis la version 2 de UniversalList, il n’est plus nécessaire de définir directement ces filtres. Ils seront automatiquement générés par la classe. **A noter que UniversalList n’est pas compatible avec les version de PHP inférieure à 5.6.**

A la différence des listes simples qui se font aider par la classe SimpleListingHelper() et ses méthodes statiques, les listes complexes s’appuient sur la classe UniversalList() qu’il faut (cette fois-ci) instancier.

## Fichiers et organisations

Pour construire une liste complexe il est nécessaire de prévoir deux fichiers :

* Le fichier de la page principale : fichier contenant le code d’affichage de la page. Appelons le par exemple exemple\_liste\_complexe.php.
* Le fichier de classe qui dérive la classe UniversalList : Appelons le exemple\_listing\_films.class.php.

Nous allons faire évoluer ces deux fichiers en parallèle.

## Surcharge de la classe UniversalList ()

Travaillons tout de suite sur la surcharge de la classe UniversalList() dans le fichier exemple\_listing\_films.php.

### Structure de base

Voici sa structure de base. Comme on peut le voir, notre classe hérite de la classe UniversalList :

<?php

/\*-----------------------------------------------------------------

Classe Exemple\_listing\_films

------------------------------------------------------------------

éè : pour enregistrement UTF-8

Auteur : Fabrice Labrousse

Date : 30 novembre 2017

------------------------------------------------------------------\*/

**class** Exemple\_listing\_films **extends** UniversalList {

//=======================================

// Méthode protégées, c'est à dire uniquement

// utilisable par les classes dérivées

//=======================================

protected function **construitColonnes**() {

}

//=======================================

// Méthode publiques

//=======================================

}

### Définition des colonnes

Première chose, la définition des colonnes. Nous allons confier la création des colonnes à la méthode protégée construitColonnes(). Comme toute méthode protégée, celle-ci va être appelée par la classe parente, sachant qu’elle saura créer les colonne éventuellement filtrées de notre liste. Le nom construitColonnes n’est pas modifiable. Chaque colonne de la liste va être créé par appel de la méthode createCol().

createCol(**string** $id, **array** $parametres)

* id est une chaine de caractères représentant la colonne, c’est cet identifiant qui sera utilisé pour toute action à réaliser sur la colonne. Attention : le mot « operation » n’est pas autorisé comme identifiant de colonne.
* parametres est un tableau de paramètres qui caractérisent la colonne id.

Voici la liste des paramètres disponibles (tous ne sont pas obligatoires). Les valeur par défaut sont suivies d’une astérisque.

* order : (entier) ordre d'affichage de la colonne dans le listing. (défaut 1).
* header : (booléen) spécifie si les données de la colonne servent d’entête à la ligne de données. (true / false\*).
* libelle : (string) titre à afficher dans l’entête de la colonne. (défaut « NAME »).
* size : (entier) taille de la colonne en %. (défaut 10%).
* align : (string) alignement de la colonne (left\* / center/ right).
* title : (string) info bulle sur l’entête de la colonne.
* titlePos : (string) position de l’info bulle (top\* / right / bottom / left).
* tri : (booléen) dit si on peut trier sur cette colonne. (true / false\*).
* triSql : (string) champ SQL à prendre en compte pour l’éventuel tri de la colonne.
* triSqlSecondaire : (string) éventuel champ SQL secondaire concerné par le tri éventuel de la colonne.
* triLibelle : (string) libellé en clair du tri de la colonne. Ce texte sera affiché en entête de la liste pour écrire en clair de quel type de tri il s’agit.
* triSens : (string) sens du tri éventuel de la colonne (ASC\*, DESC).
* filtre : (booléen) indique la présence d'un filtre sur ce champ (true / false\*).
* filtreType : (string) un des trois type de filtres vus plus haut ( text\*, select, checkbox).
* filtreActif : (booléen) filtre actif ou non ? (true\* / false).
* filtreScope : étendue des données proposées par le filtre.
* filtreRange : indice du choix en cours dans le scope.
* filtreValue : (string) valeur de la recherche.
* filtreSqlField : (string) nom du champ SQL ciblé par le filtre.
* filtreCaption : (string) libellé sur le filtre (uniquement sur les filtres select et checkbox).
* filtreColor : (string) couleur du champs UniversalForm du filtre (défaut primary).
* filtreHelp : (string) libellé d’aide au survol du filtre par la souris.
* display : (booléen) affichage ou non de la colonne (true\* / false).

#### Colonne Titre

Créons notre première colonne, il s’agit d’une colonne possédant un filtre de type text :

$this->**createCol**('titre', array(

'order' => 1,

'header' => true,

'libelle' => 'Titre',

'size' => 35,

'align' => 'left',

'title' => 'Titre du film',

'tri' => true,

'triSql' => 'titre',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le titre',

'filtre' => true,

'filtreType' => 'text',

'filtreScope' => UniversalListColonne::MENU,

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => '',

'filtreSqlField' => 'titre',

'filtreColor' => 'success',

'filtreHelp' => 'Filtre sur le titre',

'dislay' => true

));

**Explications** : La colonne nommée « titre » sera visible et la première affichée de la liste construite. Elle aura comme libellé Titre, une taille de 35% et son contenu sera cadré à gauche. Elle fera référence au champ titre (trié) de la requête SQL qui se chargera d’envoyer les données. Elle possèdera un filtre de type text de couleur success (couleur prédéfinie Bootstrap) qui par défaut affichera une zone de saisie vide correspondante au choix UniversalListColonne::TOUT d’une étendue de choix possibles définie par UniversalListColonne::MENU. Au survol de l’entête de la colonne apparaîtra l’info bulle Titre du film et au survol du filtre apparaîtra l’info bulle Filtre sur le titre. Enfin cette colonne sert d’entête aux lignes de données.

Comme nous l’avons vu plus haut, un filtre text comporte une zone de saisie et une liste déroulante d’options à appliquer au filtre : ces options sont : Tous, Egal à, Différent de, Commence par, Contient, Ne contient pas, Se termine par. L’ensemble de ces choix possible sont prédéfinis dans la constante UniversalListColonne::MENU. Pour que notre filtre propose donc l’ensemble de ces choix, il suffit d’appliquer cette constante à la valeur filtreScope de notre définition de colonne.

Les valeur filtreRange et filtreValue vont quant à elles recevoir la valeur par défaut du filtre, c’est-à-dire à la première création de notre liste. filtreRange va donc recevoir le choix par défaut dans la liste déroulante d’options. Ici en entrant UniversalListColonne::TOUT et une chaine vide pour filtreValue nous spécifions que par défaut, le filtre prend en compte toutes les données, autrement dit que la colonne n’applique aucun filtre. D’autres choix sont possibles :

* UniversalListColonne::TOUT, toutes les valeurs.
* UniversalListColonne::EGAL, toutes les valeurs égales à la saisie de l’utilisateur.
* UniversalListColonne::DIFFERENT, toutes les valeurs différentes de la saisie.
* UniversalListColonne::COMMENCE, toutes les valeurs qui commencent par la saisie.
* UniversalListColonne::CONTIENT, toutes les valeurs qui contiennent la saisie.
* UniversalListColonne::CONTIENTPAS, toutes les valeurs qui ne contiennent pas la saisie.
* UniversalListColonne::FINIT, toutes les valeurs qui se terminent par la saisie.
* UniversalListColonne::IGNORE, toutes les valeurs puisque la saisie de l’utilisateur est ignorée. Attention, pour pouvoir utiliser cette option, il faut que le scope reçoivent la constante UniversalListColonne::MENU\_IGNORE à la place de UniversalListColonne::MENU.

D’autres constantes réservées aux comparaisons numériques existent :

* UniversalListColonne::SUPERIEURA, toutes les valeurs supérieures à la saisie.
* UniversalListColonne::SUPERIEUROUEGALA, toutes les valeurs supérieures ou égales à la saisie.
* UniversalListColonne::INFERIEURA, toutes les valeurs inférieures à la saisie.
* UniversalListColonne::INFERIEUROUEGALA, toutes les valeurs inférieures ou égales à la saisie.
* UniversalListColonne:: EGALA, toutes les valeurs égales à la saisie

Et une autre un peu différente :

* UniversalListColonne::COMMENCENUM, toutes les valeurs qui commencent par un chiffre.

#### Colonne Année

La colonne suivante est très simple puisque on ne souhaite pas lui appliquer de filtre. Elle doit cependant permettre le tri que le champ SQL annee.

$this->**createCol**('annee', array(

'order' => 2,

'libelle' => 'Année',

'size' => 10,

'align' => 'center',

'title' => 'Année de production',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'annee',

'triSqlSecondaire' => '',

'triLibelle' => 'sur l\'année de production',

'filtre' => false,

'filtreType' => '',

'filtreActif' => '',

'filtreScope' => '',

'filtreRange' => '',

'filtreValue' => '',

'filtreSqlField' => '',

'filtreColor' => '',

'filtreHelp' => '',

'display' => true

));

On peux aussi l’écrire de manière plus simple :

$this->**createCol**('annee', array(

'order' => 2,

'libelle' => 'Année',

'size' => 10,

'align' => 'center',

'title' => 'Année de production',

'tri' => true,

'triSql' => 'annee',

'triLibelle' => 'sur l\'année de production'

));

#### Colonne Réalisateur

Encore plus simple puisque on ne fera aucun tri ni aucun filtre sur cette colonne.

$this->**createCol**('real', array(

'order' => 3,

'libelle' => 'Réalisateur',

'size' => 25,

'align' => 'left',

'title' => 'Réalisateur du film'

'tri' => false,

'triSens' => '',

'triSql' => '',

'triSqlSecondaire' => '',

'triLibelle' => '',

'filtre' => false,

'filtreType' => '',

'filtreActif' => '',

'filtreScope' => '',

'filtreRange' => '',

'filtreValue' => '',

'filtreSqlField' => '',

'filtreColor' => '',

'filtreHelp' => '',

'display' => true

));

Où plus simplement :

$this->**createCol**('real', array(

'order' => 3,

'libelle' => 'Réalisateur',

'size' => 25,

'title' => 'Réalisateur du film'

));

#### Colonne Visuel

La colonne visuel va afficher 1 pour les films en couleur et 0 pour les films en noir & blanc. La colonne pourra être triée et on ajoutera un filtre de type checkbox (binaire) qui - si la boite à cocher est cochée - devra n’afficher QUE les films en couleur.

$this->**createCol**('visuel', array(

'order' => 4,

'libelle' => '<span class="fa fa-television"></span>',

'size' => 5,

'align' => 'center',

'title' => 'En couleur',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'visuel',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le visuel couleur',

'filtre' => true,

'filtreType' => 'checkbox',

'filtreScope' => array(UniversalListColonne::TOUT, 1),

'filtreRange' => UniversalListColonne::EGAL,

'filtreValue' => UniversalListColonne::TOUT,

'filtreSqlField' => 'visuel',

'filtreCaption' => '<span class="fa fa-television"></span>',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre sur le visuel du film (couleur ?)',

'display' => true

));

Dans le cas d’un filtre de type checkbox, filtreScope doit recevoir un tableau de deux valeurs : la première est l’information que doit retourner la liste si la case à cocher **N’EST PAS** cochée. La deuxième est la valeur que doit retourner la liste si la case à cocher **EST** cochée.

Comme nous l’avons vu plus haut, l’information filtreRange doit recevoir le type de recherche. S’agissant d’une case à cocher, une seule possibilité : UniversalListColonne::EGAL. En effet ce type de filtre ne peut que produire un filtre de recherche à l’égalité. Toute autre entrée ne sera pas prise en compte. On pourra même ne pas renseigner cette entrée.

Enfin filtreValue reçoit la valeur par défaut du filtre, c’est-à-dire celle fournie lors de la première création de la liste. Ici toutes les valeurs, faisant référence à la première entrée du tableau fourni à filtreScope : dans notre cas UniversalListColonne::TOUT. Noter que l’on aurait tout aussi bien pu fournir à filtreScope le tableau suivant : array('tout', 'couleur') et à filtreValue : tout. Pour être tout à fait précis, le tableau filtreScope renseigne - dans l’ordre - les paramètres valueInverse et value d’une boite à cocher UniversalForm.

#### Colonne Genre

Enfin, notre cinquième et dernière colonne possède un filtre de type select dans lequel on pourra choisir parmi plusieurs genres cinématographiques :

$this->**createCol**('genre', array(

'order' => 5,

'libelle' => 'Genre',

'size' => 25,

'align' => 'left',

'title' => 'Genre du film',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'genre',

'triLibelle' => 'sur le genre de films',

'filtre' => true,

'filtreType' => 'select',

'filtreScope' => 'fillGenres',

'filtreRange' => UniversalListColonne::EGAL,

'filtreValue' => 'TOUS',

'filtreSqlField' => 'genre',

'filtreCaption' => 'Genre',

'filtreColor' => 'primary',

'filtreHelp' => 'Filtre sur le genre',

'display' => true

));

Dans le cas d’un filtre de type select, filtreScope (étendue des choix disponibles) droit recevoir le nom de la fonction de **callback** à appliquer pour remplir la liste déroulante. Ainsi il va falloir écrire une fonction nommée fillGenres qui va se charger de cela. Voici ce à quoi elle peut ressembler.

function **fillGenres**($defaut) {

$genres = array('TOUS','Western','Science-fiction','Drame','Comédie musicale','Horreur','Comédie');

$texte = '';

foreach($genres as $genre) {

($defaut == $genre) ? $selected = ' selected' : $selected = '';

$texte.= '<option value="'.$genre.'"'.$selected.'>'.$genre.'</option>';

}

return $texte;

}

Cette fonction de **callback** doit créer le code HTML des <option>s d’une liste déroulante <select>. C’est ni plus ni moins que le paramètre complement que l’on doit fournir à un champ UniversalFieldSelect de la classe UniversalForm. Au choix on pourra implémenter cette fonction dans n’importe quel script qui sera appelé par l’application.

Tout comme le filtre de type checkbox, le filtre de type select ne permet **QUE** la recherche à l’égalité. Il n’est à cet égard pas même nécessaire de lui donner une valeur puisque non pris en compte. Mais pour la forme entrons UniversalListColonne::EGAL.

#### Construction des colonnes

Toutes les colonnes de notre liste sont construites. On remarquera qu’en ne renseignant QUE les paramètres nécessaires, la déclaration de la méthode protégée construitColonnes() devient plus simple :

protected function **construitColonnes**() {

$this->**createCol**('titre', array(

'order' => 1,

'libelle' => 'Titre',

'size' => 35,

'title' => 'Titre du film',

'tri' => true,

'triSql' => 'titre',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le titre',

'filtre' => true,

'filtreType' => 'text',

'filtreScope' => UniversalListColonne::MENU,

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => '',

'filtreSqlField' => 'titre',

'filtreColor' => 'success',

'filtreHelp' => 'Filtre sur le titre'

));

$this->**createCol**('annee', array(

'order' => 2,

'libelle' => 'Année',

'size' => 10,

'align' => 'center',

'title' => 'Année de production',

'tri' => true,

'triSql' => 'annee',

'triLibelle' => 'sur l\'année de production'

));

$this->**createCol**('real', array(

'order' => 3,

'libelle' => 'Réalisateur',

'size' => 25,

'title' => 'Réalisateur du film'

));

$this->**createCol**('visuel', array(

'order' => 4,

'libelle' => '<span class="fa fa-television"></span>',

'size' => 5,

'align' => 'center',

'title' => 'En couleur',

'tri' => true,

'triSql' => 'visuel',

'triLibelle' => 'sur le visuel couleur',

'filtre' => true,

'filtreType' => 'checkbox',

'filtreScope' => array(**UniversalListColonne::TOUT**, 1),

'filtreValue' => **UniversalListColonne**::**TOUT**,

'filtreSqlField' => 'visuel',

'filtreCaption' => '<span class="fa fa-television"></span>',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre sur le visuel du film (couleur ?)',

));

$this->**createCol**('genre', array(

'order' => 5,

'libelle' => 'Genre',

'size' => 25,

'title' => 'Genre du film',

'tri' => true,

'triSql' => 'genre',

'triLibelle' => 'sur le genre de films',

'filtre' => true,

'filtreType' => 'select',

'filtreScope' => '**fillGenres**',

'filtreValue' => 'TOUS',

'filtreSqlField' => 'genre',

'filtreCaption' => 'Genre',

'filtreHelp' => 'Filtre sur le genre',

));

}

### Remplissage des colonnes par les données

De la même manière que nous l’avons fait pour construire un listing simple, il faut maintenant définir comment les données vont venir s’afficher dans les colonnes de notre listing. Là encore, sur le même principe, nous allons ajouter non pas une fonction mais une méthode publique par colonne vouée à réaliser ce travail. Chaque méthode doit avoir la syntaxe suivante :

public function **Col\_[id\_de\_la\_colonne]**(**array** $ligne)

La méthode publique de notre classe Listing\_film destinée à afficher le contenu de la colonne titre va donc devoir s’écrire ainsi :

public function **Col\_titre**($ligne) {

echo $ligne['titre'] ;

}

Ainsi, le code nécessaire au remplissage de chaque colonne est le suivant :

public **function Col\_titre**($ligne) {

echo $ligne['titre'] ;

}

public function **Col\_annee**($ligne) {

echo $ligne['annee'] ;

}

public function **Col\_real**($ligne) {

echo $ligne['realisateur'] ;

}

public function **Col\_visuel**($ligne) {

echo $ligne['visuel'] ;

}

public function **Col\_genre**($ligne) {

echo $ligne['genre'] ;

}

On remarquera que le paramètre page qui était présent et utile pour les listings simples n’est ici plus nécessaire, c’est l’objet UniversalList qui va entièrement gérer les pages.

### Récupération des données.

Pour récupérer les données à afficher depuis la base de données, il est nécessaire de faire appel à deux requêtes. Une première, chargée de renvoyer le nombre total de tuples pour notre liste, et une seconde, chargée de renvoyer UNIQUEMENT les tuples qui correspondent à la page en cours. Chacune de ces requête va devoir être implémentée dans une fonction ou méthode particulière qu’il va falloir écrire. Il y a deux façons différentes d’implémenter ce code :

1. La première consiste à créer deux fonctions externes à la manière de ce que nous avons fait pour les listings simples.
2. La deuxième consiste à implémenter deux nouvelles méthodes chargées de faire le travail dans notre classe exemple\_listing\_films.

#### Méthode classique : fonctions externes

Peu de différences avec les fonctions créées pour les listings simples. La différence réside surtout dans les paramètres passés, mais aussi elle intègre les filtres qui seront appliqués par l’utilisateur. Voici un exemple de fonctions qui peuvent être employées pour notre listing :

function **getListeNombre**($id\_listing) {

$requete = "SELECT count(\*) nombre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $\_SESSION[$id\_listing]->**buildFiltres**();

$res = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($res !== false) {

if ($nombre != 0) {

return $res[0]['nombre'];

}

else return $nombre;

}

return false;

}

function **getListe**($id\_listing, $start, $nb\_lignes, &$laListe) {

$requete = "SELECT titre, annee, realisateur, genre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $\_SESSION[$id\_listing]->**buildFiltres**();

$requete.= "ORDER BY ".$\_SESSION[$id\_listing]->**buildTris**()." ";

if ($nb\_lignes != 0)

$requete.= "LIMIT ".$start.", ".$nb\_lignes;

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $nombre;

}

$laListe = null;

return false;

}

**NB** : Noter dans la requête la clauses WHERE 1. Celle-ci est nécessaire car la méthode buildFiltres() ne fait que rajouter des clauses AND. C’est du SQL !

La fonction getListingNombre() (ou tout autre nom que vous voudrez lui donner, c’est ici un exemple) est chargée de renvoyer le nombre de lignes totales qui vont être ramenées sur notre liste en prenant en compte les filtres positionnés par l’utilisateur. En cela elle fait appel à la méthode buildFiltres() de la classe UniversalList. Cette méthode construit automatiquement le code SQL qui correspond aux filtres positionnés par l’utilisateur.

**Syntaxe**

function **votre\_fonction**(**string** $id\_listing)

**Avec**

* id\_listing : l’identifiant de la liste que nous mettrons en œuvre très bientôt.

La fonction getListe() (ou tout autre nom que vous voudrez lui donner, c’est ici un exemple) est chargée de récupérer (seulement) les données à afficher sur la page en cours tout en prenant en compte les filtres positionnés par l’utilisateur. En cela elle fait également appel à la méthode buildFiltres() de la classe UniversalList qui construit automatiquement le code SQL des filtres sélectionnés mais également à la méthode buildTris() qui construit le code SQL correspondant au tris des tuples au regard des choix réalisés par l’utilisateur.

**Syntaxe**

function **votre\_fonction**(**string** $id\_listing, **int** $start, **int** $nb\_lignes, **array** &$laListe)

**Avec**

* id\_listing : l’identifiant du listing que nous mettrons en œuvre très bientôt.
* start : l’indice de la données qui doit être affichée en première ligne de la page en cours.
* nb\_lignes : le nombre de lignes à afficher par page.
* laListe : le tableau en retour qui va contenir les tuples à afficher sur la page en cours.

En reprenant à notre compte l’affichage du navigateur de pages vu plus haut lorsque nous avons étudié les listings simples (classe PageNavigator), le code pour la récupération des données peut alors et dans notre exemple être celui-ci :

//lecture du nombre total de lignes pour le listing

//$totalLignes = **getListeNombre**(\_ID\_LISTING\_);

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), 2, $\_SESSION[\_ID\_LISTING\_]->**getPageEncours**());

$debut = $pn->**getItemDebut**();

$pn->setPageOff();

$navigation = $pn->**draw**();

//récupération des tuples qui correspondent àp la page à afficher

$nbLignes = **getListe**(\_ID\_LISTING\_, $debut, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), $listing);

#### Méthode objet

De cette deuxième façon (notre préférée), nous allons implémenter le même code SQL dans deux nouvelles méthodes de notre classe Exemple\_listing\_films dérivée de UniversalList. Ceci va nous offrir quelques avantages, dont un code plus compact… mais aussi un tout petit inconvénient : il est impératif de nommer ces méthodes publiques getListeNombre() et getListe().

protected function **getListeNombre**() {

$requete = "SELECT count(\*) nombre ";

$requete.= "FROM films ";  
 $requete.= "WHERE 1 ";  
 $requete.= $this->**buildFiltres**();

$res = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($res !== false) {

if ($nombre != 0) {

return $res[0]['nombre'];

}

else return $nombre;

}

return false;

}

protected function **getListe**() {

$laListe = array();

$requete = "SELECT titre, annee, realisateur, genre ";

$requete.= "FROM films ";  
 $requete.= "WHERE 1 ";  
 $requete.= $this->**buildFiltres**();

$requete.= "ORDER BY ".$this->**buildTris**()." ";

if ($this->**getNbLinesParPage**() != 0)

$requete.= "LIMIT ".$this->**getSqlLimitStart**().", ".$this->**getNbLinesParPage**();

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $laListe;

}

return false;

}

On voit que ces méthodes font appel à quelques autres méthodes directement héritées de la classe Listing que nous détaillerons maintenant :

* buildFiltres() : génère le code SQL qui correspond aux filtres activés par l’utilisateur.
* buildtris() : génère le code SQL qui correspond au tri de la liste activé par l’utilisateur.
* getNbLinesParPage() : renvoie le nombre de lignes à afficher par page.
* getSqlLimitstart() : donne l’indice du premier tuple des données à afficher sur la première ligne de la page en cours du listing.

Le gros avantage de cette méthode, c’est qu’il suffit de faire simplement appel à la méthode héritée getData() pour récupérer les données. En effet getData() se charge elle-même de faire appel aux méthodes getListeNombre() et getListe() développées dans notre classe. Ainsi il ne sera jamais nécessaire de les appeler directement. D’ailleurs, ces dernières étant des classes protégées (mot clé protected) il n’est pas possible de les appeler en dehors du code de leur classe (ou classes dérivées).

Toujours avec la mise en œuvre d’un navigateur de pages, voici le code correspondant à notre récupération de données :

//lecture du nombre total de lignes pour le listing

$totalLignes = $\_SESSION[\_ID\_LISTING\_]->**getData**($listing);

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), 2, $\_SESSION[\_ID\_LISTING\_]->**getPageEncours**());

$pn->setPageOff();

$navigation = $pn->**draw**();

On remarquera tout de suite qu’il n’est plus nécessaire de construire le navigateur de page avant de charger les données à afficher. L’ordre n’a d’ailleurs lui non plus pas d’importance.

Voici le code complet de notre classe Exemple\_listing\_films.

<?php

/\*------------------------------------------------------------------

Classe Exemple\_listing\_films

------------------------------------------------------------------

éè : pour enregistrement UTF-8

Auteur : Fabrice Labrousse

Date : 30 novembre 2017

------------------------------------------------------------------\*/

//----------------------------------------

// fonction callback dont la responsabilité est de remplir

// la liste déroulante du filtre 'select' pour la colonne 'genres'

// Entrée :

// defaut : l'indice de la liste déroulante sélectionné

// Retour :

// le code HTML

//----------------------------------------

function **fillGenres**($defaut)

{

$genres = array('TOUS','Western','Science-fiction','Drame','Comédie musicale','Horreur','Comédie');

$texte = '';

foreach($genres as $genre) {

($defaut == $genre) ? $selected = ' selected' : $selected = '';

$texte.= '<option value="'.$genre.'"'.$selected.'>'.$genre.'</option>';

}

return $texte;

}

class Exemple\_listing\_films extends UniversalList {

//=======================================

// Méthode protégées, c'est à dire uniquement

// utilisable par les classes dérivées

//=======================================

//-------------------------------------

// Construction des colonnes du listing

//-------------------------------------

protected function **construitColonnes**() {

$this->**createCol**('titre', array(

'order' => 1,

'libelle' => 'Titre',

'size' => 35,

'align' => 'left',

'title' => 'Titre du film',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'titre',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le titre',

'filtre' => true,

'filtreType' => 'text',

'filtreActif' => 'true',

'filtreScope' => UniversalListColonne::MENU,

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => '',

'filtreSqlField' => 'titre',

'filtreColor' => 'success',

'filtreHelp' => 'Filtre sur le titre'

));

$this->**createCol**('annee', array(

'order' => 2,

'libelle' => 'Année',

'size' => 10,

'align' => 'center',

'title' => 'Année de production',

'tri' => true,

'triSql' => 'annee',

'triLibelle' => 'sur l\'année de production'

));

$this->**createCol**('real', array(

'order' => 3,

'libelle' => 'Réalisateur',

'size' => 25,

'title' => 'Réalisateur du film'

));

$this->**createCol**('visuel', array(

'order' => 4,

'libelle' => '<span class="fa fa-television"></span>',

'size' => 5,

'align' => 'center',

'title' => 'En couleur',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'visuel',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le visuel couleur',

'filtre' => true,

'filtreType' => 'checkbox',

'filtreScope' => array(**UniversalListColonne::TOUT**, 1),

'filtreValue' => UniversalListColonne::TOUT,

'filtreSqlField' => 'visuel',

'filtreCaption' => '<span class="fa fa-television"></span>',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre sur le visuel du film (couleur ?)',

'display' => true

));

$this->**createCol**('genre', array(

'order' => 5,

'libelle' => 'Genre',

'size' => 25,

'align' => 'left',

'title' => 'Genre du film',

'tri' => true,

'triSens' => 'ASC',

'triSql' => 'genre',

'triLibelle' => 'sur le genre de films',

'filtre' => true,

'filtreType' => 'select',

'filtreScope' => '**fillGenres**',

'filtreValue' => 'TOUS',

'filtreSqlField' => 'genre',

'filtreCaption' => 'Genre',

'filtreColor' => 'primary',

'filtreHelp' => 'Filtre sur le genre',

'display' => true

));

}

//-------------------------------------

// Récupération des données du listing

//-------------------------------------

protected function **getListeNombre**() {

$requete = "SELECT count(\*) nombre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $this->**buildFiltres**();

$res = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($res !== false) {

if ($nombre != 0) {

return $res[0]['nombre'];

}

else return $nombre;

}

return false;

}

protected function **getListe**() {

$laListe = array();

$requete = "SELECT titre, annee, realisateur, visuel, genre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $this->**buildFiltres**();

$requete.= "ORDER BY ".$this->**buildTris**()." ";

if ($this->**getNbLinesParPage**() != 0)

$requete.= "LIMIT ".$this->**getSqlLimitStart**().", ".$this->**getNbLinesParPage**();

DEBUG\_('$requete', $requete);

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $laListe;

}

return false;

}

//=======================================

// Méthode publiques

//=======================================

public function **Col\_titre**($ligne) {

echo $ligne['titre'] ;

}

public function **Col\_annee**($ligne) {

echo $ligne['annee'] ;

}

public function **Col\_real**($ligne) {

echo $ligne['realisateur'] ;

}

public function **Col\_visuel**($ligne) {

echo $ligne['visuel'] ;

}

public function **Col\_genre**($ligne) {

echo $ligne['genre'] ;

}

}

## Page principale de notre listing

Nous venons de terminer la classe Exemple\_listing\_films qui hérite de la classe standard UniversalList pour la conception de notre liste. Nous allons à présent créer la page HTML principale qui va afficher notre travail : exemple\_liste\_complexe.php.

Nous allons reprendre la même base de code que pour les listing simples avec 2 endroits où insérer notre code, avant la balise <body> (après les commentaires « Code personnalisé préparatoire à écrire ici ») dans le corps du script (après les commentaires « Code personnalisé de la page commence ici »).

Dans la partie préparatoire, avant la balise <body> nous allons commencer par donner un identifiant unique à notre listing, ce qui consiste à préparer une définition de type string. Ici nous nommerons notre listing 'lstFilms'.

//-------------------------------------

//Id unique du listing

//-------------------------------------

defined('\_ID\_LISTING\_') || define('\_ID\_LISTING\_', 'lstFilms');

Cet identifiant unique sera particulièrement utile car notre liste, sera comme on la vu, une instanciation de notre classe Exemple\_listing\_films qu’il faut conserver d’une page à l’autre via une variable de session. Voici le code pour l’instanciation de notre listing de films :

//-------------------------------------

// Construction du listing

//-------------------------------------

if ((!isset($\_SESSION[\_ID\_LISTING\_])) || ($\_SESSION[\_ID\_LISTING\_] == null)) {

$\_SESSION[\_ID\_LISTING\_] = new **Exemple\_listing\_films**();

$\_SESSION[\_ID\_LISTING\_]->**setNbLinesParPage**(5);

//on cache les boutons OK et RAZ du formulaire

$\_SESSION[\_ID\_LISTING\_]->**showFormButtons**(false);

}

Tout de suite après on initialise le nombre de lignes souhaitées par page en faisant appel à la méthode setNbLinesParPages() et ici en lui donnant la valeur 5. Si cette action est omise, par défaut la page affichera un maximum de 25 lignes (notons que si l’on souhaite afficher 1 seule et unique page, il faut passer comme valeur 0). On pourra aussi dire à notre objet de ne pas afficher les boutons standards de validation de filtres en les remplaçant par des boutons un peu plus discrets. C’est ce que fait la méthode showFormButtons().

//on souhaite trier par défaut sur la colonne année de la plus grande à la plus petite

//$\_SESSION[\_ID\_LISTING\_]->**setTriEncours**('annee');

//$\_SESSION[\_ID\_LISTING\_]->**setTriSensEncours**('DESC');

En faisant appel aux méthodes setTriEncours() et setTriSensEncours() on peut accessoirement définir sur quelle colonne et comment trier notre liste à sa création. Si ces informations sont omises, le tri par défaut est systématiquement réalisé sur la première colonne que l’on peut trier. Dans notre exemple ci-dessus, le tri par défaut se fera sur l’année et les films seront affichés du plus récent au plus ancien.

//modification de la classe CSS pour l'entete de la liste

$\_SESSION[\_ID\_LISTING\_]->**setHeadClass**('bg-warning table-sm');

//modification de la classe CSS pour le bandeau de filtres de la liste

$\_SESSION[\_ID\_LISTING\_]->**setFiltresClass**('bg-warning');

Accessoirement on peut aussi modifier la classe CSS de notre entête de tableau (setHeadClass) et de notre bandeau de filtres (setFiltresClass).

Après la construction du listing, il nous faut maintenant appeler la méthode qui va prendre en charge sa gestion. Comme pour les listes simples, il s’agit de la méthode getParams(). Mais ici, en plus de gérer les tris de la liste, elle gère aussi les filtres activés par l’utilisateur. Noter que contrairement aux listes simples, la méthode n’accepte plus de paramètres… tant mieux !

//-------------------------------------

// Gestion du listing

//-------------------------------------

$\_SESSION[\_ID\_LISTING\_]->**getParams**();

Puis il faut faire appel aux méthodes de chargement des données et d’affichage du navigateur de pages, ce que nous avons vu un peu plus haut (ici utilisation de la méthode objet) :

//lecture du nombre total de lignes pour le listing

$totalLignes = $\_SESSION[\_ID\_LISTING\_]->**getData**($listing);

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), 2, $\_SESSION[\_ID\_LISTING\_]->**getPageEncours**());

$pn->**setPageOff**();  
$navigation = $pn->**draw**();

**Petite remarque** : Nous avons maintenant récupéré dans la variable $listing tous les tuples à afficher sur notre listing. Chaque tuple contient les champs de la base de données qui correspondent aux colonnes que nous avons défini. Par exemple :

[0] => Array

(

[titre] => 2001: l'odyssée de l'espace

[annee] => 1968

[realisateur] => Stanley Kubrick

[visuel] => 1

[genre] => Science-fiction

)

Nous pouvons si besoin ajouter pour chaque tuple un champ nommé ['line-color']. Celui-ci est reconnu par la classe UniversalList et si présent affiche la ligne dans la couleur souhaitée. Par exemple si j’ajoute à mon tuple le champ ['line-color'] => 'table-danger', toute la ligne correspondante au film **2001 : l’odyssée de l’espace** sera colorée en rouge… chouette non ?

[0] => Array

(

[titre] => 2001: l'odyssée de l'espace

[annee] => 1968

[realisateur] => Stanley Kubrick  
 [visuel] => 1

[genre] => Science-fiction  
 [line-color] => table-danger

)

Les couleurs disponibles sont par défaut celle de Bootstrap V4 : table-primary, table-secondary, table-success, table-danger, table-warning, table-info, table-light et table-dark. Bien entendu vous pouvez créer vos propres couleurs avec une dose ultra-minime de CSS. Voici le genre de code de colorisation que l’on peut ajouter (ici chaque ligne dont le champ deleted est marqué, colore la ligne en rouge. Pratique vous afficher en rouge les produits supprimés ou sans stock !)

//-------------------------------------

// Colorisation particulière des lignes à afficher

// On ajoute le champ 'line-color' aux données à afficher.

// Il contient la classe background-color de la ligne

//-------------------------------------

foreach($listing as $indice => $ligne) {

($ligne['deleted']) ? $listing[$indice]['line-color'] = 'table-danger' : $listing[$indice]['line-color'] = '';

}

Pour finir, affichons notre listing !

//affichage du nombre de lignes trouvés

//--------------------------------------

**SimpleListingHelper**::**drawTotal**($totalLignes);

echo ' / '.$\_SESSION[\_ID\_LISTING\_]->**getTriSensEncoursLibelle**().' '.$\_SESSION[\_ID\_LISTING\_]->**getTriEncoursLibelle**();

//affichage barre de navigation

//--------------------------------------

echo $navigation;

//affichage du tableau

//--------------------------------------

echo '<table class="table table-hover table-striped">';   
 //affichage des filtres en entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawFiltresColonnes**();

//affichage de l'entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawHead**();

//affichage du corps du tableau : donnees

$\_SESSION[\_ID\_LISTING\_]->**drawBody**($listing);

echo '</table>';

Vous aurez sans doute remarqué qu’il est fait appel à la méthode statique statique drawTotal() de la classe SimpleListingHelper(), non ? Et bien, s’agissant d’une classe d’aide au développement (de listes en particulier), sa méthode drawTotal() est toute à fait appropriée pour afficher le total de tuples de notre listing (à ne pas confondre avec le nombre de tuples affichés sur la page en cours).

On procède ensuite à l’affichage du listing proprement dit, là encore une table au sens HTML, agrémentées de CSSs Bootstrap.

La méthode drawFiltresColonnes() est chargée de dessiner les filtres de colonne (bandeau) présent au-dessus de chaque colonne dont on a défini un filtre. Aucun paramètre n’est nécessaire.

La méthode drawHead() est chargée de dessiner l’entête du listing (nom des colonnes et informations de tris). Elle non plus n’attend aucun paramètre.

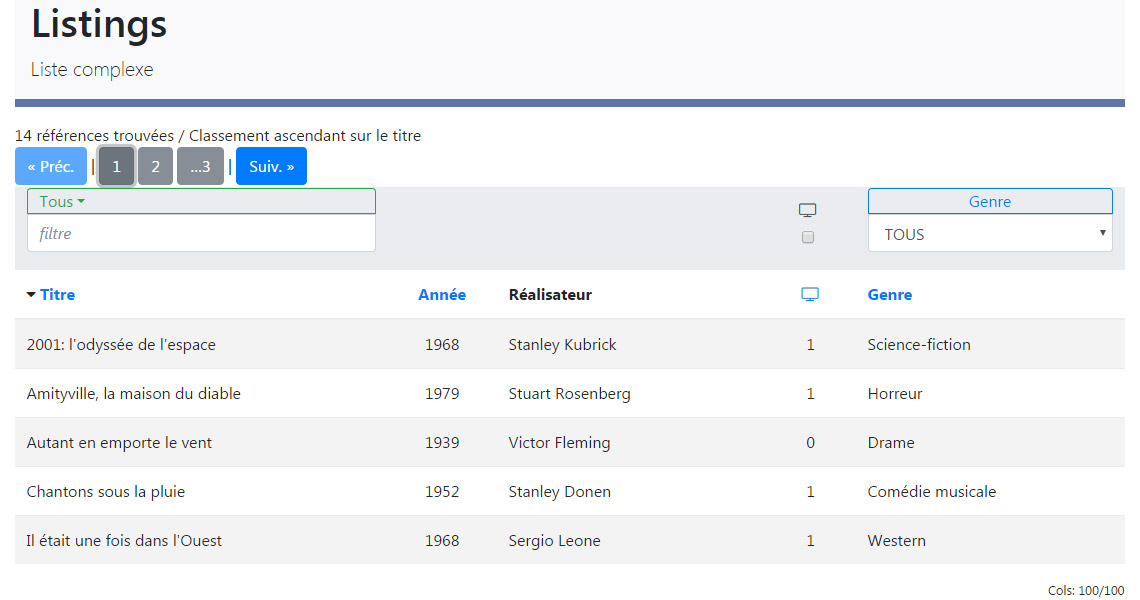
Enfin la méthode drawBody(**array** $listing) est chargée d’afficher le contenu (les données) de la liste. On lui passera en paramètre le tableau des tuples récupérés par la méthode getData() appelée plus haut.

On pourra également, en fin de tableau ajouter en information la taille totale des colonnes visibles de la liste :

//affichage de la taille totale des colonnes visibles en %

echo '<p class="small text-right">Cols: '.$\_SESSION[\_ID\_LISTING\_]->**getDisplayedSize**().'/100</p>';

Voici notre liste enfin terminée :



Figure

Et le code complet de exemple\_listing\_complexe.php. Finalement, grâce à la classe UniversalList c’est un jeu d’enfant que de créer des listes complexes non ? Notons aussi que comme pour les listes simples et de la même manière, il est possible d’ajouter une touche de javascript pour permettre à l’utilisateur de modifier à la volée la taille de chaque colonne de notre liste…

<?php

//--------------------------------------------------------------------------

// LISTING DES FILMS

//--------------------------------------------------------------------------

// ééàç : pour sauvegarde du fichier en utf-8

//--------------------------------------------------------------------------

require\_once('libs/common.inc.php');

//-------------------------------------

//Id unique du listing

//-------------------------------------

defined('\_ID\_LISTING\_') || define('\_ID\_LISTING\_', 'lstFilms');

//-------------------------------------

// Construction du listing

unset($\_SESSION[\_ID\_LISTING\_]);

//-------------------------------------

if ((!isset($\_SESSION[\_ID\_LISTING\_])) || ($\_SESSION[\_ID\_LISTING\_] == null)) {

$\_SESSION[\_ID\_LISTING\_] = new **Exemple\_listing\_films**();

$\_SESSION[\_ID\_LISTING\_]->**setNbLinesParPage**(5);

//on cache les boutons OK et RAZ du formulaire

$\_SESSION[\_ID\_LISTING\_]->**showFormButtons**(false);

}

//-------------------------------------

// Gestion du listing

//-------------------------------------

$\_SESSION[\_ID\_LISTING\_]->**getParams**();

//-------------------------------------

// Récupération des données à afficher

//-------------------------------------

//lecture du nombre total de lignes pour le listing

$totalLignes = $\_SESSION[\_ID\_LISTING\_]->**getData**($listing);

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), 2, $\_SESSION[\_ID\_LISTING\_]->**getPageEncours**());

$pn->setPageOff();

$navigation = $pn->**draw**();

//---------------------------------------------------------

// head

//---------------------------------------------------------

$titrePage = \_APP\_TITLE\_;

$scriptSup = '';

$fJquery = '';

echo writeHTMLHeader($titrePage, '', '');

//---------------------------------------------------------

// body

//---------------------------------------------------------

echo '<body>';

//---------------------------------------------------------

// Corps APPLI

//---------------------------------------------------------

echo '<div class="container">';

echo '<div class="row">';

//------------ colonne centrale ---------------

//lg et xl -> taille 10 colonnes

//en dessous prend toute la largeur de la fenetre (12 colonnes)

echo '<div class="col-12">';

//panel de l'application

echo '<section class="bg-light px-3 souligne">';

echo '<div class="row">';

//marque

echo '<div class="col-12">';

echo '<p class="h1">Listings</p>';

echo '<p><span class="lead">Liste complexe</span></p>';

echo '</div>';

echo '</div>';

echo '</section>';

//code propre à la page

echo '<article style="margin-bottom:3rem;">';

echo '<div class="row">';

echo '<div class="col-12">';

//-----------------------------------------

//Code personnalisé de la page commence ICI

//-----------------------------------------

//affichage du nombre de lignes trouvés

//--------------------------------------

**SimpleListingHelper**::**drawTotal**($totalLignes);

echo ' / '.$\_SESSION[\_ID\_LISTING\_]->**getTriSensEncoursLibelle**().' '.$\_SESSION[\_ID\_LISTING\_]->**getTriEncoursLibelle**();

//affichage barre de navigation

//--------------------------------------

echo $navigation;

//affichage du tableau

//--------------------------------------

echo '<table class="table table-hover table-striped">';

//affichage des filtres en entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawFiltresColonnes**();

//affichage de l'entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawHead**();

//affichage du corps du tableau : donnees

$\_SESSION[\_ID\_LISTING\_]->**drawBody**($listing);

echo '</table>';

//affichage de la taille totale des colonnes visibles en %

echo '<p class="small text-right">Cols: '.$\_SESSION[\_ID\_LISTING\_]->**getDisplayedSize**().'/100</p>';

echo '</div>'; //col

echo '</div>'; //row

echo '</article>';

echo '</div>';

//------ fin colonne centrale -----------

echo '</div>'; //row

//---------------------------------------------------------

// Footer APPLI

//---------------------------------------------------------

include\_once(\_BRIQUE\_FOOTER\_);

echo '</div>'; //container

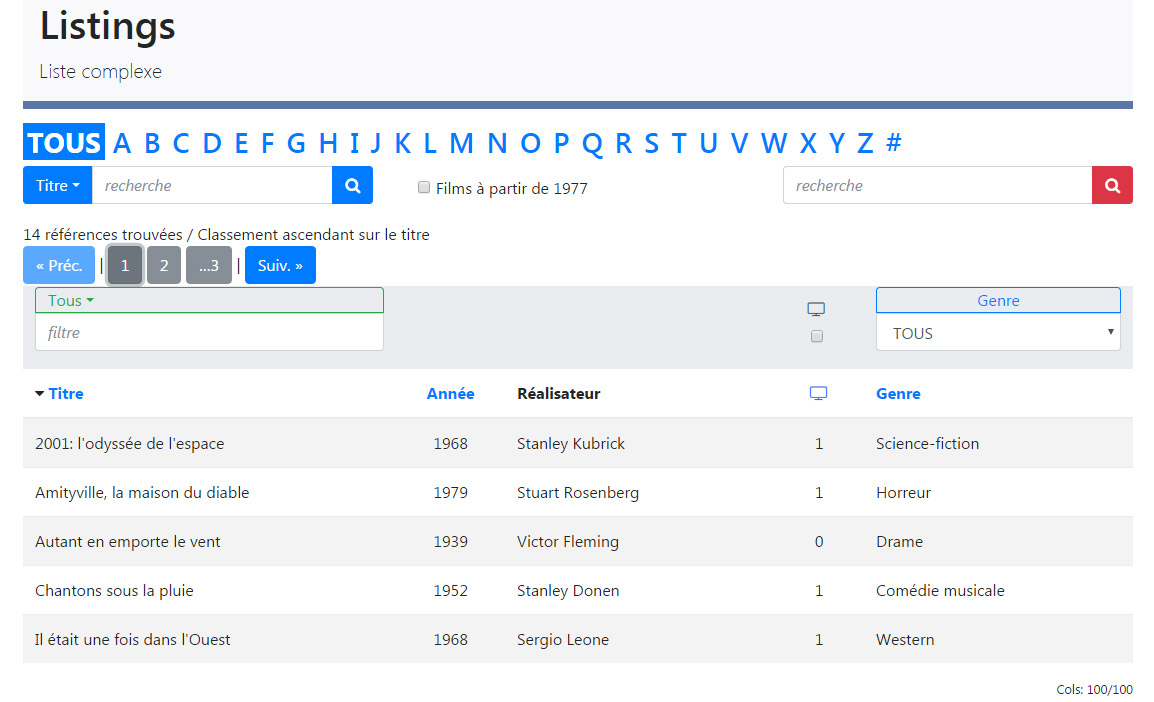
echo '</body>';  
echo '</html>';

## Filtres externes

Jusque là nous avons vu qu’il était possible de créer des listes filtrées avec des filtres positionnés sur chaque colonne de la liste. Nous allons maintenant voir que la classe UniversalList permet aussi de créer des filtres externes qui ne sont pas directement reliés à une colonne mais à l’ensemble de la liste. Pour poursuivre ce tutoriel sur le même exemple, nous nous proposons d’installer trois nouveaux filtres externes prédéfinis :

1. Un filtre de recherche simple qui sera de couleur rouge qui permettra de lancer une recherche sur le titre d’un film.
2. Un filtre de recherche à choix multiple de couleur bleue qui permettra de lancer une recherche sur le titre d’un film ou sur un genre cinématographique.
3. Une case à cocher qui permettra de n’afficher que les films dont la date de production est supérieure à 1977.

Enfin nous verrons qu’il est aussi possible d’utiliser la puissance de la classe pour créer un abécédaire qui nous permettra (par exemple) de filtrer les titres de films par leur première lettre.



Figure

### Filtre de recherche simple

Pour ajouter un filtre externe, il n’y à rien de plus simple. En gros il suffit de le paramétrer, puis de l’afficher. A la manière de ce que nous avons fait pour les colonnes (voir méthode construitColonnes), nous allons surcharger la méthode protégée construitFiltresExternes() de la classe UniversalList pour déclarer et paramétrer nos filtres externes. Voici le code à entrer dans le fichier exemple\_listing\_films.class.php.

//-------------------------------------

// construction des filtres externes

//-------------------------------------

protected function **construitFiltresExternes**() {

$this->**createFiltreExterne**('simple', array(

'filtreType' => 'search',

'filtreScope' => 'titre',

'filtreRange' => UniversalListColonne::CONTIENT,

'filtreValue' => '',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre simple'

));

}

La méthode étant protégée, son nom ne peut être modifié. Dans cette méthode, nous allons paramétrer chacun de nos filtres externes en faisant appel à la classe createFiltreExterne(). Sa syntaxe est la suivante :

**createFiltreExterne**(**string** $id, **array** $parametres)

* id est une chaine de caractères qui identifie de manière unique le filtre externe. Attention le nom donné au filtre ne peut pas être un nom déjà donné à un filtre de colonne. Nous avons choisi ici « simple ».
* parametres est un tableau contenant tous les paramètres qui vont permettre de caractériser le filtre.

Voici les paramètres qu’il faut renseigner :

* filtreType : type de filtre prédéfini. A choisir parmi « search », « multisearch », « checkbox » ou « none ». Pour un filtre de recherche simple il faut choisir « search ». Ce type de filtre présente un champ de type search au sens UniversalForm, composé d’une zone de saisie et d’un bouton de lancement de la recherche.
* filtreScope : Ce paramètre reçoit le champ SQL qui va être utilisé par la requête de récupération des données pour satisfaire au filtre choisi. Deux choses sont à noter : la première, dans notre cas, il n’est pas nécessaire de réécrire les méthodes d’appel aux données car ajouter un filtre externe ne vient modifier que la part WHERE d’une requête SQL qui est renvoyée par la méthode buildFiltres(). Ensuite, il faut garder à l’esprit qu’il est tout à fait possible de créer un filtre externe sur le même champ que celui pris en charge par un filtre de colonne. Le résultat sera alors une suite logique de AND SQL. Dans notre exemple nous avons choisi de faire un filtre de recherche simple sur le titre des films.
* filtreRange : Il s’agit de l’opérateur que le filtre va utiliser pour ramener les éléments qui correspondent à la recherche. En utilisant la constante UniversalListColonne::CONTIENTnous faisons appel à une constante connue qui signifie que **« le titre du film à ramener doit contenir le terme présent dans la zone de saisie du filtre externe appelé simple**». Pour voir l’ensemble des constantes disponibles reportez vous au chapitre 2.3.2.1.
* filtreValue : contient la valeur recherchée par défaut, c’est-à-dire la première valeur cherchée dès la construction du filtre et de la liste. Ici le paramètre est laissé vide pour signifier que l’on ne recherche rien.
* filtreColor : couleur au sens bootstrap à donner à notre filtre externe.
* filtreHelp : texte qui sera affiché au survol du filtre par la souris.

Et c’est tout ! reste maintenant à afficher notre filtre. Pour ceci, aller dans le code de la page principale et ajoutez le code suivant au-dessus du code de création de la liste :

//--------------------------------------

// Filtre externe recherche simple

//--------------------------------------

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('simple')->**afficher**();

Notre liste instanciée et passée en session fait appel à la méthode getFiltreExterne() pour pouvoir accéder aux méthodes propres au filtre. A celui-ci on fait appel à sa méthode afficher() pour renvoyer le code HTML d’affichage du filtre.

**NB** : tout comme la méthode construitColonnes() la méthode construitFiltresExternes() est appelée **une seule et unique fois** lors de la création de l’objet de classe Exemple\_listing\_films. Pour prendre en compte le nouveau filtre créé, penser à détruire l’objet en mémoire pour le recréer :

$\_SESSION[\_ID\_LISTING\_] = new Exemple\_listing\_films();

### Filtre de recherche multiple

Par multiple il faut entendre « *qui permet d’effectuer des recherches sur différents champs d’une table de la base de données* ». Un champ UniversalForm de type « search » avec addon est donc mis en œuvre pour ce type de filtre. Ajoutons le code suivant à la méthode construitFiltresExternes() :

$this->**createFiltreExterne**('recherche', array(

'filtreType' => 'multisearch',

'filtreScope' => array(

'titre' => 'Titre',

'genre' => 'Genre'),

'filtreRange' => UniversalListColonne::CONTIENT,

'filtreValue' => array('titre', ''),

'filtreColor' => 'primary',

'filtreHelp' => 'Filtre à sources multiples'

));

Le filtre est identifié « recherche ». Il est construit à partir des paramètres suivant :

* filtreType : le type de filtre. Ici multisearch (recherche multiple !)
* filtreScope : à la différence d’un filtre de recherche simple, il faut renseigner quels champs de la base SQL seront utilisé pour chaque fonctions multiple choisie par l’utilisateur. Il s’agit donc d’entrer ici non par une simple valeur mais un tableau de couples « champ SQL » => « Libellé affiché pour le filtre ». Par exemple lorsque l’utilisateur choisira l’entrée « Titre » le filtre externe cherchera une valeur sur le champ SQL « titre ». On pourrait aussi passer des champs plus complexe comme CONCAT(nom, prenom) => Nom Prénom …
* filtreRange : Il s’agit de l’opérande que le filtre va utiliser pour ramener les éléments qui correspondent à la recherche. En utilisant la constante UniversalListColonne::CONTIENTnous faisons appel à une constante connue qui signifie que **« le titre du film à ramener doit contenir le terme présent dans la zone de saisie du filtre externe appelé simple**». Pour voir l’ensemble des constantes disponibles reportez vous au chapitre 2.3.2.1.
* filtreValue : contient la valeur recherchée par défaut, c’est-à-dire la première valeur cherchée dès la construction du filtre et de la liste. Puisque le filtreScope est un tableau, il faut ici aussi renseigner le paramètre avec un tableau contenant le couple « champ SQL » => « valeur du filtre », deux infos qui – combinées – permettent de réaliser la recherche. Dans notre exemple, nous choisissons une recherche sur le champ SQL titre avec aucune valeur saisie (vide). Si l’on avait voulu initialiser notre filtre externe en recherchant d’entrée tous les westerns il aurait fallu écrire 'filtreValue' => array('genre', 'western'). **ATTENTION** : le champ SQL doit obligatoirement exister dans le paramètre filtreScope.
* filtreColor : couleur au sens bootstrap à donner à notre filtre externe. Ici bleu.
* filtreHelp : texte qui sera affiché au survol du filtre par la souris.

Pour afficher à son tour le filtre, il suffit ici aussi d’ajouter simplement le code suivant dans la page principale:

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('recherche')->**afficher**();

### Filtre case à cocher

Un filtre case à cocher est un peu différent. En effet, il n’y a pas de zone de saisie. On va lui donner un libellé et son activation devra être prise en compte via le paramètre actif. Voici dans notre exemple le code qui convient d’appliquer :

$this->**createFiltreExterne**('datation', array(  
 'filtreType' => 'checkbox',

'libelle' => 'Films à partir de 1977',

'filtreScope' => 'annee',

'filtreRange' => UniversalListColonne::SUPERIEUROUEGALA,

'filtreValue' => '1977',

'actif' => true,

));

* filtreType : Ici checkbox.
* libelle : libellé de la case à cocher qui sert de filtre. Noter que ce paramètre n’est utilisé que pour les filtres externes de type checkbox.
* filtreScope : tout comme les filtres de recherche simple search, ce paramètre ne reçoit qu’une simple valeur, c’est-à-dire le champ SQL auquel s’applique le filtre.
* filtreRange : opérateur que le filtre va utiliser pour ramener les éléments qui correspondent à la recherche. Pour voir l’ensemble des constantes disponibles reportez vous au chapitre 2.3.2.1.
* filtreValue : la valeur qui sert d’opérande dans la recherche.
* actif : booléen qui permet de donner l’état coché (true) ou non coché (false) du filtre.

Là non plus, aucune difficulté supplémentaire pour l’affichage du filter :

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('datation')->**afficher**();

### Abécédaire

Comme nous l’avons dit plus haut, il est également possible d’utiliser la puissance de la classe UniversalList pour créer des filtres tout à fait originaux comme par exemple un abécadaire. Comme tout filtre non prédéfini, il faudra alors gérer soi-même la gestion du filtre (que se passe-t-il lorsque l’utilisateur clique dessus ? Quels sont les évènements à récupérer ?). Voici à titre d’exemple (qui marche) la construction d’un abécédaire. Tout d’abord, comme tout autre type de filtre il faut le créer via l’appel à la méthode construitFiltresExternes() :

$this->**createFiltreExterne**('alpha', array(

'filtreType' => 'none',

'filtreScope' => 'titre',

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => 'tous',

));

* filtreType : Ici none. Ceci signifie que le type de filtre externe est libre.
* libelle : non utilisé dans notre exemple.
* filtreScope : tout comme les filtres de recherche search et checkbox, ce paramètre reçoit une simple valeur, c’est-à-dire le champ SQL auquel s’applique le filtre. Ici nous allons filtrer (à nouveau) sur le titre des films.
* filtreRange : opérateur que le filtre va utiliser pour ramener les éléments qui correspondent à la recherche. Pour voir l’ensemble des constantes disponibles reportez vous au chapitre 2.3.2.1.
* filtreValue : la valeur qui sert d’opérande dans la recherche.

A la lecture de ses paramètres, on comprend tout de suite que le filtre externe nommé alpha va devoir ramener tous les films dont le titre commence par « tous » : c’est-à-dire tous les films ! Du moins dans un premier temps, lors de son premier appel à la construction de l’objet de classe Exemple\_listing\_films.

On l’a dit, s’agissant d’un filtre externe dit *libre*, c’est au développeur de déterminer comment le filtre va se comporter (gestion des évènements) et comment l’afficher. Tout ceci pourra être écrit au choix dans la classe Exemple\_listing\_films ou bien dans le code de la page principale. Pour notre exemple nous allons écrire cela dans la page principale de notre liste.

#### Gestion du filtre

Tout d’abord, voyons la gestion du filtre, ou comment prendre en compte ses interactions avec l’utilisateur. Nous avons décidé de créer un abécédaire, donc il s’agit d’autant de liens href que de lettres de l’alphabet, plus une entrée « tous » (tel que le filtre a été initialisé) et une entrées pour les titres qui commencent par un chiffre comme **2001, l’odyssée de l’espace**. On pourra écrire le code nécessaire à sa gestion immédiatement après la méthode getParams() qui gère tous les autres filtres de cette liste … et il commence à y en avoir pas mal. Notons qu’il serait aussi possible de créer notre propre classe de gestion qui dans un premier temps appellerait la classe parente getParams(). A vous de voir… Voici le code proposé pour la gestion de notre abécédaire :

// Traitement spécifique pour l'abécédaire

//-------------------------------------

(isset($\_GET['do'])) ? $do = **MySQLDataProtect**($\_GET['do']) : $do = 'aucun';

if ($do != 'aucun') {

if ($do == 'tous') {

//tout les titres

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(UniversalListColonne::TOUT);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

elseif ($do == '0') {

//commence par un chiffre

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(UniversalListColonne::COMMENCENUM);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

else {

//commence par une lettre

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(UniversalListColonne::COMMENCE);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

}

**Explications** : Chaque fois que l’utilisateur clique sur une lettre de l’alphabet nous allons rappeler le même script (notre page de code) en passant en paramètre le code de la lettre cliquée. La gestion de l’évènement se limite donc seulement à la récupération de la lettre cliquée sur la ligne de commande via un $\_GET. Selon l’information récupérée on va paramétrer le range et la valeur notre filtre externe. Pour un clic sur « tous » il faudra simplement lui envoyer les infos de range UniversalListColonne::TOUT et de valeur « tous ». Pour un titre qui commence par un chiffre, le range UniversalListColonne::COMMENCENUM et la valeur « 0 » feront l’affaire. Enfin pour toutes les autres lettre, il faudra passer le range UniversalListColonne::COMMENCE et la lettre en question recherchée.

#### Affichage du filtre

L’affichage de notre abécédaire interviendra plus bas.

echo '<p class="h3">';

//affichage menu

//recupération de la valeur du filtre 'alpha' en cours

$lettre = $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**getValue**();

//affichage

//TOUS

($lettre == 'tous') ? $chaine='<span class="font-weight-bold bg-primary text-white mr-1 px-1">TOUS</span>' : $chaine = '<span class="mr-1 px-1">TOUS</span>';

echo '<a class="d-inline-block" href="'.$\_SERVER['PHP\_SELF'].'?operation=filtreAlpha&do=tous">'.$chaine.'</a>';

//LETTRES ALPHABET

for ($i='A'; $i!='AA'; $i++) {

($lettre == $i) ? $chaine='<span class="font-weight-bold bg-primary text-white mr-1 px-1">'.$i.'</span>' : $chaine = '<span class="mr-1 px-1">'.$i.'</span>';

echo '<a class="d-inline-block" href="'.$\_SERVER['PHP\_SELF'].'?operation=filtreAlpha&do='.$i.'">'.$chaine.'</a>';

}

//CHIFFRES

($lettre == '0') ? $chaine='<span class="font-weight-bold bg-primary text-white px-1">#</span>' : $chaine = '<span class="px-1">#</span>';

echo '<a class="d-inline-block" href="'.$\_SERVER['PHP\_SELF'].'?operation=filtreAlpha&do=0">'.$chaine.'</a>';

echo '</p>';

# Annexes

## Modifications

V2.0.0 : Encapsulation du formulaire des filtres de colonnes. Moins de code.

V2.0.1 : changement du nom de la méthode protégée buildColonne() en construitColonne().

V2.1.0 : Encapsulation de filtres externes commun de recherche search, multisearch, checkbox.

V2.2.0 :

* Les méthodes suivantes ont été renommées :
  + sens() en triSens(),
  + getSens() en getTriSens(),
  + setSens() en setTriSens(),
  + getSensEncoursLibelle() en getTriSensEncoursLibelle().
* Création du paramètre de colonne titlePos qui positionne le title (top/right/bottom/left).

V2.3.0 :

* Les méthodes getCols() et setCols() redeviennent publiques et non privées.
* Ajout des méthodes publiques cols() et col() pour récupérer les objets UniversalListColonne de la liste.
* Ajout de la constante NB\_LIGNES\_PAR\_PAGE qui renvoie le nombre de lignes max par page par défaut de la classe.
* Ajout de la constante SHOW\_BUTTONS qui renvoie le type de boutons de validation par défaut de la classe.
* Création de la méthode getShowButtonsState() qui renvoie le style de boutons du formulaire (true : boutons standards, false : boutons simplifiés).

V2.4.0 :

* Ajout de la méthode publique setTriEncours() modification du tri en cours.
* Ajout de la méthode publique setTriSensEncours() modification du sens du tri en cours.
* Ajout de la méthode publique getTriEncours() qui renvoie le champ trié en cours.
* Ajout de la méthode publique getTriSensEncours() qui renvoie le champ trié en cours.

V2.5.0 :

* Ajout de la méthode publique setFiltreValueDefault() pour la classe UniversalListColonne qui positionne la valeur par défaut du filtre de la colonne.
* Ajout de la méthode publique colExist() pour la classe UniversalList qui permet de savoir si une colonne de la liste existe (renvoie true / false).

V2.5.1 :

* Correction de la méthode **createTable**() pour corriger la structure de la table Listing. tinyint(3) à la place de tinyint(3).

V2.6.0 :

* Ajout des méthodes publiques **setHeadClass**() et **getHeadClass**() pour modifier la classe CSS de l'entête du tableau
* Ajout des méthodes publiques **setFiltresClass**() et **getFiltresClass**() pour modifier la classe CSS du bandeau de filtres du tableau (valeur défaut "thead-light")
* Ajout du paramètre "header" dans la construction de la colonne de la table : true (la colonne est une entête pour la ligne), false sinon (valeur par défaut). Ce paramètre indique si la donnée de la colonne doit servir d'entête pour la ligne.
* Affichage du listing : supprimé la taille de la colonne pour permettre à du code javascript de la modifier en drag & drop (voir code https://www.brainbell.com/javascript/making-resizable-table-js.html)

## Méthodes publiques de la classe UniversalList

|  |
| --- |
| setTriEncours |
| V2.4.0  Défini la colonne de tri par défaut.  **Description**  **setTriEncours**(**string** $id)  **Liste des paramètres**  **id**  Identifiant de la colonne choisie.  **Retour**  **Aucun.** |

|  |
| --- |
| setTriSensEncours |
| V2.4.0  Défini le sens du tri par défaut.  **Description**  **setTriSensEncours**(**string** $sens)  **Liste des paramètres**  **sens**  constante texte ‘ASC’ pour un tri ascendant ou ‘DESC’ pour un tri descendant.  **Retour**  **Aucun.** |

|  |
| --- |
| setNbLinesParPage |
| Positionne le nombre de lignes à afficher par page  **Description**  **setNbLinesParPage**(**int** $valeur)  **Liste des paramètres**  **valeur**  **Nombre de lignes maximum à afficher sur une page du listing.**  **Si** valeur **vaut 0 alors la liste affichera tous les tuples sur une seule page.**  **Retour**  **Aucun.** |

|  |
| --- |
| setHeadClass |
| Positionne la classe CSS pour l’entête de la liste.  **Description**  **setHeadClass**(**string** $valeur)  **Liste des paramètres**  **valeur**  **CSS de personnalisation de l’entête de la liste.**  **Retour**  **Aucun.** |

|  |
| --- |
| setFiltresClass |
| Positionne la classe CSS pour le bandeau de filtres de la liste.  **Description**  **setFiltresClass**(**string** $valeur)  **Liste des paramètres**  **valeur**  **CSS de personnalisation du bandeau de filtres de la liste.**  **Retour**  **Aucun.** |

|  |
| --- |
| setPageEncours |
| Positionne le numéro de la page en cours de la liste.  **Description**  **setPageEncours**(**int** $valeur)  **Liste des paramètres**  **valeur**  **Numéro de la page à positionner.**  **Retour**  **Aucun.**  **Attention, ceci ne modifie pas le paginateur, mais donne simplement l’information à la classe qui l’utilisera elle-même pour sa propre gestion et qui se chargera de l’affichage du paginateur et des informations de la liste.** |

|  |
| --- |
| getFiltresExternes |
| Renvoie le tableau des filtres externes de la liste. Chaque élément du tableau est un objet de la classe UniversalListFiltreExterne.  **Description**  array **getFiltresExternes**()  **Liste des paramètres**  **Aucun**  **Retour**  **Un tableau d’objets** UniversalListFiltreExterne. |

|  |
| --- |
| getFiltreExterne |
| Renvoie l’objet de la classe UniversalListFiltreExterne qui correspond à un filtre externe.  **Description**  UniversalListFiltreExterne **getFiltreExterne**(**string** $id)  **Liste des paramètres**  **id**  **identifiant du filtre externe à retrouver.**  **Retour**  **Un objets** UniversalListFiltreExterne. |

|  |
| --- |
| getHeadClass |
| Renvoie le code CSS affecté à l’entête de la liste.  **Description**  string **getHeadClass**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le code CSS activé pour personnaliser l’entête de la liste.** |

|  |
| --- |
| getFiltresClass |
| Renvoie le code CSS affecté au bandeau de filtres de la liste.  **Description**  string **getFiltresClass**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le code CSS activé pour personnaliser le bandeau de filtres de la liste.** |

|  |
| --- |
| getPageEncours |
| Renvoie la page en cours de la liste.  **Description**  int **getPageEncours**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le numéro de la page en cours.** |

|  |
| --- |
| getTriEncours |
| Renvoie l’identifiant de la colonne sur laquelle la liste est actuellement tirée.  **Description**  string **getTriEncours** ()  **Liste des paramètres**  **Aucun**  **Retour**  **id**  Identifiant de la colonne. |

|  |
| --- |
| getTriEncoursLibelle |
| Renvoie le libellé en clair de la colonne actuellement triée.  **Description**  string **getTriEncoursLibelle**()  **Liste des paramètres**  **Aucun**  **Retour**  **Renvoie ni plus ni moins que le champ «**triLibelle**» utilisé lors de la construction de la colonne.** |

|  |
| --- |
| getTriSensEncours |
| Renvoie le sens de tri en cours.  **Description**  string **getTriSensEncours**()  **Liste des paramètres**  **Aucun**  **Retour**  **Renvoie la chaine « ASC » (c**lassement ascendant) ou **« DESC » (classement descendant).** |

|  |
| --- |
| getTriSensEncoursLibelle |
| Renvoie le libellé en clair du sens d’affichage du tri en cours.  **Description**  string **getTriSensEncoursLibelle**()  **Liste des paramètres**  **Aucun**  **Retour**  **Renvoie la chaine «**Classement ascendant » pour un tri **ascendant ou « Classement descendant » pour un tri descendant.** |

|  |
| --- |
| getNbLinesParPage |
| Renvoie le nombre de lignes affichées par page.  **Description**  int **getNbLinesParPage**()  **Liste des paramètres**  **Aucun**  **Retour**  **Le nombre de lignes affichées par page.** |

|  |
| --- |
| getSqlLimitStart |
| Renvoie l’indice (dans la table SQL) de la première ligne à afficher sur la page.  **Description**  int **getSqlLimitStart**()  **Liste des paramètres**  **Aucun**  **Retour**  **Indice dans le flow d’enregistrements de celui qui doit être affiché en haut de page.**  **Valeur que l’on doit passer à une requête SQL pour signifier la première ligne à ramener. Concrètement, c’est la valeur à passer à l’instruction SQL** LIMIT**. On se sert de cette information pour que SQL ne ramène QUE les lignes qui seront affichées sur la page (gain de performances)** |

|  |
| --- |
| getShowButtonState |
| Renvoie l'état des boutons du formulaire.  **Description**  booléen **getShowButtonState** ()  **Liste des paramètres**  **Aucun**  **Retour**  true : boutons standards  false : boutons simplifiés |

|  |
| --- |
| colExist |
| V2.5.0  Renseigne sur l’existence de la colonne dont l’identifiant est id.  **Description**  booléen **colExist**(id)  **Liste des paramètres**  **id**  **Identifiant de la colonne recherchée.**  **Retour**  true : la colonne existe dans la liste  false : la colonne n’existe pas dans la liste |

|  |
| --- |
| createFiltreExterne |
| Ajout d'un filtre externe au tableau de filtres externes.  **Description**  **createFiltreExterne**(**string** $nomFiltre, **array** $filtre)  **Liste des paramètres**  **nomFiltre**  **Identifiant du filtre externe.**  **filtre**  **Tableau contenant les paramètres de caractérisation du filtre.**  **Les paramètres à renseigner sont :**   * filtreType : search, multisearch, checkbox, none   (constante indiquant le type de filtre)   * filtreScope :   valeur ou tableau contenant les champs SQL à prendre en compte pour le filtre.   * filtreRange :   Opérateur de la recherche (constante issue de la classe UniversalListColonne) (UniversalListColonne::TOUT/EGAL/DIFFERENT/COMMENCE/… etc)   * filtreValue :   valeur de la recherche sur la colonne concernée.   * filtreColor :   Couleur bootstrap du filtre lorsqu’applicable.   * filtreHelp :   Texte qui apparaît au survol du filtre par la souris.   * libelle :   Libellé du filtre (en particulier pour les filtres de type checkbox).   * actif :   positionne l’activité du filtre. Dans le cas d’un filtre externe de type checkbox, permet de cocher (true) ou de décocher (false) la case.  **Retour**  Aucun  **Exemple**  Soit la construction du filtre externe identifié ‘search’ :  $this->**createFiltreExterne**('recherche', array(  'filtreType' => 'multisearch',  'filtreScope' => array(  'CONCAT (T1.nom, T1.prenom)' => 'Nom',  'poste' => 'Poste',  'matricule' => 'Matricule',  'service' => 'Service',  'fonction' => 'Fonction',  'statut' => 'Statut',  'piece' => 'Pièce'),  'filtreRange' => UniversalListColonne::CONTIENT,  'filtreValue' => array('CONCAT (T1.nom, T1.prenom)', ''),  'filtreColor' => 'primary',  'filtreHelp' => ''  )); |

|  |
| --- |
| createCol |
| Création d’une nouvelle colonne à ajouter au listing.  **Description**  **createCol**(**string** $id, **array** $tabInfos)  **Liste des paramètres**  **id**  **Identifiant de la colonne.**  **tabInfos**  **Tableau contenant les paramètres de caractérisation de la colonne.**  **Les paramètres à renseigner sont :**  order**: (int) Ordre d'affichage de la colonne dans le listing. (défaut 1)**  header**: (booléen) Positionne le contenu de la colonne comme entête des données de la ligne (**true / false\***). Recommandé 0 ou 1 seul** header **par liste.**  libelle**: (string) Titre a afficher en entête de la colonne. (défaut NAME)**  size**: (int) Taille de la colonne en % (défaut 10%)**  align**: (string) Alignement de la colonne (left\* / center/ right)**  title**: (string) Title info-bulle sur la colonne.**  titlePos : (string) position de l’info-bulle (top\* / right / bottom / left).  tri**: (booléen) Détermine si la colonne est triable (true / false\*)**  triSens**: (string) Sens du tri de la colonne (ASC\* / DESC)**  triSql**: (string) Champ SQL (ou groupe de champs) concerné par le tri de la colonne.**  triSqlSecondaire**: (string) Champ SQL secondaire concerné par le tri de la colonne.**  triLibelle**: (string) Libellé en clair du tri défini de la colonne. Sert pour l’affichage.**  filtre**: (booléen) Indique la présence d'un filtre sur ce champ (true / false\*)**  filtreType**: (string) Type de filtre (text\* / select)**  filtreScope**: Reçoit l’étendue de recherches possible. Différent selon le type de filtre.**  filtreRange**: (string) Opérateur de la recherche sur la colonne.**  filtreValue**: (string) Valeur de la recherche sur la colonne.**  filtreSqlField**: (string) Nom du champ SQL ciblé par le filtre.**  filtreCaption **: (string) libellé du filtre (vide par défaut).**  filtreColor**: couleur bootstrap du filtre (primary\*).**  filtreHelp**: libellé affiché au survol du filtre par la souris.**  display**: (booléen) Affichage de la colonne (true\* / false)**  **\*Valeurs par défaut**  **Retour**  Aucun  **Exemple**  Soit la construction de la colonne identifiée ‘titre’ :  $this->createCol('titre', array(  'order' => 1,  'header' => true,  'libelle' => 'Titre',  'size' => 35,  'align' => 'left',  'title' => 'Titre du film',  'titlePos' => 'right',  'tri' => true,  'triSens' => 'ASC',  'triSql' => 'titre',  'triSqlSecondaire' => '',  'triLibelle' => 'sur le titre',  'filtre' => true,  'filtreType' => 'text',  'filtreActif' => 'true',  'filtreScope' => UniversalListColonne::MENU,  'filtreRange' => UniversalListColonne::TOUT,  'filtreValue' => '',  'filtreSqlField' => 'titre',  'filtreColor' => 'success',  'filtreHelp' => 'Filtre sur le titre'  ));  Les caractéristiques de cette colonne sont :   * Première colonne affichée. * Cette colonne sert d’entête pour les données de la ligne (la représentation des données est davantage marquée). * Affiche « Titre » dans son entête. * Sa largeur est de 35%. * Elle est triable. * Le champ SQL primaire utilisé pour le tri est titre. * Il n’y a pas de champ secondaire pour générer ce tri. * Lorsque la colonne sera triée il sera possible d’afficher le texte « sur le titre » via la méthode getTriEncoursLibelle(). * Au survol de la souris sur l’entête de la colonne, apparaîtra l’info bulle cadrée à droite « Titre du film ». * La colonne possède un filtre de colonne. * Le filtre de colonne est de type text. * Le filtres est actif. * Le scope utilise le menu tout prêt UniversalListColonne::MENU de la classe UniversalListColonne. Il est composé des éléments ainsi :   const MENU = array( self::TOUT => 'Tous',  self::EGAL => 'Egal à',  self::DIFFERENT => 'Différent de',  self::COMMENCE => 'Commence par',  self::CONTIENT => 'Contient',  self::CONTIENTPAS => 'Ne contient pas',  self::FINIT => 'Se termine par');   * La valeur du filtre est vide. * Le champ SQL utilisé pour le filtre est titre. * Le filtre est de couleur verte. * Le texte « Filtre sur le titre » est affiché au survol du filtre par la souris. |

|  |
| --- |
| cols |
| V2.3.0 Renvoie la structure des colonnes de la liste en cours.  **Description**  array **cols**()  **Liste des paramètres**  **Aucun**  **Retour**  Renvoie un tableau contenant pour chaque colonne un objet de type UniversalListColonne qui la caractérise. Toutes les méthodes sont alors applications à chacun des objets contenus dans ce tableau.  **NB** : *Ne pas confondre cette méthode avec la méthode getCols() qui renvoie la structure des colonnes de la liste en cours sous forme de chaîne compressée.* |

|  |
| --- |
| col |
| V2.3.0 Renvoie la structure d’une colonne de la liste en cours.  **Description**  UniversalListColonne **col**(string $id)  **Liste des paramètres**  **id**  **identifiant de la colonne.**  **Retour**  Renvoie un objet de la classe UniversalListColonne. |

|  |
| --- |
| showFormButtons |
| Affiche ou cache les boutons standard de validation ou de réinitialisation des filtres de colonne.  **Description**  **showFormButtons**(**boolean** $valeur)  **Liste des paramètres**  **valeur**  true**: affiche les boutons standards.**  false**: affiche les boutons simplifiés.**  **Retour**  Aucun |

|  |
| --- |
| getShowButtonsState |
| V2.3.0  Renvoie le mode d’affichage des boutons de validation ou de réinitialisation des filtres de colonne.  **Description**  boolean **showFormButtons**()  **Liste des paramètres**  **Aucun**  **Retour**  true**: la liste affiche les boutons standards.**  false**: la liste affiche les boutons simplifiés.** |

|  |
| --- |
| debugCols |
| Affiche des informations de debug des colonnes du listing.  **Description**  **debugCols**()  **Liste des paramètres**  **Aucun**  **Retour**  Aucun  Affiche les objets colonne du listing. |

|  |
| --- |
| getSize |
| Renvoie la taille totale du listing en pourcentage.  **Description**  int **getSize**()  **Liste des paramètres**  **Aucun**  **Retour**  Taille totale en pourcentage.  La taille totale correspond à la somme de toutes les colonnes, qu’elles soient visible et affichées ou pas. |

|  |
| --- |
| getDisplayedSize |
| Renvoie la taille totale du listing en pourcentage en ne prenant en compte que les colonne affichées.  **Description**  Int **getDisplayedSize**()  **Liste des paramètres**  **Aucun**  **Retour**  Taille totale en pourcentage.  La taille totale correspond à la somme de toutes les colonnes affichées. |

|  |
| --- |
| getFiltres |
| Renvoie un tableau contenant des informations sur les filtres de colonne du listing.  **Description**  Array **getFiltres**()  **Liste des paramètres**  **Aucun**  **Retour**  Tableau des filtres de colonne.  Les informations sont renvoyées dans un tableau indexé par l’identifiant de la colonne. Il renseigne sur l’état du filtre, le scope en cours du filtre, et la valeur de filtrage en cours.  $filtres = (tableau) Array  (  [titre] => Array  (  [actif] => 1  [range] => tout  [value] =>  )  [genre] => Array  (  [actif] => 1  [range] => egal  [value] => Science-fiction  )  ) |

|  |
| --- |
| getParams |
| Cette méthode gère le listing en prenant en compte ses informations interactives. En particulier les informations de tri et de page, mais aussi les évènement issus des filtres externes.  **Description**  **getParams**()  **Liste des paramètres**  **Aucun**  **Retour**  Aucun.  C’est le cœur de la classe UniversalList. La méthode vient entre autres, lire les informations présentes en GET qui sont passées pour faire vivre le listing (tri, page en cours, etc.). Elle est obligatoire pour que la gestion du listing soit cohérente. Elle vient aussi prendre en compte les évènements des filtres externes (lecture du POST). |

|  |
| --- |
| drawFiltresColonnes |
| Dessine les filtres de la liste au dessus de l’entête.  **Description**  string **drawHeadColonnes**()  **Liste des paramètres**  **Aucun**  **Retour**  Code HTML d’affichage des filtres de colonnes.  Renvoie le code HTML d’affichage des filtres de la liste. |

|  |
| --- |
| drawHead |
| Dessine l’entête de la liste.  **Description**  string **drawHead** ()  **Liste des paramètres**  **Aucun**  **Retour**  Code HTML d’affichage de l’entête de la liste.  Renvoie le code HTML d’affichage de l’entête du listing.  Ne pas confondre l’entête du tableau qui comporte les noms des colonnes avec l’entête des filtres de colonnes qui contient les filtres à disposition de l’utilisateur sur chaque colonne filtrable. |

|  |
| --- |
| drawBody |
| Dessine le corps (les données) de la liste.  **Description**  string **drawBody**(**array** $listing)  **Liste des paramètres**  **listing**  **Tableau contenant pour chaque colonne les informations à afficher.**  **Retour**  Code HTML d’affichage du corps de la liste.  **Exemple**  $listing = (tableau) Array  (  [0] => Array  (  [titre] => 2001: l'odyssée de l'espace  [annee] => 1968  [realisateur] => Stanley Kubrick  [genre] => Science-fiction  )  [1] => Array  (  [titre] => L'Empire contre-attaque  [annee] => 1980  [realisateur] => Irvin Kershner  [genre] => Science-fiction  )  ) |

|  |
| --- |
| getCols |
| V2.3.0 Renvoie la structure des colonnes de la liste sous une forme compressée afin d’être éventuellement stockée en base de données.  **Description**  string **getCols**()  **Liste des paramètres**  **Aucun**  **Retour**  Chaine de caractère encodée.  Le tableau des objets UniversalListColonne qui caractérise la liste est compressé puis renvoyé sous forme de chaîne. Cette chaine peut alors être sauvegardée en base de données. La compression consiste en une sérialisation du tableau d’objets puis en un encodage en base 64.  **NB** : *Ne pas confondre cette méthode avec la méthode cols() qui renvoie la structure des colonnes de la liste en cours.* |

|  |
| --- |
| setCols |
| V2.3.0 Affecte à la structure des colonnes de la liste une structure sous une forme compressée qui peut provenir d’une base de données. Les filtres de colonnes sont automatiquement mis à jour.  **Description**  **setCols**($data)  **Liste des paramètres**  **data**  **chaine de caractère encodée 64 et sérialisée qui contient la structure des colonnes d’une liste. Cette valeur aura été fournie par l’appel préalable de la méthode** getCols()**.**  **Retour**  Aucun. |

|  |
| --- |
| saveList |
| Sauvegarde une liste en base de données  **Description**  int **saveList(string** $titre**, string** $id\_user**, string** $id\_listing**)**  **Liste des paramètres**  **titre**  **titre de la liste.**  **id\_user**  **Identifiant de la personne qui a demandé la sauvegarde de la liste .**  **id\_listing**  **Identifiant de la liste. Cet identifiant servira à retrouver la liste**  **Retour**  **Identifiant unique de la sauvegarde de la liste. Il pourra être réutilisé pour venir la recharger.**  **Exemple :**  $\_SESSION[\_ID\_LISTING\_]->**saveList**('Titres qui contiennent le mot pluie', 'fabrice', 'lstFilms');  **La liste (et tous les filtres de colonnes) est stockée dans l’état ou elle se trouve au moment de la sauvegarde. Attention, seule la conception de la liste et des colonnes sont stockées, pas les lignes qui la composent. Cette méthode permet donc de rappeler facilement des listes triées et filtrées préalablement.**  **La liste est sauvegardée dans une table nommée «**listings**» qu’il faut éventuellement créer en utilisant la méthode** createTable()**;**  **NB : Ne sauvegarde pas les filtres externes.** |

|  |
| --- |
| loadList |
| Recharge une une liste depuis la base de données  **Description**  boolean **loadList(int** $id**)**  **Liste des paramètres**  **Id**  **Identifiant unique de la sauvegarde.**  **Retour**  true**: tout s’est bien passé**  false**: erreur SQL**  **Exemple :**  $\_SESSION[\_ID\_LISTING\_]->**loadList**(20);  **La liste (et tous les filtres de colonnes) sont réaffichés tels qu’ils étaient au moment de leur stockage. Attention, seule la conception de la liste et des colonnes sont restitués, pas les lignes qui la composent.**  **NB : les filtres externes ne sont pas restitués.** |

|  |
| --- |
| isFiltreActif |
| Renseigne si une colonne filtrée est active (présence d’un filtre)  **Description**  boolean **isFiltreActif**(**string** $id)  **Liste des paramètres**  **id**  **Identifiant de la colonne dont on veux obtenir l’information.**  **Retour**  True : le filtre id est actif.  False : le filtre id n’est pas actif.  **Le filtre est dit actif si il s’agit bien d’un filtre (colonne filtrable), qu’il soit opérationnel (ne veux pas forcément dire utilisé) et à condition qu’il soit affiché.** |

|  |
| --- |
| isAnyFiltreEnCours |
| Renseigne si il existe au moins 1 filtre de colonne en cours d’utilisation.  **Description**  boolean **isAnyFiltreEnCours**()  **Liste des paramètres**  **Aucun**  **Retour**  true : Au moins 1 filtre de colonne est actif dans la présentation des résultats.  false : Aucun filtre de colonne n’est impliqué dans la présentation des résultats.  **Concrètement, un filtre en cours d’utilisation est un filtre dont les paramètres de filtrage (range et valeur) ne sont pas ceux définis par défaut lors de la construction de la colonne.** |

|  |
| --- |
| isColonneActive |
| Renseigne si une colonne est active.  **Description**  boolean **isColonneActive**(**string** $id)  **Liste des paramètres**  **id**  **Identifiant de la colonne dont on veux obtenir l’information.**  **Retour**  true : La colonne id est active.  false : La colonne id n’est pas active.  **Pour être active, une colonne doit être "affichée" ET ("sans filtre" OU "possède un filtre actif")** |

|  |
| --- |
| buildFiltres |
| Construction du code SQL de filtrage  **Description**  string **buildFiltres**()  **Liste des paramètres**  **Aucun**  **Retour**  Code SQL.  **Construit le code SQL en fonction des filtres en cours. (**WHERE **…)** |

|  |
| --- |
| buildTris |
| Construction du code SQL de tri  **Description**  string **buildTris**()  **Liste des paramètres**  **Aucun**  **Retour**  Code SQL.  **Construit le code SQL en fonction du tri en cours. (**ORDER BY **…)** |

|  |
| --- |
| setFiltre |
| Positionne un filtre de colonne  **Description**  **setFiltre**(**string** $colonne, **string** $range, **string** $value)  **Liste des paramètres**  **colonne**  **Identifiant de la colonne**  **range**  **Nouveau range à passer au filtre de la colonne**  **value**  **Nouvelle valeur à passer au filtre de la colonne**  **Retour**  Aucun |

|  |
| --- |
| setFiltreExterne |
| Positionne un filtre externe  **Description**  **setFiltreExterne**(**string** $filtreId, **string** $range, **string** $value)  **Liste des paramètres**  **filtreId**  **Identifiant du filtre externe.**  **range**  **Nouveau range à passer au filtre.**  **value**  **Nouvelle valeur à passer au filtre.**  **Retour**  Aucun  Cette méthode revient à appeler les méthode setFiltreRange() et setFiltreValue() de la classe UniversalFiltreExterne et en plus positionne la page en cours à 1.  **Attention** : ne pas utiliser cette méthode sur les filtres de type checkbox. En effet on ne peut pas changer la valeur d’un filtre externe de type checkbox. Seul la valeur actif sera prise en compte (voir méthode UniversalFiltreExterne::setActif()). |

|  |
| --- |
| filtresInit |
| Initialisation des filtres de colonne de la liste.  **Description**  **filtresInit**()  **Liste des paramètres**  **Aucun**  **Retour**  Aucun |

|  |
| --- |
| filtresUpdate |
| Met à jour les filtres en fonction des choix fait par l’utilisateur.  **Description**  **filtresUpdate**(**array** $donnees)  **Liste des paramètres**  **donnees**  **Données issues du formulaire des filtres sous forme de tableau. Le tableau doit comporter une entrée par identifiant de filtre.**   * **Pour les filtres de type ‘text’, l’information renvoyée est un tableau composé d’un couple de valeurs avec les clés suivantes :**   **Indice 0 => le range du filtre**  **Indice 1 => la valeur du filtre**   * **Pour les filtres de type ‘select’, l’information renvoyée est directement la valeur du**   **filtre.**  **Le tableau peut aussi contenir des autres champs, comme les boutons du formulaire, mais ces informations ne sont pas nécessaire à la mises à jour des filtres puisque comme son nom l’indique la méthode est (seulement) chargée de mettre à jour les filtres. Ci-dessous un exemple de données que l’on fournir à la méthode** filtresUpdate()**.**  $donnees = (tableau) Array  (  [titre] => Array  (  [0] => tout  [1] =>  )  [genre] => Science-fiction  [bouton\_ok] => 1  [bouton\_raz] => notposted  )  **Informations supplémentaire**   * Si la valeur du filtre est UniversalListColonne::IGNORE, on désactive le filtre et on l'active dans le cas contraire. * Dans le cas d’un filtre de type text, l’indice 0 renvoyé par le paramètre données (issu du formulaire de filtres) correspond au range du filtre * Dans le cas d’un filtre de type text, l’indice 1 renvoyé par le paramètre données (issu du formulaire de filtres) correspond à la valeur du filtre   **Retour**  Aucun |

|  |
| --- |
| getData |
| Appel aux méthodes de récupération des données du listing.  **Description**  int **getData**(**array** &$laListe)  **Liste des paramètres**  **laListe**  **Tableau recevant en retour la liste des tuples de la base de données à afficher dans chaque ligne du listing.**  **Retour**  **Le nombre total de lignes pour le tableau si Ok, false sinon.**  **Pour récupérer les données de la base de données qui correspondent aux valeurs à afficher dans la page en cours du listing, il est nécessaire de faire appel à deux requêtes SQL, l’une pour récupérer le nombre total de tuples, l’autre pour en obtenir la liste. Ces deux requêtes sont :**   * **soit réalisées soit par deux fonctions externes dont les noms sont laissés à la discrétion du développeur,** * **soit réalisées par deux méthodes protégées intégrées à la classe du Listing.**   **Dans ce dernier cas, le développeur doit développer deux classes obligatoirement nommées** getListeNombre() **et** getListe()**.** getData() **se charge d’appeler ces deux classes de manière transparente en leurs fournissant les paramètres demandés. Ainsi la lecture et l’utilisation de la récupération de données est beaucoup plus simple, l’appel se fait par la seule méthode** getData() **sans se soucier des paramètres à fournir aux requêtes SQL.**  **Pour poursuivre notre exemple de listing de films, voici l’implémentation de ces ceux classes :**  **protected function getListeNombre() {**  **$requete = "SELECT count(\*) nombre ";**  **$requete.= "FROM films ";**  **$requete.= "WHERE 1 ";**  **$requete.= $this->buildFiltres();**  **$res = executeQuery($requete, $nombre, \_SQL\_MODE\_);**  **if ($res !== false) {**  **if ($nombre != 0) {**  **return $res[0]['nombre'];**  **}**  **else return $nombre;**  **}**  **return false;**  **}**  **protected function getListe() {**  **$laListe = array();**  **$requete = "SELECT titre, annee, realisateur, genre ";**  **$requete.= "FROM films ";**  **$requete.= "WHERE 1 ";**  **$requete.= $this->buildFiltres();**  **$requete.= "ORDER BY ".$this->buildTris()." ";**  **if ($this->getNbLinesParPage() != 0)**  **$requete.= "LIMIT ".$this->getSqlLimitStart().", ".$this->getNbLinesParPage();**  **$laListe = executeQuery($requete, $nombre, \_SQL\_MODE\_);**  **if ($laListe !== false) {**  **return $laListe;**  **}**  **return false;**  **}** |

## Méthodes publiques de la classe UniversalListFiltreExterne

|  |
| --- |
| setActif |
| Modifie l’état actif d’un filtre externe.  **Description**  **setActif**(**boolean** $valeur)  **Liste des paramètres**  **valeur**  true**: on active le filtre. Dans le cas d’un filtre de type checkbox, ceci a pour effet de cocher la case.**  false**: on désactive le filtre. Dans le cas d’un filtre de type checkbox, ceci a pour effet de décocher la case.**  Cette méthode n’a d’intérêt que sur les filtres de type checkbox pour lesquels c’est le seul moyen de modifier l’état. |

|  |
| --- |
| setFiltreRange |
| Modifie le range d’un filtre externe.  **Description**  **setFiltreRange**(**string** $valeur)  **Liste des paramètres**  **valeur**  Nouveau range à affecter au filtre. |

|  |
| --- |
| setFiltreValue |
| Modifie la valeur d’un filtre externe.  **Description**  **setFiltreValue**(**string** $valeur)  **Liste des paramètres**  **valeur**  Nouvelle valeur à affecter au filtre. Ce peut être une valeur simple (cas d’un filtre de type search), ou un tableau (cas d’un filtre de type multisearch).  **IMPORTANT** : Ne pas utiliser cette méthode sur les filtres de type checkbox, sauf à recevoir les valeurs 0 (pour décocher) ou 1 (pour cocher). En effet les valeurs de ce type de champs checkbox ne peuvent pas être changés, c’est l’état (checked ou actif) qui doit l’être. |

|  |
| --- |
| getId |
| Renvoie l’identifiant d’un filtre externe.  **Description**  string **getId**() |

|  |
| --- |
| getValue |
| Renvoie la valeur d’un filtre externe.  **Description**  string **getValue**()  Ne pas confondre la valeur avec la valeur du filtre (voir getFiltreValue()). |

|  |
| --- |
| getFitreRange |
| Renvoie le range d’un filtre externe.  **Description**  string **getFiltreRange** () |

|  |
| --- |
| getFitreValue |
| Renvoie la valeur complète d’un filtre externe.  **Description**  mixed **getFiltreValue**()  Renvoie soit une valeur simple (cas d’un filtre de type search), soit un tableau (cas d’un filtre de type multisearch).  Dans ce dernier cas, la clé du tableau correspond au l’indice du choix fait dans le addon du filtre. 0 correspond à la première valeur, 1, la deuxième, etc. quand à la valeur elle contient la saisie, ou valeur à rechercher. exemple  Array  (  [0] => 1  [1] => western  ) |

|  |
| --- |
| getChampSql |
| Renvoie le champ SQL qui doit servir pour réaliser le filtre.  **Description**  string **getChampSql**() |

|  |
| --- |
| getActif |
| Renvoie l’état d’activité du filtre externe.  **Description**  boolean **getActif**() |

|  |
| --- |
| initValue |
| Réinitialise le range et la valeur d’un filtre externe avec ses valeurs par défaut, c’est-à-dire celles constatées à la création de la liste.  **Description**  **initValue**()  **Attention** : ne fonctionne pas pour les filtres externes de type checkbox. |

|  |
| --- |
| afficher |
| Renvoie le code HTML nécessaire à l’affichage du filtre.  **Description**  string **afficher**() |

## Code complet

Voici le code complet pour notre exemple de liste complexe dons le résultat apparaît sur la figure 10.

Classe Exemple\_listing\_films.class.php

<?php

//------------------------------------------------------------------

// Classe Exemple\_listing\_films

//------------------------------------------------------------------

// éè : pour enregistrement UTF-8

// Auteur : Fabrice Labrousse

// Date : 30 novembre 2017 - 14 mars 2019

//------------------------------------------------------------------

//----------------------------------------

// fonction callback dont la responsabilité est de remplir

// la liste déroulante du filtre 'select' pour la colonne 'genres'

// Entrée :

// defaut : l'indice de la liste déroulante sélectionné

// Retour :

// le code HTML

//----------------------------------------

function **fillGenres**($defaut)

{

$genres = array('TOUS','Western','Science-fiction','Drame','Comédie musicale','Horreur','Comédie');

$texte = '';

foreach($genres as $genre) {

($defaut == $genre) ? $selected = ' selected' : $selected = '';

$texte.= '<option value="'.$genre.'"'.$selected.'>'.$genre.'</option>';

}

return $texte;

}

class Exemple\_listing\_films extends UniversalList {

//=======================================

// Méthode protégées, c'est à dire uniquement

// utilisable par les classes dérivées

//=======================================

//-------------------------------------

// Construction des colonnes du listing

//-------------------------------------

protected function **construitColonnes**() {

$this->**createCol**('titre', array(

'order' => 1,

'header' => true,

'libelle' => 'Titre',

'size' => 35,   
 'align' => 'left',  
 'title' => 'Titre du film',

'tri' => true,   
 'triSens' => 'ASC',

'triSql' => 'titre',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le titre',

'filtre' => true,

'filtreType' => 'text',

'filtreActif' => 'true',

'filtreScope' => UniversalListColonne::MENU,

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => '',

'filtreSqlField' => 'titre',

'filtreColor' => 'success',

'filtreHelp' => 'Filtre sur le titre'

));

$this->**createCol**('annee', array(

'order' => 2,

'libelle' => 'Année',

'size' => 10,   
 'align' => 'center',  
 'title' => 'Année de production',

'tri' => true,

'triSql' => 'annee',

'triLibelle' => 'sur l\'année de production'

));

$this->**createCol**('real', array(

'order' => 3,

'libelle' => 'Réalisateur',

'size' => 25,

'title' => 'Réalisateur du film'

));

$this->**createCol**('visuel', array(

'order' => 4,

'libelle' => '<span class="fa fa-television"></span>',

'size' => 5,   
 'align' => 'center',  
 'title' => 'En couleur',

'tri' => true,   
 'triSens' => 'ASC',

'triSql' => 'visuel',

'triSqlSecondaire' => '',

'triLibelle' => 'sur le visuel couleur',

'filtre' => true,

'filtreType' => 'checkbox',

'filtreScope' => array(**UniversalListColonne::TOUT**, 1),

'filtreRange' => UniversalListColonne::EGAL,

'filtreValue' => UniversalListColonne::TOUT,

'filtreSqlField' => 'visuel',

'filtreCaption' => '<span class="fa fa-television"></span>',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre sur le visuel du film (couleur ?)',

'display' => true

));

$this->**createCol**('genre', array(

'order' => 5,

'libelle' => 'Genre',

'size' => 25,   
 'align' => 'left',  
 'title' => 'Genre du film',

'tri' => true,   
 'triSens' => 'ASC',

'triSql' => 'genre',

'triLibelle' => 'sur le genre de films',

'filtre' => true,

'filtreType' => 'select',

'filtreScope' => 'fillGenres',

'filtreRange' => UniversalListColonne::EGAL,

'filtreValue' => 'TOUS',

'filtreSqlField' => 'genre',

'filtreCaption' => 'Genre',

'filtreColor' => 'primary',

'filtreHelp' => 'Filtre sur le genre',

'display' => true

));

}

//-------------------------------------

// construction des filtres externes

//-------------------------------------

protected function **construitFiltresExternes**() {

$this->**createFiltreExterne**('recherche', array(

'filtreType' => 'multisearch',

'filtreScope' => array(

'titre' => 'Titre',

'genre' => 'Genre'),

'filtreRange' => UniversalListColonne::CONTIENT,

'filtreValue' => array('titre', ''),

'filtreColor' => 'primary',

'filtreHelp' => 'Filtre à sources multiples'

));

$this->**createFiltreExterne**('simple', array(

'filtreType' => 'search',

'filtreScope' => 'titre',

'filtreRange' => UniversalListColonne::CONTIENT,

'filtreValue' => '',

'filtreColor' => 'danger',

'filtreHelp' => 'Filtre simple'

));

$this->**createFiltreExterne**('alpha', array(

'filtreType' => 'none',

'filtreScope' => 'titre',

'filtreRange' => UniversalListColonne::TOUT,

'filtreValue' => 'tous',

));

$this->**createFiltreExterne**('datation', array(

'filtreType' => 'checkbox',

'libelle' => 'Films à partir de 1977',

'filtreScope' => 'annee',

'filtreRange' => UniversalListColonne::SUPERIEURA,

'filtreValue' => '1977',

'actif' => true,

));

}

//-------------------------------------

// Récupération des données du listing

//-------------------------------------

protected function **getListeNombre**() {

$requete = "SELECT count(\*) nombre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $this->**buildFiltres**();

$res = executeQuery($requete, $nombre, \_SQL\_MODE\_);

if ($res !== false) {

if ($nombre != 0) {

return $res[0]['nombre'];

}

else return $nombre;

}

return false;

}

protected function **getListe**() {

$laListe = array();

$requete = "SELECT titre, annee, realisateur, visuel, genre ";

$requete.= "FROM films ";

$requete.= "WHERE 1 ";

$requete.= $this->**buildFiltres**();

$requete.= "ORDER BY ".$this->**buildTris**()." ";

if ($this->**getNbLinesParPage**() != 0)

$requete.= "LIMIT ".$this->**getSqlLimitStart**().", ".$this->**getNbLinesParPage**();

$laListe = **executeQuery**($requete, $nombre, \_SQL\_MODE\_);

if ($laListe !== false) {

return $laListe;

}

return false;

}

//=======================================

// Méthode publiques

//=======================================

public function **Col\_titre**($ligne) {

echo $ligne['titre'] ;

}

public function **Col\_annee**($ligne) {

echo $ligne['annee'] ;

}

public function **Col\_real**($ligne) {

echo $ligne['realisateur'] ;

}

public function **Col\_visuel**($ligne) {

echo $ligne['visuel'] ;

}

public function **Col\_genre**($ligne) {

echo $ligne['genre'] ;

}

}

Fichier exemple\_liste\_complexe.php

<?php

//--------------------------------------------------------------------------

// LISTING DES FILMS

//--------------------------------------------------------------------------

// ééàç : pour sauvegarde du fichier en utf-8

//--------------------------------------------------------------------------

require\_once('libs/common.inc.php');

//-------------------------------------

//Id unique du listing

//-------------------------------------

defined('\_ID\_LISTING\_') || define('\_ID\_LISTING\_', 'lstFilms');

//-------------------------------------

// Construction du listing

//-------------------------------------

if ((!isset($\_SESSION[\_ID\_LISTING\_])) || ($\_SESSION[\_ID\_LISTING\_] == null)) {

$\_SESSION[\_ID\_LISTING\_] = new **Exemple\_listing\_films**();

$\_SESSION[\_ID\_LISTING\_]->**setNbLinesParPage**(5);

//on cache les boutons OK et RAZ du formulaire

$\_SESSION[\_ID\_LISTING\_]->**showFormButtons**(false);

//on souhaite trier par défaut sur la colonne année de la plus grande à la plus petite

$\_SESSION[\_ID\_LISTING\_]->**setTriEncours**('annee');

$\_SESSION[\_ID\_LISTING\_]->**setTriSensEncours**('DESC');

//modification de la classe CSS pour l'entete de la liste

$\_SESSION[\_ID\_LISTING\_]->**setHeadClass**('bg-warning table-sm');

//modification de la classe CSS pour le bandeau de filtres de la liste

$\_SESSION[\_ID\_LISTING\_]->**setFiltresClass**('bg-warning');

}

//-------------------------------------

// Gestion du listing

//-------------------------------------

$\_SESSION[\_ID\_LISTING\_]->**getParams**();

// Traitement spécifique pour l'abécédaire

//-------------------------------------

(isset($\_GET['do'])) ? $do = MySQLDataProtect($\_GET['do']) : $do = 'aucun';

if ($do != 'aucun') {

if ($do == 'tous') {

//commence par un chiffre

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(UniversalListColonne::TOUT);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

elseif ($do == '0') {

//commence par un chiffre

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(  
 UniversalListColonne::COMMENCENUM);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

else {

//commence par une lettre

$\_SESSION[\_ID\_LISTING\_]->getFiltreExterne('alpha')->setFiltreRange(UniversalListColonne::COMMENCE);

$\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**setFiltreValue**($do);

}

}

//-------------------------------------

// Récupération des données à afficher

//-------------------------------------

//lecture du nombre total de lignes pour le listing

$totalLignes = $\_SESSION[\_ID\_LISTING\_]->**getData**($listing);

//construction de la barre de navigation

$pn = new **PageNavigator**($totalLignes, $\_SESSION[\_ID\_LISTING\_]->**getNbLinesParPage**(), 2, $\_SESSION[\_ID\_LISTING\_]->**getPageEncours**());

$pn->setPageOff();

$navigation = $pn->**draw**();

//---------------------------------------------------------

// head

//---------------------------------------------------------

$titrePage = \_APP\_TITLE\_;

$scriptSup = '';

$scriptSup.= '<script>';

$scriptSup.= 'var table = document.getElementById(\'tableId\');';

$scriptSup.= 'resizableGrid(table);';

$scriptSup.= '</script>';

$fJquery = '';

echo writeHTMLHeader($titrePage, '', '');

//---------------------------------------------------------

// body

//---------------------------------------------------------

echo '<body>';

//---------------------------------------------------------

// Corps APPLI

//---------------------------------------------------------

echo '<div class="container">';

echo '<div class="row">';

//------------ colonne centrale ---------------

//lg et xl -> taille 10 colonnes

//en dessous prend toute la largeur de la fenetre (12 colonnes)

echo '<div class="col-12">';

//panel de l'application

echo '<section class="bg-light px-3 souligne">';

echo '<div class="row">';

//marque

echo '<div class="col-12">';

echo '<p class="h1">Listings</p>';

echo '<p><span class="lead">Liste complexe</span></p>';

echo '</div>';

echo '</div>';

echo '</section>';

//code propre à la page

echo '<article style="margin-bottom:3rem;">';

echo '<div class="row">';

//--------------------------------------

// Bloc ABCDAIRE

//--------------------------------------

echo '<div class="col-12">';

echo '<p class="h3">';

//affichage menu

//recupération de la valeur du filtre 'alpha' en cours

$lettre = $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('alpha')->**getValue**();

//TOUS

($lettre == 'tous') ? $chaine='<span class="font-weight-bold bg-primary text-white   
 mr-1 px-1">TOUS</span>' : $chaine = '<span class="mr-1 px-1">TOUS</span>';

echo '<a class="d-inline-block"   
 href="'.**$\_SERVER['PHP\_SELF']**.'?operation=filtreAlpha&do=tous">'.$chaine.'</a>';

//LETTRES ALPHABET

for ($i='A'; $i!='AA'; $i++) {

($lettre == $i) ? $chaine='<span class="font-weight-bold bg-primary text-white   
 mr-1 px-1">'.$i.'</span>' : $chaine = '<span class="mr-1 px-1">'.$i.'</span>';

echo '<a class="d-inline-block"   
 href="'.**$\_SERVER['PHP\_SELF']**.'?operation=filtreAlpha&do='.$i.'">'.$chaine.'</a>';

}

//CHIFFRES

($lettre == '0') ? $chaine='<span class="font-weight-bold bg-primary text-white   
 px-1">#</span>' : $chaine = '<span class="px-1">#</span>';

echo '<a class="d-inline-block"   
 href="'**.$\_SERVER['PHP\_SELF']**.'?operation=filtreAlpha&do=0">'.$chaine.'</a>';

echo '</p>';

echo '</div>';

//--------------------------------------

// Filtre externe recherche multiple

//--------------------------------------

echo '<div class="col-12 col-sm-4">';

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('recherche')->**afficher**();

echo '</div>';

//--------------------------------------

// Filtre checkbox

//--------------------------------------

echo '<div class="col-12 col-sm-4">';

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('datation')->**afficher**();

echo '</div>';

//--------------------------------------

// Filtre externe recherche simple

//--------------------------------------

echo '<div class="col-12 col-sm-4">';

echo $\_SESSION[\_ID\_LISTING\_]->**getFiltreExterne**('simple')->**afficher**();

echo '</div>';

//--------------------------------------

// Affichage de la liste

//--------------------------------------

echo '<div class="col-12">';

//affichage du nombre de lignes trouvés

//--------------------------------------

SimpleListingHelper::drawTotal($totalLignes);

echo ' / '.$\_SESSION[\_ID\_LISTING\_]->**getTriSensEncoursLibelle**().'   
 '.$\_SESSION[\_ID\_LISTING\_]->**getTriEncoursLibelle**();

//affichage barre de navigation

//--------------------------------------

echo $navigation;

//affichage du tableau

//--------------------------------------

echo '<table id="tableId" class="table table-hover table-striped">';

//affichage des filtres en entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawFiltresColonnes**();

//affichage de l'entête du tableau

$\_SESSION[\_ID\_LISTING\_]->**drawHead**();

//affichage du corps du tableau : donnees

$\_SESSION[\_ID\_LISTING\_]->**drawBody**($listing);

echo '</table>';

//affichage de la taille totale des colonnes visibles en %

echo '<p class="small text-right">Cols: '.  
 $\_SESSION[\_ID\_LISTING\_]->**getDisplayedSize**().'/100</p>';

echo '</div>'; //col

echo '</div>'; //row

echo '</article>';

echo '</div>';

//------ fin colonne centrale -----------

echo '</div>'; //row

//---------------------------------------------------------

// Footer APPLI

//---------------------------------------------------------

include\_once(\_BRIQUE\_FOOTER\_);

echo '</div>'; //container

echo '</body>';

echo '</html>';

1. Disponibles dans le squelette de développement UniversalWeb. [↑](#footnote-ref-1)