

Divers points pour réaliser votre projet (suite)

1. Gestion des erreurs

En ajoutant l'entrée suivante dans le web.config :

```
<system.web>
  <customErrors mode="On" />
</system.web>
```

Il y a automatiquement une redirection vers la page Shared.Error.cshtml.

Ensuite logger l'erreur dans la DB par exemple ...

Dans toutes les actions des Controllers par exemple :

```
try
{
    . . .
}
catch (Exception ex)
{
    WriteLog(
        ControllerContext.RouteData.Values["controller"].ToString(),
        ControllerContext.RouteData.Values["action"].ToString(),
        C,
        ex.Message.ToString());
    throw;
}
```

Et la fonction WriteLog qui effectue un insert dans la DB :

```
public static void WriteLog(string ControllerName, string ActionName, string Param, string error)
{
    try
    {
        string sLogcontent =
            "MyWebSite/" +
            ControllerName + "/" +
            ActionName + "(" +
            (Param != null ? Param : string.Empty) + ") : " +
            error;
        using (DAL.MainDAL dal = new DAL.MainDAL())
        {
            dalLog.InsertError(sLogcontent);
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

2. Session

Pour affecter un objet dans la session :

```
Session["ItemSession"] = _lst;
```

Et ensuite le récupérer :

```
(List<ObjetModel>) Session["ItemSession"]
```

Ce qui est stocké dans la session est disponible d'une page à l'autre pour autant que l'internaute n'a pas fermé son navigateur. De plus, ce sont des informations relatives à chaque internaute. Dans votre projet, si vous stocker le panier, chaque internaute aura son panier stocké dans sa propre session. Par défaut, la session expire après 20 minutes d'inactivité et est stockée dans la mémoire du serveur web.

On peut modifier cela via le web.config :

```
<system.web>
  <sessionState mode="InProc" timeout="10" />
</system.web>
```

3. DateTimePicker

Il faut rajouter jQueryUI.

Et pour une version française des libellés des dates jquery.ui.datepicker-fr.js :

```

/*! jQuery UI - v1.10.2 - 2013-03-14
* http://jqueryui.com
* Includes: jquery.ui.datepicker-fr.js
* Copyright 2013 jQuery Foundation and other contributors; Licensed MIT */
/* French initialisation for the jQuery UI date picker plugin. */
/* Written by Keith Wood (kbwood{at}iinet.com.au),
   Stéphane Nahmani (sholby@sholby.net),
   Stéphane Raimbault <stephane.raimbault@gmail.com> */

jQuery(function($){
    $.datepicker.regional['fr'] = {
        closeText: 'Fermer',
        prevText: 'Précédent',
        nextText: 'Suivant',
        currentText: 'Aujourd\'hui',
        monthNames: ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin',
            'Juillet', 'Août', 'Septembre', 'Octobre', 'Novembre', 'Décembre'],
        monthNamesShort: ['Janv.', 'Févr.', 'Mars', 'Avril', 'Mai', 'Juin',
            'Juil.', 'Août', 'Sept.', 'Oct.', 'Nov.', 'Déc.'],
        dayNames: ['Dimanche', 'Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi'],
        dayNamesShort: ['Dim.', 'Lun.', 'Mar.', 'Mer.', 'Jeu.', 'Ven.', 'Sam.'],
        dayNamesMin: ['D', 'L', 'M', 'M', 'J', 'V', 'S'],
        weekHeader: 'Sem.',
        dateFormat: 'dd/mm/yy',
        firstDay: 1,
        isRTL: false,
        showMonthAfterYear: false,
        yearSuffix: '';
    };
    $.datepicker.setDefaults($.datepicker.regional['fr']);
});

```

On rajoutera les entrées nécessaires dans le BundleConfig.cs :

```

bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include(
    "~/Scripts/jquery-ui-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryui-datepicker-fr").Include(
    "~/Scripts/jquery.ui.datepicker-fr.js"));

//css
bundles.Add(new StyleBundle("~/Content/cssjqryUi").Include(
    "~/Content/themes/base/jquery-ui.css"));

```

Dans la vue, rien de très spécial :

```

<div class="form-group">
    @Html.LabelFor(model => model.DateNais, htmlAttributes: new { @class = "control-label col-md-2" })
    <div class="col-md-10">
        @Html.TextBoxFor(model => model.DateNais, new { @class = "form-control" })
        @Html.ValidationMessageFor(model => model.DateNais, "", new { @class = "text-danger" })
    </div>
</div>

```

@section Scripts {

```

@Scripts.Render("~/bundles/jqueryval")
@Scripts.Render("~/bundles/jqueryui")
@Scripts.Render("~/bundles/jqueryui-datepicker-fr")
@Styles.Render("~/Content/cssjqryUi")

<script type="text/javascript">
    $(document).ready(function () {
        var options = $.extend(
            {},
            $.datepicker.regional['fr'],
            {
                dateFormat: "dd/mm/yy",
                changeMonth: true,
                changeYear: true,
                yearRange: "-100:+0"
            }
        );
    });

```

```

        $('#DateNaiss').datepicker(options);
    });
</script>
}

```

ATTENTION : jqueryval ne valide pas les dates au format « french ».

Soit on n'utilise pas jqueryval, soit on travaille avec une propriété « string » et notre propre validation coté serveur ...

4. DataTable

Ajouter le nuget « jquery.datatables ».

Rajouter les entrées utiles dans BundleConfig :

```

bundles.Add(new ScriptBundle("~/bundles/datatables").Include(
    "~/Scripts/DataTables/jquery.dataTables.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(
    "~/Content/bootstrap.css",
    "~/Content/site.css",
    "~/Content/DataTables/css/jquery.dataTables.css"));

```

Dans notre exemple, on travaille avec une table qui contient plus de 1500 rows (les inscriptions 2014 à l'EPHEC en Prom. Soc.).

Première solution : charger toutes les données de la DB, et laisser le plug-in gérer pour nous la pagination, la recherche, ...

L'action du controller qui renvoie tout :

```

public ActionResult DataTableLoadAll()
{
    using (DAL.MainDAL dal = new DAL.MainDAL())
    {
        return View(dal.InscriptionSelectAll().ToList());
    }
}

```

Et la vue associée

```

@model IEnumerable<WebAppDataTable.Models.InscriptionModels>

<table class="table" id="myfirstdatatable">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Section)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Matricule)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Nom)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Prenom)
            </th>
        </tr>
    </thead>
    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Section)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Matricule)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Nom)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Prenom)
            </td>
        </tr>
    }
}

```

```

        @Html.DisplayFor(modelItem => item.Prenom)
    </td>
    <td>
        @Html.ActionLink("Edit", "Edit", new { id=item.Id }) |
        @Html.ActionLink("Details", "Details", new { id=item.Id }) |
        @Html.ActionLink("Delete", "Delete", new { id=item.Id })
    </td>
</tr>
}
</table>

@section Scripts {
    @Scripts.Render("~/bundles/datatables")

    <script>
    $(document).ready(function () {
        $('#myfirstdatatable').dataTable()
    });
    </script>
}

```

Il suffit simplement d'ajouter la référence aux scripts « datatables », et dans le « document ready » appeler la fonction « datatable » sur la table des données.

ATTENTION, il faut encapsuler les colonnes de la table avec les balises « thead » :

```

<thead>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Section)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Matricule)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Nom)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Prenom)
        </th>
        <th></th>
    </tr>
</thead>

```

C'est formidable, tout fonctionne, pagination, tri, tri multiple et même la recherche.

Show entries Search:

Section	Matricule	Nom	Prenom	
Administration et gestion du personnel	PSR05949	DOMBRET	Nicolas	Edit Details Delete
Administration et gestion du personnel	PSR06387	IMPELLIZZERI	Loredana	Edit Details Delete
Administration et gestion du personnel	PSR06633	DELAYE	Tom	Edit Details Delete
Administration et gestion du personnel	PSR06651	DIALLO	Elhadj Oumar	Edit Details Delete
Administration et gestion du personnel	PSR07216	IFUFA	Alice	Edit Details Delete
Administration et gestion du personnel	PSR07311	VAN BRUSSEL	Mélanie	Edit Details Delete
Administration et gestion du personnel	PSR07542	RODRIGUES DE SOUZA	Alice	Edit Details Delete
Administration et gestion du personnel	PSR08162	WIKTORKO	Ewelina	Edit Details Delete
Administration et gestion du personnel	PSR08188	VAN DE VOORDE	Hella	Edit Details Delete
Administration et gestion du personnel	PSR08231	UMUHOZA	Charite	Edit Details Delete

Showing 1 to 10 of 1,505 entries Previous 1 2 3 4 5 ... 151 Next

MAIS toute la liste des 1500 rows est chargée dans la page html.

Imaginez ce que cela pourrait produire pour une table qui contient 50000 rows, alors que l'internaute ne souhaite réellement afficher qu'une dizaine ...

La bonne méthode, appel ajax qui charge uniquement les lignes nécessaires

Créer une simple vue (non liée à un model), y ajouter une table vide :

```
<table id="myfirstdatatable">
  <thead>
    <tr>
      <th>Section</th>
      <th>Matricule</th>
      <th>Nom</th>
      <th>Prénom</th>
      <th>Action</th>
    </tr>
  </thead>
</table>
```

Modifier l'appel de la fonction « datatable » en précisant que le chargement sera effectué coté serveur et l'url de la source ajax :

```
<script>
$(document).ready(function () {
  var dataTable = $('#myfirstdatatable').dataTable({
    // activate Ajax call
    "bServerSide": true,
    // show loader
    "bProcessing": true,
    // Ajax call
    "sAjaxSource": "/Inscription/LoadInscription"
  });
});
</script>
```

L'action doit renvoyer un json :

```
var filteredResult = from b in lst
                      select new[] { b.Section, b.Matricule, b.Nom, b.Prenom, b.Id.ToString() };
return Json(new
{
  sEcho = Request["sEcho"],
  iTotalRecords = nbttot,
  iTotalDisplayRecords = nbttot,
  aaData = filteredResult
},
JsonRequestBehavior.AllowGet);
```

On renvoie donc le nombre total de rows, le nombre total affiché (idem sauf si on veut différencier nombre total filtré du nombre total) et la liste à afficher sous forme de tableau de string.

Lorsque l'action est appelée, un grand nombre de paramètres sont passés qu'on peut obtenir dans l'objet Request :

```
Request["sEcho"] : nécessaire pour datatables
Request["iSortCol_0"] : indice de la colonne à trier
Request["sSortDir_0"] : sens du tri
Request["iDisplayLength"] : nombre de row par page
Request["iDisplayStart"] : page de départ
Request["sSearch"] : texte encodé dans le champ recherche
```

On comprend donc que pour être efficace, on va rechercher dans la DB les rows nécessaires selon la sélection (nombre de rows par page, page de départ, texte de recherche, ...).

Il faudra minimum 2 query :

- Une première pour compter le total
- Les rows demandées

L'ordre de tri est dynamique, Request["iSortCol_0"] donne l'indice de la colonne à trier et Request["sSortDir_0"] le sens du tri (« asc » ou « desc »).

Si on trie sur une colonne supplémentaire (en maintenant la touche « Shift » enfoncée), on obtiendra les indices des colonnes suivantes pour le tri dans `Request["iSortCol_1"]`, `Request["iSortCol_2"]`, ...
Idem pour le sens du tri : `Request["sSortDir_1"]`, `Request["sSortDir_2"]`, ...

Un exemple de query pour rechercher les 10 premiers éléments d'une liste triée uniquement sur la section :

```
SELECT
    X.Id ,
    X.Section ,
    X.Matricule ,
    X.Nom ,
    X.Prenom
FROM
(
    SELECT
        Id ,
        Section ,
        Matricule ,
        Nom ,
        Prenom ,
        ROW_NUMBER() OVER(ORDER BY Section asc) Ind
    FROM
        dbo.Inscription
) AS X
WHERE X.Ind BETWEEN 1 AND 10
ORDER BY Ind
```

Pour déterminer les éléments de l'ordre by on écrira par exemple ceci :

```
// liste des colonnes « triables »
string[] listCol = { "Section", "Matricule", "Nom", "Prenom" };

// liste pour stocker les éléments de l'ordre by
List<string> lstOrderByItem = new List<string>();

// bouclons sur le nombre de colonnes « triables »
for (int i = 0; i < listCol.Length; i++)
{
    if (Request["iSortCol_" + i.ToString()] != null)
    {
        // on a un ième tri

        // indice de la colonne à ajouter au tri
        int indCol = int.Parse(Request["iSortCol_" + i.ToString()]);

        // ajoutons le nom de la colonne suivi de asc ou desc à la liste des éléments de l'ordre by
        lstOrderByItem.Add(listCol[indCol] + " " + Request["sSortDir_" + i.ToString()]);
    }
    else
    {
        // on a pas de ième tri, on quitte le for, les suivants seront null aussi
        break;
    }
}
string sOrder = string.Join(",", lstOrderByItem);
sOrder contiendra par exemple : « Section asc, Nom asc », ce qui sera exploitable pour construire
notre query.
```

On peut filtrer aussi, le contenu du filtre est accessible via `Request["sSearch"]`, on rajoutera donc à notre query un where supplémentaire, par exemple :

```
if (Request["sSearch"] != string.Empty)
{
    sWhere = string.Format("WHERE Section LIKE '{0}%' OR Matricule LIKE '{0}%' OR Nom LIKE '{0}%' OR Prenom LIKE '{0}%", Request["sSearch"]);
}

```

Dans cet exemple, on filtre sur toutes les colonnes.

Finalement, on construit donc nos 2 query comme ceci :

```
// query count
string sQueryCount = string.Format("SELECT COUNT(*) FROM dbo.Inscription {0}", sWhere);

// indice de la première row à montrer [0 ... x]
int iDisplayStart = int.Parse(Request["iDisplayStart"]);

// nombre de row par page
int iDisplayLength = int.Parse(Request["iDisplayLength"]);

// query result
string sQuery = string.Format(
    @"SELECT
        X.Id ,
        X.Section ,
        X.Matricule ,
        X.Nom ,
        X.Prenom
    FROM
    (
        SELECT
            Id ,
            Section ,
            Matricule ,
            Nom ,
            Prenom ,
            ROW_NUMBER() OVER(ORDER BY {0}) Ind
        FROM
            dbo.Inscription
        {1}
    ) AS X
    WHERE X.Ind BETWEEN {2} AND {3}
    ORDER BY Ind", sOrder, sWhere, iDisplayStart + 1, iDisplayStart + iDisplayLength);
```

Remarque : sWhere et sOrder ayant été initialisés préalablement (voir point précédent).

Il reste ensuite à exécuter les query et renvoyer le résultat en Json :

```
// exécution des query en Linq To SQL
using (DAL.MainDAL dal = new DAL.MainDAL())
{
    var lstTot = dal.ExecuteQuery<int>(sQueryCount).ToList();
    nbtot = lstTot.First();
    lst = dal.ExecuteQuery<InscriptionModels>(sQuery).ToList();
}

// générer le résultat
var filteredResult = from b in lst
    select new[] {
        b.Section,
        b.Matricule,
        b.Nom,
        b.Prenom,
        string.Format("<a href='/Inscription/Edit/{0}'>Edit</a> | " +
            "<a href='/Inscription/Details/{0}'>Details</a> | " +
            "<a href='/Inscription/Delete/{0}'>Delete</a>", b.Id.ToString())
    };

// renvoyer le Json
return Json(new
{
    sEcho = Request["sEcho"],
    iTotalRecords = nbtot,
    iTotalDisplayRecords = nbtot,
    aaData = filteredResult
},
    JsonRequestBehavior.AllowGet);
```

Remarque : si on veut être plus précis, il faudrait rajouter une troisième query qui compte le total sans le filtre where pour afficher un résultat comme ceci par exemple :

Show entries

Search:

Section	Matricule	Nom	Prenom	
Comptabilité	PS900053	VRINDTS	Sylvia	Edit Details Delete
Comptabilité	PS920017	BACON	Anke	Edit Details Delete
Comptabilité	PS920198	GREGOIRE	Daniel	Edit Details Delete
Comptabilité	PSR00925	LANCKMAN	Michaël	Edit Details Delete
Comptabilité	PSR01175	KAYUMBA	Victoire	Edit Details Delete
Comptabilité	PSR02129	KUNYU LOMBE TABU		Edit Details Delete
Comptabilité	PSR02400	DIALLO	Mamadou Aliou	Edit Details Delete
Comptabilité	PSR02573	EL BAKKALI	Ikrame	Edit Details Delete
Comptabilité	PSR03025	OUSSEINI KABO	Oumarou	Edit Details Delete
Comptabilité	PSR03202	EL MOURTJI	Rachid	Edit Details Delete

Showing 1 to 10 of 466 entries (filtered from 1,505 total entries)

Previous 2 3 4 5 ... 47 Next

On aurait dans ce cas le retour Json suivant :

```
return Json(new
{
    sEcho = Request["sEcho"],
    iTotalRecords = 1505,
    iTotalDisplayRecords = 466,
    aaData = filteredResult
},
JsonRequestBehavior.AllowGet);
```

On peut ajouter ses propres filtres, par exemple pour filtrer sur section et/ou nom uniquement :

```
<thead>
    <tr>
        <th>Section</th>
        <th>Matricule</th>
        <th>Nom</th>
        <th>Prénom</th>
        <th>Id</th>
    </tr>
</thead>
<thead>
    <tr>
        <td><input type="text" id="0" class="inscription-search-input" placeholder="Search Section"></td>
        <td>&nbsp;</td>
        <td><input type="text" id="2" class="inscription-search-input" placeholder="Search Nom"></td>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
</thead>
```

On rajoutera ceci dans le script de la vue :

```
// cacher la recherche par défaut
$("#myfirstdatatable_filter").css("display", "none");

// event lorsqu'on modifie un des critères de recherche
$('.inscription-search-input').on('keyup click change', function () {
    var i = $(this).attr('id'); // getting column index
    var v = $(this).val(); // getting search input value
    dataTable.column(i).search(v).draw();
});
```


Et on traitera la recherche dans l'action comme suit :

```
if (Request["sSearch_0"] != string.Empty)
{
    sWhere = string.Format("WHERE Section LIKE '{0}%', Request["sSearch_0"]);
}
if (Request["sSearch_2"] != string.Empty)
{
    if (sWhere == string.Empty)
    {
        // rien que search sur début Nom
        sWhere = string.Format("WHERE Nom LIKE '{0}%', Request["sSearch_2"]);
    }
    else
    {
        // search sur début section et début nom
        sWhere = string.Format("{0} AND Nom LIKE '{1}%', sWhere, Request["sSearch_2"]);
    }
}
}
```

Si on accède aux données avec EntityFramework le code est plus simple et nous évite de construire la query « à la main » :

```
DALEF.EtudiantCoursesEntities context = new DALEF.EtudiantCoursesEntities();

// liste des colonnes
string[] listCol = { "Section", "Matricule", "Nom", "Prenom" };

// Liste inscription
var lst = from b in context.Inscription
select b;

// search général
if (Request["sSearch"] != string.Empty)
{
    lst = lst.Where(x =>
        x.Section.Contains(Request["sSearch"]) ||
        x.Matricule.Contains(Request["sSearch"]) ||
        x.Nom.Contains(Request["sSearch"]) ||
        x.Prenom.Contains(Request["sSearch"]));
}

// liste pour stocker les éléments de l'order by
List<string> lstOrderByItem = new List<string>();

// toutes les colonnes sont triables, donc on peut avoir au max 4 tris activés
for (int i = 0; i < listCol.Length; i++)
{
    if (Request["iSortCol_" + i.ToString()] != null)
    {
        // on a un ième tri

        // indice de la colonne à ajouter au tri
        int indCol = int.Parse(Request["iSortCol_" + i.ToString()]);
        // ajoutons le nom de la colonne suivi de asc ou desc à la liste de l'order by
        lstOrderByItem.Add(listCol[indCol] + " " + Request["sSortDir_" + i.ToString()]);
    }
    else
    {
        // on a pas de ième tri, on peut quitter le for, les suivants seront null aussi
        break;
    }
}

string sOrder = string.Join(",", lstOrderByItem);
// trier le résultat
lst = lst.OrderBy(sOrder);

// nombre total d'inscription
int nbtot = lst.Count();

// indice de la première row à montrer [0 ... x]
int iDisplayStart = int.Parse(Request["iDisplayStart"]);
```

```

// nombre de row par page
int iDisplayLength = int.Parse(Request["iDisplayLength"]);

// filtrer le résultat selon la pagination
lst = lst.Skip(iDisplayStart).Take(iDisplayLength);

// générer le résultat
var filteredResult =
    from b in lst.ToList()
    select new[] {
        b.Section,
        b.Matricule,
        b.Nom,
        b.Prenom,
        "<a href='/Inscription/Edit/' + b.Id.ToString() + '>Edit</a> | " +
        "<a href='/Inscription/Details/' + b.Id.ToString() + '>Details</a> | " +
        "<a href='/Inscription/Delete/' + b.Id.ToString() + '>Delete</a>";
return Json(new
{
    sEcho = Request["sEcho"],
    iTotalRecords = nbtot,
    iTotalDisplayRecords = nbtot,
    aaData = filteredResult
},
JsonRequestBehavior.AllowGet);

```

ATTENTION, pour pouvoir trier avec une chaîne de caractère `lst.OrderBy(sOrder)`, `sOrder` étant un string avec par exemple « Section asc, Nom desc », il faut ajouter le nugget « System.Linq.Dynamic ».