May 12, 2015

2 Approaches when rendering a list using the Bootstrap grid system

Introduction

Rendering a list of items in ASP.NET MVC inside a grid (HTML Table) is straight forward and easy. You simply but everything inside an HTML Table tag and generate table rows for each item in the list you want to render. But what to do when you want to render it as a list of rows and columns? Well, Bootstrap has a grid system which allows you to layout items as rows and columns.

I am not going to go into too much detail on the grid system, but basically it boils down to the fact that the Bootstrap grid system consists of 12 columns. You will create a `div` container with the `row` class, and inside that you can create `div` containers for the columns. Each `div` container for the columns can specify how many columns it will take up at a specific breakpoint.

So, in the example below I have set it up so that at the `medium` breakpoint, each row consists of 3 items which take up 4 columns each, to make up the total of 12 columns in a row.

```
<div class="row">
    <div class="col-md-4">
        ...
    </div>
    <div class="col-md-4">
        ...
    </div>
    <div class="col-md-4">
        ...
    </div>
</div>
<div class="row">
    <div class="col-md-4">
        ...
    </div>
    <div class="col-md-4">
        ...
    </div>
    <div class="col-md-4">
        ...
    </div>
</div>
...
```

When rendering out a list of items, the tricky part is that after very 3 items you have rendered, your should close the current row `div` and open a new one. There are 2 approaches you can follow when doing this in ASP.NET MVC, and I will discuss both below.

Approach 1: Closing and Opening new row tags

The first approach is to simply iterate over the list of items, and keep a counter for the number of items which has been rendered. Every time after I have rendered 3 items and I want to render a following item, I simply close the current `row` and and open a new one. Here is what that solution looks like:

```
@model IEnumerable<BootstrapGridLayout.Models.Article>
@using BootstrapGridLayout;
@{
    ViewBag.Title = "Home Page";
```

```
<div class="row">
@foreach (var article in Model)
{
    if (counter != 0 && counter % 3 == 0)
    {
        @:</div>
        @:<div class="row">
    }

    <div class="col-md-4">
        <div class="thumbnail">
            <img src="@article.Thumbnail" data-holder-rendered="true" style="height: 200px; width: 100%; display: block;">
            <div class="caption">
                <h3 id="thumbnail-label">@article.Name</h3>
                <p>@article.Description</p>
            </div>
        </div>
    </div>

    counter++;
}
</div>
```

So I render the outer opening and closing `div` tags for the row. Inside that I iterate over each item in the list and render a `div` with the `col-md-4` class, and the actual content inside of that. What I also do is to keep a counter which I start at 0 and increment after every item has been rendered.

Before I render a new item I do a simple check to see whether modulus of the number of items rendered divided by 3 is equal to 0, and also whether the current counter is not 0 (i.e. the start of the list). So this `if` statement will therefore be true for every item after a multiple of 3, i.e. the 4th, 7th, 10th items etc. What I do in those cases is to close the current row and start a new one.

It is simple and effective, but not necessarily very readable, which brings me to the next approach.

Approach 2: Split list into rows

For the second approach what I am going to do is to divide the list of items into a multidimensional array containing the rows, and each row will contain a array with the actual items. So in effect I take an array of items like this:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

and turn it into this:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10]]
```

To do that I use a simple extension method which I got from this Stackoverflow answer, and looks like this:

```
public static class ArrayExtensions
{
    /// <summary>
    /// Splits an array into several smaller arrays.
    /// </summary>
    /// <typeparam name="T">The type of the array.</typeparam>
    /// <param name="array">The array to split.</param>
    /// <param name="size">The size of the smaller arrays.</param>
    /// <returns>An array containing smaller arrays.</returns>
    public static IEnumerable<IEnumerable<T>> Split<T>(this T[] array, int size)
    {
        for (var i = 0; i < (float)array.Length / size; i++)
        {
            yield return array.Skip(i * size).Take(size);
        }
    }
}
```

Ok, so I promised that this approach will be more readable, and so far you are probably thinking that this does not make much sense. Well here is that same list of items rendered using the new approach:

```
@model IEnumerable<BootstrapGridLayout.Models.Article>
@using BootstrapGridLayout;
@{
    ViewBag.Title = "Home Page";
{
```

```
    <div class="row">
        @foreach (var article in row)
        {
            <div class="col-md-4">
                <div class="thumbnail">
                    <img src="@article.Thumbnail" data-holder-rendered="true" style="height: 200px; width: 100%; display: block;">
                    <div class="caption">
                        <h3 id="thumbnail-label">@article.Name</h3>
                        <p>@article.Description</p>
                    </div>
                </div>
            </div>
        }
    </div>
}
```

To me this last approach is more readable, as you can easily see that I am iterating over a number of rows, and inside each row I iterate over a number of items. Do you have another approach for doing this? If so please let me know in the comments below.

PREVIOUS
Using the ASP.NET OAuth providers without ASP.NET Identity

NEXT
Resolve your DbContext as an interface using the ASP.NET 5 dependency injection framework