**Anchor Tag Helper**

The Anchor Tag Helper enhances the standard HTML anchor (<a ... ></a>) tag by adding new attributes such as   
**asp-controller**  
**asp-action**  
**asp-route-{value}**

<**a** **asp-controller**="home" **asp-action**="details"

**asp-route-id**="@employee.Id">View</**a**>

Before That Example:-

/home/details/5  
  
There are several ways we could do this in a razor view  
  
**Option 1 :** Manually generating the links

@foreach (var employee in Model)

{

    <a href="/home/details/@employee.Id">View</a>

}

**Option 2 :** Using HTML helpers

@Html.ActionLink("View", "details", new { id = employee.Id })

generates an anchor element

<a href="/hom/details/5">View</a>

@Url.Action("details", "home", new { id = employee.Id })

generates a string  
  
/hom/details/5

**Advantage of using Tag helpers**  
  
Tag helpers generate links based on the application routing templates. This means if we later change routing templates, the links generated by tag helpers will automatically reflect those changes made to the routing templates. The generated links just work.   
  
Where as if we have hard-coded the URL paths manually, we would have to change the code in lot of places when the application routing templates change.

**Disable Browser Cache**

For some reason, if you do not want a browser to use it's cache you can disable it. For example, to disable cache in Google chrome

* Using **F12** key, launch Browser Developer Tools
* Click on the **"Network"** tab
* Check **"Disable Cache"** checkbox

google chrome disable cache  
  
The obvious problem with disabling the browser cache is that, the images have to be downloaded from the server, every time you visit the page.  
  
**ASP.NET Core Image tag helper**  
  
From a performance standpoint, wouldn't it be great to download the image only if it has changed on the server. If the image has not changed, use the image from the browser cache. This means we will have the best of both the worlds.  
  
**Image Tag Helper**can help us achieve this. To use the Image tag helper, include ***asp-append-version*** attribute and set it to true.

<**img** **src**="~/images/noimage.jpg" **asp-append-version**="true" />

**Image Tag Helper** enhances the <img> tag to provide **cache-busting behavior** for static image files. Based on the content of the image, a unique hash value is calculated and is appended to image URL. This unique string prompts the browser to reload the image from the server and not from the browser cache.

<img class="card-img-top" src="/images/noimage.jpg?v=IqNLbsazJ7ijEbbyzWPke-xWxkOFaVcgzpQ4SsQKBqY" />

Each time the image on the server changes a new hash value is calculated and cached. If the image has not changed the hash isn't recalculated. Using this unique hash value, the browser keeps track of whether the image content on the server has changed

**ASP.NET Core Environment Tag Helper**

<**environment** **include**="Development">

    <link href="~/lib/bootstrap/css/bootstrap.css" rel="stylesheet" />

</**environment**>

***Environment tag helper*** supports rendering different content depending on the application environment. The application environment name is set using using ***ASPNETCORE\_ENVIRONMENT*** variable.  
  
This example loads the non-minified bootstrap css file, if the application environment is "Development"

<**environment** **include**="Development">

    <link href="~/lib/bootstrap/css/bootstrap.css" rel="stylesheet" />

</**environment**>

This example loads the minified bootstrap css file from the CDN (Content Delivery Network), if the application environment is "Staging" or "Production".

<**environment** **include**="Staging,Production">

    <link rel="stylesheet"

            href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

            integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"

            crossorigin="anonymous">

</**environment**>

***"include"*** attribute accepts a single hosting environment name or a comma-separated list of hosting environment names. On the ***<environment>*** tag helper, we also have ***"exclude"*** attribute. The content of the ***<environment>*** tag is rendered when the hosting environment doesn't match an environment listed in the ***exclude*** attribute value.  
  
This example loads the minified bootstrap css file from the CDN (Content Delivery Network), if the application environment IS NOT "Development". 

<**environment** **exclude**="Development">

    <link rel="stylesheet"

            href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

            integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"

            crossorigin="anonymous">

</**environment**>

The ***"integrity"*** attribute on the <link> element is used for **Subresource Integrity check**. Subresource Integrity (SRI for short), is a security feature that allows a browser to check if the file being retrieved has been maliciously altered. When the browser downloads the file, it recalculates the hash and compares it against the "integrity" attribute hash value. If the hash values match, the browser allows the file to be downloaded otherwise it is blocked.  
  
**What if the CDN is down**  
  
If the CDN is down or for some reason, our application is not able to reach the CDN, we want our application to fallback and load the minified bootstrap file (bootstrap.min.css) from our own application web server. Consider the following example

<**environment** **include**="Development">

    <link href="~/lib/bootstrap/css/bootstrap.css" rel="stylesheet" />

</**environment**>

<**environment** **exclude**="Development">

    <**link** rel="stylesheet"

            integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"

            crossorigin="anonymous"

**href**="https://sstackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"

**asp-fallback-href**="~/lib/bootstrap/css/bootstrap.min.css"

**asp-fallback-test-class**="sr-only" **asp-fallback-test-property**="position"

**asp-fallback-test-value**="absolute"

**asp-suppress-fallback-integrity**="true" />

</**environment**>

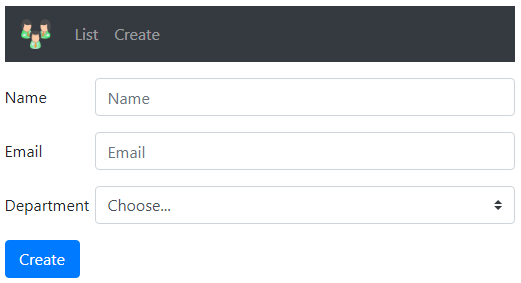
If the application environment is "Development", non-minified bootstrap css file (bootstrap.css) is loaded from our application web server  
  
If the application environment IS NOT "Development", minified bootstrap css file (bootstrap.min.css) is loaded from the CDN

A fallback source is specified using asp-fallback-href attribute. This means, if the CDN is down, our application fallsback and load the minified bootstrap file (bootstrap.min.css) from our own application web server.  
  
The following 3 attributes and their associated values are used to check if the CDN is down  
***asp-fallback-test-class="sr-only"***  
***asp-fallback-test-property="position"***  
***asp-fallback-test-value="absolute"***  
  
Obviously, there is some processing involved to calculate hash and compare it with the integrity attribute hash value. For most applications, fallback source is their own server. You have the option to turn off integrity check for the files downloaded from the fallback source by setting ***asp-suppress-fallback-integrity***attribute to ***true***

### Form tag helpers in asp.net core

**Tag Helpers to create forms**  
  
We use the following common tag helpers to create a form in ASP.NET Core

* Form Tag Helper
* Label Tag Helper
* Input Tag Helper
* Select Tag Helper
* Textarea Tag Helper

We also have Validation tag helpers. We will discuss form validation and model binding in our upcoming videos.  
  
By the end of this video, we want to create a form using the form tag helpers and style it using Bootstrap 4. The ***"Create Employee Form"*** should be as shown below.  
  
  
  
**Form Tag Helper**  
  
To create a form we use the ***<form>*** tag helper  
  
Notice, we are using ***asp-controller*** and ***asp-action*** tag helpers. These 2 tag helpers specify the controller and the action method to which the form data must be posted when the form is submitted. We want to issue a POST request when the form is submitted, so we have set the ***method*** attribute to ***post***

<**form** **asp-controller**="home" **asp-action**="create" method="post">

</**form**>

The above code produces the following HTML when the form is rendered on the client browser. As you can see from the generated HTML when the form is submitted it will be posted to the ***index()*** action of the ***HomeController.***

<**form** method="post" action="/home/create"></**form**>

**Please note :** By default, when a form is submitted, it will be posted to the same action of the controller that rendered the form. So this means, even if we did not specify the controller and action using the ***asp-controller***and ***asp-action***tag helpers, the form will still be posted to the ***index()*** action of the ***HomeController.***  
  
**Input Tag Helper**  
  
The Input Tag Helper binds an HTML **<input>** element to a model expression in your razor view.  
  
In our case, we want a form to create a new employee. So the model for our ***Create.cshtml*** view is **Employee**class. We specify that using the ***model***directive. 

**@model Employee**

To b able to capture the ***employee name***we want a text box. We want the text box to bind to the ***Name***property of the ***Employee***model class. We do this by using ***asp-for***tag helper and setting it's value to the ***Name*** property of the ***Employee***model class. Notice we also have intellisense. Later if we change the property name form ***Name*** to ***FullName*** on the ***Employee***class, and if we do not change the value assigned to the tag helper, we get a compiler error.

<**input** **asp-for**="Name">

The above code generates an ***input***element with ***id***and ***name***attributes. Notice both of them are set to a value of ***Name***. 

<input type="text" id="Name" name="Name" value="">

The ***name*** attribute is required and it is used to map the value of the input element to the corresponding property of the model class when the form is submitted. This is done by a process called model binding in ASP.NET Core. We will discuss model binding in our next video.  
  
**Label Tag Helper**  
  
The Label Tag Helper generates a label with ***for***attribute. The ***for***attribute links the label with it's associated input element. Consider the following example.

<**label** **asp-for**="Name"></**label**>

<**input** **asp-for**="Name">

The above code generates the following HTML. 

<label for="Name">Name</label>

<input type="text" id="Name" name="Name" value="">

The label is linked to the input element, because both the label ***for***attribute and the input element ***id***attribute have the same value ***(Name)***. So this means when we click on the label, the corresponding input element receives the focus.  
  
Similarly, the following code generates a label and an input element to capture employee email.

<**label** **asp-for**="Email"></**label**>

<**input** **asp-for**="Email">

**Select Tag Helper**  
  
Generates ***select***element and it's associated option elements. In our case we want a select element to display the list of departments.  
  
Ultimately we want a label and a select element with list of department options as shown below.

<label for="Department">Department</label>

<select id="Department" name="Department">

    <**option** **value**="0">None</**option**>

    <**option** **value**="1">HR</**option**>

    <**option** **value**="2">Payroll</**option**>

    <**option** **value**="3">IT</**option**>

</select>

The options for the ***department***select element can be hard-coded like in the example above, or they can come from an ***enum***or a database table. We do not have a database hooked up yet. So for our example, let's get the options from an ***enum***.  
  
Place the following ***enum***in ***Dept.cs***file in the ***Models***folder

namespace EmployeeManagement.Models

{

    public enum Dept

    {

        None,

        HR,

        Payroll,

        IT

    }

}

Update the Employee class in *Employee.cs*file in the *Models*folder  
  
*Department*property data type is ***Dept***enum.

namespace EmployeeManagement.Models

{

    public class Employee

    {

        public int Id { get; set; }

        public string Name { get; set; }

        public string Email { get; set; }

        public Dept Department { get; set; }

    }

}

In ***Create.cshtml*** view, include the following code

<**label** **asp-for**="Department"></**label**>

<**select** **asp-for**="Department"

**asp-items**="Html.GetEnumSelectList<Dept>()"></**select**>

Notice, we are using ***asp-items***tag helper and ***Html.GetEnumSelectList<Dept>()*** to get the options for the ***select***element.  
  
The above code produces the following HTML

<label for="Department">Department</label>

<select id="Department" name="Department">

    <**option** **value**="0">None</**option**>

    <**option** **value**="1">HR</**option**>

    <**option** **value**="2">Payroll</**option**>

    <**option** **value**="3">IT</**option**>

</select>

**Create.cshtml - Complete Code without Bootstrap**

@model Employee

@{

    ViewBag.Title = "Create Employee";

}

<**form** **asp-controller**="home" **asp-action**="create" method="post">

    <div>

        <**label** **asp-for**="Name"></**label**>

        <**input** **asp-for**="Name" />

    </div>

    <div>

        <**label** **asp-for**="Email"></**label**>

        <**input** **asp-for**="Email">

    </div>

    <div>

        <**label** **asp-for**="Department"></**label**>

        <**select** **asp-for**="Department"

**asp-items**="Html.GetEnumSelectList<Dept>()"></**select**>

    </div>

    <button type="submit">Create</button>

</**form**>

The above code generates the following HTML

<form method="post" action="/home/create">

    <div>

        <label for="Name">Name</label>

        <input type="text" id="Name" name="Name" value="" />

    </div>

    <div>

        <label for="Email">Email</label>

        <input type="text" id="Email" name="Email" value="">

    </div>

    <div>

        <label for="Department">Department</label>

        <select id="Department" name="Department">

            <option value="0">None</option>

            <option value="1">HR</option>

            <option value="2">Payroll</option>

            <option value="3">IT</option>

        </select>

    </div>

    <button type="submit">Create</button>

</form>

**Create.cshtml - Complete Code with Bootstrap**

@model Employee

@{

    ViewBag.Title = "Create Employee";

}

<**form** **asp-controller**="home" **asp-action**="create" method="post" class="mt-3">

    <div class="form-group row">

        <**label** **asp-for**="Name" class="col-sm-2 col-form-label"></**label**>

        <div class="col-sm-10">

            <**input** **asp-for**="Name" class="form-control" placeholder="Name">

        </div>

    </div>

    <div class="form-group row">

        <**label** **asp-for**="Email" class="col-sm-2 col-form-label"></**label**>

        <div class="col-sm-10">

            <**input** **asp-for**="Email" class="form-control" placeholder="Email">

        </div>

    </div>

    <div class="form-group row">

        <**label** **asp-for**="Department" class="col-sm-2 col-form-label"></**label**>

        <div class="col-sm-10">

            <**select** **asp-for**="Department" class="custom-select mr-sm-2"

**asp-items**="Html.GetEnumSelectList<Dept>()"></**select**>

        </div>

    </div>

    <div class="form-group row">

        <div class="col-sm-10">

            <button type="submit" class="btn btn-primary">Create</button>

        </div>

    </div>

</**form**>