بسم الله الرحمن الرحيم .

HTML Helper :

الـ (Asp.Net MVC) تتيح لك العديد من المساعدين (Helpers) لتوليد (HTML) بسهولة و بكفاءة عالية ، ووحده من هذه (Helper) هي (HTML Helper) .

ونستعملها في (View) لترجمة أو تفسير محتويات (HTML) ، وفي عالم (Asp.net MVC) فأن (HTML Helpers) مشابه الى حدآ ما للـ (Controls) في (Asp.Net We Form) لكنها تفرق في ثلاث وهي أنها أخف و لاتستخدم (View State) ولاتحتوي على (Event Models) .

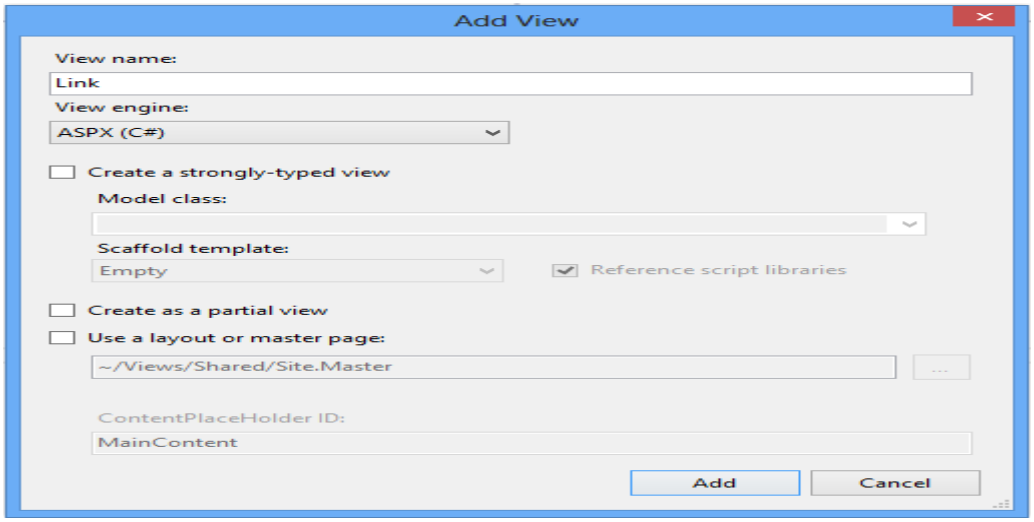
أنشاء الروابط :

هناك طريقة بسيطة لأنشاء الروابط لأي أكشن في الكونترولر داخل (View) بأستعمال الأجراء (HTML.ActionLink ()) .

دعونا ننشئ مشروع جديد بأسم (Fourth) بنفس طريقة انشاء المشاريع السابقة ، وداخل الكونترولر (Home) ننشئ أكشن جديد بأسم (Link) :

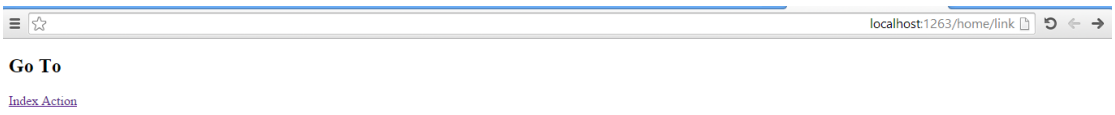
```
public ActionResult Link()
        {
            return View();
        }
```

ثم ننشئ (View) لهذا الاكشن ، من خلال (Right Click) على الاكشن ونختار (Add View) :



```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <meta name="viewport" content="width=device-width" />
    <title>Link</title>
</head>
<body>
    <div>
        <h2> Go To </h2>
        <%= Html .ActionLink ("Index Action","Index") %>
    </div>
</body>
</html>
```

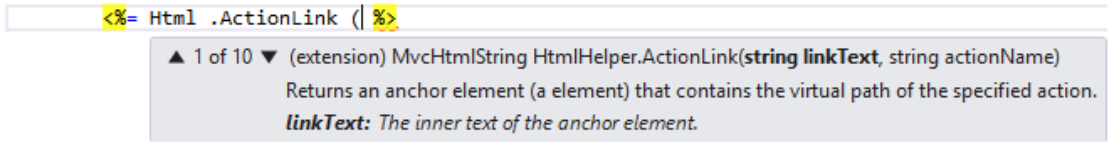
في الكود أعلاه فأن الاجراء (Html.ActionLink()) أخذ معاملين الاول عنوان نص الرابط والثاني الاكشن الذي سينتقل اليه .



وهو مشابه للوسم هذا :

```
<a href ="/Home/Index">Index Action</a>
```

(Html.ActionLink()) يدعم المعاملات التالية :



- Link Text : عنوان نص الرابط .
- Action Name : أسم الأكشن الذي سينتقل اليه .
- Route Values : مجموعة القيم التي تمرر للأكشن .
- Controller Name : أسم الكونترولر الذي سسنتقل للأكشن الذي داخله .
- Html Attributes : مجموعة خصائص (HTML) التي تضاف للرابط .

- Protocol : اسم البروتوركول الذي سنستخدمه في الرابط (مثلا ان كان الانتقال الى اكشن اخر سنستخدم البروتوكول HTTP) .
- Host Name : عنوان المستضيف ان كنا نريد الانتقال الى موقع آخر .
- Fragment : الهدف المعتمد للرابط ، على سبيل المثال عندما نريد الانتقال من اسفل الصفحة الى الاعلى .

في (Route Values) يفيدنا في تمرير قيم من أكشن معين الى أكشن اخر ، لنأخذ مثال بسيط حول هذا ، حيث سنمرر قيمة من الاكشن (Link) الى أكشن جديد أسمه (target) :

انشئ أكشن جديد بأسم (target) داخل الكونترولر (Home) وفيو جديد لهذا الاكشن ، ويكون الاكشن هكذا :

```
public ActionResult taget(int id)
        {
            ViewBag.Id = id;
            return View();
        }
```

هذا الميثود يستقبل معامل واحد فقط وهذا المعامل سنضعه في (View bag) من أجل تمريره الى الفيو (target) .

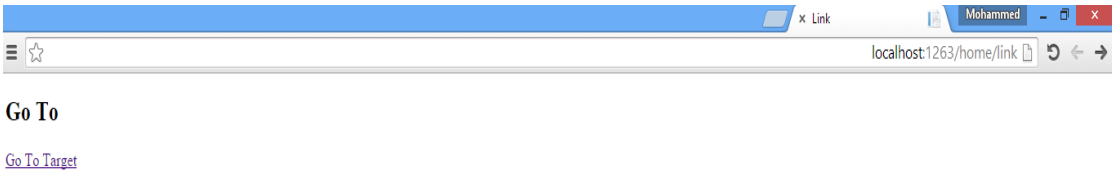
نعدل على الفيو (Link) :

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <meta name="viewport" content="width=device-width" />
    <title>Link</title>
</head>
<body>
    <div>
        <h2> Go To </h2>
        <%= Html.ActionLink("Go To Target", "taget", new {Id=3 })%>

    </div>

</body>
</html>
```

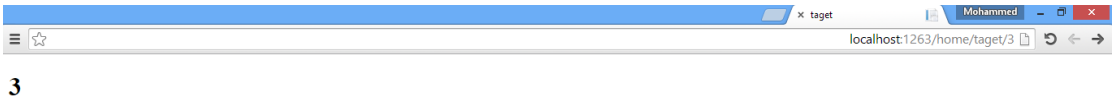
كما تلاحظ في الكود أعلاه سنجعل (Action Link) يمرر قيمة (Id) للأكشن (target) الذي جعلناه يستقبل قيمة واحده ، وسيضعها في (View bag) ويمررها بالاخير الى الفيو (target) الذي بدوره سيعرضها ، ويكون كود الفيو (target) :

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <meta name="viewport" content="width=device-width" />
    <title>taget</title>
</head>
<body>
    <div>
       <h1><%= ViewBag .Id %></h1>
    </div>
</body>
</html>
```

عند التنفيذ نكتب في رابط المتصفح أسم الأكشن (Link) :



وعند النقر على الرابط (Go To Target) فانه سينتقل للأكشن (Target) حاملا معه قيمة (Id) :



هناك أجراء مهم أسمه (Route Link) وهو مشابه للـ (Action Link) من حيث الشكل لكنه يقبل (Route Name) ولا يملك اسم الكونترولر أو الاكشن . مثال بسيط :

```
<%= Html.RouteLink("Go To Index", new {Action ="Index" })%>
```

سنتطرق اليه أكثر في موضوع (Routing) .

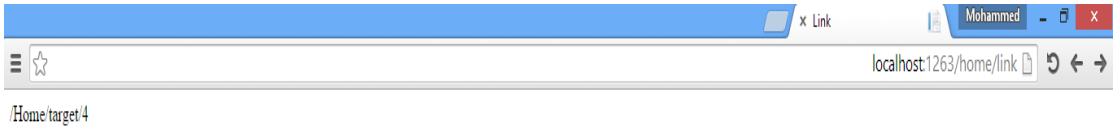
: URL Helper

مشابه للـ (Action Link) و (Route Link) لكنه بدلاً من أرجاع (Html) هو يُرجع (URL) كـ (String) ، ويحوي على ثلاثة (helpers) هم :

- Action : مشابه جدا للـ (Action Link) لكنه لا يعيد (Anchor Tag) .

```
<span><%= Url.Action ("target","Home",new{Id=4})%></span>
```

سوف لا يرجع (URL) لكنه سيرجع الرابط كنص هكذا :



وهي مشابه (/Home/target/4) في HTML .

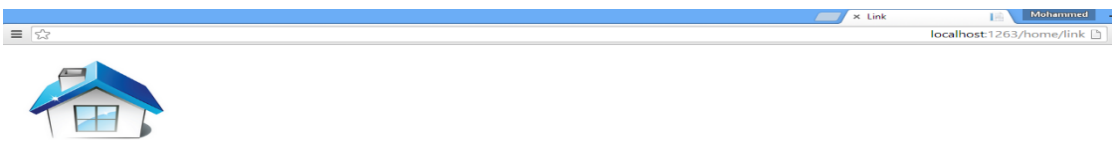
- Content : هذا يساعد في تحويل مسار التطبيق النسبي الى تام .
- Route URL : مشابه جداً للـ (Route Link) .

: Image Link

لسوء الحظ لايمكنك أن تستخدم (Action Link) لأنشاء رابط صوري (Image Link) لأن (Action Link) يشفر (Encode) نص الرابط اوتوماتيكياً ، لذلك بدله سنستخدم (URL Action.) لتوليد رابط مناسب مثال :

```
<a href ="<%= Url.Action ("Index") %>">
        <img src ="../../Content/home.jpg" alt ="Home Page" width ="200px"
height ="180px" />
    </a>
```

كما تلاحظ لقد أستخدمنا الوسم (< a>) لعمل الرابط، وجعلناه ياخذ مسار الاكشن من خلال (URL .Action) لانه يعيد رابط الاكشن على شكل نص وهكذا :



وعند النقر على الصور ، سننتقل الى الأكشن (Index) .

: Form Elements

بدون استخدام النماذج (Form) فأن الانترنت سيصبح أشبه بمستودع لخزن المستندات وللقراءة فقط ، بحيث انك لا تستطيع ان تبحث عن شي في الانترنت او شراء شي .

النموذج (Form) يحتوي على عدد من العناصر الخاصة بالادخال (Input Elements) مثل (Buttons , Check Boxes , Text Input ..... الخ) وهذه العناصر تمكن المستخدم أرسال المعلومات الى الصفحة وثم أرسالها الى الخادم (Server) لكن كيف ذلك ؟

الجواب هو أعتماداً على خاصيتين يملكهما وسم النموذج وهما (Action و Method) ، حيث أن الخاصية (Action) تسأل المتصفح الى اين يريد أرسال المعلومات ، لذلك بطبيعة الحال فأن (Action) يحتوي على (URL) وهذا الـ (URL) ممكن أن يكون نسبي (Relative) بمعنى رابط لصفحة داخل نفس الموقع او على نفس الخادم ، وفي حالات أخرى يكون تام (Absolute) بمعنى رابط لموقع آخر على خادم آخر .

مثلاً هذا الرابط تام ، حيث سيقوم بأرسال نص من أي موقع لمحرك البحث (Bing) :

```
<form action="http://www.bing.com/search">
      <input name="q" type="text" />
      <input type="submit" value="بحث " />
</form>
```

للتنويه فقط كان اسم مربع النص في الكود أعلاه (q) لأن (Query String) لمحرك البحث (Bing) هو (q) بمعنى ان محرك البحث (Bing) يبحث في قاعدة بياناته اعتمادا على قيمة (q) .

أما الخاصية (Method) فأنها ستسأل المتصفح ماذا سيستخدم هل(HTTP Post) او (HTTP Get) .

هناك العديد من (HTML Helpers) التي نستعملها لأنشاء عناصر النموذج (HTML Form Elements) منها :

i. Begin Form()
ii. CheckBox()
iii. DropDownList()
iv. End Form()
v. Hidden()
vi. ListBox()
vii. Password()

viii. RadioButton()
ix. TextArea()
x. TextBox()

Begin Form( ) و End Form() :

في (HTML) لكل عنصر وسم بداية ووسم اغلاق او نهاية ، مثلا للنموذج كنا نكتب وسم البداية والنهاية هكذا :

```
<form>
.
form elements
.
</form>
```

في (Asp.Net MVC) فأن (Begin Form ()) يعني وسم البداية ، و (End Form()) يعني وسم النهاية ، لو أعدنا المثال اعلاه بأستخدام (Asp.Net mvc) سيكون هكذا :

```
<% using ( Html.BeginForm ()) { %>
form elements
<% } %>
```

اما المعاملات التي تقبلها (Begin.Form()) :

- ✓ Route values : مجموعة القيم التي تمرر للأكشن .
- ✓ Action Name : أسم الاكشن الهدف لـ (Form Post) .
- ✓ Controller Name : اسم الكونترولر الهدف لـ (Form Post) .
- ✓ Method : تكلمنا عنها سابقآ (لكن دائمآ نستخدمها في في الجافا سكربت) .

DropDown List و ListBox :

كلاهما يرجعان عناصر على شكل قائمة (Select /) ، حيث أن Dropdown List يسمح بأخيار قيمة واحدة بينما الـ Listbox يسمح بأكثر من قيمة (من خلال الخاصية multiple) .

كذلك يمكن من خلال هذين العنصرين تمثيل سجلات من قاعدة البيانات ، سنأخذ مثال بسيط حولهما :

دعونا ننشئ أكشن جديد بأسم (Dropdown) ، وداخله نعمل (List) من الدول ثم بعد ذلك نضعها في (DataView) من خلال تحويلها الى (SelectList) ونرسلها للفيو :

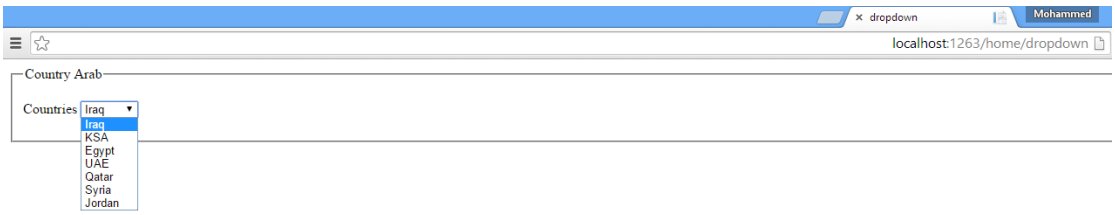
```
public ActionResult  dropdown()
        {
            List <string> li =new List <string >();
            li.Add("Iraq");
            li.Add("KSA");
            li.Add("Egypt");
            li.Add("UAE");
            li.Add("Qatar");
            li.Add("Syria");
            li.Add("Jordan ");
            ViewData["Countries"] =new SelectList ( li);
            return View();
        }
```

ثم (Right Click) على الاكشن نختار (Add View) لإضافة فيو جديد بإسم (Dropdown) :

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <meta name="viewport" content="width=device-width" />
    <title>dropdown</title>
</head>
<body>
    <div>
        <% using (Html .BeginForm ()){ %>
        <fieldset >
            <legend >Country Arab </legend>
            <p><%= Html.Label ("Countries") %>
                <%=Html.DropDownList("Countries",ViewData["Countries"]as
SelectList )%>


            </p>

        </fieldset>
        <% } %>
    </div>
</body>
</html>
```

في الكود إعلاه بدأنا بإستعمال وسم البداية للنموذج (Html.BeginForm()) ثم ضفنا عنصر (Label) وبعده ضفنا (DropdownList) يأخذه قيمة من الـ (View Data) وبما ان الـ (

View Data) هي عبارة عن (Dictionary) لذلك قمنا بتحويلها إلى (Dropdown List) لتتناسب مع (Dropdown) .



الـ (Dropdown) يدعم معامل أختياري نسميه (Optional Label) لن أشرحه ، ولكن بعد هذا التعديل على الكود أعلاه ستفهمه ، فقط نعدل على كود (Dropdown) في الفيو :

```
<%=Html.DropDownList("Countries",ViewData["Countries"]as SelectList ,"Select
your Country")%>
```

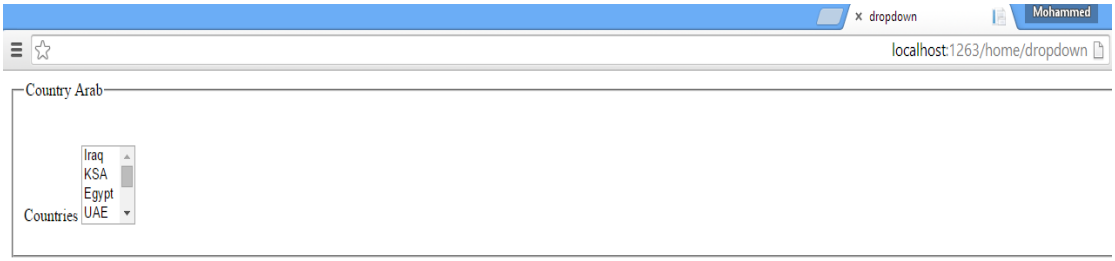
وعند التنفيذ :

لو أستخدمنا الـ (Listbox) بدل من (Dropdown) يكون فقط من خلال التعديل على الفيو :

```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!DOCTYPE html>

<html>
<head runat="server">
    <meta name="viewport" content="width=device-width" />
    <title>dropdown</title>
</head>
<body>
    <div>
        <% using (Html .BeginForm ()){ %>
        <fieldset >
            <legend >Country Arab </legend>
            <p><br />
                <%=Html .Label ("Countries") %>
                <%= Html.ListBox ("Countries",ViewData["Countries"]as
SelectList) %>
            </p>

        </fieldset>
        <% } %>
    </div>
</body>
</html>
```

إذا لاحظت أننا لم نضع (Html.EndForm()) لغلق النموذج لأن الـ (Asp.net MVC) بصورة تلقائية إن لم تجدها فأنها تضعها .

عموما بعد التعديل على الفيو وتنفيذ التطبيق يكون هكذا :



Textbox و Text Area :

بالنسبة للـ (Textbox) يقابل الوسم (input) في (Html) والذي خاصية النوع فيه (type= text) ، هذا العنصر يمكّن المستخدم من الكتابة فيه ولكن بسطر واحد فقط

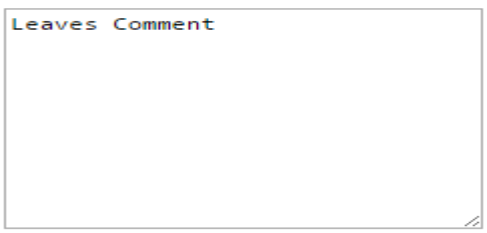
```
<%= Html.TextBox ("UserName","YourName") %>
```

المعامل الاول (UserName) هو أسمه البرمجي ، اما المعامل الثاني وهو أختياري (Your Name) ويمثل قيمته ، لو أعدنا كتابة الكود إعلاه بـ (Html) يكون هكذا :

```
<input id="UserName" name ="UserName" value ="YourName" type ="text" />
```

في بعض الأحيان تريد المسخدم أن يكتب أكثر من سطر في هذه الحالة ستضطر لأستخدام العنصر (Text Area) :

```
<%= Html.TextArea ("Comment","Leaves Comment" )%>
```



كذلك يمكن فيه تحديد عدد الاسطر (row) والاعمدة (col) هكذا :

```
<%= Html.TextArea ("Comment","Leaves Comment",10 ,80 ,null    )%>
```

اما العنصر (Password()) فهو يفيدنا في النماذج التي فيها تسجيل او تسجيل دخول :

```
    <%= Html.Password ("Password")%>
```

اما العنصر (Label) فهو نستخدمة لعرض نص معين :

```
  <%= Html.Label ("Password","Your Password") %>
```

إذا لاحظت في بعض الامثلة السابقة جعلنا أسم (Label) مشابه لأسم العنصر الذي يليه مثلا :

```
<%= Html.Label ("Password","Your Password") %>
  <%= Html.Password ("Password")%>
```

طيب ما هي الغاية من ذلك ؟ هذا يفيدنا في ربط الـ (Label) مع العنصر (Password) بتركيز واحد (Focus) يعني جرب تنقر على الـ(Label) ستجد أن المؤشر صار في العنصر (Password) .

اما عن كيفية ربط العناصر إعلاه مع المودل فسنكلم عنه أكثر فيه موضوع (الوصول للبيانات – Data Access) .

تشفير محتويات HTML :

لدى بعض المخترقين (Hackers) القدرة على حقن سكربت جافاسكربت وعند إعادة عرض قيم حقول النموذج (Form) فأن هذا السكرب يقوم بسرقة المعلومات وإرسالها للهكر ، لذلك يفضل

دائماً تشفير المحتويات المدخلة (HTML Encoding) في النموذج وبالاخص تلك الخاصة بكلمات السر .... الخ .

على سبيل المثال عند عرض (username) في الفيو يجب تشفيره هكذا :

```
<%= Html.Encode(UserName) %>
```

حيث يقوم (HTML Encoding) بإستبدال بعض الحروف برموز معينة وامينة ، منها :

- < تصبح < .
- > تصبح > .
- " تصبح " .
- & تصبح & .

مثلا لو كان السكربت التالي لأحد حقول النموذج :

```
<Script > alert (' Boom ! ' ) </Scritp>
```

وعند أستخدام (Html Encoding) يصبح هكذا :

```
&lt;script&gt;alert(‘Boom!’)&lt;/script&gt;
```

الـ (Asp.net MVC) تملك سمة مهمة جداً تسمى (Request Validation) مهمتها منع الهكر بصورة تلقائية من الحصول على محتويات النموذج (Form) .

Antiforgery Tokens

هناك نوع من حقن سكربت للجافا سكربت يسمى (Cross-Site Request Forgery) ومختصره هو (CSRF) هذا النوع من الاختراق بأختصار يعتمد على الهندسة الاجتماعية ويستخدم موقع آخر لسرقة معلوماتك . لناخذ مثال حتى تتوضح الصورة أكثر ، لنفترض لديك حساب بأحد المصارف ، موقع هذا المصرف يتحقق منك من خلال (Cookies) الفريد الخاص بك ، لو سجلت وفي نفس الوقت قمت بزيارة موقع آخر مثلا منتدى وهذا المنتدى به أمكانية رفع صور بالتعليقات او الردود فسيقوم الهكر بوضع تعليق هكذا :

```
<img src=”http://www.BigBank.com/withdraw?amount=9999” />
```

لاحظ أن مسار الصورة هو URL للمصرف ، وعندما تشاهد هذه الرسالة على متصفحك (حصلت على 9,999$) فأن الهكر يحصل على المال من حسابك المصرفي لان موقع المصرف يستخدم الكوكيز للتحقق منك وقد أخترق الهكر متصفحك .

لذلك ولمنع (CSRF) يفضل أن تستخدم :

```
<%= Html.AntiForgeryToken () %>
```

هذا المساعد (Helper) ينشئ حقل ادخال مخفي (hidden Input) الذي يحوي قيمة عشوائية سرية ، وفي كل مرة تطلب الفيو فأن هذه القيمة تتغير . مثلا :

```
<input
name="__RequestVerificationToken"
type="hidden"
value="6tbg3PWU9oAD3bhw6jZwxrYRyWPhKede87K/PFgaw
6MI3huvHgpjlCcPzDzrTkn8" />
```

وبنفس الوقت المساعد ينشئ كوكيز يمثل هذه القيمة والتي تقارن مع قيمة الحقل المخفي لمعرفة فيما إذا كان هناك (CSRF) .

هذا المساعد يقبل بعض المعاملات الاختيارية وهي :

- ✓ Salt : يعني تشفير هذه القيمة العشوائية لزيادة امنيتها أكثر .
- ✓ Domain : يعني أن الموقع يرسل الكوكيز فقط عندما تكون زيارته بصورة مباشرة عن طريق (Domain) الخاص به وليست من موقع آخر .
- ✓ Path : يعني ان الموقع سيرسل الكوكيز فقط عندما تزوره من مسار معين .

أنشاء مساعد (Helper) خاص بك :

في بعض الاحيان تحتاج لمساعد خاص بك وبمواصفات معينة لأن (Asp.Net MVC) لديها عدد محدد وقليل من (Helpers) .

ان انشاء مساعد عملية سهله جدا وذلك من خلال توسيع أجراء على الفئة (HTML Helper Class) ، دعونا ننشئ (Submit Button) وكالتالي :

اولا : نضيف مجلد جديد بإسم (Helpers)

ثانيا : نضيف داخله فئة (Class) بإسم (SubmitButtonHelper)

ثالثا: نستدعي مكتبة (MVC) في الفئة :

```
using System.Web.Mvc ;
```

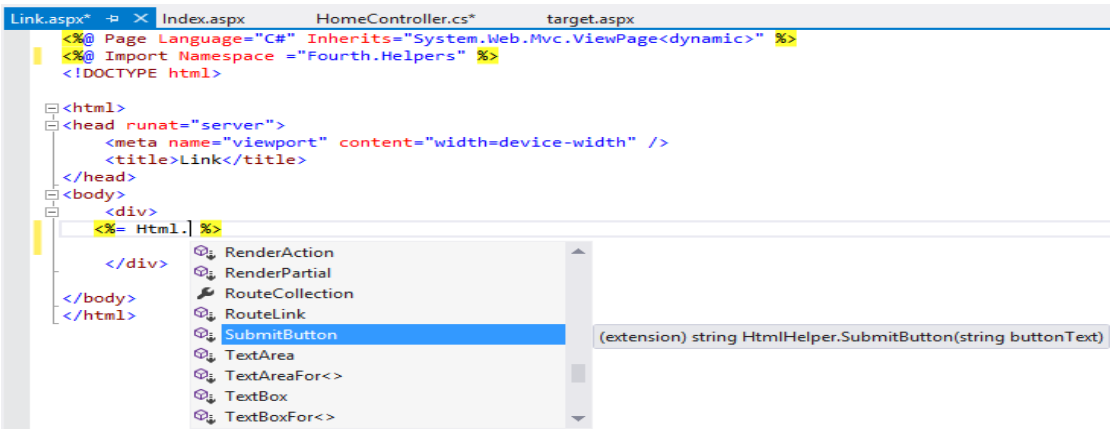
رابعآ : نعرف داخل الفئة إجراء من نوع (String) بإسم (SubmitButton) والتي تقبل معاملين الاول من نوع (Html Helper) والثاني نص يمثل (ButtonText) ونجعلها ترجع قيمة (Html) هكذا :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc ;
namespace Fourth.Helpers
{
    public static  class SubmitButtonHelper
    {
        public static string SubmitButton(this HtmlHelper helper , string
buttonText)
        {
            return string.Format("<input type=\"submit\" value=\"{0}\" />",
buttonText);
        }
    }
}
```

الى هنا أكتمل انشاء المساعد (SubmitButtonHelper) ، اما إستدعاءه يكون بسيط جدا ، مثلا في الفيو (Link) استدعي (Helpers) هكذا :

```
<%@ Import Namespace ="Fourth.Helpers" %>
```

وبالتالي يكون كود الفيو هكذا :



ومن الاشياء الجميلة التي تعطيها لك (ASP.Net MVC) هو الفئة (Tag Builder Class) التي تساعدك في أنشاء وسوم (HTML) الخاصة بك بسهولة ويسر .

اما الاجراءات التي يدعمها هذا الـ (Class) وهي :

- AddCssClass : - يمكنك من أن تضيف فئات (Css) للوسم .
- GenerateID : يمكنك من الخاصية (ID) للوسم .
- MergeAttribute : تمكنك من أضافة خصائص للوسم .
- SetInnerText : - تمكنك من أضافة النص الداخلي للوسم الذي يتم تشفيره تلقائياً .
- ToString : تمكنك من تفسير او تحويل الوسم ، بمعنى أنها تساعدك في تحديد طبيعة الوسم هل هو وسم بداية او نهاية او وسم اغلاق ذاتي .

أما خصائص (Tag Builder) فهي :

- Attributes : تمثل كل خصائص الوسم .
- IDAttributeDotReplacement : يمثل الحرف المستخدم في الإجراء (GenerateID()) .
- InnerHTML : يمثل المحتويات الداخلية للوسم .
- TagName : يمثل إسم الوسم .

هذه الاجراءات والخصائص إعلاه تعطيك كل الاجراءا والخصائص الي تحتاجها لبناء وسم (HTML) .

هناك فة أخرى (HTML Text Write) مشابه الى حد ما للـ (Tag Builder) لكنها تحتوي على أجراءات اضافية منها :

- AddStyleAttributes : تمكنك من أضافة خاصية (Style) للوسم وتستدعى مع بداية فتح الوسم .
- RenderBeginTag : فتح وسم البداية للإخراج .
- Render End Tag : غلق اخر وسم للأخراج .
- Write : كتابة النص للإخراج .
- WriteLine : كتابة سطر جديد للإخراج .

الى هنا أنتهى الدرس الرابع

الدرس الخامس إن شاء الله سيكون عن (Routing) .

محبتي وتحياتي .

محمد الساعدي

mohamediddan.wordpress.com