Asp.Net MVC Part1

محمد الساعدي

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

ماهي Asp.Net MVC ؟

هي عبارة عن بيئة (Framework) تطوير تطبيقات الويب من أنتاج شركة مايكروسوفت ، الغاية منها تسهيل بناء تطبيقات جيده ، وهي جزء من بيئة التطوير الرئيسية (.Net Framework) .

MVC هي اختصار لـ (Model , View , Controller).

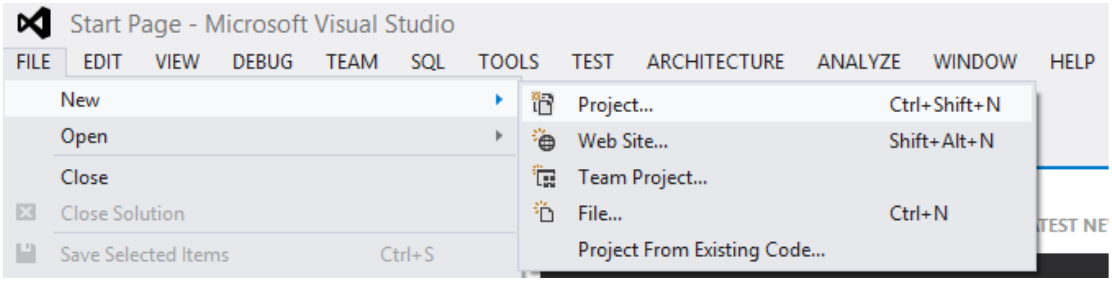
والتي تتألف من ثلاثة أجزاء هي :

١. Model : يمثل جميع العمليات المنطقية مثل (Validation Logic) او (Data Access Logic) (سنتطرق اليه أكثر في الدروس القادمة) وبأختصار هو عبارة عن Class .
٢. View : يمثل صفحة Html أو View Logic .
٣. Controller : كذلك هو عبارة عن Class ، يكون وسيط بين (Model) و (View) .

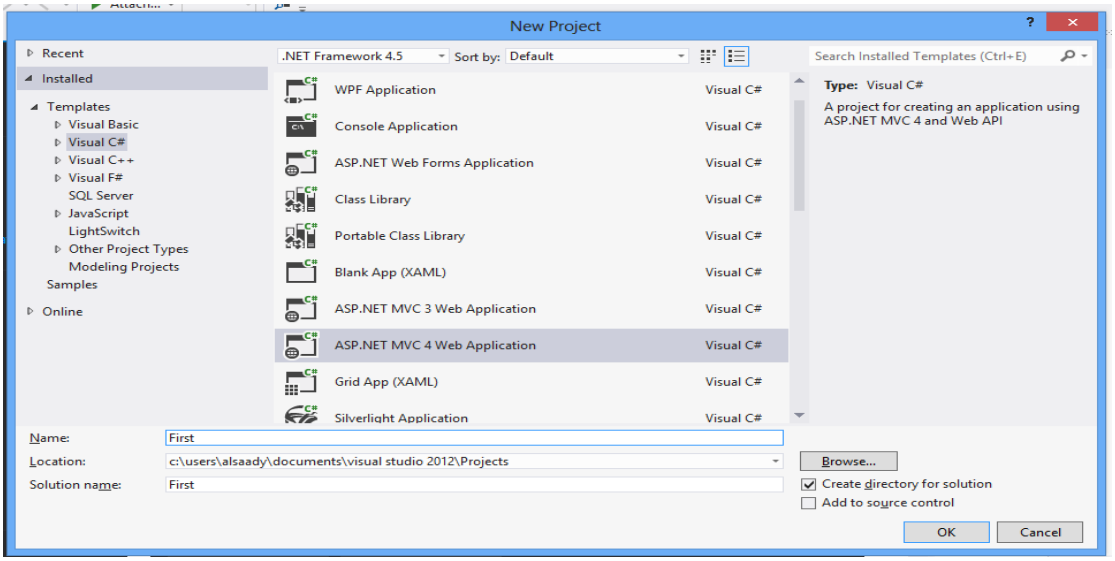
قبل التعمق أكثر في MVC لنتعرف أولا حول كيفية انشاء مشروع Asp.Net MVC لأول مرة :

الخطوة الاولى : نشغل Visual Studio 2012 .

الخطوة الثانية : من File نختار New ثم Project

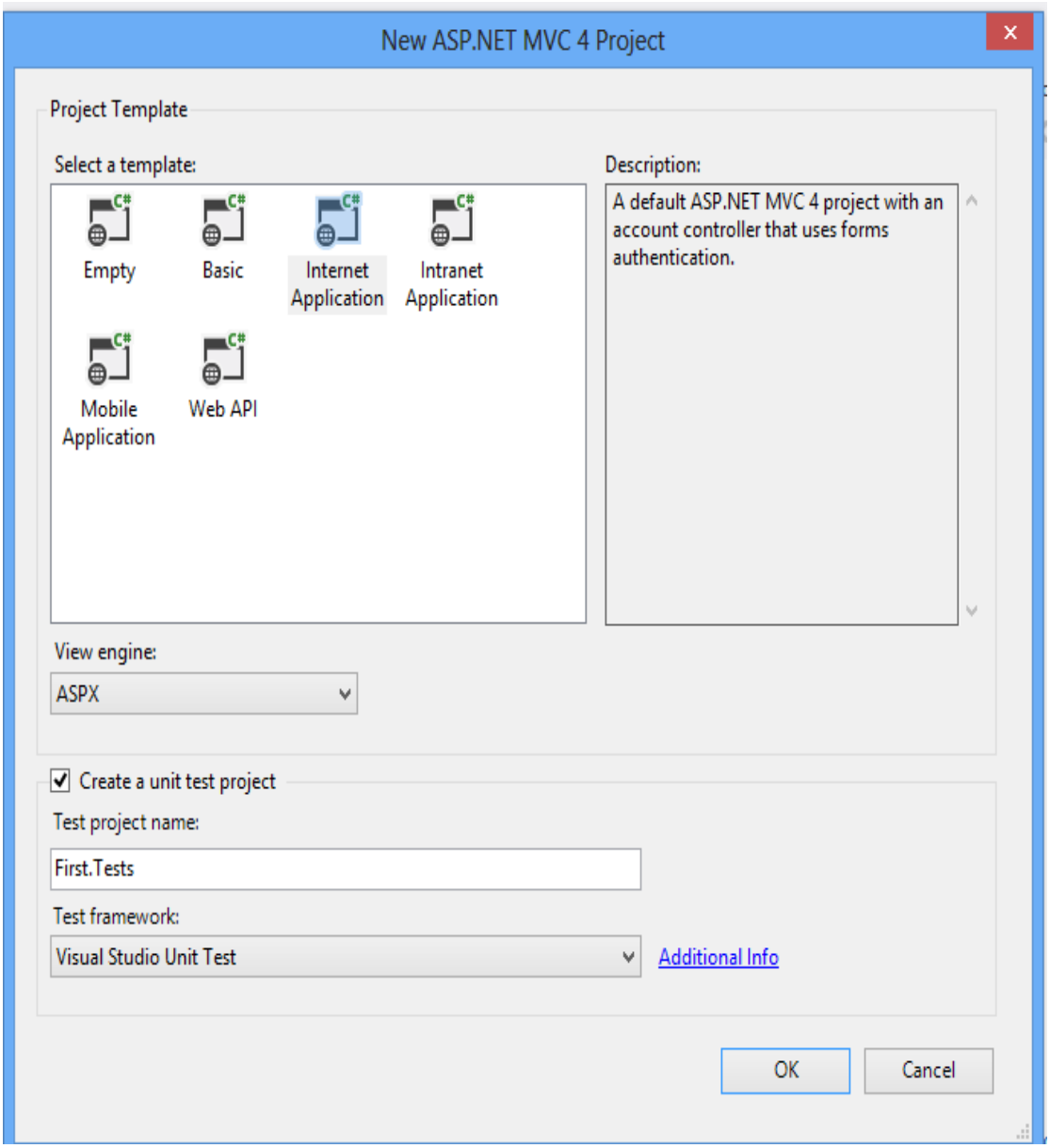


الخطوة الثالثة : تختار لغة البرمجة التي تفضلها للعمل بها ثم تختار (Asp.Net MVC Web application) وتضع أسم مناسب للمشروع كما في الصورة أدناه :

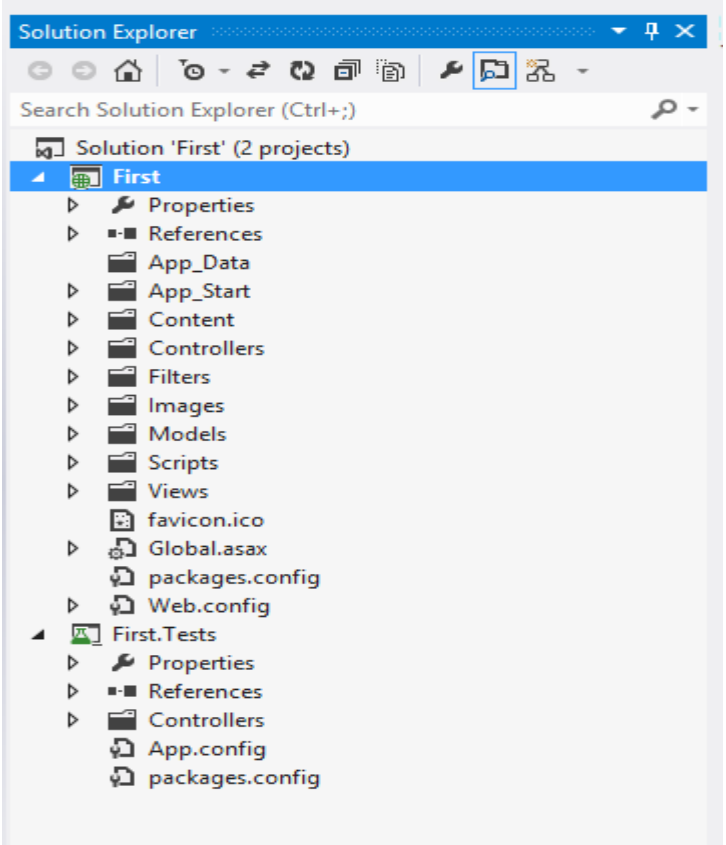


ثم تنقر على (OK) .

الخطوة الرابعة : بعد ان أتممت الخطوة الثالثة ستظهر شاشة تطلب منك أن تختار القالب (Template) سنختار (Internet Application) هذا القالب سيعطينا مشروع افتراضي جاهز ومنظم نستطيع التعديل عليها . ومن View Engine نختار نوع العرض الذي نريده (كيف سيكون View) اما (razor) أو (Aspx) بالنسبة للأشخاص الذين شبق لهم العمل على (Web Form) فسيكون (Aspx) مناسب لهم جدا ، وكذلك يمكنك ان تنشئ مع مشروع وحدة الاختبار (Unit Test Project) .



الى هنا أنتهينا من انشاء المشروع وسيكون بهذا الشكل : .



: Controller

بأختصار الكونترولر هو عبارة عن (Class) يرث من (System.Web.Mvc.Controller Class) وهو المسؤول عن انسيابية التنفيذ ، حيث عندما المتصفح يطلب تطبيق (Asp.Net MVC) فان الكونترولر هو المسؤول عن الاستجابة وتجهيز المتصفح بالمطلوب .

يجب ان يكون أسم الكونترولر هكذا (Controller + اسم الكونترولر) ، كل كونترولر يمكن ان يحتوي على أكثر من دالة (Method) وهذه الدالة تسمى بـ (action) وهذا الاكشن يمكن ان يعيد أنواع مختلفة من النتائج للمتصفح كأن يعيد (View) أو ملف او قد يوجه الى أكشن آخر .

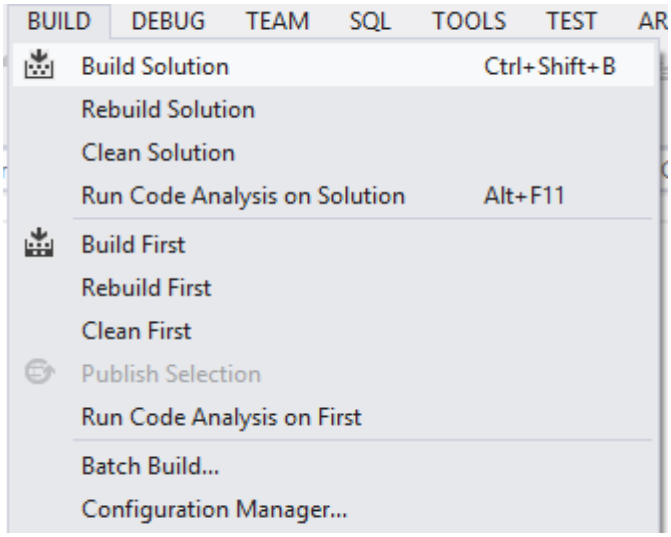
عند أنشاء اي كونترولر بصورة أفتراضية سيكون داخله اكشن اسمه (Index) التي تعرض بصورة افتراضية عند طلب الكونترولر ، كذلك يمكن لأي شخص على الانترنت مشاهدة الاكشن في الكونترولر ويمكن حل هكذا اشكالية بأستخدام الخاصية Non Action .

عرض الصفحات يكون بهذا الشكل

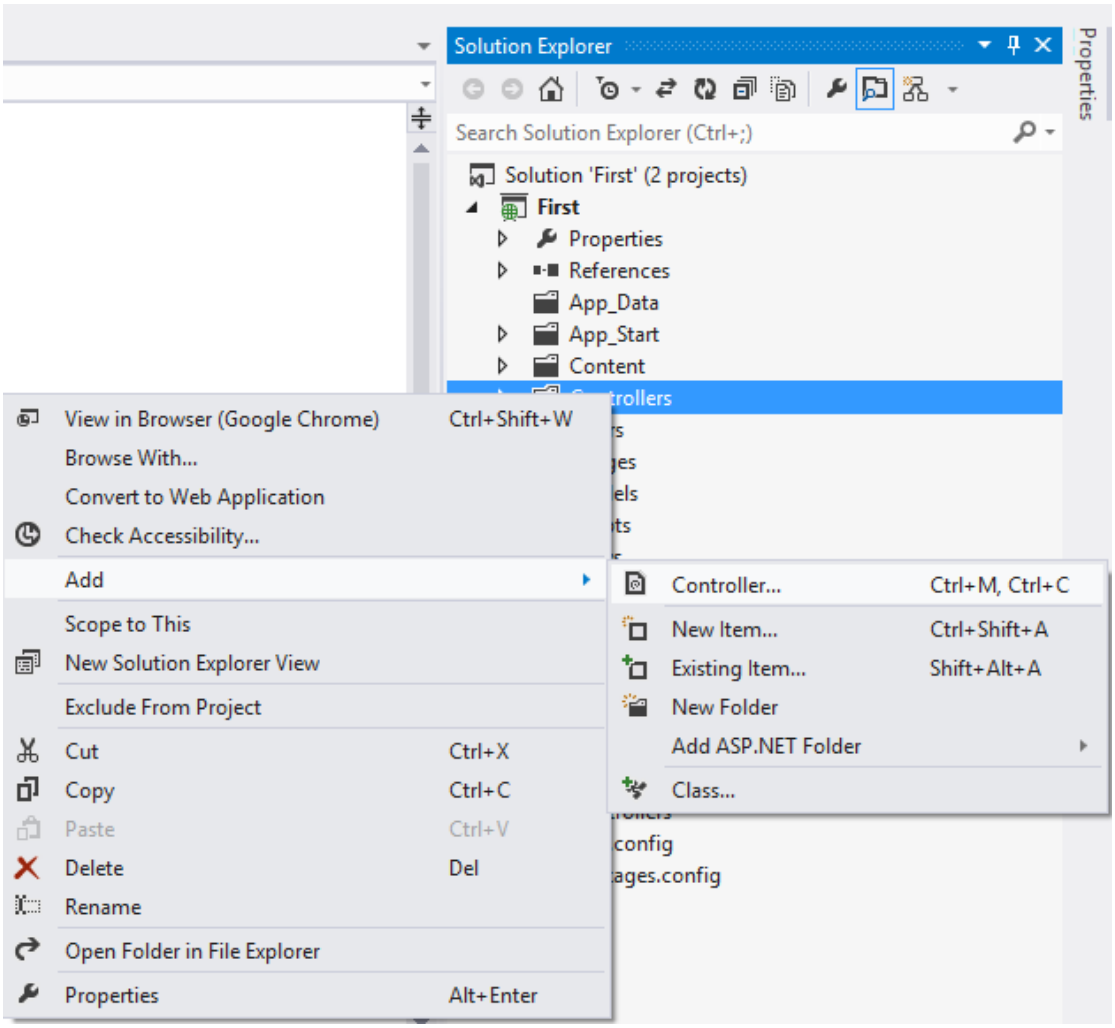
{ أسم الكونترولر } / { أسم الاكشن } / {id }

الـ ID يكون أختياري بمعنى بأمكانك أن تضع له قيمة او لا ، واذا لم نضع اسم الاكشن فأنه بصورة أفتراضية سوف يأخذ الاكشن الذي يحمل أسم (Index) ويمكن التعديل على نوذج العرض اعلاه من خلال Route .

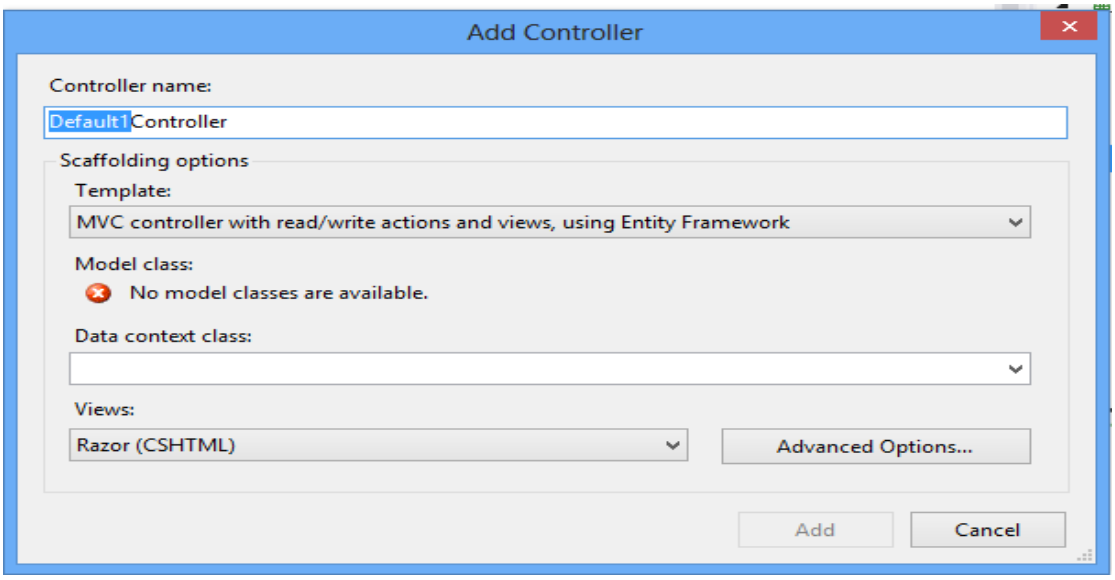
دعونا بالبداية ننشئ كونترولر جديد وقبل ان ننشئه يجب ان نعمل (Build) للمشروع



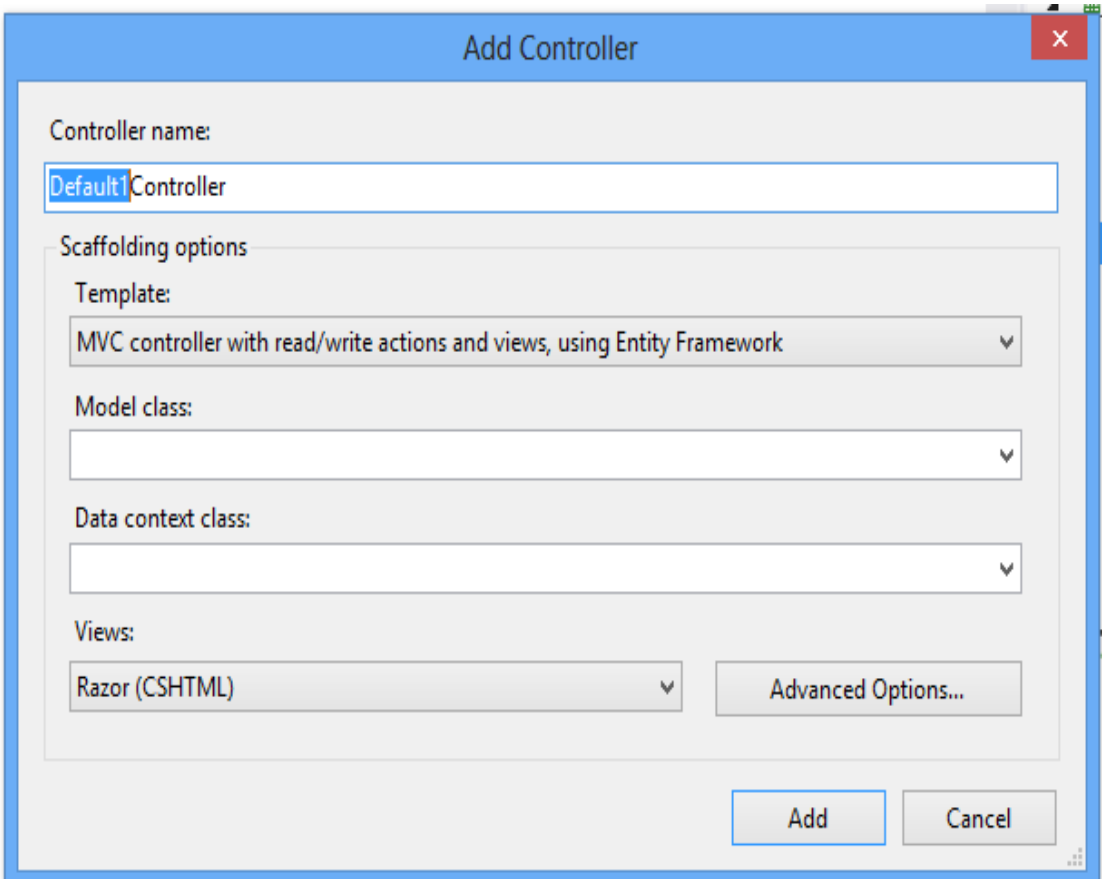
ثم من خلال (Right Click) على المجلد (Controllers) هذا المجلد سيحوي كل الكونترولز الخاصة بالمشروع ، نختار (Add) ثم (Controller) :



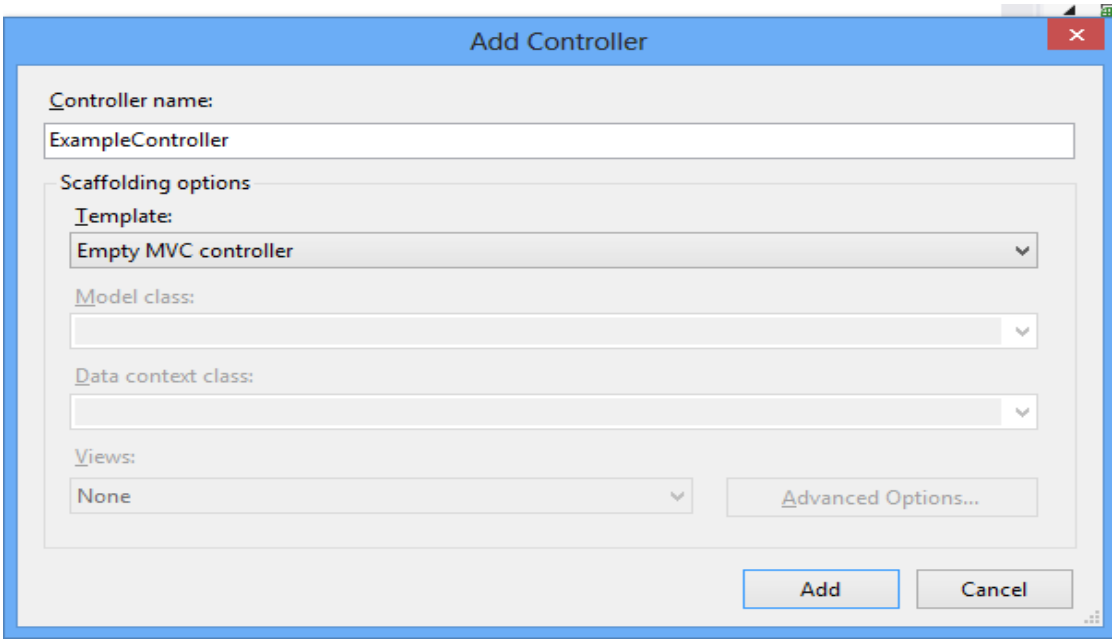
ان لم تعمل (Build) سوف تظهر لك هذه الرسالة :



التي تطلب منك ان تعمل (build) للمشروع ، وبعده عمله سوف تكون النافذه بهذا الشكل :



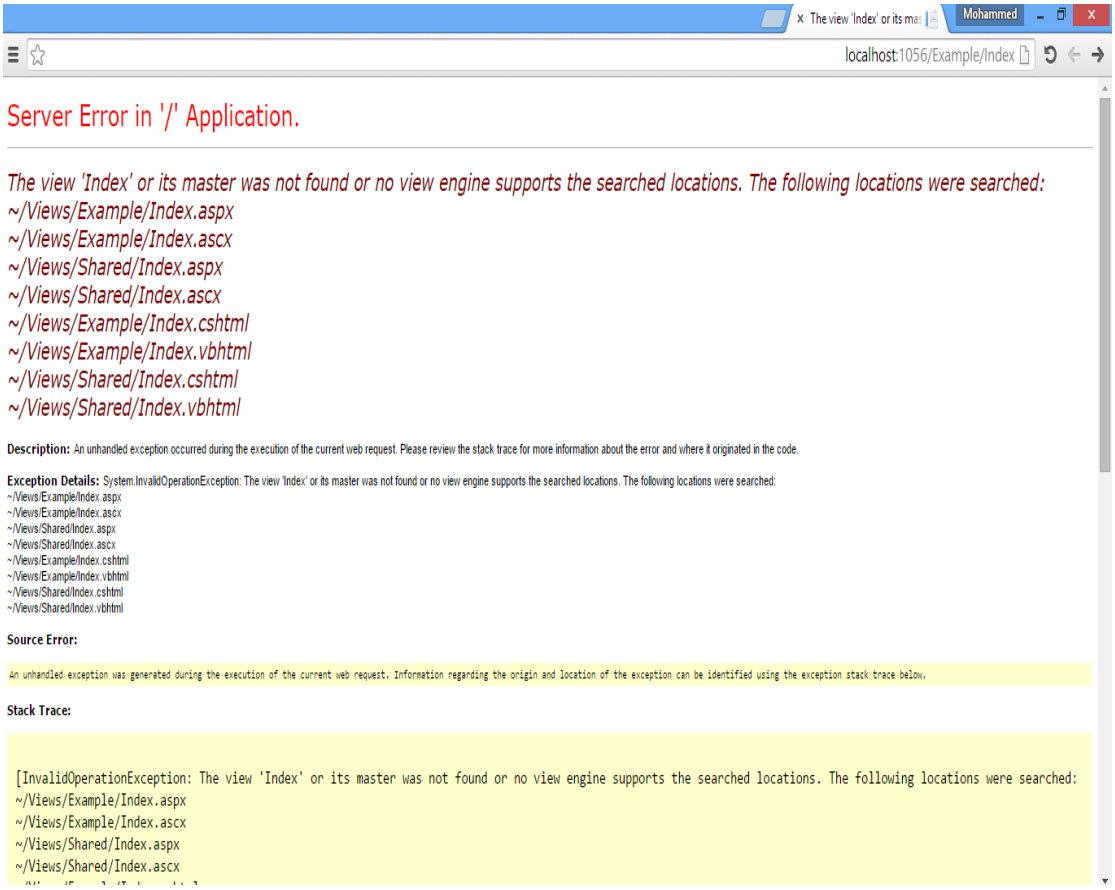
في حقل (Controller Name) نضع أسم الكونترولر ولاتحذف كلمة (Controller) وانما الكلمة المظلله فقط ، ثم من (Template) نختار (Empty MVC Controller) (سنتطرق للأنواع الاخرى لاحقاً) ثم Add .



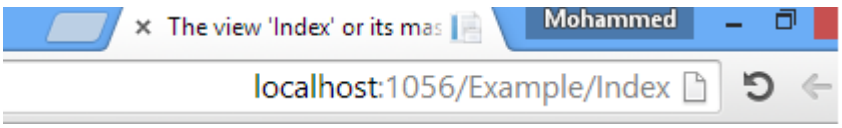
كما تلاحظ في الكود للكونترولر أدناه قد أضاف أكشن بأسم (Index) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace First.Controllers
{
    public class ExampleController : Controller
    {
        //
        // GET: /Example/

        public ActionResult Index()
        {
            return View();
        }

    }
}
```

لو نفذنا المشروع الان وأردنا عرض الكونترولر (Example) فانه سوف يظهر رسالة خطأ كما في الصورة أدناه ، لكن لماذا ؟



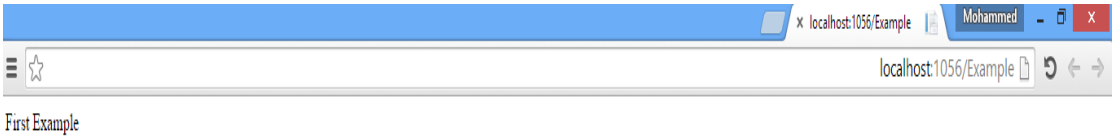
قبل ان أجيب عن هذا السؤال لاحظ كيف كتبنا الرابط من أجل عرض الاكشن (Index) :



لو لم نكتب (Index) في الرابط فأنه بصورة أفتراضية سوف يعرض الاكشن الذي يحمل أسم (Index) ، لكن المهم نعود للسؤال كما تلاحظ في الكود الخاص بالكونترولر أن نوع الاكشن (Action Result) وهذا النوع يجب ان يعيد (View) بمعنى انه سوف يقوم بعرض (View) أسمه نفس أسم الاكشن موجود في مجلد (Views\ Example) وبما أن هذا الـ (View) غير موجود فأنه سوف يعرض راسلة الخطأ التي تخبرنا بذلك .

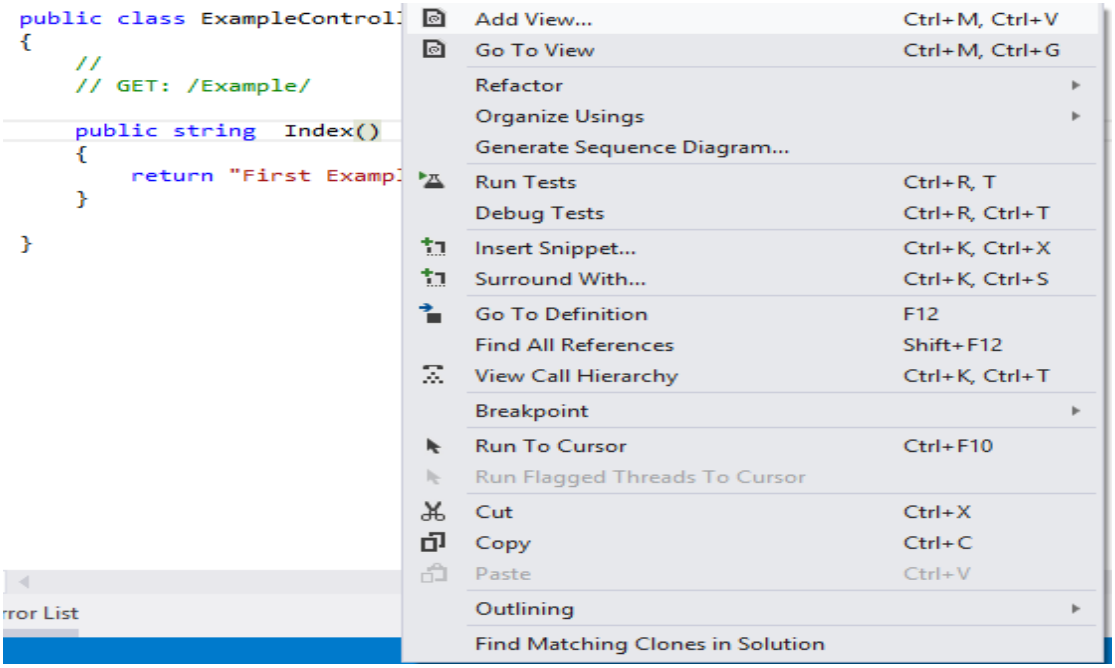
لو عدلنا على كود الاكشن (Index) وجعلناه يعيد نص بدلاً من (View) ونوعه (String) فلا تحصل أية مشكلة هكذا :

```
public string  Index()
        {
            return "First Example";
        }
```

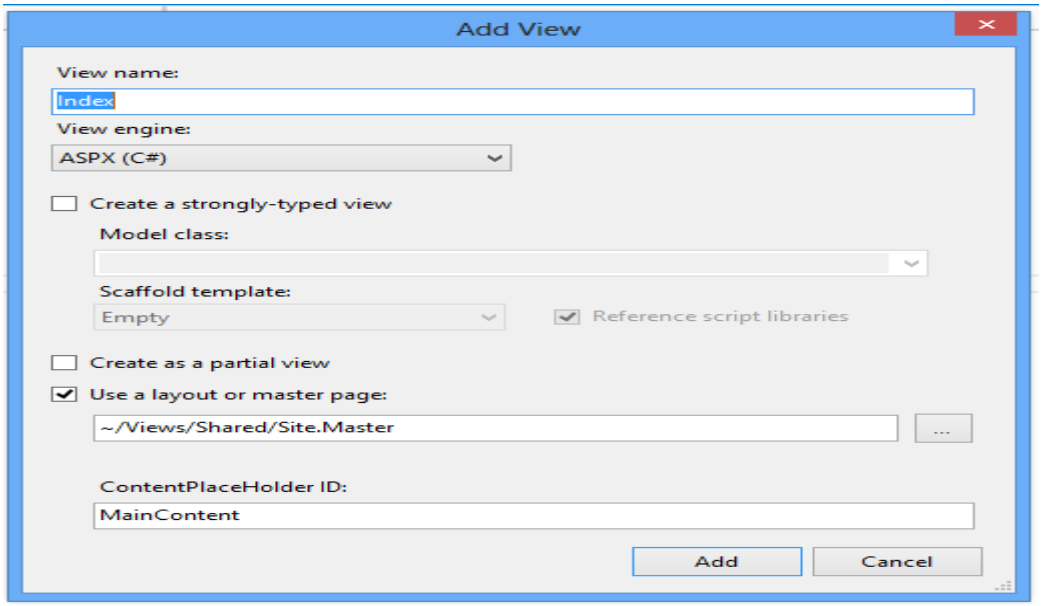


First Example

يجب علينا أن نعمل (View) لكل أكشن نريد عرضها ، ولعمل (View) للأكشن يكون بـ (Right Click) على الأكشن (Index) ونختار (Add View) :



ستظهر النافذة التالية :

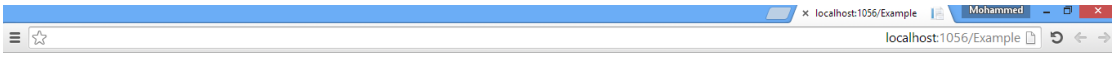


يجب ان يكون (View) بنفس أسم الأكشن ، وبما View Engine لمشروعنا هو Aspx نختارها ، ونترك الاعدادت البقية على حالها ، لكن فقط للتنويه بأن الخيار (Use Layout or Master Page) نختاره اذا اردنا ال(View) يرث من (Master Page) في حالة Aspx ، و من (Layout) في حالة (Razor) .

ننقر على Add ، ونرجع الكود الاصلي للأكشن (Index) ، ثم في (View) الجديد الذي أضفناه ويحمل اسم (Index) نعدل على محتوياته بمسحها ونكتب فيها :

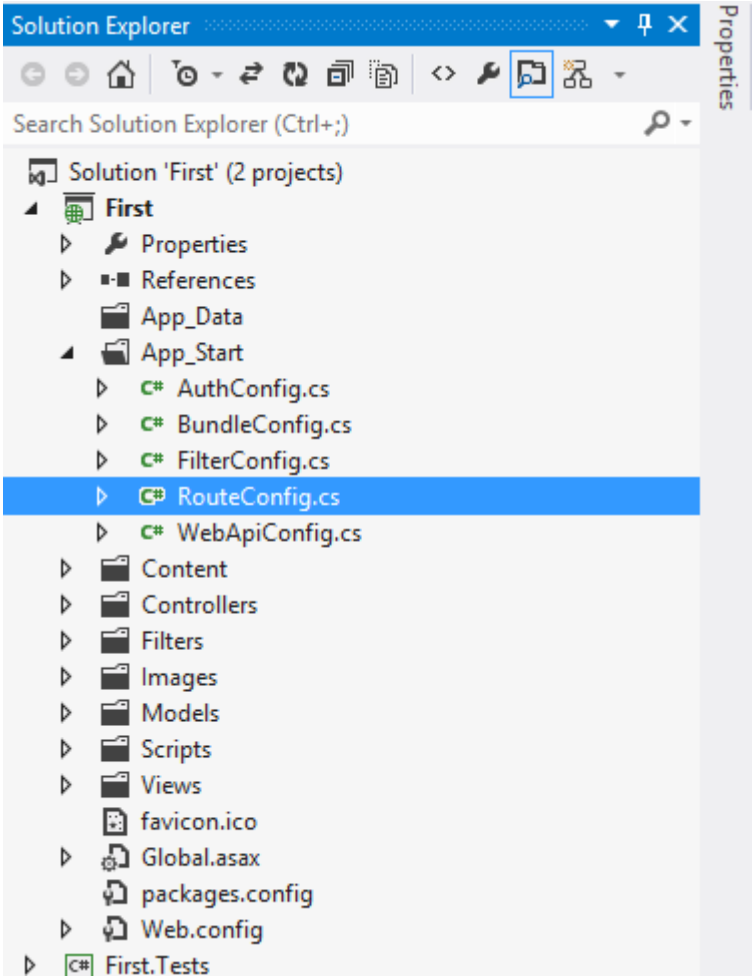
```
<h1> First Exampe in MVC</h1>
```

وبعد التنفيذ :



First Exampe in MVC

كما لاحظ عند تنفيذ المشروع فأنه سوف يعرض الكونترولر (Home) ولكي نجعله يعرض الكونترولر (Example) عند بداية التنفيذ من خلال التعديل على (RouteConfig) ، من المجلد (App_Start) نختار (RouteConfig.cs) .



وبدل (Home) نضع اسم الكونترولر الذي نريد ان يعرض في بداية تنفيذ المشروع مثلا سنضع (Example) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace First
{
    public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
```

```
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
        }
    }
}
```

كذلك حتى الاكشن نستطيع تغييره .

أذا كان نوع الأكشن (Action Result) فماهي الانواع التي يمثلها (بمعنى االنواع التي يعيدها) :

- ❖ View Result : يمثل (View) .
- ❖ Partial View Result : يمثل جزء من (View) .
- ❖ Redirect Result : يمثل التوجيه أما الى اكشن آخر او الى URL .
- ❖ Content Result : يمثل سطر محتويات ترسل الى المتصفح .
- ❖ JSON Result : يمثل JSON .
- ❖ File Result : يمثل ملف يمكن تنزيله (Download) .
- ❖ Empty Result : يمثل نتيجة فارغ من المحتويات ترجع للأكشن .
- ❖ HTTP Unauthorized Result : يمثل HTTP Unautherized .
- ❖ JAVAScript Result : يمثل ملف جافا سكربت .
- ❖ Redirect To Route Result : يمثل توجيه الى اكشن آخر او URL .

هذه الانواع سنتطرق لها كاثر فيما بعد ، كما تلاحظ إعلاه أن (Action Result) له أنواع عديده لكن نجعله يعيد لنا النوع الذي نريده ؟ الحل جدا بسيط عن طريق أستدعاء (Controller Method) لكل نوع ، على سبيل المثال لو أردنا ارجاع نوع (View Result) فأننا سنستدعي الميثود (View()) ، لنتعرف أكثر عن هذه (Methods) التي تعيد لنا (Action Result) وهي :

- View() : تعيد View Result .
- Partial View() : تعيد Partial View Result .
- Redirect To Action() : تعيد Redirect To Route Result .
- Redirect() : تعيد Redirect Result .
- Content() : تعيد Content Result .
- JSON() : تعيد JSON Result .
- File() : تعيد File Result .
- JavaScript() : تعيد JS result .

View Result :

هذا النوع بالاغلب يرجع لنا (View) وقصدي بيرجع يعني أن الاكشن عندما ينفذ فأنه سوف يستدعي (View) معين ويعرضه بالمتصفح ، والاستدعاء يكون على نوعين ، الاول يسمى بالاستدعاء الضمني (Implicitly) ويكون بهذا الشكل :

```
Return View();
```

لو أفترضنا ان اسم الاكشن (Create) وعند تنفيذه فأنه سوف يذهب الى المجلد (Views) وداخله يبحث عن المجلد الذي يحمل اسم الكونترولر الذي يقع فيه هذا الاشكن ثم يبحث بداخله عن (View) بأسم (Create) يبعثه للمتصفح ليعرضه ، ويكون هذا حسب نمط معين هو :

\Views\ {Controller Name} \ {Action Name }.aspx

مثال لو أنشئنا أكشن جديد بأسم (Impilictly) وأضفنا له (View) بنفس اسم الاكشن :

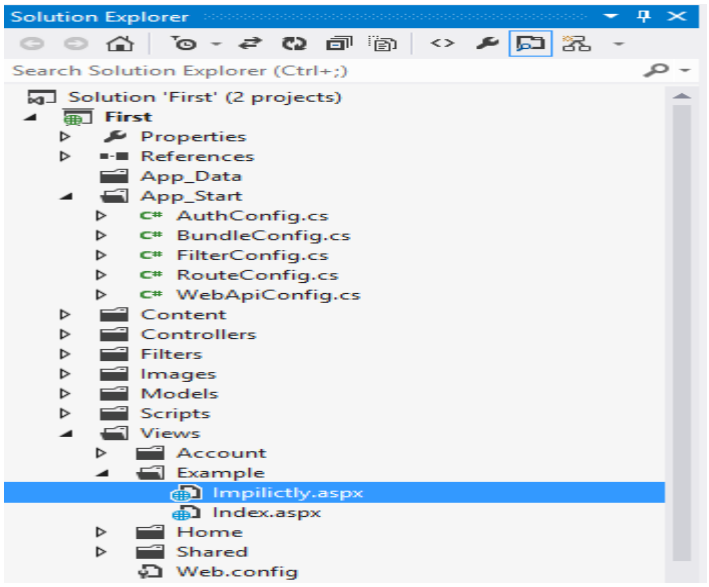
كود الأكشن :

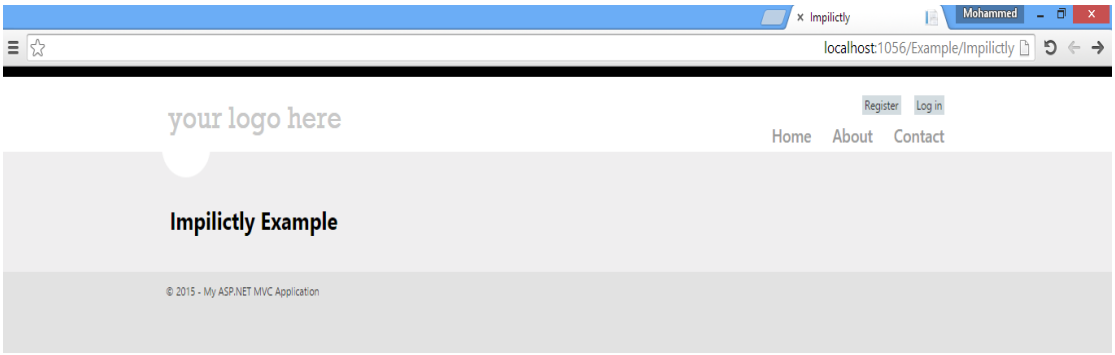
```
public ActionResult Impilictly()
        {
            return View();
        }
```

وكود الـ (View) :

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Impilictly
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

<h2>Impilictly Example </h2>

</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="FeaturedContent"
runat="server">
</asp:Content>

<asp:Content ID="Content4" ContentPlaceHolderID="ScriptsSection"
runat="server">
</asp:Content>
```

نفذ المشروع وشاهد الاكشن كيف أستدعى الـ(View) .





أما النوع الثاني يسمى بالاستدعاء الصريح (Explicitly) وهذا تكون صيغته :

```
Return View ("View Name") ;
```

بدون ذكر أمتداد الـ (View) بمعنى بدون أن تذكر (.aspx) ، بينما (View Name) يمكن أن يكون المسار النسبي(Relative) للـ (View) مثل ("First/Create") او يكون المسار التام (Absolute) للـ (View) مثلاً ("~/Create.aspx") .

مثال : سننشى أكشن ، باسم (Explicitly) :

```
public ActionResult Explicitly()
        {
            return View("Explicitly");
        }
```

ونضيف له (View) يكون بأسمه .

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<dynamic>" %>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Explicitly
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">

<h2>Explicitly</h2>

</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="FeaturedContent"
runat="server">
</asp:Content>

<asp:Content ID="Content4" ContentPlaceHolderID="ScriptsSection"
runat="server">
</asp:Content>
```

نفذ وشاهد .

هناك مجموعة من المعاملات (Parameters) يمكن تمريرها للـميثود (View()) هي :

١) ViewName : أسم الفيو .

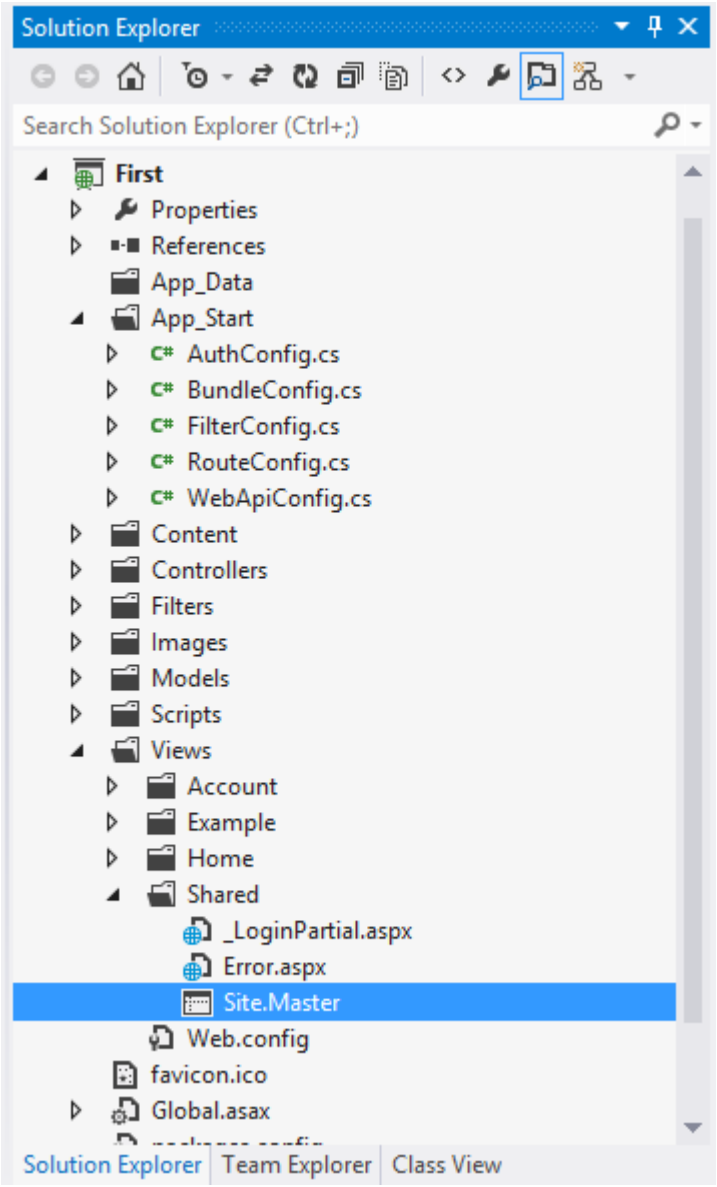
٢) MasterName : اسم الماستربيج أو (PageLayout) التي يرث منها الفيو .

٣) Mode Object : اسم الـ (Object Model) الذي يمرر للفيو .

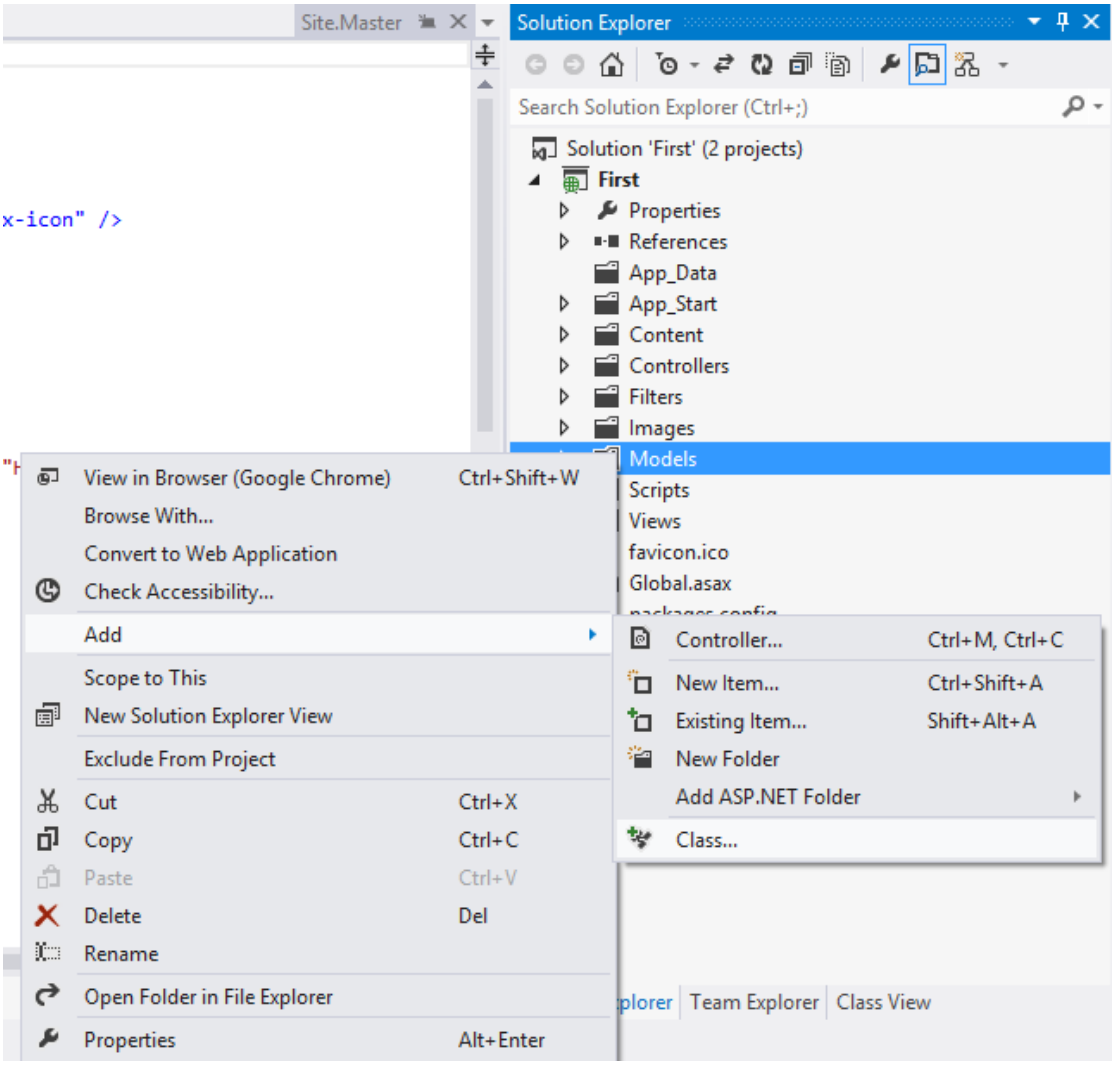
دعونا نعدل على الأكشن (Explicitly) ونمرر له (Master Name) :

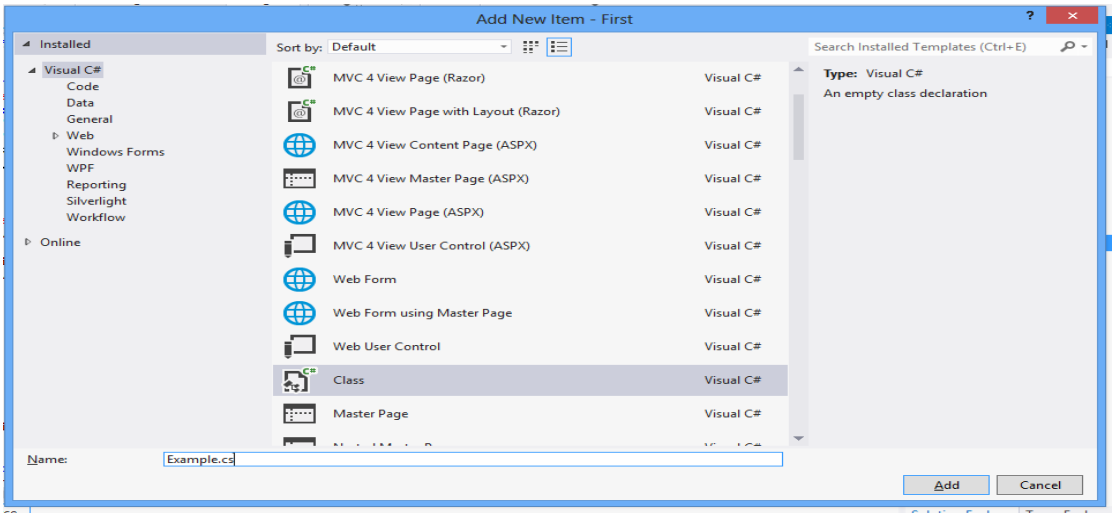
```
public ActionResult Explicitly()
        {
            return View("Explicitly","Site");
        }
```

لاحظ قد ممرت له أسم الماستر بيج بدون ذكر أمتدادها لانه بصورة أفتراضية يعرف مسارها ، والماستر بيج او (PageLayout) تكون في المجلد (Views) داخل مجلد ضمني أسمه (Shared) :



لكن لو أردنا تمرير (Model) للفيو يكون بصورة بسيطة الى حدآ ما ، اولا ننشئ مودل (عبارة عن كلاس عادي) بنفس أسم الكونترولر ، من خلال (RightClick) على المجلد (Models) ونختار (Add) ونختار (Class) :





لنعرف داخله ثلاثة متغيرات (تسمى بالخصائص) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
namespace First.Models
{
    public class Example
    {
        public int Id { get; set; }
        public string Name {get; set; }
        public int Age { get; set; }
    }
}
```

اما في الكونترولر (Example) يجب ان نستدعي المكتبة (Using First.Models ;) حتى نستطيع الوصول الى المودلز .

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using First.Models;
namespace First.Controllers
{
    public class ExampleController : Controller
    {
        //
        // GET: /Example/

        public ActionResult Index()
        {
            return View();
        }
        public ActionResult Impilictly()
        {
            return View();
        }
        public ActionResult Explicitly()
        {
            Example exm = new Example();
            exm.Id = 21;
            exm.Name = "Muhammed";
            exm.Age = 27;
            return View(exm );
        }

    }
}
```

وبالنسبة للفيو ، فيكون استدعاء المودل هكذا :

```
<H1><%= Html.Encode ( Model.Id.tostring() + ":" + Model .Name) %></H1>
    <p><%= Html .Encode ( Model.Age) %> </p>
```

كما تلاحظ فأننا استخدمنا (<% %>) لكتابة كود (C#) بينما لو كان (View Engine) هو (Razor) فأننا سنستخدم الرمز (@) لكتابة كود (C#) .

: Redirect To Action

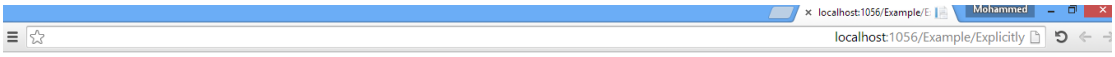
هذا النوع يقوم بالتوجيه من أكشن معين الى أكشن آخر سواء كانوا في نفس الكونترولر او في كونترولرز مختلفين، الصيغة تكون بهذا الشكل اذا كانوا بنفس الكونترولر :

```
Return Redirect to Action ("أسم الاكشن الموجه اليه") ;
```

لنأخذ مثال بسيط ، أعمل أكشن جديد في الكونترولر (Example) سمه (RTA) وعند تنفيذه سيوجهنا الى الأكشن (Explicitly) :

```
public ActionResult RTA()
        {
            return RedirectToAction("Explicitly");
        }
```

لو نفذنا المشروع وكتبنا رابط الاكشن فأنه سوف يعرض لنا الاكشن (Explicitly) .



21:Muhammed

فقط للتنويه ، الرابط الذي تكتبه في المتصفح غير حساس للأحرف ، بمعنى سواء كتبت اسم الاكشن بأحرف كبيرة (RTA) أو صغيرة (rta) فأنه سوف يفهمهم .

لو أردنا التوجية من أكشن في كونترولر معين الى أكشن في كونترولر آخر فأننا سنستخدم هذه الصيغة :

```
Return RediectToAction ( " أسم الاكشن ", "أسم الكونترولرز " ) ;
```

لو عدلنا على كود الأكشن (RTA) وجعلناه يوجهنا الى الاكشن (Index) في الكونترولر (Home) الذي أضافة لنا الفيجوال ستوديو بصورة أفتراضية .

```
public ActionResult RTA()
        {
            return RedirectToAction("Index","Home");
        }
```

نفذ المشروع واكتب رابط الاكشن (RTA) في المتصفح وستلاحظ أنه يوجهك الى الاكشن (Index) .

أذا المعاملات (Parameters) التي يمكن تمريرها لهذه (method) هي :

١. Action Name : أسم الاكشن .
٢. Controller Name : أسم الكونترولر .
٣. Route Value : هذا المعامل يفيدنا بتمرير قيمة للـ (Id) لأشكن آخر . لنأخذ مثال بسيط عليه :.

لو عدلنا على كود الاكشن (Explicitly) وأضفنا له معامل :

```
public ActionResult Explicitly(int id=0)
        {
            Example exm = new Example();
            exm.Id = id;
            exm.Name = "Muhammed";
            exm.Age = 27;
            return View(exm );
        }
```

المعامل (id) من نوع int وأعطيناه قيمة أبتدائية قيمتها صفر حتى عند الاستعراض لا يحصل خطأ ، اما كود (RTA) سيكون :

```
public ActionResult RTA()
        {
            return RedirectToAction("Index","Home", new { id = 53 });
        }
```

كما تلاحظ فأنه سيقوم بتوجيهنا الى الاشكن (Index) في الكونترولر (Home)، كذلك نعدل على الاكشن (Index) وذلك بجعله ياخذ معامل (id) :

```
public ActionResult Index(int id)
        {
            ViewBag.Message = "Modify this template to jump-start your
ASP.NET MVC application.";
            ViewData["id"] = id;
            return View();
        }
```

واسندنا قيمة id للـ (ViewData) (سنتكلم عليها أكثر لاحقآ) لكن بأختصار هي عبارة عن متغير (مصفوفة) ديناميكيه بمعنى أنها سوف تأخذ البيانات تلقائيا من الكونترولر الى الفيو .

اما الفيو (Index) سيكون هكذا بعد التعديل عليه :

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage" %>

<asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent"
runat="server">
    Home Page - My ASP.NET MVC Application
</asp:Content>

<asp:Content ID="indexFeatured" ContentPlaceHolderID="FeaturedContent"
runat="server">
    <section class="featured">
        <div class="content-wrapper">
            <hgroup class="title">
                <h1>Home Page.</h1>
                <h2><%: ViewBag.Message %></h2>
            </hgroup>
            <p>
                To learn more about ASP.NET MVC visit
                <a href="http://asp.net/mvc" title="ASP.NET MVC
```

```
Website">http://asp.net/mvc</a>.
                The page features <mark>videos, tutorials, and samples</mark>
to help you get the most from ASP.NET MVC.
                If you have any questions about ASP.NET MVC visit
                <a href="http://forums.asp.net/1146.aspx/1?MVC"
title="ASP.NET MVC Forum">our forums</a>.
            </p>
        </div>
    </section>
</asp:Content>

<asp:Content ID="indexContent" ContentPlaceHolderID="MainContent"
runat="server">

    <h1><%=ViewData["id"] %></h1>
</asp:Content>
```

بعد الاستعراض ستشاهد النتيجة .

: Content Result

هذا النوع يرجع محتويات مثل نص او تاريخ او رقم ، حيث أن MVCFramework تقوم بتحويل اي قيمة ليست Action Result الى هذا النوع : مثال سنقوم بأضافة أكشن اسمه (Hello) وهذا الاكشن لا يحتاج ان نعمل له فيو لأنه سوف يعيد نص للمتصفح ولا حاجة للفيو.

```
public ActionResult Hello()
        {
            return Content("Hi This is first example ");
        }
```

اما بالنسبة للمعاملات (Parameters) التي تتقبلها هذه الميثود فهي :

- String: نص عادي المثال اعلاه .
- Content Type : نص من نوع HTMl او غير ذلك .

```
    public ActionResult Hello()
            {
                return Content("<h1>Hi This is first example</h1> ");
            }
```

- Content Encoding: مثل Ascii .

: File Result

هذا النوع يرجع لنا ملف ، ويفيدنا عندما نريد عمل Download لملف معين وطريقته جدا بسيطة .

لنأخذ مثال بسيط ، أعمل مجلد في المشروع سميه (Files) وضع فيه ملف pdf بسيط :

```
public ActionResult Download()
        {
            return File("~/Files/Muhammed.pdf", "application/pdf",
"muhammed_Download");
        }
```

لايحتاج الى عمل (View) له .

هذا الميثود صيغته كالتالي :

```
Return File ( "File Name" , "Content Type" , "File Download Name" ) ;
```

هذه الصيغة تفيدنا لو كان الملف مخزون بمجلد ومساره مخزون في قاعدة البيانات والمعاملات هي :

- File Name : مسار الملف كامل (يعني مع اسم الملف وأمتداده) .
- Content Type : نوع الملف .
- File DownLoad Name : اسم الملف عند تنزيلة .

بالنسبة للـ (Content Type) هذه قائمة بانواع كل الملفات :

Suffixes applicable	Media type and subtype(s)
.au	audio/basic
.avi	video/msvideo, video/avi, video/x-msvideo
.bmp	image/bmp
.bz2	application/x-bzip2
.css	text/css
.dtd	application/xml-dtd
.doc	application/msword

Suffixes applicable	Media type and subtype(s)
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document
.dotx	application/vnd.openxmlformats-officedocument.wordprocessingml.template
.es	application/ecmascript
.exe	application/octet-stream
.gif	image/gif
.gz	application/x-gzip
.hqx	application/mac-binhex40
.html	text/html
.jar	application/java-archive
.jpg	image/jpeg
.js	application/x-javascript
.midi	audio/x-midi
.mp3	audio/mpeg
.mpeg	video/mpeg
.ogg	audio/vorbis, application/ogg
.pdf	application/pdf
.pl	application/x-perl
.png	image/png
.potx	application/vnd.openxmlformats-officedocument.presentationml.template
.ppsx	application/vnd.openxmlformats-officedocument.presentationml.slideshow
.ppt	application/vnd.ms-powerpointtd>

Suffixes applicable	Media type and subtype(s)
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation
.ps	application/postscript
.qt	video/quicktime
.ra	audio/x-pn-realaudio, audio/vnd.rn-realaudio
.ram	audio/x-pn-realaudio, audio/vnd.rn-realaudio
.rdf	application/rdf, application/rdf+xml
.rtf	application/rtf
.sgml	text/sgml
.sit	application/x-stuffit
.sldx	application/vnd.openxmlformats-officedocument.presentationml.slide
.svg	image/svg+xml
.swf	application/x-shockwave-flash
.tar.gz	application/x-tar
.tgz	application/x-tar
.tiff	image/tiff
.tsv	text/tab-separated-values
.txt	text/plain
.wav	audio/wav, audio/x-wav
.xlam	application/vnd.ms-excel.addin.macroEnabled.12
.xls	application/vnd.ms-excel
.xlsb	application/vnd.ms-excel.sheet.binary.macroEnabled.12
.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet

Suffixes applicable	Media type and subtype(s)
.xltx	application/vnd.openxmlformats-officedocument.spreadsheetml.template
.xml	application/xml
.zip	application/zip, application/x-compressed-zip

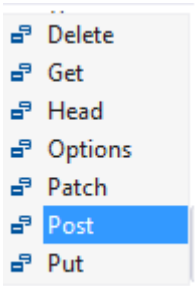
أما في حالة كان الملف مخزون كبايتات في قاعدة البيانات فأننا سوف نستخدم File Type Content بدل من مسار الملف ، وكذلك بدل من مسار الملف نسخدم File Stream .

هناك مجموعة من المشاكل ستواجهنا في الاكشن ، على سبيل المثال لو كان لديك 2 Actions بنفس الاسم ، واردت تنفيذ أحدهما ؟ MVC تتجاوز هكذا مشكلة وذلك من خلال خاصية تسمى Accept Verbs وهذه الخاصية تمكنك من منع أكشن من التنفيذ الى أن تنجز فعالية HTTP وهذه الفعاليات مثل (GET , Post) :

مثلا لو كان لديك ٢ أكشن بأسم Create ، وأردت الاولى ان تعمل مع فعالية HTTP Get ، والاخرى مع HTTP Post فيكون :

```
[AcceptVerbs(HttpVerbs.Get)]
        public ActionResult Create()
        {
            return View();
        }
        [AcceptVerbs(HttpVerbs.Post)]
        public ActionResult Create()
        {
            return View();
        }
```

هناك سبعة فعاليات HTTP هي :



وللمزيد عن فعاليات (HTTP) (http://www.w3schools.com/tags/ref_httpmethods.asp) .

هناك خاصية مهمة أسمها ((Action Name)) تمكنك من جعل أسم الاكشن مختلف عن أسم (Method) وهناك حالتين نستخدمها فيه :

الحالة الاولى : لو كان لديك كونترولر فيه ٢ Method بنفس الاسم ، تستطيع من خلال هذه الخاصية ان تجعل اسم الاكشن لهما مختلف .

مثال :

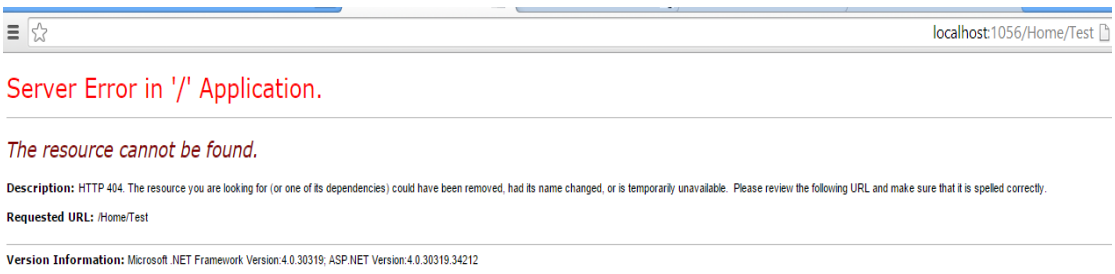
```
        [ActionName ("Delete1")]
        [AcceptVerbs (HttpVerbs.Get )]
        public ActionResult Delete()
        {
            return View();
        }
        [ActionName("Delete2")]
        [AcceptVerbs (HttpVerbs.Post )]
        public ActionResult Delete(int id)
        {
            return View();
        }
```

الحالة الثانية : لو كان لديك كونترولر فيه ٢ مثيثود بأسماء مختلفة ، تقدر من خلال هذه الخاصية ان تجعل لهما اسم اكشن متشابه .

مثال :

```
[ActionName("Edit")]
        [AcceptVerbs (HttpVerbs .Get)]
        public ActionResult Update1()
        {
            return View();

        }

        [ActionName ("Edit")]
        [AcceptVerbs (HttpVerbs.Post )]
        public ActionResult Update2(int id)
        {
            return View();
        }
```

بقي شي أخير ، وهو ماذا يحدث لو كتبنا بالمتصفح أسك أكشن غير موجود ؟ مثلا (URL /Home/Test) بصورة أفتراضية الكونترولر فيه ميثود أسمها (HandleUnknownAction()) حيث هذه الميثود تستدعى بصورة تلقائية عندما يكون الاكشن المطلوب تنفيذه غير موجود . ودائما تعرض لنا رسالة (٤٠٤) .



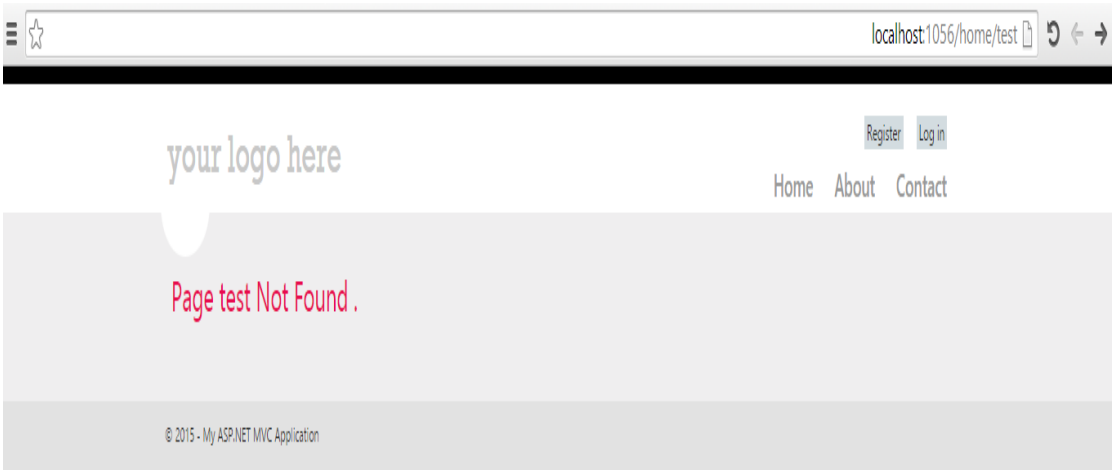
يمكن التعديل على هذه الميثود وعرض من خلالها رساله معينه مثلما نريد ، دعونا نجربها في الكونترولر (Home) :

```
protected override void HandleUnknownAction(string actionName)
        {

            ViewData["ActionName"] = actionName;
            View("Error").ExecuteResult(this.ControllerContext);
        }
```

نذهب الى الفيو (error) (هذا الفيو مخصص للأخطاء) ونجعله يعرض (View Data) والرسالة التي نريدها .

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<System.Web.Mvc.HandleErrorInfo>" %>

<asp:Content ID="errorTitle" ContentPlaceHolderID="TitleContent"
runat="server">
    Error - My ASP.NET MVC Application
</asp:Content>

<asp:Content ID="errorContent" ContentPlaceHolderID="MainContent"
runat="server">
    <hgroup class="title">
             <h2 class="error">Page<%= ViewData["ActionName"] %> Not Found
.</h2>
    </hgroup>
</asp:Content>
```

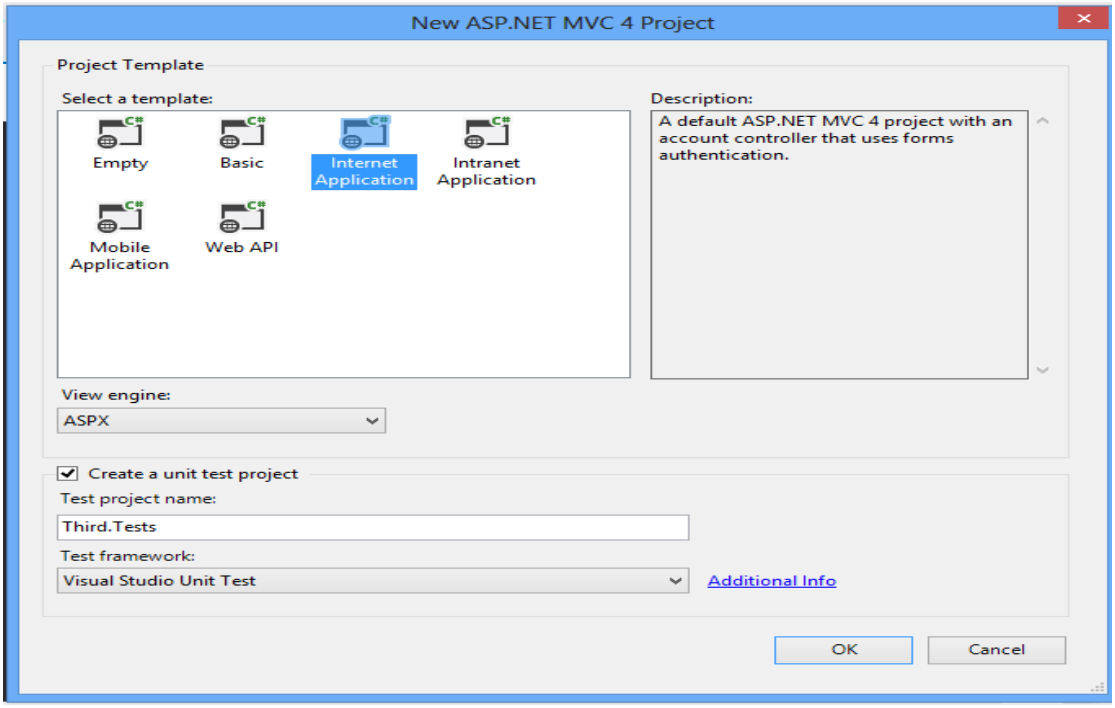
جرب بالمتصفح تكتب الاكشن (test) :



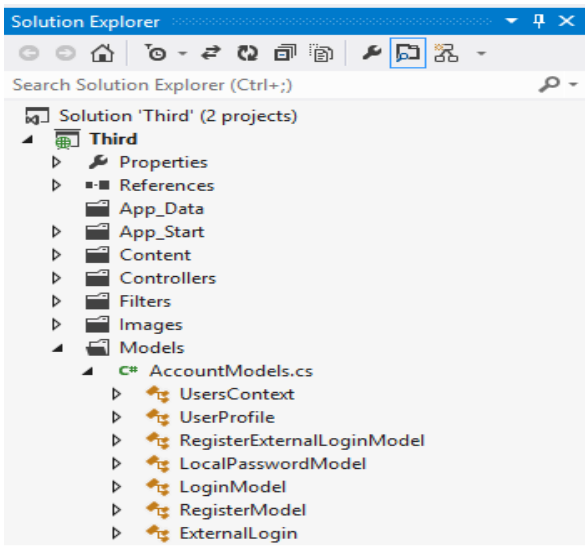
: Models

هو عبارة عن كائن عادي (Object) له خصائص (Properties) وسلوكيات (Behavior) ، يمثل (Business Logic) و (Data Logic) ، مثلاً يمكن أستخدامه لأرسال واسترجاع البيانات من قاعدة البيانات .

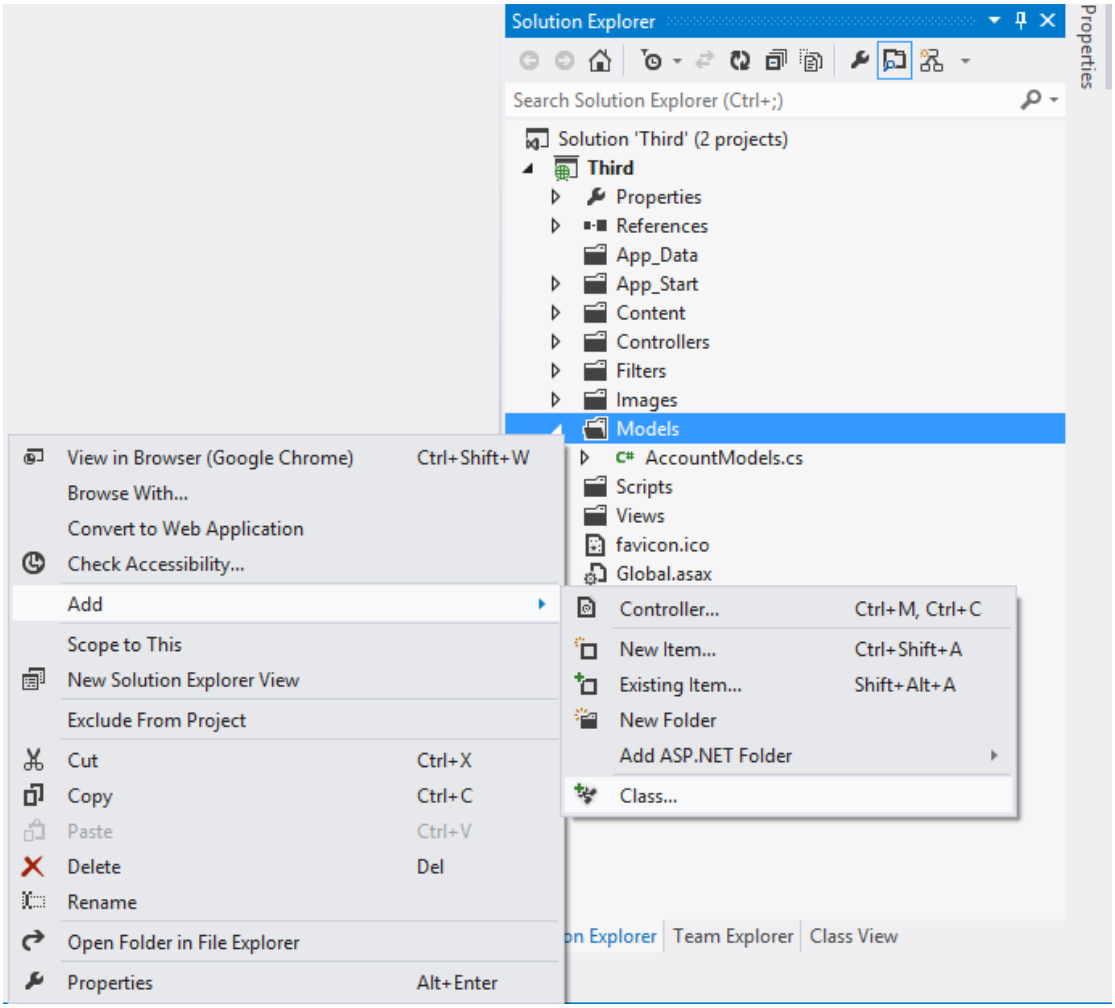
لنعمل مشروع جديد بأسم (Third) :



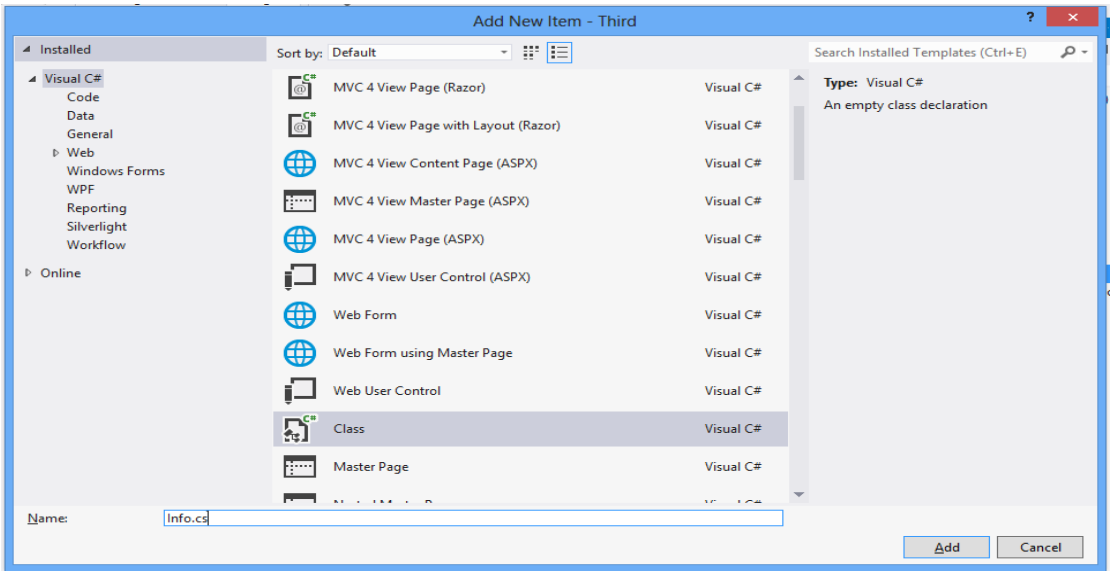
بصورة أفتراضية فأن الفيجوال ستوديو سيعمل لك مجلد (Models)يحوي فقط على مودل واحد (AccountModels) وهذا المودل خاص بـ (Authentication) :



كما تلاحظ اللون الاخضر يشير بأن هذا مودل ، والجوزي الفاتح يشير الى ان هذا (Method) ، دعونا نضيف مودل نسميه (Info.cs) ، من خلال (Right Click) على المجلد (Models) وثم (Add) نختار (Class) :



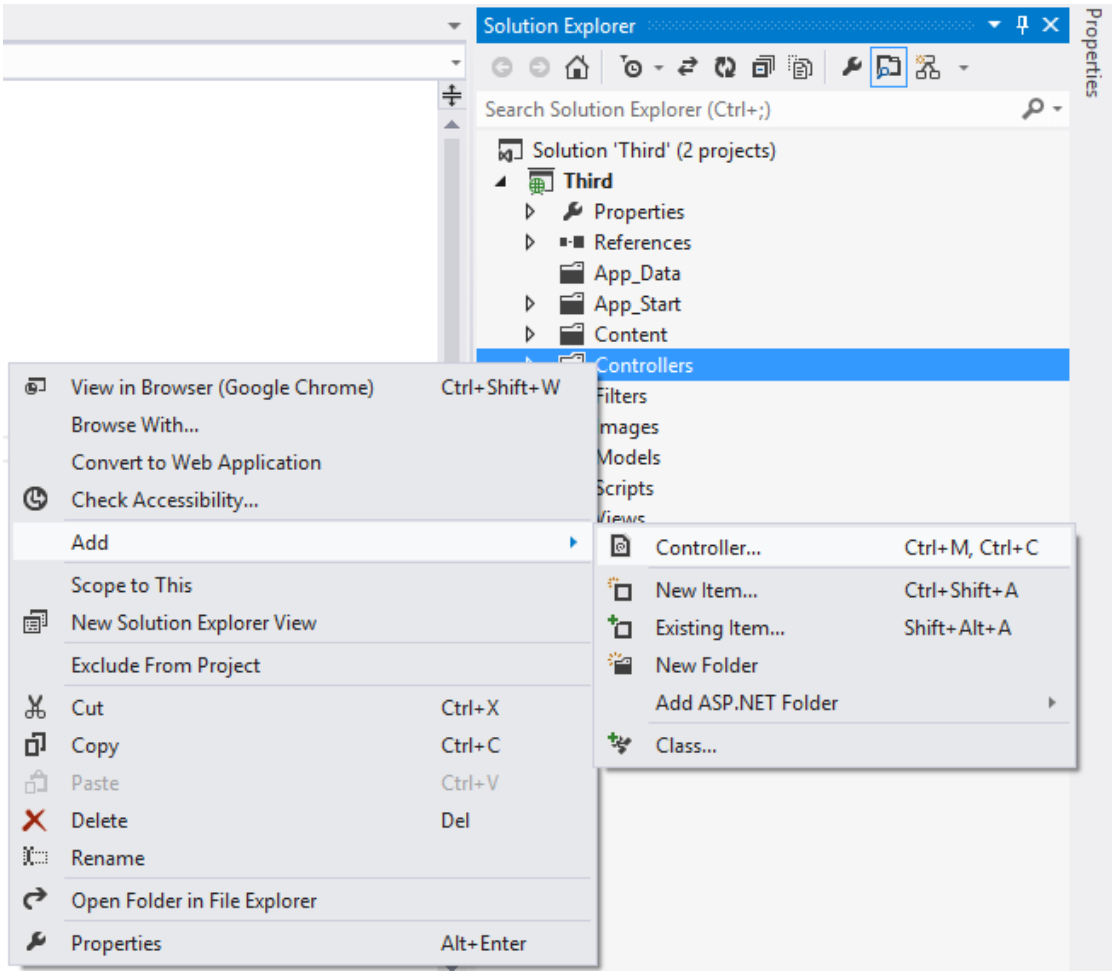
ثم :



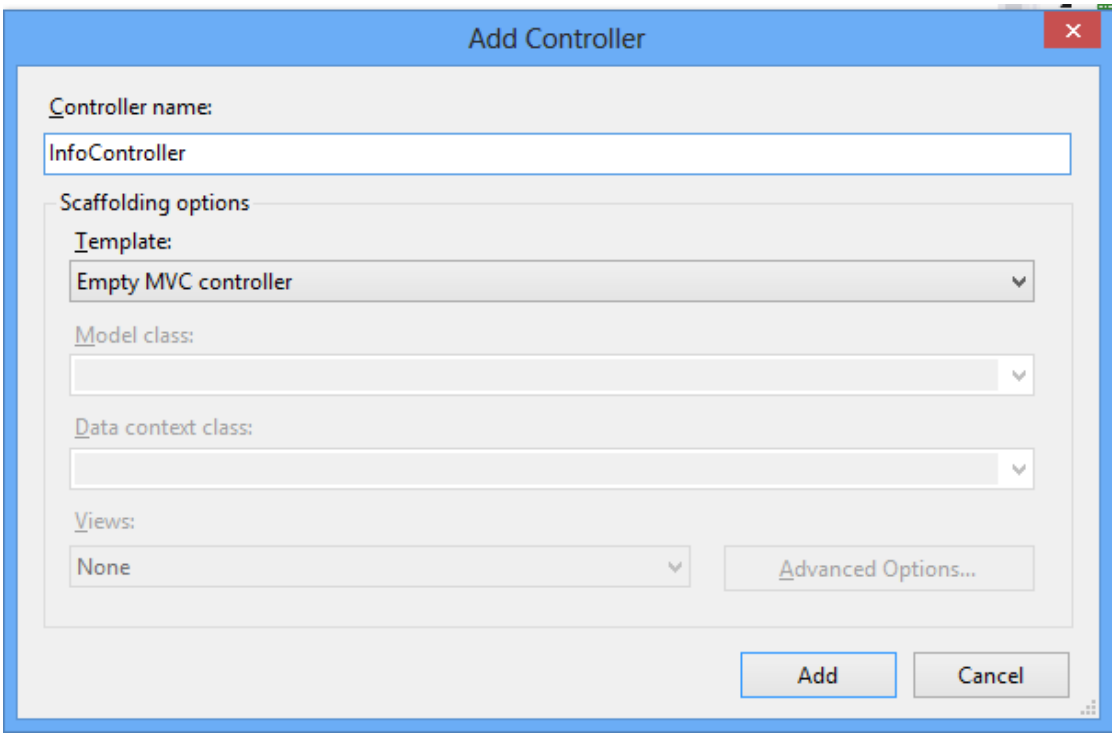
سنضيف فيه ٣ متغيرات (خصائص) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace Third.Models
{
    public class Info
    {
        public int ID { get; set; }
        public string Category { get; set; }
        public DateTime Create_Date { get; set; }

    }
}
```

ثم نضيف كونترولر ، (Right Click) على المجلد (Controllers) ونختار (Add) ثم (Controller) :



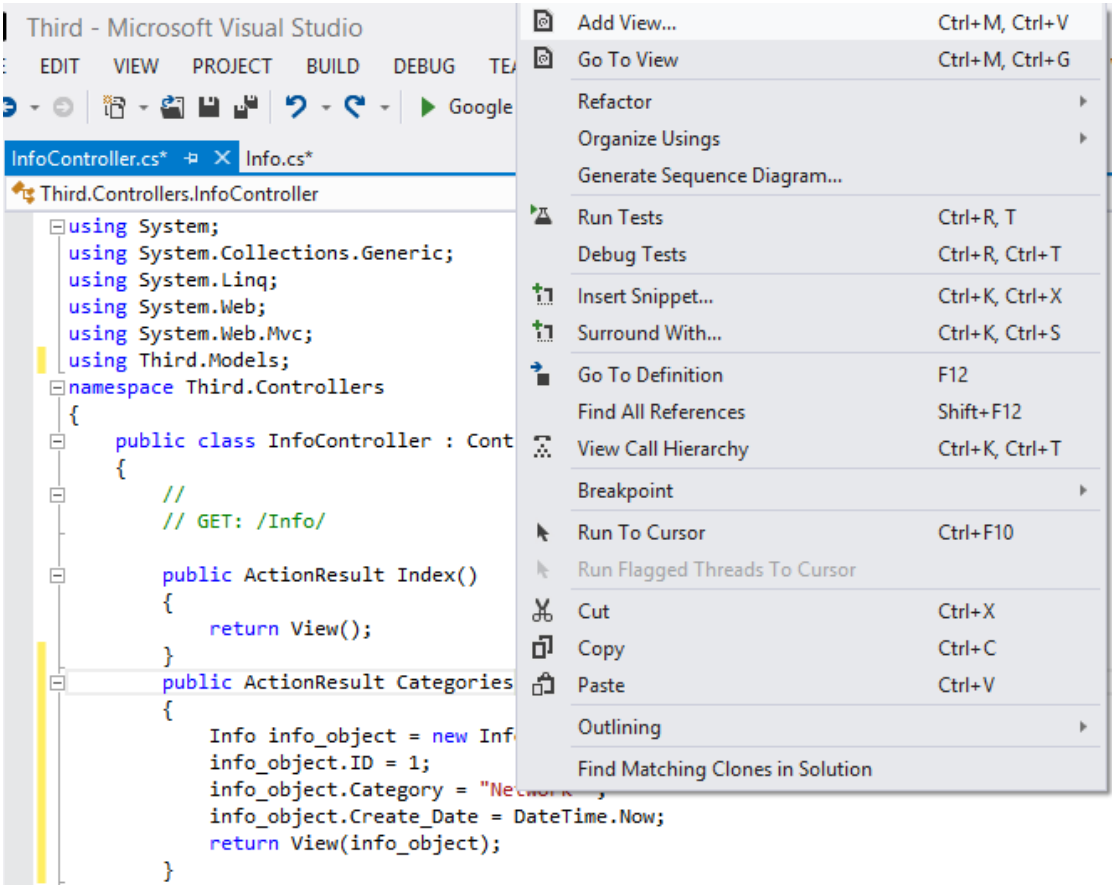
يجب ان يكون الكونترولر بنفس أسم المودل (Info) :



نعمل أكشن جديد (Categories) ونعرف داخله (Instance) من المودل (Info) لكي نملأه بالمعلمات .

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Third.Models;
namespace Third.Controllers
{
    public class InfoController : Controller
    {
        //
        // GET: /Info/

        public ActionResult Index()
        {
            return View();
        }
        public ActionResult Categories()
        {
            Info info_object = new Info();
            info_object.ID = 1;
            info_object.Category = "Network ";
            info_object.Create_Date = DateTime.Now;
            return View(info_object);
        }

    }
}
```

كما تلاحظ بالكود أعلاه بعد أن ملأنا المودل بالبيانات قمنا بأرساله للفيو ، الان يجب أن نعمل فيو بأسم الاكشن (Categories) ، من (Right Click) على الاكشن ونختار (Add View) :



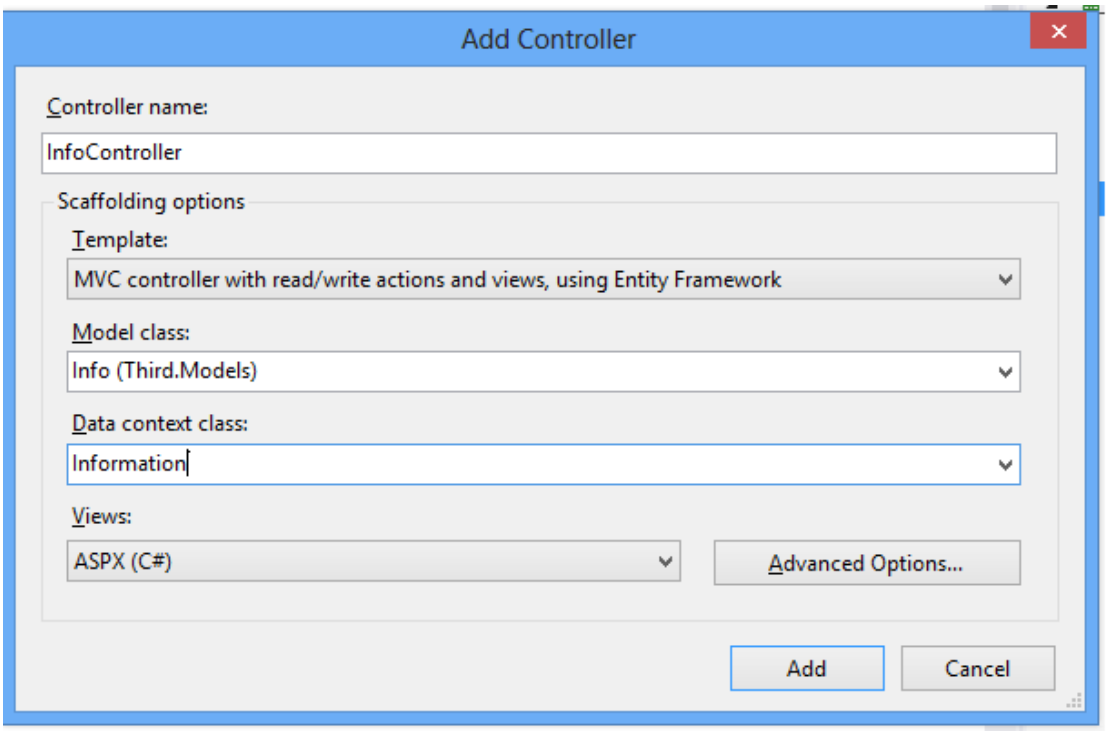
ونستدعي المودل في الفيو :

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Categories
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h1> <%= Model .ID +" : "+Model .Category %></h1>
    <br />
    <p><%= Model .Create_Date %></p>
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="FeaturedContent"
runat="server">
</asp:Content>

<asp:Content ID="Content4" ContentPlaceHolderID="ScriptsSection"
runat="server">
</asp:Content>
```

بعد الأستعراض سيكون العرض هكذا :

حتى هنا أعتقد الامور بسيطة ، لو تعمقنا أكثر فأنها سوف تزداد تعقيداً بعض الشي ، دعونا نعمل مثال آخر لكن هذه المرة سوف نبني قاعدة بيانات فيها جدول (Info) ، ولكي نبني قاعدة بيانات امامنا طريقين كلاهما يوصلنا لنفس الهدف ، الطريق الاولى أما ان ننشئ قاعدة بيانات وفيها جدول يدوياً ثم نستخدم تقنية (Entity Framework) لتولد لنا مودل من نفس الجدول ، او نستخدم (Code First) وهي أن نكتب كود المودل وأعتماداً عليه يتم توليد جدول اوتوماتيكياً .

الان لو أردنا أستخدام (Code First) وذلك ببناء جدول في قاعدة البيانات من المودل يكون من خلال أضافة كونترولر ، لكن قبل هذا يجب ان نحذف الكونترولر (Info) ثم (Right Click) على المجلد (Controller) ونضيف كونترولر جديد . :

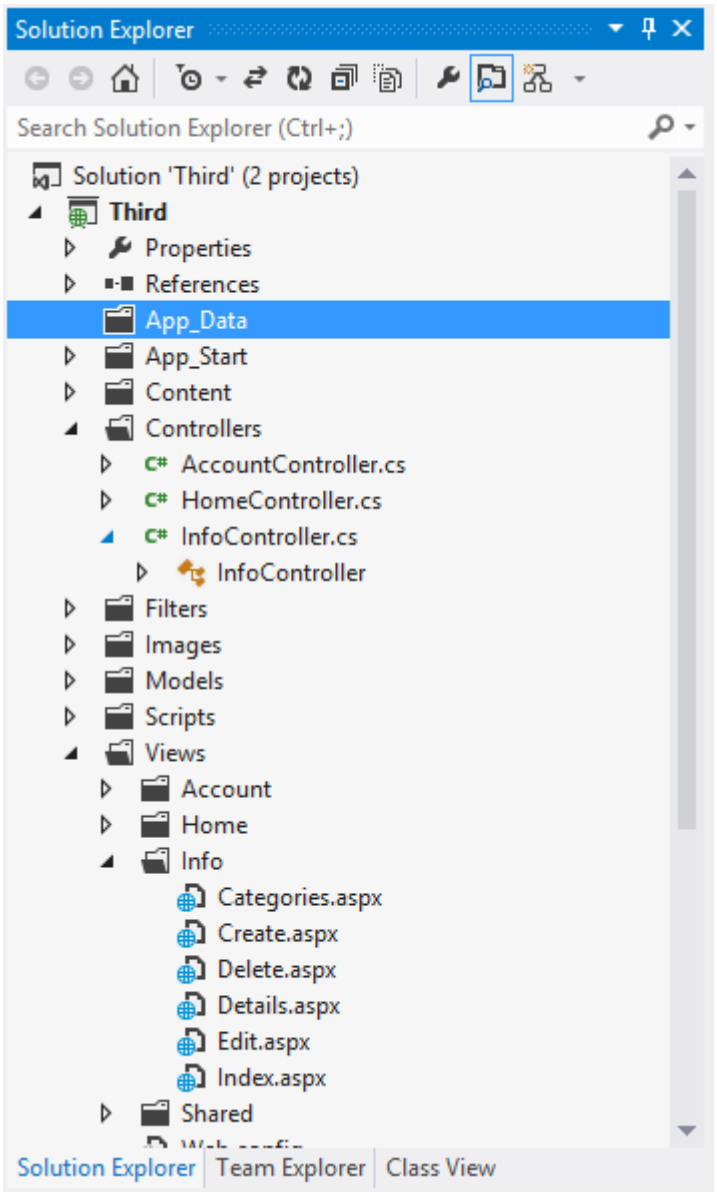


كما تلاحظ فأن الكونترولر(Info) سنختار له من (Template) (MVC Controller with read /write action views, using entity Framework) بينما قمنا بتسمية (Data Context Class) له (Information)

وهي تمثل اسم قاعدة البيانات ، وأخترنا المودل (Info) .

انقر على (add) :

كما ستلاحظ انه قام بأضافة كونترولر من المودل الموجود فيه عدة أكشن وكذلك أضاف لكل أكشن فيو خاص به : .



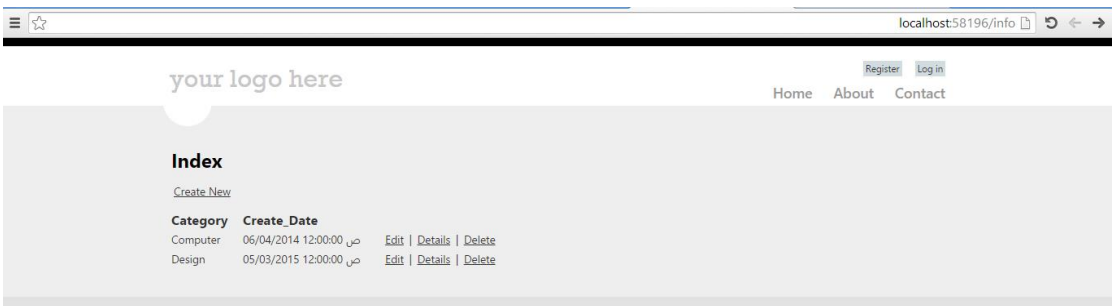

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Third.Models;

namespace Third.Controllers
{
```

```
public class InfoController : Controller
{
    private Information db = new Information();

    //
    // GET: /Info/

    public ActionResult Index()
    {
        return View(db.Infoes.ToList());
    }

    //
    // GET: /Info/Details/5

    public ActionResult Details(int id = 0)
    {
        Info info = db.Infoes.Find(id);
        if (info == null)
        {
            return HttpNotFound();
        }
        return View(info);
    }

    //
    // GET: /Info/Create

    public ActionResult Create()
    {
        return View();
    }

    //
    // POST: /Info/Create

    [HttpPost]
    public ActionResult Create(Info info)
    {
        if (ModelState.IsValid)
        {
            db.Infoes.Add(info);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        return View(info);
    }

    //
    // GET: /Info/Edit/5

    public ActionResult Edit(int id = 0)
    {
        Info info = db.Infoes.Find(id);
        if (info == null)
        {
            return HttpNotFound();
        }
        return View(info);
    }
```

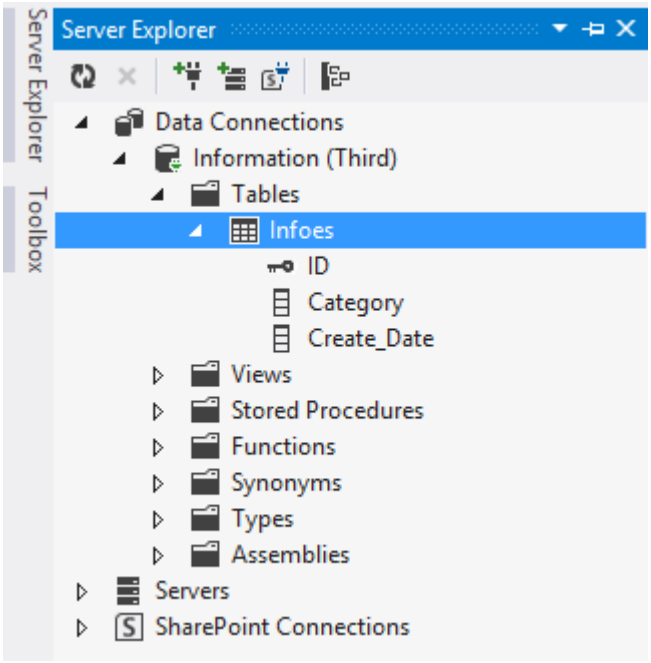
```
        //
        // POST: /Info/Edit/5

        [HttpPost]
        public ActionResult Edit(Info info)
        {
            if (ModelState.IsValid)
            {
                db.Entry(info).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(info);
        }

        //
        // GET: /Info/Delete/5

        public ActionResult Delete(int id = 0)
        {
            Info info = db.Infoes.Find(id);
            if (info == null)
            {
                return HttpNotFound();
            }
            return View(info);
        }

        //
        // POST: /Info/Delete/5

        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int id)
        {
            Info info = db.Infoes.Find(id);
            db.Infoes.Remove(info);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

وبالامكان التعديل على أي أكشن .

بعد التنفيذ :



بأمكانك الان ان تضيف وتعدل البيانات وحتى الحذف ، وقد أضاف لنا قاعدة بيانات بأسم (Information) وهو أسم (Data Context) التي وضعناها سابقآ وفيها جدول (Infoes) :



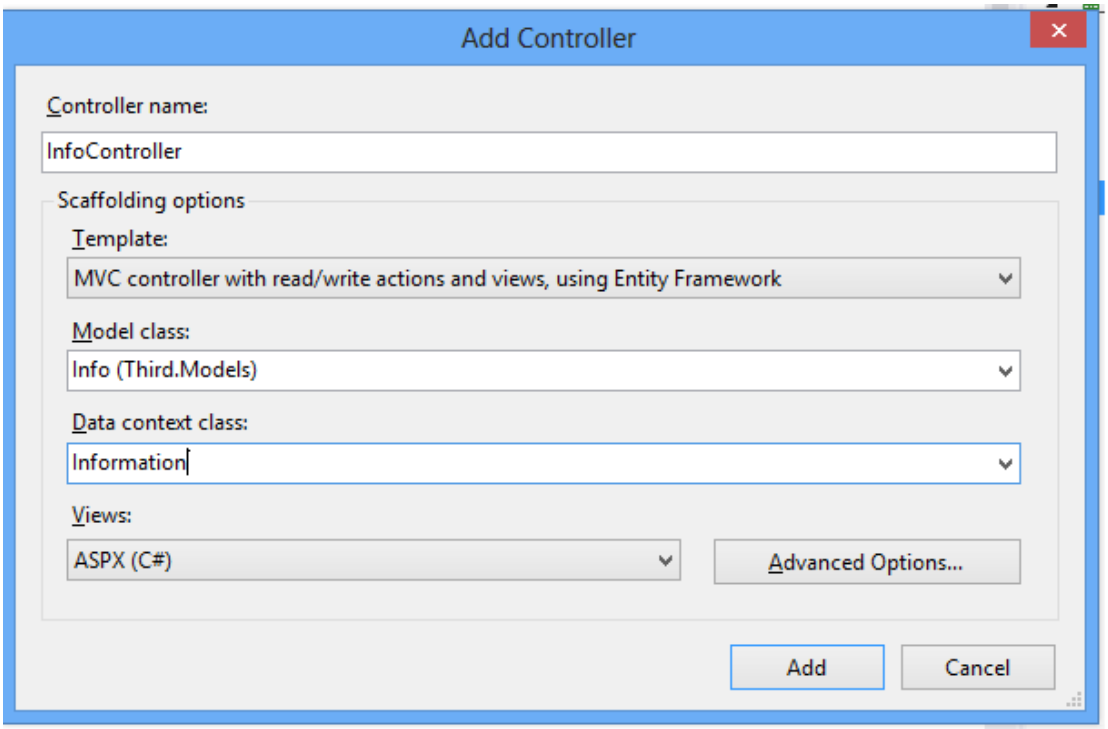
هناك عدة أمور ربما تكون غامضة في المثال أعلاه ، سنتطرق لبعضها الان ومنها :

Entity Framewrok :

الـ (MVC) ليس فيها نوع محدد من (Data Access) تستخدمها بصورة مباشرة للتعامل مع قاعدة البيانات وأنما هناك العديد من مكتبات (Data Access) بأمكانك أن تستخدمها للعمل مع قاعدة البيانات واحده من هذه المكتبات هي (Entity Framework) ومختصرها (EF) وهذه الاخيرة تمتاز بالمرونة وتساعد المطوير بالاستعلام والتعديل على البيانات المخزنة في قاعدة البيانات بطريقة الكائنات الموجهة (Object Oriented) ، وهي جزء من بيئة (.Net Framework) ، الـ (EF) تعطينا عدة طرق تقنية مختلفة لتعريف مودل البيانات (Data Model) واحدة من هذه الطرق تسمى (Code First) وهذه الطريقة تعني انشاء قاعدة البيانات بصورة تلقائية من المودل وكائناتها مثل (الجداول و العلاقات الخ) عند تنفيذ البرنامج بمعنى أوضح اننا نكتب كود في المودل (حسب شروط معينة) وأعتمادآ على هذا الكود سيتم انشاء قاعدة البيانات ووجداولها .

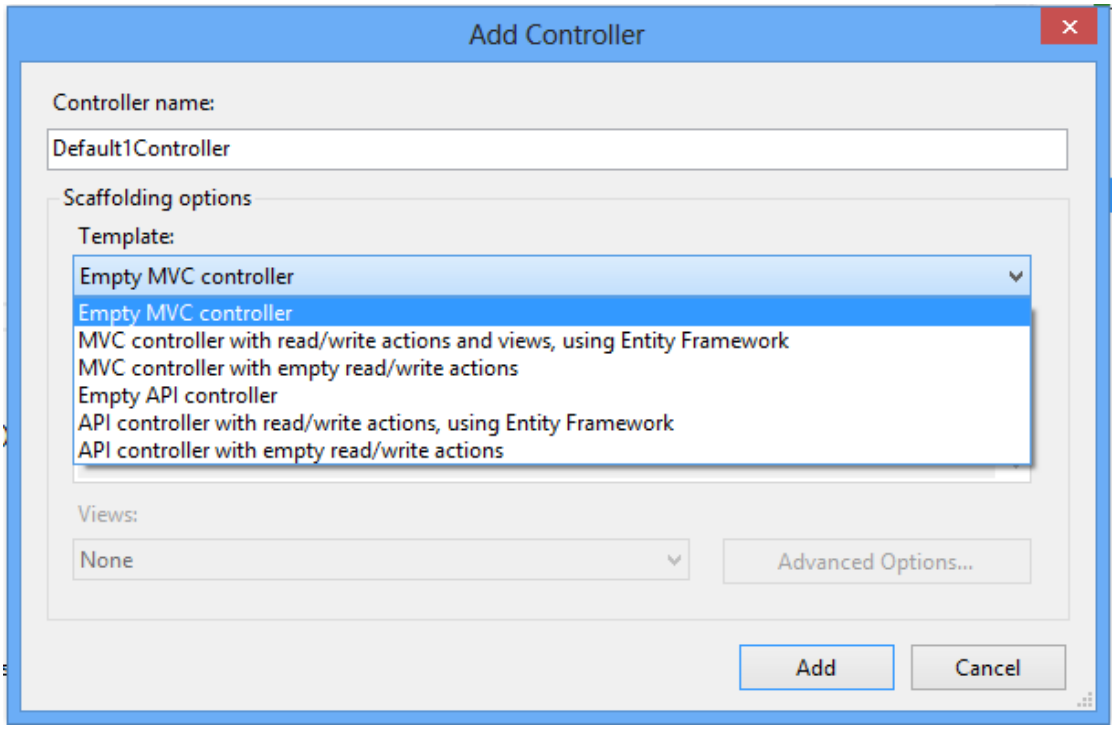
Scaffolding :

في (MVC) فأن (Scaffolding) تستطيع ان تولد لك كود معياري تحتاجه في علميات الانشاء والقراءة والتعديل والحذف (CRUD) في التطبيق .



قوالب (Scaffolding Templates) تستطيع أن تختير نوع التعاريف للمودل وأعتمادا عليه تولد لنا كونترولر مع الفيو الخاص به ، مثلا هي ستعرف ماهو أسم الكونترولر وما أسم الفيو ، وماهو الكود الضروري لعناصرهم ، وفي أي مجلد ستضعهم في التطبيق .

لا تعتقد أن (Scaffolding) تبني لك تطبيق كامل وانما هي تساعدك في بناء بعض العناصر بصورة افتراضية تستطيع التعديل عليها بما يتلائم مع تطبيقك ، هناك عدة انواع من القوالب هي :



- ☒ Empty MVC Controller : هذا النوع يضيف لنا كونترولر بأسم أنت تحدده داخل المجلد (Controllers) داخله أكشن واحد أسمه (Index) فارغ من الكود ولا ينشئ أي فيو له.
- ☒ MVC Controller with read/write action and view using EF : - هذا القالب ليس فقط يولد لك كونترولر داخله الاكشنات ((Index ,Create ,Details , Edit , Delete) وانما يولد لك معها جميع الفيو والكود الضروري لأرسال واستلام العملومات من قاعدة البيانات .
- ☒ MVC Controller with empty read /write action :- هذا القالب يضيف لنا كونترولر للشمروع بداخله عدة أكشنات هي (Index ,Create ,Details , Edit , Delete) وينشئ لكل أكشن فيو خاص به ، ولكل أكشن كود خاص به يمكن التعديل عليه .
- ☒ API Controller with empty read/write action : - هذا القالب يضيف لنا كونترولر مشتق من (API Controller) نستطيع أستخدامة في بناء تطبيقات (WEB API) (لنا كلام مفصل حول هذا الموضوع فيما بعد) .

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 (MVC Asp.net) 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) .

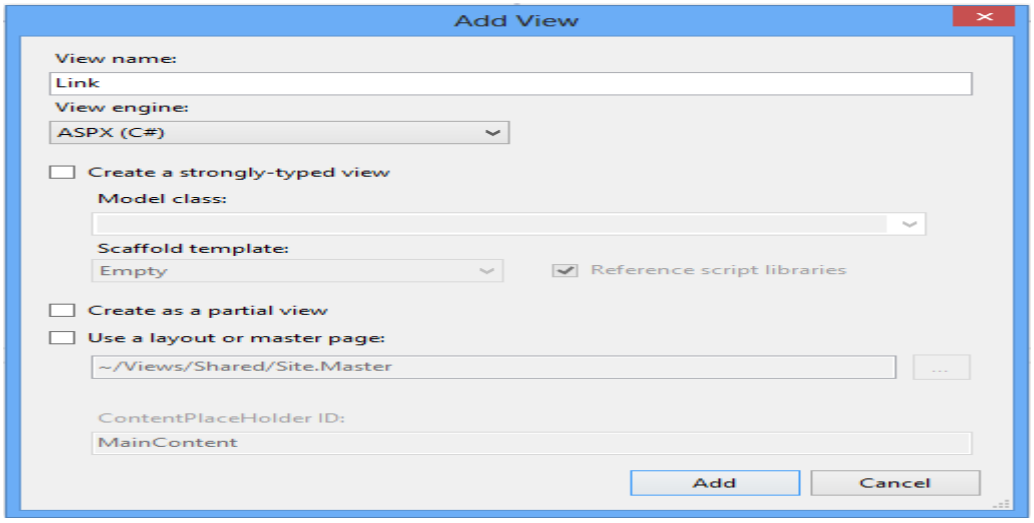
أنشاء الروابط :

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) :

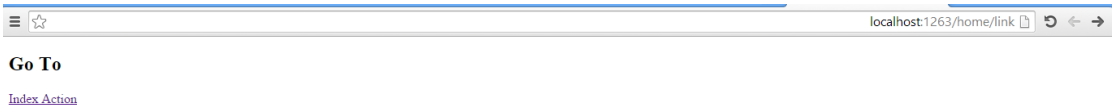
```
public ActionResult Link()
        {
            return View();
        }
```

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>
```

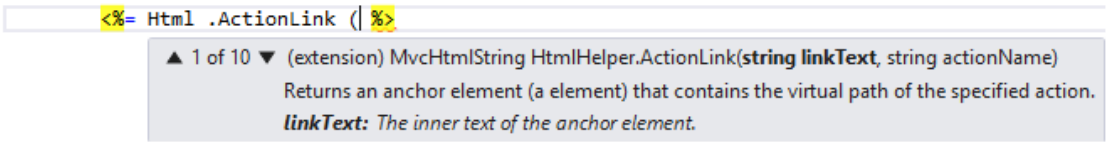
في الكود أعلاه فأن الاجراء (Html.ActionLink()) أخذ معاملين الاول عنوان نص الرابط والثاني الاكشن الذي سينتقل اليه .



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) :

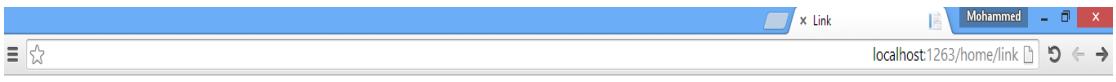
```
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<!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>
```

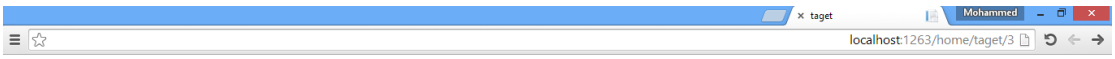
عند التنفيذ نكتب في رابط المتصفح أسم الأكشن (Link) :



Go To

Go To Target

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



3

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) .

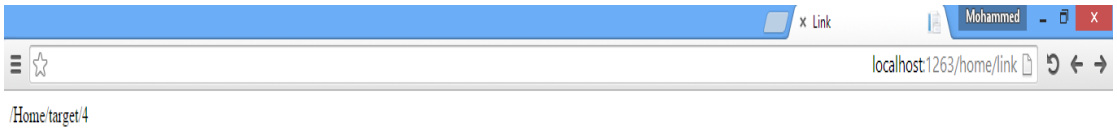
: URL Helper

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 .

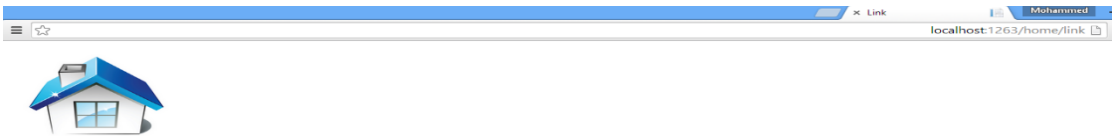
- Content : هذا يساعد في تحويل مسار التطبيق النسبي الى تام .
- Route URL : مشابه جداً للـ (Route Link) .

: 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 :

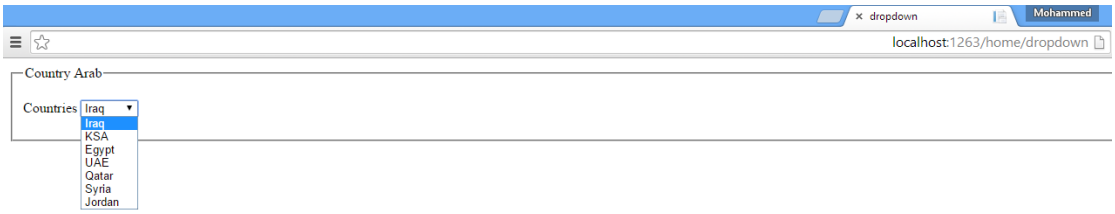
```
public ActionResult  dropdown()
        {
            List <string> li =new List <string >();
            li.Add("Iraq");
            li.Add("KSA");
            li.Add("Egypt");
            li.Add("UAE");
            li.Add("Qatar");
            li.Add("Syria");
            li.Add("Jordan ");
            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")%>
```

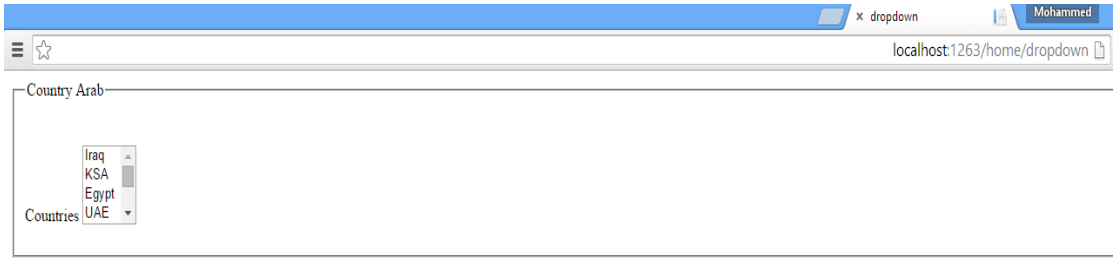
وعند التنفيذ :

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) :

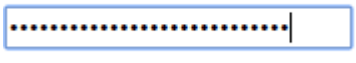
```
<%= Html.TextArea ("Comment","Leaves Comment" )%>
```

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 &lt .
- > تصبح &gt .
- " تصبح &quot .
- & تصبح &amp .

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 :

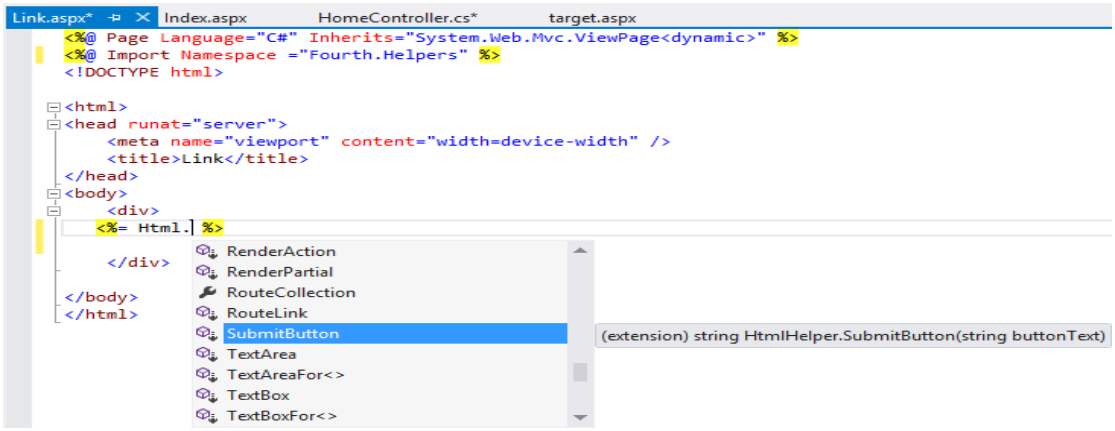
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc ;
namespace Fourth.Helpers
{
    public static  class SubmitButtonHelper
```

```
    {
        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 .