مقدمه ای بر

Microsoft

ASP.NET MVC 5

ترجمه و تالیف : مهندس مهدی کیانی

ویرایش اول
پاییز ۹۴

Introduction to Microsoft ASP.NET MVC 5

By

Mehdi, Kiani

www.mkiani.ir

تقدیم به

همه آنانی که عاشق آموختن هستند

و با سپاس فراوان از همسر بزرگوارم که همیشه مشوق من بودند.

فهرست مطالب

درباره نویسنده

مهدی کیانی متولد اصفهان می باشد.او بیش از یک دهه است که در حوزه کامپیوتر و به صورت تخصصی در زمینه برنامه نویسی پلت فرم دات نت فعالیت می کند. در این مدت دوره های آموزشی متعددی را در آموزشگاه های خصوصی،دولتی و نیمه دولتی برگزار کرده است. از یادگیری در همه امور لذت می برد و علاوه بر برنامه نویسی در زمینه های هنری نیز فعالیت می کند.در لینک زیر می توانید بیوگرافی او را مشاهده نمائید.

http://mkiani.ir/pages/biography

درباره کتاب

این کتاب دارای شش فصل بوده که در مجموع بیش از ۱۹۰ صفحه مطلب را شامل می شود.

در فصل اول در باره تکنولوژی MVC و اجزای آن صحبت خواهد شد.

در فصل دوم اولین برنامه ای که از تکنولوژی MVC استفاده می کند را ایجاد و اولین کنترلر را درون آن استفاده خواهیم کرد.

در فصل سوم نحوه استفاده از ویو ها را در برنامه های MVC خواهیم دید.

درفصل چهارم بحث روتینگ ها در MVC بررسی خواهد شد.

درفصل پنجم اکشن متد ها با جزئیات بیشتر و همچنین فیلتر ها توضیح داده خواهند شد.

و نهایتا در فصل ششم توضیحات کاملتری در رابطه با ویو ها و همچنین متد های راهنما مورد بررسی قرار خواهند گرفت.

در نوشتن مطالب این کتاب به ترجمه صرف پرداخته نشده است بلکه برداشت نویسنده از کتاب های مرجع در این زمینه همراه با سایر منابع به ویژه سایت رسمی این تکنولوژی و نیز تجربیات شخصی نویسنده ترکیب شده است و نگارش این کتاب بر این اساس شکل گرفته است.

افرادی که تجربیاتی در زمینه نوشتن مقاله یا مطالب علمی دارند نیک می دانند که چه کار بسیار دشواری است. با تمامی کمبود وقتی که داشتم سعی کردم تا حد امکان مطالب کتاب بدون ایراد در اختیار شما قرار گیرد. لذا صمیمانه تقاضامندم ایرادات مطالب را به همراه نقطه نظرات خود از طریق پست الکترونیک ذیل با من در میان گذاشته تا در نسخه های بعدی رفع گردند.

mkiani3000@gmail.com

این کتاب برای چه کسانی می باشد/نمی باشد

اگر شما از دسته برنامه نویسانی که دارای تجربه کار با یکی از زبان های برنامه نویسی دات نت نظیر سی شارپ یا ویژوال بیسیک و کاربرد آن ها در توسعه پروژه های تحت وب، دارید می توانید با این کتاب جهت شروع حرکت در مسیر استفاده از تکنولوژی MVC قدم بردارید.علاوه بر دانش در زمینه دات نت نیاز به مفاهیم پایه در خصوص دستورات html ، Css نیز خواهید داشت.لذا چنانچه شما هیچ اطلاعاتی در زمینه های مذکور ندارید این کتاب کمکی به شما نخواهد کرد. علاوه بر این همانطور که از نام کتاب مشخص است این کتاب صرفا مقدمه ای در خصوص تکنولوژی MVC به شما ارائه می دهد. هدف از این کتاب صرفا ارائه یک نقشه راه در جهت حرکت به سمت این تکنولوژی است. لذا خواننده گان گرامی می بایستی با مطالعه سایر منابع و مراجع اصلی این تکنولوژی توانمندی های خود را در این زمینه بهبود بخشند تا بتوانند از آن در پروژه های واقعی خود بهره گیرند.

مقدمه

نوشتن برنامه های تحت وب از زمان ظهور تکنولوژی های تحت وب تا به امروز تغییرات بسیاری را به همراه داشته است. ظهور ASP.Net توسط ماکروسافت در بستر دات نت یکی از بزرگترین این تغییرات بوده است.تکنولوژی که هم اکنون نیز در بسیاری از پروژه ها مورد استفاده قرار می گیرد. یکی از مفاهیمی که ماکروسافت همواره سعی کرده در تکنولوژی های برنامه نویسی تحت وب مد نظر قرار دهد کاهش وابستگی ها بین اجزای یک پروژه است. در این خصوص تغییرات زیادی در تکنولوژی ASP.Net Web Forms تا به امروز انجام شده است. شاید یکی از تغییرات مهم در زمینه توسعه برنامه های تحت وب را بتوان پیاده سازی الگوی MVC در این زمینه توسط ماکروسافت دانست. ماکروسافت با ایجاد بستر جدید بر روی دات نت اقدام به پیاده سازی این الگو کرد و آن را ASP.NET MVC نام نهاد. این کتاب آغازی است برای قدم نهادن در مسیر این تکنولوژی و بهره برداری از آن در پروژه های متعدد. امیدوارم که این کتاب راهنمای مناسبی برای شما در این زمینه باشد.

باتشکر

مهدی کیانی

پاییز ۱۳۹۴

فصل اول : MVC چیست؟ ASP.Net MVC چیست؟

مقدمه

واژه MVC سر آغاز سه کلمه Model، View و Controller می باشد. MVC یک الگوی طراحی است و برخلاف اینکه بعضا تصور می شود MVC ویژه تکنولوژی ASP.Net می باشد چنین نیست. به طور کلی الگوهای طراحی روش هایی را برای کد نویسی بهتر ارائه می دهند. الگو های طراحی به شما کمک می کنند تا بتوانید ساختار برنامه خود را به شکلی طرح ریزی نمائید که بتواند مزایایی از جمله موارد ذیل را برای شما فراهم آورد:

۱- **نگهداری آسان تر کد:** منظور از نگهداری در اینجا مدیریت تغییرات در آینده می باشد. هر چه کد شما بهتر و ساخت یافته تر نوشته شده باشد طبیعتا فهم و درک و اعمال تغییرات در آن به مراتب راحت تر خواهد بود.

۲- **توسعه پذیری بهتر:** اگر ساختار کد نویسی شما بر اساس الگوی صحیحی طرح ریزی شده باشد توسعه آن راحت تر و سریعتر انجام می پذیرد.

۳- **اشکا زدایی بهتر:** هرچقدر ساختار کد نویسی یک پروژه بهتر باشد طبیعتا اشکال زدایی برنامه نیز به مراتب راحت تر و سریعتر خواهد بود.

۴- **قابلیت تست پذیری بهتر:** یکی از فعالیت هایی که برای هر پروژه لازم است انجام پذیرد اما در بسیاری از پروژه ها بنا به علل مختلف انجام نمی شود و یا کمتر به آن توجه می شود تست کردن برنامه و کد ها ی نوشته شده می باشد.کد نویسی بهتر باعث می شود که عملیات تست کردن برنامه بهتر و دقیق تر صورت پذیرد.

الگوی MVC بدین منظور طراحی شد تا امکانات فوق را فراهم آورد.

ساختار MVC

همانطور که گفته شد الگوی MVC دارای سه بخش Model ، View و Controller می باشد. وظیفه هر یک از این سه بخش به شرح ذیل است:

- **Model :** در الگوی MVC به مجموعه کلاس های دات نت که معرف مدل های داده ای می باشند Model می گویند.
- **View :** بخشی از الگوی MVC که وظیفه نمایش اطلاعات به کاربر را دارد View می گویند.
- **Controller :** مهمترین بخش الگوی MVC بخش Controller های آن است. وظیفه این بخش این است که درخواست های کاربر را دریافت کرده سپس برای انجام درخواست ، مدل(های) مورد نظر را فراخوانی وپس از حصول نتیجه یک View را انتخاب و مدل را به View ارسال می کند تا View داده های دریافتی را به کاربر نشان دهد.

تفاوت View Model و Domain Model :

مدل ها در الگوی MVC را می توان به دو دسته کلی View Model و Domain Model دسته بندی کرد. اگر مدل داده ای صرفا جهت نگهداری و نمایش داده ها بکار رود به آن View Model و اگر علاوه بر نگهداری داده ها عملیات منطقی مختلفی نیز در آن صورت بگیرد به آن Domain Model می گویند. می توانید View Model ها را همان DTO ها در نظر بگیرید که برای جابجایی داده ها بین اجزای مختلف یک پروژه بکار می روند.

در یک برنامه ممکن است از هر دو نوع مدل به صورت همزمان استفاده شود. بدین صورت که هنگام پردازش درخواست ها از یک کلاس به عنوان Domain Model و پس از پردازش و برای نمایش از یک کلاس به عنوان View Model استفاده گردد. بدیهی است در این حالت نیاز به مکانیزمی برای نگاشت Domain Model ها به View Model های متناظر خود و بلعکس می باشد.

ASP.Net MVC چیست؟

شرکت ماکروسافت در تکنولوژی ASP.Net مبحثی را مطرح کرد به نام ASP.Net MVC ودر آن به پیاده سازی دقیق الگوی MVC در تکنولوژی ASP.Net پرداخت. در واقع ASP.Net MVC مجموعه ای از کلاس هایی هستند که از بستر دات نت فریم ورک بهره برده و به پیاده سازی این الگو می پردازد. در

ASP.NET MVC حاوی کلاسهایی مشتق شده از کلاس Controller به عنوان کنترلر ها، کلاس هایی به عنوان مدل و نهایتا فایل هایی با پسوند aspx یا cshtml و یا vbhtml به عنوان ویو می باشد. فایل هایی با پسوند aspx یقینا برای شما آشنا می باشند. دو فایل بعدی یعنی cshtml و vbhtml ها مربوط به موتور ویویی به نام Razor می باشند که در ASP.NET MVC مورد استفاده قرار می گیرند که در ادامه مطالب با جزئیات آن ها آشنا خواهید شد.

ASP.Net MVC و ASP.Net Web Form

یکی از سوالات و ابهاماتی که معمولا با به وجود آمدن یک تکنولوژی جدید در ذهن برنامه نویسان شکل می گیرد این است که آیا تکنولوژی جدید جایگزین تکنولوژی های قبلی است و هر آنچه که قبلا بوده باید به دست فراموشی سپرده شود؟ متاسفانه پاسخ سطحی" بلی" به این سوال در بین برخی از برنامه نویسان بسیار زیاد است. واقعیت این است که برای پاسخ به این سوال می بایستی شرایط بسیاری را مد نظر قرار گرفت و بر اساس آن ها به این سوال پاسخ داد. شرایط و سوالاتی از جمله :

- ما در حال توسعه چه برنامه ای هستیم؟
- برنامه ما در چه سطح از لحاظ وسعت قرار دارد؟
- زمان ما برای نوشتن یک برنامه چقدر است؟
- چه میزان الزام در به کار بردن تکنولوژی خاصی داریم؟
- شرایط نرم افزاری و سخت افزاری محیط نهایی کار با نرم افزار تولید شده چیست؟
- چه جنبه هایی از برنامه نویسی برای برنامه مورد نظر ما بیشتر اهمیت دارد؟
- دیدگاه مالک و توسعه دهنده آن تکنولوژی در رابطه با شرایط مساعد برای استفاده از آن تکنولوژی چیست؟
- تکنولوژی های قبلی به موازات تکنولوژی جدید به روز رسانی خواهند شد یا خیر؟(در لحظه نوشتن این کتاب هنوز اطلاعیه ای از سوی ماکروسافت مبنی بر عدم پشتیبانی از Web Form مطرح نشده است)

همانطور که می دانید هر برنامه دارای شرایط خاصی است و با توجه به آن شرایط باید تصمیم گیری شود که از چه بستری برای توسعه برنامه استفاده گردد. به عنوان مثال اگر شما نیاز به یک فرم بسیار ساده دارید و فقط چند ساعت فرصت دارید تا آن را به مرحله اجرا برسانید آیا صلا ح است از برنامه نویسی لایه ای یا DDD و یا هر الگوی دیگری که اثربخشی آن در برنامه های متوسط به بالا است استفاده گردد یا اینکه ساده ترین و سریعترین راه مقرون به صرفه خواهد بود؟ یقینا چون در این مثال فاکتور زمان بسیار مهم است و همچنین برنامه نهایی یک

فرم ساده می باشد بنابر این به هیچ وجه به صلاح نیست که برای این کار به سمت الگوی برنامه نویسی لایه ای و پیاده سازی چند لایه مختلف برای ایجاد یک فرم ساده بروید.

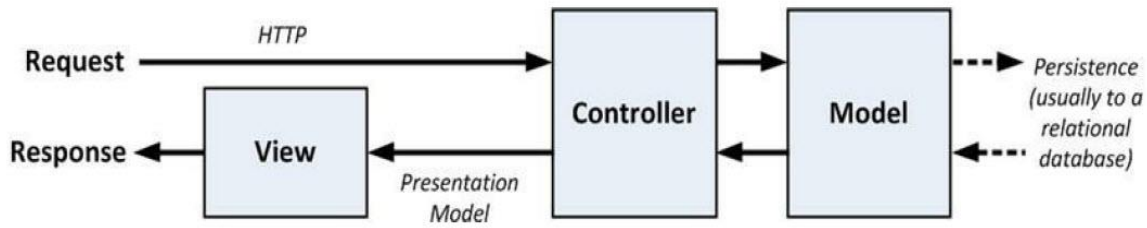
پس همانطور که بیان شده به راحتی نمی توان یک تکنولوژی را جایگزین قبلی ها کرد. بنابر این بهتر است به جای اینکه بگوییم آیا ASP.Net MVC بهتر است یا Web Form بگوییم مزایا و معایب هر کدام نسبت به یکدیگر کدام است و اصولا چه زمانی از کدام یک باید استفاده کرد؟

صرفا جهت اطلاع!

زمانی که در سال ۲۰۰۶ تکنولوژی WPF همراه با دات نت فریم ورک ۳,۰ رو نمایی شد جوی عجیب مبنی بر این که Windows Forms دیگر مرده است. همه سوئیچ کنند به WPF و از این دست جملات شکل گرفته بود که گذر زمان ثابت کرد که نباید در جایگزین کردن تکنولوژی ها عجله کرد.(جالب اینجا ست که خود ماکروسافت زمانی که WPF را معرفی کرد هیچگاه از کلمه جایگزین برای Windows Forms استفاده نکرد) متاسفانه این موضوع در بین برنامه نویسان ایرانی بیشتر ملموس است. همین اتفاق زمان ظهور MVC نیز رخ داد!

چرخه کار در ASP.Net MVC چگونه است؟

زمانی که یک درخواستی برای سرور فرستاده می شود این درخواست توسط کنترلر مربوطه دریافت خواهد شد. سپس کنترلر از بین مدل های تعریف شده در برنامه مدل مربوطه را انتخاب و درخواست را پردازش می کند. پردازش ها توسط متد هایی انجام می شود که به آن ها اکشن متد می گویند. پردازش درخواست می تواند به سادگی نمایش یک رشته متنی باشد یا می تواند یک مجموعه از اطلاعات را در پایگاه داده تغییر دهد. پس از اینکه در خواست پردازش شد یک ویو توسط کنترلر انتخاب شده و سپس داده های آماده نمایش به کاربر(در صورت وجود) توسط مدل به ویو فرستاده می شوند و نهایتا ویوی مربوطه خروجی مورد نیاز را برای کاربر مهیا می سازد. این روند در کتاب Apress Pro MVC 5 نوشته Adam Freeman به صورت زیر نشان داده شده است.



شکل ۱ - ۱

به عنوان مثال درخواستی به صورت آدرس زیر به سرور ارسال می شود:

http://www.domain.com/Account/Register

در این حالت کنترلری به نام Account درخواست فوق را دریافت و متناظر با آن متدی به نام Register که درون کنترلر Account تعریف شده است فراخوانی و پس از پردازش مذکور خروجی مورد نظر به کاربر ارسال می شود. همانطور که گفته شد متدهایی که در MVC برای پردازش درخواست های کاربر فراخوانی می شوند به اکشن متد معروف هستند. خروجی این متد ها می تواند یک داده ثابت نظیر یک رشته متنی ساده و یا از جنس ActionResult (نظیر ViewResult) باشد. نگران نباشید این جزئیات در ادامه مباحث به تفضیل توضیح داده خواهند شد.

برخی از امکانات و مزایای کلیدی ASP.Net MVC

۱- **گسترش پذیری :** از آن جا که کد های فریم ورک MVC بر بستر دات نت به صورتی کاملا انتزاعی همراه با پیاده سازی های پیش فرض نوشته شده اند شما به راحتی می توانید از پیاده سازی های اولیه ان استفاده و یا آن را گسترش دهید. به عنوان مثال

a. شما می توانید از پیاده سازی های پیش فرضی که در MVC برای هر مبحثی تعبیه شده است استفاده نمائید. مثلا سیستم View Engine یا Controller Factory و...
b. شما می تواند با ارث بری از کلاس های موجود در MVC پیاده سازی پیش فرض را به نحو مقتضی و مورد نیاز خود تغییر دهید. مثلا ایجاد یک View Engine سفارشی.
c. همچنین می توانید با پیاده سازی اینترفیس های موجود در این فریم ورک بخشی از فریم ورک را از ابتدا تا انتها خودتان پیاده سازی کنید. مثلا تغییر ساختار انتخاب و ایجاد کنترلر ها

نکته: توسعه و تغییر روال های پیش فرضی که در ASP.NET MVC رخ می دهد نیاز به تجربه و مهارت می باشد. لذا برای برنامه نویسان مبتدی در این زمینه پیشنهاد نمی گردد.

۲- **کنترل کامل بر روی کد های html تولید شده**

a. موتور نمایشی MVC به شکلی است که این امکان را به شما می دهد تاکنترل کاملی بر روی کد های html تولید شده داشته باشید. این یکی از مزایای مهم MVC در قیاس با کامپوننت های Web Form می باشد. در ASP.Net Web Form کنترل های این تکنولوژی دارای متدی به نام رندر می باشند که مسئولیت تولید کد های html را به عهده دارند. اگر چه امکان شخصی سازی کنترل های ASP.NET Web Forms ها نیز وجود دارد تا بتوان خروجی مورد نیاز را تولید کرد اما این ار مستلزم مهارت و گاها کد نویسی زیادی می باشد. در

MVC شما می بایستی در ویو ها کد های html را بر اساس مدل داده ای و نیز عملکرد ویو تولید نمائید. یقینا در این حالت نیاز است تا برنامه نویس بسیاری از کد های html ای که کنترل های Web Form مسئول تولید آن ها بودند را به صورت دستی بنویسند. اینکه کدام روش بهتر و ساده تر است بستگی به برنامه نویس نیز دارد. بسیاری از برنامه نویسان مشتاق هستند تا کنترل همه چیز را خودشان در درست بگیرند. برخی نیز ترجیح می دهند بر اساس امکانات موجود اقدام به توسعه برنامه های خود نمایند. اگر شما نیز مانند من دوست دارید تا ۰ تا ۱۰۰ کد های ایجاد شده تحت کنترل خودتان باشد MVC برای شما خواهد بود. البته موتور های تولید واسط کاربری MVC امکاناتی را در جهت تولید سرعتر کد های html برای شما مهیا می کند.

۳- قابل تست بودن بخش های مختلف

از آنجا که اصل عدم وابستگی یکی از مهمترین خواص الگوی MVC می باشد و در ASP.Net MVC این اصل به خوبی پیاد ه سازی شده است شما به راحتی خواهید توانست تا برای بخش های مختلف برنامه خود کد های تست بنویسید.

٤- سیستم قوی آدرس دهی(نگاشت آدرس ها – URL Routing)

یکی دیگر از جنبه های قوی و دوست داشتنی ASP.Net MVC که البته در Web Form نیز می توان آن را پیاده سازی کرد سیستم آدرس دهی می باشد. برای درک بهتر این مورد دو آدرس زیر را در نظر بگیرید:

http://www.apress.com/default.aspx?book=promvc5&author=adam_freeman&year=2001

http://www.apress.com/book/promvc5/adamfreeman/2001

همانطور که مشاهده می کنید آدرس دومی خوانایی بهتری نسبت به اولی دارد. همچنین کاربران راحت تر می توانند آدرس فوق را به ذهن بسپارند و یا به صورت دستی بتوانند آن را تغییر دهند یا آن را به اشتراک بگذارند.

ASP.Net MVC به شکل هوشمندانه ای اقدام به پیاده سازی آدرس ها می کند و به شما این قابلیت را می دهد تا آدرس های شفاف وبا کاربری بالا تعریف کنید.

٥- استفاده از امکانات دات نت

همانطور که گفته شد فریم ورک ASP.Net MVC بر پایه دات نت فریم ورک طراحی شده است. این بدان معنی است همانطور که شما از سایر کد های دات نت استفاده می کنید از کد های MVC نیز به همین شکل

استفاده خواهید کرد. همچنین بروزرسانی های متعدد همراه با بروز راسانی هایی که برای دات نت فریم ورک ارائه می گردد باعث می شود تا برنامه نویسان دات نت بتوانند از آخرین امکانات دات نت فریم ورک هنگام کار با ASP.Net MVC بهره ببرند.

٦- ASP.Net MVC یک فریم ورک سورس باز می باشد

کد های نوشته شده برای ASP.Net MVC به صورت Open Source نوشته شده و شما می توانید به راحتی آن را دانلود نمائید وبا تغییرات در آن نسخه خودتان را تولید کنید.(البته این عمل نیاز به تخصص بالایی می باشد که برای کاربران مبتدی توصیه نمی گردد). جهت دانلود سورس ها به آدرس http://aspnetwebstack.codeplex.com/ مراجعه نمائید.

خلاصه

در این فصل کلیات تکنولوژی MVC مورد بررسی قرار گرفتند. درک مهمی که شما از این فصل می باید بدست آورید این است که سه جزء اصلی در یک برنامه ASP.NET MVC عبارتند از مدل ها، ویوها و کنترلر ها. مدل ها نقش مدل داده ای برنامه را بازی می کنند. ویو ها مسئولیت نمایش اطلاعات به کاربر می باشد(بر اساس مدل داده ای) و کنترلر ها که قلب MVC نیز می باشند مسئول پردازش درخواست های کاربر هستند. درخواست های کاربر توسط اکشن متد هایی که در کنترلر تعریف شده است پردازش خواهند شد.در فصل بعدی به صورت عملی یک پروژه MVC ایجاد خواهیم کرد تا روند آن چه که در این فصل ارائه شده را به صورت عملی مشاهده نمائیم.

فصل دوم: اولین برنامه mvc

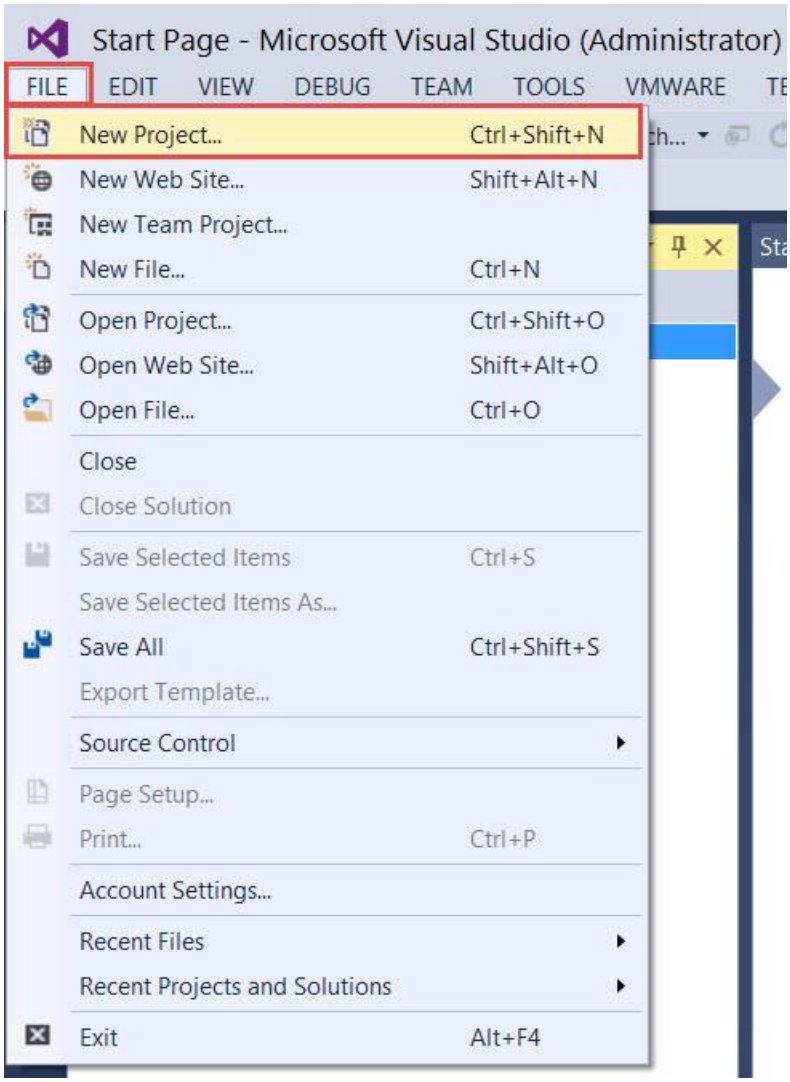
مقدمه

در این فصل اولین برنامه MVC را ایجاد خواهیم کرد. انتظار می رود در پایان این فصل بتوانید یک پروژه MVC ایجاد کنید. یک کنترلر به پروژه اضافه و روند ارسال درخواست تا حصول نتیجه توسط MVC را به خوبی درک نمائید.

در زمان نوشت این کتاب آخرین نسخه پایدار ویژوال استودیو منتشر شده نسخه ۲۰۱۳ می باشد. لذا من نیز از همین نسخه برای ایجاد پروژه ها استفاده کردم.

ایجاد پروژه

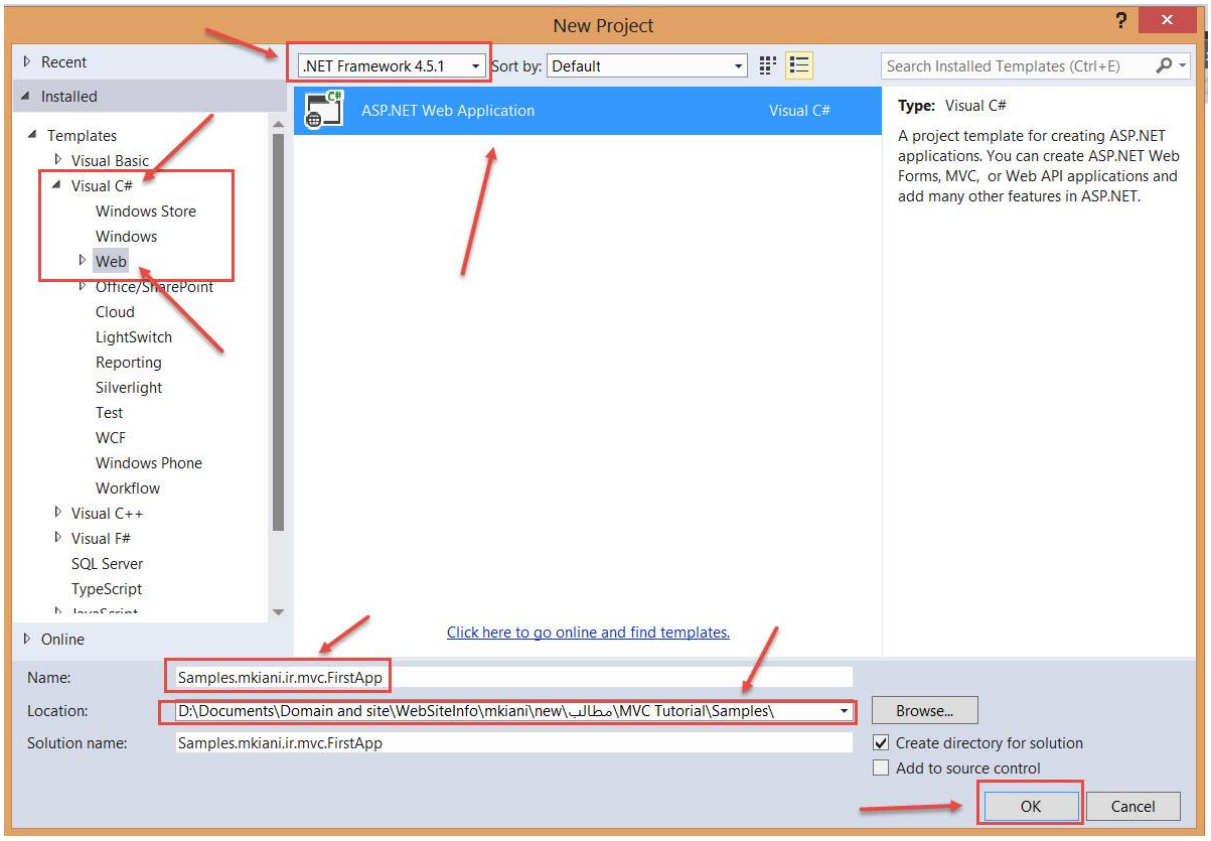
برای ایجاد پروژه نرم افزار ویژوال استودیو را باز کنید. از منوی فایل گزینه New Project را کلکید نمائید.



شکل ۲ - ۱

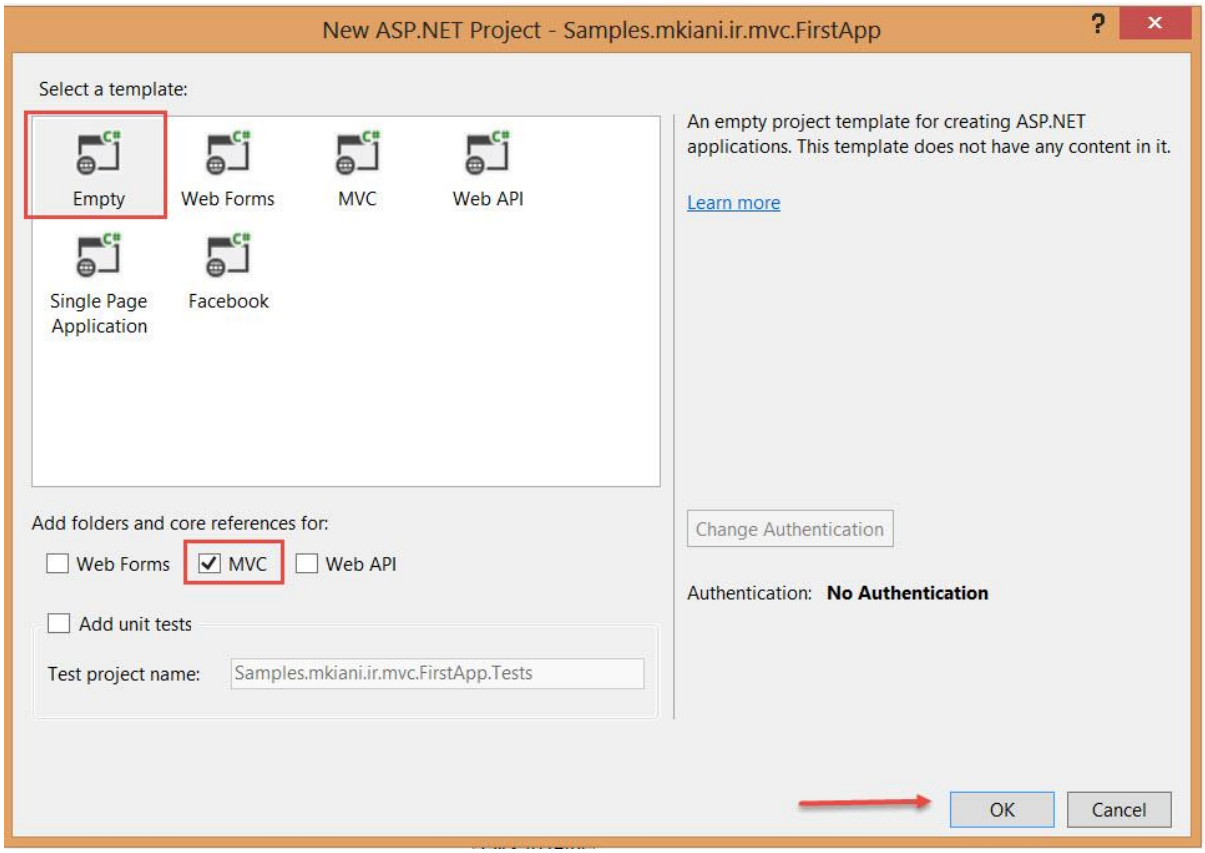
در پنجره New Project در کادر سمت چپ زبان برنامه نویسی C# و در زیر مجموعه آن گزینه Web را انتخاب نمایید. سپس از کادر وسط گزینه ASP.NET Web Application را انتخاب کنید. دقت کنید که نسخه دات نت فریم ورک انتخاب شما .Net Framework 4.5.1 باشد.[1] در کادر Name نامی برای پروژه تخصیص دهید و در قسمت Location محل ذخیره سازی پروژه را مشخص نمائید. این کار را می توانید توسط دکمه Brows انجام دهید یا به صورت دستی تایپ کنید.

[1] اگر احیانا در زمان مطالعه کتاب ویژوال استودیو ۲۰۱۵ منتشر شده است و شما از آن نسخه استفاده می کنید می توانید از دات نت فریم ورک ۶ برای پروژه های خود استفاده نمائید.



شکل ۲ - ۲

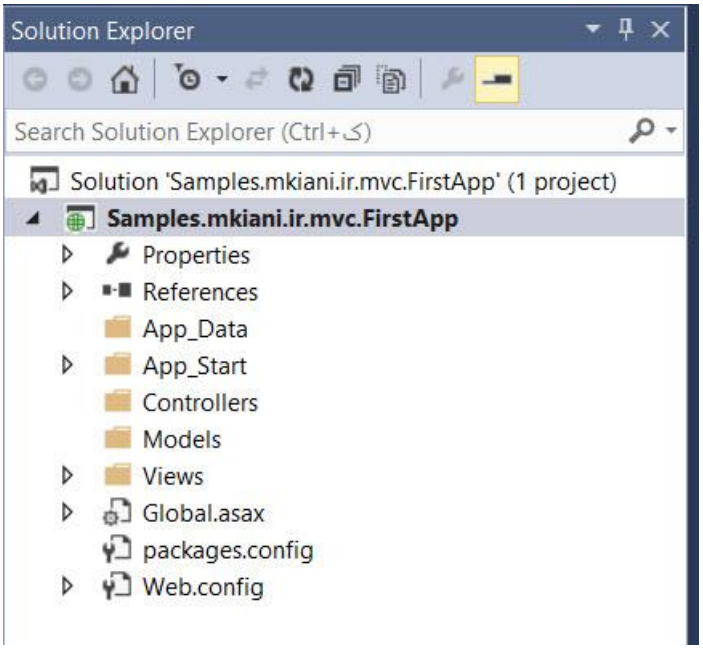
پس انجام تنظیمات فوق روی دکمه Ok کلیک کنید تا پنجره ASP.Net New Project باز شود.



شکل ۲ - ۳

در این پنجره از قسمت Select a template گزینه empty را انتخاب کنید. دقت کنید که گزینه MVC درقسمت Add folders and core references for: تیک خورده باشد. سپس بر روی دکمه Ok کلیک کنید تا پروژه ایجاد شود.

پس از ایجاد پروژه پنجره Solution Explorer را اگر باز نیست از منوی View در ویژوال استودیو باز کنید.



شکل ۲ - ۴

پنجره Solution Explorer حاوی پوشه ها و ارجاعات مورد نیاز می باشد که در ادامه با آن ها آشنا خواهید شد.

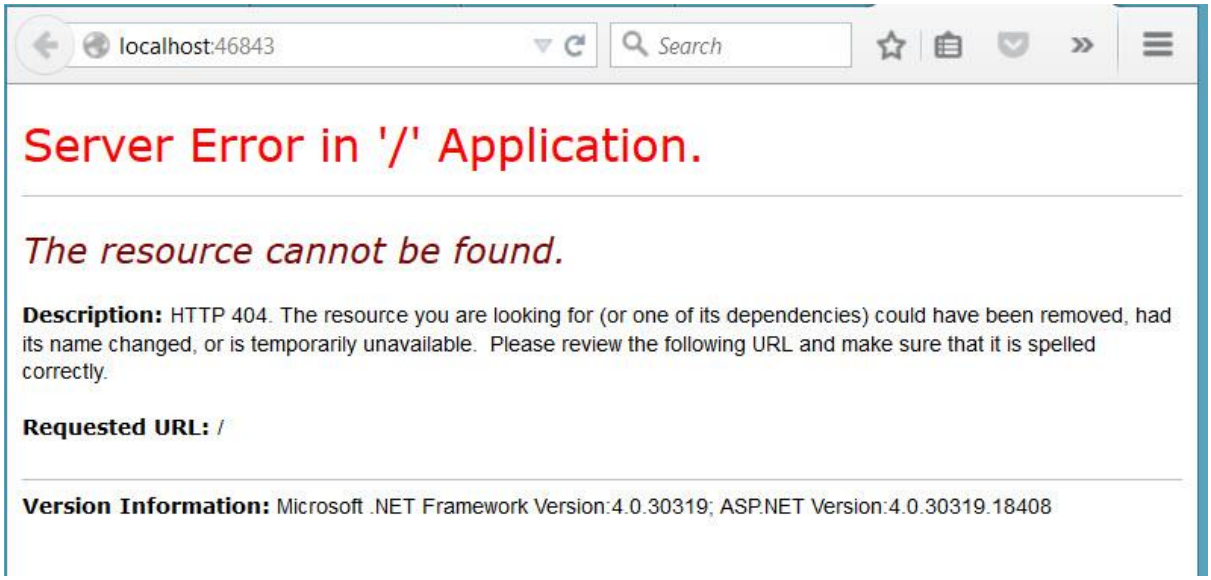
پوشه Controllers : این پوشه حاوی کنترلر های پروژه شما خواهند بود.

پوشه Models : نگهداری کننده کلاس های مربوط به مدل های داده ای خواهند بود.

پوشه Views : فایل های مربوط به بخش نمایش را در خود دارند.

نگران نباشید با این پوشه ها و محتویات درون آن ها در ادامه آشنا خواهید شد.

کلید F5 را برای اجرا شدن برنامه از صفحه کلید فشار دهید و یا از منوی Debug گزینه Start Debugging را انتخاب کنید تا پروژه اجرا شود. مرور گر شما باز شده و برنامه اجرا می شود.



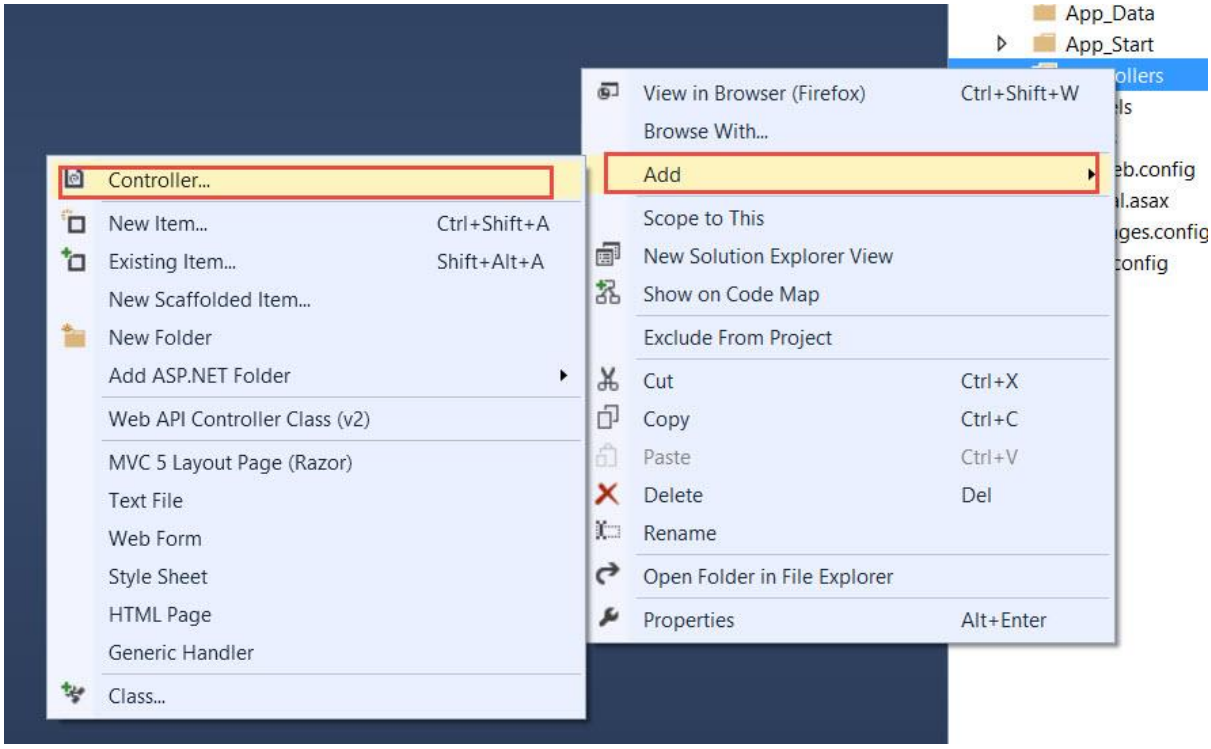
شکل ۲ - ۵

همانطور که در شکل فوق مشاهده می کنید برنامه با خطای The Resource Cannot be found مواجه شده است. این بدان علت است که هیچ کنترلری در برنامه تعریف نشده است که بتواند درخواست کاربر را پاسخ دهد. بنابر این باید یک کنترلر به برنامه اضافه کنیم.

تذکر: از این پس منظور از کلمه MVC همان ASP.Net MVC می باشد که به صورت خلاصه بیان خواهیم کرد.

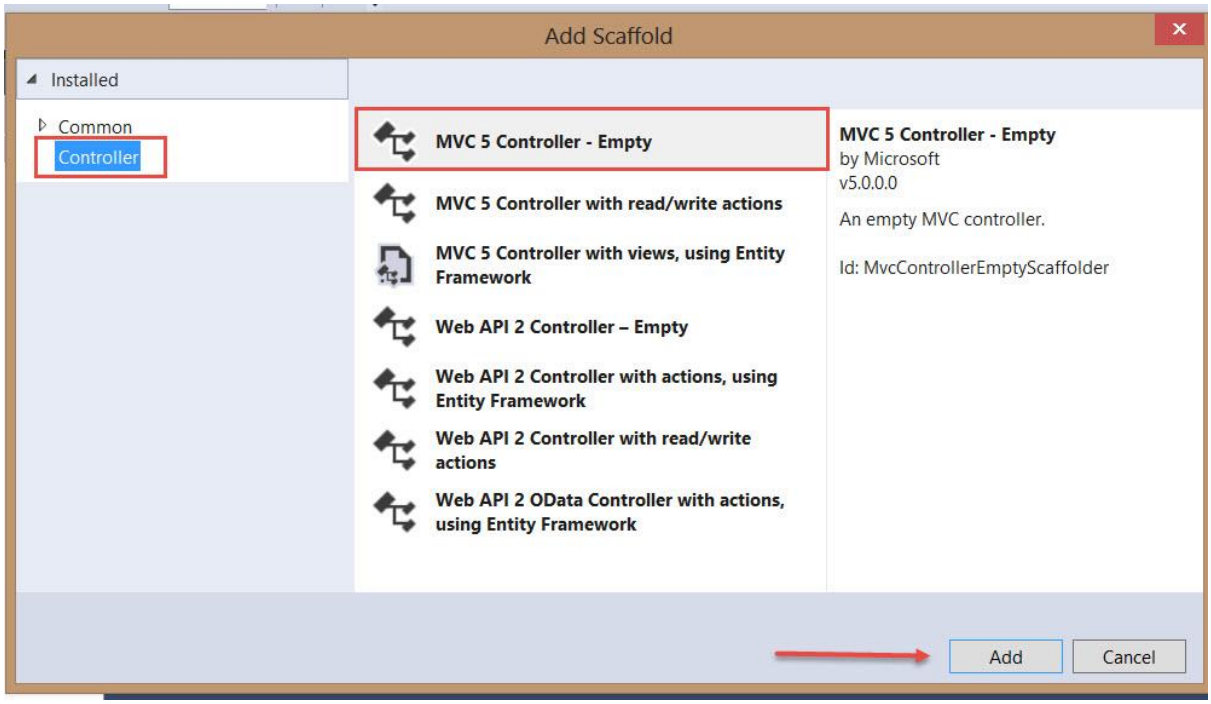
ایجاد کنترلر Home

بر روی پوشه Controllers در پنجره Solution Explorer کلیک راست کنید و از گزینه Add گزینه Controller را کلیک کنید:



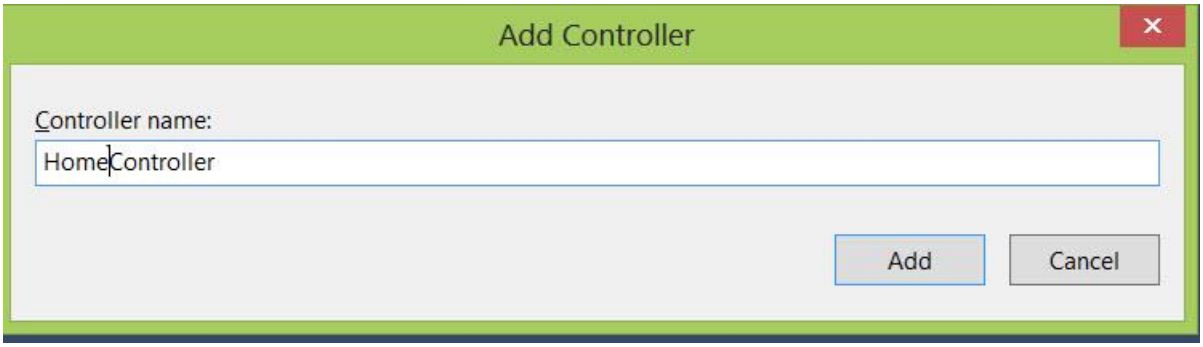
شکل ۲ – ۶

در پنجره Add Scaffold از سمت چپ گزینه Controller و از کادر وسط گزینه MVC5 Controller – Empty را انتخاب و دکمه Add را کلیک کنید.



شکل ۲ – ۷

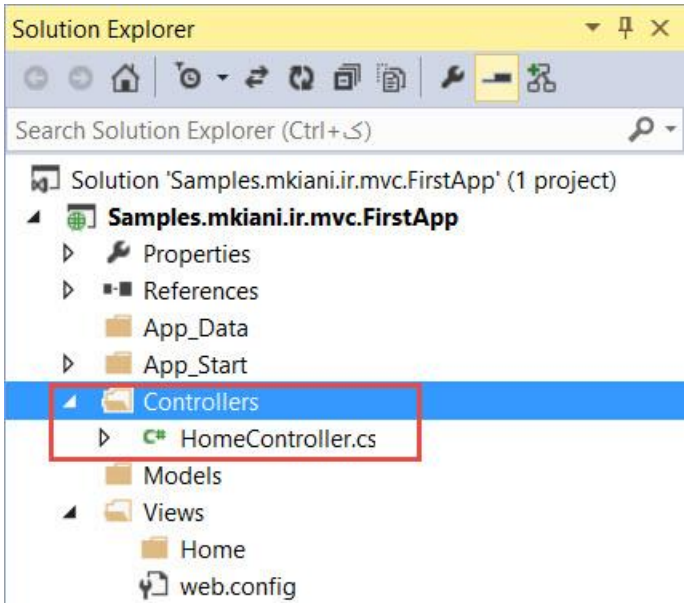
سپس در پنجره Add Controller می بایستی نام کنترلر را وارد نمائید. عنوان HomeController را در این پنجره تایپ و دکمه Add را کلیک کنید تا کنترلر Home ایجاد شود.



شکل ۲ - ۸

تذکر: پایان نام کنترلرها می بایستی کلمه Controller باشد.

اگر مراحل را به درستی انجام داده باشید می بایستی یک کنترلر به نام HomeController در پوشه Controllers در Solution Explorer ایجاد شده باشد.



شکل ۲ - ۹

بر روی HomeController.cs دوبار کلیک کنید تا باز شود. همانطور که مشاهده می کنید Controller ها در ASP.Net کلاس هایی هستند که از کلاس Controller در فضای نام System.Web.Mvc مشتق می شوند. دستورات پیش فرض این این کلاس به صورت زیر می باشد:

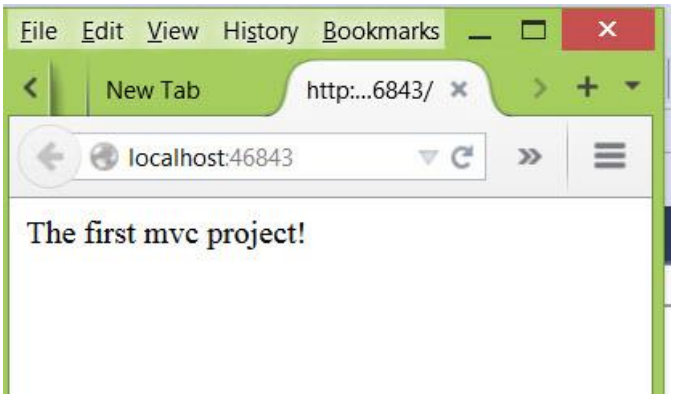
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
using System.Web.Mvc;

namespace Samples.mkiani.ir.mvc.FirstApp.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
     }
}
```

متد Index را به صورت زیر تغییر دهید:

```
public String Index()
        {
            return "The first mvc project!";
        }
```

سپس مجددا برنامه را اجرا کنید. این بار مشاهده می کنید که خطایی در برنامه رخ نداده و خروجی آن به صورت زیر خواهد بود.



شکل ۲ - ۱۰

نکته: به متد Index یک Action Method می گویند.

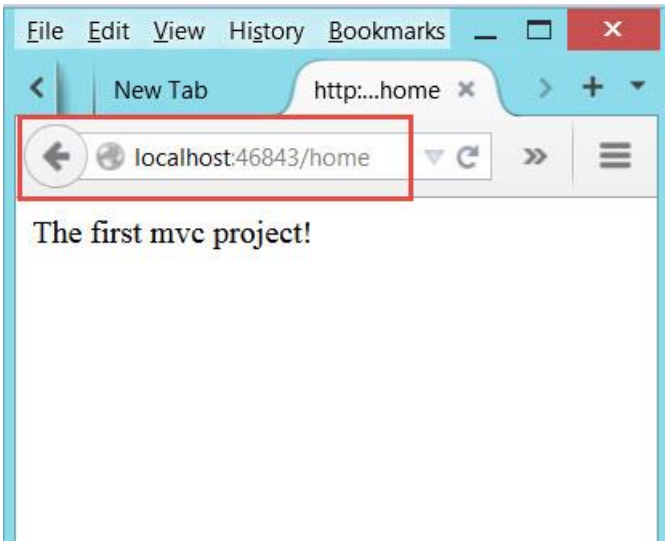
چه اتفاقی افتاد؟ چرا زمانی که یک Controller اضافه کردیم خطایی که در اجرای ابتدایی بود از بین رفت؟ آیا باید نام کنترلر ما حتما Home باشد؟ MVC از کجا متوجه می شود که دستورات متد Index را برگرداند؟

اگر نام متد Index را تغییر دهیم چه اتفاقی می افتد؟ اینها سوالاتی است که ممکن است الان در ذهن شما شکل گرفته باشد.نگران نباشید به تمامی این سوالات در ادامه پاسخ خواهم داد.

قبل از پاسخ به سوالات فوق برنامه را مجددا اجرا کنید و آدرس زیر را در بخش url تایپ نمائید:

http://localhost:46843/home

خروجی همانطور که مشاهده می کنید به صورت زیر است که با اجرای قبل تفاوتی ندارد:



شکل ۲ - ۱۱

این عمل را برای آدرس های زیر تکرار کنید:

http://localhost:46843/home/index

http://localhost:46843/home/index/test

http://localhost:46843/home/index/test2

به جای test یا test2 یک کلمه دلخواه یا یک عدد تایپ کنید و کلید enter را بزنید.

همانطور که می بینید نتیجه اجرا برای تمامی آدرس های فوق یکسان است.

چه اتفاقی رخ داد؟ چرا واکنش mvc به آدرس های متفاوتی که در فوق به برخی از آن ها اشاره شد یکسان واکنس نشان می دهد؟

جواب این سوال مربوط به سیستم روتینگ در MVC می باشد که در جای خود به صورت کامل دراین رابطه صحبت خواهم کرد. اما چیزی که در اینجا باید بدانید این است که روتینگ ها از اجزای اساسی برنامه های ASP.Net MVC می باشند.

در پوشه App_Start فایلی به نام RouteConfig.cs وجود دارد. برروی آن دوبار کلیک کنید تا باز شود.

محتویات این فایل شبیه به دستورات زیر است:

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

namespace Samples.mkiani.ir.mvc.FirstApp
{
    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 }
            );
        }
    }
}
```

همانطور که مشاهده می کنید این کلاس دارای متد استاتیکی به نام RegisterRoutes می باشد که یک پارامتر از جنس RouteCollection دریافت می کند. کلاس RouteCollection دارای متدی است به نام MapRoute که برای نگاشت آدرس های MVC به کار می رود. دستور دوم متد RegisterRoutes به MVC می گوید که یک Route با نام Default و با آدرسی به صورت {controller}/{action}/{id} خواهیم داشت که به جای {controller} ، {action} و {id} هر چیزی می تواند وجود داشته باشد. همانطور که در پارامتر defaults مشخص شده است مقدار پیش فرض controller کلمه Home و مقدار پیش فرض Action کلمه Index می باشد. این بدان معنی است که اگر به جای controller و action چیزی تایپ نشود مقادیر Home و Index برای آن ها در نظر گرفته می شود. حال زمانی که آدرس localhost:46843 در مرورگر تایپ می شود MVC در نگاشت های خود در می یابد که چون هیچ کنترلری و هیچ اکشن متدی در

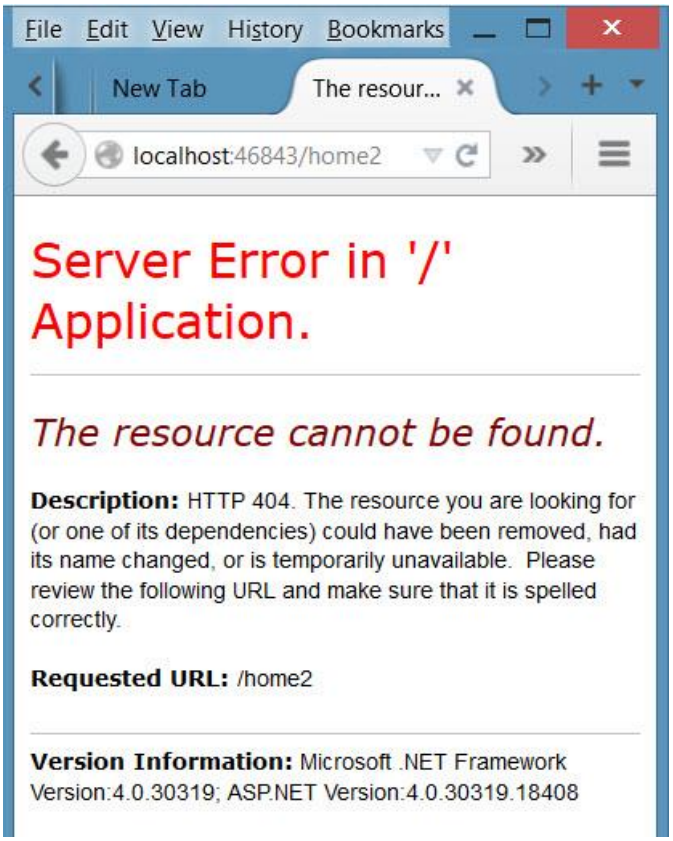
آدرس درخواستی توسط کاربرمشخص نشده است پس مقادیر پیش فرض این عبارات که به ترتیب Home و Index می باشند را مورد استفاده قرار می دهد. بنابر این وقتی آدرس localhost:46843 در مرورگر تایپ می شود MVC به سراغ متد Index در کنترلر Home خواهد رفت و بنابر این دستورات درون متد Index اجرا خواهد شد.

نکته : MVC برای یافتن کلاس کنترلر مورد نظر به صورت خودکار کلمه Controller را در انتهای نام کنترلر (در اینجا یعنی Home) اضافه خواهد کرد بنابر این کلاس HomeController را خواهد یافت.

حال برنامه را اجرا کنید و آدرس زیر را در مرورگ تایپ کنید:

http://localhost:46843/home2

خروجی شبیه شکل زیر خواهد بود.



شکل ۲ - ۱۲

در این حالت mvc به دنبال کنترلری به نام home2 خواهد گشت و چون چنین چیزی در پروژه تعریف نشده است بنابر با خطای فوق مواجه خواهید شد.

خلاصه

در این فصل توانستید اولین برنامه MVC را ایجاد و اجرا نمائید.اگر چه این برنامه بسیار مقدماتی بود اما شروع خوبی است برای قدم گذاشتن در مراحل بعدی. علاوه بر این آموختید زمانی که یک درخواست به صورت آدرس url به برنامه فرستاده می شود، MVC سعی می کند از آدرس درخواستی کاربر کنترلر و اکشن متد مورد نظر را تشخیص دهد. سپس با اجرای اکشن متد مورد نظر اقدام به پردازش درخواست کاربر می کند.خروجی اکشن متد همانطور که در فصل اول بیان شد می تواند یک داده ثابت نظیر یک رشته متنی یا یک نمونه از کلاسی که از ActionResult مشتق شده است باشد(نظیر ViewResult که در فصل بعدی با آن آشنا خواهید شد). در فصل سوم علاوه بر کنترلر، اقدام به افزودن ویو به پروژه خواهیم کرد و رفتار MVC را با ویو ها خواهیم دید.

فصل سوم:افزودن View به پروژه

مقدمه

در بخش قبلی اولین پروژه MVC را ایجاد کردیم. خروجی آن پروژه یک رشته متنی بود. در این بخش یک ویو را به پروژه اضافه خواهیم کرد تا بتوانیم خروجی html در نتیجه پردازش به کاربر نشان دهیم.پس بنابر این برای نمایش مجموعه ای از دستورات html نیاز به ویو خواهیم داشت.

در نسخه اولیه MVC از Web Form ها به عنوان موتور نمایش MVC استفاده شد. در نسخه های بعدی موتوری ویژه MVC به نام Razor به ساختار MVC اضافه شد و به عنوان موتور اصلی این تکنولوژی به کار رفت. از آنجا که در نسخه های بعدی نیز از این موتور استفاده خواهد شد ما نیز از همین روش استفاده خواهیم کرد.

همانطور که می دانید Web Form ها فایل های هستند با پسوند aspx که در برنامه نویسی ASP.Net Web Form نیز مورد استفاده قرار می گیرند. فایل های مربوط به ویو در موتور Razor برای زبان سی شارپ فایل هایی با پسوند cshtml و برای زبان ویژوال بیسیک vbhtml می باشند.

ایجاد پروژه

برای شروع یک پروژه به نام ContactForm ایجاد کنید. سپس یک کنترلر به نام Home به لیست پروژه های خود اضافه کنید. حال در پوشه Controllers بر روی فایل HomeController.cs دوبار کلیک کنید.

دستورات این فایل به صورت زیر می باشد.

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

namespace mkianiir.mvc.ContactForm.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
```

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

همانطور که مشاهده می کنید خروجی متد Index از جنس ActionResult می باشد. کلاس ActionResult یک کلاس Abstract می باشد که برای خروجی های اکشن متد ها به کار می رود. البته می توانید به جای کلاس ActionResult از کلاس ViewResult که یک کلاس مشتق شده از آن(البته با یک واسطه)می باشد نیز استفاده کنید.

یاداوری : از مباحث برنامه نویسی شی گرایی و در مبحث ارث بری می دانید که هر جا نیاز به نمونه ای از یک کلاس بود می توان از کلاس های مشتق شده آن نیز استفاده کرد.

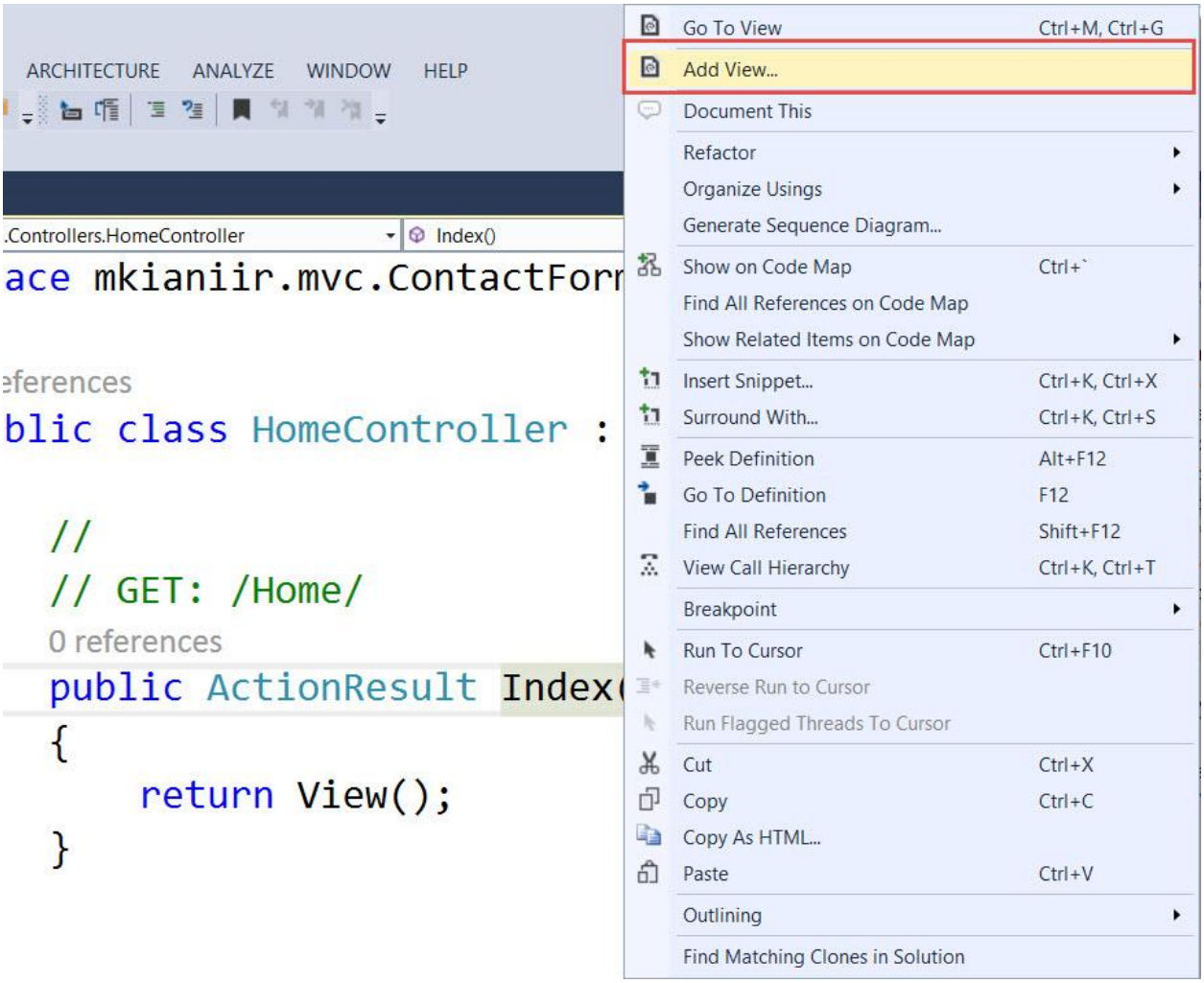
بنابر این می توانیم به جای ActionResult از ViewResult استفاده نمائیم.

```
public ViewResult Index()
        {
            return View();
        }
```

دستور return View(); به این معناست که خروجی این متد یک ویو می باشد. اما این ویو کجاست؟ MVC در کجای پروژه باید به دنبال آن بگردد؟ همانطور که در فصل دوم گفته شد ویو ها در پوشه Views در ساختار پروژه قرار می گیرند. برای هر کنترلر نیاز به یک پوشه هم نام با آن کنترلر می باشد و برای هر اکشن متد نیاز به یک فایل ویو (فایلی با پسوند cshtml) هم نام با اکشن متد مورد نظر وجود دارد. خوشبختانه ویژوال استودیو مکانیزمی برای ایجاد ویو های متناظر با اکشن متد ها دارد.

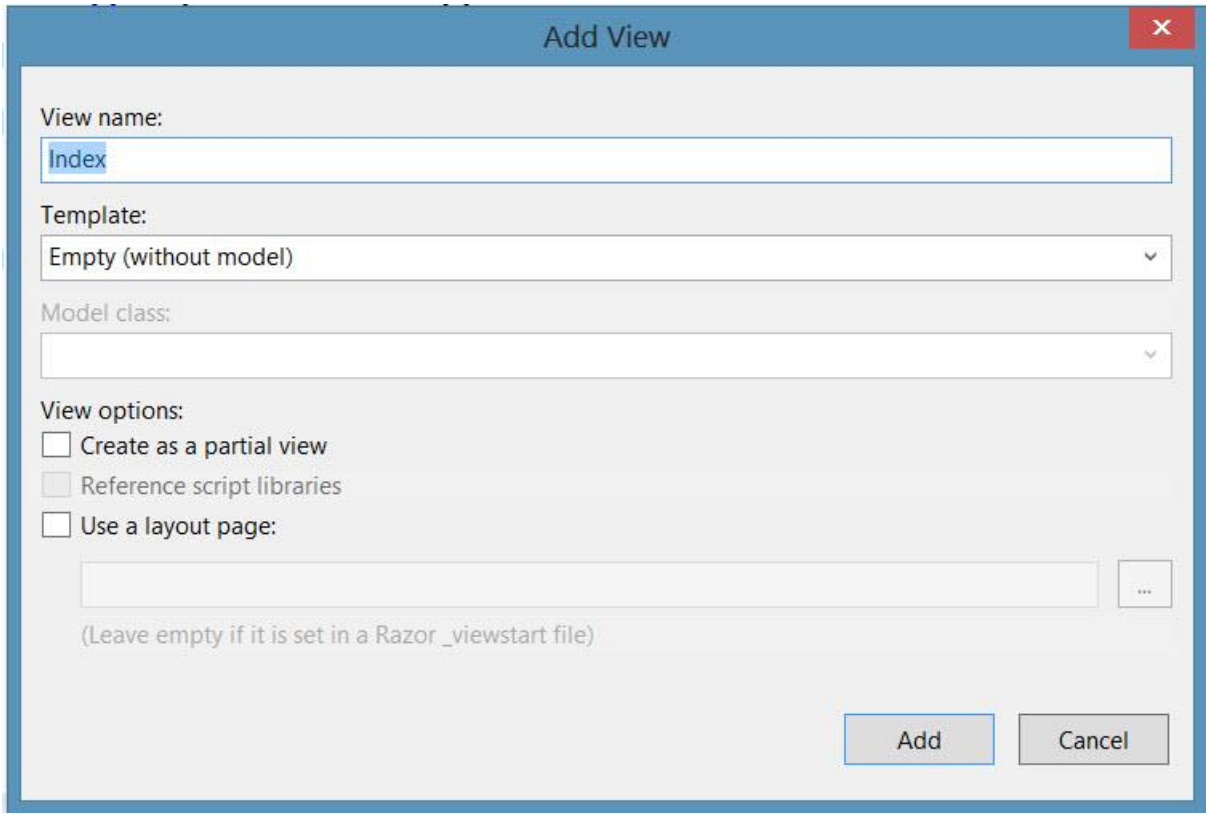
افزودن ویو به پروژه

برای ایجاد یک ویو برای متد Index بر روی نام متد کلیک راست کنید و از منوی باز شده گزینه Add View را کلیک کنید.



شکل ۳ - ۱

پنجره Add View باز خواهد شد.



شکل ۳ - ۲

پنجره Add View را به صورت زیر تکمیل نمائید:

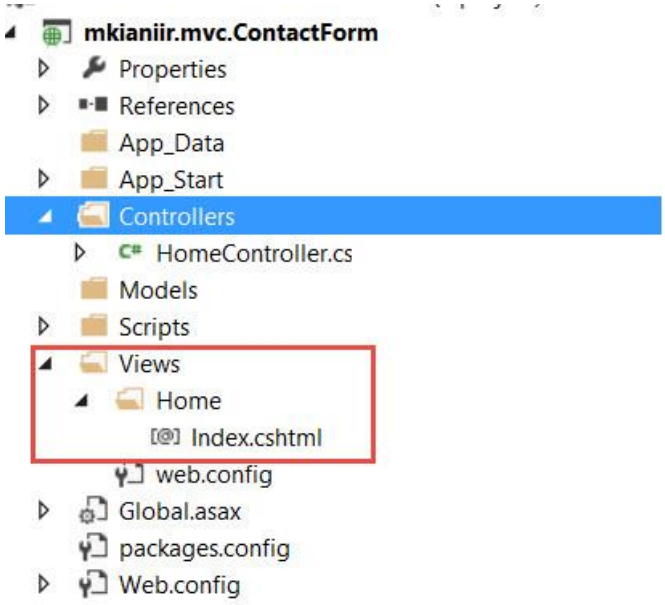
در قسمت View Name کلمه Index را تایپ نمائید(این نام ویوی متناظر با متد Index خواهد بود)

در قسمت Template گزینه Empty(without model) انتخاب شده باشد.

هیچ یک از گزینه های View options فعال نباشند.

سپس روی دکمه Add کلیک کنید.

اگر مراحل را درست انجام داده باشید در پنجره Solution Explorer و در پوشه Views یک پوشه به نام Home ایجاد شده و در این پوشه یک فایل با نام Index ایجاد شده است.



شکل ۳ - ۳

بر روی فایل Index.cshtml دوبار کلیک کنید. دستورات پیش فرض این فایل به صورت زیر است:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

خط اول این فایل با کاراکتر @ شروع می شود. زمانی که بخواهیم از دستورات زبان C# در ویو ها استفاده کنیم باید از این کاراکتر بهره ببریم.

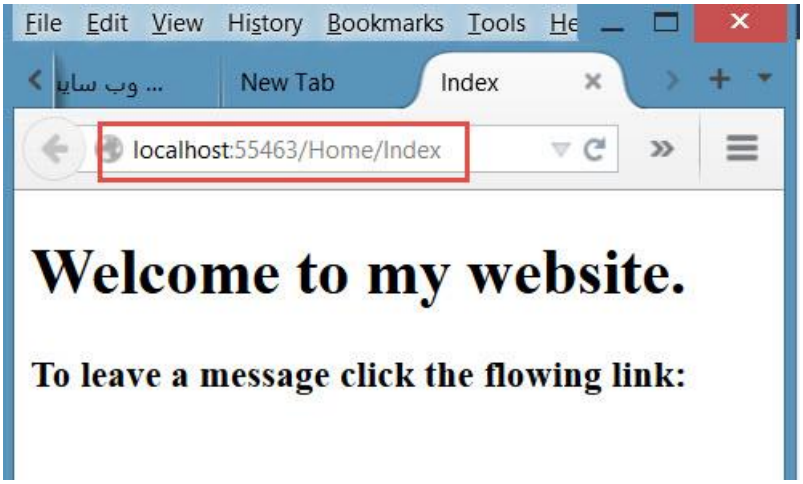
دستورا Layout=null بیانگر این است که این ویو از هیچ ویوی دیگری به نام Layout استفاده نمی کند. در مورد Layout ها در بخش های بعدی صحبت خواهیم کرد. در اینجا همینقدر کافی است که بدانید Layout ها در MVC همان نقشی را دارند که Master Page ها در Web Form دارند.

دستورات بعدی نیز که همان دستورات آشنای یک فایل Html ساده می باشند.

فایل فوق را به صورت زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <h1>Welcome to my website.</h1>
        <h3>To leave a message click the flowing link:</h3>
    </div>
</body>
</html>
```

حال برنامه را اجرا کنید تا خروجی را مشاهده نمائید.

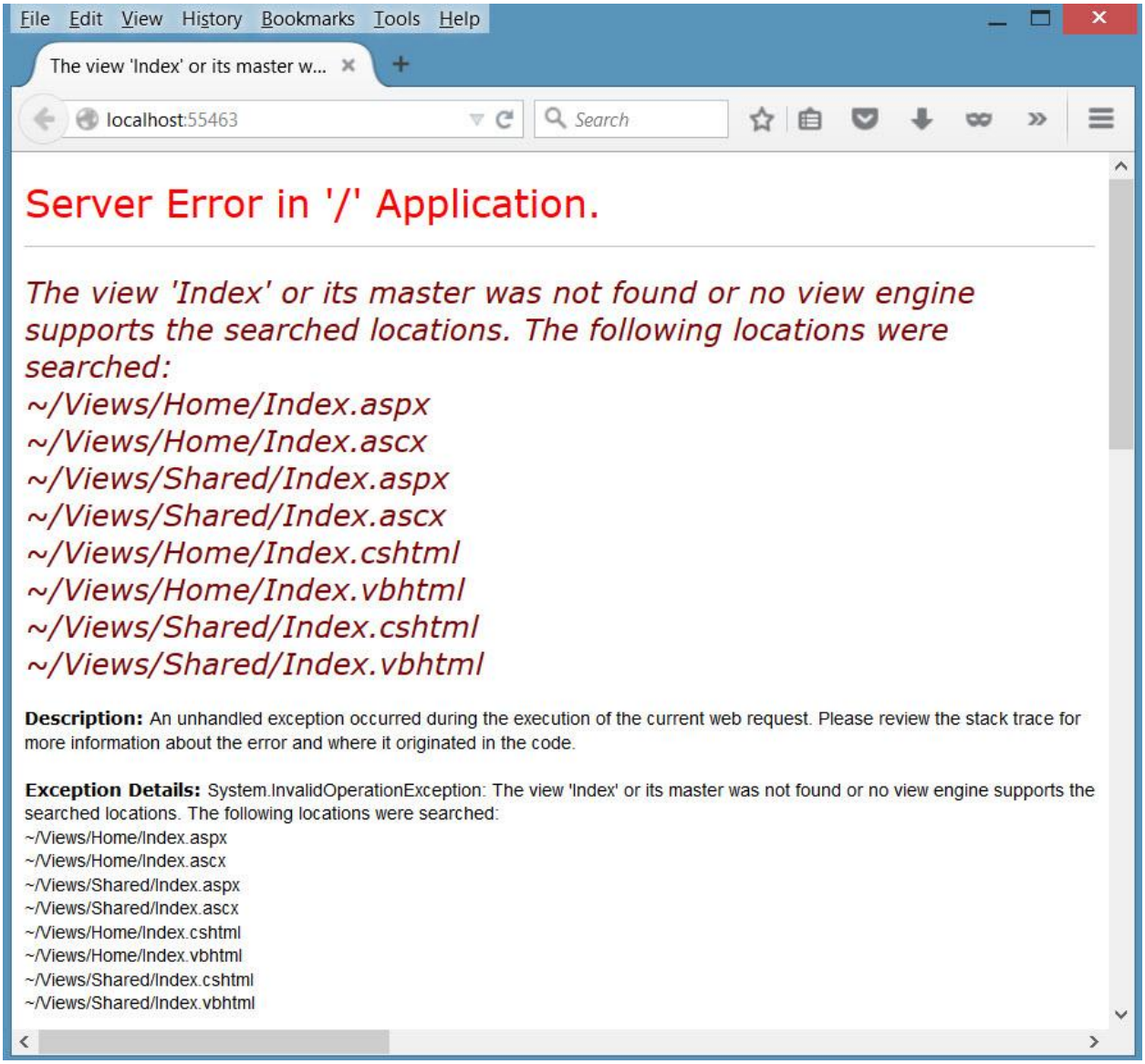


شکل ۳ - ۴

به آدرسی که در شکل فوق آمده است توجه کنید. همانطور که مشخص است این آدرس به کنترلر Home و متد Index اشاره می کند. چون در متد Index دستور return View(); قرار دارد MVC به دنبال ویوی هم نام با متد (Index) خواهد گشت و چون این متد درون کنترلر Home قرار دارد بنابر این به پوشه Views و در داخل آن به پوشه Home مراجعه خواهد کرد.

اگر ویوی مورد نظر تعریف نشده باشد چه اتفاقی خواهد افتاد؟

اگر MVC نتواند ویوی متناظر با اکشن متد مورد نظر را پیدا کند پیغام خطایی صادر خواهد کرد. برای اینکه این پیغام خطا را مشاهده کنید نام Index.cshtml را موقتا به Index2.cshtml تغییر دهید.(برای این کار بر روی فایل Index.cshtml کلیک راست کنید و گزینه Rename را انتخاب کنید و یا از کلید F2 برای اینکار استفاده نمائید). سپس مجددا برنامه را اجرا کنید. نتیجه مشابه به شکل زیر خواهد بود.



شکل ۳ - ۵

همانطور که مشاهده می کنید اجرای برنامه با خطا روبرو شده است. این خطا بیانگر این است که هیچ ویوی متناظر با متد Index مربوط به کنترلر Home پیدا نشده است.همانطور که مشاهده می کنید MVC در دو پوشه Home و Shared در داخل Views به جستجوی ویوی مورد نظر پرداخته که متاسفانه هیچ ویویی در آن پیدا نکرده است. اینکه MVC در کدا پوشه ها برای یافتن ویو ها جستجو می کند در تنظیمات مربوط به موتور

نمایشی MVC یعنی ViewEngine تعریف می شود که قابل تغییر نیز می باشد. در مورد پوشه Shared در بخش های بعدی صحبت خواهد شد.

افزودن یک اکشن متد دیگر جهت نمایش فرم اطلاعاتی

در این پروژه قصد داریم یک فرم اطلاعاتی به کاربر نشان دهیم تا کاربر بتواند با ارسال پیام با ما ارتباط برقرار کند.

برای این منظور متد Contact را به صورت زیر به کلاس HomeController اضافه کنید.

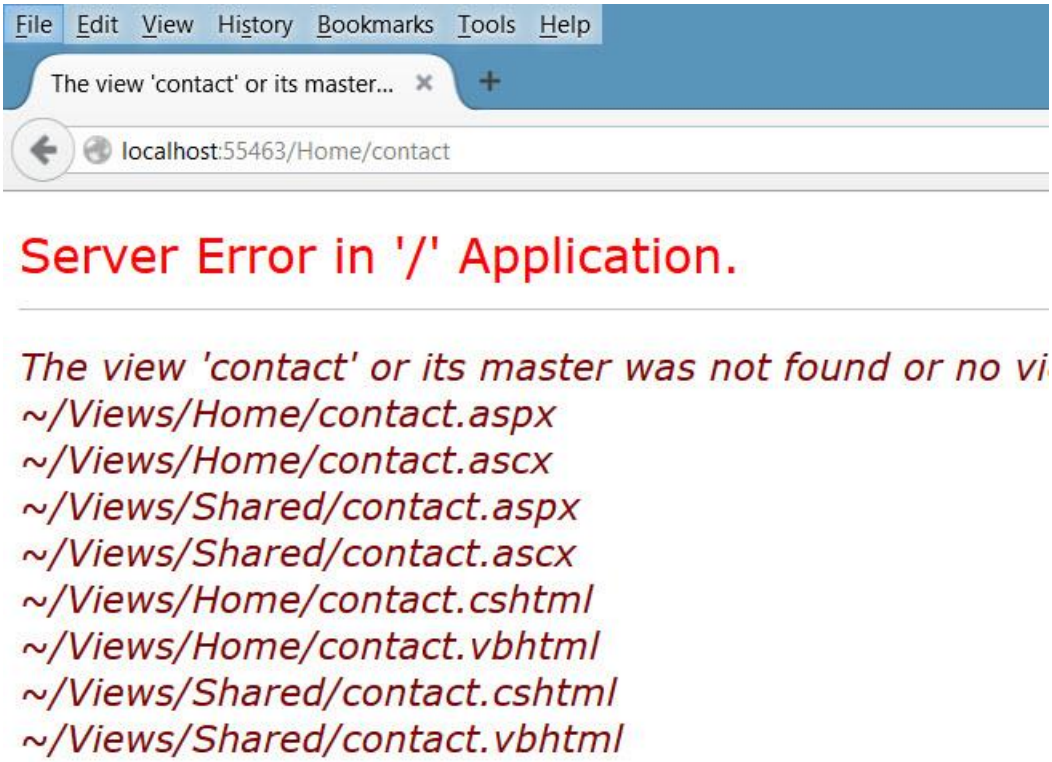
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace mkianiir.mvc.ContactForm.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ViewResult Index()
        {
            return View();
        }
        public ViewResult Contact()
        {
            return View();
        }
    }
}
```

حال برنامه را اجرا کنید و آدرس زیر را در مرور گر تایپ کنید:

http://localhost:55463/Home/contact

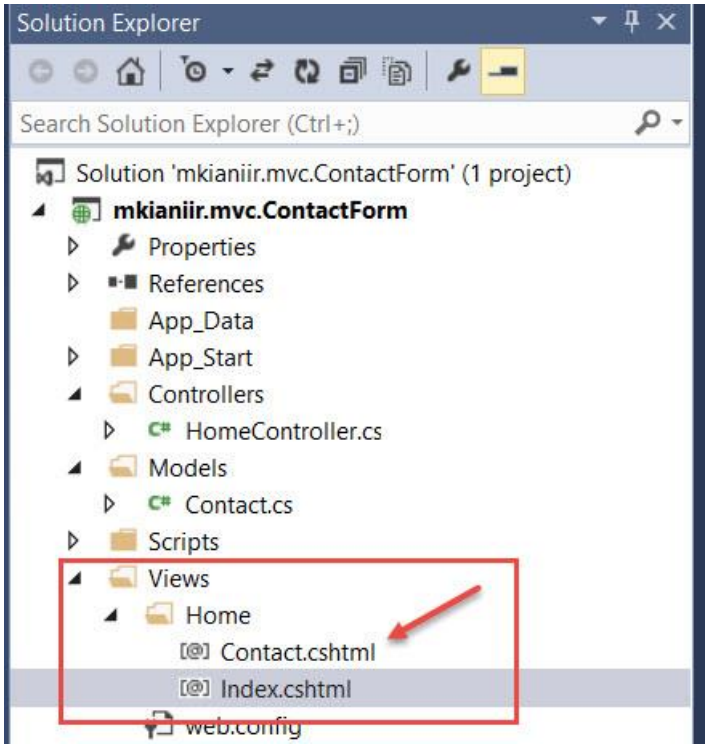
تذکر : عددی که برای پرت در آدرس بالا مشاهده می کنید ممکن است در کامپیوتر شما متفاوت با این باشد.

خروجی دستور بالا شبیه به شکل زیر خواهد بود.



شکل ۳ - ۶

این خروجی برای شما آشناست. بله همانطور که مشخص است MVC فایل contact.cshtml را پیدا نمی کند. همان مراحلی که برای ایجاد ویوی متناظر با Index انجام دادید برای Contact نیز تکرار کنید تا یک ویو به نام Contact اضافه شود.



شکل ۳ - ۷

حال فایل Contact.cshtml را به شکل زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Contact</title>
</head>
<body>
    <div>
        <h1>Contact and leave a message to help me:</h1>
    </div>
</body>
</html>
```

همانطور که در سطر هایلایت شده مشخص است یک متن ساده درون تگ h1 به فایل Contact.cshtml اضافه شده است. حال مجددا برنامه را اجرا و آدرس زیر را در مرورگر تایپ کنید.

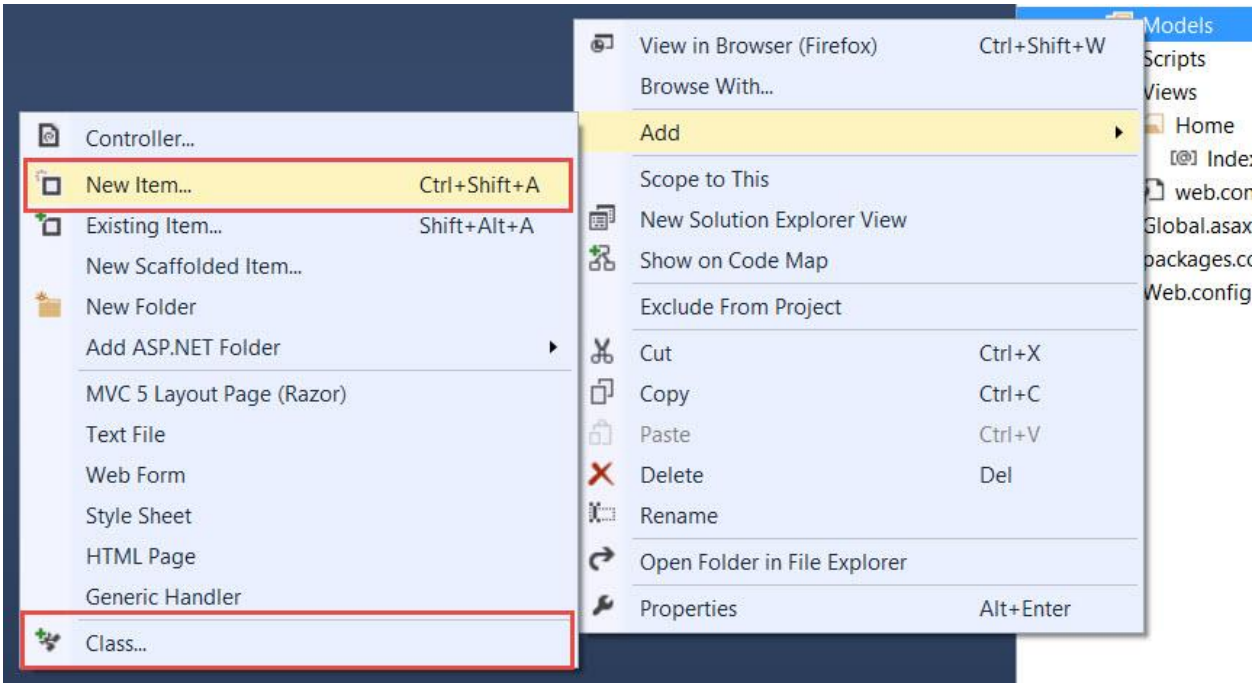
http://localhost:55463/Home/contact

همانطور که مشاهده می کنید برنامه بدون خطا اجرا می شود.

شکل ۳ - ۸

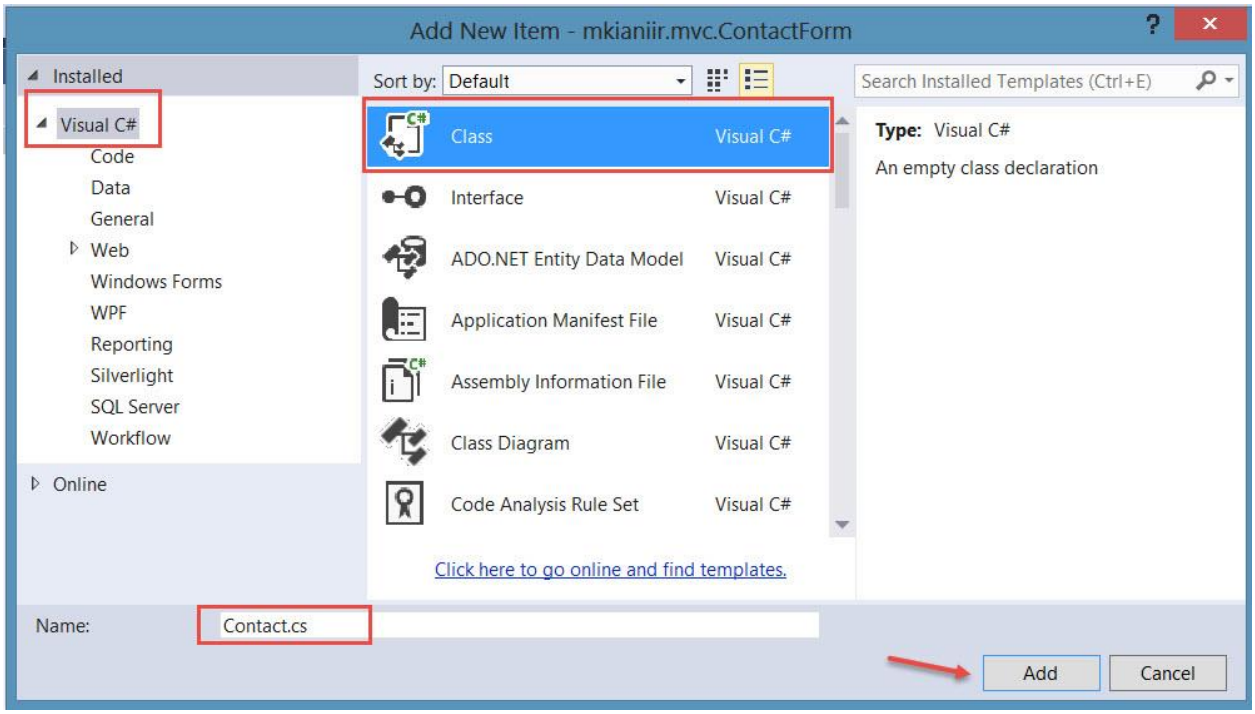
افزودن مدل

حال نیاز به یک مدل داریم که بتواند نمایانگر داده های مورد نظر باشد.بدین منظور بر روی پوشه Models کلیک راست کنید و از گزینه Add گزینه new item را کلیک کنید.(می توانید مستقیما گزینه Class را نیز انتخاب کنید. با این کار مستقیما گزینه Class در پنجره Add New Item انتخاب خواهد شد)



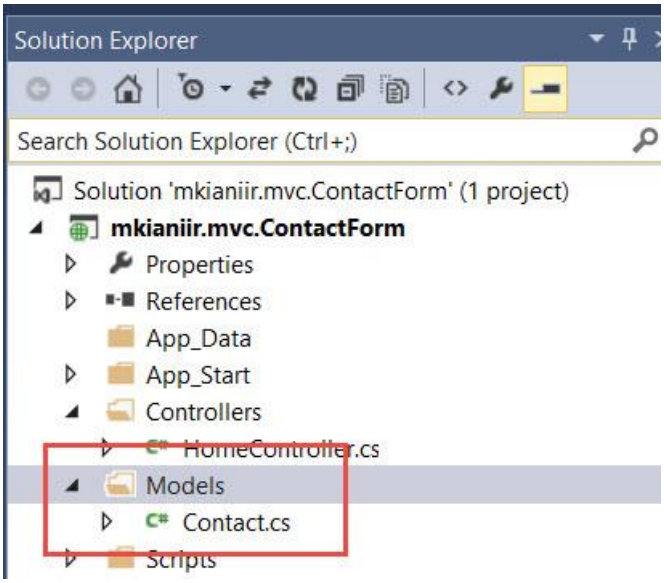
شکل ۳ - ۹

پنجره Add New Item باز خواهد شد:



شکل ۳ - ۱۰

در پنجره Add New Item از سمت چپ گزینه Visual C# و از قسمت وسط گزینه Class را انتخاب نمائید. نام کلاس را Contact قرار دهید و دکمه Add را بفشارید. با این کار یک کلاس به نام Contact.css در پوشه Models در Solution Explorere قرار خواهد گرفت.



شکل ۳ - ۱۱

بر روی فایل Contact.cs دوبا رکلیک کنید و کد های آن را به شکل زیر تغییر دهید.

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

namespace mkianiir.mvc.ContactForm.Models
{
    public class Contact
    {
        public String Name
        {
            get;
            set;
        }
        public String Phone
        {
            get;
            set;
        }
        public String Email
        {
            get;
            set;
        }
        public String Website
        {
            get;
            set;
        }

        public String Comment
        {
            get;
            set;
        }
    }
}
```

پنج خاصیت Name ، Phone، Email ، Website و Comment برای کلاس Contact تعریف کرده ایم. فایل Contact.cs را ذخیره کنید و ببندید.

ارسال مدل به ویو و ساختن فرم بر اساس خواص مدل

در ادامه می خواهیم کلاس Contact را به عنوان مدل برای ویویی که در مرحله قبل ساختیم معرفی کنیم.

فایل Contact.cshtml را باز کنید و دستور زیر را به ابتدای فایل اضافه کنید:

```
@model mkianiir.mvc.ContactForm.Models.Contact
```

تذکر: اگر فضای نام کلاس Contact شما متفاوت با فضای نامی است که من در پروژه دارم، می بایستی فضای نام خود را استفاده کنید.

توسط دستور @model می توانیم یک مدل را به یک ویو نسبت دهیم .این کار باعث می شود تا بتوانیم به صورت Strongly Type با ویو ها رفتار کنیم.

ایجاد فرم Contact بر اساس خواص تعریف شده در مدل آن

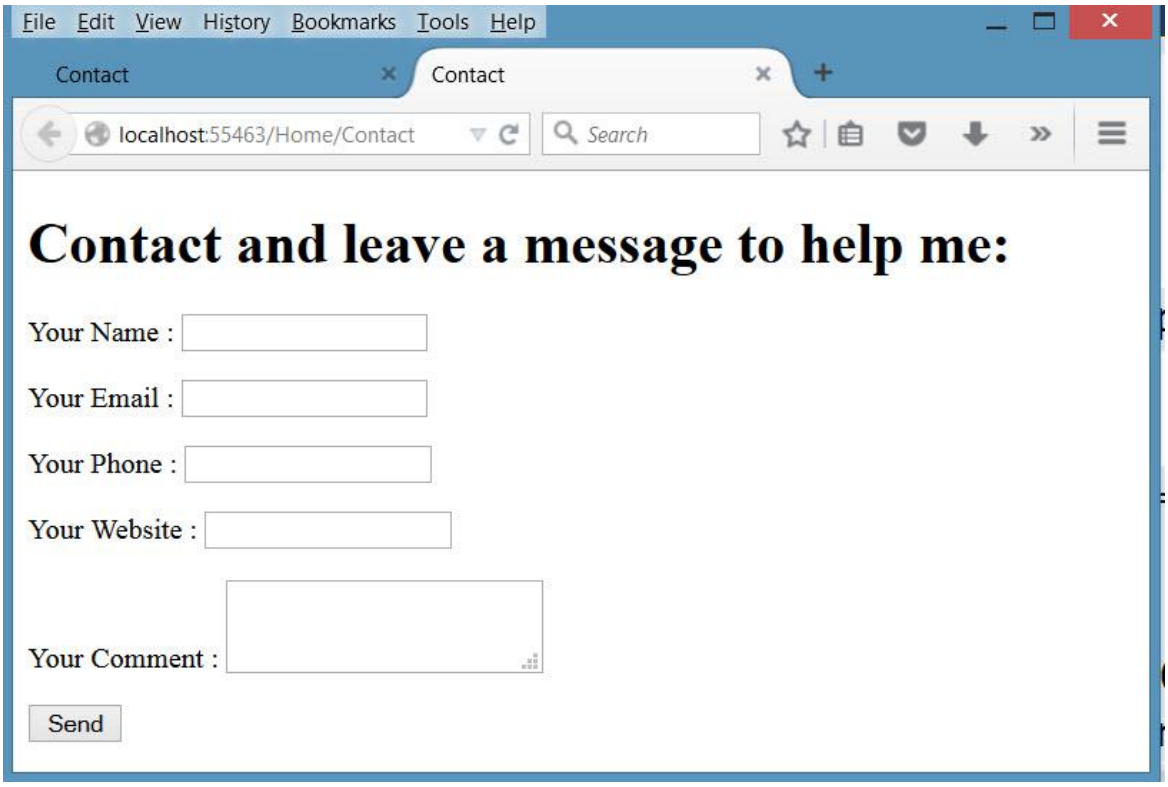
حال که کلاس Contact را به عنوان مدل برای ویو تعریف کردیم می توانیم اقدام به ایجاد فرم نمائیم:

```
@model mkianiir.mvc.ContactForm.Models.Contact
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Contact</title>
</head>
<body>
    <div>
        <h1>Contact and leave a message to help me:</h1>
        @using (Html.BeginForm())
        {
            <p>Your Name : @Html.TextBoxFor(x => x.Name)</p>
            <p>Your Email : @Html.TextBoxFor(x => x.Email)</p>
            <p>Your Phone : @Html.TextBoxFor(x => x.Phone)</p>
            <p>Your Website : @Html.TextBoxFor(x => x.Website)</p>
            <p>Your Comment : @Html.TextAreaFor(x => x.Comment)</p>
            <input type="submit" value="Send" />
        }
    </div>
</body>
</html>
```

دستورات هایلایت شده مربوط به موتور Razor می باشند که برای تولید کد های html به کار می روند که در ادامه شرح داده خواهند شد.

حال مجددا برنامه را اجرا کنید و آدرس زیر را در مرورگر تایپ کنید و کلید Enter را بفشارید.

http://localhost:55463/Home/contact



شکل ۳ - ۱۲

همانطور که مشاهده می کنید یک فرم اطلاعاتی دارای فیلد های متناظر با مدل داده ای ما ایجاد شده است.

پنجره View Source را در مرورگر خود باز کنید(در مرورگر IE بر روی صفحه کلیک راست کنید و گزینه View Source و در مرورگر Firefox گزینه View Page Source را کلیک کنید)

خروجی html تولید شده به شکل زیر خواهد بود:

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Contact</title>
</head>
<body>
    <div>
        <h1>Contact and leave a message to help me:</h1>
<form action="/Home/Contact" method="post">
          <p>Your Name : <input id="Name" name="Name" type="text"
value="" /></p>
            <p>Your Email : <input id="Email" name="Email"
type="text" value="" /></p>
```

```
            <p>Your Phone : <input id="Phone" name="Phone"
type="text" value="" /></p>
            <p>Your Website : <input id="Website" name="Website"
type="text" value="" /></p>
            <p>Your Comment : <textarea cols="20" id="Comment"
name="Comment" rows="2">
</textarea></p>
            <input type="submit" value="Send" />
</form>    </div>
</body>
</html>
```

چه اتفاقی افتاده است؟ بله درست است. دستوراتی که توسط موتور Razor نوشتیم توسط MVC در خروجی نهایی تبدیل به دستورات html شده اند.

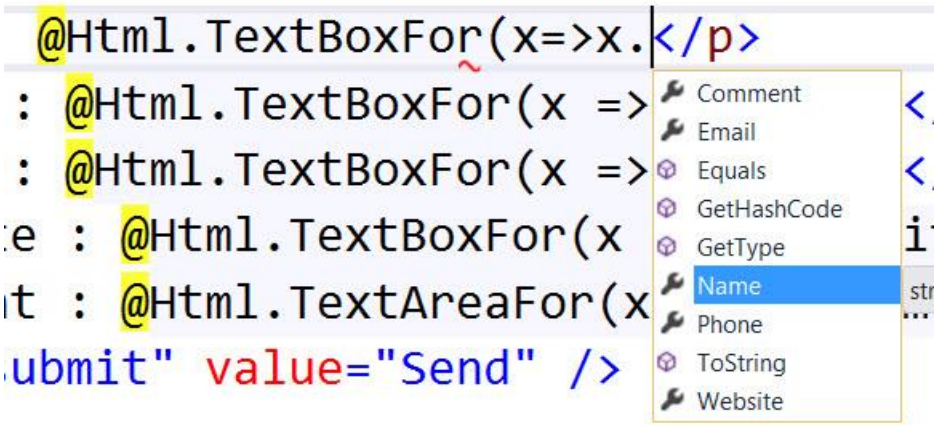
دستور Html.BeginForm برای ایجاد تگ Form به کار می رود. همچنین از متد های TextBoxFor برای ایجاد فیلد های متنی و از متد TextAreaFor برای ایجاد تگ textarea استفاده شده است. به متد های BeginForm ، TextBoxFor و TeaxtAreaFor اصلاحا متد های کمکی (Helper Method) گفته می شود که در فصلی مجزا به عملکرد آن ها خواهیم پرداخت.

یادآوری : به عباراتی شبیه به `x => x.Email` عبارات لامبدا گفته می شود. همانطور که مشاهده می کنید در تعریف فیلد ها نیز همین عبارات استفاده شده است.

روش دیگری که می توان به جای عبارات لامبدا از آن ها استفاده کرد استفاده از نام فیلد بصورت رشته می باشد. به عنوان مثال فیلد Name را می تواند به صورت زیر نیز تعریف کرد:

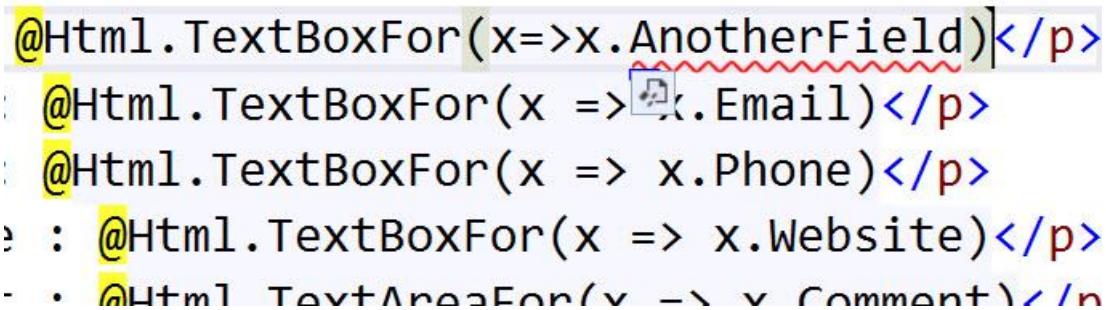
```
@Html.TextBox("Name")
```

اما از آنجا که ممکن است در این روش نام فیلد ها اشتباه تایپ شود استفاده از روش اول به شدت توصیه می گردد چرا که ویژوال استودیو در این زمینه نیز یاری رسان شما خواهد بود. زمانی که شما بعد از حرف x کاراکتر نقطه(dot) را تایپ می کنید لیستی از خواصی که در مدل شما تعریف شده است برای شما نشان داده خواهد شد و شما می توانید خاصیت مورد نظر را انتخاب کنید.



شکل ۳ - ۱۳

همچنین در این حالت اگر نام خاصیتی را به اشتباه تایپ کنید کمپایلر به شما اخطار خواهد داد و جلوی اجرای برنامه گرفته خواهد شد.



شکل ۳ - ۱۴

همانطور که مشاهده می کنید کمپایلر تشخیص می دهد که فیلدی به نام AnotherField در مدل مربوطه یعنی کلاس Contact تعریف نشده است.

دریافت اطلاعات کاربر و پردازش آن:

حال نوبت آن رسیده تا پس از تکمیل فرم توسط کاربر و کلیک کردن دکمه Send اطلاعات به کنترلر جهت پردازش ارسال شود. MVC عموما به دوروش درخواست های کاربر را دریافت و پردازش می کند. روش اول استفاده از آدرسی است که کاربر ارسال می کند که بر اساس آن کنترلر و اکشن متد مورد نظر شناسایی خواهد شد و روش دوم از طریق کلیک کردن دکمه و اصطلاحا post شدن فرم به سرور درخواست مورد نظر پردازش خواهد شد. در حالت دوم برای اکشن متد مورد نظر صفت HttpPost را به کار می بریم که در ادامه نحوه استفاده از آن ها خواهید دید.

یک متد دیگر به نام Contact به کنترلر HomeController به صورت زیر اضافه خواهیم کرد :

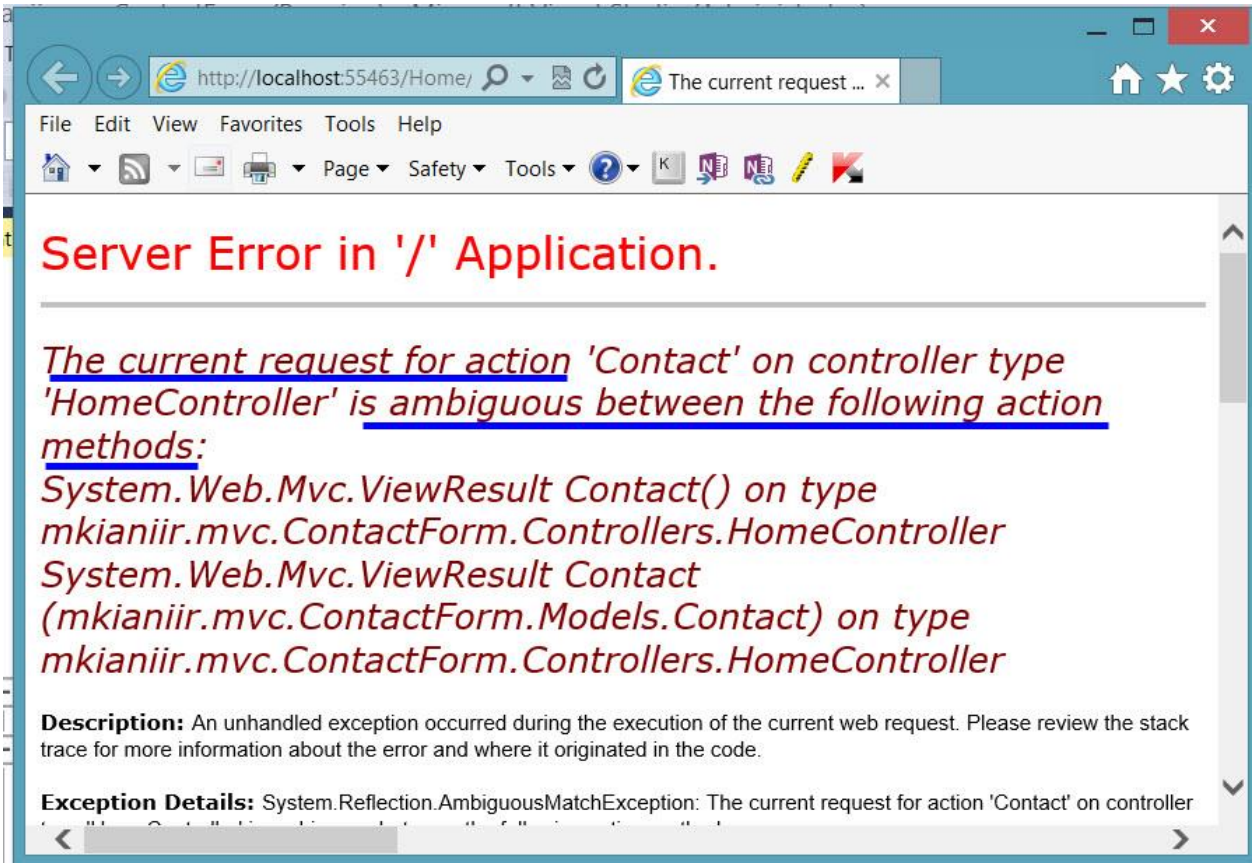
```
using mkianiir.mvc.ContactForm.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace mkianiir.mvc.ContactForm.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ViewResult Index()
        {
            return View();
        }
        public ViewResult Contact()
        {
            return View();
        }
        public ViewResult Contact(Contact model)
        {
            return View();
        }
    }
}
```

همانطور که مشخص شده است متد Contact جدید دارای یک آرگومان از نوع کلاس Contact می باشد. MVC به صورت خود کار داده های وارد شده توسط کاربر را به نمونه ای از کلاس Contact تبدیل کرده و برای پردازش به متد Contact ارسال خواهد کرد.

حال برنامه را اجرا کنید و آدرس زیر را در مرور گر تایپ کنید

http://localhost:55463/Home/Contact

نتیجه حاصل از اجرای دستورات فوق در شکل زیر نشان داده شده است:



شکل ۳ - ۱۵

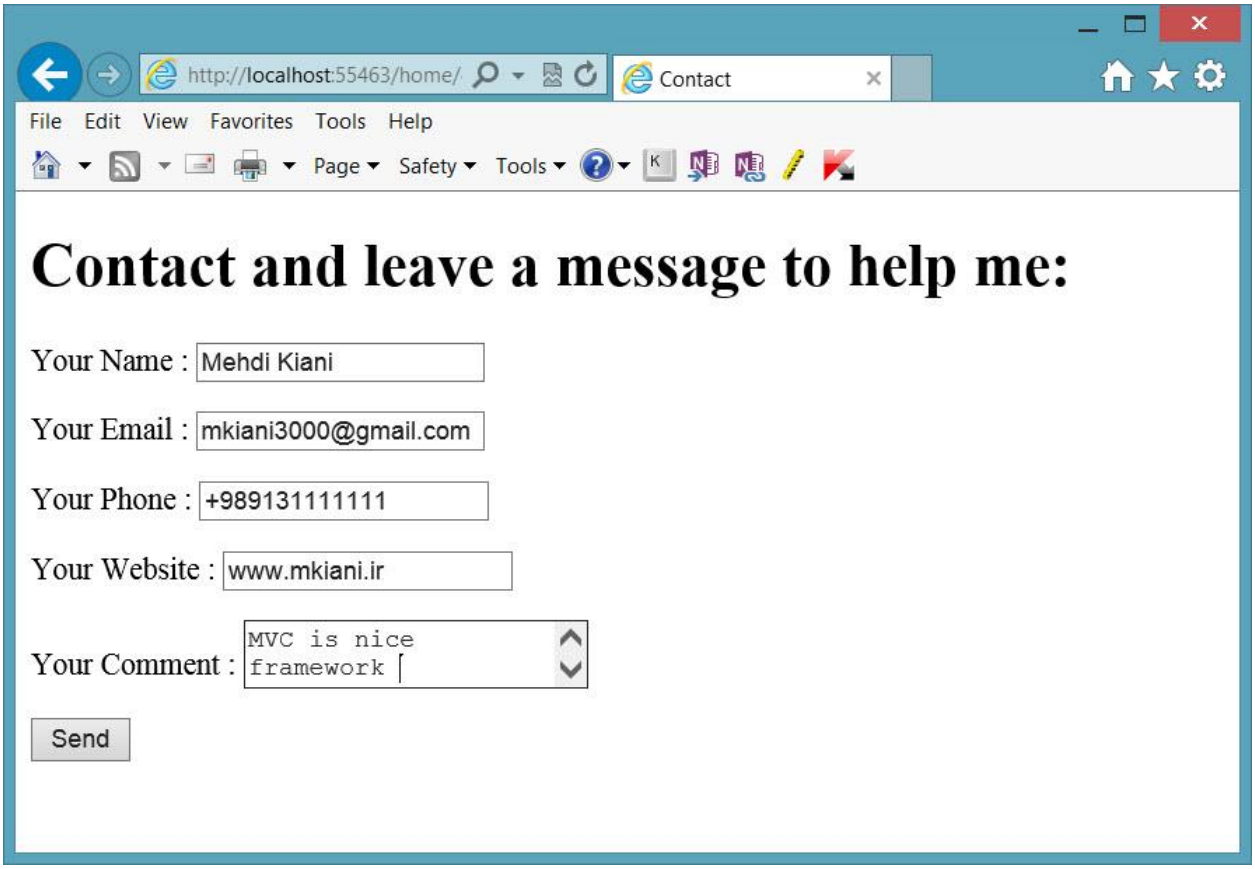
چه اتفاقی افتاد؟همانطور که می دانید آدرس فوق به این معناست که اکشن متدی به نام Contact در کنترلری به نام Home مد نظر کاربر است. اما در کنترلر مذکور دو متد به نام Contact تعریف شده است. اولی بدون آرگومان و دومی دارای یک آرگومان از نوع کلاس Contact.

بنابر این MVC نمی داند از بین دو متد Contact که در کنترلر Home تعریف شده است به کدام یک باید رجوع کند؟برای حل این مشکل باید از صفت HttpGet و HttpPost برای متد های Contact در کنترلر Home استفاده کنیم. متدی که می خواهیم زمانی که صفحه دفعه اول بارگزاری می شود(از طریق آدرس) از آن استفاده کند را با صفت HttpGet و متدی که می خواهیم پس از پر شدن فیلد های فرم توسط کاربر و کلیک شدن دکمه Send به آن رجوع شود را با صفت HttpPost (چیزی شبیه به عملیات Postback در Web Form ها)نشانه گذاری می کنیم. این دو کلاس یعنی HttpGet و HttpPost که نام کامل کلاس آن ها HttpGetAttribute و HttpPostAttribute می باشد دو صفت هستند که از کلاس ActionMethodSelectorAttribute در فضای نام System.Web.Mvc مشتق می شوند. کلاس HomeController را به صورت زیر تغییر دهید.

```
using mkianiir.mvc.ContactForm.Models;
using System;
using System.Collections.Generic;
```

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

namespace mkianiir.mvc.ContactForm.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ViewResult Index()
        {
            return View();
        }
        [HttpGet]
        public ViewResult Contact()
        {
            return View();
        }
        [HttpPost]
        public ViewResult Contact(Contact model)
        {
            return View();
        }
    }
}
```

حال برنامه را مجددا اجرا کنید.فیلد ها را پر کنید و دکمه Send را کلیک کنید:



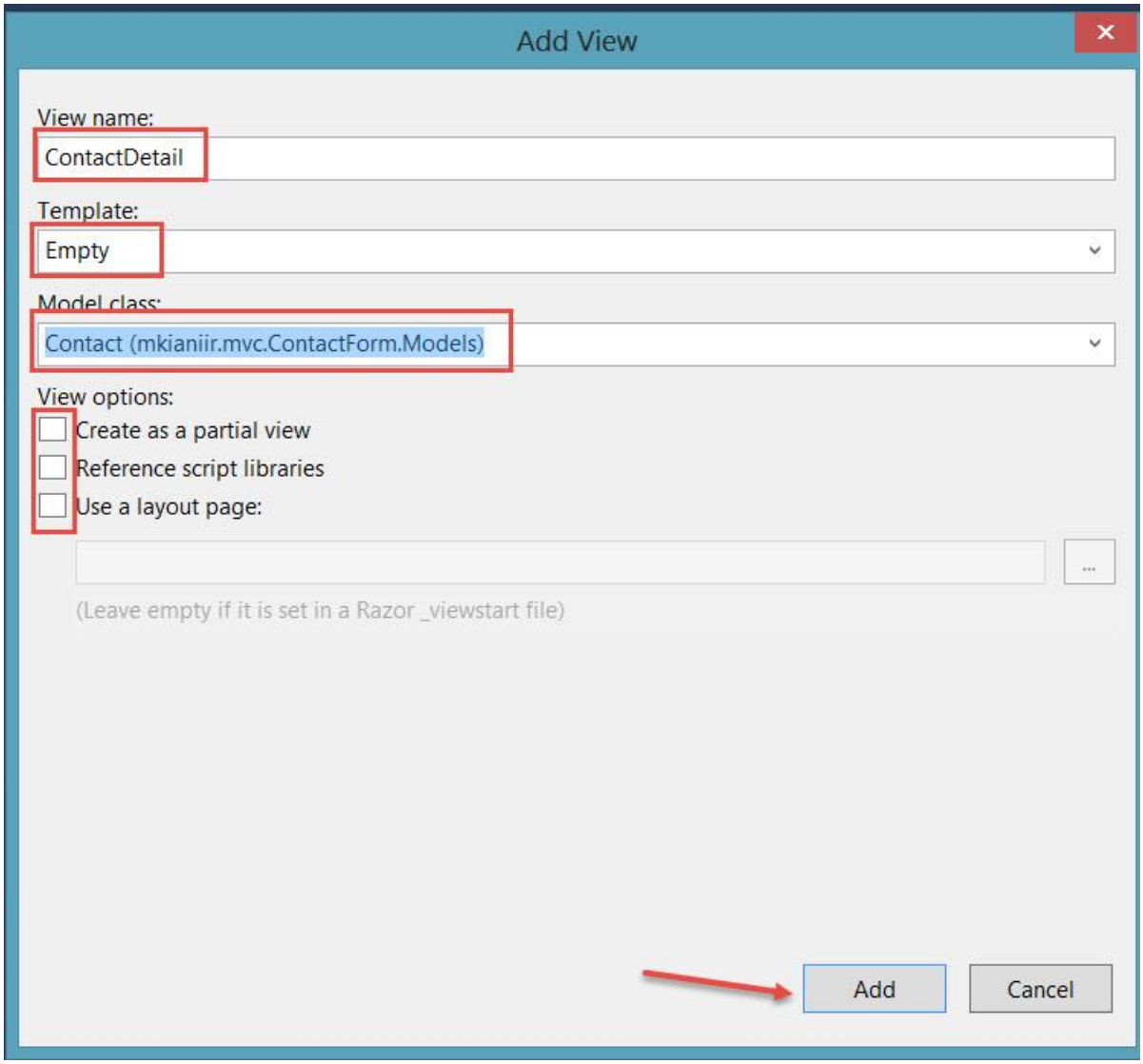
شکل ۲ - ۱۳

همانطور که مشاهده می کنید برنامه بدون خطا اجرا می گردد.پس ازپر شدن فیلد ها و فشرده شدن دکمه Send یک نمونه از مدل (در اینجا کلاس Contact) ایجاد شده و فیلد ها آن با اطلاعات تکمیل شده توسط کاربر پر می شود و به متد Contact ای که دارای صفت HttpPost می باشد در کنترلر Home ارسال می شود.

نمایش جزئیات و ارسال ایمیل به کاربر

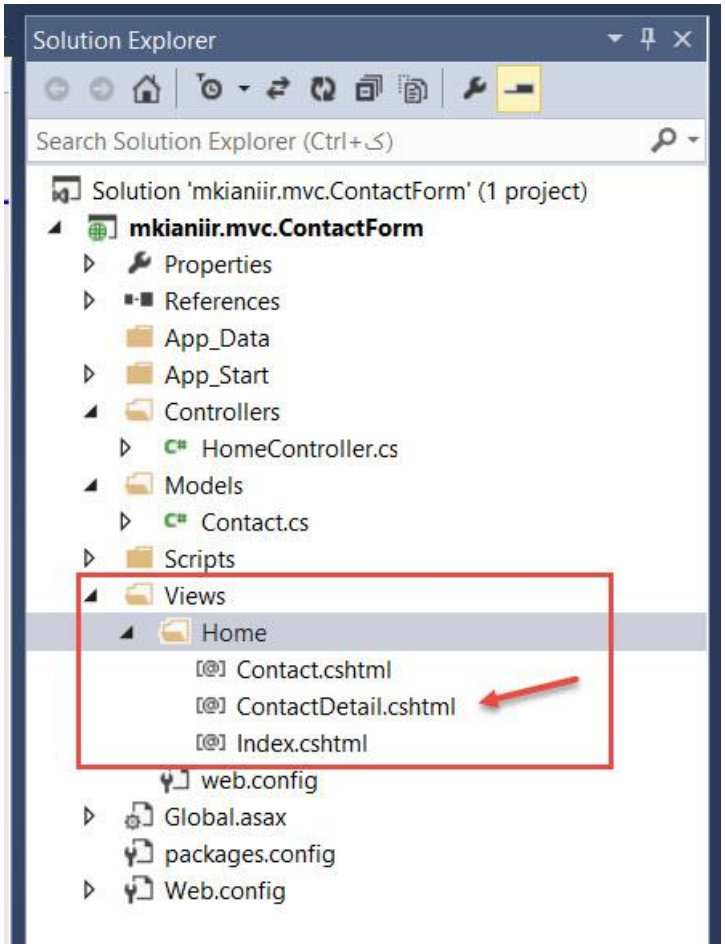
در این بخش می خواهیم اطلاعات دریافتی از کاربر را به یک ویوی دیگر جهت نمایش به کاربر ارسال نمائیم.

برای این منظور ابتدا بر روی پوشه Home در پوشه Views در Solution Explorer کلیک راست کنید و از گزینه Add گزینه View را کلیک کنید تا پنجره Add View باز شود. فیلد View name را برابر با ContactDetail مقدار دهی کنید. در قسمت Template گزینه Empty را انتخاب و سپس از قسمت Model Class نام کلاس Contact را انتخاب کنید. مطمئن شوید که هیچ یک از گزینه های View options انتخاب نشده باشند. پس از انجام تنظیمات بر روی دکمه Add کلیک کنید.



شکل ۳ - ۱۶

پس از اینکه بر روی دکمه Add کلیک کنید یک View با نام ContactDetail در پوشه Home در پوشه Views ایجاد می شود.



شکل ۳ - ۱۷

بر روی فایل ContactDetail.cshtml دوبار کلیک کنید. کد های این فایل به صورت زیر خواهند بود:

```
@model mkianiir.mvc.ContactForm.Models.Contact

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ContactDetail</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

همانطور که در خط اول مشاهده می کنید کلاس Contact به عنوان مدل این ویو تعریف شده است.

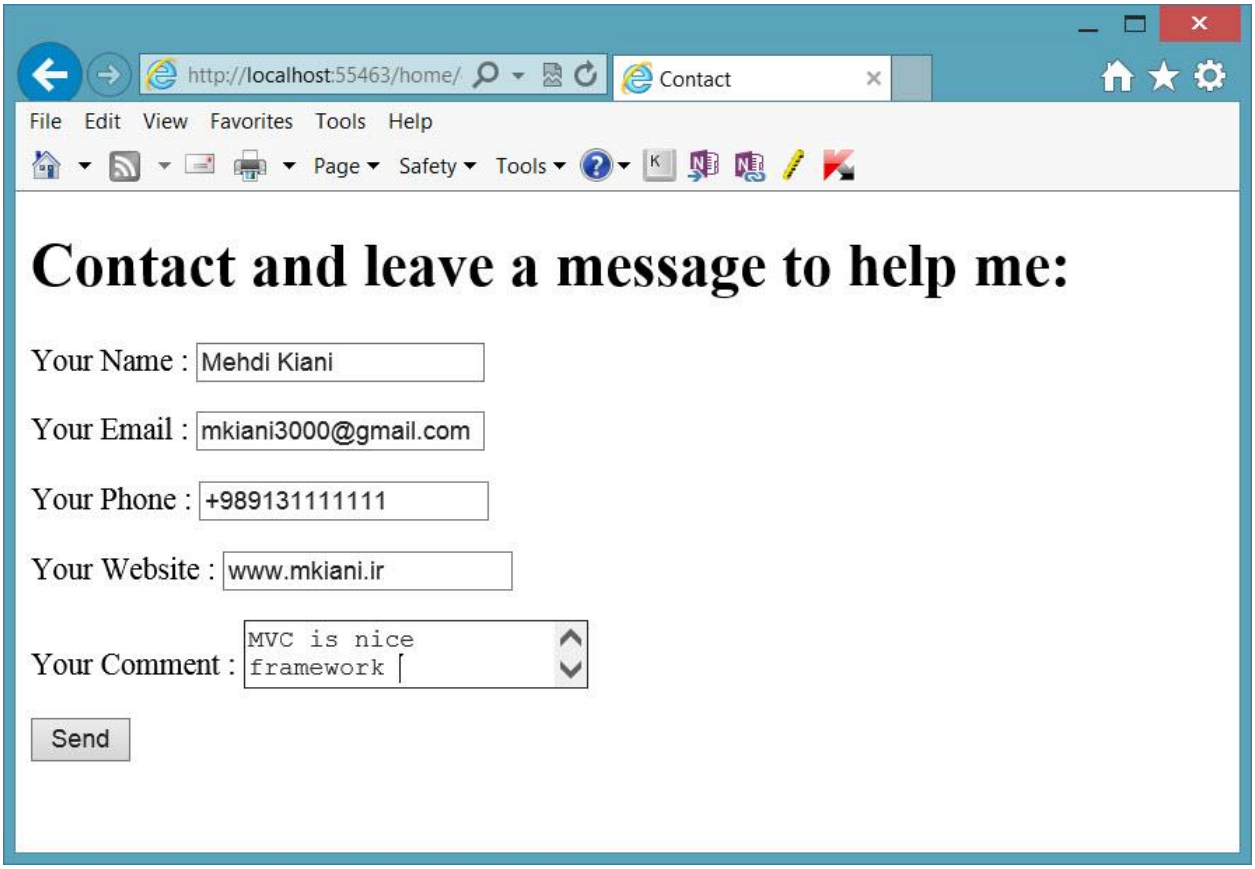
کد های دورن تگ body را به صورت زیر تغییر دهید:

```
<div>
        <h1>Dear  '@Model.Name'</h1>
        <p>Thank you spending your time.</p>
        <p> an email was sent to '@Model.Email' with a tracking
code.</p>
        later you can check your answer in the site.
    </div>
```

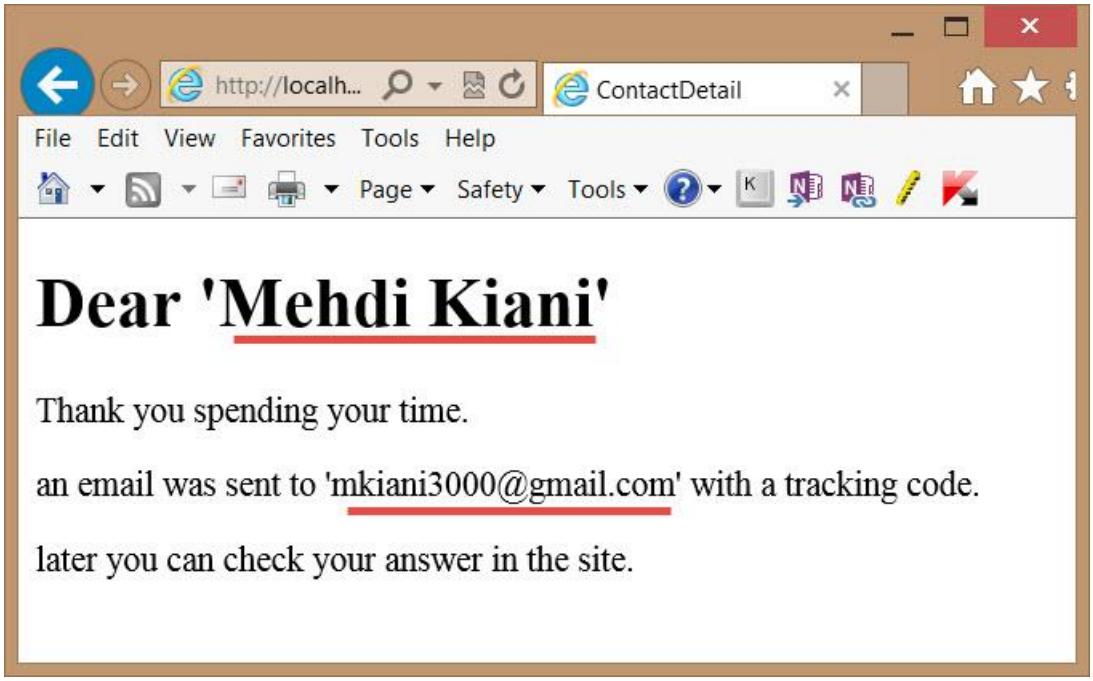
حال متد Contact ای که دارای صفت HttpPost می باشد را به صورت زیر تغییر دهید:

```
[HttpPost]
        public ViewResult Contact(Contact model)
        {
            return View("ContactDetail", model);
        }
```

همانطور که مشاهده می کنید به متد View دو آرگومان اضافه شده است. آرگومان اول نام ویویی است که می خواهیم به آن مراجعه کنیم و آرگومان دوم نمونه ای از کلاس Contact می باشد که فیلد های آن توسط کاربر تکمیل شده است. برنامه را با آدرس مشخص شده درشکل اجرا کنید و نتیجه را مشاهده کنید.



شکل ۳ - ۱۸



شکل ۳ - ۱۹

همانطور که مشاهده می کنید پس از فشرده شدن دکمه Send اطلاعات به کنترلر Home و به متد Contact دارای صفت HttpPost ارسال شده است و آن متد ویوی ContactDetail را فراخوانی کرده و داده های کاربر را تحت عنوان نمونه ای از کلاس Contact به ارسال کرده و ContactDetail نیز بر اساس اطلاعات دریافتی خروجی html مورد نظر را تعریف کرده است.

نکته: هماطنور که در دستورات فایل ContactDetail مشاهده کردید توسط دستور @model (با حرف کوچک m) اقدام به معرفی مدل به ویو کرده ایم و توسط دستور @Model (با حرف بزرگ M) به فیلد های مدل دسترسی پیدا کرده ایم.

سورس html نهایی که برای ContactDetail تولید شده است به صورت زیر خواهد بود:

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ContactDetail</title>
</head>
<body>
    <div>
        <h1>Dear  'Mehdi Kiani'</h1>
        <p>Thank you spending your time.</p>
        <p> an email was sent to 'mkiani3000@gmail.com' with a
tracking code.</p>
        later you can check your answer in the site.
    </div>
</body>
</html>
```

برنامه نویسان ASP.Net WebForm گاها با معضلی به نام ViewState برخورد می کنند که برای نگهداری حالت(State Management) در برنامه های WebForm مورد استفاده قرار می گیرد. در MVC چیزی به نام ViewState وجود ندارد.چون مکانیزم MVC کاملا متفاوت با آن چیزی است که در WebForm مشاهده می کنید. نبود ViewState در خروجی های html نهایی یک مزیت نسبت به برنامه های WebForm می باشد.چون حجم ViewState ها در برنامه های WebForm می تواند گاها بسیار زیاد شده و این می تواند هم در سرعت بارگزاری و هم در مصرف پهنای باند اینترنت تاثیر بگذارد. اما نکته مهم این است که الزاما هر برنامه ای که با MVC نوشته می شود نمی تواند گفت که صد در صد سریعتر از برنامه ای است که WebForm نوشته می شود. چون این موضوع به پارامتر های زیادی می تواند بستگی داشته باشد که یکی از مهمترین آن ها نحوه کد نویسی می باشد. یک کد بد اگر در بهترین فریم ورک هم نوشته شود بد است و بد اجرا خواهد شد!

ایجاد لینک برای برقرای ارتباط بین اکشن متد ها

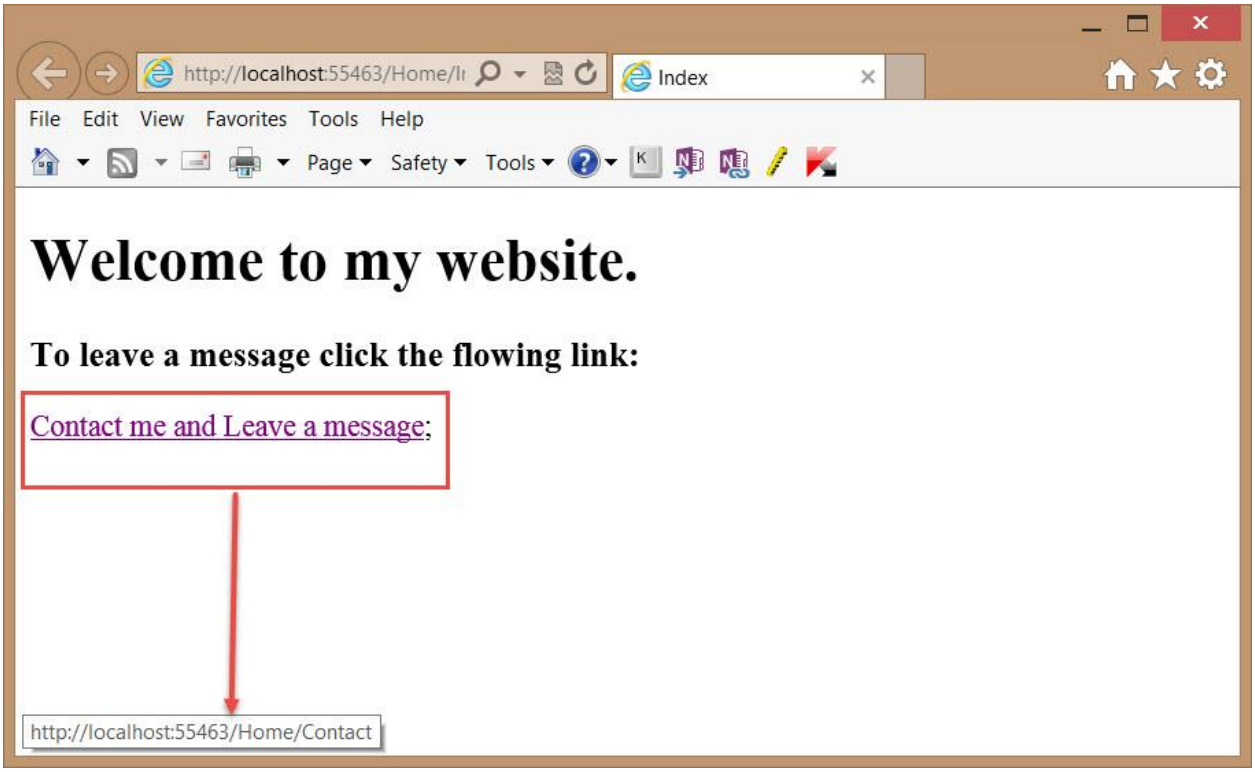
همانطور که می دانید برای ایجاد یک لینک در html از تگ a با صفت href استفاده می کنیم. متد راهنمای ActionLink امکان ایجاد یک تگ a با صفت href را مهیا می کند که بتوانید بین اکشن ها سوئیچ کنید.در این قسمت می خواهیم یک لینک با استفاده از متد ActionLink به اکشن Contact ایجاد نمائیم.

برای این منظور فایل Index.cshtml را باز کنید و دستورات آن را مطابق زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <h1>Welcome to my website.</h1>
        <h3>To leave a message click the flowing link:</h3>
        @Html.ActionLink("Contact me and Leave a message",
"Contact");
    </div>
</body>
</html>
```

همانطور که مشاهده می کنید متد ActionLink فوق با دو آرگومان مورد استفاده قرار گرفته است. اولین پارامتر عنوان لینک خواهد بود.(همان صفت value در تگ a) و دومین پارامتر نام اکشن متدی خواهد بود که می خواهیم به آن اشاره کنیم(ایجاد صفت href در تگ a)

حال برنامه را مجددا اجرا کنید. همانطور که مشاهده می کنید یک لینک که اشاره به اکشن متد Contact دارد ایجاد شده است که با کلیک کردن بر روی آن به اکشن Contact ارجاع داده خواهید شد.



شکل ۳ - ۲۰

ارسال کد رهگیری به کاربر

در این قسمت می خواهیم یک کد رهگیری تولید و آن را به ایمیل کاربر ارسال نمائیم. برای ارسال ایمیل می توانیم از کلاس MailMessage در فضای نام System.Net.Mail استفاده کنیم. برای این منظور متد Contact در کلاس HomeController را به صورت زیر تغییر دهید:

```
[HttpPost]
     public ViewResult Contact(Contact model)
     {

          MailMessage mail = new MailMessage();
          mail.To.Add(model.Email);
          mail.From = new MailAddress("Sender email address");
          mail.Subject = "Email From www.mkiani.ir";
          mail.Body = String.Format("<p>Dear: {0} </p><p>Your
Tracking Code Is :{1}</p>", model.Name, Guid.NewGuid());
          mail.IsBodyHtml = true;
          SmtpClient smtp = new SmtpClient();
          smtp.Host = "smtp.gmail.com";
          smtp.Port = 587;
          smtp.UseDefaultCredentials = false;
          smtp.Credentials = new System.Net.NetworkCredential
          ("Your user name", Your password");
```

```
            smtp.EnableSsl = true;
            smtp.Send(mail);


            return View("ContactDetail", model);

        }
```

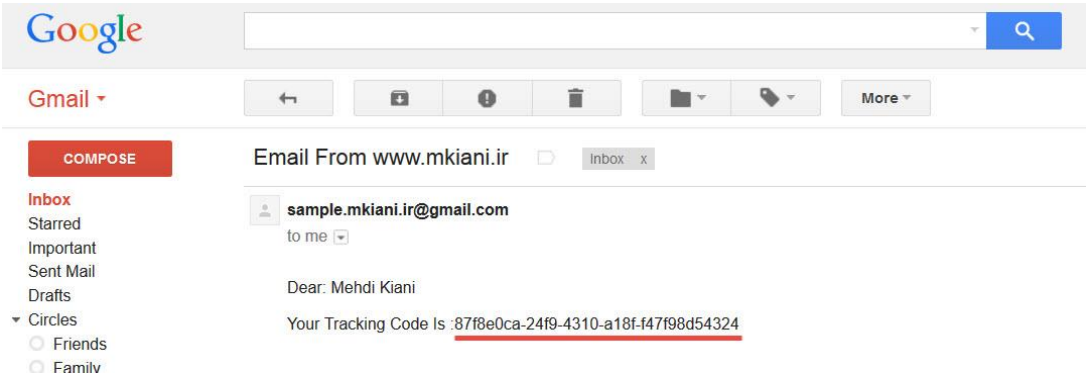
در قسمت mail.From می بایستی آدرس ایمیل فرستنده را بنویسید. همچنین در قسمت smtp.credentials می بایستی نام کاربری و کلمه عبور مربوط به ایمیل فرستنده(آدرس ایمیل خودتان) را وارد نمائید.

جهت اطلاعات بیشتر و نحوه کار با کلاس MailMessage به آدرس زیر مراجعه نمائید.

https://msdn.microsoft.com/en-us/library/system.net.mail.mailmessage٢٨٪v=vs.٢٩٪١١٠.aspx

حال مجددا برنامه را اجرا کنید. به صفحه Contact بروید و پس از پر کردن فیلد های درون صفحه روی دکمه Send کلیک کنید.در این حالت شما به ویو ContentDetail هدایت می شوید.همچنین یک ایمیل به آدرسی که در فیلد ایمیل صفحه Contact تایپ کرده بودید به همراه یک کد رهگیری ارسال خواهد شد.

کد رهگیری با استفاده از کلاس Guid ایجاد شده است. همانطور که می دانید این کلاس یک رشته ۳۲ کاراکتری یونیک ایجاد می کند. شما می توانید در برنامه خود الگوریتم مخصوص به خود را جهت ایجاد کد های رهگیری ایجاد نمائید.



شکل ۳ - ۲۱

اعتبار سنجی داده های کاربر

هر زمان که شما از کاربران نهایی اطلاعاتی را دریافت می کنید می بایستی صحت اطلاعات را بررسی نمائید. به این عمل اعتبار سنجی داده ها(Data Validation)می گویند.در MVC اعتبار سنجی داده ها توسط صفت ها انجام می شود. این صفت ها کلاس هایی هستند در فضای نام

System.ComponentModel.DataAnnotations که از کلاس ValidationAttribute مشتق شده و جهت اعتبار سنجی داده ها به کار می روند.

کلاس Contact را از پنجره Solution Explorer در پوشه Models باز کنید و کد های آن را به شکل زید تغییر دهید:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.ComponentModel.DataAnnotations;
namespace mkianiir.mvc.ContactForm.Models
{
    public class Contact
    {

        [Required(ErrorMessage = "The name field is required.")]
        public String Name
        {
            get;
            set;
        }
        public String Phone
        {
            get;
            set;
        }
        [Required(ErrorMessage = "The email field is required.")]
        public String Email
        {
            get;
            set;
        }
        public String Website
        {
            get;
            set;
        }
        [Required(ErrorMessage = "The comment field is required.")]
        public String Comment
        {
            get;
            set;
        }
    }
}
```

همانطور که مشاهده می کنید برای خواص Name ، Email و Comment سه صفت Required با یک آرگومان به نام ErrorMessage مورد استفاده قرار گرفته است.صفت Required مشخص می کند که این فیلد می بایستی حتما مقدار داشته باشد و آرگومان ErrorMessage متن خطایی است که در زمان خطای اعتبار سنجی داده ها به کاربر می خواهیم نشان داده شود. همانطور که مشاهده می کنید فضای نام System.ComponentModel.DataAnnotations در ابتدای کلاس Contact اضافه شده است.

بررسی وضعیت اعتبار سنجی مدل قبل از پردازش آن

حال برای اینکه بررسی کنیم که مدل در وضعیت صحیح قرار دارد یا خیر از خاصیت ModelState در کلاس Controller استفاده می کنیم. هر کنترلر دارای خاصیتی به نام ModelState می باشد. خاصیت ModelState یک دیکشنری می باشد که دارای خاصیت IsValid است. چنانچه این خاصیت مقدار true را برگرداند یعنی اعتبارسنجی داده ها به درستی انجام شده و خطایی در داده ها وجود ندارد اما اگر مقدار false برگردانده شود به این معناست که خطایی در داده های ورودی کاربر رخ داده است.

متد Contact را در کلاس HomeController به صورت زیر تغییر دهید:

```
[HttpPost]
        public ViewResult Contact(Contact model)
        {
            if (ModelState.IsValid)
            {

                MailMessage mail = new MailMessage();
                mail.To.Add(model.Email);
                mail.From = new
MailAddress("sample.mkiani.ir@gmail.com");
                mail.Subject = "Email From www.mkiani.ir";
                mail.Body = String.Format("<p>Dear: {0} </p><p>Your
Tracking Code Is :{1}</p>", model.Name, Guid.NewGuid());
                mail.IsBodyHtml = true;
                SmtpClient smtp = new SmtpClient();
                smtp.Host = "smtp.gmail.com";
                smtp.Port = 587;
                smtp.UseDefaultCredentials = false;
                smtp.Credentials = new System.Net.NetworkCredential
                ("sample.mkiani.ir@gmail.com", "!@#456QWErty
smtp.EnableSsl = true;
                smtp.Send(mail);
```

```
            return View("ContactDetail", model);
        }
        else
        {
            return View();
        }

    }
```

همانطور که مشاهده می کنید با استفاده از خاصیت IsValid مربوط به خاصیت ModelState بررسی کرده ایم که داده های ورودی دارای خطا هستند یا خیر.چنانچه مدل داده ای معتبر نباشد به خود View رجوع خواهد شد(نگاه کنید به دستورات بلاک else)

حال برنامه را اجرا کنید و بدون پر کردن فیلد های لازم دکمه Send را کلیک کنید. مشاهده خواهید کرد برنامه از ورود شما به صفحه ContactDetail جلوگیری کرده و شما را مجددا به همان صفحه Contact هدایت خواهد کرد.(چرا؟)

اعلان وضعیت اعتبار سنجی به کاربر

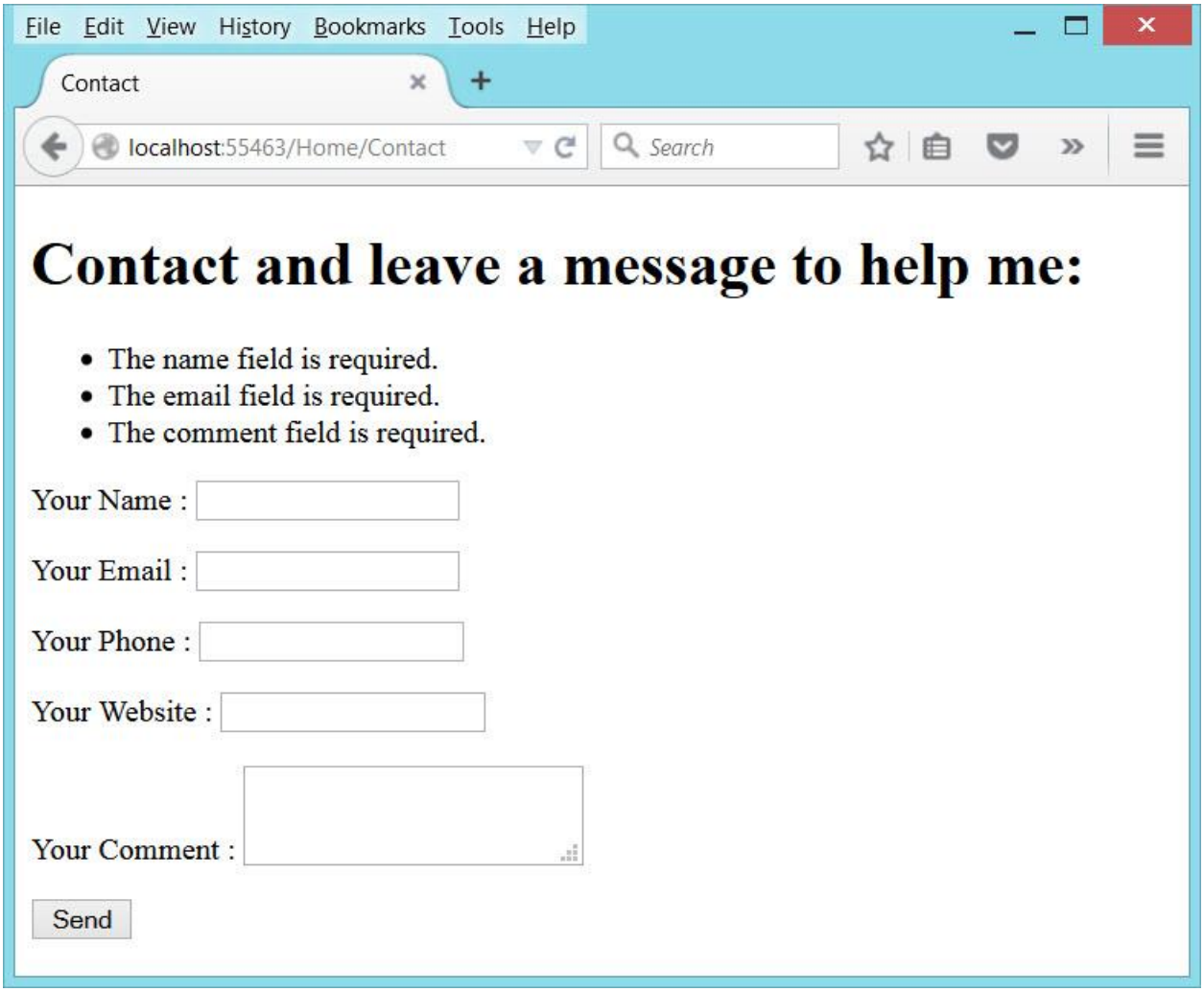
دستورات بالا فقط بررسی می کند که داده های ورودی معتبر هستند یا خیر.چنانچه داده های ورودی نامعتبر باشند کاربر مجددا به همان فرم Contact هدایت می شود اما به کاربر هیچ پیامی مبنی بر اینکه داده های ورودی معتبر نمی باشند داده نمی شود. یکی از روش های نمایش اطلاعات اعتبار سنجی به کاربر استفاده از متد راهنمای ValidationSummary می باشد.

برای این منظور فایل Contact.cshtml را باز کنید و کدهای آن را طبق زیر تغییر دهید. به قسمت هایلایت زرد رنگ توجه کنید.

```
@model mkianiir.mvc.ContactForm.Models.Contact
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Contact</title>
</head>
<body>
```

```
    <div>
        <h1>Contact and leave a message to help me:</h1>
        @Html.ValidationSummary()
        @using (Html.BeginForm())
        {
            <p>Your Name : @Html.TextBoxFor(x=>x.Name)</p>
            <p>Your Email : @Html.TextBoxFor(x => x.Email)</p>
            <p>Your Phone : @Html.TextBoxFor(x => x.Phone)</p>
            <p>Your Website : @Html.TextBoxFor(x => x.Website)</p>
            <p>Your Comment : @Html.TextAreaFor(x => x.Comment)</p>
            <input type="submit" value="Send" />
        }
    </div>
</body>
</html>
```

حال برنامه را اجرا کنید و بدون پر کردن فیلد های لازم دکمه Send را کلیک کنید. همانطور که مشاهده می کنید پیغام های خطای اعتبار سنجی به کاربر نشان داده می شود.



شکل ۳ - ۲۲

تغییر درظاهر برنامه(استفاده از کتابخانه Bootstrap)

یکی از جنبه های دیگر برنامه نویسی تحت وب که شاید مستقیما به عملکرد منطقی برنامه ارتباطی نداشته باشد اما به کاربرپسند بودن برنامه کمک شایانی می کند استفاده از شیوه نامه ها (فایل های css)جهت زیبا سازی برنامه می باشد. در این بخش می خواهیم تغییراتی را در ظاهر برنامه به وجود بیاوریم تا برنامه ظاهری کاربر پسند تر به خود بگیرد.

یکی از کتابخانه هایی که امروزه برای این منظور در بیشتر برنامه های تحت وب مورد استفاده قرار می گیرد کتابخانه Bootstrap می باشد. مسلما امکان توضیح همه بخش های کتابخانه مذکور در این کتاب امکان پذیر نیست. لذا توصیه می شود جهت آشنایی با کامپوننت های این کتابخانه به آدرس زیر مراجعه نمائید

http://getbootstrap.com/components /

نصب کتابخانه Bootstrap

برای دریافت و نصب کتاب خانه Bootstrap می توانید به یکی از دو روش زیر عمل کنید.

۱- به سایت مرجع این کتابخانه به آدرس http://getbootstrap.com مراجعه کنید و از بخش دانلود، این کتابخانه را دانلود نمائید و به صورت دستی به پروژه خود اضافه نمائید.

۲- با استفاده از کنسول نیوگت (Nuget) مستقیما این کتابخانه را دانلود و در پروژه نصب نمائید.(استفاده از روش دوم توصیه می شود).جهت آشنایی با nuget به آدرس http://docs.nuget.org/ مراجعه نمائید.

برای این منظور در ویژوال استویدو از منوی Tools به گزینه Library Package Manager رفته و از زیر منو های آن گزینه Package Manager Console را انتخاب کنید. پنجره Package Manager Console در پاین ویژوال استودیو نمایان می شود.

در خط فرمان این پنجره دستور زیر را وارد نمائید:

```
Install-Package -version 3.0.0 bootstrap
```

```
PM> Install-Package -version 3.0.0 bootstrap
Attempting to resolve dependency 'jquery (≥ 1.9.0)'.
Installing 'jQuery 1.9.1'.
Successfully installed 'jQuery 1.9.1'.
Installing 'bootstrap 3.0.0'.
Successfully installed 'bootstrap 3.0.0'.
Adding 'jQuery 1.9.1' to mkianiir.mvc.ContactForm.
Successfully added 'jQuery 1.9.1' to mkianiir.mvc.ContactForm.
Adding 'bootstrap 3.0.0' to mkianiir.mvc.ContactForm.
Successfully added 'bootstrap 3.0.0' to mkianiir.mvc.ContactForm.
```

شکل ۳ - ۲۳

دستور فوق باعث می شود نسخه ۳ از این کتابخانه نصب شود. پس از نصب کتابخانه Bootsrtap اگر به پنجره Solution Explorer بنگرید متوجه خواهید شد که دو پوشه به نام های Scriptsو contents اضافه شده است. در پوشه contents فایل های css و در پوشه scripts فایل های js (جاوا اسکریپت) قرار گرفته اند. همچنین پوشه دیگری به نام fonts جهت استفاده از فونت های مورد نیاز.

در این بخش ما به همه جنبه های Bootsrtap نخواهیم پرداخت و صرفا استفاده از چند کلاس ساده برای ایجاد تغییر در ظاهر برنامه بسنده خواهیم کرد.

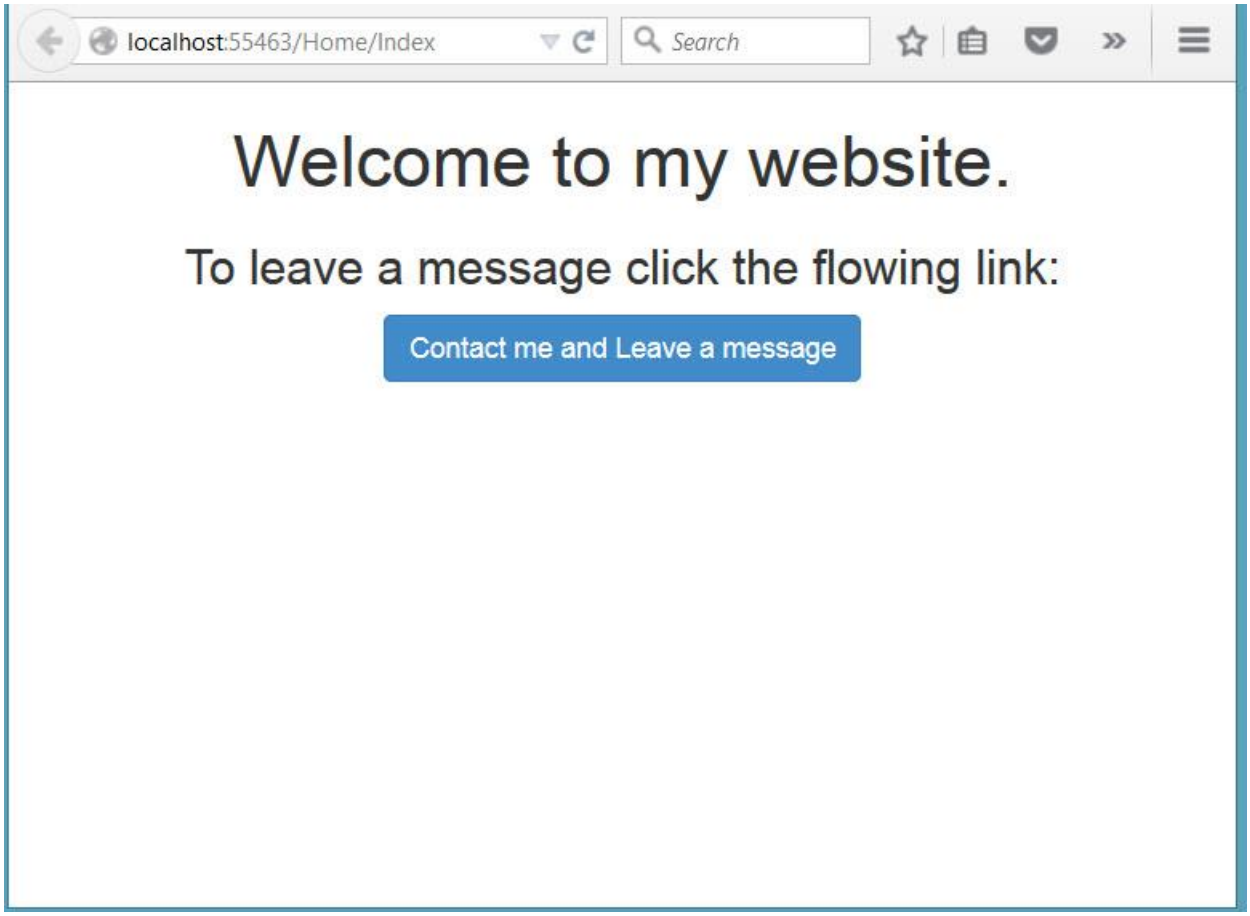
فایل Index.cshtml را باز کنید و دستورات آن را به صورت زیرتغییر دهید.

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />

    <title>Index</title>
</head>
<body>
    <div class="text-center">

        <h1>Welcome to my website.</h1>
        <h3>To leave a message click the flowing link:</h3>

        @Html.ActionLink("Contact me and Leave a message",
"Contact", null, new
        {
            @class = "btn btn-primary"
```

```
        })

    </div>
</body>
</html>
```

همانطور که مشاهده می کنید در تگ head توسط تگ link کتابخانه Bootstrap به پرورژه اضافه شده است. همچنین تگ div داخل تگ body یک کلاس text-center و همچنین به متد ActionLink کلاس btn btn-primary به صفت class نسبت داده شده است.

حال برنامه را اجرا کنید و نتیجه را مشاهده کنید:



شکل ۳ - ۲۴

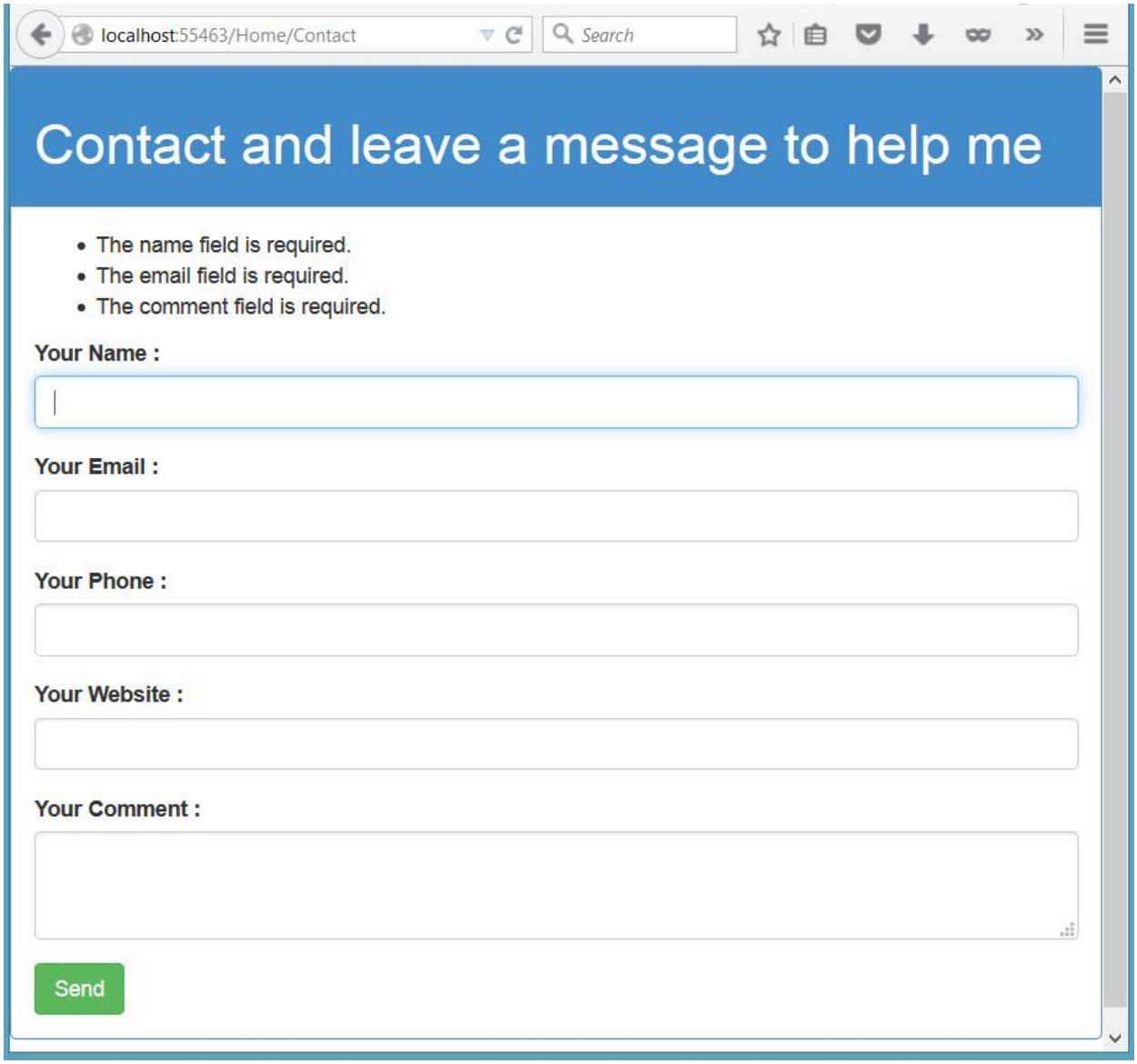
فرمت بندی فرم Contact

حال با استفاده از کلاس های استایل مربوط به ایجاد فرم ها ی اطلاعاتی در Bootstrap فرم Contact را به صورت زیر دو باره نویسی کنید.(به دستورات هایلایت شده دقت فرمائید)

```
@model mkianiir.mvc.ContactForm.Models.Contact
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>Contact me</title>
</head>
<body>
    <div>


        <div class="panel panel-primary">
            <div class="panel-heading">
                <h1>Contact and leave a message to help me</h1>
            </div>
            <div class="panel-body">
                @using (Html.BeginForm())
                {
                    @Html.ValidationSummary()
                    <div class="form-group">
                        <label>Your Name :</label>  @Html.TextBoxFor(x => x.Name,
new
                                                {
                                                    @class = "form-control"
                                                })
                    </div>
                    <div class="form-group">
                        <label>  Your Email : </label>@Html.TextBoxFor(x =>
x.Email, new
                                                {
                                                    @class = "form-control"
                                                })
                    </div>
                    <div class="form-group">
                        <label>
                            Your Phone :
                        </label>
                        @Html.TextBoxFor(x => x.Phone, new
                                                {
                                                    @class = "form-control"
                                                })
                    </div>
                    <div class="form-group">
                        <label>
                            Your Website :
                        </label>
                        @Html.TextBoxFor(x => x.Website, new
                                                {
                                                    @class = "form-control"
                                                })
                    </div>
                    <div class="form-group">
                        <label>
```

```
                    Your Comment :
                </label>
                @Html.TextAreaFor(x => x.Comment, new
                                        {
                                            @class = "form-control"
                                        })
            </div>
            <input type="submit" class="btn btn-success" value="Send" />


        }
    </div>
</div>

</div>
</body>
</html>
```

حال برنامه را اجرا کنید و به فرم Contact بروید.نتیجه مشابه شکل زیر خواهد بود.



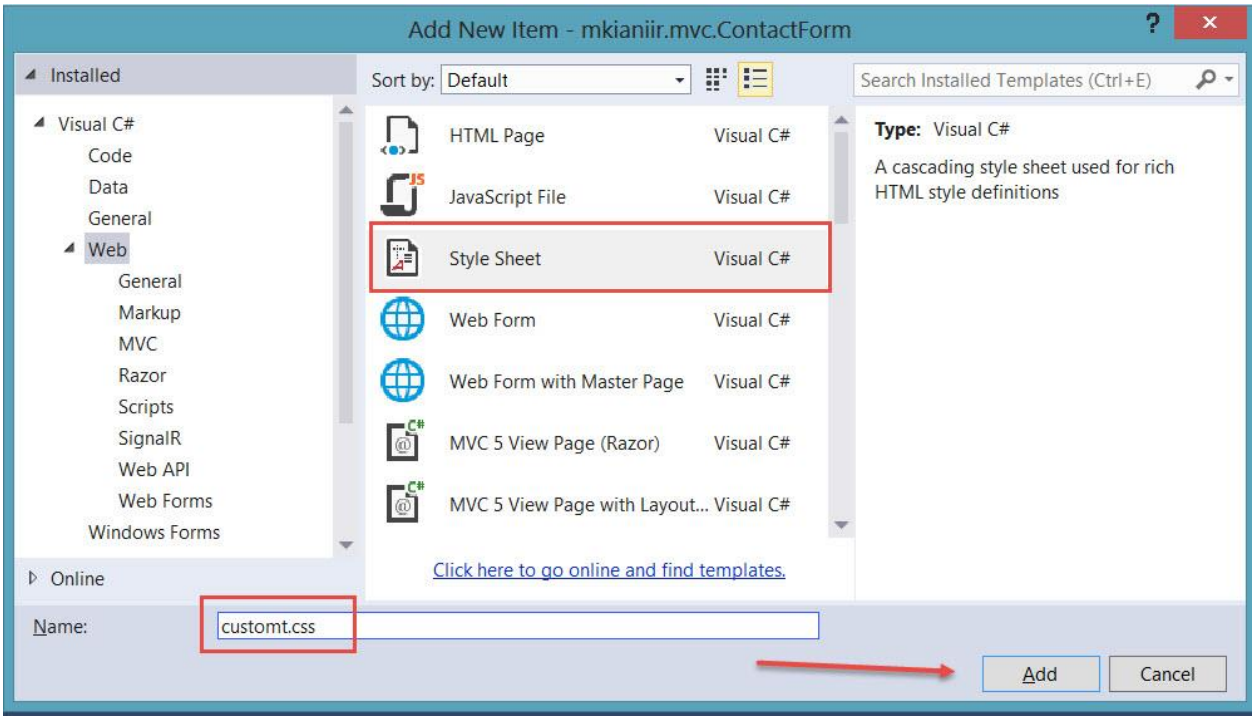
شکل ۳ - ۲۵

همانطور که در شکل فوق مشاهده می کنید ظاهر فرم فوق بسیار شکیل تر از نسخه قبلی آن می باشد.

تغییر در ظاهر خطا ها

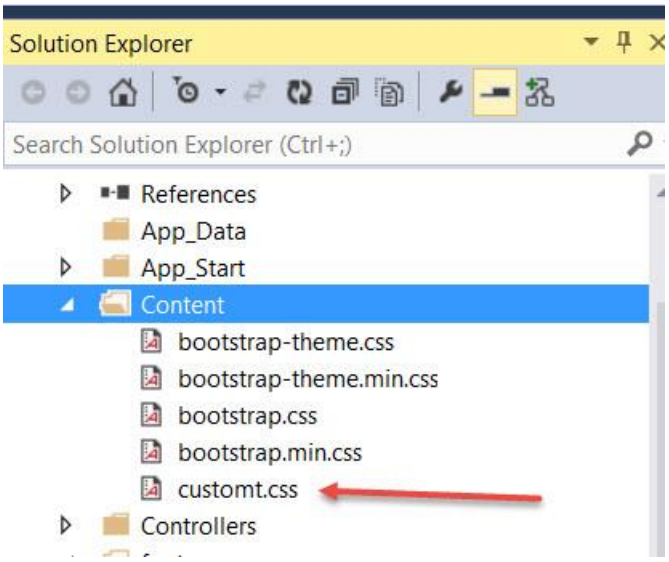
اگر خروجی برنامه را در زمانی که با خطا مواجه می شویم نگاه کنید متوجه خواهید شد که کلاس `validation-summary-errors` برای تگ `ul` که شامل `li` های خطا می باشد اضافه می شود. همچنین کنترل هایی که در آن ها خطای اعتبار سنجی رخ دهد کلاس `input-validation-error` برای آن کنترل ها اضافه خواهد شد. این کلاس ها به صورت خودکار توسط MVC ایجاد می شود. لذا شما می توانید با نوشتن کد های CSS برای این کلاس ها ظاهر مورد نظر کنترل های دارای خطای اعتبار سنجی را تغییر دهید.

برای این منظور بر روی پوشه Contents در Solution Explorer راست کلیک کنید و از منوی Add گزینه New Item را کلیک کنید.



شکل ۳ - ۲۶

در پنجره Add New Item گزینه Stylesheet را انتخاب کنید و نام آن را custom.css قرار دهید. با این کار یک فایل به نام custom.css به پوشه Content اضافه خواهد شد.



شکل ۳ - ۲۷

بر روی این فایل ایجاد شده دوبار کلیک کنید و دستورات زیر را برای آن بنویسید.

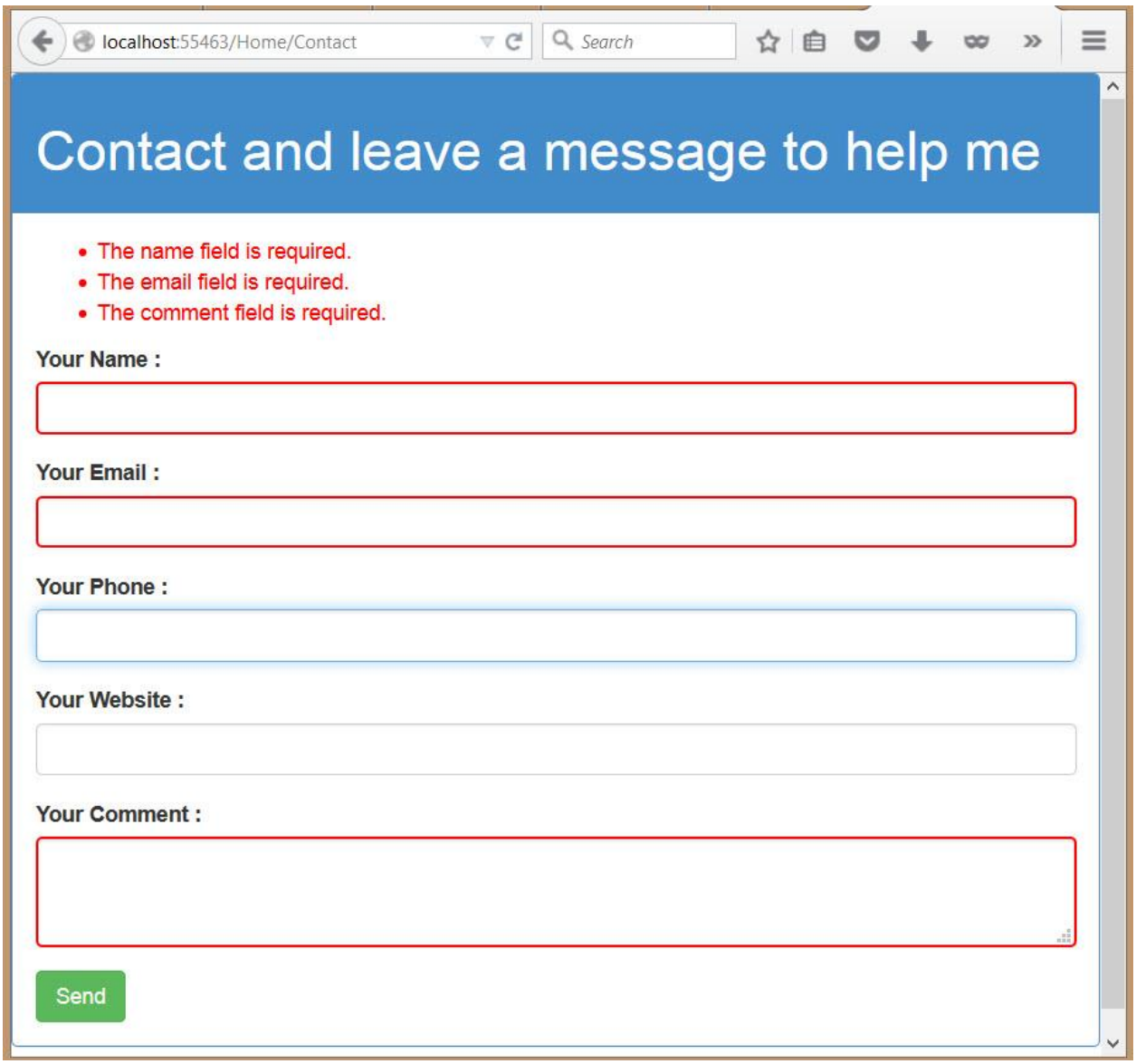
```
.validation-summary-errors li {
    color:#FF0000;
}
```

```
.input-validation-error {
    border:2px solid #FF0000;
}
```

حال دستور زیر را در فایل Contact.html در تگ head وارد نمائید.

```
<link href="~/Content/customt.css" rel="stylesheet" />
```

برنامه را اجرا کنیدو به فرم Contact بروید و بدون پر کردن فیلد ها دکمه Send را کلیک کنید.اگر همه موارد را به درستی انجام داده باشید نتیجه ای شبیه به شکل زیر خواهید داشت.



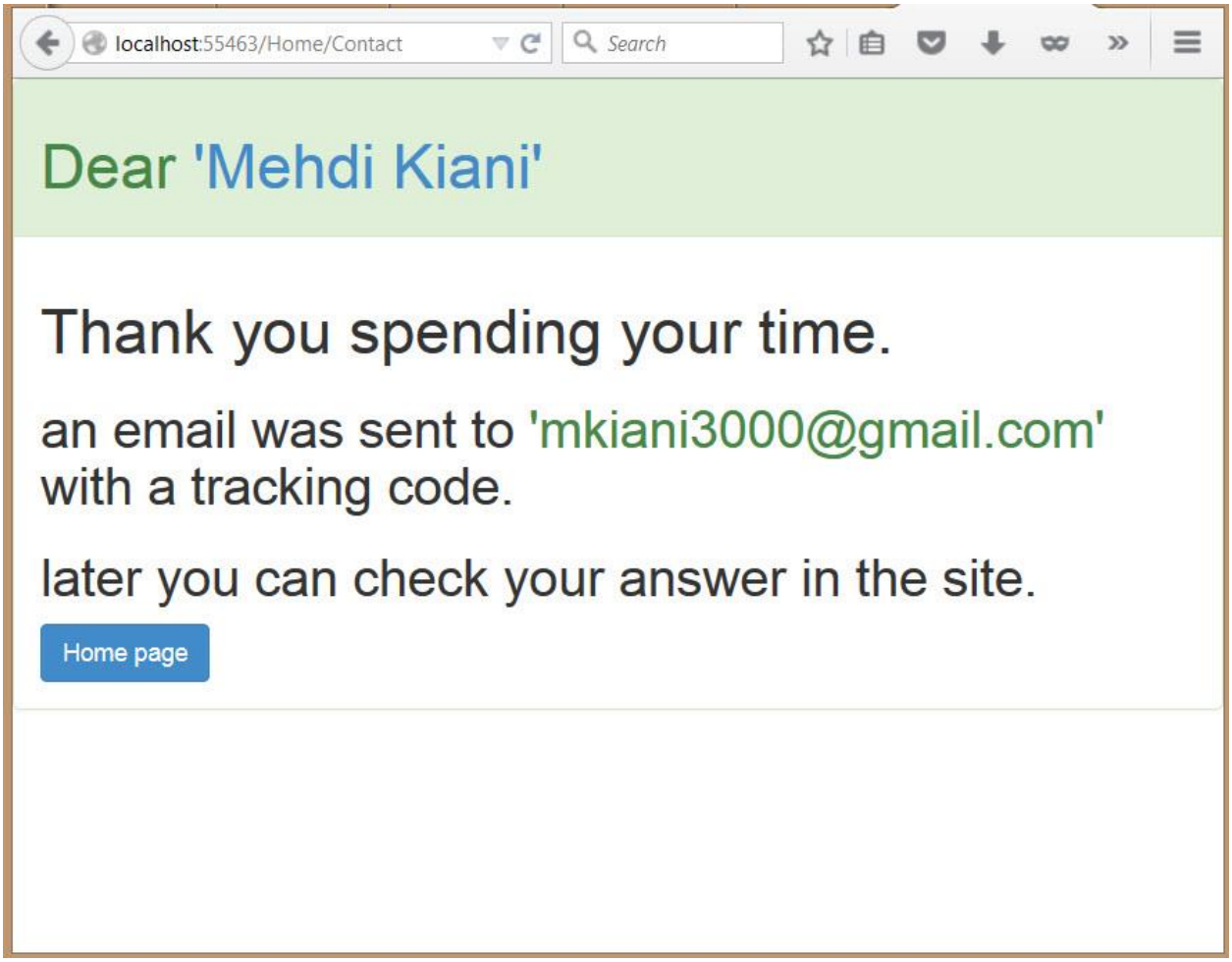
شکل ۳ - ۲۸

تغییر در ویو ContactDetail.cshtml

فایل ContactDetail.cshtml را باز کنید و دستورات آن را به صورت زیر تغییر دهید:

```
@model mkianiir.mvc.ContactForm.Models.Contact

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>ContactDetail</title>
</head>
<body>
    <div>
        <div class="panel panel-success">
            <div class="panel-heading">
                <h1>Dear  <span class="text-
primary">'@Model.Name'</span></h1>
            </div>
            <div class="panel-body">
                <h1>Thank you spending your time.</h1>
                <h2> an email was sent to <span class="text-
success">'@Model.Email'</span>  with a tracking code.</h2>
                <h2>later you can check your answer in the
site.</h2>
                @Html.ActionLink("Home page", "Index", null, new
       {
           @class = "btn btn-primary"
       })
            </div>
        </div>


    </div>
</body>
</html>
```

برنامه را مجددا اجرا کنید.فرم Contact را تکمیل و بر روی دکمه Send کلیک کنید.اگر همه چیز به درستی انجام شده باشد باید با نتیجه ای به شکل زیر مواجه شوید.



شکل ۳ - ۲۹

خلاصه

در این فصل سعی شد تا امکانات MVC به صورت ساده و خلاصه در قالب یک مثال بیان شود. در حال حاضر باید بتوانید ارتباط بین سه جزء اصلی MVC یعنی مدل ها، ویو ها و کنترلر ها را درک کرده باشید.ارسال اطلاعات از کنترلر به ویو و بلعکس را فراگرفته باشید. بتوانید فرم های ورود اطلاعات را اعتبار سنجی کنید و نیز ظاهر برنامه های خود را با استفاده از کتابخانه Bootstrap کاربر پسند تر کنید. انتظار می رود با تکرار مثال ها و دستوراتی که در این فصل بیان شد سعی کنید تا جزئیات عمیق تری از ارتباطات این اجزا بدست آورید.

فصل چهارم : URL Routing

مقدمه

در فصل های قبلی از سیستم روتینگ MVC در مثال ها بهره بردیم اما به جزئیات آن ها پرداخته نشد. در این فصل جزئیات بیشتری در را بطه با این سیستم مورد بررسی قرار خواهد گرفت. در پایان این فصل شما خواهید توانست روت های مختلف را تعریف کنید. از فضای نام ها برای تغییر مکان جستجوی کنترلر ها در روت ها بهره ببرید. همچنین برخی قواعد را در روت ها تنظیم نمائید.

کلاس RoutConfig.cs

زمانی که یک پروژه جدید از نوع MVC ایجاد می کنید در پوشه App_Start کلاسی به نام RoutConfig ایجاد می شود که دارای یک متد استاتیک به نام RegisterRoutes می باشد. در این متد که ساختار اولیه آن به صورت زیر می باشد روتینگ های اولیه تعریف شده است:

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

namespace mkianiir.mvc.ContactForm
{
    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 }
            );
        }
    }
}
```

متد RegisterRoutes آرگومانی از جنس RouteCollection دریافت می کند که توضیحات آن در ادامه آمده است.

کلاس RouteCollection و RouteCollectionExtensions

متد RegisterRoutes دارای آرگومانی از نوع RouteCollection می باشد. کلاس RouteCollection در فضای نام System.Web.Routing قرار دارد و دارای متد هایی مانند MapPageRoute می باشد که برای نگاشت آدرس ها در تکنولوژی ASP.Net WebForm به کار می رود.

اگر به تعریف این کلاس در دات نت توجه کنید متوجه خواهید شد که این کلاس از کلاسی به نام Collection ژنریکی که در فضای نام System.Collections.ObjectModel قرار دارد مشتق می شود. بنابر این کلاس RouteCollection یک مجموعه می باشد که برای نگهداری نگاشت های آدرس ها به کار می رود.

تیم MVC برای اینکه از همین کلاس برای نگاشت های فریم ورک MVC نیز استفاده کند به جای استفاده از یک کلاس مجزا برای این منظور اقدام به ایجاد متد های الحاقی[۲] برای این کلاس نموده است. این متد های الحاقی درون کلاسی به نام RouteCollectionExtensions در فضای نام System.Web.Mvc قرار دارد. این کلاس دارای سه متد MapRoute ، IgnoreRoute و GetVirtualRouteForArea می باشد.

متد MapRoute

متد MapRoute دارای شش حالت مختلف می باشد(6 override). در این ۶ حالت به طور کلی آرگومان های زیر با ترکیب های مختلفی قرار خواهند گرفت:

آرگومان name: این آرگومان نام روت(Route) را در مجموعه روت ها نگهداری می کند. به عنوان مثال نام Default را می توانید در متد MapRoute واقع در متد RegisterRoute در دستورات فوق مشاهده کنید.

[۲] Extenssion method

آرگومان url: این آرگومان الگوی آدرس برای نگاشت آدرس ها را نگهداری می کند.

آرگومان defaults: این آرگومان برای مقدار دهی های اولیه نگهدارنده های آرگومان url بکار می رود. هر url می تواند شامل چند نگهدارنده باشد.

آرگومان namespace : این آرگومان برای مشخص کردن روتینگ ها در فضای نام های مختلف بکار می رود. زمانی که در پروژه دو یا چند کنترلر هم نام داشته باشیم (این اتفاق عموما در زمانی که از Area ها در mvc استفاده می کنیم رخ خواهد داد) این آرگومان کارایی خواهد داشت. حتی ممکن است کنترلر های هم نامی در اسمبلی های مختلف وجود داشته باشند. در این حالت به کمک آرگومان namespace می توانیم MVC را در یافتن صحیح کنترلر ها یاری کنیم.

آرگومان Constrains : این آرگومان برای ایجاد قواعدی جهت مقادیری که می توانند به جای بخش های url قرار گیرند به کار می رود. به عنوان مثال فرض کنید بخشی به نام id در url قرار داشته باشد. با Constrains ها می توان قواعدی را برای مقدار هایی که id می تواند اختیار کند ایجاد کرد.به عنوان مثال می توان تعریف کرد که مقدار id بین ۵ تا ۱۰۰ باشد.

متد IgnoreRoute

اگر بخواهیم سیستم روتینگ از آدرسی یا الگوی آدرسی صرف نظر کند از متد IgnoreRoute استفاده خواهیم کرد.

به عنوان مثال دستور اول در متد RegisterRoutes باعث می شود تا سیستم روتینگ از آدرس هایی را که دارای پسوند axd هستند(این آدرس ها توسط برخی کامپوننت های ASP.Net نظیر ScriptResource تولید می شوند و توسط HttpHandler مخصوص خود پردازش می شوند) صرف نظر کند.

تعریف Route

هر روت در MVC عموما دارای یک الگوی آدرس دهی می باشد در سیستم روتینگ این الگو ها از چند بخش یا قطعه تشکیل شده است که از این پس به آن ها نگهدارنده می گوییم. (نام دامنه و همچنین رشته های کوئری (Query String) ها شامل این بخش ها نمی گردند.)

این بخش ها یا قطعات درون آکولاد قرار می گیرند و با کاراکتر اسلش(/) از یکدیگر جدا می شوند.

به عنوان مثال آدرس زیر دارای دو بخش controller و Action می باشد:

{controller}/{action}

بنابر این تمامی آدرس های زیر در سیستم روتینگ نگاشت خواهند شد:

http://localhost/Home/Index

http://localhost/Customer/Add

http://localhost/Customer/List

در آدرس های فوق Home و Customer به عنوان controller و Index، Addو List به عنوان Action درنظر گرفته می شوند.

اما آدرس زیر نگاشت نخواهد شد.زیرا دارای سه بخش مجزا می باشد و لی روت تعریف شده صرفا داری دو بخش است.

http://localhost/Customer/Edit/5

البته الگوی آدرس دهی در روتینگ می تواند دارای هیچ نگهدارنده ای نباشد و صرفا شامل یک رشته ثابت باشد. به عنوان مثال روت زیر را در نظر بگیرید:

```
routes.MapRoute(
                name: "Default",
                url: "plugins/rss",
                defaults: new { controller = "Plugin", action =
"Rss" }
            );
```

در روت بالا آدرس domain.com/plugins/rss اشاره می کند به اکشن متدی به نام rss در کنترلری به نام PluginController . مقادیر مربوط به کنترلر و اکشن در آرگومان defaults تعریف شده است که این آرگومان در ادامه توضیح داده خواهد شد.

تعریف مقادیر پیش فرض برای روت ها

همانطور که گفته شد هر روت داری یک الگو می باشد که معمولا از چند نگهدارنده تشکیل شده است. MVC زمانی که یک آدرس را مشاهده می کند در صورتی می تواند آن را پردازش کند که الگوی آدرس با الگوی روت نوشته شده یکسان باشد. به عنوان مثال اگر روت ما دارای دو بخش به صورت زیر باشد:

{controller}/{action}

آن گاه هر آدرسی که در آن دارای دو بخش فوق باشد توسط mvc تشخیش داده و پردازش خواهد شد.به عنوان مثال آدرس زیر را در نظر بگیرید:

http://localhost/Home/Index

چون این آدرس دارای دو بخش Home و Index می باشد بنابر این توسط mvc تشخیص داده خواهد شد. مقدار Home را به عنوان کنترلر و Index را به عنوان اکشن در نظر خواهد گرفت.لذا mvc به دنبال اکشن متدی به نام Index در کلاسی به نام HomeController خواهد گشت.

حال اگر بخواهیم برخی از بخش های یک روت یا همه آن ها را مقدار دهی اولیه نمائیم از آرگومان defaults در متد MapRoute به شکل زیر استفاده می کنیم.

مثال:

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

همانطور که مشاهده می کنید مقادیر Home و Index به عنوان مقادیر پیش فرض نگهدارنده های controller و action در نظر گرفته نشده است.

حال آدرس های زیر را در نظر بگیرید:

http://localhost/Home

http://localhost/

در آدرس اول مقدار action وارد نشده است بنابر این مقدار اولیه آن یعنی Index و در آدرس دوم هیچ یک از نگهدارنده های controller و action تعریف نشده اند بنابر این مقدار Home برای کنترلر و مقدار Index برای action در نظر گرفته می شود.

در تعریف مقادیر اولیه از مفهومی به نام انواع بی نام(Anonymous Types) استفاده شده است. در واقع توسط دستور

```
new
          {
              controller="Home",
              action="Index"
          }
```

یک نوع بدون نام با دو خاصیت controller و action تعریف شده است. انواع بی نام از مجموعه مفاهیم زبان سی شارپ می باشد.

استفاده از مقادیر ثابت در روت ها

یکی از امکاناتی که می توان در روت ها به کار برد استفاده از مقادیر ثابت در الگوی آن ها می باشد. در بخش قبلی نمونه روتی تعریف کردیم که شامل هیچ نگهدارنده ای نبود. در این بخش مشاهده خواهید کرد که می تواند مقادیر ثابت را در کنار نگهدارنده ها برای الگو های روت در نظر گرفت.

الگوی زیر را در نظر بگیرید:

admin/{controller}/{action}

کلمه admin به عنوان یک مقدار ثابت و {controller} و{action} به عنوان مقادیر متغیر در الگوی فوق تعریف شده اند. بنابراین هر آدرس که بخواهد طبق الگوی فوق شناخته شود باید دارای کلمه admin قبل از قسمت controller باشد.به عنوان مثال آدرس های زیر طبق الگوی فوق شناخته خواهد شد:

http://localhost/admin/Home/Index

تعریف نگهدارنده اختیاری در الگوی آدرس

اگر بخواهیم در الگوی آدرس دهی بخشی داشته باشیم که کاربر بخواهد به صورت دلخواه آن را مقدار دهی کند یا خیر می بایستی آن را به صورت پارامتر اختیاری تعریف کنیم. برای این منظور از دستور UrlParameter.Optional استفاده می کنیم.

مثال

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

همانطور که مشاهده می کنید در الگوی فوق در بخش مقادیر اولیه مقدار بخش id با دستور UrlParameter.Optional تعریف شده است. بنابراین علاوه بر تمامی آدرس های سه بخشی(شامل

controller و action و id) ، آدرس های دو بخشی نیز می توانند توسط روت بالا تشخیص داده شوند. به عنوان مثال بر اساس الگوی روت بالا تمامی آدرس های زیر توسط mvc تشخیص داده خواهند شد:

http://localhost/Home/Index

http://localhost/Customer/Show/all

http://localhost/Customer/Edit/5

در تمامی آدرس های فوق Home و Customer نقش controller را دارند. Index و Show و Edit نقش action و all و ۵ نقش id را خواهند داشت.

توجه داشته باشید که چون بخش id به صورت اختیاری تعریف شده است بنابر این آدرس اول که دارای id نیز نمی باشد توسط روت بالا تشخیص داده خواهند شد.

الگوهای روت با طول متغیر

همانطور که پیشتر گفتیم mvc آدرس ها را بر اساس الگوی آن بررسی می کند. الگوی روتی که دارای دو نگهدارنده می باشد می تواند تمامی آدرس های دو بخشی را تشخیص دهد.یک الگوی سه بخشی نمی تواند توسط الگویی که دارای دو بخش است تشخیص داده شود مگر اینکه بخش سوم در روت به صورت اختیاری(optional) تعریف شده باشد.

حال با استفاده از کاراکتر * قبل از نام بخش در الگو می توانیم روتی با طول متغیر تعریف کنیم. مثال:

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

با توجه به الگوی فوق آدرس های زیر تشخیص داده می شوند:

http://localhost/Home

http://localhost/Home/Index

http://localhost/Home/Index/5

http://localhost/Home/Index/5/a/b

http://localhost/Home/Index/5/a/b/c/d/e/f

در الگوی فوق سه نگهدارنده به نام های controller ، action و id تعریف شده است و چون id به صورت اختیاری تعریف شده است بنابر این تمامی آدرس های دوبخشی و سه بخشی می توانند توسط این الگو تشخیص داده شوند. علاوه بر سه بخش مذکور بخش دیگری به نام others تعریف شده است. توجه داشته باشید که قبل از کلمه others کاراکتر * قرار گرفته است. این بدین معناست که در آدرس هایی که دارای بیش از ۳ بخش باشند(بر اساس الگوی فوق) بدین صورت توسط mvc تفسیر می شوند:

بخش اول به عنوان controller ، بخش دوم به عنوان action ، بخش سوم به عنوان id و سایر بخش ها به عنوان others

به عنوان مثال آدرس های زیر را در نظیر بگیرید:

http://localhost/Home/Index/5

http://localhost/Home/Index/5/data1

http://localhost/Home/Index/5/data1/data2

http://localhost/Home/Index/5/data1/data2/data3/.../datan

در آدرس اول، Home به عنوان controller و Index به عنوان action و ۵ به عنوان id در نظر گرفته می شود.

در آدرس دوم Home به عنوان controller و Index به عنوان action و ۵ به عنوان id و data1 به عنوان others در نظر گرفته می شود.

در آدرس سوم Home به عنوان controller و Index به عنوان action و ۵ به عنوان id و data1/data2 به عنوان others در نظر گرفته می شود.

و نهایتا در آدرس آخر Home به عنوان controller و Index به عنوان action و ۵ به عنوان id و data1/data2/data3/…/datan به عنوان others در نظر گرفته می شود.

همانطور که مشاهده می کنید تعداد مقادیری که می تواند برای بخش others بکار رود متغیر می باشد. توجه داشته باشید که مقادیر others تحت یک مقدار رشته ای نشان داده می شود و این شما هستید که باید آن ها را در برنامه خود تفکیک کنید.

برای مثال دستورات متد Index را به صورت زیر تغییر دهید:

```
public ActionResult Index()
        {
            ViewBag.Controller = RouteData.Values["controller"];
            ViewBag.Action = RouteData.Values["action"];
```

```
            ViewBag.Id = RouteData.Values["id"] ?? "<No value>";
            ViewBag.Others = RouteData.Values["others"] ?? "No
value";
            return View();
        }
```

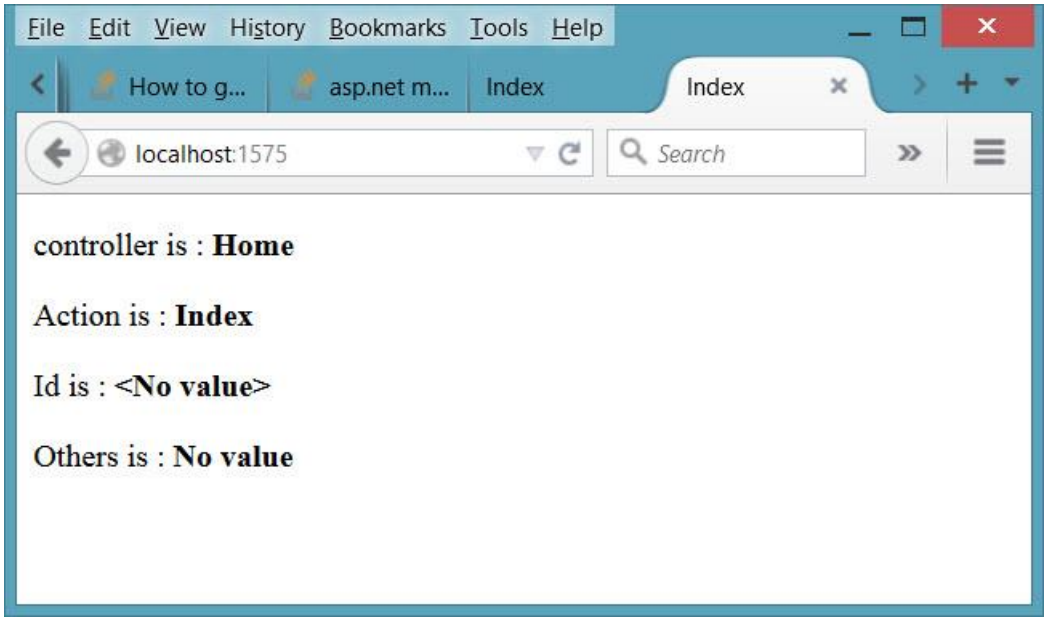
دستورات فوق به قدری ساده هستند که نیاز به هیچ توضیح اضافه ای ندارند.

حال یک ویو برای Index ایجاد کنید و کد های آن را مطابق زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <p>controller is :<b> @ViewBag.Controller</b></p>
    <p>Action is :<b> @ViewBag.Action</b></p>
    <p>Id is :<b> @ViewBag.Id</b></p>
    <p>Others is : <b>@ViewBag.Others</b></p>
</body>
</html>
```

حال برنامه را اجرا کنید و آن را با آدرس های زیر تست کنید.(شما می بایستی پرت(Port) مربوط به خود را استفاده نمائید)

http://localhost:1575



شکل ۴ - ۱

همانطور که مشاهده می کنید در آدرس فوق مقادیر پیش فرض controller و action که به ترتیب Home و Index می باشند در نظر گرفته شده اند. همانطور که مشاهده می کنید مقادیر id و others وارد نشده اند. حال مجددا برنامه را با آدرس زیر تست کنید:

http://localhost:1575/home/index/5/data1/data2/data3

شکل ۴ - ۲

همانطور که مشاهده می کنید مقادیر ۵ برای id و data1/data2/data3 برای بخش others در نظر گرفته شده اند.

نکته: ViewBag یک فیلد داینامیک می باشد که می توان از آن برای ارسال مقادیر به عنوات مدل به view استفاده کرد. در رابطه با ViewBag در فصل بعدی مفصل توضیح داده خواهد شد.

استفاده از فضای نام برای دسترسی به کنترلرها

همانطور که در بخش های قبلی بیان شد زمانی که یک کنترلر به پروژه اضافه می کنیم در پایان نام کلاس آن از کلمه کنترلر استفاده می شود.زمانی که mvc به آدرسی شبیه به آدرس زیر برخورد می کند:

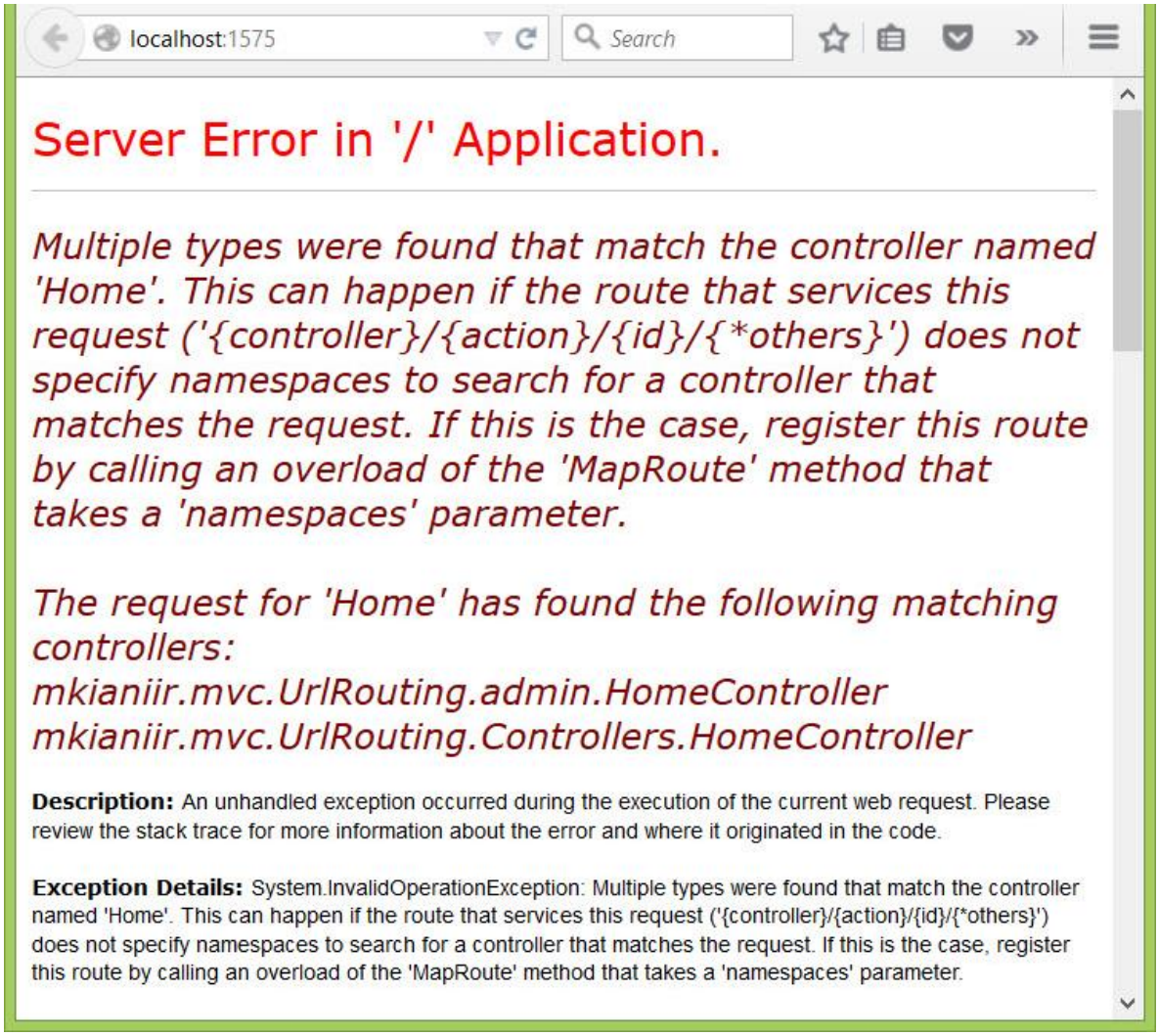
http://lcoalhost/Home/Index

تشخیص می دهد که ما به دنبال کنترلری به نام Home هستیم. بنابر این mvc به دنبال کلاسی به نام HomeController خواهد گشت.کلمه Controller را به صورت خودکار به نام کنترلری که از آدرس واکشی می کند قرار می دهد.

حال اگر در پروژه دو کلاس به نام HomeController (در فضای نام های مختلف)وجود داشته باشد mvc از کجا باید تشخیص دهید که کدام کلاس کنترلر مد نظرکاربر می باشد؟

این اتفاق در پروژه های بزرگ ممکن است زیاد رخ دهد.یا حتی ممکن است شما از یک دات نت اسمبلی دیگری در پروژه خود استفاده کنید که مثلا دارای کنترلری به نام Home می باشد. اگر شما نیز یک کنترلر به نام Home داشته باشید mvc به کدامیک باید مراجعه کند؟ این اتفاق می تواند زمانی که از Area ها در mvc استفاده می کنیم نیز رخ دهد. مثلا یک Area به نام admin تعریف می کنیم و در آن یک HomeController ایجاد می کنیم.حال زمانی که نیاز به کنترلر Home می باشد mvc به کدام یک باید رجوع کند؟

برای اینکه واکنش mvc را در این مواقع ببینید. یک پوشه به نام admin در پروژه ایجاد کنید و سپس در آن یک کنترلر به نام Home ایجاد کنید. پس از آن برنامه را اجرا کنید. نتیجه باید شبیه به عکس زیر باشد.



شکل ۴ - ۳

همانطور که مشاهده می کنید mvc دو کلاس HomeController یکی در فضای نام mkianiir.mvc.UrlRouting.admin.HomeController و دیگری در فضای نام mkianiir.mvc.UrlRouting.Controllers پیدا کرده است. بنابر این قادر به تشخیص نیست که از کدام کنترلر باید استفاده کند.

در این حالت استفاده از فضای نام ها در تعریف الگوهای آدرس دهی می تواند به mvc در تشخیص کنترلر صحیح کمک کند.

حال اگر الگوی آدرس دهی را به صورت زیر تغییر دهیم:

```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new
                {
```

```
                controller = "Home",
                action = "Index",
                id = UrlParameter.Optional
            },
            namespaces:
            new[]
            {
                "mkianiir.mvc.UrlRouting.Controllers"
            }
        );
```

آنگاه mvc صرفا به دنبال کلاس HomeController در فضای نام mkianiir.mvc.UrlRouting.Controllers خواهد گشت و برنامه بدون خطا اجرا خواهد شد.

ایجاد محدودیت برای مقادیر نگهدارنده ها

یکی دیگر از امکاناتی که سیستم روتینگ به شما می دهد این است که بتوانید با ایجاد محدودیت کنترل دقیق تری برای مقادیری که در آدرس ها به جای نگهدارند های الگوی آدرس دهی به کار می روند اعمال نمائید. به عنوان مثال الگوی زیر را در نظر بگیرید:

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

        );
```

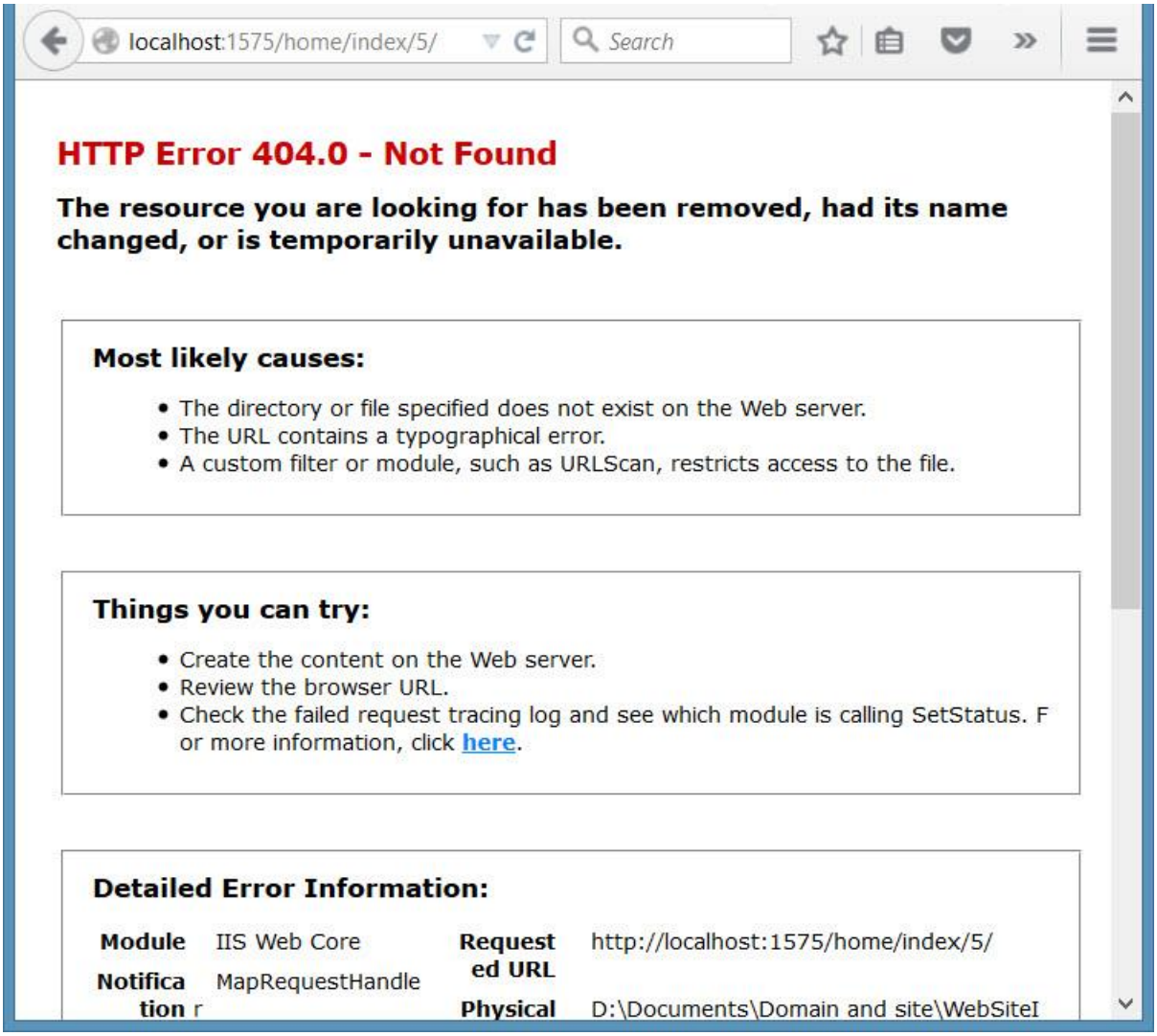
برای این منظور کلاس هایی در فضای نام System.Web.Mvc.Routing.Constraints تعبیه شده است که می توانید از آن ها در اعمال قواعد برای روت ها استفاده نمائید.

به عنوان مثال فرض کنید بخواهیم مقدار id صرفا عددی بین ۱۰ تا ۱۰۰ باشد. در ابتدا مقدار پیش فرض id را به مقدار ۱۰ تغییر می دهیم.برای این منظور از کلاسی به نام RangeRoutConstraint به شکل زیر استفاده می کنیم.

```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}/{*others}",
                defaults: new
                {
                    controller = "Home",
                    action = "Index",
                    id = “10”
                },
                namespaces: new[]
                {
                    "mkianiir.mvc.UrlRouting.Controllers"
                },
                constraints: new
                {
                    id=new RangeRouteConstraint(10,100)
                }

            );
```

حال برنامه را اجرا کنید و با آدرس زیر تست کنید:

http://localhost:1575/home/index/5/



شکل ۴ - ۴

همانطور که مشاهده می کنید اگر چه آدرس طبق الگوی روت می باشد اما برنامه با خطا مواجه شده است.علت آن ، مقداری است که برای id در آدرس تنظیم شده است. در الگوی آدرس مقادیر مجاز برای id مقادیر بین ۱۰ تا ۱۰۰ تنظیم شده است بنابر این هر مقداری خارج از این محدوده منجر به ایجاد خطا خواهد شد.

نکته: همانطور که گفته شد کلاس هایی که برای اعمال قواعد در mvc به کار می روند در فضای نام System.Web.Mvc.Routing.Constraints تعریف شده اند. لذا در فایل RouteConfig می بایستی این فضای نام را به لیست فضای نام های کلاس اضافه کنید.

کلاس های دیگری نظیر AlphaRouteConstraint ، BoolRouteConstraint ، DateTimeRouteConstraint، GuidRouteConstraint و ... نیز در فضای نام مذکور وجود دارد که نحوه استفاده از آن ها را به خواننده واگذار می کنم.

فراخوانی متد RegisterRoutes

فایل global.asax را از پنجره Solution Explorer باز کنید.

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

namespace mkianiir.mvc.ContactForm
{
    public class MvcApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            RouteConfig.RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

همانطورکه مشاهده می کنید درون این فایل کلاسی به نام MvcApplication قرار دارد که از کلاس HttpApplication مشتق می شود. کلاس HttpApplication دارای رویداد های مختلفی است. رویداد Application_Start یکی از این رویداد ها می باشد. این رویداد زمانی که اولین بار پروژه Mvc شما اجرا می گردد فراخوانی می شود. همانطور که مشاهده می کنید در خط دوم ، متد RegisterRoutes از کلاس RouteConfig فراخوانی شده است. این دستور باعث می شود که روت های نوشته شده توسط شما در مجموعه روت ها تعریف گردد تا MVC بتواند نگاشت آدرس ها را به درستی انجام دهد.

فایل های فیزیکی اولویت بالاتری نسبت به روت ها دارند

روت زیر را درنظر بگیرید:

{controller}/{action}/{id}

همانطور که گفته شد روت بالا می تواند آدرس زیر را پردازش کند:

http://lcoalhost/Customer/Edit/5

در این حالت کلمه Customer به عنوان controller و کلمه Edit به عنوان اکشن و ۵ به عنوان id مورد پردازش قرار خواهند گرفت.

حال فرض کنید در پروژه پوشه ای به نام images درون آن پوشه ای به نام icons و نهایتا درون پوشه icons یک فایل به نام smile.png دارید.

حال آدرس زیر را در نظر بگیرید:

http://lcoalhost/images/icons/smile.png

اگر بر طبق مباحث قبلی بخواهیم این آدرس را بررسی کنیم می بایستی images را به عنوان controller و icons را به عنوان Action و smile.png را به عنوان id در نظر بگیریم!!!

خوب چه اتفاقی می افتد؟ چون کنترلری به نام images وجود ندارد بنابر این آدرس بالا توسط سیستم روتینگ قابل ردیابی نبوده و در نتیجه با پیغام خطا مواجه خواهید شد. به همین علت زمانی که mvc به آدرسی که شبیه به یک فایل است مواجه می شود ابتدا بررسی می کند که آیا آن آدرس به صورت فیزیکی درسرور وجود دارد یا خیر؟ اگر آن آدرس به صورت فیزیکی در سرور وجود داشته باشد از پردازش آن خودداری می کند و اجازه می دهد تا موتور Asp.Net به صورت معمول و توسط هندلر مخصوص به آن فایل پردازش فایل مورد نظر را به عهده بگیرد.

این رفتار پیش فرض mvc می باشد. شما می توانید با مقدار دهی خاصیت RouteExistingFIles به مقدار true این رفتار را تغییر دهید .

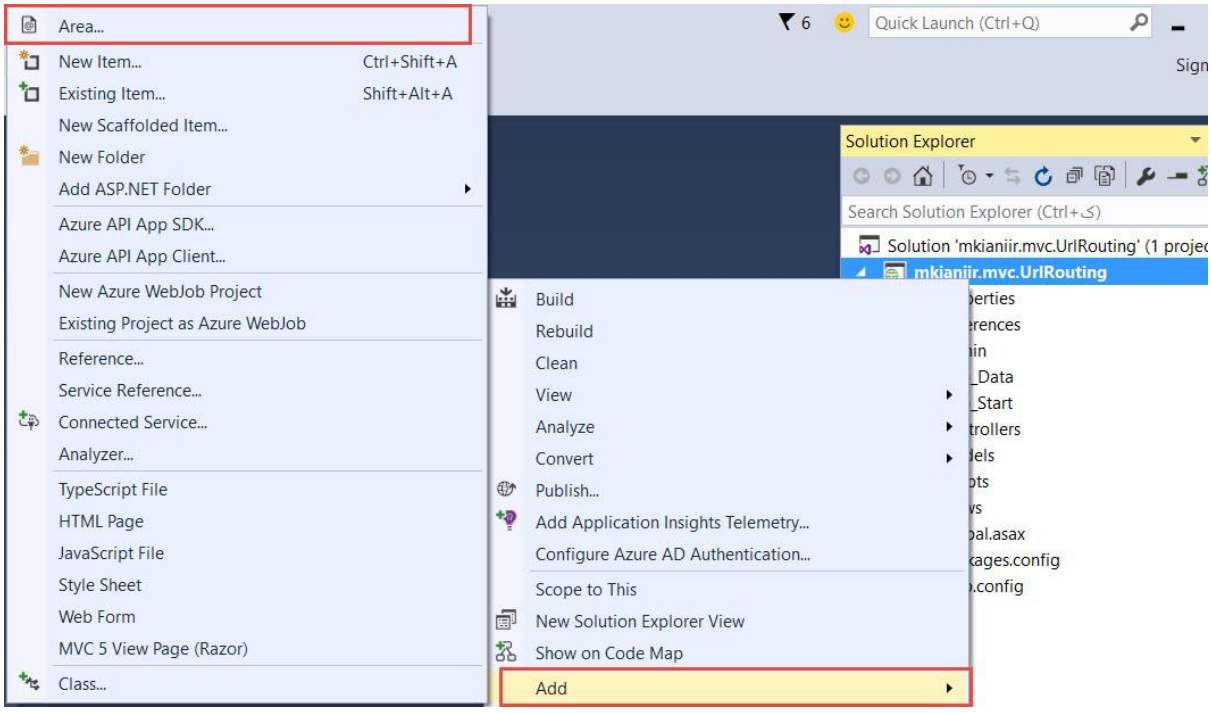
نکته مهم این است که سیستم روتینگ mvc فقط فایل هایی که وجود فیزیکی در سرور داشته باشند را نادیده می گیرد و در مورد فایل هایی که وجود فیزیکی در سرور ندارند طبق روال معمول خود عمل خواهد کرد.

در برخی از مواقع فایل هایی وجود دارند که وجود فیزیکی در سرور ندارند و هندلر خاصی برای پردازش آن ها مورد استفاده قرار میگیرد. فایل های ScriptResource.axd و WebResource.axd که در بسیاری از کامپوننت های ASP.Net مورد استفاده قرار میگیرند از این دسته فایل ها هستند. به همین جهت نیاز است که این فایل ها را توسط متد IgnoreRoute از سیستم پردازشی روتینگ مستثنی کنیم. کاری که در خط اول متد RegisterRoutes انجام شده است.

محدوده بندی پروژه(Areas)

یکی دیگر از امکاناتی که MVC در اختیار شما قرار می دهد ایجاد Area در پروژه می باشد. Area ها به شما کمک می کند تا پروژه خود را محدوده بندی کنید. به عنوان مثال فرض کنید می خواهید یک محدوده برای بخش مدیریت پروژه در نظر بگیرید. بدین منظور می توانید یک Area مثلا با نام administrator تعریف نمائید. هر Area در mvc کاملا مستقل از سایر Area ها کار می کند. هر Area درون خود دارای ساختار کاملی از MVC می باشد. یعنی درون خود تمامی پوشه هایی نظیر Controllers ، Models، Views و .. را دارد.

برای درک بهتر موضوع اجازه دهید که یک Area به نام administrator به پروژه اضافه کنیم. برای این منظور بر روی نام پروژه کلیک راست کرده و از منوی Add گزینه Area را کلیک نمائید.

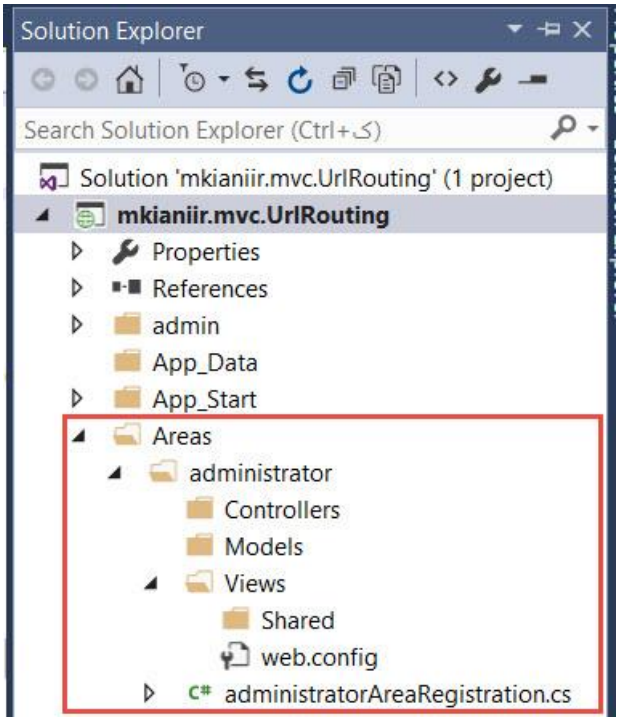


شکل ۴ - ۵

پنجره Add Area باز می شود. نام administrator را در آن وارد نمائید و سپس دکمه Add را کلیک کنید.

شکل ۴ - ۶

حال اگر به پنجره Solution Explorer دقت کنید مشاهده خواهید کرد که یک پوشه به نام Areas ایجاد شده است. درون پوشه مذکور یک پوشه دیگر به نام administrator (هم نام با Area تعریف شده در بخش قبلی) ایجاد شده است و درون پوشه administrator پوشه های Controllers و .. تعریف شده اند.



شکل ۴ - ۷

همانطور که گفته شد هر Area به صورت کاملا مستقل عمل می کند و خود، دارای پوشه های MVC می باشد. اگر با دقت بیشتری به اجزای تشکیل شده دقت کنید به کلاسی به نام

برخواهید خورد. اگر این کلاس را باز کنید دستورات شبیه به administratorAreaRegistration دستورات زیر را مشاهده خواهید کرد:

```
public class administratorAreaRegistration : AreaRegistration
{
    public override string AreaName
    {
        get
        {
            return "administrator";
        }
    }

    public override void RegisterArea(AreaRegistrationContext
context)
    {
        context.MapRoute(
            "administrator_default",
            "administrator/{controller}/{action}/{id}",
            new { action = "Index", id = UrlParameter.Optional }
        );
    }
}
```

همانطور که مشاهده می کنید کلاس مذکور از کلاس AreaRegistration مشتق شده است. درون این کلاس خاصیت AreaName و متد RegisterArea به صورت override در آمده است . خاصیت AreaName نام Area را مشخص می کند و و درون متد RegisterArea یک روت تعریف شده است. اگر به الگوی روت تعریف شده دقت کنید مشاهده می کنید که کلمه administrator در ابتدای الگو تعریف شده است. این بدان معنی است که آدرس هایی که پس از نام دامنه با کلمه administrator تعریف می شوند مربوط به Area جدید می باشد. حال یک کنترلر به نام Home در پوشه Controllers مربوط به administrator ایجاد کنید و دستورات آن را مطابق زیر تغییر دهید:

```
public class HomeController : Controller
{
    // GET: administrator/Home
    public string Index()
    {
        return "this is home index in administrator are";
    }
}
```

حال اگر برنامه را اجرا کنید با خطا مواجه خواهید شد چرا؟ بله درست حدس زدید. چون در پروژه بیش از یک کنترلر به نام Home وجود دارد و MVC نمی داند که منظور کدامیک می باشد. اگر خاطرتان باشد در بخش های قبلی یاد اور شدیم که یکی از کاربردهای فضای نام ها در تعریف روت ها هنگام کار با Area ها می باشد. برای حل مشکل کافیست روت تعریف شده در متد RegisterArea در کلاس administratorAreaRegistration را به صورت زیر تغییر دهید:

```
public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "administrator_default",
                "administrator/{controller}/{action}/{id}",
                new {controller="Home", action = "Index", id =
UrlParameter.Optional },
                new[] {
"mkianiir.mvc.UrlRouting.Areas.administrator.Controllers" }
            );
        }
```

همانطور که مشاهده می کنید نام Home به عنوان مقدار پیش فرض برای نگهدارنده کنترلر تعریف شده است. همچنین فضای نامی که MVC باید به دنبال کنترلرهای administrator بگردد را معرفی کرده ایم(بخش های هایلایت شده در دستورات فوق)

حال برنامه را اجرا و عبارت administrator/ را در پایان آدرس درج کنید. اگر همه چیز به درستی انجام شده باشد می بایتسی خروجی برنامه شبیه به عکس زیر باشد.

شکل ۴ - ۸

خلاصه

در این فصل مفاهیم عمیق تری از روتینگ ها در MVC مورد بحث قرار گرفتند. برای درک بهتر این مفاهیم لازم است تا دستورات و مثال های فصل را چندین بار مرور نمائید و سعی کنید تا با روند اجرایی هر یک را کاملا آشنا شوید.

فصل پنجم: اکشن متدها و فیلتر ها

مقدمه

در فصل های قبلی از اکشن متد های مختلف به صورت ساده استفاده شد. در این فصل جزئیات بیشتری از اکشن متد ها مورد بررسی قرار می گیرند. انتظار می رود در پایان فصل بتوانید انواع خروجی های اکشن متد ها را شناخته و کاربرد هر یک را بدانید همچنین با نحوه ارسال پارامتر به اکشن متد ها و رفتار MVC در خصوص آن ها آشنا شوید.در بخش پایانی این فصل فیلتر ها معرفی خواهند شد. فیلتر های اولیه و کاربرد آن ها بیان و نهایتا نحوه نوشتن یک فیلتر سفارشی مورد بحث و گفتگو قرار خواهد گرفت.

یادآوری

همانطور که پیش تر گفته شد مکانیزم Mvc بدین صورت است که زمانی که یک آدرس توسط کاربر در مرورگر تایپ می شود mvc بر اساس آدرس مذکور که همان درخواست کاربر نیز محسوب می شود به یک کلاسی که به آن controller می گویند رجوع می کند. کلاس controller درون خود دارای متد هایی است که به آن ها اکشن متد گفته می شود. کلاس کنترلر بر اساس مقادیری که در آدرس وجود دارد(اکشن و سایر پارامتر ها) پردازش های مربوطه را انجام داده و در نهایت از بین ویو های موجود در پروژه یک ویو را انتخاب و با ارسال داده های مورد نیاز(model) به آن ویو باعث ایجاد خروجی نهایی به صورت کدهای html می شود.

اکشن متد ها

اگر از تکنولوژی Asp.Net Web Form استفاده کرده باشید حتما می دانید که اساس کار این تکنولوژی بر روی رویداد ها می باشد. به عنوان مثال فرمی که برای ورود به برنامه طراحی شده است را در نظر بگیرید.در ساده ترین حالت این فرم دارای دو کنترل TextBox جهت ورود نام کاربری و کلمه عبور و یک کنترل Button جهت دکمه ورود دارد. ساختار Web Form به این صورت است که زمانی که بر روی دکمه ورود کلیک می

کنید یک درخواست به سرور ارسال می شود. (عمل Postback) در سرور رویدادی به نام Click که مربوط به کنترل Button است فراخوانی شده و دستورات آن (منظور دستورات متد هندلر رویدا می باشد)اجرا می شود. سپس به فرم وب دستور داده می شود که کد html معادل خودش را تولید کند. فرم وب نیز به همین ترتیب از کنترل های درونی خود می خواهد که کد html معادل خودشان را تولید کنند. به این عملیات رندرینگ می گویند. پس از اینکه همه کنترل ها رندر شدند خروجی html به مرورگر کاربر فرستاده و نمایش داده می شود.

پس بنابراین رویدد ها نقش کلیدی در تکنولوژی Asp.Net Web From دارند. شبیه به این اتفاق در برنامه نویسی Windows Forms Application نیز رخ می دهد. به این دسته از تکنولوژی های برنامه نویسی، رویداد گرا یا بر پایه رویداد (Event Base) می گویند. این مدل برنامه نویسی اگر چه در بسیاری از سناریو ها می تواند کار توسعه یک برنامه را سریعتر جلو ببرد اما دارای معایبی نیز هست که در این بخش مجال بحث بر روی آن نیست. صرفا جهت ارائه نمونه می توان به ایجاد وابستگی بین کدهای منطق برنامه و کدهای واسط کاربری(UI) اشاره کرد که این موضوع می تواند مشکلاتی از قبیل عدم قابلیت تست پذیری مناسب برنامه را به دنبال داشته باشد.

در تکنولوژی Mvc استفاده از رویداد موضوعیت ندارد. برخلاف رویداد ها که کلید اصلی در برنامه نویسی Web Form یا Windows Form می باشند در mvc آدرس(Url)ها این نقش اصلی را بازی می کنند. ساز و کار mvc بر اساس آدرس هایی است که کاربر وارد می کند. همانطور که گفته شد مسئول رسیدگی به درخواست های کاربران در MVC کنترلر ها می باشند. به عنوان مثال در آدرس زیر:

http://localhost/Home/Index

کلمه Home بیانگر کنترلری به نام Home می باشد که توسط کلاس HomeController.cs مشخص می شود. همچنین کلمه Index بیانگر یک اکشن متد درون کنترلر می باشد.mvc زمانی که با آدرس فوق مواجه می شود می داند که باید به سراغ کنترلری به نام Home رفته و دستورات درون متد Index را اجرا نماید.

نکته: البته در mvc هم گاهی مجبور خواهیم بود که از دکمه ها(Button) استفاده کنیم. اما مکانیزم آن چه که در mvc رخ می دهد با آن چه که در Web Form راخ خواهد داد متفاوت است. که نمونه آن را قبلا در فرم Contact مشاهده نمودید.

انواع خروجی در اکشن متدها

خروجی اغلب اکشن متد ها نمونه ای از یک کلاس مشتق شده از کلاس ActionResult خواهد بود. کلاس ActionResult یک کلاس انتزاعی در فضای نام System.Web.Mvc می باشد که به عنوان کلاس پایه برای کلاس هایی است که به عنوان خروجی یک اکشن متد به کار می روند. این که از چه کلاسی برای خروجی یک اکشن متد استفاده می شود بستگی به عملکرد آن متد دارد. در ادامه برخی از کلاس های مشتق شده ازکلاس ActionResult و کاربرد آن ها شرح داده شده اند.

کلاس ViewResult

زمانی که خروجی اکشن متد یک ویو باشد از این کلاس استفاده می کنیم. متدی به نام View() در کلاس کنترلر تعریف شده است. متد View بر اساس نام اکشن متد خود و یا یا توجه به آرگومان های ورودی آن یک ویو بر می گرداند. همانطور که قبلا اشاره شد ویو ها در پوشه ای هم نام با کنترلر خود و درون پوشه Views در پروژه قرار دارند.

کلاس PartialViewResult

برنامه نویسان Asp.Net Web Form با مفهومی به نام UserControl ها آشنا هستند. در mvc معادلی برای UserControl ها وجود دارد که به آن PartialView می گویند. درواقع PartialView ها View هایی هستند که به صورت مستقیم قابل دسترسی نبوده و می بایستی درون سایر View ها رندر شوند. کلاس PartialViewResult برای زمانی است که خروجی یک اکشن متد از جنس PartialView باشد. در این حالت برای انتخاب PartialView از متد PartialView که در کلاس Controller تعریف شده است استفاده می کنیم.

کلاس RedirectResult

گاهی مواقع نیاز پیدا خواهیم کرد که از یک اکش متد به یک اکشن متد دیگر رجوع کنیم. در این صورت از کلاس RedirectResult در خروج اکشن متد استفاده خواهیم کرد.برای این منظور از متد Redirect همراه با آدرس اکشن متد مورد نظر استفاده خواهیم کرد.

کلاس RedirectToRouteResult

از این کلاس زمانی که بخواهیم به یک روت دیگر تغییر مسیر دهیم استفاده خواهیم کرد. دو متد RedirectToAction و RedirectToRoute برای این منظور در کلاس Controller تعریف شده اند.

کلاس ContentResult

زمانی که بخواهیم یک محتوای رشته ای را به عنوان نتیجه یک اکشن متد تعریف کنیم از این کلاس استفاده خواهیم کرد. متد Content که دارای سه مدل مختلف می باشد برای این منظور استفاده می شود.

کلاس JsonResult

گاهی مواقع خروجی یک اکشن متد یک شی Json (Java Script Object Notation) می باشد. در این صورت از کلاس JsonResult به عنوان نوع خروجی اکشن متد استفاده می کنیم. متد Json در کلاس کنترلر یک شی جیسون سریال شده (Sterilized Json Object)را بر می گرداند. در مواقعی که بخواهیم اکشن متد را توسط جاوااسکریپت صدا بزنیم این متد کارایی بالایی خواهد داشت.

کلاس FileResult

زمانی که خروجی اکشن متد از جنس باینری باشد(مانند فایل ها) از این کلاس به عنوان خروجی متد استفاده می کنیم. متد File بدین منظور به کار می رود.

کلاس EmptyResult

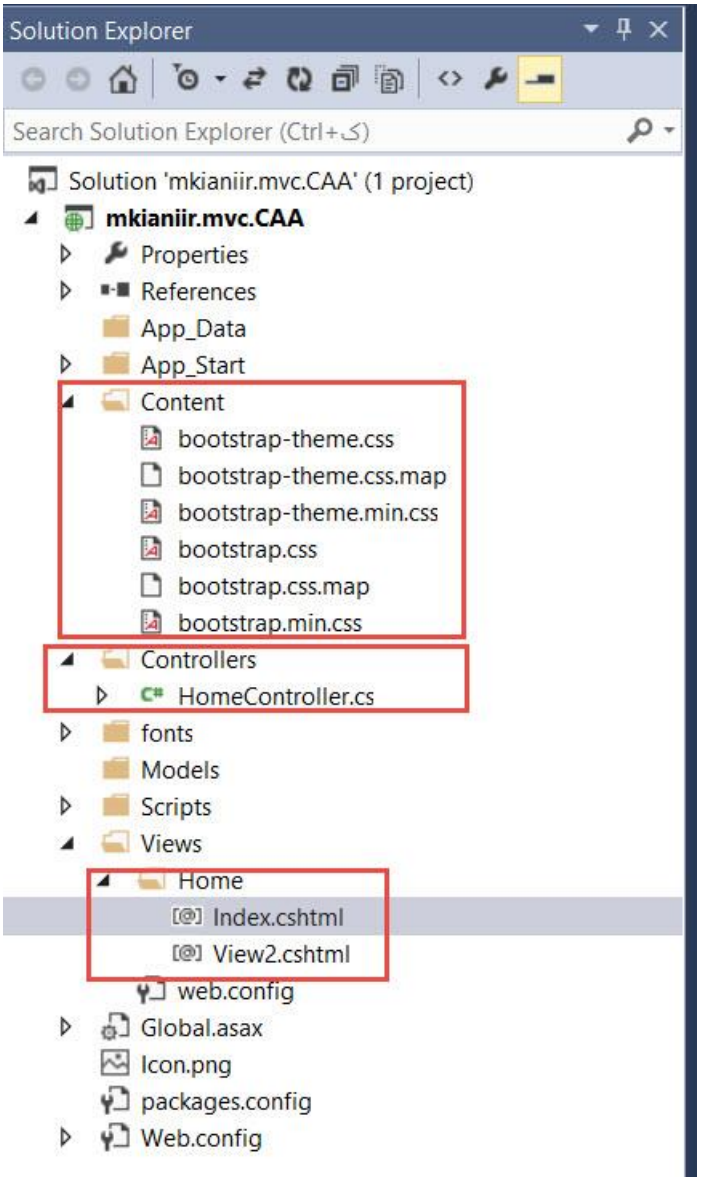
اگر اکشن متد ما هیچ مقداری را برنگرداند از این کلاس استفاده خواهیم کرد .

مثال هایی از انواع خروجی های اکشن متد ها

در این قسمت به جهت آشنایی بیشتر با اکشن متد ها برای برخی از کلاس های فوق مثال هایی آورده شده است.

برای این منظور یک پروژه ایجاد کنید. من نام آن را mkianiir.mvc.CAA گذاشتم. یک کنترلر به نام Home به پروژه خود اضافه کنید و برای متد Index آن یک ویو ایجاد کنید. همچنین یک ویوی دیگر در

پوشه Views اضافه کنید و نام ان را View2 بگذارید. توسط کنسول پکیج منیجر هم با استفاده از دستور install-package bootstrap کتابخانه bootstrap را به پروژه خود اضافه کنید.



شکل ۵ - ۱

حال کلاس HomeController را باز کنید و کد های آن را به صورت زیر تغییر دهید:

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

namespace mkianiir.mvc.CAA.Controllers
```

```
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
        public ViewResult GotoView2()
        {
            return View("View2");
        }
        public RedirectResult RedirectToView2ViaUrl()
        {
            return Redirect("~/home/GotoView2");
        }
        public RedirectToRouteResult RedirectToView2ViaRoute()
        {
            return RedirectToRoute(new
            {
                action = "GotoView2"
            });
        }
        public ContentResult ShowContentResult()
        {
            return Content("<h1>this is a sample content result</h1>
<a href=\"/\" >Back to home</a>");
        }
    }
}
```

متد GotoView2 دارای دستور return View(“View2”) می باشد. این دستور View2 را به عنوان یک ViewResult بر می گرداند. همانطور که مشاهده می کنید لزومی ندارد که نام اکشن متد با نام ویو یکسان باشد. بلکه با استفاده از آرگومانی که می توان به متد View ارسال کنیم می توانیم ویوی مورد نظر را انتخاب نمائیم.

متد RedirectToView2ViaUrl یک خروجی از جنس RedirectResult دارد. همانطور که در توضیحات قبلی گفته شد این متد آدرس یک اکشن متد دیگر را به عنوان ورودی می گیرد و کاربر را به آن اکشن متد ارجاع می دهد.

خروجی متد RedirectToViewViaRoute از جنس RedirectToRoute می باشد. عملکرد این متد شبیه به متد Redirect است با این تفاوت که متد RedirectToRoute به شما این قابلیت را می دهد که بتوانید با مقدار دهی پارامتر های روت از یک روت به روت دیگری سوئیچ کنید.

خروجی متد ShowContentResult از جنس کلاس ContentResult می باشد. همانطورکه گفته شد از این کلاس زمانی که بخواهیم خروجی یک اکشن متد به صورت محتوای رشته ای باشد از آن استفاده می کنیم. متد Content در کلاس Controller برای همین منظور طراحی شده است.

حال در ادامه ویو هایی که در دستورات فوق مورد استفاده قرار گرفته اند را به صورت زیر ایجاد یا تکمیل نمائید:

فایل های Index.cshtml را باز کنید و کد های آن را به ترتیب زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <link href="~/Content/bootstrap-theme.min.css" rel="stylesheet"
/>
    <title>Index</title>
</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h1>Action Methods Demo.</h1>
        </div>
        <div class="panel-body">
            @Html.ActionLink("Go to view2", "GotoView2", null, new
{
    @class = "btn btn-success"
})
            @Html.ActionLink("Redirecto to view2 via url",
"RedirectToView2ViaUrl", null, new
{
    @class = "btn btn-primary"
})
```

```
            @Html.ActionLink("Redirecto to view2 via route",
"RedirectToView2ViaRoute", null, new
{
    @class = "btn btn-warning"
})
            @Html.ActionLink("Show content result",
"ShowContentResult", null, new
{
    @class = "btn btn-success"
})
        </div>
    </div>
</body>
</html>
```

دو تگ link موجود در تگ head برای بارگزاری فایل های css مربوط به پروژه کتابخانه bootstrap می باشد. فایل bootstrap-theme.min.css بر اساس فایل bootstrap.css تغییراتی را در ظاهر پیش فرض کتابخانه bootstrap ایجاد کرده است.

همانظور که مشاهده می کنید در فایل Index.cshtml از متد راهنمای ActionLink برای ایجاد لینک هایی به سایر اکشن متد های تعریف شده در کلاس HomeController تعریف شده است.

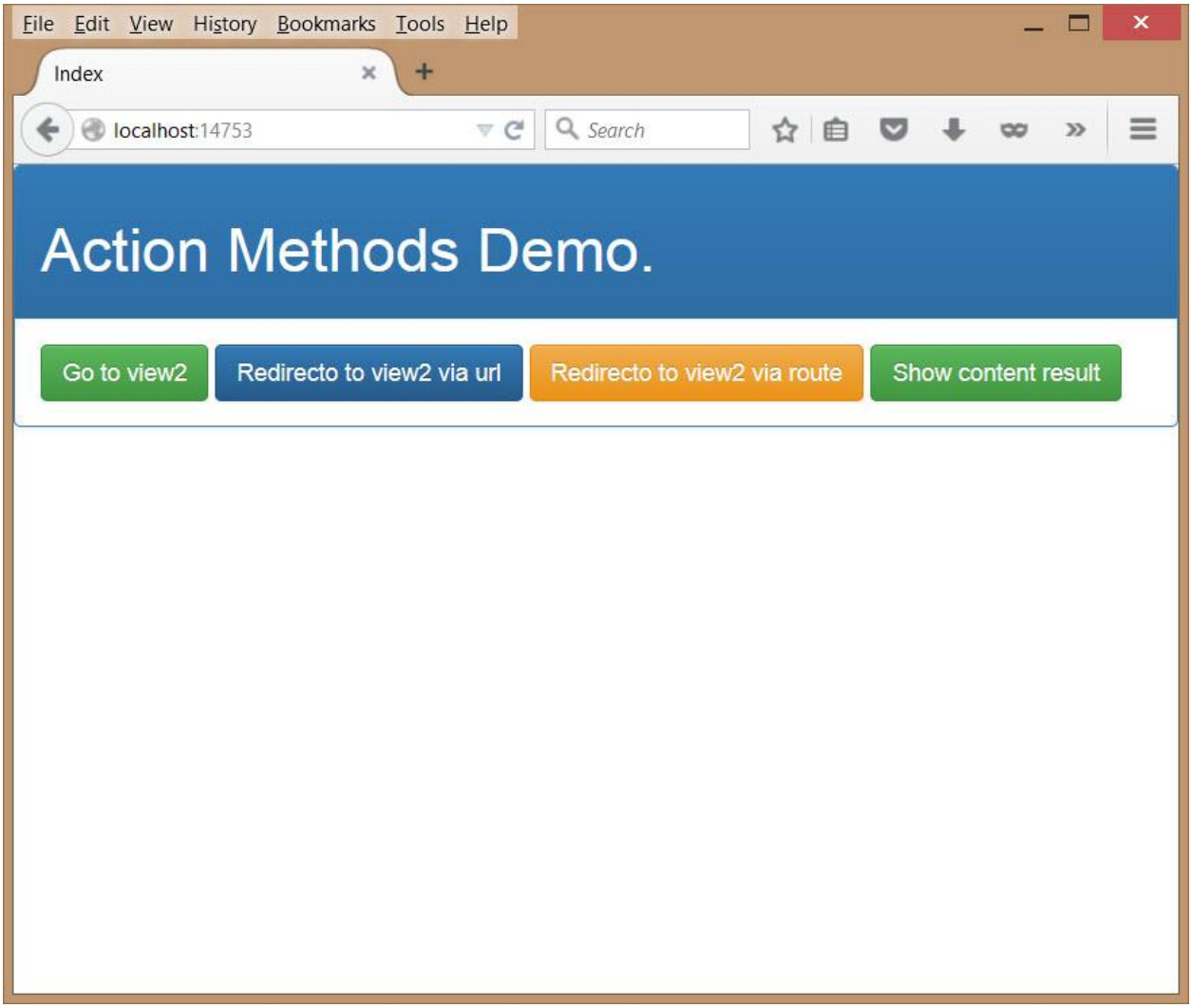
تغییرات این فایل را ذخیره و فایل رابیندید.

حال فایل View2.cshtml را باز کنید و دستورات آن را به صورت زیر تغییر دهید:

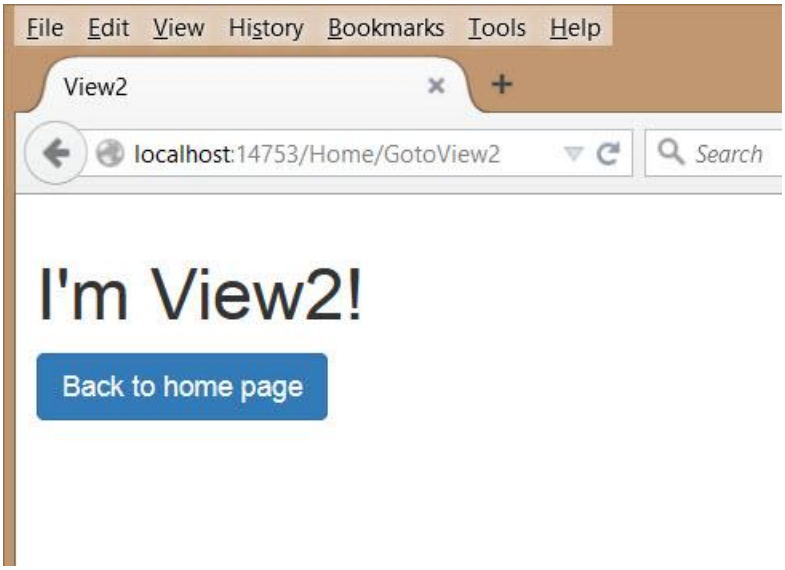
```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>View2</title>
</head>
<body>
    <div style="padding:10px">
        <h1> I'm View2!</h1>

        @Html.ActionLink("Back to home page", "Index", null, new
   {
       @class = "btn btn-primary"
```

```
    })
     </div>
</body>
</html>
```

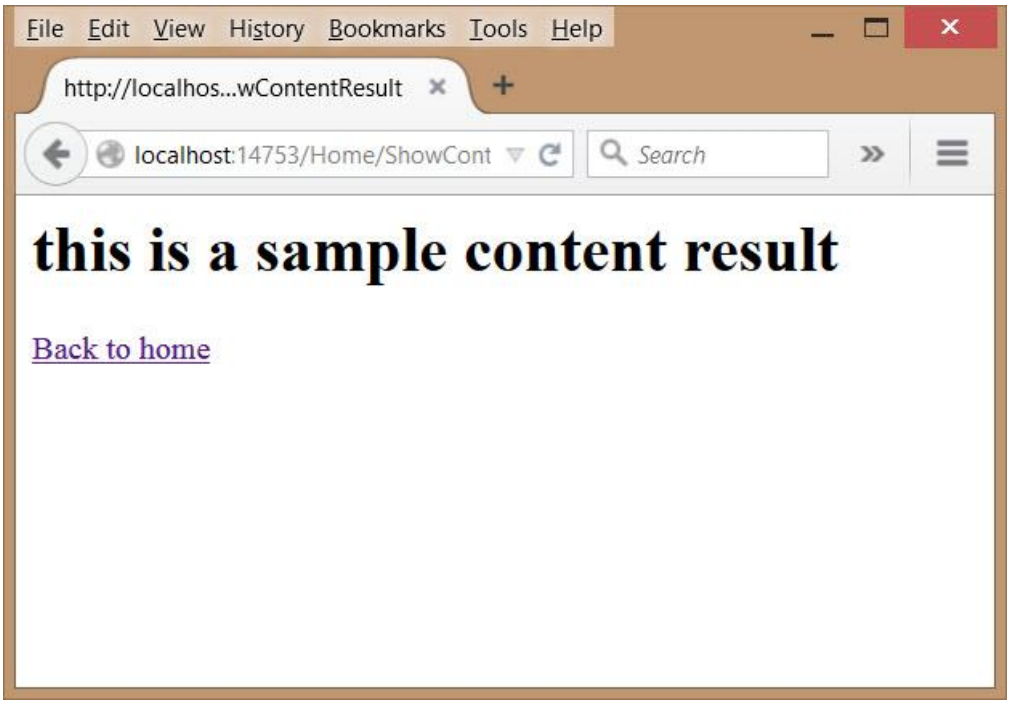
پس از انجام تغییرات عملیات فوق برنامه را اجرا کنید و بر روی لینک های مختلف کلیک کنید و نتایج را به دقت بررسی کنید.عکس های زیر نمایشی از اجرای برنامه را نشان می دهد:



شکل ۵ - ۲



شکل ۵ - ۳



شکل ۵ - ۴

ارسال پارامتر به اکشن متد ها

اکشن متد ها می توانند دارای آرگومان نیز باشند. Mvc میتواند بر اساس پارامتر هایی که در آدرس مرورگر کاربر وارد می شوند مقادیر را به صورت پارامتر به اکشن متد مربوطه ارسال نمائد. برای درک بهتر این موضوع اجازه دهید با یک مثال وارد بحث شویم.

کلاس HomeController را باز کنید و متد زیر را به آن اضافه کنید:

```
public ContentResult ShowParameter(String action, string id)
        {
            return Content(String.Format("<h1>The action is
:<b>'{0}'</b></h1><br/><h1>The id is :<b>'{1}'</b></h1>", action,
id));
        }
```

متد فوق یک اکشن متد است با دو آرگومان به نام های actionو id.

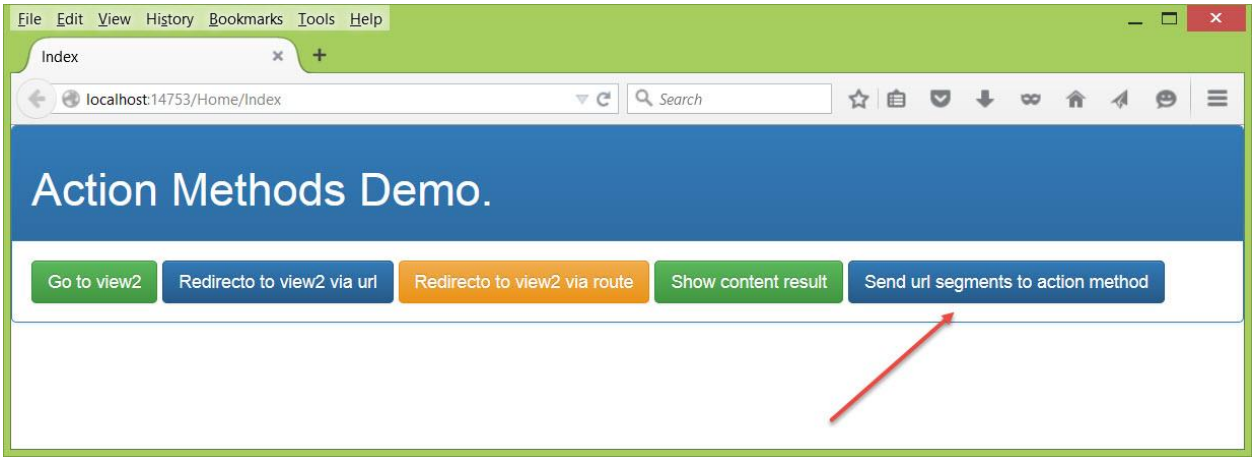
همانطور که می دانید روت تعریف شده در متد RegisterRoutes مربوط به کلاس RouteConfig دارای دو بخش actionو id است. بنابر این mvc مقادیری را که برای این بخش ها در آدرس وارد می شوند را به صورت خودکار به این متد ارسال می کند.

خروجی این متد یک ContentResult می باشد که مقادیر آرگومان های متد را به صورت کد html نشان می دهد.

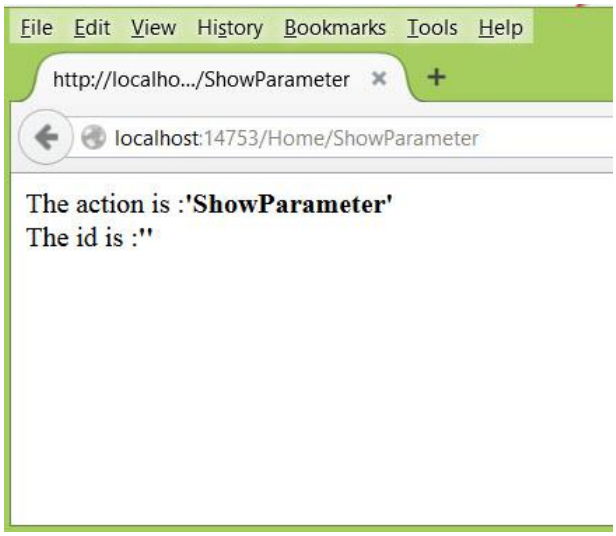
فایل Index.cshtml را باز کنید و کد های زیر را به ان اضافه کنید:

```
@Html.ActionLink("Send url segments to action method",
"ShowParameter",null, new
{
    @class = "btn btn-primary"
})
```

حال برنامه را اجرا و روی لینک Send url segments to action method کلیک کنید و نتیجه را مشاهده کنید.



شکل ۵ - ۵



شکل ۵ - ۶

همانطور که مشاهده می کنید مقدار ShowParameter به عنوان پارامتر به اکشن متد ارسال شده و در خرورجی نشان داده شده است. چون هیچ مقداری برای id در آدرس وجود ندارد بنابر این مقدار null در نظر گرفته می شود.

حال آدرس را در مرورگر به صورت زیر تغییر دهید:

http://localhost:14753/Home/ShowParameter/**some-id**

و کلید enter را بفشارید و نتیجه را مشاهده کنید:

شکل ۵ - ۷

همانطور که مشاهده می کنید مقدار id نیز در این حالت به به اکشن متد ارسال شده و در خروجی صفحه نمایش داده می شود.

این رفتاری mvc در خصوص آدرس دهی ها باعث می شود که کاربران راحت تر بتوانند آدرس های برنامه شما را درک کرده و در به ذهن سپاری و نیز اشتراک گذاری آن ها بهتر و سریعتر عمل کنند.

نحوه تشخیص پارامترها توسط mvc

یک سوال اساسی که ممکن است الان در ذهن شما شکل گرفته باشد این است که mvc چگونه مقادیر را از آدرس ها واکشی می کند و همچنین نوع پارامتر هایی که در اکشن متد ها مورد استفاده قرار می گیرد را چگونه تشخیص می دهد؟

فریم ورک mvc دارای دو کامپوننت برای پاسخ به این دو سوال دارد. ابتدا توسط کامپوننتی به نام Value Provider داده ها واکشی شده و سپس توسط کامپوننتی به نام Model Binder داده های دریافتی برای استفاده در اکشن متد ها اماده می شوند.

کلاس Controller دارای خاصیتی به نام Request می باشد. خاصیت Request نمونه ای از کلاس HttpRequest می باشد. این کلاس دارای خواصی از جمله Form و QueryString می باشد که خود این خواص نمونه ای از کلاس NameValueCollection می باشند. همچنین کلاس HttpRequest دارای خاصیت دیگری به نام Files از جنس HttpFileCollectionBase می باشد. علاوه بر این کلاس Controller دارای خاصیتی به نام RouteData است. خاصیت RouteData که نمونه ای از کلاس RouteData می باشد دارای خاصیتی به نام Values از جنس RouteValueCollection است. فریم ورک mvc دارای Value Provider ای است که می تواند داده ها را از بخش های مختلفی که بیان شد واکشی کند و در اختیار کامپونت دیگری به نام Model Binder قرار دهد. کامپونت Model Binder وظیفه دارد که داده دریافتی را به نوعی که مورد نیاز اکشن متد می باشد تبدیل کند.

به عنوان مثال متد زیر را به کلاس HomeController اضافه کنید:

```
public ContentResult ShowPowerOfId(int id)
        {
            return Content(String.Format("Power of id is
:<b>'{0}'</b>",id*id));
        }
```

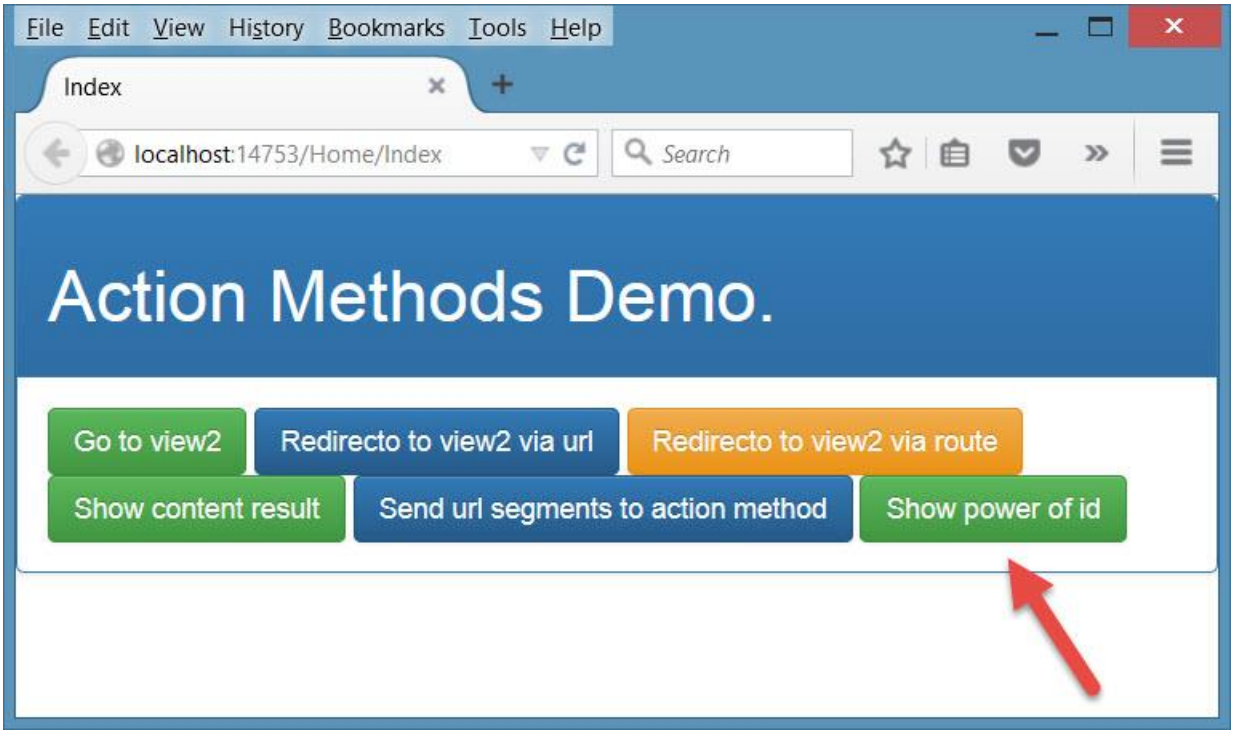
همانطور که مشاهده می کنید وروردی این متد از جنس int می باشد.

فایل Index.cshtml را باز کرده و کد های زیر را به آن اضافه کنید:

```
@Html.ActionLink("Show power of id", "ShowPowerOfId", new
        {
            id=10
        }, new
{
    @class = "btn btn-success"
})
```

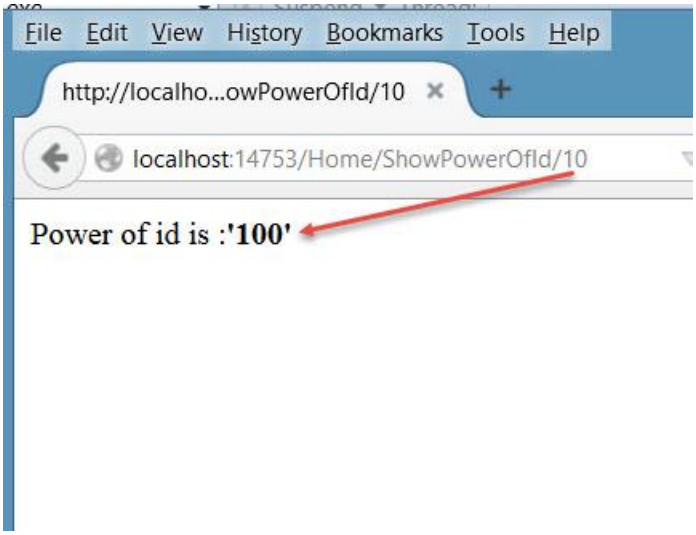
دستور فوق یک اکشن لینک ایجاد می کند که اشاره به اکشن متد ShowPowerOfId دارد. همچنین مقدار id را برابر با ۱۰ تنظیم کرده است.

حال برنامه را اجرا کنید و روی لینک Show power of id کلیک کنید.



شکل ۵ - ۸

نتیجه در شکل زیر نشان داده شده است:



شکل ۵ - ۹

همانطور که مشاهده می کنید مقدار ۱۰۰ یعنی توان دوم عدد ۱۰ در خروجی چاپ شده است. این بدان معنی است که mvc تشخیص می دهد که آرگومان ورودی اکشن متد ما از نوع int می باشد.بنابر این کامپوننت Model Binder مقدار ۱۰ واکشی شده توسط Value Provider را تبدیل به int کرده و در اختیار اکشن متد قرار می دهد.

مقدار دهی اولیه آرگومان های اکشن متد

در آخرین دستوری که به فایل Index.cshtml برای اشاره به اکشن متد ShowPowerOfId ایجاد کردیم مقدار id را برابر با ۱۰ در نظر گرفتیم. این مقدار دهی در اکشن لینک باعث می شود که به صورت خودکار پارامتر id در آدرس صفحه قرار گیرد. حال مقدار دهی پیش فرض id را از دستور مذکور حذف کنید. یعنی دستورات

```
@Html.ActionLink("Show power of id", "ShowPowerOfId", new
        {
            id=10
        }, new
{
    @class = "btn btn-success"
})
```

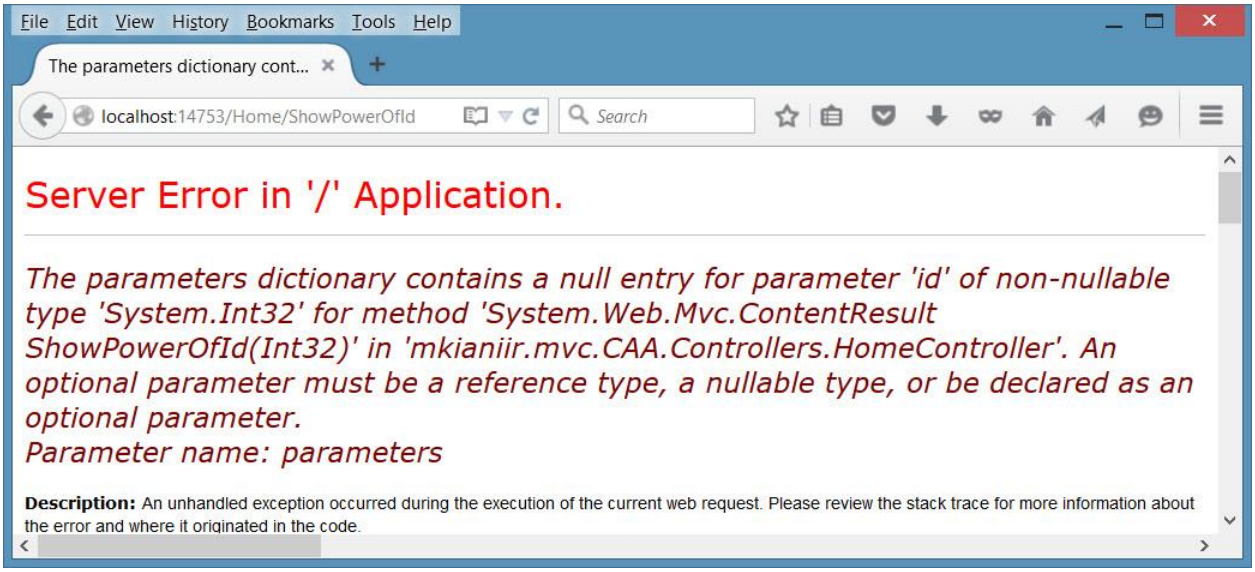
را به دستورات

```
@Html.ActionLink("Show power of id", "ShowPowerOfId",null
, new
{
```

```
    @class = "btn btn-success"
})
```

تبدیل کنید.

حال برنامه را مجددا اجرا کنید و روی لینک Show power of id کلیک کنید.نتیجه شبیه به شکل زیر خواهد بود:



شکل ۵- ۱۰

همانطور که در شکل فوق مشخص است در آدرس صفحه اثری از id وجود ندارد. بنابر این کامپوننت Value Provider مقدار null را به کامپوننت Model Binder ارسال می کند. چون مقدار id از نوع int می باشد و int یک value type بوده و در حالت عادی نمی تواند مقدار null داشته باشد بنابر این کامپوننت Model Binder نمی تواند مقدار null را به id درون اکشن متد ShowPowerOfId نسبت دهد و در نتیجه برنامه با خطا مواجه خواهد شد.

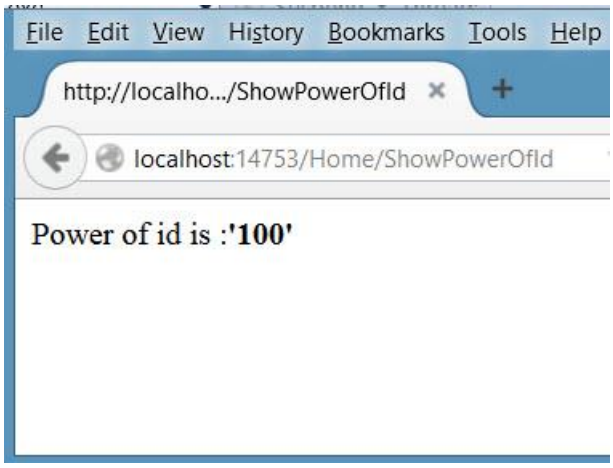
در این مواقع می توان علاوه بر تعیین مقدار پیش فرض id در اکشن لینک از مقدار پیش فرض آرگومان ها در اکشن متد ها استفاده کرد.

برای درک این موضوع دستورات مربوط به اکشن متد ShowPowerOfId را به صورت زیر تغییر دهید:

```
  public ContentResult ShowPowerOfId(int id=10)
        {
            return Content(String.Format("Power of id is
:<b>'{0}'</b>",id*id));
```

```
}
```

به قسمت هایلایت شده در دستورات فوق دقت کنید. مقدار پیش فرض id برابر با ۱۰ در نظر گرفته شده است. حال مجددا برنامه را اجرا کنید. بر روی لینک Show power of id کلیک کنید. نتیجه مشابه شکل زیر خواهد بود.



شکل ۵- ۱۱

همانطور که مشاهده می کنید با آن که در آدرس اثری از id وجود ندارد اما برنامه بدون خطا اجرا شده و مقدار ۱۰۰ را که توان دوم عدد ۱۰ است در خروجی نشان می دهد.

روش دیگر برای اینکه با خطایی که چندی پیش توضیح داده شد مواجه نشوید این است که آرگومان های value type مثل int را به nullable تبدیل کنید. یعنی به جای int در اکشن متد ShowPowerOfId از int? (یا Nullable<int>) استفاده نمائید. در این حالت Model Binder می تواند مقدار Null را به آرگومان شما نسبت دهد. فقط در این حالت باید هنگام استفاده از آرگومان مورد نظر null بودن آن را مورد بررسی قرار دهید .

ارسال اطلاعات از کنترلر به View توسط ViewBag

در بخش ایجاد فرم Contact مشاهده کردید که نمونه ای از کلاس Contact به عنوان model به ویو های مورد نظر ارسال شد و ویو توانست جهت نمایش اطلاعات به کاربر از مدل بایند شده به خود استفاده کند. گاهی

مواقع نیاز است که اطلاعاتی از سمت کنترلر به ویو ارسال شود. ولی این اطلاعات نیاز به یک کلاس جهت پیاده سازی مدل نیست. به عنوان مثال فرض کنید بخواهیم تاریخ سرور را به ویو فرستاده و در آن جا نمایش دهیم. یا بخواهیم عنوان صفحه را از کنترلر به ویو ارسال کنیم تا در تگ title درون head به نمایش در آید. یکی از روش هایی که می توان در اینگونه مواقع از آن بهره برد استفاده از ViewBag می باشد. در کلاس Controller خاصیتی به نام ViewBag از نوع فیلد dynamic می باشد. فیلد های دینامیک همزمان با C# 2010 ظهور کردند.

جهت اطلاعات بیشتر در مورد این فیلد به آدرس زیر مراجعه نمائید.

https://msdn.microsoft.com/en-us/library/dd۲۶۴۷۳۶.aspx

متد ShowDate را به کلاس HomeController اضافه نمائید:

```
public ViewResult ShowDate()
        {
            DateTime dt = DateTime.Now;
            System.Globalization.PersianCalendar prc = new
System.Globalization.PersianCalendar();
            String pdate =
String.Format("{0:0000}/{1:00}/{2:00}",prc.GetYear(dt),prc.GetMonth(
dt),prc.GetDayOfMonth(dt));
            String gdate = dt.ToShortDateString();
            ViewBag.Pdate = pdate;
            ViewBag.Gdate = gdate;
            return View();
        }
```

در متد ShowDate تاریخ جاری سرور به دو شکل میلادی و شمسی تعریف شده است. مقادیر Pdate و Gdate نگهدارنده این داده ها در فیلد ViewBag خواهند بود.

حال بر روی متد ShowDate کلیک راست کرده و یک ویو برای آن ایجاد کنید. سپس کد های ویوی ایجاد شده را به صورت زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>
```

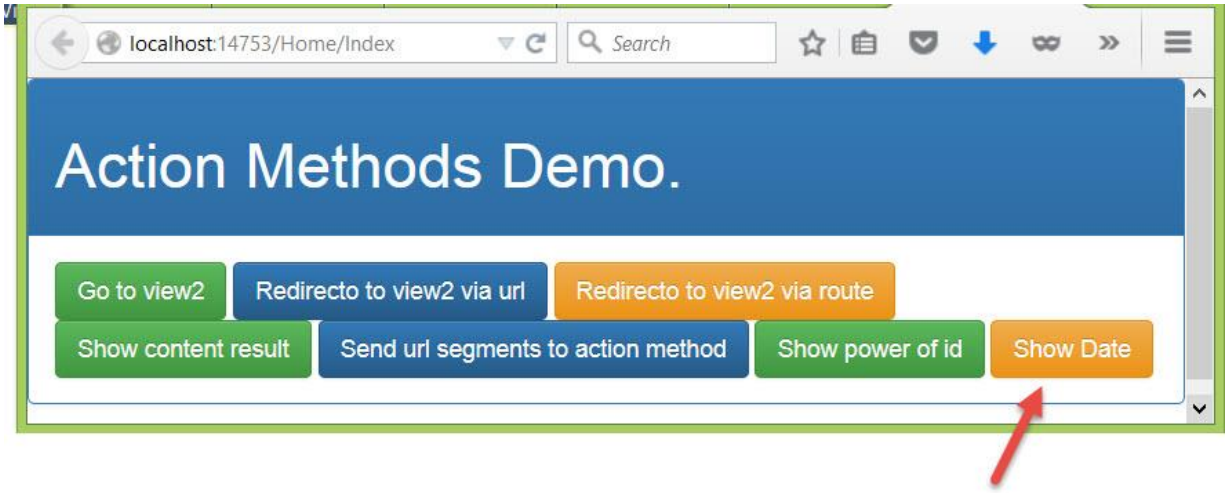
```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>ShowDate</title>
</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h1>Show Date View</h1>
        </div>
        <div class="panel-body">
           The server date is : <b>@ViewBag.Gdate</b>
            <hr />
            The Persian date is : <b>@ViewBag.Pdate</b>
        </div>
    </div>
</body>
</html>
```

همانطور که مشاهده می کنید توسط دستور @ViewBag می توانیم به مقادیر آن دسترسی داشته باشیم. فایل ShowDate.chtml را ذخیره کرده و ببندید.

فایل Index.cshtml را باز کنید و دستور زیر را به آن اضافه کنید.

```
@Html.ActionLink("Show Date", "ShowDate", null, new
{
    @class = "btn btn-warning"
})
```

فایل Index.cshtml را ذخیره کرده وببندید. سپس برنامه را اجرا کنید و روی لینک Show Date کلیک کنید.



شکل ۵ - ۱۲

نتیجه شبیه به شکل زیر خواهد بود.

شکل ۵ - ۱۳

فیلتر ها

همانظورکه فراگرفتید اکشن متد ها یک رابطه یک به یک با درخواستی که کاربر ارسال می کند دارند.زمانی که آدرسی شبیه به آدرس

http://localhost/Home/Index

توسط کاربر در مرورگر مورد درخواست قرار می گیرد Mvc به سراغ کنترلری به نام Home رفته و درون آن به دنبال متدی به نام Index گشته و دستورات درون آن را اجرا کرده و نهایتا خروجی به یکی از روش هایی که در بخش قبلی توضیح داده شد به کاربر نشان داده می شود.

حال فرض کنید که درون کنترلری به نام Home اکشن متد هایی داریم که صرفا کاربرانی که هویت آن ها بررسی و تائید شده امکان اجرای آن ها را دارند. در این حالت اکشن متد مورد نظر چگونه باید نوشته شود ؟

برای مشخص شدن جواب اجازه دهید تا طبق روال قبل با یک مثال شروع کنیم.

پس بنابر این یک پروژه جدید ایجاد کنید. سپس یک کنترلر به نام Home به صورت زیر به آن اضافه کنید.

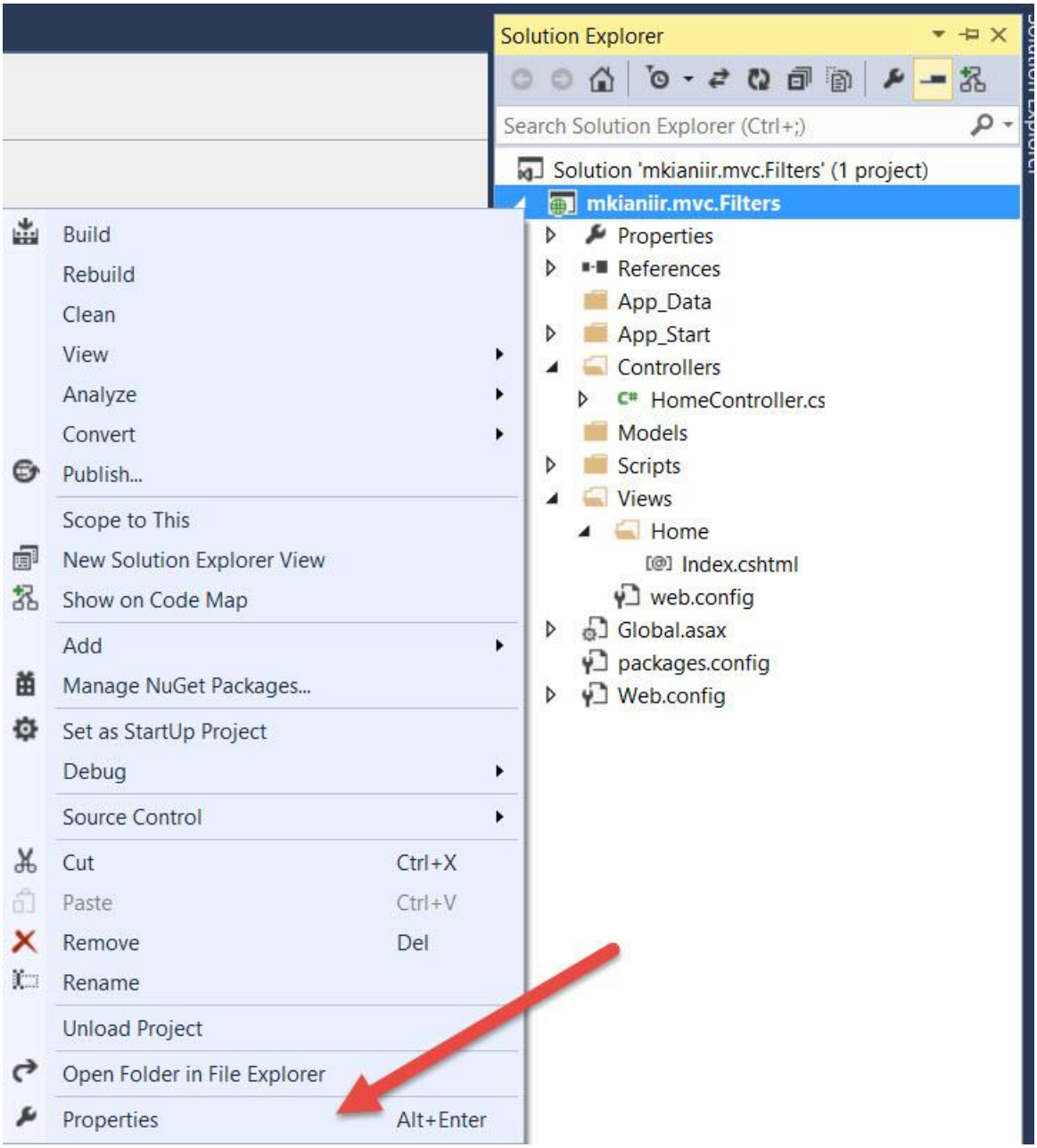
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace mkianiir.mvc.Filters.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
     }
}
```

بر روی متد Index کلیک راست کرده و از منوی باز شده گزینه Add View را کلیک کنید تنظیمات مورد نظر را طبق آموخته های خود از بخش های قبلی انجام داده تا یک ویو به نام Index ایجاد شود. سپس کد های آن را به صورت زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
```

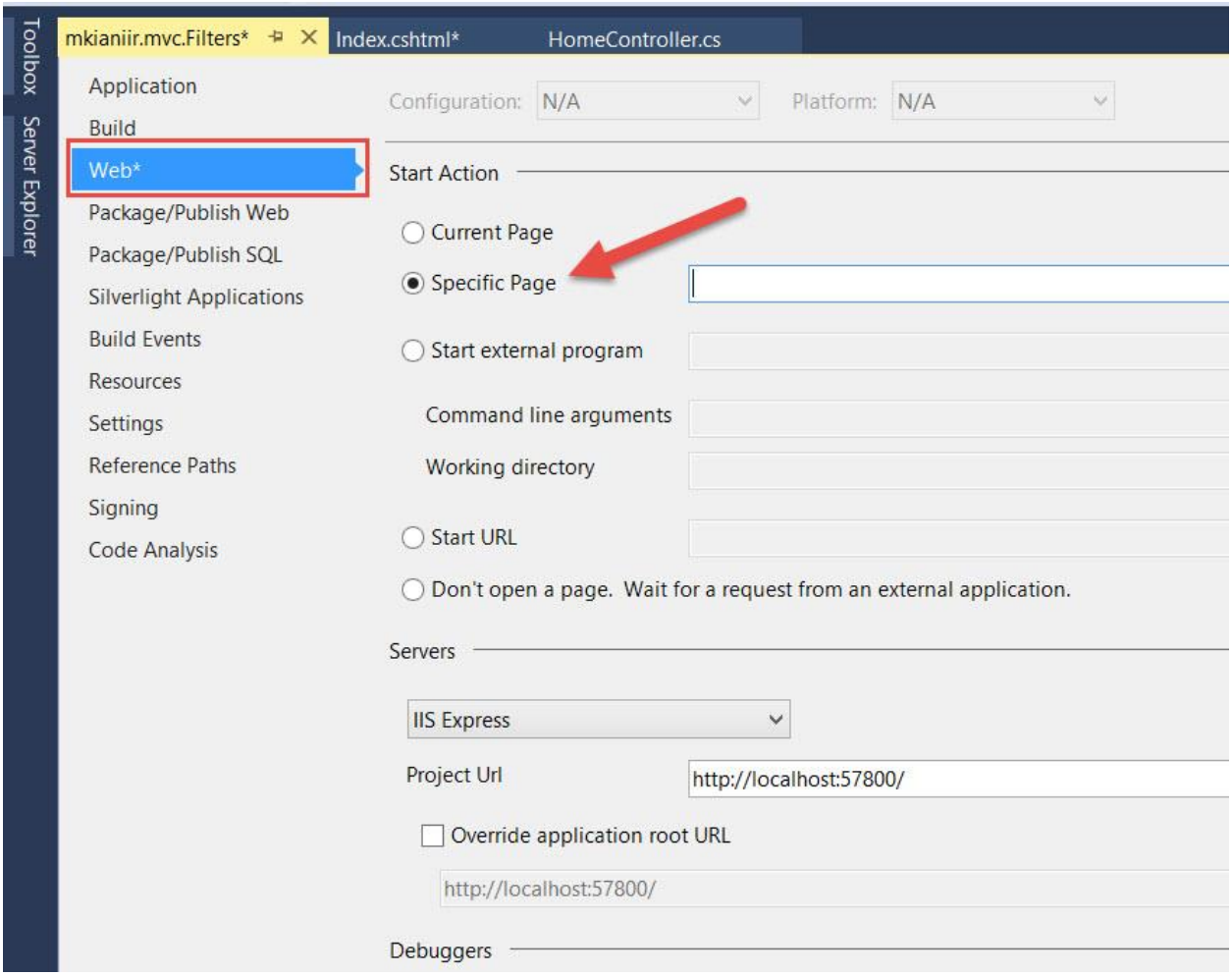
```
    <title>Index</title>
</head>
<body>
    <div>
        <h1>I'm index method, only autheticated users can see
me!</h1>
    </div>
</body>
</html>
```

بر روی نام پروژه در Solution Explorer کلیک راست کنید و از منوی باز شده گزینه Properties را کلیک کنید.



شکل ۵ - ۱۴

در صفحه خصوصیات پروژه به قسمت Web رفته و از بخش Start Action گزینه Spesific Page را انتخاب کنید. هیچ مقداری نیاز نیست که در باکس مقابل آن وارد نمائید. سپس تنظیمات را ذخیره کنید. این کار باعث می شود که پروژه صرف نظر از اینکه در حال ویرایش کدام بخش پروژه هستیم همواره از آدرس اصلی (خانه) اجرا شود.



شکل ۵ - ۱۵

حال برنامه را اجرا کنید.مشاهده خواهید کرد که اکشن Index اجرا شده و خروجی مربوط به Index.cshtml نشان داده می شود.

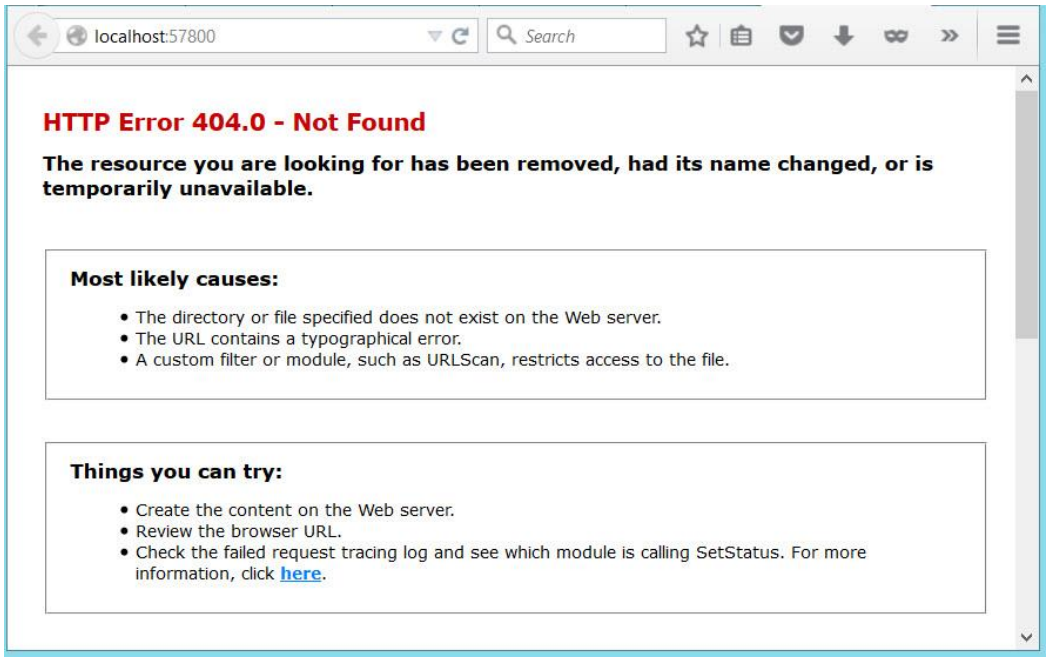
شکل ۵ - ۱۶

این خواسته ما نبود، چرا که می می خواستیم صرفا کاربرانی که هویت آن ها توسط برنامه تائید شده است به متد Index دسترسی داشته باشند. چه باید کرد؟ اولین راهی که ممکن است به ذهن برسد این است که در داخل

متد Index بررسی کنیم که آیا کاربر فعلی تائید هویت شده است یا خیر؟ برای این منظور دستورات متد index را به صورت زیر تغییر دهید:

```
public ActionResult Index()
        {
            if (!Request.IsAuthenticated)
                return HttpNotFound();
            return View();
        }
```

حال برنامه را مجددا اجرا کنید.نتیجه شبیه به زیر خواهد بود:



شکل ۵ - ۱۷

چه اتفاقی افتاد؟

همانطور که مشاهده می کنید در متد Index از دستور Request.IsAuthenticated استفاده شده است تا بررسی شود که آیا هویت کاربر تائید شده است یا خیر؟ و همانطور که مشاهده می کنید چون کاربر اعتبار سنجی نشده است شرط مذکور دارای مقدار false بوده و در نتیجه متد Index به جای بازگردانی ویوی مورد نظر، متد HttpNotFound را فراخوانی می کند. این عمل باعث می شود تا خطای آشنای ۴۰۴ ایجاد و به کاربر نشان داده شود.

با این عمل توانستیم از ورود افراد بدون هویت به متد Index جلوگیری کنیم. این روش اگر چه جوابگو می باشد اما در پروژه های بزرگ نمی تواند روش بهینه ای باشد. چرا؟

فرض کنید چند متد دیگر نیز در کنترلر Home داریم که همگی آن ها نیاز به اعتبار سنجی کاربر دارند. بنابر این مجبور خواهیم بود که یک کد تکراری را در متد های مختلف بنویسیم. حال اگر در کنترلر های مختلف، متد های مختلفی بخواهند چنین مکانیزمی را پیاده سازی کنند چه؟ این روش در یک پروژه واقعی باعث ایجاد کد های تکراری زیادی خواهد شد که مدیریت آن را نیز سخت می کند.

روش بهینه تر برای این منظور استفاده از فیلتر ها می باشد که در ادامه با آن ها و انواع آن ها آشنا خواهیم شد.

فیلتر چیست؟

فیلتر ها در واقع صفاتی (Attribute) هستند که می توانند قبل از اجرای اکشن متدها و نیز پس از اجرای آن ها فراخوان شوند. در بسیاری از مواقع که یک نمونه آن در بخش قبلی بیان شد(تائید هویت کاربران) شما نیاز خواهید داشت به اینکه قبل از اجرای دستورات اکشن متد، دستورات دیگری اجرا شوند. و یا ممکن است پس از اجرا شدن یک اکشن متد، دستورات دیگری اجرا شوند. در این جاست که بحث فیلتر ها به میان می آیند. پس به طور خلاصه می توانیم بگوییم که فیلتر ها امکان کنترل بیشتر بر روی روند پردازش درخواست های کاربر را مهیا می کنند.

انواع فیلتر ها در mvc

در حال حاضر(در زمان نوشتن این کتاب) از فیلترهای زیر در MVC پشتیبانی می شود.

Authorization Filters

این دسته از فیلتر ها اینترفیسی به نام IAuthorizationFilter را پیاده سازی می کنند که برای بررسی اعتبار کاربر درخواست کننده می تواند به کار رود و می توان قبل از اینکه اکشن متد مورد نظر کاربر اجرا شود بررسی شود که آیا کاربر مجوز اجرای درخواست را دارد یا خیر.

Action Filters

این دسته از فیلتر ها که اینترفیس IActionFilter را پیاده سازی می کنند دارای دو متد OnActionExecting و OnActionExecuted می باشد. متد OnActionExecuting قبل از اجرای اکشن متد اجرا شده و متد OnActionExecuted پس از اجرای اکشن متد اجرا خواهد شد.

Result Filter

این دسته از فیلتر ها که اینترفیس IResultFilter را پیاده سازی می کنند دارای دو متد OnResultExecuting و OnResultExecuted می باشند. متد OnRsultExecuting قبل از اجرای

شی ActionResult که مسئول ایجاد خروجی اکشن متد است اجرا می شود و متد OnResultExecuted بعد از آن.

Execption Filters

این دسته از فیلتر ها که اینترفیس IExecptionFilter را پیاده سازی می کنند چنانچه در طی اجرای دستورات به استثنایی برخورد کنیم که مدیریت نشده باشند اجرا خواهند شد. کلاس HandledErrorAttribute یک نمونه از پیاده سازی این نوع فیلتر می باشد.

نکته: فیلتر ها بسته به تعریفشان می توانند در سطوح مختلف از جمله سطح اکشن متد یا سطح کنترلر و... فراخوانی شوند.

حال اجازه دهید نمونه ای از کاربرد فیلتر ها را در عمل مشاهده نمائیم.

برای این منظور متد Index را به صورت زیر تغییر دهید:

```
[Authorize]
  public ActionResult Index()
  {

      return View();
  }
```

همانطور که مشاهده می کنید از کلاس AuthorizeAttribute برای متد Index استفاده شده است(دستور هایلایت شده). کلاس AuthorizeAttribute یک کلاس پیش فرض پیاده سازی شده از نوع فیلتر Authorize می باشد. با استفاده از صفت مذکور برای متد Index دیگر نیاز به بررسی وضعیت اعتبار سنجی کاربر نخواهیم بود. چرا که قبل از اجرای دستورات متد Index این اعتبار سنجی به صورت خودکار بررسی می شود.اگر برنامه را با دستورات فوق اجرا کنید، نتیجه شبیه به شکل قبلی(خطای ۴۰۴) خواهد بود.

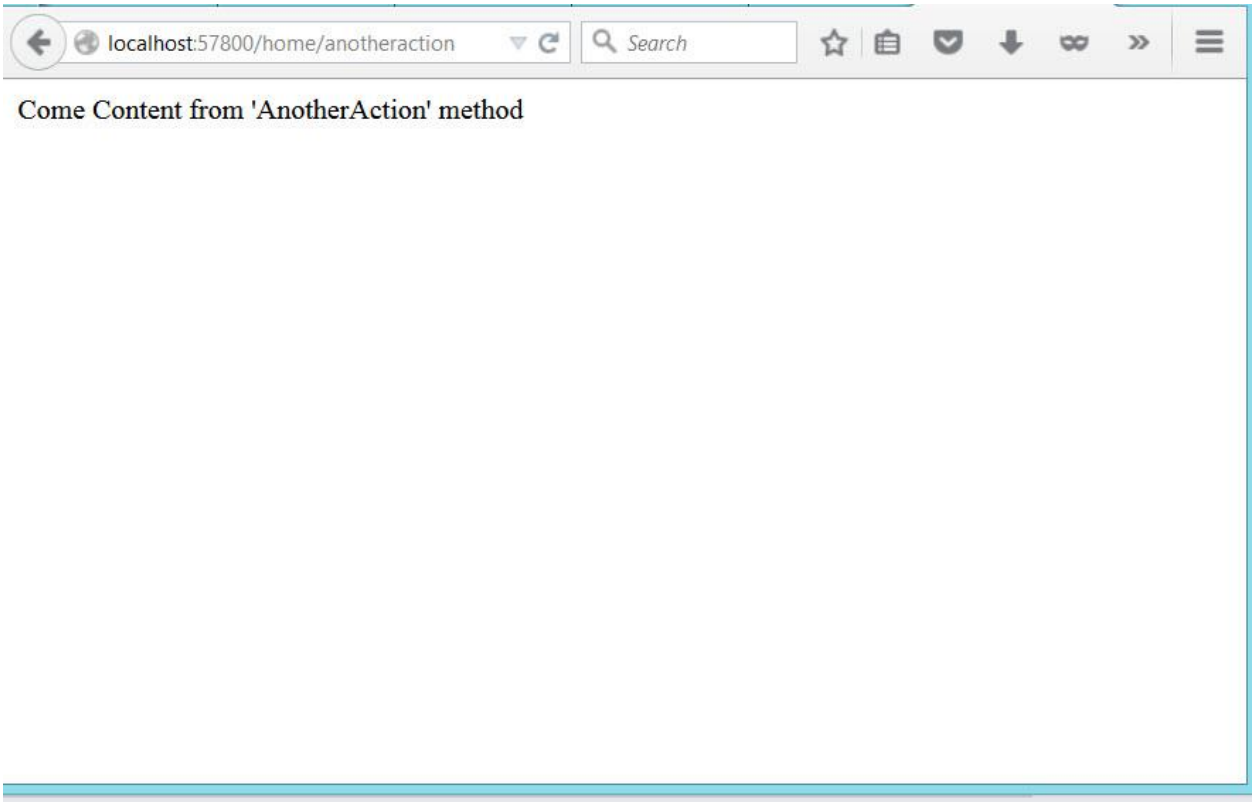
استفاده از فیلتر ها در کنترلر یا متد ها

کلاس HomeController را باز کنید و متد زیر را به آن اضافه کنید:

```
public ContentResult AnotherAction()
        {

            return Content("Come Content from 'AnotherAction'
method");
        }
```

حال برنامه را اجرا کنید و به آدرس زیر را در مرورگر تایپ کنید:

http://localhost:57800/home/anotheraction



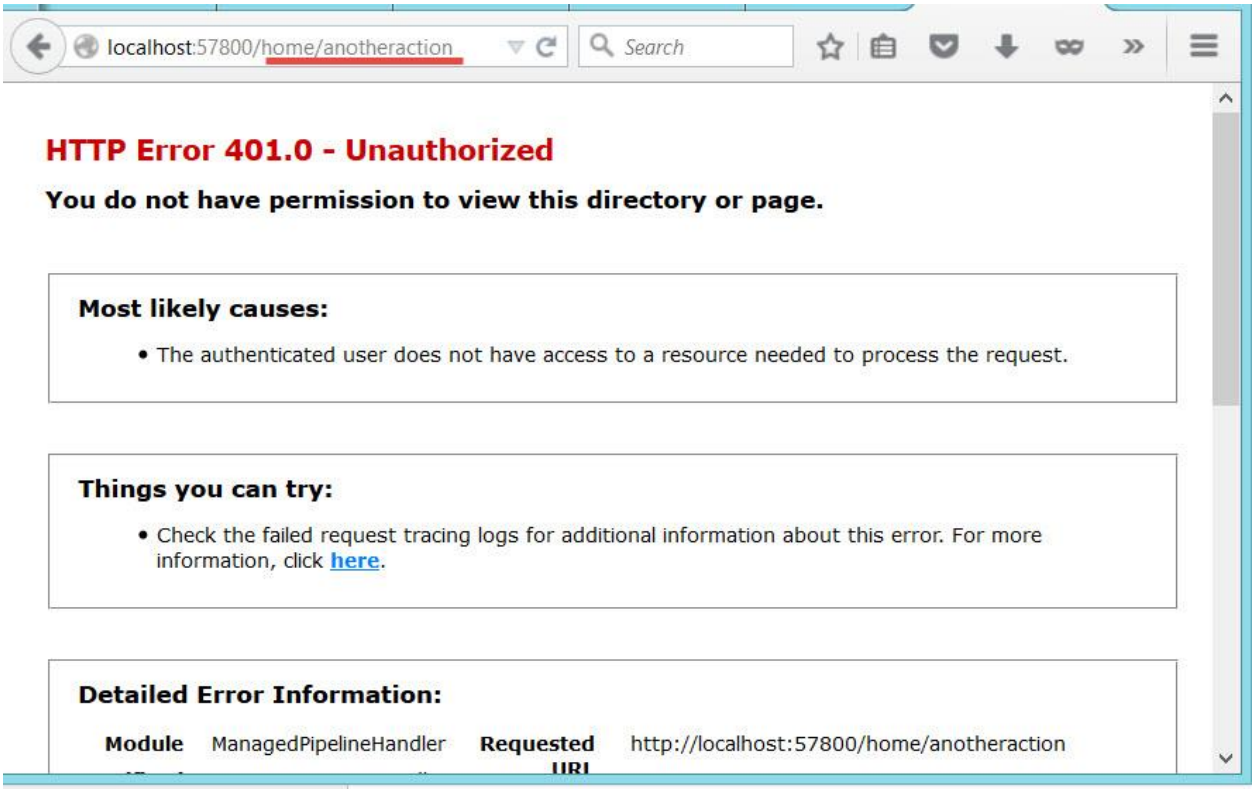
شکل ۵ - ۱۸

همانطور که مشاهده می کنید برنامه اجرا شده و خروجی به کاربر نشان داده می شود.

حال اگر صفت Authorize را به متد AnotherAction نیز اضافه کنید. در این صورت همانند متد Index در زمان اجرای برنامه به کاربر خطای ۴۰۴ نشان داده خواهد شد.

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

namespace mkianiir.mvc.Filters.Controllers
{
    public class HomeController : Controller
```

```
    {
        //
        // GET: /Home/
        [Authorize]
        public ActionResult Index()
        {

            return View();
        }
        [Authorize]
        public ContentResult AnotherAction()
        {

            return Content("Come Content from 'AnotherAction'
method");
        }
    }
}
```



شکل ۵ - ۱۹

همانطور که پیشتر اشاره شد Mvc به شما این اجازه را می دهد که به جای استفاده از فیلتر ها برای تک تک متد های یک کنترلر، برای خود کلاس فیلتر را اعمال کنید(این موضوع در تعریف فیلتر ها مشخص می شود). در این

صورت دیگر نیازی نیست که برای متد ها از فیلتری که برای کلاس تعریف شده است استفاده کنید. درواقع فیلتری که در سطح کنترلر نوشته می شود به صورت خودکار برای تمامی اکشن متد ها اعمال خواهد شد.

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

namespace mkianiir.mvc.Filters.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        public ActionResult Index()
        {

            return View();
        }

        public ContentResult AnotherAction()
        {

            return Content("Come Content from 'AnotherAction'
method");
        }
    }
}
```

همانطور که مشاهده می کنید صفت Authorize از متد ها حذف و برای کلاس به کار برده شده است. در این صورت صفت مذکور تمامی متد های کنترلر اعمال خواهد شد.

کش کردن اطلاعات با استفاده از فیلتر OutputCache

فیلتر OutputCache بر اساس زمانی که بر حسب ثانیه برای آن تنظیم می شود می تواند خروجی یک اکشن متد را کش کند. برای درک بهتر موضوع یک کنترلر به نام ActionFilterController به پروژه اضافه کنید.

سپس متد Index آن را به صورت زیر تغییر دهید:

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

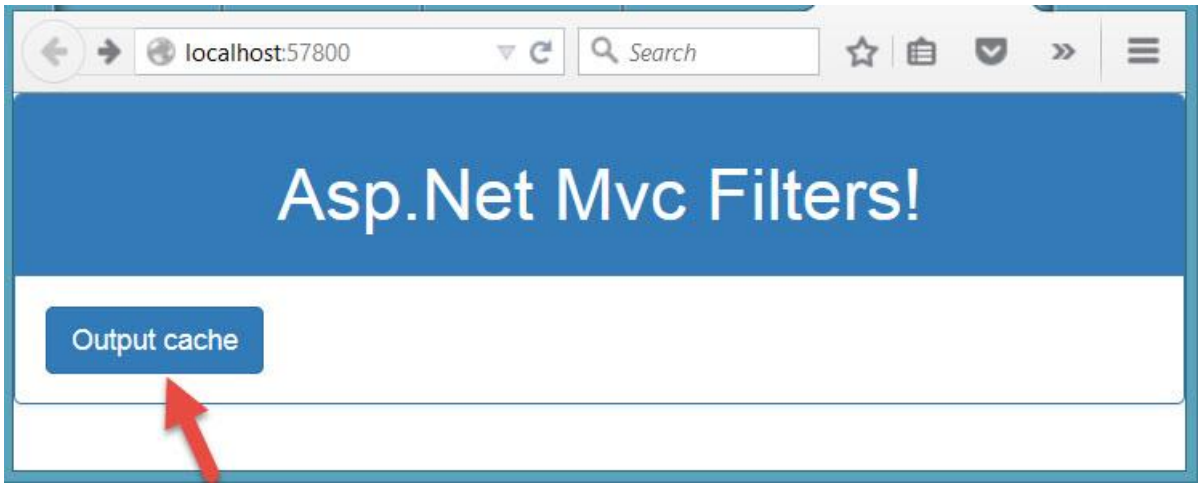
```
using System.Web;
using System.Web.Mvc;

namespace mkianiir.mvc.Filters.Controllers
{
    public class ActionFilterController : Controller
    {
        //
        // GET: /ActionFilter/
        [OutputCache(Duration=60)]
        public String Index()
        {

            return "The time is : "
+DateTime.Now.ToLongTimeString();
        }
    }
}
```

سپس فایل Index.cshtml را باز کنید و کد های آن را به صورت زیر تغییر دهید. دستورات زیر یک اکشن لینک به متد Index درون کنترلر ActionFilter ایجاد می کند.

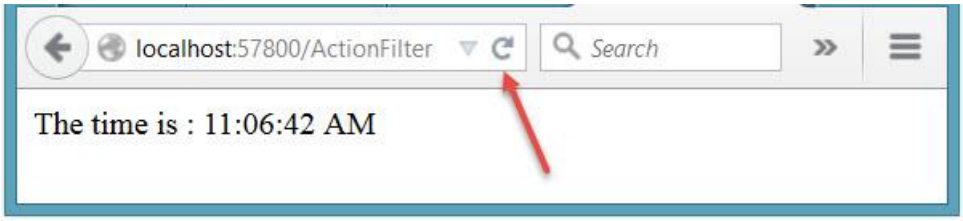
```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>Index</title>
</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h1 class="text-center">
                Asp.Net Mvc Filters!
            </h1>
        </div>
        <div class="panel-body">
            @Html.ActionLink("Output cache", "Index",
"ActionFilter", null, new
       {
           @class="btn btn-primary"
```

```
        })
         </div>
    </div>
</body>
</html>
```

حال برنامه را اجرا کنید و بر روی لینک Output cacheکلیک کنید.



شکل ۵ - ۲۰



شکل ۵ - ۲۱

حال اگر صفحه ActionFilter را رفرش کنید. مشاهده خواهید کرد که تا یک دقیقه خروجی تغییری نخواهد کرد. این بدان علت است که اکشن متد Index درون کنترلر ActionFilter دارای فیلتر OutputCache با

دوره زمانی ۱ دقیقه(۶۰ ثانیه) می باشد.این بدان معناست که خروجی متد Index برای یک دقیقه کش خواهد شد و اگر در عرض کمتر از یک دقیقه درخواست های مکرر به این متد ایجاد شود همان خروجی اولیه را تولید خواهد کرد.

مدیریت استثنا ها با استفاده از فیلتر HandleError

با استثنا ها در برنامه یقینا آشنایی دارید. یکی از فیلتر هایی که می تواند در این زمینه کمک شایانی به مدیریت استثنا ها بکند فیلتری به نام HandleError می باشد. این فیلتر از دسته Exception Filter ها بوده و برای مدیریت استثناهایی که در برنامه ممکن است با آن ها برخورد کنیم به کار می رود. در این قسمت می خواهیم نحوه استفاده از این فیلتر را بیان کنیم.

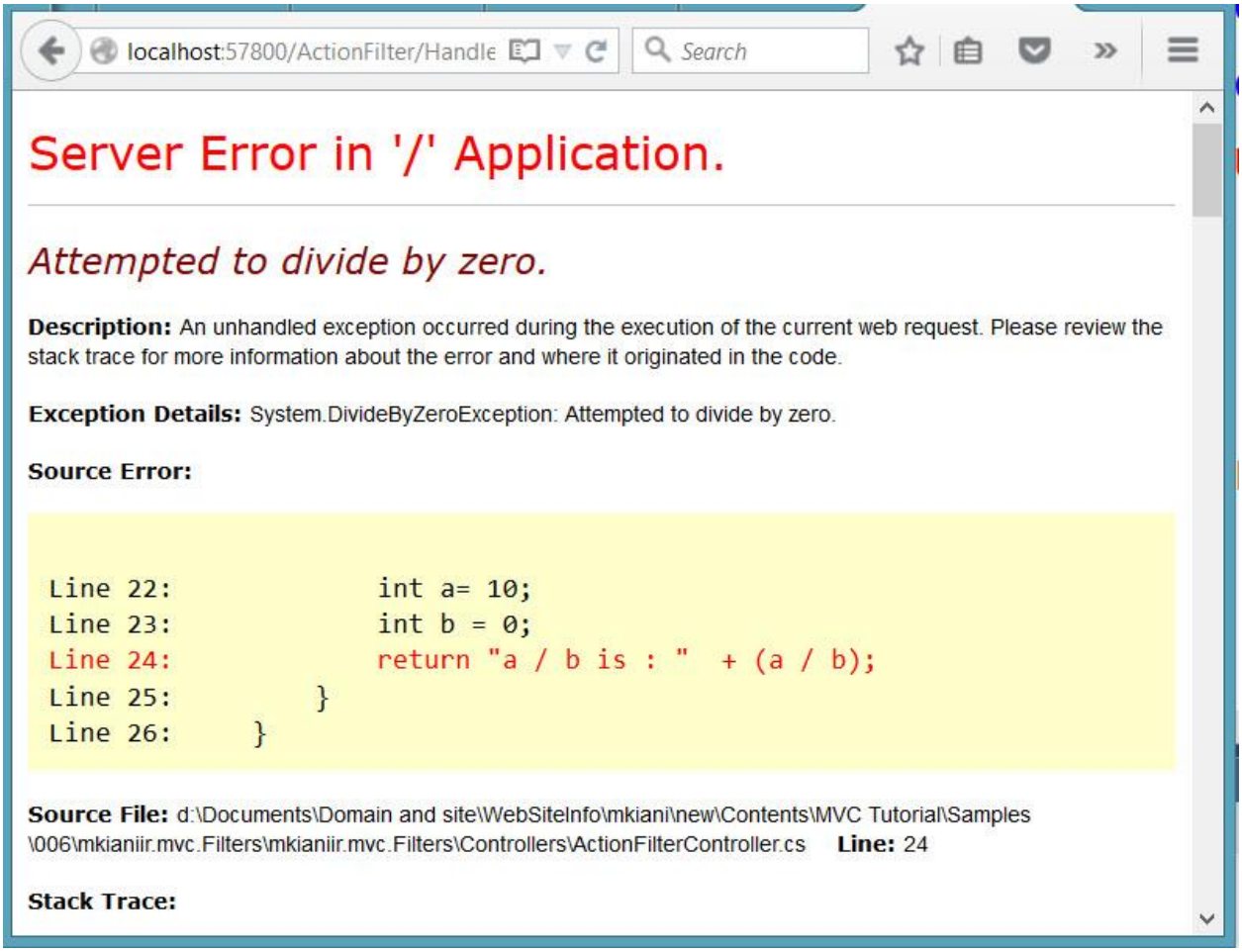
برای این منظور کلاس ActionFilterController را بازکنید و متد زیررا به آن اضافه کنید:

```
public String HandleErrorAction()
        {
            int a= 10;
            int b = 0;
            return "a / b is : "  + (a / b);
        }
```

فایل Index.cshtml را باز کنید و دستورات زیر را به آن اضافه کنید :

```
@Html.ActionLink("Handle error", "HandleErrorAction",
"ActionFilter", null, new
       {
           @class="btn btn-success"
       })
```

دستور فوق یک لینک به متد HandleErrorAction در کنترلر ActionFilter ایجاد می کند. حال برنامه را اجرا کنید و بر روی لینک Handle Error کلیک کنید.



شکل ۵ - ۲۲

همانطور که مشاهده می کنید استثنایی به نام DivideByZeroException رخ داده است که متاسفانه مدیریت نشده است. یکی از روش های مدیریت استثنا ها استفاده از صفت HandleError می باشد.

متد HandleErrorAction را به صورت زیر تغییر دهید:

```
[HandleError(View = "GeneralError")]
   public String HandleErrorAction()
   {
       int a = 10;
       int b = 0;
       return "a / b is : " + (a / b);
   }
```

همانطور که مشاهده می کنید صفت HandleError برای متد مذکور استفاده شده است. خاصیت View در کلاس HandleErrorAttribute نام ویویی را مشخص می کند که در صورت وجود استثنا mvc به آن رجوع خواهد کرد.

در پوشه Views در Solution Explorer یک پوشه به نام Shared ایجاد کنید. سپس یک ویو با نام GeneralError.cshtml در آن ایجاد کنید و کد های آن را به شکل زیر تغییر دهید:

```
@model System.Web.Mvc.HandleErrorInfo
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>GeneralError</title>
</head>
<body>
    <div class="panel panel-warning">
        <div class="panel-heading">
            <h1>Sorry some error occured</h1>
        </div>

        <div class="panel-body">
            Error occured in '<b>@Model.ActionName'</b> action
method inside '<b>@Model.ControllerName</b>' controller.
            <br />
            Error Message :<b> @Model.Exception.Message</b>
        </div>

    </div>
</body>
</html>
```

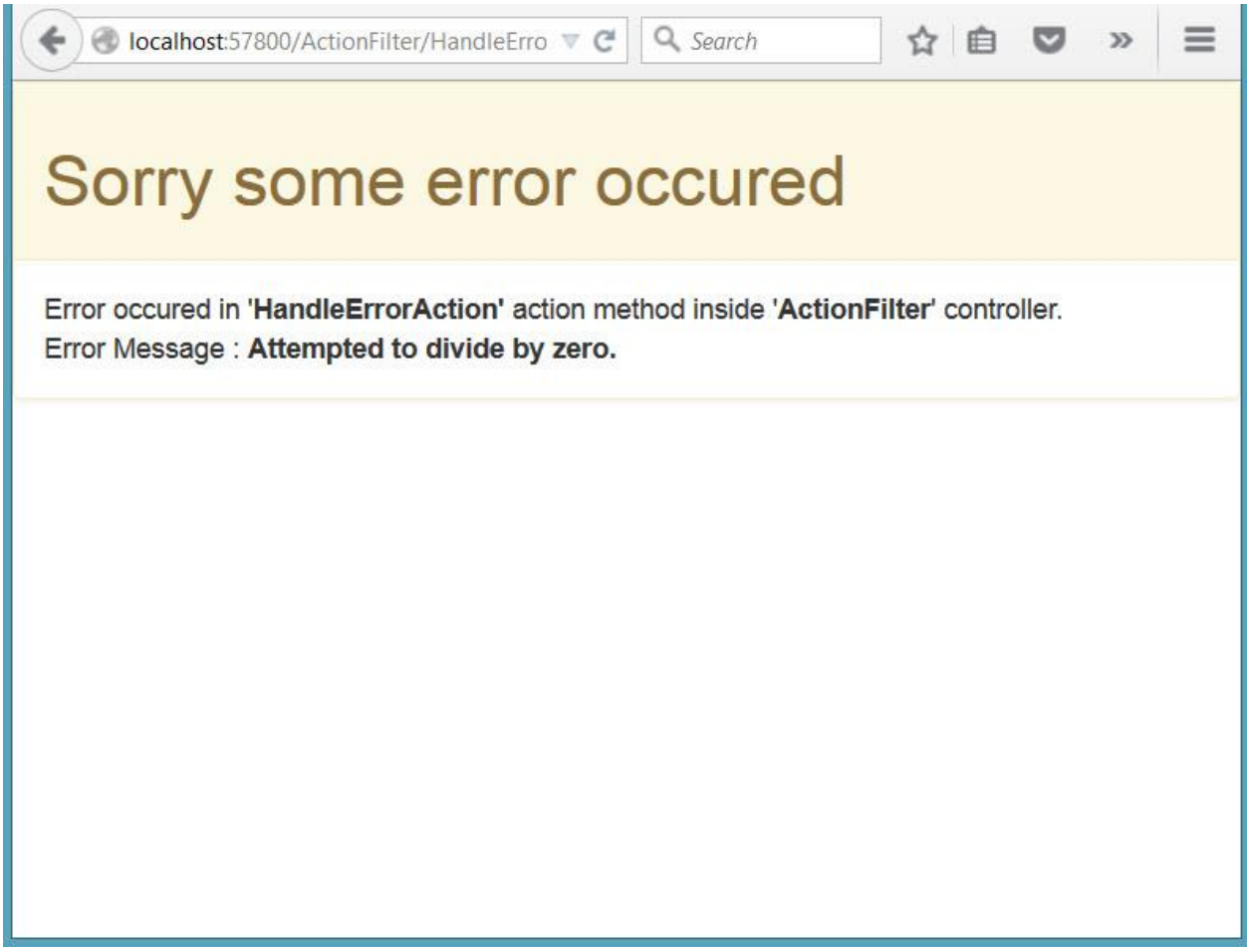
در خط اول فایل GeneralError.cshtml کلاس HandleErrorInfo از فضای نام System.Web.Mvc به عنوان مدل برای ویو در نظر گرفته شده است. صفت HandleError زمانی که به یک استثنا برخورد می کند نمونه ای از کلاس مذکور را مقدار دهی کرده و به View تعریف شده در HandleError ارسال خواهد کرد. این عمل توسط ViewData انجام می پذیرد. کلاس HandleErrorInfo اطلاعاتی راجع به استثنای رخ داده را در اختیار شما قرار می دهد.

نکته: زمانی که از HandleError استفاده می کنید می بایستی مقدار CustomError در فایل Web.config را فعال کنید.

برای این منظور فایل Web.config مربوط به پروژه را باز کنید و در تگ System.Web کد های زیر را قرار دهید:

```
<customErrors mode="On"/>
```

حال برنامه را مجددا اجرا کنید و روی دکمه Handle error کلیک کنید.



همانطور که مشاهده می کنید هنگام ایجاد استثنا برای متد مذکور برنامه محتویات استثنای رخ داده اعم از نام اکشن متد و همچنین کنترلری که استثنا در آن ایجاد شده است را به GeneralError.cshtml ارسال کرده و خروجی مورد نظر را ایجاد کرده است.

صفت HandleError دارای خاصیتی به نام ExceptionType می باشد.توسط این خاصیت می تواند برای انواع مختلف استثناهایی که ممکن است رخ دهد ویو ها ی مختلف را لود نمائید.

به عنوان مثا ل می توان متد HandleErrorAction را به صورت زیر تغییر داد:

```
  [HandleError(ExceptionType=typeof(System.FormatException), View =
"FormatError")]
```

```
        [HandleError(ExceptionType =
typeof(System.DivideByZeroException), View = "GeneralError")]

        public String HandleErrorAction()
        {
            int a = 10;
            int b = 0;
            return "a / b is : " + (a / b);
        }
```

همانطور که مشاهده می کنید صفت HandleError دو بار و هر بار با یک استثنای خاص تعریف شده است. در این حالت اگر استثنای فرمت ورودی اطلاعات رخ دهد آنگاه ویویی به نام FormatError و در صورتی که خطای DivideByZero اتفاق بیافتد ویویی به نام GeneralError نمایش داده خواهد شد. بدیهی است در این حالت فقط دو استثنای تعریف شده رصد خواهند شد.

نکته: اگر خاصیت ExceptionType مقدار دهی نشود آنگاه تمامی استثنا ها رصد خواهند شد.

ایجاد یک اکشن فیلتر سفارشی

در بخش قبلی با فیلتر های اولیه و نحوه استفاده از آن ها آشنا شدید. همانطور که اشاره شد در MVC این امکان وجود دارد تا بتوانید اکشن فیلتر های سفارشی نیز تولید کنید. در این بخش یک اکشن فیلتر سفارشی ایجاد خواهیم کرد. برای این منظور می بایستی یک کلاس از کلاس ActionFilterAttribute مشتق نمائیم.

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

namespace mkianiir.mvc.Filters
{
    public class CustomFilterAttribute :ActionFilterAttribute
    {
        public override void
OnResultExecuting(ResultExecutingContext filterContext)
        {
            filterContext.HttpContext.Response.Write("<b>This is
custom filter, executed befor action method execution</b>");
            base.OnResultExecuting(filterContext);
```

```
        }
        public override void OnResultExecuted(ResultExecutedContext
filterContext)
        {
            filterContext.HttpContext.Response.Write("<b>This is
custom filter, executed after action method execution<b/>");
            base.OnResultExecuted(filterContext);
        }
    }
}
```

کلاس ActionFilter برای پیاده سازی اکشن فیلتر های سفارشی به کار می رود. این کلاس دارای چهار متد OnActionExecuting ، OnActionExecuted ، OnResultExecuting و OnResultExecuted می باشد که می توانید آن ها را دوباره نویسی (Override) کنید.

همانطور که مشاهده می کنید در دستورات فوق دومتد مربوط Result دوباره نویسی شده اند. متد OnResultExecuting قبل از اجرا شدن خروجی اکشن متد(ActionResult) و متد OnResultExecuted پس از اجرا شدن آن اجرا می شوند.

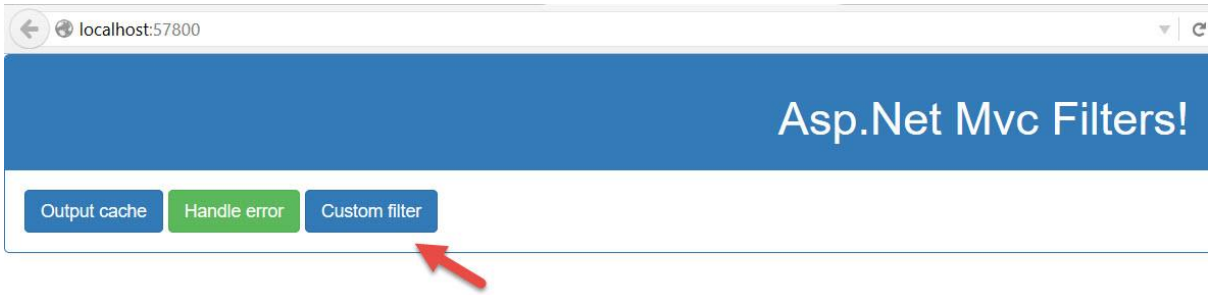
حال یک متد به نام CustomFilter در کنترلر ActionFilterController به صورت زیر تعریف کنید:

```
        [CustomFilter]
        public string CustomFilter()
        {
            return "<br/>This is action method Result<br/>";
        }
```

حال در فایل Index.cshtml یک اکشن لینک به اکشن متد جدید به صورت زیر تعریف کنید:

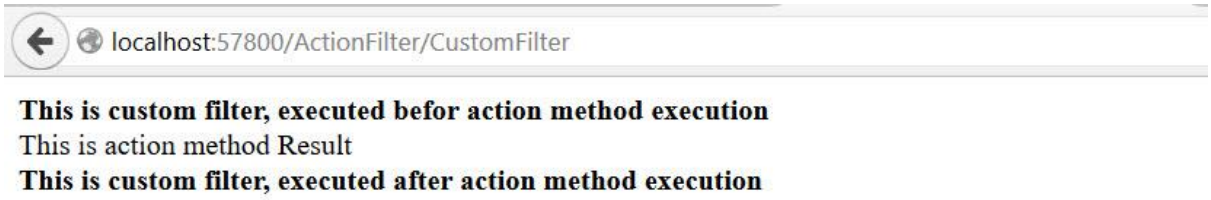
```
@Html.ActionLink("Custom filter", "CustomFilter", "ActionFilter",
null, new
        {
            @class = "btn btn-primary"
        })
```

سپس برنامه را اجرا و روی اکشن لینک جدید کلیک نمائید:



شکل ۵ - ۲۳

اگر همه کارها را به درستی انجام داده باشید خروجی می بایستی شبیه به شکل زیر باشد:



شکل ۵ - ۲۴

همانطور که مشاهده می کنید قبل از اجرا شدن ActionResult (مهیا شدن خروجی اکشن متد) متد OnResultExecuting و پس از آن متد OnResultExecuted اجرا شده است.

خلاصه

در این فصل با جزئیات بیشتری در رابطه با اکشن متد ها آشنا شدید. ارسال پارامتر به آن ها، ارسال پارامتر از آن ها به ویو های مورد نظر و ...

همچنین در بخش پایانی این فصل مبحث جدیدی به نام فیلتر ها مورد بررسی قرار گرفتند. همانطور که بیان شد فیلتر ها صفاتی هستند که با استفاده از آن ها می توان کنترلر بیشتری بر روی پردازش درخواست ها داشت. در این فصل علاوه بر معرفی فیلترهای اولیه آموختید که چگونه یک فیلتر سفارشی نیز ایجاد کنید. در فصل بعد که فصل پایانی کتاب نیز می باشد در رابطه با ویو ها و متد های راهنما صحبت خواهیم کرد.

فصل ششم:ویو ها و متد های راهنما

مقدمه

این فصل به عنوان فصل پایانی کتاب به جزئیات بیشتر در رابطه با ویو ها و همچنین متد های راهنمایی که برخی از آن ها را در فصل های قبلی استفاده کردید خواهد پرداخت. انتظار می رود در پایان این فصل موارد زیر را به خوبی درک نمائید:

۱- ویو ها و ساختار آنها
۲- انواع ویو ها
۳- استفاده از Layout ها
۴- PartialView ها
۵- ارسال اطلاعات به ویو ها
۶- بایند کردن مدل داده ای به ویوها
۷- متد های راهنما و انواع آن ها
۸- ایجاد متد های راهنمای شخصی

ویو چیست و ساختار ویو ها در mvc چگونه است؟

همانطورکه در معرفی معماری mvc بیان شد یکی از قسمت های مهم این معماری ویو ها می باشند. ویو در mvc وظیفه تولید واسط کاربری برنامه را بر عهده دارد.کاربر از طریق ویو ها می تواند اقدام به ارسال اطلاعات به سمت سرور کرده و توسط آن ها نتایج حاصله را دریافت نماید.چیزی که در مورد ویو ها در معماری mvc اهمیت دارد این است که ویو ها هیچ اطلاعای از داده هایی که به کاربر نمایش می دهند ندارند(همچنین از نحوه تولید آن ها).در واقع ویو ها نمی دانند که داده ها از کجا و به چه شکلی تولید می شوند، آیا اطلاعات از فایل خوانده می شوند؟ آیا از پایگاه داده خوانده می شوند؟نوع پایگاه داده یا فایل ها چگونه است؟نحوه ذخیره سازی

اطلاعات چگونه است؟ این ها سوالاتی است که ویو نمی تواند و نباید پاسخی به آن ها بدهد. ویو ها اطلاعاتی را که توسط کنترلر به آن داده می شود(در قالب مدل داده ای) به کاربر نشان می دهند. این مدل داده ای می تواند یک مقدار ساده نظیر یک رشته اطلاعات یا یک نمونه از یک کلاس به عنوان مدل داده ای باشد.

ویو ها در mvc را می توان متناظر با Form ها در Windows Forms Application و Web Form ها در Asp.Net Web Forms Application دانست. تفاوت عمده در خصوص ویو های mvc با فرم ها و یا وب فرم ها این است که در فرم ها یا وب فرم ها غالبا بخشی از واسط کاربری با منطق برنامه تجمیع شده است(وابستگی دارند) در صورتی که ساختار mvc مخالف این وابستگی بوده و نباید چنین وابستگی هایی در یک برنامه mvc اتفاق بیفتد.

انواع ویو در MVC

نسخه های اولیه mvc از فایل های aspx (همان وب فرم ها) برای ایجاد ویو ها استفاده می کرد. در نسخه ۳ از mvc علاوه بر وب فرم های aspx موتور جدید برای ویو ها معرفی شد به نام Razor . فایل هایی که تحت این موتور کار می کنند دارای پسوند cshtml (برای زبان سی شارپ) و vbhtml (برای زبان ویژوال بیسیک) می باشند. در ادامه مباحث آموزشی mvc از موتور Razor استفاده خواهیم کرد.(از این پس هر کجا در مورد ویو ها صحبت شد منظور فایل های موتور Razor که دارای پسوند cshtml هستند می باشد)

دستورات فایل های ویو شامل دو بخش می باشند. بخش کد های html و بخش کد های زبان برنامه نویسی(در اینجا زبان سی شارپ). دستورات برنامه نویسی با کاراکتر @ شروع می شوند و نیازی به درج کاراکتر سمی کلن(;) در پایان دستورات نمی باشد. موتور Razor در زمان اجرای برنامه تشخیص خواهد داد که کدام دستورات مربوط به زبان برنامه نویسی و کدامیک مربوط به دستورات html می باشند. زمانی که برنامه کمپایل می شود Mvc فایل های ویو را تبدیل به کلاس های متناظر خود کرده و آن ها را کمپایل می کند که از ترکیب دستورات زبان برنامه نویسی و کد های html خروجی نهایی را تولید می کند.

ساختار پوشه بندی ویو ها در پروژه های mvc

زمانی که یک پروژه Mvc ایجاد می کنید یک پوشه به نام Views درون پنجره Solution Explorer ایجاد می گردد.ویو های مربوط به هر کنترلر درون پوشه ای هم نام با کنترلر که درون پوشه Views ایجاد می شوند قرار خواهد گرفت.

آدرس زیر را در نظر بگیرید:

http://localhost/Home/Index

زمانی که mvc درخواستی را با آدرس فوق در یافت می کند(با فرض اینکه در اکشن متد Index دستور return View(); قرار گرفته باشد) به دنبال ویویی به نام Index در مسیر های زیر خواهد گشت:

~/Views/Home/Index.cshtml

~/Views/Home/Index.vbhtml

~/Views/Shared/Index.cshtml

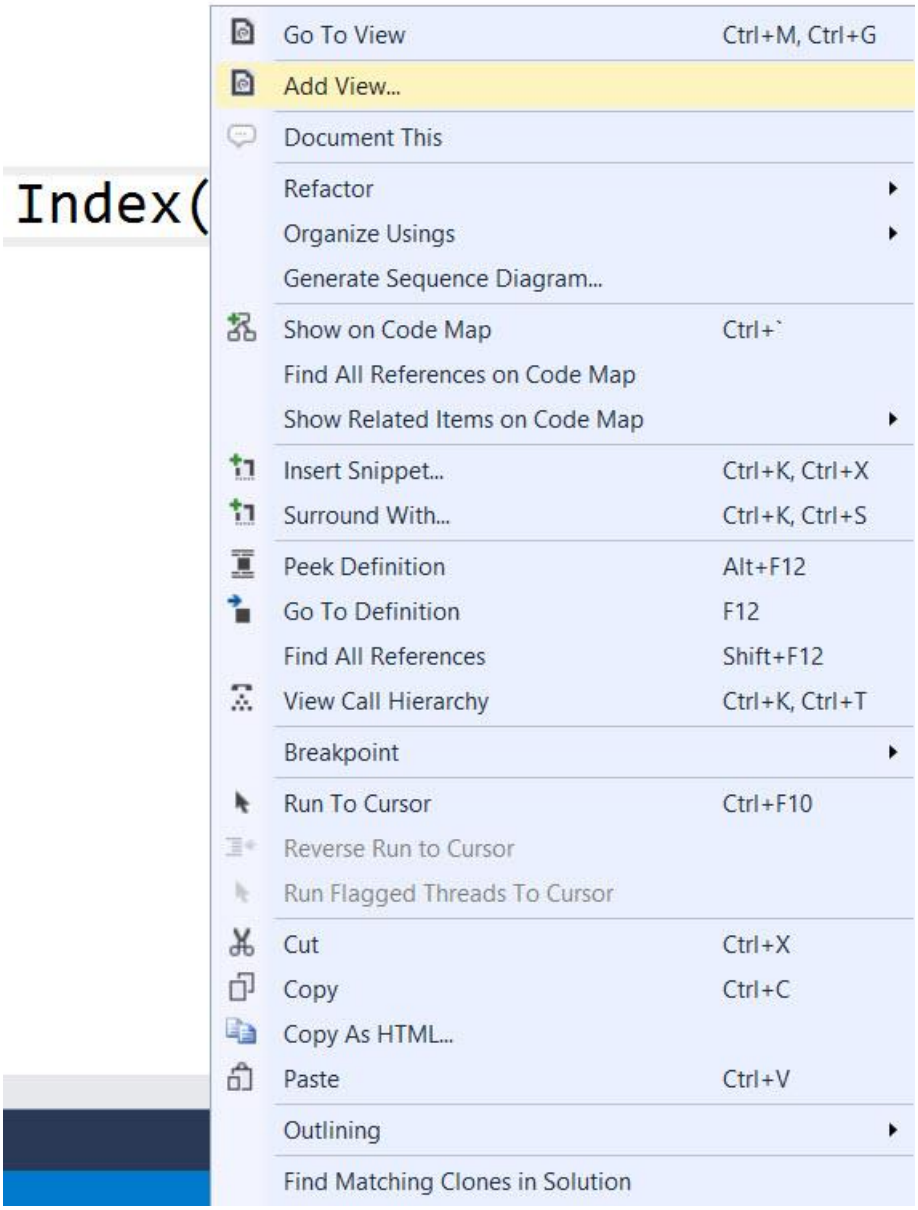
~/Views/Shared/Index.vbhtml

در مسیر های فوق، عبارات Index.cshtml و Index.vbhtml نام ویوی مورد نظر است. کلمه Home بیانگر کنترلر Home می باشد که در آدرس مورد درخواست کاربر آمده است. پوشه Shared محلی برای ذخیره ویو های اشتراکی است. همانطور که از معنای نام پوشه مشخص است این پوشه محل نگهداری ویو هایی است که مربوط به یک کنترلر خاص نبوده(عموما Partial Viewها) و در مکان های مختلف می توانند مورد استفاده قرار گیرند. ویو هایی که به عنوان Layout به کار می روند نیزعموما در این پوشه قرار می گیرند.

اجازه دهید طبق روال مباحث را با ایجاد یک پروژه آغاز نمائیم:

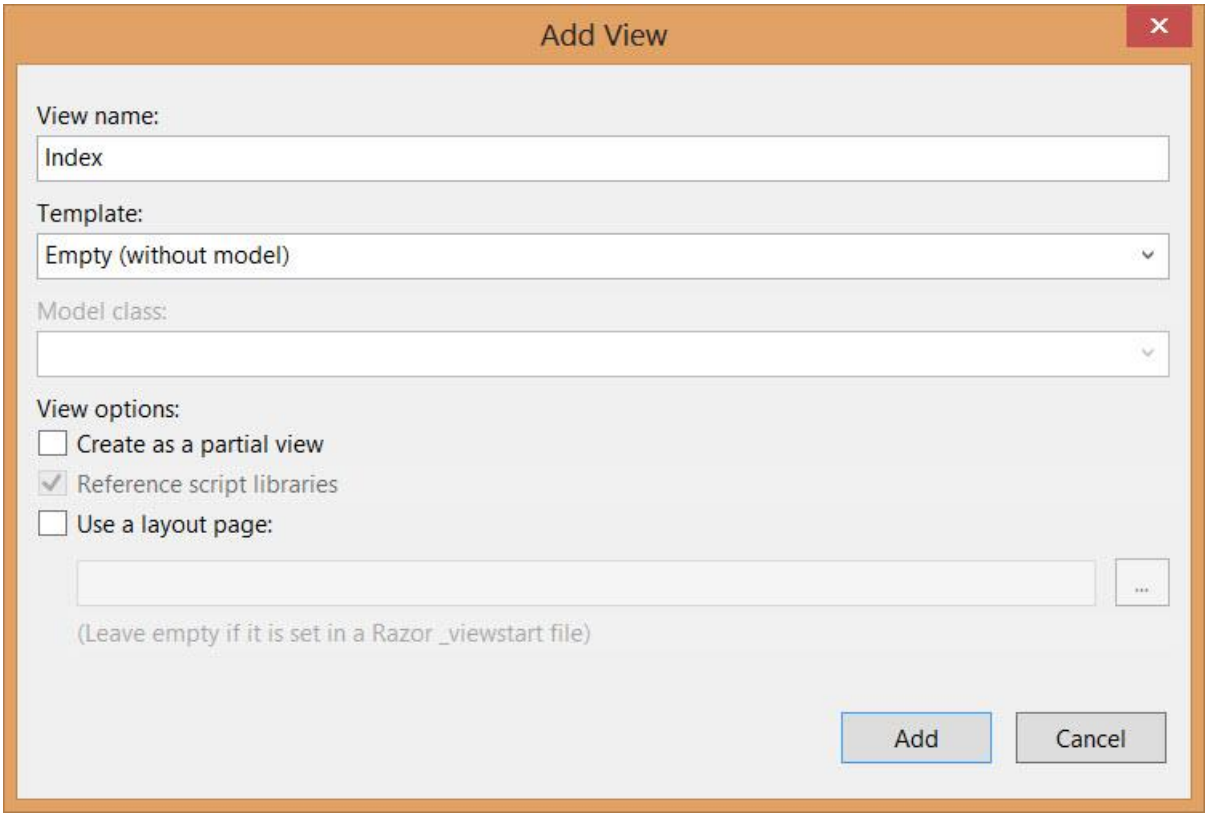
یک پروژه جدید ایجاد کنید (من نام آن را mkianiir.nvc.VAHM گذاشتم. کلمه VAHM مخفف Views And Helper Methods می باشد)و یک کنترلر به نام HomeController به پروژه اضافه نمائید.

حال بر روی نام متد Index در کنترلر Home کلیک راست کنید و از منوی باز شده گزینه Add View را کلیک کنید.



شکل ۶ - ۱

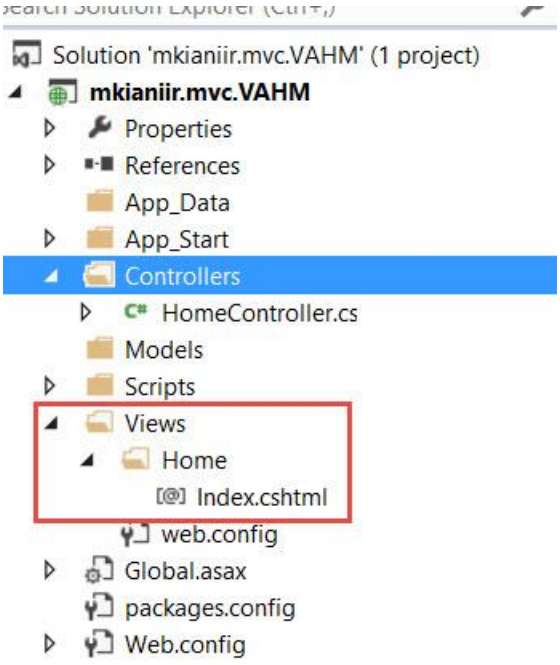
پنجره Add View باز می شود.



شکل ۶ - ۲

مقادیر پنجره Add View را مانند شکل فوق تنظیم کنید و بر روی دکمه Add کلیک کنید. نگران آیتم های درون این پنجره نباشید. در ادامه این فصل توضیح داده خواهند شد.

پس از فشرده شدن دکمه Add پنجره Solution Explorer را باز کنید.



شکل ۶ - ۳

همانطور که مشاهده می کنید در پوشه Views یک پوشه به نام Home و در داخل پوشه Home فایلی با نام Index.cshtml ایجاد شده است. نام پوشه Home از نام کنترلری که ویوی Index برای آن ایجاد شده است گرفته شده است. همانطور که پیشتر گفته شد برای هر کنترلر یک پوشه هم نام با آن کنترلر در ویو مورد نیاز است.

فایل Index.cshtml ویو متناظر با اکشن متد Index درون کنترلر Home را نشان می دهد. دستورات پیش فرض این فایل به صورت زیر است:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

سه خط اول این دستورات یعنی

```
@{
    Layout = null;
}
```

به این معناست که ویوی Index از هیچ ویوی دیگری به عنوان Layout استفاده نمی کند. Layout ها که در ادامه همین بخش به آن ها خواهیم پرداخت نقشی شبیه به Master Page ها در Asp.Net Web Forms یا موتور aspx در Mvc بازی می کنند.

سایر دستوراتی که در فایل Index.cshtml مشاهده می کنید کد های ساده و ابتدایی یک فایل html ساده می باشند که نیاز به توضیح خاصی ندارند.

ارسال اطلاعات از کنترلر به ویو

در mvc به روش های متفاوتی می توان اطلاعات را از کنترلر به ویو ارسال کرد. در بخش های قبلی استفاده از ViewBag را در این زمینه تجربه کردید.

دو ابزار دیگر برای ارسال داده ها از کنترلر به ویو استفاده از ViewData و TempData می باشد.

استفاده از ViewData

در کلاس ControllerBase خاصیتی به نام ViewData از جنس ViewDataDictionary تعریف شده است. پس بنابر این ViewData یک دیکشنری است و همانطور که می دانید داده های دیکشنری ها داری دو جزء می باشند. یکی کلید و دیگری داده. کلید های دیکشنری ViewData از جنس String می باشند.

دستورات متد Index در کنترلر Home را به صورت زیر تغییر دهید:

```
public ActionResult Index()
        {
            ViewData["Name"] = "Mehdi Kiani";
```

```
            ViewData["Date"] = DateTime.Now;
            return View();
        }
```

فایل Index.cshtml را باز کنید و دستورات آن را مطابق زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        Hello  <b>'@ViewData["Name"]'</b>
        <br />
        The time is : <b>'@ViewData["Date"]'</b>
        <br />
        Good @if (((DateTime)ViewData["Date"]).Hour < 12)
        {
            <b>'Morning'</b>
        }
        else
        {
            <b>'Afternoon'</b>
        }

    </div>
</body>
</html>
```

همانطور که در دستورات فوق مشاهده می کنید مقدیر ViewData نیاز به عمل Casting دارند.

خروجی دستورات فوق به صورت شکل زیر می باشد:

شکل ۶ - ۴

نکته: مقادیر ViewData و ViewBag صرفا برای درخواست جاری معتبر می باشند. یعنی با تغییر درخواست(عملیات Redirect) مقادیر آن ها به null تبدیل می شوند و ازدرجه اعتبار ساقط می گردند.

استفاده از TempData

یکی دیگر از ابزار های انتقال اطلاعات بین کنترلر و ویو استفاده از TempData می باشد. خاصیت TempData نمونه ای از کلاس TempDataDictionary می باشد. روش استفاده از این خاصیت نیز شبیه به خاصیت ViewData می باشد که در بخش قبلی مشاهده کردید.تفاوت TempData با ViewData و ViewBag در این است که مقادیر TempData بین درخواست های مختلف(عملیات Redirect) نیز معتبر می باشند. چون روش مقدار دهی و استفاده از آن شبیه به ViewData و ViewBag می باشد نیازی به توضیح اضافه تر در این زمینه نیست.

استفاده از مدل جهت ارسال اطلاعات از کنترلر به ویو

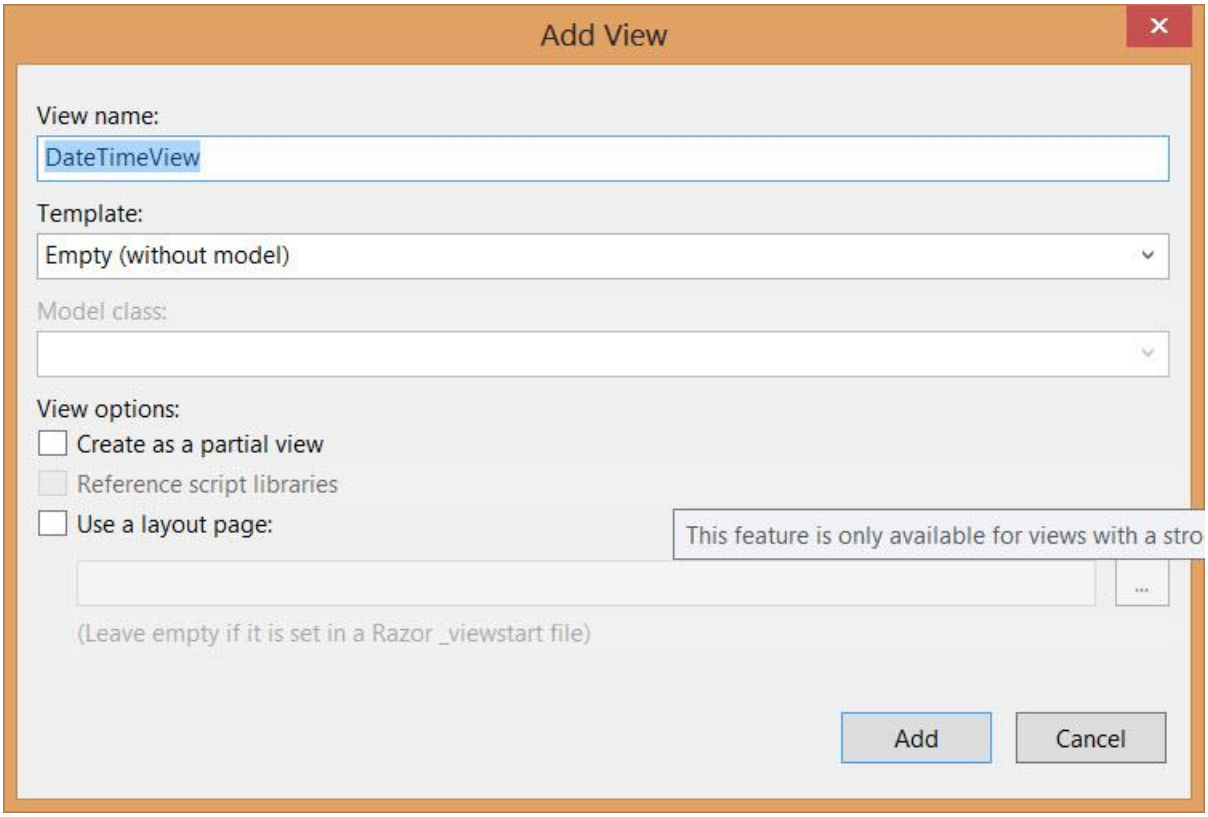
یکی دیگر از روش های ارسال اطلاعات از کنترلر به ویو استفاده از مدل داده ای می باشد. نحوه استفاده از این روش را هنگام ایجاد فرم Contact در بخش های قبلی مشاهده کردید. در این قسمت لازم است این نکته بیان شود که هر نوعی می تواند به عنوان داده به ویو ارسال شود. منظور از مدل داده ای شی است که به عنوان مدل برای ویودر نظر گرفته می شود. این شی می تواند یک نوع اولیه شبیه DateTime یا String باشد و یا می تواند یک کلاس و یا لیستی از کلاس ها باشد.

برای مثال یک اکشن متد به نام DateTimeView به کنترلر Home اضافه کنید. و دستورات آن را به صورت زیر تغییر دهید:

```
public ViewResult DateTimeView()
        {
            return View(DateTime.Now);
        }
```

همانطور که مشاهده می کنید مقدار DateTime.Now که از جنس DateTime بوده و تاریخ فعلی سرور را نشان می دهد به عنوان مدل به ویوی DateTimeView ارسال شده است.

سپس بر رو نام متد فوق کلیک راست کره و منوی Add View را انتخاب کنید.تنظیمات را مانند شکل زیر انجام داده و بر روی دکمه Add کلیک کنید.



شکل ۶ - ۵

با این کار یک ویو به نام DateTimeView در پوشه Home درون پوشه Views ایجاد می شود. بر روی فایل DateTimeView.cshtml دو بار کلیک کنیدو دستورات آن را طبق زیر تغییر دهید:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>DateTimeView</title>
</head>
<body>
    <div>
      Today  is : '<b>@(((DateTime)Model).DayOfWeek)
@(((DateTime)Model).ToShortDateString())</b>'
    </div>
</body>
</html>
```

در ویوی فوق مدل داده ای را به نوع DateTim تبدیل کرده ایم(عمل Casting) تا بتوانیم از خواص و متد های آن استفاده کنیم.

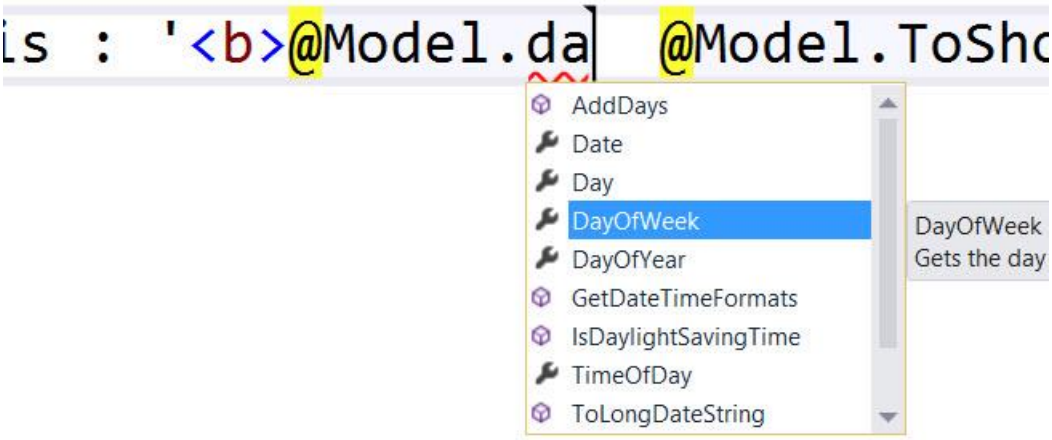
علاوه بر روش فوق می توانید با استفاده از دستور @model ، مدل داده ای ویو را به آن معرفی کنید تا در استفاده از مدل ویژوال استویو به کمک شما بیاید و از اشتباهات نحوی شما را رها سازد.

دستورات ویوی فوق را به صورت زیر تغییر دهید.

```
@model DateTime
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>DateTimeView</title>
</head>
<body>
```

```
    <div>
      Today  is : '<b>@Model.DayOfWeek  
@Model.ToShortDateString()</b>'
    </div>
</body>
</html>
```

در خط اول ویو، نوع مدل را از جنس DateTime معرفی کردیم. بنابر این با استفاده از دستور @Model می توانیم به اعضای آن دسترسی پیدا کنیم و دیگر نیازی به عملیات Casting نمی باشد. علاوه بر این زمانی که بعد از دستور @Model کاراکتر dot را تایپ کنید، ویژوال استودیو لیستی از اعضای مدل داده ای شما که در اینجا شی DateTime می باشد را نشان می دهد.(Strongly Typed)



شکل ۶ - ۶

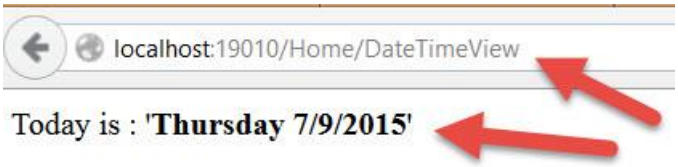
حال فایل Index.cshtml را باز کرده و اکشن لینک زیر را برای اشاره به اکشن متد DateTimeView به آن اضافه کنید:

```
@Html.ActionLink("Goto datetime view", "DateTimeView")
```

سپس برنامه را اجرا و بر روی لینک goto datetime view کلیک کنید.

شکل ۶ - ۷

خروجی در شکل زیر نشان داده شده است.



شکل ۶ - ۸

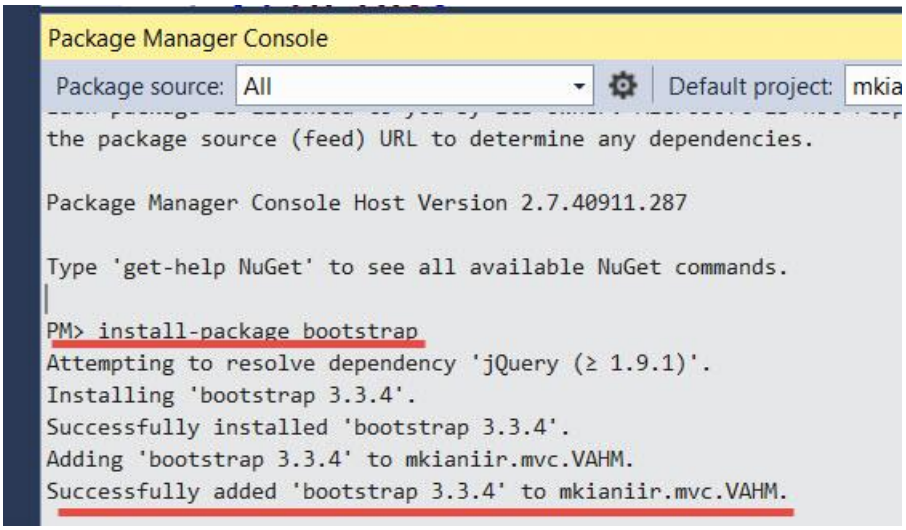
نکته: توجه داشته باشید برای معرفی مدل از دستور @model (m کوچک) و برای استفاده از مدل از @Model (m بزرگ) استفاده می کنیم.

استفاده از Layout در ویو ها

برنامه نویسان Asp.Net Web Form با مفهوم MasterPage ها آشنایی دارند. MasterPage ها صفحاتی هستند که از کلاس MasterPage مشتق شده اند. در این صفحات طراحی هایی که قرار است در یک یا چند صفحه(صفحاتی که از آن MasterPage بهره می برند) به صورت مشترک نمایش داده شوند قرار می گیرند.به عنوان مثال فرض کنید عکسی در تمامی صفحات شما به عنوان Header قرار است وجود داشته باشد. به جای اینکه عکس مذکور را در همه صفحات قرار دهید می توانید آن را در یک Master Page قرار دهید و صفحات دیگر از این MasterPage استفاده کنند. بدیهی است در این حالت میزان کدنویسی و طراحی کمتر شده و تغییر و نگهداری برنامه ساده تر خواهد شد. در MVC معادلی برای MasterPage ها به نام Layout وجود دارد. در واقع Layout ها ویوهایی هستند که می توانند به صورت اشتراکی بین یک یا چند ویو مورد استفاده

قرار گیرند.بهتر است Layout ها در پوشه Shared درون پوشه Views در Solution Explorer قرار گیرند تا از سایر ویو ها جدا شوند.

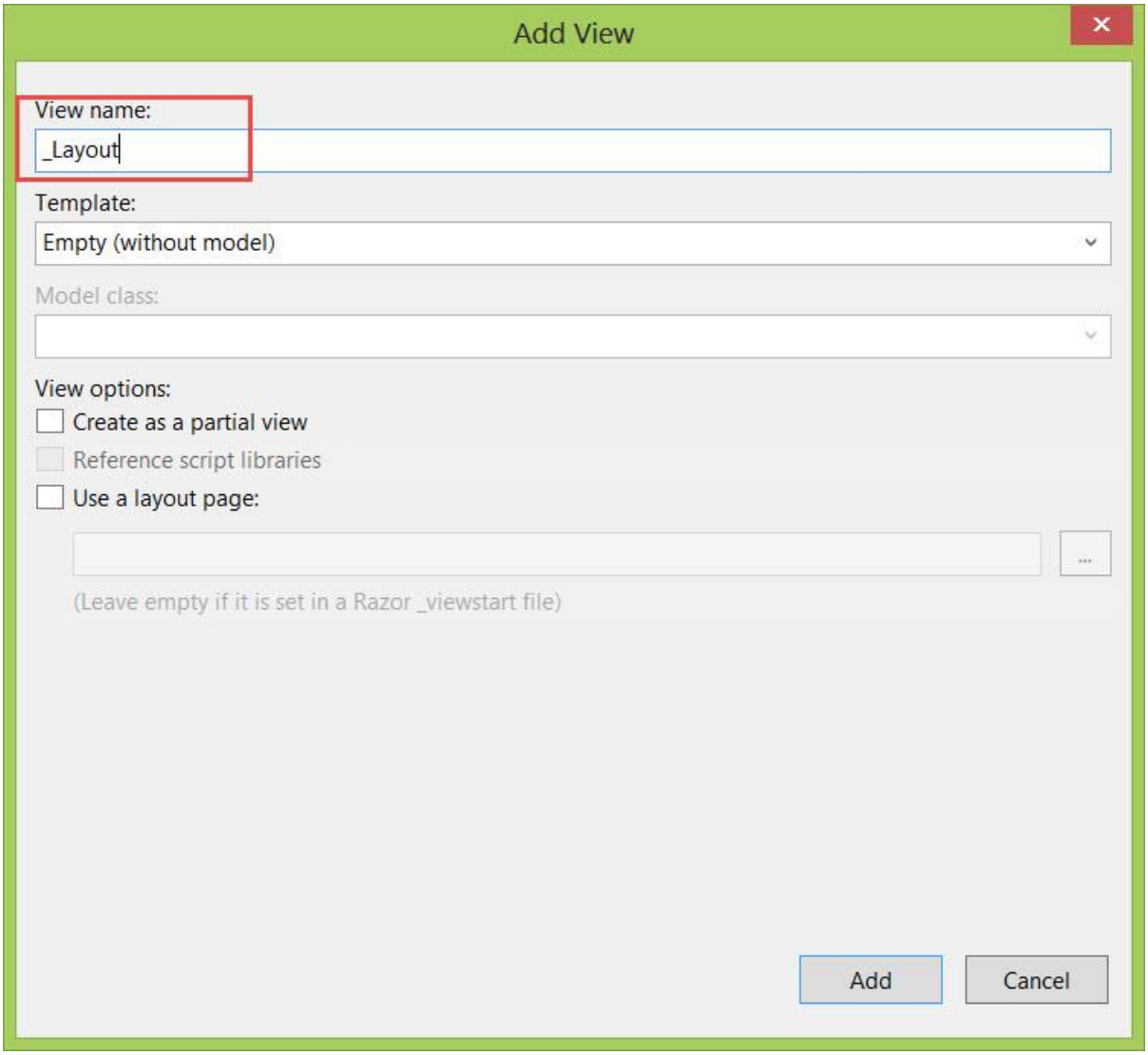
نکته: قبل از ادامه، کتابخانه Bootstrap را به پروژه خود اضافه کنید.



شکل ۶ - ۹

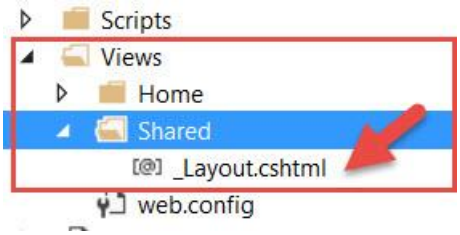
حال بر روی پوشه Views کلیک راست کرده و از منوی Add گزینه New Folder را کلیک کنید. نام پوشه ایجاد شده را Shared قرار دهید.

سپس بر روی نام پوشه Shared کلیک راست کنید و از گزینه Add گزینه View را کلیک کنید.



شکل ۶ - ۱۰

نام ویو را Layout_ قرار دهید و سایر تنظیمات را مانند شکل فوق انجام دهید و سپس دکمه Add را کلیک کنید. با این کار یک فایل به نام Layout.cshtml_ در پوشه Shared درون پوشه Views اضافه خواهد شد.



شکل ۶ - ۱۱

فایل Layout.cshtml_ را باز کنید و دستورات آن را به صور زیر تغییر دهید:

```
<!DOCTYPE html>
<html>
<head>

    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>@ViewBag.Title</title>

</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            Asp.Net mvc views and helper methods!
        </div>
        <div class="panel-body">
            @RenderBody()
        </div>
        <div class="panel-footer">
            thnk's for choosing mkiani.ir
        </div>
    </div>
</body>
</html>
```

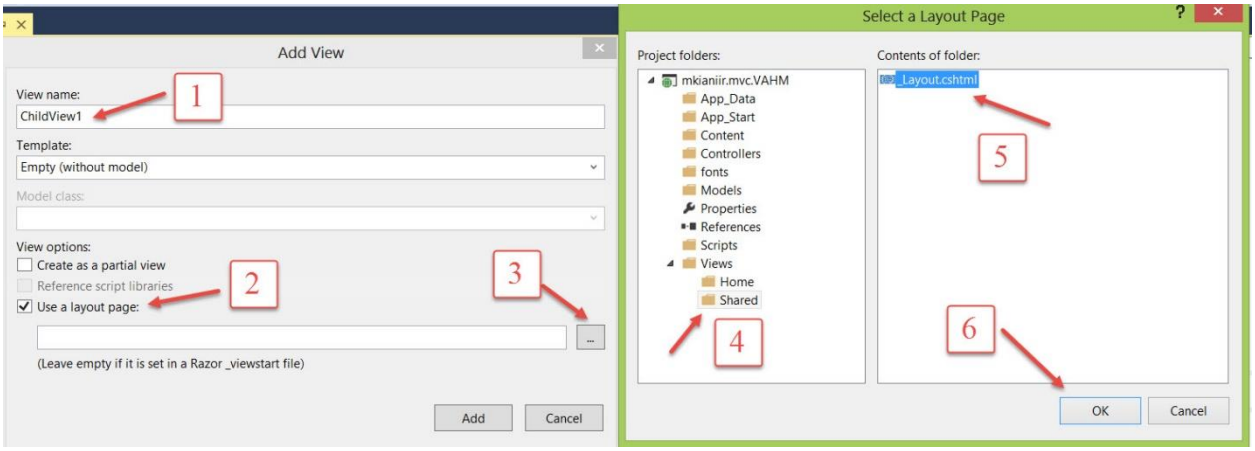
در دستورات فوق کتابخانه Bootstrap جهت زیبا سازی برنامه اضافه شده است. دستور @ViewBag.Title بدین منظور اضافه شده است که بتوان توسط آن عنوان هر ویو را به تگ title صفحه اضافه کرد. سایر دستورات همان دستورات آشنای html می باشند به جز دستور @RenderBody که در ادامه با عملکر آن آشنا خواهید شد.

فایل Layout_ را ذخیره و ببندید.

کلاس HomeController را باز کرده و اکشن متد زیر را به آن اضافه کنید:

```
public ViewResult ChildView1()
      {
          return View();
      }
```

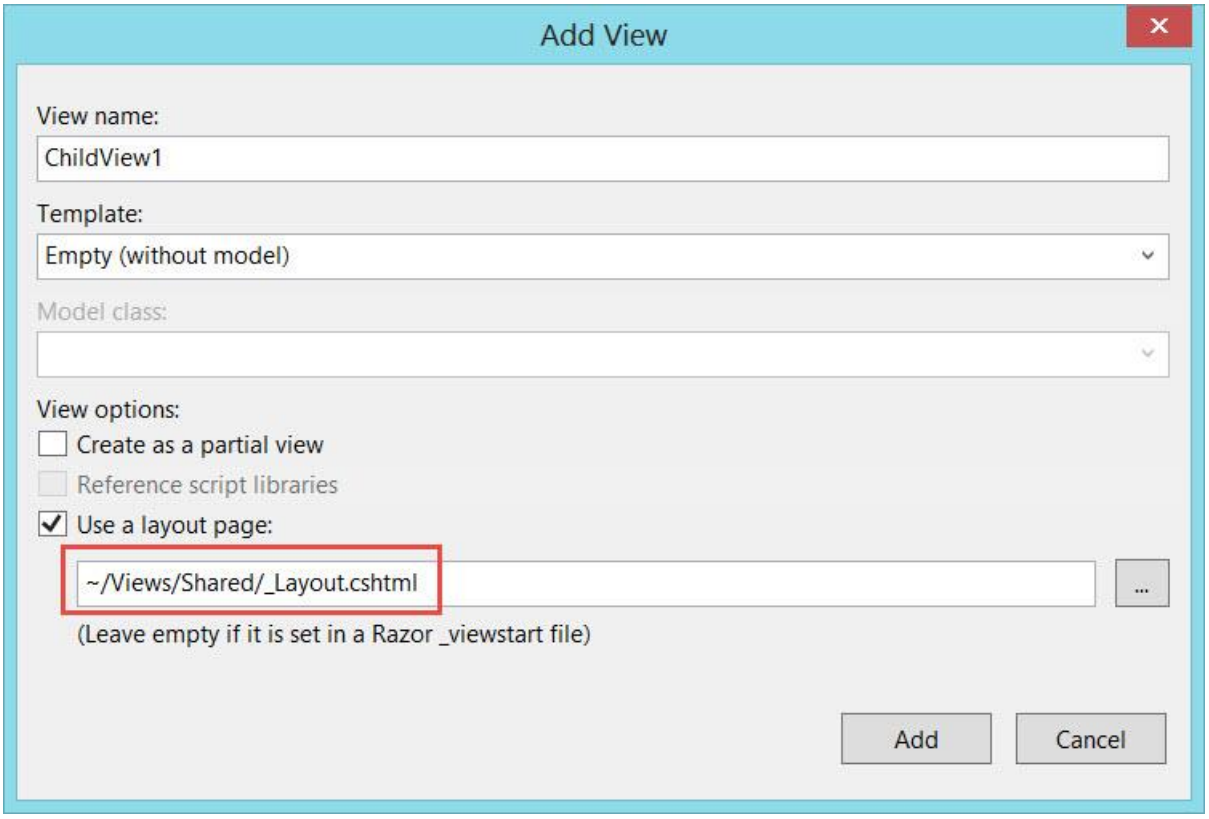
حال بر روی نام اکشن متد کلیک راست کرده گزینه Add View را انتخاب کنید.



شکل ۶ - ۱۲

نام ویو را ChildView1 قرار دهید. سایر تنظیمات را طبق مراحل ۲ تا ۶ تکمیل کنید.

پس از این کار در بخش Use a layout page عبارت ~/Views/Shared/_Layout.cshtml قرار خواهد گرفت.



شکل ۶ - ۱۳

سپس بر روی دکمه Add کلیک کنید. اگر مراحل را به درستی انجام داده باشید یک ویو به نام ChildView1 در پوشه Home در Views ایجاد می شود.

فایل ChildView1.cshtml را باز کنید. دستورات آن به صورت زیر خوهد بود:

```
@{
    ViewBag.Title = "ChildView1";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>ChildView1</h2>
```

چه اتفاقی افتاد؟ تگ های html و body و head و title کجا رفتند؟

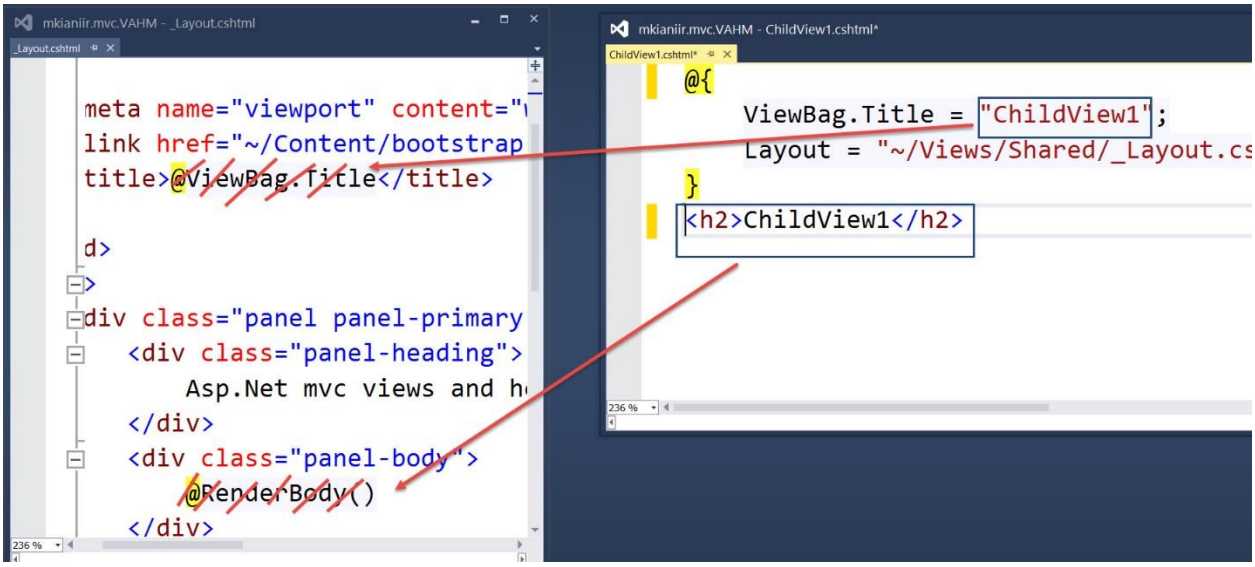
همانطور که مشاهده می کنید در خط سوم دستورات فوق دستور زیر قرار گرفته است:

```
Layout = "~/Views/Shared/_Layout.cshtml";
```

اگر خاطرتان باشد تاکنون هر ویویی که ایجاد کرده بودیم مقدار Layout آن null بود. بدین معنی که آن ویو از هیچ ویوی دیگری به عنوان layout استفاده نمی کرد. در اینجا چون ویوی ChildView1 از ویویی به نام _Layout در پوشه Shared واقع در پوشه Views به عنوان layout(همان Master در Asp.net web form) بهره می گیرد بنابر این مقدار آن طوری تنظیم شده است که به فایل _layout.cshtml اشاره کند و چون در ویوی _Layout تگ های head ، title، body و html تعریف شده اند دیگر نیازی به این تگ ها در ChildView1 نمی باشد.

اگر کمی به دستورات قبل تر بر گردید. آنجا که فایل _Layout.cshtml را توضیح دادیم. دو دستور ویژه علاوه بر سایر کد های html در آن قرار داشت. اولی دستور @ViewBag.Title در تگ title بود. اگر به دستورات ویوی ChildView1 دقت کنید مشاهده خواهد کرد که دستور ViewBag.Title برابر با مقدار رشته ای ChildView1 تنظیم شده است. در این حالت زمانی که برنامه اجرا می شود و ویوی ChildView1 مورد ارجاع قرار می گیرد مقدار ChildView1 به جای دستور @ViewBag.Title در ویوی _Layout قرار خواهد گرفت.

همچنین دستور دیگری به نام @RenderBody() در ویوی _Layout تعریف شده بود. هر آنچه که بعد از دستورات ViewBag و Layout در ویوی ChildView1 مشاهده می کنید به عنوان body برای ویوی ChildView1 قرار خواهند گرفت. درست حدس زدید بخش body مربوط به ChildView1 در زمان اجرای برنامه به جای دستور @RenderBody() قرار خواهد گرفت. این مراحل در شکل زیر نشان داده شده است:



شکل ۶ - ۱۴

دستورات ویوی ChildView1 را به صورت زیر تغییر دهید:

```
@{
    ViewBag.Title = "ChildView1";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>
    Hiiiiii,I'm childview1 body.
    <br />
    I'm using <b>'~/Views/Shared/_Layout.cshtml'</b> for my layout!
</h2>
```

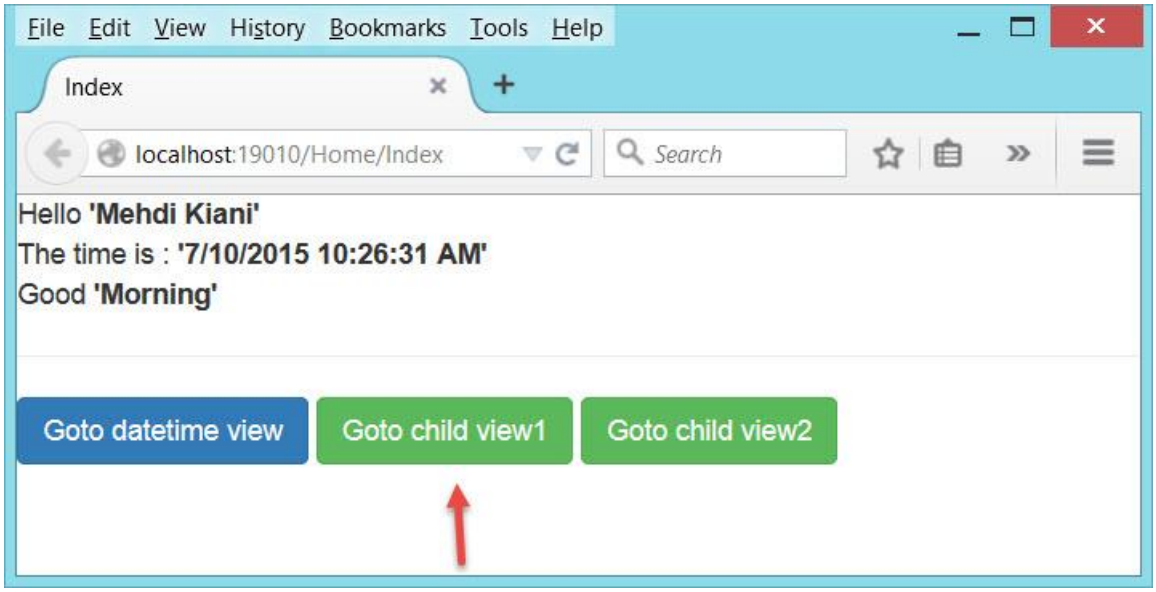
مراحل ایجاد ChildView1را از اکشن متد آن تا ویو و دستورات آن انجام دهید تا یک ویوی دیگری به نام ChildView2 ایجاد کنید. دستورات آن را به صورت زیر تغییر دهید:

```
@{
    ViewBag.Title = "ChildView2";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>
    Hiiiiii,I'm childview2 body.
    <br />
    I'm using <b>'~/Views/Shared/_Layout.cshtml'</b> for my layout!
</h2>
```

فایل Index.cshtml را باز کرده و دستورات زیر را به آن اضافه کنید:

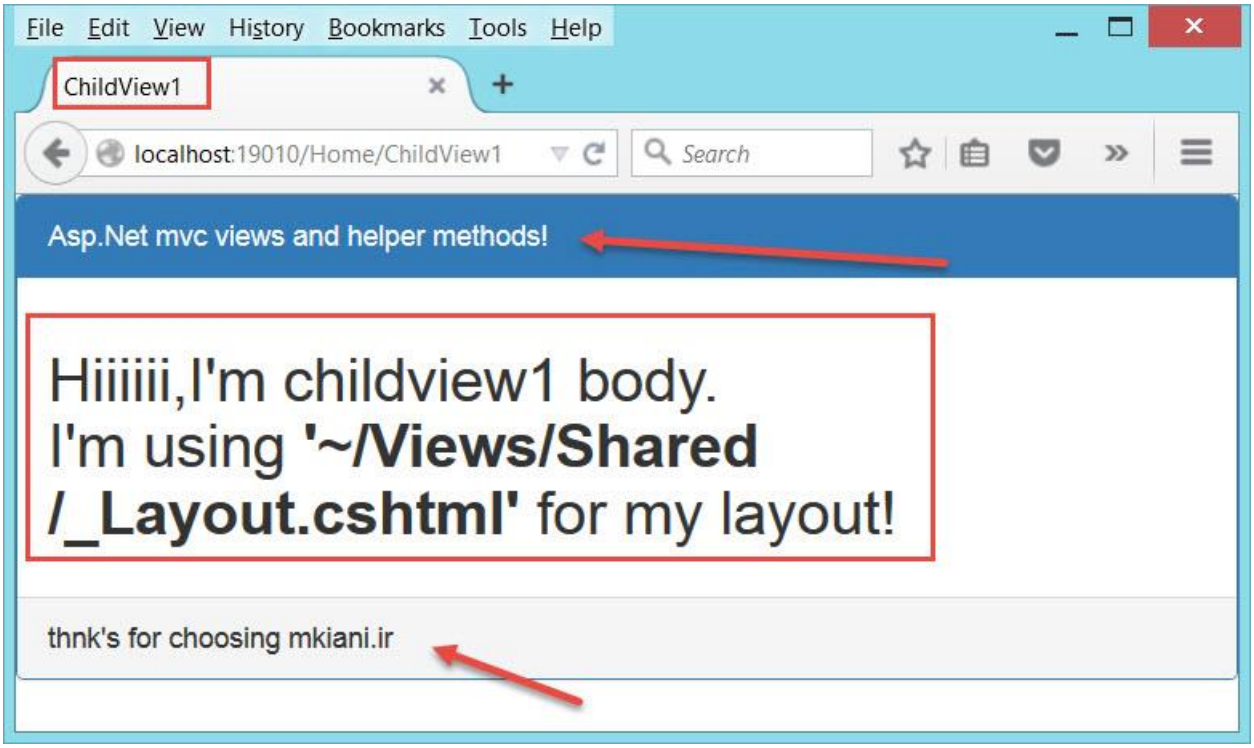
```
@Html.ActionLink("Goto child view1", "ChildView1", null, new
    {
        @class = "btn btn-success"
    })
     @Html.ActionLink("Goto child view2", "ChildView2", null, new
    {
        @class = "btn btn-success"
    })
```

حال برنامه را اجرا کنید و بر روی لینک Goto child view1 کلیک کنید:



شکل ۶ - ۱۵

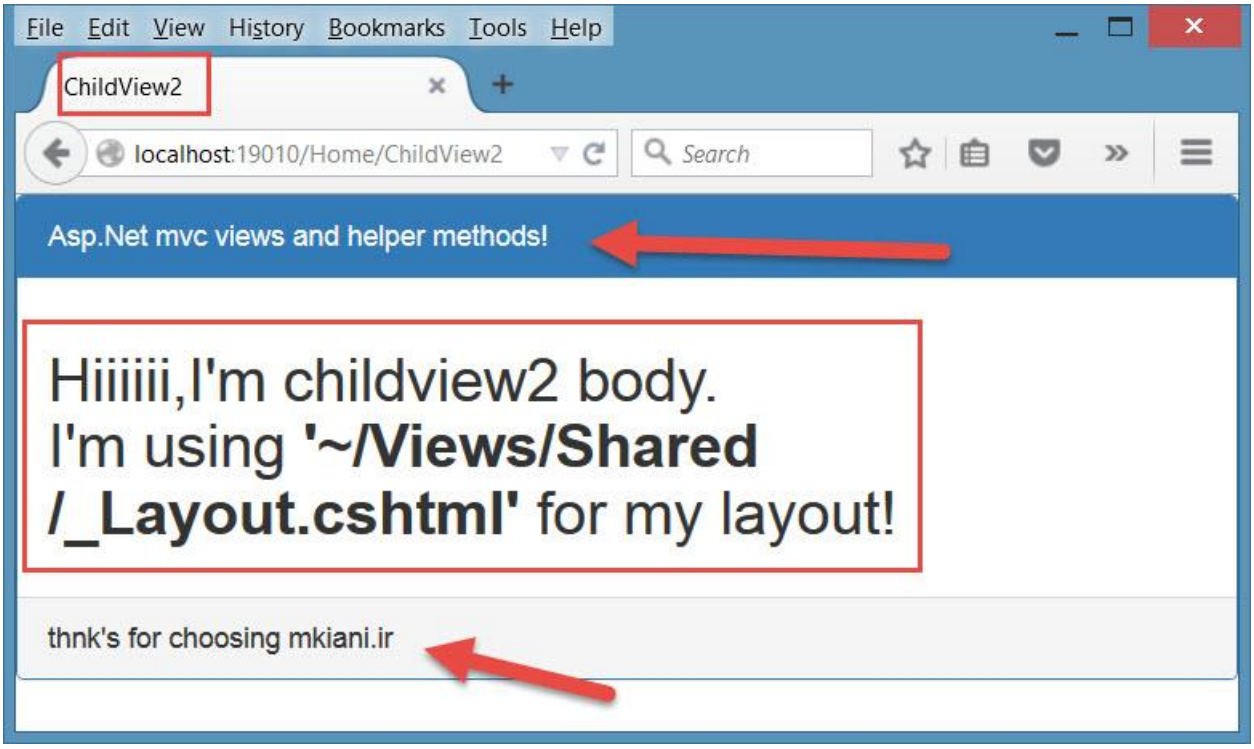
و نتیجه :



شکل ۶ - ۱۶

همانطورکه می بینید قسمت های فلش دار از ویوی Layout_ و قسمت های کادر قرمز رنگ از ویو ChildView1 در خروجی ظاهر شده اند.

حال اگر به صفحه اصلی بازگردید و روی لینک Goto child view2 کلیک کنید. نتیجه زیر را خواهید دید:



شکل ۶ - ۱۷

همانطور که مشاهده می کنید title و body از ویوی ChildView2 با کد های Layout_ تجمیع شده و در خروجی نشان داده شده است.

ایجاد بخش های سفارشی

همانطور که مشاهده کردید عملکرد دستور @RenderBody() این است که محتویات body ویو های فرزند را درون خود رندر کرده و در خروجی نهایی ظاهر می کند.حال در برخی از مواقع نیاز داریم تا به جای قرار دادن کلیه محتویات ویوی فرزندان(مثلا ChildView1) در تنها یک بخش از ویوی والد(Layout_) بخواهیم بخش های مختلفی از ویوی فرزندان را در مکان های متفاوتی از ویوی والد رندر کنیم. در این حالت mvc امکان ایجاد بخش های مختلف را توسط دستور @Section در ویو فرزندان را برای برنامه نویسان فراهم کرده است. همچنین توسط دستور @RenderSection در ویوی والد می توان بخش های تعریف شده در ویوی فرزندان را در ویوی والد رندر و در خروجی نمایش داد.

به عنوان مثال فایل ChildView1.cshtml را باز کنید و دستورات آن را مطابق زیر تغییر دهید :

```
@{
```

```
    ViewBag.Title = "ChildView1";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@section Header{
    <h1>
        I'm custom header for child view1!!!
    </h1>
}

<h2>
    Hiiiiii,I'm childview1 body.
    <br />
    I'm using <b>'~/Views/Shared/_Layout.cshtml'</b> for my layout!
</h2>
@section Footer{
    <h3>
        I'm custom footer for child view1!!!
    </h3>
}
```

همانطور که مشاهده می کنید توسط دستور @section(با s کوچک) اقدام به ایجاد دو بخش به نام های Header و Footer در فایل ChildView1.cshtml کرده ایم. این فایل را ذخیره کرده وببنیدید.

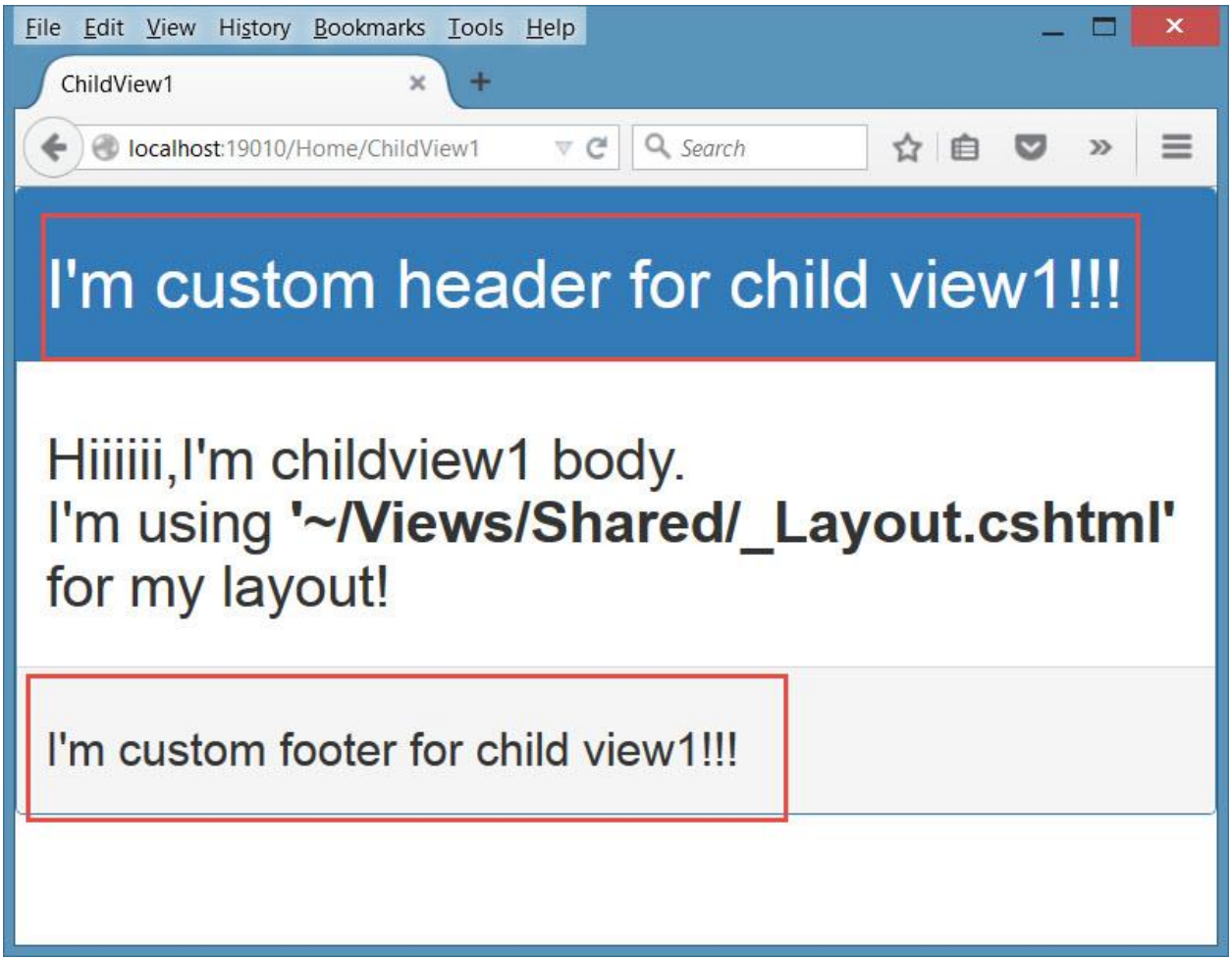
حال فایل _Layout.xshtml را باز کنید و دستورات آن را مطابق زیر تغییر دهید:

```
<!DOCTYPE html>
<html>
<head>

    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>@ViewBag.Title</title>

</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            @RenderSection("Header")
        </div>
        <div class="panel-body">
            @RenderBody()

        </div>
        <div class="panel-footer">
            @RenderSection("Footer")
        </div>
    </div>
```

```
</body>
</html>
```

در فایل _Layout.chtml توسط دستور @RenderSection مشخص کرده ایم که بخش های Header و Footer هر یک در کدام قسمت از بخش های ویوی والد باید رندر شوند.

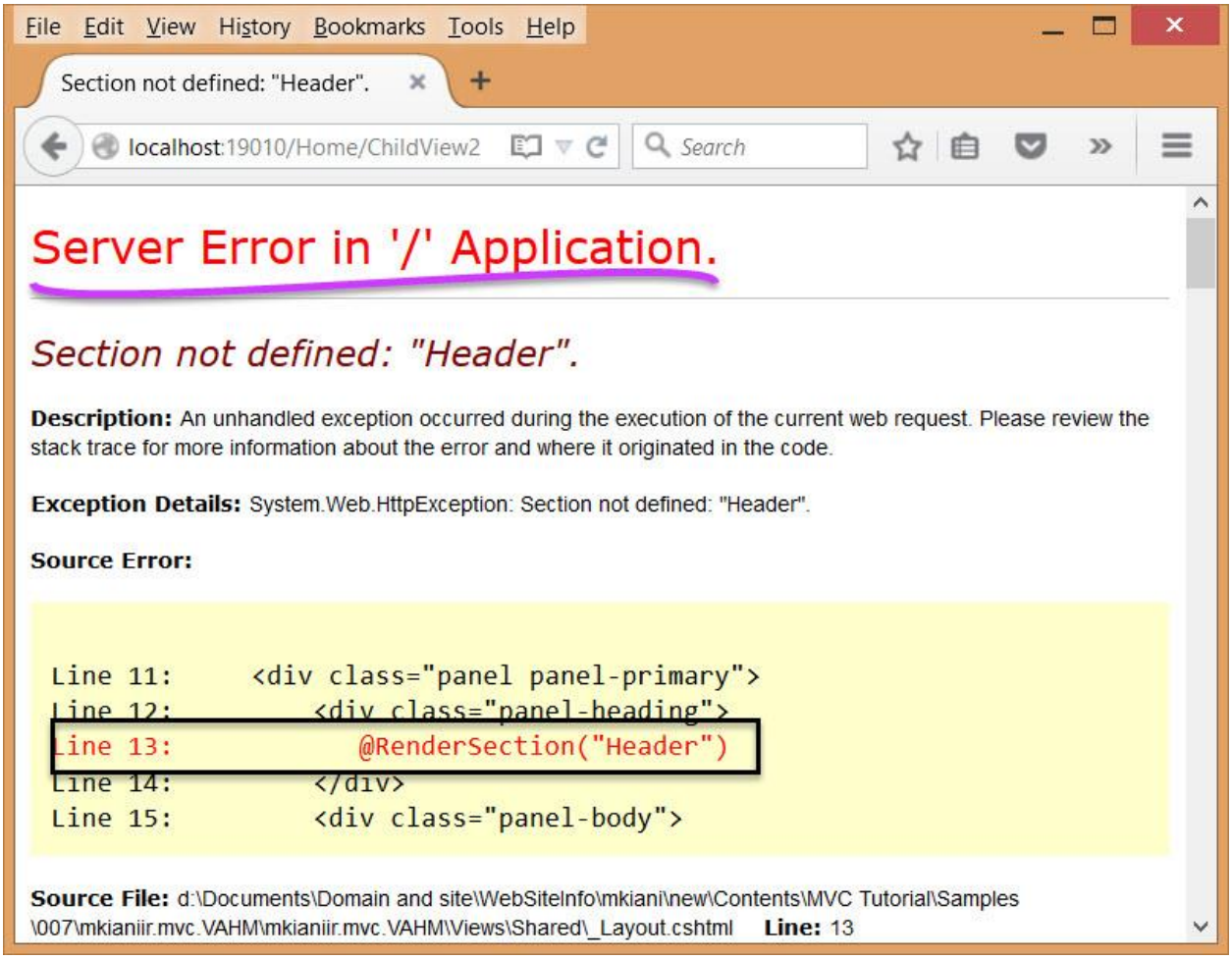
حال برنامه را اجرا کنید و بر روی لینک Goto child view1 کلیک کنید. خروجی شبیه به شکل زیر خواهد بود:



شکل ۶ - ۱۸

همانطور که مشاهده می کنید بخش های Header و Footer ویوی CHildView1 در ویوی والد(_Layout) رندر شده اند.

حال به صفحه اصلی باز گشته و این بار بر روی لینک Goto child view2 کلیک کرده و نتیجه را مشاهده کنید.



شکل ۶ - ۱۹

چه اتفاقی افتاد؟ همانطور که مشاهده می کنید برنامه با خطا مواجه شده است. علت این خطا این است که در ویوی ChildView2 بخش های Header و Footer تعریف نشده اند. بنابر این برنامه نمی تواند چیزی که وجود ندارد را رندر کند!

برای رفع خطای فوق دو راهکار وجود دارد.

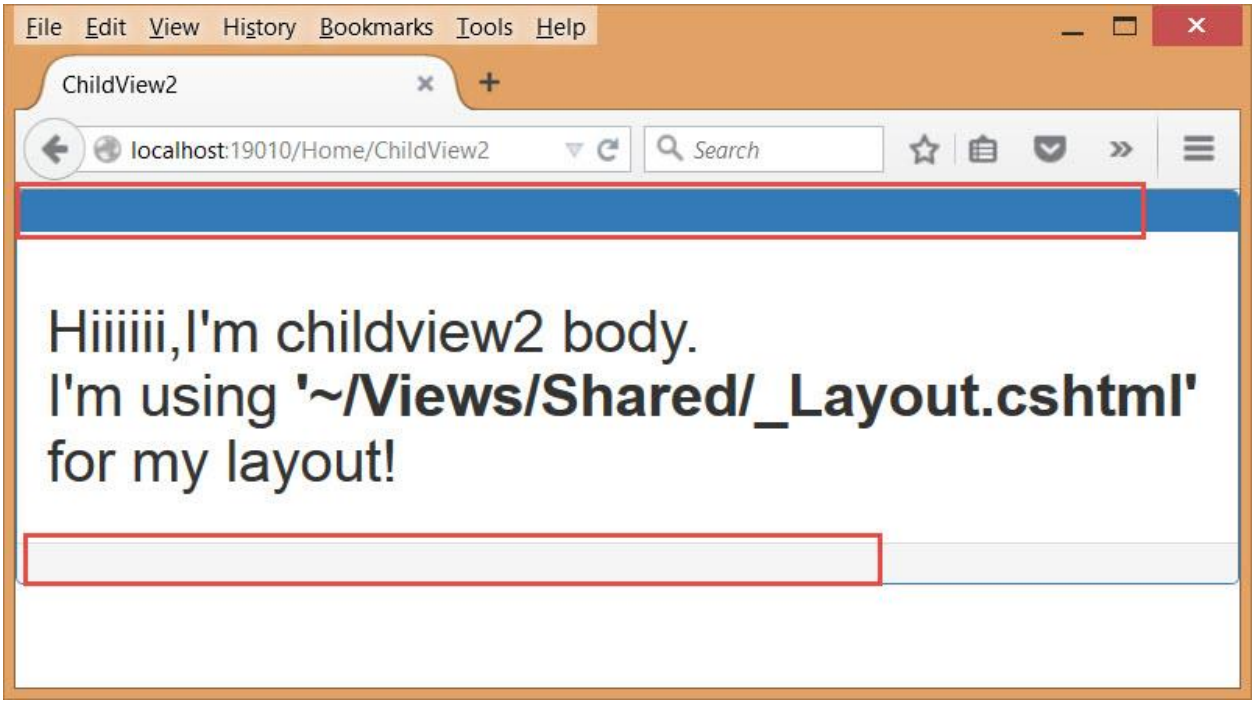
راهکار اول این است که برای ChildView2 هم همانند ChildView1 اقدام به تعریف بخش های Header و Footer کنید. اما این راهکار ممکن است شما را دچار مشکل سازد. چرا که در یک پروژه واقعی تعداد ویو ها ممکن است بسیار زیاد باشد. همچنین تعداد بخش ها می توان از یک ویو به یک ویوی دیگر متفاوت باشد.با این راهکار شما مجبو خواهید بود تا تمامی بخش های موجو در برنامه را در تمامی ویو ها تعریف کنید که عاقلانه نیست.

خوشبختانه mvc برای این مشکل راهکار مناسب تری نیز ارائه داده است. متد RenderSection در ویوی والد را به یاد بیاورید. ما آن را با یک آرگومان که در واقع نام بخش مورد نظر بود مورد استفاده قرار دادیم. این متد حالت دیگری نیز دارید که می تواند علاوه بر نام بخش مورد نظر یک آرگومان دیگر از جنس Boolean دریافت کند. نام این آرگومان که معرف عملکردش نیز می باشد required است. توسط این آرگومان می توانید مشخص کنید که آیا بخش مورد نظر الزاما می بایستی در ویوی فرزند تعریف شده باشد یا خیر.مقدار پیش فرض آن true می باشد که شما می توانید آن را به false تغییر دهید.

```
<!DOCTYPE html>
<html>
<head>

    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
    <title>@ViewBag.Title</title>

</head>
<body>
    <div class="panel panel-primary">
        <div class="panel-heading">
            @RenderSection("Header",false)
        </div>
        <div class="panel-body">
            @RenderBody()

        </div>
        <div class="panel-footer">
            @RenderSection("Footer",false)
        </div>
    </div>
</body>
</html>
```

حال چنانچه بخش های Header و View در ویو های فرزند تعریف شده باشند در خروجی نهایی رندر می شوند در غیر این صورت از این بخش ها صرف نظر می شود.

مجددا برنامه را اجرا کرده و بر روی لینک Goto childview2 کلیک کنید. نتیجه شبیه به شکل زیر خواهدبود:



همانطور که مشاهده می کنید جای بخش های Header و Footer چون که در ChildView2 تعریف نشده است خالی می باشند.

استفاده از _ViewStart

اگر به دستورات ویو های ChildView1 و ChildView2 توجه کنید می بینید که در هر دوی آن ها دستور

```
Layout = "~/Views/Shared/_Layout.cshtml";
```

تعریف شده است.

حال اگر نام فایل _Layout.cshtml را بخواهیم تغییر دهیم آنگاه مجبور خواهیم بود به سراغ تک تک ویو هایی که از _Layout به عنوان ویوی والد خود استفاده می کردند رفته و در تمامی آن ها نام _Layout را تغییر دهیم.

یا فرض کنید بخواهیم ویوی والد را برای ویو های برنامه تغییر دهیم. باز در این حالت مجبور هستیم برای تک تک ویوها این عمل را انجام دهیم.

خوشبختانه mvc برای این مشکل نیز راه حل مناسبی ارائه داده است و آن استفاده از View Start file می باشد. زمانی که یک ویو برای عملیات رندرینگ در نظر گرفته می شود mvc به دنبال فایلی به نام

_ViewStart.cshtml می گردد و هر آنچه که درون این فایل باشد را برای تک تک ویو فایل ها در نظر می گیرد. در واقع مانند این است که دستورات این فایل را برای تک تک ویو ها نوشته باشیم.

برای درک بهتراین موضوع بر روی پوشه Views در پنجره Solution Explorer کلیک راست کنید و از منوی Add گزینه View را انتخاب نمائید و نام آن را _ViewStart قرار دهید.

حال کد های درون این فایل را به صورت زیر تغییر دهید:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

همانطور که مشاهده می کنید در فایل _ViewStart.cshtml خاصیت Layout با اشاره به فایل _Layout.cshtml نوشته شده است. با نوشتن این دستور دیگر نیازی به دستور مشابه در فایل های ChildView1و ChildView2 نمی باشد. بنابراین می توانیم به راحتی آن ها را حذف نمائیم و برنامه بدون ایراد و مانند قبل اجرا خواهد شد.

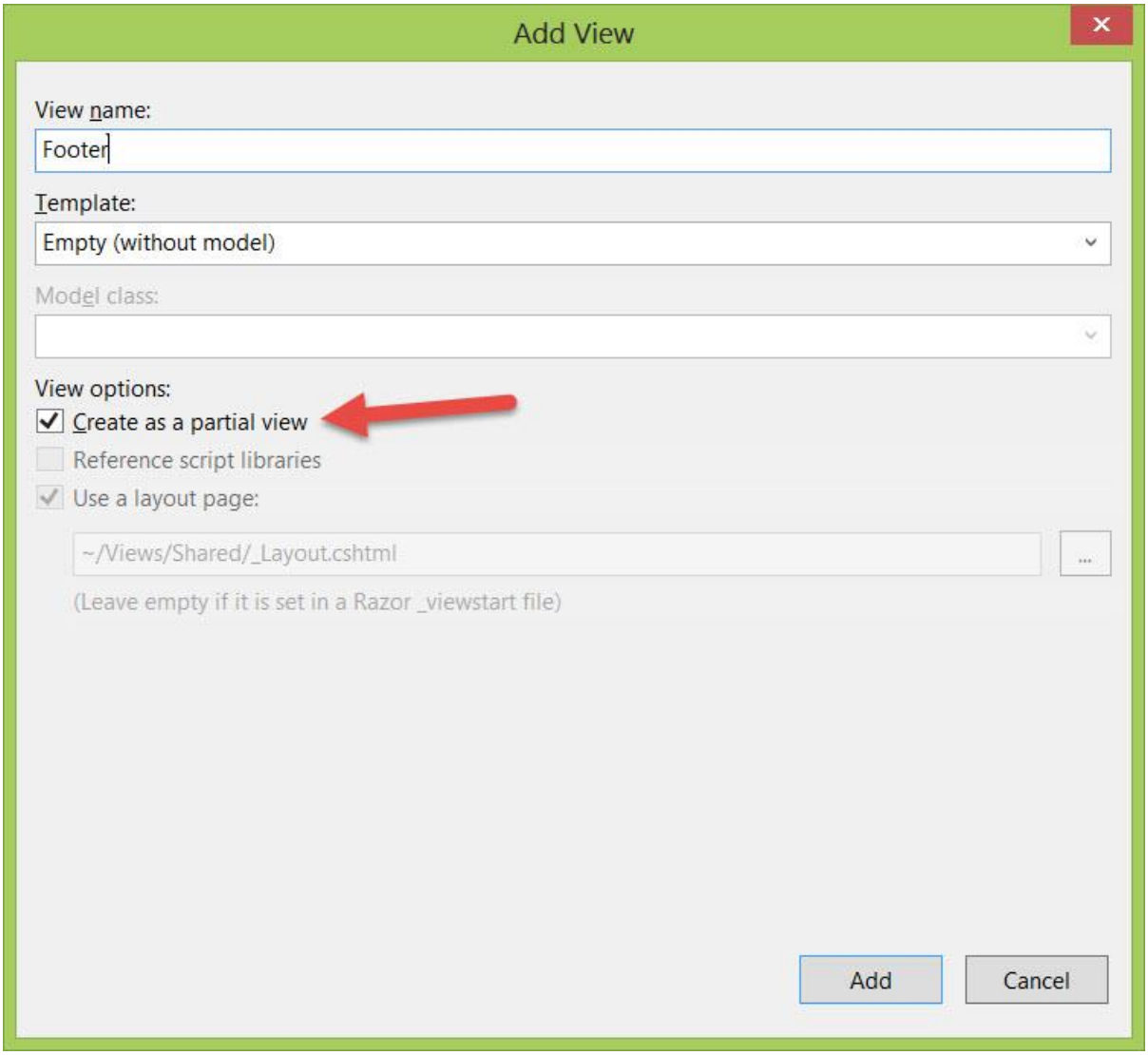
نکته: چنانچه بخواهید ویویی از _Layout به عنوان ویوی والد استفاده نکند می توانید مقدار آن null قرار دهید و یا اینکه یک ویوی دیگر را به عنوان ویوی والد برای آن در نظر بگیرید.

استفاده از PartialView ها

همانطور که پیشتر گفته شد PartialViewها ویو هایی هستند که به صورت مستقیم قابل مشاهده نبوده و محتویات آن ها در یک ویوی دیگر رندر خواهد شد. PartialView ها در mvc همان نقشی را دارند که UserControl ها در Asp.Net WebForm دارند. در واقع با PartialView ها می توانید محتویاتی را یک بار تعریف و به تعداد دلخواه از آن استفاده کنید.

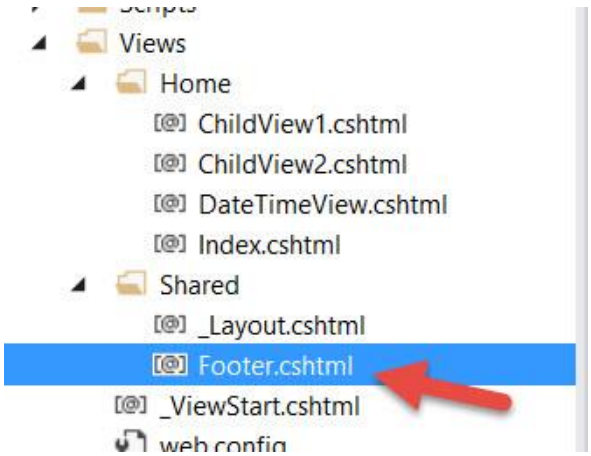
بر روی پوشه Shared در پوشه Views از پنجره SolutionExplorer کلیک راست کرده و از منوی Add گزینه View را انتخاب نمائید.

درپنجره Add View نام ویو را به Footer تغییر دهید. همچنین قسمت Create as partial view را فعال نموده و سپس دکمه Add را کلیک کنید.



شکل ۶ - ۲۰

با این کار یک فایل با نام Footer.cshtml در پوشه Shared ایجاد می گردد.



شکل ۶ - ۲۱

دستورات زیر را در فایل Footer وارد نمائید و آن را ذخیره کنید.

```
<h3>I'm a shared footer inside ~/Views/Shared/Footer.cshtml
file!</h3>
```

حال فایل ChildView1.cshtml را باز کنید و دستورات آن را مطابق ذیل تغییر دهید:

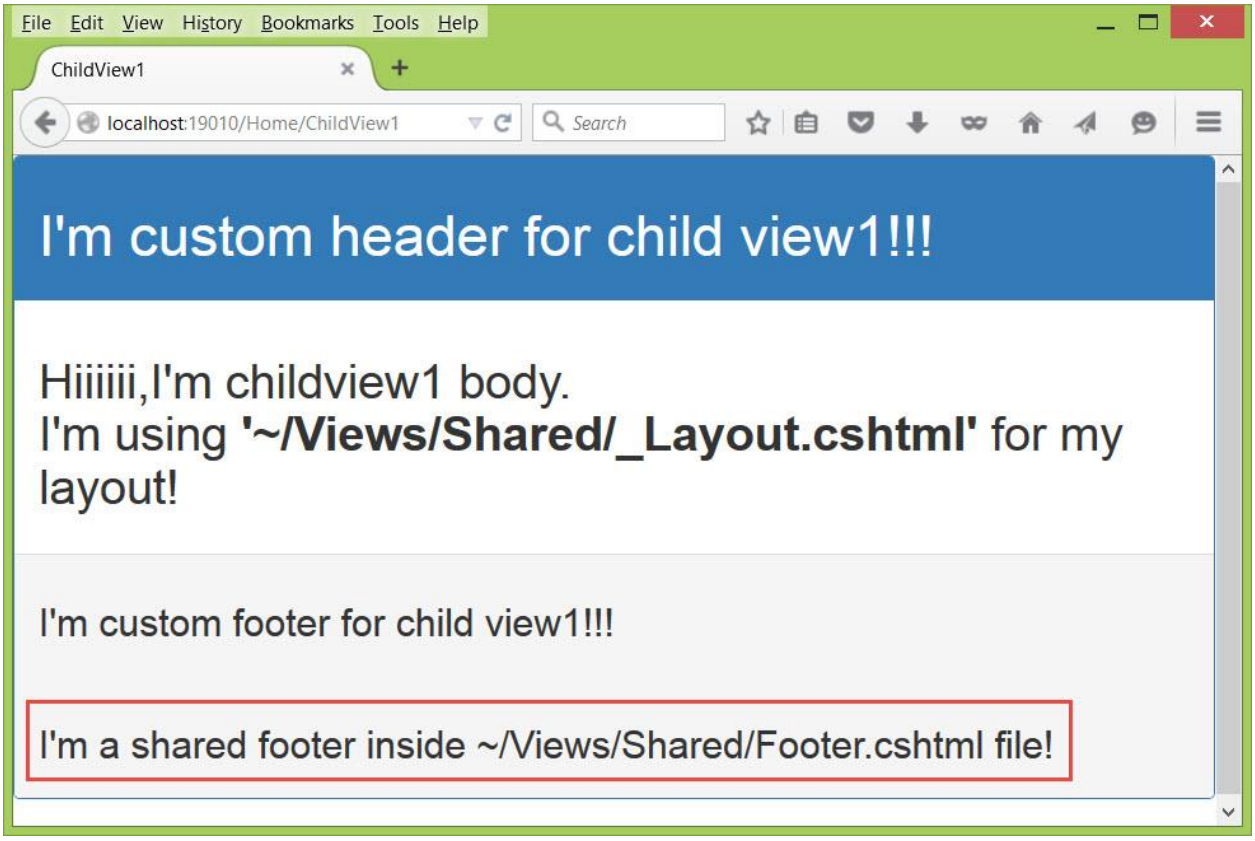
```
@{
    ViewBag.Title = "ChildView1";
}
@section Header{
    <h1>
        I'm custom header for child view1!!!
    </h1>
}

<h2>
    Hiiiiii,I'm childview1 body.
    <br />
    I'm using <b>'~/Views/Shared/_Layout.cshtml'</b> for my layout!
</h2>
@section Footer{
    <h3>
        I'm custom footer for child view1!!!
    </h3>
    <br />
    @Html.Partial("Footer")
}
```

همین کار را با فایل ChildView2.cshtml انجام دهید:

```
@{
    ViewBag.Title = "ChildView2";
}
<h2>
    Hiiiiii,I'm childview2 body.
    <br />
    I'm using <b>'~/Views/Shared/_Layout.cshtml'</b> for my layout!
    <br />
    @Html.Partial("Footer")
</h2>
```

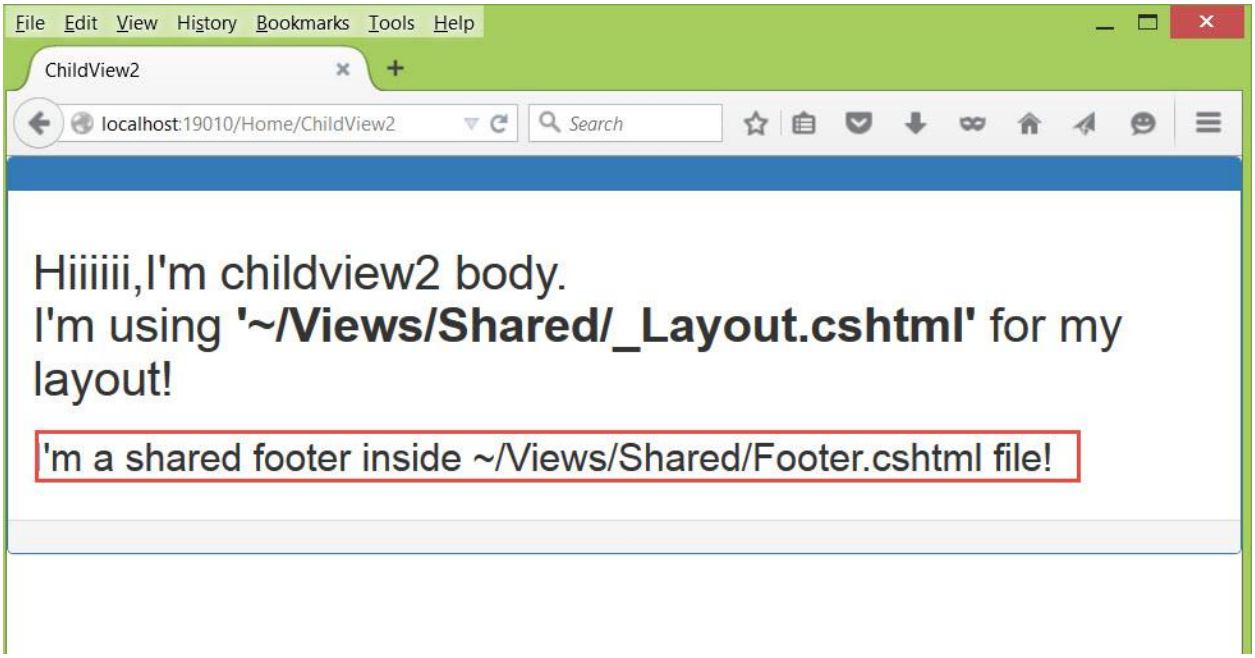
توسط متد راهنمای Partial می توان به یک PartialView دسترسی پیدا کرد.

حال برنامه را اجرا کنید و بر روی لینک Goto child view1 کلیک کنید.



شکل ۶ - ۲۲

همانطور که مشاهده می کنید محتویات فایل Footer.cshtml در ویوی ChildView1 رندر شده است. اگر به صفحه اصلی باز گردید و بر روی لینک Goto child view2 نیز کلیک کنید مشاهده می کنید که محتویات فایل Footer.cshtml در این ویو نیز رندر شده است.



شکل ۶ - ۲۳

با این کار یک ویوی اشتراکی بین سایر ویو ها ایجاد کرده ایم. در نتیجه زمانی که تغییری در فایل Footer.cshtml ایجاد کنیم به صورت خود کار این تغییر در تمامی ویوهایی که آن را رندر می کنند اعمال شده و بنابر این حجم کد نویسی با استفاده از این روش بسیار کاهش می باشد.

نکته: بهتر است در نامگذاری ویوهای جزئی(PartialView) از کاراکتر _ استفاده شود(من چون در اینجا یک مثال ساده را آموزش می دهم این قاعده را رعایت نکردم اما شما در پروژه های واقعی خود این مورد را رعایت کنید.)

به طور کلی این نکته را در نظر داشته باشید که در نامگذاری ویوهایی که به صورت مستقیم قابل مشاهده نیستند مانند PartialView ها ، Layout ها و ViewStart ها از کاراکتر _ در ابتدای نام آن ها استفاده کنید.

باند کردن مدل به PartialView ها

همانند View های معمولی PartialView ها نیز می توانند دارای مدل داده ای باشد. فایل Footer.cshtml را باز کنید و کد های آن را به صورت زیر تغییر دهید:

```
@model DateTime
<h3>I'm a shared footer inside ~/Views/Shared/Footer.cshtml file!</h3>
<h4>The date time is : @Model.ToString()</h4>
```

همانطور که مشاهده می کنید نوع DateTime برای مدل Footer در نظر گرفته شده است و مقدار آن در تگ h4 چاپ شده است.

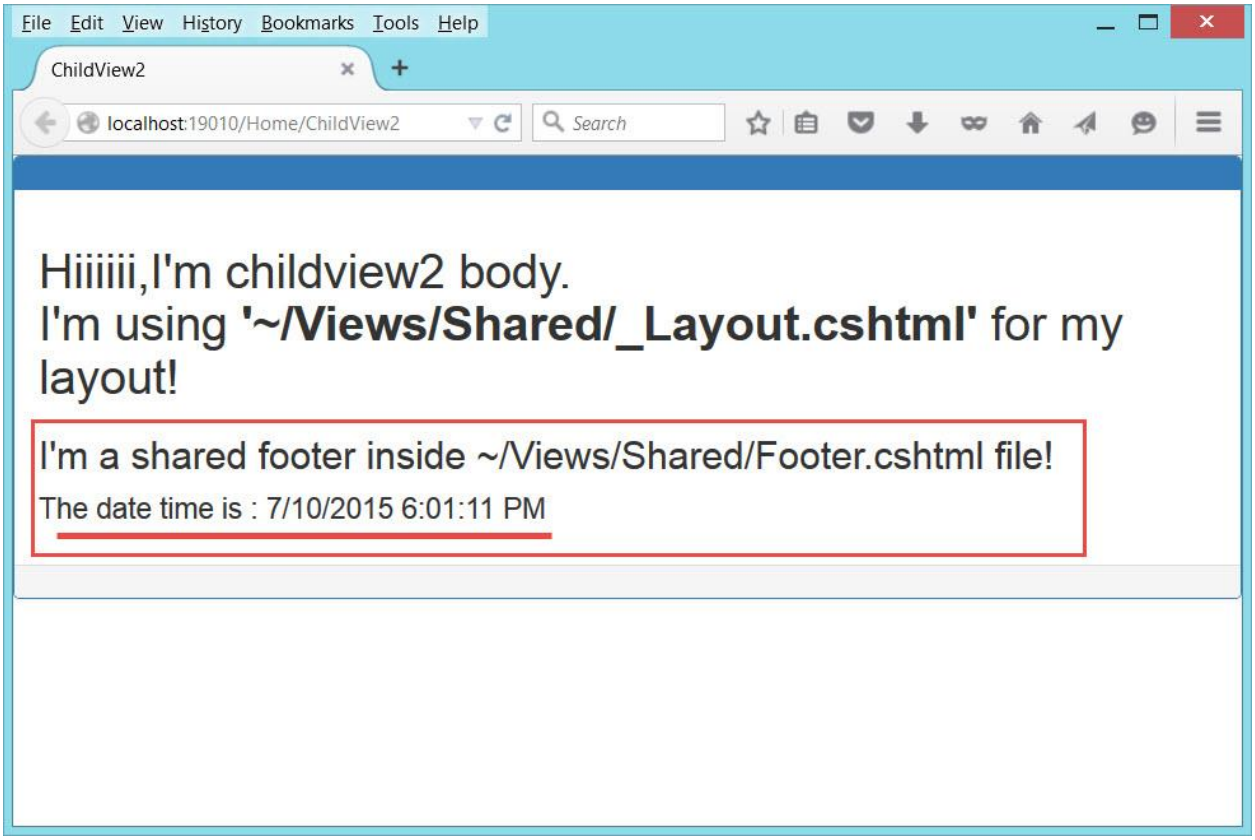
حال فایل ChildView1.cshtml را باز کنید و دستور @Html.Partial را به صورت زیر تغییر دهید:

```
@Html.Partial("Footer", DateTime.Now)
```

متد Partial علاوه بر نام PartialView می تواند آرگومان دیگری دریافت کند که معرف مدل داده ای برای Partial View مورد نظر است.

همین کار را با فایل ChildView2.cshtml انجام دهید.

حال برنامه را اجرا کنید و روی لینک Goto child view1 کلیک کنید.خروجی شبیه به عکس زیر خواهد بود.

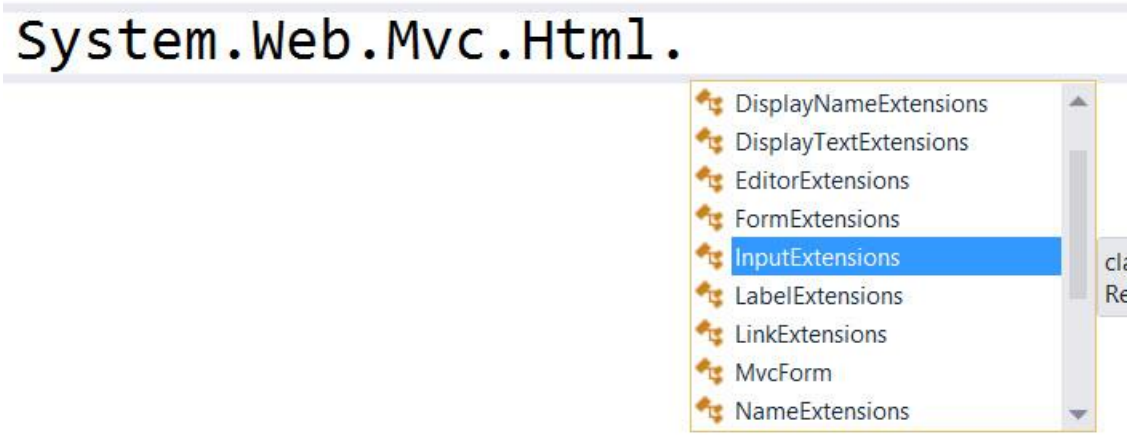


شکل ۶ - ۲۴

متد های راهنما

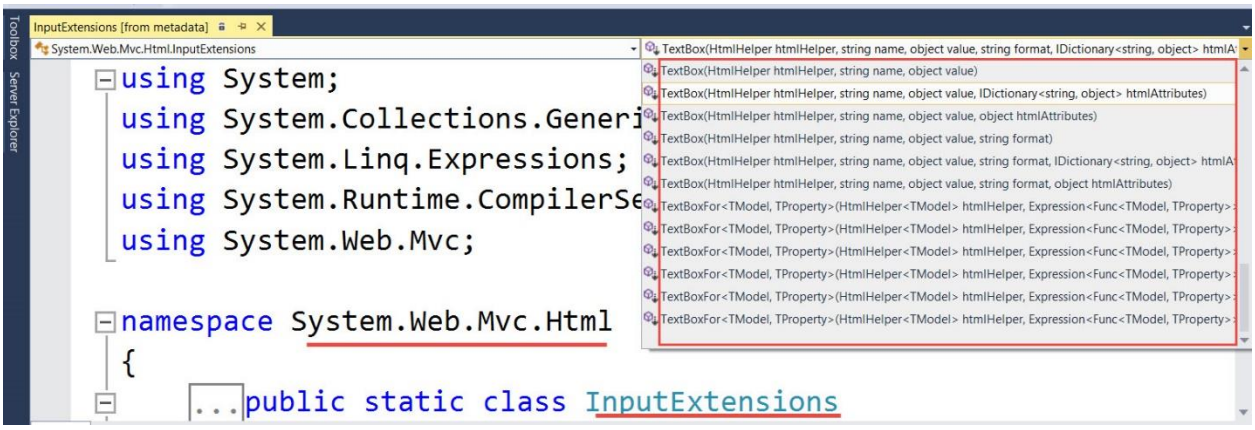
همانطور که می دانید خروجی نهایی یک برنامه تحت وب کد ها ی html بوده که توسط مرورگر تفسیر شده و خروجی قابل فهم برای کاربر نهایی را ایجاد می کند. در Asp.Net Web Form کنترل های وب هر کدام دارای متدی به نام Render می باشند که خروجی Html متناظر با خود را تولید می کنند. همچنین این کنترل ها دارای خاصیتی به نام ViewState بوده که مسئولیت نگهداری مقادیر خواص کنترل(حالت کنترل) را دارد. در Mvc این وظیفه را متد های راهنما انجام می دهند. خروجی متد های راهنما یک رشته ساده(نمونه ای از کلاس MvcHtmlString) می باشد. دستور @Html اشاره به کلاس HtmlHelper دارد و چون متد های راهنما متد های الحاقی برای کلاس HtmlHelper می باشند بنابر این با استفاده از دستور @Html می توانید از این متد ها استفاده نمائید.

در فضای نام System.Web.Mvc.Html کلاس هایی برای تعریف متد های راهنما به صورت متد های الحاقی تعبیه شده اند.



شکل ۶ - ۲۵

به عنوان مثال متد TextBoxیا TextBoxFor درون کلاس InputExtensions تعریف شده است.



همانطور که مشاهده می کنید علاوه بر نوع عادی متدهای الحاقی نوشته شده برای کلاس HtmlHelper ، نوع ژنریک این متد ها نیز بعضا تعریف شده است. به عنوان مثال نوع ژنریک متد TextBox متد TextBoxFor می باشد.این متد های ژنریک متدهای الحاقی برای نوع ژنریک کلاس HtmlHelper می باشند که در ادامه با استفاده برخی از آن ها آشنا خواهیم شد.

قبل از ادامه لازم است تا یک اکشن متد به نام HelperMethods به کنترلر Home اضافه نمائید.

```
public ViewResult HelperMethods()
        {
            return View();
        }
```

همچنین یک ویو هم نام با اکشن متد فوق ایجاد نمائید.

فایل Index.cshtml را باز کنید و دستور زیر را به آن اضافه کنید:

```
@Html.ActionLink("Html helper methods", "HelperMethods", null, new
    {
        @class = "btn btn-warning"
    })
```

این دستور یک لینک به اکشن متد HelperMethods ایجاد می کند.

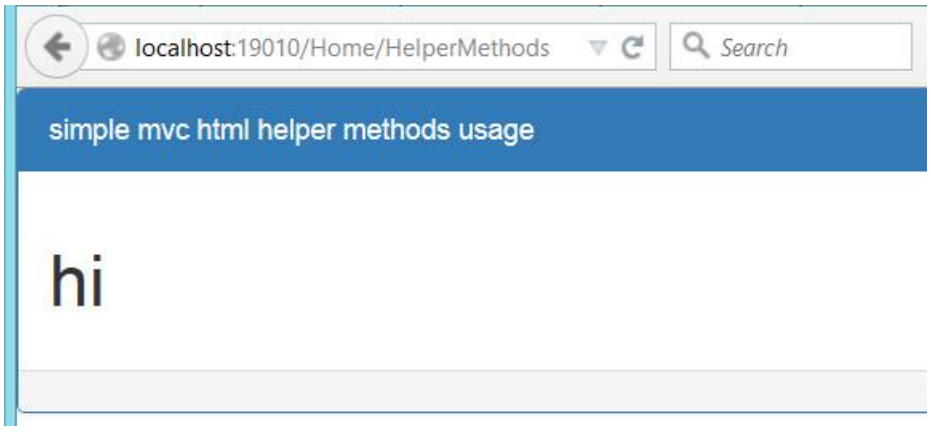
متد راهنمای ایجاد فرم

توسط متد BeginForm و EndForm می توانید اقدام به ایجاد فرم های html نمائید. خروجی این متد ها معادل تگ form در html می باشد.

فایل HelperMethods.Cshtml را باز کنید و دستورات زیر را درون آن بنویسید.

```
@{
    ViewBag.Title = "Mvc helper methods";
}
@section Header{
    simple mvc html helper methods usage
}
@{Html.BeginForm();}
<h1>hi</h1>
@{Html.EndForm();}
```

برنامه را اجرا کنید و روی لینک Html helper methods کلیک کنید:



شکل ۶ - ۲۶

خروجی نهایی html ایجاد شده از دستورافوق به شکل زیر خواهد بود.

```
<!DOCTYPE html>
<html>
<head>

    <meta name="viewport" content="width=device-width" />
    <link href="/Content/bootstrap.min.css" rel="stylesheet"
/>
    <title>Mvc helper methods</title>

</head>
<body>
```

```
    <div class="panel panel-primary">
        <div class="panel-heading">

    simple mvc html helper methods usage

        </div>
        <div class="panel-body">
```

```
<form action="/Home/HelperMethods" method="post">
<h1>hi</h1>
</form>
```

```
        </div>
        <div class="panel-footer">

        </div>
    </div>
</body>
</html>
```

همانطور که مشاهده می کنید توسط متد های BeginForm و EndForm یک فرم html ایجاد شده است.

استفاده از دستور using و حذف متد EndForm

در دستورات فوق برای ایجاد تگ آغازین form از متد BeginForm و برای تگ پایانی آن از متد EndForm استفاده شده است. با استفاده از دستور using به صورت زیر می توانید بدون استفاده از متد EndForm اقدام به ایجاد تگ فرم نمائید:

```
@using (Html.BeginForm())
{
}
```

دستور using باعث می شود تا بلافاصله پس از پایان بلاک دستورات به صورت خودکار تگ پایانی فرم یعنی </form> به خروجی اضافه می گردد. همانطور که مشاهده می کنید خوانایی دستور فوق بیشتر از دستور قبلی (استفاده از متد های BeginForm EndForm به صورت مجزا) است.

متد های راهنمای input :

همانطور که می دانید در Html کنترل های Chechbox، RadioButton، TextBox، PasswordBox و Hidden همگی از نوع کنترل input بوده که با صفت type نوع آن ها مشخص می شود. در کلاس InputExtemsions در فضای نام System.Web.Mvc.Html متد هایی را برای ایجاد کنترل های فوق تعریف کرده است.

ایجاد TextBox

برای ایجاد TextBox از دستور @Html.TextBox به صورت زیر

```
@Html.TextBox(“Name")
```

خروجی دستور فوق به صورت زیر خواهد بود

```
<input id=”Name” name=”Name” type=”text” value=”” />
```

ایجاد CheckBox

برای ایجاد کنترل CheckBox از دستور @Html.CheckBox به صورت زیر استفاده می کنیم:

```
@Html.CheckBox("chb1", false)
```

پارامتر اول صفت name و پارامتر دوم که از جنس bool می باشد مشخص می کند که کنترل checkbox دارای تیک می باشد یا خیر.

خروجی دستور فوق به صورت زیر خواهد بود

```
<input id="chb1" name="chb1" type="checkbox" value="false" />
```

ایجاد RadioButton

برای ایجاد RadioButton از دستور @Html.RadioButton به صورت زیر استفاده می شود:

```
@Html.RadioButton("sex", "Man")
```

خروجی دستور فوق به صورت زیر خواهد بود:

```
<input id="sex" name="sex" type="radio" value="Man" />
```

ایجاد فیلد پسورد

برای ایجاد فیلد متنی که نوع آن پسورد باشد از دستور @Html.Password به صورت زیر استفاده می کنیم:

```
@Html.Password("password")
```

خروجی دستور فوق به صورت زیر خواهد بود

```
<input id="password" name="password" type="password" />
```

ایجاد فیلد Hidden

برای ایجاد فیلد مخفی از دستور @Html.Hidden به صورت زیر استفاده می کنیم:

```
@Html.Hidden("hid","hid-val")
```

خروجی دستور فوق به صورت زیر خواهد بود

```
<input id="hid" name="hid" type="hidden" value="hid-val"
/></form>
```

ایجاد TextAre

برای ایجاد TextArea از دستور @Html.TextArea که در کلاس TextAreaExtensions در فضای نام System.Web.Mvc.Html تعریف شده است به صورت زیر استفاده می کنیم:

```
@Html.TextArea("Description",null,5,20,null)
```

آرگومان اول نام کنترل خواهد بود. آرگومان دوم مقدار پیش فرض کنترل. آرگومان سوم تعداد سطر های کنترل.آرگومان چهارم تعداد ستون های کنترل و آرگومان آخر سایر صفت های کنترل که می توان توسط آن مقدار دهی شوند.

خروجی دستور فوق به صورت زیر خواهد بود

```
<textarea cols="20" id="Description" name="Description"
rows="5">
</textarea>
```

ایجاد DropDownList

برای ایجاد یک DropDownList از دستور @Html.DropDownList استفاده می کنیم.

ساده ترین حالت متد DropDownList به صورت زیر است:

```
DropDownList(string name, IEnumerable<SelectListItem> selectList);
```

آرگومان اول نام کنترل می باشد و آرگومان دوم که لیستی از کلاس SelectedListItem می باشد آیتم های کنترل را نگه داری می کند.

آرگومان دوم را به روش های مختلف می تون مقدار دهی کرد. می توان به صورت دستی مقادیر آن را ساخت و یا داده های آن را از مدل به روش بایندینگ به آن نسبت داد. در دستورات زیر یک لیست به صورت دستی برای آیتم ها ساخته شده است.

```
@Html.DropDownList("language", new List<SelectListItem>
        {
            new SelectListItem{
             Text="C-sharp",
             Value="cs"
            },
            new SelectListItem{
             Text="Visual Basic",
             Value="vb"
            },
            new SelectListItem{
             Text="F-Sharp",
             Value="fs"
            }
        },
        "Select a language")
```

آرگومان سوم یعنی Select a language به عنوان مقدار پیش فرض کنترل در زمانی که هیچ آیتمی انتخاب نشده باشد به کار می رود.

خروجی دستور فوق به صورت زیر خواهد بود:

```
<select id="language" name="language"><option value="">Select
a language</option>
<option value="cs">C-sharp</option>
<option value="vb">Visual Basic</option>
<option value="fs">F-Sharp</option>
</select>
```

ایجاد ListBox:

همانطور که می دانید در html کنترل ListBox و کنترل DropDown هردو با تگ select نمایش داده می شوند. تفاوت آن ها در این است که در کنترل ListBox می توان همزمان بیش از یک آیتم را انتخاب کرد. در صورتی که در DropDown در هر لحظه تنها می توان یک گزینه را انتخاب کرد.این تفاوت با مقدار دهی صفت multiple امکان پذیر می باشد.

برای ایجاد کنترل ListBox از دستور @Html.ListBox استفاده خواهیم کرد:

```
@Html.ListBox("groups", new List<SelectListItem>
      {
          new SelectListItem{
           Text="User",
           Value="user",
           Selected=true
          },new SelectListItem{
           Text="Admin",
           Value="admin"
          },
          new SelectListItem{
           Text="SuperAdmin",
           Value="superadmin",
           Selected=true
          }
      })
```

همانطور که مشاهده می کنید توسط خاصیت Selected ، آیتم های اول و سوم نیز انتخاب شده اند.

```
<select id="groups" multiple="multiple" name="groups"><option
selected="selected" value="user">User</option>
<option value="admin">Admin</option>
<option selected="selected" value="superadmin">SuperAdmin</option>
</select>
```

آیتم های ListBox نیز همانند DropDown از نوع IEnumerable<SelectListItem> می باشد.

ایجاد لینک(تگ a)

برای ایجاد لینک بسته به اینکه بخواهیم به یک اکشن متد اشاره کنیم یا به یک روت از دستور @Html.ActionLink و یا @Html.RouteLink استفاده می شود.متد ActionLink را که بار ها مشاهده کردید. در اینجا مثالی از RouteLink میاوریم.

```
@Html.RouteLink("Redirect to home page", new
   {
       controller="Home",
       action="Index"
   })
```

آرگومان اول متن لینک و آرگومان دوم نام و مقادیر مربوط به بخش های Route را نشان می دهد.

خروجی دستور فوق به صورت زیر است:

```
<a href="/">Redirect to home page</a>
```

همانطور که مشخص است لینک فوق به صفحه Home اشاره دارد.(آدرس اصلی پروژه)

نکته: متدهای ListBox و DropDownList در کلاس SelectExtensions و متد ActionLink و RouteLink در کلاس LinkExtensions در فضای نام System.Web.Mvc.Html تعریف شده اند.

دستورات زیر نمونه ای از استفاده متدهای راهنما را نشا ن می دهد. این دستورات درون فایل HelperMethods نوشته شده اند:

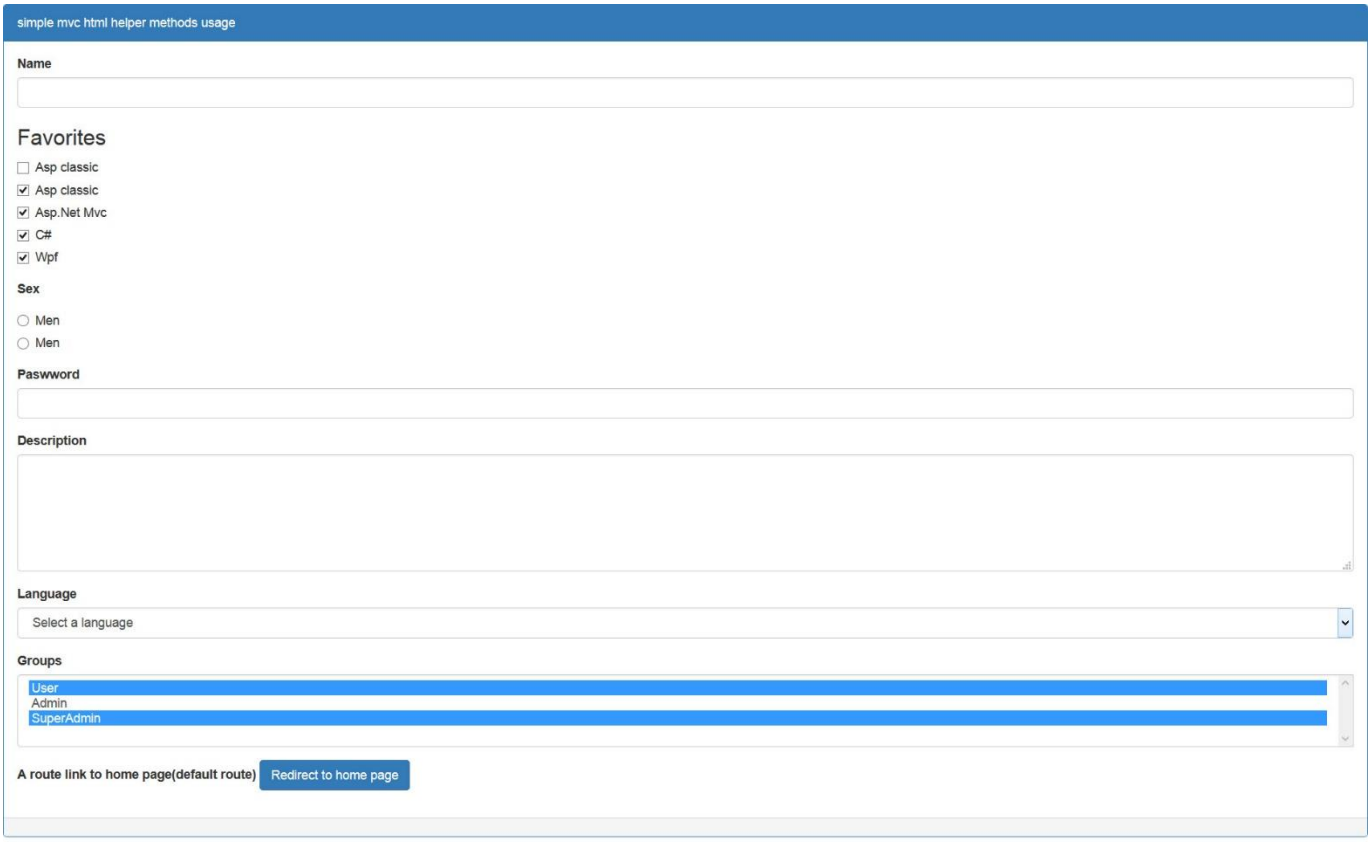
```
@{
    ViewBag.Title = "Mvc helper methods";
}
@section Header{
    simple mvc html helper methods usage
```

```
}
@using (Html.BeginForm())
{

    <div class="form-group">
        <label>Name</label>
        @Html.TextBox("name", null, new
{
    @class = "form-control"
})
    </div>
    <div class="form-group">
        <h3>Favorites</h3>
        <div class="checkbox">
            <label> @Html.CheckBox("asp_Classic", false) Asp
classic</label>
        </div>
        <div class="checkbox">
            <label> @Html.CheckBox("asp_Net", true) Asp
classic</label>
        </div>
        <div class="checkbox">
            <label> @Html.CheckBox("asp_Net_Mvc", true) Asp.Net
Mvc</label>
        </div>
        <div class="checkbox">
            <label> @Html.CheckBox("c_Sharp", true) C#</label>
        </div>
        <div class="checkbox">
            <label> @Html.CheckBox("wpf", true) Wpf</label>
        </div>
    </div>
    <div class="form-group">
        <label>Sex</label>
        <div class="radio">
            <label>@Html.RadioButton("sex", "Man") Men</label>
        </div>

        <div class="radio">
            <label>@Html.RadioButton("sex", "Woman") Men</label>
        </div>
    </div>
    <div class="form-group">
        <label>Paswword</label>
        @Html.Password("password", null, new
   {
       @class = "form-control"
   })
    </div>
```

```
@Html.Hidden("hid", "hid-val")
<div class="form-group">
    <label>Description</label>
    @Html.TextArea("description", null, 5, 20, new
{
    @class = "form-control"
})
</div>
<div class="form-group">
    <label>Language</label>

    @Html.DropDownList("language", new List<SelectListItem>
    {
        new SelectListItem{
         Text="C-sharp",
         Value="cs"
        },
        new SelectListItem{
         Text="Visual Basic",
         Value="vb"
        },
        new SelectListItem{
         Text="F-Sharp",
         Value="fs"
        }
    },
    "Select a language", new
    {
        @class = "form-control"
    })
</div>
<div class="form-group">
    <label>Groups</label>
    @Html.ListBox("groups", new List<SelectListItem>
   {
       new SelectListItem{
        Text="User",
        Value="user",
        Selected=true
       },new SelectListItem{
        Text="Admin",
        Value="admin"
       },
       new SelectListItem{
        Text="SuperAdmin",
        Value="superadmin",
        Selected=true
       }
   }, new
```

```
            {
                @class = "form-control"
            })
        </div>
        <div class="form-group">
            <label>A route link to home page(default route)</label>
            @Html.RouteLink("Redirect to home page", new
    {
        controller = "Home",
        action = "Index"
    }, new
          {
              @class = "btn btn-primary"
          })
        </div>


}
```

حال برنامه را اجرا کنید. خروجی شبیه به شکل زیر خواهد بود.



شکل ۶ - ۲۷

ارسال اطلاعات از ویو به کنترلر

در این قسمت می خواهیم اطلاعاتی که کاربر در فرم بالا تکمیل کرده است را به کنترلر ارسال کنیم. برای این منظور یک اکشن متد دیگر با نام HelperMethods در کنترلر Home به صورت زیر تعریف کنید.(به صفت HttpPost دقت کنید)

```
    [HttpPost]
        public ViewResult HelperMethods(String name,
            bool asp_Classic,
            bool asp_Net,
            bool asp_Net_Mvc,
            bool c_Sharp,
            bool wpf,
            String sex,
            String password,
            String description,
            String language,
            List<String> groups)
        {
            ViewBag.name = name;
            ViewBag.asp_Classic = asp_Classic ? "asp_Classic" : "";
            ViewBag.asp_Net = asp_Net ? "asp_Net" : "";
            ViewBag.asp_Net_Mvc = asp_Net_Mvc ? "asp_Net_Mvc" : "";
            ViewBag.c_Sharp = c_Sharp ? "c_Sharp" : "";
            ViewBag.wpf = wpf ? "wpf" : "";
            ViewBag.sex = sex;
            ViewBag.password = password;
            ViewBag.description = description;
            ViewBag.language = language;
            String selectedGroups = "";
            foreach (var item in groups)
            {
                selectedGroups += item;
                selectedGroups += ",";
            }
            ViewBag.groups = groups;
            return View("HelperMethodDetails");
        }
```

برای هر کنترلی که در ویوی تعریف شده است یک آرگومان هم نام با نام کنترل در متد تعریف شده است. زمانی که فرم post می شود مقادیر وارد شده توسط کاربر به صورت خودکار توسط mvc جمع آوری شده و به اکشن متد ارسال می شود.

دقت کنید که صفت HttpPost را برای متد فوق فراموش نکنید.

دستور آخر این متد ویویی به نام HelperMethodDetails را بر می گرداند.

یک ویو به نام HelperMethodDetails ایجاد کنید و دستورات زیر را در آن بنویسید:

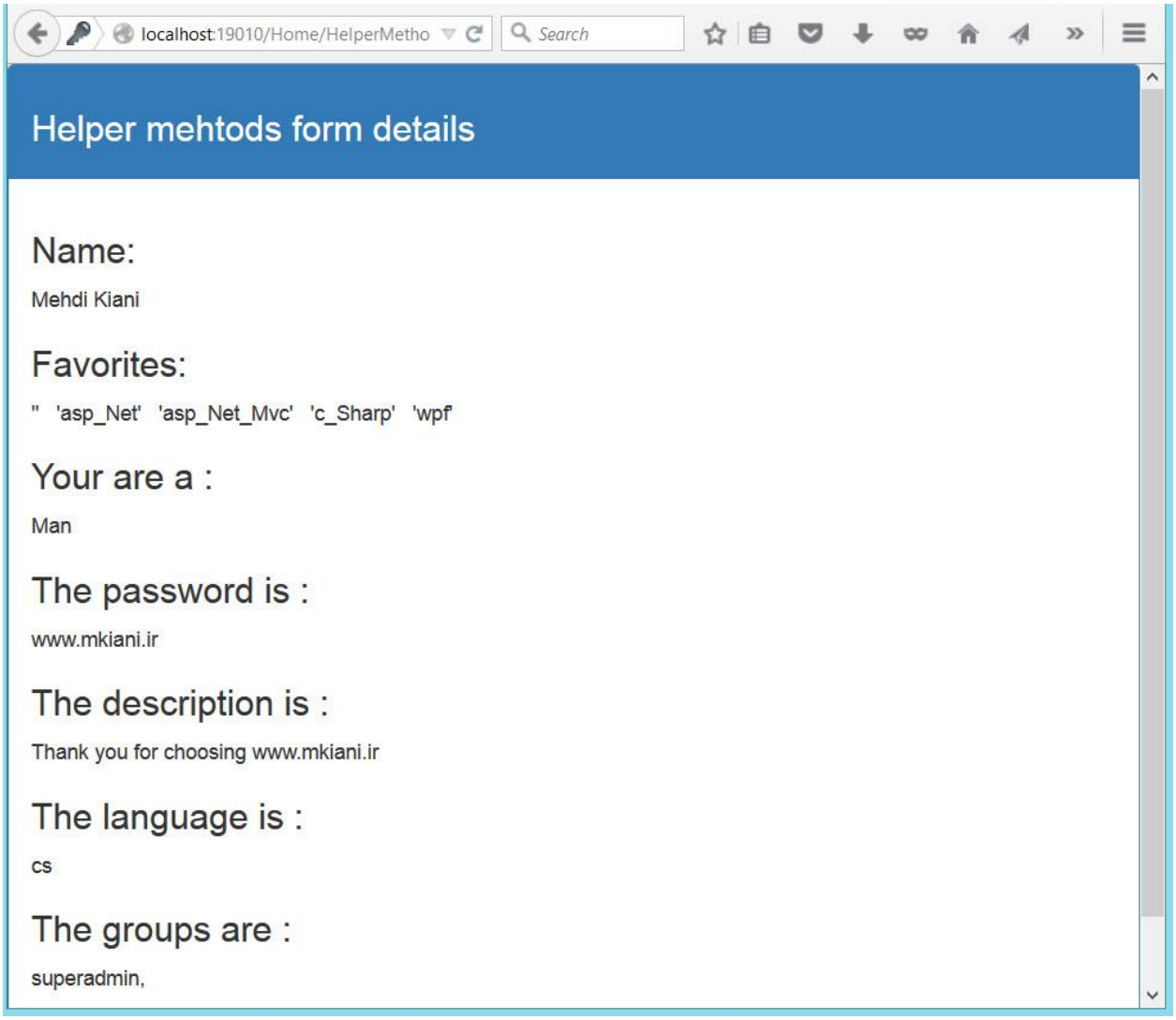
```
@{
    ViewBag.Title = "HelperMethodDetails";
}
@section Header{
    <h3>Helper mehtods form details</h3>
    }
<h3>Name:</h3>
@ViewBag.name
<h3>Favorites:</h3>
'@ViewBag.asp_Classic'  
'@ViewBag.asp_Net'  
'@ViewBag.asp_Net_Mvc'  
'@ViewBag.c_Sharp'  
'@ViewBag.wpf'  
<h3>Your are a : </h3>
@ViewBag.sex
<h3>The password is : </h3>
@ViewBag.password
<h3>The description is : </h3>
@ViewBag.description
<h3>The language is : </h3>
@ViewBag.language
<h3>The groups are : </h3>
@ViewBag.groups
```

دستور زیر را در فایل HelperMethods.cshtml وارد کنید.

```
<input type="submit" class="btn btn-success" value="send" />
```

دستور فوق یک دکمه برای ارسال اطلاعات فرم به کنترلر ایجاد می کند. حال برنامه را اجرا کنید.به فرم Helper Methods رفته و پس از تکمیل فرم دکمه send را کلیک کنید.

عکس زیر نمونه ای از اجرا را نشان می دهد:



شکل ۶ - ۲۸

اگرچه دستورات فوق برنامه خاصی را ارائه نمی کنند اما به شما کمک می کند تا با ایجاد کنترل ها در ویو و ارسال فرم تکمیل شده به کنترلر و استفاده از آن ها آشنا شوید.

ایجاد فرم با استفاده از مدل داده ای

در بخش قبلی ایجاد یک فرم اطلاعاتی را به صورت کاملا دستی و بدون هیچ مدل داده ای مشاهده کردیم. در این بخش یک مدل داده ای و بر اساس آن فرم مربوطه را ایجاد خواهیم کرد.

بر روی پوشه Models کلیک راست کرده و یک کلاس به نام Person ، یک کلاس به نام Address و یک نوع داده شمارشی به نام City به صورت زیر ایجاد کنید:

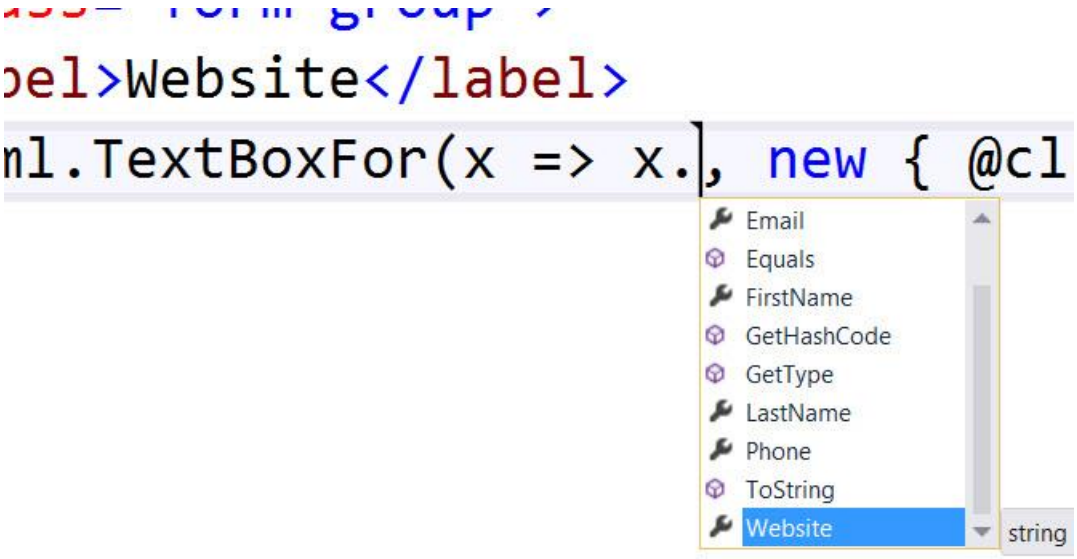
```
public enum City
    {
        Isfahan,
        Tehran,
        Mashhad
    }


public class Address
    {
        public String Street { get; set; }
        public String Blvd { get; set; }
        public City City { get; set; }
        public String PostalCode { get; set; }

    }
public class Person
    {
       public Int32 Age { get; set; }
        public String FirstName { get; set; }
        public String LastName { get; set; }
        public String Email { get; set; }
        public String Phone { get; set; }
        public String Website { get; set; }
        public Address Address { get; set; }
    }
```

حال یک ویو به نام CreatePerson ایجاد کنید و کد های آن را به صورت زیر تغییر دهید:

```
@model mkianiir.mvc.VAHM.Models.Person
@{
    ViewBag.Title = "CreatePerson";
}
@section Header{
    <h1>Create person</h1>
}
@using (Html.BeginForm())
{
<div class="form-group">
    <label>Age</label>
    @Html.TextBoxFor(x => x.Age, new { @class = "form-control" })
</div>
<div class="form-group">
    <label>FirstName</label>
```

```
    @Html.TextBoxFor(x => x.FirstName, new { @class = "form-control"
})
</div>
<div class="form-group">
    <label>LastName</label>
    @Html.TextBoxFor(x => x.LastName, new { @class = "form-control"
})
</div>
<div class="form-group">
    <label>Email</label>
    @Html.TextBoxFor(x => x.Email, new { @class = "form-control" })
</div>
<div class="form-group">
    <label>Phone</label>
    @Html.TextBoxFor(x => x.Phone, new { @class = "form-control" })
</div>
<div class="form-group">
    <label>Website</label>
    @Html.TextBoxFor(x => x.Website, new { @class = "form-control"
})
</div>

}
```

در خط اول کلاس Person به عنوان مدل داده ای ویوی CreatePerson تعریف شده است. همانطور که پیشتر گفته شد متد های راهنما علاوه بر نسخه های عادی دارای نسخه هایی هستند که به خوبی با مدل های داده ای کار می کنند. این متد ها غالبا به کلمه For ختم می شوند. به عنوان مثال در دستورات فوق از متد TextBoxFor استفاده شده است. همانطور که مشاهده می کنید در آرگومان اول این متد از عبارات لامبدا استفاده شده است. به جای اینکه نام کنترل را به صورت دستی وارد کنیم با استفاده از عبارات لامبدا می توانیم به نام خاصیت مورد نظر از مدل را به راحتی دسترسی پیدا کنیم. در این حالت ویژوال استوید نیز به شما کمک زیادی خواهد کرد. زمانی که بعد از حرف x کاراکتر dot را تایپ کنید ویرایشگر ویژوال استویو لیست خواص مربوط به مدل(در اینجا کلاس Person) را به شما نشان می دهد و می توانید از لیست نمایش داده شده خاصیت خود را انتخاب کنید. این موضوع باعث می شود که از تایپ اشتباه نام خاصیت جلوگیری شود.



شکل ۶ - ۲۹

این دست از متد ها برای سایر متد های راهنما نظیر CheckBox و ... نیز تعریف شده است که نحوه استفاده از آن ها شبیه به دستورات فوق است.

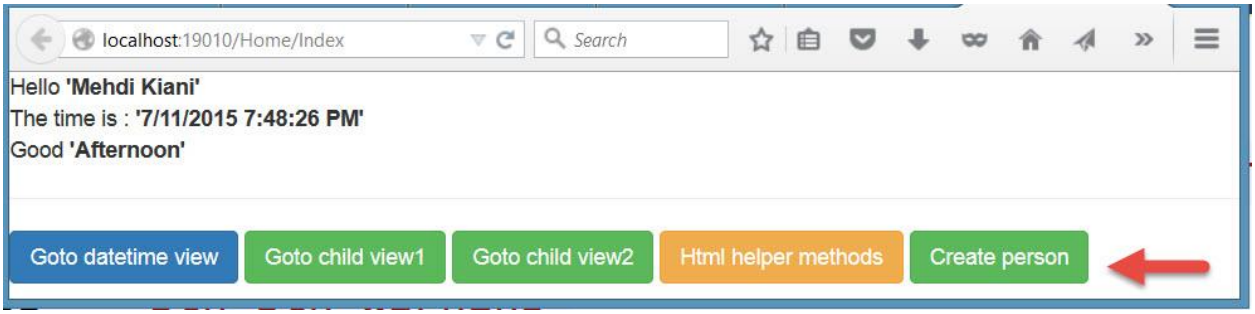
متدی به نام CreatePerson به صورت زیر در کنترلر Home ایجاد کنید:

```
public ViewResult CreatePerson()
      {
          return View();
      }
```

حال فایل Index.cshtml را باز کنید و لینک زیر را به دستورات آن اضافه کنید:

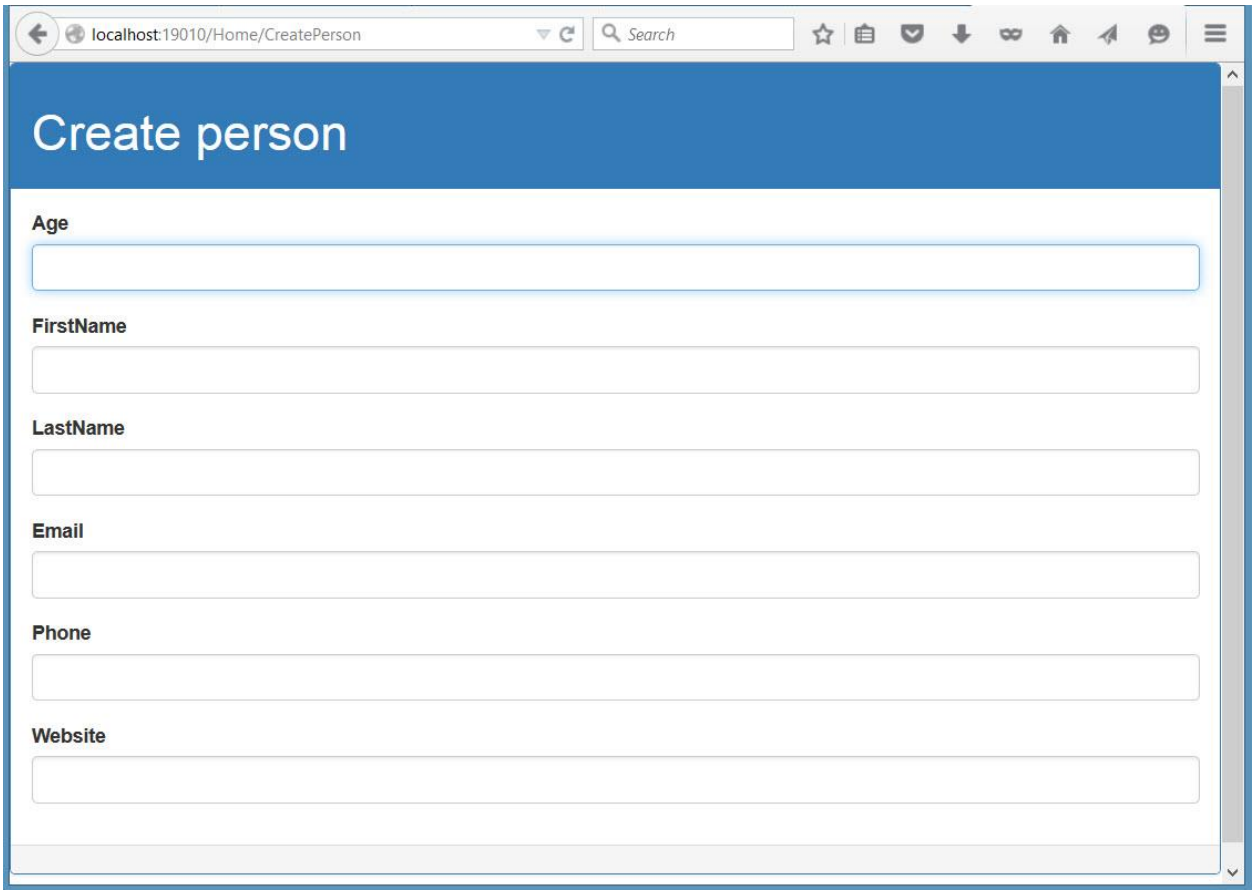
```
@Html.ActionLink("Create person", "CreatePerson", null, new
   {
       @class = "btn btn-success"
   })
```

حال برنامه را اجرا کنید و روی لینک Create person کلیک کنید:



شکل ۶ - ۳۰

اگر همه موارد را به درستی انجام داده باشید بایستی صفحه ای شبیه به عکس زیر را مشاهده کنید:



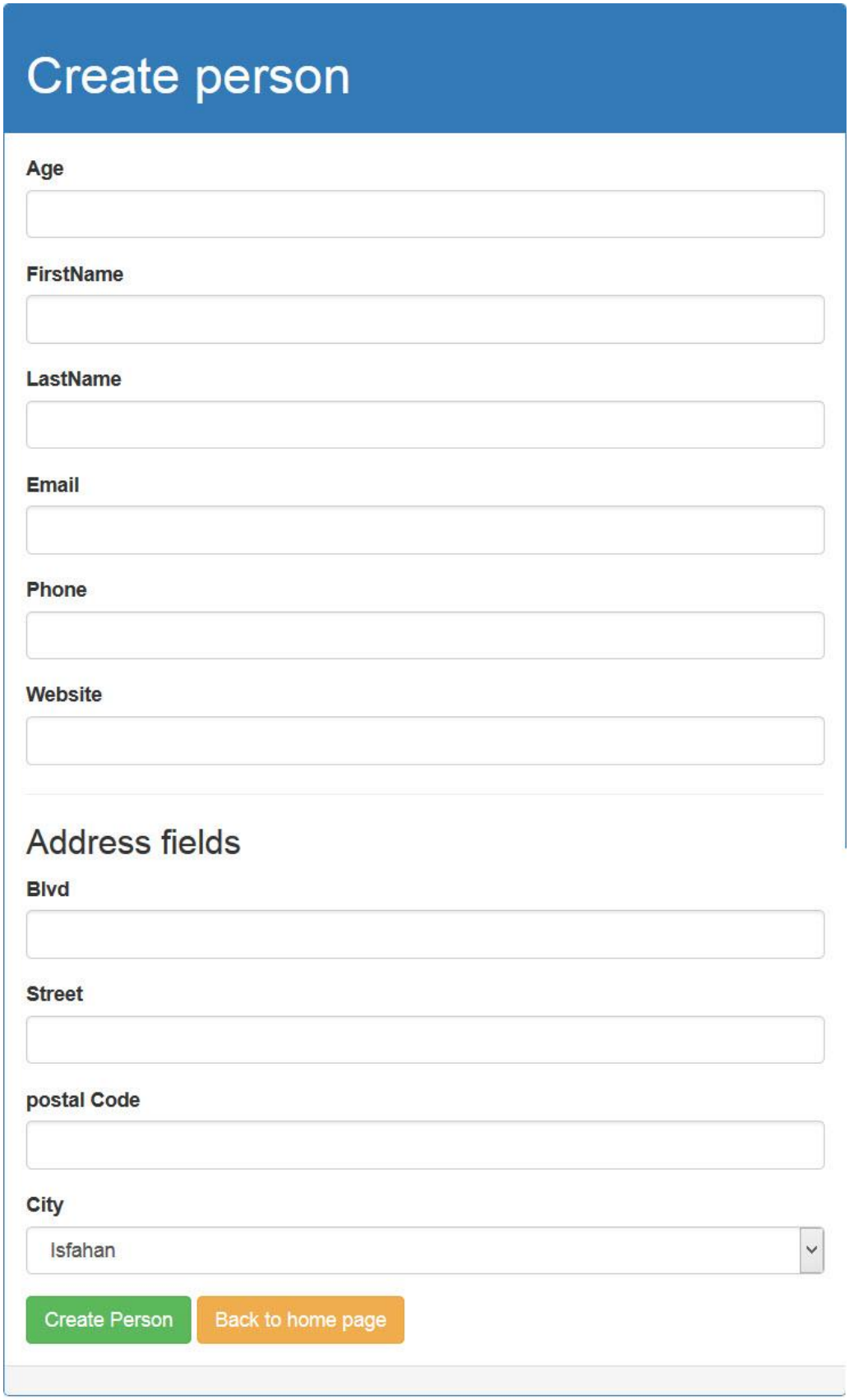
شکل ۶ - ۳۱

دستورات زیر را به ویوی CreatePerson اضافه کنید تا کنترل های مربوط به آدرس نیز اضافه شوند.

```
<hr />
<h3>Address fields</h3>
<div class="form-group">
    <label>Blvd</label>
    @Html.TextBoxFor(x => x.Address.Blvd, new
{
```

```
    @class = "form-control"
})
</div>
<div class="form-group">
    <label>Street</label>
    @Html.TextBoxFor(x => x.Address.Street, new
{
    @class = "form-control"
})
</div>
<div class="form-group">
    <label>postal Code</label>
    @Html.TextBoxFor(x => x.Address.PostalCode, new
{
    @class = "form-control"
})
</div>
<div class="form-group">
    <label>City</label>
    @Html.DropDownListFor(x => x.Address.City,new
SelectList(Enum.GetNames(typeof(mkianiir.mvc.VAHM.Models.City))),
new
{
    @class = "form-control"
})
</div>
<input type="submit" name="CreatePerson" value="Create Person"
class="btn btn-success"/>
@Html.ActionLink("Back to home page", "Index",null, new
{
    @class="btn btn-warning"
})
```

اگر مجددا برنامه را اجرا کنید باید فرمی شبیه به فرم زیر داشته باشید:


Create person
Age
FirstName
LastName
Email
Phone
Website
Address fields
Blvd
Street
postal Code
City
Isfahan
Create Person
Back to home page

شکل ۶ - ۳۲

به دستور @Html.DropDownList توجه کنید. همانطور که پیشتر نیز اشاره شد مقادیر آیتم های این کنترل به روش های مختلفی می توانند مقدار دهی شوند. چون فیلد City از نوع داده شمارشی می باشد بنابر این با استفاده از کلاس Enum و دریافت لیست نام های آیتم های داده شمارشی مقادیر کنترل DropDown استفاده شده اند.

ارسال اطلاعات از ویو به کنترلر

یک ویو به نام PersonDetails ایجد کنید و دستورات آن را به صورت زیرتغییر دهید:

```
@model mkianiir.mvc.VAHM.Models.Person
@{
    ViewBag.Title = "PersonDetail";
}

<h3>Person cretaed :</h3>


@Html.LabelFor(x=>x.FirstName)
@Html.DisplayFor(x=>x.FirstName)
<br/>
@Html.LabelFor(x => x.LastName)
@Html.DisplayFor(x => x.LastName)
<br />
@Html.LabelFor(x => x.Age)
@Html.DisplayFor(x => x.Age)

<br />
@Html.LabelFor(x => x.Email)
@Html.DisplayFor(x => x.Email)
<br />
@Html.LabelFor(x => x.Phone)
@Html.DisplayFor(x => x.Phone)
<br />
@Html.LabelFor(x => x.Website)
@Html.DisplayFor(x => x.Website)
```

متد راهنمای LableFor برای ایجاد تگ label برای خاصیت داده شده به کار می رود و متد راهنمای DisplayFor برای نمایش مقدار خاصیت موجود در مدل.

ایجاد متد های راهنمای سفارشی

علاوه بر متد های راهنمای بسیاری که وجود دارد و می توانید از آن ها استفاده کنید mvc به شما این اجازه را می دهد تا بتوانید متد های راهنمای سفارشی خودتا ن را بنویسید.

به دو روش می توانید اقدام به ایجاد یک متد راهنمای سفارشی(اختصاصی) کنید.

۱- استفاده از دستور @helper در ویو ها(به این نوع متد راهنمای داخلی "inline" می گویند)

۲- استفاده از یک کلاس و ایجاد یک متد الحاقی برای کلاس HtmlHelper همانند کاری که برای تمامی متدهای راهنمای پیش فرض mvc انجام شده است(به این نوع متدها، متد راهنمای خارجی "external" می گویند)

در اینجا یک متد راهنما برای نمایش مقادیر آدرس از مدل Person ایجاد می کنیم.

دستورات زیر را در فایل PersonDetails و پس از بلاکی که در آن دستور ViewBag.Title قرار دارد تعریف کنید:

```
@helper ShowAddress(mkianiir.mvc.VAHM.Models.Address address)
{
    <label>Address : </label>@(address.City) <text>,</text>
@(address.Blvd) <text>,</text>
@(address.Street)<text>,</text>@(address.PostalCode)
}
```

دستورات فوق یک متد راهنما به نام ShowAddress ایجاد می کند.

حال دستور زیر را در ادامه دستورات قبلی در فایل PersonDetails تعریف کنید:

```
<br />
@ShowAddress(Model.Address)
```

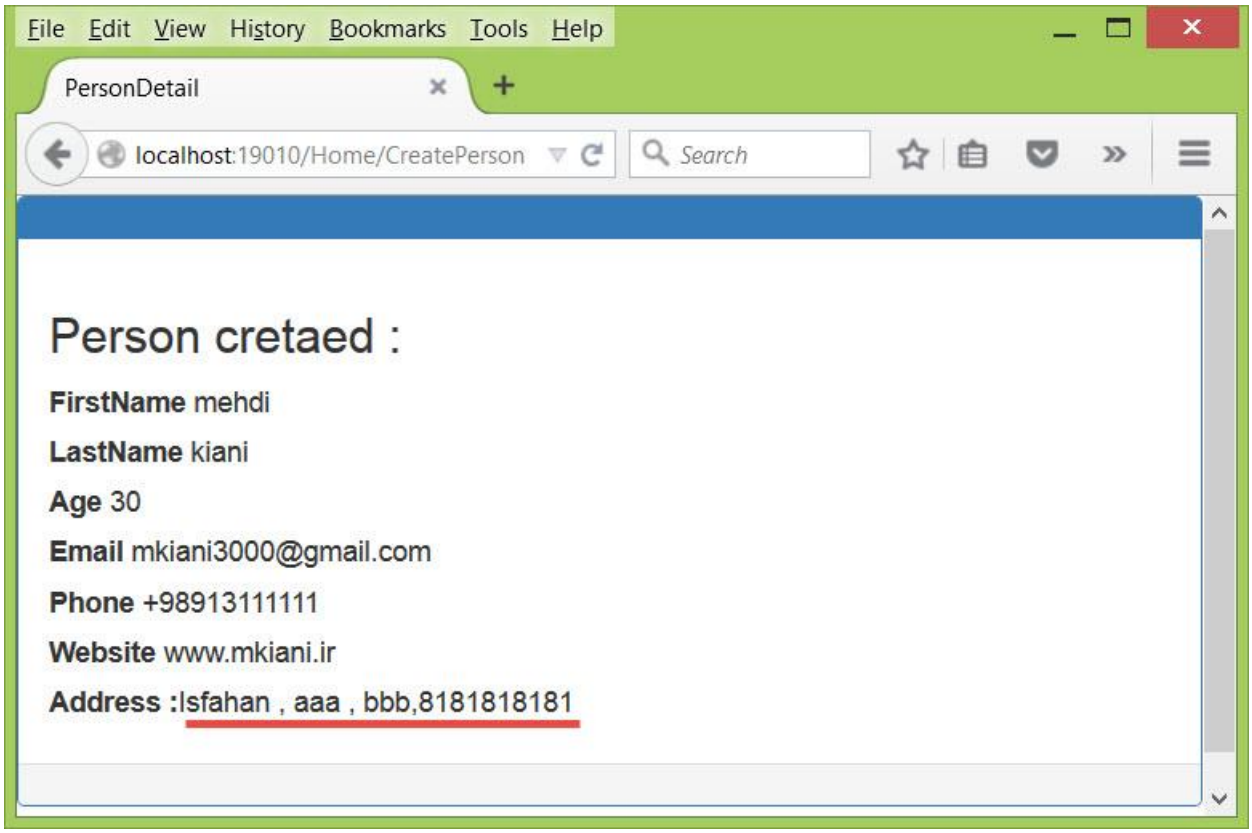
همانطور که مشاهده می کنید همانند سایر متدهای راهنما با استفاده از کاراتر @ می توانیم به اکشن متد سفارشی نوشته شده دسترسی پیدا کنیم و آن را فراخوانی کنیم. مقدار Address از مدل به متد مذکور ارسال شده است.

یک اکشن متد دیگری به نام CreatePerson به صورت زیر در کنترلر Home تعریف کنید:

```
[HttpPost]
      public ViewResult CreatePerson(Person model)
      {
          return View("PersonDetails",model);
      }
```

دقت کنید که صفت HttpPost برای این متد اضافه شده باشد. زمانی که فرم Person به کنترلر ارسال می شود ، متد مذکور فراخوانی شده و توسط کامپوننت های Value Provider و Model Binder که قبلا توضیح داده شدند اطلاعات وارد شده توسط کاربر تحت یک نمونه از کلاس Person به این متد ارسال می شود. این متد نیز ویوی PersonDetails را انتخاب و اطلاعات دریافتی را به آن ارسال می کند.

حال برنامه را اجرا کنید. بر روی لینک Create person کلیک کنید. فرم Create person را تکمیل کنید و سپس بر روی دکمه send کلیک کنید. اگر همه موارد را به درستی انجام داده باشید نتجه ای مطابق شکل زیر خواهید داشت.



شکل ۶ - ۳۳

همانطور که مشاهده می کنید متد ShowAddress خروجی آدرس را به صورتی که برایش تعریف شده بود رندر کرده است.

ایجاد متد راهنمای سفارشی(حالت external)

در بخش قبلی متد راهنمای ShowAddress به صورت داخلی برای ویوی CreatePerson تعریف شد. چنانچه بخواهید متد راهنمایی تعریف کنید که بتوانید به صورت عمومی (در طول پروژه) از آن بهره ببرید می بایستی یک متد الحاقی برای کلاس HtmlHelper تعریف کنید.

پوشه ای به نام Core درون Solution Exploere ایجاد کنید.حال بر روی پوشه Core کلیک راست کرده و از منوی Add گزینه کلاس را انتخاب کنید.درپنجره Add New Item باز شده نام کلاس را PersonHelperExtenssions قرار داده و دستورات زیر را برای آن بنویسید:

```
namespace mkianiir.mvc.VAHM.Core
{
    public static class PersonHelperExtensions
    {
        public static MvcHtmlString ShowAddress(this HtmlHelper
helper,Address model)
        {
            return new MvcHtmlString(String.Format("Address :
{0},{1},{2},{3}",model.City,model.Blvd,model.Street,model.PostalCode
));
        }
    }
}
```

حال ویوی PersonDetails را باز کنید و دستور زیر را در ابتدای آن اضافه کنید:

```
@using mkianiir.mvc.VAHM.Core;
```

همچنین به جای دستور

```
@ShowAddress(Model.Address)
```

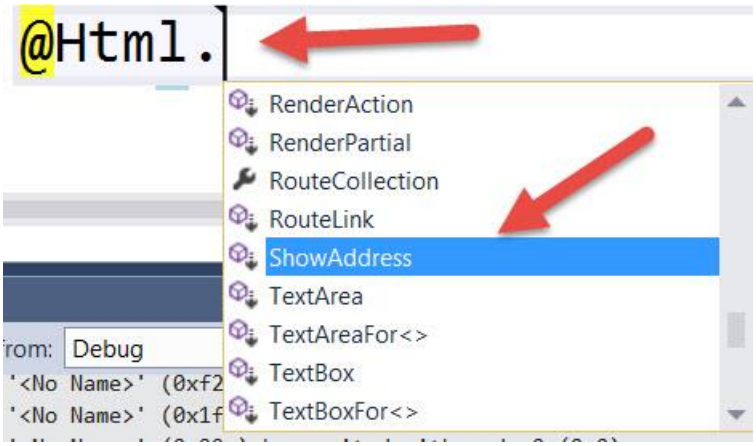
از دستور

```
@Html.ShowAddress(Model.Address)
```

استفاده کنید.

در دستورات فوق یک متد الحاقی به نام ShowAddress برای کلاس HtmlHelper تعریف کردیم. در ویوی مورد نظر توسط دستور using فضای نامی که کلاس حاوی متد الحاقی در آن وجود دارد را اضافه کردیم. سپس با استفاده از دستور @Html به متد ShowAddress دسترسی پیدا کرده و آن را فراخوانی کرده ایم.

تصویر زیر نشان می دهد که متد ShowAddress به عنوان یک متد الحاقی برای کلاس HtmlHelper تشخیص داده شده است:



شکل ۶ - ۳۴

خلاصه

در فصل پایانی کتاب توضیحات نسبتا کاملی در رابطه ویو ها، ویو های جزئی(Partial View) و همچنین متد های راهنمای html (Html Helper Methods) ارائه شد. همچنین در بخش پایانی فصل روش ایجاد متد های راهنمای سفارشی را فرا گرفتید که امیدوارم بتوانید از این مطالب در پروژه های خود بهره برداری کنید.

منابع

1- Professional ASP.NET MVC 3,John Galloway,Phil Haak,Brad Wilson,K. Scott Allen, Worx Publishing
2- Beginning ASP.NET MVC 4, José Rolando Guay Paz , Apress Publishing
3- Professional ASP.NET MVC 4,John Galloway,Phil Haak,Brad Wilson,K. Scott Allen, Worx Publishing
4- Pro ASP.NET MVC 4, Adam Freeman,Apress Publishing
5- Pro ASP.NET MVC 5, Adam Freeman,Apress Publishing
6- http://www.asp.net/mvc