آموزش کاربردی

Pro
ASP.NET Core MVC

ویراست ۶

جلد ۱

آدام فریمن

ترجمه: مهندس نادر نبوی

انتشارات پندار پارس

سرشناسه	:	فریمن، آدام، ۱۹۷۲ - م. Freeman, Adam
عنوان و نام پدیدآور	:	آموزش کاربردی ASP.NET Core MVC/ آدام فریمن ؛ ترجمه نادر نبوی.
مشخصات نشر	:	تهران : پندار پارس ، ۱۳۹۶.
مشخصات ظاهری	:	۲ ج.: مصور، جدول.
شابک	:	978-600-8201-37-3: ۲۳۰۰۰۰ ریال
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	عنوان اصلی:.Pro ASP.NET Core MVC,6th ed,2016
موضوع	:	مایکروسافت دات‌نت فریمورک
موضوع	:	Microsoft .NET Framework
موضوع	:	وبگاه‌ها -- برنامه‌های تالیفی
موضوع	:	Web sites -- Authoring programs
موضوع	:	علوم کامپیوتر
موضوع	:	Computer science
موضوع	:	نرم‌افزار -- مهندسی
موضوع	:	Software engineering
شناسه افزوده	:	نبوی، نادر، ۱۳۴۰ -، مترجم
رده بندی کنگره	:	۱۳۹۶۷۶/۷۶QA ۴ف۲۳م/
رده بندی دیویی	:	۰۰۵/۲۷۶
شماره کتابشناسی ملی	:	۴۶۶۵۷۱۵

انتشارات پندارپارس

دفتر فروش: انقلاب، ابتدای کارگرجنوبی، کوی رشتچی، شماره ۱۴، واحد ۱۶ **www.pendarepars.com**
تلفن: ۶۶۵۷۲۳۳۵ – تلفکس: ۶۶۹۲۶۵۷۸ همراه: ۰۹۲۱۴۳۷۱۹۶۴ **info@pendarepars.com**

نام کتاب : **آموزش کاربردی ASP.NET Core MVC (جلد ۱)**

ناشر : **انتشارات پندار پارس**

تالیف : **Adam Freeman**

ترجمه : **نادر نبوی**

چاپ نخست : **اردیبهشت ۹۶**

شمارگان : **۵۰۰ نسخه**

طرح جلد : **سارا یعسوبی**

چاپ، صحافی : **روز**

قیمت : **۲۳۰۰۰ تومان** **شابک : ۳-۳۷-۸۲۰۱-۶۰۰-۹۷۸**

فهرست

فهرست فصل‌های جلد ۲

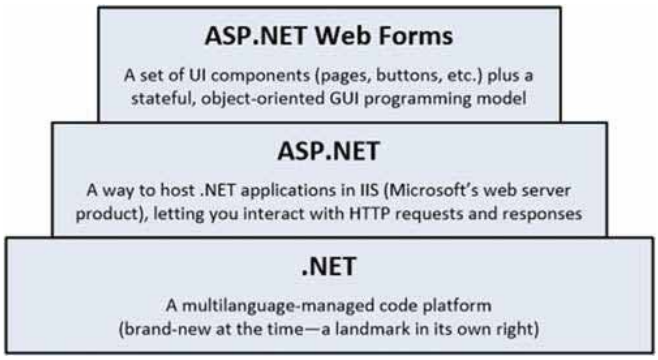
فصل یکم

آشنایی با ASP.NET Core MVC

ASP.NET Core MVC چارچوب توسعه‌ی برنامه‌های کاربردی[1] مایکروسافت است که اثربخشی و سازماندهی مناسب معماری مدل-نما-کنترلر[2] را با بهترین بخش‌های .NET. در هم آمیخته است. در این بخش، همراه با فهم چرایی کاربرد ASP.NET Core MVC، مقایسه‌ای خواهیم داشت با دیگر معماری‌های مشابه آن و در پایان، با تازه‌های افزوده شده به این معماری و آنچه که در این کتاب مورد بررسی قرار خواهد گرفت، آشنا خواهید شد.

۱ تاریخ ASP.NET Core MVC

در سال ۲۰۰۲ که مایکروسافت سعی بر حفظ موقعیت برتر خود در دنیای پروژه‌های دسکتاپ و ویندوز داشت، متوجه خطر توسعه‌ی نرم‌افزارهای وب شد و در پی آن ASP.NET را به عنوان چارچوب توسعه‌ی وب، عرضه کرد. شکل ۱-۱، وضعیت فن‌آوری مایکروسافت را در آن زمان نشان می‌دهد.



شکل ۱-۱

۱-۱ پروژه‌های فرم‌های وب

معرفی پروژه‌های فرم‌های وب، روشی بود که مایکروسافت برای دور زدن پروتکل انتقال ابرمتن (HTTP)، به دلیل ناماندگاری[3] آن، و زبان علامت‌گذاری ابرمتن (HTML که در آن زمان هنوز برای برنامه‌نویسان مفهوم آشنایی نبود)، به وسیله‌ی معرفی صفحه‌ای به نام فرم وب، شامل سلسله‌مراتبی

---

[1] Application DevelopmentFramework

[2] Model-View-Controller (MVC)

[3] صفحات منتقل شده در این پروتوکل Stateless هستند. یعنی برخلاف فرم‌های ویندوز، وقتی در وب از صفحه‌ای به صفحه‌ی دیگر حرکت می‌کنید، عملا همه‌ی اطلاعات صفحه‌ی پیشین را از دست می‌دهید، م.

از کنترل‌های سمت وب[1]، در پیش گرفت. در دریافت درخواست مرورگر و پاسخ سرور، هر کنترل مسئول نگهداری وضعیت (State) خود بود و رویدادهای (Events) سمت مشتری، مانند کلیک بر روی یک دکمه، به صورت خودکار به کد رویدادهایی (Event Hnadlers) که باید در سمت سرور اجرا می‌شدند، متصل بود. در نتیجه، فرم‌های وب، لایه‌ای غول‌آسا برای انتقال رابط کاربری کلاسیک (GUI) بر روی بستر وب هستند.

هدف اصلی این بود که برنامه‌نویسی وب تا جای ممکن شبیه برنامه‌نویسی ویندوز شود. برنامه‌نویسان می‌توانستند بر پایه‌ی یک رابط گرافیکی که به خوبی توانایی حفظ وضعیت خود را در حرکت بین فرم‌ها دارا بود، فکر کنند و نیازی به درگیری با درخواست‌های HTTP و پاسخ‌های آنها، نبود. به این ترتیب، هدف مایکروسافت در انتقال جمعیت عظیم برنامه‌نویسان ویندوز به سمت برنامه‌نویسی وب، در حال عملی شدن بود.

۱-۱-۱ اشکال پروژه‌های فرم‌های وب

پایه‌های تفکر توسعه‌ی فرم‌های وب اشکالی نداشت، ولی در عمل مشکلاتی در این زمینه پیدا شد:

- حجم وضعیت فرم: روشی که در حفظ وضعیت فرم‌ها در پیش گرفته شده بود، موجب انتقال حجم بزرگی از داده‌ها بین سرور و مشتری می‌شد. حتی در فرم‌های وب معمولی، حجم این داده‌ها می‌توانست به چند صد کیلو بایت برسد. این داده‌ها می‌بایست در هر درخواست و پاسخ، بین سرور و مشتری حرکت کنند که نتیجه، پهنای باند وسیع مورد نیاز بر روی سرور، و کندی زمان پاسخ‌گویی بود.
- چرخه‌ی عمر صفحه: روش ارتباط دادن بین رویدادهای سمت مشتری با کد مدیریت این رویدادها (Event Handlers) به عنوان بخشی از چرخه‌ی عمر صفحه‌ی وب، بر روی سرور، پیچیده بود.
- پیاده نشدن جداسازی دغدغه‌ها[2]: روش کدنویسی فرم‌های وب، ظاهرا می‌خواهد کد سمت سرور را از HTML صفحه جدا کند و آن را در کلاسی جداگانه به نام کد پشتی (Code Behind) قرار دهد. این کار برای جداسازی کد منطق برنامه از کد رابط کاربری انجام شده بود. با این حال، برنامه‌نویسان در عمل وادار به ترکیب کد سمت سرور (مثلا کد C# یا VB) با کد HTML بودند.
- نبود کنترل بر HTML: کنترل‌های سمت سرور در انتقال به سمت مشتری تبدیل به کد HTML می‌شدند ولی این HTML همانی نبود که برنامه‌نویس انتظار داشت. در روزهای نخستین فرم‌های

---

[1] Server side Controls روش نخستین مایکروسافت در ایجاد کنترل‌ها بر روی سرویس‌دهنده و ارسال ترجمه‌ی HTML آنها به سمت مشتری. مایکروسافت در حال حاضر خودش این روش را نادرست معرفی می‌کند و در معماری‌های جدید از آن استفاده نمی‌کند م.

[2] Separation of Concerns

وب، HTML تولید شـده با اسـتانداردهای وب همخوانی نداشـت و به سـختی می‌توانسـت با CSS شـکل‌دهی شـود. افزون بر این، IDهای ایجاد شـده در کد HTML با جاوا اسـکریپت قابل دسـتیابی نبودند.

- آزمـایش ناپذیری: طراحان فرم‌های وب نمی‌توانسـتند پیش‌بینی کنند که زمانی آزمایش کد، بخش مهمی از توسـعه‌ی هر نرم‌افزاری می‌شـود. معماری ارائه شده توسط آنها در فرم‌های وب، به هیچ روی مناسب روش‌های امروزی آزمایش کد نیست.

باید یادآور شـد که توسعه‌ی فرم‌های وب چندان هم بد نبود. مایکروسافت تا جای ممکن برای هماهنگی با استانداردهای وب و آسان کردن برنامه‌نویسی آن، کوشش کرد و حتی بعدها از برخی از ویژگی‌های نخسـتین ASP.NET MVC در معماری فرم‌های وب هم اسـتفاده کرده اسـت. هنوز هم اگر نیاز سـریع به برنامه‌ی وبی داشـته باشـید و به گفته‌ی برنامه‌نویسان بخواهید در کمترین زمان سایت را بالا بیاورید، فرم‌های وب حرف اول را می‌زنند.

۲-۱ پروژه‌های MVC قدیمی

در اکتبر سـال ۲۰۰۷، مایکروسـافت، برای پاسـخ‌گویی به انتقادهای فرم‌های وب و محبوب بودن برخی روش‌های توسـعه‌ی جدید مانند Ruby، سـکوی توسـعه‌ی[1] جدیدی بر پایه‌ی ASP.NET را اعلام کرد. سـکوی جدید، ASP.NET MVC نام داشـت و هدف آن، برآورده کردن استانداردهای نوین توسعه‌ی وب، مانند CSS و HTML، سـرویس‌های REST، آزمایش‌پذیری کد و تسلیم بر این باور بود که برنامه‌نویسان باید بدون وضعیت (Stateless) بودن ذاتی HTTP را بپذیرند.

معرفی MVC، تغییر دیگری را هم در سـیاسـت‌های پیشـین مایکروسـافت موجب شـد. پیش از آن، مایکروسافت تاکید زیادی بر کنترل اجزای ایجاد کننده‌ی نرم‌افزار داشت. ولی با معرفی MVC، و کاربرد ابزار متن‌بازی (Open Source) مانند jQuery در آن، مایکروسافت سعی کرد از بهترین‌های رقبای خود اسـتفاده کند و از همین روی، متن کد اصـلی MVC را در اختیار برنامه‌نویسـان به عنوان کاربران اصلی MVC، قرار داد.

۲-۱-۱ *مشکل پروژه‌های قدیمی MVC*

در زمان ایجاد MVC برای مایکروسـافت طبیعی بود که نرم‌افزار جدید را بر پایه‌ی ASP.NET بر پا کند. تغییرات زیادی لازم بود تا MVC بتواند بر پایه‌ی نرم‌افزاری که برای فرم‌های وب طراحی شده بود، کار

[1] Development Platform

کند. بنابراین طراحان اولیه‌ی MVC مجبور به تغییر تنظیمات و پیکربندی‌هایی شـدند که ربطی به اجرای کد MVC نداشتند ولی به هر حال برای این که همه چیز به درستی کار کند، لازم بودند.

با محبوب شدن MVC، مایکروسـافت شـروع به اسـتفاده از برخی ویژگی‌های پایه‌ی آن در فرم‌های وب کرد. در نتیجـه، کـدی که به خودی خود دارای ویژگی‌های عجیبی بود (زیرا بر پایه‌ی ASP.NET و برای MVC طراحی شده بود) در کاربرد برای توسعه‌ی فرم‌های وب، با تلاش بر این که، همه چیز به هر حال باید کار کن، منجر به کدی پیچیده‌تر و عجیب‌تر شـد. در همان زمان، مایکروسـافت شـروع به گسترش ASP.NET با ویژگی‌های جدیدی مانند سـرویس‌ها وب (Web Services) کرد که هر یک نیاز به پیکربندی و تنظیمات مخصوص به خود داشت. نتیجه‌ی پایانی، کدی پاره پاره و بدفرم بود.

۱-۳ فهم ASP.NET Core

در سـال ۲۰۱۵ مـایکروسـافت خبر از مسـیرهـای جدیدی برای ASP.NET و MVC داد، که منجر به ASP.NET Core MVC، یعنی موضوع مورد بررسی این کتاب شد.

ASP.NET Core بر پایه‌ی .NET Core. استوار است که نگارشی از .NET.، مستقل از سیستم عامل و بدون واسـط برنامه‌نویسـی ویندوز اسـت. ویندوز هنوز هم سـیسـتم عاملی برتر به حسـاب می‌آید؛ ولی برنامه‌های وب نه تنها روز به روز از کاربرد و محبوبیت بیشـتری برخوردار می‌شـوند، بلکه بر روی سـکوهای دیگری مانند فضـای ابری (Cloud) هم باید بتوانند میزبانی (Host) شوند. مایکروسافت با این کـار، گسـتره‌ی کـارکرد .NET را افزایش داده اسـت؛ بـه این معنی کـه می‌توان برنـامه‌های کاربردی ASP.NET Core را بر روی بـازه‌ی گسـترده‌ای از محیط‌هـای مختلف میزبانی کرد. هم اینک، می‌توانید پروژه‌های وب ASP.NET Core را برای Linux یا macOS هم تولید کنید.

ASP.NET Core، در مقایسـه با MVC نخسـتین، سـاده‌تر اسـت و برخلاف آن هیچ ارتباطی با پروژه‌های فرم‌های وب ندارد و از آنجا که بر پایه‌ی .NET Core بنا شـده اسـت، ایجا پروژه‌های وب را بر روی بسیاری از سیستم‌عامل‌های گوناگون پشتیبانی کرده و امکان میزبانی در محیط‌های مختلف را داراست.

ASP.NET Core همه‌ی امکانات ASP.NET MVC را بر پایه‌ی سکوی جدید ASP.NET Core ، فراهم می‌کند. افزون بر این که شـامل همه‌ی کارآیی‌های واسـط کاربری وب اسـت، روش‌های طبیعی‌تری برای ایجاد محتوای پیچیده پیشـنهاد می‌کند و امکان می‌دهد که بسـیاری از کارهای کلیدی مربوط به توسعه، مانند آزمایش‌های واحد، به سادگی انجام شوند.

۱-۳-۱ مزایای اصلی ASP.NET Core

در بخش‌هـایی کـه در ادامـه آورده می‌شـوند، نشـان خواهیم داد که چگونه سـکوی جدید MVC، بر پروژه‌های قدیمی فرم‌های وب و MVC نخستین، برتری یافته است.

۱-۱-۳-۱ معماری MVC

ASP.NET Core MVC از الگوی مدل-نما-کنترلر برای ایجاد پروژه‌های وب استفاده می‌کند. در اینجا باید بین الگوی معماری MVC و پیاده‌سازی ASP.NET Core MVC از این الگو، تفاوت قائل شویم. الگوی MVC جدید نیست و به سال ۱۹۷۸ و پروژه‌ی Smalltalk بازمی‌گردد. از آن زمان تا کنون، به دلایلی که در ادامه آورده می‌شوند، در ایجاد پروژه‌های وب محبوبیت زیادی پیدا کرده است.

- برخورد کاربر با برنامه‌ای کاربردی که از الگوی MVC استفاده کرده است، مسیری طبیعی را طی می‌کند؛ کاربر عملی را انجام می‌دهد و در پاسخ به آن، برنامه با تغییر مدل داده‌ها، نمای جدیدی را نمایش می‌دهد. این چرخه به همین صورت ادامه پیدا می‌کند. این روش برای برنامه‌های وب، به عنوان مجموعه‌ای از درخواست‌ها و پاسخ‌های HTTP.، مناسب است.
- در برنامه‌های کاربردی وب با فن‌آوری‌های گوناگونی مانند پایگاه‌های داده، HTML و کد اجرایی، سروکار پیدا می‌کنیم که در عمل، به صورت لایه‌های جدای پروژه پیاده‌سازی می‌شوند. الگوی به دست آمده از این ترکیب با تفکری که MVC بر پایه‌ی آن ایجاد شده، هماهنگی دارد.

از آنجا که ASP.NET Core MVC الگوی MVC را پیاده می‌کند، در مقایسه با پروژه‌های قدیمی فرم‌های وب، موضوع جداسازی لایه‌های پروژه را به خوبی انجام می‌دهد. در فصل سوم، این معماری را با جزئیات بیشتری بررسی خواهیم کرد.

۲-۱-۳-۱ گسترش‌پذیری

ASP.NET Core و ASP.NET Core MVC شامل عناصر (Component) مستقلی هستند. این عناصر دارای مشخصه‌های روشنی بوده و معمولا از یک واسط (Interface) و یا یک کلاس مجرد (Abstract Class) ارث‌بری کرده‌اند. به سادگی می‌توانید هر یک از این عناصر را با آنچه که خودتان پیاده‌سازی کرده‌اید، جایگزین کنید. برای هر عنصر، سه انتخاب در پیش رو دارید:

- پیاده‌سازی پیش‌فرض آن را، همان‌گون که هست به کار برید که برای بسیاری از برنامه‌ها کافی است.
- کلاس دیگری از پیاده‌سازی پیش‌فرض مشتق کنید و رفتار آن را به دلخواه تغییر دهید.
- عنصر مورد نظر را، با ایجاد کلاس جدیدی که از واسط یا کلاس مجرد نخستین ارث‌بری می‌کند، کاملا جایگزین کنید.

در فصل ۱۴ با این روش‌ها آشنا خواهید شد.

۳-۱-۳-۱ کنترل کامل بر HTML و HTTP

خروجی HTMLای که ASP.NET Core MVC ایجاد می‌کند، کاملا استاندارد است. برای شکل دادن به این HTML می‌توانید از سبک‌های CSS استفاده کنید. افزون بر این، می‌توانید از jQuery، Angualr یا Bootstrap برای ایجاد عناصر پیچیده‌ی نما، مانند تقویم یا منوهای تو در تو استفاده کنید.

ASP.NET Core MVC با HTTP هماهنگ است. به این معنی که کنترل درخواست‌های ارسال شده از مرورگر به سرور را در دست دارید. این، امکان می‌دهد که تجربه‌ی کاربر از برنامه را آن‌گونه که می‌خواهید شکل دهید. کاربرد Ajax ساده شده است و همان‌طور که در فصل ۲۰ آمده است، به راحتی می‌توانید از سرویس‌های وب (Web Service) برای دریافت درخواست‌های ارسالی از مرورگر استفاده کنید.

۴-۱-۳-۱ آزمایش‌پذیری

از آنجا که در معماری ASP.NET Core MVC رابط کاربری، مدل داده‌ها و کد پردازش کننده به خوبی از هم جدا شده‌اند، زمینه‌ی خوبی برای اجرای آزمایش‌های واحد (Unit Tests) فراهم می‌شود. این کار را می‌توانید با هر یک از نرم‌افزارهای آزمایش متن‌باز موجود، مانند xUnit.net که در فصل ۷ معرفی خواهد شد، انجام دهید.

در این کتاب خواهید دید که چگونه می‌توان با استفاده از روش‌های شبیه‌سازی گوناگون برای ایجاد پیاده‌سازی‌های ساختگی[1] از عناصر پروژه، کدهای آزمایشی ساده و روشن برای کنترلرها و متدهای اکشن، نوشت.

آزمایش‌پذیری تنها به ایجاد آزمایش‌های واحد مربوط نمی‌شود. برنامه‌های کاربردی ASP.NET Core MVC با ابزاری که برای آزمایش خودکار واسط کاربری به کار می‌روند، به خوبی کار می‌کنند. بدون نیاز به دانستن ساختار عناصر HTML، CSS و یا IDهایی که برنامه ایجاد می‌کند، می‌توانید کدی بنویسید که برخورد کاربر با برنامه را شبیه‌سازی کند.

۵-۱-۳-۱ روش مسیریابی قوی

روش استفاده از URLها، با تکامل فن‌آوری وب، تغییر کرده است. آدرس‌هایی مانند:

```
/App_v2/User/Page.aspx?action=show%20prop&prop_id=82742
```

به ندرت پیدا می‌شوند و به جای آنها از آدرس‌های مشخص‌تری مانند:

```
/to-rent/chicago/2303-silver-street
```

استفاده می‌شود.

---

[1] Fake Implementations

چگونگی ساختار URL دلایل زیادی دارد. نخست این که موتورهای جست‌وجو به واژه‌های کلیدی موجود در آدرس‌های اینترنتی اهمیت می‌دهند. دیگر این که بسیاری از کاربران اکنون معنی آدرس‌های اینترنتی را می‌دانند و ترجیح می‌دهند خودشان آن را در نوار آدرس مرورگر وارد کنند. افزون بر این، زمانی که شخصی معنی یک آدرس اینترنتی را به روشنی بفهمد، تمایل بیشتری برای سهیم شدن آن با دیگران و یا کاربرد لینک آن در صفحه‌ی وب خود خواهد داشت. مهم‌تر از همه‌ی آنها این است که کاربرد چنین URLهایی ساختار پوشه‌ها و فایل‌های پروژه را برای دیگران آشکار نمی‌کند و در صورت تغییر پیاده‌سازی ساختار برنامه، در مورد شکسته شدن آدرس‌ها و یا لزوم تغییر آنها، نگرانی نخواهید داشت.

استفاده از URLهای روشن، مانند آنچه که در بالا آمد، در چارچوب‌های وب قدیمی مشکل بود. ولی ASP.NET Core MVC از روش آدرس‌دهی پیشرفته‌ای به نام مسیردهی URL (URL Routing) استفاده می‌کند، که به طور پیش‌فرض چنین URLهایی را به کار می‌برد. این امکان می‌دهد که کنترل خوبی بر ساختار آدرس‌های صفحات خود داشته باشید، و URLهایی با معنی و روشن در اختیار کاربران خود قرار دهید.

۶-۱-۳-۱ واسط برنامه‌نویسی قوی

از آنجا که ASP.NET Core MVC بر پایه‌ی .NET Core بنا شده است، از بسیاری از ویژگی‌های قدرت‌مندآن که برای برنامه‌نویسان C# آشنا هستند مانند، کاربرد await، متدهای توسعه یافته[1]، عبارت‌ها لاندا[2]، انواع پویا و بی‌نام[3] و کوئری آمیخته با زبان[4] (LINQ) استفاده می‌کند.

۷-۱-۳-۱ چند سکویی[5]

نگارش‌های گذشته‌ی ASP.NET، هم برای نوشتن برنامه نیازمند ویندوز بودند و هم برای میزبانی به سرور ویندوز نیاز داشتند. ASP.NET Core، هم برای برنامه‌نویسی و توسعه و هم برای انتشار به محیط ویژه‌ای وابسته نیست.

---

[1] Extension Methods
[2] Lamda Expressions
[3] Anonymous and Dynamic Types
[4] Language Integrated Query (LINQ)
[5] Cross Platform

۷-۱-۳-۱ متن باز بودن

بر خلاف چارچوب‌های گذشته‌ی توسعه‌ی مایکروسافت، اکنون می‌توانید کد سورس ASP.NET Core و ASP.NET Core MVC را دانلود کنید و حتی پس از انجام تغییرات و کامپایل، نگارش خودتان از آنها را به کار برید. هر دوی این نرم‌افزارها از آدرس https://github.com/aspnet قابل دانلود هستند.

۱-۴ نیازمندی‌ها

برای این که بهترین استفاده را از این کتاب ببرید، باید افزون بر آشنایی با مفاهیم پایه‌ی برنامه‌نویسی و توسعه‌ی وب، HTML و CSS دانشی از برنامه‌نویسی با زبان C# داشته باشید.

اگر با برنامه‌نویسی سمت مشتری مانند جاوا اسکریپت به خوبی آشنایی ندارید، نگران نباشید. در این کتاب بر برنامه‌نویسی سمت سرور تمرکز خواهیم داشت و آنچه که در مورد جاوا اسکریپت لازم است را از کد مثال‌ها خواهید آموخت. در فصل چهارم، بررسی سریعی بر مهم‌ترین و جدیدترین ویژگی‌های C# خواهیم داشت. اگر تنها با برنامه‌نویسی سنتی C# کار کرده‌اید، توصیه می‌شود این فصل را به دقت بخوانید.

۱-۵ ساختار کتاب

این کتاب به دو بخش اصلی تقسیم می‌شود و هر بخش مطالب مرتبطی را مورد بررسی قرار می‌دهد.

بخش نخست در مورد آشنایی با ASP.NET Core MVC، ویژگی‌های کاربردی الگوی سه بخشی MVC را مورد بررسی قرار می‌دهد و همراه با ارائه‌ی روش‌های امروزی توسعه‌ی وب، ابزارهایی را که C# برای رسیدن به این هدف در اختیار قرار می‌دهد بررسی می‌کند.

در فصل دوم، همراه با ایجاد پروژه‌ای ساده، با مفاهیم پایه‌ی MVC آشنا خواهید شد. ادامه‌ی بخش نخست کتاب به پیاده‌سازی پروژه‌ای واقعی به نام SportsStore اختصاص پیدا کرده است که سعی بر پیاده‌سازی ویژگی‌های مهم ASP.NET Core MVC خواهد داشت.

در بخش دوم کتاب، که بخش پیشرفته‌ی آن را تشکیل می‌دهد، به بررسی زیرساخت‌ها و کارکردهای درونی ویژگی‌هایی که از آنها در پروژه‌ی SportsStore استفاده شد، خواهیم پرداخت. همراه با شرح چگونگی کارکرد هر یک از این ویژگی‌ها، پیکربندی‌های مختلف نیز بررسی می‌شوند.

توجه: به دلیل زمان اندک باقیمانده تا نمایشگاه بین‌المللی کتاب تهران، و آماده نبودن بخش دوم کتاب، تصمیم بر آن شد بخش نخست کتاب در قالب **جلد ۱** ارائه شود و به محض تکمیل ترجمه بخش دوم، **جلد ۲** آن ارائه شود. سپس یک کتاب تک جلدی که هردو بخش را پوشش دهد به چاپ خواهیم رساند.

فصل دوم

ایجاد نخستین پروژه MVC

بهترین راه برای درک یک فریم ورک توسعه نرم‌افزار، تجربه استفاده از آنست. در این فصل، یک پروژه ساده‌ی ورود داده با استفاده از MVC ایجاد خواهیم کرد. برای این که درک خوبی از چگونگی ایجاد و عملکرد چنین پروژه‌ای داشته باشید، کار را به صورت گام به گام پیش خواهیم برد. همچنین برای ساده نگاه داشتن کار، در این مرحله، از تشریح برخی جزئیات، پرهیز خواهیم کرد.

۲-۱ نصب ویژوال استدیو

در این کتاب از نگارش ۲۰۱۵ ویژوال استدیو استفاده می‌کنیم که همه امکانات لازم برای استفاده از ASP.NET MVC را در اختیارمان قرار می‌دهد. نسخه مجانی نرم‌افزار را می‌توانید از آدرس www.visualstudio.com دانلود کنید. در هنگام نصب ویژوال استدیو حتما گزینه‌ی Web developer Tools را انتخاب کرده باشید.

اگر نسخه‌ی نصب شده‌ای از ویژوال استدیو را در اختیار دارید، به احتمال قوی باید به‌روزرسانی ۳ (Update 3) را انجام دهید. در صورتی که چنین است، برای اعمال این به‌روزرسانی به آدرس زیر مراجعه کنید:

http://go.microsoft.com/fwlink/?LinkId=691129

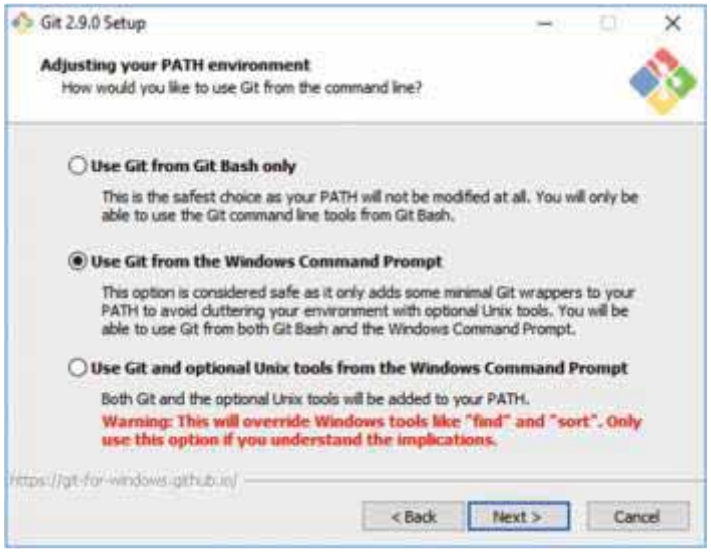
در مرحله‌ی بعد، برای نصب .NET Core باید به آدرس زیر مراجعه کنید:

https://go.microsoft.com/fwlink/?LinkId=817245

این موضوع حتی برای نصب‌های جدید ویژوال استدیو هم الزامی است. آخرین گام، نصب ابزاری به نام git است که از آدرس زیر قابل دستیابی است:

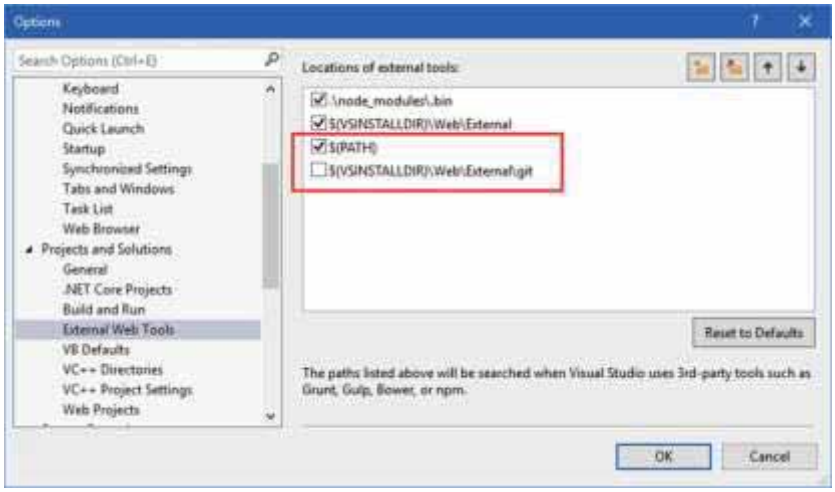
https://git-scm.com/download

ویژوال استدیو نگارشی از git را نصب می‌کند که متاسفانه در هنگام استفاده با ابزار دیگر، از جمله Bower که در فصل ۶ به آن خواهیم پرداخت، نتایج نادرستی ایجاد می‌کند. هنگام نصب git، همانطور که در شکل ۱-۲ نشان داده شده است، دقت کنید که ابزار مورد گفتگو حتما به متغیر محیطی PATH اضافه شود. در این صورت ویژوال استدیو به راحتی قادر به یافتن آن خواهد بود.



شکل ۱-۲

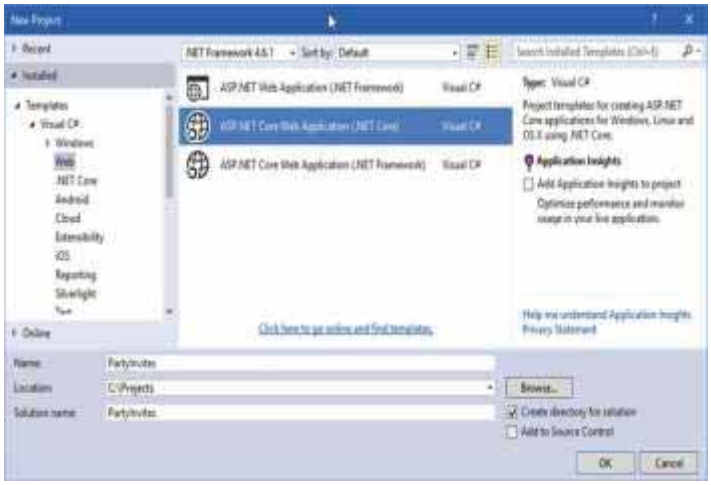
ویژوال استدیو را اجرا کنید و پس از انتخاب گزینه‌ی Options از فهرست Tools، همانطور که در شکل ۲-۲ نشان داده شده است، با انتخاب External Web tools از بخش Projects and Solutions، گزینه‌ی $(VSINSTALLDIR)\Web\External\git item را از حالت انتخاب خارج کنید. به همین صورت مطمئن شوید که گزینه $(PATH) فعال است. با انجام کارهای بالا می‌توانید مطمئن باشید که نسخه git نصب شده توسط شما، به جای نسخه پیش‌فرض ویژوال استدیو، اجرا خواهد شد.



شکل ۲-۲

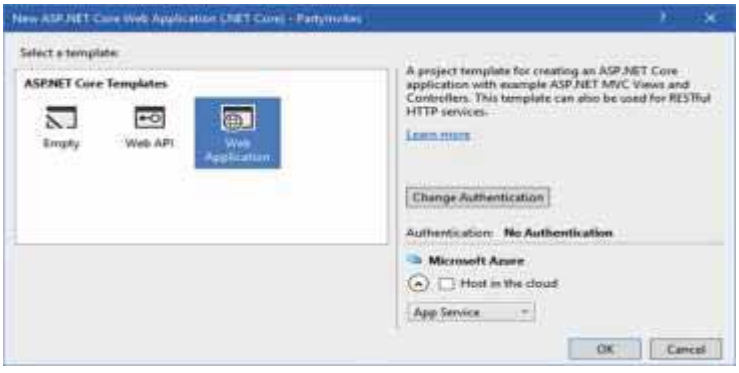
۲-۲ ایجاد پروژه جدید MVC

کار را با ایجاد پروژه‌ی جدیدی از نوع ASP.NET Core MVC در محیط ویژوال استدیو شروع می‌کنیم (شکل ۲-۳). اکنون از فهرست File گزینه‌ی New Project را انتخاب کنید. در این وضعیت، کادر گفتگوی پروژه‌ی جدید، گشوده می‌شود. اکنون در صورتی که از پنل سمت چپ، ابتدا Templates و پس از آن، Visusl C# را انتخاب کنید، در بخش Web، نوع پروژه مورد نظرمان، یعنی ASP.NET Core Web Application را خواهید دید.



شکل ۲-۳

نام پروژه را PartyInvites گذاشته و همان‌گونه که در شکل ۲-۳ نشان داده شده است، توجه کنید که گزینه‌ی Add Application Insights to Project انتخاب نشده باشد. پس از کلیک بر دکمه OK، همان‌گونه که در شکل ۲-۴ نشان داده شده است، کادر گفتگوی نشان داده شده، در مورد محتویات پروژه پرسش خواهد کرد.



شکل ۲-۴

هر یک از سه نوع الگوی پروژه‌ی نشان داده شده در شکل، پروژه‌ای با محتوای آغازین متفاوتی را ایجاد می‌کند. برای کار این فصل، الگوی Web Application را انتخاب کنید.

همان‌گونه که در شکل ۵-۲ نشان داده شده است، با کلیک بر روی دکمه‌ی Change Authentication و اطمینان از اینکه گزینه‌ی No Authenticaion انتخاب شده است، پروژه را ایجاد کنید. در این پروژه نیازی به هیچ نوع خاصی از اعتبارسنجی نخواهیم داشت. مطالب مربوط به امنیت را در فصل‌های ۲۸ تا ۳۰ بررسی خواهیم کرد.

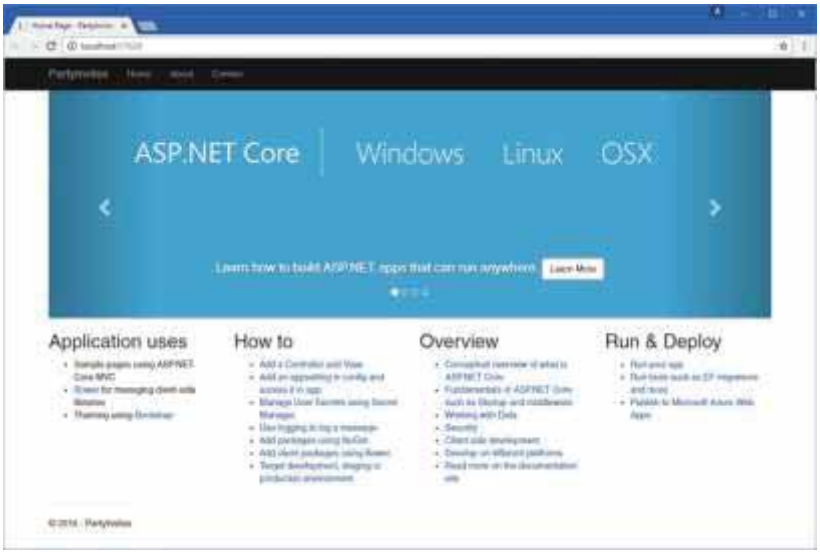
شکل ۵-۲

پس از بستن کادر بالا، در بازگشت به کادر پیشین، مطمئن شوید که گزینه Host in Cloud انتخاب نشده باشد. با کلیک بر روی OK پروژه‌ی PartyInvites را ایجاد کنید. پس از ایجاد پروژه و باز شدن محیط آن در ویژوال استدیو، فایل‌ها و پوشه‌های مختلفی را در Solution Explorer خواهید دید (شکل ۶-۲).

شکل ۶-۲

برای اجرای پروژه (با محتوای پیش‌فرضی که ویژوال استدیو در آن ایجاد کرده ست)، از فهرست debug گزینه‌ی Start debugging را انتخاب کنید (اگر پیامی در مورد فعال کردن دیباگ برنامه نمایان شد، بر روی OK کلیک کنید). با انجام این کار، ویژوال استدیو پس از کامپایل پروژه، برنامه‌ای به نام IIS Express را برای اجرای آن، به کار می‌برد. پس از این کار، با باز کردن یکی از مرورگرهای اینترنت

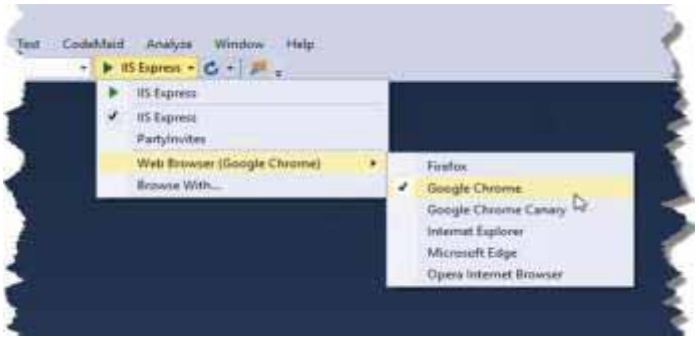
نصب شده بر روی کامپیوتر، تقاضایی برای نمایش محتوای برنامه به IIS Express ارسال می‌کند (شکل ۷-۲).



شکل ۷-۲

در ادامه این فصل، برای ایجاد برنامه‌ی مورد نظرمان، محتوای پیش‌فرض ویژوال استدیو برای پروژه را، به آرامی با کدی مناسب، جایگزین خواهیم کرد. هم اینک برای پایان دادن به اجرای پروژه، یا پنجره‌ی مرورگر را ببندید و یا در محیط ویژوال، از فهرست Debug گزینه‌ی Stop debugging را انتخاب کنید.

همان‌گونه که در شکل ۸-۲ می‌بینید، با استفاده از دکمه‌ی IIS Express در نوار ابزار، می‌توانید هر یک از مرورگرهای نصب شده در کامپیوتر را برای اجرای پروژه به ویژوال استدیو معرفی کنید.



شکل ۸-۲

۱-۲-۲ افزودن کنترلر به پروژه

در مدل برنامه‌نویسی MVC تقاضاهای رسیده (به سایت یا وب اپلیکیشن) توسط کنترلرها بررسی و پاسخ داده می‌شوند. در مدل پیاده شده توسط مایکروسافت، ASP.NET Core MVC، کنترلرها کلاس‌های معمولی #C هستند که از Microsoft.AspNetCore.Mvc.Controller که کلاس پایه کنترلر محسوب می‌شود، ارث‌بری کرده‌اند.

هر متدی با دسترسی عمومی (Public) در کنترلر، به عنوان یک متد عملیاتی (Action Method) شناخته می‌شود. این به معنی آن است که چنین متدی را می‌توانید از داخل یک صفحه وب، توسط یک URL فراخوانی کنید تا عمل (Action) ویژه‌ای را انجام دهد. بر اساس آنچه که در MVC مرسوم است، ویژوال استدیو کنترلرها را در پوشه‌ای به نام Controllers قرار می‌دهد. همان پوشه‌ای که در هنگام ایجاد پروژه به طور خودکار توسط ویژوال استدیو ایجاد شد.

با باز کردن پوشه Controllers در مرورگر سالوشن متوجه می‌شوید که ویژوال استدیو کنترلری پیش‌فرض به پروژه اضافه کرده است. نام فایل کنترلر HomeController است. فایل کلاس کنترلر همیشه داری یک پیشوند (در اینجا، Home) و سپس واژه‌ی Controller است. با مشاهده‌ی چنین نامی متوجه می‌شوید که فایل مورد نظر باید دارای کنترلری به نام Home باشد (کنترلر پیش‌فرض در پروژه‌های MVC ویژوال استدیو). برای باز کردن این فایل، یک بار بر روی آن کلیک کنید. ویژوال استدیو این فایل را یا کدی که در لیست ۱-۲ نمایش داده شده است، برای ویرایش باز می‌کند.

لیست ۲-۱: کد فایل کنترلر پیش‌فرض ویژوال استدیو

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
        public class HomeController : Controller {
                public IActionResult Index() {
                return View();
        }
        public IActionResult About() {
                ViewData["Message"] = "Your application description page.";
                return View();
        }
        public IActionResult Contact() {
                ViewData["Message"] = "Your contact page.";
                return View();
        }
        public IActionResult Error() {
                return View();
        }
}}
```

کد بالا را با کدی که در لیست ۲-۲ آمده است، جایگزین کنید. در اینجا، عبارت‌های using بدون استفاده و همه متدها به غیر از یکی از آنها حذف شـده‌اند. همان‌گونه که می‌بینید، پیاده‌سـازی متد Index و نوع برگشتی آن هم تغییر کرده‌اند.

لیست ۲-۲: کد تغییر یافته کنترلر Home

```
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
        public class HomeController : Controller {
                public string Index() {
                return "Hello World";
                }
        }
}
```

متد ()Index رشته‌ای به شکل Hello World از نوع string بازمی‌گرداند. دوباره پروژه را، به همان روش پیشین اجرا کنید.

مرورگر یک تقاضـای[1] HTTP به سرور ارسال می‌کند. بر اساس پیکربندی پیش‌فرض MVC این تقاضا با متد ()Index پاسـخ داده می‌شود. این متد را action method یا به اختصار، action می‌گویند و خروجی این متد در پاسخ به تقاضای یاد شده، برای مرورگر فرستاده می‌شود (شکل ۹-۲).

شکل ۹-۲

در شـکل بالا، ویژوال استدیو مرورگر را به پورت ۵۷۶۲۸ هدایت کرده است. از آنجا که ویژوال استدیو همیشـه این پورت را به شـکل تصـادفی انتخاب می‌کند، شـماره آن در کامپیوتر شـما می‌تواند متفاوت باشد. افزون بر این با دقت در نوار وظیفه، بخش اعلانات، متوجه آیکن کوچکی در رابطه با IIS Express می‌شوید. این برنامه، به عنوان سرویس دهنده‌ی وب، نگارش کوسبک شده‌ای از برنامه اصلی IIS نصب شده در ویندوز است.

[1] HTTP Request

۲-۲-۲ بررسی و فهم مسیرها

افزون بر مـدل‌هـا[1]، نمـاها[2] و کنترلرها[3]، برنامه‌های کاربردی MVC از روش آدرس‌دهی ASP.NET برای مربوط کردن URLها به کنترلرها و متدهای آنها (اکشن‌ها) استفاده می‌کنند. مسیر[4] قانونی است که معین می‌کند یک درخواســت چگونه پاســخ داده شــود. ویژوال اســتدیو در هنگام ایجاد پروژه، چند مســیر پیش‌فرض برای شروع کار، تولید می‌کند. هر یک از URLهای زیر که اکشنی در یک کنترلر اجرا می‌کنند، را می‌توانید به کار برید.

- /
- /Home
- /Home/Index

بنابراین هر بار مرورگری یکی از آدرس‌های http://yoursite/ یا http://yoursite/Home را تقاضا کند، خروجی متد اکشــن Index را دریافت خواهد کرد. این موضــوع را می‌توانید با تغییر آدرس موجود در مرورگر آزمایش کنید. هم اینک این آدرس به صــورت http://localhost:57628 ولی شــاید با پورتی متفاوت اســت. اگر رشــته‌های Home/ و یا Home/Index/ را به دنباله‌ی آن اضــافه کنید، همان نتیجه‌ی گذشته، Hello World را دریافت خواهید کرد.

وضــعیت بالا نشــان‌دهنده‌ی مثال خوبی از پذیرفتن پیکربندی پیش‌فرض ASP.NET Core MVC اســت. تنظیم پیش‌فرض در اینجا این است که، پروژه دارای کنترلی به نام Home (با نام فایل HomeController) و این کنترلر نقطه شــروع برنامه اســت (با اکشــن Index). اگر این پیکربندی پیش‌فرض را دنبال نکرده بودیم، می‌بایســت تنظیمات آدرس‌دهی را بر اســاس کنترلی که ایجاد کرده بودیم (متفاوت از Home) تغییر می‌دادیم.

۲-۳ پردازش و نمایش صفحات وب

خروجی مثال گذشــته به جای این که HTML باشــد، رشــته Hello World بود. برای این که در پاســخ درخواســت مرورگر پاســخ HTML تولید کنیم، باید نمایی داشــته باشــیم که برنامه را برای تولید چنین خروجی به درستی هدایت کند.

---

[1] Models
[2] Views
[3] Controllers
[4] Route

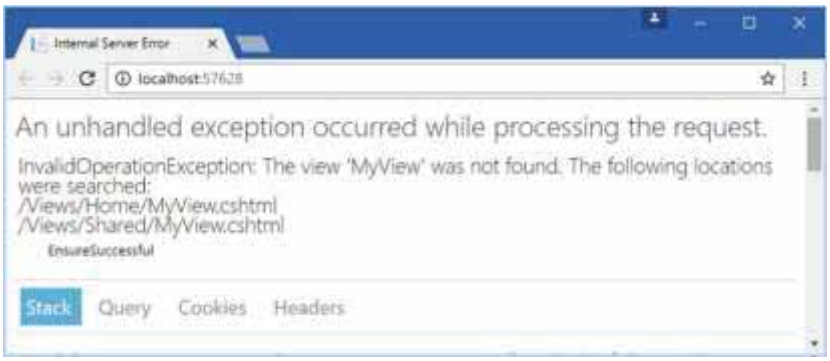
۱-۳-۲ ایجاد نما

نخستین گام، ویرایش اکشن Index به فرم نشان داده شده در لیست ۳-۲ است.

لیست ۳-۲: ویرایش کنترلر برای نمایش صفحه وب

```
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
        public class HomeController : Controller {
                public ViewResult Index() {
                return View("MyView");
                }}
}
```

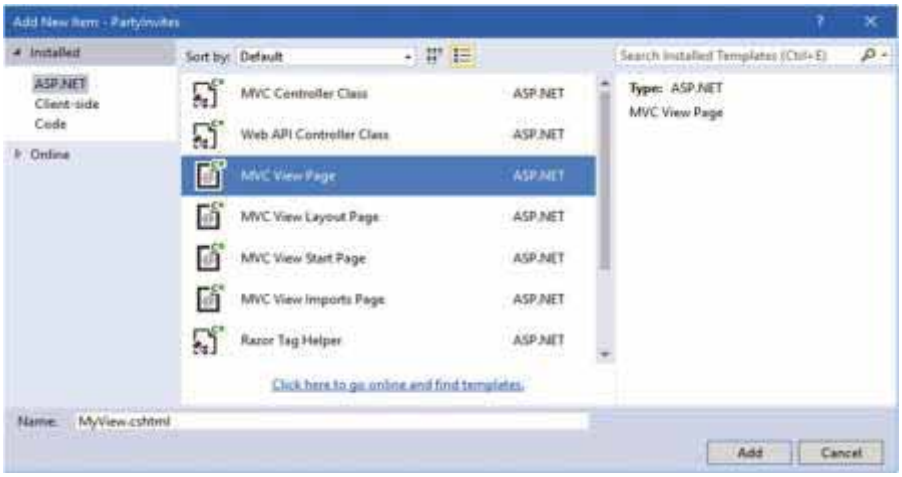
متد اکشنی که خروجی آن شیئی از نوع ViewResult باشد، موجب پردازش یک نما می‌شود. در اینجا شیء ViewResult با فرخوانی متد ()View و ارائه نام نما به صورت پارامتر متد (MyView) تولید شده است. با اجرای برنامه، خواهید دید که MVC سعی بر یافتن این نما (که البته وجود ندارد) کرده و در نتیجه با پیام خطایی همانند شکل ۱۰-۲ برخورد خواهید کرد.



شکل ۱۰-۲

پیام خطای بالا نه تنها نشان دهنده‌ی این است که برنامه به دنبال یافت نما بوده است، بلکه مشخص کننده‌ی محل‌های جست‌وجو نیز هست. نماها در پوشه‌ای به نام Views که دارای زیرشاخه‌هایی (پوشه فرعی سطوح پایین‌تر) است، ذخیره و دسته‌بندی می‌شوند. نام این پوشه‌ها همیشه با نام کنترلر یکی است. نماهایی که به کنترلر Home مربوط می‌شوند، در پوشه‌ای به همین نام ذخیره می‌شوند (Views/Home). توجه داشته باشید که هر متد اکشن داخل کنترلر، دارای یک نما در پوشه‌ی یاد شده خواهد بود. نماهایی که به کنترلر خاصی مربوط نیستند، در پوشه‌ای به آدرس Views/Shared ذخیره می‌شوند.

برای ایجاد نما، بر روی پوشه‌ی Views/Home کلیک راست و سپس از فهرست Add گزینه‌ی New Item را انتخاب کنید. مانند شکل ۱۱-۲، ویژوال استدیو فهرستی از قالب‌های قابل استفاده را نشان می‌دهد. در اینجا از کادر سمت چپ، ASP.NET و پس از آن، در کادر مرکزی، MVC View را انتخاب کنید.



شکل ۱۱-۲

نام نما را در فیلد Name، MyView.cshtml وارد و بر روی دکمه‌ی OK کلیک کنید. ویژوال اســـتدیو فایل نمای Views/Home/MyView.cshtml را ایجاد و آن را برای ویرایش باز می‌کند. محتوای کنونی آن را با آنچه که در لیست ۴-۲ آمده است، جایگزین کنید.

لیست ۴-۲: کدی که باید در نمای Views\MyView.cshtml وارد کنید

```
@{
Layout = null;
}
<!DOCTYPE html>
<html>
<head>
        <meta name="viewport" content="width=device-width" />
        <title>Index</title>
</head>
<body>
        <div>
        Hello World (from the view)
        </div>
</body>
</html>
        The new contents of the view file are mostly HTML. The exception is the part that
        looks like this:
...
@{
        Layout = null;
}
...
```

کد بالا توسط موتور تولید نمای[1] Razor تفسیر شده و موجب تولید HTML و فرستادن آن به مرورگر می‌شود. عبارت ساده‌ی بالا مشخص می‌کند که برای تولید صفحه‌ی وب، به دنبال استفاده از Layout نیستیم. Layout مانند الگویی برای تولید صفحه وب به‌کار می‌رود و در فصل ۵ مورد بررسی قرار خواهد گرفت. بررسی بیشتر کد Razor را به آینده موکول می‌کنیم. برای دیدن تأثیر ایجاد نما، برنامه را با انتخاب Start Debugging از فهرست Debug اجرا کنید. نتیجه را مانند شکل ۱۲-۲ خواهید دید.

شکل ۱۲-۲

اکشن ()Index در نخستین ویرایش خود، تنها یک رشته را بازمی‌گرداند. معنی آن این بود که MVC بدون آن که کاری انجام دهد، این رشته را برای مرورگر می‌فرستاد. اکنون که این اکشن شیئی از نوع ViewResult نتیجه می‌دهد، MVC نمایی را پردازش کرده و HTML تولید شده را برای مرورگر می‌فرستد. آز آنجا که نام نما را هم برای MVC مشخص کردیم (MtView)، به صورت خودکار از مسیرهای پیش‌فرض قراردادی برای یافتن آن استفاده کرد. قرارداد مرسوم این است که نما هم‌نام متد اکشن بوده و در پوشه‌ای با نام کنترلر دربرگیرنده‌ی اکشن ذخیره شده باشد:

/Views/Home/MyView.cshtml

متدهای اکشن، افزون بر آنچه که در بالا دیدید، می‌توانند انواع دیگری را هم بازگردانند. به صورت مثال، بازگرداندن شیئی از نوع RedirectResult موجب هدایت مرورگر به آدرس جدیدی می‌شود. اگر شیئی از نوع HttpUnauthorizedResult بازگردانده شود، کاربر را مجبور به لاگین می‌کند. مجموعه‌ی این اشیاء را به نام نتایج اکشن[2] می‌شناسند. در فصل هفدهم آشنایی بیشتری با این مجموعه پیدا خواهیم کرد.

۲-۳-۲ خروجی پویا[3]

هدف نهایی از یک برنامه کاربردی وب، ایجاد و نمایش محتوای پویاست. در MVC پردازش و ایجاد داده و ارسال آن برای نما، وظیفه کنترلر است. پس از آن، وظیفه‌ی نما، تولید HTML است. یک روش برای ایجاد داده‌ی مورد نیاز نما، استفاده از شیء ViewBag (عضوی از کلاس پایه‌ی Controller) است.

[1] Razir View Engine
[2] Action results
[3] Dynamic Output

ViewBag شـیئی پویاسـت که می‌توانید خصـوصـیات دلخواهی را به آن نسـبت دهید. پس از آن، این خصـوصـیات می‌توانند در دسـترس نمای مربوط به متد اکشـن قرار گیرند. لیسـت ۵-۲ نشـان‌دهنده‌ی داده‌های ساده‌ی پویاییست که به این صورت در HomeController ایجاد شده‌اند.

لیست ۵-۲: تهیه‌ی داده‌ی مورد نیاز نما در کنترلر

```
using Microsoft.AspNetCore.Mvc;
namespace PartyInvites.Controllers {
        public class HomeController : Controller {
                public ViewResult Index() {
                        int hour = DateTime.Now.Hour;
                        ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good
                        Afternoon";
                        return View("MyView");
                }
        }
}
```

با نسـبت دادن مقداری به خصـوصـیت ViewBag.Greeting داده‌ی مورد نیاز نما تولید می‌شـود. توجه کنید که این خصـوصـیت درسـت زمانی ایجاد می‌شـود که مقداری به آن نسـبت می‌دهید. این موضوع امکان فراهم کردن داده برای نما به صـورتی روان و سـاده و بدون آن که لازم باشـد از پیش کلاسـی ایجاد کرده باشـید، را فراهم می‌کند. همان‌گونه که در شـکل ۶-۲ می‌بینید، برای دریافت داده‌ی یاد شده، دوباره در کد نما از خصـوصـیت ViewBag.Greeting اسـتفاده شـده اسـت. این شـکل نشـان دهنده‌ی تغییرات نمای MyView.cshtml هم هست.

لیست ۶-۲: دستیابی به محتوای خصوصیت ذخیره شده در ViewBag داخل کد نما

```
@{
        Layout = null;
}
<!DOCTYPE html>
<html>
<head>
        <meta name="viewport" content="width=device-width" />
        <title>Index</title>
</head>
<body>
        <div>
                @ViewBag.Greeting World (from the view)
        </div>
</body>
</html>
```

هنگامی که متد ()View در اکشـن ()Index فرخوانی می‌شـود، MVC فایل MyView.cshtml را یافته و از موتور Razor تقاضـای تفسـیر کد آن را می‌کند. Razor به دنبال عباراتی شـبیه آنچه که در کد بالا (با پیش‌نشـانه‌ی @) آمده اسـت گشـته و آنها را اجرا می‌کند. در این مثال، نتیجه‌ی اجرای عبارت، درج مقداری است که در اکشن به خصوصیت ViewBag.Greeting نسبت داده شده است.