آموزش کاربردی

Pro
ASP.NET Core MVC

ویراست ۶

جلد ۲

آدام فریمن

ترجمه: مهندس نادر نبوی

انتشارات پندار پارس

سرشناسه	:	فریمن، آدام، ۱۹۷۲ - م. Freeman, Adam
عنوان و نام پدیدآور	:	مرجع کامل ASP.NET Core MVC/ آدام فریمن ؛ ترجمه نادر نبوی.
مشخصات نشر	:	تهران : پندار پارس ، ۱۳۹۶.
مشخصات ظاهری	:	۲ ج.: مصور، جدول.
شابک	:	ج.۱ : 978-600-8201-37-3: ؛ ج.۲: 978-600-8201-52-6
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 (جلد۲)**

ناشر : **انتشارات پندار پارس**

تالیف : **Adam Freeman**

ترجمه : **نادر نبوی**

چاپ نخست : **آبان ماه ۹۶**

شمارگان : **۵۰۰ نسخه**

طرح جلد : **رامین شکراللهی**

چاپ، صحافی : **روز**

قیمت : **۳۲۰۰۰ تومان** **شابک : ۶-۵۲-۸۲۰۱-۶۰۰-۹۷۸**

سخن مترجم

کتابی که در دست دارید، ویرایش ششم کتاب Pro ASP.NET Core MVC است. در جلد نخست کتاب، در فصل‌های ۱ تا ۳ کتاب در مورد تکامل برنامه‌نویسی سمت سرور مایکروسافت، از پروژه‌های وب ASP.NET تا چرخش مثبتی که به سمت پروژه‌های MVC ایجاد شد و در پایان منجر به پروژه‌های Core MVC شد، توضیح جامعی داده شد. همچنین، با تشریح مفاهیم پایه‌ی MVC، به پیاده‌سازی یک پروژه‌ی کامل پرداختیم. خوانندگانی که از پیش با برنامه‌نویسی فرم‌های وب آشنایی داشته‌اند، با مشکلات این پروژه‌ها و دردسرهای آنها در پیاده‌سازی پروژه‌های بزرگ و سازمانی، دست و پنجه نرم کرده‌اند.

در جلد دوم (این کتاب)، به تشریح جزئیات مباحث گفته شده در جلد یکم و تحکیم مبانی تئوریک آن پرداخته شده است. هر فصل، با پیاده‌سازی یک پروژه، به تشریح مباحث آن فصل می‌پردازد که این، موجب کاربردی شدن کتاب برای خواننده‌ای که قصد فراگیری کامل MVC را دارد، خواهد شود.

خوشبختانه برای یادگیری معماری جدید Core MVC با خواندن این کتاب، نیاز به آشنایی با واسطهای برنامه‌نویسی قدیمی وب، که به آنها اشاره شد ندارید. به عنوان تنها پیش‌نیاز لازم، آشنایی با مفاهیم وب به همراه توانایی کار با HTML و CSS، زبان C# به همراه Entity Framework و نوشتار کوئری‌های LINQ، کافی است. معنی این گفته این است که می‌توانید برنامه‌نویسی سمت سرور وب را از ابتدا با همین کتاب شروع کنید.

توجه: کد کامل پروژه‌ی اصلی کتاب و سایر کدهایی که به شکل مثال در فصل‌های مختلف آورده شده‌اند را می‌توانید از آدرس زیر به آسانی به دست آورید:

http://www.github.com/apress/pro-asp.net-core-mvc

نسخه تک جلدی کتاب نیز که ادغام دو جلد در یک جلد است و مبحث امنیت در Core MVC نیز به آن افزوده خواهد شد، در اختیار علاقه مندان قرار خواهد گرفت. برای دوستانی که دارای جلد ۱ و جلد ۲ کتاب هستند نیز راهکاری خواهیم اندیشید تا بتوانند از مباحثی که به کتاب تک جلدی خواهیم افزود، استفاده کنند.

در پایان، از خوانندگان عزیز تقاضا دارم که پرسش‌ها و مشکلات خود را در سایت انتشارات به آدرس www.pendarepars.com و یا مستقیما به آدرس پست الکترونیکی خودم، nanavijobmail@yahoo.com در میان بگذارند.

نادر نبوی

تابستان سال ۱۳۹۶

فهرست

فصل سیزدهم

پیکربندی پروژه‌ها

گرچه ممکن است عنوان پیکربندی و مطالب مربوط به آن در نگاه نخست، خیلی جذاب به نظر نرسد، ولی مطمئن باشید که آشنایی با جزئیات آن، نکات فراوانی در رابطه با چگونگی کارکرد MVC و نیز مدیریت رفتار درخواست‌های HTTP برای شما آشکار خواهد کرد. بنابراین بهتر است به جای تسلیم در برابر رد شدن از این فصل، مدتی وقت خود را صرف آن کنید، تا به این ترتیب با چگونگی شکل‌گیری یک پروژه‌ی MVC به وسیله‌ی سیستم پیکربندی آشنا شوید. انجام این کار پایه‌ای قوی برای دنبال کردن فصل‌های آینده برایتان ایجاد خواهد کرد.

اگر از پیش با نگارش‌های دیگری از ASP.NET کار کرده باشید، روش پیکربندی پروژه را یکی از بارزترین تغییرات ASP.NET Core، خواهید یافت. مجموعه‌ای از فایل‌ها مانند Global.asax، FilterConfig.cs و RouteConfig.cs ناپدید شده‌اند و به جای آنها با مجموعه‌ی جدیدی از فایل‌های JSON و کلاس‌هایی مانند Startup و Program برخورد خواهید کرد. در این فصل روش پیکربندی پروژه‌ی MVC به وسیله‌ی این فایل‌ها را شرح خواهیم داد و افزون بر این، خواهید دید که چگونه MVC بر پایه‌ی ویژگی‌های برگرفته از ASP.NET Core ساخته می‌شود. جدول ۱-۱۳ تعدادی از سوالات مربوط به پیکربندی را لیست کرده است.

پرسش	پاسخ
پیکربندی چیست؟	تعیین چگونگی کارکرد پروژه و بسته‌های نرم‌افزاری که پروژه به آنها وابسته است، به وسیله‌ی کلاس‌های Startup و Program به همراه فایل‌های JSON.
چرا مفید است؟	پیکربندی، افزون بر مدیریت بسته‌های پروژه، محیطی برای کارکرد درست همه‌ی اجزای نرم‌افزار فراهم می‌کند.
چگونه به کار می‌رود؟	مهم‌ترین بخش پیکربندی، کلاس Startup است که برای ایجاد سرویس‌ها (که کارآیی‌های لازم را در اختیار سایر اجزای برنامه قرار می‌دهند) و عناصری به کار می‌رود که چگونگی تعامل با درخواست‌های HTTP را مدیریت می‌کنند.
آیا محدودیت‌هایی در این زمینه	در پروژه‌های خیلی بزرگ، مدیریت پیکربندی می‌تواند

پرسش	پاسخ
وجود دارد؟	قدری مشکل باشد.
آیا روش دیگری هم برای دست‌یابی به همین نتایج وجود دارد؟	خیر. پیکربندی پروژه به وسیله‌ی کلاس‌هایی که گفته شد و فایل‌های JSON تنها روش انجام درست تنظیمات در پروژه‌های ASP.NET Core MVC است.
آیا روش‌های پیکربندی از نگارش ۵ MVC به بعد، تغییری کرده است؟	به طور کامل دگرگون شده است و اکنون به سمتی می‌رود که از وابستگی اجرای برنامه‌ها در محیط متداول IIS رهایی یابد.

جدول ۲-۱۳، خلاصه‌ای از مطالب این فصل را فهرست کرده است.

مشکل	راه حل	شماره‌ی لیست کد
افزودن کارآیی جدید به پروژه	افزودن بسته‌های جدید به بخش‌های dependencies و tools در فایل project.json.	۱-۶
مدیریت شروع به کار پروژه	استفاده از فایل Program.cs.	۷
پیکربندی پروژه	استفاده از متدهای ()Configure و ()ConfigureServices در کلاس Startup.	۸ و ۹
فراهم کردن یک کارآیی مشترک برای بخش‌های مختلف پروژه	از متد ()ConfigureServices برای ایجاد سرویس مورد نظر استفاده کنید.	۱۰-۱۲
تولید محتوا در پاسخ درخواست	از میان-افزارهای[1] تولید محتوا استفاده کنید.	۱۳-۱۵
جلوگیری از پیمایش همه‌ی زنجیره‌ی درخواست	از میان-افزارهای میان‌بر[2] استفده کنید.	۱۶-۱۷

[1] میان-افزار یا Middleware اصطلاح جاافتاده‌ای در MVC است که به عناصر تشکیل دهنده‌ی زنجیره‌ی درخواست اشاره دارد. درخواست جدید در ابتدا به نخستین میان‌افزار تشکیل‌دهنده‌ی زنجیره‌ی درخواست تحویل داده می‌شود و الی آخر. دسته‌ی مهمی از این عناصر وظیفه‌ی ایجاد محتوا را به عهده دارند.

[2] short-circuiting middleware

مشکل	راه حل	شماره‌ی لیست کد
توسط درخواست رسیده		
ویرایش درخواست پیش از پردازش سایر میان‌افزارها	از میان‌افزارهای ویرایشی استفاده کنید.	۱۸–۲۰
ویرایش پاسخی که توسط عناصر میان‌افزار پردازش شده است	میان–افزار ویرایش‌کننده‌ی[1] پاسخ ایجاد کنید.	۲۱ و ۲۲
تنظیم کارآیی‌های MVC	از متدهای ()UseMvc یا ()UseMvcWithDefaultRoute استفاده کنید.	۲۳
تغییر پیکربندی پروژه برای محیط‌های مختلف	از سرویس میزبانی محیط[2] استفاده کنید.	۲۴
ایجاد لاگ برای داده‌های پروژه	از میان‌افزار ایجاد لاگ[3] استفاده کنید.	۲۵–۲۷
مدیریت خطاهای برنامه	از میان‌افزار مدیریت خطاهای زمان توسعه[4] استفاده کنید.	۲۸ و ۲۹
کاربرد چندین مرورگر در زمان توسعه	از ویژگی Browser Link استفاده کنید.	۳۰
فعال کردن جاوااسکریپت، CSS و تصاویر	میان‌افزار محتوای استاتیک را فعال کنید.	۳۱
جداسازی اطلاعات پیکربندی از کد C#	فایل‌های پیکربندی مانند فایل‌های Json ایجاد کنید.	۳۲–۳۷
پیکربندی سرویس‌های MVC	از متد ()AddMvcOptions و انتخاب‌های مربوط به آن، در ایجاد سرویس استفاده کنید.	۳۸

[1] Response Editing Middleware
[2] Hosting Environment Service
[3] Logging Middleware
[4] Developer Error-Handling Middleware

مشکل	راه حل	شماره‌ی لیست کد
پیکربندی پروژه‌های بسیار پیچیده	از چندین کلاس یا فایل‌های بیرونی استفاده کنید.	۴۳-۳۹

۱-۱۳ آماده‌سازی پروژه

برای این فصل، پروژه‌ای به نام ConfiguringApps با استفاده از الگوی پروژه‌ی خالی (EmptyTemplate)، ایجاد کنید. پیکربندی این پروژه را در بخش‌های آینده انجام خواهیم داد ولی اکنون برخی تظیمات پایه باید پیاده شوند.

از آنجا که برای شکل‌دهی محتوا از بوت استرپ استفاده خواهیم کرد، با استفاده از الگوی Bower Configuration File، پس از افزودن فایل bower.json، با توجه به لیست ۱-۱۳ بسته‌ی مورد نیاز را در آن تعریف کنید.

لیست ۱-۱۳: معرفی بسته‌ی بوت استرپ در فایل bower.json

```
{
  "name": "asp.net",
  "private": true,
  "dependencies": {
    "bootstrap": "3.3.6"
  }
}
```

در گام بعدی، پس از ساختن پوشه‌ی Controllers، کلاس HomeController.cs را با توجه به کد کنترلر Home در لیست ۲-۱۳، در آن ایجاد کنید.

لیست ۲-۱۳: کد کلاس HomeController.cs در پوشه‌ی Controllers

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;

namespace ConfiguringApps.Controllers {
  public class HomeController : Controller {
    public ViewResult Index()
    => View(new Dictionary<string, string> {
      ["Message"] = "This is the Index action"
    });
  }
}
```

همان‌طور که می‌دانید، اکنون باید ابتدا پوشه‌ی Views/Home را بسازید و پس از آن، فایل نمای Index.cshtml را به آن اضافه کنید (لیست ۳-۱۳).

لیست ۳-۱۳: کد نمای Index.cshtml

```
@model Dictionary<string, string>
@{ Layout = null; }

<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width" />
  <link asp-href-include="lib/bootstrap/dist/css/*.min.css" rel="stylesheet"
  />
  <title>Result</title>
</head>
<body class="panel-body">
  <table class="table table-condensed table-bordered table-striped">
    @foreach (var kvp in Model) {
      <tr><th>@kvp.Key</th><td>@kvp.Value</td></tr>
    }
  </table>
</body>
</html>
```

عنصر link در کد بالا، با به کارگیری یک تگ کمکی (Tag Helper)، سعی بر انتخاب فایل Bootstrap.css دارد. بنابراین برای فعال‌سازی Tag Helperهای پیش‌ساخته، با استفاده از الگوی MVC View Imports Page، فایل _ViewImports.cshtml را با توجه به کد لیست ۴-۱۳، به پروژه اضافه کنید.

لیست ۴-۱۳: کد فایل _ViewImports.cshtml در پوشه‌ی Views

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

از آنجا که نما و کنترلر ایجاد شده نیاز به ویژگی‌هایی دارند که هنوز در پروژه موجود نیستند، در حال حاضر امکان اجرای برنامه را ندارید. این مشکل را در بخش‌های آینده، همراه با پیکربندی کامل پروژه حل خواهیم کرد.

۲-۱۳ فایل‌های پیکربندی JSON

قالب Javascript Object Notation (JSON) در پروژه‌های MVC Core دارای دو نقش متفاوت است. در نقش نخست، به عنوان قالب اصلی تبادل داده بین برنامه‌ی کاربردی MVC و کلاینت‌ها، ظاهر می‌شود. در فصل ۱۹ کنترلرهایی که به جای HTML داده‌های JSON را به مرورگر کلاینت ارسال می‌کنند را به

طور مفصل مورد بررسی قرار خواهیم داد. این کنترلرها به درخواست‌های غیرسنکرون[۱] HTTP امکان می‌دهند که تنها داده‌های مورد نیاز کلاینت را برگردانند. این درخواست‌های غیرسنکرون را به نام درخواست‌های Ajax می‌شناسند. خوب است بدانید هم اینک، JSON به مقدار زیادی قالب XML که با حرف x در Ajax مشخص شده است را حذف کرده است.

در این فصل بر روی نقش دیگر JSON، به عنوان قالب به کار رفته در فایل‌های پیکربندی، تمرکز خواهیم داشت. جدول ۳-۱۳، فایل‌های پیکربندی مختلفی که می‌توانید در پروژه‌های MVC Core مورد استفاده قرار دهید را لیست کرده است.

در ادامه توجه کنید که JSON قالبی برای تعریف اشیای ذخیره شده است و در این وضعیت، دارای هیچ نوع منطق برنامه‌نویسی نیست. از سوی دیگر، فایل‌هایی با پسوند js.، یعنی فایل‌های جاوااسکریپت، برنامه‌هایی قابل اجرا هستند. این بدان معنی است که فایل‌های JSON نمی‌توانند دارای کد جاوااسکریپت باشند، در حالی که فایل‌های js. می‌توانند دارای داده‌های JSON باشند. دلیل این موضوع این است که JSON بر پایه‌ی چگونگی روش تعریف اشیاء در جاوااسکریپت پایه‌ریزی شده است.

نام فایل	شرح کاربرد
Global.json	این فایل که در ریشه‌ی پوشه‌ی مرورگر سالوشن قرار دارد، محل پروژه‌های سالوشن و شماره‌ی نگارش .NET. را برای ویژوال استدیو مشخص می‌کند.
launchSettings.json	این فایل تعیین‌کننده‌ی چگونگی شروع به کار برنامه است.
appsettings.json	این فایل، همان‌گونه که در بخش "کاربرد داده‌های پیکربندی" در همین فصل شرح داده خواهد شد، تنظیمات خاصی که به خود برنامه‌ی کاربردی مربوط می‌شوند را تعریف می‌کند.
bower.json	همان‌گونه که در فصل ۶ گفته شد، Bower از این فایل برای لیست کردن بسته‌های سمت کلاینت نصب شده در پروژه، استفاده می‌کند.
bundleconfig.json	همان‌گونه که در فصل ۶ گفته شد، این فایل برای بسته‌بندی و کمینه کردن حجم فایل‌های جاوااسکریپت و

[۱] Asynchronous HTTP Requests، در حالی که برنامه به ادامه‌ی کار خود مشغول است، داده‌ها را از سرور به کلاینت منتقل می‌کنند. کاربرد آنها در کنترلر به ویژه از refresh صفحه در تبادل اطلاعات بین سرور و کلاینت جلوگیری می‌کند.

نام فایل	شرح کاربرد
	CSS به کار می‌رود.
project.json	همان‌گونه که در فصل ۶ گفته شد، بسته‌های NuGet نصب شده در پروژه در این فایل مشخص می‌شوند. تنظیمات دیگری هم در این فایل برای پروژه موجود است که در بخش "پیکربندی پروژه" شرح داده خواهند شد.
project.lock.json	این فایل که با باز کردن نود project.json در مرورگر سالوشن آشکار می‌شود، شامل جزئیات وابستگی‌های بین بسته‌های نصب شده در پروژه است. این فایل به صورت خودکار ایجاد شده و نباید به صورت دستی ویرایش شود.

۱-۲-۱۳ *پیکربندی سالوشن*

فایل global.json مسئول پیکربندی کلی سالوشن است. در ادامه، محتوای ایجاد شده در این فایل، توسط ویژوال استدیو در هنگام ایجاد پروژه‌ای از نوع ASP.Net Core را می‌بینید:

```
{
  "projects": [ "src", "test" ],
  "sdk": {
    "version": "1.0.0-preview2-003121"
  }
}
```

نود projects در تکه کد بالا، مشخص‌کننده‌ی پوشه‌های دربرگیرنده‌ی پروژه ها و سورس کد سالوشن است. در روش مرسوم، پروژه‌ی قابل انتشار سالوشن، به عنوان مثال، پروژه‌ی MVC در پوشه‌ی src و پروژه‌های آزمایش در پوشه‌ی test جای داده می‌شوند. البته آنچه که گفته شد تنها یک قرارداد است و شما می‌توانید در فایل global.json هر پوشه‌ای را برای هر منظوری معرفی کنید.

نود sdk مشخص کننده‌ی نگارشی از .NET است که برای اجرای پروژه مورد استفاده قرار می‌گیرد. تنظیم این نود، برای همه‌ی پروژه‌های موجود در سالوشن، به کار خواهد رفت.

> اگر در کار با json تازه‌کار هستید، توصیه می‌شود مدت زمانی را به مطالعه‌ی قراردادهای Json در سایت www.json.org اختصاص دهید. قالب فایل‌های json نه تنها ساده هستند، بلکه بسیاری از محیط‌های نرم‌افزاری، از جمله برنامه‌های کاربردی mvc، از تولید و پردازش داده‌های json حمایت

می‌کنند. کامپیوترهای کلاینت این کار را با یک واسط برنامه‌نویسی (API) ساده از نوع جاوااسکریپت، انجام می‌دهند. حقیقت این است که بیشتر برنامه‌نویسان به طور مستقیم با کد json درگیر نمی‌شوند و تنها در فایل‌های پیکربندی با آن برخورد می‌کنند.

برنامه‌نویسان در کار با json معمولا با دو مشکل برخورد می‌کنند. آشنایی با این دو، به ویژه در جایی که ویژوال استدیو و ASP.NET Core توانایی پردازش کد json را نداشته باشند، نقطه‌ی شروع خوبی برای شما خواهد بود.

```
{
  "projects": [ "src", "test" ],
  "sdk": {
    "version": "1.0.0-preview2-003121"
  }
  mysetting : [ fast, slow ]
}
```

نکته‌ی نخست این است که تقریبا همه‌ی عبارات و خاصیت‌ها به جز مقادیر منطقی و عددی باید داخل نمادهای گیومه قرار گیرند:

```
{
  "projects": [ "src", "test" ],
  "sdk": {
       "version": "1.0.0-preview2-003121"
    }
  "mysetting" : [ "fast", "slow"]
}
```

نکته‌ی دوم این است که در هنگام نسبت دادن خاصیتی به شیئی در json، پس از آکولاد انتهایی خاصیت پیشین، باید از یک نماد کاما استفاده شود:

```
{
  "projects": [ "src", "test" ],
  "sdk": {
       "version": "1.0.0-preview2-003121"
    },
  "mysetting" : [ "fast", "slow"]
}
```

۱۳-۲-۲ پیکربندی پروژه

فایل project.json برای پیکربندی یک پروژه در سالوشن به کار می‌رود. در ادامه، محتوای پیش‌فرض این فایل برای یک پروژه‌ی MVC که توسط الگوی پروژه‌ی خالی (Empty Template) ایجاد شده است را می‌بینید:

```
{
  "dependencies": {
    "Microsoft.NETCore.App": {
    "version": "1.0.0",
    "type": "platform"
    },
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0"
  },
  "tools": {
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-
    final"
  },
  "frameworks": {
    "netcoreapp1.0": {
      "imports": ["dotnet5.6", "portable-net45+win8"]
    }
  },
    "buildOptions": { "emitEntryPoint": true, "preserveCompilationContext":
    true },
    "runtimeOptions": {
      "configProperties": {
      "System.GC.Server": true
    }
  },
  "publishOptions": {
    "include": ["wwwroot", "web.config"]
  },
  "scripts": {
    "postpublish": [ "dotnet publish-iis --publish-folder
    %publish:OutputPath% --framework
    %publish:FullTargetFramework%" ]
  }
}
```

بخش‌های مهم فایل project.json در زیر، شرح داده شده‌اند. مهمترین قسمت‌ها عبارتند از dependencies و tools.

- dependencies: مشخص‌کننده‌ی بسته‌های نرم‌افزاری است که پروژه به آنها وابسته است.
- tools: تعیین کننده‌ی بسته‌هایی است که به عنوان ابزار توسعه در پروژه به کار می‌روند.
- frameworks: تعیین کننده‌ی نگارش‌هایی از .NET و فایل‌های مورد نیاز آنها است که در پروژه به کار می‌روند.
- buildOptions: روش کامپایل پروژه (ها) و تنظیمات مربوط به آن، در این بخش تعیین می‌شود.
- runtimeOptions: چگونگی اجرای پروژه در این بخش مشخص می‌شود.
- publishOptions: چگونگی انتشار پروژه در این بخش مشخص می‌شود.
- scripts: تعیین کننده‌ی فرامینی است که در نقاط کلیدی چرخه‌ی ساخت پروژه، مانند زمان پیش از انتشار آن، باید اجرا شوند.

۱-۲-۲-۱۳ *تنظیمات بخش dependencies*

در روند توسعه‌ی پروژه و افزودن بسته‌هایی برای ایجاد کارآیی‌های جدید، بیشتر از همه با این بخش سروکار خواهید داشت. در لیست ۵-۱۳، ویژگی‌های پایه و مهم برای توسعه‌ی یک پروژه MVC آورده شده‌اند.

لیست ۵-۱۳: ویژگی‌های مهم بخش وابستگی‌ها برای یک پروژه‌ی MVC

```
...
"dependencies": {
  "Microsoft.NETCore.App": {
    "version": "1.0.0",
    "type": "platform"
  },
    "Microsoft.AspNetCore.Diagnostics": "1.0.0",
    "Microsoft.AspNetCore.Server.IISIntegration": "1.0.0",
    "Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
    "Microsoft.Extensions.Logging.Console": "1.0.0",
    "Microsoft.AspNetCore.StaticFiles": "1.0.0",
    "Microsoft.AspNetCore.Mvc": "1.0.0",
    "Microsoft.VisualStudio.Web.BrowserLink.Loader": "14.0.0",
    "Microsoft.AspNetCore.Razor.Tools": {
        "version": "1.0.0-preview2-final",
```

```
        "type": "build"
    }
},
...
```

برای بیشتر بسته‌ها می‌توانید از نوشتار ساده، یعنی نام بسته و شماره‌ی نگارش آن استفاده کنید:

```
...
"Microsoft.AspNetCore.Mvc": "1.0.0",
...
```

در نوشتار کامل‌تر، می‌توانید نوع وابستگی، که در چگونگی کاربرد آن موثر خواهد بود را نیز بیاورید:

```
...
"Microsoft.NETCore.App": {
  "version": "1.0.0",
  "type": "platform"
},
...
```

خاصیت version تعیین کننده‌ی نگارش بسته‌ی مورد نظر است. خاصیت type اطلاعات بیشتری در مورد نقش بسته‌ی نرم‌افزاری به دست می‌دهد. همانطور که در ادامه آورده شده است، یکی از سه مقدار زیر را می‌توانید برای این خاصیت تعیین کنید:

- Default: به معنی یک وابستگی عادی است و تنها نشان می‌دهد که برنامه برای اجرا، به اسمبلی‌های درون بسته نیازمند است.
- Platform: به معنی آن است که بسته‌ی مورد گفتگو، ویژگی‌هایی در سطح سیستم‌عامل ارائه می‌دهد. برای بسته‌ی Microsoft.NETCore.App باید از این مقدار استفاده شود.
- build: به معنی این است که اسمبلی‌های موجود در بسته، تنها در فرآیند ساخت و کامپایل به کار رفته است و تأثیری در مرحله‌ی اجرا نخواهند داشت. برای بسته‌ی Visual Studio Scaffolding، که شرح چگونگی پیکربندی آن را در فصل ۸ مطالعه کردید، باید این مقدار را به کار برید.

۲-۲-۲-۱۳ *تنظیمات بخش tools*

برخی از بسته‌هایی که به بخش dependencies اضافه می‌شوند، مربوط به ابزارهای مورد نیاز در فرآیند توسعه‌ی نرم‌افزار هستند و باید در نود tools معرفی گردند. در لیست ۶-۱۳، اضافه شدن بسته‌ی Microsoft.AspNetCore.Razor.Tools را که فراهم کننده‌ی ویژگی IntelliSense در ویرایشگر ویژوال‌استدیو، برای نماهای razor است می‌بینید.

لیست ۶-۱۳: معرفی ابزارهای توسعه در project.json

```
...
"tools": {
  "Microsoft.AspNetCore.Razor.Tools": "1.0.0-preview2-final",
  "Microsoft.AspNetCore.Server.IISIntegration.Tools": "1.0.0-preview2-final"
},
...
```

هنگامی که بخواهید ابزاری به پروژه اضافه کنید، بسته‌ی نرم‌افزاری مربوط به آن، معمولا فرامین لازم برای ثبت مشخصات آن در فایل project.json را به همراه خود دارد. معمول‌ترین ابزاری که با آنها برخورد می‌کنید، افزون بر بسته‌ای که برای تگ‌های کمکی (tag helper) در لیست ۶-۱۳ می‌بینید، بسته‌ای است که فرامین Entity Framework Core برای مدیریت پایگاه‌های داده را به پروژه اضافه می‌کند (فصل ۸).

۳-۱۳ آشنایی با کلاس Program

کلاس program که در فایل Program.cs تعریف می‌شود، فراهم‌کننده‌ی متد ()main اصلی برنامه است. در این متد، افزون بر پیکربندی محیط میزبانی (Host) پروژه، کلاس مربوط به پیکربندی خود پروژه هم انتخاب می‌شود. لیست ۷-۱۳، محتوای پیش‌فرض این کلاس که توسط ویژوال‌استدیو به پروژه اضافه می‌شود را نشان می‌دهد.

معمولا برای بیشتر پروژه‌ها و محیط‌های میزبانی متداول، نیازی به تغییر فایل کلاس Program نخواهید داشت. در بخش مربوط به استفاده از پیکربندی‌های پیچیده در همین کتاب، با مواردی از تغییرات این فایل برخورد خواهید داشت؛ ولی همان‌گونه که گفته شد برای بیشتر محیط‌های میزبانی مانند IIS و Azure نیازی به این کار نخواهد بود.

لیست ۷-۱۳: کد پیش‌فرض کلاس program

```
using System.IO;
using Microsoft.AspNetCore.Hosting;

namespace ConfiguringApps {
  public class Program {
    public static void Main(string[] args) {
      var host = new WebHostBuilder()
      .UseKestrel()
      .UseContentRoot(Directory.GetCurrentDirectory())
      .UseIISIntegration()
      .UseStartup<Startup>()
      .Build();
```

```
            host.Run();
        }
    }
}
```

همان‌طور که در ادامه آورده شده، نخستین عبارت متد ()main، برای بر پا کردن محیط میزبانی، اقدام به ایجاد شیء WebHostBuilder کرده و پس از آن، متدهایی را با نمونه‌ی ایجاد شده فراخوانی می‌کند.

- متد ()UseKestrel: سرویس‌دهنده‌ی وب Kestrel را پیکربندی می‌کند.
- متد ()UseContentRoot: دایرکتوری ریشه‌ی پروژه که برای بارگذاری فایل‌های پیکربندی و محتوای استاتیک مانند تصاویر به کار می‌رود را پیکربندی می‌کند.
- متد ()UseIISIntegration: امکان استفاده از IIS و IIS Express را فراهم می‌کند.
- متد ()UseStartup: این متد مشخص‌کننده‌ی کلاسی است که باید برای پیکربندی ASP.NET به کار رود.
- متد ()Build: پیکربندی‌هایی که توسط سایر متدها انجام شده است را ترکیب کرده و برای استفاده آماده می‌کند.

پس از آماده شدن پیکربندی، عبارت بعدی متد ()Main، با فراخوانی متد ()Run، اقدام به اجرای برنامه می‌کند. در این نقطه، نرم‌افزار میزبان (مثلا IIS) آماده‌ی دریافت درخواست‌های HTTP و ارسال آنها برای برنامه، به منظور پردازش آنهاست.

اجرای مستقیم KESTREL: اگر به فرآیند افزودن بسته‌ها به پروژه دقت کرده باشید، متوجه شده‌اید که یکی از نودهایی که به صورت خودکار به بخش dependencies اضافه می‌شود، Kestrel است:

```
...
"Microsoft.AspNetCore.Server.Kestrel": "1.0.0",
...
```

این نرم‌افزار، وب سرور جدیدی است که برای اجرای برنامه‌های ASP.NET Core، با ویژگی قابلیت اجرا بر روی سکوهای مختلف[۱]، طراحی شده است. وقتی برنامه‌های ASP.NET Core را با میزبانی IIS (که سکوی وب مرسوم برای اجرای ASP.NET است) یا IIS Express (که سکوی وب مایکروسافت برای مرحله‌ی توسعه است) اجرا می‌کنید، Kestrel هم به طور خودکار اجرا می‌شود.

اگر بخواهید می‌توانید این نرم‌افزار را به طور مستقیم اجرا کنید. این به معنی کنار گذاشتن وابستگی به وب سرور IIS ویندوز است. دو راه برای اجرای مستقیم برنامه با میزبانی Kesterl وجود دارد. در

[۱] منظور، وب سرورهای مختلف است.

روش نخست، باید پس از کلیک بر روی نشانه‌ی کوچک فلش مانند در سمت راست دکمه‌ی IIS Express در نوار ابزار ویژوال استدیو، نام پروژه‌ی خود را (که در حال اجراست) انتخاب کنید. این کار موجب باز شدن یک پنجره‌ی جدید فرمان (Command Prompt) برای اجرای برنامه با Kesterl می‌شود.

در روش دوم، می‌توانید با گشودن یک پنجره‌ی فرمان به طور مستقیم و سپس حرکت به پوشه‌ای که فایل‌های پیکربندی در آن ذخیره شده‌اند نیز به همان نتیجه‌ی نخست برسید (پوشه‌ی حاوی project.json). در مرحله‌ی بعد، باید فرمان زیر را اجرا کنید:

```
Dotnet run
```

سرور Kestrel به طور پیش‌فرض بر روی درگاه ۵۰۰۰ منتظر دریافت درخواست‌های HTTP می‌شود.

۱۳-۴ آشنایی با کلاس Startup

ASP.NET Core از کلاسی به نام Startup برای پیکربندی کارآیی‌های برنامه استفاده می‌کند. کلاس پیکربندی[1] در فایل Startup.cs که ویژوال استدیو به ریشه‌ی پروژه اضافه می‌کند، تعریف می‌شود. بررسی کارکرد این کلاس، دید خوبی در مورد روش پردازش درخواست‌های HTTP و چگونگی ترکیب MVC با ASP.NET به دست می‌دهد.

در این بخش، کار را با ساده‌ترین شکل کلاس Startup شروع می‌کنیم و گام به گام با افزودن ویژگی‌های جدید، در پایان به کلاسی خواهیم رسید که برای بسیاری از پروژه‌ها کاربرد خواهد داشت. در نقطه‌ی شروع، لیست ۸-۱۳، کلاسی را نشان می‌دهد که ویژوال استدیو به الگوی پروژه‌ی خالی اضافه می‌کند و تنها کارآیی آن، امکان پاسخ به درخواست‌های HTTP است.

لیست ۸-۱۳: ساده‌ترین شکل کلاس Startup

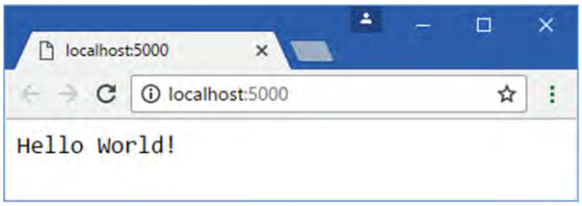
```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace ConfiguringApps {
  public class Startup {
    public void ConfigureServices(IServiceCollection services) {
    }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory) {
       loggerFactory.AddConsole();
```

[1] Configuration Class

```
            if (env.IsDevelopment()) {
                app.UseDeveloperExceptionPage();
            }
        app.Run(async (context) => {
            await context.Response.WriteAsync("Hello World!");
        });
    }
  }
}
```

متدهای ConfigureServices() و Configure() که ویژگی‌های مورد نیاز برنامه و چگونگی کاربرد آنها را برای ASP.NET مشخص می‌کنند، در کلاس Startup تعریف می‌شوند. در بخش‌های آینده در مورد چگونگی اجرای این دو متد، صحبت خواهیم کرد. تنها کارآیی ایجاد شده توسط این کلاس ساده، توانایی پاسخ به درخواست‌های HTTP و ایجاد یک پیام ساده است. اجرای برنامه این پیام را نمایش می‌دهد (شکل ۱-۱۳).



شکل ۱-۱۳

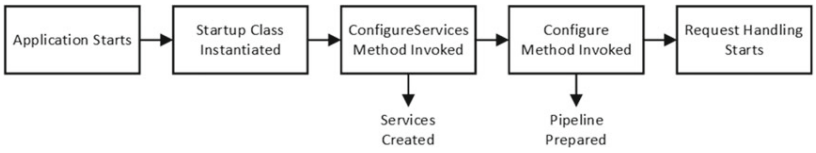
۱-۴-۱۳ کارکرد کلاس Startup

با شروع اجرای برنامه، ASP.NET نمونه‌ی جدیدی از کلاس Startup را ایجاد کرده و متد ConfigureServices() آن را فراخوانی می‌کند، که به نوبه‌ی خود سرویس‌های برنامه را راه‌اندازی می‌کند. به عنوان تعریفی بسیار کلی از سرویس‌ها (که در بخش آشنایی با سرویس‌های ASP.NET به صورت مشروح بررسی خواهند شد)، می‌توان گفت سرویس شیئی است که کارآیی‌های مورد نیاز سایر بخش‌های برنامه‌ی کاربردی را فراهم می‌کند.

پس از ایجاد سرویس‌ها، متد Configure() فراخوانی می‌شود. هدف این متد تشکیل عناصر زنجیره‌ی[1] درخواست‌هاست. این زنجیره که مانند یک صف عمل می‌کند، متشکل از عناصری به نام میان‌افزار[2] است که برای مدیریت درخواست‌های رسیده، پردازش آنها و ایجاد پاسخ مناسب به کار می‌روند. شرح جزئیات کارکرد این زنجیره و چگونگی ایجاد عناصر میان‌افزار را به بخش آشنایی با میان‌افزار در

[1] Request Pipeline Components
[2] Middleware

ASP.NET واگذار می‌کنیم. شکل ۲-۱۳، روش بـه کـارگیری کـلاس Startup توسـط ASP.NET را نشـان می‌دهد.



شکل ۲-۱۳

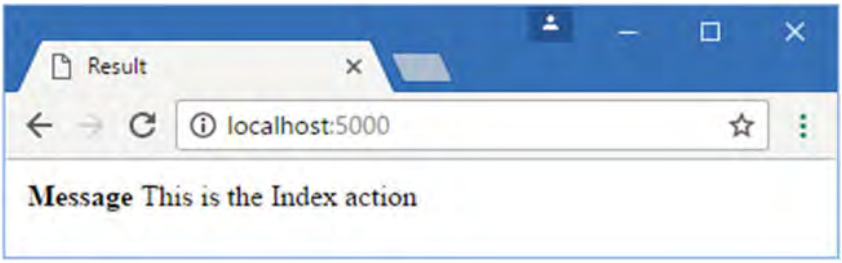
کار با کلاسی که برای همه‌ی درخواست‌ها پیام Hello, World را ارسال می‌کند، جالـب نیسـت. بنـابراین پیش از ورود به شرح جزئیات کارکرد متدهای کلاس، بهتر است MVC را، همانطور که در لیسـت ۹-۱۳ نشان داده شده، فعال کنیم.

لیست ۹-۱۳: فعال‌سازی MVC در کلاس Startup

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;

namespace ConfiguringApps {
  public class Startup {
    public void ConfigureServices(IServiceCollection services) {
      services.AddMvc();
    }
    public void Configure(IApplicationBuilder app, IHostingEnvironment env,
    ILoggerFactory loggerFactory) {
      app.UseMvcWithDefaultRoute();
    }
  }
}
```

با کارهای انجام شده، هم اینک زیرساخت کافی برای دریافت و پردازش درخواست‌های HTTP و تولیـد پاسخ مناسب به وسیله‌ی کنترلرها و نماها، به وجود آمده است. اگر برنامه را اجرا کنید، نتیجه‌ای ماننـد شکل ۳-۱۳ را خواهید دید.



شکل ۳-۱۳

توجه کنید که محتوای ایجاد شده دارای هیچ قالب خاصی نیست. پیکربندی ساده‌ای که در لیست ۹-۱۳ انجام شد، امکانی برای استفاده از محتوای استاتیک مانند CSS و فایل‌های جاوااسکریپت، فراهم نمی‌کند. بنابراین با این که تگ link در HTML تولید شده‌ی نمای Index.cshtml درخواستی برای CSS بوت استرپ خواهد داشت، ولی برنامه توانایی پردازش آن را نخواهد داشت. این مشکل را در بخش‌های آینده بر طرف خواهیم کرد.

۱۳-۴-۲ آشنایی با سرویس‌های ASP.NET

همانطور که گفته شد، ASP.NET متد Startup.ConfigureServices() را برای ایجاد سرویس‌های مورد نیاز پروژه فراخوانی می‌کند. هر شیئی که به نوعی کارآیی (های) مورد نیاز سایر بخش‌های برنامه را فراهم می‌کند را می‌توان service نامید. به عنوان مثال، پس از ساختن پوشه‌ای به نام Infrastructure در ریشه‌ی پروژه، کلاسی به نام UptimeService.cs با توجه به کد لیست ۱۰-۱۳ را در آن ایجاد کنید.

لیست ۱۰-۱۳: کد کلاس UptimeService در پوشه‌ی Infrastructure

```
using System.Diagnostics;

namespace ConfiguringApps.Infrastructure {
  public class UptimeService {
    private Stopwatch timer;
    public UptimeService() {
      timer = Stopwatch.StartNew();
    }
    public long Uptime => timer.ElapsedMilliseconds;
  }
}
```

با ایجاد نمونه‌ای از این کلاس، متد سازنده‌ی آن تایمری را راه‌اندازی می‌کند که می‌تواند زمان اجرای برنامه را اندازه‌گیری و ارائه دهد. این مثال ساده، نمونه‌ی خوبی از یک سرویس را نشان می‌دهد، زیرا هم باید در ابتدای اجرای برنامه کار خود را شروع کند و هم سایر بخش‌های برنامه می‌توانند از کارآیی آن (دریافت طول زمان اجرای برنامه)، استفاده کنند.

کار ثبت (Register) سرویس‌های ASP.NET باید در متد ()ConfigureServices کلاس Startup انجام شود. لیست ۱۱-۱۳، ثبت کلاس UptimeService را به عنوان یک سرویس در کلاس Startup نشان می‌دهد.

لیست ۱۱-۱۳: ثبت سرویس UptimeService در کلاس startup

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using ConfiguringApps.Infrastructure;

namespace ConfiguringApps {
     public class Startup {
       public void ConfigureServices(IServiceCollection services) {
          services.AddSingleton<UptimeService>();
          services.AddMvc();
       }
       public void Configure(IApplicationBuilder app, IHostingEnvironment env,
       ILoggerFactory loggerFactory) {
          app.UseMvcWithDefaultRoute();
     }
   }
}
```

پارامتر دریافتی متد ()ConfigureServices شیئی است که باید از اینترفیس IServiceCollection ارث‌بری کرده باشد. در حقیقت، سرویس‌ها توسط متدهای توسعه یافته‌ای که هر کدام از آنها بر پایه‌ی نیاز خود، اینترفیس گفته شده را پیاده‌سازی کرده باشند، ثبت می‌شوند. این متدهای توسعه یافته ممکن است انتخاب‌های متفاوتی را برای پیکربندی تعریف کرده باشند. در مورد این انتخاب‌ها در آینده مطالب بیشتری خواهیم داشت ولی در اینجا همان‌طور که می‌بینید از متد ()AddSingleton استفاده شده است. نتیجه‌ی اجرای این متد این است که تنها یک نمونه از UpTimeService برای کل پروژه استفاده خواهد شد.

سرویس‌ها به مقدار زیادی بر پایه‌ی ویژگی خاصی به نام تزریق وابستگی (Dependency Injection)، کار می‌کنند. همین ویژگی که در فصل ۱۷ به صورت مشروح بررسی خواهد شد، امکان کاربرد آسان سرویس‌ها را برای کنترلرها فراهم می‌کند. برای دسترسی به سرویس‌هایی که توسط متد ()Startup.ConfigureServices ثبت می‌شوند، باید سازنده‌ای داشته باشید که پارامتر دریافتی آن از نوع سرویس مورد نظر باشد. لیست ۱۲-۱۳، چنین سازنده‌ای را برای دسترسی به سرویس

UpTimeService آورده شده در لیست ۱۱-۱۳، نشان می‌دهد. همچنین توجه کنید که کد متد اکشن Index() برای ارسال مقدار خاصیت UpTime سرویس برای نما، هم تغییر کرده است.

لیست ۱۲-۱۳: استفاده از سرویس در کنترلر Home

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using ConfiguringApps.Infrastructure;

namespace ConfiguringApps.Controllers {
  public class HomeController : Controller {
    private UptimeService uptime;
    public HomeController(UptimeService up) {
      uptime = up;
    }
    public ViewResult Index()
    => View(new Dictionary<string, string> {
      ["Message"] = "This is the Index action",
      ["Uptime"] = $"{uptime.Uptime}ms"
    });
  }
}
```

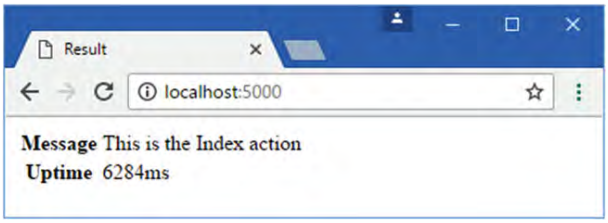
هرگاه که MVC برای پاسخ دادن به یک درخواست نیاز به نمونه‌ای از کنترلر Home داشته باشد، با توجه به سازنده‌ی کلاس کنترلر، مجبور به ایجاد نمونه‌ای از UpTimeService می‌شود. سپس با مراجعه به سرویس‌های ثبت شده در Startup درمی‌یابد که تنها یک شیء از این سرویس باید برای کل پروژه به کار رود. در نتیجه، تنها نمونه‌ی ایجاد شده از سرویس را برای سازنده‌ی کنترلر ارسال می‌کند.

ثبت و اجرای سرویس‌ها می‌تواند به مراتب از آنچه که نشان داده شد پیچیده‌تر باشد. ولی این مثال ساده، ایده‌ی اصلی تعریف یک سرویس در کلاس Startup و چگونگی استفاده از کارآیی ایجاد شده توسط آن در پروژه را نشان داد.

همانطور که در شکل ۴-۱۳ می‌بینید، اجرای برنامه و درخواست آدرس پیش‌فرض آن، مدت زمان گذشته از زمان اجرا را بر حسب میلی ثانیه نشان می‌دهد.

با هر درخواست جدید برای آدرس پیش‌فرض، نمونه‌ی جدیدی از کلاس HomeController و در پی آن، داده‌ی جدیدی از شیء به اشتراک گذاشته شده‌ی UpTimeService به دست می‌آید. این روش، به کنترلر

Home امکان می‌دهد بدون نیاز به دانستن روش محاسبه یا پیاده‌سازی آن، به مدت زمان اجرای برنامه دسترسی داشته باشد.



شکل ۴-۱۳

۱-۲-۴-۱۳ آشنایی با سرویس‌های MVC

یک بسته‌ی نرم‌افزاری به پیچیدگی MVC از سرویس‌های فراوانی، هم برای کاربرد داخلی خودش و هم برای ایجاد کارآیی‌های مورد نیاز برنامه‌نویس، استفاده می‌کند. بسته‌های نرم‌افزاری برای ایجاد سرویس‌های مورد نظرشان متدهای توسعه‌یافته‌ای را که تنها یک متد را فراخوانی می‌کنند، به کار می‌برند. این متد برای MVC، متد ()AddMvc است که در کلاس Startup به کار بردیم.

```
...
public void ConfigureServices(IServiceCollection services) {
  services.AddSingleton<UptimeService>();
  services.AddMvc();
}
...
```

این متد بدون پر کردن کد متد ()ConfigureServices با لیست بلندی از اسامی سرویس‌ها، همه‌ی سرویس‌های لازم برای MVC را در اختیار می‌گذارد.

۳-۴-۱۳ آشنایی با میان‌افزارهای MVC

در ASP.NET Core، اصطلاح میان‌افزار به عناصر تشکیل‌دهنده‌ی زنجیره‌ی درخواست‌ها گفته می‌شود. زنجیره‌ی درخواست‌ها که مانند یک صف عمل می‌کند شبیه زنجیری است که این عناصر حلقه‌های آن را تشکیل می‌دهند و درخواست جدید همواره به نخستین عنصر تشکیل‌دهنده‌ی این صف سپرده می‌شود. عنصر گفته شده درخواست را بررسی کرده و بر پایه‌ی آن تصمیم می‌گیرد که آیا پاسخ مناسب را تولید کند و یا آن را به عنصر بعدی موجود در صف، ارسال کند. اگر پاسخی تولید شود، به عناصر قبلی سپرده می‌شود، یعنی به طرف عقب در صف برگشت می‌کند و این به عناصر پیشین امکان می‌دهد که در صورت لزوم آن را تغییر دهند.

روش کارکرد میان‌افزار به صورتی که شرح داده شد ممکن است عجیب به نظر برسد ولی در واقع انعطاف‌پذیری بسیاری در رابطه با چگونگی شکل‌گیری پروژه (و آماده شدن پاسخ درخواست) ایجاد