آدام فریمن

آموزش حرفه ای
ASP.NET Core 6

جلد دوم:

کار با داده ها

مروّت گیوی

ترجمک
arjomac

آدام فریمن

آموزش حرفه‌ای ASP.NET Core 6

جلد دوم: کار با داده

ترجمه

مروّت گیوی

انتشارات ترجمک

تابستان ۱۴۰۲

آموزش حرفه‌ای ASP.NET CORE 6

توسعه وب با ASP.NET Core

توجه :

کتاب حاضر ترجمه **مروّت گیوی** از انتشارات ترجمک می‌باشد. فایل کتاب حاوی اطلاعات DRM (مدیریت حقوق دیجیتال) است. وقتی برای اولین بار فایل را باز می کنید، کد شناسایی کتاب به همراه آدرس IP سیستم شما ذخیره شده و زمانیکه آنلاین شوید، به سرور انتشارات ترجمک انتقال می یابد.

خواهشمند است به حقوق مترجمان و گروه فنی ترجمک احترام گذاشته و از توزیع بدون مجوز فایل کتاب اجتناب نمایید. شما با خرید و دانلود این کتاب موافقت نموده اید که اطلاعات فایل DRM به سرور انتشارات ترجمک انتقال یابد و در صورت محرز شدن نقض حقوق صاحب اثر، کلیه خسارات حاصله در طی فرآیند حقوقی و مطابق قانون حمایت حقوق مؤلفان و مصنفان و هنرمندان و ناشران جمهوری اسلامی (مصوب دوازده اسفند ۱۳۶۵ یا بعد از آن) از شما دریافت شود.

از اینکه با عرضه مقرون به صرفه کتاب های الکترونیک و شکوفایی انتشارات ترجمک همیاری می کنید، سپاسگزاریم.

انتشارات ترجمک

برنامه نویسی و وب

عنوان و نام پدیدآور	:	آموزش حرفه‌ای ASP.NET Core 6/آدام فریمن ؛ ترجمه مروت گیوی.
مشخصات نشر	:	همدان: ترجمک، ۱۴۰۲
مشخصات ظاهری	:	جلد ۲: کار با داده
شابک	:	دوره: ۱-۵۳-۷۸۵۵-۶۲۲-۹۷۸؛ جلد ۲: ۴-۴۹-۷۸۵۵-۶۲۲-۹۷۸
موضوع	:	مایکروسافت دات‌نت فریم‌ورک، Microsoft .NET Framework، وبگاه‌ها – برنامه‌های تالیفی، Web sites -- Authoring programs، علوم کامپیوتر، Computer science، نرم‌افزار – مهندسی، Software engineering
مندرجات	:	جلد۱: ساخت وبسایت تجارت الکترونیک، ۳: ویژگی های پیشرفته، ۴: اجرای پروژه مثال
عنوان دیگر	:	آموزش کاربردی Pro ASP.NET Core MVC 6 همراه با پیاده‌سازی کامل یک پروژه با MVC و صفحات Razor
رده بندی کنگره	:	QA76/76
رده بندی دیویی	:	005/276
شماره کتابشناسی ملی	:	۹۳۰۱۹۶۲

شناسنامه کتاب

نام کتاب: آموزش حرفه‌ای ASP.NET Core 6: جلد دوم: کار با داده

ترجمه: مروت گیوی

ناشر: انتشارات ترجمک

صفحه آرایی: انتشارات ترجمک

طراحی جلد: محمدحسین گیوی

نوبت چاپ: چاپ اول، ۱۴۰۲

قیمت: ۶۵۰۰۰ تومان

چاپ: گروه نشر الکترونیک ترجمک

شابک دوره: ۹۷۸-۶۲۲-۷۸۵۵-۵۳-۱

شابک جلد: ۹۷۸-۶۲۲-۷۸۵۵-۴۹-۴

تلفن تماس: ۰۹۱۸۱۵۰۶۱۰۰

تارنمای اینترنتی: https://tarjomac.ir

ISBN : 978-622-7855-53-1

ISBN : 978-622-7855-49-4

مقدمه

کتاب آموزش حرفه‌ای ASP.NET Core 6 آدام فریمن که اکنون چاپ نهم خود را منتشر کرده است یکی از منابع اصلی و قدرتمند فراگیری توسعه وب و نوشتن اپلیکیشن های وب کامل و عملیاتی است.

به منظور تسهیل تهیه کتاب بر حسب سطح برنامه نویسی و نیاز هر فرد، کتاب به چهار جلد براساس بخش های کتاب اورجینال تقسیم و منتشر شده است. مخاطبان این کتاب برنامه نویسان و توسعه دهندگان وب است که دانش پایه تولید وبسایت و زبان C# را داشته و می خواهند کارهای خود را تقویت و به روز رسانی کنند.

نرم افزار ویژوال بیسیک ۲۰۲۳ و تکنولوژی دات نت ۶ در این کتاب استفاده شده و با پیروی گام به گام از فصول کتاب می توان اپلیکیشن وب کاملا حرفه ای و قدرتمند تولید کرد. امید است خوانندگان این مجموعه کتاب چهار جلدی بتوانند وبسایت های مورد نظر خود را تولید کنند. ساختار کتاب به صورت زیر است:

- جلد اول: ساخت وبسایت تجارت الکترونیک
- جلد دوم: کار با داده ها
- جلد سوم: ویژگی های پیشرفته
- جلد چهارم: اجرای پروژه نمونه

مترجم

پیشگفتار

برنامه نویسان و طراحان وب حرفه ای با استفاده از راهنمایی های این کتاب پرفروش، که اکنون در نسخه نهم آن است و برای ASP.NET Core for .NET 6 به روز شده است، برنامه های ناب‌تری را برای پلتفرم ASP.NET Core تولید خواهند کرد. کتاب شامل توضیحات مفصلی در مورد پلتفرم ASP.NET Core و اپلیکیشن فریمورک‌های مورد پشتیبانی آن است.

این راهنمای بنیادی به زمینه چینی برای ASP.NET Core 6 می پردازد و ابزارها و تکنیک های مورد نیاز برای ساختن اپلیکیشن های وب مدرن و قابل توسعه را معرفی می کند. ویژگی ها و قابلیت های جدیدی مانند MVC، Razor Pages، Blazor Server و Blazor WebAssembly همراه با مثال هایی از کاربرد آنها پوشش داده شده است.

ASPNET Core 6 آخرین تحول پلتفرم وب ASP.NET مایکروسافت است و یک فریمورک «آگونیست میزبان» و یک مدل برنامه‌نویسی با بهره‌وری بالا ارائه می‌کند که معماری کد تمیزتر، توسعه مبتنی بر تست و بسط پذیری قدرتمند را ترویج می‌کند.

آدام فریمن، نویسنده این کتاب پیشروی بازار، کتاب را به طور کامل بازبینی کرده است و توضیح می دهد که چگونه از ASP.NET Core بیشترین بهره را ببرید. او با مباحث مقدماتی شروع می‌کند، به شما در مورد کامپوننت های میان‌افزار، سرویس‌های سرخود، درخواست پیوند مدل و موارد دیگر آموزش می‌دهد. با کسب دانش و اعتماد به نفس، او موضوعات پیچیده تر و ویژگی های پیشرفته تر، از جمله مسیریابی نقطه پایان و تزریق وابستگی را معرفی می کند. او آنقدر عمقی پیش می رود تا دانش مورد نیاز را به شما بدهد.

این کتاب از همان قالب و سبک نسخه‌های محبوب قبلی پیروی می‌کند، اما همه چیز را برای نسخه جدید ASP.NET Core یعنی برای NET 6 بروز می‌کند و تمرکز مطلب را بسط می‌دهد تا تمام پلتفرم ASP.NET Core را در بر بگیرد. یک مطالعه موردی کامل بصورت یک اپلیکیشن وب ASP.NET Core ارائه شده که می توانید به عنوان یک الگو برای پروژه های خود از آن استفاده کنید.

کد منبع این کتاب را می‌توانید در https://github.com/Apress/pro-asp.net-core-6 بیابید.

آنچه خواهید آموخت:

- کاوش کل پلتفرم ASP.NET Core
- اعمال ویژگی های جدید ASP.NET Core 6 به محیط توسعه خود
- نحوه ایجاد خدمات وب RESTful، اپلیکیشن های وب و اپلیکیشن های سمت کلاینت
- راه اندازی سریع و موثر مدل های برنامه نویسی جدید بر پایه دانش موجود خود

آدام فریمن یک متخصص IT با تجربه است که پست مدیر ارشد را در طیف وسیعی از شرکت‌ها داشته است و اخیراً به عنوان مدیر ارشد فناوری و مدیر عامل یک بانک جهانی خدمت کرده است. او که اکنون بازنشسته شده است، وقت خود را صرف نوشتن و دویدن در مسافت های طولانی می کند.

مخاطبان این کتاب

مخاطبان این کتاب توسعه دهندگان وب با دانش اولیه توسعه وب و C# است که می خواهند آخرین پیشرفت ها و عملکردهای ASP.NET Core را در پروژه های خود بگنجانند.

اطلاعات کتابشناختی کتاب اورجینال

عنوان کتاب: ASP.NET Core ۶ پرو: ساخت اپلیکیشن های وب آماده کلاود با استفاده از صفحات MVC، بلیزور و ریزور

Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages

نویسنده: آدام فریمن (Adam Freeman)

کد DOI: 10.1007/978-1-4842-7957-1

ناشر: انتشارات ای پرس، برکلی، کالیفرنیا

تاریخ انتشار: ۲۵ فوریه ۲۰۲۲

شابک (پرینت): 978-1-4842-7956-4

شابک (ایبوک): 978-1-4842-7957-1

نوبت چاپ: ۹ (9th edition)

تعداد صفحات: XXXIII، ۱۲۵۳

ساختار کتاب

کتاب ASP.NET Core 6 پرو: ساخت اپلیکیشن های وب آماده کلاود با استفاده از صفحات MVC، بلیزور و ریزور در چهار جلد تهیه شده است.

جلد اول کتاب همان بخش اول کتاب اورجینال می باشد که در ۱۱ فصل به مقدمات کار پرداخته است. در این بخش از پروژه مثال SportsStore استفاده شده است و مقدمات توسعه وب از قبیل آماده سازی محیط توسعه آموزش داده شده است. ابتدا نحوه تهیه و نصب نرم افزارهای ویژوال استودیو و ویژوال استودیو کد آموزش داده شده است و طریقه نصب پکیج های میان افزار و فریمورک ها بحث شده است. سپس با استفاده از پروژه فروشگاه ورزشی به آموزش نحوه تولید صفحات، منو، سبد خرید و امنیت و انتقال اپلیکیشن پرداخته شده است.

جلد دوم کتاب با معرفی پلتفرم ASP.NET Core شروع می شود. سپس در ۶ فصل متوالی به آموزش نحوه مسیریابی، تزریق وابستگی، استفاده از ویژگی های خاص پلتفرم و نهایتا کار با داده ها پرداخته شده است.

جلد سوم شامل فصل ۱۸ تا فصل ۳۱ است. در این جلد از کتاب که همان بخش سوم کتاب اورجینال است، ابتدا یک پروژه مثال ساخته می شود، سپس به وب سرویس ها، ویژگی های وب سرویس ها، کار با کنترلرها و نماها، استفاده از صفحات ریزور، کار با کامپوننت نما، تگ هلپرهای سفارشی و تگ هلپرهای سرخود، پیوند مدل، اعتبارسنجی مدل، استفاده از فیلتر و ساخت یک اپلیکیشن فرم پرداخته می شود.

جلد چهارم کتاب شامل فصل ۳۲ تا فصل ۳۹ است. در این جلد که بخش چهارم کتاب اورجینال است، ابتدا یک پروژه مثال ساخته می شود، سپس به بحث در مورد بلیزور سرور، بلیزور وب اسمبلی، ویژگی های پیشرفته بلیزور، استفاده از آیدنتیتی برای احراز هویت و تعیین مجوزها و نقش پرداخته می شود.

این کتاب شامل یک پکیج آنلاین است که کلیه کدهای مورد استفاده در کتاب، پروژه ها و رفع خطای کتاب در آن گنجانده شده است و از مخزن گیتهاب ناشر قابل دسترسی است.

آدام فریمن

محتوای کتاب در یک نگاه

فصل	صفحه

فهرست مندرجات کتاب

عنوان	صفحه

فصل ۱۲

شناخت پلتفرم ASP.NET Core

پلتفرم ASP.NET Core بنیانی برای خلق اپلیکیشن‌های وب است و ویژگی هایی را ارائه می دهد که به فریمورک‌هایی مانند MVC و Blazor امکان استفاده را می دهد. در این فصل من نحوه کارکرد ویژگی های پایه ASP.NET Core را توضیح می‌دهم، هدف از فایل‌های آن را در یک پروژه ASP.NET Core شرح می‌دهم و توضیح می‌دهم که چگونه پایپلاین درخواست ASP.NET Core برای پردازش درخواست‌های HTTP استفاده می‌شود و راه های مختلفی که می توان آن را سفارشی کرد را نشان می دهم.

نگران نباشید اگر همه چیز در این فصل بلافاصله برای اعمال در اپلیکیشن هایی که قصد ایجاد آن را دارید منطقی نیست یا منطقی به نظر می رسد. ویژگی‌هایی که در این فصل توضیح می‌دهم، زیربنای تمام کارهایی است که ASP.NET Core انجام می‌دهد و درک نحوه کار آنها به ایجاد زمینه‌ای برای درک ویژگی‌هایی که روزانه از آنها استفاده خواهید کرد کمک می کند، همچنین به شما دانش لازم برای تشخیص مشکلات را ارائه می‌دهد، وقتی رفتاری را که انتظارش را دارید را نمی بینید. در جدول ۱۲-۱ پلتفرم ASP.NET Core در این زمینه قرار می گیرد.

جدول ۱۲-۱ زمینه چینی برای پلتفرم ASP.NET Core

سئوال	پاسخ
پلتفرم چیست؟	پلت فرم ASP.NET Core بنیانی است که اپلیکیشن های وب بر روی آن ساخته می شوند و ویژگی هایی را برای پردازش درخواست های HTTP فراهم می کند.
برای چه کاری مفید است؟	پلت فرم ASP.NET Core به جزئیات سطح پایین اپلیکیشن های وب رسیدگی می کند تا توسعه دهندگان بتوانند بر روی ویژگی های لازم برای کاربر نهایی تمرکز کنند.
چگونه استفاده می شود؟	بلوک‌های ساختمانی کلیدی پلتفرم شامل سرویس‌ها و کامپوننت های میان‌افزار هستند که هر دو را می‌توان با استفاده از دستورات سطح بالا در فایل Program.cs ایجاد کرد.
مانع یا محدودیتی دارد؟	استفاده از فایل Program.cs می تواند گیج کننده باشد و باید به ترتیب دستوراتی که در آن وجود دارد دقت کرد.
جایگزین و بدیلی دارد؟	پلت فرم ASP.NET Core برای اپلیکیشن های ASP.NET Core نیاز است، اما می توانید انتخاب کنید که مستقیماً با پلتفرم کار نکنید و فقط به ویژگی های سطح بالاتر ASP.NET Core متکی باشید که در فصول بعدی بحث خواهد شد.

در جدول ۱۲-۲ خلاصه این فصل ارائه شده است.

جدول ۱۲-۲ خلاصه ی فصل

مشکل	راه حل	لیست کد
ایجاد یک کامپوننت میان افزار	برای افزودن یک تابع یا کلاس به پایپلاین درخواست، متد Use یا UseMiddleware را فراخوانی کنید.	۶-۸
تعدیل یک پاسخ	یک کامپوننت میان افزاری بنویسید که از مسیر پایپلاین برگشتی استفاده کند.	۹
جلوگیری از پردازش درخواست توسط سایر کامپوننت ها	پایپلاین درخواست را اتصال کوتاه کنید یا میان افزار ترمینال خلق کنید.	۱۰، ۱۲-۱۴
استفاده از مجموعه های مختلف میان افزار	یک انشعاب به پایپلاین خلق کنید.	۱۱
پیکربندی کامپوننت های میان افزار	از الگوی گزینه ها استفاده کنید.	۱۵-۱۸

آماده شدن برای این فصل

برای آماده شدن برای این فصل، قصد دارم با استفاده از تمپلیتی که کمینه تنظیمات ASP.NET Core را ارائه می دهد، یک پروژه جدید به نام Platform ایجاد کنم. خط فرمان PowerShell را از منوی استارت ویندوز باز کنید و فرامین ذکر شده در لیست ۱۲-۱ را اجرا کنید.

■ **نکته**: می‌توانید پروژه مثال این فصل (و سایر فصول کتاب) را از https://github.com/apress/pro-asp.net-core-6 دانلود کنید. اگر در اجرای مثال ها مشکل دارید، برای چگونگی دریافت کمک به فصل ۱ مراجعه کنید.

لیست ۱۲-۱ خلق پروژه

```
dotnet new globaljson --sdk-version 6.0.100 --output Platform
dotnet new web --no-https --output Platform --framework net6.0
dotnet new sln -o Platform
dotnet sln Platform add Platform
```

اگر از ویژوال استودیو استفاده می کنید، فایل Platform.sln را در پوشه Platform باز کنید. اگر از ویژوال استودیو کد استفاده می کنید، پوشه Platform را باز کنید. وقتی از شما خواسته شد که سرمایه های مورد نیاز برای ساخت و اشکال زدایی پروژه را اضافه کنید، روی دکمه بله کلیک کنید.

فایل launchSettings.json را از پوشه Properties باز کنید و پورت هایی که برای رسیدگی به درخواست های HTTP استفاده می شود را تغییر دهید، همانطور که در لیست ۱۲-۲ نشان داده شده است.

لیست ۱۲-۲ تنظیم پورت ها در فایل launchSettings.json در پوشه Properties

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 0
    }
  },
  "profiles": {
    "Platform": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

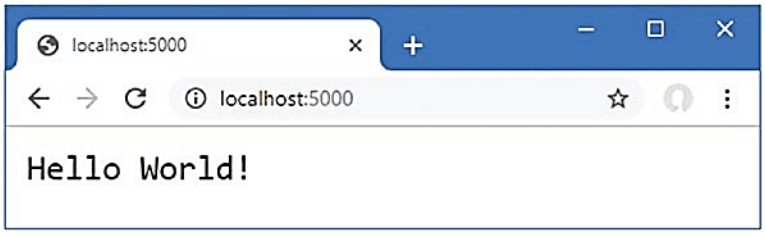
اجرای اپلیکیشن مثال

برای شروع برنامه، فرمان ذکر شده در لیست ۱۲-۳ را در پوشه Platform اجرا کنید.

لیست ۱۲-۳ شروع برنامه مثال

```
dotnet run
```

یک پنجره مرورگر جدید باز کنید و از آن برای درخواست http://localhost:5000 استفاده کنید. خروجی را در شکل ۱۲-۱ مشاهده خواهید کرد.



شکل ۱۲-۱. اجرای اپلیکیشن مثال

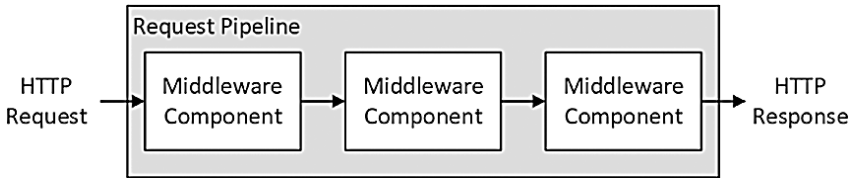
آشنایی با پلتفرم ASP.NET Core

برای درک ASP.NET Core، تمرکز بر روی ویژگی های کلیدی آن مفید است: پایپلاین درخواست، میان افزار و سرویس ها. درک اینکه چگونه این ویژگی ها با هم همسو و منطبق می شوند (حتی بدون پرداختن به جزئیات)، زمینه مفیدی را برای درک محتویات پروژه ASP.NET Core و کلیت پلتفرم ASP.NET Core فراهم می کند.

آشنایی با میان افزار و پایپلاین درخواست

هدف پلتفرم ASP.NET Core دریافت درخواست های HTTP و ارسال پاسخ ها به آنها است که ASP.NET Core آن را به کامپوننت ها یا همان کامپوننت های میان افزاری[1] خود واگذار می کند. کامپوننت های میان‌افزاری در یک زنجیره چیده شده اند که تحت عنوان پاپلاین یا پایپلاین درخواست[2] شناخته می شوند.

هنگامی که یک درخواست HTTP جدید می رسد، پلتفرم ASP.NET Core یک شی ایجاد می کند که آن درخواست را توصیف می کند و یک شی متناظر که پاسخی را که در ازای آن برگردانده می شود، را توصیف می‌کند. این اشیاء به اولین کامپوننت میان‌افزاری در زنجیره ارسال می‌شوند، که درخواست را بازرسی کرده و پاسخ را تعدیل می‌کند. سپس درخواست به کامپوننت میان‌افزاری بعدی در زنجیره ارسال می‌شود و هر کامپوننت درخواست را بررسی کرده و به پاسخ اضافه می‌کند. هنگامی که درخواست از میان پایپلاین عبور کرد، پلتفرم ASP.NET Core پاسخ را ارسال می کند، همانطور که در شکل ۱۲-۲ نشان داده شده است.



شکل ۱۲-۲ پایپلاین درخواست ASP.NET Core

برخی از کامپوننت ها روی ایجاد پاسخ برای درخواست‌ها تمرکز دارند، اما برخی دیگر به ارائه ویژگی‌های پشتیبانی، از قبیل قالب‌بندی انواع داده‌های خاص یا خواندن و نوشتن کوکی‌ها می پردازند. ASP.NET Core شامل کامپوننت های میان‌افزاری است که مشکلات رایج را حل می‌کند، همانطور که در فصل‌های ۱۵ و ۱۶ توضیح داده خواهد شد و من نحوه ایجاد کامپوننت میان‌افزار سفارشی را در ادامه این فصل نشان می‌دهم. اگر هیچ پاسخی توسط کامپوننت‌های میان‌افزاری ایجاد نشود، ASP.NET Core پاسخی با کد وضعیت HTTP 404 Not Found برمی‌گرداند.

آشنایی با سرویس ها

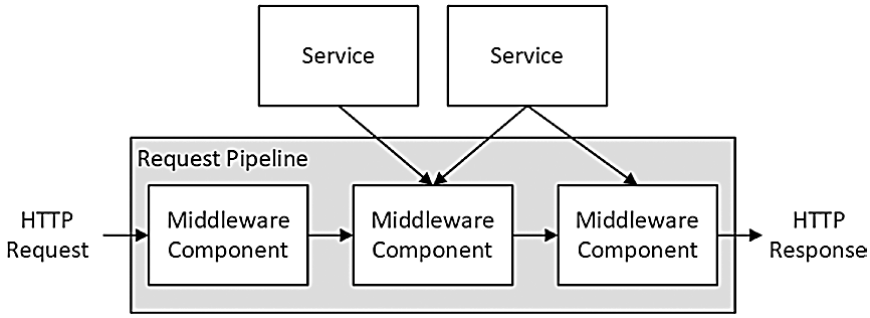
سرویس ها اشیایی هستند که ویژگی های یک اپلیکیشن وب را فراهم می کنند. هر کلاسی را می توان به عنوان یک سرویس استفاده کرد و هیچ محدودیتی برای ویژگی هایی که سرویس ها ارائه می دهند وجود ندارد. چیزی که سرویس ها را خاص می کند این است که توسط ASP.NET Core مدیریت می شوند و قابلیتی به نام تزریق وابستگی[3] دسترسی آسان به سرویس ها در هر نقطه ای از اپلیکیشن، از جمله کامپوننت های میان افزاری را ممکن می‌سازد.

---

[1] Middleware Components
[2] Request Pipeline
[3] Dependency Injection

تزریق وابستگی می تواند موضوعی دشوار برای درک باشد و من آن را مفصلا در فصل ۱۴ توضیح می دهم.

در حال حاضر، کافی است بدانید که اشیایی وجود دارند که توسط پلتفرم ASP.NET Core مدیریت می‌شوند و می‌توانند توسط کامپوننت های میان‌افزاری به اشتراک گذاشته شوند، یا برای هماهنگی بین کامپوننت ها یا برای جلوگیری از تکرار ویژگی‌های مشترک از قبیل ورود به سیستم یا بارگذاری داده‌های پیکربندی، همانطور که در شکل ۱۲-۳ نشان داده شده است.

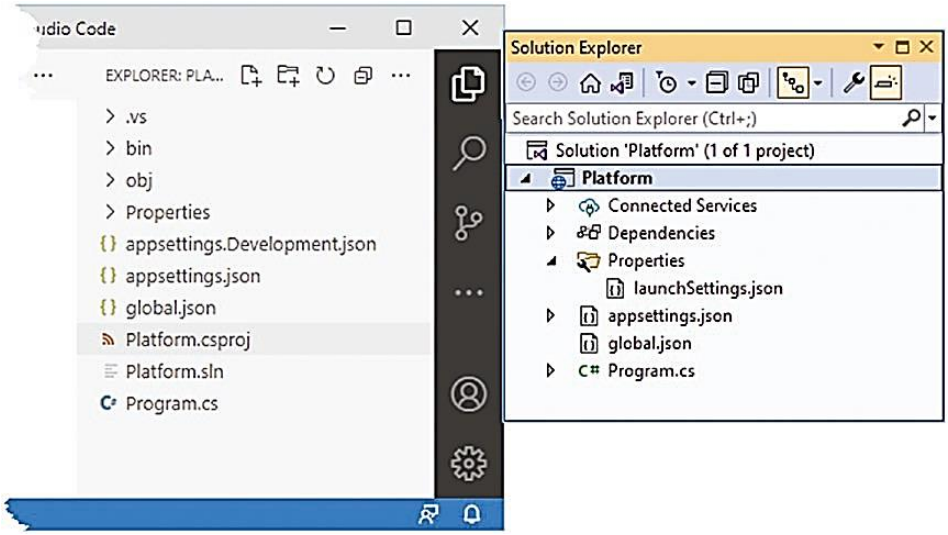


شکل ۱۲-۳ سرویس ها در پلتفرم ASP.NET Core

همانطور که شکل نشان می دهد، کامپوننت های میان افزاری فقط از سرویس هایی استفاده می کنند که برای انجام کار خود نیاز دارند. همانطور که در فصل های بعدی خواهید آموخت، ASP.NET Core برخی از سرویس های پایه را ارائه می دهد که می توانند با سرویس های اضافی که مختص یک اپلیکیشن هستند، تکمیل شوند.

آشنایی با پروژه ASP.NET Core

تمپلیت web پروژه ای را با کد و پیکربندی کافی تولید می کند که بتوان برای شروع زمان اجرا ASP.NET Core با برخی از سرویس های پایه و کامپوننت های میان افزاری، آن را استفاده کرد. شکل ۱۲-۴ فایل های اضافه شده به پروژه توسط تمپلیت را نشان می دهد.



شکل ۱۲-۴ فایل های موجود در پروژه مثال

ویژوال استودیو و ویژوال استودیو کد رویکردهای متفاوتی برای نمایش فایل ها و پوشه ها دارند. ویژوال استودیو مواردی را که معمولاً توسط توسعه‌دهندگان استفاده نمی‌شود را پنهان می‌کند و موارد مرتبط را در کنار هم قرار می‌دهد، در حالی که ویژوال استودیو کد همه چیز را نشان می‌دهد.

به همین دلیل است که دو نمای پروژه نشان داده شده در شکل متفاوت است: ویژوال استودیو پوشه bin و obj را مخفی کرده و فایل appsettings.Development.json را در فایل appsettings.json قرار داده است. دکمه های بالای پنجره Solution Explorer را می توان برای جلوگیری از تودرتو شدن و نمایش تمام فایل های پروژه استفاده کرد.

اگرچه فایل های کمی در پروژه وجود دارد، اما آنها زیربنای توسعه در ASP.NET Core هستند و در جدول ۱۲-۳ توضیح داده شده اند.

جدول ۱۲-۳ فایل ها و پوشه های پروژه مثال

نام	شرح
appsettings.json	این فایل همانطور که در فصل ۱۵ توضیح داده می شود، برای پیکربندی اپلیکیشن استفاده می شود.
appsettings.Development.json	از این فایل برای تعریف تنظیمات پیکربندی که مخصوص توسعه هستند، همانطور که در فصل ۱۵ بحث خواهد شد، استفاده می شود.
bin	این پوشه حاوی فایل های کامپایل شده اپلیکیشن است. ویژوال استودیو این پوشه را پنهان می کند.
global.json	این فایل برای انتخاب نسخه خاصی از .NET Core SDK استفاده می شود.
Properties/launchSettings.json	این فایل برای پیکربندی اپلیکیشن در هنگام استارت استفاده می شود.
Obj	این پوشه حاوی خروجی میانی از کامپایلر است. ویژوال استودیو این پوشه را پنهان می کند.
Platform.csproj	این فایل پروژه را برای ابزارهای .NET Core شرح می دهد، از جمله وابستگی‌های آن به پکیج و دستورالعمل‌های بیلد، همانطور که در بخش «آشنایی با فایل پروژه» توضیح داده شده است. ویژوال استودیو این فایل را پنهان می کند، اما با کلیک راست روی آیتم پروژه در Solution Explorer و انتخاب Edit Project File از منوی پاپ آپ می توان آن را ویرایش کرد.
Platform.sln	این فایل برای سازماندهی پروژه ها استفاده می شود. ویژوال استودیو این پوشه را پنهان می کند.
Program.cs	این فایل نقطه ورودی پلتفرم ASP.NET Core است و همانطور که در بخش «آشنایی با نقطه ورودی» توضیح داده شده است، برای پیکربندی پلتفرم استفاده می شود.

آشنایی با نقطه ورود

فایل Program.cs حاوی دستورات کدی است که هنگام راه اندازی اپلیکیشن اجرا می شوند و برای پیکربندی پلتفرم ASP.NET و هر یک از فریمورک هایی که پشتیبانی می کند، استفاده می شود. در اینجا محتوای فایل Program.cs برای پروژه مثال آمده است:

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.MapGet("/", () => "Hello World!");
app.Run();
```

این فایل فقط حاوی عبارات سطح بالا است. دستور اول WebApplication.CreateBuilder را فراخوانی می کند و نتیجه را به متغیری به نام builder اختصاص می دهد:

```
...
var builder = WebApplication.CreateBuilder(args);
...
```

این متد مسئول تنظیم ویژگی های اساسی پلتفرم ASP.NET Core است، از جمله خلق سرویس های مسئول داده های پیکربندی و ثبت گزارش، که هر دو در فصل ۱۵ توضیح داده می شوند. این متد همچنین سرور HTTP به نام Kestrel را برپا می کند که برای دریافت درخواست های HTTP استفاده می شود.

نتیجه متد CreateBuilder یک شی WebApplicationBuilder است که برای رجیستر کردن سرویس های اضافه استفاده می شود، اگرچه در حال حاضر هیچ یک تعریف نشده است. کلاس WebApplicationBuilder یک متد Build را تعریف می کند که برای نهایی کردن تنظیمات اولیه استفاده می شود:

```
...
var app = builder.Build();
...
```

نتیجه متد Build یک شی Web Application است که برای راه‌اندازی کامپوننت های میان‌افزاری استفاده می‌شود. این تمپلیت با استفاده از متد افزونه MapGet یک کامپوننت میان‌افزاری را نصب کرده است:

```
...
app.MapGet("/", () => "Hello World!");
...
```

MapGet یک متد افزونه برای اینترفیس یا واسط IEndpointRouteBuilder است که توسط کلاس WebApplication پیاده سازی می شود و تابعی را نصب می کند که درخواست های HTTP را با یک مسیر URL مشخص مدیریت می کند. در این نمونه، تابع به درخواست‌های مسیر پیش‌فرض URL که با / نشان داده می‌شود، پاسخ می‌دهد و تابع با برگرداندن یک پاسخ رشته‌ای ساده به تمام درخواست‌ها پاسخ می‌دهد، که خروجی نشان‌داده‌شده در شکل ۱۲-۱ به این صورت است.

اکثر پروژه‌ها به مجموعه پیچیده‌تری از پاسخ‌ها نیاز دارند و مایکروسافت میان‌افزار دیگری را به عنوان بخشی از ASP.NET Core ارائه می‌کند که با رایج‌ترین ویژگی‌های مورد نیاز برنامه‌های وب سروکار دارد، که در فصل‌های ۱۵ و ۱۶ توضیح می‌دهم. همچنین می‌توانید میان‌افزار خود را ایجاد کنید. همانطور که در بخش «ایجاد میان‌افزار سفارشی» توضیح داده می شود، زمانی که ویژگی‌های داخلی مطابق با نیازهای شما نیستند.

دستور نهایی در فایل Program.cs متد Run تعریف شده توسط کلاس WebApplication را فراخوانی می کند که شروع به گوش دادن به درخواست های HTTP می کند.

گرچه تابع استفاده شده با متد MapGet یک رشته را برمی گرداند، ASP.NET Core به اندازه کافی باهوش است تا یک پاسخ HTTP معتبر ایجاد کند که توسط مرورگرها قابل درک باشد. در حالی که ASP.NET Core هنوز در حال اجرا است، یک خط فرمان جدید PowerShell را باز کنید و فرمان ذکر شده در لیست ۱۲-۴ را اجرا کنید تا یک درخواست HTTP به سرور ASP.NET Core ارسال شود.

لیست ۱۲-۴ ارسال درخواست HTTP

```
(Invoke-WebRequest http://localhost:5000).RawContent
```

خروجی این فرمان نشان می دهد که پاسخ ارسال شده توسط ASP.NET Core حاوی یک کد وضعیت HTTP و مجموعه ای از هدرهای پایه است، همانند این:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: text/plain; charset=utf-8 Date: Thu, 11 Nov 2021 18:08:35 GMT
Server: Kestrel Hello World!
```

آشنایی با فایل پروژه

فایل Platform.csproj که به فایل پروژه معروف است حاوی اطلاعاتی است که NET Core. برای ساخت پروژه و ردگیری وابستگی ها استفاده می کند. در اینجا محتوایی آمده است که توسط تمپلیت Empty هنگام ایجاد پروژه به فایل اضافه می شود:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

</Project>
```

فایل csproj هنگام استفاده از ویژوال استودیو مخفی می شود. می توانید با کلیک راست بر روی آیتم پروژه Platform در Solution Explorer و انتخاب Edit Project File از منوی پاپ آپ آن را ویرایش کنید.

فایل پروژه حاوی عناصر XML است که پروژه را به MSBuild، موتور ساخت مایکروسافت، معرفی می کند. MSBuild را می توان برای ایجاد فرآیندهای ساخت پیچیده استفاده کرد و به طور مفصل در https://docs.microsoft.com/en-gb/visualstudio/msbuild/msbuild توضیح داده شده است.

در اکثر پروژه ها نیازی به ویرایش مستقیم فایل پروژه نیست. رایج ترین تغییر در فایل، افزودن وابستگی ها به پکیج های دات نت دیگر است، اما این پکیج ها معمولاً با استفاده از ابزارهای خط فرمان یا رابط ارائه شده توسط ویژوال استودیو اضافه می شوند.

برای افزودن یک پکیج به پروژه با استفاده از خط فرمان، یک خط فرمان جدید PowerShell را باز کنید، به پوشه پروژه Platform (پوشه ای که حاوی فایل csproj است) بروید و فرمان ذک شده در لیست ۱۲-۵ را اجرا کنید.

لیست ۱۲-۵ افزودن پکیج به پروژه

```
dotnet add package Swashbuckle.AspNetCore --version 6.2.2
```

این فرمان پکیج Swashbuckle.AspNetCore را به پروژه اضافه می کند. این پکیج را در فصل ۲۰ مشاهده خواهید کرد، اما در حال حاضر، این اثر دستور dotnet add package است که مهم است.

وابستگی جدید در فایل Platform.csproj نشان داده خواهد شد:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.2.2" />
  </ItemGroup>

</Project>
```

خلق میان افزار سفارشی

همانطور که گفته شد، مایکروسافت کامپوننت های میان‌افزار مختلفی را برای ASP.NET Core ارائه می‌کند که ویژگی‌هایی را که معمولاً برای اپلیکیشن های وب نیاز است، مدیریت می‌کند. همچنین می‌توانید میان‌افزار خود را ایجاد کنید، که راهی مفید برای درک نحوه عملکرد ASP.NET Core است، حتی اگر فقط از کامپوننت های استاندارد در پروژه‌های خود استفاده می کنید. همانطور که در لیست ۱۲-۶ نشان داده شده است، متد کلیدی برای ایجاد میان افزار متد Use است.

لیست ۱۲-۶ خلق میان افزار سفارشی در فایل Program.cs از پوشه Platform

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    if (context.Request.Method == HttpMethods.Get
    && context.Request.Query["custom"] == "true")
    {
        context.Response.ContentType = "text/plain";
        await context.Response.WriteAsync("Custom Middleware \n");
    }
    await next();
});

app.MapGet("/", () => "Hello World!");
app.Run();
```

جدول ۱۲-۴ اعضای مفید HttpContext

نام	شرح
Connection	این ویژگی یک شی ConnectionInfo را برمی‌گرداند که اطلاعاتی درباره اتصال شبکه زیربنای درخواست HTTP، از جمله جزئیات آدرس‌ها و پورت‌های IP محلی و راه دور ارائه می‌دهد.
Request	این ویژگی یک شی HttpRequest را برمی گرداند که درخواست HTTP در حال پردازش را توصیف می کند.
RequestServices	این ویژگی دسترسی به سرویس های موجود برای درخواست را، همانطور که در فصل ۱۴ توضیح داده می شود، فراهم می کند.
Response	این ویژگی یک شی HttpResponse برمی‌گرداند که برای ایجاد پاسخ به درخواست HTTP استفاده می‌شود.
Session	این ویژگی داده های جلسه مرتبط با درخواست را برمی گرداند. ویژگی داده های جلسه در فصل ۱۶ توضیح داده خواهد شد.
User	این ویژگی جزئیات کاربر مرتبط با درخواست را برمی‌گرداند که در فصول ۳۷ و ۳۸ توضیح داده می‌شود.
Features	این ویژگی دسترسی به ویژگی های درخواست را فراهم می کند که امکان دسترسی به جنبه های سطح پایین رسیدگی به درخواست را فراهم می کند. برای مثالی از استفاده از ویژگی درخواست، به فصل ۱۶ مراجعه کنید.

متد Use یک کامپوننت میان‌افزاری را رجیستر می‌کند که معمولاً به عنوان یک تابع لامبدا بیان می‌شود که هر درخواست را هنگام عبور از پایپلاین دریافت می‌کند (روش دیگری برای کلاس‌ها استفاده می‌شود، همانطور که در بخش بعدی توضیح داده خواهد شد).

آرگومان های تابع لامبدا یک شی HttpContext و تابعی است تحریک می شود تا به ASP.NET Core بگوید که درخواست را به کامپوننت میان افزاری بعدی در پایپلاین ارجاع دهد.

شی HttpContext درخواست HTTP و پاسخ HTTP را توصیف می کند و زمینه اضافی، از جمله جزئیات کاربر مرتبط با درخواست را ارائه می دهد. جدول ۱۲-۴ مفیدترین اعضای ارائه شده توسط کلاس HttpContext را توضیح می دهد که در فضای نام Microsoft.AspNetCore.Http تعریف شده است.

پلتفرم ASP.NET Core مسئول پردازش درخواست HTTP برای خلق شی HttpRequest است، به این معنی که میان افزار و نقاط پایانی نیازی به نگرانی در مورد داده های خام درخواست ندارند. جدول ۱۲-۵ مفیدترین اعضای کلاس HttpRequest را نشان داده است.

جدول ۱۲-۵ اعضای مفید HttpRequest

نام	شرح
Body	این ویژگی جریانی را برمی گرداند که می توان از آن برای خواندن تنه درخواست استفاده کرد.
ContentLength	این ویژگی مقدار هدر Content-Length را برمی گرداند.
ContentType	این ویژگی مقدار هدر Content-Type را برمی گرداند.
Cookies	این ویژگی کوکی های درخواست را برمی گرداند.
Form	این ویژگی بازنمایی تنه درخواست به شکل یک فرم را برمی گرداند.
Headers	این ویژگی هدرهای درخواست را برمی گرداند.
IsHttps	اگر درخواست با استفاده از HTTPS انجام شده باشد، این ویژگی true است.
Method	این ویژگی فعل HTTP (متد HTTP نیز گفته می شود) استفاده شده برای درخواست را برمی گرداند.
Path	این ویژگی قسمت مسیر URL درخواست را برمی گرداند.
Query	این ویژگی بخش رشته ای کوئری از URL درخواستی را به صورت جفت کلید–مقدار برمی گرداند.

شی HttpResponse پاسخ HTTP را توصیف می کند که زمانی که درخواست از میان پایپلاین عبور کرد، به کلاینت ارسال می شود. جدول ۱۲-۶ مفیدترین اعضای کلاس HttpResponse را توضیح می دهد. پلتفرم ASP.NET Core رسیدگی به پاسخ ها را تا حد امکان آسان می کند، هدرها را به صورت خودکار تنظیم می کند و ارسال محتوا به کلاینت را تسهیل می کند.

جدول ۱۲-۶ اعضای مفید HttpResponse

نام	شرح
ContentLength	این ویژگی مقدار هدر Content-Length را تعیین می کند.
ContentType	این ویژگی مقدار هدر Content-Type را تعیین می کند.
Cookies	این ویژگی اجازه می دهد تا کوکی ها با درخواست همراه شوند.
HasStarted	اگر ASP.NET Core شروع به ارسال هدرهای پاسخ به کلاینت کرده باشد، این ویژگی true برمی گرداند و پس از آن امکان تغییر در کد وضعیت یا هدرها وجود ندارد.
Headers	این ویژگی امکان ست کردن هدرهای پاسخ را فراهم می سازد.
StatusCode	این ویژگی کد وضعیت را برای پاسخ تنظیم می کند.
WriteAsync(data)	این متد آسنکرون یک رشته داده در تنه پاسخ می نویسد.
Redirect(url)	این متد یک پاسخ تغییر مسیر ارسال می کند.

هنگام خلق میان‌افزار سفارشی، اشیاء HttpContext، HttpRequest و HttpResponse مستقیماً استفاده می‌شوند، اما همانطور که در فصل‌های بعدی خواهید آموخت، معمولاً هنگام استفاده از ویژگی های سطح بالاتر ASP.NET Core از قبیل فریمورک MVC و صفحات ریزور، نیازی به این کار نیست.

تابع میان‌افزاری که در لیست ۱۲-۶ تعریف کرده ام از شی HttpRequest برای چک متد HTTP و رشته query برای شناسایی درخواست‌های GET استفاده می‌کند که دارای یک پارامتر custom در رشته کوئری هستند که مقدار آن true است، مانند این:

```
...
if (context.Request.Method == HttpMethods.Get
    && context.Request.Query["custom"] == "true") {
...
```

کلاس HttpMethods رشته های استاتیک را برای هر متد HTTP تعریف می کند. برای درخواست‌های GET با رشته کوئری مورد انتظار، تابع میان‌افزار از ویژگی ContentType برای تنظیم هدر Content-Type استفاده می‌کند و از روش WriteAsync برای اضافه کردن یک رشته به تنه پاسخ استفاده می‌کند.

```
...
context.Response.ContentType = "text/plain";
await context.Response.WriteAsync("Custom Middleware \n");
...
```

تنظیم هدر Content-Type مهم است زیرا از تلاش بعدی کامپوننت میان‌افزاری برای تنظیم کد وضعیت پاسخ و هدرها جلوگیری می‌کند. ASP.NET Core همیشه سعی می‌کند مطمئن شود که یک پاسخ HTTP معتبر ارسال شده است و این می‌تواند موجب شود که هدرهای پاسخ یا کد وضعیت بعدا ست شود، پس از اینکه کامپوننت های قبلی محتوایی را برای تنه پاسخ نوشته باشند، این موجب صدور یک استثنا می شود (زیرا هدرها باید قبل از شروع تنه پاسخ برای کلاینت ارسال شوند).

■ **توجه:** در این قسمت از کتاب، تمام مثال‌ها نتایج رشته‌ای ساده ای را به مرورگر ارسال می‌کنند. در بخش ۳، من به شما نشان می دهم که چگونه خدمات وب ایجاد کنید که داده های JSON را برگرداند و راه های مختلفی را معرفی می کنم که ASP.NET Core می تواند نتایج html تولید کند.

آرگومان دوم به میان‌افزار تابعی است که به طور متعارف next نام دارد و به ASP.NET Core می‌گوید درخواست را به کامپوننت بعدی در پایپلاین درخواست ارسال کند.

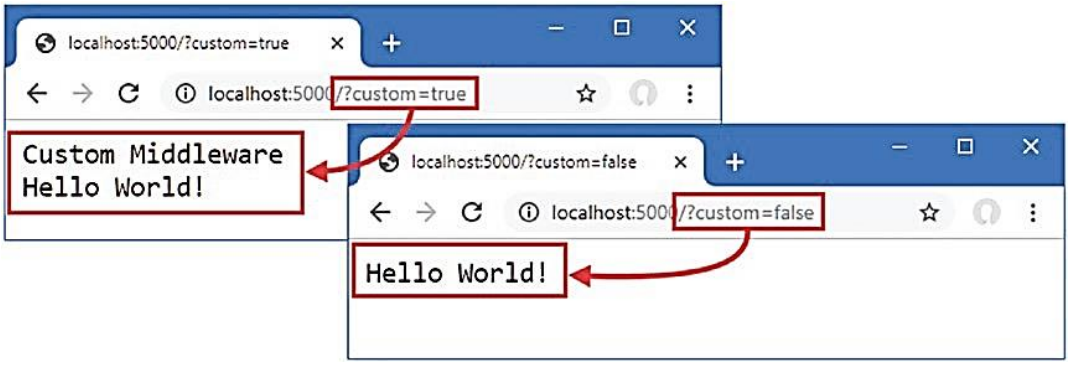
```
...
if (context.Request.Method == HttpMethods.Get
   && context.Request.Query["custom"] == "true")
{
    context.Response.ContentType = "text/plain";
    await context.Response.WriteAsync("Custom Middleware \n");
}
await next();
...
```

هنگام تحریک کامپوننت میان‌افزاری بعدی، نیازی به آرگومان نیست، زیرا ASP.NET Core ترتیب فراهم شدن شی HttpContext و تابع next خودی برای کامپوننت را می دهد تا بتواند درخواست را پردازش کند. تابع next آسنکرون است، به همین دلیل از کلمه کلیدی await استفاده می شود و تابع lambda با کلمه کلیدی async تعریف می شود.

■ **نکته:** ممکن است با میان افزاری مواجه شوید که به جای تابع next() تابع next.Invoke() را فراخوانی کند. اینها معادل هستند، و next() برای راحتی توسط کامپایلر برای تولید کد مختصر تولید می شود.

ASP.NET Core را با استفاده از فرمان dotnet run اجرا کنید و آدرس http://localhost:5000/?custom=true را در مرورگر باز کنید. خواهید دید که تابع میان‌افزار جدید، همانطور که در شکل ۱۲-۵ نشان داده شده است، قبل از ارسال درخواست به کامپوننت میان‌افزاری بعدی، پیام خود را به پاسخ می‌نویسد.

رشته کوئری را حذف کنید یا true را به false تغییر دهید و جزء میان افزار بدون افزودن چیزی به پاسخ، درخواست را ارجاع می دهد.



شکل ۱۲-۵ ایجاد میان افزار سفارشی

تعریف میان افزار با استفاده از کلاس

تعریف میان‌افزار با استفاده از توابع لامبدا راحت است، اما می‌تواند به یک سری دستورات طولانی و پیچیده در فایل Program.cs منجر شود و استفاده مجدد از میان‌افزار در پروژه‌های مختلف را دشوار می‌کند.

میان افزار را می توان با استفاده از کلاس ها نیز تعریف کرد. یک فایل کلاس به نام Middleware.cs را به پوشه Platform اضافه کنید و کد نشان داده شده در لیست ۱۲-۷ را به آن اضافه کنید.

لیست ۱۲-۷ محتوای فایل Middleware.cs از پوشه Platform

```
namespace Platform
{

    public class QueryStringMiddleWare
    {
        private RequestDelegate next;

        public QueryStringMiddleWare(RequestDelegate nextDelegate)
        {
            next = nextDelegate;
        }

        public async Task Invoke(HttpContext context)
```

```
        {
            if (context.Request.Method == HttpMethods.Get
        && context.Request.Query["custom"] == "true")
            {
                if (!context.Response.HasStarted)
                {
                    context.Response.ContentType = "text/plain";
                }
                await context.Response.WriteAsync("Class-based Middleware \n");
            }
            await next(context);
        }
    }
}
```

کلاس های میان افزاری یک RequestDelegate را به عنوان پارامتر سازنده دریافت می کنند که برای ارسال درخواست به کامپونت بعدی پایپلاین استفاده می شود. متد Invoke توسط ASP.NET Core زمانی فراخوانی می شود که درخواستی دریافت می شود و یک شی HttpContext را دریافت می کند که دسترسی به درخواست و پاسخ را با استفاده از همان کلاس هایی که میان افزار تابع لامبدا دریافت می کرد، دریافت می کند. RequestDelegate یک Task را برمی گرداند که به آن اجازه می دهد به صورت آسنکرون کار کند.

یک تفاوت مهم میان افزارهای مبتنی بر کلاس در این است که شی HttpContext باید به عنوان آرگومان استفاده شود، زمانی که RequestDelegate برای فروارد درخواست تحریک می شود، همانند این:

```
...
await next(context);
...
```

کامپوننت های میان‌افزار مبتنی بر کلاس با متد UseMiddleware به پایپلاین اضافه می‌شوند، که میان‌افزار را به عنوان آرگومان نوع می پذیرد، همانطور که در لیست ۱۲-۸ نشان داده شده است.

لیست ۱۲-۸ افزودن یک کامپوننت میان‌افزاری مبتنی بر کلاس در فایل Program.cs از پوشه Platform

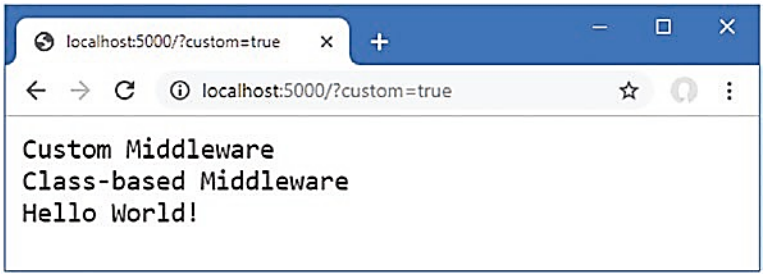
```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.Use(async (context, next) =>
{
    if (context.Request.Method == HttpMethods.Get
        && context.Request.Query["custom"] == "true")
    {
        context.Response.ContentType = "text/plain";
        await context.Response.WriteAsync("Custom Middleware \n");
    }
    await next();
});
app.UseMiddleware<Platform.QueryStringMiddleWare>();

app.MapGet("/", () => "Hello World!");
app.Run();
```

هنگامی که ASP.NET Core استارت می شود، کلاس QueryStringMiddleware نمونه برداری می شود و متد Invoke آن برای پردازش درخواست ها به محض دریافت آنها فراخوانی می شود.

■ **احتیاط**: یک شی میان‌افزاری منفرد برای رسیدگی به همه درخواست‌ها استفاده می‌شود، به این معنی که کد موجود در متد Invoke باید از نظر صف امن باشد.

از دستور dotnet run برای راه اندازی ASP.NET Core استفاده کنید و از یک مرورگر برای درخواست http://localhost:5000/?custom=true استفاده کنید. همانطور که در شکل ۱۲-۶ نشان داده شده است، خروجی هر دو جزء میان افزار را مشاهده خواهید کرد.



شکل ۱۲-۶ استفاده از یک کامپوننت میان افزاری مبتنی بر کلاس

آشنایی با مسیر پایپلاین بازگشت

کامپوننت های میان‌افزار می‌توانند شی HTTPResponse را پس از فراخوانی تابع بعدی تغییر دهند، همانطور که توسط میان‌افزار جدید لیست ۱۲-۹ نشان داده شده است.

لیست ۱۲-۹ افزودن میان افزار جدید به فایل Program.cs از پوشه Platform

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

app.Use(async (context, next) =>
{
    await next();
    await context.Response
    .WriteAsync($"\nStatus Code: {context.Response.StatusCode}");
});

app.Use(async (context, next) =>
{
    if (context.Request.Method == HttpMethods.Get
        && context.Request.Query["custom"] == "true")
    {
        context.Response.ContentType = "text/plain";

        await context.Response.WriteAsync("Custom Middleware \n");
    }
```