

آموزش ASP.NET Core



روزبه امیرعصامی



مؤسسه فرهنگی هنری
دیباکران تهران

به نام خدا

آموزش

ASP.NET Core

مؤلف:

روزبه امیر عصامی



هرگونه چاپ و تکثیر از محتویات این کتاب بدون اجازه کتبی ناشر ممنوع است. متخلفان به موجب قانون حمایت حقوق مؤلفان، مصنفان و هنرمندان تحت پیگرد قانونی قرار می‌گیرند.

◀ عنوان کتاب: آموزش ASP.NET Core

◀ مولف: روزبه امیر عصامی

◀ ناشر: مؤسسه فرهنگی هنری دیباگران تهران

◀ ویراستار: مهدیه مخبری

◀ صفحه آرای: نازنین نصیری

◀ طراح جلد: داربوش فرسای

◀ نوبت چاپ: اول

◀ تاریخ نشر: ۱۴۰۳

◀ چاپ و صحافی: درج عقیق

◀ تیراژ: ۱۰۰ جلد

◀ شابک: ۹۷۸-۶۲۲-۲۱۸-۸۶۱-۰

◀ نشانی واحد فروش: تهران، خیابان انقلاب، خیابان دانشگاه

◀ تقاطع شهدای ژاندارمری - پلاک ۱۵۸ ساختمان دانشگاه -

◀ طبقه دوم - واحد ۴ تلفن ها: ۶۶۹۶۵۷۴۹-۶۶۹۶۵۱۱۱-۲۲۰۸۵۱۱۱

◀ فروشگاه‌های اینترنتی دیباگران تهران:

WWW.MFTBOOK.IR

www.dibagarantehran.com

سرشناسه: امیر عصامی، روزبه، ۱۳۶۱ -
عنوان و نام پدیدآور: آموزش ASP.NET Core / مولف:
روزبه امیر عصامی؛
ویراستار: مهدیه مخبری.
مشخصات نشر: تهران: دیباگران تهران: ۱۴۰۳
مشخصات ظاهری: ۱۹۶ ص: مصور،
شابک: ۰-۸۶۱-۲۱۸-۶۲۲-۹۷۸
وضعیت فهرست نویسی: فیبا
موضوع: مایکروسافت دات نت فریم ورک
موضوع: Microsoft .NET Framework
موضوع: وبگاه ها- برنامه های تالیفی
موضوع: Web sites-authoring programs
موضوع: علوم کامپیوتر computer science
موضوع: نرم افزار- مهندسی software engineering
رده بندی کنگره: ۷۶/۷۶ QA
رده بندی دیویی: ۰۰۵/۲۷۶
شماره کتابشناسی ملی: ۹۶۹۷۴۸۳

نشانی تلگرام: @mftbook نشانی اینستاگرام دیبا dibagaran_publishing

هر کتاب دیباگران، یک فرصت جدید علمی و شغلی.

هر گوشی همراه، یک فروشگاه کتاب دیباگران تهران.

از طریق سایتهای دیباگران، در هر جای ایران به کتابهای ما دسترسی دارید.

فهرست مطالب

پیش‌نیازها ۹

ابزارهای مورد نیاز ۹

فصل اول

چرا ASP.NET Core؟ ۱۰

ASP.NET Core چیست؟ ۱۰

چرا ASP.NET Core را انتخاب کنیم؟ ۱۱

ایجاد اولین برنامه ASP.NET Core ۱۱

اجرای وب اپلیکیشن ۱۴

درک ساختار پروژه ۱۴

کلاس Program ۱۵

کلاس Startup ۱۶

متد ConfigureServices ۱۶

متد Configure ۱۷

فصل دوم

Middleware و مدیریت خطاها ۱۹

Middleware چیست؟ ۱۹

نحوه تعریف Middleware در Pipeline ۲۰

Exception Handling چیست؟ ۲۲

مدیریت خطا با استفاده از DeveloperExceptionHandlerMiddleware و ExceptionHandlerMiddleware ۲۳

Hosting Environment چیست؟ ۲۵

تنظیم محیط میزبانی ۲۸

ایجاد صفحه سفارشی ۳۰

مدیریت خطاها با استفاده از StatusCodePagesMiddleware ۳۱

فصل سوم

۳۶ MVC Design Pattern

۳۶ MVC چیست؟
۳۶ MVC اجزای
۳۷ سیستم MVC چطور کار می کند؟
۳۷ مزایای MVC Design Pattern
۳۸ پیاده سازی MVC در وب اپلیکیشن ASP.NET Core
۳۹ Model چیست؟
۳۹ Validation Attribute ها
۴۱ ایجاد Model
۴۴ Controller و اکشن متد چیست؟

فصل چهارم

۴۸ Routing سیستم

۴۸ Routing چیست؟
۴۸ مزایای سیستم Routing
۴۹ سیستم Routing چگونه کار می کند؟
۴۹ قسمت های یک الگوی مسیر
۵۰ روش های Mapping
۵۰ Conventional Routing
۵۱ Attribute Routing
۵۴ برنامه MVC با چندین مسیر
۵۵ Constraint بر روی مسیرها
۵۶ چطور Constraint ها را اعمال نماییم؟

فصل پنجم

۶۰ Razor view با استفاده از HTML

۶۰ View چیست؟
۶۱ Razor چیست؟
۶۱ چرا یادگیری Razor مهم است؟
۶۱ نحوه استفاده از Razor

۶۶	روش‌های انتقال داده به View
۶۶	انتقال داده با View Model
۶۶	ایجاد View Model
۷۰	انتقال داده با استفاده از ViewData
۷۳	انتقال داده با استفاده از ViewBag
۷۴	نوشتن عبارات با سینتکس Razor
۷۴	متغیرها
۷۴	عبارات شرطی
۷۴	حلقه‌ها
۷۶	کامنت
۷۶	Layout چیست؟
۷۷	مزایای Layout
۷۷	چطور از Layout استفاده کنیم؟
۷۹	Section چیست؟
۸۲	Partial view چیست؟
۸۳	ایجاد یک Partial View
۸۳	استفاده از Partial View
۸۴	استفاده از Partial View های Strongly Type
۸۶	ViewStart چیست؟
۸۷	ایجاد فایل ViewStart
۸۸	ViewImports چیست؟
۸۸	ایجاد فایل ViewImports

فصل ششم

۹۱	Tag Helper ها چیست؟
۹۱	Tag Helper چیست؟
۹۲	فعال کردن Tag Helper در اپلیکیشن
۹۳	استفاده از Tag Helper ها
۹۳	Environment Tag Helper
۹۴	Link Tag Helper و Script Tag Helper
۹۵	Form Tag Helper
۹۵	Label Tag Helper
۹۶	ایجاد یک Tag Helper سفارشی

فصل هفتم

تزریق وابستگی چیست؟ ۱۰۲

- ۱۰۲..... (Dependency Injection) DI چیست؟
- ۱۰۳..... تزریق وابستگی در ASP.NET Core
- ۱۰۴..... استفاده از تزریق وابستگی
- ۱۰۴..... مراحل ایجاد کدهای loosely coupled
- ۱۰۸..... طول عمر سرویس چیست؟
- ۱۱۴..... پیاده‌سازی‌های مختلف از یک سرویس

فصل هشتم

ایجاد WebAPI در ASP.NET Core ۱۲۱

- ۱۲۱..... Web API چیست و چه زمانی باید از آن استفاده کنید؟
- ۱۲۲..... REST چیست و HTTP چگونه کار می‌کند؟
- ۱۲۲..... Controller و اکشن متدها
- ۱۲۳..... ایجاد اولین اپلیکیشن Web API
- ۱۲۷..... افزودن Domain Model ها
- ۱۳۲..... Dapper و Entity Framework Core
- ۱۳۷..... رجیستر DbContext
- ۱۴۰..... Data Seeding چیست؟
- ۱۴۲..... ایجاد و آپدیت دیتابیس با Migration
- ۱۴۵..... پیاده‌سازی Command
- ۱۵۰..... افزودن UpdateDepartmentCommand
- ۱۵۱..... افزودن DeleteDepartmentCommand
- ۱۵۳..... پیاده‌سازی Query
- ۱۵۹..... افزودن Controller ها
- ۱۶۰..... افزودن اکشن متدها
- ۱۶۴..... ایجاد اکشن متد CreateDepartment
- ۱۶۴..... تست API ها با استفاده از PowerShell
- ۱۶۹..... افزودن اکشن متد UpdateDepartment
- ۱۷۱..... افزودن اکشن متد DeleteDepartment

فصل نهم

۱۷۳ Authorization و Authentication چیست؟

۱۷۳..... مقدمه‌ای در مورد Authorization و Authentication

۱۷۴..... Authentication در ASP.NET Core

۱۷۴..... پیاده‌سازی Authentication در ASP.NET Core

۱۷۶..... پی‌یکربندی ASP.NET Core Identity

۱۸۳..... استفاده از Authorization در اپلیکیشن

۱۸۷..... افزودن فرم الگین

۱۹۲..... Seeding داده‌های کاربر

خط‌مشی انتشارات مؤسسه فرهنگی هنری دیباگران تهران در عرصه کتاب‌هایی با کیفیت عالی است که بتواند
خواسته‌های به روز جامعه فرهنگی و علمی کشور را تا حد امکان پوشش دهد.
هر کتاب دیباگران تهران، یک فرصت جدید شغلی و علمی

حمد و سپاس ایزد منان را که با الطاف بی‌کران خود این توفیق را به ما ارزانی داشت تا بتوانیم در راه ارتقای دانش عمومی و فرهنگی این مرز و بوم در زمینه چاپ و نشر کتب علمی و آموزشی گام‌هایی هرچند کوچک برداشته و در انجام رسالتی که بر عهده داریم، مؤثر واقع شویم.

گسترده‌گی علوم و سرعت توسعه روزافزون آن، شرایطی را به وجود آورده که هر روز شاهد تحولات اساسی چشمگیری در سطح جهان هستیم. این گسترش و توسعه، نیاز به منابع مختلف از جمله کتاب را به عنوان قدیمی‌ترین و راحت‌ترین راه دستیابی به اطلاعات و اطلاع‌رسانی، بیش از پیش برجسته نموده است.

در این راستا، واحد انتشارات مؤسسه فرهنگی هنری دیباگران تهران با همکاری اساتید، مؤلفان، مترجمان، متخصصان، پژوهشگران و محققان در زمینه‌های گوناگون و مورد نیاز جامعه تلاش نموده برای رفع کمبودها و نیازهای موجود، منابعی پُر بار، معتبر و با کیفیت مناسب در اختیار علاقمندان قرار دهد.

کتابی که در دست دارید تألیف "جناب آقای روزبه امیر عصامی" است که با تلاش همکاران ما در نشر دیباگران تهران منتشر گشته و شایسته است از یکایک این گرامیان تشکر و قدردانی کنیم.

با نظرات خود مشوق و راهنمای ما باشید

با ارائه نظرات و پیشنهادات و خواسته‌های خود، به ما کمک کنید تا بهتر و دقیق‌تر در جهت رفع نیازهای علمی و آموزشی کشورمان قدم برداریم. برای رساندن پیام‌هایتان به ما از رسانه‌های دیباگران تهران شامل سایتهای فروشگاهی و صفحه اینستاگرام و شماره‌های تماس که در صفحه شناسنامه کتاب آمده استفاده نمایید.

مدیر انتشارات

مؤسسه فرهنگی هنری دیباگران تهران
dibagaran@mftplus.com

پیش‌نیازها

برای مطالعه این کتاب شما باید پیش‌نیازهای زیر را داشته باشید:

- (۱) دانستن اینکه وب اپلیکیشن چیست؟
- (۲) داشتن یک تجربه اولیه از ایجاد یک وب اپلیکیشن با سی شارپ.

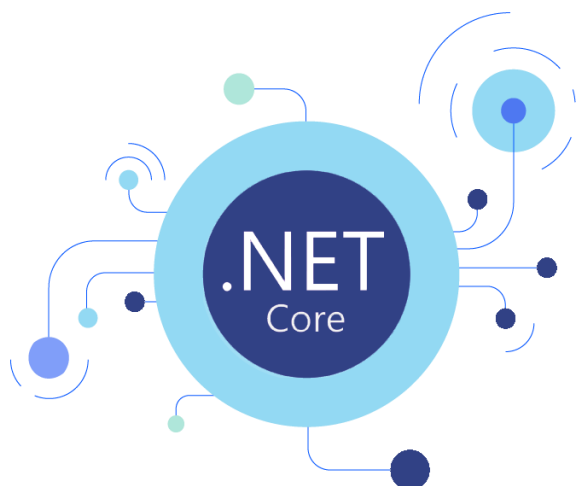
ابزارهای مورد نیاز

تنها ابزاری که برای یادگیری به آن نیازی دارید Visual Studio 2019 است.

نکته

- هنگام نصب Visual Studio، اطمینان حاصل کنید که کامپوننت‌ها، ASP.NET و NET Core را انتخاب کرده‌اید.

فصل اول



چرا ASP.NET Core؟

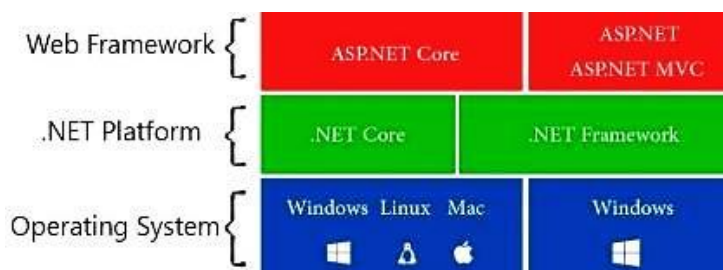
- ◀ ASP.NET Core چیست؟
- ◀ چرا ASP.NET Core را انتخاب کنیم؟
- ◀ ایجاد اولین اپلیکیشن ASP.NET Core
- ◀ اجرای وب اپلیکیشن
- ◀ درک ساختار پروژه



ASP.NET Core چیست؟

ASP.NET Core با اصول طراحی نرم افزار مدرن، بر روی پلتفرم جدید .NET Core ایجاد شده است که در آن فقط کامپوننت هایی که نیاز دارید را استفاده می کنید و می توانید برنامه خود را تا جایی که امکان دارد جمع و جور و با عملکرد بالا بسازید.

ASP.NET Core این امکان را به شما می دهد که وب اپلیکیشن های خود را در ویندوز لینوکس و Mac ایجاد و اجرا نمایید.





چرا ASP.NET Core را انتخاب کنیم؟

ASP.NET Core یک وب فریم‌ورک قدرتمند است که از آن می‌توان برای ایجاد سریع‌تر، راحت‌تر و امن‌تر انواع اپلیکیشن‌ها استفاده نمود. این وب فریم‌ورک پر از ویژگی‌های جالب است که در اینجا چند نمونه از مهم‌ترین آنها بیان می‌شود:

- ASP.NET Core یک فریم‌ورک مدرن، Scalable، Open Source و با Performance بالا است.
- ASP.NET Core یک معماری ماژولار برای Maintenance راحت‌تر دارد.
- ASP.NET Core مدیریت Cross-Site Request Forgery (CSRF) را برعهده می‌گیرد و می‌تواند روی هر پلتفرمی اجرا شود.

اما این فریم‌ورک برخی پیشرفت‌های زیربنایی هم داشته که در ادامه چند نمونه ذکر شده است:

- Middleware Pipeline برای تعریف رفتارهای اپلیکیشن
- داشتن یک Dependency Injection تو کار
- ترکیب زیرساخت (MVC) UI و (Web API) API
- سیستم پیکربندی بسیار گسترده
- قابل بودن برای پلتفرم‌های Cloud (با استفاده از برنامه‌نویسی غیرهمزمان)

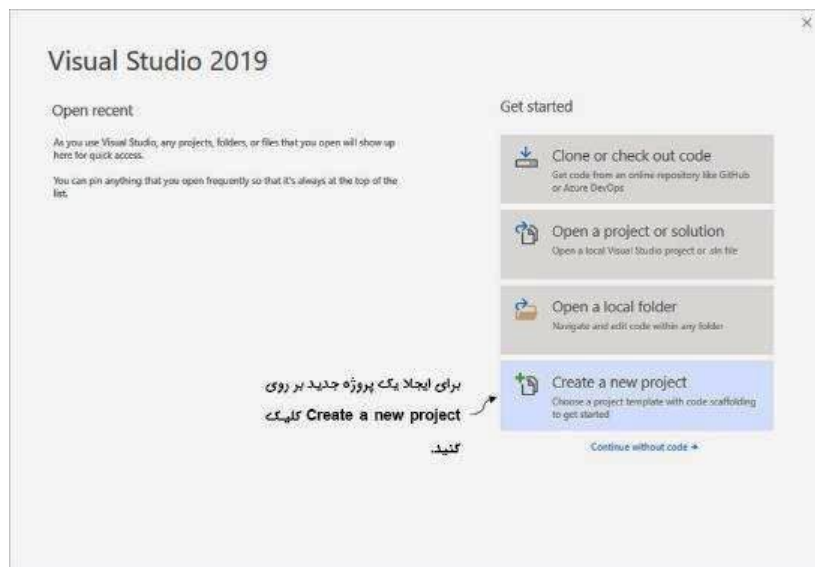


ایجاد اولین برنامه ASP.NET Core

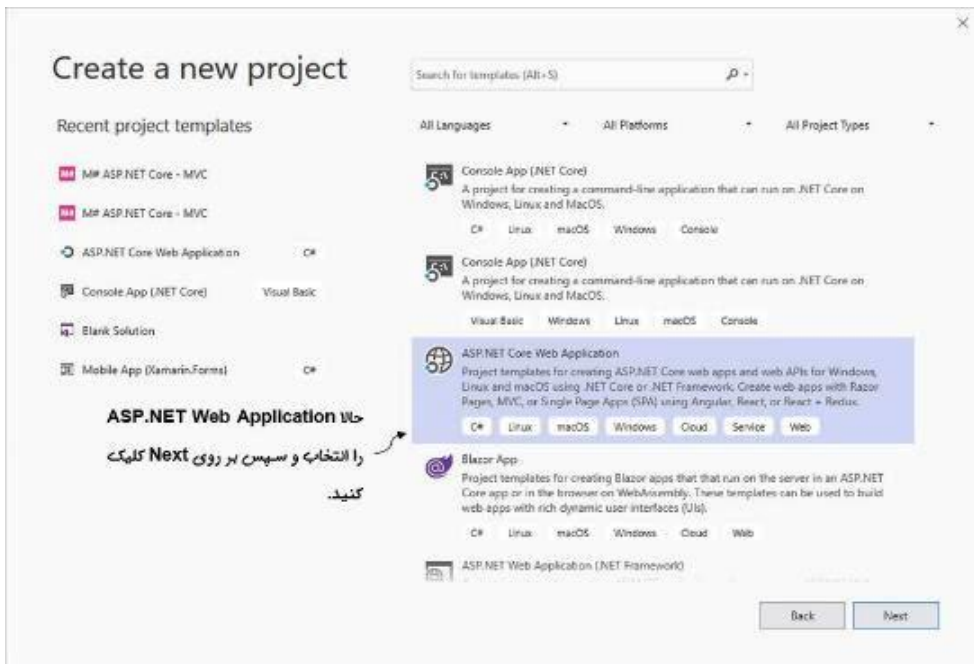
اولین وب اپلیکیشن خود را ایجاد کنید.

برای درک بهتر با ویژوال استودیو ۲۰۱۹ یک اپلیکیشن Empty از ASP.NET Core ایجاد کنید.

- ویژوال استودیو ۲۰۱۹ را باز کنید و بر روی Create a new project کلیک کنید.



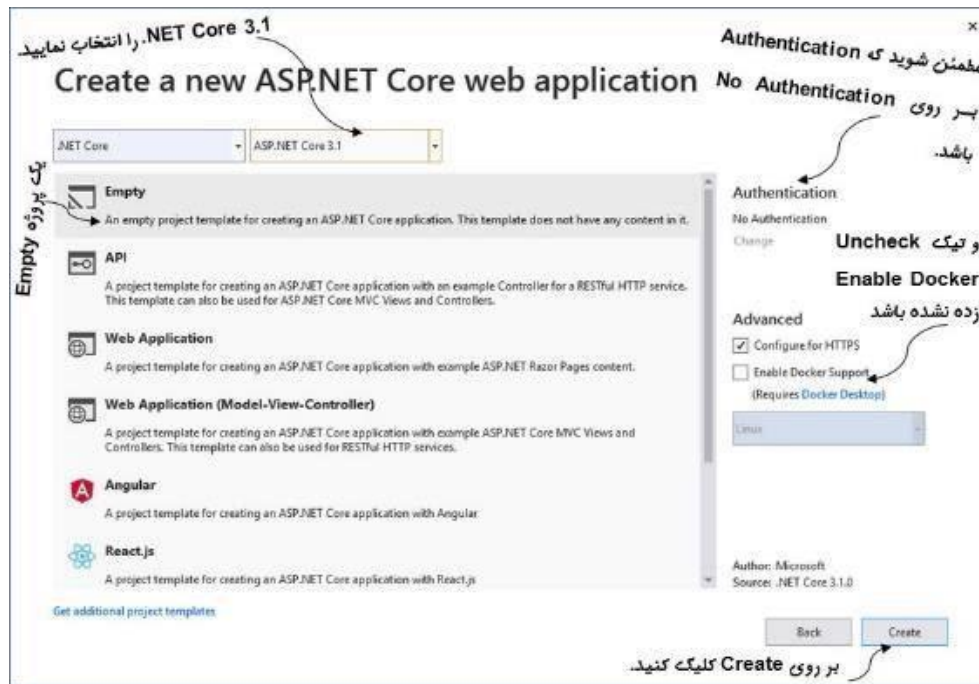
- در کادر بعدی ASP.NET Core Web Application را انتخاب و بر روی Next کلیک کنید.



- حالا نام پروژه را Microdev.ASPNETCore بگذارید، سپس مکان ذخیره سازی و نام Solution را همانند تصویر وارد و بر روی Create کلیک نمایید.

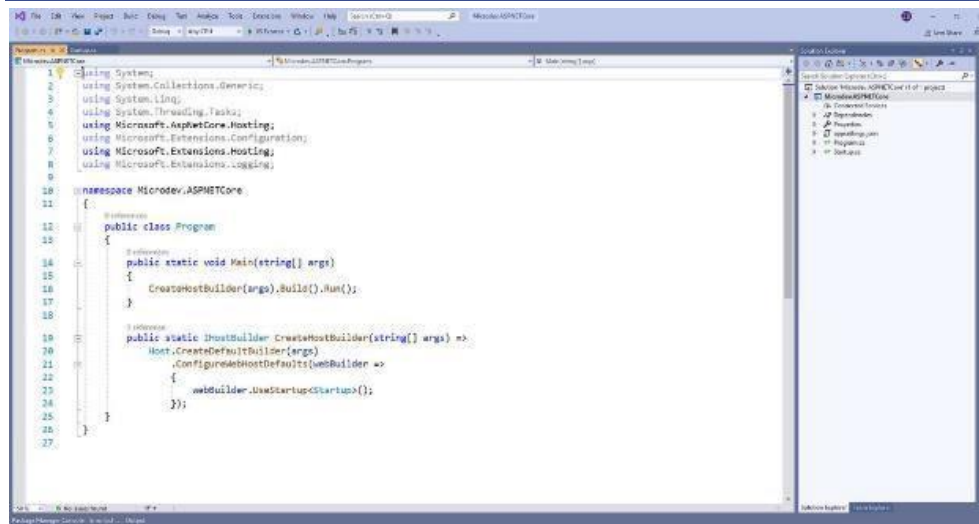


- در کادر بعدی Empty و ASP.NET Core 3.1 را انتخاب کنید.



نکته

مطمئن شوید که Enable Docker Support تیک نخورده و اپلیکیشن بر روی No Authentication تنظیم شده است. منتظر بمانید تا Visual Studio اپلیکیشن شما را ایجاد نماید.

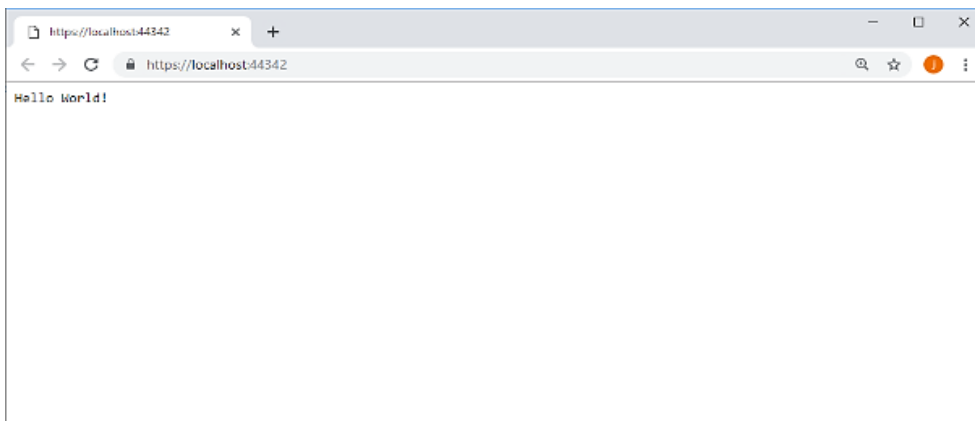


اولین وب اپلیکیشن ایجاد شد.



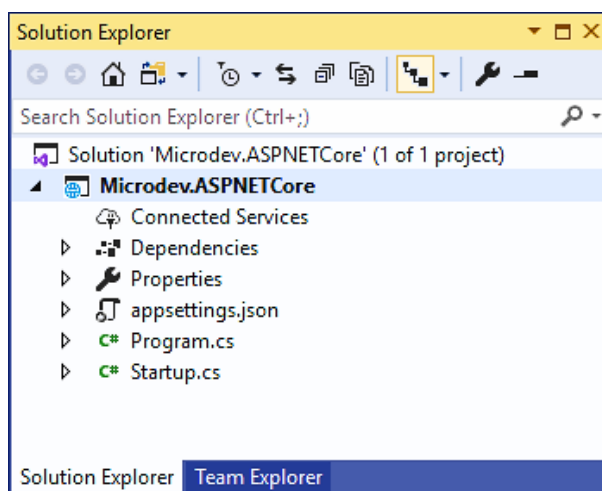
اجرای وب اپلیکیشن

برنامه را اجرا کنید. پروژه را Build و سپس F5 را بزنید. (یا اینکه بر روی پیکان سبز رنگ در نوار ابزار کنار IIS Express کلیک نمایید.)



درک ساختار پروژه

به عناصر درون Solution نگاه بیندازید:



- **Dependencies و Connected Services:** این دو قسمت برای نمایش تمام وابستگی‌های NuGet packages، وابستگی‌های Client-Side و سرویس‌های از دور وابسته به پروژه می‌باشد.
- **Properties:** فولدر Properties تنها یک فایل launchSettings.json برای کنترل نحوه اجرای Visual Studio و Debug اپلیکیشن دارد.
- **Program.cs و Startup.cs:** این دو فایل هم برای راه‌اندازی وب سرور و Pipeline استفاده می‌شود.



کلاس Program

کلاس Program مسئول پیکربندی بسیاری از زیرساخت‌های اپلیکیشن شماست و تمامی اپلیکیشن‌های ASP.NET Core با این فایل شروع می‌شوند.

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }
    public static IHostBuilder CreateHostBuilder(string[] args)
    =>Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
}
```

متد Main نقطه شروع برنامه است که در آن بسیاری از پیکربندی‌های برنامه انجام می‌شود. هنگامی که اپلیکیشن شما Start می‌شود، این متد باید یک Host را پیکربندی و اجرا کند که این Host وظیفه مدیریت Startup و Lifetime اپلیکیشن را به عهده دارد و حداقل باید پیکربندی سرور و Request processing Pipeline را انجام دهد.

شاید بپرسید Host چیست؟ برای کنترل شروع و پایان اپلیکیشن، مدیریت چرخه عمر اپلیکیشن و تمام منابع وابسته به آن (مثل Dependency injection، Logging، Configuration) پیاده‌سازی‌های IHostedService را درون یک object به نام Host کپسوله می‌کنیم.

پیکربندی، ساخت و اجرای Host معمولاً توسط کدی که در کلاس Program نوشته شده انجام می‌شود. ابتدا متد Main، متدی با نام CreateHostBuilder را جهت پیکربندی و ساخت شی Builder فراخوانی و در پایان متد Build و Run شی Builder را صدا می‌زند.

نکته

- اگر از Entity Framework Core استفاده می‌کنید، باید Signature متد CreateHostBuilder را تغییر ندهید، زیرا Entity Framework Core tools برای اینکه بتواند بدون اجرای اپلیکیشن، پیکربندی Host را انجام دهد، از متد CreateHostBuilder استفاده می‌کند.
- کلاس Startup که در متد جنریک <> UseStartup رفرنس داده شده، جایی است که پیکربندی سرویس‌ها و تعریف Middleware Pipeline انجام می‌شود.

**کلاس Startup**

اپلیکیشن‌های ASP.NET Core برای پیکربندی برخی رفتارهای اپلیکیشن از یک کلاس Startup استفاده می‌کنند. این کلاس از هیچ کلاس پایه‌ای ارث‌بری نمی‌کند و هیچ اینترفیسی را هم پیاده‌سازی نخواهد کرد و کار پیکربندی سرویس و تعیین Middleware Pipeline را انجام می‌دهد.

کلاس Startup برای بارگذاری و پیکربندی کامپوننت‌های اپلیکیشن، شامل دو متد ConfigureServices و Configure می‌باشد.

نکته

- HostBuilder ایجاد شده در کلاس Program ابتدا متد ConfigureServices و سپس Configure را صدا می‌زند، بنابراین تمام سرویس‌های رجیستر شده در متد ConfigureServices در متد Configure قابل دسترسی هستند.
- سرویس، اشاره به کلاسی دارد که بتواند یک Functionality را در اپلیکیشن فراهم کند.

**متد ConfigureServices**

متد ConfigureServices یک متد اختیاریست که رجیسترکردن سرویس‌های اپلیکیشن با استفاده از سیستم تزریق وابستگی (Dependency Injection) ASP.NET Core درون این متد انجام می‌شود. بنابراین این متد جایی است که تزریق وابستگی در آن پیکربندی می‌شود.

تزریق وابستگی (Dependency Injection) چیست؟

تزریق وابستگی یک تکنیک بسیار مهم و کاربردیست که باعث می‌شود Instance های یک کلاس در زمان اجرا، ایجاد شده و شما به راحتی کدهای Tastable و Loosely Coupled بنویسید.

برای مثال: فرض کنید شما یک کلاس EmployeeService و یک کنترلر EmployeeController دارید که این کنترلر در زمان اجرا نیاز به یک Instance ای از EmployeeService دارد.

رویکرد کلی این است که هر زمان که نیاز به یک سرویس داشتید از کلمه کلیدی new استفاده کنید و یک Instance از آن سرویس ایجاد نمایید. متأسفانه مشکل این رویکرد این است که کد شما به پیاده‌سازی خاصی وابسته (Tightly couple) می‌شود و نگهداری و تست اپلیکیشن سخت خواهد شد.

Tightly couple چیست؟

Tightly couple یعنی یک کلاس به پیاده‌سازی کلاس دیگر وابسته باشد. در این روش تغییر کلاس کمکی بدون دست زدن به کلاس اصلی غیرممکن خواهد بود. چون در کلاس اصلی مستقیماً به کلاس کمکی رفرنس داده شده است.

اما راه حل این مشکل: بعد از نوشتن سرویس، شما می‌توانید وابستگی‌های خود را اعلام کرده و اجازه دهید تا کلاسی دیگر این وابستگی‌های را برای شما فراهم کند. این تکنیک با نام تزریق وابستگی یا معکوس شدن کنترل IOC (Inversion Of Control) شناخته می‌شود.

در برنامه‌های ASP.NET Core، رجیستر شدن سرویس‌ها در متد `ConfigureServices` انجام می‌شود و هر زمان که از یک ویژگی جدید در ASP.NET Core در اپلیکیشن‌تان استفاده کنید، باید به این متد برگردید و سرویس‌های مورد نیاز را اضافه نمایید.

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddScoped<EmpolyeeService>();
    }
}
```

پارامتر `IServiceCollection` (در ورودی این متد) شامل لیستی از سرویس‌های مورد نیاز اپلیکیشن است.

◀ AddScoped چیست؟

`AddScoped` متدی است که طول عمر یک سرویس را مشخص می‌کند. این طول عمر بدین صورت است که با هر بار درخواست وب، یک `Instance` جدید از کلاس `EmpolyeeService` ایجاد خواهد شد.

نکته

- اگر فراموش کنید یک سرویس را رجیستر نمایید، در زمان اجرا یک `InvalidOperationException` دریافت خواهید کرد.



متد Configure

`Configure` متدی است که با استفاده از اکستنشن متدهای موجود بر روی `IApplicationBuilder` ماژول‌هایی را به `Request Pipeline` اضافه و رفتارهای اپلیکیشن را تعریف می‌نماید. به این ماژول‌ها `Middleware` گفته می‌شود.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
```

```
{
    endpoints.MapGet("/", async context =>
    {
        await context.Response.WriteAsync("Hello World!");
    });
}
```

همانطور که می‌بینید این متد معمولاً دو پارامتر می‌پذیرد:

IApplicationBuilder و IWebHostEnvironment

- پارامتر IApplicationBuilder، ترتیب اجرای Middleware ها را تعریف می‌کند. بنابراین Middleware تنها با این اینترفیس به Pipeline اپلیکیشن اضافه خواهد شد. برای مثال UseDeveloperExceptionPage()، اکستنشن متدی بر روی آرگومان IApplicationBuilder است.
- پارامتر IWebHostEnvironment شی است که اطلاعاتی در مورد محیط جاری را در خود نگه می‌دارد. بنابراین از آن می‌توان برای ارائه رفتارهای متفاوت در محیط‌های توسعه استفاده نمود.