مجموعه کتابهای برنامه نویسی

ASP .NET Core

مارینکو اسپاسوویک - ولادیمیر پکاناک

آموزش ساخت API با ASP.NET Core

مروّت گیوی

آموزش ساخت API

با ASP.NET CORE

توجه :

کتاب حاضـر حاصـل ترجمه **مروّت گیوی**، کارشـناس ارشـد آموزش پرسـتاری می باشـد. فایل کتاب حاوی اطلاعات **DRM** (مدیریت حقوق دیجیتال) اسـت. 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.

خواهشـمند اسـت به حقوق نگارنده و انتشـارات ترجمک احترام گذاشـته و از توزیع بدون مجوز فایل کتاب اجتناب نمایید. شـما با خرید و دانلود این کتاب موافقت نموده اید که اطلاعات فایل DRM به سـرور انتشـارات ترجمک انتقال یابد و در صـورت محرز شـدن نقض حقوق صـاحب اثر، کلیه خسـارات حاصـله 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 عرضـه مقرون به صـرفه کتاب های الکترونیک و شـکوفایی انتشـارات

آموزش ساخت API

با ASP.NET Core

مارینکو اسپاسوویک

ولادیمیر پکاناک

ترجمه

مروت گیوی

پاییز ۱۴۰۰

سرشناسه	: اسپاسویویک، مارینکو - Spasojevic, Marinko	
عنوان و نام پدیدآور	: آموزش ساخت API با ASP.NET Core/مارینکو اسپاسوویک، ولادیمیر پکاناک ؛ ترجمه مروت گیوی.	
مشخصات نشر	: همدان – ترجمک ۱۴۰۰	
مشخصات ظاهری	: ز، ۳۲۲ ص.: مصور (بخشی رنگی)، جدول (بخشی رنگی).	
شابک	: ۴۲۰۰۰۰ ریال 978-622-98398-9-8	
یادداشت	: عنوان اصلی: .Ultimate ASP.NET Core Web API	
شناسه افزوده	: گیوی، مروت، ۱۳۴۷-، مترجم	
رده بندی کنگره	:	QA76/76
رده بندی دیویی	:	005/3
شماره کتابشناسی ملی	:	8448640

شناسنامه کتاب

نام کتاب: آموزش ساخت API با ASP.NET Core

نویسنده: مارینکو اسپاسوویک و ولادیمیر پکاناک

گردآوری و ترجمه: مروّت گیوی

ناشر: انتشارات ترجمک

صفحه آرایی: انتشارات ترجمک

طراحی جلد: محمدحسین گیوی

نوبت چاپ: چاپ اول، ۱۴۰۰

قیمت: ۴۲۰۰۰ تومان

چاپ: گروه نشر الکترونیک ترجمک

شابک: ۸-۹-۹۸۳۹۸-۶۲۲-۹۷۸

تلفن تماس: ۰۹۱۸۱۵۰۶۱۰۰

تارنمای اینترنتی: https://tarjomac.ir

ISBN:978-622-98398-9-8

پیشگفتار

API چیست؟ واسط برنامه نویسی اپلیکیشن (API)[1] واسط نرم افزاری است که امکان برقراری ارتباط بین دو اپلیکیشن را فراهم می کند. هر بار کا شما از اپلیکیشن هایی مثل فیسبوک استفاده می کنید، پیام رسان استفاده می کنید یا آب و هوا را در گوشی همراه خود چک می کنید، از یک API استفاده می کنید.

اما واقعا API چیست؟

وقتی شما از یک اپلیکیشن بر روی تلفن همراه یا کامپیوتر خود استفاده می کنید، اپلیکیشن به اینترنت وصل شده و داده را به سرور خود انتقال می دهد. سرور داده را بازیافت کرده و تفسیر می کند، سپس اقدامات لازم را روی آن انجام داده و پاسخ را به تلفن همراه و شما بر می گرداند. سپس اپلیکیشن داده ها را تفسیر کرده و اطلاعاتی که شما می خواستید و درخواست داده بودید را به شیوه ای قابل خواندن توسط انسان عرضه می کنید. این کاری است که یک API انجام می دهد و تمام اتفاقی است که در یک API می افتد.

معرفی API با مثال

تصور کنید در رستوران نشسته اید و منو را برای انتخاب غذا در دست گرفته اید. آشپزخانه بخشی از سیستمی است که سفارش شما را آماده می کند. چیزی که وجود ندارد یک لینک ارتباطی حیاتی است که شما را به آشپزخانه وصل کند و سفارش شما را به آشپزخانه بدهد و سپس غذا را به میز شما برگرداند. این کاری است که پیشخدمت رستوران یا API انجام می دهد.

[1] Application Programming Interface

پیشخدمت رستوران یک پیام رسان یا API است که درخواست سفارش شما را می گیرد و به آشپزخانه (سیستم) اطلاع می دهد که چکار کند. سپس آشپزخانه سفارش را آماده کرده و پیشخدمت سفارش شما یا غذای شما را به سر میز شما بر می گرداند.

این یک مثال از API در زندگی واقعی بود. ممکن است با فرآیند جستجوی آنلاین بلیط هواپیما آشنا باشید. همانند رستوران، گزینه های زیادی وجود دارد که می توانید از بین آنها انتخاب کنید از جمله شهرهای مختلف، تاریخ رفت و برگشت و مانند اینها.

فرض کنید شما پروازی را از یک وبسایت خطوط هوایی رزرو می کنید. نام شهر و تاریخ پرواز، شهر و تاریخ برگشت، کلاس کابین هواپیما و متغیرهای دیگر را انتخاب می کنید. برای اینکه بتوانید بلیط را رزرو کنید بایستی بتوانید با وبسایت خطوط هوایی تعامل برقرار کنید و به بانک اطلاعاتی آنها دسترسی داشته باشید تا بتوانید صندلی های خالی در تاریخ مورد نظر را پیدا کرده و قیمت ها را مقایسه کنید.

هرچند اگر از وبسایت خطوط هوایی استفاده نکنید، کانالی که به اطلاعات دسترسی دارد، چطور؟ اگر از خدمات مسافرتی آنلاین استفاده کنید چطور که بانک اطلاعاتی تجمعی چندین خطوط هوایی را در دسترس دارند؟

در این مثال سرویس مسافرتی با API خطوط هوایی ارتباط برقرار می کند. API واسطی است که همانند خدمتکار رستوران، می توانید از او در مورد خدمات مسافرتی سئوال کنید و اطلاعات لازم را بگیرید. بنابراین API خود یک نرم افزار یا دقیق تر بگوییم یک میان افزار است که رابط مشتریان و مصرف کنندگان و بانک یا بانک های اطلاعاتی مرتبط است. مصرف کننده از طریق وبسایت یا اپلیکیشن سرویس دهنده خود خدماتی را درخواست می کند. این درخواست به API فرستاده می شود. API درخواست را پردازش کرده و اطلاعات لازم را از بانک یا بانک های اطلاعاتی متصل به آن دریافت می کند. سپس صحت اطلاعات را بررسی کرده و رمز گذاری می کند. سپس پاسخ رمز گذاری شده را به سرویس دهنده شما بر می گرداند تا با ظاهری قابل درک برای انسان به شما نمایش داد شود. بنابراین API یک لایه امنیتی نیز هست.

API یک لایه امنیتی فراهم می کند

داده های تلفن همراه بطور کامل در معرض دید سرور قرار نمی گیرد، همانطور که سرور بطور کامل در اختیار تلفن همراه شما نیست. بجای آن، هر کدام چند بسته کوچک داده را به ارتباط گذاشته و اطلاعات

ضروری را به اشتراک می گذارند. شما به رستوران می گویید می خواهید چه بخورید، آنها به شما پاسخ داده و در نهایت همان وعده غذایی را به شما می دهند.

امروز API نقش بسیار مهمی پیدا کرده و در ارتباطات آنلاین سرمایه ای ارزشمند محسوب می شود بیشتر بخاطر اینکه بخش مهمی از عایدات تجاری را تشکیل می دهد. شرکت های بزرگی چون گوگل یا آمازون چند نمونه از شرکت هایی است که از API خود پول سازی می کنند. امروزه واژه جدیدی بنام اقتصاد API [1] مصطلح شده است که به بازار درآمد APIها اشاره دارد.

API مدرن

در طول سالیان گذشته API به عنوان واسط ارتباطی ژنریک با یک اپلیکیشن توصیف می شد. اخیرا API مدرن ویژگی های جدیدی گرفته است که آن را فوق العاده مفید و ارزشمند کرده است:

- یک API مدرن از استانداردها (غالبا HTTP و REST) تبعیت می کند، استانداردی که برنامه نویس پسند بوده، به سادگی در دسترس بوده و بخوبی درک شده است
- بیشتر و بیشتر به عنوان محصول در نظر گرفته می شوند تا کد برنامه نویسی. APIها برای مصرف مخاطبان خاص طراحی می شوند (مثلا برنامه نویسان موبایل)، کاملا مستند بوده و نسخه بندی می شوند به شیوه ای که کاربران بتوانند توقعات خاص خود را همزمان با نگهداری و به روز کردن چرخه عمر، برآورده کنند.
- از آنجایی که APIهای مدرن استاندارد شده تر هستند، نظم قوی تری از نظر امنیت و حاکمیت دارند و عملکرد و مقیاس کار خود را پایش و مدیریت می کنند.
- به عنوان یک قطعه نرم افزاری قابل بازاری کردن، API مدرن دارای چرخه برنامه نویسی مختص خود (SDLC)[2] برای طراحی، تست، ساخت، مدیریت و نسخ بندی است. همچنین APIهای مدرن از نظر مصرف و نسخه بندی کاملا مستند هستند.

[1] API economy

[2] software development lifecycle

کتاب حاضر

اگر به API نیاز دارید، به تهیه API علاقمند هستید و می خواهید اطلاعات جامع تری در مورد API و طراحی API داشته باشید، از این کتاب می توانید بهره ببرید. کتاب راهنمای ساخت API با ASP.NET Core در ۲۹ فصل و به شرح زیر نیاز شما را برآورده می سازد.

ابتدا طریقه ساخت و پیکربندی پروژه API بحث شده است، سپس سرویس لاگینگ یا یاداشت برداری جهت ذخیره اقدامات و خطاها آموزش داده شده است. در مرحله بعد طریقه ساخت بانک اطلاعاتی و الگوی مخزن داده، شیوه هندل درخواست ها، خطاهای گلوبال، بدست آوردن منابع برای API خود، مذاکره محتوا و ایمنی و پایایی متدها و غیره بحث شده است.

در فصول ۱۰ تا ۱۵ کتاب شیوه کار با درخواست های DELETE، درخواست های PUT، درخواست های PATCH، اعتبارسنجی، کد آسنکرون و فیلترهای اکشن بحث شده است. فصل ۱۶ تا ۲۱ از بین فیلترهای اکشن به آموزش صفحه بندی، فیلترینگ، جستجو، مرتب کردن و شکل دهی داده و نهایتا پشتیبانی از HATEOAS پرداخته است.

فصول بعدی کتاب نیز به مباحث مهمی چون کار با OPTIONS و درخواست های HEAD، سند ریشه، نسخه بندی API و کش کردن پرداخته است. محدودیت ریت و گلوگاه، JWT و آیدنتیتی، مستند کردن APIبا SWAGGER و نهایتا انتقال به IIS در فصول ۲۵ تا ۲۹ پوشش داده شده است.

کلیه مراحل آموزش داده شده در کتاب با استفاده از درخواست های Postman تست و ارزیابی شده است. بنابراین می توانید از مثال مدیریت شرکت ها و کارمندان کتاب به عنوان قالب زمینه استفاده کنید.

مروّت گیوی

فهرست مندرجات

۱- پیکربندی پروژه

پیکربندی[1] پروژه ها در NET Core. با آنچه که عادت داشـــتیم در پروژه های دات نت فریم ورک انجام می دادیم، تفاوت زیادی دارد. حالا مجبور نیســـتیم از فایل web.config اســـتفاده کنیم، بلکه بجای آن از یک فریم ورک پیکربندی سر خود که با NET Core. عرضه شده است، استفاده می کنیم.

برای اینکه بتوانیم یک اپلیکیشن خوب و قوی ایجاد کنیم، لازم است که طریقه پیکربندی اپلیکیشن خود و همچنین سرویس هایی که استفاده خواهد کرد، را ابتدا بیاموزیم.

در این بخش ما به متدهای پیکربندی اپلیکیشن در کلاس Startup می پردازیم و سپس اپلیکیشن خود را تنظیم می کنیم. همچنین طریقه رجیستر کردن سرویس های مختلف و شیوه استفاده از **متدهای بسط یافته**[2] یا متد افزونه برای دستیابی به این هدف را بحث می کنیم.

البته قبل از هر چیزی لازم اســت که یک پروژه جدید خلق کنیم. بنابراین ابتدا به خلق پروژه در **ویژوال استودیو**[3] می پردازیم.

---

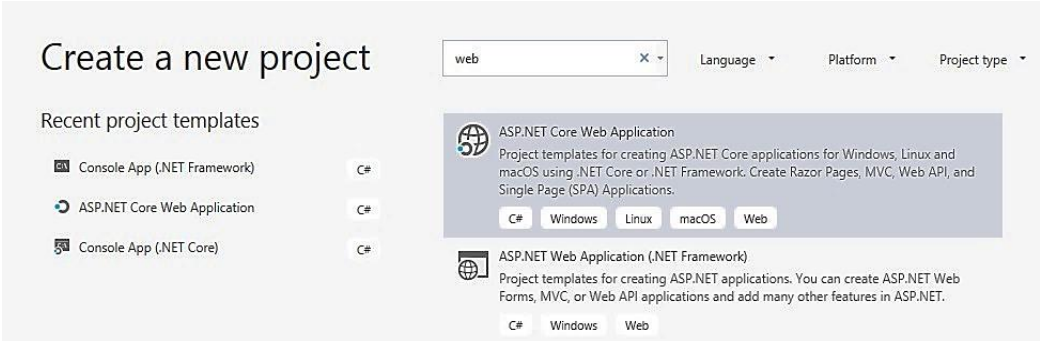
[1] Configuration

[2] Extension Methods

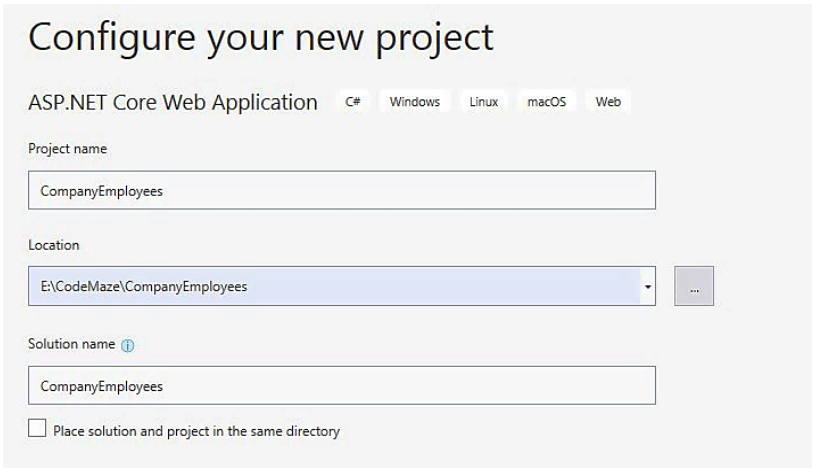
[3] Visual Studio

۱-۱ خلق پروژه جدید

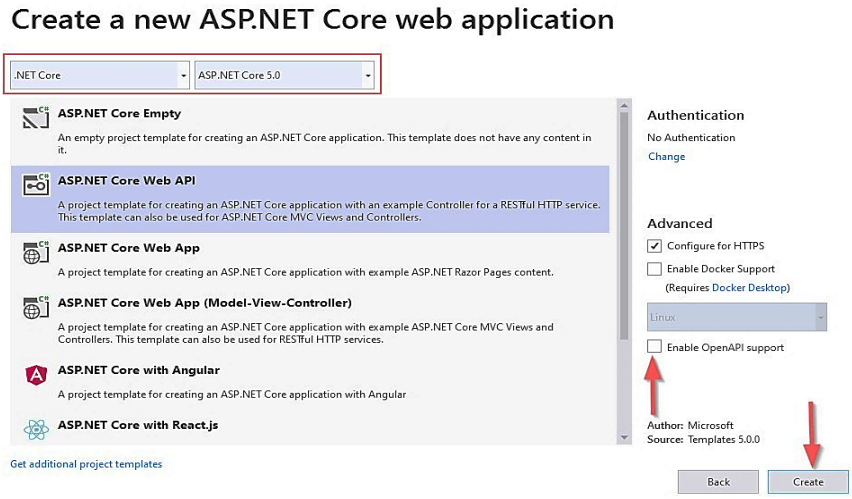
ابتدا برنامه ویژوال اســتودیو را باز کرده و یک اپلیکیشــن ASP.NET Core Web Application از منوی فایل ایجاد می کنیم:



حالا در این دیالوگ نام و محل ذخیره پروژه خود را مشخص می کنیم:



در مرحله بعد می خواهیم .NET Core و ASP.NET Core 5.0 را از لیســت کشــویی به ترتیب انتخاب کنیم. همچنین فعلا نمی خواهیم پشــتیبانی از OpenAPI را فعال کنیم. در ادامه کتاب بطور دســتی اینکار را خواهیم کرد. حالا پیش رفته و دکمه Create را کلیک می کنیم تا مقداردهی (ساخت) پروژه ما شروع شود:



۲-۱ پیکربندی فایل launchSettings.json

پس از اینکه پروژه خلق شـد، نوبت اصـلاح فایل launchSettings.json آن اسـت که می توان آن را در بخش Properties پنجره کاوشگر پروژه یا Solution Explorer پیدا کرد.

این پیکربندی مشـخص می کند که رفتار زمان اجرا و بالا آمدن اپلیکیشـن ASP.NET Core چطور باشـد. همانطور که خواهیم دید، این فایل حاوی دو تنظیمات راه اندازی و پیکربندی برای IIS و اپلیکیشن های خود-میزبان (Kestrel) است.

فعلا خصـیصـه launchBrowser را به false تغییر می دهیم تا از باز شـدن مرورگر وب در هر بار اسـتارت اپلیکیشن جلوگیری شود.

```
{
    "$schema": "http://json.schemastore.org/launchsettings.json",
    "iisSettings": {
        "windowsAuthentication": false,
        "anonymousAuthentication": true,
        "iisExpress": {
            "applicationUrl": "http://localhost:58753",
            "sslPort": 44370
        }
    },
```

```
    "profiles": {
        "IIS Express": {
            "commandName": "IISExpress",
            "launchBrowser": false,
            "launchUrl": "weatherforecast",
            "environmentVariables": {
                "ASPNETCORE_ENVIRONMENT": "Development"
            }
        },
        "CompanyEmployees": {
            "commandName": "Project",
            "dotnetRunMessages": "true",
            "launchBrowser": false,
            "launchUrl": "weatherforecast",
            "applicationUrl": "https://localhost:5001;http://localhost:5000",
            "environmentVariables": {
                "ASPNETCORE_ENVIRONMENT": "Development"
            }
        }
    }
}
```

این محیط ما خواهد بود از آنجایی که در حال ساخت یک پروژه Web API هستیم و نیازی به مرورگر برای چک کردن API خود نداریم. ما برای اینکار از Postman استفاده خواهیم کرد (بعدا بحث می شود).

اگر قبلا تیک گزینه Configure for HTTPS را در فاز setup برنامه زده باشید، در بخش applicationUrl دو URL خواهید داشت: یکی برای HTTP و دیگری برای HTTPS.

در این فایل متوجه خصیصه sslPort نیز شدید که نشان می دهد اپلیکیشن ما در زمان اجرا در IISExpress طوری پیکربندی می شود که از HTTPS (port 44370) نیز استفاده کند.

اطلاعات بیشتر: توجه داشته باشید که این پیکربندی HTTPS تنها در محیط لوکال معتبر است. اگر بخواهید اپلیکیشن را منتشر کنید، بایستی یک مجوز SSL معتبر و ریدایرکت HTTPS را پیکربندی کنید.

یک خصیصه مفید دیگر برای ساخت محلی اپلیکیشن وجود دارد و آن خصیصه launchUrl است. این خصیصه مشخص می کند کدام URL را اپلیکیشن در شروع ناوبری کند. برای اینکه خصیصه launchUrl کار

کند، لازم است که خصیصه launchBrowser به true ست شود. بنابراین برای مثال، اگر ما خصیصه launchUrl را به weatherforecast ست کنیم، زمان اجرای اپلیکیشن خود به آدرس https://localhost:5001/weatherforecast ریدایرکت خواهیم شد.

۳-۱ شرح فایل های Program.cs و Startup.cs

فایل Program.cs نقطه ورود اپلیکیشن ماست و به شکل زیر خواهد بود:

```
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>();
        });
}
```

اگر با روند کار در .NET Core 1.0 آشنا باشید، این کد را خیلی کوچکتر از چیزی که باید باشد، در می یابید. ممکن است تعجب کنید که چرا بخش هایی مثل UseKestrel() یا UseIISIntegration() وجود ندارد. متد CreateDefaultBuilder(args) تمامی مسائل آنها را کپسولی کرده و کد را خواناتر کرده است، اما همه کارهای آنها را انجام می دهد. اما هنوز می توانید پیکربندی سابق را اعمال کنید، اگر خودتان بخواهید.

متد CreateDefaultBuilder(args) فایل ها و متغیرهای دیفالت پروژه و پیکربندی لاگر[1] را ست میکند. این حقیقت که لاگر قبلا و در فرآیند بوت استرپ کردن[2] پیکربندی می شود بدین معنی است که حالا می توانیم مسائلی که در طی بوت استرپ کردن رخ می دهد را نیز یاداشت کنیم، چیزی که در نسخه های قبلی اندکی سخت بود.

1 Logger Configuration

2 Bootstrapping Process