

مجموعه کتابهای برنامه نویسی

ASP.NET Core

مارینکو اسپاسویک - ولادیمیر پکاناک

آموزش ساخت API با

ASP.NET Core

مروّت گیوی

ترجمه
Barjoniac

ASP.net



آموزش ساخت API

با ASP.NET CORE

توجه :

کتاب حاضر حاصل ترجمه **مروّت گیوی**، کارشناس ارشد آموزش پرستاری می باشد. 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 نگارنده و انتشارات 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



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 اسپاسویویک، ولادیمیر پکاناک : ترجمه مروت گیوی.
مشخصات نشر	: همدان – ترجمک ۱۴۰۰
مشخصات ظاهری	: ز، ۳۲۲ ص: مصور (بخشی رنگی)، جدول (بخشی رنگی).
شابک	: ۹۷۸-۶۲۲-۹۸۳۹۸-۹-۸ ریال ۴۲۰۰۰۰
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 اسپاسویویک و ولادیمیر پکاناک

گردآوری و ترجمه: مروت گیوی

ناشر: انتشارات ترجمک

صفحه آرای: انتشارات ترجمک

طراحی جلد: محمدحسین گیوی

نوبت چاپ: چاپ اول، ۱۴۰۰

قیمت: ۴۲۰۰۰ تومان

چاپ: گروه نشر الکترونیک ترجمک

شابک: ۹۷۸-۶۲۲-۹۸۳۹۸-۹-۸

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 که 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 گوئید 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 API¹ 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 (غالباً HTTP و REST) تبعیت می کند، استاندارد می کند، استاندارد می کند 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 می‌خواهید اطلاعات جامع‌تری در مورد API و طراحی API داشته باشید، از این کتاب می‌توانید بهره‌بردار شوید. کتاب راهنمای ساخت API با ASP.NET Core در ۲۹ فصل و به شرح زیر نیاز شما را برآورده می‌سازد.

ابتدا طریقه ساخت و پیکربندی پروژه API بحث شده است، سپس سرویس لاگینگ یا یادداشت برداری جهت ذخیره اقدامات و خطاها آموزش داده شده است. در مرحله بعد طریقه ساخت بانک اطلاعاتی و الگوی مخزن داده، شیوه‌های هندل درخواست‌ها، 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 درخواست‌های DELETE، درخواست‌های PUT، درخواست‌های 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 نهایتاً پشتیبانی از HATEOAS پرداخته است.

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

کلیه مراحل آموزش داده شده در کتاب با استفاده از درخواست‌های Postman تست و ارزیابی شده است. بنابراین می‌توانید از مثال مدیریت شرکت‌ها و کارمندان کتاب به عنوان قالب زمینه استفاده کنید.

مروت گیوی

فهرست مندرجات

صفحه	عنوان
۱	۱- پیکربندی پروژه
۲	۱-۱ خلق پروژه جدید
۳	۲-۱ پیکربندی فایل LAUNCHSETTINGS.JSON
۵	۳-۱ شرح فایل های PROGRAM.CS و STARTUP.CS
۶	۴-۱ متدهای افزونه و پیکربندی CORS
۸	۵-۱ پیکربندی IIS
۱۰	۶-۱ سایر کدهای کلاس استارت آپ
۱۱	۷-۱ تنظیمات وابسته به محیط
۱۳	۲- پیکربندی سرویس یادداشت برداری
۱۴	۱-۲ ایجاد پروژه های لازم
۱۴	۲-۲ ایجاد واسط ILOGGERMANAGER و نصب NLOG
۱۶	۳-۲ پیاده سازی اینترفیس و فایل NLOG.CONFIG
۱۸	۴-۲ پیکربندی سرویس یادداشت برداری برای پیام های لاگینگ
۱۹	۵-۲ DI، IOC و تست سرویس لاگر
۲۲	۳- مدل بانک اطلاعاتی و الگوی مخزن داده
۲۳	۱-۳ خلق مدل ها
۲۵	۲-۳ کلاس زمینه و اتصال به بانک اطلاعاتی
۲۷	۳-۳ مهاجرت و کاشت داده اولیه
۳۱	۴-۳ منطق الگوی مخزن

۳۳.....	۵-۳ واسط کاربری و کلاس های مخزن
۳۵.....	۶-۳ ایجاد مدیر مخزن
۳۹.....	۴- رفع و رجوع درخواست های GET.....
۳۹.....	۱-۴ کنترلرها و مسیریابی در WEB API.....
۴۲.....	۲-۴ نامگذاری منابع خود
۴۳.....	۳-۴ گرفتن همه شرکت ها از بانک اطلاعاتی
۴۶.....	۴-۴ تست نتیجه با پستمن
۴۸.....	۵-۴ کلاس های DTO در مقایسه با کلاس های مدل ENTITY
۵۰.....	۶-۴ استفاده از نقشه کننده خودکار در ASP .NET CORE
۵۴.....	۵- رفع و رجوع خطای گلوبال.....
۵۴.....	۱-۵ هندل گلوبال خطاها با استفاده از میان افزار سر خود
۵۶.....	۲-۵ اصلاح کلاس STARTUP
۵۷.....	۳-۵ تست کردن نتیجه.....
۵۹.....	۶- بدست آوردن منابع بیشتر
۵۹.....	۱-۶ بدست آوردن یک منبع منفرد از بانک اطلاعاتی
۶۱.....	۲-۶ رابطه والد /دختر در WEB API
۶۶.....	۳-۶ برگرداندن یک کارمند از شرکت
۶۸.....	۷- مذاکره محتوا
۶۸.....	۱-۷ بدون تنظیمات سفارشی چه چیزی بدست می آوریم؟
۶۹.....	۲-۷ تغییر پیکربندی پیش فرض پروژه خود
۷۰.....	۳-۷ تست مذاکره محتوا
۷۱.....	۴-۷ محدود کردن نوع رسانه
۷۲.....	۵-۷ شرح بیشتر فرمت کننده ها.....
۷۳.....	۶-۷ پیاده سازی یک فرمت کننده سفارشی
۷۶.....	۸- ایمنی و پایایی متد

۷۸.....	۹- خلق منابع
۷۸.....	۹-۱ هندل کردن درخواست های POST
۸۱.....	۹-۲ شرح کد
۸۳.....	۹-۳ خلق منبع دختر
۸۶.....	۹-۴ خلق منابع دختر به همراه والد
۸۸.....	۹-۵ خلق مجموعه ای از منابع
۹۲.....	۹-۶ پیوند مدل در API
۹۵.....	۱۰- کار با درخواست های DELETE
۹۷.....	۱۰-۱ حذف منبع والد به همراه دختر
۱۰۰.....	۱۱- کار با درخواست های PUT
۱۰۰.....	۱۱-۱ بروزرسانی EMPLOYEE
۱۰۳.....	۱۱-۱-۱ شرح متد Update/از کلاس RepositoryBase
۱۰۴.....	۱۱-۲ درج چند منبع در حین بروزرسانی یک منبع
۱۰۷.....	۱۲- کار با درخواست های PATCH
۱۰۹.....	۱۲-۱ اعمال درخواست PATCH به کلیت کارمند
۱۱۴.....	۱۳- اعتبارسنجی
۱۱۵.....	۱۳-۱ اعتبارسنجی در حین خلق منبع
۱۲۰.....	۱۳-۱-۱ تعیین اعتبار نوع عدد صحیح (Int)
۱۲۱.....	۱۳-۲ اعتبارسنجی درخواست های PUT
۱۲۴.....	۱۳-۳ اعتبارسنجی درخواست های PATCH
۱۲۸.....	۱۴- کد آسنکرون
۱۲۸.....	۱۴-۱ برنامه نویسی آسنکرون چیست؟
۱۳۰.....	۱۴-۲ کلمات کلیدی ASYNC، AWAIT و نوع برگشتی
۱۳۱.....	۱۴-۲-۱ شرح واسط IRepositoryBase و کلاس RepositoryBase
۱۳۱.....	۱۴-۳ ویرایش واسط ICOMPANYREPOSITORY و کلاس COMPANYREPOSITORY
۱۳۲.....	۱۴-۴ تغییرات IREPOSITORYMANAGER و REPOSITORYMANAGER

۱۳۳.....	۵-۱۴ ویرایش کنترلر.....
۱۳۸.....	۱۵- فیلترهای اکشن.....
۱۳۹.....	۱-۱۵ پیاده کردن فیلتر اکشن.....
۱۴۰.....	۲-۱۵ حوزه فیلترهای اکشن.....
۱۴۱.....	۳-۱۵ ترتیب فراخوانی.....
۱۴۲.....	۴-۱۵ بهبود کد با فیلترهای اکشن.....
۱۴۳.....	۵-۱۵ اعتبارسنجی با فیلترهای اکشن.....
۱۴۷.....	۶-۱۵ تزریق وابستگی در فیلترهای اکشن.....
۱۵۴.....	۱۶- صفحه بندی.....
۱۵۵.....	۱-۱۶ صفحه بندی (PAGING) چیست؟.....
۱۵۵.....	۲-۱۶ پیاده سازی صفحه بندی.....
۱۵۸.....	۳-۱۶ کوئری عینی.....
۱۶۰.....	۴-۱۶ بهینه سازی راه حل.....
۱۶۴.....	۱-۴-۱۶ توصیه بیشتر.....
۱۶۵.....	۱۷- فیلتر کردن.....
۱۶۵.....	۱-۱۷ فیلترینگ چیست؟.....
۱۶۶.....	۲-۱۷ تفاوت فیلترینگ با جستجو چیست؟.....
۱۶۷.....	۳-۱۷ چگونه در ASP.NET CORE WEB API فیلتر پیاده سازی کنیم؟.....
۱۷۰.....	۴-۱۷ ارسال و تست کوئری.....
۱۷۲.....	۱۸- جستجو کردن.....
۱۷۲.....	۱-۱۸ جستجو چیست؟.....
۱۷۳.....	۲-۱۸ پیاده سازی سرچ در اپلیکیشن خود.....
۱۷۵.....	۳-۱۸ تست کردن پیاده سازی سرچ.....
۱۷۷.....	۱۹- مرتب کردن.....
۱۷۷.....	۱-۱۹ مرتب کردن چیست؟.....
۱۷۹.....	۲-۱۹ طریقه پیاده سازی مرتب کردن در ASP.NET CORE WEB API.....

- ۱۸۱..... ۱۹-۳ پیاده سازی - گام به گام.....
- ۱۸۴..... ۱۹-۴ تست پیاده سازی رویه.....
- ۱۸۵..... ۱۹-۵ بهینه سازی کارکرد مرتب کردن.....
- ۱۸۸-۲۰- شکل دهی به داده.....**
- ۱۸۸..... ۲۰-۱ شکل دهی داده چیست؟.....
- ۱۸۹..... ۲۰-۲ چگونه شکل دهی داده را پیاده سازی کنیم؟.....
- ۱۹۲..... ۲۰-۳ پیاده سازی گام به گام.....
- ۱۹۷..... ۲۰-۴ رفع مشکل سریالی سازی XML.....
- ۱۹۹-۲۱- پشتیبانی از HATEOAS.....**
- ۱۹۹..... ۲۱-۱ HATEOAS چیست و چرا اهمیت دارد؟.....
- ۲۰۰..... ۲۱-۱-۱ پاسخ معمول در زمانی که HATEOAS پیاده شده باشد.....
- ۲۰۰..... ۲۱-۱-۲ لینک چیست؟.....
- ۲۰۱..... ۲۱-۱-۳ مزایا و معایب پیاده سازی HATEOAS.....
- ۲۰۲..... ۲۱-۲ اضافه کردن لینک به پروژه.....
- ۲۰۴..... ۲۱-۳ تغییرات دیگر پروژه.....
- ۲۰۶..... ۲۱-۴ اضافه کردن انواع رسانه سفارشی.....
- ۲۰۷..... ۲۱-۴-۱ رجیستر کردن نوع رسانه سفارشی.....
- ۲۰۸..... ۲۱-۴-۲ پیاده سازی فیلتر اعتبارسنجی نوع رسانه.....
- ۲۱۰..... ۲۱-۵ پیاده سازی HATEOAS.....
- ۲۱۶-۲۲- کار کردن با OPTIONS و درخواست های HEAD.....**
- ۲۱۶..... ۲۲-۱ درخواست OPTIONS HTTP.....
- ۲۱۷..... ۲۲-۲ پیاده سازی OPTIONS.....
- ۲۱۸..... ۲۲-۳ درخواست HEAD HTTP.....
- ۲۱۸..... ۲۲-۴ پیاده سازی HEAD.....
- ۲۲۰-۲۳- سند ریشه.....**
- ۲۲۰..... ۲۳-۱ پیاده سازی سند ریشه.....

۲۲۶	۲۴- نسخه بندی API
۲۲۷	۲۴-۱ نصب پکیج لازم و پیکربندی
۲۲۸	۲۴-۲ یک مثال از نسخه بندی
۲۲۹	۲۴-۲-۱ استفاده از رشته کوئری
۲۳۰	۲۴-۲-۲ استفاده از نسخه بندی URL
۲۳۱	۲۴-۲-۳ نسخه بندی هدر HTTP
۲۳۲	۲۴-۲-۴ نسخه های منسوخ شده
۲۳۳	۲۴-۲-۵ استفاده از قواعد قراردادی
۲۳۴	۲۵- کش کردن
۲۳۴	۲۵-۱ کش کردن چیست
۲۳۵	۲۵-۱-۱ انواع کش
۲۳۶	۲۵-۱-۲ خصیصه کش پاسخ
۲۳۶	۲۵-۲ اضافه کردن هدر کش
۲۳۷	۲۵-۳ اضافه کردن ذخیره-کش
۲۴۰	۲۵-۴ مدل انقضاء
۲۴۲	۲۵-۵ مدل اعتبارسنجی
۲۴۴	۲۵-۶ پشتیبانی از اعتبارسنجی
۲۴۵	۲۵-۶-۱ پیکربندی
۲۴۷	۲۵-۷ استفاده از ETAG و اعتبارسنجی
۲۵۰	۲۶- محدودیت ریت و گلوگاه
۲۵۱	۲۶-۱ پیاده سازی محدودیت نرخ
۲۵۵	۲۷- JWT و آیدنتیتی
۲۵۶	۲۷-۱ پیاده سازی آیدنتیتی در پروژه ASP.NET CORE
۲۵۸	۲۷-۲ خلق جداول و درج نقش ها
۲۶۰	۲۷-۳ خلق کاربر
۲۶۴	۲۷-۴ مروری بر تصویر اصلی
۲۶۵	۲۷-۵ معرفی JWT

۲۶۶JWT پی‌کربندی
۲۶۹ ۷-۷ محافظت از نقطه پایانی
۲۷۰ ۸-۲۷ پیاده سازی تعیین هویت
۲۷۸ ۹-۲۷ تایید مجوز مبتنی بر نقش
۲۸۰ مستند کردن API با SWAGGER
۲۸۱ ۱-۲۸ SWAGGER چیست؟
۲۸۱ ۲-۲۸ تلفیق SWAGGER در پروژه
۲۸۵ ۳-۲۸ اضافه کردن پشتیبانی مجوز دسترسی
۲۸۸ ۴-۲۸ بسط پی‌کربندی SWAGGER
۲۹۲ انتقال به IIS
۲۹۳ ۱-۲۹ خلق فایل های انتشار
۲۹۴ ۲-۲۹ باندل میزبانی سرور ویندوز
۲۹۵ ۳-۲۹ نصب IIS
۲۹۷ ۴-۲۹ پی‌کربندی فایل محیطی
۲۹۹ ۵-۲۹ تست استقرار اپلیکیشن

۱- پیکربندی پروژه

پیکربندی^۱ پروژه ها در NET Core. با آنچه که عادت داشتیم در پروژه های دات نت فریم ورک انجام می دادیم، تفاوت زیادی دارد. حالا مجبور نیستیم از فایل web.config استفاده 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**^۲ یا متد افزونه برای دستیابی به این هدف را بحث می کنیم.

البته قبل از هر چیزی لازم است که یک پروژه جدید خلق کنیم. بنابراین ابتدا به خلق پروژه در **ویژوال استودیو**^۳ می پردازیم.

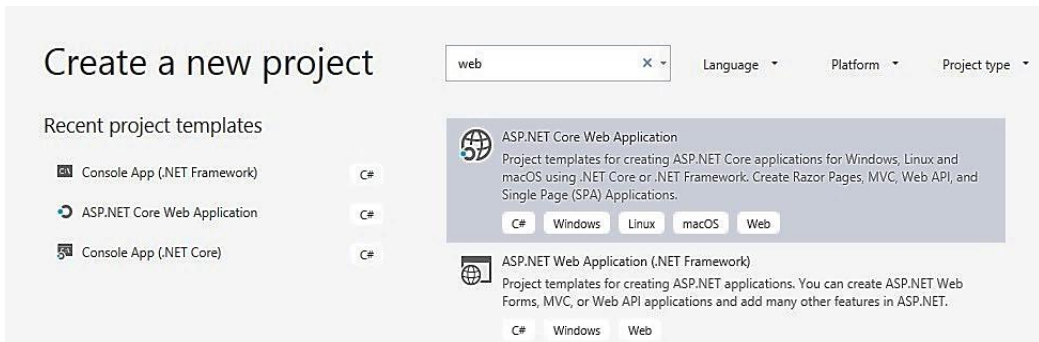
^۱ Configuration

^۲ Extension Methods

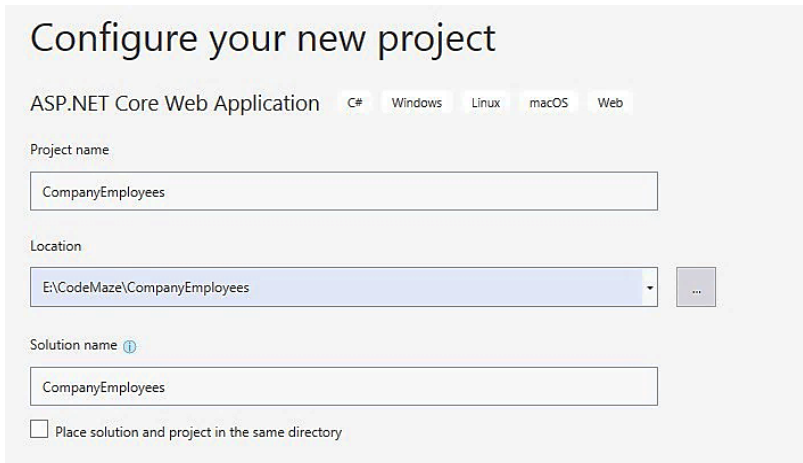
^۳ Visual Studio

۱-۱ خلق پروژه جدید

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

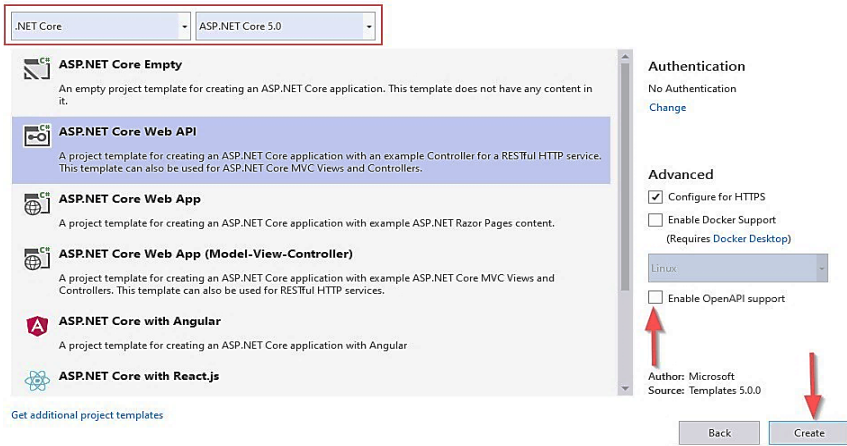


حالا در این دیالوگ نام و محل ذخیره پروژه خود را مشخص می کنیم:



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

Create a new ASP.NET Core web application



۲-۱ پیکربندی فایل launchSettings.json

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

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

فعلا خصیصه launchBrowser را به false تغییر می دهیم تا از باز شدن مرورگر وب در هر بار استارت 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 Startup.cs و Program.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 کیسولی 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^۱ را ست میکند. این حقیقت که لاگر قبلا و در فرآیند بوت استرپ کردن^۲ پیکربندی می شود بدین معنی است که حالا می توانیم مسائلی که در طی بوت استرپ کردن رخ می دهد را نیز یادداشت کنیم، چیزی که در نسخه های قبلی اندکی سخت بود.

1 Logger Configuration

2 Bootstrapping Process