درس 14

در این درس ابتدا در کلاس program سیستم مسیریابی رو تنظیم می کنیم



قسمت هایی که با رنگ زرد مشخص شدن برای مسیر یابی ضروری هستند.

بعد از این کار یک کنترولر میسازیم و با دادن روت ان را اجرا می کنیم.



قسمتی که با رنگ زرد مشخص شده است روت است و با همین ادرس می توان به این کنترولر دست پیدا کرد.

درس 15 تنظیم مسیر یابی کنترولر ها

برای تنظیم مسیر یابی کنترولر ها در جلسته قبل روت رو اکشن بود که کارایی نداشت. در این جلسه روت رو کنترولر هست. به دو روش می توان گذاشت که با توجه به تجریه استاد مورد سبز بهتر است اما روش زرد رنگ هم درست است.



درس 16

معرفی و نصب Postman

درس 17 ساخت DTO

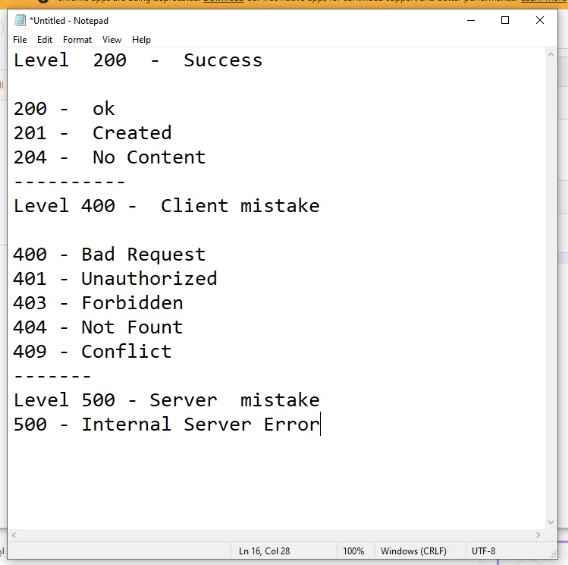
در این جلسه DTO ها رو ساختیم و مدل city بعد این کار موردی که جلسه پیش بود و اگر دو کنترلر با اندیس get داشتیم چطور این دو تا رو از هم متمایز کنیم را پاسخ دادیم.



باتوجه به قسمت زرد و سبز می توان بادادن اندیس ورودی روت این دو تایع Get ر ا از هم متمایز کنیم.

درس 18

کد های خطا در سمت سرور و کلاینت و موفقیت



درس 19

پیاده سازی کدها و کدها ی خطا و مشاهده و استفاده از IActionResult



درس 20

ایجاد مدل جدید و مقدار دهی به ان



و ایجاد ارتباط با مدل قبلی



قسمت رنگی یک پراپرتی تعریف شده که در همین سمت کلاینت مقدار دهی می شود و در دیتا بیس فیلد ندارد. فقط set دارد در سمت کلاینت محاسبه می شود و مقدار ان برگشت داده می شود. فقط readonly هست.

درس 21

در این قسمت برای pointofinterest کنترولر می نوسیم.



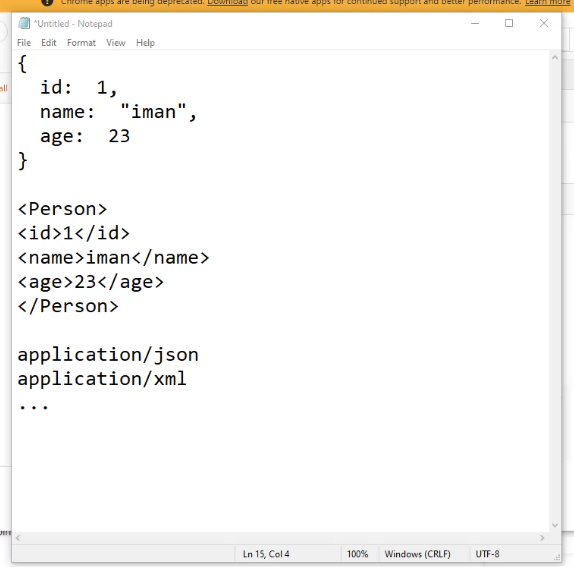
برای کل کنترولر از روت زرد رنگ استفاده می کنیم. با توجه به اینکه زیاد نمی خواهیم روت ایجاد کنیم روت را در ادامه Cities می نوسیم.

در قسمتی که سبز است روت ادامه روت زرد می شود.



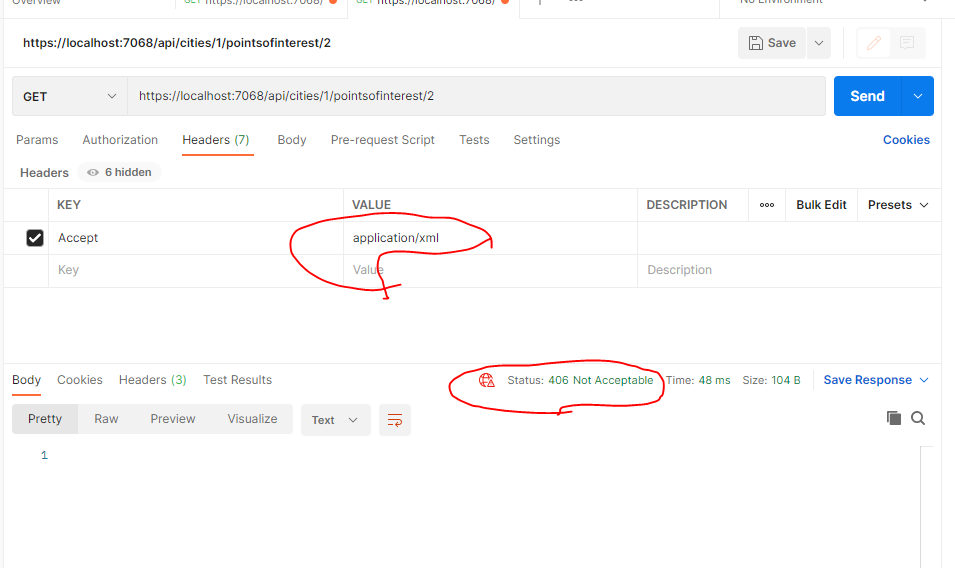
درس 22

در این جلسه در مورد فرمت وردی وخروجی صحبت شد. دو فرمت معروف JSON و XML بررسی شد. ورودی و خروجی باید فرمت مورد تایید خود را در هدر معرفی کنند.



درس 23

این جلسه ادامه جلسه قبل هست و تعیین فرمت وردی وخروجی بررسی می شود. این کار در قسمت Heder صورت می گیرد





با اضافه کردن کد بالا به Program زمانی که فرمت درخواست پشتیبانی نشود کد خطای 406 نمایش داده می شود.

برای پشتیبانی از فرمت XML از باید کد زیر را اضافه کنیم.



فرمت خروجی و وردی به صورت پیش فرض json است برای تغییر این پیش فرض از کدهای زیر استفاده می کنیم:



* 1. درس 24 دانلود فایل از طریق WebApi

برای دانلود فایل ابتدا یک کنترولر میسازیم :

اکشن دانلود را میسازیم از نوع GET هست. برای شناسایی فایل دانلودی به عنوان ورودی یک ایدی به فایل دانلود شده میدهیم.

1. ابتدا مسیر فایل را می دهیم. رنگ قرمز. ازانجای که الان فایل در روت است فقط اسم فایل وارد شده اگر داخل فولدر بود آدرس ان را می دادیم.
2. از قسمت زرد رنگ زرد برای چک وجود فایل استفاده می کنیم که در صورت نبودن فایل خطا برگرداند.
3. اگر فایل وجود داشت از خط سبز رنگ برای تبدیل به بایت کردن فایل استفاده می کنیم.
4. برای دانلود باید نوع فایل را به خروجی بدهیم در aspcore ما از کتابخانه فیروزه ای برای فهمیدن نوع فایل رنگ استفاده می کنیم. البته باید این کنابخانه را به سرویس های معرفی کنیم و اینجکت می کنیم رنگ خاکستری.



1. در قسمت صورتی رنگ هم فایل را برمی گردانیم. بایت شده فایل دانلود سپس نوع فایل و اسم فایل.

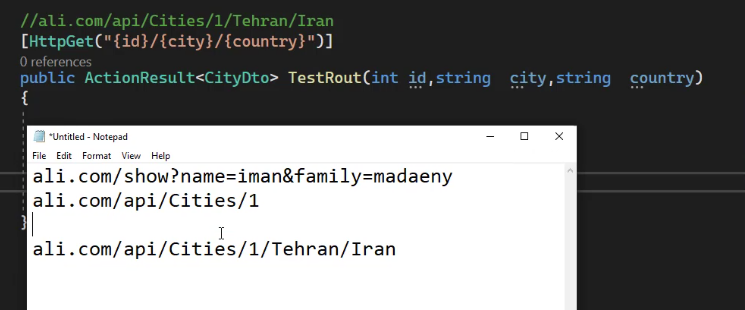


* 1. درس 25: روش های ارسا ل اطلاعات به API

1. استفاده از کوئری استرینگ ها: در این روش بعد از ? مطالب ارسالی به صورت json ارسال می شوند.



1. گرفتن اطلاعات از روت. در این مورد با قرار دادن اطلاعات در روت داده ها به api ارسال می شوند. دقت کنید روت در هنگام ران تایم قابل تغییر نیست. ولی کوئری استرینگ قابل تغییر است.



* 1. درس 26 ثبت اطلاعات از طریق frombody

در این قسمت Post بررسی می شودو اطلاعا ت FromBody به مرکز منتقل می شود.



* 1. جلسه 27 اعتبار سنجی مقادیر وردی API

در این قسمت با معرفی کتابخانه زیر اعتبار سنجی انجام شد.



* 1. درس 28 ویرایش اطلاعات از طریق HttpPut

در این جلسه از PUT برای ویرایش اطلاعات استفاده می کنیم.

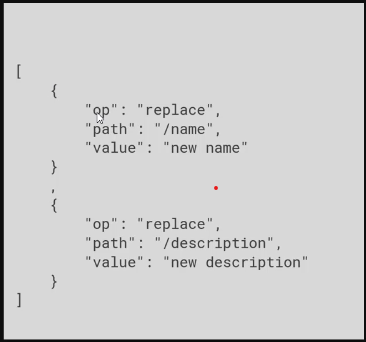


این اپ دیت کلی ابجکت هست و در ان تمام فیلدهای ابجکت به روز می شود.

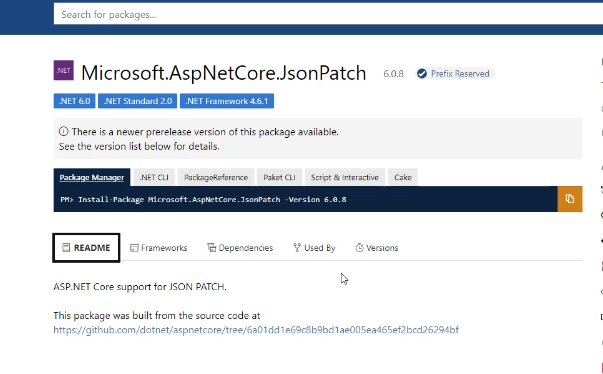
* 1. درس 29 معرفی HttpPatch برای ویرایش جزئی

برای ویرایش جزئی یک شی به جای استفاده از Put که تمام پراپرتی ها را اپدیت می کند از Patch استفاده می کنیم. در این روش بجای اینکه کل شی اپدیت شود فقط جزئی که می خواهیم را به روز می کنیم.

همانطور که از شکل زیر مشخص هست op عملیات را مشخص می کند که در اینجا جایگزینی هست path پراپرتی یا کلید را مشخص می کندو value مقدار جدید را مشخص می کند اگر قرار بود کلید دیگری تغییر بکند با کاما جدا می کنیم و json جدید را می نوسیم:



برای استفاده از Patch بایداز پکیج زیر استفاده کنیم:



* 1. درس 30 نصب بسته ها مورد نیاز Patch و تغییر فرمت از Json.Net

در این قسمت دو پکیج زیر را به لایه Api اضافه می کنیم:



سپس در فایل program.cs این سرویس را اضافه می کنیم:



* 1. درس 31پیاده سازی متد Patch

قسمت زرد رنگ ورودی های متد هستند و قسمت سبز مربوط به ورودی Json هستند که با پکیجی که نصب کردیم در واقع Json ورودی را به مدلی که می خواهیم تغییر میدهد.

بعد از پیدا کردن شهر ازId در قسمت آبی رنگ PointOf.. را پیدا می کنیم. از انجایی که این مورد را از حافظه خوانیدم در اسم ان از FromStore استفاده می کنیم که مشخص کنیم این از حافظه خوانده شده و با Object که از ورودی می اید وبرای Patch کردن متفاوت است.

در قسمت صورتی یک مدل میسازیم که قرار است تغییرات روی ان اعمال شود. دقت کنید که مقدار اولیه ان را از FromStore می دهیم و می سازیم. از انجایی که FromStore از دیتا بیس خوانده شده و ممکن است به صورت ناخواسته و بدون اعتبار سنجی اطلاعتی را با Update به دیتابیس اعمال کنیم . این مدل را میسازیم.

در قسمت قرمز ورودی Patch را روی مدلی از روی اصلی ساختیم اعمال می کنیم ToPatch . برای اعتبار سنجی Modelstate را هم میدهیم. ازانجای که اگر Modelstate درست نباشد هیچ خطای نمی دهد (تابع Apply)در ادامه خودمان مشخص می کنیم در صورت درست نبودن ان خطا بدهد.

در اخر هم در صورت درست بودن همه موارد روی مدل اصلی که از روی دیتابیس خوانیدم اعمال می کنیم.

در شکل زیر هم روش نوشتن تغییرات برای Patch می نویسیم. Path فیلد مورد نظر را مشخص می کند OP عملیات را مشخص می کند و Value مقدار جدید را مشخص می کند . دقت کنید که ابتدا از[] براکت استفاده کردیم و به ما این امکان را می دهد که اگر قصد تغییر فیلد دیگری را داشتیم در ادامه اضافه کنیم.





* 1. درس 32 اعتبار سنجی مدل ها در HttpPatch

تفاوت Update عادی با Put و Patch این است که در روش عادی اگر به یک فیلد مقداری ندهیم (قصد تغییر ان را نداشته باشیم)، ان فیلد در زمان Update با Null جایگرین می شود ولی Patch فقط مقدار فیلد مورد نظر تغییر می کند و بقیه به همان صورت قبل باقی می مانند.

در قسمت زرد رنگ درست بودن مدلی که قرار است به مدل اصلی FromeStore بررسی شده است. قسمت قبلی مدل که از وردی امده است (PatchDocument) را بررسی می‌کند و در ادامه مدل که قرار است به مدل که روی دیتا بیس برود بررسی می شود.



* 1. درس33 پیادهسازی HTTPDELETE روی پروژه

در این قسمت متد Delete را پیاده کردیم. به صورت زیر"



* 1. درس 34معرفی Dependencyinject و IOCدرAsp core

صحبت در مورد DI واینکه با استفاده از تزریق وابستگی سرویس ها ولایه ها رااز هم جدا می کنیم.

قسمت اضافه کردن سرویس ها که به کانتینر معرف است و در این قسمت با معرفی سرویس و در ادامه با تزریق سرویس در مکان هایی که نیاز داریم از ان استفاده می کنیم.

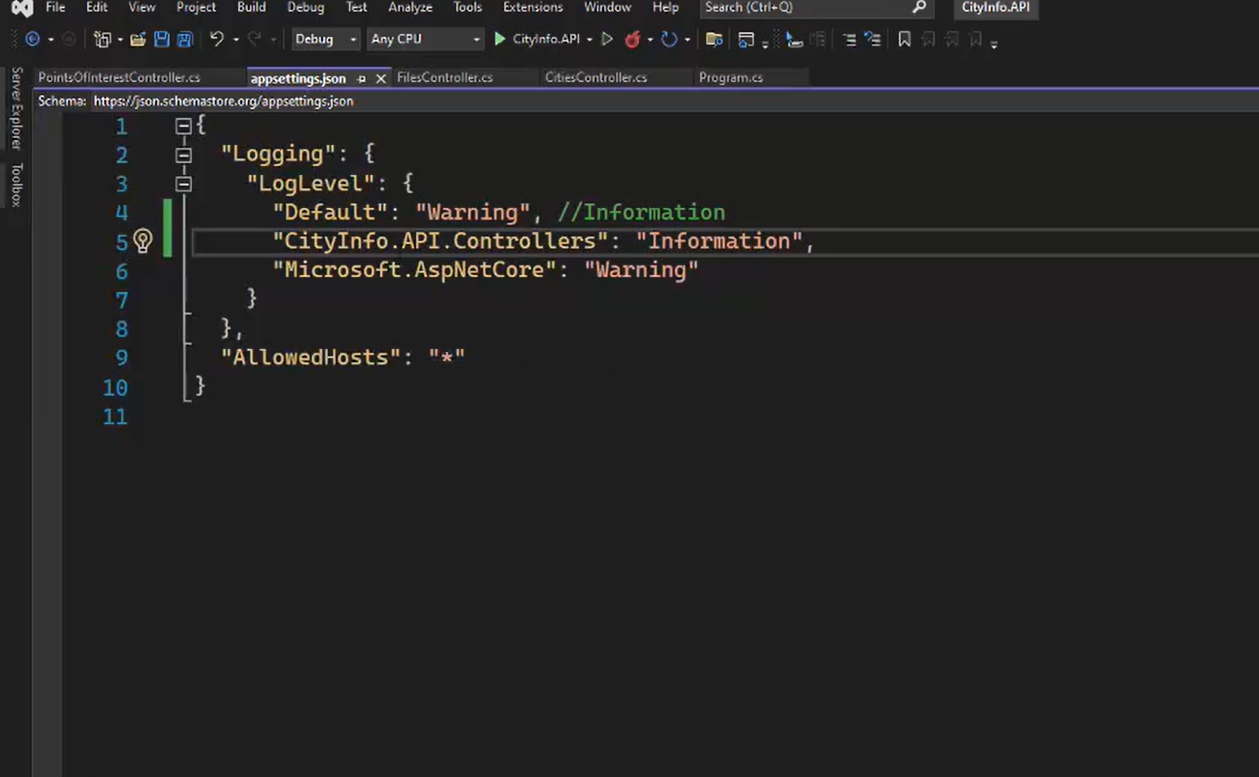
* 1. درس 35 استفاده از سیستم Logging با استفاده از Injection

سرویس Loging به صورت توکار در .NET قرار داده شده است برای همین نیاز نیست سرویس در فایل Program.cs معرفی شود. فقط کافی است در هر کنترلر تزریق شود. در این مورد تزریق به صورت کانستراکتور (قسمت زرد رنگ )است. در قسمت سبز مشخص می کنیم که اگر ورودی Logger Null بو یک خطا بده. رنگ صورتی روش استفاده از را نشان می دهد.

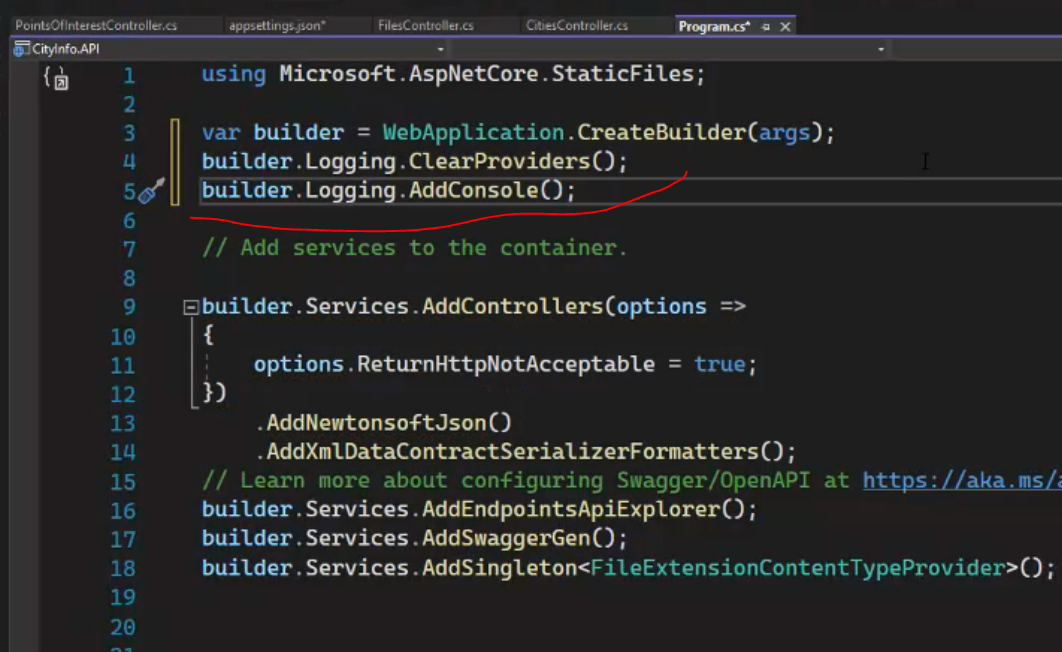


قسمت زیر هم تنظیمات Logger را در کل پروژه نشان می دهد.می توان تنظیمات ان را برای کل پروژه تغییر داد. برای یک کنترلر خاص هم می توان تنظیم کرد.





برای دسترسی و کنترل Logger از طریق هسته .Net هم می توان به صورت زیر در فایل Program.cs اقدام کرد.



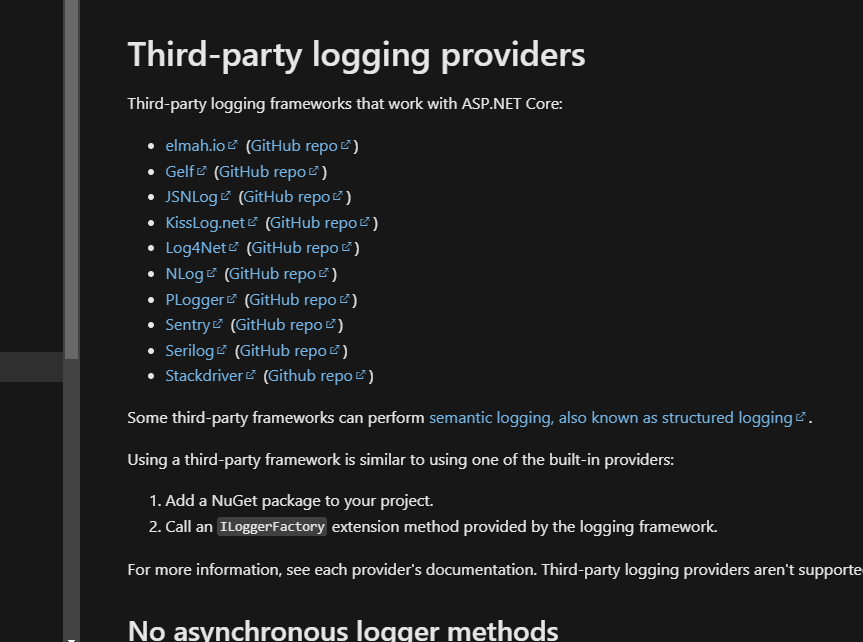
* 1. درس 36 لاگ زدن خطا ها در API

در این قسمت به لاگ زدن خطا ها پرداختیم که با عنوان زرد رنگ لاگ خطا ثبت می شود. قسمت سبز رنگ متنی است که در سیستم لاگ نشان داده می شود و قسمت ابی متنی است که در صفحه برای کاربر نشان داده می شود.

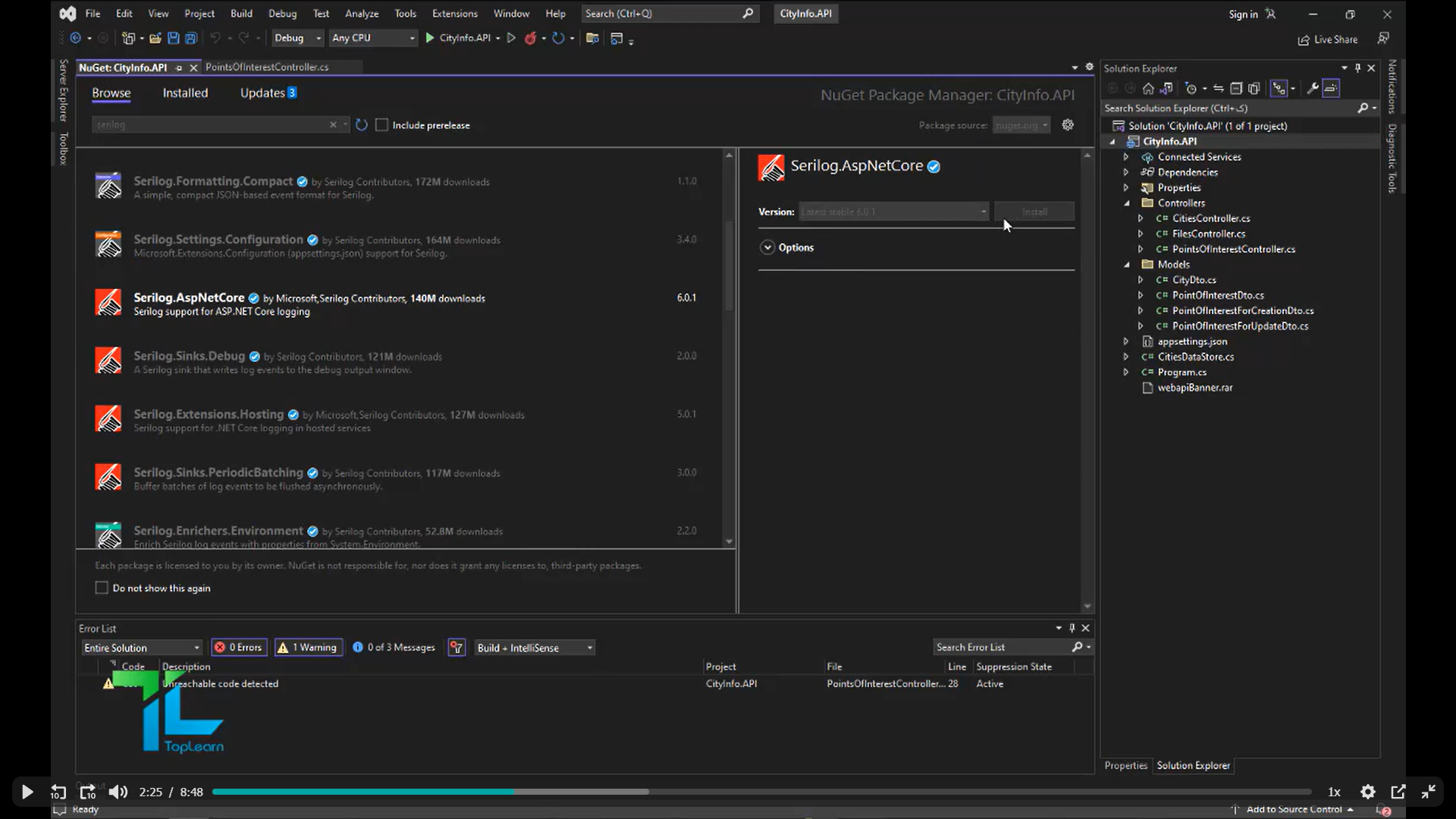


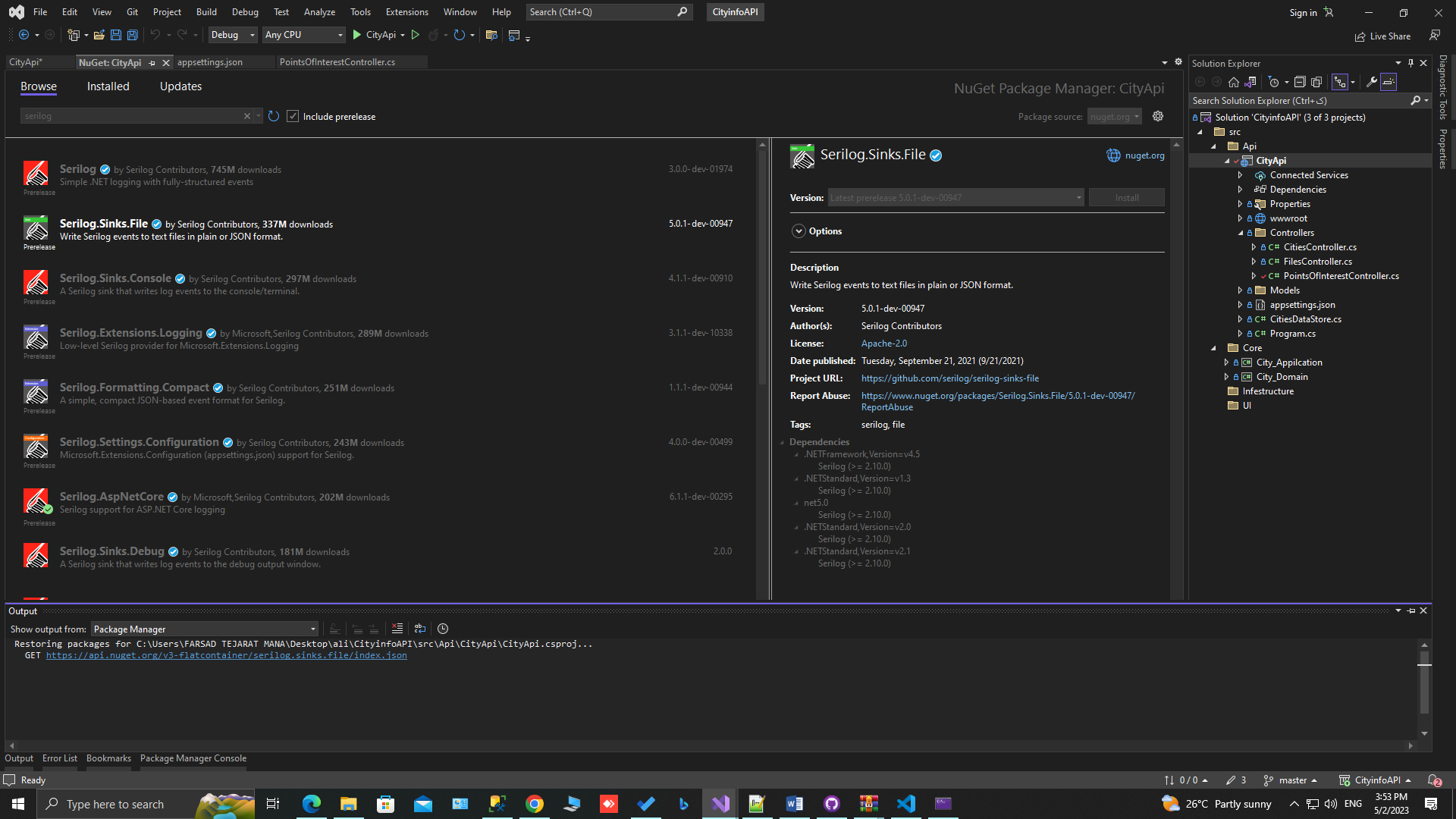
* 1. درس 37لاگ زدن Logger در فایل WebApi

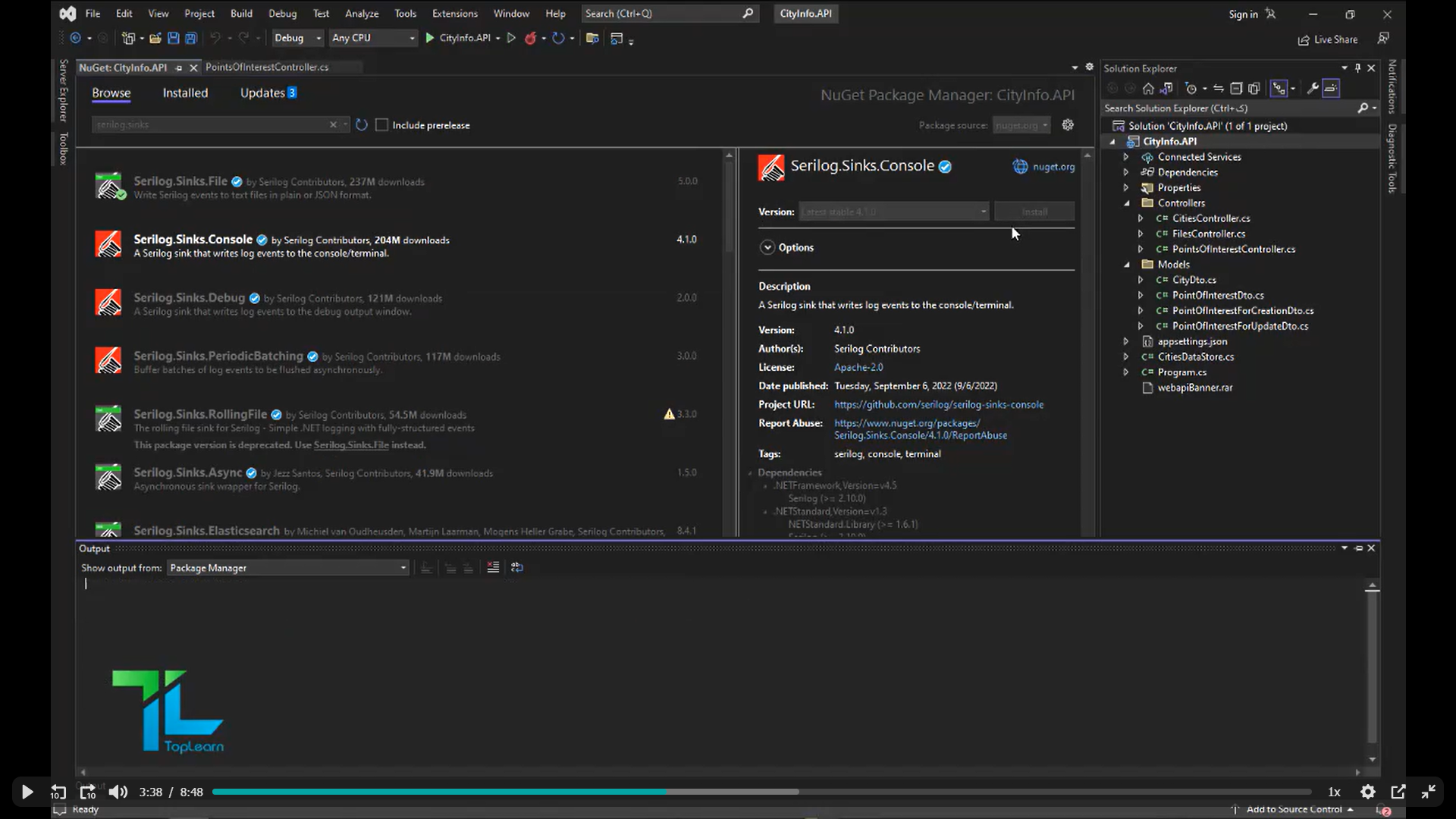
برای نگه داری لاگ های برنامه از Privider های دیگر استفاده می کنیم.



برای مثال در پروژه از Logger Serilogاستفاده می کنیم. برای استفاده از این لاگر سه پکیج زیر را دانلود می کنیم:







بعد از نصب پکیج ها لاگر ار در فایل Program.cs کانفیگ می کنیم:

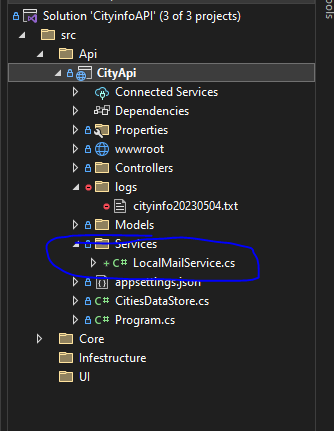
لاگر را به صورت زیر تعریف می کنیم. قسمت زرد رنگ مکان ذخیره لاگ ها و اینک روزانه درست شود را تعیین می کنیم. در قسمت صورتی رنگ هم به کور دستور می دهیم که از این لاگر استفاده کند.



در نهایت بعد از لاگ ها را می توانیم با این Provider مدیریت کنیم.

* 1. درس 38 ساخت سرویس ارسال ایمیل در WebApi

برای ایجاد سرویس ایمیل فولدر Services را ایجاد می کنیم واز کلاس LocalMailService را هم ایجاد می کنیم:





* 1. درس 39 ثبت سرویس ایمیل در Container و پیاده سازی DI

در این قسمت سرویس ایمیل را در فایل Program.cs معرفی و کانفیگ می کنیم. قسمت زرد رنگ این معرفی را نشان می دهد.



سرویس ایمیل را به صورت زیر Inject می کنیم:

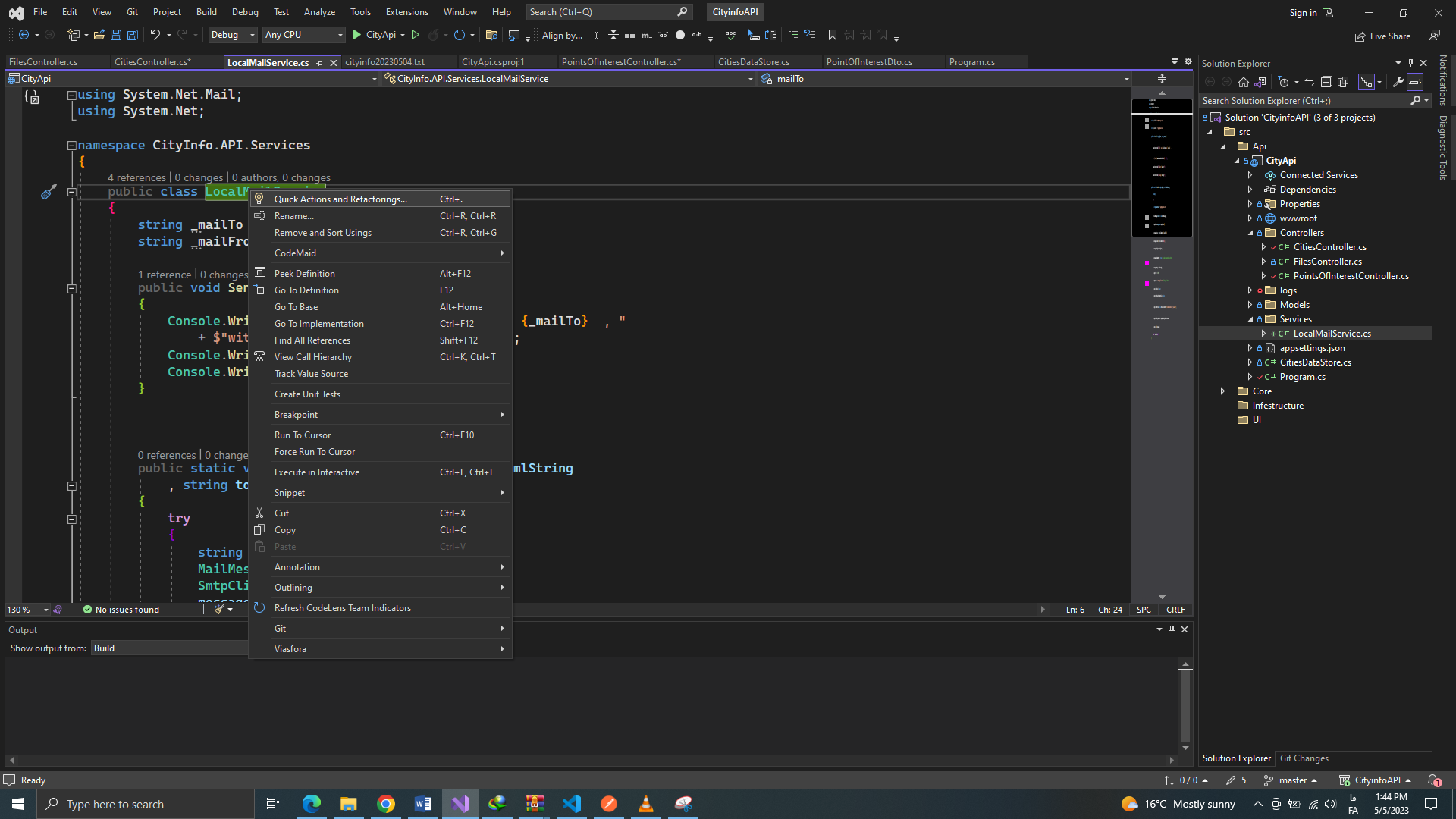


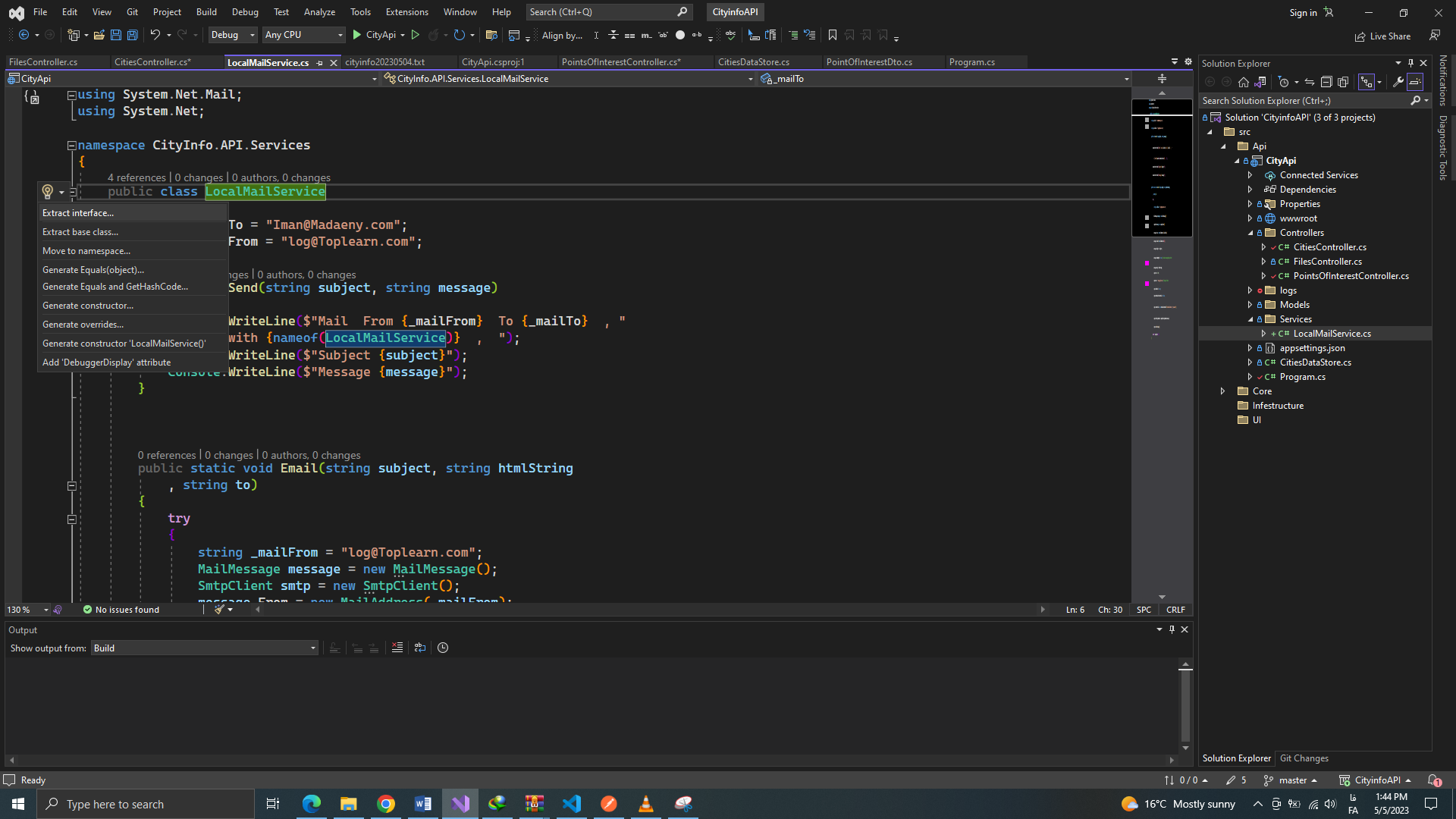
و در داخل متدها به صورت زیر استفاده می کنیم:



* 1. درس 40 بهبود ساخت و تعریف سرویس ها در WebApi

در این درس برای استفاده بهتر از سرویس ها و رعایت عدم وابستگی به سرویس ها از Interface یا ..استفاده می کنیم. برای ساخت اینترفیس به صورت زیر عمل می کنیم:





اینترفیس به صورت زیر ساخته می شود:



بعد از ساخت اینترفیس سرویس دیگری را برای ایمیل معرفی می کنیم:



و فایل Program.cs را هم به صورت زیر تغییر میدهیم:



DI را هم تغییر میدهم:



* 1. درس 41ساخت و ایجاد سرویس ها با Service Interface

در این درس سرویس های ایمیل خود را در Program.cs کانفیگ می کنیم. قطعه کد پایین هم برای حالت دولوپ و سرور هست که زمانی که داریم در حالت دیباگ کار می کنیم از یک سرویس استفاده می کنیم و در حالت سرور از سرویس دیگر.



در ادامه برای تمرین کلاس CitiesDataStore که وظیفه ذخیره اطلاعات را دارد به صورت سرویس می نویسیم. چون کلاس فقط یکبار در زمان اجرای پروژه ایجاد می شود از Singleton استفاده می کنیم:



در ادامه در هر کنترولر این سرویس را انجکت می کنیم:



* 1. درس 42 کار با فایل های Configuration در .net

در این جلسه فایل appsetting.json را بررسی می کنیم. مواردی که قبلا در مورد ایمیل هست را به این فایل منتقل می کنیم که اگر خواستیم تغییری در سرویس ایمیل بدهیم دیگر نیازی استاپ کردن پروژه نباشد. این فایل به همین صورت که در اینجا است در فایل پابلیش قرار می گیرد. مواردی مثل کانکشن استرینگ و موارد ایمیل در این فایل قرار می گیرند:



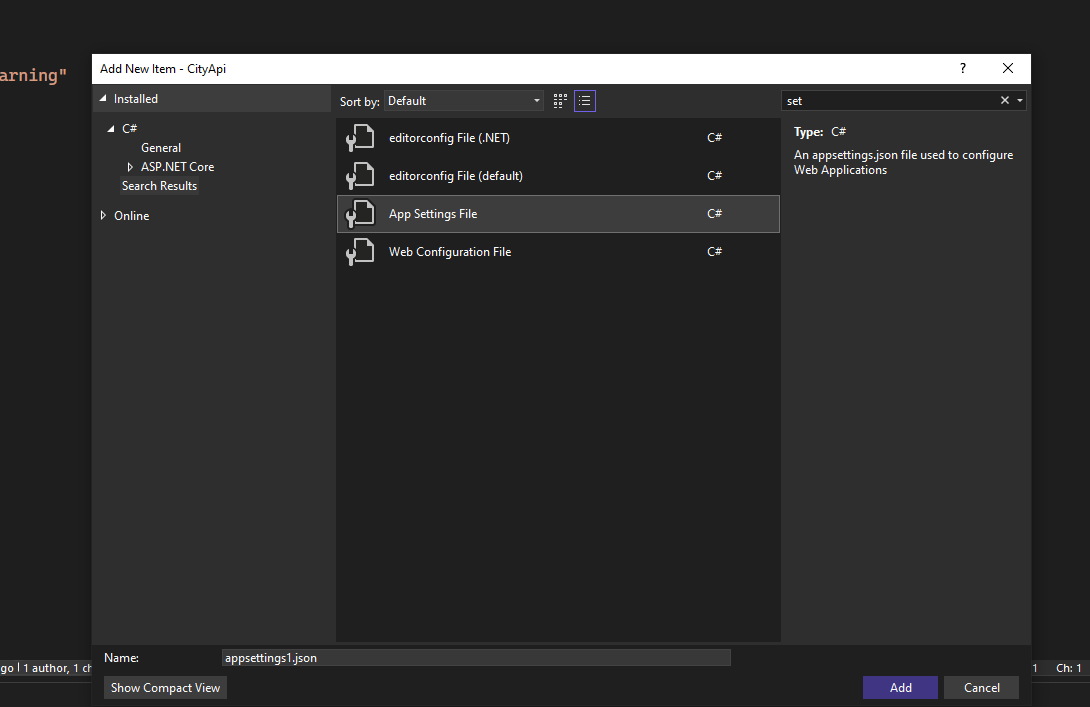
کلید اصلی با رنگ سبز مشخص شده و رنگ ابی کلید های فرعی هستند.



برای خواند فایل appsetting باید از IConfiguration استفاده کنیم. این سرویس به صورت توکار در دات نت هست و نیازی نیست که معرفی کنیم فقط اینجکت می کنیم. که طریقه نوشتن ان همان طور که مشخص شده بعد از کانفیگوریشن از براکت باز و ابتدا کلید کلی و در ادامه کلید فرعی را می نویسیم.

* 1. درس 43 استفاده از چند کانفیگ Json در Asp

در این جلسه سعی می کنیم که از یک فایل appsetting.json دیگر برای کانفیگ استفاده کنیم.علت این کار هم این است که مثلا فرض کنید که در زمان توسعه و پاپلیش نیاز به تنظیمات متفاوت داشته باشیم از این روش استفاده می کنیم:



فایل بالا را ایجاد می کنیم و نام ان را appsettings.Production می گذاریم. به طور مثال این تنظیمان را در ان قرار می دهیم:



حالا تنظمات اجرای پروژه را به حالت Production می بریم و در این حالت تنظیمات از این فایل خوانده می شوند.

* 1. درس 44 معرفی EFCore و Sqlite

معرفی دیتا بیس Sqlite انجام شد.

* 1. درس45 ساخت Entity های EF Core

در این قسمت انتیتی هر کدام از جداول مورد نیاز در دیتابیس خودمان را میسازیم. برای جدول City به صورت زیر عمل می کنیم. فیلد ایدی را می نویسیم و دو اتریبیوت Key و سبز رنگ را هم می گذاریم. سبز برای درست کردن فیلد ایدی به صورت خودکار است. قسمت صورتی برای اینکه وارد کردن نام شهر را اجباری کنیم نام را در کانستراکتور کلاس می گذاریم که هروقت کلاس نمونه سازی شد تا موقه ای که نام نگیرد کلاس نمونه سازی نشود. و قسمت خاکستری برای ایجاد ارتباط با جدول دیگر است . نمونه سازی از این لست نیاز است واگرنه خطا میدهد.



* 1. درس 46 ایجاد رابطه بین Entity های EF Core

در این قسمت PointOfInterest را میسازیم. قسمتی که زرد است برای مشخص کردن این است که زمانی که ریلیشن برقرار شد ایدی شهر کجا قرار بگیرد. کلید اصلی این رابطه به ستون CityId قرار بگیرد.



* 1. درس 47 ساخت DbContext و اضافه کردن به Container

در این جلسه DbContext تعریف می شود و به پروژه معرفی می شود:

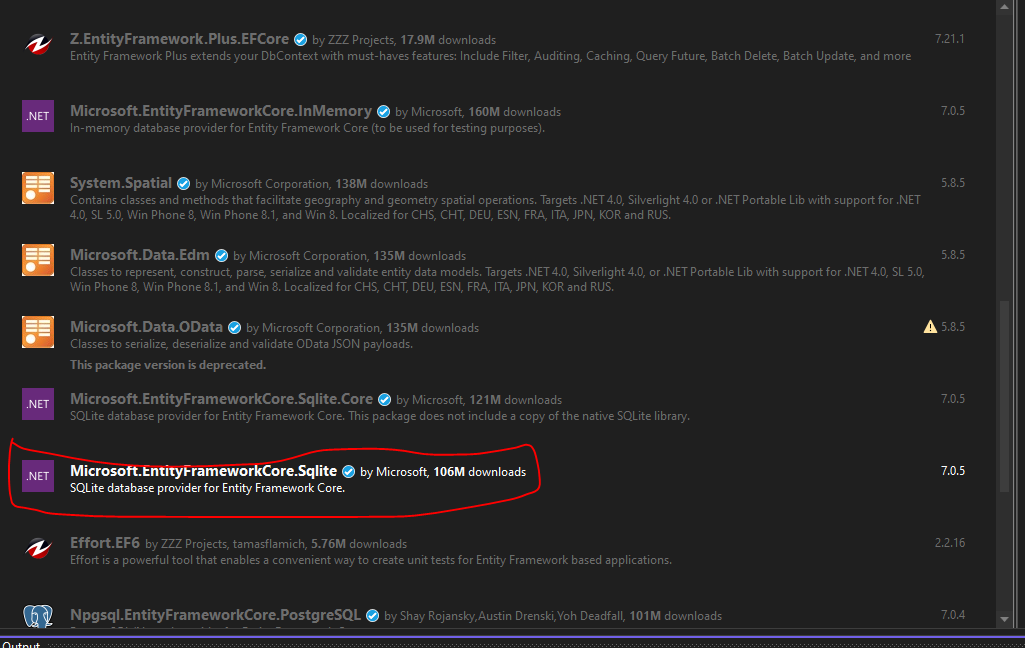


به فایل Program.cs هم به صورت زیر معرفی می شود:



* 1. درس 48 اتصال DbContext به بانک

برای استفاده از دیتابیس Sqlite باید پرووایدر ان را نصب کنیم.



برای دادن کانکشن استرینگ از دو روش می توان استفاده کرد یکی استفاده از :



با با استفاده از Program.cs است:



اگر از روش دوم استفاده کریدم نیاز هست چند مورد به برنامه اضافه کنیم:



باید یک کانستراکتور بسازیم و Options که در Program هست را به Cityinfo و سپس با استفاده از base به کلاس پدر که DbContext است بدهیم.

* 1. درس 45 امنیت رشته اتصال در AspCore

در این جلسه قرار دادن رشته اتصال در فایل Appsetting را بررسی می کنیم



و در فایل Program به صورت زیر آدرس می دهیم:

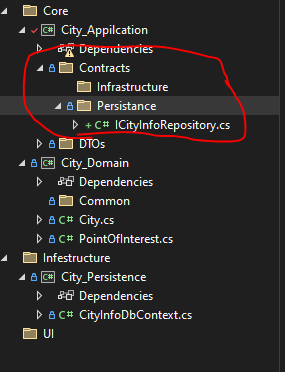


* 1. درس 50 بررسی Repository Pattern یا همان الگور ریپوزیتوری

معرفی و بررسی الگوی ریپازیتوری

* 1. درس 51 ساخت Repasitory در WepApi

در این جلسه برای کلاس City یک اینترفیس میسازیم.



دقت کنید که می توان برای هر متد روش Async و معمولی نوشت.



* 1. درس 52 ساخت متدهای Async در WebApi

در این جلسه اینترفیس و ریپازیتوری را کامل می کنیم:



و خود ریپازیتوری به صورت زیر پیاده می شود. قسمتی که زرد رنگ مشخص شده زمانی است که اگر Null بود یک خطا پرتاب کند که نام را بدهد که بعدا کنترل کنیم:



در نهایت در Program این سرویس را معرفی می کنیم:



* 1. درس 53 استفاده از Repository درکنترل های API

در این جلسه کنترلر ها برای استفادهاز Repository اماده می کنیم. از انجایی که فیلد هیا دیتابیس و Dto متفاوت هستند مجبور شدیم یک Dto جدید تعریف کنیم CityWithoutPointOfInterest :



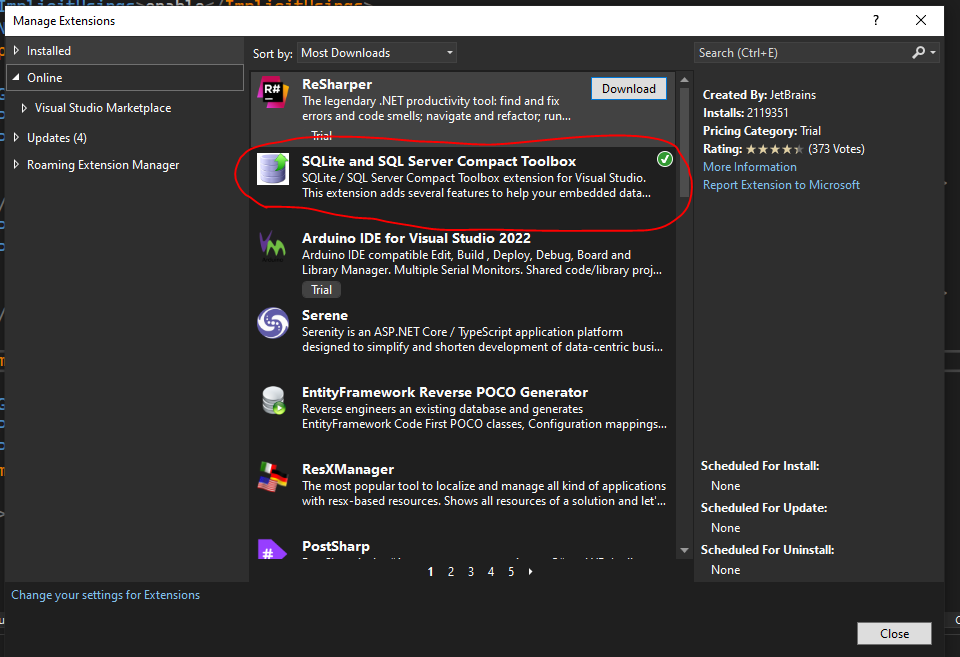
* 1. درس 54 ساخت دیتا بیس با استفاده از Migration

در این درس با اضافه کردن دو پیکیج زیر و دستورات Update-database و Add-migration دیتابیس را میسازیم.



* 1. درس 55 نصب و استفاده از افزونه CompactToolbox در V2022

د ر این جلسه افزونه زیر را نصب می کنیم



با این افزونه می توانیم دیتابیس Sqlite را ببینم.

در ادامه به کلاس Poin.. فیلد Description را اضافه می کنیم.

* 1. درس 56 وارد کردن SeedDate در EFCore

برای اینکه در موقع شروع پروژه دیتابیس داده داشته باشد از SeedData استفاده می کنیم:



بعد از وارد کردن دیتا اولیه از Migration استفاده می کنیم.

* 1. درس 57 نصب و کانفیگ Automapper در Asp.netCore

در این جلسه پکیج های مورد نیاز برای Automapper را نصب می کنیم.



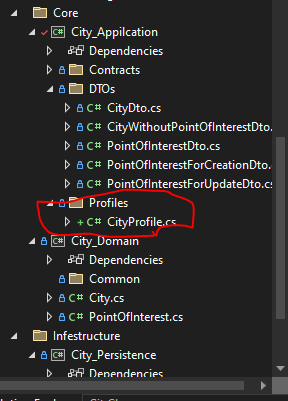
سپس در فایل Program.cs کانفیگ می کنیم:



* 1. درس 58 Map کردن Entity ها با استفاده از AutoMapper

در این جلسه AutoMapper را پیاده سازی می کنیم.

ابتدا فولدر Profile را میسایم:



داخل کلاس CityProfile به صورت زیر عمل می کنیم:



کلاس از Profile ارث بری می کند. سبز کلاس اصلی و ابی کلاسی که میخواهیم به ان تبدیل شود.

برای استفاده هم به صورت زیر عمل می کنیم. ابتداAutomappr را اینجکت می کنیم و سپس به صورت قسمت رنگی سبز و ابی استفاده می کنم:



* 1. درس 59 واکشی دیتا از ریپازیتوری ها در WebApi

در این جلسه متد دیگر کنترولر City را تکمیل می کنیم. دقت کنید از انجایی که خروجی تابع هم ممکن است CityDto و یا CityWithoutPointOfInterestDto باشد برای همین خروجی متد به صورت IActionResult تعریف شده است.

Mapper هم به صورت زیراپدیت می شود:





دقت کنید برای وارد کردن دو مقدار در ورودی تابع از ? استفاده می کنیم:



* 1. درس 60 درج زیرمجوعه ها و روابط جداول در AutoMapper

ابتدا برای PointOf.. هم Automapper را کانفیگ می کنیم:



بعد از این کار کنترولر Poinof… را کامل می کنیم:



برای چک کردن وجود یک ایدی شهر یک متد به ریپازیتوری اضافه می کنیم:



* 1. درس 61 تکمیل اکشن های واکشی اطلاعات

بقیه اکشن های Point.. را کامل می کنیم:



* 1. درس 62 افزودن اطلاعت به بانک با Repository

در این قسمت متد Post را کامل می کنیم.

Automapper به صورت زیر کامل می شود:



بعد از کار باید متد Add را به ریپازیتوری اضافه کنیم:



برای ذخیره تغییرات در بانک نیاز هست که متد SavecgngesAsync نوشته شود. این متد را به صورت جدا در ریپازیتوری می نویستم.



و در ادامه متد Post را کامل می کنیم. برای ذخیره مقدار ابتداان را به صورت Entity بانک تبدیل می کنیم (رنگ ابی). بعد از ذخیره به متد GetPointofInterest بر می گردیم. برای این کار با توجه به ورودی های این متد ابتدا داد ه این که ذخیره کردیم و اینبار ID دارد (به صورت خودکار ایجاد میشود) را به صورت Dto تبدیل می کنیم (رنگ زرد). از Return CreatedAtRoute استفاه می کنیم. ورودی ها را هم به صورت رنگ سبز وارد می کنیم.



* 1. درس 63 ویرایش اطلاعات با استفاده از Repository

متد Edit را با ریپازیتوری میسازیم.

ابتدا پروفایل Automapper را میسازیم:



سپس متد را به صورت زیر پیاده می کنیم. قسمت ابی رنگ از Mapper استفاده کردیم که در ان مقادیر ورودی pointofinterst داخل point جایگزاری می شوند.



* 1. درس 64 ویرایش جزئی با استفاده از Repasitory

برای اپدیت با Patch ابتدا Automapper را کامل می کنیم:



خود کنترلر هم به صورت زیر است:



* 1. درس 65 حذف اطلاعت توسط Repository

ابتدا به Repository متد Delete را اضافه می کنیم:



و خود متد را در کنترولر به صورت زیر اضافه می کنیم:

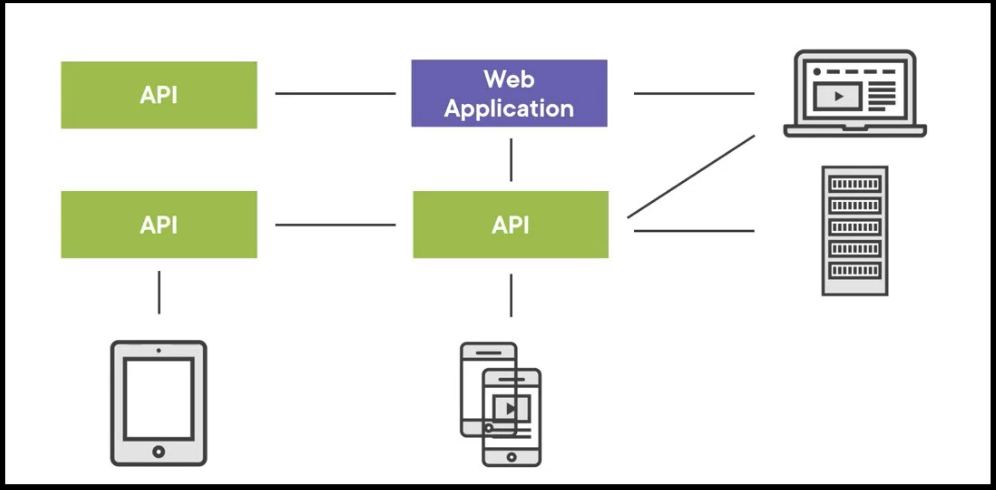


* 1. درس 66 امن کردن WebApi

توضیحاتی در مورد JWT داده شد.

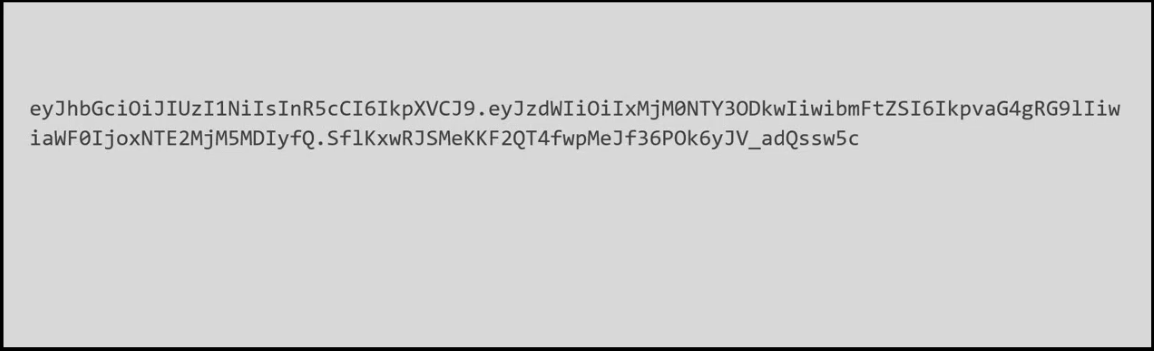
* 1. درس 67 بررسی امنیت WebApi و روش های ایمن کردن Web Api

توضیح در مورد JWT

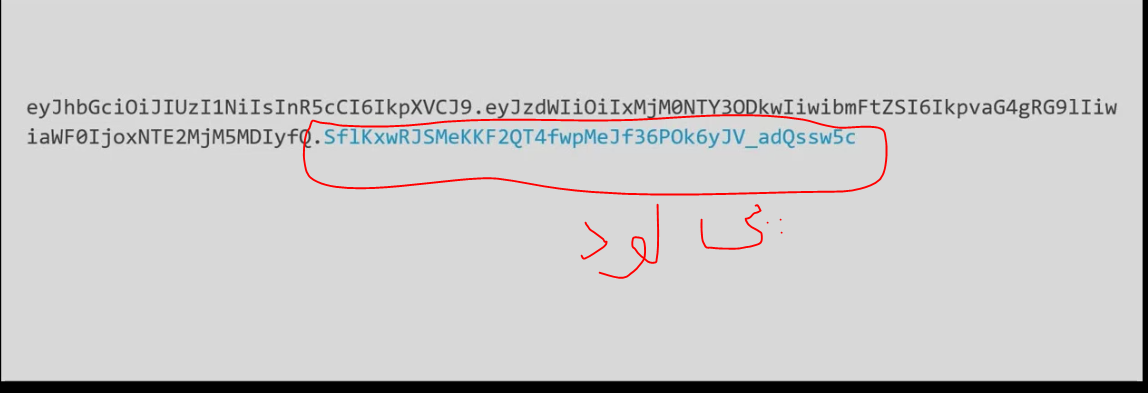


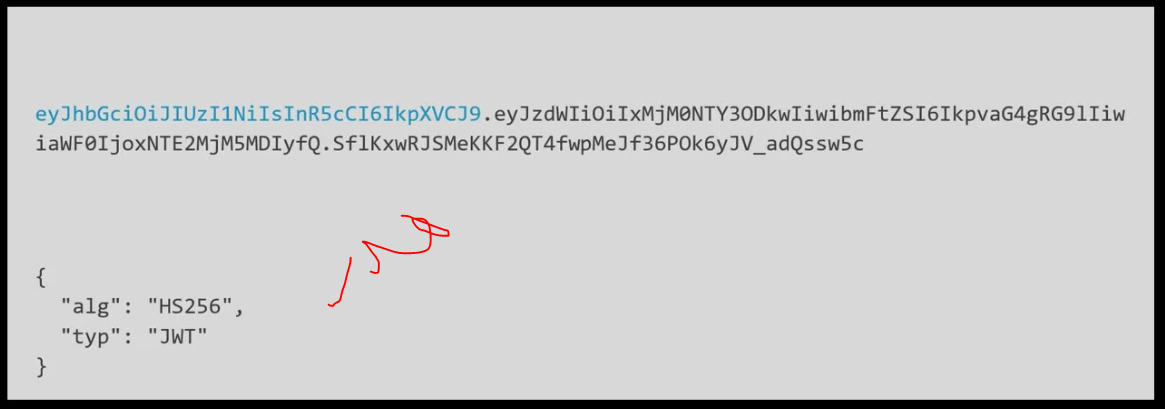
* 1. درس 68 بررسی ساختار توکن در Webapi

توکن از سه قسمت تشکیل شده است. پی لود، امضای کاربر یا Signature و هدر



پی لود شامل اطلاعتی از خود کاربر است مثل نام کاربری و ایمیل. امضای کاربر در سرور است و اطلاعاتی هست که قبلا ذخیره شده. هدر هم شامل الگوریتم و نوع هش است. سرور این توکن را ایجاد می کند و نگه نمی دارد.





* 1. درس 69 ساخت کنترلر Authentication جهت ایجاد توکن