قسمت 6

در این قسمت لایه Domain را به قسمت Core اضافه می‌کنیم. در این لایه Entity های خودمان را قرار می‌دهیم. موجودیت‌های که قرار است جدول انها در دیتابیس ایجاد شود.

جلسه 7

ساخت BaseDomainEntity که این کلاس abstract شامل تمام قسمت های فیلد های مشترک داخل همه کلاس های Domain است. بعد ساخت کلاس بقسه کلاس ها از این کلاس ارث بری می کنند.این کلاس را توی فولدر common قرار میدهیم.



قسمت 8

در این قسمت لایه core رو میسازیم. این لایه بین دیتابیس و برنامه ما و هر چیزی که می خواهد به دیتابیس وصل شود قرار میگیرد.برای این اتصال از الگوی Repository استفاده می کنیم.این اینترفیس ها اجازه می دهند که ما برنامه هایی که قرار هست با دیتا بیس ارتباط برقرار کنند را کنترل کنیم. به عنوان یک گلوگاه این کار را انجام بدیم. در این جلس IGenericRepository را میسازیم.



قسمت9

دراین جلسه IGenericRepository را می‌سازیم و فیلدهای اه را پر می کنیم.



بعد از این کار برای هرکدام از کلاس های Domain یک ریپوزیتوری خاص می‌سازیم. که از کلاس IGenric ارث بری کرده است. و وردی ان خود کلاس مربوط به ان است.



قسمت 10

AutoMapper

در این قسمت با نصب این پکیج برای تبدیل داده ها از Dto و entity ها استفاده می کنیم



کلاس را هم در این قسمت ایجاد می کنیم. از profile که مربوط به Automapper هست ارث بری می کنیم.



فولدر DTOs (Data Transfer Object)که مربوط به مدل ویو ها هست را هم در لایه Application ایجاد می کنیم.برای ارتباط بین Entity ها و برنامه ها ما از مدلهای DTO استفاده می کنیم.Automapper مدل های که در DTOs ساختیم را به Entity های Domain مپ می کند.

درس 11

در این قسمت برای هر کدام از entity های Domain یک DTO میساریم. بیس و همه موجودیت ها. هدف از این بخش این است که فقط خود موجودیت ها با دیتا بیس ارتباط داشته باشندو برای ازتباط بیقه برنامه ها با موجودیت ها باید از DTO استفاده کنند.



بعد از این بخش Automapper رو هم کانفیگ می کنیم.



همانطور که داخل کدها مشخص هست از داخل constractor از createmap استفاده می کنیم و در انتها با ReverseMap این عمل را معکوس می کنیم که هم درجهت و معکوس کار کند.

درس 12

ساخت DTO های اختصاصی. به طور مثال برای یک ویو خاص نیاز به همه فیلد های موجودیت نیست. در این موارد به صورت اختصاصی برای ان ویو مدل dto میسازیم.



بعد ایجاد این DTO در قسمت َAutomapinng این قسمت رو اضافه می کنمیم تا در صورت نیاز استفاده کنیم.



در ادامه برای اینکه وابستگی ها مدیریت بشوند و پروژه مقصد برای وابستگی های سایر لایه ها سنگین نشود ، در هر لایه وابستگی ها و سرویس ها را در یک کلاس قرار میدهیم و در پروژه مقصد فقط اون رو فراخوانی می کنیم.

ساخت اکستنشن متد برای کانفیگورلایه Application برای این کار کلاس زیر را میسازیم. کلاس و متد باید staic باشند و و با کلمه کلیدی this این کلاس را به صورت اکستنشن متد می کنیم. حالا سرویس Automapper را تنظیم می کنیم. در کد زیر خط کامت شده فقط کلاس Mapping profile را به سرویس میشناساند ولی خط بعدی که با Assembly نوشته شده همه مواردی که با Profile هستند را انتخاب می کند. این برای زمانی هستکه قصد داریم برای Dto ها کلاس Automapper جدا بسازیم.



درس13

الگویCQRS که به معنای مسئولیت جدا سازی کامند و کوئری است که در اینجا کامند همان عملیات کراد هست وکوئری منظور خواند اطلاعت از دیتا بیس هست. در این الگو قصد دارد با جدا سازی این دو نوع ارتباط با دیتا بیس سرعت و کیفیت این ارتباط را بالا ببرد. زمانی که درخواست ها از سرور زیاد می شود این الگو کمک بسیار زیادی می کند.



الگوی Mediaror این الگو برای مدیریت ارتباط بین اشیا و کتابخانه های پروژه استفاده می شود. این الگو شبه به تزریق وایستگی هست و از ارتباط مستقیم اشیا با یکدیگر جلوگیری می کند.زمانی که درخواستی به سمت برنامه ارسال می شود Mediaor در یافت می کند و با استفاده از Handler ها پاسخ میدهد.کتابخانه MediateR برای پیاده سازی این الگو در .net استفاده می شود. به نوعی برای پیاده سازی CQRS از Meiator استفاده می کنیم.



درس 14

در این بخش CQRS را با الگوی Mediator پیاده می کنیم. داخل Application فولدر Features را میسازیم. سپس داخل ان برای هر کدام از اشیا یک فولدر دیگه میسازیم. LeaveType داخل این فولدر دو فولدر Handler و Requests را میسازیم.و داخل فولدر Handlr دو فولدر Commands و Queries را میسازیم.



در ادامه در فولدر Requests کلاسGetLeaveTypeListRequest را می سازیم. و از کلاس IRequest که در پکیج MediatR هست ارث بری می کند.



کلاس به صورت زیر پیاده سازی می شود. داخل IRequest باید به صورت جنریک خروجی این متد که در این جا یک لیست از جنس LeavetypeDto هست داده شود.



بعد از این که ریکوست ثبت شد باید Handler ان هم ساخته شود با توجه به این که این درخواست از جنس کوئری هست در پوشته کوئری فولدر Handler کلاس هندلر ان ساخته می شود.



این کلاس از IRequesrHandler ارث بری می کند. و در ورودی ان دو مقدار در یافت می کند که مقدار اول کلاسی که در خواست دهنده است را می گیرد به عبارتی کلاس جاری Handler ان کلاس هست که در اینجا کلاس GetLeaveTypeListRequest و مقدار نوع خروجی یا پاسخ که باید به درخواست دهنده خود بدهد List<LeaveTypeDto> را مشخص می کند.



قسمت 15

در این قسمت Handler ها را تکمیل می کنیم.



از ابتدا برای برقراری ارتباط به پایگاه داده ریپوزیتور و مپپر را انجکت می کنیم.

private readonly ILeaveTypeRepository \_leaveTypeRepository;

private readonly IMapper \_mapper;

سپس تابع رو پیاده می کنیم.



سپس کلاس GetLeaveTypeDetailRequest پیاده سازی میکنیم.



دقت شود در این قسمت Id ورودی هست و نیاز هست که بگیرم و در ادامه اطلاعت جزئیات مربوط به هر مرخصی را بگیریم.



دقت شود باتوجه به اینکه این تابع ورودی داردId در قسمت مشخص شده از request.Id باید استفاده کنیم.

قسمت 16 و 17 تکمیل فولدر فیچرز

در این قسمت ها برای دو موجودیت leaveAllocation و Leavrrequest قسمت Query ساخته شد.



قسمت 18 تکمیل DTOs



در کلاس بالا زمان اکشن به درخواست مرخصی است که این اکشن توسط مدیر انجام می شود نه کارمند. پس در زمان ایجاد ممکن است خالی بماند. برای همین با ؟ Nullable می کنیم.

قسمت 19 ساخت command

در این قسمت برای ایجاد یک شی که در این جا Leavetype است با استفاده از الگوی CQRS استفاده می کنیم. چون از نوع در خواست نیست و کاری رو دیتا بیس انجام می شود باد در فولدر command فایل ها ایجاد شود. ابتدا در فولدر Request وفولدر command فایل درخواست را ایجاد میکنیم.



مقدار int به عنوان id شی ساخته شده بر می گردد و یک شی از مدل ورودی به handler داده می شود.



در فایل Handler هم تابع ساخته می شود. ابتدا مدل Dto به مدل انتیتی مپ می شود و در ادامه با ریپوزیتوری به دیتا بیس اد می شود ودر نهایت مقدار ای دی برگشت داده می شود.

قسمت 20

ادامه ساخت crate برای بقیه موجودیت ها



و هندلر به صورت زیر است



قسمت 21

اضافه کردن update مدل برای DTO برای ویرایش مدل ها. نکته ای که هست برای ویراش می توان چند تا مدل داشت برای مثال یک مدل برای در خواست کننده مرخص و یک مدل برای ریس که فقط مرخصی تایید میکنه.





این دو مدل هر دو برای اپ دیت هستند ولی در مدل اول فقط یک فیلد هست ولی در مدل پایین تمام فیلدها به روز می شوند.

جلسه 22

ساخت cammandHandler برای اپ دیت



در قسمت command به صورت بالا عمل می کنیم. منظور ازuint هم این است که گاهی چیزی بر می گردونیم وگاهی نه. برای MediateR این مفهوم را دارد.



قسمت Handler هم به صورت بالا ساخته می شود. برای بقیه موارد هم میسازیم.

درس 23

تکمیل کردن update

درس 24

در این قسمت با توجه به اینکه برای به روز کردن leaveRequest دو DTo داشتیم



برای اینکه این روش به روز کردن هم انجام شود تغییراتی دادیم که به این صورت هست:



قسمت های رنگی را به کلاس کامند اضافه کردیم. تایع هندلر را هم به صورت زیر تغییر می دهیم



درس 25

ایجاد بخش Delete



و در بخش هندلر هم تایع به صورت زیر پیاده می شود:



درس 26

نصب وفعال سازیFluentValidation

برای اعتبار سنجی روش های مختلفی هست که چیزی که مرسوم هست استفاده از validator ها در زمان تعریف entity یا مدل است. در این جا از Fliuentvalidation استفاد می کنیم. این پکیج رااز NUGET می گیرم. و در HR\_Management.Application که شامل DTO ها است نصب می کنیم.



بعد از نصب داخل هر موجودیت یک کلاس ایجاد می کنیم:



نحوه ی نوشن اعتبار سنجی برای یک کلاس هم به صورت زیر است:



نکته Rule داخل constractor تعریف می شود.

قسمت زرد زنگ شامل کلاس است که برایش اعتبار سنجی می نوسیم.

درس 27

در این قسمت از validation در زمان کامند استفاده می کنیم. نحوه ی پیاده سازی اعتبار سنجی:



قسمت زرد نحوه ی استفاده از اعتبار سنجی را نشان میدهد.

درس 28

در این قسمت برای CreateLeaveRequestDtoValidator اعتبار سنجی می نوسیم.



برای انکه اعتبار سنجی دقیقتر انجام شود داخل ریپوزیتوری یک متد اضافه می کنیم:



درس 29

در این قسمت ابتدا برای اعتبار سنجی LeaveTypeId از روش زیر استفاده می کنیم:



برای اعتبار سنجی وجود یک ای دی ابتدا یک mustAsync تعریف می کنیم که ورودی ان ایدی و توکن است . توکن همان cancellationToken است. از ریپوزیتوری استفاده می کنیم و بررسی می کنیم که ایدی در دیتا بیس هست یا نه وخروجی را بر می گردانیم.

در قسمت که برای leaveType اعبتار سنجی نوشتیم دو کلاس پراپرتی های یکسانی داشته اند. برای پر هیز از تکرار وسالید این کارها را انجام میدهیم. ابتدا یک اینترفیس از پراپرتی های مشترک دو کلا س ایجاد می کنم.



سپس دوکلاس از این اینترفیس ارث بری می کنند:



حالا در قسمت اعتبار سنجی یک کلاس اعتبار سنجی برای اینترفیس می نوسیم:



حالا در کلاس هایی که از این اینتر فیس ارث بری کرده اندبه صورت زیر عمل می کنیم



نکته این است که برای تکرار نکردن کدها قسمت زرد را می نویسیم. و با include کردن اینترفیس اعتبار سنجی ان را استفاده می کنیم.

* 1. درس 30 ادامه اعتبار سنجی مدل ها

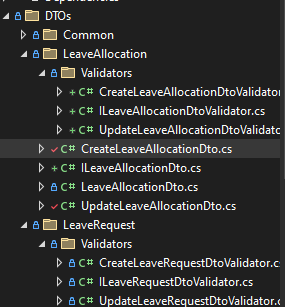
در این جلسه ادامه اعتبار سنجی انجام شد. برای اضافه کردن rule اختصاصی به هر کلاس به روش زیر عمل می کنیم:



قسمت زرد رنگ نشان میدهد که بعد از include کردن اینترفیس rule دلخواه خودمان را اضافه می کنیم.

* 1. درس 31 تکمیل validator ها با fluentValidator

نوشتن validator برای LeaveAllocation



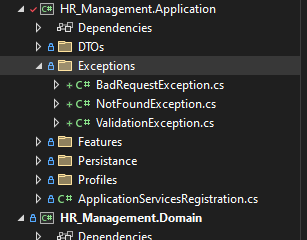
* 1. درس 32 تکمیل اعتبارسنجی Handler ها درCQRS

در این جلسه اعتبار سنجی ها را برای بقیه موارد کامل می کنیم:



* 1. درس 33 ساخت Exception اختصاصی در AspCore

در این جلسه Exception های اختصاصی میسازیم. برای این کار در لایه Application فولدر را میسازیم و برای شروع کلاس های زیر را میسازیم:



برای این که خاصیت Exception داشته باشند از کلاس ApplicationException ارث بری می کنیم.







* 1. درس 34 نحوه ی استفاده از CustonException های ایجاد شده درCQRS

در این جلسه از Exception های اختصاصی که برای مدیریت خطا ها ساخته‌ایم استفاده می‌کنیم کد زیر نحوه‌ی استفاده از این موارد را نشان میدهد.



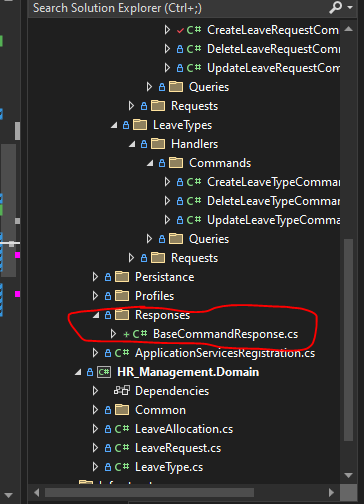
در این مورد اعتبار سنجی استفاده شده است.

برای حذف هم از مورد زیر استفاده می شود:



* 1. درس 35 ساخت Respons های سفارشی و شخصی سازی شده در CQRS

در جلسات قبل ما ValidationException را برای مدیریت شکست در انجام یک تسک ایجاد کردم مثلا زمانیکه create موفق نبود از ValidationException استفاده کردیم . ولی گاهی نیاز هست که جواب اختصاصی ایجاد کنیم گاهی جواب خارج از خطا ساده و برگشت دادن یک id سادست. البته این اجباری نیست و فقط زمانیکه نیاز داشتیم برای یک تسک خاص این کار را انجام میدهیم.



فولدر بالا وکلاس بیس را ایجاد می کنیم. کلاس بیس را به صورت زیر تعریف می کنیم:



Id که درابتدا تعریف شده است برای زمانی هست که پاسخ تسک مورد نظر یک Id برگرداند. بعد این کار به طور مثال برای CreateLeaveRequestCommand یک پاسخ اختصاصی میسازیم:



مواردی که قرمز هستد حذف شدند و موارد زرد اضافه شدند. البته برخی موارد هم در Handler تغییر کردند.

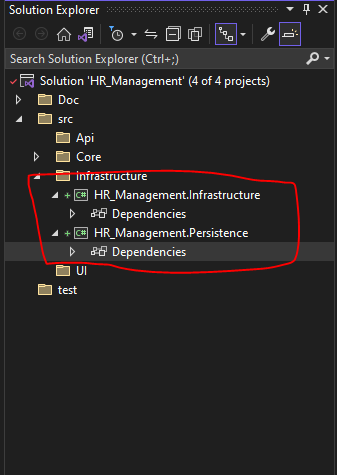
* 1. 36 تکمیل Mapping Profile



در این قسمت پروفایل mapping رو کامل کردیم.

* 1. 37 بررسی و پیکر بندی لایه Infrastructure

در این لایه مواردی مثل context ارتباط به دیتا بیس و Repository ها ایجاد می شوندو این لایه از لایه Core استفاده می کند.



لایه Persistence برای ایجاد ارتباط با دیتابیس ایجاد شده است. کلاس ها و EFCore .در این لایه هستند.

لایه Infrastructure هم برای پیاده سازی استفاده می شود. پیاده سازی سرویس ها و ریپوزیتوری ها را دارد.

* 1. 38-ساخت DBcontext و Entity ها

در این قسمت Context را می سازیم. برای این کار ابتدا به لایه Persistanse لایهApplication را رفرنس میدهیم و دو پیکیج را نصب می کنیم:



بعد از نصب این دو پکیج کلاس کانتکت رو میسازیم:



* 1. درس 39-تنظیمات Context در EFCore

در این مرحله بعد از تعریف جدول ها در context عبارت زیر را قرار می دهیم:



در این متد ما زمان ایجاد جدول های مواردی که خواست خود ما است را هم اعمال می کنیم. در این مورد هم از assembly خود leaveManagment استفاده می کنیم.

در مرحله بعد کد های زیر را اضافه می کنیم:



در کد های بالا متد Savechange و SaveChangeasync را override می‌کنیم و تنظیمات دلخواه خود را در موقه ذخیره اعمال می‌کنیم. برای این کار زمانی که ChngeTracker که یکی از متدهای EF است و Entity هایی که در حال Trach شدن هستن را انتخاب می کنیم و روی BaseDomainEntity که شامل موارد عمومی هر موجودیت است حلقه می زنیم که اگر موجودیت در حال به روز رسانی بود زمان بروز رسانی ثبت شود و اگر موجودیت در حال ایجاد بود زمان ایجاد موجودیت هم ثبت شود. برای تکمیل کار هم می توان در همین قسمت زمان Update یک موجودیت می توان حالت اولیه و ثانویه ان بعد تغییر را ذخیره کرد.

* + 1. درس 40- پیاده سازی GenericRepositoy ها

در این قسمت IGenericRepository که در لایه application هست را در لایه Persistanse پیاده می کنیم:



* 1. درس 41. پیاده سازی Repository ها در لایه Persistanse

در این بخش بقیه Repository ها را پیاده سازی می کنیم:



دقت شود که جنیریک ریپوزیتوری ارث بری می کند و کانتکس ایجاد شده هم باید به بیس که جنیریک ریپوزیتوری هست باید پاس داده شود.

برای دو ریپوزیتوری باقی مانده هم به صورت زیر عمل می کنیم:



* 1. درس 42 ساخت و کانفیگ Persistence Registration در CQRS

در این بخش سرویس های مربوط به لایه Persistance را در یک کلاس استاتیک می نوسیم. از انجای که اگر قرار باشد همه کانفیگ ها در فایل Services انجام شود این کلاس بسیار شلوغ می شود همه سرویس های هر لایه را در همان لایه انجام میدهیم.



* 1. درس 43 ساخت سرویس thirdParty در Infrastructure

در این قسمت یک سرویس ایمیل ایجاد می کنیم. برای این کار در لایه application قسمت Contracs این سرویس را ایجاد می کنیم. ابتدا یک فولدر Infestructure ایجاد می کنیم و داخل ان سرویس IEmailSende را ایجاد می کنیم.



در ادامه مدل Email را در فولدر Models ایجاد می کنیم.



برای تنظیمات ایمیل هم از یک کلاس استفاده می کنیم به نام EmailSetting



ار سرویس ایمیل در درخواست مرخصی استفاده می کنیم leave request handler

در این قسمت ابتدا سرویس را اضافه میکنیم و بعد از اضافه کردن سرویس ایمیل را تعریف می کنیم. بعد از این تعریف چون از یک سرویس خارجی استفاده می کنی از try cach استفاده می کنیم.



* 1. درس 44 پیاده سازی سرویس EmailSender

در این جلسه سرویس ایمیلی که قبلا ساختیم را پیاده سازی می کنیم.برای شروع در لایه Infestructure فولدر mail را ایجاد می کنیم و کلاس EmailSender را میسازیم.



دو پکیج را به لایه Infrastructure اضافه می کنیم:



بعد کلاس EmailSender را با استفاده از Sendgrid پیاده سازی می کنیم:



برای تنظیمات سرویس های این لایه هم به صورت زیر انجام میشود:



* 1. درس 45 ایجاد WebAPI و کانفیگ سرویس ها

در این قسمت ابتدا یک WebApi به عنوان لایه UI میسازیم.بعد از ایجاد پروژه در قسمت appsetting تنظیمات مورد نیاز خودمان را قرار می دهیم.در شکل زیر دو قسمت تنظیمات رشته اتصال و ارسال ایمل را کامل می کنیم.



بعد از قسمت سرویس ها را در فایل Program.cs پروژهWebApi ست می کنیم:



* 1. درس 46 تعریف و تنظیم Cors درApi

در این قسمت تنظیمات Api را انجام میدهیم که شامل دسترسی ها است.



* 1. درس 47 ساخت دیتا بیس و Migration

براین استفاده از دیتا بیس از پکیج های زیر را در لایه Api نصب می کنیم:



در ادامه با دستورات Migration دیتا بیس را میسازیم:



* 1. درس 48 ساخت اولین کنتولر

برای شروع کار پکیج MediateR را به لایه Api اضافه می کنیم:

و کنترولر LeaveType را میسازیم:

ابتدا برای استفاده از MediateR این سرویس را inject می کنیم. در ادامه در قسمت Get برای گرفتن لیست داده ها از متد send این سرویس استفاده می کنیم. برای گرفتن این داده ها از آدرس:



استفاده می کنیم.



* 1. درس 49 Mapکردن Handler ها با MediatR

در این قسمت کنترولر مربوط بهLeaveType را می سازیم.

کلیه عملایت های داخل این کنترلر را ایجاد می کنیم:



* 1. درس 50 تکمیل کنترولر های Api

در این جلسه کنترولر LeaveAoolcation را میسازیم:



* 1. درس 51 تکمیل کنترولر های Api

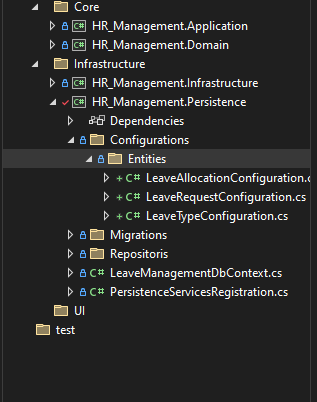
در این جلسه کنترولر LeaveRequest را کامل می کنیم:



* 1. درس 52 افرزودن اطلاعات اولیه سیستم با SeedData در EF

در این جلسه اطلاعات اولیه جدول هایی که در دیتا بیس داریم را ایجاد می کنیم. این جداول شامل اطلاعاتی هستند که در در ابتدای پروژه برای راه اندازی نیاز است. برای این کار از SeedData استفاده می کنیم:

برای شروع پوشه Configurations را به لایه Persistence اضافه می کنیم. در ادامه پوشه Entities را اضافه می کنیم که این پوشه شامل اطلاعات اولیه Entity ها است.



برای شروع کلاس LeaveTypeConfiguration را میساریم.برای شناساندن این کلاس به عنوان SeedData به EF کلاس از IEntityTypeConfiguration<LeaveType> ارث بری می کند و با استفادهاز Builder و HasData کلاس های اولیه را تعریف می کنیم. برای دو Entity دیگر هم به همین صورت تعریف می کنیم. بعد از این کار با Migration دیتا بیس را به روز می کنیم.



* 1. درس 53 معرفی و استفاده از Swagger

توضیحاتی برای استفادهاز Swagger

* 1. درس 54 تزریق وابستگی های MediatR در Registeration

در این درس برای استفاده از MediatR باید بسته ای برای تزریق وابستگی به ان اضافه کنیم:



* 1. درس 55 معرفی UnitTest و تست نویسی در معماری تمیز

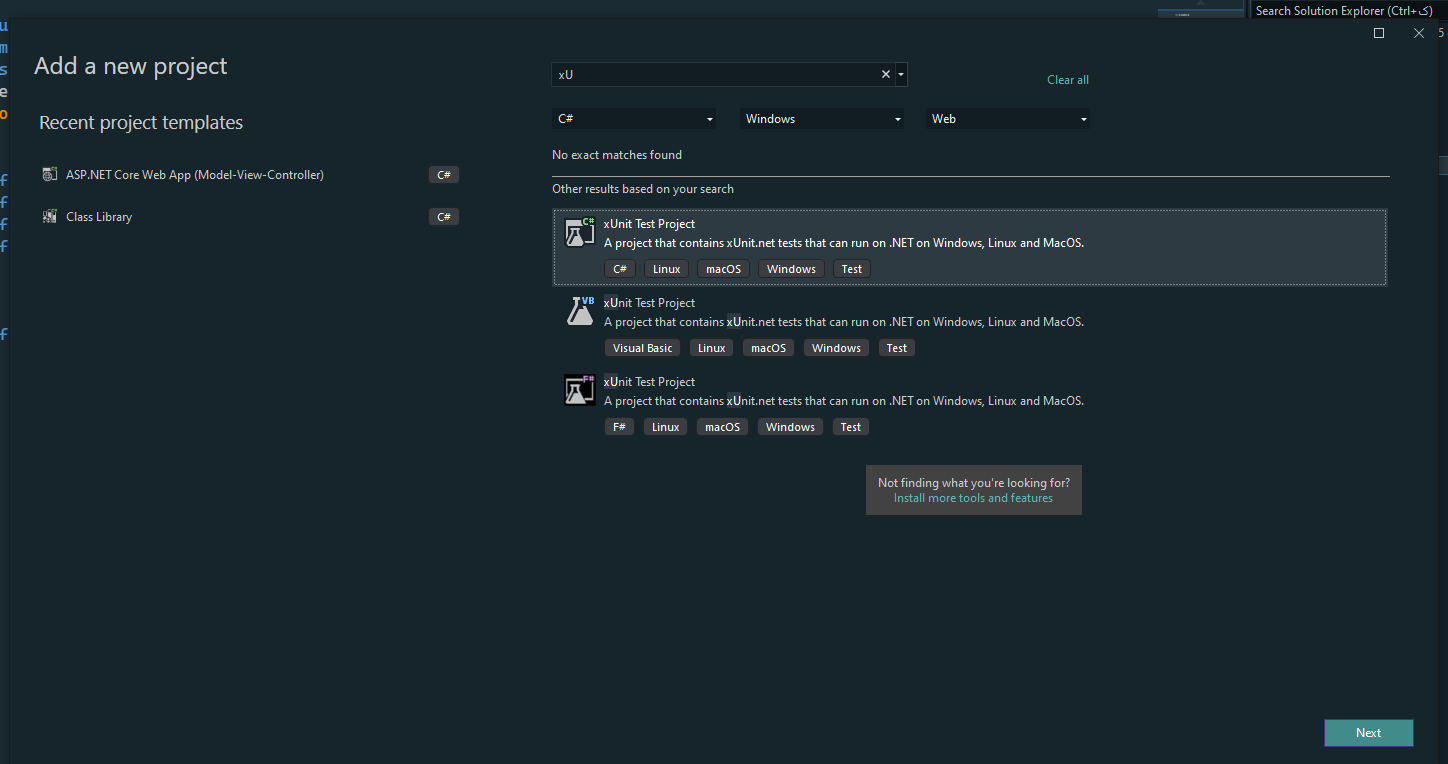
صحبت و بحث در مورد اهمیت تست نویسی.

* 1. درس 56 رفع باگ هندلرها

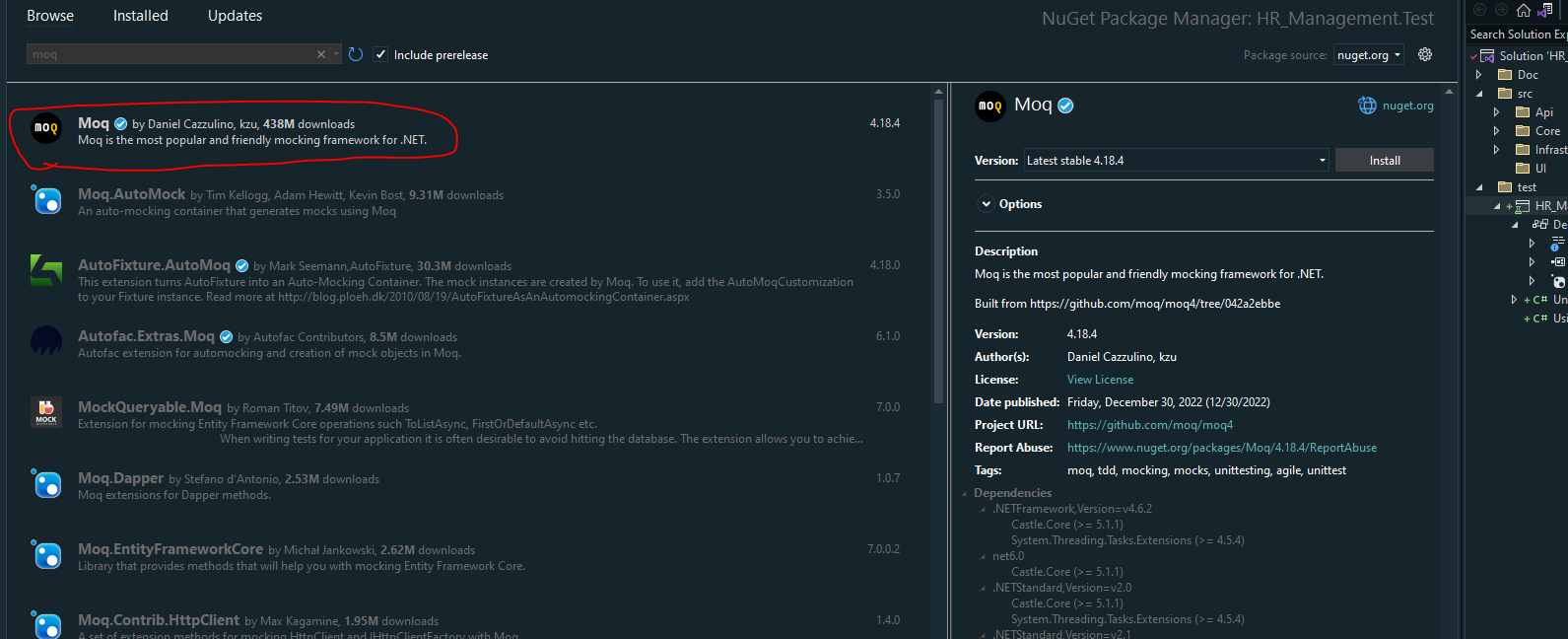
اصلاح باگ ها و اشتباهات.

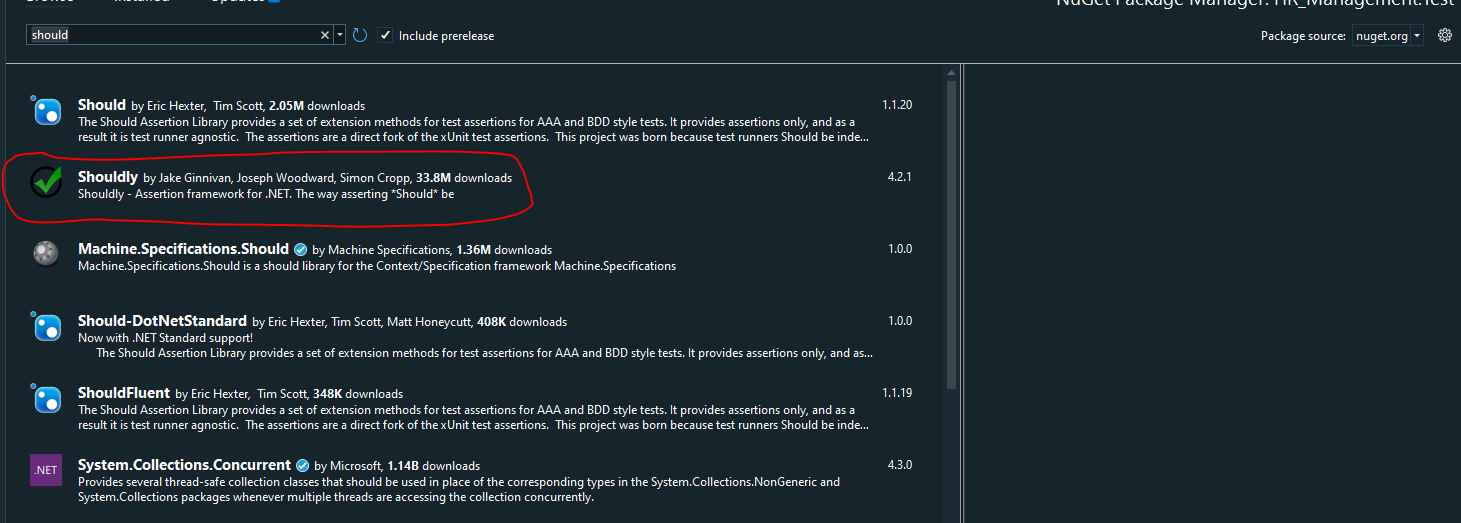
* 1. درس 57 ساخت پروهUnitTestو نصب بسته های مورد نیاز

در شروع ایجاد پروژه تست ابتدا پروژه تست را اضافه می کنیم:



در ادامه 2 پکیج زیر را اضافه می کنیم:





این دو پکیج کمک می کند از ریپوزیتوری خودمان یک شبیه ساز درست کنیم تا در زمان تست روی داده‌های اصلی کار نکییم. کلاس ها و ... را ماک کنیم.