# **شرح پروژه (**تمرین) جلسه دوم :

یک پروژه براساس ساختاری که در جلسه دوم نوشتیم ایجاد بفرمایید به همراه Entity های زیر که شامل Service ها و Controller ها باشید :

* Supplier
* Customer
* Employee

# **مرحله اول: ساختار بندی پوشه ها در سطح اول پروژه قبل از لایه بندی**

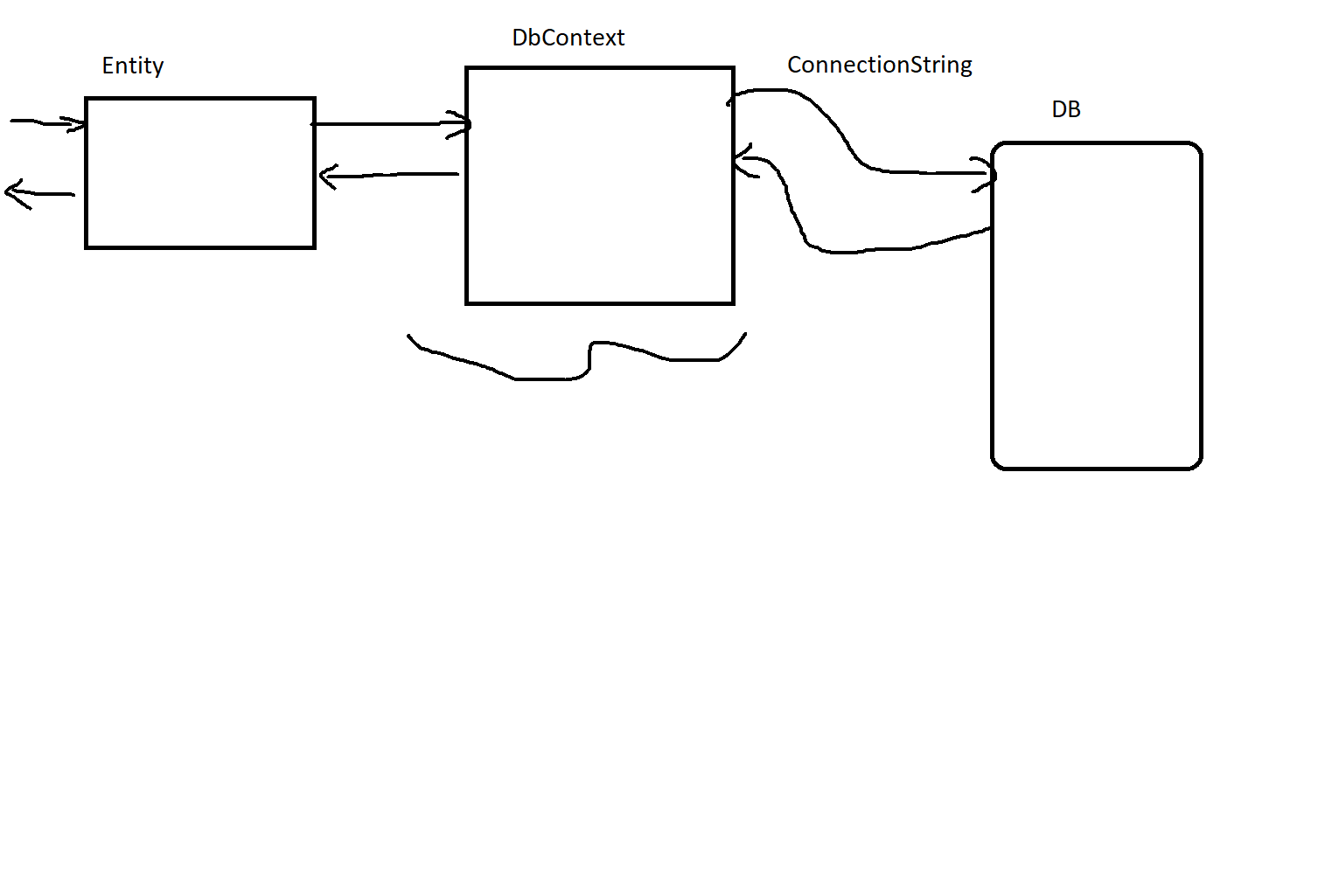
* Doc
* Src
* Test

# **مرحله دوم : ساخت solution و پروژه های لازم جهت لایه های مختلف**

* OnlinesShop -> Solution
  + API -> Web API
  + Application -> class Library
  + Infrastructure -> class Library
  + Core -> class Library

# مرحله سوم : ساخت **Entities در پروژه Core**

* Supplier
* Customer
* Employee
* Product



# مرحله چهارم : ساخت **Context** و **Dto** در لایه **Infrastructure** و اقدامات مربوطه شامل:

* نصب پکیج های لازم در همین لایه شامل :
* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Tools
* ساخت فایل **OnlineShopDbContext**
  + ارث بری از کلاس **DbContext**
  + تعیین مسیر در سازنده
  + ساخت DbSet ها بعنوان نماینده جدول های دیتابیس
  + ساخت Dto ها جهت استفاده در لایه های بالاتر

# مرحله ششم : ساخت **اینترفیس ها و سرویس ها** در لایه **Application**

* ساخت پوشه Interfaces >> ابتدا اینترفیس ها جهت هر Entity ساخته می شود > امضای توابع با استفاده از Dto
* ساخت پوشه Services >> جهت هر Entity یک سرویس ساخته شده و از اینترفیس ارث برده و آنرا پیاده می­کند
* در این قسمت چون نیاز به ذخیره و بازیابی دیتا داریم باید از ریپازیتوری استفاده کنیم. اما چون در این پروژه هنوز ریپازیتوری نداریم، فعلا از OnlineShopDbContext استفاده می کنیم.
* نکته خیلی مهم! جهت ذخیره دیتا ها باید از خود Entity استفاده کنیم و در نتیجه باید از Dto به entity تبدیل کنیم. ساده ترین راه استفاده از AutoMapper هست ولی این کار خیلی performance را در پروژه های بزرگ کم میکند و در نتیجه بهتر است خودمان بصورت دستی این تبدیل را در سرویس ها انجام دهیم
* از آنجا که بازای هر request یک Thread اختصاص داده میشود، عملیات ها را استفاده از async / await انجام می دهیم که Performance کار ما بالاتر برود. با این کار Thread دیگر منتظر جواب نمی ماند (بلاک نمی شود) و در نتیجه می تواند کار دیگری را هم انجام دهد. در نتیجه با این کار، بدون افزودن منابع سخت افزاری به سرور، ممکن است توانایی پاسخ دهی به درخواست ها از 5K Request/Second به 50K Request/Second افزایش یابد.
* نصب پکیج های لازم در همین لایه شامل :
* Microsoft.EntityFrameworkCore

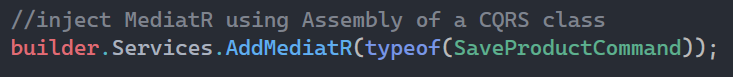
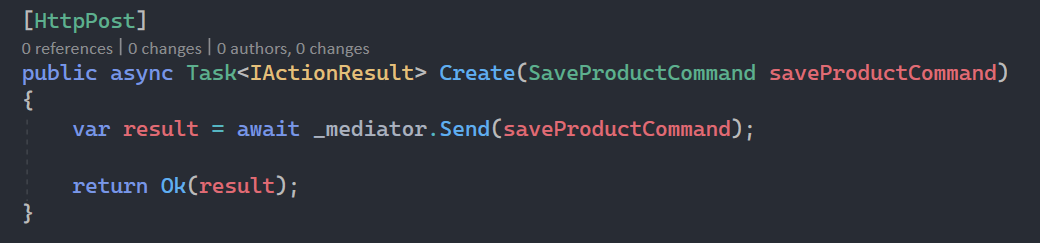
# مرحله هفتم : تزریق وابستگی های دیتابیس و سرویس ها در لایه **API**

* نصب پکیج های لازم در همین لایه شامل :
* Microsoft.EntityFrameworkCore
* Microsoft.EntityFrameworkCore.SqlServer
* Microsoft.EntityFrameworkCore.Design
* ایجاد کانکشن استرینگ درون فایل appsettings.json جهت داینامیک بودن آن و توانایی تغییر راحت
* رجیستر کردن DbContext
* رجیستر کردن سرویس ها و تزریق آنها به درون اینترفیس ها
* پیاده سازی Controller ها با استفاده از سرویس ها

# مرحله هشتم : **Migration** دیتابیس از لایه **Infrastructure**

* اجرای دستورات زیر:
  + Add-Migration init
  + Update-Database

# \*\*\* شروع کار با **CQRS و** افزودن پکیج **MediatR در لایه Application**

* در این فاز کامند ها و کوئری را از هم جدا می­کنیم.
* یک لایه برای نوشتن بیزینس لاجیک در نظر میگریم مثلا Application
* در لایه انتخابی پکیج MediatR را نصب میکنیم.
* در لایه انتخابی یک پوشه به نام CQRS ایجاد کنیم که دارای دو زیرپوشه Command و Query می باشد.
* کلاسهای لازم اینجا نوشته می شوند.
* در مرحله بعد به سراغ پروژه API میرویم:
  + ابتدا پکیج MediatR.Extensions.Microsoft.DependencyInjection در پروژه API اضافه میکنیم
  +  سپس باید MediatR در پروژه API در فایل Program.cs اینجکت شود
  + حال یک کنترلر میسازیم و به خیر و خوشی از این سرویسهای جدید استفاده می­کنیم. ترجیحا درون نام کنترلر از عبارت CQRS استفاده شود مثلا ProductCQRSController

ادامه...