**AutoMapper**

Biz security cəhətdən project daxilində Front’a database’dən gələn Entity object’ləri bir başa olaraq göndərməyimiz təhlükəlidir. Çünki biz bu object’ləri bir başa ötürsək əlçatan olmamalı bir sıra dataları da (property’ləri) göndərmiş olarıq. Bunun qarşısını almaq üçün ViewModel və ya DTO class’ları yaradıb müəyyən olunmuş property’ləri saxlayırıq.

Hər dəfə datanın ötürülməsi və ya qəbul olunması zamanı yeni ViewModel & DTO və ya Entity object yaradılması və dəyərlərin mənimsədilməsi həm kod təkrarına gətirib çıxarır həm də vaxt itkisinə. Bu problemin qarşısını isə reflection ilə rahatlıqla ala bilərik. Lakin bunu bizim üçün əvvəlcədən edib, AutoMapper package halına salıblar.

**AutoMapper DI(Dependency Injection)**

AutoMapper və AutoMapper.Extensions.Microsoft.DependencyInjection package yüklənlənməlidir. AutoMapper tətbiq etmək istədiyimiz Entity adına uyğun olaraq sonu Profile ilə bitən class yaradılmalı, Package vasitəsilə gələn Profile class’dan miras almalıdır. Boş constructor override edilməli, CreateMap<T,R>(); generic method işə salınmalıdır. Bu halda göndərilən T type(class) və R type(class) daxilindəki property’lər və onların value’ları Assembly tərəfindən oxunur, T type daxilindəki property’lərdən adları R type daxilindəki property’lər ilə üst-üstə düşənlər yeni R type object’ə mənimsədilir və geriyə qaytarılır. Burada adları eyni lakin datatype’ları fərqli olan property’lər də ola bilər. Bu halda runtime exception baş verəcək.

**ReverseMap();**

ReverseMap(); method’dan da istifadə edərək həm Entity göndərib ViewModel & DTO object’i həm də ViewModel & DTO göndərib Entity objecti qəbul edə biləcəyimizi bildiririk.

Sonda services olaraq adlandırdığımız hissədə gəlib builder.Services.AddAutoMapper(typeof(mapperAssembly)); vasitəsilə Mapper class’ların yerləşdiyi Assembly’ni tanıdaraq prosesi yekunlaşdırırıq.

Dependency injection ilə Mapper’dan istifadə edəcəyimiz class daxilində IMapper object’i çağırırıq, Map<R>(T); method’u tətbiq edib R type object’i alırıq.