**Repository Design Pattern**

Günümüzdə bir çox dataya əsaslanan application’lar, database’lərdə yerləşən məlumatlara əlçatan olmalıdır. Ən asan və bəsit yaxınlaşma database’də yerləşən məlumatlarla həyata keçirilən əməliyyatların hamısını əsas kodlarla birlikdə yazmaqdır.

Məsələn hər hansısa ASP.NET MVC project’də controller varsa(Employee controller deyək) burada tipik CRUD(Create, Read, Update, Delete) əməliyyatları həyata keçirəcək action’lar olduğunu, və həmçinin database ilə əlaqəli bütün bu proseslər üçün Entity Framework istifadə etdiyimizi fərz edək. Bu zaman Entity framework bir başa olaraq DbContext class’ı ilə əlaqəyə girir, data almaq və ya ötürmək üçün sorğular həyata keçirir. Entity Framework təməldə SQL Server verilənlər bazası ilə “danışır”.

Repository pattern, interface istifadə edərək(abstraktlaşdıraraq) DbContext(yəni data access layer) ilə business logic layer’i bir-birindən ayrılmasını və onlar arasında vasitəçi və ya orta təbəqə(layer) rolunu təmin etmək üçün istifadə edilən memari design pattern’dir. Bu o deməkdir ki Repository pattern database’lə bağlı olan kodları, application’un geri qalan hissəsindən təcrid edir. Bizə qazandırdığı ən böyük üstünlük database’lər bağlı olan əməliyyatların hamısını bir yerdən idarə edə bilməkdir. Başqa bir faydası da controller’ləri daha rahat test edə bilməkdir. Lakin Repository Pattern **SOLID** prinspini pozur. Çünki həm data çəkmək, həm də data göndərib onun üzərində hər hansısa bir iş görmək üçün olan method’lar eyni class daxilində yerləşir. **SOLID** prinsipini pozmaq istəmiriksə Read & Write olaraq Repository’ni 2 yerə bölürük.

**Repository Pattern’in tətbiqi**

Repository Pattern’i tətbiq etmək üçün əgər Onion (Clean) Architecture istifadə ediriksə Application folder daxilində Interface Repository’lərin saxlanılması üçün Repositories folder yaradılır. IRepository, IReadRepository və IWriteRepository burada yerləşir.

IRepository’də DbSet type Table property’miz yer alır, IReadRepository’də read’lə bağlı get method’larımız yer alır və IWriteRepository’də isə Create, Update və Delete üçün lazım olan method’larımız və SaveChanges method yer alır.

Bu interface’lərin Concrete Repository class’ları isə Persistence folder daxilindəki Repositories folder daxilində olacaq. ReadRepository class IReadRepository interface’i , WriteRepository class isə IWriteRepository interface’i implement etməlidir. Lakin IRepository’nin concrete class’ı olmamalıdır.

**Generic Repository Pattern**

Repository’ləri Generic olaraq istifadə etməli olduğumuz üçün interface’lər aşağıdaki şəkildə olmalıdır. IRepository<T> , IReadRepository<T> və IWriteRepository<T> yəni T type qəbul etməlidir. Marker pattern vasitəsi ilə də biz müəyyən constraint’lər verərək T’in yalnız müəyyən class’lardan miras alan objectlər ola biləcəyini də qeyd edirik. Bu interface’lərin concrete’ləri də həmçinin T type object qəbul etməli, interface’lərdə marker pattern istifadə edilmişdisə concrete class’larda həmçinin istifadə edilməlidir. Daha sonra hər bir entity üçün müvafiq olaraq interface və concrete class’lar yaradılmalıdır. Interface’lər uyğun olaraq IRead və ya IWrite repository’dən implement etməli və Generic type olaraq entity’in özü göndərilməlidir. Entity’lərə uyğun olaraq yaradılan interface’lərin concrete class’ları isə həm də uyğun olaraq Read və ya Write repository class’lardan extend etməlidir.



