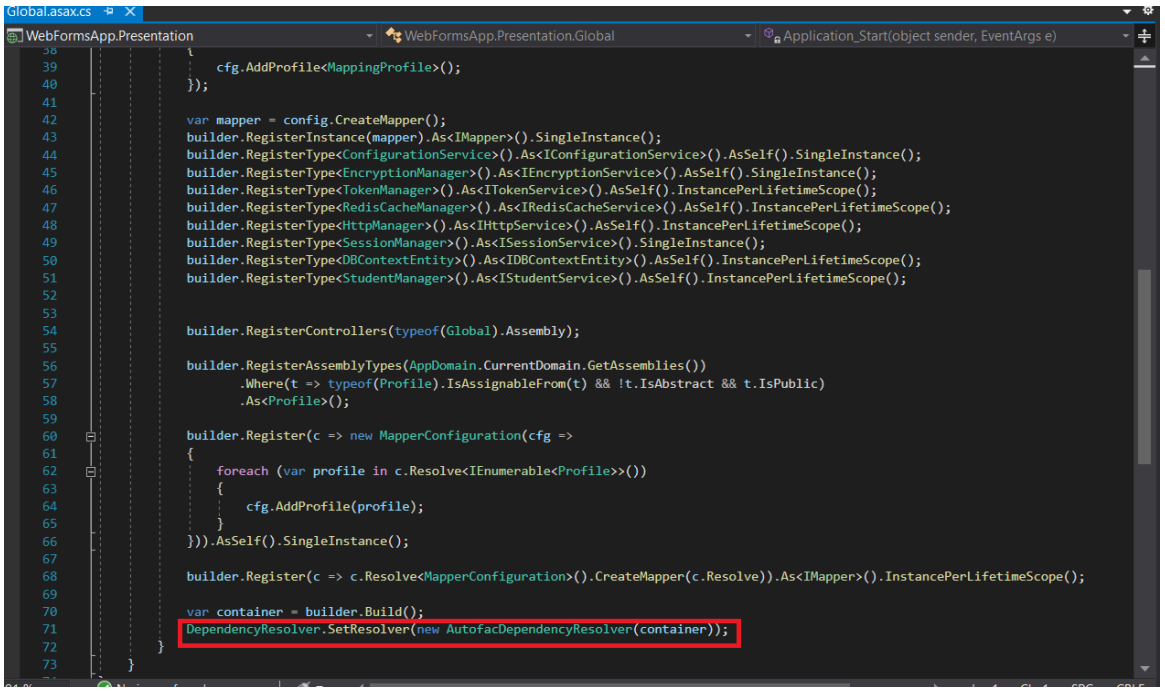
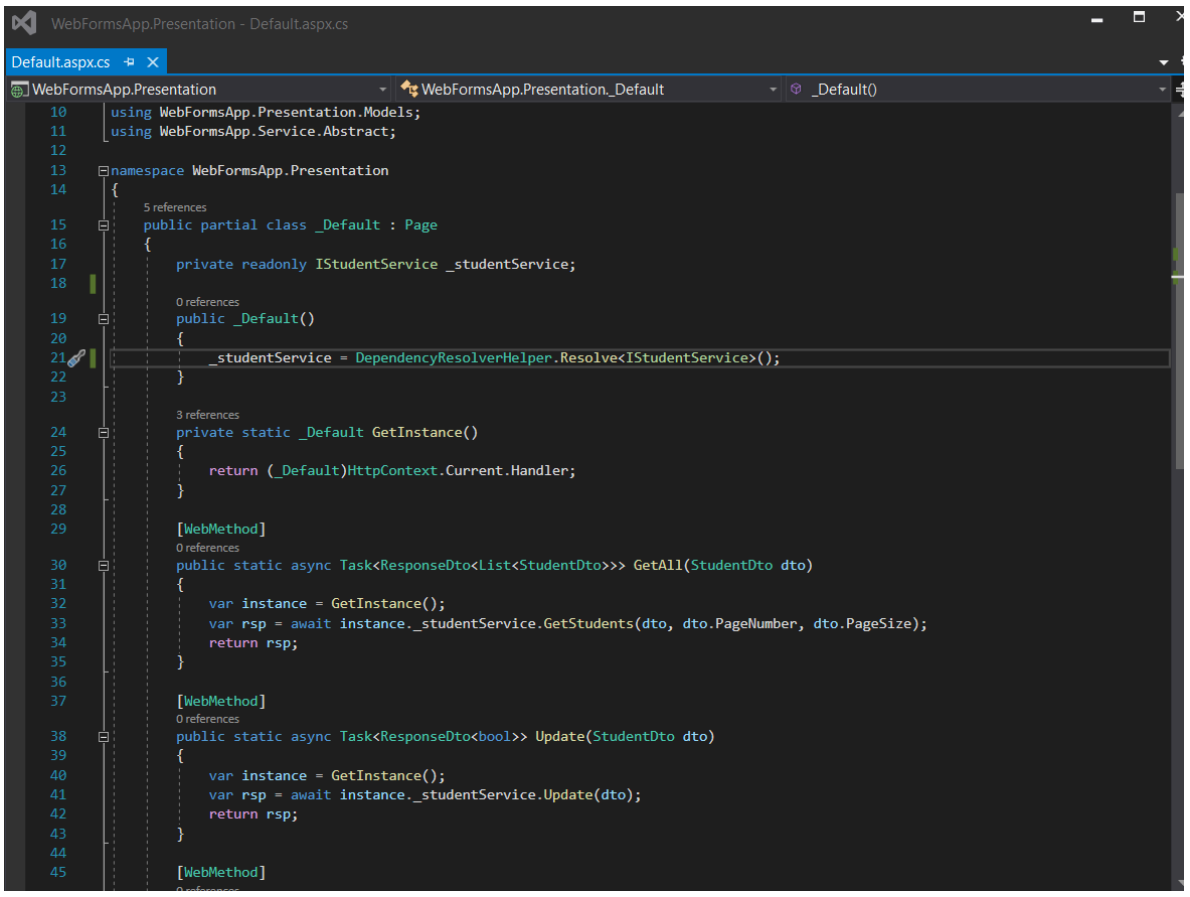
AÇIKLAMALAR

1. WebFormsApp yazılımı, Data, Entity, Service, Shared ve Presentation katmanlarından oluşur. Asp.Net .Net Framework 4.7 WebForms projesidir. Yazılım geliştirme süreçlerinde daha sürdürülebilir, esnek ve bakımı kolay sistemler oluşturmak amacıyla solid prensiplerine uygun yaklaşımlar benimsenerek projede uygulandı. Dependency injection yönetimi için AutoFac kütüphanesi presentation katmanına eklendi ve Global.asax.cs'de konfigürasyonlar buna göre ayarlandı. Projeye Authorization eklendi. Proje çalıştırıldığında /About linkine gitmek istediğinizde 401 hatası vermektedir. Class'ların birbirine dönüşümünü kolaylaştırmak amacıyla Automapper kütüphanesi eklendi.





2. Data Katmanında Ado.Net Entity Data Model eklenerek data katmanı oluşturulmuş oldu. Burada sadece Data Model eklenmesinin sebebi, sonradan tablolarda yapılan değişiklikler modelde uygulanmaya çalışılırken karışıklıklara ve değişikliklerin modele uygulanmamasına sebep olmaktadır. Nitekim kodlara da bu sorunlar yansımaktadır. Proje katmanının derlenmesine engel olmaktadır. Bu strateji uygulanırsa, böyle bir durum yaşandığında, ado.net data modeli kolaylıkla silinip yeniden oluşturulması sağlanabilir. Bu nedenle Data katmanında herhangi bir kodlama yapılmadı. DBContext'in arayüz üzerinden kontrolünü sağlamak amacıyla Servis katmanında IDBContextEntity arayüzü ve buna bağlı Concrete yapısı olarak DBContextEntity oluşturuldu.

3. Entity katmanında Concrete ve Dtos olmak üzere iki klasör bulunmaktadır. Concrete klasöründe sınıflar oluşturulurken başına M harfi eklenir. Değişkenler ingilizce olarak oluşturulduğu için, bazen başka sınıfların modelleri ile çakışmalar yaşanmaktadır. Bunun bizim modelimiz olduğunu belirtmek amacıyla başına M harfi eklenir. Concrete klasörüne Data katmanındaki model ile aynı özelliklerde modeller tanımlanır. Çünkü, bazı senaryolarda, entityler ile aynı özelliklere sahip classlara ihtiyaç olmaktadır. Entityleri sadece veritabanı işlemlerinde kullanırız ve sonradan karışıklık yaşamamak için bunları başka yerlerde kullanmamaya özen gösterilmesi gerekmektedir. Ayrıca dtoların da amacı farklı olduğu için, bunları dto adı altında oluşturup kullanmaya çalışmak da karışıklıklara sebep olacaktır. Bu sebeplerden dolayı entity ile aynı özelliklerde Concrete klasöründe classlar oluşturulur.

4. Service katmanında Abstract, Concrete, Helpers ve Validation olmak üzere 4 klasör bulunur. Bu proje yapısında aynı anda hem session hem de token kullanılmasını sağlayan özellikler eklendi. Örneğin /Home, /About, /Contact gibi linklere tıklamak istendiğinde, front end tarafında önceden java script kodu yazarak token göndermesini sağlayan bir fonksiyon eklemek gerekir. Böyle linklere gidebilmek ve bunu token validasyonu ile doğrulamaya çalışmak hem zaman hem de performans kaybı olabilir. Sadece bu senaryo için session kullanımı daha mantıklı olacaktır.

5. Service katmanında, sonradan kullanmak amacıyla Redis Cache Servisi eklendi ve Presentation katmanında da konfigürasyonlarda çalışması sağlandı. Eğer projeyi çalıştırmak isterseniz Redis Server'ın bilgisayarınızda kurulu ve çalışıyor olması gerekmektedir. Bunların yanı sıra Encryption, Token, Session, Student gibi önemli servisler de eklendi.

6. Validasyonlar için Fluent Validation kütüphanesi eklendi. Validation klasöründe hangi dto nesinesine validasyon işlemi yapılacak ise onlar oluşturularak, kuralları bunların içine yazılır.

```
2 references
public class StudentValidator:AbstractValidator<StudentDto>
{
    1 reference
    public StudentValidator()
    {
        RuleFor(x => x.UniqueId)
            .NotEmpty()
            .Length(11, 11);

        RuleFor(x => x.FirstName)
            .NotEmpty()
            .WithMessage("First name cannot be empty")
            .Must(x => !string.IsNullOrWhiteSpace(x) && x == x.Trim())
            .Length(2, 50);

        RuleFor(x => x.LastName)
            .NotEmpty()
            .WithMessage("Last name cannot be empty")
            .Must(x => !string.IsNullOrWhiteSpace(x) && x == x.Trim())
            .Length(2, 50);

        RuleFor(x => x.BirthDate)
            .LessThan(DateTime.Now)
            .WithMessage("Birth Date can't be greater than Today's date");

        RuleFor(x => x.PlaceOfBirth)
            .NotEmpty()
            .WithMessage("PlaceOfBirth cannot be empty")
            .Length(2,50);
    }
}
```