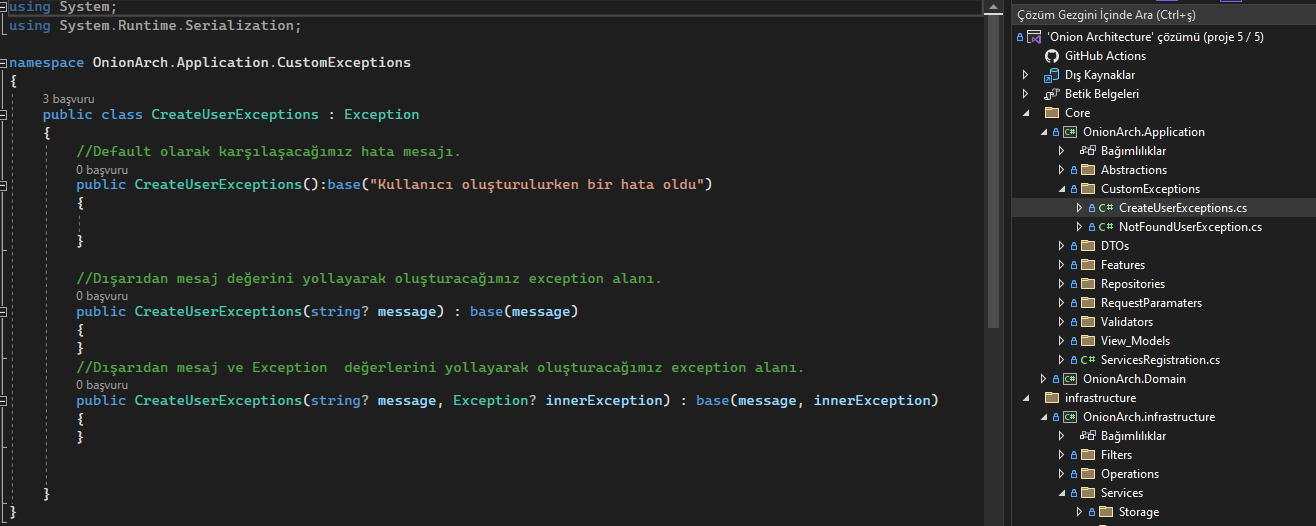
**Global Exception Handler Oluşturma**

Şimdi uygulamızda oluşan herhangi bir hatayı loglamak için yazılımcıların karşısına çıkarmak için biz ne yapıyorduk bir exception oluşturuyorduk. Örnek olarak , Core katmanı içinde oluştuduğumuz exceptionlara bir bakalım;



Gördüğünüz gibi burada bizim bir adet CustomException classımız mevcut.   
Şimdi burada olduğu gibi örnek olarak “Dosya eklenemedi”, “Dosya silinemedi” ,”Dosya bulunamadı” yada “Ürün eklenemedi” ,”Ürün bulunamadı” gibi durumlarada tek tek bu şekilde custom bir Exception oluşturamızın maliyet açısından ne kadar fazla olduğunu düşünebiliyor musunuz.

Bizim tek bir Exception classımız olsa ,   
Bu class üzerinden gerekli bilgileri mesela hata kodunu , hata mesajını vb. bilgileri girdiğimiz bir exception oluştursak tek bir exception içinden bu bilgileri düzelterek içeriğe müdahale edebilsek projemizim maliyeti ne kadar azalır düşünebiliyor musunuz ? Yani biz herhangi bir noktada tek tek hata mesajı oluşturmak yerine merkezi bir noktada bir exception oluşturmanın işimizi ne kadar kolaylaştıracağını düşünün bi onlarca belkide yüzlerce dosyadan kurtulacağız bu sayede değil mi ?

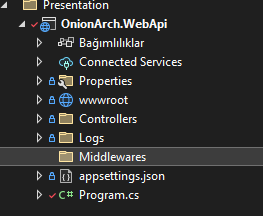
Şimdi ilk olarak burada bir önceki dersimizde yani Serilog ile loglama işleminden sonra burayı dikkate alırsanız sevinirim.

Şimdi ilk olarak ,

**app.UseExceptionHandler(); middleware’ini kullanacağız.**

Bunu ilk olarak aklımızda tutalım





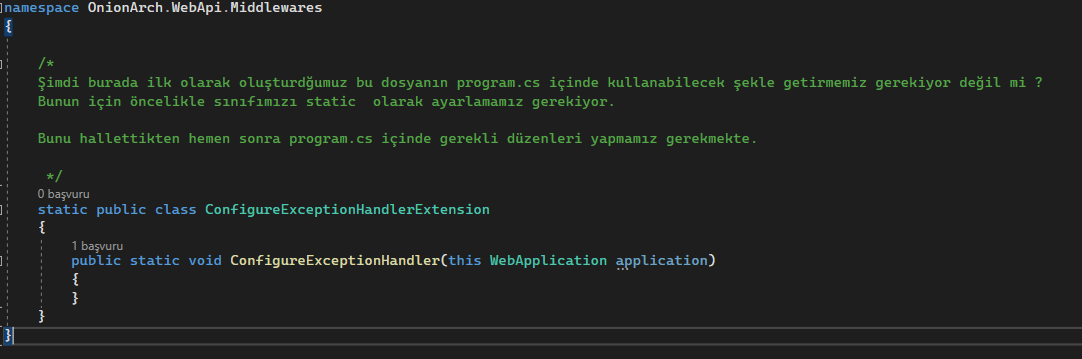
Program.cs üzerinden burada düzenelme yapabiliriz bu mümkün fakat,

best practices olarak bu çok tercih edilen bir yöntem değildir. Çünkü bize her zaman söylenen program.cs dosyasını her zaman termiz tutmamızdir.

İlk olarak program.cs üzerinden bu geliştirmeyi yapmak istemediğimizi bunu extension function olarak geliştirmemiz bizim için daha yararlı olacaktır.

**Bunun için Middlewares adında bir dosya oluşturalım program.cs ile aynı hizadada olsun .**

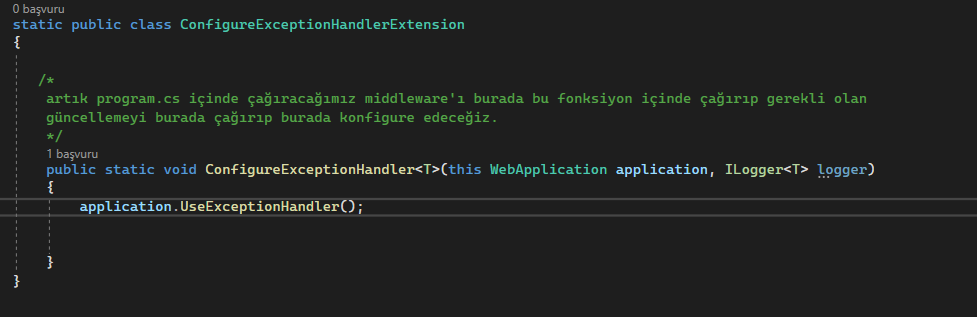
Daha sonra bu dosyanın içinde bir class oluşturalım ve içine bizim global olarak geliştirmek istediğimiz handler sınıfımızın metodunu extension olacak şekilde oluşturalım.



Şimdi dosyamızı oluşturduk bunu hemen program.cs içinde tanımlamamız gerekiyor.



Bu işlemi hallettikten hemen sonra gerekli düzenlemelerimize başlayabiliriz.



Artık program.cs içinde çağıracağımız middleware'ı burada bu fonksiyon içinde oluşturup gerekli olan

güncellemeyi burada yazacağız yani burada konfigüre edeceğiz.

Burada bulunan **this WebApplication application** sayesinede program.cs içinde olan düzenlemeleri entegre edebileceğimiz bir alan oluşturuyoruz..

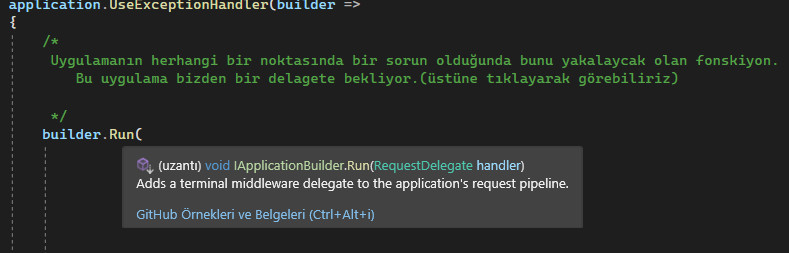
Ilogger<T> kısmında ise artık loglama işlemlerini burada yapabileceğiz.

Şimdilik bu log kısmını görmezden gelelim :)

Builder’ı parametre olarak vererek arrow fonksiyon olarak geliştirmelerimize başlayalım.

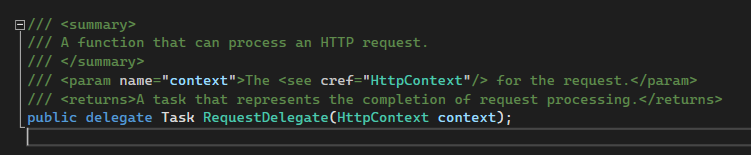
İlk olarak bizim uygulamada hataları yakalayan ve bu hatalara göre bizim işlem yapmamızı sağlayan fonksiyon üzerinden ilerlememiz gerekmekte.

Bu işlemi bizim için yapan **.Run** fonksiyonu ile ilerleyelim,



Şimdi bu fonksiyon bizden ne bekliyor bir adet Delegate hmm.

Peki bu delagete hangi parametreleri istiyor ona bakalım.



bir context istiyor bizden,  
Şimdi bu context’i kullanarak arrow fonksiyonlar aracılığıyla gerekli geliştirmeleri yapalım.

Yukarıda bulunan görselde bu bir task olacak bundan dolayı async olacak bunu unutmayalım.

Şimdi devam edelim tamam task olarak ayarladık ve bize bir context geliyor, peki biz ne yapıcaz diye sorabilirsiniz.



Şimdi burada artık biz uygulamada herhangi bir yerde bir hata olduğu zaman uygulamızın kullanıcıya, ne olursa olsun her türlü bir sonuç göndermesi gerekiyor değil mi ?

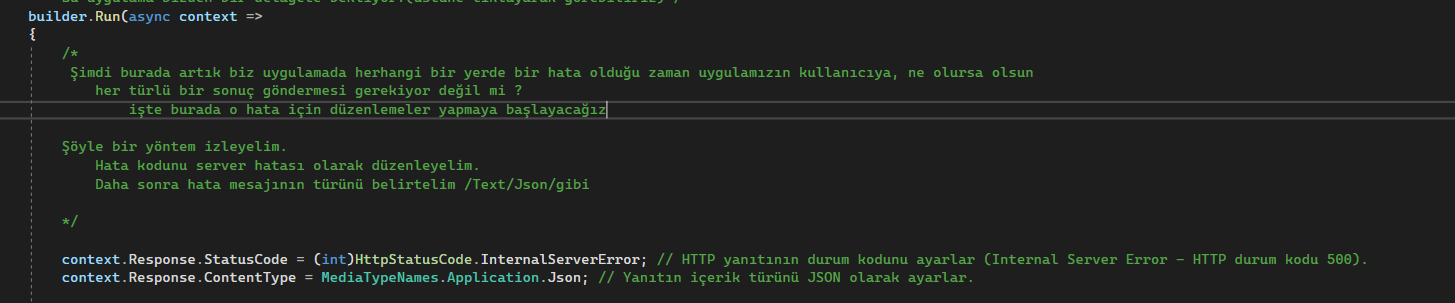
işte burada o hata için düzenlemeler yapmaya başlayacağız

Şöyle bir yöntem izleyelim,

context üzerinden

İlk olarak hata kodunu server hatası olarak düzenleyelim.

Daha sonra hata mesajının türünü belirtelim /Text/Json/gibi

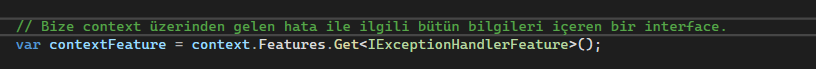


Şimdi uygulamanın herhangi bir noktasında bizimde ne olduğunu bilmediğimiz bir hata oluştu.

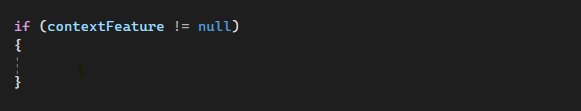
Bizim bu hatanın bilgilerine erişmemiz gerekiyor değil mi ?

İşte bu gibi durumda ilk olarak ,

context üzerinden bu hata ile ilgili bilgileri getirecek olan fonksiyona erişim sağlayalım.

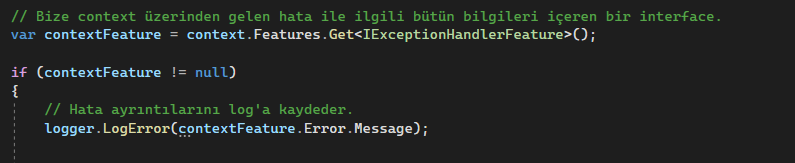


Daha sonra bu obje boş mu değil mi kontrol edelim.



Şimdi artık burada istediğimiz işlemi gerçekleştirebiliriz.

Mesela bir Log işlemi oluşturalım oluşan hatayı log olarak veri tabanına yada txt dosyasına yazsın.



Gördüğünüz gibi burada logger’ı kullandık.

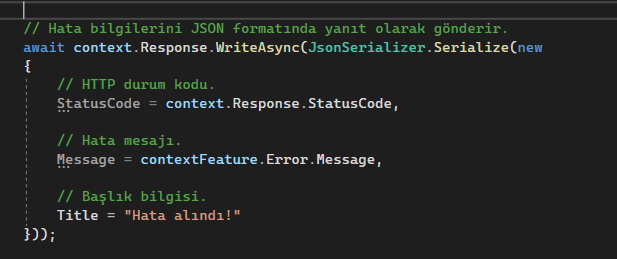
Şimdi kullanıcıya bizim bir sonuç döndürmemiz gerekiyor değil mi ?

Hata mesajını 500 yaptık.

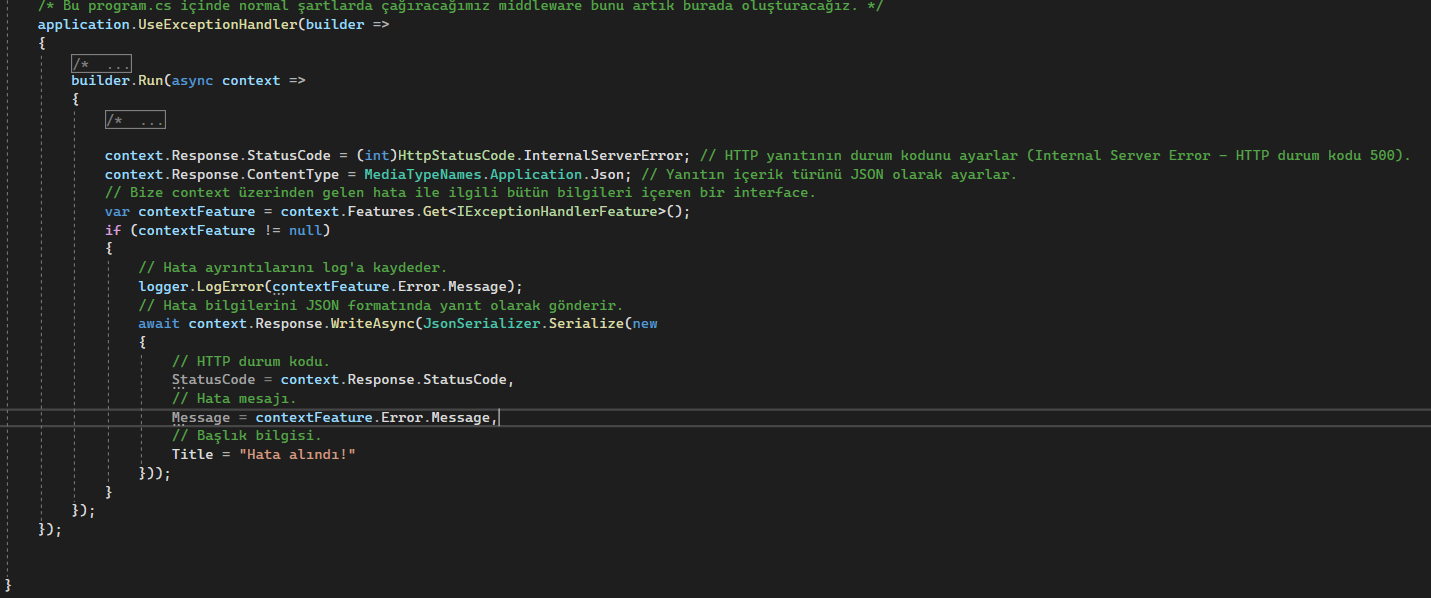
Gönderim Tipini belirledik.

Peki hata ile ilgili bilgileri nasıl vereceğiz ?

Response değerini değiştirerek.



Şimdi kodumuzun son haline bakalım.



Son olarak yapmamız gereken,

**program.c**s içinde yazdığımız methodu generic olarak çalışacak şekilde düzenlememiz.  
