**Serilog İle Loglama ve Seq İle Görselleştirme**

Log yapıları uygulamalarda bulunan en önemli yapılardan birisidir.

Uygulamanın runtime yani production aşamasında davranışının bize rapor edilmesi herhangi bir t anında neler yaşanmış neler oluyor gibi durumları bize söyleyen bir mekanizmadır.

**Neden önemlidir ?**

Bize kim saldırmak istiyor şüpheli ne gibi durumlar söz konusu nerede ne şekilde saldırılar yapılıyor gibi durumları gözlemleyebiliriz. Olası bir hatada anında hatanın nerede ne şekilde karşılaştığını görebiliriz.

**Log Mekanizması ne gibi ayrıntılar içermelidir ?**

* Sistemi kim kullanıyordu ?
* Sistemin içinde arıza nerede gerçekleşti ?
* Hata Kodu ? Hata ile ilgili kodların ne olduğunu çözümleyebiliriz
* Ne zaman olduğuna dair bir tarih bilgisi (EN AMA EN ÖNEMLİ).
* Uygulama neden başarısız oldu ?

**Log Mekanizması ne gibi bilgileri kayıt altına almalıdır?**

* Alınan Exception bilgileri.
* Veri değişiklikleri .(Kritik olan veriler tanımlama verileri , Banka verileri , yazılım için kritik arz eden veriler yani kullanıcıların verileri değil )
* Şüpheli etkinlikler (Kimlik doğrulama isteğinin durmadan başarısız olması gibi,hacking olayları vs,Kısıtlı olan verilere erişme çabası kim erişmek istiyor ne kadar erişmek istiyor vs vs, Geçersiz parametreler geliyor ise)
* Requestler: Kullanıcılan gelen tüm istekleri kayıt etmemiz gerekiyor ki kullanıcı sistemde neler almış sisteme neler vermiş bunların hepsini bilmemiz gerekiyor. Tarih ve Saat olarak .
* Kullanıcı bilgierini tutmamız gerekiyor ki hangi kullanıcı ne yapmaya çalışıyor bunları inceleyebilelim.
* Kısa bir açıklama.(Burada )

**Loglara asla ama asla hassas veriler kullanıcı şifrleri , kullanıcı bilgileri vs vs gibi bilgiler asla ama asla kayıt altına alınmamalı.**

**Log nerede ne nasıl tutulmalı?**

Loglar console üzerinden tutulur genelde.

Fakat burada tarihe göre filtreleme kullanıcı için runtime üzerindeki davranışları takip etmek console üzerinden oldukça zordur.

Bundan dolayı bunları ya harici dosyaların içinde yada veri tabanlarında fiziksel olarak saklanmaları gereklidir.

Şimdi uygulamamızda Log mekanizması için **SERİLOG** kütüphanesini kullanacağız.

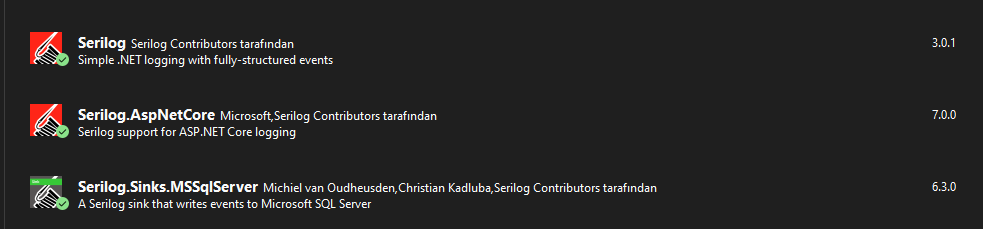
Serilog sayesinde uygulamamızın içinde bulunan logları => Console, File, VeriTabanı vs gibi yöntemler kullanarak kolayca aktarmamızı sağlayan açık kaynak bir kütüphanedir.

Şimi kendi projemiz üzerinden bu işleme başlayalım.

Öncelikle biz loglama işlemlerini nerede yapacağız.? => Tabiki de **Api** uygulamamızda.

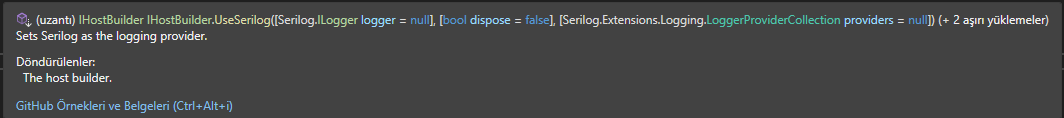
İlk olarak kurumlara başlayalım.

Ekte olan 3 paketi projemize kurmanız gerekmekte biz MSSQL kullandığımız için MSSQL olan paketi ekliyoruz eğer ihtiyaç dahilinde Postgresql gibi sql serverlar kullanıyorsanız buna özel paket yüklemeniz gerekmektedir.

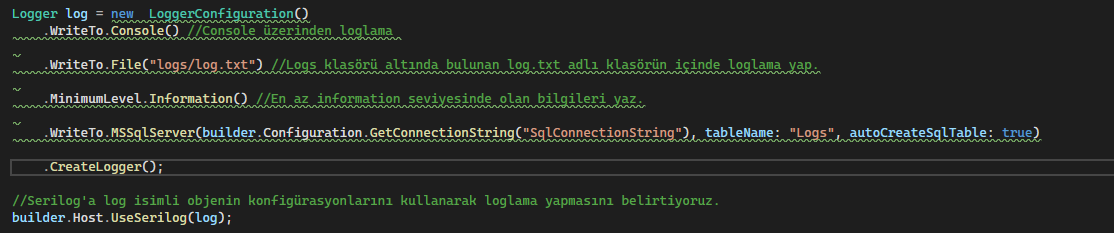


Daha sonra Program.cs içine girip gerekli olan konfigürasyonları yapmamız gerekmekte.

İlk olarak ;

Serilog bizden ne istiyor ona bakalım gördüğünüz gibi bir adet Logger türünde bir referans istiyor.   


| Şimdi burada önemli bir nokta söz konusu bu Logger nesnesi bir LoggerConfiguration instance yani örneğini oluşturuyoruz.  Fakat oluşturduğumuz bu örnek’de LoggerConfiguration oluşuturuken kullandığımız Logger ifadesinin Serilog.Core’dan gelmesi gerekiyor. |
| --- |
| Daha sonra oluşturduğumuz bu örneği UseSeriLog’a bildirerek artık bu konfigürasyon dosyaları kullanarak  loglamaları yap diyoruz. |

****

SeriLog yapılanmasında WriteTo ile başlayan tüm methodlar nereye loglama yapılacağını bildirim.

WriteTo.Console() =>Console üzerinde loglama yap

WriteTo.File(“.../….txt”) => … isimli klasör içinde …txt dosyası içinde loglamaya yap.WriteTo.MSSqlServer()

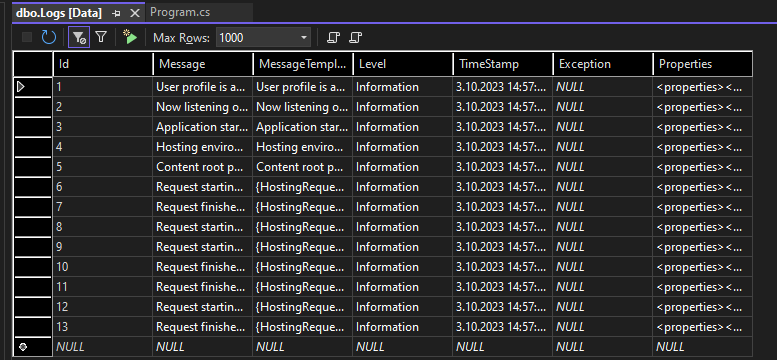
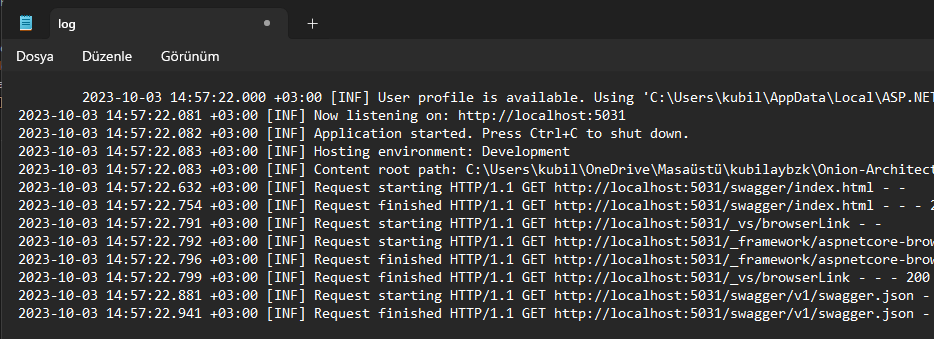
=> Bu biraz özel yapılara ihtiyaç duyuyor.

=>Bir adet connection string

=>Bir adet tablo adı .

=> Tablo otomatik mi olsun yoksa geliştirici kendisimi oluşturacak .

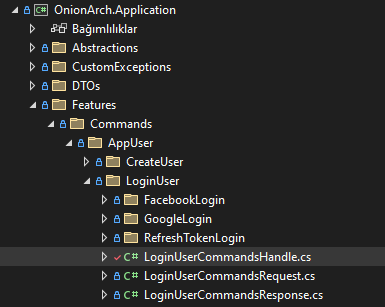
Şimdi bu şekilde projemizi çalıştıralım ve dosyamız oluşmuş mu veri tabanımızda tablo oluşmuş mu bi göz atalım.

****

Görüldüğü gibi hem tablomuz başarıyla oluştu hemde dosyamız .

Şimdi basit bir şekilde bu geliştirdiğimiz Logger ‘ı kullanrak kendimiz nasıl log oluşturabiliriz ona göz atalım.

Öncelikle bizim Exception fırlattığımız bir kodumuz olsun bu örnek olarak Login işlemine göz atalım.

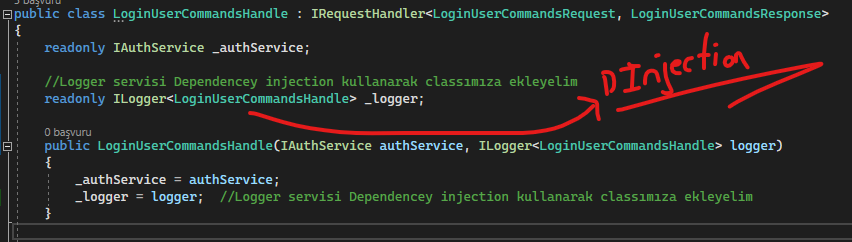
Bunun için öncelikle yapmamız gereken işlem,   


LoginUserCommandsHandle dosyamıza geçiş yapmak bunun için ilk olarak

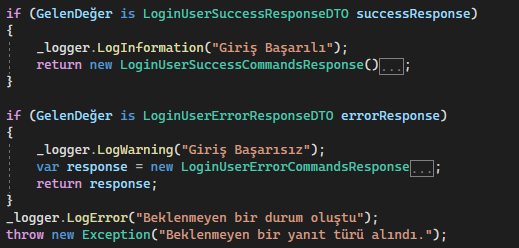
Application katmanı içinde bulunan

Features => Commands=>AppUser=> LoginUserCommandsHandle.cs dosyamına geçiş yapalım.

İlk olarak Dependency Injection yöntemi kullanarak gerekli olan implementasyonlarımızı gerçekleştiriyoruz.

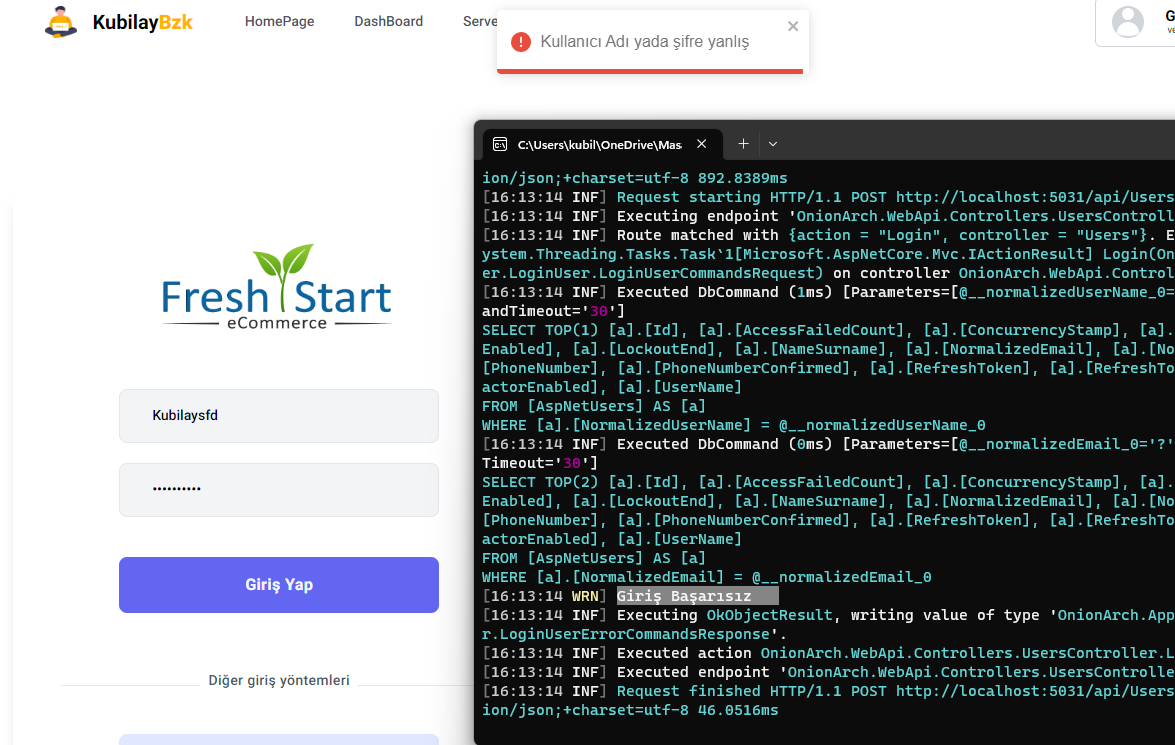


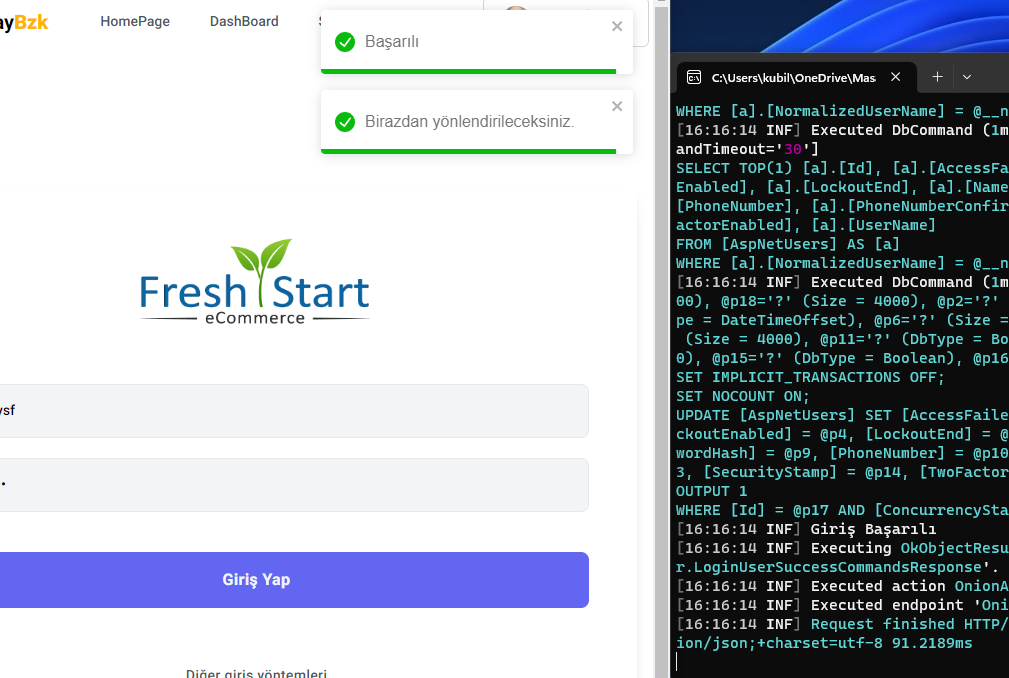
Daha sonra loglama işlemine oluşturduğumuz \_logger değişkenini kullanarak gerçekleştirebiliriz.



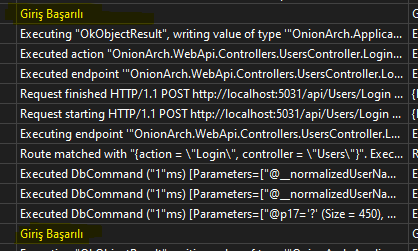
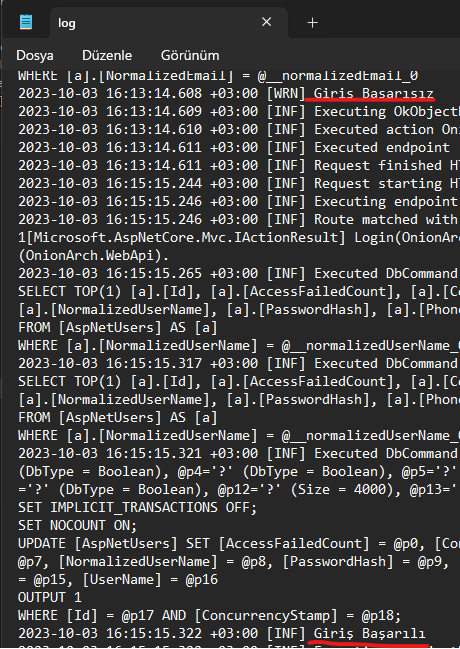
Görüldüğü gibi 3 farklı durum için bir loglama işlemi gerçekleştirdik.

Şimdi projemizi çalıştıralım ve , bu gerçekleştirdiğimiz işlemler için sonuçlara konsol txt dosyamız ve veri tabanımıda inceleyelim.





Görüldüğü gibi iki farklı durum için iki farklı sonucuda başarı ile aldık . Veri tabanını ve txt dosyalarına bakalım.



Şimdi biraz daha advance konulara geçiş yapalım.

Tablomuz içine ek kolon eklemek isteyelim.

Yada tablonun içinde olan kolonların isimlerini değiştirmek isteyelim.

**Tabloya kolon ekleme yada tablodaki kolonların isimlerini değiştirme .**

ilk olarak tablomuzun konfigürasyonunu yaptığımız yerde bunun için SeriLog tarafından hazırlanan bir alan mevcut olduğunu artık tahmin ediyoruz 🙂

İlk olarak bu alanı yada metodu kullanarak bu geliştirmeyi nasıl yaparız buna göz atalım.

**Öncelikle bu durum MSSQL yada Postgres uzantılarına göre değişiklik göstemekte.**

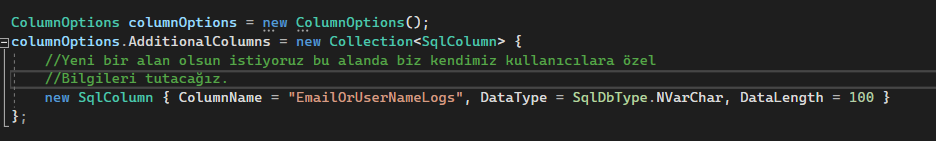
İlk olarak tablomuza bir alan eklememiz gerekmekte . Örnek olarak Postgress için geliştirilen Nugget kütüphanesinde ek bir alan tanımlamak istersek bu alan için öncelikle daha önce eklenti tarafındanda tanıtılmış olan alanları ekleyerek bu işleme başlıyor olmamız gerekir. Örnek olarak ;



Biz projemizde MSSQL kullandık bundan dolayı bizim ayarlarımız biraz daha farklı olacak, biz tablomuzda kullanıcıların adlarını tutmak istiyor olalım.

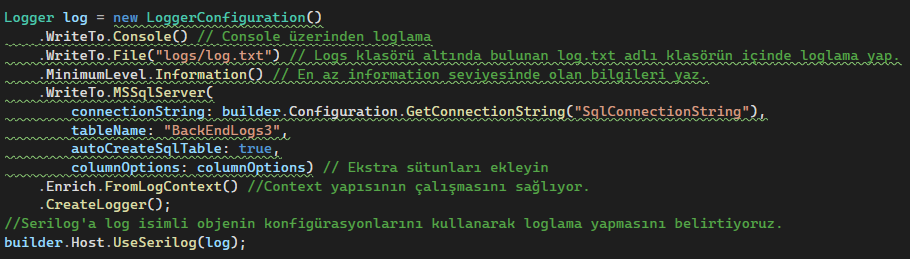
Öncelikle tablomuza kullanıcı adı gibi bir alan eklememiz gerekmekte.

Bu işlem için **AdditionalColumns** özelliğini kullanacağız.



Gördüğünüz gibi artık tablomuz içinde sql komutlarını kullanarak bir alan ekledik.

Daha sonra bu eklediğimiz yeni alanı Logger içinde tanımlamamız gerekmekte..



Bu alan bizim kullanıcı adı bilgilerimizi tutacak.

Şimdi projemizi çalıştıralım ve tablomuzu inceleyelim.



Tablomuzda gördüğünüz gibi istediğimiz alan oluştu.

Şimdi burada biz her istek gerçekleştiğinde bu alanın hangi kullanıcı tarafından aksiyon alınıp sonucun loglandığına göz atmak istiyor olalım.

İlk olarak burada amacımız bir şekilde kullanıcıdan istek aldığımız o kullanıcının bilgilerine erişme şansımız olsun Hmm bunu nasıl yapabiliriz aslında çok basit biz Authantication Authorization işlemlerini uygularken JWT kullanmaya başladık değil mi

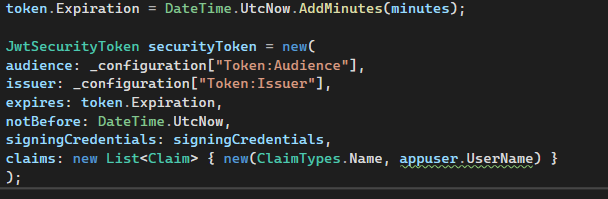
Bu JWT oluşurken claim denen bir özelliğinden yararlanabiliriz.

| JWT'nin "claim" adı verilen bir kavramı vardır. Bir JWT içindeki "claim"lar, taşınan verinin belirli özelliklerini ve bilgilerini tanımlar. "Claim"lar, JWT'nin yük (payload) bölümünde bulunur ve JSON formatında ifade edilir. JWT'deki "claim"lar, JWT'nin amacına ve kullanımına göre farklılık gösterebilir. |
| --- |

Bunun için öncelikle JWT oluşturduğumuz sınıfımıza geçiş yapmamız gerekmekte.

Biz JWT ayarlarımızı infrastructure katmanında Services klasöründe bulunan TokenHandler içinde yapmaktayız.

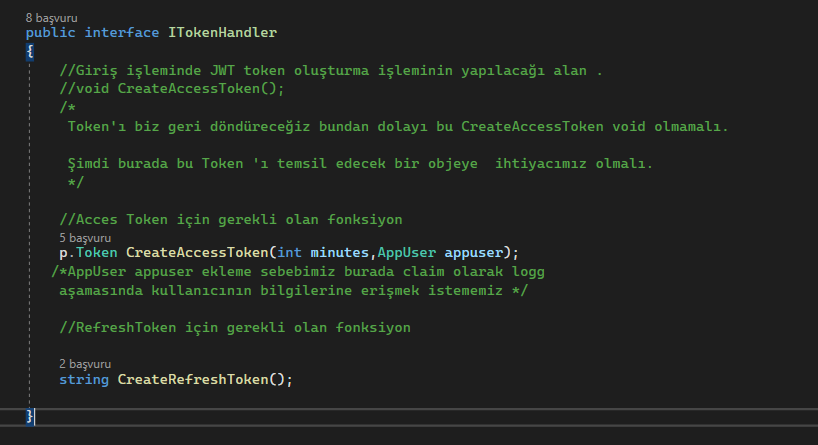
Şimdi bahsi geçen dosya geçiş yapalım ve JWT ayarlamalarımıza ulaşalım.



Görüldüğü gibi buraya claims adında yeni bir property tanımladık. Bu property bizden bir liste istiyor Key,Value olacak şekilde burada Key değerine Name Value değerine ise AppUser içinde bulunan UserName ismini verdik .

Hmm burada biz Appuser’a nasıl erişeceğiz dediğinizi duyar gibiyim aslında çok basit burada oluşturduğumuz bu methodumuzun içine kullanıcıyı parametre olarak vereceğiz.

Bunun için öncelikle interface üzerinde düzenleme yapmamız gerekmekte.Bunun sebebi projemizde Generic Repository Design Pattern kullanmamız.

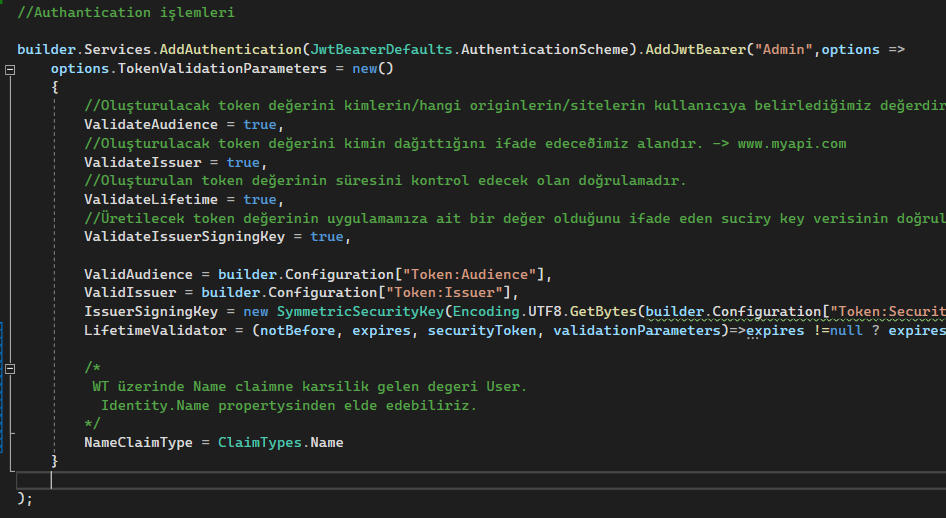


Şimdi bu düzenlemeyi yaptıktan sonra kodumuzda CreateAccessToken ‘ı kullandığımız her alanda AppUser appuser şeklinde parametreleri kullanarak user bilgilerini vermemiz gerekmekte.

Bu işlem oldukça basit olduğu için gösterme gereği duymadım.

Bu düzeltmeleri hallettikten sonra yapmamız gereken bu JWT çözülürken bu Name değerin neye karşılık gelecek onu ayarlamamız gerekmekte . Bunun için Program.cs içinde token ayarlarına geçiş yapmamız gerekmekte.

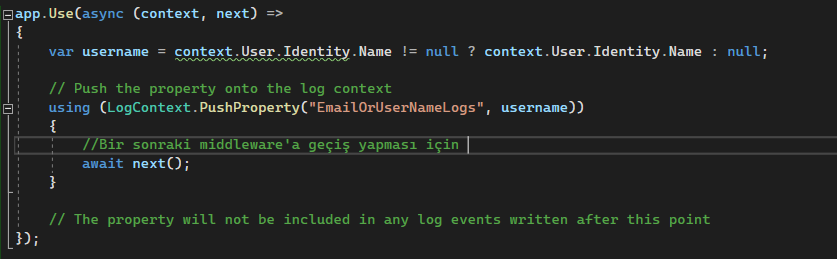
Token ayarları üzerinde NameClaimType adında bir property üzerinden belirtiyoruz.



Şimdi her istekte bu değeri bizim loglama esnasında veri tabanına kayıt etmemiz gerekmekte .

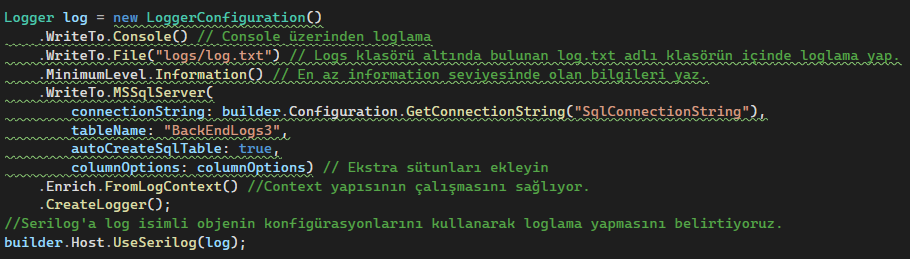
Bu işlem için middleware denen yöntemi kullanabiliriz.

| .NET Core'da bulunan middleware, ASP.NET Core uygulamalarında HTTP isteklerini işlemek ve yanıtlamak için kullanılan bir tür bileşendir. Middleware, isteklerin uygulama üzerinde nasıl işlendiğini ve yanıtlandığını kontrol etmek için kullanılır. Bir ASP.NET Core uygulaması, bu middleware bileşenlerini isteğe göre sıralayabilir ve yapılandırabilir. |
| --- |



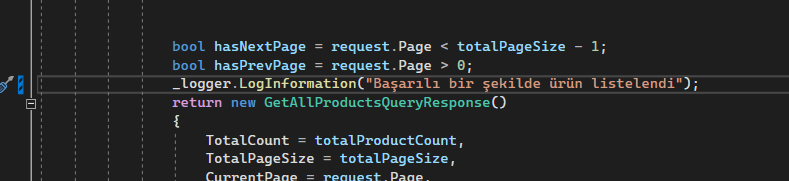
Bu middleware’i kullanarak gerekli işlemi gerçekleştirebilmesi için bunu LoggerConfiguration içined

**.Enrich.FromLogContext()** tanımlamamızı yapıyoruz.

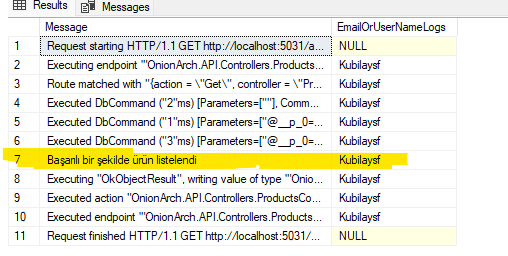


Şimdi bir test uygulayalım.

Ürünlerin hepsinin listelendiği CQRS kullandığımız handler alanına ufak bir log ekleyelim .

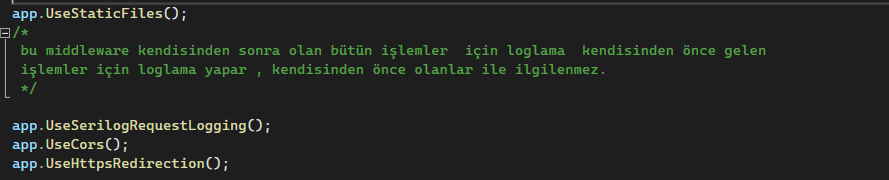


Şimdi Api’ı tetikleyelim ve sonuca göz atalım..



Şimdi ek olarak **UseSerilogRequestLogging** adında bir middleware’de mevcut bu middleware ise ,kendisinden sonra olan bütün işlemler için loglama kendisinden önce gelen

işlemler için loglama yapar , kendisinden önce olanlar ile ilgilenmez.

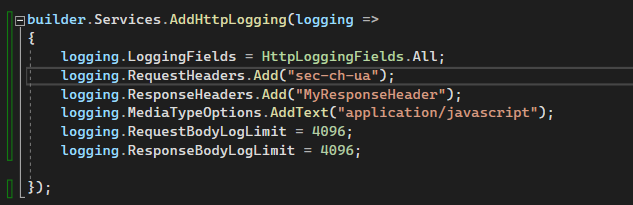


**HTTP Logging mekanizması .**

Şimdi bu mekanizmanın bize gelen istekte kullanıcının client bilgilerini getireceği bir loglama verisidir.

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/http-logging/?view=aspnetcore-7.0>

Bunun için üst tarafta olan makaleden yararlanabiliriz.



Bunu program.cs içinde kendi oluşturduğumuz loging mekanizmasının hemen altına oluşturalım.

Daha sonra bu servis için bir middleware ekleyelim program.cs içine

