**Kullanılan Araçlar**

* Visual Studio 2022
* SQL Server Management Studio 2019
* Docker Desktop

**Kullanılan Kütüphaneler**

* Microsoft.AspNetCore.Authentication.JwtBearer (8.0.8)
* Serilog.AspNetCore (8.0.2)
* Swashbuckle.AspNetCore (6.4.0)
* Microsoft.Extensions.Configuration.Abstractions (8.0.0)
* Microsoft.Extension.DependencyInjection.Abstractions (8.0.1)
* StackExchange.Redis (2.8.12)
* Microsoft.EnitityFrameworkCore (8.0.8)
* Microsoft.EnitityFrameworkCore.SqlServer (8.0.8)
* Microsoft.EnitityFrameworkCore.Tools (8.0.8)
* Microsoft.Extensions.Configuration (8.0.0)
* Microsoft.Extensions.Configuration.FileExtensions (8.0.1)
* Microsoft.Extensions.Configuration.Json (8.0.0)
* RabbitMQ.Client (6.8.1)
* Microsoft.Extensions.Hosting (8.0.0)

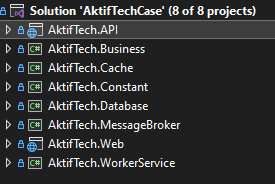
**Gerçekleştirilmiş Talepler**

Case dokümanında belirtilen başlıkların detaylarında belirtilen taleplerin tamamı gerçekleştirilmiş olup aşağıda başlıklar hakkında özet bilgi verilecektir. Daha sonra ise geliştirmeler hakkında detaylı açıklamalara yer verilecektir.

1. **Technology Stack:** .NET Core kullanılmış olup .NET 8 versiyonu seçilmiştir.
2. **Rest API Design:** Belirtilen endpointler oluşturulmuş olup kaynak adları RESTful mimarisine uygun bir şekilde belirlenmiştir.
3. **CustomerOrder Model:** Customer, Product, CustomerOrder gibi model tanımlamaları yapılmış olup validation kontrolleri yapılmıştır.
4. **API Endpoints:** Gerekli endpointler tanımlanmış olup CustomerOrder için gerekli işlemlerin yapılabilmesi sağlanmıştır.
5. **Security:** API düzeyinde kimlik doğrulama işlemi için JWT kullanılmıştır. Ayrıca Password gibi önem düzeyi yüksek veriler için şifreleme işlemi gerçekleştirilmiştir.
6. **Database and ORM:** Database olarak MSSQL kullanılmış olup ORM aracı olarak EF Core kullanılmıştır.
7. **Redis Caching:** StackExchange.Redis kütüphanesi kullanılarak Product listesi cahcelenmiştir.
8. **Logging:** Serilog kütüphanesi kullanılarak loglama işlemi gerçekleştirilmiştir.
9. **RabbitMQ Integration:** Tamamlanmış siparişlerin mesaj bilgisi bir RabbitMQ kuyruğuna aktarılmış olup, arkaplanda çalışan bir Worker uygulaması tarafından dinlenerek kuyruktan gelen mesajlar bir .txt dosyasına yazılmıştır.
10. **API Documentation:** Endpoint ve uygulama dokümantasyonu hazırlanmıştır.

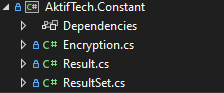
**Uygulama Hakkında Genel Bilgiler**

Projemizde Çok Katmanlı Mimari kullanılmıştır.



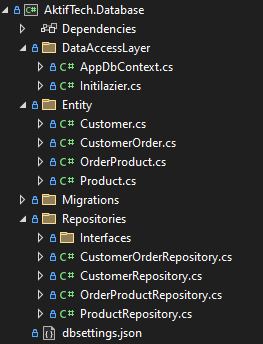
Bu katmanları mantıksal sıralama ile birlikte inceleyeceğiz. API katmanında örnek istekler ve cevaplar ile Case dokümanında talep edilen isteklerin gerçekleştiğini gözlemleyeceğiz.

**1-) Constant Katmanı**

****

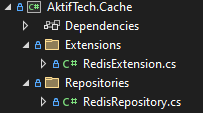
Bu katmanda tüm projede kullanabileceğim sınıfları oluşturdum.

**2-) Database Katmanı**

****

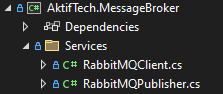
Entityleri tanımladıktan sonra DbContext sınıfından türettiğim AppDBContext sınıfı ile veritabanı işlemlerini gerçekleştirdim. SqlConnectionString bilgisini dbsettings.json dosyasında tuttum. Bu katmanda Repository Design Pattern kullanarak repository işlemlerini yaptım.

**3-) Cache Katmanı**

****

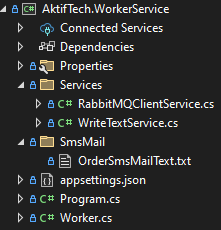
Bu katmanda Product listesini tutacağım Redis kaynağı için gerekli tanımlamalar ve Redis işlemleri yapıldı.

**4-) MessageBroker Katmanı**



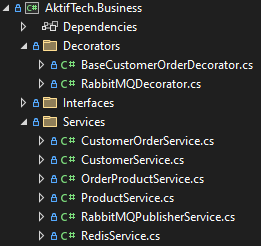
Bu katmanda tamamlanan sipariş sonrası gönderilecek olan mail/sms mesajının bir RabbitMQ kuyruğuna gönderilmesi için gerekli işlemler yapıldı.

**5-) WorkerService Katmanı**

****

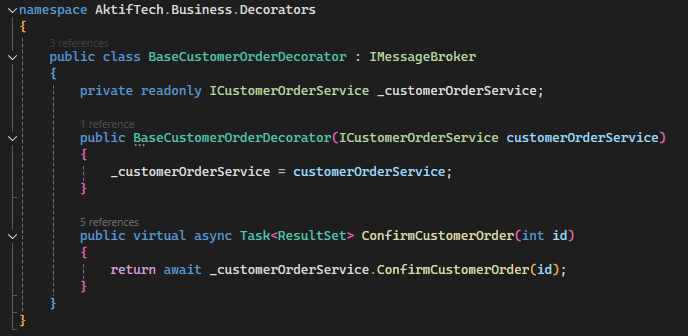
Bu katmanda RabbitMQ kuyruğunu dinleyen bir Worker uygulaması hazırlandı. Bir background servis gibi çalışan uygulama gelen mesajı bir .txt dosyasına yazdı.

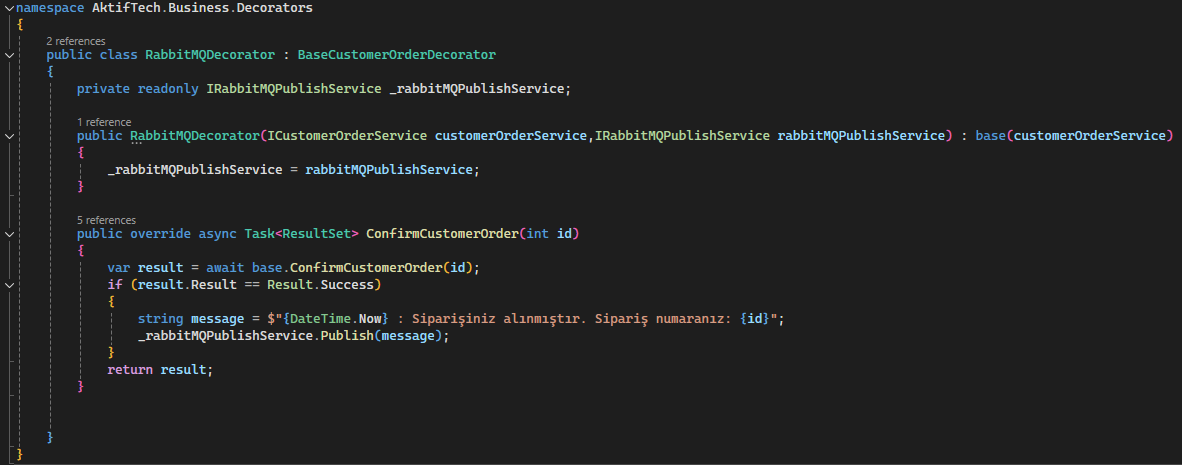
**6-) Business Katmanı**

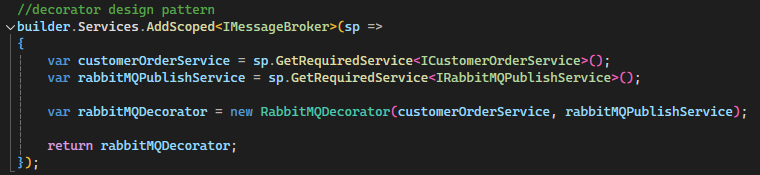
****

Bu katman, katmanlar arası haberleşmeyi ve logic işlemlerin yapıldığı bir katmandır. Örneğin API-database haberleşmesi, database yazılacak verilerin logic işlemleri (stok kontrolü gibi), API - Redis haberleşmesi gibi.

Projenin tamamında olduğu gibi loosely coupled kodlar yazmak bu katmanda da hedeflenmiştir. Buna en iyi örnek RabbitMQ çalışmasının Decorator Design Pattern kullanılarak yapılması gösterilebilir.

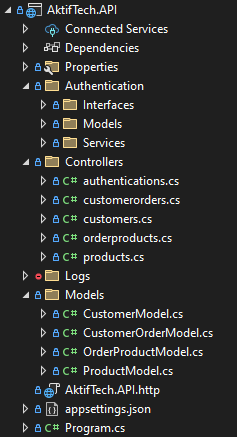






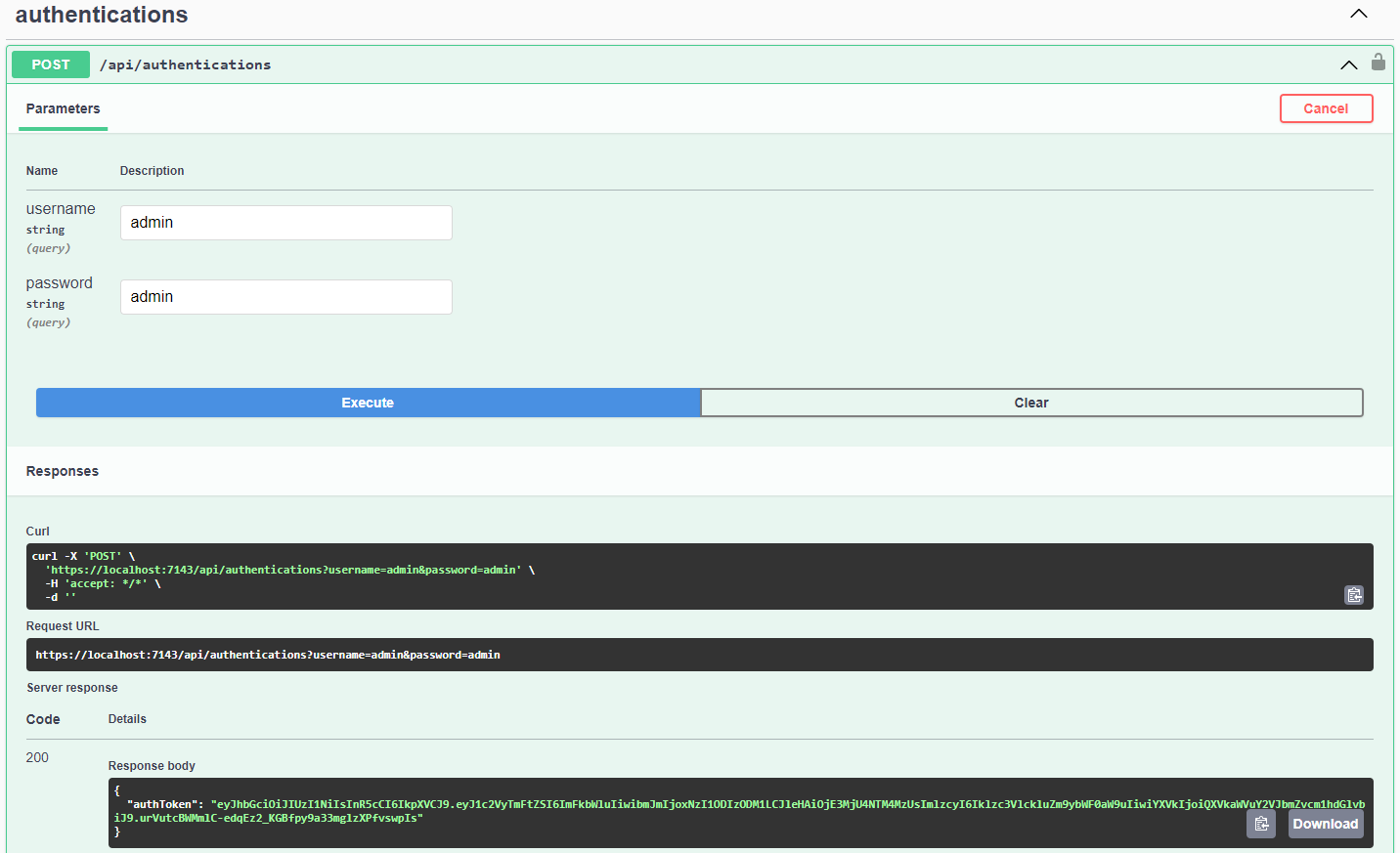
Burada amaç birbirine bağlı olmadan iki sınıfın da işlemlerini gerçekleştiriyor olabilmelerini sağlamaktır. Burada önce repository işlemi sonrasında RabbitMQ işlemi gerçekleştirilmektedir. İlerde gelebilecek üçüncü bir özellik için bu design pattern kullanılarak yeni gelecek sınıfın bu sınıflara bağımlılığı olmayacaktır. Bu da bize yapıyı bozmadan projenin büyümesine yardım edecektir.

**7-) API Katmanı**

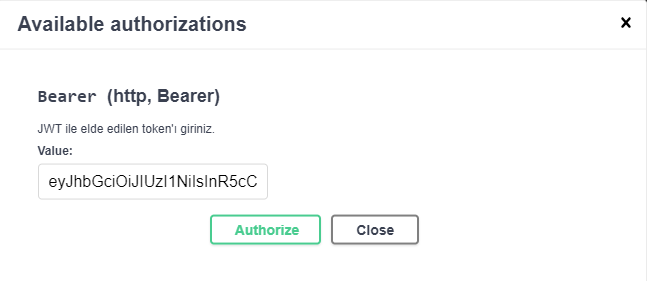


Bu katmanda dış dünya ile haberleşecek olan kaynaklarımız ve endpointlerimiz mevcuttur. Kaynak ve endpoint isimlendirmeleri RESTful mimari standartlarına göre yapılmıştır. Data Transfer Object (DTO) olarak uygun modeller kullanılmıştır. Validation işlemleri gerçekleştirilmiştir. Authentication için JWT kullanılmış olup bu işlem için yine modüler bir yapı kurulmuştur. JWT, Serilog, Redis, RabbitMQ configurasyonları gibi bir çok ayar appsettings.json dosyasında tutulmuştur.

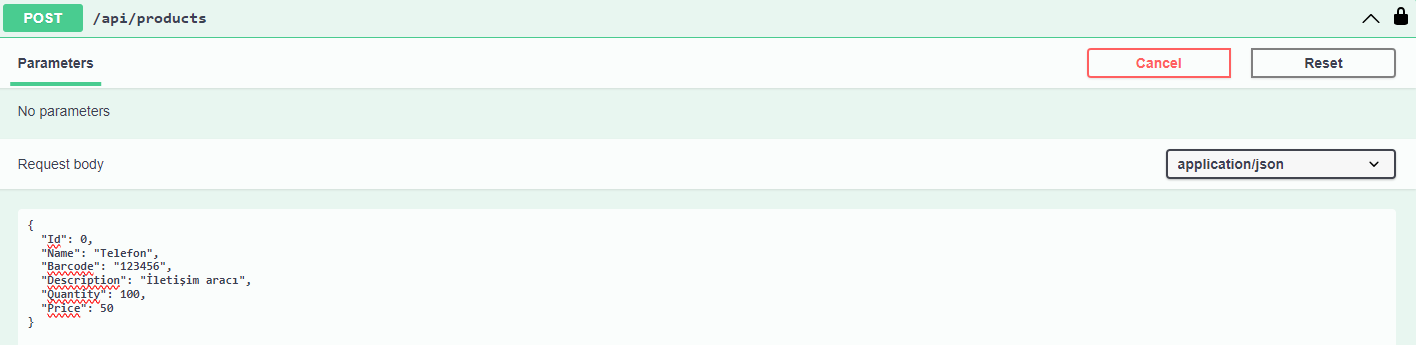
Sadece case dokümanında talep edilen endpointler değil, hem gerekli hem de gerekebilecek endpointler oluşturulmuştur. API dokümantasyonu ayrı bir dosya içinde mevcuttur. Endpointlere erişebilmek için öncelikle authentication işlemi gerçekleştirmemiz gerekiyor. (Web uygulamasında kullanılan bazı özel endpointler istisna tutulmuştur, detaylı açıklaması web katmanında yapılacaktır) **username** ve **password** için “**admin**”, ”**admin**” kullanılmaldır.

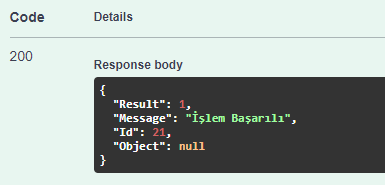


Buradan elde ettiğimiz token’ı swaggerda yer alan Authorize butonu ile açılan inputa yazacağız.

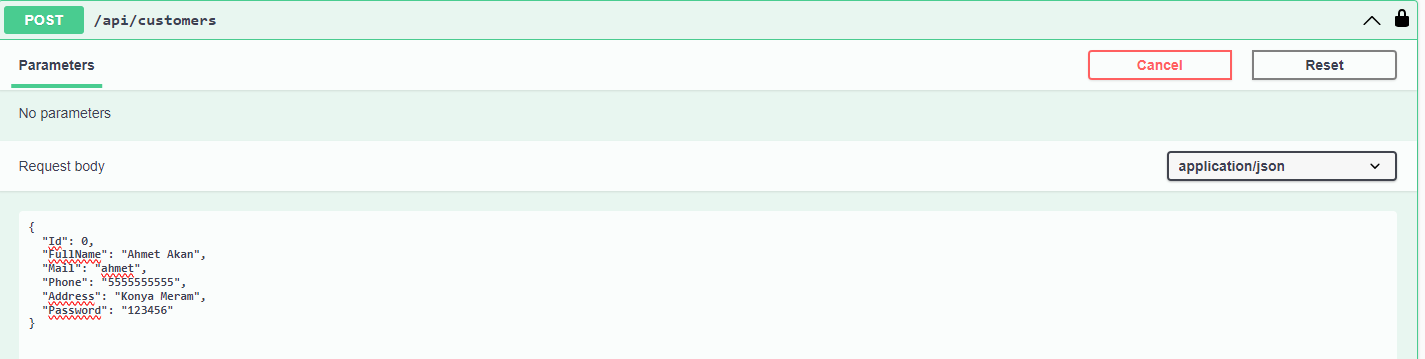


Artık erişim yetkisi isteyen endpointlerimize ulaşabiliriz. Şimdi sırayla bir Product, Customer, CustomerOrder oluşturalım.



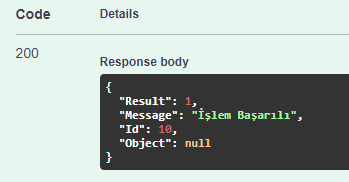


Id’si **21** olan bir **Product** oluşturduk.

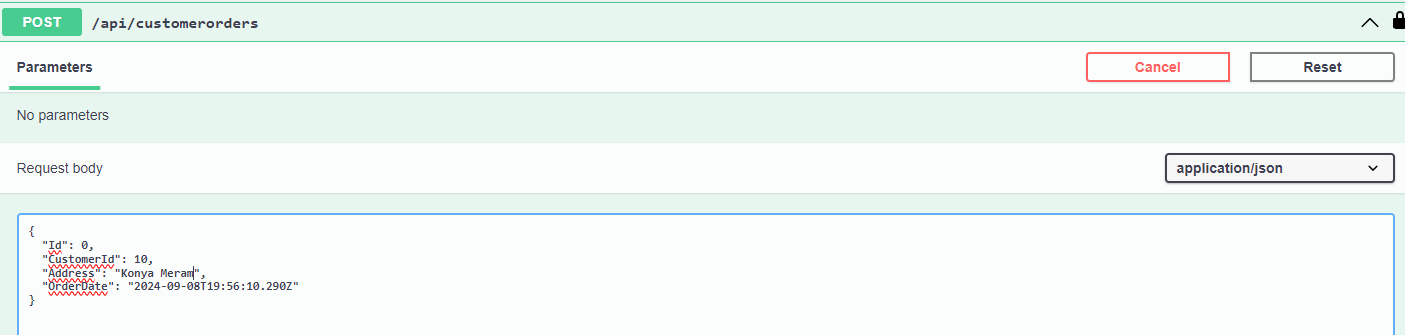


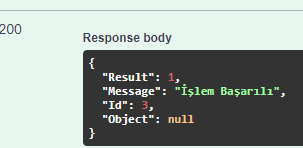


Mail adresi geçersiz olduğu için işlem gerçekleştirilemedi. Geçerli bir mail adresi ile tekrar istek atalım.

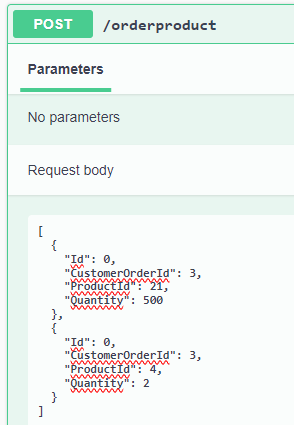


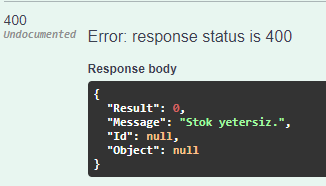
Id’si **10** olan bir **Customer** oluşturduk.



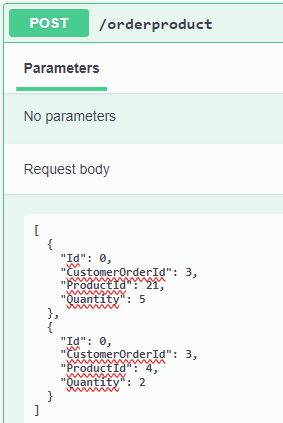


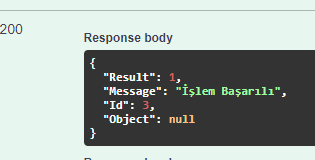
Id’si **3** olan bir **CustomerOrder** oluşturduk. Artık bu order’a productlarımızı ekleyebiliriz. Önce liste halinde bir product ekleyelim.



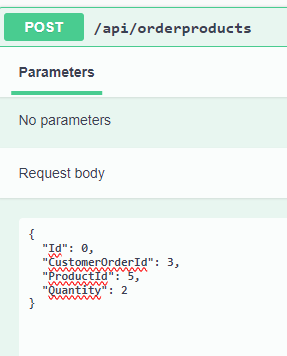


İki adet ürün ekledik ama bir tanesi stok miktarında fazla adette eklendiği için işlem başarısız oldu.



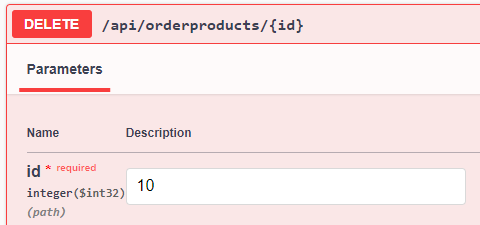


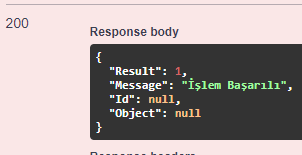
Bu sefer işlem başarılı oldu. Şimdi siparişe tek bir ürün ekleyelim.



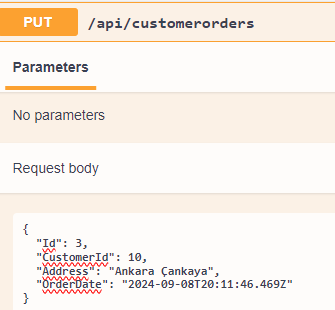


CustomerOrder’a yeni bir ürün daha ekledik. Id’si 10 olan bu OrderProduct’ı şimdi CustomerOrder’dan silelim.



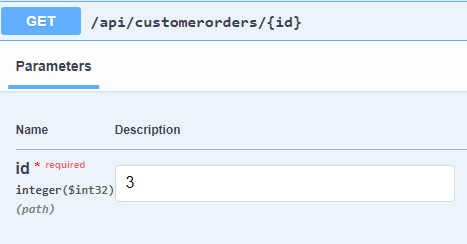


İşlem başarılı. Şimdi CustomerOrder’ın adresini güncelleyelim.



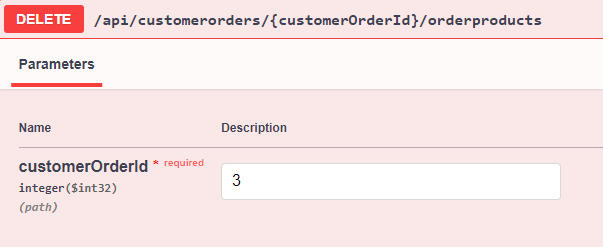


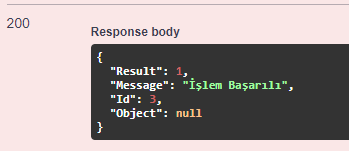
Şimdi CustomerOrder’ın son halini görelim.



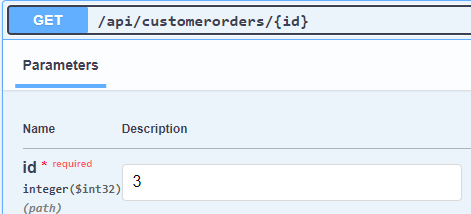


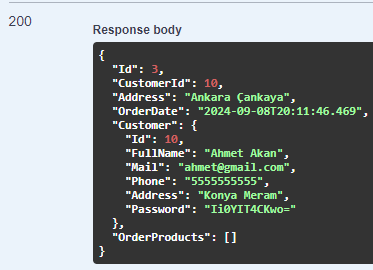
Şimdi ise bu CustomerOrder’daki tüm ürünleri tek seferde silelim.



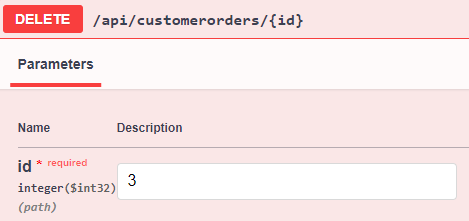


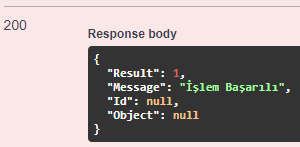
Şimdi tekrar son durumuna bakalım.





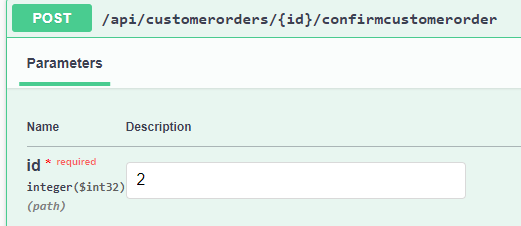
Ürünlerin tek seferde silinmiş olduğunu gözlemledik. Şimdi ise CustomerOrder’ı silelim.

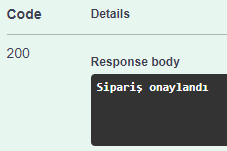


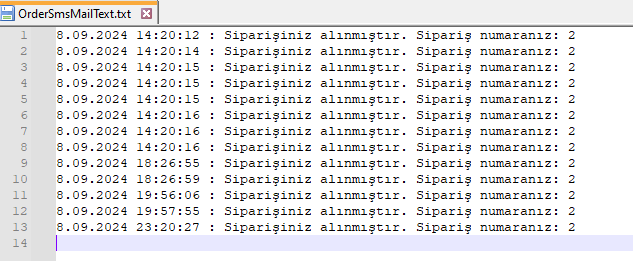


Silme işlemi de başarılı bir şekilde sonuçlandı. Böylelikle case dokümanında yer alan CustomerOrder ile ilgili tüm istekler gerçekleştirilmiş oldu.

Şimdi daha önce yine aynı istekleri kullanarak oluşturmuş olduğum bir siparişi tamamlayalım ve RabbitMQ ile .txt dosyasına yazma işlemini inceleyelim.

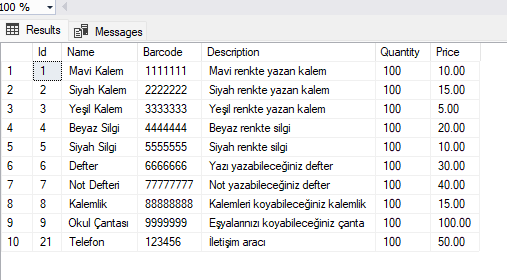


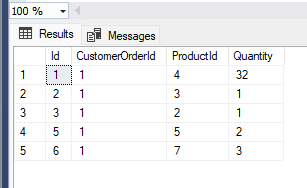


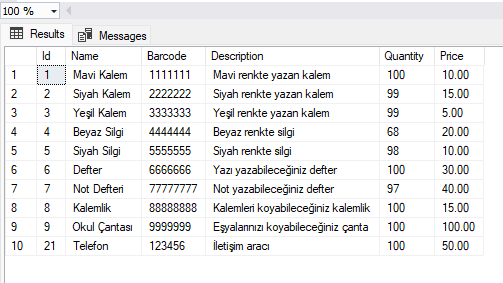


Ayrıca sipariş onayı sonrası stokların azalması işlemini de gözlemeyebiliriz.(Başka bir sipariş için)

Onay öncesi

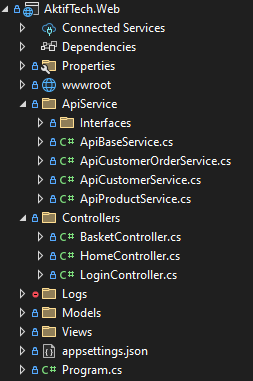




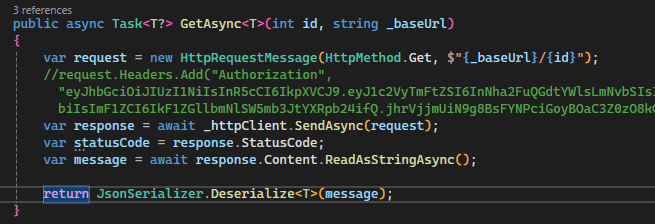


CustomerOrder içinde bulunan Product adetleri kadar kendi stok(Quantity) adedinden düşüm sağlanmıştır.

**8-) Web Katmanı**

****

Bu katman case dokümantasyonunda yer almasa da iş görüşmesindeki konuşulan ön yüz tasarımı sohbetinden yola çıkılarak hazırlanmıştır. Basit bir müşteri login sayfası, ürünleri listeleyen bir anasayfa, sepete ekleme işlevi ve sepetim sayfası hazırlanmıştır. Buradaki tüm repository işlemleri API ile haberleşerek yapılmıştır. Bu projeyi eklememin en büyük sebebi Web-API haberleşmesini göstermek içindir. Bu yüzden bu projeden erişim isteği yapılan endpointlere erişim yetki kontrolü konulmamıştır. Çünkü amaç sadece haberleşebildiğini göstermektir. Aşağıda paylaşacağım ApiService kodunda **token** kullanımını da yorum satırında görebilirsiniz. Gerekli durumlarda yetki kontrolü çerçevesinde de iletişim sağlanabilir. Login olabilmek için API’den bir Customer eklemeniz gerekmektedir. Bu customer mail şifresi ile login olabilir ve bahsi geçen işlevleri deneyimleyebilirsiniz. **Uygulamayı ayağa kaldırmak için Web ve API projeleri birlikte çalışıyor olmalıdırlar.**



ApiBaseService sınıfında generic olarak API ile haberleşen metotlar bulunmaktadır.

**Uygulamanın Çalışabilmesi İçin Gerekli Yapılandırmalar**

* RabbitMQ ve Redis bir **Docker Container** ile çalışabilmektedir. Gerekli host, port, şifre yapılandırmalarını appsettings.json dosyasından local docker uygulamanıza göre düzenlemeniz gerekebilir.
* Web uygulamasına girebilmek için öncelikle API’den bir Customer oluşturmanız ve bu bilgilere göre login olmanız gerekmektedir.
* Web uygulamasının çalışabilmesi için API uygulamasının çalışır halde olması gerekmektedir.
* Veritabanı .bak dosyası Database katmanında projeye eklenmiştir.
* Veritabanını kendiniz oluşturmak için migration kodlarını kullanabilirsiniz. SqlConnectionString bilgisi dbsettings.json dosyasında mevcuttur. (Migration işlemi için Initilazier.cs içindeki “builderMigration” değişkenini kullanın.)
* Tabloları oluşturduktan sonra test datası(Customer ve Product) oluşturmak için Web uygulamasında Home/ InitialData url’sini kullanabilirsiniz.
* API Authentication için kullanıcı adı = “admin”, şifre = “admin” kullanılmalıdır.
* Bu dosya API katmanında Documentation klasörünün içine eklenmiştir.