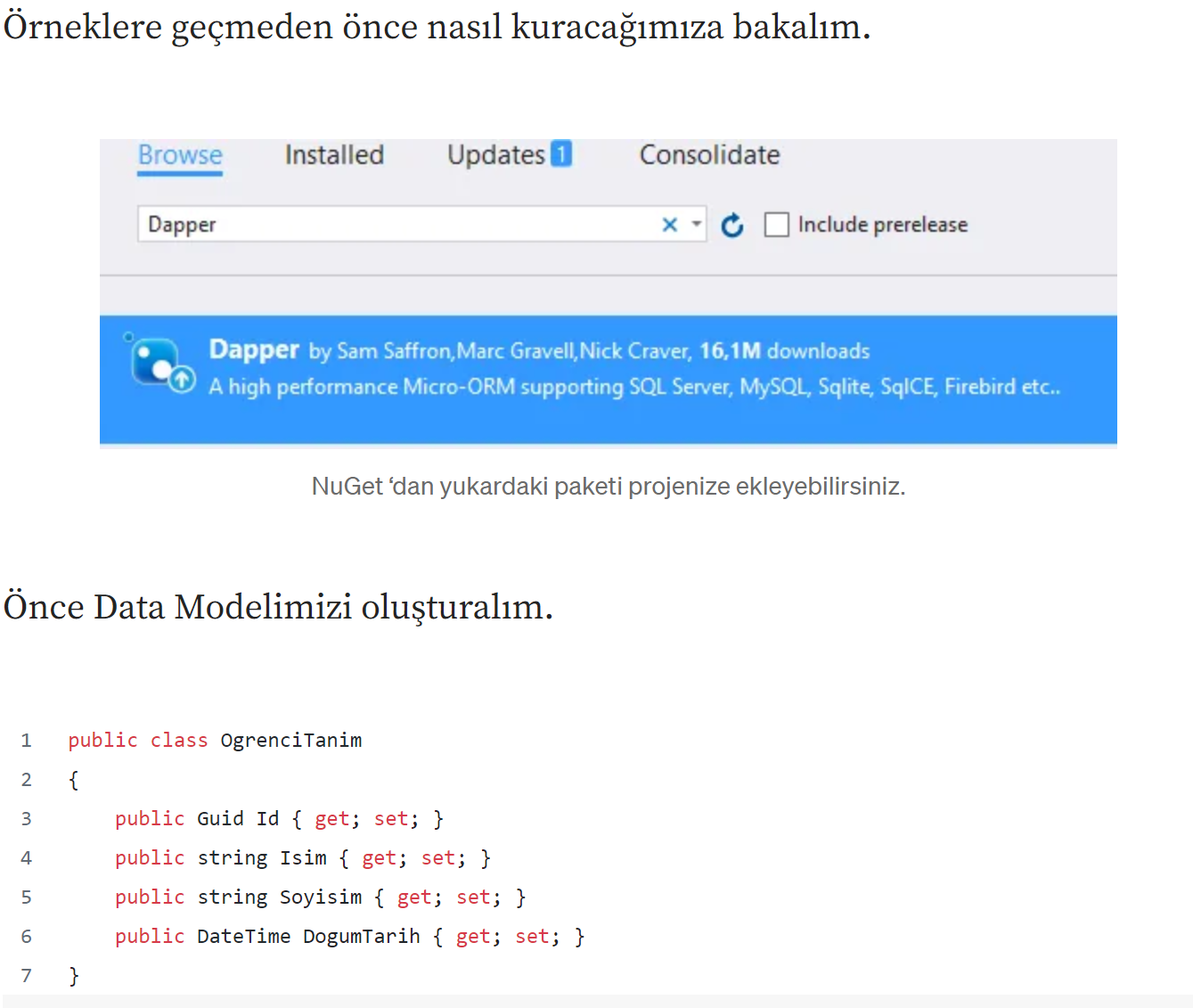
**DAPPER NEDİR ? NE İÇİN KULLANILIR ?**

Bir Micro ORM Olan **Dapper’ı**diğer ORM’lerden ayıran en büyük özelliği **Ado.Net** hızına yakın bir hızda çalışmasıdır.

*Dapper***, [Stack overflow](https://stackoverflow.com/" \t "_blank)**ekibi tarafından **LightWeight(Arka tarafta herhangi bir mapping işlemi veya konfigürasyonu yapmaz)**olarak geliştirilmiş [Github ta](https://github.com/StackExchange/Dapper" \t "_blank) open source yayınlanan bir **ORM( Object Relationship Mapper)** aracıdır. Çoğu veritabanına desteği vardır(SQL Server,MySQL..v.b).

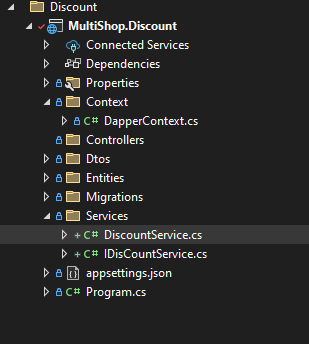
**Ado.Net**’te sorgularımızı veya prosedürlerimizi SqlDataReader, SqlCommand v.b nesneleri kullanarak gerçekleştiririz. **Dapper**,bu nesneleri yazma yükünü bizden alıyor. Generic ve extension olan metotlarını kullanarak filtreleme işlemlerimizi yapabiliriz. Daha az kod yazarak sorgularımızı kısa sürede execute edip istediğimiz tipe çevirebiliriz.



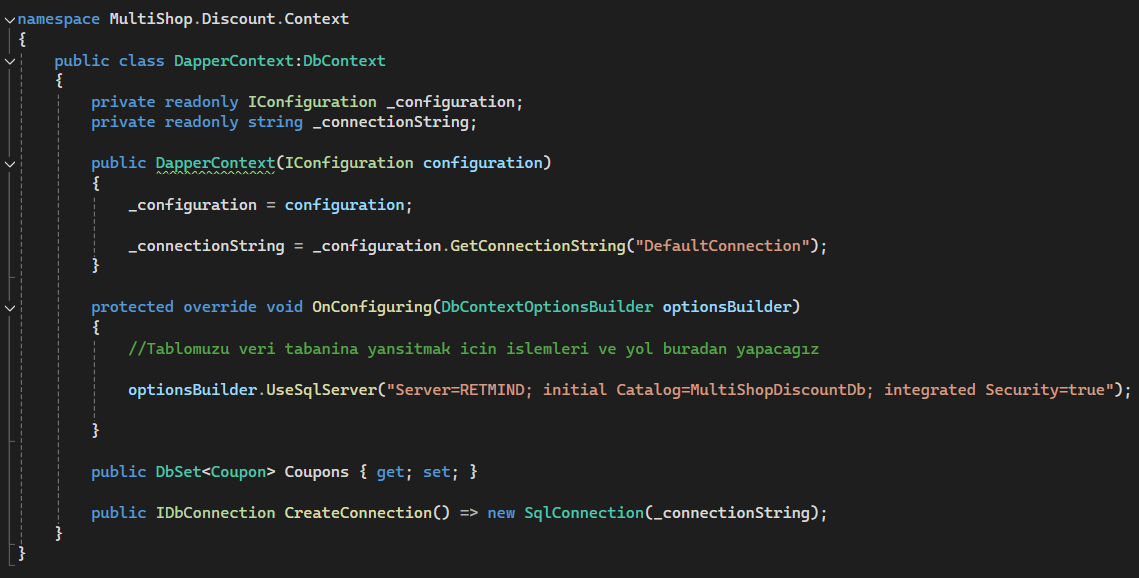
Burada dikkat etmemiz gereken bir husus var tablolarımızın kolonlarını veritabanında nasıl isimlendirdiysek Data Modellerimizde de o şekilde ayarlamalıyız aksi halde Data Modellerimiz, tablolarımız ile eşleşmeyecektir.

**Execute:**Dapper’ın bu fonksiyonu sayesinde sorgularımızı direkt gönderebildiğimiz gibi Ado.Net’teki gibi parametreleri tek tek eklememize de gerek kalmıyor. Fonksiyonumuzun ikinci parametresi, object bir veri istediği için her veriyi hatta birden fazla veriyi, ister isimsiz tipte isterseniz Data Modeliniz ile içeriye parametre geçebilirsiniz.

Veri büyüdükçe hız çok önemli bizim için bu yüzden dapper bizim kurtarıcımız oluyor .

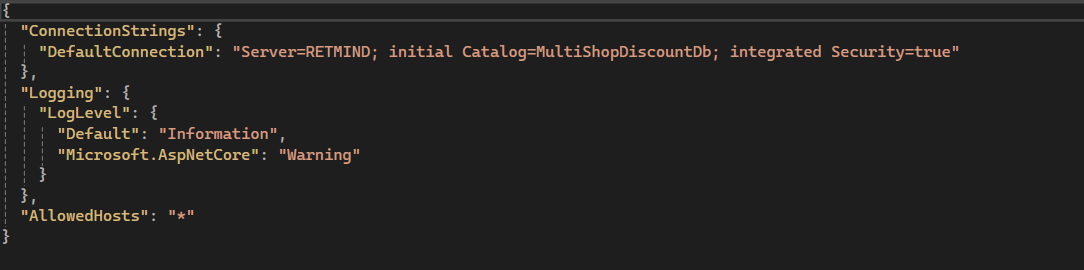
 DapperContext denilen sınıfı oluşturduk . ardından service kısmına gidip Discount ile ilgili Class ve Interfacelerimizi oluşturuyoruz.

DAPPER Context Sınıfımız

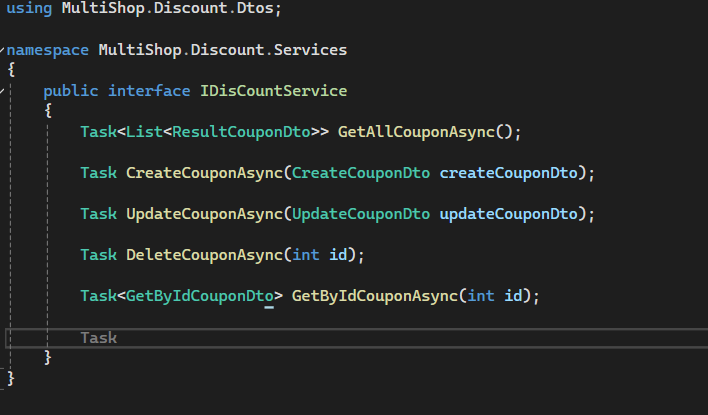


Burda Sql ile bağlantısını yaptığımız ve migration kısmını yaptığımız sınıf burası .

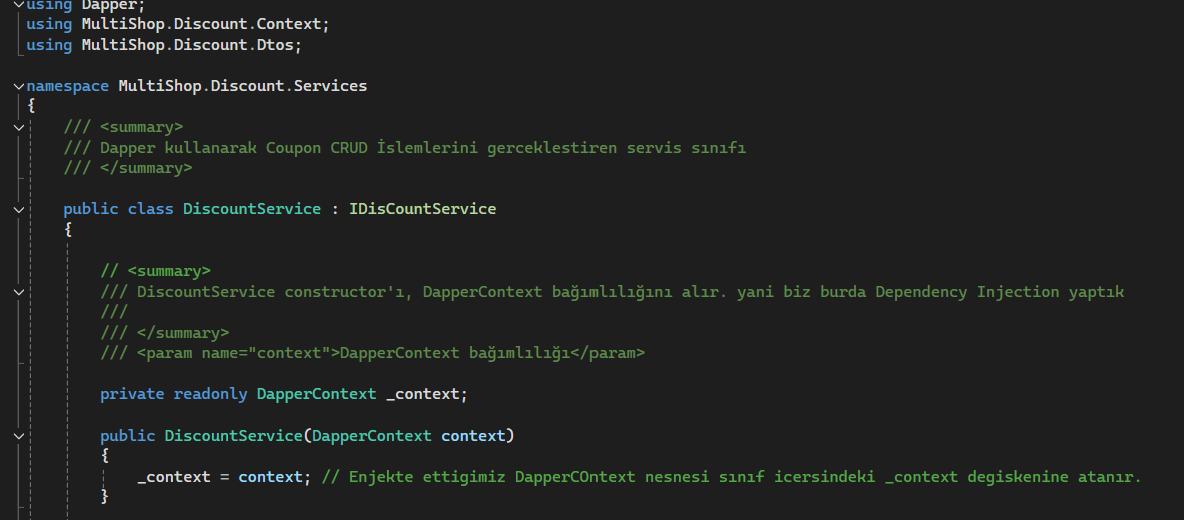
Database de tablomuzun isimlendirmesini de yapıyoruz .



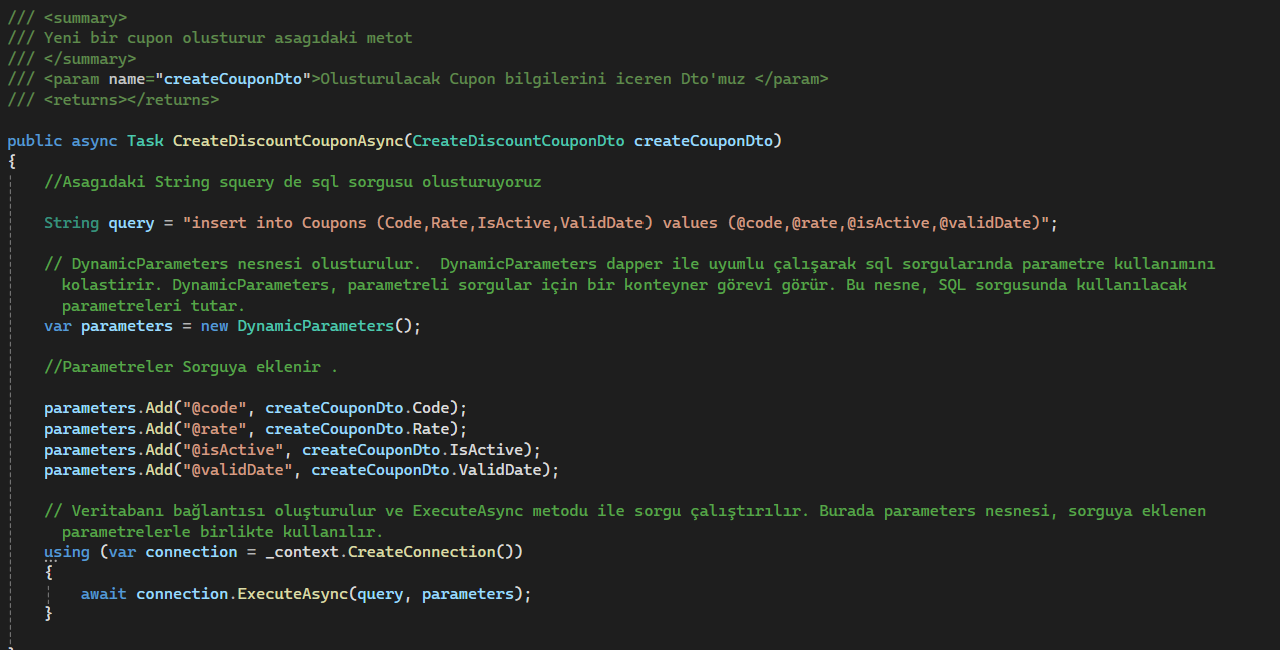
Bu işlemleri Context sınıfını oluşturduktan sonra appsettings.json dosyasının ayarlamasını bu şekilde yapıyoruz . Burada database yolunu belirtiğimiz ayarlarda mevcut .

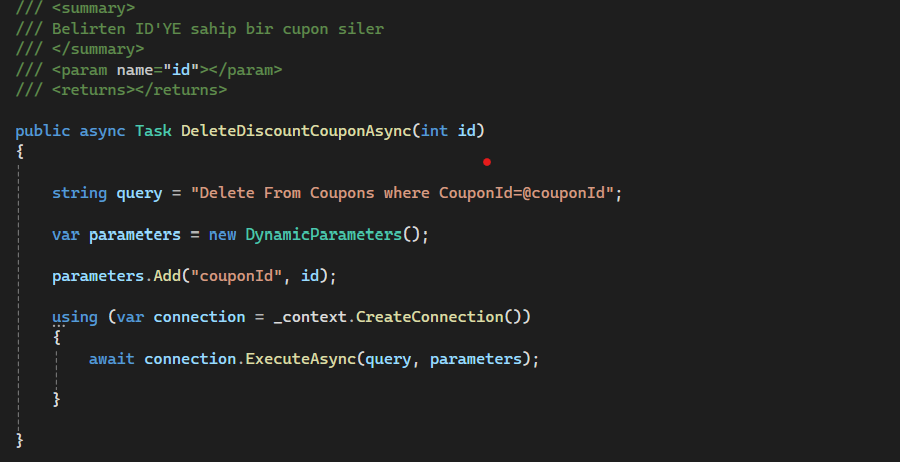


Yukarıda ki sınıf ise IDiscount ismindeki Interface sınıfıdır bunu DiscountService kısmına Implement yapacağız.



Discount service kısmında IDisCount miras alıyoruz . ve Injeksiyon yapılıyor .





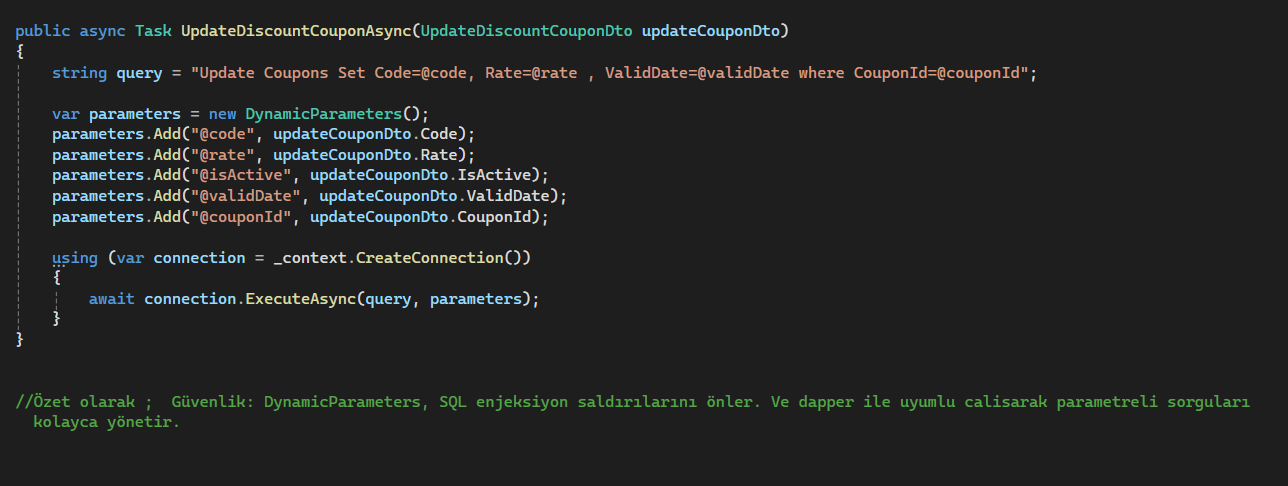
Burada delete metodunu kullanıyoruz önceki yaptığımız metoda çok benzer.



### Tüm Kuponları getirdiğimiz metod ve Id’e göre getireceğimiz metodlar burada yer alıyor. ueryFirstOrDefaultAsync Metodunun İşlevi ve Kullanımı:

**Temel İşlev:**

* **SQL Sorgularını Çalıştırma**: QueryFirstOrDefaultAsync, bir SQL sorgusunu veritabanında çalıştırır ve sorgu sonucundaki ilk satırı döner. Eğer sonuç yoksa, varsayılan değeri döner.
* **Asenkron İşlem**: QueryFirstOrDefaultAsync, işlemi asenkron olarak gerçekleştirir. Bu, işlemin uzun sürebileceği durumlarda ana iş parçacığının (UI veya diğer iş parçacıkları) bloke olmasını engeller ve uygulamanın performansını artırır.

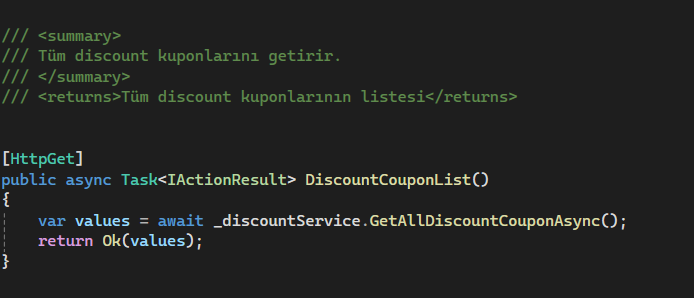


Database de olan kuponları değiştirilmesini yapıyoruz . ExecuteAsync metodu, Dapper kütüphanesinde kullanılan bir metottur ve SQL sorgularını asenkron olarak çalıştırmak için kullanılır. Genellikle veri ekleme, güncelleme veya silme gibi işlemler için kullanılır ve sonuç olarak etkilenen satır sayısını döner.

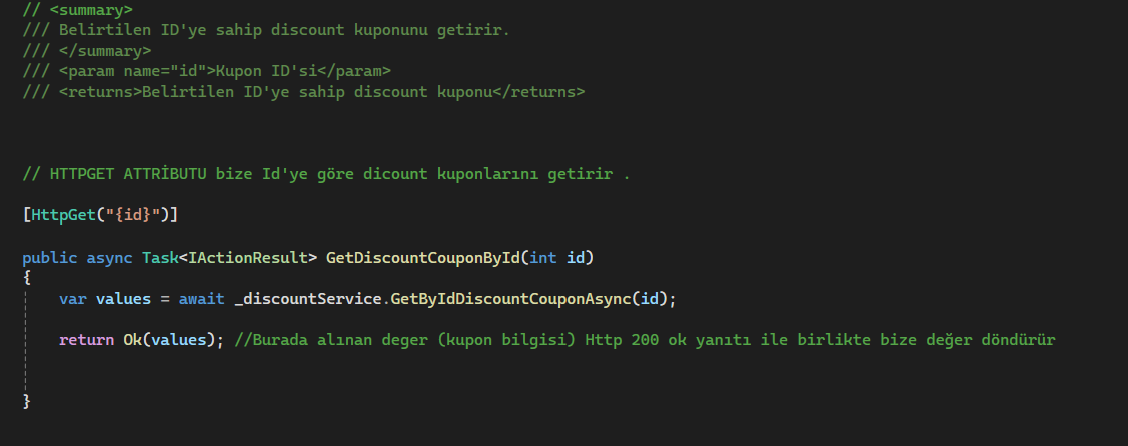
**APİ TARAFINDAKİ CONTROLLERDA YAPTIĞIM METOTLAR**

****

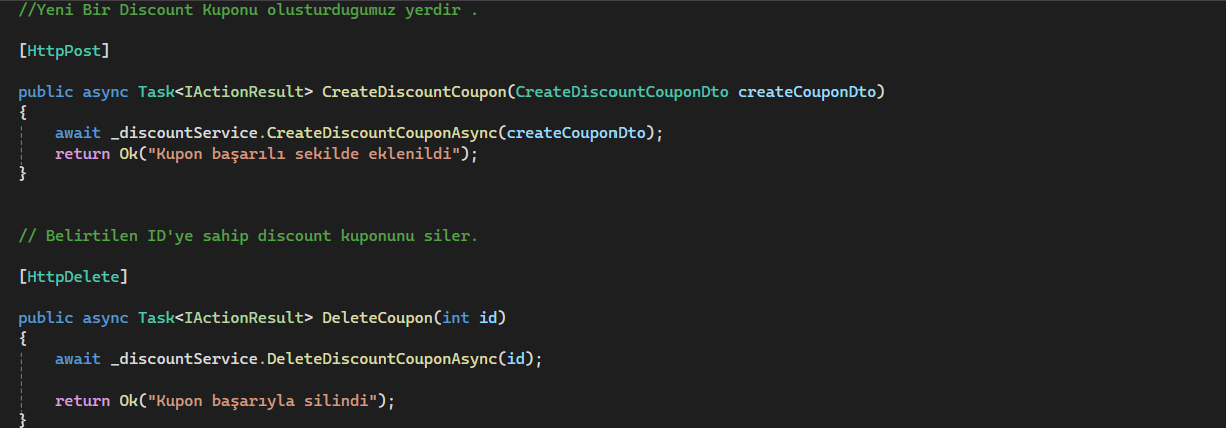
Constructor yapıp atamlarını gerçekleştirdim .



Var olan Kuponların hepsini getirme işlemini yaptığımız yer

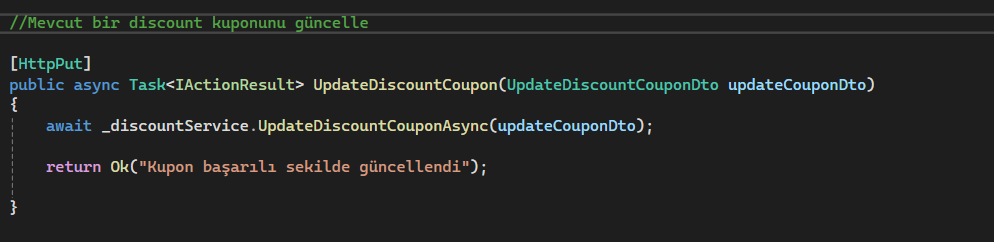


Burada Kupon Id’lerini database içeriisnde getirdiğimiz api dir.



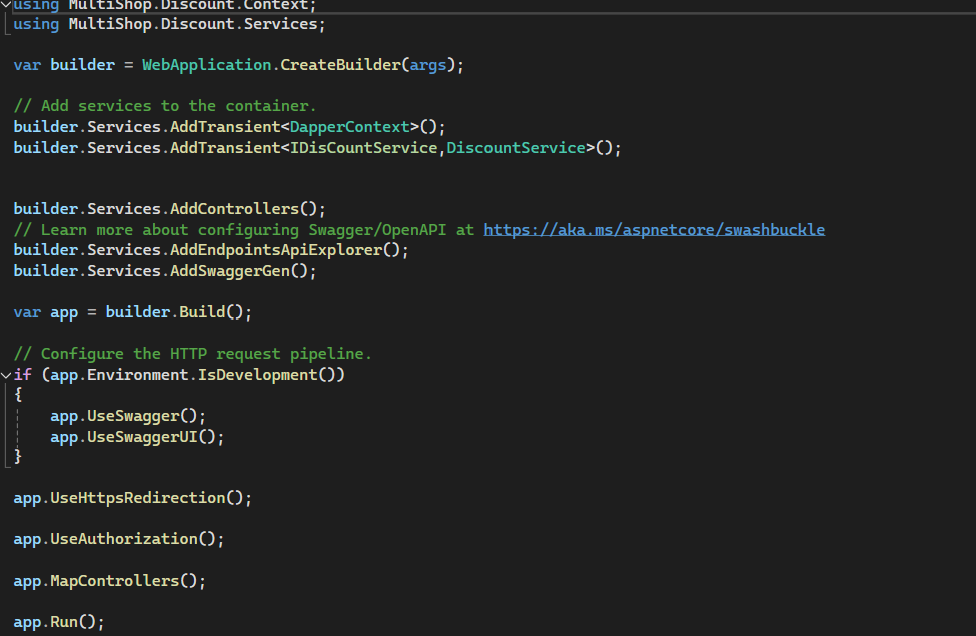
Kuponları olusturdugumuz ve ve sildiğimiz metotların yer aldı api controllerımız .

Birinde CreateCouponDto nesnesini alır diğerinde sadece id bulur ve siler .



Json ile Mevcut olan kuponlarımızda ki Verileri güncellediğimiz api ve icerinsinde UpdateDiscountCouponAsync metodu çalışır .

Not : Bu api içerisinde yazdığımız metotlar hepsi IDisCount service tarafında oluşturduğumuz interface aracılığı ile erişebiliyoruz .



En son olarak Program.cs içerisinde (Middalware) yukarıdaki serviceleri tanımlıyoruz .

Servislerin bağımlılık enjeksiyonu (Dependency Injection, DI) konteynerine eklenmesini sağlar.

* **AddTransient<TService>**: Her istek için yeni bir örnek oluşturur.
* **DapperContext**: DapperContext sınıfını DI konteynerine ekler.
* **IDisCountService, DiscountService**: IDisCountService arayüzünü ve bu arayüzü uygulayan DiscountService sınıfını DI konteynerine ekler. Bu, bir istek yapıldığında IDisCountService arayüzüne DiscountService örneğinin enjekte edilmesini sağlar.