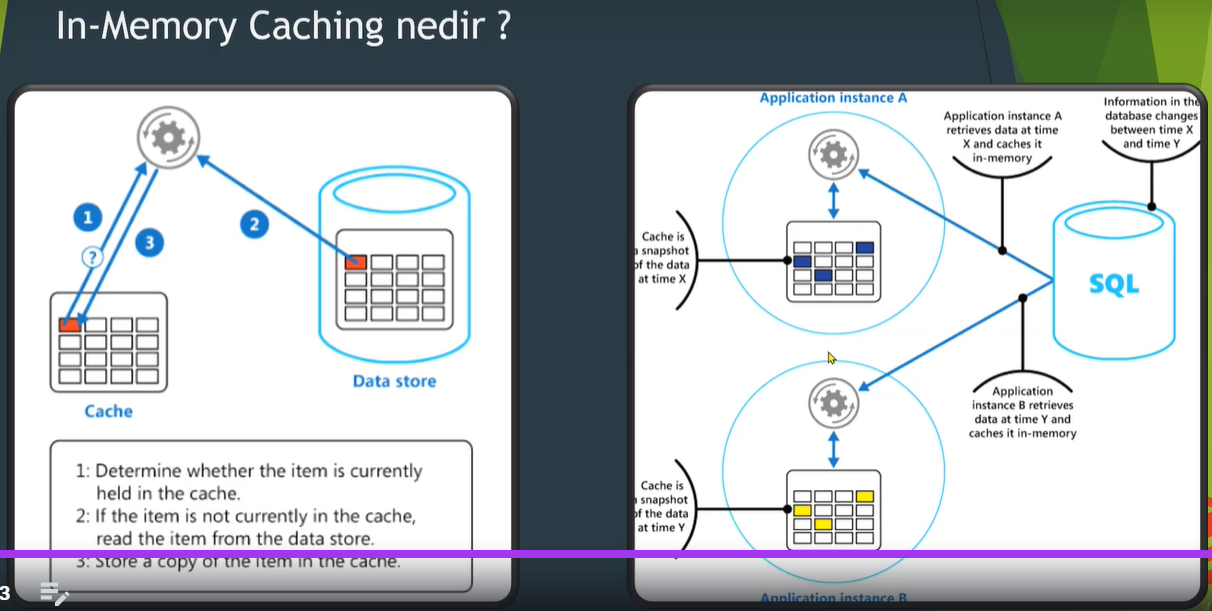
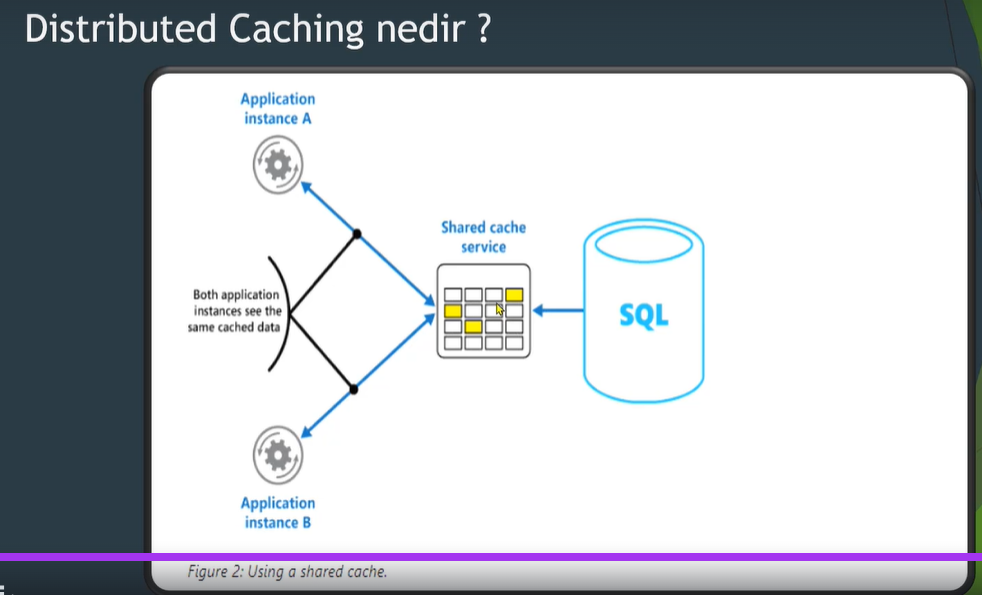
**BÖLÜM 1 GİRİŞ**

* **Caching Nedir Caching Çeşitleri Nedir?:** Cache Çok sık kullandığımız dataların kaydedilme tekniğine diyoruz. Bu çok sık kullandığımız dataları belli bir ortama kaydettikten sonra ihtiyaç halinde bu datanın birde okunma süreci olacaktır. İşte genel anlamda hem datanın kaydedilmesi hemde ilgili datanın ihtiyaç halinde okunma işlemi sürecine Caching diyoruz. Yapma sebebimiz mesela web sitemizin daha hızlı olmasını sağlamak daha iyi bir kullanıcı deneyimi sağlamak ve performansı arttırmak için Ölçeklenebilirliğini arttırmak için. Web sitemiz 10 kişi girdiğinde nasıl bir hızda çalışıyorsa 1 milyon kişi girdiğinde de aynı hızda çalışması gerekiyor performanstan bahsettiğimiz bu. Bu imkânı sağlamak için kullanacağımız tekniklerden bir tanesi cachingtir.
* **Caching 2 ye ayrılmaktadır**. Birsi In-Memory Caching diğeri ise Distributed Cachingtir. In-Memory Cache’e aynı zamanda Private Cache deniyor. Distributed’a ise aynı zamanda Shared Cache denebiliyor.
* **In-Memory Caching Nedir?:** Uygulamayla ilgili dataların uygulamayı barındıran web serverin raminde yani memorysinde kaydedilme işlemine denir. Ramdan data okumak ile bir sabit diskten data okumak arasında ciddi bir hız farkı vardır. Rame çok daha hızlı bir şeklde dosyalarımızı yazabilir veya okuyabiliriz. O yüzden cachelemede ilgili datalara hızlı erişebilmek için datalar genellikle memoryde tutulur. İlk görselde uygulamam kullanıcıya veri gözeteceği zaman direkt veri tabanına bağlanmak yerine ilk olarak ilgili memoryde datanın olup olmadığına bakıyor. Eğer ilgili data memoryde yoksa datayı almak için veritabanına bağlanıyor ve ilgili datayı alıyor sonrasında kullanıcıya göstermeden önce bu datayı cache’e yani rame datayı kaydediyor. Bu data artık ramde durmaktadır daha sonra başka bir kullanıcı aynı datayı görmek istediği zaman bu sefer veritabanına bağlanmak yerine datayı direkt olarak cacheden yani memroyden datayı alıp kullanıcıya gösteriyor. Bu sayede veritabanına bir sql connection açıp kapatmak ve ilgili datayı sqlin kendi içerisinden okumak yerine direkt olarak ramden okumak sitenin performansını ciddi anlamda arttıracaktır. In-Memory cache’de süreç bu şekilde ilerliyor. Tabi burada cache’de tutabileceğiniz data miktarı uygulamayı barındıran host sistemin ram miktarına bağlıdır.
* Burada bir handikap var yani In-Memory cachede bir sıkıntılı durumu var. İkinci görselde bunu inceleyelim. Buradaki şekilde aynı uygulamanın 2 tane farklı instancesi mevcut bu istanceler birbirinden farklı web serverlarda çalışıyor birbirinden bağımsızlar. Bu uygulamaların önünde load balancer olması gerekiyor gelen istekleri ilgili 2 sunucunun müsaitlik durumuna göre yönlendirdiğini düşünelim. In-Memory cache kullandığından dolayı eğer ben uygulamada bir datayı cachlemek istersem uygulamanın bulunduğu web serverin bana tanımlamış olduğu memoryi kullanıyorum. Aynı şekilde alttaki uygulamada farklı bir web serverda çalıştığından dolayı o da aynı işlemi bağımsız olarak kendi raminde yapacak. Diyelim ki load balancer’a bir istek geldi load balancer bu isteği yukarıdaki sunucuya yönlendirdi. Gelen istek sonucunda döneceğim data eğer çok sık kullanılan bir dataysa bunu cachledeğimizi düşünelim saat10 da sqlserverdan çekilmiş data cachede artık var diyelim. Sonra saat 11 de yeni bir istek geldi load blaancer bu sefer aşağıdaki sunucuya yönlendirdi uygulamam kendi memorysine baktı böyle bir data yok ama aynı istek bizden aynı datayı isteyecek bir istek geldi bu sefer gitti saat 11de sqlserverdan aldığı datayı kendi memorysine cachledi. İşin sonunda yukarıdaki sunucunun memorysinde saat10da tutulmuş achace varken aşağıdaki sunucunun memorysinde saat11de tutulmuş olan cache var. Eğer ki saat 10 ile 11 de sql serverden çekmiş olduğum data aynıysa problem yok. Sıkıntı şurada başlıyor. Saat 10 da çekmiş olduğum data ile saat 11 de çekmiş olduğum data farklı ise benim farklı memorylerde aynı datanın farklı versiyonları kaydedilmiş oluyor. Daha sonra load balaner’a bir istek geldiği zaman eğer yukarıdaki sunucuya yönlendirirse buradan farklı bir data aşağıdaki sunucuya yönlendirirse buradan da farklı bir data dönüyor olacak burada bir veri tutarsızlığı olacak. Ben illaki In-Memory Cache kullanacaksam bu kötü durumu kısmi olarak engellemek mümkün. Session Sticky denilen bir özellik var. Yani yapışkan session deniyor buna. Biz önde çalışan load balancer’a şunu diyebiliriz. Sana bir istek geldiği zaman bu hangi kullanıcıdan geliyorsa bunu bil ve arkasından ilk isteği load balancer hangi sunuya yönlendirmişse bundan sonra a kullanıcısından gelen bütün istekleri yukarıdaki sunucuya yönlendir diyebiliriz. Yine bir veri tutarsızlığı var ama en azından kullanıcıya farklı datalar göstermemiş oluruz kısmi olarak geçici olarak bu durumu engellemiş oluruz. Siz burada a kullanıcısına da b kullanıcısına da aynı datayı göstermek istiyorsanız ve farklı instancelerda çalışan uygulamalarınız varsa o zaman işte Distribute Cache yani cachelerini kaydedebileceğiniz merkezi bir yere ihtiyacınız var. Yani uygulamayı bir serverda tek bir instance ile ayağa kaldırıyorsak In-Memory cache sorun değil ama birden fazla istancesini farklı sunucularda ayağa kaldırıyorsak o zaman problem başlıyor.



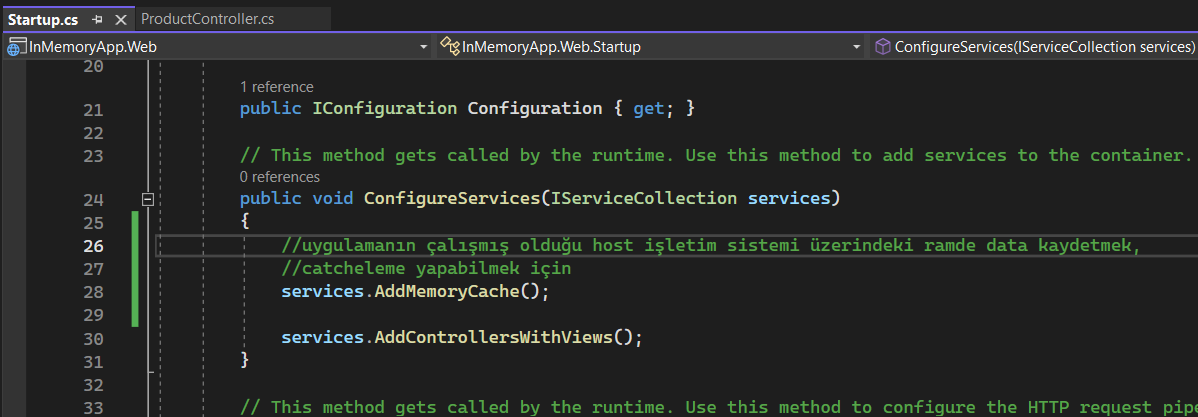
* Distributed Cache Nedir?: Cachelenecek olan dataların uygulamanın ayağa kalkmış olduğu host işletim sisteminin memorysinde değil tamamen ayrı bir yerde cache servisinde redis gibi, dataların tutulma işlemine deniyor. Cachlenecek olan datalar tamamen uzakta çalışan ayrı bir servisin memorysinde tutulduğundan dolayı burada bize bazı avantajlar sağlar. Bunlardan ilki az önce bahsettiğimiz veri tutarsızlığının önüne geçmiş oluyoruz. A instnaceside B instanceside cachedeki dataya ulaşmak istediği zaman merkezi bir yerden erişeceğinden dolayı her zaman aynı datayı almasını garanti etmiş oluruz. İkinci avantaj eğer sizi In-Memory cachede uygulmanızı ayağa kaldıran host işletim sistemi veya bir web server bir restart gördüğü zaman memoryde tutulmuş olunan tüm cacheler silinecektir. Ama eğer siz bir distrivuted cache servisi kullanırsanız artık uygulamalarınız restart olsa daha cachelnmiş datalar ayrı bir serviste tutulacağından dolayı cachedeki dataları kaybetmemiş oluruz. Inmemorye göre 2 dezavantajı var bunlardan birisi hız konusu inmemorycache verilere çok daha hızlı bir erişim sağlar ama gözle görülecek kadar büyük bir fark yoktur. Bizim için hızdan daha önemli olan konu verilerin tutarlı olmasıdır takılı olunacak bir konu değil hız. İkincisi ise inmemorycache’i kullanmak kolaydır. Ama distributecache ‘in implementasyonu biraz daha zordur. Senaryosu ise şöyle çalışıyor load balancera a kullancısı istek yaptığı zaman load balancer isteği yukarıdaki sunucuya yönlendirdi uygulamam önce gitti cache serviceye baktı eğer burada data yoksa sqlden okuyor daha sonra cache servicenin içerisindei memorye bu datayı kaydettikten sonra kullanıcıya gösteriyor. Daha sonra yine aynı datayı görmek isteyen bir istek geldiğinde load balancer onu alttaki sunucya yönlendirse bile cache servicede bu data olacağından dolayı datayı hızlı bir şekilde cache serviceden çekip kullanıcıya gösteriyor. Yani uygulamaların cache servis olarak bakacakları yer ortak oluyor burada.

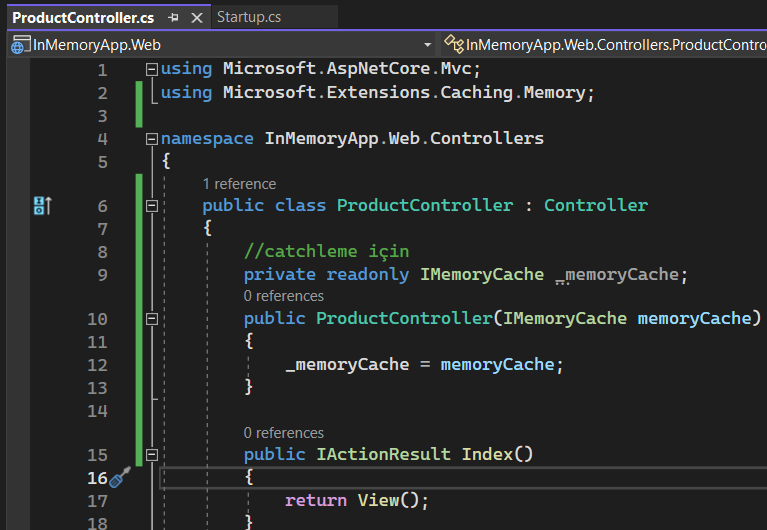


* **On-Demand ve PrePopulation Caching Nedir?:** Bu konulara geçmeden önce şu soruyu sormak lazım. Biz hangi datayı cachleyeceğiz. Bizim cacheleceğimiz datada şu özellikler olmalı. Çok güncellenmemeli çok değişmeyen bir data olmalı iki çok sık erişeceğimiz bir data olmalı. Yani uygulamamda çok update almayan çok sık kullandığım bir data varsa benim için cachleme açısından uygun bir adaydır. Peki ben datayı ne zaman cacheleyeceğim bu durumda On-Demand ve PrePopulation Caching kavramları ortaya çıkıyor. On-Demand datayı sadece talep olduğunda catchleyebilirsiniz. Yani datayı almak için bir istek geldiğinde gidersin bu datayı önce sql serverdan çekersin daha sonra catch’e kaydedersin ve ilgili requeste karşılık response dönersiniz yani ilgili datayı sadece talep edildiği anda catchlersiniz. Prepopulation ise uygulamanızı ayağa kaldırdığınız zaman eğer bir datayı catchlemek istiyorsanız uygulamanızı ayağa kaldırmakla beraber dataları sqlserverdan çekip daha talep gelmeden önce catchleyebilirsiniz. Bunların birbirlerine bir üstünlükleri yok. Uygulamadan uygulamaya değişebilecek bir yapıdır. İllaki bunu kullanın diye bir yöntem yok. Siz bu 2 stratejiyi uygulamanızda deneyerek hangisinin daha performanslı olduğuna karar vereceksiniz.
* **Cache Ömrü (Absolute Time ve Silding Tİme) Nedir?:** Cachlenecek olan datalarımıza bir ömür biçerken kullandığımız Absolute Time ve Sliding Time. Biz bir datayı cachlerken aynı zamanda catchlenecek dataya bir ömür verebiliyoruz yani memoryde ne kadar süre kalabileceğini belirleyebiliyoruz ve bunu her bir catchlenecek datayı memoryde muhafaza ederken belirleyebiliyoruz. Yani bu data 5 gün kalsın bu data 10 gün kalsın şeklinde belirleyebiliyoruz. Burada 2 kavram devreye giriyor. Absolute Time bir catche’e kesin bir ömür tarihi veriyoruz. Bir catche oluştururken o catchein ömrünü absulute time olarak belirlersek 5 dk verirsek catche oluştrulduktan 5 dk sonra memoryden silinir ne olursa olsun. Sliding Time ise bir catchein memoryde ne kadar inaktif kalacağıdır. Siz slidng time olarak 5 dakika verirseniz 5 dakika içerisinde bu catche’e erişilirse siz bu catchin ömrünü 5 dk daha arttırmış olursunuz. Yani vermiş olduğunuz süre içinde catche’e erişilirse bu catchin süresi o kadar artar. Eğer 5 dk içerisinde bu catche’e erişilmezse bu catche silinir. Sliding time’ın şöyle bir dezavantajı var her zaman bayat dataya ulaşabilirsiniz. Yani siz bir datayı catcheledikten sonra sadece slinding time verirseniz ve bunu da 5 dk olarak ayarlarsanız uygulamanızda 5dk içerisinde bu catche erişmeye çalıştığınız zaman sürekli olarak ilgili dataya yani cathlemiş olduğunuz dataya erişirsiniz ve bu catchin ömrü bu şekilde günlerce sürerse hep eski bayat dataya erişiyor olabiliriz. Hiç veritabanına gidip acaba yeni data var mı diye kontrol yapmamış oluruz. Bunun önüne geçmek için sliding time belirlerken aynı zamanda absolute time belirleyebiliyoruz. Diyelim ki siz sliding time olarak 2dk belirlediniz ve asbsolute time olarak 10dk belirlediniz. Şimdi 2 dk içinde catche’E erişim sağlanırsa catche ömrü 2 dk daha uzayacak böyle böyle sliding time çalışacak ama nereye kadar çalışacak işte absolute timeda belirlediğimiz en fazla 10dk ya kadar bu işlemi yapabilirz 10dk sonra kesin olarak catche memoryden silinecektir. Bir sonraki istek için veri tabanına gidilecek ve güncel data cachlenecektir. Bu sayede bayat data olayından kurtulmuş oluruz bayat datadan kastımız hep catcheden data okuamktır hiç veri tabanına gidipte acaba güncel data var mı yok mu kontrol edilmesidir işte uygulama sürekli olarak catcheden data okuyorsa bayat data okuyabiliyor olma ihtimali artar. O yüzden tavsiye edilen tek başına Sliding Time kullanmayalım mutlaka Absolute time ile beraber kullanın. Eğer datamız sabitse hiç değişmeyecek bir dataysa o zaman sadece Sliding Time kullanabilirsiniz hiç Absolute time kullanmaya gerek kalmaz veya Absolute Time kullanıp süresini çok uzun bir süre verebiliriz bu şekilde catchi sürekli olarak memoryden okumayı sağlayabilirsiniz. Hem In-Memory Catchede hemde Distirubuted Cache’de bu olaylar vardır.

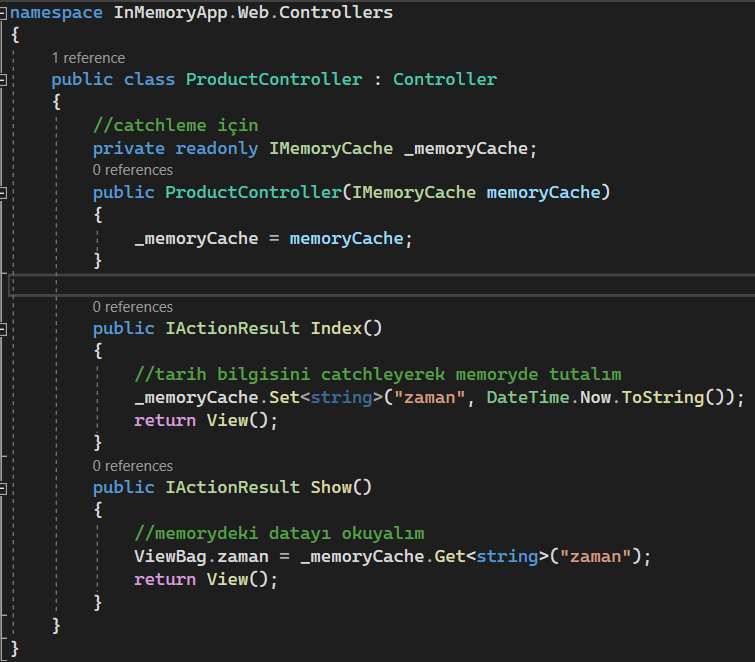
**BÖLÜM 2 ASP.NET CORE + IN-MEMORY**

* Asp.Net Core uygulamalarımızda memorye bir datayı cachleyebilmemiz için öncelikli olarak bir servise ihtiyacımız var yani uygulamamıza bir servis eklememiz gerekiyor. AddMemoryCache() bunu startup tarafında ConfigureServicez metoduna eklediğimiz zaman artık uygulamamızın herhangi bir yerinde data cachleme işlemi gerçekleştirebiliriz. Bu data cachlemeyi nasıl gerçekleştiricez onuda herhangi bir sınfıın konstraktırında depeendency injection olarak IMemoryCache interfacesini geçerek bu interface üzerinden gelen metotlar yardımıyla biz kolayca datalarımızı memoryde catchleyebiliyor olacağız.
* Bir ASP.Net Core web application mvc uygulaması oluşturalım. Dediğimiz ayarlamaları yapalım.

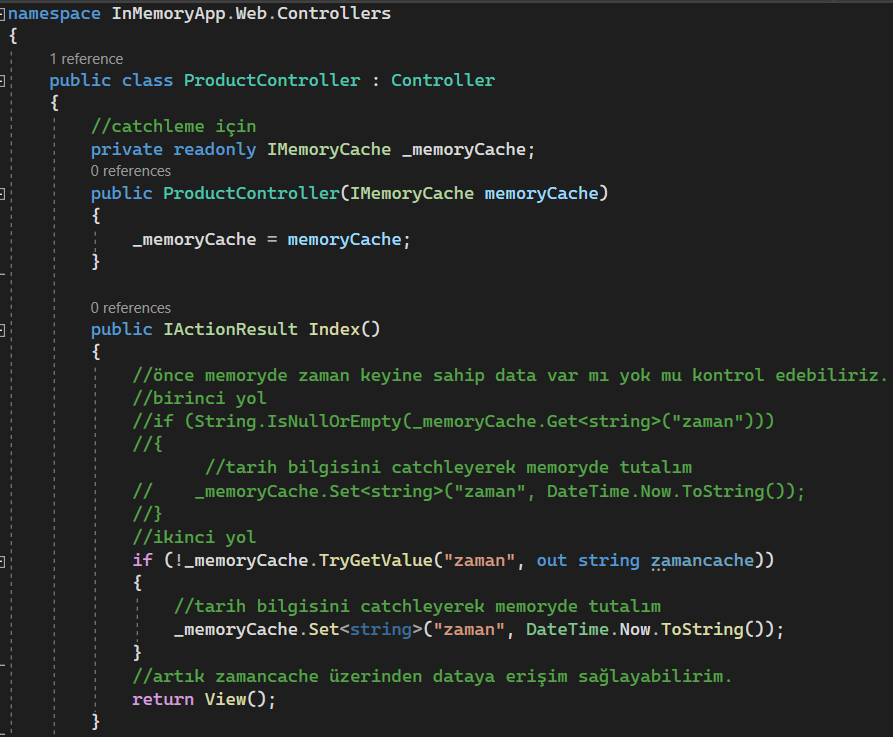


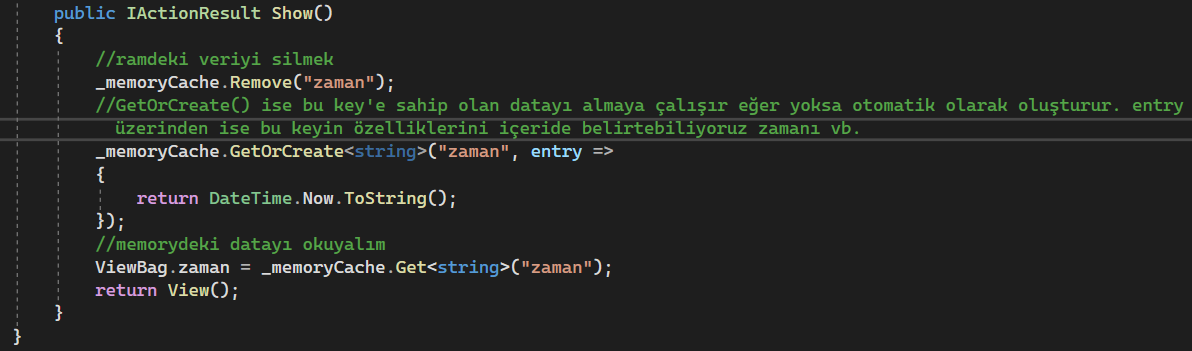


* Şimdi IMemoryCache interface’i üzerinden gelen Get() ve Set() metotlarıyla beraber memory’e bir data kaydedeceğiz ve daha sonra bu datayı memoryden okuyacağız. Get() memoryden bir datayı alabilmek için Set() ise memorye data kaydetmek için kullanılır. Bunlar aynı zamanda generic metotlardır ve memoryde her şeyi tutabiliriz, bir resim, bir pdf, bir sınıfı seriliaze edip tutabilirsin. İstediğin her şeyi seriliaze ettikten sonra tutabilirsiniz tek sınır memoryin kapasitesi olacaktır. Biz cachede değerlerimizi her zaman key value şeklinde tutarız. Set metotunun parametresinde key olarak object türünde bir değer bekliyor yani ben demek ki bunun içerisine her şeyi verebilirim .net tarafında object en üst sınıftır int,string,short hepsi object sınfıından türemiştir. Metot generic olduğundan dolayı tip güvenli hale getirebiliriz mesela <string> dersek metot bizden parametre olarak object türünde key beklerken artık string türünde value bekleyecektir. Daha sonra Get metotu ile beraber bu key ‘e verdiğimiz değer ile datayı memoryden çekeceğiz. <string> diyerek ise string türünde bir data alacağım diyebiliyoruz get metotunda. Artık buradaki veri sayfama bilgisayarımın raminden gelecek ram dataları geçici olarak tutar eğer bilgisayarı kapatırsam bu veri ramden silinecektir. Cache süresi belirtmediğimden dolayıda pcye restart atmadığım sürece bu data ramimde sınırsız süreyle kalıyor olacaktır.

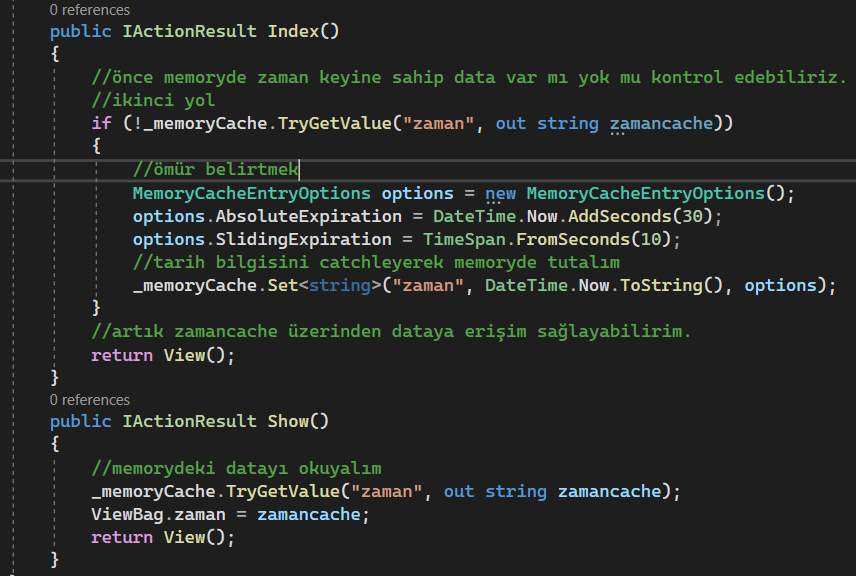


* Şimdi IMemoryCache interfacesi üzerinden bir kaç tane daha metodu tanıyacağız. TryGetValue() ve Remove() metodu. TryGetValue()’nu değeri almak için kullanırız geriye bool döner eğer içine verilen key ramde varsa true yoksa false döndürür bu datanın bu keyin ramde var olup olmadığını kontrol etmek için kullanılabilir bir metot. İkinci parametresinde out keywordu var c#da out keywordunu biz bir metotta birden fazla değer dönebilmek için kullanıyoruz. Bu metotun kendisi bool değer dönüyor aynı zamanda eğer ki tryget ile beraber bu keye sahip olan datayı alabiliyorsa buna value üzerinden erişebiliyor olruuz. Yani metot hem geriye bool döencek hemde eğer bu keye sahip value’yi alabilirse onu zamanchache değişkenin içine atmış olacak. Remove() ise memorydeki herhangi bir keyi silmek için kullanılır.

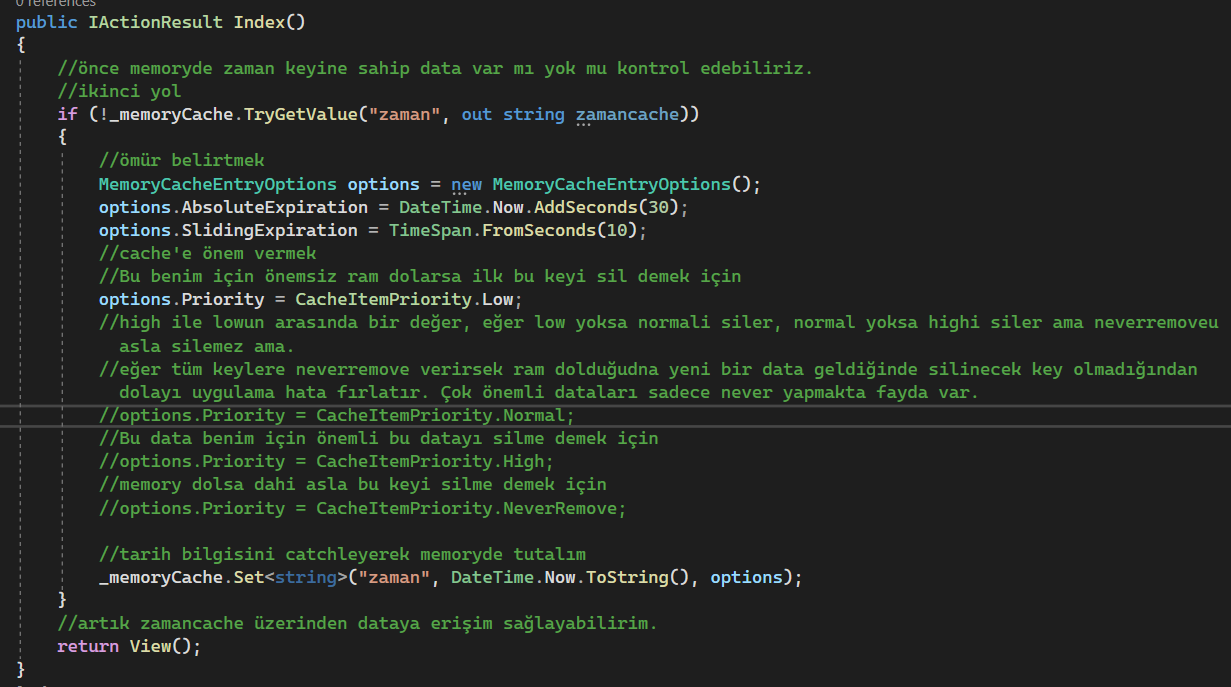




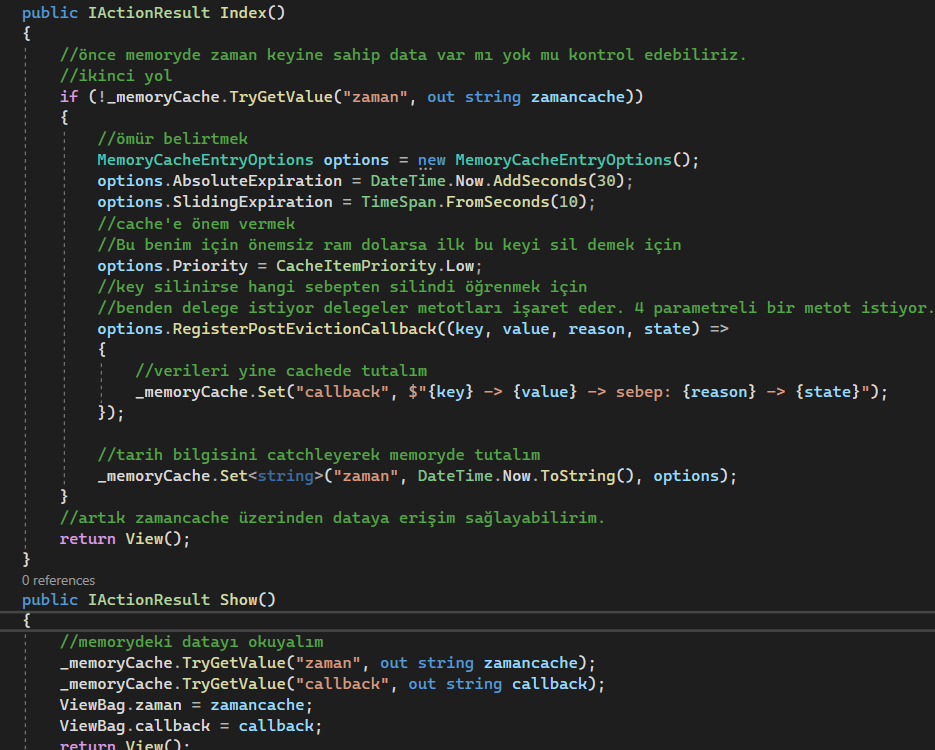
* Bir cache oluştururken ömrünü belirten propertyleri inceleyelim AbsoluteExpiration ve SlidingExpration.

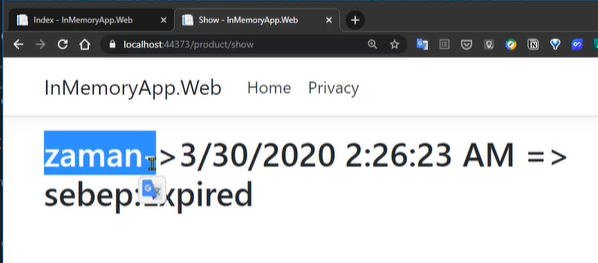


* Cache Options sınıfı üzerinden set edebiliyeceğimiz Priority propertysini inceleyelim. Siz mermoyinizde datalarınızı kaydetmeye başladınız ve memoryniz dolduğu zaman ve siz yeni bir data kaydedttiğiniz zaman memorydeki keylerden hangisi silinecek sizin için önemli olan keyler hangileri .net core bunu bilmiyor. İşte bunu .net core ‘a hangi keyi sileceğini haber etmek için cache oluştururken bu cache’e öncelik verebiliyoruz bu cache benim için önemli diyebiliyoruz bunu hiç silme. Veya cache oluşurken bu key benim için önemsiz eğer ram dolarsa ilk bu keyi sil diyebiliyoruz. Bunu Priority propertysi üzerinden yapıyoruz.

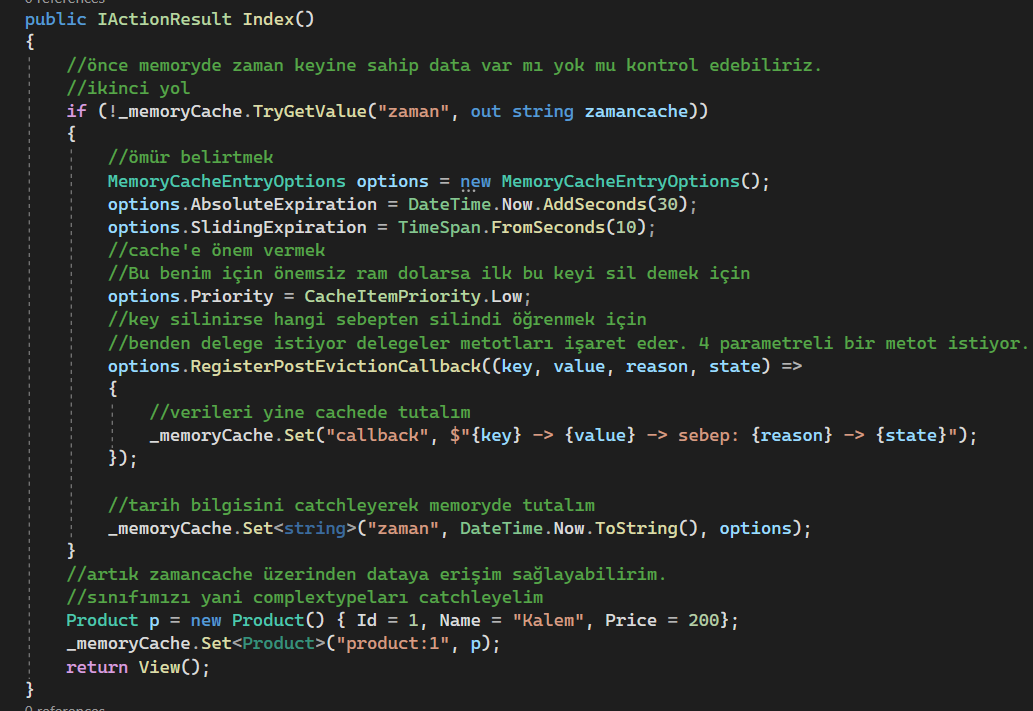


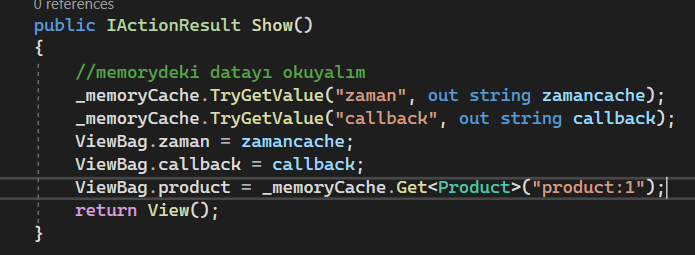
* Şimdi yine Options sınıfı üzerinden RegisterPostEvictionCallBack metodunu inceleyeceğiz. Şu işe yarıyor memoryden datalar siliniyor dolduğu zaman bu silinme önceliğini ise Priority de belirtmiş olduğumuz high low gibi levellera göre öncelik vererek datayı siliyor. Biz şunu tespit edebiliyoruz memoryden bir data silindiği zaman hangi sebeple silindi bu metot ile sebebini görebiliyoruz.





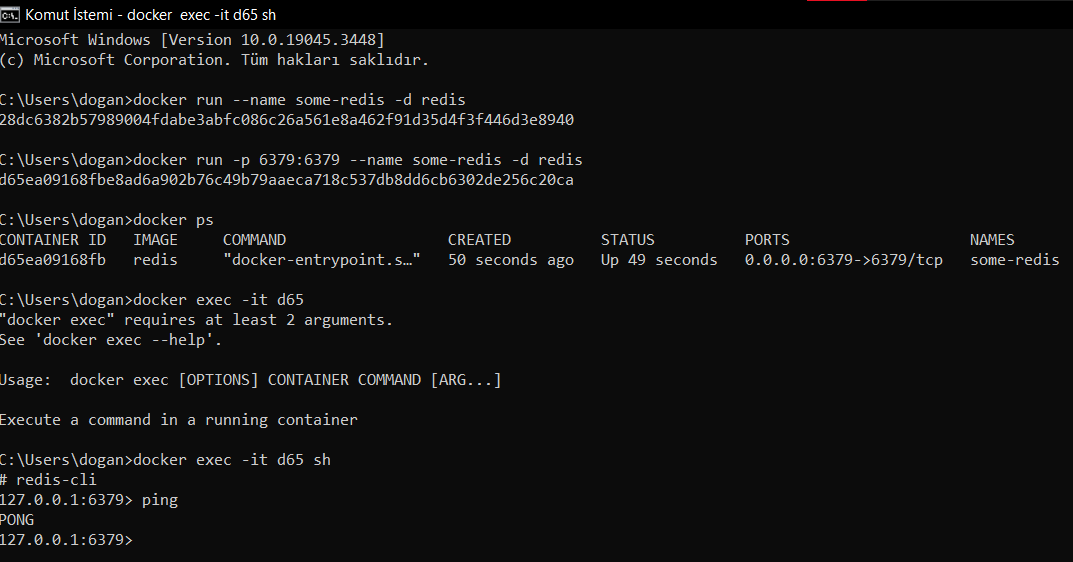
* Complex typeların catchlenmesini göreceğiz. Complex type dediğmiz aslında bizim uygulamamız içerisinde kullanmış olduğumuz classlar ve bu classlardan almış olduğumuz nesne örneklerinin catchlenmesi olayını göreceğiz. Bizim yerimize seriliaze işemini IMemoyCatche’im kendisi otomatik olarak gerçekleştirecek. Complex Typelerimizi ekstra seriliaze işlemine sokmamıza gerek yok bu sayede. Ama redis tarafında durum böyle olmayacak serliaze işlemini kendim yapmam gerekecek.





**BÖLÜM 3 REDİS**

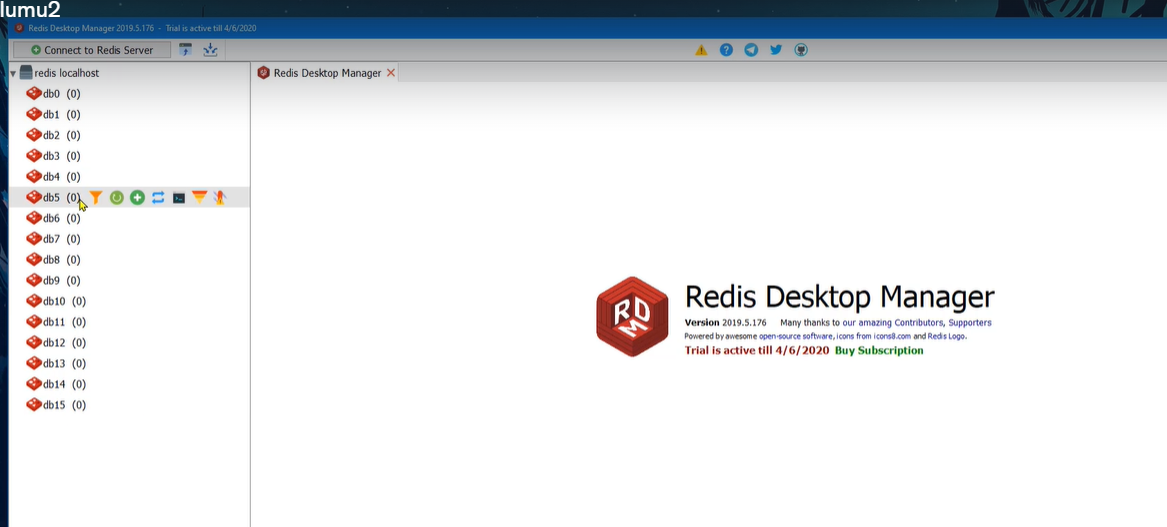
* **Redis Nedir?:** Çok kaba anlamla bir distribute catch olarak düşünebiliriz. Remote Dıctionary Server kelimelerini baş harflerinden oluşur. Open Source olarak geliştirilen ve dataları memoryde tutan bir no-sql veri tabanıdır. Redisin türevlerine göre yani rakiplerine göre en büyük avantajlarından bir tanesi veri tiplerine sahip olmasıdır. Yani redis çok basit bir şekilde key value şeklinde datalarınızı tutmaz sizin oop tarafında kullanmış olduğunuz veri tiplerine karşılık rediste de veri tipleri bulunur. Bu veri tipleri kolay bir şekilde implement edilebileceklerinden dolayı redisin güçlü olan özelliklerinden bir tanesidir. Yani siz c# tarafında List kullandığınız zaman redis tarafında buna karşılık gelen bir veri tipi var. O yüzden redis aslında çok basit bir şekilde key value nosql bir veritabanı değil aynı zamanda veri tipleride bulunmaktadır. Redis dataları memorde tuttuğundan dolayı okuma ve yazmada çok hızlıdır. Başka bir büyük avantajı ise veri tutarlığına sahip olmasıdır distribute catche olduğundan dolayı. Kullanım senaryosu şu şekilde siz uygulamanızda cacthlemek isteyeceğiniz datayı öncelikli olarak dbden çekiyorsunuz daha sonra redise kaydediyorsunuz daha sonra okuyorsunuz daha sonraki isteklerde veriyi veritabanından okumak yerine catchte var olup olmadığına bakıyorsunuz eğer catchte yani redisin raminde varsa datayı catcheden dönmüş oluyoruz. Distributed catchın ayrıca şu avantajı var biz kullanıcılarımızın session bilgilerini de redis serverda saklayabiliriz. Eğer In-Memory catche kullanırsanız session tarafında veri tutarsızlığı yaşıyorken redis tarafında kullanıcının oturum bilgilerini tutarken bu veri tutarsızlığından kurtulmuş olursunuz.
* Redisin Windows işletim sistemi için resmi bir sürümü yok çünkü redis server tarafında daha çok Linux kullanıldığından dolayı Linux için bir kurulumu var. Zaten server tarafında genelde Linux tercih edilir. Windows tarafında da kullanabiliyoruz. Windows tarafında open source olarak yani bağımsız geliştiriciler tarafından geliştirilmiş olan versiyonları var. Ama bu versiyonlar orijinal redisin birebir versiyonları ile eşzamanlı ilerlemez. Windwos tarafında redis kurmanın en basit yolu Chocolatey isminde Windows için bir paket yönetimiz var bu paket yöneticisini windowsa kurduktan sonra bunun üzerinden redis kurulumunu gerçekleştirebiliyoruz. Redisin bir server tarafı var birde bu server tarafını kullancabileceğimiz bir client tarafı var. Client tarafı dediğimiz server ile haberleşen taraf yani ben bir CLI üzerinden redis server ile haberleşmek istediğimde bir client’ı ayağa kaldırmam lazım. Ancak redisi kullanmanın en temiz yöntemi redisi docker container olarak ayağa kaldırıp çalıştırmaktır. Çünkü docker containerımız linux container olacağı için içerisinde en son güncel sürümlü orijinal redisi kullanıyor olabileceğiz.
* Şimdi docker container olarak redisi ayağa kaldıracağız. Docker hub üzerinden redis imagesini bulalım. Cmd ekranı açıp docker run --name some-redis -d redis komutunu yazarak önce bilgisayarda bu image var mı diye bakılacak yoksa docker hubtan bu image çekilecek ve bu image ile beraber bir container oluşturulup ayağa kaldırılacak. Tamam şimdi redis 6379 bu porttan ayağa kalktı ama ben bu porta localhosttan erişebilmem lazım o yüzden bu portu maplemem lazım. O yüzden bu containeri silip yeni bir kod yazalım. docker run -p 6379:6379 --name some-redis -d redis kodunu yazarak bu containeri 6379 portundan dışarıya açabiliriz. Docker içerisindeki container ise 6379 default redis portundan çalışacak. Şimdi containerimiz oluştu biz localhosttan bu container’a bağlanalım. docker exec -it d65 sh komutu ile d65 idsi ile başlayan container’a sh ile poweshell üstünden bağlan diyoruz. Sonra redis-cli diyerek 127.01.01:6379 adresinde çalışan redise bağlamış oluruz Redis CLI üzerinden. Sonra ping yazdığımda PONG geri bildirimini alırız.



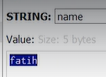
* Ayrıca sonradan direkt redis CLI üzerinden docker container içerisinde çalışan redise bağlanmak için containerin cmd ekranına redis-cli -h localhost -p 6379 komutunu kullanabiliriz.



* Şimdi redis desktop manager kurulumu gerçekleştirebiliriz. Bu aslında bir programdır. Bu programın amacı sizin memoryde tutmuş olduğunuz dataları görsel bir arayüz üzerinden size gösterir. Bu program ile en azından ben redise bir data kaydettiğim zaman bu data memoryde nasıl duruyor nasıl gözüküyor veya memoryde kaç tane datam var bunları görüntülemek için bu arayüzü kullanabiliriz. Bu işlermi Redis CLI üzerinden komut satırından da yapabiliyoruz ileriki aşamada genelde cli tercih edilir.
* Redis ayağa kalktığında default olarak 16 tane veritabanı ile beraber ayağa kalkar. Bunların hepsi ayrı birer veri tabanıdır. Bunun amacı datalarımızı gruplayabilmek için. Örneğin bu databasede sadece test sorguları yazıcam ben şu databasede sadece kullanıcı bilgilerini tutucam şu database’de uygulamamla ilgili genel cachler varsa onları tutucam şeklinde her bir database’de ayrı ayrı datalarımızı tutabiliyoruz.



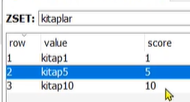
* **Redis Veri Tipleri Nelerdir?:** Günümüzde redisin bu kadar çok yaygın kullanılma nedenlerinden birisi redisin bize sunmuş olduğu zengin veri tipleri. Bu zengin veri tipleri sayesinde biz uygulamalarımızdaki senaryoya göre redis tarafında uygun veri tipini seçerek memoryde datalarımızı catcheleyebiliriz. Toplam 5 tane veri tipi var. Redis String, Redis List, Redis Set, Redis Sorted Set, Redis Hash. Redis dediğimiz zaman 2 tane yapı var bir tanesi redis server bu redis serverde işlemler yapabilmemiz için bir de bizim clienta ihtiyacımız var birde redis CLI isminde bir clientimızı ayağa kaldırıyoruz. Bu client üzerinden yazmış olduğumuz komutlarla redis server ile haberleşebiliyoruz. Bu komutlar redis server üzerinde configuration komutları olabileceği gibi memoryde hangi datalar tutulacaksa veya alınacaksa bunlarla ilgili komutlar bulunmaktadır. Bu komutları öğrenmemiz şu açıdan önemlidir redisi production ortamda kullandığınız zaman büyük ihtimal bir linux server üzerinde çalıştıracaksınız ve linux server default olarak kurulu olduğu zaman Windows server gibi size bir arayüzle gelmez o yüzden ssh üzerinden linux serverinıza bağlanıp burada kurulu olan redis üzerinden acaba şu data var mıydı yok muydu diye komutlar üzerinden redis server ile haberleşmemiz gerekiyor. Yani bu komutlar özellikle production ortamındaki redisimizde verilerimizi sorgularken veya kaydederken önemli. Bu komutları öğrendikten sonra c# tarafında güzel bir kütüphanemiz var bunun aracılığıyla biz direkt olarak redis ile haberleşiyor olacağız. C# tarafında bu öğreneceğimiz komutları yazmayacağız bunları bizim yerimize API yazmış oluyor. Biz sadece bir kütüphane indireceğiz bu kütüphane üzerinden gelen yardımcı metotlarla direkt redis server üzerinde işlemler gerçekleştirebileceğiz. Birde redis bir key-value depodur yani datalarımızı kaydederken mutlaka bir key olmalı ve birde valuesi olmalı bu key en fazla 512 mb boyutunda olabilir. Biz resim vs her şeyi key olarak verebiliriz demiştik ama önerilen çok uzun olmayan keyler vermemiz. Örneğin idsi 1 olan user kaydetmek istiyorsanız user:1 diyin böyle bir key belirtmek çok uygun olacaktır gidipte bir imageyi key vermek uygun değildir ama verebiliriz.
* **Redis CLI sadece ingilizce karakterleri destekler dikkat. Eğer redis-cli - -raw parametresi ile CLI’ı ayağa kaldırdığımız zaman encode edilmemiş olan ingilizce dışındaki karakterlerinizi okuyabileceğiniz yani encode edilmiş hale dönüştürüyor bu sefer data okurken Türkçe karakterli dataları bozuk şekilde görmemiş oluruz.**
* **Redis String Nedir?**: Kelimesi bizi aldatmasın sadece string veri tipini kaydedebileceği anlamına gelmiyor. Buradaki stringten kasıt tek bir değer olarak yani tek bir key valuesu olarak bir datayı kaydedebileceğiniz anlamına geliyor. User keyine karşılık gelen value illaki string olmak zorunda değil. Stringi key value şeklinde istediğimiz tek bir datayı tutabileceğimiz bir veri tipi olarak düşünebiliriz. Bu dataları redis kendi içerisinde bind binary olarak kaydedeceğinden dolayı istediğimiz her şeyi kaydedebiliyoruz. String veri tipi için CLI komutlarımız var. İlk inceleyeceğimiz CLI komutu eğer memoryde istediğiniz herhangi bir datayı kaydetmek istiyorsak setten sonra önce key sonra value yazıyoruz, SET name fatih şeklinde komutumuz var. Bu datayı okumak için ise get metotundan sonra key yazarak kullanabiliriz. GET name komutunu kullanabiliriz. Başka bir metot getrange metoduyla beraber okuyacağımız stringin hangi index aralığımda okumak istediğimi belirtebilirim. Getrange komutundan sonra key belirtiyorum sonra başlangıç indexini belirtiyorum sonra hangi indexe kadar okuyacağını belirtiyorum. GETRANGE name 0 2 bu kod sonunda “fat” sonucu gelecektir. Başka bir metot siz sitenizdeki ziyaretçi sayılarını tuttuğumuzu düşünelim eğer INCR komutundan sonra key verirsek ve bu keyin valuesi sayısal bir değerse ben bunun değerini bir arttırırım diyor. INCR ziyaretçi şeklinde kullanabiliriz. Peki ben bir bir arttırmak istemiyorum 10 10 arttırmak istiyorum bunu da şöyle yapabiliriz INCRBY ziyaretci 10. Peki bir azaltmak azaltmak istersem ise bu komudu kullanabiliriz DECR ziyaretci. Peki bir azaltmak değil 10 ar azaltmak istersem ise DECRBY ziyaretci 10 komutunu kullanabiliriz. Son komutumuz ise append komutu var olan bir dataya ek yapmak istediğimiz zaman kullanabiliriz append komutundan sonra bir key belirttikten sonra name içinde daha önce olan değere birde emre değerini eklemiş oluruz. APPEND name emre.



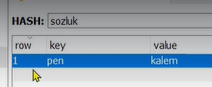
* İkinci veri tipi olarak Redis Listleri inceleyeceğiz. İsminden anlaşılabileceği gibi içerisinde dizi şeklinde data tutabileceğimiz bir veri tipidir. Diyelim ki kitaplar isminde bir keyiniz var ve bu key içerisinde kitap1 kitap2 kitap3 şeklinde veriler tutmak istiyorsunuz bir dizin tutmak istiyorsunuz işte burada Listi kullanabilirsiniz. Liste bir data eklediğiniz zaman her eklenen data listenin sonuna eklenir yani herhangi bir şey belirtmezseniz. Şöyle bir güzelliği var liste bir data eklerken şunu diyebiliyorsunuz işte listenin başına veya sonuna eklensin aynı şekilde silme yapıldığı zaman baştan silsin veya sondan silsin diyebiliyorsunuz. C# tarafındaki List’e benzetebilirsiniz. Listin kendine özgü CLI komutları var bunları inceleyelim. LPUSH komutuyla beraber bir listeye data ekleyebilirsiniz lpushtan sonra bir key veriyoruz sonrasında iste value veriyoruz. LPUSH kitaplar kitap1 Bu komutu her çalıştırdığımızda kitaplar listesine yeni belirtilen valueler eklenecek. Lpushun açılımı leftpushtur yani listenin soluna yani başına eleman ekler. Bunun RPUSH versiyonu var yani rightpush ise listenin sağına eleman ekleyecek sonuna yani. RPUSH kitaplar kitap2. Peki ben bunları nasıl sıralayacağım bunu inceleyelim LRANGE arkasından key değeri arkasından index olarak 0’dan itibaren -1 verirsek sonsuza kadar ne kadar data varsa laır 2 dersek index 2 ye kadar olanları alır buradaki l ifadesinin anlamı yok. LRANGE kitaplar 0 2. Peki ben indexi 2 olan elemanı almak istersem ise LINDEX kitaplar 1 komutunu yazarsak kitaplar key listesindeki 1. İndexteki datayı bize verir. Ben bu listeden data da silebiliyorum LPOP dersek gider soldan yani listenin en başındaki elemanı siler R ise tam tersi right yani listenin en sonundaki elamanı siler. LPOP kitaplar. RPOP kitaplar.



* Şimdi bir başka veri tipi olan Redis Set veri tipine bakalım. List veri tipi gibi içerisinde dizi tutabildiğimiz bir veri tipidir. Listten önemli 2 farkı vardır. Birisi Sette tutacağımız dizi içerisinde tutacağımız datalar benzersiz olmak zorundadır aynı datadan 2 tane bulunmaz. İkincisi Setlerde diziye eleman eklediğimiz zaman sağa veya sola ekle kararını veremiyoruz tamamen random bir şekilde data diziye eklenir. Bazen başa bazen sona bazen ortaya eklenir karar bizde değildir. Setlerde de datalarımız binary olarak kaydedileceğinden dolayı hem key hem value içerisine istediğimiz datayı kaydedebiliriz redis tarafında tüm veri tiplerinde bu olay böyledir. Eğer kaydedeceğimiz data image veya pdf dosyası veya class ise tek yapmamız gereken bunu ya byte’a ya da json’a seriliaze işlemi gerçekleştirmek ve arkasından kaydetmek. Şimdi gelelim Set veri tipine özgü olan CLI komutlarını görmeye. SADD komutu ile beraber sete bir data kaydedebiliyoruz sadd komutundan sonra key ve sonrasnda value belirtiyoruz. SADD color blue bu komutu birden fazla farklı valueler ile çalıştırırsak blue keyinin içine her eklediğimiz value dizi şeklinde tutulacaktır. Başka bir komut olan SMEMBERS ise değer okumak için kullanılır key bilgisi veriyoruz ve tüm seti okuyabiliyoruz. SMEMBERS color. Birde silme işlemini görelim SREM dedikten sonra key belirtip sonrasında value belirtiyoruz. SREM color green.
* Şimdi ise Redis Sorted Set veri tipini inceleyelim. Bir sıralama üzerine veri tipdiir. Set veri tipinde dizi içerisinde datalarımızı kaydederken herhangi bir şekilde sırasını belirleyemiyorduk. Ama Sorted Sette biz dizi içerisine bir data kaydederken sırasını belirleyebiliyoruz. 2. Sırada olsun 5. Sırada olsun gibi. Bunu da scor değişkeni üzerinden yapıyoruz. CLI komutlarını inceleyelim. ZADD komutundan sonra key belirliyoruz sonra scor belirliyoruz yani önceliğini veriyoruz 1 dersek 1. Sırada olsun demiş oluyoruz sonra value belirliyoruz. ZADD kitaplar 1 kitap1. Bu sayede row value dışında birde score kolonu olacak ve veriler scoreu artan şekilde sıralanır yani scoru düşük olan daha önde olur. Valuelerimiz yine burada benzersiz olmak zorundadır. Ama score değerlerimiz benzersiz olmak zorunda değildir. Sıralama yapmak istersek yani verileri görmek istersek ise ZRANGE kitaplar 0 -1 yazarsak tüm dataları gösterecektir 0 başlangıç indexi -1 ise bu indexe kadar olan dataları göster anlamına gelir. Eğer dataları score değerleri ile birlikte görmek istersem ise ZRANDE kitaplar 0 -1 WITHSCORES komutunu kullanabiliriz. Data silmek istersem ise ZREM kitaplar kitap22 komutunu kullanabiliriz.

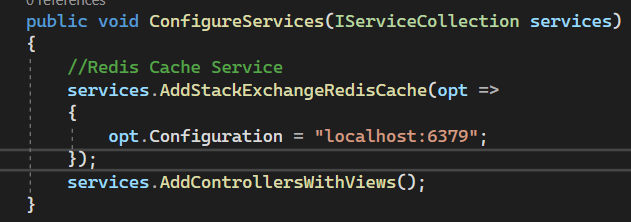


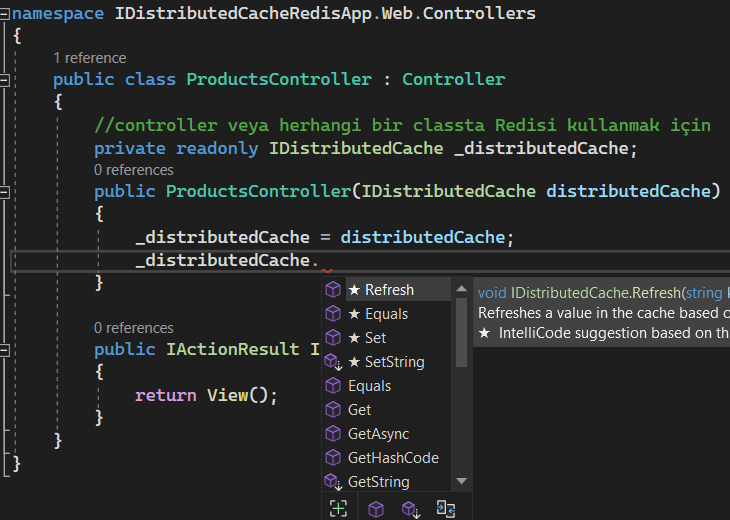
* Şimdi ise başka bir tür olan Redis Hash veri türünü görelim. Bu ise dizi olarak key value şeklinde dataları tutacağımız c# tarafındaki Sözlük yapısına benzeyen veri tipidir. Key value olarak siz bir array tutabilirsiniz. CLI komutlarını inceleyelim. Data haydedebilmek için HMSET sozluk pen kalem komutunu kullanabiliriz. Burada key sozluk oluyor pen sözlük yapısının keyi oluyor kalem ise valuesi oluyor. Bu dataları okumak için ise HGET sozluk book komutunu kullaabilriiz komuttan sonra key yazıyoruz sonra ise tablo içindeki key değerini veriyoruz. Data silmek için ise HDEL sozluk book konutunu kullanabiliirz data okumak ile aynı mantık. Datalarımızın hepsini listelemek istersek ise HGETALL sozluk yazabiliriz. Sozluk keyi içindeki tüm dataları verecektir.



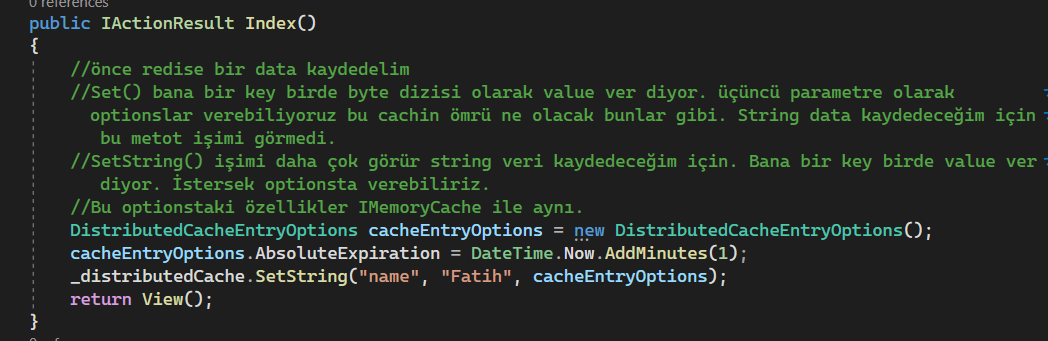
**BÖLÜM 4 ASP.NET CORE + REDİS (IDİSTRİBUTEDCACHE)**

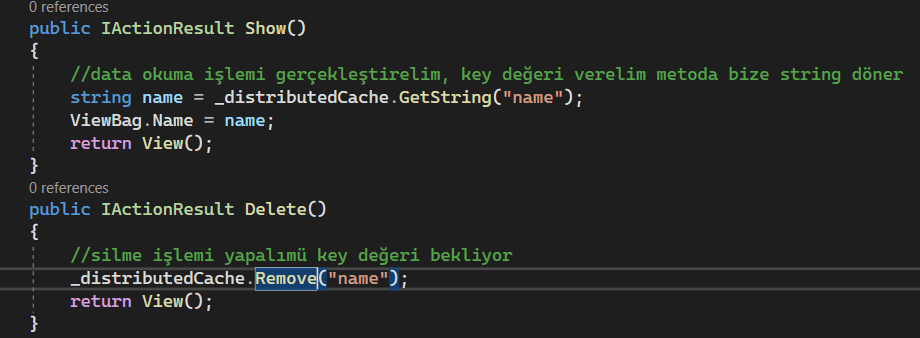
* Asp.Net Core uygulamalarımızda redisi nasıl kullanabiliriz bunu inceleyeceğiz. Core tarafında Redis kullanmak için 2 adet yolumuz var. Bunlardan bir tanesi Core uygulamalarınızda IDistributedCache intefacesini herhangi bir controller’ın ctorunda implemente yani DI olarak geçerek çok hızlı bir şekilde datalarınızı Redise cachleyebilirsiniz. IDistributedCache üzerinden Get ve Set metotları geliyor bunlar ile beraber dataları uzak sunucudaki Redisteki memory’e cachleyebilirsiniz. İkinci bir yol ise eğer ki siz Redis üzerine daha fazla hakimiyet kurmak istiyorsanız, Redis tarafındaki tüm veri tiplerini kullanmak istiyorsanız ve Redis CLI aracılığıyla mevcut olan tüm komutları kullanmak istiyorsanız o zaman daha farklı bir API kullanarak uygulamamızı inşa edeceğiz. Bu bölümde daha kolay olan ilk yöntem kullanılacak diğerini daha sonra anlatacağız. Redis üzerinde çok fazla işlem yapmak istemiyorsak sadece get ve set yapmak istiyorsak, Redis tarafındaki veri tipleriyle öyle çok çalışmak istemiyorsak bu ilk yöntemi kullanmak daha mantıklıdır.
* IDistributedCache interfacesini kullanabilmek için uygulamama Microsoft.Extensions.Caching.StackExchangeRedis kütüphanesini kurmam gerekiyor. Daha sonra AddStackExchangeRedisCache servisini startup tarafında ekliyor olacağız. Bu servisim içerisinde portumu ve pathimi hangi host üzerinde çalıştığıyla ilgili belirledikten sonra uygulamamın herhangi bir yerinde hangi classta kullanmak istiyorsam o classın ctorunda dependency injection olarak IDistributedCache interfacesini geçip kullanacağız.
* IDistributedCache interfacesinin bize sunduğu metotlara bakarsak aslında çok bir şey yok. Çok basit anlamda herhangi bir veri tipi olmaksızın data kaydetmek istiyorsak bu işlemleri bize sağlıyor. Burada hiç Hashleri Listleri Setleri göremiyoruz. Şu Listten bana şu datayı getir diyemiyoruz. En basit haliyle redise key value çiftleri kaydedeyim diyorsak bu ihtiyacımızı karşılar.



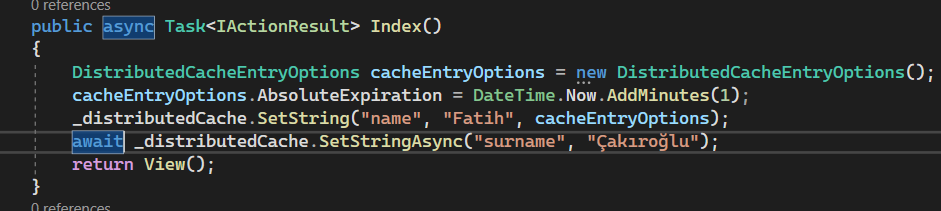


* Şimdi IDistributedCache interfacesi üzerinden basit dataları nasıl Redise kaydedebiliriz veya silebiliriz bunu inceleyelim.

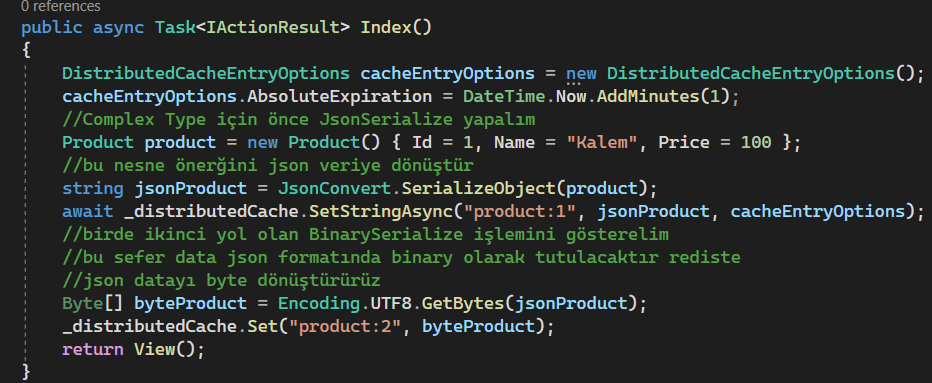


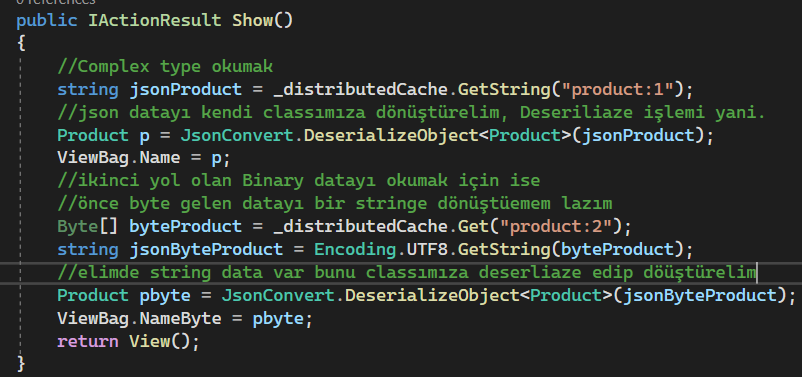


* Şimdi bu işlemlerin asenkron versiyonlarını inceleyelim.



* Şimdi ise Complex Typeları nasıl cachleyebileceğimize bakacağız. Complex derken en yaygın olan kavram uygulamamız içindeki classlarımız mesela. Classlardan bir nesne örneği aldığım zaman bu nesne örneğini nasıl cachleyebiliriz. Bunun için 2 adet yol var. JsonSerilalize işlemi gerçekleştirmek veya BinarySeriliaze işlemi gerçekleştirirsiniz ikisi de tercih edilebilir ama JsonSerialize daha basittir.



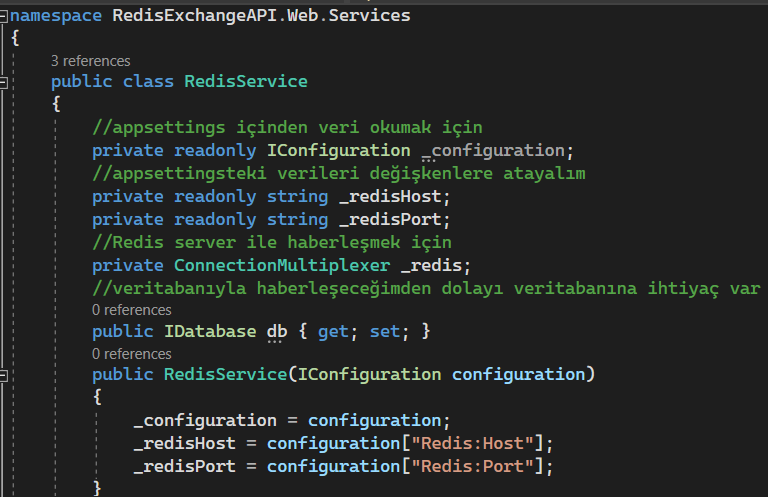


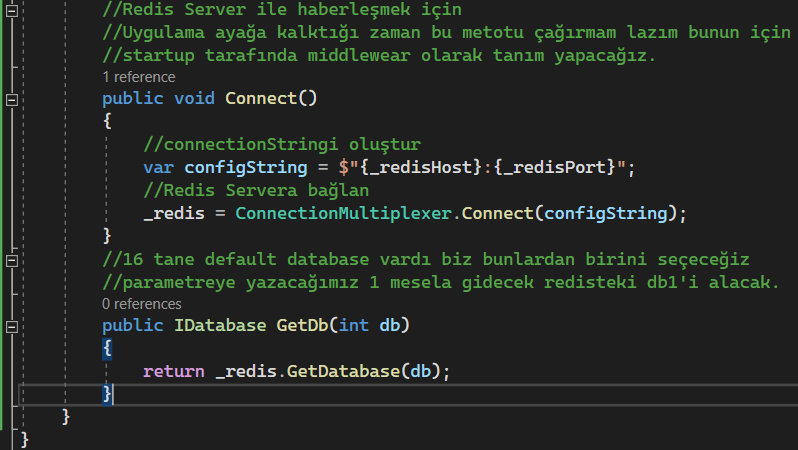
* Şimdi ise dosyalarımızı (image, pdf) gibi cachleme işlemine bakacağız.

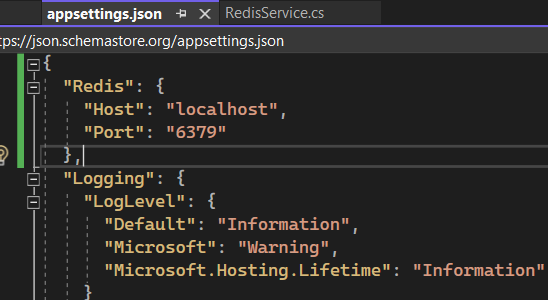


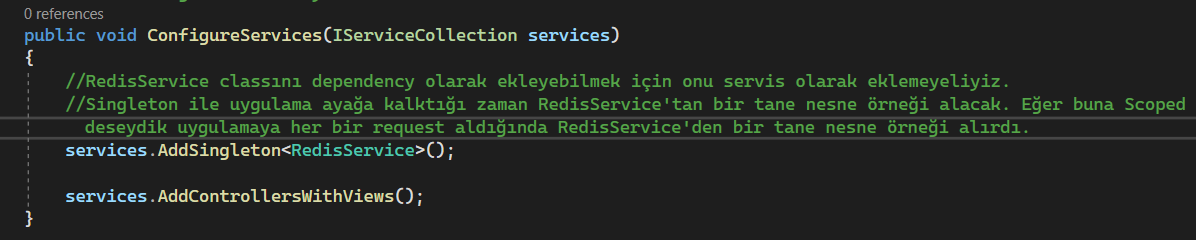
**BÖLÜM 5 ASP.NET CORE + STACKEXCHANGE.REDİS API**

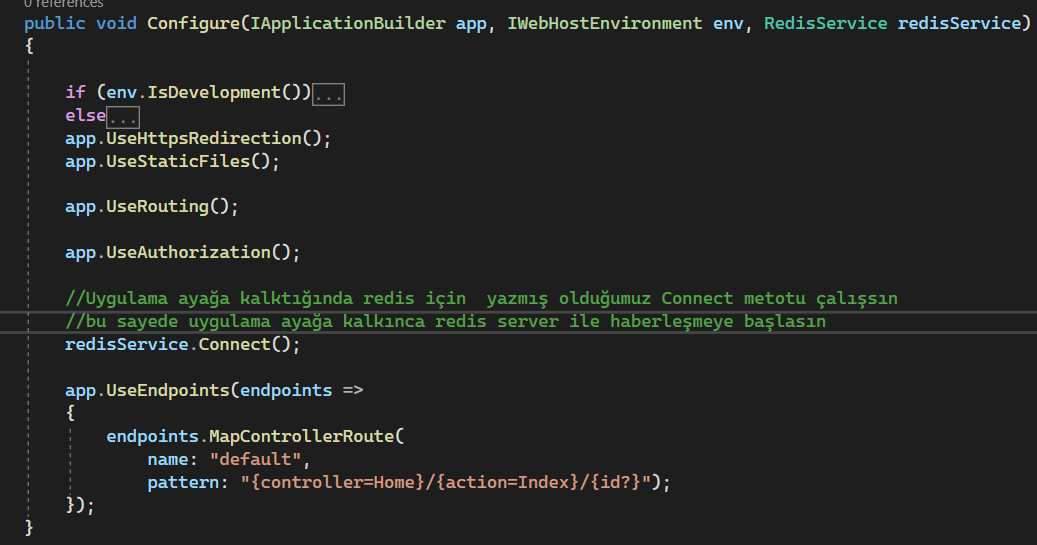
* Bu bölümde redis üzerinde daha fazla hakimiyet elde edebilmek. Redis üzerindeki veri tipleri ile daha rahat çalışabilmek için StackExchange.Redis API’ını kullanıyor olacağız projelerimizde. Bu API üzerinden biz redis tarafındaki tüm veri tipleriyle çalışabilir aynı zamanda Redis CLI tarafında görmüş olduğumuz komutların çoğunu redis API ile beraber bize birer metot olarak gelecek yani metot isimleri ile redis CLI tarafındaki kullanmış olduğumuz komutlar birbirine çok yakın bu şekilde metotların ne iş yaptığını kolay bir şekilde anlayabiliriz. Ayrıca bu API nin çok güzel bir dokümantasyonu mevcut. Biz projemizde daha fazla Redis Serverin bize sunmuş olduğu tüm olanakları kullanmak istersek bir API kullanmak daha uygun bir tercih olacaktır. Çok küçük birkaç tane data kaydedeceğim diyorsanız yukarıdaki gibi ilk yolu kullanabiliriz hiç Redis API kullanmaya gerek yoktur.
* Bu nokta classlarımızdan dependency injection ile redise bağlanmak için kendi servicemizi hazırlayacağız bu service classımız redis ile haberleşecek bizde bu class üzerinden redise bağlanacağız en uygun yöntem budur. Yeni bir proje oluşturup bu projeye StackExchange.Redis paketini kuruyoruz bu sefer. Arkasından bunu uygulamalarımda kullanabilmem için yine dependency injection olarak istediğim classın ctorunda geçeceğim. O yüzden projemize Services adında klasör oluşturup içerisine RedisService adında bir class oluşturalım. Bu sefer redise bağlanacak olan portu appsettings içerisinde belirleyeceğim. Uygulamamın herhangi bir yerinde Redis ile haberleşmek istediğim zaman ilgili classta dependency injection olarak bu oluşturduğumuz RedisService classını kullanacağız.



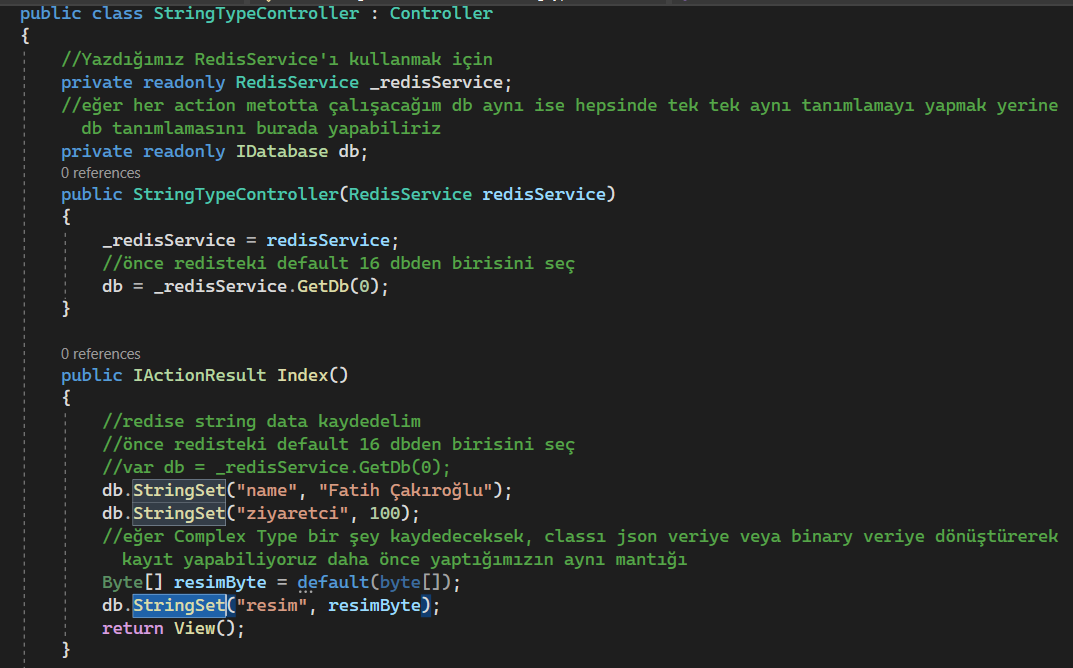


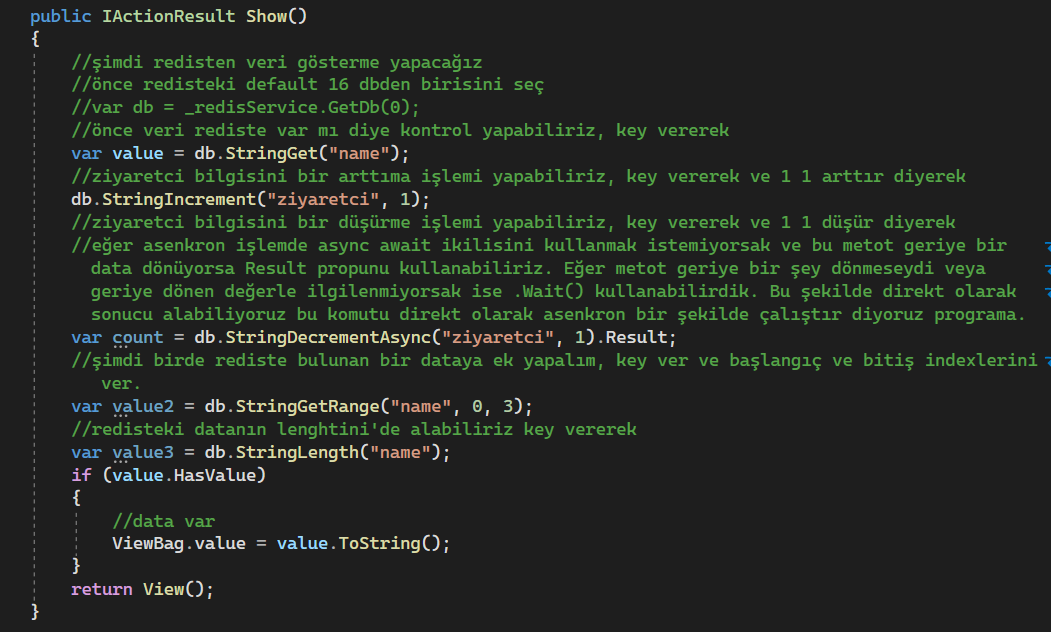




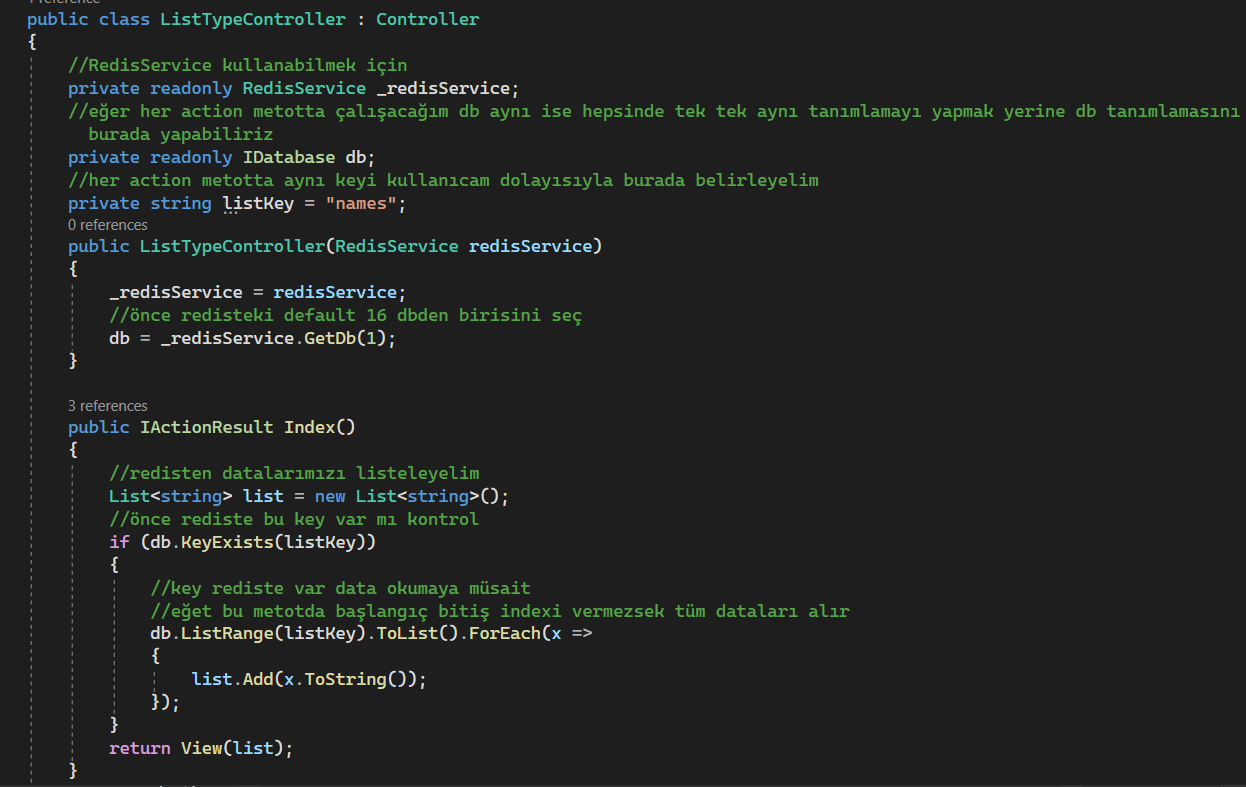


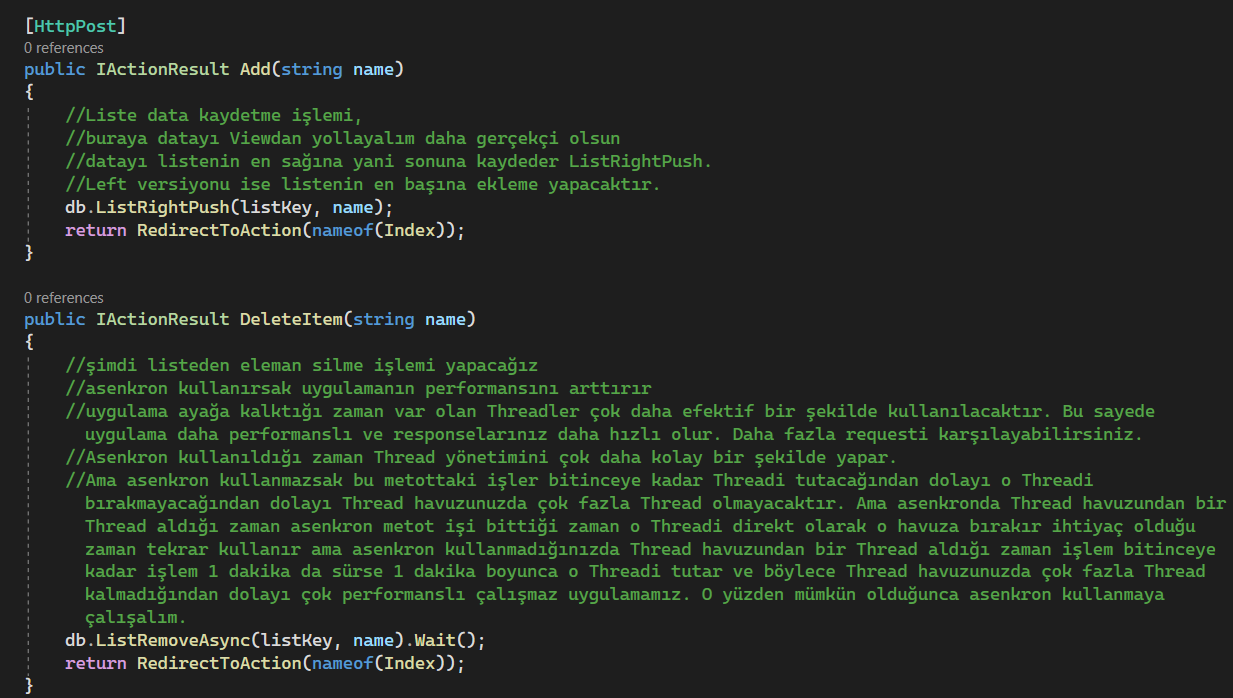
* RedisService içinde oluşturduğumuz Connect metotunu startup tarafında kullanarak uygulama ayağa kalktığında bu metodun çalışmasını sağlayarak redis server ile haberleşmeye başlayacağız. GetDb metodunu ise istediğimiz class içerisinde default olarak gelen 16 databaseden birini seçmek ve daha sonra bu database üzerinden redis server ile haberleşmek için kullanacağız.
* Şimdi oluşturduğumz RedisService classı üzerinden Redis API üzerinden string veri tipleri ile çalışacağız.

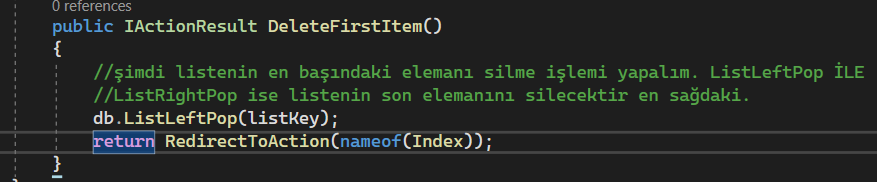




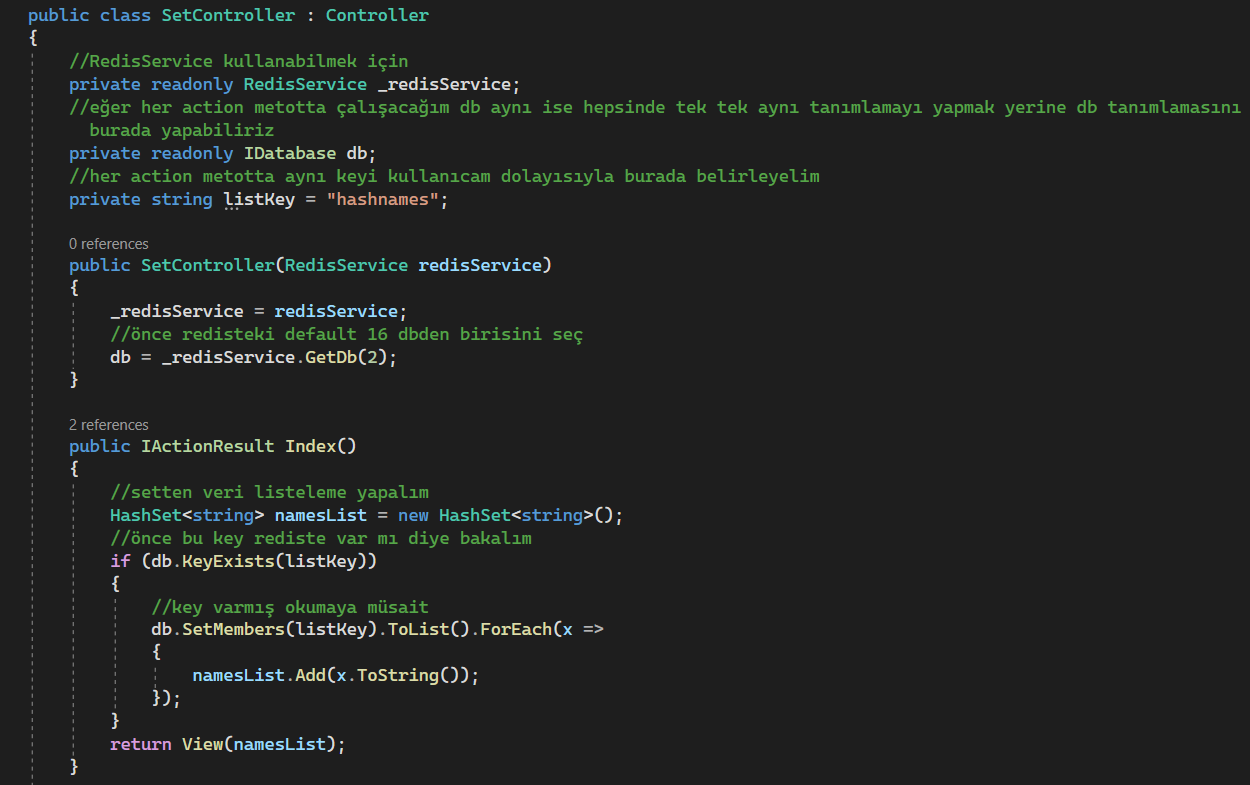
* Şimdi ise ikinci veri tipi olarak Redis List veri tipini göreceğiz. Bu veri tipinin C# tarafındaki karşılığı LinkedList idi. LinkedList’in normal liste göre biraz daha gelişmiş metotları vardı. Herhangi bir datayı listenin başına veya sonuna ekleyebiliriz, silebiliriz. Normal bir Liste göre biraz daha fazla metot geliyor.

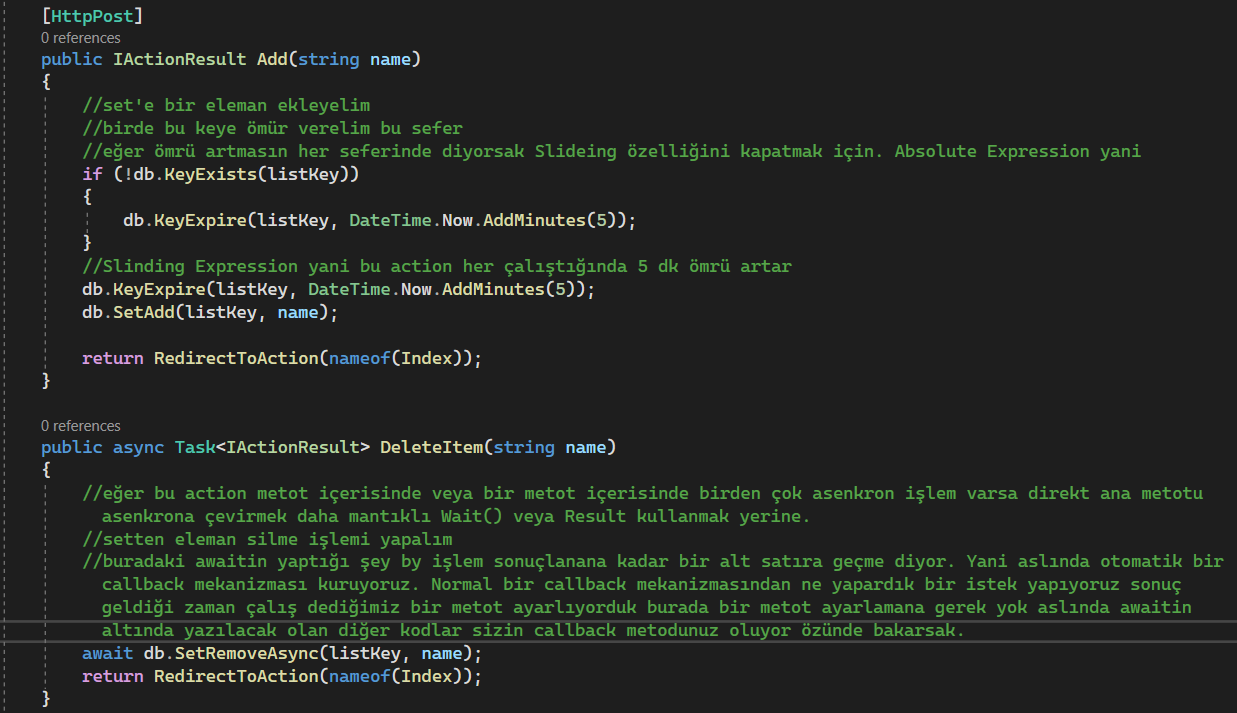




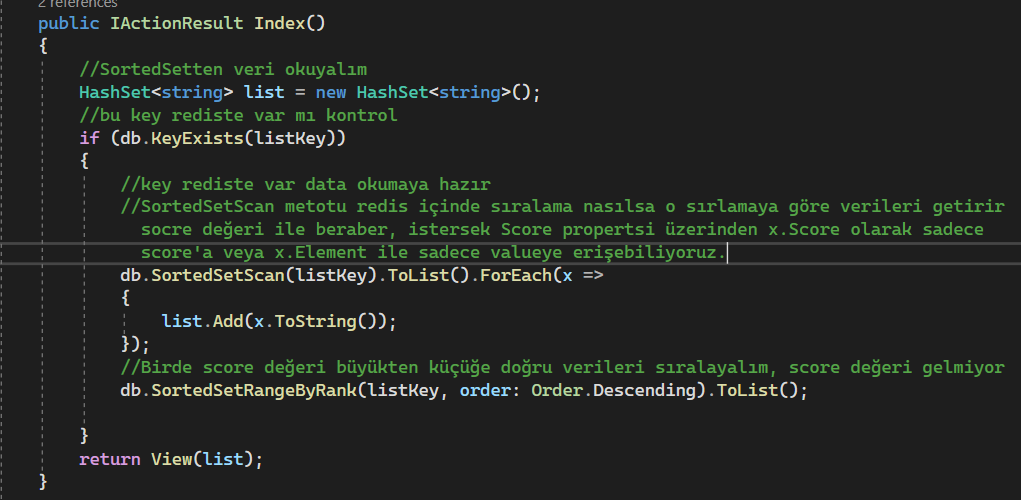


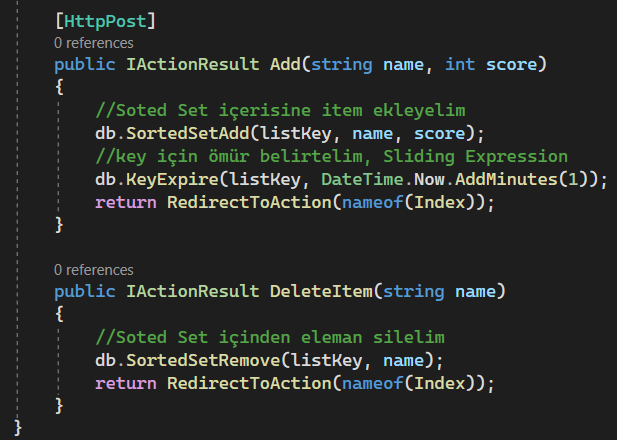
* Şimdi üçüncü veri tipi olarak Redis Set veri tipini inceleyelim. Bu veri tipinin C# tarafındaki karşılığı ise HashSet veri tipiydi. C#taki bu veri tipi liste benzer bu listeye eklenen veriler sırasız bir şekilde eklenir ve her bir data benzersizdir.
* Şu ana kadar hiç keylerimize catch için ömür belirtmedik. Burada API tarafında önceki konulardan bir farklılık var bu tarafta Sliding Expression yok sadece Absolute Expression var. C# taki TimeOut’a benziyor Absolute Expression. 5 dakika dersek 5 dakika sonra ramden data silinir. Memoryden düşeceği kesin zamandır. Eğer Redis API tarafında datalarımıza Sliding Expression Expression kazandırmak istiyorsak burada bir TimeOut() metodu var bu metodu her güncellediğimizde Sliding Expression özelliği kazandırmış oluruz. Yani normalde yok ama sonradan kazandırabiliyoruz.



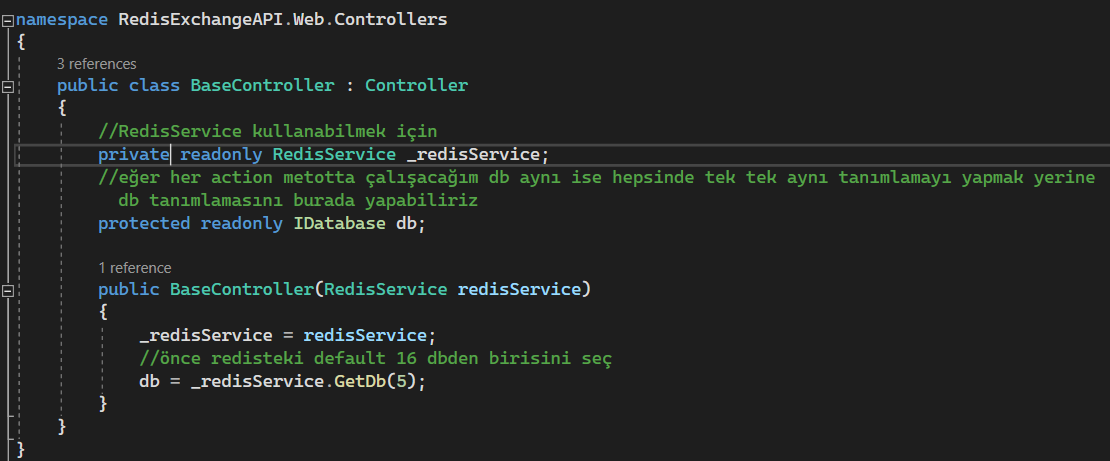


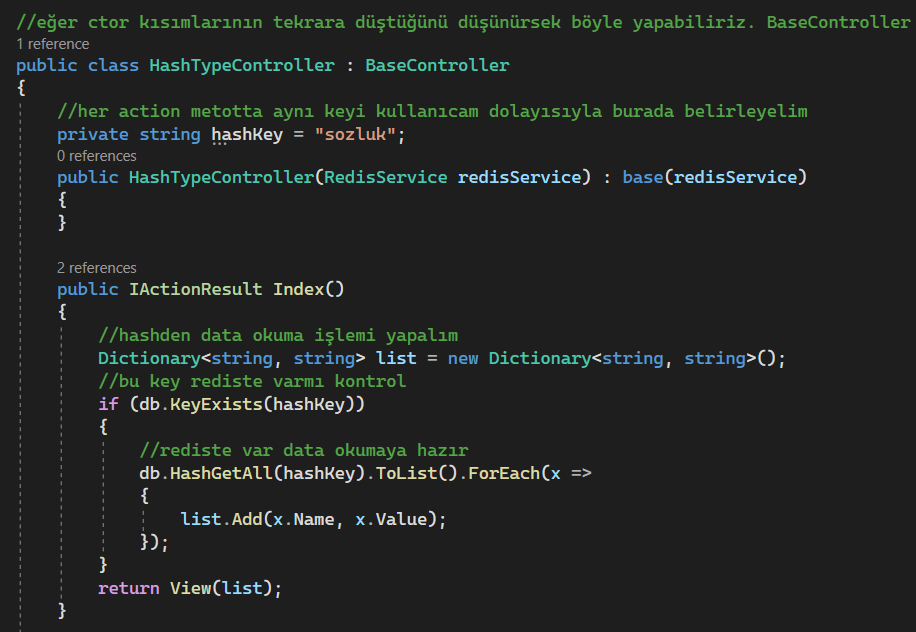
* Şimdi dördüncü veri tipi olarak Redis Sorted Set veri tipini inceleyeceğiz. Sorted Setin normal Setten tek farkı set içerisine item eklerken bir score değeri belirtiyoruz ve sıralama bu score değerine göre ya küçükten büyüğe ya da büyükten küçüğe sıralama yapabiliyoruz. İstersek set arasında da score belirterek item ekleme işlemi yapabiliyoruz.

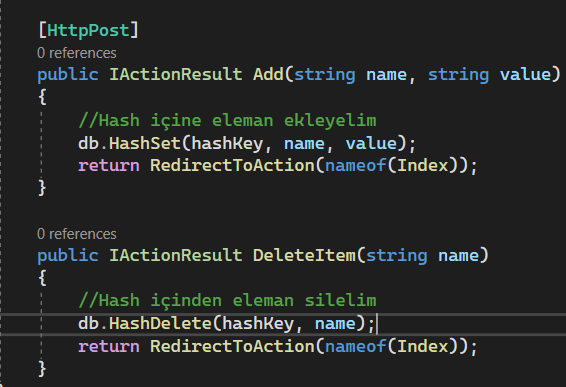




* Şimdi beşinci veri tipi olan Redis Hash veri tipini inceleyeceğiz. Hash veri tipi bizim C# tarafındaki Dictionary’e denk geliyordu. Key value çiftlerini bir array olarak tutuyor.
* Eğer bu zamana kadar contollerlarımzın üstlerinde ctor kısımlarında vs hep aynı tekrarı yapıyoruz böyle yapmayalım diye düşünürsek şöyle bir şey yapabilriiz. Controllers klasörünün içine BaseController oluşturuyoruz.

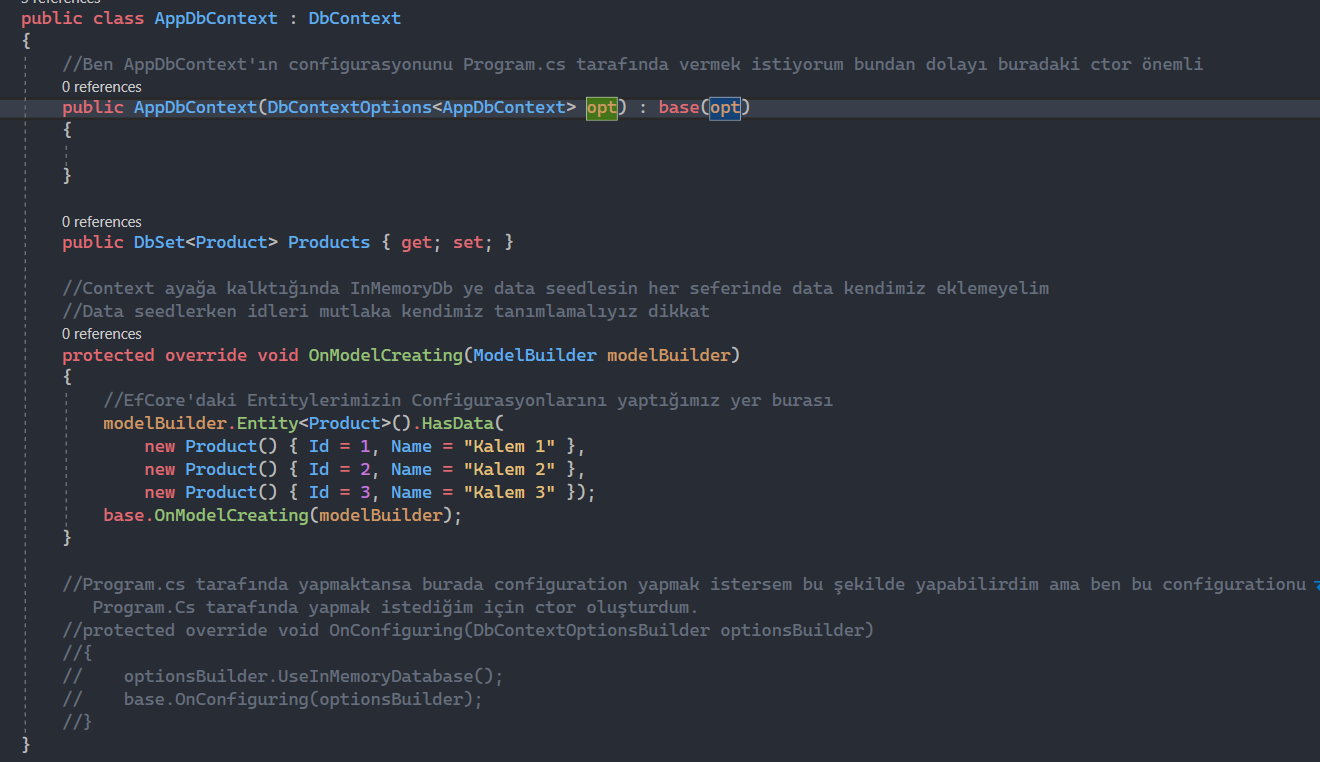


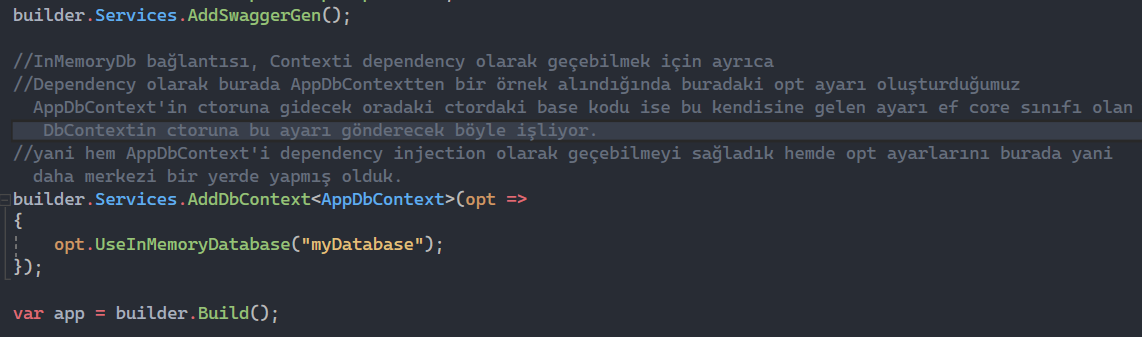


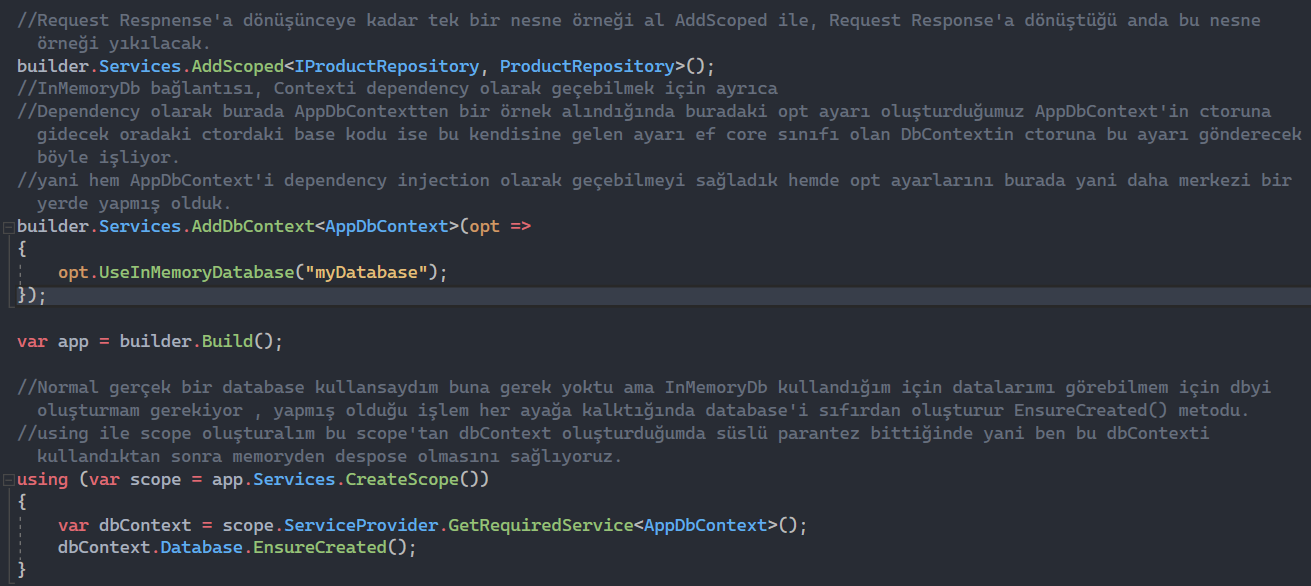


**BÖLÜM 6 ÖRNEK SENARYO .NET 7**

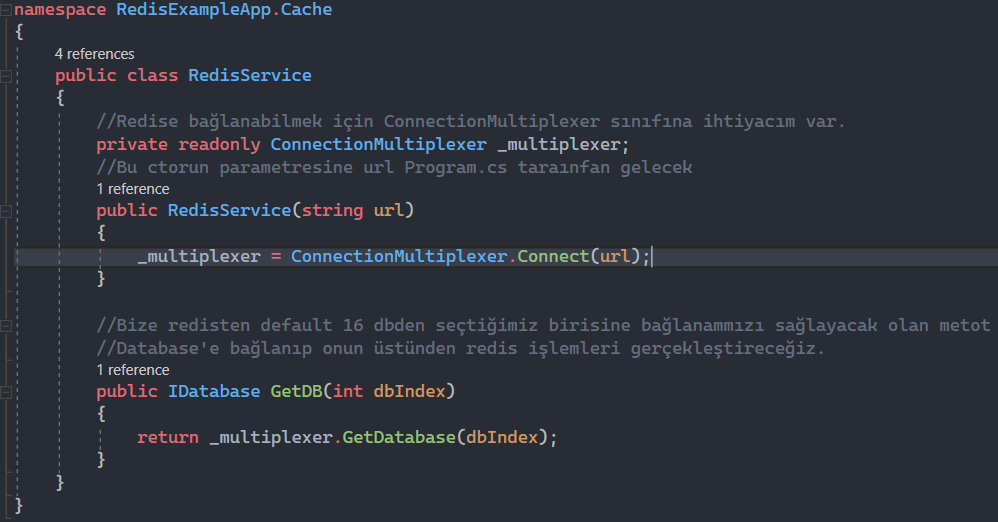
* Şimi .Net 7.0 ile bir proje oluşturalım. İçerisinde Ef Core kurarak repositorylerimizi vs oluşturarak küçük bir proje haline getireceğiz ve sonra bu projeye redis implement edeceğiz. Projede dataları saklamak için gerçek bir veri tabanı kullanmak yerine InMemory DB kullanacağız yani dataları bellekte tutacağız. Amaç özellikle bir şeyleri test etmek istediğinizde yani hızlı bir uygulama ayağa kaldırmak istediğinizde veya özellikle unit test tarafında Integration Testlerimizi yazarken gerçek bir db kullanmak yerine InMemory DB kullanırsanız testleriniz çok çok daha hızlı bir şekilde gerçekleşecektir. Sonuçta memorye yazdığı için normal bir veri tabanına yazmaya göre çok fazla hız farkı olacaktır. InMemory DB lerde gerçek bir ilişki yoktur çoka çok bire çok gibi. Normal bir databasede bire çok bir ilişki varsa bir category sildiğinizde bu categorye bağlı olan tüm productlar silinir. Ama InMemoryde böyle bir şey yok categoryi silsek bile bu categorye bağlı productlar aynı şekilde kalır. Ayrıca migration yapmamıza gerek yoktur. Bu InMemoryDb kullanmak için projeye ef core dışında birde Microsoft.EntityFrameworkCore.InMemory paketini kurmamız gerekiyor.
* NET 6.0 dan itibaren modellerimizdeki proplar vs de stringler yani null olabilecek değerler tanımladığımızda bize uyarıda bulunur compailer ona public string? Name { get; set; } diyerek yani ? koyarak compailera bu alan null olabilir ben okey veriyorum benim bilgim dahilinde demiş oluyoruz ve uyarı gidiyor. Bu uyarıyı vermesindeki amaç en çok hataların null exceptiondan kaynaklandığından dolayı size compailer null olma ihtimali olan yerleri size uyarıyor.
* DbContextlerimiz içerisinde neden ctor yapıyoruz sorusu

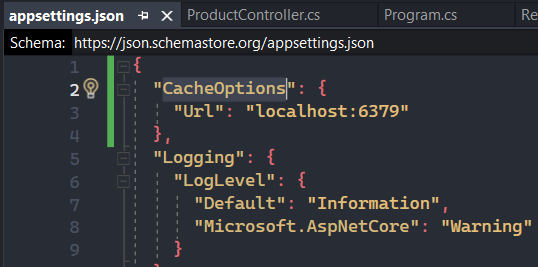


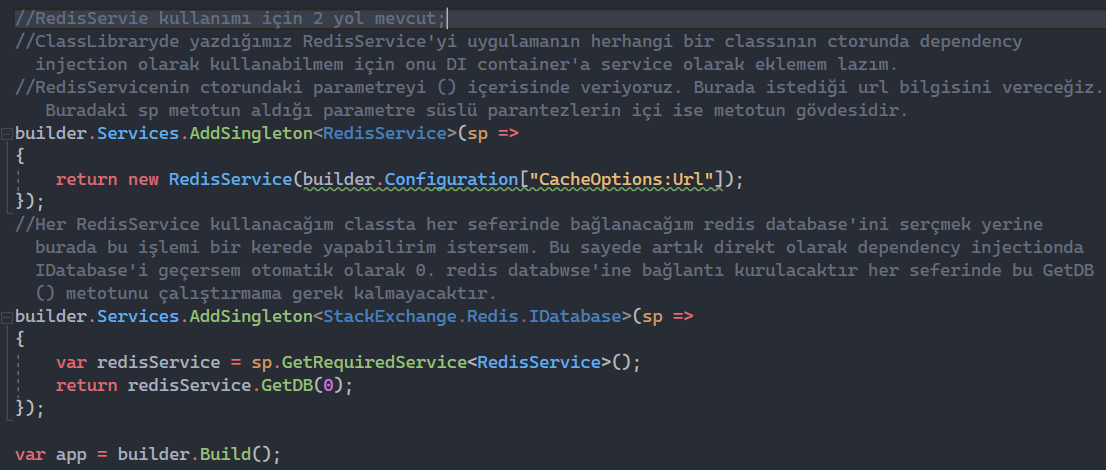




* Redisi docker run -d -p 6379:6379 - -name redis redis diyerek docker container olarak ayağa kaldırmalıyız. Redisi ayağa kaldırmanın en uygun yolu budur. Redisin windwos tarafında resmi bir kurulumu yoktu. Ya bunu bir Cloud tarafında ayağa kaldırmamız gerekiyordu ya da kendi bilgisayarımızda ayağa kaldırmak istersek en güzel çözüm docker içerisinde bir container olarak ayağa kaldırmaktır. -d dediğimizde bu container ayağa kalktığı zaman bu containera bağlanma içindeki logları alma demiş oluruz. -p ile beraber ise kendi localimdeki portu container içerisindeki porta mapliyoruz. İlk redis benim containerimin ismi olacak sonraki redis ise kullandığım imagenin ismi oluyor. Sonuna tag vermediğimiz için en son redis image versiyonu neyse onu çekecektir.
* Şimdi projemizde Cache ile ilgili kodları tutacağımız ayrı bir CacheClassLibrary katmanı oluşturacağız. Bu katman içerisinde RedisService classımızı oluşturacağız. Bu classı Singleton olarak DI container’a ekleyeceğiz. Singleton yapma sebebimiz bu class içerisinde biz ConnetcionMultiplaxer isminde redisin bir sınıfını kullanacağız RedisExchange kütüphanesi diyor ki bu ConnetcionMultiplaxer sınıfını her zaman uygulama ayakta olduğu sürece tek bir nesne örneği tut diyor çünkü bu ağır bir nesne her seferinde yeni nesne örneği üretilmesin. Bizde RedisExchange kütüphanesinin nu Best Preatice’ine uygun bir kodlama gerçekleştireceğiz.
* ClassLibrary oluşturma sırasında bir BestPreactice var Framework seçerken .NET Standardt 2.0 2.1 gibi olanlar şu anlama geliyor bu class libraryleri klasik .Net frameworkte, Xamarinde, .Net ekosistemindeki farklı projelerde de kullanabilirsiniz yani bu class library her yerde geçerli anlamına geliyor. Ama Core ile ilgili yaparsak long veya priview olanları sadece Core olan frameworklerde kullanabiliriz örneğin bir windowsta kullanamazsınız çünki Windows core değil. O yüzden aklımızda olsun Class Librarylerimizi olabildiğince .NET Standard ile oluşturun bunuda bir best preatice olarak söyleyelim. Hem .Net Core’da kullanabilicem hem de core olmayan bir mimaride örneğin Xamarinde de bu ClassLibraryi istersem kullanabilirim bu sayede.
* Şimdi bu yeni ClassLibrarymize StackExchange.Redis kütüphanesini kuralım.
* Artık redisle bağlantıyı sağlayacak olan RedisService classımızı yazamaya başlayabiliriz.







* Şimdi var olan uygulamamıza Cache yapısını entegre edeceğiz. Çünkü redisi daha çok cache tarafında kullanıyoruz. Diyelim ki bize Product ile ilgili dataların hepsi cacheden gelsin denildi. İlk aklımıza gelen yöntem Product Controller’a hemen dependency injection olarak oluşturduğumuz RedisService’mizi entegre etmek. Daha sonra listeleme endpointlerinde ilgili data rediscache’de var mı yok mu varsa catcheden dön yoksa git dbden dön şeklinde yazabiliriz ama bu doğru bir yaklaşım değil. Ben bu durumda ne yapmış oldum uygulamama bir özellik eklemek istediğimde var olan classta bir değişiklik yaptım. Bu istediğimiz bir olay değil yazılımda bir özellik eklerken mümkün olduğunca var olan koda dokunmadan bu özelliği eklemek bizim için en uygun yöntem olacaktır. Burada aklıma gelen 2. Yöntem ise Controllera dependency injection olarak RedisServiceyi eklemek yerine ProcutRepository’e eklemek ve ilgili işlemi orada yapmak. Bu seferde repository classta değişiklik yapmış oldum hemde solidin ilk prensipi olan single responsibility’i çiğnemiş oldum çünkü bir class bir iş yapmalı bir contexte sahip olmalı. Ama bu durumda ProductRepository hem cache ile ilgili kodları içerecek hemde db ile ilgili kodları içerecek bu doğru değil. Bu Repository sadece db ile ilgili kodları içermeli. Bu noktada hem Controller hem Repository kodlarımızı değiştirmeden uygulama Cache yapısını entegre etmek için yardımımıza Decorator Design Pattern yetişiyor. Çözdüğü problem şu bir uygulamanın kodlarını değiştirmeden o uygulamanın yeni davranışlar eklemenize imkân veren bir design paterndir. Sadece Cachleme değil, Loglama gibi birçok uygulamaya yeni ekleyeceğiniz featureları uygulamanın var olan kodlarında değişiklik yapmadan eklemenize imkân veriyor. Burada 3 design patterni bilmek çok önemli Decorator, Adapter, Strategy bunlarun üçü çok ömemli. Design patternları anlamak için en başta Has A is A ilişkisini anlamak önemli.
* Design patternların ezberlememiz gereken kısımları çözmüş olduğu problemlerdir. Yani UMl Diagramlarını ezberlememize gerek yok önemli olan çözüm bulduğu sorunların ne olduğunu bilmektir. Örneğin Zaten yukarıda Decorator Design Patternin neyi çözdüğünü söyledik başka örnekler mesela Strategy Design Pattern runtime’da algoritma değişikliği, Adapter Design Pattern bir projeye yeni third party librarylerin var olan projenin yapısını bozmadan bu third party libraryleri nasıl ekleyebilirim bunu çözüyor.
* Decorator Design Pattern için ayrı bir repository oluşturacağız ProductRepositoryWithCache isminde bu class hem cache ile ilgili kodları kontrol edecek hemde cachede yoksa datayı dbden çekecek artık bu classlın kendi içerisi haricinde uygulumamamın harhangi bir yerinde dependency injection olarak IProductRepository geçildiğinde Prdocut repository örneği değil bu yeni classımın örneği alınacak bu yeni classın kendi içerisinde IProductRepository dependency olarak geçilirse ise bu sefer eski repositorymiz yani ProductRepositoryden bir örnek alınacak olay bunun üzerine kurulu olacak bu ayarlamayı ise daha sonra Program.cs tarafında uygulamamıza haber edeceğiz. Contoller içerisinde dependency olarak Business Service kullanalım kullanmayalım şimdi ek yazacağımız kod her zaman Repository’i referans alacaktır. Yani bu yeni classımızı kesinlik business servicelerimizden değil repositorylerimizden miras almalıyız. Çünkü bizim yacağımız cache kodumuz DB ile ilgili bir koddur. Sonuçta Redis No-SQL bir veri tabanı biz sadece onun catchleme özelliğinden de faydalanıyoruz dataları memoryde tuttuğu için. Dolayısıyla Repositories klasörümüze yeni bir repository ekleyeceğiz bu sefer ismi ProductRepositoryWithCache olacak. Bu classımız IProductRepository interfacesinden miras alacak ama bu sefer datalar dbden değil redis catchden gelecek.
* Cache tarafında datalarımızı Redis Hash olarak tutacağız key value çiftleri ile yani. Key kısmında product ıd sini kullancağız value kısmında ise tüm product bilgilerini tutacağız bu sayede key üzerinden yani Id üzerinden bir arama yaptığım zaman çok hızlı bir şekilde verileri alabileceğim. Yani işlem ID => value product class şeklinde olacak diğer veri tiplerinde ise önce tüm datayı memorye almam lazım memory üzerinden datayı aramam lazım. Hashte ise direkt olarak key yani ID üzerinden daha hızlı arama yapabilirim.
* Yeni Repository classımı oluşturduktan sonra Program.cs tarafında uygulamaya diyeceğim ki sen ProductRepositoryWithCacheDecorator classının dışında uygulamanın herhangi bir yerinde dependncyinjection olarak IProductRepositoryi görürsen artık ProductRepositoryWithCacheDecorator sınıfından nesne örneği üreteceksin ama ProductRepositoryWithCacheDecorator classının içinde dependencyinjectionda sen IProductRepository interfacesini görürsen bu sefer bizim normal ProductRepository classımızından nesne örneği alacaksın diyeceğiz mantık böyle çalışacak.

