### **SignalR Nedir?**

Normal bir web yazılımı, klasik olarak Client ile Server arasında kullanıcıdan gelen talebe/request karşılık, Server tarafından verilen tepki/cevap/response ilişkisi üzerinden eşzamanlı olarak haberleşmektedir. Lakin bu durum hepimizinde yıllarca deneyimlediği gibi bir bekleme süreci yahut sayfanın gidip gelmesiyle sonuçlanmakta ve böylece kullanıcı açısından zamansal maliyetle birlikte, deneyim açısından günümüze yakışmayan ilkelliğe alamet etmektedir. Şöyle bi düşünürsek eğer; ülke genelinde yapılan seçimin sonuçlarını anlık olarak takip eden bir web yazılımının beslendiği kaynağa gelen verileri dinamik olarak yorumlayıp göstermesi gerekirken, burada güncel verileri görebilmesi için kullanıcıdan sayfayı yenilemesini istemek sizce ne kadar modern bir hizmet olacaktır? Ya da günümüzün imtihanı Covid-19(nam-ı değer Korona Virüs) verilerini yayınladığınız güncel bir web uygulamasının anlık olarak girilen verilerinin grafiksel olarak anında yansıtıldığını tahayyül ederseniz, klasik web yaklaşımının(request/response) bu süreçte ne kadar efektif bir rol üstlenebildiğini söyleyebilirsiniz…

Günümüz ihtiyaçlarını değerlendirirsek eğer klasik web yaklaşımının tek başına pek yeterli olmadığı ve çözüm olarak farklı kütüphanelere ve hatta protokollere ihtiyacımız olduğu ve olunacağı konusunda kaçınılmaz olarak hem fikir olduğumuz kanaatindeyim. Misal, yukarıda söz gelimi verilen örneklerde hayali tasarlanan web uygulamalarının ideal olarak Real Time bir hizmet vermesi yerinde olacak ve bunun için HTTP’den farklı olarak TCP protokolonü benimseyen Websocket altyapılı sistemler yahut kütüphanelerin kullanılması gerekecektir.

Asp.NET(Core) mimarisi WebSocket programlayabilmek için birçok farklı kütüphane barındırmaktadır. Bunlardan en popüler ve community(topluluk)’si en geniş olanı SignalR’dır.

**SignalR, web uygulamalarına Real Time fonksiyonellik kazandıran bir Open Source kütüphanedir.**

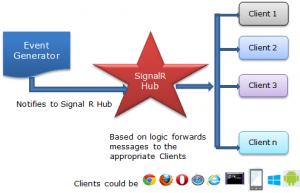
Nasıl ki; Node.JS‘de Socket.IO yahut Python‘da WebSockets onca kütüphaneye nazaran rüştlerini ispatlamışlarsa, Asp.NET(Core)’de de SignalR kendisini ispatlamıştır.

**Peki… SignalR, yapısal olarak nasıldır?**

SignalR altında yatan teknoloji esasında WebSocket’tir. WebSocket, client ile server arasında iki yönli iletişimi sağlayan bir protokoldür. Eğer bir yerde protokol kelimesi geçiyorsa orada bir kurallar bütünü var demektir. Protokol, iki farklı sistemin birbiriyle nasıl haberleşeceğini belirleyen kurallar bütünüdür. SignalR, bu kurallar çerçevesinde client ile server’ı haberleştirmekte ve ilişki yönünü sadece client->server’dan alarak server->client olacak şekilde desteklemekte ve RPC(Remote Procedure Call – Uzak Yordam Çağrısı)’yi benimsemektedir.

**RPC, server tarafından client’ta bulunan herhangi bir metodun(JS) tetiklenmesini sağlayan bir protokoldür.**

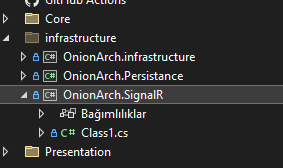
Böylece RPC sayesinde uygulamalar server’dan sayfayı yenilemeksizin data transferini sağlamış olacak ve gerçek zamanlı uygulama davranışı sergileyecektirler. Uygulamanın gerçek zamanlı olması client ile server’ın anlık olarak karşılıklı haberleşmesi anlamına gelmektedir

**SignalR Nasıl Çalışır?**

SignalR’ın kalbinde ‘Hub’ denilen bir merkezi yapı vardır. ‘Hub’ içerisinde tanımlanan metoda subscribe(abone) olan tüm client’lar ilgili Hub üzerinden iletilen mesajı alacaktır. Yandaki şemada görüldüğü üzere bir client tarafından gönderilen herhangi bir mesaj yahut data uygun mantıkta ilişkisel tüm client’lara gönderilecektir. Ve buradaki bildiriyi Hub yürütecektir.

Şimdi hep beraber yapımıza SignalR projemizi entegre edelim ve geçici olarak neler olup bitiyor neler oluyor loglama sistemini signalR ile frontEnd üzerinde gösterelim.  
  
ilk önce mevcut projemizde SignalR’ı nerede bulundurmamız lazım bunu inceleyelim.

Şimdi burada mantığı şöyle düşünmemiz lazım SignalR bir iş üstleniyor kendi içinde iş mantıkları yöntebilmeli.  
Buna ek olarak kendi içinde iş mantıklarıda yürütecek dolayısıyla bunu **infrastructure** içinde tanımlamak gerekir.  
Ama genel olarak bunu bu **infrastructure içinde yeni bir proje olarak oluşturmak tercih edilir.**

**Şimdi infrastructure içinde yeni bir class projesi oluşturalım ve bu proje içinde gerekli kurulumları yapalım.**

Artık projemiz hazır her zaman olduğu gibi ilk yapmamız gereken bu projenin içine gerekli olan kurulum dosyalarını Nuget üzerinden indirmek.   
Burada sırayla beraber gidelim.

**Burada öncelikle 2 farklı klasör oluşturuyoruz.  
Hubs ve HubServices olmak üzere** .

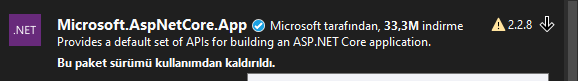
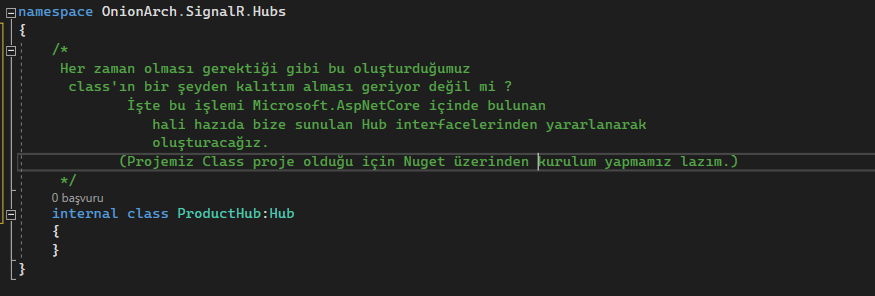
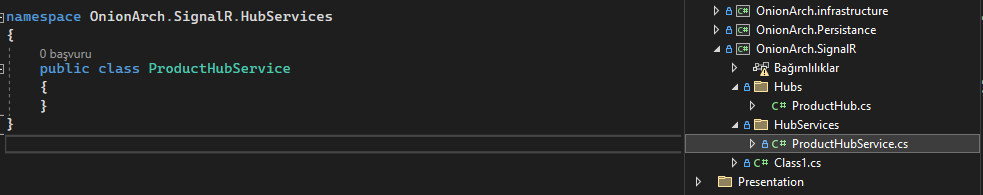
**Hubs ve HubServices nedir ?**

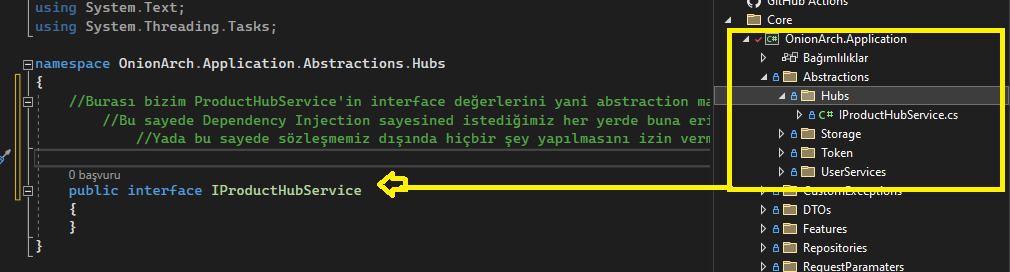
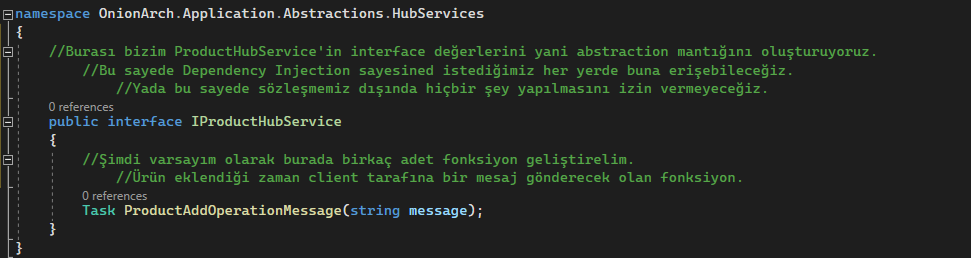
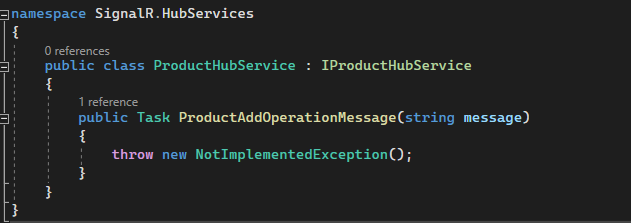
**Hub**'lar, sunucu ve istemciler arasındaki iletişimi yönetmek için kullanılır. İstemciler, bir hub'a bağlandıklarında, hub üzerinden iletişim kurabilirler. Hub'lar, istemcilere veri gönderme ve istemcilerden veri alma yeteneği sunar. Bu, gerçek zamanlı uygulamalarda oldukça kullanışlıdır. Örneğin, canlı sohbet uygulamaları, oyunlar, anlık bildirimler ve diğer gerçek zamanlı uygulamalar, SignalR hub'larını kullanarak sunucu ve istemciler arasında hızlı ve etkileşimli iletişim sağlayabilir.

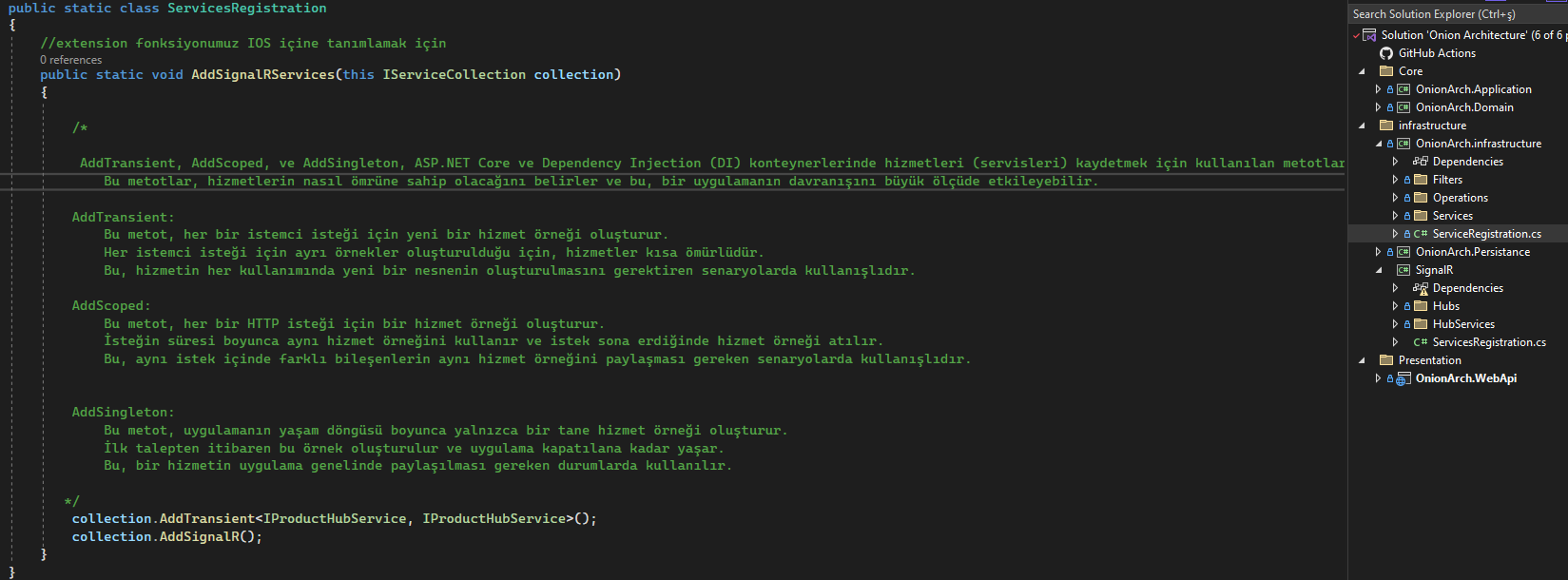
**HubServisler** ise Hublar üzerinden işlem yapmamızı sağlayacak ve bütün mimaride bulunan classlar üzerinden socket işlemlerimizi yapmamızı sağlayacak servislerimiz olacak.  
  
Peki bunu nasıl yapacağız tabiki de Abstraction kullanarak .

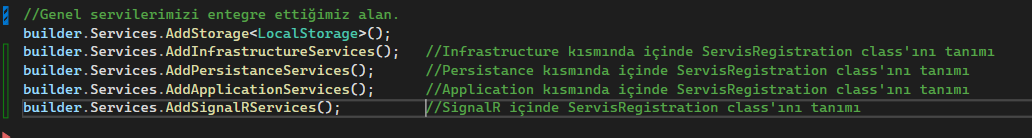
Şimdi beraber productlar üzerinden ilerleme yapalım neden product’ı seçtik bir sipariş oluştu bir ürünün stoğu bitti gibi durumlarda anlık olarak admine veya kullanıcıya bilgi verebileceğimiz şekilde işlem oluşturacağız.

Şimdilik burada test için bahsettiğim product işlemini yaparak öğrenmeye başlayalım.

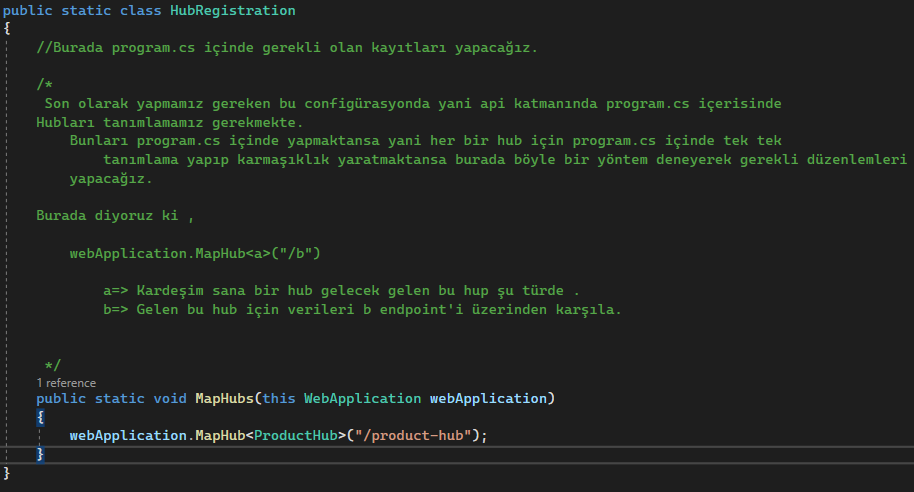
* Öncelikle Hub klasörünün içine girip ProductHub adında bir class oluşturalım.
  + Oluşturduğumuz bu class’ın her zaman olduğu gibi kalıtım yoluyla bazı bilgilere erişmesi gerekiyor değil mi ? İşte burada Hub yapıları için biz hali hazırad olan ve AspNetCore ile geliştirilen hubları kullanacağız.  
    Burada şöyle bir problemimiz var oluşturduğumuz proje Class Library olduğu için içinde AspNetCore.App kurulu değil bunu Nuget üzerinden kurmamız gerekiyor.  
     
* Şimdi gerekli kurulumu yaptıktan hemen sonra gerekli kalıtım ile gerçekleştiriyoruz. **:Hub** diyerek.   
  
* Şimdi gerekli kalıtımı gerçekleştirdik. SignalR’ın temeline göre Hubları kullanarak doğrudan clientlar arasından iletişim kurabiliriz. Fakat mimarisel olarak bu mümkün değildir. İşte burada mimarisel olarak yapmamız gereken işlem her zaman olduğu gibi önce bir interface tanımlamak daha sonra bu interface üzerinden gerekli kurulumları gerçekleştirmek.
* Şimdi öncelikle HubServices klasör altında ProductHub adında bir class oluşturalım.  
  

Daha sorna bu sınıfın interface değerini oluşturalım daha sonra bu interface ile gerekli sözleşmeyi gerçekleştirelim.   
 İlk olarak Abstrack sınıflarımızı biz nerede yapıyorduk . Application katmanı altında değil mi hemen burada mimarimize uygun olacak şekilde gerekli güncellemeleri yapalım.  
   
Artık bu oluşturduğumuz abstrack yapı sayesinde gerekli fonksiyon ve metodları tanımlamaya başlayabiliriz.  
 Şimdilik biz burda bir adet Ürün ekleme işlemlerinde neler olur bitiyor hata mesajı yollayan bir fonksiyon method tanımlayalım.  
  
 Tanımladığımız bu metodu classımıza entegre edelim.  


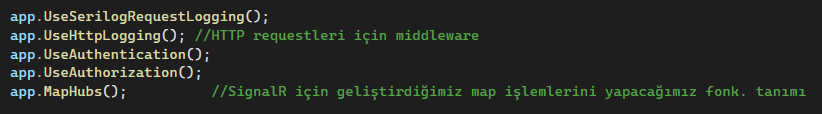
Şimdi yapmamız gereken 2. aşama bu oluşturduğumuz abstrack nesne ve class için IOC kaydını oluşturma.  
 Bunun için SignalR projemizin altında daha önce yaptığımız gibi konfigürasyonlar yapmamız gerekiyor.  


Gerekli konfigürasyonları yaptıktan hemen sonra program.cs içinde tanımlamaları yapmamız gerekiyor.  


Şimdi yapmamız gereken tek bir konfigürasyon daha kaldı bunun için öncelikle program.cs içinde gerekli düzenlemeleri yapabiliriz.  
  
Şimdi bu şekilde düzenlemeleri yaptıkça her bi hub için işlem oldukça fazla şekilde uzayacak. Bunu istemiyoruz bu tercih edeceğimiz bir durum değil biz program.cs dosyamızı oldukça basit bir şekilde bırakmayı amaçlıyoruz.  
Burada şöyle bir yöntem izleyeceğiz,

**HubRegistration** adında yeni bir class dosyası oluşturuyoruz.SignalR projemizin hemen altına.  
Daha sonra bu dosya içine program.cs içinde tanımlayacağız dosyaları tek tek tanımlıyoruz.  


| **Burada diyoruz ki ,**  **webApplication.MapHub<a>("/b")**    **a=> Kardeşim sana bir hub gelecek gelen bu hup şu türde .**  **b=> Gelen bu hub için verileri b endpoint'i üzerinden karşıla.** |
| --- |

**Son olarak oluşturduğumuz bu MapHubs isimli class ‘ı program.cs içinde tanımlıyoruz.  
**

Artık tüm düzenlemeleri bitirdik.Şimdi yapmamız gereken işlem artık oluşturduğumuz ProductHub servisinin nasıl çalışacağını ayarlamak.  
ProducthubService içine giriş yapalım.  
Burada amacımız neydi herhangi bir ürün eklendiği zaman bütün clinetlere bu ürünün eklendiğini bildireceğimiz bir yapı oluşturmak değil mi ?

Şimdi çalışmalara başlamadan önce bizim oluşturduğumuz ve IOC içinde kayıt ettiğimiz  
Bu IHhubContext interface referansı üzerinden içeride hangi Hub çalışacağını belirtmemiz gerekiyor.  
 Biz daha önce IHhubContext tanımlamadık bu nereden geliyor diye düşünebiliriz?   
 => Bu bizim serviseRegistration içinde tanımladığımız addSingralR() üzerinden geliyor.

| **readonly IHubContext<ProductHub> \_hubContext;**  **// Gelen bu interface DependencyUnj. yöntemi ile oluşturuyoruz.  public ProductHubService(IHubContext<ProductHub> hubContext)**  **{**  **\_hubContext = hubContext;**  **}** |
| --- |

Gerekli tanımlamaları yaptıktan sonra **IProductHubService** içinde tanımladığımız imzaları uygulayıp geliştirmelere başlıyoruz.  
  
 Burada Clientler'in hepsine mesaj yollamak istediğimizi;

| await \_hubContext.Clients.All…… |
| --- |

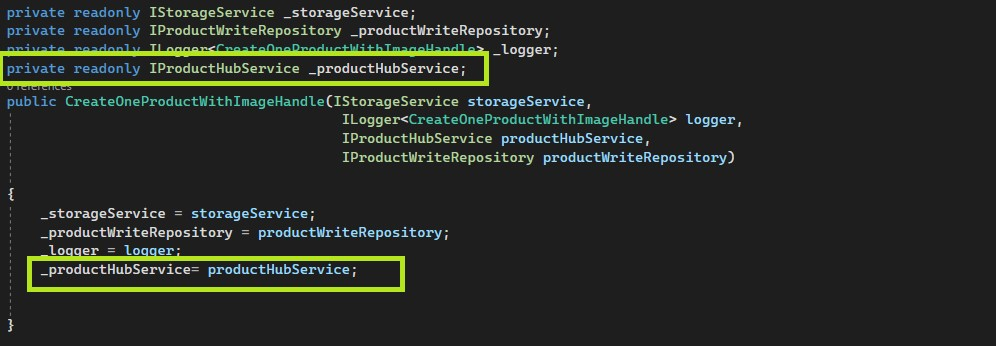
ile tanımladık.Daha sonra ne yapmamız istediğimizi belirtmemiz gerekiyor mesela herkese yollamak.  
  
 SendAsync() Nedir?

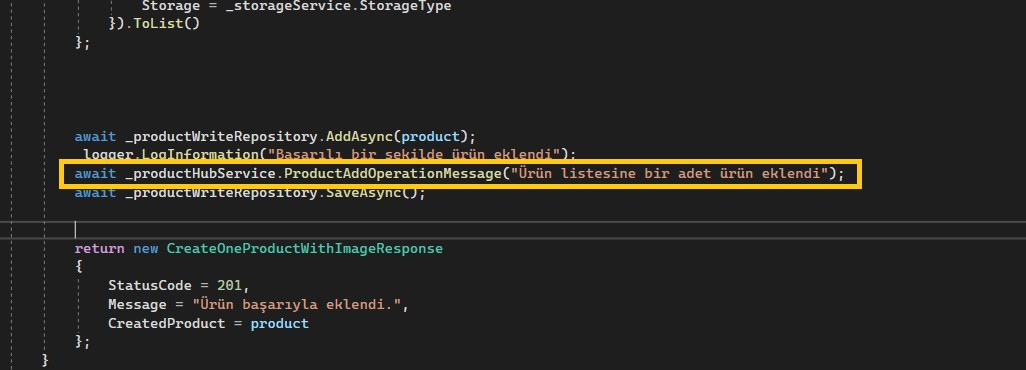
Şimdi bu tanımlamayı yaptıktan hemen sonra hangi fonksiyon ile her yollacacağımız belirtmemiz gerekiyor.  
 Bunun için şöyle bir yöntem izliyeceğiz,  
 İlk parametre hangi fonksiyona karşılık verilecek yani fonksiyon ismini belirteceğiz.,   
 ikinci parametre ise ne yollayacağımızı belirteceğiz.

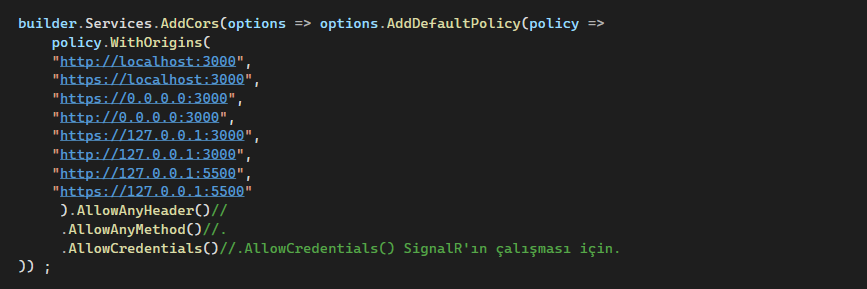
| **await \_hubContext.Clients.All.SendAsync("receiveProductAddedMessage", message);** |
| --- |

Normalde bizim client tarafından tetiklenecek olan fonksiyonların isimlerini string olarak ifade etmek bek kullanılır değil bunları constant bir yapıya çevirmek daha mantıklı Bunun için ayrı bir class içinde **ReceiveFunctionNames** adında bir değişken tanımlayıp Bu değişken üzeriden bu değişkenleri ayarlamak daha mantıklı.

Son olarak yapmamız gereken işlem CreateProductWithImage isimle CQRS pattern içinde gelişitirdiğimiz,   
Handler sınıfı içinde IOC üzerinden kurulum yapıp bu oluşturduğumuz fonksiyona mesajı yollamak.





Son olarak CORS politikalarını uyması için bizim Credentials’lara izin vermemiz gerekiyor program.cs içinde daha önce oluşturduğumuz CORS politikalarına bu seçeneğide ekleyelim.  
  
  
Artık geri kalan tüm işlem frontEnd tarafında olacak.