|  |  |
| --- | --- |
| A picture of a winding road and trees  ASP.NET CORE 3.1 | Bu Döküman ASP.NET CORE hakkında bilgiler içermektedir.  Berkay Akar |

Contents

[ASP.NET CORE 2](#_Toc97320711)

[CONTROLLER : 4](#_Toc97320712)

[Routing Conifgürasyonu 13](#_Toc97320713)

[VİEW 17](#_Toc97320714)

[MODEL 19](#_Toc97320715)

[HTTP GET VE POST METHODLAR 23](#_Toc97320716)

[VİEWİMPORT DOSYASI İLE ÇALIŞMAK 24](#_Toc97320717)

[LAYOUT VE VİEWSTART DOSYASI İLE ÇALIŞMAK 24](#_Toc97320718)

[WWWROOT 26](#_Toc97320719)

[SECTİON MANTIĞI İLE ÇALIŞMAK 27](#_Toc97320720)

[CONVERTIONAL ROUTING 28](#_Toc97320721)

[SESSİON 29](#_Toc97320722)

[KATMANLI MİMARİ İLE İŞ SÜREÇLERİNİN ANLAŞILMASI 30](#_Toc97320723)

# ASP.NET CORE

Kurumsal Yapıları destekleyen ve açık kaynak kodlu platform olan bir web geliştirme aracıdır.

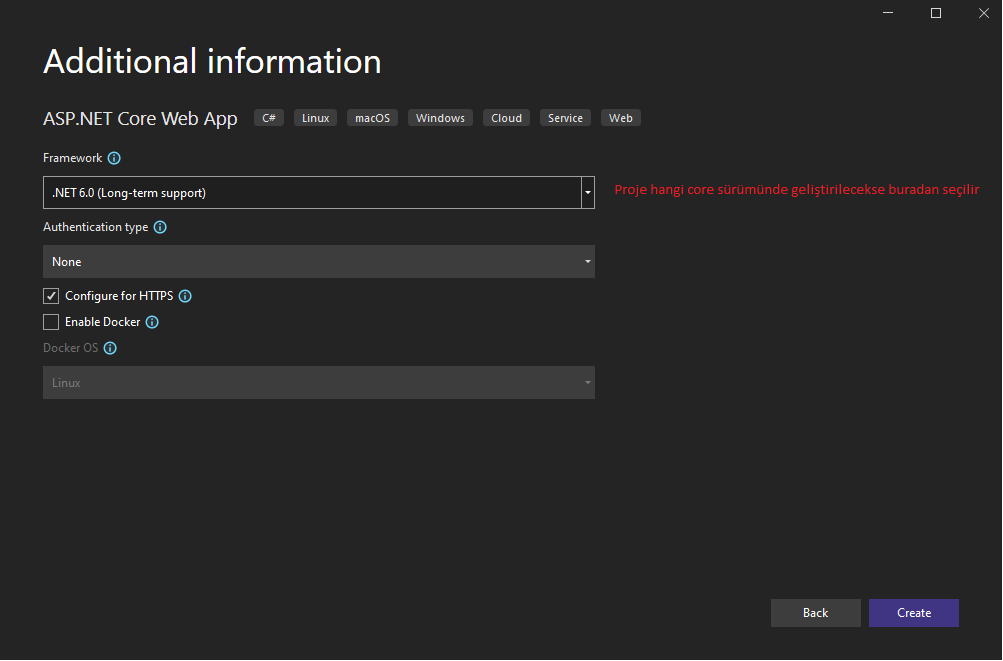
Text

Description automatically generated

MVC şablonu ile oluşturmak istenirse halihazırda hazırlanmış ve klasörlenmiş bir mvc şablonu ile karşılaşacağızdır. Bu sebeple işaretlemiş olduğum u değil parantezli bir şekilde (Model- View – Template ) yazılı şablon ile iş yapmamız gerekecektir.

Graphical user interface, text, application

Description automatically generated



Text

Description automatically generated

Yukarıdaki şekil hazır MVC şablonu ile oluşturulmuş olan bir **ASP.NET CORE** projesidir.

MVC mimarisi birçok frameworkte bulunmakla birlikte kısaca bahsetmemiz gerekirse

Client tarafından bir istek (request) geldiğinde bizleri ilk karşılayacak yapının ismi controller yapısıdır. Domain ve routing yapısının çözümlenmesinde ve bu çözümlenmeye karşı gelecek olan sonuçların üretilmesi için kurulmuş bir mekanizmadır. Dış isteğe (request) karşılık veren (response) ilk noktadır. Controller bir sonraki aşamada View üzerinden bir görüntü düzenler ve bizlere getirmektedir. Bir yandanda controller için hangi view (sayfayı) açacağını belirleyen bir yapıdır. Içerisinde HTML, CSS, JAVASCRİPT I barındıran yerdir. View üzerindeki model yapılarının oluşturulabilmesi için model katmanı gerekmektedir. Verilerimiz üzerinde barındırmaktadır. Daha açıklayıcı bir ifade ile controller dan view e gönderilen verilere model denmektedir.

## CONTROLLER :

Text, website

Description automatically generated with medium confidence

Hazır bir MVC şablonu seçildiği zaman HomeController isimli bir adet controller sınıfı oluşmaktadır.

Text

Description automatically generated

Varsayılan olarak içerisinde 3 adet birbirlerinden farklı fonkisyon ile gelmektedir. IActionResult bir aksiyon sonucu sonuç döndürmek amacı ile kullanılan bir fonkisyon tipidir (Gelen veriye göre emptyresult json veya view gibi içerisindeki çeşitli yapılardan veriler döndürebilen bir interface yapısıdır). Genellikle bir view döndürmektedir.

Burada normal c# tan bilindiği üzere çeşitli fonksiyonlar kullanılabilmektedir. URL e göre gelen controller ve fonkisyon ismine göre gerekli fonkisyon tetiklenerek bir çıktı üretilmektedir. String int float başta olmak üzere gibi çeşitli veri tipleri dönüşü ve buna uygun fonksiyon yazılabilmektedir. Hazır bir MVC şabonu içerisinde 3 farklı method gözükmektedir. Hepsinin IActionResult döndürmesi ile hepsinin bir web sayfası döndüreceği belirlenmiştir. Sırası ile fonksiyonları inceleyelim

Graphical user interface, text, application, email

Description automatically generated

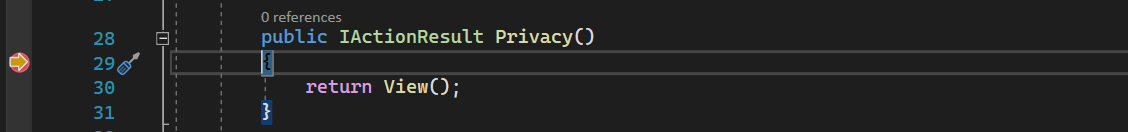
Görülen sayfa HomeController Altındaki İndex fonksiyonunun tetiklenmesi ile oluşturulmuş sayfadır. Controller varsayılan olarak HomeController ve altındaki Index fonkisyonuna bağlı olduğundan ötürü ilk etapta burası karşımıza gelmiştir.

Error ve privacy fonksiyonlarının hangi sayfaları getirdiğine bakalım

Graphical user interface, application

Description automatically generated

Görüldüğü üzere homecontroller altındaki privacy url ine gidildiğinde



Bizleri bu fonksiyon karşıladı ve bir sonraki aşamada bir view döndürdü.

Graphical user interface, text, application, email

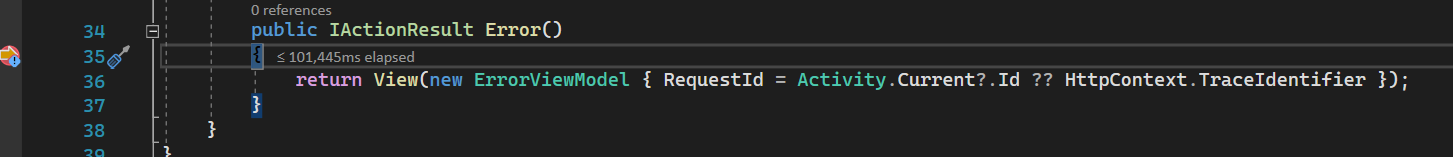
Description automatically generated

Görüldüğü üzere privacy indexi bizlere bu indexi getirdi. Birde error sayfasına gitmeye çalışalım …

Graphical user interface, text, application, email

Description automatically generated

Istek atılan url bizlere Error fonksiyonunu tetiklettirdi.



Graphical user interface, text, application, email

Description automatically generated

Görüldüğü üzere bizlere hata ekranı gelmiş bulundu.

Error fonkisyonunun içeriği gözüktüğü üzere biraz diğerlerinden farklıdır. Içerisinde zorunlu olarak bir model istemektedir. Bu model gönderilmeyip direk url döndürülmeye kalkıldığı zaman hata vermektedir. HTML dosyalarına Shared klasörü altından error.cshtml yolu ile ulaşılabilmektedir.

Text

Description automatically generated

Model dosyasının hata verilmemesi için gerekli yapı açıklama satırına alınarak bir istek daha gerçekleştriliyor.

Graphical user interface, text, application

Description automatically generated

Görüldüğü üzere bizler bu şekilde bir sayda döndürüldü. View e error id yollanması zorunlu kılındığı için bu şekilde kapatarak ta çalıştırabildik. Bu aşamaya kadar var olan controller ile işlerimi yaptık. Şimdi kendi controllerimizi yazalım

Controller içerisinde add -> controller… yı seçerek kendi controllerimize isim vererek oluşturuyoruz.

A picture containing graphical user interface

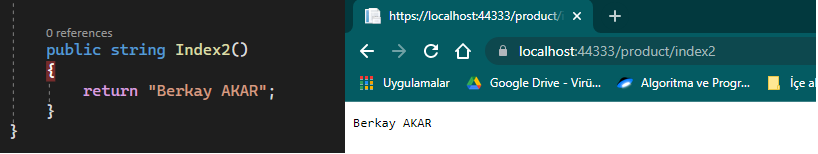
Description automatically generated

Controller verilirken hangi işlem için bir control oluşturacaksak onun ismi ve controller diye yazılması gerekmektedir. Özünde burada bu işlemde bir hata meydana geliyorsa ProductController isimli bir class oluşturup Controller isimli bir class ile kalıtım verdirerek te yapılabilmektedir bu işlem.

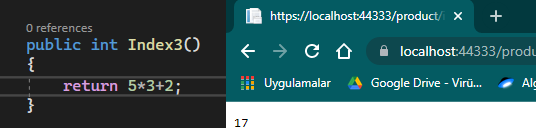
Graphical user interface, text

Description automatically generated

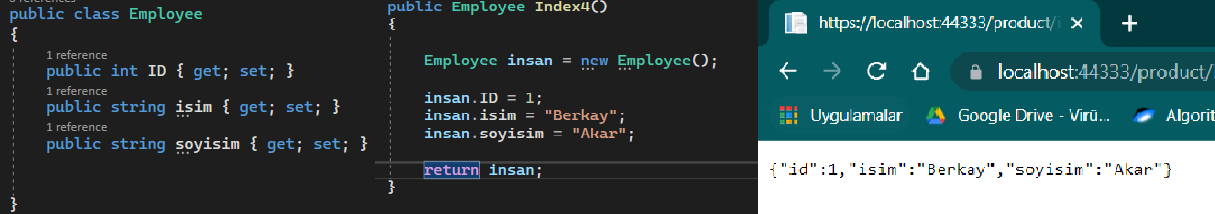
IActionResult bizlere burada bir view döndürmektedir içerisinde. Bu sebepten ötürü fonksiyon tiplerinde denemeler yaparak sonuçlarını inceleyelim…



String tipinde bir dönüş gerçekleşti.



Int tipinde bir dönüş gerçekleşti



Sınıf tipinde bir dönüş

Text

Description automatically generated Graphical user interface, text, website

Description automatically generated

Generic bir liste tipinde dönüş …

Görüldüğü üzere fonksiyon tipimiz neyse ona göre bizlere bir tip döndürmektedir. Şimdi IActionResult Fonksiyonunun yapısını incelemeye başlayalım.

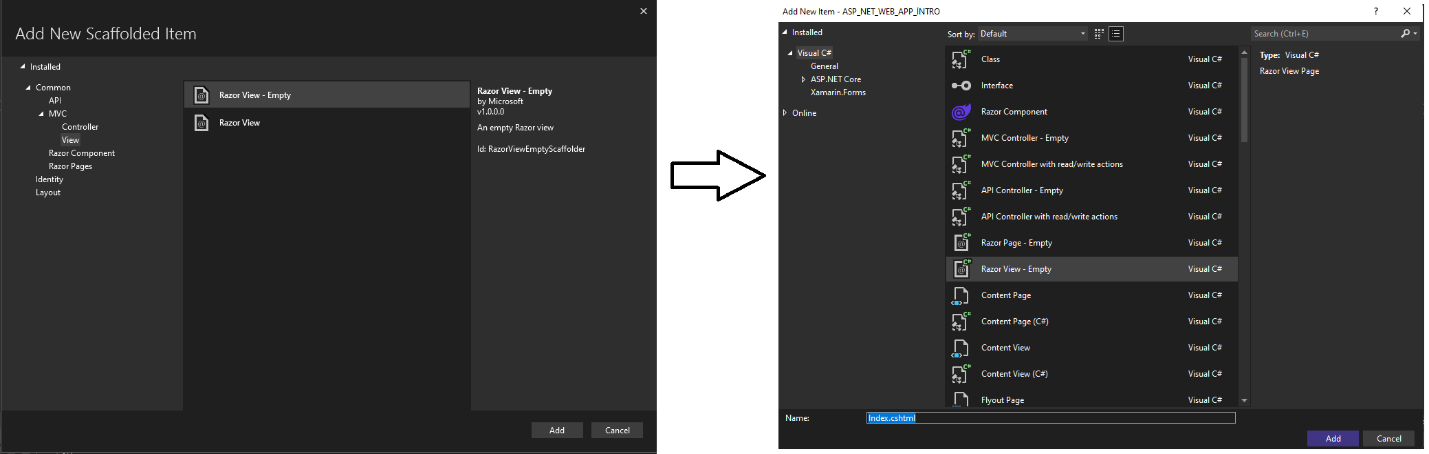
Daha öncede Belirttiğimiz gibi IActionResult bizlere json emptyresult ve view Döndüren bir fonksiyondur. Şimdilik view döndürmesi baz alınarak anlatıma devam edilecektir. Html dosyalarını cs uzantılı olacak şekilde views klasörü altında görebilmekteyiz.

Graphical user interface, text, application

Description automatically generated

Görüldüğü üzere her controllerin viewi için bir klasötü mevcuttur.

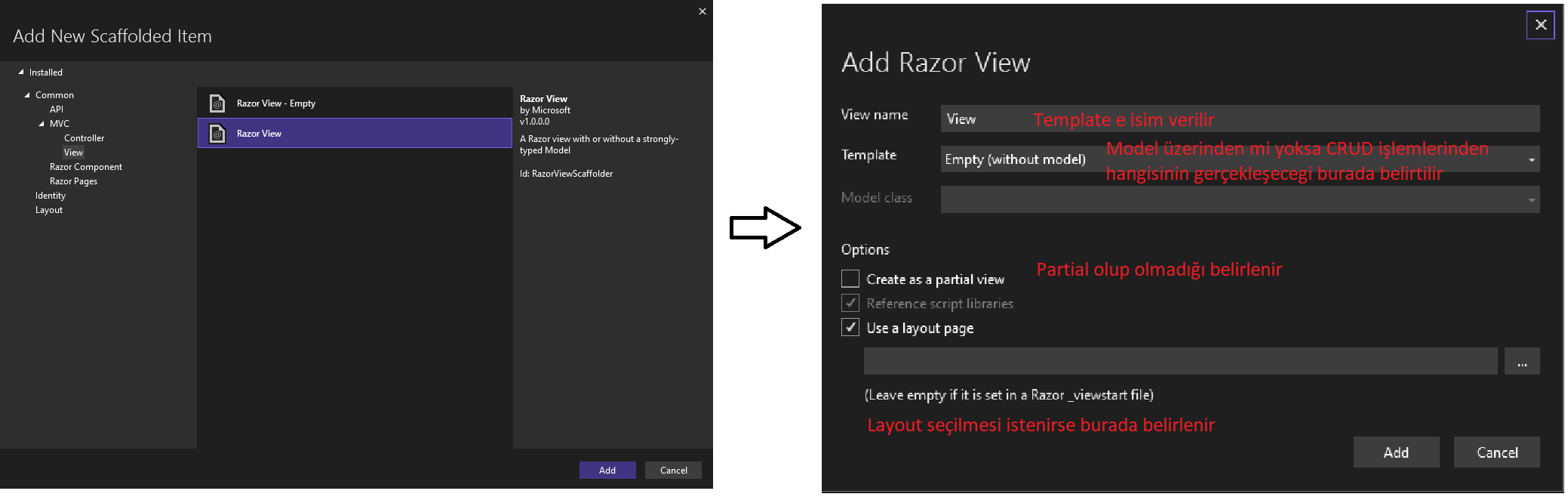
Bizde Buraya hangi controller için view ler oluşturacaksak burada belirtmemiz uygun olacaktır. Controllerimizin ismi ile view de klasörümüzü oluşturduktan sonra hangi fonksiyonumuz hangi sayfayı tetikleyecekse onun ismini kullanarak viewler oluşturmamız gerekecektir.



Graphical user interface, text, application

Description automatically generated

Ben burada empty bir şablon oluşturduğumdan ötürü bu şekilde bir yapılandırma gelmiş bulunmaktadır. Boş bir şablon seçmemek (içerisi dolu bir şablon için) oluşturulan view altındaki controllerismi ile verilmiş klasörün üzerine gelinerek sag tık -> new -> view seçilir ve sonrasında aşağıdaki adımlar takip edilir.



Yukarıdaki yapılandırmada fonkisyonumuzun ismi ney ise onu birebir aynı yazarak veriyoruz.

A screenshot of a computer

Description automatically generated

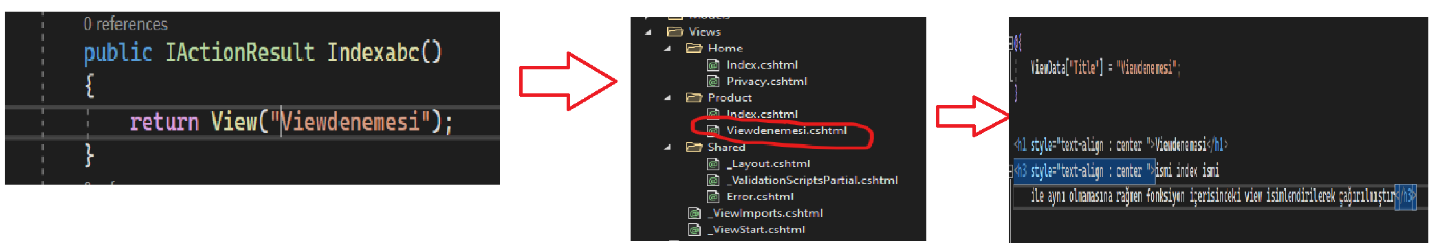
Gerekli eklemeleri yaptıktan sonra bir çalıştıralım.

Graphical user interface, text, application, email

Description automatically generated

Görüldüğü üzere ilk view imizi başarı ile ekleniş bulunduk.

Bir view çağırılırken fonksiyon ismi ile sadece aynı isimli bir view oluşturduğumuzda klasör ve view ismine bakarak IActionResult otomatik olarak bu view i tanıyarak çalıştırmaktadır. View ismi fonksiyondan farklı bir isim olarakta verilebilirdi. Bu işlem için indexabc ve viewdenemesi isimli bir fonksiyon ve bir view oluşturuldu.

fonksiyon içindeki return view içerisine döndürülecek view ismi yazılarak bu fonksiyonun hangi view I döndürebileceğini bu şekilde belirleyebiliriz.

Graphical user interface, text, application

Description automatically generated

Görüldüğü gibi bu şekilde bir tetikleme de gerçekleştirilebilir. Bir fonkisyon tetiklendiğinde view içerisinde bir değer belirtmediğimiz sürece otomatik olarak kendi isminde views klasörü içerisinde bir view arar ve bulursa onu çalıştırır. Ama bu şekilde de biz view isimlerini manüel bir şekilde berirterek hangi view I getireceğinide belirleyebiliriz.

## Routing Conifgürasyonu

Startup dosyası içerisinde useendpoint noktaları içerisinde pattern belirlenebilir. Bu pattern ile url yapısı konfigüre edilebilmektedir.

Text

Description automatically generated

= işareti ile hiçbir şey yazılmaması durumunda ve action kısmının yazılmamamsı durumunda varsayılan olarak bir controller ve bir indexin belrilenmesi amacı ile oluşturulmuş bir yapıdır.

Graphical user interface, text, application

Description automatically generated

Görüldüğü üzere hiçbir şey yazılmadığı durumda proje direk olarak HomeController içerisindeki İndex isimli iactionresult a gönderecektir.

Deneme amaçlı olarak = işaretlerini silerek bir işlem yapalım bakalım web sayfasını otomatik olarak getirecekmi ?

Text

Description automatically generated

Şimdi Çalıştıralım.

Graphical user interface, text, application, email

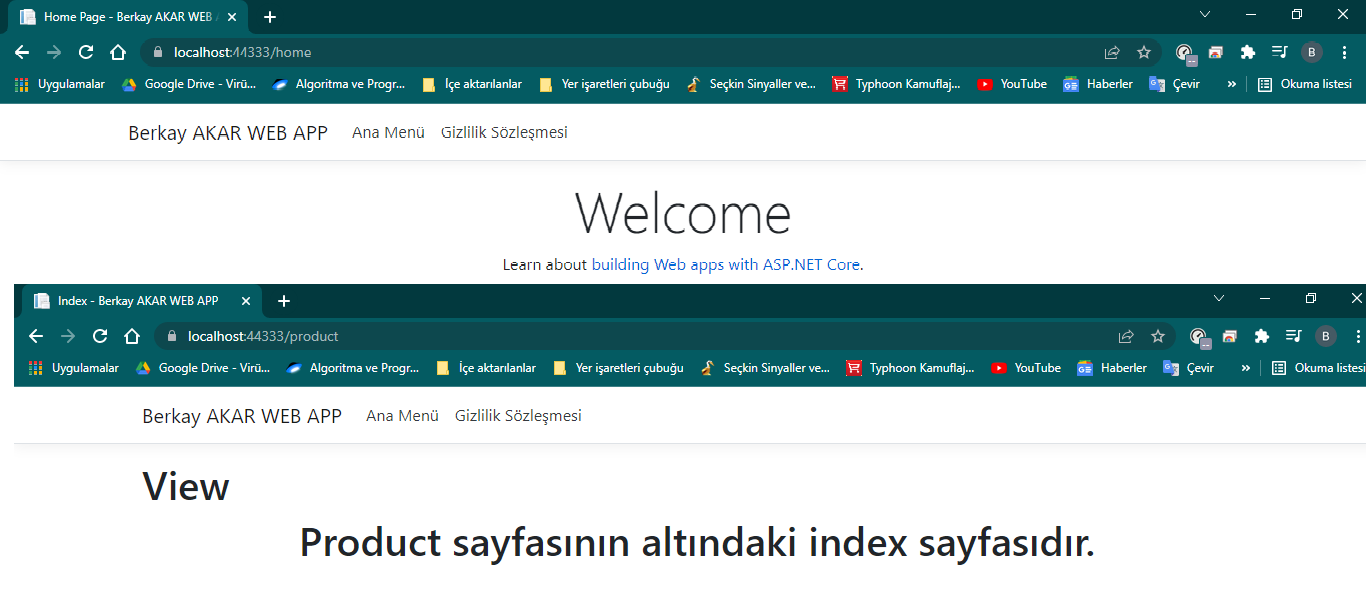
Description automatically generated

Görüldüğü gibi bir hata mesajı ile karşılaşıldı. Isteğe bağlı olarak istenilen controller ismi burada verilerek iş yapılabilmektedir.

Text

Description automatically generated

Bu şekilde bir kod yazdım. Burada controller ismi ilk başta verilmediği için system başlatırken bir hata verecektir. Ama controller ismi ile başlatırsam hangi controller ismini verirsem onun içindeki index methodunu çağıracaktır.



Görüldüğü gibi diren controller ismi vermem yeterli kalmaktadır. Home verirsem home controller, product verirsem product controller e götürmektedir.

Id kısmında ? işareti ile belirtilmesinin sebebi fonksiyonlarımızın bir post işlemi esnasında (form üzerinden veri gönderme) veya fonksiyonlarımız için bir parameter gerekliligi olma durumunda dışarıdan bir parameter alabilmemizi sağlamaktadır.

Graphical user interface, text

Description automatically generated

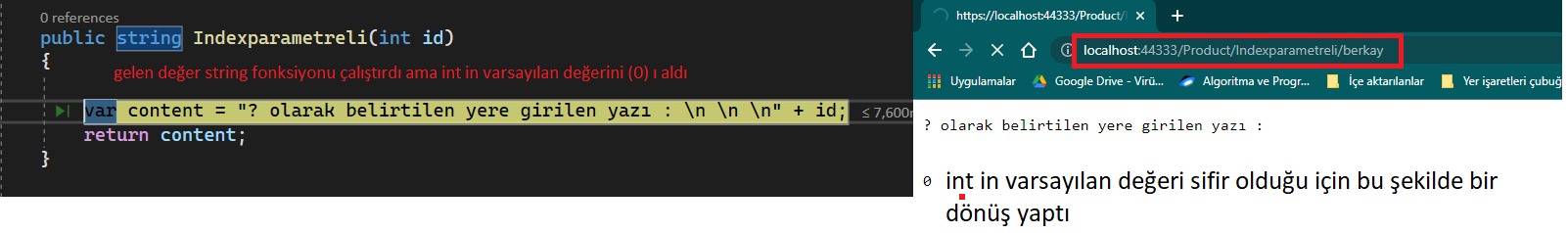
String olarak bir fonkiyon oluşturduk ve buraya dışarıdan bir string ifade geleceğini belirterek bir parametre belirttik. Bu fonkisyonu biz product controller içerisinde belirtmiş bulunmaktayız. Şimdi fonkisiyonumuzu çağıralım ve içerisine parameter atayarak değerini analiz edelim.

!!! ÖNEMLİ NOT (Startup.CS içerisindeki rouing dinamigini hatırlamamız gerekirse controller / action / id ) şeklinde gitmektedir. Bu sebepten ötürü string yazı olarak yaptıgımız değişken ismi dinamiğin çalışabilmesi için fonkisyona özel olarak route dinamiği yazmamız gerekmektedir.

A screenshot of a computer

Description automatically generated with medium confidence

Eğer direk int id şeklinde dışarıdan gelen veriyi kullanmak isteseydik route dinamiğine ihtiyacımız olmayacaktı. Dışarıdan gelen id nin tipi önemli değildir. Sayısal ve alphanumeric bir değişken tipi de olsa system ismini aynı olarak algıladığı için işlemine devam edecektir. Bu sebepten ötürü farklı parameter isimleri kullanılmak istenirse işlemler bu şekilde gerçekleştirilecektir. Unutulmammlıdır ki routing içerisindeki son indisteki parameter ismi fonksiyondaki ile aynı olmalıdır yoksa dışarıdan gelen değeri algılamamaktadır.



Görüldüğü üzere dışarıdan gelen deger string olduğu ve bizimde fonksiyonumuz int olduğu için değeri alamadı. Fakat fonksiyonumuzu çalıştırdı ve bize 0 değerini döndürdü.

## VİEW

İçerisinde Web sayfalarını barındıran bir yapı olduğundan bahsetmiştik. Yüzeysel olarak bir klasör düzenine ve kod örneğine bakalım.

A screenshot of a computer

Description automatically generated with medium confidence

Her bir controller için bir klasör olması gerektiğinden (zorunluluk değil ama olması daha olduğundan) bahsetmiştik. Shared yapısı ise içerisinde layout mekanizmasının bulunduğu bir yapıdır.

A screenshot of a computer

Description automatically generated with medium confidence

Gözüktüğü üzere içerisinde html kodları ve blazor yapısı bulunmaktadır. @{ } arasındaki kodlar razor kod blogu olarak adlandırılamtadır. HTML kısmındaki çeşitli yapılandırmaları burada sağlayabilmekteyiz. Model katmanından gelen veriler, layout yapılandırması, html üzerinde çeşitli senaryolara göre C# kodlarının yazılabilmesi, (Döngülerin kullanılabilmesi, şart bloklarının kullanımı vs.) html taglarına müdahale başta olmak üzere çeşitli yapılandırmalar ve sınırlamalar burada yapılabilmektedir. Djangodaki Jinja yapısı ile benzerlik te göstermektedir.

**A screenshot of a computer

Description automatically generated**

Görüldüğü üzere blazor alanına ulaşmak için @ile başlayarak içerisindeki nesnelere tek tek ulaşılabilmekte ve hatta üzerinde fonksiyonlar yazarak çalıştırabilmemizi sağlamaktadır. Bu kodların ekran çıktısı bir aşağıda verilmiştir.

Graphical user interface, text, application, email

Description automatically generated

## MODEL

IActionResult kısmında viewin içerisinde veritabanından çekilen verilerin model olarak aktarılması işine yaramaktadır. Yani biz burda dönen veriler üzerinde bir crud operasyonu gerçekleştirebilmekle birlikte veri tabanından dönen verileri izole edebilme imkanımız mevcuttur.

A screenshot of a computer

Description automatically generated with medium confidence

Modellerin bu şekilde oluşturulduğunu varsayalım

İlk aşamada view kısmında yapılması gereken ise model dosyamızı tanımlamamız olacaktır.

Text

Description automatically generated

Tanımlaması yapıldıktan sonraki aşamada uygun view I yazalım.

Text

Description automatically generated

View içerisinde parameter ile oluşturulan model yollanabilmektedir.

Oluşturulan modeler liste , sınıf , IQERYABLE gibi çeşitli tipler ile yollanabilmektedir. Yollanan verinin Views te karşılanması gerekmektedir. Karşılama esasında ilk öncelik gelen modelin tipine göre bir deklerasyon ile belirlemektir.

Text

Description automatically generated

Modelimiz liste tipinde gönderildiğinden ötürü @model list<classismi> şeklinde @model değişkenine atayarak bir karşılık oluşturabiliriz.

Text

Description automatically generated

Görüldüğü üzere blazor kodlarını tek tek başında @ işareti koyarak belirlememiz gerekmektedir. Bu şekilde deklerasyonlar sağlayarak sayfalarımızı modelden dönen koda göre dinamik hale getirmiş bulunduk. Esasında NTier Mimarisi düşünüldüğünde ve ıaction result düşünüldüğünde bir web sayfasına en fazla 1 adet model gönderebilmekteyiz. Bu esas düşünüldüğünde bizler çeşitli ViewModeller kullanarak oluşturulan modelleri tek bir model dosyası içerisinde yollayabilir veya partialview kullanmamız gerekmektedir. Şimdi ben hem ürünleri hemde kullanıcıları gönderebileceğim bir entity daha oluşturarak modellerimi buraya çekecegim

Text

Description automatically generated

Görüldüğü üzere bir nesne içerisinde hangi tip veriler gelecek ise bu şekilde bir doldurma işlekmi yaparak devam ettik .

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Bu sefer yolladığımız veri bir sınıf nesnesi olduğu için içeriğinde model kısmında referansımızı değiştirdik. Beraberinde tek yolladığımız model in hangi listesini seçecek isek ona bir foreach atarak işe devam ettik.

## HTTP GET VE POST METHODLAR

Var olan bir verinin sorgulanması işlemine get methodlar denilmektedir. Bir istek atıldığında view de bizlere verilerin gözükmesi işlemine bizler get method (talepte bulunması sonucunda bir data getirmesi), kullanıcı tarafından alınan çeşitli imputları arka plana yollama işlerine (güncelleme slime kaydetme vs) de post methodlar denilmektedir. Bu zamana kadar bizler get methodlar ile uğraşmış bulunduk. Get methodların çalıştırılması için fonksiyon üstüne herhangi bir taghelper belirmemiz gerekmez. HTTPGET olarak belirtilmesi ise bizlere bir sorun teşkil etmez.

From işlemi varsayılan olarak httppost çalıştıracaktır.

## VİEWİMPORT DOSYASI İLE ÇALIŞMAK

Sıradan bir MVC projesinde görüldüğü üzere her controller için bir klasör , altlarında cshtml dosyaları, ve viewimport dosyası bulunmaktadır.

A screenshot of a computer

Description automatically generated with medium confidence

Viewimport klasörü tüm viewlerde tanımalanacak olan (ön tanımlı gelen) yapıları global olarak almaktadır. Bu sebepten ötürü. Tüm taglarda ortak olarak kullanılmasını planladığımız herşeyi içerisine dahil edebilmekteyiz.

A screenshot of a computer

Description automatically generated with medium confidence

## LAYOUT VE VİEWSTART DOSYASI İLE ÇALIŞMAK

Layout tan bahsetmemiz gerekirse bir web sayfası boyunca sabit olarak gelmesini istediğimiz yapılara denmektedir. Genellikle Navbar footer arama kısmı gibi yapılar layout üzerinden çalışmaktadır.

Text

Description automatically generated

Görüldüğü üzere blazor kodları içerisinde layout kavramı bulunmaktadır. Bu sayfanın herhangi bir layoutunun bulunmayacağından bahsedilmiştir. Bu yazı varsayılan bir projede kaldırılsa bile sistemde otomatik olarak default bir mvc layout u belirecektir. Bu sebeple her proje içerisinde tek tek layoutlarımızı ya elle vermeli yada null yapmamız gerekecektir.

Varsayılanda gelecek olan bu layout a müdahale edebilmemiz için viewstart kısmına gelerek burada manipülasyonlar yapmamız gerekecektir.

Graphical user interface, text

Description automatically generated

Aynı viewimports mantıgındaki gibi düşünebiliriz. Blazor sayfaları için tüm proje içerisinde default olarak import edilmiş dosyaları gördüğümüz gibi burada da viewstart içerisinde layoutumuzu belirleyebilmekteyiz. Kısacası bütün viewlerde çalışacak kodları belirlediğimiz bir yerdir.

## WWWROOT

A screenshot of a computer

Description automatically generated with medium confidence

## SECTİON MANTIĞI İLE ÇALIŞMAK

Kelime Anlamı : Bölüm manasına gelmektedir. Razor sayfalarında ortak bir sayfa çalışacaksa (Proje genelinde)

## CONVERTIONAL ROUTING

Startup Dosyasının içerisinde public void configure isimli fonksiyon Middleware dediğimiz yapının oluşturulduğu yerdir.

Routing mekanizmasıda bir önceki bahsettiğimiz yapıdaki süreçlere benzerlik göstererek burada detaylanmıştır.

{} yapısı içerisinde gelen yapının neye karşılık geleceği = ile de hiçbir değer gelmeemsi koşulunda otomatik olarak ney ile eşleşeceğini ve ? gelmesi koşulunda ise isteğe bağlı olarak eklenecek olan parametreden bahsetmiş bulunduk.

Text

Description automatically generated

Endpint içerisinde süslü parantez ile belirtilmeyen yapı sabit olan bir yapıya bedeldir.

## SESSİON

7:40

Web teknolojilerinde gelen istekleri belirli bir ön bellekte tutabilme işidir. Alışveriş sistemlerindeki sepet mantığı verilebilmektedir. Çeşitli sayfaları gezdikten sonra aynı oturum içerisinde verileri sepette tutma işidir.

Sessionların core uygulamalarında çeşitli tiplerde konfigürasyonları bulunmaktadır . Bazılarını şu şekilde yapılmaktadır.

Text

Description automatically generated

ConfigüreServices içerisinden ilk olarak projeye Session ların eklenmesi gereklidir.

2. Olarak ise Middleware olarak sisteme dahil edilmesi gerekmektedir.

Text

Description automatically generated

Her bir istek attıgımızda controller newlenerek iş yapar. Bu sebepten ötürü controller in newlenmesinin önüne geçebilmek için bir adet session açmamız gerekmektedir.

# KATMANLI MİMARİ İLE İŞ SÜREÇLERİNİN ANLAŞILMASI

9:15

Katmanlı mimari tanımından yüzeysel bir şekilde bahsetmek gerekirse iş yüklerinin farklı yapılara bölündüğü ve her katmanın kendisine özel işlerin tanımlandığı bir tasarım mimarisidir. Çok katmanlı mimariye MVC örnek verilebilmektedir. Veritabanında modelleri oluşturmak için Model, Web sayfalarının bulunduğu yerler View ve fonksiyonların oluşturulduğu Controller yapılarını düşünürsek MVC yapısı N-Tier mimarisine örnek olarak verilebilmektedir.

Tüm süreçlere yakından bakabilmemiz için bir proje üzerinden ilerlenmesi ve süreçlerin incelenmesi daha uygun olacaktır.

1. Projeye Başlangıç:

Katmanlı mimaride bir proje için ilk öncelik boş bir solution dosyası açmaktır.

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

Solution içerisinde birden çok proje barındıran ve tüm projeleri tek çatı altında toplamayı sağlayan yapıdır.

Graphical user interface, text, application, Teams

Description automatically generated

Sırası ile katmanları oluşturmaya geliyoruz. Classlibrary olarak oluşturulacak olan dosyalarımızı solution üzerine gelerek sağ tık, yeni proje diyerek belirliyoruz. (.NET Framework olmamasına dikkat edin).

Entity Layer : Varlıkları (Proje içerisindeki tabloları) tanımladığımız katmandır. Her tablo için bir adet sınıf açılır. Veritabanındaki nesnelere karşılık gelen nesneleri tanımlarız. DTO nesneleride tutabiliriz. (joinlenmiş tabloları DTO üzerinde tutarız.). Joinlenmiş tablolar içerisinde ViewModeller oluşturarak daha efektif bir şekilde kullanılabilirliği sağlayabiliriz.

DataAccess Layer: Veritabanı işlerini yaptıgımız yerdir. Temel sorguları yazdığımız yerlerdir.

Business Layer: Kelime manası olarak iş katmanı demektir. Projeye yönelik iş kuralları yazılmaktadır. DataAccess e erişim kararını veren yapıdır. Gelen datanın uygunluğunu control eder.

WebUI : MVC kodlarını yazdığımız yer burasıdır.

Core Katmanı : Core Katmanı bir Framework Katmanıdır. Hashleme kodu gibi tüm projede ortak kullanılacak olan kodlar buraya tanımlanır.

Graphical user interface, text, application

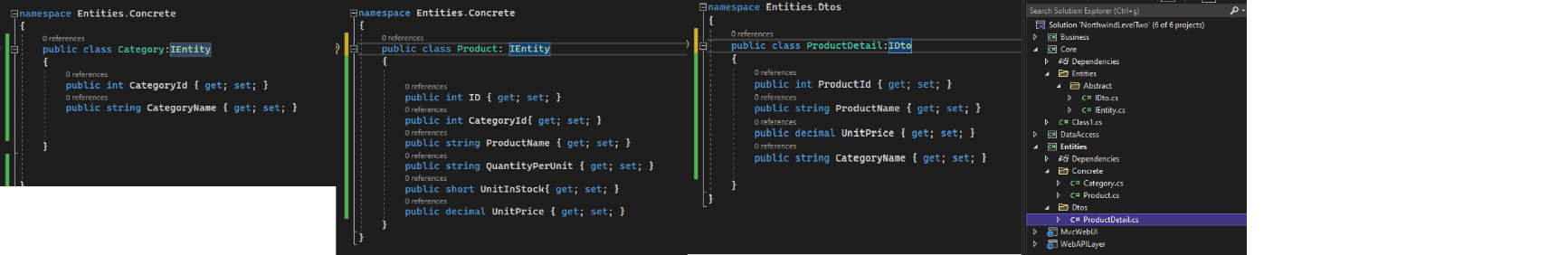
Description automatically generated

1. Katmanları Oluşturma Ve Projeyi Programlama:

2.1) Entity Layer :

Daha öncesinden de tanımlandığı gibi katmanları bu bölümde doldurmaya başlayacağız. Proje boyunca Nordwith veritabanı baz alınarak iş yapılacaktır. İlk işimiz DB ye nordwith veritabanını yüklemek ve sonrasında sınıfları oluşturmaya başlamak olacaktır.

Veriler Script olarak yüklenmesi ile birlikte modellemeye geçecegiz. Mimariler gereği Abstract ve Concrete yapılarını oluşturmamız gerekecektir. Classları sınırlandırmamız için İnterfaceler oluşturmalı ve bu interface leri classlara implamente etmemiz gerekmektedir. İnterfacelere implamente ederken İnterfaceleri Abstract classları ise Concrete isimli iki klasörde toplamamız gerekecetir.



Görüldüğü üzere her bir sınıf oluşturuldu fakat alışagelmişin haricinde interface leri Core katmanında tanımlamış olduk. İsteğe bağlı bir süreçtir. Tüm projede ortak olacağı ve ilerleyen aşamalarda apilerinde ihtiyacı olacağından ötürü bu katmanda tanımlamayayı daha uygun gördük.

2.2) Business Layer:

Bu aşamada projeye nugetpackage üzerinden EntityFrameworkCore indirilerek ve Context yapısı oluşturularak devam edilecektir. Repositroy Design Pattern denen yapının kullanılabilmesi amacı ile Abstract ve Concrete yapıları bu aşamada oluştuırudlu. Sebebi Depency İnjection vasıtası ile ilerleyen yapılarda orm süreçlerinin farklı veritabanlarında da idame ettirilmesi istenmesinden ötürü gelmektedir. Yüzeysle bir şekilde bahsetmek gerekirse ORM yapıları birden çok veritabanı Yönetim sistemi ile senktronize çalışabilecek şekilde üretilmiştir. Yarın öbürgün bu veritabanlarının değiştirlmesinin istenmesi üzerine sadece Depency İncejtion kullanılarak tüm sürecin daha Rahat yönetilebilmesi için bu mimariyi kullanmak gerekmektedir.

Aynı zamanda EntityFramework gibi bir ORM yapısı kullanılacagından ötürü RepositoryDesingPAttern kullanılarak İş süreçleri dahada kısalacaktır.