**MVC ve ASP.NET CORE**

**# .Net ve Asp.Net ve .Net.Framework ve .Net.Core nedir?**

**.Net :** Microsoftundevoloperlar için geliştirdiği genel yapıya denebilir (Desktop da winform , Web de asp.net , cloud da azure gibi)

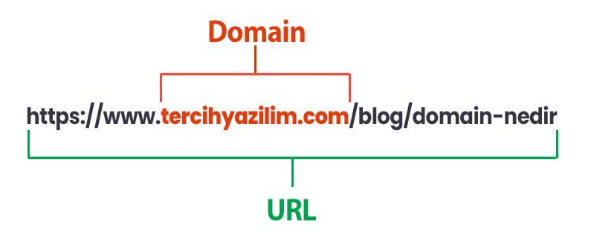
**Asp.Net**: ise .net yapısının web durumunda kullanılması için oluşturulmuş mimaridir Asp.Net mimarisi

**.Net FrameWork**: ise Winform Asp.Net gibi yapılarda yanlızca windows ekosistemine yönelik çözümler

**.Net Core:** Çeşitli işletim sistemlerini destler ve modüler olarak kullanılabilir

.net 5 den sonra tekrardan tek çatı altına girerek devam edilmiştir buradaki duruma .

**# Domain,Url, Hosting kavramları nelerdir ?**

**Domain :** Kişilerin web sitelerini  ziyaret etmek için tarayıcı url kısmına yazdıkları [web site](https://www.tercihyazilim.com/blog/web-site-nedir) adresleridir.

**Hosting** : Domain kısmında olan her türlü verinin sunucu ya da server gibi cihazlarda barınma durumunda kullanılan terim şeklinde ifade edile bilir ayrıca hostinglere ait depolanan web uygulamalarında ip le rile karıştırılmadan çalışma sağlanmaktadır

**Url:** Internet tarayıcısına yazılan adresdeki gidilmesi gereken yerin tamamaıdır

**# Http nedir ?**

Http yapısı bir protocol olup request ve response arasında client-server arasında ilişkiyi sağlayan model dir .Aradaki ilişki Get, Head, Post,Put,Delete,Trace,Options,Connect,Patch şeklinde olmaktadır.

**Get**: Sunucudan verileri listelemek için kullanılır

**Post**: Sunucuya verileri göndermek için kullanılır

**Put**: Var olan bir veriyi değiştirmek istediğinde kullanılmaktadır

**Patch**: Bir verinin bütün kısmı olmayıp bir kısmının güncellenmesi sağlanıyorsa kullanılır .

**Delete** : Var olan veriyi silmek için kullanılır

**# Server Nedir ? Hangi türleri bulunmaktadır?**

Web Uygulamaları yapılandırma ve geliştirmeleri bittikden sonra , online olarak diğer userların başvurabileceği temel yapılandırmalar olarak görülebilir , bunların bazı türleri şu şekildedir ;

**IIS**: Internet Information Services (IIS), esnek güvenli ve configure edilebilir bir web sunucusu olup asp.net dahil bir çok web uygulamasında kullanılmaktadır.

**Kestral** : Asp.Net Core uygulamalarında dahili olarak gelen bir web sunucusudur. IIS olmadan herhangi bir Microsoft sunucusu olmadığından Linux ya da farklı tabanlı sunucularda çalışabilmektedir

**Apache ve Nginix** : Linux gibi ortamlarda Asp.net Core uygulamalrında kestrel yapısı ile senkronize haberleşe bilen yapılardır

**Docker** : Yeni nesil sanallaştırma platformu olup , microservice yapılandırmalarında elimizdeki birden fazla servis tarafını ayağa kaldırmakda ciddi kullanılmaktadır .

**# MVC yapısı ve terimleri ile ilgili kısa açıklama yapabilir misiniz ?**

**MVC** yapısı bir framework yaklaşımı olduğu gibi aslında bir tasarım deseni olarak da kullanılabilmektedir

**Model** : Projenin iş mantığının tasarlandığı bölümdür, veritabanı modelleri başta olmak üzere görüntü verebilecek ve işlem yapılabilinecek modelleri burada tanımlarsınız. Bu modelleri projenizde size fazlasıyla yardımı olurlar. Doğrulama (Validation) işlemleri bu bölümde yapılabilinir.(Entity Framework , Entity Models , Ado.Net, Repository)

**View** : Projenin arayüzü veya kullanıcı arabimi de denilebilir (User Interface – UI), Bu bölümde HTML dosyaları üzerinde işlemler gerçekleştirilir.

**Control** : Projenin akışını, Model ile View arasındaki etkileşimleri kontrol eder. View dan gelen istekler Model yarımıyla yorumlanıp ne yapılacağı konusunda değerlendirilir. Fonksiyonel işlemlerin yapıldığı katmandır.Gelen isteklerin karşılanması durumuna bakıcaktır, verilerin görselleştirip , görselleştirmeme konusunda da controller katmanı aslında görevlidir

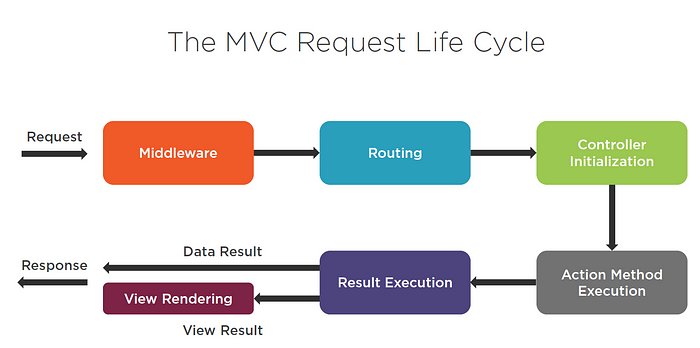
**Razor**: Asp.Net Mvc 3 teknolojisiyle birlikte gelen görüntüleme biçimidir

**ViewBag**: Verilerimizi Controllerdan View kısmına aktarmak için kullanırız.Runtime içerisinde oluşan dinamik bir nesnedir.

**ViewData**: Verilerimizi Controller dan View kısmına aktarmak için kullanırız.

**PartialView**: Sayfaların istediğimiz kısmına rahatça ekleyebileceğimiz, özel işlevleri yerine getirebilen, ancak kendi başına çalışmayan, bir sayfanın içerisinde entegre halde çalışan kısımlardır

# **MVC Pipeline asp.net core için nasıl çalışmaktadır** ?



# **MVC mimarisini çalıştırmada yapılıcak adımlar nelerdir?**

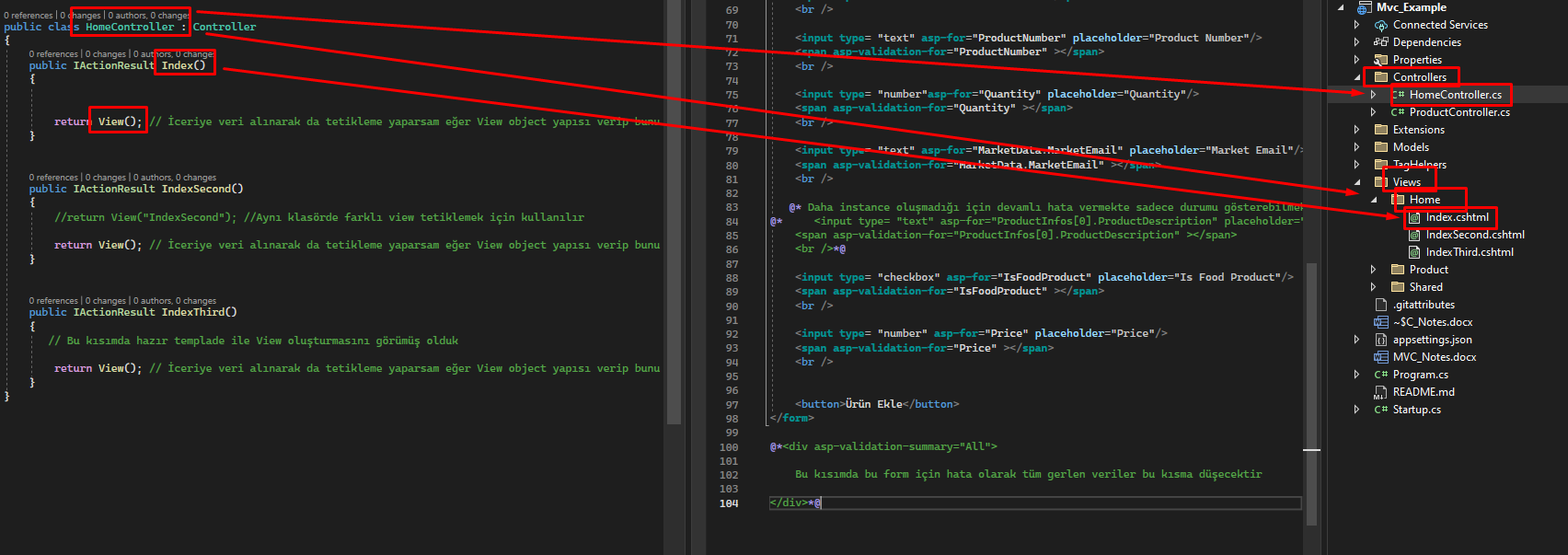
Asp.net core uygulamasında MVC mimarisini kullanmak için , ilk olarak startup tarafında ConfigureServices içinde addControllers(Sadece Controller kullanınca) ya da addControllersWithViews(Cshtml yapılı views tabanlı işlemlere yer verilicekse) şeklinde yapılandırma ile kullanım belirtilir

Configure tarafında ise gelen isteğin rota route yapısı UseRouting() middleware işlemi sayesinde belirlenir. Gelen isteğin end pointine(varış noktası yani url tarafına) UseEndPoints(), hangi rota ve şablonla gideleceğinin kararını veren yapılandırmadır.

Verilerdeki tetikleme konusuna gelirsek eğer buradaki yapı genelde UseEndPoints içinde middleware larda defaultda {controller=Home}/{action=Index}/{id?} bu şekilde tanımlanmıştır, yapı gereği farklı controller ve action lar oluşturulsa da default olarak system hiçbir parameter almassa home controller daki index actionana gidip işlem yapıcaktır .

Controller sınıfları controller olan classdan türetilmiştir(Özellikle View yapılı işlemlerin kullanılması için) bu kısımda Controller Base sınıfından türetilmiştir(Bu kısımdaki classda ilgili Request bilgileri model state , url ile ilgili bilgileri ve response bilgilerini getiren yapıdır) .

View dediğimiz yapı ise cshtml kullanan dosyalar olup front tarafın yanında razor gibi yapılarıda sağlıklı şekilde destekler. Asp.net core tarafına özel view tarafında c# kodlarımızı çalıştırmamızı sağlayan yapıdır.



Üsteki kısımda anlatmaya çalışıldığı gibi Controllera ait bir Action kullanılıyor ve bu View ile metot bazlı render edilmeli ise View ile bunun için bir Views folderına ihtiyacımız kesinlikle olup bunun içinde controller bazda folder yapılmış ve action bazında da cshtml I oluşturularak devam edilir en basit halde . return View içinde obje döndürerek bu kısımdaki objeyi razor ile front tarafında kullanılabilecktir

Return View içinde veri olmassa action ismindeki cshtml I tetikliyicektir fakat başka isimde bir dosya varsa içinde (mesaka Soykan.cshtml olsun) return view(“soykan”) şeklinde yazara onuda tetiklemesi yapılabilir

**# Controller içindeki action type kısımları nelerdir ?**

**ViewResult** = View render etmek sırasında ilgili gelen result ı render etmek için, client tarafına gönderilirken kullanılır.

public ViewResult GetProducts()

{ViewResult result = View();return result;}

**PartielViewResult** = Genelde view gibi olmayıp ajax yapılı isteklerde parça olarak render edilmesi gereken bir kısım olursa bu şekil tercih edilir. Teknik olarakda View Result \_ViewStar.cshtml baz alırken PartielViewResult ilgili dosyayı baz almadan render der

public PartialViewResult GetProducts()

{PartialViewResult result = PartialView();return result;}

**JsonResult** = Client tarafına Json türüne döndürülüp üretilen datadır

public JsonResult GetProducts()

{JsonResult result = Json(new Product{Id = 1,ProductName = "Computer",Quantity = 3});}

**EmptyResult** = Gelen isteklerde bir veri dönüdürlmesine gerek yoksa bu yapı kullanılır. Void keyword tarafı ilede yapılır

public EmptyResult GetProducts()

{ return new EmptyResult();}

**ContentResult** = Bu kısım metinsel ifade döndürmek için kullanılıp genellikle ajax json ve content verilerinde kullanılır

public ContentResult GetProducts()

{ ContentResult result = Content("Testing Testing");return result;}

**AcitonResult** = Genel olarak temelde kullanılan tip olup birden fazla türde döbdürme inkanın varsa bu kullanılır.

public ActionResult GetProducts()

{if (true){return Json(new object())}

else{return Content("Testing");}}

**IActionResult** = Polimorfizm tarafına uyması için kulllanılır.ActionResult tarafı ile aynı mantıkda kullanılmaktadır

public IActionResult GetProducts()

{ return View();}

**ViewCompnentResult** = ViewCompanent render edilmesini sağlayan perdormanslı bir yapsı vardır

**# Controller içinde non actiontype metotları nelerdir ?**

Data annotions tagı kullanarak yapılır controllerda olması tavsiye edilmez.

[NonAction]

public void X(){}

**# View yapısı ve view controller ilişkisi ve controllerdan view tarafına veri gönderme türleri nelerdir ?**

View ASP.NET framework ünde üretilen datanın görselleştirildiği, verinin render edildiği bir dosyadır. Cshtml tarafını sadece asp.net core tarafında kullanılmalıdır .View yapısı bir controllerdaki actionlara ait Views klasörü altında ana controller altında oluşturulan cshtml dosyasına denmektedir

Dört farklı controllerdan view tarafına gönderim bulunmaktadır.

**Modal bazlı veri gönderimi:** İki aşamadan olup view tarafında boxing yapılarak gönderim sağlanır ve sonrasında view tarafında bu nesne ile ilgi bilgi verilmelidir

Controller için : return View(product) (Controller tarafında view için obje aktarımı sağlandı)

View tarafı için : @model List<Mvc\_Example.Models.Product> (Türü bildirir) (View taraflı tanımlama sağlandı)

@Model (Büyük harfle yazılan ise ilgili datayı kullanmakiçin yapılan yapı)

**# Veri Taşıma kontrolleri:** Üç farklı yöntemi bulunmaktadır

**ViewBag** :View tarafına taşınacak olan datayı dynamic olarak tanımlanan bir değişkenle taşınmasını sağlamaktadır

Controller için : ViewBag.products = products (dynamic olarak Viewbag içine tanımlanması yapılmıştır)

View için : ViewBag.products as List<Mvc\_Example.Models.Product> (as olan kısım developda erişebilmek için yoksa runtime tarafında erişimi olucağından intelisense propertileri göstermicektir)

**ViewData** : ViewBag de olduğu gibi Action daki datayı view ve taşır (Boxing Edilerek taşır)

Controller için: ViewData[“products”] = products; şeklinde boxin yapılarak bulunacaktır.

View için : ViewData[“products”] as List<Mvc\_Example.Models.Product> Object şeklinde geldiği için unboxing yapılması lazım ondan as yapıldı

**TempData** : ViewData tarafında olduğu gibi boxing edilerek taşıma sağlanır , render edilmeden farklı bir action tarafına verinin taşınması için kullanılan yapıdır TempData temelde cookies yapısı kullanmaktadır ViewData ile View tarafı aynı şekilde gösterilicek fakat controller farklılıklar oluşabilir

Controller için : TempData[“products”] = JsonSerializer.Serialize(products); (Cookies tututğu için serilize ediyoruz , complex datalarda böyle bir hata vermekte yoksa )

return RedirectToAction("IndexRoute","Product") (Bu kısımda ise redirect yapılmakta)

TempData tarafında cookies taraflı bir ekleme olduğu için veriyi diğer actionlarda yönlendirebilir

**# Helpers Yapılanmaları nelerdir?**

Url ve Html Helper ASP.Net tarafında gelirken , TagHelper ASP.Net Core tarafında sonradan geliştirilmiştir.

**Url Helper** MVC uygulamalarında url oluşturmak için yardımcı metotlar ve url e dair bazı bilgileri barındıran metotdur

UrlHelper daki önemli bazı kısımlar ;

**Action:** Verilen controller ve actiona ait url oluşturur.Url.Action() şeklinde kullanılabilir,Url propertisi base den geldiği için farklı bir yapıya gerek kalmaz, system de yapılan kullanım ne ise ona göre bir yapı oluşturucaktır Action yapısı

**ActionLink:** Verilen controller ve actiona ait bir url oluşturmayı sağlar, actiondan farklı temel host bilgisini , protocol ve port bilgisinide beraberinde vermektedir

**Content:** Genellikle css ve script dosyalarının dizinlerini vermek için kullanlılır

**RouteUrl:** Mimaride tanımlı olan route isilerine uygun bir şekilde url oluşturan bir metotdur

**ActionContext:**Gelen request tarafına ait bilgiler veren bir property tarafında bulunmaktadır , bir çok bilgiye yer vermekte olup kullanışlıdır

**Html Helper** Html etiketlerini server tabanlı oluşturmamaızı sağlayan yardımcı metotlar , o anki context e dair bilgiler edinmemizi sağlamakta cshtml tarafında render yapılmasını sağlamakta

Html Helper daki önemli bazı kısımlar;

**Html.Partial :** Hedef Viewi render etmemizi sağlayan bir fonksiyondur (Parçalı html yazabilmek için kullanılır) dönüt olarak string döner

**Html.RenderPartial :** Hedef Viewi render etmeyi sağlar partial tarafından farklı html.Partial tek satırda çaprılırken bu kısım için razor anlamında süslü parantez kullanarak çağırılmalıdır, void döndürür. Teknik olarak sayfanın text writer yani response tarafını stream tarafına yazdığı için Html.Partiala göre daha tercih edilir

**Html.ActionLink :** Url oluşturur oluşturduğu link <a> tagında oluşturarak devam edilmesini sağlıyıcaktır

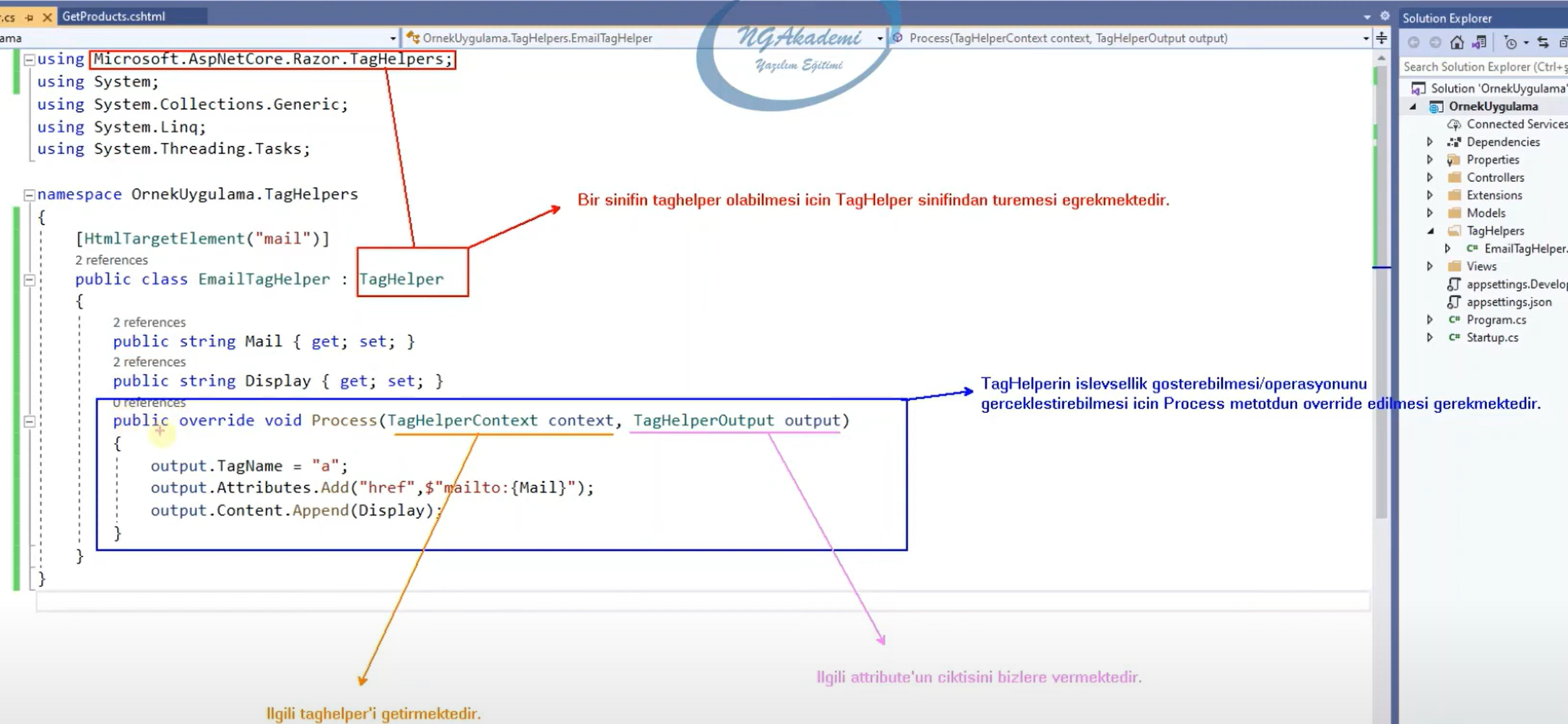
**HtmlForm Metotları :** Form input nesnelerinin oluşturulmasını sağlamaktadır, artık çok kullanılmıyordur.

**Custom Html Helper :** Customize edilecek devamlı Html Helper yapılanması varsa generic yapıda Html Helper Custom olarak kullanalıbilir.Html helper yapılandırması sonucu eklenen yapıda kullanışlı olara kullanılabilir

**Tag Helper** Html heperların yerine gelen yapılar olarak kullanılmaktadır, kod maliyetlerini oldukça düşürmektedir, cshtml tarafında kütüphaneyi buraya entegre etmek gerekmektedir @addTagHelper \* ,Microsoft.AspNetCore.Mvc.TagHelpers

Kendi içinde kulanıma sahiptir ve kullanımı yapılmaktadır, birden çok kullanımı vardır araştırılmalıdır

Custom TagHelper yapılması için takip edilicek durumlar aşağıdaki şekilde kullanılarak html helper gibi custom bir yapı oluşturulabilir.



**# Model Binding Mekanizması nedir?**

Request ile gelen verilerin ayrıştırılıp Action metotlarında dönüştürülmesi işlemine genel olarak verilen tanımlamadır.Genel olarak IActionResult tarafında model bazılı işlem yapma durumuna denmektedir

Bir Action fonsiyon aksini belirtmediğimiz süreçe get fonksiyon ile çalışmaktadır .

Eğerki action tarafında view için return View(); şeklinde gönderim yaparken içi boş şeklinde boş form oluşturucaktır , instance ise post yapıldığında oluşturup içeride gönderilicektir fakat eğerki yapılan işlemde return View(product); bu şekilde ise instance oluşturulup içindeki veriler ilgili view deki verilerle eşleşerek , yapılan değişiklik aynı instance ile controllera gönderilecektir .

**# Kullanıcıdan veri alma yöntemleri ?**

**Form Üzerinden Veri Alma :** Html taglari ile bir form tagı oluşturup buradaki yapıyı ilgili controller ve action kısmına bağlıyarak veriyi post etme durumudur. Html tag içindeki verilerdeki name propertisine göre ayrılıcaktır , controller kısmında ise bunu IActionResult VeriAl(IFormCollection datas) ilgili kütüphane formdaki değerleri getiricektir bunun yerine IActionResult VeriAl(string txtValu1,textValue2) şeklinde de yakalanabilir , üçüncü olarak da bu şekil yerine bunlara karşılık gelicek bir nesneye bağlı yapı tanımlarsak IActionResult VeriAl(Veri veriValue) şeklindede alınmasını sağlıcaktır .

**Form Üzerinden Bind Edilmiş Veri Alma :** İlgili view için @model şeklinde model tarafını belirlemek lazım , buna bağlı yapıda form tagi içinde hangisi ile eşleşiyorsa bu kısımlar için attribute ları oluşturarak systemin map edilmesi sağlanmaktadır

**Query String Üzerinden Veri Alma :** Güvenlik gerektirmeyen url üzerinde taşınan veriler için kullanılmaktadır **,** <http://...com/sehir?ilce=2>&b=5 gibi kullanımda kullanılan yapıdır.Requestin türü her ne olursa olsun Query string değerlerini taşıyabilirsin(Post get,put …) IActionResult VeriAl(string ilce, string b) şeklinde tanımlamada doğrudan Quey stringden ifadelerimizi aldık.Gelen veride içeride bu şekilde gönderilen verileri bu şekilde değişkene almadan HttpContext.Request.QueryString(Request yapılan ent pointe query string parametresi eklenmiş mi bunun ile ilgili bilgi verir) HttpContext.Request.Query ile metot içinde parameter almaksızında ulaşabiliriz, doğrudan bir model tasarlanıp buradan gelecen verilere actionResult tarafından alınabilicektir (IActionResult VeriAl(Product model))

**Route Parametreleri ile Veri Alma :** İlgili rotanın içinde değerleri gimerek gönderebiliyoruz bu durum da kullanılmaktadır , queryString tarafından farklı güvenliz anlamında biraz daha güvenlidir çünkü QueryString tarafında /user?name=max iken router için /user/max gibi bir kullanım olup max ın name ile anlamlı olduğunu bilemeyiz .

Route üzerinde veri taşıma için StartUp tarafında middleware de route yapılanması için system özelleştirilmeli default tarafında Id yi kullanabilmekteyiz . Başak bir yol olarak parameter olarak metoda gelmesede Request.RootValues ile bu verileri alabiliriz . Bir başka yöntem de gelen veriler bir tür ile karşılanabilinir

**Header Parametreleri ile veri alma :** Kullanıcının göndermiş olduğu http isteğinde bulunan veridir. Genellikle ilgili istek ile nitelikleri barındırır. Browser tarafında header tarafına müdahale etmek normalde olmadığı için postman tarafında örnekle çalıştırılabilir Headerlar genelde Request.Headers tarafından veriler yakılanır metot içinde.Headerlarda türkçe yada farklı dilde karakter kullanılmaz sadece latince kullanılır.

**Ajax ile Client tabanlı veri alma :** JQuery trafını ajax mimarisi kullanrak alıncak bir yapıdır , Kullanımı çok bilmeye gerek yok . J Query tarafı ilk javascript tarafına indirilmelidir metotda parameter olarak AjaxData parametresi olduğundan bu kısım ile halledilecektir

**# Gelen Bilgilerin Doğrulanması (Validation)**

Bir değerin istenen şartlaraq uygun olması durumudur.Amaca uygunluğunu control edicek durumdur. Web yazılımında iki türde yapılabilmektedir Server Side ve Client Side taraflı validation yapılmaktadır.İki taraftada güvenlik açısından uygulanmalıdır.Yapılabileck validationlar aşağıda şekilde kullanılabilir.

**Data Annatotation yapısı ile validation**

İlgili propertilere ait kurallar oluşturulduktan sonra buna ait controller kısımlarına gelen veride ModelState yapısı ile ilgili annotationslardaki duruma uygun validasyon yapılmış ya da yapılmamış mı bu konularda bilgi veren bir instance bize dönücektir .Genellikle burada yapılan validasyonlar entity nesneleri yerine modallar için oluşturulması istenmektedir. Fakat bu yapı Solid(S) yapısına aykırı bir system içermektedir .

**ModalMetaDataType yapısı ile Validation**

İlk kısımdaki yapı Solid tarafına aykırı olduğu için bu validationların başka classa alınarak yapılmasının sağlandığı bir yapıdır .Sadece validasyona uygun bir sınıf tasarlayarak solid prensibine uygun hale getiriyoruz bunu yapmak içinde annotations için sınıf kısmında [MOdelMetadataType(typeof(ProductMetadata))] şeklinde yaparak validasyon bildirmede yapmış oluyoruz

**Fluent Validation ile validation**

Hazır bir kütüphane olup validation yapılmasını sağlamaktadır.En çok kullanılanlardan biri olan bir validasyon kütüphanesidir ,kütüphane olduğu için nuget tarafından kütüphane yüklenip startup.cs tarafına configure service tarafına eklenmesini sağlayabiliriz . Models içinde Validators folder oluşturulması tercih edilir , ilgili sınıf içinse AbstractValidator sınıfından extend edilmesi gerekmektedir.

**Client tarafında dinamik olarak validation**

Server tarafında oluşan kodları bazı kütüphanelerle client tarafına taşıyabiliriz , böylece server tarafındaki maliyet azaltılmış olucaktır . Bu kısım için jquery taraflı kütüphaneler eklenilerek server taraflı yapı bire bir client tarafınada eklenebilir Bunu yapmak için Projeye Add diyip ClientSide Libraryy diğerek client yani ui kütüphaneleri ekleyebiliriz ,bu kütüphaneleri wwwroot denilen kısma ekleyerek devam edilmeli yoksa çalışmayacaktır.İlgili cshtml tarafına ilgili eklenen kütüphaneleri minify dosyalarını script bazında eklemek gerekli olucaktır

**# Layout yapılanması nedir ?**

Layout yapısı temel bazda oluşturulması istenen footer header gibi bazı temel ve genel olarak her zaman kullanılan yapılar layout denilen yapılar kullanılarak buradaki ciddi tekrar ve karmaşadan kurtarılması sağlanır , yapılardaki özellikle değişen view kısmının render edilesi sağlanarak ciddi performansda kazanıldı , yapılan değişiklikler tüm sayfalarda yapıldığı için bu gibi konulardaki sorun en aza indirilmesi sağlanacaktır .

Bu yapılandırmayı yapmak için bir layout dosyası yapılandırılmalıdır, genellikle Views in altında Shared klasöründe \_Layout.cshtml şeklinde oluşturulur(bub u şekilde oluşturulmak zorunda değil ama kültürde böyledir ) Ana kısım header footer navigation tarafları yazıldıktan sonra değişicek kısımdaki verinin olucağı yere **@RenderBody** yazılarak Layout yapılanması oluşturulmuş olmuştur. İlgili fonksiyon render edilen View in result ı nereye basılıcağını ifade eden fonksiyondur.Bir view yapısının, bir view tarafını layout olarak bilmesi için , Layout = “~/Views/Shared/Layout.cshtml” şeklinde belirtilmesi sağlanmıştır , @Renderbody tanımlaması sadece 1 tane olmlaıdır ve zorumludur

**@RenderSection** ise body anlamında olmayıp bazı view kısımları için farklılık olmasını sağlamak için kullanılır ana \_Layout içinde olup @RenderSection(“LeftMenu”,false) ikinci parameter kesinlikle her view da kullanılsın mı için oluşturulmuş kısımdır, view içinde bu kullanım @section LeftMenu{} olarak kullanılarak burası için layouttaki kısım özelleştirilir

**\_ViewStart** asıl amacı tüm view larda yapılması gereken ortak çalışmaların yapıldığı view dır.Ana view denilebilir .View klasörü altında \_ViewStart.cshtml olarak oluşturulması gerekir.Genelllikle view ların ortak kullanacağı Layout tanımlaması bu dosya içerisinde olmaktadır .

**\_ViewImport** razor sayfaları için kütüphane ve namespace tanımlamalarını sayfa sayfa farklı tanımlamaktansa ortak olarak yapabilmeyi sağlamaktadır Views klasörü altında \_ViewImports.cshtml şeklinde oluşturulmalıdr ViewImport yapısı @using LayoutExample.Models gibi bir yapılandırma durumlarında bir kerelik tanımlanarak diğer tüm viewlarda model oluşturuken hızlıca erişime açılmasını sağlıyacak yapı olara kullanılmaktadır

**# Modüler Tasarım Yapılanması Nedir ?**

PartialView ve ViewCompanent gibi kısımlarla sistem küçükten büyüğe doğru parça bütün ilişkisi oluşturularak sistemli hale getirilir.

**PartialView:** Modüler tasarım için her bir modülün parça bir view olarak tasarlayıp ihtiyaç durumlarında ilgili parçanın çağrılıp kullanılması durumudur @Html.Partial(“path…”) ilgili view için parçalayarak bu şekilde oluşturulmasını sağlayacaktır. RenderSection tarafına PartialView dan değer gönderilmesi yoktur.