**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)

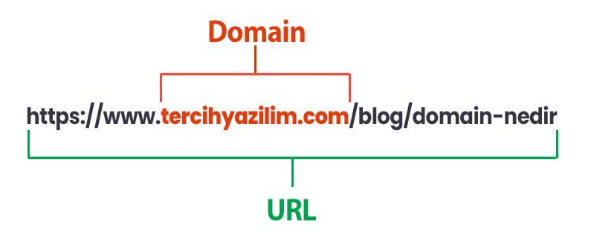
**Asp.Net**: ise .net yapısının web durumunda kullanılması için oluşturulmuş mimaridir Asp.Net mimarisi

**.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

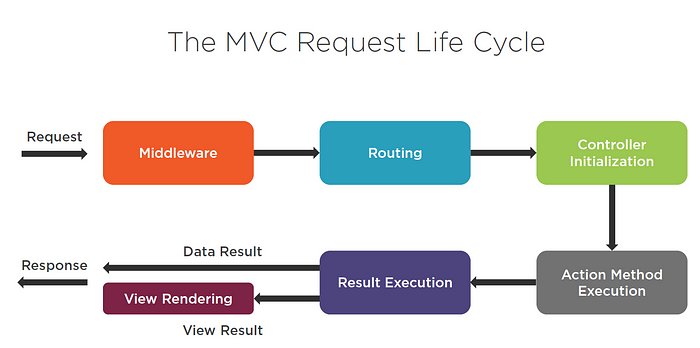
**Razor**: Asp.Net Mvc 3 teknolojisiyle birlikte gelen görüntüleme biçimidir

**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?**

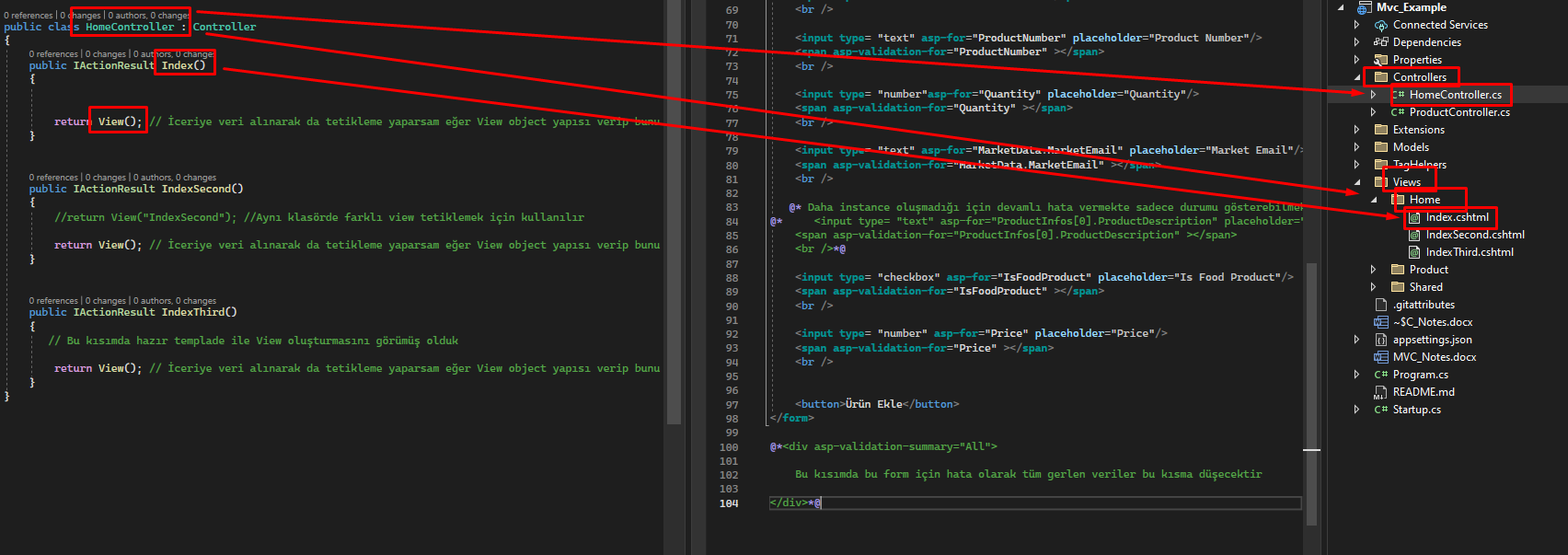
Asp.net core uygulamasında MVC mimarisini kullanmak için , ilk olarak startup tarafında ConfigureServices içinde addControllers(Sadece Controller kullanınca) ya da addControllersWithViews(Cshtml yapılı views tabanlı işlemlere yer verilicekse) şeklinde yapılandırma ile kullanım belirtilir

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 yönlendirebilirken