Web Programlama

Model-View-Controller (MVC) → Bir "mimari desen"dir. Kullanıcıya yüklü miktarda verinin sunulduğu karmaşık uygulamalarda veri ve gösterimin soyutlanması esasına dayanır. Böylece veriler (model) ve kullanıcı arayüzü (view), birbirini etkilemeden controller adı verilen ara bileşenle veri gösterimi ve kullanıcı etkileşiminden veri erişimi ve iş mantığını çıkarma suretiyle çözümlenmektedir.

MVC Nedir? (Model View Controller)

- MVC bir yazılım değil, bir mimaridir.
- MVC kavramı Microsoft'a ait bir mimari ürün değildir.

MVC Mimarisi

Model

- Veri kaynağının (veritabanı) genelde bulunduğu yerdir. Bunun yanı sıra, prosedürler ve işleyiş kuralları da bu bölümdedir. Katmanlı mimariye müthiş uyumludur. Tek katmanlı olabileceği gibi fazlaca katmana da sahip olabilir. **(.cs)**

View

- Arayüze ait olup, kullanıcının gördüğü şeyler bu bölümde yer alır. Html, JavaScript, css kodları burada yer alır. **(.cshtml)**

Controller

- İş akışının gerçekleştiği, arayüzden gelen kullanıcı etkileşimlerinin değerlendirildiği, işlendiği, gerekli metotların çalıştırıldığı, değişkenlerin ve nesnelerin oluşturulduğu, gerekirse Model ile View bölümleri arasındaki iletişimin sağlandığı yerdir. **(.cs)**

Özetle

- Client tarafından başlatılan request, controller tarafında işleme alınıyor ve neler yapılması gerekiyorsa yapıyor, sonra tekrar controller'a geliniyor ve bizim response dediğimiz sunucu cevabı Html çıktısı olarak bize geliyor.
- Her View için bir Controller vardır fakat her Controller için View şartı yoktur.

www.sau.edu.tr/Ogrenci/OgrenciEkle/5

Burada Ogrenci yazan kısım controllerdır ve bir classtır. Yani controllers klasörümüz içerisinde OgrenciController adında bir class var ve bu sınıf controller sınıfından miras alarak tüm controller özelliklerini taşıyor. OgrenciEkle ise bir **action**, yani OgrenciController class'ına ait bir metot. 5 olan kısım ise id'dir. Yani OgrenciController'ı içerisinde bulunan OgrenciEkle metodu id parametresi alıyor. Bu id örneğin kullanıcının tc nosu vs. gibi bir şey olabilir.

Get metodu ile gönderilen veriler url ile gider, post ile gönderilenler ise direkt gider.

Metodun üzerine [HttpGet] default olarak konulmuş durumdadır çünkü halihazırda zaten urlyi okuyor. Post metodu olması içinse [HttpPost] konulur. Formlarda da metot get olarak defaulttur.

Yani bir metodu hem post hem get için farklı tanımlayabiliriz yani overloading yapabiliriz.

`<form method="post" action="../Ogrenci/Kaydet">` veya bu şekilde farklı bir metot ismi yani Kaydet vererek yalnızca post işleminde çalışacak bir tanımlama da yapabiliriz.

Veritabanına alınacak yani güvenliği sağlanması gereken verileri yalnızca host tarafında javascript ile bu işi yaparsak bu güvenilir olmaz çünkü tarayıcı ayarlarından js disable seçeneği bulunmaktadır. Bu kontroller yalnızca server tarafından da yapılamaz çünkü veri kontrolü durumunda yoğunluk oluşturulabilir. Dolayısıyla bu kontrol her 2 tarafta da yapılmalıdır.

Bir modeli view'e `@model FirstProject.Models.XClass` olarak import etmeye gerek yoktur çünkü halihazırda Views klasörü içerisindeki _ViewImports.cshtml dosyasında varsayılan olarak Models klasörü her view'e import edilmektedir.

MVC'nin güzelliği üst ve alt bar sabit ve yalnızca ortası değişecek bir layout'a sahip olmasıdır da. (header ve footer sabit, yalnızca ortası yani container değişiyor.) Bunu bazı sayfalarımız için kaldırmak istiyorsak razor etiketi olan view dosyasının başındaki

@{ } arasında `Layout=null;` dediğimizde kaldırmış oluruz.

Bu layout dosyası Views/Shared klasöründe bulunur. (_Layout.cshtml)

Bu dosya incelenirse container divinin içerisinde RenderBody() fonksiyonu ile istediğimiz view'in kodlarının oraya çalıştırıldığında entegre edildiğini görürüz.

_ViewStart.cshtml dosyasında ise bu layout'un dosya adı tanımlı olarak bulunuyor.

Paylaşımlı yani ortak kullanılan dosyaların isimlerinin başları "_" ile başlar.

Herhangi bir view'de taglere bootstrap classları eklendiğinde layout html sayfasında bunlar import edildiği için otomatik algılayacaktır. (Layout=null değilse)

Bootstrap css, css, js kodları wwwroot klasörü içerisinde bulunur. Bootstrap, jquery dosyaları wwwroot klasörü içerisindeki lib klasöründe aynı isimde klasörlerinde dist klasöründe bulunur. Bunları layout kullanmayacağımız sayfalara tutup sürükleyip de import edebiliriz.

`return RedirectToAction("OgrenciListele", "Ogrenci");` ile Ogrenci controller'ı içerisindeki OgrenciListele action'ını başka bir action içerisinde döndürebiliriz.

cshtml dosyası içerisinde @{ } arasına C# kodları yazılabilir. Tek satırlık bir C# kodu yazılacaksa yalnızca @ ile yazılabilir. Yazılan bir döngü de süslüler dahil olmak üzere 1 satır olarak yazılabilir. Buna da Razor Motoru denir.

`@Html.ActionLink("Düzenle", "OgrDuzenle", "Ogrenci", new { id = ogr.OgrNo })` ile Düzenle yazan bir buton belirtebilir ve bunun Ogrenci controller'ı içerisindeki OgrDuzenle action'ını çalıştırması

ve parametre olarak ogr nesnesinin OgrNo değişkeninin id olarak gitmesini sağlayabiliriz. Tabii OgrDuzenle action'ını yani metodunu yazarken de parametre olarak int id almamız gerekir.

Model bizim için view'e ne gönderdiğimizdir.

.cshtml dosyasında razor yorum satırı `@* *@` arasına yazılır.

Html etiketleri içerisinde tek satır C# kodu yazarken ; kullanılmaz. Örn;

`<h1>@DateTime.Now.ToShortDateString()</h1>`

Daha önce tanımlanmış bir değişkeni bastırmak içinse @degiskenAdi kullanılır.

Razor etiketleri içerisinde tekrar html geçiş yapmak için `@:` kullanılır ve sonrasında html kodları yazılır.

Örneğin login sayfasında bilgileri girdikten sonra gelen yeni sayfada merhaba x yazdıracağı için o x'i login sayfasından çekmesi gerekir ve bu durumun sağlanması için server tarafında bu session'da tutulur. Yani controllerdan actionlara geçişler. Host tarafında ise bunu cookie yapar.

Bu işi server tarafında her şeyi sessionlarda yapmak mantıklı değildir, bunun yerine URL'de de bu bilgiler taşınabilir. Bu da mantıklı olmayabilir çünkü bu da manipüle edilebilir.

Ya da en mantıklısı controller'dan view'e data aktarmak için model kullanılır.

Ya da bir diğer mantıklı olan şey, ViewBag, ViewData, TempData'dır.

ViewBag'de tip dönüşümüne gerek yoktur. Controller to view data aktarımında ve sadece tek bir istekte geçerlidir, başka bir istekte data yok olur.

ViewData bir dictionarydir. Key ve value mantığıyla çalışır. Data object olarak saklanır dolayısıyla tip dönüşümü yapılmalı. Controller to view data aktarımında kullanılır ve viewbag gibi sadece tek istekte geçerlidir.

TempData da bir dictonarydir. Key ve value mantığıyla çalışır. Data object olarak saklanır dolayısıyla tip dönüşümü yapılmalı. Controller to view ya da controller to controller (controller içerisindeki actionlar yani metotlar arasında kullanabiliriz) data aktarımında kullanılır. Tek bir istekte değil, 2 istekte geçerlidir. 2 isteğe kadar datayı saklar, sonrasında yok eder.

Tanımlamaları view'e gönderilecek metot içerisinde şu şekilde tanımlanır;

```
ViewBag.msg = "ViewBag message";
ViewData["msg2"] = "ViewData message";
TempData["msg3"] = "TempData message";
```

View içerisinde şu şekilde görüntülenebilir;

```
<h2>Bu bir ViewBag -> @ViewBag.msg</h2>
<h2>Bu bir ViewData -> @ViewData["msg2"]</h2>
<h2>Bu bir TempData -> @TempData["msg3"]</h2>
```

Controller'dan view'e yalnızca 1 model parametre olarak gönderilebilir. 1'den fazla model gönderilmek istendiği durumda bunu viewbag, viewdata veya tempdata ile gönderebiliriz.

Her 2 veya daha fazla modeli 1 view'e direkt göndermek içinse bu modelleri birleştirmek gerekir. Yani tek class haline getirmek gerekir ve o class'ı (modeli) view'e göndermek gerekir. (Tavsiye edilen budur)

TagHelper'lar razor sayesinde C# kodlarını HTML'e çevirir.

CSRF (Cross Site Request Forgery) yani siteler arası istek sahtekarlığını engellemek için taghelperlardan yararlanarak form içerisinde id gibisinden input type hiddenla bir crypto kod ile bunu engelleyebiliyoruz. Bunu taghelper ile form oluşturursak razor otomatik olarak yapıyor. Form'u şu şekilde oluşturursak taghelper ile oluşturmuş oluyoruz; (asp-'ler)

`<form asp-action="OgrEkle" asp-controller="Ogrenci" method="post"> ..... </form>`

asp append version=true ile de aynı addaki fotoğrafın değişmesi sonucu cookielerden etkilenmemek için kullanılabilir. Örn;

```
<img src="/images/bir.png" asp-append-version="true" />
```

Environment Tag Helper ile de tek sunucudan ayrılan farklı sunucularda farklı işlevler sağlanabilir.

Aynı işi yapan farklı kodlar (tag helper kullanımı son örnek);

```
<a href="/Home/Privacy/10">Normal HTML Linki</a>
<br />
@Html.ActionLink("HTML Helper", "Privacy", "Home", new { id=10 });
<br />
<a href="@Url.Action("Privacy", "Home", new { id = 10 })">URL Action ile Link</a>;
<br />
<a asp-controller="Home" asp-action="Privacy" asp-route-id="10">Tag Helper ile Link</a>
```

Burada TagHelper'ın yararı örneğin sitemizin alan adından sonra x/Home/Privacy olarak yani hepsinin x altında olmasını istiyorsak bunu buradan sürekli sonradan başlarına x/ koymak yerine route'den bunu değiştirdiğimizde tag helper default gidiş adresini bileceği için bu sorun olmayacaktır.

Bir razorlu kodu html tarafından yorumlanmasını istiyorsak etiket başı ve sonuna ! koyarız

`<!a asp-controller="Home" asp-action="Privacy">Tag Helper ile Link</!a>`

Veya view kodumuzun içerisinde bir ad tanımlaması yaparak etiketin başında bu ad varsa bu kodu razor olarak yorumla da diyebiliriz. Örn th ile başlayan etiketleri razor algıla diyeceksek;

`@tagHelperPrefix th:` dedikten sonra aynı örneği şöyle yapabiliriz; (eğer bu tanımlamayı yaparsak bundan sonraki kodlarda th: koymadığımız hiçbir kodu taghelper olarak algılamaz, normalde default olarak razor kodlarını algılayabilir)

`<th:a asp-controller="Home" asp-action="Privacy">Tag Helper ile Link</th:a>`

wwwroot klasörünün altındaki x isimli bir klasöre şu şekilde view'lerden erişilebilir;

`"~/x/y.png"`

Bu TagHelper'lar _ViewImports.cshtml dosyasının içinde import edilmiş halde bulunurlar.

`<label asp-for="OgrAd">` asp-for ile model içerisindeki proplara erişim sağlanabilir fakat bunu view'in algılayabilmesi için @model Ogrenci olarak modeli önce tanıtmak gerekir.

Model içerisinde tanımladığımız propertyleri bir yerde label olarak asp-for ile verdiğimizde görünen ad değişken adının kendisi olur fakat bunu default olarak görünen adının farklı olmasını istiyorsak prop tanımlamasının üst satırına `[Display(Name = "Öğrenci Ad")]` yazmak gerekir.

ORM, (Object Relational Mapping) uygulama ile database katmanı arasındaki işleri yapan frameworktur. Yani kodumuzdaki sınıfın databasede hangi tabloya denk geldiğini ve örneğin sınıfta string olanın tabloda varchar'a denk geldiğini söyler yani mapleme işlemi yapar. C# için en sık kullanılan ORM Entity Framework'tür. Alternatif olarak Dapper da bulunur.

Code First → Önce bu yaklaşımla modeller tasarlanır ve db'ye aktarılır.

DB First → Önce db oluşturulur ardından modele bağlanır.

Modelde class'ı oluşturduğumuzda prop üzerine [Key] olarak primary key olduğunu belirtmek gerekir.

Örneğin yazar ve kitap sınıfımız var ise ve biz bunun modelini tasarlıyorsak, her yazara ait 1'den fazla kitap olabiliri sınıf içerisinde belirtmek içinse yazar classı içerisinde public ICollection<Book> Book { get; set; } demek gerekir. Kitap sınıfı içerisinde de Yazar Yazar; yani yazar tipinde yazar adında bir değişken oluşturmak gerekir.

Code First yani önce modeli oluşturuyoruz, context tanımlıyoruz (yani hangi sql'i kullanacaksak onunla ilgili olan entity framework kurulumu ve yol, server adı, id, pw tanımı vs !! bu da model içerisinde bir sınıf olacak ve DbContext sınıfından kalıtım almalı (bu sınıf içerisinde hangi modelin hangi adda tablo oluşturacağı vs. gibi işlemler)) ardından ise migration işlemi (bunun için de entity framework tools gerekiyor) yani database'e taşıma işlemini otomatik olarak orm yapar.

Sonrasında tabloya yeni bir prop eklemek için bunu database üzerinde tablodan değil de yine model içerisinden yapmak gerekir. Model üzerinde güncelleme yaptıktan sonra yeni bir migration yaratmak gerekir.

Database'i bağlama işlemi startup dosyası içerisindeki ConfigureServices metodu içerisinde tanımlanır.

`[Bind ("AuthorFName", "AuthorLName")]Author a` 'ı parametre alarak listeye yalnızca bind ile belirttiğimiz parametreleri almasını sağlayabiliyoruz.

Migration için database update komutu en son migration'ı update eder.

Model içerisindeki tanımlamalarda eğer key tagı bulundurmayıp, primary key'i belirtmez isek bu durumda proplardan adında ilk ID geçeni primary key olarak algılar. Aynı şekilde eğer foreign key olduğunu da bir prop'un belirtmez isek bu durumda diğer bir modelde bulunan id var mı ona bakar ve eğer var ise onun foreign key olduğunu anlar.

Eğer ki migration eklerken bir isim vermezsek son migration'ın up metodunu çağırarak üzerine güncelleme yapılacağını anlar. Remove migration yaptığımızda ise çağırdığımız migration'ın down metodunu çağırır yani bir önceki haline geri döner.

db nesnesi adı örneğin db olsun; `db.Add(a→listeye yeni eleman)` ve sonrasında `db.SaveChanges();` dediğimizde veritabanı üzerinde güncelleme olur.

LINQ (Language Integrated Query) → Kodumuz üzerinden sorgu yapabilmemizi sağlayan ve bu sorgumuzda bir hata varsa bunu compile esnasında fark edebilen bir yapıdır.

EF için Linq to EF, XML için Linq to XML yani farklı data sourcelar için onlar için sorgu yapan gibi bir yapısı sayesinde bunu yapabiliyor ve bu yapıya da Linq Provider denir.

Linq içerisinde Query, Method ve her ikisi barındıran Mixed yapıları bulunur.

Query tamamen sql içerisinde yazdığımız seçme ve veri çekmedir.

Method ise Where() gibi metotlu işlemlerdir.

Mixed ise her ikisini de barındıran yapıdır.

.ToList() yerine .FirstOrDefault() dersek bulduğu ilkini getirir. Bulamazsa NULL döndürür.

Query'nin Method şeklinde yazımı örneğin şu query için;

```
(from bk in db.Books where bk.BookTitle.Contains("suc") select bk).ToList();
```

```
db.Books.Where(x => x.BookTitle.Contains("suc")).ToList();
```

Bu where sonrası bir değişken adı ve => ile erişimi sağlanıyor LinQ'da.

View Component → Web sitemiz üzerinde hemen hemen her modülde yahut sayfada birebir benzer işlemleri gerçekleştireceksek yani aynı kodları çalıştırmamız gerekecekse bunun için kullanırız. Örneğin bir sayfa üzerinde farklı şehirlerin ilçelerini yazdıracaksak aslında bu durumda ortak bölge için de ortak bir kod bölümü oluşturabiliriz. (Layout'taki footer ve header mantığı) Bu yapı yalnızca model ile view component arasında iletişim kurar, controller'a ihtiyaç duymaz.

Örneğin SehirListele adında bir view component'imiz var ise bunu tag helperlar ile de veya c# kodu ile de çağırabiliyoruz. (Tag helper ile çağırmak için _ViewImports dosyasının içerisine @addTagHelper *, projeadi şeklinde tanımlama yapılması gerekiyor)

C# → @await Component.InvokeAsync("SehirListele")

TagHelper → <vc:sehir-listele</vc:sehir-listele>

TagHelper kullanımında vc: kullanımı view component anlamındadır.

Alttaki 2 kavram birbirleri ile irintili kavramlardır.

Authentication → Bir yere erişim

Authorization → Erişimden sonra erişim dahilindeki izinler, yetkilendirme

Yani örneğin bir web sitesine kullanıcı girişi yaparken olan şey authentication yani doğrulama, sonrasında bu girişi yapan kişinin ne konularda yetkilerinin olacağının belirlenmesi konusu ise authorization yani yetkilendirmedir.

Authentication'ın farklı türleri vardir;

SFA(Single Factor Authentication) → Yalnızca kullanıcı adı ve şifrenin kontrolünün yapıldığı sistemdir. Bilgiler doğru ise sisteme giriş izni veriyor, yanlışsa atıyor.

TFA(Two Factor Authentication) → 2 doğrulamanın olduğu türdür. Öncelikle sfa yapılır ardından başka bir yöntemle yeniden doğrulama yapılır, örneğin kredi kartı dört hanesi.

MFA(Multi Factor Authentication) → Çoklu doğrulamanın olduğu türdür. Öncelikle sfa yapılır, sonrasında cihaz eşleşiyor mu, sonrasında ise sms atıyor.

Örneğin veritabanında şifre tutuyoruz fakat bunu yöneticiler dahi göremiyor çünkü hashlememiz yani şifreleme fonksiyonundan geçirmemiz gerekiyor. Bunun sorgulamasını yaparken nasıl yapacağız? Sorgulamasını da aynı şekilde şifreleme fonksiyonundan geçirerek şifrelenmiş verinin eşleşip eşleşmediğini kontrol edeceğiz.

Örneğin bir sayfaya yalnızca login olan yani kişiler erişebilsin istiyorsak o controllerda istediğimiz view'in metodunun üzerine `[Authorize]` eklememiz gerekir.

Controller içerisinde herhangi bir view'e yani metodun başına örneğin `[Authorize(Roles="Admin")]` yazmak yalnızca "Admin" roluna sahip olan kullanıcılar bu sayfayı görebilsin demek. Bu kodu bir view metodunun üstüne değil de komple controller'ın üzerine yazarsak controller içerisindeki tüm actionları kapsar.

[AllowAnonymous] demek ise kullanıcı veya değil herhangi herkes girebilsin demek. Bu default olarak eğer başka bir authorize eklemezsek her controller'da veya action'da var.

Bir şeyin bir role sahip olanlar haricinde görülmemesi isteniyorsa o view içerisinde if ile User.IsInRole("Admin") komutu ile örneğin Adminse görebilsin gibisinden işlem yapılabilir. Tabii öncesinde de bir başka if ile User.Identity.IsAuthenticated koşulu da sağlanmalı.

Web Api uygulamadaki bazı veya tüm servisleri dışarıya açık hale getirmektir. Örneğin bir mağazamız varsa bizim belirlediğimiz url ile bu urlde bu mağaza kodunu parametre olarak dışarıdan biri verirse o mağazadaki tüm ürünler json (veya xml) olarak geri döndürülür. Bu Web Api'yi biz bu derste sunan tarafta olacağız.

SOAP → Bir protokoldür. Apilere göre daha hantaldır fakat apilere göre bir nebze daha güvenilirdir. XML kullanır, istekleri HTTP veya SMTP aracılığıyla alır.

Rest/RestFull Api → Mimari bir tarzdır, protokol değildir. JSON kullanılarak oluşturulur. Verilere erişmek ve onları kullanmak için HTTP isteklerini kullanan bir mimari yapıdır.

(HTTP) Method-Verb → İstek yöntemi sunucuya istemcinin sunucunun ne tür bir işlem yapmasını istediğini söyler. Her istek bir HTTP metoduna sahip olmalıdır. RestFull API'lerde en sık kullanılan yöntemler, get, post, put ve deletedir. Get veri çekme, post veri ekleme, put veri güncelleme, delete ise veri silme işlemlerinde kullanılan yöntemlerdir. Bu işlemlere CRUD (Create, Read, Update, Delete) denir.

Get ile gönderilen veriler URL'de, post ile gönderilen veriler ise Body'de gönderilir.

Bir controller'ı api controller'ı haline getirmek için ControllerBase sınıfından kalıtım almak ve o sınıfın başına [ApiController] ve [Route("api/[controller]")] eklemek gerekir. Örneğin Ogrenci

adında bir api oluşturacaksak;

HTTP METODU	URI	İŞLEM
GET	http://localhost/api/Ogrenci/	Tüm öğrenci öğelerini getir
GET	http://localhost/api/Ogrenci/10	10 numaralı kimlik bilgisine sahip öğrenci verisini getir
DELETE	http://localhost/api/Ogrenci/10	Kimlik numarası 10 olan öğrenci kaydını sil
PUT	http://localhost/api/Ogrenci/10	10 kimlik numaralı öğrenci kaydını güncelle (Güncelleme için veriler http isteğinin gövde kısmında)
POST	http://localhost/api/Ogrenci/	Yeni bir öğrenci ekle (Öğrenci verileri http isteğinin gövde kısmında)

URL'ler örnekteki gibi girilirse ve yapılacak işlemler http metodları gibi olursa olacak işlem görünmekte.

```
[Route("api/[controller]")]
[ApiController]
public class ApiDenemeController : ControllerBase
{
    [HttpGet]
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

[HttpGet("{id}")]
    public string Get(int id)
    {
        return "value";
    }

    [HttpPost]
    public void Post([FromBody] string value)
    {
    }

    [HttpPut("{id}")]
    public void Put(int id, [FromBody] string value)
    {
    }

    [HttpDelete("{id}")]
    public void Delete(int id)
    {
    }
}
```

Postman programı, apiyi test edebilmemizi sağlar.

Bir MVC projesi içerisinde Api Controller, Add Controller>Api with read/write action ile oluşturulur.

Olmayan bir prop adı ile bir put veya post isteğinde bulunulursa veya örneğin bir prop'a model içerisinde required özniteliği verilirse ve istekte bu prop boş bırakılırsa da bu durumda 400 bad req hatası alınır. Olmayan ID'ye istekte bulunulması durumunda ise 404 not found hatası alınır. Başarılı isteklerde ise 200 bilgisi alınır. Bunun doğruluğu ApiController'ın başına koyduğumuz [ApiController] özniteliğinden kaynaklı, yani bu öznitelik model üzerinden kontrolleri gerçekleştirmeyi sağlıyor.

Globalization ve Localization, uygulamamız için çoklu dil desteği sağlamamıza yarar. Bu aynı zamanda o dilin ve dilin kullanıldığı bölgenin örneğin tarih, sayı, para birimi gibi şeylerin de o bölgeye uygun olmasını sağlar.

Culture ve UICulture olarak tanımlanır. Örneğin en-US

Bu işlem url üzerinden, cookie üzerinden veya tarayıcıya ait bir yapı ile yapılabilir.

Bunu MVC projemize entegre etmek için öncelikle Startup.cs dosyası içerisindeki ConfigureServices metoduna

```
services.AddMvc().AddViewLocalization(LanguageViewLocationExpanderFormat.Suffix);
            services.Configure<RequestLocalizationOptions>(options =>
            {
                var supportedCultures = new[]
                {
                    new CultureInfo("tr")
                };
                options.DefaultRequestCulture = new RequestCulture(culture: "tr", uiCulture: "tr");
                options.SupportedCultures = supportedCultures;
                options.SupportedUICultures = supportedCultures;
            });
```

eklemek gerekiyor.

Configure metodu altındaki app.UseRouting(); komutu altına ise

```
var locOptions = app.ApplicationServices.GetService<IOptions<RequestLocalizationOptions>>();
app.UseRequestLocalization(locOptions.Value);
```

eklemek gerekiyor.

Cookie ile bu işi yapacaksak Home Controller'ı içerisine cookie için de bir metot ekleriz. Bunu da ekledikten sonra .resx dil dosyalarını dosyaların bulunduğu yere yerleştiririz. View dosyası içerisinde yaptığımız çevirinin dil değiştiğinde çalışabilmesi için view dosyası başına IViewLocalizer'ı inject etmeliyiz.

`@inject Microsoft.AspNetCore.Mvc.Localization.IViewLocalizer localizer`

Ardından çevirinin olacağı yerde @localizer kullandığımızda orada çeviri işlemi sağlanacaktır. Örn;

`@localizer["İçerik"]` dediğimizde örneğin dili eng yaptığımızda orada Context olarak görülecektir.

Dependency Injection, Interface ile kullanılan bir tasarım desenidir.

Nesne yönelimli programlama dillerinde arayüz, değişik sınıflardan nesnelerin kategorize edilmesini sağlayan bir soyut tür çeşitidir. Sözleşmedir.

Örneğin bir programda bir veritabanına bağlanmamız gerekiyor ve bunu örneğin MySQL'e göre değil de herhangi bir veritabanına bağlayabilecek şekilde organize etmemiz gerekiyor. Bunu da interfaceler sağlayabilir.

Interfacede ve interface'e bağlanacak sınıflarda aynı imzaya sahip fonksiyonlar bulunmak zorundadır.

Örn:

```
public interface VeritabaniSurucu {
    public void baglan();
    public void baglantiSonlandir();
}
```

şeklinde interface anahtar kelimesiyle interface oluşturulur.

```
public class PostgreSQLSurucu : VeritabaniSurucu {
    public void baglan() {
        //PostgreSQL veritabanına bağlanıyor...
    }
    public void baglantiSonlandir() {
        //PostgreSQL veritabanı bağlantısı sonlandırılıyor...
    }
}
public class MySQLSurucu : VeritabaniSurucu {
    public void baglan() {
        //MySQL veritabanına bağlanıyor...
    }
    public void baglantiSonlandir() {
        //MySQL veritabanı bağlantısı sonlandırılıyor...
    }
}
```

şeklinde ise soyutlanacak sınıflarda : interfaceAdi şeklinde kalıtım alınır.

Daha sonra Startup.cs dosyası içerisindeki ConfigureServices metodu içerisine IMesaj tipinde değişken alan yerler yerine hangi tipte bu interface'den kalıtım alan nesne gönderilsini belirtiyoruz. Örneğin PostgreSql'i alsın.

`services.AddScoped<VeritabaniSurucu, PostgreSQLSurucu>();`

AddScoped ile VeritabaniSurucu'ye ait kurucuda parametre alınan instancelar her yerde aynı kabul edilir yani farklı oluşturulmaz. AddTransient ile de aynı işi yapabilirdik fakat bunun farkı her instance her yerde tekrar oluşturulur.

```
public  class VeritabaniIslemleri {
    private VeritabaniSurucu veritabani;

    public VeritabaniIslemleri(VeritabaniSurucu veritabani) {
        this.veritabani = veritabani;
    }
    public void baglan(){
        veritabani.baglan();
    }
    public void baglantiSonlandir(){
        veritabani.baglantiSonlandir();
    }
}
```

şeklinde olsun. Bu durumda artık kurucu fonksiyona PostgreSQLSurucu gönderilecektir.

Session server üzerinde tutulan bir depolama yeridir. Yani çok fazla sayfada örneğin kullanıcının adını tutacaksak yani her sayfada bunun görüntülenebilmesini istiyorsak örneğin x kullanıcısının adını session'da tutarız. Her session'ın bilgilerin birbirinden ayırt edilebilmesi için id'si vardır. Örneğin x kullanıcısının bilgileri id=1, y kullanıcısının id=2 session'ında tutuluyor.

Cookie ise client tarafında tutulan bir depolama yeridir. Mantığı session ile aynıdır, tek farkı tutuldukları yer. Dolayısıyla bu tarz bir işlem yapmak için en mantıklı, güvenilir şey bunları session'da tutmaktır. Cookie'de dil, arkaplan rengi gibi basit şeyleri tutmak mantıklıdır.

QueryString ise urlden bilgi çekmek oluyor.

Örn action adından sonra ? ve sonrasında herhangi bir parametre alabiliriz. Örn ad'ı querystring'den çekeceksek ve QueryStringTest adında bir action'dan bunu yaptığımızı varsayalım;

```
public string QueryStringTest()
{
  string msg="";
  if (!string.IsNullOrEmpty(HttpContext.Request.Query["ad"]))
  {
    msg = "Merhaba " + HttpContext.Request.Query["ad"];
  }
  else
  {
    msg = "Herhangi bir ad yok.";
  }
  return msg;
}
```

Yani şöyle ki eğer https://localhost:44354/Home/QueryStringTest?ad=baris şeklinde gidersek Merhaba baris yazar fakat yalnızca

Eğer https://localhost:44354/Home/QueryStringTest olarak gidersek bu durumda herhangi bir ad yok yazdıracaktır.

HttpContext bu işe yarıyor, aynı zamanda session ve cookie kullanımında da HttpContext kullanılıyor.

Session kullanabilmek için öncelikle Startup içerisinde ConfigureServices metodunda `services.AddSession();` ve Configure içerisinde ise `app.UseSession();` komutlarını yazmak gerekiyor. Controller action'ın içerisinde session'ı tanımlamak içinse örn Index'de;

```
public IActionResult Index()
{
  HttpContext.Session.SetString("DogumYili", "2000");
  HttpContext.Session.SetInt32("Yas", 18);
  return View();
}
```

HttpContext.Session.SetString veya SetInt32 ile key-value şeklinde session tanımlaması yapılabilir. View içerisinde bu sessionlara erişebilmek içinse yine Startup içerisinde ConfigureServices içerisine `services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();` eklemek gerekiyor.

Örneğin Index view'inde bu sessionlara erişebilmek için öncelikle view dosyası içerisinde;

```
@using Microsoft.AspNetCore.Http
@inject Microsoft.AspNetCore.Http.IHttpContextAccessor HttpContextAccessor
```

eklemek gerekiyor. Ardından;

```
<h2>@HttpContextAccessor.HttpContext.Session.GetString("DogumYili")
</h2>
<h2>@HttpContextAccessor.HttpContext.Session.GetInt32("Yas")</h2>
```

şeklinde Get ile session verilerine erişilebilir.

Home Controller'ı içerisinde Index action'ında biz bu tanımlamaları yapmamıza rağmen başka bir controller'ın başka bir action'ının view dosyasında da aynı yukarıda Index view'inde yaptığımız gibi o Controller veya action içerisinde tanımlamayı yapmamamıza rağmen session'lara erişebiliriz. Asp.Net'de default session süresi 20dk'dır. Yani 20dk sonra sessionlar kaybolur, log in olmuşsak log out yapar. Bu süre arttırılabilir.

Görüleceği üzere session'da yalnızca SetString ve SetInt32 fonksiyonları bulunuyor. Eğer biz bir sınıfa ait nesne göndermek istersek? Yani OgrAd ve OgrNo proplarına sahip Ogrenci sınıfından ogr adında bir nesneyi göndermek istesek bu durumda ne yapabiliriz? Bu durumda nesnenin proplarını serialize ederiz ve string haline getirmiş olduğumuzdan string olarak gönderir, sonrasında getstring ile alır ve deserialize ederek değere erişiriz. Serialize-deserialize işlemini de JsonConverter ile yapıyoruz.

Bunun için SessionExtension adında bir sınıf oluşturuyoruz ve SetObject, GetObject metotları tanımlıyoruz. T'ler Generic Sınıf anlamına geliyor, yani sınıftan bir nesne alacak.

```
public static void SetObject<T>(this ISession session, string key, T value)
{
  session.SetString(key, JsonConvert.SerializeObject(value));
}
```

```
public static T GetObject<T>(this ISession session, string key)
{
  var value = session.GetString(key);
  return value == null ? default(T) : JsonConvert.DeserializeObject<T>(value);
}
```

GetObject'de return value'de bulunan default(T) burada default(T) T'nin yani sınıfın default değerini yani asp.net'de null'u döndürüyor.

Bu sayede artık HttpContext.Session.SetObject ve GetObject kullanabiliyoruz. Index metodu içerisinde Ogrenci sınıfından ogr nesnesi oluşturup set edelim örneğin;

```
Ogrenci ogr = new Ogrenci() { OgrAd = "Baris", OgrNo = 10};
```

```
HttpContext.Session.SetObject<Ogrenci>("Ogrenci", ogr);
```

bu şekilde nesneyi Session ile gönderebilmiş olduk.

Sonrasında Index view içerisinde;

```
var ogr=HttpContextAccessor.HttpContext.Session.GetObject<Ogrenci>("Ogrenci");
```

tanımladıktan sonra artık `<h2>@ogr.OgrAd</h2>` şeklinde istediğimiz prop'una erişebiliriz.

Cookies içinse aynı şekilde Home Controller'ı içerisindeki Index action'ına `HttpContext.Response.Cookies.Append("WebId", "10");` şeklinde WebId adında bir cookie tanımlarsak bunu view içerisinde;

```
<h2>@HttpContextAccessor.HttpContext.Request.Cookies["WebId"].ToString()</h2>
```

şeklinde görüntüleyebiliriz.

Aynı şekilde session'da olduğu gibi başka controller'ların başka viewlerinde de bu şekilde cookie değerine erişebiliriz.

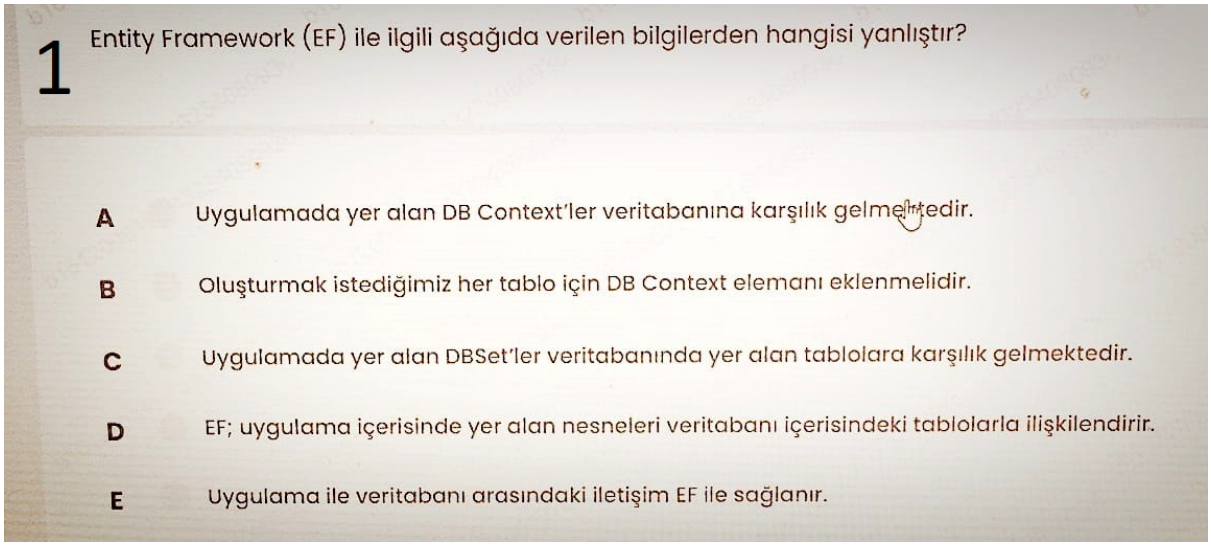
Fakat! Örneğin x controller'ının index'inde bu şekilde cookie'ye eriştik ve cookie değerini değiştirdik, bu durumda değiştirdiğimiz değeri x controller'ının index'inde görebiliriz fakat Home controller'ının index'inde göremeyiz çünkü biz cookie'yi o metot içerisinde response ettik yani tanımladık, dolayısıyla tekrar yazma 10 yazma işlemi var. Bu durumda bunu nasıl aşarız? Bu durumda cookie tanımlamasını Home>Index içerisinde şu şekilde yaparsak;

```
if (!HttpContext.Request.Cookies.ContainsKey("WebId"))
{
  HttpContext.Response.Cookies.Append("WebId", "10");
}
```

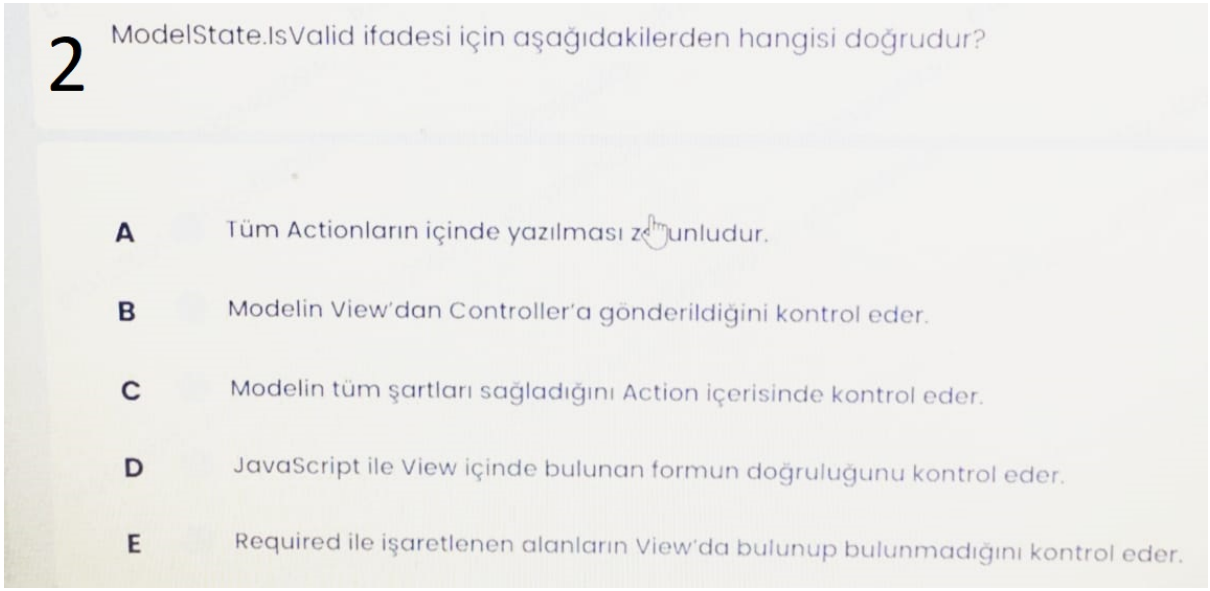
bu durumda eğer WebId adında bir cookie mevcut değilse yeniden yazmamasını sağlayabiliriz.

SORULAR

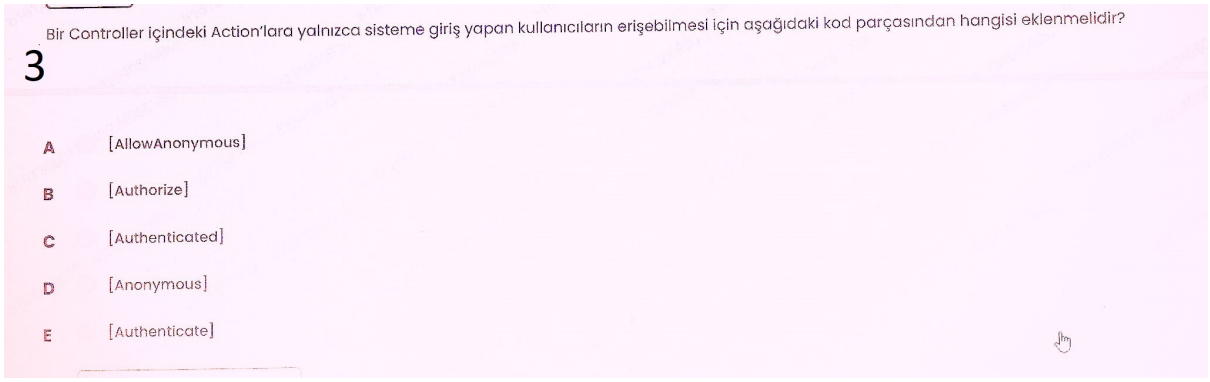
B


1
Entity Framework (EF) ile ilgili aşağıda verilen bilgilerden hangisi yanlıştır?
A Uygulamada yer alan DB Context'ler veritabanına karşılık gelmektedir.
B Oluşturmak istediğimiz her tablo için DB Context elemanı eklenmelidir.
C Uygulamada yer alan DBSet'ler veritabanında yer alan tablolara karşılık gelmektedir.
D EF; uygulama içerisinde yer alan nesneleri veritabanı içerisindeki tablolarla ilişkilendirir.
E Uygulama ile veritabanı arasındaki iletişim EF ile sağlanır.

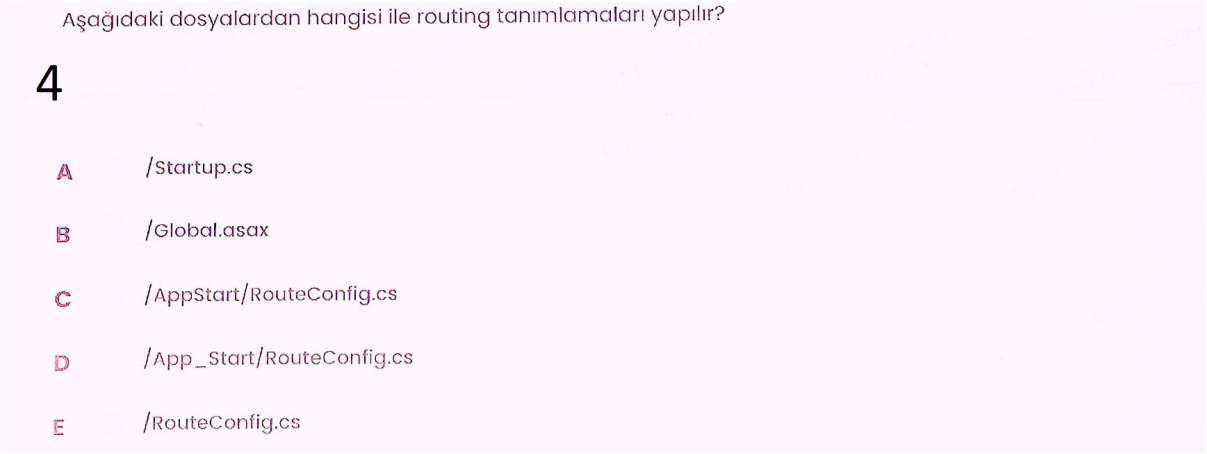
C


2
ModelState.IsValid ifadesi için aşağıdakilerden hangisi doğrudur?
A Tüm Actionların içinde yazılması zorunludur.
B Modelin View'dan Controller'a gönderildiğini kontrol eder.
C Modelin tüm şartları sağladığını Action içerisinde kontrol eder.
D JavaScript ile View içinde bulunan formun doğruluğunu kontrol eder.
E Required ile işaretlenen alanların View'da bulunup bulunmadığını kontrol eder.

B

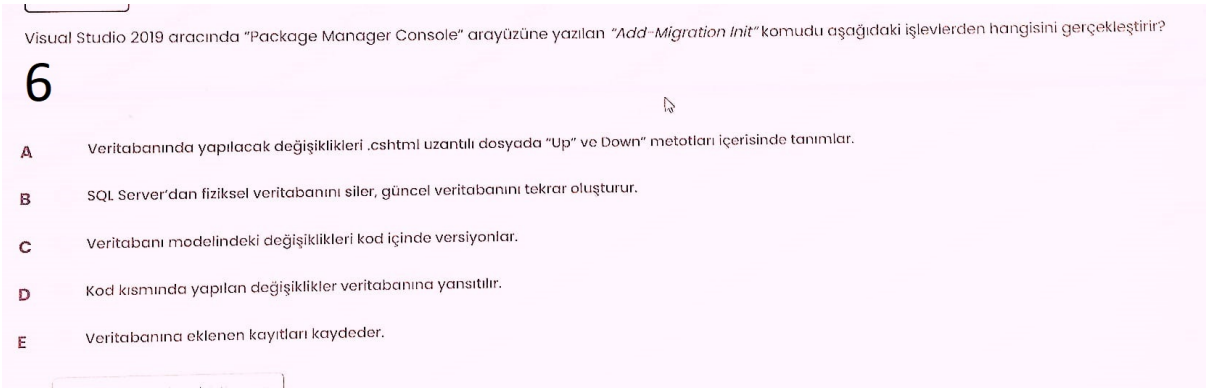

3
Bir Controller içindeki Action'lara yalnızca sisteme giriş yapan kullanıcıların erişebilmesi için aşağıdaki kod parçasından hangisi eklenmelidir?
A [AllowAnonymous]
B [Authorize]
C [Authenticated]
D [Anonymous]
E [Authenticate]

A

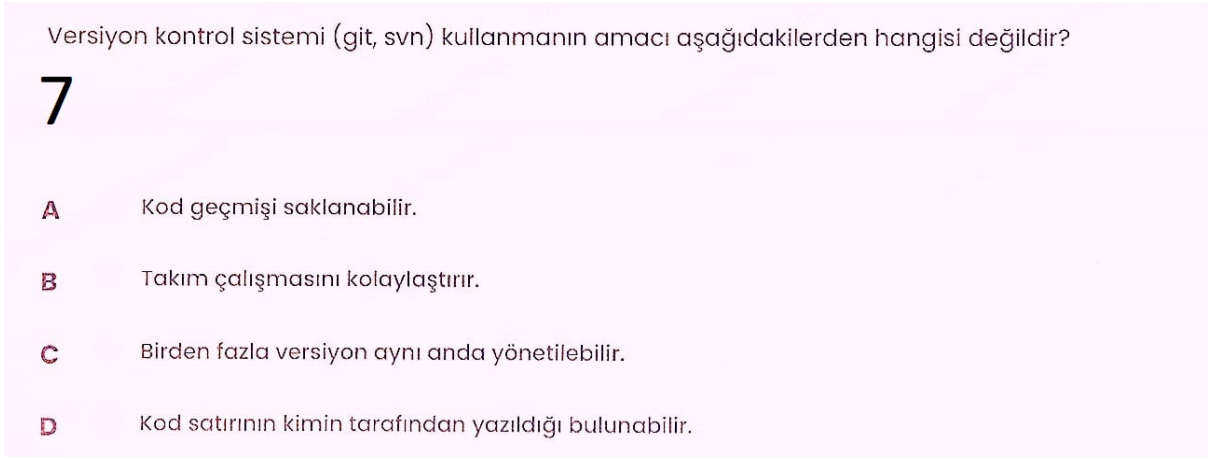


Olası js hatasına karşın sayfanın geç yüklenmesini engellemek, hata oluştuğunda sayfa yüklendikten sonra yavaşlamayı sağlamak.

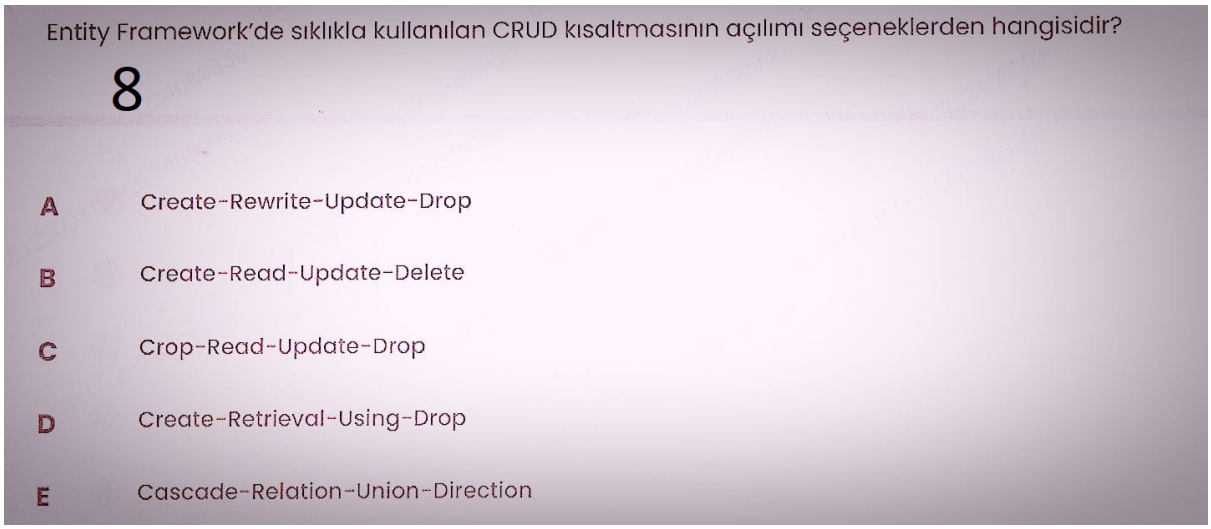
A



D


Versiyon kontrol sistemi (git, svn) kullanmanın amacı aşağıdakilerden hangisi değildir?
7
A Kod geçmişi saklanabilir.
B Takım çalışmasını kolaylaştırır.
C Birden fazla versiyon aynı anda yönetilebilir.
D Kod satırının kimin tarafından yazıldığı bulunabilir.

B


Entity Framework'de sıklıkla kullanılan CRUD kısaltmasının açılımı seçeneklerden hangisidir?
8
A Create-Rewrite-Update-Drop
B Create-Read-Update-Delete
C Crop-Read-Update-Drop
D Create-Retrieval-Using-Drop
E Cascade-Relation-Union-Direction

A

9

Database First yaklaşımı kullanılarak bir Veritabanı modeli oluşturulmuştur. Bu modele göre aşağıdaki soruyu cevaplandırınız.

UrunID=2 olan bir ürünün bugün sepete kaç adet eklendiği bilgisini sunan Linq sorgusu aşağıdakilerden hangisinde doğru olarak verilmiştir?

A `var appDb = _context.Sepetler.Where(x => x.FKUrunID == 2 && x.Tarih == DateTime.Today).Sum(x => x.Adet);`

B `var appDb = _context.Sepetler.Where(x => x.FKUrunID == 2 && x.Tarih == DateTime.Today).Select(x => x.Adet);`

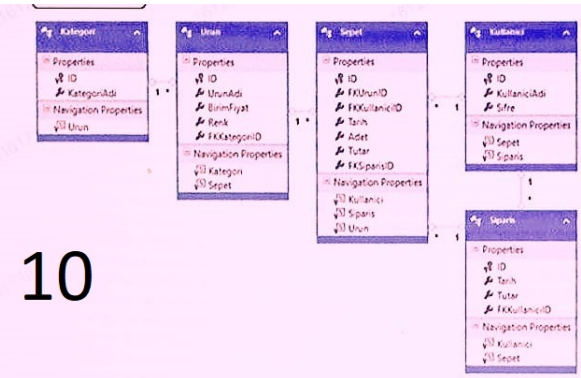
C `var appDb = _context.Sepetler.Any(x => x.FKUrunID == 2 && x.Tarih == DateTime.Today).Select(x => x.Adet);`

D `var appDb = _context.Sepetler.Where(x => x.FKUrunID == 2 && x.Tarih == DateTime.Now).Sum(x => x.Adet);`

E `var appDb = _context.Sepetler.Having(x => x.FKUrunID == 2 && x.Tarih == DateTime.Today).Sum(x => x.Adet);`

A

10



Database First yaklaşımı kullanılarak bir Veritabanı modeli oluşturulmuştur. Bu modele göre aşağıdaki soruyu cevaplandırınız.

Ayakkabı kategorisinden bir ürün içeren siparişleri listeleyen Linq sorgusu aşağıdakilerden hangisinde doğru olarak verilmiştir?

A `db.Siparis.Where(x => x.Sepet.Any(y => y.Urun.Kategori.KategoriAdi == "Ayakkabı"))`

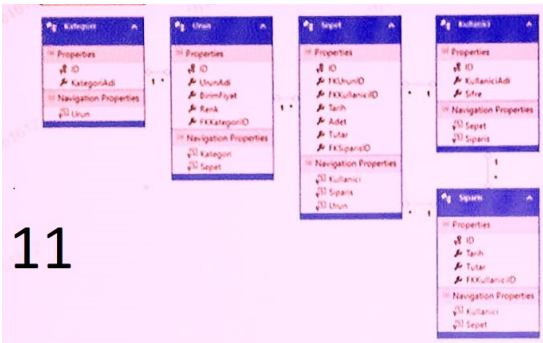
B `db.Kategori.Where(x => x.KategoriAdi == "Ayakkabi").Select(x=>x.Siparis)`

C `db.Kategori.Where(x => x.KategoriAdi == "Ayakkabı").Select(x=>x.Sepet.Siparis)`

D `db.Sepet.Where(x => x.Siparis.Any(y => y.Urun.Kategori.KategoriAdi == "Ayakkabı"))`

E `db.Siparis.Where(x => x.Urun.Any(y => y.Kategori.KategoriAdi == "Ayakkabı"))`

C



11

Database First yaklaşımı kullanılarak bir Veritabanı modeli oluşturulmuştur. Bu modele göre aşağıdaki soruyu cevaplandırınız.

KullaniciID=2 olan kullanicinin 2 numaralı siparişine ait toplam tutarı gösteren Linq sorgusu aşağıdakilerden hangisinde doğru olarak verilmiştir?

A `var appDb = _context.Sepet.Sum(x => x.Tutar). Where(x => x.FKKullaniciID == 2 && x.FKSiparisID == 2);`

B `var appDb = _context.Siparis.Where(x => x.FKKullaniciID == 2 || x.FKSiparisID == 2).Average(x => x.Tutar);`

C `var appDb = _context.Sepet.Where(x => x.FKKullaniciID == 2 && x.FKSiparisID == 2).Sum(x => x.Tutar);`

D `var appDb = _context.Sepet.Any(x => x.FKKullaniciID == 2 && x.FKSiparisID == 2).Sum(x => x.Tutar);`

E `var appDb = _context.Sepet.Where(x => x.FKKullaniciID == 2 || x.FKSiparisID == 2).Sum(x => x.Tutar);`

E

ASP.Net Web Api için aşağıdaki ifadelerden hangisi doğrudur?

12

A Json formatında çıktı alınamaz

B XML formatında çıktı alınamaz

C SOAP ile çalışamaz

D View kullanılır

E Controller zorunludur

A

Layoutta "scripts" adında section tanımlayabilmek için aşağıdakilerden hangisi yazılmalıdır?

13

A @RenderPartial("scripts")

B @CreateSection("scripts", required: false)

C @Html.Scripts()

D @Html.Section("scripts", required: false)

A

Aşağıdakilerden hangisi "_Footer" isimli partial viewın çıktısını ekrana basar?

14

A @Html.Partial("_Footer")

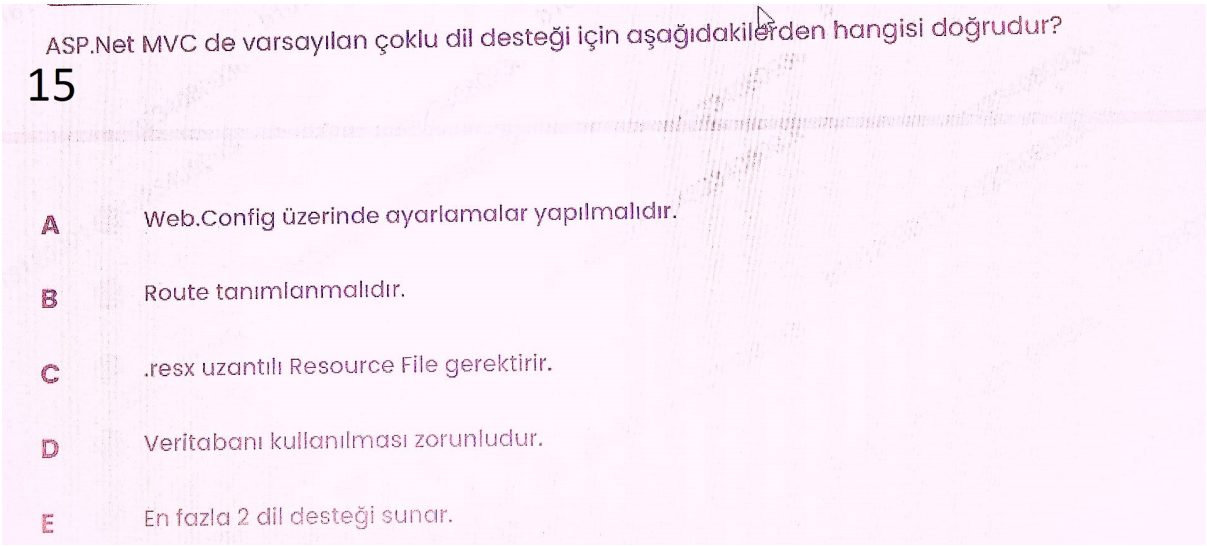
B @Partial("_Footer")

C @Render("_Footer")

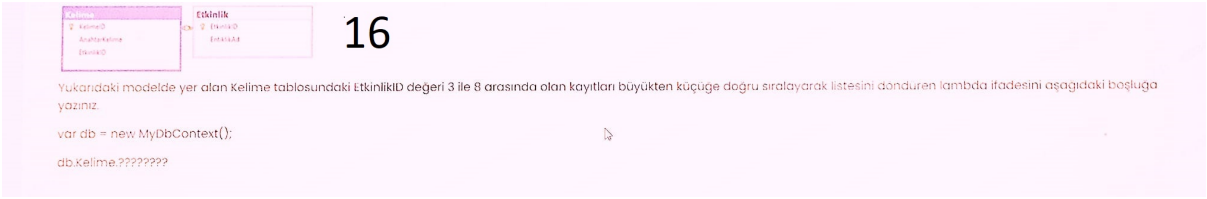
D @Html.RenderAsView("_Footer")

E @Html.View("_Footer")

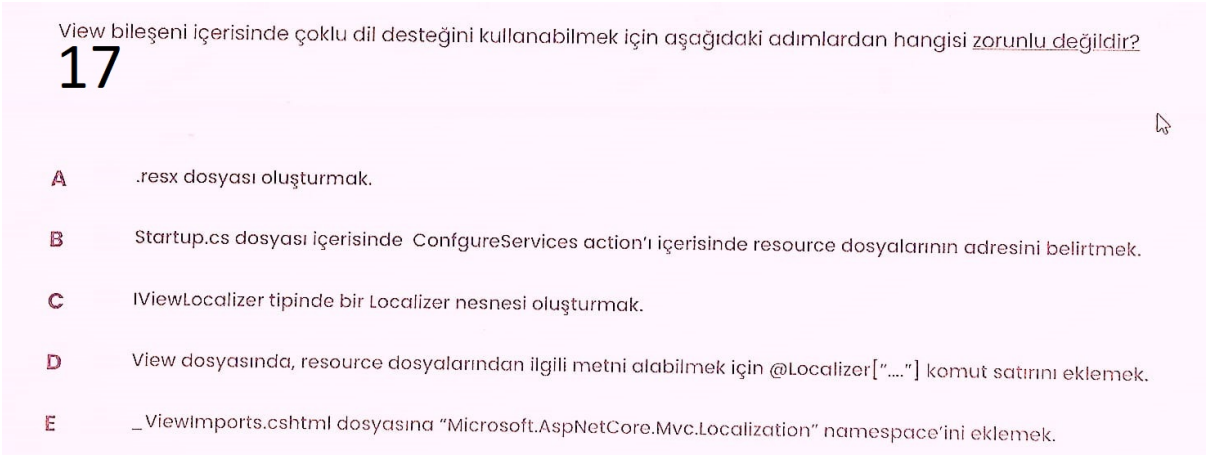
C



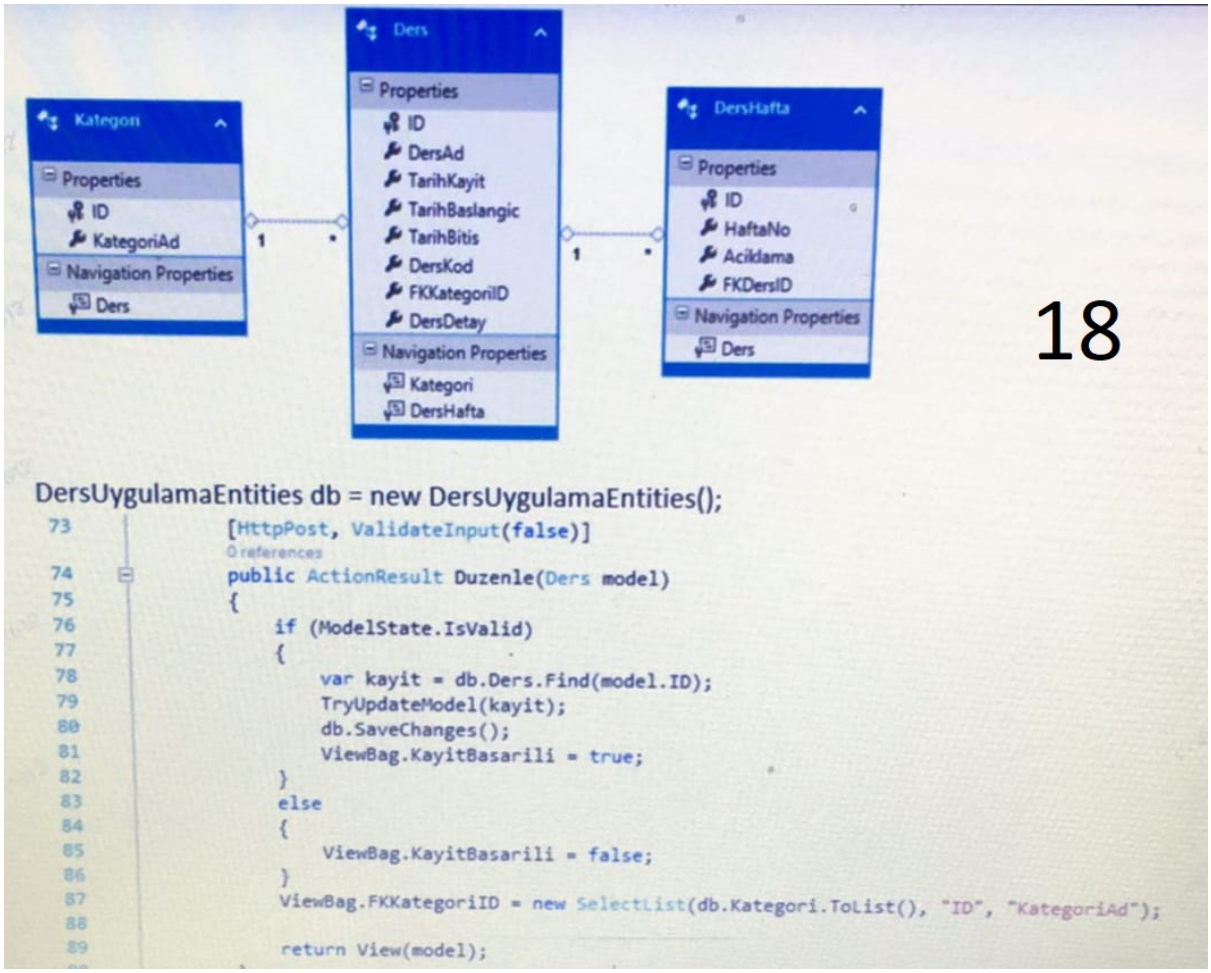
db.Kelime.Where(x => x.EtkinlikID < 8 && x.EtkinlikID > 3).OrderByDescending(y => y.EtkinlikID).ToList()

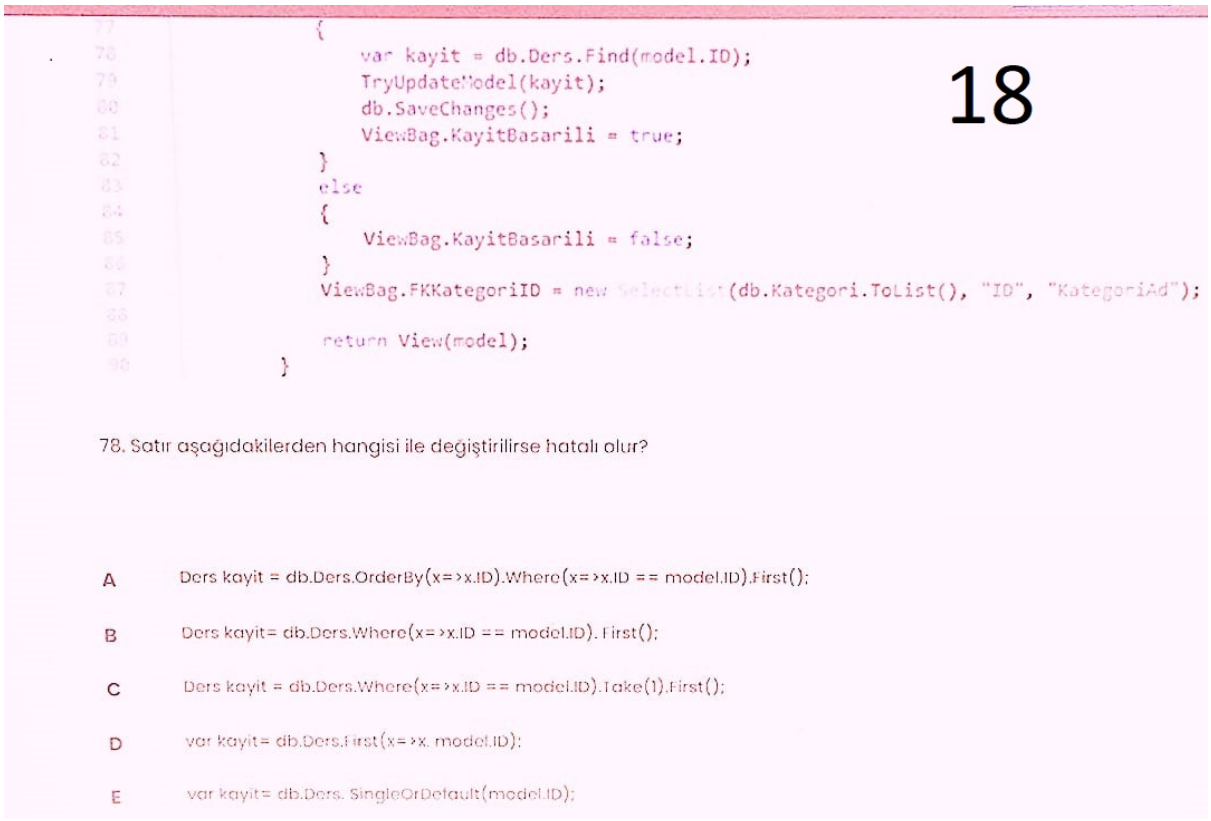


E



D





78. Satır aşağıdakilerden hangisi ile değiştirilirse hatalı olur?

A Ders kayit = db.Ders.OrderBy(x=>x.ID).Where(x=>x.ID == model.ID).First();

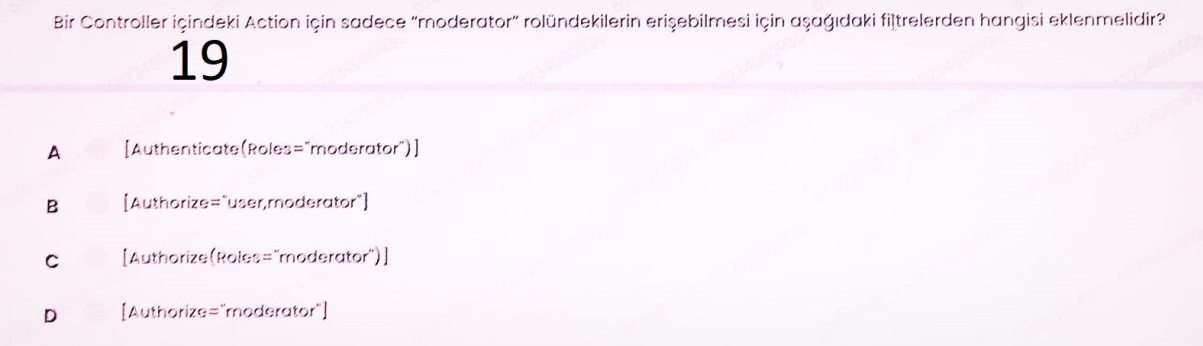
B Ders kayit= db.Ders.Where(x=>x.ID == model.ID). First();

C Ders kayit = db.Ders.Where(x=>x.ID == model.ID).Take(1).First();

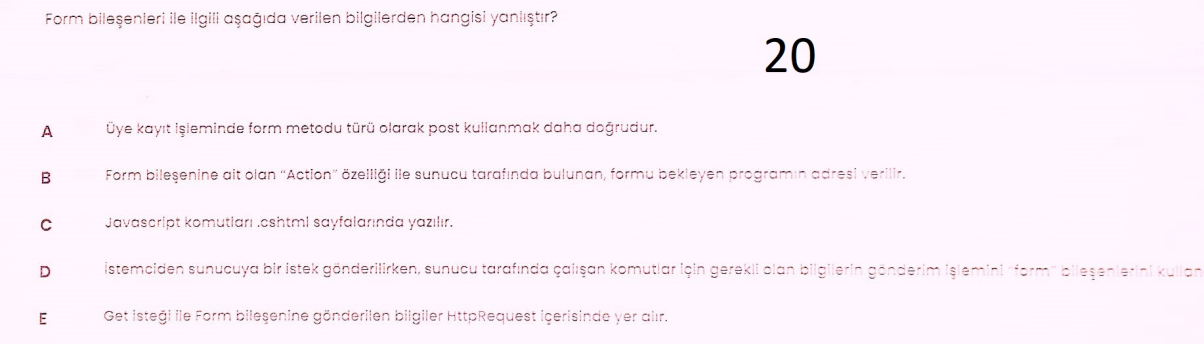
D var kayit= db.Ders.First(x=>x. model.ID);

E var kayit= db.Ders. SingleOrDefault(model.ID);

C



D



Görünüm (view) dosyalarına yazılan Razor Görünüm Motoru (Razor View Engine) komutlarının başına ne konur?

@

Bir kontrolcüden (controller), görünüm (view) dosyalarına bilgi aktarmak için kullanılan dinamik nesneye ne ad verilir?

ViewBag

ViewBag hangi tipte bir nesnedir?

Dinamik

Projenizde bootstrap vb. kütüphaneleri eklemek ve yönetmek için kullandığımız Visual Studio ile entegre gelen paket yöneticinin adı?

nuget

Lambda ifadesinin (lambda expression) sembolü aşağıdakilerden hangisidir?

=>

El ile yazılan model ve veritabanı bağlam (dbcontext) dosyaları yardımı ile veritabanı, tablolar ve arasındaki tüm ilişkilerin entity framework tarafından otomatik olarak oluşturulmasını sağlayan geliştirme yaklaşımına ne ad verilir?

Önce Kod (Code First)

Asp.Net MVC'de veri doğrulama için kullanılan ve köşeli parantezler içerisine yazılan ifadelere (örn, [Required]) ne ad verilir?

Veri açıklama notu (Data annotation)

ASP:NET MVC çatısında görünümlerin (view) kaynak kodların saklandığı dosyaların uzantısı nedir?

cshtml

HTML'de yorum satırı nasıl oluşturulur?

<!-- -->

Güçlü tipli görünüm (strong type view) dosyasındaki kaynak kodlar aşağıdaki anahtar kelimelerden hangisi ile başlar?

Model

.Net MVC uygulamasında web tarayıcıları adres çubuğuna adres girildiğinde çağırılan controller action metotlarına verilen genel isim nedir?

Aksiyon Metodu

MVC'de Razor'un kullanım amacı nedir?

View'de Sunucu taraflı kod yazabilmek

LINQ için aşağıdakilerden hangisi yanlıştır?

Web Api'de kullanılmaz.

ASP.Net MVC de varsayılan route sıralaması aşağıdakilerden hangisidir?

Controller / Action / Parameter

ModelState.IsValid nedir?

Gerekli doğrulamalardan geçtiği takdirde (boş geçilemez, doğru format vb.) true olacak ve içerisinde yazacağımız işlemleri yapacak. Doğrulama olmadığı takdirde aynı sayfa ekrana tekrar yazdırılacak.

Aşağıdakilerden hangisi form güvenliğinin sağlanması için gizli bir anahtar oluşturur?
@Html.AntiForgeryToken()

Bir MVC uygulamasında istemciden gelen ilk istek nereye düşer?

Controller

ASP.Net Web Api için aşağıdaki hangi ifadeler yanlıştır?

View kullanılır.

Bir ASP.Net MVC sayfası için aşağıdakilerden hangisi yanlış bir ifadedir?

Model zorunludur.

Aşağıdaki ifadelerden hangisi yanlıştır? (CF: Code First, DF: Database First)

CF ve DF yaklaşımlarının her ikisinde de bir DbContext vardır.

Aşağıdakilerden hangileri ASP.NET MVC Partial View için doğrudur?

1. **Bakımı kolaydır.**
2. **Yeniden kullanılabilirliği arttırır.**
3. Sayfa boyutunu azaltır.

Bir layout sayfasının tüm sayfalarda ön tanımlı olması için aşağıdaki dosyaların hangisinde tanımlama yapılmalıdır?

/Views/_ViewStart.cshtml

Çoklu dil için kullanılan dosyaların uzantısı nedir?

.resx

HTML aksiyon metoduna bağlantı sağlayan Razor komutu

Html.ActionLink

HTML biçimlendirme yapılarını döndüren Razor komutu

Html.DisplayFor

ModelState hata kontrolü

If (ModelState.IsValid)

Bellekte duran değişikliğin veritabanına kaydedilmesi

SaveChanges()

Bir görünüm (view) içerisinde yorum satırı oluşturmak için razor sembolleri aşağıdakilerden hangisidir?

@* *@