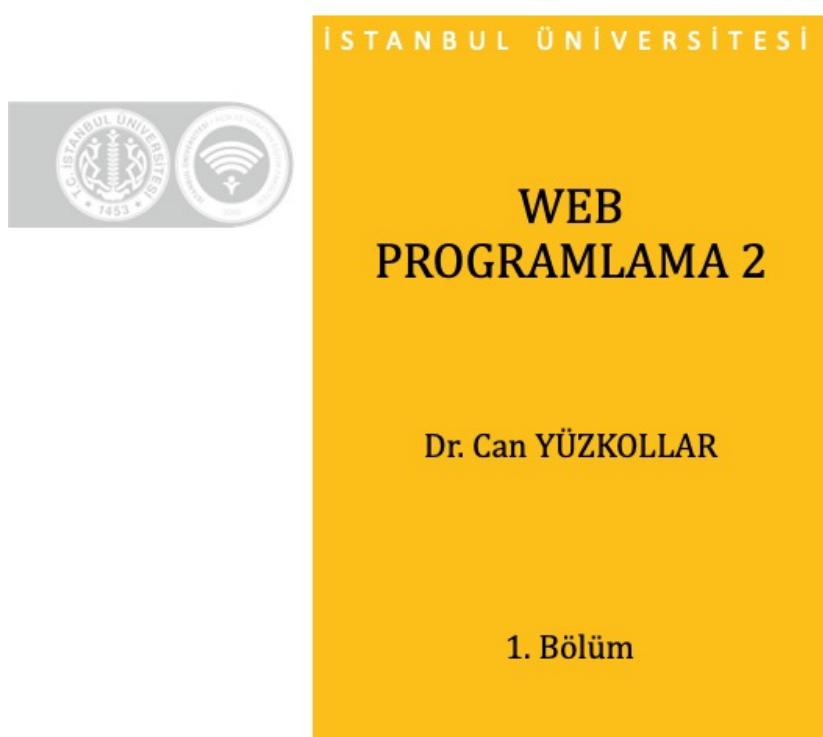


WEB PROGRAMLAMA II

SUNUM DOSYALARI

DR. ÖĞR. ÜYESİ CAN YÜZKOLLAR

İSTANBUL ÜNİVERSİTESİ AÇIK VE UZAKTAN EĞİTİM FAKÜLTESİ



SUNUM PLANI

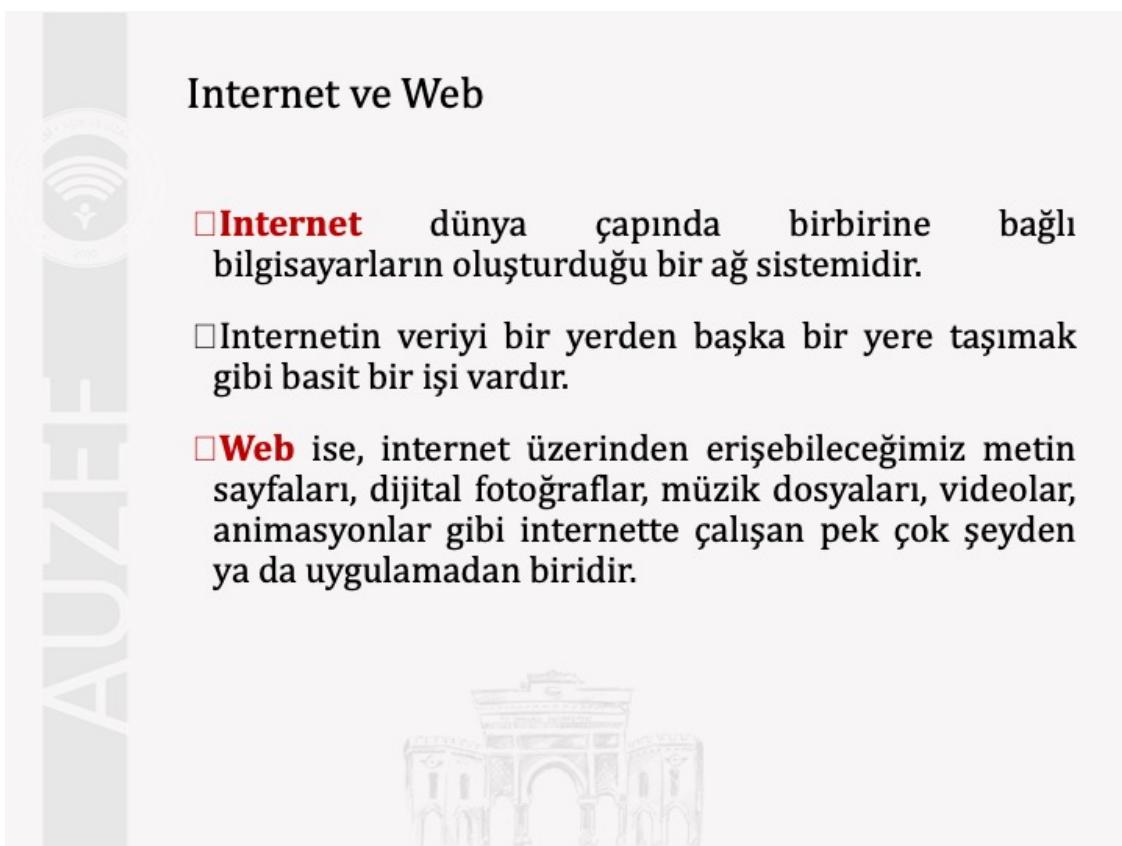
Internet –Web Temel Kavramlar

1. Web Nedir ?
2. Internet Nedir ?
3. Internet Tarihçesi ?
4. Framework Nedir ?
5. Backend, Frontend ?
6. .Net, .Net Core, Asp. Net Core ?
7. Yazılım Mimari Modelleri Nedir ?



Internet ve Web

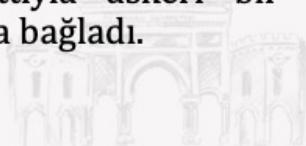
- **Internet** dünya çapında birbirine bağlı bilgisayarların oluşturduğu bir ağ sistemidir.
- Internetin veriyi bir yerden başka bir yere taşımak gibi basit bir işi vardır.
- **Web** ise, internet üzerinden erişebileceğimiz metin sayfaları, dijital fotoğraflar, müzik dosyaları, videolar, animasyonlar gibi internette çalışan pek çok şeyden ya da uygulamadan biridir.





Internetin Tarihçesi

- 1962 yılında, MIT'deki Leonard Kleinrock, verilerin paketler olarak aktarılmasını sağlayan teknoloji (packet switching theory), üzerine ilk makaleyi yayınladı.
- Aynı zamanda, J.C.R. MIT'den Licklider, insanların herhangi bir yerden bilgiye erişmelerini sağlayacak olan "Galaktik Ağ"ı tanımlayan bir dizi not yazdı.
- 1965 yılında, MIT araştırmacısı Lawrence G. Roberts, Thomas Merrill ile birlikte, Massachusetts'deki TX-2 bilgisayarını California'daki Q-32'ye düşük hızlı bir telefon hattıyla askeri bir proje olan ARPANET kapsamında bağladı.



Internetin Tarihçesi

- 1972'de Ray Tomlinson, ARPANET için e-posta oluşturdu ve e-posta adresleri için "@" simbolünü kullanmaya başladı.
- Yine 1972 yılında, ARPANET farklı ülkelerde bulunan 23 düğümle birlikte dünyaya yayıldı ve böylece bu **internet** olarak tanındı.
- Zamanla, FTP (Dosya Aktarım Protokolü) TCP / IP protokolleri, DNS, www, tarayıcılar, betik dilleri vb. gibi yeni teknolojilerin icadıyla internet, web üzerinden bilgi paylaşmak ve erişmek için bir ortam sağladı ve web doğdu.





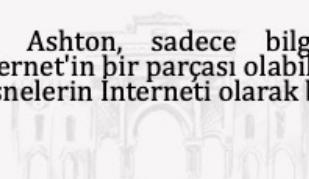
Internetin Tarihçesi

- Bilgisayarların 1960-1980 yıllarında birbirleriyle bilgi alışverişinde bulunmaları çok nadir.
- 1980: IBM küçük işletmeler için kişisel bir bilgisayar tanıttı. Microsoft, tüm "IBM uyumlu" bilgisayarların aynı programları çalıştırmasına izin veren, Windows adlı yazılım geliştirdi.
- 1980'lerde, Tim Berners-Lee CERN'de farklı bilgisayarları kullanan kişilerin araştırmalarını paylaşmanın kolay bir yolunu aradı.
- Berners-Lee, ASCII (American Standard Code for Information Interchange),'yi bilgiyaların haberleşmesinde bir kural olarak kullandı ve bunu HTTP (HyperText Transfer Protocol) olarak adlandırdı. (örnek curl -IL www.google.com)
- Berners-Lee ikinci kural olarak, tüm CERN bilgisayarlarının HTML (HyperText Markup Language) adlı ortak bir dilde yazılmış dosyaları değişim tokus etmesini sağladı. Planladığı HTML ile etiked adı verilen özel kodlarla metinleri yapılandırdı.



Internetin Tarihçesi

- **1993:** Marc Andreessen, daha sonra Netscape ve Mozilla'ya dönüsen ilk kullanıcı dostu web tarayıcısı Mosaic'i yazdı.
- **1994:** İnsanlar kısa sürede hızla büyüyen World Wide Web'de gezinmek için yardıma ihtiyaç duyduklarını anladılar. WebCrawler ve Yahoo gibi arama motorları geliştirildi.
- **1995:** Amazon, eBay'in kurulmasıyla beraber e-ticaret düzgün bir şekilde başladı.
- **1996:** ICQ ile birlikte Internet'teki ilk kullanıcı dostu anında mesajlaşma sistemi geliştirildi.
- **1997:** Jorn Barger ilk blogu yayınladı (web günlüğü).
- **1998:** Larry Page ve Sergey Brin, Google'da bir arama motoru geliştirdi.
- **1999:** Kevin Ashton, sadece bilgisayarların değil, gündelik nesnelerin Internet'in bir parçası olabileceği fikrini ortaya koydu. Bu fikir şimdi Nesnelerin Interneti olarak bilinir.





Internetin Tarihçesi

- **2013:** Yapılan bir araştırmaya göre Amerika'daki yetişkinlerin %50'si bankacılık işlemlerini online olarak yaptıkları belirlendi.
- **2015:** İlk olarak fotoğraf paylaşma sitesi olarak yayınlanan Instagram, Twitter'ı geride bırakarak 400 milyon kullanıcıya ulaştı.
- **2016:** Google, ses ile akınfleşen bir kişisel asistan programı olan Google Asistan'ı yayınladı. Böylece, Google Amazon'un Alexa'sı, Apple'in Siri'si ve Microsoft'un Cortana'sına katıldı.



WEB 1.0 -> WEB 2.0 -> WEB 3.0

- Web 1.0 Kullanıcı etkileşimin olmadığı sadece statik sayfalardan oluşan, World Wide Web'in evriminin ilk aşamasını ifade eden yapıya verilen addır.
- Web 2.0 İkinci nesil web hizmetlerini barındıran, içeriklerin kullanıcılar tarafından oluşturulduğu, katılımcı web - sosyal web olarak da adlandırılan yapıdır. Web 2.0 ile AJAX ve JavaScript frameworkleri çok popüler bir araç haline gelmiştir.
- Web 3.0 Web'in bir veritabanına dönüştürülmesini içerir. Web'de yer alan uygulamaların farklı verilere bağlam sağladığı, bu verilerin farklı uygulamalar tarafından anlaşılıp yorumlandığı bir yapıdır.
- Web 4.0 Simbiyotik ağ olarak da adlandırılır. Bu ağın amacı, insanlar ve yapay zeka kullanan makineler arasındaki etkileşimin arttırılmasıdır.



Web 1.0	Web 2.0	Web 3.0
Çoğunlukla Salt Okunur	Çoğunlukla okuma yazma	Taşınabilir ve Kişisel
Ticari Odaklı	Topluluk Odaklı	Bireysellik odaklı
Web Formları	Web Uygulamaları	Akıllı Uygulamalar
Dizinler	Etiketler	Kullanıcı Davranışları
İçerik Sahipliği	İçerik Paylaşımı	İçerik Birleştirme-Anlamlaştırma
Britannica Online	Wikipedia	The Semantic Web
HTML/Portals	XML / RSS	RDF / RDFS / OWL

WEB 1.0 -> WEB 2.0 -> WEB 3.0

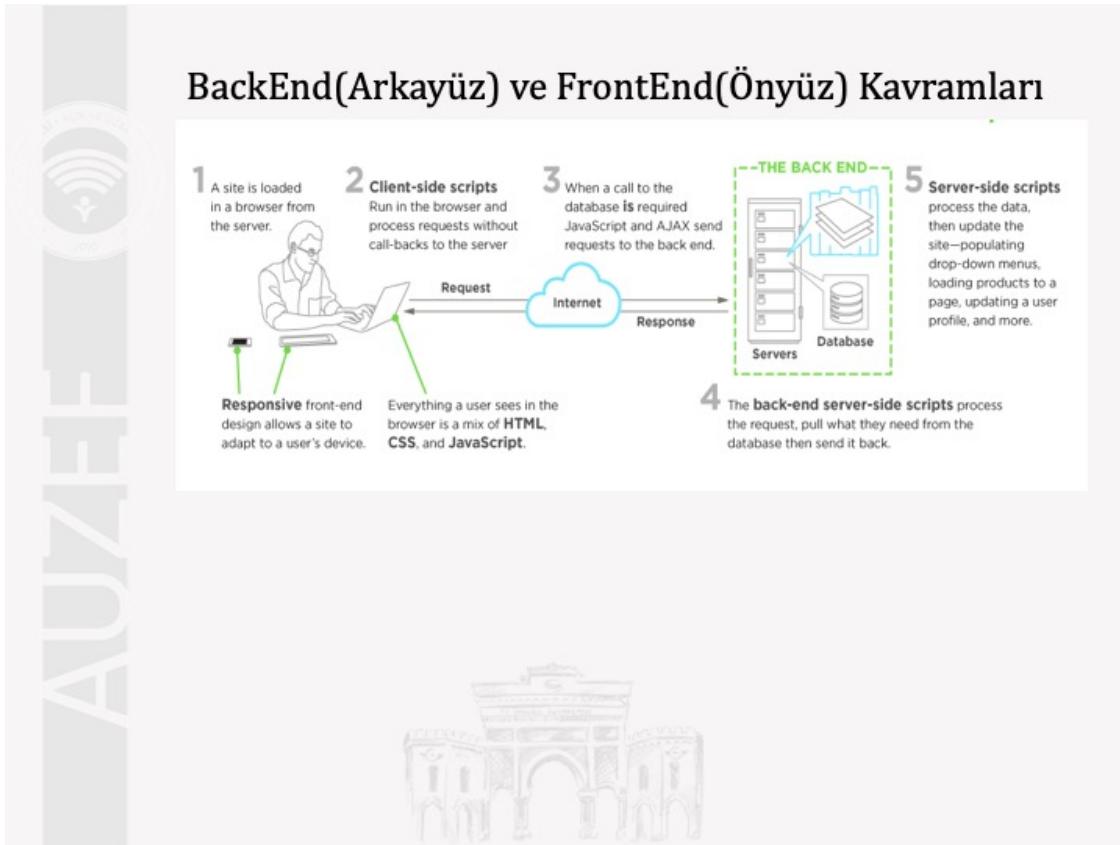
Web Framework

- Frameworkler, web geliştiricilerine web uygulamaları, web hizmetleri ve web sitelerini kolayca oluşturma ve yönetme yeteneği verir.
- Uygulamaların kolay ve hızlı oluşturmasına olanak sağlarlar
- Geliştiriciler tekrarlayan koddan sakınmış olurlar ve verimli bir şekilde projelerini üretirler.
- Çeşitli veritabanlarına kolay bir şekilde bağlanmaya olanak sağlayan entegrasyon özelliği mevcuttur



AUZEF
Eğitim
Mühendislik
Fakültesi





FrontEnd (İstemci Tarafı) Programlama

- Bootstrap ve SemanticUI gibi çerçeveler görsel tasarıma odaklanıp sürdürülebilir Html ve Css üzerine yoğunlaşırken, Vue, React ve Angular gibiler ise, verileri manipüle edip veri akşını yapılandırmaya çalışırlar.
- İstemci tarafında yazılımcılar tarafından tercih edilen jQuery, Emberjs, Backbonejs gibi çerçeveler de mevcuttur

```
<html>
  <head>
    <title>HTML Örnek</title>
  </head>

  <body>
    <h3>Paragraf Başlığı</h3>
    <p>Paragraf Metni</p>
  </body>
</html>
```

```
body{
  background: #fff;
  color: #333;
  font: 14px / 28px Arial;
  font-family: 'Raleway';
}

h1{
  font-size: 52px;
  letter-spacing: 4px;
  margin-bottom: 20px;
}
```

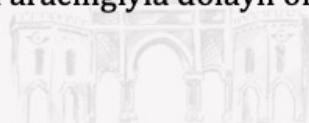
```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <script>
      var a = 1;
      var b=2;

      function f(a) {
        console.log(a + b);
      }
    </script>
  </body>
</html>
```



BackEnd(Sunucu Taraflı) Programalama

- Bir istemci sunucundan bir içerik talep ettiğinde, uygun programlama dilinin yardımıyla web sunucusu tarafından uygun komutlar işletilir.
- Sunucu tarafında sık yapılan işlemlerden biri, ihtiyaç duyulan verileri bir veritabanından çekmek ve bunları web projesinde uygun biçimde kullanmaktır. Kullanıcı bir web uygulamasına istekte bulunduğuunda sunucu tarafından işletilen kodlar neticesinde kullanıcıya temel olarak HTML,CSS, Javascript şeklinde bir dönüş sağlanmaktadır.
- Yani kullanıcı sunucu üzerinde işletilen kodları göremez ve onlara erişemez.
- Kullancılar; arka yüz tasarımcıları tarafından geliştirilen parçalara ve özelliklere, kullanıcılar tarafından bir ön yüz uygulaması aracılığıyla dolaylı olarak erişilir.



BackEnd(Sunucu Taraflı) Web Çerçeveleri

- Sunucu taraflı kullanılan web çerçevelerine .Net, Net Core, Laravel, Django, Spring, CodeIgniter örnek olarak verilebilir.
- Bu web çerçevelerin her biri Asp.Net Core (C#), Flask (Python), Laravel (PHP), Spring (Java) vb gibi bir programlama dili ile beraber kullanılır

Özellikleri

- URL yönlendirme
- Form yönetimi ve doğrulama
- Object-relational mapping (ORM) aracılığıyla veritabanı bağlantısı yapılandırma
- CSRF, XSS, SQL injection gibi sık görülen kötü niyetli saldırırlara karşı web güvenliği
- Oturum Yönetimi





.Net Framework

- .Net framework'u (yazılım çatısı-çerçevesi) microsoft tarafından geliştirilmiş bir yazılım geliştirme platformudur.
- .Net tarafından desteklenen tek bir programlama dili ile masaüstü uygulamaları, web uygulamaları, web servisleri ve mobil uygulamalar tek bir çatı altında gerçekleştirilebilmektedir.
- İşlemleri kolaylaştıran önceden yazılmış birçok faydalı temel türler ve sınıflar bulunmaktadır.
- İlk sürümü Şubat 2002'de piyasaya sürülen .Net platformu çeşitli iyileştirmeler ve yenilikler geçirerek 2019 yılına kadar farklı versiyonlar ile karşımıza çıkmış, 2019 yılında ise son sürümü olan 4.8 versiyonu yayımlanmıştır.



.Net Core

- .NET Core, Microsoft tarafından geliştirilen ücretsiz, açık kaynaklı ve platform bağımsız (Windows, macOS ve Linux işletim sistemlerinde çalışabilen) bir yapıdadır.
- .NET Framework'ün yeni bir sürümüdür
- .NET Core, mobil, masaüstü, web, bulut, IoT, makine öğrenimi, mikro hizmetler, oyun vb. gibi farklı türde uygulamalar oluşturmak için kullanılabilir.
- ilk sürümünden sonra Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1 sürümleri yayımlanmıştır. Bu sürümden sonra Net Framework versiyonları ile karıştırılmaması adına yeni sürüm Net 5 adını almıştır



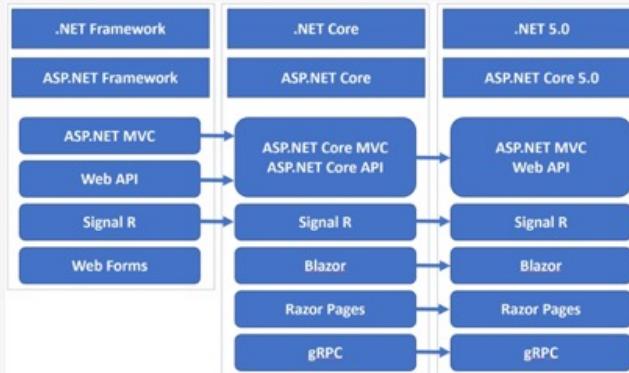
.Net Core Avantajları

- Çapraz Platform (Cross-Platform)
- Açık Kaynak Kod Esnekliği
- Performans
- Mikroservis Mimarisi Uyumu



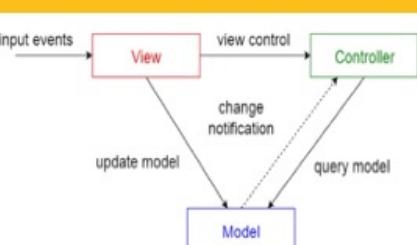
Asp.Net Core

- ASP.NET, microsoft tarafından .NET Framework'un bir parçası olarak 2002 yılında yayıldı.
- .NET Core'da iki farklı programlama modeli (ASP.NET Core MVC ve API) birleştirilmiş olarak sunulmaktadır.



Software Architectural Patterns (Yazılım Mimari Modelleri-Desenleri)

- Mimari model, çeşitli sistemlerin tasarım yapılarını ve yeniden kullanılabilecekleri yazılım öğelerini yakalar.
- Belirli bir bağlamda yazılım mimarisinde yaygın olarak ortaya çıkan bir soruna genel, yeniden kullanılabılır bir çözümüdür.
- Mimari desenler, yazılım tasarım desenine benzer ancak daha geniş bir kapsamı vardır.
- Yazılım mimarisi, yazılım programının yapısal bileşimini ve öğeler arasındaki etkileşimleri açıklar. Model ayrıca alt sistemler arasındaki ilişkileri düzenlemek için kuralları ve yönergeleri de açıklar.

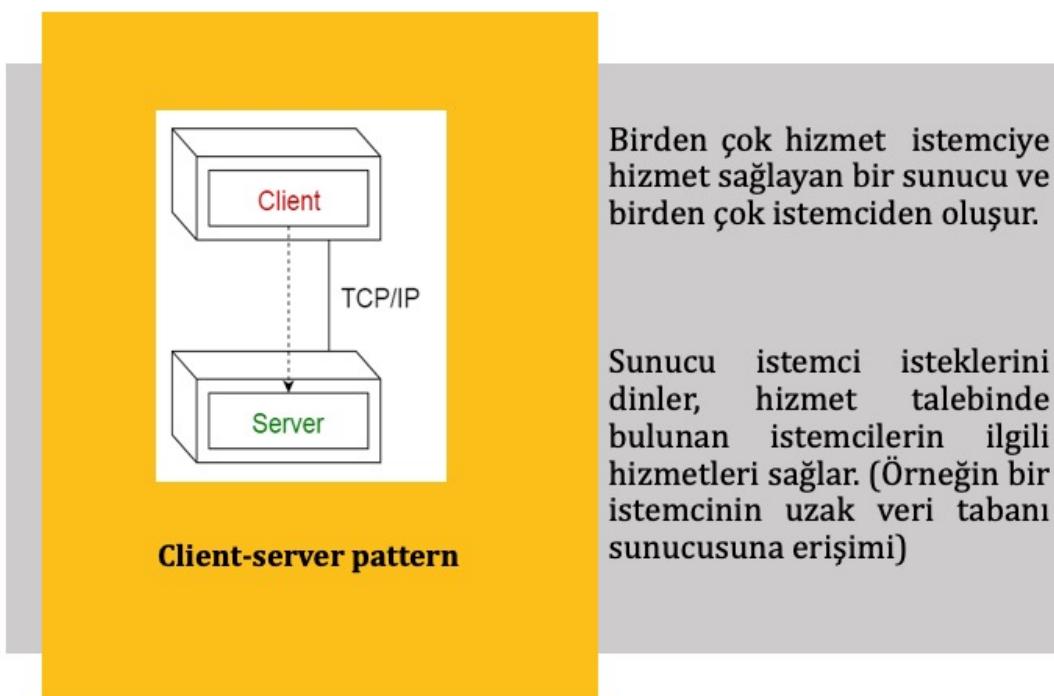
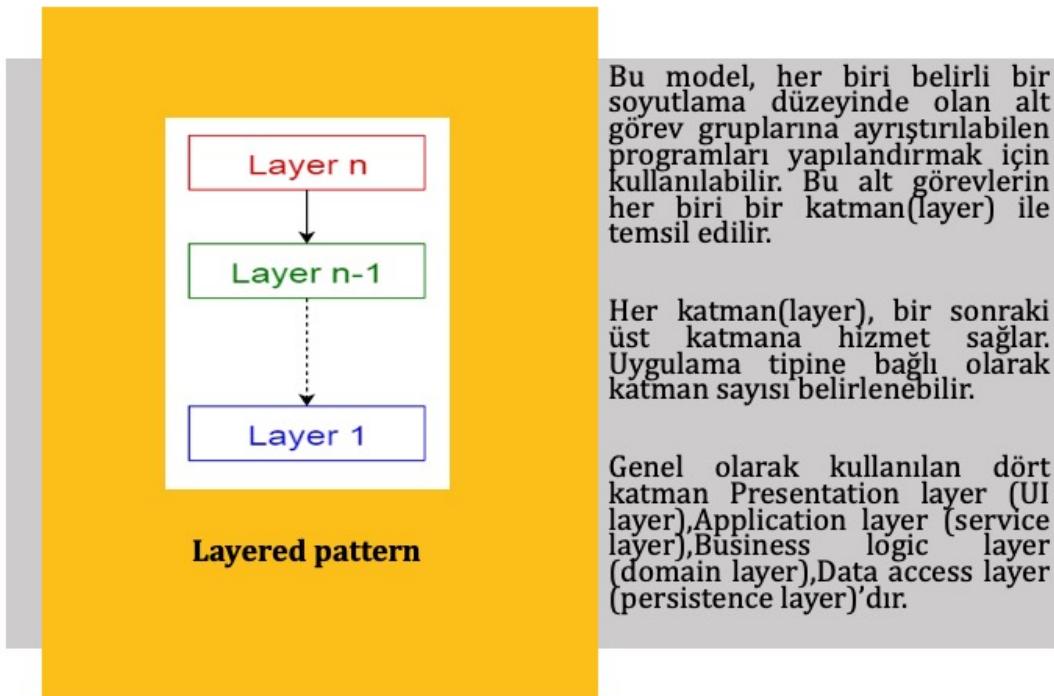


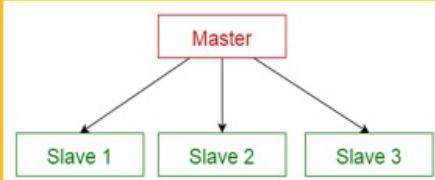
Model-view-controller(MVC)

MVC modeline ait mimari bileşenler, bir uygulamanın geliştirme aşamasındaki farklı yönlerini ele almak için tasarlanmıştır.

Bu model uygulamayı model, view(görünüm) ve controller (denetleyici) olmak üzere 3 farklı mantıksal bölüme ayırrı.

Sunum katmanını iş mantığından ayırmaya yarar. Günümüzde pek çok mobil ve web uygulama tarafından kullanılmaktadır.

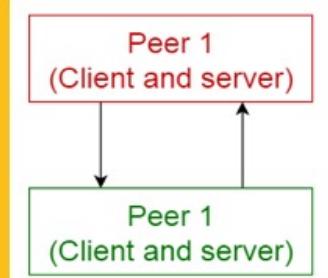




Master-slave pattern

Master-Slave modeli genellikle aynı problemin birçok örneğinin çözülmesi gereken çok iş parçacıklı uygulamalar için kullanılır.

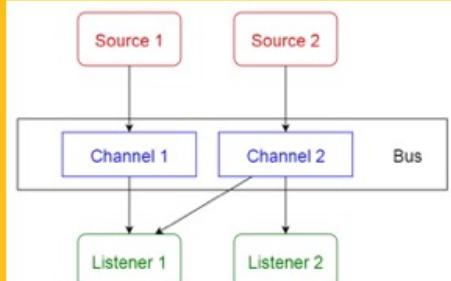
Master bileşen, işi tüm slave bileşenler arasında dağıtır bu bileşenlerden gelen sonuçları toplayarak nihai sonucu hesaplar.



Peer-to-peer pattern

Geleneksel olarak istemcinin istek gönderip sunucun bu isteği karşılaması yaklaşımından farklı olarak eş olarak adlandırılan her bir sistem, ağır parçası olan diğer eşlerden hizmet alır ve verir.

Bu modele örnek olarak Domain Name System (DNS), BitTorrent verilebilir.

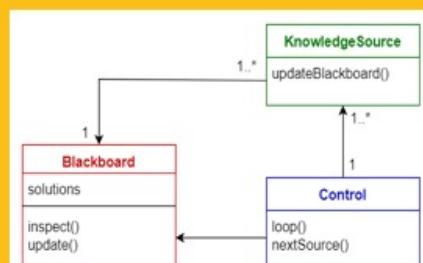
**Event-bus pattern**

Olay kaynağı, olay dinleyicisi, kanal ve olay veriyolu olmak üzere 4 ana bileşeni vardır.

Kaynaklar, olay veri yolundaki belirli kanallara mesajlar yayınlar.

Dinleyiciler, daha önce abone oldukları bir kanalda yayınlanan mesajlar hakkında bilgilendirilir.

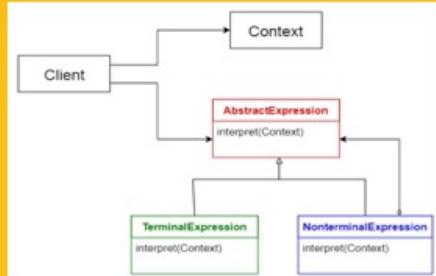
Mesajların üretilmesi ve bildirilmesi asenkrondur, tüm olay dinleyicileri mesajı alana kadar beklemez.

**Blackboard pattern**

Blackboard mimarisi, ortak bir veri yapısı üzerinde işbirliği içinde çalışan bağımsız programlardan oluşur.

Burada her program işbirliği içinde, birbirinden bağımsız olarak genel bir görevi çözmek için uzmanlaşmıştır.

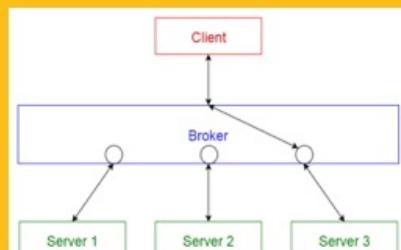
Birbirlerini çağrırmak yerine sistemin belirlediği yön esas alınarak ilerleme kaydedilir.

**Interpreter pattern**

Bu model, özel bir dilde yazılmış programları yorumlayan bir bileşen tasarlamak için kullanılır.

Temel olarak, belirli bir dilde yazılmış cümleler veya ifadeler olarak bilinen program satırlarının nasıl değerlendirileceğini belirtir.

Temel fikir, dilin her sembolü için bir sınıfı sahip olmaktadır. Uzman sistemler gibi kural tabanlı sistemler, javascript gibi web script diller bu modele örnektir.

**Broker pattern**

Broker modeli, uzaktan hizmet çağrılarıyla etkileşime giren ayrılmış bileşenlerle dağıtılmış yazılım sistemlerini yapılandırmak için kullanılabilir.

Broker bileşeni, bileşenler arasındaki iletişim koordinasyonundan sorumludur.

ÖZET

Dünya çapında birbirine bağlı cihazların oluşturduğu küresel bir ağ sistemi olan internetin veriyi bir yerden başka bir yere taşımak gibi basit bir görevi vardır. World Wide Web (www) olarak da adlandırılan Web, internet üzerinden erişebileceğimiz metin sayfaları, dijital fotoğraflar, müzik dosyaları, videolar ve animasyonlar gibi internet üzerinden erişilen bilgi topluluğudur.

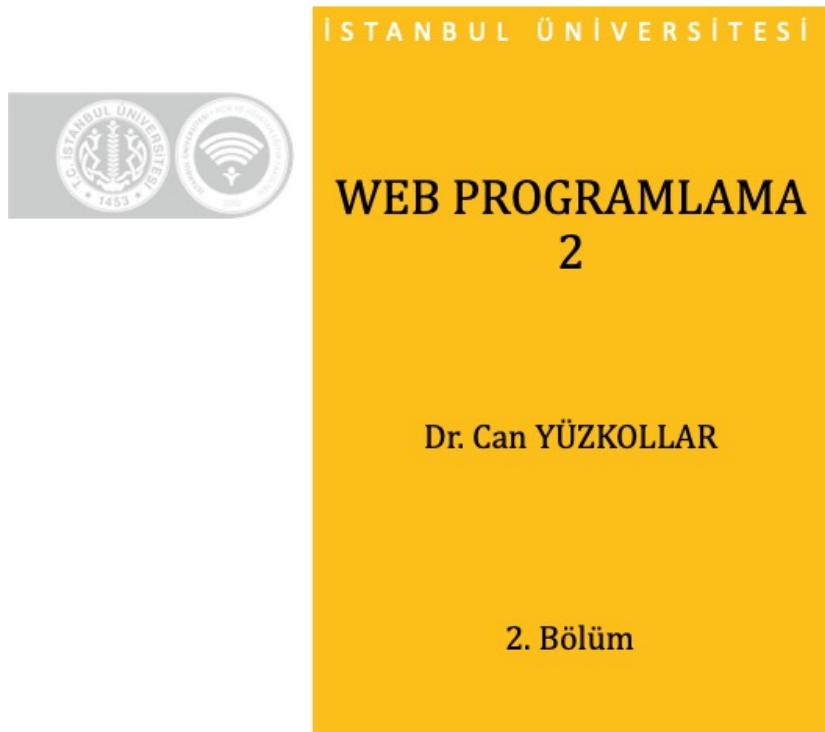
Web uygulamalarından bekleneler ile beraber ihtiyacı karşılayacak teknolojinin karmaşılığı da arttıkından, çerçeveler web geliştirmenin önemli bir parçası haline gelmiştir. Web çerçevelerinin çoğu açık kaynaklı ve ücretsizdir. Uygulamaların kolay ve hızlı oluşturmamasına olanak sağlarlar. Bu sayede geliştiriciler çok sayıda tekrarlayan koddan sakınmış olurlar.

Ön yüz ve Arka yüz, web geliştirmede kullanılan en popüler iki terimdir. Bir web uygulamasının bir arka yüzü(sunucu tarafı) ve bir ön yüzü (istemci tarafı) vardır. Web projelerinin işlevsellliğini geliştirmek için her iki taraf da diğeriley etkin bir şekilde iletişim kurmalı ve çalışmalıdır.

Web uygulamaları geliştirme için .Net Core'un bir parçası olan ASP.NET, piyasada bulunan en tutarlı, kararlı ve zengin özelliklere sahip çerçevelerden biridir.

Yazılım mimarisi, yazılım programının yapısal bileşimini ve öğeler arasındaki etkileşimleri açıklar. Bu noktada MVC modeli günümüzde pek çok mobil ve web uygulama tarafından kullanılmaktadır.





MVC YAPISI VE ASP.NET CORE İLE MVC KULLANIMI

SUNUM PLANI

- 1. Visual Studio Kurulumu
- 2. Asp.Net Core Proje Yapısı
- 3. Model-View Controller Kavamları
- 4. Asp.Net Core ile MVC Kullanımı



AUZEF

Visual Studio

Microsoft, ASP.NET Core uygulamaları geliştirmek için Visual Studio ve Visual Studio Code çeşitli araçlar sunar.

Visual Studio, GUI (Grafik Kullanıcı Arayüzü), web uygulamaları, konsol uygulamaları, mobil uygulamalar, bulut ve web hizmetleri vb. geliştirmek için Entegre Geliştirme Ortamıdır (IDE-Integrated Development Environment).

Visual Studio Code ise Visual Studio'nun gelişmiş özelliklerine sahip olmamakla birlikte ona daha hafif bir alternatiftir.

AUZEF



Visual Studio Kurulum

Microsoft, öğrenme (ticari olmayan) amacıyla ücretsiz bir sürüm olan Visual Studio Community'i sağlamıştır. Tüm sürümlere ait kurulumu gerçekleştirmek için <https://visualstudio.microsoft.com/tr/downloads/> adresinde yer alan kurulum dosyasını indirmek gereklidir.

Visual Studio 2019	Topluluk	Professional	Enterprise
Sürüm notları >	Güçlü IDE, öğrenciler, açık kaynak katkıda bulunanlar ve bireysel kullanım için ücretsiz	Profesyonel IDE, küçük ekpler için en uygun seçenek	Hangi boyutta olursa olsun tüm ekpler için olğeklenebilir, uçaç uca çözüm
Sürümüleri karşılaştırın > Çevrimiçi yükleme >	Ücretsiz indirin	Ücretsiz deneme	Ücretsiz deneme

Değişiklikler -- Visual Studio Community 2019 -- 16.8.3

İş yükleri Bağımsız bileşenler Dil paketleri Yükleme konumları

Web ve Bulut (4)

ASP.NET ve web geliştirme Docker desteği dahil olmak üzere ASP.NET Core, ASP.NET, HTML/Javascript ve Kapasiteler kullanarak web uygulamaları geliştirmek ve kaynak paylaşmak.

Python geliştirme Python için dizeriye, hata ayıklama, etkileşimi geliştirme ve kaynak yönetimi.

Masaüstü ve Mobil (5)

.NET masaüstü geliştirme .NET Core ve .NET Framework ile C#, Visual Basic ve F# dillerini kullanarak WPF, Windows Forms ve konsol uygulamaları geliştirmek.

Evrensel Windows Platformu geliştirme C#, VB veya isteğe bağlı olarak C++ ile Evrensel Windows Platformuna yönelik uygulamalar oluşturun.

Azure geliştirme .NET Core ve .NET Framework ile bulut uygulamaları geliştirmek ve kaynak paylaşmak için kullanılan Azure Services.

Node.js geliştirme Zaman uyumsuz olay temelli bir JavaScript çalışma zamanı olan Node.js kullanarak déploypenabilir ağ uygulamaları geliştirmek.

C++ ile masaüstü geliştirme Terch ettiğiniz MSVC, Clang, CMake veya MSBuild gibi bir aracı kullanarak Windows için modern C++ uygulamaları geliştirmek.

.NET ile Mobil uygulama geliştirme Xamarin kullanarak iOS, Android veya Windows için çoklu platform uygulamaları oluşturun.

Yükleme ayrıntıları

- .NET ile Mobil uygulama geliştirme
- Veri depolama ve işleme
- Veri bilimi ve analitik uygulamalar
- .NET Core çoklu platform geliştirme
 - Ekleni
 - ✓ .NET Core geliştirme aracları
 - ✓ .NET Framework 4.7.2 geliştirme aracları
 - ✓ ASP.NET ve web geliştirme aracları önbellek
 - ✓ IntelliCode
 - İsteğe bağlı
 - ✓ .NET Core 2.1 Çalışma Zamanı (LTS)
 - ✓ Web geliştirme için bulut aracları
 - ✓ .NET profil oluşturma aracları
 - ✓ Developer Analytics Tools
 - ✓ Web Dagitör
 - ✓ Live Share
 - ✓ ML.NET Model Builder (Önizleme)
 - ✓ MSIX Packaging Tools

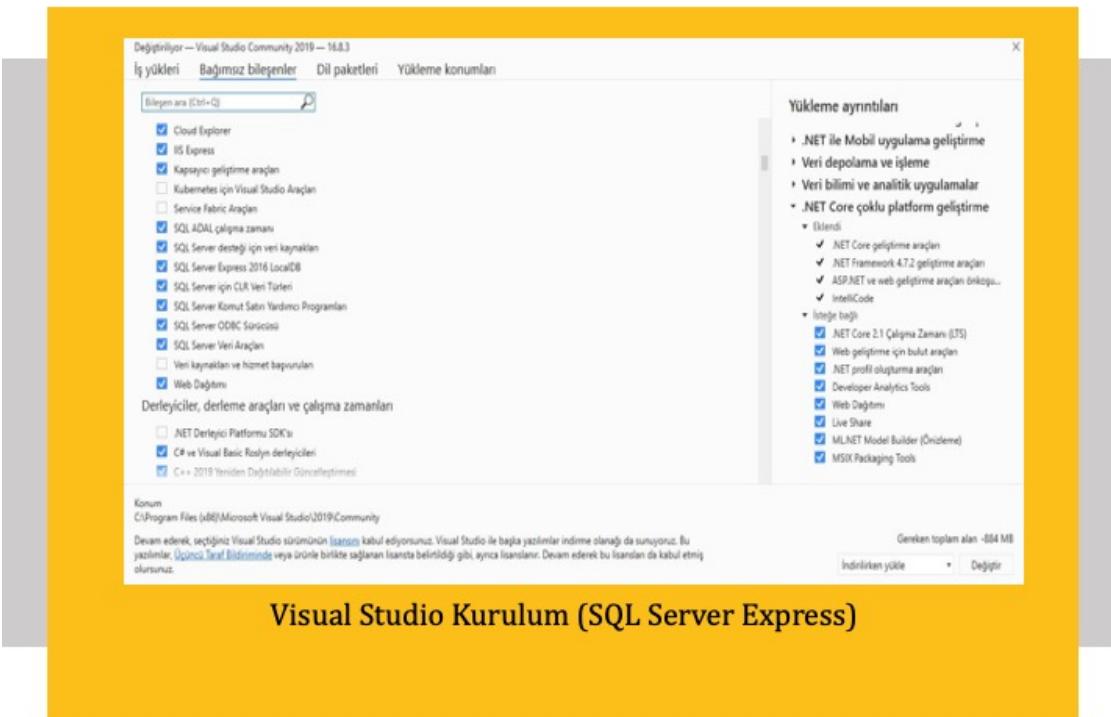
Konum C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

Devam ederek, seçtiğiniz Visual Studio sürümünün [lisansı](#) kabul ediyorsunuz. Visual Studio ile başka yazılımlar indirme izni de sunuyoruz. Bu yazılımlar, [Uzunca Təraf Bildirimində](#) veya onlara birlikte sağlanan lisansla belirtildiği gibi, ayrıca lisanslanır. Devam etmek bu lisandan kabul etmiş olunur.

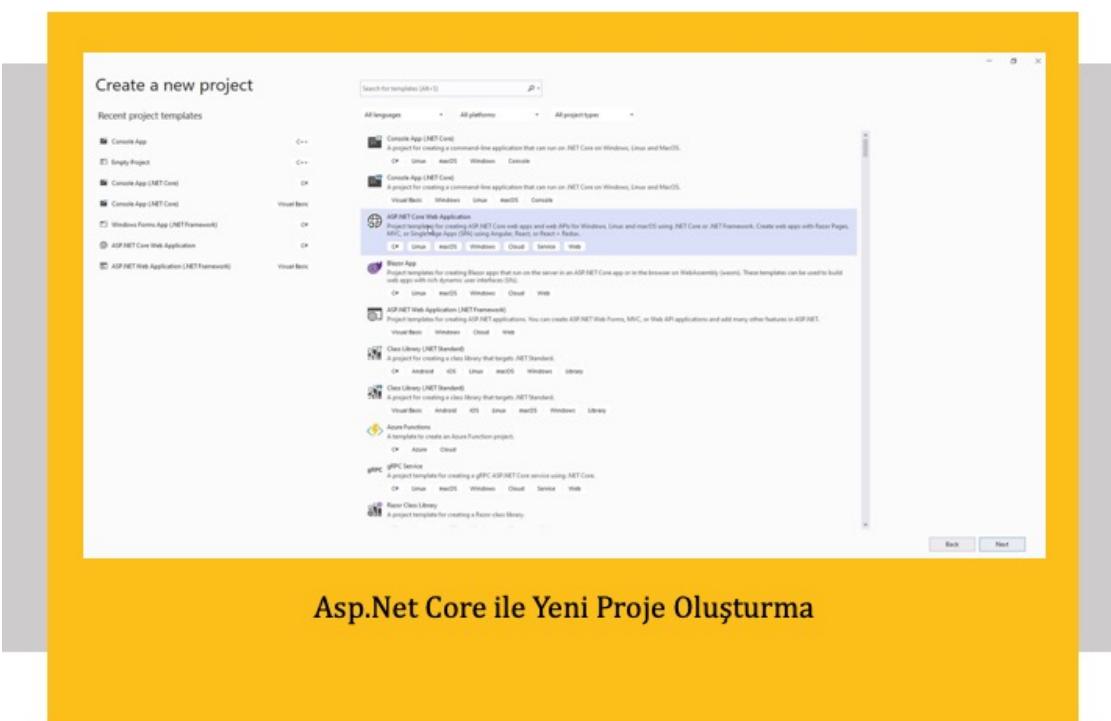
Genelen toplam alan -884 MB

İndirilen yerde | Değerlendir

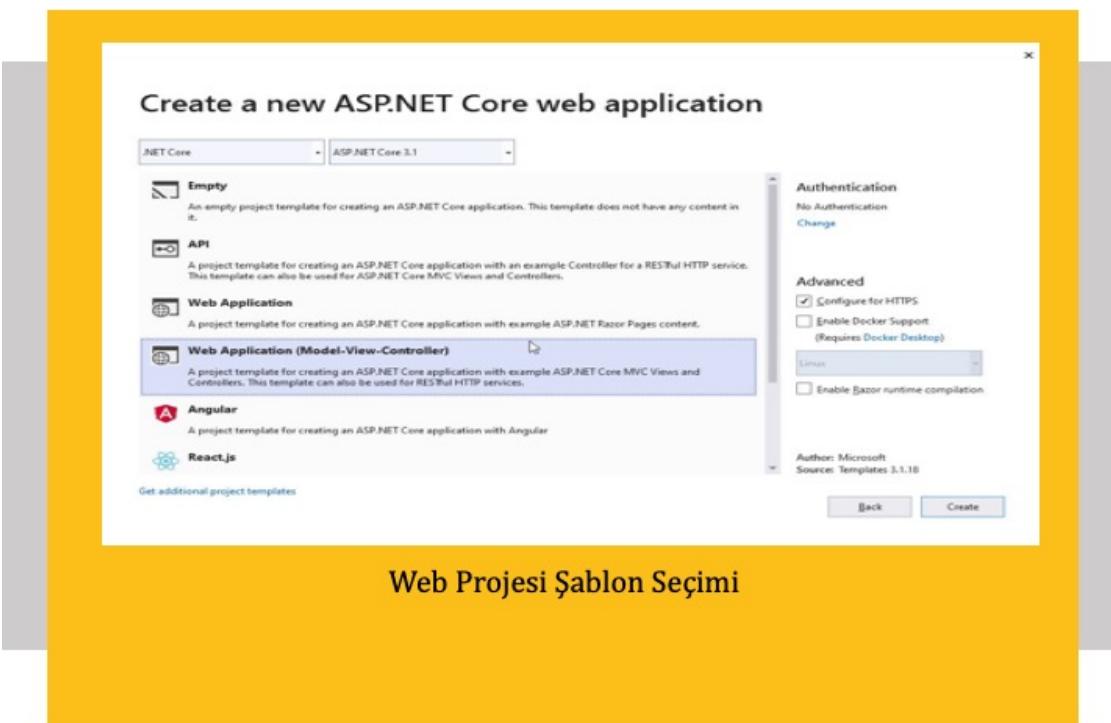
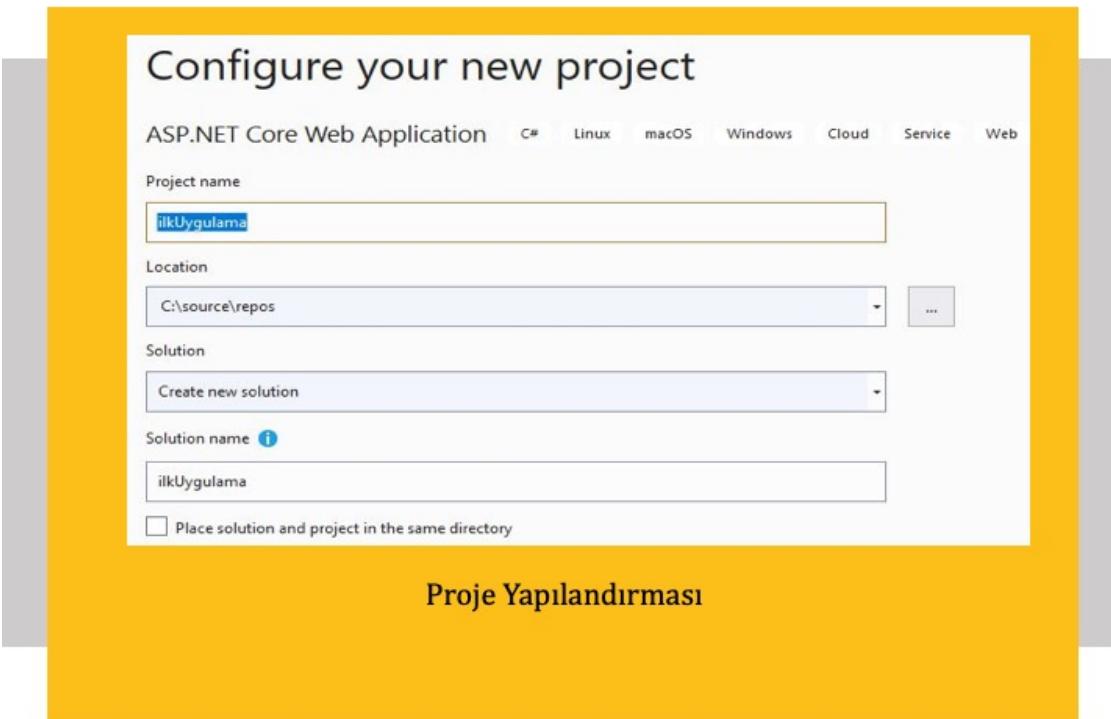
Visual Studio Kurulum

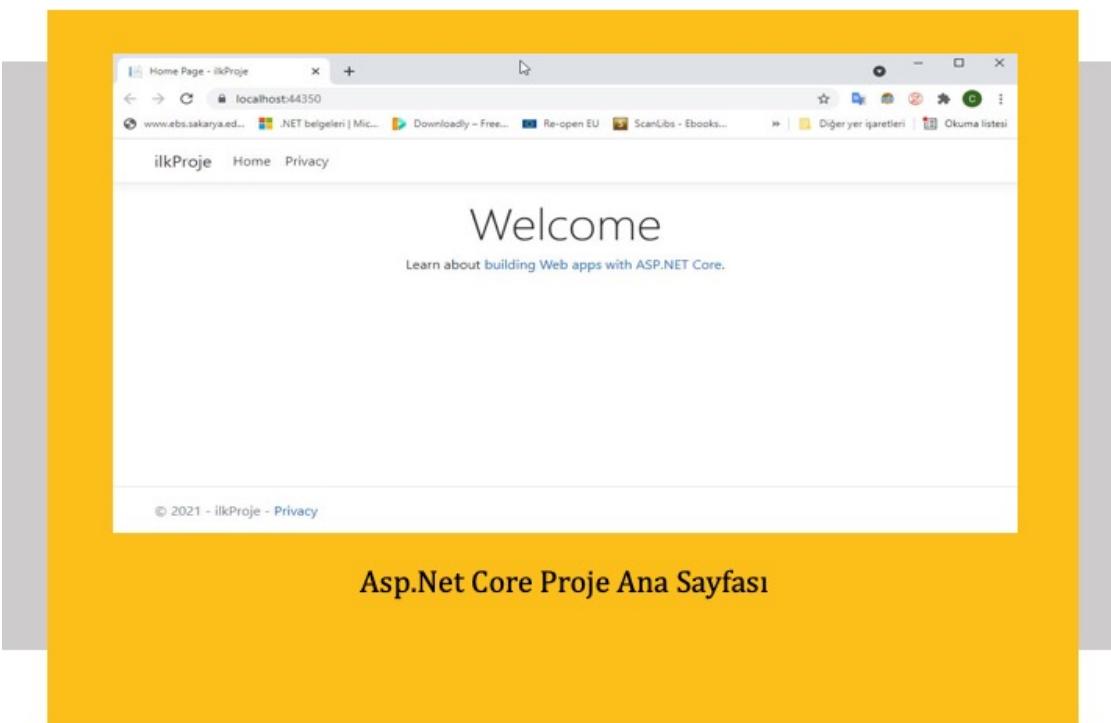
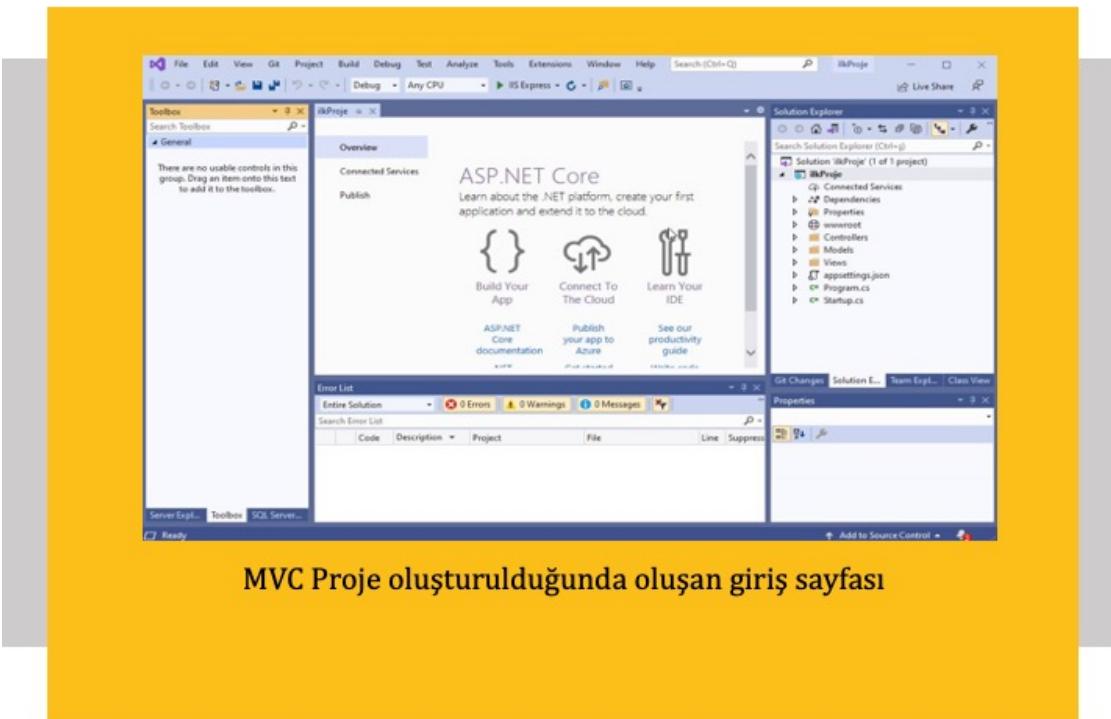


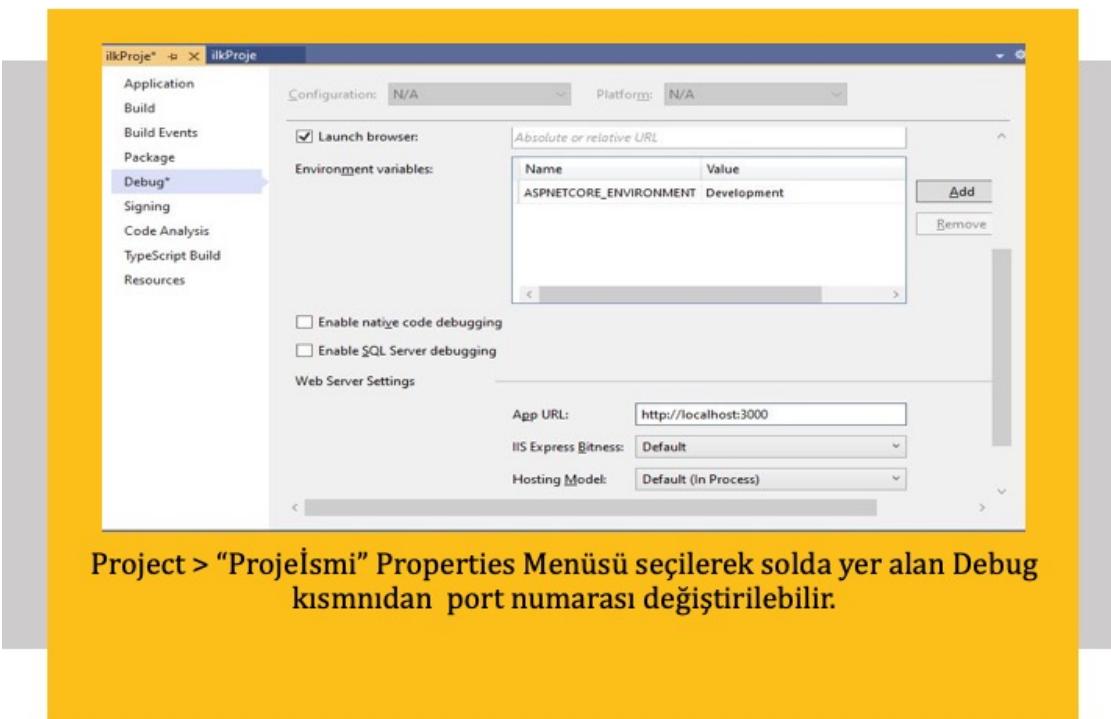
Visual Studio Kurulum (SQL Server Express)



Asp.Net Core ile Yeni Proje Oluşturma







.ASP.Net Core MVC Proje Yapısı

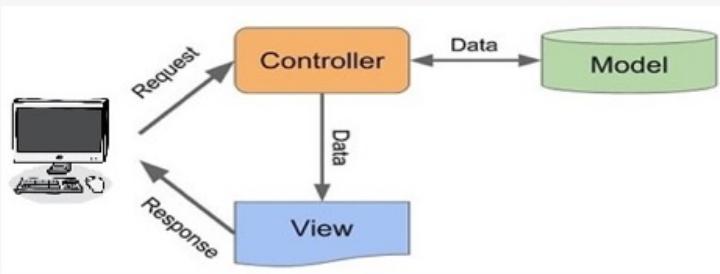
Bir Asp.Net Core Mvc projesi oluşturulduğunda <ProjeAdı>.csproj isiminde bir proje dosyası oluşacaktır. Bu proje dosyası proje klasörleri, NuGet paket referansları vb. ile ayarları içerir. Proje isimi üzerinde sağ tıklayıp Edit Project File seçeneği ile proje dosyasının içeriği görüntülebilir. Örneğin EntityFrameworkCore paketi eklenen proje için bu dosyanın içeriği aşağıdakine benzer şekilde olacaktır.

```

<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="5.0.9" />
  </ItemGroup>
</Project>

```

MVC(Model View Controller)



Model-View-Controller (MVC) mimari modeli, bir uygulamayı model, görünüm(View) ve denetleyici(Controller) olmak üzere üç ana bileşene ayırrı. Sorumlulukları ayrı her bir bileşen birbirini etkilemeden birbirinden bağımsız olarak çalışır. Böylelikle proje boyutu arttıkça projenin yönetimi ve kontrolü daha kolay bir şekilde sağlanır. Aynı zamanda eş zamanlı olarak projenin ilerlemesi de gerçekleşmiş olur.

Model Bileşeni

Model, MVC'de uygulamaya ait verileri, uygulamanın durumunu ve uygulama tarafından gerçekleştirilmesi gereken tüm iş mantığını (business logic) veya işlemleri temsil eder. Proje yapısı ve büyüklüğüne göre tek katman veya birden fazla katmandan da oluşabilir.

View Bileşeni

View, içeriğin kullanıcı arabirimini aracılığıyla sunulmasından sorumludur. MVC'de projenin arayüzleri bu bölümde oluşturulur. .NET kodunu HTML içersine yerleştirmek için Razor View Engine (Razor görünüm motoru) kullanır. İçerisinde minimum mantıksal işlemler yer almmalıdır.

Projenin geliştirildiği yazılım dillerine göre dosya uzantıları da değişebilmektedir. Karmaşıklığı gidermek adına View'larının yer aldığı klasörlerinin hiyerarşisi iyi ayarlanmalıdır. (HTML, Javascript, CSS ve resim dosyalarını aynı klasör içinde barındırmamak vb)



Controller Bileşeni

Controller, kullanıcı etkileşimini yöneten, modelle çalışan ve sonucta oluşturulacak görünümü seçen bileşenlerdir. Bu bileşen kullanıcidan gelen istekleri(request) işler View ile Model arasında bağlantı kurarak kullanıcıya hangi View'in geleceği (response) belirtir.

Kısaca MVC modelinde controller ilk giriş noktasıdır ve hangi model türleriyle çalışılacağını ve hangi görünümün oluşturulacağını seçmekten sorumludur.





Routing

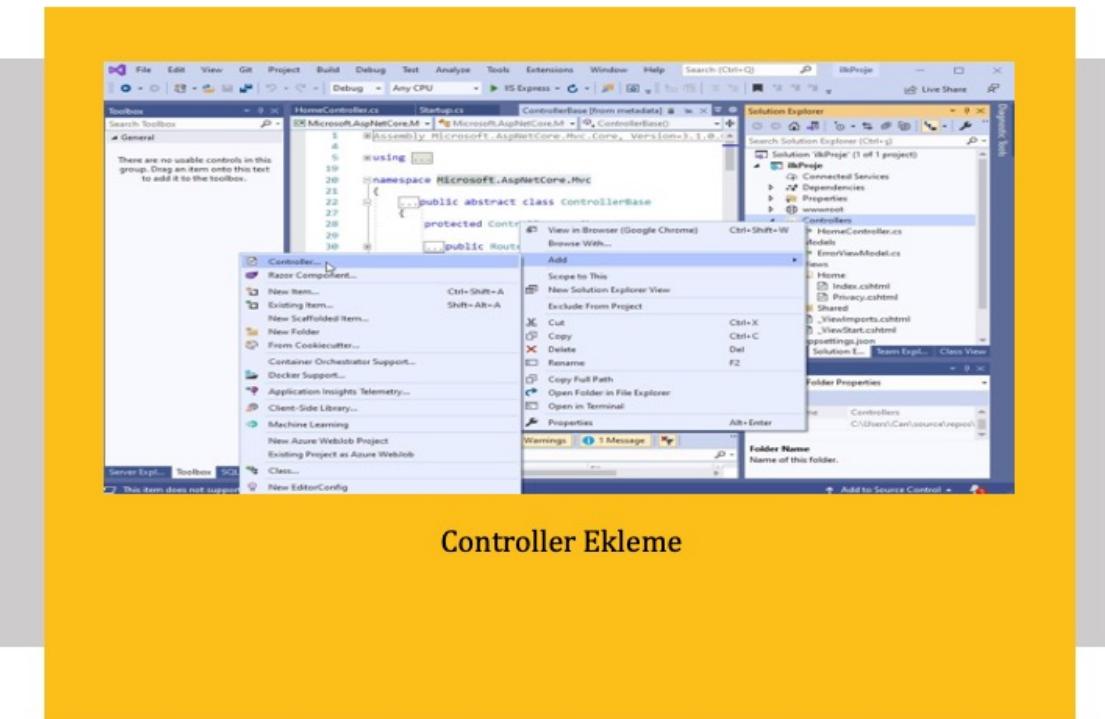
ASP.NET Core MVC, anlaşılır ve arama işlemleri için uygun URL'lere sahip uygulamalar oluşturulmasına olanak sağlayan güçlü bir URL eşleme bileşenine sahiptir. Projeye ait web sunucusundaki dosyaların düzenleninden ve hiyerarşisinden bağımsız olarak, arama motoru iyileştirmesi (SEO) için de faydalı olacak şekilde uygulamaya ait URL'lerinin düzgün olarak tanımlanmasını sağlar.

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

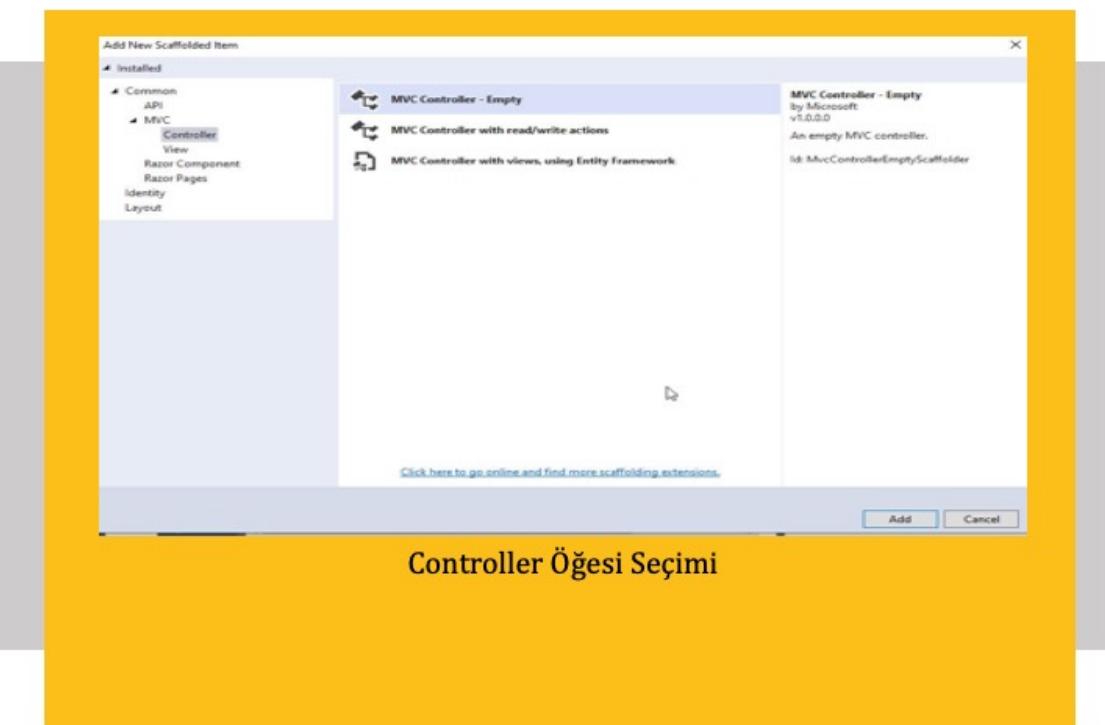


URL	Controller	Action	Id
http://localhost:3000/Home	HomeController	Index	null
http://localhost:3000/home/Index/100	HomeController	Index	100
http://localhost:3000/Home/About	HomeController	About	null
http://localhost:3000/Home/Contact	HomeController	Contact	null
http://localhost:3000/Ogrenci	OgrenciController	Index	null
http://localhost:3000/Ogrenci/Create/100	OgrenciController	Create	100

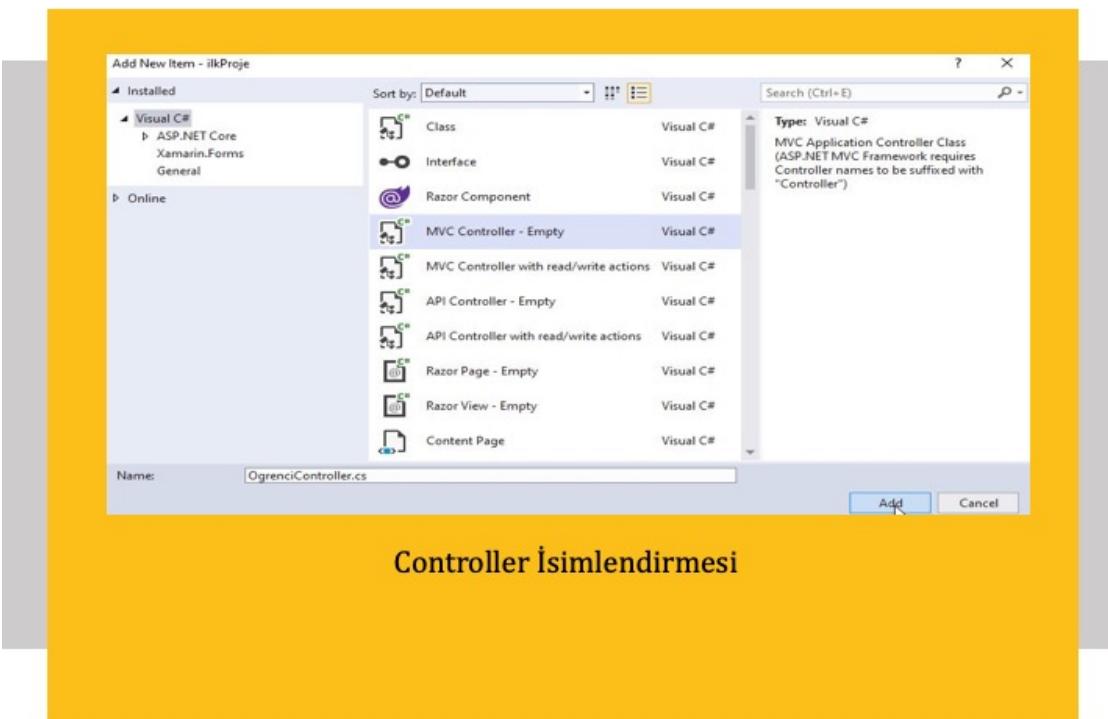
Url'ye göre Controller, Action ve Id değerleri



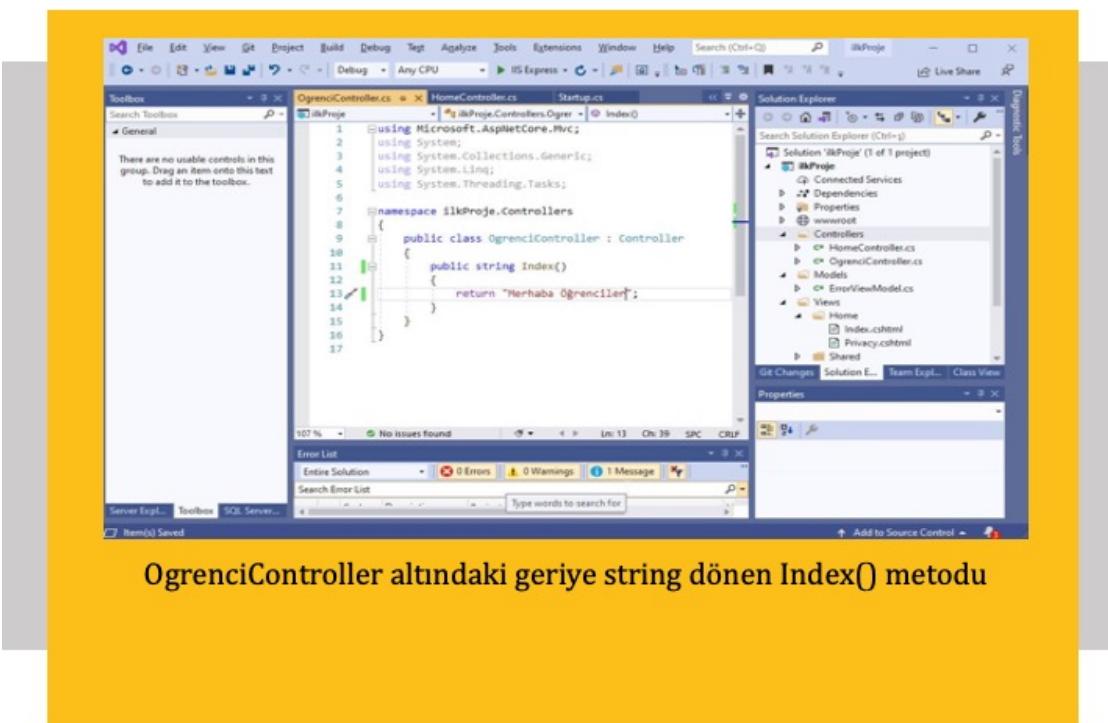
Controller Ekleme



Controller Öğesi Seçimi



Controller İsimlendirmesi



OgrenciController altındaki geriye string dönen Index() metodu

Action Metodlar

Controller sınıfının tüm public metodları “**Action Metod**” olarak adlandırılır. Action Metodlar public olmalıdır. Static ve aşırı yüklenmiş metodlar Action Metod olamazlar.

OgrenciController.cs ye ait kodlarda ilgili kısım aşağıdaki gibi güncellenip tarayıcıda <http://localhost:3000/Ogrenci/Index/100> şeklinde bir url adresi girilerek çalıştırıldığında çalıştırıldığında geriye string bir değer döner.

```
public class OgrenciController : Controller
{
    public string Index(int id)
    {
        return "Merhaba Öğrenciler ID=" + id;
    }
}
```



```
public IActionResult Home()
{
    return View();
}
```

```
public ActionResult Home()
{
    return View();
}
```

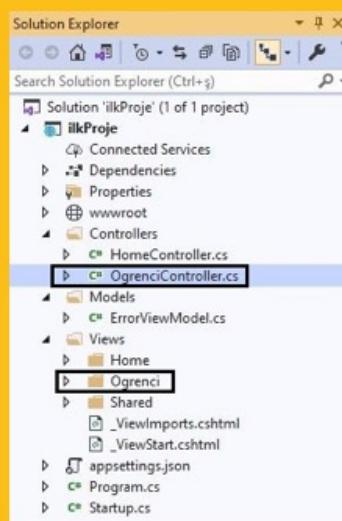
```
public ContentResult Index()
{
    string icerik = "<div><b> Merhaba </b> </div>";
    return Content(icerik, "text/html");
}
```

```
public JsonResult Index()
{
    return Json(JsonData);
}
```

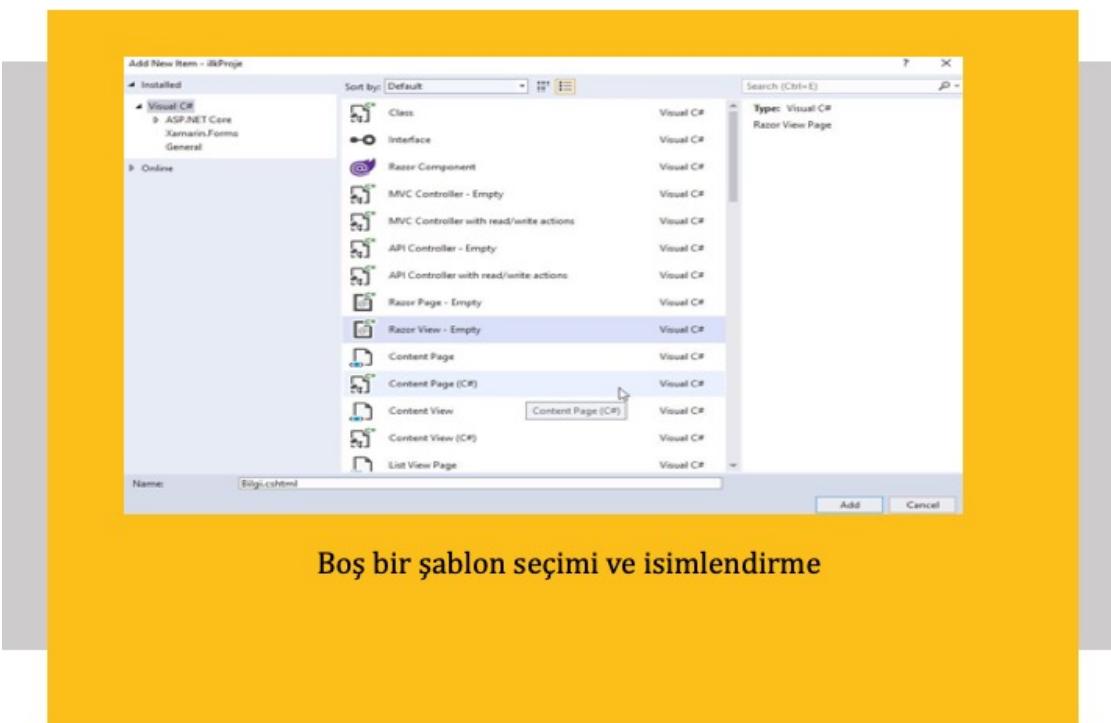
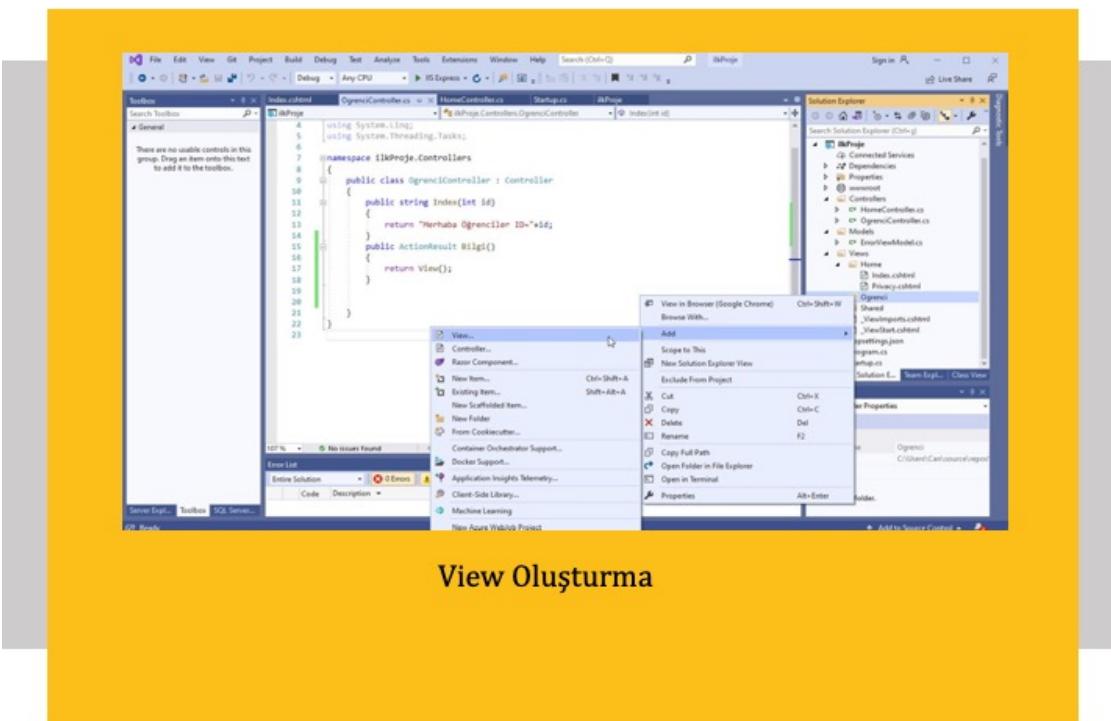
Bunların dışında ASP.NET Core; PageResult, ViewResult, RedirectResult, FileResult, StatusCodeResult, NotFoundResult gibi birçok farklı IActionResult türüne sahiptir

ASP.NET CORE'da View

- ASP.NET Core MVC'de View'ler, Razor biçimlendirmesinde C# programlama dilini kullanan .cshtml dosyalarıdır. Genellikle, View dosyaları, uygulamanın Controller'ların her biri için adlandırılmış klasörler halinde gruplandırılır. Bu klasörler, uygulamaya ait Views klasöründe yer alırlar
- ASP.NET Core MVC'de View'ler, Razor biçimlendirmesinde C# programlama dilini kullanan .cshtml dosyalarıdır. Genellikle, View dosyaları, uygulamanın Controller'ların her biri için adlandırılmış klasörler halinde gruplandırılır. Bu klasörler, uygulamaya ait Views klasöründe yer alırlar



Controller'lar için View klasörleri

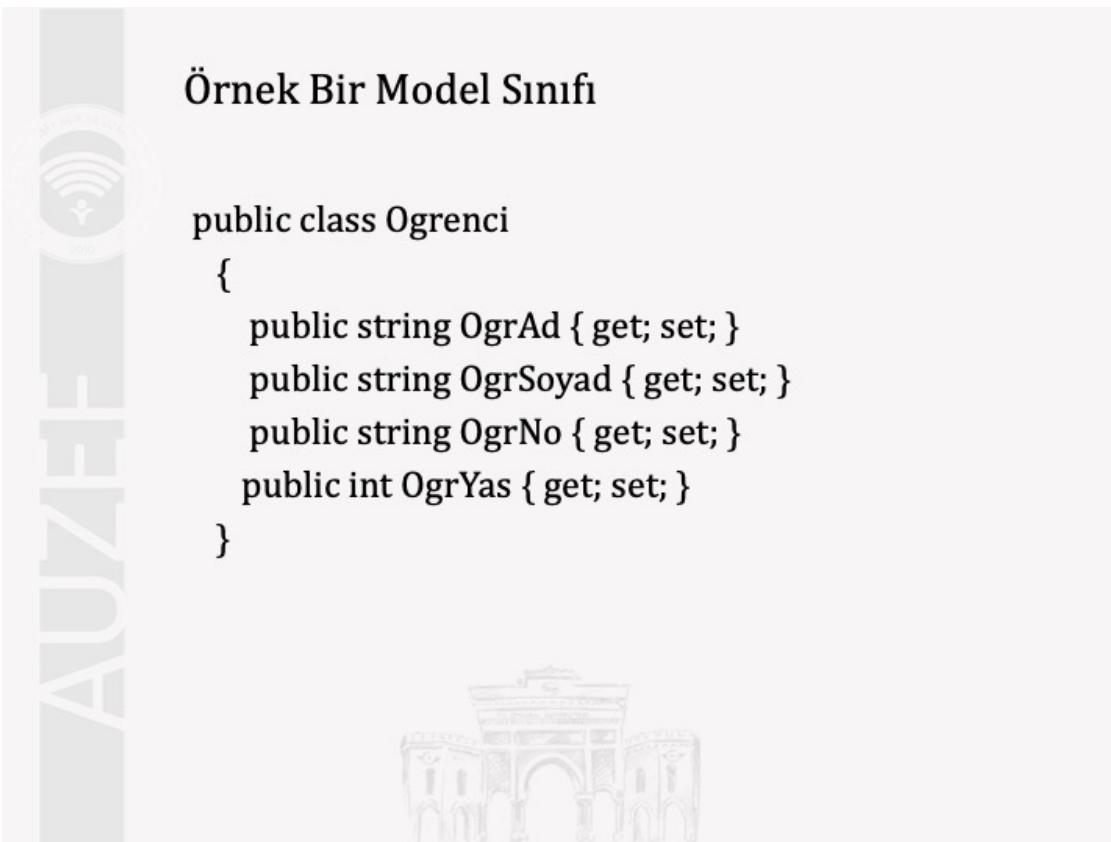


Boş bir şablon seçimi ve isimlendirme

ASP.NET CORE MVC uygulamalarında Model sınıfları Model klasöründe oluşturulmalıdır. Model klasörüne sağ tıklanarak, Add >> Class'a tıklandığında Yeni Öğe Ekle iletişim kutusu açılacaktır. Gelen ekranda Class seçilerek oluşturulacak class'a bir isim girilir.

Örnek Bir Model Sınıfı

```
public class Ogrenci
{
    public string OgrAd { get; set; }
    public string OgrSoyad { get; set; }
    public string OgrNo { get; set; }
    public int OgrYas { get; set; }
}
```

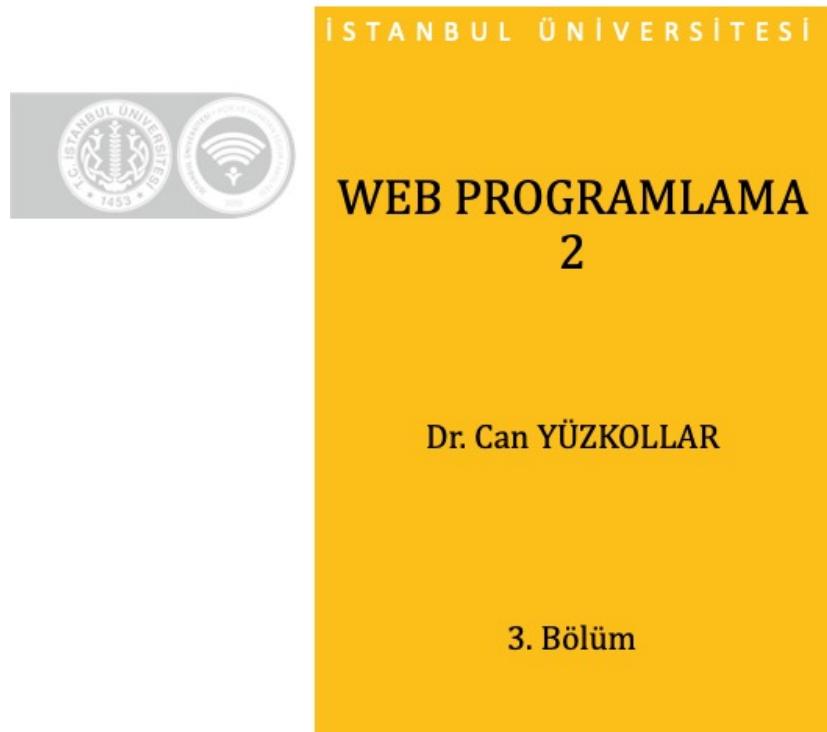




ÖZET

Bu bölümde Visual Studio kurulumu gerçekleştirirerel ile ASP.NET Core geliştirme ortamında MVC proje yapısından bahsetik. Daha sonra MVC yapısına ait Model (verileri temsil eder), View(Kullanıcı Arayüzüdür) ve Controller(istekleri karşılar) kavramlarını örnekledirerek açıkladık.





A yellow rectangular card with a grey border. On the left side, it says 'SUNUM PLANI'. To the right, under the heading 'Deneme', there is a list of four items:

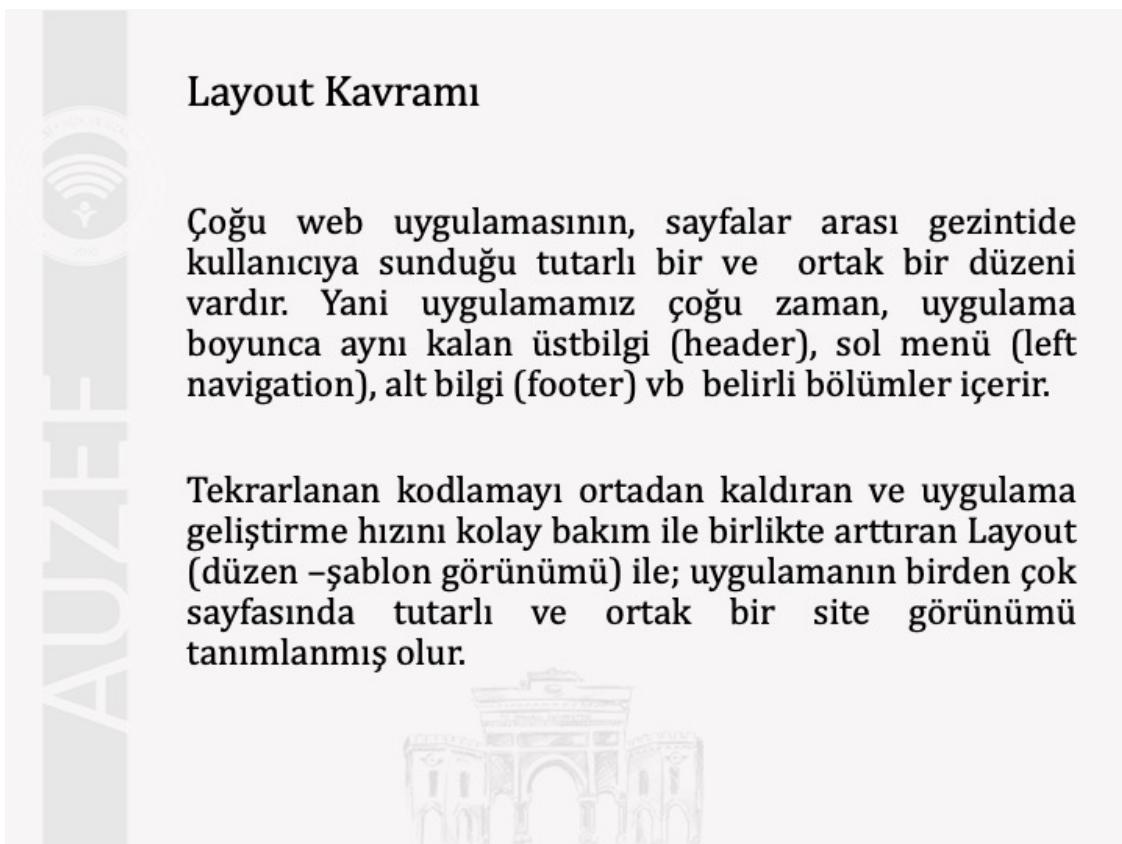
1. Layout Kavramı
2. Layout Kullanımı
3. Razor Görünüm Motoru
4. Razor Kullanımı

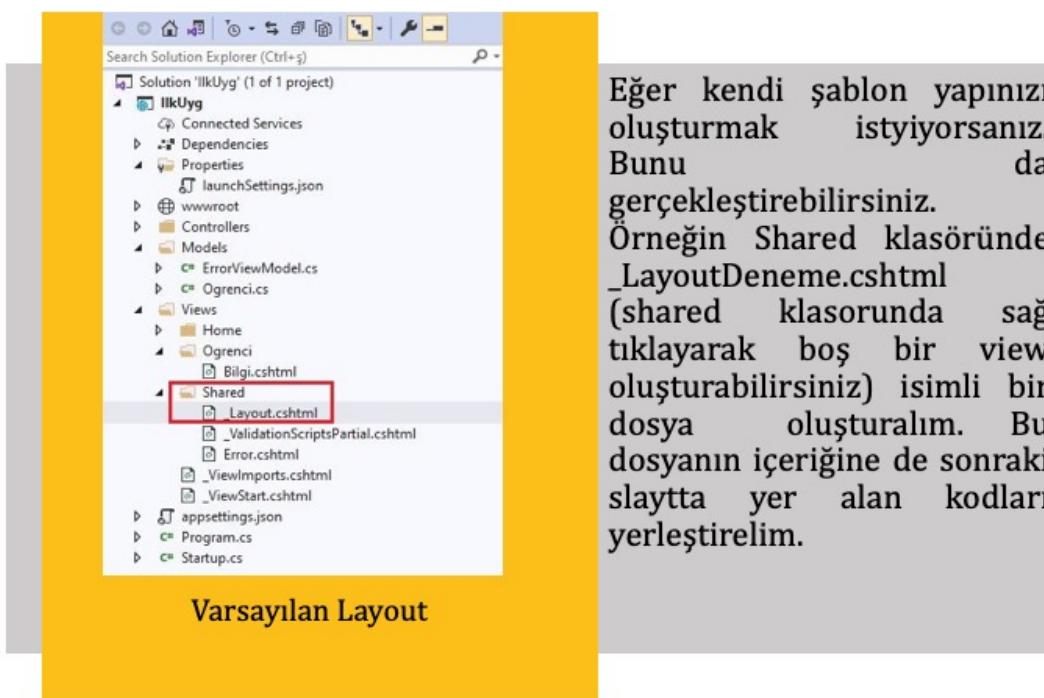
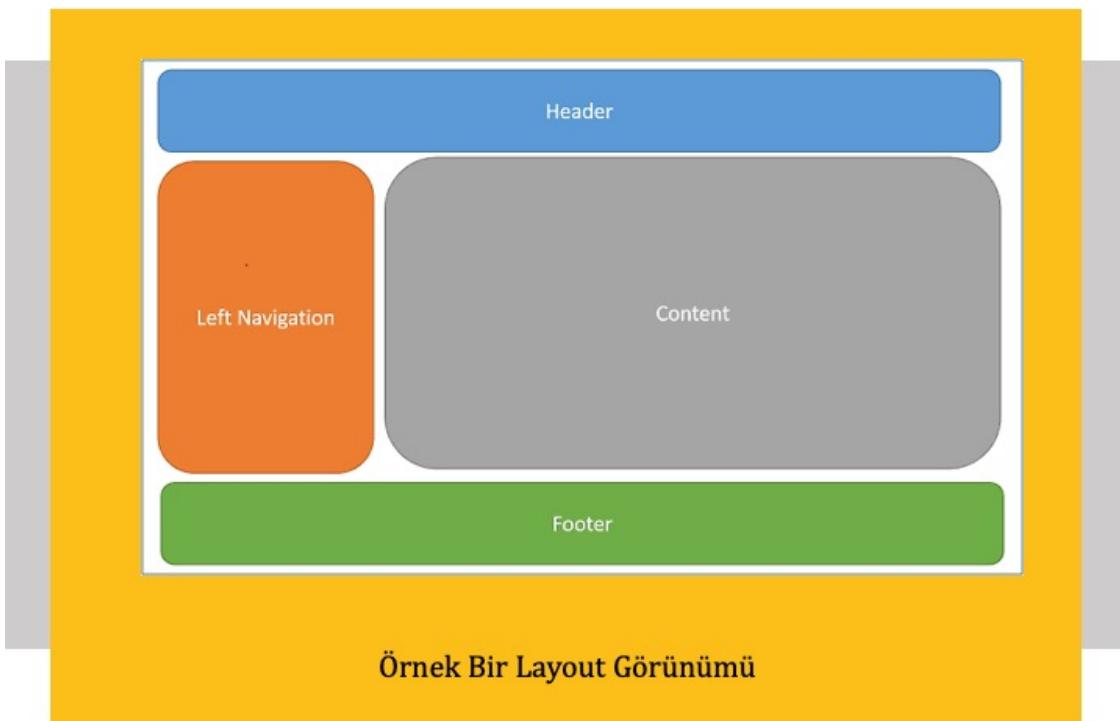


Layout Kavramı

Çoğu web uygulamasının, sayfalar arası gezintide kullanıcıya sunduğu tutarlı bir ve ortak bir düzeni vardır. Yani uygulamamız çoğu zaman, uygulama boyunca aynı kalan üstbilgi (header), sol menü (left navigation), alt bilgi (footer) vb belirli bölümler içerir.

Tekrarlanan kodlamayı ortadan kaldırın ve uygulama geliştirme hızını kolay bakım ile birlikte arttıran Layout (düzen -şablon görünümü) ile; uygulamanın birden çok sayfasında tutarlı ve ortak bir site görünümü tanımlanmış olur.





Layout Oluşturma

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <table border="1" style="width:1024px; height:768px;">
        <tr>
            <td colspan="2" style="text-align:center; height:100px;">
                <h3>Header Bölümü</h3>
            </td>
        </tr>
        <tr>
            <td style="width:150px">
                <h3>Sol Menü</h3>
            </td>
            <td style="width:600px">
                @RenderBody()
            </td>
        </tr>
        <tr>
            <td colspan="2" style="text-align:center; font-size:x-small; height:60px;">
                <h3>Footer Bölümü</h3>
            </td>
        </tr>
    </table>
</body>
</html>
```

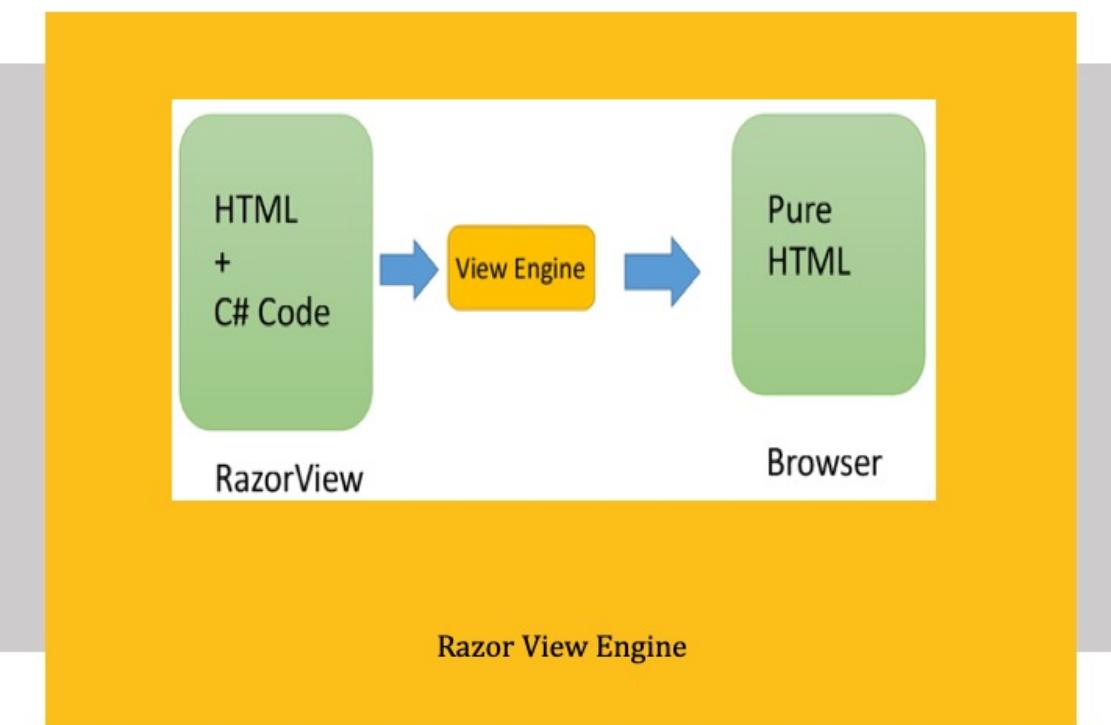
Layout Oluşturma

_ViewStart.cshtml dosyasında da artık bizim oluşturduğumuz _LayoutDeneme.cshtml isimli dosyanın kullanılacağını aşağıdaki kod ile belirttiğimizde artık projemiz bizim şablonumuza uygun olarak çalışacaktır.

```
@{
    Layout = "_LayoutDeneme";
}
```

Razor View Engine

- Programlama dilleri ile web projeleri üretildiğinde bazı zamanlar View'ler (görünüm) üzerinde o program diline ait kodlar yazılması gerekebilir.
- Yani görüntüler HTML ve bir programlama dili karışımıdır.
- Görünüler Html verilerini göstermek zorundadır. Bunun için kullanılan programala diline ait kodların Html çıktısının üretilmesi gereklidir.
- View Engine (görünüm motorları) bu kodları Html'ye dönüştürürler. (Controller'daki bir Action tarafından çağrıldığında bir HTML yanıtı üretmekten sorumludur.)



Razor Söz dizimi

Razor, HTML'den C# koduna geçiş yapmak için @ simbolünü kullanır. Geçişleri gerçekleştireceğiniz iki yol vardır.

- Tekli işlemler yapmak için @ simbolünden sonra kod ifadesi
- Birden fazla işlem yapılacağı zaman @*{ kodlar }* şeklinde bir blok yapısı kullanılmalıdır.

Tarih : @DateTime.Now.ToString() şeklinde bir kod → @DateTime.Now.ToString() kodu Razor tarafından C# kodu olarak yorumlanıp ekrana basılacaktır.

(Tarih : 14.08.2021 16:33:28 gibi günün tarih ve saatini veren bir çıktı)

Tarih : DateTime.Now.ToString() şeklinde yazılmış olsaydı Herhangi bir yorumlama olmadan çıktı aynı şekilde ekrana basılacaktır.

(Çıktı : Tarih : DateTime.Now.ToString())

Razor - Koşullu İfade Örneği if / if-else

```
@{var fiyat = 100;}  
<html>  
<body>  
    @if (fiyat > 50)  
    {  
        <p>Fiyat 50 lirada fazla.</p>  
    }  
</body>  
</html>  
  
@{  
    var sayı1 = 100;  
    var sayı2 = 200;  
}  
<div>  
    @if (sayı1 > sayı2)  
    {  
        <p>Sayı1 Büyük</p>  
    }  
    else  
    {  
        <p>Sayı2 Büyük</p>  
    }  
</div>
```

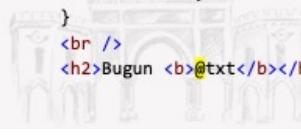
Razor- Koşullu İfade Örneği else if

```
@{var fiyat = 40;}  
<div>  
    @if (fiyat >= 100)  
    {  
        <h2>Fiyat Yüksek.</h2>  
    }  
    else if (fiyat > 30 && fiyat < 50)  
    {  
        <h2>Fiyat Normal</h2>  
    }  
    else  
    {  
        <h2>Fiyat düşük.</h2>  
    }  
</div>
```



Razor- Koşullu İfade Örneği else if

```
@{  
    string txt = "";  
    int day = Convert.ToInt32( DateTime.Now.DayOfWeek);  
    switch (day)  
    {  
        case 1:  
            txt = "Pazartesi";  
            break;  
        case 2:  
            txt = "Salı";  
            break;  
        case 3:  
            txt = "Çarşamba";  
            break;  
        case 4:  
            txt = "Perşembe";  
            break;  
        case 5:  
            txt = "Cuma";  
            break;  
        default:  
            txt = "Tatil";  
            break;  
    }  
    <br />  
    <h2>Bugün <b>@txt</b></h2>
```



Razor- for döngüsü

```

@{
    string[] Ogrenciler = { "Semra", "Hasan", "Tuncay", "Mehmet" };
}
<ul>
@for (int i = 0; i < Ogrenciler.Length; i++)
{
    <li>@Ogrenciler[i]</li>
}
</ul>

```

Kodun Ekran Görüntüsü	Kodun Html çıktısı
<ul style="list-style-type: none"> • Semra • Hasan • Tuncay • Mehmet 	<p> Semra Hasan Tuncay Mehmet </p>

Razor- foreach döngüsü

```

@{
    string[] Ogrenciler = { "Semra", "Hasan", "Tuncay", "Mehmet" };
}
<ul>
@foreach (var ogr in Ogrenciler)
{
    <li>@ogr</li>
}
</ul>

```

Kodun Ekran Görüntüsü	Kodun Html çıktısı
<ul style="list-style-type: none"> • Semra • Hasan • Tuncay • Mehmet 	<p> Semra Hasan Tuncay Mehmet </p>

Razor- while döngüsü

```
@{
    string[] ogrenciler = { "Semra", "Hasan", "Tuncay", "Mehmet" };

    int i = 0;
}
<p>
    <ul>
        @while (i < ogrenciler.Length )
        {
            @:
            <li>@ogrenciler[i]</li>
            i++;
        }
    </ul>
</p>
```

Kodun Ekran Görüntüsü	Kodun Html çıktısı
<ul style="list-style-type: none"> • Semra • Hasan • Tuncay • Mehmet 	<p> Semra Hasan Tuncay Mehmet </p>

Razor- do while döngüsü

```
@{
    string[] ogrenciler = { "Semra", "Hasan", "Tuncay", "Mehmet" };
    int i = 0;
}
<p>
    <ul>
        @do
        {
            <li>@ogrenciler[i]</li>
            i++;
        } while (i < ogrenciler.Length);
    </ul>
</p>
```

Kodun Ekran Görüntüsü	Kodun Html çıktısı
<ul style="list-style-type: none"> • Semra • Hasan • Tuncay • Mehmet • Mehmet 	<p> Semra Hasan Tuncay Mehmet </p>

Razor Yorum-Açıklama Bloğu

- Razor, yorum-açıklama bloğu için "@*...*@" sözdizimini kullanır.
- C# çoklu satır açıklamalarda /*...*/
- C# tek satır açıklama // kullanılabilir.
- HTML yapısı içerisinde ise açıklama bloğu <!-- ... --> şeklindedir.

```
@* açıklama *@  
{@  
/*  
  Çoklu satır  
  açıklama  
 */  
  
  //Tek satır açıklama  
}  
  
<!--  
    HTML blogu açıklama  
-->
```

ÖZET

Bu bölümde sayfa yapısının (uygulama boyunca çoğu zaman aynı kalan, üstbilgi (header), sol menü (left navigation), alt bilgi (footer) vb) tanımlanması olarak adlandırılacak Layout kavramına ait bilgiler verildi. Asp.net Core Mvc'de bu kavramın nasıl kullanıldığı ve istediği takdirde nasıl düzenlenip değiştirelebileceği bölüm boyunca tartışıldı.

Ayrıca ASP.NET Core uygulamaları için varsayılan View Engine'ları ve görünüm dosyalarında Razor işaretlemesini parse ederek HTML yanıtı üreten Razor View Engine'den bahsedildi. Razor kod yapısının view'lerde nasıl kullanılabileceği şart idadeleri ve döngü yapıları örnekleri ile açıklandı.



SUNUM PLANI

FORM İŞLEMLERİ VE MODEL BAĞLAMA

1. Web Form
2. Sık Kullanılan Form Elemanları
3. Form Gönderim Yöntemleri
4. Model Bağlama
5. Form verilerinin Model ile Bağlanması

AUZEF



Web Form

Web formları, kullanıcılar ile bir web sitesi veya web uygulaması arasındaki ana etkileşim noktalarından biridir. Formlar, verilerin işlenmesi ve depolanması için gerekli olan bir takım verilerin kullanıcidan alınarak web sunucusuna gönderilmesi şeklinde çalışır.

Form kontrolleri istenen formatta veya değerde veri girişini gerçekleştirmek (form doğrulama) ve kullanıcıyı bu yönde zorlamak için programlanabilir. Ayrıca form elamanlarını normal ve görme engelli kullanıcılara amaçlarını açıklayan metin etiketleriyle eşleştirilebilir. Bu sayede sağlıklı bir bilgi alış verisi gerçekleştirilmiş olur.

Input Form Elemanları

- o `<input type="text">`

Tek satırlık bir metin giriş alanı görüntüler

- o `<input type="button">`

Tıklanabilir bir buton görüntüler

- o `<input type="checkbox">`

Bir onay kutusu görüntüler (birden fazla seçenek seçmek için)

- o `<input type="radio">`

Birçok seçenekten birini seçmek için radyo düğmesi görüntülenir

- o `<input type="hidden">`

Form gönderildiğinde kullanıcılar tarafından görülemeyen veya değiştirilemeyen verileri içermesine izin verir.



Input Form Elemanları - Devam

- o `<input type="submit">`

Form göndermek için bir Gönder butonu görüntüler

- o `<input type="reset">`

Tüm form değerlerini başlangıç değerlerine sıfırlayan bir sıfırlama butonu tanımlar.

- o `<input type="file">`

Kullanıcının cihazından bir veya daha fazla dosya seçmesine olanak tanır.

- o `<input type="color">`

Kullanıcının görsel bir renk seçici arabirimini kullanarak veya rengi bir metin alanına #rrggbb gibi onaltılık biçimde bir renk belirtmesine olanak tanıyan bir kullanıcı arabirimini ögesi sağlar.



Input Form Elemanları - Devam

- o `<input type="date">`

Metin kutusu veya özel bir tarih seçici arabirimini kullanmanın bir tarih girmesine izin veren giriş alanları oluşturur.

- o `<input type="time">`

Kullanıcının bir zaman (saat ve dakika ve isteğe bağlı olarak saniye) girmesine izin vermek için tasarlanmış giriş alanları oluşturur.

- o `<input type="url">`

Kullanıcının bir URL'yi girmesine ve düzenlemesine izin vermek için kullanılır.

- o `<input type="email">`

Bir e-posta adresi için(Otomatik olarak doğrula yapılan) bir alan tanımlar.

- o `<input type="password">`

Karakterlerin maskelendiği bir şifre alanı tanımlar



Select ve Textarea Form Elamanı

```
<select name="select_ismi">
    <option value="deger1"> Görünen Değer </option>
    <option value="deger2"> Görünen Değer </option>
    <option value="deger3"> Görünen Değer </option>
</select>

<textarea name="txt_isim" rows="10" cols="30"></textarea>
```



Select ve Textarea Form Elamanı

```
<select name="select_ismi">
    <option value="deger1"> Görünen Değer </option>
    <option value="deger2"> Görünen Değer </option>
    <option value="deger3"> Görünen Değer </option>
</select>

<textarea name="txt_isim" rows="10" cols="30"></textarea>
```





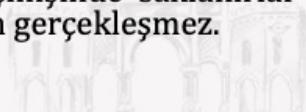
Post ve Get Metodları

- Get ile gönderim de url => www.deneme.com/?ad=Can&yas=40 (Form verileri url de gözükür durumda. ? ile query string olarak adlandılan verilerin başlayacağı belirtilir. Her bir ad/değer çifti & karakteri ile ayrılmış durumdadır.)
- Post ile gönderim de url => www.deneme.com (form verileri http istek gövdesinde)



Post ve Get Metodları- Devam

- POST metodu URL'de görüntülenmesi istenmeyen istekler için kullanılırken, get metodunda istekler URL de gözükür ve manipüle edilebilir. Dolayısı ile POST metodu daha güvenlidir.
- POST metodu GET metoduna göre daha yavaştır. Get metodu belirli karakter uzunluğunda (maksimum 2048) veri gösterilebilirken, post'da böyle bir limit yoktur.
- POST ile dosya tarzı formatlar (resim vb) sunucuya upload edebilebilirken GET ile bu mümkün değildir.
- GET metodu ile yapılan form işlemlerine ait kayıtlar tarayıcı geçmişinde saklanırlar böyle bir saklama POST metodu için gerçekleşmez.



Form Özellikleri

- *Action* özelliği formda girilen bilgilerin gönderileceği adresi belirler.
- *Method* özelliği ise form bilgilerinin hangi yöntemle (Get ya da Post) server'a gönderileceğini belirler.
- Form da action'ın tanımlı olmadığı durumda gönderim varsayılan olarak aynı sayfaya gerçekleştirilir.
- Method özelliği tanımlı değilse gönderim varsayılan olarak Get ile gerçekleştirir.

```
<form action="isle" method="POST">

    AD: <input name="ad" >
    YAŞ: <input name="yas" >
        <input type="submit" value="Gönder" />
</form>
```

Örnek Uygulama

Yeni bir proje oluşturup, Ogrenci sınıfını projeye ekleyelim.
(Model klasörüne sağ tıklayıp Add >> Class seçeneği ile *Ogrenci.cs* isminde bir class)

```
public class Ogrenci
{
    public string OgrAd { get; set; }
    public string OgrSoyad { get; set; }
    public string OgrNo { get; set; }
    public int OgrYas { get; set; }
}
```

Örnek Uygulama-Post Metodu

Projemize *OgrenciController.cs* isminde yeni bir Controller ekleyelim.
(Controller klasörüne sağ tıklayıp Add>>Controller)

View klasörüne sağ tıklayıp Add >>New Folder seçeneği ile) Bu Ogrenci klsörünün altında da Index.cshtm isminde bir oluşturup aşağıdaki formu tasarlayalım.

```
<form method="post" action="..\Ogrenci\Kaydet">
    <table>
        <tr>
            <td>Ad </td>
            <td>: </td>
            <td><input type="text" name="ad" /></td>
        </tr>
        <tr>
            <td>Soyad </td>
            <td>: </td>
            <td><input type="text" name="soyad" /></td>
        </tr>
        <tr>
            <td>Numara </td>
            <td>: </td>
            <td><input type="text" name="numara" /></td>
        </tr>
        <tr>
            <td colspan="3" align="center"><input type="submit" value="Gönder" /></td>
        </tr>
    </table>
</form>
```

Örnek Uygulama-Post Metodu Verilerin Alınması

- OgrenciController'ı altına *Kaydet()* isimli bir action oluşturalım.
- Bu action'ın başına *HTTP POST* isteklerine yanıt veren bir yöntem olan [*HttpPost*] ekleyelim.
- Kaydet()* action'ında form elemanına ait verileri almak için *HttpContext.Request.Form["form_elamani_adi"]* kodunu kullanalım
- txt* isimli bir değişken ile gelen tüm string verilerini birleştirelim

```
public class OgrenciController : Controller
{
    public IActionResult Index()
    {
        return View();
    }

    [HttpPost]
    public string Kaydet()
    {
        string ad = HttpContext.Request.Form["ad"];
        string soyad = HttpContext.Request.Form["soyad"];
        string no = HttpContext.Request.Form["numara"];
        string txt = ad + " " + soyad + " " + no;
        return (txt);
    }
}
```

The image shows two adjacent browser windows. The top window is titled 'FormUygulamasi' and contains a form with three fields: 'Ad' (Hasan), 'Soyad' (Tan), and 'Numara' (100). A 'Gönder' (Send) button is at the bottom. The bottom window is titled 'https://localhost:44364/Ogrenci/Kaydet' and displays the submitted data: 'Hasan Tan 100'. Below this, there is some footer text: '© 2021 - FormUygulamasi - Privacy'.

Formun Post metodu ile gönderilip karşılanması

Örnek Uygulama-Get Metodu

- Kodumuzda Post yerine Get kullanalım.
- Form verileri url de gözükmür durumda

```
<form method="GET" action="..\Ogrenci\Kaydet">
    <table>
        <tr>
            <td>Ad </td>
            <td>: </td>
            <td><input type="text" name="ad" /></td>
        </tr>
        <tr>
            <td>Soyad </td>
            <td>: </td>
            <td><input type="text" name="soyad" /></td>
        </tr>
        <tr>
            <td>Numara </td>
            <td>: </td>
            <td><input type="text" name="numara" /></td>
        </tr>
        <tr>
            <td colspan="3" align="center"><input type="submit" value="Gönder" /></td>
        </tr>
    </table>
</form>
```

Örnek Uygulama-Get Metodu Verilerin Alınması

- Kaydet() action'ında Get isteklerini karşılayan [HttpGet] kodu ekleyelim
- Kaydet() action'ında form ile gönderilen form elamanlarına ait verileri almak için HttpContext.Request.Query["form_elami_adi"] kodunu kullanalım
- NOT: Bu url'de veriler üzerinde değişiklik yaptığınızda değişikliğin aldiğiniz veriler üzerine de yansındığını görebilirsiniz.

```
[HttpGet]  
public string Kaydet()  
{  
    string ad = HttpContext.Request.Query["ad"];  
    string soyad = HttpContext.Request.Query["soyad"];  
    string no = HttpContext.Request.Query["numara"];  
    string txt = ad + " " + soyad + " " + no;  
    return (txt);  
}
```



Formun Get metodu ile gönderilip karşılanması



Model Binding (Model Bağlama)

- .Net projelerinde HTTP isteklerinden gelen değerlerin Model yapısı ile eşleştirilmesi işlemine Model bağlama(binding) denir.
- Böylelikle action yöntemlerinde verilere kolay erişim sağlanır.
- Model bağlama ile, Controller'lar ve sayfa işleyicileri C# tür ve verilerini incelemek zorunda kalmadan verileri otomatik olarak gelen istekten alır.
- <http://localhost:3000/Home/DegerAl/100?ogrAd=Hasan> url'si için Aşağıdaki DegerAl() action yöntemi, yukarıdaki istek URL'sini işler ve 100 değerini id parametresine ve ogrAd değerini ogrAd parametresiyle eşler.

```
public string DegerAl(int? id, string ogrAd)
{
    return "id = " + id.Value.ToString() + " and Ad = " + ogrAd;
}
```



Model Binding –Form Verilerini Bağlama

- Formda, tüm elemanların isimlerinin Ogrenci sınıfına ait özellik (property) adları ile eşleşmesi gerekmektedir. (Büyük-küçük harf duyarlı değil)
- Yukarıda Ogrenci sınıfımız ve view içinde oluşturduğumuz post metodu ile gösterilen bir form için model bağlama işlemini gerçekleştirelim.

```
[HttpPost]
public string Kaydet(Ogrenci ogr)
{
    string txt = ogr.OgrAd + " " + ogr.OgrSoyad + " " + ogr.OgrNo;
    return (txt);
}
```



ÖZET

Bu bölümde kullanıcı ile etkileşim noktalarından biri olan web form kavramına genel bir bakış gerçekleştirdik. Sık kullanılan form elemanlarını inceleyerek form gönderim yöntemlerini (Get, Post) tartıştık. Form verilerinin sunucu taraflı Get ve Post metodlarına göre elde edilmesi işlemlerini örneklendirdik.

HTTP isteklerinden gelen değerlerin Model yapısı ile eşleştirilmesi işlemi olan Model bağlama(binding) işleminden bahsettim. Bir istek URL'sinden gelen veriler ile model bağlama ve form verilerinin model sınıflarımız ile bağlanması işlemlerini öğrendik ve konu üzerine bir örnek gerçekleştirdik.





**DURUM YÖNETİMİ VE VIEW'E
VERİ AKTARIMI**

**SUNUM
PLANI**

1. Durum Yönetimi
2. Sunucu Taraflı Durum Yönetimi
3. İstemci Taraflı Durum Yönetimi
4. Controller'dan View'e Veri
Aktarım Yöntemleri



Durum Yönetimi (State Management)

Web tarayıcısı ve sunucular için Ağ Protokolleri Durumsuz Protokol(Stateless Protocol) ve Durum Bilgili protokol (Stateful protocol) olmak üzere iki türü ayrılr.

HTTP, stateless (durumsuz) bir protokol olarak adlandırılır, çünkü her komut isteği, kendisinden önce yürütülen istekler hakkında herhangi bir bilgi olmadan bağımsız olarak yürütülür.

Bir web sayfası görüntülemek isteyen bir kullanıcı sunucudan cevabı alıp aynı sunucudan başka bir sayfa talep ederse bu her seferinde yeni bir istek olarak değerlendirilir.



Durum Yönetimi Teknikleri

- İstemci taraflı durum yönetimi (client-based state management)
- Sunucuda taraflı durumu yönetimi (server-based state management)
- Her ikisinde birlikte kullanıldığı seçenekler



İstemci Taraflı Durum Yönetimi Teknikleri

Query String (Sorgu Dizisi)

Sayfa URL'sinin sonuna eklenen bir dizedir. Sorgu dizeleri, verileri bir Web sayfasından diğerine göndermek için çok basit ve popüler bir tekniktir. & karakteri kullanarak birden çok değer ayrılabilir. Sorgu dizesindeki bilgiler, tarayıcının adres satırında doğrudan kullanıcı tarafından görülebilir ve editlenebilir olduğundan sorgu dizeleri güvenli değildir.

(Örneğin url => www.deneme.com/?user=ali&limit=10)



İstemci Taraflı Durum Yönetimi Teknikleri

Hidden Fields

Gizli alanlar, `<input type="hidden">` şeklinde html form yapısı ile gönderilen verilerdir. Veriler, gizli form alanlarına kaydedilebilir ve bir sonraki istekte geri gönderilebilir.

Web sayfasında Kaynağı Görüntüle işlemi ile gizli alanlar ve onlarda yer alan değerler görüntülenebilir. İstemci potansiyel olarak verileri kurcalayabileceğinden, uygulama tarafından gizli alanlarda depolanan verilerin her zaman yeniden doğrulaması gereklidir.



İstemci Taraflı Durum Yönetimi Teknikleri

□HTTP Cookies (Çerezler)

Bir sunucunun kullanıcının web tarayıcısına gönderdiği küçük veri parçalarıdır. İstemci bilgisayarda metin dosyaları halinde depolanırlar. Çerezler, her istekte sunucuya gönderildiği için boyutları minimumda tutulmalıdır. Coğu tarayıcı, çerez boyutunu 4096 bayt ile sınırlar.

Tanımlama bilgileri genellikle içeriğin bilinen bir kullanıcı için özelleştirildiği kişiselleştirme için kullanılır. Kullanıcı yalnızca tanımlanır ve çoğu durumda kimliği doğrulanmaz. Çerez, kullanıcının tercih ettiği web sitesi arka plan rengi gibi kişiselleştirilmiş ayarlarına erişmek için kullanılabilir.

Çerez kullanımının bazı dezavantajları vardır. Bir kullanıcı, tarayıcı ayarlarını kullanarak çerezleri devre dışı bırakabilir, silebilir ve değiştirebilir. Bu nedenle hassas bilgilerin çerezlerle saklanması pek uygun değildir.



Asp.Net Core'da Cookie Kullanımı

Microsoft.AspNetCore.Http namespace'i projeye eklenmelidir.

`HttpContext.Response.Cookies.Append("CookieName", "value");` kodu ile cookie'ye değer atanabilir.

`HttpContext.Request.Cookies["CookieName"];` kodu ile cookie'deki değer okunabilir.

```
var cookieOptions = new CookieOptions
{
    Expires = DateTime.Now.AddMinutes(10),
    Domain = ".test.com"
};
```

Kodu ile de cookie'ye ait özellikler tanımlanabilir.

Sunucu Taraflı Durum Yönetimi

Session State

Bir web oturumu, bir kullanıcının ilk sayfaya gelip siteden ayrıldığı ana kadar web sitesinde gezinmek için harcadığı süredir.

Her kullanıcı bir siteye ilk ziyaretini gerçekleştirdiğinde kendi için benzersiz bir oturum bilgisi atanır ve her oturumun benzersiz bir kimliği (ID) vardır. Bu oturum, bir web sitesi veya web uygulamasında kullanımı boyunca kalıcı olan verilerden veya dosyalardan oluşur.

Örneğin kullanıcı adı Session'a kaydedilerek her sayfada veri tabanına tekrar takrar bağlanmak zorunda kalmadan sayfalar arasındaki geçişte kullanılabilir.

Asp.Net Core'da Session Kullanımı

- Projede .Net Core 2.0 veya daha düşük versiyon kullanılıyorsa ilk olarak Nuget'ten yükleme paketi'nden *Microsoft.AspNetCore.Session* yüklenmelidir.
- Startup.cs dosyasında ConfigureServices metoduna services.AddSession(); eklenmelidir.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSession();
}
```
- Startup.cs dosyasında Configure metoduna app.UseSession(); eklenmelidir.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseSession();
}
```
- Microsoft.AspNetCore.Http; namespace'i eklenmelidir.
- HttpContext.Session.SetString("SessionUser", "değer") atama işlemi



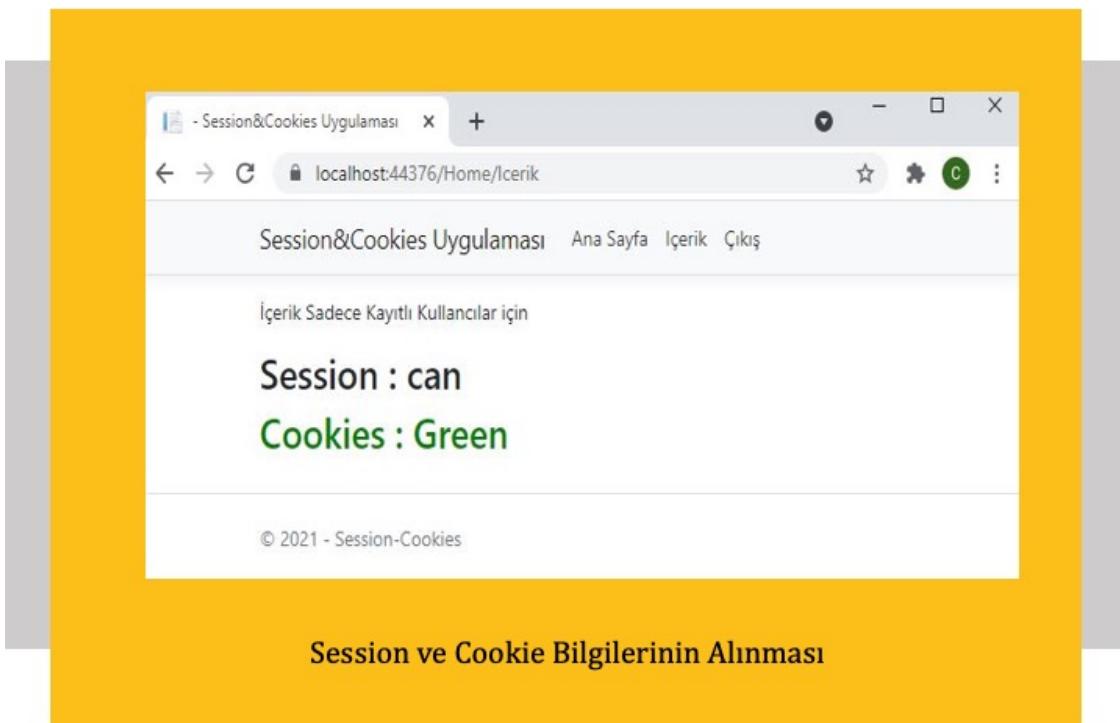
Session&Cookies Uygulaması Ana Sayfa İçerik Çıkış

Kullanıcı Adı:

Şifre :

© 2021 - Session-Cookies

Örnek bir Session ve Cookie Uygulaması

The sidebar features the logo of Marmara University (AUZEF) on the left, which includes a circular emblem with a person icon and the text "MARMARA ÜNİVERSİTESİ" and "AUZEF". To the right of the logo, the word "AUZEF" is written vertically in large, bold, black letters.

View'e Veri Aktarımı

- Bir MVC modelinde, tüm HTTP isteklerini karşılayıp işlemek ile görevli olan Controller çoğu zaman Action metodunda ViewResult dönüş tipi ile uygun bir view döndürür.
- View'lerde herhangi bir bilgiyi görüntülemek Controller'dan View'lere veri akışı ile gerçekleşmelidir.
- Verileri View'lere aktarmanın çeşitli yolları vardır.



ViewBag

```

graph LR
    Controller[Controller] --> View[View]
    Controller["ViewBag.OgrAd = \"Hasan\";"] --> View["@ViewBag.OgrAd"]
  
```

Genelde modele dahil olmayan, program çalıştırılmadan önce yapısı bilinmeyen çalışma zamanında belirlinen çok büyük boyutta olmayan verilerin Controller'dan View'e aktarmak için kullanılır.

Controller sınıfının temel sınıfı olan ControllerBase sınıfının dinamik bir tür özelliğidir.

ViewBag özelliğine değer olarak basit veya karmaşık tipte bir nesne atanabilir. Sadece geçerli istek sırasında okunabilir. Yönlendirme sonrasında kendini imha eder, null değer alır. Yani aktarımı yapıldığı view de son bulur.

ViewData

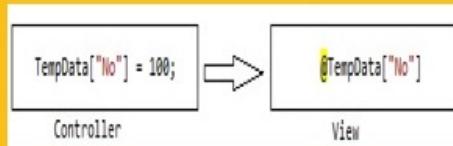
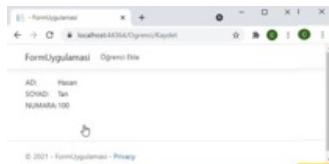
```

graph LR
    Controller[Controller] --> View[View]
    Controller["ViewData[\"Ad\"] = \"Hasan\";"] --> View[" ViewData[\"Ad\"]"]
  
```

ViewData, Controller'dan View'e veri aktarmak için kullanılan bir başka yöntemdir.

ViewData [Dictionary](#) türünde olduğundan, anahtar/değer çiftlerini (key-value pairs) içerir.

ViewData, verileri yalnızca Controller'dan View'e veri aktarırken sadece mevcut istek sırasında geçerlidir.



TempData

ViewDataDictionary türünü kullanan bir Dictionary'dir.

ControllerBase sınıfının özelliklerine sahiptir.

Viewbag ve ViewData bir istekten sonra gerçeliliklerini yitirlerken, TempData ilk istek ve sonraki istek sırasında da kullanılabilir.

ViewBag, ViewData ve TempData

- ViewData ve TempData değerlere erişmek için dictionary sözdizimini (key-value) kullanırken, ViewBag property sözdizimini (dot notation) kullanır.
- ViewData ve TempData ile kullanılan değer, bir değişene aktarılmak istendiğinde tip dönüşümü yapılması gereklidir. ViewBag'de tip dönüşümüne gerek yoktur.

```
@{
    string msj = (string)ViewData["Ad"];
    string mj2 = ViewBag.OgrAd;
    int a = (int)TempData["No"];
}
```

- ViewData, ViewDataDictionary'den türetilmiştir, bu nedenle ContainerKey, Add, Remove ve Clear gibi yararlı olabilecek özelliklere sahiptir.

ViewBag, ViewData ve TempData

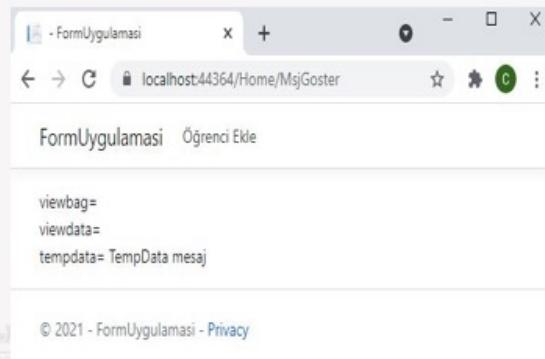
ViewBag ve ViewData yalnızca geçerli istek süresince kullanılır. Yaşam döngüsü bir sonraki isteğe kadardır (ilk istekten sonra NULL değerlerini alırlar), TempData geçerli ve sonraki istek sırasında da kullanılabilir. Yani yaşam döngüsü o anki ve bir sonraki HTTP istek süresindedir.

```
public ActionResult Index()
{
    ViewBag.Msg1 = "ViewBag mesaj";
    ViewData["Msg2"] = "ViewData mesaj";
    TempData["Msg3"] = "TempData mesaj";
    return RedirectToAction("MsjGoster");
}

public IActionResult MsjGoster()
{
    return View();
}
```

MjsGoster.cshtml View'inin içeriği verilmiştir.

```
viewbag= @ViewBag.Msg1 <br />
viewdata= @ViewData["Msg2"] <br />
tempdata= @TempData["Msg3"]
```



Model Verilerinin Aktarılması-View Model

Çoğu uygulamada Controller'lardan view'ler gerçekleştirilecek veri akışı, Modelde tanımlı class'lara ait nesnelerin, yani veri tipinin açıkça tanımlanıp kullanıldığı durumlarda gerçekleşmektedir.

Bu kullanımda ise ViewModel'in türünü belirten View dosyasının en üstüne yerleştirilen @model yönergesi ile intellisense yardımlarından ve hata varsa derleme zamanı hatalarından da yararlanılmış olur.

Model datası gönderilen bir view kendisine gönderilen dataya @model yönergesini kullanarak ulaşabilir.

Controller'da bir action metoddan veri model aktarımı

```
[HttpPost]
public ViewResult Kaydet(Ogrenci ogr)
{
    //Burada veri tabanına kayıt vb yapılabilir
    return View(ogr);
}
```

Kaydet.cshtml dosyası içeriği

```
@model Ogrenci
 @{
    <p>Ad :@ Model.OgrAd </p>
    <p>Soyad :@ Model.OgrSoyad </p>
    <p>Numara :@ Model.OgrNo </p>
 }
```

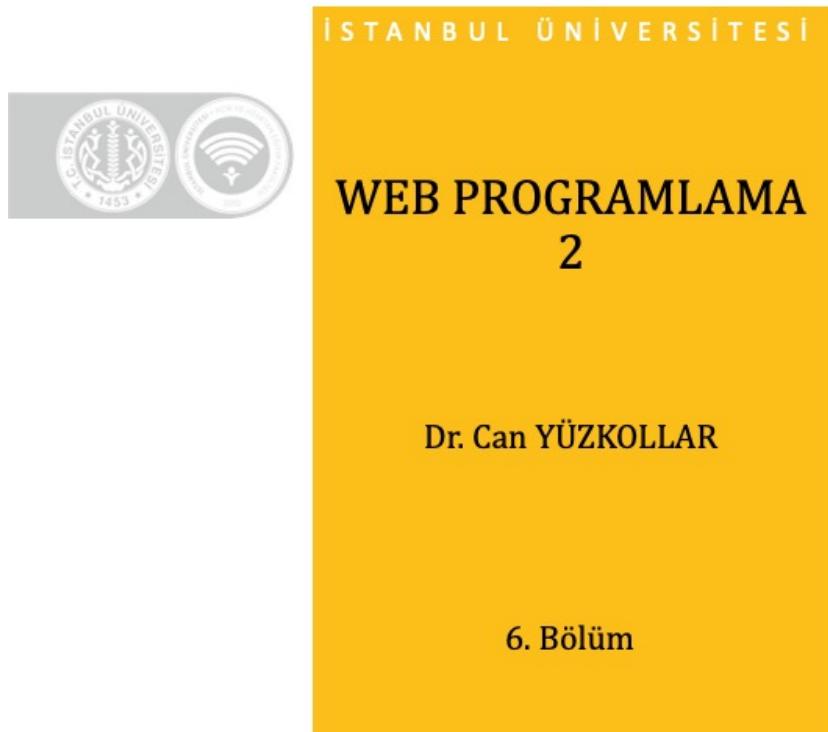
ÖZET

Bu bölümde stateless (durumsuz) bir protokol olan http protokülünde gerçekleşen her isteğin kendisinden önceki bağlantılarla hiçbir ilişkisi olmadan benzersiz ve bağımsız olarak değerlendirildiğinden bahsettim. Kullanıcılara özgür kimi web uygulamalarında kullanıcı ilerlemesi, tercihleri vb durumlar için özelleştirme yapmak istendiğinde bazı verilerin uygulanmanın tamamında ulaşılabilir olması gerektiğini tartıştık.

Bu noktada kullanıcıya ait bilgilerin istemcide ya da sunucuda saklandığı tekniklerin avantaj ve dezavantajlarını inceledik ve örnek bir uygulama gerçekleştirdik.

Aynı zamanda controller'dan view'e veri aktarmak için kullanılan ViewBag, ViewData, TempData ve Model Verilerinin Aktarılması olarak adlandıralan View Model yöntemlerini inceleyerek bunlar hakkında örnekler gerçekleştirdik.





The image shows a presentation slide. On the left, there is a yellow rectangular box containing the text 'SUNUM PLANI' in black. To the right of this box, the title 'Tag Helper' is displayed in a large, bold, black font. Below the title, there is a list of four items, each starting with a number and a dot:

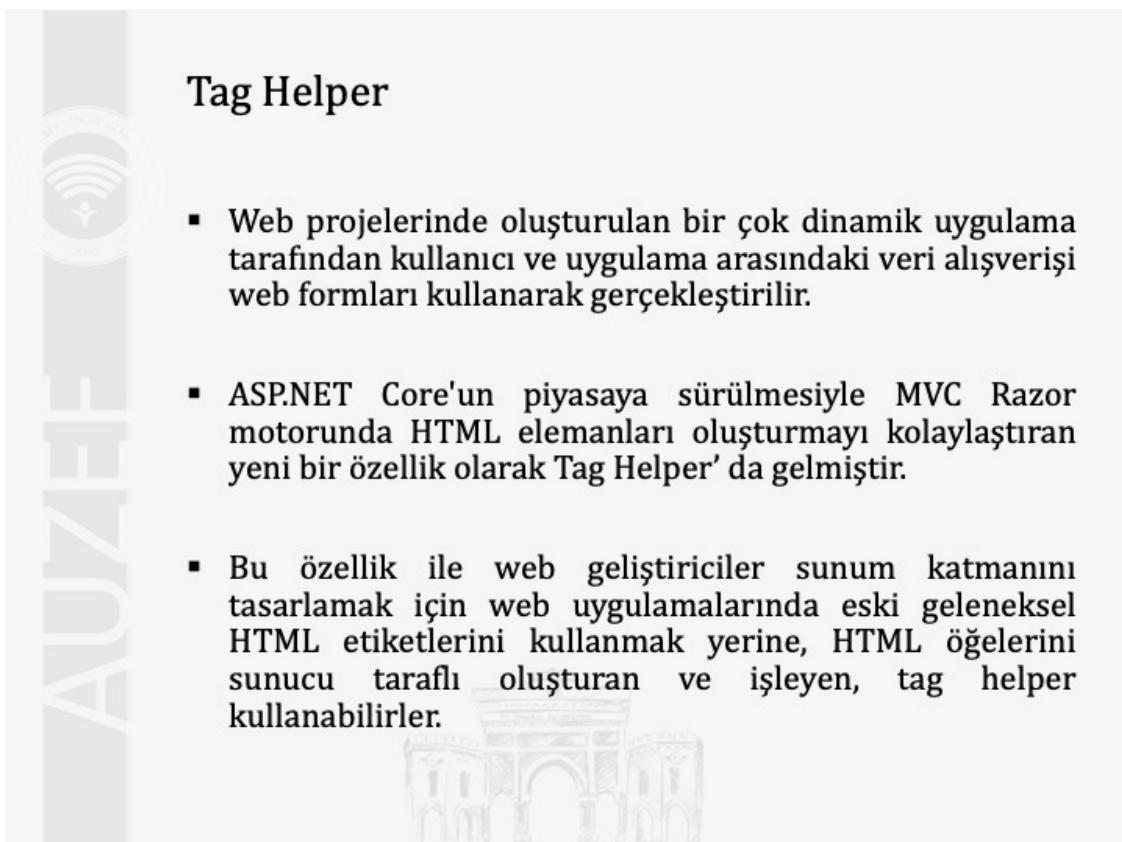
1. Tag Helper Nedir?
2. Tag Helper Avantajları
3. Sık Kullanılan Tag Helper'lar
4. Hazır Razor Sayfa Şablonları

On the far right edge of the slide, there is a large, stylized yellow arrow pointing towards the bottom right corner.



Tag Helper

- Web projelerinde oluşturulan bir çok dinamik uygulama tarafından kullanıcı ve uygulama arasındaki veri alışverişi web formları kullanarak gerçekleştirilir.
- ASP.NET Core'un piyasaya sürülmESİyle MVC Razor motorunda HTML elemanları oluşturmayı kolaylaştıran yeni bir özellik olarak Tag Helper' da gelmiştir.
- Bu özellik ile web geliştiriciler sunum katmanını tasarlAMak için web uygulamalarında eski geleneksel HTML etiketlerini kullanmak yerine, HTML öğelerini sunucu taraflı oluşturan ve işleyen, tag helper kullanabilirler.





Tag Helper

- Gramatik olarak html etiketlerinin yapısına benzer fakat burada bu elemanlar C# yardımıyla oluşturulur. Bu sayede Razor ve C#'ın gücünü kullanarak IntelliSense desteği ile de HTML ile çalışmayı çok basit hale getirir.
- Tag helper'ler HTML'de yazılan kodlama miktarını azaltır.
- HTML Helper'ların (halen kullanımı mevcut) mantıksal halefidir. Aşağıda örnek HTML kodunda, kullanılan link tag helper'ına ait sözdiziminin HTML'ye ne kadar benzediği açıkça görülmektedir.

```
<a asp-controller="Ogrenci" asp-action="Kaydet" class="orn_class">Tıkla</a>
```



Tag Helper Avantajları

- Tag Helper derleme zamanında kontrol edilir. Model ile bağlanabilir. Yanlış yazılan ya da değişikliğe uğrayan bir Model olduğunda derleyici hata verir ve bunu düzeltmeye zorlar.
- Razor Tag Helper kullanan biçimlendirme standart HTML gibi görünür. Front-End tasarımcılar HTML/CSS/JavaScript ile, Razor C# sözdizimi öğrenmeksızın düzenleme yapabilir.
- HTML ve Razor biçimlendirme oluşturmaya yönelik zengin bir IntelliSense(aklılı kod veya metin tamamlamaya izin veren bir araç) desteği sağlar.
- Yazılan kod, HTML Helper'lara kıyasla daha temiz ve okunabilir hale gelir. C# kodu ve HTML kodları arasında geçiş yapmak için @ kaçış dizisini kullanmaya gerek yoktur.
- ASP.NET Core birçok yerleşik Tag Helper barındırır. Bunlar ihtiyaçları karşılamıyorsa, kendi tag helper'ları oluşturmamıza olanak sağlar.





Anchor Tag Helper (Bağlantı Etiketi Yardımcısı)

- **asp-controller** : Bu öznitelik, URL'yi oluşturmak için kullanılan denetleyici adını belirler. Böylece yönlendirme belirtilen controller'a gerçekleşmiş olur. Tanımlı olmazsa view'i oluşturan controller varsayılan olarak atanır.
- **asp-action** : Bu öznitelik, action yöntemi adını belirtmek için kullanılır. Böylece yönlendirme belirtilen action'a gerçekleşmiş olur. Bu öznitelik adına herhangi bir değer atanmazsa, görünümü oluşturmak için denetleyicideki varsayılan asp-action değeri yürütülür.
- **asp-route**: Bu öznitelik, eylemin URL'sini oluşturmak için kullanılacak yolun adını belirtmek için kullanılır.

```
<a asp-controller="Ogrenci" asp-action="Kaydet" >Tıkla</a>
```



Form Tag Helper (Form Etiket Yardımcısı)

- Form Etiket Yardımcısı, MVC Controller eylemi için eylem özniteliklerine sahip (asp-controller, asp-action, asp-route vb) sahip HTML form öğelerini oluşturur.
- Aynı zamanda CSRF (Cross Site Request Forgery- Siteler Arası İstek Sahtekârlığı) şeklinde saldırular için form içinde "Request Verification Token" isimli hidden(gizli) bir değişken de oluşturur.

```
<form asp-action="Kaydet" asp-controller="Ogrenci">
...
</form>
```

(Yukarıdaki form tag helper'ına ait kaynak kod aşağıda yer almaktadır.)

```
<form action="/Ogrenci/Kaydet" method="post">
    <input name="__RequestVerificationToken" type="hidden"
    value="CfDJ8BmTpDLY1bNNg970arED8smRThJMZg72qxFd0c30UnfNLGxqJqXd7KGr0eoP2TkF4vI
    wCKmgTsxNnrnQvuNGCTthhzUZR-p5v2PZ-FCbxk7UyF-
    gFMCLCv2jaagOsuzOBnOxMp5VenWarLm-oHHLQ" />
</form>
```



Form tag helper'ı ile kullanılabilen özellikler

- **asp-route-{value}** : Örneğin {controller}/{action}/{id?} için
`<form asp-controller="Home" asp-action="Delete" asp-route-id="10">` kodu
`<form method="post" action="/Home/Delete/10">` html çıktısını üretir
- **asp-area**: URL oluşturulurken kullanılacak alan adını ayarlar.
`<form asp-area="Sirket" asp-controller="Home" asp-action="Delete">` kodu
`<form method="post" action="/Sirket/Home/Delete">` html çıktısını üretir
- **asp-fragment**: Bu özellik, URL'ye eklenmesi istenen URL parçasını ayarlar.
URL parçası, "#" sonra karakterinden sonra URL'nin sonuna eklenir.
`<form asp-controller="Home" asp-action="Delete" asp-route-id="10"`
`asp- fragment="test">`
şeklinde bir bir kod
`<form method="post" action="/Home/Delete/10#test">` çıktısını üretir.
- **asp-page** : Razor Pages ile birlikte kullanılır. Yönlenecek razor sayfası tanımlanır.
`<form asp-page="/Kayit">` kodu
`<form action="/?page=Kayit" method="post">` html kodunu üretir.



Form tag helper ile kullanılabilen özellikler

- **asp-host** : Oluşturulacak URL'de bir Host tanımları.
`<form asp-host="test.com" asp-controller="Home" asp-action="Deneme">` kodu
`<form href="http://test.com/Home/Deneme">` html kodunu üretir.
- **asp-protocol** : URL'de kullanılacak protokolü ayarlar
`<form asp-protocol="https" asp-controller="Home" asp-action="Delete">` kodu
`<form href="https://localhost/Home/Delete">Create</form>` html kodunu üretir.
- **asp-antiforgery**: Siteler arası istek sahteciliği (CSRF) saldırısını önlemek için otomatik olarak hidden bir alan üretir. Kullanılmaması durumunda default değeri true'dur.
`<form asp-controller="Home" asp-action="Delete" asp-antiforgery="true"><form`
`action="/Home/Delete" method="post">`
`<input name="_RequestVerificationToken" type="hidden"`
`value="CfDJ8BmTpDLY1bNNg970arED8smRThJMZg72qxFd0c30UnfNlGxqJqXd7KGr0eoP2TkF`
`4vlwCKmgTsxNnrnQvuNGCTthhzZUZR-p5v2PZ-FCbxk7UyF-`
`gFMCLCv2jaagOsuzOBnOxMp5VenWarLm-oHLLQ" />`
`</form>`





Input Tag Helper

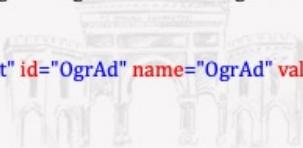
- Input (giriş) ögesi, HTML formlarının omurgasıdır ve bir kullanıcının bir uygulama sunabileceği ana araçları sağlar. Input tag helper, asp-for özniteliği kullanılarak model özelliğine uygun <input> HTML ögesini oluşturur.
- asp-for özniteliği ile model özelliğine bağlanır ve model özelliğinin türü, adı ve veri açıklamalarına dayalı olarak HTML'yi oluşturur.

Örneğin Ogrenci class'ımızda OgrAd alanı için bir input tag helper kullanalım.

```
public class Ogrenci
{
    public string OgrAd { get; set; }
    public string OgrSoyad { get; set; }
    public string OgrNo { get; set; }
    public int OgrYas { get; set; }
}
```

<input asp-for="OgrAd" > şeklinde tanımlanan bir tag helperde görebileceğiniz gibi asp-for ile tag helper'in değerini Ogrenci sınıfının OgrAd özelliğine ayarlıyoruz. (**asp-for="OgrAd"**) Ogrenci sınıfındaki

```
<input type="text" id="OgrAd" name="OgrAd" value="" />
```



.NET türlerine karşılık oluşturulan HTML input özellikleri

.NET type	Input Type
Bool	type="checkbox"
String	type="text"
DateTime	type="datetime-local"
Byte	type="number"
Int	type="number"
Single, Double	type="number"

```
public DateTime DogumTarihi { get; set; }
```

<input asp-for="DogumTarihi" /> şeklinde oluşturduğumuzda bu koda ait html görünümü aşağıdaki gibi olacaktır. Burada input ögesine ait tipin type özelliğinin **type="datetime-local"** olduğunu dikkat edin.

```
<input class="form-control" type="datetime-local" data-val="true" data-val-required="The DogumTarihi field is required." id="DogumTarihi" name="DogumTarihi" value="" />
```



Label Tag Helper

Çeşitli form elemanlarına ait veri işlemlerinde (`<input>`, `<textarea>`) kullanıcının bu öğeye ait kendinden ne tür bilgi istendiğine dair fikir sahibi olması gerekmektedir.

Bu manuel olarak oluşturulan bir text yazısı olabileceği gibi dinamik olarak modele bağlı oluşturulan bir text de olabilir. Bu açıklamada niteliğinde ki text bilgisi, Label tag helper'in `asp-for` özniteliği ile modele ait özelliğe bağlı olarak dinamik olarak oluşturulabilir.

Oluşturduğumuz Öğrenci sınıfına ait bir label tag helperi aşağıdaki gibi kullanılabilir.

```
<label asp-for="OgrSoyad" class="control-label"></label>
<input asp-for="OgrSoyad" class="form-control" />
```

Yukarıdaki kod çalıştırıldığında input etiketinden önce bu input alanına ait otomatik olarak oluşturulan açıklama aşağıda gösterilmiştir.

OgrAd

[Display (Name = "Öğrenci Adı")]

public string OgrAd { get; set; }

Öğrenci Adı

Display özelliği ile bir etikete için Label tag helperi ile bağlanan görüntülenen metin değerlerini istedimiz şekilde oluşturmak için kullanılır.

Textarea Tag Helper

Birden fazla satırda metin girişine müsade eden ve kaydırma çubukları desteğiyle birlikte daha da çok satırda paylaşılabilen bir metin giriş desteği sağlar.

```
public string OgrAdres { get; set; }
```

Öğrenci sınıfının içersine ekelenen yukarıdaki özellik textarea tag helper'i ile aşağıdaki şekilde kullanılabilir

```
<label asp-for="OgrAdres" class="control-label"></label>
<textarea asp-for="OgrAdres" class="form-control"></textarea>
```

Bu koda ait html çıktısı ve kodun görünümü aşağıda verilmiştir.

```
<label class="control-label" for="OgrAdres">OgrAdres</label>
<textarea class="form-control" id="OgrAdres" name="OgrAdres"></textarea>
```

OgrAdres

Select Tag Helper

- HTML Select etiketi, açılır bir liste oluşturmak için kullanılır. Açılmış listedeki her bir eleman <option> öğeleri ile tanımlanır. Select Tag Helper, asp-for özniteliği ile modele ait özelliğe bağlanabilir.
- Açılmış listedeki elemanlar bir veri kaynağından dinamik olarak yükleniyorsa, asp-items etiket yardımcı özniteliği kullanılabilir. Bunun için asp-items özniteliği seçim listesinde görünen seçenekleri temsil eden bir SelectListItem veya SelectList koleksiyonu veya bu iki koleksiyonu döndüren bir ifade olabilir.



Select Tag Helper

```
public string Sehirler { get; set; }
```

Bir Controller içerisinde aşağıda yer alan 4 farklı şehrde ait bilgiler oluşturulmuş ve ViewBag'e atanmıştır.

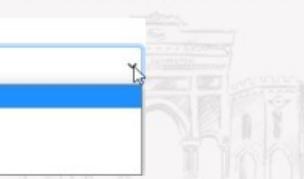
```
public IActionResult Index()
{
    ViewBag.SehirKod = new List<SelectListItem>
    {
        new SelectListItem{Value = "06", Text = "Ankara"},
        new SelectListItem{Value = "34", Text = "İstanbul"},
        new SelectListItem{Value = "35", Text = "İzmir"},
        new SelectListItem{Value = "54", Text = "Sakarya"}
    };
    return View();
}
```

SelectListItem kullanımı için ilgili Namespace eklenmelidir. (`using Microsoft.AspNetCore.Mvc.Rendering;`)

```
<label asp-for="Sehirler" class="control-label"></label>
<select class="form-control" asp-for="Sehirler" asp-items="ViewBag.SehirKod"></select>
```

Sehirler

Ankara
Ankara
İstanbul
İzmir
Sakarya



Hazır Razor Sayfa Şablonları Oluşturma(Scaffolding)

- Web Frameworkler, yazılımcının yazması gereken kod miktarını azaltarak yazılımcıya çeşitli kolaylıklar sağlamaktadır. Bunlardan biri sık tekrarlanan kod yapılarının tasarımları zamanında IDE tarafından otomatik üretilmesidir.
- Scaffolding, otomatik olarak kodlar ekleyen ve projemiz için controller ve View'ler oluşturan bir kod oluşturma çerçevesidir. Veri modeli için oluşturulan Controller ve View'ler geliştiricinin işini kolaylaştırır. CRUD(Oluştur, Oku, Güncelle ve Sil) işlemleri için kodlar ve sayfalar üretir.

View Template'leri

- **Empty:** HTML <body> kısmı boş bir Razor Sayfası oluşturur.
- **Create:** Belirtilen model sınıfını oluşturmak için form öğelerini içeren bir Razor Sayfası oluşturur.
- **Edit:** Belirtilen model sınıfını düzenlemek form elemanlarını barındıran razor sayfası oluşturur
- **Delete:** Seçili varlığın silinmesi için bu varlığa ait hidden bir alan ile bu varlığa ait benzersiz bir değerin tutulduğu bir form üretir.
- **Details:** Seçili varlığın ayrıntılarını görüntüleyen bir razor sayfa üretir.
- **List:** Seçili varlığın tüm örneklerinin ayrıntılarını görüntüleyen bir sayfa üretir



ÖZET

Bu bölümde HTML Formları oluştururken güçlü ve yeni bir özellik olan Tag Helper'larının (Etiket Yardımcıları) ne olduğunu ve bunları nasıl kullanabileceğimizi öğrendik.

Aynı zamanda tag helper'ların derleme zamanında kontrol, model ile bağlanabilme, html görünümü ile kolay uyum, zengin bir IntelliSense desteği ve @ kaçış dizisini kullanmaya gerek olmadan kod yazma gibi avantajlarından bahsederek ASP.NET Core'da yerleşik olarak bulunan ve sık kullanılan tag helper'ları ve bunlar ait özellikleri açıkladık. Son olarak hazır razor şablonlarını kısaca inceledik.



SUNUM PLANI

MODEL VALIDATION (MODEL DOĞRULAMA)

1. Veri Doğrulama Nedir?
2. Model Doğrulama Nedir?
3. İstemci Taraflı Model Doğrulama Nedir ve Nasıl Gerçekleştirilir?
4. Sunucu Taraflı Model Doğrulama Nedir ve Nasıl Gerçekleştirilir?

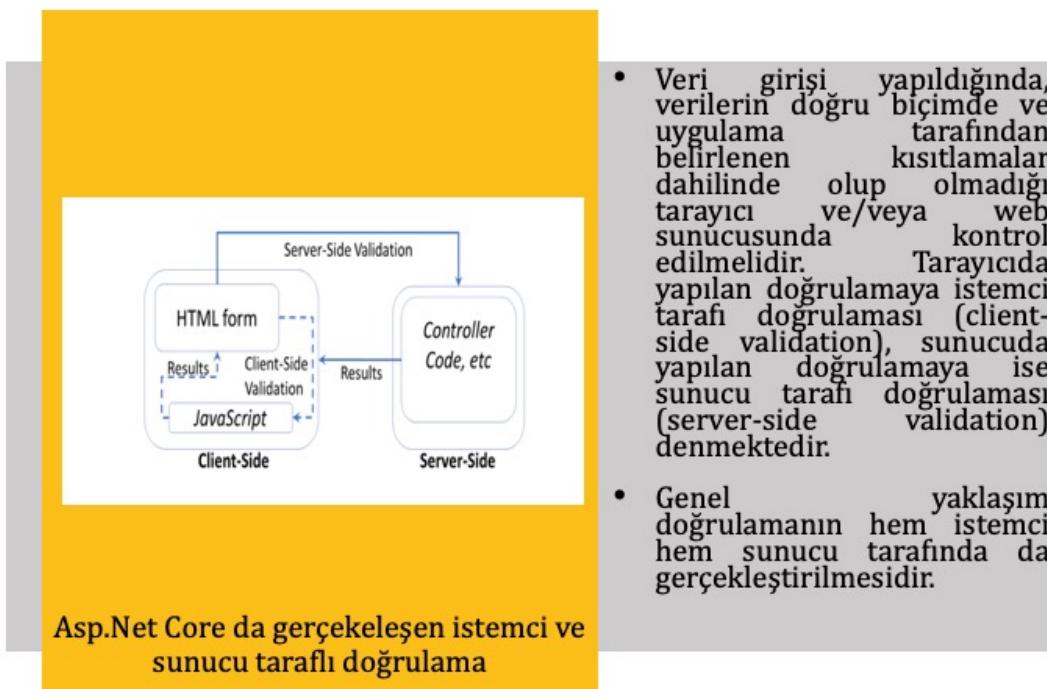
AUZEF





Veri Doğrulama

- Kullanıcılar ile interaktif bir şekilde bir veri paylaşımı gerçekleştirdiğinizde, gelen değerlerin beklenen veri türünde, izin yerilen aralikta ve gerekli olan tüm verilerin mevcut olduğundan emin olunması gereklidir. Bu işlem veri doğrulama olarak bilinir.
- Veri doğrulama için genel yaklaşım, tüm kullanıcı girdilerinin güvenilmez olarak kabul edilmesi yönündedir. Bu yüzden veri, işlenmeden önce kontrol edilip, doğrulanmalıdır.
- Kontrol sürecinin ilk kısmı verilerin kontrol edilerek, istenen kısıtlamalar dahilinde olup olmadığıın kontrolüdür. Bu işlem verinin bütünlüğünü korumanın en önemli yollarından biridir. İkinci kısım ise verinin hatalı olması durumunda gerekli geri bildirimler ile sorunun düzeltmesine yardımcı olmaktadır.



Model Doğrulama

- Model doğrulama, uygulama tarafından alınan verilerin modele bağlanmaya uygun olmasını sağlama sürecidir. Model bağlama HTTP isteklerinden gelen verileri action metoda ait parametrelere otomatik olarak eşler ve model doğrulama ile, istemci ve sunucu tarafı doğrulamasını otomatik olarak gerçekleştirir.
- Model verilerine uygun olarak view tarafında kullanılan tag helper'lar ile istemci tarafında, Controller içinde ilgili action da gerçekleştirilen kontroller ile de sunucu taraflı bir doğrulama gerçekleştirilir.



Model Doğrulama

Doğrulama öznitelikleri, model özellikleri için doğrulama kuralları berlirlenmesini sağlar. Bunun için System.ComponentModel.DataAnnotations; namespace'i modelin tanımlandığı dosyaya eklenmelidir.

Bir öznitelik sınıfı ait hangi özelliğe tanımlanacaksa o özelliğin hemen önünde yer almalıdır ve sadece o özelliğe ait olarak tanımlanmış olur.





Doğrulama için Sık Kullanılan Öznitelikler

- **[Compare]** Bu öznitelik, model sınıfındaki iki özelliğin birbiri ile eşleşip eşleşmediğini tanımlar. (örneğin parola oluştururken parolanın iki kere girilmesi ve bunların eşlenip eşlenmediğinin kontrolü)

```
public string Sifre { get; set; }
[Compare("Sifre",ErrorMessage ="Şifreler eşleşmiyor")]
public string SifreDogrulama { get; set; }
```

- **[EmailAddress]** Özelliğin e-posta adresi biçimine sahip olup olmadığını doğrular.

```
[EmailAddress(ErrorMessage ="Geçerli bir mail adres giriniz")]
public string Email { get; set; }
```

- **[Phone]** Özelliğin geçerli bir telefon numarası biçimine sahip olup olmadığını doğrular. Eğer “-” vb gibi karakterle ayrılmış olarak özelleştirilmiş bir şekilde gösterilmek istenirse uygun bir regular expression ifadesi tanımlanmalıdır.

```
[Phone(ErrorMessage ="Lütfen Geçerli bir telefon giriniz")]
public string Telefon { get; set; }
```



Doğrulama için Sık Kullanılan Öznitelikler

- **[Range(min,max)]** Özellik değerinin belirtilen aralık değerlerinde olup olmadığını doğrular.

```
[Range(1, 120, ErrorMessage = "Yaş aralığı 1-120 olmalıdır")]
public int Yas { get; set; }
```

- **[RegularExpression]** Özelliğin verilen Regular Expression ifadesine uyup uymadığını doğrular.

```
[RegularExpression("^(?0+$)(\\+\\d{1,3}[- ]?)?(?!0+$)\\d{10,15}$",
ErrorMessage = "Telefon numarasını uygun formatta giriniz.")]
public string Telefon { get; set; }
```

- **[Required]** Özelliğin boş veya NULL hariç bir değere sahip olup olmadığını doğrular.

```
[Required(ErrorMessage = "Kullanıcı Adı Alanı Zorunludur")]
public string KullaniciAdi { get; set; }
```

Doğrulama için Sık Kullanılan Öznitelikler

- **[MinLength]** Özellik değeri uzunluğunun belirtilen uzunluk değerinden daha küçük olmaması gerektiğini denetler.

```
[MinLength(10)]  
public string KullanciAdi { get; set; }
```

- **[MaxLength]** Özellik değeri uzunluğunun belirtilen uzunluğu aşmaması gerektiğini denetler. Entity Framework'te de kullanılır. Veritabanındaki sütunun maksimum uzunluğunun ne olacağını da belirler

```
[MaxLength(50)]  
public string KullanciAdi { get; set; }
```

- **[StringLength]** Özellik değeri uzunluğunun belirtilen uzunluğu aşmaması gerektiğini denetler.

```
[StringLength (50)]  
public string KullanciAdi { get; set; }
```

- **[CreditCard]** Özellik değerinin kredi kartı numarasına uygunluğunu denetler.

```
[CreditCard]  
public string CreditCardNumber { get; set; }
```

Doğrulama Öznitelikleri Örnek Kullanım

```
public class Kullanici  
{  
    [Required(ErrorMessage ="Kullanıcı Adı Alanı Zorunludur")]  
    [StringLength(50)]  
    public string KullaniciAdi { get; set; }  
  
    [Required]  
    [EmailAddress(ErrorMessage ="Geçerli bir mail adres giriniz")]  
    public string Email { get; set; }  
}
```

Model Doğrulama Kullanımı

Model doğrulama hem istemci hem sunucu taraflı olarak gerçekleştirilebilir.

İstemci taraflı model doğrulama tag helper'lar yardımı ile JQuery ile gerçekleştirken, sunucu taraflı doğrulama modelin bağlandığı form sunucuya gönderildiğinde sunucu tarafında gelen verilerin kontrolü ile olur.



İstemci Taraflı Model Doğrulama

- ASP.NET Core, istemci tarafı doğrulama kodu eklemeyi kolaylaştıran, doğrulama kütüphanesi içerir. Bunun için istemci tarafı doğrulama kütüphaneleri (*JQuery, jquery.validate & jquery.validate.unobtrusive*) belirtilen sırada projeye dahil edilmelidir.
- Bu komut dosyalarının tüm viewler tarafından kullanılabilir olmaları için Layout dosyasında (*_Layout.cshtml*) ekli olmaları yeterlidir.

```
<script src="~/lib/jquery/dist/jquery.min.js"></script>
```

JQuery, jquery.validate & jquery.validate.unobtrusive yine shared klasörü altındaki *_ValidationScriptsPartial.cshtml* dosyası içinde aşağıdaki gibi eklenmiştir. Bu dosya da *Layout.cshtml* tarafından projeye eklenir.

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
```

```
<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
```





İstemci Taraflı Model Doğrulama

- Modelleriniz için veri doğrulama özniteliklerini kullandığımızda tag helper'lar yardımı ile form, formlara ait öğeler ve bu öğeler için doğrulama verileri otomatik olarak oluşturulabilir.
- Doğrulama mesajı tag helper'i, genellikle herhangi bir doğrulama mesajını görüntülemek için aynı özellik için bir giriş etiketi yardımcısından sonra kullanılır. Modelimizin için hata mesajının görüntülenebileceği tag helper (`` vb) `asp-validation-for` özniteliği ile modele bağlanır

```
<label asp-for="KullaniciAdi" class="control-label"></label>
<input asp-for="KullaniciAdi" class="form-control" />
<span asp-validation-for="KullaniciAdi" class="text-danger"></span>
```

Yukarıdaki kod'da ile Kullanici sınıfında KullaniciAdi adlı özelliğe ait model doğrulama işleminde oluşacak hatalar `` tag helper'ında gösterilecektir.



İstemci Taraflı Model Doğrulama

- Modelleriniz için veri doğrulama özniteliklerini kullandığımızda tag helper'lar yardımı ile form, formlara ait öğeler ve bu öğeler için doğrulama verileri otomatik olarak oluşturulabilir.
- Doğrulama mesajı tag helper'i, genellikle herhangi bir doğrulama mesajını görüntülemek için aynı özellik için bir giriş etiketi yardımcısından sonra kullanılır. Modelimizin için hata mesajının görüntülenebileceği tag helper (`` vb) `asp-validation-for` özniteliği ile modele bağlanır

```
<label asp-for="KullaniciAdi" class="control-label"></label>
<input asp-for="KullaniciAdi" class="form-control" />
<span asp-validation-for="KullaniciAdi" class="text-danger"></span>
```

Yukarıdaki kod'da ile Kullanici sınıfında KullaniciAdi adlı özelliğe ait model doğrulama işleminde oluşacak hatalar `` tag helper'ında gösterilecektir.

Kullanıcı Ekle

Kullanıcı Adı

Kullanıcı Adı Alanı Zorunludur

Kullanıcı Mail

Geçerli bir mail adres giriniz

Oluştur

```
public class Kullanici
{
    [Display (Name = "Kullanici Adı")]
    [Required(ErrorMessage = "Kullanıcı Adı Alanı Zorunludur")]
    [StringLength(50)]
    public string KullaniciAdi { get; set; }

    [Display(Name = "Kullanici Mail")]
    [Required (ErrorMessage ="Mail adresini boş bırakmayınız")]
    [EmailAddress(ErrorMessage = "Geçerli bir mail adres giriniz")]
    public string Email { get; set; }
}
```

Kullanıcı Taraflı Hata Mesajları

Sunucu Taraflı Model Doğrulama

İstemci tarafı doğrulaması istenen formatta verilerin alınmasında tek başına yeterli değildir. İstemci taraflı doğrulama kullanıcı bilgisayarında gerçekleştiğinden çeşitli müdahalelerle etkisiz hale getirilebilir.

Bir öznitelik sınıfı ait hangi özelliğe tanımlanacaksa o özelliğin hemen önünde yer almalıdır ve sadece o özelliğe ait olarak tanımlanmış olur.





Sunucu Taraflı Model Doğrulama

Model bağlama ile form elamanlarına bağlanan model gönderildiğinde, model bağlayıcı, model bağlama ve doğrulamanın sonucuyla ModelState nesnesini günceller.

ModelState, model bağlama ve model doğrulama olarak adlandırılan iki alt sistemden gelen hataları temsil eder.

Doğrulama başarısız olursa model bağlayıcı herhangi bir hata oluşturmaz. Ancak ModelState nesnesini hatalar listesiyle günceller ve bir action metod çağrımadan önce isValid özelliğini false olarak ayarlar.

```
if (ModelState.IsValid)
{
    //Model geçerli. İstenen işlem gerçekleştirilebilir
}
else
{
    //Doğrulama başarısız. Kullanıcıya hata Mesajı verdirilmeli
}
```

ÖZET

Bu bölümde kullanıcıdan alınan verilerin beklenen veri türünde, izin verilen aralıkta ve gereklili olan tüm verilerin mevcut olduğundan emin olunması anlamına gelen veri doğrulama kavramı üzerinde durduk. Veri girişi yapıldığında, verilerin doğru biçimde ve uygulama tarafından belirlenen kısıtlamalar dahilinde olup olmadığını kontrolünü, istemci tarafı doğrulama (client-side validation) ve sunucu tarafı doğrulama (server-side validation) şeklinde olabileceği tartıştık. Uygulamalarımızda veri doğrulama kontrollerinin hem istemci hem sunucu taraflı olması gerektiğini vurguladık.

ASP.NET Core doğrulama çerçevesinin bir parçası olarak kullanılan ve uygulama tarafından alınan verilerin modele bağlanmaya uygun olmasını sağlayan süreci olarak tanımlayabileceğimiz model doğrulama işlemlerinin istemci ve sunucu taraflı nasıl gerçekleştigini inceledik. Ayrıca ufak bir proje ile model doğrulama işlemlerini istemci ve sunucu taraflı gerçekleştirdik.



SUNUM PLANI

LINQ - LANGUAGE INTEGRATED QUERY (DİLE ENTEGRE SORGU)

1. LINQ Nedir?
2. LINQ Avantajları Nelerdir?
3. Sık Kullanılan LINQ Sorğu Operatörleri ve Metotları Nelerdir ve Nasıl Kullanılırlar?

AUZEF



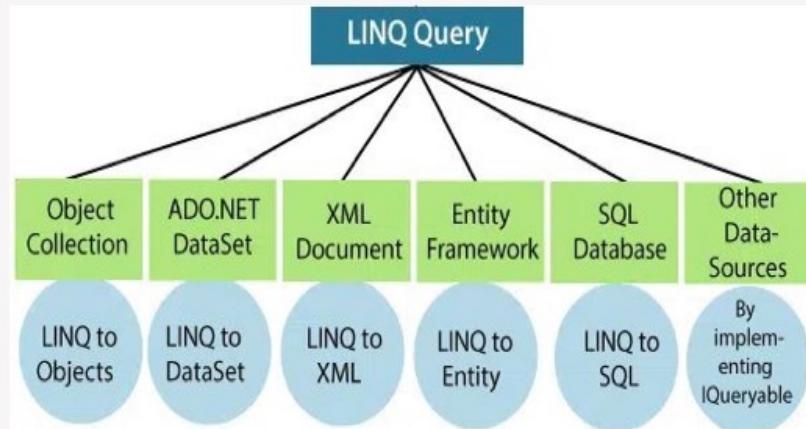
LINQ

Dile Entegre Soru (LINQ-Language-Integrated Query), sorgu yeteneklerinin doğrudan C# dilinin entegrasyonuna dayalı olduğu bir dizi teknolojinin adıdır.

Sorgu, bir veri kaynağından veri veya bilgi talebi isteyen bir ifadedir. Çeşitli veri kaynakları türleri için (ilişkisel veritabanları için SQL ve XML için Xquer vb) farklı farklı sorgu yapıları mevcuttur.

Geleneksel olarak, verilere yönelik sorgular, derleme zamanında veya IntelliSense desteğinde tür denetimi olmaksızın basit string yapıları olarak ifade edilir ve her veri kaynağı türü için farklı bir sorgu dili öğrenilmesi gerekir

LINQ Veri Kaynakları İçin Sorgular





LINQ'nun Avantajları

- Sorgular için derleme zamanı güvenliği ile hata kontrolü sağlar.
- Geliştiricilerin, her bir veri kaynağı veya veri formatı türü için yeni bir sorgu dili öğrenmesi gerekmeyez.
- LINQ, genel koleksiyonlar için IntelliSense (akıllı kod ile metin tamamlaya desteği) sağlar.
- Yazılacak sorgu kod miktarını azaltır.
- LINQ kodlarının kolayca anlaşılması sayesinde diğer geliştiriciler için anlaşılması kolay ve sürdürülebilir bir yapı olmuş olur.
- Birden çok veri kaynağını sorgulamak için Aynı LINQ sözdizimi kullanılabilir.
- C# dili ile entegre olduğu için hata ayıklamayı kolaylaştırır.



LINQ Sorgularının İşletilmesi

Kullanılabilmesi içim **System.Linq** namespace'i projeye eklenmelidir.

Tüm LINQ sorgu işlemleri veri kaynağına erişim, sorgu ifadesi oluşturulması ve sorgunun işletilmesi şeklinde üç farklı eylemden oluşur.

```
string txt = "";
// 1. Veri Kaynağı
int[] sayilar = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

// 2. Sorgu ifadesi.
// SayiQuery IEnumerable<int> tipinde
var SayiQuery =
    from sayi in sayilar
    where (sayi % 2) == 0
    select sayi;

// 3. Sorgunun işletilmesi
foreach (int deger in SayiQuery)
{
    txt = txt + " - " + deger;
}
```



LINQ Sorgularının İşletilmesi

- LINQ ile birlikte lambda ifadesi (`=>`) de kullanılabilmektedir. Lambda ifadesi, bazı özel sözdizim yapıları ile anonim metodlar (anonymous method) oluşturmanın kısa bir yoludur. Örneğin önceki sayfadaki sorgu ifadesi lambda ifadesi ile aşağıdaki gibi daha basit hale getirilebilir.

```
var SayiQuery = sayilar.Where(x => x % 2 == 0);
```

- Burada x ile belirtilen input parametresidir. (x yerine başka bir değişken adda kullanılabilir) `x % 2 == 0` ise bu input parametresi geldikten sonra işletilecek ifadedir.
- Kodun tam olarak anlamı anonim bir fonksiyona x parametre olarak gönderilmiştir. (sayilar dizisi içerisindeki her bir eleman için) ve şart gereği çift sayılar döndürülmüştür.

LINQ Sorgularının Yürütülmesi

- Ertelenmiş Sorgu Yürütme (Deferred Query Execution)

```
public string Deneme()
{
    string txt = "";
    int[] sayilar = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
    var SayiQuery =
        from sayi in sayilar
        where (sayi % 2) == 0
        select sayi;
    sayilar[1] = 8;
    foreach (int deger in SayiQuery)
    {
        txt = txt + " - " + deger;
    }
    return (txt);
}
```

Kodun Çıktısı - 0 - 8 - 2 - 4 - 6

LINQ Sorgularının Yürütülmesi

- Anlık Sorgu Yürütme (Deferred Query Execution)

```
public string Deneme()
{
    string txt = "";
    int[] sayilar = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
    var SayiQuery =
        (from sayi in sayilar
         where (sayi % 2) == 0
         select sayi).ToList();

    sayilar[1] = 8;
    txt = "Toplam Çift Eleman = " + SayiQuery.Count+"\n";
    foreach (int deger in SayiQuery)
    {
        txt = txt + " - " + deger;
    }
    return (txt);
}
```

Toplam Çift Eleman = 4
- 0 - 2 - 4 - 6

LINQ Sorgularının İşletilmesi

IQueryable veri kaynaklarına ve IEnumerable koleksiyonlarına LINQ sorgusu temel olarak Query Syntax(Sorgu sözdizimi) ve Method Syntax(Yöntem sözdizimi) olarak adlandırılan iki farklı yol ile yazılabilir. (Bazı durumlarda bu iki farklı yöntem birleştirilerek de LINQ sorguları oluşturulabilir)

Query Syntax(Sorgu sözdizimi)

```
int[] dizi = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var sonucQuery = from herbirSayi in dizi
                  where herbirSayi > 5
                  select herbirSayi;
```

Method Syntax(Yöntem sözdizimi)

```
int[] dizi2 = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var sonucMethod = dizi2.Where(x => x > 5).ToList();
```

Örnek Üzerinde Sık Kullanılan LINQ Sorgu Operatör ve Metotlarının İncelenmesi

```
public class Kitap
{
    public int KitapID { get; set; }
    public string KitapAdi { get; set; }
    public int KitapSayfasi { get; set; }
}

List<Kitap> Kitaplar = new List<Kitap>()
{
    new Kitap(){KitapID= 1,KitapAdi = "Suç Ve Ceza",YazarAdi="Dostoyevski", KitapSayfasi = 690} ,
    new Kitap(){KitapID= 2,KitapAdi = "Karamazov Kardeşler",YazarAdi="Dostoyevski",KitapSayfasi = 850},
    new Kitap(){KitapID= 3,KitapAdi = "Martin Eden", YazarAdi="Jack London", KitapSayfasi = 520 } ,
    new Kitap(){KitapID= 4,KitapAdi = "İki Şehrin Hikayesi",YazarAdi="Charles Dickens",KitapSayfasi= 400},
    new Kitap(){KitapID= 5,KitapAdi = "Satranç" ,YazarAdi="Stefan Zweig", KitapSayfasi = 90 }
};
```



LINQ- Where Operatörü

Where operatörü veri kümesinde verilen kriterlere göre filtreleme yapar. *Kitaplar* listesinde 500 sayfadan büyük ve 1000 sayfadan küçük kitaplar filtrelenmiştir.

```
var sonuc = from x in Kitaplar
            where x.KitapSayfasi > 500 && x.KitapSayfasi < 1000
            select x;

var sonuc2 = Kitaplar.Where(x => x.KitapSayfasi > 500 &&
                           x.KitapSayfasi < 1000);
```



LINQ- OrderBy- OrderByDescending Operatörü

OrderBy sıralama operatörü sorgu sözdiziminde, koleksiyonun öğelerini artan veya azalan düzende sıralar. Koleksiyonu varsayılan olarak artan düzende sıralar. Koleksiyonu azalan düzende sıralamak için *descending* anahtar kelimesi kullanılır. Yöntem sözdizimi kullanıldığında azalan düzende sıralama *OrderByDescending* operatörü ile gerçekleştirilebilir.

```
var sonuc3 = from x in Kitaplar
              orderby x.KitapAdi
              select x;

var sonuc4 = Kitaplar.OrderBy(x => x.KitapAdi);

sonuc3 ve sonuc4 değişkenlerine İki Şehrin Hikayesi, Karamazov Kardeşler, Martin Eden, Satranç, Suç Ve Ceza ismi doğrultusunda sıralı şekilde kitap nesneleri atanacaktır
```

Kitapların isimlere göre tersten sıralanması sorgu ve yöntem sözdizimleri ile aşağıdaki gibidir.

```
var sonuc5 = from x in Kitaplar
              orderby x.KitapAdi descending
              select x;

var sonuc6 = Kitaplar.OrderByDescending(x => x.KitapAdi);
```

LINQ- Group By Operatörü

LINQ'da gruplama operatörleri, ortak öznitelikleri içeren dizi veya koleksiyonun öğelerini seçerek ve bunları bir grupta sunar. Başka bir deyişle, gruplama operatörüne verilen anahtara göre elemanları gruplar.

Yazar adlarına göre gruplama yapılmak istendiğinde sırayla sorgu ve yöntem sözdizimleri için yazılabilecek kodlar aşağıdadır.

```
var sonuc7 = from s in Kitaplar
              group s by s.YazarAdi;

var sonuc8 = Kitaplar.GroupBy(s => s.YazarAdi);
```

Yukarıdaki örnek için, grup oluşturmak için kullanılan *YazarAdı* grup özellik adı olarak gruba ait key içinde bir kez yer alacaktır.

```
string txt = "";
foreach (var item in sonuc7)
{
    txt=txt+" - "+item.Key ;
```

Foreach döngüsü çalıştırıldığında *YazarAdı*'na gruplama yapıldığından *txt* değişkeninin içinde her bir yazar adı aşağıdaki gibi bir kere yer alacaktır.

- Dostoyevski - Jack London - Charles Dickens - Stefan Zweig

LINQ- Select Operatörü

Select operatörünü SQL'de kullandığımızda hangi sütunları sorguda almak istediğimizi belirtmemize olanak tanıyan bir yapıdaydı. LINQ'da da Select operatörü, istediğimiz bazı özellikleri ya da tüm özelliklerini belirtip seçmemimize olanak tanır.

Örneğin kitap sayfası 500 den fazla olan kitaplara ait sadece yazar adlarını getiren bir sorgu yazılmak istendiğinde sırayla sorgu ve yöntem sözdizimleri için yazılabilecek kodlar aşağıdadır.

```
var sonuc9 = (from x in Kitaplar  
              where x.KitapSayfasi > 500  
              select x.KitapAdi).ToList();  
  
var sonuc10 = Kitaplar.Where(x => x.KitapSayfasi > 500).Select  
                      (x => x.KitapAdi).ToList();
```



LINQ- Contains Metodu

Contain() metodu, koleksiyonda belirtilen bir ögenin olup olmadığını kontrol eder ve bir boolen bir değer döndürür. Aranan öğe koleksiyonda varsa True, yoksa False değeri döndürür.

```
Kitap k = new Kitap(){KitapID = 8, KitapAdi = "x", YazarAdi = "y", KitapSayfasi = 10 };  
Kitaplar.Add(k);  
  
var sonuc12 = Kitaplar.Contains(k); //returns true  
  
var sonuc13 = (from ktp in Kitaplar //returns true  
              select ktp).Contains(k);
```



LINQ- Any-All Metotları

Any() metodu, bir veri kaynağının öğelerinden en az birinin belirli bir koşulu karşılayıp karşılamadığını kontrol etmek için kullanılır. Öğelerden herhangi biri verilen koşulu sağlıyorsa, true döndürür, aksi takdirde false döndürür. Ayrıca bir koleksiyonun veri içerip içermediğini kontrol etmek için de kullanılabilir. Aşağıdaki kod *Kitaplar* listesinde eleman olduğundan *sonuc14* değişkenine true değeri döndürecektr.

```
var sonuc14 = Kitaplar.Any();
```

All() metodu bir veri kaynağının tüm öğelerinin belirtilen koşulu karşılayıp karşılamadığını kontrol eder. Tüm öğeler şartı karşılyorsa true, karşılamıyorsa false değeri döndürür. Aşağıda *Kitaplar* listesindeki tüm kitapların sayfa sayılarının 500-1000 arasında olup olmadığı kontrol edilmiştir. Tüm kitaplara ait sayfa sayıları bu aralıktı olmadığından *sonuc15* değişkeni false değerini alacaktır.

```
var sonuc15 = Kitaplar.All(x => x.KitapSayfasi > 500 && x.KitapSayfasi <1000);
```

LINQ- Count Metodu

Koleksiyonda bulunan tüm öğelerin veya belirli bir koşulu karşılayan öğelerin sayısını döndürmek için kullanılır.

Aşağıdaki kod ile *Kitaplar* listesi içerisinde 500 den küçük sayfa sayısına sahip kitap sayısı geri döndürülmüştür. İlk tanımlı kitaplar listesini düşündüğümüzde *sonuc17* değişkeni 2 değerini alacaktır.

```
var sonuc17 = Kitaplar.Count(x => x.KitapSayfasi < 500);
```

Kitaplar listesinde Count() metodu hiç parametre almadan kullanılsaydı listedeki toplam eleman sayısını verirdi. (Burada 5 değerini alır)

```
var sonuc18 = Kitaplar.Count();
```

LINQ- Max- Min- Sum-Average Metotları

- Max() metodu uygulandığı koleksiyondan en büyük sayısal değeri döndürmek için kullanılır. Kitap listesinde bulunan kitaplardan en yüksek sayfa sayısını veren kod aşağıda yer almaktadır. Burada *sonuc18* değişkeni 850 değerini alır.

```
var sonuc18 = Kitaplar.Max(s=>s.KitapSayfasi);
```

- Min() metodu uygulandığı koleksiyondan en küçük sayısal değeri döndürmek için kullanılır. Kitap listesinde bulunan kitaplardan en az sayfa sayısını veren kod aşağıda yer almaktadır. Burada *sonuc19* değişkeni en az sayfa sayısı olan 90 değerini alır

```
var sonuc19 = Kitaplar.Min(s=>s.KitapSayfasi);
```

- Sum() metodu, koleksiyondaki sayısal öğelerin toplamını döndürmek için kullanılır. *Kitaplar* listesindeki kitaplara ait toplam sayfa sayısı aşağıdaki gibi hesaplanabilir.

```
var sonuc20 = Kitaplar.Sum(s => s.KitapSayfasi);
```

Burada *sonuc20* değişkeni *Kitaplar* listesindeki tüm kitapların sayfa sayıları toplamı olan 2550 değerini alacaktır.

LINQ- First Metodu

Bu yöntem için iki aşırı yüklenmiş sürüm mevcuttur. First metodunun hiç bir parametre almayan hali veri kaynağından ilk öğeyi döndürürken, parametre olarak bir koşul belirtilen ikinci hali belirtilen koşulu karşılayan ilk öğeyi döndürür. İlk durum için veri kaynağı boşsa bu yöntem bir hata fırlatır. İkinci durumda ise koşulu sağlayan herhangi bir öğe yoksa hata fırlatır.

Aşağıda *Kitaplar* listesinde *KitapId*'si çift olan ilk kayıt *sonuc22* değişkenine atanmıştır. Eğer bu şartı sağlayan bir kayıt olmasaydı bu metod hata fırlatır. (Try-Catch-Finaly blokları ile uygulamada meydana gelen hatalar yakalanabilir)

```
var sonuc22 = Kitaplar.First(x => x.KitapID % 2 == 0);
```

LINQ- First Metodu

FirstOrDefault() metodu, First() metodunda olduğu gibi hiç bir parametre almayan hali veri kaynağından ilk öğeyi, parametre olarak bir koşul belirtilen ikinci hali ise belirtilen koşulu karşılayan ilk öğeyi döndürür benzer.

First() metodundan farklı olarak koşul ölçütleri ile eşleşen bir kayıt yoksaFirstOrDefault() boş değerleri işleyebilir ve bir istisna fırlatmaz. Bu metot bunun yerine ögenin veri türüne göre varsayılan değer (null vb) döndürür. First yöntemi gibi,FirstOrDefault yöntemi için de aşırı yüklenmiş parametre almayan ve parametre olarak bir koşul belirtilen iki sürüm vardır.

Aşağıda Kitaplar listesinde KitapID'si 5 ten büyük ilk kayıt aranacaktır. Böyle bir kayıt olmadığından sonuc23FirstOrDefault() metoduundan gelen Null değerini alır.

```
var sonuc23 = Kitaplar.FirstOrDefault(x => x.KitapID > 5);
```



ÖZET

Bu bölümde LINQ(Language Integrated Query) 'nın çeşitli veri kaynaklarından aynı soru sözdizimine sahip verileri alıp kullanamamıza olanak sağlayan teknolojinin adı olduğunu öğrendik.

LINQ ile veri kaynaklarına erişip sorgular yazmanın, derleme zamanında hata kontrolü, IntelliSense desteği, farklı kaynaklar için aynı sözdizim kullanma vb avantajlarının olduğunu tartıştık.

LINQ ile oluşturulan sorguların anlık sorgu yürütme (Immediate query execution) ve ertelenmiş sorgu yürütme (Deferred query execution) olarak ikiye ayırdığını ve yine sorguların Query Syntax(Sorgu sözdizimi) ve Method Syntax(Yöntem sözdizimi) olarak adlandırılan iki farklı yol ile yazılabilğini öğrendik. Son olarak sık kullanılan LINQ sorgu operatör ve metotları örnekler ile inceledik.



SUNUM PLANI

ENTITY Framework Core

1. ORM Nedir?
2. EF Core Nedir Nasıl Kullanılır?
3. Code Fist Yaklaşımı ile Proje Nasıl Oluşturulur?

AUZEF





Veri Depolama

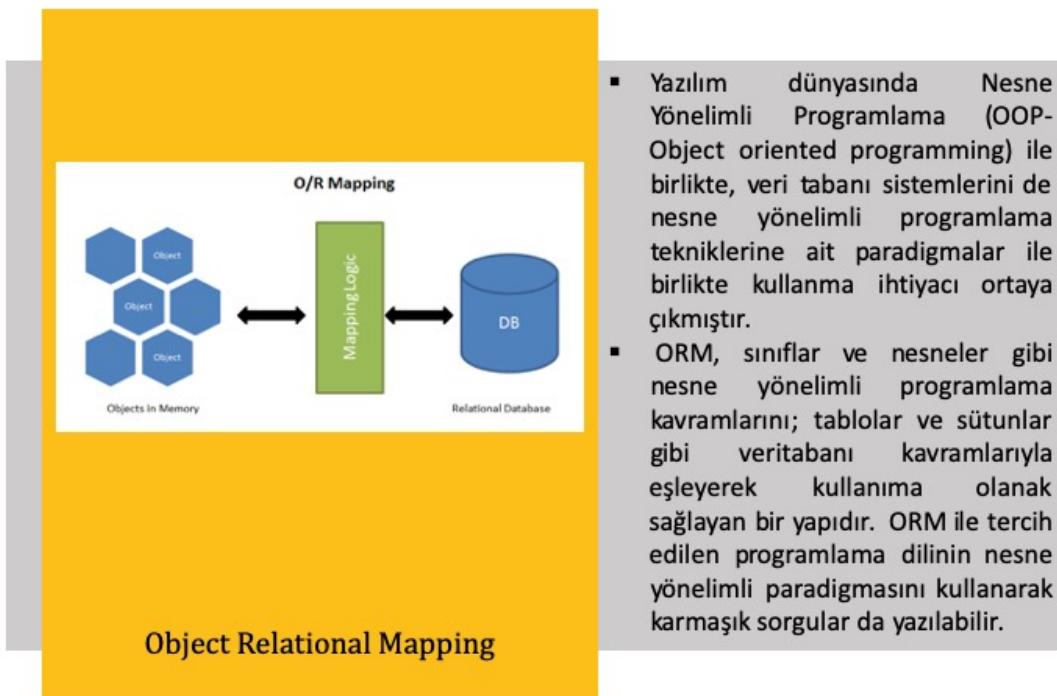
- Yazılım projelerinde çoğu uygulama veri depolamak için bir veritabanına ihtiyaç duyar. Veritabanlarıyla çalışmak, veritabanına olan bağlantıları yönetmek, uygulamanızdaki verileri veritabanının anlayabileceği bir formata çevirip kullanmak ve çok sayıda başka sorunlarla uğraşmak gibi gerekçelerle oldukça zahmetli bir süreç olabilir.
- Günümüz dünyasında verinin artan önemi ile birlikte iyi bir yazılım yazmak, yalnızca iyi yapılandırılmış, verimli koda değil, aynı zamanda bu kodun önemli verileri nasıl yönetip bunlarla etkileşime girebildiğine de bağlıdır. Bu noktada geleneksel yaklaşım ile veri tabanı sunucusuna bağlantı yapıp kodun içerisinde geleneksel olarak SQL (Structured Query Language) kullanarak sorgular yazmak, CRUD (Create, Read, Update, Delete) işlemlerini gerçekleştirmek büyüyen gelişen veriler ve yazılım dünyası yenilikleri ile birlikte çok tercih edilen bir yaklaşım değildir. Bu noktada ORM (Object Relational Mapping) araçları devreye girmiş görece bu işi kolaylaştırmışlardır.



Veritabanı

- Veritabanı; verilerin bir bilgisayar sisteminde elektronik olarak bir sistem dahilinde depolandığı, veriler üzerinde gelişmiş sorgular yapılabildiği yapılandırılmış bir bilgi veya veri topluluğudur.
- Bir veritabanı genellikle bir veritabanı yönetim sistemi (DBMS- database management system) tarafından kontrol edilir.
- Günümüzde çalışmakta olan en yaygın veritabanları türlerindeki veriler, işlemeyi ve veri sorgulamayı verimli hale getirmek için tipik olarak bir tablo ve tablodaki satırlar ve sütunlar halinde modellenir.





ORM (Object Relational Mapping)

ORM'ler, soyutlama ile nesne yönelimli programın bir modelini oluşturur. Verinin nasıl yapılandırıldığını bilmeden bir nesne ile veri arasındaki ilişkiyi haritalama yaparak tanımlar. Veri etkinliklerini yönetmek ve uygulamayı SQL koduna bağlamak için model kullanılabilir. Bir kez bağlamadan sonra kodun tekrar tekrar yazılması gerekmez, bu da geliştiriciye çok büyük bir zaman kazandırır.

Çeşitli programlama dilleri için popüler ORM araçları (Object Relational Mapper) aşağıda listelenmiştir.

PHP: Eloquent ORM, Doctrine, Propel

Python: Django, ORM, SQLAlchemy

Java: Hibernate, Ebean, Java Data Objects

.NET.Core: Entity Framework Core, Dapper, NHibernate

Entity Framework Core (EF Core)

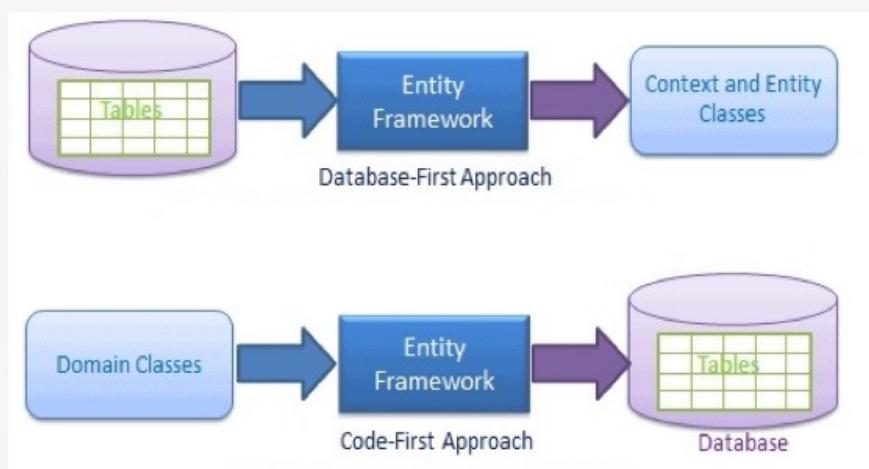
Entity Framework Core, .NET Core uygulamalarıyla kullanılmak üzere tasarlanmış bir ORM aracıdır.

EF Core ilişkisel veritabanı ile .NET yazılım kodları arasında eşleme yapar.

İlişkisel Veritabanı	.NET Core
Table (Tablo)	Class (Sınıf)
Table columns (Tablo kolonları)	Class properties/fields (Sınıf özellikleri, alanları)
Table records (Tablo Kayıtları)	.NET Core koleksiyonlarındaki öğeler, nesneler
SQL örnekleri, WHERE	.NET Core LINQ, where için Linq kodu Where(p =>

Bir veritabanı ve .NET yazılımı arasında EF Core ile eşlenen öğeler

Entity Framework Core (EF Core)



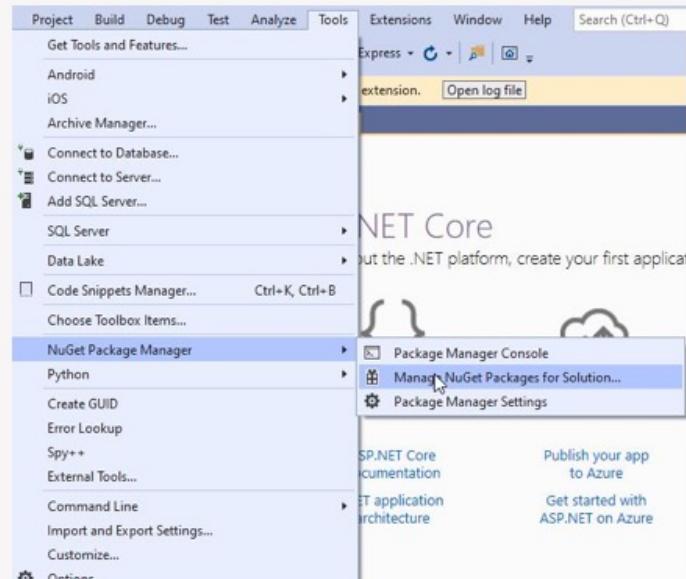
EF Core, code-first ve database-first şeklinde iki yaklaşımı sahiptir.

Entity Framework Core (EF Core)

Veri Sağlayıcı(Data provider), bir veritabanına bağlanmak, komutları yürütmek ve sonuçları almak için kullanılan kütüphanedir. Örneğin, SQL için SQL veri sağlayıcısı, Oracle için Oracle veri sağlayıcısı vb birçok farklı veritabanına erişmek için farklı bir sağlayıcı model kullanılır. Aşağıda, EF Core için veritabanı sağlayıcıları ve NuGet paketleri listelenmektedir. Hangi veritabanı kullanılcaksa ilgili EF Core paketi NuGet paket yöneticisi kullanılarak projeye yüklenmelidir.

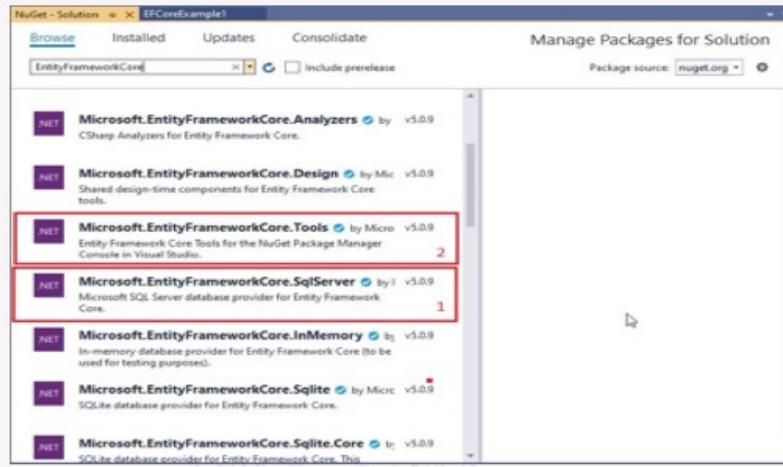
SQL Server	: Microsoft.EntityFrameworkCore.SqlServer
MySQL	: MySql.Data.EntityFrameworkCore
PostgreSQL	: Npgsql.EntityFrameworkCore.PostgreSQL
SQLite	: Microsoft.EntityFrameworkCore.SQLite
SQL Compact	: EntityFrameworkCore.SqlServerCompact40
In-memory	: Microsoft.EntityFrameworkCore.InMemory

Nugget Package Manager



EF Core için Gerekli Paketlerin Yüklenmesi

Microsoft.EntityFrameworkCore.SqlServer paketi ile beraber EF Core komutlarını yürütmek için DB sağlayıcı paketiyle birlikte migrations, scaffolding vb işlemler için EF Core Tools paketini de projemize yüklememiz gereklidir.



EF Core'da Data Annotations Kullanımı

System.ComponentModel.DataAnnotations

- Key**: Bir varlıktaki özelliği primary key yapmak için kullanılır. Varsayılan olarak sınıf özelliğinin isminde Id geçen özellik key tanımlanmasa da primary key olarak ayarlanır. Key özniteliği kullanıldığından bu varsayılan kuralı geçersiz kılar.
`[Key]`
`public int OgrId { get; set; }`
- TimeStamp**: SQL Server veritabanında zaman damgası veri tipine sahip bir sütun oluşturur. EF, veritabanındaki UPDATE deyiminde eşzamanlılık kontrolünde bu Zaman damgası sütununu otomatik olarak kullanır.
`[Timestamp]`
`public byte[] Timestamp { get; set; }`
- Required**: EF'de uyguladığı bir özellik için veritabanı tablosunda bir NOT NULL sütunu oluşturur.(Veri doğrulama da kullanıldığını unutmayın)
`[Required]`
`public string KtakAdi{ get; set; }`
- MaxLength**: Veritabanında ki ilgili sütunda izin verilen maksimum string uzunluğunu belirtmek için bir özellîte uygulanabilir.
`[MaxLength(50)]`
`public string StudentName { get; set; }`



EF Core'da Data Annotations Kullanımı

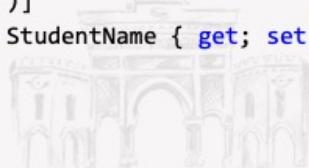
System.ComponentModel.DataAnnotations.Schema'da tanımlı bazı öznitelikler

Table : Varlık sınıfına uygulandığında veri tabanındaki ilgili tablo ve şema adı yapılandırılabilir. Aşağıdaki örnek için veri tabanında tablo ismini Student yapar.

```
[Table("Student")]
public class Ogrenci
{
    public int OgrId { get; set; }
    public string OgrAd { get; set; }
}
```

Column : bir varlık sınıfındaki özelliğe uygulandığında bir veritabanı tablosundaki ilgili sütun adı, veri türü ve sırası yapılandırılabilir.

```
[Column("Name")]
public string StudentName { get; set; }
```



EF Core'da Data Annotations Kullanımı

System.ComponentModel.DataAnnotations.Schema'da tanımlı bazı öznitelikler

- **ForeignKey**: İlişki kurulacak tabloda özellikle bir alanın yabancı anahtar(foreinkey) olması istendiğinde ForeignKey anahtar kelimesi ile birlikte ilgili gezinme özelliği (navigation property) adı parametre olarak belirtilebilir.(Burada Yazar isimli navigation property üzerinden Yazar tablosuna ait primary key ile eşleme olacaktır.)

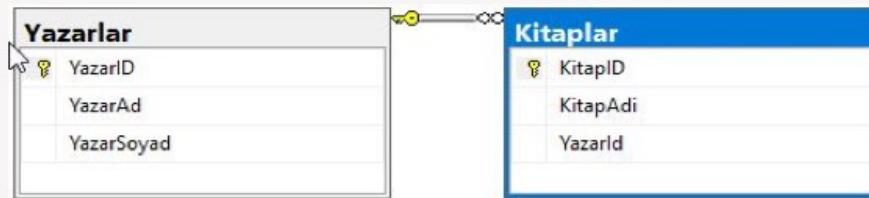
```
[ForeignKey("Yazar")]
public int YazarNo { get; set; }
public Yazar Yazar { get; set; }
```

- **NotMapped**: Modelde tanımlanıp veritabanında oluşturmaması gereken bir özelliğe uygulanır.

```
[NotMapped]
public int OgrYas { get; set; }
```



Code-First Yaklaşımı Örnek Proje



KitapID	KitapAdi	YazarID
1	Martin Eden	1
2	Beyaz Diş	1
3	Suç ve Ceza	2
4	Budala	2
5	Karamazov Kar...	2

YazarID	YazarAd	YazarSoyad
1	Jack	London
2	Fyodor	Dostoevsky

Veri Modeli Oluşturma

```
public class Yazar
{
    public int YazarID { get; set; }
    [Required]
    [MaxLength(100)]
    [Display(Name = "Yazar Ad")]
    public string YazarAd { get; set; }
    [Required]
    [MaxLength(100)]
    [Display(Name = "Yazar Soyad")]
    public string YazarSoyad { get; set; }
    public ICollection<Kitap> Kitaplar { get; set; }
}
```

Veri Modeli Oluşturma

```
public class Kitap
{
    [Key]
    public int KitapID { get; set; }
    [Required]
    [MaxLength(100)]
    [Display(Name = "Kitap Adı")]
    public string KtakAdi{ get; set; }
    public int YazarID { get; set; }
    public Yazar Yazar { get; set; }
}
```



Navigation Properties

Entity Framework modelindeki varlıklar arasındaki ilişkiler, Navigation Properties (Gezinme Özellikleri) tarafından tanımlanır. Navigation Properties, kullanılan veritabanı sağlayıcısının herhangi bir türle eşlenemeyeceği bir özelliktir. Yukarıda ki her iki sınıf da incelediğinde bu sınıflar içerisinde int, string vb mevcut veritabanı türleriyle eşlenebilen özellikler olduğu gibi *Kitap* sınıfında *Yazar*, *Yazar* sınıfında *Kitap* olarak tanımlı özellikler de mevcuttur. Bu özellikler için veritabanlarında eşdeğer bir tür yoktur. Bunlar Navigation Properties olarak görülürler.

Yazar sınıfının tanımında her yazara ait *ICollection<Kitap>* *Kitaplar* koleksiyon gezinme özelliği ile (*Kitaplar Collection navigation property*) birden çok kitaba sahip olmasına olmak tanır. Bire çok ilişkideki asıl varlık, koleksiyon gezinme özelliğine(*collection navigation property*) sahip olan varluktur ve bağımlı varlık, başvuru gezinme özelliğine (*reference navigation property*) sahip varluktur. Yani *Kitap* sınıfındaki *public Yazar Yazar { get; set; }* şeklinde tanımlanan *Yazar* Navigation Property tanımı yapılmasa da olur. Burada işlem kolaylaştırması açısından her kitabın en fazla bir yazara sahip olmasına izin verir.



DbContext Oluşturma

Veri tabanı ile veri modelleri arasındaki koordinasyonu sağlayan sınıfıdır. Temel olarak aşağıdaki işlemlerden sorumludur.

- Veritabanı bağlantısını yönetme
- Varlıklar ve aralarındaki ilişkileri yapılandırma
- Veritabanındaki verileri okuma, oluşturma, güncelleme, silme
- Değişiklik izlemeyi yapılandırma
- Verileri önbelleğe alma
- Transaction yönetimi

Veritabanındaki tabloların nesne yönelimli temsilleri olan varlık sınıfları, veritabanında saklamak istediğiniz verileri temsil ederler. EF Core'u yapılandırmak, yönetmek, çalışma zamanında veritabanına erişmek vb için uygulamalarda DbContext kullanılır. DbContext'i uygulamamızda kullanmak için, DbContext'ten türetilen ve bu bağlam sınıfı içerisinde genellikle modeldeki her varlık için DbSet< TEntity > özelliklerini içeren bir sınıf oluşturulmalıdır.

DbContext Metodları

Sık Kullanılan DbContext Metodları aşağıda listelenmiştir. (NOT
örnekler KitaplikContext k = new KitaplikContext(); e göre verilecektir)

Add: DbContext'e yeni bir varlık ekler ve onu izlemeye başlar. Bu yeni varlığa ait veriler SaveChanges() çağrılığında veritabanına eklenir.

```
k.Add(kitap);  
k.SaveChanges();
```

Remove: Belirtilen varlık üzerinde silme işlemini gerçekleştirir. Bu silme işleminin veritabanında da olması için SaveChanges() metodu çağrılmalıdır.

```
k.Remove(kitap);  
k.SaveChanges();
```

Update: Modifiye edilmiş bir varlığa ait değişikliği yapar ve varlığı izlemeye başlar. SaveChagnes() çağrılığında veriler kaydedilir.

```
k.Update(kitap);  
k.SaveChanges();
```

DbContext Metodları

Find: Parametre olarak verilen birincil anahtar değerine sahip bir varlık bulur.

```
k.Kitaplar.Find(1);
```

FindAsync: Parametre olarak verilen birincil anahtar değerlerine sahip bir varlığı bulmak asenkron bir yöntemdir.

SaveChanges: Ekleme, silme ve değiştirme(Add,Remove,Update) işlemi yapılmış varlıklar henüz veritabanına kaydedilmemiştir. Bu durumda varlıklar için veritabanına INSERT, UPDATE veya DELETE komutunu çalıştırır.

SaveChangesAsync: Asenkron olarak kullanılan işlemlerde SaveChanges() ile aynı işlemi gerçekleştirir.

KitaplikContext Sınıfı

```
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

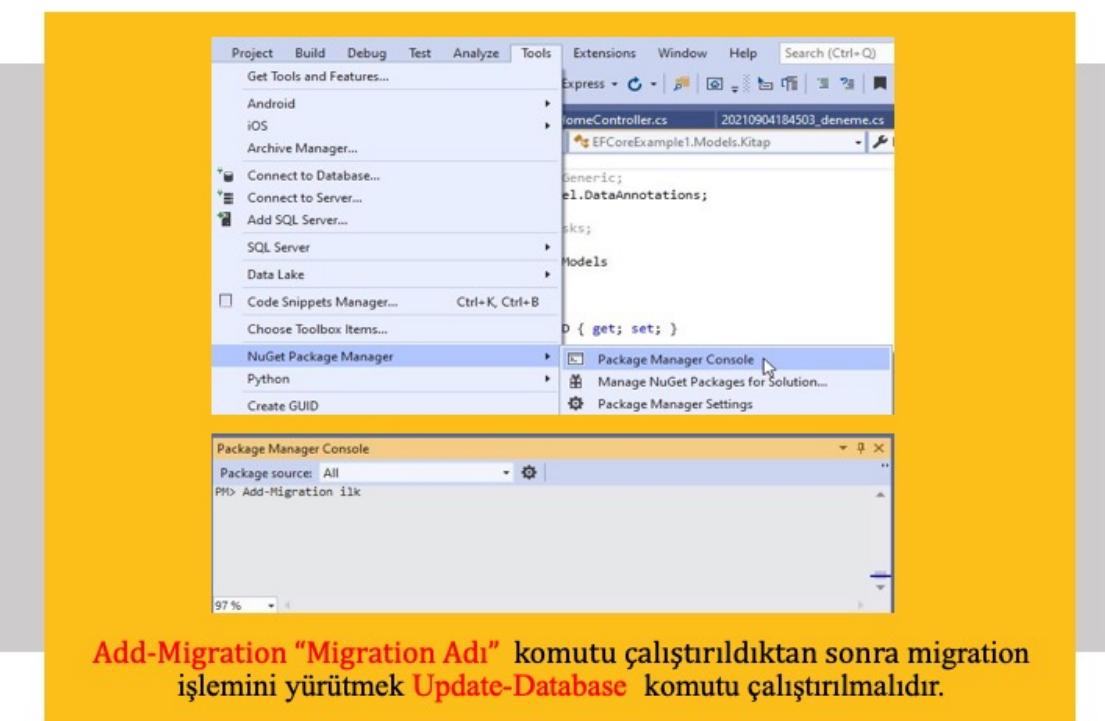
namespace EFCoreExample1.Models
{
    public class KitaplikContext: DbContext
    {
        public DbSet<Kitap> Kitaplar { get; set; }
        public DbSet<Yazar> Yazarlar { get; set; }

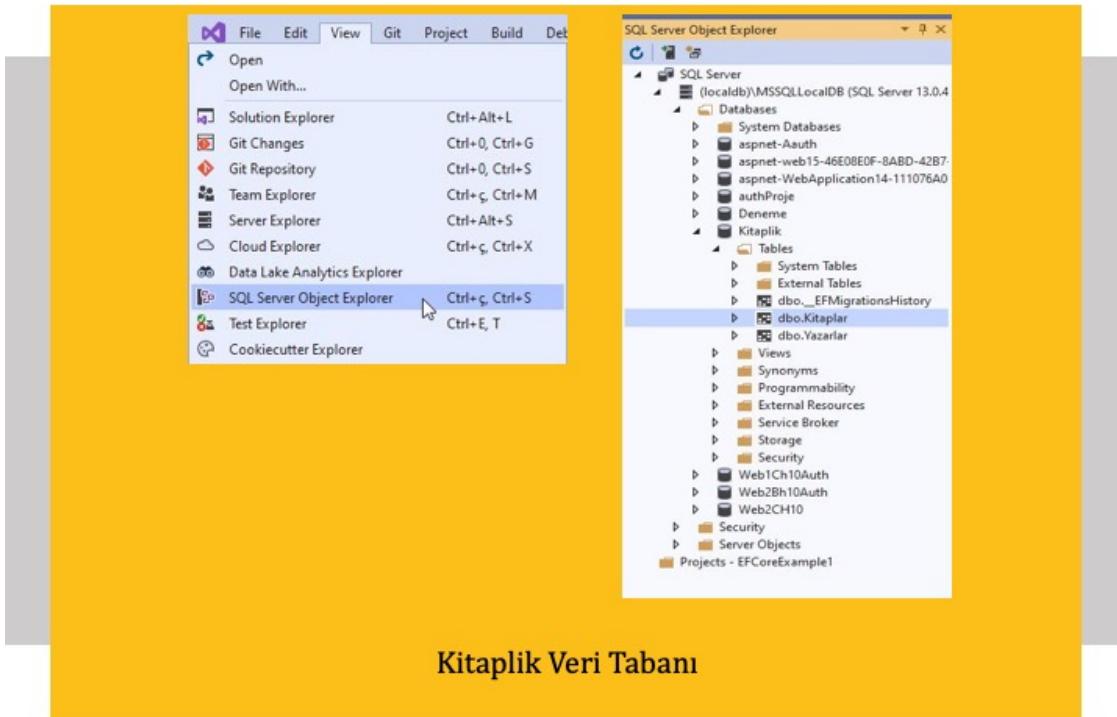
        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;
Database=Kitaplik;Trusted_Connection=True;");
        }
    }
}
```

Migration Oluşturma

Code first yaklaşımı ile model ilk oluşturulduğunda veritabanı ile modelin senkron edilmesi gereklidir. Aynı zamanda uygulamalar geliştirilmeye devam ettikçe yeni gereksinimler ve güncellemeler ile birlikte modelin sık sık değişmesi muhtemeldir. Veritabanının modelle senkronize tutulması gereklidir.

Migration özelliği ile, modelimizin o anki durumu veritabanı ile senkron edilir. Veritabanı olmadığından, ilk migration oluşturulduğunda KitaplikContext'te DbSet özellikleri tarafından temsil edilen Kitaplar ve Yazarlar için tablolar eklenecektir.





Model Güncelleme

Artan ihtiyaçlar ve çeşitli bakım sebepleriyle çoğu uygulamada güncelleme yapmak kaçınılmazdır. Modellerimize yeni özellikler eklemek, yeni varlıklar eklemek, bazı sınıfları kaldırmak vb durumların zamanla oluşması muhtemeldir.

```
public class Kitap
{
    public int KitapID { get; set; }
    [Required]
    [MaxLength(100)]
    [Display(Name = "Kitap Adı")]
    public string KtakAdi{ get; set; }

    [Display(Name = "Kitap Sayfa Sayısı")]
    [Range(10,5000)]
    public int KtakSayfasi{ get; set; }

    public int YazarID { get; set; }
    public Yazar Yazar { get; set; }
}
```

Add-Migration KitapSayfasiEkleme
Update-Database

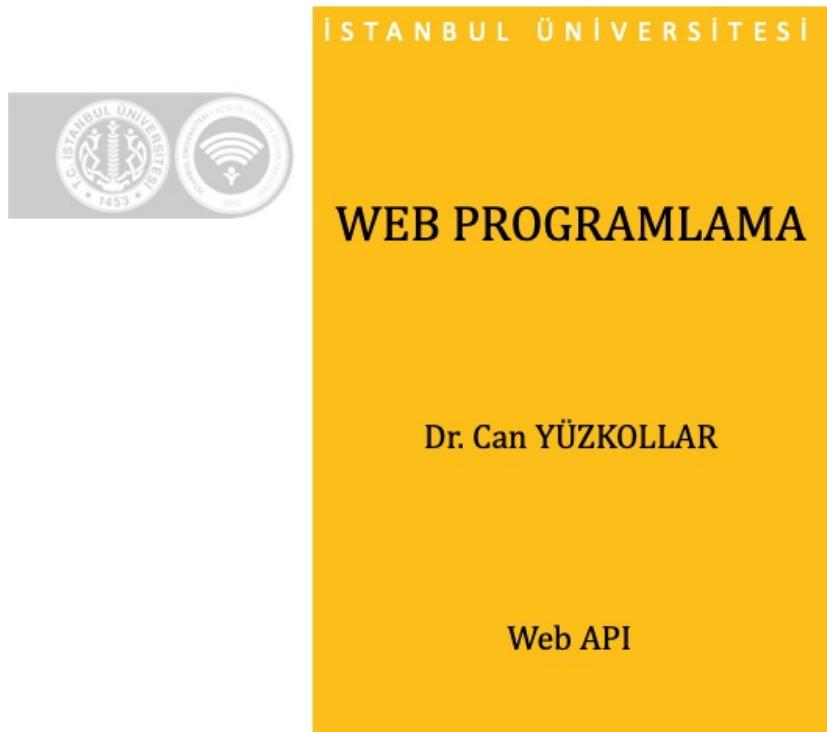
ÖZET

Bu bölümde veri tabanı sistemlerini nesne yönelimli programlama tekniklerine ait paradigmalar ile birlikte kullanma ihtiyacı ile ortaya çıkan ORM'lerden bahsettim. ORM'lerin sınıflar ve nesneler gibi nesne yönelimli programlama kavramlarının; tablolar ve sütunlar gibi veritabanı kavramlarıyla eşleşme yaparak kullanımına olanak sağlayan bir yapı olduğunu tartıştık.

.Net Core'da yaygın bir ORM aracı olan Entity Framework Core'da veri tabanı işlemleri için gerçekleştirilen Code First ve Database First yaklaşımlarını inceledik. Projede kullanılacak veri tabanına uygun şekilde ilgili EF Core paketinin NuGet paket yöneticisi ile projeye eklenmesi gerektiğini tartıştık.

Code-First Yaklaşımı ile örnek bir proje geliştirerek veri modelimizdeki sınıflar içerisinde int, string vb mevcut veritabanı türleriyle eşlenebilen özellikler veritabanında ki olmasını istediğimiz özellikler çerçevesinde data annotation'lar ile yapılandırmayı inceledik. Veritabanlarında eşdeğer bir türü olmayan Navigation Properties olarak adlandırılabilir yapıları tartıştık. Veri tabanı ile veri modelleri arasındaki koordinasyonu sağlayan sınıf olan DbContext sınıfının özelliklerini inceleyerek örnek proje için bu sınıfı yapılandırdık. Code first yaklaşımı ile model ilk oluşturulduğunda veritabanı ile modelin senkron edilmesini sağlayabildiğimiz, uygulamalar geliştirmeye devam ettikçe yeni gereksinimler ve güncellemeler ile birlikte model ile beraber veritabanının güncellenmesini de gerçekleştirebileceğimiz Migration özelliğinden bahsettim.



This is a slide titled "SUNUM PLANI" (Presentation Plan) for the topic "Web API". The slide features a yellow central area with the title "Web API" and a list of five questions. To the left is a grey sidebar, and to the right is a large yellow chevron-shaped graphic pointing towards the list.

SUNUM PLANI

Web API

1. Web API Kavramı Nedir ?
2. SAOP ve Rest Kavramları Nelerdir ?
3. Rest API Mimarisi Nedir ?
4. HTTP Protokolü- RestFull API İlişkisi Nasıldır?
5. Asp.Net Core'da Web API kullanımı Nasıl Gerçekleştirilir ?



AUZEF

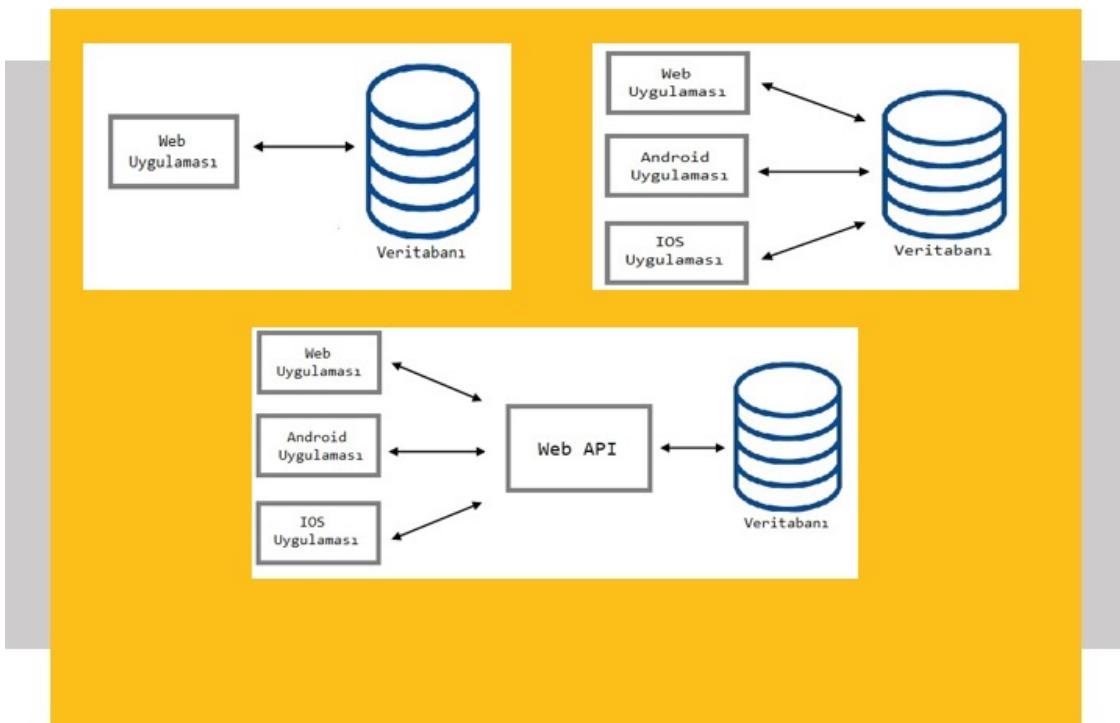
API Kavramı

API(Application Programming Interface-Uygulama Programlama Arayüzü), programcılar bir uygulamanın, işletim sisteminin veya diğer hizmetlerin belirli özelliklerine veya verilerine erişmesine izin veren bir dizi işleve sahip bir tür arayüzdür.

Bilgi almak veya bir işlevi gerçekleştirmek için bir bilgisayar veya sistemle etkileşim kurmak istendiğinde, API; isteği anlayıp yerine getirebilmek için o sistemle iletişime yardımcı olur.

Web API, bir web sitesinin verilerini bir bilgisayar için anlaşılabilir hale getiren araçtır.

AUZEF



SOAP

Bilgi alışverisini standartlaştırmaya yardımcı olmak için SAOP olarak bilinen Basit Nesne Erişim Protokol(Simple Object Access Protocol) adlı bir protokol tanımlanmış ve geliştirilmiştir.

SAOP ile tasarlanan servisler, mesaj biçimleri için XML kullanır ve istekleri HTTP veya SMTP aracılığıyla alır. SOAP, farklı ortamlarda çalışan veya farklı dillerde yazılmış uygulamaların bilgi paylaşımını kolaylaştırır. SOAP servislerinin kullanım yöntemi uzaktan bir servisin çağrılması RPC(Remote Process Call) şeklindedir. Güvenlik protokollerini içerisinde barındırır, durum bilgisini request ve response'larda saklar.



Rest / RerstFull API'ler

Web API kullanımında bir başka tanım Temsili Durum Aktarımıdır (Representational State Transfer - REST). REST mimari kısıtlamalarına uyan Web API'lerine RestFull API'ler denir. REST bir mimari tarz iken, SOAP bir protokoldür. Bu durum, Restful web API'leri için resmi bir standart olmadığı anlamına gelir.

SOAP'da istemci bağlantı kuracağı servisle ilgili herşeyi bilmek zorundayken, REST mimarisinde REST servisler bu servisin yapısını ve detayını bilmek zorunda olmadan doğrudan bir URL çağırarak çalışır, arada ek bir bileşen veya protokol kullanılmaz. Bu sebeple daha esnek ve kullanımı daha kolaydır.

RestFull API Kısıtlamaları

- İstemci-Sunucu Mimarisi (Client-server architecture)
- Tek Tip Arayüz (Uniform Interface)
- Durumsuzluk(Stateless)
- Önbellekleme (Cacheable)
- Katmanlı Sistem (Layered System)
- İstek Üzerine Kod (Code on Demand)

API'ler için Veri Formatları

Genellikle istek mesajları için HTTP kullanan web API'ler yanıt mesajlarının yapısının tanımını sağlarlar. Text ve csv gibi birçok farklı yapıda dönüş tipi kullanılabilir olsa da yanıt mesajları genellikle farklı uygulama türleri için kolayca anlaşılıp işlenebilen format olan XML ve JSON (JavaScript Object Notation) biçimindedir. Web API'ler varsayılan olarak JSON ve XML verilerini işler.



JSON Veri Yapısı

JSON web üzerinde veri alışverişi için en yaygın kullanılan veri formatıdır. Birçok API, JSON'u bir biçim olarak benimsemiştir. Tüm modern programlama dilleri (Java, JavaScript, C#, PHP, Python vb) ve uygulama platformları, JSON verilerinin üretilmesi (serializing) ve tüketilmesi (unserializing) için desteği sahiptir.

JSON, anahtar(key) ve değer(value) şeklinde iki parçadan oluşan çok basit bir format yapısına sahiptir. Anahtar-değer çifti, anahtarın ardından iki nokta üst üste işaret ve ardından değerin geldiği belirli bir sözdizimi ile belirtilir ve bu çiftler virgülle ayrılmıştır.

```
{  
    "Ad": "Ömer",  
    "Numara": 100,  
    "Durum": 1  
}
```



XML Veri Yapısı

XML (eXtensible Markup Language -Genişletilebilir İşaretleme Dili) JSON gibi API'ler de verilerin yapılandırılması için de kullanılabilen bir veri formatıdır.

XML, html gibi bir işaretleme dilidir. Html'de yer alan etiketlerden farklı olarak XML için etiketler dokümanı oluşturan kişiler tarafından belirlenip oluşturulur. Etiketler oluşturulurken hiyerarşik bir yapıda oluşturulmalı, açılan etiketler kapatılmalıdır. Ayrıca XML etiketleri büyük-küçük harf duyarlıdır ve etiketlere nitelik atanabilir. Ana bloğa kök(node) denir ve XML dokümanları her zaman bir kök düğümle başlar.

```
<Ogrenci>
  <Ad>Ömer</Ad>
  <Numara>100</Numara>
  <Durum>1</Durum>
</Ogrenci >
```

Http Protokolü- İstek

İstek-yanıt döngüsü adı verilen bir kavram dahilinde gerçekleşen http protokolünde geçerli bir http isteği için istemcinin aşağıda yer alan dört öğeyi içermesi gereklidir.

URL : Bir URL, Web'deki belirli bir benzersiz kaynağın adresidir. Teoride, her geçerli URL benzersiz bir kaynağı işaret eder. API'ler, kaynak adı verilen URL'ler ile çeşitli durumlar için istemcinin sunucuya etkileşim kurmasını sağlarlar.

Method-Verb: İstek yöntemi, sunucuya, istemcinin sunucunun ne tür bir işlem yapmasını istediğini söyler. Her istek bir HTTP metoduna sahip olmalıdır. API'lerde en sık kullanılan dört yöntem Get, Put, Post ve Delete'tir. Bu yöntemler ileride ayrıntılı olarak açıklanacaktır.



Http Protokolü-İstek

Header(s)(Başlık): Bir http isteğinde header bilgisi istek veya yanıtla ilgili bir bilgi listesidir. Header bölümünde verilerin hangi formatta olduğunu söyleyen bir veri vardır. (Content-Type). İstemci sunucuya JSON verilerini göndermek isterse, içerik türünü *Content-Type "application/json"* olarak ayarlayacaktır. Sunucu isteği alıp içerik türünü değerlendirdikten sonra, sunucu tarafından verilerin nasıl okuyacağını bilinecektir. Benzer şekilde, sunucu tarafından istemciye gönderilen yanıtta, yanıtın gövdesini nasıl okuyacağını belirtmek için Content-Type ayarlanacaktır.

Body(Gövde): İstek gövdesi, istemcinin sunucuya göndermek istediği verileri içerir. Sunucuya gönderilmek istenen her istek, gönderilmek istenen verileri içeren bir gövde verisine sahip olmalıdır. HTTP protokolü katı bir yapı gerektirdiği metot, url ve header için katı bir sınırlama getirirken, body (gövde) ile istemcinin ihtiyaç duyduğu her şeyi göndermesine izin verir.



HTTP Protokolü-Yanıt

Sunucu, istemciden bir istek aldıktan sonra, isteği yerine getirmeye ve istemciye bir yanıt döndürmeye çalışır. HTTP yanıtları, isteklere çok benzer bir yapıya sahiptir. Temel fark, bir yöntem ve bir URL yerine yanıtın bir durum kodu içermesidir. HTTP yanıtı aşağıdaki bileşenleri içerir.

HTTP Durum Kodları (HTTP Status Code): Her bir yanıt bir durum koduna sahip olmalıdır.

Yanıt Başlığı (Response Headers): Yanıtlarda bir veya daha fazla yanıt başlığı olabilir.

Veri (Data): İstemciye dönen veriyi belirtir.



HTTP Durum Kodları (Status-Code)

Bilgilendirici Yanıtlar: 100-199 arasında değerler alır.

- 100 - Devam: İstemci talebi işleniyor.
- 101 - Protokol Değiştirme: İstemci protokol tipini değiştirmek istediler ve sunucu kabul etti.
- 102 - İşleme: İşleme normalden daha uzun sürüyor.

Başarılı Yanıtlar: 200-299 arasında değerler alır.

- 200 - Tamam: İstek ve yanıt başarılı bir şekilde gerçekleştirildi.
- 201 - Oluşturuldu: Sunucuya gerçekleştirilen isteğin başarılı olması ile sunucuda yeni bir kaynak oluşturuldu.
- 202 - Kabul Edildi: İstemci sunucuda bir şey oluşturmak istediler. Bu istek sunucu tarafından kabul edildi, fakat henüz işletilmemişti.
- 204 - İçerik Yok: İstek alındı, ancak istemciye henüz hiçbir veri gönderilmedi.

Yönlendirme Mesajları: 300-399 arasında değerler alır.

- 301 - Kahçı Olarak Taşındı: Bir kaynağın başka bir yere taşındığını bildirir. İsteğin başka bir kaynağa yönlendirilir.
- 302 - Geçici Taşındı: Aranılan kaynağın geçici olarak verilen URL'ye taşındığı anlamına gelir.

HTTP Durum Kodları (Status-Code)

İstemci Hata Yanıtları: 400-499 arasında değerler alır.

- 401 - İzinsiz: Eğer erişimi olmayan kaynağa erişmeye çalıştığını ifade eder.
- 403 - Yasak: İstekte bulunan kaynağın yasaklandığını belirtir.
- 404 - Bulunamadı: İstek yapılan kaynağın bulunamadığını belirtir. 404 sayfaya yönlendiren bağlantılar genellikle bozuk veya ölü bağlantılar denir.
- 408 - İstek Zaman Aşımı: Belirli bir sürede isteğin tamamlanmadığını belirtir.

Sunucu Hata Yanıtları: 500-599 arasında değerler alır.

- 500 - Dahili Sunucu Hatası: Sunucuda bir hata oluştu ve istek sunucu tarafından karşılanmadı.
- 502 - Geçersiz Ağ Geçidi: Gateway veya Proxy sunucusu, kaynağın bulunduğu sunucudan cevap alamadı.
- 503 - Hizmet Kullanılamıyor: Sunucunun şuan hizmet veremedğini belirtir.
- 504 - Ağ Geçidi Zaman Aşımı: Gateway veya Proxy sunucusu, kaynağın bulunduğu sunucudan belirli bir zaman içinde cevap alamadı.

HTTP Metotları ve CRUD İşlemleri

Web üzerinden sunulan REST API'lerinin doğası gereği, bu API'ler, GET, POST, DELETE ve PUT gibi HTTP protokolüne ait metotlar kullanarak istemcilerle iletişim kurar. Bu noktada, HTTP metotları ile CRUD işlevleri arasında bir örtüşme tanımlanmıştır. Her API, POST, GET, PUT, PATCH, DELETE gibi standart bir HTTP istek metodu ile çağrırlır.

CRUD	SQL	HTTP METOT
CREATE	INSERT	POST
READ	SELECT	GET
UPDATE	UPDATE	PUT/PATCH
DELETE	DELETE	DELETE

Temel Http Metotları İşlem Örneği

```
[Route("api/[controller]")]
[ApiController]
public class OGRENCIController : ControllerBase
```

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 isteğin gövde kısmında)
POST	http://localhost/api/Ogrenci/	Yeni bir öğrenci ekle (Öğrenci verileri http isteğin gövde kısmında)

Asp.Net Core Web API- Örnek API Controller

```
[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)
    {
    }
}
```

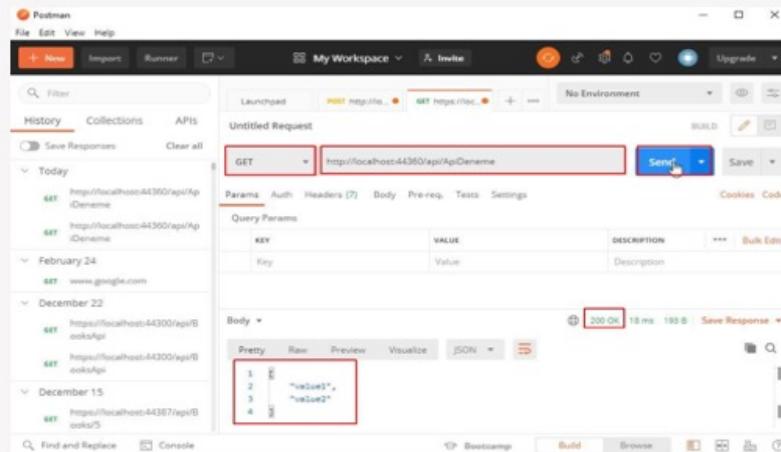
API Controller

Önceki slaytta görüldüğü gibi oluşturulan *ApiDenemeController*, varsayılan olarak *ControllerBase* sınıfından türetilmiştir.

[ApiController] özniteliği Controller'ın web API isteklerine yanıt verdiği gösterir. Ayrıca, bu öznitelik, ASP.NET Core için kullanıma hazır doğrulama ve model bağlama sağlar. Bu da her action'da ModelState.IsValid özniteliğini açıkça kontrol etmemize gerek olmadığı anlamına gelir. Model doğrulama hatalarının otomatik olarak bir HTTP 400 yanıtını tetiklemesini de bu öznitelik gerçekleştirir.

Route özniteliği varsayılan template olarak api/[controller]" şablonunu kullanır. Burada ApiDenemeController [Controller], çalışma zamanında ApiDenemeController metniyle (controller'in kendi adı) yer değiştirilecek özel bir adlandırma kuralı sağlar. (örnek için [Route("api/ApiDeneme")] şecline çevirir)

Postman ile API Kullanımı



Yukarıda Postman ile Get isteği yapılmıştır. Url ise projemizde yer alan API'ye aittir. <http://localhost:44360/api/ApiDeneme>

Bu noktada *ApiDenemeController*'da Get методу çağrılmış olur. ASP.NET Core MVC, yanıt verilerini biçimlendirme desteğine sahiptir. Yanıt verileri, belirli biçimler kullanılarak veya istemcinin talep ettiği biçimde yanıt olarak biçimlendirilebilir.

EF Core Projesinde API Kullanımı

```
[Route("api/[controller]")]
[ApiController]
public class YazarApiController : ControllerBase
{
    KitaplikContext k = new KitaplikContext();

    // GET: api/<OgrenciController>
    [HttpGet]
    public List<Yazar> Get()
    {
        return k.Yazarlar.ToList();
    }

    // GET api/<OgrenciController>/5
    [HttpGet("{id}")]
    public Yazar Get(int id)
    {
        var y = k.Yazarlar.FirstOrDefault(x => x.YazarID == id);
        return y;
    }

    // POST api/<OgrenciController>
    [HttpPost]
    public void Post([FromBody] Yazar y)
    {
        k.Yazarlar.Add(y);
        k.SaveChanges();
    }

    // PUT api/<OgrenciController>/5
    [HttpPut("{id}")]
    public IActionResult Put(int id, [FromBody] Yazar y)
    {
        var y1 = k.Yazarlar.FirstOrDefault(x => x.YazarID == id);
        if (y1 is null)
        {
            return NotFound();
        }
        else
        {
            y1.YazarAd = y.YazarAd;
            y1.YazarSoyad = y.YazarSoyad;
            k.Update(y1);
            k.SaveChanges();
            return Ok();
        }
    }

    // DELETE api/<OgrenciController>/5
    [HttpDelete("{id}")]
    public ActionResult Delete(int id)
    {
        var y1 = k.Yazarlar.FirstOrDefault(s => s.YazarID == id);
        if (y1 is null)
        {
            return NotFound();
        }
        else
        {
            k.Remove(y1);
            k.SaveChanges();
            return Ok();
        }
    }
}
```

Http Get Metodu ile istekte bulunan Api'ye ait yanıt - Yazarlar tablosundaki tüm kayıtlar getiriliyor

```

1 [
2   {
3     "yazarID": 1,
4     "yazarAd": "Jack",
5     "yazarSoyad": "London",
6     "kitaplar": null
7   },
8   {
9     "yazarID": 2,
10    "yazarAd": "Pyodor",
11    "yazarSoyad": "Dostoevsky",
12    "kitaplar": null
13  }
14 ]
  
```

ÖZET

Bu bölümde API, Rest API kavramlarından bahsederek bunlara neden ihtiyaç duyduğunu ayrıntılı olarak tartıştık. Bir web API'sinin Rest API olabilmesi için gerekli kısıtları öğrendik.

API'lerde yanıt mesajlarının yapılandırmasında sıkılıkla kullanılan veri formatları olan Json ve Xml kavramları üzerinde durduk. Rest API kullanıldığında istek ve yanıtlar http protokolü üzerinden gerçekleştiğinden, http istek ve yanıt döngüsüne ait birtakım öğeleri inceledik.

HTTP yanıtının önemli bileşenlerinden olan HTTP durum kodlarını ele aldık. Rest API'ler ile CRUD işlemlerinin hangi http metotları ile yapılabileceğini inceledik.

Son olarak da Asp.Net Core MVC'de code first yaklaşımı ile oluşturduğumuz veri tabanı uygulaması üzerinde Rest API ile CRUD işlemlerini gerçekleştirdiğimiz bir web API uygulaması tasarladık.



İSTANBUL ÜNİVERSİTESİ

AUZEF