ASP.NET CORE 8.0 MVC DERS NOTLARI-5. HAFTA

İçindekiler

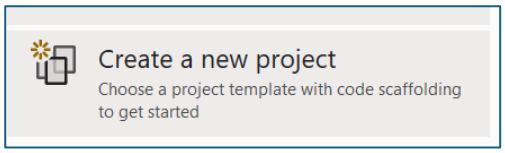
GİRİŞ

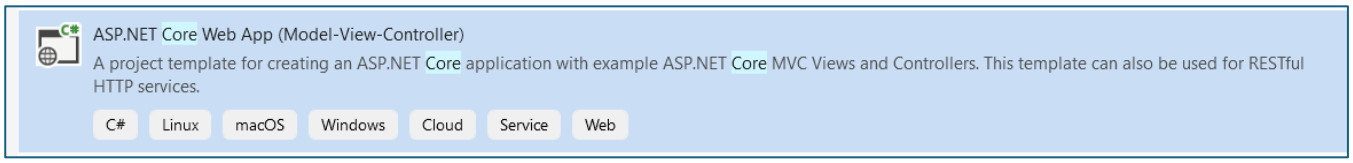
A. Asp.Core un özellikleri

- Microsoft tarafından geliştirilen **Platformdan bağımsız** (Windows, Unix, Mac, IO) bir platformdur.
- **Open source** (açık kaynaklı) dur. GitHup yükleyip başkalarınında geliştirmesine olanak tanınabilir.
- **Kullanıcı taraflı** (client-side) yapılar ile (framework) ile tam uyum içindedir. (AngularJS, KnockhoutJS, ReactJS)
- **Bulut ortamlar** içinde kolay konfigure edilebilir.
- **Host etmek** kolaydır. Ayrıca IIS hostingi destekler.
- **Katmanlı mimari** ile geliştirmek kolaydır. Sınıf yapıları ile çalışır (Object oriented)
- **Visual Studio** IDE (normal VS) yada Code ile geliştirmek kolaydır. Biz Windows ortamında çalıştığımızdan IDE olan versiyonu kullanacağız.

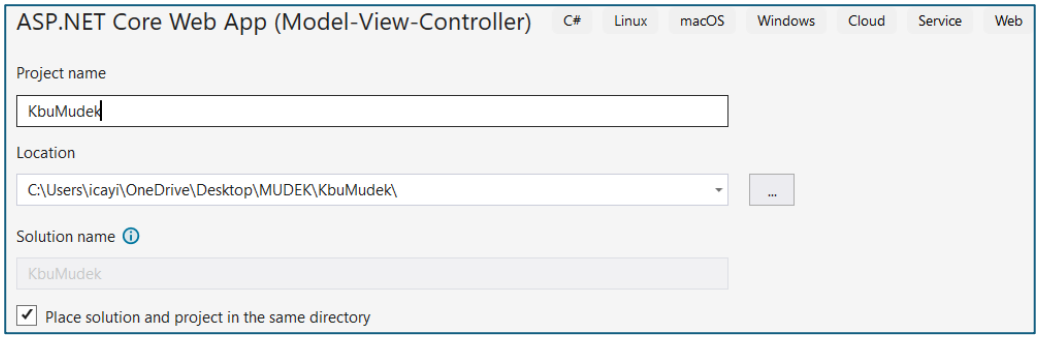
B. Proje Oluşturma

Visual Studio açalım. Create New Project kısmından girelim ve Asp.net Core Web App C# olanı seçelim.

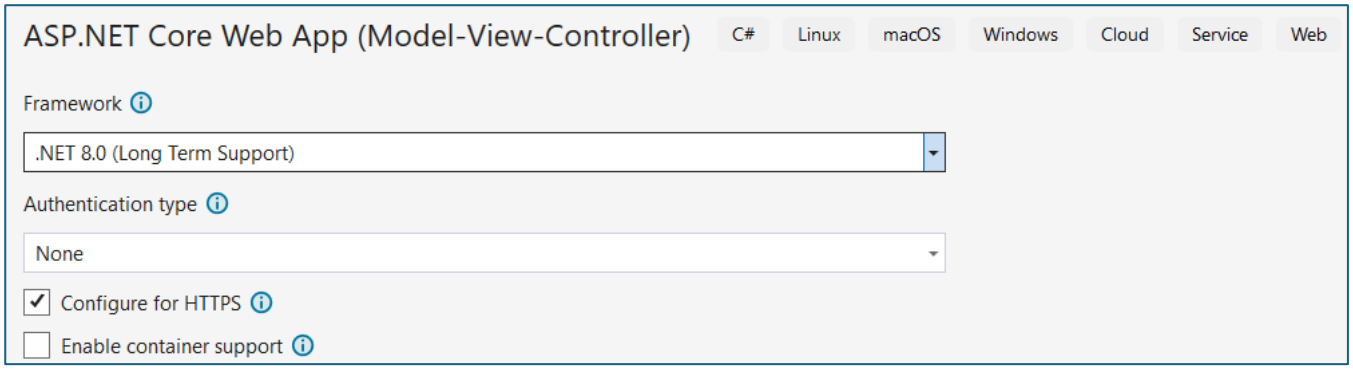




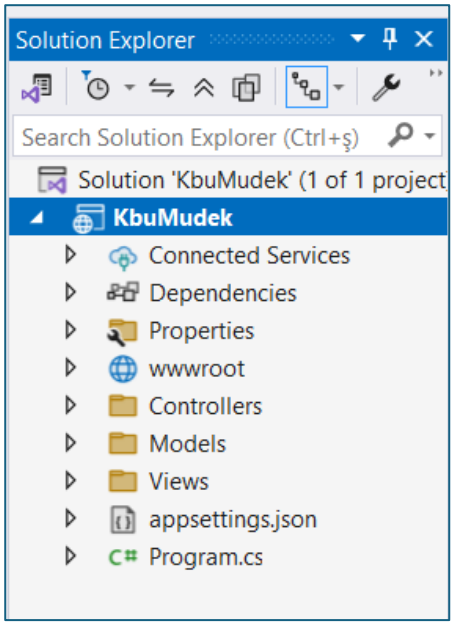
Proje adını ve yolunu belirtelim.



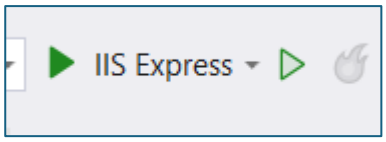
Core NET 6.0 versiyonunu seçelim.



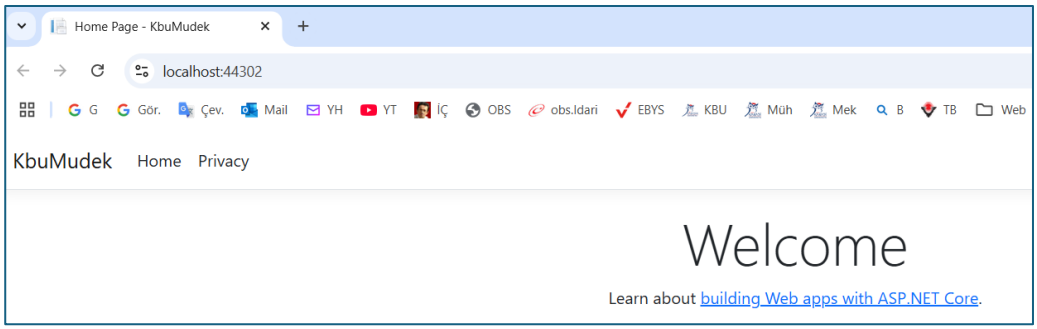
Proje klasörlerimiz aşağıdaki şekilde oluşmuş oldu.



Sitemizi test edelim. IIS Express seçip çalıştıralım.

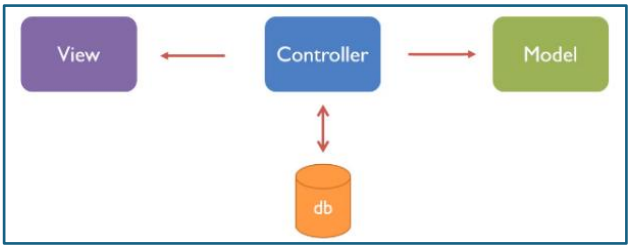


Sitemizin çalıştığını görüyoruz.



C. MVC Yapısı (Model,View,Controller)

Proje içerisinde MVC kalasör yapısının oluştuğunu gördük. Açılan sayfalarımızın görünen kısmı View klasörü içerisindedir. Bilgileri taşıyan sınıf yapılarının şemaları Model klasörü içerisindedir. Veritabanı ile birlikte bu üçü arasında kontrolü sağlayan C# kodlarının bulunduğu kısımlar Controller klasörü içerisindedir.

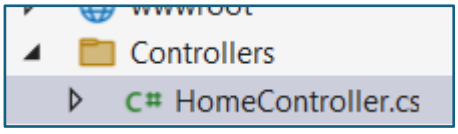


İnternetten ilk istek alındığında (Request) karşılayan ve nereye gideceğine karar veren yer Program.cs içindeki aşağıdaki kodlardır.

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

Bu adres bilgisi bize ilk olarak Home Controller sınıfına gidilecek onun içindeki Index alt metodunun çalıştırılacağını söylüyor. Oraya giderken yanımızda Id bilgisini de götürebiliriz ama bu tercihlidir. Yanındaki soru işareti zorunlu olmadığını göstermektedir.

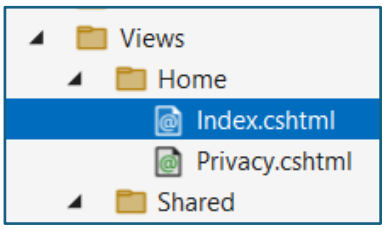
Şimdi bu adresin bize tarif ettiği yerlere sırayla gidelim.




```
public class HomeController : Controller
{
```

```
    private readonly ILogger<HomeController> _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

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

Burada HomeController içindeki Index action metodu View() sayfasına gidip oradan dönecek sonucu bize göstereceğini ifade ediyor. View() olduğu yere gidelim.

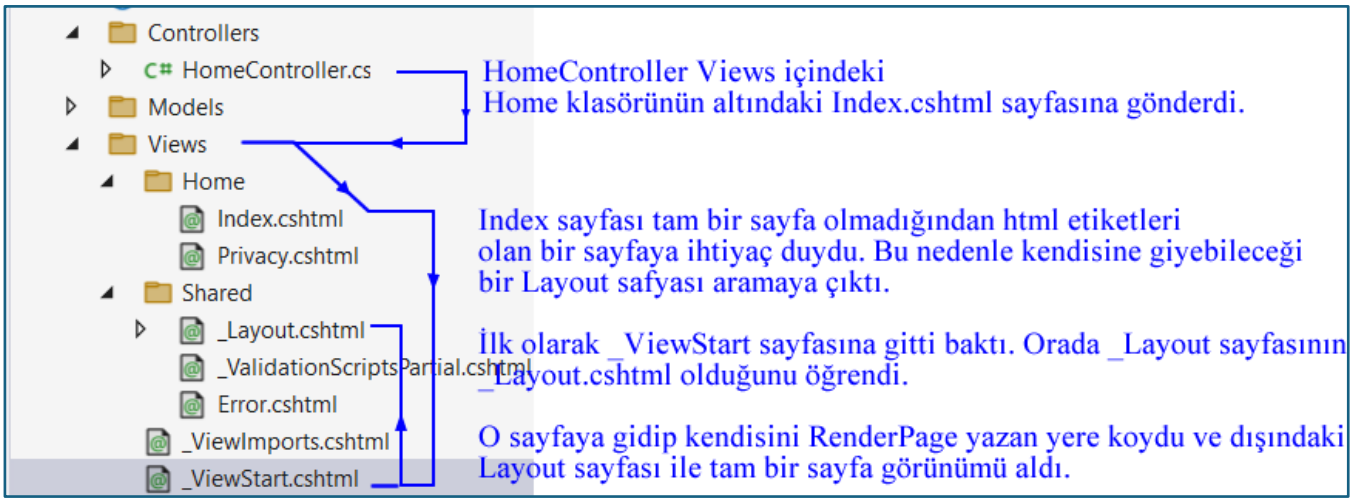


İçindeki kodları açalım. (Index.cshtml)

```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps
with ASP.NET Core</a>.</p>
</div>
```

Dikkat edersek bu sayfanın içinde ekranda gördüğümüz herşey yok. Örneğin başlıktaki “KbuMudek” yazısı yoktur.

Bu sayfa aslında dış kısmına bir _Layout şablon sayfasını giymektedir. Bu nedenle bu sayfadan sonra _Layout sayfasına gitmeye çalışacak fakat çok sayıda _Layout tipinde sayfa olabilir. Hangi _Layout kullanacağını da _ViewStart.cshtml dosyasına bakarak anlamaktadır.



Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
```

```
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps
with ASP.NET Core</a>.</p>
</div>
```

_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - KbuMudek</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
    <link rel="stylesheet" href="~/KbuMudek.styles.css" asp-append-version="true" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white
border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">KbuMudek</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
controller="Home" asp-action="Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-
controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">
            @RenderBody()
        </main>
    </div>

    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2025 - KbuMudek - <a asp-area="" asp-controller="Home" asp-
action="Privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
```

```
    <script src="~/js/site.js" asp-append-version="true"></script>
    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Not: Index.cshtml içinde aşağıdaki gibi bir ifade olsaydı bu sefer _Layout arayışına çıkmayacaktı. O zaman sayfanın çalışabilmesi için <html> etiketlerinin olduğu tam bir sayfa şeklinde olması gerekirdi.

```
@{
    Layout = null;
}
```

D. View sayfaları içinde C# kodlarının kullanımı

Bu arada Index.cshtml sayfasından _Layout.cshtml sayfasına giderken yanımızda bir tane değişken içerisinde bilgi taşıyoruz. ViewData ifadesiyle içerisinde "Home Page" yazısını taşıyoruz. Burada @ sembolü ile başlayan kısımlar C# kodlarını yazdığımız kısımları yani içerisinde program scriptlerini içeren kısımlardır.

```
@{
    ViewData["Title"] = "Home Page";
}
```

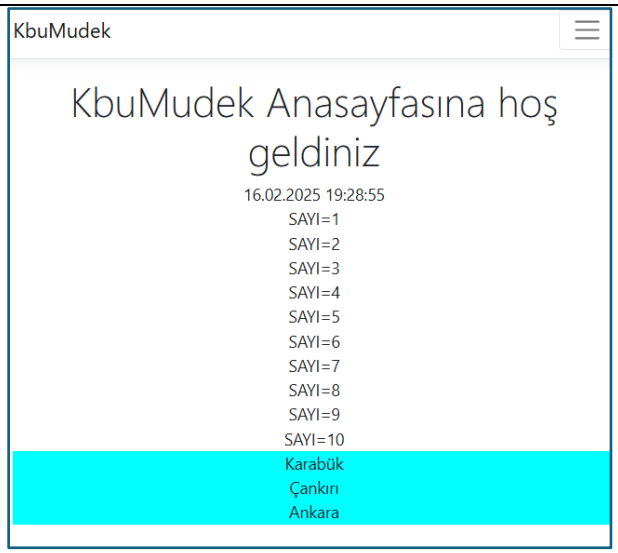
Aşağıdaki satırla da ViewData nın yanın "KbuMudek" yazısı getirilmiş oluyor.

```
    <title>@ViewData["Title"] - KbuMudek</title>
```

Örnek 1: Başka bir örnek yapalım. Sayfamız içerisinde Günün saatini gösterelim. Aynı zamanda 1-10 kadar sayıları yazdıralım. Bir de foeach döngüsünü kullanmak için bir dizi oluşturup içeriğini gösterelim.

Index.cshtml

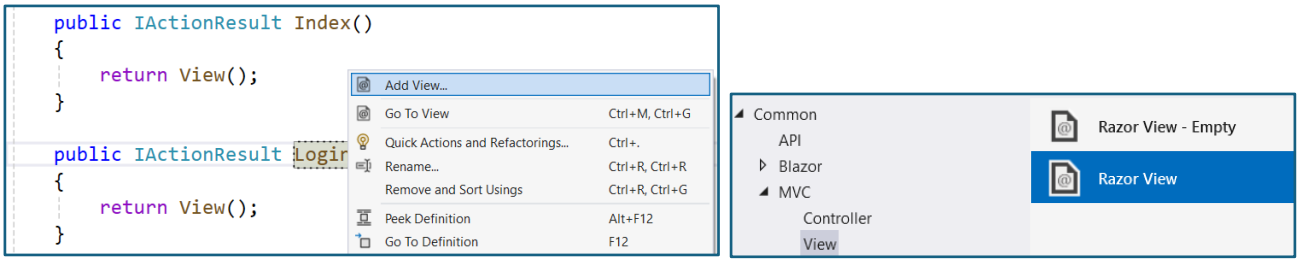
```
@{
    ViewData["Title"] = "Ana Sayfa";
}

<div class="text-center">
    <h1 class="display-4">KbuMudek
Anasayfasına hoş geldiniz</h1>
    <div>
        @DateTime.Now.ToString()

    </div>

    <div>
        @for(int i=1; i<=10;i++)
        {
            <span>SAYI=</span>@i
            <br />
        }
    </div>
    <div>
        @{
            var Dizi = new List<string>() {
"Karabük", "Çankırı", "Ankara" };
        }
        @foreach (string Sehir in Dizi)
        {
            <div style="background-
color:aqua"> @Sehir</div>
```



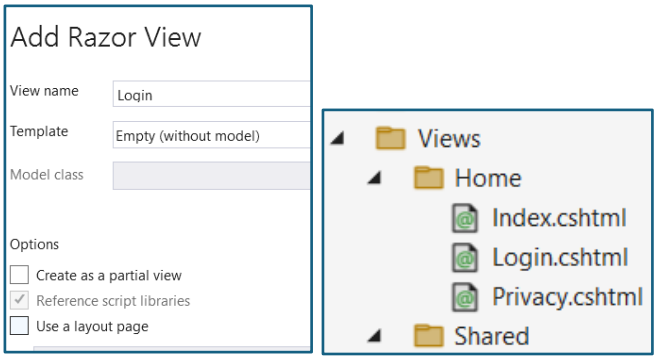
```
        }
    </div>
</div>
```

Örnek 2: HomeController tarafından kontrol edilen bir tane Login sayfası oluşturalım. Bu sayfa herhangi bir _Layout şablonu kullanmasın.

Bunun için HomeController içine Login action metodunu ekleyelim. Başlığın üzerine sağ tuşa tıklayıp View sayfasını oluşturmasını sağlayalım. Çıkan ekranda Razor View sayfası şeklinde View sayfasını oluşturalım. Razor sayfalar <html> kodları içinde @ işaretleriyle C# kodlarını yazabildiğimiz sayfa türünü ifade ediyor.



Kendi başına çalışacak bir sayfa olacağı için "Use a layout page" kısmını iptal ettik. Böylece login sayfası kendi başına görüntüyü oluşturabilen _layout dan bağımsız bir sayfa olmuş oldu. İçerisinde sayfa yapısını oluşturan <html> etiketleri de bulunmaktadır.



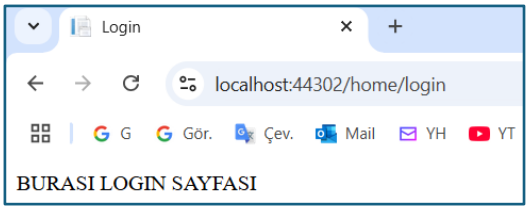
Login.cshtml

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Login</title>
</head>
<body>
   BURASI LOGIN SAYFASI
</body>
</html>
```

Şimdi bu sayfayı nasıl çalıştıracağız. Direk çalıştırsak Home>Index sayfası gelecektir. Dikkat edersek Program.cs içindeki başlangıç adres satırımızda önce Controller ifade edilmiş ardından Action ifade edilmiş.

```
pattern: "{controller=Home}/{action=Index}/{id?}");
```

Buna göre bizde adres satırına **/home/login** yazmamız yeterli olacaktır. Deneyelim.



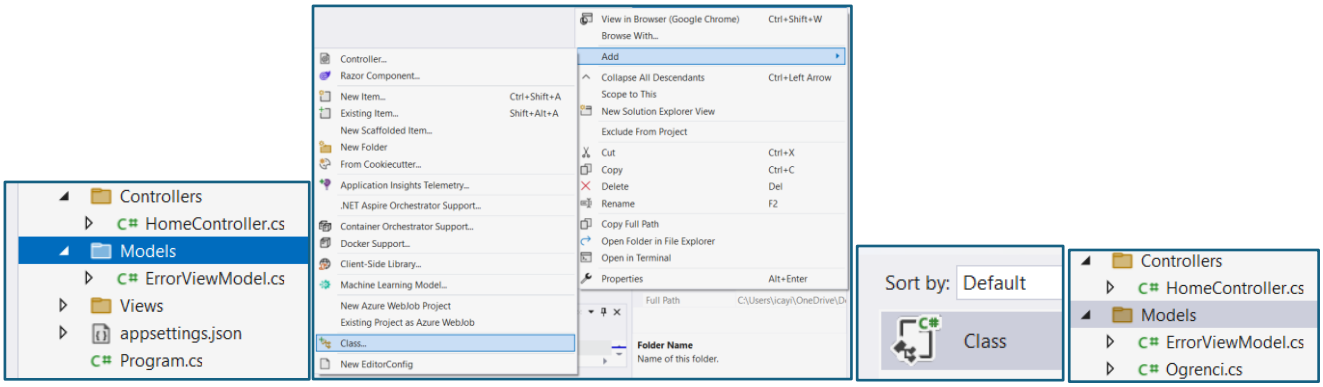
MODEL KULLANIMI ve BİLGİLERİN VIEW'e TAŞINMASI

1. Model İçinde Bilgilerin Taşınması

Modeli projemizde kullanacağımız nesneleri temsil eden ya da taşınacak bilgileri temsil eden bir yapı olarak düşünebiliriz. Örneğin Öğrenci bilgilerini tutarken tanımlayacağımız bir öğrenci bir model dir. Onun alt özelliklerine de Property (özellik) diyeceğiz. Model aslında bir sınıftır (class). İçindeki özelliklerde Properties dir. Verileri taşımak için kullanacağımız bir araba gibi düşünebiliriz. Aynı zamanda Veritabanında tablolara karşılık gelen yapılardır. Properties lerde tabloların sütunları olacaktır.

Örnek 3: Öğrenci bilgilerini tutmak için Model yapısı oluşturun. Bu modeli kullanarak Controller içinden bilgileri View sayfasına göndererek tablo şeklinde gösterin.

Models üzerine sağ tuşa tıklayıp yeni bir class ekleyelim.



Sınıfın içinde "Prop" yazıp 2 defa "tap" tuşuna tıklayınca Property lerin formatını getirecektir.

```
public int MyProperty { get; set; }
```

Bu şekilde property formatlarını oluşturup, öğrencinin Ad, Soyad, OgrNo, DTarih şeklinde 4 tane property tanımlayalım. Modelin son hali şu şekilde olacaktır.

Ogrenci.cs

```
namespace KbuMudek.Models
{
    public class Ogrenci
    {
        public string Ad { get; set; }
        public string Soyad { get; set; }
        public int OgrNo { get; set; }
        public DateTime DTarih { get; set; }
    }
}
```

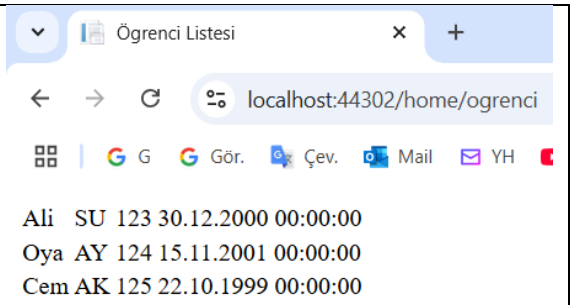
Şimdi HomeController içinde Ogrenci adında bir Action metodu oluşturalım. Henüz Veritabanından bilgi getirmediğimizden bu metod içinde öğrenci bilgilerini taşıyacak bir dizi oluşturalım ve bunu View sayfasına gönderelim, orada görüntüleyelim.

HomeController.cs *(bir kısmı)*

```
public class HomeController : Controller
{

  public IActionResult Ogrenci()
  {
    var Ogrenciler = new List<Ogrenci>()
    {
       new Ogrenci(){Ad="Ali", Soyad="SU", OgrNo=123, DTarih=new DateTime(2000,12,30) },
       new Ogrenci(){Ad="Oya", Soyad="AY", OgrNo=124, DTarih=new DateTime(2001,11,15) },
       new Ogrenci(){Ad="Cem", Soyad="AK", OgrNo=125, DTarih=new DateTime(1999,10,22) },
    };

    return View(Ogrenciler);
  }
```

Ogrenci.cshtml

```
@model List<KbuMudek.Models.Ogrenci>
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-
width" />
    <title>Ögrenci Listesi</title>
</head>
<body>
    <table>
        @foreach (var Ogr in Model)
        {
            <tr>
                <td>@Ogr.Ad</td>
                <td>@Ogr.Soyad</td>
                <td>@Ogr.OgrNo</td>
                <td>@Ogr.DTarih</td>
            </tr>
        }
    </table>
</body>
</html>
```



Burada sayfaya model içinde gelen öğrenci bilgileri List<> şeklinde bir dizinin içine atılıyor. Daha sonra foreach döngüsü ile içindeki her bir öğrenci okunup tablonun gözlerinde alt properties leri ile görüntüleniyor.

2. Model Harici Bilgi Taşıma Araçları

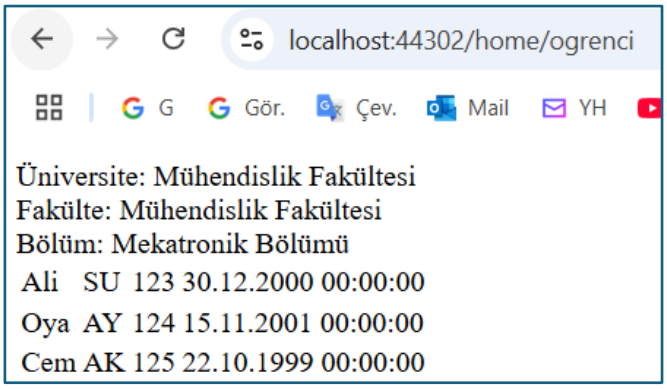
Model içerisinde belli bir formatta bilgileri taşıyabiliyoruz. Fakat bazen kısa süreli harici bilgileri taşımamız da gerekebilir. Örneğin Öğrencilerin bilgilerini listelerken hangi bölüme ait öğrenciler olduğu bilgisine de ihtiyaç duyabiliriz. Bu durumda Bölüm bilgisini geçici olarak Controller'dan View taşımamız gerekebilir.

Bilgileri kısa süreli taşımak için 3 yöntemden bahsedilebilir. Bunlar tablo halinde aşağıdaki şekildedir.

Açıklama	ViewBag	ViewData	TempData
Controller'da Kullanımı	`public IActionResult Index() { ViewBag.Message = "Merhaba, ViewBag!"; return View(); }`	`public IActionResult Index() { ViewData["Message"] = "Merhaba, ViewData!"; return View(); }`	`public IActionResult Index() { TempData["Message"] = "Bu mesaj diğer sayfada da görünecek!"; return RedirectToAction("Hakkimizda"); }`
View'de Kullanımı	`<p>@ViewBag.Message</p>`	`<p>@ViewData["Message"]</p>`	`<p>@TempData["Message"]</p>`
Avantaj/Dezavantajları	Arka planda ViewData kullanılarak Veri saklar. Tür güvenliği yoktur. Bu yüzden hata ayıklamak zor olabilir. ViewData'ya göre biraz daha yavaştır ama ondan daha okunaklıdır.	ViewData Dictionary (<string,object>) tabanlı olarak çalışır. Tür güvenliği yoktur. Object olarak sakladığı için kullanırken dönüşüm yapmak gerekebilir. ViewBag'a göre daha hızlıdır. Verilere string anahtar ile erişilir.	TempData, ASP.NET Core ve MVC'de bir Controller'dan diğerine veya bir Action'dan diğerine veri taşımak için kullanılan bir mekanizmadır. HTTP request tamamlandıktan sonra bile veriyi saklayabilmesi, onu ViewBag ve ViewData'dan ayıran en büyük farktır. Session tabanlıdır: Veriler, Session veya Cookie mekanizmasıyla saklanır.
Veri Türü	Dynamic	Dictionary<string, object>	ViewData gibi Dictionary<string, object> mantığında saklanır.
Tür güvenliği	Yok	Yok	Yok
Performans	Biraz daha yavaş	Daha hızlı	Daha hızlı
Kullanım kolaylığı	Daha okunaklı	Daha az okunaklı	Daha az okunaklı
Veri Miktarı	Küçük veriler için uygun	Küçük veriler için uygun	
Süre	Sadece aynı Request içinde geçerlidir. Tek kullanımlıktır. Yönlendirme sonrası silinir.	Sadece aynı Request içinde geçerlidir. Tek kullanımlıktır. Yönlendirme sonrası silinir.	Request'ten sonra da veri saklar: Veriler bir sonraki request'e kadar korunur. Sayfa yenilendiğinde veya başka bir Action'a yönlendirme yapıldığında bile veri kullanılabilir. Eğer veri bir kez okunduktan sonra saklanmaya devam etsin istiyorsanız: `var mesaj = TempData["Message"]; TempData.Keep("Message"); // Veri, bir sonraki request için de saklanır.` Yada veri okunduktan sonra silinmesini istemiyorsanız.

			`var mesaj = TempData.Peek("Message"); // Veri okunur ama TempData'dan silinmez.`

Örnek 4: Yukarıdaki aynı örneği kullanarak Controller tarafında öğrenciler hangi bölüme, Fakülteye ve Üniversiteye ait ise bu bilgileri 3 farklı yöntemle Controller'dan View'e taşıyalım.



HomeController

```
public IActionResult Ogrenci()
{
  ViewBag.Message1 = "Karabük Üniversitesi";
  ViewData["Message2"] = "Mühendislik Fakültesi";
  TempData["Message3"] = "Mekatronik Bölümü";

  var Ogrenciler = new List<Ogrenci>()
  {
    new Ogrenci(){Ad="Ali", Soyad="SU", OgrNo=123, DTarih=new DateTime(2000,12,30) },
    new Ogrenci(){Ad="Oya", Soyad="AY", OgrNo=124, DTarih=new DateTime(2001,11,15) },
    new Ogrenci(){Ad="Cem", Soyad="AK", OgrNo=125, DTarih=new DateTime(1999,10,22) },
  };
  return View(Ogrenciler);
}
```

Ogrenci.cshtml

```
@model List<KbuMudek.Models.Ogrenci>
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Ögrenci Listesi</title>
</head>
<body>
    <div>
        Üniversite: @ViewBag.Message1
    </div>
    <div>
        Fakülte: @ViewData["Message2"]
    </div>
    <div>
        Bölüm: @TempData["Message3"]
    </div>
    <table>
        @foreach (var Ogr in Model)
        {
            <tr>
                <td>@Ogr.Ad</td>
                <td>@Ogr.Soyad</td>
                <td>@Ogr.OgrNo</td>
                <td>@Ogr.DTarih</td>
            </tr>
        }
    </table>
```

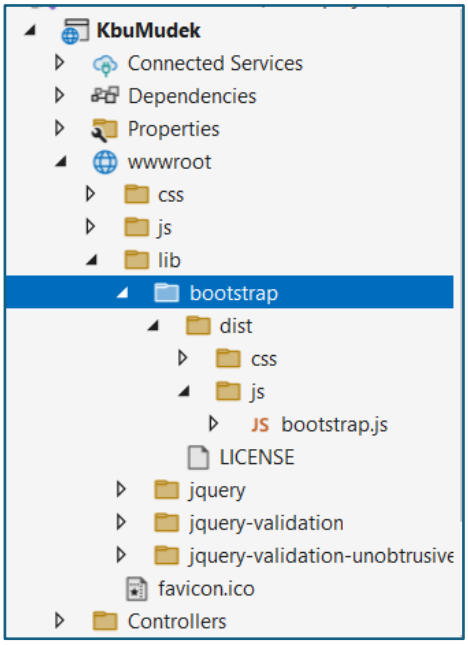
	</body> </html>

wwwroot KLASÖRÜNÜN KULLANIMI

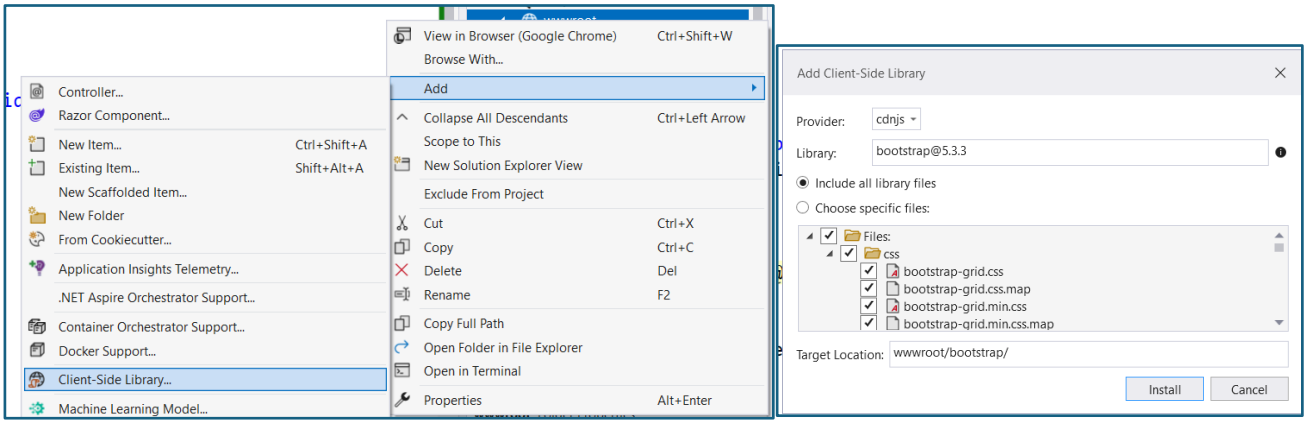
Sayfalarımızı estetik hale getirmek için Css yada Boostrap kütüphanelerini kullanırız. Bu kütüphanelere ait sabit (static) dosyalarımızı wwwroot altında tutarız. Şimdi buraya Boostrap kütüphanesini ekleyip yukarıdaki örnekte geçen tablomuzu daha estetik hale getirelim.

Örnek 5: Yukarıdaki örnekte geçen sayfadaki tabloyu Boostrap kütüphanesi kullanarak daha estetik hale getirin.

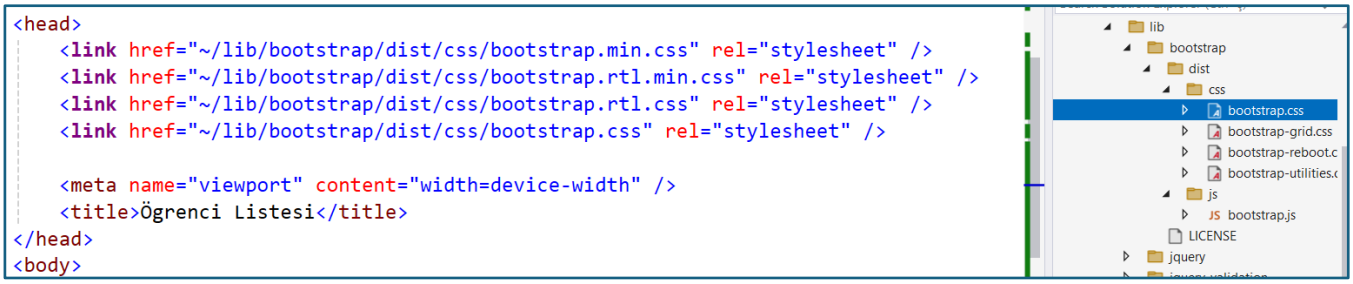
Öncelikle Bootstrap kütüphanesini wwwroot'un altına getirmemiz gerekir. Fakat biz baştan MVC formatını yükleyerek projeyi oluşturduğumuzdan en azından bu konuda Boostrap kütüphanesini bu klasörün altına yüklenmiş olarak geldiğini görüyoruz.



Eğer yüklenmemiş olsayda VisualStudio üzerinden bu kütüphaneyi şu şekilde ekleyebiliriz. Wwwroot üzerine sağ tuşa tıklayıp Add>Client-Side Library (İstemci taraflı kütüphane) seçeneğini seçip oradan Boostrap kütüphanesini yükleyebiliriz. Biz burada daha önceden yüklenen kütüphaneyi kullanalım.



Şimdi ilgili View sayfamızdan bu kütüphaneye bağlantıyı sağlayacak linki oluşturalım. Sayfamızı açalım. Yandan Bootstrap.css dosyasını tutup sayfamızın head kısmı içine sürükleyince ilgili linkleri oluşturmuş olacaktır. Yalnız sadece <link href ile başlayanları tutun. Diğerlerini kaldırın.



Fakat programın bu dosyaları kullanabilmesi için Program.cs dosyası içinde aşağıdaki satırın bulunması gerekmektedir. Kontrolünü yapalım. Bu satır bizde hazır olarak geldi.

```
app.UseStaticFiles(); //wwwroot altındaki dosyaların çalışması bulunmalıdır.
```

Artık sayfamızdaki ilgili etiketlerin içine bootsrap class larını ekleyebiliriz. Başlıklarımızda <h2> şeklinde etiketlerimizi kullanalım. Sayfanın son hali şu şekilde olsun.

Ogrenci.cshtml

```
@model List<KbuMudek.Models.Ogrenci>
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <link
href="~/lib/bootstrap/dist/css/bootstrap.min.css"
rel="stylesheet" />
    <link
href="~/lib/bootstrap/dist/css/bootstrap.rtl.min.css"
rel="stylesheet" />
    <link
href="~/lib/bootstrap/dist/css/bootstrap.rtl.css"
rel="stylesheet" />
    <link
href="~/lib/bootstrap/dist/css/bootstrap.css"
rel="stylesheet" />
    <meta name="viewport" content="width=device-
width" />
    <title>Ögrenci Listesi</title>
</head>
<body>
    <h2>
        Üniversite: @ViewBag.Message1
    </h2>
    <h3>
        Fakülte: @ViewData["Message2"]
    </h3>
    <h4>
        Bölüm:
@TempData["Message3"]
    </h4>
    <table class="table table-
bordered">
        @foreach (var Ogr in
Model)
        {
            <tr>
                <td>@Ogr.Ad</td>

<td>@Ogr.Soyad</td>

<td>@Ogr.OgrNo</td>

<td>@Ogr.DTarih</td>
            </tr>
        }
    </table>
</body>
</html>
```

Not: ViewBag içindeki Üniversite bilgisi taşınırken ViewDatayı gösteriyor. Bunun sebebi ViewBag ve ViewData nın ikinsinde aynı "Message" teriminin kullanılmasıdır. Sonlarına 1,2,3 sayısı eklenerek düzeltildi.

SINIF İÇİ UYGULAMALAR

Uygulama 1: Değişken içerisinde bir bilginin Controller'dan View sayfasına taşınması

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController>
_logger;

    public
HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    public IActionResult Index()
    {
        string Ad = "Ali";

        return View(model: Ad);
    }

}
```

```
@model string

@{
    Layout = null;
}

<html>
<head>
</head>

<body>
  <div>
      @Model
  </div>
</body>
</html>
```

Uygulama 2: Bir list dizi içinde bilgilerin Controller'dan View sayfasına taşınması

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
    }

    public IActionResult Index()
    {
        List<string> isimler = new List<string>() { "Ali", "Oya", "Can" };

        TempData["Ogrenciler"] = isimler;

        return View();
    }
```

```
    }
```

View sayfası: Views>Index.cshtml

```
@{
    Layout = null;
}

@{
    List<string> Adlar = TempData["Ogrenciler"] as List<string>;
}

<html>
<head>

</head>

<body>

  <div>
       @foreach(string isim in Adlar)
        {
            @isim
            <br/>

        }

  </div>
</body>
</html>
```

Uygulama 3: Bir model Sınıfı içinde bilgilerin Controller'dan View sayfasına taşınması

Models>Ogrenci.cs

```
namespace Proje_A.Models
{
    public class Ogrenci
    {
        public int OkulNo { get; set; }     // Öğrencinin numarası
        public string Ad { get; set; }      // Öğrencinin adı
        public string Soyad { get; set; }   // Öğrencinin soyadı

    }
}
```

Controllers>HomeController>Index()

```
    public class HomeController : Controller
    {

        public IActionResult Index()
        {

            List<Ogrenci> ogrenciler = new List<Ogrenci>

            {
                new Ogrenci { OkulNo = 1001, Ad = "Ali", Soyad = "Demir" },
                new Ogrenci { OkulNo = 1002, Ad = "Ayşe", Soyad = "Kaya" },
                new Ogrenci { OkulNo = 1003, Ad = "Mehmet", Soyad = "Yılmaz" }
            };
```

```
            return View(ogrenciler);
        }
    }
```

Views>Home>Index.cshtml

```
@model List<Proje_A.Models.Ogrenci>

@{

    Layout = null;
}

<html>
    <head>

    </head>

    <body>

    <h2>Öğrenci Listesi</h2>

    <table class="table table-bordered">
        <thead>
            <tr>
                <th>Okul No</th>
                <th>Ad</th>
                <th>Soyad</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var ogr in Model)
            {
                <tr>
                    <td>@ogr.OkulNo</td>
                    <td>@ogr.Ad</td>
                    <td>@ogr.Soyad</td>
                </tr>
            }
        </tbody>
    </table>

</body>
</html>
```