[1.Bölüm](#Bölüm1)

* Projede yapılacak görselleri göreceksiniz…

[2.Bölüm](#Bölüm2)

* Proje oluştuma görselleri…

[3.Bölüm](#Bölüm3)

* Entities Katmanın Veri tabandaki tablolalar karşılık gelir.

[4.Bölüm](#Bölüm4)

* DataAccess Katmanı oluşturma
* Generic Repository Oluşturma
* DbContext Sınıfını Oluşturma

[5.Bölüm](#Bölüm5)

* Bussiness Katmanı oluşturma

[6.Bölüm](#Bölüm6)

* WebUI Katmanını oluşturma
* Gerekli Migrations,DI,Startup dosyasında ayarlar var

[7.Bölüm](#Bölüm7)

* View Nesnesine model göndermek

[10.Bölüm](#Bölüm10)

* Custom Middleware
* Kendi middleware oluşturarak Extension method oluşturulur.(Bootstrap)

[11.Bölüm](#Bölüm11)

* Client Side Framework Entegrasyonu
* Yüklenen paketleri (bootstrap,..) projemizde kullanmak için.
* Node\_modules klasöründe olan paketleri layout sınıfına referansları yazmamız gerek.

[12.Bölüm](#Bölüm12)

* View Component kullanımı,neden gereklidir?

[13.Bölüm](#Bölüm13)

* Model Binding ile sayfalama Category ile Product ile

[14.Bölüm](#Bölüm14)

* Category menude seçilen itemın belirgin hale getirmek için

[15.Bölüm](#Bölüm15)

* Genel bir sayfalama yazılacak, diğer controller metodlarıda kullanabilsin diye. Asp.net Core kendine özel Tag helperları var biz burda yeni bir tag helper oluşturacağız.

[16.Bölüm](#Bölüm16)

* ViewImport Kullanımı

[17.Bölüm](#Bölüm17)

* Session kullanımı

[18.Bölüm](#Bölüm18)

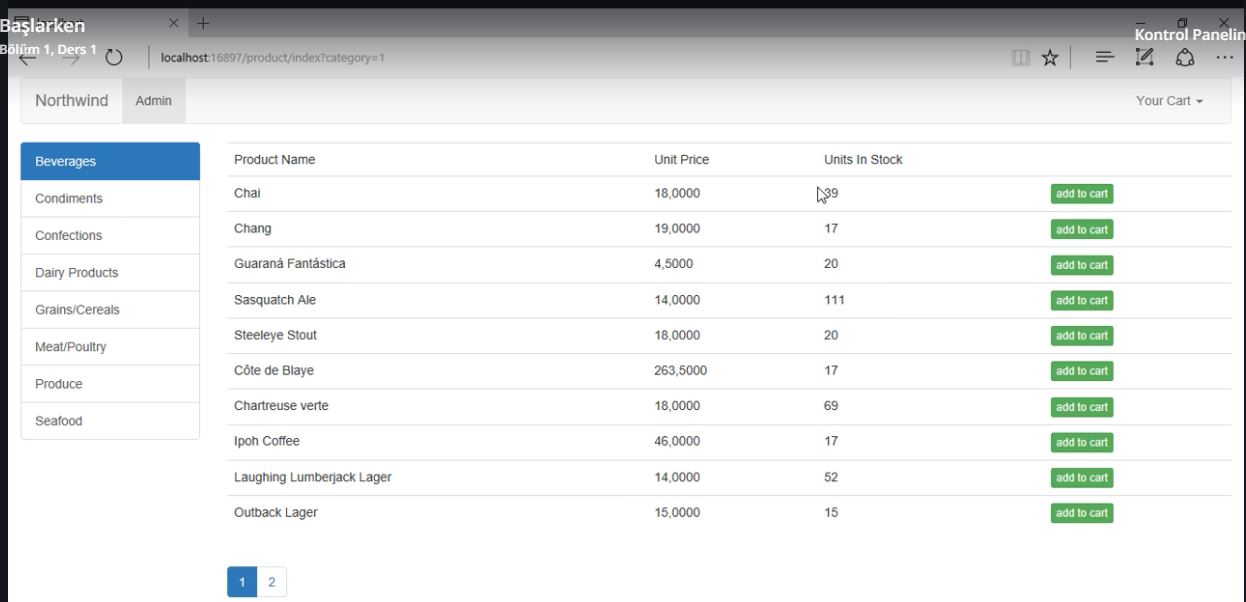
* Cart Summary
* Sepet ile ilgili işlemler
* Validation ile ilgili işlemler
* Admin Panelin Hazırlanması

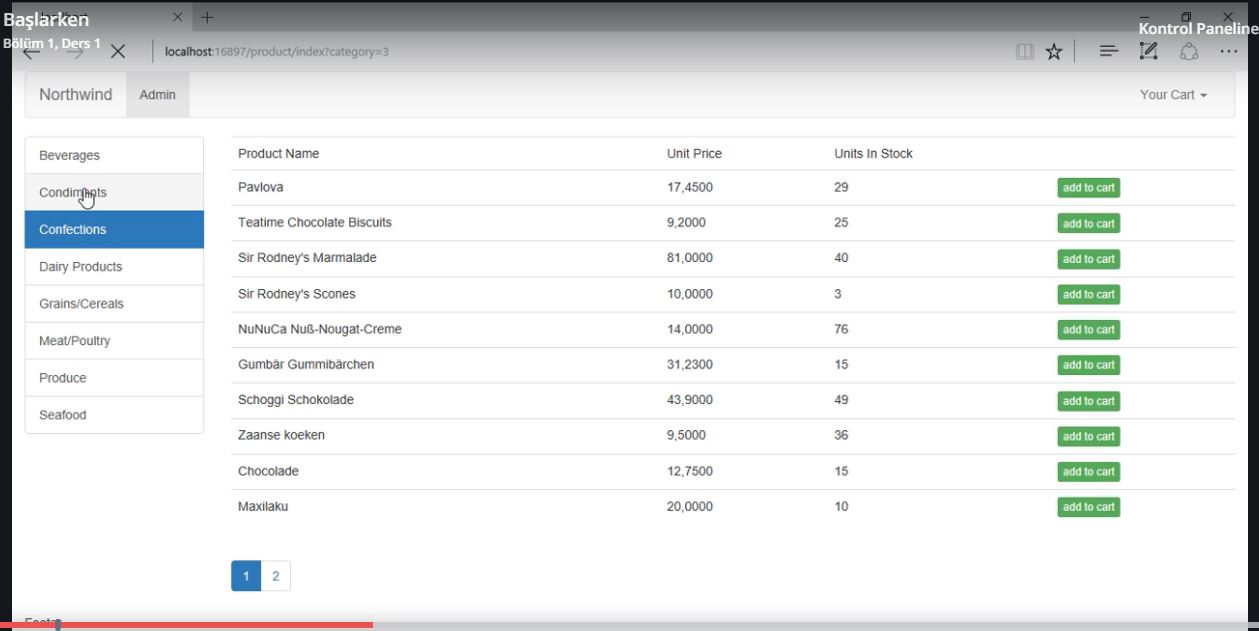
[19.Bölüm](#Bölüm19)

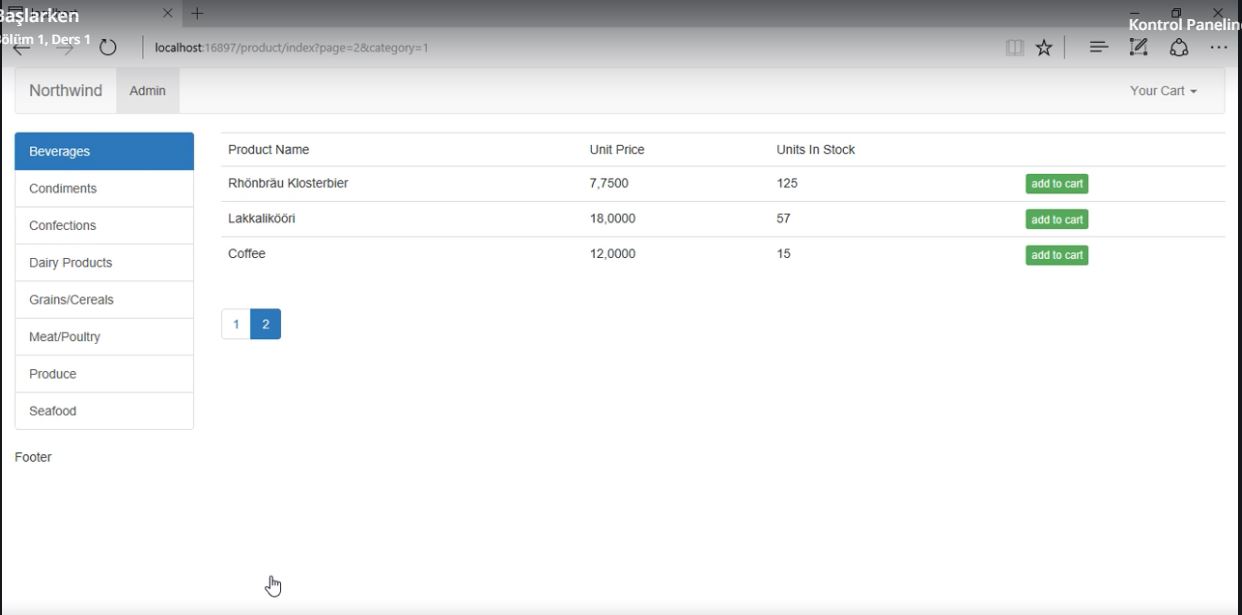
* Asp.Net Core Identity
* Account Controller
* Login/Register
* Migration

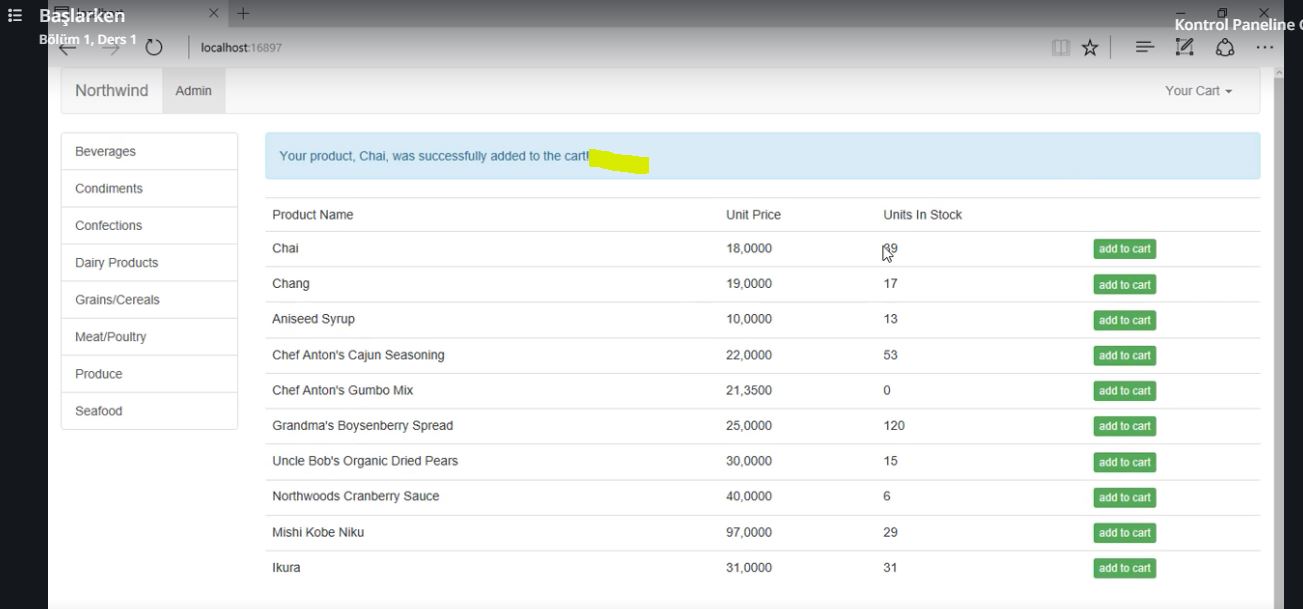
Bölüm 1

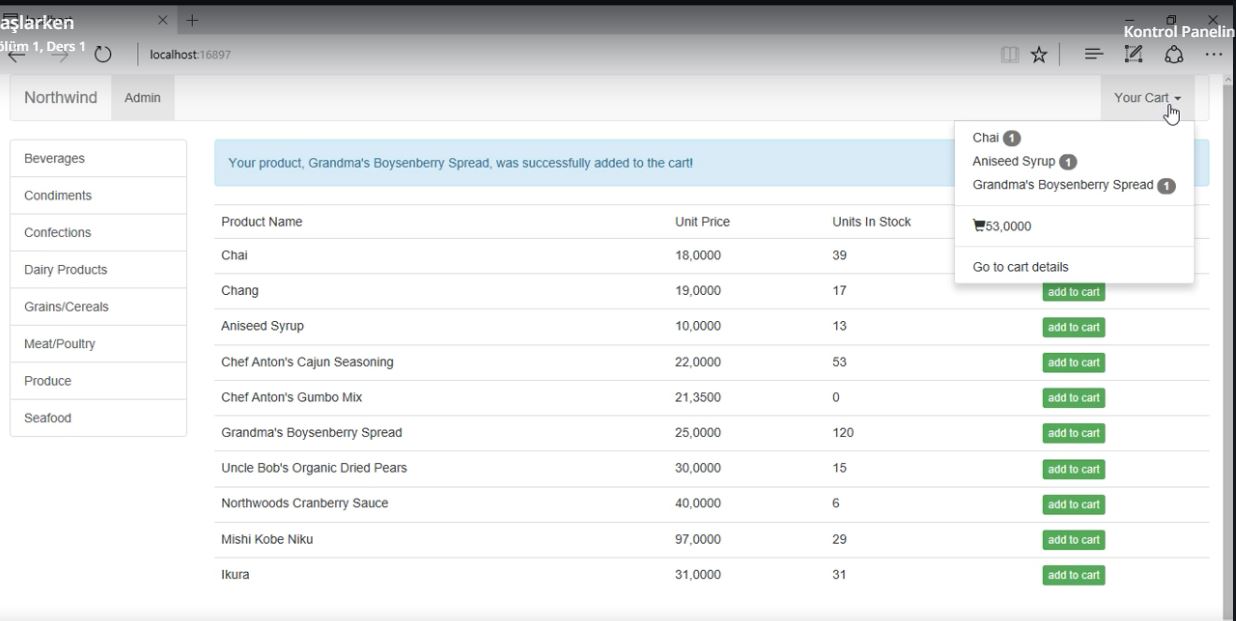
Projede yapılacaklar

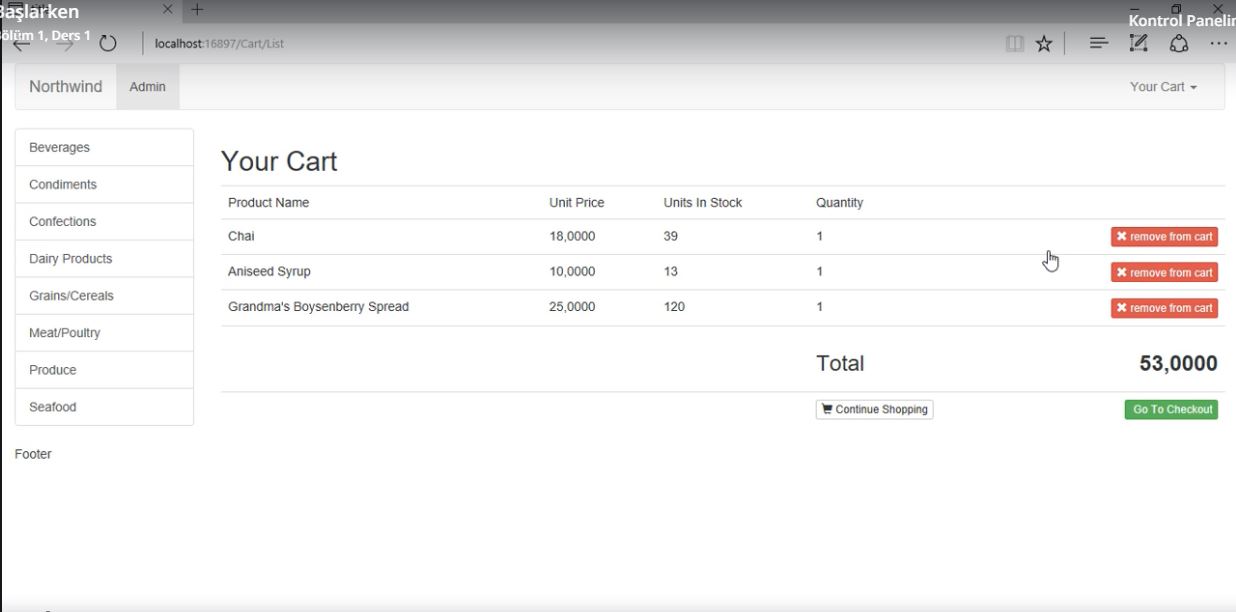


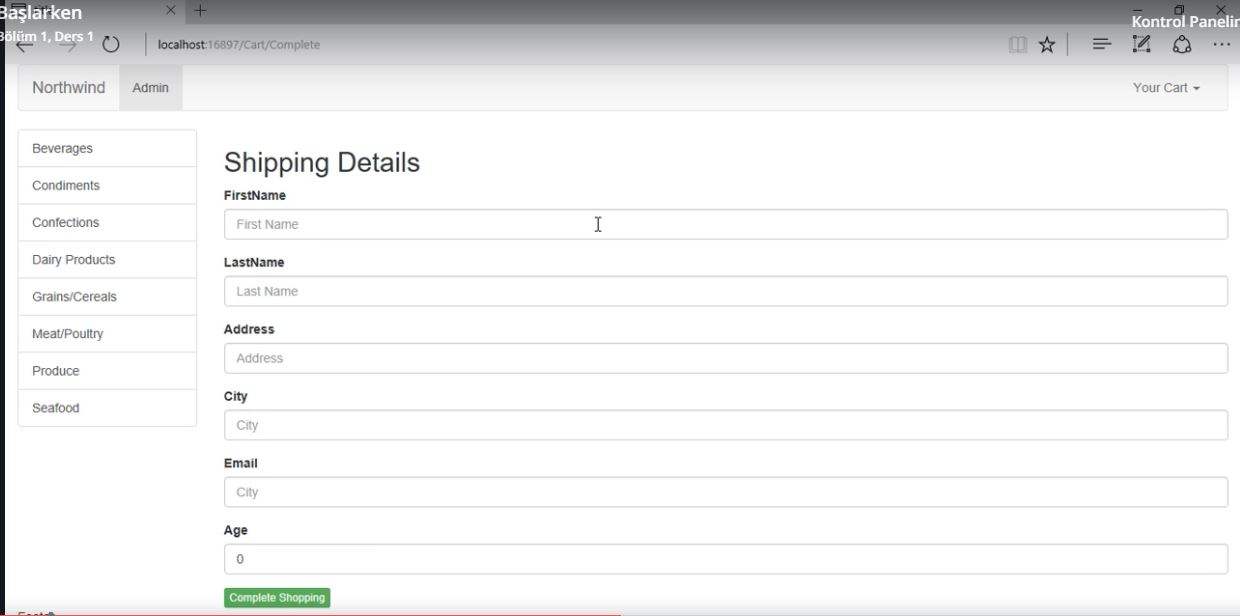


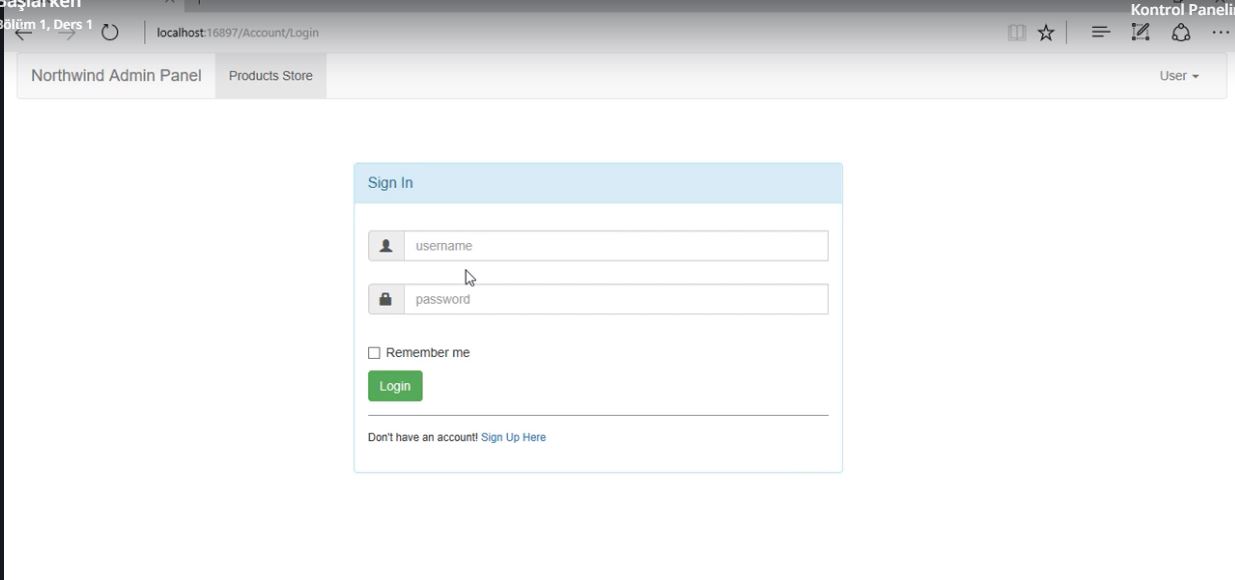


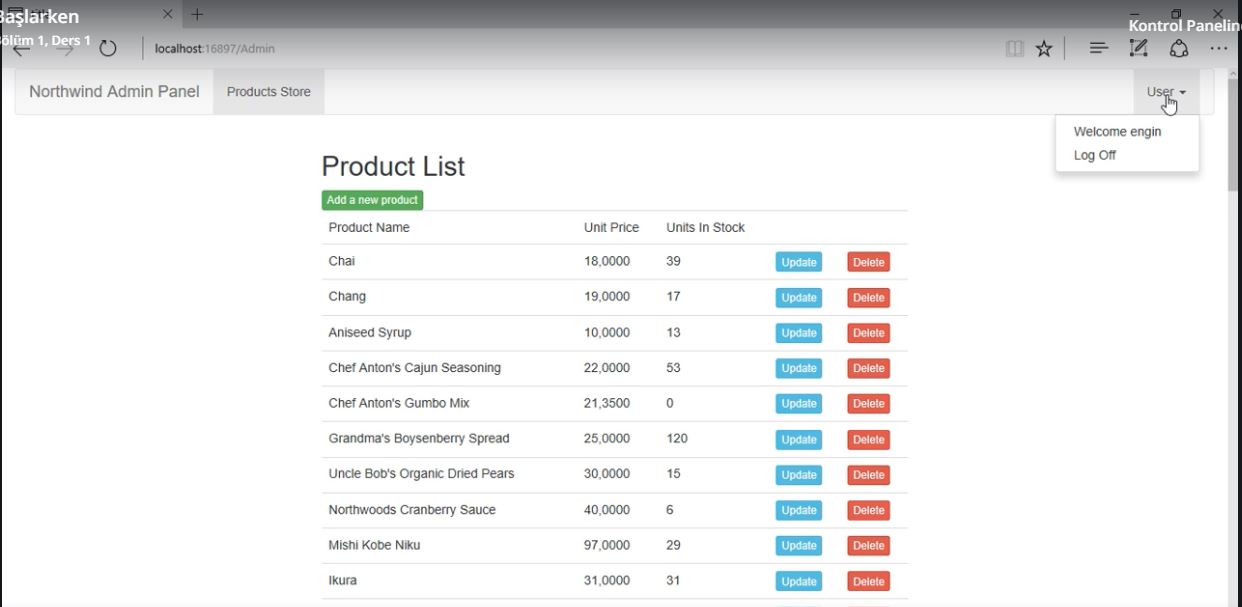


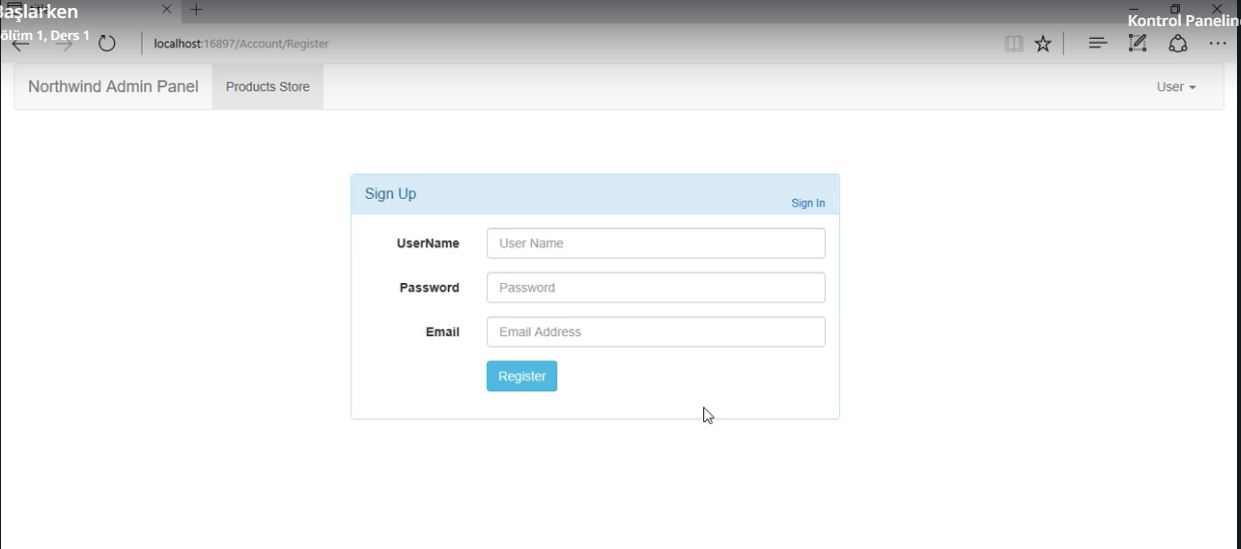


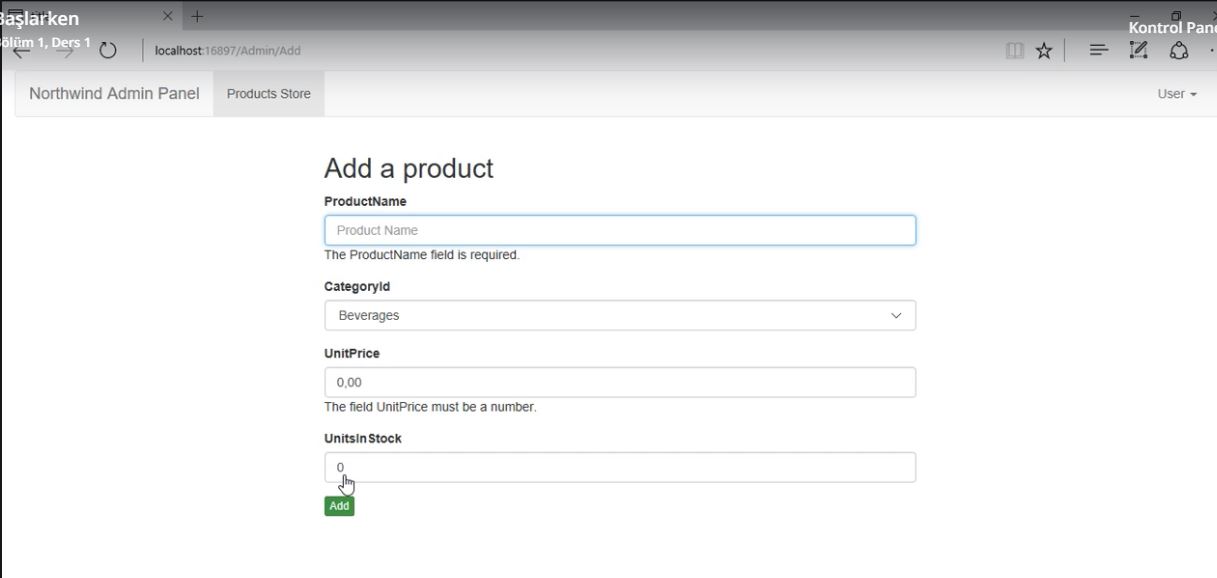


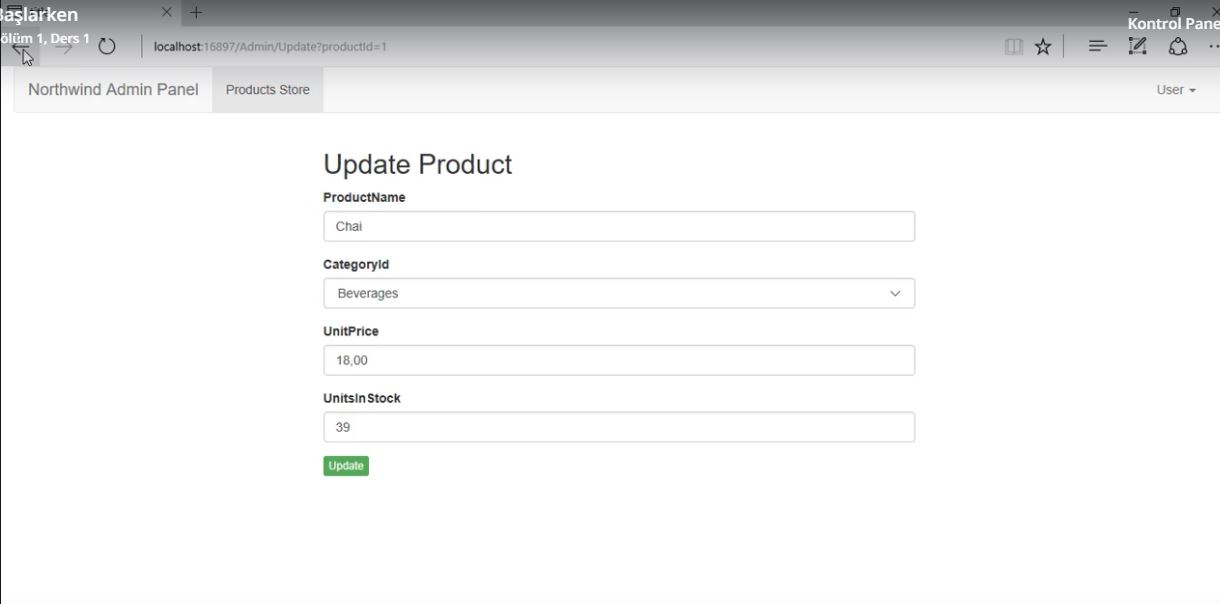








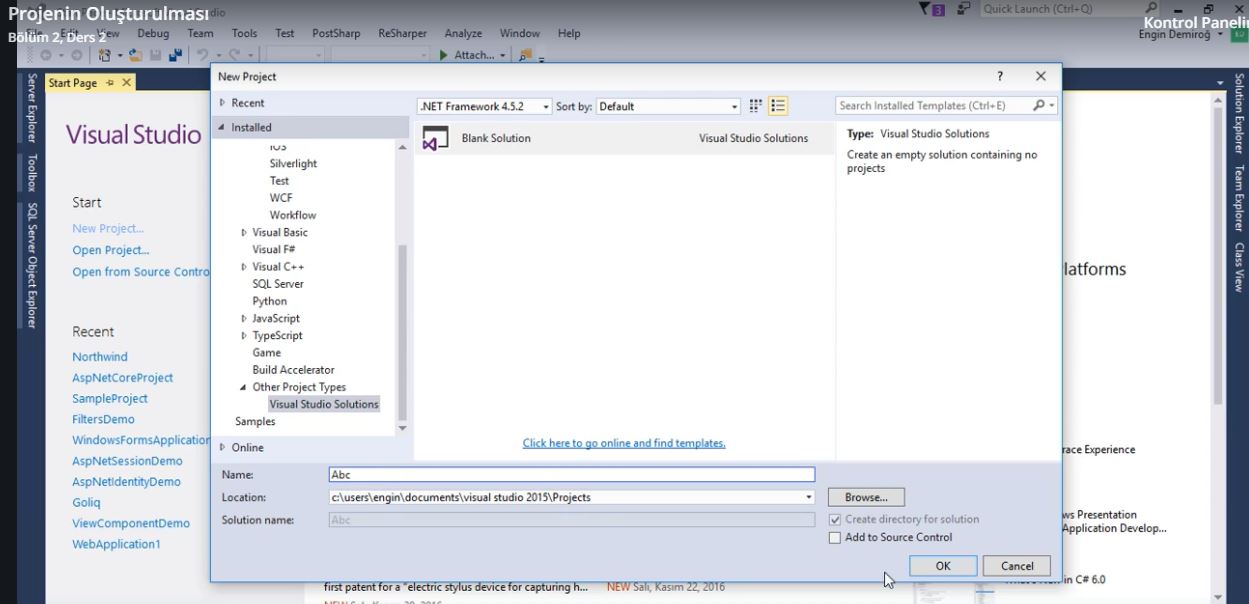


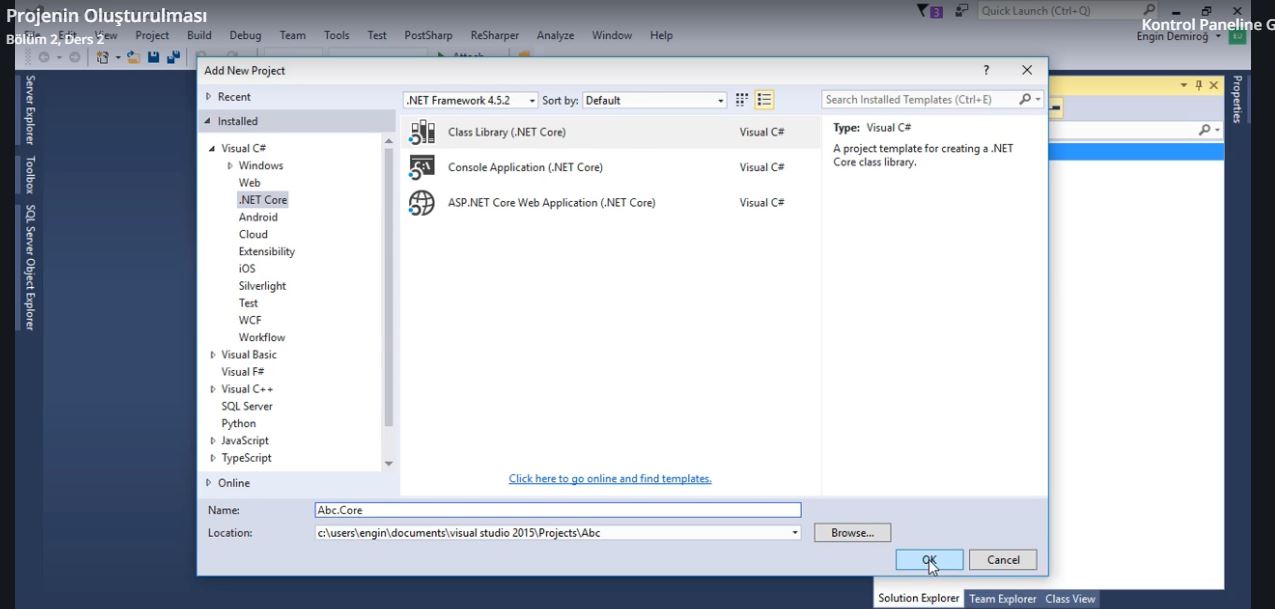


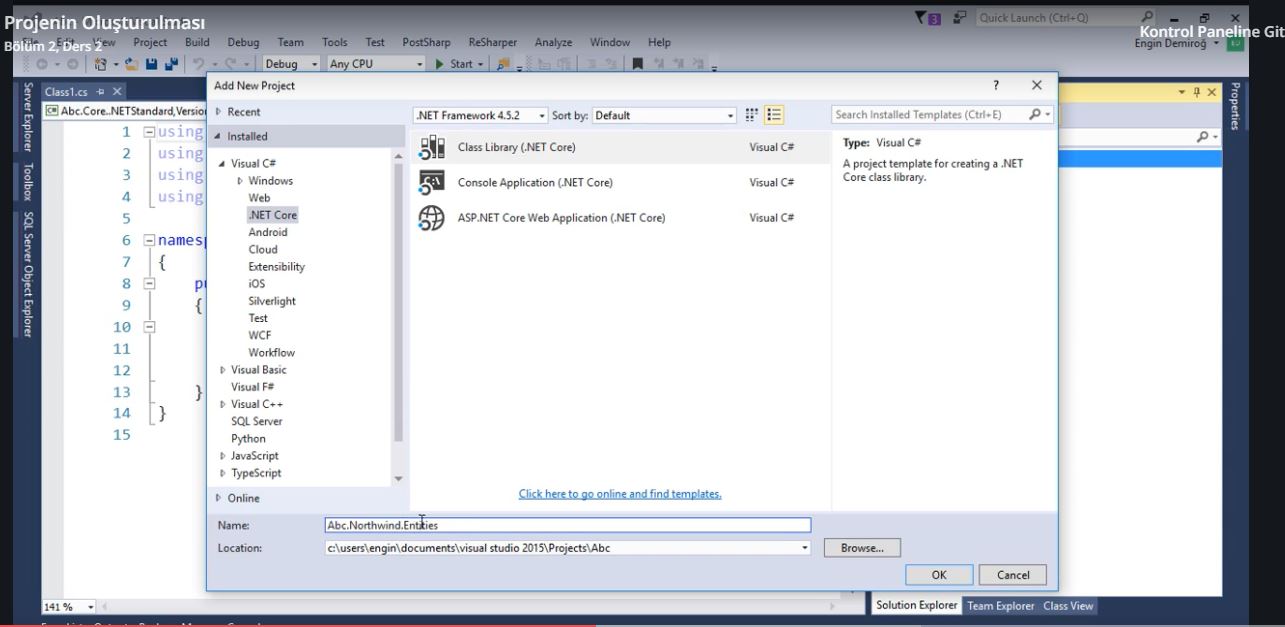
Bölüm 2

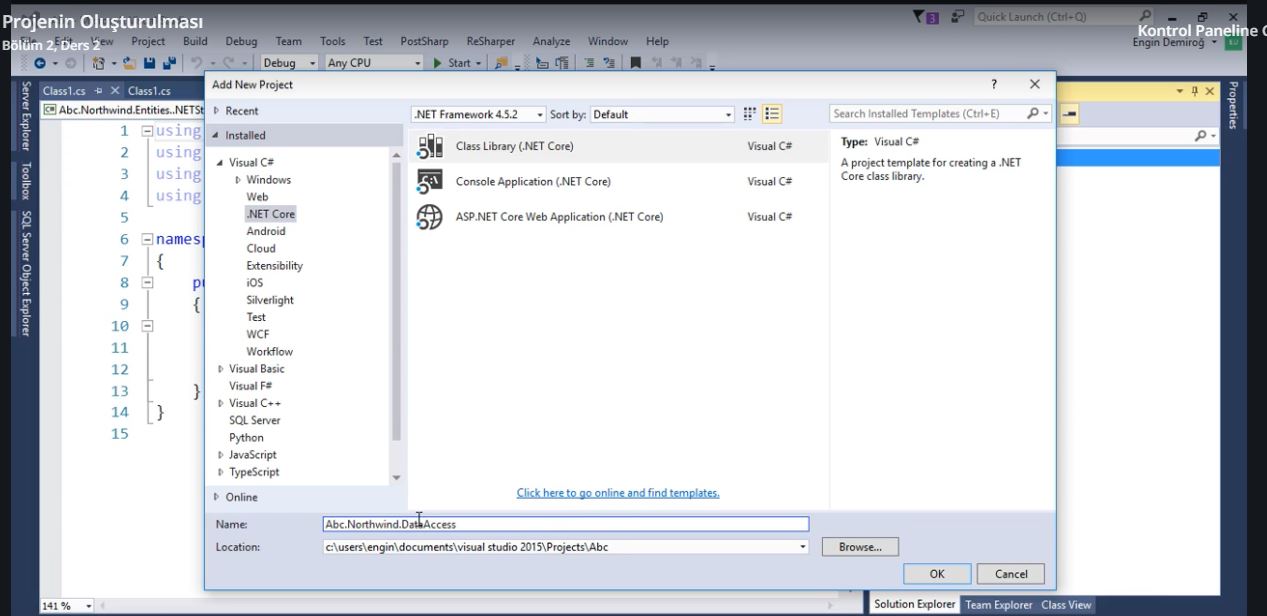
Projede yapılacak olanların görselleri böyledir. Şimdi ASP.NET Core ile katmanlı mimaride yapark projeyi gerçekleştirmeye geldi.

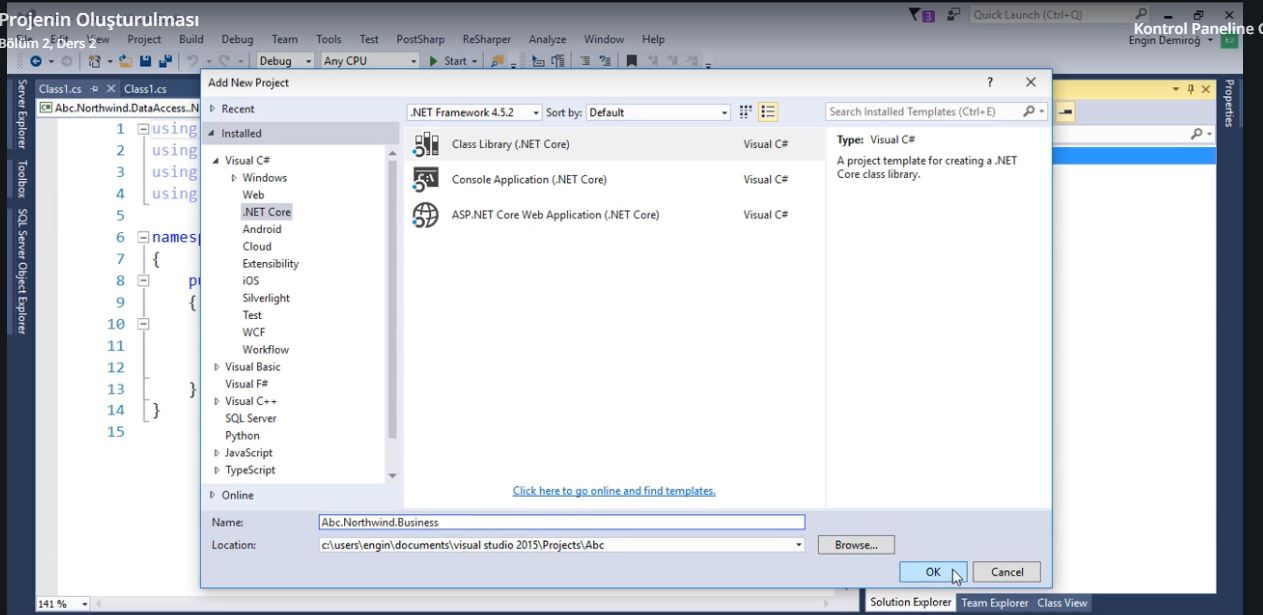
Proje oluşturma : Abc

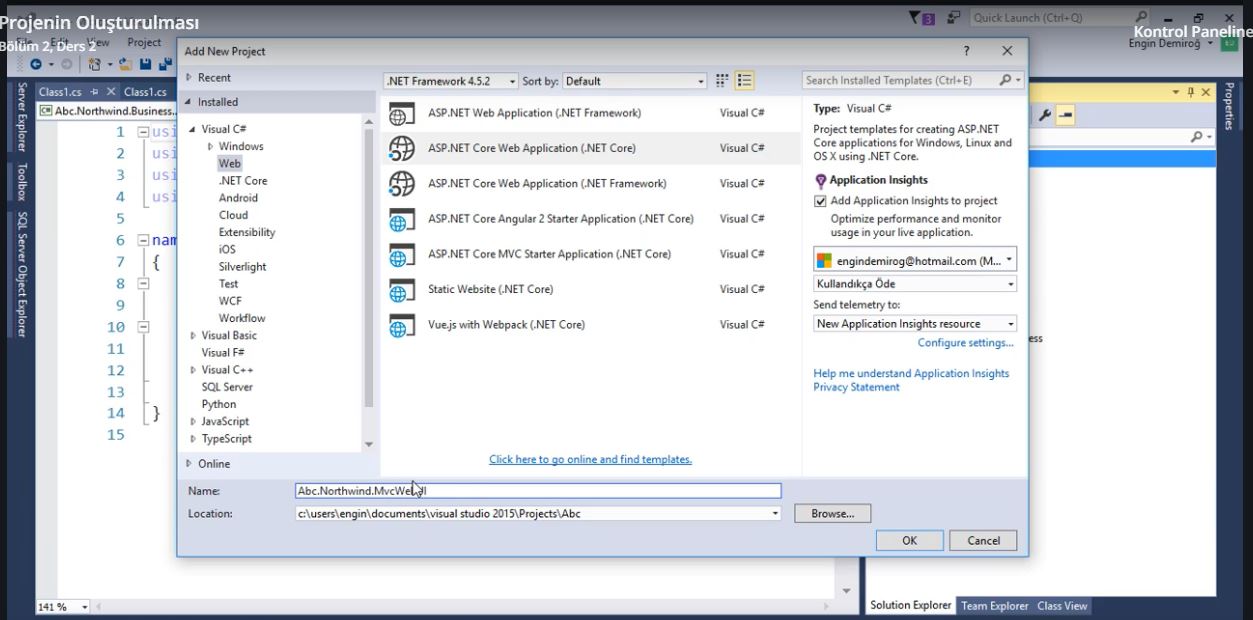


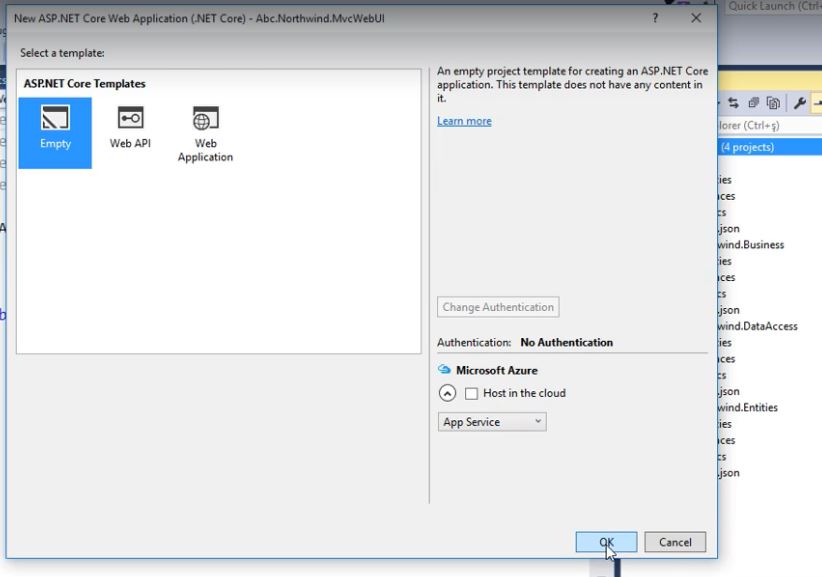












Bölüm 3

Abc.Entities katmanı ile başlıyoruz :

Veri tabanı tabloları, ve bunlarla birlikte oluşturulacak complex tipler(ilişkili iki tablonun birleştirilmesi) ayrıcı proje boyunca kullanılacak nesneler içerir.

Concrete ve ComplexTypes adında iki klasör oluşturuyoruz.

* Concrete VT tablolara karşılık gelen sınıflar oluşturulur.
* CompexTpes da İki tablonun birleşmiş halini tutan sınıflar oluşturulur.

using Abc.Core.Entities; //Entities de Core referans verdil IEntity için

namespace Abc.Entities

{ //IEntity Core da tanımlanmalı

public class Product : IEntity //IEntity : bu sınıf VT nesnesi oldugu söyler.

{

public int ProductId { get; set; }

public string ProductName { get; set; }

public decimal UnitPrice { get; set; }

public short UnitsInStock { get; set; }

public int CategoryId { get; set; }

}

}

* Core katmanında tüm projeler de kullanılacaklar yer alır Core + Entities + Ientity.cs interface oluşturduk.

using Abc.Core.Entities;

namespace Abc.Entities

{

public class Category:IEntity

{

public int CategoryId { get; set; }

public string CategoryName { get; set; }

}

}

Bölüm 4

Abc.DataAccess katmanı

* Sadece VT işlemlerinin yapıdığı katmandır. Insert,Update,Delete gibi.Sql komutların çalıştığı yer diye düşünebiliriz.

Abstract ve Concrete adında iki klasör oluşturuyoruz.

* Concrete : Somut nesneler EF ile ilgili işlemler yapılacaksa kod buraya yazılır.
* Abstract : İş katmanı ile iletişim kuran sınıflar (interface) olur. Sebebi yarın bir gün nhibernate kullanırsam kolayca geçebileyim diye.

Repository

Tüm sınıflar için CRUD kodları yazmak yerine bir kez yazarsak daha avantajlı olur buda Repository ile yapılır. Bunu tüm projelerde kullanabilmek için Generic yapıda Core katmanında oluşturmalıyız.

Core + DataAccess + IEntityRepository.cs (Interface olustururum.)

using Abc.Core.Entities;

using System;

using System.Collections.Generic;

using System.Linq.Expressions;

namespace Abc.Core.DataAccess

{

public interface IEntityRepository<T> where T : class, IEntity, new()

{

T Get(Expression<Func<T,bool>> filter=null); //id,string tc, gibi farklı alanlara göre getirme işlemlerimiçin expression yapılır.Parametre geçmezse filter=null

List<T> GetList(Expression<Func<T, bool>> filter = null); //null olursa tümünü listele

void Add(T entity);

void Update(T entity);

void Delete(T entity);

}

}

where = Generic kısıtlama

where : T class = Referansı class olacak

where : T class, IEntity = Sadece VT nesnesi olacak //neden çünkü VT da IUDC işlemlerini gerçekleştiriyoruz.

where : T class, IEntity, new() = Abstract ve Interface dısındakiler classlar yazılabilir.

Repository yazdık şimdi bu implement edilmeli, yine tüm projelerde kullanılması için Core + EntityFramework +EfEntityRepositoryBase.cs sınıfı oluşturup : IEntityRepository interfaceden kalıtım alır.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Linq.Expressions;

using Abc.Core.Entities;

using Microsoft.EntityFrameworkCore; //DbContext için

namespace Abc.Core.DataAccess.EntityFramework

{

//EF sorgu yazabilmek için Class(hangi nesneyle) ve Context 'e ihtiyac vardır.

public class EfEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>

where TEntity : class, IEntity, new()

where TContext : DbContext, new()

{

static TContext context = new TContext();

public void Add(TEntity entity)

{

var addedEntity = context.Entry(entity);

addedEntity.State = EntityState.Added;

context.SaveChanges();

}

public void Delete(TEntity entity)

{

var deletedEntity = context.Entry(entity);

deletedEntity.State = EntityState.Deleted;

context.SaveChanges();

}

public TEntity Get(Expression<Func<TEntity, bool>> filter )

{

return context.Set<TEntity>().SingleOrDefault(filter);

}

public List<TEntity> GetList(Expression<Func<TEntity, bool>> filter = null)

{

return filter == null //null ise ? false ise : bu çalışır.

? context.Set<TEntity>().ToList() //null değilse parametre gelmiş

: context.Set<TEntity>().Where(filter).ToList();

}

public void Update(TEntity entity)

{

var updatedEntity = context.Entry(entity);

updatedEntity.State = EntityState.Modified;

context.SaveChanges();

}

}

}

DataAccess katmanda veri erişim arayüzlerinin tanımlanması

DataAccess + Abstract klasöründe

using Abc.Core.DataAccess; //Core katmanı

using Abc.Entities; //Entities katmanı

namespace Abc.DataAccess.Abstract

{

//Product için Repositorydeki tüm metodlar hazır olur.Product için özel metod tanımları olabilir veya ?var olan metodlar değiştirilebilir IPRoductDal kalıtım alanda.

public interface IProductDal : IEntityRepository<Product>

{

}

}

DataAccess + Concrete + EntityFramework klasöründe + EfProductDal.cs

using Abc.Core.DataAccess.EntityFramework;

using Abc.DataAccess.Abstract;

using Abc.Entities;

namespace Abc.DataAccess.Concrete.EntityFramework

{

public class EfProductDal : EfEntityRepositoryBase<Product,NorthwindContext>,IProductDal

{

}

}

Category için

using Abc.Core.DataAccess;

using Abc.Entities;

namespace Abc.DataAccess.Abstract

{

public interface ICategoryDal : IEntityRepository<Category>

{

}

}

using Abc.Core.DataAccess.EntityFramework;

using Abc.DataAccess.Abstract;

using Abc.Entities;

namespace Abc.DataAccess.Concrete.EntityFramework

{

public class EfCategoryDal : EfEntityRepositoryBase<Category, NorthwindContext>, ICategoryDal

{

}

Context sınıfın olusturulması :

Veritabanına karşılık gelen obje yapısıdır. İçinde tablo yapısında karşılık gelen DbSet objelerini bulundurur.

DbContext kullanarak tablo ve view yapılarına erişebilir, DbSet yapısını kullanarak tablo üzerinde CRUD işlemlerini gerçekleştirebilirsiniz.

Projelerde birden fazla olusuturulabilir.

Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 2.1.1

using Abc.Entities;

using Microsoft.EntityFrameworkCore; //DbContext

namespace Abc.DataAccess.Concrete.EntityFramework

{

public class NorthwindContext : DbContext

{

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer(@"Data Source=DESKTOP-2TS36OR\SQLSERVER2017EXP;Initial Catalog=Northwind;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False");

}

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

}

}

Bölüm 5

Business katmanı

İş kurallarını yapıldığı veya yönetildiği katman. Cache,Validation,..gibi işlemler.

EF kodları yazılmaz session burda yapılmaz

Abc.DataAccess + Abstract ve Concrete adında iki klasör oluşturulur.

Product için

using Abc.Entities;

using System.Collections.Generic;

namespace Abc.Business.Abstract

{

public interface IProductService

{

List<Product> GetAll();

List<Product> GetByCategory(int categoryId);

void Add(Product product);

void Update(Product product);

void Delete(int productId);

}

}

using Abc.Business.Abstract;

using Abc.DataAccess.Abstract;

using Abc.Entities;

using System.Collections.Generic;

namespace Abc.Business.Concrete

{

public class ProductManager : IProductService

{

private IProductDal \_productDal;

public ProductManager(IProductDal productDal)

{

\_productDal = productDal;

}

public void Add(Product product)

{

\_productDal.Add(product);

}

public void Delete(int productId)

{

\_productDal.Delete(new Product { ProductId = productId }); //??

}

public List<Product> GetAll()

{

return \_productDal.GetList();

}

public List<Product> GetByCategory(int categoryId)

{

return \_productDal.GetList(p => p.CategoryId == categoryId);

}

public void Update(Product product)

{

\_productDal.Update(product);

}

}

}

Category için

using Abc.Entities;

using System.Collections.Generic;

namespace Abc.Business.Abstract

{

public interface ICategoryService

{

List<Category> GetAll();

List<Category> GetByCategory(int categoryId);

void Add(Category category);

void Update(Category category);

void Delete(int categoryId);

}

}

using Abc.Business.Abstract;

using Abc.DataAccess.Abstract;

using Abc.Entities;

using System;

using System.Collections.Generic;

using System.Text;

namespace Abc.Business.Concrete

{

public class CategoryManager : ICategoryService

{

private ICategoryDal \_categoryDal;

public CategoryManager(ICategoryDal categoryDal)

{

\_categoryDal = categoryDal;

}

public void Add(Category category)

{

throw new NotImplementedException();

}

public void Delete(int categoryId)

{

throw new NotImplementedException();

}

public List<Category> GetAll()

{

return \_categoryDal.GetList();

}

public List<Category> GetByCategory(int categoryId)

{

throw new NotImplementedException();

}

public void Update(Category category)

{

throw new NotImplementedException();

}

}

}

Üst katman alt katmana new lenemez.

Bölüm 6

WebUI katmanı : Core,Business,Entities referens alınır.

Kullanıcıya sunum yapıldığı yer. İş katmandaki kodlar Controllera konulmaz.

.WebUI'da

Controllers / Models / Views Klasörleri ekle.

Controllers Ekleme

Controllers + Add + Controller + Empty => ProductController

Views Ekleme

1.Controllerdaki metodun ismi ile sağtık AddView ile

2 Views + ControllerName + sağ + AddView + MetodName

\_Layout => Views + Shared + sag + Add New Item + web + Razor \_Layout

\_ViewStart => Views + sag + Add New ITem + web + Razor \_ViewStart

\_ViewImport => Views + sag +Add New Item + web + Razor \_VıewImport ==>html tag & View model eklemek ıcın

\_VıewImport

@using Edura.Models

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

wwwroot /css, / js, /img klasörleri olustur

wwwroot dısarı acılması ıcın startup.cs ayarı var.

????? bootstrap sonra ????

startup.cs

public class Startup

{

public IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public void ConfigureServices(IServiceCollection services)

{

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles(); // /css/style.css

app.UseStatusCodePages(); //404 hatası sayfaları gostermek ıcın

app.UseMvcWithDefaultRoute(); // aşağıdakine eşittir.kendi url yapısınıa göre olusturur.

/\*

app.UseMvc(routes => {

routes.MapRoute(

name: "default",

template: "{controller=Home}/{action=Index}/{id?}"

);

}); \*/ } } }

Bootstrap Tema ekle

C:\Users\hakan\Desktop\..\Edura.WebUI ==>> cmd.

npm i bootstrap //bootstrap@4.3.1 or bootstrap@4.0.0-alpha.6

npm init --yes //package.json dosyası

npm install

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{ ...

// /css/style.css

// /img/1.jpg

app.UseStaticFiles();

//node\_modules ismini kullanmıcaz. yoksa show AllFiles + node\_modules + sag + ınclude

// /modules/bootstrap/dist/css/bootstrap.min.css

app.UseStaticFiles(new StaticFileOptions

{

FileProvider = new PhysicalFileProvider(Path.Combine

(Directory.GetCurrentDirectory(), "node\_modules")),

RequestPath = "/modules"

});

...

}

Database Ayarı

.Entity : VT tablolara karsılık gelen sınıflar olusturulur . (?????????1\* -\*\* 11 ???????? sonra)

.Data DbContext için

.Data + Manage nuget => Microsoft.EntityFrameworkCore.SqlServer (2.1) yada Console Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 2.1.1

bunları ekleyelim.

<ItemGroup>

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.1.1" />

</ItemGroup>

<<<Edura.Data.Concrete.Context

public class EduraContext:DbContext // (1) .Entity proje referans edilmeli.(.WebUI da .Entity ve.Data ekle)

{

public EduraContext(DbContextOptions<EduraContext> options) // (2) Bu bloga gelen ConntectionString ile VT oluşacak

:base(options)

{

}

public DbSet<...> ...s { get; set; }

}

}

ConnectionString için

.Data + Edit aşaıdaki eklnemiş olur.

.WebUI + Add + New Item + AppSettingFile (An appsetting.json file used to configure Web Applications)

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=EduraDB;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

}

Sonra .WebUI startup.cs ayarı var

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<EduraContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"))); //ConnectionString ile BlogContext ilişkili oldu.

}

Migration Islemleri

DbContext => Database Migrations ile olur..

DbSet => Table

DbSet<Product> => Product(Table)

Buradaki amaç DbContexte uygun bir Veritabanı oluşturmak.

1.Kütüphane Kurma .

.WebUI+ sag + edit .csproj

<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.Dotnet" Version="2.1.1"></DotNetCliToolReference>

</ItemGroup>

</Project>

yada manage nuget ile Microsoft.EntityFrameworkCore.Tools.Dotnet

<<< bu paketi uygulamaya indirir.

2. Projede migration aktif etme.

Peoje yoluna cmd yazılır ve komut sistemi açılır.

dotnet ef migrations add InitialCreate <<< yazılır ve Migration eklenir projeye. aşahıdakiler DbSet ve Model deki lere göre otomatik oluşur.

Hata!!!

C:\Users\hakan\Desktop\CoreProject\BlogApp\BlogApp.WebUI>dotnet ef migrations add initial

infoYour target project 'BlogApp.WebUI' doesn't match your migrations assembly 'BlogApp.Data'. Either change your target project or change your migrations assembly.

Change your migrations assembly by using DbContextOptionsBuilder. E.g. options.UseSqlServer(connection, b => b.MigrationsAssembly("BlogApp.WebUI")). By default, the migrations assembly is the assembly containing the DbContext.

Change your target project to the migrations project by using the Package Manager Console's Default project drop-down list, or by executing "dotnet ef" from the directory containing the migrations project.: Microsoft.EntityFrameworkCore.Infrastructure[10403]

Entity Framework Core 2.1.8-servicing-32085 initialized 'BlogContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None

Bu Hatanın cozumu!!!

.WebUI + startup.cs

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<EduraContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"),b=>b.MigrationsAssembly("Edura.WebUI"))); //ConnectionString ile BlogContext ilişkili oldu.

Sonra!!!

Tekrar build et : dotnet ef migrations add InitialCreate

Sonra!!!

dotnet ef database update : komut ile VT Localde oluşmus olur //SeedData ile yapıyorduk burda komut ile yaptık tabi komu o projenın yoluna. Olustugu an update annı.

Sonra!!! VT olusutugu an.

LocalDb de VT olustugu gorulur.

///

///

///

<<< Modelin tablosuna otomatik veri ekleme test verileri için static SeedData sınıfında Seed metodu olusturma

namespace Edura.Data.Concrete.Context

{

public static class SeedData

{

public static void Seed (IApplicationBuilder app)

{

EduraContext context = app.ApplicationServices.GetRequiredService<EduraContext>();

context.Database.Migrate(); //VT nın oluştuğu an.

if (!context.Categories.Any()) //Herhangi Bir kayıt yoksa.

{

context.Categories.AddRange(

new Category() { Name="Category 1"},

new Category() { Name = "Category 2" },

new Category() { Name = "Category 3" }

);

context.SaveChanges();

}

if (!context.Blogs.Any())

{

context.Blogs.AddRange(

new Blog() { Title = "T1", Description = "D1", Body = "B1", Image = "1.jpg", Date = DateTime.Now.AddDays(-5), isApproved=true, CategoryId=1 },

new Blog() { Title = "T2", Description = "D2", Body = "B2", Image = "2.jpg", Date = DateTime.Now.AddDays(-6), isApproved = true, CategoryId = 1 },

new Blog() { Title = "T3", Description = "D3", Body = "B3", Image = "3.jpg", Date = DateTime.Now.AddDays(-7), isApproved = false, CategoryId = 2 },

new Blog() { Title = "T4", Description = "D4", Body = "B4", Image = "4.jpg", Date = DateTime.Now.AddDays(-8), isApproved = true, CategoryId = 3 }

);

context.SaveChanges();

}

}

}

}

Sonra!!!

<<< Sonra Startup + Configurede SeedData ayarı yapılmalı.

SeedData.Seed(app);

Sonra!!!

<<< Program.cs de UseDefaultServiseProvider eklenir.

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>()

.UseDefaultServiceProvider(options => options.ValidateScopes = false);

Sonra!!!

<<<Şimdi Modele propertyi ekleyelim yada değişiklik olunca

Sonra projenın bulundugu klasörün yoluna C:\Users\hakan\Desktop\CoreProject\EFCore\EFCore cmd ile açalım

Yeni bir migration ekleyelim

dotnet ef migrations add Yeni

Sonra!!!

Vt eklenmedi Proje çalısınca context.Database.Migrate() ile Vt eklenir yada cmd ile dotnet ef database update

Sonra!!!

Eger Migration içinde önceki sınıflara gitmek ıstersek dotnet ef database update "sınıfAdı"

Örn : dotnet ef database update initial

Controller

using Abc.Business.Abstract;

using Microsoft.AspNetCore.Mvc;

namespace Abc.WebUI.Controllers

{

public class ProductController : Controller

{

private IProductService \_productService;

public ProductController(IProductService productService)

{

\_productService = productService;

}

public IActionResult Index()

{

return View();

}

}

}

Dependenct Injection

public void ConfigureServices(IServiceCollection services)

{

//D.ınjection

services.AddScoped<IUserService, UserManager>();

services.AddScoped<IUserDal, EfUserDal>();

startupda yapılandırılıması lazım

Transiet -Singleton olarakda yapabilirdik

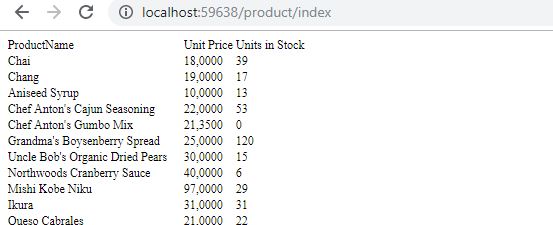
AddSingleton : Bir kez instance olusur herkes bu instance kullanır. Ortak kullanım aykırı olmaması gerekir.

AddScoped : Yapılan her istekde ayrı Manager olusur. A kullanıcı aynı anda 2 tane service ihtiyac duyarsa onun ıcın aynı manager olturulur ve verilir.

AddTransiet : Yapılan her istekte ayrı manager olsurur. Burda ise aynı durumda 2 tane manager olusturulur.

Bölüm 7

View Nesnesine Model Göndermek :



Controllerda yapımı :

public IActionResult Index()

{

var products = \_productService.GetAll();

**ProductListViewModel** model = new ProductListViewModel()

{

Products = products

};

return View(model); ÇOK ONEMLI VERILERI VIEWMODELS OLARAK YOLLA KURAL EDIN

}

Bir tane modelView oluşturarak gönderiyoruz.

using Abc.Entities;

using System.Collections.Generic;

namespace Abc.WebUI.Models

{

public class ProductListViewModel

{

public List<Product> Products { get; internal set; }

}

}

Index.cshtml de

@model **Abc.WebUI.Models.ProductListViewModel**

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Index</title>

</head>

<body>

<table>

<tr>

<td>ProductName</td>

<td>Unit Price</td>

<td>Units in Stock</td>

</tr>

@**foreach (var product in Model.Products)**

{

<tr>

<td>@product.ProductName</td>

<td>@product.UnitPrice</td>

<td>@product.UnitsInStock</td>

</tr>

}

</table>

</body>

</html>

**Bölüm 10 : Custom Middleware**

Node Package paketin yüklenmesi (npm)

1.WebUI + Add New Item + (npm yaz)

Package.json yüklenirken gizli dosyalarda node\_modules adın da bir klasöün oluştuğu görülür.

>>> Bootstrap,jquery gibi paketler yüklenir.

{

"version": "1.0.0",

"name": "asp.net",

"private": true,

"devDependencies": {

}

**"dependencies": {**

**"bootstrap": "3.3.7",**

**"jquery": "3.1.1",**

**"jquery-validation": "1.16.0",**

**"jquery-validation-unobtrusive": "3.2.6"**

**}**

}

Kayıt edilir edilmez dosya paketlerin install edilir.

2.Bir başka yöntem WebUI bulundugu dizinde cmd açılır ve;

npm i bootstrap //bootstrap@4.3.1 or bootstrap@4.0.0-alpha.6

npm init --yes //package.json dosyası

npm install

>>> paketler yüklendi şimdi bunalrı kullanmaya yapılandırmak için ; (1 ve 2 içinde geçerli)

Custom Middleware

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{ ...

**// /css/style.css**

**// /img/1.jpg**

app.UseStaticFiles(); **//middleware wwwroot klasörü için**

**//node\_modules ismini kullanmıcaz. yoksa show AllFiles + node\_modules + sag + ınclude**

**// /modules/bootstrap/dist/css/bootstrap.min.css**

app.UseStaticFiles(new StaticFileOptions

{

FileProvider = new PhysicalFileProvider(Path.Combine

(Directory.GetCurrentDirectory(), "node\_modules")),

RequestPath = "/modules"

});

...

}

>>> Yada kendi middleware oluşturarak Extension method olur.

WebUI da Middlewares klasörü oluştur ve Bir tane ApplicationBuilderExtension oluştur.

namespace Abc.WebUI.Middlewares

{

public static class ApplicationBuilderExtensions

{

public static IApplicationBuilder UseNodeModules(this IApplicationBuilder app, string root)

{

var path = Path.Combine(root, "node\_modules");

var provider = new PhysicalFileProvider(path);

var opt = new StaticFileOptions();

opt.RequestPath = "/node\_modules";

opt.FileProvider = provider;

app.UseStaticFiles(opt);

return app;

}

}

}

&&&

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseStaticFiles(); // /css/style.css

app.UseNodeModules(env.ContentRootPath); // eklenen bootstrap middleware için.

**11.Bölüm Client Side Framework Entegrasyon**

Yüklenen paketleri (bootstrap,..) projemizde kullanmak için.

Node\_modules klasöründe olan paketleri layout sınıfına referansları yazmamız gerek.

<head>

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

<link href="/node\_modules/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />

<script src="/node\_modules/bootstrap/dist/js/bootstrap.js"></script>

<script src="/node\_modules/jquery/dist/jquery.min.js"></script>

</head>

Yukardaki gibi yazılır.

Sonra <https://getbootstrap.com/docs/4.3/components/navbar/> sitesinden bir navbar kullanarak layouta ekledik.

**12.Bölüm ViewComponents**

Kullanım mantığı partial view gibi çalışması farklıdır. Verileri gidip kendisi alır ViewComponents’de.

* Category menu hazırladık, ancak menu dataları içinde bulunduğu view’in model içinde almak zorundaydı.(Partial View’da) Dolayısıyla bu işlemi her view için tekrarlamak zorunda olmamız kod kalabalıgıdır. Bağımlılık söz konusudur.
* Partial View ile sayfaya modeli biz gönderirken burda Component veriyi kendisi gidip alır verileri.
* Controller yapısı vardır yanı HomeControllerda Home kullanıllır gıbı ayrıca Controller dan kalıtım alırken bu viewComponent den kalıtım alır

Component gidip verileri alsın...

İsimlendirme önemli SonuViewComponent olmalı.

namespace Abc.WebUI.ViewComponents{

public class CategoryListViewComponent:ViewComponent {

private readonly ICategoryService \_categoryService;

public CategoryListViewComponent(ICategoryService categoryService) {

\_categoryService = categoryService;

}

public ViewViewComponentResult Invoke() {

var model = new **CategoryListViewModel**() {

**Categories** = \_categoryService.GetAll()

};

return View(model); //=> View'da bir ViewComponent oluşturulmalı.

//=> Views/Shared/Components/CategoryList/Default.cshtml

&&&

namespace Abc.WebUI.Models{

public class CategoryListViewModel {

public List<Category> Categories { get; internal set; }

}

&&&

@model Abc.WebUI.Models.CategoryListViewModel

<div class="list-group-item">

@foreach (var category in Model.Categories) {

<a class="list-group-item">@category.CategoryName</a>

}

</div>

&&&

<div class="col-md-2">

@await Component.InvokeAsync("CategoryList")

</div>

Sonuç Sol tarafta Veritabandan verileri çekip category menu oluştu.

**13.Bölüm Model Binding**

>>>Amacımız Ürünlerde **sayfalama** özelliği getirmek.

public IActionResult Index(int **page=1**){

int **pageSize** = 10;

var products = \_productService.GetAll();

ProductListViewModel model = new ProductListViewModel(){

Products = **products.Skip((page - 1) \* pageSize).Take(pageSize).ToList()**

};

return View(model);

}

Mantık, Şuan için sayfa dinamik, sayfada kaç tane olucak ise statik tanımladık.Sayfa atlama aşağıdaki url gibi olur.

<http://localhost:59638/product/index?page=2>

İstenilen sayfada PageSize kadar olması için küçük bir algoritma yazıldı.

>>>Şimdi ise ürünleri istenilen category’e göre getirmek için,

<http://localhost:59638/product/index?Page=1&category=1>

<http://localhost:59638/product/index?Page=2&category=2>

public IActionResult Index(int page=1, **int category=0**){

int pageSize = 10;

var products = \_productService.**GetByCategory(category);**

ProductListViewModel model = new ProductListViewModel(){

Products = products.Skip((page - 1) \* pageSize).Take(pageSize).ToList()

};

return View(model);

}

Yukardaki gibi çalışınca sıkıntı olmuyor ancak aşağıdaki gibide olunca listelenmesi için

<http://localhost:59638/product/index>

<http://localhost:59638/product/index?Page=1>

public List<Product> GetByCategory(int categoryId)

{

return \_productDal.GetList(p => p.CategoryId == categoryId **|| categoryId == 0**);

}

İşlemi uygulanır.

>>> Şimdi Category Tıklayınca değişmesi için ComponentList’de ki default.cshmtl de ıd alanı linke eklenir.

<http://localhost:59638/product/index?category=2&page=2>

@model Abc.WebUI.Models.CategoryListViewModel

<div class="list-group-item">

@foreach (var category in Model.Categories)

{

<a **href="/product/index?category=@category.CategoryId"** class="list-group-item">@category.CategoryName</a>

}

</div>

**14.Bölüm Razor Syntax**

Category **menude seçilen itemın belirgin hale getirmek için** query stringden category değerini yakalamam gerekicek dolayısıyla ilk önce CategoryListViewComponent’de

namespace Abc.WebUI.ViewComponents

{

public class CategoryListViewComponent:ViewComponent

{

private readonly ICategoryService \_categoryService;

public CategoryListViewComponent(ICategoryService categoryService)

{

\_categoryService = categoryService;

}

public ViewViewComponentResult Invoke()

{

var model = new CategoryListViewModel()

{

Categories = \_categoryService.GetAll(),

**CurrentCategory = Convert.ToInt32(HttpContext.Request.Query["category"])**

};

return View(model); //=> View'da bir ViewComponent oluşturulmalı.

//=> Views/Shared/Components/CategoryList/Default.cshtml

}

}

&&&

namespace Abc.WebUI.Models

{

public class CategoryListViewModel

{

public List<Category> Categories { get; internal set; }

**public int CurrentCategory { get; set; }**

}

}

&&&

Default.cshtml

@model Abc.WebUI.Models.CategoryListViewModel

<div class="list-group-item">

@foreach (var category in Model.Categories)

{

**var css = "list-group-item";**

**if (category.CategoryId == Model.CurrentCategory)**

**{**

**css += "active";**

**}**

<a href="/product/index?category=@category.CategoryId" **class="@css">@**category.CategoryName</a>

}

</div>

Sonuc olarak Menuden seçilenin, seçildiğinin class değiştirme.

**15.Bölüm Custom Tag Helper**

Genel bir sayfalama yazılacak, diğer controller metodlarıda kullanabilsin diye. Asp.net Core kendine özel Tag helperları var biz burda yeni bir tag helper oluşturacağız.

public IActionResult Index(int page=1, int category=0)

{

int pageSize = 10;

var products = \_productService.GetByCategory(category);

ProductListViewModel model = new ProductListViewModel()

{

//sol tarftakiler ProductListViewModel deki propertyiler.

**Products = products.Skip((page - 1) \* pageSize).Take(pageSize).ToList(),**

**PageCount = (int)Math.Ceiling(products.Count / (double)pageSize),**

**PageSize = pageSize,**

**CurrentCategory = category,**

**CurrentPage = page**

};

return View(model);

}

}

&&&

Tag Helper sınıfı

namespace Abc.WebUI.TagHelpers

{

[HtmlTargetElement("product-list-pager")]

public class PagingTagHelper : TagHelper

{

[HtmlAttributeName("page-size")]

public int PageSize { get; set; }

[HtmlAttributeName("page-count")]

public int PageCount { get; set; }

[HtmlAttributeName("current-category")]

public int CurrentCategory { get; set; }

[HtmlAttributeName("current-page")]

public int CurrentPage { get; set; }

public override void Process(TagHelperContext context, TagHelperOutput output)

{

output.TagName = "div";

StringBuilder stringBuilder = new StringBuilder();

stringBuilder.Append("<ul class='pagination'>");

for (int i = 1; i <= PageCount; i++)

{

stringBuilder.AppendFormat("<li class='{0}'>", i == CurrentPage ? "active" : "");

stringBuilder.AppendFormat("<a href='/product/index?page={0}&category={1}'>{2}</a>",i,CurrentCategory,i);

stringBuilder.Append("</li>");

}

output.Content.SetHtmlContent(stringBuilder.ToString());

base.Process(context, output);

}

}

**16.Bölüm ViewImports**

Oluşturulan pager(Tag Helperı) projede kullanma.

@model ProductListViewModel

@\*ViewImportsda tanımı yapıldı uzun uzun yazmaktan kurtulduk.\*@

<!DOCTYPE html>

<div>

<table class="table table-responsive">

<tr>

<td>ProductName</td>

<td>Unit Price</td>

<td>Units in Stock</td>

</tr>

@foreach (var product in Model.Products)

{

<tr>

<td>@product.ProductName</td>

<td>@product.UnitPrice</td>

<td>@product.UnitsInStock</td>

</tr>

}

</table>

</div>

**<product-list-pager current-category="@Model.CurrentCategory"**

**current-page="@Model.CurrentPage"**

**page-count="@Model.PageCount"**

**page-size="@Model.PageSize">**

**</product-list-pager>**

&&&

\_ViewImports.cshtml

**@using Abc.WebUI.Models**

**@addTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers"**

**@addTagHelper "\*, Abc.WebUI"**

**17.Bölüm Session**

Bundan önce Aynı kullanıcı bir istek yaptığında ve ardındanikinci istek yaptığında ilk istekten farklıdır. Kısacası server isteğin kimin yaptığını bilmediğinden olur. Stateless bir yapı durum yönetim bakımından zayıftır.

Şimdi Session ile hangikullanıcıdan istek gelimiş o bilgileri tutacagız.

Asp.NetCore Session

Startup.cs ayarları yapılır.

#region Session

services.AddSession();

services.AddDistributedMemoryCache();//cahce de tutulur.

#endregion

&&&

app.UseSession(); //Session

&&&

String yada int olarak tutmak yerine bir obje olarak tutmak daha mantıklıdır. Objeleride serileştirme ve deserileştirme işlemleri yapmalıyız.

namespace Abc.WebUI.ExtensionMethods

{

public static class SessionExtensionMethod

{

public static void SetObject(this ISession session, string key, object value)

{

string objectString = JsonConvert.SerializeObject(value);

session.SetString(key, objectString);

}

public static T GetObject<T>(this ISession session,string key) where T : class

{

string objectString = session.GetString(key);

if (string.IsNullOrEmpty(objectString))

{

return null;

}

T value = JsonConvert.DeserializeObject<T>(objectString);

return value;

}

}

}

&&&

Müsteri ürünü sepete ekleyebilir ve satın alabilir. Bunun için Session ile besleniyor olması gerekir.;

Sepetteki elemanlar

namespace Abc.Entities

{

public class CartLine

{

public Product Product { get; set; }

public int Quantity { get; set; } //üründen kac tane alınacağını tutar.

}

}

Sepet

namespace Abc.Entities

{

public class Cart

{

public Cart()

{

//obj. null ref. hatası almamak için.

CartLines = new List<CartLine>();

}

public List<CartLine> CartLines { get; set; }

public Decimal Total

{

get { return CartLines.Sum(c => c.Product.UnitPrice \* c.Quantity); }

}

}

}

&&&

Sepette işlemler(ekleme,…) yapabilmesi için

namespace Abc.Business.Abstract

{

public interface ICartService {

void AddToCart(Cart cart, Product product);

void RemoveFromCart(Cart cart, int productId);

List<CartLine> List(Cart cart);

}

}

&&&

namespace Abc.Business.Concrete{

public class CartManager : ICartService {

public void AddToCart(Cart cart, Product product) {

CartLine cartLine = cart.CartLines.FirstOrDefault(c => c.Product.ProductId == product.ProductId);

if(cartLine != null) {

cartLine.Quantity++;

return;

}

cart.CartLines.Add(new CartLine() { Product = product, Quantity = 1 });

}

public List<CartLine> List(Cart cart) {

return cart.CartLines;

}

public void RemoveFromCart(Cart cart, int productId) {

cart.CartLines.Remove(cart.CartLines.FirstOrDefault(c => c.Product.ProductId == productId));

}

}

}

&&&

İndex de cart ekleme alanı eklendi

<table class="table table-responsive">

<tr>

<td>ProductName</td>

<td>Unit Price</td>

<td>Units in Stock</td>

**<td></td>**

</tr>

@foreach (var product in Model.Products)

{

<tr>

<td>@product.ProductName</td>

<td>@product.UnitPrice</td>

<td>@product.UnitsInStock</td>

**<td><a class="btn btn-xs btn-success" asp-controller="Cart" asp-action="AddToCart" asp-route-productId="@product.ProductId">add to cart</a></td>**

</tr>

}

</table>

Add to cart tıklayınca => <http://localhost:59638/Cart/AddToCart?productId=1> olur.

Product Service GetById metodu eklenicek

namespace Abc.Business.Abstract{

public interface IProductService {

List<Product> GetAll();

List<Product> GetByCategory(int categoryId);

void Add(Product product);

void Update(Product product);

void Delete(int productId);

**Product GetById(int productId);**

}

Managerında ise

public Product GetById(int productId)

{

**return \_productDal.Get(p => p.ProductId == productId);**

}

&&&

namespace Abc.WebUI.Services{

public interface ICartSessionServices {

//session oku(get)/ yaz(set)

Cart GetCart();

void SetCart(Cart cart);

}

}

namespace Abc.WebUI.Services{

public class CartSessionService : ICartSessionServices {

//neden yazılıd? çünkü sessin Controllarda çalışır burdada çalışması için.

public IHttpContextAccessor \_httpContextAccessor;

public CartSessionService(IHttpContextAccessor httpContextAccessor) {

\_httpContextAccessor = httpContextAccessor;

}

public Cart GetCart() {

Cart cartToCheck = \_httpContextAccessor.HttpContext.Session.GetObject<Cart>("cart");

if (cartToCheck == null)

{

\_httpContextAccessor.HttpContext.Session.SetObject("cart", new Cart());

cartToCheck = \_httpContextAccessor.HttpContext.Session.GetObject<Cart>("cart");

}

return cartToCheck;

}

public void SetCart(Cart cart) {

\_httpContextAccessor.HttpContext.Session.SetObject("cart", cart);

}

}

}

Startup.cs eklenir.

services.AddSingleton<ICartService, CartManager>();

services.AddSingleton<ICartSessionService, CartSessionService>();

services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();

CartController

namespace Abc.WebUI.Controllers

{

public class CartController : Controller

{

private readonly ICartSessionService \_cartSessionService;

private readonly ICartService \_cartService;

private readonly IProductService \_productService;

public CartController(ICartSessionService cartSessionService, ICartService cartService, IProductService productService)

{

\_cartSessionService = cartSessionService;

\_cartService = cartService;

\_productService = productService;

}

public ActionResult AddToCart(int productId)

{

var productToBeAdded = \_productService.GetById(productId);

var cart = \_cartSessionService.GetCart();

\_cartService.AddToCart(cart, productToBeAdded);

\_cartSessionService.SetCart(cart);

TempData.Add("message", String.Format("Your product, {0}, was successfully added to the cart!", productToBeAdded.ProductName));

**return RedirectToAction("Index", "Product");**

} }

}

<http://localhost:59638/product/index> AddToCart’dan sonra

<http://localhost:59638/Product> olur.

TempData Tek bir requestlik yollanır. Layputta kullanılırken

<div class="col-md-10">

**@if(TempData["message"]!= null)**

**{**

**<div class="alert alert-info" role="alert">@TempData["message"]</div>**

**}**

@RenderBody()

</div>

**18.Bölüm Proje uygulanmasına devam**

**Cart Summary ViewComponent**

Yapılan işlem, Listeden ürünü sepete atınca sepete eklenmesi aynı ürün eklenirses +1 yapılması Sol üst köşede dropdown ile ViewComponent olucak şekilde yapılması.

namespace Abc.WebUI.Models

{

public class CartSummaryViewModel

{

public Cart Cart { get; set; }

}

&&&

namespace Abc.WebUI.ViewComponents

{

public class CartSummaryViewComponent : ViewComponent

{

ICartSessionService \_cartSessionService;

public CartSummaryViewComponent(ICartSessionService cartSessionService)

{

\_cartSessionService = cartSessionService;

}

public ViewViewComponentResult Invoke()

{

var model = new CartSummaryViewModel()

{

Cart = \_cartSessionService.GetCart()

};

return View(model);

}

}

}

&&&

@model CartSummaryViewModel

<ul class="dropdown-menu">

@foreach (var item in Model.Cart.CartLines)

{

<li>

<a>@item.Product.ProductName <span class="bedge">@item.Quantity</span> </a>

</li>

}

<li role="separator" class="driver"></li>

<li>

<a><span class="" aria-hidden="true"></span>@Model.Cart.Total</a>

</li>

<li role="separator" class="driver"></li>

<li>

<**a** **asp-controller**="Cart" **asp-action**="List">Go to Cart List</**a**>

</li>

</ul>

&&&

@await Component.InvokeAsync("CartSummary")

**Sepetin Listelenmesi**

namespace Abc.WebUI.Models

{

public class CartListViewModel

{

public Cart Cart { get; set; }

}

}

&&&

CartController’da

public ActionResult List()

{

var cart = \_cartSessionService.GetCart();

CartListViewModel cartListViewModel = new CartListViewModel()

{

Cart = cart

};

return View(cartListViewModel);

}

&&&

List.cshtml

@model CartListViewModel

<h2>Your Cart</h2>

<table class="table">

<tr>

<td>Product Name</td>

<td>Unit Price</td>

<td>Units In Stock</td>

<td>Quantity</td>

<td></td>

</tr>

@foreach (var item in Model.Cart.CartLines) {

<tr>

<td>@item.Product.ProductName</td>

<td>@item.Product.UnitPrice</td>

<td>@item.Product.UnitsInStock</td>

<td>@item.Quantity</td>

<td class="text-right"><**a** class="btn btn-xs btn-danger" **asp-controller**="Cart" **asp-action**="Remove" **asp-route-productId**="@item.Product.ProductId"><span class="glyphicon glyphicon-remove"></span> remove from cart</**a**></td>

</tr>

}

<tr>

<td></td>

<td></td>

<td></td>

<td>

<h3>Total</h3>

</td>

<td class="text-right">

<h3>

<strong>@Model.Cart.Total</strong>

</h3>

</td>

</tr>

<tr>

<td> </td>

<td> </td>

<td> </td>

<td>

<**a** **asp-controller**="Product" **asp-action**="Index" class="btn btn-xs btn-default">

<span class="glyphicon glyphicon-shopping-cart"></span> Continue Shopping

</**a**>

</td>

<td class="text-right">

<**a** **asp-controller**="Cart" **asp-action**="Complete" class="btn btn-xs btn-success">

<span class="glyphicon glyphicon-shopping-play"></span> Go To Checkout

</**a**>

</td>

</tr>

**Sepetten Silinmesi**

public ActionResult Remove(int productId)

{

var cart = \_cartSessionService.GetCart();

\_cartService.RemoveFromCart(cart, productId);

\_cartSessionService.SetCart(cart);

TempData.Add("message", String.Format("Your product, was successfully removed from the cart!"));

return RedirectToAction("List");

}

TempData daha önce Layoutta tanımlamıstık yani kullanmayı tanımlamıstık aynısı uzerinden çalışır.

**Model Validation**

Cart/List gotoCheckout Cart/Complete

using System.ComponentModel.DataAnnotations; //Bu yapı yerine Fluent kullanılabilir.

namespace Abc.Entities

{

public class ShippingDetails

{

[Required] //Alanın girilmesi gereklidir...

public string FirstName { get; set; }

[Required]

public string LastName { get; set; }

[Required]

public string City { get; set; }

[Required]

[DataType(DataType.EmailAddress)] //Email address formatında olmasını istemek.Password,DateTime,...

public string Email { get; set; }

[Required]

public string Address { get; set; }

[Required]

[Range(18,100)] //Min ve Max değerler arasında alabilir...

public int Age { get; set; }

}

}

&&&

using Abc.Entities;

namespace Abc.WebUI.Models

{

[HttpGet] //Form gönderme

public class ShippingDetailsViewModel

{

public ShippingDetails ShippingDetails { get; set; }

}

}

&&&

CartController’da

public ActionResult Complete()

{

var shippingDetailsViewModel = new ShippingDetailsViewModel//()

{

ShippingDetails = new ShippingDetails()

};

return View(shippingDetailsViewModel);

}

&&&

Şimdi form oluşturup doldurulacak post metodu.

Cart/Complete.cshtml

@model ShippingDetailsViewModel

<h2>Shipping Details</h2>

@\*Html Tag Helper\*@

<**form** **asp-controller**="Cart" **asp-action**="Complete" method="post">

<div class="form-group">

<**label** **asp-for**="ShippingDetails.FirstName"></**label**>

<**input** **asp-for**="ShippingDetails.FirstName" class="form-control" placeholder="First Name">

@\*Validation aykırı durumda hemen buraya uyarıyı yazar...ayrıca ilk açıldığında validation gözükmez\*@

<**span** **asp-validation-for**="ShippingDetails.FirstName"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="ShippingDetails.LastName"></**label**>

<**input** **asp-for**="ShippingDetails.LastName" class="form-control" placeholder="Last Name">

<**span** **asp-validation-for**="ShippingDetails.LastName"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="ShippingDetails.Address"></**label**>

<**input** **asp-for**="ShippingDetails.Address" class="form-control" placeholder="Address">

<**span** **asp-validation-for**="ShippingDetails.Address"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="ShippingDetails.City"></**label**>

<**input** **asp-for**="ShippingDetails.City" class="form-control" placeholder="City">

<**span** **asp-validation-for**="ShippingDetails.City"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="ShippingDetails.Email"></**label**>

<**input** **asp-for**="ShippingDetails.Email" class="form-control" placeholder="City">

<**span** **asp-validation-for**="ShippingDetails.Email"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="ShippingDetails.Age"></**label**>

<**input** **asp-for**="ShippingDetails.Age" class="form-control" placeholder="City">

<**span** **asp-validation-for**="ShippingDetails.Age"></**span**>

</div>

<input type="submit" value="Complete Shopping" class="btn btn-xs btn-success" />

</**form**>

Post metodu ise (TempData Layoutta çağırılıyor.)

[HttpPost]

public ActionResult Complete(ShippingDetails shippingDetails)

{

if(!ModelState.IsValid) //girilen model geçerli değilse

{

return View(); //hata mesajlarını gösterir.

}

TempData.Add("message", string.Format("Thank you {0},you ordeer is in process", shippingDetails.FirstName));

return View();

}

**Client Side Validation Entegrasyon**

<link href="/node\_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />

<script src="/node\_modules/jquery/dist/jquery.min.js"></script>

<script src="/node\_modules/bootstrap/dist/js/bootstrap.js"></script>

<script src="~/node\_modules/jquery-validation/dist/jquery.validate.js"></script>

<script src="~/node\_modules/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.js"></script>

<script src="/node\_modules/bootstrap/dist/js/bootstrap.min.js"></script>>

Paketler bu sırada olmalı ki client side aktifleşsin.

Sayfanın yenilenmeden hata mesajlarını gösterir.

**Admin Panelin Hazırlanması**

Admin Panelinde Index hazırlanması

namespace Abc.WebUI.Controllers

{

public class AdminController : Controller

{

private readonly IProductService \_productService;

public AdminController(IProductService productService)

{

\_productService = productService;

}

public IActionResult Index()

{

var productListViewModel = new ProductListViewModel

{

Products = \_productService.GetAll()

};

return View(productListViewModel);

}

}

}

&&&

Admin/Index hazırlanması

@model ProductListViewModel

@{

ViewBag.Title = "title";

Layout = "~/Views/Shared/\_AdminLayout.cshtml"; /\*\_AdminLayout kullandık\*/

}

<h2>Product List</h2>

@\*Tag Helper kullanımı\*@

<**a** class="btn btn-xs btn-success" **asp-controller**="Admin" **asp-action**="Add">Add a new product</**a**>

<div>

<table class="table table-responsive">

<tr>

<td>Product Name</td>

<td>Unit Price</td>

<td>Units In Stock</td>

<td></td>

<td></td>

</tr>

@foreach (var product in Model.Products)

{

<tr>

<td>@product.ProductName</td>

<td>@product.UnitPrice</td>

<td>@product.UnitsInStock</td>

<td><**a** class="btn btn-xs btn-info" **asp-controller**="Admin" **asp-action**="Update" **asp-route-productId**="@product.ProductId">Update</**a**></td>

<td><**a** class="btn btn-xs btn-danger" **asp-controller**="Admin" **asp-action**="Delete" **asp-route-productId**="@product.ProductId">Delete</**a**></td>

</tr>

}

</table>

</div>

\_AdminLayout Hazırlanması

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

<link href="/node\_modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />

<script src="/node\_modules/jquery/dist/jquery.min.js"></script>

<script src="/node\_modules/bootstrap/dist/js/bootstrap.js"></script>

<script src="~/node\_modules/jquery-validation/dist/jquery.validate.js"></script>

<script src="~/node\_modules/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.js"></script>

<script src="/node\_modules/bootstrap/dist/js/bootstrap.min.js"></script>

</head>

<body>

<div class="col-md-12">

<!--Satırın tamamını kapsar.-->

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="#"> Northwind Admin Panel </a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarSupportedContent">

<ul class="navbar-nav mr-auto">

<li class="nav-item active">

<a class="nav-link" href="#">Products Store <span class="sr-only">(current)</span></a>

</li>

</ul>

<ul class="nav navbar-nav navbar-right">

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

Dropdown

</a>

@\*//Login-Logout\*@

</li>

</ul>

</div>

</nav>

</div>

<div class="col-md-12"> @\*//bootstrap 12 alana böler\*@

@if (TempData["message"] != null)

{

<div class="alert alert-info" role="alert">@TempData["message"]</div>

}

@RenderBody()

</div>

<div class="col-md-12">

footer

</div>

</body>

</html>

**Admin Panelinde Ürün Ekleme**

Admin Controller da

[HttpGet]

public ActionResult Add()

{

return View();

}

[HttpPost]

public ActionResult Add(Product product)

{

if (ModelState.IsValid) //doğrulama başarılı ise...

{

\_productService.Add(product);

TempData.Add("message", "product was successfully added");

}

return View(); //aksi halde sayfayı yeniden göster...

}

&&&

Admin/Add.cshtml de

@model ProductAddViewModel //Dikkat model

@{

ViewBag.Title = "title";

Layout = "~/Views/Shared/\_AdminLayout.cshtml"; /\*\_AdminLayout kullandık\*/

}

<h2>Add a product</h2>

<**form** **asp-controller**="Admin" **asp-action**="Add" method="post">

<div class="form-group">

<**label** **asp-for**="Product.ProductName"></**label**>

@\*placeholder input içinde yazan kullanıcı bilgilendirmek için.\*@

<**input** **asp-for**="Product.ProductName" class="form-control" placeholder="Product Name" />

<**span** **asp-validation-for**="Product.ProductName"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.CategoryId"></**label**>

<**input** **asp-for**="Product.CategoryId" class="form-control" placeholder="Category Id"></select>

<**span** **asp-validation-for**="Product.CategoryId"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.UnitPrice"></**label**>

<**input** **asp-for**="Product.UnitPrice" class="form-control" placeholder="Unit Price" />

<**span** **asp-validation-for**="Product.UnitPrice"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.UnitsInStock"></**label**>

<**input** **asp-for**="Product.UnitsInStock" class="form-control" placeholder="Units In Stock" />

<**span** **asp-validation-for**="Product.UnitsInStock"></**span**>

</div>

<input type="submit" value="Add" class="btn btn-xs btn-success" />

</**form**>

**Category için Ürün Eklemede Select List Eklenmesi**

[HttpGet]

public ActionResult Add()

{

var model = new ProductAddViewModel

{

Product = new Product(),

Categories = \_categoryService.GetAll()

};

return View(model);

}

[HttpPost]

public ActionResult Add(Product product)

{

if (ModelState.IsValid) //doğrulama başarılı ise...

{

\_productService.Add(product);

TempData.Add("message", "product was successfully added");

}

return RedirectToAction("Add"); //o Action'ın çalışmasını isteriz. DIKKAT!!!

//return View(); //aksi halde sayfayı yeniden göster...

}

&&&

public class ProductAddViewModel

{

public List<Category> Categories { get; internal set; }

public Product Product { get; set; }

}

&&&

@model ProductAddViewModel

<div class="form-group">

<**label** **asp-for**="Product.CategoryId"></**label**>

<**select** **asp-for**="Product.CategoryId" class="form-control"

**asp-items**="@(new SelectList(Model.Categories,"CategoryId","CategoryName"))"></**select**>

<**span** **asp-validation-for**="Product.CategoryId"></**span**>

</div

**Admin Panalinde Ürünlerin Güncellenmesi**

[HttpGet]

public ActionResult Update(int productId)

{

var model = new ProductUpdateViewModel

{

Product = \_productService.GetById(productId),

Categories = \_categoryService.GetAll()

};

return View(model);

}

[HttpPost]

public ActionResult Update(Product product)

{

if (ModelState.IsValid) //doğrulama başarılı ise...

{

\_productService.Update(product);

TempData.Add("message", "product was successfully updated");

}

return RedirectToAction("Update"); //o Action'ın çalışmasını isteriz. DIKKAT!!!

}

&&&

public class ProductUpdateViewModel

{

public List<Category> Categories { get; internal set; }

public Product Product { get; set; }

}

Admin/Update.cshtml

@model ProductUpdateViewModel

@{

ViewBag.Title = "title";

Layout = "~/Views/Shared/\_AdminLayout.cshtml"; /\*\_AdminLayout kullandık\*/

}

<h2>Update Product</h2>

<**form** **asp-controller**="Admin" **asp-action**="Update" method="post">

<div class="form-group">

<**label** **asp-for**="Product.ProductName"></**label**>

<**input** **asp-for**="Product.ProductName" class="form-control" placeholder="Product Name" />

<**span** **asp-validation-for**="Product.ProductName"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.CategoryId"></**label**>

<**select** **asp-for**="Product.CategoryId" class="form-control"

**asp-items**="@(new SelectList(Model.Categories,"CategoryId","CategoryName"))"></**select**>

<**span** **asp-validation-for**="Product.CategoryId"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.UnitPrice"></**label**>

<**input** **asp-for**="Product.UnitPrice" class="form-control" placeholder="Unit Price" />

<**span** **asp-validation-for**="Product.UnitPrice"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="Product.UnitsInStock"></**label**>

<**input** **asp-for**="Product.UnitsInStock" class="form-control" placeholder="Units In Stock" />

<**span** **asp-validation-for**="Product.UnitsInStock"></**span**>

</div>

<**input** **type**="hidden" **asp-for**="Product.ProductId" />

<input type="submit" value="Update" class="btn btn-xs btn-success" />

</**form**>

**Admin Panelinde Ürünlerin Silinmesi**

public ActionResult Delete(int productId)

{

\_productService.Delete(productId);

TempData.Add("message", "product was successfully deleted");

return RedirectToAction("Index"); //Sildikten sonra Index Actiona git...

}

**19.Bölüm Asp.Net Core Identity**

Identity Paketlerinin Yüklenmesi

Admin paneline giriş yapılmadan erişilmemesi gerekir dolayısıyla AdminController üstüne [Authorize] yazılır.

Microsoft.AspNetCore.Authentication.Cookies 2.1.0

Microsoft.AspNetCore.Identity.EntityFrameworkCore 2.1.1

User ve Role Sınıflarının Oluşturulması

using Microsoft.AspNetCore.Identity;

namespace Abc.WebUI.Entities

{

public class CustomIdentityUser : IdentityUser

{

//kendi özellikleri burada yazılır. Kullanıcının birimi gibi...

}

}

&&&

using Microsoft.AspNetCore.Identity;

namespace Abc.WebUI.Entities

{

public class CustomIdentityRole : IdentityRole

{

//kendi role özellikler burada oluşturulur...

}

}

IdentityDbContext Sınıfının Oluşturulması

namespace Abc.WebUI.Entities

{

public class CustomIdentityDbContext : IdentityDbContext<CustomIdentityUser,CustomIdentityRole,string>

{

public CustomIdentityDbContext(DbContextOptions<CustomIdentityDbContext> options):base(options)

{

}

}

}

Startup Dosyasında Yapılandırma

#region Identity

services.AddDbContext<CustomIdentityDbContext>(option => option.UseSqlServer(@"Data Source=DESKTOP-2TS36OR\SQLSERVER2017EXP;Initial Catalog=Northwind;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"));

services.AddIdentity<CustomIdentityUser, CustomIdentityRole>().AddEntityFrameworkStores<CustomIdentityDbContext>().AddDefaultTokenProviders();

#endregion

&&&

App.UseIdentity();

ViewModellerin Oluşturulması

namespace Abc.WebUI.ViewModels

{

public class LoginViewModel

{

[Required]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

}

&&&

namespace Abc.WebUI.ViewModels

{

public class RegisterViewModel

{

[Required]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[DataType(DataType.Password)]

public string ConfirmPassword { get; set; }

[Required]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

}

}

AccounControllerın Yazılması

namespace Abc.WebUI.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<CustomIdentityUser> \_userManager;

private readonly RoleManager<CustomIdentityRole> \_roleManager;

private readonly SignInManager<CustomIdentityUser> \_signInManager;

public AccountController(UserManager<CustomIdentityUser> userManager,RoleManager<CustomIdentityRole> roleManager,SignInManager<CustomIdentityUser> signInManager)

{

\_userManager = userManager;

\_roleManager = roleManager;

\_signInManager = signInManager;

}

//form oluşturcak

public ActionResult Register()

{

return View();

}

//form dolduracak

[HttpPost]

[ValidateAntiForgeryToken] //güvenlik açısından

public ActionResult Register(RegisterViewModel registerViewModel)

{

if (ModelState.IsValid)

{

CustomIdentityUser user = new CustomIdentityUser

{

UserName = registerViewModel.UserName,

Email = registerViewModel.Email

};

IdentityResult result =

\_userManager.CreateAsync(user, registerViewModel.Password).Result;

if (result.Succeeded)

{

if (!\_roleManager.RoleExistsAsync("Admin").Result)

{

CustomIdentityRole role = new CustomIdentityRole

{

Name = "Admin"

};

IdentityResult roleResult = \_roleManager.CreateAsync(role).Result;

if (!roleResult.Succeeded)

{

ModelState.AddModelError("", "We can't add the role!");

return View(registerViewModel);

}

}

\_userManager.AddToRoleAsync(user, "Admin").Wait();

return RedirectToAction("Login", "Account");

}

}

return View(registerViewModel);

}

public ActionResult Login()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginViewModel loginViewModel)

{

if (ModelState.IsValid)

{

var result = \_signInManager.PasswordSignInAsync(loginViewModel.UserName,

loginViewModel.Password, loginViewModel.RememberMe, false).Result;

if (result.Succeeded)

{

return RedirectToAction("Index", "Admin");

}

ModelState.AddModelError("", "Invalid login!");

}

return View(loginViewModel);

}

//[HttpPost]

//[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

\_signInManager.SignOutAsync().Wait();

return RedirectToAction("Login");

}

**Login ve Register Sayfalarının Oluşturulması**

Account / Login

@using System.Threading.Tasks @\*Asenkron hatalarını önlemek için\*@

@model LoginViewModel

@{

ViewBag.Title = "title";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<div class="container">

<div id="loginbox" style="margin-top:50px;" class="mainbox col-md-6">

<div class="panel panel-info">

<div class="panel-heading">

<div class="panel-title">Sign In</div>

</div>

<div style="padding-top:30px" class="panel-body">

<div style="display:none" id="login-alert" class="alert alert-danger col-sm-12"></div>

<**form** **asp-controller**="Account" **asp-action**="Login" method="post" class="form-horizontal" role="form">

<div style="margin-bottom: 25px" class="input-group">

<span class="input-group-addon"><i class="glyphicon glyphicon-user"></i></span>

<**input** **asp-for**="UserName" class="form-control" placeholder="username">

<**span** **asp-validation-for**="UserName"></**span**>

</div>

<div style="margin-bottom: 25px" class="input-group">

<span class="input-group-addon"><i class="glyphicon glyphicon-lock"></i></span>

<**input** **asp-for**="Password" class="form-control" placeholder="password">

<**span** **asp-validation-for**="Password"></**span**>

</div>

<div class="input-group">

<div class="checkbox">

<label>

<**input** **asp-for**="RememberMe"> Remember me

</label>

</div>

</div>

<div style="margin-top:10px" class="form-group">

<!-- Button -->

<div class="col-sm-12 controls">

<input type="submit" class="btn btn-success" value="Login">

</div>

</div>

<div class="form-group">

<div class="col-md-12 control">

<div style="border-top: 1px solid#888; padding-top:15px; font-size:85%">

Don't have an account!

<**a** **asp-controller**="Account" **asp-action**="Register">

Sign Up Here

</**a**>

</div>

</div>

</div>

</**form**>

</div>

</div>

</div>

</div>

Account / Register

@using System.Threading.Tasks

@model RegisterViewModel

@{

ViewBag.Title = "title";

Layout = "~/Views/Shared/\_AdminLayout.cshtml";

}

<div class="container">

<div id="signupbox" style="margin-top:50px;" class="mainbox col-md-6">

<div class="panel panel-info">

<div class="panel-heading">

<div class="panel-title">Sign Up</div>

<div style="float:right; font-size: 85%; position: relative; top:-10px"><**a** **asp-controller**="Account" **asp-action**="Login">Sign In</**a**></div>

</div>

<div class="panel-body">

<**form** **asp-controller**="Account" **asp-action**="Register" method="post" class="form-horizontal" role="form">

<div class="form-group">

<**label** **asp-for**="UserName" class="col-md-3 control-label"></**label**>

<div class="col-md-9">

<**input** **asp-for**="UserName" **type**="text" class="form-control" placeholder="User Name">

<**span** **asp-validation-for**="UserName"></**span**>

</div>

</div>

<div class="form-group">

<**label** **asp-for**="Password" class="col-md-3 control-label"></**label**>

<div class="col-md-9">

<**input** **asp-for**="Password" class="form-control" placeholder="Password">

<**span** **asp-validation-for**="Password"></**span**>

</div>

</div>

<div class="form-group">

<**label** **asp-for**="Email" class="col-md-3 control-label"></**label**>

<div class="col-md-9">

<**input** **asp-for**="Email" class="form-control" placeholder="Email Address">

<**span** **asp-validation-for**="Email"></**span**>

</div>

</div>

<div class="form-group">

<!-- Button -->

<div class="col-md-offset-3 col-md-9">

<input type="submit" class="btn btn-info" value="Register"><i class="icon-hand-right"></i>

</div>

</div>

</**form**>

</div>

</div>

</div>

</div>

Migration

WebUI paketi ekle.

Microsoft.EntityFrameworkCore.Tools 2.1.1

PM> Add-Migration Identity

PM> Update-Database

UserSummary ViewComponent Yapılışı

namespace Abc.WebUI.ViewComponents

{

public class UserSummaryViewComponent : ViewComponent

{

public ViewViewComponentResult Invoke()

{

UserDetailsViewModel model = new UserDetailsViewModel

{

UserName = HttpContext.User.Identity.Name //Bu alanı tutar.

};

return View(model);

}

}

}

&&&

namespace Abc.WebUI.ViewModels

{

public class UserDetailsViewModel

{

public string UserName { get; set; }

}

}

&&&

&&&

Views/Shared/Components/SummaryComponent/Default.cshtml

@model UserDetailsViewModel

<ul class="dropdown-menu">

@if (!String.IsNullOrEmpty(Model.UserName))

{

<li>

<a>Welcome @Model.UserName</a>

</li>

<li>

<**a** **asp-controller**="Account" **asp-action**="LogOff">Log Off</**a**>

</li>

}

</ul>

&&&

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav">

<li class="active"><**a** **asp-controller**="Product" **asp-action**="Index">Products Store <span class="sr-only">(current)</span></**a**></li>

</ul>

<ul class="nav navbar-nav navbar-right">

<li class="dropdown">

<a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">User <span class="caret"></span></a>

@await Component.InvokeAsync("UserSummary")

</li>

</ul>

</div>

LoggOff çalışması içinde

//[HttpPost]

//[ValidateAntiForgeryToken] //LogOff Çalışması için

public ActionResult LogOff()

{

\_signInManager.SignOutAsync().Wait();

return RedirectToAction("Login");

}

}