

T.C.
KOCAELİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

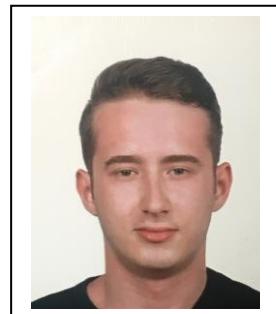


STAJ RAPORU

DÖNEM:6

YIL :3

T.C.
KOCAELİ ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ
BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ MÜHENDİSLİĞİ BÖLÜMÜ



Öğrencinin Kimlik Bilgileri

T.C. Kimlik No:25933413314	Doğum Tarihi:04.10.1997
Adı- Soyadı:Hamdi Uraz Alkış	Bölümü:Bilişim Sistemleri Mühendisliği
Baba Adı:Ömer	Fakülteye Giriş Yılı:2016
Ana Adı:Güzin	Sınıfı:4
Doğum Yeri:İnegöl	Fakülte No:1307
	S.G.K. No:1601200839786

Öğrencinin Staj Süresince Bulunduğu İkametgâh ve İletişim Bilgileri

Görevi: Stajyer Öğrenci	Ülke:Türkiye
İli:Bursa	İlçe:İnegöl
Açık Adres:Bursa/İnegöl Süleymaniye mah.yücel sk safir park sitesi a blok kat2 daire 12	
E-Posta:alkisuraz@gmail.com	
Cep Telefonu:05457698820	Ev Telefonu:0224 715 56 34

Staj Yapılan Kurum Bilgileri

Kurumun Adı:Kocaeli Üniversitesi	Telefon No:+90 (262) 303 2402
Çalışma Alanı:Bilişim Teknolojileri	Faks No: +90 (262) 303 2403
Adresi: Kocaeli Üniversitesi Eğitim Fakültesi Umuttepe Yerleşkesi 41380 Kocaeli	E-posta Adresi:egitim@kocaeli.edu.tr
Haftalık Çalışma Gün Sayısı: 5 İşgünü	Web Adresi: http://www.kocaeli.edu.tr

Çalışmayı Onaylayan Kurum Yetkilisinin

Adı SOYADI: Unvanı: E-Posta: Telefon No:	İmza/Kaşesi:
---	---------------------

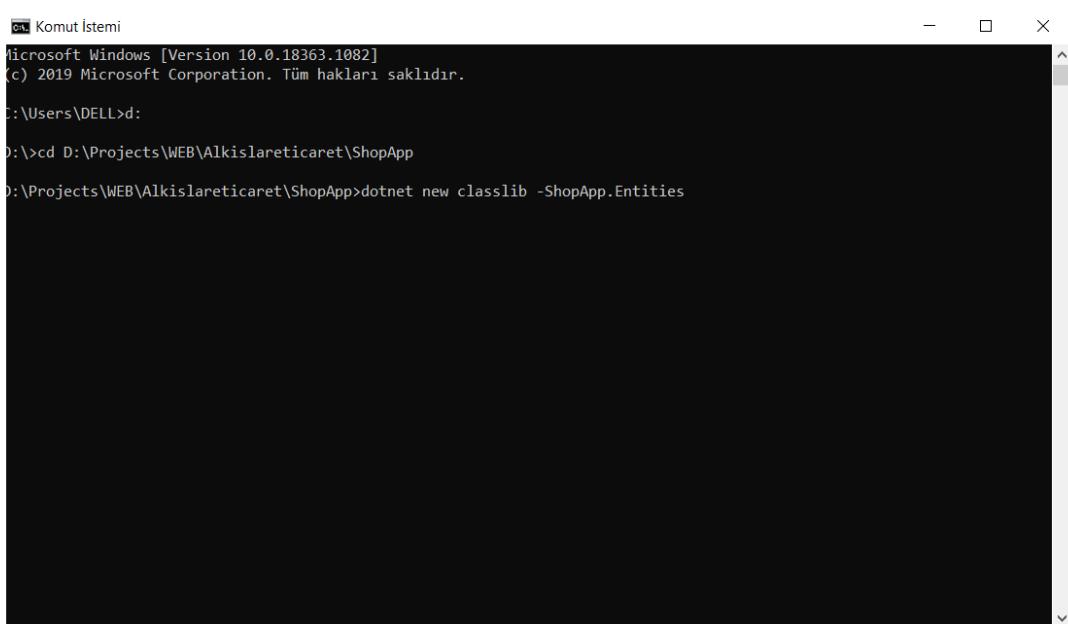
06/07/2020 Tarihinden 10/07/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
06/07/2020	Pazartesi	Proje Ortamının Oluşturulması	1/1	9
07/07/2020	Salı	Katmanlı Mimari Yapısını Anlama	2/2	9
08/07/2020	Çarşamba	Entity Katmanını Ekleme	3/3	9
09/07/2020	Perşembe	Repository Pattern Yapısını Anlama	4/4	9
10/07/2020	Cuma	Repository Pattern Oluşturma	5/5	9
..../..../....	Cumartesi		.../...	
TOPLAM SÜRE (Saat)				45
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış İmzası: Hamdi Uraz Alkış Çalıştığı Bölüm: Bilişim Teknolojileri		Adı SOYADI: Unvanı: İmza/Kaşesi:		

Not: Gerektiği kadar sayfa numarası vererek çoğaltınız.

Yapılan İş: Proje Ortamının Oluşturulması

Tarih:06/07/2020

Açıklama:Proje Asp.net core mvc ve katmanlı mimari yapısıyla geliştirileceği için,katmanlı mimarinin sahip olduğu entities,dataAccess,business ve WebUI katmanları için sınıflar oluşturuldu.

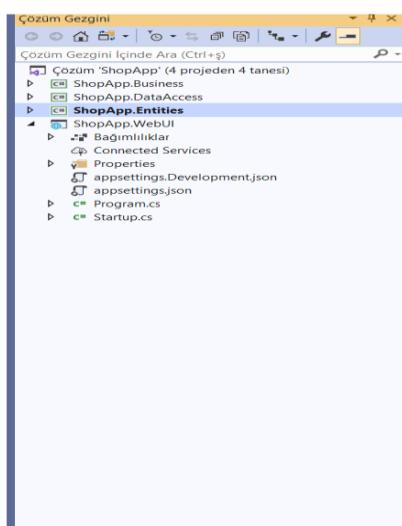


```
microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\DELL>d:
D:>cd D:\Projects\WEB\Alkislareticaret\ShopApp

D:\Projects\WEB\Alkislareticaret\ShopApp>dotnet new classlib -ShopApp.Entities
```

Her bir işlem diğer katmanlar içinde gerçekleştirildi.WebUI katmanında farklı olarak classlib komutu değilde,web komutuyla WebUI katmanı oluşturuldu.



1

ÖĞRENCİNİN:

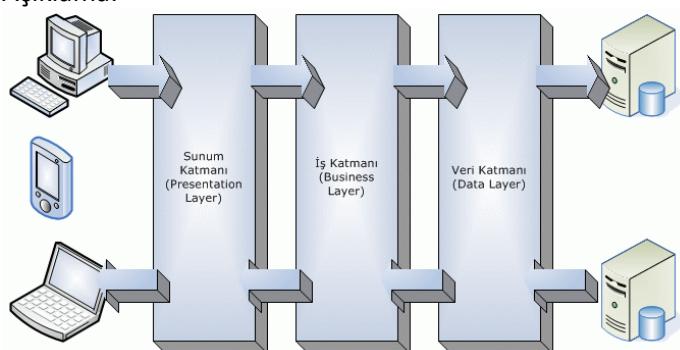
İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:



Katmanlı mimari yapısı bize eğer büyük çaplı projeler içerisinde bulunuyorsak,kod karmaşasını önleyerek daha anlaşılır bir kod düzeni oturtmamızı sağlar.Katmanlı mimari yapısı temel olarak 3 katmandan oluşur.Veri katmanında,veri tabanıyla ilgili bilgiler tutulur.Bu katmanın görevi veri ekleme,güncelleme,silme ve veri tabanından veri çekmekle ilgilidir.

İş katmanında,iş yüklerimizi azaltırız.Uygulamalarda öncelik olarak Data Access katmanı kullanılmaz.Kullanıcıdan gelen bilgiler ilk olarak iş katmanına gider daha sonra veri katmanına aktarılır.Veritabanından verileri hangi sorguya alacağımızı,verileri nasıl okuyacağımızı,verileri okuduktan sonra ne yapacağımızı gibi olayları bu katmanda yapıyor.Kısaca veri tabanıyla projemiz arasında bir köprü vazifesi görüyor.

Sunum katmanı ise bizim projemiz web projesi olacağı için,webUI katmanı olacak.Burada kullanıcının projeye etkileşime geçtiği noktalar barındırılacak.

Bizim projemizde bu 3 katmana ek olarak entities katmanı da yer alıyor.Burada proje boyunca kullanacağımız ana classlarımızı belirliyoruz.

ÖĞRENCİNİN:**İMZASI:** Hamdi Uraz Alkış**KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace ShopApp.Entities
6  {
7      public class Product
8      {
9          public int Id { get; set; }
10         public string Name { get; set; }
11         public string ImageUrl { get; set; }
12         public decimal Price { get; set; }
13
14         public List<ProductCategory> ProductCategories { get; set; }
15     }
16 }
17

```

Şekilde görüldüğü gibi, entities katmanına bir Product sınıfı oluşturduk. Burada her ürünün id, ismi, resim urlsi, fiyatı bulunuyor. Ek olarak sonradan ekleyeceğimi Category sınıfıyla ilişki tablosu için, oluşturacağımız ProductCategory listesini de dahil ediyoruz.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace ShopApp.Entities
6  {
7      public class Category
8      {
9          public int Id { get; set; }
10         public string Name { get; set; }
11
12         public List<ProductCategory> ProductCategories { get; set; }
13     }
14 }
15

```

Aynı şekilde Category sınıfımızı oluşturduk.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace ShopApp.Entities
6  {
7      public class ProductCategory
8      {
9          public int CategoryId { get; set; }
10         public Category Category { get; set; }
11
12         public int ProductId { get; set; }
13         public Product Product { get; set; }
14     }
15 }
16

```

İki tablo arasında n-n ilişki olduğu için bunu belirtmek adına ortak bir sınıf oluşturuyoruz

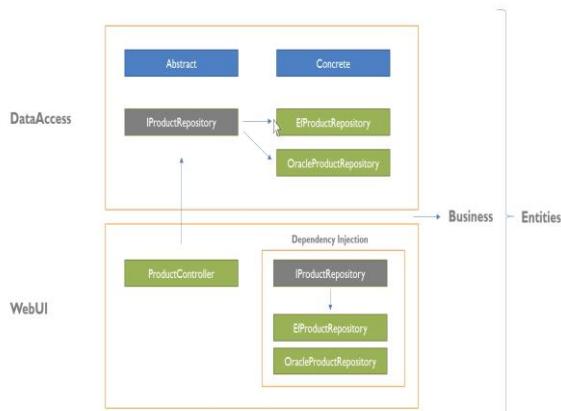
**ÖĞRENCİNİN:
İMZASI: Hamdi Uraz Alkış**

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:



Yazılım uygulamaları yoğun bir şekilde veritabanı işlemleri gerçekleştiren yapılar olduğundan dolayı, ilgili uygulamanın her bir noktasında gerekli veritabanı işlemlerini tekrar tekrar yazmak yerine bu işlemleri tekrar kullanabilmek adına, daha pratik bir şekilde tek seferde yapmamızı sağlayan bir yapılanma geliştirmemiz gerekecektir. İşte bu yapılanma Repository sınıfı olacak.

Bunun için DataAccess katmanına Abstract ve Concrete adında iki klasör oluşturacağız. Abstract sanal sınıfları yani interfaceleri tutarken Concrete bu interface'nin doldurulmuş hallerini tutacak.

Şekilde görüldüğü gibi, Abstract klasörüne, product entity ile ilişişime geçecek bir IProductRepository interface'i oluşturacağız. Concrete klasöründe ise, bu interface'nin implemente edilmiş sınıflarını ekleyeceğiz.

Açıklama: DataAccess katmanında oluşturduğumuz Abstract klasörüne IProductDal adında bir interface oluşturduk.

```

ShopApp.DataAccess
4  using System.Linq;
5  using System.Linq.Expressions;
6  using System.Text;
7
8  namespace ShopApp.DataAccess.Abstract
9  {
10     public interface IProductDal
11     {
12         Product GetById(int id);
13         Product GetOne(Expression<Func<Product, bool>> filter);
14         IQueryble<Product> GetAll(Expression<Func<Product, bool>> filter);
15
16         void Create(Product entity);
17         void Update(Product entity);
18         void Delete(Product entity);
19     }
20 }
21
22

```

Entity katmanından Product sınıfıyla çalışacağımız için using ShopApp.Entities olarak referans ekledik. Kullanıcı dışarıdan bir id gönderdiği zaman, Product tablosundan bunu bulup geri göndermek için GetById adında bir product tanımladık. Aynı şekilde geri dönüş değeri olmayan create, update, delete isminde dışarıdan product alan yapılar oluşturduk.

```

using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
namespace ShopApp.DataAccess.Concrete.EfCore
{
    public class EfCoreProductDal : IProductDal
    {
        public void Create(Product entity)
        {
            throw new NotImplementedException();
        }

        public void Delete(Product entity)
        {
            throw new NotImplementedException();
        }

        public IQueryble<Product> GetAll(Expression<Func<Product, bool>> filter)
        {
            throw new NotImplementedException();
        }

        public Product GetById(int id)
        {
            throw new NotImplementedException();
        }
    }
}

```

Concrete dosyasına, entityframework ile çalışıldığı için EfCoreProductDal adında bir sınıf oluşturduk. Bunu da IProductDal sınıfından türettik. Zaten IProductDal sınıfının methodları, implement işlemi yaptığımda otomatik olarak geliyor.

13/07/2020 Tarihinden 17/07/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
13/07/2020	Pazartesi	Repository Pattern Oluşturma	6/6	9
14/07/2020	Salı	Repository Pattern Oluşturma	7/8	9
15/07/2020	Çarşamba	RESMİ TATİL	/	-
16/07/2020	Perşembe	Business Katmanı Oluşturma	8/9	9
17/07/2020	Cuma	Database Oluşturma	10/11	9
.../.../...	Cumartesi		.../...	
TOPLAM SÜRE (Saat)				36
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış İmzası: Hamdi Uraz Alkış Çalıştığı Bölüm: Bilişim Teknolojileri		Adı SOYADI: Unvanı: İmza/Kaşesi:		

Yapılan İş: Repository Pattern Yapısı Oluşturma

Tarih: 14/07/2020

Açıklama:

```

Repository.cs
-----
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Linq.Expressions;
5 using System.Text;
6
7 namespace ShopApp.DataAccess.Abstract
8 {
9     public interface IRepository<T>
10    {
11        T GetById(int id);
12        T GetOne(Expression

```

Abstract klasörüne işlem tekrarını azaltmak için IRepository adında bir interface oluşturduk.IRepository generic bir tipte olacak.Böylece ICategoryDal ve IProductDal IRepository içindeki bütün methodlara sahip olacak.

```

ProductDal.cs
-----
1 using ShopApp.Entities;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Linq.Expressions;
6 using System.Text;
7
8 namespace ShopApp.DataAccess.Abstract
9 {
10    public interface IProductDal : IRepository<Product>
11    {
12        IEnumerable<Product> GetPopularProducts();
13    }
14 }

```

Şekildeki gibi IProductDal: IRepository<Product> diyerek,IProductDal'ın IRepository deki bütün methodlara sahip olmasını sağlıyoruz.Bunun aynısını ICategoryDal,IOrderDal içinde gerçekleştirdik.

```

ShopContext.cs
-----
1 using Microsoft.EntityFrameworkCore;
2 using ShopApp.Entities;
3 using System;
4 using System.Collections.Generic;
5 using System.Text;
6
7 namespace ShopApp.DataAccess.Concrete.EfCore
8 {
9     public class ShopContext : DbContext
10    {
11        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
12        {
13            optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=ShopDb;Integrated security=true");
14        }
15        public DbSet<Product> Products { get; set; }
16        public DbSet<Category> Categories { get; set; }
17    }
18 }

```

Database adresi için bir shopContext class'ını Concrete/efcore içine ekledik.ShopContext DbContext sınıfından türetilmiş olacak.Onun için bunu DataAccess kısmında Nuget paketlerine eklememiz gerekiyor

6

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Repository Pattern Yapısı Oluşturma

Tarih: 14/07/2020

Açıklama:



```
ShopApp.DataAccess.csproj
1 <Project Sdk="Microsoft.NET.Sdk">
2   ...
3     <PropertyGroup>
4       <TargetFramework>netcoreapp2.2</TargetFramework>
5     </PropertyGroup>
6   ...
7   <ItemGroup>
8     <Folder Include="Concrete\Oracle" />
9   </ItemGroup>
10  ...
11  <ItemGroup>
12    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="2.1.4" />
13    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="2.1.4" />
14  </ItemGroup>
15  ...
16  <ItemGroup>
17    <ProjectReference Include="..\ShopApp.Entities\ShopApp.Entities.csproj" />
18  </ItemGroup>
19  ...
20 </Project>
21
```

DbContext entityFrameworkCore yapısının içinde olduğu için dataAccess entitityFrameworkCore'yi nuget paketlerinden yükliyoruz.



```
ShopContexts.cs
1 using Microsoft.EntityFrameworkCore;
2 using ShopApp.Entities;
3 using System;
4 using System.Collections.Generic;
5 using System.Linq;
6
7 namespace ShopApp.DataAccess.Concrete.EfCore
8 {
9   ...
10  public class ShopContext : DbContext
11  {
12    ...
13    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
14    {
15      optionsBuilder.UseSqlServer($"Server=[localdb]\MSSQLLocalDB;Database=ShopDb;integrated security=true;");
16    }
17    ...
18  }
19}
```

Database olarak sql server kullanacağımız için, optionsBuilder.UseSqlServer bölümünü aktif etmek adına,yine nuget paketlerinden sqlserver'i indiriyoruz ve tırnak içindeki belirtilen yere database adresini yazıyoruz.

Açıklama:

```

1  using ShopApp.Entities;
2  using System;
3  using System.Collections.Generic;
4  using System.Text;
5
6  namespace ShopApp.Business.Abstract
7  {
8      public interface IProductService
9      {
10         void GetByProductId(int id);
11         List<Product> GetAll();
12         void Create(Product entity);
13         void Update(Product entity);
14         void Delete(Product entity);
15     }
16 }

```

DataAccess katmanında olduğu gibi, Abstract ve Concrete klasörlerini ekliyoruz. Yine aynı şekilde Abstract klasöründe sanal sınıflar yani interfaceler, Concrete içinde ise bunun doldurulmuş halleri olacak.

Abstract klasörü içine IProductService isminde bir interface oluşturuyoruz. Bütün ürün listesini çeviren, product tipinde bir liste tanımlıyoruz. Yine diğerlerinde olduğu gibi create update Delete methodlarını tanımlıyoruz. Bu tanımlamaların aynalarını yine ICategory service içinde yapıyoruz.

```

1  using ShopApp.Business.Abstract;
2  using ShopApp.DataAccess.Abstract;
3  using ShopApp.DataAccess.Concrete.EFCore;
4  using ShopApp.Entities;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9
10 namespace ShopApp.Business.Concrete
11 {
12     public class ProductManager : IProductService
13     {
14         private IProductDal _productDal;
15
16         public ProductManager(IProductDal productDal)
17         {
18             _productDal = productDal;
19         }
20
21         public void Create(Product entity)
22         {
23             _productDal.Create(entity);
24         }
25
26         public void Delete(Product entity)
27     }
28 }

```

Concrete bölümünde, IProductService'i doldurmak için bir ProductManager sınıfı oluşturuyoruz. Bu sınıfı IProductService'den ctrl.implement diyerek implemente işlemi yapıyoruz ve IProductService içindeki methodlarımız ekleniyor.

8

ÖĞRENCİNİN:**İMZASI: Hamdi Uraz Alkış****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:

Oluşturduğumuz ProductManager sınıfı, WebUI da örneğin bir controller içinde ulaşılacak ve ProductManager üzerinden örneğin getAll dediğimizde bütün ürünler databaseden alması gerekiyor.

The screenshot shows the Visual Studio IDE. On the left, there are two open files: 'ProductManager.cs' and 'IPrductService.cs'. The 'ProductManager.cs' file contains C# code for a concrete implementation of a product manager. It includes namespaces for ShopApp.Business.Abstract, ShopApp.DataAccess.Abstract, ShopApp.DataAccess.Concrete.EfCore, ShopApp.Entities, System, System.Collections.Generic, System.Linq, and System.Text. The class 'ProductManager' implements the interface 'IPrductService'. It has a private field '_productDal' of type IProductDal. A constructor takes an IProductDal parameter and initializes the field. There are also methods for creating and deleting products. On the right, the 'Çözüm Gezgini' (Solution Explorer) is visible, showing the project structure for 'Çözüm ShopApp' which contains four projects: ShopApp.Business, ShopApp.DataAccess, ShopApp.Entities, and ShopApp.WebUI.

```

ProductManager.cs
1  using ShopApp.Business.Abstract;
2  using ShopApp.DataAccess.Abstract;
3  using ShopApp.DataAccess.Concrete.EfCore;
4  using ShopApp.Entities;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9
10 namespace ShopApp.Business.Concrete
11 {
12     public class ProductManager : IPrductService
13     {
14         private IProductDal _productDal;
15
16         public ProductManager(IProductDal productDal)
17         {
18             _productDal = productDal;
19         }
20
21         public void Create(Product entity)
22         {
23             _productDal.Create(entity);
24         }
25
26         public void Delete(Product entity)
27     }
}

```

Burada efCoreProductDal'ı kullanmak yerine, IProductDal'ı kullandığımız zaman zaten efCoreProductDal'a da hakimiyet sağlayacağımız için IProductDal üzerinden işlem yapıyoruz. Eğer efCoreProductDal üzerinden işlem yapsaydık, ilerleyen günlerde efCoreProductDal üzerinde çalışmama kararı alsaydık, efCoreProductDal ProductManager'e bağımlı olduğu için, katmanlı mimarının bir anlamı kalmayacaktı. Çünkü birisini değiştirdiğimizde diğer kod bütünlüğümüzde bundan etkilenecekti.

Yapılan İş: Database Oluşturma

Tarih: 17/07/2020

Açıklama:

The screenshot shows the Visual Studio IDE. On the left, there is a code editor window titled 'ShopContext.cs' with the following C# code:

```
ShopContext.cs
1  using Microsoft.EntityFrameworkCore;
2  using ShopApp.Entities;
3  using System;
4  using System.Collections.Generic;
5  using System.Text;
6
7  namespace ShopApp.DataAccess.Concrete.EfCore
8  {
9      public class ShopContext : DbContext
10     {
11         protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
12         {
13             optionsBuilder.UseSqlServer($"Server=(localdb)\MSSQLLocalDB;Database=ShopDB;integrated security=true");
14         }
15
16         protected override void OnModelCreating(ModelBuilder modelBuilder)
17         {
18             modelBuilder.Entity<ProductCategory>()
19                 .HasKey(c => new { c.CategoryId, c.ProductId });
20         }
21
22         public DbSet<Product> Products { get; set; }
23         public DbSet<Category> Categories { get; set; }
24     }
}
```

On the right, the 'Solution Explorer' window is open, showing the project structure:

- Çözüm Gezgini (Solution Explorer)
 - Çözüm ShopApp (4 projeden 4 tanesi)
 - ShopApp.Business
 - ShopApp.DataAccess
 - Abstract
 - ICategoryDals
 - IOrderDals
 - IProductDals
 - IRepository.cs
 - Concrete
 - EfCore
 - EFCoreCategoryDals.cs
 - EFCoreGenericRepository.cs
 - EFCoreOrderDals.cs
 - EFCoreProductDals.cs
 - ShopContexts
 - Memory
 - MySQL
 - Oracle
 - Migrations
 - ShopApp.Entities
 - ShopApp.WebUI

Database oluşturmadan önce ProductCategory tablosundaki CategoryID ve ProductID'i entity tablosunun bir birincil anahtarı olarak atamamız gerekiyor. Bunun için shopContext içinde fluent Api yazılması gerekiyor. Fluent api basitçe veri tabanı sınıflarını ve ilişkilerini yapılandırabilmenin bir yoludur.

Burada oneModelCreating altına modelBuilder.Entity<ProductCategory>()

.HasKey(c => new { c.CategoryId, c.ProductId });

Diyerek, hasKey methoduyla buna 2 adet birincil anahtar atadık.

Şimdi DataAccess'te olan shopContext için birtane migration oluşturuyoruz. Kod kısmında yaptığımız değişiklikleri veritabanına yansıtma işlemine Migration deniyor.

The screenshot shows a terminal window with the following command history:

```
Komut İstemi
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\DELL>d:
D:>cd D:\Projects\WEB\Alkışlareticaret\ShopApp\ShopApp.DataAccess
D:\Projects\WEB\Alkışlareticaret\ShopApp\ShopApp.DataAccess>dotnet ef migrations add CreateDatabase
```

Diyerek migration oluşturuyoruz.

10

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

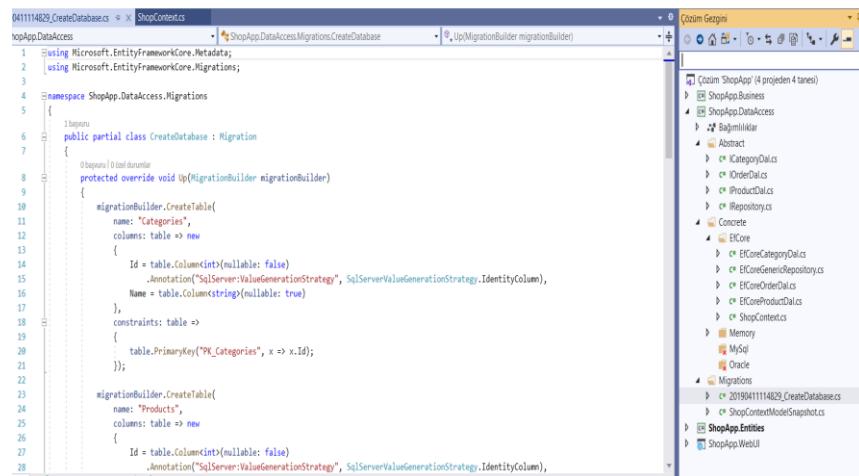
ADI VE SOYADI:

İMZASI:

Yapılan İş: Database Oluşturma

Tarih: 17/07/2020

Açıklama:



The screenshot shows the Visual Studio IDE with the code editor open. The file is named '04111114829_CreateDatabase.cs' located in the 'ShopApp.DataAccess' project under the 'Migrations' folder. The code defines a partial class 'CreateDatabase' that inherits from 'Migration'. It contains two 'CreateTable' methods for 'Categories' and 'Products' tables, each with a primary key column 'Id' annotated with 'IdentityColumn'. The solution explorer on the right shows other projects like 'ShopApp.Business' and 'ShopApp.DataAccess' along with their respective files.

```
04111114829_CreateDatabase.cs
ShopApp.DataAccess
ShopApp.DataAccess.Migrations.CreateDatabase
Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Categories",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
            Name = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Categories", x => x.Id);
        });
    migrationBuilder.CreateTable(
        name: "Products",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGenerationStrategy.IdentityColumn),
            Name = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Products", x => x.Id);
        });
}
```

Şekilde görüldüğü gibi Migrations klasörüne CreateDatabase adında bir migrations oluşturuldu. Böylece bahsedilen iki alanda primary key olarak işaretlendi. Aynı şekilde komut satırına dotnet ef database update dendiğinde oluşturulan migration uygulanır ve oluşturduğumuz database tablolarımız database'e aktarılmış olur.

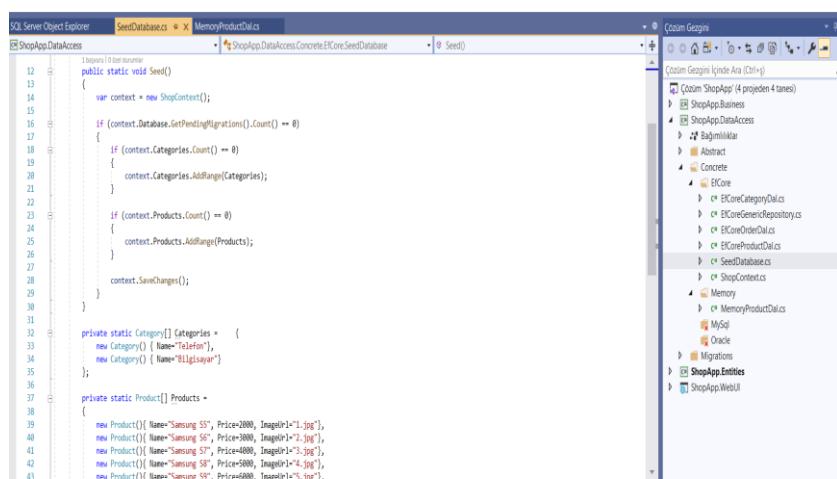
20/07/2020 Tarihinden 24/07/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
20/07/2020	Pazartesi	Test Verileri Ekleme	12/12	9
21/07/2020	Salı	Ürünleri Listeleme	13/14	9
22/07/2020	Çarşamba	Site Tasarımına Giriş	15/17	9
23/07/2020	Perşembe	Ürünlerin Görüntülenmesi	18/18	9
24/07/2020	Cuma	Ürün Detay Sayfası	19/19	9
.../.../...	Cumartesi		.../...	
TOPLAM SÜRE (Saat)				45
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış		Adı SOYADI:		
İmzası: Hamdi Uraz Alkış		Unvanı:		
Çalıştığı Bölüm: Bilişim Teknolojileri		İmza/Kaşesi:		

Yapılan İş: Test Verileri Ekleme

Tarih: 20/07/2020

Açıklama:

Bu işlem dataAccess'in efcore alanına ait olduğu için, efcore klasörüne SeedDatabase adında bir sınıf oluşturalım.



```

public static void Seed()
{
    var context = new ShopContext();

    if (context.Database.GetPendingMigrations().Count() == 0)
    {
        if (context.Categories.Count() == 0)
        {
            context.Categories.AddRange(categories);
        }

        if (context.Products.Count() == 0)
        {
            context.Products.AddRange(products);
        }
    }

    context.SaveChanges();
}

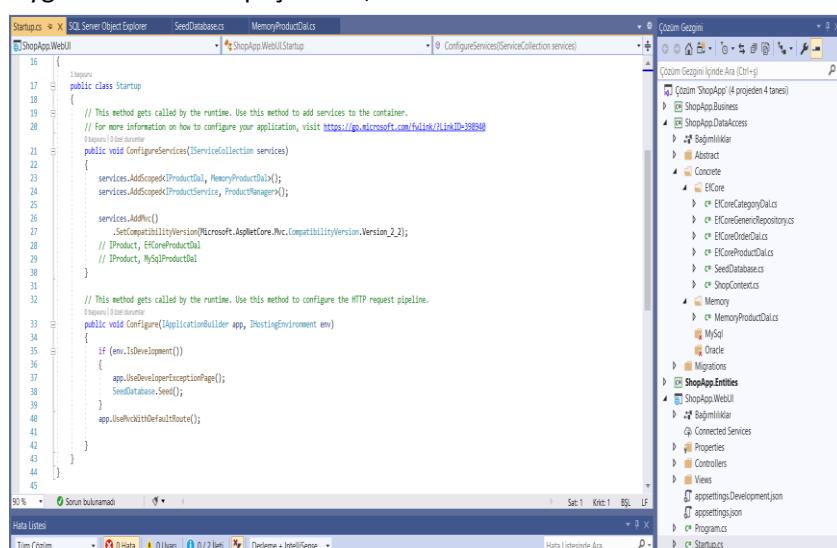
private static Category[] Categories =
{
    new Category() { Name="Telefon" },
    new Category() { Name="Bilgisayar" };
};

private static Product[] Products =
{
    new Product() { Name="Samsung S7", Price=2000, ImageUrl="1.jpg" },
    new Product() { Name="Samsung S7", Price=3000, ImageUrl="2.jpg" },
    new Product() { Name="Samsung S7", Price=4000, ImageUrl="3.jpg" },
    new Product() { Name="Samsung S7", Price=5000, ImageUrl="4.jpg" },
    new Product() { Name="Samsung S7", Price=6000, ImageUrl="5.jpg" },
    new Product() { Name="Samsung S7", Price=7000, ImageUrl="6.jpg" }
};

```

Burada görüldüğü gibi getPendingMigration methoduyla, bekleyen migration olup olmadığını öğrenebiliyoruz. Eğer migration sayısı sıfırsa, test verilerini database'e aktar kodunu yazıyoruz.

Örnek açısından Product dizisine birkaç tane veri ekledik. Bunu uygulamaya tanıtmak için Uygulamanın startup içeresine;



```

public class Startup
{
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398969
    IHostEnvironment env;

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddScoped(typeof(Product), typeof(MemoryProductDal));
        services.AddScoped(typeof(ProductService), typeof(ProductManager));

        services.AddEfCore();
        .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_2);
        // Product, EfCoreProduct
        // Product, MySqlProduct
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            SeedDatabase.Seed();
        }
        app.UseHttpsRedirection();
    }
}

```

Env.IsDevelopment bölümünde SeedDatabase.Seed(); kodunu ekliyoruz.

Category ve Productları kontrol ettiğimizde verilerin tablolara eklediğini görürüz.

12

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

```

1  using ShopApp.Entities;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Threading.Tasks;
6
7  namespace ShopApp.WebUI.Models
8  {
9      public class ProductListModel
10     {
11         public List<Product> Products { get; set; }
12     }
13 }
14

```

Açıklama:

Öncelikle WebUI katmanına bir Models klasörü oluşturduk ve bu klasörün altına ProductListModel adında bir sınıf tanımladık. Bu sınıfın içi elbette daha çok genişletilebilir fakat şu anlık sadece Liste tipinde productları alacağımız bir yapı oluşturduk.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6  using ShopApp.Business.Abstract;
7  using ShopApp.WebUI.Models;
8
9  namespace ShopApp.WebUI.Controllers
10 {
11     [ApiController]
12     [Route("api/[controller]")]
13     public class HomeController : Controller
14     {
15         private IProductService _ productService;
16         public HomeController(IProductService productService)
17         {
18             _ productService = productService;
19         }
20
21         [HttpGet("[action]")]
22         public IActionResult Index()
23         {
24             return View(new ProductListModel()
25             {
26                 Products = _ productService.GetPopularProducts()
27             });
28         }
29     }
30 }

```

Şimdi önceden oluşturduğumuz HomeController altındaki IActionResult methodundaki _productService.getAll methodunu silip yerine new productListModel diyerek productListModel'i belirtelim.

```

1  using ShopApp.Entities;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace ShopApp.Business.Abstract
6  {
7      public interface IProductService
8      {
9          Product GetById(int id);
10         List<Product> GetAll();
11         List<Product> GetPopularProducts();
12         void Create(Product entity);
13         void Update(Product entity);
14         void Delete(Product entity);
15     }
16 }

```

Burada bütün ürün bilgilerini getirmek yerine, farklı olaylarda yapabildiğimizi görebilmek adına, IProductService sınıfında GetAll methodu gibi yine liste tipinde bir GetPopularProduct yapısı oluşturduk. Böylece HomeController altında bunu çağrılabiliyoruz.

13

ÖĞRENCİNİN:**İMZASI: Hamdi Uraz Alkış****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:

The screenshot shows the Visual Studio IDE interface. On the left, the code editor displays the `ProductManager.cs` file under the `ShopApp.Business` project. The code implements the `IProductManager` interface with methods for deleting, getting all products, getting a product by ID, getting popular products, and updating a product. On the right, the Solution Explorer shows the `productMana` solution containing the `ShopApp.Business` project, which includes the `Concrete` folder and the `ProductManager.cs` file.

```
ProductManager.cs
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using ShopApp.DataAccess;
6  using ShopApp.Business.Concrete;
7  using ShopApp.Business.IBusiness;
8
9  namespace ShopApp.Business
10 {
11     public class ProductManager : IProductManager
12     {
13         private readonly IProductDal _productDal;
14
15         public ProductManager(IProductDal productDal)
16         {
17             _productDal = productDal;
18         }
19
20         public void Delete(Product entity)
21         {
22             _productDal.Delete(entity);
23         }
24
25         [Bölgeye | Dözel durumlar]
26         public List<Product> GetAll()
27         {
28             return _productDal.GetAll();
29         }
30
31         [Bölgeye | Dözel durumlar]
32         public Product GetById(int id)
33         {
34             return _productDal.GetById(id);
35         }
36
37         [Bölgeye | Dözel durumlar]
38         public List<Product> GetPopularProducts()
39         {
40             return _productDal.GetAll();
41         }
42
43         [Bölgeye | Dözel durumlar]
44         public void Update(Product entity)
45         {
46             _productDal.Update(entity);
47         }
48
49     }
50 }
51
52 }
```

Göründüğü üzere, oluşturduğumuz `GetPopularProduct` yapısı, `ProductManager` Sınıfına da gelmiş oldu.

Açıklama:

Site tasarımına geçmeden önce, kullanıcı sayfaları için ShopController, yönetici sayfaları içinse AdminController'i WebUI katmanındaki Controller klasörüne ekliyoruz. Views klasörünün içine de _Layout klasörü ekliyoruz. Shared klasöründe, kod tekrarına düşmemek adına her sayfada olacak navbar, footer, layout gibi alanları ekleyeceğiz.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6
7  namespace ShopApp.WebUI.Controllers
8  {
9      public class AdminController : Controller
10     {
11         public IActionResult Index()
12         {
13             return View();
14         }
15     }
16 }

```

Öncelikle bootstrap yardımıyla basit bir navbar oluşturalım.

```

1 <nav class="navbar-expand-lg navbar-dark bg-primary fixed-top">
2     <div class="container">
3         <a class="navbar-brand" href="#"> Shopping
4         <ul class="navbar-nav mr-auto">
5             <li class="nav-item">
6                 <a href="#" class="nav-link active">Home</a>
7             </li>
8             <li class="nav-item">
9                 <a href="#" class="nav-link">Products</a>
10            </li>
11            <li class="nav-item">
12                <a href="#" class="nav-link">Cart</a>
13            </li>
14            <li class="nav-item">
15                <a href="#" class="nav-link">Orders</a>
16            </li>
17            <li class="nav-item">
18                <a href="#" class="nav-link">Add Product</a>
19            </li>
20            <li class="nav-item">
21                <a href="#" class="nav-link">Admin Products</a>
22            </li>
23            <li class="nav-item">
24                <a href="#" class="nav-link">Add Category</a>
25            </li>
26            <li class="nav-item">
27                <a href="#" class="nav-link">Admin Categories</a>
28            </li>
29        </ul>
30
31        <ul class="navbar-nav mr-auto">
32            <li class="nav-item">
33                <a href="#" class="nav-link">Logout</a>
34            </li>
35        </ul>
36    </div>
37 </nav>

```

Açıklama:

```

1 <!DOCTYPE html>
2
3 <html>
4   <head>
5     <meta name="viewport" content="width=device-width" />
6     <link href="/modules/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet" />
7     <link href="/css/style.css" rel="stylesheet" />
8     <title>ShopApp</title>
9   </head>
10  <body>
11    <partial name="_navbar" />
12
13    <div class="container pt-100 mb-5">
14      <@RenderBody()>
15    </div>
16
17
18    <script src="/modules/jquery/dist/jquery.min.js"></script>
19    <script src="https://cdnjs.cloudflare.com/ajax/libs/cooper.js/1.14.7/umd/cooper_min.js"></script>
20    <script src="/modules/bootstrap/dist/js/bootstrap.min.js"></script>
21  </body>
22 </html>

```

Layout sayfası ana gövdeyi oluşturacağı için, oluşturduğumuz _navbar sayfاسını, partial name ile buraya import ediyoruz. @RenderBody() bunu işlememizi sağlıyor. Partial name tag helper'inin aktif olabilmesi için View klasörüne Razor View Start ve Razor View Import'u ekliyoruz.

_ViewImports'a @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers'ı ekliyoruz. Böylece partial name tag helper'i aktif oldu.

Css dosyalarımızı yazmak için, wwwroot klasörünün içine bir css dosyası oluşturuyoruz ve içine style.css ekliyoruz.

```

1 <model>ShopApp.WebUI.Models.ProductListModel</model>
2
3
4 <div class="row">
5   <div class="col-md-3">
6     <ul class="list-group">
7       <li class="list-group-item">Telefon</li>
8       <li class="list-group-item">Bilgisayar</li>
9       <li class="list-group-item">Elektronik</li>
10    </ul>
11  </div>
12  <div class="col-md-9">
13    <partial name="_slider" />
14
15    <foreach var item in Model.Products>
16    {
17      <p>@item.Name</p>
18    }
19
20  </div>
21 </div>
22 </div>
23
24
25

```

Index.cshtml sayfasına da kategorilerimizi belli eden bir liste yapısı oluşturduk.

16

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

```

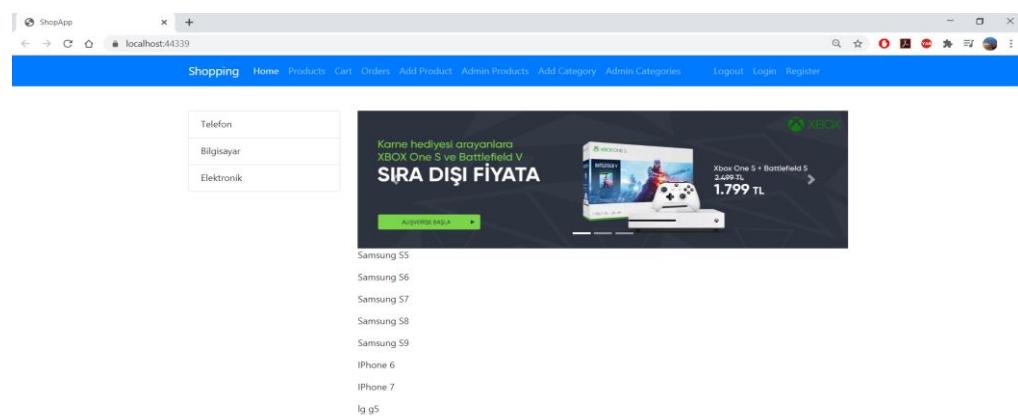
1 <div id="mySlider" class="carousel slide" data-ride="carousel">
2   <ol class="carousel-indicators">
3     <li data-target="#mySlider" data-slide-to="0" class="active"></li>
4     <li data-target="#mySlider" data-slide-to="1"></li>
5     <li data-target="#mySlider" data-slide-to="2"></li>
6   </ol>
7
8   <div class="carousel-inner">
9     <div class="carousel-item active">
10       
11     </div>
12     <div class="carousel-item">
13       
14     </div>
15     <div class="carousel-item">
16       
17     </div>
18   </div>
19
20   <a href="#mySlider" data-slide="prev" class="carousel-control-prev">
21     <span class="carousel-control-prev-icon"></span>
22   </a>
23
24   <a href="#mySlider" data-slide="next" class="carousel-control-next">
25     <span class="carousel-control-next-icon"></span>
26   </a>
27
28 </div>
29
30

```

The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `_slider.cshtml` file with the provided HTML and CSS code. On the right, the Solution Explorer shows the project structure for "Çözüm ShopApp" with files like `Index.cshtml`, `ViewStart.cshtml`, `_ViewImports.cshtml`, `_Layout.cshtml`, `_navbar.cshtml`, and `AdminController.cs`. The status bar at the bottom indicates 0 errors and 0 warnings.

Siteye ayrıca basit bir slider ekledim.

Projeyi çalıştırduğumızda;

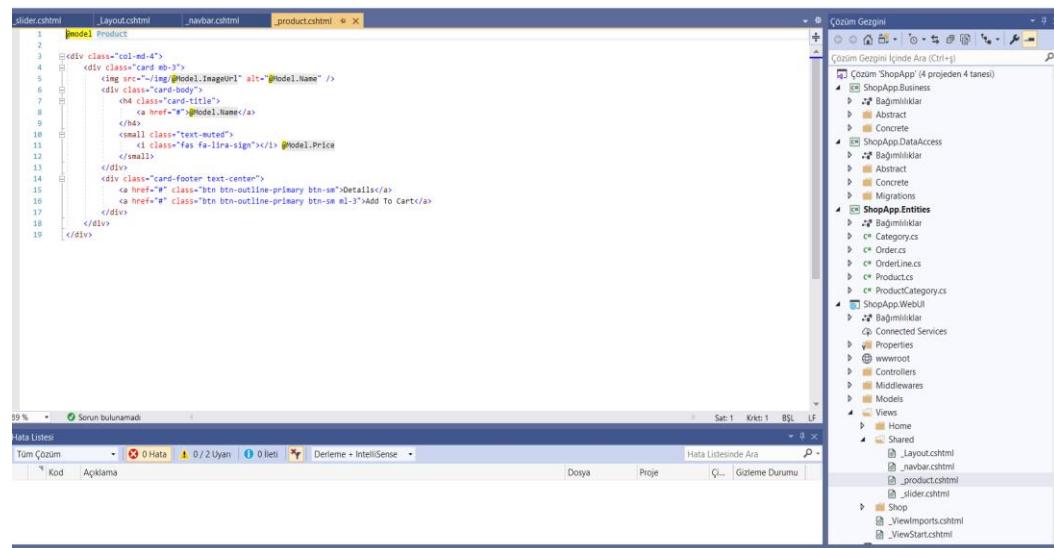


Göründüğü gibi bir görünüm elde ettim.

Yapılan İş: Ürünlerin Görüntülenmesi

Tarih: 23/07/2020

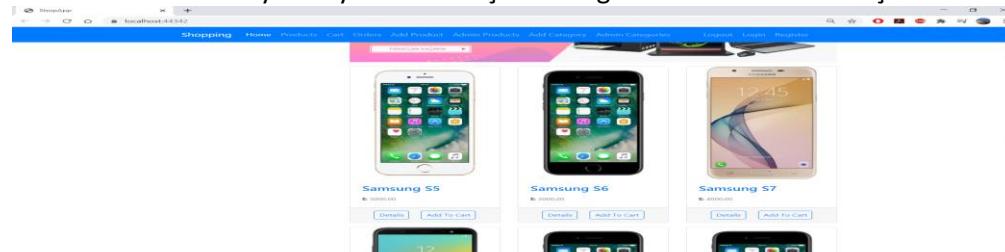
Açıklama: Gösterilecek ürünlerin kart yapısını oluşturmamız gereklidir. Bunun için öncelikle shared klasörüne _product adında bir view ekledik.



Ürünlere erişeceğimiz için, sayfanın bir modele sahip olması gerekiyor. Bu modelde Product model olacak. Bu yüzden @Model Product ile model import işlemini gerçekleştiriyoruz fakat bunu sağlayabilmek için, viewimport dosyasında,



Şekildeki gibi @using ShopApp.Entities katmanını import etmemiz gereklidir. Çünkü Product sayfamız bu katmanda bulunuyor. Böylece alttaki şekildeki gibi ürünlerimizi eklemiştir.



18

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `ShopController.cs` with the following code:

```

12     public class ShopController : Controller
13     {
14         private IProductService _productService;
15         protected override void OnActionExecuting(ActionExecutingContext context)
16         {
17             if (_productService == null)
18             {
19                 throw new NullReferenceException("Product service is null");
20             }
21             context.ModelState.AddModelError("Product", "Product is null");
22         }
23         [HttpGet]
24         public IActionResult Details(int? id)
25         {
26             if (id == null)
27             {
28                 return NotFound();
29             }
30             var product = _productService.GetById((int)id);
31             if (product == null)
32             {
33                 return NotFound();
34             }
35             return View(product);
36         }
37         [HttpGet]
38         public IActionResult List()
39         {
40             return View(new ProductListModel())
41             {
42                 Products = _productService.GetAll()
43             };
44         }
45     }
    
```

The Solution Explorer on the right shows the project structure for `shop`, including `ShopContext.cs`, `Migrations`, `ShopContextModelSnapshot.cs`, `ShopContextModelSnapshot`, `ShopApp.Entities`, `ShopApp.WebUI`, and `Bağımlilikler`.

ShopController altındaki public IActionResult methodunu, details olarak isim değişikliği yaptık. Ve parametre olarak id parametresini ekledik. int yanında soru işaretri olmasının nedeni, değerin null olarak da atanabilmesini sağlamak. Bize null kontrolü yapmamızda yardımcı olacak. Eğer id null ise bizi not found helper methoduyla, kullanıcıyı 404 sayfasına yönlendiriyor.

Aynı şekilde product için de aynı kontrolü yaptık. Eğer ürün bulunmuşsa, return view methoduyla ekrana ürünleri yazdırıyoruz. Şimdi detay sayfası için bir görünüm oluşturacağız.

The screenshot shows the Visual Studio IDE. On the left, the code editor displays `Details.cshtml` with the following code:

```

1 <@model ShopApp.Models.Product>
2 
3 <@{
4     ViewData["Title"] = "Details";
5 }>
6 
7 <div class="row">
8     <div class="col-md-3">
9         " class="img-fluid" />
10    </div>
11    <div class="col-md-9">
12        <h1><@Model.Name@>
13        <p><@Model.Description@>
14        <hr/>
15        <a href="#" class="btn btn-link p-0 mb-3">Telefon</a>
16        <div class="mb-3">
17            <div class="text-primary mb-3">
18                <@Model.Price@>
19            </div>
20            <button type="submit" class="btn btn-primary btn-lg">Add To Cart</button>
21        </div>
22    </div>
23 </div>
24 
25 <div class="row">
26     <div class="col-md-12">
27         <p class="p-3">
28             <img alt="Samsung S5" style="width: 100px; height: auto; margin-right: 10px;"/>
29             Samsung S5
30             <br/>
31             İde: 3000,00 TL
32             <button type="button" class="btn btn-primary">Add To Cart
33         </p>
34     </div>
35 </div>
    
```

The Solution Explorer on the right shows the project structure for `details`, including `Controllers`, `ShopController.cs`, and `Details.cshtml`.

Şekilde görüldüğü gibi, details linkine tıkladığımızda basit bir detay sayfası ekrana geliyor.

27/07/2020 Tarihinden 29/07/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
27/07/2020	Pazartesi	Kategori Menüsü Oluşturma	20/22	9
28/07/2020	Salı	Kategori Linklerini Ekleme	23/23	9
29/07/2020	Çarşamba	Admin İle Ürün Oluşturma	24/25	9
30/07/2020	Perşembe	RESMİ TATİL	/	-
31/07/2020	Cuma	RESMİ TATİL	/	-
.../.../...	Cumartesi		.../...	
TOPLAM SÜRE (Saat)				27
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış		Adı SOYADI:		
İmzası: Hamdi Uraz Alkış		Unvanı:		
Çalıştığı Bölüm: Bilişim Teknolojileri		İmza/Kaşesi:		

Yapılan İş: Kategori Menüsü Oluşturma

Tarih:27/07/2020

Açıklama:Öncelikle WebUI katmanına ViewComponent adında bir klasör oluşturuyoruz.Bunun içine de CategoryListView adında bir class oluşturuyoruz.

```

CategoryListViewComponent.cs
1 using Microsoft.AspNetCore.Mvc;
2 using ShopApp.Business.Abstract;
3 using ShopApp.WebUI.Models;
4 using System;
5 using System.Collections.Generic;
6 using System.Linq;
7 using System.Threading.Tasks;
8
9 namespace ShopApp.WebUI.ViewComponents
10 {
11     [ViewComponent]
12     public class CategoryListViewComponent : ViewComponent
13     {
14         private ICategoryService _categoryService;
15         public CategoryListViewComponent(ICategoryService categoryService)
16         {
17             _categoryService = categoryService;
18         }
19         public IViewComponentResult Invoke()
20         {
21             return View(new CategoryListModel());
22             Categories = _categoryService.GetAll();
23         }
24     }
25 }
26
27

```

The screenshot shows the Visual Studio interface. On the left, the code editor displays the `CategoryListViewComponent.cs` file. On the right, the Solution Explorer shows the project structure for `ShopApp`, including `ShopApp.Business`, `ShopApp.DataAccess`, `ShopApp.Entities`, and `ShopApp.WebUI`. The `ShopApp.WebUI` folder contains `ViewComponents`, `Views`, and configuration files like `appsettings.Development.json`.

Göründüğü üzere oluşturduğumuz sınıfı, ViewComponent sınıfından türettik. ViewComponent aslında partial View'lere alternatif olarak oluşturulmuş bir şey. Server tarafına daha az yük bindirdiği için daha tercih edilemişim ben de yeni yeni öğreniyorum:) Invoke methodu sabit bir method. İçerisine return view ile bir view göndereceğiz. Bu view'in içinde bir kategori listesi olması gerekiyor. Bunun için ICategoryService sınıfındaki getAll methodunu kullanacağız. Bu yüzden Concrete versiyonunu hazırlamak gerekiyor. Concrete Klasörüne CategoryManager adında bir sınıf oluşturuyoruz.

```

ProductManager.cs
1 using ShopApp.Business.Abstract;
2 using ShopApp.DataAccess.Abstract;
3 using ShopApp.DataAccess.Concrete.EFCore;
4 using ShopApp.Entities;
5 using System;
6 using System.Collections.Generic;
7 using System.Linq;
8 using System.Text;
9
10 namespace ShopApp.Business.Concrete
11 {
12     [Serializable]
13     public class ProductManager : IProductService
14     {
15         private IProductDal _productDal;
16
17         public ProductManager(IProductDal productDal)
18         {
19             _productDal = productDal;
20         }
21
22         [Serializable]
23         public void Create(Product entity)
24         {
25             _productDal.Create(entity);
26         }
27
28         [Serializable]
29         public void Delete(Product entity)
30         {
31             _productDal.Delete(entity);
32         }
33     }
34 }
35
36

```

The screenshot shows the Visual Studio interface. On the left, the code editor displays the `ProductManager.cs` file. On the right, the Solution Explorer shows the project structure for `ShopApp`, including `ShopApp.Business`, `ShopApp.DataAccess`, `ShopApp.Entities`, and `ShopApp.WebUI`. The `ShopApp.Business` folder contains `Abstract`, `Concrete`, and `CategoryManager.cs`. The `ShopApp.DataAccess` folder contains `Abstract`, `Concrete`, and `Migrations`. The `ShopApp.Entities` folder contains `Model`. The `ShopApp.WebUI` folder contains `ViewComponents`, `Views`, and configuration files like `appsettings.Development.json`.

ICategoryService'den implement işlemi yaptığımızda methodlarımız geliyor.

Yapılan İş: Kategori Menüsü Oluşturma

Tarih: 27/07/2020

Açıklama:

The screenshot shows the Visual Studio IDE. In the center, the code editor displays the `Startup.cs` file for the `ShopApp.WebUI` project. The code defines a `ConfigureServices` method that registers services like `IProductDal`, `ICategoryDal`, and `ICategoryService`. The `Configure` method sets up the HTTP request pipeline. The Solution Explorer on the right shows the project structure, including `ShopApp.Business`, `ShopApp.DataAccess`, `ShopApp.Entities`, and `ShopApp.WebUI`.

```
Startup.cs
using ShopApp.Business.Concrete;
using ShopApp.DataAccess.Abstract;
using ShopApp.DataAccess.Concrete.EFCore;
using ShopApp.DataAccess.Concrete.Memory;
using ShopApp.WebUI.Middlewares;
namespace ShopApp.WebUI
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?linkid=3089846
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddScoped<IProductDal, EfCoreProductDal>();
            services.AddScoped<ICategoryDal, EfCoreCategoryDal>();
            services.AddScoped<ICategoryService, CategoryManager>();
            services.AddScoped<CategoryService, CategoryManager>();

            services.AddMvc()
                .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_2);
            // IProduct, EfCoreProductDal
            // IProduct, MySqlProductDal
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseCookiePolicy();
            app.UseMiddleware<RequestLocalizationMiddleware>();
            app.UseRouting();
            app.UseAuthorization();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

Projenin startup dosyasında bunu diğer servisler üzerinde de yaptığımız gibi `services.AddScoped<ICategoryDal, EfCoreCategoryDal>();` diyerek `ICategoryDal` çağrırlığında `EfCoreCategoryDal`'ı gönderelim. Aynı şekilde `IProductService` çağrırlığında da `ProductManager`'ı, `ICategoryService` çağrırlığında `CategoryManager`'ı gönderelim.

The screenshot shows the Visual Studio IDE. In the center, the code editor displays the `CategoryListViewComponent.cs` file for the `ShopApp.WebUI` project. The component uses `ICategoryService` to get all categories and render them in a view. The Solution Explorer on the right shows the project structure, including `ShopApp.Business`, `ShopApp.DataAccess`, `ShopApp.Entities`, and `ShopApp.WebUI`.

```
CategoryListViewComponent.cs
using Microsoft.AspNetCore.Mvc;
using ShopApp.Business.Abstract;
using ShopApp.WebUI.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
namespace ShopApp.WebUI.ViewComponents
{
    public class CategoryListViewComponent : ViewComponent
    {
        private ICategoryService _categoryService;
        public CategoryListViewComponent(ICategoryService categoryService)
        {
            _categoryService = categoryService;
        }
        public IViewComponentResult Invoke()
        {
            return View(new CategoryListViewModel()
            {
                Categories = _categoryService.GetAll()
            });
        }
    }
}
```

Daha önceden oluşturduğumuz `CategoryListViewComponent` içerisinde de `Invoke` methodu içerisine `_categoryService.GetAll()` diyerek kategorileri getirelim. Tabii ki bunun view içerisinde de bir karşılığı olması gerekiyor. Bunu shared klasörü içerisinde ekleyeceğimiz component içerisindeki `CategoryList` klasörüne `Default.cshtml` isminde görünüm dosyası oluşturarak yapalım.

Yapılan İş: Kategori Menüsü Oluşturma

Tarih:27/07/2020

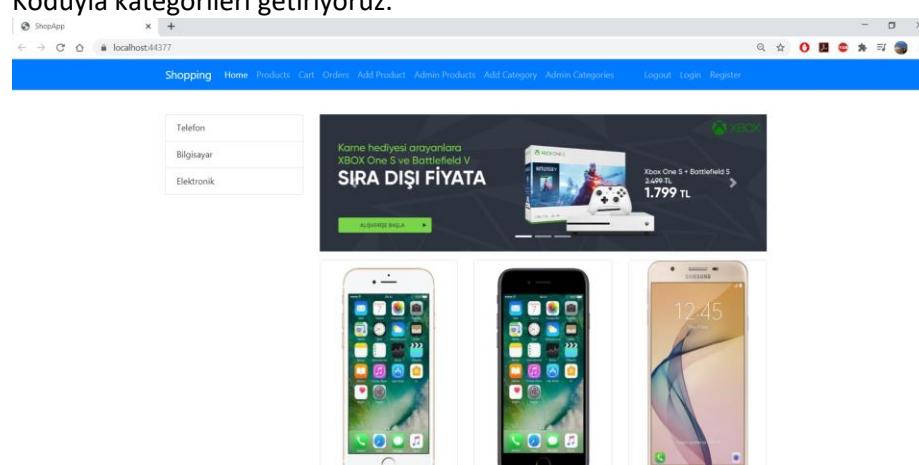
Açıklama:

```
Default.cshtml
1<ul class="list-group">
2    @foreach (var item in Model.Categories)
3    {
4        <li class="list-group-item">
5            @item.Name
6        </li>
7    }
8</ul>
```

Göründüğü gibi Default.cshtml görünümünde

```
@foreach (var item in Model.Categories)
{
    <li class="list-group-item">
        @item.Name
    </li>
}
```

Koduyla kategorileri getiriyoruz.



Göründüğü gibi kategorilerimiz sol tarafta görünür durumda.

22

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Kategori Linklerini Ekleme

Tarih: 28/07/2020

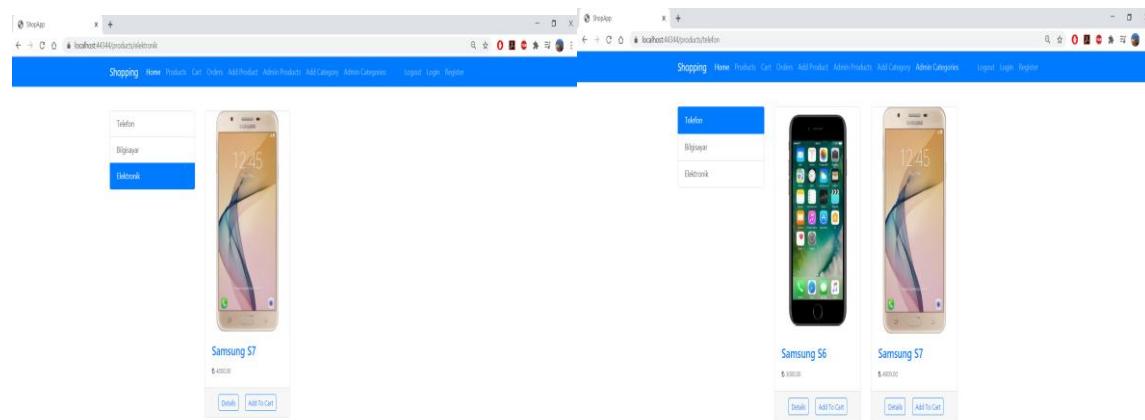
Açıklama:

The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `CategoryListViewModel.cs` file with the following code:

```
1 <div class="list-group">
2   <asp:foreach var="item" in Model.Categories>
3     <a href="#" asp-controller="Shop" asp-action="List" asp-route-category="@item.Name.ToLower()"
4       class="list-group-item list-group-item-action @(Model.SelectedCategory==item.Name.ToLower()?"active":"")">
5       @item.Name
6     </a>
7   </asp:foreach>
8 </div>
```

The Solution Explorer on the right shows the project structure for `ShopApp`, including `Entities`, `WebUI`, `Models`, `Views`, and `Shop` folders.

Default görünüm dosyasında,kategorinin ismini route olarak gönderiyoruz.



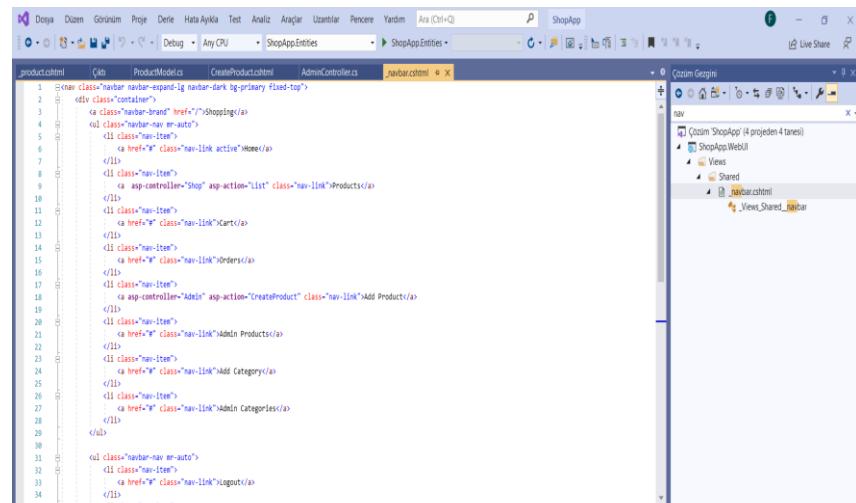
Göründüğü gibi her farklı kategoriye tıklandığında, önceden ayarladığımız farklı kategorideki ürünler ekrana geliyor.

Yapılan İş: Admin ile ürün oluşturma

Tarih: 29/07/2020

Açıklama:

İlk önce _navbar üzerindeki linkleri aktifleştirelim.

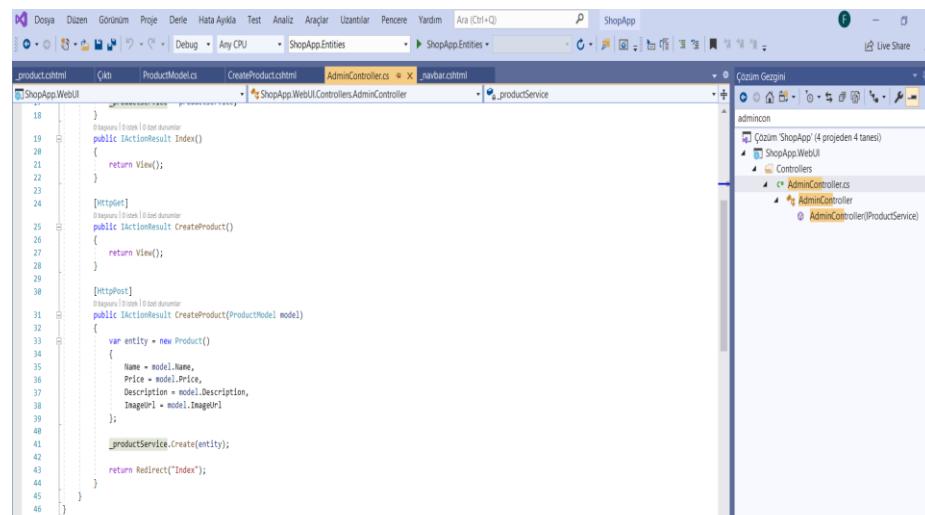


```

<nav class="navbar navbar-expand-lg navbar-dark bg-primary fixed-top">
  <div class="container">
    <a class="navbar-brand" href="/">Shopping</a>
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a href="#" class="nav-link active" href="#">Home</a>
      </li>
      <li class="nav-item">
        <a href="#" asp-controller="Shop" asp-action="List" class="nav-link">Products</a>
      </li>
      <li class="nav-item">
        <a href="#" class="nav-link">Cart</a>
      </li>
      <li class="nav-item">
        <a href="#" class="nav-link">Orders</a>
      </li>
      <li class="nav-item">
        <a href="#" asp-controller="Admin" asp-action="CreateProduct" class="nav-link">Add Product</a>
      </li>
      <li class="nav-item">
        <a href="#" class="nav-link">Admin Products</a>
      </li>
      <li class="nav-item">
        <a href="#" class="nav-link">Add Category</a>
      </li>
      <li class="nav-item">
        <a href="#" class="nav-link">Admin Categories</a>
      </li>
    </ul>
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a href="#" class="nav-link">Input</a>
      </li>
    </ul>
  </div>

```

<a asp-controller="Admin" asp-action="CreateProduct" class="nav-link">Add Product
Koduyla bir action oluşturduk.



```

public class AdminController : Controller
{
    [HttpGet]
    public IActionResult Index()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult CreateProduct(ProductModel model)
    {
        var entity = new Product()
        {
            Name = model.Name,
            Price = model.Price,
            Description = model.Description,
            ImageUrl = model.ImageUrl
        };

        productService.Create(entity);

        return RedirectToAction("Index");
    }
}

```

Bu işlemler adminController üzerinden yapılacağı için, burada CreateProduct adında bir method oluşturduk. Birinci CreateProduct bize formu, ikinci CreateProduct methodu ise işin post tarafı. Yukarıda da httppost olarak belirtiyoruz. Diğerini de httpget olarak belirledik.

Sımdı Admin sayfası için admin klasörünün altına CreateProduct view'i ekliyoruz.

24

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Admin ile Ürün Oluşturma

Tarih: 29/07/2020

Açıklama:

The screenshot shows the Visual Studio IDE. In the center, the code editor displays the `CreateProduct.cshtml` file, which contains an ASP.NET form for creating a product. The form includes fields for Name, ImageUrl, Description, and Price. The right side of the interface shows the Solution Explorer, which lists four projects: `ShopApp.Business`, `ShopApp.DataAccess`, `ShopApp.Entities`, and `ShopApp.WebUI`. The `ShopApp.WebUI` project is expanded, showing its structure including controllers like `AdminController.cs`, `HomeController.cs`, and `ShopController.cs`, and models like `CategoryListViewModel.cs` and `ProductModel.cs`.

```
product.cshtml ProductModel.cs CreateProduct.cshtml AdminController.cs _navbar.cshtml
1 <body>
2     <div>ProductModel</div>
3     <div> ViewData["Title"] = "CreateProduct";</div>
4 
5     <div><h1>Create Product</h1></div>
6     <div><form asp-controller="Admin" asp-action="CreateProduct" method="post">
7         <div class="form-group row">
8             <label asp-for="Name" class="col-md-2 col-form-label"></label>
9             <div class="col-md-10">
10                 <input asp-for="Name" value="" class="form-control" />
11             </div>
12         </div>
13         <div class="form-group row">
14             <label asp-for="ImageUrl" class="col-md-2 col-form-label"></label>
15             <div class="col-md-10">
16                 <input asp-for="ImageUrl" value="" class="form-control" />
17             </div>
18         </div>
19         <div class="form-group row">
20             <label asp-for="Description" class="col-md-2 col-form-label"></label>
21             <div class="col-md-10">
22                 <textarea id="editor" asp-for="Description" class="form-control" />
23             </div>
24         </div>
25         <div class="form-group row">
26             <label asp-for="Price" class="col-md-2 col-form-label"></label>
27             <div class="col-md-10">
28                 <input asp-for="Price" value="" class="form-control" />
29             </div>
30         </div>
31         <div class="form-group row">
32             <div class="col-10 offset-md-2">
33                 <button type="submit" class="btn btn-primary">Save Product</button>
34             </div>
35         </div>
36     </form>
37 </div>
38 </div>
```

Göründüğü gibi sayfayı hazırladık.Controllerimiz Admin,actionumuz ise CreateProduct olarak belirledik.

The screenshot shows a web browser window with the URL `localhost:44354/Admin/CreateProduct`. The page title is "Create Product". It contains four input fields: "Name" (iPhone XR), "ImageUrl" (3.jpg), "Description" (iPhone telefon), and "Price" (7000). Below the form is a blue "Save Product" button.

The screenshot shows the SQL Server Object Explorer with the database `dbo.Products` selected. A table named `Products` is displayed with columns `Id`, `Name`, `Description`, `ImageUrl`, and `Price`. The table contains several rows of data, including the newly added iPhone XR entry.

	<code>Id</code>	<code>Name</code>	<code>Description</code>	<code>ImageUrl</code>	<code>Price</code>
1	1	Samsung S5	şırıgazlı telefon	1.jpg	3000.00
2	2	Samsung S6	şırıgazlı telefon	2.jpg	4000.00
3	3	Samsung S7	şırıgazlı telefon	3.jpg	4500.00
4	4	Samsung S8	şırıgazlı telefon..	4.jpg	5000.00
5	5	Samsung S9	şırıgazlı telefon..	5.jpg	6000.00
6	6	iPhone 6	şırıgazlı telefon..	6.jpg	4000.00
7	7	iPhone 7	şırıgazlı telefon..	7.jpg	5000.00
8	8	iPhone 8	şırıgazlı telefon..	8.jpg	5500.00
9	9	iPhone XR	iPhone telefon	9.jpg	7000.00
	NULL	NULL	NULL	NULL	NULL

Uygulamayı çalıştırduğumızda şekildeki gibi eklediğimiz son ürün database'e geldi.

25

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

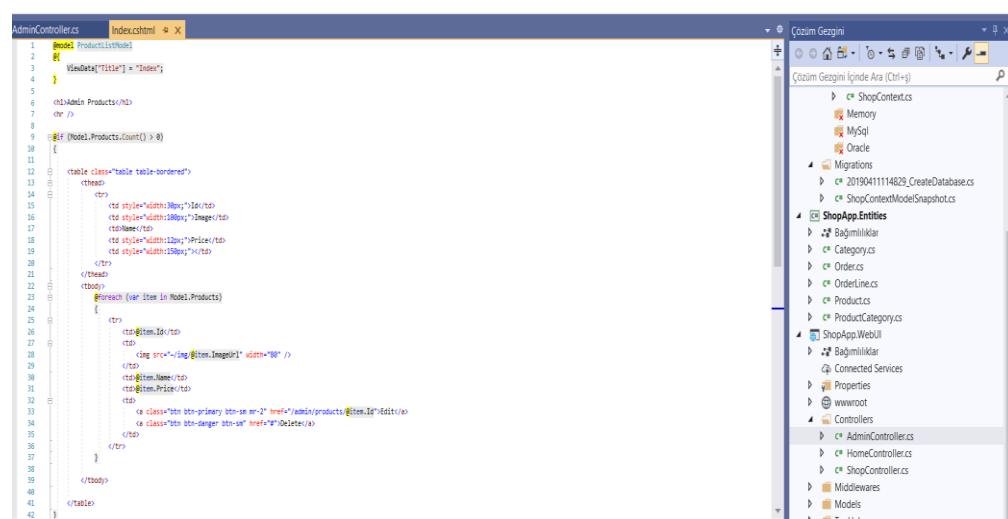
04/08/2020 Tarihinden 07/08/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
03/08/2020	Pazartesi	RESMİ TATİL	/	-
04/08/2020	Salı	Ürünleri Listeleme	26/26	9
05/08/2020	Çarşamba	Ürün Düzenleme	27/28	9
06/08/2020	Perşembe	Ürün Silme İşlemi	29/29	9
07/08/2020	Cuma	Üyelik İşlemleri Identity Yapısı	30/31	9
.../.../...	Cumartesi		.../....	
TOPLAM SÜRE (Saat)				36
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış İmzası: Hamdi Uraz Alkış Çalıştığı Bölüm: Bilişim Teknolojileri		Adı SOYADI: Unvanı: İmza/Kaşesi:		

Açıklama:

Öncelikle AdminController sayfasında public ActionResult Index()

```
{
    return View(new ProductListModel())
{
    Products = _productService.GetAll()
});
}
```

Koduyla ürünleri sayfaya gönderme işlemi yapıyoruz. Sonra bu index sayfası için bir view oluşturuyoruz.



The screenshot shows the Visual Studio IDE. On the left, the code editor displays AdminController.cs with the following code:

```

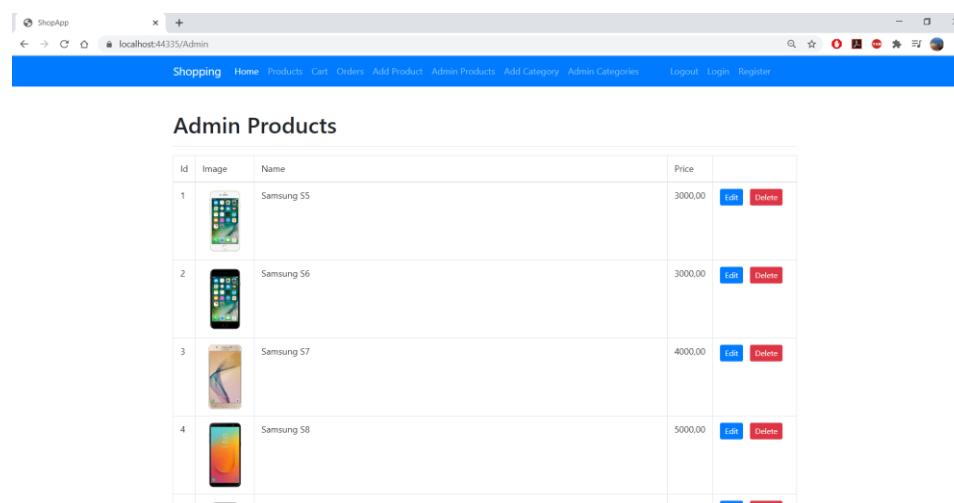
1  [Route("Admin")]
2  public class AdminController : Controller
3  {
4      private readonly IProductService _productService;
5
6      public AdminController(IProductService productService)
7      {
8          _productService = productService;
9      }
10
11     [HttpGet]
12     public IActionResult Index()
13     {
14         ViewData["Title"] = "Index";
15
16         var model = new ProductListModel();
17
18         if (model.Products.Count() > 0)
19         {
20             foreach (var item in model.Products)
21             {
22                 item.ImageUrl = item.Image;
23             }
24         }
25
26         return View(model);
27     }
28
29     [HttpPost]
30     public IActionResult Add([Bind("Name,Price")] Product product)
31     {
32         if (ModelState.IsValid)
33         {
34             _productService.Add(product);
35         }
36
37         return RedirectToAction("Index");
38     }
39
40     [HttpDelete]
41     public IActionResult Delete(int id)
42     {
43         var product = _productService.GetById(id);
44
45         if (product != null)
46         {
47             _productService.Delete(product);
48         }
49
50         return RedirectToAction("Index");
51     }
52
53     [HttpGet]
54     public IActionResult Edit(int id)
55     {
56         var product = _productService.GetById(id);
57
58         if (product != null)
59         {
60             return View(product);
61         }
62
63         return RedirectToAction("Index");
64     }
65
66     [HttpPost]
67     public IActionResult Update(Product product)
68     {
69         if (ModelState.IsValid)
70         {
71             _productService.Update(product);
72         }
73
74         return RedirectToAction("Index");
75     }
76
77     [HttpGet]
78     public IActionResult Details(int id)
79     {
80         var product = _productService.GetById(id);
81
82         if (product != null)
83         {
84             return View(product);
85         }
86
87         return RedirectToAction("Index");
88     }
89
90     [HttpGet]
91     public IActionResult GetProducts()
92     {
93         var products = _productService.GetAll();
94
95         foreach (var item in products)
96         {
97             item.ImageUrl = item.Image;
98         }
99
100        return Json(products);
101    }
102}

```

The Solution Explorer on the right shows the project structure:

- ShopContext.cs
- Memory
- MySQL
- Oracle
- Migrations
- 20190411114829_CreateDatabase.cs
- ShopContextModelSnapshot.cs
- ShopApp.Entities
- Bağlantılar
- Category.cs
- Order.cs
- OrderLine.cs
- Products
- ProductCategory.cs
- ShopApp.WebUI
- Bağlantılar
- Connected Services
- Properties
- wwwroot
- Controllers
- AdminController.cs
- HomeController.cs
- ShopController.cs
- Models
- Tanımlılar

Şekildeki gibi ProductListModel yapısını kullanarak sayfamızı hazırlıyoruz.



Göründüğü gibi ürünlerimizi sayfaya çekebildik.

Açıklama:

The screenshot shows the Visual Studio IDE with the AdminController.cs file open. The code implements an Edit action method that retrieves a product by ID from a service, creates a view model, and returns it to a view. It also contains a POST method for updating the product in the database if the entity is not null.

```

    public IActionResult Edit(int? id)
    {
        if (id == null)
        {
            return NotFound();
        }

        var entity = _productService.GetById(id);

        if (entity == null)
        {
            return NotFound();
        }

        var model = new ProductModel()
        {
            Id = entity.Id,
            Name = entity.Name,
            Price = entity.Price,
            Description = entity.Description,
            ImageUrl = entity.ImageUrl
        };

        return View(model);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Edit(ProductModel model)
    {
        var entity = _productService.GetById(model.Id);

        if (entity == null)
        {
            return NotFound();
        }

        entity.Name = model.Name;
        entity.Description = model.Description;
        entity.ImageUrl = model.ImageUrl;
        entity.Price = model.Price;

        _productService.Update(entity);

        return RedirectToAction("Index");
    }

```

Şekilde görüldüğü gibi benzer şekilde AdminControllere bir action methodu oluşturduk.Yine bir view döndüren methodla,post döndüren method tanımladık.Eğer id bilgisi bulunamazsa kullanıcıyı 404 not found sayfasına yönlendiriyoruz.Değilse entity tanımlama işlemi yapıyoruz.

The screenshot shows the Visual Studio IDE with the Edit.cshtml view file open. The view uses Bootstrap's grid system to display form fields for editing a product. It includes fields for Name, Image URL, Description, and Price, each with its corresponding label and input element. A hidden field for the ID is also present.

```

<form asp-controller="Admin" asp-action="Edit" method="post">
    <input type="hidden" name="Id" value="@Model.Id" />
    <div class="form-group row">
        <label asp-for="Name" class="col-md-2 col-form-label"></label>
        <div class="col-md-10">
            <input asp-for="Name" value="@Model.Name" class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="ImageUrl" class="col-md-2 col-form-label"></label>
        <div class="col-md-10">
            <input asp-for="ImageUrl" value="@Model.ImageUrl" class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Description" class="col-md-2 col-form-label"></label>
        <div class="col-md-10">
            <textareas asp-for="Description" class="form-control" value="@Model.Description" />
        </div>
    </div>
    <div class="form-group row">
        <label asp-for="Price" class="col-md-2 col-form-label"></label>
        <div class="col-md-10">
            <input asp-for="Price" value="@Model.Price" class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <div class="col-md-10 offset-md-2">
            <button type="submit" class="btn btn-primary">Save Product</button>
        </div>
    </div>
</form>

```

Şekilde görüldüğü gibi yine bir edit görünümü oluşturduk

27

**ÖĞRENCİNİN:
İMZASI: Hamdi Uraz Alkış**

KURUM SORUMLUSUNUN:

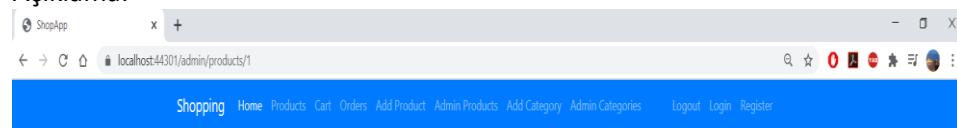
ADI VE SOYADI:

İMZASI:

Yapılan İş: Ürün Düzenleme

Tarih: 05/08/2020

Açıklama:



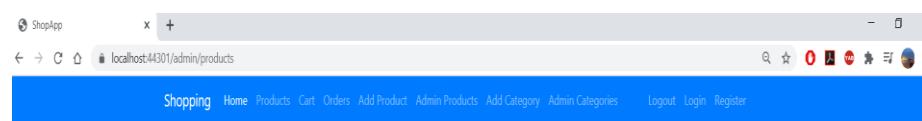
The screenshot shows a browser window titled "ShopApp" with the URL "localhost:44301/admin/products/1". The page is titled "Edit Product". It contains four input fields: "Name" with value "Samsung S5", "ImageUrl" with value "1.jpg", "Description" with value "<p>güzel telefon</p>", and "Price" with value "3000,00". Below the form is a blue "Save Product" button.

Edit Product

Name	Samsung S5
ImageUrl	1.jpg
Description	<p>güzel telefon</p>
Price	3000,00

[Save Product](#)

Görüldüğü gibi ürün düzenleme sayfasına geldiğimizde, oluşturduğumuz sayfa karşımıza çıkıyor.



The screenshot shows a browser window titled "ShopApp" with the URL "localhost:44301/admin/products". The page is titled "Admin Products". It displays a table with four rows, each representing a product: Samsung S5 (Id: 1, Price: 2000,00), Samsung S6 (Id: 2, Price: 3000,00), Samsung S7 (Id: 3, Price: 4000,00), and Samsung S8 (Id: 4, Price: 5000,00). Each row has an "Edit" button and a "Delete" button.

Admin Products

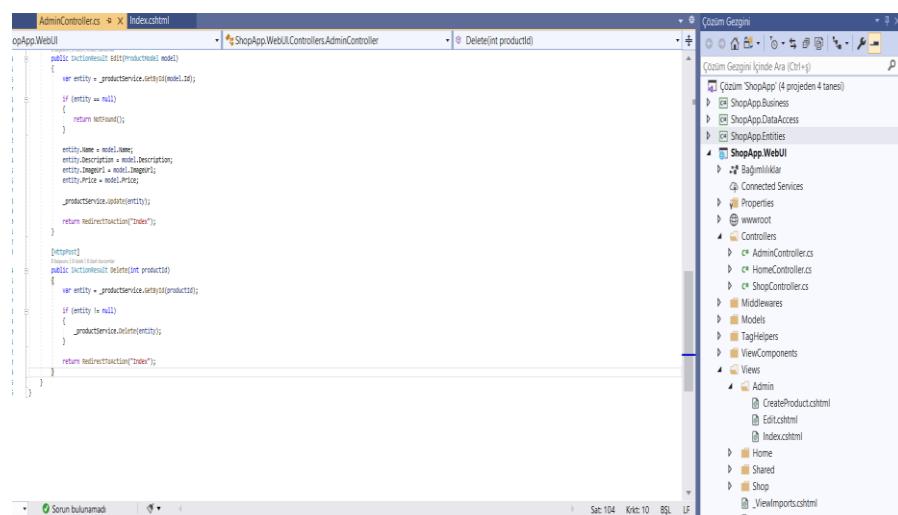
ID	Image	Name	Price	
1		Samsung S5	2000,00	Edit Delete
2		Samsung S6	3000,00	Edit Delete
3		Samsung S7	4000,00	Edit Delete
4		Samsung S8	5000,00	Edit Delete

Görüldüğü gibi 3000 lira olan ürün bedelini 2000 liraya düşürdük.

28

<u>ÖĞRENCİNİN:</u> İMZASI: Hamdi Uraz Alkış	<u>KURUM SORUMLUSUNUN:</u> ADI VE SOYADI: İMZASI:
---	---

Açıklama: Aslında ürün düzenleme ile çok benzer işleri yapıyoruz.



```

public IActionResult Edit(ProductsModel model)
{
    var entity = _productService.GetById(model.Id);

    if (entity == null)
    {
        return NotFound();
    }

    entity.Name = model.Name;
    entity.Description = model.Description;
    entity.ImageUrl = model.ImageUrl;
    entity.Price = model.Price;

    _productService.Update(entity);

    return RedirectToAction("Index");
}

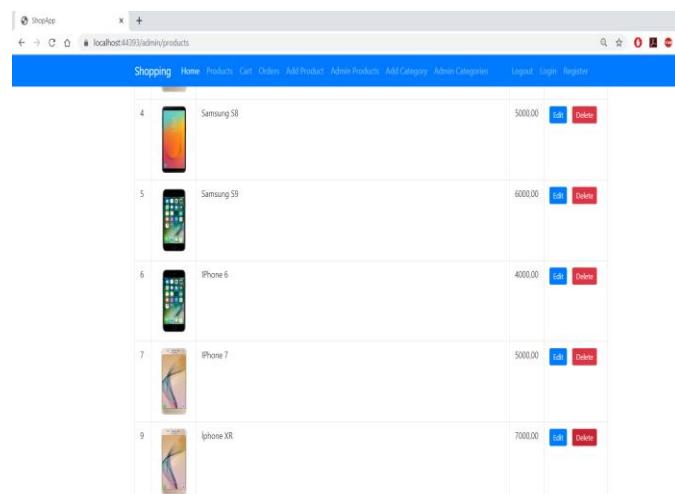
[HttpPost]
[ValidateAntiForgeryToken]
public IActionResult Delete(int productId)
{
    var entity = _productService.GetById(productId);

    if (entity != null)
    {
        _productService.Delete(entity);
    }

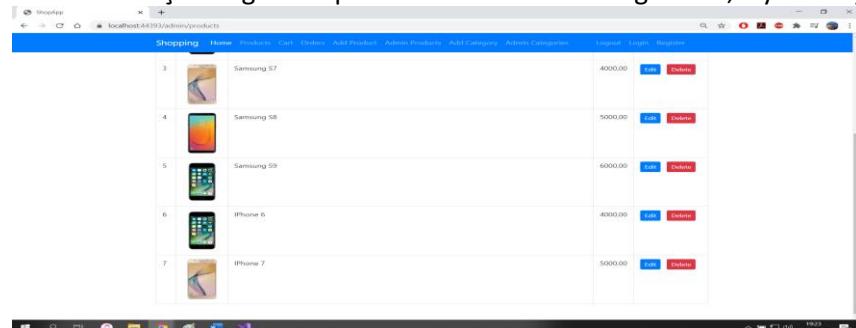
    return RedirectToAction("Index");
}

```

Bir delete actionu oluşturduk. Eğer gelen id null değilse,_productservice sınıfındaki delete methodunu aktif hale getiriyoruz.Ve return RedirectToAction ile işimizi index'e aktarıyoruz.



Önceden oluşturduğumuz iphoneXR telefonunu sildiğimizde,kayıt siliniyor.



Açıklama:

Ürün ekleme,silme,güncelleme işlemlerinin aynısı kategoriler için de yaptıktan sonra,Identity yapısını kullanıcı işlemleri için eklememiz gerekiyor. İlk olarak webUI katmanına Identity adında bir klasör oluşturuyoruz.İçine kullanıcıyı temsilen ApplicationUser adında bir sınıf oluşturuyoruz.

```

    1  using Microsoft.AspNetCore.Identity;
    2  using System;
    3  using System.Collections.Generic;
    4  using System.Linq;
    5  using System.Threading.Tasks;
    6
    7  namespace ShopApp.WebUI.Identity
    8  {
    9      public class ApplicationUser : IdentityUser
    10     {
    11         public string FullName { get; set; }
    12     }
    13 }
    14
    15
  
```

Bunun için Identity framework'unu kullanacağımız için,IdentityUser den sınıfımızı türetiyoruz. DbContext türünde bir sınıf daha aynı klasör içine ApplicationDbContextDbContext adında bir sınıf oluşturuyoruz.

```

    1  using Microsoft.Extensions.DependencyInjection;
    2  using Microsoft.AspNetCore.Builder;
    3  using Microsoft.AspNetCore.Hosting;
    4  using Microsoft.Extensions.Configuration;
    5  using Microsoft.Extensions.DependencyInjection;
    6  using Microsoft.Extensions.Hosting;
    7  using Microsoft.Extensions.Logging;
    8  using Microsoft.Extensions.Options;
    9  using Microsoft.Extensions.DependencyInjection;
    10 using Microsoft.Extensions.Configuration;
    11 using Microsoft.Extensions.DependencyInjection;
    12 using ShopApp.Business.Abstract;
    13 using ShopApp.Business.Concrete;
    14 using ShopApp.DataAccess.Concrete;
    15 using ShopApp.DataAccess.Concrete.EntityFramework;
    16 using ShopApp.WebUI.Identity;
    17 using ShopApp.WebUI.Middleware;
    18
    19 namespace ShopApp.WebUI
    20 {
    21     public class Startup
    22     {
    23         // This method gets called by the runtime. Use this method to add services to the container.
    24         // For more information on how to configure your app's service, visit https://go.microsoft.com/fwlink/?linkid=398968
    25         public void ConfigureServices(IServiceCollection services)
    26         {
    27             services.AddDbContext<ApplicationDbContext>(options =>
    28                 options.UseSqlServer(Configuration.GetConnectionString("IdentityConnection")));
    29
    30             services.AddIdentity()
    31                 .AddEntityFrameworkStores<ApplicationDbContext>()
    32                 .AddDefaultTokenProviders();
    33
    34             services.AddScoped<IProductService, EfCoreProductService>();
    35             services.AddScoped<ICategoryService, EfCoreCategoryService>();
    36             services.AddScoped<IPriceService, ProductPriceService>();
    37         }
    38
    39         // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    40         public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    41         {
    42             if (env.IsDevelopment())
    43             {
    44                 app.UseDeveloperExceptionPage();
    45             }
    46             else
    47             {
    48                 app.UseExceptionHandler("/Home/Error");
    49                 app.UseHsts();
    50             }
    51
    52             app.UseHttpsRedirection();
    53             app.UseStaticFiles();
    54
    55             app.UseRouting();
    56
    57             app.UseAuthorization();
    58
    59             app.UseEndpoints(endpoints =>
    60             {
    61                 endpoints.MapControllerRoute(
    62                     name: "default",
    63                     pattern: "{controller=Home}/{action=Index}/{id?}");
    64             });
    65         }
    66     }
    67 }
  
```

Bu işlemleri gerçekleştirebilmemiz için projenin startup dosyasına
services.AddDbContext<ApplicationDbContext>(options =>
 options.UseSqlServer(Configuration.GetConnectionString("IdentityConnection")));
diyerek tanımladık.Yazılan IdentityConnection'u appsettings.json dosyasında {
"ConnectionStrings": {
 "IdentityConnection": "Server=(localdb)\\MSSQLLocalDB;Database=ShopDb;integrated
 security=true;"
}, diyerek belirttiğimiz.

ÖĞRENCİNİN: İMZASI: Hamdi Uraz Alkış	KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:
---	---

Açıklama:

```

Komut İstemi
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

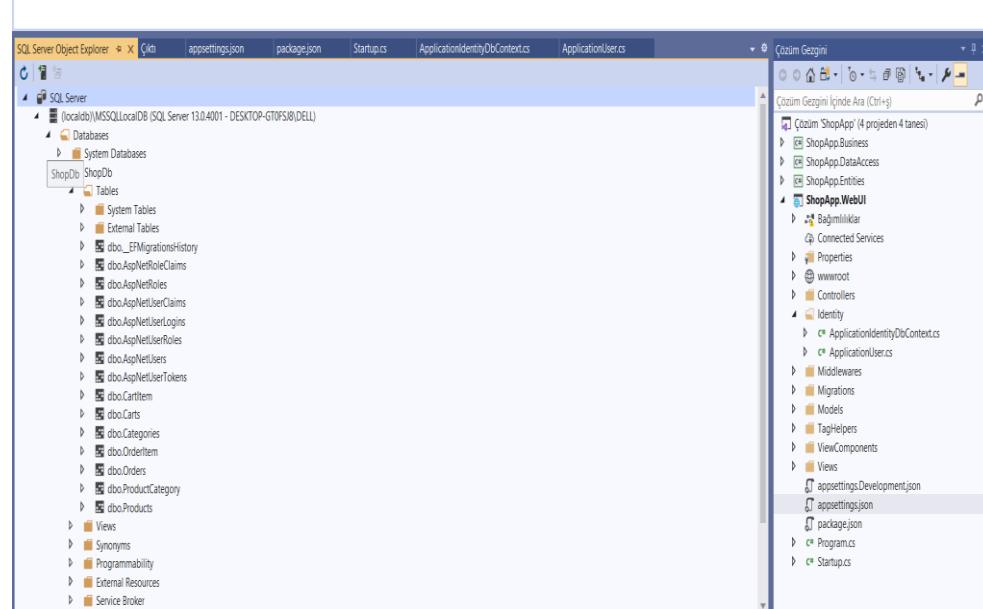
C:\Users\DELL>d:

D:>cd D:\Projects\WEB\AlkislaretiCaret\ShopApp\ShopApp.WebUI

D:\Projects\WEB\AlkislaretiCaret\ShopApp\ShopApp.WebUI>dotnet ef migrations add AddingIdentity
  
```

Diyerek, Identity ile alakalı bir migration oluşturuyoruz.

Aynı şekilde dotnet ef database update komutuyla da database'i getiriyoruz.



Göründüğü gibi database gelmiş oldu. IdentityUserden gelen kolonlar AspNetUser tablosunda mevcut.

ÖĞRENCİNİN: İMZASI: Hamdi Uraz Alkış	KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:
---	---

10/08/2020 Tarihinden 14/08/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
10/08/2020	Pazartesi	Kullanıcı Kayıt İşlemi	32/36	9
11/08/2020	Salı	Onay Maili İle Hesap Aktif Etme	37/38	9
12/08/2020	Çarşamba	Şifremi Unuttum	39/39	9
13/08/2020	Perşembe	Şifre Sıfırlama	40/40	9
14/08/2020	Cuma	Admin Kullanıcısı Yaratma	41/44	9
.../.../...	Cumartesi		.../....	
TOPLAM SÜRE (Saat)				45
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış İmzası: Hamdi Uraz Alkış Çalıştığı Bölüm: Bilişim Teknolojileri		Adı SOYADI: Unvanı: İmza/Kaşesi:		

Yapılan İş: Kullanıcı Kayıt İşlemi

Tarih: 10/08/2020

Açıklama:

İlk olarak AdminController sınıfının başına [Authorize] yazarak, kullanıcının sayfaya erişebilmesi için login işlemi gerçekleştirmesi gerektiğini sağlıyoruz.

```
ShopApp.WebUI.Controllers.AccountController.cs (RegisterModel.cs tab)

using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Logging;
using ShopApp.WebUI.Controllers;
using ShopApp.Business;
using ShopApp.DataAccess;
using ShopApp.Entities;
using ShopApp.WebUI.Models;
using ShopApp.WebUI.Controllers.TaskHelper;

namespace ShopApp.WebUI.Controllers
{
    public class AccountController : Controller
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;
        private readonly ILogger<AccountController> _logger;
        private readonly TaskHelper _taskHelper;

        public AccountController(UserManager<ApplicationUser> userManager, SignInManager<ApplicationUser> signInManager, ILogger<AccountController> logger, TaskHelper taskHelper)
        {
            _userManager = userManager;
            _signInManager = signInManager;
            _logger = logger;
            _taskHelper = taskHelper;
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser
                {
                    Username = model.Username,
                    Email = model.Email,
                    FullName = model.FullName
                };

                var result = await _userManager.CreateAsync(user, model.Password);

                if (result.Succeeded)
                {
                    // generate token
                    // send email

                    return RedirectToAction("account", "Login");
                }
            }

            ModelState.AddModelError("", "Bilinmeyen hata oluştu lütfen tekrar deneyiniz.");
            return View(model);
        }
    }
}
```

İlk olarak AccountController adında bir sınıf oluşturduk. Şekildeki gibi constructor methodunu doldurduk ve kullanıcı oluşturmak için register isminde bir action methodu oluşturduk. Bizim bunun için bir model ile çalışıyor olmamız gereklidir. Bunun için RegisterModel isminde bir model oluşturduk.

```
ShopApp.WebUI
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Threading.Tasks;
6
7  namespace ShopApp.WebUI.Models
8  {
9      public class RegisterModel
10     {
11         [Required]
12         [EmailAddress]
13         public string Email { get; set; }
14
15         [Required]
16         [EmailAddress]
17         public string Username { get; set; }
18
19         [Required]
20         [DataType(DataType.Password)]
21         public string Password { get; set; }
22
23         [Required]
24         [DataType(DataType.Password)]
25         [Compare("Password")]
26         public string RePassword { get; set; }
27
28         [Required]
29         [DataType(DataType.EmailAddress)]
30         [EmailAddress]
31         public string Email2 { get; set; }
32     }
33 }
```

Özelliklerimizi şekildeki gibi tanımladık.

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADLIVE SOYARD

İMZASI:

Yapılan İş: Kullanıcı Kayıt İşlemi

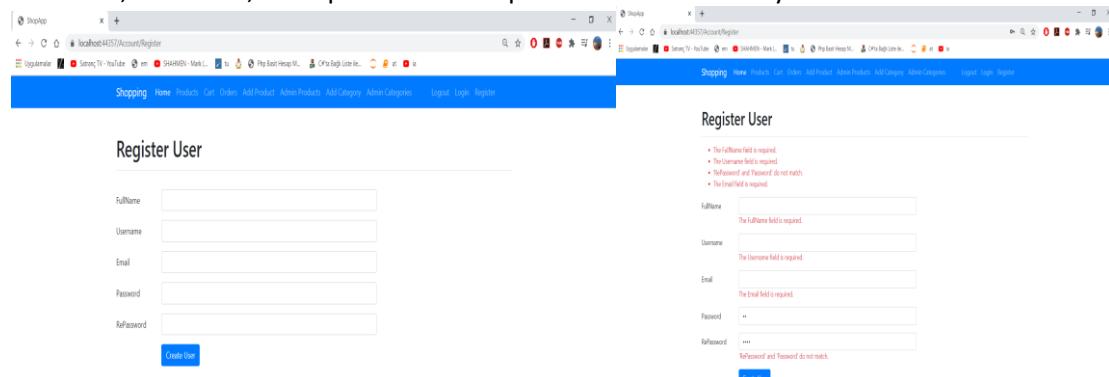
Tarih: 10/08/2020

Açıklama:

The screenshot shows the Visual Studio interface. In the center, the code editor displays the `Register.cshtml` file, which contains ASP.NET MVC view code for user registration. The code includes HTML forms, labels, and validation messages. On the right, the Solution Explorer shows the project structure with files like `AccountController.cs`, `AdminController.cs`, `CategoryListViewModel.cs`, etc.

```
<div class="row">
    <div class="col-md-4">
        <div class="form-group">
            <label for="FullName">Full Name</label>
            <input type="text" id="FullName" name="FullName" value="John Doe" />
            <span class="text-danger" for="FullName" id="ValidationMessageForFullName">The full name field is required.</span>
        </div>
        <div class="form-group">
            <label for="Username">Username</label>
            <input type="text" id="Username" name="Username" value="johndoe" />
            <span class="text-danger" for="Username" id="ValidationMessageForUsername">The username field is required.</span>
        </div>
        <div class="form-group">
            <label for="Email">Email</label>
            <input type="text" id="Email" name="Email" value="john.doe@example.com" />
            <span class="text-danger" for="Email" id="ValidationMessageForEmail">The email field is required.</span>
        </div>
        <div class="form-group">
            <label for="Password">Password</label>
            <input type="password" id="Password" name="Password" value="P@ssw0rd" />
            <span class="text-danger" for="Password" id="ValidationMessageForPassword">The password field is required.</span>
        </div>
        <div class="form-group">
            <label for="RePassword">Repassword</label>
            <input type="password" id="RePassword" name="RePassword" value="P@ssw0rd" />
            <span class="text-danger" for="RePassword" id="ValidationMessageForRePassword">Repassword and Password do not match.</span>
        </div>
    </div>
    <div class="col-md-4">
        <h3>Create New Account</h3>
        <hr/>
        <div class="form-group">
            <label for="FirstName">First Name</label>
            <input type="text" id="FirstName" name="FirstName" value="John" />
            <span class="text-danger" for="FirstName" id="ValidationMessageForFirstName">The first name field is required.</span>
        </div>
        <div class="form-group">
            <label for="LastName">Last Name</label>
            <input type="text" id="LastName" name="LastName" value="Doe" />
            <span class="text-danger" for="LastName" id="ValidationMessageForLastName">The last name field is required.</span>
        </div>
        <div class="form-group">
            <label for="Address">Address</label>
            <input type="text" id="Address" name="Address" value="123 Main Street" />
            <span class="text-danger" for="Address" id="ValidationMessageForAddress">The address field is required.</span>
        </div>
        <div class="form-group">
            <label for="City">City</label>
            <input type="text" id="City" name="City" value="Anytown" />
            <span class="text-danger" for="City" id="ValidationMessageForCity">The city field is required.</span>
        </div>
        <div class="form-group">
            <label for="State">State</label>
            <input type="text" id="State" name="State" value="CA" />
            <span class="text-danger" for="State" id="ValidationMessageForState">The state field is required.</span>
        </div>
        <div class="form-group">
            <label for="ZipCode">Zip Code</label>
            <input type="text" id="ZipCode" name="ZipCode" value="90210" />
            <span class="text-danger" for="ZipCode" id="ValidationMessageForZipCode">The zip code field is required.</span>
        </div>
        <div class="form-group">
            <label for="Country">Country</label>
            <input type="text" id="Country" name="Country" value="USA" />
            <span class="text-danger" for="Country" id="ValidationMessageForCountry">The country field is required.</span>
        </div>
        <div class="form-group">
            <label for="Phone">Phone</label>
            <input type="text" id="Phone" name="Phone" value="555-555-5555" />
            <span class="text-danger" for="Phone" id="ValidationMessageForPhone">The phone number field is required.</span>
        </div>
        <div class="form-group">
            <label for="Email2">Email</label>
            <input type="text" id="Email2" name="Email2" value="john.doe@example.com" />
            <span class="text-danger" for="Email2" id="ValidationMessageForEmail2">The email field is required.</span>
        </div>
        <div class="form-group">
            <label for="Password2">Password</label>
            <input type="password" id="Password2" name="Password2" value="P@ssw0rd" />
            <span class="text-danger" for="Password2" id="ValidationMessageForPassword2">The password field is required.</span>
        </div>
        <div class="form-group">
            <label for="RePassword2">Repassword</label>
            <input type="password" id="RePassword2" name="RePassword2" value="P@ssw0rd" />
            <span class="text-danger" for="RePassword2" id="ValidationMessageForRePassword2">Repassword and Password do not match.</span>
        </div>
    </div>
    <div class="col-md-4">
        <div class="form-group">
            <input type="button" value="Create User" />
        </div>
    </div>
</div>
```

Kayıt işlemi sayfasını da şekildeki gibi oluşturduk. Model sayfasında belirttiğimiz gibi kullanıcıdan `fullName`, `username`, `email`, `password` ve `repassword` alanlarını alıyoruz.



Şekildeki gibi kullanıcı kayıt sayfasını oluşturduk ve belirtildiği gibi hata mesajlarını da yazdırabiliyoruz.

Açıklama:

AccountController sınıfında methodları oluşturmamız gerekiyor.

The screenshot shows the Visual Studio interface. On the left, the 'AccountController.cs' file is open in the code editor, displaying C# code for handling user login and registration. On the right, the 'Çözüm Gezgini' (Solution Explorer) shows the project structure, including controllers like AccountController.cs, AdminController.cs, and HomeController.cs, and views like Login.cshtml and Register.cshtml.

```

52     ModelState.AddModelError("", "Kullanıcı adı veya şifre yanlış.");
53     return View(model);
54   }
55 
56   [HttpPost]
57   [ValidateAntiForgeryToken]
58   public ActionResult Login(LoginModel model, string returnUrl = null)
59   {
60     returnUrl = returnUrl ?? "~/";
61 
62     if (ModelState.IsValid)
63     {
64       return View(model);
65     }
66 
67     var user = await _userManager.FindByNameAsync(model.Username);
68 
69     if (user == null)
70     {
71       ModelState.AddModelError("", "Bu kullanıcı ile bir kez hesap oluşturulmuştur.");
72       return View(model);
73     }
74 
75     var result = await _signInManager.PasswordSignInAsync(model.Username, model.Password, true, false);
76 
77     if (result.Succeeded)
78     {
79       return RedirectToAction(returnUrl);
80     }
81 
82     ModelState.AddModelError("", "Kullanıcı adı ve ya şifre yanlış.");
83     return View(model);
84   }
85 }

```

Şekilde görüldüğü gibi AccountController üzerinde gerekli kullanıcı kontrollerini yapıyoruz.
Kullanıcı Login olduktan sonra da oluşturduğumuz login view’inde bunları gösteriyoruz.

The screenshot shows the Visual Studio interface. On the left, the 'Login.cshtml' file is open in the code editor, displaying the HTML and Razor code for the login form. On the right, the 'Çözüm Gezgini' (Solution Explorer) shows the project structure, including controllers like AccountController.cs, AdminController.cs, and HomeController.cs, and views like Login.cshtml and Register.cshtml.

```

1 <form class="form">
2   <div>[...]
3   <div>[...]
4   <div>[...]
5   <div>[...]
6   <div>[...]
7   <div>[...]
8   <div>[...]
9   <div>[...]
10  <div>[...]
11  <div>[...]
12  <div>[...]
13  <div>[...]
14  <div>[...]
15  <div>[...]
16  <div>[...]
17  <div>[...]
18  <div>[...]
19  <div>[...]
20  <div>[...]
21  <div>[...]
22  <div>[...]
23  <div>[...]
24  <div>[...]
25  <div>[...]
26  <div>[...]
27  <div>[...]
28  <div>[...]
29  <div>[...]
30  <div>[...]
31  <div>[...]
32  <div>[...]
33  <div>[...]
34  <div>[...]
35  <div>[...]
36  <div>[...]
37  <div>[...]
38  <div>[...]
39  <div>[...]
40  <div>[...]
41  <div>[...]
42  <div>[...]
43  <div>[...]
44  <div>[...]
45  <div>[...]
46  <div>[...]
47  <div>[...]
48  <div>[...]
49  <div>[...]
50  <div>[...]
51  <div>[...]
52  </div>
53 </form>

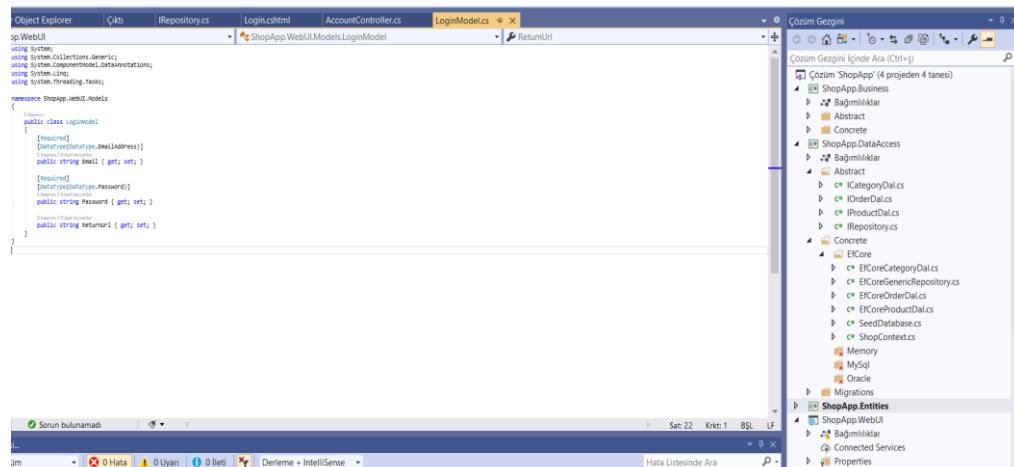
```

Tabii ki de bu sayfayı oluşturmuş olduğumuz LoginModel den alıyoruz.

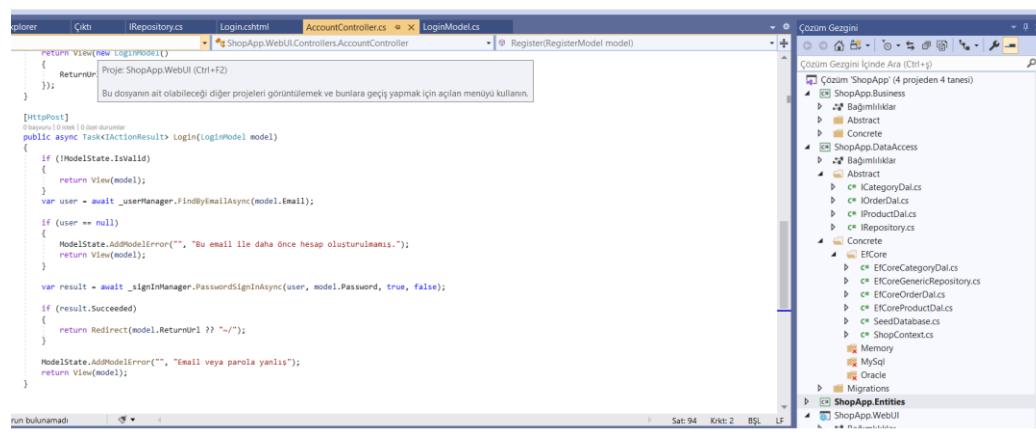
Yapılan İş: Email ile Kullanıcı Kayıt

Tarih: 10/08/2020

Açıklama:



LoginModel sayfasını email olarak güncelliyoruz.



AccountController sayfasındaki FindByName yapısını FindByEmail olarak değiştiriyoruz.

Gerekli kelime düzeltmelerini Login view'ı üzerinde de gerçekleştirdik. Kullanıcı login olduktan sonra kayıt ile ilgili ilgili verileri görememesi gerekiyor bunun için, _navbar view'inde



Kullanıcının login kontrolünü yapıp eğer login olmuşsa logout linkini görmeli eğer değilse giriş yap ve kayıt linklerini görmeli.

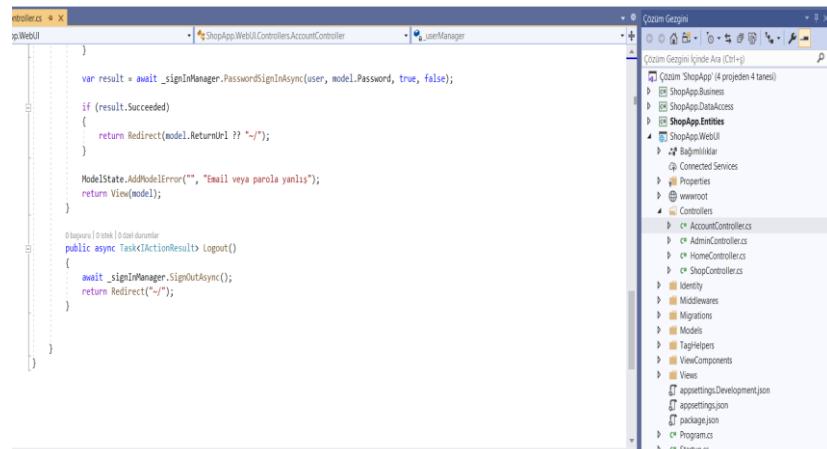
35

**ÖĞRENCİNİN:
İMZASI: Hamdi Uraz Alkış**

**KURUM SORUMLUSUNUN:
ADI VE SOYADI:
İMZASI:**

Açıklama:

Kullanıcı logout'a tıkladığı zaman, tarayıcısındaki cookie'i silmesi gerekiyor.



The screenshot shows the Visual Studio IDE with the AccountController.cs file open in the editor. The code defines a Logout action method that signs out the user using the _signInManager.SignOutAsync() method and then redirects to the root URL. A note in the code states: "Kullanıcı logout'a tıkladığı zaman, tarayıcısındaki cookie'i silmesi gerekiyor." (The user needs to have their cookie deleted when they click the logout button). The Solution Explorer on the right shows the project structure, including the ShopApp.WebUI project and its subfolders like Controllers, Models, and Views.

```
public class AccountController : Controller
{
    private readonly SignInManager<User> _signInManager;
    private readonly UserManager<User> _userManager;

    public AccountController(SignInManager<User> signInManager, UserManager<User> userManager)
    {
        _signInManager = signInManager;
        _userManager = userManager;
    }

    [HttpPost]
    public async Task<ActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return Redirect("~/");
    }
}
```

AccountController sınıfına bir logout action methodu ekledik.

Açıklama:

Kullanıcı şu anda kayıt olduğunda, adresine hiçbir onay mesajı gelmeden direkt kayıt oluyor. Şimdi mail gönderim yoluyla onay sağlamaya çalışacağız. AccountController'de ConfirmEmail adında bir method oluşturalım.

```

ShopApp.WebUI
107     await _userManager.SignInUserAsync();
108     return Redirect("~/");
109 }
110 }
111 }
112 }

113 //başka bir istek | 0 özel durumları
114 public async Task<ActionResult> ConfirmEmail(string userId, string token)
115 {
116     if (userId==null || token ==null)
117     {
118         TempData["message"] = "Geçersiz token.";
119         return View();
120     }
121
122     var user = await _userManager.FindByIdAsync(userId);
123     if (user!=null)
124     {
125         var result = await _userManager.ConfirmEmailAsync(user, token);
126         if (result.Succeeded)
127         {
128             TempData["message"] = "Hesabınız onaylandı";
129             return View();
130         }
131     }
132
133     TempData["message"] = "Hesabınız onaylanmadı.";
134     return View();
135 }

```

Eğer gönderdiğimiz userId boşsa veya token boşsa, return view ile mail onayla sayfasını getirelim. Daha sonra bunun görünüm sayfasını oluşturualım.

```

1 <h1>Confirm Email</h1>
2 <br>
3 <div class="alert alert-warning">
4     @if(TempData["message"]!=null)
5     {
6         @TempData["message"]
7     }
8 </div>

```

Eğer view datanın içi boş değilse, TempData'yı yazdırıyoruz.

Doğrulama emaili göndermek içinse www.sendgrid.com sitesinden yardım alacağız. Oradan ücretsiz planı seçip, üyelik hesabı oluşturduktan sonra API key yaratma bölümünden api key kodu alıyoruz. Projenin startup bölümünde services kısmında services.AddTransient<IEmailSender, EmailSender>(); Kodunu ekliyoruz. Sonra projenin webUI katmanına emailsender adında bir sınıf oluşturuyoruz.

```

1 <using> System;
2 <using> System.Threading.Tasks;
3 <using> Microsoft.AspNetCore.Mvc;
4 <using> Microsoft.Extensions.Logging;
5 <using> ShopApp.Data;
6 <using> ShopApp.Entities;
7 <using> SendGrid;
8 <using> SendGrid.Helpers.Mail;
9
10 public class EmailSender : IEmailSender
11 {
12     private const string SendGridKey = "SG.wgqIgAn_CSnK2vBe2pFQgQ.QjQINURqeq9Aniab0SY-BPCCB3J_xlgWzrmOsycFw";
13
14     public Task SendEmailAsync(string email, string subject, string htmlMessage)
15     {
16         return Execute(SendGridKey, subject, htmlMessage, email);
17     }
18
19     private Task Execute(string sendGridKey, string subject, string htmlMessage, string email)
20     {
21         var client = new SendGridClient(sendGridKey);
22
23         var msg = new SendGridMessage()
24         {
25             From = new EmailAddress("info@shopapp.com", "Shop App"),
26             Subject = subject,
27             PlainTextContent = message,
28             HtmlContent = htmlMessage
29         };
30
31         msg.Addto(new EmailAddress(email));
32         return client.SendEmailAsync(msg);
33     }
34 }

```

Açıklama:

Öncelikle email için sendGrid paketini yüklememiz gerekiyor.

```
D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI>dotnet add package SendGrid
  Geri yüklenenek projeler belirleniyor...
  Writing C:\Users\DELL\AppData\Local\Temp\tmp3DD.tmp
info : D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI\ShopApp.WebUI.csproj' projesine 'SendGrid' paketi için PakageReference ekleniyor.
info : D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI\ShopApp.WebUI.csproj için paketler geri yükleniyor...
info : GET https://api.nuget.org/v3-flatcontainer/sendgrid/index.json
info : OK https://api.nuget.org/v3-flatcontainer/sendgrid/index.json 174 ms
info : GET https://api.nuget.org/v3-flatcontainer/sendgrid/9.21.0/sendgrid.9.21.0.nupkg
info : OK https://api.nuget.org/v3-flatcontainer/sendgrid/9.21.0/sendgrid.9.21.0.nupkg 102 ms
info : starkbank-ecdsa/index.json 497 ms
info : GET https://api.nuget.org/v3-flatcontainer/starkbank-ecdsa/1.2.0/starkbank-ecdsa.1.2.0.nupkg
info : OK https://api.nuget.org/v3-flatcontainer/starkbank-ecdsa/1.2.0/starkbank-ecdsa.1.2.0.nupkg 102 ms
info : starkbank-ecdsa 1.2.0 yükleniyor.
info : SendGrid paketi ile yükleniyor.
info : SendGrid paketi sürüm '9.21.0' için PackageReference, 'D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI\ShopApp.WebUI.csproj' dosyasında güncelleştirildi.
info : Geri yükleme işleniyor...
info : Varsayılan dosyası diske yazılıyor. Yol: D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI\obj\project.assets.json
log : D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI\ShopApp.WebUI.csproj geri yüklendi (2,73 sec içinde).

D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.WebUI>
```

Şekildeki gibi projeye kurulumu gerçekleştiriyoruz.

```
AccountController.cs
ShopApp.WebUI
ShopApp.WebUI.Controllers.AccountController
userManager

111     await _signInManager.SignOutAsync();
112     return RedirectToAction("Index");
113 }
114 }

115 // GET: /Account/ConfirmEmail/{userId}/{token}
116 public async Task ConfirmEmail(string userId, string token)
117 {
118     if (userId==null || token ==null)
119     {
120         TempData["message"] = "Geçersiz token.";
121         return View();
122     }
123 }

124 var user = await _userManager.FindByIdAsync(userId);
125 if (user==null)
126 {
127     var result = await _userManager.ConfirmEmailAsync(user, token);
128     if (result.Succeeded)
129     {
130         TempData["message"] = "Hesabınız onaylandı";
131         return View();
132     }
133 }
134 }

135 TempData["message"] = "Hesabınız onaylanmadı.";
136 return View();
137 }
```

```
AccountController.cs
ShopApp.WebUI
ShopApp.WebUI.Controllers.AccountController
RegisterRegisterModel model

41     username = model.username;
42     Email = model.Email;
43     FullName = model.FullName;
44 };

45 var result = await _userManager.CreateAsync(user, model.Password);

46 if (result.Succeeded)
47 {
48     // generate token
49     var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
50     var callbackUrl = Url.Action("ConfirmEmail", "Account", new
51     {
52         userId = user.Id,
53         token = code
54     });
55 }

56 // send email
57 await _emailSender.SendEmailAsync(model.Email, "Hesabınızı Onaylayınız.", $"Lütfen email hesabınızı onaylamak için linke 

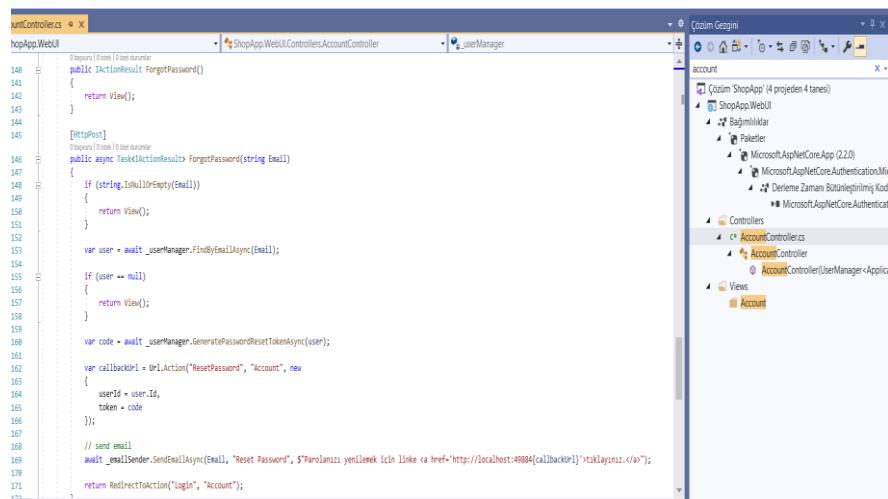
```

AccountController sınıfında da callback url yi aktif hale getiriyoruz.

ÖĞRENCİNİN:**İMZASI: Hamdi Uraz Alkış****KURUM SORUMLUSUNUN:****ADI VE SOYADI:****İMZASI:**

Açıklama:

İlk olarak AccountController sınıfında, get ve post versiyonları hazırlamamız gerekiyor.

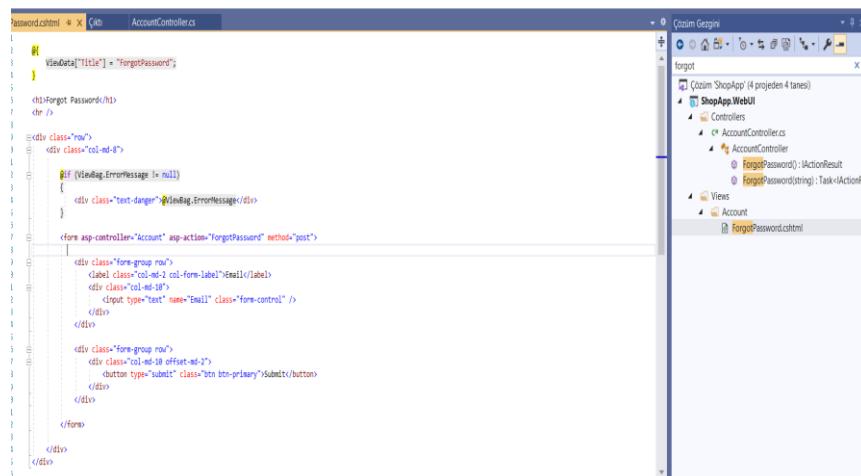


```

148     [HttpGet]
149     public IActionResult ForgotPassword()
150     {
151         return View();
152     }
153
154     [HttpPost]
155     [ValidateAntiForgeryToken]
156     public async Task<ActionResult> ForgotPassword(string Email)
157     {
158         if (string.IsNullOrEmpty(Email))
159         {
160             return View();
161         }
162
163         var user = await _userManager.FindByEmailAsync(Email);
164
165         if (user == null)
166         {
167             return View();
168         }
169
170         var code = await _userManager.GeneratePasswordResetTokenAsync(user);
171
172         var callbackUrl = Url.Action("ResetPassword", "Account", new
173         {
174             userId = user.Id,
175             token = code
176         });
177
178         // send email
179         await _emailSender.SendEmailAsync(Email, "Reset Password", $"Parolamı yenilemek için lütfen tıklayınız.");
180
181         return RedirectToAction("Login", "Account");
182     }

```

Kullanıcı dışarıdan bir mail bilgisi girecek. Eğer gelen bilgi boş ise kullanıcıyı sayfaya geri gönderiyoruz. `_userManager.FindByEmailAsync` kullanıcının mail'i alıyor. Kullanıcıyla ilgili de gerekli kontrolleri yaptıktan sonra, yine aynı şekilde mail onaylama sayfasında yaptığıımız işlemleri tekrarlıyoruz.

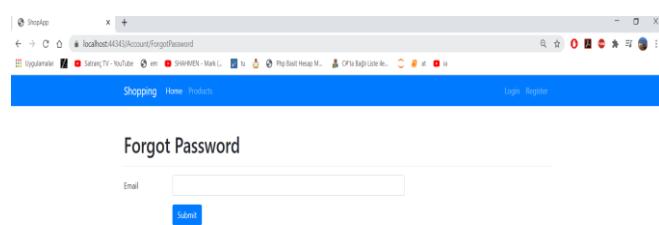


```

1 <h1>@ViewData["Title"] = "ForgotPassword";</h1>
2
3 <div><h1>Forgot Password</h1>
4 <br />
5 <div class="row">
6     <div class="col-md-8">
7         <div>@ViewBag.ErrorMessage</div>
8         <form asp-controller="Account" asp-action="ForgotPassword" method="post">
9             <div class="form-group row">
10                <label class="col-md-2 col-form-label">Email</label>
11                <div class="col-md-8">
12                    <input type="text" name="Email" class="form-control" />
13                </div>
14            </div>
15            <div class="form-group row">
16                <div class="col-md-8 offset-md-2">
17                    <button type="submit" class="btn btn-primary">Submit</button>
18                </div>
19            </div>
20        </form>
21    </div>
22 </div>

```

Şekildeki gibi ForgotPassword görünüm sayfasını da oluşturdukten sonra aşağıdaki gibi bir görünüm elde ediyoruz.



Yapılan İş: Şifre Sıfırlama

Tarih: 13/08/2020

Açıklama:

Gelen maile göre doğrulama işlemleri yaptıktan sonra, şifre resetleme işlemini yapalım. Öncelikle benzer şekilde AccountController üzerinde ResetPassword action'u oluşturmamız gereklidir.

The screenshot shows the Visual Studio interface. In the center, there is a code editor window titled 'AccountController.cs' containing C# code for a 'ResetPassword' action. The code handles token validation, finds the user by email, and performs password reset. On the right side, there is a 'Çözüm Gezgini' (Solution Explorer) window showing the project structure for 'ShopApp.WebUI'. It includes 'Controllers', 'Models', and 'Views' folders, with specific files like 'ResetPasswordModel.cs' and 'ResetPassword.cshtml' visible under the 'Views/Account' folder.

```
174     [HttpPost]
175     [ValidateAntiForgeryToken]
176     public async Task<ActionResult> ResetPassword(ResetPasswordModel model)
177     {
178         if (model == null)
179         {
180             return RedirectToAction("Home", "Index");
181         }
182         var model = new ResetPasswordModel { Token = token };
183         return View(model);
184     }
185 
186     [HttpPost]
187     [ValidateAntiForgeryToken]
188     public async Task<ActionResult> ResetPassword(ResetPasswordModel model)
189     {
190         if (ModelState.IsValid)
191         {
192             return View(model);
193         }
194         var user = await userManager.FindByEmailAsync(model.Email);
195         if (user == null)
196         {
197             return RedirectToAction("Home", "Index");
198         }
199         var result = await userManager.ResetPasswordAsync(user, model.Token, model.Password);
200         if (result.Succeeded)
201         {
202             return RedirectToAction("Login", "Account");
203         }
204         return View(model);
205     }
206 }
```

Kullanıcı parolasını unuttuğunda mail adresi aracılığıyla bir token geliyor ve bu token ile kullanıcının parolasını güncellemesi gerekiyor. Ürettiğimiz kodda bir userID ve token almamız gerekiyor. Kullanıcı ResetPassword view'ini post ettiği zaman, resetpassword'e doğrulama kodu, mail adresi ve parola göndermemiz gerekiyor. Bunun için resetpassword model'i oluşturduk.

The screenshot shows the Visual Studio interface. In the center, there is a code editor window titled 'ResetPasswordModel.cs' containing C# code for a 'ResetPasswordModel' class. This class has properties for 'Token' (string), 'Email' (string), and 'Password' (string). Annotations like [Required] and [DataType(DataType.EmailAddress)] are used for the Email property. On the right side, there is a 'Çözüm Gezgini' (Solution Explorer) window showing the project structure for 'ShopApp'. It includes 'Controllers', 'Models', and 'Views' folders, with specific files like 'ResetPasswordModel.cs' visible under the 'Models' folder.

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Threading.Tasks;
6 
7  namespace ShopApp.WebUI.Models
8  {
9      public class ResetPasswordModel
10     {
11         [Required]
12         [DataType(DataType.EmailAddress)]
13         public string Token { get; set; }
14         [Required]
15         [DataType(DataType.EmailAddress)]
16         public string Email { get; set; }
17         [Required]
18         [DataType(DataType.Password)]
19         public string Password { get; set; }
20     }
21 }
```

40

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

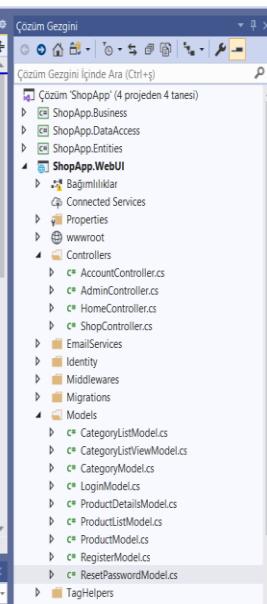
İMZASI:

Açıklama:

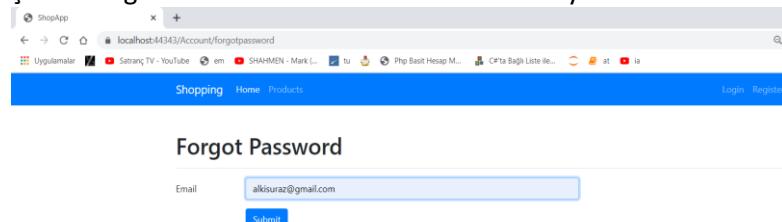
```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>Şifre Sıfırlama</title>
        <link href="~/css/site.css" rel="stylesheet" />
        <script src="~/modules/jquery-validation/dist/jquery.validate.min.js"></script>
        <script src="~/modules/jquery-validation-unobtrusive/dist/jquery.validate.unobtrusive.min.js"></script>
    </head>
    <body>
        <div class="row">
            <div class="col-md-4">
                <h1>Şifre Sıfırlama</h1>
                <div>
                    <div class="form-group row">
                        <label asp-for="Email" class="col-md-2 col-form-label"></label>
                        <div class="col-md-10">
                            <input asp-for="Email" value="Model.Email" class="form-control" />
                            <span asp-validation-for="Email" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group row">
                        <label asp-for="Password" class="col-md-2 col-form-label"></label>
                        <div class="col-md-10">
                            <input asp-for="Password" value="Model.Password" class="form-control" />
                            <span asp-validation-for="Password" class="text-danger"></span>
                        </div>
                    </div>
                    <div class="form-group row">
                        <div class="col-md-10 offset-md-2">
                            <button type="submit" class="btn btn-primary">Submit</button>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </body>
</html>

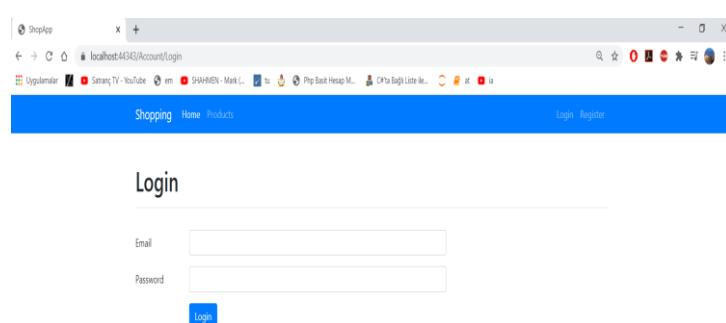
```



Şekildeki gibi ResetPassword view'ini de hazırlıyoruz.



Submit'e tıkladığımız zaman,



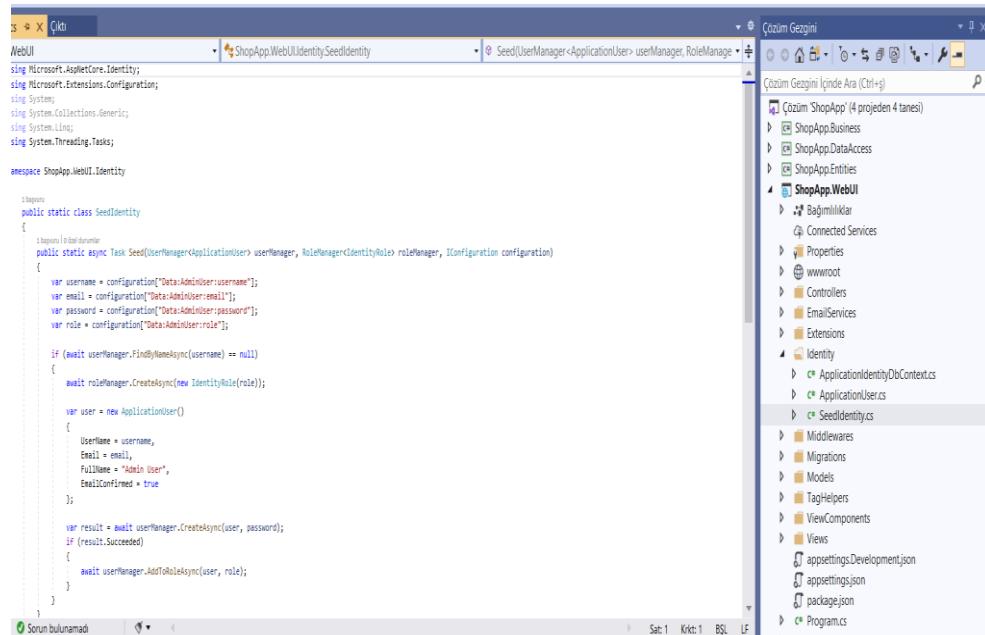
Bizi ana sayfaya yönlendiriyor. Mail adresine gelen bildiriye tıkladığımızda, şifre değiştirme sayfasının ekrana geldiğini görürüz.

Yapılan İş: Admin Kullanıcısı Yaratma

Tarih: 14/08/2020

Açıklama:

Admin yetkilerine sahip bir kullanıcı yetkilendirme işlemi yapmamız gerekiyor. Bunun için Identity klasörüne bir SeedIdentity adında bir sınıf oluşturuyoruz.



```
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

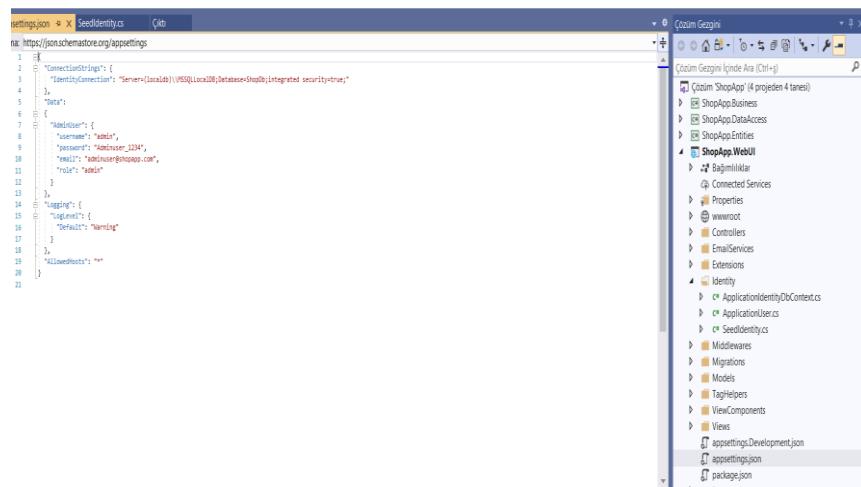
namespace ShopApp.WebUI.Identity

{
    public static class SeedIdentity
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA1801:AvoidUncalledPrivateMethods")]
        public static async Task Seed(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager, IConfiguration configuration)
        {
            var username = configuration["Data:AdminUser:username"];
            var email = configuration["Data:AdminUser:email"];
            var password = configuration["Data:AdminUser:password"];
            var role = configuration["Data:AdminUser:role"];

            if (await userManager.FindByNameAsync(username) == null)
            {
                await roleManager.CreateAsync(new IdentityRole(role));
                var user = new ApplicationUser()
                {
                    UserName = username,
                    Email = email,
                    FullName = "Admin User",
                    EmailConfirmed = true
                };

                var result = await userManager.CreateAsync(user, password);
                if (result.Succeeded)
                {
                    await userManager.AddToRoleAsync(user, role);
                }
            }
        }
    }
}
```

IConfiguration isminde bir interfaceye ihtiyacımız var. Çünkü appsettings içerisinde bir user oluşturacağız.



```
{
  "ConnectionStrings": {
    "IdentityConnection": "Server=(localdb)\\MSSQLLocalDB;Database=ShopDB;Integrated security=true"
  },
  "appSettings": {
    "AdminUser": {
      "username": "admin",
      "email": "adminuser_123@outlook.com",
      "password": "adminuser@shopapp.com",
      "role": "admin"
    }
  },
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

İlk resimde yazıldığı gibi, appsettingdeki bilgileri çekiyoruz. Eğer kullanıcı null ise, bilgileri gönderiyoruz.

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

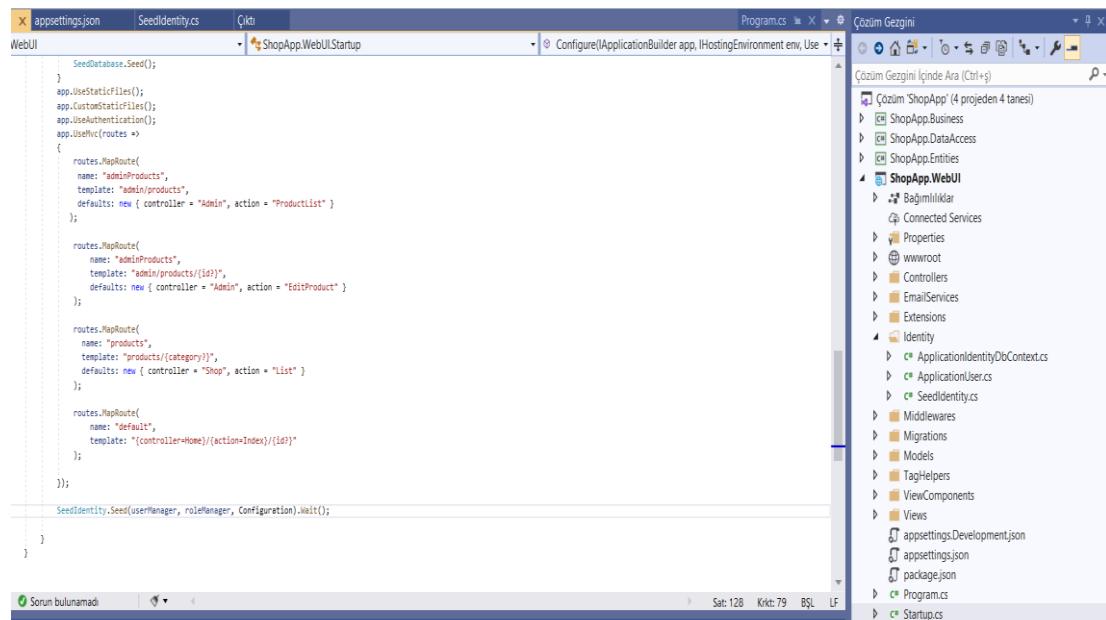
ADI VE SOYADI:

İMZASI:

Yapılan İş: Admin Kullanıcısı Yaratma

Tarih: 14/08/2020

Açıklama:



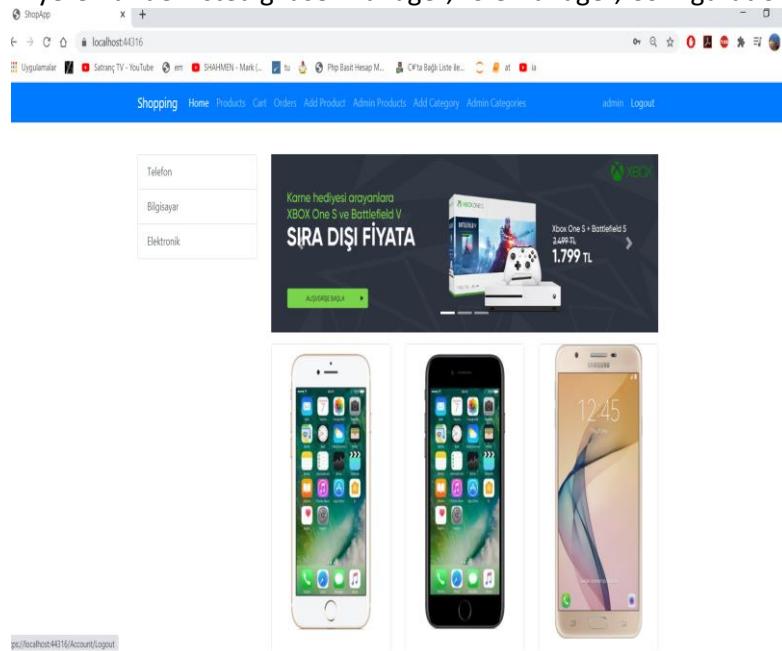
```
app.UseDatabase.Seed();
}
app.UseStaticFiles();
app.UseCustomStaticFiles();
app.UseAuthentication();
app.UseHvc(routes =>
{
    routes.MapRoute(
        name: "adminProducts",
        template: "admin/products/{id}",
        defaults: new { controller = "Admin", action = "ProductList" }
    );

    routes.MapRoute(
        name: "products",
        template: "products/{category}",
        defaults: new { controller = "Shop", action = "List" }
    );

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

SeedIdentity.Seed(userManager, roleManager, Configuration).Wait();
}
```

Projenin startup dosyasında da
SeedIdentity.Seed(userManager, roleManager, Configuration).Wait();
Diyerek bizden istediği userManager, roleManager, Configurationları veriyoruz.



Default olarak oluşturduğumuz admin bilgilerini girdiğimizde admin kullanıcıı olarak giriş yapmış oluyoruz.

43

**ÖĞRENCİNİN:
İMZASI: Hamdi Uraz Alkış**

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Açıklama:

Yaratılan admin kullanıcı modelinin, admin rolüne sahip işlemleri yapabilmesini sağlamalıyız.
Öncelikle AdminController içindeki Authorize ifadesini [Authorize(Roles = "admin")] olarak değiştirmemiz gerekiyor.



```
_navbar.cshtml
AdminController.cs
1 <nav class="navbar navbar-expand-lg navbar-dark bg-primary fixed-top">
2   <div class="container">
3     <a class="navbar-brand" href="/">Shopping</a>
4     <ul class="navbar-nav mr-auto">
5       <li class="nav-item">
6         <a href="#" class="nav-link active">Home</a>
7       </li>
8       <li class="nav-item">
9         <a asp-controller="Shop" asp-action="List" class="nav-link">Products</a>
10      </li>
11    </ul>
12    <ul class="navbar-nav ml-auto" >
13      <li>
14        <small>If (User.Identity.IsAuthenticated)</small>
15        <ul class="nav-item">
16          <li> <a href="#" class="nav-link">Cart</a>
17        </li>
18        <li> <a href="#" class="nav-link">Orders</a>
19      </li>
20      <li>
21        <small>If (User.IsInRole("admin"))</small>
22        <ul class="nav-item">
23          <li> <a asp-controller="Admin" asp-action="CreateProduct" class="nav-link">Add Product</a>
24        </li>
25        <li> <a asp-controller="Admin" asp-action="ProductList" class="nav-link">Admin Products</a>
26      </li>
27      <li>
28        <small>If (User.IsInRole("Admin" asp-action="CreateCategory" class="nav-link">Add Category</a>
29      </li>
30      <li>
31        <small>If (User.IsInRole("Admin" asp-action="CategoryList" class="nav-link">Admin Categories</a>
32      </li>
33    </ul>
34  </li>
35 </ul>
36 </div>
37 </div>
38 <ul class="navbar-nav ml-auto" >
39   <li>
40     <small>If (User.Identity.IsAuthenticated)</small>
```

Navigationbar sayfasının güncellemesinde de istediğimiz alanları eğer kullanıcı admin rolüne sahipse, if (User.IsInRole("admin")) diyerek koşul ifadesinin içine alıyoruz.

ÖĞRENCİNİN: İMZASI: Hamdi Uraz Alkış	KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:
---	---

17/08/2020 Tarihinden 20/08/2020 Tarihine Kadar Bir Haftalık Çalışma Tablosu				
Tarih	Gün	YAPILAN İŞLER	Sayfa No	Çalışılan Süre (Saat)
17/08/2020	Pazartesi	Alışveriş Sepeti Ekleme	45/48	9
18/08/2020	Salı	Cart View Oluşturma	48/51	9
19/08/2020	Çarşamba	Kart İtemlerini Görüntüleme	52/53	9
20/08/2020	Perşembe	Karta İtem Ekleme Ve Silme	54/57	9
.../.../...	Cuma		/	-
.../.../...	Cumartesi		.../....	
TOPLAM SÜRE (Saat)				36
Öğrencinin		Kurum Yetkilisi		
Adı SOYADI: Hamdi Uraz Alkış İmzası: Hamdi Uraz Alkış Çalıştığı Bölüm: Bilişim Teknolojileri		Adı SOYADI: Unvanı: İmza/Kaşesi:		

Açıklama:

Alışveriş sepeti oluştururken, verileri databasede mi yoksa session olarak mı saklayacağımız buna karar vermemiz gerekiyor. Eğer kullanıcı login değilse, eklediği ürünleri geçici olarak session ortamında saklayabiliriz. Kullanıcı login olmuşsa bilgileri session ortamından database' e aktarırız. Öncelikli olarak entity katmanına iki adet sınıf eklememiz gerekiyor. Birtanesi cart sınıfı ve

```

Entites
using System;
using System.Collections.Generic;
using System.Text;
namespace ShopApp.Entities
{
    public class Cart
    {
        public int Id { get; set; }
        public string UserId { get; set; }
        public List CartItems { get; set; }
    }
}

```

Göründüğü gibi karta ait olan id, userid ve o kartın sahip olduğu ürünleri temsil eden CartItems değişkenlerini belirliyoruz.

```

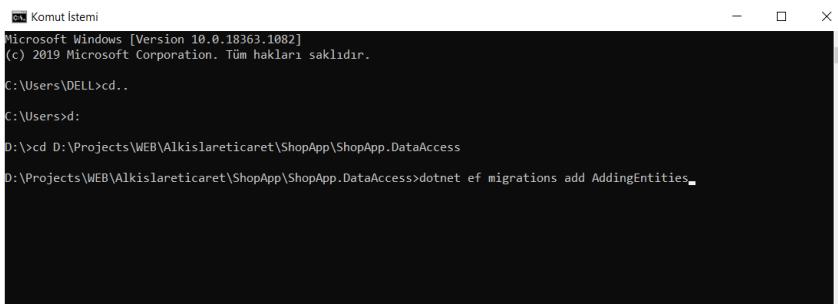
CartItems
using System;
using System.Collections.Generic;
using System.Text;
namespace ShopApp.Entities
{
    class CartItem
    {
        public int Id { get; set; }
        public Product Product { get; set; }
        public int ProductId { get; set; }
        public Cart Cart { get; set; }
        public int CartId { get; set; }
        public int Quantity { get; set; }
    }
}

```

CartItem sınıfında ise belirtildiği gibi değişkenleri tanımlıyoruz. Her kartta bir tane ürün olabilir eğer kart sayısı artiyorsa demek ki ürün sayısını da artırmamız gerekiyor.

Açıklama:

Oluşturmuş olduğumuz sınıflardaki yapıları database'e aktarmamız gerekiyor fakat bunun öncesi Hazırladığımız entity'i ShopContext içinde `public DbSet<Cart> Carts { get; set; }` diyerek tanımlamamız gerekiyor.



```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. Tüm hakları saklıdır.

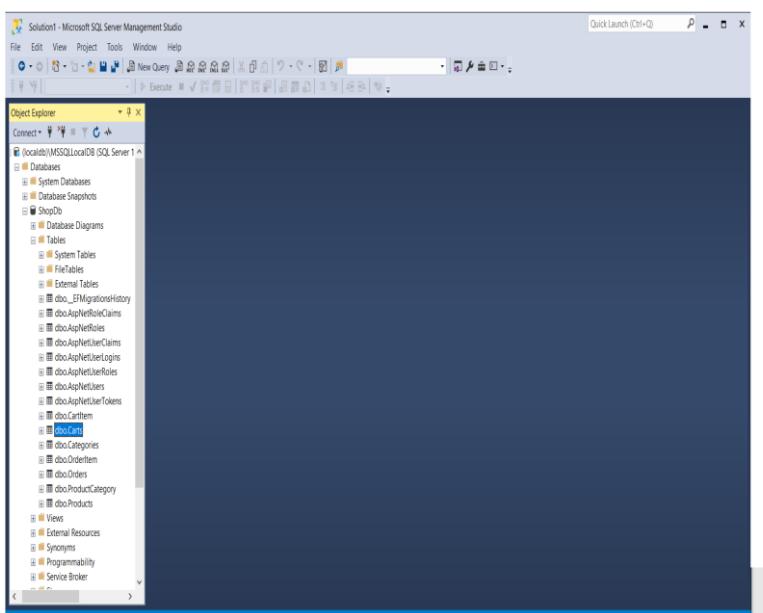
C:\Users\DELL>cd..

C:\Users>d:

D:\>cd D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.DataAccess

D:\Projects\WEB\Alkislareticaret\ShopApp\ShopApp.DataAccess>dotnet ef migrations add AddingEntities
```

Diyerek bir migrations ekliyoruz. Daha sonra dotnet ef database update diyerek veritabanını güncelliyoruz. Veritabanını kontrol ettiğimizde iki alanında veritabanı tablolarına eklendiğini görürüz.



Şimdi yeni bir kullanıcı oluşturduğumuz zaman, bu kullanıcıya ait olan bir kart kaydı olması gerekiyor. Onu AccountController üzerinde ekleyeceğiz fakat şimdi kart üzerinden bir işlem yapmak için business katmanına cart servisi ekleyelim.

ÖĞRENCİNİN: İMZASI: Hamdi Uraz Alkış	KURUM SORUMLUSUNUN: ADI VE SOYADI: İMZASI:
---	---

```

Service.cs  ✘ ICategoriesService.cs AccountController.cs
ShopApp.Business
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4
5  namespace ShopApp.Business.Abstract
6  {
7      public interface ICartService
8      {
9          void InitializeCart(string userId);
10     }
11 }

```

Su anlık kart kaydı oluşturmak için bir method oluşturalım. İşlemin Concrete sayfasında CartManager sınıfı da hazırlayalım.

```

CartManager.cs  ✘ ICartService.cs CategoryService.cs AccountController.cs
ShopApp.Business
1  using ShopApp.Business.Abstract;
2  using ShopApp.DataAccess;
3  using ShopApp.Entities;
4  using System;
5  using System.Collections.Generic;
6  using System.Text;
7
8  namespace ShopApp.Business.Concrete
9  {
10     public class CartManager : CartService
11     {
12         private _CartDal _cartDal;
13         public CartManager(ICartDal cartDAL)
14         {
15             _cartDAL = cartDAL;
16         }
17         public void InitializeCart(string userId)
18         {
19             _cartDAL.Create(new Cart() { UserId = userId });
20         }
21     }
22 }

```

Tabii ki sonrasında ICartDal interfacayı DataAccess katmanında Abstract katmanında oluşturuyoruz. Aynı şekilde eforeCartDal sınıfını da, Concrete bölümünde oluşturuyoruz.

```

AccountController.cs
ShopApp.WebUI
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Http;
6  using Microsoft.AspNetCore.Identity;
7  using Microsoft.AspNetCore.Authentication;
8  using ShopApp.Business.Abstract;
9  using ShopApp.WebUI.Extensions;
10 using ShopApp.WebUI.Identity;
11 using ShopApp.Model.Models;
12
13 [AutoValidateAntiforgeryToken]
14 [Authorize]
15 public class AccountController : Controller
16 {
17     private UserManager<ApplicationUser> _userManager;
18     private SignInManager<ApplicationUser> _signInManager;
19     private IEmailSender _emailSender;
20     private ICartService _cartService;
21
22     public AccountController(ICartService cartService, UserManager< ApplicationUser> userManager, SignInManager< ApplicationUser> signInManager, IEmailSender emailSender)
23     {
24         _cartService = cartService;
25         _userManager = userManager;
26         _signInManager = signInManager;
27         _emailSender = emailSender;
28     }
29 }

```

Account Controller sayfasında oluşturduğumuz ICartServiceden bir nesne türetip ilgili alanlara ekliyoruz.

Açıklama:

```

136     public async Task<IActionResult> ConfirmEmail(string userId, string token)
137     {
138         if (userId == null || token == null)
139         {
140             TempData.Put("message", new ResultMessage()
141             {
142                 Title = "Hesap Onayı",
143                 Message = "Hesap onayı için bilgileriniz yanlış",
144                 Css = "danger"
145             });
146 
147             return Redirect("~/");
148         }
149 
150         var user = await _userManager.FindByIdAsync(userId);
151         if (user != null)
152         {
153             var result = await _userManager.ConfirmEmailAsync(user, token);
154             if (result.Succeeded)
155             {
156                 // create cart object
157                 _cartService.InitializeCart(user.Id);
158 
159                 TempData.Put("message", new ResultMessage()
160                 {
161                     Title = "Hesap Onayı",
162                     Message = "Hesabınız başarıyla onaylanmıştır.",
163                     Css = "success"
164                 });
165 
166                 return RedirectToAction("Login");
167             }
168         }
169 
170         TempData.Put("message", new ResultMessage()
171         {
172             Title = "Hesap Onayı",
173             Message = "Hesabınız onaylanmadı."
174         });
175     }

```

`_cartService.InitializeCart(user.Id);` koduyla, mail onayından sonra kullanıcı için otomatik bir kart kaydı oluşturacak.

Son olarak webUI katmanındaki startup dosyasının ConfigureServices methodunda bu servisleri tanımlıyoruz.

```

    services.ConfigureApplicationCookie(options =>
    {
        options.LoginPath = "/account/login";
        options.LogoutPath = "/account/logout";
        options.AccessDeniedPath = "/account/accessdenied";
        options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
        options.SlidingExpiration = true;
        options.Cookie = new CookieBuilder()
        {
            HttpOnly = true,
            Name = "ShopApp.Security.Cookie",
            SameSite = SameSiteMode.Strict
        };
    });

    services.AddScoped<IProductDal>, EfCoreProductDal();
    services.AddScoped<ICategoryDal>, EfCoreCategoryDal();
    services.AddScoped<ICartDal>, EfCoreCartDal();

    services.AddScoped<IProductService>, ProductManager();
    services.AddScoped<ICategoryService>, CategoryManager();
    services.AddScoped<ICartService>, CartManager();

    services.AddTransient<IEmailSender>, EmailSender();

    services.AddMvc().SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_2_2);

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    app.UseRouting();
    app.UseAuthorization();
    public void Configure(IApplicationBuilder app, IHostingEnvironment env, UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager)
    {
        if (env.IsDevelopment())
        {
            ...
        }
    }
}

```

Açıklama:

İlk olarak login olmayan kullanıcının add to cart butonunu görmemesi gerekiyor.Daha önceden bahsettiğimiz gibi,login olmayan kullanıcı için session da işlemleri tutabiliriz.

The screenshot shows the Visual Studio interface. On the left is the code editor with the file `_product.cshtml` containing ASP.NET Core Razor code. On the right is the Solution Explorer showing the project structure for a Shop application, including `Products`, `ProductCategory.cs`, `ShopApp.WebUI`, and `ShopController.cs`.

```

<div class="col-md-4">
    <div class="card mb-3">
        
        <div class="card-body">
            <h4 class="card-title">
                @Html.DisplayFor(m => m.Name)
            </h4>
            <small class="text-muted">
                @Html.DisplayFor(m => m.Price)
            </small>
        </div>
        <div class="card-footer text-center">
            <form asp-controller="Cart" asp-action="AddToCart" method="post" style="display:inline">
                <input type="hidden" name="productId" value="@Model.ID" />
                <input type="hidden" name="quantity" value="1" />
                <button type="submit" class="btn btn-outline-primary btn-sm ml-3">Add To Cart</button>
            </form>
        </div>
    </div>
</div>

```

Şekilde belirtildiği gibi add to cart butonunu login işlemi kontrollüne alıyoruz.Ayrıca ürün miktarını bildirmek için bir quantity alanı oluşturuyoruz.

The screenshot shows the Visual Studio interface. On the left is the code editor with the file `ProductDetail.cshtml` containing ASP.NET Core Razor code. On the right is the Solution Explorer showing the project structure for a Shop application, including `Products`, `ProductCategory.cs`, `ShopApp.Business`, `ShopApp.DataAccess`, and `ShopApp.WebUI`.

```

<div class="row">
    <div class="col-md-3">
        
        <div class="text-center">
            @Model.Product.Name
        </div>
        <foreach var="category" in Model.Categories>
            <a href="#" class="btn btn-link p-0 mb-3">@category.Name</a>
        </foreach>
        <div class="mb-3">
            <h4 class="text-primary mb-3">
                @Model.Product.Price<br/>
            </h4>
            <form asp-controller="Cart" asp-action="AddToCart" method="post" class="form-inline">
                <input type="hidden" name="productId" value="@Model.Product.ID" />
                <div class="input-group mb-3">
                    <input type="number" name="quantity" class="form-control" value="1" min="1" step="1" style="width:100px;" />
                    <div class="input-group-append">
                        <button type="button" class="btn btn-primary">Add To Cart</button>
                    </div>
                </div>
            </form>
        </div>
    </div>
    <div class="col-md-9">
        Sorun bulunamadı
    </div>
</div>

```

Detay sayfasını da şekildeki gibi güncelliyoruz.Son olarak CartController sınıfını oluşturmamız gerekiyor.

Yapılan İş: Cart View Oluşturma

Tarih: 18/08/2020

Açıklama:

The screenshot shows the Visual Studio interface. In the center, there is a code editor window displaying the `CartController.cs` file. The code defines a controller that inherits from `Controller`. It contains two actions: `Index()` and `AddToCart()`. The `Index()` action returns a view. The `AddToCart()` action is annotated with `[HttpPost]` and `[ValidateAntiForgeryToken]`, and it also returns a view. To the right of the code editor is the 'Çözüm Gezgini' (Solution Explorer) window, which lists the project structure. The `Controllers` folder contains `AccountController.cs`, `AdminController.cs`, `CartController.cs`, `HomeController.cs`, and `ShopController.cs`.

```
ShopApp.WebUI
  Controllers
    AccountController.cs
    AdminController.cs
    CartController.cs
    HomeController.cs
    ShopController.cs
```

Kontroller sayfasında da index methodu bize kullanıcı kart verilerini,addtocart ile de bilgi ekleyeceğiz.

Öncelikle ICartService interfacesinde GetCartByUserId adında ve cart türünde bir method oluşturuyoruz ve bu methodu cart manager içinde de implemente ediyoruz.

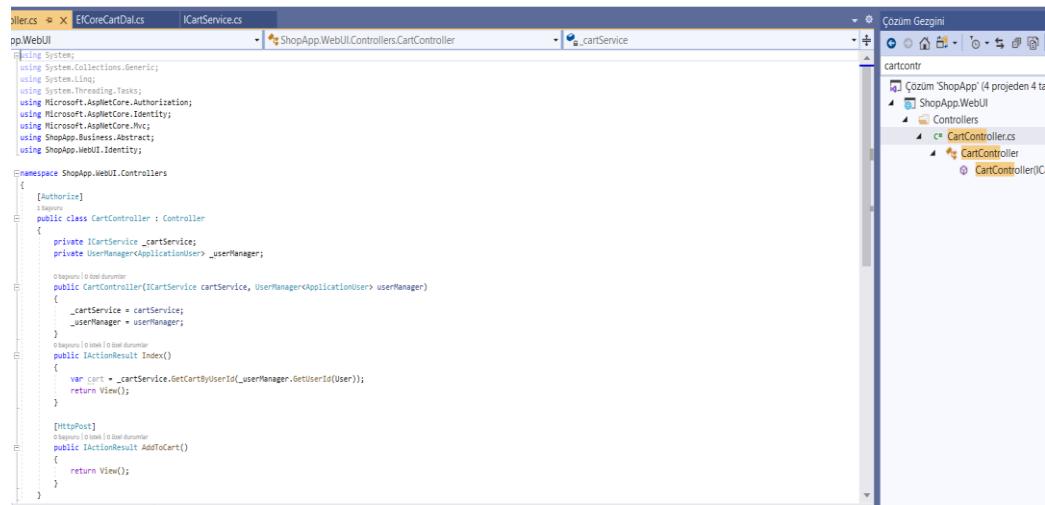
The screenshot shows the Visual Studio interface. On the left, there is a code editor window for `eCartDal.cs`, which is part of the `ShopApp.DataAccess` namespace. It contains a class `EFCoreCartDal` that implements `ICartDal`. The `GetByUserId(string userId)` method uses a `using` statement to create a new `ShopContext` instance. It then queries the database to find a cart item where the user ID matches the provided `userId`. On the right, there is another code editor window for `CartServices.cs`, which is part of the `ShopApp.Business` namespace. It contains a method `GetCartByUserId(string userId)` that calls the `GetByUserId` method from `eCartDal`. Below the code editors is the 'Çözüm Gezgini İçinde Ara (Ctrl+F)' (Solution Explorer Search) window, which shows the search results for 'Cart' across all projects.

```
eCartDal.cs
ShopApp.DataAccess
  EFCoreCartDal.cs
    public Cart GetByUserId(string userId)
    {
        using (var context = new ShopContext())
        {
            return context
                .Carts
                .Include(i => i.CartItems)
                .Include(i => i.Product)
                .FirstOrDefault(i => i.UserId == userId);
        }
    }
}

CartServices.cs
ShopApp.Business
  ICartService.cs
  CartManager.cs
    public Cart GetCartByUserId(string userId)
    {
        var cart = eCartDal.GetByUserId(userId);
    }
}
```

Implemente işlemini `efcoreCartDal` altında da yaptığımızda oluşturduğumuz `GetCartByUserId` methodu gelmiş oluyor. `Context` üzerinden `include` methodunu kullanarak kartlara gidiyoruz ve kart itemlerini alıyoruz.

Açıklama:



The screenshot shows the Visual Studio IDE. On the left, the code editor displays the `CartController.cs` file. The code defines a controller that injects `ICartService` and `IUserManager< ApplicationUser >`. It contains two actions: `Index` (GET) which retrieves the cart for the current user, and `AddToCart` (POST) which adds items to the cart. On the right, the Solution Explorer shows the project structure under `ShopApp.WebUI`, specifically the `Controllers` folder containing the `CartController.cs` file.

```

    public class CartController : Controller
    {
        private ICartService _cartService;
        private UserManager< ApplicationUser > _userManager;

        public CartController(ICartService cartService, UserManager< ApplicationUser > userManager)
        {
            _cartService = cartService;
            _userManager = userManager;
        }

        public IActionResult Index()
        {
            var cart = _cartService.GetCartByUserId(_userManager.GetUserId(User));
            return View();
        }

        [HttpPost]
        public IActionResult AddToCart()
        {
            return View();
        }
    }
  
```

CartController üzerinde inject işlemi yapıyoruz ve index içinde userid alma işlemi yapıyoruz.

Yapılan İş: Kart İtemlerini Görüntüleme

Tarih: 19/08/2020

Açıklama:

İlk olarak kart için bir görünüm oluşturuyoruz. Ve bu view'ın kullanacağı kart modelini models klasöründe oluşturuyoruz.

The screenshot shows the Visual Studio IDE. In the top-left, there's a code editor window titled "CartController.cs". The code defines two classes: "CartModel" and "CartItemModel". The "CartModel" class has properties for CartId, CartItems, and CartTotalPrice. The "CartItemModel" class has properties for CartItemId, ProductId, Name, Price, and Quantity. Below the code editor is a status bar showing "Sorum bulunamadı" and some other project details. To the right of the code editor is the "Solution Explorer" window, which lists the project structure under "ShopApp.WebUI". The "Models" folder is expanded, showing "CartModel.cs" and "CartItemModel.cs".

```
1  #nullable enable
2  #region namespaces
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Threading.Tasks;
7  #endregion namespaces
8
9  namespace ShopApp.WebUI.Models
10
11     public class CartModel
12     {
13         private int? cartId;
14
15         public int? CartId { get; set; }
16
17         private List cartItems;
18
19         public List CartItems { get; set; }
20     }
21
22
23     public class CartItemModel
24     {
25         private int? cartItemId;
26
27         public int? CartItemId { get; set; }
28
29
30         private int? productId;
31
32         public int? ProductId { get; set; }
33
34         private string name;
35
36         public string Name { get; set; }
37
38         private decimal price;
39
40         public decimal Price { get; set; }
41
42         private int? quantity;
43
44         public int? Quantity { get; set; }
45     }
46
47 
```

The screenshot shows the Visual Studio IDE. In the top-left, there's a code editor window titled "Index.cshtml". The code is an ASP.NET MVC view. It starts with a title "Cart" and then a table. The table has a header row with columns "Name", "Price", and "Quantity". Below the header, there's a loop that iterates over "model.CartItems". For each item, it displays the product name, price, and quantity. If the quantity is less than 1, it adds a class "text-danger". There are also "Delete" and "Edit" buttons for each item. Below the table, there's a "Create New Item" button. Below that is another table with a single row containing a "Create New Item" button. To the right of the code editor is the "Solution Explorer" window, showing the project structure under "ShopApp.WebUI". The "Models" folder is expanded, showing "CartModel.cs" and "CartItemModel.cs".

```
1  #nullable enable
2  #region namespaces
3  using System;
4  using System.Collections.Generic;
5  using System.Linq;
6  using System.Threading.Tasks;
7  #endregion namespaces
8
9  #region ShoppingCart
10
11  <div class="row">
12      <div class="col-12">
13          <table class="table table-bordered">
14              <thead>
15                  <tr>
16                      <th>Name</th>
17                      <th>Price</th>
18                      <th>Quantity</th>
19                  </tr>
20              </thead>
21              <tbody>
22                  <tr>
23                      <td>@foreach (var item in Model.CartItems)
24                      <td>@item.Price.ToString("C")</td>
25                      <td>@item.Quantity</td>
26                      <td>@item.Quantity.ToString("N")</td>
27
28                      <td><a href="#" class="text-decoration-none text-danger" style="color: red;">Delete</td>
29                      <td><a href="#" class="text-decoration-none" style="color: blue;">Edit</td>
30                  </tr>
31              </tbody>
32          </table>
33      </div>
34  </div>
35 
```

52

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Kart İtemlerini Görüntüleme

Tarih: 19/08/2020

Açıklama:

Oluşturdugumuz model'i CartController üzerinde kullanıyoruz.

Daha sonra kart için bir view oluşturuyoruz.

The screenshot shows the Microsoft Visual Studio IDE interface. The left pane displays the `CartController.cs` file, which contains the code for a shopping cart controller. The right pane shows the **Solution Explorer** with the project structure:

- Çözüm Gezgini** (Solution Explorer)
 - ShopController.cs
 - EmailServices
 - Extensions
 - Identity
 - Middlewares
 - Migrations
 - Models
 - CartModel.cs
 - CategoryListViewModel.cs
 - CategoryListModel.cs
 - LoginModel.cs
 - ProductDetailsModel.cs
 - ProductListModel.cs
 - ProductModel.cs
 - RegisterModel.cs
 - ResetPasswordModel.cs
 - ResultMessage.cs
 - TagHelpers
 - ViewComponents
 - Views
 - Account
 - Admin
 - Cart
 - Index.cshtml
 - Home
 - Shared
 - Shop
 - ViewImports.cshtml

The `Index.cshtml` file under the `Cart` view is currently selected in the Solution Explorer.

53

ÖĞRENCİNİN;

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

Açıklama:

```

public class CartController : Controller
{
    private ICartService _cartService;
    private ICartManager _userManager;

    [HttpGet]
    public ActionResult Index()
    {
        var cart = _cartService.GetCartByUserId(_userManager.GetCurrentUser());
        if (cart != null)
        {
            CartItem cartItem;
            foreach (var item in cart.CartItems)
            {
                cartItem = new CartItemModel()
                {
                    CartId = item.Id,
                    ProductId = item.Product.Id,
                    Name = item.Product.Name,
                    Price = item.Product.Price,
                    ImageUrl = item.Product.Image,
                    Quantity = item.Quantity
                };
                cartItem.TotalPrice = cartItem.Price * cartItem.Quantity;
                cartItems.Add(cartItem);
            }
        }
        return View("Index");
    }

    [HttpPost]
    public ActionResult AddToCart(int productId, int quantity)
    {
        _userManager.AdminUserManager.GetSerial();
        _cartService.AddToCart(productId, quantity);
        return RedirectToAction("Index");
    }
}

```

The screenshot shows the Visual Studio IDE with the CartController.cs file open in the editor. The controller inherits from Controller and uses two services: ICartService and ICartManager. It has an Index action that retrieves the cart for the current user and displays its items. It also has an AddToCart action that adds a product to the cart.

CartController sınıfının addtocart methoduna product id ve miktarı belirtecek quantity adında iki parametre ekliyoruz. ICartService'de ise geri dönüş değeri olmayan void AddToCart(string userId, int productId, int quantity); methodunu da ekliyoruz. CartManager bölümündeyse ICartService de oluşturduğumuz methodları implemente ediyoruz.

```

public class CartManager : ICartService
{
    private ICartDal _cartDal;
    private ICartManager _cartManager;

    public CartManager(ICartDal cartDal)
    {
        _cartDal = cartDal;
    }

    public void AddToCart(string userId, int productId, int quantity)
    {
        var cart = GetCartByUserId(userId);
        if (cart != null)
        {
            var index = cart.CartItems.FindIndex(i => i.ProductId == productId);

            if (index > -1)
            {
                cart.CartItems[index].Quantity += quantity;
                cart.TotalPrice += quantity * cart.CartItems[index].Price;
            }
            else
            {
                cart.CartItems.Add(new CartItem()
                {
                    ProductId = productId,
                    Quantity = quantity,
                    TotalPrice = quantity * cart.CartItems[0].Price
                });
            }
        }
        else
        {
            cart = new Cart();
            cart.TotalPrice = 0;
            cart.CartItems.Add(new CartItem()
            {
                ProductId = productId,
                Quantity = quantity,
                TotalPrice = quantity * cart.CartItems[0].Price
            });
        }

        _cartDal.Update(cart);
    }
}

```

The screenshot shows the CartManager.cs file in the ShopApp.Business project. It implements the ICartService interface. The AddToCart method checks if a cart exists for the given user ID. If it does, it finds the index of the item with the same product ID and updates its quantity. If the item doesn't exist, it adds a new one. Finally, it updates the cart in the database using the _cartDal object.

Addtocart methodunda, ilk olarak kullanıcının bir kartı var mı bunu kontrol etmemiz gerekiyor. Sonra findindex ile kayıt yoksa yeni kart ekliyoruz. Eğer kart varsa o indexteki elemanın, kullanıcının belirttiği miktar kadar değerini artırıyoruz. Sonra _cartDal üzerinden update methodunu çağırıyoruz.

Açıklama:

The screenshot shows the Visual Studio interface. The Solution Explorer on the right lists projects like 'ShopApp.Business', 'ShopApp.DataAccess', and 'ShopApp.WebUI'. The code editor on the left contains the 'CartManager.cs' file, which includes imports for EntityFrameworkCore, ShopApp.DataAccess.Concrete.EFCore.EFCoreCartDal, and System. The class 'EfCoreCartDal' is defined as a concrete implementation of 'ICartRepository'. It overrides the 'Update(Cart entity)' method to update the database context and save changes. It also includes a 'GetUserId(string userId)' method that returns a user's cart from the database.

```

    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using ShopApp.DataAccess.Concrete.EFCore;

    namespace ShopApp.DataAccess.Concrete.EFCore
    {
        public class EfCoreCartDal : ICartRepository
        {
            private readonly ShopContext _context;
            private readonly ICartManager _cartManager;
            private readonly ICategoryManager _categoryManager;
            private readonly IProductManager _productManager;

            public EfCoreCartDal(ShopContext context, ICartManager cartManager, ICategoryManager categoryManager, IProductManager productManager)
            {
                _context = context;
                _cartManager = cartManager;
                _categoryManager = categoryManager;
                _productManager = productManager;
            }

            public void Update(Cart entity)
            {
                using (var content = new ShopContent())
                {
                    content.Carts.UpdateEntity(entity);
                    content.SaveChanges();
                }
            }

            public Cart GetUserId(string userId)
            {
                using (var content = new ShopContent())
                {
                    return content
                        .Carts
                        .Include(i => i.CartItems)
                        .ThenInclude(i => i.Product)
                        .FirstOrDefault(i => i.UserId == userId);
                }
            }
        }
    }
  
```

EfCoreCartDal üzerinde de ilişkilidataları update yapmak için methodu override ile hazırlıyoruz.

CartController üzerinde de AddToCart methoduna

```

    _cartService.AddToCart(_userManager.GetUserId(User), productId, quantity);
    return RedirectToAction("Index");
  
```

Ekleyleerek, uygulamayı kart sayfasına yönlendiriyoruz.

Ürün silme işlemleri için, sayfa tasarımdındaki gerekli butonu ekledikten sonra, AddtoCart işleminde olduğu gibi busefer DeleteToCart methodu oluşturuyoruz.

The screenshot shows the Visual Studio interface. The Solution Explorer on the right shows the 'CartController' under the 'Controllers' folder. The code editor on the left contains the 'CartController.cs' file. It has three actions: 'Index', 'AddToCart', and 'DeleteFromCart'. The 'AddToCart' action takes a product ID and quantity, adds the item to the cart using '_cartService.AddToCart(_userManager.GetUserId(user), productId, quantity);', and then redirects to the 'Index' view. The 'DeleteFromCart' action takes a product ID and removes the item from the cart using '_cartService.DeleteFromCart(_userManager.GetUserId(user), productId);', and then redirects to the 'Index' view.

```

    [HttpGet]
    public IActionResult Index()
    {
        var products = _productManager.GetProducts();
        var cartItems = _cartManager.GetCartItems();
        var cart = new Cart
        {
            Name = _userManager.GetUserName(),
            Price = (decimal)_product.Price,
            ImageUrl = _product.ImageUrl,
            Quantity = _product.Quantity
        };
        return View(cart);
    }

    [HttpPost]
    public IActionResult AddToCart(int productId, int quantity)
    {
        _cartService.AddToCart(_userManager.GetUserId(user), productId, quantity);
        return RedirectToAction("Index");
    }

    [HttpPost]
    public IActionResult DeleteFromCart(int productId)
    {
        _cartService.DeleteFromCart(_userManager.GetUserId(user), productId);
        return RedirectToAction("Index");
    }
  
```

Yapılan İş: Karta İtem Ekleme Ve Silme

Tarih: 20/08/2020

Açıklama:

Bunu yazdığımızda, ICartService üzerinde de gerekli delete methodunu implemente ederek oluşturuyoruz. Aynı işlemi cartManager üzerinde de implemente ederek methodu getiriyoruz.

```

    CartController.cs
    ICartService.cs
    ShopApp.Business
    CartManager.cs
    _cartDal

    24
    25
    26
    27
    28
    29
    30
    31
    32
    33
    34
    35
    36
    37
    38
    39
    40
    41
    42
    43
    44
    45
    46
    47
    48
    49
    50
    51
    52
    53
    54
    55
    56
    57
    58
    59
    60
    61
  
```

DeleteFromCart'da belirtildiği gibi ürün id ve kullanıcı id bilgileriyle, veriyi alıyoruz.

EfCoreCartDal üzerinde,

```

    EfCoreCartDal.cs
    CartController.cs
    ICartService.cs
    ShopApp.DataAccess
    CartManager.cs
    _cartDal

    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
    11
    12
    13
    14
    15
    16
    17
    18
    19
    20
    21
    22
    23
    24
    25
    26
    27
    28
    29
    30
    31
    32
    33
    34
    35
    36
    37
    38
    39
    40
    41
    42
  
```

Yeni bir shopContext oluşturup, silme işlemi için yeni bir sql sorgusu yazıyoruz.

56

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

Yapılan İş: Karta İtem Ekleme Ve Silme

Tarih: 20/08/2020

Açıklama:

Cart Items				Cart Summary	
Product Name	Price	Quantity	Total	Cart Total	18.000,00 ₺
Samsung S5	2.000,00 ₺	4	8.000,00 ₺		
Samsung S6	3.000,00 ₺	2	6.000,00 ₺		
Samsung S7	4.000,00 ₺	1	4.000,00 ₺		

Continue Shopping Checkout

Shopping Cart

Cart Items

Product Name	Price	Quantity	Total	
Samsung S5	2.000,00 ₺	4	8.000,00 ₺	

Cart Summary

Cart Total	8.000,00 ₺
Shipping	Free
Order Total	8.000,00 ₺

Continue Shopping Checkout

Şekilde görüldüğü gibi butona tıkladığımızda, belirtilen ürünü silebiliyoruz. Aynı şekilde product sayfasında ürün ekleme işlemi yaptığımızda, ürünlerin karta eklendiğini görürüz.

ÖĞRENCİNİN:

İMZASI: Hamdi Uraz Alkış

KURUM SORUMLUSUNUN:

ADI VE SOYADI:

İMZASI:

