Bu uygulamanın amacı, çeşitli ürünlerin bilgileriyle birlikte listelenmesi ve filtreleme ile arama yapmak olacak.

// kategori sınıfı

// kategorilerin id ve adını tutuyor

public class Category

{

public int CategoryId { get; set; }

public string? Name { get; set; } = string.Empty;

}

// ürün sınıfı

// ürünlerin id, ad, fiyat, resim, aktiflik durumu ve kategori id'sini tutuyor

// kategori id'si ile kategorilerle ilişkilendirilecek

// aktiflik durumu ile ürünlerin aktif olup olmadığını belirleyeceğiz vs.

public class Product

{

public int ProductId { get; set; }

public string? Name { get; set; } = string.Empty;

public int Price { get; set; }

public string Image { get; set; } = string.Empty;

public bool IsActive { get; set; }

public int CategoryId { get; set; }

}

// Veri tabanı yerine geçiyor

// Repository Türkçe’de depo anlamına gelir.

public class Repository

{

// ürünlerin ve kategorilerin listesi

// ürünleri ve kategorileri buradan ekleyeceğiz, ve burayı geçici veri tabanıymış gibi kullanacağız

private static readonly List<Product> \_products = new();

private static readonly List<Category> \_categories = new();

// ürünlerin ve kategorilerin eklenmesini bu kurucu metotta yapacağız

static Repository()

{

\_categories.Add(new Category { CategoryId = 1, Name = "Telefon" });

\_categories.Add(new Category { CategoryId = 2, Name = "Bilgisayar" });

\_products.Add(new Product { ProductId = 1, Name = "iPhone 14", Price = 2000, Image = "1.jpg", IsActive = true, CategoryId = 1 });

\_products.Add(new Product { ProductId = 2, Name = "iPhone 15", Price = 3000, Image = "2.jpg", IsActive = true, CategoryId = 1 });

\_products.Add(new Product { ProductId = 3, Name = "Samsung S7", Price = 4000, Image = "3.jpg", IsActive = true, CategoryId = 1 });

\_products.Add(new Product { ProductId = 4, Name = "Samsung S8", Price = 5000, Image = "4.jpg", IsActive = true, CategoryId = 1 });

\_products.Add(new Product { ProductId = 6, Name = "Macbook Air", Price = 7000, Image = "5.jpg", IsActive = true, CategoryId = 2 });

\_products.Add(new Product { ProductId = 7, Name = "Macbook Pro", Price = 8000, Image = "6.jpg", IsActive = true, CategoryId = 2 });

}

// ürünlerin çağırılması için oluşturduğumuz metot

public static List<Product> Products

{

get { return \_products; }

}

// kategorilerin çağırılması için oluşturduğumuz metot

public static List<Category> Categories

{

get { return \_categories; }

}

}

public IActionResult Index()

{

//Ürün listesi View sayfasına gönderiliyor

return View(Repository.Products);

}

<tr>

@\* İlgili Product classında yer alan Display(Name) özelliği kullanılarak tablo başlıkları doğrudan buraya geliyor \*@

<th>@Html.DisplayNameFor(m => m.ProductId)</th>

<th>@Html.DisplayNameFor(m => m.Name)</th>

<th>@Html.DisplayNameFor(m => m.Price)</th>

<th style="width: 50px">@Html.DisplayNameFor(m => m.Image)</th>

<th>@Html.DisplayNameFor(m => m.IsActive)</th>

<th>@Html.DisplayNameFor(m => m.CategoryId)</th>

<th>Düzenle</th>

</tr>

Product sayfası

public class Product

{

[Display(Name = "Ürün Id")]

public int ProductId { get; set; }

[Display(Name = "Ürün Adı")]

public string? Name { get; set; } = string.Empty;

[Display(Name = "Fiyat")]

public int Price { get; set; }

[Display(Name = "Resim")]

public string Image { get; set; } = string.Empty;

[Display(Name = "Durum")]

public bool IsActive { get; set; }

[Display(Name = "Kategori")]

public int CategoryId { get; set; }

}

[HttpGet]

public IActionResult Index(string searchString)

{

// products değişkenine tüm ürünler koyuluyor

var products = Repository.Products;

// searchstring null veya boş değilse eğer bu koşul çalışıyor

if (!String.IsNullOrEmpty(searchString))

{

// Ürünlerin içinde arama yapılıyor

// Mevcut ürünler arasından where koşuluyla linq expression kullanılıyor

// p denilen değişken products içerisinde yer alan her bir ürünü temsil ediyor

// bu, products içerisindeki ürünlerin adları içerisinde, bizim search stringimiz var mı kontrol ediliyor

// varsa eğer, listeye çevirilerek, productsa kaydediliyor

products = products.Where(p => p.Name.ToLower().Contains(searchString)).ToList();

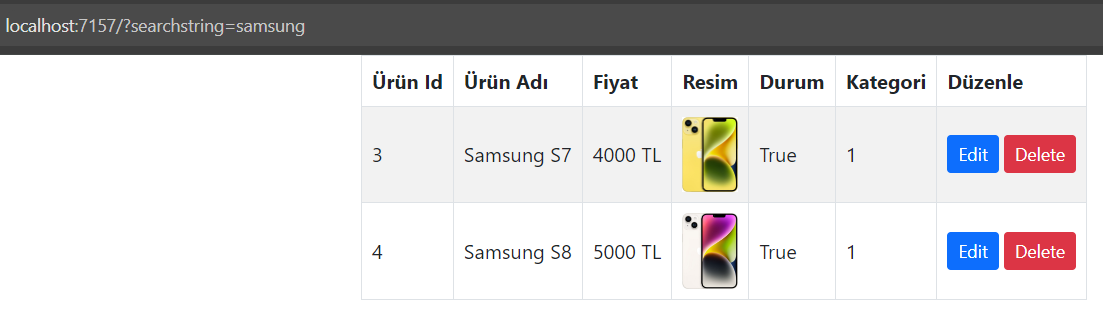
}

// bundan sonra tarayıcıda linkin sonuna ?searchString=iphone yazarak arama yapabiliriz

return View(products);

}

Sadece samsung kelimesini içeren ürünler:



// bind edilecek alanları burada da tanımlayabiliyoruz

// bind edilmesini istemediğimiz alanlara giderek BindNever yazarsak da bind edilmelerini engellemiş oluyoruz

[Bind("Name", "Price")]

public class Product

{

// Product Id'nin namenin çağırılmasıyla tabloda nasıl görünmesi gerektiğini belirtiyoruz

[Display(Name = "Ürün Id")]

[Required]

// BindNever attribute'u ile bu alanın bind edilmemesini sağlıyabiliyoruz

[BindNever]

public int ProductId { get; set; }

Yukarıdaki yöntemlere ek olarak controller üzerinde de bind işlemi gerçekleştirebiliyoruz.

// bind kullanarak, viewden sadece Name ve Price alanlarını alıyoruz

[HttpPost]

public IActionResult Create([Bind("Name", "Price")]Product model)

{

ViewBag.Categories = new SelectList(Repository.Categories, "CategoryId", "Name");

return View();

}

@\* kategori id için asp for kullanıldı

kategori dropdown listesi için viewbag kullanıldı \*@

<div class="mb-3">

<label class="form-label" for="CategoryId">Category</label>

<**select** **asp-for**="CategoryId" **asp-items**="@ViewBag.Categories" class="form-select">

<**option** **value**="0">Hepsi</**option**>

</**select**>

</div>

<div class="mb-3">

<label class="form-label" for="Image">Resim</label>

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

@\* span etiketi içerisinde, asp-validation-for kullanılarak, Image alanı için validation mesajı gösterildi

aynısını diğer alanlarda da yapabiliyoruz \*@

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

</div>

Veya tüm alanları field required yapmak istiyorsak bu kodu view sayfamızda yazmalıyız

<**form** method="post" **asp-controller**="home" **asp-action**="create">

<**div** **asp-validation-summary**="All"></**div**>

<div class="mb-3">

<label for="Name" class="form-label">Name</label>

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

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

</div>

Bu değişkenlerin validationının controller kodunda kontrol edilmesi gerekiyor.

[HttpPost]

public IActionResult Create(Product model)

{

ViewBag.Categories = new SelectList(Repository.Categories, "CategoryId", "Name");

if (ModelState.IsValid)

{

model.ProductId = Repository.GetProductId();

Repository.CreateProduct(model);

return RedirectToAction("Index");

}

return View(model);

}

Bu field required özelliklerinin çalışması için, ilgili değerin kodlarda null değer alabilmesi gerekiyor.

[Display(Name = "Ürün Adı")]

[Required]

// null değer alamasaydı, required özelliğinin anlamı olmazdı. Çünkü null değer alamazsa, zaten null değer alamazdı

public string? Name { get; set; } = string.Empty;

// string length attribute'u ile bu alanın maksimum kaç karakter alabileceğini belirleyebiliyoruz

[StringLength(8)]

public string? Image { get; set; } = string.Empty;

8 karakterden fazla girmeye çalıştığımızdaysa hiçbir şey olmaz ve giremeyiz.

A screenshot of a computer

Description automatically generated

IMAGE FILE VALIDATION FARKLILIĞI

A screenshot of a computer

Description automatically generated

@\* enctype="multipart/form-data"

bu kodu yazmasaydık, resim alanı gönderilmeyecekti \*@

<**form** method="post" **asp-controller**="home" **asp-action**="create" enctype="multipart/form-data">

<**div** **asp-validation-summary**="All"></**div**>

@\* Image dosyası koymak için artık dosya ismi girmek yerine, kendi bilgisayarımızdan dosya yükleyeceğiz \*@

<div class="input-group mb-3">

<input type="file" name="imageFile" id="imageFile" class="form-control">

</div>

Enctype kodunu eklemeseydik, resim yüklediğimizde bile “imageFile is required” hatasını almaya devam edecektik. Bunun sebebi imageFile’ın bu kod olmadan gönderilmemesi.

ImageFileyi alabilmek için parametre olarak Create metoduna ekledik.

Bunun yerine, ImageFile bilgisini Product classında da yazabilirdik.

[HttpPost]

public IActionResult Create(Product model, IFormFile imageFile)

{

...

Artık resim yükleyip kaydete bastığımızda “imageFile is required” hatası olmaz.

A screenshot of a computer

Description automatically generated

DOSYA YOLU VE DOSYA OLUŞTURMA

Resim eklemek istiyoruz fakat kendi projemiz üzerinden eklemek istiyoruz. Ayrıca dosya uzantısını da belirtmemiz gerekiyor. Bunun dışında seçeceğimiz dosyanın da adını rastgele olarak belirlemek istiyoruz.

// Metodumuzu, async ve Task yapıyoruz. Bunun sebebi: async çalıştırmak istiyoruz ve async çalıştırabilmek için

// View Metodlarının Task olması gerekiyor

[HttpPost]

public async Task<IActionResult> Create(Product model, IFormFile imageFile)

{

ViewBag.Categories = new SelectList(Repository.Categories, "CategoryId", "Name");

var extension = "";

// Gönderilen dosyanın null olup olmadığını kontrol ediyoruz

if (imageFile != null)

{

// sadece izin verdiğimiz uzantılı dosyalar yüklenebilmeli

var allowedExtensions = new[] { ".jpg", ".png", ".jpeg" };

// yüklenilen dosyanın uzantısını öğreniyoruz

extension = Path.GetExtension(imageFile.FileName);

// yüklenilen dosyanının uzantısına izin verip vermediğimizi kontrol ediyoruz.

// izin vermiyorsak hata veriyoruz

if (!allowedExtensions.Contains(extension))

{

ModelState.AddModelError("", "Geçerli bir resim seçiniz.");

}

}

if (ModelState.IsValid)

{

// yüklenilen dosya null değilse işlemleri gerçekleştireceğiz

if (imageFile != null)

{

// yüklenecek dosya projeye klasörüne yükleneceği ve çakışma olabileceği için

// rastgele bir isim belirliyoruz

var randomFileName = Path.GetRandomFileName() + extension;

// resimlerin yükleneceği klasör yolunu belirliyoruz

var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomFileName);

// resimleri yükleme işlemleri

using (var stream = new FileStream(path, FileMode.Create))

{

// asenkron bir şekilde resmin projemize yüklenmesini sağlıyoruz

await imageFile.CopyToAsync(stream);

}

// resmin adının producta eklenmesini sağlıyoruz

// bunu yapmazsak, index.cshtml bunu görüntüleyemez

model.Image = randomFileName;

model.ProductId = Repository.GetProductId();

Repository.CreateProduct(model);

return RedirectToAction("Index");

}

}

return View(model);

}

// kullanıcıya editlemek istediği sayfayı gösterebilmek için

// IActionResult metodu yazıyoruz

public IActionResult Edit(int? id)

{

ViewBag.Categories = new SelectList(Repository.Categories, "CategoryId", "Name");

// id null ise, kullanıcıya sayfa bulunamadı hatası veriyoruz

if (id == null)

{

return NotFound();

}

// kullanıcıya sadece editlemek istediği ürünü gösterebilmek için gönderilen id yi değişkene kaydediyoruz

var entity = Repository.Products.FirstOrDefault(p => p.ProductId == id);

// id ye göre editlenmek istenen ürün null ise kullanıcıya sayfa bulunamadı hatası gönderiyoruz

if (entity == null)

{

return NotFound();

}

return View(entity);

}

@\* bunun sayesinde, controllerdaki edit IActionResult metoduna düzenlenecek

olan ProductId bilgisini göndermiş oluyoruz.

Kullanıcıların görmesine gerek olmadığı için gizli olarak yolluyoruz.\*@

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

// ürünümüzü düzenlemek için Repository classındayken

// bir EditProduct metodu yazıyoruz ve ürününün çeşitli özelliklerini gelen parametrelere

// göre değiştiriyoruz

public static void EditProduct(Product updatedProduct)

{

var entity = \_products.FirstOrDefault(p => p.ProductId == updatedProduct.ProductId);

if (entity != null)

{

entity.Name = updatedProduct.Name;

entity.Price = updatedProduct.Price;

entity.Image = updatedProduct.Image;

entity.CategoryId = updatedProduct.CategoryId;

entity.IsActive = updatedProduct.IsActive;

}

}

@\* bunun sayesinde, controllerdaki edit IActionResult metoduna düzenlenecek

olan ProductId bilgisini göndermiş oluyoruz.

Kullanıcıların görmesine gerek olmadığı için gizli olarak yolluyoruz.\*@

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

[HttpPost]

public async Task<IActionResult> Edit(int? id, Product model, IFormFile? imageFile)

{

if (id != model.ProductId)

{

return NotFound();

}

if (ModelState.IsValid)

{

var extension = "";

if (imageFile != null)

{

var allowedExtensions = new[] { ".jpg", ".png", ".jpeg" };

extension = Path.GetExtension(imageFile.FileName);

var randomFileName = Path.GetRandomFileName() + extension;

var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomFileName);

using (var stream = new FileStream(path, FileMode.Create))

{

await imageFile.CopyToAsync(stream);

}

model.Image = randomFileName;

}

Repository.EditProduct(model);

return RedirectToAction("Index");

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> Edit(int? id, Product model, IFormFile? imageFile)

{

// gönderilen id, ürün id ile aynı mı kontrol ediyoruz

if (id != model.ProductId)

{

return NotFound();

}

// formdan gönderilen değerler uygun mu diye kontrol ediyoruz

if (ModelState.IsValid)

{

// resim yükleme işlemlerini gerçekleştiriyoruz

var extension = "";

if (imageFile != null)

{

var allowedExtensions = new[] { ".jpg", ".png", ".jpeg" };

extension = Path.GetExtension(imageFile.FileName);

var randomFileName = Path.GetRandomFileName() + extension;

var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomFileName);

using (var stream = new FileStream(path, FileMode.Create))

{

await imageFile.CopyToAsync(stream);

}

// Productın mevcut Image parametresine yeni bir değer atıyoruz

model.Image = randomFileName;

}

// Create metodundan farklı olarak, EditProduct metodunu kullanıyor

// ve ürünün özelliklerini değiştiriyoruz

Repository.EditProduct(model);

// ürünü düzenledikten sonra Index sayfasına geri gönderiyoruz

return RedirectToAction("Index");

}

// ModelState.IsValid false olduğu için model bilgileriyle birlikte Edit sayfasının

// bir daha açılmasını sağlıyoruz

return View(model);

}

Edit sayfasını doğru hale getirdikten sonra düzenlemenin controller tarafında gerçekleşmesini sağladık. Tabi bunun doğru şekilde gerçekleşmesi için, Repository classında EditProduct adında bir metot yazdık ve controller tarafında kullanımını sağladık.

Şimdiki amacımız, delete tuşuna bastığımızda ürünlerin silinmesi.

Bunun için önce Index sayfasındaki Delete butonunu düzenleyeceğiz. Bunun ardından Repository classımızda, ürünün gerçekten silinebilmesi için bir ürün silme metodu yazıyoruz. Daha sonra Controller’da bir delete IActionResult metodu tanımlıyoruz ve bu metotta ürünün silinmesini sağlayacak kodu yazıyoruz.

@\* ürün id yi delete metoduna gönderiyoruz \*@

<a href="/home/Delete/@item.ProductId" class="btn btn-sm btn-danger">Delete</a>

Controller:

public IActionResult Delete(int? id)

{

// gönderilen ürün id'ye göre, ürün modelini entity değişkenine aktarıyoruz

var entity = Repository.Products.FirstOrDefault(p => p.ProductId == id);

if (id == null || entity == null)

{

return NotFound();

}

else

{

Repository.DeleteProduct(entity);

}

return RedirectToAction("Index");

}

Repository:

public static void DeleteProduct(Product deletedProduct)

{

// gönderilen ürünü entity değişkenine aktarıyoruz

var entity = \_products.FirstOrDefault(p => p.ProductId == deletedProduct.ProductId);

// gönderilen ürün null değilse, Remove metodu sayesinde ürünü siliyoruz

if (entity != null)

{

\_products.Remove(entity);

}

}

Bu derste ürünü silmeyi biraz değiştiriyoruz. Yaptığımız şey, ürünü silme butonuna tıklayınca doğrudan silme durumunu değiştirip, “ürünü gerçekten silmek istiyor musunuz?” uyarısını yapacağımız bir sayfaya yönlendireceğiz ve bu sayfada verilen evet veya hayır yanıtına göre ürünü silmek olacak.

Teknik olarak yapacağımız şey, bir DeleteConfirm viewi tanımlamak, bir HttpPost Delete metodu yazmak olacak.

@model Product

<div>

<p>@Model.Name isimli ürünü silmek istediğinize emin misiniz?</p>

<**form** action="" method="post">

// HttpPost Delete IActionResult metoduna id bilgisini gönderiyoruz.

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

<button type="submit" class="btn btn-danger">Evet</button>

<a href="/" class="btn btn-primary">Hayır</a>

</**form**>

</div>

[HttpPost]

public IActionResult Delete(int? id, int ProductId)

{

if (id != ProductId)

{

return NotFound();

}

var entity = Repository.Products.FirstOrDefault(p => p.ProductId == ProductId);

if (entity == null)

{

return NotFound();

}

// ürünü siliyoruz

Repository.DeleteProduct(entity);

// ürünü sildikten sonra Index sayfasına yeniden yönlendiriyoruz

return RedirectToAction("Index");

}

Toplu şekilde ürün veya kayıt güncellemeye batch update denir. Bu derste bunu yapacağız.

Teknik olarak yapacağımız şeyler:

1. Index sayfasına bir buton ekleyeceğiz, gönderilecek bilgileri bir form içerisinde post olarak, controllera EditProducts şeklinde gönderilecek.
2. Controllerda bir EditProducts metodumuz olacak, bu metot sayesinde toplu şekilde ürün güncelleyeceğiz.
3. EditProducts’ın veritabanında güncelleme yapması için Repository classında bir EditIsActive metodu yazacağız.

Index.cshtml:

<**form** **asp-action**="EditProducts" method="post">

<table class="table table-sm w-100 text-center mx-auto table-bordered table-striped">

<thead>

<tr>

@\* İlgili Product sayfasında yer alan Display(Name) özelliği kullanılarak tablo başlıkları doğrudan buraya geliyor \*@

<th>@Html.DisplayNameFor(m => m.Products[0].ProductId)</th>

<th>@Html.DisplayNameFor(m => m.Products[0].Name)</th>

<th>@Html.DisplayNameFor(m => m.Products[0].Price)</th>

<th style="width: 50px">@Html.DisplayNameFor(m => m.Products[0].Image)</th>

<th>@Html.DisplayNameFor(m => m.Products[0].IsActive)</th>

<th>@Html.DisplayNameFor(m => m.Products[0].CategoryId)</th>

<th>Seç</th>

<th>Düzenle</th>

</tr>

</thead>

<tbody class="align-middle">

@\* for döngüsüyle, controller’daki EditProducts metoduna bilgileri yolluyoruz \*@

@for (int i = 0; i < Model.Products.Count; i++)

{

<tr>

<td>

@Model.Products[i].ProductId

Edit Products’a her bir ürünün ProductId’sini yolladık

<**input** **type**="hidden" **asp-for**="@Model.Products[i].ProductId"/>

</td>

<td>@Model.Products[i].Name</td>

<td>@Model.Products[i].Price TL</td>

<td><img src="/img/@Model.Products[i].Image" alt="@Model.Products[i].Image" class="img-fluid" /></td>

<td>@Model.Products[i].IsActive</td>

<td>@Model.Products[i].CategoryId</td>

<td>

Edit Products’a her bir ürünün ProductId’sini yolladık

<**input** **type**="checkbox" **asp-for**="@Model.Products[i].IsActive">

</td>

<td>

@\* ürün id yi edit metoduna gönderiyoruz \*@

<a href="/home/Edit/@Model.Products[i].ProductId" class="btn btn-sm btn-primary">Edit</a>

<a href="/home/Delete/@Model.Products[i].ProductId" class="btn btn-sm btn-danger">Delete</a>

</td>

</tr>

}

</tbody>

</table>

<button type="submit" class="btn btn-primary">Kaydet</button>

</**form**>

Repository.cs:

public static void EditIsActive(Product updatedProduct)

{

// burada bir LİNQ expression kullandığımız için, \_products ile entity aynı nesneye işaret eder

var entity = \_products.FirstOrDefault(p => p.ProductId == updatedProduct.ProductId);

if (entity != null)

{

entity.IsActive = updatedProduct.IsActive;

}

}

HomeController.cs:

public IActionResult EditProducts(List<Product> Products)

{

foreach (var item in Products)

{

// EditIsActive metoduna Liste halindeki Products bilgilerini foreach kullanarak teker teker yolladık

// ve veritabanını güncelledik

Repository.EditIsActive(item);

}

return RedirectToAction("index");

}