Aula 3 – Create

## Vamos criar a razor page Create.

Abaixo o html:

@page

@model Microsoft.eShopWeb.Web.Pages.Admin.CreateModel

<div class="container">

<div class="row">

<div class="col-md-8">

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

<img class="esh-catalog-thumbnail" src="@Model.Modelo.PictureUri" />

<div class="form-group">

<**label** **asp-for**="Modelo.Name"></**label**>

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

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

</div>

<div class="form-group">

<**label** **asp-for**="Modelo.Description"></**label**>

<**input** **asp-for**="Modelo.Description" class="form-control" />

<**span** **asp-validation-for**="Modelo.Description"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="@Model.Modelo.CatalogBrandId"></**label**>

<**select** **asp-for**="@Model.Modelo.CatalogBrandId" **asp-items**="@Model.Modelo.Brands" class="form-control"></**select**>

<**span** **asp-validation-for**="Modelo.CatalogBrandId"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="@Model.Modelo.CatalogTypeId" data-title="type"></**label**>

<**select** **asp-for**="@Model.Modelo.CatalogTypeId" **asp-items**="@Model.Modelo.Types" class="form-control"></**select**>

<**span** **asp-validation-for**="Modelo.CatalogTypeId"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="@Model.Modelo.Price"></**label**>

<**input** **asp-for**="Modelo.Price" class="form-control" />

<**span** **asp-validation-for**="Modelo.Price"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="@Model.Modelo.PictureUri"></**label**>

<**input** **asp-for**="Modelo.PictureUri" class="form-control" />

<**span** **asp-validation-for**="Modelo.PictureUri"></**span**>

</div>

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

<input type="submit" value="Save" class="esh-catalog-button" />

</div>

</**form**>

</div>

</div>

</div>

@section Scripts {

<**partial** **name**="\_ValidationScriptsPartial" />

}

Note que:

* O model deve ter uma variável chamada Modelo que conterá todos os dados necessários para preencher o formulário;
* Para preencher os combos de Types e Brands, tem que ter no modelo essas 2 propriedades, que são uma lista de SelectListItem

Então iremos ter que criar outro viewModel, pois o que utilizamos na tela de Alteração era bem simplificada.

Crie na mesma pasta o CatalogItemCreateViewModel¸com as propriedades abaixo:

public class CatalogItemCreateViewModel

{

[Required(ErrorMessage ="Nome é obrigatório")]

public string Name { get; set; }

[Required(ErrorMessage = "Description é obrigatório")]

public string Description { get; set; }

[Display(Name = "Tipo")]

[Required(ErrorMessage = "Type é obrigatório")]

public int CatalogTypeId { get; set; }

[Display(Name ="Brand")]

[Required(ErrorMessage = "Brand é obrigatório")]

public int CatalogBrandId { get; set; }

[Display(Name="URL da imagem")]

public string PictureUri { get; set; }

[Required(ErrorMessage = "Preço é obrigatório")]

[Range(minimum:0.01d, maximum:999d, ErrorMessage ="Faixa de valor inválida")]

public decimal Price { get; set; }

public List<SelectListItem> Brands { get; set; }

public List<SelectListItem> Types { get; set; }

}

Note que:

* já tem validações;
* alguns tem também as anotações de Display Name.

Na classe de pageModel, crie uma variável Modelo do tipo CatalogItemCreateViewModel e marque-a como BindProperty para que funcione o DataBind:

[BindProperty]

public CatalogItemCreateViewModel Modelo { get; set; } = new CatalogItemCreateViewModel();

No **OnGet** só precisamos preencher os combos de Types e Brands. Na classe de serviço CatalogViewModelService temos 2 métodos que fazem isso, o GetTypes e GetBrands.

Receba esse serviço no Construtor, e use dessa forma:

Modelo.Brands = (await \_catalogViewModelService.GetBrands()).ToList();

Modelo.Types = (await \_catalogViewModelService.GetTypes()).ToList();

Teste a aplicação para ver se está preenchendo a tela corretamente.

## Gravando o registro

Crie o método onPost assíncrono:

public async Task<IActionResult> OnPost()

{

A primeira coisa que temos que fazer é verificar se o modelo está válido. Note que não precisamos dar o TryUpdateModel, o bind é automático:

if (ModelState.IsValid)

{

}

else

{

ModelState.AddModelError(string.Empty, "Ocorreu um erro ao inserir.");

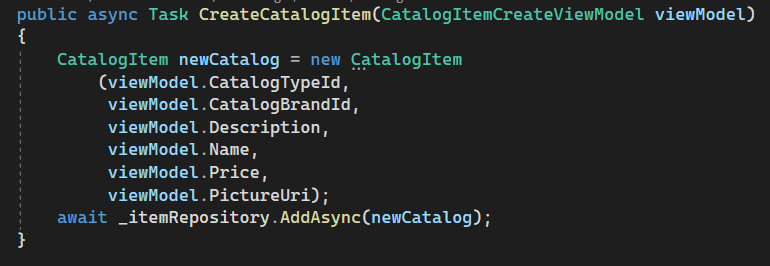
await PreencheCombos(Modelo);

return Page();

}

Nesse else fazemos um tratamento do erro e preenchemos novamente os combos, pois essas listas não são “binded” a nenhum controle.

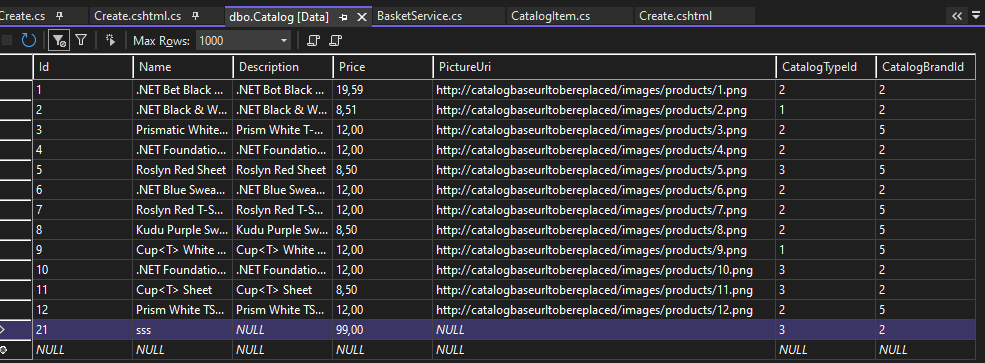
No IF temos que converter o ViewModel em CatalogItem (entity) e adicionar no repository. Para isso vamos criar um método no CatalogViewModelService que já recebemos:



Pode testar e gravar um item de catálogo (C.D.V.)

## Validações

Caso você tenha inserido corretamente o registro, deve ter percebido que ao voltar para a lista dá um erro de null pointer. É porque a URL tem que ter um padrão específico para funcionar, veja os dados:



Apague o registro que você criou para poder rodar a aplicação novamente.

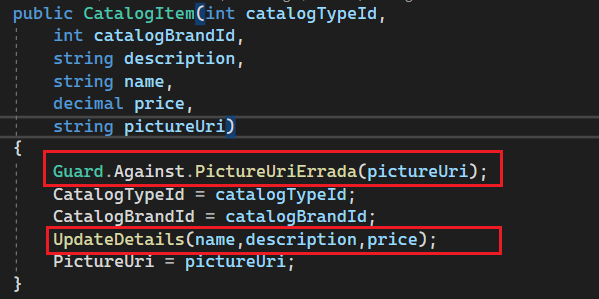
A validação dessa coluna PictureUri deveria ser feito em 2 lugares:

* na tela, validando no viewmodel
* na entidade ou serviço do Core

Pois essa validação deve funcionar também para a API ou outras formas de inserir o CatalogItem.

Vamos fazer a validação na entidade (core). Aqui fiquei na dúvida se deveríamos criar um Service e um método de Create, e validar lá. Mas também teria que validar no Update, e vi também que a API grava diretamente no Repository; no fim achei melhor validar logo na entidade pois é algo que só depende dela. Não sei se é o certo.

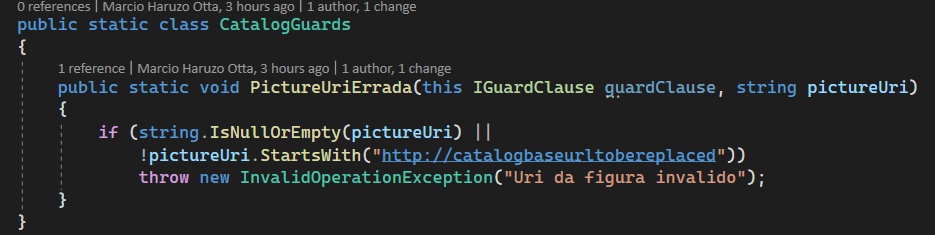
Então, abra a entidade CatalogItem e faça essa alteração no construtor:



Essa alteração no updateDetails é pra já validar dados null em campos obrigatórios.

A validação PictureUriErrada temos que implementar.

Dentro do arquivo GuardExtensions.cs (ou em um outro arquivo... aqui também não sei o que seria melhor) crie uma classe chamada CatalogGuards:

Implemente a validação como essa: 

Se não tiver preguiça pode criar uma exception customizada, seria o certo.

Achei que esses guards são trabalhosinhos mas fica fácil de testar e separa bem as validações.