07

Validando Dados do Formulário no Servidor

Transcrição

Temos a validação do lado do cliente, e já podemos começar a tratar do cadastro quando ele for recebido no servidor por meio da action `resumo`. Essa action receberá o parâmetro `Cadastro`, e ao final gravaremos essa informação no banco de dados.

Para realizar a gravação no banco, utilizamos `CastroRepositoy.cs`. Neste arquivo, teremos a interface `ICadastroRepository`.

```
namespace CasaDoCodigo.Repositoires
{

    public interface ICadastroRepository
    {

    }

    public class CadastroRepository : BaseRepository<Cadastro> ICadastroRepository
    {

        public CadastroRepository(ApplicationContext contexto) : base(contexto)
        {
        }
    }
}
```

Criaremos um novo método chamado `Update()` para atualizar as informações do cadastro. Tal método retornará um objeto `Cadastro` e receberá duas informações: `cadastroId` e `Cadastro novoCadastro`.

```
namespace CasaDoCodigo.Repositoires
{

    public interface ICadastroRepository
    {
        Cadastro Update(int cadastroId, Cadastro novoCadastro)
    }

    public class CadastroRepository : BaseRepository<Cadastro>, ICadastroRepository
    {

        public CadastroRepository(ApplicationContext contexto) : base(contexto)
        {
        }
    }
}
```

Assim feito, criaremos um método na classe concreta `CadastroRepository` a partir da interface. Clicaremos com o botão direito sobre `ICadastroRepository` e selecionaremos a opção "Ações Rápidas e Refatorações > Implementar interface". Dessa

forma teremos um método novo, mas por enquanto não inseriremos o seu código definitivo, apenas um break point na linha `throw new NotImplementedException();` .

```
namespace CasaDoCodigo.Repositoires
{

    public interface ICadastroRepository
    {
        Cadastro Update(int cadastroId, Cadastro novoCadastro)
    }

    public class CadastroRepository : BaseRepository<Cadastro>, ICadastroRepository
    {

        public CadastroRepository(ApplicationContext contexto) : base(contexto)
        {
        }

        public Cadastro Update(int CadastroId, novoCadastro)
        {
            throw new NotImplementedException();
        }
    }
}
```

Para acessarmos esse novo método do repositório de cadastro, acessaremos o repositório de pedido, afinal é ele que contém informações do pedido que está na sessão do Asp.NET Core. Portanto iremos até `PedidoRepository.cs` e criaremos uma referência para `CadastroRepository.cs` por meio de um campo privado. Esse campo será alimentado via injeção de dependência, logo criaremos um novo parâmetro no construtor (`cadastroRepository`) que será alimentando no corpo do construtor no momento em que acessarmos `this.cadastroRepository = cadastroRepository ;`

```
public class PedidoRepository : BaseRepository<Pedido>, IPedidoRepository
{
    private readonly IHttpContextoAccessor contextAccessor;
    private readonly IItemPedidoRepository itemPedidoRepository;
    private readonly ICadastroRepository cadastroRepository;

    public PedidoRepository(ApplicationContext contexto,
        IHttpContextoAccessor contextAccessor,
        IItemPedidoRepository itemPedidoRepository,
        ICadastroRepository castroRepository) : base(contexto)
    {
        this.contextAccessor = contextAccessor;
        this.itemPedidoRepository = itemPedidoRepository;
        this.cadastroRepository = cadastroRepository;
    }
```

Precisamos criar outro método na `PedidoRepository` que será responsável pela atualização do cadastro. Na interface, criaremos esse novo método chamado `UpdateCadastro()` que retornará `Pedido` . Por sua vez, o método receberá o cadastro sendo modificado na view (`Cadastro cadastro`).

```
public interface IPedidoRepository
{
```

```
    Pedido GetPedido();
    void AddItem(string codigo);
    UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido);
    Pedido UpdateCadastro(Cadastro cadastro);
}
```

Faremos a classe concreta implementar o novo método clicando sobre `IPedidoRepository` com o botão direito e selecionando as opções "Ações Rápidas e Refatorações > Implementar Interface". Ao final da classe, teremos o método `UpdateCadastro()`.

```
<****!****>

public Pedido UpdateCadastro(Cadastro cadastro)
{
    throw new NotImplementedException():
}
```

Acessaremos `cadastroRepository.Update()`, que receberá `Cadastro.Id`, que está associado a `pedido`, portanto primeiramente pegaremos `pedido`, declarando uma variável `pedido = GetPedido()`. O segundo parâmetro do método `Update()` será `cadastro`. Ao final das alterações, deveremos retornar `Pedido` a partir do método `UpdateCadastro()`, faremos isso escrevendo `return pedido`, isto é, a variável local.

```
<****!****>

public Pedido UpdateCadastro(Cadastro cadastro)
{
    var pedido = GetPedido();
    cadastroRepository.Update(pedido.Cadastro.Id, cadastro);
    return pedido;
}
```

Votaremos à `PedidoController.cs` e consumiremos o método `UpdateCadastro()`. Trabalharemos no seguinte trecho do código:

```
[HttpPost]
public IActionResult Resumo(Cadastro.castro)
{
    return View(pedidoRepository.GetPedido());
}
```

Ao invés de utilizarmos `return View(pedidoRepository.GetPedido());` inseriremos `return View(pedidoRepository.UpdateCadastro(cadastro));`

```
[HttpPost]
public IActionResult Resumo(Cadastro.castro)
{
    return View(pedidoRepository.UpdateCadastro(cadastro));
}
```

Contudo, temos de levar em consideração o seguinte quadro: pode ser que não tenhamos preenchido corretamente as informações na view, isto é, `Cliente.cshtml`. Por isso é sempre importante validar os dois lados, tanto cliente quanto servidor. Para termos certeza de que o modelo foi preenchido corretamente e que as regras de negócio presentes são válidas, devemos verificar o estado do modelo. Para realizarmos essa verificação, inseriremos `if` e acessaremos o objeto `ModelState`, em seguida faremos uso da propriedade `IsValid`. Caso o modelo esteja correto, o cadastro será atualizado.

```
[HttpPost]
public IActionResult Resumo(Cadastro.castro)
{
        if(ModelState.IsValid)
        {
            return View(pedidoRepository.UpdateCadastro(cadastro));
        }
}
```

Se o modelo não for válido, retornaremos o usuário para a view de cadastro até que as informações tenham sido preenchidas corretamente. Para fazer esse redirecionamento do usuário, utilizamos `return RedirectToAction()`, que receberá como parâmetro o nome action correta, no caso, `"Cadastro"`.

```
[HttpPost]
public IActionResult Resumo(Cadastro.castro)
{
        if(ModelState.IsValid)
        {
            return View(pedidoRepository.UpdateCadastro(cadastro));
        }
        return RedirectToAction("Cadastro");
}
```

Nas próximas aulas iremos programar a gravação do cadastro no banco de dados.