05

Refatorando Repositórios

Transcrição

Criamos a classe `UpdateQuantidadeResponse` que será retornada para `Carrinho.js` na função `ajax()`, que contém a função `done()` que por sua vez, receberá como resposta uma instância dessa nova classe.

Iremos preparar o método `UpdateQuantidadade()`, que está na `PedidoController.cs` para retornar uma instância da nova classe.

Iremos até a `PedidoController.cs` para retornar o `UpdateQuantidadeResponse`. Adicionaremos a palavra `return` logo a frente de `itemPedidoRepository`.

```
  [HttpPost]
        public void UpdateQuantidade([FromBody]ItemPedido itemPedido)
        {
            return itemPedidoRepository.UpdateQuantidade(itemPedido);
        }
    }
}
```

No entanto, o método `UpdadeQuantidade()` retorna um `void`, portanto iremos até `itemPedididoRepository` e encontraremos a assinatura do método na interface `IItemPedidoRepository`.

```
namespace CasaDoCodigo.Repositories
{
    public interface IItemPedidoRepository
    {
        void UpdateQuantidade(ItemPedido itemPedido);
    }
```

Mais abaixo no código encontraremos a implementação concreta do método `UpdateQuantidade()`.

```
 public void UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB =
            dbSet
                .Where(ip => ip.Id == itemPedido.Id)
                .SingleOrDefault();

            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                contexto.SaveChanges();
            }
        }
```

Trocaremos a palavra `void` por `UpdateQuantidadeResponse`. É importante notarmos que o objeto `UpdateQuantidadeResponse` retorna não apenas o item que é alterado, mas também o `CarrinhoViewModel` que contém informações do pedido. Como estamos no repositório de `ItemPedido` não teremos como acessar o pedido a partir daqui, por isso moveremos o método `UpdateQuantidade()` para o repositório de pedidos, isto é, `ItemPedidoRepository.cs`. Colaremos todo o trecho destacado ao final da classe, e o removeremos de seu local original.

```
public UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB =
            dbSet
                .Where(ip => ip.Id == itemPedido.Id)
                .SingleOrDefault();

            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                contexto.SaveChanges();
            }
        }
```

Moveremos, ainda, a assinatura da interface. Copiaremos a linha `void UpdateQuantidade(ItemPedido itemPedido)` em `ItemPedidoRepository.cs`, depois podemos remover essa assinatura do código.

```
namespace CasaDoCodigo.Repositories
{
    public interface IItemPedidoRepository
    {
        void UpdateQuantidade(ItemPedido itemPedido);
    }
```

Depois colaremos em `PedidoRepository.cs`, ou seja, na interface do repositório de pedidos.

```
namespace CasaDoCodigo.Repositories
{
    public interface IPedidoRepository
    {
        Pedido GetPedido();
        void AddItem(string codigo);
                void UpdateQuantidade(ItemPedido itemPedido);
    }
```

Iremos retornar a parte do `UpdateQuantidade`, modificando o retorno da interface de `void` para `UpdateQuantidadeResponse`

```
namespace CasaDoCodigo.Repositories
{
    public interface IPedidoRepository
    {
        Pedido GetPedido();
        void AddItem(string codigo);
                UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido);
```

```
        }
```

Mais abaixo no código, começaremos a implementar o `UpdateQuantidade` que foi movida para `PedidoRepository.cs` .

Temos `ItemPedidoDB` que será obtido a partir do banco de dados e que inicialmente acessávamos a partir do `dbset` , que era de `ItemPedido` , inicalmente. Aqui o `dbset` é referente a `Pedido` . Teremos de obter por meio do `itemPedidoRepository` a informação de `ItemPedido` .

```
public UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB =
            dbSet
                .Where(ip => ip.Id == itemPedido.Id)
                .SingleOrDefault();

            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                contexto.SaveChanges();
            }
        }
```

Criaremos em `ItemPedidoRepository` um novo método para obter somente `ItemPedido` a partir do `itemPedidoId` . Iseriremos uma assinatura de método que retornará `ItemPedido` e criaremos um novo método chamado `GetItemPedido()` , que receberá como parâmetro `int itemPedidoId` .

```
namespace CasaDoCodigo.Repositories
{

    public interface IItemPedidoRepository
    {
        ItemPedido GetItemPedido(int itemPedidoId);
    }

        public class ItemPedidoRepository : BaseRepository<ItemPedido>, IItemPedidoRepository
        {
            public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
            {
            }
        }
}
```

Feito isso, implementaremos o método na classe concreta `IItemPedidoRepository` . Dessa forma teremos `GetItemPedido` recebendo `itemPedidoId`

```
namespace CasaDoCodigo.Repositories
{
    public interface IItemPedidoRepository
    {
        ItemPedido GetItemPedido(int itemPedidoId);
    }
```

```
        public class ItemPedidoRepository : BaseRepository<ItemPedido>, IItemPedidoRepository
        {
            public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
            {
            }

            public ItemPedido GetItemPedido(int itemPedidoId)
            {
                throw new NotImplementedException();
            }
        }
}
```

Dessa forma teremos `GetItemPedido()` recebendo `itemPedidoId`. Esse método irá retornar a seguinte expressão registrada em `PedidoRepository.cs`.

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

            dbSet
                .Where(ip => ip.Id == itemPedido.Id)
                .SingleOrDefault();

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

Iremos copiar essa expressão e colá-la no método `GetItemPedido()` e adicionar a expressão `return`. Além disso, substituiremos `itemPedido.Id` por `ItemPedidoId`.

```
namespace CasaDoCodigo.Repositories
{
    public interface IItemPedidoRepository
    {
        ItemPedido GetItemPedido(int itemPedidoId);
    }

        public class ItemPedidoRepository : BaseRepository<ItemPedido>, IItemPedidoRepository
        {
            public ItemPedidoRepository(ApplicationContext contexto) : base(contexto)
            {
            }

            public ItemPedido GetItemPedido(int itemPedidoId)
            {
                return
                dbSet
                .Where(ip => ip.Id == itemPedidoId)
                .SingleOrDefault();

            }
        }
}
```

Consumiremos o método `GetItemPedido()` em `PedidoRepository.cs`, faremos isso chamando um repositório a partir do outro, realizando uma injeção de dependência por meio da interface. Em `PedidoRepository` declararemos um campo privado que será chamado de `IItemPedidoRepository itemPedidoRepository`. Esse item será fornecido via injeção de dependência, portanto precisamos criar um novo parâmetro no construtor da classe, que será `itemPedidoRepository`. Forneceremos no corpo do construtor `this.itemPedidoRepository`, atribuindo a este campo local o parâmetro `itemPedidoRepository`.

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

        public PedidoRepository(ApplicationContext contexto,
            IHttpContextAccessor contextAccessor,
            IItemPedidoRepository itemPedidoRepository) : base(contexto)
        {
            this.contextAccessor = contextAccessor;
            this.itemPedidoRepository = itemPedidoRepository;
        }
```

Ao final do código, trabalharemos no método `UpdateQuantidade()`. Removeremos o seguinte trecho do código:

```
dbSet
    .Where(ip => ip.Id == itemPedido.Id)
    .SingleOrDefault();
```

Assim, obteremos `itemPedidoDB` a parir de `ItemPedidoRepository.GetItemPedido`, passando como parâmetro `itemPedido.Id`. A nova forma do código ficará desse modo:

```
public UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB = itemPedidoRepository.GetItemPedido(itemPedido.Id);

            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                                contexto.SaveChanges();

            }
```

Conseguimos pegar `ItemPedidoDB` e salvando a quantidade de itens no banco de dados. Teremos de retornar do método `UpdateQuantidade()` uma nova instância de `UpdateQuantidadeResponse()`, para isso passaremos como parâmetro no construtor `ItemPedidoDB` e `CarrinhoViewModel`, que declararemos mais acima do código em uma variável local para que este se torne mais legível. Para `CarrinhoViewModel()` iremos acionar o método `GetPedido()` e depois acessaremos a propriedade `Itens`. Assim feito, passaremos `carrinhoViewModel` para como outro parâmetro do método `UpdateQuantidadeResponse()`.

```
public UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB = itemPedidoRepository.GetItemPedido(itemPedido.Id);
```

```
            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                                contexto.SaveChanges();

                                var carrinhoViewModel = new CarrinhoViewModel(GetPedido().Itens);

                                return mew UpdateQuantidadeResponse(itemPedidoDB, carrinhoViewModel)

            }
```

Ainda resta resolver um problema: quando `itemPedidoDB` for igual a nulo, isto é, quando não for encontrado o `Id` correto, o que faremos? Precisamos criar uma exceção. Escreveremos `trow new ArgumentException()` , passando como parâmetro a mensagem `"ItemPedido não encontrado"` .

```
public UpdateQuantidadeResponse UpdateQuantidade(ItemPedido itemPedido)
        {
            var itemPedidoDB = itemPedidoRepository.GetItemPedido(itemPedido.Id);

            if (itemPedidoDB != null)
            {
                itemPedidoDB.AtualizaQuantidade(itemPedido.Quantidade);

                                contexto.SaveChanges();

                                var carrinhoViewModel = new CarrinhoViewModel(GetPedido().Itens);

                                return mew UpdateQuantidadeResponse(itemPedidoDB, carrinhoViewModel)

            }

                        throw new ArgumentExecption("ItemPedido não encontrado");
            }
        }
}
```

Voltemos à `PedidoController` e veremos o método `UpdateQuantidade()` que é chamado no `ItemPedidoRepository` .

```
  [HttpPost]
        public UpdateQuantidadeResponse UpdateQuantidade([FromBody]ItemPedido itemPedido)
        {
            return itemPedidoRepository.UpdateQuantidade(itemPedido);
        }
    }
}
```

Faremos uma pequena alteração e chamaremos diretamente o método em `pedidoRepository`

```
    [HttpPost]
           public UpdateQuantidadeResponse UpdateQuantidade([FromBody]ItemPedido itemPedido)
           {
               return pedidoRepository.UpdateQuantidade(itemPedido);
           }

       }
   }
```

Dessa forma temos todas as chamadas operando corretamente. Nas próximas aulas veremos como utilizar o código JavaScript para tratar os retornos e fazer a atualização na tela de carrinho.