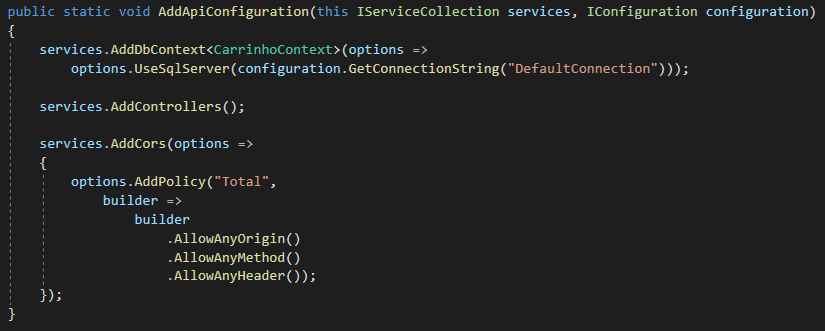
**API de Carrinhos**

* **Setup inicial**

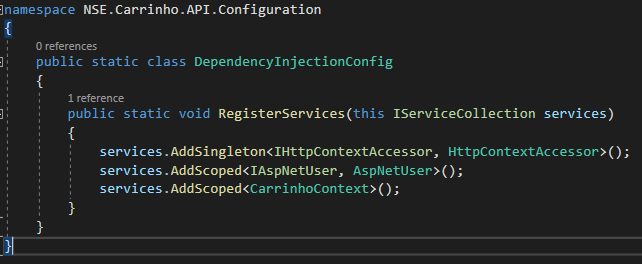
->Criar a pasta Configuration

->Dentro dela, ApiConfig

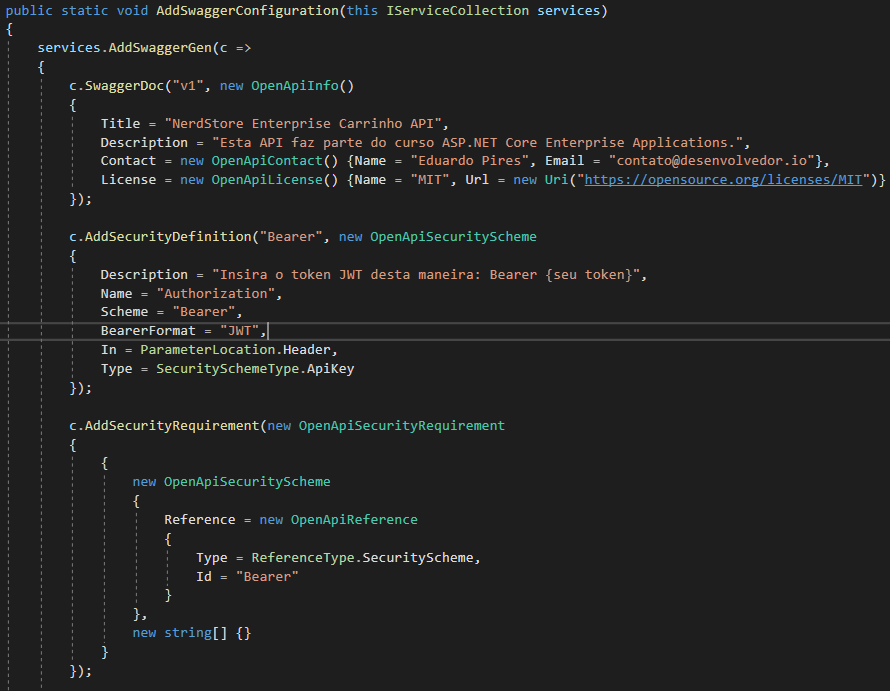




->Criar a DependencyInjectionConfig

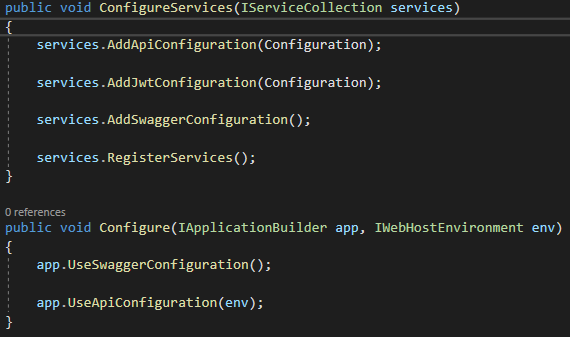


->Criar a SwaggerConfig



->Configurar a StartUp





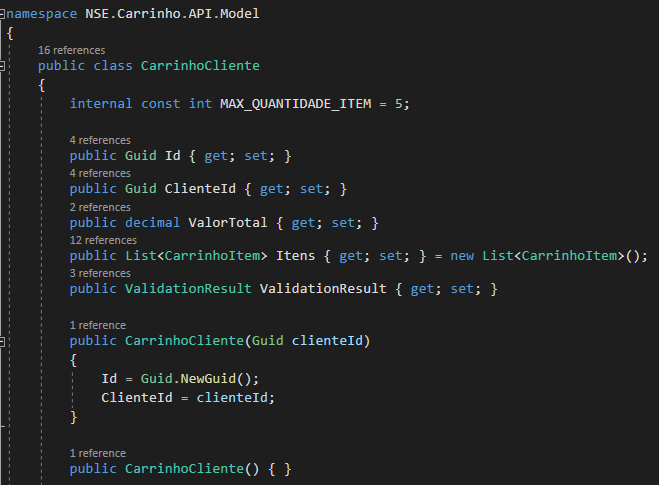
->Criar a pasta Model

->Criar a classe CarrinhoItem

>Usar o DataAnnotation no CarrinhoCliente para evitar uma referência

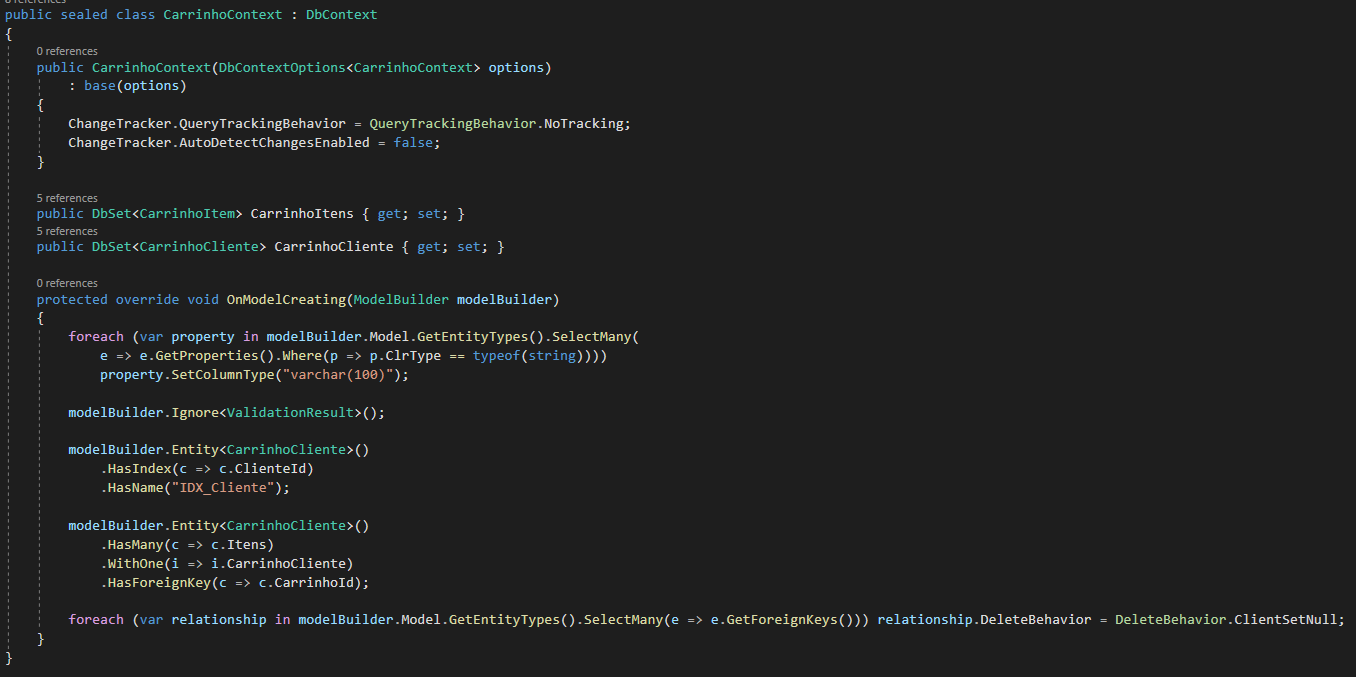


->Criar a classe CarrinhoCliente



->Criar a pasta Data

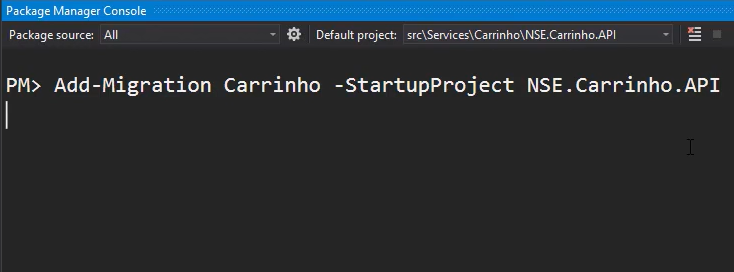
->Dentro dela, criar CarrinhoContext



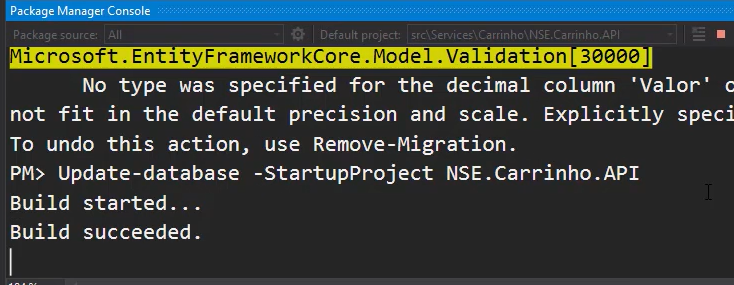
->Apontar o appSettings.Development para a base



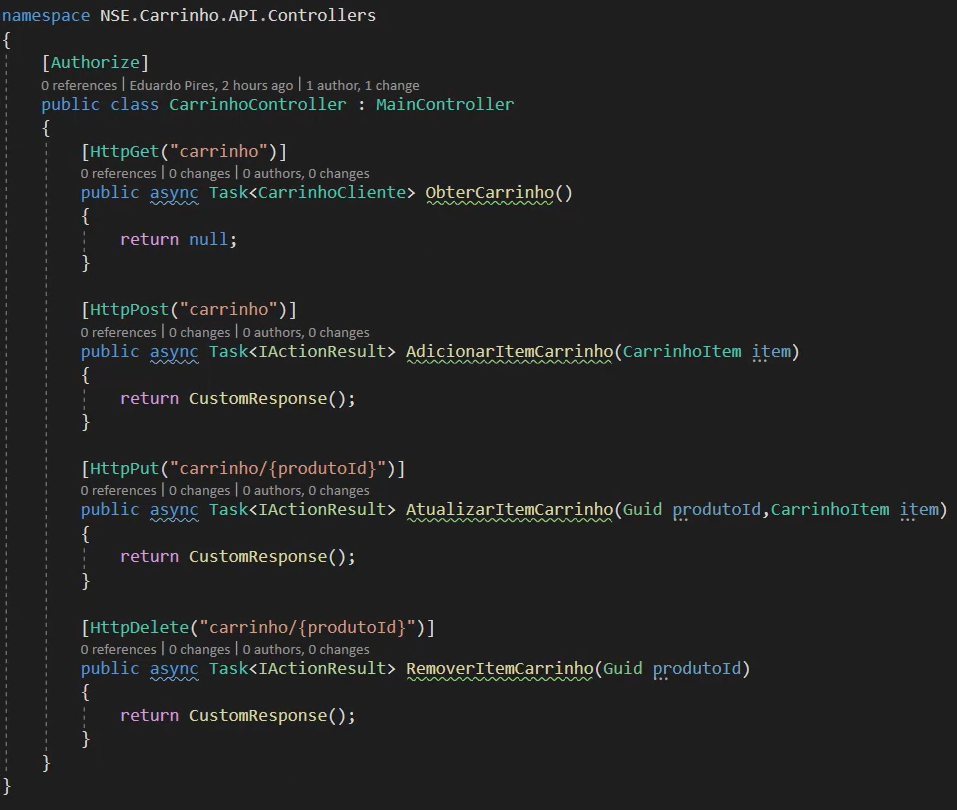
->Adcionar a migration com a start up Project especificada



->Realizar o update-database também com o startup Project especificado



-> Criar a controller com seus métodos



* **Globalização da Interface IUser**

->Em Building Blocks/Services/NSE.WebAPI.Core, criar uma pasta chamada Usuario

->Dentro dela, criar uma classe chamada AspNetUser

->Copiar tudo que dentro da classe Web/MVC/NSE.WebApp.MVC/Extensions/IUser e colar nessa classe criada anteriormente

->Substituir o nome da interface e da classe para AspNetUser

->Extrair a interface para um arquivo separado

->Extrair a classe ClaimsPrincipalExtensions para um arquivo separado

->Isso inicialmente quebrará a aplicação

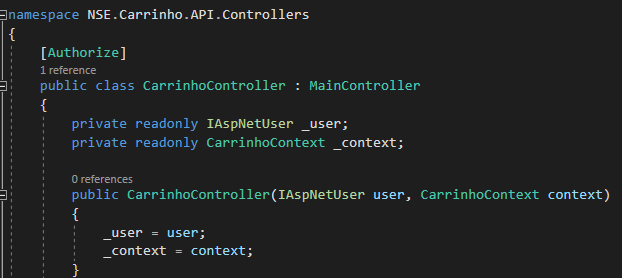
->Dar um build para identifica onde foi que quebrou

->Referenciar corretamente dessa vez já com a interface globalizada

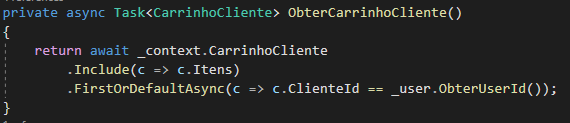
->Excluir a classe IUser da aplicação mvc(Extensions/IUser.cs)

* **Implementando Task ObterCarrinho**

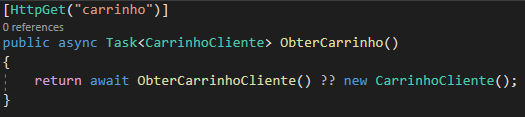
->Injetar o IAspNetUser e o CarrinhoContext na controller



->Criar o Método ObterCarrinhoCliente

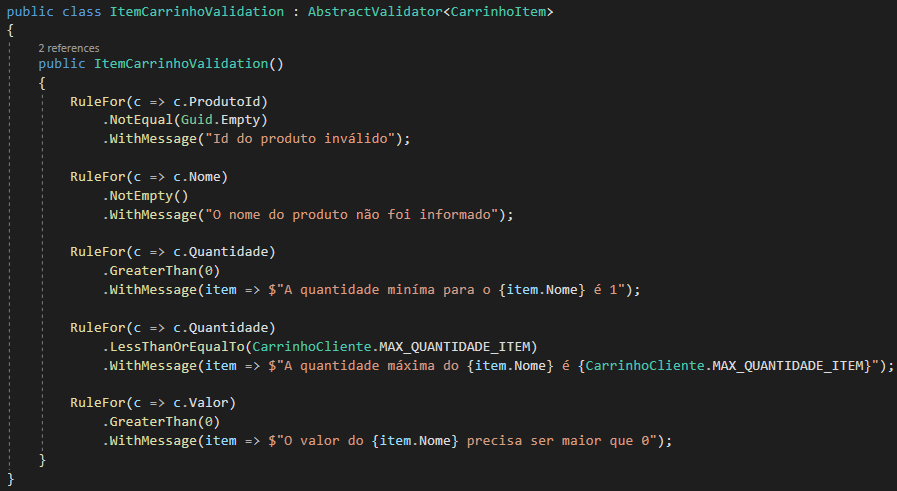


->Implementá-lo no método ObterCarrinho

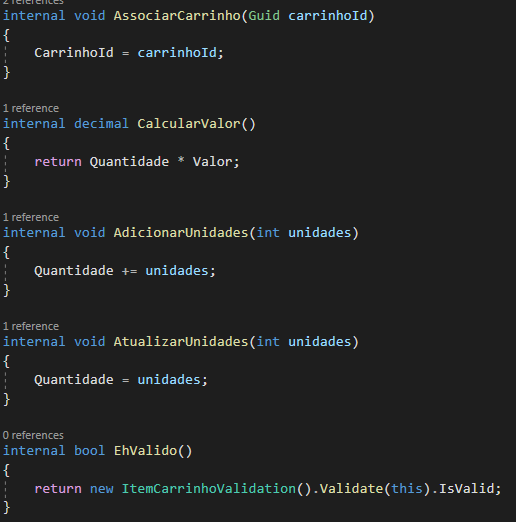


* **Criando métods base na classe CarrinhoItem**

->Criar uma classe de validação aninhada Chamada ItemCarrinhoValidation

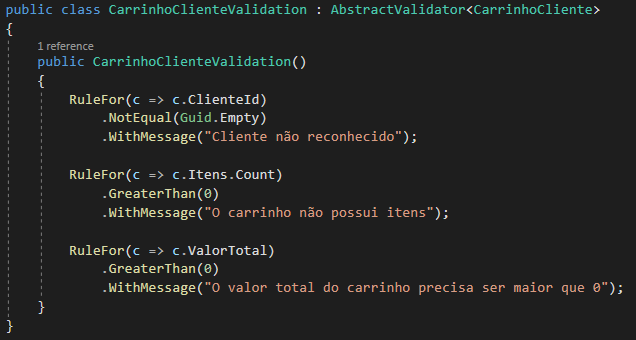


->Depois disso, criar os métodos base para incrementar os métodos da controller e da Classe CarrinhoController

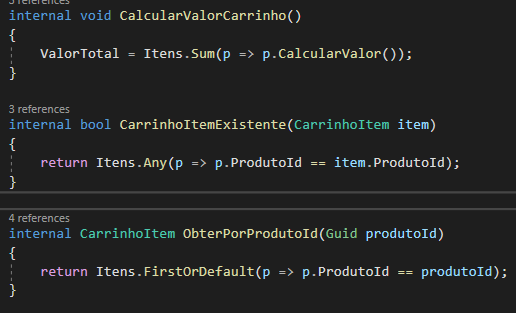


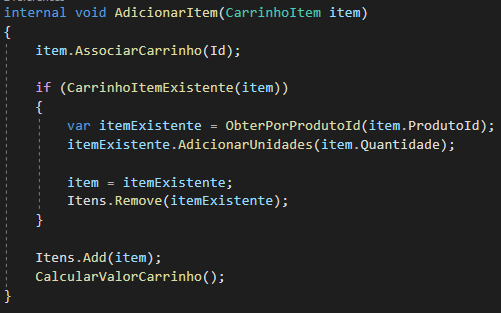
* **Criando métodos Base na classe CarrinhoCliente**

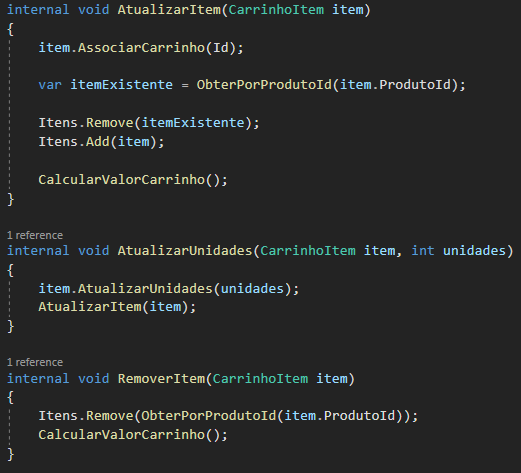
->Criar uma classe de validação aninhada Chamada CarrinhoClienteValidation

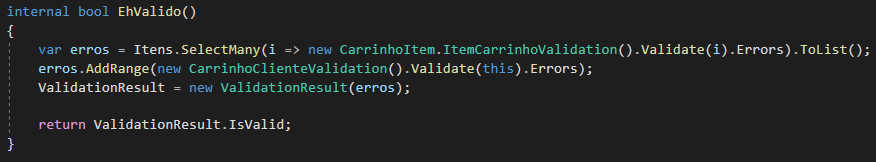


->Criar métodos base para serem utilizados na Controller









->Inicializar o ResponseResult e o ResponseErrosMessages na view. Isso fará que checagem de nulo seja feita corretamente

Texto

Descrição gerada automaticamente

Texto

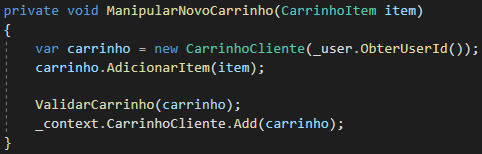
Descrição gerada automaticamente

->

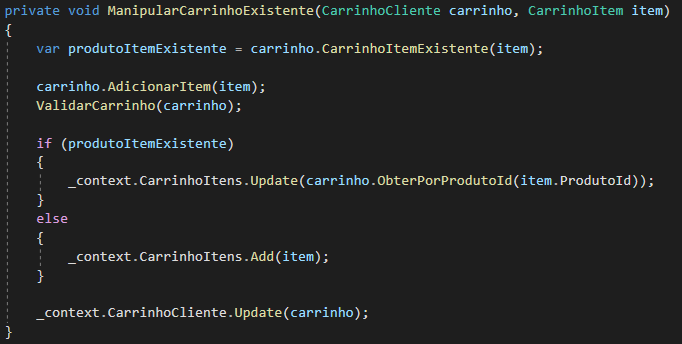
* **Implementando Task AdicionarItemCarrinho**

->Na controller, criar os métodos bases para Implementar os métodos

->Criar o método ManipularNovoCarrinho



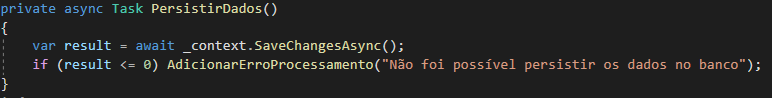
->Criar o método ManipularCarrinhoExistente



->Criar a devidas validações para implementar a Task AdicionarItemCarrinho

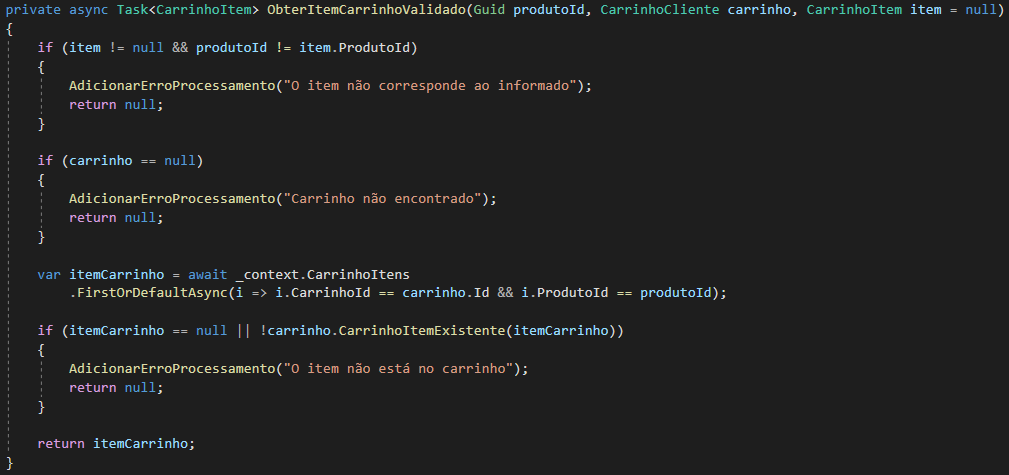


->Criar método PersistirDados

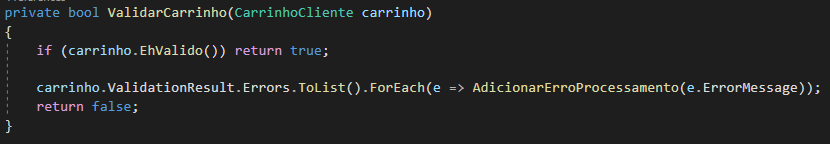


* **Implementando Task AtualizarItemCarrinho**

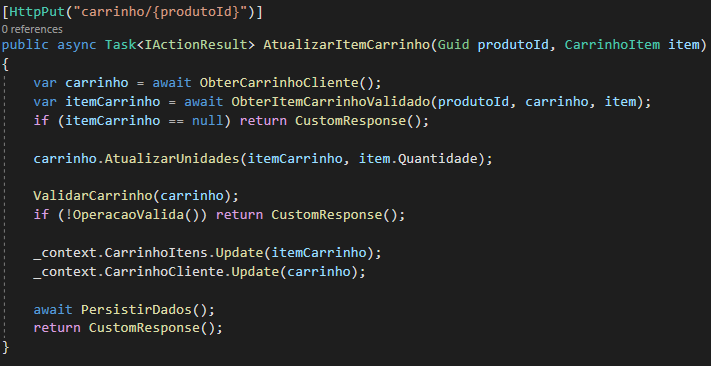
->Criar método ObterItemCarrinhoValidado



->Criar método validar Carrinho

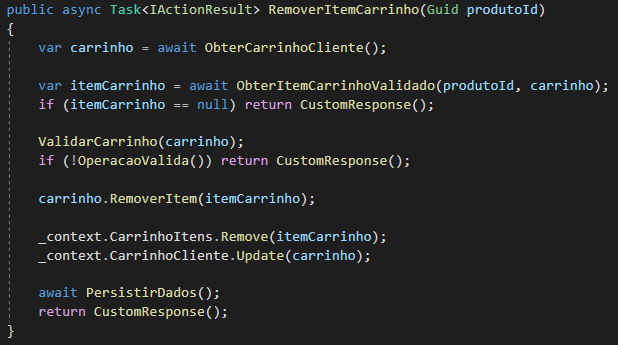


->Implementar Task AtualizarItemCarrinho



* **Implementando Task RemoverItemCarrinho**

->Implementar a Task de remover



* **Preparando o ambiente para a implementação do FrontEnd**

->Na pasta Models, criar a classe CarrinhosViewModel. Essa classe será uma réplica do da model la da api de carrinho

Texto

Descrição gerada automaticamente

->Em WebApp.MVC/Controllers, criar a CarrinhoController

->Dentro dessas Tasks será chamados serviços que implementarão os métodos lá da api de carrinho

Texto

Descrição gerada automaticamente

->Adicionar o endpoint carrinhoUrl no appSettings.Development da WebApp

Texto

Descrição gerada automaticamente

->Setar a variável CarrinhoUrl no em WebApp.Mvc/Extensions/AppSettings

Texto

Descrição gerada automaticamente

->Em WebApp/Services/Service, criar um novo método para retornar apenas um ok

Texto

Descrição gerada automaticamente

->Na pasta Service, criar a interface ICarrinhoService e escrever suas interfaces

Texto

Descrição gerada automaticamente

->Na mesma pasta, criar a classe CarrinhoService e implementá-la

->Injetar o httpcliente

->Gerar o Construtor

Texto

Descrição gerada automaticamente

->Implementar os métodos ObterCarrinho, AdicionarItemCarrinho, AtualizarItemCarrinho e RemoverItemCarrinho

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

->Em WebApp.Mvc/Views, criar uma pasta chamada carrinho e dentro dela criar a view Index

->Pegar o código fonte e realizar as alterações em cima dele

->Registrar o serviço em DependencyInjectionConfig

->Colocar tudo dentro de uma region

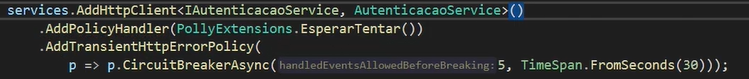
Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

->Aproveitar adicionar um pouco mais de resiliência no AutenticacaoSerivcice



* **Implementando a CarrinhoController**

->Injetar os serviços na controller

->Isso é feito para evitar passar as informações por input hidden no formulário. Caso seja feito desse jeito, esses dados são ficam aptos a serem facilmente manipulados no html.

Texto

Descrição gerada automaticamente

-> Na main controller, adicionar os Métodos AdicionarErroValidacao e OperacaoValida

Texto

Descrição gerada automaticamente

-> Na própria controller, criar o método ValidarItemCarrinho,

Texto

Descrição gerada automaticamente

->Na controller

->Implementar a task Index

Texto

Descrição gerada automaticamente

->Implementar a task AdicionarItemCarrinho

Texto

Descrição gerada automaticamente

->Implementar a task AtualizarItemCarrinho

Texto

Descrição gerada automaticamente

->Implementar a task RemoverItemCarrinho

Texto

Descrição gerada automaticamente

->Em views/Catalogo/ProdutoDetalhe, implementar a ponte para quando for adicionar o produto no carrinho. Para isso:

->Criar um formulário. Abrir nessa posição

Tela preta com letras brancas

Descrição gerada automaticamente

->Fechar nessa posição

Texto

Descrição gerada automaticamente

->Adicionar o ID como hidden field

Tela de celular com aplicativo aberto

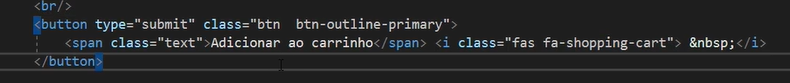
Descrição gerada automaticamente

->Adicionar o id e quantidade no select do produto. Com isso, essa informação conseguirá ser capturada

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

->Transformar o link do Adicionar ao Carrinho em um botão



* **Ajustes no FrontEnd**

->Realizar ajuste através do razor extensions

->Em WebApp/Extensions/RazorHelpers, adicionar o método UnidadesPorProduto

Texto

Descrição gerada automaticamente com confiança média

->Na mesma classe, criar outro método chamado SelectOptionsPorQuantidade

Texto

Descrição gerada automaticamente

->Na view index de Carrinho,

->Aplicar o método UnidadesPorProduto

Texto

Descrição gerada automaticamente

->Aplicar o método SelectOptionsPorQuantidade

Texto

Descrição gerada automaticamente

->Agora deve-se implementar o ícone do carrinho.

->Em WebApp.MVC/Extensions, criar CarrinhoViewComponent

Texto

Descrição gerada automaticamente

->Na pasta views/Shared/Components, criar uma pasta chamada Carrinho

->Dentro dela criar um Default.cshtml

Texto

Descrição gerada automaticamente

->Em Views/Shared/\_Layout.cshtml, substituir a section para que seja possível realizar o layout necessário

Texto

Descrição gerada automaticamente

->Na mesma pasta, no \_LoginPartial, também deve-se realizar a restruturação da div.

Texto

Descrição gerada automaticamente

-> Testar

->Lembrar que o rabbitmq tem de estar rodando em docker