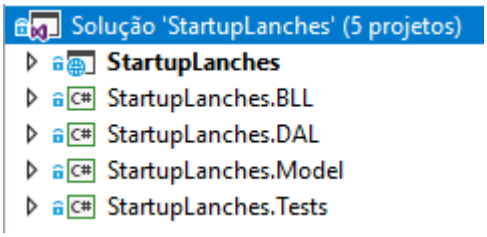
Justificativa de Design de Código

O projeto foi nomeado StartupLanches, e estruturado com o padrão MVC + padrão de 3 camadas e conta com testes de integração contínua. Ele foi desenvolvido com aspnet core, JQuery v2.2.0, Bootstrap v3.3.7, CSS e Razor Web Page para entrega do HTML. Os testes de integração contínua foram desenvolvidos usando o xUnit.

A solução do projeto foi desenvolvida usando o Visual Studio 15.5.6, na versão 2.0.0 do .NET Core, e se divide nos seguintes componentes:



Além das camadas de Visão, Modelo e Controle oferecidas pelo padrão MVC, a estrutura conta com outros 3 componentes com o intuito de isolar o acesso a dados e negócio do MVC, deixando-o responsável apenas por construir interfaces e gerenciar requisições. Portanto, as responsabilidades dos componentes ficam distribuídas da seguinte forma:

StartupLanches: Responsável por entregar conteúdo HTML e gerenciar requisições feitas pelos clientes HTTP, consultando a camada de negócios **StartupLanches.BLL** para obter e salvar dados.

StartupLanches.BLL: Responsável por gerenciar as regras de negócio do sistema e garantir que informações são salvas e obtidas de forma consistente.

StartupLanches.DAL: Responsável realizar a interface com o banco de dados, que neste caso é inexistente devido a condição apresentada no enunciado do desafio, onde o banco poderia ser guardado em memória;

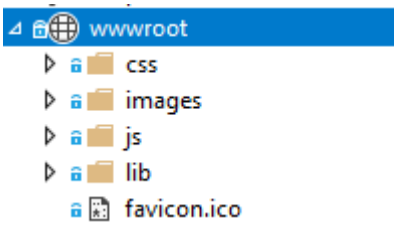
StartupLanches.Model: Responsável por oferecer todas as entidades de modelo do sistema, estas são trabalhadas em todas as camadas do projeto, desde a interface, até a camada de negócios e dados;

StartupLanches.Tests: Responsável por controlar a integração contínua do projeto, realizando testes para garantir que a camada de negócios está se comportando corretamente.

Detalhando um pouco mais cada camada:

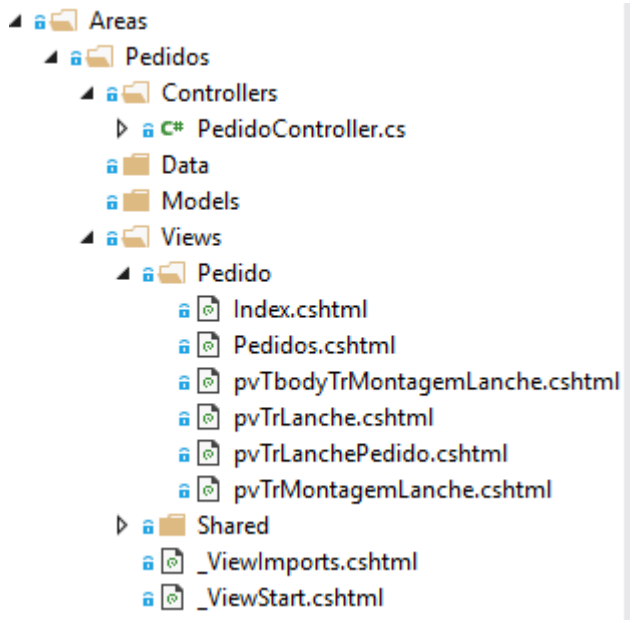
StartupLanches

O projeto conta com um diretório para entrega de conteúdo estático nomeado *wwwroot*:



Neste diretório é possível encontrar scripts javascript, css, imagens, gifs e etc. Sua existência é dada para desvincular a entrega deste conteúdo da aplicação, tornando possível distribui-los em outros servidores para redução de tráfego buscando maior agilidade no carregamento das páginas do sistema.

Este projeto faz uso do fornecimento de Areas, para melhor organizar as funcionalidades do sistema e permitir que ao longo de seu crescimento, as páginas html e controladoras fiquem fáceis de localizar e entender. No caso deste projeto, a Area Pedidos foi criada:



Para fazer uso desta área, a classe PedidoController teve de ter o atributo `[Area("Pedidos")]` mapeado em seu cabeçalho, e uma rota para sua interpretação mapeada no método Configure localizado em Startup.cs.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseBrowserLink();
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
          name: "PedidoAreaRoute",
          template: "{area:exists}/{controller=Pedido}/{action=Index}/{id?}"
        );

        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

Como a rota é atribuída à requisição baseada na ordem de mapeamento, a rota da Area deve ser adicionada antes da rota padrão para que possa ser identificada corretamente.

Nativamente, o projeto padrão do aspnet core MVC não oferece suporte a tratativa de erros para requisições Ajax, e, portanto, uma tratativa customizada para tais erros foi desenvolvida. O aspnet core oferece tratativas de exceção customizadas para classes que implementem a interface IExceptionFilter. Esta é a classe desenvolvida:

```
public class ExceptionHandlerFilter : IExceptionFilter
{
    0 referências | Augusto Gregório, há 11 horas | 1 autor, 1 alteração | 0 exceções
    public void OnException(ExceptionContext context)
    {
        //Identifica se é requisição Ajax, se for, retorna um JSON com os dados do erro, do contrário, passa.
        if (context.HttpContext.IsAjaxRequest())
        {
            var result = new ObjectResult(new
            {
                code = 500,
                message = context.Exception.Message,
                stackTrace = context.Exception.StackTrace,
                type = context.Exception.GetType().Name
            });

            result.StatusCode = 500;
            context.Result = result;
        }
    }
}
```

No corpo do método OnException, realizamos a verificação de requisição Ajax no HttpContext do contexto passado por parâmetro, caso seja, um objeto é construído e atribuído como retorno para o cliente Http.

Após a implementação da interface na classe, ela deve ser registrada meio aos serviços do MVC no Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(typeof(ExceptionHandlerFilter));
    });
```

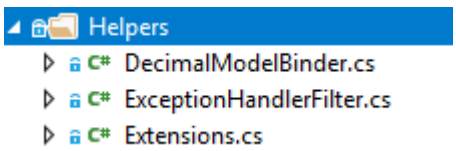
Esta implementação se justifica para os casos de erro ao fazer uma requisição Ajax, seja para carregar um Json ou obter uma pequena parcela de HTML de conteúdo dinâmico, onde por padrão, o MVC retorna uma página HTML completa contendo os dados do erro, quando na verdade se espera um Json ou PartialView. Portanto, a classe ExceptionHandlerFilter é responsável por identificar se a requisição que gerou exceção é Ajax, se for, um objeto com dados da exceção é criado para retornar para o cliente.

Este é o trecho de código desenvolvido em Javascript – Jquery responsável por fazer uso desta tratativa:

```
$(document).ajaxStart(function () {
    $("body").append("<div class='carregando'></div>");
});

$(document).ajaxStop(function () {
    $("div.carregando:first").remove();
});

$(document).ajaxComplete(function () {
    blockRequest = false;
});

$(document).ajaxError(function (e, xhr) {
    $("div.carregando:first").remove();
    try {
        var ex = JSON.parse(xhr.responseText);
        alert(ex.message + " - " + ex.type);
    }
    catch (err) {
        console.log(err.message);
        console.log(xhr);
    }
});
```

Sempre que um erro ocorrer em uma requisição Ajax, a função atribuída em $(document).ajaxError será executada, e o tipo criado no ExceptionHandlerFilter será enviado e a mensagem será exibida corretamente. No caso acima, o tipo da exceção disparada é enviado também, possibilitando diferenciação no formato das mensagens de erros exibidas. Para este projeto, a exceção criada para erros de negócio é a BusinessException. Mais sobre ela será mencionado na camada StartupLanches.BLL.

A classe ExceptionHandlerFilter está localizada na pasta Helpers do Projeto, junto de outras classes:



Veremos agora o conteúdo das classes Extensions e DecimalModelBinder.

A classe Extensions nada mais é do que uma classe estática responsável por oferecer métodos de extensão usados por outras áreas do sistema. Para o projeto, apenas um método foi desenvolvido, e o mesmo é usado na classe ExceptionHandlerFilter para identificar a requisição Ajax:

```
public static class Extensions
{
    1 referência | Augusto Gregório, há 12 horas | 1 autor, 1 alteração | 0 exceções
    public static bool IsAjaxRequest(this HttpContext context)
    {
        if (context == null || context.Request == null)
            throw new ArgumentNullException(nameof(context));
        if (context.Request.Headers["X-Requested-With"] == "XMLHttpRequest")
            return true;
        return false;
    }
}
```

No código, checamos se nos cabeçalhos da requisição existe algum com a chave “X-Requested-With” com valor “XMLHttpRequest”, se possuir, é uma requisição Ajax e retorna true, caso contrário retorna false.

Sobre o DecimalModelBinder, sua justificativa é dada devido ao uso do tipo decimal para o armazenamento de valores das entidades de modelo, nativamente, o ModelBinder padrão não atende nossas necessidades, portanto a classe foi implementada. Para sua implementação, duas classes foram criadas, um model binder provider e um model binder em si, sendo eles:

InvariantDecimalModelBinderProvider, que implementa a interface IModelBinderProvider:

```
public class InvariantDecimalModelBinderProvider : IModelBinderProvider
{
    0 referências | Augusto Gregório, há 12 horas | 1 autor, 1 alteração | 0 exceções
    public IModelBinder GetBinder(ModelBinderProviderContext context)
    {
        if (context == null) throw new ArgumentNullException(nameof(context));

        if (!context.Metadata.IsComplexType
         && (context.Metadata.ModelType == typeof(decimal)
         ||  context.Metadata.ModelType == typeof(decimal?)))
        {
            return new InvariantDecimalModelBinder(context.Metadata.ModelType);
        }

        return null;
    }
}
```

E InvariantDecimalModelBinder, que implementa a interface IModelBinder:

```
public Task BindModelAsync(ModelBindingContext bindingContext)
{
    if (bindingContext == null) throw new ArgumentNullException(nameof(bindingContext));

    var valueProviderResult = bindingContext.ValueProvider.GetValue(bindingContext.ModelName);

    if (valueProviderResult != ValueProviderResult.None)
    {
        bindingContext.ModelState.SetModelValue(bindingContext.ModelName, valueProviderResult);

        var valueAsString = valueProviderResult.FirstValue;
        decimal result;

        // Use invariant culture
        if (decimal.TryParse(valueAsString,
            NumberStyles.AllowDecimalPoint
          | NumberStyles.AllowLeadingSign,
            CultureInfo.CurrentCulture,
            out result))
        {
            bindingContext.Result = ModelBindingResult.Success(result);
            return Task.CompletedTask;
        }
    }

    // If we haven't handled it, then we'll let the base SimpleTypeModelBinder handle it
    return _baseBinder.BindModelAsync(bindingContext);
}
```

No código, extraímos o valor atribuído ao objeto decimal que desejamos realizar o bind, se ele for válido, o extraímos como texto e realizamos o parse para decimal, se for bem-sucedido, o valor é atribuído, do contrário, não é.

Assim como na tratativa de erros Ajax, precisamos registrar o ModelBinder na inicialização do MVC para que ele possa ser executado corretamente nas requisições feitas pelos clientes. O registro também é feito em Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc(options =>
    {
        options.Filters.Add(typeof(ExceptionHandlerFilter));
    });
    services.AddMvc(config =>
    {
        config.ModelBinderProviders.Insert(0, new InvariantDecimalModelBinderProvider());
    });

}
```

Para o lado do cliente, um script Javascript foi desenvolvido com o nome Utils.js. Este contém funcionalidades genéricas para uso do sistema, além de possuir a tratativa de requições Ajax mencionadas anteriormente. Uma das funcionalidades oferecida pelo objeto StartupLanches, que possui um sub objeto Utils, que por sua vez contém dois atributos: Confirm e Ok, que são usados para exibir mensagens para o usuários utilizando Modals do Bootsrap:

```
var StartupLanches =
    {
        Utils:
        {
            Confirm: function (titulo, mensagem, callback_nao, callback_sim)...,

            Ok: function (titulo, mensagem, callback_ok)...
        }
    };
```

O HTML consumido por estas funções fica ao final do _LayoutPedido.cshtml, localizado na Area Pedidos:

```
<div class="modal fade" id="modal-message" role="dialog">
    <div class="modal-dialog">
        <!-- Modal content-->
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal">&times;</button>
                <h4 class="modal-title"></h4>
            </div>
            <div id="modal-body-mensagem" class="modal-body">
                <p></p>
            </div>
            <div class="modal-footer">
                <button id="btNao" style="display: none;" title="Não" type="button" class="btn btn-danger">
                    <span class="glyphicon glyphicon-"></span> Não
                </button>
                <button id="btSim" style="display: none;" title="Sim" type="button" class="btn btn-success">
                    <span class="glyphicon glyphicon-"></span> Sim
                </button>
                <button id="btOk" style="display: none;" title="Sim" type="button" class="btn btn-info">
                    <span class="glyphicon glyphicon-check"></span> Ok
                </button>
            </div>
        </div>
    </div>
</div>
```

Dependendo do método acionado, os botões são exibidos ou ocultados.

O _LayoutPedido.cshtml contém uma seção de Scripts localizada ao final de seu conteúdo. Este é responsável por delegar a renderização de scripts fornecidos pelas sub Views. Neste projeto, apenas a View Index.cshtml faz uso, onde sua declaração é feita logo no início:

```
@section Scripts
    {
    <script type="text/javascript">
        var Url = new Object();
        Url.TableMontagemLanche = "@Url.Action("TableMontagemLanche")";
        Url.TrMontagemLanche = "@Url.Action("TrMontagemLanche")";
        Url.CalcularPromocoes = "@Url.Action("CalcularPromocoes")";
        Url.AdicionarLanchePedido = "@Url.Action("AdicionarLanchePedido")";
        Url.ConfirmarPedido = "@Url.Action("ConfirmarPedido")";
    </script>
    <script src="~/js/Areas/Pedidos/Pedido/Index.js" asp-append-version="true"></script>
}
```

Ou seja, este conteúdo será construído onde a declaração `@RenderSection("Scripts", required: false)` foi feita no _LayoutPedido.cshtml. (O objeto Url construído na imagem acima é utilizado para realizar as requisições do cliente, seja ela ajax ou de nova página. O Método Url.Action é responsável por construir a Url de acesso as ações da controladora. Se nenhuma controladora for acionada na chamada de Url.Action, a controladora padrão será considerada.)

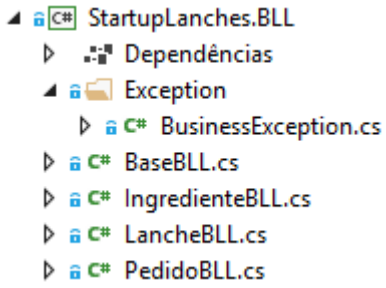
Para construir os campos de quantidades com botões de incremento usados na tela, uma extensão foi adicionada ao JQuery para tornar simplificar sua criação:

```
jQuery.fn.extend({
    MinusPlusInputify: function ()...
});

var preparar_inputs = function (inputGroups)...
```

Devido a quantidade de código presente nestes métodos, não foi possível colar uma imagem, mas seu conteúdo pode ser checado no arquivo localizado em ~/js/Utils.js.

StartupLanches.BLL

O projeto com 5 classes, uma para a classificação de exceções de negócio, a BusinessException, e outras 4 responsáveis para tratar o negócio em si, onde uma delas é a BaseBLL, e as outras 3 são as classes de negócio para Ingrediente, Lanche e Pedido.



A classe BusinessException herda funcionalidades da Classe Exception e oferece a opção de formatar a mensagem de erro em sua declaração:

```
class BusinessException : System.Exception
{
    7 referências | Augusto Gregório, há 3 dias | 1 autor, 1 alteração | 0 exceções
    public BusinessException(string mensagem, params object[] args) : base(string.Format(mensagem, args))
    {

    }
}
```

A classe BaseBLL é abstrata, e foi criada pois oferece serviços que serão usados pelas demais classes de negócio.

```
public abstract class BaseBLL
{
    11 referências | Augusto Gregório Helena, há 4 dias | 1 autor, 1 alteração | 0 exceções
    protected static DataBase dataBase { get; set; }
    1 referência | Augusto Gregório Helena, há 4 dias | 1 autor, 1 alteração | 0 exceções
    public bool IsConfigured
    {
        get
        {
            return dataBase != null;
        }
    }
    3 referências | Augusto Gregório Helena, há 4 dias | 1 autor, 1 alteração | 0 exceções
    protected BaseBLL()
    {
        if (!IsConfigured)
        {
            dataBase = new DataBase();
            dataBase.PrepareDatabase();
        }
    }
}
```

A instancia estática do banco em memória é gerenciada por esta classe, onde caso não tenha sido iniciada ainda, ela faz a chamada de configuração do banco.

As demais classes de negócio herdam da classe BaseBLL e implementam seus próprios métodos baseados no negócio. Seus conteúdos podem ser checados na raiz deste projeto.

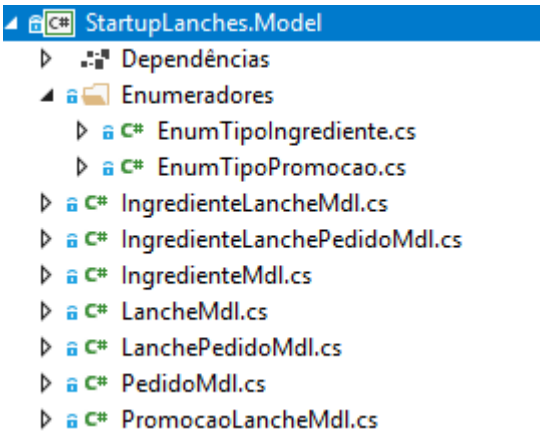
StartupLanches.DAL

Este é o projeto de menor complexidade de toda a solução, pois não houve necessidade em implementar uma persistência real em banco de dados, portanto, esta conta apenas com um método de preparação estática de alguns dados e a declaração de cada uma de suas tabelas em memória, que nada mais são do que Lists simples.

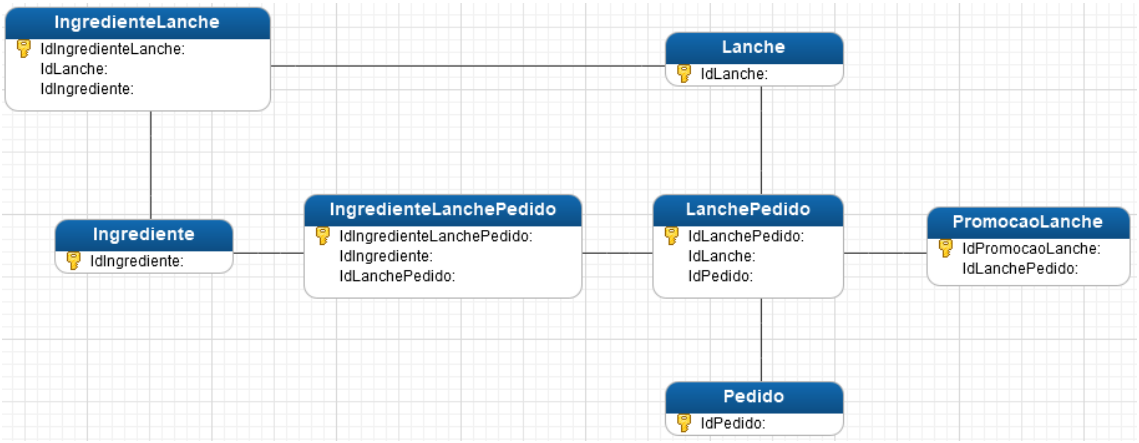
O método PrepareDatabase cria os dados de acordo com o descrito no enunciado do desafio e instancia todas as tabelas virtuais usadas pela BLL.

StartupLanches.Model

Este projeto contém todos os modelos e enumeradores usados pelo sistema, e estão organizados da seguinte forma:



O relacionamento entre as classes foi implementado de acordo com o modelo de entidade relacional abaixo, pois foi desenvolvido com um banco relacional em mente. (PS: As tabelas não contêm todos os atributos apenas por simplicidade, uma vez que não existe banco):



As tabelas Lanche, IngredienteLanche e Ingrediente se relacionam para armazenar as informações de lanches do cardápio. As tabelas LanchePedido, IngredienteLanchePedido e Ingrediente se relacionam para armazenar as informações dos lanches e seus nomes e preços no momento da compra para armazenamento de histórico. A tabela Pedido e LanchePedido se relacionam para armazenar informações do pedido e quantos lanches pertencem em cada pedido. A tabela PromocaoLanche se relaciona com LanchePedido para armazenar qual desconto foi dado e a qual promoção ele pertence.

As multiplicidades podem ser explicadas da seguinte forma:

- Pedido
  - Deve ter 1 ou N LanchesPedido;
- LanchePedido
  - Pode ter 0 ou N PromocoesLanche;
  - Pode ter 0 ou 1 Lanche;
  - Deve ter 1 ou N Ingrediente, gerando a tabela de relacionamento IngredienteLanchePedido;
- Lanche
  - Deve ter 1 ou N Ingrediente, gerando a tabela de relacionamento IngredienteLanche;

Para melhor classificar as informações de Ingrediente e Promocao, foram criados os enumeradores EnumTipoIngrediente e EnumTipoPromocao:

```
public enum EnumTipoIngrediente : short
{
    Nenhum = 0,
    Carne = 1,
    Queijo = 2,
    Vegetal = 3,
    Tempero = 4,
    Outros = 9,
}
```

Usado para classificar os tipos de Ingredientes e poder identifica-los nos cálculos de promoções.

```
public enum EnumTipoPromocao
{
    Light,
    MuitoQueijo,
    MuitaCarne
}
```

Usado para classificar a promoção.

StartupLanches.Tests

Este projeto possui os testes de comportamento das regras de negócio do sistema. Ele conta com 5 métodos de teste:

```
[Fact]
| 0 referências | Augusto Gregório, há 3 dias | 1 autor, 1 alteração | 0 exceções
public void TesteValorSemPromocao()...

[Fact]
| 0 referências | Augusto Gregório, há 3 dias | 1 autor, 1 alteração | 0 exceções
public void TestePromocaoMuitaCarne()...

[Fact]
| 0 referências | Augusto Gregório, há 3 dias | 1 autor, 1 alteração | 0 exceções
public void TestePromocaoMuitoQueijo()...

[Fact]
| 0 referências | Augusto Gregório, há 3 dias | 1 autor, 1 alteração | 0 exceções
public void TestePromocaoLight()...

[Fact]
| 0 referências | Augusto Gregório, há 18 horas | 1 autor, 2 alterações | 0 exceções
public void TesteValorComPromocao()...
```

Cada um deles testa um trecho da regra de negócio voltada para a precificação dos lanches com e sem promoções. Abaixo descrevo como cada teste se comporta:

TesteValorSemPromocao: Este teste cria um LanchePedido e adiciona 5 ingredientes com valores fixos, ele então soma a quantidade x valor de cada ingrediente e compara com o valor obtido do método CalcularValor encontrado na LancheBLL;

TestePromocaoMuitaCarne: Este teste cria um LanchePedido e adiciona 1 ingrediente do tipo carne com quantidade 2. Em seguida solicita o CalcularValor e CalcularPromocao na respectiva ordem e certifica de que a promoção não foi ativada. Em seguida, mais um ingrediente do tipo carne é adicionado com a quantidade 1 e os métodos CalcularValor e CalcularPromocao são executados novamente para se certificar que ao menos uma promoção deveria existir, e que esta deve ser do tipo EnumTipoPromocao.MuitaCarne.

TestePromocaoMuitoQueijo: Este teste cria um LanchePedido e adiciona 1 ingrediente do tipo queijo com quantidade 2. Em seguida solicita o CalcularValor e CalcularPromocao na respectiva ordem e certifica de que a promoção não foi ativada. Em seguida, mais um ingrediente do tipo queijo é adicionado com a quantidade 1 e os métodos CalcularValor e CalcularPromocao são executados novamente para se certificar que ao menos uma promoção deveria existir, e que esta deve ser do tipo EnumTipoPromocao.MuitoQueijo.

TestePromocaoLight: Este teste cria um lanche com duas porções de bacon, alface e hambúrguer, faz a chamada dos métodos CalcularValor e CalcularPromocao e se certifica de que nenhuma promoção foi ativada. Em seguida, ele remove o bacon do lanche e faz a chamada dos métodos CalcularValor e CalcularPromocao, então ele se certifica de que existe uma promoção do tipo EnumTipoPromocao.Light e compara o valor de soma dos ingredientes advindo do método com o cálculo feito no próprio teste, certificando de que ambos são iguais.

TesteValorComPromocao: Este teste faz um compilado de todos os testes anteriores onde o valor é validado para cada critério de promoção ativado. Inicialmente o valor sem promoções é testado, em seguida o valor é testado após atribuir 4 para a quantidade do ingrediente Queijo, depois o mesmo é feito para o Hambúrguer de carne, e por último o bacon é retirado e então se verifica o desconto de 10% aplicado. Se todas as contas forem iguais, o teste passa.

Considerações

Quanto ao negócio: Tive algumas dúvidas referentes a como o sistema deveria se comportar, mas não obtive uma resposta, e por isso decidi implementar algumas das regras da forma que julguei mais correta. São elas:

Cálculo de promoções para Muita Carne: no enunciado do desafio, destaca-se: "*A cada 3 porções de carne o cliente só paga 2. Se o lanche tiver 6 porções, o cliente pagará 4. Assim por diante...*". Contudo, nada se fala sobre tipos diferentes de carne, no caso, o bacon e hambúrguer, e como a regra deve se comportar contendo os dois. Portanto, esta promoção é aplicada individualmente para cada ingrediente, ou seja, se tiver 3 porções de bacon, apenas 2 serão cobradas, se possuir 3 porções de hambúrguer, apenas duas serão cobradas, se conter 2 de bacon e 1 de hambúrguer, nenhuma promoção será ativada.

Cálculo de promoção Light: Quanto ao desconto de 10% oferecido nas opções Light, escolhi aplicar a regra **antes** de aplicar todas as promoções, ou seja, se a somatória de ingredientes de um lanche é R$ 10,00 e contém um montante de descontos de promoção de R$ 2,00, o desconto será de R$ 1,00 e não de R$ 0,80, pois se aplica em cima da somatória de ingredientes.

Quanto ao formato deste documento: Nunca havia escrito um relatório descrevendo um projeto antes, por isso, me desculpo se esqueci de algum detalhe que venha a ter faltado e me disponho a tirar qualquer dúvida que venha a existir.

Para a criação do projeto, foi necessário aprender sobre um pouco de aspnet core, que até então não havia tido contato, portanto, este desafio acabou sendo um aprendizado ao longo de seu desenvolvimento. O projeto todo foi desenvolvido do zero, e não conta com nenhum framework desenvolvido de antemão.

Outro ponto, devido a minha disponibilidade e prazo para a entrega das atividades, não pude fazer tudo o que gostaria para a entrega deste projeto, portanto, ele não está tão polido como gostaria que estivesse, mas acredito que será possível avaliar meus conhecimentos com o desenvolvimento.