FOME ZERO

Matheus Diniz Marchi

Esse documento visa explicar os motivos para a escolha do design do código.

A solução está dividido em quatro projetos:

- FomeZero.Entidades
- FomeZero.Negocio
- FomeZero.Testes
- FomeZero.Web

FomeZero.Entidades: Esse projeto consiste nas entidades utilizadas no sistema. A pasta de "Dominio" que se encontra nesse projeto além de simular os dados vindos do Banco de Dados como o nome dos ingredientes, nome dos lanches e valores de desconto de promoções e de ingredientes, também complementa a organização do código de maneira que os valores e os nomes fiquem concentrados em um único ponto.

FomeZero.Negocio: Esse projeto consiste na camada de negócio da aplicação. Aqui podemos ver métodos para obter o cardápio da loja e montar um lanche personalizado. A separação desse projeto como camada de negócio facilita possíveis evoluções futuras. Se quisermos adicionar uma camada de serviços que fará a comunicação entre a aplicação Front-end (seja ela qual for) com a camada de negócio o código já está separado não havendo necessidade de refatoração.

FomeZero.Testes: Projeto de testes que visa garatir as regras de negócio dando uma segurança grande ao fazer novas implementações e evoluções.

FomeZero.Web: Projeto que implementa uma aplicação Web simples para visualização dos produtos vendidos pela loja, além de exibir as regras de negócio também.

Comecei o desenvolvimento do exercício garantindo as regras de negócio por meio dos testes automatizados, dessa maneira ficaria transparente caso as futuras alterações que eu precisasse realizar no código não condizerem com a regra de negócio especificada, pois o teste acusaria falha.

Para realizar os testes automatizados, utilizei de um técnica chamada TDD (Test Driven Development). Essa técnica, como o próprio nome diz, visa o desenvolvimento de código baseado e dirigido por testes e possui três passos simples a serem seguidos, os quais são: Escrita do teste de maneira que o mesmo venha a falhar, Evolução do código da maneira mais simples possível para que o teste possa ter sucesso e ,por fim, Refatoração para melhorar o código.

Essa técnica me permitiu garantir as regras de negócio de uma maneira muito simples e rápida.

```
#region [ Valida preço do X-Egg ]

/// <summary>
/// Critério de aceite: Valor do lanche X-Egg deve ser 5,30 sem inflação
/// </summary>
[TestMethod]
public void ValidaValorLancheXEgg()
{
    //Cria lista de ingredientes
    List<Ingrediente> listaIngredientes = new List<Ingrediente>
    {
        new Ingrediente { Nome = NomeIngrediente.Ovo },
        new Ingrediente { Nome = NomeIngrediente.HamburguerCarne },
        new Ingrediente { Nome = NomeIngrediente.Queijo }
    };

    Lanche xEgg = new Lanche();
    xEgg.Ingredientes = listaIngredientes;

    Assert.AreEqual(ExecutaCalculo(listaIngredientes), xEgg.Preco);
}

#endregion
```

Ilustração 1: Exemplo de método de teste automatizado

Em seguida comecei o desenvolvimento do Front-end em ASP .NET MVC, o que me permitiu uma

construção rápida e fluída dos recursos necessários para exibição das funcionalidades e regras de negócio da aplicação.

Utilizei o jQuery para implementar o comportamento de Accordion visto na tela de Cardápio e para facilitar o desenvolvimento do lado Front-end. No arquivo **FomeZeroUtil.js** extendi o *prototype* de *string* para que nos arquivos *JavaScript* das páginas não fosse repetido o id do elemento, desse modo o nome do elemento está concentrado na página *HTML*. Isso evita que o mesmo nome seja em string seja repetido diversas vezes no mesmo arquivo. Um exemplo pode ser visto na página em que montamos um lanche personalizado.

```
String.prototype.jqueryId = function () {
    return '#' + this;
};
```

Ilustração 2: Extensão da string – Arquivo FomeZeroUtil.js

```
@section scripts{

    <script type="text/javascript">

        local = {

            Id: {

                CampoIngredienteAlface: '@Html.IdFor(m => m.Alface)'.jqueryId(),
                CampoIngredienteBacon: '@Html.IdFor(m => m.Bacon)'.jqueryId(),
                CampoIngredienteHamburguerCarne: '@Html.IdFor(m => m.HamburguerCarne)'.jqueryId(),
                CampoIngredienteOvo: '@Html.IdFor(m => m.Ovo)'.jqueryId(),
                CampoIngredienteQueijo: '@Html.IdFor(m => m.Queijo)'.jqueryId(),
                FormularioMontarLanchePersonalizado: '@LanchePersonalizadoFormulario.Salvar'.jqueryId(),
                BotaoMontarLanche: 'botaoMontarLanche'.jqueryId(),
            }
        }

    </script>

    @Scripts.Render("~/Areas/LanchePersonalizado")
}
```

Ilustração 3: Utilização da extensão na página web de montagem de lanches personalizados

```
//Aplica máscara nos campos
function AplicarMascaras() {

    $(local.Id.CampoIngredienteAlface).mask('00');
    $(local.Id.CampoIngredienteBacon).mask('00');
    $(local.Id.CampoIngredienteHamburguerCarne).mask('00');
    $(local.Id.CampoIngredienteOvo).mask('00');
    $(local.Id.CampoIngredienteQueijo).mask('00');
}
```

Ilustração 4: Utilização dos nomes no JavaScript da página web de montagem de lanches personalizados

A separação das entidades de negócio das entidades da tela se dá pois na exibição em uma página Web não precisamos de todas as informações que as entidades de negócio possuem. Para realizar o mapeamento das entidades de tela para as entidades de negócio e vice-versa, utilizei o *AutoMapper*, uma biblioteca para realizar esse mapeamento de atributos de entidades de uma maneira simples e elegante.