Curso: Dominando Testes de Software

Sumário

[01 – Introdução 5](#_Toc68212824)

[01-01 – Apresentação 5](#_Toc68212825)

[02 - A importância de testar 5](#_Toc68212826)

[02-01 - Por que testar? 5](#_Toc68212827)

[Como garantir a qualidade do software? 5](#_Toc68212828)

[02-02 - Como testar? 5](#_Toc68212829)

[Caixa-preta 6](#_Toc68212830)

[Caixa-branco 6](#_Toc68212831)

[Como testar? 6](#_Toc68212832)

[Baixo Nível 6](#_Toc68212833)

[Alto Nível 6](#_Toc68212834)

[“Quadrante mágico” do Teste Ágil 6](#_Toc68212835)

[Consequências de não testar 6](#_Toc68212836)

[02-03 - Regra 10 de Myers 5](#_Toc68212837)

[02-04 - Mitos sobre os testes de software 5](#_Toc68212838)

[02-05 - Tipos de testes mais comuns 5](#_Toc68212839)

[Teste de Unidade 6](#_Toc68212840)

[Teste te integração 6](#_Toc68212841)

[Teste automatizado 6](#_Toc68212842)

[Teste de carga 6](#_Toc68212843)

[03 - Testes de Unidade 6](#_Toc68212844)

[03-01 - Objetivos 6](#_Toc68212845)

[03-02 - Frameworks de testes 7](#_Toc68212846)

[XUnit.net 7](#_Toc68212847)

[03-03 - Padrões e Nomenclaturas 7](#_Toc68212848)

[AAA - Arrange, Act, Assert 7](#_Toc68212849)

[Nomenclatura de Testes de unidade 7](#_Toc68212850)

[03-04 - A importância do Mock 7](#_Toc68212851)

[O que é um Mock? 8](#_Toc68212852)

[MOQ - Framework de Mock 8](#_Toc68212853)

[03-05 - O básico sobre testes de unidade 8](#_Toc68212854)

[03-06 - Asserções (Assert) 8](#_Toc68212855)

[03-07 - Utilizando Traits 8](#_Toc68212856)

[03-08 - Utilizando Fixtures 8](#_Toc68212857)

[Collection Fixtures 9](#_Toc68212858)

[03-09 - Ordenação de testes 8](#_Toc68212859)

[03-10 - Gerando dados humanos 8](#_Toc68212860)

[03-11 - Realizando mock de objetos 9](#_Toc68212861)

[03-12 - Simulando retorno de métodos com Mock 9](#_Toc68212862)

[03-13 - Utilizando AutoMock 9](#_Toc68212863)

[03-14 - Utilizando Fluent Assertions 9](#_Toc68212864)

[03-15 - Escapando de testes 9](#_Toc68212865)

[03-16 - Mensagens de saída nos testes 9](#_Toc68212866)

[03-17 - Playlists de testes 9](#_Toc68212867)

[03-18 - Rodando os testes via linha de comando 9](#_Toc68212868)

[03-19 - Trabalhando com testes no Visual Studio Code 9](#_Toc68212869)

[03-20 - Analisando a cobertura de código dos testes 9](#_Toc68212870)

[04 - TDD - Test Driven Development 9](#_Toc68212871)

[04-10 – Introdução 9](#_Toc68212872)

[Ciclo do TDD: 9](#_Toc68212873)

[04-02 - Por que utilizar TDD? 9](#_Toc68212874)

[As três leis do TDD 9](#_Toc68212875)

[04-03 - Comece com Baby Steps 9](#_Toc68212876)

[Dicas para começar com TDD 10](#_Toc68212877)

[Teste Driven Design? 10](#_Toc68212878)

[Leituras Indicadas 10](#_Toc68212879)

[04-04 - Setup do projeto 10](#_Toc68212880)

[04-05 - Pedido - Adicionar Item - Parte 1 10](#_Toc68212881)

[04-06 - Pedido - Adicionar Item - Parte 2 10](#_Toc68212882)

[04-07 - Pedido - Adicionar Item - Parte 3 10](#_Toc68212883)

[04-08 - Pedido - Adicionar Item - Parte 4 10](#_Toc68212884)

[04-09 - Pedido - Atualizar Item 10](#_Toc68212885)

[04-10 - Pedido - Remover Item 11](#_Toc68212886)

[04-11 - Pedido - Aplicar Voucher - Parte 1 11](#_Toc68212887)

[04-12 - Pedido - Aplicar Voucher - Parte 2 11](#_Toc68212888)

[04-13 - Pedido - Aplicar Voucher - Parte 3 11](#_Toc68212889)

[04-14 - Pedido - Adicionar Item Command 11](#_Toc68212890)

[04-15 - Pedido - Command Handler - Parte 1 11](#_Toc68212891)

[04-16 - Pedido - Command Handler - Parte 2 11](#_Toc68212892)

[04-17 - Pedido - Command Handler - Parte 3 11](#_Toc68212893)

[04-18 - Refatorando Testes 11](#_Toc68212894)

[04-19 - Code Coverage - Considerações Finais 11](#_Toc68212895)

[05 - Testes de Integração 11](#_Toc68212896)

[05-01 - Apresentação - Cenário de testes 11](#_Toc68212897)

[05-02 - Propósito dos testes de integração 12](#_Toc68212898)

[05-03 - Configurações necessárias 12](#_Toc68212899)

[05-04 - Registro do usuário - pt1 12](#_Toc68212900)

[05-05 - Registro do usuário - pt2 12](#_Toc68212901)

[05-06 - Registro do usuário - pt3 12](#_Toc68212902)

[05-07 - Pedidos - Web - pt1 12](#_Toc68212903)

[05-08 - Pedidos - Web - pt2 12](#_Toc68212904)

[05-09 - Pedidos - API - pt1 12](#_Toc68212905)

[05-10 - Pedidos - API - pt2 12](#_Toc68212906)

[Palavras Finais / Diga 12](#_Toc68212907)

[06 - BDD - Behavior Driven Development 12](#_Toc68212908)

[06-01 - Sobre o que é? 12](#_Toc68212909)

[Ciclo de Desenvolvimento 13](#_Toc68212910)

[06-02 - User Stories 13](#_Toc68212911)

[User Stories 13](#_Toc68212912)

[Cenários 13](#_Toc68212913)

[06-03 - Ferramentas e configuração 13](#_Toc68212914)

[SpecFlow 14](#_Toc68212915)

[06-04 - Escrevendo user stories com SpecFlow 13](#_Toc68212916)

[06-05 - Gerando os testes de comportamento 13](#_Toc68212917)

[07 - Testes Automatizados 13](#_Toc68212918)

[07-01 - Apresentação 13](#_Toc68212919)

[07-02 - Apresentando o Selenium 13](#_Toc68212920)

[07-03 - Configurando o Selenium da forma certa 14](#_Toc68212921)

[07-04 - Customizando o uso do Selenium 14](#_Toc68212922)

[07-05 - Automatizando a criação do WebDriver do Selenium 14](#_Toc68212923)

[07-06 - POM - Page Object Model 14](#_Toc68212924)

[07-07 - Automatizando a interface de cadastro de usuário 14](#_Toc68212925)

[07-08 - Automatizando a interface de cadastro e login 14](#_Toc68212926)

[07-09 - Automatizando a interface de carrinho de compras 14](#_Toc68212927)

[07-10 - Execução headless via comando e considerações finais 14](#_Toc68212928)

[08 - Testes de Carga 15](#_Toc68212929)

[08-01 - Tipos de testes de carga 15](#_Toc68212930)

[Teste de desempenho 15](#_Toc68212931)

[Teste de carga 15](#_Toc68212932)

[Teste de Stress 15](#_Toc68212933)

[08-02 - Ferramentas disponíveis no mercado 15](#_Toc68212934)

[08-03 - Índice de satisfação do usuário com APDEX 16](#_Toc68212935)

[Medindo o incide de satisfação do cliente om APDEX 16](#_Toc68212936)

[Classificação do “Response time” 16](#_Toc68212937)

[08-04 - Introdução ao JMeter 16](#_Toc68212938)

[08-05 - Plano de testes completo com JMeter e BlazeMeter 16](#_Toc68212939)

[08-06 - Teste os seus conhecimentos 16](#_Toc68212940)

[09 - Encerramento 16](#_Toc68212941)

[09-01 - Palavras Finais 16](#_Toc68212942)

# 01 – Introdução

## 01-01 – Apresentação

...

# 02 - A importância de testar

## 02-01 - Por que testar?

Testar é o processo d executar um programa ou sistema com a intenção de encontrar defeitos (teste negativo). – Myers, 1979

Testar é qualquer atividade que, a partir da avaliação e um atributo ou capacidade, permita determinar se o programa ou sistema obtém resultados desejados. – Hetzel, 1988

### Como garantir a qualidade do software?

A certificação por terceiros envolve três principais:

* Normas
  + ISO
* Órgãos Certificadores
  + InMetro
* Organismos Credenciadoras
  + Sistema de Gestão em Tecnologia da Informação – OTI
  + Sistemas de Gestão Ambiental - OCA

## 02-02 - Como testar?

Testes de Caixa-preta e Caixa Branca

### Caixa-preta

* Feitos no Nível de Interface da Aplicação
* Os testadores executam testes de função e regressão por meio da interface do usuário

### Caixa-branco

* Feitos no Nível de Código
* Os testadores (ou desenvolvedores) olham diretamente para o código para encontrar bugs e erros.

### Como testar?

* Functional Testing - Funcional
* Unit Testing - De Unidade
* Exploratory Testing - Exploratório
* Add-hoc Testing
* Integration Testing 0 Integração
* UAT (User Acceptance Testing) – Aceitação do Usuário
* Usability Testing - De usabilidad
* Alpha Testing –
  + é um tipo de teste de aceitação; realizada para identificar todos os possíveis problemas e bugs antes de liberar o produto final para os usuários finais
* Beta Testing
  + é realizado por "usuários reais" do aplicativo de software em "ambiente real" e pode ser considerado uma forma de Teste de Aceitação do Usuário externo
* Compilance Testing – De conformidade

Testes

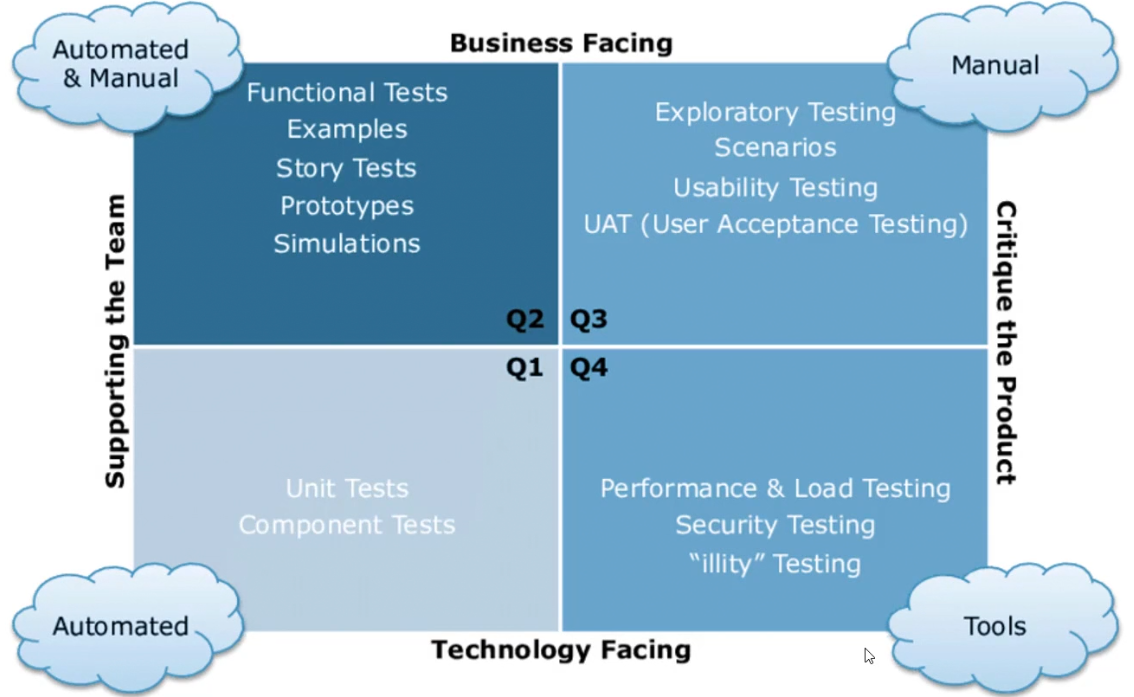
### Baixo Nível

* Análise estática de código
* Programação par a par
* Testes de Unidade
* Testes de Integração entre componentes

### Alto Nível

* Testes Exploratórios
* Testes Não funcionais
  + Funcionalidade
  + Confiabilidade
  + Usabilidade
  + Eficiência
  + Manutenibilidade
  + Portabilidade
* Testes de Integração do Sistema
* Testes de Aceitação / Story Testes

### “Quadrante mágico” do Teste Ágil



### Consequências de não testar

* Muitos bugs e problemas de qualidade
* Falhas de codificação
* Possíveis atrasos na entrega
* Perda de confiança do cliente
* Desmotivação do Time
* Aumento da rotatividade de pessoas
* Prejuízos financeiros
* Possíveis implicações legais

## 02-03 - Regra 10 de Myers

* A Regra 10 de Myers, foi criada por Glenford Myers, um dos pioneiros na área em testes de software
* Esta regra estabelece que o custo da correção de defeitos é bem mais custoso quanto mais tarde o defeito é encontrado.
* Corrigir um defeito encontrado em produção custa ~muito mais carto do que se fosse encontrado na fase de análise ou desenvolvimento.
* Glenford Myers, além de contribuir com o conceito da "Regra 10 de Myers", foi também o autor do livro The art of software testing, lançado em 1979, este livro tornou-se uma das bíblias da qualidade de software

## 02-04 - Mitos sobre os testes de software

* Eu não tenho tempo de testar!
* Testar é escrever o dobro de código
* Se fizer o código certinho não precisa testar
* Testar é papel do testador
* Que adianta eu testar se o resto da equipe não testa?
* Os testes garantem 100% de software livre de erros
* Se passou nos testes é por que não tem bug
* Eu já terminei. Só falta testar
* Teste de unidade só funcionam se for com TDD
* Testar manualmente é mais produtivo
* Não consigo convencer meu gerente!

## 02-05 - Tipos de testes mais comuns

* Teste de Unidade
* Testes de Integração
* Testes Automatizados
* Testes de Carga

### Teste de Unidade

* Um teste de unidade é aquele que testa uma única unidade do sistema. Ele a testa de maneira isolada, geralmente simulando as prováveis dependências que aquela unidade tem.
* Em sistemas orientados a objetos, é comum que a unidade seja uma classe.
* Quando queremos escrever testes de unidade para a classe ex *Pedido*, essa bateria de testes testará o funcionamento da classe *Pedido*, isolada, sem interações com outras classes.

### Teste te integração

* Um Teste de integração é aquele que testa a integração entre duas partes do seu sistema.
* Os testes que você escreve para a sua classe *PedidoService* e *PedidoRepository*, por exemplo, onde seu teste vai até o banco de dados, é um teste de integração.
* De fato, você está testando a integração da sua aplicação com suas diversas partes, por exemplo o banco de dados, uma camada ou um serviço externo.
* Estes testes garantes algo que os testes de unidade não garantem, eles garantem que as unidades da sua aplicação estão se integrando conforme o esperado.

### Teste automatizado

* O teste automatizado é um teste de sistema, podemos considerar como um teste de aceitação, funciona como um teste de caixa preta, já que o sistema é testado de ponta a ponta através das operações executadas no sistema.
* A automatização garante que todo o processa será executado da mesma maneira que fosse um usuário utilizando a aplicação.
* Pode ser considerado um teste de aceitação, pois este teste além de garantir o funcionamento ponta a ponta, podemos validar características do negócio e funcionalidades.
* Pode ser considerado um teste de regressão, pois sua execução garante que a aplicação não regrediu, ou seja, que não seguiu novos defeitos em componentes que já estavam funcionamento nas versões anteriores.

### Teste de carga

* O teste de carga (*load test*) consiste em testar as capacidades da aplicação, muitas vezes até seu limite, de forma que a aplicação não consiga mais responder. Pode ser considerado como um teste de stress.
* Existe objetivos por trás do teste de carga:
  + Testar a performance do código e componente em situações extremas. Descobrir possíveis gargalos.
  + Testar como a aplicação irá se comportar com um determinado número de usuários simultâneos e garantir que irá atender a demanda.
  + Testar um possível balanceamento de carga no servidor ou até mesmo a escava elástica na nuvem.
* O teste de carga é muito importante em aplicações Web e em aplicações que é esperado um grande volume de requisições.

Qual dos 4 tipos de testes é o melhor?

# 03 - Testes de Unidade

## 03-01 - Objetivos

* Frameworks
* Nomenclaturas
* Ferramentas
* Escrita de testes
* Features do framework de teste
* Execução no VS, VSCode, Command Line
* Mock
* AutoMock
* Fixture
* Geração de dados
* Fluent Assertions

## 03-02 - Frameworks de testes

Frameworks para Teste de Unidade

* **MSTest**
  + Da microfost
  + <https://github.com/microsoft/testfx>
* **NUnit** (Portabilidade do JUnit do Java)
  + Famoso JUnit
  + <https://nunit.org/>
* **XUnit**
  + Imão mais novo do Nunit
  + <https://xunit.net/>

### XUnit.net

* Escrito pelos criados do NUnit
* Compatível com o Visual Studio
* Utilizado pelo time de desenvolvimento do .NET Core e ASP.NET

Instarão do framework

Install-Package xunit

Instalação para reconhecimento do Visual Studio

Install-Package xunit.runner.visualstudio

## 03-03 - Padrões e Nomenclaturas

### AAA - Arrange, Act, Assert

* Arrange – Arranjar (preparar os dados que serão testados)
* Act – Atuação - A execução do teste
* Assert – Asserção – Validar se o resultado confere com a intenção dos testes

### Nomenclatura de Testes de unidade

**ObjetoEmTeste\_MetodoComportamentoEmTeste\_ComportamentoEsperado**

* Pedido\_AdicionarPedidoItem\_DeveIncrementarUnidadesSeItemJaExistente
* Estoque\_RetirarItem\_DeveEnviarEmailSeAbaixoDe10Unidades

**MetodoEmTeste\_EstadoEmTeste\_ComportamentoEsperado**

* AdicionarPedidoItem\_ItemExisteCarrinho\_DeveIncrementarUnidadesDoItem
* RetirarItemEstoque\_EstoqueAbaixoDe10Unidades\_DeveEnviarEmailDeAviso

## 03-04 - A importância do Mock

### O que é um Mock?

* Objetos Mock, objetos simulados ou simplesmente Mock (do inglês Mock Object) são objetos que simulam o comportamento de objetos reais de forma controlada;
* São normalmente criados para testar o comportamento de outros objetos. Em outras palavras, os objetos mock são objetos “falsos” que simulam o comportamento de uma classe ou objeto “real” para que possamos focam o teste na unidade a ser testada.
* Uma vantagem do Mock é que o objeto simulado pode ser criado dinamicamente através de um framework de Mock e poupando o desenvolvedor de ter que criar uma classe física para simular aquele objeto.
* Uma classe física que simula o objeto costuma ser chamada de Fake, mas na teoria é um Mock, a diferença é que foi criada manualmente.

### MOQ - Framework de Mock

* Realize Mock com facilidade
* Compatível com VisualStudio
* Utilizado pelo time de desenvolvimento do .NET Core e ASP.NET

Instalação do framework

Install-Package Moq

Link: <https://github.com/Moq/moq4>

## 03-05 - O básico sobre testes de unidade

*Código...*

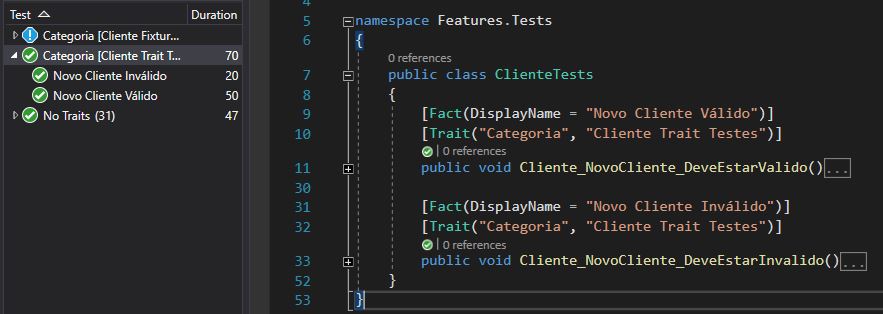
## 03-06 - Asserções (Assert)

* Strings
  + Equal
  + Contains
  + StartsWith
  + EndsWith
  + Matches
* Number
  + Equal
  + NotEqual
* NullBool
  + False
  + True
  + Null
* Ranges
  + InRange
  + NotInRange
* ObjectTypes
  + IsType
  + IsAssignableFrom
* Collections
  + All
  + Contains
  + DoesNotContain
  + Equal
* Exceptions
  + Throws

## 03-07 - Utilizando Traits

Em resumo: Categorização / Organização do Código

Exemplo:



## 03-08 - Utilizando Fixtures

Em resumo:

* Contexto compartilhado entre testes
* Reaproveitamento de classes

Tipos:

* **Class Fixtures**
  + instância de objeto compartilhado entre os testes em uma única classe
* **Collection Fixtures**
  + instâncias de objetos compartilhados em várias classes de teste

Link: <https://xunit.net/docs/shared-context>

### Collection Fixtures

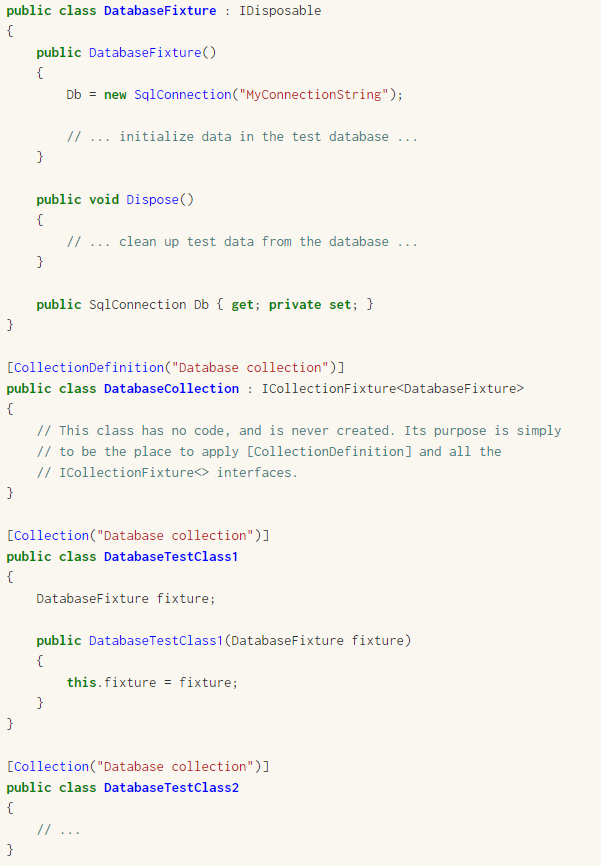
**Quando usar**: quando você deseja criar um único contexto de teste e compartilhá-lo entre os testes em várias classes de teste, e limpá-lo após a conclusão de todos os testes nas classes de teste.

Às vezes, você desejará compartilhar um objeto de fixação entre várias classes de teste. O exemplo de banco de dados usado para fixtures de classe é um ótimo exemplo: você pode querer inicializar um banco de dados com um conjunto de dados de teste e, em seguida, deixar esses dados de teste no local para uso por várias classes de teste. Você pode usar o recurso de *collection fixture* de xUnit.net para compartilhar uma única instância de objeto entre os testes em várias classes de teste.

Para usar *collection fixture*, você precisa seguir os seguintes passos:

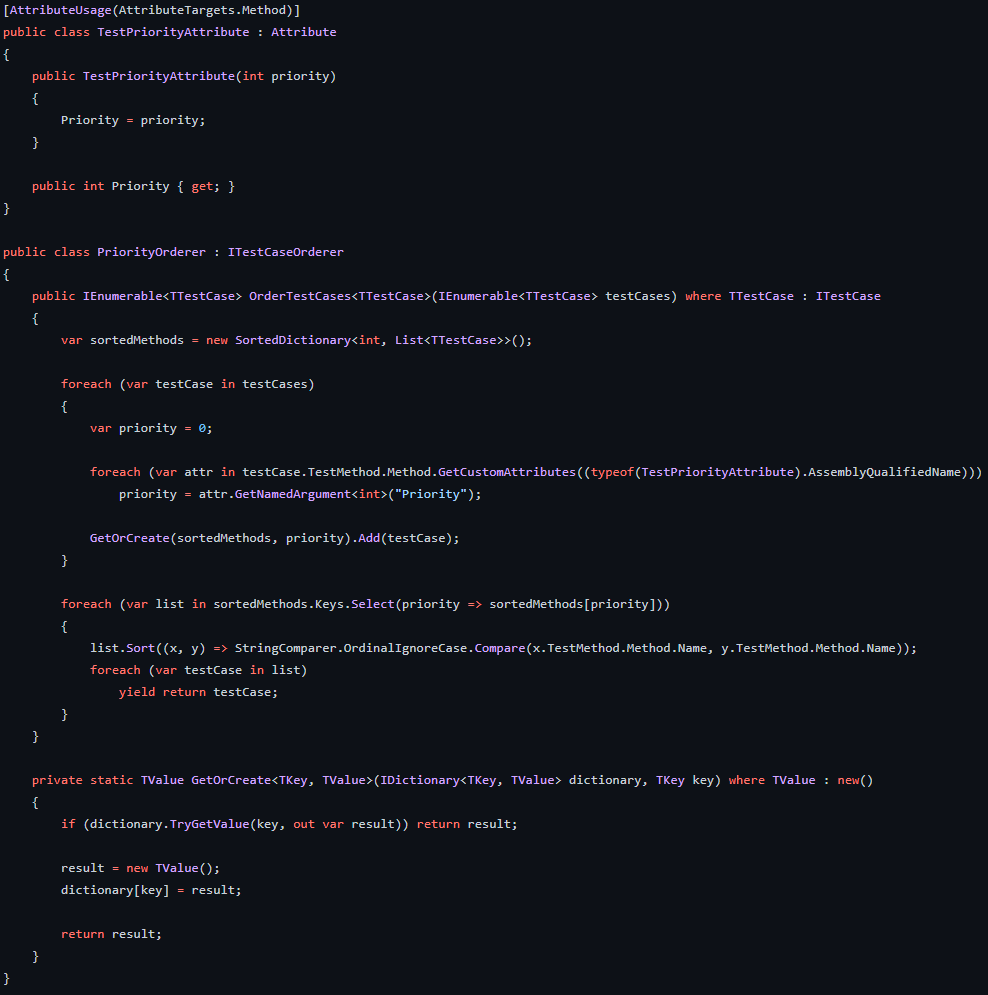
* Crie a classe fixture e coloque o código de inicialização no construtor da classe fixture.
* Se a classe de fixture precisar realizar uma limpeza, implemente IDisposable na classe de fixture e coloque o código de limpeza no Dispose() método.
* Crie a classe de definição de coleção, decorando-a com o atribulo [CollectionDefinition], dando a ela um nome único que identificará a coleção de teste.
  + *Esta classe não possui código e nunca é criada.*
  + *Seu objetivo é simplesmente ser o local para aplicar [CollectionDefinition] e todas as interfaces ICollectionFixture <>.*
* Adicione ICollectionFixture<> à classe de definição de coleção.
* Adicione o atributo [Collection] a todas as classes de teste que farão parte da coleção, usando o nome exclusivo que você forneceu ao  atributo da classe de definição da coleção de teste [CollectionDefinition].
* Se as classes de teste precisam de acesso à instância do fixture, adicione-o como um argumento do construtor, e ele será fornecido automaticamente.

Aqui está um exemplo simples:



xUnit.net trata os acessórios de coleção da mesma maneira que os acessórios de classe, exceto que o tempo de vida de um objeto de acessório de coleção é mais longo: ele é criado antes que qualquer teste seja executado em qualquer uma das classes de teste na coleção e não será limpo até que todas as classes de teste na coleção tenham concluído a execução.

## 03-09 - Ordenação de testes



## 03-10 - Gerando dados humanos

**Bogus**

Link:

<https://github.com/bchavez/Bogus>

Instalação:

Install-Package Bogus

## 03-11 - Realizando mock de objetos

**Moq**

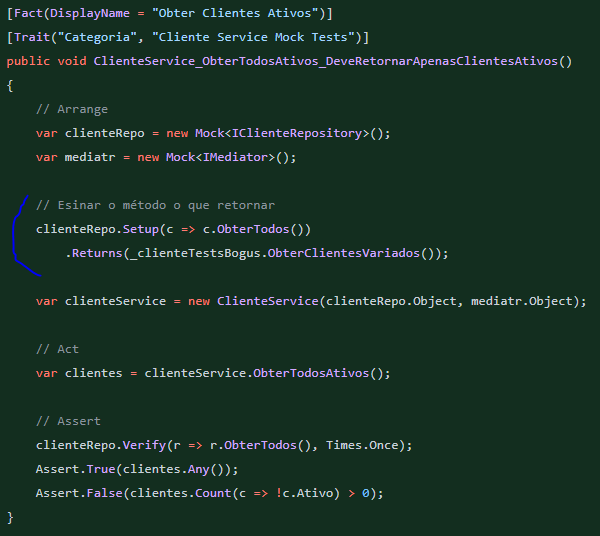
Link:

<https://github.com/Moq/moq4/wiki/Quickstart>

Instalação:

Install-Package MOQ

## 03-12 - Simulando retorno de métodos com Mock



## 03-13 - Utilizando AutoMock

Um contêiner de automocking para Moq.

Use isso se você investiu em seu contêiner IoC e deseja desacoplar seus testes de unidade das alterações em seus argumentos de construtor.

Link:

<https://github.com/moq/Moq.AutoMocker>

Instalação

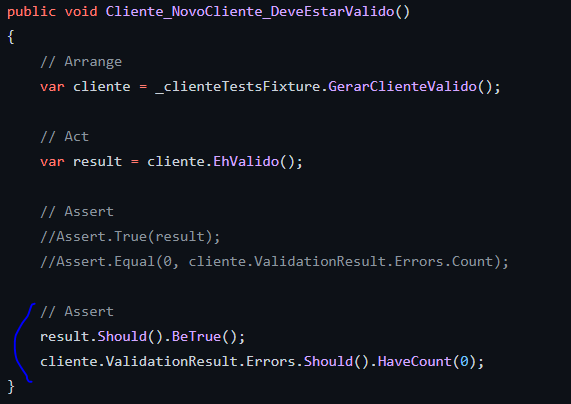
Install-Package MOQ.automock

## 03-14 - Utilizando Fluent Assertions

**Fluent Assertions**

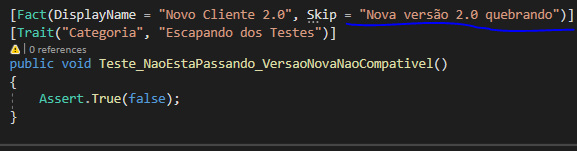
<https://fluentassertions.com/>

Install-Package FluentAssertions



## 03-15 - Escapando de testes

Uma forma de, sabendo que um teste vai dar erro, pular durante a execução!



## 03-16 - Mensagens de saída nos testes

Injetar a interface ITestOutputHelper

Escrever com o método WriteLine()

## 03-17 - Playlists de testes

...

## 03-18 - Rodando os testes via linha de comando

Instalação do pacote,

Install-Packge xunit.runner.console

- Fazer o Publish do Projeto

Abrir o Terminal no diretório do Publish e executar o comando:

dotnet vstest Features.Tests.dll

## 03-19 - Trabalhando com testes no Visual Studio Code

Abrir o VSCode

Instalar o plugin **.NET Core Test Explorer**

## 03-20 - Analisando a cobertura de código dos testes

https://docs.microsoft.com/pt-br/dotnet/core/testing/unit-testing-code-coverage

**OpenCover** (opção free)

* OpenCover é uma ferramenta de cobertura de código para .NET 2 e superior (sistemas operacionais Windows apenas - sem MONO) \* , com suporte para processos de 32 e 64 e abrange pontos de ramificação e sequência. O OpenCover foi iniciado depois que as tentativas de fazer o PartCover suportar processos de 64 bits se tornaram um pouco complicadas.

Link: <https://github.com/OpenCover/opencover>

# 04 - TDD - Test Driven Development

## 04-10 – Introdução

Em resumo, testar o código que ainda não existe. Ou seja, criar primeiro o teste e depois o código.

### Ciclo do TDD:

1. Vermelho – Escreva um teste e assista falhar
2. Verde – Escrever o mínimo de código para passar no teste
3. Refatorar – Refatore, melhore e teste novamente

## 04-02 - Por que utilizar TDD?

Escrever o teste antes do código?

### As três leis do TDD

1. Você não pode escrever nenhum código até ter escrito um teste que detecte uma possível falha.
2. Você não pode escrever mais testes de unidade do que o suficiente para detectar a falha – não compilar não é ter efeito.
3. Você não pode escrever mais código do que o suficiente para passar nos testes.

## 04-03 - Comece com Baby Steps

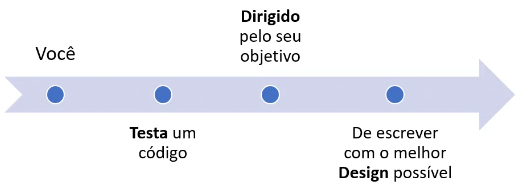
* É indicado para desenvolvedores que estão iniciando na prática do TDD.
* É utilizado quando não se tem muita confiança sobre o código a ser escrito e testado.
* Não existe uma prática exata sobre a utilização baby steps, a recomendação é fazer um processo por vez, identificar as necessidades avaliar o cenário e continuar.

### Dicas para começar com TDD

* Pratique os baby steps e respeite as 3 leis do TDD
* Saiba exatamente o que precisa entregar antes de começar
* Não deixe para descobrir a regra de negócio durante a codificação
* Tenha uma lista de anotações ou uma documentação clara sempre a mão
* Minta! No “green” o teste só precisa passar, depois utilize Mocks
* Mantenha os testes simples
* Após a escrita de alguns testes, analise a possibilidade de reaproveitamento de objetos.
* Saiba quando usar TDD, nem sempre é vantajoso

### Teste Driven Design?

* O conceito de TDD pode confundir, mas não é sobre escrever testes e sim sobre escrever código com um bom design. (Código escrito de forma certa é fácil de testar).
* Escrever testes depois do código também ajuda, mas não pode ocasionar retrabalho caso seja necessário modificar um código para possibilitar o teste.



### Leituras Indicadas

* Test-Driven Development (Casa do Código)
* Teste Drivem Development (Kent Back)
* Growing Object-Oriented Software, Guided By tests (Steve Freeman Nat Pryce)
* Pratical Test-Driven Development Using C#7 (Clayton Hunt, John Callaway)

## 04-04 - Setup do projeto

Requisitos

**DESENVOLVIMENTO DO DOMINIO DE VENDAS**

**PEDIDO - ITEM PEDIDO – VOUCHER**

Um item de um pedido representa um produto e pode conter mais de uma unidade Independente da ação, um item precisa ser sempre valido:

* Possuir: Id e Nome do produto, quantidade entre 1 e 15 unidades, valor maior que 0

Um pedido enquanto não iniciado (processo de pagamento) está no estado de rascunho e deve pertencer a um cliente.

1. 1 - Adicionar Item
   1. Ao adicionar um item é necessário calcular o valor total do pedido
   2. Se um item já está na lista então deve acrescer a quantidade do item no pedido
   3. O item deve ter entre 1 e 15 unidades do produto
2. - Atualização de Item
   1. O item precisa estar na lista para ser atualizado
   2. Um item pode ser atualizado contendo mais ou menos unidades do que anteriormente
   3. Ao atualizar um item é necessário calcular o valor total do pedido
   4. Um item deve permanecer entre 1 e 15 unidades do produto
3. - Remoção de Item
   1. - O item precisa estar na lista para ser removido
   2. Ao remover um item é necessário calcular o valor total do pedido

Um voucher possui um código único e o desconto pode ser em percentual ou valor fixo

Usar uma flag indicando que um pedido teve um voucher de desconto aplicado e o valor do desconto gerado

1. 4. Aplicar voucher de desconto
   1. O voucher só pode ser aplicado se estiver válido, para isto:
      1. Deve possuir um código
      2. A data de validade é superior a data atual
      3. O voucher está ativo
      4. O voucher possui quantidade disponível
      5. Uma das formas de desconto devem estar preenchidas com valor acima de 0
   2. Calcular o desconto conforme tipo do voucher
      1. Voucher com desconto percentual
      2. Voucher com desconto em valores (reais)
   3. Quando o valor do desconto ultrapassa o total do pedido o pedido recebe o valor: 0
   4. Após a aplicação do voucher o desconto deve ser re-calculado após toda modificação da lista de itens do pedido

**PEDIDO COMMANDS - HANDLER**

O command handler de pedido irá manipular um command para cada intenção em relação ao pedido.

Em todos os commandos manipulados devem ser verificados:

* Se o command é válido
* Se o pedido existe
* Se o item do pedido existe

Na alteração de estado do pedido:

* Deve ser feita via repositório
* Deve enviar um evento

1. 1 - AdicionarItemPedidoCommand
   1. Verificar se é um pedido novo ou em andamento
   2. Verificar se o item já foi adicionado a lista

## 04-05 - Pedido - Adicionar Item - Parte 1

*Código..*

## 04-06 - Pedido - Adicionar Item - Parte 2

*Código...*

## 04-07 - Pedido - Adicionar Item - Parte 3

*Código...*

## 04-08 - Pedido - Adicionar Item - Parte 4

*Código..*

## 04-09 - Pedido - Atualizar Item

*Código..*

## 04-10 - Pedido - Remover Item

*Código...*

## 04-11 - Pedido - Aplicar Voucher - Parte 1

*Código...*

## 04-12 - Pedido - Aplicar Voucher - Parte 2

*Código...*

## 04-13 - Pedido - Aplicar Voucher - Parte 3

*Código...*

## 04-14 - Pedido - Adicionar Item Command

*Código..*

## 04-15 - Pedido - Command Handler - Parte 1

*Código...*

## 04-16 - Pedido - Command Handler - Parte 2

*Código...*

## 04-17 - Pedido - Command Handler - Parte 3

*Código...*

## 04-18 - Refatorando Testes

*Código...*

## 04-19 - Code Coverage - Considerações Finais

# 05 - Testes de Integração

Implementação das migrations do projeto

Abrir o Package Manager Console, selecionar o projeto `NerdStore.Domain.Data` executar os comandos:

add-migration Initial -Context CatalogoContext

update-database -Context CatalogoContext

Abrir o Package Manager Console, selecionar o projeto `NerdStore.Vendas.Data` executar os comandos:

add-migration Initial -Context VendasContext

update-database -Context VendasContext

Artigo: <https://medium.com/@RobertKhou/asp-net-core-mvc-identity-using-postgresql-database-bc52255f67c4>

## 05-01 - Apresentação - Cenário de testes

Configuração, escrita e execução de testes de integração

## 05-02 - Propósito dos testes de integração

*Código...*

## 05-03 - Configurações necessárias

Artigo: <https://docs.microsoft.com/pt-br/aspnet/core/test/integration-tests>

*Código...*

## 05-04 - Registro do usuário - pt1

*Código...*

## 05-05 - Registro do usuário - pt2

*Código...*

## 05-06 - Registro do usuário - pt3

*Código...*

## 05-07 - Pedidos - Web - pt1

AngleSharp

* é a melhor biblioteca de analisador de colchetes angulares. Ele analisa HTML5, CSS3 e XML para construir um DOM baseado na especificação oficial do W3C

Link: <https://anglesharp.github.io/>

Instalação

Install-Package AngleSharp

*Código...*

## 05-08 - Pedidos - Web - pt2

*Código...*

## 05-09 - Pedidos - API - pt1

*Código...*

## 05-10 - Pedidos - API - pt2

*Código...*

### Palavras Finais / Diga

Para / Usar

* **MVC** – Teste de automatização
* **WebApi** – Teste de integração

# 06 - BDD - Behavior Driven Development

## 06-01 - Sobre o que é?

Desenvolvimento guiado por Comportamentos

* Envolver as pessoas no processo através do “Desenvolvimento de fora para dentro” (Outside-in developement).
* Escrever estórias para apresentar as necessidades de um usuário e explicar as funcionalidades da aplicação
* Desenvolver a aplicação guiada por cenários de testes dos comportamentos descritos nas estórias do usuário
* Escrever testes de integração ou automação para validar os comportamentos como uma bateria de testes de regressão.

### Ciclo de Desenvolvimento

1. Mapear a necessidade
2. Escrever um cenário que falhe
3. Faze de codificação
   1. Teste de unidade que reprove
   2. Teste de unidade que passe
   3. Refatorar
4. Cenário Passando
5. Refatoração

## 06-02 - User Stories

## User Stories

* Como um <Papel do negócio>
* Eu quero <Objetivo do negócio>
* Para <Beneficio / Resultado>

Exemplo

* Como um: **Cliente**
* Eu quero> **Colocar um produto no carrinho**
* Para: **Eu compra-lo posteriormente**

## Cenários

* Cenário 1: **Cenários que a operação dá certo**
  + Dado que **o cenário possui um contexto específico**
  + E **um outro detalhe adicional**
  + Quando **o usuário realizar tal ação**
  + Então **alguma coisa que dará certo acontecerá**

## 06-03 - Ferramentas e configuração

### SpecFlow

* Versão do Cocumber para .ET
* Utilizar o Gherkin Parser para transformar o texto em ações
* Compatível com todas as versões do VC atuais

Link: https://specflow.org/

Instalação do framework

Install-Package SpecFlow

Instalação para reconhecimento do Visual Studio com XUnit

Install-Package SpecFlow.xUnit

Passo a passo

1. No Visual Studio
2. Barra de menu > Extensions > Manage Extensions
3. Ir em Online
4. Pequisar por SpecFlow e instalar

## 06-04 - Escrevendo user stories com SpecFlow

*Código...*

## 06-05 - Gerando os testes de comportamento

*Código...*

# 07 - Testes Automatizados

## 07-01 - Apresentação

* Utilização do Selenium
* Page Object Model
* Testes de Aceitação / regreção com SpecFlow
* Boas Práticas no design dos testes

## 07-02 - Apresentando o Selenium

* Realiza a automação de ações para browsers
* Utilizado para automação de testes e Web Crawling
* Compatível com Visual Studio

Link: <https://www.selenium.dev/>

Instalação do framework

Install-Package Selenium.WebDriver

Driver do Google Chrome para utilização do Selenium

<https://chromedriver.chromium.org/downloads>

Baixar a mesma versão do navegador atual

Colocar o arquivo ‘chromedriver’ em ‘C:\WebDriver’

Install-Package Selenium.WebDriver

Install-Package Selenium.Support

## 07-03 - Configurando o Selenium da forma certa

*Código...*

## 07-04 - Customizando o uso do Selenium

*Código...*

## 07-05 - Automatizando a criação do WebDriver do Selenium

*Código...*

## 07-06 - POM - Page Object Model

* É um padrão de projeto descrito por Martin Fowler.
* Diz como criar um repositório de objetos com elementos contidos numa página Web. Sob este modelo, para cada página, deve haver uma classe correspondente.
* Benefícios;
  + Reaproveitamento de código
  + Código mais limpo
  + Facilidade na manutenção
  + Maior independência dos testes como uma bateria de testes de regressão

**ATENÇÃO**

Para rodar o teste de automação da aplicação, deve-se rodar a aplicação fora do Visual Studio. Exemplo, usando o terminal com o comando dotnet run <projeto>

*Código...*

## 07-07 - Automatizando a interface de cadastro de usuário

*Código...*

## 07-08 - Automatizando a interface de cadastro e login

*Código...*

## 07-09 - Automatizando a interface de carrinho de compras

*Código...*

## 07-10 - Execução headless via comando e considerações finais

*Código...*

- Fazer o Publish do Projeto

Abrir o Terminal no diretório do Publish e executar o comando:

dotnet vstest NerdStore.BDD.Tests.dll

# 08 - Testes de Carga

## 08-01 - Tipos de testes de carga

* Tipos de teste
* Ferramentas do mercado
* APDEX - Índice de satisfação
* Como realizar um teste de carga

Quais são os tipos de testes disponíveis e quais são os objetivos deles?

### Teste de desempenho

* **Sua aplicação possui 5 mil usuários. Será que ela é capaz de suportar 1/3 dos usuários simultaneamente?** 
  + O teste de desempenho “performance” é o tipo de teste que valida um canário real. Você precisa garantir que sua aplicação responda bem a um número de conexões esperadas.
  + Neste tipo de teste a preocupação é realizar uma “regressão” para garantir que a aplicação não perdeu desempenho após uma mudança e que ela vai conseguir atender a demanda esperada com um tempo de responsa satisfatório.
  + Ex. Menos de 500ms para o “response time”.

### Teste de carga

* **Quantas transações serão suportadas se o número de usuários aumentar em 50%?**
  + O objetivo principal é encontrar o limite de capacidade da aplicação e identificar qual o gargalho (banco de dados, hardware, cache). Não é o objeto do teste de carga encontrar problemas funcionais na aplicação.
  + É o tipo de teste que precisa ser feito se a aplicação possui picos sazonais onde o aumento de usuários é expressivo (ex. promoções, Natal, black-Friday).
  + É utilizado também para validar se uma a implementação de performance produziu resultados ao comprar um resultado anterior dos testes com o atual após a implementação.
  + Pode ser utilizado para validar se a escada automática (horizontal) está funcionando corretamente no tempo esperado.

### Teste de Stress

* **Como a aplicação irá se comportar se o número não previsto de requisições for disparado simultaneamente?**
  + O teste é baseado no disparo de massivas quantidades de requests simultâneos durante alguns minutos. O objetivo é entender o comportamento da aplicação neste tipo de cenário
  + O que se espera é entender que tipo de “crash” a aplicação pode apresentar, qual o tempo de recuperação da aplicação após uma onda massiva de requests e se aplicação retorna ao restado normal se houver algum tipo de intervenção manual.
  + É utilizado também para validação de cenários de “disaster recovery”. No caso de cluster é desligado um nó ou mais para analisar o comportamento esperado e a recuperação.
  + Muitas vezes o teste é disparado contra uma funcionalidade específica para entender como o resto da aplicação é afetada.

## 08-02 - Ferramentas disponíveis no mercado

* Azure Devops Load Testing – Interrompido
* Apache JMeter
* BrazerMeter
* Gatling
* AutoCannon
* WebSurge
* Artillery.io
* Locus

Links

* <https://docs.microsoft.com/pt-br/aspnet/core/test/load-tests?view=aspnetcore-5.0>
* <https://jmeter.apache.org/>
* <https://www.blazemeter.com/>

## 08-03 - Índice de satisfação do usuário com APDEX

### Medindo o incide de satisfação do cliente om APDEX

* Suponha que o ideal do “response time” da sua aplicação seja algo em torno de 500 a 800 milissegundos. É um tempo de resposta que garante a satisfação do seu cliente com a performance da aplicação.
* Dada algumas circunstâncias (processamento server-side, imagens, css, js, latência, etc). este número será fixado de forma mais pessimista em 800ms.
* O APDEX é um “score” que vai apontar a classificação de satisfação do cliente com base nos resultados de milhares de requests, considerando os requests com melhor tempo de resposta e os demais dentro do limite tolerável.

### Classificação do “Response time”

## 08-04 - Introdução ao JMeter

Download:

<https://jmeter.apache.org/download_jmeter.cgi>

## 08-05 - Plano de testes completo com JMeter e BlazeMeter

*Demonstração...*

## 08-06 - Teste os seus conhecimentos

*Demonstração...*

# 09 - Encerramento

## 09-01 - Palavras Finais

* Teste de unidade
* Teste de integração
* Teste de automação

...

* Teste de unidade:
  + Domínio
  + Application

Links úteis

MSTest

https://github.com/microsoft/testfx

NUnit (Portabilidade do JUnit do Java)

https://nunit.org/

XUnit

https://xunit.net/

Moq

https://github.com/Moq/moq4

Fixtures

https://xunit.net/docs/shared-context

Fluent Assertions

https://fluentassertions.com/

AutoMock

https://github.com/moq/Moq.AutoMocker

SpecFlow

https://specflow.org/

Selenium

https://www.selenium.dev/

https://chromedriver.chromium.org/downloads

Teste de Carga

https://docs.microsoft.com/pt-br/aspnet/core/test/load-tests?view=aspnetcore-5.0

https://jmeter.apache.org/

https://www.blazemeter.com/

Artigos

https://jozimarback.medium.com/testes-de-unidade-e-integra%C3%A7%C3%A3o-com-net-core-e-xunit-fad7c18a29a1

https://medium.com/@RobertKhou/asp-net-core-mvc-identity-using-postgresql-database-bc52255f67c4

https://docs.microsoft.com/pt-br/aspnet/core/test/integration-tests

https://docs.microsoft.com/pt-br/aspnet/core/test/integration-tests

https://anglesharp.github.io/

https://specflow.org/

https://www.seleniumhq.org/

https://chromedriver.chromium.org/downloads

https://docs.microsoft.com/en-us/azure/devops/pipelines/test/continuous-test-selenium?view=azure-devops