Diretrizes para o uso eficiente da Biblioteca Bogus em C# para geração de dados falsos

Nagib Sabbag Filho
FIAP (Faculty of Informatics and Administration Paulista) Avenida Paulista, 1106 - 7º andar - Bela Vista, São Paulo, Brazil.
e-mail: profnagib.filho@fiap.com.br
PermaLink: https://leaders.tec.br/article/diretrizes-para-o-uso-eficiente-da-biblioteca-bogus-em-c-para-geracao-de-dados-falsos
ago 19 2024

Abstract:
A biblioteca Bogus é amplamente utilizada para a geração de dados falsos em projetos de desenvolvimento de software, especialmente em ambientes de teste. Este artigo apresenta um conjunto de diretrizes e melhores práticas para a utilização eficiente da Bogus em C#. O objetivo é auxiliar desenvolvedores na criação de dados realistas e variados, garantindo que os testes de software sejam robustos e representativos. O artigo também explora algumas funcionalidades avançadas da biblioteca.

Key words:
bogus, c#, biblioteca bogus, geração de dados falsos, melhores práticas, dados de teste, mocking, criação de objetos fictícios, faker library, dados aleatórios.

Introdução à Biblioteca Bogus

A biblioteca Bogus é uma ferramenta poderosa para gerar dados falsos de forma fácil e rápida em aplicações C#. É amplamente utilizada em testes, desenvolvimento e protótipos, permitindo que desenvolvedores simulem dados realistas sem a necessidade de um banco de dados real. Com uma interface simples e altamente personalizável, o Bogus é uma escolha popular entre desenvolvedores que necessitam de dados de teste confiáveis.

Instalação da Biblioteca

Para começar a usar o Bogus, você precisa instalá-lo via NuGet. A maneira mais simples de fazer isso é através do Package Manager Console do Visual Studio:

```
Install-Package Bogus
```

Você também pode instalar usando a CLI do .NET:

```
dotnet add package Bogus
```

Gerando Dados Simples

Após a instalação, você pode começar a gerar dados simples. Aqui está um exemplo básico de como criar uma lista de usuários fictícios:

```
using Bogus;

var faker = new Faker("pt_BR");
var users = faker.Make(10, () => new
{
    Name = faker.Name.FullName(),
    Email = faker.Internet.Email(),
    Address = faker.Address.FullAddress()
```

```
});

foreach (var user in users)
{
    Console.WriteLine($"{user.Name}, {user.Email}, {user.Address}");
}
```

Esse código cria 10 usuários com nomes, e-mails e endereços falsos, utilizando a localidade brasileira.

Gerando Dados Relacionados

Uma das funcionalidades mais poderosas do Bogus é a capacidade de gerar dados relacionados. Por exemplo, você pode criar uma lista de produtos que pertencem a uma lista de categorias:

```
var categoryFaker = new Faker()
    .RuleFor(c => c.Id, f => f.IndexFaker + 1)
    .RuleFor(c => c.Name, f => f.Commerce.Department());

var productFaker = new Faker()
    .RuleFor(p => p.Id, f => f.IndexFaker + 1)
    .RuleFor(p => p.Name, f => f.Commerce.ProductName())
    .RuleFor(p => p.Price, f => f.Commerce.Price())
    .RuleFor(p => p.CategoryId, f => f.Random.Int(1, 10));

var categories = categoryFaker.Generate(10);
var products = productFaker.Generate(50);

foreach (var product in products)
{
    Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, CategoryId:
{product.CategoryId}");
}
```

Esse exemplo gera 10 categorias e 50 produtos, onde cada produto possui um ID de categoria aleatório.

Personalização Avançada de Dados

O Bogus permite uma personalização extensiva dos dados. Por exemplo, se você deseja que os e-mails gerados sigam um padrão específico, pode fazer o seguinte:

```
var faker = new Faker("pt_BR");
var customEmailFaker = new Faker()
    .RuleFor(u => u.Name, f => f.Name.FullName())
    .RuleFor(u => u.Email, (f, u) => $"{u.Name.ToLower().Replace(" ",
".")}@example.com");

var users = customEmailFaker.Generate(5);

foreach (var user in users)
{
    Console.WriteLine($"{user.Name}, {user.Email}");
}
```

Neste exemplo, os e-mails gerados seguem o padrão de nome do usuário em minúsculas, substituindo espaços por pontos.

Melhores Práticas no Uso do Bogus

Integrar o Bogus em seus testes automatizados é uma das melhores práticas ao utilizar a biblioteca. Isso permite que você tenha dados consistentes e previsíveis para validar o comportamento de sua aplicação. Aqui está um exemplo de como usar o Bogus em um teste unitário:

```
using Xunit;

public class UserServiceTests
{
    private readonly UserService _userService;

    public UserServiceTests()
    {
        _userService = new UserService();
    }

    [Fact]
    public void Should_Create_User_With_Valid_Data()
    {
        var faker = new Faker()
            .RuleFor(u => u.Name, f => f.Name.FullName())
            .RuleFor(u => u.Email, f => f.Internet.Email());

        var user = faker.Generate();

        var result = _userService.CreateUser(user);

        Assert.NotNull(result);
        Assert.Equal(user.Name, result.Name);
        Assert.Equal(user.Email, result.Email);
    }
}
```

Este exemplo mostra como você pode criar um teste que valida a criação de um usuário com dados gerados pelo Bogus.

Funcionalidades Avançadas

O Bogus oferece uma variedade de funcionalidades avançadas que permitem criar dados falsos de maneira ainda mais sofisticada. Abaixo, apresentamos alguns exemplos práticos para demonstrar como você pode tirar proveito dessas funcionalidades em seus projetos.

1. Geração de Dados Baseada em Distribuição

Em alguns casos, pode ser útil gerar dados que sigam uma distribuição específica, como uma distribuição normal para simular eventos naturais. O Bogus permite isso usando a classe Randomizer:

```
using Bogus;
```

```
// Gera idades seguindo uma distribuição normal (média 30, desvio padrão 5)
var ageFaker = new Faker().Random.Int(18, 50).OrNull(f => f.Random.Float() < 0.1f);

var ages = new List();
for (int i = 0; i &lt; 100; i++)
{
    ages.Add(ageFaker);
}

foreach (var age in ages)
{
    Console.WriteLine(age);
}
```

Esse código gera 100 idades com uma chance de 10% de serem nulas, seguindo uma distribuição que simula uma população adulta.

2. Geração Sequencial de Dados

O Bogus permite gerar dados que seguem uma sequência personalizada. Isso é útil, por exemplo, para criar identificadores únicos ou números de série:

```
using Bogus;

// Gera um número de série sequencial no formato "ABC-001", "ABC-002", etc.
var serialFaker = new Faker()
    .RuleFor(s => s.SerialNumber, f => $"ABC-{f.IndexFaker + 1:000}");

var serials = serialFaker.Generate(10).Select(s => s.SerialNumber);

foreach (var serial in serials)
{
    Console.WriteLine(serial);
}
```

Neste exemplo, são gerados 10 números de série no formato "ABC-001", "ABC-002" e assim por diante.

3. Geração de Dados Dependentes

Em cenários onde os dados de um campo dependem dos dados de outro campo, o Bogus pode ser configurado para criar essa relação. Veja o exemplo abaixo:

```
using Bogus;

var dependentFaker = new Faker()
    .RuleFor(o => o.FullName, f => f.Name.FullName())
    .RuleFor(o => o.UserName, (f, o) => o.FullName.Replace(" ", "").ToLower());

var users = dependentFaker.Generate(5);

foreach (var user in users)
{
    Console.WriteLine($"Nome Completo: {user.FullName}, Nome de Usuário:
{user.UserName}");
```

```
}
```

Aqui, o UserName é gerado com base no FullName, garantindo que o nome de usuário esteja relacionado ao nome completo da pessoa.

4. Geração de Dados Aninhados

Em sistemas mais complexos, pode ser necessário gerar objetos aninhados, onde um objeto contém outros objetos. O Bogus facilita essa tarefa:

```
using Bogus;

// Define a classe Address
public class Address
{
    public string Street { get; set; }
    public string City { get; set; }
}

// Define a classe User com um Address aninhado
public class User
{
    public string FullName { get; set; }
    public Address Address { get; set; }
}

// Gera uma lista de usuários com endereços aninhados
var userFaker = new Faker()
    .RuleFor(u => u.FullName, f => f.Name.FullName())
    .RuleFor(u => u.Address, f => new Address
    {
        Street = f.Address.StreetAddress(),
        City = f.Address.City()
    });

var users = userFaker.Generate(5);

foreach (var user in users)
{
    Console.WriteLine($"Nome: {user.FullName}, Endereço: {user.Address.Street},
{user.Address.City}");
}
```

Este exemplo cria uma lista de usuários, cada um com um endereço aninhado, demonstrando como o Bogus pode ser usado para simular estruturas de dados complexas.

5. Geração de Dados Condicionais

O Bogus também permite gerar dados baseados em condições específicas, o que é útil para simular diferentes cenários de teste:

```
using Bogus;

// Gera um status de pedido baseado no valor total do pedido
```

```
var orderFaker = new Faker()
    .RuleFor(o => o.TotalAmount, f => f.Finance.Amount(100, 1000))
    .RuleFor(o => o.Status, (f, o) => o.TotalAmount > 500 ? "Aprovado" : "Pendente");

var orders = orderFaker.Generate(5);

foreach (var order in orders)
{
    Console.WriteLine($"Total: {order.TotalAmount}, Status: {order.Status}");
}
```

Neste exemplo, o status do pedido é definido como "Aprovado" se o valor total for maior que 500, caso contrário, é "Pendente".

Conclusão

A biblioteca Bogus é uma ferramenta essencial para a geração de dados falsos em C#, oferecendo flexibilidade e personalização para atender a diversas necessidades de desenvolvimento e testes. Sua integração com testes automatizados e a capacidade de criar dados realistas e relacionados a tornam uma escolha valiosa para desenvolvedores que buscam simular cenários reais sem a necessidade de um banco de dados real.

Referências

Para obter mais informações sobre a biblioteca Bogus, consulte a documentação oficial e outros recursos disponíveis:

BCHAVEZ. Repositório Oficial do Bogus no GitHub. Disponível em: https://github.com/bchavez/Bogus. Acesso em: 29 jul. 2024.
BCHAVEZ. Documentação do Bogus. Disponível em: https://github.com/bchavez/Bogus#readme. Acesso em: 29 jul. 2024.
NUGET. Pacote NuGet do Bogus. Disponível em: https://www.nuget.org/packages/Bogus/. Acesso em: 29 jul. 2024.

Nagib Filho é Professor Universitário e Tech Manager.
Possui uma trajetória de conquistas em certificações técnicas e ágeis,
incluindo MCSD, MCSA e PSM1.
PG em Gestão de TI pelo SENAC e MBA em Tecnologia de Software pela USP,
Nagib cursou programas de extensão do MIT e Universidade de Chicago.
Outras conquistas incluem a autoria de um artigo sobre chatbots, revisado por pares e apresentado na Universidade de Barcelona.