Introducción a BDD

Una herramienta para mejorar la comunicación con los usuarios

© Miguel Veloso coderepo.blog @mvelosop

Objetivos de hoy

1. Entender el concepto y aplicaciones del BDD
2. Conocer detalles de implementación con SpecFlow y EF Core 2
3. Compartir experiencias

BDD – Conceptos

Behaviour Driven Development

© Miguel Veloso coderepo.blog @mvelosop

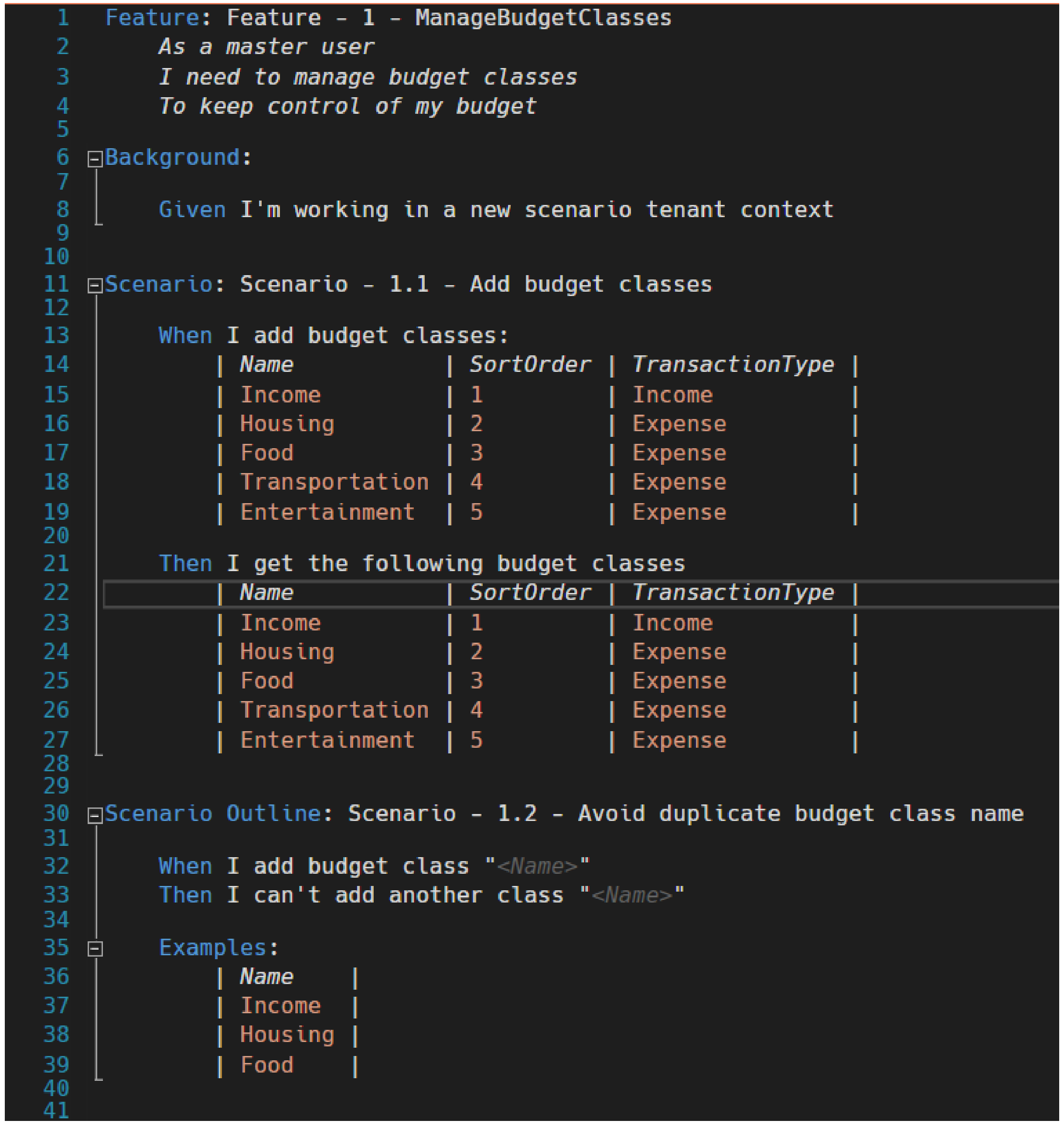
BDD es un proceso de desarrollo

Especificaciones entendibles por los usuarios

No es una metodología de pruebas

Gherkin
Lenguaje de las especificaciones

¿Cómo es una especificación en Gherkin?



```
Feature: Feature - 1 - ManageBudgetClasses
    As a master user
    I need to manage budget classes
    To keep control of my budget

Background:

    Given I'm working in a new scenario tenant context


Scenario: Scenario - 1.1 - Add budget classes

    When I add budget classes:
        | Name           | SortOrder | TransactionType |
        | Income         | 1         | Income          |
        | Housing        | 2         | Expense         |
        | Food           | 3         | Expense         |
        | Transportation | 4         | Expense         |
        | Entertainment  | 5         | Expense         |

    Then I get the following budget classes
        | Name           | SortOrder | TransactionType |
        | Income         | 1         | Income          |
        | Housing        | 2         | Expense         |
        | Food           | 3         | Expense         |
        | Transportation | 4         | Expense         |
        | Entertainment  | 5         | Expense         |


Scenario Outline: Scenario - 1.2 - Avoid duplicate budget class name

    When I add budget class "<Name>"
    Then I can't add another class "<Name>"

    Examples:
        | Name    |
        | Income  |
        | Housing |
        | Food    |


```

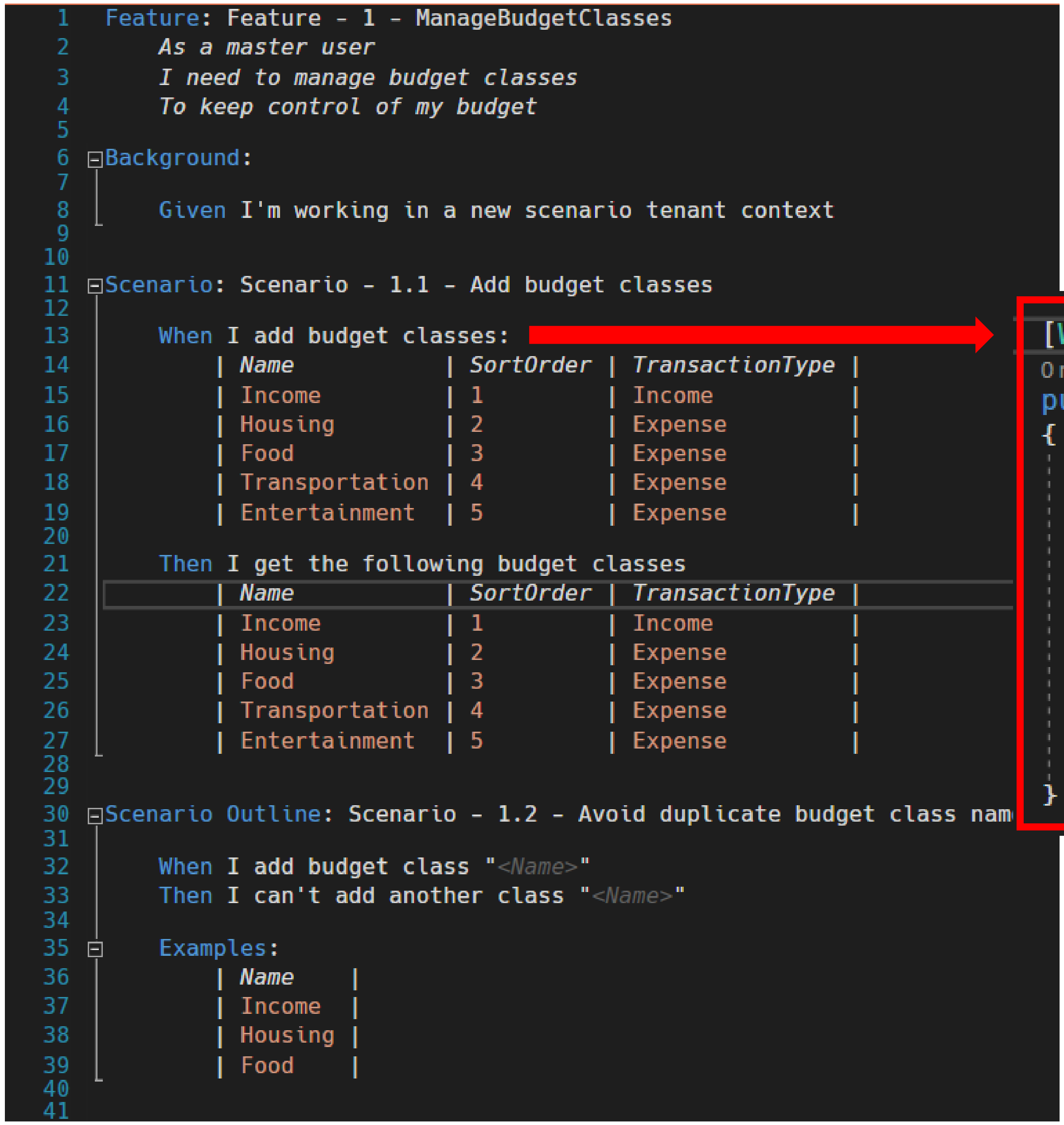
© Miguel Veloso coderepo.blog @mvelosop

ARRANGE → GIVEN

ACT → WHEN

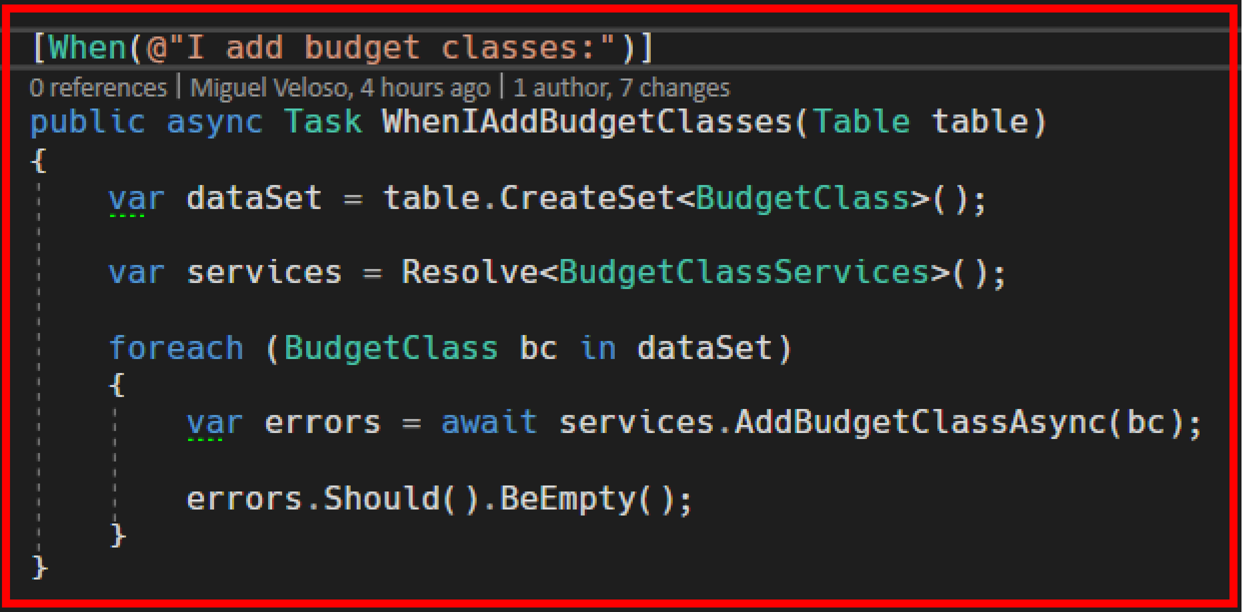
ASSERT → THEN

SpecFlow
Implementación BDD en .NET



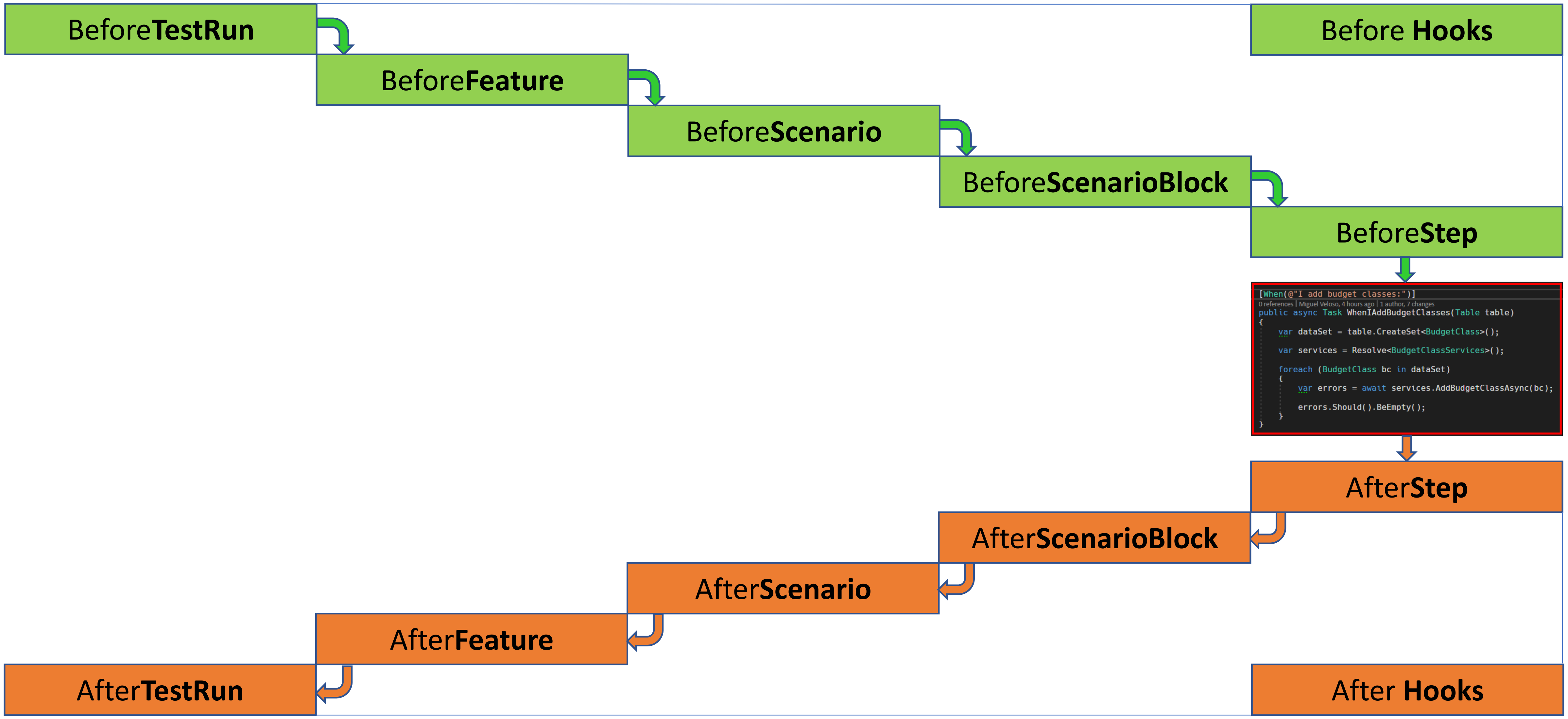
```gherkin
Feature: Feature - 1 - ManageBudgetClasses
	As a master user
	I need to manage budget classes
	To keep control of my budget

Background:

	Given I'm working in a new scenario tenant context


Scenario: Scenario - 1.1 - Add budget classes

	When I add budget classes:
		| Name           | SortOrder | TransactionType |
		| Income         | 1         | Income          |
		| Housing        | 2         | Expense         |
		| Food           | 3         | Expense         |
		| Transportation | 4         | Expense         |
		| Entertainment  | 5         | Expense         |

	Then I get the following budget classes
		| Name           | SortOrder | TransactionType |
		| Income         | 1         | Income          |
		| Housing        | 2         | Expense         |
		| Food           | 3         | Expense         |
		| Transportation | 4         | Expense         |
		| Entertainment  | 5         | Expense         |


Scenario Outline: Scenario - 1.2 - Avoid duplicate budget class nam

	When I add budget class "<Name>"
	Then I can't add another class "<Name>"

	Examples:
		| Name    |
		| Income  |
		| Housing |
		| Food    |

```

BINDING – Step Definition



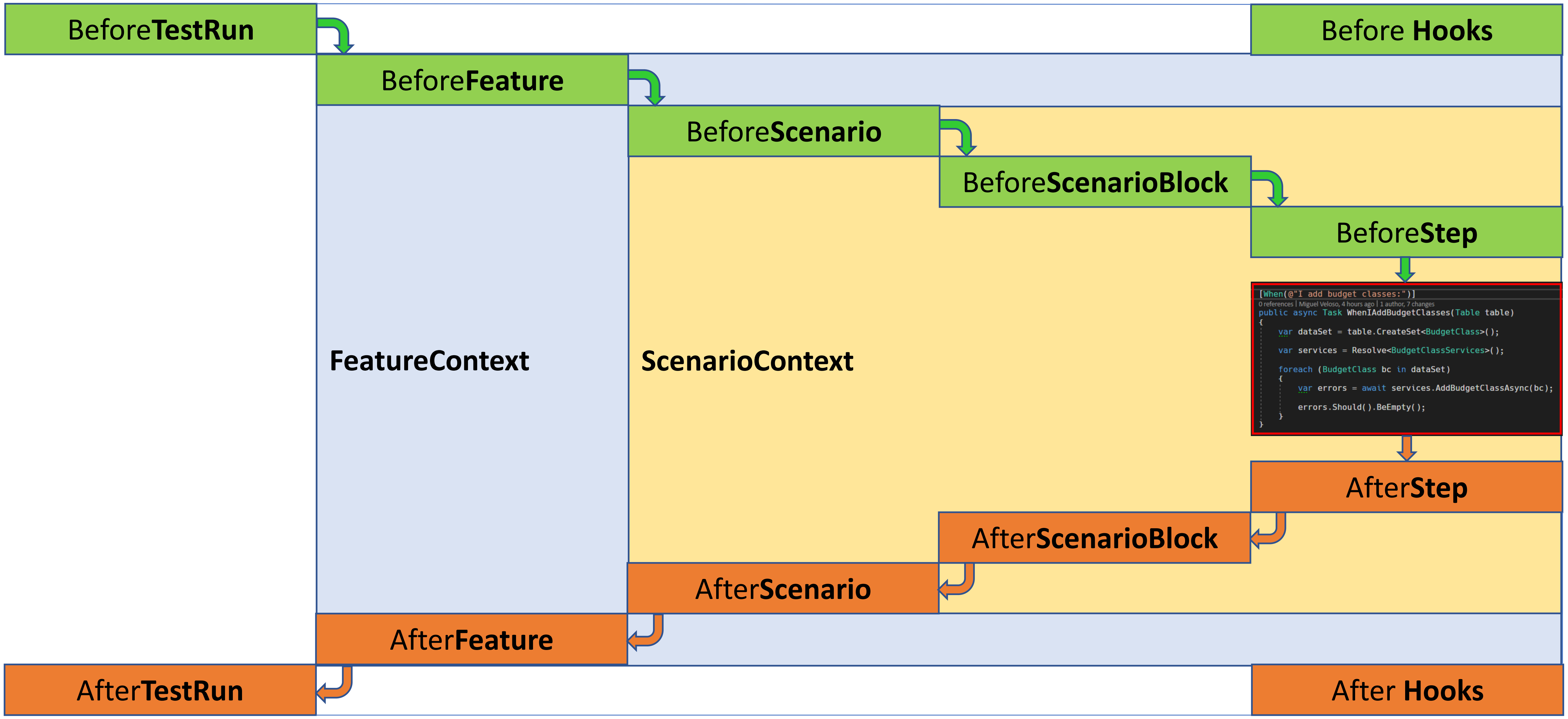
```csharp
[When(@"I add budget classes:")]
0 references | Miguel Veloso, 4 hours ago | 1 author, 7 changes
public async Task WhenIAddBudgetClasses(Table table)
{
    var dataSet = table.CreateSet<BudgetClass>();

    var services = Resolve<BudgetClassServices>();

    foreach (BudgetClass bc in dataSet)
    {
        var errors = await services.AddBudgetClassAsync(bc);

        errors.Should().BeEmpty();
    }
}
```

HOOKS + CONTEXTS



```
[When(@"I add budget classes:")]
0 references | Miguel Veloso, 4 hours ago | 1 author, 7 changes
public async Task WhenIAddBudgetClasses(Table table)
{
    var dataSet = table.CreateSet<BudgetClass>();

    var services = Resolve<BudgetClassServices>();

    foreach (BudgetClass bc in dataSet)
    {
        var errors = await services.AddBudgetClassAsync(bc);

        errors.Should().BeEmpty();
    }
}
```

HOOKS + CONTEXTS



```
[When(@"I add budget classes:")]
0 references | Miguel Veloso, 4 hours ago | 1 author, 7 changes
public async Task WhenIAddBudgetClasses(Table table)
{
    var dataSet = table.CreateSet<BudgetClass>();

    var services = Resolve<BudgetClassServices>();

    foreach (BudgetClass bc in dataSet)
    {
        var errors = await services.AddBudgetClassAsync(bc);

        errors.Should().BeEmpty();
    }
}
```

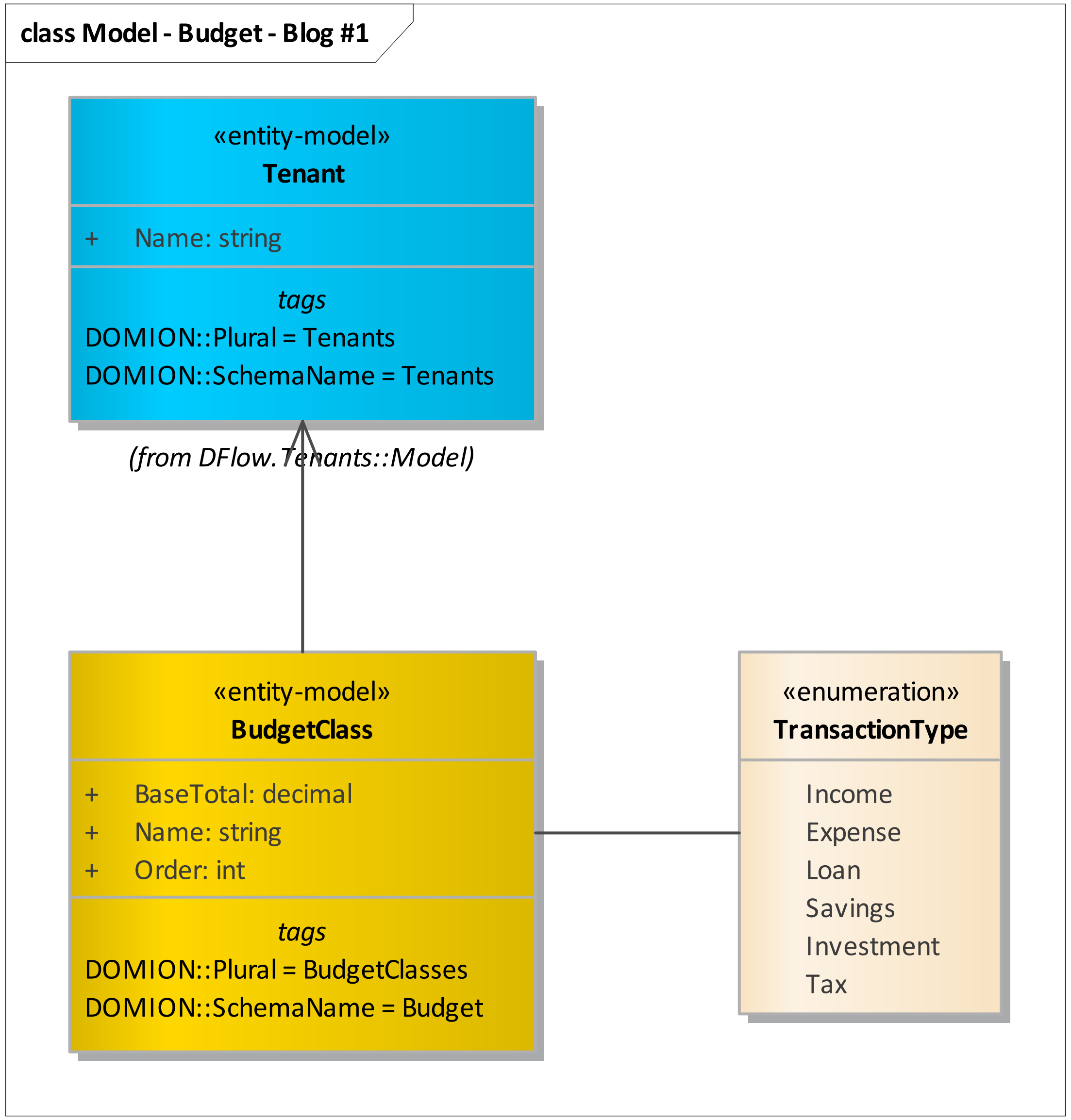
SpecFlow: Escenario → Prueba

xUnit: Ejecuta pruebas

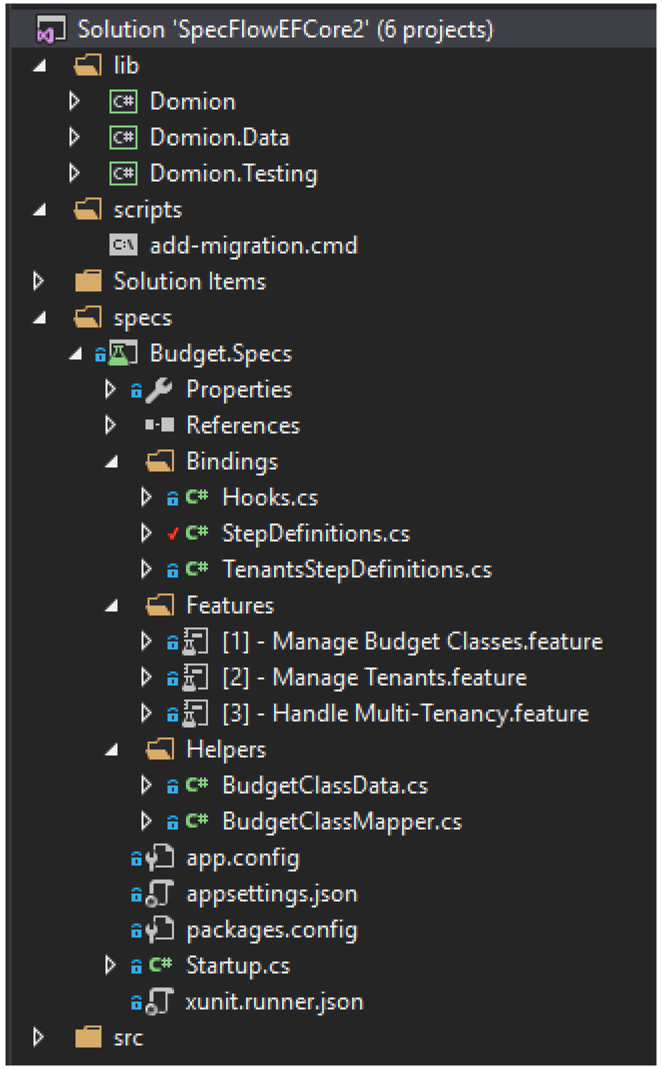
Frameworks de pruebas: NUnit, MSTest, mbUnit

BDD – Demo

https://github.com/mvelosop/SpecFlowEFCore2

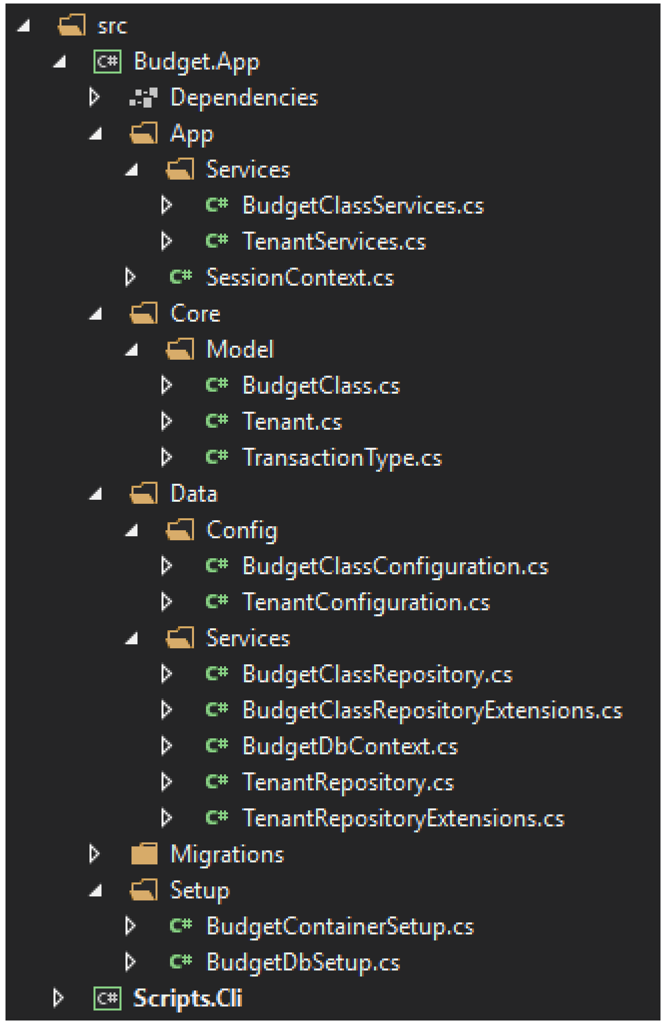

class Model - Budget - Blog #1
«entity-model»
Tenant
+ Name: string
tags
DOMION::Plural = Tenants
DOMION::SchemaName = Tenants
(from DFlow.Tenants::Model)
«entity-model»
BudgetClass
+ BaseTotal: decimal
+ Name: string
+ Order: int
tags
DOMION::Plural = BudgetClasses
DOMION::SchemaName = Budget
«enumeration»
TransactionType
Income
Expense
Loan
Savings
Investment
Tax

Solución



```
Solution 'SpecFlowEFCore2' (6 projects)
◢ lib
    ▷ Domion
    ▷ Domion.Data
    ▷ Domion.Testing
◢ scripts
    add-migration.cmd
▷ Solution Items
◢ specs
  ◢ Budget.Specs
      ▷ Properties
      ▷ References
      ◢ Bindings
          ▷ Hooks.cs
          ▷ StepDefinitions.cs
          ▷ TenantsStepDefinitions.cs
      ◢ Features
          ▷ [1] - Manage Budget Classes.feature
          ▷ [2] - Manage Tenants.feature
          ▷ [3] - Handle Multi-Tenancy.feature
      ◢ Helpers
          ▷ BudgetClassData.cs
          ▷ BudgetClassMapper.cs
        app.config
        appsettings.json
        packages.config
      ▷ Startup.cs
        xunit.runner.json
▷ src
```

Aplicación



```
◢ src
  ◢ Budget.App
    ▷ Dependencies
    ◢ App
      ◢ Services
        ▷ BudgetClassServices.cs
        ▷ TenantServices.cs
      ▷ SessionContext.cs
    ◢ Core
      ◢ Model
        ▷ BudgetClass.cs
        ▷ Tenant.cs
        ▷ TransactionType.cs
    ◢ Data
      ◢ Config
        ▷ BudgetClassConfiguration.cs
        ▷ TenantConfiguration.cs
      ◢ Services
        ▷ BudgetClassRepository.cs
        ▷ BudgetClassRepositoryExtensions.cs
        ▷ BudgetDbContext.cs
        ▷ TenantRepository.cs
        ▷ TenantRepositoryExtensions.cs
    ▷ Migrations
    ◢ Setup
      ▷ BudgetContainerSetup.cs
      ▷ BudgetDbSetup.cs
  ▷ Scripts.Cli
```

© Miguel Veloso coderepo.blog @mvelosop

Demo – 1. Crear Migración Inicial

1-1. Add DbContext factory	**BudgetDbContextDesignTimeFactory.cs**
1-2. Initialize DbContext and apply migration to verify it's working	**Program.cs**

Demo – 2. Configurar contenedor de Autofac

2-1. Include BudgetDbSetup helper	**Startup.cs**
2-2. Configure Autofac module	**Startup.cs**
2-3. Create database / apply migrations	**Startup.cs**
2-4. Configure database service	**Startup.cs**

© Miguel Veloso coderepo.blog @mvelosop

Demo – 3. Agregar y consultar BudgetClass

3-1. Add BudgetClassServices	**BudgetClassServices.cs**
3-2. Get budget classes step	**StepDefinitions.cs**
3-3. Add budget classes step	**StepDefinitions.cs**
3-4. Save to repo	**BudgetClassServices.cs**
3-5. Get data from repo	**BudgetClassServices.cs**

Demo – 4. Asegurar estado inicial limpio

4-1. Clear data step	StepDefinitions.cs
4-2. Store scope in scenario context	Hooks.cs
4-3. Dispose scope from scenario context	Hooks.cs
4-4. Resolve dependency from current scope	StepDefinitions.cs
4-5. Refactor dependency resolution	StepDefinitions.cs

Demo – 5. Evitar duplicados de BudgetClass

5-1. Add budget class step	**StepDefinitions.cs**
5-2. Verify duplicate name step	**StepDefinitions.cs**
5-3. Validate name duplication	**BudgetClassRepository.cs**

© Miguel Veloso coderepo.blog @mvelosop

Demo – 6. Agregar manejo de Tenants

6-1. Add tenant model	**Tenant.cs**
6-2. Add TenantConfiguration	**TenantConfiguration.cs**
6-3. Add tenant configuration	**BudgetDbContext.cs**
6-4. Add TenantRepository	**TenantRepository.cs**
6-5. Add TenantRepositoryExtensions	**TenantRepositoryExtensions.cs**
6-6. Add TenantServices	**TenantServices.cs**
6-7. Add TenantsStepDefinitions	**TenantsStepDefinitions.cs**

Demo – 7. Manejar multi-tenancy

7-1. Add SessionContext	SessionContext.cs
7-2. Add new tenant step	TenantsStepDefinitions.cs
7-3. Add budget classes to tenant step	StepDefinitions.cs

Demo – 8. Agregar FK BudgetClass → Tenant

8-1. Add tenant reference	**BudgetClass.cs**
8-2. Configure tenant navigation and FK properties	**BudgetClassConfiguration.cs**
8-3. Inject SessionContext	**BudgetClassRepository.cs**
8-4. Include SessionContext in query	**BudgetClassRepository.cs**
8-5. Include SessionContext on saving	**BudgetClassRepository.cs**

Demo – 9. Tenant por escenario

9-1. Register session context in scope	**Hooks.cs**
9-2. Create Scenario tenant context	**StepDefinitions.cs**
9-3. Create tenant context for session	**StepDefinitions.cs**
9-4. Fix to properly clean previous state	**TenantsStepDefinitions.cs**

Demo – 10. Preparar escenario solo una vez

10-1. Inject FeatureContext	**StepDefinitions.cs**
10-2. Reset tenant data just once per scenario	**StepDefinitions.cs**

Demo – 11. Actualizar BudgetClass

11-1. Add BudgetClassData helper class	**BudgetClassData.cs**
11-2. Add BudgetClassData helper class	**BudgetClassMapper.cs**
11-3. Implement BudgetClass finder	**BudgetClassServices.cs**
11-4. Implement BudgetClass updater	**BudgetClassServices.cs**
11-5. Map "Given" Clause	**StepDefinitions.cs**
11-6. Add update step	**StepDefinitions.cs**
11-7. Register Mapper types	**Startup.cs**

Demo – 12. Eliminar BudgetClass

12-1. Implement BudgetClass remove	**BudgetClassServices.cs**
12-2. Add delete step	**StepDefinitions.cs**

Comentarios – Q&A

Blog: coderepo.blog
Twitter: @mvelosop

Repo: https://github.com/mvelosop/SpecFlowEFCore2