Testowanie aplikacji Prism/MVVM
Piotr Cholewa

Testy jednostkowe - instrukcja krok po kroku

Będziemy testować aplikację prismMVVM, która jest prostym projektem wykorzystującym wzorzec MVVM i framework Prism.

Utworzenie projektu testowego

Aby napisać testy jednostkowe trzeba będzie utworzyć projekt. Klikamy prawym przyciskiem na solucji, kierujemy kursor myszy na napis "Add" i pojawieniu się listy klikamy na "New Project".

Pojawi nam się okienko do wyboru projektu. Z lewej strony, z grupy "installed" wybieramy język "Visual C#", klikamy na kategorię projektów "test". W środkowym widoku pojawią się nam dostępne projekty do pisania testów, nas interesuje ten o nazwie "Unit Test Project". Skoro testujemy aplikację o nazwie "prismMVVM", to nazwijmy nasz testowy projekt "prismMVVMUnitTest". Po kliknięciu "OK" do solucji zostanie dodany projekt wraz z plikiem, gdzie będziemy tworzyli kod testujący (pliki takie możemy dodawać klikając prawym przyciskiem na projekcie do testów jednostkowych, gdzie z listy rozwijalnej "Add" wybieramy "Unit Test...").

Pisanie testów

Najpierw przetestujemy model, który znajduje się w folderze Models w pliku Person.cs. Wygląda on następująco:

```
namespace prismMVVM.Models
{
    4 references
    public class Person
    {
        1 reference
        public string Name { get; set; }
        4 references | 1/1 passing
        public int Year { get; set; }
        0 references
        public int Age { get; set; }
        1 reference
        public string FavouriteColor { get; set; }

        2 references | 1/1 passing
        public int CalculateAge()
        {
            return System.DateTime.Now.Year - Year;
        }
    }
}
```

Jedyną rzeczą, która będzie testowana to metoda CalculateAge. Zasadniczo testy składają się z 3 faz Arrange, Act i Assert - przygotowania testu, wykonania i porównanie rezultatu ze

spodziewanym wynikiem. Klasa Assert zawiera metody porównujące, w których zwykle w 1. argumencie podajemy wartość oczekiwaną, a w 2. wartość otrzymaną. Oto kod testujący model:

```
namespace prismMVVMUnitTest
{
    [TestClass]
    0 references
    public class UnitTestModel
    {
        [TestMethod]
        0 references
        public void TestMethod1()
        {
            //arrange
            Person person = new Person();
            person.Year = 1993;
            int actualYear = DateTime.Now.Year;

            //act
            int age =person.CalculateAge();

            //assert
            Assert.AreEqual(actualYear - 1993, age);
        }
    }
}
```

Następnie zostanie przetestowany model widoku, lecz nie będzie on cały testowany, lecz jego część, aby pokazać inne możliwości testowania. Po pierwsze zostanie sprawdzona poprawność działanie konstruktora i właściwości Year.

```
3 references | 3/3 passing
public WidokViewModel()
{
    CalculateCommand = new DelegateCommand(CalcAge);
    ResetCommand = new DelegateCommand(Reset);
    AddCommand = new DelegateCommand(AddPerson);

    this.AllColors = new[] { "czerwony", "żółty", "zielony" };
}

4 references | 2/2 passing
public int Year
{
    get
    {
        return _year;
    }
    set
    {
        SetProperty<int>(ref _year, value);
    }
}
```

Konstruktor jest testowany pod względem inicjalizacji instancji klasy modelu widoku i komend, które są tam przypisywane. Warto zwrócić uwagę na drugi test, który testuje charakterystyczny dla MVVM kod, czyli wywołanie zdarzenia PropertyChanged przy zmianie właściwości. W tym celu trzeba przypisać do zdarzenia metodę, która zapisuje do zmiennej typu string nazwę zmienianej właściwości - tej która wywołała zdarzenie PropertyChanged. Oto kod testujący konstruktor i właściwość:

```
namespace prismMVVMUnitTest
{
    [TestClass]
    0 references
    public class UnitTestViewModel
    {
        [TestMethod]
        0 references
        public void ConstructorTest()
        {
            WidokViewModel widokViewModel = new WidokViewModel();
            Assert.IsNotNull(widokViewModel);
            Assert.IsNotNull(widokViewModel.AddCommand);
            Assert.IsNotNull(widokViewModel.CalculateCommand);
            Assert.IsNotNull(widokViewModel.ResetCommand);
        }

        [TestMethod]
        0 references
        public void YearTest()
        {
            int expected = 1993;
            string actual = null;

            WidokViewModel widokViewModel = new WidokViewModel();
            widokViewModel.PropertyChanged += delegate(object sender, PropertyChangedEventArgs e)
            {
                actual = e.PropertyName;
            };
            widokViewModel.Year = expected;
            Assert.IsNotNull(actual);
            Assert.AreEqual(expected, widokViewModel.Year);
        }
```