# Poznamky z vesmiru: (Unit testy jsou schovane ve WPFUniverseWithRequestResponse v assembly WPFUniverse.Core.Tests)

* Do Solution zalozime novy projekt , bude se jmenovat UnitTests a bude mit reference na assembly, podle toho co testujeme .
* Nastavime reference , aby testovaci projekt videl na vsechny ostatni projekty.
* Stahneme balicek **NUnit** coz je testovaci framework a balik **Moq**.
* Pridame tridu PropertiesViewModelFixture.cs

Je to vlastne testovaci trida ktera nam nahradi v aplikaci PropertiesViewModel aniz by to aplikace poznala.

* Aby se vedelo ze jde o Unit test nad tridu prijde metadata klauzule : [TestFixture]
* A nad nazev testovaci metody [Test]
* Jelikoz ma skutecna trida PropertiesViewModel nejake zavislosti musime ji tyto zavislosti vytvorit.

Na to slouzi prave balicek Moq.

* Ten vytvori falesne instance rozhrani ktere jsou v testovaci tride jako zavislosti Mock<IPlanetSelector> planetSelectorMock = new Mock<IPlanetSelector>();

**Co testujeme :** Chceme vedet jestli po kliknuti na checkbox v PropertiesViewModelu , bude pridana do **SelectedPlanet.Properties** na spravne misto hodnota true do vlastnosti **IsChecked**.

* Vytvorime si vlastni testovaci data . V tomto pripade potrebujeme kolekci VlastnostDataContractu a rovnou do ni nekolik datacontractu vlozime.
* Testy se provadi pomoci metod.
* Nezapomenout zaregistrovat pripadne eventy ktere ve tride spousti nejakou cinnost:

viewModel.RegisterToEvent();

* Rekneme propertiesManageru , ze chceme vymenit ListOfPossibleVlastnost ze skutecne tridy , za nasi observableCollection.

propertiesManagerMock.Setup(d => d.ListOfAllPossibleVlastnosts).Returns(() => observableCollection);

* Vytvorime instanci PropertiesViewModelu, pricemz ji dame zavislosti na nase falesne MockObjekty.

PropertiesViewModel viewModel = new PropertiesViewModel(planetSelectorMock.Object, propertiesManagerMock.Object, planetPropertiesDaoMock.Object);

* Vytvorime si PlanetDataContract selectedPlanet na kterem budeme testovat. Po zmene hodnoty IsChecked v nekterem VlastnostDataContractu v nasi falesne kolekci

observableCollection[0].IsChecked = true;

by se mel datacontract objevit v kolekci (seznamu vlastnosti) v selectedPlanet.Propeties.

* Otestujeme pomoci Assert:

Assert.IsNotEmpty(selectedPlanet.Properties);

Assert.IsTrue(selectedPlanet.Properties.Count == 1);

Assert.AreEqual(selectedPlanet.Properties[0].Id, observableCollection[0].Id);

Assert.AreEqual(selectedPlanet.Properties[0].Nazev, observableCollection[0].Nazev);

Příkaz assert, umožňuje do přeloženého programu zařadit kontrolu, zda je splněna podmínka uvedená za klíčovým slovem assert. Pokud tato podmínka splněna není, vyvolá se výjimka AssertionError.

* Test spustime pomoci ReSharpu kliknutim vlevo na zelenou tecku jako run nebo debug s breakem
* Testujeme pomoci metod na tridach ktere jsou public. Bud jsou v nejakem interfacu , nebo se daji zavolat pomoci commandu ktery je public.
* Aby se spustila metoda v testu musi se taky namockovat

transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

jinak probehne prazdna medota ( DoAllChanges mi nejezdilo)

**Kod udrzovat rozdeleny na atomicke casti , prehledny cisty a pokud mozno blby.**

**Testy by meli byt jednoduche a kratke jinak je pravdepodobne , ze je spatne napsana aplikace.**

**Mappery a Unit Testy**

* Mappery zjednodusi kod v testech .
* Udelame slozku v Core
* Vytvorili jsme public class GalaxyDataContractMapper : IMapper<Galaxie, GalaxyDataContract>
* Rodicovska trida prebira dva parametry , jeden vstup a jeden vystup
* Maper je jednoducha zalezitost ktera prevadi pomoci metody **map**

public GalaxyDataContract Map(Galaxie obj)

{

return new GalaxyDataContract(obj.Id, obj.Jmeno, obj.PolohaX, obj.PolohaY, obj.PolohaZ);

}

* Vraci vystupni objekt.
* Testy by mohly byt taky rozdeleny do logickych celku takze pro testy Servisnich akci udelame slozku ServiceActions a do ni tridu s testem.
* Tim ze mame mapper , nepotrebujeme v testu delat prevody mezi tridama a tim je tam mene kodu.
* V metode **TestSetUp()** se naMockuji instance a mappery ktere potrebujeme a predame falesne objekty servisni akci.
* V metode  **Execute ()**  je samotne provadeni operaci a jejich kontrola pomoci **Assert.** (execute jsem pozdeji zmenil ma tam byt vystizny nazev co se vlastne testuje. Nove se jmenuje SelectAllGalaxiesServiceActionTest )
* Metoda m\_galaxyDao.**VerifyAll()** overuje jestli na falesnem objektu dao opravdu probehla metoda ktera je v definovana v :

var galaxie = new Galaxie{Id = galaxyId};

m\_galaxyDao.Setup(d => d.SelectAll()).Returns(new List<Galaxie> {galaxie});

* Metoda **.Setup()**  je dostupna po instalaci BLToolkitu. Tady mi VerifyAll() proveri jestli probehla spravne a na spravnych prostredcich. Napr. kdyz vymenime objekt galaxie za new Galaxy() tak sice probiha na objektu typu Galaxie , ale ne na tom , ktery jsme si vytvorili sami a predali mu Id.
* **NOVA SLOZKA S INSTALLEREM**
* Jelikoz jsou mappery v Coru potrebujeme tady dalsi slozku Installers . (Prvni je v Shellu )
* Trida CoreInstaller, ktera je zde zaregistruje Mapper do Windsor containeru.
* Ve tride Bootstrapper se musi metoda na tride CoreInstaller zavolat v metode InitializeContainer.

///////////////////////////////

# jediny zachovany test:

using System.Collections.Generic;

using System.Linq;

using Mediaresearch.Framework.Mapping;

using Moq;

using NUnit.Framework;

using WpfUniverse.Common.Datacontracts;

using WpfUniverse.Common.Requests;

using WpfUniverse.Core.ServiceActions;

using WpfUniverse.Entities;

namespace WpfUniverse.Core.Tests.ServiceActions

{

[TestFixture]

public class SelectAllGalaxiesServiceActionTest

{

private Mock<IGalaxyDao> m\_galaxyDao;

private Mock<IMapper<Galaxie, GalaxyDataContract>> m\_mapper;

private SelectAllGalaxiesServiceAction SUT;

[SetUp]

public void TestSetUp()

{

m\_galaxyDao = new Mock<IGalaxyDao>();

m\_mapper = new Mock<IMapper<Galaxie, GalaxyDataContract>>();

SUT = new SelectAllGalaxiesServiceAction(m\_galaxyDao.Object, m\_mapper.Object);

}

[Test]

public void Execute()

{

const int galaxyId = 5;

var galaxie = new Galaxie{Id = galaxyId};

m\_galaxyDao.Setup(d => d.SelectAll()).Returns(new List<Galaxie> {galaxie});

m\_mapper.Setup(d => d.Map(galaxie)).Returns(new GalaxyDataContract(galaxyId, string.Empty, 0, 0, 0));

var response = SUT.Execute(new SelectAllGalaxiesRequest());

m\_galaxyDao.VerifyAll();

m\_mapper.VerifyAll();

Assert.IsTrue(response.Galaxies.Any(d=>d.Id == galaxyId));

}

}

}

# Poznamky z vesmiru pokracovani:

* **Nejprve se podivame na skutecnou servisni akci. Vidime, ze ma zavislosti na galaxyDao a mapper.**

public SelectAllGalaxiesServiceAction(IGalaxyDao galaxyDao, IMapper<Galaxie, GalaxyDataContract> mapper)

{

m\_galaxyDao = galaxyDao;

m\_mapper = mapper;

}

* **Tyto dve property si teda namockujeme:**

private Mock<IGalaxyDao> m\_galaxyDao;

private Mock<IMapper<Galaxie, GalaxyDataContract>> m\_mapper;

[SetUp]

public void TestSetUp()

{

m\_galaxyDao = new Mock<IGalaxyDao>();

m\_mapper = new Mock<IMapper<Galaxie, GalaxyDataContract>>();

**vytvorime si instanci servisni akce a jako parametry ji dame falesne objekty**

serviceAction = new SelectAllGalaxiesServiceAction(m\_galaxyDao.Object, m\_mapper.Object);

}

* **To byla priprava a ted samotny test:**

[Test]

public void SelectAllGalaxiesReturnsSomeGalaxies()

{

const int galaxyId = 5;

var galaxie = new Galaxie{Id = galaxyId};

// specifikujeme co se ma vratit

m\_galaxyDao.Setup(d => d.SelectAll()).Returns(new List<Galaxie> {galaxie});

// specifikujeme co se ma vratit

m\_mapper.Setup(d => d.Map(galaxie)).Returns(new GalaxyDataContract(galaxyId, string.Empty, 0, 0, 0));

var response = serviceAction.Execute(new SelectAllGalaxiesRequest());

// overime zda na falesnem objektu opravdu probehla definovana metoda

m\_galaxyDao.VerifyAll();

// overime vsechny ocekavane akce

m\_mapper.VerifyAll();

// overime zda response obsahuje Id ktere se rovna zadanemu galaxyId

Assert.IsTrue(response.Galaxies.Any(d=>d.Id == galaxyId));

Assert.IsTrue(response.Galaxies.Count > 0);

}

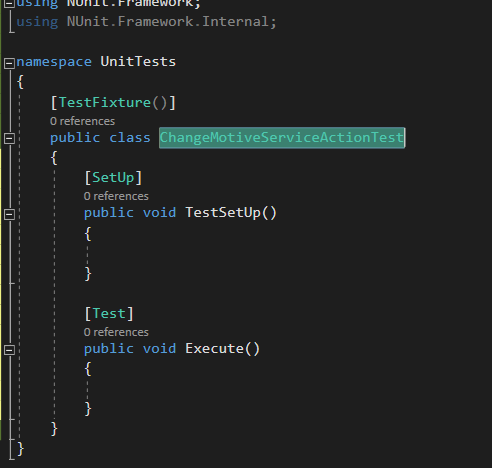
# Zmenovadlo 8/13/2018 :

Delam test zmenovaci servisni akce ChangeMotiveServiceActionTest

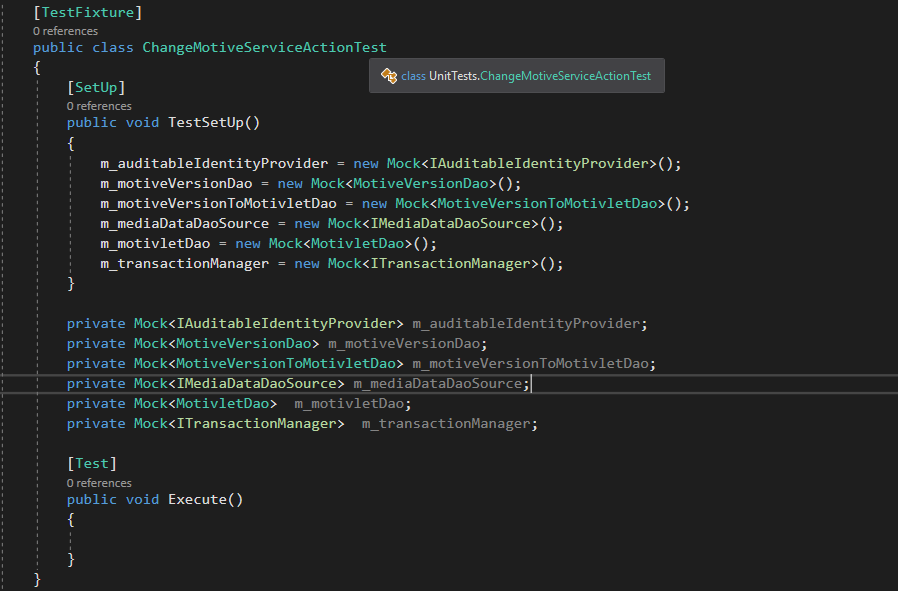
Pridal jsem assembly UnitTests

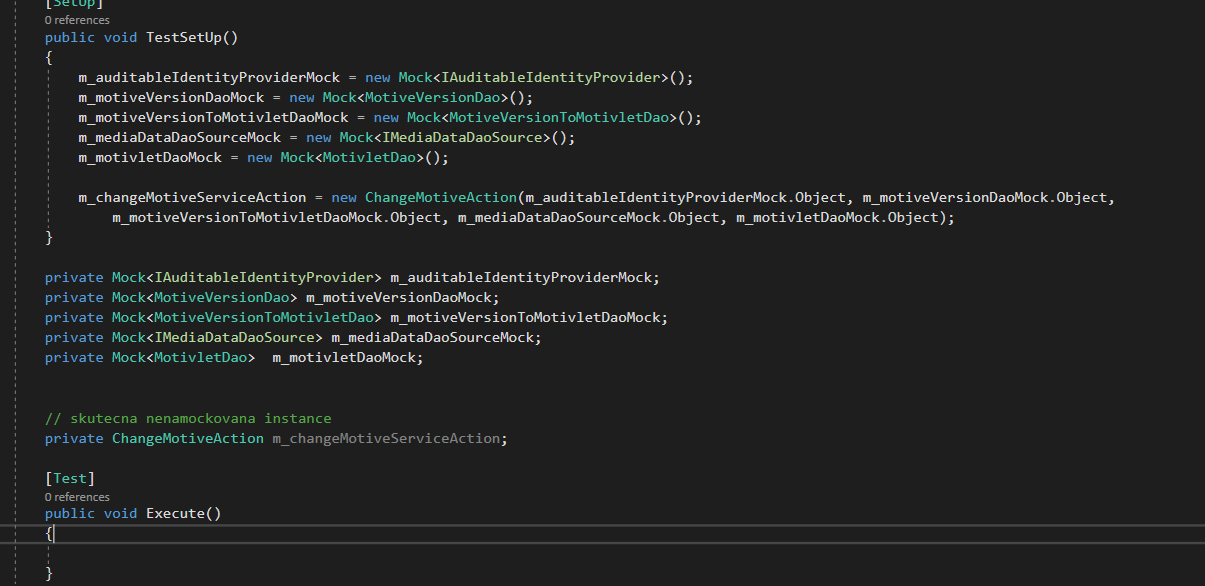
Nareferencoval ostatni projekty v solution a nainstaloval balicky Nunit , Moq, BLToolkit, Castle

Vytvoril jsem zaklad pro test:



Podivame se na zavislosti na skutecne servisni akci a namockujeme je v TestSetUp:



**vytvorime si skutecnou instanci servisni akce a jako parametry ji dame falesne objekty (.Object):**

Vytvorim testovaci data a muzu testovat jestli pri pouziti mojich dat se provedou pozadovane zmeny.

* Aby se spustila metoda v testu musi se taky namockovat

**TransactionManager**

Cau , prosim Te , jak si mam naMockovat TransactionManagera  ?  v servisni akci si ho taham z MediaDataDaoSource ale tohle nefunguje : m\_transactionManagerMock = new Mock<ITransactionManager>(m\_mediaDataDaoSourceMock.Object.TransactionManager);

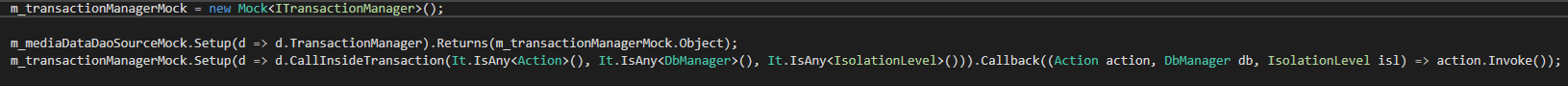
[‎2/‎9/‎2018 9:20 AM] Petr Mitrofan:

m\_mediaDataDaoSourceMock.Setup(x=> x.TransactionManager).Returns(m\_transactionManagerMock .Object)

pisu z hlavy

Z hlavy a spravne to je silenec. Neni potreba to davat do zadne promenne , proste jen napisu tento prikaz. TransactionManagerMock uz ale musi existovat:

****

**Dulezite je toto poradi: **

[‎2/‎9/‎2018 10:07 AM] Peter Hlavenka:

Testuju jednu servisni akci ve ktere se ale deje vice cinnosti . Kdyz chci otestovat vsechny moznosti, potrebuju dat testu vzdy jine testovaci data. Predpokladam ze potrebuju pro kazdou cinnost jednu testovaci tridu. Bude tam vzdy stejny zaklad , muze se pro UnitTest taky vytvorit bazovka ?

[‎2/‎9/‎2018 10:08 AM] Petr Mitrofan:

ano muze. a neni lepsi udelat vice testovacich metod?

v jedne testovaci tride muze byt libovolne mnozstvi metod oznacenych atributem test

# Prvni testovaci metoda

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using BLToolkit.Data;

using Mediaresearch.Framework.Communication.Common;

using Mediaresearch.Framework.DataAccess.BLToolkit.Dao;

using Mediaresearch.Framework.DataAccess.BLToolkit.Transactions;

using Mediaresearch.Framework.DataAccess.Core.Auditable;

using MIR.Common2.MediaData.DataContracts;

using MIR.Entities.MediaData.Dao.Media;

using MIR.Media.Changing2.Common.Requests;

using MIR.Media.Changing2.Core.Actions;

using Moq;

using NUnit.Framework;

namespace UnitTests

{

[TestFixture]

public class ChangeMotiveServiceActionTest

{

[SetUp]

public void TestSetUp()

{

m\_auditableIdentityProviderMock = new Mock<IAuditableIdentityProvider>();

m\_motiveVersionDaoMock = new Mock<IMotiveVersionDao>();

m\_motiveVersionToMotivletDaoMock = new Mock<IMotiveVersionToMotivletDao>();

m\_mediaDataDaoSourceMock = new Mock<IMediaDataDaoSource>();

m\_motivletDaoMock = new Mock<IMotivletDao>();

m\_transactionManagerMock = new Mock<ITransactionManager>();

m\_mediaDataDaoSourceMock.Setup(d => d.TransactionManager).Returns(m\_transactionManagerMock.Object);

m\_changeMotiveServiceAction = new ChangeMotiveAction(m\_auditableIdentityProviderMock.Object, m\_motiveVersionDaoMock.Object,

m\_motiveVersionToMotivletDaoMock.Object, m\_mediaDataDaoSourceMock.Object, m\_motivletDaoMock.Object);

}

private Mock<IAuditableIdentityProvider> m\_auditableIdentityProviderMock;

private Mock<IMotiveVersionDao> m\_motiveVersionDaoMock;

private Mock<IMotiveVersionToMotivletDao> m\_motiveVersionToMotivletDaoMock;

private Mock<IMediaDataDaoSource> m\_mediaDataDaoSourceMock;

private Mock<IMotivletDao> m\_motivletDaoMock;

private Mock<ITransactionManager> m\_transactionManagerMock;

// skutecna nenamockovana instance

private ChangeMotiveAction m\_changeMotiveServiceAction;

/// <summary>

/// Testuje jestli se zmeni properta na ComposedDataContractu ( Metoda ResolveContracts )

/// </summary>

[Test]

public void ChangeDataContractPropertiesTest()

{

var newValues = new List<ValueDataContract>();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = null;

// var selectedDate = new DateTime(2015, 1, 1);

newValues.Add(new MotiveNameDataContract {Name = "MotiveName"});

newValues.Add(new OwnerDataContract {Id = 1});

newValues.Add(new CompanyBrandDataContract {Id = 2});

newValues.Add(new ProductBrandDataContract {Id = 3});

newValues.Add(new ProductDetailDataContract {Id = 4});

newValues.Add(new CategoryDataContract {Id = 5});

newValues.Add(new RoleDataContract {Id = 6});

newValues.Add(new MarketDataContract {Id = 7});

newValues.Add(new GenderDataContract {Id = 8});

newValues.Add(new PlatformDataContract {Id = 9});

contracts.Add(new ComposedDataContract

{

MotivletId = 123,

MotiveVersionId = 5,

AllConditionsMet = true,

MotiveNameWillBeChanged = true,

OwnerWillBeChanged = true,

CompanyBrandWillBeChanged = true,

ProductBrandWillBeChanged = true,

ProductDetailWillBeChanged = true,

CategoryWillBeChanged = true,

RoleWillBeChanged = true,

MarketWillBeChanged = true,

GenderWillBeChanged = true,

PlatformWillBeChanged = true

});

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse) m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

var contract = contracts.First();

// jedna moznost je ze vrati stejne Id jako ma motivlet

m\_motivletDaoMock.Setup(d => d.CheckIfMotivletExists(contract)).Returns(123);

m\_motiveVersionDaoMock.VerifyAll();

Assert.IsTrue(contract.MotiveVersionName == "MotiveName");

Assert.IsTrue(contract.OwnerId == 1);

Assert.IsTrue(contract.CompanyBrandId == 2);

Assert.IsTrue(contract.ProductBrandId == 3);

Assert.IsTrue(contract.ProductDetailId == 4);

Assert.IsTrue(contract.CategoryId == 5);

Assert.IsTrue(contract.RoleId == 6);

Assert.IsTrue(contract.MarketId == 7);

Assert.IsTrue(contract.GenderId == 8);

Assert.IsTrue(contract.PlatformId == 9);

Assert.IsTrue(response.WasSuccessfull);

}

[Test]

public void Execute()

{

}

}

}

Contract.VerifyAll() z nejakeho duvodu pada, ale explicitni volani funguje :

m\_motivletDaoMock.Verify(d => d.CheckIfMotivletExists(contract));

m\_motiveVersionDaoMock.Verify(d => d.UpdateMotiveName(contract.MotiveVersionId, contract.MotiveVersionName));

# Cely test ze zmenovadla , nez jsme zmenovadlo predelali tak aby vsechny zmeny provadel na serveru:

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using BLToolkit.Data;

using Mediaresearch.Framework.Communication.Common;

using Mediaresearch.Framework.DataAccess.BLToolkit.Dao;

using Mediaresearch.Framework.DataAccess.BLToolkit.Transactions;

using Mediaresearch.Framework.DataAccess.Core.Auditable;

using MIR.Common2.MediaData.DataContracts;

using MIR.Entities.MediaData.Dao.Media;

using MIR.Media.Changing2.Common.Requests;

using MIR.Media.Changing2.Core.Actions;

using Moq;

using NUnit.Framework;

namespace UnitTests

{

[TestFixture]

public class ChangeMotiveServiceActionTest

{

[SetUp]

public void TestSetUp()

{

m\_auditableIdentityProviderMock = new Mock<IAuditableIdentityProvider>();

m\_motiveVersionDaoMock = new Mock<IMotiveVersionDao>();

m\_motiveVersionToMotivletDaoMock = new Mock<IMotiveVersionToMotivletDao>();

m\_mediaDataDaoSourceMock = new Mock<IMediaDataDaoSource>();

m\_motivletDaoMock = new Mock<IMotivletDao>();

m\_transactionManagerMock = new Mock<ITransactionManager>();

m\_mediaDataDaoSourceMock.Setup(d => d.TransactionManager).Returns(m\_transactionManagerMock.Object);

m\_changeMotiveServiceAction = new ChangeMotiveAction(m\_auditableIdentityProviderMock.Object, m\_motiveVersionDaoMock.Object,

m\_motiveVersionToMotivletDaoMock.Object, m\_mediaDataDaoSourceMock.Object, m\_motivletDaoMock.Object);

}

private Mock<IAuditableIdentityProvider> m\_auditableIdentityProviderMock;

private Mock<IMotiveVersionDao> m\_motiveVersionDaoMock;

private Mock<IMotiveVersionToMotivletDao> m\_motiveVersionToMotivletDaoMock;

private Mock<IMediaDataDaoSource> m\_mediaDataDaoSourceMock;

private Mock<IMotivletDao> m\_motivletDaoMock;

private Mock<ITransactionManager> m\_transactionManagerMock;

// skutecna nenamockovana instance

private ChangeMotiveAction m\_changeMotiveServiceAction;

private ComposedDataContract CreateComposedDataContract()

{

return new ComposedDataContract

{

MotivletId = 123,

MotiveVersionId = 5,

MotiveVersionName = "ContractsMotiveName",

AllConditionsMet = true,

MotiveNameWillBeChanged = true,

OwnerWillBeChanged = true,

CompanyBrandWillBeChanged = true,

ProductBrandWillBeChanged = true,

ProductDetailWillBeChanged = true,

CategoryWillBeChanged = true,

RoleWillBeChanged = true,

MarketWillBeChanged = true,

GenderWillBeChanged = true,

PlatformWillBeChanged = true,

IsPrimary = true

};

}

private List<ValueDataContract> CreateNewValues()

{

var newValues = new List<ValueDataContract>

{

new MotiveNameDataContract {Name = "NewMotiveNameValue"},

new OwnerDataContract {Id = 1},

new CompanyBrandDataContract {Id = 2},

new ProductBrandDataContract {Id = 3},

new ProductDetailDataContract {Id = 4},

new CategoryDataContract {Id = 5},

new RoleDataContract {Id = 6},

new MarketDataContract {Id = 7},

new GenderDataContract {Id = 8},

new PlatformDataContract {Id = 9}

};

return newValues;

}

private ComposedDataContract CreateContractWithProperties(int motiveVersionId, int owner, int company, int productBrand, int productDetail, int category, byte role, byte market, byte gender, byte platform)

{

return new ComposedDataContract

{

MotiveVersionId = motiveVersionId,

OwnerId = owner,

CompanyBrandId = company,

ProductBrandId = productBrand,

ProductDetailId = productDetail,

CategoryId = category,

RoleId = role,

MarketId = market,

GenderId = gender,

PlatformId = platform,

AllConditionsMet = true

};

}

/// <summary>

/// Testuje jestli se zmeni properta na ComposedDataContractu ( Metoda ResolveContracts )

/// </summary>

[Test]

public void ChangeDataContractPropertiesTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = null;

contracts.Add(CreateComposedDataContract());

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse)m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

var contract = contracts.First();

Assert.IsTrue(contract.MotiveVersionName == "NewMotiveNameValue");

Assert.IsTrue(contract.OwnerId == 1);

Assert.IsTrue(contract.CompanyBrandId == 2);

Assert.IsTrue(contract.ProductBrandId == 3);

Assert.IsTrue(contract.ProductDetailId == 4);

Assert.IsTrue(contract.CategoryId == 5);

Assert.IsTrue(contract.RoleId == 6);

Assert.IsTrue(contract.MarketId == 7);

Assert.IsTrue(contract.GenderId == 8);

Assert.IsTrue(contract.PlatformId == 9);

Assert.IsTrue(response.WasSuccessfull);

}

/// <summary>

/// Testuje metodu GetGroupedBySameProperties. Tato metoda ma seskupit datacontracty do skupin ve kterych jsou dc se stejnymi propertami.

/// </summary>

[Test]

public void GetGroupedBySamePropertiesTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = null;

contracts.Add(CreateContractWithProperties(4, 1, 1, 1, 1, 1, 1, 1, 1, 1));

contracts.Add(CreateContractWithProperties(5, 2, 2, 2, 2, 2, 2, 2, 2, 2));

contracts.Add(CreateContractWithProperties(6, 1, 1, 1, 1, 1, 1, 1, 1, 1));

contracts.Add(CreateContractWithProperties(7, 1, 1, 1, 1, 1, 1, 1, 1, 1));

contracts.Add(CreateContractWithProperties(8, 3, 3, 3, 3, 3, 3, 3, 3, 3));

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

m\_changeMotiveServiceAction.UpdateMotiveVersion(contracts, newValues);

Assert.AreEqual(3, m\_changeMotiveServiceAction.GetGroupedBySameProperties(contracts).Count());

Assert.AreEqual(3, m\_changeMotiveServiceAction.Dict.Count);

}

/// <summary>

/// Testuje, zda se z kolekce vyberou jen verze ktere probihaji ve zvolenem datumu, pokud je ActiveFrom stejne jako

/// SelectedDate, do kolekce nepatri => bude se updatovat

/// </summary>

[Test]

public void GetRunningVersionIdsTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

var selectedDate = new DateTime(2000, 1, 1);

contracts.Add(new ComposedDataContract { ActiveFrom = new DateTime(1500, 1, 1), ActiveTo = new DateTime(1800, 1, 1), AllConditionsMet = true });

contracts.Add(new ComposedDataContract { ActiveFrom = new DateTime(2001, 1, 1), ActiveTo = new DateTime(2100, 1, 1), AllConditionsMet = true });

contracts.Add(new ComposedDataContract { ActiveFrom = new DateTime(1500, 1, 1), ActiveTo = new DateTime(2001, 1, 1), AllConditionsMet = true });

contracts.Add(new ComposedDataContract { ActiveFrom = new DateTime(2000, 1, 1), ActiveTo = new DateTime(2001, 1, 1), AllConditionsMet = true });

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse)m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

Assert.AreEqual(1, m\_changeMotiveServiceAction.GetRunningVersionIds(selectedDate, contracts).Count);

Assert.IsTrue(response.WasSuccessfull);

}

/// <summary>

/// Testuje, zda daoMetody probihaji na MockObjektech kdyz selectedDate je null a CheckIfMotivletExists vraci Id = 0

/// (Nebyl nalezeny motivlet)

/// </summary>

[Test]

public void UpdateMotiveVersionWhenMotivletIdDoesNotExistTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = null;

contracts.Add(CreateComposedDataContract());

var contract = contracts.First();

m\_motivletDaoMock.Setup(d => d.CheckIfMotivletExists(contract)).Returns(0);

m\_motivletDaoMock.Setup(d => d.UpdateMotivlet(contract));

m\_motiveVersionDaoMock.Setup(d => d.UpdateMotiveName(contract.MotiveVersionId, contract.MotiveVersionName));

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse)m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

m\_motivletDaoMock.Verify(d => d.CheckIfMotivletExists(contract));

m\_motivletDaoMock.Verify(d => d.UpdateMotivlet(contract));

m\_motiveVersionDaoMock.Verify(d => d.UpdateMotiveName(contract.MotiveVersionId, contract.MotiveVersionName));

Assert.IsTrue(response.WasSuccessfull);

}

/// <summary>

/// Testuje, zda daoMetody probihaji na MockObjektech kdyz SelectedDate je null a CheckIfMotivletExists vraci Id = 456

/// (Metoda UpdateMotiveVersion)

/// Testuje, zda probehla zmena vazebni tabulky a zmena PrimaryId

/// </summary>

[Test]

public void UpdateMotiveVersionWhenMotivletIdExistsTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = null;

contracts.Add(CreateComposedDataContract());

var contract = contracts.First();

m\_motivletDaoMock.Setup(d => d.CheckIfMotivletExists(contract)).Returns(456);

m\_motiveVersionDaoMock.Setup(d => d.UpdateMotiveName(contract.MotiveVersionId, contract.MotiveVersionName));

m\_motiveVersionDaoMock.Setup(d => d.UpdatePrimaryId(contract.MotiveVersionId, contract.MotivletId));

m\_motiveVersionDaoMock.Setup(d => d.ChangeCurrentMotivletId(contract.MotiveVersionId, contract.MotivletId, 456));

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse)m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

m\_motivletDaoMock.Verify(d => d.CheckIfMotivletExists(contract));

m\_motiveVersionDaoMock.Verify(d => d.UpdateMotiveName(contract.MotiveVersionId, contract.MotiveVersionName));

m\_motiveVersionDaoMock.Verify(d => d.UpdatePrimaryId(contract.MotiveVersionId, 456));

m\_motiveVersionDaoMock.Verify(d => d.ChangeCurrentMotivletId(contract.MotiveVersionId, contract.MotivletId, 456));

Assert.IsTrue(response.WasSuccessfull);

}

[Test]

public void InsertTest()

{

var newValues = CreateNewValues();

var contracts = new List<ComposedDataContract>();

DateTime? selectedDate = new DateTime(2018, 2, 12);

contracts.Add(CreateComposedDataContract());

m\_transactionManagerMock.Setup(d => d.CallInsideTransaction(It.IsAny<Action>(), It.IsAny<DbManager>(),

It.IsAny<IsolationLevel>())).Callback((Action action, DbManager db, IsolationLevel isl) => action.Invoke());

var response = (DoneResponse)m\_changeMotiveServiceAction.Execute(new ChangeMotiveRequest(newValues, contracts, selectedDate));

// Assert.t

}

}

}

# Testy Zmenovadlo : zacatek s Filipem

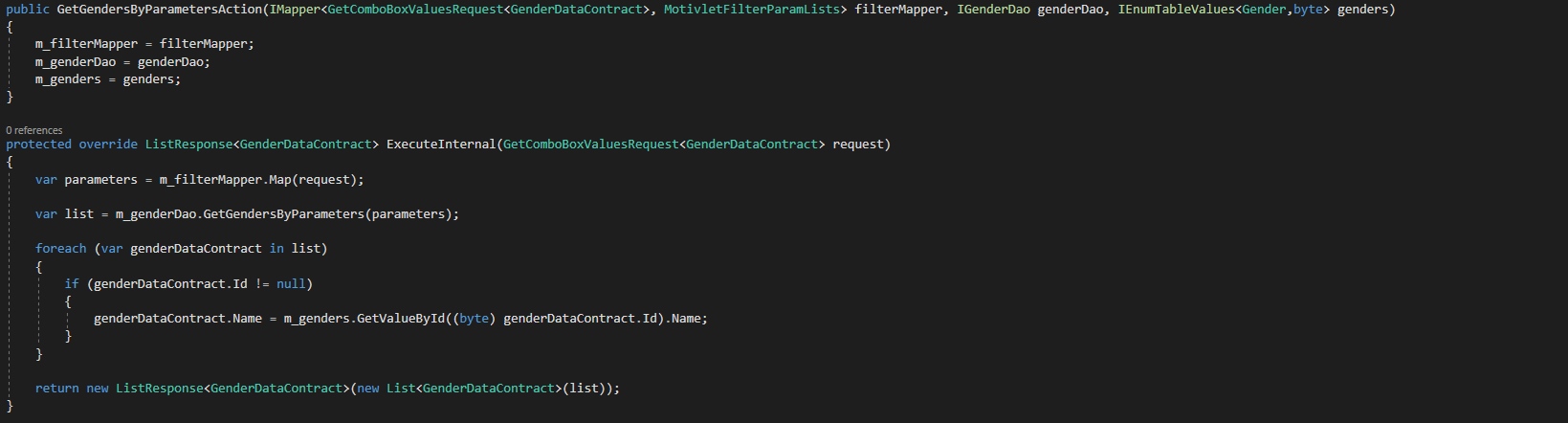


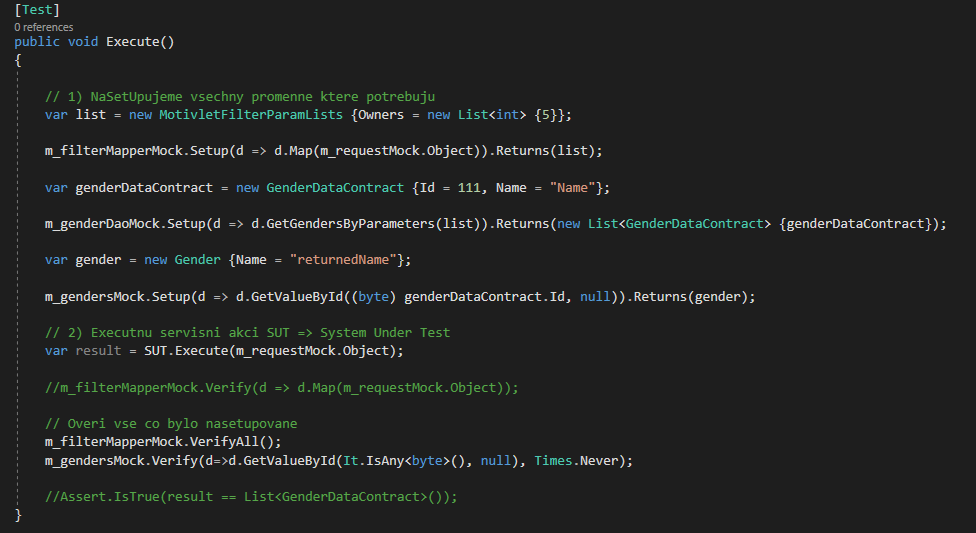
Vytvorim promenne pro vsechny Mock objekty ktere budu potrebovat

Vytvorim jejich instance

SUT => System Under Test je jen jeden, je to skutecna testovana trida.

**Skutecna testovana trida:**





Vytvorime promennou list ktera se bude vrace v setupech

Pri mapovani vrat list

Budeme potrebovat dalsi promennou ktera se bude vracet z metody m\_genderDao.GetGenders

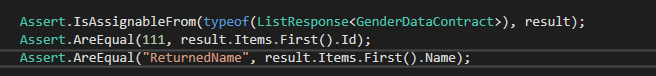
Z metody GetGendersByParameters vrat List<> s nasi promennou genderDataContract

Tato metoda vraci objekt typu Gender. Tento objekt ma na sobe propertu Name takze skutecne volani

ve skutecne akci vi co si ma na objektu vzit. Null v parametru je nutny I kdyz ve skutecnem volani neni.

Zavolame skutecnou akci (kod v debugu prochazi skutecnou tridou)

VerifyAll proveruje vse co bylo nasetupovano.

 Vraci se ze servisni akce ListResponse<GenderDataContract> ?

Vraci spravne data ?

**Zajimavost:**

m\_gendersMock.Verify(d=>d.GetValueById(It.IsAny<byte>(), null), Times.Once);

**VerifyAll**  overuje jestli byly zavolane vsechny nasetupovane metody. Volani metod ktere nebyly nasetupovane (ty ktere nic nevraci) se musi overit explicitne.

Overuje se tim jestli byly metody zavolany. Metoda muze byt za nejakym If a nemusi se zavolat.

m\_filterMapperMock.VerifyAll();

m\_marketsMock.VerifyAll();

m\_marketDaoMock.VerifyAll();

Overuje se dokonce I to ze metody neprobehnou kdyz nemaji probehnout (jsou napriklad za podminkou if ktera neni splnena):

m\_motiveVersionDaoMock.Verify(d => d.UpdateTempVersionsMotiveName(It.IsAny<string>(), It.IsAny<IAuditableIdentityProvider>()), Times.Never);

**Privatni a protected** metody se nedaji testovat pomoci instance tridy.

Priklad: Neni mozne otestovat metody na bazovce protoze ta je abstract. Jeji metody jsou teda protected. Testovat se da pouze implementace (potomek) bazovky. Ve zmenovadle to byla napr trida

ChangeAllService : ChangingServiceBase, **IchangeAllService**

Pomocne metody na bazovce jsou sice netestovatelne, ale ovlivnuji result testovane servisni akce.

[‎3/‎19/‎2018 4:43 PM] Peter Hlavenka:

Nektere pomocne metody nepujdou otestovat. Jsou to ty, ktere vezmou na vstupu nejaky objekt, a vrati jiny objekt treba do promenne. Ta se pak treba jen insertne do Temp tabulky

muzu vlastne testovat jen result metody  a volani daoMetod

[‎3/‎19/‎2018 4:44 PM] Peter Hlavenka:

result cele metody, ne te pomocne

[‎3/‎19/‎2018 4:45 PM] Filip Čálek:

ano presne tak

ale vis, ze ta "pomocna" metoda ti nejakym zpusobem na zaklade vstupu modifikuje vystup testovane metody. a to muzes testovat

[‎3/‎19/‎2018 4:47 PM] Peter Hlavenka:

konkretne tady ani ne protoze to jen insertnu do tempTabule a pak si sam urcuju co se mi vrati ze serveru.  Jen jsem se chtel ujistit ze to nejde testovat

[‎3/‎19/‎2018 4:50 PM] Filip Čálek:

jo tak- nj tak to nejde

Implementacim bazovky jsem vytvoril rozhrani  napr: **IchangeAllService** diky kteremu snadno namockuju cele servisni akce, aniz bych musel resit a zbytecne Mockovat zavislosti ktere maji v konstruktoru:

private Mock<IChangeAllService> m\_changeAllServiceMock;

# Jak setnout propertu na mockovanem objektu – tady auditableIdentityProvider.UserIdentity.UserId

Ve zmenovadle jsem zmenil zmenovaci tridy a prestali makat unit testy. Properta UserIdentity byla null.

**Pridame propertu**

private Mock<IAuditableIdentity> m\_auditableIdentityMock;

**Nainicializujeme**

m\_auditableIdentityMock = new Mock<IAuditableIdentity>();

**Rekneme co ma vracet**

m\_auditableIdentityProviderMock.Setup(d => d.UserIdentity).Returns(m\_auditableIdentityMock.Object);

m\_auditableIdentityProviderMock.SetupGet(m => m.UserIdentity.UserId).Returns(1);

**A setneme metodu ve ktere to padalo**

m\_motivletDaoMock.Setup(d => d.GetJoinedMotivlets(It.IsAny<MotivletFilterParamLists>())).Returns(new List<Motivlet>());

.